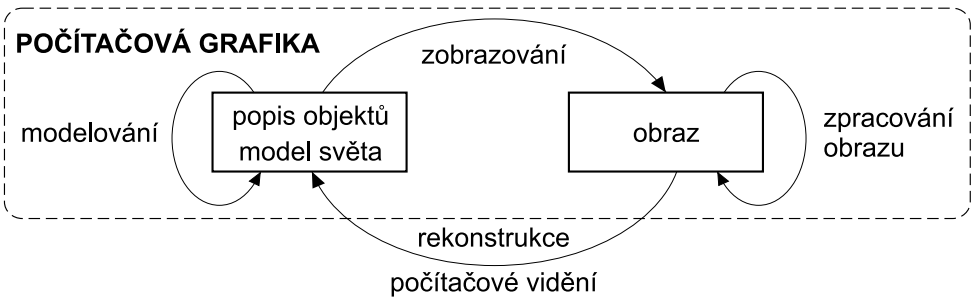
Obsah

Jak číst tuto knihu

Kniha *Moderní počítačová grafika* obsahuje průřez metodami, které se v současné době používají v oblasti rovinné a prostorové počítačové grafiky. Ta se od svých počátků v 70. letech rozvinula do samostatné vědní disciplíny a současně se těší stále velkému zájmu jak uživatelů, tak programátorů. Vývoj výpočetní techniky přesunul těžiště používání grafiky z ryze technických oblastí na obrazovky běžných uživatelů.

Kniha je psána tak, aby s ní mohli efektivně pracovat středoškolsky i vysokoškolsky vzdělaní čtenáři. Každá kapitola obsahuje úvod, ve kterém je řečeno, na jakých místech se dané téma v počítačové grafice objevuje a k čemu slouží. Poté následuje základní část, ve které jsou uvedeny informace a postupy postačující k principiálnímu pochopení problematiky. Tato část dává odpovědi na otázky typu: „Jak se to dělá?“, případně „Jak se to programuje?“ Většina kapitol obsahuje také část určenou pokročilejším čtenářům. Hlubší výklad, jiné způsoby řešení a speciální postupy popsané v této části odpovídají na otázky typu: „Proč se to dělá právě takto?“ či „Jak to udělat lépe?“



Na schématu, který zobrazuje způsoby zpracování informací grafického charakteru, je vidět, kterým oblastem je tato kniha věnována – totiž tvorbě obrazu na základě jeho geometrického popisu. K tomu je třeba nejprve popsat objekty, které mají být zobrazovány, ať již mají dvourozměrný nebo trojrozměrný charakter. Objekty jsou podrobovány různým transformacím a operacím a posléze převedeny do podoby (rastrového) obrazu. Ten může být dále upravován. Ze schématu je zřejmé, že počítačová grafika zahrnuje prostředky pro modelování, zobrazování a úpravy obrazu. O všech těchto oblastech kniha pojednává.

První část knihy je věnována *rovinné grafice* (2D grafice). Zabývá se popisem objektů definovaných v rovině, jejich rasterizací a úpravami vzniklého obrazu. Další tři části jsou zaměřeny na prostorovou grafiku (3D grafiku). Nejprve jsou uvedeny metody *popisu prostorových objektů*, poté způsoby jejich *zobrazování* a nakonec techniky používané při *animacích a aplikacích v reálném čase*. Poslední část knihy potěší programátory – obsahuje výběr *matematických vzorců* a praktických výpočetních postupů, které se nejčastěji používají v počítačové grafice.

Věříme, že čtenáři, kteří teprve začínají projevovat hlubší zájem o počítačovou grafiku, v této knize naleznou dostatek dalších podnětů. Stejně tak věříme, že zkušení uživatelé a programátoři zde získají hodnotné informace, které rozšíří objem jejich znalostí.

autoři
Jiří Žára, Bedřich Beneš, Jiří Sochor, Petr Felkel

Poznámky ke druhému vydání

Toto vydání je podstatně přepracovanou verzí stejnojmenné knihy z roku 1998. Autorský kolektiv obohatil Jiří Sochor, zkušený pedagog a vědec, který se počítačovou grafikou zabývá řadu let. Na základě podnětů od čtenářů jsme opravili chyby a překlepy z prvního vydání, vynechali texty, které se věnovaly málo používaným či zastaralým metodám, a především jsme přidali další informace zachycující novinky, které se v oblasti počítačové, zejména prostorové grafiky objevily od prvního vydání knihy. Z mnoha nových témat jmenujme například zpracování obrazů s vysokým dynamickým rozsahem, nefotorealistické zobrazování, stochastické metody v globálním osvětlování či techniky pro zobrazování rozsáhlých scén v reálném čase.

Na tomto místě bychom chtěli poděkovat našim kolegům, doktorandům a studentům, kteří byli prvními čtenáři rukopisu, pomáhali nám odstraňovat chyby a odborně se podíleli na přípravě obrázků a specializovaných částí textu. Byli to (v abecedním pořadí):

- David Ambrož – kapitola 12,
- Petr Beneš – část 5.13,
- Jiří Bíca Bittner – část 19.1,
- Martin Čadík – část 4.2,
- Robert Goliáš – část 18.2,
- Ladislav Kavan – části 18.3 a 21.4,
- Jaroslav Křivánek – část 11.5,
- Radamez Cruz Moreno – část 15.6.1,
- Lenka Proňková – část 15.10,
- Adam J. Sporka – část 20.4,
- Vladimír Štěpán – část 18.4,
- Nestor Gomez Villanueva – část 8.1.5
- Václav Těšínský – 8.2.2 a
- Roman Ženka – kapitola 17.

Děkujeme též našim pracovištím – Katedře počítačů FEL ČVUT v Praze, Departamento de Computación Tecnológico de Monterrey Campus Ciudad de México a Fakultě informatiky MU v Brně – za technickou podporu a podmínky, díky nimž jsme mohli toto vydání knihy připravit.

Část A

ROVINNÁ GRAFIKA

Tento oddíl knihy pojednává o široké oblasti, která se nazývá dvojrozměrná (2D) počítačová grafika. Setkáváme se s ní například při tvorbě publikací pomocí počítače (DTP – *Desk Top Publishing*), při návrhu novin, časopisů, billboardů, atp. Dnes je asi obtížné najít dokument, který neprošel v některé fázi zpracování počítačem. Prostředky dvojrozměrné grafiky se používají při vytváření webových stránek a při tvorbě prezentačních pásem. Stejně tak jsou využívány na pracovištích zabývajících se digitálním videem a pořizováním filmových triků. S dvojrozměrnou počítačovou grafikou se setkáváme ve stále intenzivnější formě při zpracovávání digitálních fotografií. Říká se, že digitální fotografie vrátila fotografům potěšení z domácího vyvolávání. Můžeme odstraňovat kazy na fotografiích ještě před jejich vytištěním, modifikovat je, opravovat jasové podmínky atp. Dvojrozměrná grafika je také důležitá pro grafická uživatelská rozhraní (GUI – *Graphical User Interface*).

Počítačová grafika se ocitá na rozhraní přinejmenším dvou vědních disciplín. Přibližuje se k počítačovému vidění (*computer vision*), které zpracovává obraz především za účelem jeho interpretace. Druhou oblastí je teorie signálů, ze které využijeme oblast zabývající se popisem přechodu od spojitého obrazu k diskrétnímu.

V úvodu této části knihy nejprve vysvětlíme, co je to barva, jak ji člověk vnímá a jak je v počítači reprezentována. Popíšeme diskrétní obraz a jeho získání z obrazu spojitého. Počítačový obraz je nejčastěji získán vzorkováním spojité scény a kvůli diskrétní povaze počítačového obrazu budeme neustále bojovat s aliasingem. V další části popíšeme uložení a reprezentaci diskrétního obrazu, jeho kompresi a záznam v některých obrazových formátech. Následuje kapitola o tom, jak z daného obrazu můžeme vytvořit obraz jiný. Nejprve se zaměříme na barevné změny, a to hned z několika pohledů. Povšimneme si úlohy, kdy máme obraz reprezentovaný v mnoha barvách a potřebujeme jejich množství omezit. Podíváme se na to, jak z jasově nevyváženého obrazu získat maximum užitečné informace, vysvětlíme, co jsou to obrazy s velkým dynamickým rozsahem, a konečně se též zastavíme u odstraňování šumu. Uvedeme i možnosti napodobování některých uměleckých směrů. Ve druhé části kapitoly pojednávající o změnách obrazu se budeme též zabývat jeho geometrickými transformacemi. Nejprve si všimneme změny rozlišení obrazu, po níž následuje odstavec o populárním warpingu a morfingu. V této části knihy uvedeme také algoritmy rasterizace a ořezávání základních grafických objektů, jako jsou úsečka, kružnice, elipsa a vyplněný mnohoúhelník.

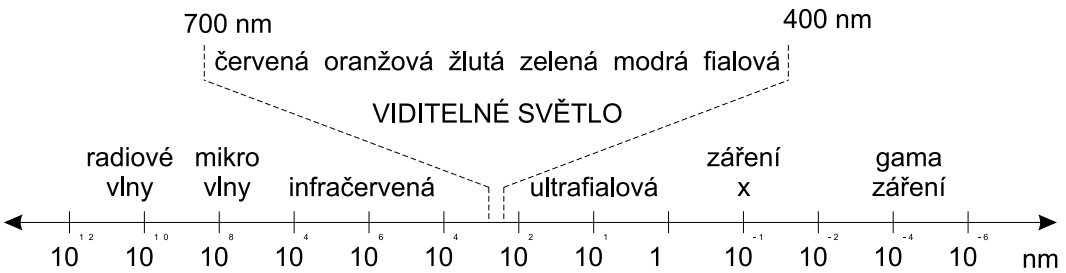
Kapitola 1

Světlo a barvy v počítačové grafice

1.1 Vlastnosti lidského systému vidění

1.1.1 Elektromagnetické spektrum

Světlo, které vnímáme, představuje viditelnou část elektromagnetického (EM) spektra (obr. 1.1), které zahrnuje všechny známé druhy záření, jako např. paprsky x nebo mikrovlny. EM záření vzniká oscilací elektricky nabitých materiálů a má charakter vlnění. Šíří se rychlostí přibližně 300 000 km/s po přímých drahách a umožňuje v pozemském měřítku předat informaci o událostech téměř okamžitě po jejich výskytu. EM záření může interagovat s látkami různým způsobem, v závislosti na vlnové délce. Obrazy pořízené při různých vlnových délkách mohou mít odlišné vlastnosti a poskytovat o jevech a objektech rozdílné informace.



Obrázek 1.1: Spektrum elektromagnetického záření

Viditelná část spektra se nalézá v oblasti vlnových délek přibližně 380 až 720 nm. Uvnitř této oblasti vnímáme záření s určitou vlnovou délkou jako *barvu*; světlo s vlnovou délkou 550 nm se jeví jako zelené, světlo o délce 720 nm je červené. EM záření s kratšími vlnovými délkami nese

více energie. V oblasti paprsků x (vlnová délka okolo 0.1 nm) nese záření dostatečnou energii, která umožňuje prostoupit větším objemem hmoty. Obrazy vytvořené paprsky x proto umožňují odhalit vnitřní strukturu objektů, které jsou neprůhledné v oblasti viditelného světla (např. lidské tělo). Gama paprsky jsou běžným produktem rozpadu radioaktivních látek. Mají vysokou schopnost procházet materiály a stejně jako paprsky x mají význam v lékařství. Obrazy získané pomocí gama záření se používají pro odhalení *funkcí*. Látka označkovaná pomocí radioaktivní stopové příměsi prostupuje různými tkáněmi a orgány podle jejich aktivity. Kamera, která zaznamenává fotony pocházející z gama záření, vytváří obrazy, na nichž jsou vidět místa soustředění označkované látky.

Infračervené záření, jehož vlnové délky jsou delší než světlo, má také význam při zobrazování. Infračervené záření vydávají zahřáté objekty a lze je tedy použít pro lokalizaci a vizualizaci těchto objektů (lidí, aut, aj.) ve tmě.

Světelný zdroj, jakým je slunce nebo obyčejná žárovka, vysílá paprsky všech frekvencí v daném pásmu, které se tak skládají ve výsledné bílé světlo. Takové světlo se nazývá *achromatické*.

Dopadne-li bílé světlo na objekt, jsou některé frekvence povrchem objektu odraženy a některé jsou pohlceny. Kombinace frekvencí přítomných v odraženém světle vytváří to, co vnímáme jako barvu objektu. Převládají-li v odraženém světle nízké frekvence, je objekt vnímán jako červený. Tehdy také hovoříme o tom, že vnímané světlo má *dominantní frekvenci* (nebo dominantní vlnovou délku) na červeném konci spektra. Dominantní frekvence je nazývána barvou, zabarvením světla, případně tónem.

Pro popis charakteristik světla jsou užitečné kromě frekvence i další vlastnosti. Pozorujeme-li zdroj světla, naše oko reaguje nejen na barvu, ale i na další podněty:

Jas (*intensity*), případně *svítivost* (*luminosity*) odpovídá intenzitě světla. Čím vyšší je intenzita světla, tím se jeví zdroj jasnější.

Sytost (*saturation*) udává čistotu barvy světla. Je tím vyšší, čím užší je frekvenční spektrum světla.

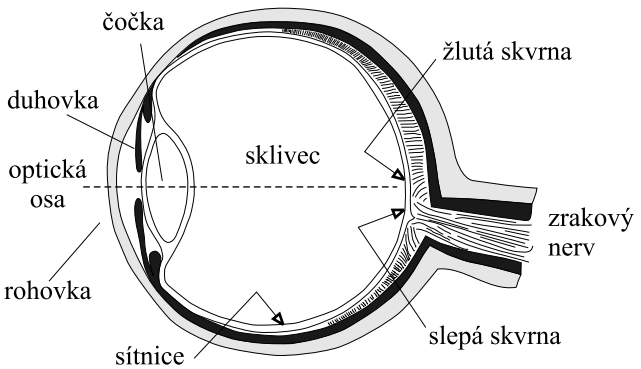
Světlost (*brightness*) určuje velikost achromatické složky ve světle s určitou dominantní frekvencí.

Uvedené charakteristiky jsou používány k popisu různých vlastností, které vnímáme u zdroje světla. Můžeme se ještě setkat s pojmem *barevnost* (*chromaticity*), který označuje dvojici vlastností charakterizující světlo: sytost a dominantní frekvenci.

1.1.2 Lidské oko

Lidské oko (obr. 1.2) se blíží tvarem kouli o průměru přibližně 20 mm. V přední části oka je je *rohovka* (cornea), tuhá průhledná tkáň, která chrání oko a zajišťuje výchozí zaostřování

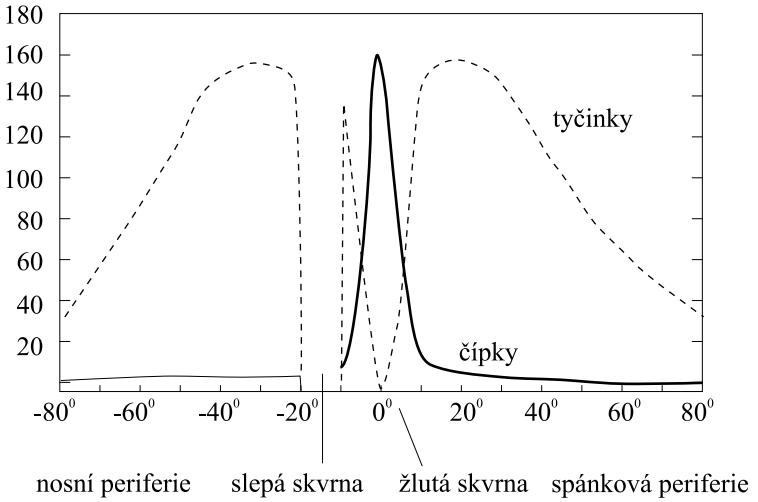
a koncentraci příchozího světla. Za rohovkou je *duhovka* (iris), která se stejně jako brána může otevírat nebo uzavírat a ovládá tak množství světla, které vstupuje do vnitřní oblasti oka. Centrální otvor v duhovce, tzv. *zornice* (panenka) má průměr přibližně 2 – 8 mm. Závěrečné zaostření vstupujících světelných paprsků zajišťuje *čočka*, průhledná struktura složená z vrstev vláknitých buněk. Čočky absorbují okolo 8 % světla ve viditelné části spektra; více je pohlcováno světlo s kratšími vlnovými délkami. Nadměrné množství záření v infračervené nebo ultrafialové oblasti spektra může rozložit bílkoviny a poškodit tak oční čočku. Čočka v kroužkovitém vláknitém závěsu je připojena ke svalům. Čočka je pružná, svaly mohou měnit její geometrii a upravovat ohniskovou vzdálenost. Při zaostřování na vzdálené předměty se čočka zplošťuje, naopak u blízkých předmětů se ztlušťuje.



Obrázek 1.2: Lidské oko

Při správném nastavení čočky se zaostřený obraz vnějšího světa promítá na *sítnici*. Je to tenká fotocitlivá vrstva pokrývající přibližně dvě třetiny vnitřního povrchu oka. Sítnice obsahuje dva druhy fotocitlivých receptorů: *tyčinky* a *čípky* (*rods, cones*). V lidském oku je přibližně 120×10^6 tyčinek a 8×10^6 čípků, které jsou rozmístěny tak, jak je ukázáno na obrázku 1.3.

Tyčinky zprostředkovávají primárně noční vidění. Jsou přibližně $10\times$ citlivější než čípky a reagují i na velmi malé změny nízké úrovně osvětlení. Čípky jsou základem barevného vidění a reagují na vyšší hodnoty intenzity osvětlení. Každý čípek obsahuje jeden ze tří druhů fotopigmentů, které se liší podle relativní citlivosti na vlnové délky světla. Jeden fotopigment má nejvyšší citlivost v oblasti krátkých vlnových délek viditelné části spektra okolo 445 nm a je necitlivý na vlnové délky delší než 520 nm. Obvykle je označován jako *modrý* fotopigment. Další dva typy fotopigmentů jsou nejcitlivější na 535 nm a 575 nm, ale reagují prakticky na všechny vlnové délky ve viditelném spektru. Jsou označovány jako *zelené* a *červené* fotopigmenty. Pojmenování není odvozeno z jejich barvy, ale z barevného vjemu, který je spojen s jejich



Obrázek 1.3: Rozložení fotoreceptorů na sítnici (0° odpovídá optické ose oka) dle [Murc87]

maximální citlivostí. Navíc, název *červeného* fotopigmentu není zcela přesný, neboť jeho citlivost je největší v oblasti žlutého světla. Barevné pigmenty nejsou zastoupeny na sítnici ve stejném poměru. Přibližně 64 % čípků obsahuje červené a 32 % zelené pigmenty. Modré pigmenty jsou obsaženy asi ve 2 % čípků.

Obrázek 1.4: Nerovnoměrné rozložení čípků a tyčinek podle vzdálenosti od žluté skvrny

Tyčinky a čípky nejsou rozmístěny rovnoměrně po celé sítnici (obr. 1.4). Čípky jsou soustředěny na malé ploše, přibližně o průměru 1.5 mm, umístěné v průsečíku optické osy oka se sítnicí. Tato oblast, nazývaná *žlutá skvrna* (fovea) obsahuje okolo 300 000 čípků. Vynikající zrak dravých ptáků je částečně vysvětlován také tím, že mají v oblasti fovey čtyřikrát více fotoreceptorů než lidé.

Rozložení fotoreceptorů je v oblasti žluté skvrny osově symetrické, s výjimkou oblasti asi

20° od optické osy. Tato oblast neobsahuje žádné fotoreceptory a odpovídá ve smyslu vnímání *slepé skvrně*. Je to místo, ve kterém se všechny spoje od fotoreceptorů sbíhají do zrakového nervu, který přenáší obrazovou informaci do mozku. Ačkoliv je na sítnici téměř 130 miliónů fotoreceptorů, zrakový nerv obsahuje pouze 1 milión vláken. Je zřejmé, že významná část integrace a zpracování obrazové informace se odehrává přímo na sítnici.

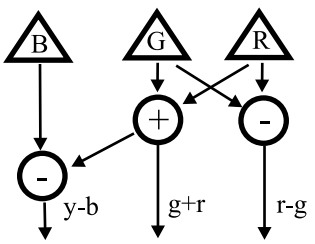
Existence slepé skvrny je důsledkem toho, že sítnice je utvořena „naruby“, tj. že fotoreceptory nejsou ve vrstvě bezprostředně sousedící se sklivcem a navíc jsou orientovány směrem od světla. Toto zvláštní uspořádání je společné všem obratlovcům, u nichž se sítnice vyvinula jako výrůstek mozkové tkáně. Neurony, které snímají informaci z fotoreceptorů, leží na sítnici před těmito receptory a tvoří překážky pro přicházející světlo. Výjimkou je žlutá skvrna, kde je sítnice tenčí, neboť většina překrývající vrstvy zde vymizela. Nepatrná tloušťka přední vrstvy v tomto místě je rozhodující pro ostré vidění.

Každý čípek je připojen k vlastnímu nervovému zakončení, čípky určují schopnost rozlišit detaily. Naproti tomu vždy několik tyčinek je připojeno k jednomu nervovému zakončení. Tyčinky jsou umístěny v oblasti vně žluté skvrny, kde výrazně převládají nad čípky. Důsledkem je, že tyčinky poskytují obraz téměř v celém zorném poli, ale s nízkým rozlišením.

1.1.3 Citlivost na barvy a jas

Zrakový nerv je tvořen svazkem nervových vláken, která předávají signály od fotoreceptorů do mozku. Na této nervové cestě dochází v místě, které se nazývá *lateral geniculate*, k rekombinaci barevné informace. Schéma rekombinace je naznačeno na obrázku 1.5.

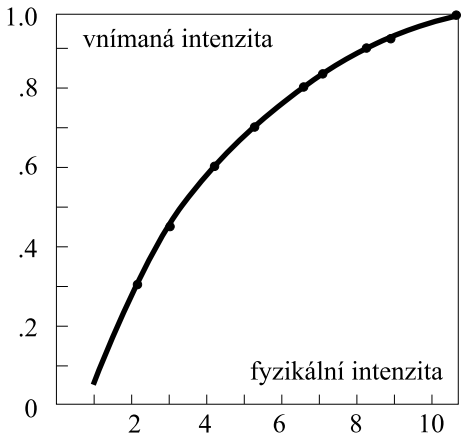
Ze tří původních informačních kanálů r, g, b vznikají po rekombinaci tři odlišné kanály: jeden kanál nese informaci o poměru *červená-zelená* (r-g), druhý o poměru *žlutá-modrá* (y-b). Tyto kanály poskytují informace o barvě, zatímco třetí kanál *zelená+červená* (g+r) indikuje jas. Jelikož mozek vyhodnocuje barvu až na základě rekombinovaných signálů, plynou z této transformace zajímavé důsledky. Faktem je, že lidé nejsou schopni vnímat některé barevné kombinace současně. Čtenář si může položit otázku, zda někdy viděl *nažloutlou modř* nebo *nazelenalou červeň*. Protože se oko zaostřuje na hrany podle výrazných změn jasu, plyne z toho, že hrany a tvary nejsou téměř rozlišitelné v odstínech modré barvy.



Obrázek 1.5: Rekombinace barevných stimulů

Lidský vizuální systém je schopen adaptace na obrovský rozsah úrovní světla – podstatně více než kterýkoliv současný elektronický systém. Horní a dolní meze intenzity (mez oslnění a práh vnímání za šera) se liší násobkem 10^{10}. Náš vizuální systém samozřejmě nepracuje

s intenzitami v tomto rozsahu současně. Používá změnu celkové citlivosti oka, nazývané *adaptace na jas*.



Obrázek 1.6: Nelineární vnímání intenzity fyzikálního světla

V porovnání s celkovým rozsahem je rozsah intenzit, které mohou být po adaptaci rozlišeny současně, malý. Běžný pozorovatel dokáže rozlišit několik desítek úrovní intenzity v jednom obrazovém bodě. Pokud ale oko prohlíží postupně další oblasti obrazu, vizuální systém vyhodnotí jinou průměrnou hodnotu intenzit a je schopen detekovat odlišný rozsah změn v intenzitě. Mezi vnímaným jasem a intenzitou světla je složitý vztah. Vnímaný jas je logaritmickou funkcí intenzity přicházejícího světla. Přírůstek intenzity fyzikálního světla vnímáme jako velkou změnu při nízké úrovni osvětlení a stejný přírůstek nejsme schopni rozlišit v blízkosti prahu oslnění (obr. 1.6). Metody, které se zabývají počítačovým zpracováním obrazů s vysokým rozsahem intenzit, jsou popsány v části 4.2.

1.2 Barevné prostory

1.2.1 Prostor RGB

Různé barvy, které se používají při vytváření obrazu, jsou tvořeny kombinací několika základních barev z barevného spektra. Na barevné obrazovce například vidíme barvu jako výsledek složení tří složek – červené (R, *red*), zelené (G, *green*) a modré (B, *blue*). Barvy lze vyjádřit trojicí (barevným vektorem), jejíž složky nabývají hodnot z intervalu $\langle 0, 1 \rangle$. Bývají uváděny i v celočíselném rozsahu $0 - 255$, což odpovídá kódování každé ze složek RGB v jednom bytu. Hodnota 0 znamená, že složka není zastoupena, maximální hodnota indikuje, že složka nabývá své největší intenzity. Vyjádření barevných složek pomocí 3 bytů je v současnosti nejběžnější. Používají se ale i jiná kódování (s nižším nebo vyšším počtem bitů), např. 12 nebo 16 bitů na barevný kanál.

Počet barevných odstínů, který lze reprezentovat trojicí bytů je $256^3 = 16\,777\,216$. Ne všechna výstupní obrazová zařízení jsou schopna takové množství barev současně zobrazit. Proto bývá počet barev před vykreslením uměle snižován, ovšem tak, aby lidské oko zaznamenalo co nejmenší ztrátu kvality obrazu. Metody pro snižování počtu barev popíšeme později,

v kapitole 4.1.1. Podobně se také k barvě vrátíme v kap. 10, věnované zpracování světla odraženého barevným povrchem prostorového objektu.

Složíme-li červenou, zelenou a modrou barvu zobrazovanou displejem v plné intenzitě, získáme barvu bílou. Podobně získáme stupně šedi skládáním všech tří barev se shodnou, postupně se snižující intenzitou. Kdybychom však chtěli z nějakých důvodů převést různobarevný obraz na šedotónový, nemůžeme jeho barvy nahradit odstíny šedi získanými prostým průměrem ze tří základních barev. Lidské oko vnímá různým způsobem intenzitu jednotlivých barevných složek (nejcitlivější je na zelenožlutou), takže se používá pro výpočet *jasu* empirický vztah

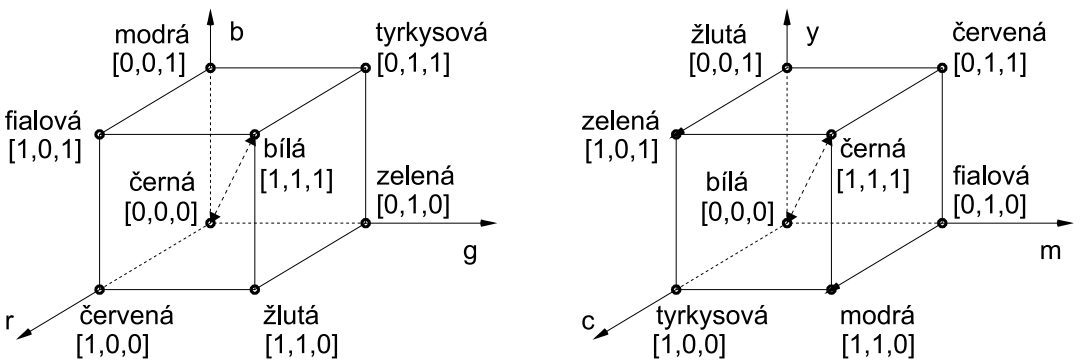
$$I = 0.299\ R + 0.587\ G + 0.114\ B. \tag{1.1}$$

V následujících kapitolách bude pojem *barva* používán nejčastěji v abstraktním smyslu – jako vlastnost nějakého objektu (úsečky, kvádru, apod.). Z programátorského hlediska je zřejmé, že konkrétní reprezentace barvy může být značně různorodá. Od jednobitové informace rozlišující pouze mezi stavy černá a bílá, přes osmibitové číslo označující stupeň šedi až po různé zápisy barevných složek. Poznamenejme, že pro reprezentaci barevných složek R,G,B ve dvou bytech ($5-6-5$ bitů) se vžilo označení *high color*, pro zápis ve třech bytech označení *true color*. Někdy jsou barvy vyjádřeny nepřímo – pomocí čísla, odkazujícího do tabulky, zvané *paleta*.

Popis barvy třemi složkami R, G a B není jedinou možností. Uvedená volba základních barev je dána technickými vlastnostmi monitorů, resp. použitými luminiscenčními prvky, které jsou zdrojem vyzařovaného barevného světla. Říká se, že barva je vyjádřena v *barevném prostoru RGB*. Jeho základní vlastností je součtové, *aditivní skládání* barev – čím více barev složíme (sečteme), tím světlejší je výsledek. Svítivost barevných luminoforů v monitorech roste nelineárně, a proto se při zobrazení barvy v prostoru RGB používá tzv. gama korekce, které se budeme podrobněji věnovat v kap. 4.5.1.

Barevný rozsah můžeme v prostoru RGB zobrazit prostorově jako jednotkovou krychli umístěnou v osách označených postupně r, g a b (obr. 1.7). Počátek souřadnic odpovídá černé barvě, zatímco vrchol o souřadnicích [1, 1, 1] odpovídá bílé. Vrcholy krychle, které leží na osách, představují základní barvy a zbývající vrcholy reprezentují doplňkové barvy ke každé ze základních barev.

Každému barevnému vektoru v prostoru RGB odpovídá v této reprezentaci jeden bod krychle. Fialová barva, která je získána součtem červené a modré, je znázorněna bodem o souřadnicích $[1, 0, 1]$. Bílá, představovaná vrcholem $[1, 1, 1]$, je složena z barev v červeném, zeleném a modrém vrcholu. Odstíny šedi odpovídají bodům na diagonále krychle spojující černý a bílý vrchol.



Obrázek 1.7: Geometrická reprezentace prostoru RGB (vlevo) a CMY (vpravo)

Prostor RGBA

V počítačové grafice se setkáváme s další zkratkou, která připomíná prostor RGB. Jedná se o kombinaci *RGBA* (či výstižněji *RGB*α). Tato zkratka je používána pro vyjádření skutečnosti, že barevný obraz zapsaný v prostoru RGB je doplněn informací o průhlednosti. Každý barevný bod takového obrazu s sebou nese skalární údaj (např. v rozmezí 0–1) , který určuje, v jakém rozsahu pokrývá barva plochu obrazového bodu. Hodnota 0.0 znamená neprůhledný barevný bod, maximální hodnota (1.0) zcela průhledný. Při zobrazení samotného obrazu nemá složka α (α*–kanál, α–channel*) význam. Pojem RGBα neznamená změnu barevného prostoru, nýbrž přidání další informace. Složka α se ukládá do rozsahu jednoho i více bytů. Používá se zejména při kombinování více obrazů do jednoho celku (část 4.4.1).

Prostory CMY, CMYK

Prostor RGB je technicky orientovaný prostor, vhodný pro displeje. Lidská zkušenost s mícháním barev však vychází ze zcela jiné práce s barvami. Například pro malíře je běžné, že nové barvy vytváří mísením jednotlivých barevných pigmentů, přičemž každé přidání pigmentu vytvoří tmavší barvu. Toto skládání barev se nazývá *subtraktivní*. Složením všech barev vznikne černá, což je právě opačná situace oproti aditivnímu skládání barevného světla.

Míchání barviv je typické nejen pro malíře, ale i pro tiskařské techniky. Těmto účelům vyhovuje prostor CMY, obsahující tři základní barvy: tyrkysovou neboli modrozelenou (C, *cyan*), fialovou (M, *magenta*) a žlutou (Y, *yellow*). Také tento prostor lze popsat jednotkovou krychlí (obr. 1.7). Sčítání hodnot CMY ovšem odpovídá subtraktivnímu skládání barev, takže vrchol $[1, 1, 1]$ reprezentuje černou barvu.

Převod mezi prostory RGB a CMY je jednoduchý. Vyjádříme-li barevný vektor v prostoru RGB tříprvkovou maticí $[r\ g\ b]$, určíme vektor $[c\ m\ y]$ v prostoru CMY odčítáním matic:

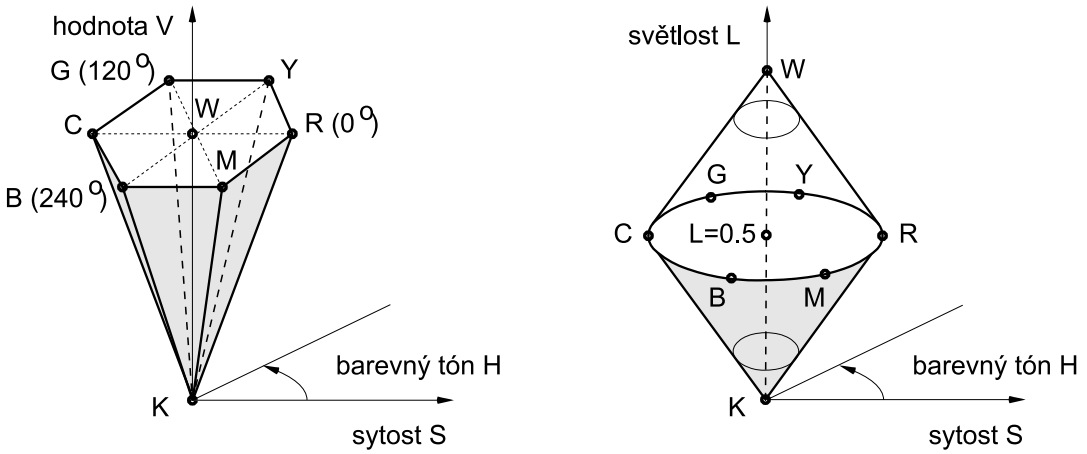
$$\begin{bmatrix} c \\ m \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} r \\ g \\ b \end{bmatrix}.$$

Při tisku jsou barevné obrazy reprodukovány jako soutisk tří obrazů, tvořených základními barvami C, M a Y. Tyto základní barevné pigmenty nesmí být dokonale krycí, neboť nové barvy vznikají vzájemným překrýváním. Složením všech tří barev proto při tisku nevznikne požadovaná černá, ale pouze špinavě hnědá. Mimoto není ekonomické skládat černou barvu ze tří jiných barev. Z těchto důvodů se černá tiskne jako samostatná barva. Lze ji použít i ke ztmavení ostatních barev. Od prostoru CMY se tak v polygrafii přechází k prostoru *CMYK* přidáním černé (K, *blacK*) jako čtvrté základní barvy. Velikost černé složky pro daný barevný bod získáme jako minimální hodnotu ze složek c, m a y, které poté snížíme o k. Tento jednoduchý postup získání hodnoty k však bývá v praxi většinou upravován s ohledem na použitá barviva a způsoby jejich mísení.

Při práci s barvami je důležitá nejen volba základních barev, ale i způsob jejich kombinování (mísení). Pokud jsou barvy zpracovávány pouze počítačem, je jejich geometrická reprezentace ve tvaru krychle výhodou. Horší je situace, má-li zadat do počítače barvu v prostoru RGB či CMY přímo uživatel. Představy o tom, jaká barva vznikne smícháním základních složek, jsou u různých lidí poměrně odlišné. Není například jednoduché odhadnout, jak změnit pro danou barvu její odstín tak, aby sytost zůstala zachována, apod. Pro tyto účely vznikly další barevné prostory, které jsou blízké intuitivnímu (lidskému) popisu barev.

Prostory HSV a HLS

Oba prostory HSV i HLS definují barvu trojicí složek, které však tentokrát nepředstavují základní barvy. Třemi hlavními parametry prostoru *HSV* jsou *barevný tón* (H, *hue*), *sytost* (S, *saturation*) a *jasová hodnota* (V, *value*). Barevný tón označuje převládající spektrální barvu, sytost určuje příměs jiných barev a jas je dán množstvím bílého (bezbarvého) světla. Pro zobrazení prostoru se nepoužívá krychle, nýbrž šestiboký jehlan (obr. 1.8 vlevo), jehož vrchol leží v počátku soustavy souřadnic HSV. Souřadnice s a v se mění od 0 do 1, souřadnice h reprezentuje úhel a nabývá hodnot z intervalu $\langle 0°, 360° \rangle$. Vrchol jehlanu představuje černou barvu. Jas roste směrem k podstavě, střed podstavy reprezentuje bílou barvu. Sytost odpovídá relativní vzdálenosti bodu od osy jehlanu. Dominantní barvy (se sytostí jedna) tedy leží na plášti, čisté barvy jsou na obvodu podstavy. Při pohybu po obvodu ve stejné výši od základny se postupně mění barevný tón, sytost a jas zůstávají nezměněny.



Obrázek 1.8: Geometrická reprezentace prostoru HSV (vlevo) a HLS (vpravo)

Prostor HSV vykazuje některé nedostatky, které sice nejsou zásadního charakteru, nicméně mohou ztěžovat práci s přesným určením barvy. Jedním z nedostatků je jehlanovitý tvar, který způsobuje, že ve „vodorovném“ řezu se musí bod o konstantní hodnotě s pohybovat při změně h po dráze ve tvaru šestiúhelníku, nikoliv po kružnici, jak by bylo přirozené. Dalším záporným jevem je nesymetrie prostoru z hlediska jasu. Tyto nedostatky odstraňuje *prostor HLS*, jehož geometrie je uvedena na obrázku 1.8 vpravo.

Název prostoru je odvozen z pojmů *barevný tón* (H, *hue*), *světlost* (L, *lightness*) a *sytost* (S, *saturation*). Prostor HLS je obdobou prostoru HSV, v němž byl jehlan nahrazen dvojicí kuželů. Barevný tón je opět vyjádřen úhlovou hodnotou, světlost se mění od 0 (černá v dolním vrcholu) do 1 (bílá v horním vrcholu). Sytost nabývá na povrchu kuželů hodnoty 1 a klesá na 0 směrem k ose kuželů. Nejjasnější čisté barvy mají tedy souřadnice $s = 1$ a $l = 0.5$ a leží na obvodu podstav kuželů.

Tvar prostoru HLS plně odpovídá skutečnosti, že nejvíce různých barev vnímáme při „průměrné“ světlosti (oblast podstav). Schopnost rozlišit barvy klesá jak při velkém ztmavení, tak při přesvětlení (oblasti obou vrcholů kuželů). Další dobrá vlastnost prostoru HLS spočívá v analogii míchání barev přidáváním černých a bílých pigmentů k základním spektrálním barvám. Zvýšení světlosti při nezměněné sytosti si lze představit jako přidání jistého množství bílých a ubrání stejného množství černých pigmentů. Samotné zvýšení sytosti odpovídá odebrání stejného množství bílých a černých pigmentů.

Prostory HSV a HLS umožňují postupně měnit barevné charakteristiky při zachování ostatních typických vlastností barvy. To je příjemné pro uživatele, kteří chtějí definovat barvy přirozenými pojmy, jako je sytost, světlost a dominantní barva. Prostory jsou vhodné

i pro vzorníky barev, kde jsou zobrazovány řezy jehlanem, či kuželem pro různé hodnoty h.

Převod barvy z prostorů HLS a HSV do prostoru RGB má charakter algoritmu. Tento nepříliš složitý algoritmus je uveden např. v [Fole90]. Připomeňme, že převod není prostým zobrazením. Pro barvy ležící na ose jehlanu HSV či kuželu HLS není hodnota h jednoznačně určena. V takových případech se zavádí zvláštní hodnota *nedefinováno*. Existují i „zakázané" kombinace, například při velikosti $s = 0$ nesmí nabývat složka h žádné platné hodnoty a může být pouze *nedefinována*.

1.2.2 Barevné prostory pro televizní a videotechniku

S dosud uvedenými barevnými prostory se setkáváme ve většině programů zpracovávajících barevné rastrové obrázky. Následující skupina prostorů je méně známá. Jsou používány hlavně tam, kde se stýká počítačová grafika a televizní technika, tj. v oblasti animací, filmových triků, multimedií apod.

Prostor YUV

Základní prostor YUV se používá pro přenos televizních signálů v normě PAL. Do stejné rodiny patří další prostory – YIQ pro americkou televizní normu NTSC a prostor $\mathrm{YC_BC_R}$ pro normu SECAM. Jejich společným rysem je oddělení jasové (luminanční) složky od barevných informací (chrominance). Toto uspořádání dovoluje přímo využít stejný jasový signál Y jak pro barevné, tak i černobílé televizory. Zbývající dva signály nesou informace o velikosti barevných složek obrazu.

Nepleťme si písmeno Y v těchto prostorech s označením žluté barvy (*yellow*) v prostoru CMY. Zde totiž písmeno Y znamená celkový jas dané barvy. S ním jsme se setkali již ve vzorci (1.1) v podobě písmene I.

Prostor YUV je někdy označován též [Y, B-Y, R-Y], resp. číselně $[Y, 0.493(B-Y), 0.877(R-Y)]$. Z prostoru RGB získáme hodnoty YUV jednoduchým maticovým násobením:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.141 & -0.289 & 0.437 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Prostor $\mathrm{YC_BC_R}$

S tímto prostorem se v počítačové grafice setkáme při zápisu rastrových obrázků ve formátu JPEG (viz kapitola 2.6).

Podle standardu CCIR–601 má hodnota Y být v intervalu $\langle 0.0, 1.0\rangle$ a hodnoty C_B, C_R v intervalu $\langle -0.5, 0.5\rangle$. Pokud chceme složky C_B a C_R při počítačovém zpracování ukládat jako celá čísla v rozsahu 0–255, musíme je nejprve posunout do kladné poloosy přičtením konstanty.

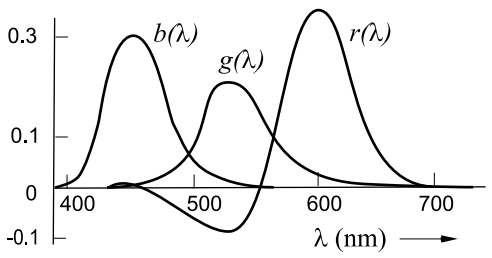
$$\begin{bmatrix} Y \\ C_B \\ C_R \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Zkráceně lze definovat hodnoty C_B a C_R jako

$$C_B = 0.5643\ (B - Y) \quad C_R = 0.7133\ (R - Y)$$

1.2.3 Chromatický diagram CIE

Aby bylo možno barvy měřit a vyjadřovat bez ohledu na subjektivitu lidského vnímání, byl v roce 1931 vytvořen mezinárodní standard základních barev ([Murc87]).



Obrázek 1.9: CIE 1931: Srovnávací funkce RGB

Jeho součástí je tzv. chromatický diagram CIE 1931 (CIE je zkratka Mezinárodní komise pro osvětlení – *Commission Internationale de l'Éclairage*). Ve standardu se předpokládá, že každou barvu lze definovat váženým součtem tří primárních barev, které jsou vyzařovány monochromatickými zdroji světla o vlnových délkách $R = 700\,nm$, $G = 541.1\,nm$ a $B = 435.8\,nm$. Libovolná monochromatická barva ve viditelném spektru je určena vztahem

$$C = r.R + g.G + b.B$$

Pomocí kolorimetrických experimentů zaměřených na srovnávání takto vytvořených barev s danými monochromatickými barvami byly stanoveny barevné srovnávací funkce $r(\lambda), g(\lambda), b(\lambda)$ s průběhy na obrázku 1.9.

Monochromatická barva o vlnové délce λ je na základě porovnání určena jako

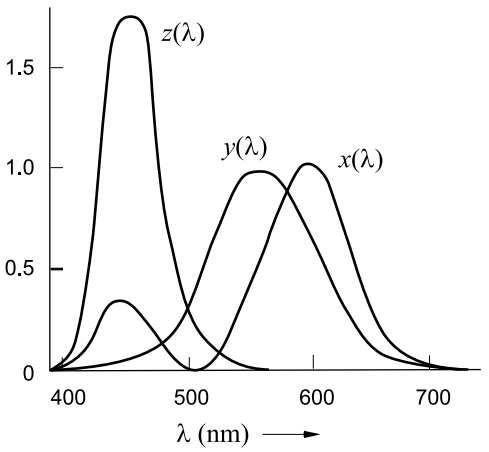
$$C_\lambda = r(\lambda) + g(\lambda) + b(\lambda)$$

a souřadnice obecné barvy se spektrálním rozložením energie $P(\lambda)$ zjistíme pomocí

$$r \;=\; k \int_\lambda P(\lambda) r(\lambda) d\lambda$$

$$g = k \int_\lambda P(\lambda) g(\lambda) d\lambda$$
$$b = k \int_\lambda P(\lambda) b(\lambda) d\lambda \,,$$

kde k je vhodná konstanta. Některé monochromatické barvy však nelze pomocí srovnávacích funkcí $r(\lambda), g(\lambda), b(\lambda)$ vytvořit. Subjektivně stanovená hodnota funkce $r(\lambda)$ nabývá v části viditelného spektra záporné hodnoty, namíchané barvy zde mají při porovnání s monochromatickou barvou „přebytek" červené, i když jsou vytvořeny pouze pomocí zelené a modré složky. Proto byly ve standardu CIE 1931 zavedeny umělé barevné srovnávací funkce s průběhy podle obrázku 1.10. Každá barva s daným spektrálním rozložením je pomocí těchto funkcí vyjádřena vahami X, Y, Z získanými ze vztahů:



Obrázek 1.10: CIE 1931: Srovnávací funkce XYZ

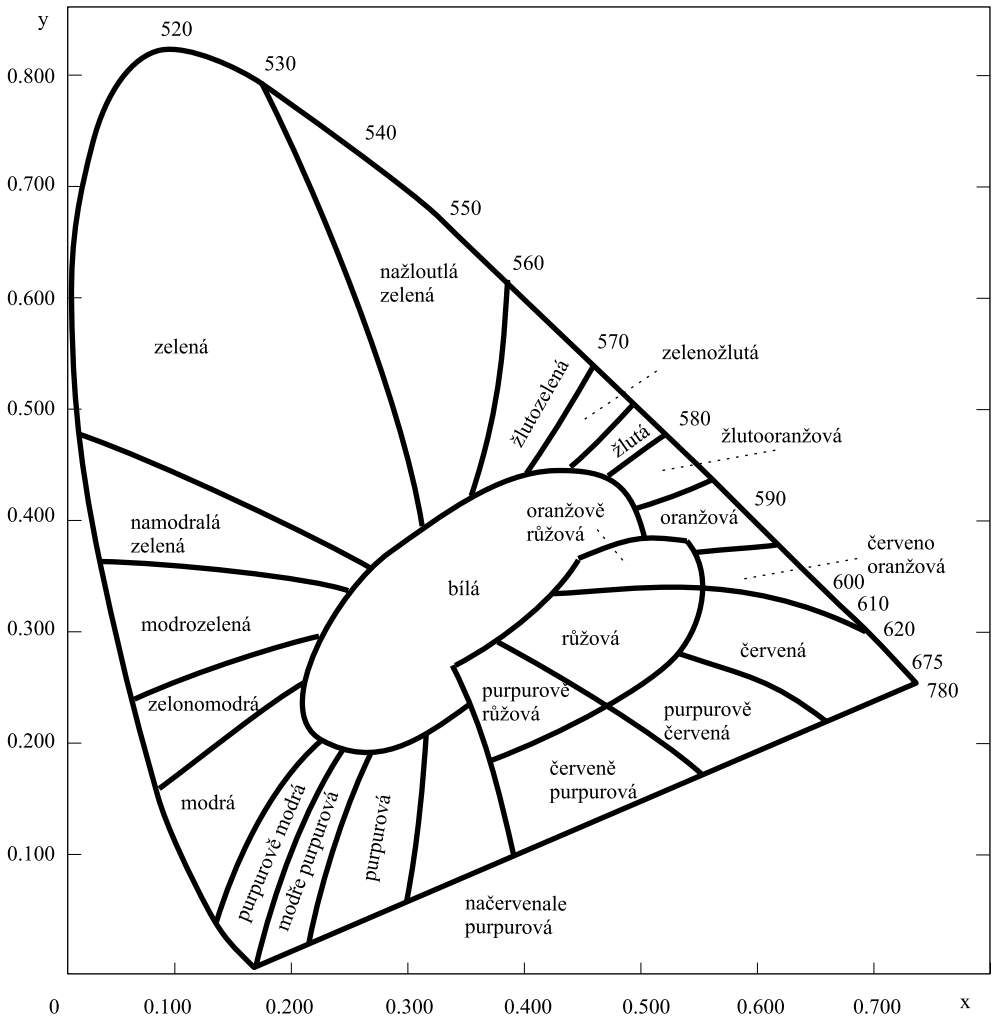
$$X = k \int_\lambda P(\lambda) x(\lambda) d\lambda$$
$$Y = k \int_\lambda P(\lambda) y(\lambda) d\lambda$$
$$Z = k \int_\lambda P(\lambda) z(\lambda) d\lambda$$

Hodnota $k = 680$ je stanovena pro objekty s vlastním vyzařováním. Subjektivně vnímatelné barvy vyjádřené v souřadnicích barevného prostoru CIE 1931 XYZ se nacházejí v tělese znázorněném na obrázku 1.12.

Po převodu trojice X, Y, Z do normalizovaného tvaru získáme barevné souřadnice

$$x = \frac{X}{X+Y+Z}, \quad y = \frac{Y}{X+Y+Z}, \quad z = \frac{Z}{X+Y+Z}. \tag{1.2}$$

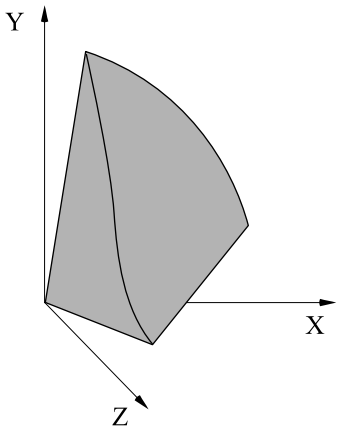
Protože platí, že $x + y + z = 1$, stačí k úplnému určení barvy souřadnice x, y doplněné o hodnotu Y. Všechny barvy (až na jasovou složku Y, pomocí níž lze zrekonstruovat původní hodnoty X, Y, Z) reprezentujeme souřadnicemi x, y ve dvourozměrném diagramu. Křivka, která ohraničuje barvy viditelného spektra, vymezuje oblast *chromatického diagramu CIE 1931 xyY* (obr. 1.11).



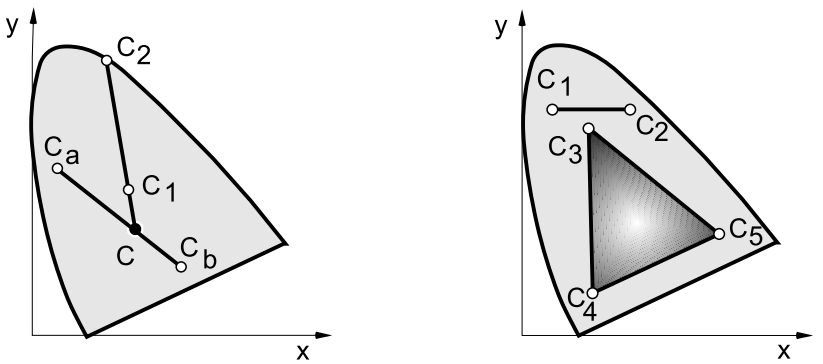
Obrázek 1.11: Barevný prostor CIE 1931 xyY (dle [Murc87])

Barevné body tvořící obalovou spektrální křivku jsou označeny vlnovou délkou v nanometrech počínaje červenou částí spektra a konče fialovou částí spektra. Bod C v diagramu odpovídá poloze bílého světla. V anglické terminologii je tento bod označován jako *illuminant C* a je používán jako standard pro průměrné denní světlo. Chromatický diagram poskytuje prostředky pro kvantitativní určení sytosti a dominantní vlnové délky, stejně jako z něj lze odvodit další barevné veličiny a vztahy.

Pro libovolný barevný bod, jako například C_1 na obrázku 1.13 vlevo, definujeme *sytost barvy* jako relativní vzdálenost barevného bodu od standardního bílého světla C. Měření vzdálenosti se provádí na polopřímce spojující body C a C_1 a protínající obalovou křivku v bodě C_2. Barva C_1 na obrázku je sytá asi z 25 %, protože leží přibližně v jedné čtvrtině orientované úsečky mezi C a C_2. *Dominantní vlnová délka* jakékoliv barvy je definována jako vlnová délka na spektrální křivce protínající úsečku spojující C a příslušný barevný bod C_1. Pro barevný bod C_1 na obrázku 1.13 vlevo je dominantní vlnová délka v bodě C_2. *Doplňkové barvy* jsou reprezentovány libovolnými koncovými body úsečky, která prochází bodem C, na obrázku 1.13 vlevo dvojice barev C_a, C_b. Pokud mají dvě doplňkové barvy stejnou sytost (relativní vzdálenost od C), vznikne jejich složením bílé světlo C.



Obrázek 1.12: CIE 1931: Těleso barev



Obrázek 1.13: CIE 1931: Sytost, dominantní vlnová délka a doplňkové barvy (vlevo), barevné rozsahy (vpravo)

Barevné rozsahy jsou v chromatickém diagramu reprezentovány konvexními množinami, které odpovídají konvexním kombinacím zvolených barev, které mícháme v různém poměru. Při míchání 2 barev se jedná o úsečky, které spojují barevné body odpovídající základním barvám. Na obrázku 1.13 vpravo je nakreslena spojnice mezi body C_1 a C_2. Všechny body na této úsečce lze získat složením různých intenzit základních barev C_1 a C_2. Podobně lze všechny vnitřní body trojúhelníka $C_3 - C_4 - C_5$ vytvořit kombinací základních barev v jeho vrcholech. Takový trojúhelník uvnitř chromatického diagramu se nazývá *barevný rozsah* (*color gamut*).

Jak vyplývá z geometrie chromatického diagramu, není možno nalézt takové tři základní barvy, které by určovaly trojúhelník pokrývající celý diagram. To odpovídá skutečnosti, že z konečného počtu základních barev nelze vytvořit všechny existující barvy. Počítačem generované obrazy proto obsahují méně barev než skutečný svět.

Gamut ve tvaru trojúhelníku je speciálním případem omezeného prostoru barev. Pokud má zařízení k dispozici více základních barev (např. tiskárna s více než 4 barevnými inkousty), má barevný gamut obecnější tvar. Mimo to závisí gamut i na technologii tvorby zobrazení a specifických vlastnostech použitých médií (kvalita papíru, rychlost zasychání inkoustu atd.).

Barevný prostor CIE 1976 Luv

V roce 1976 byl definován barevný prostor CIE 1976 Luv (*Uniform Color Space*), v němž jsou barvy rozloženy rovnoměrněji, než v prostoru CIE 1931. Rovnoměrnější rozložení znamená, že prostorová vzdálenost mezi dvojicí barev v prostoru CIE 1976 lépe odpovídá subjektivně vnímaným rozdílům intenzity mezi barevnými odstíny. Tento barevný prostor je ukázán na obrázku 1.14.

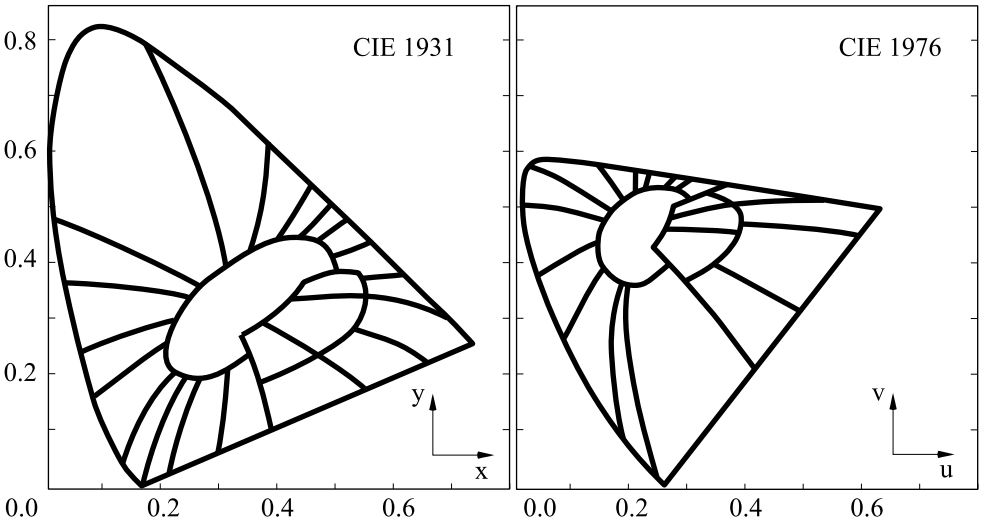
Převod hodnot x,y z barevného prostoru CIE 1931 na souřadnice u',v' v prostoru CIE 1976 je definován transformací

$$u' = \frac{2x}{6y - x + 1.5} , \quad v' = \frac{4.5y}{6y - x + 1.5} .$$

Zpětná transformace CIE 1976 na CIE 1931 je

$$x = \frac{27u'}{4.(4.5u' - 12v' + 9)} , \quad y = \frac{3v'}{4.5u' - 12v' + 9} .$$

V prostoru CIE 1976 je definována metrika, která umožňuje určit vizuální rozdíl, *barevný kontrast*, mezi dvěma barvami. Metrika stanoví barevný kontrast jako poměr vzdáleností porovnávaných barev od referenčního bodu (u_0, v_0) zvolené bílé barvy. Vzdálenosti jsou měřeny ve 3D prostoru Luv podle následujících vztahů. Světlost barvy L je odvozena z poměru jasové složky Y $[cd/m^2]$ vůči základní úrovni osvětlení Y_0 $[cd/m^2]$ a je definována



Obrázek 1.14: Uniformní barevný prostor CIE 1976 Luv (dle [Murc87])

$$L = 116 \left(\frac{Y}{Y_0} \right)^{1/3} - 16 \ .$$

Pro dvě barvy se souřadnicemi $[L_1, u_1, v_1]$ a $[L_2, u_2, v_2]$ jsou jejich relativní souřadnice vzhledem k referenčnímu bílému bodu $[L_0, u_0, v_0]$ určeny vztahy

$$\Delta u_1 = 13 L_1 (u_1 - u_0), \quad \Delta v_1 = 13 L_1 (v_1 - v_0),$$
$$\Delta u_2 = 13 L_2 (u_2 - u_0), \quad \Delta v_2 = 13 L_2 (v_2 - v_0).$$

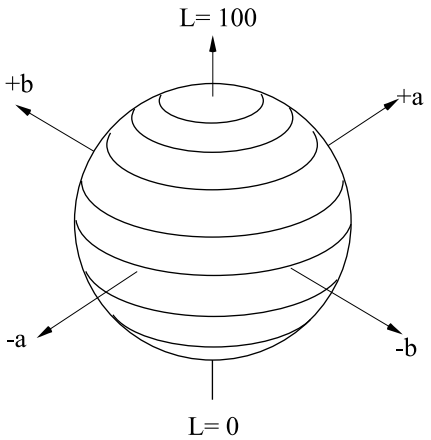
Kontrastní metrika ΔE, která měří euklidovskou vzdálenost v barevném prostoru CIE 1976, je definována jako

$$\Delta E = \sqrt{(L_1 - L_2)^2 + (\Delta u_1 - \Delta u_2)^2 + (\Delta v_1 - \Delta v_2)^2} \ .$$

Barevný prostor CIE 1976 L*a*b*

Dříve uvedený uniformní barevný prostor CIE 1976 Luv je určen pro definici barev vytvářených mícháním barevných zdrojů s vlastním zářením. Prostor CIE 1976 L*a*b* (též CIELAB, Lab)

naproti tomu slouží pro vyjádření barev materiálu (tzv. povrchových barev) podobně, jako tomu bylo u prostorů HSV a HLS. Vnímaná barva je určena souřadnicemi v 3D barevném prostoru. Osa L*, zvaná též osa světlosti, pokrývá rozsah od 0 (černá) do 100 (bílá). Další souřadnice a* a b* reprezentují rozsahy *zelená*(-a) až *červená*(+a) a *modrá*(-b) až *žlutá*(+b). V porovnání s prostorem CIE 1931XYZ odpovídá popis a rozložení barev v prostoru CIE 1976 L*a*b* lépe barvám vnímaným lidským zrakem. Uspořádání prostoru je na obrázku 1.15.



Obrázek 1.15: Uniformní barevný prostor CIE 1976 L*a*b*

Jednotlivé složky mohou být nastavovány individuálně. Je-li například nějaký obraz reprezentovaný ve formátu Lab, můžeme změnit jeho barevný tón, aniž by současně došlo ke změně jeho jasu. Protože prostor CIE 1976 L*a*b* nezávisí na zařízení, při převodu z RGB (použitý displej) na CMYK (použitá tiskárna) se provede převod prostřednictvím prostoru CIE 1976 L*a*b*. Hodnoty L*, a* a b* získáme z tristimulů X,Y,Z transformacemi

$$\begin{aligned} L^* &= 116(Y/Y_n)^{1/3} - 16 \\ a^* &= 500[(X/X_n)^{1/3} - (Y/Y_n)^{1/3}] \\ b^* &= 200[(Y/Y_n)^{1/3} - (Z/Z_n)^{1/3}] \end{aligned}$$

kde X_n, Y_n a Z_n jsou X,Y,Z hodnoty zářiče, který byl použit pro výpočet barevného vzorku a X/X_n, Y/Y_n a Z/Z_n jsou všechny větší než 0.008856. Pro menší hodnoty se používá trochu odlišná soustava rovnic.

1.2.4 Barvy a monitory

Ačkoliv prostory CIE, RGB, CMY a jejich varianty definují přesně polohu barevného bodu v barevném prostoru, při zobrazení barev musíme vzít v úvahu ještě další odlišnosti. Různá technická zařízení mívají různé základní barvy, čemuž odpovídají různé polohy a velikosti trojúhelníků barevných rozsahů. Pomocí těchto trojúhelníků můžeme porovnávat barevné možnosti různých zařízení, například monitorů. Monitory nejsou standardizovány a shodné trojice RGB hodnot budou na různých monitorech zobrazeny jako rozdílné barvy. Je to způsobeno především těmito faktory:

- Barva na monitoru není vytvářena smícháním tří světel, ale k superpozici dochází až v oku, které vnímá a integruje u každého obrazového bodu signály ze tři blízkých světelných

zdrojů (např. trojice RGB luminoforů na stínítku obrazovky). Důsledkem je to, že syté barvy nelze zobrazit s plným jasem, neboť např. sytá červená je prezentována jedním červeným bodem a dvěma tmavými body.

- Zdroje světla v monitorech mají různé spektrální charakteristiky; podle požadované doby dosvitu se volí látky s odlišným chemickým složením.
- Vztah mezi intenzitou zobrazení na monitoru a hodnotami RGB není lineární. Charakteristiky je nutné kompenzovat pomocí nelineární transformace – *gama korekce*, viz část 4.5.1.
- Při syntéze obrazu mohou být vytvořeny barvy, které leží mimo barevný rozsah monitoru. Tyto barvy je nutné nějak převést do zobrazitelného rozsahu, a to je též nelineární transformace. Podrobněji viz část 4.3.1.

Z uvedeného je také zřejmé, že obraz popsaný v některém z aditivních prostorů (CIE XYZ, RGB) nemusí být stejně vytištěn na tiskárně v barevném režimu (CMYK). Barevné rozsahy těchto zařízení jsou odlišné a i přes výpočetní korekce dochází ke ztrátám či posunům barev. Připomeňme, že také lidský zrak má svůj vlastní barevný rozsah, neboť barvy detekuje třemi různými druhy receptorů v oku.

Problematikou zpracování barevného obrazu s ohledem na změnu různých barevných charakteristik se zabývá kapitola 4.5. Další informace o barevných prostorech lze nalézt např. v [Glas95, Skal93].

Kapitola 2

Obraz a jeho reprezentace

Nejobyčejnější pojmy jsou obyčejně obtížně definovatelné a nejinak je tomu s obrazem. Definice obrazu se liší podle své aplikační oblasti a většina disciplín používá definice *ad hoc*, které nejlépe vyhovují jejich potřebám. V dalším textu budeme chápat obraz intuitivně, jako průmět reálného světa na sítnici oka, jako fotografii, obrazovku počítače, obrázek na papíře, či odraz v nehybné vodní hladině. Při formálním vymezení pak použijeme matematický model, kterým je spojitá funkce dvou proměnných – *obrazová funkce* [Gonz87]:

$$z = f(x, y). \tag{2.1}$$

Protože u obrazu předpokládáme omezené rozměry, můžeme definiční obor obrazové funkce zapsat jako kartézský součin dvou spojitých intervalů z oboru reálných čísel, které vymezují rozsah obrazu. Obrazová funkce realizuje zobrazení

$$f : (\langle x_{min}, x_{max}\rangle \times \langle y_{min}, y_{max}\rangle) \to H \ .$$

Reálná čísla x, y, kde $x : x_{min} \geq x \geq x_{max}$ a $y : y_{min} \geq y \geq y_{max}$ jsou souřadnice bodu ve dvou rozměrech, v nichž funkce nabývá nějaké hodnoty z z oboru hodnot, tj. $z \in H$.

Hodnotou obrazové funkce může být jediné reálné číslo (například jas, intenzita červené barvy, atp.), v počítačové grafice je to ale obvykle více hodnot, typicky trojice červená, zelená a modrá složka obrazové informace v modelu RGB (viz kapitola 1). V některých případech může z reprezentovat celé spektrum hodnot (například data z počítačového tomografu, či spektrální měření svitu hvězd), jindy je dokonce výhodné pracovat s komplexními čísly. Funkce tedy může nabývat hodnot zapsaných jako uspořádaná n-tice údajů $z = [z_1, z_2, \ldots, z_n]$, což zapíšeme ve tvaru

$$f : (\langle x_{min}, x_{max}\rangle \times \langle y_{min}, y_{max}\rangle) \to (H_1 \times H_2 \times \ldots \times H_n) \ .$$

Při práci s obrazem v počítačové grafice máme zřídka k dispozici spojitý definiční obor funkce (2.1). Obyčejně častěji pracujeme v rastru, který je složen z obrazových elementů, *pixelů*. Termín pixel je zkratkou z anglického *picture element*. Pojem pixel je v počítačové grafice oblíbený a vedl k rozšíření analogií. Například prvek textury se jmenuje texel, objemový element voxel, povrchový element surfel atp.

V praxi je obor hodnot funkce (2.1) různě omezen, například použitou aplikací, fyzikálními vlastnostmi technického zařízení, které obraz vytváří či pořizují, aj. V počítačové grafice se nejčastěji pracuje s 16 miliony barev, s 256 stupni šedi, s 256 barvami, v DTP se někdy pracuje jen s dvojicí barev, atp.

V dalším textu uvedeme základní pojmy z teorie vzorkování, uvedeme alias a vysvětlíme způsoby jeho zmenšení. Pro zjednodušení často nebudeme uvažovat obor hodnot funkce (2.1) složený z více údajů, ale budeme pracovat pouze s jedinou hodnotou, například odstínem šedi.

Obrazová funkce (2.1) je definována na podmnožině dvojrozměrného prostoru. V mnoha případech je však výklad názornější při použití jednorozměrného definičního oboru. V dalším textu budeme proto hovořit o spojité funkci jedné proměnné $f(x)$ a jejím diskrétním obrazu I_i později zobecníme na spojitou funkci dvou proměnných $f(x, y)$ a jí odpovídající diskrétní funkci $I_{i,j}$.

2.1 Digitalizace

Většinu modelů v počítačové grafice lze s výhodou charakterizovat nějakou spojitou funkcí. Obraz na obrazovce je však diskrétní. Jedním ze základních dějů, který je spjat se získáváním počítačového obrazu, je přechod od spojité funkce $f(x, y)$ k diskrétní funkci $I_{i,j}$ a to jak v definičním oboru funkce $f(x, y)$, tak v jejím oboru hodnot H. Za příklad může posloužit digitální videokamera či digitální fotoaparát. Obraz, který je snímán, má teoreticky nekonečný rozsah obrazových hodnot, stejně jako je možné ho přibližovat či vzdalovat téměř neomezeně. Bude však zobrazen v konečném množství pixelů a s konečným množstvím barev. Proces přechodu od spojitého obrazu k diskrétnímu se nazývá *digitalizace* a odehrává se ve dvou nezávislých krocích, jimiž jsou *kvantování* a *vzorkování*.

2.1.1 Kvantování

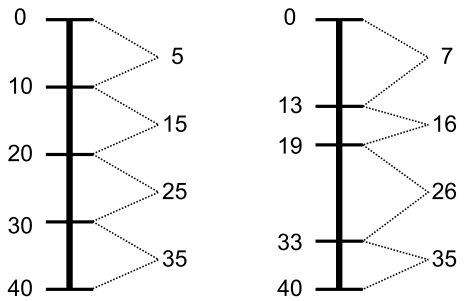
Kvantování probíhá v oboru hodnot obrazové funkce (2.1), který se rozdělí na intervaly, jimž je pak přidělena jediná, zástupná hodnota. Podle způsobu rozdělení kvantované veličiny hovoříme o kvantování *uniformním* a *neuniformním*, viz obrázek 2.1.

Uniformní kvantování používá konstantní délku intervalu, zatímco neuniformní kvantování používá proměnnou délku intervalu. Neuniformní kvantování se používá méně často, i když

umožňuje zohlednit nerovnoměrné rozložení hodnot měřené veličiny. Důvodem je jednodušší realizovatelnost uniformního kvantování v technickém vybavení počítačů a přístrojů pořizujících digitální obraz. Jednou z důležitých aplikací neuniformního kvantování je „rekvantizace“ obrazu. Jedná se o případy, kdy převádíme obraz s velkým barevným rozlišením na obrázky s menším počtem barev, či obraz s velkým dynamickým rozsahem do intervalu zobrazitelného počítačem (viz část 4.1.2).

Způsob výběru zástupné hodnoty závisí na použité aplikaci a na jejích cílech. Nejčastěji se používá průměr celého intervalu, někdy je zastupující hodnotou vážený průměr, medián, průměr z okrajů, aj.

Při kvantování dochází ke ztrátě informace. Množina hodnot je nahrazena jedinou hodnotou. Tato ztráta se označuje jako *kvantizační chyba* a v počítačové grafice se projevuje například na plochách s malou změnou gradientu jako náhlý skok barev, jak demonstruje obrázek 2.2.



Obrázek 2.1: Uniformní (vlevo) a neuniformní kvantování.

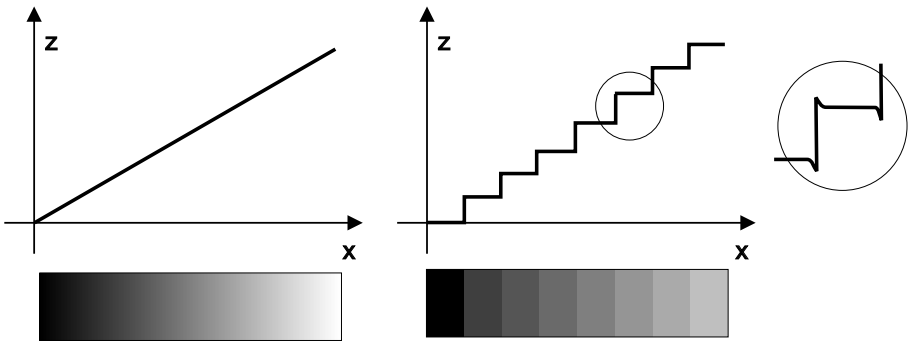
Původně hladký barevný přechod je nahrazen skokovou změnou. Faktor, který tuto chybu zesiluje, je lidské vnímání. Oko je citlivé na výskyt hran a vnímá tento přechod jako novou informaci v obraze. Změny gradientu ovlivňují vnímání přilehlých ploch s konstantním jasem. Tento jev byl poprvé popsán na konci osmnáctého století fyzikem Ernstem Machem, a proto se mu říká Machovy proužky (*Mach-band effect*). Kvantizační chyba působí rušivě. Uniformní kvantování nijak nezohledňuje hodnoty uvnitř intervalu a proto se u něj tato chyba obyčejně projevuje více. Adaptivní výběr zástupné hodnoty může kvantizační chybu částečně odstranit.

2.1.2 Vzorkování

Vzorkováním (*sampling*) spojité funkce $f(x)$ rozumíme zaznamenávání hodnot – vzorků, v předem daných intervalech, tak jak je naznačeno na obr. 2.3. Jednorozměrnou funkci získanou pravidelným vzorkováním budeme označovat I_i a vzdálenost dvou vzorků označíme Δx. Původní spojitá funkce může být definována na libovolném intervalu $x \in \langle x_0, x_1 \rangle$. Vzorky budeme indexovat následujícím způsobem:

$$I_i = f(x_0 + i\Delta x), i = 0, 1, \ldots \tag{2.2}$$

Vzorkování je zcela běžnou technickou záležitostí. Například videokamera snímá scénu v diskrétních intervalech, digitální teploměr připojený k počítači zasílá hodnoty v předem



Obrázek 2.2: Kvantizační chyba. Původně hladký barevný přechod (vlevo) je nahrazen skokovým přechodem (uprostřed). V obraze vznikají hrany, které v původním signálu nebyly přítomné. Lidské vnímání jejich subjektivní intenzitu zesiluje (vpravo).

definovaných časových intervalech, hudba uložená jako soubor MP3 je posloupnost čísel reprezentujících úroveň signálu snímanou s relativně vysokou frekvencí a webová kamera snímá scénu například každých deset vteřin. Obraz získaný postupy v počítačové grafice je obyčejně výsledkem vzorkování trojrozměrné scény.

Při vzorkování dochází ke ztrátě informace. Předpokládejme, že funkce je konstantní a došlo k její prudké změně v okamžiku $f(x_0 + \Delta x/2)$. Funkce je vzorkována s krokem Δx. Hodnotě $f(x_0)$ odpovídá vzorek I_0 a $f(x_0 + \Delta x)$ vzorek I_1, přesněji $f(x_0)$=I_0 a $f(x_0 + \Delta x)$=I_1. Co se stalo s hodnotou v polovině vzorkovaného intervalu? Jednoduše není v posloupnosti výstupních vzorků zaznamenána. Je to podobné, jako když webová kamera sejme snímek opuštěné ulice. Poté projede auto a za okamžik kamera sejme další snímek. Oba snímky budou stejné a o události, která proběhla, se z nich nedozvíme nic.

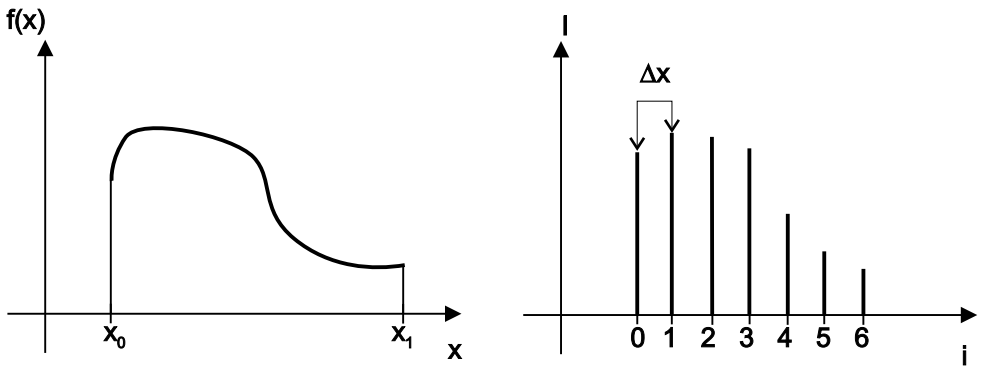
Výše popsaný způsob vzorkování, ve kterém je hodnota vzorku sejmuta v jediném bodě, se nazývá bodové vzorkování (*point sampling*) a je v počítačové grafice nejběžnější. Pokud se uvádí pojem vzorkování bez přívlastku, obyčejně se rozumí bodové vzorkování

Technicky je nejběžnější vzorkování kdy je zaznamenána reprezentace hodnot celého vzorkovaného intervalu Δx. Této metodě se říká plošné vzorkování (*area sampling*) a hodnota vzorku se určí měřením ze všech hodnot, například jako jejich průměr:

$$I_i = \frac{1}{\Delta x} \int_{x_0+i\Delta x}^{x_0+(i+1)\Delta x} f(t)dt. \tag{2.3}$$

Plošné vzorkování vyžaduje vyšší výpočetní náročnost než vzorkování bodové.

Výše jsme definovali vzdálenost dvou vzorků jako Δx. *Vzorkovací frekvencí* f_s rozumíme



Obrázek 2.3: Vzorkování

hodnotu

$$f_s = \frac{1}{\Delta x} \tag{2.4}$$

a její jednotkou je počet vzorků za jednotku času [Hz] či na jednotku vzdálenosti [dpi]. Zřejmě čím bude vyšší vzorkovací frekvence, tj. čím větší bude počet výsledných vzorků, v případě obrazu jeho rozlišení, tím bude paměťová náročnost reprezentace vyšší. Vyšším počtem vzorků zaznamenáme více detailů z původní spojité funkce. Otázka je, zda je možné v diskrétní mřížce rastru nějaký signál reprezentovat naprosto přesně.

2.2 Fourierův obraz

Fourierova transformace slouží k převodu obrazu do duálního prostoru, který zjednodušuje některé operace s obrazem a je vhodný pro pochopení některých jevů a vlastností obrazu. V této části budeme používat dvě reprezentace. Budeme pracovat s obrazem, který je reprezentován tak jak ho známe, tj. v diskrétní matici pixelů. Této reprezentaci budeme říkat *prostorová oblast*. V teorii signálů se rovněž používá pojem *oblast časová*. *Fourierův obraz* je reprezentací obrazu v *frekvenční oblasti*. Tato reprezentace je obecně složením nekonečně mnoha sinusových signálů, které mají různou amplitudu a jsou různě fázově posunuté. Analogií je reprezentace zvuku jako signálu, který se skládá z „hloubek“ (signálů nízké frekvence) a „výšek“ (signálů frekvence vysoké). Čím silnější jsou „hloubky“, tím větší je amplituda nízkých frekvencí, a naopak čím silnější jsou „výšky“ tím větší je amplituda frekvencí vysokých. Šum, který je obyčejně nežádoucí, se projevuje jako vysoké frekvence, naproti tomu nežádoucí nízkofrekvenční signál budeme nazývat *alias* a jeho vznik budeme nazývat *aliasing* (odstavec 2.3). Většinu následujících pojmů budeme definovat v jednorozměrném případě; zobecnění do vyšších rozměrů je snadné.

2.2.1 Spojitá Fourierova transformace

Výpočet, kterým získáme Fourierův obraz z funkce reprezentované v časové oblasti, se nazývá *dopředná Fourierova transformace*; původní obraz získáme z Fourierova obrazu *zpětnou Fourierovo transformací*.

Fourierovým obrazem jednorozměrné funkce $f(x)$ definované na spojitém definičním oboru, budeme rozumět komplexní funkci $F(u)$, kterou získáme integrací

$$F(u) = \int_{-\infty}^{+\infty} f(x) e^{-i2\pi ux} dx, \tag{2.5}$$

kde $i = \sqrt{-1}$ je komplexní jednotka. Této integraci se říká dopředná *Fourierova transformace*.

Zpětná Fourierova transformace komplexní integrovatelné funkce $F(u)$, která realizuje přechod od frekvenční oblasti k prostorové, je definována vztahem

$$f(x) = \int_{-\infty}^{+\infty} F(u) e^{+i2\pi ux} du. \tag{2.6}$$

Fakt, že $f(x)$ a $F(u)$ jsou dvojím vyjádřením téže funkce svázané relací Fourierovy transformace, bývá někdy zvykem označovat:

$$f(x) \Longleftrightarrow F(u)$$

2.2.2 Diskrétní Fourierova transformace

Při práci se signálem, který je dán omezenou posloupností vzorků, která se periodicky opakuje, používáme diskrétní Fourierovu transformaci. Přímá a zpětná *diskrétní Fourierova transformace* (DFT) funkce I_k je definována jako

$$\begin{aligned} F_n &= \frac{1}{N} \sum_{k=0}^{N-1} I_k e^{-i2\pi \frac{nk}{N}} \\ I_k &= \sum_{n=0}^{N-1} F_n e^{+i2\pi \frac{nk}{N}}, \end{aligned} \tag{2.7}$$

Dualitu mezi prostorovou I_k a frekvenční reprezentací F_n obrazu vyjádříme pomocí

$$I_k \Longleftrightarrow F_n$$

Výpočet DFT pomocí algoritmu rychlé Fourierovy transformace je popsán v části 22.7.2. V následujícím textu budeme předpokládat, že vlastnosti obrazu a jeho úpravy popsané pomocí spojité FT zjistíme a realizujeme v případě diskrétního obrazu pomocí DFT.

2.2.3 Fourierova transformace a obraz

Fourierova transformace je dekompozicí původní funkce $I(x)$ na různě fázově posunuté sinusové funkce s různou amplitudou. Připomeňme Eulerovu formuli:

$$e^{\pm i2\pi ux} = \cos 2\pi ux \pm i \sin 2\pi ux.$$

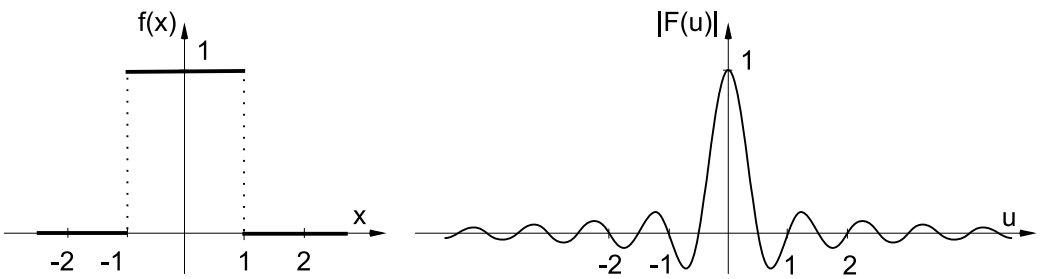
Nyní definujeme dva důležité pojmy, které charakterizují obraz. Mějme bod o souřadnici $u = Re(u) + iIm(u)$ komplexního Fourierova obrazu. Jeho reálnou část označíme $Re(u)$ a imaginární část $Im(u)$. *Amplitudové spektrum* funkce $F(u)$ označujeme $|F(u)|$

$$|F(u)| = \sqrt{Re^2(u) + Im^2(u)} \tag{2.8}$$

a *fázové spektrum* je funkce

$$\varphi(u) = atan\left(\frac{Re(u)}{Im(u)}\right).$$

Amplitudové spektrum určuje velikosti jednotlivých frekvencí (sinusových složek) a fázové spektrum určuje jejich fázové posuny. Bod ve Fourierově oblasti, který má frekvenci u [Hz], má tedy amplitudu $|F(u)|$ a fázový posun $\varphi(u)$. Obrázek 2.4 ukazuje příklad funkce v časové oblasti (vlevo) a část jejího amplitudového spektra.



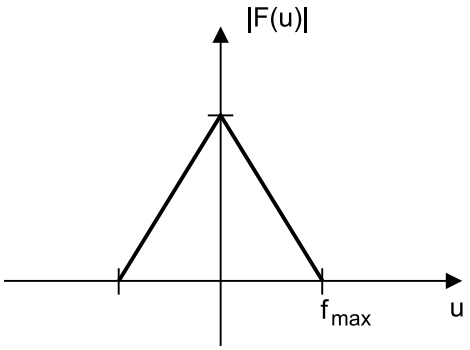
Obrázek 2.4: Funkce $f(x)$ (vlevo) a část jejího amplitudového spektra $|F(u)|$.

2.2.4 Shannonův vzorkovací teorém a frekvenčně omezená funkce

Frekvenčně omezená funkce (*bandlimited function*) má konečné amplitudové spektrum. V konečném amplitudovém spektru existuje nejvyšší frekvence (označme ji f_{max}) taková, že pro všechny frekvence u, pro které platí že $u > f_{max}$ je $|F(u)| = 0$. Amplituda těchto frekvencí je nulová a tyto frekvence nenesou žádnou energii. Obrázek 2.5 demonstruje příklad frekvenčně

omezené funkce. Nejjednodušším případem frekvenčně omezené funkce je samozřejmě přímo funkce $sin(x)$, které ve Fourierově spektru odpovídá jediný bod.

Nejvyšší nenulové frekvenci f_{max} se říká *Nyquistovo kritérium*. Vzorkovací frekvenci f_s [viz vztah (2.4)] a nejvyšší frekvenci f_{max} v signálu uvádí do vztahu *Shannonova vzorkovací věta* [Gonz87] (též Shannonův vzorkovací teorém), která zní:



Obrázek 2.5: Amplitudové spektrum $|F(u)|$ frekvenčně omezené funkce.

Signál spojitý v čase je plně určen posloupností vzorků odebíraných ve stejných intervalech Δx, je-li jejich frekvence $f_s = 1/\Delta x$ větší nežli dvojnásobek nejvyšší frekvence v signálu f_{max}, tj. je-li

$$f_s > 2f_{max}. \tag{2.9}$$

Každá frekvenčně omezená funkce je reprezentována od určité velikosti rastru přesně a další zvyšování vzorkovací frekvence nevede k přidání detailů a je tedy zbytečné. Většina scén v počítačové grafice *není* frekvenčně omezená.

2.2.5 Konvoluce

Dalším důležitým pojmem je konvoluce, kterou budeme označovat operátorem $\star$. Konvoluce dvou funkcí $I(x)$ a $h(x)$ je definována jako

$$I(x) \star h(x) = \int_{-\infty}^{\infty} I(x-\alpha)h(\alpha)d\alpha. \tag{2.10}$$

Funkci $h(x)$ se říká konvoluční jádro. Konvoluční jádro můžeme přirovnat k oknu, které se posouvá po obraze. Hodnoty konvolučního jádra určují způsob výpočtu nového pixelu v obraze. Konvolučnímu jádru se také říká (*windowed function*).

Je-li Fourierovým obrazem funkce $I(x)$ funkce $F(u)$ a obrazem $h(x)$ funkce $H(u)$, pak je obrazem funkce $I(x) \star h(x)$ součin $F(u)H(u)$ a naopak obrazem $F(u) \star H(u)$ je součin $I(x)h(x)$:

$$\begin{aligned} F(u) \star H(u) &\iff I(x)h(x), \\ F(u)H(u) &\iff I(x) \star h(x) \end{aligned}$$

Zobecnění spojité konvoluce do dvou rozměrů je snadné. Při práci s digitálním obrazem se však používá *diskrétní konvoluce*, která je diskrétní dvojrozměrnou podobou integrálu (2.10).

$$I'_{i,j} = I_{i,j} \star h_{i,j} = \sum_{x=-k}^{k} \sum_{y=-k}^{k} I_{i-x,j-y} h_{i,j} \tag{2.11}$$

Konvoluční jádro diskrétní konvoluce lze popsat jako tabulku o rozměrech $\langle -k, k\rangle \times \langle -k, k\rangle$. Význam konvoluce je následující. Máme vstupní obraz $I_{i,j}$ a konvoluční jádro $h_{i,j}$. Výstupní obraz $I'_{i,j} = I_{i,j} \star h_{i,j}$ získáme tak, že na každý bod funkce $I_{i,j}$ položíme konvoluční jádro $h_{i,j}$ a vypočítáme součet (2.11). Tento postup je základem mnoha operací s diskrétním obrazem, například odstraňování šumu (odstavec 4.6.1), detekce hran (odstavec 4.6.2), aj.

Jednou z vlastností konvoluce je, že se pro obraz provede konstantní počet operací bez ohledu na jeho obsah. To konvoluci zvýhodňuje při použití v paralelních výpočtech. Většina dnešních grafických akcelerátorů podporuje rozšíření OpenGL verze 1.2 pro práci s obrazy (*OpenGL Imagining Subset*), které provádí konvoluci přímo na grafické kartě.

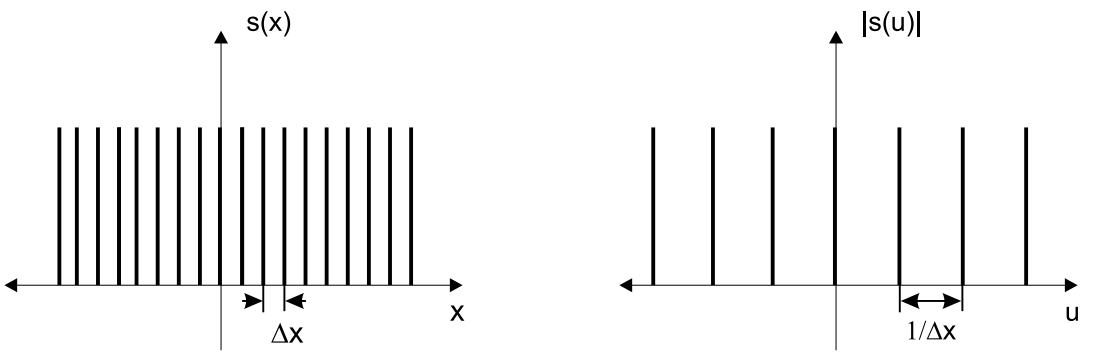
Vzorkování v obrazové oblasti je možné popsat jako konvoluci v oblasti frekvenční. Mějme funkci $s(x)$, které budeme říkat *vzorkovací funkce*, nebo vzorkovací hřebínek. Jeden vzorek se získá pomocí *Diracova pulsu* $\delta(x)$, který je definován jako $\delta(x) = 0$ pro všechna $x \neq 0$ a $\int_{-\infty}^{\infty} \delta(x)dx = 1$. Vzorkovací funkce $s(x)$ (viz obrázek 2.6) je potom nekonečnou posloupností Diracových pulsů v konstantní vzdálenosti Δx

$$s(x) = \sum_{i=-\infty}^{\infty} \delta(x - i\Delta x). \tag{2.12}$$

Fourierovým obrazem funkce (2.12) je funkce

$$s(u) = \frac{1}{\Delta x} \sum_{i=-\infty}^{\infty} \delta\left(\frac{u - i}{\Delta x}\right). \tag{2.13}$$

Z obrázku 2.6 je patrné, že zvyšování vzorkovací frekvence (zmenšování vzdálenosti mezi dvěma pulsy) způsobuje zvětšování vzdálenosti mezi Fourierovými obrazy dvou pulsů.



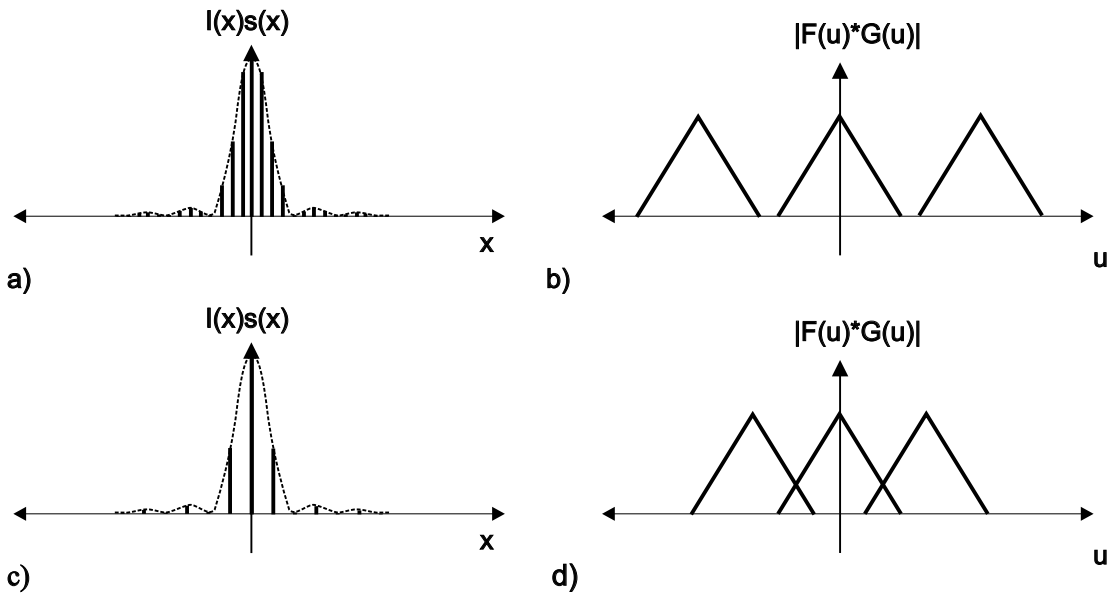
Obrázek 2.6: Posloupnost Diracových pulsů (vlevo) a jejich amplitudové spektrum.

Vzorkování funkce $f(x)$ vzorkovací funkcí $s(x)$ lze zapsat jako součin $f(x)s(x)$. Fourierův obraz tohoto součinu má tvar

$$I(x) = f(x)s(x) \Longleftrightarrow \frac{1}{\Delta x} \sum_{i=-\infty}^{\infty} F\left(\frac{u-n}{\Delta x}\right). \tag{2.14}$$

Fourierův obraz posloupnosti vzorků funkce $I(x)$ (2.14) je složen z obrazů funkce $I(x)$, které se opakují ve vzdálenosti Fourierových obrazů pulsů vzorkovací funkce.

Obrázek 2.7 demonstruje tento fakt pro dva případy. Funkce (vlevo) má Fourierův obraz trojúhelník. V prvním případě (a-b) je vzorkovací frekvence taková, že se Fourierovy obrazy $F(u)$ funkce $I(x)$ nepřekrývají. Ve druhém případě (c-d) dochází k jejich překrytí. Po zpětné rekonstrukci se v signálu projeví alias.



Obrázek 2.7: Vzorky původně spojité funkce (a,c) a jejich amplitudové spektrum (b,d). V případě nízké vzorkovací frekvence dochází k překrytí kopií Fourierových obrazů.

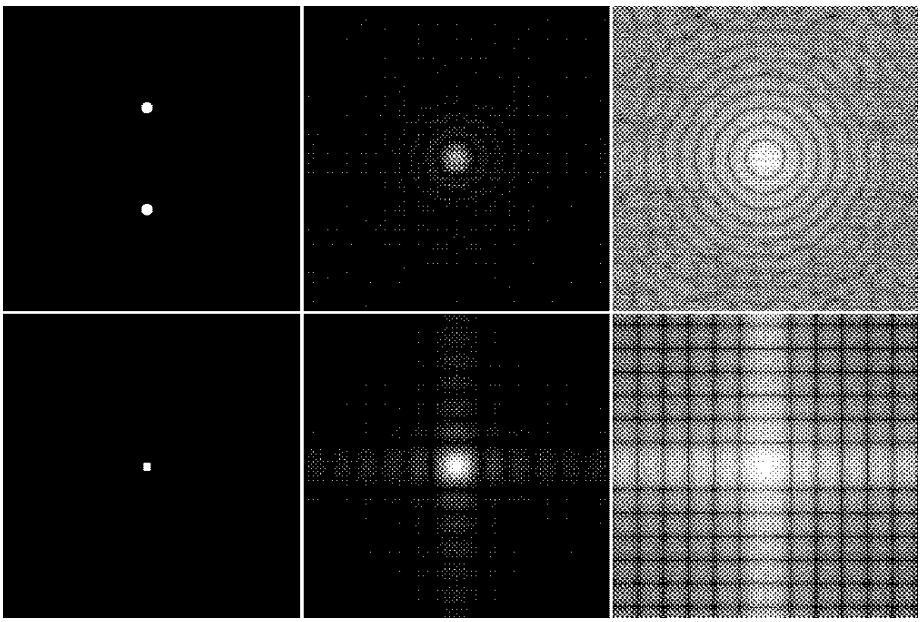
Chceme-li funkci zrekonstruovat z jejich Fourierových vzorků, je nutné, aby tyto vzorky ve frekvenční oblasti nebyly poškozeny vzájemným překrytím. To je zaručeno, pokud jsou splněny dvě podmínky:

- Funkce je frekvenčně omezená, maximální frekvence obsažená v signálu je $f_{max} < \infty$.

- Původní funkce byla vzorkována minimálně dvojnásobkem maximální frekvence f_{max}. Jen v tomto případě totiž nedojde k překrytí frekvenčních spekter. Jinými slovy řečeno, frekvenčně omezené kopie (jejichž šířka je $2f_{max}$, jak je patrné z obrázku 2.5) se nebudou překrývat, pokud jsou od sebe vzdáleny minimálně $\Delta x = 1/2f_{max}$. Neboli

$$f_s > 2f_{max}.$$

Tato nerovnost je vyjádřením již zmíněné Shannonovy vzorkovací věty (2.9).



Obrázek 2.8: Příklad (zleva) dvojrozměrných binárních funkcí (dva bílé kroužky a jeden čtverec) a jejich amplitudových spekter $|F(u,v)|$. Amplitudové spektrum poměrně rychle klesá se vzdáleností od počátku, a proto je na obrázcích vpravo zvýrazněno funkcí $\log(1+|F(u,v)|)$.

V celé této kapitole jsme základní principy vzorkování demonstrovali na jednorozměrném případě. Obrazy používané v počítačové grafice jsou však dvojrozměrné. Zobecnění Fourierovy transformace do dvou dimenzí je snadné. Dvojrozměrný Fourierův obraz spojité funkce $I(x,y)$ označíme jako $F(u,v)$. Dopředná a zpětná transformace mají tvar

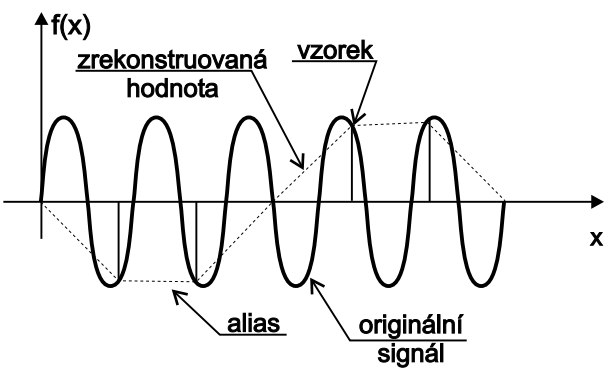
$$\begin{aligned} F(u,v) &= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} I(x,y)e^{-i2\pi(ux+vy)}dxdy, \\ I(x,y) &= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} F(u,v)e^{+i2\pi(ux+vy)}dudv. \end{aligned} \tag{2.15}$$

Zatímco v jednorozměrném případě měla hodnota $F(u)$ význam jediné funkce sinus s určitou amplitudou a frekvencí, ve dvojrozměrném případě se jedná o běžící vlnu ve směru $atan\,(v/u)$ o amplitudě $\sqrt{u^2+v^2}$. Amplitudové spektrum má obvykle maximum v počátku a klesá se vzdáleností (viz obrázek 2.8). Omezená funkce bude v amplitudovém spektru od jisté vzdálenosti od počátku nulová. Další informace k této problematice je možno vyhledat například v [Gonz87, Watt92].

2.3 Alias

Alias je jedním z nejdůležitějších pojmů počítačové grafiky. Vzniká při rekonstrukci signálu vzorkovaného pod Nyquistovým limitem a projevuje se jako nová, nízkofrekvenční informace, která nebyla v původním signálu přítomna. Prakticky se jedná o případ, kdy originální funkce obsahuje detaily, které není možné v rastru zobrazit. Alias vzniká ve dvou případech:

1. Pokud je původní funkce frekvenčně neomezená, tj. neexistuje žádná maximální frekvence a funkci není tedy možné v diskrétní mřížce rastru reprezentovat přesně.
2. Je-li původní funkce frekvenčně omezená, tj. v jejím Fourierově spektru existuje určitá maximální frekvence f_{max}, a tuto funkci vzorkujeme s frekvencí menší nežli $2f_{max}$, tedy pod Nyquistovým limitem.



Obrázek 2.9: Vznik aliasu

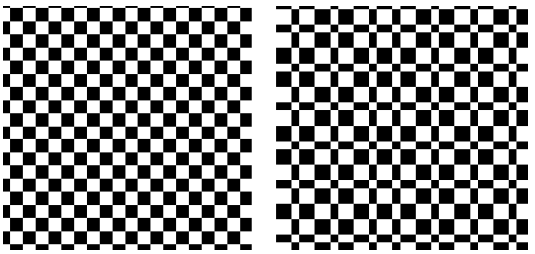
Druhý případ dokumentuje obrázek 2.9. Frekvenčně omezená spojitá funkce $sin(x)$ je podvzorkována a její hodnoty jsou rekonstruovány, tj. odhadnuty lineární interpolací. Výsledkem rekonstrukce je nová funkce, podobná funkci $sin(x)$, avšak s nižší frekvencí.

Příkladem aliasu v praktickém životě je televizní obrazovka snímaná kamerou. Vzhledem k tomu, že kamera snímá obraz v diskrétních časových intervalech a obrazovka promítá po půlsnímcích, dochází k interferenci frekvencí a následnému aliasu, který se na obrazovce projevuje jako tmavé, různou rychlostí nahoru a dolů se pohybující pruhy či blikání. Tento případ je projevem takzvaného časového aliasu (*temporal alias*).

V počítačové grafice se setkáváme s aliasem v mnoha situacích, například při vzorkování textur a při zobrazování objektů na jejich hranách. Zubatému zobrazení čar a okrajů polygonů

se v počítačové grafice říká „zubatice“ (*jaggies*). Při vzorkování šachovnice se může stát, že všechny vzorky zasáhnou pouze černé políčko a výsledkem takového vzorkování pak bude jednobarevná plocha. Při vzorkování textury, která obsahuje hustý pravidelný vzorek, se může někdy alias projevit jako tzv. moire. Jiným příkladem je objevování se a mizení malých objektů; tj. objektů, které jsou svou velikostí po průmětu do obrazu srovnatelné s velikostí pixelu, pro tyto objekty se vžil název *crowlies*.

Připomeňme, že alias je přidaná, nízkofrekvenční informace, která vzniká jako důsledek procesu vzorkování a rekonstrukce, jak ukazuje obrázek 2.10. Textura šachovnice byla zmenšena interpolací nejbližším sousedem na 23 % původní velikosti. Poté byla stejnou metodou zvětšena na původní velikost. Originál je na obrázku vlevo a výsledek experimentu na pravé straně. Jak je vidět, vznikly nové vzory, které se nepodobají původní šachovnici.



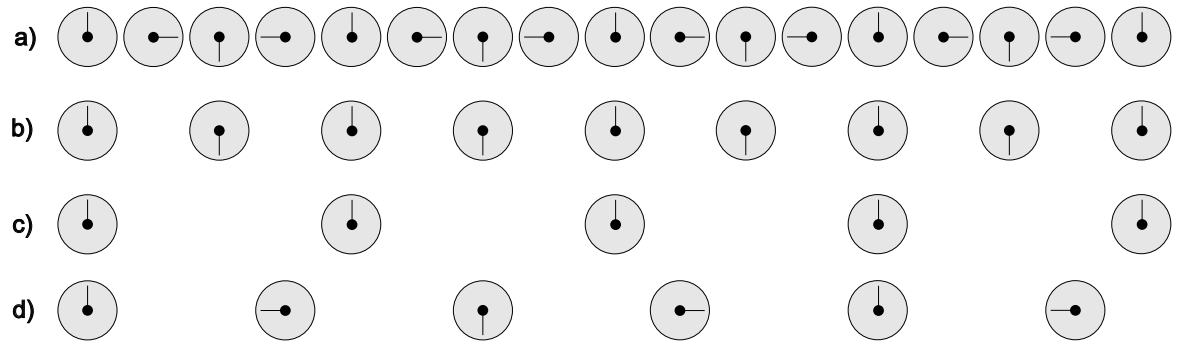
Obrázek 2.10: Příklad vzniku aliasu. Textura šachovnice (vlevo) je zmenšena a zvětšena na původní velikost. Výsledkem je nový vzor – alias.

V počítačové animaci se setkáváme především s časovým aliasem, kde může mít dosti bizarní projevy. Například běžící postava se může pohybovat a při tom nehýbat nohama, případně s nimi pohybovat v opačném směru aj. Obrázek 2.11 je příkladem tohoto druhu aliasu. Na prvním řádku je otáčející se ručička vzorkována tak, že v jejím pohybu nedochází k aliasu. Na řádku b) je vzorkována tak, že dochází k přeskokům z jedné strany na druhou, na řádku c) ručička stojí a konečně na posledním řádku se dokonce otáčí na opačnou stranu.

2.4 Antialiasing

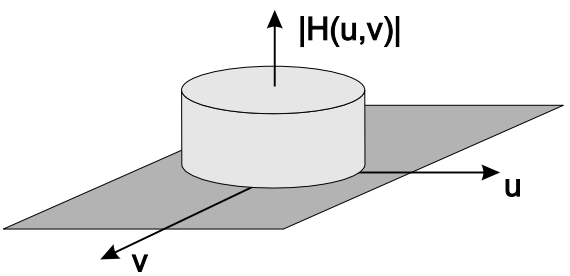
Odstranění či zmenšení aliasu se říká antialiasing. Nejjednodušší cestou je odstranit z obrazu informaci, kterou nelze vzorkovat, tj. odstranit vysoké frekvence. Pokud bude funkce neomezená, je možné její vysoké frekvence eliminovat odříznutím, tj. za pomoci filtru, který má ve dvojrozměrné Fourierově oblasti tvar válce se středem v počátku a jehož poloměr pak určuje, které frekvence se mají ořezat (viz obrázek 2.12). Takto upravený obraz vnímá pozorovatel jako rozostřený, ostré linie a přechody jsou rozmazané.

K tomu, abychom mohli se signálem takto manipulovat, je zapotřebí, aby byl spojitý. Počítačová grafika však pracuje téměř výhradně s diskrétní informací a spojitá funkce není obyčejně k dispozici. Spojitá funkce je aproximována numerickým řešením. Například při sledování paprsku (viz kapitola 15.9) jsou diskrétní hodnoty obrazu pořizovány ve středech



Obrázek 2.11: Alias v časové oblasti. Otáčející se ručička na řádku a) je vzorkována tak, že nedochází k aliasu. Na druhém řádku je vzorkována s poloviční frekvencí a dochází k vizuálnímu přeskoku ručičky z jedné strany na druhou. Na řádku c) ručička stojí a na posledním se otáčí na druhou stranu (podle [Watt92]).

pixelů a zpřesňování metody vede pouze k zvyšování počtu vzorků – spojitá informace však není k dispozici. Druhým důležitým faktorem je, že naprostá většina signálů převáděných na obraz je frekvenčně neomezená. Z toho je patrné, že se s aliasem setkáváme v počítačové grafice prakticky na každém kroku.



Obrázek 2.12: Ideální nízkofrekvenční filtr

Druhou cestou zmenšení aliasu je zvýšení hustoty vzorkování. Alias většinou (výjimkou jsou frekvenčně omezené funkce) nemůžeme odstranit, ale můžeme ho částečně potlačit. V počítačové grafice se nejčastěji používají dvě metody. První spočívá v posunutí aliasu k vyšším frekvencím, druhou metodou je převedení aliasu na šum. První se docílí pravidelným vzorkováním s vyšší frekvencí, u druhé metody se uplatňují náhodná čísla.

Rozšiřme rovnici jednorozměrného vzorkování (2.3) do dvou rozměrů

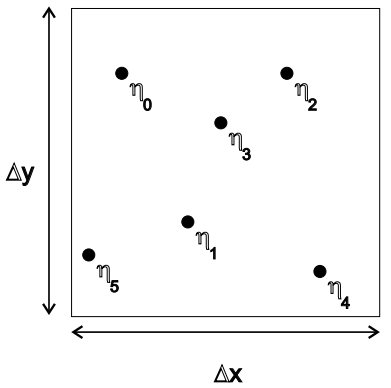
$$I_{i,j} = \frac{1}{\Delta x \Delta y} \int_{x_0+i\Delta x}^{x_0+(i+1)\Delta x} \int_{y_0+j\Delta y}^{y_0+(j+1)\Delta y} f(p,q)dpdq. \tag{2.16}$$

Plošné vzorkování vypočítává průměr ze vzorkované oblasti. Hodnota, která se takto získá, je přiřazena celé oblasti pixelu a bere v úvahu rozložení hodnot barvy či intenzity jasu ve vzorkované oblasti. Nejjednodušším a nejčastějším případem je bodové vzorkování ve středu pixelu, které nahrazuje výpočet (2.16) prostým odhadem na základě hodnoty v jediném bodě.

Tato metoda vede k silnému aliasu. Plošné vzorkování je obtížné neboť obyčejně nemáme spojitou funkci k dispozici. Z těchto důvodů se volí kompromisní techniky, které aproximují dvojitý integrál (2.16) součtem (viz obrázek 2.13)

$$I(i,j) = \sum_{k=1}^{n} w_k \eta_k. \tag{2.17}$$

Plochu pixelu $\Delta x \times \Delta y$ vzorkujeme n vzorky $\eta_k, k = 1, 2, \ldots, n$. Váhové koeficienty w_k určují vliv každého vzorku na výsledek. Podmínky, kladené na váhy jsou $\sum_{k=1}^{n} w_k = 1$ a $w_k > 0$. Poznamenejme, že v naprosté většině praktických případů jsou váhové koeficienty stejné pro všechny vzorky, čímž se rovnice (2.17) redukuje na výpočet prostého průměru. S rostoucím n a se zvyšuje přesnost vzorkování. Pokud je zároveň pro $w_k = 1/n$ tak pro $n \to \infty$ přechází (2.17) v (2.16).



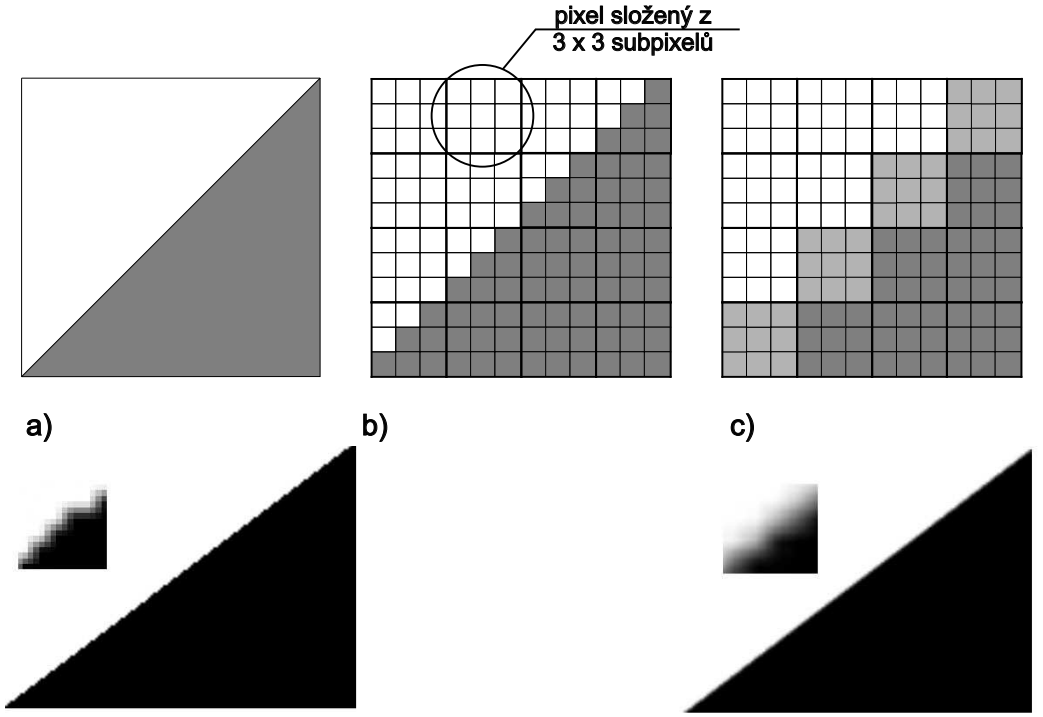
Obrázek 2.13: Vzorkování s vyšší frekvencí

Pokud se do oblasti pixelu umístí více jak jeden vzorek, hovoříme o vzorkování s vyšší frekvencí (*supersampling*). Podle způsobu rozmístění vzorků se jedná o vzorkování pravidelné (*regular supersampling*) nebo náhodné (*stochastic supersampling*, též *random supersampling*). Při vzorkování s vyšší frekvencí musíme pořídit více vzorků, nežli v případě bodového vzorkování s jedním vzorkem. Vzorkování s vyšší frekvencí vždy vyžaduje vyšší výpočetní výkon. Není to nic překvapujícího, k potlačení aliasu potřebujeme získat více informací.

2.4.1 Pravidelné vzorkování s vyšší frekvencí

FSAA, neboli celoobrazovkový antialiasing (*Full Screen AntiAliasing – FSAA*) je dnes běžnou záležitostí ve většině grafických karet. Princip této metody tkví v získání obrazu ve vyšším rozlišení (tedy v jeho přesnější reprezentaci), jeho filtraci a zmenšení. U pravidelného vzorkování je každý pixel rozdělen do menších superpixelů. Řekněme, že máme obraz v rozlišení 800×600 pixelů. Pokud nahradíme každý pixel 2×2 superpixely získáme superobraz v rozlišení 1600×1200 pixelů. Při výpočtu hodnot pixelů se vypočítá průměr ze získaných hodnot a tedy použijeme shodné $w_i = 0.25$ (viz rovnice (2.17)).

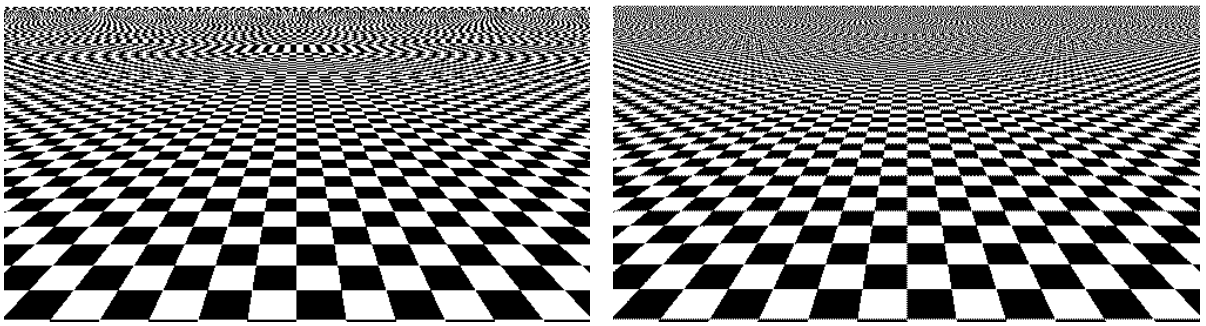
Uvedená metoda neodstraní alias, ale pouze ho posune k vyšším frekvencím. Filtrace se projevuje *rozmazáním* (*blur*) vysokých frekvencí. Výsledek pravidelného vzorkování s vyšší frekvencí je na obrázku 2.15. Na nekonečné šachovnici, která se perspektivně sbíhá směrem



Obrázek 2.14: Vzorkování s vyšší frekvencí. Původní spojitý signál a) je vzorkován ve vysokém rozlišení (každý pixel se skládá z devíti subpixelů) b). Virtuální obraz je filtrován c). Dolní řádek vlevo je polygon bez vyhlazování a vpravo vyhlazený. Malé čtverce uvnitř dolního obrazu jsou silně zvětšené detaily hrany.

k horizontu, se od určité vzdálenosti projeví alias jako série po sobě jdoucích čtverců stejné barvy. Tento obraz se často používá pro porovnávání různých metod antialiasingu. Jeho získání je jednoduché a obraz obsahuje nekonečně malé detaily. V konečném rastru se tedy alias musí někde projevit.

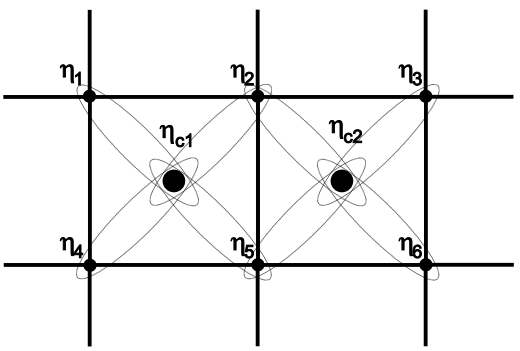
1x2, 2x1 je implementačně jednodušší antialiasing. Jak název napovídá, ve směru osy y resp. x se vzorkuje s dvojnásobnou přesností. Výsledkem je velice rychlý antialiasing, který však neposkytuje tak kvalitní výsledky jako předcházející algoritmus. Tato metoda bývá implementována zejména v levnějších grafických kartách či herních konzolích.



Obrázek 2.15: Vlevo obraz vzniklý vzorkováním jedním vzorkem ve středu pixelu, vpravo funkce vzorkovaná s vyšší frekvencí. Velikost superpixelu je 3×3. Z obrázku je patrné, že se alias stále objevuje, je však posunut směrem k vyšším frekvencím.

HRAA, Quincunx byl implementován firmou NVIDIA$^{\text{tm}}$ v jejích grafických kartách řady GeForce$^{\text{tm}}$. Quincunx znamená uspořádání pěti objektů tak, že čtyři v nich jsou v rozích a jeden v jejich středu (viz obrázek 2.16). HRAA, druhý název pro stejnou metodu, znamená antialiasing s vysokým rozlišením (*High Resolution AntiAliasing*).

Principem této metody je výpočet pěti vzorků pro každý pixel. Klasickým bodovým vzorkováním se nejprve zjistí hodnota ve středu pixelu (hodnota η_C) a pro každý z vrcholů pixelu se vypočítá ještě jeden vzorek. Výhodou a zásadní předností je, že hodnoty v rozích pixelů se sdílejí. V podstatě se tedy vzorkuje pouze ve dvojnásobném rozlišení; jednou pro středy pixelů a podruhé v rozích (plus jeden sloupec a řádek na okraji, ty jsou však vzhledem k velikosti obrazu zanedbatelné).



Obrázek 2.16: Quincunx

Quincunx nepoužívá konstantní váhu vzorku. Vzorek ve středu pixelu je vážen hodnotou $w_C = 1/2$ a ostatní vzorky $\eta_i = 1/8$, $i = 1, 2, \ldots 4$. Při podrobnějším pohledu si všimneme, že váhy vzorků definují dvojrozměrnou podobu lineární interpolace ve tvaru jehlanového stanu (*tent filter*). Výsledkem je velice kvalitní obraz při relativně malých výpočetních nákladech.

Nevýhodou tohoto algoritmu je právě to, co ho činí výpočetně rychlým; sdílení hodnot mezi pixely. Tato vlastnost může vést k nežádoucímu aliasu a moire na texturách s pravidelným vzorkem.

Akumulační paměť (*Accumulation Buffer*) je obraz s implementovanou množinou funkcí, která se provádí pro každý pixel. Typické operace prováděné s akumulační pamětí jsou přičtení obrazu, výpočet průměru z obsahu akumulační paměti a nového obrazu a vrácení hodnoty akumulační paměti. Akumulační paměť má různá využití, jedním z nich je i antialiasing.

1. Vynuluj akumulační paměť
2. Proveď $n\times$
   (a) Definuj polohu vzorků
   (b) Smaž obraz a zobraz scénu
   (c) Přičti 1/n násobek obrazu k akumulační paměti
3. Přenes obsah akumulační paměti do paměti obrazové

Algoritmus 2.1: Využití akumulační paměti pro antialiasing

Při antialiasingu musíme vypočítat více vzorků pro každý pixel. Předcházející metody to řeší výpočtem obrazu ve vyšším rozlišení. Akumulační paměť používá stejné rozlišení jako je výsledný obraz, ale obraz se musí zpracovat vícekrát.

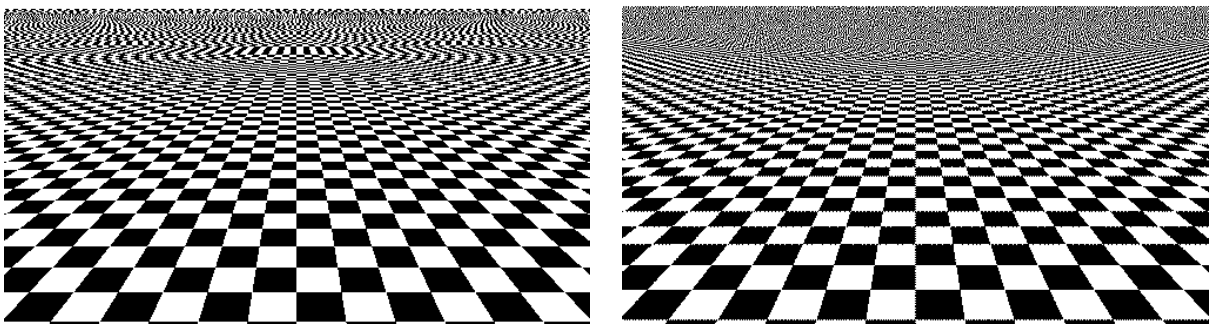
Nejdůležitější vlastností akumulační paměti je, že se nemusí nutně vzorkovat ve středu pixelu a tak je tato technika využitelná při náhodném vzorkování, které je popsáno v následující kapitole. Zde ji uvádíme proto, že se využívá při RGSS (*Rotated Grid Supersampling*), vzorkování s vyšší frekvencí s otočenou mřížkou. Alias, v případě objektů, je nejvíce patrný na hranách s malým sklonem, či naopak se sklonem blízkým 90^o. Metoda založená na otáčení mřížky vzorkuje každý pixel v bodech $[0, 1/4]$, $[1/2, 0]$, $[3/4, 1/2]$ a $[1/4, 3/4]$. Výsledkem je podstatně lepší vzorkování právě čar s kritickým sklonem.

Akumulační paměť je programovatelná v moderních API typu OpenGL a D3D, rovněž je běžně podporována v technickém vybavení grafických karet.

Akumulační paměť se rovněž využívá pro simulaci rozmazání pohybu (*motion blur*), rozmazání zaostřením (*field of view*) aj. Akumulační paměť je důležitým doplňkem metody distribuovaného sledování paprsku.

2.4.2 Stochastické vzorkování

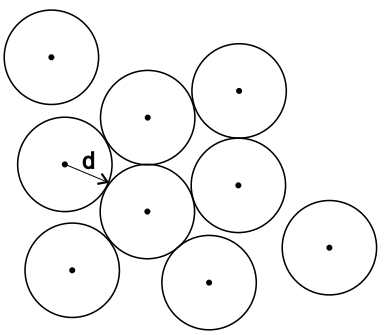
Pravidelné vzorkování přináší do obrazu novou informaci, která posune alias k vyšším frekvencím, ale pro frekvenčně neomezené funkce ho úplně neodstraní. Filtrace alias částečně rozmaže. Lidské vnímání je tolerantní k šumu, citlivě vnímá pravidelné chyby. Jednou z nejelegantnějších technik antialiasingu je tedy převod aliasu na šum tak, jak ukazuje obrázek 2.17.



Obrázek 2.17: Vlevo originální funkce, vpravo funkce vzorkovaná s vyšší frekvencí s roztřesením 3×3 vzorků. Oblasti, kde se vlevo projevuje alias jsou vpravo zašuměné.

Poisson disc. Alias se převede na šum buď stochastickým vzorkováním, nebo častěji stochastickým vzorkováním s vyšší frekvencí [Cook86]. Otázka je, jak by se měl původní spojitý obraz vzorkovat, aby šum byl co nejméně rušivý. Odpověď na tuto otázku nalezneme v rozložení tyčinek a čípků v lidském oku. Rozložení světločivých buněk [Yell82] odpovídá speciálnímu případu Poissonova rozložení, u kterého je zaručena minimální vzdálenost (označená d) mezi dvěma vzorky (*Poisson disc*) jak ukazuje obrázek 2.18

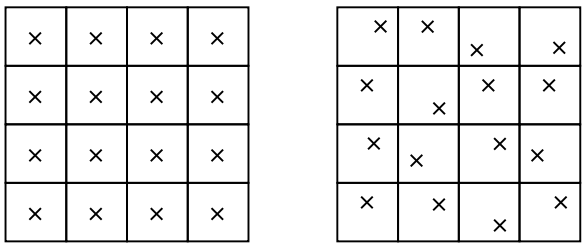
Naivní algoritmus spočívá v generování vzorku a testování, zda se v jeho blízkosti vyskytuje jiný vzorek. Pokud je tomu tak, vzorek se zruší a vygeneruje znovu. Při volbě umístění $n$tého vzorku musíme testovat $n-1$ předchozích vzorků, jedná se tedy o algoritmus se složitostí $\mathcal{O}(n^2)$. Podstatnější nevýhodou je, že podmínka vnitřního cyklu nemusí být pro velmi „husté scény“ splněna nikdy. Tento algoritmus sice generuje optimální rozložení, avšak jeho výpočetní náročnost ho činí prakticky nepoužitelným. Často se tedy několik rozložení vypočítá předem a uloží do tabulky. Voláním jediného náhodného čísla se vybere vzorek, který se aplikuje. Jedná se o generování pseudonáhodných vzorků a u těchto technik je kritická délka periody. Periodu lze prodloužit různými způsoby; permutací náhodných čísel, mícháním vzorků z tabulky mezi sebou aj.



Obrázek 2.18: Příklad Poissonova rozložení vzorků s minimální vzdáleností vzorků d

Roztřesení (*jittering*) je patrně nejčastěji používaným algoritmem stochastického antialisingu. Tato metoda generuje dobrou aproximaci optimálního rozložení vzorků. Může se používat

přímo, jako bodové vzorkování, nebo jako vzorkování s vyšší frekvencí. Vysvětlíme jeho princip na druhém případu.



Obrázek 2.19: a) Pravidelné rozložení vzorků a b) jejich roztřesení

Principem algoritmu je generování vzorků do superpixelů a jejich umístění do jejich středů stejně jako u pravidelného vzorkování s vyšší frekvencí (viz odstavec 2.4.1). Vzorky se ale náhodně posunou z okolí středu pixelu tak, aby žádný z nich neopustil hranici svého superpixelu. Roztřesení je tedy přidáním šumu do pravidelného vzorkování. Obrázek 2.19 demonstruje rozdíl mezi pravidelným a roztřeseným umístěním vzorků. Náhodná čísla, která se k posunu vzorků používají, by měla mít rovnoměrné rozložení, čímž se pokryje plocha superpixelu rovnoměrně.

Roztřesením je docíleno náhodného rozložení vzorků v pixelu při zachování jejich relativně rovnoměrného rozložení po celé ploše obrazu. Nejhorší případ, ke kterému může z hlediska vzájemné polohy dojít, je setkání čtyř vzorků v jednom rohu.

Výpočet polohy vzorků popisuje algoritmus 2.2.

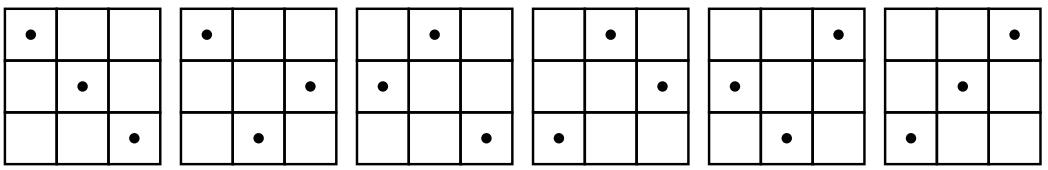
1. Umísti vzorky do středů superpixelů o straně délky Δx.
   $i-$tý vzorek má souřadnice x_i, y_i.
2. Pro všechny vzorky i proveď:
   (a) Generuj náhodná čísla $\delta x, \delta y$ s rovnoměrným rozložením taková že δx, $\delta y \in (-\Delta x/2, \Delta x/2)$
   (b) $x_i = x_i + \delta x$
   (c) $y_i = y_i + \delta y$

Algoritmus 2.2: Roztřesení

Někdy bývá šum nežádoucí (například při animaci pohybující se textury dochází k jejímu „va-

ření“), a proto se volí kompromis, tj. částečné roztřesení (*semi-jittering*), kdy se zmenší maximální vzdálenost o kterou se může vzorek v superpixelu posunout a amplituda šumu se tak zmenší.

N–věží Pravidelné vzorkování je rychlé, avšak stochastické vzorkování poskytuje vizuálně lepší výsledky, než vzorkování pravidelné. Kompromisem mezi pravidelným a stochastickým vzorkováním je metoda zvaná N–věží (*N–rooks*). Tato technika je jednou z mála metod stochastického vzorkování, která byla použita v grafickém technickém vybavení počítačů.



Obrázek 2.20: 3věžové vzorkovací schéma

Ze šachů je známa úloha rozmístit n věží tak, aby se navzájem neohrožovaly. Nejjednodušší je umístit je na diagonálu šachovnice a lze ukázat, že všechny možnosti jsou permutace diagonály tak, jak ukazuje případ tří věží na obrázku 2.20. Toto rozložení se uloží do tabulky a generováním jediného náhodného čísla se vybere jako vzorek. Toto vzorkovací schéma je vhodné pro odstranění aliasu hran se sklonem blízkým ose x nebo ose y. Superpixely mohou být vzorkovány ve svých středech, případně polohy vzorků mohou být ještě dále roztřeseny.

2.5 Reprezentace rastrového obrazu

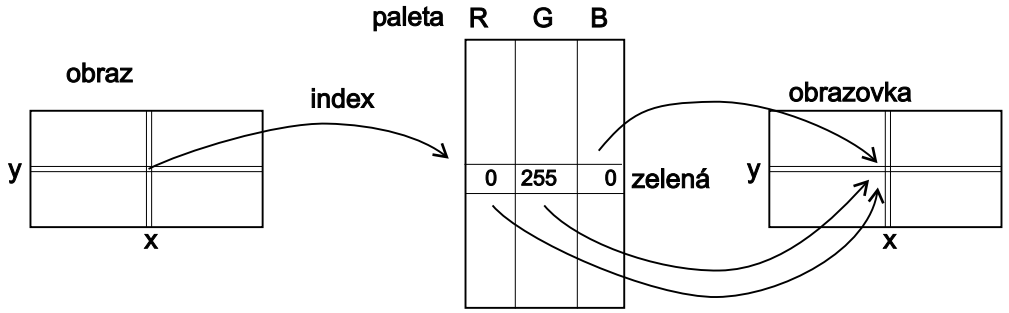
Nejjednodušší, nejsnáze pochopitelnou, ale nejméně úspornou reprezentací diskrétního obrazu je dvourozměrná matice bodů (pixelů), z nichž každý nabývá hodnot podle typu obrazu.

Pokud je každý pixel na obrazovce popsán jediným bitem, říkáme, že je takový obraz *monochromatický* neboli černobílý. Toto označení má své historické důvody a používá se, i když je poněkud zavádějící, protože binární informace nemusí reprezentovat právě černou a bílou barvu, ale dvě libovolné barvy.

Indexový mód je spojen s používáním tzv. *barevné palety* (*palette*), neboli *mapy barev* (*colormap*) na obrázku 2.21. U takového obrazu nereprezentuje hodnota pixelu přímo barvu, ale je ukazatelem do tabulky – barevné palety.

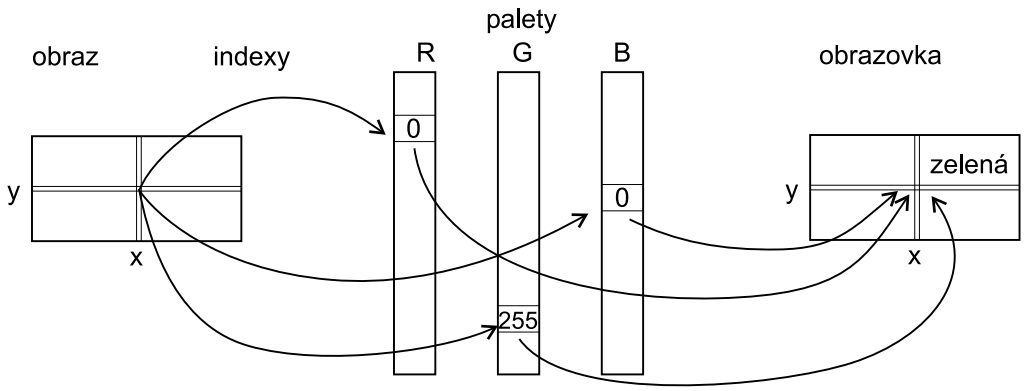
Barevná paleta je převodní tabulka, která určuje konkrétní barvu pixelu s daným indexem. V indexovém módu je index reprezentován jedním bytem, a proto má paleta maximálně 256 řádků. Paleta označovaná jako $8-8-8$ reprezentuje barvu ve třech bytech, po jednom pro každý barevný kanál. Tím je určeno, že každý bod obrazu může nabývat některé z 256 barev,

které se vybírají z celkového množství 2^{24} barev. Tento způsob přiřazení barev bývá také označován jako *pseudo color*. Barevná paleta je buď součástí obrazu nebo je vytvářena až při jeho zobrazování.



Obrázek 2.21: Reprezentace obrazu v indexovém módu; každý pixel obsahuje ukazatel do barevné palety

Další možností je reprezentace obrazu v odstínech šedi. Každý bod v obraze reprezentuje buď přímo odstín šedi (*static gray*), nebo je opět odkazem do palety (*gray scale*).



Obrázek 2.22: *Direct color* mód. Pixel je třemi hodnotami, které se interpretují jako indexy. Výsledná barva se získá mapováním přes tři barevné palety

Posledním případem je obraz, v němž je každý pixel reprezentován třemi barvami, nejčastěji v modelu RGB. *True color* obraz obsahuje přímo barevné hodnoty v jednotlivých pixelech. Naproti tomu barevný mód označovaný jako *direct color* pracuje s RGB hodnotami, které neslouží přímo k zobrazení, nýbrž jsou odkazem do barevných palet pro každou jednotlivou

barevnou složku tak, jak ukazuje obrázek 2.22. Při zobrazení musí tedy být k dispozici tři palety, pro každý barevný kanál jedna. Tento způsob je výhodný zejména proto, že umožňuje snadnou změnu všech barev, aniž bychom měnili hodnoty pixelů v obrazu a využívá se například při gama korekci [viz rovnice (4.20)].

2.6 Komprese rastrového obrazu

Rastrové obrazy se vyznačují vysokou paměťovou náročností, která roste kvadraticky s jejich rozlišením. Kompresi obrazů je tedy po právu věnována značná pozornost. Na rozdíl od komprese obecných souborů lze vycházet z vlastností a charakteristických rysů konkrétního rastrového obrazu. Nejprve popíšeme základní kompresní metody pro rastrové obrazy a poté i některé formáty. Počet grafických rastrových formátů je překvapivě vysoký, ačkoliv jejich společným cílem je ve většině případů pouze úschova dvourozměrného pole pixelů, reprezentujících obrázek. Existence mnoha formátů má několik příčin:

Historické důvody
: formáty odrážejí technický vývoj, zejména postupně se zvyšující barevné možnosti zařízení jak pro snímání obrazu, tak pro jeho zobrazení,

Vazba na program
: podle druhu aplikace vznikaly specializované formáty, například pro úschovu skic a kreseb (PCX), černobílých dokumentů (TIFF) či pro přenos barevných fotografií a jejich prezentaci na WWW (GIF, JPEG),

Technické důvody
: mnoho formátů bere ohled na rozlišení skenerů, pomocí kterých jsou obrazy zaznamenávány, na obrazová rozlišení v různých osách, na odlišné architektury obrazové paměti v grafických kartách, či na interpretaci dvojice bytů v jednom 16bitovém slově – některé procesory ukládají do bytu s nižší adresou nižší řády 16bitového čísla (*little endian*), jiné je ukládají do bytu s vyšší adresou (*big endian*).

Metoda komprese
: vzhledem k velkému paměťovému objemu barevných obrazů je žádoucí uchovávat obraz v komprimované podobě. Volba vhodné kompresní metody je často závislá na charakteru obrazu a na jeho dalším použití. Základní dělení rozlišuje kompresní metody na *ztrátové* (*lossy*) a *bezeztrátové* (*lossless*).

Uvedený přehled nezaznamenává zcela všechny důvody existence široké škály obrazových formátů. Existuje celá řada dalších aplikačních požadavků, jakými je například uložení více obrázků v jednom souboru, zápis sekvence obrazů pro animaci, multimediální obrázky rozšířené

o informace potřebné pro interakci (oblasti zájmu), vícerozměrné snímky z medicínských aplikací (prostorové řezy MRI, CT), prokládání při přenosu po síti s cílem ukázat obrázek již po přenesení části dat (*interlacing*), uložení jednoho obrazu v několika stupních kvality (*multiresolution*), přidávání poznámek (anotací) a dalších informací týkajících se daného obrazu (*metadata*) apod.

Většina formátů je schopna uchovávat obrázky jak černobílé, tak barevné. Zvyšující se počet barev vede pochopitelně k větším paměťovým nárokům. Základní typy barevných obrazů ukládaných v souborech jsou uvedeny v následující tabulce:

Obrázek		bit/pixel	Poznámka
černobílý	B/W	1	délka řádku zaokrouhlována na celé byty, (*padding*)
v odstínech šedi	gray scale	8	někdy jen 6 bitů
s paletou	paletted	8	256 barev v paletě, někdy jen 128, 64, ...
plně barevný	color	24	RGB, CMY, YUV, YC_BC_R
		32	RGBA, CMYK
s vysokým dynamickým rozsahem	HDR	48	až 96 bitů na pixel

Mezi další varianty patří tzv. *high color* obrázky s přímým barevným kódováním barevných složek po 5 bitech (16bitové slovo na pixel) či paletované obrázky s kódováním CMYK. Velký objem dat a zároveň specifický tvar obrazových informací jsou podnětem pro používání různých druhů kompresí. Pokud zmenšíme objem dat tak, aby informace zůstala nezměněna, hovoříme o bezeztrátové kompresi. Bezeztrátová komprese nedosahuje takové úspory paměti jako komprese ztrátová, při níž se odstraňuje informace, která není příliš významná. Pokud například posloupnost 77876778778 nahradíme řadou ze samých sedmiček, ztratíme část informace, ale novou posloupnost budeme moci komprimovat podstatně lépe. V počítačové grafice je důležitá tzv. psychovizuální redundance. Označuje tu část informace, jejíž nepřítomnost nepostřehneme, a proto ji můžeme zanedbat.

V dalších částech popíšeme principy v grafice nejčastěji používaných kompresních metod. Detailní informace může čtenář nalézt např. v [Murr95]. Následující tabulka obsahuje jednoduchý přehled vlastností kompresních metod:

Kompresní metoda	Zkratka	Ztrátová	Příklad formátu
Run length encoding	RLE	ne	PCX (ZSoft PaintBrush)
Huffmanovo kódování	CCITT	ne	TIFF
Slovníkové kódování	LZW	ne	GIF, PNG, ZIP, ARJ
Diskrétní kosinová transformace	DCT	ano	JPEG

Poznamenejme, že u všech obrazových formátů se komprese týká pouze vlastních obrazových dat. Hlavička souboru, definice palety a další doplňující informace se nekomprimují.

2.6.1 Run length encoding

Jednoduchá a pro velkou třídu obrázků i efektivní metoda vychází z předpokladu, že v rastrovém obrázku, vzniklém jako kresba či skica, se opakují hodnoty sousedních pixelů. Do souboru tedy zapíšeme nejprve počet opakujících se totožných hodnot a poté hodnotu samotnou. Například při kódování pixelů definovaných jedním bytem rozlišíme příznak opakování hodnotou nejvyššího bitu:

1	čítač	hodnota	`hodnota` se opakuje `(1 + čítač)`×
0	hodnota		přímý zápis jediné 7-bitové neopakující se hodnoty
10000000		hodnota	zápis neopakující se hodnoty větší než binárně `10000000`

Efektivnější varianta RLE[1] je schopna zachytit posloupnost různých hodnot a opakovací bit využít ve významu příznaku zápisu posloupnosti:

0	čítač N	hodnota 0	hodnota 1	.....	hodnota N

Základní varianta kódovacího algoritmu RLE je uvedena v algoritmu 2.3. Efektivita kódování klesá při zpracování hodnot s nenulovým nejvyšším bitem. Pokud zapisujeme obrázky definované pomocí palety, je vhodné uspořádat paletu (a přečíslovat hodnoty pixelů) tak, aby méně často používané odstíny byly umístěny v horní polovině palety.

Metoda RLE je vhodná pro barevné rozlišení 1 či 8 bitů na pixel. Nepoužívá se pro kódování pixelů definovaných přímo hodnotami RGB, kdy není zajištěna fyzická sousednost bytů opakujících se pixelů. Výjimkou je kódování po jednotlivých barevných rovinách (*color planes*).

Ve většině případů je kódování RLE prováděno v rámci jednoho řádku. Všimněme si, že zatímco obrázek tvořený mnoha vodorovnými čárami je kódován velmi efektivně, tentýž obrázek otočený o 90° je zapsán téměř beze změny. Proto se v některých aplikacích objevuje použití speciálního kódu ve významu opakovače řádků. V případě, že kódovaný obrázek obsahuje neopakující se hodnoty v sousedních pixelech, dochází u RLE k *záporné kompresi* (*negative compression*), tj. ke zvětšení výsledného souboru. Kódování RLE je proto vhodné pro obrázky kreslené „od ruky“ nebo pro tzv. *cartoons* – ilustrace s většími stejnobarevnými plochami.

Vzácněji se můžeme setkat i se ztrátovou variantou metody RLE, při níž se nejprve testují sousední (barevné) pixely a pokud se liší jen málo, nahradí se hodnotou jedinou.

[1] Český překlad zní *kódování délky běhu*, kde slovo *běh* označuje řadu stejných hodnot. Název se však u nás neujal a používá se pouze zkratka RLE.

1. Vynuluj *Čítač*
2. Do proměnné $BarvaA$ přiřaď hodnotu prvního pixelu
3. Dokud jsou na vstupu pixely, opakuj:
   (a) Do proměnné $BarvaB$ přiřaď hodnotu dalšího pixelu
   (b) Pokud je $BarvaA = BarvaB$ a zároveň hodnota *Čítače* nepřesáhla maximální hranici (danou velikostí bytu či slova),
   pak zvyš *Čítač* o jedna,
   jinak
      i. Zapiš dvojici (*Čítač*, $BarvaA$)
      ii. Do proměnné $BarvaA$ přiřaď hodnotu $BarvaB$
      iii. Vynuluj *Čítač*
4. Zapiš dvojici (*Čítač*, $BarvaA$)

Algoritmus 2.3: Princip kódování RLE

2.6.2 Huffmanovo kódování

Toto kódování bylo původně navrženo komisí CCITT *(Comité Consultatif International Téléphonique et Télégraphique)* pro přenos černobílých dokumentů faxem. Myšlenka kódování pochází z r. 1952 od D. Huffmana a je založena na použití různě dlouhých bitových kódů pro symboly s různou frekvencí výskytu. Často používané symboly mají kratší kódy, přičemž frekvence se nestanovuje individuálně pro každý dokument, nýbrž je brána z tabulek CCITT, vzniklých statistickým zpracováním mnoha typických dokumentů. Postupně vzniklo několik variant kódování, lišících se stupněm komprese i určením:

označení	kompresní poměr	použití
G31D	5:1	odolné proti poruchám
G32D	8:1	citlivé na poruchy
G42D	15:1	(bezchybný) zápis na disk

G31D

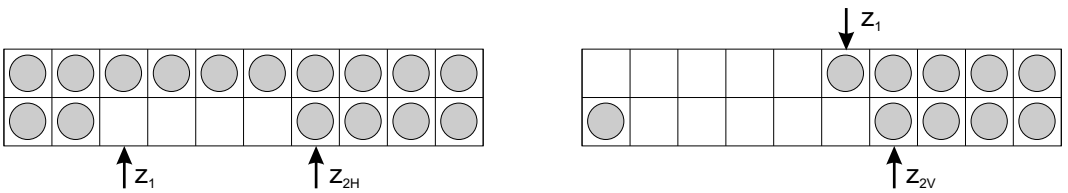
Úseky opakujících se bílých, resp. černých pixelů na jednom řádku jsou nejprve zakódovány metodou RLE, avšak opakovače jsou nahrazeny Huffmanovými kódy. Často se vyskytující

skupiny dvou, tří a čtyř bitů se shodnou barvou jsou tedy zapsány pomocí kratších výsledných kódů. Další krátké kódy, většinou pro bílé pixely, jsou určeny pro střední úseky (63 bitů) a velmi dlouhé úseky (až do 2 623 bitů). Svůj vlastní kód mají i konce řádků (EOL – *End Of Line*), takže výsledný obraz lze poměrně dobře rekonstruovat i při ztrátě dat při přenosu. K synchronizaci (tj. dekódování od začátku nového řádku) dojde vždy po nalezení kódu EOL. Dalším efektivním kódem je FILL – vyplnění všech pixelů stejnou barvou až do konce řádku. Protože při práci s faxem není předem znám počet přenášených řádků, je dalším kódem RTC (*Return To Control*) označen konec dat.

Zaměření na faxy je patrné nejen z odolnosti proti poruchám, ale i z toho, že dekódování lze provádět okamžitě bez použití vyrovnávací paměti.

G32D

Tato varianta využívá dvourozměrný charakter dat (viz 2D v názvu). Hlavní myšlenkou je zapisování pozic pixelů, ve kterých dochází ke změně barev. Změna barvy z se detekuje jak v rámci jednoho řádku, tak vzhledem ke změnám z_i na řádku předchozím (viz obr. 2.23).



Obrázek 2.23: Kódování černobílého obrazu metodou G32D

Zvláštní krátké kódy jsou určeny pro změny v rozmezí do 4 pixelů, ať již na témže řádku (kód z_{2H} na obrázku vlevo), nebo na předchozím řádku (kód z_{2V} na obrázku vpravo). Změna barvy z černé na bílou a naopak je zapisována relativně vůči předchozí pozicí, takže při ztrátě dat dochází k výrazné deformaci obrazu. Proto bývá většinou každý čtvrtý řádek kódován bezpečnou metodou G31D.

G42D

Při zápisu na disk se nepředpokládá ztráta dat při přenosu, takže není nutno do obrázku dodávat synchronizační informace. Metoda G42D je přímo odvozena z metody G32D odebráním nepotřebných kódů a je používána například ve formátu TIFF. Vyznačuje se velmi vysokým stupněm komprese, který je ovšem dán omezeným použitím na černobílé, typicky kancelářské, textové dokumenty.

2.6.3 Slovníkové kódování

Na rozdíl od předchozích kompresních schémat specializovaných na obrazová data, je tato metoda zcela obecná a setkáme se s ní ve většině běžných kompresních programů jako jsou ZIP či RAR. Původní algoritmus z r. 1977 známý pod názvem LZ77 (autoři A. Lempel a J. Ziv) byl v r. 1984 doplněn T. Welchem [Welc84] a je označován zkratkou LZW. Metoda je také nazývána *dictionary based encoding*.

Princip spočívá v nahrazení vzorků vstupních dat binárními kódy proměnné (postupně rostoucí) délky. Vstupní vzorky se překládají pomocí slovníku (tabulky), který je průběžně doplňován o nové vzorky. Délka slovníku je dána aktuálním počtem bitů použitých pro kódování. Po zaplnění slovníku se zvýší počet bitů určených pro výstupní kód o jedničku, takže délka slovníku se zdvojnásobí. Předpokládejme zpracování obrázku s barevným rozlišením jeden byte na pixel. Pak

- základní slovník je určen osmi bity (má délku 256),
- základní počet bitů pro kódování vzorků je devět,
- příznak vyčištění slovníku (*Clear code*) je binárně `100000000`.

Nově nalezené vzorky jsou zaznamenávány do slovníku pomocí kódů vyšších, než je *clear code*. Při kódování větších souborů se slovník může poměrně rychle zaplňovat a pro praktické účely bývá jeho délka omezena na 4 096 položek (tedy na max. 12bitové výstupní kódy). Vyslání dohodnutého symbolu *clear code* znamená vyčištění slovníku a jeho zkrácení do základní podoby.

Sympatickou vlastností je to, že slovník nemusí být zapisován do výstupních dat. Prvních 256 položek slovníku (tj. základní slovník) se nepřenáší, protože jsou totožné s hodnotami od 0 do 255, další položky pak lze odvodit z aktuálně zpracovávaných dat. Předpokládá se, že každý nový byte je počátečním symbolem nějakého vzorku. Máme-li (na počátku) na vstupu posloupnost bytů 5-3-5-3, je první číslo 5 bráno jako počátek nového vzorku 5-3. Na výstup je sice zapsán pouze kód čísla 5, ale slovník je automaticky rozšířen o vzorek 5-3. Obdobně je tomu při zpracování čísla 3. Opakující se dvojice 5-3 je pak nejen zapsána pomocí vlastního kódu (většího než *clear code*), ale spolu s následujícím číslem (či případným vzorkem) je uložena do slovníku.

Dekódování je prováděno zrcadlově vůči kódování. Z přicházejících kódů je dynamicky budován slovník a z něj vytvářen výsledný obraz. Slovník vlastně představuje zřetězené vzorky původních hodnot. S výjimkou nejnižších 256 kódů jsou všechny kódy reprezentovány jako dvojice ukazatelů na již nalezené vzorky s nižším kódovým označením.

Metoda LZW se vyskytuje v několika obměnách. Ze základních variant uveďme:

1. Namísto rušení celého slovníku lze symbol *clear code* považovat za příznak, po kterém

následují informace potřebné pro nahrazení málo používaných vzorků častěji používanými skupinami (například zrušení kódu pro dvojici 3-5 v předchozím příkladu).

2. Především při kódování obrázků je vhodné vstupní data nejprve vyjádřit pomocí rozdílů mezi sousedními pixely (*differencing*), čímž většinou nalezneme mnoho stejných vzorků. Příkladem je posloupnost pixelů 1-5-9-13-17, která se v přírůstkovém (rozdílovém) tvaru zjednoduší na 1-4-4-4-4.

2.6.4 Diskrétní kosinová transformace a JPEG

Metody RLE a LZW nejsou efektivní při kompresi plně barevných obrázků s mnoha barevnými přechody. Naneštěstí právě nejkvalitnější obrázky (ať již jde o reálné fotografie či počítačem generované obrazy) se vyznačují tím, že jen málokteré sousední pixely mají totožné hodnoty. Pro takové obrazy byla navržena metoda, při níž je kompresní poměr řízen požadavkem na výši kvality dekomprimovaného obrazu. V praxi se ukazuje, že snížení kvality na 75 % je pro většinu uživatelů nepozorovatelné, přitom kompresní poměr v takovém případě může být 20:1 až 25:1.

Metoda řízené ztrátové komprese využívající diskrétní kosinové transformace byla vyvinuta skupinou **JPEG**[2] (Joint Photographic Experts Group) v rámci mezinárodní standardizační organizace ISO v roce 1991 [ISO94a]. Je vhodná především pro kódování fotografií, u nichž sousední pixely (na řádku či ve sloupci) mají sice odlišné, ale přesto *blízké barvy*. Snižování kvality obrazu se projeví potlačováním rozdílů v blízkých barvách.

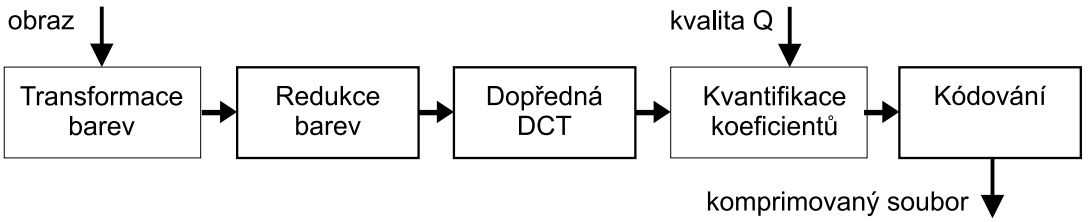
Metoda není vhodná pro obrazy s nižším barevným rozlišením (s paletou, upravené rozptylováním), také na velkých jednobarevných plochách vytváří artefakty v podobě čtverců či pruhů. Zcela nepoužitelná je pak pro černobílé obrázky, které rozmazává. Optimálním vstupem je obraz ve 24 bitech na pixel.

Diskrétní kosinová transformace je formou diskrétní Fourierovy transformace (viz kap. 2.2). Obrazová data jsou považována za vzorky spojitých funkcí naměřené v diskrétní síti pixelů. Výsledkem kosinové transformace je nalezení sady parametrů kosinových funkcí, jejichž složením lze rekonstruovat původní obraz. Při kódování obrazu ve formátu JPEG se používá *dopředná transformace* (FDCT), při dekódování *zpětná transformace* (IDCT).

Postup při kompresi JPEG je poměrně složitý a sestává z pěti kroků, jak ukazuje obrázek 2.24. Při dekompresi jsou tyto kroky prováděny analogicky v obráceném pořadí. Podívejme se na jednotlivé etapy blíže:

1. *Transformace barev*
   Ať již jsou vstupem barvy pixelů v prostoru RGB, CMY či CMYK, pro další zpracování je třeba je převést do barevného modelu YC_BC_R (viz část. 1.2.2) se třemi byty na pixel. V další fázi jsou totiž jasové složky Y a barevné složky C_B, C_R zpracovávány odděleně.

[2] Vyslov [džej-peg].



Obrázek 2.24: Posloupnost operací při kompresi JPEG

2. *Redukce barev*
Lidský vizuální systém je mnohem citlivější na rozdíly v úrovních jasu, než na změny barevných odstínů. V tomto kroku je proto možno snížit objem dat zprůměrováním barevných složek několika sousedních pixelů a nahrazením jedinou hodnotou. Používají se dvě varianty:

2h1v: průměrování sousedních dvojic C C redukce 6 → 4 byty

2h2v: průměrování čtveřic C C / C C redukce 12 → 6 bytů

Tento krok nezpůsobí zásadní zhoršení obrazu, neboť jeho kvalita je přímo ovlivňována až ve čtvrté fázi. Jasové složky Y všech pixelů zůstávají zachovány.

3. *Dopředná diskrétní kosinová transformace*
Obrazová data jsou rozdělena do čtverců 8×8. Každý čtverec je podroben diskrétní kosinové transformaci. Výsledkem je 8×8 koeficientů $F(u, v)$ získaných výpočtem z hodnot pixelů $f(x, y)$ podle vzorce:

$$F(u,v) = \frac{1}{4}\, C(u)\, C(v) \left[\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\, \cos\frac{(2x+1)u\pi}{16}\, \cos\frac{(2y+1)v\pi}{16}\right]$$

$$C(u),\ C(v) = \frac{1}{\sqrt{2}} \quad \text{pro} \quad u, v = 0, \quad \text{resp. 1 jinde.}$$

Zpětná (inverzní) transformace má tvar:

$$f(x,y) = \frac{1}{4} \left[\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)\, C(v)\, F(u,v)\, \cos\frac{(2x+1)u\pi}{16}\, \cos\frac{(2y+1)v\pi}{16}\right].$$

Hodnoty získané dopřednou transformací nejsou celočíselné a jsou zaokrouhleny až v další fázi zpracování. Na obrázku 2.25 můžeme sledovat postupné změny hodnot kódované oblasti 8×8 pixelů (posloupnost obrázků a–d) a jejich dekódování (e–f). Operace jsou prováděny s daty, která reprezentují jasovou složku Y.

139	144	149	153	155	155	155	155
144	151	153	156	159	156	156	156
150	155	160	163	158	156	156	156
159	161	162	160	160	159	159	159
159	160	161	162	162	155	155	155
161	161	161	161	160	157	157	157
162	162	161	163	162	157	157	157
162	162	161	161	163	158	158	158

a) vstupní data $f(x, y)$

1259.6	-1.0	-12.1	-5.2	2.1	-1.7	-2.7	1.3
-22.6	-17.5	-6.2	-3.2	-2.9	-0.1	0.4	-1.2
-10.9	-9.3	-1.6	1.5	0.2	-0.9	-0.6	-0.1
-7.1	-1.9	0.2	1.5	0.9	-0.1	-0.0	0.3
-0.6	-0.8	1.5	1.6	-0.1	-0.7	0.6	1.3
1.8	-0.2	1.6	-0.3	-0.8	1.5	1.0	-1.0
-1.3	-0.4	-0.3	-1.5	-0.5	1.7	1.1	-0.8
-2.6	1.6	-3.8	-1.8	1.9	1.2	-0.6	-0.4

b) koeficienty $F(u, v)$ po FDCT

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	61
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

c) kvantizační tabulka

79	0	-1	0	0	0	0	0
-2	-1	0	0	0	0	0	0
-1	-1	0	0	0	0	0	0
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

d) kvantované koeficienty

1264	0	-10	0	0	0	0	0
-24	-12	0	0	0	0	0	0
-14	-13	0	0	0	0	0	0
-14	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

e) expandované koeficienty před IDCT

142	144	147	150	152	153	154	154
149	150	153	155	156	157	156	156
157	158	159	161	161	160	159	158
162	162	163	163	162	160	158	157
162	162	162	162	161	158	156	155
160	161	161	161	160	158	156	154
160	160	161	162	161	160	158	157
160	161	163	164	164	163	161	160

f) rekonstruovaná data

Obrázek 2.25: Příklad zpracování dat při kompresi JPEG

4. *Kvantování koeficientů DCT*
   Právě v této fázi může uživatel stanovit ztrátu informace (a tedy i stupeň komprese),

v jaké má být obrázek komprimován. Rozsah koeficientů $F(u, v)$ je snižován vydělením určitým kvantizačním činitelem. Posléze jsou koeficienty zaokrouhleny.

Zvýrazněný koeficient $F(0, 0)$ v levém horním rohu čtverce na obr. 2.25b je nazýván *DC člen* a představuje stejnosměrnou složku harmonického rozkladu. Má největší celkový vliv na veškerá obrazová data ve čtverci. Ostatní koeficienty se nazývají *AC členy* a ovlivňují vyšší frekvence, tedy změny mezi jednotlivými pixely. Čím větší je jejich index (vzdálenost od DC členu), tím menší mají vliv. Velikost DC členu odpovídá osminásobku průměru všech hodnot ze vstupní matice.

Kvantizační hodnoty byly stanoveny komisí JPEG a jsou uloženy v různých tabulkách (obr. 2.25c) pro složky Y, C_B a C_R. Každý koeficient ve zpracovávaném čtverci je dělen jednou odpovídající hodnotou z kvantizační tabulky, velikost hodnot v tabulce se opět zvětšuje se vzdáleností od levého horního rohu.

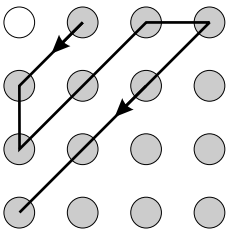
Uživatel stanovuje koeficient kvality Q v rozsahu od 1 do 100. Kvantizační tabulka byla definována pro průměrnou kvalitu obrazu (cca 75 %) a po zadání Q jsou její hodnoty přepočítávány nepřímo úměrně na Q. Takto upravenými hodnotami jsou dělena zpracovávaná data. Kvantizační tabulky jsou stejné pro celý obrázek a jsou ukládány do výstupního souboru kvůli rekonstrukci obrazu.

5. *Kódování*
   Výsledkem předchozích kroků jsou numerické úpravy čtvercové podoblasti vstupního obrazu, které zjednodušily tvar dat až do podoby řídké matice s dominantním levým horním rohem (obr. 2.25d). Tato matice je nyní kódována a zapisována do výstupního souboru, přičemž je využíváno toho, že většina koeficientů v okolí pravého dolního rohu má nulovou hodnotu. Matice čísel je převedena na posloupnost tak, jak ukazuje obr. 2.26, a tato posloupnost je zakódována Huffmanovým nebo aritmetickým kódováním. Členy DC se zapisují samostatně. Nejprve se sdruží DC členy z celého obrázku a poté jsou kódovány jejich rozdílové hodnoty (relativní změny vůči předchozí hodnotě).

Kódování JPEG se vyznačuje nejen vysokým stupněm komprese, ale i tím, že umožňuje rychlý *náhled* (*preview*)na zakódovaný obrázek. Využívá se přitom toho, že členy DC jsou sdružovány do skupin a ukládány samostatně. Chceme-li si co nejrychleji prohlédnout obrázek ve formátu JPEG, stačí v souboru nalézt sekce se všemi členy DC pro jasovou složku Y a pro zbylé dvě barevné složky a zobrazit je jako obrázek. Takto vytvořený náhled bude mít sice osmkrát menší rozměry oproti původnímu obrazu, ale dá nám jasnou představu o vzhledu.

JPEG 2000

Formát JPEG je velmi úspěšnou technologií, vždyť v současné době je používán ve většině digitálních fotoaparátů. Jeho nepříjemnou vlastností je však vnitřní dělení obrazu do čtverců pevné velikosti 8×8, které je překážkou na cestě k dosažení ještě vyššího kompresního poměru a které se současně na určitých částech obrazu projevuje rušivě jako viditelné makropixely. Toto omezení překonává formát *JPEG 2000* [ISO00], který při kódování bere do úvahy obraz jako celek a podrobuje jej transformacím za pomoci funkcí, nazývaných *vlnky* (*wavelet*). Opakované použití vlnkových transformací umožňuje uchovat obraz ve více rozlišeních, optimalizovat data pro postupný přenos po síti (*streaming*) a definovat oblasti zájmu (ROI) kódované ve vyšší kvalitě. Formát dovoluje použít ztrátovou i bezeztrátovou kompresi. Pokročilé vlastnosti formátu JPEG 2000 jsou pochopitelně spojeny se složitostí implementace příslušných kodérů a dekodérů.



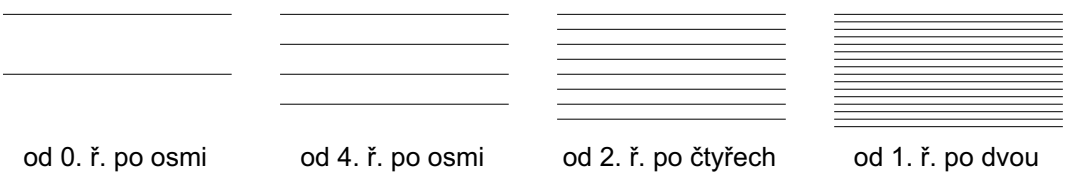
Obrázek 2.26: Kódování členů AC

2.7 Příklady rastrových formátů

[Murr95] uvádí, že existuje okolo 40 rozšířených formátů pro úschovu rastrových obrázků. Nemá jistě smysl je všechny uvádět, stejně tak není třeba detailně rozebírat vnitřní strukturu vybraných formátů. V této části proto popíšeme jen několik nejznámějších formátů a všimneme si jejich případných omezení.

2.7.1 Graphics Interchange Format (GIF)

Jeden z nejoblíbenějších a nejstarších formátů, specializovaný na obrázky s paletou a s jedním bytem na pixel, byl vyvinut firmou CompuServe.



Obrázek 2.27: Prokládání řádků ve čtyřech krocích

Použitá kompresní metoda LZW přináší pro většinu obrázků velké zmenšení objemu dat. Původní určení pro přenos obrázků po telefonních linkách se projevuje ve složitější struktuře formátu, je však vítané pro webové aplikace. Mezi základní charakteristiky a možnosti patří:

- *více obrázků* v jednom souboru, každý z nich může mít vlastní barevnou paletu,
- možnost *prokládání řádků* (*interlacing*) je vhodná pro přenos obrázků po síti. Uživatel je schopen již po získání 1/4 či 1/2 objemu obrazových dat rozpoznat vzhled obrázku (obr. 2.27) a přenos dat případně ukončit;

⋆ ukládání *textových informací*, a to buď jako součást zobrazovaných dat nebo v podobě komentáře čitelného při prohlížení souboru běžným editorem;

⋆ *řídicí prvky* pro interaktivní práci s obrazem, např. časové prodlevy při zobrazování posloupnosti obrázků uložených v souboru (tzv. *animovaný GIF*) nebo aktivní oblasti určené pro identifikaci (*pick*);

⋆ blíže neurčená *aplikační data*.

Základní verze **GIF87a** je stále používána. Nové prvky ve verzi **GIF89a** (označeny ⋆ v předchozím přehledu), které jsou zaměřeny na multimedia, jsou využívány především v prostředí WWW. Jedna z položek přiřazené palety může být označena jako zcela průhledná. Všechny pixely s touto barvou jsou pak při vykreslení obrazu nahrazeny barvou pozadí. Hlavní nevýhodou tohoto jinak velmi oblíbeného formátu je omezení na maximální počet 256 barev v jednom obrázku.

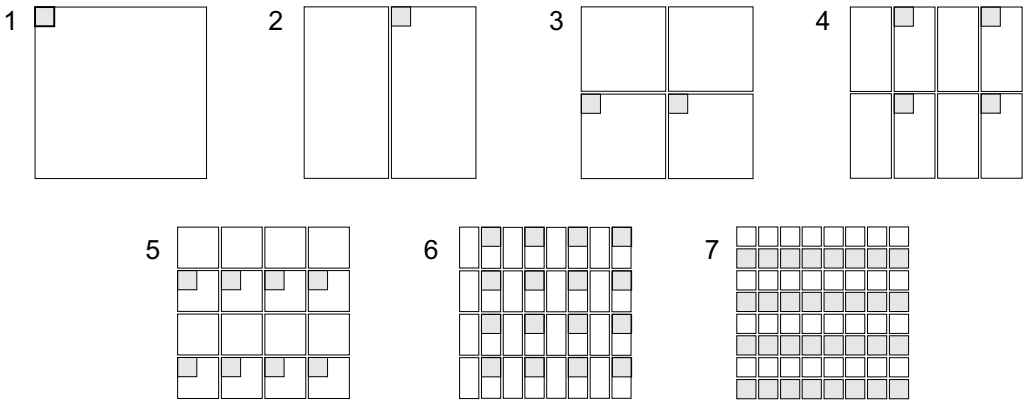
2.7.2 Portable Graphics Network (PNG)

Formát PNG[3] je normou ISO [ISO04] a je podporován konsorciem W3C starajícím se o vývoj webu. Je primárně zaměřen na přenos obrazu v síti, ale stejně tak dobře poslouží i pro archivaci dat. Formát je schopen ukládat obraz v mnoha barevných rozlišeních, kódování je bezeztrátové na bázi algoritmu LZ77.

Zásadní vlastností tohoto formátu je *předzpracování* každého pixelu. Je definováno celkem pět způsobů, jak zacházet s pixelem:

Typ 0:	**None**	–	pixel není vůbec upravován, jeho hodnota je přímo zakódována a uložena,
Typ 1:	**Sub**	–	je ukládán rozdíl s předchozím pixelem vlevo,
Typ 2:	**Up**	–	je ukládán rozdíl s předchozím pixelem nahoře (na předchozím řádku),
Typ 3:	**Average**	–	je ukládán průměr daného pixelu a jeho dvou sousedů (vlevo a nahoře),
Typ 4:	**Paeth**	–	je ukládána hodnota získaná z daného pixelu a jeho tří sousedů (vlevo, nahoře a vlevo nahoře) pomocí algoritmu, který vychází z metody navržené A. W. Paethem [Paet91].

[3] Vyslov [ping].



Obrázek 2.28: Dvourozměrné prokládací schéma ve formátu PNG

Žádná z metod předzpracování není ztrátová. Na každý řádek obrazu lze aplikovat jinou metodou a docílit tak v rámci daných postupů nejlepší možné komprese. Slovníková komprese LZ77 je použita na hodnoty pixelů až po předzpracování.

Dalším významným rysem je dvourozměrné prokládací schéma. Zatímco ve formátu GIF bylo schéma jednorozměrné (prokládání řádků), PNG dovoluje rozdělit přenášené informace do sedmi skupin. Dekódované pixely pak mohou vyplňovat čtvercové a obdélníkové oblasti, jejichž vzhled je postupně zjemňován (viz obr. 2.28). Již po přenesení jedné osminy celkového množství dat lze rozpoznat základní barevnou dispozici obrazu, další data obraz průběžně zpřesňují. Ve srovnání s formátem GIF může uživatel rozpoznat obsah obrazu po přenesení velmi malého množství dat. Dvourozměrné prokládací schéma je také vhodnější pro rozpoznání textu. Jedinou nevýhodou oproti formátu GIF je skutečnost, že formát PNG je určen pro uložení pouze jediného obrazu v jednom souboru, nedovoluje tudíž jednoduché animace obrázků.

Důležitou vlastností formátu PNG je schopnost bezeztrátově ukládat obrazy v barevném rozlišení *true color*. Dosažený kompresní poměr není tak výrazný jako u JPEG, je ovšem mnohem lepší oproti metodě RLE a překonává i slovníkové metody komprese. Barvy navíc mohou být uloženy ve vyšším rozsahu hodnot – až 16 bitů na barevnou složku.

PNG dokáže ukládat i obrazy v barevném modelu RGBA, tedy s informacemi o *průhlednosti*. U obrazů s paletou je údaj o průhlednosti přiřazen k jednotlivým položkám palety, u plnobarevných obrazů ke každému pixelu. Na rozdíl od formátu GIF je průhlednost ukládána ve stejném rozsahu jako mají barevné složky (maximum až 16 bitů), což umožňuje plnohodnotné míchání obrazů. Díky těmto vlastnostem je formát PNG s oblibou používán ve 3D grafice v podobě textury ovlivňující průhlednost objektu (viz část 13).

2.7.3 Targa (TGA)

Tento formát byl vyvinut firmou Truevision specializovanou na výrobu obrazových adaptérů. Používá kompresi RLE, mezi programátory je však oblíbena především jeho varianta bez komprese, umožňující po uložení jednoduché 18bytové hlavičky zapsat obrázek ve formátu 24 bitů/pixel do souboru přímo pixel po pixelu. Dokáže uložit obrázky v barevném rozlišení 1, 8, 16, 24 a 32 bitů/pixel. Poslední případ je určen pro barevný model RGBA. Původní verze 1 z r. 1987 je nejrozšířenější. Novější verze 2 zachovává původní strukturu formátu a pouze přidává na konec souboru blok aplikačních dat. Následující přehled obsahuje čtyři nejčastěji používané nekomprimované zápisy obrázků ve formátu Targa.

typ 1: 8 bitů/pixel + paleta

hodnoty v bytech	význam (dekadická hodnota)
`00 01 01`	základní typ hlavičky (0), paleta (1), typ obrazu (1)
`00-00`	počáteční index palety (zde 0)
`00-01 18`	délka palety (zde 256), počet bitů na položku palety (zde 24)
`xx-xx yy-yy`	počátek `x,y` umístění obrázku
`XX-XX YY-YY`	rozměry `X` a `Y` obrázku
`08 20`	počet bitů/pixel (8), kód umístění na obrazovce (vlevo nahoře)
...	položky palety v pořadí B-G-R
...	obrázek po řádcích (1 byte/pixel)

typ 2: 24 bitů/pixel

hodnoty v bytech	význam (dekadická hodnota)
`00 00 02`	základní typ hlavičky (0), bez palety (0), typ obrazu (2)
`00-00 00-00 00`	nulové údaje o paletě
`xx-xx yy-yy`	počátek `x,y` umístění obrázku
`XX-XX YY-YY`	rozměry `X` a `Y` obrázku
`18 20`	počet bitů/pixel (24), kód umístění na obrazovce (vlevo nahoře)
...	obrázek po řádcích a pixelech v pořadí B-G-R

typ 3: 256 odstínů šedi

hodnoty v bytech	význam (dekadická hodnota)
`00 00 03`	základní typ hlavičky (0), bez palety (0), typ obrazu (3)
`00-00 00-00 00`	nulové údaje o paletě
`xx-xx yy-yy`	počátek `x,y` umístění obrázku
`XX-XX YY-YY`	rozměry `X` a `Y` obrázku
`08 20`	počet bitů/pixel (8), kód umístění na obrazovce (vlevo nahoře)
...	obrázek po řádcích (1 byte/pixel)

typ 3: černobílý

hodnoty v bytech	význam (dekadická hodnota)
`00 00 03`	základní typ hlavičky (0), bez palety (0), typ obrazu (3)
`00-00 00-00 00`	nulové údaje o paletě
`xx-xx yy-yy`	počátek `x,y` umístění obrázku
`XX-XX YY-YY`	rozměry `X` a `Y` obrázku
`01 20`	počet bitů/pixel (1), kód umístění na obrazovce (vlevo nahoře)
...	obrázek po řádcích (1 bit/pixel)

2.7.4 Tag Image File Format (TIFF)

Tento formát je spolu s formáty JPEG a PNG mezinárodním standardem pro kódování statických obrazů [ISO96] a je tedy podporován všemi běžně používanými výpočetními platformami (UNIX, MS Windows, Macintosh), takže se objevuje ve variantách *little* i *big endian*. Podobně jako GIF dokáže uložit více obrázků do jednoho souboru. Prošel složitým historický vývojem a je schopen zapsat obrazy v nejširší škále barevných rozlišení a modelů:

verze	rok	komprese	poznámka
3.0	1986	CCITT	černobílé obrazy a odstíny šedi
4.0	1987	RLE	RGB
5.0	1988	LZW	palety
6.0	1992	JPEG	CMYK, YC_BC_R, standard ISO

Z hlediska vnitřní struktury je soubor TIFF rozdělen do tří tříd:

H	*Header*	hlavička souboru (jediná)
IFD	*Image File Directory*	popis významu dat
I	*Image data*	vlastní data specifikovaná pomocí IFD

První nepříjemností je několik variant zápisu uvedených tříd. Za hlavičkou mohou následovat třídy IFD a I buď pravidelně se střídající, nebo nejprve zcela všechny IFD následované všemi I, případně opačně.

Speciálním rysem formátu je právě třída IFD, která je tabulkou proměnlivé délky s položkami o délce 12 bytů. Každá taková položka se nazývá *tag* (česky např. *visačka*) a obsahuje kromě svého identifikátoru i specifikaci datového typu (byte, short, long, apod.), počet příslušných dat a adresu počátku dat v odpovídající sekci I (Image data).

Existuje více než 70 typů visaček, aplikačním programům je dokonce povoleno definovat svoje vlastní. Ačkoliv mohou být visačky v sekci IFD libovolně prostřídány, formát TIFF definuje čtyři druhy povinných skupin, u kterých je nutno dodržet pořadí. Podle barevného rozlišení je třeba vždy začít zapisovat do souboru visačky z patřičné povinné skupiny B (černobílé), G (odstíny šedi), P (s paletou) nebo R (plně barevné).

Dalším stupněm volnosti u formátu TIFF je rozdělení vlastních obrazových dat. Kromě běžného uložení obrazu vcelku je sympatická možnost ukládat obraz po *pruzích* (*strips*), které jsou menší než 64 kB. Kromě zadání šířky pruhů lze definovat i jejich prokládání. Ve verzi TIFF 6.0 je podporováno uspořádání obrazových dat do *dlaždic* (*tiles*), které například umožní aplikačnímu programu zpracovat jen část velkého obrazu (plakátu), na běžné obrazovce jinak naprosto nezobrazitelného. Z tohoto důvodu je formát TIFF hojně používán při přípravě profesionálních barevných tisků.

2.7.5 Formáty pro animované sekvence

O formátech pro animované sekvence se zmíníme jen krátce, abychom ukázali na rozdíly mezi kódováním statických obrazů (*still images*) a animovaných sekvencí či přímo videozáznamu (*motion pictures, movie*). Jako ukázku jsme vybrali standardizovaný formát MPEG pro zápis video i audio informací.

Do jisté míry by přitom mohl být za formát pro animace považován i GIF, který je v prostředí WWW používán pro jednoduché „rozhýbání“ statických obrazů. Paměťová efektivita je ovšem velmi nízká, neboť jednotlivé obrazy jsou v GIFu ukládány samostatně, bez ohledu na to, jak hodně či málo se od sebe navzájem liší.

Obdobně jako JPEG, také formát MPEG[4] je výsledkem oficiální standardizační aktivity, tentokrát skupiny Motion Picture Experts Group. Formát je postupně vylepšován a obohacován o nové schopnosti. Výhradně pro uchování a přenos obrazu se zvukem jsou určeny verze MPEG-1 [ISO93] a MPEG-2 [ISO94b] . Pro interaktivní televizi a jiné 2D a 3D aplikace je určen formát MPEG-4 [ISO01a], pro archivaci multimediálních souborů a vyhledávání informací v nich je navržen formát MPEG-7 [ISO02]. Nejnovější verze, pokrývající vlastnosti všech předchozích, nese název MPEG-21 (pro 21. století) [ISO01b].

Zaměříme se na první dvě verze. Obě používají metody ztrátové komprese pro obraz a zvuk. MPEG-1 je určen pro úschovu a přenos obrazu a zvuku ve vysoké kvalitě, tj. až do rozměru $4\,095 \times 4\,095$ pixelů, 30 snímků za sekundu při přenosových rychlostech do 1.5 Mb/s. Obraz je kódován v barevném modelu YUV. Verze MPEG-2 je schopna při přenosových rychlostech do 15 Mb/s zvládnout přenos televizního obrazu v diskrétní podobě a v různých rozlišeních, např. pro HDTV.

Pomineme otázky kódování zvukových informací a všimneme si záznamu obrazu. Formát MPEG používá diskrétní kosinovou transformaci obdobně jako JPEG, avšak vzhledem k charakteru zpracovávaných dat dosahuje běžně mnohem vyššího kompresního poměru, typicky 20–40:1 (teoreticky až 200:1). Nepleťme si přitom MPEG s formátem MJPEG (*Motion JPEG*), který je pouhým záznamem posloupnosti samostatně zpracovaných JPEG obrázků.

[4] Vyslov [em-peg].

Snímky jsou při zpracování rozděleny do tří skupin:

I-frame (intraframe)	–	běžný snímek
P-frame (predictive)	–	rozdílový obrázek mezi běžným snímkem a předchozím I nebo P-snímkem
B-frame (bi-directional)	–	rozdílový obrázek mezi dvěma nejbližšími I či P-snímky

Každý druh snímku je samostatně komprimován obdobnou metodou jako JPEG. Nejmenší komprese se dosahuje u I-snímků, nejlepší u B-snímků. Typicky jsou objemy snímků I:P:B po kompresi v poměru 15:5:2. Ačkoliv by bylo výhodné vytvořit co nejvíce B-snímků, z praktických důvodů jsou snímky kódovány v přesně definovaných posloupnostech s periodou 12 snímků

I B B P B B P B B P B B I

tak, aby se každé 0.4 sekundy objevil jeden I-snímek. To umožňuje přehrávat film v původní rychlosti i na pomalém dekódovacím zařízení. Jakmile není možno včas připravit obrazová data pro obrazovku, přeskočí se dekódování (a zobrazení) B-snímků a poté případně i P-snímků. V nejhorším případě dojde i k vynechání I-snímků. Rychlost prezentovaného děje (animace) je tak při použití formátu vždy zachována za cenu přeskakování obrázků. Existence pravidelně uložených I-snímků v souboru navíc dovoluje rychlé převíjení či skok dopředu.

Vzhledem k tomu, že B-snímky jsou vztaženy k okolním snímkům, není při přenosu či zápisu dat do souboru dodrženo výše uvedené pořadí snímků. První část posloupnosti

$$I_1\ B_2\ B_3\ P_4\ B_5\ B_6\ P_7\ B_8\ B_9\ P_{10}\ B_{11}\ B_{12}\ I_{13}$$

je souboru zapsána v pořadí

$$I_1\ P_4\ B_2\ B_3\ P_7\ B_5\ ...,$$

protože pro zobrazení mezilehlých snímků B_2 a B_3 je potřeba znát snímek P_4. Při dekódování je tedy nutno mít dostatek paměti na několik snímků, typicky na posloupnost $I_1\ B_2\ B_3\ P_4$, která je již vhodná pro zobrazování. Nejprve je dekódován a zobrazen snímek I_1. Z něj je za pomoci P_4 a B_2 vytvořen snímek I_2 (nahrazující B_2). Totéž se opakuje pro B_3. Čtvrtý obrázek vznikne přičtením rozdílového snímku P_4 k aktuálnímu snímku I_3.

Ani kódování rozdílů mezi snímky nevede k nejvyšší možné kompresi. Při snímání pohyblivých dějů statickou kamerou či při snímání scény pohybující se kamerou jsou zjevně dva po sobě následující snímky posunuty a jejich přímým odečtením příliš nezískáme. Proto se v souboru uchovávají údaje o pohybu (*motion vectors*), které se aplikují na celý snímek či na jeho vybranou část před tím, než dojde k odečtení s předchozím snímkem.

Metody kódování ve verzích MPEG-1 a MPEG-2 byly pevně dány normou, což vedlo ke kompatibilitě při přehrávání animací a filmů. Neumožňovalo to však aplikovat nové či speciální kódovací (kompresní) algoritmy. Proto je ve verzi MPEG-4 dovoleno použití libovolných

kódovacích schémat implementovaných v podobě tzv. *kodeků*. Programátoři mohou při dodržení normalizovaného rámce MPEG-4 kódovat pohyblivé obrazy různými způsoby. To vedlo k dosažení vyšších stupňů komprese (např. na bázi vlnkových transformací), ale také ke snížení kompatibility – kodeky nejsou součástí datového souboru, ale jsou to samostatné funkce, které je třeba do počítače instalovat podobně jako programy.

Možnosti formátů řady MPEG jsou velmi široké a jejich popis přesahuje rámec této knihy. Proto jen na závěr krátce uveďme, že verze MPEG-4 představuje zásadní zlom v kódování obrazových a obecně multimediálních dat. Obraz v tomto formátu totiž není chápán jen jako pole pixelů, ale je na něj nahlíženo jako na výsledek skládání různorodých informací z mnoha datových toků. Rozmanitost dat je skutečně široká – zahrnuje statické i animované obrazy, statické i pohyblivé texty (např. titulky), definici interaktivních (citlivých) oblastí, popis samostatných trojrozměrných objektů i celé scény včetně virtuálních kamer (dle norem VRML a X3D), popis lidského obličeje a animaci jeho výrazu (*talking-head*), popis hlasových elementů a zvuků (*phoneme*) pro potřeby automatického převodu textu na mluvenou řeč, definici způsobu míchání zvukových stop ve virtuálním zvukovém studiu a mnoho dalších. Výsledný obraz a zvuk vzniká kompozicí těchto prvků, což obecně klade velmi vysoké nároky na systém, který má data ve formátu MPEG-4 interpretovat. Proto se v současné době setkáváme většinou jen se specializovanými přehrávači dat MPEG-4, které jsou zaměřeny na zpracování vybrané podmnožiny výše uvedených datových toků, např. film či animaci lidské tváře.

Kapitola 3

Dvourozměrné objekty

Za základní dvourozměrné objekty považujeme úsečky, lomené čáry, kružnice, elipsy, mnohoúhelníky, křivky a textové řetězce. Těmto objektům říkáme *základní grafické prvky* (*output primitives*) a jsou obsaženy ve většině programů pro kreslení v rovině. Základní prvky mohou mít liniový charakter (úsečky, křivky) nebo plošný charakter. Ve druhém případě se u nich rozlišuje obrys a vnitřek, který lze různými způsoby vyplňovat.

Podle typu zobrazovacího zařízení jsou výsledkem algoritmů pro kresbu grafických prvků buď posloupnosti bodů (pixelů), nebo posloupnosti úseček. V prvním případě tak získáme *rastrový obraz*, jehož reprezentaci jsme se věnovali v předchozí kapitole. Druhý typ algoritmů vytváří obraz *vektorový*. Ten může být buď vykreslován na vektorových kreslicích zařízeních nebo převeden do rastrové podoby.

Současná počítačová grafika je orientována především na tvorbu rastrového obrazu. Při kresbě v rastru je třeba nalézt všechny pixely, reprezentující tvar a polohu grafického prvku, a přiřadit jim barvu daného prvku. Určování souřadnic a barvy těchto pixelů se nazývá *rasterizace*. Je zřejmé, že rasterizace není nic jiného než výsledek vzorkování daného grafického prvku v místech, která tento prvek částečně nebo úplně překrývá.

V této kapitole uvedeme postupy používané při rasterizaci úsečky, lomené čáry, kružnice a elipsy. V závěru kapitoly se zaměříme na vyplňování oblastí a ořezávání základních grafických prvků. Popis, vlastnosti a kresbu různých druhů křivek jsme zařadili do samostatné kapitoly 5, v níž jsou společně uvedeny křivky v rovině i v trojrozměrném prostoru.

3.1 Úsečka a lomená čára

Přestože úsečka patří mezi nejjednodušší grafické prvky, její kresbě je věnována pozornost v téměř každé publikaci o počítačové grafice. Z úseček lze skládat *lomené čáry* (*polyline*), jimiž

se často nahrazují složitější křivočaré obrazce. Proto by vykreslení úsečky mělo být prováděno co nejefektivněji. Z geometrického hlediska je lomená čára posloupností úseček, v níž koncový bod jedné úsečky je počátečním bodem úsečky následující. Úsporně tedy lomenou čáru popíšeme buď posloupností bodů nebo počátečním bodem a posloupností relativních přírůstků.

Samotná úsečka bývá nejčastěji zadávána pomocí svých koncových bodů $[x_1, y_1]$ a $[x_2, y_2]$. Pro výpočty je výhodné, jsou-li tyto body uspořádány tak, že první koncový bod leží vlevo od druhého bodu, v případě shodných souřadnic x pak nad druhým bodem. Je možno také definovat úsečku počátečním bodem $[x_1, y_1]$ a vektorem rozdílů souřadnic $(\Delta x, \Delta y) = (x_2 - x_1, y_2 - y_1)$. Obecně jde o neceločíselné hodnoty, které jsou před vykreslením v rastru zaokrouhleny.

Popis úsečky i dalších liniových grafických prvků bývá doplněn *atributy*, které charakterizují její vzhled. Jde především o *sílu* (tloušťku) a způsob kresby *přerušované čáry*. Tloušťka rovna jedné označuje nejtenčí možnou čáru, některé systémy ji však značí jako tloušťku nula.

Pro popis přerušované čáry se zavádí pojem *úsek* – plný a prázdný. Několik úseků definuje *vzor* (*pattern*) přerušované čáry, který je opakovaně vykreslován od počátečního ke koncovému bodu úsečky. Vzor je tvořen sudým počtem úseků. Většinou začíná plným, tedy kresleným úsekem a končí prázdným úsekem. Pro obyčejnou přerušovanou čáru vystačíme se vzorem tvořeným dvěma úseky, pro čerchovanou čáru potřebujeme vzor se čtyřmi úseky.

Kresbu lomené čáry lze snadno převést na kresbu elementárních úseček. Pokud je však lomená čára přerušovaná nebo má větší tloušťku, je nutno věnovat pozornost napojování jednotlivých úseček v koncových bodech. Lomené čáře s různými atributy se také říká *dráha* (*path*). Pod tímto názvem se můžeme setkat i s kombinovanou čárou, která je tvořena navazujícími úsečkami a křivkami.

3.1.1 Rasterizace úsečky

Úsečka je vzorkována s konstantním krokem vždy v jedné z os x a y, a to podle svého sklonu, vyjádřeného *směrnicí* m (viz obr. 3.1). Tato charakteristická hodnota je dána podílem rozdílů souřadnic koncových bodů úsečky:

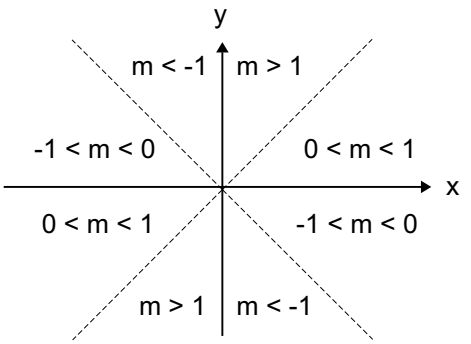
$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}. \tag{3.1}$$

Pokud je $|m| < 1$, úsečka se přimyká k ose x a bude proto vzorkována na ose x s krokem jeden pixel. Úsečky, jejichž absolutní hodnota směrnice je větší než jedna, jsou vzorkovány na ose y. Osa, v jejímž směru se vzorkuje, se nazývá *řídicí osa*, zbylé ose se říká *vedlejší osa*. Diagonální úsečky ($|m| = 1$) mohou mít řídicí osu libovolnou.

Při tomto způsobu rasterizace je úsečka převedena na takovou posloupnost pixelů, která se nazývá *osmispojitá* (8spojitá, *8-connected*). Pro libovolný pixel v této posloupnosti platí, že

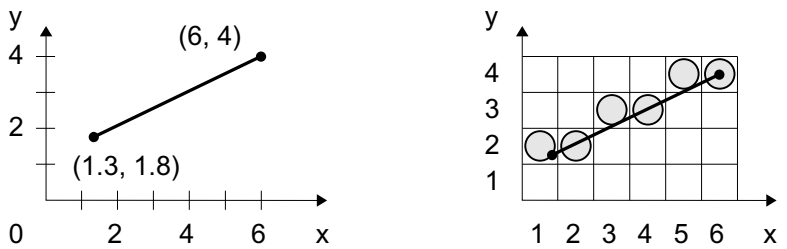
jeho nejbližší sousedé mohou ležet až v osmi směrech – vodorovně, svisle a diagonálně. Znalost této vlastnosti je důležitá pro algoritmy vyplňování oblastí (část 3.3).

Další možností v rastru je *čtyřspojitá kresba* (4spojitá, *4-connected*), při níž se nejbližší sousedé mohou nacházet pouze ve vodorovném či svislém směru. V tomto případě je diagonální krok při rasterizaci nahrazen dvojicí horizontálního a vertikálního kroku. Čtyřspojitá kresba úsečky je používána zřídka.



Obrázek 3.1: Hodnoty směrnice m pro různé sklony úsečky

Úsečka je obecně zadána neceločíselnými souřadnicemi; také směrnice bývá necelé číslo. Většina algoritmů pro kresbu úsečky ovšem předpokládá celočíselné vstupní souřadnice. Chyba, které se dopustíme zaokrouhlením souřadnic koncových bodů není považována za významnou (obr. 3.2), pokud není porušena návaznost tahu u lomené (kombinované) čáry. Se směrnicí je však nutno pracovat přesněji.



Obrázek 3.2: Spojité a rastrové zobrazení úsečky

Algoritmus DDA

Jeden z prvních algoritmů používaných v počítačové grafice (popsán např. v [Fole90]) je založen na postupném přičítání konstantních přírůstků k oběma souřadnicím, počínaje počátečním bodem úsečky. Obecně mohou mít přírůstky libovolnou velikost, vzhledem k vzorkování v rastru bývá přírůstek na řídicí ose roven jednomu pixelu.

Algoritmus DDA (*Digital Differential Analyzer*) je tvořen jediným cyklem, ve kterém je postupně k souřadnici na řídicí ose přičítána jednička, ke zbylé souřadnici neceločíselný přírůstek. Ten je roven směrnici m v případě řídicí osy x, resp. $1/m$ v případě řídicí osy y. Pro každý kreslený pixel je nutno souřadnici na vedlejší ose před vykreslením zaokrouhlit. Algoritmus 3.1

představuje zkrácenou verzi pouze pro řídicí osu x a předpokládá uspořádání koncových bodů úsečky zleva doprava.

1. Z koncových bodů $[x_1, y_1]$ a $[x_2, y_2]$ urči směrnici m podle vzorce (3.1)
2. Inicializuj bod $[x, y]$ hodnotou $[x_1, y_1]$
3. Dokud je $x \leq x_2$, opakuj:
   (a) Vykresli bod [x, zaokrouhlené(y)]
   (b) $x = x + 1$
   (c) $y = y + m$

Algoritmus 3.1: Algoritmus DDA pro řídicí osu x

Důležité je, že se v algoritmu DDA při výpočtu nových souřadnic pixelu využívá znalosti předchozího vzorku úsečky. Z parametrického popisu úsečky

$$\begin{aligned} x &= x_1 + t\ \Delta x \\ y &= y_1 + t\ \Delta y, \end{aligned}$$

lze pro řídicí osu x odvodit iterační zápis

$$\begin{aligned} x_{k+1} &= x_k + 1 \\ y_{k+1} &= y_k + m, \end{aligned}$$

který se provede pro Δx kroků. Algoritmus DDA je příkladem iteračního (opakovaného) výpočtu, při němž jsou nové hodnoty získávány z dosud vypočítaných. Tento způsob je v počítačové grafice často využíván. Efektivní implementace algoritmů vychází z geometrie a algebry, ale vyhýbá se přímému použití vzorců, které jsou sice přehledné, avšak výpočetně náročné. Dokladem je i následující metoda.

Bresenhamův algoritmus pro kresbu úsečky

J. E. Bresenham vyvinul velmi efektivní algoritmus [Bres65], který při rasterizaci nachází body ležící nejblíže skutečné úsečce pouze pomocí celočíselné aritmetiky. Postup vysvětlíme na příkladu podle obrázku 3.3, kde je nakreslena část rastru, ve kterém mají být zobrazeny pixely úsečky s *řídicí osou* x. Po nakreslení levého koncového bodu úsečky je třeba zvolit, zda další bod vpravo má být vykreslen na pozici se stejnou souřadnicí y nebo se souřadnicí

o jedničku větší. Pixel, který vybereme, je ten, jehož souřadnice y se více blíží skutečné hodnotě y na dané úsečce.

Předpokládejme, že již byl nakreslen pixel o souřadnicích $[x_i, y_i]$. Dva další možné pixely leží na pozicích $[x_i + 1, y_i]$ a $[x_i + 1, y_i + 1]$. Rozdíly mezi souřadnicí y středů uvedených pixelů a skutečnou souřadnicí y na úsečce v bodě $x_i + 1$ jsou na obrázku 3.3 označeny jako d_1 a d_2. Vyjdeme-li z obecné rovnice přímky

$$y = m\ x + b,$$

kde m je směrnice a b posun na ose y, získáme souřadnici y dosazením jako

$$y = m(x_i + 1) + b.$$

Pak

$$\begin{aligned} d_1 &= y - y_i &&= m(x_i + 1) + b - y_i, \\ d_2 &= y_i + 1 - y &&= y_i + 1 - m(x_i + 1) - b. \end{aligned}$$

Rozdíl těchto dvou vzdáleností vyjádříme jako

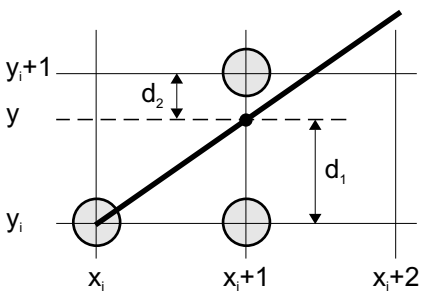
$$\Delta d = d_1 - d_2 = 2m(x_i + 1) - 2y_i + 2b - 1. \tag{3.2}$$

Podle proměnné Δd můžeme snadno určit, který ze dvou možných kandidátů leží blíže skutečné úsečce. Záporná hodnota Δd znamená, že blíže leží pixel o souřadnici y_i, kladná hodnota určuje jako bližší pixel o souřadnici $y_i + 1$. Pro stanovení vykreslovaných pixelů není tedy důležitá skutečná hodnota proměnné Δd, nýbrž její znaménko. Tento fakt je výhodný pro převod výpočtu Δd do celočíselné aritmetiky.

Jedinou neceločíselnou proměnnou ve vztahu (3.2) je směrnice m. Ta je přitom zadána jako podíl dvou celých čísel Δy a Δx. Jsou-li koncové body úsečky uspořádány zleva doprava, můžeme celou rovnici vynásobit kladným číslem Δx a po úpravě získáme nový vztah

$$p_i = \Delta d \Delta x = 2\Delta y\ x_i - 2\Delta x\ y_i + 2\Delta y + \Delta x(2b - 1), \tag{3.3}$$

v němž $2\Delta y + \Delta x(2b - 1)$ je konstanta, která bude při dalších úpravách vyloučena. Rovnici (3.3) lze ještě dále zjednodušit. Hodnotu p_i na levé straně, podle jejíhož znaménka určujeme souřadnice následujících pixelů, budeme nazývat *rozhodovacím členem* (*decision*). Zapíšeme-li rozhodovací člen p_{i+1} následující po p_i jako



Obrázek 3.3: Výběr pixelů úsečky

$$p_{i+1} = 2\Delta y\ x_{i+1} - 2\Delta x\ y_{i+1} + konst., \tag{3.4}$$

1. Z koncových bodů $[x_1, y_1]$ a $[x_2, y_2]$ urči konstanty
   $k1 = 2\Delta y$
   $k2 = 2(\Delta y - \Delta x)$
2. Inicializuj rozhodovací člen p na hodnotu $2\Delta y - \Delta x$
3. Inicializuj $[x, y]$ jako $[x_1, y_1]$
4. Vykresli bod $[x, y]$
5. Dokud je $x \leq x_2$, opakuj:
   (a) $x = x + 1$
   (b) Je-li p kladné, pak $y = y + 1$ a $p = p + k2$
   (c) Není-li p kladné, pak $p = p + k1$
   (d) Vykresli bod $[x, y]$

Algoritmus 3.2: Bresenhamův algoritmus pro řídicí osu x

můžeme odečtením rovnic (3.3) a (3.4) vyjádřit p_{i+1} pomocí p_i:

$$p_{i+1} = p_i + 2\Delta y - 2\Delta x(y_{i+1} - y_i), \tag{3.5}$$

když $x_{i+1} = x_i + 1$.

Pomocí této rovnice budeme iteračním způsobem počítat hodnoty každé další proměnné p z její předcházející hodnoty. Znaménko bude určovat způsob její aktualizace:

$p_i \leq 0$	$p_{i+1} = p_i + 2\Delta y$
$p_i > 0$	$p_{i+1} = p_i + 2\Delta y - 2\Delta x$

První hodnota $p_1 = 2\Delta y - \Delta x$ se získá výpočtem z rovnice (3.3), kde použijeme $[x_1, y_1]$ jako souřadnice počátečního bodu.

Hodnota rozhodovacího členu p je základním kritériem pro výběr pixelů tvořících obraz úsečky v rastru. Hodnotu p_i aktualizujeme pro každý pixel pouze sčítáním. Pokud je její hodnota záporná, souřadnice y následujícího pixelu se nemění. Při kladném znaménku p_i má následující pixel souřadnici y o jedničku větší.

Připomeňme, že uvedený postup (dokumentovaný algoritmem 3.2) rasterizuje pouze úsečky, jejichž směrnice je kladná a menší než jedna. Při zpracování libovolných úseček je proto třeba v iniciální fázi algoritmu nejprve rozhodnout, která osa je řídicí a podle toho zaslat úsečku

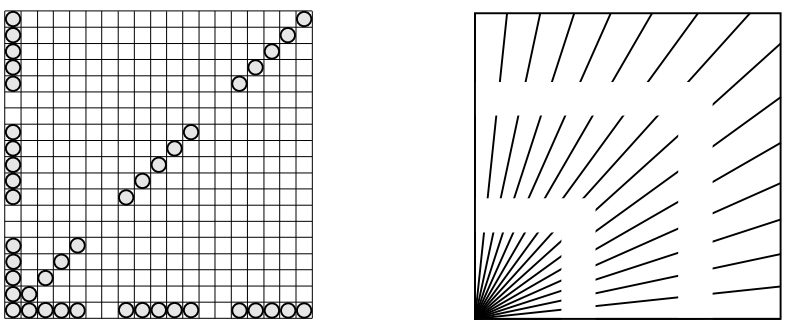
do jedné ze dvou samostatných, analogicky vytvořených částí algoritmu. Vhodnou inicializací konstant docílíme vykreslování v příslušném směrů (zleva doprava, zprava doleva apod.).

Bresenhamův algoritmus je příkladem iteračního postupu, v němž se na základě pomocné proměnné (rozhodovacího členu) rozhoduje o podmíněném provedení určité varianty. Algoritmus je velmi jednoduchý, v těle cyklu se používá pouze sčítání a test znaménka.

Myšlenka kreslit naráz úseky o délce několika pixelů byla základem pro vznik dalších, speciálních algoritmů. V nich se úsečka chápe jako posloupnost vodorovných, resp. svislých úseků. Výpočet délky úseku nebo vyhodnocení kritéria přechodu na nový úsek však vyžaduje oproti Bresenhamovu algoritmu několik operací navíc, takže jsou tyto metody efektivní jen pro téměř svislé či téměř vodorovné úsečky. Teoreticky zajímavé, ale v praxi nepoužívané, jsou pak vícekrokové metody, při nichž se ve vnitřním cyklu algoritmu určuje vzorek, tj. skupina několika po sobě následujících pixelů. Zatímco Bresenhamův algoritmus má na výběr pouze dvě možnosti polohy dalšího pixelu, u dvoukrokové varianty s řídicí osou x je možno umístit některý ze čtyř vzorků $E-E$, $E-NE$, $NE-E$ a $NE-NE$, kde písmena označují směry doprava (*East*) a šikmo doprava nahoru (*North-East*).

3.1.2 Kresba přerušované čáry

Algoritmy pro kresbu přerušovaných čar (*dashed line*) pracují dvěma způsoby. Při prvním, jednodušším způsobu, jsou postupně vypočítávány souřadnice všech pixelů úsečky například Bresenhamovým algoritmem a jejich kresba je povolena či zakázána podle toho, zda pixel patří do plného či prázdného úseku. Informace o aktuálním úseku jsou uloženy do dvou pomocných proměnných. Jedna obsahuje index aktuálního úseku, druhá počet zbývajících pixelů v tomto úseku. Druhá proměnná je zmenšena o jedna při každém určení nové souřadnice pixelu. Jakmile klesne její hodnota na nulu, přesune se ukazatel v první proměnné na další úsek a do druhé proměnné se uloží délka úseku.



Obrázek 3.4: Přerušované úsečky kreslené jednoduchým způsobem v celočíselné aritmetice

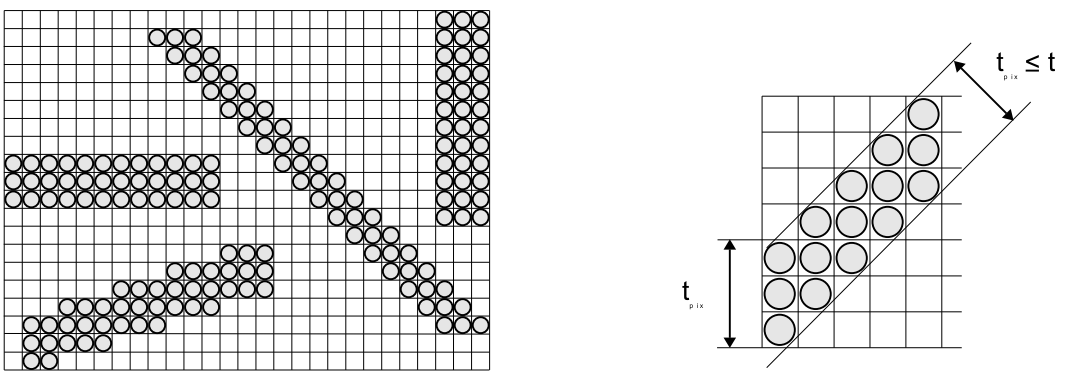
Tento způsob generování přerušovaných čar je rychlý, protože pracuje v celočíselné aritmetice, ale vzhled výsledné přerušované úsečky je ovlivňován jejím sklonem. Na obrázku 3.4 vlevo vidíme tři úsečky vycházející ze stejného bodu a vykreslené stejnou čárkovanou čárou. Diagonální úsečka má kreslené úseky zjevně delší oproti vodorovné a svislé úsečce. Tento nepříjemný jev je způsoben vlastnostmi rastru, resp. jeho *metrikou*. Chápeme-li v rastru délku úsečky jako počet jejích pixelů, snadno dojdeme k závěru, že např. diagonála čtverce má stejnou délku jako jeho strana. Pokud vliv sklonu vhodně nekorigujeme přepočtem délek úseků přerušované čáry, můžeme při kresbě paprsků na obrázku 3.4 vpravo získat dojem, že vidíme bílé, zalomené pásy.

V praxi se proto většinou používá přesnější postup, založený na výpočtu délek. Pomocí geometrických výpočtů se určí koncové body kreslených úseků a takto získané elementární úsečky se teprve pak vykreslí běžným algoritmem.

Při kresbě přerušovaných čar je žádoucí, aby v obou koncových bodech úsečky byly zobrazeny plné úseky, jinak úsečka vypadá jako nedokreslená, což je výrazné zejména u lomených čar. Jednoduché řešení spočívá v tom, že se poslední úsek na úsečce vždy nakreslí plnou čárou bez ohledu na skutečnou definici vzoru. Jiné algoritmy dokáží kreslit přerušované čáry pomocí tzv. plovoucího počátku, při kterém nejsou jednotlivé čárky přenášeny na obrazovku od prvního úseku, ale tak, aby byla vždy nakreslena alespoň část plného úseku u koncových bodů úsečky. To lze docílit při „rozumně“ zadaném vzoru, v němž nepřevládají prázdné úseky nad plnými.

3.1.3 Kresba silné čáry

Podobně jako u přerušované čáry rozlišujeme dva typy algoritmů pro kresbu silné čáry (*thick line*). Jednoduchý postup je rychlý, ale vykazuje nepřesnosti. Přesný postup pak vyžaduje náročnější výpočty. Do první třídy metod patří upravený Bresenhamův algoritmus, při kterém se namísto jednoho pixelu v každém kroku kreslí několik pixelů nad sebou či vedle sebe.



Obrázek 3.5: Silné čáry kreslené upraveným Bresenhamovým algoritmem

Výsledek použití této metody je ukázán na obrázku 3.5. Všimneme si dvou negativních jevů:

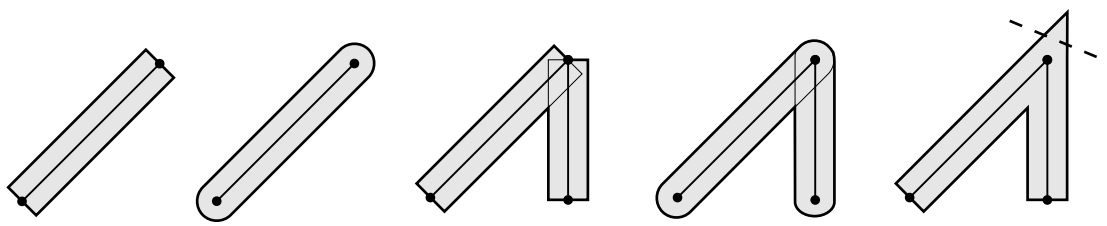
1. Síla čáry se mění podle sklonu úsečky.
2. Zakončení čar je nepřirozené. Je ostré a rovnoběžné s některou ze souřadnicových os.

První z nich opět souvisí s metrikou rastru. Šikmé čáry se jeví tenčí než svislé a vodorovné. Tento jev je zřetelný i při kresbě čar o minimální síle (tloušťce). U silných čar jej lze částečně odstranit přepočítáním požadované tloušťky čáry t na počet pixelů t_{pix} v závislosti na sklonu úsečky, jak naznačuje obrázek 3.5 vpravo. Pro případ prvního kvadrantu a směrnice $m \leq 1$ platí:

$$t_{pix} = t \frac{\sqrt{(\Delta x)^2 + (\Delta y)^2}}{|\Delta x|}. \tag{3.6}$$

Druhý nedostatek uvedené metody napravit nelze. U malé tloušťky čar nemusí působit rušivě, u velké je však výrazný.

Kvalitnější algoritmy proto pracují se silnou čárou jako s vyplněnou plochou, která se vykresluje pomocí postupů uvedených v kapitole 3.3. Převedení kresby silné čáry na vyplnění ohraničené oblasti je vhodné i z hlediska kresby silných lomených čar, jak ukazuje obrázek 3.6. Na něm jsou příklady obvyklého zakončení samostatných a lomených čar. Nejjednodušším tvarem silné čáry je obdélník. Jeho hranice se snadno vypočítá i vyplní. Zadávané koncové body leží ve středu kratších hran obdélníku.

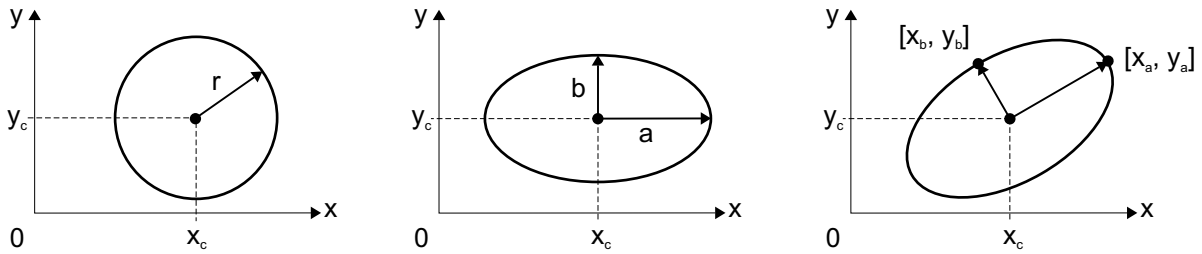


Obrázek 3.6: Zakončení silných (a, b) a lomených silných (c, d, e) čar

Výpočetně náročnější, ale často vzhledově lepší, je zakončení do oblouku (obr. 3.6b). To také automaticky řeší problém navazování úseček u lomené čáry (obr. 3.6d), zatímco pomocí obdélníků nelze dobré návaznosti docílit (obr. 3.6c). Některé systémy kreslí silnou lomenou čáru pomocí mnohoúhelníkové hranice (obr. 3.6e). V takovém případě je nutno ošetřit situaci, kdy sousední úsečky spolu svírají extrémně ostrý úhel. Obrys silné čáry v těchto místech vytvoří dlouhý hrot, který je třeba oříznout, jak na obrázku naznačuje čárkovaná čára.

3.2 Kružnice a elipsa

Kružnice je obyčejně definována středem $[x_c, y_c]$ a poloměrem r (obr. 3.7 vlevo). Elipsa je charakterizována středem a délkami dvou poloos a, b (obr. 3.7 uprostřed). Pokud její poloosy nejsou rovnoběžné s osami souřadnicové soustavy (obr. 3.7 vpravo), lze ji do otočením převést na předchozí případ.



Obrázek 3.7: Charakteristické hodnoty potřebné k zadání kružnice, osově orientované elipsy a elipsy v obecné poloze

Do stejné kategorie jako kružnice a elipsa se řadí kruhové a eliptické oblouky, při jejichž specifikaci je nutno navíc definovat buď dvojici koncových bodů oblouku nebo dvojici úhlů měřených od středu a zvoleného vztažného bodu objektu.

3.2.1 Rasterizace kružnice

Při kresbě kružnice v rastru můžeme využít již existujících metod pro kresbu úsečky a nahradit kružnici lomenou čárou. Její vrcholy lze vypočítat iteračním způsobem pomocí transformace otáčení (viz část 21.2).

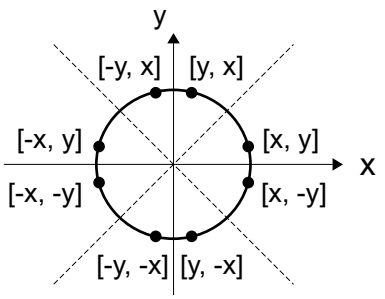
Poznamenejme, že poloha středu kružnice či elipsy pouze určuje posun od počátku soustavy souřadnic. Většina algoritmů proto pracuje s kružnicí a elipsou se středem v počátku. Získané body v rastru jsou posunuty do cílové polohy až před vykreslením.

Bresenhamův algoritmus pro kresbu kružnice

Podobně jako u úsečky můžeme získat body rasterizující kružnici pouze pomocí celočíselné aritmetiky. Navíc využijeme skutečnosti, že kružnice je středově symetrická, jak ukazuje obrázek 3.8. Z jediného vypočítaného bodu kružnice lze odvodit dalších sedm bodů záměnou souřadnic a změnou jejich znaménka. Pro rasterizaci celé kružnice tedy stačí vypočítat hodnoty souřadnic bodů ležících v jednom oktantu, např. v úseku od $x = 0$ do $x = y$.

J. E. Bresenham publikoval algoritmus [Bres77], v němž vyřešil generování bodů na kružnici obdobným způsobem jako v případě úsečky. V literatuře je tento algoritmus uváděn také jako *midpoint algorithm*.

Na obrázku 3.9 vidíme horní část kružnice. Tato část leží celá v jednom oktantu, ve kterém se souřadnice x sousedních bodů rasterizované kružnice liší právě o jeden pixel. Krok v ose x je tedy konstantní, vedlejší osou je osa y. Algoritmus začíná v bodě $[0, r]$ a končí v průsečíku kružnice s hlavní diagonálou, kdy $x = y$.



Obrázek 3.8: Symetrie na kružnici

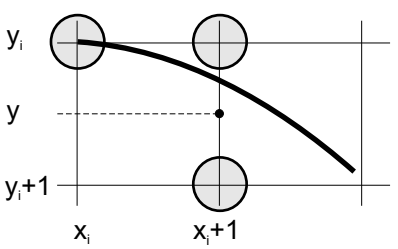
V následujícím textu se budeme odvolávat na pojmy, zavedené při popisu Bresenhamova algoritmu pro kresbu úsečky v části 3.1.1. I zde bude použit rozhodovací člen, určující souřadnici y následujícího bodu na kružnici. Výpočet však nebudeme odvozovat podrobně.

Pro body na kružnici platí implicitní rovnice $x^2 + y^2 - r^2 = 0$, kterou zapíšeme jako funkci:

$$F(x, y) : x^2 + y^2 - r^2 = 0. \tag{3.7}$$

Znaménko funkce určuje polohu bodu $[x, y]$ vůči kružnici. Funkční hodnota pro body uvnitř kružnice je záporná, pro body vně je kladná. Funkce F je vhodným kritériem při zavedení rozhodovacího členu.

Předpokládejme situaci na obrázku 3.9. Bod $[x_i, y_i]$ byl určen jako bod nejbližší skutečné kružnici. Následující bod může mít buď souřadnice $[x_i + 1, y_i]$, nebo $[x_i + 1, y_i - 1]$. Na obrázku je dále naznačen bod ležící v polovině mezi uvedenými dvěma kandidáty (*midpoint*). Dosadíme-li jeho souřadnice $[x_i + 1, y_i - 1/2]$ do funkce (3.7), znaménko výsledku určí, zda tento bod leží vně nebo uvnitř kružnice. Výslednou hodnotu budeme opět nazývat rozhodovacím členem p_i:



Obrázek 3.9: Část kružnice v rastru

$$\begin{aligned} p_i &= F(x_i + 1, y_i - \tfrac{1}{2}) \\ &= (x_i + 1)^2 + (y_i - \tfrac{1}{2})^2 - r^2. \end{aligned} \tag{3.8}$$

Je-li znaménko p_i záporné, bude pro další kresbu vybrán bod se stejnou souřadnicí y_i, jinak bude nakreslen bod ležící o jeden pixel níže.

Hodnotu rozhodovacího členu budeme stejně jako v případě úsečky určovat během iteračního

1. Inicializuj pomocné proměnné: $dvex = 3$, $dvey = 2r - 2$
2. Inicializuj rozhodovací člen p na hodnotu $1 - r$
3. Inicializuj $[x, y]$ jako $[0, r]$
4. Dokud je $x \leq y$, opakuj:
   (a) Vykresli osm bodů symetrických s bodem $[x, y]$
   (b) Je-li hodnota p kladná, pak
      i. $p = p - dvey$
      ii. $dvey = dvey - 2$
      iii. $y = y - 1$
   (c) $p = p + dvex$
   (d) $dvex = dvex + 2$
   (e) $x = x + 1$

Algoritmus 3.3: Bresenhamův algoritmus pro kresbu kružnice

výpočtu z předcházející hodnoty. Po úpravě získáme výsledný vzorec:

$$p_{i+1} = p_i + 2x_i + 3 + \left(y_i - \frac{1}{2}\right)^2 + \left(y_{i+1} - \frac{1}{2}\right)^2. \tag{3.9}$$

Vzorec (3.9) můžeme tedy vyčíslit podle znaménka p_i:

$p_i \leq 0$	$p_{i+1} = p_i + 2x_i + 3$
$p_i > 0$	$p_{i+1} = p_i + 2x_i + 5 - 2y_i$

Výsledný algoritmus 3.3 je analogický algoritmu 3.2 pro kresbu úsečky. Poznamenejme jen, že ačkoliv vzorce pro aktualizaci rozhodovacího členu obsahují násobení, lze je implementovat jen pomocí sčítání a odečítání. Pro členy $2x_i$ a $2y_i$ zavedeme pomocné proměnné $dvex$ a $dvey$, obsahující dvojnásobek příslušné souřadnice. Když se hodnota x, resp. y změní o jedničku, aktualizujeme příslušnou pomocnou proměnnou přičtením, resp. odečtením dvojky. Tyto proměnné inicializujeme tak, abychom se zbavili konstant tři a pět ve výše uvedené tabulce.

Algoritmus se používá i při kresbě kruhového oblouku. Podle koncových bodů se nastaví omezující podmínky ve funkci, vykreslující osm symetrických bodů. Kresba bodů mimo oblouk je potom těmito podmínkami potlačena.

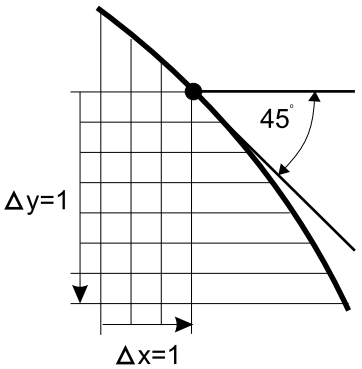
3.2.2 Rasterizace elipsy

Nejjednodušší je kresba osově orientované elipsy. S využitím iteračního postupu Bresenhamova algoritmu lze nalézt body na elipse pomocí celočíselné aritmetiky. Na rozdíl od kružnice lze symetrii využít pouze pro tři další body a algoritmus musí vygenerovat body elipsy v jednom celém kvadrantu.

Rovnici elipsy se středem v počátku vyjádříme pomocí implicitní funkce:

$$F(x,y) : b^2\ x^2 + a^2\ y^2 - a^2\ b^2 = 0. \qquad (3.10)$$

Z té potom odvodíme pomocí středového bodu (*midpoint*) rozhodovací člen obdobně jako u kružnice.



Obrázek 3.10: Změna řídicí osy při rasterizaci elipsy

Na obrázku 3.10 vidíme, že při vzorkování elipsy dochází ke změně řídicí osy. Generujeme-li body v prvním kvadrantu zleva doprava, je řídicí osou nejprve osa x. Ve druhé části algoritmu se stane řídicí osou osa y. Bod, ve kterém dojde ke změně, je na obrázku zvýrazněn. V něm má tečna k elipse směrnici -1. Vyjádříme-li vztah (3.10) jako funkci proměnné y, položíme její derivaci podle x rovnu -1 a získáme hledaný bod o souřadnicích:

$$\left[\frac{a^2}{\sqrt{a^2+b^2}}, \frac{b^2}{\sqrt{a^2+b^2}}\right]$$

V části s řídicí osou x se pro výpočet rozhodovacího členu používají následující pravidla:

$p_i \leq 0$	$p_{i+1} = p_i + b^2\ (2x_i + 1)$
$p_i > 0$	$p_{i+1} = p_i + b^2\ (2x_i + 1) - 2a^2\ y_i$

Ve druhé části algoritmu jsou vztahy podobné. Při implementaci lze součiny ve výrazech pro rozhodovací člen nahradit přičítáním pomocných proměnných podobně jako v algoritmu 3.3 pro kresbu kružnice.

V literatuře [Fell93, Fole90] nalezneme metody pro kresbu elipsy v obecné poloze i pro kresbu eliptického oblouku. Také tyto algoritmy lze provádět v celočíselné aritmetice.

3.3 Oblast

Dosud uvedená grafická primitiva měla liniový (čárový) charakter. Pro kreslení plošných obrazců je vhodné využít velmi obecného a tedy univerzálního prvku, nazývaného oblast (*area*). Oblast

je vždy uzavřená a její vnitřek lze vyplnit různými způsoby. Není přitom důležité, zda oblast je či není konvexní.

Definice oblasti souvisí především s popisem její hranice. Rozlišujeme dva základní způsoby popisu.

1. *Geometricky určená hranice*
   Je zadána nejčastěji pomocí posloupnosti bodů definujících mnohoúhelník (polygon). Poslední vrchol se implicitně spojuje s prvním vrcholem, aby takto vzniklá hranice byla uzavřená. Jinou možností je zadání hranice pomocí jedné nebo více navazujících křivek, případně definice jako průnik poloprostorů.
2. *Hranice nakreslená v rastru*
   Tvar hranice může být libovolný. Je třeba znát buď barvu hranice, nebo barvu vnitřních bodů oblasti, případně barvu bodů vně oblasti. Před vyplňováním takové hranice uživatel dále zadává souřadnice vnitřního bodu oblasti.

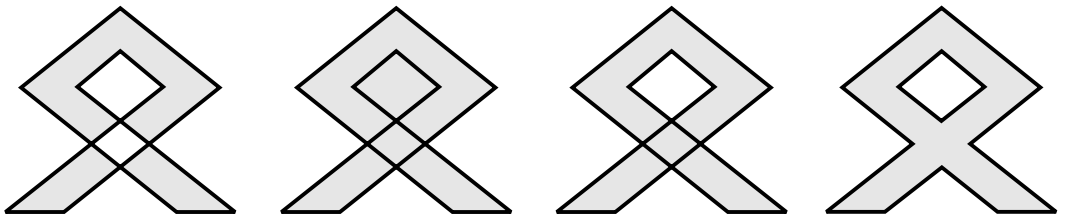
Úlohu vyplňování oblasti můžeme rozdělit na dvě části – nalezení vnitřních bodů a obarvení vnitřních bodů. Hledání vnitřních bodů je ve většině případů algoritmicky složitější než jejich obarvení. K obarvení vnitřních bodů může být použita jedna *barva* (*solid fill*), *šrafování* (*hatch fill*), opakovaně nanášený *vzorek* složený z více barevných bodů (*pattern tilling*), případně může být barva výplně stanovena pomocí složitějšího výpočtu, jako je např. interpolace barvy. Vykreslení samotné hranice oblasti bez vyplnění vnitřku se anglicky označuje různými názvy – *empty, void, hollow.*

3.3.1 Vyplňování geometricky určené hranice

V případě, že se geometricky zadaná hranice sama protíná, je důležité rozhodnout, které body budeme považovat za vnitřní. Na obrázku 3.11 jsou znázorněny různé možnosti. *Paritní vyplňování* (vlevo) je nejběžnější. Odpovídá intuitivnímu pohledu na hranici jako na entitu oddělující vyplněný a nevyplněný prostor. Pro vnitřní bod platí, že libovolná polopřímka z něj vedená protne lichý počet hran oblasti.

Ve druhém případě (obr. 3.11b) jsou vyplněny všechny body, které neleží vně hranice. Všechny polopřímky vedené z vnitřního bodu protnou hranici oblasti. Další variantou je tzv. obtočení bodu hranicí, která je určena uzavřeným polygonálním tahem s možností sebeprotínání (obr. 3.11c).

Paritní vyplňování je přirozené a nejjednodušší, další dvě varianty vyžadují složitější výpočty. V dalším textu uvedeme různé algoritmy paritního vyplňování. Jsou schopny zpracovat i oblast určenou několika hranicemi (obr. 3.11d). Základní metodou je řádkové vyplňování. Každým řádkem rastru je vedena pomyslná vodorovná čára a jsou hledány její průsečíky

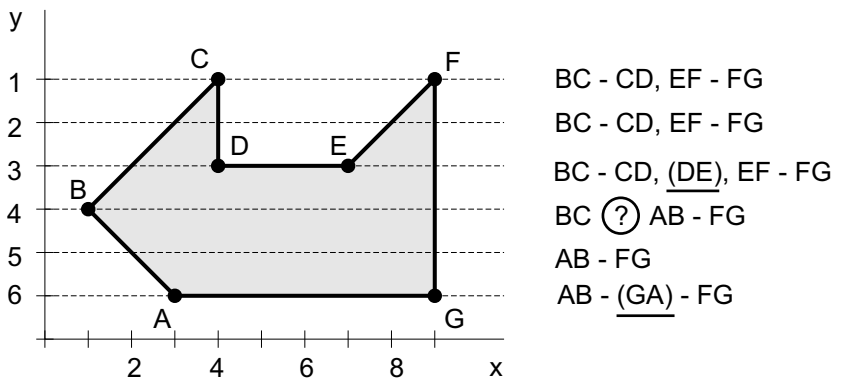


a) paritní vyplňování b) vnitřní vyplňování c) obtočení bodu d) více hranic

Obrázek 3.11: Varianty vyplňování složitější geometricky určené hranice

s hranicemi oblasti. Nalezené průsečíky na jedné čáře se seřadí dle souřadnic x. Dvojice těchto průsečíků definují úsečky ležící uvnitř oblasti. Tato metoda se někdy také nazývá *paritní řádkové vyplňování* nebo *vyplňování rozkladovými řádky* (*scan–line conversion*).

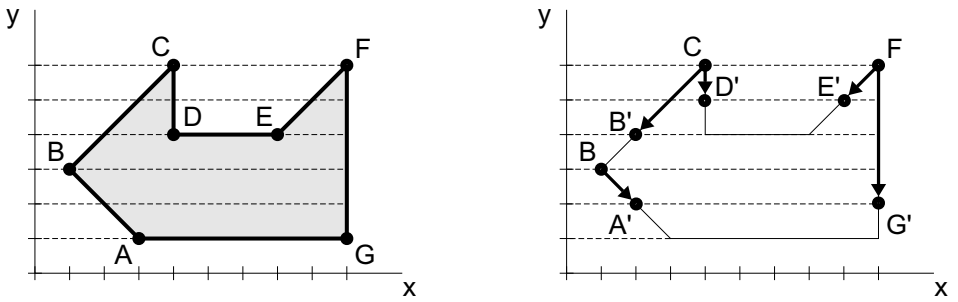
Vzhledem k tomu, že se řádky zpravidla číslují shora dolů, použijeme v dalším textu takové značení a orientaci osy y, které budou odpovídat číslování řádků. Kladná poloosa y bude proto mířit dolů, levý horní roh obrazovky bude mít souřadnice $[0, 0]$. Tato konvence je běžná u mnoha jednodušších kreslicích programů a připomíná, že počítačová grafika sice vychází z klasické geometrie (která používá osu y orientovanou vzhůru), ale přizpůsobuje ji počítačové praxi.



Obrázek 3.12: Mnohoúhelník vyplňovaný řádkovým rozkladem

Na jednoduché oblasti (obr. 3.12) si všimneme typických poloh rozkladového řádku a hraničních čar. Mnohoúhelník tvořící hranici je určen sedmi vrcholy A až G. Hledání vnitřních úseček probíhá shora dolů. V pravé části obrázku jsou vypsány hraniční úsečky, které daný řádek protíná.

První rozkladový řádek protíná celkem čtyři hraniční čáry, takže dojde k nakreslení dvou úseček. Ty mají v tomto případě obě nulovou délku, neboť leží ve vrcholech C a F. Grafický systém může úsečku o nulové délce vykreslit jako jeden pixel. Druhý řádek je zpracován obdobně. Ve třetím řádku musí být z výpočtu vyloučena hrana DE, neboť má s rozkladovým řádkem nekonečně mnoho průsečíků. Zbylá čtveřice průsečíků opět určuje dvě vnitřní úsečky.



Obrázek 3.13: Mnohoúhelníková oblast a úpravy jejích hraničních úseček před vyplňováním

Problém nastane na čtvrtém řádku, kde je nalezen lichý počet průsečíků. Tuto situaci způsobuje vrchol B. Lze jej charakterizovat jako vrchol, který není lokálním extrémem hranice (minimem či maximem) ve smyslu souřadnice y a přitom neleží na vodorovné hraniční čáře. Standardním řešením tohoto problému je *zkrácení* hranice BC zdola o 1 pixel ve směru osy y. Toto zdánlivé rozpojení dosud uzavřené hranice nemá vliv na činnost algoritmu. Ukazuje se naopak, že *systematické zkrácení* všech hran vede ke snížení počtu průsečíků, přičemž funkčnost metody zůstává zachována. Obrázek 3.13 ukazuje, že po takové úpravě zbudou například na třetím řádku pouze dva průsečíky místo nadbytečných čtyř.

Řešení všech možných vzájemných poloh rozkladových řádků s hranicí oblasti tedy vyžaduje předzpracování jednotlivých hraničních úseček. Je vhodné vynechat vodorovné hraniční úsečky, ostatní pak orientovat shora dolů a zkrátit je u dolního vrcholu. Celkově popisuje metodu paritního řádkového vyplňování algoritmus 3.4.

K tomu, zda má být na závěr algoritmu znovu vykreslena hranice, se vrátíme později, kdy budeme uvažovat o tom, zda hranice je či není součástí vyplňované oblasti.

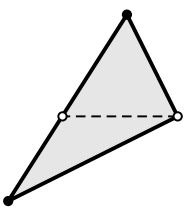
Uvedená metoda je efektivní pro vyplňování oblasti ohraničené mnohoúhelníkem. Lze ji použít i pro oblast ohraničenou obecnými křivkami. Při požadavku na velkou rychlost zpracování může být mnohdy jednodušší nahradit křivky lomenou čárou a použít původní, neupravený algoritmus.

1. Pro všechny hraniční úsečky ověř:
   (a) je-li vodorovná, vynechej ji (případně ji vykresli)
   (b) uprav orientaci shora dolů a zkraťji zdola o 1 pixel
   (c) aktualizuj mezní souřadnice celé hranice y_{max} a y_{min}
2. Pro y od y_{min} do y_{max} proveď:
   (a) nalezni průsečíky hraničních úseček s řádkem y
   (b) uspořádej všechny průsečíky podle souřadnice x
   (c) vykresli úseky mezi lichými a sudými průsečíky
3. Vykresli hranici oblasti (je-li třeba)

Algoritmus 3.4: Algoritmus vyplňování řádkovým rozkladem

3.3.2 Vyplňování trojúhelníka

Vyplňování trojúhelníků je častou úlohou, která je využívána zejména v trojrozměrné počítačové grafice. Namísto obecného algoritmu pro vyplňování polygonu je pro trojúhelník vhodné použít efektivnější, specializovanou metodu, která je naznačena na obrázku 3.14. Obecný trojúhelník rozdělíme vodorovným řezem na dvě části – horní a dolní – dle toho vrcholu trojúhelníka, jehož souřadnice y leží mezi souřadnicemi y zbylých dvou vrcholů. Každý z takto vzniklých elementárních trojúhelníků je plně popsán právě dvěma hranami [třetí, zbylá hrana je vodorovná a lze ji tudíž dle kroku 1(a) v algoritmu 3.4 vynechat]. Vyplnění oblasti mezi dvěma hranami již nevyžaduje žádné řazení průsečíků. Stačí pouze souběžně postupovat po obou hranách shora dolů a vykreslovat mezi nimi vodorovné spojnice.



Obrázek 3.14: Rozdělení trojúhelníka na dvě elementární oblasti

3.3.3 Další metody vyplňování polygonů

Algoritmus 3.4 řádkového vyplňování geometricky zadané hranice opakuje některé výpočty zbytečně a nevyužívá dříve získaných výsledků. Především kroky 2a) a 2b) lze výrazně urychlit, a to využitím různých *souvislostí* (*coherence*) a vlastností zpracovávané oblasti. *Koherence* (jinak též spojitost) znamená, že hodnota určité veličiny může být poměrně snadno odvozena

z hodnot již zpracovaných veličin bez nutnosti provádění celého výpočtu. V našem případě využijeme dvou koherencí:

- hranová koherence umožní určit průsečík s rozkladovým řádkem ze znalosti předchozího průsečíku,
- řádková koherence urychlí řazení průsečíků.

Uvažujme nejprve *hranovou koherenci*. Jakmile je jednou spočítána souřadnice průsečíku rozkladové řádky s určitou hranou, lze snadno určit všechny ostatní průsečíky této hrany s následujícími řádkami postupným přičítáním konstanty. Tato konstanta je rovna převrácené hodnotě směrnice dané hrany. Neceločíselnou hodnotu výsledné souřadnice x je nutno před vykreslením zaokrouhlit. Připomeňme, že výpočet průsečíků rozkladové řádky s hranou je vlastně rasterizací hrany prováděnou podle řídicí osy y (bez ohledu na směrnici hrany). Použijeme-li algoritmus DDA (viz alg. 3.1), je optimalizace založena na uchování dílčích výpočtů DDA pro každou z hran.

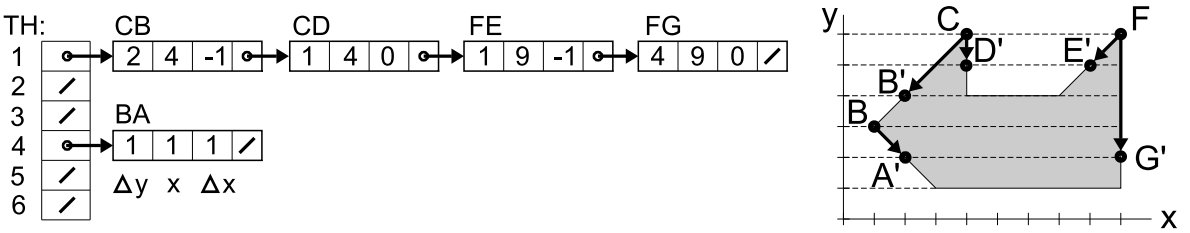
Další úspory času docílíme využitím *řádkové koherence*. Předpokládáme, že při přechodu na další rozkladový řádek budou protnuty především ty hrany, které byly protnuty nyní. Navíc se ve většině případů nezmění pořadí průsečíků na řádku. Z těchto úvah vychází následující úprava.

Řádkové vyplňování se seznamem aktivních hran

Nejprve seřadíme všechny zkrácené a zorientované hrany podle horní souřadnice y. Příklad je uveden na obrázku 3.15, kde bylo použito „přihrádkové třídění“, jehož výsledkem je zařazení hran do *tabulky hran*. Jedna položka tabulky odpovídá jednomu rozkladovému řádku a obsahuje seznam těch hran, jejichž horní bod má souřadnici y shodnou s tímto rozkladovým řádkem. Hrany jsou orientovány shora dolů a popsány záznamem s následujícími položkami:

int	Δy	počet řádků, do nichž hrana zasahuje
float	x	aktuální souřadnice x průsečíku s rozkladovým řádkem
float	Δx	přírůstek x při přechodu na další řádek

Další datovou strukturou je *seznam aktivních hran*, který obsahuje pouze ty hrany, které mají průsečík s právě zpracovávaným rozkladovým řádkem. Hrany v tomto seznamu budeme udržovat seřazené podle souřadnic x. Především zde se projeví časová úspora metody – při přechodu na nový řádek zůstává uspořádání průsečíků ve většině případů beze změny. Uspořádání je narušeno pouze v případě protínajících se hran nebo pokud je do seznamu přidána nová hrana z tabulky hran. Tyto případy snadno řeší *bublinkové řazení* (*bubble–sort*) s detekcí počtu záměn.



Obrázek 3.15: Uspořádaná tabulka hran

Při přechodu na nový řádek směrem dolů provedeme jednoduchou aktualizaci seznamu aktivních hran: u každé hrany zmenšíme Δy o jedničku a při dosažení nuly hranu z dalšího zpracování vyloučíme, přičtením přírůstků Δx určíme nové hodnoty x, přidáme případné nové hrany z tabulky hran a seznam uspořádáme dle x. Postup je zapsán v algoritmu 3.5. Při implementaci není nutno používat dvě datové struktury – tabulku TH a seznam SAH. Všechny hrany lze seřadit do jediného seznamu a SAH pak tvoří podseznam na jeho začátku.

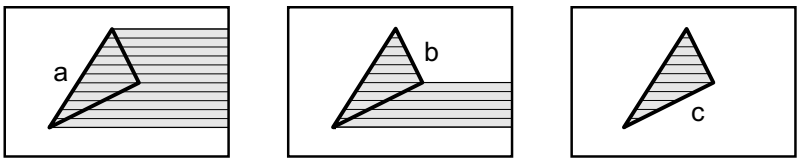
1. Vytvoř uspořádanou tabulku hran TH
2. Vytvoř prázdný seznam aktivních hran SAH
3. Nastav y na první souřadnici y v TH
4. Dokud nejsou TH a SAH prázdné, opakuj:
   (a) Přesuň do SAH hrany z TH[y]
   (b) Uspořádej SAH (bublinkovým řazením) dle x
   (c) Vykresli úseky mezi lichými a sudými hranami v SAH
   (d) Zruš ze SAH hrany, jejichž $\Delta y = 0$
   (e) Pro všechny záznamy hran v SAH proveď:
      i. $\Delta y = \Delta y - 1$
      ii. $x = x + \Delta x$
   (f) Zvyš y o 1

Algoritmus 3.5: Řádkové vyplňování se seznamem aktivních hran

Inverzní vyplňování a šablona

Metoda řádkového rozkladu není jedinou vyplňovací metodou pro geometricky určené hranice. Její slabinou je nutnost řazení průsečíků na každém rozkladovém řádku, které můžeme pouze různými způsoby zrychlovat, ale nikoliv odstranit. Byly proto hledány způsoby vyplňování, které by nevyžadovaly žádné řazení. Jedním z nich je inverzní vyplňování, které vykazuje lineární časovou závislost na počtu hraničních úseček. My jej uvedeme především kvůli jeho principu, na němž ukážeme využití pomocné paměti – *šablony.*

Šablonou (*stencil*) nazýváme část paměti, ve které je prováděno vyplňování jedné oblasti. Jeden pixel obrazové paměti může být v šabloně reprezentován jediným bitem, který postačí pro informaci „vyplněno/nevyplněno“. Šablona určuje místa (souřadnice pixelů), do kterých se později vykreslí výplň oblasti.



Obrázek 3.16: Postup inverzního vyplňování při zpracování hranic a, b, c

Při metodě inverzního vyplňování jsou jednotlivé hraniční úsečky zpracovávány samostatně. Každou hranu nejprve zkrátíme o jeden pixel zdola a poté ji rasterizujeme algoritmem DDA s řídicí osou y. Od každého vypočítaného pixelu na hraně vedeme vodorovnou polopřímku k pravému okraji šablony. Pixely takto určené úsečky se zaznamenají do šablony. Zápis se provede invertováním (negací) hodnot v šabloně. Po zpracování první úsečky je takto vyplněn lichoběžník zleva omezený touto úsečkou a zprava okrajem šablony. Zpracování dalších hraničních úseček způsobí smazání části tohoto lichoběžníku, případně jeho doplnění o nové vyplněné oblasti. Postup je dokumentován obrázkem 3.16.

Metoda inverzního vyplňování je velmi rychlá, neboť nevyžaduje ani iniciální třídění hran podle souřadnice y (jako tomu bylo v alg. 3.5), ani řazení průsečíků podle souřadnice x. Dalšího zrychlení lze docílit efektivnějším zpracováním informací v šabloně. Příkladem je algoritmus 3.6, který se od základní metody inverzního vyplňování liší tím, že do šablony zaznamenává pouze polohy hraničních, nikoliv vnitřních pixelů oblasti. Tím se významně sníží počet zápisů do šablony. Pro výslednou šablonu v tomto případě neplatí, že logické jedničky určují vyplněný pixel v obrazové paměti. Jedničky v šabloně je třeba interpretovat jako okraje vodorovných úseků, které budou vyplněny v obrazové paměti.

Šablona se stala běžnou součástí grafických akcelerátorů. V nich mívá rozměr totožný

1. Vyčisti (vynuluj) šablonu
2. Pro každou nevodorovnou hranu dělej:
   (a) zkrať hranu o jeden pixel zdola,
   (b) rasterizuj hranu metodou DDA s řídicí osou y a pro každý vypočítaný pixel $[x, y]$ hrany invertuj místo v šabloně na pozici $[x, y]$.
3. Pro každý řádek šablony najdi pozice nastavené na logické jedničky. Do obrazové paměti nakresli vodorovné úseky odpovídající lichým a sudým pozicím v šabloně.

Algoritmus 3.6: Vyplňování s pomocí zápisu hranice oblasti do šablony

s velikostí obrazovky a slouží pak jako univerzální pomocná paměť (*stencil buffer*) vhodná k ukládání dočasných hodnot pro rozmanité algoritmy.

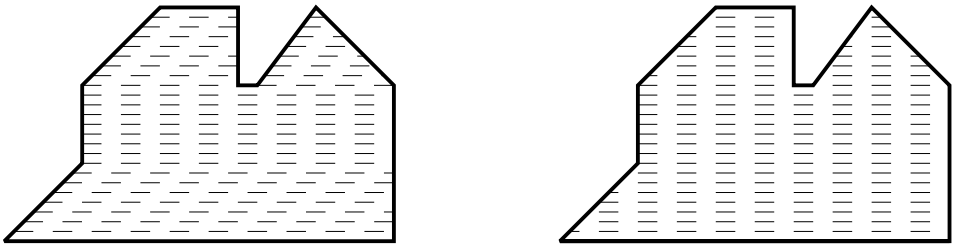
Vyplňování vzorem

Přirozeným zobecněním metody řádkového vyplňování je vyplňování vzorem definovaným v rastru. Zatímco v základní metodě jsou vnitřní úsečky přímo vybarveny požadovanou vnitřní barvou, nyní je na ně opakovaně kladen barevný vzor. Definice vzoru je uložena v obdélníkové matici $\mathbf{V_{m,n}}$. Hodnoty, definující cílovou barvu každého pixelu, se z matice vybírají podle souřadnic kresleného pixelu. Hledané souřadnice prvku matice $\mathbf{V}$ jsou určeny celočíselným zbytkem po dělení (*modulo*) příslušnými rozměry této matice. Protože se vyplňování uskutečňuje po řádcích, je možno zvýšit efektivitu algoritmu tím, že do obrazové paměti budeme opakovaně kopírovat celé řádky matice $\mathbf{V}$.

Šrafování

Vyplňování šrafováním je rozšířeno především v technických aplikacích. Rozšíření řádkového vyplňování na šrafování není složité. V případě vodorovných šraf stačí jen změnit krok změny souřadnice y z hodnoty 1 na m (m je celočíselné a kladné) a získáme oblast vyplněnou vodorovnými šrafami s roztečí m pixelů.

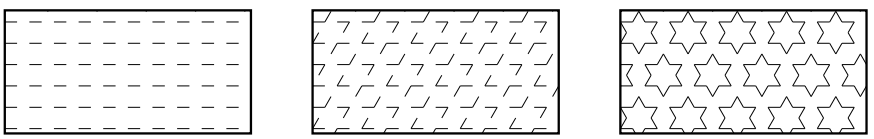
Při šrafování přerušovanými čárami může dojít k optickému jevu, který je naznačen na obrázku 3.17a. Pokud bude kreslení každé z přerušovaných čar zahájeno od levého kraje vodorovného úseku, čárkování vytvoří při pohledu z dálky určitou siluetu, která je do jisté míry kopií levé hranice oblasti. Odstranění tohoto nežádoucího jevu spočívá v úpravě algoritmu pro



Obrázek 3.17: Použití přerušovaných čar při šrafování oblasti
a) s pevným začátkem b) vzhledem ke vztažnému bodu

kreslení přerušovaných čar tak, aby čárkování bylo nanášeno relativně k určitému vztažnému bodu, například k počátku soustavy souřadnic. Umístění jednotlivých čárek ve vyšrafované oblasti se tak stane nezávislým na tvaru a umístění oblasti (obr. 3.17b).

Šrafování přerušovanou čárou se v tomto smyslu podobá vyplňování rastrovým vzorem, který je také umístěn nezávisle na poloze hranice. Oblast je považována za výřez, kterým uživatel nahlíží na „nekonečný“ vzor na pozadí. V některých aplikacích, jako jsou například animace, je naopak žádoucí, aby se vzor pohyboval spolu s oblastí. Tehdy zvolíme jeden z vrcholů hranice za vztažný bod, vůči kterému provádíme jak šrafování, tak vyplňování vzorem.



Obrázek 3.18: Vytváření vzoru opakovaným šrafováním pod třemi různými úhly

Spíše než vodorovné šrafování se však v praxi uplatňuje šrafování pod obecným úhlem α. Běžné vodorovné vyplňování proto zkombinujeme s transformací otáčení (viz kap. 21.2). Nejprve otočíme všechny hraniční úsečky oblasti o úhel -α. Na takto upravenou hranici použijeme algoritmus vodorovného šrafování z předchozího odstavce s tím, že vypočítané vnitřní úsečky otočíme o úhel α a teprve poté je vykreslíme. Opakovaným šrafováním téže oblasti pod různými úhly lze vytvořit z čárkovaných čar složitější vzory. Jeden z příkladů je na obrázku 3.18.

Kreslení hranice

Dosud jsme podrobněji nestudovali vztah mezi hranicí a vnitřkem oblasti z hlediska kresby v rastru. Hranici můžeme chápat buď jako součást vyplňované oblasti nebo jako samostatnou

entitu, která odděluje oblast od zbylého prostoru a je tedy „neutrální“. Výše uvedené algoritmy paritního vyplňování označí za vnitřní body i některé body na hranici oblasti. Neplatí však, že množina hraničních pixelů je podmnožinou vnitřních pixelů. Jinak řečeno, pokud byla nejprve nakreslena určitou barvou hranice oblasti, následné vykreslení vnitřních bodů jinou barvou hraniční pixely nepřekryje. To se projeví na hraničních úsečkách, které byly při samostatném vykreslení rasterizovány s řídicí osou x. Při hledání vnitřních bodů oblasti však rasterizujeme vždy s řídicí osou y, čímž získáme odlišné množiny pixelů.

Kresba hraničních bodů je důležitá i v situaci, kdy se na společné hranici stýkají dvě sousední oblasti. Pokud vyplňování sousedních polygonů vede na překreslení hranice, závisí výsledný vzhled obrázku na pořadí vykreslování ploch. Opakované kreslení bodů na společné hranici je zbytečné a tedy neefektivní. Řešením je dohoda, že například ta část hranice, která je *vlevo* a *nahoře*, bude patřit k vyplňované oblasti, zatímco zbylá část nikoliv – ta bude patřit případné sousední oblasti. Všimněme si, že právě zkracování hran o jeden pixel zdola automaticky „uvolní“ dolní část hranice pro sousední oblast, ač tato úprava byla původně určena pro řešení problému násobného průsečíku dvou navazujících hran. Uvolnění hranice vpravo realizujeme zkrácením vykreslovaných vodorovných úseků, tedy kresbou úsečky $[x_L, y], [x_R - 1, y]$ namísto $[x_L, y], [x_R, y]$. V případě $x_L = x_R$ kresbu osamoceného pixelu zcela potlačíme.

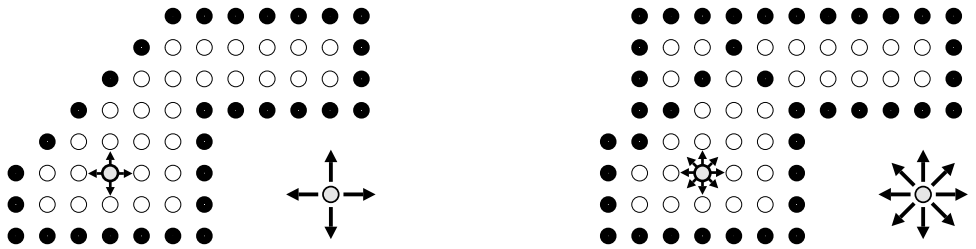
3.3.4 Vyplňování hranice nakreslené v rastru

Metody pro vyplňování již nakreslené hranice v rastru jsou obecně nazývány *semínkové vyplňování* (*seed fill*), neboť jejich nezbytným parametrem jsou souřadnice *semínka* – vybraného vnitřního bodu oblasti. Vzhledem k tomu, že hranice není geometricky jasně definována, všechny informace o vyplňované oblasti se získávají čtením z obrazové, tedy rastrové paměti. Od semínka se postupně rozšiřuje prohledávání obrazové paměti a nalezeným vnitřním bodům se nastaví nová barva.

Směr prohledávání a vyplňování úzce souvisí s charakterem hranice, resp. s tím, co považujeme za její vnitřek. Rozlišujeme *4spojité (4souvislé)* a *8spojité (8souvislé)* oblasti podle toho, zda mezi jejich libovolnými dvěma vnitřními body existuje cesta složená ze 4spojitých, resp. 8spojitých spojnic.

Každému typu oblasti odpovídá jiný typ hranice, jak ukazuje obrázek 3.19. Například Bresenhamův algoritmus vytváří 8spojité posloupnosti pixelů, které ohraničují 4spojité oblasti. Pro ohraničení 8spojité oblasti jsou však nedostačující a algoritmy pro vyplňování by mohly vést k vyplnění rastrových bodů i za takto nakreslenou hranicí. Pro ohraničení 8spojité oblasti musíme použít 4spojitou hranici. Ve většině systémů jsou zpracovávány 8spojité hranice a tedy 4spojité oblasti.

Při semínkovém vyplňování tedy postupujeme od zadaného semínka a zkoumáme, zda jeho



Obrázek 3.19: 4spojitá oblast (vlevo) ohraničená 8spojitou hranicí a 8spojitá oblast (vpravo) ohraničená 4spojitou hranicí, v obou případech se zvýrazněným semínkem

sousední body patří k vnitřku oblasti. O příslušnosti bodu k oblasti rozhodujeme podle nějaké testované vlastnosti, například barevné intenzity. Existují dvě základní varianty:

1. *Hraniční vyplňování*
   Testovaný bod je vnitřní, má-li testovanou vlastnost odlišnou od vlastnosti hranice, např. jinou barvu.

2. *Záplavové vyplňování*
   Testovaný bod je vnitřní, má-li shodnou vlastnost jako zadané semínko. Této metodě se také říká lavinové vyplňování či *přebarvování*.

Test porovnání vlastností může být složitější, než jen ověření shody dvou hodnot. Můžeme požadovat, aby testovaná vlastnost byla v určitém rozsahu hodnot, například v intervalu jasu či barevného odstínu. To je vhodné v situaci, kdy hranice byla vyhlazena a získala nestejný odstín.

Nejjednodušší metoda je popsána rekurzivním algoritmem 3.7. Na první pohled elegantní, přehledný zápis je ve skutečnosti téměř nepoužitelný. Šíření semínek do všech směrů má totiž za následek, že každý vnitřní pixel je testován několikrát i poté, co již byl obarven. Čtení z obrazové paměti přitom patří mezi pomalé operace.

Řádkové semínkové vyplňování

Metoda, která snižuje počet přístupů do obrazové paměti, se nazývá *řádkové semínkové vyplňování* (*scan–line seed fill*). Je uvedena v algoritmu 3.8. Princip algoritmu spočívá ve vyplňování souvislých vodorovných úseků a prohledávání intervalů nad a pod těmito úseky. Každá vodorovná řada vnitřních bodů nad, resp. pod daným úsekem tvoří nový úsek, který je rekurzívně zpracován. Algoritmus je třeba inicializovat nalezením prvního úseku, který obsahuje zadané semínko.

UmístiSemínko (x, y)

pokud je bod [x, y] vnitřním bodem a dosud nebyl obarven, pak

1. obarvi bod [x, y] požadovanou barvou
2. UmístiSemínko (x+1, y)
3. UmístiSemínko (x−1, y)
4. UmístiSemínko (x, y+1)
5. UmístiSemínko (x, y−1)

Algoritmus 3.7: Semínkové vyplňování rekurzivním postupem

Na obrázku 3.20 je nakreslen stav vyplňované oblasti při postupném provádění algoritmu. Startovním semínkem je bod A. Při inicializaci je nalezen úsek obsahující tento bod a končící na hranicích oblasti. Poté je možno zavolat algoritmus 3.8.

Po vyplnění úseku v okolí bodu A jsou testovány intervaly nad a pod tímto úsekem. Výsledkem je volání algoritmu pro úseky s bodem B (z horní oblasti) a C (z dolní oblasti). Po vyplnění úseku s bodem C jsou nalezeny volné úseky s body D, E a F. Postupně je vyplněna oblast pod úsekem E, nad úsekem D a nakonec nad úsekem B.

Vyplňování rastrové oblasti vzorem a šrafování

Barevný vzor definovaný polem barev lze nanášet do rastru stejně jako při vyplňování geometricky určené hranice. Pokud vzor obsahuje pixely s takovými vlastnostmi, jaké mají vnitřní body oblasti, mohlo by dojít u výše uvedených algoritmů k zacyklení. Vyplněné body by mohly být považovány za vnitřní a algoritmus by se je pokoušel znovu a znovu vyplňovat. Tomu předejdeme tím, že do šablony zaznamenáme stav vnitřních bodů – vyplněn/nevyplněn.

Přesné geometrické šrafování je pro oblasti zadané v rastru poměrně složitou operací. Můžeme je převést na vyplňování vzorem, a to tehdy, pokud šrafy v sousedních vzorcích na sebe navazují. Při obecném sklonu šraf se může stát, že rozměr vzoru dosáhne rozměru rastru. Proto je vhodné opět použít šablonu, do níž zaznamenáme vnitřní body oblasti. Přesné šrafovací čáry pak vedeme pomyslně přes šablonu. Pro každý pixel šrafovací čáry porovnáme, zda padne do vyplněné části šablony a v kladném případě pixel vykreslíme do obrazové paměti.

VyplňÚsek (y, x_L, x_R)

1. vyplň pixely v úseku od $[x_L, y]$ do $[x_R, y]$
2. v (horním) intervalu mezi $[x_L, y-1]$ a $[x_R, y-1]$ hledej souvislé vnitřní úseky. Pro každý i-tý úsek proveď:

   VyplňÚsek ($y-1$, x_{Li}, x_{Ri})
3. v (dolním) intervalu mezi $[x_L, y+1]$ a $[x_R, y+1]$ hledej souvislé vnitřní úseky. Pro každý j-tý úsek proveď:

   VyplňÚsek ($y+1$, x_{Lj}, x_{Rj})

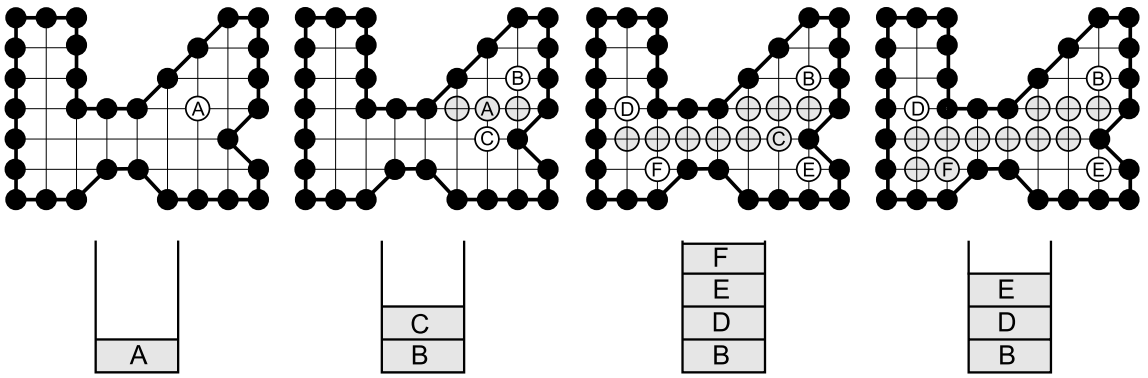
Algoritmus 3.8: Řádkové semínkové vyplňování

3.4 Ořezávání dvourozměrných objektů

Ořezávací (*clipping*) algoritmy umožňují vybrat z rozsáhlé kresby pouze ty části, které jsou viditelné v určité oblasti, například v oknu na obrazovce. Proto je ořezávací oblast obvykle definována jako osově orientovaný obdélník, tj. obdélník se stranami rovnoběžnými s osami souřadnicového systému a pro ořezávání se používají specializované algoritmy. Další oblastí, ve které se používají ořezávací algoritmy, je řešení viditelnosti 3D scény. Zde hrají důležitou roli algoritmy pro ořezávání polygonů, které umožňují korektně vymezit části polygonů viditelné v okně ve formě uzavřených polygonálních tahů. Některé aplikace však obsahují i obecnější algoritmy pro ořezání nekonvexním mnohoúhelníkem.

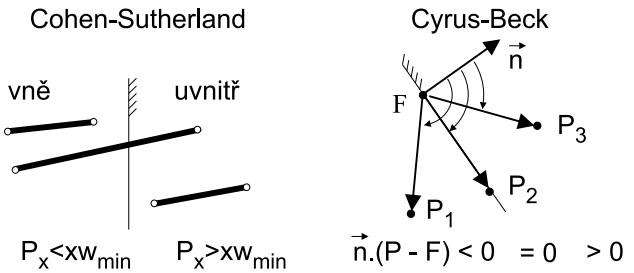
Ořezávací algoritmy jsou specializovány na:

1. *Test polohy bodu*
   Spíše než o ořezání se jedná o rychlý test, zda bod patří či nepatří do zadané ořezávací oblasti. Tento univerzální test je použitelný v řadě algoritmů, které pracují s jednotlivými body.
2. *Ořezání úsečky*
   Tímto způsobem se nejčastější ořezávají liniové objekty, tj. úsečky, lomené čáry a libovolné křivky nahrazené posloupností úseček.
3. *Ořezání oblasti*
   Uzavřené oblasti musí být zpracovávány speciálním postupem tak, aby výsledkem ořezání byla opět uzavřená oblast (či několik uzavřených oblastí). Na okraji ořezávacího okna přitom mohou vzniknout zcela nové části hranice původní oblasti.



Obrázek 3.20: Postup při řádkovém semínkovém vyplňování

V dalších algoritmech se zaměříme zejména na ořezání obdélníkovým oknem. Je určeno krajními body o souřadnicích $[xw_{min}, yw_{min}]$ (levý dolní roh) a $[xw_{max}, yw_{max}]$ (pravý horní roh).



Obrázek 3.21: Test pozice bodu vzhledem k hranici

3.4.1 Test polohy bodu vzhledem k oknu

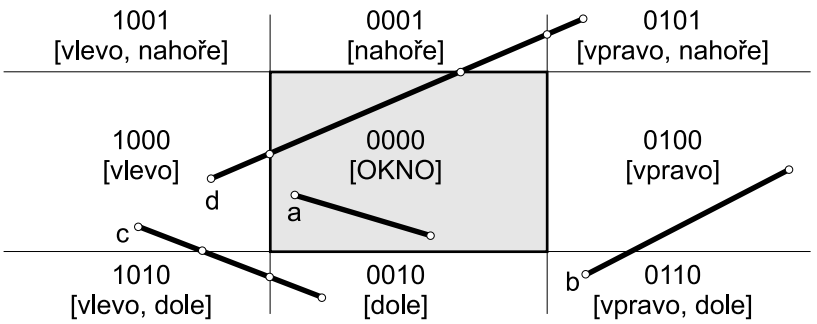
Při ořezávání se používá test, který zjišťuje polohu bodu vůči hraniční přímce. Výsledkem testu je určení, zda bod leží ve vnitřní nebo vnější polorovině vymezené hraniční přímkou. Osově orientované okno umožňuje použít jednoduché testy porovnáním příslušné souřadnice s hraniční hodnotou. Tento způsob používá algoritmus Cohen-Sutherland. U obecně orientované hranice konvexního okna je vhodné použít rovnici hraniční přímky ve tvaru $F(x, y) = a.x + b.y + c = 0$ a testovat pozici bodu $P = [xp, yp]$ pomocí testu $F(xp, yp) > 0, = 0, < 0$. Tento test lze převést

na test ve vektorovém tvaru, který je použit např. u algoritmu Cyrus-Beck. U testu prochází obecně orientovaná hranice bodem F. Hraniční přímka je kolmá na normálový vektor $\vec{n}$, který směřuje dovnitř okna. P je testovaný bod. Oba druhy testu jsou uvedeny na obrázku 3.21. Pokud je hranice ořezávací oblasti zadána jako souvislý polygonální tah, pak můžeme použít testy popsané v části 22.3.8.

3.4.2 Ořezání úsečky

Algoritmus Cohen-Sutherland

Pro ořezání úseček oknem, jehož hrany jsou rovnoběžné s osami souřadnicového systému, se v praxi používá několik algoritmů. Algoritmus *Cohen–Sutherland* [Spro68] nejprve ohodnotí koncové body úsečky tzv. *hraničními kódy*. Jde o čtyřbitové informace popisující polohu bodu vůči oknu. Jednotlivé bity hraničního kódu (zleva doprava) určují, zda bod se nachází vlevo, vpravo, dole či nahoře. Tyto bity se snadno nastaví jako hodnota znaménkového bitu rozdílu odpovídající souřadnice bodu a hranice okna. Možné kombinace hraničního kódu odpovídají rozdělení roviny na devět oblastí, jak ukazuje obrázek 3.22.



Obrázek 3.22: Kódy devíti oblastí určených oknem

Algoritmus dále zkoumá tři možné polohy úsečky vůči oknu. Označme koncové body úsečky P a Q a jejich odpovídající hraniční kódy $Kód(P)$ a $Kód(Q)$. Pak mohou nastat tyto možnosti:

1. $Kód(P) \cup Kód(Q) = \emptyset$
   Úsečka je celá v oknu a není třeba ji ořezávat. To je případ úsečky a na obrázku 3.22.

2. $Kód(P) \cap Kód(Q) \neq \emptyset$
   Celá úsečka leží mimo okno a je možno ji z dalšího zpracování vyřadit. Tento test vyřadí nejen úsečky, jejichž koncové body leží ve stejné oblasti mimo okno (mají totožné kódy), ale i některé úsečky procházející více vnějšími oblasti. Příkladem je úsečka b

na obrázku 3.22. Úsečka c přitom představuje složitější případ, který nelze na základě tohoto kritéria odhalit.

Pozn.: Operace průnik se provádí po jednotlivých bitech hraničních kódů.

3. $Kód(P) \cap Kód(Q) = \emptyset$
Úsečka prochází několika oblastmi a je třeba ji oříznout. Takové jsou úsečky c a d na obrázku 3.22. Zvolíme koncový bod s nenulovým (neprázdným) hraničním kódem a podle nastavené jedničky v kódu vybereme hranici, pomocí které úsečku zkrátíme. Poté zopakujeme předchozí dva testy s aktualizovanými koncovými body.

   Při určování nového koncového bodu není třeba hledat průsečík dvou obecných přímek. Hraniční přímky jsou rovnoběžné se souřadnicovými osami a proto lze výpočet zjednodušit. Jedna z nových souřadnic bude totiž vždy totožná s mezní souřadnicí okna.

Při tomto postupu nastávají situace, kdy průběžně vypočítávané koncové body nejsou skutečnými koncovými body ořezané úsečky. U úsečky d může být například nejprve nalezen průsečík s pravou hranicí okna a teprve při dalších testech průsečík s horní hranicí okna. Obdobně je tomu u úsečky c. Nejprve je zkrácena a teprve poté vyloučena jako zcela vnější.

Parametrické ořezávání úseček – algoritmus Cyrus-Beck

Algoritmus Cyrus–Beck [Cyru78] vychází z parametrické rovnice přímky $P(t) = P_1 + (P_2 - P_1).t$, procházející body P_1 a P_2. Pro $t \in \langle 0, 1 \rangle$ je touto rovnicí definována úsečka P_1P_2. Při ořezávání potřebujeme určit, které body přímky $P(t)$ (úsečky P_1P_2) leží vně a uvnitř hranice konvexní oblasti. Algoritmus používá vektorový test s vnitřním normálovým vektorem hranice, který je uveden na obrázku 3.21 vpravo. Hranice prochází bodem F a $P(t)$ je testovaný bod. Pomocí skalárního součinu $\vec{n}.(P(t) - F)$ rozlišíme tři případy:

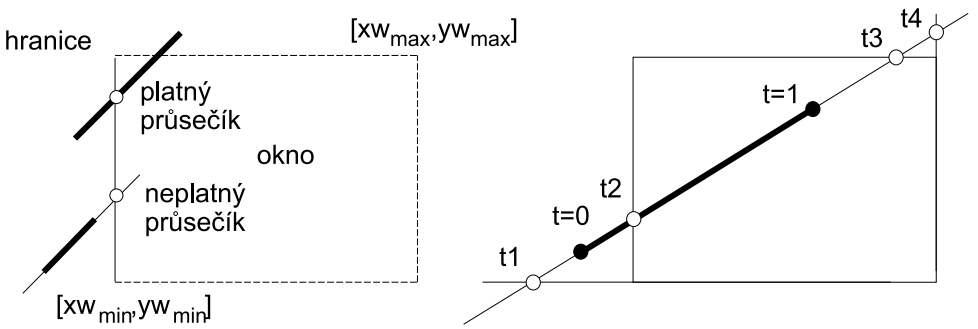
- $\vec{n}.[P(t) - F] < 0$ odpovídá bodu P_1, kdy vektor $[P(t) - F]$ směřuje z bodu F ven z oblasti
- $\vec{n}.[P(t) - F] = 0$ odpovídá bodu P_2, vektor $[P(t) - F]$ je rovnoběžný s hranicí
- $\vec{n}.[P(t) - F] > 0$ odpovídá bodu P_3, vektor $[P(t) - F]$ směřuje z bodu F dovnitř oblasti.

Po dosazení za $P(t)$ vypočteme průsečík pomocí rovnice

$$t = -\frac{\vec{n}.(P_1 - F)}{\vec{n}.(P_2 - P_1)}$$

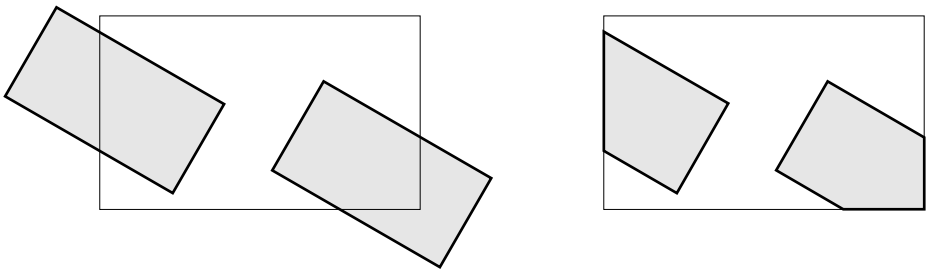
Označíme $\vec{w} = P_1 - F$ jako váhový faktor a $\vec{d} = P_2 - P_1$ jako směrový vektor. Algoritmus využívá dva rozhodovací členy. Pokud je skalární součin $\vec{d}.\vec{n}_i \neq 0$, pak existuje průsečík

s hraniční přímkou i-té hranice. V tomto případě je vypočten průsečík (parametr t) a následnými testy je tento průsečík přijat nebo odmítnut. Aby bylo možné nalézt koncové body ořezané úsečky při libovolném pořadí hraničních přímek, zahrnuje algoritmus nalezení minimálního (t_l) a maximálního (t_u) parametru. V příkladu uvedeném na obrázku 3.23 jsou vyloučeny vypočtené průsečíky t_3, t_4 a zůstává koncový bod s parametrem $t = 1$. Ve druhém směru je vyloučen průsečík t_1 a původní koncový bod s parametrem $t = 0$. Výsledkem je úsečka $P(t_2)P(1)$. Výpočetní postup je popsán v algoritmu 3.9.



Obrázek 3.23: Vyloučení neplatných průsečíků

Alternativní řešení parametrického ořezávání představuje algoritmus Liang-Barsky. Tento algoritmus ořezává úsečky podle osově orientovaného obdélníkového okna a využívá optimalizované testy polohy koncových bodů vzhledem ke hranici. Podrobnosti jsou uvedeny v [Lian83].



Obrázek 3.24: Ořezání polygonů

3.4.3 Ořezání polygonu

Ačkoliv algoritmus ořezání polygonů v sobě obsahuje ořezání hranic polygonu oknem, nelze jej provádět jednoduše po jednotlivých hraničních úsečkách polygonu. Tím by mohlo dojít

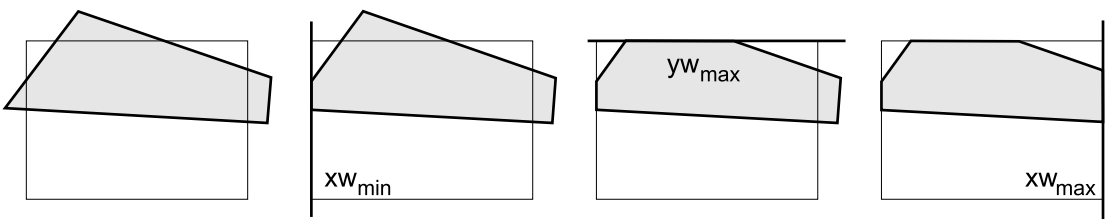
Inicializace: $t_l = 0,\ t_u = 1,\ \vec{d} = P(1) - P(0)$.
Pro všechny hranice i s vnitřní normálou $\vec{n}_i$ dělej:
  Pokud $\vec{d}.\vec{n}_i \neq 0$
    pak $t = -(\vec{w}_i.\vec{n}_i) \ / \ (\vec{d}.\vec{n}_i)$;
      Pokud $(\vec{d}.\vec{n}_i > 0)$ & $(t \leq 1)$
        pak $t_l = max(t, t_l)$ /* oprav průsečík t_l */
        jinak
      Pokud $(\vec{d}.\vec{n}_i < 0)$ & $(t \geq 0)$
        pak $t_u = min(t, t_u)$; /* oprav průsečík t_u */
    jinak
      Pokud $\vec{w}_i.\vec{n}_i < 0$
        pak Ukonči výpočet; /* úsečka je zredukována na bod mimo okno */

Pokud $t_l < t_u$
  pak OřezanáÚsečka$(P(t_l),\ P(t_u))$; /* jinak úsečka leží mimo okno */

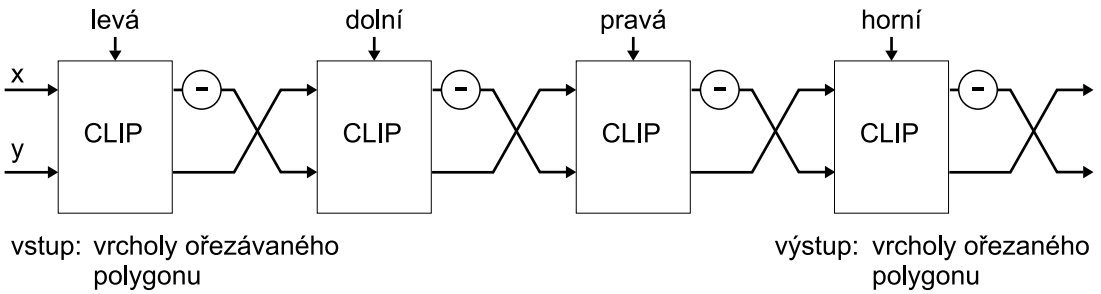
Algoritmus 3.9: Parametrické ořezávání úsečky – algoritmus Cyrus-Beck

k rozpadu hranice na samostatné, nenavazující úsečky a nebylo by pak například možno polygon vyplnit barvou. Algoritmus proto musí zajistit uzavřenost hranice polygonu případným doplněním nových úseček. Na obrázku 3.24 vlevo vidíme dva polygony, oba původně se čtyřmi hranami. Oříznuté polygony vpravo mají tři hrany, resp. pět hran.

Postup při ořezání polygonu obdélníkovým oknem lze logicky rozdělit do čtyř kroků – v každém s nich se provede ořezání jednou hranicí okna, jak ukazuje obrázek 3.25. Při implementaci je vhodné provést následující úpravy:



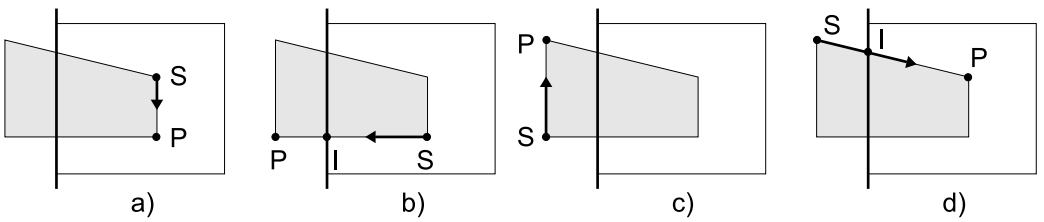
Obrázek 3.25: Postupné ořezání polygonu



Obrázek 3.26: Použití stejného kódu pro ořezání čtyřmi hranicemi s využitím otáčení o 90°

- Abychom nemuseli po dílčím ořezání jednou hranicí zaznamenávat (proměnlivý) počet vrcholů polygonu, je vhodné každou ořezanou hranu „zaslat“ okamžitě k ořezání další hranicí. Tento postup je anglicky nazýván *reentrant polygon clipping* a je znám jako *Sutherland–Hodgmanův* algoritmus [Suth74].
- Zjednodušení testů polohy bodu vůči hranici okna můžeme docílit tím, že budeme stále ořezávat svislou polorovinou. K tomu stačí, abychom vždy před ořezání další hranicí zaměnili souřadnice a jedno znaménko koncových bodů ořezávané úsečky. Tak provedeme otočení o 90° a můžeme použít stejný kód lišící se pouze jedním parametrem konkrétní hranice. Situace je naznačena na obrázku 3.26.

Sutherland–Hodgmanův algoritmus zpracovává v každém vnitřním kroku právě jeden vrchol P ořezávaného polygonu. Tento vrchol spolu s naposledy oříznutým předchozím vrcholem S tvoří elementární úsečku. Na obrázku 3.27 jsou znázorněny možné polohy úsečky SP vůči svislé hranici (vnitřek okna je vpravo od hranice). Průsečík úsečky s hranicí je označen jako I.

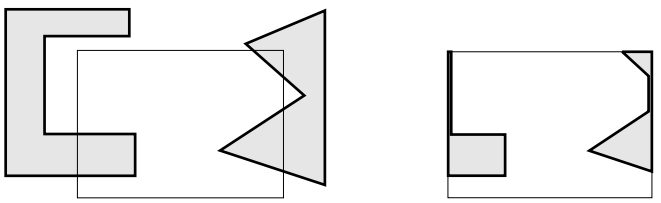


Obrázek 3.27: Test elementární úsečky SP vůči svislé hranici

V případě a) postupuje do dalšího zpracování (ořezání následující hranicí) aktuální bod P. V případě b) je bod P nahrazen průsečíkem I, který je zaslán další hranici. Situace c) indikuje,

Inicializuj bod S = $[x_S,\ x_S]$ souřadnicemi posledního vrcholu polygonu.
Pro vrcholy $P_i = [x_{Pi},\ y_{Pi}]$ $(i = 1, ..., n)$ ořezávaného polygonu postupně proveď:
Pokud $x_{Pi} > xw_{min}$
pak Pokud $x_S > xw_{min}$
pak Předej P_i další hranici k ořezání /* viz obrázek 3.27a) */
jinak Vypočti průsečík I jako $[xw_{min},\ y_S + (xw_{min} - x_S)\ \frac{y_{Pi}-y_S}{x_{Pi}-x_S}]$
Předej I další hranici k ořezání /* viz obrázek 3.27b) */
Předej P_i další hranici k ořezání
jinak
Pokud $x_S > x_{wmin}$
pak Vypočti průsečík I jako $[xw_{min},\ y_S + (xw_{min} - x_S)\ \frac{y_{Pi}-y_S}{x_{Pi}-x_S}]$
Předej I další hranici k ořezání /* viz obrázek 3.27d) */
Aktualizuj S jako P_i

Algoritmus 3.10: Zahájení zpracování polygonu jeho ořezáním levou hranicí

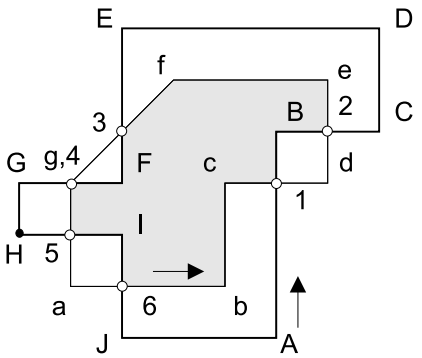


Obrázek 3.28: Některé nezvyklé výsledky algoritmu ořezávání polygonu

že další hranici není předán žádný bod a naopak v případě d) jsou následující hranici zaslány dva body: I a P. Právě poslední uvedený případ zajišťuje zvýšení počtu hran v ořezaném polygonu (viz obr. 3.24). Popsaná metoda je uvedena v algoritmu 3.10. Pokud je zpracováván *nekonvexní* polygon, může výslednou oblast po ořezání tvořit několik samostatných polygonů propojených hranami (obr. 3.28). Stejně tak vlivem pevného pořadí, ve kterém se provádí postupné ořezání hraničními poloprostory, mohou vzniknout některé nadbytečné úsečky. Tyto anomálie většinou nepůsobí rušivě, neboť „nepatřičné“ úsečky leží pouze na hranicích okna (na obrázku 3.28 jsou ovšem pro názornost kresleny vedle sebe) a na obrazovce nejsou vidět. Algoritmy pro rasterizaci vyplněných oblastí (viz kap. 3.3) jejich kresbu většinou potlačí. Pokud ovšem chceme získat „čistě“ ořezaný polygon bez výše uvedených nadbytečných úseček, musíme po algoritmu ořezání provést netriviální analýzu nové hranice a nevhodné úsečky odstranit.

Algoritmus Weiler-Atherton

Dosud uvedené metody byly určeny pro ořezávání konvexním, nejčastěji obdélníkovým oknem. Pro obecné ořezávání nekonvexních polygonů s vnitřními dírami nekonvexním oknem se používá algoritmus Weiler–Atherton [Weil77].



Obrázek 3.29: Obecné ořezávání polygonů

Okno i polygon jsou reprezentovány seznamem smyček – orientovanými uzavřenými posloupnostmi hran tvořících hranice. Orientace určuje vnitřek ohraničené oblasti. Vnitřní smyčky, nakreslené s opačnou orientací než má vnější hranice, určují díry v oblasti. Základem algoritmu je nalezení všech průsečíků mezi dvěma množinami úseček v rovině. Pro vysvětlení použijeme příklad (bez vnitřních smyček) na obrázku 3.29. Okno je tvořeno vrcholy $a-b-c-d-e-f-g-a$, polygon vrcholy $A-B-C-D-E-F-G-H-I-J-A$. Průsečíky obou polygonálních útvarů jsou body $1, 2, 3, 4, 5, 6$. Algoritmus Weiler-Atherton využívá tří seznamů – vrcholy ořezávaného polygonu, vrcholy okna a pomocný seznam průsečíků. Neformálně popsaný algoritmus (pro polygony bez vnitřních děr) je uveden jako algoritmus 3.11.

P: seznam vrcholů polygonu, W: seznam vrcholů okna, H: seznam průsečíků
1. Do H ulož průsečíky mezi hranicemi polygonu a okna.
   Průsečíky zařaď mezi vrcholy v seznamech P a W a propoj obousměrnými ukazateli.
2. Dokud H není prázdný opakuj
   Vyjmi průsečík ze seznamu H.
   Podle zvolené orientace přejdi do seznamu P nebo W.
   Opakuj
     Opakuj
     Výstup vrcholu nebo průsečíku ze seznamu (P nebo W) do nalezení dalšího průsečíku;
     Přejdi do druhého seznamu (W nebo P).
   dokud není uzavřen tah návratem do výchozího průsečíku.

Algoritmus 3.11: Ořezávání nekonvexních polygonů – algoritmus Weiler-Atherton

Projdeme-li seznamy v příkladu na obrázku 3.29 podle alg. 3.11, nalezneme v seznamu průsečíků H první průsečík (např. 1) a ořezaný polygon bude určen vrcholy 1-B-2-e-f-3-F-4-5-I-6-b-c-1. Pokud vhodně změníme směr procházení seznamy, můžeme získat nejen všechny odstřižené části původního polygonu, ale i části okna, které polygon nevyplňuje. Začneme-li např. u průsečíku 6 procházet opačným směrem, pak obdržíme polygon 6-J-A-1-c-b.

Kapitola 4

Úpravy obrazu

V této kapitole se budeme věnovat operacím s rastrovým obrazem. Uvedeme rozličné algoritmy, které zpracovávají dvojrozměrné pole pixelů s cílem transformovat je geometricky či barevně, zvýraznit detaily či naopak vyhladit ostré přechody.

Nejprve uvedeme algoritmy pro omezení počtu barev, používané zejména při přípravě obrazu pro tisk, ale potřebné i při převodu obrazů z jednoho formátu na jiný. Speciálním případem omezení počtu barev je zpracování obrazů s vysokým dynamickým rozsahem. V další části se zaměříme na efektivní postupy provádění lineárních transformací rastrového obrazu (změnu velikosti a otáčení) i na nelineární transformace, které jsou známé pod názvem *warping* a *morfing*. Seznámíme se i s dalšími užitečnými úpravami, jakými jsou například odstranění šumu či ostření obrazu. Všimneme si exotických postupů, jimiž lze obraz přetvořit tak, aby získal vzhled typický pro některý ze známých malířských směrů.

Poznamenejme, že způsoby zpracování diskrétního obrazu mají často společný základ s metodami používanými v oblasti počítačového vidění. Hlavním cílem počítačové grafiky však v tomto případě není nalezení objektů v obrazu, nýbrž vytvoření aparátu, s jehož pomocí může uživatel měnit obraz podle svých představ.

4.1 Transformace barev

Změna barevné charakteristiky obrazu je často prováděnou operací. Transformací barev můžeme tmavému obrazu dodat jas, nevýraznému kontrast, a docílit celkových či jen lokálních změn vzhledu obrazu.

Požadavek transformace barev přitom nevychází pouze z potřeby „vylepšování“ obrazu, ale je nutný například při přípravě obrazu před tiskem. Tehdy obraz upravujeme s ohledem na technologii barevného tisku a často snižujeme celkový počet použitých barev.

4.1.1 Omezení barevného prostoru

V kapitole 1 jsme uvedli, že počet barevných odstínů, které se mohou vyskytovat v rastrovém obrazu, dosahuje velmi vysokých hodnot. Často používané barevné rozlišení *true color* představuje více než 16 milionů různých barevných odstínů. V řadě případů je vhodné počet barev v jednom obrazu snížit. K tomuto kroku nás může vést jak snaha zmenšit velikost souboru, ve kterém je obraz uložen, tak potřeba připravit obraz pro tisk na barevné nebo černobílé tiskárně. Jindy zase chceme obraz vykreslit na obrazovce počítače, který pracuje v režimu zobrazování 256 barev zapsaných v tzv. *paletě*.

Obrázek 4.1: Fotografie Sarah Bernhardtové v 256 stupních šedi posloužila jako vstup pro dále uváděné algoritmy

Snížením počtu barevných odstínů v obrazu dochází ke ztrátě informace. Počítačová grafika disponuje metodami, které tuto ztrátu minimalizují. Techniky, které dokáží z několika barev vytvořit iluzi bohaté barevné škály, se nazývají *polotónování* (*halftoning*) a *rozptylování* (*dithering*). Pojem polotónování je používán především u tiskáren, kde je jeden barevný pixel původního obrazu převeden na matici bodů s výrazně méně barvami. Dochází tedy ke zvětšení rozlišení obrazu.

Rozptylovací metody jsou vhodné k zobrazení obrazu, který byl např. vypočítán v kvalitě 24 bitů na pixel a který má být vykreslen na obrazovce v nezměněné velikosti. V této kapitole se zaměříme především na techniku rozptylování – polotónování budeme chápat jako její speciální případ.

Obě metody využívají schopnosti lidského oka vytvářet z několika blízkých barevných bodů vjem jediného bodu, jehož barva je dána *aditivním složením* barev původních bodů. Tato vlastnost je základem všech metod počítačové grafiky pro generování obrazů o mnoha odstínech na zařízeních s omezenou velikostí palety. Tato technika je využívána i v barevné televizi, ještě dříve ji však používal umělecky směr zvaný impresionismus.

Rozptylovací techniky se používají při zpracování jak černobílých, tak barevných obrazů. V dalším textu vysvětlíme nejprve jejich základní principy na převodu šedotónového obrazu na černobílý. Vzhledem k tomu, že zpracování barevných obrazů se ve většině případů provádí odděleně po jednotlivých barevných složkách, lze barevné rozptylování chápat jako speciální případ rozptylování obrazu ve stupních šedi.

Předpokládejme, že chceme obraz obsahující 16 ($= 2^4$) odstínů šedi vykreslit na displeji, který zobrazuje pouze černou a bílou. Původní barevnou kvalitu 4 bity na pixel je nutno snížit na 1 bit na pixel tak, aby si lidské oko dokázalo z různých kombinací sousedních bílých a černých bodů vytvořit představu několika odstínů šedi. Střídání černých a bílých bodů musí brát ohled na vstupní odstín šedi. Výstupní obraz má stejné rozměry (rozlišení) jako vstupní obraz. Existuje několik základních metod:

- *náhodné rozptýlení*,
  vyžadující pro zpracování pouze původní velikost intenzity zpracovávaného pixelu,
- *pravidelné (maticové) rozptýlení*,
  využívající k výpočtu také souřadnice vstupního pixelu,
- *distribuce zaokrouhlovací chyby*,
  při níž je nutno zpracovávat vstupní pixely v určitém pořadí, zpravidla po řádcích.

V následujících odstavcích popíšeme všechny uvedené postupy. Jako C_{in} budeme označovat vstupní intenzitu (odstín šedi) jednoho obrazového bodu. Její hodnota se bude pohybovat v intervalu celých čísel $\langle 0, C_{MAX} \rangle$. Obdobně označíme výstupní intenzitu $C_{out} \in \{0, 1\}$. Hodnota 0 reprezentuje černou barvu.

Náhodné rozptýlení

V této metodě využijeme generátor náhodných čísel, reprezentovaný funkcí $random(x)$ s rovnoměrným rozložením. Tato funkce vrací celočíselné náhodné hodnoty v intervalu $\langle 0, x)$. K určení výsledné intenzity C_{out} černobílého obrazového bodu použijeme jednoduchý postup zapsaný v algoritmu 4.1.

1. $C_{out} = 0$
2. Pokud $(C_{in} > random(C_{MAX}))$
   $C_{out} = C_{out} + 1$

Algoritmus 4.1: Zpracování intenzity pixelu při náhodném rozptýlení

Obrázek 4.2: Převod šedotónového obrazu na černobílý. Zleva doprava náhodné rozptýlení, rozptýlení maticí typu $\mathbf{M}_d$, rozptýlení maticí typu $\mathbf{M}_p$ a distribuce chyby podle Floyd–Steinberga.

O tom, zda výsledný pixel bude rozsvícen nebo ne, rozhoduje výsledek porovnání velikosti vstupní intenzity a náhodně vygenerovaného čísla. To se ve výsledném obrazu projeví tím, že například plocha o vstupní intenzitě $2/3\ C_{MAX}$ bude tvořena nepravidelně se střídajícími bílými a černými pixely. Poměr počtu bílých pixelů k černým bude přibližně dvě třetiny, jak to odpovídá poměru $C_{in} : C_{max}$. Tímto způsobem zůstanou zachovány původní jasové poměry v celém obrazu.

V posledním řádku algoritmu 4.1 není do proměnné C_{out} přímo přiřazena maximální hodnota výstupní intenzity, ale je použito zvýšení hodnoty o jedničku. Tato konstrukce je důležitá v situaci, kdy výstupní škála barev obsahuje více než dvě barvy. Tehdy se C_{out} nastavuje v kroku 1 algoritmu 4.1 na hodnotu, určenou kvantováním vstupní škály barev (viz také část 2.1.1) a tato hodnota je v dalším kroku případně zvýšena o jedničku.

Náhodné rozptýlení je jednoduchou metodou zachovávající původní počet pixelů obrazu. Dodává větším plochám vzhled drsného povrchu, výsledné obrazy působí dojmem fotografie s velkým zrněním.

Maticové rozptýlení

Maticová rozptylovací metoda používá pro generování různých odstínů šedi pravidelných, předem daných vzorků složených z bílých a černých bodů. Pokud jsou tyto vzorky dobře navrženy, působí výsledný obraz jako kresba, ve které malíř tužkou vystínoval tmavší, či světlejší místa pravidelnými jemnými šrafami.

Původně byla tato metoda navržena pro takový převod obrazů, při kterém je jeden vstupní pixel nahrazen skupinou (maticí) pixelů výsledných. Dojde tedy ke *zvětšení obrazu*. Pro další výklad použijeme záměrně zjednodušení, které neodpovídá v praxi používaným numerickým hodnotám, ale dovolí nám ukázat princip metody na maticích malé velikosti. Budeme předpokládat, že rozsah vstupních intenzit je v intervalu $\langle 0, 4\rangle$. Připravíme tedy pět matic, kterými na výstupu nahradíme vstupní pixely a vytvoříme tak černobílý obraz s dvakrát většími rozměry:

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$C_{in} = 0 \quad C_{in} = 1 \quad C_{in} = 2 \quad C_{in} = 3 \quad C_{in} = 4$$

Všimněme si, že v posloupnosti těchto matic se každá následující matice liší od předchozí přidáním jedničky na jednu novou pozici. Toto pravidlo je důležité, neboť permutace jedniček a nul uvnitř matic způsobuje vznik artefaktů v případě, kdy spolu sousedí pixely s různými vstupními odstíny. Bylo by tedy např. chybou použít pro vstupní intenzitu $C_{in} = 2$ matici

ve tvaru $\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$. Rozdíl v jediné pozici mezi sousedními maticemi navíc umožňuje zapsat celou výše uvedenou posloupnost pěti matic úsporně do jediné matice $\begin{bmatrix} 0 & 3 \\ 2 & 1 \end{bmatrix}$, jejíž hodnoty pak určují, při jaké velikosti C_{in} se na výstupu objeví v odpovídajícím místě jednička. Pokud je C_{in} větší než daná hodnota v matici, bude výstupní pixel rozsvícen. Čísla použitá v této matici musejí být v rozsahu od 0 do $C_{MAX} - 1$.

Čím větší je rozsah vstupních intenzit, tím větší potřebujeme rozptylovací matici. Čísla v matici lze uspořádat různými způsoby. Příkladem jsou následující matice řádu 4×4:

$$\mathbf{M}_d = \begin{bmatrix} 0 & 12 & 3 & 15 \\ 8 & 4 & 11 & 7 \\ 2 & 14 & 1 & 13 \\ 10 & 6 & 9 & 5 \end{bmatrix}, \qquad \mathbf{M}_p = \begin{bmatrix} 1 & 5 & 9 & 2 \\ 8 & 12 & 13 & 6 \\ 4 & 15 & 14 & 10 \\ 0 & 11 & 7 & 3 \end{bmatrix}.$$

Pro různé typy zobrazovacích zařízení (displej, tiskárna) se používají různá uspořádání čísel v matici. Ve výše uvedeném příkladu je matice $\mathbf{M}_d$ vhodná pro displeje, kde vytváří „křížkový“ vzor podobný šrafování – rostoucí posloupnost čísel je do matice umísťována v diagonálním smyslu. Jiné uspořádání je vhodné pro tiskárny, kde je žádoucí sdružit výstupní černé, resp. bílé body do „puntíků“. Takový vzor lze vygenerovat pomocí matice $\mathbf{M}_p$. V ní jsou postupně rostoucí čísla kladena do zavírající se spirály, což vytvoří jak oblast (puntík) uprostřed vzoru, tak další oblasti v rozích vzorů, kde se opticky spojují se sousedními vzory. Důvodem pro sdružování bodů do větších puntíků je skutečnost, že vytištěné body se v rastru částečně překrývají, aby bylo zajištěno, že při tisku 100 % černé plochy nezůstane na papíře žádné bílé místo. Každý černý bod tak zabere i část plochy patřící sousednímu (bílému) bodu. Kdybychom například při tisku pravidelně střídali bílé a černé body jako na šachovnici, získáme na tiskárně velmi tmavý vzorek, nikoliv teoreticky předpokládanou „poloviční šeď“. Sdružování bodů tento problém překlene. Pozitivním vedlejším efektem je zvýšení robustnosti tisku – pokud se inkoust na papíře rozpíjí, u puntíků se to projeví menším poklesem kvality obrazu ve srovnání s použitím maticí $\mathbf{M}_d$.

Při barevném tisku se používají různé matice pro různé barevné složky, přičemž jednotlivé vzory se ještě natáčejí, aby se potlačila opticky zřetelná pravidelnost vzorků v maticích a minimalizoval vliv rozpíjení barevných inkoustů. Pro černou barvu, která je výrazná, se často používá natočení jinak pravidelného vzorku o optimální úhel 45°. Pro fialovou se volí úhel natočení 75°, pro žlutou 90° a pro modrozelenou 105°.

Je zajímavé, že rozptylovací matici pro displeje lze generovat jednoduchým algoritmem pro různé velikosti matice. Předpokládáme, že řád matice je mocninou dvou. Matici řádu $(2n)$ vytvoříme z matice řádu (n) jako

$$\mathbf{M}_{(2n)} = \begin{bmatrix} 4\mathbf{M}_{(n)} & 4\mathbf{M}_{(n)} + 3\mathbf{J}_{(n)} \\ 4\mathbf{M}_{(n)} + 2\mathbf{J}_{(n)} & 4\mathbf{M}_{(n)} + \mathbf{J}_{(n)} \end{bmatrix},$$

kde $\mathbf{J}_{(n)}$ řádu (n) obsahuje samé jedničky. $\mathbf{M}_{(1)}$ je jednoprvková matice obsahující nulu.

Pro nejčastěji používaný rozsah stupňů šedi (256 jasových hodnot) používáme matice řádu 16×16. Všimněme si, že každá rozptylovací matice určuje o jedničku více vzorů, než je počet jejích prvků. Matice s 256 prvky tak generuje 257 vzorů, podobně jako čtyřprvková matice použitá na začátku tohoto výkladu generovala vzory pro pět vstupních intenzit. Abychom zabezpečili shodu počtu vstupních intenzit s počtem vzorů, upravíme matice tak, aby žádná z nich neobsahovala hodnotu C_{MAX}, tj. v našem případě číslo 255. Toto číslo tedy z matice vypustíme a naopak některé z nižších čísel, např. v polovině intervalu vstupních intenzit, do matice umístíme dvakrát.

Metoda maticového rozptýlení se používá často i v případě, kdy *nezvětšujeme* výsledný obraz. Tehdy použijeme místo celé matice pouze jeden její prvek. Pro výběr prvku poslouží souřadnice $[x, y]$ zobrazovaného pixelu. Zbytková čísla po celočíselném dělení rozměry matice (operace *modulo*) představují indexy v matici. To je obdobný přístup jako při vyplňování oblasti vzorem (viz kap. 3.3, str. 99). Algoritmus 4.2 je navržen pro matici 4×4 s tím, že je indexována od nuly.

1. $C_{out} = 0$
2. Pokud $(C_{in} > \mathbf{M}[x \ mod \ 4, y \ mod \ 4])$
   $C_{out} = C_{out} + 1$

Algoritmus 4.2: Zpracování intenzity pixelu při maticovém rozptýlení

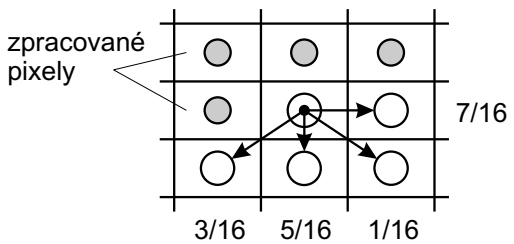
Maticové rozptýlení dává dobré výsledky pro většinu počítačem generovaných obrazů. Pravidelné střídání odstínů nepůsobí rušivě ani na velkých plochách. Při zpracování fotografií však tato pravidelnost dělá dojem uměle vytvořeného obrazu. Právě pro vykreslování obrazů získaných fotografickou cestou byly vyvinuty další metody, založené na zpracování chyby vzniklé zaokrouhlením vstupní intenzity.

Distribuce zaokrouhlovací chyby

Při zachování stejné velikosti obrazu je intenzita vstupního pixelu vždy nahrazena minimální nebo maximální hodnotou intenzity na výstupu, čímž nutně dochází ke ztrátě informace. Jeden ze způsobů, jak využít vstupní informaci v co nejvyšší míře, se nazývá *distribuce chyby* (*error*

diffusion či *error distribution*). V základní metodě zpracováváme vstupní pixely po řádcích, shora dolů. Když pro vstupní intenzitu C_{in} určíme pomocí prahování výstupní intenzitu C_{out}, zanedbanou hodnotu (chybu) vzniklou *zaokrouhlením* C_{in} si zapamatujeme a použijeme ji k modifikaci hodnot dalších pixelů na vstupu. Chyba se distribuuje jen mezi *sousední pixely*, které dosud nebyly zpracovány, tj. k následujícímu pixelu na stejném řádku a k pixelům na dalším řádku (obr. 4.3).

Chceme-li například při hodnotě $C_{MAX} = 15$ zpracovat pixel o vstupní intenzitě 5 a prahové hodnotě nastavené na 7, pak výstupnímu pixelu přiřadíme intenzitu 0, čímž se dopustíme chyby o velikosti 5. Tuto chybu rozdělíme ve zvoleném poměru a její části přičteme k intenzitám sousedních, dosud nezpracovaných pixelů. Obdobně pro vstupní intenzitu 11 vykreslíme výstupní pixel maximální intenzitou a chybu velikosti -4 $(= 11 - 15)$ distribuujeme do okolí. Jedna z možností dělení a distribuce chyby je uvedena na obrázku 4.3 a nazývá se Floyd–Steinbergova metoda distribuce chyby. Poměry, podle nichž se chyba dělí, byly navrženy empiricky a v praxi se můžeme setkat s několika variantami. Různé způsoby distribuce chyby se zpravidla nazývají podle autorů (Burkes, Floyd, Jarvis, Sierra, Stucki).



Obrázek 4.3: Rozdělení chyby okolním pixelům podle Floyd–Steinberga

Koeficienty udávající podíl chyby distribuovaný do okolních pixelů jsou pochopitelně neceločíselné. Původní Floyd–Steinbergův algoritmus byl navržen pro implementaci v celočíselné aritmetice. Aby nebyla celočíselným dělením zanedbána žádná informace, při rozdělování chyby z jednoho pixelu do jeho sousedů je třeba poslední díl distribuované chyby určit nikoliv dělením, nýbrž jako rozdíl celé chyby a součtu všech dříve vypočítaných dílů této chyby. Kromě Floyd–Steinbergova způsobu distribuce chyby se používají i metody spojené s výpočtem v pohyblivé řádové čárce [Knut87]. Alternativně se směr distribuce chyby mění po každém novém řádku a provádí se tedy „cik–cak“ (*zig–zag*) nebo se vstupní obraz prochází po speciálních trajektoriích, např. po tzv. Hilbertově křivce.

Různé rozptylovací metody použité na fotografii Sarah Bernhardtové z obrázku 4.1 jsou ukázány na obrázcích 4.2a–d. Vstupní digitalizovaný obraz měl 256 stupňů šedi.

4.1.2 Barevná paleta

Dosud jsme vysvětlili princip rozptylovacích metod na příkladech redukce šedotónových obrazů do černobílých. V barevném prostoru je situace analogická s následujícím rozšířením:

- *vstupní pixel* nese barevnou informaci v určitém barevném modelu, nejčastěji RGB. Ve

většině případů lze barevný pixel zpracovávat po jednotlivých složkách r, g, b. Rozptylovací algoritmus je tedy prováděn ve třech samostatných částech,

- *barevná škála* na výstupu bývá větší než u černobílé výstupní informace. Namísto dvouprvkové množiny možných výstupních hodnot se objevuje několikaprvková množina barevných intenzit, do kterých jsou vstupní barvy převáděny. Zaokrouhlování do víceprvkové škály barev znamená jen drobnou úpravu metody.

Pro přiblížení funkce rozptylovacích metod v barevném prostoru použijeme velmi časté situace, kdy vstupní barevné informace jsou reprezentovány v kvalitě 24 bitů/pixel a mají být převedeny do 8 bitů/pixel. Předpokládejme dále, že oněch 8 bitů/pixel je indexem do palety, ve které je možno nastavit 256 odstínů z 16 miliónů možných barevných kombinací.

Kromě volby rozptylovací metody se nyní objevuje i otázka vhodného *nastavení barevné palety*. V zásadě lze rozlišit palety dvojího druhu – přednastavené (univerzální) a přizpůsobené konkrétnímu obrazu. Přednastavená paleta je použitelná pro libovolný obraz. Škála barev v paletě přizpůsobené obrazu lépe postihuje barevné složení daného obrazu. K jejímu vytvoření je však potřeba nejprve obraz analyzovat a poté většinou výslednou paletu uložit spolu s novým obrazem. V následujících odstavcích ukážeme vlastnosti přednastavené palety označované jako 3–3–2 a popíšeme různé způsoby vytvoření palety přizpůsobené obrazu.

Barevná paleta 3–3–2

Paleta, která má být univerzálně použitelná pro obrazy s různým barevným složením, musí zřejmě obsahovat pravidelně vybrané zástupce z barevné krychle RGB. Krychli však nemůžeme rozdělit ve všech třech osách stejnoměrně, protože pro zakódování jednotlivých barevných zástupců máme jen 8 bitů, které nelze rozdělit na tři stejné díly. Proto rozdělíme RGB krychli na 8 ($= 2^3$) řezů v ose r, 8 řezů v ose g a na 4 ($= 2^2$) řezy v ose b. Počet bitů pro kódování modré bude tedy nižší než pro ostatní dvě základní barvy. Toto omezení není příliš významné – lidské oko je na odstíny modré málo citlivé [viz vzorec (1.1)], a tak redukce stupnice modré z 256 možných na 4 odstíny ve většině případů nevadí.

Vrcholy takto vytvořené sítě řezů v krychli RGB představují 256 zástupců do palety, která se standardně nazývá *paleta 3–3–2*. Mezi 8bitovým obsahem pixelu a barevným složením položky palety je jednoznačný vztah. V následujících vzorcích, určených pro programátory, použijeme konvence programovacího jazyka C: symboly „>>“ a „<<“ znamenají bitové posuvy, symbol „&“ logický součin. Dělení je celočíselné.

Má-li tedy pixel hodnotu i, pak i-tá položka palety obsahuje následující velikosti složek r, g a b (v rozsahu 0, 1, . . . , 255):

$$\begin{array}{rcl} r & = & (((i \;\; >> 5) \;\;\;\;\;\;\;\;) \;\; * 255) \;/\; 7, \\ g & = & (((i \;\; >> 2) \;\; \& \; 7) \;\; * 255) \;/\; 7, \\ b & = & (((i \;\;\;\;\;\;\;\;) \;\; \& \; 3) \;\; * 255) \;/\; 3. \end{array} \tag{4.1}$$

Prvé tři bity každého pixelu obsahují kódy 0–7 pro 8 stupňů červené, jimž v paletě 3–3–2 odpovídají velikosti červené složky v plném rozsahu 0–255. Vzorec (4.1) je vlastně předpisem pro nastavení obsahu palety 3–3–2.

Pro účely rychlého orientačního kreslení obrazu je takto zvolená paleta postačující již sama o sobě. Vstupní barevnou informaci stačí převést z rozsahu 8–8–8 bitů na příslušné 3–3–2 bity. Tento převod lze provést v celočíselné aritmetice. Složku r_8 z 8 bitů převedeme do rozsahu 3 bitů na hodnotu r_3 podle vzorce:

$$r_3 = (r_8 \;\; * 7) \;/\; 255 \tag{4.2}$$

a výsledný index I v paletě získáme bitovým složením hodnot redukovaných barevných složek

$$I \;\; = (r_3 << 5) + (g_3 << 2) + b_2. \tag{4.3}$$

Jedná se vlastně o jednoduché zaokrouhlení rozsahu barvy do 256 předem zvolených hodnot, tedy o kvantování barev (viz též část 2.1.1). Kvalitnějšího vzhledu obrazu dosáhneme, použijeme-li některé z rozptylovacích technik dříve uvedených pro černobílá zobrazovací zařízení. Tehdy vezmeme do úvahy i onu *chybovou hodnotu*, která se ztrácí při převodu z 8 bitů na nižší počet. Tuto hodnotu budeme značit r_{rest} (pro červenou složku). Jednotlivé barevné složky jsou rozptylovány samostatně a index výsledné barvy vznikne složením podle vzorce (4.3).

1. Pomocí vzorce (4.2) najdi nejbližší barevnou hodnotu existující v paletě 3–3–2. Ke vstupní intenzitě r_8 tedy nalezni intenzitu r_3.
2. Urči chybu (např. $r_{rest} \in \langle 0, 31 \rangle$ pro červenou složku), která vznikla zaokrouhlením vstupní barvy do nižšího výsledného rozsahu.
3. Podle použité rozptylovací metody (náhodné nebo maticové rozptýlení) použij chybu r_{rest} k případnému zvýšení výsledné intenzity r_3 o jedničku.

Algoritmus 4.3: Zpracování jedné barevné složky pixelu při rozptýlení

Postup je uveden v algoritmu 4.3. U metody distribuce chyby je řešení obdobné, využívá se však navíc tří pomocných pamětí chyby pro jednotlivé barevné složky.

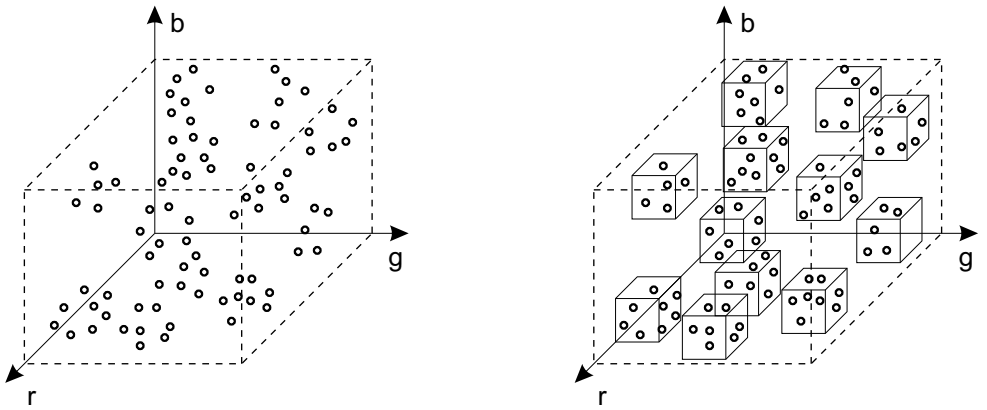
Vedle vskutku univerzální palety 3–3–2, která vznikla s cílem rozdělit barevný prostor rovnoměrně, se můžeme setkat i s přednastavenými paletami, které byly navrženy na základě jiných kritérií. Některé palety jsou vytvořeny statistickým vyhodnocením barevného složení velkého množství obrazů z určité aplikační domény. Jiné přednastavené palety obsahují barvy preferované určitým programem (např. webovým prohlížečem) nebo zvolené uživatelem při přizpůsobení pracovní plochy jeho estetickému cítění.

Paleta přizpůsobená obrazu

Pevně daná paleta typu 3–3–2 dovoluje použít rozptylovacích technik, aniž by byla předem známa barevná struktura celého obrazu. Při požadavku na co nejvyšší kvalitu obrazu je taková paleta nevýhodná, protože často obsahuje i takové odstíny, které se na obrazu vůbec nevyskytují, případně jsou zastoupeny v minimálním počtu. Je tedy vhodné umět vytvořit paletu, která bude odpovídat rozložení barevných odstínů v konkrétním obrazu. Jakmile je paleta „ušita na míru“ danému obrazu, je obraz převeden do tvaru využívajícího tuto paletu a spolu s ní uložen. S tímto přístupem se setkáváme např. u obrazů kódovaných ve formátu GIF.

Nalezení palety pro daný obraz vyžaduje jak čas, tak paměť. V prvé fázi je vytvářen *histogram* (viz též část 4.5) barevných odstínů obsažených v celém obrazu. Pro další výpočty je důležitý nejen fakt, že se určitý odstín v obrazu objevuje, ale i to, jak často je v něm obsažen. Na rozdíl od jiných technik pro úpravu rastrových obrazů nelze složky barev separovat a histogram uchovávat pomocí tři jednorozměrných polí pro složky r, g a b, ale je třeba jej uložit do trojrozměrného pole. Pro zpracování barev uložených v 3×8 bitech potřebujeme pole s 256^3 položkami, tedy paměťový prostor pro uložení více než 16 milionů hodnot četností barev. Alokovat tak velkou paměť by bylo neefektivní, praktičtější je snížit rozsah barev kvantováním. Obvykle můžeme zanedbat tři nejméně významné bity každé barevné složky a snížit tak barevné rozlišení vstupních barev na 3×5 bitů. Pro histogram pak vystačíme s trojrozměrným polem s 32^3 položkami.

V další části je potřeba na základě histogramu nalézt tolik oblastí, kolik má mít vytvářená paleta barev, typicky tedy 256. Zde nám opět poslouží reprezentace barevného prostoru jako krychle RGB (obr. 4.4 vlevo), ve které jsou použité barvy znázorněny tečkami. Za každou tečkou si představme číslo, udávající počet pixelů, které mají odpovídající barvu. Naším úkolem je nalézt oblasti (obr. 4.4 vpravo), které obklopují skupiny blízkých barev [Heck82]. Algoritmy shlukování, pracující metodou zdola nahoru, nejsou v tomto případě efektivní. Používají se techniky nazývané *zmenši a rozděl* (*shrink &split*), které postupně rozdělují prostor barev na menší oblasti, viz algoritmus 4.4.



Obrázek 4.4: Histogram vstupních odstínů (vlevo) a oblasti blízkých barev (vpravo) při vytváření palety

Techniky, které vycházejí z algoritmu 4.4, se liší v určení polohy řezu v kroku 3(b) algoritmu a ve stanovení zástupců barev v paletě v kroku 4.

Při hledání řezné roviny lze brát do úvahy různá kritéria. Přímočarým řešením je rozdělit oblast v geometrické polovině. Další možností je dělit s cílem, aby vzniklé oblasti obsahovaly stejný počet (různých) odstínů. Patrně nejlepší přístup bere do úvahy četnosti barev a volí řez tak, aby nově vzniklé oblasti reprezentovaly stejný počet pixelů. Tato metoda, označovaná jako *median cut*, tedy vede řez bodem, který má hodnotu mediánu ze všech pixelů oblasti. Medián je střed seřazené posloupnosti hodnot. Nechť například daná oblast reprezentuje dva pixely s barvou se složkami [21, 0, 42], tři pixely barvy [43, 0, 125], jeden pixel s barvou [67, 0, 196] a pět pixelů barvy [255, 0, 0]. Když budeme oblast dělit v ose r, medián určí předposlední z barev, neboť seřazená posloupnost obsahuje čísla (21, 21, 43, 43, 43, 67, 255, 255, 255, 255, 255). Při dělení v ose b by byla pomocí mediánu vybrána první barva.

Každá oblast nalezená algoritmem 4.4 bude reprezentována právě jednou položkou v paletě (krok 4 algoritmu). Její konkrétní barvu stanovíme buď jako geometrický střed oblasti, jako aritmetický nebo vážený průměr barev pixelů v dané oblasti, nebo jako tu barvu, která se v oblasti vyskytuje nejčastěji. Pokud byl při dělení veden řez přímo nějakou barvou, tato barva musí ovlivnit obě oblasti, na jejichž stranách leží. Zahrneme ji tedy do průměru při výpočtu obou odpovídajících položek palety. Pokud byl pro volbu řezu použit medián, výsledkem je téměř rovnoměrné přiřazení barev v paletě – každá její položka je přiřazena přibližně stejnému počtu pixelů.

Stanovením palety pro daný obraz práce nekončí. Pixely vstupního obrazu je nutno převést na indexy položek v paletě. Pro každou vstupní barvu pixelu musíme nalézt nejbližší barvu

1. Urči iniciální oblast jako osově orientovaný ohraničující kvádr (obálku) všech barev použitých v obrazu.
2. Počet oblastí nastav na 1.
3. Dokud nedosáhne počet oblastí požadované velikosti palety:
   (a) Nalezni oblast s největším rozměrem v jedné z os r, g, b.
   (b) Podle zvoleného kritéria rozděl tuto oblast řezem kolmým na vybranou souřadnicovou osu (*split*).
   (c) Aktualizuj obálky obou nových oblastí vzniklých rozdělením (*shrink*).
   (d) Zvyš počet oblastí o jedničku.
4. Každou z nalezených oblastí reprezentuj jednou barvou, kterou ulož do palety.

Algoritmus 4.4: Hledání oblastí pro určení barev v paletě

v nově vytvořené v paletě. Základním způsobem je přímé přiřazení indexu nalezené barvy do každého pixelu. To je podobné jednoduchému prahování barev. Žádáme-li co nejkvalitnější obraz, použijeme metody rozptylování. Vhodnou technikou je *distribuce chyby*, při níž využijeme rozdíl mezi vstupní barvou a existující (nejbližší) barvou v paletě.

Při hledání nejbližší barvy v paletě je třeba porovnat barevnou vzdálenost mezi vstupním pixelem a každou z položek palety. Pro urychlení této úlohy lze využít datových struktur, pomocí nichž jsme v algoritmu 4.4 hledali barevné oblasti v RGB prostoru a jejichž obsah jsme si uchovali v paměti. Vhodnou strukturou je např. binární strom popisující rekurzivní dělení RGB krychle na menší oblasti. Tento strom umožňuje nalézt nejbližší barvu s logaritmickou složitostí. Šikovným trikem je také přepsání všech položek barevného (prostorového) histogramu. Tyto položky dosud obsahovaly počet vstupních barev, nyní do nich zapíšeme indexy položek z nově vytvořené palety. Hledání nejbližší barvy je pak redukováno na pouhé „vyzvednutí“ indexu barvy z tabulky, reprezentující histogram.

Samotná kombinace hledání palety pomocí stejně velkých oblastí a následná distribuce chyby nemusí vždy dávat dokonalé výsledky. Příkladem je zpracování posloupnosti sousedních pixelů, jejichž vstupní odstíny jsou všechny umístěny na „světlejším okraji“ téže oblasti histogramu. Při rozptylování je barva každého z těchto pixelů nahrazena tmavším odstínem z palety (představovaným bodem uvnitř oblasti histogramu) a jas sousedních pixelů je distribucí chyby zvýšen. Dále zpracovávané sousední pixely získají akumulováním chyby tak světlý odstín, že pro něj nemusí v paletě existovat žádná vhodná položka. Tuto situaci vyřešíme zařazením

všech osmi vrcholů základní barevné krychle RGB mezi položky palety. Tyto položky hrají roli „nárazníku“ pro výše uvedený příklad příliš velké akumulace chyby.

Významným faktorem, který jsme zatím do tvorby palety nezařadili, je charakter vnímání barev lidským okem. Určitého zlepšení pro některé obrazy dosáhneme, vytváříme-li v algoritmu 4.4 barevné oblasti nikoliv dělením nejdelší hrany [krok 3(a)], ale dělením, které vyjadřuje citlivost lidského vnímání barev. Znamená to, že cílem dělení jsou kvádry s délkami stran R, G, B v poměru 3,3 : 1,7 : 10 [převrácené hodnoty koeficientů ze vztahu (1.1)]. V této souvislosti lze také přejít do jiného barevného prostoru, např. HSV nebo HLS a vytvářet oblasti vhodných vlastností v něm [Fole90].

Rozptylování barev je operací, jejíž výsledek často závisí na subjektivním hodnocení uživatele. Je zajímavé, že rozdíly mezi původním obrazem a obrazem s redukovaným počtem barev vyjádřené *střední kvadratickou odchylkou* (*RMS error*) nemají velkou vypovídací hodnotu. Z řady pozorování vyplynulo, že hezčí obrázky mají dokonce větší odchylky od originálu než ty, které byly označeny jako „nepříliš věrné“, byť numericky se originálu blížily více.

4.2 Obrazy s vysokým dynamickým rozsahem

Rozsah intenzity světla, se kterým se setkáváme v reálném světě, je značný. Za svitu hvězd jsme schopni v lese rozlišit temné stíny, které mají jas[1] pouze cca $10^{-5}\mathrm{cd/m^2}$, zatímco sluncem ozářená závěj sněhu může emitovat až $10^{5}\mathrm{cd/m^2}$. Představu o světelných poměrech ve scéně udává *dynamický rozsah* (*dynamic range*) scény, který je definován jako rozsah jasů přítomných ve scéně. Dynamický rozsah se vyjadřuje ve formě poměru nejvyššího ku nejnižšímu jasu v dané scéně. Plošně osvětlená pláž u moře má dynamický rozsah cca 15 : 1, sluncem osvětlená krajina s částí temného lesa a hlubokými stíny cca 200 : 1, oknem ozářený interiér kostela cca 600 : 1, scény obsahující přímo viditelný zdroj světla mohou mít rozsah vyšší než 50 000 : 1.

Ve fotografické praxi se dynamický rozsah vyjadřuje v jednotkách EV (*Exposure Value*), jako rozdíl EV nejsvětlejšího a EV nejtmavějšího místa scény. Expoziční hodnota EV je absolutní veličina udávající velikost expozice (osvětlení).

Chceme-li věrný obraz reálné či syntetické scény zobrazit na běžných výstupních zařízeních, musíme se vypořádat s problémem jejich nedostatečného dynamického rozsahu. Klasické monitory mají dynamický rozsah cca 100 : 1 a výtisk z běžné tiskárny pouze cca 30 : 1. Musíme tedy nějakým způsobem transformovat (mapovat) původní vysoký dynamický rozsah (*HDR, High Dynamic Range*) na takový rozsah, který je schopno dané výstupní zařízení reprodukovat.

[1] Základní jednotkou jasu je kandela na metr čtvereční ($\mathrm{cd/m^2}$). $1\,\mathrm{cd/m^2}$ je jas zdroje, jehož svítivost na $1\,\mathrm{m^2}$ zdánlivé plochy je rovna 1 kandele. Zdánlivou plochou se rozumí obsah průmětu skutečně osvětlené plochy do roviny kolmé ke směru záření.

Obrázek 4.5: Obraz HDR se světelným zdrojem mapovaný globální, lineární metodou (vlevo). Stejný obraz mapovaný pomocí pokročilé lokální metody (vpravo). (Obrázky poskytl M. Čadík, ČVUT v Praze.)

Tato úloha se jmenuje *mapování tónů* (*tone mapping*) obrazů s vysokým dynamickým rozsahem na rozsah zobrazovacího zařízení (obr. 4.5). V dalším textu budeme obrazy s vysokým dynamickým obrazem označovat zkratkou HDR.

4.2.1 Získání a uložení obrazů s vysokým dynamickým rozsahem

Obraz HDR můžeme získat jako výsledek fyzikálně založené simulace světelné interakce v *prostorové syntetické scéně*, např. pokročilými metodami radiozity resp. sledování paprsku, viz kapitola 15.

Reálnou scénu s vysokým dynamickým rozsahem není možno zachytit konvenčními fotografickými metodami, neboť žádné jednotlivé nastavení expozice nezachytí současně nejjasnější a nejtmavější oblasti scény – dochází k přeexponování či podexponování části snímku a ztrátě detailů. Obdobná je situace v oblasti běžně dostupných digitálních fotoaparátů a kamer. Debevec a Malik [Debe97] navrhli metodu, pomocí které lze získat obraz HDR jedné scény kombinací několika běžných fotografií pořízených s různými expozicemi. V současnosti jsou k dispozici též různé specializované snímače HDR [Yang99], resp. metody, jak s běžně dostupným zařízením získat obraz s vyšším dynamickým rozsahem scény [Kang03]. Existují také pokusy o konstrukci zobrazovacího zařízení HDR, např. na základě kombinace digitálního projektoru a modulačního LCD displeje [Seet04].

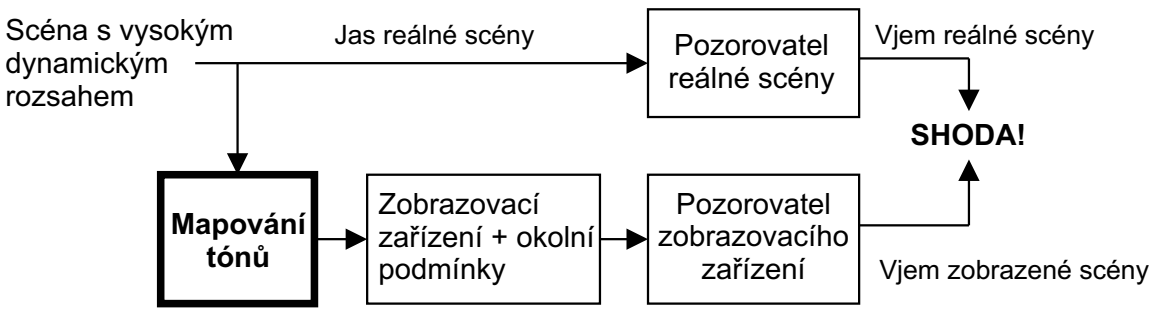
Obrazy HDR mají v počítačové grafice zajímavé aplikace, např. při doplňování syntetických

objektů do existující scény, kterou máme k dispozici pouze ve formě obrázku HDR [Debe98], nebo při osvětlování z obrázku (*image-based lighting*), kde se obraz HDR používá jako zdroj světla v prostorové scéně [Koll03].

Pro uložení obrazů HDR nelze použít běžných 24bitových (3×8 bitů) obrazových formátů popsaných v části 2.6, neboť ty umožňují uchovat jasový rozsah pouze do 255 : 1. Přímé uložení obrazu HDR v takové kvalitě, ve které byl pořízen, není vyhovující, neboť vede na značně veliké soubory dat ($3 \times$ hodnota v plovoucí řádové čárce $= 3 \times 32 = 96$ bit/pixel, což je čtyřikrát více, než obvyklých 24 bitů). Běžné kompresní metody, například slovníkové, fungují v případě obrazů HDR špatně. Z těchto důvodů vzniklo několik specializovaných formátů, které zachovávají dynamický rozsah vstupního obrazu a udržují velikost výstupního souboru v přijatelných mezích za cenu snížené přesnosti. Mezi nejčastěji používané patří formát RADIANCE RGBE (32bit/pixel), formát SGI LogLuv TIFF (48bit/pixel) a v roce 2003 zveřejněný formát OpenEXR (48 nebo 96bit/pixel + bezeztrátová komprese) vyvinutý firmou Industrial Light & Magic.

4.2.2 Techniky mapování tónů

Cílem technik mapování tónů je řešení dvou do jisté míry protichůdných požadavků: redukce dynamického rozsahu na rozsah, který je dané výstupní zařízení schopno reprodukovat, a přitom vyvolání takového subjektivního vjemu v pozorovateli tohoto zařízení, který je co možná nejbližší vjemu pozorovatele skutečné scény HDR (obr. 4.6). Naštěstí jsou tyto cíle do značné míry splnitelné díky tomu, že lidský vizuální systém je citlivý spíše na relativní, než absolutní hodnoty jasu, a při sledování scény HDR se koncentruje na části scény, než aby ji vnímal jako celek.



Obrázek 4.6: Problém mapování tónů podle [Tumb93].

Většina metod pro mapování tónů pracuje pouze s jasovou složkou, tzn. s úrovněmi šedi. Metody, které uvažují barevné složky, jsou spíše heuristické povahy. Obecně můžeme metody

rozdělit na globální (prostorově neměnné) a na lokální (prostorově se měnící). *Globální metody*, též nazývané křivky pro reprodukci tónů (*tone reproduction curves – TRCs*), jsou v principu podobné editačním křivkám pro obrazy s běžným dynamickým rozsahem (viz část 4.5.1). Mapují po jednotlivých obrazových bodech vstupní hodnoty na výstupní tak, že jedna hodnota vstupního jasu je vždy, v rámci celého obrazu, mapována na stejnou výstupní hodnotu (obr. 4.7 vpravo). *Lokální metody*, též operátory pro reprodukci tónů (*tone reproduction operators – TROs*), se snaží využít schopnosti lokální adaptace vizuálního systému a mapují vstupní hodnoty v závislosti na jejich pozici v obraze – obecně tedy dochází k namapování jedné vstupní hodnoty na různé výstupní hodnoty.

Metody pro mapování tónů lze též rozdělit podle implementovaného vizuálního modelu. Nejjednodušší (a nejrychlejší) metody neuvažují *žádný vizuální model* a jsou založeny na účelových postupech. Metody implementující vizuální model mohou být založeny na experimentálně získaných *prahových* (*threshold*) hodnotách vidění, *nadprahových* (*suprathreshold*) hodnotách vidění, nebo mohou uvažovat obě tyto varianty. Prahové hodnoty definují pro lidský vizuální systém hranici mezi rozlišitelnými částmi obrazu. Nadprahové hodnoty popisují odezvu vizuálního systému člověka nad hranicí viditelnosti, např. závislost vnímaného jasu na okolním osvětlení.

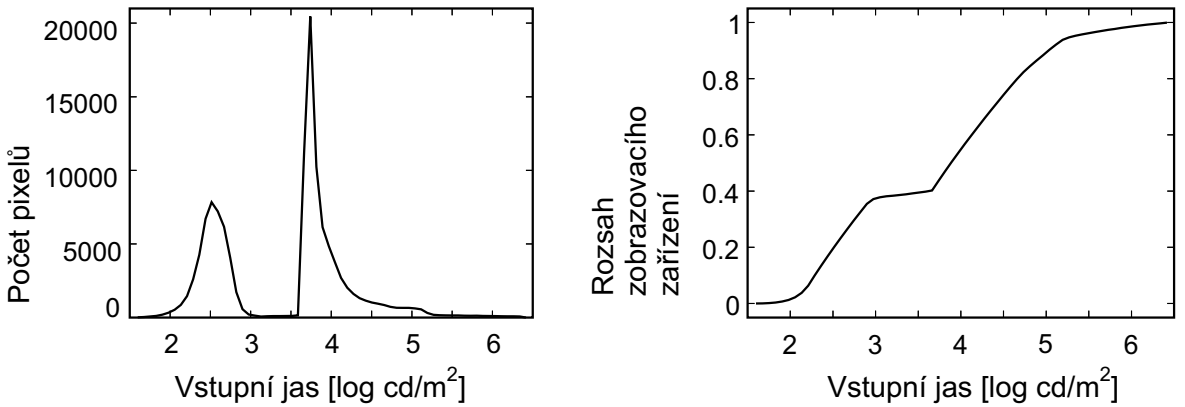
Samostatnou skupinu tvoří *časově závislé* metody, které berou v úvahu časovou závislost adaptace lidského vizuálního systému na změny v jasu scény. Časově závislé metody jsou vhodné pro interaktivní aplikace a animace – lze např. simulovat průjezd tunelem během slunečného dne, kdy se na rozhraní tunelu a osvětleného okolního prostředí prakticky skokově mění úroveň osvětlení, adaptace vizuálního systému je však pomalejší, a tak je vidění řidiče dočasně omezeno.

Dále můžeme identifikovat specializované metody, které využívají skutečnosti, že mají o obrazu, resp. o zobrazované scéně, k dispozici další informace. Například pro obraz scény získané radiozitní metodou je velice jednoduchým řešením mapování všech ploch scény, které nejsou světelnými zdroji, na 0 až 80 % rozsahu monitoru, přičemž světelné zdroje mapujeme na maximální hodnotu. Nevýhoda tohoto přístupu spočívá v tom, že jeho výsledky nemusejí vůbec odpovídat subjektivnímu vjemu pozorovatele: dvě geometricky identické scény, jedna osvětlena mdlým svitem svíčky, druhá ozářena silnou výbojkou, budou touto metodou na monitoru zobrazeny prakticky stejně.

Globální metody

Prvními, kdo popsali problém mapování tónů v počítačové grafice, byli Tumblin a Rushmeierová [Tumb93], kteří zároveň navrhli obecný koncept, na jehož základě implementovali jednu z prvních metod. Tato technika spočívá v modelování pozorovatele scény s vysokým dynamickým rozsahem a pozorovatele zobrazovacího zařízení. Cílem metody je sjednocení vjemů

těchto modelovaných pozorovatelů, zejména pak sjednocení *vnímaného jasu* (*brightness*), viz obrázek 4.6.



Obrázek 4.7: Vlevo histogram jasů pro obrázek 4.9. Vpravo převodní křivka zkonstruovaná na základě tohoto histogramu.

Velmi jednoduchý model je založen na zachování *vnímaného kontrastu* [Ward94]. Hodnoty vstupního jasu jsou transformovány na zobrazitelný rozsah pouze pomocí multiplikativní konstanty. Mapovací funkce je tedy lineární a nejmenší pozorovatelný rozdíl – JND (*Just Noticeable Difference*) původních jasů je mapován jako JND na zobrazovacím zařízení. Díky tomu je zachován vjem kontrastu, ale ve většině případů dochází k ořezání vysokých a nízkých hodnot jasu. Výpočetní náročnost Wardovy metody je velmi malá.

Ward a kol. [Lars97] navrhli metodu, jejímž hlavním cílem je zachování *viditelnosti objektů* (*object visibility*). Zachování viditelnosti v tomto případě znamená, že objekty viditelné ve skutečnosti by měly být vidět i na displeji, a podobně objekty, které ve skutečnosti viditelné nejsou, by na displeji být vidět neměly. Metoda je založena na výpočtu *histogramu jasů* vstupního obrazu HDR (obr. 4.7 vlevo). Na základě tohoto histogramu[2] se zkonstruuje převodní křivka, pomocí které se převedou hodnoty jasů vstupního obrazu na hodnoty výstupních jasů (obr. 4.7 vpravo). Díky tomu, že převodní křivka se konstruuje jako kumulativní distribuční funkce histogramu (s omezeními podle známých parametrů lidského vidění), dochází ke zhuštění kontrastu pro v obraze málo zastoupené hodnoty jasu (čímž se předejde přeexponování či podexponování), zatímco pro vysoce zastoupené hodnoty je kontrast zachován.

Druhým cílem metody je vyvolání takového subjektivního vjemu, který je co možná nejbližší

[2]Histogram jasů se ve skutečnosti nepočítá pro vstupní obraz, ale konstruuje se na základě obrazu adaptace. Obraz adaptace vznikne ze vstupního obrazu průměrováním v rámci oblastí o velikosti 1° prostorového úhlu, což přibližně odpovídá oblasti lokální adaptace lidského oka.

vjemu pozorovatele skutečné scény. Z tohoto důvodu jsou k samotnému mapování tónů ještě přidány simulace *oslnění* a změn ve vnímání detailů, kontrastu a barev v závislosti na jasu. Silné světelné zdroje v oblasti periferního vidění snižují citlivost vizuálního systému na kontrast. Snížení citlivosti je způsobeno rozptylem světla v čočce a očním moku – oslněním oční komory. Efekt oslnění je méně patrný, pokud se díváme přímo do světelného zdroje, protože se oko adaptuje na vyšší úroveň osvětlení. Při redukci barev se berou do úvahy aspekty tyčinkového (skotopického) a čípkového (fotopického) vidění. Tyčinky jsou velice citlivé na světlo, ale neumožňují barevné vidění. Je-li tedy oko adaptováno ve skotopické oblasti (tzn. na jas do $1 \cdot 10^{-3}\mathrm{cd/m^2}$), mizí vjem barvy. Mezi skotopickou a fotopickou oblastí vidění je přechodná, mezopická oblast (rozsah jasů $1 \cdot 10^{-3}\mathrm{cd/m^2}$ až $5\,\mathrm{cd/m^2}$), kde se uplatňují jak tyčinky, tak čípky. Postup je popsán algoritmem 4.5. Mezi výhodné vlastnosti metody patří *idempotence* – dříve namapovaný obraz přivedený znovu na vstup metody se již nezmění.

Vstup: Obraz HDR
Výstup: Obraz RGB zobrazitelný na monitoru

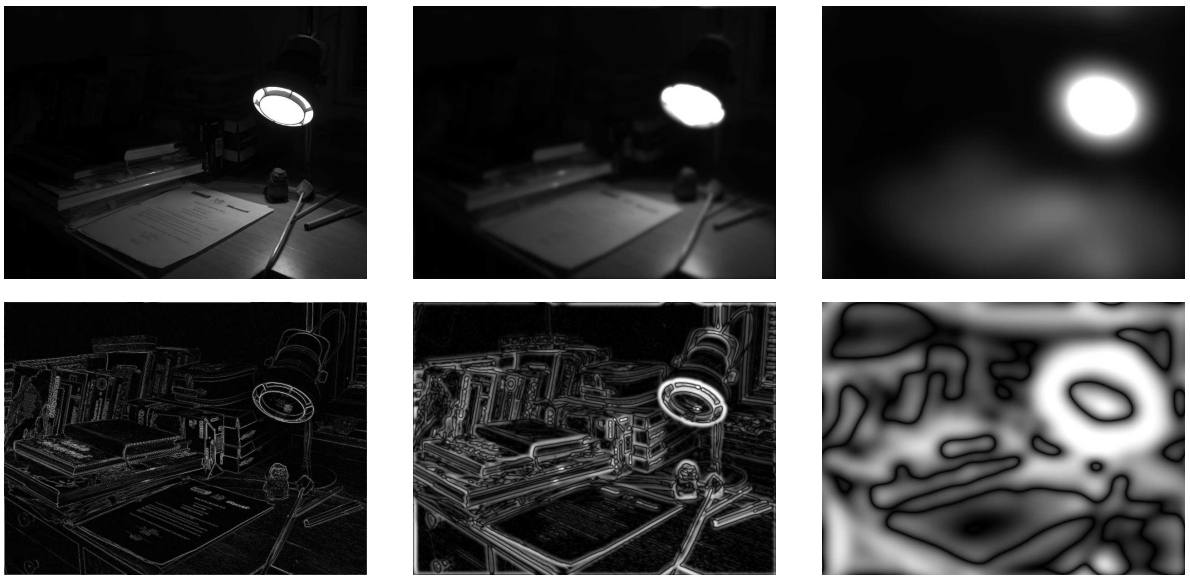
1. Vypočítej obraz adaptace převzorkováním a průměrováním obrazu HDR se vzorky o velikosti 1° srad.
2. Vypočítej obraz oslnění a přičti jej k obrazu adaptace i k původnímu obrazu.
3. Dle závislosti rozlišovací schopnosti oka na jasu proveď lokální rozmazání obrazu.
4. Dle závislosti vnímání barev na jasu proveď redukci barev.
5. Zkonstruuj histogram jasů pro obraz adaptace.
6. Přizpůsob histogram dle závislosti vnímání kontrastu na jasu.
7. Transformuj obraz podle přizpůsobeného histogramu.
8. Transformuj získané hodnoty jasů do prostoru RGB a proveď gama korekci.

Algoritmus 4.5: Globální mapování tónů dle [Lars97]

Globální metody mapování tónů se díky nízké výpočetní složitosti (v porovnání s lokálními metodami) uplatňují v interaktivních aplikacích, animacích a videu. Všeobecně uznávaným principem při snaze o dosažení co největší věrnosti reprodukce obrazu HDR je zachování relativních poměrů jasů v obraze, *lokálního kontrastu*. Pro globální metody je zachování lokálního kontrastu v obraze velkým, v některých případech neřešitelným problémem. Řešení problému lokálního kontrastu je hlavní motivací pro použití lokálních metod.

Lokální metody

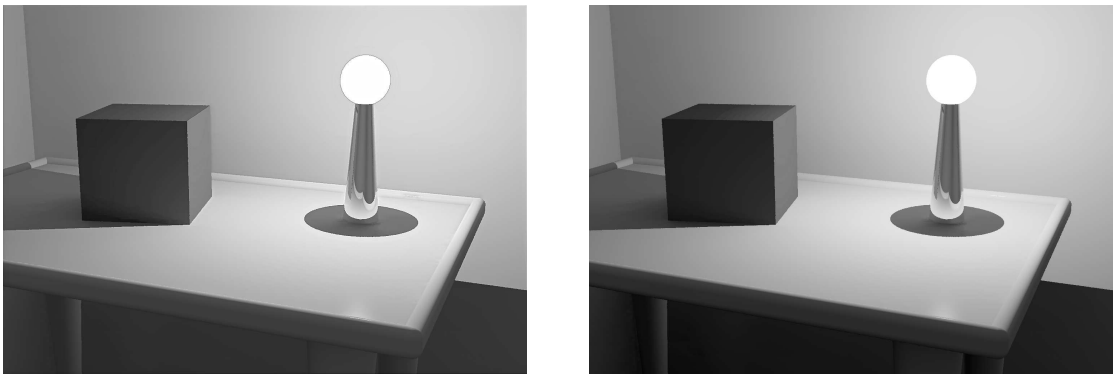
Tyto metody mapují vstupní hodnoty jasu na výstupní hodnoty v závislosti na jasu pixelů v určitém okolí. Jedna vstupní hodnota jasu tak může být na různých pozicích v obraze mapována na různé výstupní hodnoty. Cílem je redukce velkých změn v osvětlení při zachování lokálních kontrastů, detailů.



Obrázek 4.8: Jas vstupního obrazu na třech různých úrovních prostorového rozlišení (horní řádek) a odpovídající obrazy lokálního kontrastu (spodní řádek)

Lokální metody jsou nejčastěji založeny na nějaké *multirezoluční dekompozici* (*multiresolution decomposition*). Jednotlivé lokální metody ([Rein02], metoda LCIS [Tumb97], Fattalova gradientní metoda [Fatt02], atd.) jsou založeny na různých dekompozicích, ale obecný princip lze zjednodušeně nastínit. Vstupní obraz je nejprve filtrován sadou filtrů typu dolní propust (typicky sadou gaussovských filtrů, viz část 4.6.1), čímž vznikne množina obrazů, které reprezentují průměr vstupního jasu v různých prostorových rozlišeních (obr. 4.8 nahoře). Od každého obrazu v této množině je následně odečten obraz na nižší úrovni rozlišení, čímž vznikne množina obrazů, které reprezentují lokální kontrast (obr. 4.8 dole). Dynamický rozsah každého obrazu lokálního kontrastu je zmenšen a výsledný obraz poté složen jako kombinace těchto lokálních kontrastů a redukovaného obrazu z nejnižší úrovně rozlišení. Lokální metody měří i mapují lokální kontrast lépe než globální metody a obecně tedy poskytují kvalitnější výsledky.

Ve výsledném obraze však mohou vznikat artefakty, jako např. tzv. *svatozář* (*halo*) – vznik světelných okrajů objektů v obraze (obr. 4.9).



Obrázek 4.9: Tmavý obrys světelného zdroje (vlevo), tzv. svatozář, je nežádoucím artefaktem, který je důsledkem lokální metody mapování tónů. Při použití globální metody (vpravo) artefakt svatozáře nevzniká. (Obrázky poskytl M. Čadík, ČVUT v Praze.)

Jiný lokální přístup, založený na maximalizaci entropie obrazu popsal A. A. Goshtasby. Metoda předpokládá množinu vstupních obrazů získaných různými expozicemi a tedy obsahující přeexponované, podexponované a informačně zajímavé oblasti. Z těchto obrazů jsou na základě výpočtu entropie vybrány informačně zajímavé podoblasti, které jsou vynásobeny směšovací funkcí (*blending*) a zkombinovány do výsledného obrazu.

Techniky mapování tónů lze implementovat pomocí programovatelného grafického hardware [Good03]. Oblast lokálních i globálních metod mapování tónů je předmětem aktivního výzkumu. Srozumitelný přehled metod lze nalézt v článku Devlinové a kol. [Devl02].

4.3 Geometrické transformace diskrétního obrazu

Geometrické transformace z kapitoly 21 pracují buďto s bodem, nebo předpokládají spojitou reprezentaci objektu. Diskrétní obraz je jiné povahy, a to s sebou nese určité problémy. Předpokládejme, že transformujeme obraz A na obraz B. Obecná geometrická transformace přiřazuje pixelu, který má diskrétní souřadnice $[i, j]$ nějaké neceločíselné místo $[x, y]$. Jak takto transformovaný obraz zobrazit? V novém obraze mohou vznikat „díry“, případně několik pixelů se může mapovat na jedno místo.

V dalším textu budeme předpokládat, že máme vstupní obraz A, který budeme transfor-

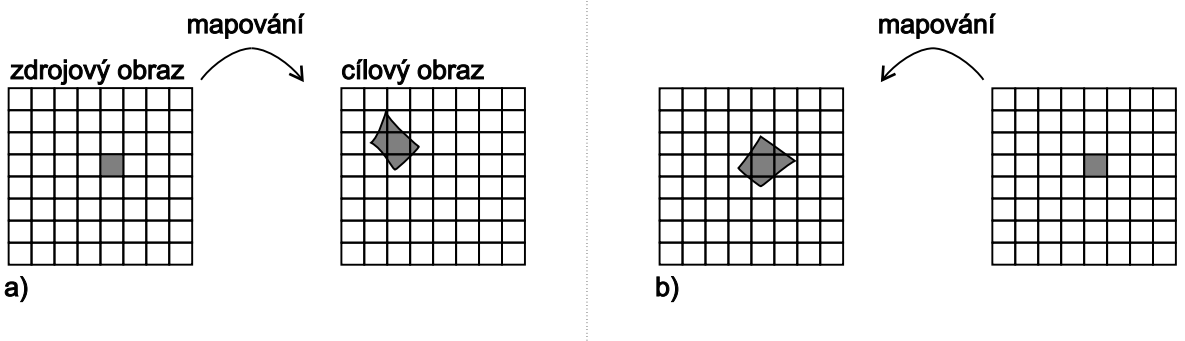
movat. Výstupem bude obraz B, nebo v některých případech ještě nějaký pomocný obraz (meziobraz) I.

Mějme dvourozměrný obraz složený z bodů $[x, y]$. Geometrická transformace obrazu je funkce

$$T(u, v) = [x(u, v), y(u, v)] \tag{4.4}$$

Vstupy funkcí $x(u, v)$ a $y(u, v)$ jsou souřadnice u a v a funkce vrací novou polohu pixelu.

Mapování určuje způsob přiřazení pixelů z jednoho obrazu do druhého a můžeme ho rozdělit na *dopředné mapování* (*forward mapping*) a *zpětné mapování* (*backward mapping*). Oba tyto případy demonstruje obrázek 4.10.



Obrázek 4.10: a) Dopředné a b) zpětné mapování. Pixel z jednoho obrazu se mapuje na oblast ve druhém obrazu, některé pixely může pokrývat zcela, jiné pouze z části.

Obecným problémem geometrických transformací obrazu je, že nemusí zachovávat velikost vstupního obrazu. Transformace může obraz zvětšovat, nebo zmenšovat. To způsobuje problémy při mapování dopředném i zpětném. Jednomu pixelu ve výstupním obraze se tak přiřadí několik pixelů, nebo také žádný, z obrazu vstupního.

Při dopředném mapování procházíme pixely vstupního obrazu A a hledáme jejich umístění ve výstupním obraze B. Při zpětném mapování procházíme pixely výstupního obrazu B a hledáme odpovídající oblasti v obraze A, které se do procházených pixelů mapují. Problémem dopředného mapování je, že ve výsledném obraze B mohou vznikat volná místa.

Další důležitý pojem je *separabilní operace*. Transformace (4.4) je separabilní, pokud ji lze zapsat jako složení dvou transformací, z nich každá je funkcí pouze jediné proměnné, tj.

$$T(u, v) = F(u, G(v)). \tag{4.5}$$

Například první transformace G pracuje pouze s řádky obrazu, zatímco transformace F transformuje sloupce. V implementaci potom transformace G generuje nějaký meziobraz

I, se kterým pracuje transformace F. Typickým reprezentantem separabilních operací jsou dvouprůchodové algoritmy otáčení a warping. Separabilitu využijeme s výhodou u proudového zpracování grafické informace.

Dalším příkladem separabilní transformace je dvourozměrná (a obecně n–rozměrná) Fourierova transformace z odstavce 2.2. Fourierův obraz vznikne aplikací jednorozměrné transformace nejprve na všechny řádky vstupního obrazu a poté na sloupce obrazu, vzniknuvšího z předchozího kroku.

Algoritmy uvedené v dalším textu pracují na principu zpětného mapování. Pokud tomu bude jinak, bude to výslovně uvedeno.

4.3.1 Převzorkování

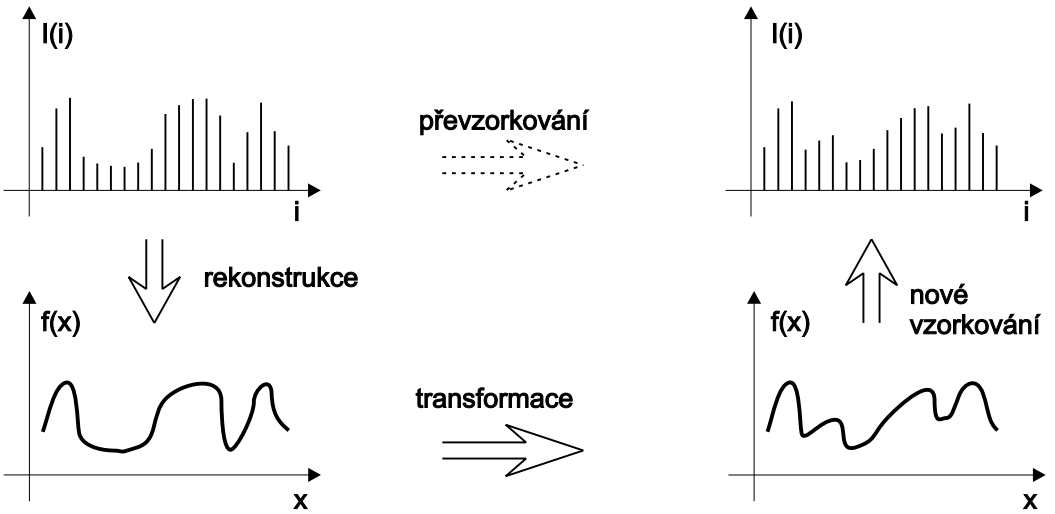
Při práci s obrazem reprezentovaným v diskrétním rastru, se setkáváme s některými zvláštnostmi, které při spojité reprezentaci nenastávají. Uvažme případ, kdy potřebujeme zvětšit velikost obrazu dvakrát. Naivním přístupem zjistíme, že při natažení dojde k tomu, že v obraze vzniknou „díry“. Každý druhý pixel na řádku i ve sloupci bude „prázdný“, protože zde není žádná informace k dispozici. V počítačové grafice se používá jednotný postup pro většinu takovýchto operací. Jeho podstatou je nalezení spojité aproximace diskrétního obrazu tak, jak je naznačeno na obrázku 4.11. Dalším krokem je provedení transformace spojitého obrazu a následuje jeho nové vzorkování. Tento postup demonstruje obrázek 4.11 podle [Wool90]. Posloupnosti operací, kdy se nejprve signál rekonstruuje, poté se s ním pracuje a opět se vzorkuje zpět do diskrétní oblasti, se říká *převzorkování* (*resampling*). Obraz obyčejně není nutné rekonstruovat celý, ale rekonstruuje se pouze oblast, která se používá pro výpočet hodnoty nového pixelu. Rekonstrukce, transformace a nové vzorkování jsou tedy úzce spjaty.

Podstatou převzorkování je nalezení spojitého obrazu k obrazu diskrétnímu. Z kapitoly 2 víme, že ideální rekonstrukční filtr má tvar funkce *sinc* (4.9). Víme již také, že tato funkce je neomezená, a proto se vždy nějakým způsobem aproximuje. V dalším textu uvedeme některé aproximace této funkce a jejich použití při získávání spojitého obrazu.

Mezi základní operace aplikované v prostorové oblasti patří především geometrické transformace. Geometrické transformace lze rozdělit na *lineární* a *nelineární*. Lineární transformace, jsou popsány v odstavci 21.2. V dalším textu si z lineárních transformací všimneme především změny měřítka, z nelineárních transformací si uvedeme tzv. *warping*.

4.3.2 Rekonstrukce

Rekonstrukcí rozumíme přechod od diskrétní funkce I_i, definované pro $i = 1, 2, \ldots$, ke spojité funkci $f(x)$, $x \in R$. Obor hodnot obou funkcí je stejný a budeme předpokládat, že se jedná o reálná čísla. Rekonstrukce je nejednoznačně určená úloha, neboť hodnoty mezi vzorky nejsou



Obrázek 4.11: Transformace diskrétního obrazu – převzorkování podle [Wool90]

definovány a musí se nějakým způsobem aproximovat. Právě způsob aproximace určuje různé metody, které se liší kvalitou a rychlostí.

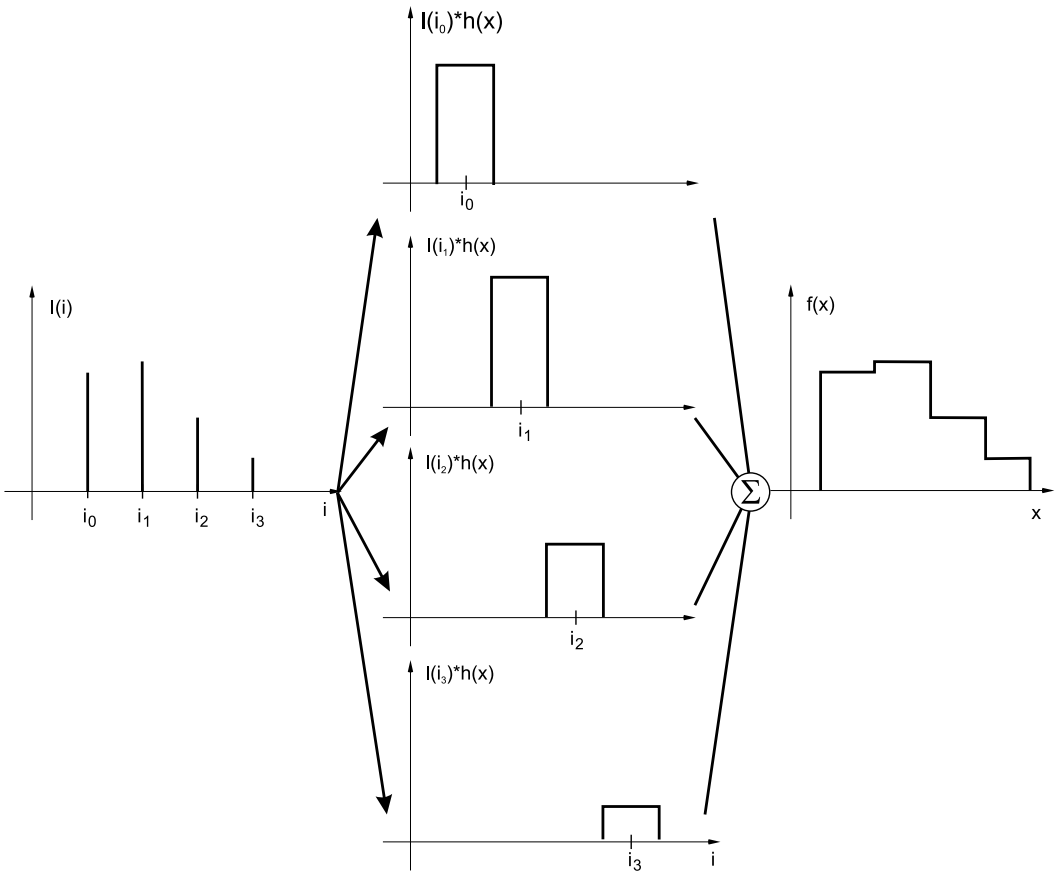
Pro rekonstrukci s výhodou využijeme konvoluci detailně popsanou v kapitole 2.2.5 na straně 46. Zde se omezíme na následující funkční zjednodušení.

Mějme funkci $h(x)$, které budeme říkat konvoluční jádro, nebo v tomto kontextu také rekonstrukční filtr. Jedinou podmínkou kladenou na tuto funkci je, že plocha vymezená touto funkcí musí být jednotková, neboli

$$\int_{-\infty}^{+\infty} h(t)dt = 1.$$

Kdyby byla hodnota tohoto integrálu větší nežli jedna, nová funkce by byla konvolucí zesílena a naopak. Při rekonstruování spojité funkce $f(x)$ vynásobíme hodnotu vzorků v bodech I_i konvolučním jádrem, které posuneme do bodu $i\Delta x$ tak, jak ukazuje obrázek 4.12. Výsledkem jsou ve směru osy y změřítkovaná konvoluční jádra posunutá ve směru osy x. Takto získané funkce se sečtou a jejich výsledek je spojitá funkce $f(x)$.

Podle typu konvolučního jádra získáme lépe či hůře aproximovanou původní funkci. Platí známé pravidlo, že za vyšší kvalitu se platí vyšší výpočetní náročností. Nejhorší, avšak nejrychlejší výsledky získáme rekonstrukcí metodou nejbližšího souseda, bilineární interpolace poskytuje výsledky kvalitnější, ale má tendenci rozmazávat obraz. Interpolace plochami vyššího stupně poskytují kvalitní výsledky, jsou však výpočetně náročné.



Obrázek 4.12: Rekonstrukce metodou nejbližšího souseda podle [Moll02]

Poznamenejme, že původní funkci ze vzorků nelze obecně získat. Jedinou, bohužel ne častou výjimkou, jsou frekvenčně omezené funkce, vzorkované s frekvencí rovnou či vyšší frekvenci dané Shanonovou vzorkovací větou. Pokud není tato podmínka splněna, je každá rekonstrukce více či méně zdařilou aproximací funkce původní. Původní funkci tedy obecně získat nemůžeme, můžeme však klást na aproximační funkci určité požadavky, které vedou k vizuálně přijatelným výsledkům. V první řadě, rekonstrukční filtr by neměl být nespojitý. Nespojitým filtrem zrekonstruovaný obraz totiž bude obsahovat nespojitosti, které se projeví jako hrany v obraze. Typickým příkladem je interpolace nejbližším sousedem. Rekonstrukční funkce by dále měla zachovávat spojitost prvních derivací a rekonstrukční filtr by neměl mít příliš velkou změnu gradientu v okolí vzorku. Příkladem rekonstrukce, která tuto podmínku nesplňuje, je lineární

interpolace. Důsledkem je poměrně vizuálně nepříznivé rozmazání obrazu, které působí rušivě zejména při rekonstrukci textu. Dále uvedeme nejčastěji používaná konvoluční jádra.

Interpolace nejbližším sousedem

Patrně nejjednodušší metoda rekonstrukce se nazývá *interpolace nejbližším sousedem* (*nearest neighbour interpolation*) nebo též *point shift* nebo také *sample and hold*. Výpočet spojité funkce $f(x)$ z diskrétní funkce I_i definované v diskrétních bodech $i = 1, ..., n$ má tvar

$$f(x) = I_i \ ; \ i - \frac{1}{2} < x \leq i + \frac{1}{2}, \tag{4.6}$$

konvoluční jádro má tvar

$$h(x) = \begin{cases} 1 & pro: \ 0 \leq |x| < 1/2 \\ 0 & pro: \ 1/2 \leq |x| \end{cases}$$

Principem této metody je prosté okopírování hodnoty nejbližšího souseda v okolí vzorku.

Obrázek 4.13: Srovnání výsledků různých rekonstrukcí aplikovaných při zvětšování obrazu. Zleva doprava: metoda nejbližšího souseda, bilineární a bikubická interpolace.

Tato metoda je zjevně velice rychlá, implementuje se jednoduše jako zaokrouhlení reálného čísla, na druhou stranu vede k nežádoucím efektům například na hranách s malým sklonem, při zvětšení zvýrazňuje skoky a při zmenšení poškozuje tenké čáry, které z obrazu jednoduše zmizí. Především z důvodu rychlosti se tato metoda používá zejména k rychlému náhledu (*preview*).

Je důležité poznamenat, že tato metoda aproximuje funkci $f(x)$ funkcí obecně *nespojitou*. Do obrazu jsou, například při zvětšení, vkládány hrany na přechodech mezi dvěma vzorky, které se projevují jako pixelizace obrazu – obraz je složen z mozaiky čtverců.

Lineární a bilineární interpolace

Mějme dva sousední vzorky v bodech i a $i + 1$ s hodnotami I_i a I_{i+1} a hledejme hodnotu $f(x)$ v bodě x, $i < x \leq i + 1$. Krajní body se proloží úsečkou a hledaná hodnota v bodě x se vypočítá jako

$$f(x) = I_i + \frac{x-i}{\Delta x}\left[I_{i+1} - I_i\right]. \tag{4.7}$$

Odpovídající konvoluční jádro má tvar jehlanového stanu, bývá nazýváno *tent function* (viz obrázek 2.5) a má tvar

$$h(x) = \begin{cases} 1-|x| & pro:\ 0 \le |x| < 1 \\ 0 & pro:\ 1 \le |x| \end{cases} \tag{4.8}$$

V textu uvádíme rekonstrukční filtry pro jednorozměrné případy. Pro lineární interpolaci uděláme v tomto odstavci výjimku, na které demonstrujeme, jak se jednorozměrné případy zobecňují.

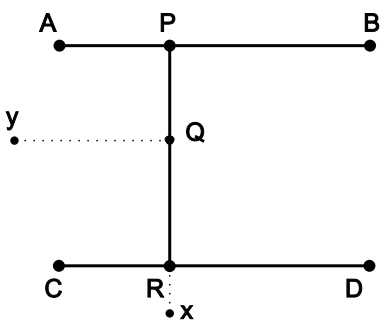
Bilineární interpolace používá pro rekonstrukci hodnoty pixelu čtyři pixely z jeho okolí tak, jak je patrné na obrázku 4.14. Budeme hledat hodnotu pixelu v bodě Q. Známé jsou hodnoty v pixelech A, B, C, D. Bilineární interpolace je separabilní, to znamená, že ji můžeme složit z jedné operace postupně aplikované nejprve na řádky a poté na sloupce obrazu. Tak z bodů A a B nejprve získáme hodnotu v bodě P a stejně tak pro druhý řádek z C a D vypočteme hodnotu pro R. Souřadnice x a y se snadno normalizují a tak je

$$P = xA + (1-x)B \quad \text{a} \quad Q = xC + (1-x)D.$$

Z těchto dvou nových bodů poté lineární interpolací vypočteme hodnoty výsledku

$$Q = yP + (1-y)R.$$

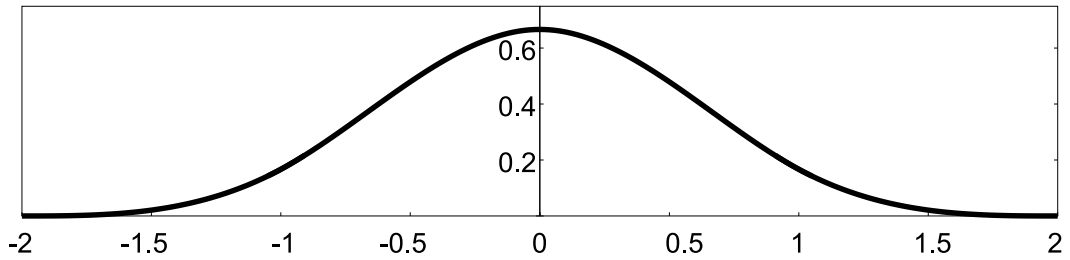
Bilineární interpolace je rychlá, neboť se pro výpočet hodnoty pixelu používají pouze čtyři body z jeho okolí. Její nevýhodou je, že rozmazává původní ostré přechody. Jak je z průběhu funkce patrné, při prudkých změnách sousedních hodnot funkce velice rychle opouští hodnotu každého z interpolovaných bodů. Tato metoda je běžně implementována v technickém vybavení počítačů.



Obrázek 4.14: Bilineární interpolace

Kubická interpolace

Lepších výsledků, nežli v předchozích případech je dosaženo, interpolačními funkcemi, které mají hladší průběh, neboť lépe respektují ostrost obrazu. Tyto metody potřebují pro svůj výpočet více vzorků ze svého okolí a jsou proto výpočetně náročnější.



Obrázek 4.15: Tvar průběhu funkce jádra kubické interpolace

Častou metodou je použití kubické interpolace či aproximace pomocí kubické B-spline křivky. Konvoluční jádro aproximace má tvar

$$h(x) = \frac{1}{6}\begin{cases} 3|x|^3 - 6|x|^2 + 4 & pro: \ 0 \leq |x| < 1 \\ -|x|^3 + 6|x|^2 - 12|x| + 8 & pro: \ 1 \leq |x| < 2 \\ 0 & pro: \ 2 \leq |x|. \end{cases}$$

Tato interpolace se též nazývá Parzenovo okno.

Sinc filtr

Zůstává otázkou, zda je možno nějakou funkci zrekonstruovat přesně. Odpověď nalezneme ve Fourierově transformaci. Předpokládejme, že máme funkci, která je frekvenčně omezená a obsahuje nejvyšší frekvenci f_s. Ideální rekonstrukční filtr ve Fourierově oblasti bude mít tvar obdélníku s šířkou f_s rozloženým symetricky kolem počátku tak, jak je vidět na obrázku 2.4. Této funkci se také říká ideální nízkopásmový filtr, protože odstraňuje všechny frekvence vyšší, nežli f_s. Pro rekonstrukci této funkce potřebujeme její Fourierův vzor, který se jmenuje *sinc*:

$$sinc(x) = \frac{sin(\pi x)}{\pi x}. \tag{4.9}$$

Pokud vzorkujeme spojitou funkci s frekvencí f_s, musíme ji rekonstruovat s $sinc(f_s \cdot x)$. Funkce $sinc(x)$ je neomezená, a proto pro výpočet jediného pixelu používá ze vstupní diskrétní funkce všechny její hodnoty. V praxi to znamená, že pro výpočet jediného bodu musíme použít celý vstupní obraz. Zjevně se jedná o metodu výpočetně náročnou. Další nevýhodou je, že funkce nabývá i záporných hodnot.

Interpolací existuje více a liší se podle svých aplikací, jako příklad uveďme Kaiserovo, Hananovo, Gaussovo či Lanczosovo okno [Wool90]. Pro některé vstupní obrazy (obsahující například písmo) je žádoucí zachovat ostré hrany, zatímco jinde rozmazání není na závadu. Aplikace obvykle nabízejí více rekonstrukčních filtrů a záleží na aplikaci, který z nich je vhodný.

Vícerozměrné interpolace

Všechny uvedené rekonstrukční filtry jsme demonstrovali na jednorozměrných případech. Ve dvojrozměrném obraze se však tyto operace musí aplikovat dvakrát, jednou pro řádky a podruhé pro sloupce obrazu. Z tohoto důvodu se spíše nežli s kubickou interpolací setkáme s interpolací bikubickou a stejně tak s interpolací bilineární. Pro interpolaci nejbližším sousedem se žádný dvojrozměrný název nezavádí, neboť nejbližší soused je obyčejně jen jeden.

4.3.3 Změna rozlišení

Častým požadavkem je změna rozlišení obrazu, a to jak jeho zmenšení, tak zvětšení. Různé algoritmy pro změnu velikosti obrazu se liší především tím, jak kvalitně rekonstruují spojitou funkci z diskrétního obrazu. Přirozeně, čím přesnější tato rekonstrukce je, tím je algoritmus časově náročnější a naopak. Většina algoritmů, které se pro tyto operace používají, je separabilní, a proto k jejich vyjádření s výhodou použijeme jednorozměrné konvoluce (2.10).

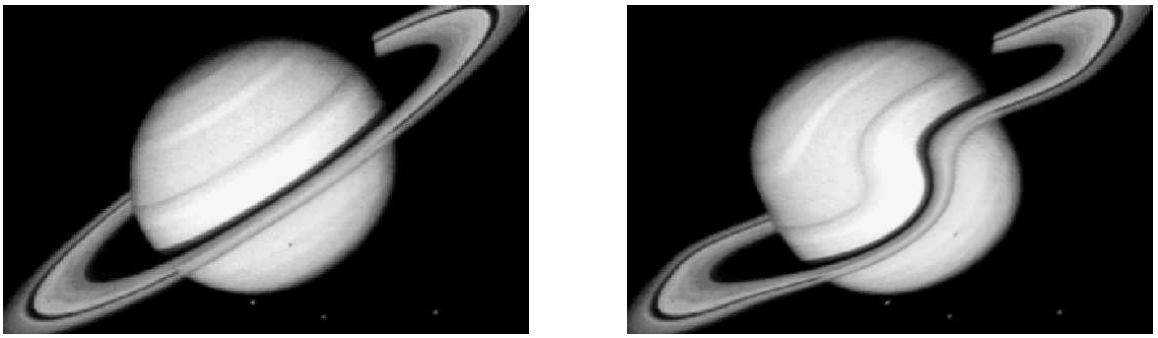
Na základě toho co jsme uvedli výše můžeme celý postup snadno poskládat dohromady. Pro každý pixel ve výstupním obraze se zpětným mapováním určí souřadnice pixelu ve vstupním obraze. Tyto souřadnice jsou neceločíselné a tak se oblast zrekonstruuje z okolních pixelů. Nejjednodušším případem je zaokrouhlení na hodnotu nejbližšího souseda, složitější algoritmy používají více pixelů z původního obrazu.

4.4 Warping a morfing

Warping obrazu, česky kroucení, zvlnění, pokřivení či deformace, se provádí aplikací nějaké nelineární transformace na jediný obraz, který se mění jako v křivém zrcadle (obrázek 4.16). Jako morfing označujeme obecný proces, kdy se jeden obrázek či objekt postupným přechodem přeměňuje v jiný (viz obrázek 4.25).

Pojem morfing má obecný význam přeměny zcela libovolného objektu v jiný a tak nemusí jít nutně o transformaci obrazu, i když ve většině případů se morfingem rozumí právě tato transformace. Pohledem do historie kinematografie zjistíme, že morfing není ničím novým. Kdysi se ve filmu dělával metodou „chytrého střihu“. V okamžiku, kdy mělo dojít ke změně objektu bylo zastaveno natáčení, herec (objekt) se pozměnil, natočil se další kousek filmu, atd. Jedná se o poměrně snadnou metodu v případě, že je deformovaný objekt v klidu (*stop–motion*), složitější je morfing pohybujících se objektů (*go–motion*).

Počítačem podporovaný morfing můžeme rozdělit do dvou a tří rozměrů. Trojrozměrný morfing se provádí v modelovém prostoru a je jednoduchý u modelů se stejným počtem definujících prvků. Předpokládejme dva objekty reprezentované pomocí okřídlených hran (*winged edge*). Pro správný morfing musí mít oba objekty stejný počet uzlů, hran a plošek a sousednost



Obrázek 4.16: Warping obrazu

uzlů stejně jako hran v obou modelech si musí odpovídat. Je snadný morfing například krychle na kvádr, nebo koule na elipsoid. Tento intuitivní postup je složitější u nekonvexních objektů, objektů s otvorem, či v případě „divoké“ změny tvaru.

Jiným požadavkem je změna tvaru objektu bez změny jeho struktury, tj. bez editace ploch, ze kterých se skládá. Řešení poskytuje technika zvaná *free form deformation* [Sede86] a její varianty, např. *NURBS based free form deformation* [Lamo94]. Model může být libovolný, protože se modifikuje prostor kolem něj. Podstatou těchto metod je ponoření modelu do trojrozměrné sítě určující řídicí body trojrozměrného Bézierova, resp. NURBS tělesa. Změnou polohy řídicích bodů se mění tvar prostoru a tím i tvar modelu (viz část 6.4.2). Později uvidíme, že se jedná o trojrozměrnou analogii síťového warpingu.

Dvojrozměrný morfing a warping lze rozdělit na práci se spojitým a diskrétním obrazem. Morfing spojitých dvourozměrných objektů se nejčastěji využívá v počítačem podporované tvorbě animací typu *cartoons* a je dnes standardní metodou tvorby animací. Algoritmy morfingu či warpingu, které pracují s diskrétním obrazem, jsou hlavním předmětem této části knihy.

Výsledkem nelineárních transformací obrazu je obyčejně sekvence obrazů, které jsou v případě morfingu přechodem od jednoho vstupního obrázku k druhému a v případě warpingu přechodem od jediného vstupního obrázku do jeho požadované změny.

4.4.1 Alfa míchání, klíčování na barvu a klíčování na modrou

Nejjednodušší algoritmus postupné přeměny jednoho obrázku v druhý je založen na interpolaci barvy pixelu. Tento postup se anglicky nazývá (*cross–dissolve*), česky bychom mohli říci *prolnutí* obrazů. Mějme dva obrazy A a B stejného rozlišení a stejné barevné hloubky. Prolnutí obrazů

interpoluje hodnoty pixelů podle vztahu pro lineární interpolaci

$$X_i = A + \frac{i}{n}(B - A), \tag{4.10}$$

kde X_i je výsledný obraz, n je číslo posledního obrázku $i = 0, 1, \ldots, n$ a celkový počet vygenerovaných obrázků je $n + 1$. Příklad je ukázán na obrázku 4.17. Při zpracovávání RGB obrázků se podle vztahu (4.10) upravují jednotlivé barevné kanály.

Obrázek 4.17: Prolnutí (*cross-dissolve*) dvou obrazů. Zleva doprava: 0, 25, 75 a 100 %.

Tato operace se obyčejně realizuje jako tzv. *alfa míchání* (*alpha blending*). Každému pixelu v obraze se přiřadí procento průhlednosti reprezentované jako α a tato hodnota se uloží spolu s hodnotami RGB jako RGBA, kde poslední písmeno značí právě průhlednost pixelu. Zcela neprůhledné pixely mají $\alpha = 1$, částečně průhledné mají $\alpha < 1$. Předpokládejme dva pixely, c_1 resp. c_2 s hodnotami α_1 resp. α_2. Hodnota výsledného pixelu c se pak získá

$$c = \alpha_1 c_1 + \alpha_2 c_2. \tag{4.11}$$

Nejjednodušší možností je alfa konstantní pro celý obraz, každý pixel však může mít jinou průhlednost. Existuje několik způsobů uložení průhlednosti do obrazu. Nejčastější je tzv. přednásobená alfa (*premultiplied alpha*). Všechny hodnoty RGB jsou hodnotou A vynásobeny a tato nová hodnota je uložena. Například pro hodnoty $[0.7, 0.8, 1, 0.5]$ získáme po vynásobení $[0.35, 0.4, 0.5, 0.5]$. Tento způsob uložení zjednoduší alfa míchání (4.11) na

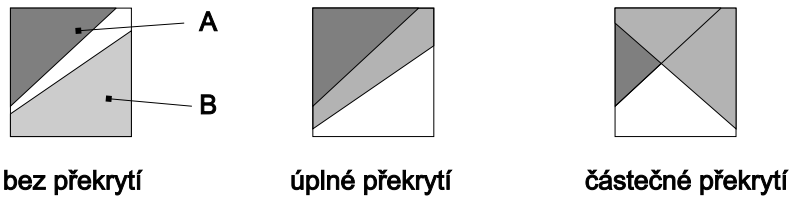
$$c = c_1 + \alpha_2 c_2.$$

Uložení v této podobě zjednodušuje alfa míchání pokud se nezmění hodnoty průhlednosti.

Druhý způsob uložení hodnoty alfa je prosté uložení hodnot tak, jak jsou dány (*unmultiplied, nonpremultiplied, unassociated alpha*). Tento způsob je méně častý a setkáme se s ním například ve formátech TIFF a PNG.

Jednoduše formulovaná operace alfa míchání má hlubší význam, než je na první pohled zřejmé. Alfa se může chápat jako výše zmíněná průhlednost každého pixelu a může být

aplikována na obrazy. Průhlednost může být použita při kompozici trojrozměrné scény, kdy může určovat procento pokrytí pixelu určitým objektem. Například obrysová hrana krychle může procházet pouze částí pixelu a krychle ho tak nevyplňuje úplně. Průhlednost pak odpovídá procentu pokrytí objektem.



Obrázek 4.18: Vzájemná poloha dvou objektů v jednom pixelu

Při kombinaci dvou obrazů, nejčastěji však při postupném kreslení jednotlivých objektů do obrazové paměti, může vznikat *alias*, tj. může dojít ke geometrickému nebo barevnému zkreslení. Na obrázku 4.18 je znázorněno několik možností pokrytí pixelu barvami dvou objektů. Uvažujeme-li pixel s jednotkovou plochou a pokrytí objektem A a B označíme α_A, α_B, pak odhadneme velikost plochy, na které není žádný z obou objektů, jako $(1-\alpha_A)(1-\alpha_B)$. Pravděpodobné velikosti ploch a možné barvy pro další kombinace objektů a jejich průhledností jsou uvedeny v tabulce 4.1.

objekty na ploše	obsah plochy	možné barvy
žádný objekt	$(1-\alpha_A)(1-\alpha_B)$	0
pouze A	$\alpha_A(1-\alpha_B)$	0,A
pouze B	$\alpha_B(1-\alpha_A)$	0,B
A a B	$\alpha_A\alpha_B$	0,A,B

Tabulka 4.1: Pravděpodobné velikosti ploch při α-míchání

Pokud tedy umíme uchovat ke každému pixelu atribut α, pak lze jednoduše realizovat kombinace obrazů podle variant uvedených na obrázku 4.19. Výsledná barva pixelu c bude určena jako lineární kombinace výchozích barev c_A a c_B s použitím koeficientů F_A a F_B uvedených na obrázku 4.19. Tedy $c = F_A \cdot c_A + F_B \cdot c_B$ a nový atribut pokrytí pixelu bude $\alpha = F_A\alpha_A + F_B\alpha_B$.

Realizace α-míchání je podporována v hardwaru grafických karet jako tzv. α-buffer. Pixel je pak představován pěticí atributů (R, G, B, Z, α), kde Z je tzv. hloubka pixelu, viz část 11.2.1.

V souvislosti s alfa mícháním musíme zmínit ještě dvě důležité techniky, ve filmovém průmyslu často používané *klíčování na barvu* (*chroma keying*) a *klíčování na modrou* (*blue*

operace	barvy	diagram	F_A	F_B
mazání	(0,0,0,0)		0	0
A	(0,A,0,A)		1	0
B	(0,0,B,B)		0	1
A přes B	(0,A,B,A)		1	$1-\alpha_A$
B přes A	(0,A,B,B)		$1-\alpha_B$	1
část A, která překrývá B	(0,0,0,A)		α_B	0

operace	barvy	diagram	F_A	F_B
část B, která překrývá A	(0,0,0,B)		0	α_A
A kde není B	(0,A,0,0)		$1-\alpha_B$	0
B kde není A	(0,0,B,0)		0	$1-\alpha_A$
A nad B	(0,0,B,A)		α_B	$1-\alpha_A$
B nad A	(0,A,0,B)		$1-\alpha_B$	α_A
A xor B	(0,A,B,0)		$1-\alpha_B$	$1-\alpha_A$

Obrázek 4.19: Možné operace při α-míchání obrazů

screening). Při těchto operacích se jedna barva, nejčastěji modrá, se prohlásí za průhlednou. Vše, co má tuto barvu, je poté za pomocí počítače nahrazeno nějakým jiným obrazem, například pozadím. Se zjednodušenou podobou se setkáme ve formátu GIF, kde se zvolí jedna barva (přesněji jeden index) a ta se poté nezobrazuje.

Blue screening a chroma keying jsou ve filmové produkci běžné. Videoklip, kdy například zpěvačka zpívá na vrcholku hory, mezi běžícími koni atd. je obyčejně vyroben právě tímto způsobem. Klíčování se také používá pro spojování sekvencí natočených nezávisle, pro spojování sekvencí generovaných počítačem s reálnými herci atd. Druhou oblastí aplikací jsou virtuální televizní studia. Předpokládejme, že hlasatel moderuje nějaký pořad. Přenos může být odkudkoli, ale hlasatel je fyzicky v místnosti, která je natřená přísně definovanou modrou barvou a osvětlena přesně definovanými světly. Vše, co je modré, je v reálném čase nahrazeno videosekvencí. Tím se získá dojem, že hlasatel je fyzicky přítomen uvnitř dění.

4.4.2 Warping

Algoritmy warpingu obrazu můžeme rozdělit do dvou základních kategorií. Jednoúčelové algoritmy, které se zadají snadno několika parametry a mají jediný výstup. Příkladem jsou různé víry a vlny na obraze, se kterými se můžeme setkat v programech editace obrazů jako je Adobe Photoshop. Těmito algoritmy se zde nebudeme zabývat.

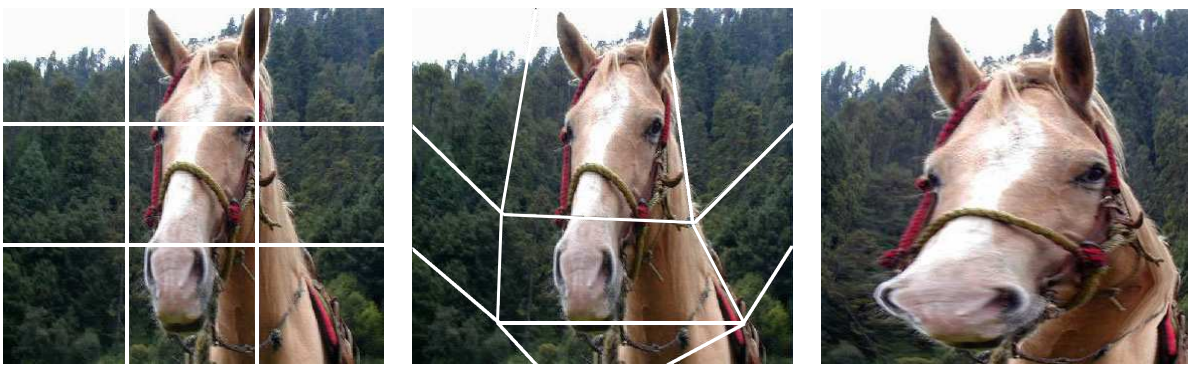
Druhá skupina algoritmů zahrnuje obecné metody, jež umožňují v podstatě libovolnou transformaci obrazu. V tomto odstavci se také nebudeme zabývat trojrozměrnými metodami,

jako je projekce na křivé zrcadlo. Jednotlivé techniky pracující ve dvou rozměrech se liší podle způsobu zadání a dnes existují dva základní algoritmy. První je založen na položení virtuální sítě na obraz. Tato síť svou změnou určuje i warping obrazu. Druhá metoda je založena na vkládání „magnetů“ do obrazu. Změna jejich polohy určuje transformaci. Síťový warping je vhodný pro globální změny v obraze a je nesnadné jím provést lokální operace, například zavření očí. Algoritmy z druhé skupiny jsou naopak vhodné právě pro lokální operace.

Síťový warping

Síťový warping publikoval v roce 1990 D. Smythe pracující v *Industrial Light Magic-ILM* [Smyt90]. Tato metoda je dnes standardní technikou a nalezneme ji prakticky v jakémkoli programu pro editaci obrazů. S výsledky se setkáme v reklamách a ve filmech a úplně poprvé byla použita v sérii filmů o Indiana Jonesovi.

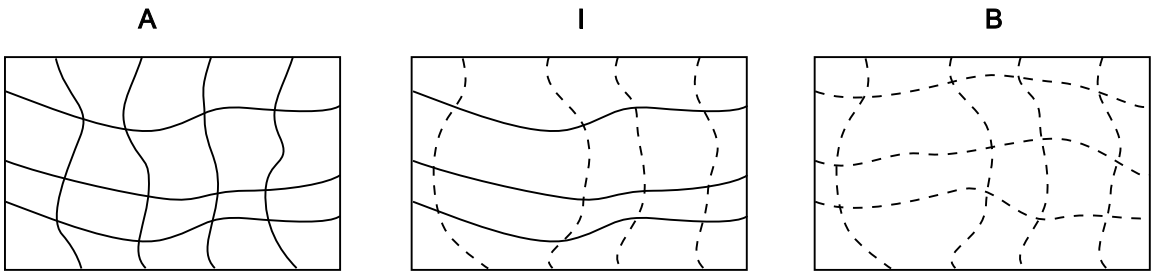
Warping se definuje pomocí pomyslné sítě, která se položí na obraz. Křivky sítě mohou být lomené čáry, častěji se používají Bézierovy kubiky či splajny. Uživatel položí síť na obraz a interaktivně určí nové polohy průsečíků. Oblast vymezená částí čtveřice křivek zůstane spojitá, změní se však její tvar (viz obrázek 4.20).



Obrázek 4.20: Síťový warping. Zleva: pokrytí sítí ve zdrojovém a cílovém obrazu a výsledný obrázek

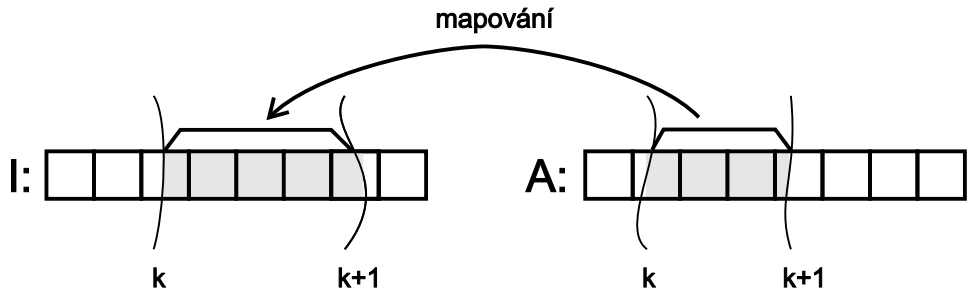
Vlastní algoritmus je dvouprůchodový a separabilní. Nejprve se aplikuje na řádky obrazu, čímž se získá meziobraz I. Poté se na sloupce meziobrazu aplikuje stejný algoritmus a tak se získá obraz výstupní. Označme počet řídicích křivek v horizontálním směru h a ve směru svislém v. Označme dále $x \times y$ rozlišení obrazu, se kterým pracujeme.

První průchod se skládá ze dvou kroků. Nejdříve se určí průsečíky svislých křivek s řádky obrazu. Tím se na řádcích vymezí intervaly (viz obrázek 4.22), které se ve druhém kroku



Obrázek 4.21: Notace pokrývacích sítí a obrazů pro síťový warping

převzorkují. Výstupem prvního průchodu je meziobraz I, ve kterém jsou na svých pozicích pixely ležící na řádcích. Definice sítí pro obraz A, I a B je na obrázku 4.21.



Obrázek 4.22: Mapování intervalu na řádku v síťovém warpingu

Předpokládejme, že pracujeme s intervalem, který je na nějakém řádku vymezen svislými křivkami s indexy k a $k+1$. Obrázek 4.22 ukazuje, jak je interval z obrazu A mapován do obrazu I. Při tomto přiřazení se hodnoty pixelů nejčastěji lineárně interpolují. Složitější je situace v okrajových pixelech, kterými procházejí křivky sítě. Protože souřadnice pixelu jsou celočíselné, jejich plocha je nenulová a křivka prochází pixelem v libovolném místě, můžeme určit velikost plochy pixelu, která ještě k intervalu patří. Tato velikost se vyjádří jako průhlednost α a představuje relativní pokrytí pixelu. Krajní body tedy nepřispívají svou celkovou intenzitou, ale intenzitou sníženou o procento pokrytí. Právě práce s částečným pokrytím pixelu, v kombinaci s kvalitním algoritmem převzorkování, je zárukou vysoké kvality výsledných obrazů.

Druhý průchod se liší od prvního pouze tím, že jeho vstupem je meziobraz I a výstupem obraz B, a tím, že se výpočet převzorkování provádí pro sloupce, které jsou určeny horizontálními křivkami.

Doposud jsme předpokládali, že vypočítáváme jediný snímek sekvence. Při výpočtu posloupnosti snímků, se pro každý snímek vypočítá nové pokrytí obrazu sítí, která vznikne (obyčejně lineární) interpolací průsečíků původní sítě a sítě cílové.

Mezi nevýhody síťového warpingu patří to, že v podstatě malá změna řídicí sítě ovlivní větší část obrazu. Je-li síť příliš řídká, jsou oblasti velké a dochází k modifikaci rozsáhlejších částí obrazu. Tomu lze zamezit zvýšením hustoty sítě, čímž sice dojde ke zpřesnění sítě v okolí detailu, ale také ke snížení přehlednosti editace a v konečném důsledku i ke snížení rychlosti výpočtu. Křivka je položena přes celý obraz a tak dochází k neefektivnímu přepočítání některých oblastí. Další nevýhodou je, že některé poměrně intuitivní lokální operace, například změna polohy koutků úst, jsou obtížně realizovatelné.

Úsečkový warping

Úsečkový warping (*feature based warping*) se používá zejména v případech, kdy síťový warping selhává. Hodí se tedy především pro lokální změny v obraze – například zavření očí.

Podstatou úsečkového warpingu je modifikace obrazu na základě úseček, které svou velikostí a polohou definují lokální změnu obrazu. Úsečky můžeme chápat jako magnety, které vytvářejí magnetické pole, a proto se této metodě říká též warping pomocí pole (*field warping*). Změnou polohy magnetů definujeme změnu tvaru pole a vzájemná korespondence mezi oběma poli definuje příslušnou transformaci. Stejně jako v předchozím případě budeme předpokládat vstupní obraz A a výstupní B. Pro výpočet pixelu v obraze B použijeme metodu zpětného mapování, vypočítáme hodnotu pixelu ve výstupním obraze B na základě oblasti z obrazu A (viz obrázek 4.10).

Princip metody ukážeme nejprve na transformaci s jedním párem úseček. První z nich je určena body P' a Q' a leží ve zdrojovém obraze A (viz obrázek 4.23), druhá leží v cílovém obraze B a je určena body P a Q. Předpokládejme, že všechny body určující úsečky jsou ve středech pixelů, mají tedy celočíselné souřadnice.

Při mapování procházíme všechny pixely z obrazu B a hledáme jejich vzory v A. Označme X pixel v obraze B a X' bod v obraze A. Algoritmus pracuje s reálnými čísly a tak bod X' může ležet v obraze kdekoliv. Pro výpočet jeho hodnoty použijeme některou z metod převzorkování obrazu z odstavce 4.3.1. Použitá metoda výpočtu hodnoty bodu X' ovlivňuje kvalitu výsledného obrazu.

Z úsečky určené body P' a Q' nejprve vypočítáme hodnoty označené u a v (viz obrázek 4.23). Hodnota u určuje polohu podél úsečky a vypočítá se:

$$u = \frac{(X - P) \cdot (Q - P)}{\|Q - P\|^2}. \tag{4.12}$$

Vztah v čitateli zlomku je skalární součin dvou vektorů a určuje projekci vektoru $\vec{XP}$ do vektoru

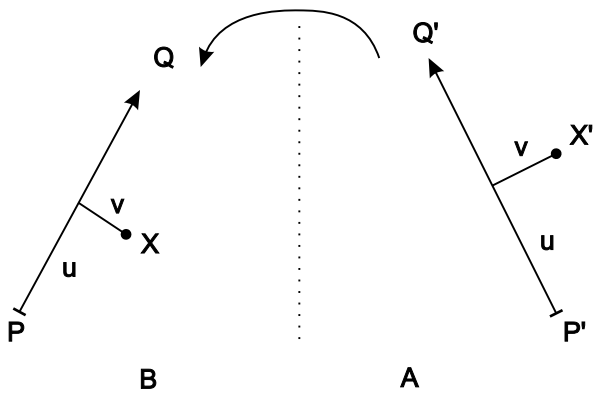
$\vec{QP}$. Hodnota v je vzdáleností bodu X od přímky PQ a vypočítáme jí jako

$$v = \frac{(X-P)\cdot(Q-P)_\perp}{\|Q-P\|}, \tag{4.13}$$

kde operátor $\perp$ označuje kolmý vektor. Poznamenejme, že ve dvojrozměrném prostoru se kolmý vektor k vektoru $\vec{p} = (x, y)$ získá jako $\vec{p_\perp} = (-y, x)$ resp. $\vec{p'_\perp} = -\vec{p_\perp} = (y, -x)$. Hodnoty u a v v rovnici (4.13) určují polohu pixelu X vzhledem k úsečce PQ, a také polohu bodu X' vzhledem k úsečce $P'Q'$. Jeho souřadnice vypočítáme podle vztahu

$$X' = P' + u(Q'-P') + \frac{v.(Q'-P')_\perp}{\|Q'-P'\|}. \tag{4.14}$$

Aplikací výše uvedeného vztahu zjistíme, že tento postup určuje pro celý obraz otočení, posun a změnu měřítka ve směru úsečky. Pomocí této transformace není možné provést změnu měřítka ve směru kolmém na definující úsečku.



Obrázek 4.23: Princip mapování pixelů metodou úsečkového warpingu

Warping obrazu se provádí s více úsečkami. Jediný pár určuje jednoznačné přiřazení bodu z obrazu A do pixelu v obraze B. Při aplikaci více úseček (viz obrázek 4.24) nalezneme hodnoty u a v pro všechny úsečky a jeden pixel X v obraze B. Korespondující úsečky v obraze A těmto hodnotám přiřazují *množinu* bodů $\{X'_i, i = 1, 2, \ldots, n\}$, kde n je počet párů. Poloha pixelu X' v obraze B je určena jako vážený součet všech těchto příspěvků. Váha každého vzorku se určí z vektoru posunutí $\vec{D}_i$ nového bodu X'_i a jeho originálu X a z váhy w_i, která je vysvětlena níže.

Označme P'_i a Q'_i, $i = 1, ..., n$ body určující úsečky v obraze A a P_i, Q_i jim korespondující koncové body úseček v obraze B. Pro každou úsečku P_iQ_i nalezneme hodnoty u_i a v_i podle vztahu (4.12) a (4.13). Souřadnice bodu X' určíme ze souřadnic všech příspěvků jako součet

$$X' = X + \frac{\sum_{i=1}^{n} \vec{D}_i \; w_i}{\sum_{i=1}^{n} w_i},$$

kde $\vec{D}_i$ je vektor daný bodem X'_i a X a w_i je váha příspěvku X'_i, kterou získáme jako

$$w_i = \left(\frac{|Q'_i - P'_i|^p}{(a + dist)}\right)^b. \tag{4.15}$$

Zde *dist* je vzdáleností bodu X od úsečky P_iQ_i a je určena

$$dist = \begin{cases} abs(v) & pro: \ u \in \langle 0,1 \rangle \\ |P,X| & pro: \ u < 0 \\ |Q,X| & pro: \ u > 1 \end{cases}$$

Konstanty a, b a p jsou zadány uživatelem a určují vliv jednotlivých úseček na celkovou transformaci. Dobrých výsledků lze dosáhnout při volbě $a = 0.001$, $b = 2$ a $p = 0.5$. Hodnota a by měla být blízká nule a určuje vliv úsečky na transformovaný bod. Čím blíže je bod k úsečce v originálu, tím blíže je jí i ve výsledném obraze. Konstanta b by měla být větší nežli nula a říká, jak je pixel ovlivněn především úsečkami ve svém okolí, neboli jak vliv magnetu slábne se vzdáleností. Pro $b = 0$ je každý pixel ovlivněn všemi úsečkami stejně, protože $w_i = 1$. Hodnota p svazuje vliv úsečky a její délku a měla by měl ležet v intervalu $\langle 0, 1 \rangle$. Hodnoty blížící se nule způsobují, že všechny úsečky mají stejný vliv na warping, čím vyšší je hodnota p, tím větší význam mají dlouhé úsečky. Algoritmus 4.6 popisuje výpočet bodu v novém obraze.

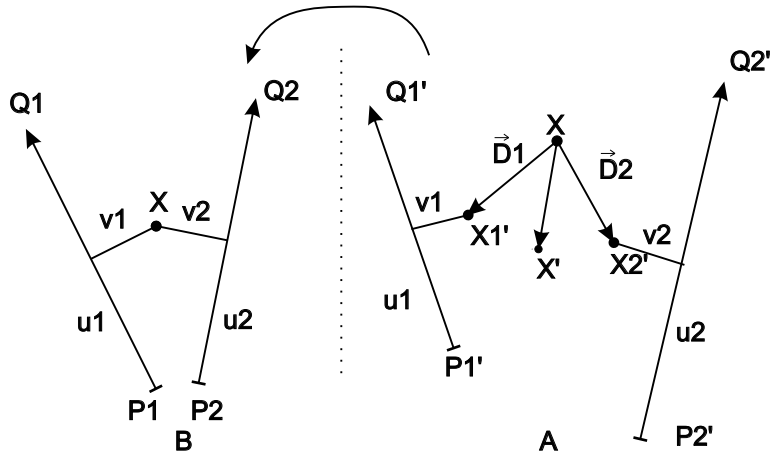
1. Pro všechny pixely v X výstupním obraze:
   (a) Vynuluj akumulovaný vektor součtu $\vec{d}_{sum}$ a akumulovaný součet vah w_{sum}
   (b) Pro všechny dvojice úseček P_iQ_i a $P'_iQ'_i$.
      i. Z P_i a Q_i vypočti u_i a v_i
      ii. Z P'_i, Q'_i, u_i a v_i vypočítej souřadnice X'_i podle vztahu (4.14)
      iii. Urči vektor $\vec{D}_i = X'_i - X$.
      iv. Urči vzdálenost *dist* mezi X a P_iQ_i.
      v. Spočítej váhu w podle vztahu (4.15).
      vi. Spočítej akumulovaný vážený součet vektoru $\vec{d}_{sum} = \vec{d}_{sum} + w\vec{D}_i$
      vii. $w_{sum} = w_{sum} + w$
2. Urči souřadnice $X' = X + \vec{d}_{sum}/w_{sum}$
3. Převzorkuj oblast určenou souřadnicemi X'

Algoritmus 4.6: Feature based warping

Výše uvedený algoritmus se používá pro výpočet jediného obrazu. Postupná transformace obrazu se vypočítá změnou poloh koncových bodů úseček zpravidla lineární interpolací. Tento způsob může vést k nežádoucímu zkrácení úsečky, které lze odstranit interpolací úhlu úsečky v polárních souřadnicích.

Warping pomocí korespondujících úseček je intuitivní a s jeho pomocí lze poměrně rychle

docílit požadovanou transformaci obrazu. Jednou z nevýhod je explicitní zákaz křížení magnetů, který vede ke vzniku nežádoucích „bublin“ tzv. *Ghostbustering effect*[3].



Obrázek 4.24: K výpočtu warpingu s více páry úseček (viz text)

4.4.3 Morfing

Morfingem rozumíme přechod od vstupního obrazu A k výstupnímu obrazu B warpingem. Zatímco u warpingu jsme zadávali jeden obraz a určovali jeho změnu, u morfingu musíme zadat předlohy dvě, označme tyto obrazy A a B. Výstupem algoritmů morfingu je sekvence přechodu od A k B tak, jak můžeme vidět například na obrázku 4.25.

Základem morfingu je nějaký algoritmus kroucení obrazu, ať již síťový warping, warping založený na úsečkách, či nějaký jiný algoritmus. Při morfingu se vypočítávají dva obrazy. První je obraz vzniklý warpingem z obrazu A, označme ho A_i a druhý je obraz vzniklý warpingem z obrazu koncového, označme ho B_i. Výsledný obraz sekvence se získá prolnutím obrazů A_i a B_i. Tyto dočasné obrazy se získají warpingem a jeho definice se určí z počáteční a koncové sítě, nejčastěji lineární interpolací řídicích prvků. U síťového warpingu se interpolují polohy průsečíků, u úsečkového warpingu koncové polohy bodů.

[3]Název tohoto jevu pochází ze stejnojmenného filmu Ghostbusters (Krotitelé duchů), kde se odehrává následující dialog:
„Don’t cross the streams!“
„Why?!“
„It would be bad.“

Obrázek 4.25: Příklad morfingu pomocí úseček. Na horním řádku je cílový a koncový obraz s korespondujícími úsečkami, dole pak odpovídající sekvence obrazů morfingu oblíbeného pedagoga v hororové monstrum.

Algoritmus morfingu demonstrujeme na příkladě, který využívá síťového warpingu. Uživatel nejprve zadá polohy řídicí sítě v obraze A a v obraze B. Poté definuje, kolik snímků přechodu se má vygenerovat. Algoritmus pro daný $i-$tý snímek vygeneruje interpolovanou síť z A a B. Ta se získá jednoduše interpolací poloh průsečíků. Tato síť se aplikuje na obraz A a tím se získá A_i. Tatáž síť se aplikuje na obraz B a získá se B_i. Poté se oba obrazy sečtou.

Asi nejdůležitější při morfingu je zadání korespondujících sítí. Odpovídající objekty v obou obrazech by si měly odpovídat a definující sítě by měly být podrobné, aby nedošlo k nežádoucím efektům a změnám. Velkou pozornost je zapotřebí věnovat pozadí. Obrázek 4.25 záměrně demonstruje chybu způsobenou nevhodnou volbou pozadí. Pozadí jsou jiná a tak na snímcích dochází k jejich prolnutí. To ve výsledné animaci působí rušivě. Jedním z možných řešení je

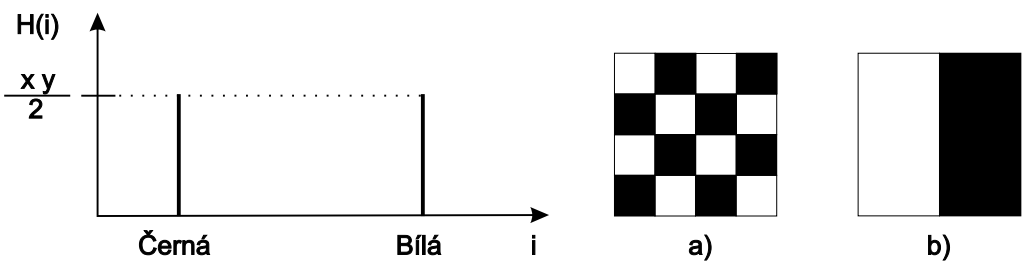
vyjmout jeden z objektů a dát ho do obrázku se stejným pozadím. Objekty by rovněž měly mít odpovídající velikost, aby nedocházelo k jejich zvětšování či zmenšování.

4.5 Histogram

Důležitým klíčem k charakterizaci obrazu je histogram, který kvantifikuje množství a frekvenci barev obsažených v obraze. Matematicky je histogram vektor absolutních četností hodnot zastoupených v obraze. Jinými slovy řečeno, hodnota histogramu H pro index i říká, kolik pixelů v obraze má intenzitu i. Pokud je původní obraz složen z jasových složek, je histogram jednorozměrný vektor, v případě složek RGB je složen ze tří vektorů, atp. Příklady histogramů jsou na obrázku 4.27. V dalším textu budeme uvažovat pouze šedotónové obrazy a tudíž pouze jediný histogram. Obrazy, které reprezentují barvu pomocí více barevných kanálů mají pro každý kanál zvláštní histogram. Někdy bývá užitečné navíc reprezentovat i kanál, který odpovídá velikosti jasu.

Je-li rozlišení obrazu $x \times y$, je v něm celkem $x \times y$ pixelů. Protože histogram reprezentuje absolutní četnosti jednotlivých složek, platí, že hodnota histogramu pro index i, tj. hodnota odpovídající určité barvě šedi, je rovna počtu pixelů této hodnoty. Z tohoto faktu potom plyne vztah (4.16): součet hodnot histogramu je roven množství pixelů v obraze.

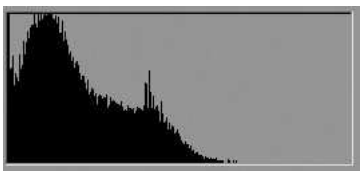
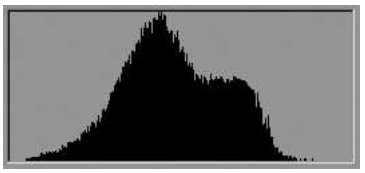
$$\sum_{i=0}^{max} H(i) = x \cdot y \tag{4.16}$$



Obrázek 4.26: Dva rozdílné černobílé obrazy (vpravo) se stejným histogramem (vlevo)

Histogram je statistickou veličinou. Můžeme ho také chápat jako pravděpodobnost výskytu pixelu barvy v obraze. Histogram kvantifikuje jasové poměry v obraze a nenese žádnou informaci o jejich plošném rozložení. Histogram na obrázku 4.26 je získán z obrazu skládajícího se ze dvou barev, z nichž obě jsou zastoupeny stejně, tj. padesáti procenty. Takový histogram mají oba obrazy a) i b).

Tmavé odstíny se nacházejí blízko nulové hodnoty a s rostoucí hodnotou ve směru vodorovné osy roste i intenzita. Zcela vlevo je tedy barva černá a zcela vpravo maximální intenzita, obyčejně odpovídající bílé barvě. Přestože histogram nenese údaje o poloze pixelů v obrazu, lze z něj vyčíst důležité informace. Tak například z histogramu na obrázku 4.27a) je patrné, že patří jasnému obrazu, neboť nejvyšší hodnoty, to jest barvy s nejvyšší frekvencí výskytu, převládají v jeho horní části. Na obrázku 4.27b) je příklad histogramu, ve kterém převažují tmavé složky. Graf na obrázku 4.27c) je histogramem obrazu s nízkým kontrastem.

a)  b)  c) 

Obrázek 4.27: Příklady histogramů

Podle tvaru histogramu se provádí i základní klasifikace obrazu. Definiční obor histogramu se rozdělí na tři stejně velké části. Dolní oblast se označuje jako stíny (*shades*), střední část jako střední tóny (*mid-tones*) a konečně nejjasnější oblasti se říká světla (*lights*). Jak názvy napovídají, reálné stíny objektů přispívají převážně k intenzitám v dolní oblasti, přítomnost světel a jasných ploch se projeví v oblasti vysokých intenzit a zbytek se projeví ve střední části.

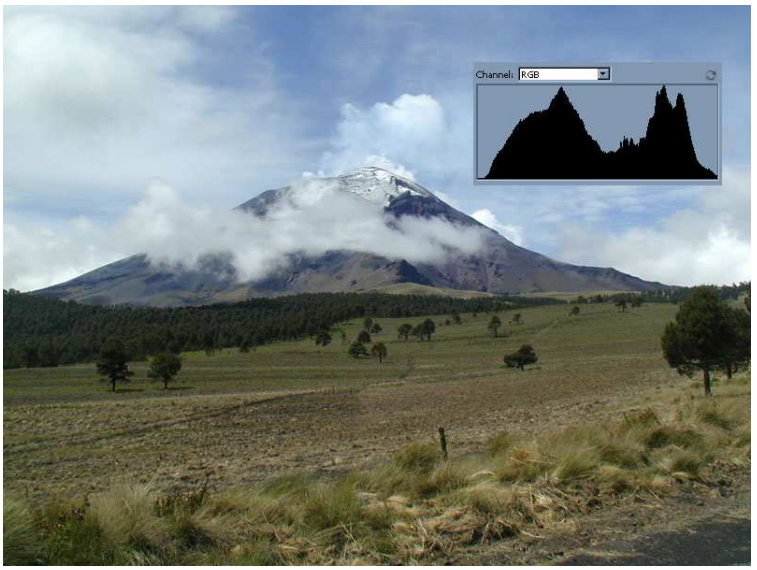
Klasifikace obrazu pomocí histogramu rozlišuje čtyři základní druhy obrazů. Jasný obraz (*high-key*) má převážnou většinu barev přítomnou ve světlech. Jeho opakem je tmavý obraz, označovaný jako (*low-key*). Středotónový obraz (*mid-key*) má většinu barev zastoupenou kolem střední hodnoty. Posledním typem je obraz s vysokým kontrastem. Velikost kontrastu je dána mírou rozdílu mezi středními hodnotami na jedné straně a světly a stíny na druhé. Z tohoto důvodu se středotónový obraz také označuje jako obraz s nízkým kontrastem. Histogram, ve kterém se nacházejí dva ostré vrcholy, se nazývá *bimodální*.

Typickým příkladem obrazu s vysokým kontrastem je fotografie města v noci. Překvapivě se nejedná o tmavý snímek. Světla, která jsou v obraze přítomná, kompenzují tmavé oblasti. Podobně fotografie detailů ostře osvětlených objektů obyčejně mají vysoký kontrast. Příkladem může být detail sluncem osvětlené rostliny. Přímo osvětlené oblasti přispívají do oblasti jasu, vržené stíny pak do oblasti nízkých intenzit. Obraz s nízkým kontrastem získáme snadno, když vyfotografujeme osvětlenou krajinu z velké vzdálenosti. Stíny, stejně jako osvětlené plochy, mají statisticky téměř gaussovské rozložení, stejně je tomu i s jejich jasovými hodnotami v histogramu takto získaného obrazu.

Další důležitou vlastností každého obrazu je *bílý* a *černý* bod. Za bílý bod se považuje

nejsvětlejší barva v obraze, nemusí se nutně jednat o bílou barvu. Analogicky je tomu s černým bodem. Obě hodnoty snadno vyčteme z histogramu jako jeho mezní nenulové hodnoty.

Histogram je důležitou pomůckou v digitální fotografii. Kvalitní digitální fotoaparáty dokonce zobrazují histogram obrazu ještě před jeho sejmutím. Z histogramu je obtížné vyčíst, zda bude obraz dobrý či špatný. Lze z něj však jasně vyčíst, zda byl obraz proveden technicky dobře. Technicky dobrý obraz je takový, který využívá celou škálu intenzit. Pokud například na digitálním fotoaparátu nastavíme špatně citlivost, může dojít k tomu, že využijeme jen část barevných intenzit. Takto zkažený obraz lze částečně opravit pomocí programů pro editaci obrazů, ale chybějící informaci již dodat nemůžeme.



Obrázek 4.28: Fotografie dýmajícího Popocatépetlu jako příklad obrazu s vysokým kontrastem

Práce s histogramem je důležitá také při převodu HDR obrazů (viz kapitola 4.2) do běžné šedotónové stupnice.

Histogram je důležitou informací pro některá optická zařízení, jako jsou kamery či filmové scanery. Existují speciální obrázky, tzv. etalony, jejichž histogramy při správném sejmutí jsou známy a pomocí kterých se tato zařízení jasově a barevně kalibrují. Z odchylky histogramu obrazů kalibračních etalonů lze pak vyčíst, k jakým změnám v optice zařízení došlo a následně tyto změny opravit.

Histogram nese velmi důležité informace o obraze. Z tohoto důvodu se jeho implementací mohou pochlubit i leckteré grafické karty a také se s ním setkáme v OpenGL Imagining Extension. Toto rozšíření rovněž obsahuje funkce pro rychlé mapování barev pomocí mapovacích funkcí a tabulek tak, jak o něm pojednává následující odstavec.

4.5.1 Změny histogramu

Operace, které mají za svůj důsledek změnu histogramu, bývá zvykem označovat jako *operace s histogramem*. I když editační programy zobrazují tyto operace jako přímé zásahy do histogramu, ve skutečnosti se jedná o operace, jejichž výsledkem jsou změny histogramu.

Nejčastěji se manipulace s barevnými složkami provádějí pomocí editačních křivek. Tyto křivky přiřazují určité vstupní hodnotě hodnotu výstupní a nejčastěji se reprezentují pomocí jednorozměrné tabulky. Z tohoto důvodu se jim říká vyhledávací tabulka (*look-up table*). Mapovací křivka je funkce (4.17), která vstupní úrovni jasu i přiřazuje výstupní úroveň i':

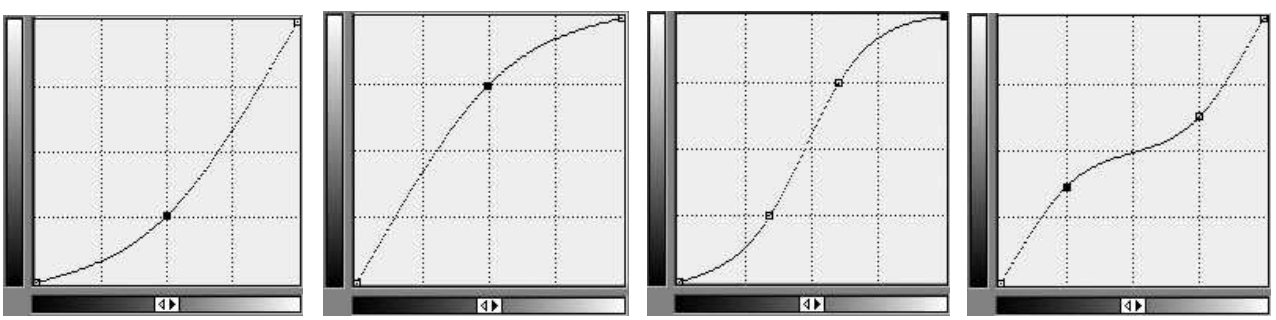
$$i' = f(i). \tag{4.17}$$

Funkce f může více hodnotám z definičního oboru přiřadit jedinou funkční hodnotu. Pokud je několik jasových složek nahrazeno jedinou, dochází k omezení jasové stupnice a tím ke zvýšení celkového počtu pixelů určité hodnoty jasu.

Programy pro práci s rastrovými obrazy nabízejí širokou škálu editačních nástrojů pro přímou změnu vyhledávací tabulky. Obyčejně můžeme pracovat s jednotlivými barevnými kanály zvlášť, nebo najednou, můžeme editovat jas obrazu, atp. Vyhledávací tabulku můžeme měnit tak, že specifikujeme křivku přímo (viz například obrázek 4.29), nebo ji interaktivně malujeme. V dalším textu uvedeme některé často používané operace s histogramem.

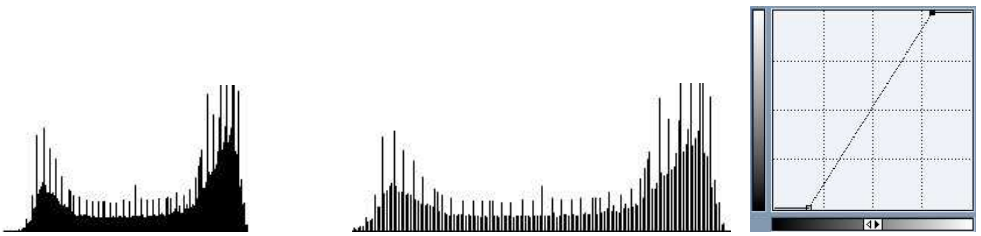
Zvýšení a snížení jasu, změny kontrastu

Zvýšení a snížení jasu lze snadno provést aplikací funkce, jejíž graf je na obrázku 4.29. Jediné dvě hodnoty, které zůstanou zachovány, jsou černá a bílá barva. Ostatní hodnoty jsou sníženy či zvýšeny.



Obrázek 4.29: Ztmavení, zesvětlení, vyšší a nižší kontrast pomocí křivek. Vstupní intenzita je na ose x, výstupní na ose y a korekční křivka je definována několika řídicími body

Zvýšení resp. snížení kontrastu obrazu se provádí pomocí tzv. s-křivky (*s-shaped curve*) resp. inverzní s-křivky. Název této křivky je odvozen z jejího tvaru. S-křivka přiřadí stínům tmavší odstíny, střední část zůstane zachována a světlům se naopak zvýší intenzita. Analogicky mění jasové intenzity křivka snižující kontrast.



Obrázek 4.30: Nevyrovnaný histogram před a po nastavení černého a bílého bodu pomocí křivek

Pomocí křivek můžeme rovněž změnit černý a bílý bod v obraze. Stačí posunout minimální hodnotu směrem do středu grafu, tedy k nejtmavější barvě v obraze a stejně tak maximální hodnotu k nejsvětlejší jak je vidět na obrázku 4.30. Tímto způsobem se původní histogram roztáhne. Je zřejmé, že roztažením histogramu v něm vzniknou prázdná místa. Některé hodnoty jasu prostě nebudou v pozměněném obraze přítomné. Chybějící informace z okrajů se pouze rovnoměrně distribuuje. Jinou možností, která v podstatě neroztáhne histogram rovnoměrně, ale bere v úvahu statistické rozložení barev, je ekvalizace histogramu uvedená na straně 162.

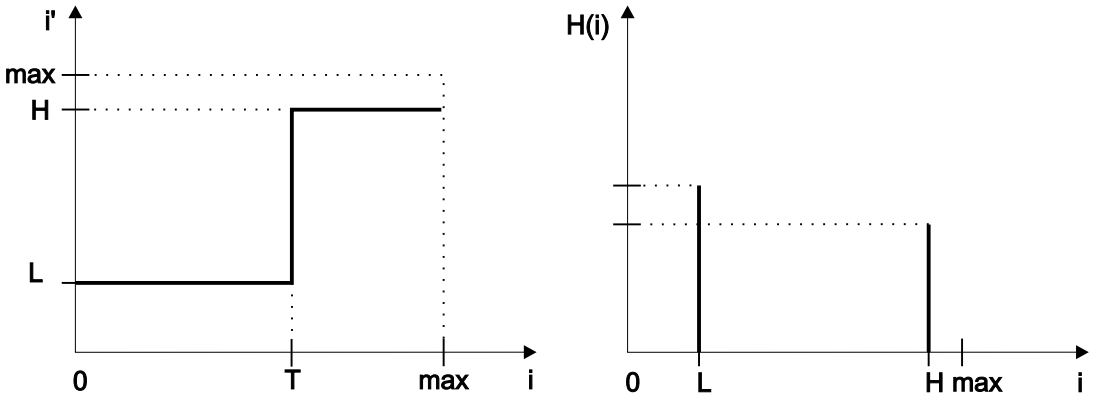
Na obrázku 4.30 je vidět, že nový histogram obsahuje prázdná místa. Jednou z možností, jak lze tuto vadu alespoň částečně odstranit, je nepracovat s 8bitovou reprezentací barevné informace pro každý barevný kanál. Před provedením operace s histogramem je výhodné převést obraz do 12bitové či 16bitové reprezentace. Po provedení transformace se obraz převede zpět do původní 8bitové reprezentace. Zvýšením bitové přesnosti se zmenší chyby prováděných matematických operací.

Prahování

Operace prahování (*thresholding*) spočívá v rozdělení jasové škály na dvě části a nahrazení každé z částí jedinou hodnotou tak, jak je uvedeno v následujícím vztahu

$$i' = \begin{cases} L & pro\ i < T \\ H & pro\ i >= T, \end{cases} \tag{4.18}$$

kde i je vstupní hodnota, i' je hodnota výstupní, T je práh a L a H jsou dvě výstupní hodnoty. Odpovídající graf je uveden na obrázku 4.31 vlevo. Všechny hodnoty s intenzitou menší než



Obrázek 4.31: Vyhledávací tabulka prahování (vlevo) a příklad výsledného histogramu

je práh T jsou nahrazeny hodnotou L, intenzity vyšší a rovné hodnotě prahu jsou nahrazeny hodnotou H. Výsledný obraz obsahuje maximálně dvě barevné intenzity. Při nevhodné volbě prahu, zvolíme-li například hodnotu prahu vyšší nežli je bílý bod, je celý obraz složen z pixelů jediné barvy. Poznamenejme, že volba parametru T má zásadní význam pro kvalitu výsledného obrazu. Vhodná hodnota prahu může být rovna například mediánu, či 50 % šedé.

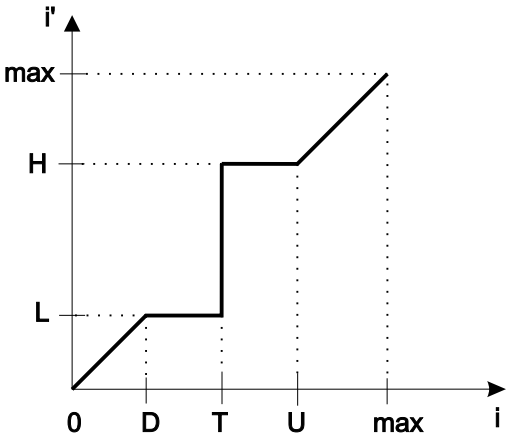
Prahování se používá pro převod obrazu do binární formy. Může se rovněž použít pro vytvoření masky, při převodu naskenovaných předloh dokumentů atd.

Obrázek 4.32 je grafem tzv. *ohraničeného prahování*. Pracuje se se třemi vstupními hodnotami $D < T < U$ a se dvěmi výstupními L a H. Hodnoty $0 \leq i \leq D$ a hodnoty $U \leq i \leq max$ zůstanou beze změny. Hodnoty $D \leq i < T$ jsou nahrazeny hodnotou L a hodnoty z intervalu $T \leq i < U$ se nahradí hodnotu H.

$$i_n = \begin{cases} i & pro\ 0 \leq i < D \\ L & pro\ D \leq i < T \\ H & pro\ T \leq i < U \\ i & pro\ U \leq i \leq max \end{cases} \tag{4.19}$$

Gama korekce

Dnes patrně dosluhující vakuové obrazovky, označované jako CRT (*Cathode Ray Tube*), mají nelineární jasovou odezvu intenzity fosforů na stínítku v závislosti na vstupním napětí katody. Křivka, která tuto nelinearitu charakterizuje, zhruba odpovídá mocnině 2.5. Jinými slovy řečeno, pokud chceme zobrazit intenzitu i, ve skutečnosti se zobrazí $i^{2.5}$. Intenzita napětí je v rozsahu



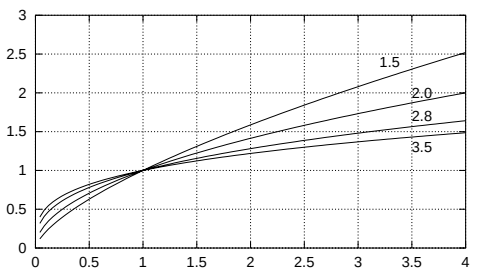
Obrázek 4.32: Ohraničené prahování

od nuly do jedné a tak tato nelinearita obraz ztmavuje. Podstatnější vlastností je, že odezva není lineární. Samozřejmým požadavkem je, aby intenzita 0.5 byla dvojnásobkem intenzity 0.25 a tak je nutné se s touto vlastností nějak vypořádat.

Operace, která tuto nelinearitu odstraňuje, se jmenuje *gama korekce* a vyhledávací tabulka této funkce se vypočítá podle vztahu

$$i' = i^{1/\gamma}. \tag{4.20}$$

Hodnota γ je závislá na typu obrazovky. Většina monitorů, a stejně tak i televizních obrazovek, má tuto konstantu předem nastavenou. Kvalitnější monitory se za pomoci speciálního čidla mohou kalibrovat automaticky, protože hodnota zkreslení závisí kromě jiného i na teplotě. Hodnota bývá $\gamma = 2.5 \pm 0.3$. Graf na obrázku 4.33 ukazuje křivky pro některé typické hodnoty γ.



Obrázek 4.33: Gama korekce.

Gama korekce je nezbytná z mnoha důvodů z nichž je nejdůležitější přenositelnost obrazů. Chceme, aby se jeden obrázek zobrazil stejně na libovolném zobrazovacím zařízení. Nelinearita podstatně ovlivňuje dithering. Nesprávně nastavená gama korekce způsobuje problémy s antialiasingem, hrana správně vyhlazená na jedné obrazovce může být roztřepená na jiné proto, že barevné pixely vyhlazující nerovnosti jsou zobrazeny s jinou intenzitou.

Ekvalizace histogramu

Jednou z často používaných metod změny jasu v obraze je vyrovnání, neboli ekvalizace histogramu. S podobnou operací jsme se již setkali při nastavení černého a bílého bodu obrazu. V tomto případě však šlo o roztažení histogramu tak, aby rovnoměrně vyplňoval požadovaný rozsah hodnot. Vyrovnání histogramu je složitější, avšak v principu podobné. Jde o nalezení mapovací funkce, která rozloží hodnoty v histogramu rovnoměrně. Výsledkem by tedy měl být jasově vyrovnaný obraz.

Obrázek 4.34: Rozdíl mezi ekvalizací histogramu a změnou černého a bílého bodu. Původní, jasný obrázek zcela vlevo, byl pořízen s nesprávně nastavenou citlivostí fotoaparátu. Nastavením černého a bílého bodu (uprostřed) získáme odlišnou barevnou dynamiku, nežli ekvalizací histogramu (vpravo)

Algoritmus ekvalizace histogramu spočívá v nalezení takové mapovací funkce, jejíž aplikací dosáhneme vyrovnaného histogramu. Je zřejmé, že tato funkce závisí na konkrétním obraze. Stejně jako v předcházejících případech budeme předpokládat, že máme obraz v rozlišení $x \times y$ bodů maximální intenzity max. Ideálně vyrovnaný histogram by obsahoval všechny hodnoty zastoupené stejnou četností, označme ji d. Hodnota d musí být průměrem ze všech hodnot, tedy

$$d = \frac{x \times y}{max}, \tag{4.21}$$

Připomeňme, že histogram neříká nic o tom, kde tyto body leží, ale pouze kvantifikuje jejich přítomnost. Jak bylo výše uvedeno, z vlastností vyhledávací tabulky vyplývá, že není možno některé intenzity rozdělit, tj. není možné rozdělit některou z hodnot histogramu. Jinými slovy řečeno, pokud některá jasová složka překračuje ideální hodnotu d, nemůžeme ji pomocí mapovací funkce zkrátit. Protože mapovací funkce není funkce prostá, můžeme však dvě a více hodnot sloučit do jedné.

Základní myšlenkou algoritmu, který vytváří mapovací funkci, je, že se chová jako by bylo možno hodnotu vyšší než d rozdělit. Algoritmus nejprve prozkoumá, do jakého intervalu je

Vstup: *histogram*
Výstup: mapovací funkce f definovaná tabulkou

1. Nastav $d = x \times y/max$
2. Vynuluj čítače $outI$ a $outHeight$
3. Pro všechny intenzity jasu ($in\ i = 0, \ldots, max$)
   (a) nastav proměnnou $firstI$ na hodnotu $outI$
   (b) nastav vstupní výšku histogramu $inHeight$ na $histogram[inI]$
   (c) dokud je $inHeight$ větší nežli $d - outHeight$, dělej:
      i. $inHeight = inHeight - (d - outHeight)$
      ii. zvyš $outI$ o jednu
      iii. nastav $outHeight$ na nulu
   (d) $outHeight$ zvyš o $inHeight$;
   (e) nastav mapovací funkci f[inI] na polovinu nalezeného intervalu $(firstI + outI)/2$

Algoritmus 4.7: Výpočet mapovací funkce vyrovnávající histogram

zapotřebí tuto hodnotu rozdělit a nerozdělenou ji potom umístí do jejího středu. Výpočet mapovací funkce vyrovnávající histogram demonstruje algoritmus 4.7. Tento algoritmus není jediný možný, existují i jiné přístupy [Gonz87].

K ekvalizaci histogramu, ostatně jako ke všem operacím s obrazem, je zapotřebí přistupovat uváženě, neboť i tato metoda má v některých případech nežádoucí efekty. Předpokládejme, že máme obrázek, který má na zcela černém pozadí tmavý šum. Ekvalizací histogramu docílíme toho, že se hodnoty šumu přesunou do jasnější oblasti a šum se tak zvýrazní.

V některých případech má smysl ekvalizovat pouze část obrazu a tuto operaci označujeme jako lokální ekvalizaci. Příkladem aplikační oblasti, ve které se lokální ekvalizace hojně využívá, je astronomie. Na fotografiích vesmírných objektů zabírá podstatnou část obrazu obloha, která je tmavá, nenese žádnou informaci, a tudíž nemá smysl zabývat se její ekvalizací.

Ekvalizaci v barevném prostoru je možno provést mnoha způsoby. Nejjednodušší metodou je ekvalizovat každý barevný kanál odděleně. Tato metoda však může vést k nežádoucím barevným změnám v obrázku. Byla-li například předloha barevně zcela vyvážená v červené i zelené barvě a modrá převládala v dolní části histogramu, dojde ekvalizací k přidání vysokých intenzit modré barvy.

Alternativní možností je provést ekvalizaci v jiném barevném prostoru. Výhodný je model YIQ popsaný v kapitole 1, který pracuje s jasem. Protože chceme mít obraz jasově vyrovnaný,

1. Spočítej histogram pro jas podle vztahu Y=0.299R+0.587G+0.114B
2. Vypočítej mapovací funkci f ekvalizující jasový histogram
3. Pro všechny pixely v obraze udělej
   (a) spočítej koeficient $y = f[jas\_pixelu]/jas\_pixelu$
   (b) vynásob hodnoty $R_n = yR$, $G_n = yG$ a $B_n = yB$

Algoritmus 4.8: Ekvalizace jasu barevného obrázku

nejprve ekvalizujeme jas obrazu tak, že aplikujeme algoritmus 4.7 pouze na složku Y. Hodnotu nového jasu označme Y_n. Barevné hodnoty se potom vypočítávají relativně k ekvalizovanému jasu. Z hodnot Y a Y_n se vypočítá relativní hodnota změny jasu pro každou barevnou intenzitu

$$y = \frac{Y}{Y_n},$$

nové hodnoty RGB označené R_n, G_n a B_n se pak vypočtou jako

$$[R_n, G_n, B_n] = [y \cdot R, y \cdot G, y \cdot B]$$

Algoritmus 4.8 naznačuje uvedený postup.

Typickou úlohou, která je řešitelná ekvalizací histogramu, je změna jasových poměrů ve fotografiích pořízených proti světlu nebo v šeru. V těchto obrázcích totiž převládá většina informace v pravé resp. v levé polovině histogramu. Přesunem některých jasových složek do druhé části se stane obrázek kontrastnějším a vystupují detaily, viz obrázek 4.34.

4.6 Odstraňování šumu a ostření obrazu

4.6.1 Odstraňování šumu

Užitečnou operací používanou při digitálních úpravách obrazu je odstraňování šumu. Šum je v teorii signálu definován jako nová informace, která byla k původní přidána pořizovacím zařízením či během transportu. Šum může být aditivní či multiplikativní, podle způsobu, jak byl do původního signálu přidán.

Dobrým příkladem šumu jsou obrazy vzniklé stochastickým renderingem, například metodou dvousměrového sledování paprsku, či metody využívající náhodného vzorkování s vyšší frekvencí.

Poznamenejme, že šum je svázán s původní informací. K určení šumu musíme mít k dispozici původní funkci, abychom mohli říci, co je k ní přidáno. V digitálním obraze však máme obyčejně k dispozici pouze pixely a význam jim přiřazuje lidské vnímání. Algoritmy odstraňující šum nakládají se šumem stejně, jako s jakoukoli vysokofrekvenční informací, například s ostrými hranami, texturami s velkou frekvencí změn mezi sousedními pixely atd.

Druh šumu se určuje podle frekvenční charakteristiky obrazu po aplikaci Fourierovy transformace. *Bílý šum* má frekvenční spektrum dokonale vyrovnané. Všechny frekvence jsou zastoupeny se stejnou pravděpodobností. Prostou úvahou dojdeme k závěru, že se jedná o matematickou abstrakci. Zpětnou transformací bychom totiž získali funkci, která má nekonečný výkon. Za bílý šum se tedy považuje i funkce, která má vyrovnané frekvenční spektrum a navíc je frekvenčně omezená. Hodnoty bílého šumu nejsou autokorelované, mají vzájemnou korelaci rovnou nule. Více se o bílém šumu dozvíme v kapitole 8.1.5 o stochastických fraktálech. *Gaussův šum* má pravděpodobnost frekvence ve frekvenčním spektru dánu vztahem (22.35). Někdy se Gaussův šum zaměňuje za bílý šum a označuje se termínem bílý Gaussův šum.

Impulsní šum je šum, který má vysokou energii a krátké trvání. V počítačové grafice se setkáme zejména s jeho variantou zvanou sůl a pepř – některé pixely mají náhodně změněnou intenzitu na černou či bílou barvu.

Opakované pořízení obrazu

Způsob odstraňování šumu závisí na jeho charakteristice a na způsobu získání obrazu. Pokud máme k dispozici originální funkci a pořizovací zařízení není zatíženo systematickou chybou, snadno odstraníme šum opakovaným pořízením exemplářů obrazu. Porovnáním pixelů na stejných souřadnicích pak určíme hodnotu výsledku. Hodnoty, které se mezi obrazy příliš liší, jsou s největší pravděpodobností chybou. Nabízí se několik způsobů výpočtu; vypočítat průměr z hodnot, použít hodnotu nejčastěji opakovanou, či použít medián (viz algoritmus 4.9). Tato metoda je variantou vzorkování s vyšší frekvencí uvedeném v odstavci 2.4.1. Případ, kdy máme k dispozici více exemplářů předlohy, není příliš častý. Obyčejně máme k dispozici pouze jediný obraz a odstraňujeme šum na základě charakteristiky okolí každého pixelu.

Odstranění šumu v jediném obraze

Odstranění šumu v jediném obraze se také nazývá filtrace šumu. Vzhledem k tomu, že nemáme k dispozici původní funkci, nevíme, co je šum a co je původní signál. Filtry tedy nějakým způsobem vyhodnocují okolí pixelu a z něj usuzují, zda se jedná o šum či ne. Za šum, jak jsme již zmínili, jsou považovány velké změny hodnoty sousedních pixelů. Filtry, které odstraňují šum z jediného obrazu, pracují buď na principu konvoluce, nebo lokální statistiky okolí. Podívejme se nejprve na první skupinu.

Nejjednodušší filtrovací metoda počítá hodnotu pixelu jako průměr z jeho okolí. Ze spektra zmizí vysoké frekvence. Ostré změny v obraze, například hrany či šum, se rozmažou. Opakovanou aplikací tohoto filtru se obraz rozmazává víc a v limitě tato operace vede k obrazu, který má jednu barvu, jež je průměrem ze všech hodnot. Tato technika se nazývá *obyčejné průměrování*. Konvoluční jádro (viz vztah 2.11) pro okolí 3×3 má tvar

$$h = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Snížením vah směrem od středu získáme filtr, který snižuje Gaussův šum:

$$h = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}.$$

Součet hodnot v konvolučním jádru funkce odstraňující šum musí být roven jedné. Pokud by byl vyšší, výsledný obraz by se zesvětloval, jádro se součtem hodnot nižším než jedna by obraz naopak ztmavovalo.

Obrázek 4.35: Obraz (vlevo) zatížený silným impulsním šumem je filtrován (vpravo) pomocí mediánu s oblastí 2×2 pixelů

Ve Fourierově oblasti filtraci pomocí obyčejného průměrování odpovídá vynásobení frekvenční oblasti funkcí, která odstraní všechny frekvence vyšší nežli určitá hodnota. Vysoké

frekvence jsou buď těmito filtry jednoduše odřezány (*low-pass filter*) nebo potlačeny s nějakou váhou [Gonz87]. Ideální nízkofrekvenční filtr ILPF (*ideal low–pass filter*), je popsán vztahem

$$H(u,v) = \begin{cases} 1 & pro\ \sqrt{(u^2+v^2)} \leq D_0 \\ 0 & pro\ \sqrt{(u^2+v^2)} > D_0 \end{cases} , \tag{4.22}$$

kde $\sqrt{(u^2+v^2)}$ označuje vzdálenost frekvence od počátku (srovnej vztah 2.15). Jak je zřejmé, jedná se o válec s poloměrem D_0 a výšce rovné jedné. Slovo „ideální“ označuje fakt, že filtr nepropouští frekvence vyšší nežli je hodnota D_0.

Druhou skupinou metod odstraňující šum jsou metody pracující na bázi lokální statistiky okolí pixelu (*order statistics*). Patrně nejčastěji používanou metodou je filtrace pomocí *mediánu*, viz obrázek 4.35. Pixel na pozici $[i,j]$ ve výsledném obraze se spočítá z obrazu vstupního jako medián z hodnot v jeho okolí. Filtrace pomocí mediánu není konvolucí.

Pro všechny pixely $[i,j]$ v obrazu A

1. Načti body z intervalu $[i-k, j-k]$ $[i+k, j+k]$ do pole M délky $l = (2k+1)^2$
2. Seřaď pole M
3. Výstupní obraz $B[i,j] = M[(l-1)/2]$

Algoritmus 4.9: Filtrování pomocí mediánu

Filtrace pomocí mediánu je výhodná pro odstranění impulsního (bodového) šum, jak je vidět na obrázku 4.35. Tato metoda však narušuje tenké čáry. Okolí bodu, které se používá, nemusí být nutně čtvercové, někdy je výhodné použít okolí ve tvaru kříže, čímž se v obraze neporuší diagonální čáry, okolí ve tvaru písmene „X“ pro změnu neporuší čáry vodorovné a svislé.

4.6.2 Ostření obrazu

Lidské vnímání je založeno na rozpoznávání hran. Podle obrysu postavy je člověk například schopen rozpoznat konkrétní osobu. Jednou z možností jak zvýraznit nějaký obraz, abychom ho vnímali jako ostřejší, je zvýraznit v něm hrany.

Hranu (*edge*) v diskrétním obraze vnímáme tam, kde dochází k výrazné změně sousedních pixelů. Hrana je vysokofrekvenční informace, a proto je její zvýraznění inverzní operací k odstranění šumu. Hrana je určena *gradientem*, tj. velikostí a směrem. Směr lze popsat vektorovým

operátorem nabla ∇

$$\nabla f(x,y) = \left(\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}\right)$$

a velikost gradientu je tedy určena jako délka vektoru:

$$\mid \nabla f(x,y) \mid = \sqrt{\left(\frac{\partial f(x,y)}{\partial x}\right)^2 + \left(\frac{\partial f(x,y)}{\partial y}\right)^2}.$$

Výše uvedené vzorce platí pro spojité funkce. V diskrétním obraze gradient odhadujeme.

Ostření obrazu je pak založeno na následujícím postupu. Označme $s(i,j)$ funkci, která reprezentuje velikost gradientu obrazu f v bodě $[i,j]$. Výsledný obraz $g(i,j)$ získáme z obrazu $f(i,j)$ ostřením koeficientem c (viz obrázek 4.36)

$$g(i,j) = f(i,j) + c.s(i,j) \tag{4.23}$$

Funkce $s(i,j)$ vrací velikost gradientu a o její patřičný násobek se zvýší intenzita pixelu v odpovídajícím bodě. Pro určení gradientu se používají postupy založené na analýze okolí pixelu s použitím konvolučních, nebo jiných operátorů.

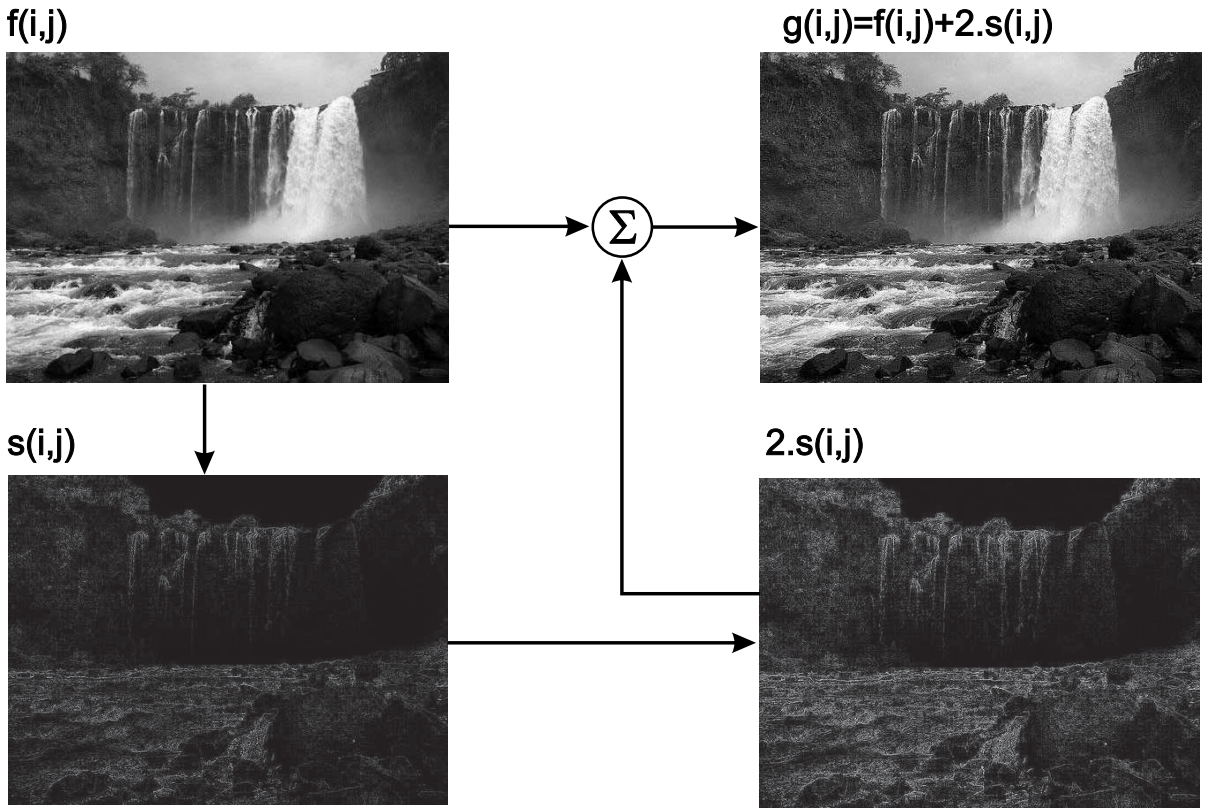
Patrně nejjednodušší metodou určení velikosti gradientu pixelu je použití tzv. *Robertsova operátoru*. Tento operátor není založen na konvoluci a jeho implementace je snadná. Robertsův operátor používá k výpočtu pixel a jeho tří sousedů pixelu a má tvar:

$$\mid \nabla f(i,j) \mid = \mid f(i,j) - f(i+1,j+1) \mid + \mid f(i,j+1) - f(i+1,j) \mid . \tag{4.24}$$

Výpočet určí velikost gradientu jako součet absolutních hodnot změn ve směru hlavní a vedlejší diagonály obrazu tak, jak je vidět na obrázku 4.37. Robertsův operátor detekuje především hrany se sklonem 45^o.

Robertsův operátor je možno rozdělit na dvě složky, z nichž každá detekuje hrany v jednom ze dvou na sebe kolmých směrech. Na podobném principu je založen směrově orientovaný *Sobelův operátor*, který aproximuje první derivaci. Sobelův operátor je složen vždy z dvojice komplementárních konvolučních masek označených jako h a $\bar{h}$. Komplementární maska se získá z původní masky rotací o devadesát stupňů kolem středu. Protože se v dalším výpočtu vypočítává druhá mocnina či absolutní hodnota, je nepodstatné, zda otáčíme doleva či doprava. Příklady komplementárních konvolučních masek jsou:

$$h = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \bar{h} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix},$$



Obrázek 4.36: Ostření obrazu pomocí zvýraznění hran

$$h = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}, \bar{h} = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}.$$

Absolutní velikost gradientu je potom získána dvojnásobnou aplikací konvoluce, nejprve pro h a potom pro $\bar{h}$, a součtem:

$$|G| = \sqrt{h^2 + \bar{h}^2}.$$

Součet se v praxi někdy zjednodušuje na

$$|G| = |h| + |\bar{h}|.$$

Srovnej poslední uvedený vztah s výpočtem pomocí Robertsova operátoru (4.24).

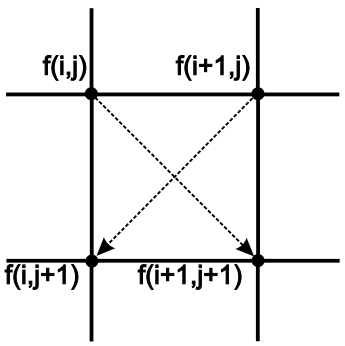
Na konvoluci je založen výpočet gradientu pomocí *Laplaceova operátoru*. Označuje se jako Δ a pro výpočet ze čtyřokolí pixelu ve směru kolmém na souřadnicové osy má jeho konvoluční jádro tvar

$$h = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

Varianta, jež používá k výpočtu velikosti gradientu hodnot z osmiokolí, má tvar

$$h = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Laplaceův operátor je invariantní k otáčení o násobky 45^o.



Obrázek 4.37: Princip Robertsova operátoru

Sobelův a Robertsův operátor aproximují výpočet první derivace, Laplaceův operátor je aproximací derivace druhé [Gome97]. Vezměme libovolnou funkci a spočtěme její první derivaci. Změny gradientu funkce původní se projeví jako lokální extrémy derivace. Pokud provedeme derivaci druhou, extrémy první derivace odpovídají bodům, kde funkce prochází nulovou hodnotou. Z tohoto důvodu se Laplaceův operátor označuje jako operátor průchodu nulou (*zero-crossing operator*). Hodnota, kterou tento operátor vypočítá, je aproximace druhé mocniny gradientu. Gradient se odhaduje jako součet diferencí ve dvou na sebe kolmých směrech

$$\Delta f(x,y) = \frac{\partial^2 f(x,y)}{\partial^2 x} + \frac{\partial^2 f(x,y)}{\partial^2 y}.$$

Obrázek 4.38 ukazuje, jak Laplaceův operátor reaguje na různé typy hran. Obecně operátor reaguje na hranu dvakrát. Poprvé na její nástupné straně, podruhé na její straně sestupné. Výsledkem Laplaceova operátoru může být i negativní hodnota. V praxi se záporné hodnoty buď oříznou, celý rozsah se převede do kladných hodnot, a nebo, a to je nejčastější, se použije absolutní hodnota.

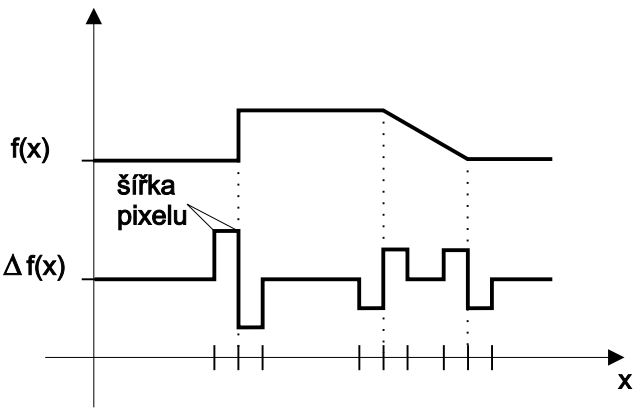
Poznamenejme, že operátory zvýrazňující hrany zvýrazňují bez rozdílu všechny vysoké frekvence a tedy i šum. Méně zvýrazňují šum operátory pracující s větší konvoluční maskou, tedy i s větším okolím pixelu. Další operátory je možno vyhledat v [Gonz87, Gome97, Lewi90].

Zatímco konvoluční maska odstraňování šumu měla vždy za součet svých hodnot jedničku, pro detekci hran musí být součet roven nule. To zaručí, že v oblastech s konstantní hodnotou bude i odezva konvoluce nulová.

Schéma na obrázku 4.36 je dvoukrokové, neboť používá pro ostření meziobraz s, který obsahuje hrany z původního obrazu. Pokud pro detekci hran používáme konvoluci, lze detekci hran a jejich připočtení k obrazu sloučit do jediného kroku. Stačí všechny hodnoty konvolučního jádra vynásobit koeficientem ostření.

Ostření obrazu ve frekvenční oblasti spočívá v potlačení nízkých frekvencí a zdůraznění vysokých. Pro zvýraznění vysokých frekvencí použijeme varianty filtru uvedeného pro odstranění šumu, tj. *high–pass filter*. Ideální vysokofrekvenční filtr IHPF (*ideal high–pass filter*) propouští pouze vysoké frekvence a je popsán následujícím vztahem [srovnej se vztahem (4.22)].

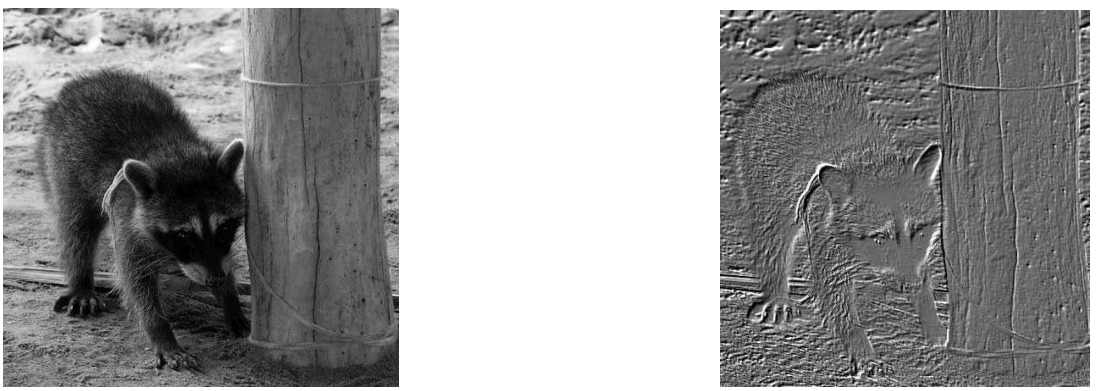
$$H(u,v) = \begin{cases} 0 & pro\ \sqrt{(u^2+v^2)} < D_0 \\ 1 & pro\ \sqrt{(u^2+v^2)} \geq D_0 \end{cases} .$$



Obrázek 4.38: Odezva jednorozměrného Laplaceova operátoru na různé typy hran (pozitivní a negativní).

4.6.3 Vytlačený vzor – emboss

Programy pro editaci obrazu poskytují širokou škálu různých efektů s obrazy. Nebudeme se v tomto textu zabývat těmito postupy podrobně, ale uvedeme jeden příklad za všechny, tzv. *vytlačený vzor* (*emboss*) (viz obrázek 4.39).



Obrázek 4.39: Originál a vytlačený vzor

Smyslem této operace je poskytnout dojem, že obraz je plastický a je jakoby vytlačen do povrchu materiálu.

V prvním kroku při tvorbě vytlačeného vzoru vytvoříme pomocný obraz, který získáme jako negativ předlohy. V dalším kroku pomocný obraz posuneme o nějaký vektor, obyčejně zadaný uživatelem. Předpokládejme, že posouváme obraz o n_x pixelů ve směru osy x a o n_y pixelů ve směru osy y. Dále tento pomocný obraz přičteme k obrazu původnímu a nakonec intenzity všech pixelů vydělíme dvěma. Poslední krok je vlastně vypočtením průměru z předlohy a pomocného obrazu. Slovně uvedený postup popisuje rovnice

$$B[i,j] = \frac{1}{2}\left(A[i,j] + (1 - A[i+n_x, j+n_y])\right), \tag{4.25}$$

kde $A[i,j]$ je pixel předlohy a $B[i,j]$ je pixel výstupního obrazu. Předpokládáme, že maximální intenzita v obraze je rovna jedné.

4.6.4 Malování pomocí počítače

V posledních letech je v počítačové grafice zřetelná tendence k simulaci různých uměleckých postupů. Poznamenejme, že počítač sám o sobě nemůže vytvořit umělecké dílo, stejně jako nemůže sama od sebe malovat tužka. Počítač však může této tvorbě napomoci algoritmizací postupů, které jsou známé. Z technik, které se rozšiřují jako různé filtry programů pro editaci obrazu, uvedeme simulaci impresionismu a pointilismu [Haeb90].

Impresionismus vznikl ve druhé polovině 19. století a je pro něj typické nanášení čistých barev kladením štětce tak, že polotóny vznikají až na sítnici diváka. Neoimpresionismus, pointilismus, klade barvy v bodech s důrazem na kontrast doplňkových barev [Kole93]. Poznamenejme, že v počítačové grafice se stejná technika, tj. vytvoření barevného vjemu kombinací několika blízko sebe umístěných bodů, které mají barvy pouze z množiny základních barev, používá pro tzv. rozptylování barev v části 4.1.1.

Algoritmus simulující tuto činnost se nazývá *brush stroking* a jeho vstupem je:

- plátno – tím je obrazovka počítače,
- štětec – má tloušťku a tvar, např. kruh, elipsu, či čtverec,
- paleta – tou je předem definovaný a konečný počet barev,
- směr, kterým táhneme štětec – vektor,
- předloha – nějaký obrázek, například fotografie.

Algoritmus vezme předlohu a rozdělí ji na čtverce. Z každého čtverce se spočítá průměrná barva a ta se nahradí několika tahy barev z palety. Předdefinovaný směr určuje i základní tahy štětce. Při výpočtu se intenzivně používají náhodná čísla; štětec má proměnnou velikost, modifikuje se směr tahu, jeho délka atd.

Obrázek 4.40: Originál a výsledek pokusu o impresionismus a pointilismus

Pokud neprovedeme tahy štětcem, ale pouze ho do určeného čtverce položíme, budeme simulovat pointilismus. Výsledky obou algoritmů demonstruje obrázek 4.40.

Snahy o napodobení uměleckých směrů nejsou v žádném případě ojedinělé. Poměrně velké popularitě se těší algoritmy, které napodobují techniku litografie, existují algoritmy simulující kreslení uhlem, mokrým štětcem, aj. Dobrým startovním bodem pro další studium jsou filtry v programech Adobe Photoshop, Corel Photopaint či GIMP.

Existují metody, které nepracují s obrazem, ale používají jako předlohu trojrozměrný model. Tyto algoritmy představují nový druh zobrazování prostorových dat: nefotorealistické zobrazování (*non-photorealistic rendering*). Více se o těchto technikách dozvíme v části 17.

Část B

TROJROZMĚRNÉ MODELY

Tato část knihy pojednává o reprezentaci objektů v počítači. Dnes je patrně nejčastější vyjádření objektů pomocí jejich hranice (*boundary representation, B-rep*). Ta může být zadána jako množina spojitých ploch, plošek, například množinou trojúhelníků. Zajímavou, byť méně častou je reprezentace pomocí množiny bodů. Vyjádření pomocí ploch se používá v aplikacích, ve kterých je kladen důraz na přesnost, tedy zejména v CAD/CAM.

Analytické vyjádření hraničně reprezentovaných těles se opírá o poměrně rozsáhlou teorii křivek a ploch. V počítačové grafice se dnes používají převážně plochy *parametrické*, i když i *implicitní plochy* nacházejí stále častěji své aplikace.

Plošková reprezentace, která je dalším tématem této části knihy, je obyčejně získána z analytických ploch a je standardní reprezentací v časově kritických aplikacích, tedy v aplikacích virtuální reality a v počítačových hrách.

Předposlední reprezentace, o které tato část knihy pojednává, je reprezentace objemová. Seznámíme se s datovými strukturami, které se používají pro objemové modely a se základními metodami pro jejich převod do hraniční reprezentace.

Oddíl uzavírá kapitola o procedurálním modelování, které zahrnuje techniky umožňující generovat objekty, se kterými se setkáváme v přírodě.

Kapitola 5

Křivky a plochy

Křivky a plochy se používají v počítačové grafice a souvisejících aplikacích na mnoha různých místech. Setkáváme se s nimi při modelování ve třech i ve dvou dimenzích, při definici fontů, při určování dráhy pohybujících se objektů v počítačové animaci, při definici objektů pro šablonování aj. Různé aplikace mají různé požadavky a proto je tato oblast poměrně široká.

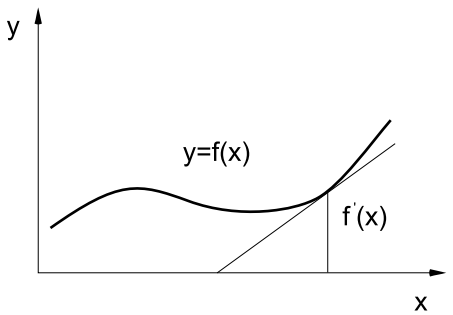
V roce 1959 používal P. de Casteljau u firmy Citroen matematický model křivek a ploch, jenž mu je umožňoval jednoduše zadávat. Podobně v šedesátých letech vedl P. Béziere vývoj programového systému UNISURF pro návrh křivek a ploch u firmy Renault. Metody tvarování křivek a ploch se postupem času zdokonalovaly a v současné době je k dispozici velmi silný nástroj, který je stále aktivně rozvíjen. Podstatou tohoto rozvoje je kvalitní matematický aparát a v neposlední řadě i zřetelný komerční efekt, který se projevil zejména v oblasti průmyslového designu.

Výrazný pokrok do této oblasti a především sjednocení dříve používaných různorodých přístupů přineslo používání racionálních B–spline křivek a ploch s neuniformní parametrizací, NURBS (Non–Uniform Rational B–Spline). Tyto metody umožňují generovat klasické geometrické prvky (úsečky, kružnice, elipsy a v prostoru koule, válce atp.) za pomoci stejných metod, které umožňují vytvořit křivky a plochy se složitými průběhy a tvary.

V této části je uveden popis nejčastěji používaných křivek a ploch pro modelování v počítačové grafice. Přehled uzavírá popis ploch vytvářených pomocí dělicích schémat (*subdivision surfaces*). Křivky vhodné pro modelování pohybujících se objektů v počítačové animaci jsou uvedeny samostatně v části 18.1.2. Podrobný popis všech zajímavých vlastností a modelovacích postupů založený na teoretickém aparátu parametrických křivek a ploch však přesahuje rámec této knihy. Podrobnější informace je možno nalézt například v [Pieg95].

5.1 Vlastnosti křivek

Křivky jsou obyčejně v počítači reprezentovány jako soustava parametrů nějaké rovnice, která je posléze generativně zobrazována. Toto vyjádření může být v podstatě trojího druhu – explicitní, implicitní a parametrické.



Obrázek 5.1: Explicitní zadání křivky

Explicitně vyjádřená křivka může být zadána například jako spojitá funkce ve tvaru

$$y = f(x)$$

a bývá orientována ve směru rostoucího x (obrázek 5.1). Jedná se však o zadání, které lze použít pouze pro křivky, které jsou zároveň funkcemi, tzn. že hodnotě x z definičního oboru odpovídá jediná funkční hodnota y.
Implicitní zadání křivky má tvar

$$F(x, y) = 0 \ ,$$

příkladem je rovnice kružnice $F(x, y) = (x - xs)^2 + (y - ys)^2 - r^2 = 0$. Toto zadání je poměrně obtížně zobrazitelné v porovnání s ostatními, neboť neumožňuje v obecnějších případech postupný výpočet křivky. Svůj význam má například při testování oblastí vymezených implicitně zadanou křivkou nebo při výpočtu průsečíku paprsku s křivkou.

V počítačové grafice se pro vyjádření křivek nejčastěji používá tvar *parametrický* (5.1). Křivku budeme chápat fyzikálně, jako dráhu pohybujícího se bodu, jehož souřadnice jsou funkcemi parametru t (času)

$$x = x(t), \ y = y(t), \ z \ = z(t) \ . \tag{5.1}$$

Parametr t je z intervalu $t \in \langle t_{min}, t_{max} \rangle$ a nejčastěji je volen v rozsahu $t \in \langle 0, 1 \rangle$. Funkcemi (5.1) je určena *bodová rovnice* křivky

$$Q(t) = [x(t), y(t), z(t)], \tag{5.2}$$

nebo *vektorová rovnice*

$$\vec{q}(t) = (x(t), y(t), z(t)). \tag{5.3}$$

Vektor $\vec{q}(t) = Q(t) - [0, 0, 0]$ se nazývá *polohový vektor*, jeho velikost je rovna vzdálenosti bodu $Q(t)$ od počátku. Výhodou parametrického zápisu je závislost souřadnic křivky na jediném parametru t, jehož fyzikální interpretací je čas. Díky tomu je možné vyjádřit průběhy, kdy

křivka prochází vícekrát (v různých časových okamžicích) stejnými body v prostoru, může se křížit, uzavřít apod.

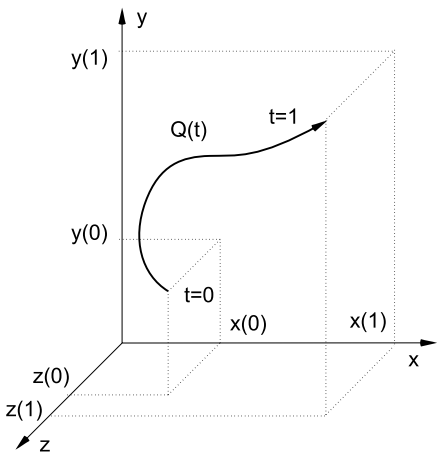
Tečný vektor v bodě $Q(t_0)$ je určen derivacemi parametricky vyjádřené křivky po složkách ve tvaru

$$\vec{q'}(t_0) = (x'(t_0),\ y'(t_0),\ z'(t_0)) = \left(\frac{dx(t_0)}{dt}, \frac{dy(t_0)}{dt}, \frac{dz(t_0)}{dt}\right). \tag{5.4}$$

Rovnice *tečny*, tj. přímky která se křivky v tomto bodě dotýká, se vypočítá z tečného vektoru a bodu na křivce jako

$$P(u) = Q(t_0) + u\ \vec{q'}(t_0) \tag{5.5}$$

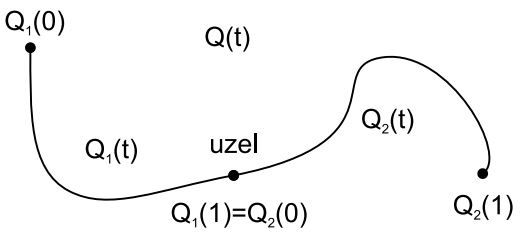
Vektor $\vec{q'}(t)$ se také nazývá *směrový vektor přímky*. Ze vztahu (5.4) je patrné, že parametrická reprezentace křivek umožňuje snadno vyjádřit tečny ke křivce. Toho se s výhodou využívá zejména při navazování křivek a skládání složitých tvarů z jednodušších částí.



Obrázek 5.2: Křivka vyjádřená parametricky

Předpokládejme, že $Q_1(t)$ a $Q_2(t)$ jsou dvě části (neboli *segmenty*) jediné křivky $Q(t)$ spojené v bodě $Q_1(1) = Q_2(0)$ (viz obrázek 5.3). Bod, ve kterém se křivky stýkají, budeme nazývat *uzel* (*knot*).

Segmenty $Q_1(t)$ a $Q_2(t)$ mohou být definovány pomocí různých, nejen polynomiálních reprezentací. Důležitý však je způsob jejich napojení, zejména tzv. *spojitost* (*continuity*) v uzlu.



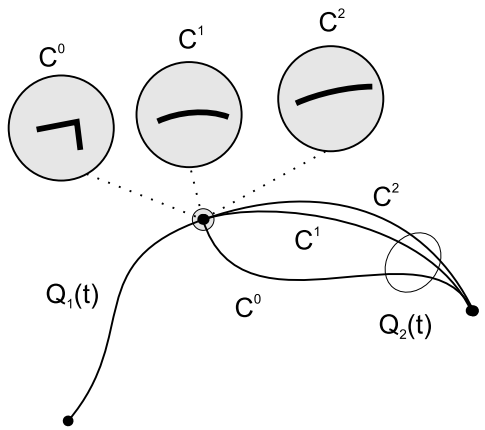
Obrázek 5.3: Křivka $Q(t)$ vzniklá spojením dvou segmentů $Q_1(t)$ a $Q_2(t)$

Říkáme, že křivka $Q(t)$ je třídy C^n, má-li ve všech bodech spojité derivace podle parametru t do řádu n. Označení C^n se nazývá *parametrická spojitost stupně n* (obrázek 5.4).

Dva segmenty jsou *spojitě* navázány, tj. mají spojení třídy C^0, pokud je koncový bod prvního segmentu počátečním bodem segmentu druhého. Dva segmenty mají spojení C^1, pokud

je tečný vektor v koncovém bodě segmentu Q_1 roven tečnému vektoru segmentu Q_2 v jeho počátečním bodě. Analogicky rovnost vektoru první a druhé derivace je požadována pro C^2 (obrázek 5.4), atd. Zkráceně zapisujeme

$$\vec{q_1}^{(i)}(1) = \vec{q_2}^{(i)}(0);\ \forall\ i = 0, 1, \ldots, n. \tag{5.6}$$

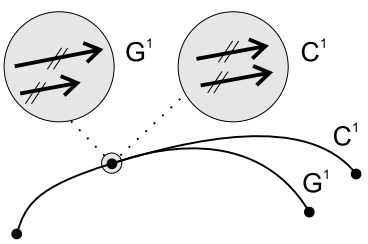


Obrázek 5.4: Spojitost C^0, C^1 a C^2

Čím vyšší spojitost je požadována, tím delší „dobu“ (ve smyslu parametru t) se oba segmenty přimykají ke stejnému směru. Zjevně $C^{n+1} \Rightarrow C^n$. Ze spojitosti C^0 plyne, že bod se pohybuje po spojité dráze, ale v uzlu může měnit skokem směr pohybu, rychlost i zrychlení. Směr pohybu a velikost rychlosti se nemůže měnit skokem při spojitosti C^1 a zrychlení zůstává nezměněné při spojitosti C^2.

Hladkost navázání křivek můžeme také posuzovat podle *geometrické spojitosti* označované G^n, nejčastěji se používají geometrické spojitosti G^0 a G^1. Dva segmenty křivky $Q(t)$ jsou G^0 spojité, pokud je koncový bod Q_1 totožný s počátečním bodem Q_2. Dva segmenty jsou G^1 spojité, pokud jsou G^0 spojité a současně tečné vektory $\vec{q'}_1(1)$ segmentu Q_1 a $\vec{q'}_2(0)$ segmentu Q_2 jsou souhlasně kolineární, tj. platí:

$$\vec{q'}_1(1) = k\ \vec{q'}_2(0);\ k > 0. \tag{5.7}$$



Obrázek 5.5: Geometrická a parametrická spojitost. Lupa ukazuje tečné vektory

Tato spojitost zaručuje totožnost tečen (nikoli tečných vektorů). Pohybující se bod v uzlu nemůže změnit skokem směr, ale může změnit skokem rychlost, křivka je vizuálně hladká.

Geometrická spojitost G^n v daném bodě je definována nezávisle na způsobu, jakým byly obě stýkající se křivky vytvořeny, tj. nezávisle na parametru t. Za předpokladu, že obě křivky jsou v místě spojení diferencovatelné, pak Q_1 a Q_2 splňují podmínku geometrické spojitosti G^n, pokud jsou v bodě $[x_0, y_0, z_0]$ G^{n-1} spojité a platí

$$\left[\frac{\partial^n Q_1}{\partial x^n}, \frac{\partial^n Q_1}{\partial y^n}, \frac{\partial^n Q_1}{\partial z^n}\right]_{[x_0,y_0,z_0]} = h.\left[\frac{\partial^n Q_2}{\partial x^n}, \frac{\partial^n Q_2}{\partial y^n}, \frac{\partial^n Q_2}{\partial z^n}\right]_{[x_0,y_0,z_0]},\ n > 0, h > 0.$$

Ze subjektivního hlediska zaručuje G^1 spojitost „skoro stejnou“ hladkost jako C^1, z hlediska použití bývá daleko snažší zaručit spojitost G^1 nežli C^1. Spojitost C^1 implikuje G^1 s výjimkou jediného případu, kdy vektor rychlosti v místě spojení dvou segmentů je $(0, 0, 0)$. Obráceně toto neplatí, neboť geometrická spojitost nepostihuje rychlosti a zrychlení pohybu.

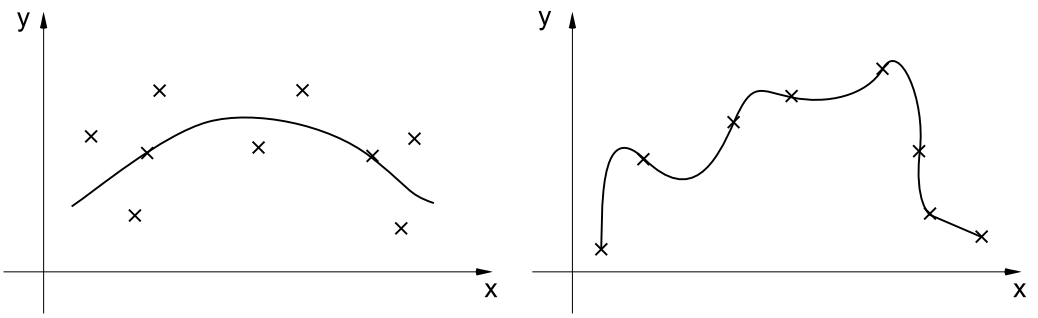
5.2 Modelování křivek

Základním druhem parametrických křivek používaných v počítačové grafice jsou křivky *polynomiální*

$$Q_n(t) = a_0 + a_1 t + \ldots + a_n t^n$$

Polynomiální křivky můžeme snadno vyčíslit a jejich další výhodou je, že jsou jednoduše diferencovatelné. Z polynomiálních křivek lze skládat křivky *po částech polynomiální*, to jsou křivky, jejichž segmenty jsou polynomiálními křivkami. Nejčastěji používané jsou křivky třetího stupně – *kubiky*, které poskytují dostatečně širokou škálu tvarů, jejich výpočet bývá nenáročný, lze s nimi snadno manipulovat a je u nich možné zaručit spojitost C^2, která je často požadovaná při modelování v CAD systémech.

Modelování probíhá obvykle tak, že je definováno několik *řídicích bodů* (*řídicí polygon*) a matematický aparát z jejich polohy určí průběh křivky. Některé metody umožňují zadávání křivek též pomocí tečných vektorů, je možné zaručit spojitost a hladkost navázání aj.



Obrázek 5.6: Aproximační (vlevo) a interpolační křivka a jejich řídicí body

V dalším textu budeme křivku označovat $Q(t)$ a její řídicí body P_i. Pokud budeme hovořit o spojení dvou segmentů křivky, bude první segment označen $Q_1(t)$ a bude zadán řídicími body P_i, zatímco druhý segment budeme označovat $Q_2(t)$ a bude zadán řídicími body Q_i. Tečný vektor ke křivce budeme označovat $\vec{q'}(t)$ a druhou derivaci pak $\vec{q''}(t)$. Tečné vektory v řídicích bodech nebo uzlech, kterými křivka prochází, budeme označovat $\vec{p'}_i$, resp. $\vec{q'}_i$. Existují dva základní způsoby interpretace řídicích bodů a to *interpolace* a *aproximace* (viz obrázek 5.6).

Křivka generovaná při interpolaci probíhá danými body, zatímco při aproximaci je řídicími body tvar křivky určen, ta jimi však procházet nemusí. Parametricky zadanou kubiku $Q(t)$ ve tvaru:

$$\begin{aligned} x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x \\ y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y \\ z(t) &= a_z t^3 + b_z t^2 + c_z t + d_z \end{aligned} \tag{5.8}$$

můžeme zapsat zkráceně v maticovém tvaru:

$$Q(t) = \mathbf{TC} = [t^3\ t^2\ t\ 1] \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix}.$$

Derivaci $\vec{q'}(t)$ získáme derivací vektoru $\mathbf{T}$

$$\vec{q'}(t) = \frac{d}{dt} Q(t) = \frac{d}{dt} \mathbf{TC} = [3t^2\ 2t\ 1\ 0]\ \mathbf{C}.$$

Kubika v prostoru je určena dvanácti parametry, které tvoří prvky matice $\mathbf{C}$. Ovládání tvaru křivky pomocí jednotlivých parametrů však není intuitivní, ze změny parametrů nelze jednoduše odhadnout změnu tvaru křivky. Při interaktivním modelování se dá s výhodou spojit tvarování křivek a ploch s viditelnými prvky, jako jsou řídicí body, směry a tečné vektory, zakřivení apod. Při definici křivek i ploch s určitými vlastnosti je výhodné oddělit charakteristiky, které jsou pro danou křivku individuální, od vlastností, které jsou společné pro všechny křivky modelované shodným způsobem. V případě kubik můžeme matici $\mathbf{C}$ rozepsat do součinu $\mathbf{C} = \mathbf{MG}$, kde matice konstant $\mathbf{M}$ je typu 4×4 a nazývá se *bázová matice* a čtyřprvkový vektor $\mathbf{G}$ se nazývá *vektor geometrických podmínek*. Součin $\mathbf{TM}$ definuje *polynomiální bázi* (skupinu polynomů), která je společná pro všechny křivky určitého typu. Vektor geometrických podmínek obsahuje konkrétní parametry, které ovlivňují tvar křivky, např. řídicí body nebo řídicí body a tečné vektory. Kubika je definována vztahem

$$Q(t) = \mathbf{TMG} = [t^3\ t^2\ t\ 1] \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix}. \tag{5.9}$$

Rovnici (5.9) rozepíšeme jako součet polynomů násobených geometrickými podmínkami

$$\begin{aligned} Q(t) &= (m_{11}t^3 + m_{21}t^2 + m_{31}t + m_{41}).G_1 + \\ &\quad (m_{12}t^3 + m_{22}t^2 + m_{32}t + m_{42}).G_2 + \\ &\quad (m_{13}t^3 + m_{23}t^2 + m_{33}t + m_{43}).G_3 + \\ &\quad (m_{14}t^3 + m_{24}t^2 + m_{34}t + m_{44}).G_4 \quad . \end{aligned} \tag{5.10}$$

Polynomy, kterými jsou geometrické podmínky násobeny, se uplatní jako proměnlivé váhy řízené parametrem t. Podle hodnoty parametru t bázové polynomy určují, která z podmínek se více uplatní na začátku křivky a která má větší vliv na vnitřní průběh nebo na koncovou část. Názorným příkladem je následující křivka sestavená ze dvou geometrických podmínek násobených bázovými polynomy prvého stupně, ve které čtenář jistě rozpozná rovnici úsečky

$$Q(t) = (1 - t).P_1 + t.P_2.$$

Geometrie je plně určena body P_1 a P_2, bázové polynomy určují vliv těchto bodů na průběh úsečky.
Mezi často požadované vlastnosti křivek patří:

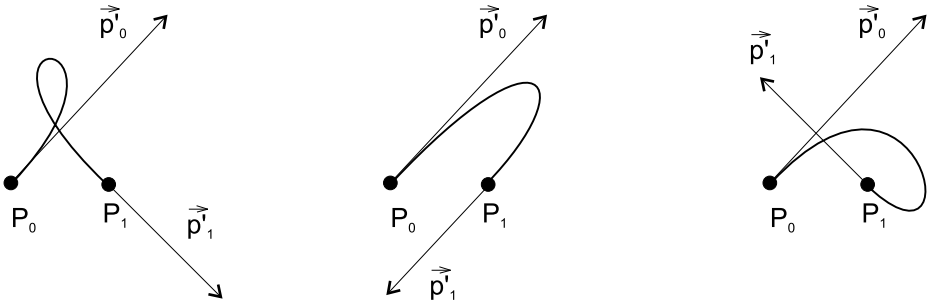
1. Invariance k lineárním transformacím a projekcím, která zaručuje, že například otočení řídicího polygonu a následné generování křivky dá stejný výsledek, jako otočení každého bodu z vygenerované křivky.
2. Vlastnost konvexní obálky (*convex hull property*):
   (a) silná podmínka – celá křivka leží v konvexní obálce všech svých řídicích bodů,
   (b) slabá podmínka – část křivky leží v konvexní obálce některých řídicích bodů (segment v obálce svého generujícího polygonu).
3. Lokalita změn – změnou polohy a/nebo váhy (viz část 5.4.5) řídicího bodu se mění jen část křivky, nikoli křivka celá.
4. Křivka má procházet krajními body svého řídicího polygonu.

5.3 Interpolační křivky

V této části popíšeme nejznámější interpolační křivky používané v počítačové grafice, kterými jsou Hermitovské křivky. Další interpolační křivky uvedeme v části pojednávající o počítačové animaci (část 18.1.2).

5.3.1 Hermitovské kubiky

Mezi často používané křivky patří *Hermitovské kubiky*, někdy také označované jako Fergusonovy kubiky [Fole90]. Tyto křivky jsou určeny dvěma řídicími body P_0, P_1 a dvěma tečnými vektory $\vec{p'}_0$ a $\vec{p'}_1$ v nich. Body P_0 a P_1 určují polohu křivky, křivka v nich začíná a končí. Směr a velikost tečných vektorů pak určuje míru jejího vyklenutí. Čím je velikost vektoru větší, tím více se k němu křivka přimyká. Jsou-li oba vektory nulové, pak se křivka stane úsečkou P_0P_1. Obrázek 5.7 demonstruje změnu tvaru křivky, je-li vektor $\vec{p'}_0$ konstantní a vektor $\vec{p'}_1$ se mění.



Obrázek 5.7: Změna tvaru Hermitovské kubiky

Předpis pro výpočet Hermitovské kubiky ve stylu (5.9) má tvar

$$Q(t) = [t^3\ t^2\ t\ 1] \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ \vec{p'}_0 \\ \vec{p'}_1 \end{bmatrix}, \tag{5.11}$$

Rozepíšeme-li vztah (5.11) do jediné rovnice, dostaneme

$$Q(t) = P_0F_1(t) + P_1F_2(t) + \vec{p'}_0F_3(t) + \vec{p'}_1F_4(t), \tag{5.12}$$

kde F_1, F_2, F_3, F_4 jsou tzv. *kubické Hermitovské polynomy* ve tvaru (5.13) s průběhy znázorněnými na obrázku 5.8.

$$\begin{aligned} F_1(t) &= 2t^3 - 3t^2 + 1, \\ F_2(t) &= -2t^3 + 3t^2, \\ F_3(t) &= t^3 - 2t^2 + t, \\ F_4(t) &= t^3 - t^2. \end{aligned} \tag{5.13}$$

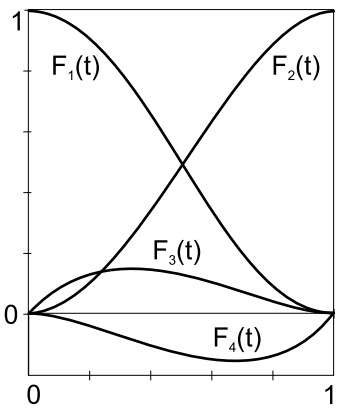
Dosazením $t = 0$, resp. $t = 1$ ověříme, že křivka začíná a končí v bodě P_0, resp. P_1. Největší přednost Hermitovských kubik se projeví při jejich navazování. Vzhledem k tomu, že součástí

definice těchto křivek jsou tečné vektory v jejich koncových bodech, můžeme jejich hladké navazování realizovat velice snadno.

Spojitost při navazování dvou Hermitovských kubik docílíme totožností posledního bodu segmentu $Q_1(1)$ a prvního bodu segmentu $Q_2(0)$ (obrázek 5.3). Spojitost C^1 je zaručena identitou tečných vektorů v uzlu. Spojitost G^1 docílíme jejich lineární závislostí, vyjádřeno pomocí geometrických vektorů $\mathbf{G}$ dvou po sobě následujících segmentů

$$\mathbf{G}_1 = \begin{bmatrix} P_1 \\ P_2 \\ \vec{p_1}' \\ \vec{p'}_2 \end{bmatrix}, \ \mathbf{G}_2 = \begin{bmatrix} P_2 \\ P_3 \\ k\ \vec{p'}_2 \\ \vec{p'}_3 \end{bmatrix}, \ k > 0 \qquad (5.14)$$

Nevýhodou těchto křivek je poměrně nesnadná editace tečného vektoru ve třech rozměrech. Detailně se Hermitovským kubikám věnuje [Fole90, Mort97, Salo99].



Obrázek 5.8: Hermitovské polynomy 3. stupně

5.4 Aproximační křivky

5.4.1 Bézierovy křivky

Patrně nejpopulárnější aproximační křivky používané pro modelování ve dvou rozměrech, ale i pro definici trojrozměrných objektů, jsou Bézierovy křivky. Tyto křivky se často používají i při definici písma (*font*).

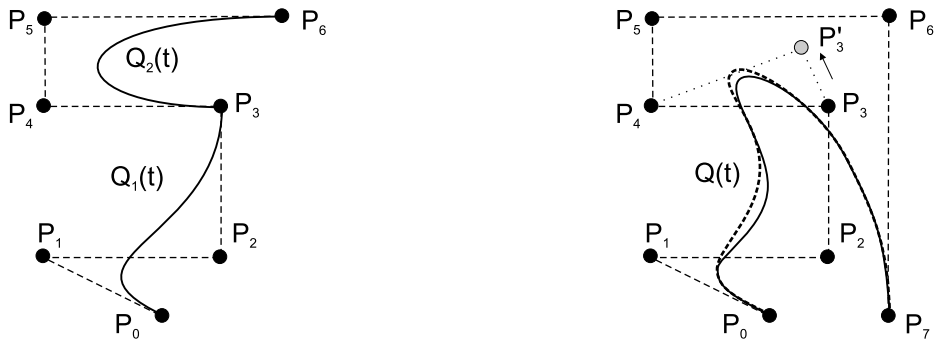
Nejprve uvedeme obecné Bézierovy křivky n-tého stupně a pak jejich nejčastěji používanou variantu, Bézierovy kubiky. Bézierovy křivky n-tého stupně jsou určeny $n+1$ body P_i řídicího polygonu a vztahem

$$Q(t) = \sum_{i=0}^{n} P_i B_i^n(t), \qquad (5.15)$$

kde B_i^n jsou *Bernsteinovy polynomy* n-tého stupně

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i} \ ; t \in \langle 0,1 \rangle \ ; \ i \ = 0, 1, \ldots n. \qquad (5.16)$$

V tomto vztahu je $\binom{n}{0} = 1$ a $0^0 = 1$.



Obrázek 5.9: Bézierovy křivky 3. stupně (vlevo) a 7. stupně (vpravo)

Položíme-li ve vztahu (5.15) $t = 0$, resp. $t = 1$, snadno odvodíme, že křivka prochází prvním, resp. posledním bodem řídicího polygonu (viz též obrázek 5.9). Dosazením $t = 0$ a $t = 1$ do derivace vztahu (5.15) dostaneme výrazy pro tečné vektory v krajních bodech:

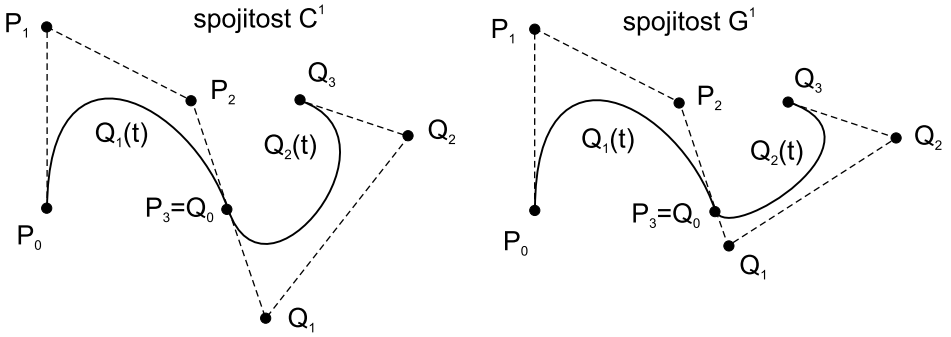
$$\begin{aligned} \vec{q'}(0) &= n(P_1 - P_0), \\ \vec{q'}(1) &= n(P_n - P_{n-1}). \end{aligned} \tag{5.17}$$

Vlastností Bézierovy křivky je, že při změně polohy jediného řídicího bodu P_i dojde ke změně tvaru celé křivky, jak je patrné na obrázku 5.9 vpravo při posunutí bodu P_3. Tato vlastnost je jedním z důvodů, proč se tyto křivky dělí na segmenty nižšího stupně (nejčastěji kubiky), které se postupně navazují.

Při navazování segmentů složených z Bézierových křivek zaručíme spojitost C^0 identitou posledního bodu řídicího polygonu segmentu Q_1 a prvního bodu segmentu Q_2. Spojitost C^1, identita a nenulovost tečných vektorů je garantována [jak je patrné ze vztahu (5.17)], je-li bod $P_n = Q_0$ středem úsečky určené body P_{n-1} a Q_1, jak je vidět z obrázku 5.10 vlevo, kde $P_3 = Q_0$. Spojitost G^1 zajistíme, pokud budou ležet tyto tři body na přímce, přitom bude $P_2 \neq P_3 \neq Q_1$ a bude zachováno jejich pořadí stejně jako na obrázku 5.10 vpravo.
Bernsteinovy polynomy B_i^n ze vztahu (5.16) mají následující vlastnosti:

1. $\forall\ i, n \in N \cup \{0\}$ a $t \in \langle 0, 1\rangle$ je $B_i^n(t) \geq 0$.
2. $\sum_{i=0}^{n} B_i^n(t) = 1$ pro $t \in \langle 0, 1\rangle$.
3. $B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t)$.

První vztah zaručuje nezápornost Bernsteinových polynomů. První a druhý pak zaručují, že výsledná křivka bude vždy ležet v konvexní obálce bodů řídicího polygonu. Třetí vztah je



Obrázek 5.10: Spojení C^1 (vlevo) a G^1 (vpravo) dvou Bézierových kubik. První segment je označen $Q_1(t)$ a je určen body P_0 až P_3, druhý je označen $Q_2(t)$ a je určen body Q_0 až Q_3

rekurentní definicí Bernsteinova polynomu stupně n pomocí lineární kombinace dvou po sobě následujících Bernsteinových polynomů stupně $n-1$.

Podívejme se nyní, jakým způsobem můžeme tyto křivky zobrazovat v diskrétním rastru obrazovky. Předpokládejme, že budeme pracovat s dvourozměrnými Bézierovými křivkami.

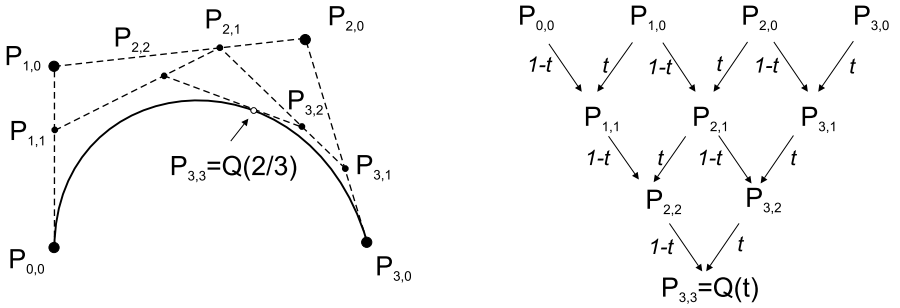
Naivní a neadaptivní způsob generování Bézierovy křivky spočívá v dosazování parametru t do vztahu (5.15) a ve spojování vypočtených bodů úsečkami. Změna parametru Δt je obyčejně konstantní. Tento způsob není příliš efektivní, neboť nerespektuje velikost zakřivení. Naopak jeho výhodou je, že generuje předem známý počet úseček.

Jinou metodou, jak vypočítat Bézierovu křivku stupně n, je použít rekurzivní *algoritmus de Casteljau*. Bod křivky o souřadnicích $Q(t)$ se vypočítá podle rekurentního vztahu

$$P_{j,i}(t) = (1-t)P_{j-1,i-1} + tP_{j,i-1}, \tag{5.18}$$

kde $i = 1, 2, \ldots, n$ a $j = i, i+1, \ldots, n$. Vstupem tohoto algoritmu jsou body řídicího polygonu. Dosadí se za $P_{i,0} = P_i$ a rekurentní vztah (5.18) postupně vypočítává body $P_{i,j}$ tak, jak ukazuje obrázek 5.11 vpravo. V posledním kroku výpočtu je koeficient $P_{n,n}(t)$ roven hodnotě křivky v bodě $Q(t)$. Na obrázku 5.11 vlevo je ukázán výpočet bodu $Q(3/4)$.

Beziérova křivka může být pomocí algoritmu de Casteljau rozdělena na dvě části v libovolném místě, tzn. pro zvolenou hodnotu parametru $t \in \langle 0, 1\rangle$. Nejjednodušší volbou je dělení ve středu křivky, tj. $t = 1/2$. Nechť původní křivka n-tého stupně je určena řídicími body $P_0, \ldots, P_n$. Při dělení vytvoříme dvě skupiny řídicích bodů $L_0, \ldots, L_n$ a $R_0, \ldots, R_n$, které určují příslušné části rozpůlené křivky. S využitím výpočetního postupu znázorněného na obrázku 5.11 získáme nové



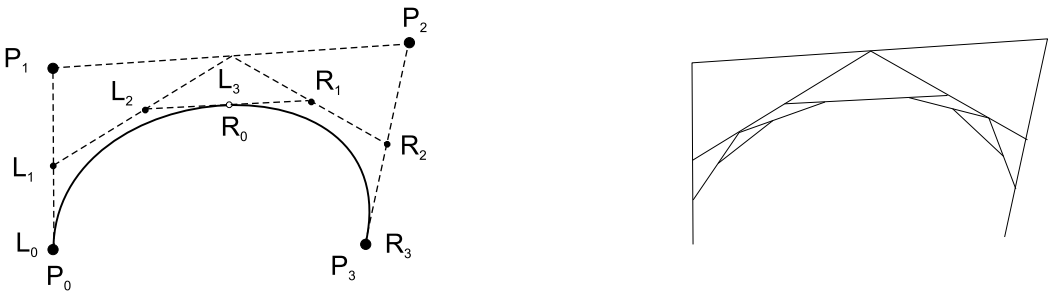
Obrázek 5.11: Algoritmus de Casteljau: Výpočet bodu Q(2/3) (vlevo) a schéma výpočtu

řídicí body pomocí vztahů

$$L_i = \sum_{j=0}^{i} \binom{i}{j} \frac{P_j}{2^i} \quad i = 0, 1, \ldots, n \tag{5.19}$$

$$R_i = \sum_{j=i}^{n} \binom{n-i}{n-j} \frac{P_j}{2^{n-i}} \quad i = 0, 1, \ldots, n\ , \tag{5.20}$$

což není nic jiného, než vytvoření nových řídicích bodů půlením úseček. Pokud tento postup opakujeme, konverguje polygon určený řídicími body k Bézierově křivce, jak je ukázáno na obrázku 5.12.



Obrázek 5.12: Dělení Bézierovy kubiky na dvě části (vlevo), konvergence řídicích polygonů k Bézierově křivce

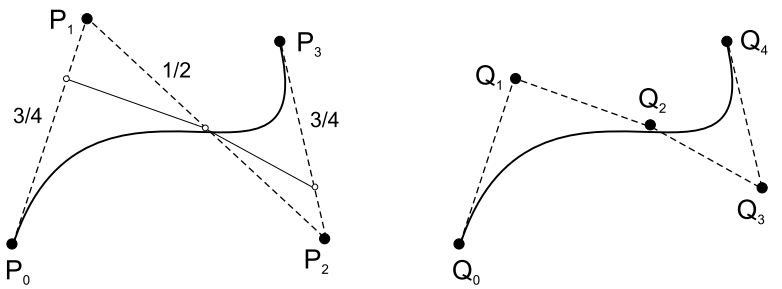
Řídicí polygon je aproximací Bézierovy křivky. Každé rozdělení generuje dva nové řídicí polygony, které jsou přesnějšími aproximacemi této křivky. Tuto vlastnost s výhodu využijeme při generování křivky v diskrétním rastru. Rekurzivní proces zastavíme například v okamžiku,

kdy je vzdálenost dvou sousedních bodů v řídicím polygonu menší, nežli je délka úhlopříčky pixelu. V tom okamžiku je již další dělení zbytečné. Z polohy úsečky v pixelu můžeme navíc určit její koeficient α (viz část 4.4.1) pro antialiasing. Rekurzivní dělení můžeme také zastavit v okamžiku, když je křivka „dostatečně rovná“, jako kritérium můžeme použít velikost plochy řídicího polygonu. Výhodou této metody je, že produkuje (díky kritériu zastavení dělení) menší objem dat. Rovné úseky jsou reprezentovány jako jediná úsečka, zatímco v křivých částech je použito jemnější rozdělení.

Pokud chceme editovat průběhy, které Bézierova křivka daného stupně neumožňuje, můžeme, kromě rozdělení křivky na části, také zvýšit její stupeň. Pro zvýšení stupně polynomu Bézierovy křivky, musíme vytvořit nový řídicí polygon, který má o jeden bod více, ale generuje shodnou křivku. Mějme $n+1$ bodů P_0, P_1, …, P_n řídicího polygonu. Nový řídicí polygon bude mít $n+2$ bodů Q_0, Q_1, …, Q_n, Q_{n+1}. Aby byla zaručena identita křivek, musí pro body nového polygonu platit

$$Q_i = \frac{i}{n+1}P_{i-1} + (1 - \frac{i}{n+1})P_i, \quad i = 0, 1, \ldots, n+1. \tag{5.21}$$

Zvýšení stupně je ukázáno na obrázku 5.13. Vlevo je křivka 3. stupně, vpravo je identická křivka s novým řídicím polygonem, který umožňuje při další editaci modelovat průběhy, definované polynomem vyššího stupně.



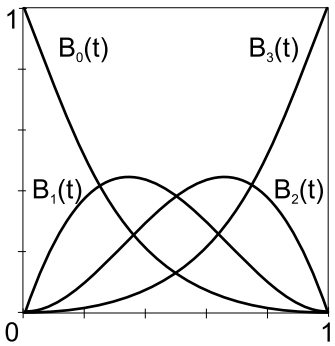
Obrázek 5.13: Zvýšení stupně Bézierovy křivky

Bézierovy křivky leží v konvexní obálce svého řídicího polygonu. Změna polohy bodu má vliv na změnu tvaru celé křivky. Proto je vhodnější skládat složitější průběhy po částech z Bézierových segmentů a využít snadné kontroly spojitosti v uzlových bodech mezi segmenty. Bézierovy křivky jsou invariantní k otáčení, posunu a změně měřítka. Protože Bézierovy křivky uvedené v této části jsou neracionální, tak nejsou invariantní k perspektivnímu promítání. S využitím homogenních souřadnic a s přechodem na racionální vyjádření Bézierových křivek lze i tento nedostatek obejít [Roge01].

5.4.2 Bézierovy kubiky

Nejčastěji používanými křivkami jsou Bézierovy kubiky. Bézierova kubika je zadána čtyřmi body P_0, P_1, P_2 a P_3. Vychází z prvního řídicího bodu, končí v posledním a je určena vztahem

$$Q(t) = \sum_{i=0}^{3} P_i B_i(t), \tag{5.22}$$



Obrázek 5.14: Bernsteinovy polynomy 3. stupně

Bernsteinovy polynomy (5.16) třetího stupně mají tvar uvedený v (5.24), průběhy polynomů jsou ukázány na obrázku 5.14.

$$\begin{aligned} B_0(t) &= (1-t)^3 \\ B_1(t) &= 3t(1-t)^2 \\ B_2(t) &= 3t^2(1-t) \\ B_3(t) &= t^3 \end{aligned} \tag{5.23}$$

Maticový zápis Bézierovy kubiky je

$$Q(t) = [t^3\ t^2\ t\ 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}. \tag{5.24}$$

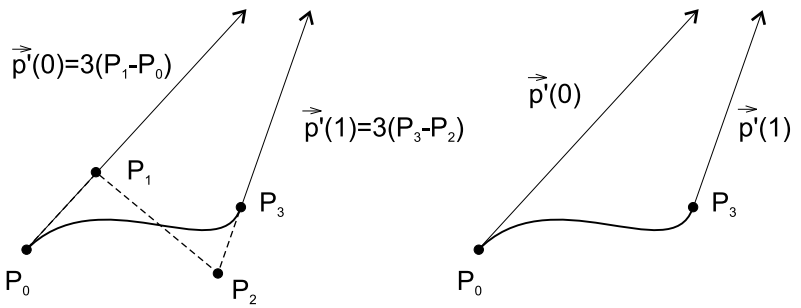
Tečné vektory v prvním a posledním bodě mají tvar:

$$\vec{p'}(0) = 3(P_1 - P_0)\ , \qquad \vec{p'}(1) = 3(P_3 - P_2).$$

Ze vztahu pro tečné vektory je zřejmý postup, jakým převedeme kubiku v Bézierově reprezentaci (čtyři řídicí body) na kubiku v Hermitovské reprezentaci (dva body a dva vektory). Konstrukce je ukázána na obrázku 5.15.

5.4.3 Coonsovy kubiky

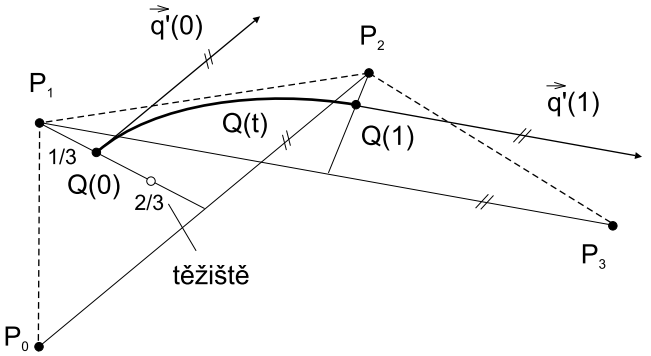
Coonsova kubika zaujímá vedle Bézierových křivek významné postavení v historii parametrických křivek a ploch. Je běžně používána při tvorbě po částech skládaných polynomiálních křivek, úvahy o uniformní a neuniformní parametrizaci výrazně přispěly k zobecněnému pohledu na vlastnosti a tvorbu polynomiálních bází a v neposlední řadě se stala základem obecného aparátu NURBS (viz část 5.4.6). Historické souvislosti nalezne čtenář v knize [Roge01].



Obrázek 5.15: Převod mezi Bézierovou (vlevo) a Hermitovskou reprezentací kubiky (vpravo)

Coonsova kubika, neboli uniformní neracionální B-spline, zkráceně B-spline, je určena čtyřmi řídicími body P_0, P_1, P_2 a P_3 a vztahem

$$Q(t) = \frac{1}{6}\,[t^3\ t^2\ t\ 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} \tag{5.25}$$



Obrázek 5.16: Coonsova kubika

Bázové polynomy $BS_0(t)$, $BS_1(t)$, $BS_2(t)$ a $BS_3(t)$ Coonsových kubik s průběhy na obrázku 5.17 jsou voleny tak, že segment křivky na rozdíl od Bézierových křivek obecně *neprochází* krajními body svého řídicího polygonu. Dosazením $t = 0$ a $t = 1$ do (5.25) zjistíme (obrázek 5.16), že křivka začíná a končí v bodech:

$$Q(0) = \frac{P_0 + 4P_1 + P_2}{6}\ ,\quad Q(1) = \frac{P_1 + 4P_2 + P_3}{6}. \tag{5.26}$$

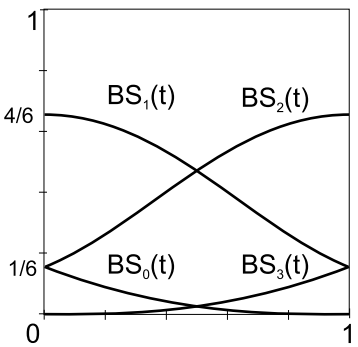
Bod $Q(0)$ (viz obrázek 5.16) leží v první třetině té těžnice trojúhelníku tvořeného body P_0, P_1 a P_2, která začíná v bodě P_1. Tento bod se nazývá antitěžiště. Podobně bod $Q(1)$ leží v první třetině těžnice trojúhelníku tvořeného body P_1, P_2 a P_3 začínající v bodě P_2.

Derivací vztahu (5.25) a dosazením $t = 0$, resp. $t = 1$ dostaneme vztahy pro tečné vektory v krajních bodech:

$$\vec{q'}(0) = \frac{P_2 - P_0}{2} \ , \quad \vec{q'}(1) = \frac{P_3 - P_1}{2}. \tag{5.27}$$

Vektory druhých derivací mají podobu:

$$\vec{q''}(0) = P_0 - 2P_1 + P_2 \ , \quad \vec{q''}(1) = P_1 - 2P_2 + P_3 \ . \tag{5.28}$$



Obrázek 5.17: Bázové polynomy B-spline kubik

Velký význam má u Coonsových kubik *násobnost* bodů řídicího polygonu. Pokud položíme $P_0 = P_1$, vytvoříme *dvojnásobný bod* a trojúhelník určený body P_0, P_1 a P_2 zdegeneruje na úsečku P_0 a P_2. Ze vztahu (5.25) pak snadno odvodíme, že křivka začíná v jedné šestině této úsečky. Položíme-li $P_0 = P_1 = P_2$, bude výsledná křivka úsečkou, počínající v P_0 a končící v jedné šestině úsečky P_0P_3, Coonsova kubika tedy prochází trojnásobným bodem.

5.4.4 Spline křivky

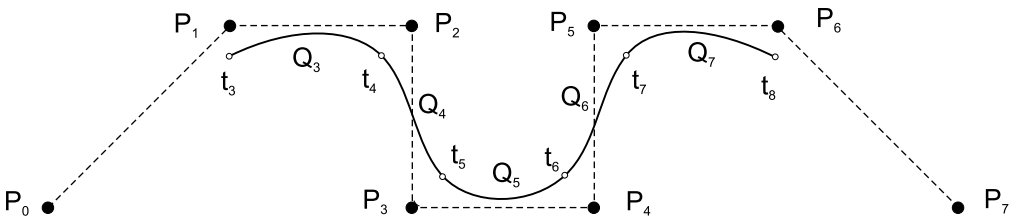
Spline křivka stupně n je po částech polynomiální křivka, která je třídy C^n. Termín spline pochází od pružného pravítka (křivítka), které tyto křivky modelují.

Přirozený spline je spline, který interpoluje své řídicí body. Přirozený kubický spline je tedy interpolační křivka skládající se z polynomiálních křivek stupně tři, která je ve svých uzlech C^2 spojitá. Výpočet přirozeného spline vede k úloze výpočtu inverzní matice k matici $(m + 1) \times (m + 1)$, kde m je počet bodů řídicího polygonu. Nevýhodou těchto křivek je, že změnou polohy jediného definujícího bodu se změní tvar celé křivky. Výpočet přirozených spline křivek je možno nalézt v [Fole90].

V počítačové grafice jsou nejčastěji používané *B-spline kubiky*, které nejsou přirozenými spline křivkami, protože se jedná o křivky aproximační. Nejprve uvedeme nejjednodušší případ, uniformní neracionální B-spline. Jeho zobecněním je neuniformní racionální B-spline, neboli *NURBS*.

5.4.5 Uniformní kubický B-spline

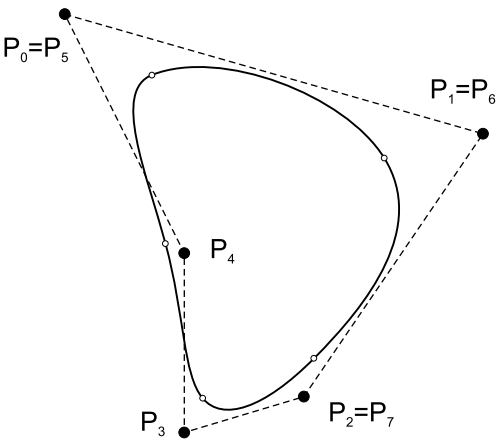
Uniformní kubický B-spline se také nazývá *Coonsův kubický B-spline*. Vznikne navázáním Coonsových kubik (5.4.3) takto: segment Q_i je určen body P_{i-3}, P_{i-2}, P_{i-1} a P_i. Následující segment Q_{i+1} je definován body P_{i-2}, P_{i-1}, P_i a P_{i+1}, tedy třemi posledními body segmentu Q_i a jedním bodem segmentu následujícího. Zatímco Coonsova křivka je určena právě čtyřmi body, Coonsův B-spline je určen $n \geq 4$ body a skládá se z $n-3$ segmentů. Je zřejmé, že Coonsova kubika sama o sobě je B-spline.



Obrázek 5.18: Coonsův kubický B-spline

Porovnáme-li vztahy pro tečné vektory a vektory druhých derivací dvou po sobě následujících segmentů Q_i a Q_{i+1} (viz (5.26), (5.27) a (5.28)), zjistíme, že segment Q_{i+1} vychází z posledního bodu segmentu Q_i a že jsou identické první a druhé derivace v tomto bodě. Křivka je tedy v uzlech C_2 spojitá.

Coonsův kubický B-spline generovaný $n+1$ body řídicího polygonu P_0, P_1, …, P_n, je složen z $n-2$ segmentů Q_3, Q_4, …, Q_n (obrázek 5.18). Parametr t probíhá interval $\langle t_3, t_{n+1}\rangle$ a hodnoty parametru t v uzlech (označujeme je t_i) definují *uzlový vektor*(*knot vector*). Tyto hodnoty mají konstantní vzdálenost $t_{i+1} - t_i = k$, uzlový vektor je tedy *uniformní* (odtud i název těchto křivek). Interval $\langle t_i, t_{i+1}\rangle$, který určuje body B-spline v rámci jednoho segmentu, nazýváme *lokální parametrizací*, interval $\langle t_3, t_{n+1}\rangle$, který pokrývá celý průběh po částech složeného B-spline, nazýváme *globální parametrizací*.

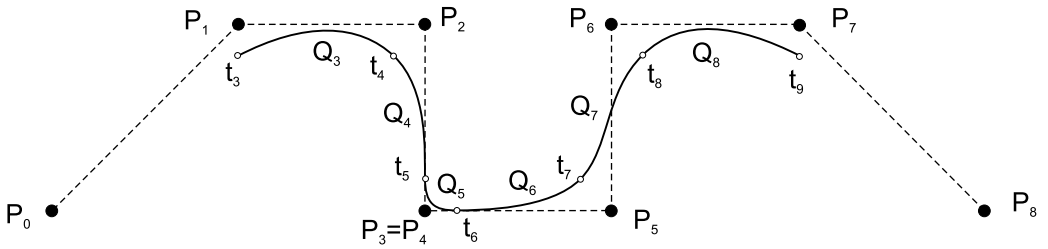


Obrázek 5.19: Uzavřený spline

Uzavřený Coonsův B-spline (též periodický spline) (obrázek 5.19) je vytvořen opakováním prvních tří bodů řídicího polygonu na jeho konci. Pro kubický B-spline je tedy řídicím polygonem uzavřené křivky posloupnost bodů: P_0, P_1, …, P_m, P_0, P_1, P_2.

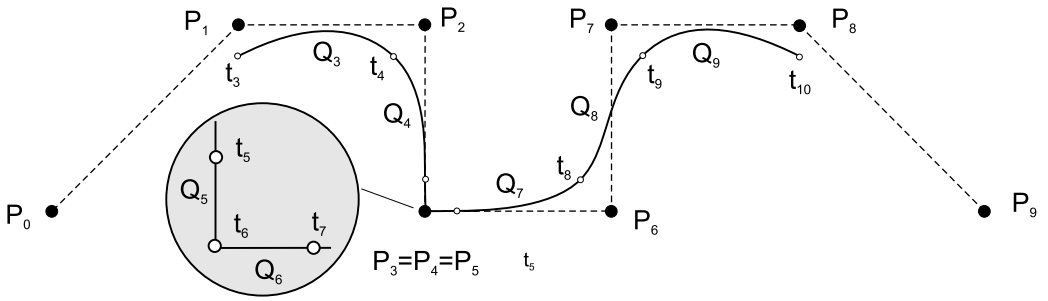
Mezi důležité vlastnosti B-spline křivek patří invariance vůči otáčení, posunutí a změně měřítka, B-spline křivka leží celá ve své konvexní obálce a její segmenty leží v konvexních obálkách svých řídicích polygonů.

Při tvorbě B-spline můžeme některé řídicí body zopakovat a vytvořit násobné body. Definujeme-li křivku posloupností P_0, P_0, P_0, P_1, …, P_{n-1}, P_n, P_n, P_n, zajistíme, že výsledný kubický spline bude procházet krajními body svého řídicího polygonu. První segment bude úsečkou spojující bod P_0 s bodem ležícím v jedné šestině úsečky P_0P_1 a analogicky poslední segment na konci řídicího polygonu. Toto je důsledkem vlastností Coonsových kubik, které jsme popsali v části 5.4.3.



Obrázek 5.20: Dvojnásobný bod

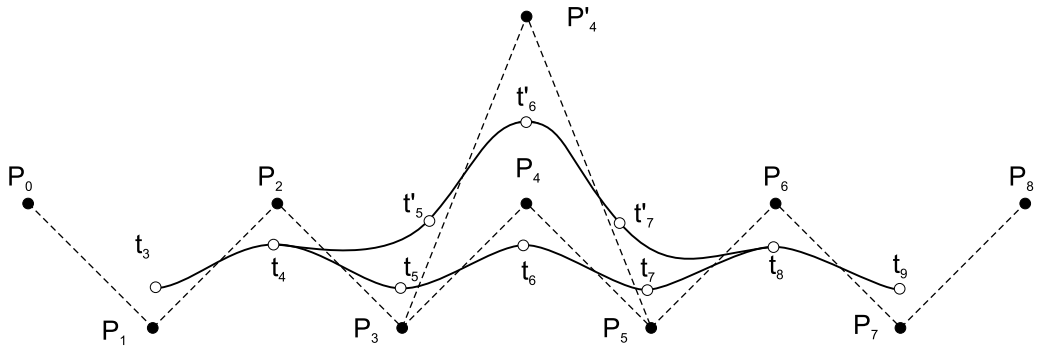
Díky tomu, že segment leží v konvexní obálce svého řídicího polygonu, je možné pochopit co se stane, je-li některý bod řídicího polygonu vícenásobný. Podívejme se na výchozí obrázek (5.18), kde jsou všechny řídicí body jednoduché a sledujme, co se bude dít, budeme-li zvyšovat násobnost bodu P_3.



Obrázek 5.21: Trojnásobný bod

Segment Q_4 je určen uzly t_4 a t_5 a řídicím polygonem P_1, P_2, P_3, P_4 a segment Q_5 uzly t_5 a t_6 a řídicím polygonem P_2, P_3, P_4, P_5. V případě, kdy není žádný bod několikanásobný, jsou všechny řídicí polygony čtyřúhelníky a konvexní obálky dvou po sobě následujících segmentů

(pro náš případ řídicí polygony segmentů Q_4 a Q_5) mají společnou plochu vymezenou trojúhelníkem P_2, P_3, P_4. V této oblasti také leží společný uzel t_5. Na obrázku 5.20 zdegenerovaly čtyřúhelníky P_1, P_2, P_3, P_4 a P_2, P_3, P_4, P_5 na trojúhelníky tím, že překrývající se oblastí je společná hrana P_2P_3. Na této hraně musí tedy ležet i uzel t_5. Na obrázku 5.21 je bod P_3 trojnásobným bodem, konvexní obálka P_2 až P_5 degeneruje na úsečku P_2P_3, stejně jako segment Q_5 degeneruje na úsečku ležící na P_2P_3.



Obrázek 5.22: Lokalita změny tvaru křivky při změně polohy bodu řídicího polygonu. Uzlové vektory jsou označeny prázdnými kolečky.

Lokalita změny znamená, že pokud změníme polohu některého z řídicích bodů, změní se tvar pouze té části křivky, která je tímto bodem určena. Pokud posuneme bod P_i kubiky $Q(t)$, změní se tvar čtyř segmentů Q_i, Q_{i+1}, Q_{i+2} a Q_{i+3}. Na obrázku 5.22 jsou zobrazeny dvě křivky. První je určena řídicím polygonem P_0, P_1, ..., P_8. Druhá se od první liší pouze tím, že bod P_4 je posunut do polohy P'_4. Uzly jsou označeny t_3 až t_9. Je patrné, že změnou polohy jediného bodu se změnila také poloha uzlů t_5 až t_7. Vždy dva uzly určují hranice segmentu, došlo tedy ke změně tvaru čtyř segmentů.

5.4.6 NURBS

Neuniformní racionální B-spline křivky (NURBS – *non uniform rational B-spline*) jsou dvojím zobecněním B-spline křivek uvedených v předcházející části. Termín *neuniformní* je odvozen od vzdálenosti uzlů ve smyslu parametru t, která nemusí být u těchto křivek konstantní. *Racionalita* znamená, že body jsou reprezentovány svými *homogenními souřadnicemi* (viz část 21).

Křivka NURBS je určena $n+1$ body $P_i, i = 0, \ldots, n$ řídicího polygonu, řádem B-spline k (nejvyšší stupeň polynomu je $k-1$) a uzlovým vektorem $\mathbf{U}$ délky $n+k+1$ a . Uzlový vektor je tvořen posloupností neklesajících reálných čísel – uzlových hodnot $t_0 \leq t_1, \leq \ldots, \leq t_{n+k}$. Uzlové hodnoty v této posloupnosti se mohou opakovat. Příkladem uzlového vektoru s uniformním

rozložením uzlových hodnot je $\mathbf{U}_1 = \{-2,-1,0,1,2,3,4,5\}$, tzv. okrajový uzlový vektor s opakovanými uzlovými hodnotami může mít tvar $\mathbf{U}_2 = \{0,0,0,1,1,1\}$.

Křivka NURBS je určena vztahem

$$Q(t) = \frac{\sum_{i=0}^{n} w_i P_i N_{i,k}(t)}{\sum_{i=0}^{n} w_i N_{i,k}(t)}, \tag{5.29}$$

kde w_i je váha i-tého bodu řídicího polygonu a $N_{i,k}(t)$ jsou *normalizované B-spline bázové funkce* definované rekurentním vztahem

$$N_{i,1}(t) = \begin{cases} 1 & \text{pro } t_i \le t < t_{i+1} \\ 0 & \text{jinde} \end{cases}$$

$$N_{i,k}(t) = \frac{t - t_i}{t_{i+k-1} - t_i} N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t) \text{ pro } t_i < t_{i+1+k},\ 0 \le i \le n. \tag{5.30}$$

Během výpočtu se vyskytují i výrazy, které mají ve jmenovateli nulu. Jejich hodnota je definitoricky položena rovna nule. Jiný zápis využívá *racionální B-spline báze*

$$R_{i,k} = \frac{w_i N_{i,k}(t)}{\sum_{j=0}^{n} w_j N_{j,k}(t)}. \tag{5.31}$$

Křivku NURBS podle rovnice (5.29) lze potom zapsat jednodušeji

$$Q(t) = \sum_{i=0}^{n} P_i R_{i,k}(t). \tag{5.32}$$

Obdobně jako u Bézierových křivek a bázových Bernsteinových polynomů (5.4.1) mají polynomy NURBS báze N_i^n následující vlastnosti:

1. $\forall\ i, n \in N \cup \{0\}$ a $t \in \langle 0,1\rangle$ je $N_i^n(t) \ge 0$.
2. $\sum_{i=0}^{n} R_i^n(t) = 1$ pro $t \in \langle 0,1\rangle$.
3. $N_{i,k}(t) = \frac{t-t_i}{t_{i+k-1}-t_i} N_{i,k-1}(t) + \frac{t_{i+k}-t}{t_{i+k}-t_{i+1}} N_{i+1,k-1}(t)$ pro $t_i < t_{i+1+k},\ 0 \le i \le n$.

První vztah zaručuje nezápornost polynomů B-spline báze. První a druhý pak zaručují, že výsledná křivka bude vždy ležet v konvexní obálce bodů řídicího polygonu. Třetí vztah je rekurentní definicí bázového polynomu řádu k (stupně $k-1$) pomocí lineární kombinace dvou po sobě následujících bázových polynomů řádu $k-1$ (stupně $k-2$). Třetí vlastnost je základem Cox-deBoorova algoritmu pro výpočet bodu na NURBS křivce.

Cox-deBoorův algoritmus Algoritmus 5.1 je zobecněním algoritmu de Casteljau (5.18) a je navržen pro spline křivky s neuniformní parametrizací. S využitím rekurentní definice bázových polynomů je bod na křivce NURBS konstruován postupným dělením úseček té části řídicího polygonu, která ovlivňuje segment křivky, k němuž tento bod náleží. Po nalezení příslušného intervalu v uzlovém vektoru (krok 2), je aplikováno dělení (krok 3), které respektuje neuniformní časové intervaly určené uzlovým vektorem.

Vstup: Uzlový vektor $\{t_i\}_{i=0}^{n+k+1}$ a řídicí body $\{P_i\}_{i=0}^{n}$.
1. $P_i^{(0)} = P_i;\ i = 0, \ldots,\ n.$
2. Pro dané $t \geq\ t_k$ nalezni interval $t \in [t_j,\ t_{j+1})$.
3. Pro $p = 1, \ldots, k$
   pro $i = j - k + p, \ldots, j$
   $P_i^{(p)} = \frac{t - t_i}{t_{i+k-(p-1)} - t_i} P_i^{(p-1)} + \frac{t_{i+k-(p-1)} - t}{t_{i+k-(p-1)} - t_i} P_{i-1}^{(p-1)}$
4. $P(t) = P_j^{(k)}$.

Algoritmus 5.1: Cox–deBoorův algoritmus pro výpočet bodu na NURBS křivce

Jedním z nejdůležitějších modelovacích prostředků, který poskytují NURBS křivky, je možnost vložení nových uzlových bodů. Namísto zvyšování stupně polynomu umožňují nově vložené uzlové body společne s příslušně pozměněnými řídicími body vymezit ty části křivky, které chceme lokálně ovlivnit. Vložení uzlového bodu zajistí následující postup, který ukážeme na příkladu neracionální křivky $Q(t)$ stupně k.

Je dána křivka $Q(t) = \sum_{i=0}^{n} P_i N_{i,k}(t)$ s uzlovým vektorem $\{t_0, \ldots, t_{n+k}\}$. Vkládaný uzlový bod má hodnotu $t' \in (t_0, \ldots, t_{n+k})$. Hledáme nové řídicí body P_i', pro něž $Q(t) = \sum_{i=0}^{n+1} P_i' N_{i,k}(t)$. Pro nové řídicí body P_i' platí:

$$P_0' = P_0\ ,\ P_{n+1}' = P_n\ ,\ P_i' = (1 - \alpha_i) P_{i+1} + \alpha_i P_i \tag{5.33}$$

$$\alpha_i = \begin{cases} 1 & i = 1, \ldots, j-k \\ \frac{t' - t_i}{t_{i+k} - t_i} & i = j-k+1, \ldots, j \\ 0 & i = j+1, \ldots, n \end{cases} \tag{5.34}$$

Křivky NURBS mají tyto vlastnosti:

1. Okrajový uzlový vektor

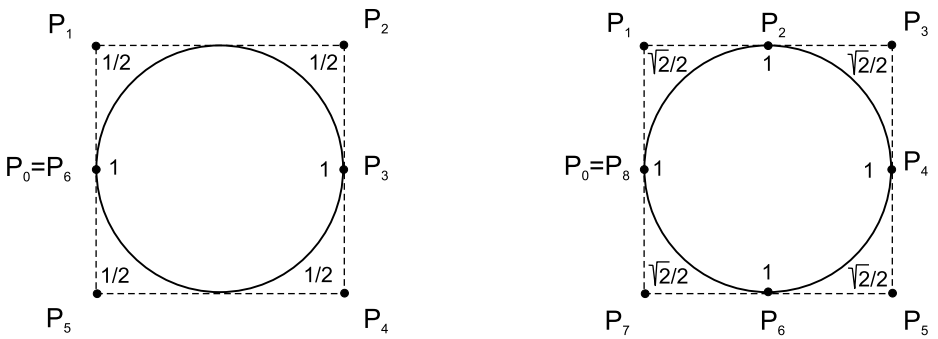
$$\mathbf{U} = \{\underbrace{\alpha, \alpha, \ldots, \alpha}_{k}, \underbrace{t_k, \ldots, t_{n-k}}_{n+1-k}, \underbrace{\beta, \beta, \ldots, \beta}_{k}\}.$$

zajistí, že křivky procházejí prvním a posledním bodem řídicího polygonu.

2. Leží v konvexní obálce svého řídicího polygonu, stejně tak jejich jednotlivé segmenty leží v obálkách svých řídicích polygonů. Změna polohy, resp. váhy jednoho bodu má tedy vliv pouze na část křivky.
3. Jsou invariantní vůči transformacím a vůči rovnoběžnému a středovému promítání.
4. Pomocí váhových koeficientů umožňují přesně vyjádřit kuželosečky jako podíl polynomů.
5. Pro uzlový vektor

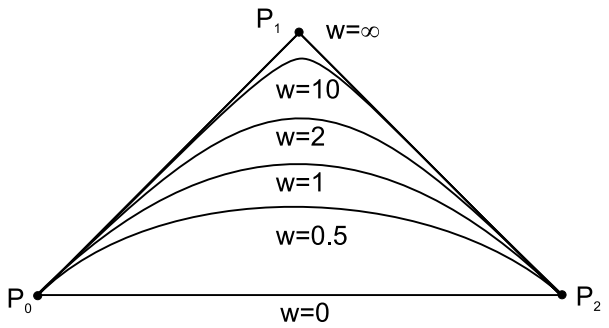
$$\mathbf{U} = \{\underbrace{0, 0, \ldots, 0}_{k}, \underbrace{1, 1, \ldots, 1}_{k}\}$$

jsou normalizované B-spline báze rovny Bernsteinovým polynomům stupně k a NURBS je racionální Bézierovou křivkou. Pro hodnoty $w_i = 1$ je NURBS Bézierovou křivkou tak, jak bylo uvedeno v části 5.4.1.



Obrázek 5.23: Kružnice jako NURBS: sedm bodů (vlevo) a devět bodů (vpravo)

Zejména čtvrtá vlastnost je důležitá pro tvorbu jednotných datových struktur, které reprezentují jak „klasické“ tvary, jako jsou kružnice, elipsa či čtverec, tak volné tvary (*free-form*). Nevýhodou je, že reprezentace „klasických“ objektů není jednoduchá. Tak například kružnice může být reprezentována sedmi body s vahami $[1, 1/2, 1/2, 1, 1/2, 1/2, 1]$ (obrázek 5.23 vlevo) a uzlovým vektorem $\mathbf{U} = \{0, 0, 0, 1/4, 1/2, 1/2, 3/4, 1, 1, 1\}$, nebo devíti body (čtyřmi 90° kruhovými oblouky) s vahami $[1, \sqrt{2}/2, 1, \sqrt{2}/2, 1, \sqrt{2}/2, 1, \sqrt{2}/2, 1]$ (obrázek 5.23 vpravo) a uzlovým vektorem $\mathbf{U} = \{0, 0, 0, 1, 1, , 2, 2, 3, 3, 4, 4, 4\}$. Za cenu trochu komplikovanější reprezentace jednoduchých tvarů však NURBS poskytují jednotný přístup k mnoha zdánlivě různorodým modelovacím prvkům a modelování a zobrazování je možné řešit společnou skupinou optimalizovaných algoritmů.



Obrázek 5.24: Vliv váhy bodu na racionální B-spline

Obrázek 5.24 ukazuje vliv váhy bodu P_1 na tvar křivky. Pro hodnotu $w = 0$ bod křivku žádným způsobem neovlivňuje [srovnej vztah (5.29)]. Váha bodu $w = 1$ odpovídá případu neracionální křivky. Se zvyšující se váhou se křivka k bodu přimyká, až konečně pro hodnotu $w = \infty$ křivka bodem prochází a dochází ke ztrátě spojitosti. Podmínka nezápornosti vah zaručuje umístění křivky v konvexní obálce.

5.5 Vlastnosti parametrických ploch

V této části popíšeme plochy nejčastěji používané v trojrozměrné počítačové grafice, způsob jejich zadávání a editace, principy jejich výpočtu, zobrazování a převodu na sítě trojúhelníků.

Bodovou rovnicí parametrické plochy $Q(u, v)$ dvou parametrů u a v (srovnej odstavec 5.1) budeme rozumět funkci

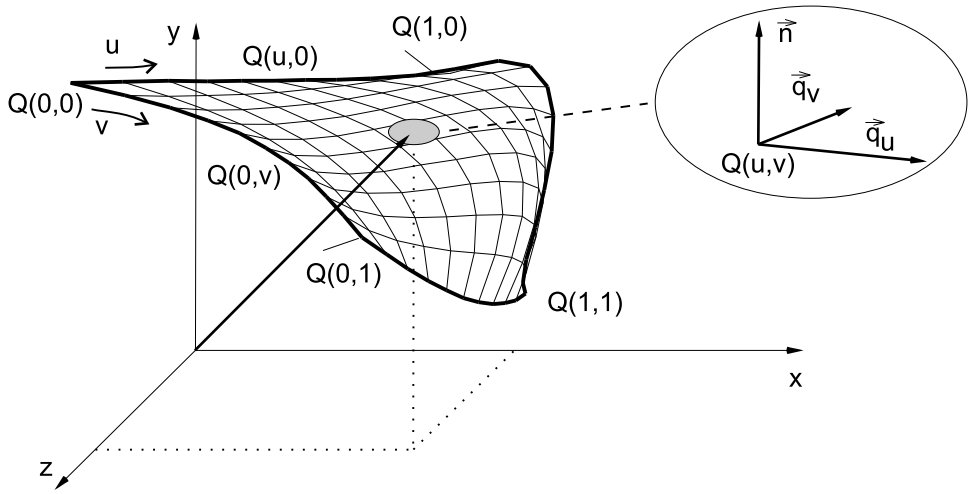
$$Q(u, v) = [x(u, v), y(u, v), z(u, v)], \tag{5.35}$$

kde $x(u, v)$, $y(u, v)$ a $z(u, v)$ jsou funkce dvou parametrů $u, v \in \langle 0, 1\rangle$. Bod Q o souřadnicích $[x, y, z]$ v trojrozměrném kartézském prostoru má souřadnice $[u, v]$ v prostoru parametrickém. Funkce $x(u, v)$, $y(u, v)$ a $z(u, v)$ jsou obvykle polynomiální, s ohledem na výhodné vlastnosti při modelování a navazování. Parametrické polynomiální plochy používají shodné matematické prostředky jako parametrické polynomiální křivky, viz část 5.

Tečný vektor $\vec{q}_u(u, v)$ ve směru parametru u k ploše $Q(u, v)$ (viz obrázek 5.25) a tečný vektor $\vec{q}_v(u, v)$ ve směru parametru v jsou určeny vztahy

$$\vec{q}_u(u, v) = \frac{\partial Q(u, v)}{\partial u} = \left(\frac{\partial x(u, v)}{\partial u}, \frac{\partial y(u, v)}{\partial u}, \frac{\partial z(u, v)}{\partial u}\right), \tag{5.36}$$

$$\vec{q}_v(u, v) = \frac{\partial Q(u, v)}{\partial v} = \left(\frac{\partial x(u, v)}{\partial v}, \frac{\partial y(u, v)}{\partial v}, \frac{\partial z(u, v)}{\partial v}\right). \tag{5.37}$$



Obrázek 5.25: Parametrická plocha

Pokud bude z kontextu jasné že se jedná o plochu, budeme zápis $Q(u,v)$ zjednodušovat zápisem Q, analogicky budeme používat zápis $\vec{q}_u$ pro tečný vektor ve směru parametru u k ploše $Q(u,v)$ atp. Parametrická rovnice *tečné roviny* $T(r,s)$ k ploše Q je určena tečnými vektory $\vec{q}_u$ a $\vec{q}_v$, bodem $Q(u,v)$ a má tvar

$$T(r,s) = Q(u,v) + r\ \vec{q}_u + s\ \vec{q}_v;\ r,s \in R. \tag{5.38}$$

Při řešení úlohy navazování plátů mají také *význam zkruty, zkrutové vektory*, které charakterizují „vyklenutí“ plochy v místech napojení. Jsou definovány pomocí smíšených parciálních derivací podle parametrů u a v vztahem

$$\vec{q}_{uv}(u,v) = \frac{\partial^2 Q(u,v)}{\partial u \partial v} = \left(\frac{\partial^2 x(u,v)}{\partial u \partial v}, \frac{\partial^2 y(u,v)}{\partial u \partial v}, \frac{\partial^2 z(u,v)}{\partial u \partial v} \right) . \tag{5.39}$$

Pro výpočet *normály* (kolmice) $\vec{n}$ k ploše v bodě Q (viz lupa na obrázku 5.25) využijeme znalosti tečných vektorů $\vec{q}_u$, $\vec{q}_v$ a normálu určíme jako jejich normalizovaný vektorový součin

$$\vec{n} = \frac{\vec{q}_u \times \vec{q}_v}{|\vec{q}_u \times \vec{q}_v|} \tag{5.40}$$

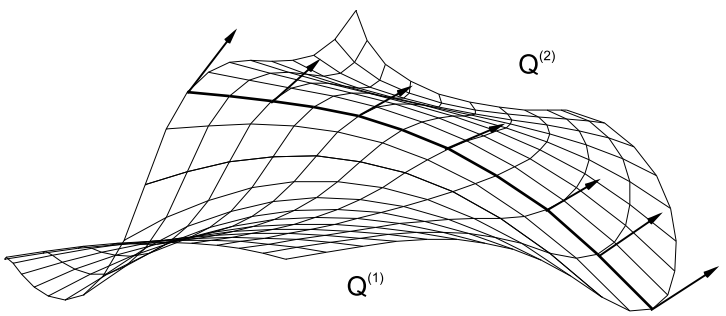
Hlavní křivka plochy ve směru parametru u je každá křivka určena rovnicí $Q(u,k)$ pevného parametru $v = k$ a proměnného parametru u a analogicky hlavní křivka plochy ve směru

parametru v je každá křivka určená rovnicí $Q(k, v)$ pevného parametru $u = k$ a proměnného parametru v.

Rohy plochy jsou body $Q(0, 0)$, $Q(1, 0)$, $Q(0, 1)$ a $Q(1, 1)$. *Strany* plochy $Q(u, v)$ jsou hlavní křivky ve směru u pro hodnoty $v = 0$, resp. $v = 1$ a ve směru v pro hodnoty $u = 0$, resp. $u = 1$. Všechny strany plochy dohromady tvoří její *okraj*.

Podobně jako se křivky při navazování skládají ze segmentů, tak i plochy se skládají z částí, kterým říkáme *pláty* (*patch*). Složitější plochy získáváme navazováním plátů, případně dělením plochy na pláty, které můžeme podrobněji tvarovat. Pokud hovoříme o navazování ploch, obvykle používáme pojem *plátování*. Při navazování plátů budeme pracovat s analogickými pojmy jako při navazování křivek. Budeme používat parametrickou C a geometrickou G spojitost (viz odstavec 5.1).

Dva pláty mají napojení C^0, mají-li společnou stranu, která je křivkou třídy alespoň C^0. Dva pláty mají spojité napojení C^1, mají-li společnou stranu a jsou-li shodné příčné parciální derivace ve všech bodech společné strany prvního i druhého plátu (viz obrázek 5.26). Jinými slovy řečeno, je-li strana C^1 spojitá křivka a příčné parciální derivace k této straně jsou identické pro první i druhý plát. Někdy je požadována C^1 spojitost pouze ve směru napojení.



Obrázek 5.26: C^1 spojení dvou plátů $Q^{(1)}$ a $Q^{(2)}$. Silně je vyznačena C^1 spojitá společná strana obou plátů, šipkami jsou naznačeny příčné tečné vektory ve směru napojení

Dva pláty jsou G^1 spojité, mají-li společnou stranu, která je alespoň G^1 spojitou křivkou a jsou-li parciální derivace obou plátů podél této strany ve směru napojení lineárně závislé s koeficientem $k > 0$, který se spojitě mění podél této společné strany [Fari93].

Plochy se zadávají *řídicími body* a *bázovými funkcemi*. Většina ploch, které se v trojrozměrném modelování používají, jsou plochy *aproximační*. *Interpolace* v dimenzích vyšších nežli dvě je překvapivě složitou úlohou, a proto se pro interpolaci řídicích bodů ve třech dimenzích (*surface fitting* nebo *scattered data interpolation*), obyčejně používá interpolace ploškami, nejčastěji trojúhelníky (viz odstavec 7.3.1). Při aproximaci určuje poloha řídicích bodů tvar výsledné

plochy, tato plocha jimi však nemusí procházet. Pro napojování ploch je důležité, aby byly známé tečné podmínky na jejich stranách.

Stejně jako u křivek, tak i v případě ploch jsou bázové funkce nejčastěji polynomy, protože jsou snadno diferencovatelné a lze je rychle vyčíslit. Při pojmenování ploch se používá stupeň použitého polynomu, a protože se jedná o plochy, je nutné v jejich názvu uvést vždy dva údaje. Tak například plocha bikubická má jako hlavní křivky v obou směrech kubiky, řez plochou kvadraticko-lineární je kvadrikou ve směru parametru u, zatímco hlavními křivkami ve směru parametru v jsou úsečky, atp.

Nejčastěji se používají polynomy stupně tři. Polynom stupně menšího nežli tři nelze určit tak, aby při procházení dvěma body měl zároveň „vhodné“ tečné vlastnosti. Kubika je polynom nejmenšího stupně, který tyto vlastnosti má, umožňuje C^1 a C^2 spojitost a poskytuje uspokojivé modelovací možnosti. Použití polynomů vyššího stupně vede ke zvýšení časové náročnosti výpočtů. Protože kubiky umožňují C^2 spojitost, je snadné jejich napojování – skládání složitých ploch z jednodušších (obyčejně bikubických) plátů.

Podobně jako u parametrických křivek se pro parametrické plochy volí reprezentace, ve které se místo uchování koeficientů jednotlivých polynomů kombinují řídicí body, tečné vektory a další geometrické parametry s bázovými funkcemi

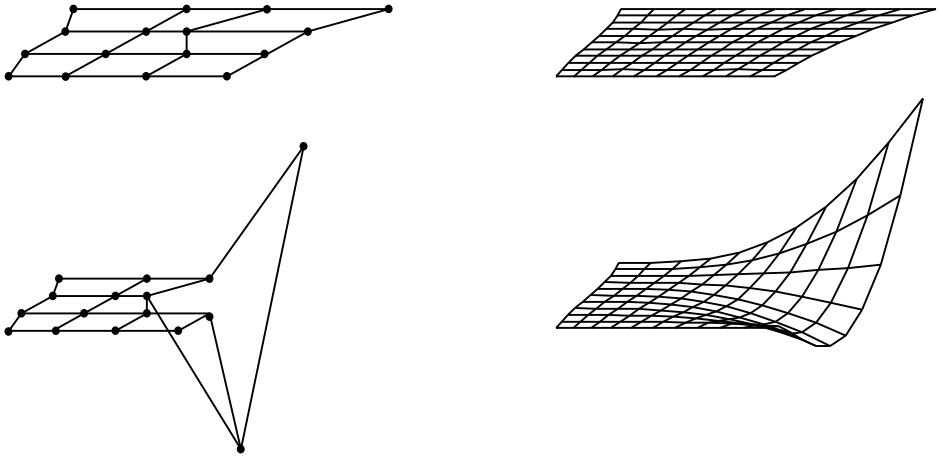
$$Q(u,v) = \mathbf{U}\ \mathbf{M}_B P \mathbf{M}_B^T\ \mathbf{V}^T = [u^3 u^2 u\ 1]\ \mathbf{M}_B P \mathbf{M}_B^T \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}. \tag{5.41}$$

Matice P obsahuje řídicí body a případně další prvky určující geometrii plochy, bázová matice $\mathbf{M}_B$ obsahuje koeficienty a_i polynomů bázových funkcí. Bázová matice $\mathbf{M}_B$ se nemění, zatímco řídicí body $P_{i,j}$ svou polohou v prostoru určují tvar plochy. Konkrétní tvary matic $\mathbf{M}_B$ uvedeme v dalších odstavcích.

Modelování obvykle probíhá zadáváním a úpravami sítě řídicích bodů (viz obrázek 5.27), která tvoří v trojrozměrném prostoru mnohostěn. Změnou polohy řídicích bodů měníme tvar plochy.

Mezi často požadované vlastnosti ploch patří:

1. *Invariance* k lineárním transformacím a projekcím, která zaručuje, že například otáčení sítě řídicích bodů a následné generování plochy má stejný výsledek, jako otáčení každého bodu z vygenerované plochy.
2. *Vlastnost konvexní obálky* (*convex hull property*)
   (a) *silná podmínka* – plocha leží v konvexní obálce všech svých řídicích bodů,



Obrázek 5.27: Editace Bézierovy bikubické plochy (vpravo). Řídicí síť se šestnácti body (vlevo) určuje změnu tvaru plochy po změně polohy dvou řídicích bodů

(b) *slabá podmínka* – část plochy leží v konvexní obálce některých řídicích bodů (typicky plát, v obálce svých řídicích bodů).

3. *Lokalita změn* – změnou polohy (u racionálních ploch i váhy) řídicího bodu se mění jen část plochy, nikoli plocha celá.
4. Plocha může *procházet krajními body* sítě řídicích bodů.

Podmínky konvexní obálky lze s výhodou využít k urychlení některých výpočtů. Například v robotice, v číslicově řízeném obrábění, nebo v trojrozměrné počítačové animaci lze využít toho, že výpočet kolize s mnohostěnem je obyčejně daleko snazší, nežli výpočet kolize s plochou. Podmínka konvexní obálky zaručuje, že pokud nedošlo ke kolizi konvexních obálek sledovaných objektů (například objektu, který je nesen robotem v tovární hale), nemohlo dojít ke kolizi s plochami, které jsou uvnitř těchto obálek. Jinou aplikací, která využívá vlastnosti konvexní obálky, je urychlení výpočtu metody sledování paprsku (kapitola 14.2.1). Pokud vržený paprsek nezasáhl konvexní obálku plochy, nemá smysl provádět (obyčejně náročné) výpočty vedoucí k získání průsečíku paprsku a plochy.

5.6 Interpolační plochy

Zobecnění metody interpolace posloupnosti bodů křivkou naráží v případě ploch na výrazné problémy. Zatímco existuje vyvinutý aparát pro popis a studium polynomiálních křivek v rovině

i v prostoru, o plochách to již neplatí. Podstatným důvodem je to, že interpolace prostorových dat vyžaduje velmi rozsáhlé výpočty.

Ve trojrozměrném prostoru mějme $(m+1)\times(n+1)$ bodů P_{ij}, $i = 0, 1, ..., m$ a $j = 0, 1, ..., n$. Interpolační plochou rozumíme takovou plochu $P(u, v)$, pro kterou existuje řešení (u, v) rovnic:

$$P(u, v) = P_{ij}, \forall i, j.$$

Interpolační plocha tedy prochází body, kterými je zadána. Možným řešením je *interpolace polynomy*. Pravá strana předchozího vztahu má tvar

$$P(u, v) = \sum_{i=0}^{m} \sum_{j=0}^{n} b_{ij} u^i v^j \ .$$

Temto vztah je možno zapsat také maticově:

$$P(u, v) = \mathbf{U}.\mathbf{B}.\mathbf{V}^T \tag{5.42}$$

kde $\mathbf{U} = [u^m, u^{m-1}, \ldots, u, 1]$ a $\mathbf{V} = [v^n, v^{n-1}, \ldots, v, 1]$. Vztah (5.42) je zápisem $3 \times (m+1)\times$ $\times (n+1)$ rovnic pro tentýž počet neznámých koeficientů b_{ij}. Pro bikubickou interpolační plochu tedy dostáváme $3 \times (3+1) \times (3+1) = 48$ rovnic.

Požadavek průchodu zadanými body, který je kladen na interpolační plochy, je velmi přísný. Při vyšších stupních polynomů vede k nepříjemnému vlnění ploch v jednotlivých směrech. Navíc je představa konstruktéra o takto vznikající ploše často odlišná od reality. Metody interpolace bodů plochami jsou pro tyto důvody v počítačové grafice málo využívány. Jejich nasazení má význam např. v numerické matematice a ve shlukové analýze.

5.7 Aproximační plochy

Motivy, které vedly k rozvoji metod popsaných v této kapitole, jsou charakterizovány především snahou odstínit uživatele od matematiky a zaručit maximální jednoduchost tvorby generovaných ploch. Uživatel tak obyčejně zadává řídicí body (např. rohy), okrajové křivky, tečné vektory, či zkruty a jejich pomocí modeluje tvar výsledné plochy. Rozmístění těchto prvků v matici geometrických podmínek znázorníme pomocí symbolů:

- bod, okrajová křivka, plocha : •
- tečné vektory, resp. funkce, které je popisují: $\rightarrow \downarrow$
- zkruty: $\nearrow \nwarrow \searrow \swarrow$

5.7.1 Hermitovské plochy

Hermitovské plochy jsou zadávány rohy a tečnými vektory v nich, případně navíc ještě zkruty. Jejich tvar je řízen Hermitovskými polynomy třetího stupně, definovanými vztahem (5.13) s průběhy na obrázku 5.8. Změny tvaru plochy je docíleno manipulací s tečnými vektory a zkruty, případně změnou polohy rohových bodů.

5.7.2 Dvanáctivektorová plocha

Dvanáctivektorová plocha je zadána čtyřmi body $P_{00}, P_{01}, P_{10}, P_{11}$ a tečnými vektory v obou směrech v každém z nich, tj. $\vec{p}_u(0,0), \vec{p}_v(0,0), \ldots, \vec{p}_u(1,1), \vec{p}_v(1,1)$. Název této plochy plyne z počtu vstupních údajů (polohových a tečných vektorů), jichž je potřeba pro její generování. Rovnice, podle které se určí dvanáctivektorová plocha, má tvar:

$$Q(u,v) = [F_3(u), F_1(u), F_2(u), F_4(u)] \cdot \mathbf{M} \cdot [F_3(v), F_1(v), F_2(v), F_4(v)]^T \ ,$$

$F_1(u), \ldots, F_4(v)$ jsou kubické Hermitovské polynomy (5.13). Matice $\mathbf{M}$ je mapou plochy, neboť uchovává geometri plochy a její vstupní údaje jsou v ní uloženy takto:

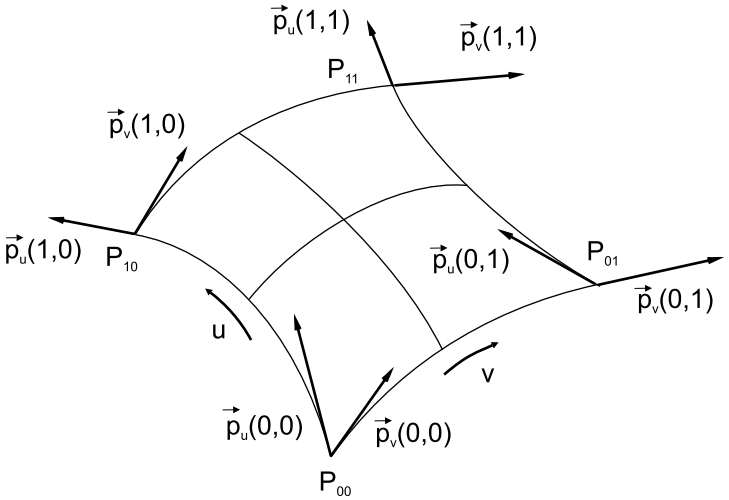
$$\mathbf{M} = \begin{pmatrix} 0 & \vec{p}_u(0,0) & \vec{p}_u(0,1) & 0 \\ \vec{p}_v(0,0) & P_{00} & P_{01} & \vec{p}_v(0,1) \\ \vec{p}_v(1,0) & P_{10} & P_{11} & \vec{p}_v(1,1) \\ 0 & \vec{p}_u(1,0) & \vec{p}_u(1,1) & 0 \end{pmatrix} \sim \begin{pmatrix} & \downarrow & \downarrow & \\ \rightarrow & \bullet & \bullet & \rightarrow \\ \rightarrow & \bullet & \bullet & \rightarrow \\ & \downarrow & \downarrow & \end{pmatrix} , \qquad (5.43)$$

kde $u, v \in \langle 0, 1 \rangle$. Obrázek 5.28 demonstruje rozložení tečných vektorů a bodů na skutečné ploše.

Dosazením krajních hodnot intervalu do rovnice (5.43) lze snadno ověřit, že okrajové křivky jsou Hermitovské kubiky, což se uplatní při jejich navazování. Při navazování plátů složených z dvanáctivektorových ploch je přirozeným požadavkem jejich spojité a hladké spojení. Podmínka spojení C^0 dvou plátů je splněna při identitě příslušných sousedních stran. Hermitovská kubika, která je stranou dvanáctivektorové plochy, je zadána dvěma body a dvěma tečnými vektory. Shodnost řídicích bodů okraje a tečných vektorů zaručuje hladké spojení C^1 dvou dvanáctivektorových ploch. V implementaci je tato podmínka splněna, pokud matice hladce spojených ploch mají ve dvou sloupcích nebo řádcích identické hodnoty.

5.7.3 Šestnáctivektorová plocha

Nahrazením nulových vektorů v matici dvanáctivektorové plochy pomocí zkrutů získáme plochu šestnáctivektorovou. Ta je zadána čtyřmi body $P_{00}, P_{01}, P_{10}, P_{11}$, tečnými vektory



Obrázek 5.28: 12ti vektorový Hermitovský plát

v obou směrech v každém z nich, tj. $\vec{p}_u(0,0), \vec{p}_v(0,0), \ldots, \vec{p}_u(1,1), \vec{p}_v(1,1)$ a čtyřmi zkruty $\vec{p}_{uv}(0,0), \vec{p}_{uv}(0,1), \vec{p}_{uv}(1,0), \vec{p}_{uv}(1,1)$. Šestnáctivektorová plocha $P(u,v)$ je určena vztahem

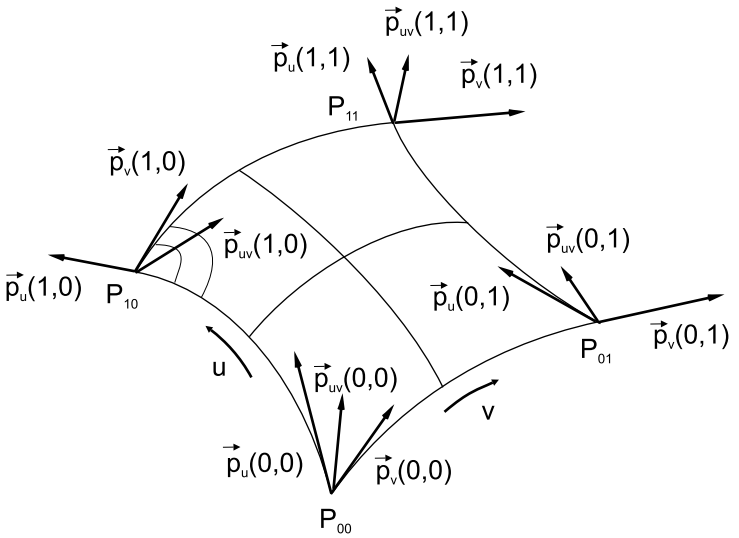
$$Q(u,v) = [F_3(u), F_1(u), F_2(u), F_4(u)].\mathbf{M}.[F_3(v), F_1(v), F_2(v), F_4(v)]^T,\ u,v \in \langle 0,1 \rangle\ .$$

Matice $\mathbf{M}$ má význam mapy plochy:

$$\mathbf{M} = \begin{pmatrix} \vec{p}_{uv}(0,0) & \vec{p}_u(0,0) & \vec{p}_u(0,1) & \vec{p}_{uv}(0,1) \\ \vec{p}_v(0,0) & P_{00} & P_{01} & \vec{p}_v(0,1) \\ \vec{p}_v(1,0) & P_{10} & P_{11} & \vec{p}_v(1,1) \\ \vec{p}_{uv}(1,0) & \vec{p}_u(1,0) & \vec{p}_u(1,1) & \vec{p}_{uv}(1,1) \end{pmatrix} \sim \begin{pmatrix} \nwarrow & \downarrow & \downarrow & \nearrow \\ \rightarrow & \bullet & \bullet & \rightarrow \\ \rightarrow & \bullet & \bullet & \rightarrow \\ \swarrow & \downarrow & \downarrow & \searrow \end{pmatrix}.$$

Obrázek 5.29 znázorňuje rozložení tečných vektorů a bodů na skutečné ploše, které je shodné s 12-ti vektorovým plátem. Navíc jsou zakresleny zkrutové vektory v rozích plátu. Význam zkrutu je především v tom, že řídí vyklenutí plochy v rohu plátu. V případě dvanáctivektorových ploch je toto vyklenutí realizováno změnou dvou tečných vektorů v okolí příslušného vrcholu. Daní za tuto výhodu je zvýšená výpočetní složitost.

Navazování plátů složených z šestnáctivektorových Hermitovských ploch se řídí stejnými pravidly jako navazování ploch dvanáctivektorových. Podmínka spojení C^0 dvou plátů je tedy splněna při identitě příslušných sousedních stran. To lze opět zaručit překrytím dvou sloupců (resp. řádků) ploch, jež mají být spojeny hladce. Navíc je možné pomocí zkrutových vektorů zajistit hladké spojení G^1 mezi čtyřmi pláty, které jsou spojeny v jednom rohu.



Obrázek 5.29: 16ti vektorový Hermitovský plát se zkruty, znázorněno vyklenutí v okolí rohu $P(1,0)$

5.7.4 Plochy spojující dvě křivky

Častým požadavkem technické praxe je vytvoření ploch, které spojují jiné plochy v jejich okrajích. Může se jednat o navázání dvou zužujících se trubek, kdy známe okrajové křivky na jejich koncích atp. Nejprve popíšeme jednoduchý typ spojovací plochy, tzv. přímkovou plochu, poté uvedeme příklad spojovací plochy s obecnějším průběhem.

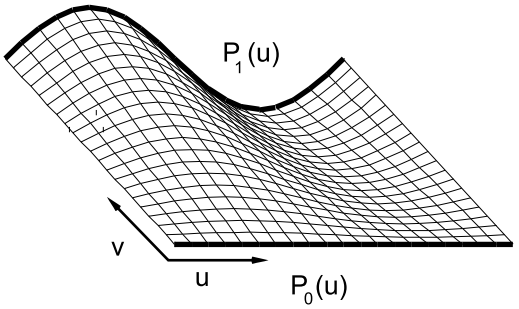
Přímková plocha spojuje dvě křivky přímkami. Obrázek 5.30 ukazuje plochu vytvořenou z úsečky jako jedné řídicí křivky a z části sinusoidy jako řídicí křivky druhé.

Přímková plocha je zadána dvěma okrajovými křivkami $P(0,v)$ a $P(1,v)$ s definičním oborem $v \in \langle 0,1 \rangle$. Rovnice přímkové plochy, jež spojuje křivky $P(0,v)$ a $P(1,v)$, má tvar:

$$Q(u,v) = (1-u).P(0,v) + u.P(1,v) \ , u \in \langle 0,1 \rangle \ . \tag{5.44}$$

Dosazením do uvedeného vztahu za body na okraji definičního oboru snadno ověříme, že výsledná plocha spojuje zadané křivky. Pro úplnost uveďme, že vztah pro přímkovou plochu je možno zapsat ve tvaru:

$$Q(u,v) = [1-u, u\ ] \begin{bmatrix} P(0,v) \\ P(1,v) \end{bmatrix} = [1-u, u\ ].\mathbf{M} \ . \tag{5.45}$$



Obrázek 5.30: Přímková plocha získaná spojením dvou okrajových křivek

V tomto vztahu má **M** význam mapy plochy obsahující okrajové křivky $P(0,v), P(1,v)$.

Kubická plocha spojující okrajové křivky je zadána těmito křivkami obdobně jako u přímkových ploch. Navíc doplníme dvě vektorové funkce pro tečné vektory podél těchto okrajových křivek, tj. $\vec{p}_u(0,v), v \in \langle 0,1\rangle$ a $\vec{p}_u(1,v), v \in \langle 0,1\rangle$. Průběhy tečných vektorů můžeme převzít od ploch, které v daných okrajích spojujeme s cílem zajistit hladké spojení C^1.

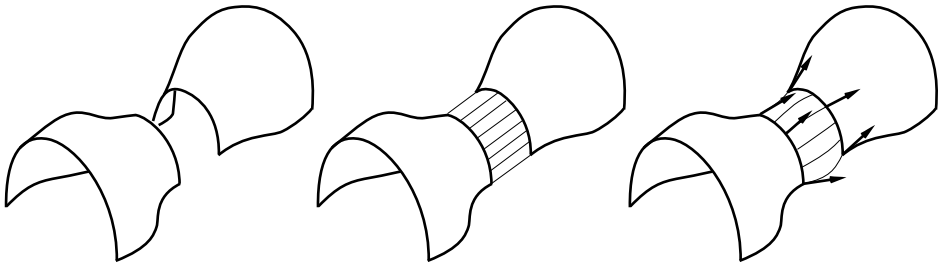
Křivky jednoznačně určují tvar plochy na jejích dvou okrajích a tečné vektory formují její tvar mezi těmito okraji. Výsledná plocha je vyjádřena vztahem:

$$Q(u,v) = F_1(u)P(0,v) + F_2(u)P(1,v) + F_3(u)\vec{p}_u(0,v) + F_4(u)\vec{p}_u(1,v),\ u,v \in \langle 0,1\rangle, \quad (5.46)$$

kde F_i jsou Hermitovské polynomy třetího stupně. Rovnici plochy lze zapsat maticově:

$$P(u,v) = [F_3(u), F_1(u), F_2(u), F_4(u)] \begin{pmatrix} \vec{p}_u(0,v) \\ P(0,v) \\ P(1,v) \\ \vec{p}_u(1,v) \end{pmatrix} = \mathbf{F}.\mathbf{M};\ \mathbf{M} \sim \begin{pmatrix} \downarrow \\ \bullet \\ \bullet \\ \downarrow \end{pmatrix}, \quad (5.47)$$

kde symboly $\bullet$ a $\downarrow$ představují bodové a vektorové funkce okraje. Po dosazení do P(u,v) za $u = konst$ zjistíme zjistíme, že křivka $P(konst., v)$ je Hermitovskou kubikou. Obecná rovnice tedy dané propojení okrajů plátu řeší pomocí Hermitovské interpolace. Porovnání obou typů propojení plátů v místě okrajových křivek je znázorněno na obrázku 5.31.



Obrázek 5.31: Spojení plátů v okrajích: spojované pláty (vlevo), spojení přímkovou plochou (uprostřed), spojení obecnější plochou s danými průběhy tečných vektorů (vpravo)

5.8 Plochy zadané okrajem

Mezi časté úlohy řešené při modelování a plátování patří konstrukce ploch, které jsou určené svým celým *okrajem*. Na rozdíl od předchozích ploch jsou zadány všechny křivky určující okraj, např. čtyři v případě čtyřúhelníkového plátu. Zde uvedeme několik příkladů konstrukce těchto ploch.

5.8.1 Bilineární Coonsova plocha

Bilineární Coonsova plocha je jednoznačně určena čtyřmi křivkami $P(u,0)$, $P(u,1)$, $P(0,v)$ a $P(1,v)$. Tyto křivky musí tvořit uzavřený okraj budoucího plátu. Coonsova bilineární plocha je určena rovnicí

$$[1-u,-1,u\,].\mathbf{C}.[1-v,-1,v]^T = 0 \tag{5.48}$$

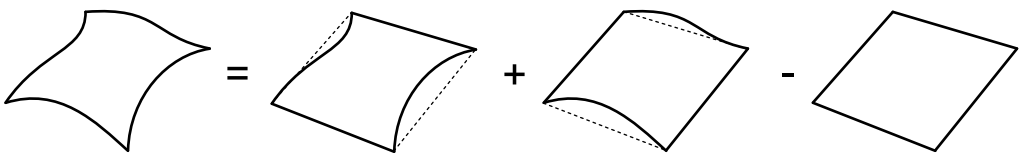
Matice $\mathbf{C}$ má tvar:

$$\mathbf{C} = \begin{pmatrix} P_{00} & P(0,v) & P_{01} \\ P(u,0) & Q(u,v) & P(u,1) \\ P_{10} & P(1,v) & P_{11} \end{pmatrix} \sim \begin{pmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{pmatrix}.$$

Plocha je definována pro parametry u, v na jednotkovém čtverci. Bod $P(u,v)$ uprostřed matice $\mathbf{C}$ je hledaným řešením implicitní rovnice (5.48) pro určité hodnoty parametrů u, v a je to bod ležící na výsledné ploše. Jedná se tedy o implicitní zadání, které je nevhodné pro generování ploch. Toto zadání je tedy potřeba převést na tvar explicitní. Po úpravě dostaneme pro hledanou plochu $P(u,v)$ rovnici

$$Q(u,v) = [1-u,u\,]\begin{bmatrix} P(0,v) \\ P(1,v) \end{bmatrix} + [P(u,0),P(u,1)]\begin{bmatrix} 1-v \\ v \end{bmatrix} - \\ -[1-u,u\,]\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}\begin{bmatrix} 1-v \\ v \end{bmatrix}. \quad (5.49)$$

Zatímco implicitní rovnice (5.48) vyjadřuje podmínky, které musí plocha splňovat, explicitní vyjádření (5.49) je řešením této rovnice a je vhodnější pro výpočet bodů plochy. Protože je plocha zadána svým okrajem a v rozích matice **C** jsou vyžadovány údaje o bodech, je nutno ze zadaného okraje tyto body spočítat. Dosazením jedničky a nuly za u, v do (5.49) ověříme, že generovaná plocha prochází zadaným okrajem. Pokud dvě protější strany okraje budou úsečkami, vygenerujeme tímto způsobem přímkovou plochu. Bilineární plocha je tedy zobecněním plochy přímkové. Obrázek 5.32 znázorňuje výpočet bilineární plochy, který je obsažen v explicitním vyjádření. V obou směrech jsou protilehlé okraje propojeny přímkovými plochami a poté je odečtena rozdílová plocha.



Obrázek 5.32: Bilineární Coonsova plocha

Zásadním problémem bilineární Coonsovy plochy (a jak uvidíme dále i plochy bikubické) je její plátování. Ze zadaného okraje nelze jednoduše vyjádřit příčné tečné vektory a není tedy snadné vytvořit hladké spojení dvou plátů složených z Coonsových bilineárních, či bikubických plátů. Tyto plochy se tedy pro hladké spojení nepoužívají. Spojité navázání G^0 se zaručí identitou okrajových křivek.

5.8.2 Bikubická plocha

Bikubická plocha zadaná okrajem se určí opět čtyřmi obecnými křivkami $P(u,0)$, $P(u,1)$, $P(0,v)$, a $P(1,v)$. Tato plocha je popsána vztahem:

$$[F_1(u),-1,F_2(u)].\mathbf{C}.[F_1(v),-1,F_2(v)]^T = 0\ ,\ u,v \in \langle 0,1\rangle. \quad (5.50)$$

Matice **C** je shodná s maticí bilineární Coonsovy plochy (5.8.1). Funkce $F_1(u)$, $F_2(u)$, $F_1(v)$, $F_2(v)$ jsou Hermitovské polynomy

$$F_1(t) = 2t^3 - 3t^2 + 1 \qquad F_2(t) = -2t^3 + 3t^2.$$

Explicitní vyjádření, které získáme po úpravě (5.50), má tvar

$$\begin{aligned} Q(u,v) &= [F_1(u),F_2(u)]\begin{bmatrix} P(0,v) \\ P(1,v) \end{bmatrix} + [P(u,0),P(u,1)]\begin{bmatrix} F_1(v) \\ F_2(v) \end{bmatrix} - \\ &\quad -[F_1(u),F_2(u)]\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}\begin{bmatrix} F_1(v) \\ F_2(v) \end{bmatrix}. \end{aligned} \tag{5.51}$$

Pro tytéž důvody, které jsou uvedeny u bilineární plochy, se této plochy nepoužívá pro plátování se spojitou první derivací C^1, G^1.

5.8.3 Obecná bikubická plocha

Obecná bikubická plocha se hodí lépe pro navazování a umožňuje splnit běžně požadované vlastnosti při plátování. Její předností je, že umožňuje hladké napojování plátů, a hodí se tedy pro generování tvarově složitých ploch. Nevýhodou je, že je potřeba zadávat i tečné vektory podle okraje.

Obecná bikubická plocha je zadána svým okrajem $P(u,0)$, $P(u,1)$, $P(0,v)$ a $P(1,v)$. Navíc je určena průběhy příčných vektorů podél okrajů $\vec{p}_v(u,0), \vec{p}_v(u,1), \vec{p}_u(0,v), \vec{p}_u(1,v)$. Dále je pak určena čtyřmi zkruty v rozích: $\vec{t}(0,0), \vec{t}(0,1), \vec{t}(1,0), \vec{t}(1,1)$ Pro vygenerování plochy jsou zapotřebí ještě údaje o hodnotách tečných příčných vektorů v bodech okraje (ty se získají snadno dosazením $u=0, u=1, v=0, v=1$ do $\vec{p}_v(u,0)$, $\vec{p}_v(u,1)$, $\vec{p}_u(0,v)$ a $\vec{p}_u(1,v)$) a hodnoty bodů v rozích, jež se získají dosazením do funkcí popisujících okraj. Nyní známe všechny údaje pro vygenerování Coonsovy obecné bikubické plochy:

$$[F_3(u),F_1(u),-1,F_2(u),F_4(u)].\mathbf{C}.[F_3(v),F_1(v),-1,F_2(v),F_4(v)]^T = 0. \tag{5.52}$$

Matice $\mathbf{C}$ je mapou plochy a má tvar:

$$\mathbf{C} = \begin{pmatrix} \vec{t}(0,0) & \vec{p}_u(0,0) & \vec{p}_u(0,v) & \vec{p}_u(0,1) & \vec{t}(0,1) \\ \vec{p}_v(0,0) & P_{00} & P_{0v} & P_{01} & \vec{p}_v(0,1) \\ \vec{p}_v(u,0) & P_{u0} & Q(u,v) & P_{u1} & \vec{p}_v(u,1) \\ \vec{p}_v(1,0) & P_{10} & P_{1v} & P_{11} & \vec{p}_v(1,1) \\ \vec{t}(1,0) & \vec{p}_u(1,0) & \vec{p}_u(1,v) & \vec{p}_u(1,1) & \vec{t}(1,1) \end{pmatrix} \sim \begin{pmatrix} \nwarrow & \downarrow & \downarrow & \downarrow & \nearrow \\ \rightarrow & \bullet & \bullet & \bullet & \rightarrow \\ \rightarrow & \bullet & \bullet & \bullet & \rightarrow \\ \rightarrow & \bullet & \bullet & \bullet & \rightarrow \\ \swarrow & \downarrow & \downarrow & \downarrow & \searrow \end{pmatrix}$$

$F_i(u), F_i(v), i = 1,\ldots,4$ jsou Hermitovské polynomy třetího stupně. Obecná bikubická plocha prochází zadaným okrajem. Zderivováním (5.52) a dosazením za $u,v = 0,1$ zjistíme, že tečné vektory výsledné plochy odpovídají zadaným příčným tečným vektorům. Podobně dosazením do podruhé zderivovaného vztahu zjistíme, že výsledná plocha má odpovídající zkruty v rozích.

Největší předností je hladké spojení dvou obecných bikubických Coonsových ploch, jež je zaručeno identitou okrajových křivek v místě navázání a identitou tečných vektorů tamtéž.

Je zřejmé, že plochy zadané svým okrajem nejsou nejlépe zadavatelné, neboť vyžadují po uživateli znalost průběhu okrajových křivek. Protože se ale v praxi pro zadávání okrajů obvykle využívají snadno modelovatelné Bézierovy nebo Hermitovské křivky, je možné většinu požadovaných podmínek zajistit automaticky a poskytnout uživateli jednoduché nástroje pro tvorbu těchto ploch.

5.9 Bézierovy plochy

Některé modelovací systémy používají k reprezentaci povrchů těles Bézierovy plochy. Tyto plochy jsou snadno diferencovatelné, jednoduše a intuitivně se modelují a relativně snadno se vypočítává průsečík s paprskem. I když se v současných systémech používají pro uchování ploch složitější reprezentace založené na technologii NURBS (část 5.10), tradiční uživatelský pohled a možnost práce s Bézierovými modelovacími nástroji jsou běžnou součástí těchto systémů. Je to umožněno především díky tomu, že Bézierovy plochy jsou speciálním případem ploch NURBS.

Bézierova plocha $n \times m$-tého stupně je určena $(n+1) \times (m+1)$ řídicími body $P_{i,j}$ a vztahem

$$Q(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{i,j} B_i^n(u) B_j^m(v). \tag{5.53}$$

Bázové funkce B_i^k, $k = n, m$ jsou *Bernsteinovy polynomy* k-tého stupně, které jsou použity pro definici Bézierových křivek (vztah 5.24, část 5). Průběhy jsou na obrázku 5.14.

Dosadíme-li do vztahu (5.53) za u a v postupně nulu a jedničku, přesvědčíme se, že strany Bézierovy plochy tvoří Bézierovy křivky (5.15) z odstavce 5.4.1. Pro hodnoty $Q(0,0)$, $Q(1,0)$, $Q(0,1)$ a $Q(1,1)$ jsou rohy Bézierovy plochy totožné s body P_{00}, P_{10}, P_{01} a P_{11} řídicí sítě. Bézierova plocha tedy těmito body prochází.

Dosazením za $u = 0$ a $v = 0$ do derivace vztahu (5.53) podle u a podle v dostaneme výrazy pro tečné vektory v rohu $Q(0,0)$ (podobně v ostatních rozích):

$$\begin{aligned} \vec{q}_u(0,0) &= n.(P_{10} - P_{00}) \\ \vec{q}_v(0,0) &= m.(P_{01} - P_{00}). \end{aligned} \tag{5.54}$$

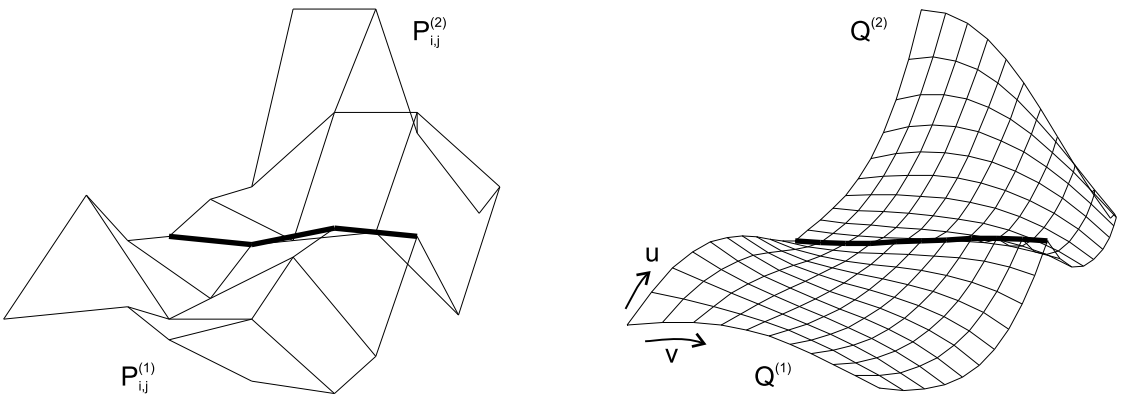
Normálu plochy $\vec{n}$ v bodě $Q(u,v)$ získáme ze vztahu (5.40) jako vektorový součin tečných vektorů v tomto bodě.

Bernsteinovy polynomy jsou nezáporné a jejich součet je roven jedné. Z těchto dvou vlastností lze dokázat [Fari93], že Bézierova plocha leží celá v konvexní obálce svých řídicích bodů.

Další vlastností Bézierovy plochy je, že změní celý svůj tvar při změně polohy jediného řídicího bodu. Tato vlastnost je nevýhodná a je jedním z důvodů, proč se Bézierovy plochy plátují. Při změně polohy jediného řídicího bodu se pak změní tvar pouze jediného plátu (není–li tento bod z důvodů spojitosti svázán s body ze sousedního plátu).

Navazování Bézierových plátů

Mějme dva Bézierovy pláty $Q^{(1)}(u,v)$ a $Q^{(2)}(u,v)$. První z nich je určen sítí řídicích bodů $P^{(1)}_{i,j}$, $i = 0,\ldots,s$, $j = 0,\ldots,m$ a druhý je určen jako $P^{(2)}_{i,j}$, $i = 0,\ldots,t$, $j = 0,\ldots,m$. Počet bodů ve směru v je stejný pro oba pláty a je roven m. Pláty budeme navazovat ve směru u a požadujeme, aby jejich stupeň v tomto směru byl alespoň tři, tj. $s \geq 3$ a $t \geq 3$.



Obrázek 5.33: C^0 napojení dvou Bézierových plátů $Q^{(1)}$ a $Q^{(2)}$. Silně zvýrazněná lomená čára v řídicí síti (vlevo) spojuje řídicí body Bézierovy kubiky zvýrazněné na druhém obrázku, která tvoří společnou stranu obou plátů.

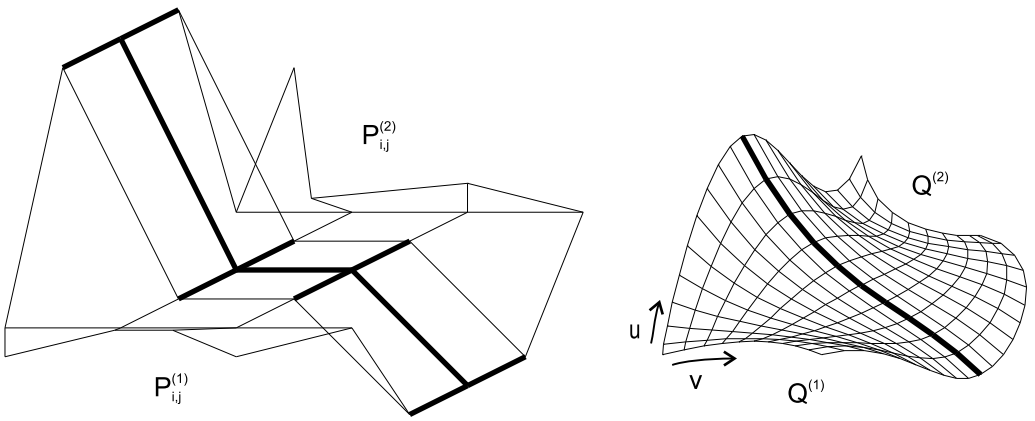
Pláty $Q^{(1)}$ a $Q^{(2)}$ jsou C^0 spojitě navázány ve směru u (viz obrázek 5.33), pokud je totožná jejich (alespoň C^0 spojitá) strana, tj. $Q^{(1)}(1,v) = Q^{(2)}(0,v)$. Této spojitosti Bézierových plátů docílíme ztotožněním řídicích bodů, které určují příslušnou stranu

$$P^{(1)}_{s,j} = P^{(2)}_{0,j};\ j = 0,\ldots,m. \tag{5.55}$$

Tento způsob spojení umožňuje vznik ostré hrany (viz obrázek 5.33 vpravo). Pláty $Q^{(1)}$ a $Q^{(2)}$ mají napojení C^1 (viz obrázek 5.26), pokud je jejich společná hrana C^1 spojitá a jsou-li identické

příčné tečné vektory ve směru u podél této strany. C^1 napojení dvou plátů docílíme shodou C^1 spojité strany a následujícím vztahem pro body řídicích sítí obou plátů

$$s.(P^{(1)}_{s,j} - P^{(1)}_{s-1,j}) = t.(P^{(2)}_{1,j} - P^{(2)}_{0,j});\ j = 0, \ldots, m. \tag{5.56}$$



Obrázek 5.34: C^1 spojení dvou Bézierových plátů

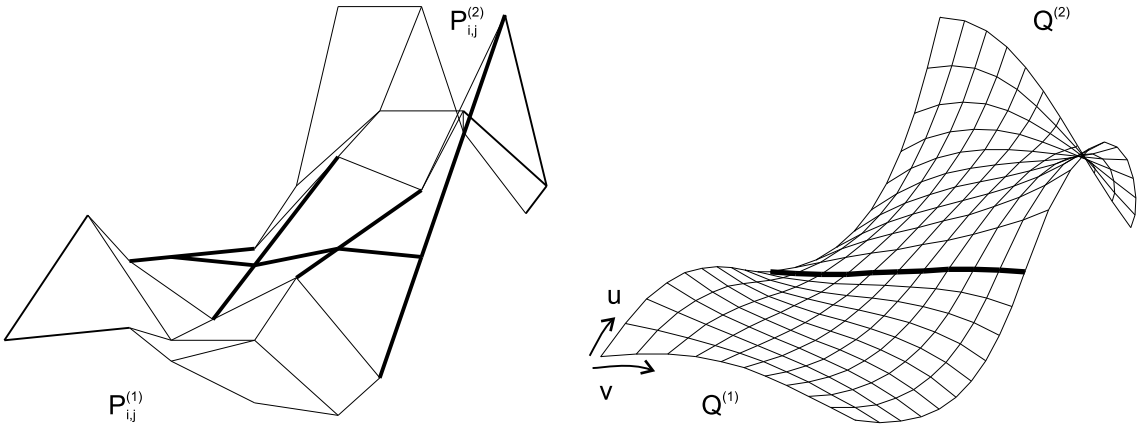
Z vlastnosti C^1 napojení dvou plátů plyne [Fari93], že hlavní křivky obou plátů ve směru napojení jsou C^1 křivkami.

Pláty $Q^{(1)}$ a $Q^{(2)}$ mají napojení G^1 (viz obrázek 5.35), pokud je jejich společná strana křivka třídy G^1 a příčné tečné vektory ve směru u jsou lineárně závislé s koeficientem $k > 0$ spojitě se měnícím podél společné strany. Toho dosáhneme společnou stranou, která je křivkou třídy alespoň G^1 a následujícím vztahem pro body řídicích sítí obou plátů

$$P^{(1)}_{s,j} - P^{(1)}_{s-1,j} = k.(P^{(2)}_{1,j} - P^{(2)}_{0,j});\ j = 0, \ldots, m;\ k > 0. \tag{5.57}$$

Tato podmínka je méně omezující nežli C^1 spojitost, neboť umožňuje větší manipulaci s takto svázanými řídicími body (viz dále). Pokud položíme $k = 0$ přechází ve spojitost pouze C^0 a je-li společná strana třídy C^1, tak přechází tato spojitost při hodnotě $k = 1$ ve spojitost C^1.

Požadavek hladkého navázání spolu svazuje body sousedních plátů a snižuje tak možnosti jejich editace. Svázání několika bodů podmínkami (5.56), resp. (5.57) určuje polohu vždy skupiny řídicích bodů. Nejvýrazněji tato podmínka omezuje společný roh čtyř spojených plátů, kde je takto svázáno celkem devět řídicích bodů. V některých případech se proto podmínky



Obrázek 5.35: G^1 napojení dvou Bézierových plátů. Řídicí body určující společnou G^1 spojitou stranu obou ploch, spolu s vždy předposledním bodem z plátu $Q^{(1)}$ a druhým bodem plátu $Q^{(2)}$ ve směru napojení u leží na přímce

(5.56) a (5.57) místo na všechny řídicí body společné strany omezují jen na první a poslední řídicí bod (rohy) této strany. Tím se sníží počet takto svázaných řídicích bodů na pět. Požadavek spojitosti C^0 (5.55), (tj. identita řídicích bodů určujících společnou stranu) trvá. Detailní popis metody s odvozením patřičných vztahů je k nalezení ve [Watt92].

Bézierovy bikubické plochy

Speciálním a nejčastěji používaným případem Bézierových ploch jsou Bézierovy bikubické plochy. Ty obdržíme po dosazení za $n = m = 3$ do (5.53), tj.

$$Q(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} P_{i,j}B_i(u)B_j(v), \tag{5.58}$$

Bernsteinovy polynomy třetího stupně jsou definovány soustavou 5.24 uvedenou v části 5 a jsou znázorněny na obrázku (5.14). Tečný vektor $\vec{q}_u$ ve směru u Bézierovy plochy $Q(u,v)$ v obecném bodě získáme derivací vztahu pro vektor $\mathbf{U}$

$$\mathbf{U}_u = \frac{\partial \mathbf{U}}{\partial u} = \frac{\partial}{\partial u}\begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} = \begin{bmatrix} 3u^2 & 2u & 1 & 0 \end{bmatrix} \tag{5.59}$$

a dosazením do (5.58)

$$\vec{q}_u = \mathbf{U}_u\ \mathbf{M}_B\mathbf{P}\mathbf{M}_B^T\ \mathbf{V}^T.$$

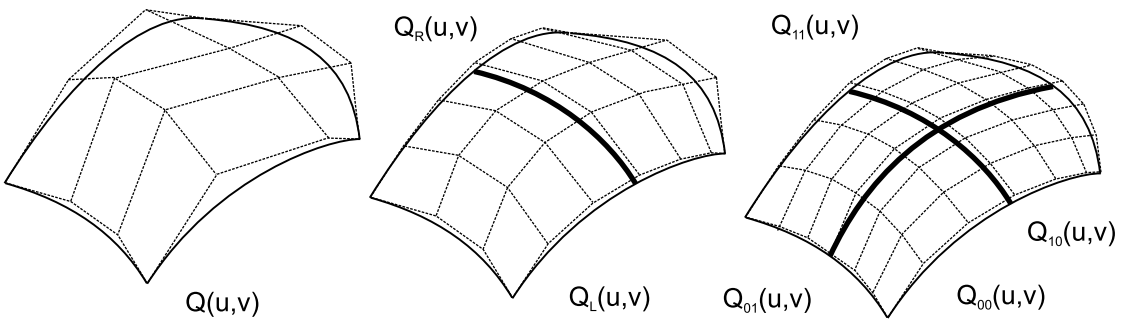
Analogicky bychom postupovali při získání tečny $\vec{q}_v$ ve směru v. Normála k Bézierově bikubickému plátu se určí ze vztahu (5.40).

Převod Bézierových bikubických ploch na síť trojúhelníků

Naivní a jednoduchou metodou zobrazování Bézierových ploch je jejich polygonizace postupným dosazováním do vztahu (5.53) za u a v s pevným krokem Δu a Δv. V každém kroku tímto způsobem získáme čtverec určený rohy Bézierovy plochy, který můžeme rozdělit na dva trojúhelníky. Normálové vektory ve vrcholech trojúhelníků získáme ze vztahu (5.40). Nevýhodou tohoto postupu, podobně jako u převodu křivek na úsečky je, že nerespektuje zakřivení plochy (viz obrázek 6.1).

Adaptivní metoda polygonizace Bézierovy plochy pracuje na principu *rekurzivního dělení plátu* (*patch splitting*). Tato metoda využívá *algoritmus de Casteljau*, který je popsán na straně 187.

Dělení plátu (viz obrázek 5.36) pracuje následovně. Vstupem algoritmu 5.2 je matice řídicích bodů (pro případ bikubického plátu je jich 16). V prvním kroku rozdělíme plochu ve směru u tak, že algoritmem de Casteljau pro dělení křivky v bodě $t = 1/2$ (viz obr. 5.11) rozdělíme křivky, určené řídicími body v jednotlivých řádcích matice $\mathbf{P}$, na dvě části (obrázek 5.36 uprostřed) a tímto dělením získáme pro každý řádek dvě skupiny nových řídicích bodů. Tím jsme rozdělili plochu reprezentovanou řídicími body z matice $\mathbf{P}$ na dvě plochy Q_L a Q_R, které jsou reprezentovány maticemi $\mathbf{P}_L$ a $\mathbf{P}_R$. Ve druhém kroku aplikujeme předcházející postup na sloupce matic $\mathbf{P}_L$ a $\mathbf{P}_R$, čímž získáme celkem čtyři matice $\mathbf{P}_{00}$, $\mathbf{P}_{01}$, $\mathbf{P}_{10}$ a $\mathbf{P}_{11}$, obsahující celkem 4×16 bodů a odpovídající plochy Q_{00}, Q_{01}, Q_{10} a Q_{11} (obrázek 5.36 vpravo).



Obrázek 5.36: Rekurzivní dělení Bézierovy bikubické plochy

Po n dělení získáme z původní matice řídicích bodů 4^n matic, které reprezentují stejnou plochu složenou z částí. Řídicí sítě uložené v maticích představují aproximaci původní plochy. Tohoto faktu využijeme při jejím zobrazení a každý ze čtveřice sousedních bodů rozdělíme

DěleníPlátu(P)
Vstup: P – matice 4×4 řídicích bodů

1. je-li odchylka řídicích bodů od roviny dostatečně malá, rozděl čtyřúhelníky na trojúhelníky a skonči
2. rozděl matici P ve směru parametru u (výsledkem jsou dvě matice P_L a P_R)
3. rozděl matici P_L ve směru parametru v (výsledkem jsou dvě matice P_{00} a P_{01})
4. rozděl matici P_R ve směru parametru v (výsledkem jsou dvě matice P_{10} a P_{11})
5. DěleníPlátu(P_{00}); DěleníPlátu(P_{01})
6. DěleníPlátu(P_{10}); DěleníPlátu(P_{11})

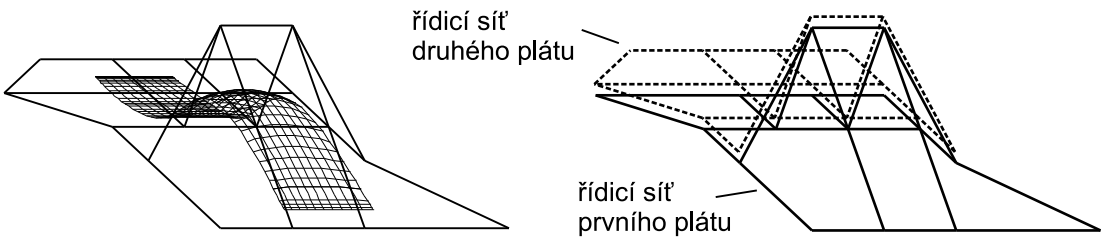
Algoritmus 5.2: Rekurzivní algoritmus rozdělení Bézierovy bikubické plochy

na dva trojúhelníky tak, jak je uvedeno na obrázku 5.36 vpravo dole. Pro další operace s těmito trojúhelníky je vhodné ještě znát hodnoty normálových vektorů v jejich rozích. Ty se stanoví ze vztahu (5.40).

Zbývá určit kritéria zastavení dělení plochy. Zjevně nejjednodušší je určit pevný počet dělení n a tím v podstatě stanovit i počet vygenerovaných trojúhelníků na hodnotu 2.4^n. Nevýhodou tohoto postupu je, že aproximujeme rovné části plochy stejným množstvím trojúhelníků jako části silně zaoblené. Tím ale nedocílíme žádného zlepšení oproti metodě s pevným krokem uvedené na začátku tohoto odstavce. Efektivnější je adaptivní dělení, končící při podmínce, která vychází z geometrie řídicí sítě. Takovou podmínkou může být například hodnota maximální odchylky bodů sítě od roviny určené třemi nejvzdálenějšími body řídicí sítě, nebo jsou-li hrany „dostatečně“ rovné aj.

5.10 B-spline plochy

B-spline plochy, které jsou zobecněním B-spline křivek z odstavce 5.4.5, se velice snadno navazují a jsou proto pro modelování daleko výhodnější, než Hermitovské a Bézierovy pláty. B-spline plochy n-tého stupně zaručují C^{n-1} spojitost ve všech svých bodech a není nutné omezovat některé jejich řídicí body vnějšími podmínkami jako v případě Bézierových ploch. Změnou jediného řídicího bodu měníme tvar vždy pouze části B-spline plochy. B-spline plochy se zadávají sítí řídicích bodů. Na rozdíl od Bézierových plátů se však navazující B-spline plát definuje použitím $m \times (n-1)$ bodů plátu předchozího a přidáním pouze m dalších bodů (viz obrázek 5.37).



Obrázek 5.37: Navázání dvou B-spline plátů – shodné části řídicích sítí

Tímto způsobem tedy dochází k překrytí reprezentací plátů. Část reprezentace jednoho plátu je součástí plátu následujícího, v případě kubik se jedná o tři sloupce nebo tři řádky v mapě plochy. Změnou polohy jediného řídicího bodu tak manipulujeme s více pláty, počet ovlivněných plátů závisí na zvoleném stupni bázových polynomů obdobně jako při modelování B-spline křivek. To však není nijak na obtíž. Podstatné je, že tato změna je lokální a nezpůsobuje změnu tvaru celé plochy.

Mezi vlastnosti B-spline ploch patří:

- plocha leží celá v konvexní obálce svých řídicích bodů,
- při změně polohy jediného řídicího bodu změní tvar pouze ty pláty, které jsou tímto bodem určeny,
- plocha obecně neprochází krajními body řídicí sítě, toho lze docílit násobností řídicích bodů a u neuniformních ploch násobností uzlů (viz dále),
- jsou invariantní k lineárním transformacím (otáčení, posunu, změně měřítka, zkosení).

Bikubické B-spline plochy

Nejjednodušší z B-spline ploch jsou bikubické B-spline plochy (jedná se o neracionální a neuniformní B-spline plochy). Tyto plochy se používají ve složitějších modelovacích programech, avšak jedinou jejich výhodou oproti Bézierových plochám je, že jsou spojité C^2 bez nutnosti zadávat nějaké vnější omezující podmínky na polohu řídicích bodů. Tyto plochy nejsou invariantní k perspektivnímu promítání.

Bikubická B-spline plocha (Coonsův plát) je určena maticovým zápisem

$$Q(t) = \frac{1}{36}\mathbf{U}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}\begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}\mathbf{V}^T \quad (5.60)$$

nebo zápisem

$$Q(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} P_{i,j}B_i(u)B_j(v), \tag{5.61}$$

kde bázové funkce jsou totožné s bázovými polynomy $BS_0(t)$, $BS_1(t)$, $BS_2(t)$ a $BS_3(t)$ Coonsových kubik v části 5 s průběhy na obrázku 5.17.

NURBS plochy

Neuniformní racionální B-spline plochy – NURBS (*non uniform rational B-spline*) jsou zobecněním B-spline ploch a představují dnes průmyslový standard v geometrickém modelování. NURBS umožňují definovat širokou třídu ploch, mezi něž patří jak volně tvarovatelné plochy na bázi racionálních polynomů (*free form surfaces*), ale i plochy založené na přímkách, kuželosečkách apod. I tyto tvary se dají vyjádřit v NURBS reprezentaci, jak je ukázáno v části 5.4. Uživatel pracuje s těmito objekty v jednotné reprezentaci. Výhodné je, že z hlediska implementace se používají jednotné datové struktury a stejně tak i algoritmy.

NURBS plochou rozumíme

$$Q(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} w_{i,j}P_{i,j}N_{i,p}(u)N_{j,q}(v)}{\sum_{i=0}^{n}\sum_{j=0}^{m} w_{i,j}N_{i,p}(u)N_{j,q}(v)}, \tag{5.62}$$

kde $w_{i,j}$ jsou váhy bodů (homogenní souřadnice) $P_{i,j}$ řídicí sítě $\mathbf{P}$, n a m je počet řídicích bodů, p a q jsou stupně polynomů a konečně $N_{i,p}(u)$,$N_{j,q}(v)$ jsou *normalizované B-spline bázové funkce* určené rekurentním vztahem (5.30) uvedeným v části 5.

NURBS plocha je dále určena dvěma uzlovými vektory – vektorem $\mathbf{U}$ délky $n+p+1$ a $\mathbf{V}$ délky $m+q+1$, kde n, resp. m je počet řídicích bodů ve směru u, resp. v a p, resp. q je stupeň plochy ve směru u, resp. v. Uzlové vektory ovlivňují průběhy jednotlivých bázových funkcí a interval jejich vlivu. Vlastnosti bázových funkcí jsou uvedeny v části 5.

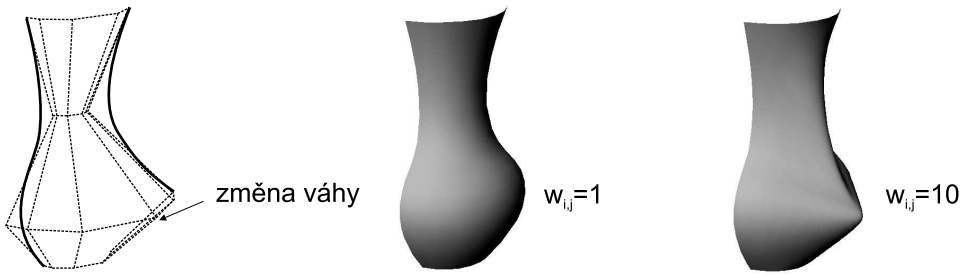
Obdobně jako u NURBS křivek i u ploch využíváme zjednodušený zápis pomocí *racionální B-spline báze*:

$$R_{i,p}(t) = \frac{w_i N_{i,p}(t)}{\sum_{j=0}^{n} w_j N_{j,p}(t)}. \tag{5.63}$$

Rovnici (5.62) lze potom zapsat jednodušeji:

$$Q(t) = \sum_{i=0}^{n}\sum_{i=0}^{m} P_{i,j}R_{i,p}(u)R_{j,q}(v). \tag{5.64}$$

Váhy $w_{i,j}$ určují vliv bodu na plochu. Pokud je váha rovna nule, nemá bod na plochu žádný vliv, s rostoucí váhou se plocha k bodu přimyká a pro hodnotu $w_{i,j} \to \infty$ plocha bodem prochází. Příklad NURBS plochy a její změny podle váhy jednoho řídicího bodu, je na obrázku 5.38.



Obrázek 5.38: Změna tvaru NURBS plochy po změně váhy jednoho řídicího bodu

NURBS jsou stejně jako Bézierovy plochy invariantní k lineárním transformacím a díky racionalitě navíc i k perspektivní projekci. Jinými slovy řečeno, plochu, jejíž každý bod zobrazujeme v perspektivní projekci (což je výpočetně velice náročné), můžeme získat tak, že nejprve podrobíme perspektivní projekci její řídicí body a potom vygenerujeme plochu bez nutnosti dále transformovat její body.

Zobrazování NURBS ploch. Podobně jako v případě Bézierových ploch, je nejjednodušším zobrazením ploch NURBS jejich polygonizace postupným dosazováním do (5.62) za u a v s proměnným parametrem Δu a Δv. Tímto způsobem získáme síť trojúhelníků, kterou můžeme zobrazovat běžnými metodami. Jiné metody používají adaptivní dělení plochy, podobně jako v případě plochy Bézierovy, případně numerické metody. Další informace je možno nalézt v [Fari93, Pieg95].

5.11 Šablonování

V tomto odstavci uvedeme poměrně silnou modelovací techniku zvanou šablonování (*sweeping*). Budeme sice hovořit o NURBS plochách, ale protože tyto postupy definují vždy řídicí síť plochy, je možné většinu těchto konstrukcí samozřejmě použít i pro Bézierovy či neuniformní B-spline plochy.

Šablonování je modelovací technika, při které získáváme plochu tažením dvourozměrného obrysu (tzv. profilu) po trojrozměrné křivce – tzv. *páteři* (*spine curve*). Je možné uvážit i šablonování trojrozměrných objektů, ale tímto způsobem získaná plocha je komplikovaná a navíc obtížně předvídatelná. Spíše nežli v modelování, se tato metoda používá k odhadu kolizí v robotice, či virtuální realitě. V [Bron90] se šablonování rozděluje do tří kategorií.

- *translační šablonování*: obrys je libovolný, páteř je úsečka. Tímto způsobem lze získat přímkovou plochu (viz odstavec 5.11.1).

- *rotační šablonování*: obrys je libovolný, trajektorie, po které je obrys tažen, je kružnice nebo její část – toto těleso lze též získat rotací kolem osy (viz odstavec 5.11.2). Výsledkem této operace je rotační plocha (*surface of revolution*).
- *obecné šablonování*: obrys i trajektorie je libovolná. Touto operací získáme zobecněnou válcovou plochu (*generalized cylinder*).

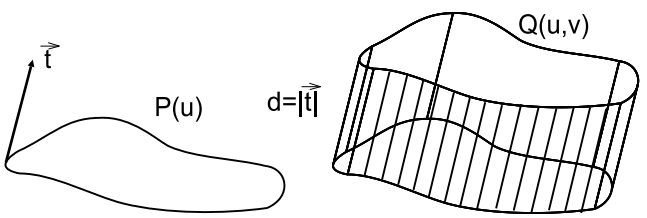
Jiné rozdělení šablonování je možné na základě toho, zda profilová křivka při pohybu mění svůj tvar nebo zda její tvar zůstává konstantní.

5.11.1 Přímkové plochy

Přímkové plochy jsme již jednou uvedli v části věnované plochám, které propojují dvě okrajové křivky. Zde se k přímkovým plochám vracíme ve spojitosti se šablonováním a možností jejich jednotné reprezentace pomocí NURBS. Přímkové plochy se v jednom ze směrů skládají z úseček. Příkladem takovéto plochy je například válcová plocha, hranolová plocha, nebo plocha uvedená na obrázku 5.30. Uvedeme si dva způsoby, jak získat přímkovou plochu – vytažením nebo spojením dvou okrajů. V obou případech je možno tyto operace chápat jako translační šablonování po úsečce; poprvé s profilovou křivkou, která nemění svůj tvar, podruhé s měnící se profilovou křivkou. Kombinace obou přístupů, tedy translační šablonování po obecné křivce s měnící se profilovou křivkou se nazývá *potahování* (*skinning*).

Translační šablonování po úsečce s neměnící se profilovou křivkou

Translačním šablonováním po úsečce získáme přímkovou plochu. Vytažená plocha (*extruded surface*) se získá z křivky $P(u)$ vytažením ve směru $\vec{t}$ o vzdálenost $d = |\vec{t}|$ (obrázek 5.39).



Obrázek 5.39: Plocha (vpravo) získaná vytažením z profilové křivky $P(u)$ (vlevo) ve směru vektoru $\vec{t}$ o vzdálenost $d = |\vec{t}|$

Plocha $Q(u, v)$ je popsána vztahem $Q(u, v) = P(u) + v.\vec{t}$. Pokud je $P(u)$ křivka NURBS stupně p určená řídicími body P_i, $i = 0, \ldots, n$ a uzlovým vektorem $\mathbf{U}$, můžeme použít NURBS reprezentaci i pro výslednou plochu. Úsečka je totiž NURBS křivka se dvěma řídicími (okrajovými)

body a uzlovým vektorem $\mathbf{V} = \{0, 0, 1, 1\}$. Translační plochu zkonstrujeme tak, že nejprve vytvoříme řídicí body dvou okrajových křivek, původní a posunuté o translační vektor, a získáme tak sadu řídicích bodů $Q_{i,0} = P_i$ a $Q_{i,1} = P_i + \vec{t}$. Výsledná plocha je pak definována těmito řídicími body a uzlovými vektory $\mathbf{U}$ (původní křivka) a $\mathbf{V} = \{0, 0, 1, 1\}$ (úsečka ve směru $\vec{t}$). Rovnice této plochy je

$$Q(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{1} P_{i,j} R_{i,p}(u) R_{j,2}(v). \tag{5.65}$$

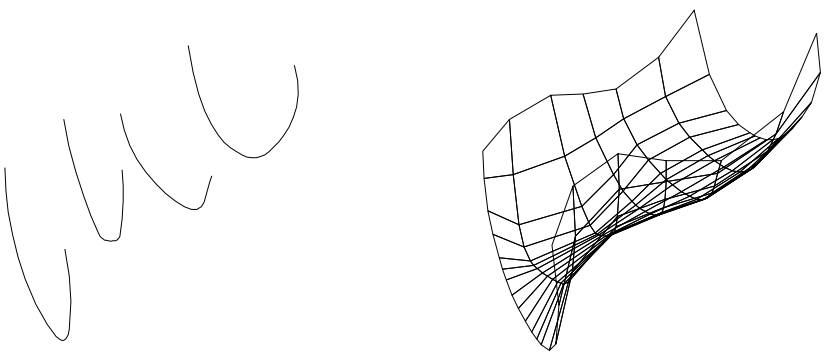
Posunutím kružnice lze vytvořit válcovou plochu, posunem úsečky obdélník, posunem čtverce hranolovou plochu, atp. Výsledkem je tedy vždy plocha, jejíž profil v řezu kolmém na trajektorii se nemění.

Translační šablonování po úsečce mezi danými profily

Tento druh šablonování propojuje dva profily (okrajové křivky) úsečkami. Přímková plocha (*ruled surface*), kterou takto získáme, vznikne spojením dvou křivek $P_0(u)$ a $P_1(u)$ úsečkami (viz obrázek 5.30) a je popsána vztahem

$$Q(u, v) = (1 - v).P_0(u) + v.P_1(u)$$

Pro zápis v reprezentaci NURBS předpokládejme, že NURBS křivky $P_0(u)$ a $P_1(u)$ jsou zadány stejným počtem řídicích bodů a shodným uzlovým vektorem $\mathbf{U}$ (jsou i stejného stupně p). První z křivek je určena řídicími body $P_{i,0}$ a druhá body $P_{i,1}$. Požadovaná plocha je potom určena body $P_{i,j}$, $i = 0, \ldots, n; j = 0, 1$, uzlovým vektorem $\mathbf{U}$ a uzlovým vektorem $\mathbf{V} = \{0, 0, 1, 1\}$ a rovnicí (5.65). Tato plocha ve svých dvou okrajích interpoluje definující křivky $Q_0(u)$ a $Q_1(u)$.

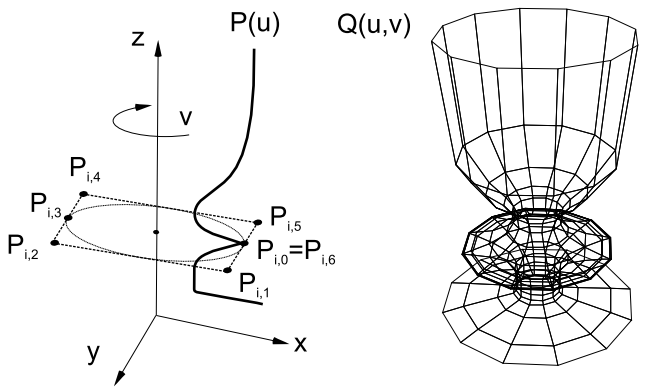


Obrázek 5.40: NURBS plocha (vpravo) vzniklá interpolací profilových křivek (vlevo)

Potahování

V úvodu této kapitoly jsme poznamenali, že interpolace ve třech rozměrech hladkou plochou je poměrně složitá úloha a funguje spolehlivě pouze v některých případech. Jedním z těchto případů, je tzv. potahování (*skinning*)[1] [Pieg91, Pieg95, Watt92]. Potahování není interpolací izolovaných bodů v prostoru, ale interpolací křivek viz obrázek 5.40.

Mějme několik příčných řezů (*cross section*), které má NURBS plocha interpolovat. V praxi jsou tyto křivky obyčejně rovinné. Příkladem takového zadání je trup lodi, kde jednotlivá žebra tvoří profilové křivky a dno lodi je páteří. Jiným příkladem jsou příčné řezy křídlem letadla. Naším cílem je tato žebra pokrýt plochou. K tomu je zapotřebí, aby všechny příčné řezy byly kompatibilní, to jest, aby se jednalo o křivky stejného stupně, aby byly určeny stejným počtem řídicích bodů, a aby měly stejný uzlový vektor (například ve směru parametru u). Jsou-li tyto požadavky splněny, vypočítáme uzlový vektor $\mathbf{V}$ ve směru potahování. Ten získáme tak, že pevně zvolíme v každém příčném řezu jeden řídicí bod (například první) a tyto body interpolujeme křivkou (algoritmus lze nalézt v [Pieg91]). Řídicí body příčných řezů spolu s uzlovými vektory $\mathbf{U}$ a $\mathbf{V}$ pak tvoří definici NURBS plochy, která dané příčné řezy interpoluje.



Obrázek 5.41: NURBS plocha (vpravo) získaná rotováním rovinné křivky (vlevo)

5.11.2 Rotační šablonování

Rotační plochu získáme z rovinné profilové křivky (obrázek 5.41) rotací okolo osy. Řezem rotační plochy libovolnou rovinou kolmou na osu rotace je tedy kružnice. Při modelování rotační plochy můžeme použít jako profil libovolné rovinné křivky v různé reprezentaci.

[1] Někdy se používá termín *lofting*. Tento termín pochází ze slangu loďařů, kteří tohoto postupu používali při stavbě trupů lodí.

Jako příklad uvedeme jednotný přístup při tvorbě rotačních ploch pomocí NURBS. Připomeňme, že v technologii NURBS máme k dispozici široký repertoár základních tvarů, od úseček a lomených čar, přes Hermitovské a Bézierovy křivky až po neuniformní neracionální B-spline křivky. Uvažujme například NURBS křivku $P(v)$ stupně k v rovině xz zadanou body P_i a uzlovým vektorem $\mathbf{V}$. Tuto křivku podrobíme otáčení kolem osy z. Řídicí síť výsledné rotační plochy získáme tak, že každý řídicí bod $P_i = [x_i, y_i, z_i, w_i]$ profilové křivky okopírujeme v rovině xy ve výšce z_i sedmkrát tak, aby vzniklé řídicí body tvořily v rovině xy řídicí polygon kružnice podle obrázku 5.23.
Váhy $w_{i,j}$ nově vzniklých bodů se vypočítají z váhy w_i bodu T_i takto

$$w_{i,j} = \{w_i, w_i/2, w_i/2, w_i, w_i/2, w_i/2, w_i\};\ j = 0, \ldots, 6.$$

Rovnice rotační plochy má tvar

$$Q(u,v) = \sum_{j=0}^{n} \sum_{i=0}^{6} P_{i,j} R_{i,3}(u) R_{j,k}(v) \tag{5.66}$$

s uzlovými vektory $\mathbf{U} = \{0, 0, 0, 1/4, 1/2, 1/2, 3/4, 1, 1, 1\}$ (kružnice) a $\mathbf{V}$ (určen profilovou křivkou). Bude-li například profilovou křivkou kružnice, která je posunutá o určitou vzdálenost od osy z, získáme její rotací kolem osy z anuloid (prstenec). Rotací úsečky rovnoběžné s osou z, která s touto osou nesplývá, lze získat válcovou plochu, rotací úsečky různoběžné s osou z lze získat otevřený kužel, rotací kružnice v základní poloze o 180 stupňů kouli, atp.

5.12 Implicitní plochy

Přestože geometrické modely objektů používaných v počítačové grafice jsou založeny především na parametrickém vyjádření (viz odstavec 5.5), praktické aplikace používají i modelování, založené na implicitním vyjádření objektů.

Myšlenka vyjádřit těleso pomocí implicitní funkce není nijak nová. Blinn v roce 1982 [Blin82a] navrhl chápat izoplochy vzniklé při modelovaní elektrického potenciálu elementárních částic jako objekty. Postupem času byla tato technika rozšiřována a tyto objekty dostávaly rozličné názvy (*blobby objects* [Blin82a], *Soft objects* [Wyvi86], *meta balls*) podle toho, jaké směšovací funkce (viz dále) v nich byly použity. J. Bloomenthal upozornil [Bloo88], že všechny tyto objekty patří do stejné kategorie a mohou proto být označeny jediným názvem *implicitní plochy* (*implicit surfaces*). V této knize budeme používat termín implicitní plocha, či implicitní těleso.

Implicitní plocha je množina bodů P ve třírozměrném prostoru, pro které platí

$$Q(P) = konst.$$

Příkladem takovéto množiny je kulová plocha, která je zadána implicitní rovnicí

$$Q(P) = x^2 + y^2 + z^2 = r^2.$$

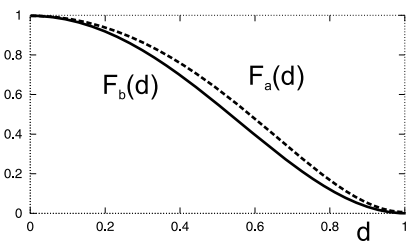
Volbou konstanty r^2 volíme v prostoru jednotlivé izoplochy, tj. místa, ve kterých nabývá implicitní funkce zvolené konstantní hodnoty. Modelovací možnosti jednoduchých implicitních ploch jsou omezené, jejich kombinací však lze modelovat tvarově bohaté objekty. Základem modelování je tzv. *kostra* (skeleton). Kostra se skládá z *generátorů, prvků kostry.* Pro všechny body P v „dosahu" generátoru je definována funkce $d = d(P)$, která určuje vzdálenost bodu P od generátoru. Dále je pro každý generátor definována *potenciálová funkce* $F(d)$, která určuje vliv generátoru na místa, která jsou ve shodné vzdálenosti d. Modelovaní pomocí implicitních objektů spočívá v tom, že používáme jednoduché prvky kostry (body, úsečky, polygony, části křivek), vyhodnotíme jejich potenciálové funkce a výsledná izoplocha pak vznikne kombinací dílčích příspěvků od generátorů. Příkladem jednoduché potenciálové funkce je

$$F_a(d) = (1 - \frac{d^2}{R^2})^2 \ , d \leq R \ , \tag{5.67}$$

kde d je vzdálenost bodu od generátoru a R je poloměr vlivu generátoru. Složitější potenciálová funkce může mít tvar

$$F_b(d) = -\frac{4}{9}\frac{d^6}{R^6} + \frac{17}{9}\frac{d^4}{R^4} - \frac{22}{9}\frac{d^2}{R^2} + 1, \tag{5.68}$$

která je od vzdálenosti $d \geq R$ nulová. Mezi další vlastnosti funkce (5.68) patří to, že má pro $d = 0$ a $r = R$ nulové derivace a že je symetrická pro hodnotu $d = 0.5R$ (viz obrázek 5.42).



Obrázek 5.42: Průběhy potenciálových funkcí F_a a F_b definovaných vztahy (5.67) a (5.68)

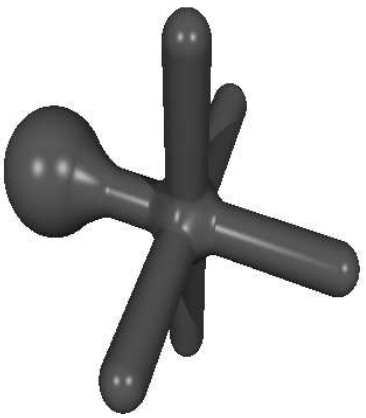
Protože potenciálová funkce závisí pouze na vzdálenosti v prostoru, prvek kostry určuje charakter izoploch spojených s implicitní funkcí (například izoplochy implicitní funkce určené bodem v prostoru jsou kulové plochy).

Každý prvek kostry je obklopen vlastním polem, jehož intenzita je maximální na povrchu a klesá se vzdáleností od prvku. Pole každého z prvků kostry vyplňuje prostor skalárním polem

$F_i(d_i)$, které přiřazuje každému bodu P v prostoru hodnotu $F_i(d_i(P))$. Celková intenzita pole v bodě P je určena funkcí

$$F(P) = \sum_{i=1}^{n} c_i F_i(d_i), \tag{5.69}$$

kde n je počet prvků kostry, c_i je skalární hodnota (reálné číslo) určující vliv i-tého prvku kostry v bodě P v prostoru na celkovou hodnotu funkce, F_i je potenciálová funkce pro i-tý prvek kostry a konečně d_i je vzdálenost i-tého prvku kostry od vyšetřovaného bodu P.

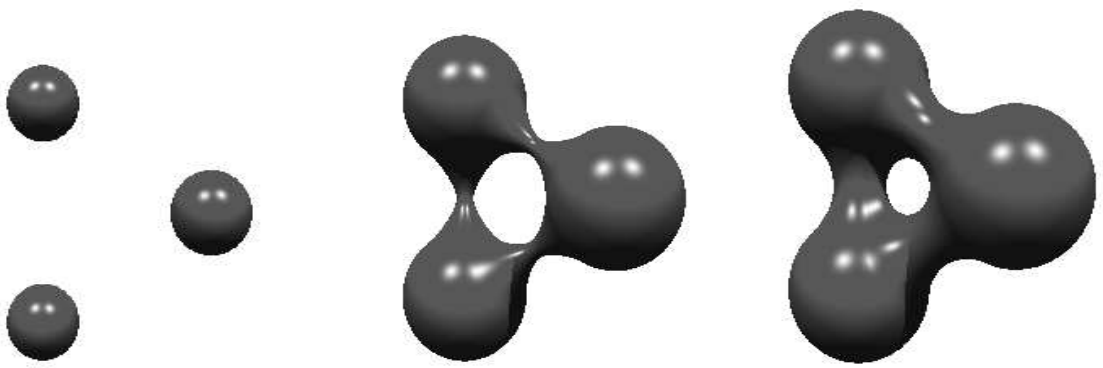


Obrázek 5.43: Implicitní plocha vzniklá ze tří úseček a jednoho bodu

Potenciálová funkce se také nazývá *směšovací funkce* (*blending function*). Název však není zcela výstižný, neboť o výši příspěvku od daného prvku kostry rozhoduje součin $c_i F_i(d)$. Modelovaný implicitní povrch (izoplocha) je určen těmi body v prostoru, jejichž hodnota $F(P) = konst$.

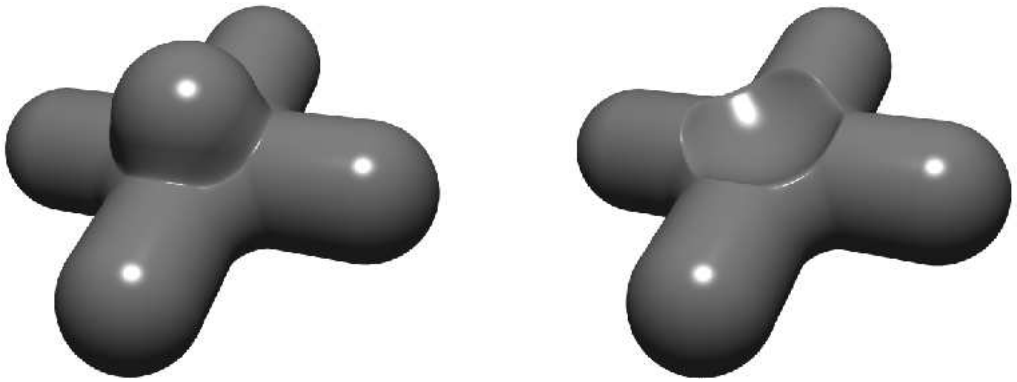
Izoplocha samozřejmě nemusí být souvislá, může se jednat o více objektů v prostoru podobně jako na obrázku 5.44.

Koeficient c_i určuje, jak daleko dosahuje vliv i-tého prvku kostry a jeho znaménko říká, jestli se má jeho vliv přičíst, nebo odečíst [viz rovnici (5.68)]. Obrázek 5.45 ukazuje vliv kladného i záporného parametru c_i na tvar plochy.



Obrázek 5.44: Postupné zvětšování parametrů, které určují rozsah vlivu polí kolem tří bodů

Určování vzdálenosti d_i, tj. vzdálenosti bodu P od prvků kostry (viz také kapitola 22.3.2) se



Obrázek 5.45: Dvě úsečky v jejichž středu je koule s parametrem c_i kladným (vlevo) a záporným

řídí pravidly, která jsou definována pro zvolený druh generátoru. Např. pokud je prvkem kostry úsečka, vypočítá se nejprve projekce tohoto bodu na přímku, na které úsečka leží a poté se testuje, zda je tento bod vnitřním bodem úsečky. Pokud tomu tak není, vypočítá se vzdálenost bodu P od bližšího z konců úsečky.

Modelování pomocí kostry má mnoho výhod. Kostra je poměrně intuitivní aproximací mnoha objektů reálného světa (např. lidské tělo má kostru). Kostra je snadno zobrazitelná, modelování je tedy názorné, relativně snadné, a na rozdíl od klasických parametrických modelářů, přehledné. Kostra obsahuje „zhuštěnou“ informaci. Nejedná se o plnou reprezentaci, ale na základě kostry implicitního tělesa, je možné získat poměrně slušnou představu (*preview*) o modelovaném objektu. Podobně jako u modelů CSG vzniká složité těleso kombinací těles jednoduchých. A konečně kostru je možno hierarchizovat tj. sdružovat jednotlivé části do bloků, určovat jejich vzájemné působení, atd.

5.12.1 Zobrazování implicitních ploch

Zopakujme, že funkce (5.68) přiřazuje každému bodu v prostoru nějakou skalární hodnotu a definuje tedy skalární pole. Námi hledaný objekt je izoplocha, která je definována jako množina těch bodů, které mají stejnou hodnotu přiřazenou funkcí (5.69). Úkolem zobrazování implicitních ploch je, jak tyto body najít.

Nejjednodušším, ale časově náročným řešením, je implicitní tělesa zobrazovat metodou sledování paprsku (viz kapitola 15.9). Při hledání průsečíku paprsku s takto reprezentovaným tělesem se používají numerické metody hledání průchodu funkce nulou. S výhodou lze využít omezeného vlivu jednotlivých prvků kostry a pro první detekci použít prostorovou obálku (např. kouli). Detaily je možno nalézt v [Bloo88].

Další metody zobrazování převádějí implicitní plochy na jinou reprezentaci, a to nejčastěji na ploškovou, existují i snahy o převod na parametrické plochy. Nejběžnějšímu postupu převodu na rovinné plošky se říká *polygonizace implicitních ploch*. Většina polygonizačních technik pracuje na principu rozdělení trojrozměrného prostoru a výpočtu jeho obsazení. Rozdělení prostoru je buďto uniformní (voxelizace), nebo neuniformní, nejčastěji pomocí oktalového stromu. Vzhledem k tomu, že funkce $F(P)$ je vlastně voxelovým prostorem rasterizována, projevují se zde všechny problémy rasterizaci vlastní – především alias, který je patrný zejména při animaci rasterizovaných objektů. Některé algoritmy polygonizace implicitních ploch jsou uvedeny v [Ning93, Over93].

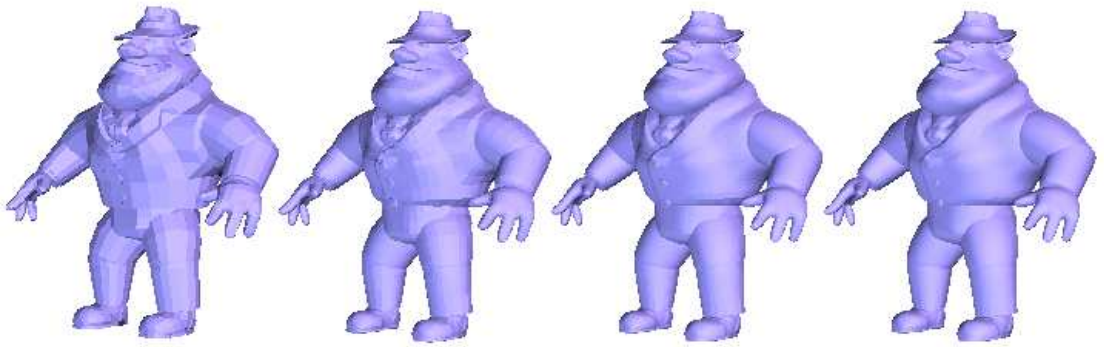
5.13 Dělené povrchy

V posledních letech se v souvislosti s rozvojem grafických procesorů dostává do popředí reprezentace modelů pomocí mnohoúhelníků – polygonů. Jen málo objektů v reálném světě má však ostré hrany. Při modelování objektů pomocí polygonů nastávají problémy s úrovní detailů. Zatímco modelování koule pomocí matematického vyjádření není problém, při polygonální reprezentaci objektů s členitým povrchem a detaily je situace mnohem složitější. Po volbě měřítka zobrazení můžeme povrch aproximovat dostatečným množstvím polygonů. Při mnohonásobném zvětšení se však problém hran objeví znovu. Řešení se nabízí v podobě dělení povrchů, které jsou popsány v této části.

Polygonální síť je množina vrcholů, určujících tvar povrchu, společně s informacemi o propojení mezi nimi, které definuje topologii povrchu. Hrany propojují vrcholy a vymezují plošky. *Dělení* je definováno jako rekurzivní proces, při kterém zjemňujeme polygonální síť přidáváním nových vrcholů a plošek, a tím ji zahlazujeme. Přináší možnost upravit modely reprezentované polygonální sítí na povrchy blížící se našemu očekávání. Podstatnou nevýhodou je zvyšování počtu vrcholů výsledného modelu, které přináší výrazné zpomalení. Přesto při zpracování animací i tvoření statických modelů hraje významnou roli. Pokud potřebujeme zvyšovat úroveň detailů, poskytuje dělení výhodné řešení. Takto generované povrchy se nazývají *dělené povrchy* (*subdivision surfaces*).

Dělené povrchy mohou být použity v celé řadě případů. Například při přenosu modelů pošleme pouze hrubý model a o vytvoření hladkého povrchu se postará příjemce. Další možností je použití v počítačových hrách a animacích v reálném čase, kdy po zjištění výkonu počítače stanovíme maximální úroveň detailů automaticky.

Nejdříve rozebereme základní pojmy a teoretickou podstatu dělení povrchů, v následujících kapitolách potom konkrétní techniky, a nakonec popíšeme implementaci jednoduché aplikace pro demonstrování základních dělicích schémat.

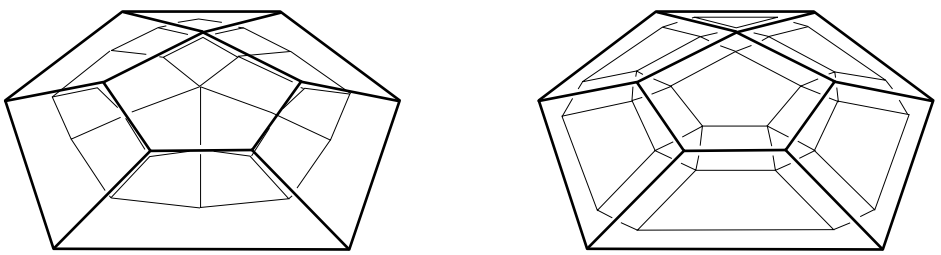


Obrázek 5.46: Příklad vytvoření hladkého povrchu dělením polygonálního modelu

Dělením generovaný povrch se v každém dalším kroku blíží limitě povrchu, která je nekonečněnásobnou aplikací dělicího schématu. V praxi však stačí provést několik kroků a výsledek ve spojení s hladkým stínováním je dostačující. Pokud je požadována vysoká kvalita (hladkost) povrchu, je proces dělení zastaven poté, co plošky vzniklé dělení mají velikost menší než jeden pixel. Postup je naznačen na obrázku 5.46.

5.13.1 Dělicí schémata

Dělicí pravidlo je předpis, který určité skupině vrcholů přiřadí rozsáhlejší skupinu vrcholů. Dělicí schéma zahrnuje pravidla dělení spolu s údaji jak propojit výsledné vrcholy do nové polygonální sítě. Dělení povrchů je aplikování určitého dělicího schématu na konkrétní polygonální síť. Při srovnávání dělicích schémat se vychází z vlastností, které charakterizují jejich aplikovatelnost a geometrické vlastnosti generovaných povrchů.



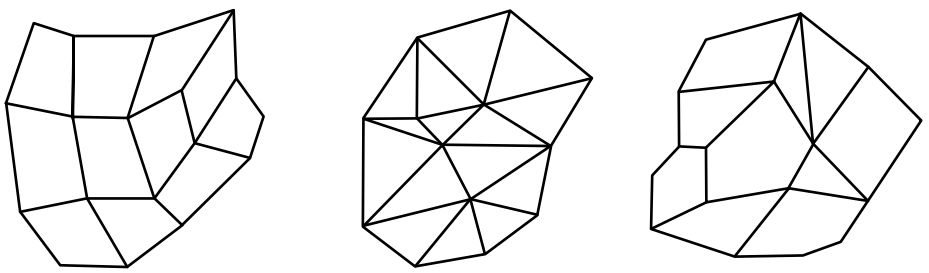
Obrázek 5.47: Dělicí schémata: dělení nad ploškami (vlevo), dělení nad vrcholy (vpravo)

Dělicí schémata je možné rozdělit podle způsobu vytváření nové sítě, a to podle toho, zda se

jedná o dělení nad ploškami (*face-split*) nebo dělení nad vrcholy (*vertex-split*). V prvním případě (obrázek 5.47 vlevo) jsou pro v okolí vrcholů plošek přidány nové vrcholy. Jejich následným propojením s dalšími nově vytvořenými vrcholy rozdělíme plošku na nové plošky. Při dělení nad vrcholy (obrázek 5.47 vpravo) se pro každý vrchol v rámci plošky vytvoří nový vrchol, který přísluší k dané plošce, k vrcholu, ke kterému byl vytvořen, a dvěma hranám sdílejícím tento vrchol. Nové plošky vznikají propojením vrcholů se stejnými příslušnostmi.

Schémata dělení nad ploškami mohou být aproximační nebo interpolační. Při aproximaci neprochází nově vytvořená polygonální síť vrcholy původní sítě a v každém kroku dělení se přibližuje k limitě povrchu. Naproti tomu při interpolaci se vrcholy původní sítě stávají zároveň vrcholy nové sítě a jsou i vrcholy limitního povrchu.

Jedním z nejdůležitějších požadavků kladených na dělicí schéma je to, aby se dalo aplikovat na model reprezentovaný polygonální sítí s libovolnou topologií. Znamená to, že plošky modelu mohou mít libovolný počet vrcholů. Na obrázku 5.48 jsou zobrazeny ukázky čtvercové a trojúhelníkové sítě a sítě s libovolnou topologií. Základní dělicí schémata prezentovaná v následujícím



Obrázek 5.48: Ukázka polygonálních sítí. (a) čtvercová síť (b) trojúhelníková síť (c) síť s libovolnou topologií

textu lze aplikovat na sítě s libovolnou topologií. Nejlepších výsledků je obvykle dosaženo na čtvercových sítích.

Mezi požadované vlastnosti dělicích schémat patří:

- *Možnost změny měřítka*, tj. volba různé úrovně detailů pro různě vzdálené objekty ve scéně. Rekurzivní výpočet dělení poskytuje možnost volit úroveň detailů dynamicky.
- *Rychlost a lokální definice*, kdy nové vrcholy jsou počítány pomocí několika jednoduchých početních operací. Výpočet by neměl záviset na příliš vzdálených vrcholech.
- *Invariance vůči afinním transformacím*, tj. možnost vytvářet objekty podrobené transformacím dělením, které je aplikovnáno na transformované sítě.

Tabulky ukazují přehled nejznámějších dělicích schémat s jejich vlastnostmi.

Dělení nad ploškami	Loop, Catmull-Clark, Butterfly, Kobbelt
Dělení nad vrcholy	Doo-Sabin, Midedge

Libovolná topologie	Catmull-Clark, Doo-Sabin, Midedge
Síť trojúhelníků	Loop, Butterfly

Aproximační dělicí schémata	Loop, Catmull-Clark
Interpolační dělicí schémata	Butterfly, Kobbelt

Podrobný přehled lze nalézt v [Zori00]. V následujících odstavcích se budeme zabývat pouze dvěma základními schématy Doo-Sabin a Catmull-Clark. Ostatní schémata jsou z těchto schémat odvozena a jsou navržena pro sítě s určitou topologií a pro splnění různých omezujících podmínek.

5.13.2 Schéma dělení Doo-Sabin

Aproximační schéma Doo-Sabin používá dělení nad vrcholy. *Ploškový bod* V_F je vypočten jako průměr všech vrcholů V_i plošky, jejichž počet je N, podle vzorce

$$V_F = \frac{1}{N}\sum_{i=1}^{N} V_i$$

Pro každý vrchol V vypočítáme nový vrchol V' příslušný k původnímu jako průměr ploškového bodu V_F, původního vrcholu V a středů hran M_E^1 a M_E^2, které z něj vycházejí a jsou hranami zpracovávané plošky:

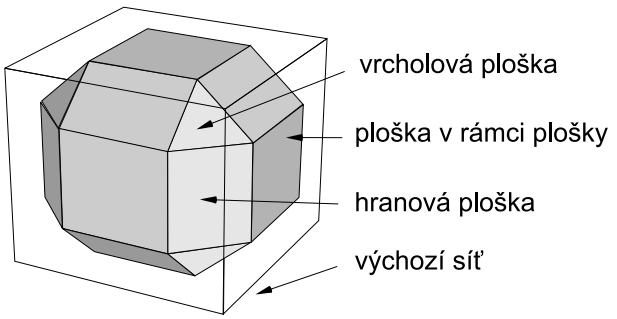
$$v' = \frac{V + V_F + M_E^1 + M_E^2}{4}$$

.

Každý nově vytvořený vrchol přísluší k původnímu vrcholu, zpracovávané plošce a jejím dvěma hranám sdílejícím původní vrchol. Pokud máme vypočítány všechny potřebné body, propojíme je do výsledné sítě. Vzniknou plošky tří základních typů podle toho, jakým způsobem byly vytvořeny.

- *Ploška v rámci plošky* (*face–face*) vznikne propojením nově vytvořených vrcholů pro danou plošku. Počet nových vrcholů v rámci této plošky je roven počtu vrcholů původní plošky.

- *Hranová ploška* (*edge–face*) vznikne propojením nově vytvořených vrcholů příslušných k dané hraně. Každá takto vytvořená ploška má čtyři vrcholy.
- *Vrcholová ploška* (*vertex–face*) vznikne propojením nově vytvořených vrcholů příslušných k původnímu vrcholu. Počet vrcholů nové plošky je roven počtu plošek sdílejících původní vrchol.

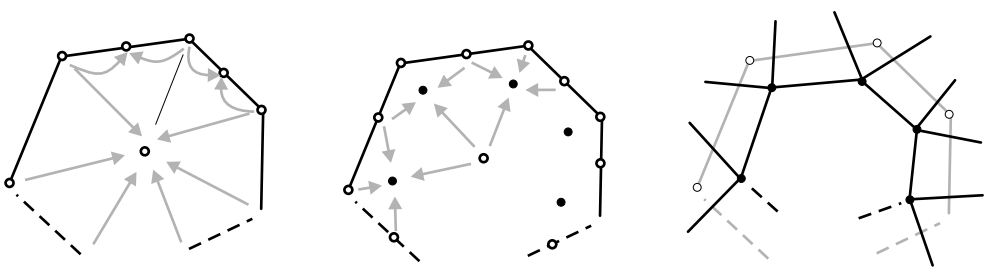


Obrázek 5.49: Typy plošek vytvářených algoritmem Doo-Sabin

Modely s otevřeným povrchem mohou obsahovat hraniční hrany. Při dělení jsou nové vrcholy V_1' a V_2' příslušné k hraničním vrcholům V_1 a V_2 vypočteny podle vztahu

$$V_1' = \frac{1}{4}(V_1 + 3V_2) \ , \quad V_2' = \frac{1}{4}(V_2 + 3V_1)$$

S těmito body vytvoříme novou hranovou plošku a vzhledem k zachování příslušnosti můžeme snadno vytvořit i krajní plošky v rámci vrcholu.



Obrázek 5.50: Doo-Sabin – postup dělení plochy nad vrcholy: průměr vrcholů a středy hran (vlevo), nové vrcholy (uprostřed), nová síť (vpravo)

Povrchy generované schématem Doo-Sabin splňují podmínky C^1 spojitosti. Tato podmínka neplatí pouze ve vrcholech na hranici otevřené polygonální sítě. Počet hraničních vrcholů zůstává při dělení konstantní. Nově vytvořené vrcholové plošky příslušné k hraničním vrcholům a plošky v dalších krocích z nich vytvořené konvergují k těmto vrcholům.

Podobným schématem k Doo-Sabin je Midedge. Nové vrcholy se počítají jednodušším způsobem a nezávisejí na všech vrcholech plošky. Nový vrchol V' vypočítáme jako průměr středů hran vycházejících z původního vrcholu.

$$V' = \frac{1}{2}M_E^1 + \frac{1}{2}M_E^2$$

Propojení polygonální sítě probíhá stejným způsobem jako v případě Doo-Sabin. Kvůli jednoduššímu výpočtu a nezohlednění všech vrcholů plošky poskytuje dělení Midedge pouze C^0 spojitost.

5.13.3 Schéma dělení Catmull-Clark

Aproximační schéma Catmull-Clark používá dělení nad ploškami. Pro každou plošku F původní sítě s vrcholy $V_1, ..., V_N$ spočítáme ploškový bod V_F jako průměr všech vrcholů plošky, analogicky jako v případě dělení Doo-Sabin. Tento bod se stává součástí výsledné sítě, zatímco pro předchozí schéma plnil pouze funkci mezivýsledku při výpočtu nových bodů.

Pro každou hranu E s koncovými body E_1 a E_2, která sdílí plošky F_1 a F_2, určíme hranový bod (*edgepoint*) V_E jako průměr vrcholů hrany a ploškových bodů V_F^1 a V_F^2 podle vztahu

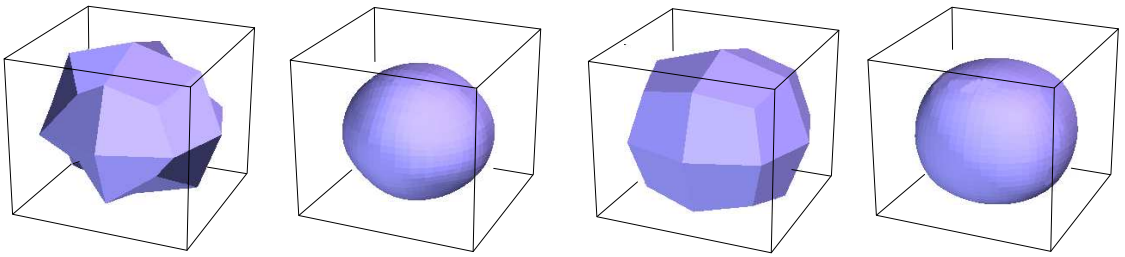
$$V_E = \frac{E_1 + E_2 + V_F^1 + V_F^2}{4}.$$

Pro výpočet vrcholového bodu V_V (*vertexpoint*) existují různé předpisy. V některých z nich dokonce máme na výběr z intervalu koeficientů a můžeme vytvořit nekonečně mnoho dělicích pravidel. Jejich výpočet je ale složitější, než v případě dvou základních vzorců, které uvedeme dále. Spočítáme průměr všech ploškových bodů V_F^i okolních plošek sdílejících původní vrchol V a průměr hranových bodů V_E^i hran z něj vycházejících. Nový vrcholový bod V_V příslušný k původnímu vrcholu V získáme dosazením do jednoho z následujících vzorců.

$$V_V = \frac{1}{N}\left(\frac{1}{N}\sum_{i=1}^{N} V_F^i + \frac{2}{N}\sum_{i=1}^{N} V_E^i + \frac{(N-3)}{N}V\right), \tag{5.70}$$

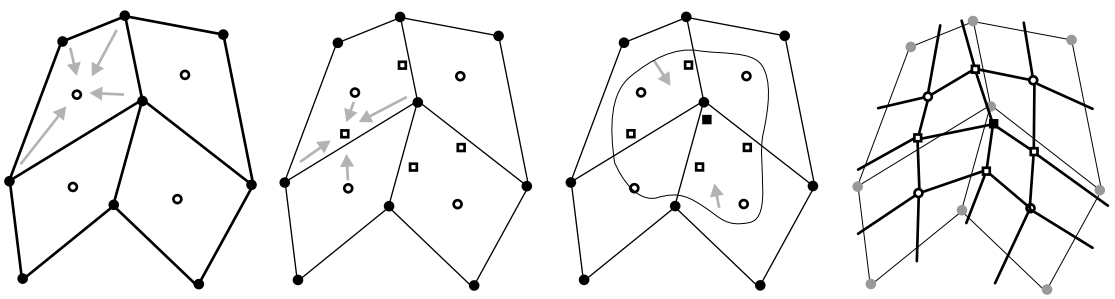
$$V_V = \frac{1}{N}\left(\frac{1}{N}\sum_{i=1}^{N} V_F^i + \frac{1}{N}\sum_{i=1}^{N} V_E^i + \frac{(N-2)}{N}V\right). \tag{5.71}$$

Rozdíly v obou vzorcích jsou patrné na obrázku 5.49 při aplikaci na konkrétní model – krychli. Ve vztahu (5.71) mají původní vrcholy větší vliv na pozici výsledného vrcholu.



Obrázek 5.51: Použití vzorce (5.70) po jednom a po čtyřech krocích (vlevo), vzorce (5.71) po jednom a po čtyřech krocích (vpravo)

Výpočet pomocí tohoto schématu je o něco složitější, než u dělení Doo-Sabin, protože k výpočtu hranových bodů musíme mít předpočítány ploškové body sousedních plošek. Pokud počítáme vrcholový bod, využíváme předpočítaných hranových i ploškových bodů. Nová ploška vznikne propojením ploškového bodu k oběma hranovým bodům a jejich následným propojením s novým vrcholovým bodem. V rámci původní plošky s vrcholy $V_1, ..., V_N$ se vytvoří N nových plošek a každá bude mít čtyři vrcholy. Čtvercová síť se vytvoří hned v prvním kroku dělení a s dalšími kroky zůstává tato topologie zachována. Postup výpočtu a vytvoření části nové sítě je ukázáno na obrázku 5.52.



Obrázek 5.52: Dělení pomocí schéma Catmull-Clark. Zleva doprava: výpočet ploškových bodů, hranových bodů, vrcholového bodu a část výsledné čtyřúhelníkové sítě.

Na hranici povrchu počítáme hranové body pouze jako průměr vrcholů hrany, nijak se tedy

nezohlední vnitřní ploškový bod. Pro výpočet vrcholového bodu V_V použijeme původní vrchol V a vrcholy V_1 a V_2, které spolu s V tvoří okrajové hrany povrchu. Pro výpočet použijeme následující vztah:

$$V_V = \frac{3}{4}V + \frac{1}{8}V_1 + \frac{1}{8}V_2.$$

Na rozdíl od dělení Doo-Sabin se každý vrchol počítá jiným způsobem podle toho, zda přísluší k hraně, vrcholu nebo plošce. Schéma Catmull-Clark generuje C^1 spojité povrchy. Mimo výjimečné vrcholy splňují povrchy dokonce podmínky C^2 spojitosti. Všechny tyto vlastnosti vyplývají z dělení bikubických B-spline křivek, podrobnější rozbor nalezneme v [Zori00].

Kapitola 6

Reprezentace a modelování těles

Mnoho počítačových objektů v trojrozměrném prostoru má charakter tělesa. Jsou obdobou skutečných hmotných předmětů, které zaujímají určitý objem. Na těleso proto můžeme nahlížet jako na množinu bodů v trojrozměrném prostoru, splňujících určitá kritéria. Definujeme-li relaci sousednosti bodů (jedna z možných definic sousednosti je uvedena v části 7.2.2), pak můžeme nahlížet na těleso jako na sjednocení dvou disjunktních množin – množiny vnitřních bodů a množiny hraničních bodů. Každý vnitřní bod sousedí pouze s vnitřními nebo hraničními body. Hraniční bod sousedí alespoň s jedním hraničním bodem, vnitřním bodem a bodem vnějším, nepatřícím do žádné z uvedených dvou množin. Těleso je přitom chápáno jako spojitý útvar, tvořený jedním celkem (byť s možnými otvory). Výše uvedená definice vylučuje ze skupiny těles takové objekty, jakými jsou například úsečky či křivky v prostoru, ale i části rovin a obecné plochy. Ty totiž nemají žádné vnitřní body ve smyslu tělesa. Přesto jsou tyto objekty pro popis těles cenné, neboť slouží k popisu množiny hraničních bodů.

V této kapitole popíšeme typické způsoby reprezentace těles. Nejvíce prostoru bude věnováno reprezentaci hraniční, která je výhodná zejména z hlediska dalšího zpracování – její zobrazování se snadno provádí v grafických procesorech. Dále uvedeme způsob popisu těles, používaný zejména v systémech CAD, tzv. konstruktivní geometrii těles. V závěru kapitoly popíšeme techniku pro modelování pomocí deformací.

6.1 Trojúhelníky a sítě trojúhelníků

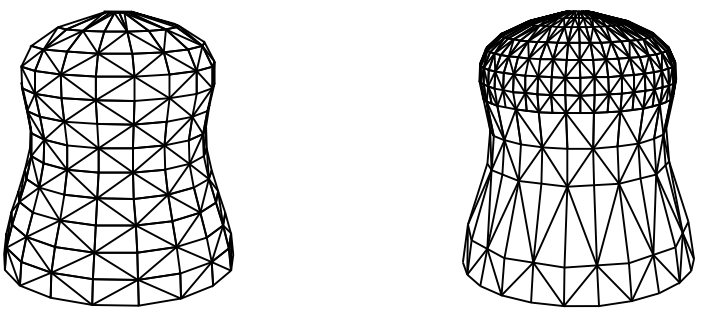
Díky své jednoduchosti je trojúhelník oblíbeným stavebním kamenem většiny reprezentací. Má mnoho dobrých vlastností – na rozdíl od obecných mnohoúhelníků je vždy konvexní a všechny jeho vrcholy leží v rovině. Pro jeho vyplňování existují velmi rychlé algoritmy (viz část 3.3.2). Zobrazování je podporováno grafickým procesorem. Mnoho geometrických výpočtů

nad trojúhelníkem lze optimalizovat, například výpočty průsečíku paprsku s trojúhelníkem, a proto se někdy v metodě sledování paprsku (viz část 15.9) místo výpočtu průsečíku se složitým objektem tento objekt převádí na ploškovou reprezentaci a výpočty průsečíků se provádějí s ní.

Dříve, než přistoupíme k popisu těles, popíšeme možnosti popisu trojúhelníků a jejich skupin – sítí (*triangle mesh*). Sítí nazýváme množinu trojúhelníků, které sdílejí své hrany. Datová struktura popisující síť bývá rozdělena do dvou logických částí – *geometrické* a *topologické*. V geometrické části jsou zaznamenány souřadnice vrcholů trojúhelníků, topologická část udržuje údaje o tom, které vrcholy tvoří trojúhelník, případně o tom, které trojúhelníky spolu sousedí. Toto rozdělení je praktické pro některé operace se sítí. Například při geometrických transformacích (viz část 21) se vypočítávají nové souřadnice vrcholů. Ty poskytuje právě geometrické část sítě nezávisle na části topologické. Při tvorbě sítí sledujeme zejména tato kritéria:

1. přesné a úsporné vyjádření tvaru, který síť vyjadřuje,
2. uspořádání vhodné pro další zpracování sítě.

První kritérium se týká geometrie sítě a uvažujeme jej zejména při převodu z jiné reprezentace do sítě trojúhelníků. Samotná trojúhelníková síť totiž není výhodná pro modelování tvaru těles a ploch a modelovací programy obyčejně pracují s reprezentací jinou (nejčastěji s NURBS, viz odstavec 5.10). Při převodu do sítě trojúhelníků je třeba síť optimalizovat tak, aby při co nejmenším počtu trojúhelníků vyjadřovala co nejpřesněji vymodelovaný tvar. Příklad dvou sítí reprezentujících stejnou plochu, avšak s různou přesností, je na obrázku 6.1.



Obrázek 6.1: Síť trojúhelníků pokrývající plochu pravidelně (vlevo) a pokrývající plochu jemněji v místech s větší křivostí (vpravo)

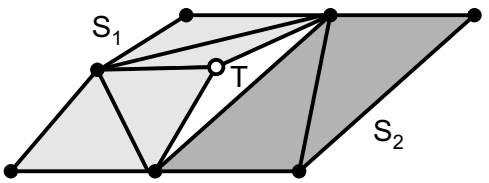
Druhá skupina požadavků kladených na síť souvisí s její topologií. Oddělení geometrických a topologických údajů do dvou datových struktur je pro některé úlohy nevyhovující. Například při zpracování dat v grafickém procesoru je většinou nutno popsat síť pouze pomocí jedné lineární struktury (pole), přitom však tak, aby byl minimalizován počet operací prováděných

s jednotlivými vrcholy sítě. Toho dosáhneme výběrem trojúhelníků do posloupnosti, která odpovídá *pruhu trojúhelníků* (*triangle strip*) na obrázku 6.1 vlevo. Toto uspořádání zajišťuje, že každý vrchol v pruhu bude zpracován právě jednou. Datová struktura je v tomto případě posloupností souřadnic vrcholů. Každý vrchol tvoří spolu s předchozími dvěma vrcholy jeden trojúhelník. Obdobně lze trojúhelníky uspořádat do *vějíře trojúhelníků* (*triangle fan*), viz obr. 6.1 vpravo. Vějíř vznikne například při triangulaci konvexního polygonu.

Obrázek 6.2: Pruh trojúhelníků (vlevo) a vějíř trojúhelníků (vpravo)

Nalezení optimální množiny pruhů či vějířů trojúhelníků pro libovolnou síť trojúhelníků je zajímavým výpočetním problémem, který souvisí nejen s charakterem zpracovávané sítě na vstupu, ale především s různými požadavky kladenými na výstup (počet pruhů, počet trojúhelníků v pruzích apod.) a nelze jej považovat za obecně jednoznačně definovanou a vyřešenou úlohu. Mezi další praktické úlohy při zpracování trojúhelníků pak patří algoritmy zjednodušování trojúhelníkové sítě (tzv. *decimace*, viz část 19.2.1) či *triangulace* obecného mnohoúhelníku. Další oblast počítačové grafiky se zabývá kompresí sítí.

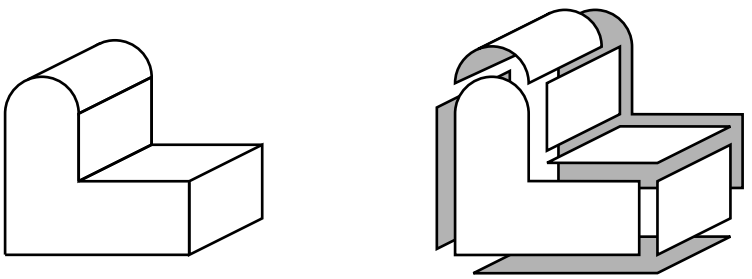
Trojúhelníky a jejich sítě mají i svoje nevýhody. Mezi ně patří zejména nesnadné mapování textur, u něhož je třeba explicitně určit vztah mezi obrázkem textury a každým vrcholem trojúhelníka. Dále je nevýhodou tzv. *geometrický alias*. Tento nežádoucí jev se projevuje při změně měřítka. Těleso, které bylo původně pokryto několika samostatnými sítěmi dotýkajícími se na hranách, se při změně měřítka může opticky roztrhnout (*crack*), neboť dojde k numerické chybě vlivem konečné reprezentace čísel v paměti počítače. K tomuto chování dochází zejména v sousedství tzv. *T-vrcholů* (viz obr. 6.3). Jejich existenci je vhodné detekovat a odstranit vhodnou úpravou sítě.



Obrázek 6.3: Roztržení dvou sousedních sítí u T-vrcholu

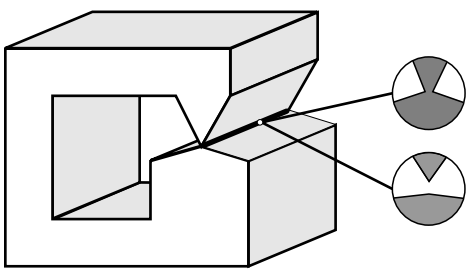
6.2 Hraniční reprezentace těles

Jeden z nejběžnějších způsobů reprezentace těles spočívá v popisu hranice (*boundary representation, B–rep*), tedy v popisu množiny hraničních bodů, jak ukazuje obr. 6.4. Informace o vnitřních bodech tělesa se buď neuchovávají, nebo je lze odvodit z popisu hranice. Hranice tělesa je jeho přirozenou reprezentací, vždyť většina lidí kreslí tělesa právě pomocí jejich obrysu, který je podmnožinou hranice.



Obrázek 6.4: Popis tělesa je v hraniční reprezentaci převeden na popis pláště

6.2.1 Manifoldy a Eulerova rovnost



Obrázek 6.5: Nonmanifold a dvě možnosti jeho převedení na manifold

Výše uvedená definice tělesa, jakožto objektu tvořeného vnitřními a hraničními body, je vhodná pro geometrické výpočty používané např. při modelování tvaru tělesa v systémech CAD. Z praktického hlediska je však příliš široká, neboť dovoluje popsat i takové objekty, které nelze ve skutečném světě vyrobit. Proto se zavádí pojem *manifold* pro takové modely těles, které odpovídají nějakému skutečnému tělesu. Někdy se také nazývají *2-manifoldy*.

Skutečnost, že počítačový popis může určovat i nevyrobitelné těleso, pramení z použití matematické a geometrické abstrakce. Všichni si například dokážeme představit, že přímka je nekonečně tenká nebo že se dva objekty dotýkají v jednom bodě. Ve skutečném světě však nejsme schopni vyrobit nekonečně tenké vlákno, ani nedokážeme spojit dvě tělesa ideálně bodovým svárem. Obrázek 6.5 ukazuje příklad nevyrobitelného tělesa, nazývaného *nonmanifold*. Zvýrazněná hrana tohoto nonmanifoldu je z geometrického hlediska nekonečně tenká úsečka, která je průsečnicí čtyř ploch (inciduje se čtyřmi plochami).

Ve skutečnosti musí být v daném místě hrany dvě a těleso je buď propojeno nebo rozpojeno, jak naznačují zvětšené detaily.

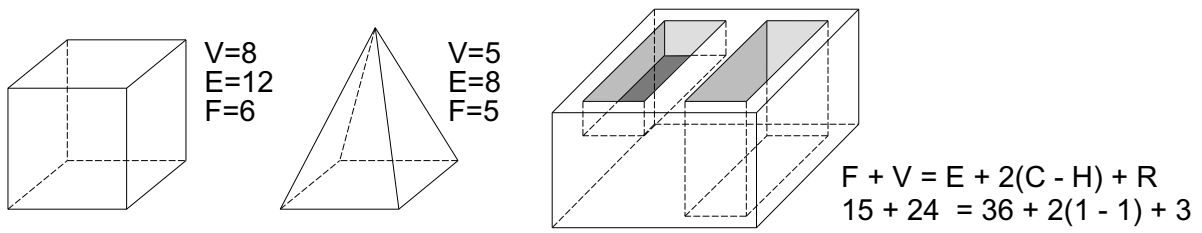
Bez uvedení přesné definice budeme za manifold z praktického hlediska považovat takové těleso, jehož každá hrana inciduje právě se dvěma plochami a jehož hrany neprotínají jiné plochy. Obdobně platí, že osamocený vrchol nesmí spojovat dvě části tělesa.

Eulerova rovnost

Velkou třídu prakticky používaných těles tvoří mnohostěny. Mnohostěn je těleso, které je ohraničeno množinou mnohoúhelníkových stěn, každou hranu sdílí vždy sudý počet stěn (v případě 2-manifoldu sdílejí každou hranu dvě stěny) a splňuje další podmínky. *Jednoduchý mnohostěn* je těleso, které lze volnou plastickou deformací převést na kouli, je to těleso bez děr. Hraniční reprezentace jednoduchého mnohostěnu splňuje Eulerovu rovnost, která udává vztah mezi počtem vrcholů (*V, vertex*), hran (*E, edge*) a stěn (*F, face*):

$$F \; + \; V \; = \; E \; + \; 2. \tag{6.1}$$

Platnost Eulerovy rovnosti sama o sobě neprokazuje, že jistá množina vrcholů, stěn a hran tvoří jednoduchý mnohostěn, který je hranicí uzavřeného objemu. Navíc musí platit, že každá hrana propojuje dva vrcholy, a stěny a hrany se nesmějí protínat.



Obrázek 6.6: Ukázky mnohostěnů a jejich charakteristických prvků splňujících Eulerovy rovnosti

Pro manifoldy, které mají otvory, platí zobecněná Eulerova rovnost (6.2). Připomeňme, že těleso bez otvoru je homeomorfní s koulí, těleso s jedním otvorem je homeomorfní s toroidem (prstencem) atd. Mezi otvory nepočítáme slepé prohlubně. V Eulerově rovnosti jsou přidány další členy, a to počet vnitřních smyček hran (*R, ring*), počet oblastí neboli samostatných komponent tělesa (*C, component*) a počet otvorů procházejících tělesem (*H, hole*). Vnitřní smyčky vymezují otvory ve stěnách, nikoliv otvory v tělese:

$$F \; + \; V \; = \; E \; + \; 2.(\; C \; - \; H \;) \; + \; R. \tag{6.2}$$

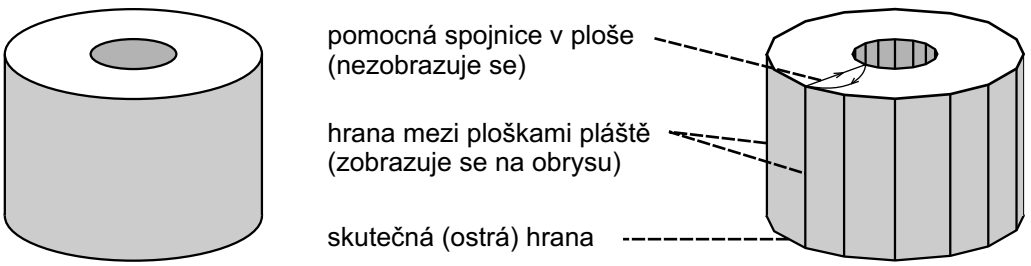
Příklady dvou jednoduchých mnohostěnů jsou na obr.6.6 vlevo. Těleso s jednou komponentou, jednou prohlubní a jedním průchozím otvorem je nakresleno vpravo.

Eulerovy formule mají úzkou souvislost s planárními grafy, na něž lze 2-manifoldy převádět – vzpomeňme na papírové skládanky prostorových útvarů, které jsou nakresleny na papíře (2D rovině) a které se tvarují překládáním (*fold*).

6.2.2 Vrcholy, hrany a stěny

K popisu hranice tělesa používáme základní prostorové prvky, jakými jsou body, úsečky a části rovinných ploch. Lze také použít části křivek a obecných ploch, avšak v této části se omezíme jen na jednodušší prvky. Všimneme si také situace, kdy je zakřivená plocha nahrazena sítí rovinných plošek.

Na obrázku 6.7 je zobrazen model válce s otvorem. V levé části vidíme jeho přesný geometrický tvar, v pravé části pak aproximaci množinou plošek. Vlivem aproximace vznikly nové hrany mezi plochami. Je vhodné každé z nich přiřadit atribut určující charakter hrany. *Pomocné* hrany tvoří nejčastěji spojnice mezi aproximujícími ploškami. Neměly by být vykreslovány, pokud netvoří obrys tělesa. Jejich zobrazení tedy závisí na natočení tělesa při promítání (podrobněji viz kap. 11).



Obrázek 6.7: Válec s otvorem (vlevo) je aproximován rovinnými ploškami (vpravo). Jeho hraniční reprezentace obsahuje hrany různých typů.

Druhý typ pomocných hran je používán v systémech, které předpokládají, že každá plocha má jen jediný (křivočarý) okraj. Tehdy je nutno propojit oba okraje horní podstavy pomocnými spojnicemi, které se nikdy nezobrazují. Častěji se ovšem tyto spojnice vůbec nedefinují, neboť současné systémy předpokládají, že každá plocha může mít jeden okraj vnější a několik vnitřních. Vnitřní okraje tvoří části hranice, nazývané v Eulerově formuli (6.2) písmenem R. *Ostré*, skutečné hrany jsou vykreslovány vždy. Na obrázku to jsou úsečky ohraničující oba okraje podstavy (vnější a vnitřní).

V některých aplikacích se nepoužívá klasifikace hran na pomocné a ostré a ponechává se rozhodnutí o typu hrany na zobrazovací fázi. Při ní je porovnáván úhel svíraný plochami, které s hranou sousedí. Pokud se neodchyluje od plného úhlu (tj. od 180°) o více, než je zvolená

odchylka, jsou hrany považovány za pomocné. Odchylka, určující kdy bude hrana považována za pomocnou či ostrou, se anglicky nazývá *crease angle*.

Geometrické prvky, ze kterých je sestavena hraniční reprezentace tělesa, bývají nejen doplněny různými příznaky, jako tomu bylo u hran plošek, ale také uspořádány do hierarchických struktur. Důvodem jsou požadavky, kladené zejména zobrazovacími algoritmy. Na kvalitní hraniční reprezentaci bývá požadováno, aby následující informace buď přímo obsahovala nebo aby je z ní bylo možno snadno odvodit:

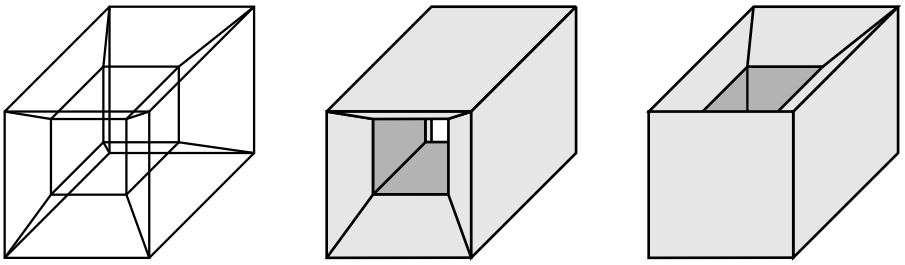
1. *klasifikace hran* na ostré a pomocné – není-li přímo obsažena v reprezentaci, je třeba znát, které plochy s hranou sousedí;
2. *normály ve vrcholech* – jednotkové vektory kolmé na těleso ve vrcholech jsou důležité ve fázi zobrazování, při určování osvětlení povrchu tělesa. Jsou buď přímo zadané nebo je lze určit ze znalosti ploch, které s vrcholem sousedí;
3. *ohraničení plochy* – je třeba umět nalézt všechny hrany dané plochy,
4. *poloha bodu v prostoru* – pro libovolný bod je třeba umět stanovit, zda leží uvnitř či vně tělesa.

Tyto požadavky přímo nebo nepřímo souvisejí především s topologií tělesa, tedy informacemi o tom, které vrcholy, hrany a plochy spolu sousedí. Geometrické informace (např. souřadnice vrcholů) by měly spolu s topologickými tvořit co nejúplnější popis tělesa. V následujících částech se seznámíme s několika možnostmi.

6.2.3 Hranová reprezentace

Nejstarší a nejjednodušší metoda popisu povrchu tělesa spočívá pouze v zápisu hran a vrcholů. Tato informačně chudá *hranová reprezentace* připomíná prostorové drátové modely těles pro školní účely. Proto se někdy nazývá *drátový model* (*wire–frame model*).

Při implementaci drátového modelu vytvoříme jeden seznam vrcholů a jeden seznam hran. V seznamu vrcholů jsou uloženy souřadnice. Každá položka seznamu hran má právě dva ukazatele do seznamu vrcholů. Výsledkem je úsporná struktura, která však obsahuje minimum topologických informací, a proto drátový model nelze jednoznačně interpretovat. Prosté vykreslení všech hran modelu (obr. 6.8 vlevo) neposkytuje dostatečnou informaci o tvaru tělesa ani o jeho pozici vzhledem k pozorovateli. Jeden model může být interpretován jako několik různých těles (obr. 6.8 uprostřed a vpravo). Samotné vykreslení hran je však praktické pro vytvoření náhledu, s jehož pomocí lze prozkoumat vnitřek tělesa, který by jinak byl zakryt hraničními plochami.



Obrázek 6.8: Nejednoznačnost drátového modelu

6.2.4 Jednoduchá plošková reprezentace

Další způsob popisu těles uvažuje hranici tvořenou plochami. Reprezentace je přirozeným rozšířením předchozí datové struktury složené pouze z vrcholů a hran o další vrstvu ploch. Plochy mohou být uspořádány do sítě trojúhelníků nebo obecných polygonů. Z implementačního hlediska jsou trojúhelníky výhodnější, neboť topologii jejich sítě snadno zaznamenáme např. do pole, jehož každá položka pevné délky obsahuje právě tři ukazatele do seznamu vrcholů.

Při použití obecných polygonů má datový záznam pro jeden polygon proměnnou délku. Tyto záznamy můžeme implementovat dynamicky v podobě zřetězeného seznamu. Stejně dobře však můžeme použít pro reprezentaci všech polygonů obyčejné jednorozměrné pole. Pro každý polygon zapíšeme do pole nejprve počet jeho vrcholů, za nímž následují pořadová čísla vrcholů (odkazy do seznamu vrcholů). Kromě explicitního vyjádření počtu vrcholů polygonu se používá varianta, při níž jsou nejprve do pole zapsány indexy vrcholů jednoho polygonu a za nimi následuje dohodnutý ukončovací symbol, např. záporné číslo.

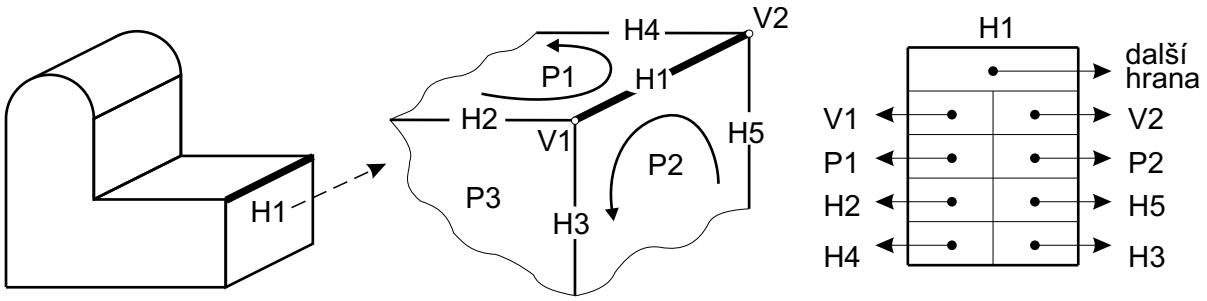
Pořadí vrcholů na obvodu polygonu je vhodné uchovávat s ohledem na orientaci stěny tělesa, kterou polygon reprezentuje. Časté je zadávání vrcholů proti smyslu pohybu hodinových ručiček (*ccw, counter clockwise*). Lze z něj odvodit směr normálového vektoru (u konvexních polygonů podle pravidla pravé ruky). Normálový vektor směřuje ven z tělesa.

Oproti hranové reprezentaci poskytuje jednoduchá plošková reprezentace více informací o tělese a je vhodná pro vykreslování zohledňující viditelnost (viz část 11). Stále však neobsahuje dostatek topologických informací. Chybějí v ní například údaje o typu hran a nelze snadno zjistit, které plošky sousedí s danou ploškou.

6.2.5 Strukturovaná plošková reprezentace

Nejznámější komplexní datovou strukturu pro hraniční reprezentaci navrhl Baumgart [Baum75]. Je známa pod názvem *okřídlená hrana* (*winged–edge*), neboť grafické znázornění jedné hrany

společně s prvky, které s ní sousedí, připomíná křidélka (obr. 6.9 uprostřed). Datový záznam hrany obsahuje ukazatele na sousední geometrické elementy (plochy, hrany, vrcholy). Všechny okřídlené hrany jsou zřetězeny. Jedno těleso je tvořeno třemi seznamy v hierarchickém uspořádání. Na nejnižší úrovni je seznam vrcholů. Na střední úrovni je seznam okřídlených hran a na nejvyšší úrovni seznam ploch daného tělesa.



Obrázek 6.9: Datový záznam okřídlená hrana

Na obrázku 6.9 vpravo je schéma záznamu pro okřídlenou hranu. Kromě odkazů na oba koncové vrcholy (V1, V2) v něm nalezneme ukazatele na sousední plochy (P1, P2). Navíc obsahuje informace o dalších čtyřech hranách. V levé části schématu jsou hrany sousedící s levou plochou (H2, H4), v pravé části hrany sousedící s pravou plochou (H3, H5), a to s ohledem na orientaci těchto ploch. Horní ukazatel pak reprezentuje spojnici na libovolnou následující hranu v zřetězeném seznamu okřídlených hran.

Seznamy vrcholů a ploch jsou mnohem jednodušší. Záznam každé plochy obsahuje ukazatel na libovolnou z jejích hran, případně na hranu patřící k vnitřní hranici (otvoru); volitelně pak normálový vektor, barvu či jinou materiálovou charakteristiku.

Ze záznamu okřídlená hrana je možno odvodit mnoho topologických údajů. Lze například snadno nalézt

- plochy sousedící s danou plochou,
- plochy incidující s danou hranou,
- plochy stýkající se v daném vrcholu,
- vrcholy a hrany dané stěny.

Na okřídlené hraně si všimneme typické vlastnosti datových struktur používaných v trojrozměrné grafice. Samotné geometrické údaje, tj. souřadnice vrcholů umístěné v seznamu vrcholů, zabírají přibližně jen 25 % paměti. Zbývající tři čtvrtiny objemu struktur popisujících těleso jsou použity k popisu topologie.

Datová struktura okřídlená hrana je určena pouze pro manifoldy, neboť každá (okřídlená) hrana inciduje právě se dvěma stěnami. Pro reprezentaci nonmanifoldů se používá odvozená datová struktura, nazvaná *půlhrana*. Jedna hrana modelovaného tělesa je rozložena na skupinu půlhran spojujících tutéž dvojici vrcholů. Samotná půlhrana představuje dvojici stěna-hrana, z cyklického zřetězení půlhran lze odvodit také informaci o sousední stěně. Běžné hrany se zapíší pomocí dvojice půlhran, případy charakteristické pro nonmanifoldy (obr. 6.5) čtyřmi a více půlhranami.

6.2.6 Bodová reprezentace

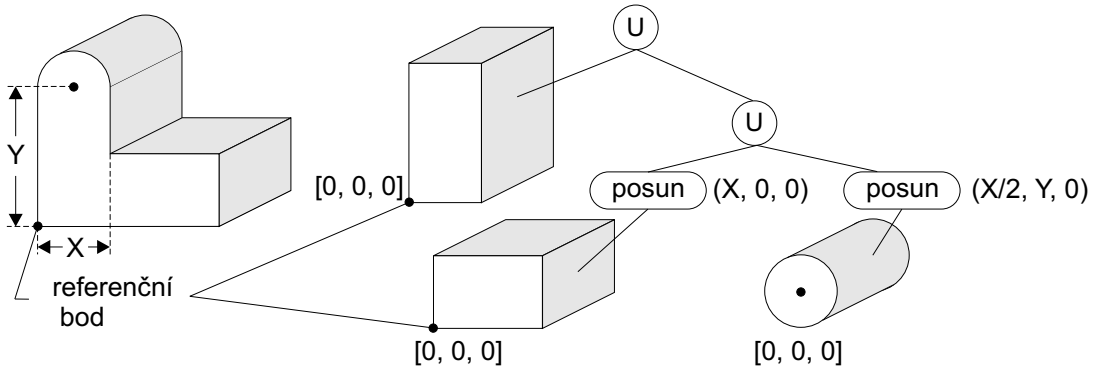
Zvláštním případem hraniční reprezentace je množina povrchových bodů. Ty jsou většinou získány digitálním snímáním reálných objektů, mohou být i výsledkem nějakého algoritmu. Každý bod představuje určitou část povrchu dané velikosti a nese informaci o své poloze (souřadnicích), normálovém vektoru, barvě, případně dalších vlastnostech týkajících se odrazu světla.

Je zřejmé, že paměťové nároky této reprezentace jsou velmi vysoké. Počítačový model, získaný digitalizací sochy Davida od Michelangela týmem z univerzity ve Stanfordu v roce 1999 [Levo00], obsahuje v nejvyšší kvalitě několik bilionů bodových vzorků, které zabírají diskový prostor o velikosti 32 GB. To je samozřejmě extrémní případ, kdy bodový model neslouží pouze pro potřeby počítačové grafiky, ale zejména pro účely přesné digitální archivace reálných objektů. Je však pravdou, že i jednodušší modely v nižší přesnosti vyžadují řádově jednotky až desítky MB dat. Pro potřeby zobrazování se bodové vzorky uspořádají hierarchicky a zobrazují se pouze do určité úrovně. Základní metody zobrazování těles popsaných bodovou reprezentací jsou uvedeny v části 11.5.

6.3 Konstruktivní geometrie těles

V oblasti CAD se často tělesa popisují způsobem, který odráží postupy používané konstruktérem při navrhování tvaru tělesa. Metoda, nazývaná *konstruktivní geometrie těles* (*CSG, Constructive Solid Geometry*), je založena na reprezentaci tělesa stromovou strukturou (CSG stromem), uchovávající historii dílčích konstrukčních kroků. Z jednoduchých geometrických objektů, tzv. CSG primitiv, je pomocí množinových operací a prostorových transformací vytvořen výsledný objekt. Jako primitiva slouží jednoduchá tělesa (kvádr, koule, válec, kužel, jehlan či toroid), lze však použít i abstraktnější entity, například poloprostor nebo plocha NURBS.

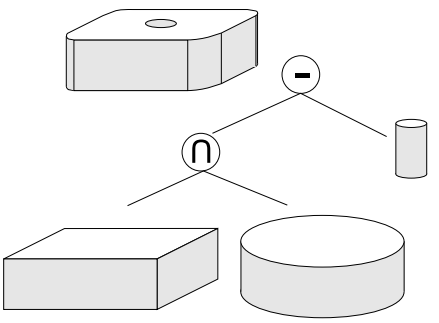
Množinové operace odpovídají konstrukčním postupům a mohou být prováděny jak s CSG primitivy, tak s celými CSG stromy. Zjednodušeně lze říci, že sjednocení představuje spojování součástí (sváření, lepení), rozdíl je obdobou vrtání a průnik odpovídá odříznutí či zbroušení.



Obrázek 6.10: Těleso a jeho popis CSG stromem, obsahujícím explicitně zadané transformace

Příklady CSG stromů jsou na obrázcích 6.10 a 6.11. Vnitřní uzly CSG stromu obsahují operace, v listech jsou zapsány údaje o CSG primitivech. Transformace mohou být chápány jako CSG operace, jak ukazuje obr. 6.10. Jiný přístup zaznamenává transformace ke každému primitivu (obr. 6.11), takže vnitřními uzly jsou pouze množinové operace.

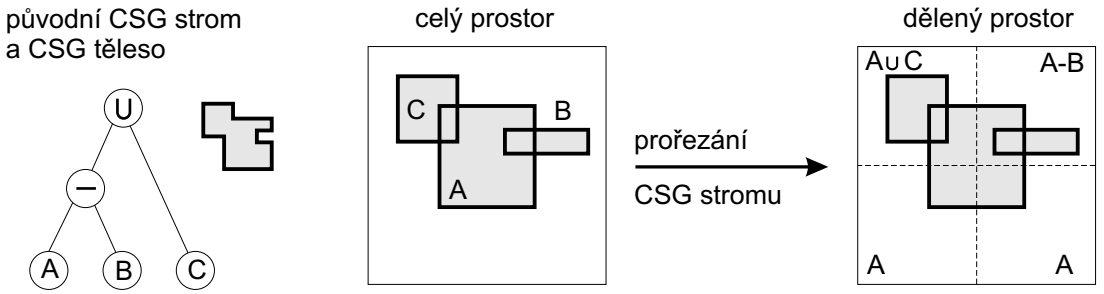
Reprezentace tělesa CSG stromem byla zavedena především pro konstruktéry, tedy pro fázi vytváření a tvarování tělesa. Pro zobrazení není příliš vhodná, neboť neobsahuje přímo vykreslitelné geometrické prvky, jakými jsou hrany nebo plochy. Existují sice zobrazovací metody, které pomocí tzv. sledování paprsku (podrobněji viz kap. 15.9) nebo upraveného algoritmu využívajícího paměť hloubky (viz kap. 11) dokáží přímo vykreslit CSG těleso, avšak z hlediska rychlosti je lepší převést CSG strom do některé jiné reprezentace, například hraniční.



Obrázek 6.11: CSG strom, jehož primitiva jsou již transformována

Vyhodnocení CSG stromu a nalezení hraniční reprezentace pro jeho kořen je poměrně složitý proces, který souvisí s modelováním těles a v tomto textu se jím nebudeme zabývat. Všimneme si však sympatické možnosti zjednodušit CSG strom v případě, kdy je modelovací, resp. zobrazovací prostor rozdělen do menších částí, typicky uspořádaných do pomocné hierarchické struktury (viz též část 14.2); zaznamenáme namísto původního CSG stromu tzv. *prořezaný CSG strom*. Princip je jednoduchý: pokud se v dané části prostoru nějaké primitivní těleso nevyskytuje, pomocí jednoduchých pravidel odstraníme nevyužité větve CSG stromu. Ve spodních kvadrantech (na obr. 6.12 vpravo) se vyskytuje pouze těleso A a je tedy zbytečné v těchto oblastech

uvažovat sjednocení s tělesem C nebo odečtení tělesa B. Proces *prořezávání* (*tree pruning*) neodstraňuje pouze jednotlivá primitiva, ale způsobuje zásadní zjednodušení původního CSG stromu.



Obrázek 6.12: Prořezávání CSG stromu umístěného do rozděleného prostoru (zjednodušený pohled v rovině)

6.4 Modelování pomocí deformací

Při modelování (tvarování těles) vychází konstruktér z omezené nabídky primitivních ploch, těles a operací a pomocí nich se snaží realizovat vlastní představy o tvaru cílového produktu. Představme si, že konstruktér navrhuje plastové pouzdro ručního mixéru, ve kterém je uložen elektrický motor a převodní mechanismus. Pro technicky vyhovující řešení lze pouzdro vytvořit pomocí dutého válce s tenkými válcovými víčky a doplnit držadlem tvořeným částí toroidní plochy. Z estetických a hlavně ergonomických důvodů je však nutné základní tvar „opracovat“ a např. držadlo mixéru přizpůsobit pomocí vlysů pro snadné uchopení rukou.

Při použití volně tvarovatelných plátů (Bézierovy, NURBS plochy aj.) lze vytvarovat velmi členité povrchy. Skládání složitějšího povrchu z malých, samostatně tvarovaných částí je však velmi náročné, nepřehledné a při opakovaných pracovních postupech nepoužitelné. Proto byly hledány a vyvíjeny metody dodatečného tvarování, které jsou obvykle označovány jako *deformace*. Při jejich výběru je nutné zvážit, zda deformace povede ke změně celého tvaru tělesa, nebo zda bude aplikována lokálně, pouze na určitou prostorovou nebo plošnou oblast tělesa. První skupinu metod označujeme jako *globální deformace*, druhou *lokální deformace*.

6.4.1 Barrovy deformace

Prvním pokus o dodatečné tvarování výchozího modelu tělesa představují *Barrovy globální deformace* [Barr84]. Jedná se o transformace, které jsou aplikovány na hotový prostorový model objektu a ovlivňují změnu tvaru tohoto objektu v celém jeho rozsahu.

Jednoduchou deformací, která ovlivní tvar celého tělesa, je *neuniformní změna měřítek* na jednotlivých osách modelových souřadnic. Transformace, která je obecně nelineární, neboť deformuje prostor lokálních souřadnic objektu a nelze ji lineárně skládat s jinými transformacemi, má tvar:

$$X = F_x(x) \ , \ Y = F_y(y) \ , \ Z \ = F_z(z) \ ,$$

kde $[x, y, z]$ je bod nedeformovaného tělesa a $[X, Y, Z]$ je deformovaný bod. Dále uvedeme výběr několika transformací toho druhu, které realizují základní deformace modelu:

1. *Deformace změnou měřítek* je definována transformací

   $$X = S_x(x) = s_x.x \ , \ Y = S_y(y) = s_y.y \ , \ Z \ = S_z(z) = s_z z,$$

   kde s_x, s_y, s_z jsou odpovídající měřítka stlačení/natáhnutí tělesa ve směru osy souřadnicového systému.

2. *Deformace zeslabování, zašpičatění* (*tapering*) je odvozena ze změny měřítka. Zvolíme osu zeslabování a plynule měníme měřítko na zbývajících osách. Např. pro zeslabení objektu ve směru osy z použijeme transformace

   $$X = r_x.x \ , \ Y = r_y.y \ , \ Z \ = z \ ,$$

   kde $r_x = f(z), r_y = g(z)$ jsou lineární nebo nelineární zeslabovací funkce.

3. *Zkroucení* (*twisting*) podle zvolené osy lze realizovat jako diferenciální rotaci, obdobně jako v případě zeslabování. Zkroucení okolo osy z realizuje transformace

   $$\begin{aligned} X &= x\cos(f(z)) - x\sin(f(z)) \\ Y &= x\sin(f(z)) + y\cos(f(z)) \\ Z &= z. \end{aligned}$$

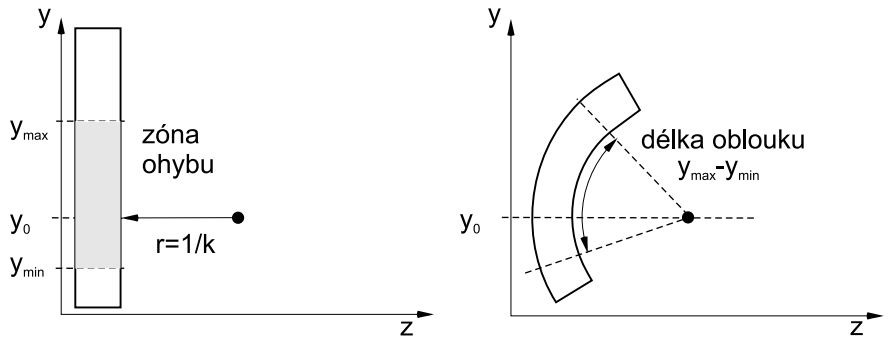
4. *Ohýbání* (*bending*) je složená transformace, která rozlišuje ohýbací zónu a vnější zóny tělesa. Ve vnějších zónách dochází k posuvu a natočení, v zóně ohybu je aplikována kruhová deformace. Jako příklad uvedeme vztahy pro ohýbání objektu kolem osy rovnoběžné

s osou x (viz obrázek 6.13). Parametry ohnutí jsou vymezeny ve směru osy y. Ohýbací zóna nedeformovaného tělesa je definována v intervalu

$$y_{min} \leq y \leq y_{max} \ ,$$

poloměr ohybu je $1/k$ a střed ohybu je v bodě $y = y_0$. Úhel ohybu θ je

$$\theta = k\ (y' - y_0) \quad , \quad y' = \begin{cases} y_{min} & y < y_{min} \\ y & y_{min} \leq y \leq y_{max} \\ y_{max} & y > y_{max} \end{cases}.$$

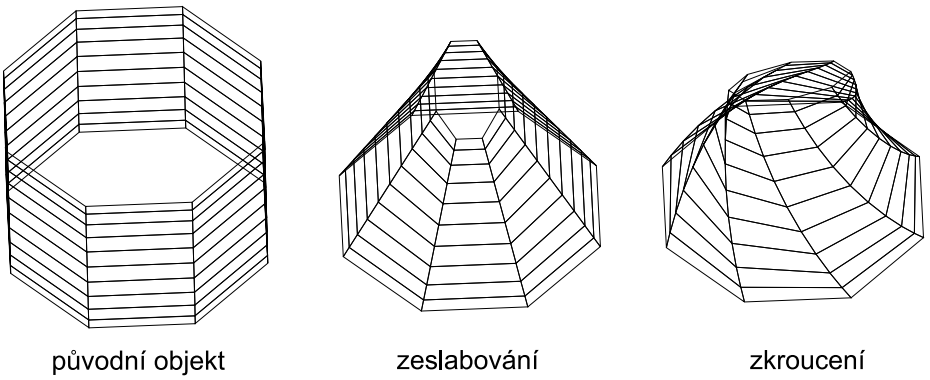


Obrázek 6.13: Parametry ohýbání tělesa

Deformační transformace je určena vztahy

$$\begin{aligned} X &= x \\ Y &= \begin{cases} -sin(\theta)(z\ -1/k) + y_0 + cos(\theta)(y - y_{min}) & y < y_{min} \\ -sin(\theta)(z\ -1/k) + y_0 & y_{min} \leq y \leq y_{max} \\ -sin(\theta)(z\ -1/k) + y_0 + cos(\theta)(y - y_{max}) & y > y_{max} \end{cases} \\ Z &= \begin{cases} cos(\theta)(z\ -1/k) + 1/k + sin(\theta)(y - y_{min}) & y < y_{min} \\ cos(\theta)(z\ -1/k) + 1/k & y_{min} \leq y \leq y_{max} \\ cos(\theta)(z\ -1/k) + 1/k + sin(\theta)(y - y_{max}) & y > y_{max} \end{cases} \end{aligned}.$$

Příklady globálních deformací jsou na obr.6.14. Globální nelineární deformace nelze obecně aplikovat na všechny druhy modelů. Deformace nesmí porušit (překročit) omezení pro daný typ modelu. Např. kvádr vyjádřený v hraničním modelu jako seznam ploch, hran a vrcholů, není



Obrázek 6.14: Globální deformace podle [Barr84]

možné libovolně deformovat zkroucením. Použitá deformace nesmí porušit topologii výchozího modelu, v opačném případě by nebylo možné použít další algoritmy, které např. vykreslují těleso s respektováním viditelnosti.

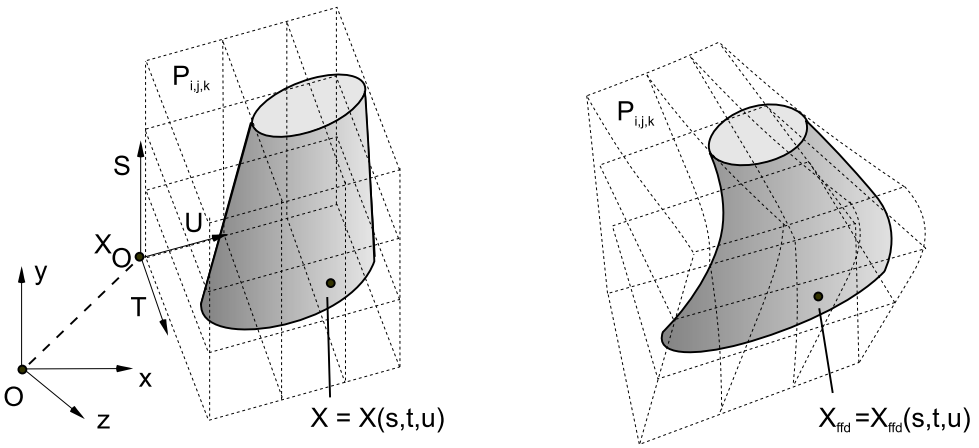
6.4.2 Volné tvarování těles

V roce 1986 publikovali Sederberg a Parry [Sede86] metodu nazvanou *Free–Form Deformation*, pro kterou použijeme v textu přibližný název *volné tvarování těles*, případně vžitou zkratku FFD. Metoda FFD je všestranným nástrojem. Může být aplikována na modely CSG stejně dobře jako na modely popsané hraniční reprezentací. Může tvarovat tělesa omezená libovolnými analytickými povrchy: rovinami, kvadrikami, parametrickými povrchy plošek nebo implicitními povrchy. Aplikace FFD nejsou omezeny jen na modely těles, ale deformovat lze také plochy a části ploch v prostoru.

Formulace FFD

Fyzikální analogie FFD vychází z představy, že objekt z elastického materiálu vložíme do formy ve tvaru hranolu, a prázdný prostor ve formě vyplníme (zalijeme) materiálem, který je po ztuhnutí průhledný a pružný. Pak aplikujeme vybranou sadu transformací, které jednoduše mění tvar plastového hranolu. Při deformaci hranolu se současně deformuje i v něm uložený objekt.

Tuto představu napodobuje následující výpočetní postup. Ve světovém souřadnicovém systému O, X, Y, Z nejprve zavedeme pomocný souřadnicový systém s počátkem v bodě X_O,



Obrázek 6.15: Těleso v deformačním kvádru – souřadnicové systémy O, X, Y, Z a (X_O, S, T, U)

jehož bázi tvoří trojice lineárně nezávislých vektorů S, T, U, které svojí orientací a velikostí vymezují rovnoběžnostěn. Tyto vektory nemusí být vzájemně kolmé. Rovnoběžnostěn obsahuje objekt nebo část objektu (prostoru), která bude podrobena deformaci. Každý bod X se světovými souřadnicemi $[x, y, z]$ má v lokálním souřadnicovém systému souřadnice $[s, t, u]$ (obr. 6.15) a platí

$$X = X_O + s.S + t.T + u.U.$$

Před aplikací deformace známe světové souřadnice bodu X a potřebujeme určit, jaké jsou jeho lokální souřadnice v souřadnicovém systému X_O, S, T, U. Získáme je řešením rovnic. Bod má v lokálním souřadnicovém systému

$$s = \frac{T \times U \ (X - X_O)}{T \times U \cdot S}, t = \frac{S \times U \ (X - X_O)}{S \times U \cdot T}, u = \frac{S \times T (X - X_O)}{S \times T \cdot U}.$$

Pro libovolný vnitřní bod $[s, t, u]$ rovnoběžnostěnu je $0 < s, t, u < 1$. Body, které tuto podmínku nesplňují, leží vně deformované části prostoru. V prostoru rovnoběžnostěnu vytvoříme pravidelnou prostorovou mřížku řídicích bodů P_{ijk}. Body mřížky $P_{i,j,k}$ jsou určeny jako průsečíky $l + 1$ rovin ve směru S, $m + 1$ rovin ve směru T a $n + 1$ rovin ve směru U a jsou indexovány v rozsahu $i = 0, \ldots, l; j = 0, \ldots, m; k = 0, \ldots, n$. Při rozložení rovin ve shodných

vzdálenostech od sebe je umístění řídicích bodů $P_{i,j,k}$ definováno jako

$$P_{i,j,k} = X_0 + \frac{i}{l}S \ + \frac{j}{m}T + \frac{k}{n}U.$$

Původně pravidelný rovnoběžnostěn pravidelně rozdělený sítí řídicích bodů na jednotlivé podprostory zdeformujeme přemístěním bodů $P_{i,j,k}$ dělicí mřížky. Změněnou pozici X_{ffd} libovolného vnitřního bodu X tohoto objemu nalezneme tak, že nejprve vypočteme lokální souřadnice $[s,t,u]$ bodu a poté zjistíme polohu tohoto bodu ve světových souřadnicích, do které se přemístí při deformaci objemu. Je zřejmé, že novou pozici bodu ovlivní jednotlivé řídicí body $P_{i,j,k}$ s určitou váhou. Nová pozice může být určena např. lineární interpolací pomocí nejbližších řídicích bodů, s ohledem na požadovanou spojitost (hladkost) deformovaného objektu je však vhodné volit takové váhy, které zaručí požadovaný stupeň hladkosti povrchu. Víme, že při tvorbě křivek a ploch spojitost zaručují různé polynomiální báze. Sederberg a Parry zvolili „Bézierovskou deformaci prostoru“ pomocí Bernsteinových polynomů:

$$X_{ffd}(s,t,u) = \sum_{i=0}^{l} \binom{l}{i} (1-s)^{l-i} s^i \left[\sum_{j=0}^{m} \binom{m}{j} (1-t)^{m-j} t^j \left[\sum_{k=0}^{n} \binom{m}{k} (1-u)^{n-k} u^k P_{i,j,k} \right] \right].$$

$X_{ffd}(s,t,u)$ v kartézských souřadnicích udává polohu přemístěného bodu, jehož lokální souřadnice $[s,t,u]$ určují hodnoty příslušných Bernsteinových polynomů použitých jako váhy řídicích bodů, a $P^{i,j,k}$ jsou (přemístěné) řídicí body v kartézských souřadnicích. Stejně jako v případě Bézierových křivek a plátů jsou u této metody zřejmé vztahy mezi rozmístěním řídicích bodů a charakterem deformace.

Velmi důležitou charakteristikou FFD je skutečnost, že deformovaný parametrický povrch zůstane parametrickým povrchem. Pokud je parametrický povrch dán vztahy $x = f(u,v), y = g(u,v), z \ = h(u,v)$ a FFD je dáno $X_{ffd} = X(x,y,z)$, pak deformovaný parametrický povrch plošky je dán $X_{ffd}(u,v) = X(f(u,v), g(u,v), h(u,v))$.

Zajímavým důsledkem je, že parametrické křivky zůstanou parametrické i po aplikaci FFD. Tento fakt naznačuje důležité možnosti pro modelování těles. Kvadriky a roviny tvoří velmi dobrý základ pro FFD, protože pro práci s nimi jsou k dispozici současně implicitní a parametrické rovnice. Parametrické rovnice umožňují rychlé vyčíslení bodů na povrchu a implicitní rovnice umožňují jednoduchou klasifikaci bodů – bod je uvnitř, vně nebo na povrchu. Při klasifikaci bodů vzhledem k deformované kvadrice (např. průsečíku prostorové přímky s parametrickým plátem při řešení metody sledování paprsku, viz část 15.9) nejprve nalezneme souřadnice tohoto bodu v souřadném systému S,T,U a poté dosadíme do implicitní rovnice. Souřadnice $[s,t,u]$ mohou být např. nalezeny postupným dělením obalového tělesa (konvexního obalu řídicích bodů) nebo pomocí newtonské iterace.

Aplikace volných deformací

Deformace FFD je možné aplikovat postupně, obdobně jako při tvorbě po částech polynomiálních křivek a ploch. Hlavním aspektem, který sledujeme při lokálně omezené deformaci, je udržení předepsané spojitosti navazujících deformovaných a nedeformovaných oblastí. V [Sede86] je dokázáno, že podmínky spojitosti jsou přímým rozšířením podmínek spojitosti při navazování Bézierových křivek a Bézierových ploch.

Všestrannost metody může být shrnuta následovně. FFD lze použít s libovolným schématem modelování těles, pracuje s povrchy různého vyjádření a stupně. Může být aplikována lokálně nebo globálně a s požadovanou spojitostí. Poskytuje možnost kontroly změny objemu po deformaci a existují třídy FFD, které zachovají objem i deformovaného objektu. Volné deformace lze použít pro tvorbu estetických povrchů.

Samozřejmě i techniky FFD mají svá omezení. Některé operace, které se běžně používají při práci s CAD systémy, nelze použít. Jedná se např. o dotahování ploch na doraz k jiným plochám, seseknutí nebo zaoblování ostrých hran, nebo o svařování. Při lokální deformaci metoda FFD vymezuje rozhraní mezi deformovanou a nedeformovanou oblastí a toto rozhraní leží v rovině. Pokud má přiléhající povrch, na který navazujeme defomovanou část, složitější tvar okraje, není použití FFD pro přesné napojení výhodné. Složitější hraniční křivku lze sice vytvořit pomocí opakované aplikace FFD, jedná se však o velmi nákladnou operaci. V takových případech je vhodné použít jiné modelovací postupy, např. navázání povrchů pomocí plátování a hladkého spojování (proložení plochy okrajovými křivkami, viz část 5.8).

Kapitola 7

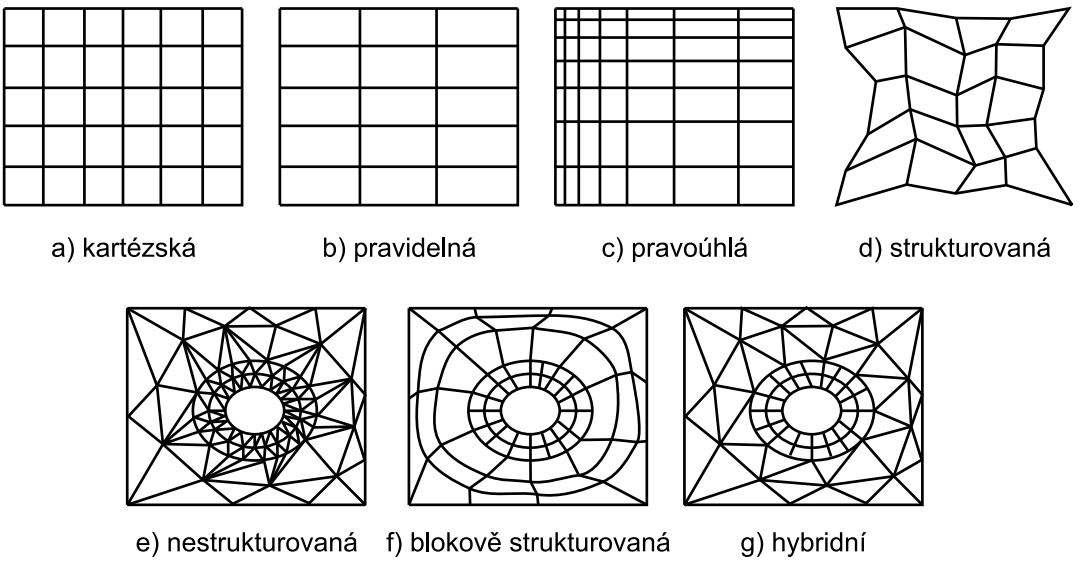
Objemová reprezentace těles

V řadě aplikací nemáme k dispozici geometrický popis těles, ale pouze sadu vzorků v určitém místě povrchu či objemu. Tato sada může obsahovat *rozptýlená data*, kde každý vzorek z množiny dat nese kromě údajů o jasu, hustotě či jiné fyzikální veličině v daném bodě také své souřadnice. V jiných aplikacích může být strukturovaná do podoby pravidelných nebo nepravidelných *mřížek*. Rozmístění vzorků do pravidelné pravoúhlé mřížky je typické například pro data získaná pomocí počítačových tomografů, nepravidelný tvar mají data získaná při simulaci proudění kapalin a rozptýlená data jsou výsledkem meteorologických měření teploty a tlaku vzduchu. V dalším textu se soustředíme převážně na data uložená v prostorových mřížkách.

7.1 Mřížky

U dat uspořádaných do mřížky je důležitý *tvar mřížky (domény)*. Dělení geometrického tvaru mřížek navrhli Speray a Kennon [Sper90] (viz též [Watt92]), kteří rozdělili obvyklé tvary do sedmi tříd. Příklady jednotlivých typů mřížek jsou uvedeny na obrázku 7.1, kde vidíme, že první čtyři případy pracují se vzorky, které jsou topologicky uspořádané do tvaru mřížky, geometrický tvar mřížky je různým způsobem zdeformován. Příkladem takové deformace jsou například mřížky vytvořené v polárních nebo válcových souřadnicových systémech. Nestrukturovaná mřížka má topologii uloženou v dalším poli – poli buněk. Každá buňka obsahuje odkazy na vrcholy, ze kterých je utvořena. Zbylé dva typy jsou kombinací ostatních.

Mřížka může mít libovolný počet rozměrů m, který se označuje jako *dimenzionalita domény*. V dalším textu se budeme zabývat zpracováním trojrozměrných mřížek. *Typ vzorků* informuje o počtu hodnot ve vzorku, tj. např. jedna hodnota – skalár, pole hodnot – vektor, matice



Obrázek 7.1: Příklady různých datových mřížek

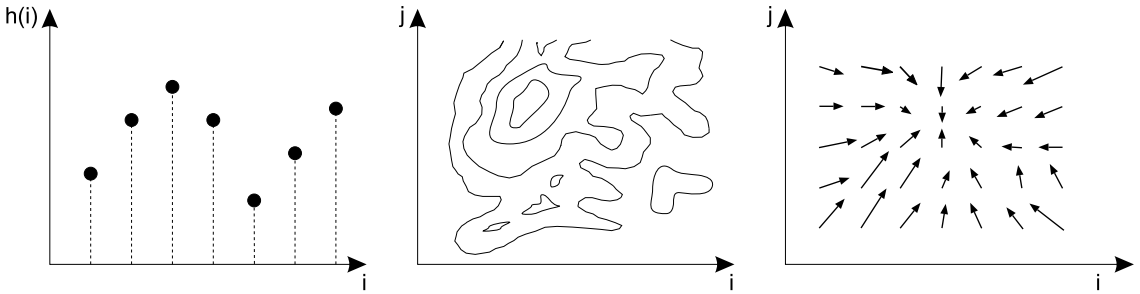
hodnot – tenzor, a o jejich základním datovém typu, tj. bit, byte, word, float, double, atd. Podrobnou klasifikaci dle těchto kritérií lze nalézt např. v [Brod92]. Příklad je na obrázku 7.2.

7.2 Trojrozměrné objekty a data v diskrétní mřížce

Jedním z problémů diskrétních objemových dat je, že je jich velké množství. Tato data kladou velké nároky na kapacitu paměti a výkon procesoru. Vzhledem k tomu, že místo spojité informace uchováváme jen diskrétní vzorky, se objemová data obtížně natáčejí o úhly jiné než pravé, tato data se obtížně zvětšují či zmenšují, převzorkováním ztrácíme informaci o příslušnosti buňky k objektu, aj.

Objemová data však mají některé výhody, které spojitě reprezentované objekty nemají. Jde zejména o snadnou práci s naměřenými daty, snadné provádění blokových a logických operací, zpracování objemu dat jako celku a nezávislost na složitosti scény (počtu objektů ve scéně) i na náročnosti generování komplikovaných objektů. Pro zobrazování libovolným způsobem definovaných těles (vzorky, rasterizovaná geometrie, procedurální textura, fraktál, apod.) postačuje jeden algoritmus.

Pokud jde o vícerozměrná data a data s neskalárními vzorky (vektor, tenzor), není jejich ukládání žádným velkým problémem. Problémem je spíše orientace v datech a metody zob-



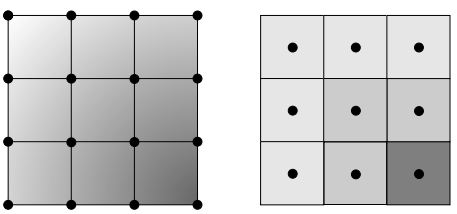
Obrázek 7.2: Příklady různých dimenzí domén a rozměrů dat: (zleva) graf jednorozměrných skalárních dat, izočáry skalárních dat ve dvou rozměrech, vektorová data ve dvou rozměrech.

razování a analýzy. V následujícím textu se zaměříme na skalární data ve tvaru pravidelné prostorové mřížky.

7.2.1 Základní objemové elementy – voxel a buňka

Pojem *voxel*, který vznikl jako analogie dvourozměrného pixelu, označuje nejmenší element ve trojrozměrném diskrétním prostoru. Voxely mají tvar kvádru nebo krychle a jsou uspořádány do pravoúhlé mřížky.

Voxel je vyplněný kvádr, který má v celém svém objemu konstantní hodnotu (obrázek 7.3 vpravo). Jako hodnota v poloze mezi vzorky (středy voxelů) se obvykle bere hodnota voxelu, který je nejblíže. Tento způsob „odhadu“ hodnot mezi souřadnicemi vzorků se nazývá interpolací hodnotou nejbližšího souseda či interpolací 0. řádu (viz též kap. 22.6).



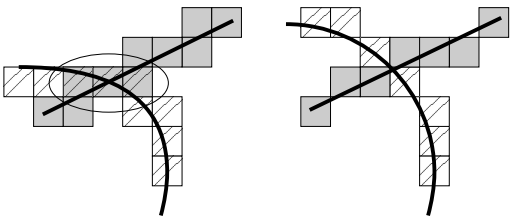
Obrázek 7.3: Základní objemové elementy: buňky a plné krychličky

Takové vzorkování je pro řadu algoritmů příliš hrubé, proto se hodnoty v uzlech mřížky častěji chápou jako bodové vzorky spojitého prostoru a osmice vzorků vytváří jednu *buňku*, *cell* (viz obr. 7.3 vlevo). Prostorová buňka může mít různý tvar – od čtyřstěnu přes kostičku po n-stěn. Hodnoty uvnitř buňky se obvykle vypočítávají lineární interpolací (1. řádu) či řidčeji interpolací kvadratickou (2. řádu).

V případě, že při popisu algoritmu nebude záležet na konkrétním způsobu interpolace hodnot, budeme v dalším textu část prostoru označovat jako *objemový element*.

7.2.2 Digitální topologie a spojitost

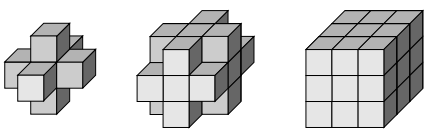
Obdobně jako ve spojitém geometrickém prostoru, by měla rovina nebo obecnější povrch vzorkovaného objektu ve třech rozměrech, rozdělit prostor na dva neprotínající se poloprostory. Nicméně souvislost oblasti a vzájemné oddělení částí prostoru je v diskrétním prostoru složitější. Situaci si vysvětlíme na příkladu přímky. Ve dvourozměrném prostoru můžeme považovat za nepřerušenou buď přímku, jejíž pixely se dotýkají celou hranou (takové sousedy má pixel čtyři a proto se nazývá *4spojitá*), nebo přímku, u níž stačí, aby se pixely dotýkaly jen v rozích (takových sousedů má pixel osm = čtyři hrany + čtyři vrcholy, a proto je přímka *8spojitá*).



Obrázek 7.4: Průsečík 4 a 8spojitých objektů

Obrázek 7.4 ilustruje na úrovni pixelů situaci při stanovování průsečíku dvou útvarů v diskrétním prostoru. U dvojice 4spojitých objektů se průsečík detekuje snadno podle existence společných pixelů. Obtížnější je však určení jeho polohy, neboť společných pixelů může být více (na obrázku 7.4 vlevo jsou označeny oválem) a poloha se musí odhadnout např. proložením přímky či křivky. U osmispojitých je obtížná i samotná detekce existence průsečíku, neboť společné pixely nemusejí existovat (viz obr. 7.4 vpravo). V trojrozměrném prostoru je situace o jednu úroveň složitější. Možnosti jsou tři (viz obr. 7.5):

- *6-ti spojitost* – dotyk povolen pouze celou stěnou voxelu,
- *18-ti spojitost* – dotyk povolen stěnou nebo hranou (6+12),
- *26-ti spojitost* – dotyk stěnou, hranou nebo vrcholem voxelu (6+12+8).



Obrázek 7.5: 6-okolí, 18-okolí a 26-okolí voxelu

Způsob, jakým jsou definovány diskrétní 3D objekty v objemu, je určující nejen při zjišťování vzájemných průsečíků objektů, ale i pro definici spojitosti zobrazovacího paprsku. Proto, pokud víme, že objekt je 18ti nebo 26spojitý, musíme použít 6spojitý paprsek, aby nedocházelo k chybné detekci povrchů. Naopak pro 6spojitý objekt (ten díry obsahovat nemůže) lze použít paprsek 18 či 26spojitý. Spojitost paprsku má podstatný vliv též na jeho délku, tj. na počet kroků, které musí při průchodu scénou vykonat, a tedy i na čas, který potřebujeme pro příslušné výpočty.

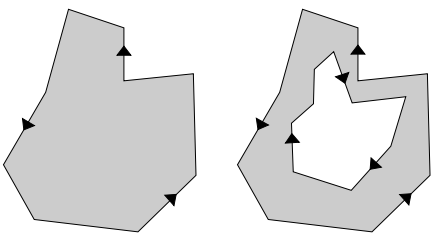
7.3 Nalezení povrchu v objemových datech

Některé algoritmy převádějí objemová data do povrchové reprezentace. Velmi častým postupem je aproximace povrchů sítí trojúhelníků. Povrchy se předpokládají obvykle v místech s konstantní hodnotou vzorků. Proto se takto vzniklé sítě nazývají *izoplochy* (srovnej kapitolu 5.12), obdobně jako *izočáry* ve dvou rozměrech.

V praxi se setkáme s následujícími úlohami. Měřením nebo jiným způsobem zpracování byla získána sada rovinných řezů části prostoru, ve kterých jsou určeny obrysové křivky – např. izočáry. Úkolem je vytvořit opláštění, které propojí obrysové křivky pomocí trojúhelníkové sítě. Tomuto tématu se věnuje část 7.3.1. Další třída algoritmů řeší situaci, kdy je dána prostorová mřížka s naměřenými hodnotami a v takto uspořádaných datech je postupným prohledáváním vytvořena trojúhelníková síť. Nejznámějším a nejpoužívanějším algoritmem je metoda *pochodující kostky*, viz část 7.3.2.

7.3.1 Sada obrysů v rovnoběžných řezech

Možným způsobem zaznamenání hranice tělesa je popis pomocí šablon či kontur. *Kontura* (u složitějších těles s více větvemi a dírami množina kontur) je průnikem hranice tělesa s rovinou. Příklad je na obrázku 7.6.



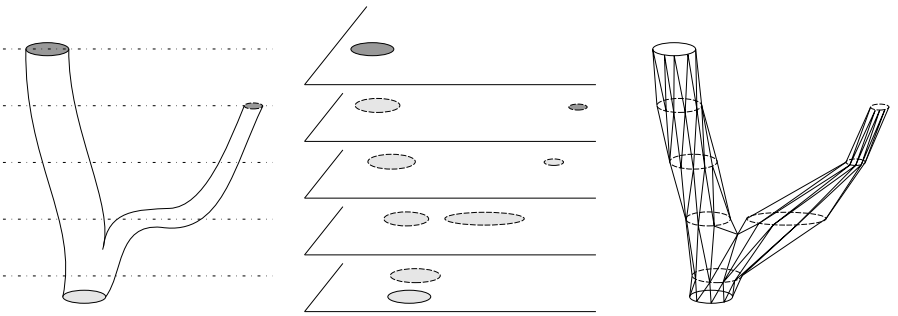
Obrázek 7.6: Kontura s orientovanou hranicí a kontura s dírou

K popisu těles se používá sada kontur, tj. množina průniků s více rovnoběžnými rovinami. Tato reprezentace vychází z praxe – např. v lodním či leteckém průmyslu se při návrhu vnějších ploch používají šablony, v lékařství a biologii se na preparátech – mechanických či optických řezech zkoumanou tkání – vyznačí relativně snadno obrysy zajímavých útvarů. Kontura je výstupem řady algoritmů při počítačovém zpracování obrazu. Kontury lze reprezentovat *binárním obrazem*, kde pixely s hodnotou logické jedničky reprezentují body na kontuře (vzorky) a pixely s nulou vše ostatní. Kompaktnější reprezentace ukládá obrys ve formě *rovinné křivky*, nejčastěji se používá aproximace obrysu *lomenou čárou*.

Rekonstrukce povrchu opláštěním kontur

Vstupem algoritmů je množina mnohoúhelníkových kontur, výstupem je odhad původního tvaru povrchu nejčastěji ve formě trojúhelníkové sítě (příklad je na obrázku 7.7). Jde tedy o interpolaci křivek (aproximovaných lomenou čarou) plochou, která je aproximována sítí trojúhelníků. Konstrukce pláště z kontur není v obecném případě triviální úlohou. Při vzorkování objektu v řezech dochází ke ztrátě informací o objemu mezi řezy, které při rekonstrukci chybějí. Proto je výsledkem rekonstrukce jen hrubá aproximace povrchu. Na obrázku 7.7

vlevo je příklad vzorkovaného objektu, uprostřed tvar kontur v řezech a napravo rekonstrukce objektu opláštěním trojúhelníkovou sítí. V místech, kde je směr osy objektu kolmý na rovinu řezu, je rekonstrukce nejpřesnější, s rostoucím vzájemným sklonem klesá relativní hustota vzorků (přesnost vzorkování) a zvyšuje se zkreslení tvaru. S rostoucím sklonem osy objektu od roviny řezu se stává obtížnějším i rozpoznání vzájemné korespondence kontur, které je podmínkou správného opláštění. Proto se tento problém řeší heuristikami, tj. v daném kontextu



Obrázek 7.7: Prostorový objekt, kontury v rovnoběžných řezech a rekonstrukce povrchu opláštěním sítí trojúhelníků

vyzkoušenými postupy, nebo interakcí s uživatelem. Podrobnější popis heuristik je uveden např. v [Soro81] a [Meye92]. Výsledkem této fáze je obvykle i lokalizace míst, kde dochází k větvení, a informace o typu větvení, která následně ovlivňuje volbu algoritmu oplášťování.

Metody opláštění lze podle použitého principu rozdělit na objemové a povrchové. *Objemové metody* předpokládají řezy tak blízko sebe, že je krok mezi nimi srovnatelný s hustotou vzorků v řezu. Vzorky v řezech chápou jako prostorovou mřížku a po vyplnění kontur použijí standardní algoritmus na hledání izoplochy. *Povrchové metody* pracují přímo s řezy a interpolují postupně dvojice sousedních kontur pásy trojúhelníků. Jako výchozí prameny lze doporučit přehledové texty [Ekou91, Meye92, Oliv95] a zejména [Kepp75].

7.3.2 Převod izoplochy na síť trojúhelníků

Pochodující kostky

Lorensen a Cline publikovali algoritmus *pochodující kostky* (*Marching cubes*) v roce 1987 [Lore87]. Použili jej k nalezení izoplochy v tomografických datech a k jejímu pokrytí sítí trojúhelníků. Získanou síť zobrazovali klasickými metodami počítačové grafiky.

Vstupem algoritmu je objem dat (prostorová pravoúhlá mřížka) a konstanta pro práh.

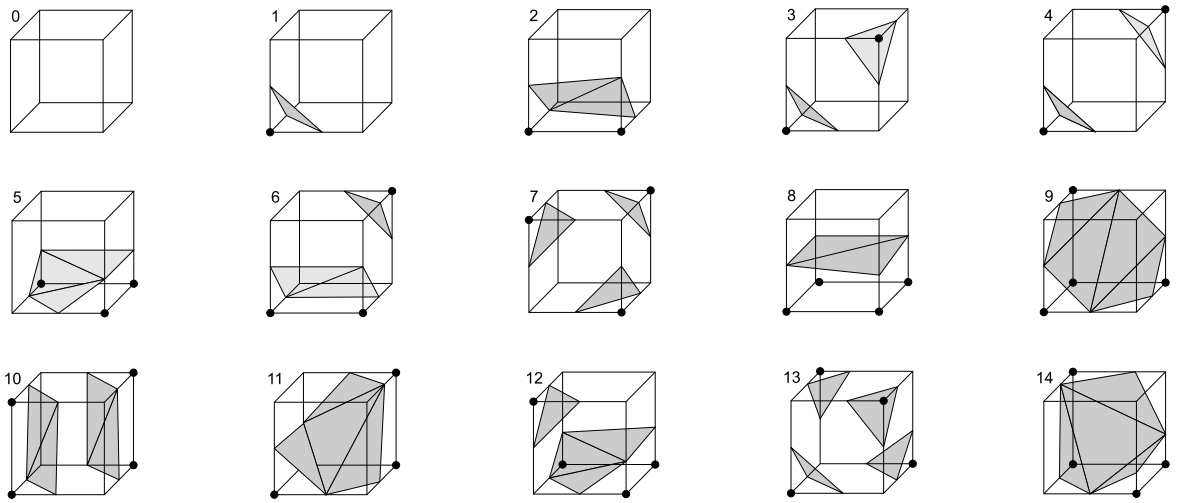
Výstupem je síť trojúhelníků. Sled kroků pro jednu buňku objemu popisuje algoritmus 7.1. Jednotlivé kroky probereme podrobněji.

1. Sestavení krychle
2. Ohodnocení vrcholů (uvnitř/vně)
3. Sestavení indexu do tabulky případů
4. Nalezení seznamu hran, které plocha protíná, v tabulce
5. Interpolace souřadnic vrcholů trojúhelníků na hranách
6. Výpočet normál ve vrcholech trojúhelníků

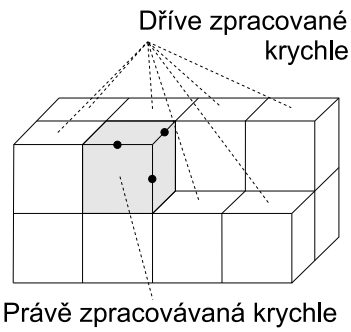
Algoritmus 7.1: Pochodující kostky – sled operací pro jednu buňku

1. Sestavení krychle. Vstupem algoritmu je prostorová mřížka, v jejíchž vrcholech jsou uloženy hodnoty vzorků. Algoritmus nepracuje s celým objemem dat najednou, ale načítá vzorky postupně po řezech, tj. např. s konstantní souřadnicí z. Krychli tvoří vždy dvě čtveřice vzorků ze sousedních řezů.
2-4. Ohodnocení vrcholů, sestavení indexu a nalezení seznamu hran. Hodnoty ve vrcholech jsou porovnány se zadanou prahovou hodnotou a vrcholy ohodnoceny buď jako vnitřní (hodnota $<$ práh) nebo jako vnější (hodnota $\geq$ práh). Je-li všech osm vrcholů ohodnoceno stejně, leží krychle celá vně či celá uvnitř, a proto se dalšího zpracování neúčastní (dále viz bod 1.).

Vzniklých osm binárních hodnot použijeme jako osmibitový index do 256-řádkové tabulky. V této tabulce je pro každý index uložen seznam hran krychle, které musí vytvářená izoplocha protínat. tj. seznam hran, na nichž leží vrcholy generovaných trojúhelníků. Díky tomu, že krychle je značně symetrické těleso, je potřebných 256 případů vytvořeno natočením či inverzí (prohozením poloh vrcholů vně a uvnitř a tedy i invertováním nul a jedniček v indexu) patnácti základních případů (obr. 7.8).
5. Výpočet souřadnic vrcholů trojúhelníků. Přesnou polohu vrcholu trojúhelníku v rámci hrany získáme jednoduše lineární interpolací. Ve vzorci (22.37) v kapitole 22.6.2 dosadíme za hodnotu $h(x)$ hodnotu prahu. Umístěním vrcholu do poloviny hrany bychom získali hrubší odhad povrchu, kvadratickou interpolací (viz kap. 22.6) odhad přesnější. Mezním nejhorším případem by byla interpolace metodou nejbližšího souseda, kdy bychom umístili vrchol trojúhelníku do vrcholu, jehož hodnota je hodnotě prahu nejblíže.
6. Výpočet normál. Normály ve vrcholech trojúhelníků vypočítáme lineární interpolací složek normál ve vrcholech krychle. Směr normál ve vrcholech krychle musíme odhadnout.



Obrázek 7.8: Základní případy konfigurace vrcholů krychle u algoritmu *pochodující kostky* a způsob, jakým povrch objektu protíná krychli. Tečky odpovídají vrcholům uvnitř, neoznačené vrcholy vrcholům vně tělesa.



Obrázek 7.9: Návaznosti krychlí

Metoda odhadu vychází z úvahy, že povrch se nachází mezi dvěma objemy lišícími se svou hodnotou (nazývanou též denzitou) a že směr normály je totožný se směrem vektoru gradientu dat $\nabla f = (g_0, g_1, g_2)$. Protože neznáme přesný průběh spojité funkce $f(X)$, jejíž vzorky máme k dispozici, aproximujeme gradient symetrickou diferencí hodnot vzorků $h(X)$ v bodech mřížky [Höhn86, Inte93].

Algoritmus postupně prochází celým prostorem buněk a pro každou krychli provádí výše uvedené kroky. Pokud využijeme toho, že každý nově interpolovaný vrchol sítě je společný čtyřem krychlím, lze počet výpočtů značně snížit (viz [Watt92]). Využitím již jednou vypočtených hodnot pak místo osmi interpolací v jednom kroku počítáme jenom tři. Tuto situaci zachycuje obrázek 7.9, kde jsou nově počítané body označeny černou tečkou.

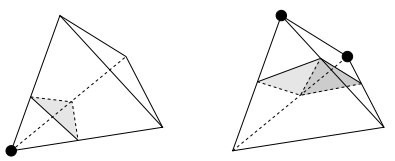
Výstupem algoritmu je izoplocha ve formě sítě trojúhelníků, která se dá zobrazovat klasickými metodami počítačové grafiky s využitím existujících grafických akcelerátorů. Získaných

trojúhelníků bývá nicméně velké množství, neboť pro každou krychli mohou vzniknout až čtyři. Například při zpracování tomografických dat v rozlišení $1024 \times 1024 \times 100$ vzorků obdržíme řádově až 2 miliony trojúhelníků. Druhou slabinou algoritmu je, že se způsob triangulace povrchu určuje pouze na základě hodnot osmi vrcholů tvořících krychli a ne globálně s ohledem na její okolí. Proto se vyskytují potenciálně nejednoznačné případy, kdy nemůžeme s jistotou říci, kudy hledaný povrch prochází.

Pochodující čtyřstěny

Pokusem o zamezení nejednoznačností při triangulaci povrchu, je algoritmus *pochodující čtyřstěny* (*Marching Tetrahedra*) [Payn90]. Místo umísťování vrcholů trojúhelníků na hranách krychliček se krychle rozdělí na pět čtyřstěnů a trojúhelníky se umísťují do nich (to znamená místo na dvanácti hranách na osmnácti). Dělení krychle na pět čtyřstěnů je možné dvěma způsoby, které je nutno z důvodu zachování návaznosti trojúhelníků sítě pravidelně střídat. Čtyřstěn je stejně jako krychle symetrický.

Po prozkoumání všech možností umístění trojúhelníků zjistíme, že vznikají ze dvou základních variant. Pro jeden vrchol uvnitř a ostatní tři vně vznikne jeden trojúhelník, pro dva vrcholy uvnitř a dva vně vyjdou dva trojúhelníky (viz obr. 7.10). Problém děr byl sice vyřešen, ale za cenu dalšího neúměrného nárůstu počtu trojúhelníků a nutnosti střídání způsobů dělení. Zlepšení návaznosti lze dosáhnout dělením na šest či 24 čtyřstěnů, nicméně pak už je počet trojúhelníků neúnosný. Proto se algoritmus *pochodující čtyřstěny* v praxi příliš neujal.



Obrázek 7.10: Algoritmus *pochodující čtyřstěny*: dva způsoby umístění trojúhelníku ve čtyřstěnu

Dělené kostky

Algoritmus *Dělené kostky* (*Dividing cubes*) navrhnul Cline [Clin88]. Problém pomalé rasterizace obrovského množství malých plošek vyřešil tím, že tento krok úplně odstranil a místo plošek generoval povrchové body s normálou (tzv. *smart points*), které mají takovou velikost, že se zobrazují do jednoho obrazového bodu. Při zpracování povrchových bodů úplně odpadá rasterizace, nutná např. pro trojúhelníky, a zrychlí se vykreslování tělesa. Díky normále, která je součástí každého takového bodu, lze vypočítat světelné podmínky jak při pohybu takto uloženého tělesa, tak při změně polohy a počtu světelných zdrojů. Nevýhodou této reprezentace je ztráta informace o tom, ke které ploše bod náleží, a nemožnost přiblížení tělesa (*zoom*), při němž přestane platit podmínka stejné velikosti pixelu a průmětu bodu a těleso začne být „děravé“.

1. Vytvoření buňky ze dvou čtveřic vzorků v sousedních vrstvách. Buňka má obvykle tvar kvádru s různými délkami hran x, y a z.
2. Výpočet gradientu ve všech osmi vrcholech buňky symetrickou diferencí (k tomu potřebujeme další dvě vrstvy).
3. Rozdělení buňky na $a \times b \times c$ malých krychliček o velikosti shodné s velikostí pixelu (např. $a = x\_cell/x\_pixel$). Hodnoty denzity v nově vzniklých vrcholech se dopočítají lineární interpolací.
4. Krychličky postupně procházíme a ohodnocujeme jejich vrcholy podle denzity na vnitřní a vnější. Na základě tohoto ohodnocení určíme, které krychličky povrch protíná a které jsou celé uvnitř či celé venku. Do dalšího zpracování postupují pouze povrchové krychličky, tj. takové, jejichž ohodnocení osmi vrcholů se liší.
5. Pro povrchové krychličky se interpolací spočítá vektor gradientu.
6. Výpočet intenzity osvětlení v místě krychličky.

Algoritmus 7.2: Algoritmus *Dělené kostky*

Algoritmus 7.2 pracuje v prostoru dat [Clin88] a dává stejně kvalitní výsledky jako algoritmus pochodující kostky, přičemž efektivně využívá možností rastrového displeje a negeneruje nezobrazitelné detaily.

Kapitola 8

Procedurální modelování

Trojrozměrný počítačový model nějakého objektu je možno získat v zásadě třemi způsoby. První možností je získání trojrozměrného *snímku* objektu, tedy hrubého modelu získaného přímo z geometrie reálného objektu. Ten můžeme pořídit za pomoci prostorového scaneru, nebo se můžeme pokusit o rekonstrukci z několika snímků z fotoaparátu. Tvorba modelu z fotografií je sice typická úloha počítačového vidění, avšak s nástupem digitálních fotoaparátů se začíná v počítačové grafice prosazovat stále častěji. Těmto postupům se říká *na obrazech založené modelování* (*image based modeling*). Trojrozměrné snímání tvaru objektů má samozřejmě svá omezení, která jsou dána jak velikostí snímaného objektu, tak jeho členitostí – je například problematické získání trojrozměrného snímku hlavy vlasatého člověka.

Druhým způsobem pořizování modelů objektů je jejich *interaktivní modelování*. Animátor (člověk) usedne k počítači a model vytvoří za pomoci myši, klávesnice, či dalšího vstupního zařízení. Tento postup je velice pracný, vyžaduje talent a zkušenosti, na druhou stranu tímto způsobem pořízená data jsou sémanticky naprosto přesná. Animátor totiž ví, co dělá, ví, co má v úmyslu modelovat a co která část modelu znamená. To je v případě scanování nesnadné.

Cílem této kapitoly je popsat získávání trojrozměrného modelu generováním pomocí nějakého algoritmu. *Procedurální modelování* (*procedural modeling*) [Eber03], jak se této metodě říká, je možno dále rozdělit na dvě základní podtřídy. První z nich jsou metody, používané v CAD a CAGD, jako například šablonování, generování ploch z křivek atp. Těmito algoritmy se v této kapitole nebudeme zabývat, i když si název procedurální modelování zajisté právem zaslouží. Za „ryzí“ procedurální modelování se totiž obyčejně považuje druhá podtřída, která se zaměřuje na automatické generování objektů, které se vizuálně či chováním podobají objektům, se kterými se setkáváme v přírodě. *Procedurální techniky jsou části kódu či algoritmy, které určují jisté charakteristiky počítačového modelu či efektu [Eber03].* Procedurální modelování, nebo chceme-li tuto jeho obecně uznávanou podčást, můžeme dále dělit na různé skupiny. První

z nich jsou algoritmy, které vycházejí z gramatik – sem patří především *L-systémy*, které se nejčastěji používají pro generování rostlin. *Fraktální geometrie*, která vymezuje druhou třídu, poskytuje algoritmy pro generování hor, krajin, kamenů, korálů, stromů, atd. *Systémy částic* určují další třídu a používají se především pro generování explozí, hejn ptáků, padajících míčů, simulaci ohně, dýmu atp.

Aby nebylo všem rozdělením konec, procedurální modely je možno rozdělit ještě podle jiného kritéria. První z nich jsou metody, které nejsou založené na simulaci, a poskytují vizuálně věrné výsledky. Jedná se buď o *ad-hoc* metody, nebo o metody, o kterých se domníváme, že modelují nějaké biologické, geologické atp. jevy, ale nevíme přesně jaké. Do této kategorie patří především fraktály. Druhou skupinou jsou metody založené na *simulaci* nějakého existujícího jevu, například modely rostlin se získávají simulací jejich růstu. Poslední uvedená třída procedurálních modelů je patrně nejzajímavější, neboť otevírá dveře do jiných vědních disciplín. Patří sem velká oblast modelování, které je založené na fyzice (*physics-based modeling*). Setkáme se s modely ekosystémů, erozí tekoucí vodou, modelováním hlíny atp.

8.1 Fraktální geometrie

Fraktální geometrie je vědní disciplína, která je intenzivně rozvíjena zhruba od šedesátých let dvacátého století. Za jejího objevitele je považován Benoit B. Mandelbrot, podle kterého se jmenuje jeden z nejkrásnějších nelineárních deterministických fraktálů tzv. Mandelbrotova množina, uvedena na obrázku 8.1. Algoritmus jejího generování je k nalezení například v [Peit92, strana 896].

Zatímco uměle vytvořené předměty se vesměs vyznačují geometrickou přesností, objekty v přírodě mají charakteristiku zcela odlišnou. Nasazení technik klasické geometrie při popisování přírodních tvarů je neefektivní. Je obtížné popsat mrak jako množinu koulí či parametrickou plochu, modelovat krajinu interaktivně jako množinu trojúhelníků atp. Fraktální geometrie poskytuje silný a relativně jednoduchý aparát, který takové modelování umožňuje. Z tohoto důvodu bývá fraktální geometrie někdy označována jako „morfologie amorfního“. Na rozdíl od Euklidovské geometrie, ve které obyčejně popisujeme objekty rovnicí, ve fraktální geometrii používáme k popisu objektů algoritmy, nejčastěji rekurzivní. Zatímco klasická geometrie popisuje invariance vzhledem k transformacím jako je otočení či posunutí, fraktální geometrie se zabývá invariancí ke změně měřítka.

8.1.1 Soběpodobnost

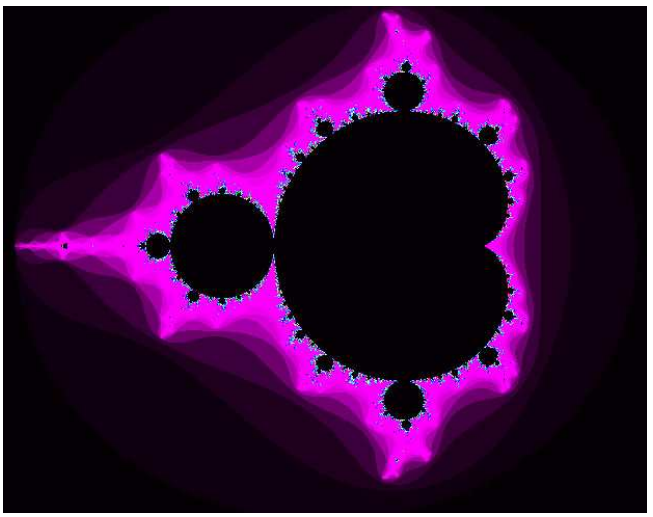
Ústřední pojem fraktální geometrie je *soběpodobnost* (*self-similarity*), což je jen jiný název pro invarianci ke změně měřítka. Soběpodobnou strukturu je možno rozložit na struktury, z nichž

každá je zmenšenou kopií originálu. Například čtverec lze složit z libovolného počtu menších čtverců, úsečku z menších úseček atd. Soběpodobnost je však pouze podmínkou nutnou, nikoliv postačují k fraktálnímu charakteru objektu. Než přistoupíme k vysvětlení dalších podmínek, uveďme přesnější definici soběpodobnosti.

Budeme rozlišovat dva druhy soběpodobnosti, soběpodobnost přesnou a statistickou [Peit92]. Množina A je *přesně soběpodobná*, pokud je sjednocením konečného počtu transformovaných kopií sebe samé

$$A = \bigcup_{i=1}^{n} \varphi_i(A). \qquad (8.1)$$

V tomto vztahu jsou transformace φ_i posunutí a rotace (21.2) a každá z nich je zároveň změnou měřítka (21.3.4) s koeficientem $S_i \in \langle 0, 1\rangle$, nebo jsou všchny tzv. průměrně kontraktivní. Pokud by součet koeficientů S_i přesáhl hodnotu jedné, množina by prostorově divergovala do nekonečna. Podmínka průměrné kontraktivity množiny transformací $\{\varphi_i, i = 1, 2, \ldots, n\}$ má tvar:



Obrázek 8.1: Mandelbrotova množina

$$0 < \sum_{i=1}^{n} S_i < 1.$$

Přesně soběpodobná je například Kochova[1] vločka (viz strana 272).

Množina A je *statisticky soběpodobná*, pokud je sjednocením konečného počtu zmenšených kopií sebe samé, podle vztahu (8.1) a každá z kopií $\varphi_i(A)$ má stejné statistické charakteristiky, jako množina A. Říkáme, že $\varphi_i(A)$ a A jsou statisticky nerozlišitelné. Transformace φ_i musí být zároveň změnou měřítka s koeficientem $s_i \in (0, 1)$. Jsou-li aplikované transformace lineární[2], resp. nelineární, je soběpodobná množina A lineární, resp. nelineární. Za zachování podmínky statistické soběpodobnosti v praxi obyčejně považujeme shodu směrodatné odchylky a průměru a ne všech statistických momentů.

[1] V prvním vydání této knihy autor této kapitoly ze jména Helge von Koch usoudil, že se jedná o ženu. Jaké bylo jeho překvapení, když z fotografie zjistil, že tato vědkyně měla vousy.

[2] Transformace $\varphi : U \to U$ je lineární, pokud $\varphi(r_1 A + r_2 B) = r_1\varphi(A) + r_2\varphi(B)$ pro všechna A,$B \in U$ a r_1,$r_2 \in R$. U je vektorový prostor a R je množina reálných čísel.

Příkladem statistické soběpodobnosti je kámen a hora. Pokud budeme porovnávat vhodně vybranou fotografii kamene a hory, bude pro nás obtížné rozhodnout, co je horou a co kamenem. Jiným příkladem je nahrávka šumu z rádia na frekvenci, kde není žádná stanice. Pokud takový šum nahrajeme, například na magnetofonový pásek, a budeme ho přehrávat libovolnou rychlostí, bude znít s největší pravděpodobností stejně. Změna rychlosti přehrávání odpovídá změně měřítka.

8.1.2 Fraktální dimenze, fraktál

Pro definici fraktálu je nutné alespoň stručně objasnit některé pojmy související s dimenzí a s mírou. *Topologickou dimenzi* budeme chápat intuitivně. Bod má topologickou dimenzi nula, přímka či spojitá křivka má dimenzi rovnu jedné, čtverec má topologickou dimenzi dvě, krychle má dimenzi tři atd. Geometrické objekty, které můžeme spojitou transformací převést na objekty výše uvedené, mají i stejnou topologickou dimenzi. V topologické dimenzi jedna je mírou délka, v dimenzi dva je mírou plocha a v dimenzi tři objem.

Základní vlastností fraktálů je, že v jejich topologické dimenzi je obtížné jejich měření. Příklad, uvedený v [Peit92], cituje dva zeměpisné atlasy. Ve španělské verzi je uvedena délka hranice mezi Španělskem a Portugalskem jako 978 km, v portugalské verzi je tatáž hranice dlouhá 1213 km. Problém není v nepřesnosti měření, problém je, že míra členitých objektů je svázána s přesností měření (délkou „měřící tyče“). Jinými slovy řečeno, pokud použijeme jako základní jednotku jeden metr získáme při měření velice členitých objektů výrazně jinou hodnotu výsledku, nežli při měření s přesností centimetrů.

Předpokládejme následující experiment. Vezměme nějaký velmi členitý objekt, například pobřeží ostrova a měřme jeho délku. Měřítko, které k tomuto měření použijeme označme ϵ. Proveďme měření a zjistíme, že k pokrytí hranice potřebujeme celkem N úseček délky ϵ a jejich součet je roven nějakému $K = N \cdot \epsilon$. Protože jsme některé zátoky a poloostrovy přeskočili, jedná se o aproximaci celkové délky. Proveďme další měření s jinou hodnotou ϵ. Uvidíme, že se zmenšujícím se ϵ roste hodnota K. Pro měřítko blížící se k nule se K blíží k nekonečnu. Aby K mělo smysluplnou hodnotu, je zapotřebí použít místo prostého vynásobení $K = N\epsilon$ vztah

$$K = N\epsilon^D. \tag{8.2}$$

V tomto vztahu je D tzv. *fraktální dimenze*[3]. Fraktální dimenze je pro nepříliš členité struktury rovna dimenzi topologické. Například na pokrytí úsečky délky $K = 1$ potřebujeme $N = 3$ úseček délky $\epsilon = 1/3$. Abychom dostali odpovídající míru, musíme ve vztahu (8.2) položit

[3]Hodnota D se původně nazývala podle svých objevitelů jako Hausdorffova či Hausdorff-Besicovitchova dimenze. Později se pro ni vžil pojem fraktální dimenze, i když fraktálních dimenzí je ve skutečnosti více. Tato se označuje jako soběpodobnostní fraktální dimenze [Peit92].

$D = 1$, a proto je fraktální dimenze rovna dimenzi topologické. Sledujme, jak se mění hodnota K v závislosti na D, budeme-li zkracovat délku úsečky ϵ

$$K = \lim_{\epsilon \to 0} N\epsilon^D.$$

Pro $D > 1$ je délka K nekonečná, je-li $D < 1$, je $K = 0$. Pouze pro $D = 1$ je $K = 1$ a jen v tomto případě D odpovídá tomu, co jsme intuitivně chápali jako topologickou dimenzi úsečky.

Problémy nastávají u výrazně členitých objektů. Existují struktury, pro které je $K = 1$ pouze pro D neceločíselná, případně celočíselná, ale vyšší, nežli je topologická dimenze těchto objektů. Příkladem je tzv. Peanova křivka, která vyplňuje plochu (*space-filling curve*). Pro tuto křivku je nutné položit $D = 2$. Zní to nelogicky, ale jedná se o křivku, která má fraktální dimenzi rovnu dvěma. Křivky, které neprotínají sebe samy, nemohou mít fraktální dimenzi vyšší nežli dvě. Spojité křivky nemohou mít fraktální dimenzi menší než jedna.

Velmi členité objekty nemůžeme měřit v tom, co chápeme jako jejich topologickou dimenzi. Například délka silně členitých objektů, které mají topologickou dimenzi jedna, je nekonečná. Plocha velice členitých objektů je nekonečná atd. Co však můžeme měřit, je míra jejich členitosti, a tu právě udává fraktální dimenze. Pokud má nějaký objekt fraktální dimenzi rovnou topologické a obě rovny jedné, jedná se topologicky o přímku. Zvyšující se fraktální dimenze nám říká, že objekt je stále více členitý. Fraktální dimenze rovna dvěma a topologická dimenze rovna jedné říká, že křivka vyplňuje plochu. Definice fraktálních útvarů se postupem času měnila podle toho, jak se objevovala stále nová fakta.

Vysoce členité objekty s výše uvedenými vlastnostmi se nazývají *fraktály*[4]. Dnes se za fraktál považuje množina, jejíž fraktální dimenze je ostře větší nežli její dimenze topologická [Mand82]. Existují však struktury, které mají charakteristiky fraktálů a přesto zmíněné podmínky nesplňují. Intuitivně můžeme považovat za fraktál jakýkoli výrazně členitý objekt, který je obtížné měřit. *Fraktál je členitý objekt, jehož struktura je dána opakováním určitého tvaru v různých velikostech [Eber03].* Shrňme, že fraktály v sobě spojují dvě základní vlastnosti; soběpodobnost a vysokou členitost.

Fraktál je matematicky popsán jako množina, která vznikne sloučením nekonečně mnoha kopií sebe samé. Musíme tedy rovnici (8.1) upravit na

$$A = \bigcup_{i=1}^{\infty} \varphi_i (A) . \tag{8.3}$$

Objekty, které pozorujeme v přírodě, zřejmě, či s největší pravděpodobností, nemají nekonečné detaily. Nejedná se tedy o fraktály ve smyslu rovnice (8.3). Přesto je obecně uznávanou dohodou

[4]Slovo fraktál vychází z latinského základu *fractus* (příčestí minulé od *frangere*), což znamená zlomit, rozdělit.

tyto objekty za fraktály považovat, i když by přesnější bylo používat označení jako fraktálům podobné, fraktálně strukturované atp.

Fraktál je soběpodobná množina. Podle druhu soběpodobnosti rozlišujeme fraktály *deterministické* (přesně soběpodobné) a *nedeterministické* (statisticky soběpodobné). Nedeterministickým fraktálům se také říká stochastické, nebo náhodné. Podle typu transformací použitých ve vztahu (8.3) fraktály dále členíme na *lineární* a *nelineární*. Celkem tedy máme k dispozici čtyři různé druhy fraktálů. Lineární deterministické lze popsat lineárními transformacemi a při jejich modelování se neuplatňují náhodná čísla. Lineární náhodné fraktály jsou v počítačové grafice nejčastější, neboť lineární transformace se snadno vypočítávají a náhodná čísla pomáhají generovat struktury podobné přírodě. Nelineární deterministické fraktály jsou v této knize zastoupeny Mandelbrotovou množinou a poslední třídou jsou nelineární náhodné fraktály.

Forma naprosté většiny členitých objektů, se kterými se v přírodě setkáváme, je fraktální, nebo je alespoň pomocí fraktálů popsatelná. Algoritmy generující fraktální množiny tedy poskytují aparát na simulování tvarů, se kterými se setkáváme v přírodě.

Výpočet fraktální dimenze

Fraktální dimenzi D nějakého objektu určíme ze vztahu (8.2) dosazením za $K = 1$ a limitním zmenšováním $\epsilon \to 0$. Dále uvedeme vztahy pro určení fraktálních dimenzí fraktálů deterministických a statistických.

Fraktální dimenzi D deterministického fraktálu, který vznikne aplikací n transformací φ_i (8.1), každá s měřítkem S_i, určíme řešením rovnice

$$\sum_{i=1}^{n} S_i^D = 1.$$

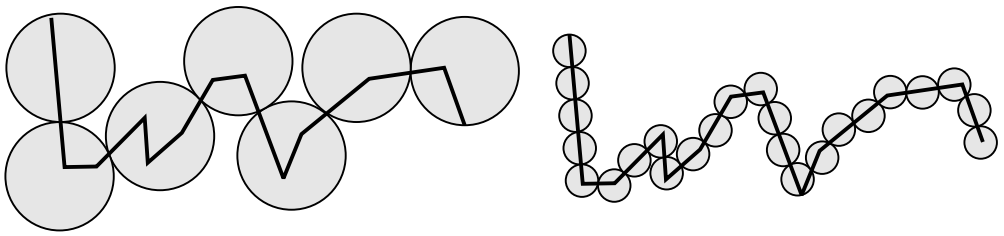
Fraktál, popsatelný n násobným opakováním jediné transformace φ s koeficientem měřítka S, bude mít dimenzi D

$$D = \frac{\log n}{\log 1/S}. \tag{8.4}$$

Pro odhad fraktální dimenze stochastických fraktálů se používají dvě základní metody, obě založené na tzv. *metodě počítání čtverců* (*box counting method*). Odhad fraktální dimenze provedeme ve dvou krocích následujícím postupem (viz obrázek 8.2). Mějme křivku, jejíž dimenzi odhadujeme. V prvním kroku ji pokryjeme N_1 nepřekrývajícími se kruhy stejné velikosti. Ve druhém ji pak pokryjeme N_2 menšími kruhy opět stejné velikosti. Poměr velikostí kruhů z prvního a druhého kroku je r. Odhad fraktální dimenze D je potom

$$D \approx \frac{\log N_2/N_1}{\log 1/r}.$$

Pokud bychom chtěli určit tuto dimenzi přesněji, museli bychom provést pokrytí několik. Tabulka 8.1 obsahuje údaje o odhadu fraktální dimenze některých přírodních útvarů.



Obrázek 8.2: Dvě různá pokrytí křivky při odhadu její fraktální dimenze

Zjevnou nevýhodou této metody je nutnost umístit kružnice přesně na křivku a navíc tak, aby se pouze dotýkaly. Jednodušší a navíc snadněji implementovatelná metoda počítá pokrytí čtverců. V prvním kroku se struktura, jejíž fraktální dimenzi měříme, pokryje pravidelnou sítí o rozměru $s_0 \times s_0$. Spočítají se všechny čtverce, které obsahují měřenou strukturu. Označme tento počet n_0. V dalším kroku zvětšíme rozlišení sítě na $s_1 \times s_1$ a spočítáme n_1. Měření opakujeme několikrát a vyneseme body na logaritmicko-logaritmické stupnici. Body se proloží nějakou aproximační přímkou jejíž sklon určuje fraktální dimenzi měřené struktury. Nejjednodušší je zvýšit počet čtverců v každém kroku dvakrát. Z poměru n_{i+1}/n_i se potom odhadne fraktální dimenze $D = \log_2(n_{i+1}/n_i)$.

objekt	*odhad fraktální dimenze*
pobřeží	1.26
povrch mozku člověka	2.76
neerodované skály	2.2 – 2.3
obvod 2D průmětu oblaku	1.33

Tabulka 8.1: Odhad fraktální dimenze některých přírodních útvarů

8.1.3 Multifraktály

Fraktální geometrie znamenala ve vědě poměrně velké nadšení. Chvíli se zdálo, že se jedná o univerzální lék, který pomůže kvantifikovat velké množství jinak obtížně postižitelných jevů. Bohužel tomu tak není. Například výše zmíněná metoda počítání čtverců předpokládá, že se měří jediná struktura, navíc ne sebeprotínající se. Co když však budeme mít na obrázku struktur více? Co když se jedná o průmět složité trojrozměrné struktury? Výsledek získaný výše zmíněnou

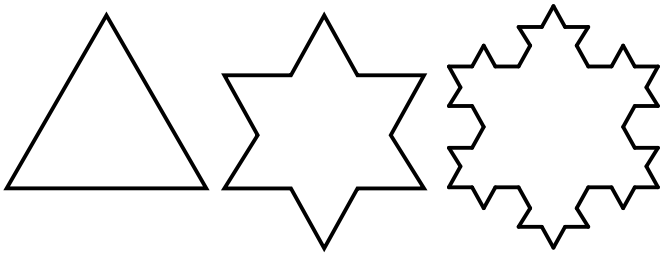
metodou bude zřejmě chybný. Pro více struktur na jediném obraze očekáváme více hodnot, jakési „Fourierovo spektrum“ fraktálních dimenzí. Tato úvaha vedla ke koncepci *multifraktálů* – struktur, složených z více fraktálů. Klasickým fraktálům se také říká monofraktály. Klasifikace a měření multifraktálů je obtížné, jejich implementace je překvapivě snadná a aplikace bohaté. V počítačové grafice se používají především pro generování krajin, které nemají ve všech směrech a ve všech částech stejné vlastnosti (jsou anizotropní a nehomogenní) a podobají se tak krajinám erodovaným [Eber03].

8.1.4 Lineární deterministické fraktály

Obrázek 8.3: Pět prvních iterací Cantorova diskontinua

Lineární deterministické fraktály lze generovat jednoduchými rekurzivními postupy a aplikováním transformací otáčení, posunutí a změny měřítka. Jedním z nejjednodušších exemplářů těchto fraktálů je *Cantorovo diskontinuum*. Tato množina vznikne tak, že vyjmeme z úsečky prostřední třetinu a tento proces aplikujeme rekurzívně na zbývající dvě třetiny (viz obr. 8.3).

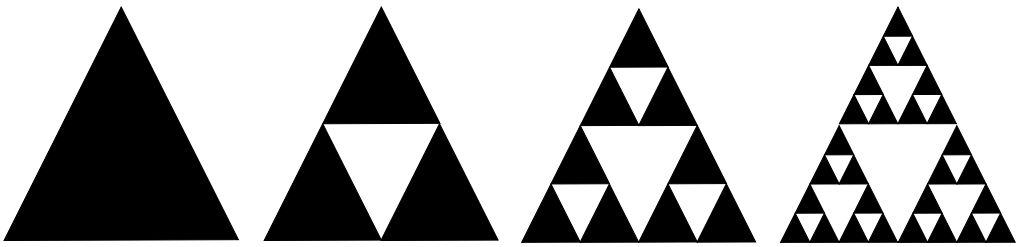
Fraktální dimenzi této množiny získáme dosazením do vztahu (8.4). Počet opakování, tedy kopií původní množiny je roven dvěma $N = 2$ a množina se zmenšuje na třetiny – tedy $r = 1/3$. Po dosazení získáme $D = \log 2 / \log 3 \approx 0.63$. Fraktální dimenze této množiny je menší než topologická dimenze přímky, ale vyšší než topologická dimenze bodu.



Obrázek 8.4: První dvě iterace Kochovy vločky

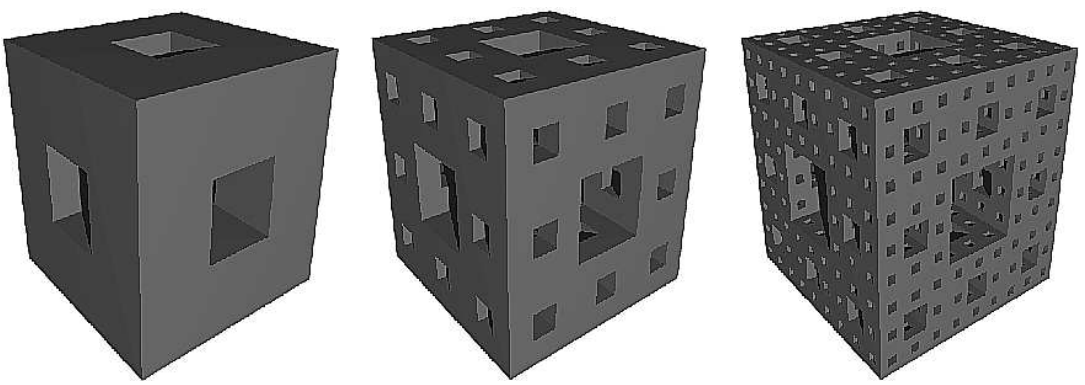
Helge von Koch popsal v roce 1904 množinu, která je dalším představitelem lineárních deterministických fraktálů, a která se jmenuje *Kochova sněhová vločka*. Několik jejích iterací je uvedeno na obrázku 8.4. Tento fraktál získáme například následujícím rekurzivním postupem. Vezmeme strany rovnostranného trojúhelníku a vyjmeme z nich prostřední třetinu, stejně jako v případě Cantorova diskontinua. Nad vzniklými mezerami vztyčíme vždy dvě strany z rovnostranného trojúhelníku o délce stran rovné jedné třetině původní úsečky. V dalším kroku aplikujeme tento postup na všechny úsečky zbývající v této množině. Tímto dělením se postupně okraj vločky zjemňuje a po nekonečně mnoha iteracích je jeho délka nekonečná, přestože ohraničuje konečnou

plochu. Navíc tato křivka nemá v žádném svém bodě derivaci. Fraktální dimenzi této množiny $\log 4/\log 3 \approx 1.26$ vypočítáme opět dosazením do vztahu (8.4).



Obrázek 8.5: Několik iterací Sierpinského fraktálu

Sierpinského fraktál (též Sierpinského krajkoví, Sierpinského košík) získáme z trojúhelníku rekurzivním vyjímáním trojúhelníku, který vznikne spojením středů jeho stran tak, jak je uvedeno na obrázku 8.5. Fraktální dimenze Sierpinského fraktálu je $\log 3/\log 2 \approx 1.59$. W. Sierpinsky původně hledal něco zcela jiného. Hledal množinu, která obsahuje všechna možná topologická jednorozměrná větvení. Tak objevil množinu později pojmenovanou *Sierpinského koberec*. Získáme ji snadno ze čtverce rekurzivním vyjímáním čtverce, který je určen jako průsečík kolmic vztyčených nad prostřední třetinou stran. Jejím obrazem je jedna strana *Mengerovy houby*, která vznikne rekurzivním vyjímáním částí krychle, jak je vidět na obrázku 8.6. Její fraktální dimenze je $\log 20/\log 3 \approx 2.726$.



Obrázek 8.6: Několik iterací Mengerovy houby

Deterministické fraktály, jako je Kochova křivka či Mengerova houba, jsou vhodným cvičením k pochopení principu fraktálů, neboť se na nich demonstrují jejich základní vlastnosti.

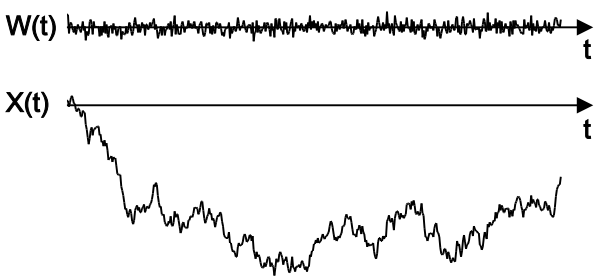
Deterministické fraktály umožňují jednoduchým způsobem vytvořit dostatečně složité povrchy a objemy a používají se proto často pro tvorbu testovacích modelů, na kterých jsou testovány rychlosti implementace metody sledování paprsku (viz kapitola 15.9). Generování Mandelbrotovy množiny se používá jako testovací úloha rychlosti CPU. Peanova křivka byla použita pro dithering a stejná množina se s výhodou aplikuje při zobrazování na obrazovce. Ukázalo se, že pokud lineární paměť grafického procesoru nebudeme mapovat do obrazu po řádcích, ale ve tvaru právě fraktální křivky, sníží se chybné zásahy vyrovnávací paměti grafického procesoru. Tato technika, zvaná *texture swizzling*, se používá například v herní konzoli Xbox.

8.1.5 Náhodné fraktály

Přírodní jevy, stejně jako objekty, které v přírodě nacházíme, jsou silně ovlivněny náhodnými procesy. Matematický formalismus, který nám umožňuje tyto jevy popsat, je založen na stochastických fraktálech; fraktálech, při jejichž generování se uplatňují náhodná čísla. Přírodní jevy jsou navíc silně nelineární. V počítačové grafice však obyčejně vystačíme s lineárními transformacemi a tak i používané náhodné fraktály jsou lineární. Pomocí nich dokážeme modelovat přírodní objekty jako hory, kameny, oblaka, zemský povrch apod., jak uvidíme dále v textu.

Brownův pohyb

Popis náhodných fraktálů má svůj základ ve studiu tzv. Brownova pohybu, který budeme označovat Bm (*Brownian motion*), jehož jednorozměrný příklad je na obrázku 8.7 dole. R. Brown pozoroval v roce 1827 chaotický pohyb mikroskopických částeček pylu ve vodě. Tento jev byl



Obrázek 8.7: Graf funkce jednorozměrných náhodných pulsů $W(t)$ s normalizovaným gaussovským rozložením $N(0,1)$ (nahoře). Tyto pulsy působí v kolmém směru na vodorovně se pohybující bod a jejich kumulativní součet je jednorozměrný Brownův pohyb $X(t)$ (dole).

v roce 1905 zdůvodněn Einsteinem jako pohyb způsobený náhodnými nárazy molekul vody

na relativně malá a lehká zrnka pylu. Mandelbrot použil Brownův pohyb pro simulaci přírodních objektů v počítačové grafice. Brownův pohyb bývá také nazýván *náhodná procházka* (*random walk*).

Nejprve popíšeme jednorozměrný Bm, později jeho zobecnění do vyšších dimenzí. Předpokládejme bod X, který je v čase t v poloze $X(t)$. Bod se pohybuje rovnoměrnou rychlostí ve směru osy x a je v diskrétních okamžicích $i = 0, 1, \ldots$ vychýlen v kolmém směru náhodnými pulsy $W(t)$. Obrázek 8.7 demonstruje slovně popsaný postup. Na horním obrázku je spojitě zobrazena velikost náhodných pulsů, dole je výsledná trajektorie pohybujícího se bodu.

Náhodné pulsy musí mít gaussovské rozložení a jejich střední hodnota musí být rovna nule. Tato podmínka charakterizuje energii molekul, které vrážejí do zrnka pylu. Náhodná, či spíše pseudonáhodná čísla, která poskytují programovací jazyky, mají rovnoměrné rozložení, zvané též *bílý šum* (*white noise*). Energetické spektrum takových čísel je přímka, a proto se mu také říká $1/f^0$ šum. gaussovská náhodná čísla generují čísla s pravděpodobností popsanou rovnicí (22.36), zobrazenou na obrázku 22.7 s energetickým spektrem $1/f^2$.

Jak získáme gaussovská náhodná čísla z náhodných čísel s rovnoměrným rozložením? Detailní popis nalezne čtenář v [Peit92], zde se omezíme na funkční popis. Podstatou tohoto postupu je fakt, že součet náhodných čísel s rovnoměrným rozložením aproximuje rozložení Gaussovo. My potřebujeme generátor gaussovských čísel s normalizovaným rozložením $N(0, 1)$, kde střední hodnota μ je rovna nule a rozptyl σ^2 je roven jedné. Náhodná čísla s rovnoměrným rozložením generovaná počítačem (například funkcí `rand()`)jsou celá čísla v intervalu $\langle 0, A\rangle$, kde A je $2^{15} - 1$ nebo $2^{31} - 1$. Předpokládejme, že vygenerujeme n takovýchto náhodných čísel d_i, $i = 1, 2, \ldots, n$. Gaussovská náhodná čísla označená W s normálním rozložením pak získáme pomocí formule

$$W = \frac{1}{A}\sqrt{\left(\frac{12}{n}\right)}\sum_{i=1}^{n} d_i - \sqrt{3n}.$$

Generátory pseudonáhodných čísel jsou obyčejně pomalé a tak v praxi vystačíme s malými hodnotami n (v rozsahu od tří do deseti), případně můžeme gaussovská náhodná čísla vygenerovat do tabulky a vhodnou hašovací funkcí z ní pak vybírat. Důležité je, aby délka periody byla co možná největší.

Označme gaussovské náhodné číslo, tj. vertikální puls působící na pohybující se bod v čase t, jako $W(t)$. Brownův pohyb je funkce $X(t)$, kterou získáme jako kumulativní součet náhodných gaussovských pulsů pohybujícího se bodu [Mand82]

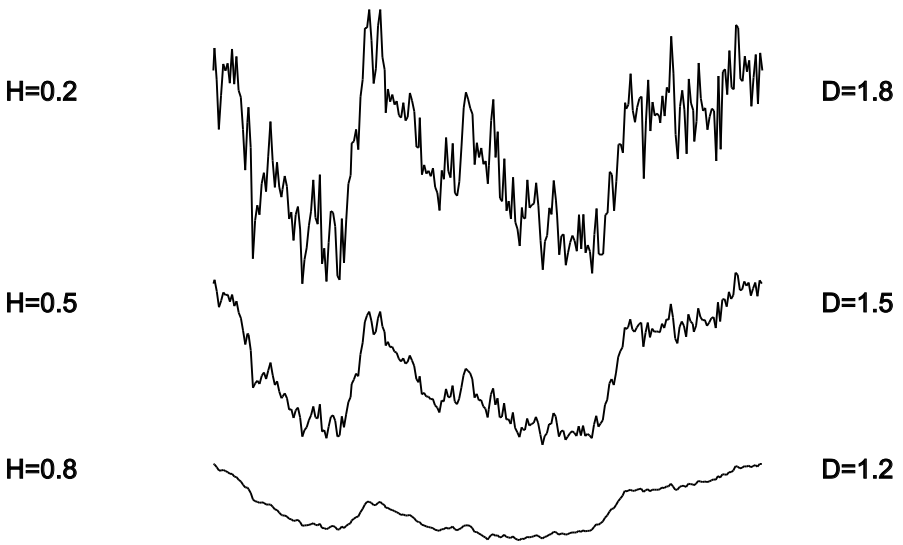
$$X(t) = \sum_{s=-\infty}^{t} W(s). \tag{8.5}$$

Bm je fraktál s fraktální dimenzí 1.5 a zajímavé je, že body získané průnikem grafu Brownovy funkce s osou x tvoří fraktální množinu s fraktální dimenzí $D = 0.5$.

Bm tedy poskytuje fraktály s dimenzí rovnou 1.5 pro jednorozměrný případ, 2.5 pro dvojrozměrný atd. V počítačové grafice však potřebujeme náhodné fraktály s libovolnou fraktální dimenzí. Ty lze generovat pomocí zobecnění Bm, kterým je tzv. *zlomkový Brownův pohyb* (*fractional Brownian motion*) označovaný jako fBm. fBm získáme z Bm změnou měřítka, musíme však použít jiný koeficient ve vodorovném směru a jiný ve směru svislém. Náhodné fraktály s dimenzí D potom získáme změnou měřítka Bm tak, že násobíme ve směru osy x koeficientem r a ve směru osy y koeficientem $1/r^H$. H je tzv. *Hurstův exponent* a svazuje fraktální dimenzi D vztahem

$$D = 2 - H, 0 \leq H \leq 1. \tag{8.6}$$

Pojdmě shrnout, jak se generuje fBm. Vygenerujeme obyčejný Bm a zvolíme fraktální dimenzi D. Z této volby a z rovnice (8.6) určíme Hurstův exponent H. Poté vygenerované hodnoty vynásobíme ve směru osy x hodnotou r, nejčastěji se volí $r = 2$ a ve směru osy y koeficientem $1/r^H$.



Obrázek 8.8: Změna tvaru fBm v závislosti na Hurstově exponentu H

Podívejme se také, jak souvisí chování fBm s koeficientem H (obrázek 8.8).

- Je-li $H = 0.5$ je fBm obyčejný Bm a fBm je tzv. nekorelovaná funkce; máme-li tři čísla $t_1 < t_2 < t_3$, jsou charakteristiky intervalů $X(t_2) - X(t_1)$ a $X(t_3) - X(t_2)$ statisticky nerozlišitelné. Fraktální dimenze křivky je 1.5.
- Pro $H > 0.5$ je fBm kladně korelovaný. Pokud $X(t)$ v nějakém intervalu $\langle t_0, t_1 \rangle$ roste, má tendenci růst i pro hodnoty $t > t_1$. Pokud funkce klesá v $\langle t_0, t_1 \rangle$, má tendenci klesat

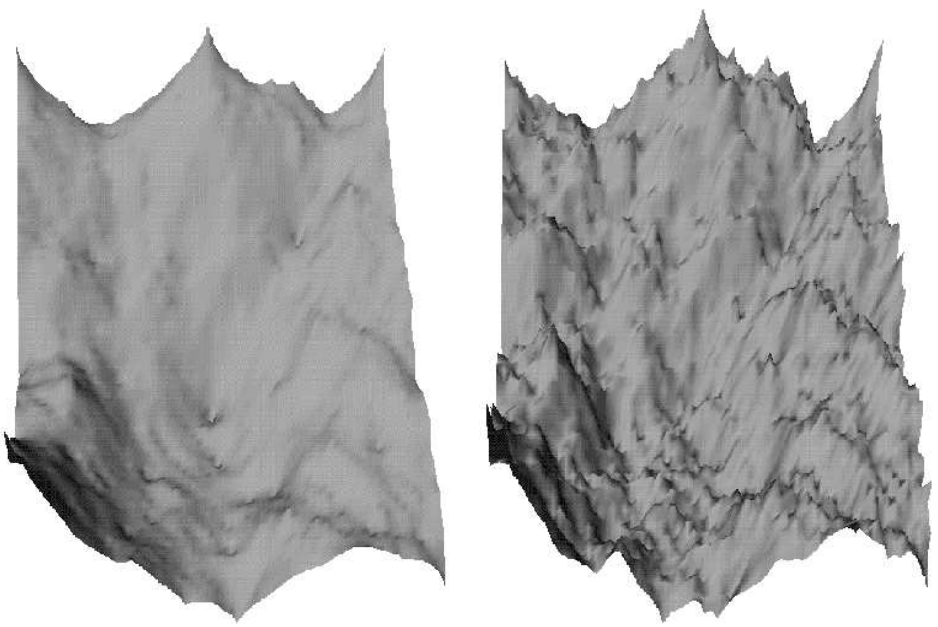
i pro $t > t_1$. Funkce se vyhlazuje. Fraktální dimenze je menší nežli dimenze Bm, křivka je méně členitá.

- Pro $H < 0.5$ je fBm záporně korelovaný. Pokud $X(t)$ v nějakém intervalu $\langle t_0, t_1 \rangle$ roste, má tendenci pro hodnoty $t > t_1$ klesat a naopak. Funkce má tendenci oscilovat. Fraktální dimenze je větší nežli dimenze Bm, křivka je více členitá.

Rozšířením jednorozměrné fBm do dvou rozměrů je tzv. *zlomková Brownova plocha* (fBm plocha). Její fraktální dimenze je dána vztahem

$$D = 3 - H, \tag{8.7}$$

kde je H je Hurstův exponent z intervalu $H \in \langle 0, 1 \rangle$ a její fraktální dimenze $D \in \langle 2, 3 \rangle$. Aby byla zachována podmínka statistické soběpodobnosti, musí mít tato plocha stejné statistické momenty. Jak jsme uvedli v úvodu, obyčejně postačuje shodnost průměru a směrodatné odchylky. Tyto veličiny musí být shodné nejen v každých dvou libovolných intervalech, ale také ve všech směrech. Změnu členitosti fBm plochy při změně Hurstova koeficientu H demonstruje obrázek 8.9.

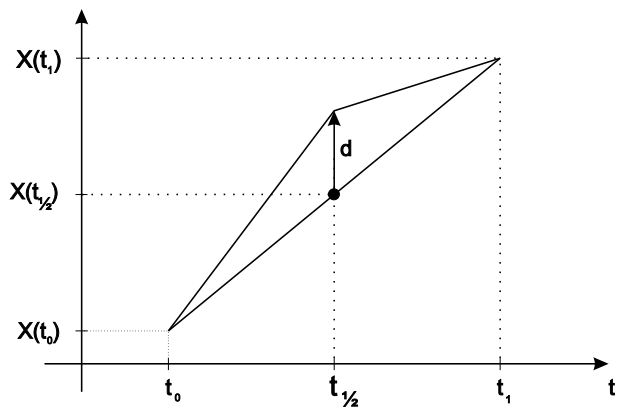


Obrázek 8.9: Vliv Hurstova koeficientu na členitost fBm plochy. Jednotlivé obrázky ukazují změnu členitosti plochy pro hodnoty $H = 2.2$ (vlevo) a $H = 2.5$

Nevýhodou měřítkování Bm za účelem získání fBm je jeho nesnadné řízení. Často totiž chceme (například) získat fraktály až do určitého detailu a vyšší detaily nás nezajímají. Z tohoto důvodu, a z důvodu snadné zobecnitelnosti do vyšších dimenzí, se nejčastěji používají metody uvedené v následujících odstavcích.

Metoda náhodného přesouvání středního bodu

Alternativní metodou modelování fBm k metodě uvedené v předchozím odstavci, je metoda *náhodného přesouvání středního bodu* (*random midpoint displacement*). Její podstata spočívá v rekurzivním dělení úsečky na polovinu a posouvání středního bodu o náhodné číslo δ tak, jak demonstruje obrázek 8.10.



Obrázek 8.10: Princip metody přesouvání středního bodu

Takovým postupem se úsečka postupně nahrazuje lomenou čárou. Vzhledem k tomu, že výsledná fraktální křivka interpoluje dva body, říká se této metodě také *fraktální interpolace*.

Vstupem algoritmu je úsečka určená dvěma body $[t_0, X(t_0)]$, $[t_1, X(t_1)]$ a Hurstův exponent H. Výstupem je fraktální křivka procházející zadanými body. Fraktální dimenze je určena Hurstovým exponentem a vypočítá se podle vztahu (8.6).

Vlastní algoritmus v prvním kroku určí střed úsečky a tento bod se posune o náhodné číslo δ ve směru osy y

$$\left[t_{1/2}, X(t_{1/2})\right] = \left[\frac{t_1 + t_0}{2}, \frac{X(t_1) + X(t_0)}{2} + \delta\right].$$

Tento proces se aplikuje rekurzívně na obě vzniklé úsečky. Velikost náhodného čísla δ_i, které v $i-$té iteraci přičítáme ke středu úsečky, ovlivňuje fraktální dimenzi výsledné křivky. Aby křivka byla fBm, musí být δ_1 náhodné číslo s normalizovaným Gaussovým rozložením $N(0,1)$ a v $i-$té iteraci musí být δ_i náhodné číslo s Gaussovým rozložením, které má oproti σ^2 modifikovaný rozptyl v závislosti na H a i takto

$$\sigma_i^2 = \frac{1}{2^{2H(i+1)}}\sigma^2. \tag{8.8}$$

Z obrázku 8.12 je patrné, že první iterace mají největší vliv na výsledný tvar křivky. V každé další iteraci klesá velikost náhodného čísla s druhou mocninou. Složitě vypadající vztah se

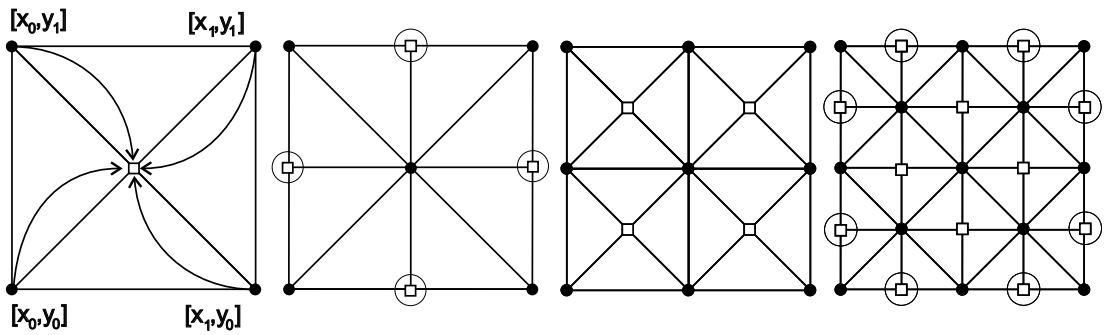
v praxi zjednodušuje tak, že se velikost náhodného čísla svazuje se vzdáleností krajních bodů úsečky. Tím zaručíme, že nebudeme přidávat detaily větší, nežli je vzdálenost těchto dvou bodů. Označme krajní body úsečky A a B. Pro výpočet posunutí bodu pak použijeme vztah

$$\delta = H.Gauss(0,1) * |A,B|,$$

kde H je Hurstův exponent, $Gauss(0,1)$ je náhodné číslo s normalizovaným Gaussovým rozložením a $|A,B|$ je vzdálenost bodů A a B.

Kritérium zastavení rekurzivního dělení určíme podle konkrétní aplikace. Vhodným ukazatelem může například být vzdálenost koncových bodů úsečky. Je-li délka úsečky menší než délka úhlopříčky pixelu, má další dělení smysl pouze v případě, že této informace využijeme pro antialiasing. Jiným kritériem je maximální počet generovaných úseček atp.

Metoda náhodného přesouvání středního bodu ve třech rozměrech se aplikuje buď na trojúhelníky, nebo na obdélníky či na čtverce. Popíšeme složitější, leč častěji používanou metodu, kdy se rekurzívně dělí čtverec.



Obrázek 8.11: První čtyři kroky metody přesouvání středního bodu ve třech rozměrech (půdorys). Původní body jsou označeny plným kroužkem, nově přidané čtverečkem.

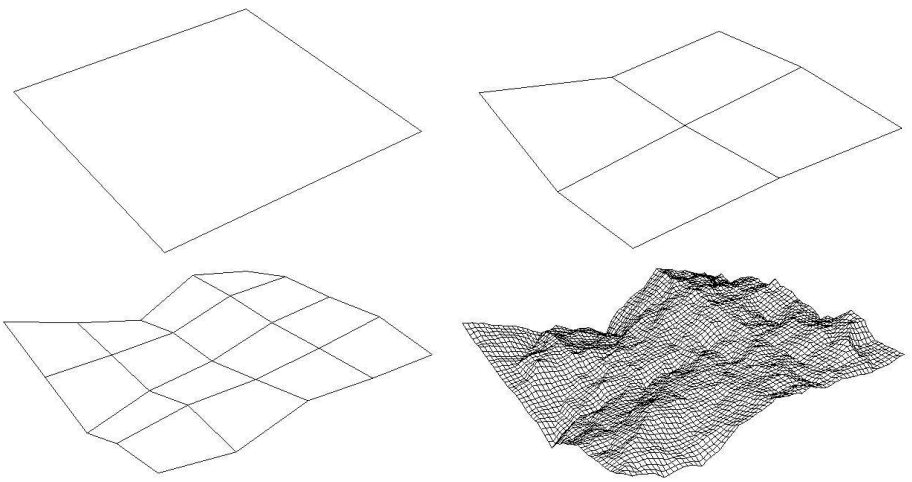
Mějme čtyři body $[x_0, y_0, f(x_0, y_0)]$, $[x_1, y_0, f(x_1, y_0)]$, $[x_0, y_1, f(x_0, y_1)]$ a $[x_1, y_1, f(x_1, y_1)]$, jejichž průmět do roviny xy určuje čtverec (obrázek 8.11). Při rekurzivním dělení nejprve zjistíme polohu středu čtverce o souřadnicích $[x_{1/2}, y_{1/2}, f(x_{1/2}, y_{1/2})]$, kde

$$\begin{aligned} x_{1/2} &= \frac{x_0 + x_1}{2} \\ y_{1/2} &= \frac{y_0 + y_1}{2} \\ f(x_{1/2}, y_{1/2}) &= \frac{f(x_0, y_0) + f(x_1, y_0) + f(x_0, y_1) + f(x_1, y_1)}{4}. \end{aligned} \tag{8.9}$$

Souřadnice $z = f(x_{1/2}, y_{1/2})$ tohoto bodu se zvětší o číslo δ, které je gaussovským náhodným číslem. Rozptyl rozložení těchto náhodných čísel se mění v závislosti na hloubce rekurze podle vztahu (8.8) a fraktální dimenze se určí z (8.7).

Ve druhém kroku se vypočítají body, které leží na hranách zadaného čtverce. To docílíme tak, že otočíme čtverec o 45° a vypočítáme požadované hodnoty stejným způsobem, jako v předchozím kroku. V dalším kroku ho otočíme o 45° zpět a provedeme rekurzívně první dva kroky na čtyři nově vzniklé čtverce. Algoritmus je vlastně dvoukrokový, nejprve se vypočtou středy čtverců, potom hodnoty na hranách. Zmíněné otočení čtverce o 45° evokuje vizuální podobu se symbolem diamantu. Z tohoto důvodu se někdy tento algoritmus označuje jako *diamond-square algorithm.*

V některých případech má nový bod pouze tři sousedy a nemůže se tedy určit jako střed čtverce. Na obrázku 8.11 se jedná o vrcholy, které jsou označeny kroužkem. V těchto případech se vypočítá hodnota pouze z existujících sousedů – chyba, ke které u okraje dochází, nebývá vizuálně patrná. Obrázek 8.12 ukazuje několik prvních kroků této metody aplikované na čtverec.



Obrázek 8.12: Metoda přesouvání prostředního bodu aplikovaná na čtverec

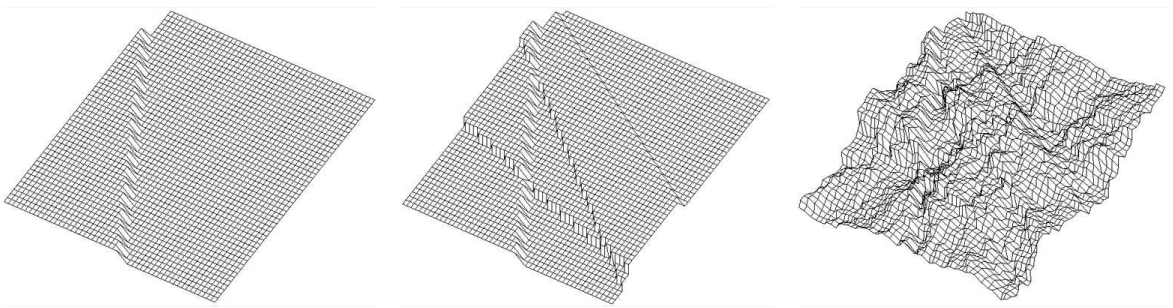
Otázkou samozřejmě je, kdy zastavit rekurzivní volání. Pokud stanovíme hloubku rekurze pevně, získáme stejně velké čtverce. Jinou možností je ukončit rekurzivní postup adaptivně: jsou-li například rohy čtverce tak blízko u sebe, že po promítnutí na obrazovku leží na ploše jediného pixelu.

Metoda náhodných poruch

Další z metod generování fBm se jmenuje metoda náhodných poruch (*random faults method*) a je patrně implementačně nejjednodušší. Popíšeme algoritmus generování zlomkové Brownovy plochy fBm.

Algoritmus se aplikuje na dvojrozměrnou matici, v níž je hodnota každého bodu výškou. Na začátku se hodnoty v celé matici nastaví na nulu. Vlastní algoritmus není rekurzivní, ale iterativní. V dostatečně dlouhém cyklu provádíme následující kroky. Rozdělíme matici na dvě libovolné části, nejjednodušeji tak, že ji přetneme přímkou. Výšky všech bodů na jedné straně zvýšíme o gaussovské náhodné δ_i, hodnoty bodů na druhé straně naopak o tuto hodnotu snížíme. V každém kroku snižujeme rozptyl náhodných čísel podle rovnice (8.8). Limitou tohoto procesu je zlomková Brownova plocha. Obrázek 8.13 demonstruje několik prvních iterací tohoto postupu.

Jak jsme uvedli, podstatné je rozdělení plochy na dvě části. Alternativou k rozdělení přímkou je „obtisk“ libovolného objektu. Můžeme například do plochy náhodně vkládat kružnice s klesajícím poloměrem. Kružnice však obecně nerozdělí plochu na dvě stejně velké části. Abychom zabránili „bobtnání“ některé části plochy, musíme měnit periodicky nebo náhodně, avšak se stejnou pravděpodobností, oblasti, jejichž hodnoty se zvyšují a snižují. Nejprve například snížíme hodnotu všeho, co je uvnitř kružnice, a zvýšíme hodnotu vnějšku a v dalším kroku naopak.



Obrázek 8.13: Metoda náhodných poruch aplikovaná na čtverec

Existují i jiné metody generování fBm. Z těch, které jsme v tomto textu neuvedli, jsou často používané algoritmy popsané v knize [Eber03].

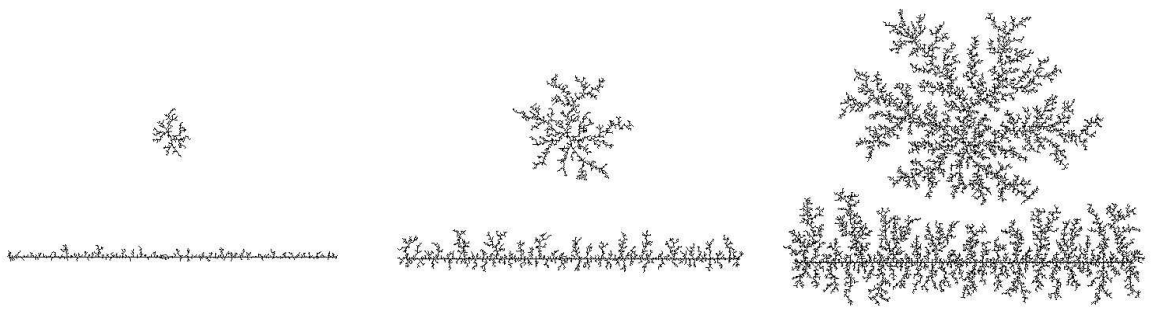
8.2 Procedurální a fraktální modely v počítačové grafice

Simulace fBm je vhodná pro modelování mnoha přírodních objektů. Pokud například změříme fraktální dimenzi obrysu nějakého ostrova a vygenerujeme náhodný fraktál se stejnou fraktální dimenzí, zjistíme, že oba objekty jsou si vizuálně podobné. Příkladem tohoto postupu je generování obrysu ostrova na obrázku 8.18 aplikací metody přesouvání středního bodu na strany trojúhelníka. Fraktální dimenze výsledného objektu je $D = 1.5$. Podobně metody pro generování fBm ve vyšších dimenzích jsou teoretickým základem pro generování syntetických hor. Fraktální horu na obrázku 8.15 jsme získali metodou náhodného přesouvání středního bodu ve třech dimenzích.

Při simulaci přírodních objektů je obyčejně vstupem použitého algoritmu fraktální dimenze (resp. Hurstův koeficient) výsledného objektu a jeho dimenze topologická. Tabulka 8.1 ukazuje naměřené fraktální dimenze některých objektů v přírodě. Je zajímavé, že pro modelování zeměpisných útvarů se nejvíce hodí objekty, jejichž fraktální dimenze je 1.2 až 1.3 násobkem jejich dimenze topologické.

8.2.1 Difúzí omezená agregace a korály

Jednou z přímých aplikací jednorozměrného Brownova pohybu je tzv. difúzí omezená agregace (*diffusion limited aggregation*), označovaná jako DLA. DLA je fyzikálním jevem, při kterém vznikají dendrity například při elektrických výbojích, podobně rostou obrazce na zamrzlých sklech. V počítačové grafice se používá algoritmus simulace DLA například pro modelování struktur a růstu korálů.



Obrázek 8.14: DLA

Předpokládejme, že máme roztok, ve kterém volně plavou molekuly nějaké látky, a v roztoku

je zároveň kondenzační jádro či jádra. Když se některá z pohybujících se molekul k jádru přiblíží, je jím zachycena, nalepí se na něj, a stane se rovněž kondenzačním jádrem. Tímto způsobem difúze molekul postupně přidává nová jádra a agreguje se tak nový tvar. Výsledkem tohoto postupu ve dvou dimenzích jsou fraktální struktury s dimenzí okolo 1.7. Existuje velké množství variant tohoto modelu, jedna z nich například nepřidává molekulu při prvním doteku s kondenzačním jádrem, ale předpokládá, že povrch má pomyslné čítače doteků. Molekula se agreguje teprve tehdy, když čítač dosáhl určitého počtu zásahů. Jiný model přidává novou molekulu pouze s určitou pravděpodobností atp.

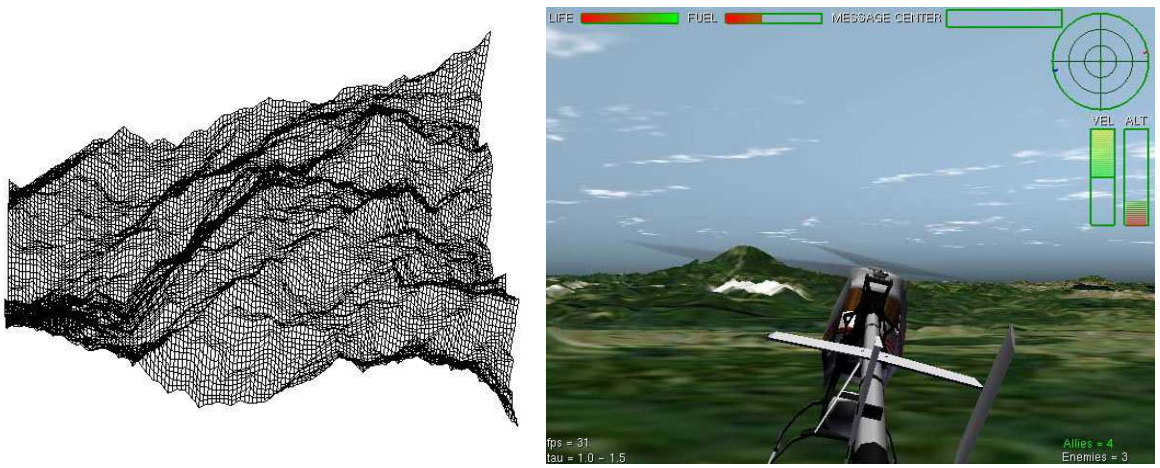
Algoritmus simulace DLA popíšeme na dvojrozměrném případě a budeme simulovat agregaci ve dvojrozměrné matici. Celá matice se na začátku vynuluje a na některá místa se zapíší nenulové hodnoty, které označují kondenzační jádra. Tím je celý algoritmus inicializován. Postupně v cyklu na okraj matice položíme částici a aplikujeme na ní algoritmus Brownova pohybu. Částice se bude chaoticky pohybovat (náhodnou procházkou) a pokud se dotkne kondenzačního jádra, tak se k němu připojí. V algoritmu jednoduše označíme prvek matice jako „obsazený“. Pokud částice náhodou opustí simulační oblast, zastavíme její trasování a vygenerujeme částici novou. Obrázek 8.14 ukazuje postupné narůstání dvou struktur. První struktura na obrázku nahoře kondenzuje z bodu, dolní útvar vzniká kondenzací na úsečce. Simulace růstu je poměrně pomalá, algoritmus má exponenciální složitost a generování velkých struktur je obyčejně zdlouhavé, i když se při zaplňování oblasti výrazně zrychluje. Existují různé možnosti jeho zrychlení. Implementačně nejjednodušší je obklopení kondenzačního jádra obdélníkovou hranicí odpovídající minimálním a maximálním souřadnicím obsazených míst. Pokud je částice mimo tuto oblast, bude se pohybovat rychleji. Rozšíření algoritmu do více rozměrů je jednoduché a velké množství variant tohoto algoritmu, stejně jako simulace růstu polygonálních struktur, je k nalezení například v [Jone00].

8.2.2 Krajiny

Generování virtuálních krajin má v počítačové grafice dlouhou historii a aplikace těchto algoritmů jsou důležité v mnoha oblastech, z nichž patrně nejpodstatnější jsou letecké simulátory a videohry. Základní postup generování virtuální krajiny je téměř vždy složen ze dvou kroků. Nejprve se nějakým způsobem vygenerují data reprezentující hrubý tvar krajiny, může jít o data reálná, či o data získaná nějakou fraktální metodou. Ve druhém kroku se tato data upravují, nejčastěji erozními algoritmy.

Při modelování struktury krajiny se setkáváme s velkým množstvím problémů, v této kapitole se zaměříme především na generování přirozeně vypadajících povrchů krajin a různých povrchových struktur. Nebudeme se zabývat algoritmy rychlého zobrazování krajin a úrovní detailu. Jejich popis nalezne čtenář například v knize [Moll02].

Základní datovou strukturou, používanou při generování povrchu krajiny, je pravidelné výškové pole (*regular height field*). Jedná se o dvojrozměrnou matici, jejíž prvky se interpretují jako výška terénu nad základní úrovní. Nevýhodou této struktury je, že neumožňuje reprezentaci plně trojrozměrných objektů, tj. neumíme popsat například převisy a jeskyně, a proto se také někdy říká, že je 2.5 rozměrná. Výšková pole se obyčejně nezobrazují přímo, ale převádí se na sítě trojúhelníků.



Obrázek 8.15: Fraktální povrch (vlevo) ($D = 2.5$) získaný metodou náhodného přesouvání středního bodu a jeho aplikace v jednoduché hře (obrázek poskytl N. Goméz, ITESM Campus Ciudad de México)

Voxely se používají pro plně trojrozměrnou reprezentaci krajin. Umožňují zachytit libovolné povrchy a členitost terénu, zásadní nevýhodou je jejich datová náročnost a pomalost zobrazování. Prostorovou náročnost této reprezentace můžeme zmenšit RLE kompresí sloupců [Bene01], což umožňuje na takto komprimovaná data aplikovat algoritmy změn povrchu. Reprezentaci lze dále rozšířit o částice modelující prach či padající písek [Onou03].

Sítě trojúhelníků či polygonů jsou další možností, jak reprezentovat krajinu. Plně trojrozměrná reprezentace však umožňuje reprezentovat pouze povrch krajiny. Základní výhodou sítí trojúhelníků je jejich rychlé zobrazení. Adaptivní algoritmy zobrazování dokonce dynamicky generují zjednodušené sítě podle zvolené úrovně detailu. Nejpoužívanější algoritmy jsou založeny na stromech rovnoměrně rozdělených trojúhelníků – BinTree a nejznámější z nich je RealTime Optimally Adapting Meshes (ROAM) [Duch97].

Existují komerčně používané či obecně přijaté formáty pro reprezentaci povrchu krajiny a na Internetu je volně ke stažení obrovské množství dat získaných snímky z družic či různých

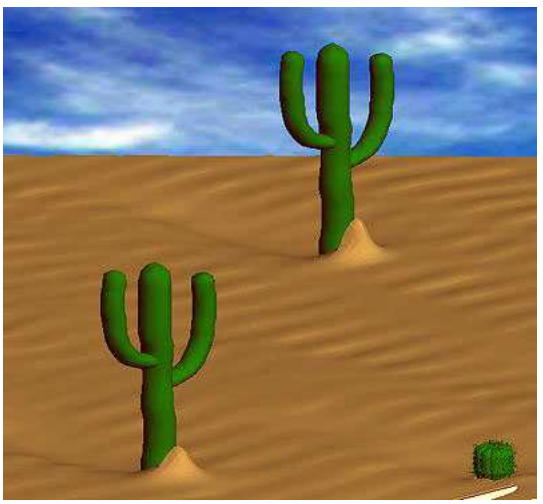
výškoměrů. Asi nejpoužívanější je formát DEM (*Digital Elevation Model USGS*) či rozšíření formátu TIFF zvané GeoTIFF. V těchto formátech jsou dostupná data částí zemského povrchu, a dokonce také data popisující některá mimozemská tělesa.

Základním algoritmem modelování povrchu krajiny je generování fBm plochy, kdy pomocí fraktální dimenze řídíme její členitost. Fraktální hory se však od hor v přírodě liší jedním zásadním způsobem [Mand82]. Fraktální modely hor totiž vypadají jako fraktální hory, i když je otočíme vzhůru nohama. V přírodě mají tuto vlastnost pouze geologicky mladé objekty. Hory, které kolem sebe běžně vidíme, jsou opotřebovány erozí a tvar vrcholů je odlišný od tvaru údolí. Další kroky vedoucí k vyšší věrohodnosti syntetických krajin se logicky ubíraly i tímto směrem. Algoritmy zabývající se erozí fraktálních povrchů lze zhruba rozdělit do tří základních kategorií – algoritmy simulující termoerozi, hydroerozi a větrnou erozi.

Termoeroze, vliv teploty na tvar povrchu, je proces opadávání částeček materiálu vlivem teplotních rozdílů. Termoeroze se vizuálně projevuje vyhlazováním povrchu, stíráním detailů, ostrých hran atp.

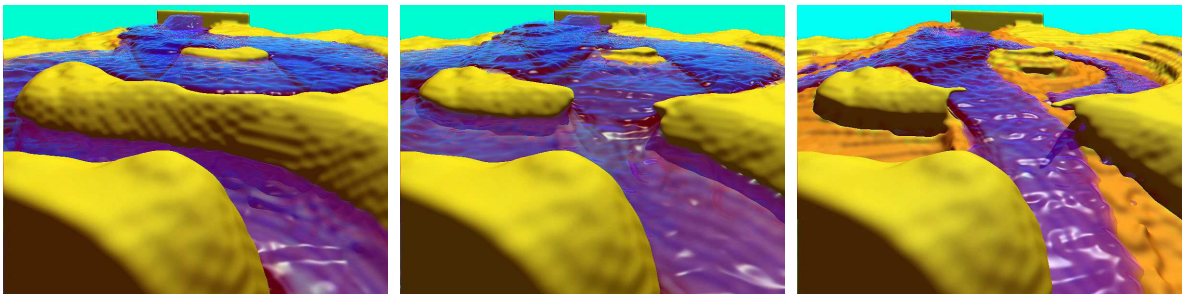
Hydroeroze, neboli působení vody, se projevuje globálně i lokálně. Globální hydroeroze je způsobena dopadem kapek deště a projevuje se podobně jako termoeroze (z hlediska času však daleko intenzivněji), ale především způsobuje přenášení rozpuštěného materiálu z vyšších míst do nižších. Lokální hydroeroze je způsobena drolením částeček materiálu tekoucí vodou (viz obrázek 8.17) a projevuje se jako vádí (vyschlé řečiště), meandry, případně jako delta.

Větrná eroze je nejméně častý případ eroze používané v počítačové grafice. Zajímavý algoritmus pro generování větrnou erozí formovaných pouštních povrchů je uveden v [Onou00]. Podstatou této metody je uchopení částic písku větrem a jejich přenesení na jiné místo. Částice se poté uloží do energeticky výhodné polohy sesunem po úbočí větrných návrší. Tím vznikají pro pouště charakteristické písečné vlny, jak je vidět na obrázku 8.16.



Obrázek 8.16: Pouštní krajina formovaná větrnou erozí [Bene04]

Jiné metody pro generování modelů erodovaných povrchů [Kell88] v prvním kroku modelují síť řek (například pomocí fBm nebo pomocí L-systémů) a ve druhém kroku k této síti generují krajinu – nejčastěji fraktální interpolací.



Obrázek 8.17: Příklad hydroeroze počítané ve voxelovém prostoru pomocí kompletního řešení Navier-Stokesových rovnic pro fluidní dynamiku. Obrázky poskytl V. Těšínský, ČVUT v Praze

8.2.3 Planety, pobřeží a oblaka

Pro generování modelů planet je asi nejčastěji používána metoda náhodných chyb (viz odstavec 8.1.5) aplikovaná na kouli. Koule je reprezentována jako hustá síť trojúhelníků. Je důležité, aby všechny trojúhelníky měly přibližně stejnou velikost, a tak se pro její generování obyčejně používá metoda rekurzivního zjemňování dvou na sobě položených čtyřstěnů, jejichž vrcholy musí ležet na povrchu koule.

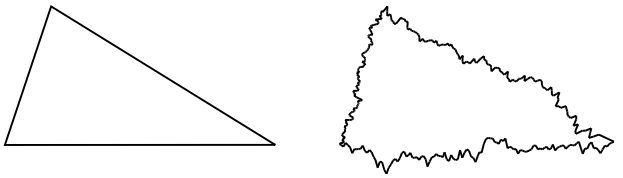
Při inicializaci se všem vrcholům trojúhelníků přiřadí stejná barva. Koule se iterativně rozdělí na dvě části a každé polokouli se přiřadí změna barvy o náhodné gaussovské číslo, které klesá s počtem iterací. Pro vedení náhodného řezu máme dvě možnosti; vést ho vždy středem koule, nebo nikoli. První algoritmus generuje fraktální kouli, jejíž jedna polokoule je negativem druhé, a není tedy pro praktické použití příliš vhodný. Samozřejmě, že nemusíme pouze měnit barvu, ale můžeme měnit i výšku každého bodu. Problémem je, že nezískáme příliš reálné objekty, neboť poměr velikosti povrchových struktur k poloměru planety je obyčejně zanedbatelný. Vodní hladina se modeluje jako určitá úroveň pod níž jsou všechny hodnoty zaokrouhleny. Obarvením vodní hladiny modrou barvou se získá věrohodný model planety s oceány a kontinenty.

Generování členitějších struktur lze snadno provést pomocí fraktální interpolace hranic nějakého povrchového modelu. Chceme-li například získat model asteroidu, je možné vytvořit ho jako hladkou implicitní plochu, převést ji na síť trojúhelníků, a na takto získaný objekt aplikovat algoritmus fraktální interpolace pomocí fBm.

Připomeňme, že takto získané modely jsou monofraktály. Je samozřejmě možné aplikovat na libovolnou síť trojúhelníků multifraktální metody a získat tak modely, které mají například jinou fraktální dimenzi v údolích a jinou na vrcholcích hor, podobně, jako je tomu mnohdy v realitě. Algoritmus generování multifraktálů nalezneme například v [Eber03].

Pobřeží Modely pobřeží vycházejí z jejich členitosti a nejsnadněji se generují prostou fraktální interpolací několika bodů, které určují jejich obrys. Nejjednodušší metodou je náhodné přesouvání prostředního bodu (viz část 8.1.5). Příklad její aplikace na trojúhelník je na obrázku 8.18.

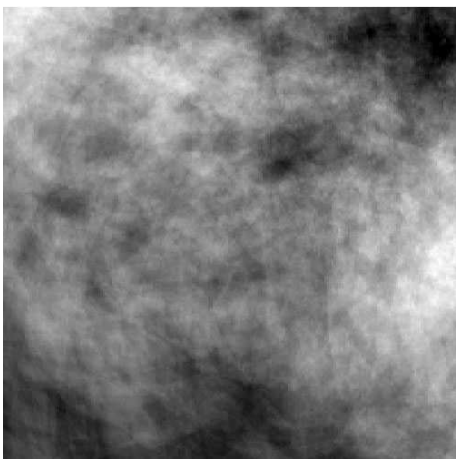
Všimněme si jednoho důležitého omezení. Touto metodou nelze generovat menší ostrovy blízko pobřeží. Z podstaty metody to není možné, neboť fraktální interpolace generuje spojitou křivku. Pokud bychom chtěli zároveň generovat ostrovy, museli bychom k okolí hranice přidat nějaké spojité objekty a aplikovat stejnou metodu ještě na ně.



Obrázek 8.18: Fraktální obrys pobřeží (vpravo) vzniklý aplikací metody náhodného přesouvání středního bodu na strany trojúhelníka

Oblaka Pro získání dvourozměrného modelu mraku můžeme využít fraktálních charakteristik fBm plochy. Zobrazíme-li tuto plochu v kolmém průmětu tak, že budeme výšku reprezentovat jako barvu – například odstínem šedi – získáme poměrně věrohodný obraz mraku (viz obrázek 8.19). Hurstův koeficient opět určuje členitost této množiny.

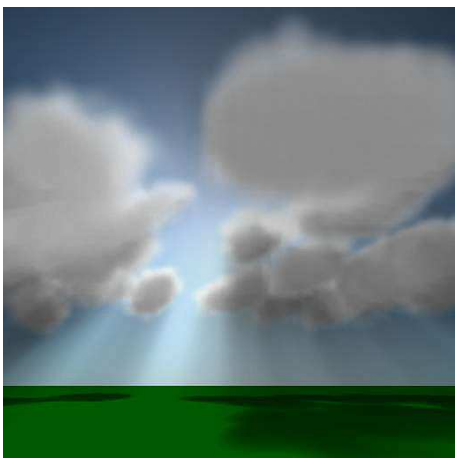
Pokud bychom chtěli získat trojrozměrný model mraku, můžeme vyjít z rozšíření metody náhodného přesouvání středního bodu do vyššího rozměru. Čtvrtý rozměr tohoto fraktálu se interpretuje jako hustota. Nevýhodou získané množiny je, že se obtížně zobrazuje. Obyčejně se pro její zobrazování používá metoda sledování paprsku, která při výpočtu průsečíku zohledňuje hustotu jako útlum. Chápeme-li čtvrtý rozměr jako čas a zobrazujeme-li získaný fraktál v čase, můžeme zobrazení interpretovat jako animovanou sekvenci jeho vývoje.



Obrázek 8.19: Průmět fBm plochy do roviny xy, výška reprezentována jako odstín šedi

Pro většinu objektů, které modelujeme pomocí procedurálních technik, je společné to, že nejsou ve středu zájmu diváka virtuální scény, ale slouží jako nějaké pozadí. Z těchto důvodů je na procedurálně generované modely možné aplikovat nejrůznější zjednodušení. Patrně nejběžnější metoda zobrazování mraků je používána v leteckých simuláto-

rech a spočívá v nahrazení mraku množinou poloprůhledných rovnoběžných bilboardů (viz část 19.2.3). Model mraku je rozdělen na vrstvy a každá z nich je promítnuta do roviny. Obraz vrstvy v rovině je potom zobrazen jako poloprůhledný billboard. Při průletu takto reprezentovaným mrakem dochází ke skokové změně viditelnosti, která je však díky průhlednosti nepostřehnutelná [Doba00].



Obrázek 8.20: Oblaka generovaná jako otexturované elipsoidy (obrázek poskytl M. Poneš, ČVUT v Praze)

Nejobecnější přístup k modelování atmosférických jevů je generování trojrozměrných šumových funkcí, reprezentovaných jako objemové struktury. Nejjednodušší varianty používají trojrozměrné elipsoidy s poloprůhlednou texturou. Jejich kombinace vede k překvapivě realistickým modelům mraků zejména typu *cumulus* či *cirrus*. Generování mraků je obyčejně založeno na modifikaci parametrů fraktálních funkcí a dosažení požadovaného výsledku může být značně pracné. Volumetrické funkce vhodné pro tento typ modelování jsou k nalezení v [Eber03], kde jsou také další odkazy. Příklad je uveden na obrázku 8.20.

8.3 Systémy částic

Silnou modelovací a zobrazovací technikou jsou systémy částic (*particle systems*), které se používají zejména k modelování objektů, jejichž tvar je natolik členitý, nebo se mění takovým způsobem, že ho není možno reprezentovat jako povrch. Takovými objekty jsou hejna ptáků či ryb, padající sníh, déšť, oheň, mlha, dým, tráva, les, atp.

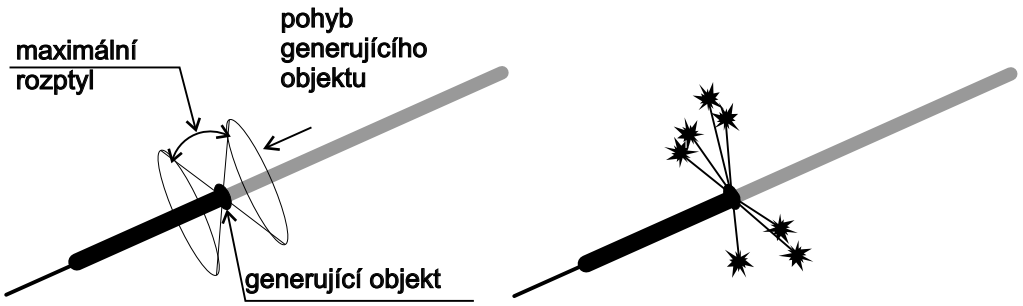
Systém částic je reprezentován jako soubor velmi malých prvků (částic), jejichž vlastnosti se mění v čase. Mezi tyto vlastnosti patří především poloha, dále barva, rychlost a směr pohybu, zrychlení, tvar částice atp. Částice mohou v jistém okamžiku a místě vznikat, po určité době života mizí, mohou emitovat další částice, srážet se mezi sebou a s okolními objekty atp. Částice může být reprezentována prakticky jako cokoliv. Nejčastěji je znázorněna bodem, může být však zobrazena i jako kapka, sněhová vločka, koule či jiný geometrický útvar, vše záleží na jevu, který systém částic modeluje.

Klíčovou roli při tomto modelování opět hrají náhodná čísla. Vlastnosti, které částice mají,

jsou vesměs charakterizovány jednoduchým vztahem

$$x = s + rand \cdot v, \tag{8.10}$$

kde s je střední hodnota hustoty pravděpodobnosti náhodných čísel, v je jejich rozptyl, $rand$ je náhodné číslo a x reprezentuje například dobu života částice, její barvu, atp. Parametry obyčejně interaktivně zadává tvůrce systému částic, stejně jako určuje objekt či plochu, která částice vysílá. Pokud bychom chtěli vytvořit například model prskavky (viz obrázek 8.21), musíme zadat dráhu, směr, kterým se mají částice pohybovat, jejich charakteristiku, barvu a tvar.



Obrázek 8.21: Prskavka jako systém částic. Generující objekt se pohybuje konstantní rychlostí a z něj vyletují částice až do maximálního úhlu, který je určen dvěma kužely. Každá primární částice končí svůj život tím, že emituje skupinu částic, které mají velmi krátkou dobu života (vpravo).

Tvorba jednoho obrázku animované sekvence založené na systému částic se skládá z posloupnosti následujících kroků:

1. Na základě parametrů každé existující částice se aktualizuje její poloha a ostatní atributy.
2. V celém systému se vytvoří nové částice. Ty vznikají buď v nějaké konkrétní oblasti, nebo mohou vznikat jako potomci jiných částic. Oblast, ve které částice vznikají, může být libovolná – mrak, ze kterého prší, děravá hadice, ze které stříká voda atp.
3. Každé nové částici se přiřadí její vlastnosti.
4. Částice, které překročily dobu svého života, nebo jiné hranice, zaniknou.
5. Systém se zobrazí.

8.3.1 Ekosystémy a rostliny

Systémy částic použili Reeves a Blau [Reev83] pro tvorbu modelu lesa. Nejednalo se o dynamickou simulaci, šlo pouze o vytvoření statické scény, skládající se z obrovského množství objektů,

které nebylo možné vytvořit ručně. Postup tvorby této scény je následující. Nejprve se plocha, na které mají být stromy rozloženy, rozdělí na čtverce a určí se střed každého z nich – místo, kde bude umístěna rostlina. Tento bod se ve druhém kroku posune o nějaké náhodné číslo tak, aby neopustil hranici svého čtverečku – tato metoda je známa z adaptivního vzorkování jako algoritmus roztřesení (*jittering*) (viz strana 56). Poté se určité množství (řekněme 10 %) těchto bodů ze scény vypustí. Na každou pozici se pak umístí jednoduchý model stromu.

Obrázek 8.22: Příklad virtuálního ekosystému zobrazeného pomocí kvantizace scény [Bene03]. Původní scéna obsahovala 100 000 rostlin a byla zredukována na pouhých 42 individuí. Scéna bez instancí by zabírala přibližně 20 GB, takto je její velikost 17 MB

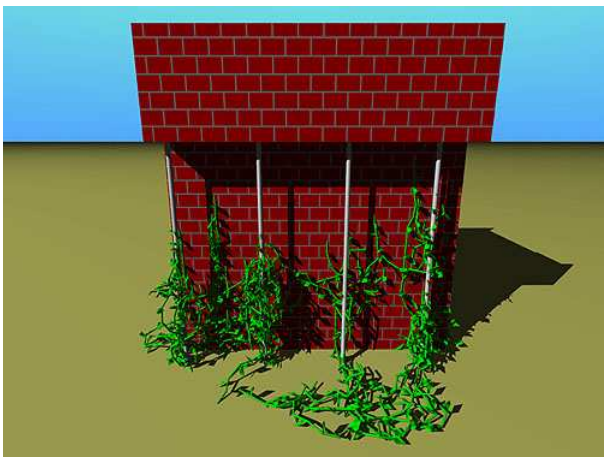
Tato metoda má řadu nedostatků, z nichž nejpodstatnější je, že nerespektuje přirozené shlukování rostlin do skupin. To modelují metody pro tvorbu virtuálních ekosystémů založené na principech umělého života (*artificial life*) [Deus98]. Rostliny jsou nejprve umístěny do ekosystému náhodně. Každá z rostlin roste a existuje v tzv. ekologickém okolí. Tato oblast je určena jako kruh, který se získá projekcí rostliny do země. Jak rostlina roste, její ekologické okolí se zvětšuje. Osud rostliny je určen jednoduchými pravidly. Při kolizi dvou ekologických okolí se určí, která rostlina je silnější, zde se mohou uplatnit pouze lokální, nebo lokální a globální pravidla, a slabší rostlina se ze simulace vyloučí. Po několika iteracích je výsledkem simulace vizuálně věrohodné rozložení rostlin.

Existuje velké množství rozšíření tohoto základního modelu. Jedna z variant nahlíží na simulaci jako na dynamický systém, který může být ovlivňován agenty, kteří do něj vstupují a vnášejí nestability, například se pokoušejí odstranit pouze některý druh rostlin a jiný druh v nějakém místě [Bene03].

Pro zobrazování velmi rozsáhlých modelů se využívají metody, které zmenšují skutečnou velikost generovaných dat. Každá rostlina by měla být jedinečná, ale na výsledných obrazech datech si nikdo nevšimne, že se některá rostlina opakuje. Pro zmenšení množství dat se proto

používají kvantizace, podobně jako při převodu obrazu z velkého množství barev do barevné palety, a jednotlivé instance modelů se pak v obraze opakují. Obrázek 8.22 [Bene03] demonstruje takový příklad. Obraz byl generován metodou sledování paprsku a maximální operační paměť nezbytná pro její zobrazení byla asi 1 G.

Další aplikací systémů částic je generování modelů rostlin, které se přizpůsobují vnějším podmínkám [Arvo88, Gree89]. Tyto modely se obyčejně získávají simulací růstu. Příkladem je břečťan pnoucí se po zdi, jak je vidět na obrázku 8.23 [Bene02]. Pro získání takového modelu je aplikace interaktivních metod modelování obtížná a je jednodušší nechat takový model simulovat metodami umělého života. Na začátku se definuje scéna. Určí se, po jakých objektech se může rostlina pnout a do scény se umístí několik částic. Každá z nich se pohybuje a její dráha se zobrazí jako větev. Po určité době částice vygeneruje další částici; dojde k větvení. Pohyb částice je sice náhodný, ale zároveň částečně řízený. Částice nejprve náhodně otestuje několik náhodně zvolených pozic ve svém okolí a z nich vybere místo, které je nejvýhodnější, tj. kde je nejvíce živin a dostatek světla. Výsledkem je rostlina pnoucí se po zdi, či hledající výhodné místo tak, jak je ukázáno na zmíněném obrázku.

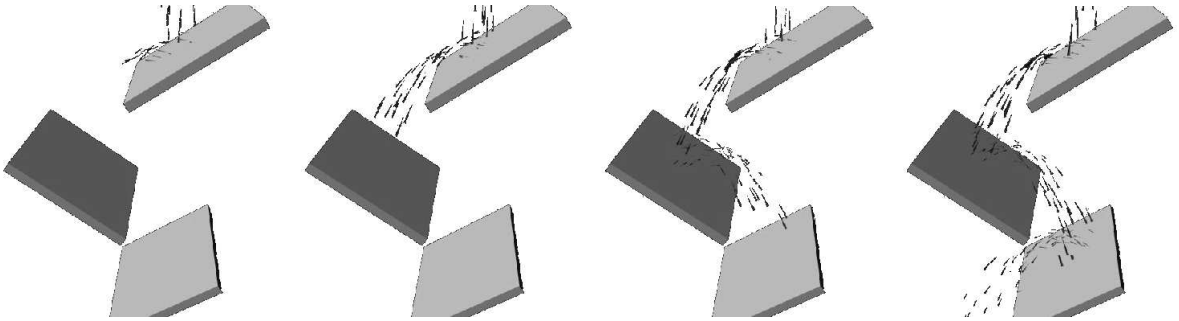


Obrázek 8.23: Virtuální rostlina získaná simulací růstu [Bene02]

8.3.2 Dynamické simulace

Důležité aplikace systémů částic se nachází v dynamických simulacích, které využívají fyzikálních modelů pohybu a interakcí. Jednoduchým příkladem takové simulace je vysypaný koš míčů na schodech. Generujícím objektem je koš, který udělil všem částicím (míčům) počáteční rychlost a směr. Každý míč má určitou pružnost a hmotnost, a pohybuje se po schodech směrem dolů. Míče se mohou mezi sebou srážet, odráží se od zábradlí a přilehlých zdí, apod. Několik snímků z podobného typu simulace je na obrázku 8.24.

Přestože se jedná o algoritmus systémů částic, je natolik specifický a důležitý, že ho uvedeme podrobněji. Jeho jádrem je totiž na fyzice založená simulace. Stav $\vec{y}(t)$ v čase t každé částice je určen jako uspořádaná dvojice $\vec{y}(t) = [X(t), \vec{v}(t)]$, kde $X(t) = [x(t), y(t), z(t)]$ je poloha částice



Obrázek 8.24: Několik snímků z dynamické simulace pomocí systému částic

a $\vec{v}(t) = (v_x(t), v_y(t), v_z(t))$ je vektor její okamžité rychlosti. Částice má hmotnost m, která je v čase neměnná.

Změna polohy a rychlosti se určí podle diferenciálních rovnic

$$\frac{dX(t)}{dt} = \vec{v}(t),$$

$$\frac{d\vec{v}(t)}{dt} = \frac{\vec{F}(t)}{m},$$

kde $\vec{F}(t)$ je vektor sil, působících na částici. Nejjednodušší metodou řešení tohoto systému rovnic je aplikace Eulerovy metody, která převede celý systém na

$$X(t + \Delta t) = X(t) + \Delta t\ \vec{v}(t) \tag{8.11}$$

$$\vec{v}(t + \Delta t) = \vec{v}(t) + \Delta t \frac{\vec{F}(t)}{m}. \tag{8.12}$$

Postup simulace výpočtu polohy částice je pak následující. Nejprve se vynuluje čítač sil, který je přiřazen každé částici, a do něj se sečtou všechny síly na částici působící. Nejčastější silou je zemská přitažlivost, která je charakterizována gravitační konstantou $\vec{g} \doteq 9.82\ [ms^{-2}]$:

$$\vec{F_g} = m\vec{g}.$$

Dále pak viskozita

$$\vec{F_v}(t) = \vec{v}(t)c,$$

kde c je konstanta udávající viskozitu prostředí ($c > 1$ zvyšuje odpor prostředí s rychlostí). Další může být například vítr $\vec{F_w}$, který je konstantní, nebo se může měnit v čase $\vec{F_w}(t)$.

Výslednice sil se použije v rovnici (8.12) k výpočtu nové rychlosti částice. Tento výsledek se pak v rovnici (8.11) aplikuje na výpočet nové polohy. Eulerova metoda je sice implementačně jednoduchá, ale není příliš výhodná pro řešení dynamických systémů, což se projevuje jako oscilace částic v případě, kdy by měly být v klidovém stavu.

Posledním problémem je detekce kolizí částic s objekty a částic mezi sebou. Nejjednodušším řešením je reprezentovat objekty v ploškové reprezentaci a každou rovinnou plošku popsat bodem P a normálovým vektorem $\vec{n}$. Rovina je pak jednoznačně určena dvojicí $(P, \vec{n})$. Kolize částice s rovinou se snadno rozpozná, neboť dojde ke změně znaménka skalárního součinu $\vec{n}.(p - X(t))$. Pokud platí podmínka

$$sign(\vec{n}.(P - X(t))) \neq sign(\vec{n}.(P - X(t + \Delta t))$$

došlo ke kolizi s rovinou, ne však nutně se zadanou ploškou v této rovině – k tomu je zapotřebí složitější test. Místo srážky se určí snadno jako průsečík úsečky zadané body $X(t)$ a $X(t + + \Delta t)$ s rovinou $(P, \vec{n})$. Pro malé Δt lze dokonce obětovat přesnost výpočtu a použít přímo polohu $X(t)$.

Po detekci kolize zbývá určit změnu směru vektoru rychlosti po odražení od roviny. Ten se získá z rozkladu vektoru rychlosti na rovnoběžnou a kolmou složku vzhledem k normálovému vektoru roviny. Předpokládáme vektor rychlosti orientovaný směrem k rovině. Označme vektor odrazu jako $\vec{v}'$. Získáme ho jako součet tečné složky $\vec{v}_t$ a obrácené složky normálové $\vec{v}_n$, tedy

$$\vec{v}' = \vec{v}_t - \vec{v}_n.$$

Normálová složka vektoru rychlosti se vypočítá

$$\vec{v}_n = \vec{n}(\vec{v}.\vec{n}),$$

kde člen v závorce je skalární součin. Konečně tečná složka vektoru se určí z vektoru rychlosti a normálové složky jako

$$\vec{v}_t = \vec{v} - \vec{v}_n.$$

Fyzikálně založený systém částic uvedený v tomto odstavci je základem mnoha složitějších systémů, z nichž asi nejpodstatnější jsou systémy částic aplikované na simulaci oblečení. Fyzikálně založené modelování je základem mnoha počítačových her a aplikací počítačové grafiky. Pro další studium doporučujeme knihu [Bour02].

8.3.3 Jiné aplikace systémů částic

Jiným příkladem systému částic je simulace bublinek v pivě. Generující objekt je vnitřní povrch sklenice, každá částice má jinou velikost a všechny se pohybují směrem vzhůru. Tyto

částice zanikají na hladině. Jinou aplikací systémů částic je generování pěny a vodních spršek vodopádů [Peac86].

Částice se nepoužívají pouze k modelování obtížně tvarovatelných objektů, ale jsou také nástrojem pro fyzikální simulace. Metoda zvaná *trasování částic* (*particle tracing*) se používá zejména tam, kde je nutno zobrazit nějaké složité fyzikální pole, není třeba tento výpočet provádět příliš přesně a navíc není na výpočet mnoho času. Tyto algoritmy se tedy používají zejména ve virtuální realitě. Příkladem složitého fyzikálního pole je větrný tunel, sloužící k simulaci tření prototypů aut či letadel. Simulace probíhá tak, že virtuální model auta či letadla je umístěn do simulovaného větrného tunelu, do něhož jsou vypouštěny částice. Systém zobrazuje dráhu každé částice a v místě, kde například dochází k velké změně dráhy, a tedy i k velkému tření, se dráha zbarví červeně. Podle různě barevných čar pak uživatel takového systému snadno pozná, kde jsou kritické části jeho modelu.

Orientované částice jsou částice, které mají svůj vlastní lokální souřadnicový systém a jsou schopny se vzájemně mezi sebou orientovat v prostoru. Příkladem takové orientace je nasměrování osy x lokálního souřadnicového systému k nejbližší částici. Vhodným stanovením podmínek je možné docílit toho, že se orientované částice v prostoru určitým způsobem rozprostřou. Některé z jejich os pak určují normálu k povrchu, který lze zobrazovat klasickými technikami (např. Phongovým stínováním ze strany 333). Tyto techniky nacházejí své uplatnění například při polygonizaci implicitních ploch.

8.4 Lindenmayerovy systémy

L-systémy (Lindenmayerovy systémy) [Prus90] jsou matematický formalismus vycházející ze systémů paralelního přepisování řetězců (*parallel string rewriting systems*). Podstatou tohoto mechanismu je paralelní přepisování řetězců – posloupnosti symbolů – podle určitých pravidel. Po několika přepsáních se získaná posloupnost symbolů interpretuje geometricky. Každý symbol má přiřazen jistý geometrický význam, například transformaci či generování objektu, či akci.

V tomto textu se budeme nejprve zabývat nejjednoduššími dL-systémy a poté uvedeme otevřené L-systémy [Měch96]. L-systémy se uplatňují především při simulaci rostlin. Kromě této oblasti byly použity i pro modelování říčních toků, modelování mořských mušlí, křivek definovaných dělicími schématy, ale i pro generování silniční sítě ve městě. Umožňují též vytváření klasických lineárních deterministických fraktálů.

8.4.1 dL-systémy

Deterministický bezkontextový L-systém (někdy označován d0L-systém[5]) je uspořádaná trojice

$$G = \langle V, P, S \rangle ,$$

kde

- V je konečná abeceda symbolů $A, B, \ldots$,
- P je konečná množina pravidel tvaru $A \to B;\ A \in V;\ B \in V^*$
- S je axiom: neprázdná posloupnost symbolů pro kterou platí $S \in V^+$.

Nula v názvu d0L-systému znamená, že se jedná o tzv. bezkontextový přepis. Determinismus dL-systému pramení z podmínky, že v množině P nesmějí být dvě pravidla se stejnou levou stranou.

Derivace ($\Rightarrow$) řetězce $A \in V^*$ znamená paralelní přepsání všech symbolů $X \in A$ řetězce V^* na pravé strany pravidel z množiny P. Například první dvě derivace L-systému G :

$$G = \langle \{A, F, -, +\}, \{A \to F - -F - -F, F \to F + F - -F + F, + \to +, - \to -\}, A \rangle ,$$

se startovním symbolem A mají podobu

$$A \to F - -F - -F \Rightarrow F + F - -F + F - -F + F - -F + F - -F + F - -F + F.$$

Posloupnost symbolů se poté geometricky interpretuje. K tomu se používá tzv. *želví grafika* (*turtle graphics*), ve které želva reprezentuje pomyslné kreslicí zařízení.

Želva je definována svým stavem a tabulkou akcí. Stav se skládá ze dvou prvků, z polohy želvy – (bodu) a její orientace. Orientace ve trojrozměrném prostoru se skládá ze tří vektorů $\vec{H}$, $\vec{U}$, $\vec{L}$ podle notace používané v knize [Prus90], které spolu s polohou tvoří lokální souřadnicový systém želvy. Jejich význam je následující:

- $\vec{H}$-(*heading*) – směr, kterým se želva pohybuje kupředu,
- $\vec{U}$-(*up*) – směr, ve kterém má krunýř a
- $\vec{L}$-(*left*) – na které straně má levou přední (i zadní) nožičku.

Želva čte postupně řetězec a pomocí tabulky akcí interpretuje jednotlivé symboly jako příkazy. Jedním z prvních kroků při specifikaci L-systému je tedy definice tabulky významů jednotlivých symbolů abecedy V.

Doplňme náš předchozí příklad o geometrii tak, že želvu umístíme do počátku souřadnicového systému takovým způsobem, aby její orientace byla totožná s vektory jeho os. Význam

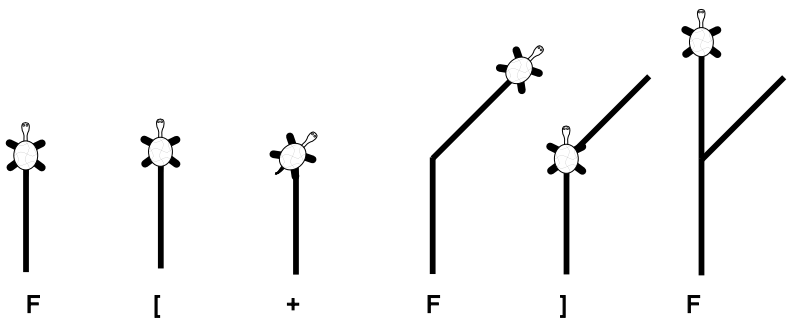
[5] Čteme [dé nula el systém].

jednotlivých symbolů je následující: F znamená posun dopředu o krok d a současně malování úsečky. Symboly + a − jsou interpretovány jako otočení o 60° doprava resp. doleva. Takto doplněný L-systém z předchozího příkladu generuje Kochovu sněhovou vločku (obrázek 8.4). Při trojrozměrném modelování se navíc používají následující symboly: &,^ – rotace dolů a nahoru podle vektoru $\vec{L}$ a /,\ – rotace doleva a doprava podle vektoru $\vec{H}$.

Při definici L-systému nebývá zvykem explicitně vypisovat pravidla, která přepisují symbol na stejný symbol, např. $+ \rightarrow +$. Od tohoto okamžiku budeme tuto dohodu v dalším textu respektovat.

Důležitým krokem v teorii L-systémů bylo zavedení tzv. *závorkových L-systémů* (*bracketed L-system*). V těchto systémech je želva obohacena o zásobník, do kterého může ukládat svůj stav, případně ho vyzvednout a zorientovat se podle něj. Dvěma symbolům abecedy V, označeným [a], jsou přiřazeny funkce pro manipulaci se zásobníkem. Symbol [se interpretuje jako uložení stavu na zásobník a symbol] znamená vyzvednutí posledního uloženého stavu z vrcholu zásobníku a změnu polohy a orientace na hodnoty, které odpovídají situaci v okamžiku interpretace symbolu [. Tím je želva vybavena zásobníkovou pamětí.

Obrázek 8.25 demonstruje interpretaci řetězce $F[+F]F$. Nejprve se želva posune dopředu (symbol F) zapamatuje si svůj stav (symbol [) otočí se doprava a posune dopředu, přečte stav ze zásobníku (symbol]) skokem se vrátí zpět a pokračuje v původním směru.

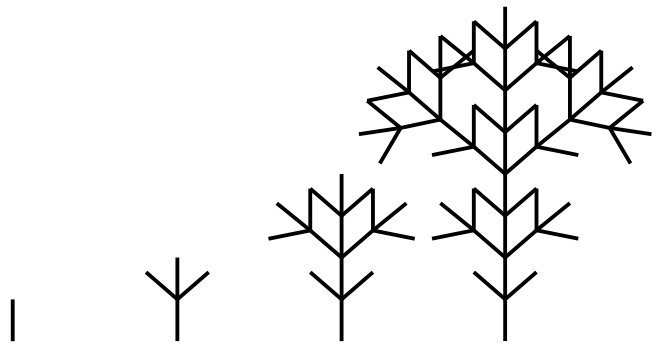


Obrázek 8.25: Interpretace řetězce F[+F]F

Závorkové L-systémy umožňují generovat rozvětvené struktury, jako jsou stromy a keře. To, co je uzavřeno do závorek, určuje samostatnou část objektu. Na obrázku 8.25 se jedná o dva symboly $+F$, které definují větev.

Jiným příkladem (na obrázku 8.26) je L-systém

$$G = \langle \{F, +, -,], [\}, \{F \rightarrow F[+F][-F]F\}, F \rangle \, .$$



Obrázek 8.26: Větvička vytvořená dL-systémem (viz text)

8.4.2 Otevřené L-systémy

Otevřené L-systémy (*Open L-systems*) [Měch96, Prus94], přesněji řečeno nedeterministické kontextové parametrické L-systémy, byly navrženy především pro potřeby simulace růstu rostlin. Jejich mechanismus umožňuje propagaci biologických signálů od kořenů k listům a zpět. Otevřenost spočívá v jejich možnosti interagovat s prostředím. Tato interakce je obousměrná, jak směrem do L-systému, tak ven. Směrem dovnitř je možno například informovat přepisovací proces o detekci kolizí s překážkami, nebo o množství dopadajícího světla, o kyselosti půdy, či o přítomnosti hmyzu. L-systém může informovat okolí o svém rozložení v prostoru, o množství látek, které z něj vycházejí (např. CO_2), atp.

Pro zavedení otevřených L-systémů musíme nejprve popsat tři další rozšíření, a to stochastické, kontextové a parametrické L-systémy.

1. *Stochastický L-systém* umožňuje, aby v množině P bylo více pravidel se stejnou levou stranou. Pravidla z množiny P se pak zapisují ve tvaru

$$A \rightarrow B : pp,$$

   kde pp je pravděpodobnost, že symbol A bude přepsán právě tímto pravidlem. Součet pravděpodobností pravidel se stejnou levou stranou musí být roven jedné.

2. *Kontextový L-systém* má pravidla rozšířena tak, že se při přepisování symbolů uvažuje posloupnost symbolů na levé a pravé straně přepisovaného symbolu jako kontext. Množina pravidel P z definice v odstavci 8.4.1 je rozšířena takto:

$$lc < A > rc \rightarrow B,$$

kde $lc \in V^*$ resp. $rc \in V^*$ je levý, resp. pravý kontext, $A \in V$ je symbol (předchůdce, přepisovaný symbol) a $B \in V^*$ je pravá strana pravidla. Symbol A bude přepsán pravou stranou pouze, když stojí uprostřed kontextu. Příkladem takového pravidla je

$$XY < A > CDE \rightarrow AAA$$

a úspěšný přepis symbolu A může mít tvar

$$\ldots ZAXYACDEF \Rightarrow ZAXYAAACDEF \ldots$$

3. *Parametrický L-systém* (pL-systém) pracuje s tzv. parametrickými slovy. Parametrická slova jsou posloupnosti modulů a moduly jsou písmena abecedy rozšířená o parametry. Modul má syntaktickou podobu $A(x_1, \ldots, x_m)$. Množina parametrů modulu může být prázdná, ale musí být konečná. Modul má ve své specifikaci formální parametry, které nabývají hodnot – skutečných parametrů z množiny reálných čísel. Z parametrů je možno tvořit aritmetické a logické výrazy.

   Parametrický L-systém je uspořádaná čtveřice $G = \langle V, \Sigma, \omega, P \rangle$, kde

   - V je abeceda,
   - Σ je množina formálních parametrů,
   - P je konečná množina pravidel,
   - $\omega \in (V \times R^*)^+$ je neprázdné parametrické slovo zvané axiom.

   Pravidla $p \in P$ v parametrických kontextových L-systémech mají tvar:

$$id : lc < pred > rc : cond \rightarrow succ : prob,$$

   kde

   - *id* je návěští – číslo pravidla,
   - *lc*,*rc* je levý a pravý kontext,
   - *cond* (*condition*) je logický výraz nabývající hodnoty 1 nebo 0,
   - *succ* (*successor*) je pravá strana pravidla, a konečně
   - *prob* (*probability*) je pravděpodobnost, s jakou bude toto pravidlo aplikováno při shodě levých stran více pravidel.

Příklad množiny pravidel stochastického parametrického kontextového L-systému vypadá takto [Prus94]:

$$\begin{aligned} \omega: &\quad A(1)B(3)A(5) \\ p1: &\quad A(x) \rightarrow A(x+1) : 0.4 \\ p2: &\quad A(x) \rightarrow B(x-1) : 0.6 \\ p3: &\quad A(x) < B(y) > A(z) : y < 4 \rightarrow B(x+z)[A(y)]. \end{aligned}$$

Stochastická pravidla $p1$ a $p2$ přepíší modul $A(x)$ buď modulem $A(x+1)$, nebo modulem $B(x-1)$. Kontextově závislé pravidlo $p3$ nahradí modul $B(y)$ posloupností modulů $B(x+z)[A(y)]$, pokud se modul $B(y)$ vyskytuje v kontextu, kde je zleva $A(x)$ a zprava $A(z)$. Toto pravidlo se navíc aplikuje pouze v případě, že je splněna podmínka $y > 4$. První derivace tohoto L-systému může mít například tvar

$$A(1)B(3)A(5) \Rightarrow A(2)B(6)[A(3)]B(4).$$

Při interpretaci čte želva postupně jednotlivé moduly a hodnoty jejich parametrů interpretuje geometricky. Parametr tak může například určovat krok, o který želva popojde, tloušťku čáry, kterou vygeneruje, úhel natočení, atp.

Otevřené L-systémy jsou parametrické kontextové stochastické L-systémy rozšířené o tzv. *komunikační moduly* tvaru

$$?E(x_1, \ldots, x_m),$$

které slouží k přenášení informací mezi posloupností modulů a okolím objektu, který L-systém reprezentuje.

Před vlastním procesem přepsání modulů se provede jakýsi mezikrok, ve kterém se získávají hodnoty skutečných parametrů komunikačních modulů $?E(x_1, \ldots, x_m)$. Na začátku tohoto kroku jsou skutečné hodnoty parametrů komunikačních modulů neznámé. Posloupnost modulů je nejprve prohlížena zleva doprava a vypočítává se stav želvy v prostoru s tím, že není vytvářen geometrický model. Při nalezení komunikačního modulu, je vytvořena zpráva a ta je předána prostředí. V této zprávě je obsažen dotaz na hodnoty parametrů. Okolí tyto parametry nastaví (stav želvy je znám) a L-systém pokračuje v prohlížení posloupnosti modulů. Poté, kdy jsou nastaveny parametry všech komunikačních modulů, začíná fáze přepisování modulů, která je shodná jako u parametrických L-systémů.

Pravidla následujícího otevřeného L-systému například zabraňují tomu, aby objekt rostl ve směru y dále, než do vzdálenosti dva:

$$\begin{array}{lll} \omega & : F(0,0)A?E(0) & \\ p1 & : A >?E(y) & : y < 2 \rightarrow F(x, y+1)A \\ p2 & : A >?E(y) & : y \geq 2 \rightarrow \varepsilon. \end{array}$$

Modul $F(x, y)$ namaluje úsečku z aktuální pozice do bodu o souřadnicích $[x, y]$ (pracujeme v rovině). První pravidlo je aplikováno, pokud vrchol A nepřekročil hranici $y = 2$. Pokud k tomu došlo, je aplikováno tzv. e-pravidlo $p2$ (pravidlo s prázdnou pravou stranou), které vrchol A smaže z posloupnosti modulů. Derivace tedy mají tvar:

$$F(0,0)A?E(0) \Rightarrow F(0,0)F(0,1)A?E(1) \Rightarrow$$
$$\Rightarrow F(0,0)F(0,1)F(0,2)A?E(2) \Rightarrow F(0,0)F(0,1)F(0,2)?E(3).$$

8.4.3 Simulace rostlin

L-systémy představují v současné době nejlépe propracovanou formální teorii modelování syntetických rostlin. Následující příklad ukazuje, jak lze pomocí jednoduché množiny pravidel vytvořit model rostoucí rostliny. K úplné definici tohoto L-systému bychom potřebovali specifikovat geometrický popis objektů, jejich závislost na parametrech aj. Mějme parametrický L-systém

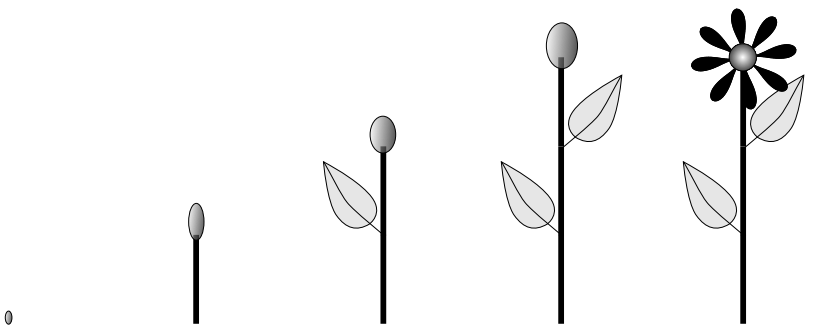
$$\begin{array}{lll} \omega & : A(0) & \\ p1 & : A(x) & : x == 0 \rightarrow FA(x+1) \\ p2 & : F < A(x) & : x == 1 \rightarrow [-L]FA(x+1) \\ p3 & : F < A(x) & : x == 2 \rightarrow [+L]FA(x+1) \\ p4 & : A(x) & : x == 3 \rightarrow B. \end{array}$$

Tabulka akcí pro želvu má následující podobu: F je konstantní krok ve směru vektoru H. Modul $A(x)$ je vrcholový pupen, parametr x je jeho stáří, modul B je květ a modul L je list. Pravidlo $p1$ probudí vrchol a pravidla $p2$ a $p3$ způsobí jeho růst a postupné vyrašení listů nalevo a napravo ve směru růstu. Pravidla $p3$ a $p4$ způsobují přechod vzrostlého vrcholu v květ.

Derivace tohoto L-systému mají následující podobu

$$A(0) \rightarrow FA(1) \Rightarrow F[-L]FA(2) \Rightarrow F[-L]F[+L]FA(3) \Rightarrow F[-L]F[+L]FB.$$

Obrázek 8.27 ukazuje výslednou rostlinku.



Obrázek 8.27: Postupná interpretace následujících řetězců: $A(0)$, $FA(1)$, $F[-L]FA(2)$, $F[-L]F[+L]FA(3)$ a $F[-L]F[+L]FB$

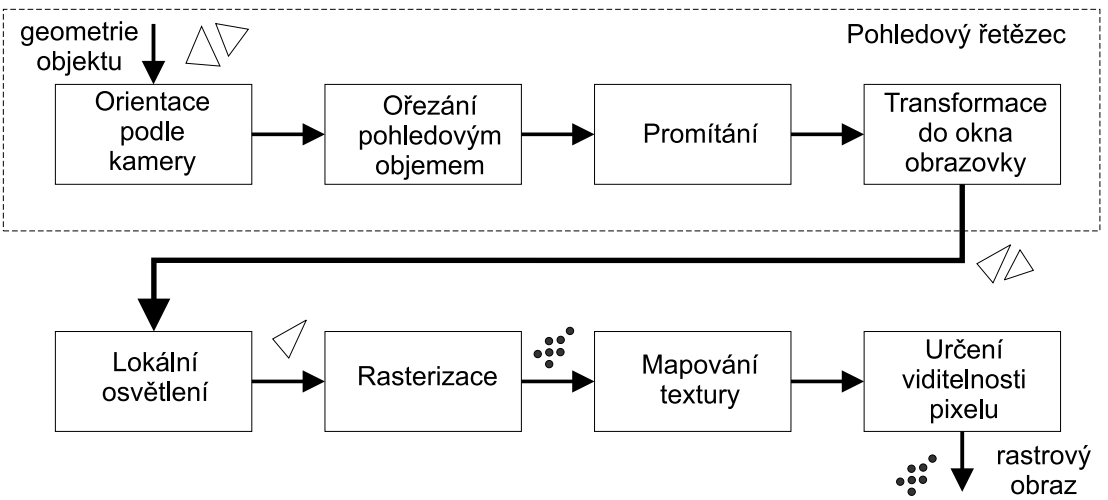
Část C

ZOBRAZOVÁNÍ PROSTOROVÝCH DAT

Zobrazování prostorových modelů a scén reprezentovaných v paměti počítače je důležitou úlohou počítačové grafiky. Převod trojrozměrné informace do dvourozměrné, obrazové podoby se v anglické literatuře nazývá *rendering*, my mu budeme ve většině případů říkat zobrazování. Úlohu zobrazování scény, která obsahuje tělesa a zdroje světla, můžeme zjednodušeně charakterizovat jako postupné řešení následujících dílčích úloh:

- *globální osvětlení scény* závisí především na zdrojích světla a na optických vlastnostech těles a prostředí,
- *pohled na scénu* ze stanoviště pozorovatele, neboli nastavení kamery a řešení promítací úlohy, společně s částečným nebo úplným řešením viditelnosti,
- *vytvoření rastrového obrazu* včetně řešení viditelnosti, lokálních osvětlovacích modelů a textur.

K těmto úlohám se dále připojují různé optimalizační postupy, které redukují rozsáhlé objemy dat před vlastním zpracováním při tvorbě zobrazení. Řešení dílčích úloh může probíhat v různém pořadí, v závislosti na požadované kvalitě a rychlosti tvorby obrazu. Na obrázku 9.1 je ukázáno jedno z možných uspořádání.



Obrázek 9.1: Schéma klasického zobrazovacího řetězce

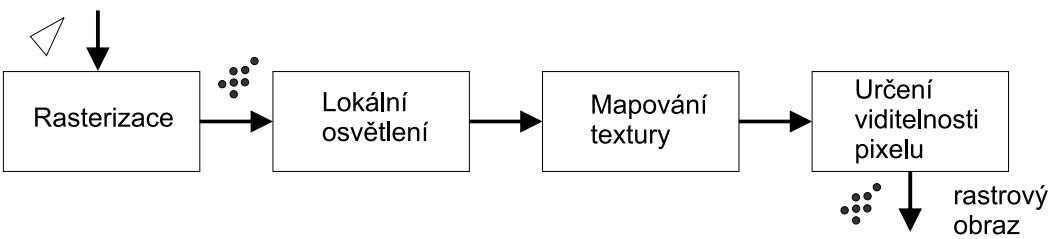
Vstupní proud dat popisujících geometrii objektů (typicky proud trojúhelníků) je podroben pohledové transformaci, poté jsou z dalšího zpracování vyloučeny části, které leží mimo pohledový objem. Následující transformace převede data do souřadnicového systému obrazovky.

Lokální osvětlení řeší barevné stínování na jednotlivých ploškách, které jsou následně rasterizovány a po nanesení textur zaznamenány jako pixely s respektováním viditelnosti. Je-li scéna strukturovaná, jsou objekty získávány systematickým procházením (traverzací) stromů či jiných datových struktur popisujících scénu.

Postupné provádění dílčích operací při zobrazování scény se nazývá *řetězec pohledových transformací* nebo *pohledový řetězec* (*viewing pipeline*), obecněji pak *zobrazovací řetězec* (*rendering pipeline*). Zjednodušeně lze říci, že pohledový řetězec je podmnožinou zobrazovacího řetězce a zajišťuje zejména promítání. Představuje transformace nutné k umístění objektu do takové polohy, ve které je možno efektivně provést určení viditelnosti a rasterizaci.

Některé dílčí kroky tohoto procesu lze provádět souběžně. Při zpracování proudu grafických dat lze například v jednom okamžiku transformovat jeden objekt, přičemž ve stejném čase probíhá rasterizace jiného objektu.

Realizace zobrazovacího řetězce nemusí vždy odpovídat schématu na obrázku 9.1. Uspořádání na obrázku 9.2 se liší „pouze“ v pořadí operací rasterizace a lokálního osvětlení. Tato zdánlivě drobná změna však umožňuje uplatnit široké spektrum programovatelných světelných modelů pro jednotlivé pixely a řešit úlohu lokálního osvětlení pomocí grafických procesorů.

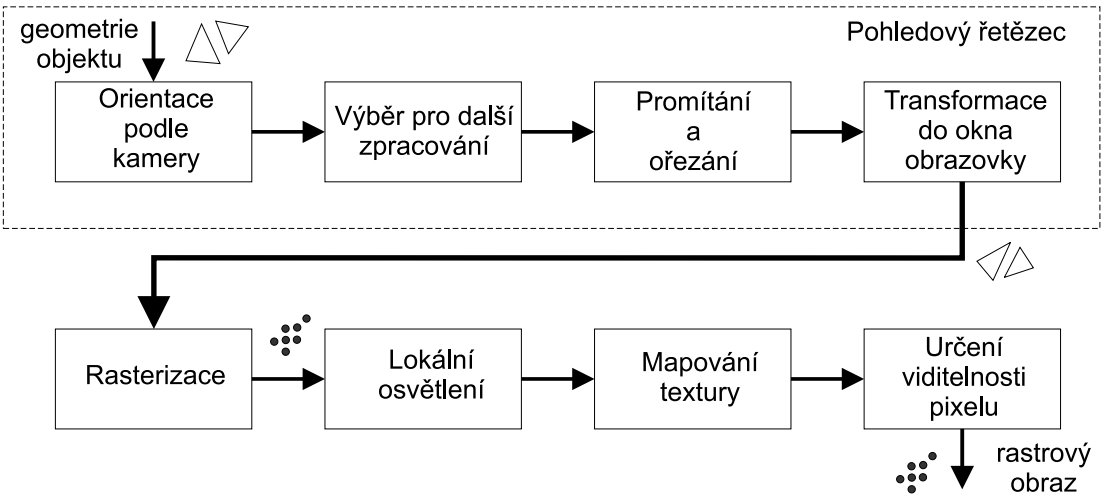


Obrázek 9.2: Část zobrazovacího řetězce s předsunutou rasterizací

Na obrázku 9.3 je pak ukázána další důležitá a v současnosti často realizovaná úprava základního zobrazovacího řetězce. Úloha *výběr pro další zpracování* v sobě zahrnuje řadu přístupů, které se v současné době používají ke snížení počtu zobrazovaných ploch v rozsáhlých scénách. Do této kategorie patří techniky pro odstraňování (*culling*) těch částí těles (plošek), které buď prokazatelně, nebo pravděpodobně neovlivní výsledný obraz.

Uvedená uspořádání zobrazovacího řetězce neřeší výpočet globálního osvětlování scény. Řešení této úlohy totiž vyžaduje mnohonásobný opakovaný přístup k jednotlivým objektům ve scéně a zatím nebyla dekomponována na části kompatibilní se zobrazovacím řetězcem.

Dílčí úlohy zobrazovacího řetězce i další související témata jsou popsány v následujících kapitolách. Za kapitolou o promítacích metodách následuje rozsáhlá kapitola věnovaná světlu a optickým jevům, které ovlivňují tvorbu obrazu. Popíšeme formální model odrazu světla



Obrázek 9.3: Zobrazovací řetězec pro rozsáhlé scény

od povrchu objektu, tzv. dvousměrovou odrazovou distribuční funkci – *BRDF*. Nejdůležitější v této části knihy je matematický model problému osvětlování, tzv. zobrazovací rovnice. Postupně se seznámíme s Monte Carlo metodami vycházejícími od pozorovatele, od zdroje světla a s dvousměrovými metodami. Pozornost je věnována dvěma výpočetně náročným metodám – sledování paprsku a radiační metodě.

Cílem řešení viditelnosti, o kterém pojednává samostatná kapitola knihy, je určit objekty viditelné, nebo částečně viditelné z určitého místa v prostoru. Jednoduše formulovaná úloha je překvapivě složitá a její aplikace jsou k nalezení v počítačové grafice prakticky všude.

Stíny se v metodách globálního osvětlování vypočítávají automaticky. V metodách rychlého zobrazování jsou natolik důležité pro realističnost scény, že je zapotřebí k jejich výpočtu použít specifické algoritmy, o kterých pojednává další kapitola.

Textury a jejich aplikace mají značný podíl na zvýšení realističnosti objektů v počítačové grafice. Popisem jejich reprezentace, tvorby, uložení a aplikace pokračuje další kapitola.

Prostorové objekty ve scéně se zaznamenávají do datových struktur, které umožňují jejich rychlé zpracování. Základní metody organizace objektů v podobě grafu scény i pokročilé techniky uspořádání trojrozměrného prostoru jsou uvedeny v následující kapitole.

Samostatná část je věnována vizualizaci. Je to oblast počítačové grafiky, která se snaží převést obecná, vícerozměrná a tedy i negrafická data do obrazové podoby, znázornit tak jejich vlastnosti, vztahy, tendence, odchylky a tím umožnit uživateli je pochopit.

Celý oddíl uzavírá kapitola o nefotorealistickém zobrazování, které se používá například v počítačových animacích, či v technickém kreslení.

Kapitola 9

Promítání

V počítačové grafice zobrazujeme trojrozměrné objekty na dvojrozměrných zobrazovacích zařízeních. Transformace, která charakterizuje převod trojrozměrného objektu do dvojrozměrné reprezentace, se nazývá promítání. Při promítání dochází ke ztrátě prostorové informace a tím i k možnému zkreslení názoru pozorovatele na skutečný tvar objektu. Proto jsou pro určité obory vybírány různé způsoby promítání a jsou doplňovány dalšími pravidly a postupy pro zvýšení reálného vjemu promítnutého objektu (řešení viditelnosti těles při zobrazování, ortogonální průměty, axonometrie apod.).

Studiem promítacích metod se zabývá deskriptivní geometrie. Ta rozlišuje celou řadu postupů určených k tomu, abychom i z dvourozměrného obrázku získaného promítáním mohli zpětně odvodit různé prostorové vztahy, např. vzdálenosti a úhly. V počítačové grafice se používají především prakticky orientované promítací metody, jejich popis nalezne čtenář např. v [Fole90], [Watt92]. Nejprve připomeneme některé základní pojmy:

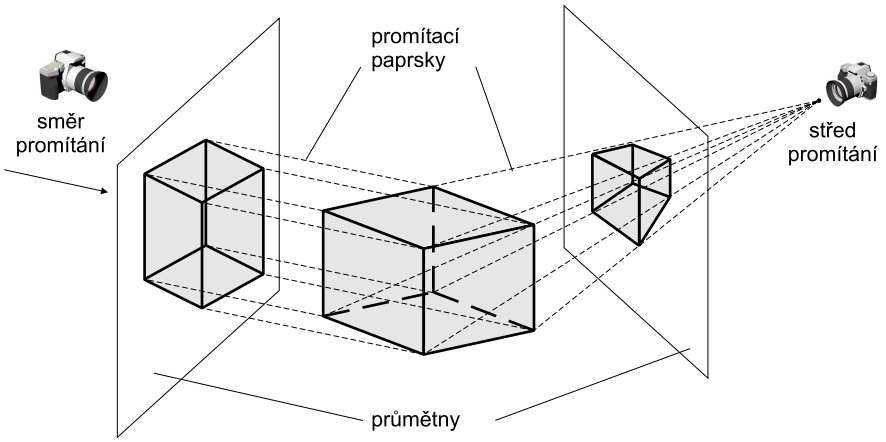
– *promítací paprsek* je polopřímka vycházející z promítaného (prostorového) bodu, jejíž směr závisí na zvolené promítací metodě;

– *průmětna* (*viewing plane*) je plocha v prostoru, na kterou dopadají promítací paprsky a v místě dopadu vytvářejí průmět (obraz v průmětně).

  Pokud je průmětna rovinná, prostorové úsečky se promítají do úseček v rovině – transformace je nezakřivuje. Nemusíme tedy promítat všechny body prostorových objektů. V případě úseček (tvořících často hrany těles), stačí podrobit promítání pouze koncové body a ty spojit v průmětně. Úsečka v průmětně bývá poté zpravidla rasterizována.

V dalším textu se omezíme pouze na promítání do rovinné průmětny. Rovinné promítání dělíme na dvě základní třídy – rovnoběžné (paralelní) a středové (perspektivní). Rovnoběžné

promítání je charakterizováno *směrem promítání* (všechny promítací paprsky mají stejný směr), středové promítání je určeno *středem promítání* (promítací paprsky vycházejí z jediného bodu), jak symbolicky naznačuje ukazuje obrázek 9.4. Pro oba druhy promítání lze volit umístění rovinné průmětny v prostoru.



Obrázek 9.4: Objekt a jeho průmět vzniklý promítáním rovnoběžným (vlevo) a středovým (vpravo)

Úlohu promítání můžeme rozložit na tyto kroky:

1. volba souřadnicových systémů – světový souřadnicový systém WCS (*WCS – World Coordinate System*) a souřadnicový systém průmětny VCS (*VCS – Viewing Coordinate System*);
2. formulace promítací úlohy, tj. výběr geometrických parametrů (bodů, vektorů, vzdáleností, souřadnicových systémů), s jejichž pomocí lze jednoznačně určit pozici pozorovatele (kamery), průmětny, směru promítání aj.;
3. stanovení transformace, která popíše promítání prostorových bodů do průmětny;
4. nalezení transformace mezi souřadnicovými systémy WCS a VCS a její vyjádření v maticovém tvaru.

Cílem je nalezení jedné nebo více transformačních matic, které definují transformace souřadnic promítaných bodů podle vztahu

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ w_p \end{bmatrix} = \mathbf{T}_{proj}. \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

kde $P_p = [x_p, y_p, z_p, w_p]$ jsou VCS souřadnice promítaného bodu $[x, y, z, 1]$ a $\mathbf{T}_{proj}$ je matice 4×4 použitá při transformaci promítání. Postupné transformace (více transformačních matic) se používají tehdy, když jsou během promítání prováděny ještě další výpočty (např. ořezávání) a tyto nelze nebo není vhodné řešit v cílovém souřadnicovém systému průmětny.

Při zobrazování prostorových objektů následuje po promítání další zpracování dat – nalezení zakrytých a viditelných částí objektů, vyhodnocení jejich barvy, nanesení textury. Z důvodu snazšího zpracování těchto kroků se stalo zvykem používat jako průmětnu rovinu xy, resp. jakoukoliv rovinu kolmou na osu z.

Také my se přidržíme této zvyklosti a budeme metody promítání uvádět s ohledem na průmětnu rovnoběžnou s rovinou xy. Promítání do jiné průmětny můžeme vždy převést pomocí transformací (nejčastěji otočení a posunutí) prostorových objektů na uvedený případ, jak bude ukázáno v části 9.4. Problematika transformací je popsána v kap. 21.

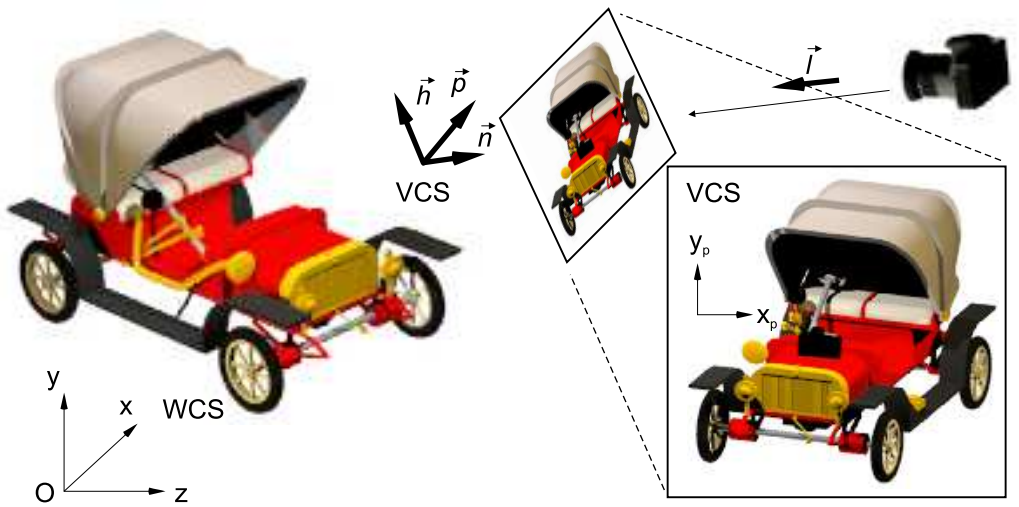
9.1 Kamera

Základem promítací úlohy je formulace geometrické situace, tj. určení místa, kde stojí pozorovatel, vymezení pozice a orientace průmětny a stanovení směru a cíle pozorování. Kamery totiž pořizují snímky na průmětnu, která je kolmá na hlavní optickou osu kamery. Úhel záběru kamery nebudeme zpočátku brát v úvahu, idealizovaná kamera může snímat buď středovým nebo rovnoběžným promítáním celý poloprostor před kamerou. Ukázka promítací úlohy s obecně umístěnou kamerou v prostoru je na obrázku 9.5.

Předpokládejme, že kamera je umístěna v prostoru v bodě se světovými souřadnicemi $[kamera_x, kamera_y, kamera_z, 1]$. Ve scéně zvolíme bod $[cil_x, cil_y, cil_z, 1]$, na který kameru namíříme. Tím je určen směr pozorování $\vec{L} = (L_x, L_y, L_z, 0)$, který je rovnoběžný s hlavní optickou osou kamery.

$$\begin{bmatrix} L_x \\ L_y \\ L_z \\ 0 \end{bmatrix} = \begin{bmatrix} cil_x \\ cil_y \\ cil_z \\ 1 \end{bmatrix} - \begin{bmatrix} kamera_x \\ kamera_y \\ kamera_z \\ 1 \end{bmatrix}.$$

Po normalizaci $\vec{L}$ dostaneme jednotkový vektor $\vec{l}$. Požadujeme, aby pohledová transformace mezi WCS a VCS převedla tento vektor na $[0, 0, -1, 0]$, tj. na vektor, který je kolmý na průmětnu



Obrázek 9.5: Promítání scény na průmětnu (model auta poskytl L. Stryk, MU v Brně)

(rovnoběžný s osou z_p) a mířící ve směru záporné poloosy z_p:

$$\begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \end{bmatrix} = \mathbf{T}_{kamera} \cdot \begin{bmatrix} l_x \\ l_y \\ l_z \\ 0 \end{bmatrix} .$$

Orientaci kamery (natočení okolo optické osy) určuje vektor $\vec{u}$ (*up vector, hook*), o kterém předpokládáme, že je kolmý na $\vec{l}$ (rovnoběžný s průmětnou) a jednotkový. Pomocí vektorového součinu určíme třetí vektor $\vec{p} = \vec{l} \times \vec{h}$, který ukazuje doprava ve směru osy x_p souřadnicového systému VCS. Až na posunutí je souřadnicový systém VCS vztažený ke kameře určen trojicí vzájemně kolmých jednotkových vektorů $\vec{l}, \vec{h}, \vec{p}$. Obdobně jako v případě zvoleného směru pohledu požadujeme, aby pohledová transformace mezi WCS a VCS převedla vektor $\vec{h}$ na $[1, 0, 0, 0]$ a vektor $\vec{p}$ na $[0, 1, 0, 0]$:

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \mathbf{T}_{kamera} \cdot \begin{bmatrix} p_x \\ p_y \\ p_z \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \mathbf{T}_{kamera} \cdot \begin{bmatrix} h_x \\ h_y \\ h_z \\ 0 \end{bmatrix} .$$

Po sestavení všech podmínek dostaneme na levé straně rovnice jednotkovou matici a hledaná

matice $\mathbf{T}_{kamera}$ musí být tedy inverzní k matici sestavené z vektorů $\vec{l}, \vec{h}, \vec{p}$:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \mathbf{T}_{kamera} \cdot \begin{bmatrix} p_x & h_x & -l_x & 0 \\ p_y & h_y & -l_y & 0 \\ p_z & h_z & -l_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Připomeňme, že všechny vektory, ze kterých je tato matice sestavena, jsou jednotkové a vzájemně kolmé, matice je tedy *ortonormální*. Hledanou matici $\mathbf{T}_{kamera}$ získáme jednoduše transpozicí, tj.

$$\mathbf{T}_{kamera} = \begin{bmatrix} p_x & h_x & -l_x & 0 \\ p_y & h_y & -l_y & 0 \\ p_z & h_z & -l_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} p_x & p_y & p_z & 0 \\ h_x & h_y & h_z & 0 \\ -l_x & -l_y & -l_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Pokud do matice doplníme posunutí, které ztotožní zvolený bod ve scéně s počátkem souřadnicového systému VCS, máme připraven základ pro řešení různých způsobů promítání.

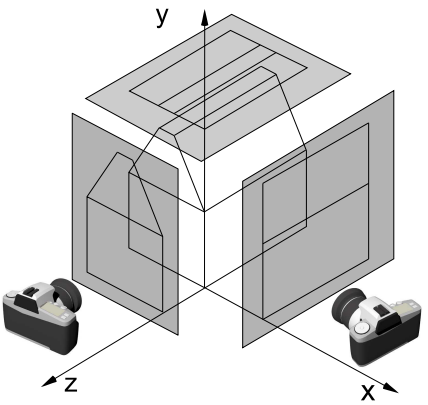
9.2 Rovnoběžné promítání

Nejjednodušším typem rovnoběžného promítání je promítání do některé z rovin $x = x_0$, $y = y_0$, $z = z_0$ ve směru příslušné osy kolmé na zvolenou průmětnu. Velmi často budou x_0, y_0 nebo $z_0 = 0$, průmětnou bude některá z hlavních rovin yz, xz nebo yz. Promítání popíšeme transformací v homogenních souřadnicích, která v sobě zahrnuje i umístění rovnoběžně snímající kamery na příslušnou osu.

$$\mathbf{T}_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{T}_{xz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & y_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{T}_{yz} = \begin{bmatrix} 0 & 0 & 0 & x_0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Skupina promítání ve směru hlavních os do průměten v hlavních rovinách, která zahrnuje nárys a podle potřeby bokorysy, půdorys (pohled shora), spodní pohled a pohled zezadu, se nazývá Mongeovo promítání. Příklad je uveden na obrázku 9.6.

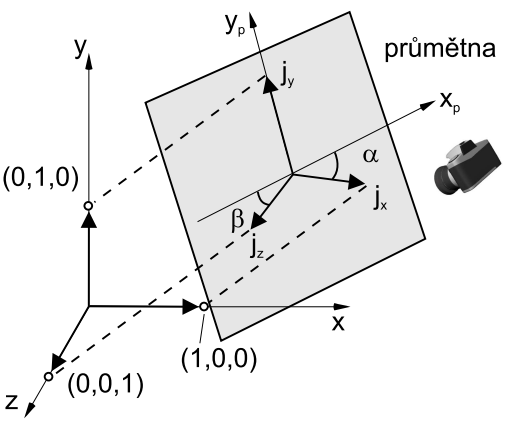
Mongeovo promítání je speciálním případem *kolmého promítání*. Promítací paprsky jsou kolmé na průmětnu a mají směr shodný s normálovým vektorem průmětny.



Obrázek 9.6: Mongeovo promítání

Axonometrie

Axonometrické promítání používá průmětnu, která není rovnoběžná s hlavními osami. Průmětna protíná dvě nebo tři hlavní osy WCS. Tři úsečky v rovině, které mají společný jeden krajní bod a které neleží v přímce, lze pokládat za rovnoběžný průmět tří vzájemně kolmých a stejně dlouhých úseček, které mají jeden krajní bod společný. V souladu s tím je axonometrie často definována pěti hodnotami $j_x, j_y, j_z, \alpha, \beta$, kde j_x, j_y, j_z jsou průměty jednotek na osách x, y, z, α a β jsou úhly, které svírají promítnuté osy j_x, j_z s kolmicí na průmět osy j_y.

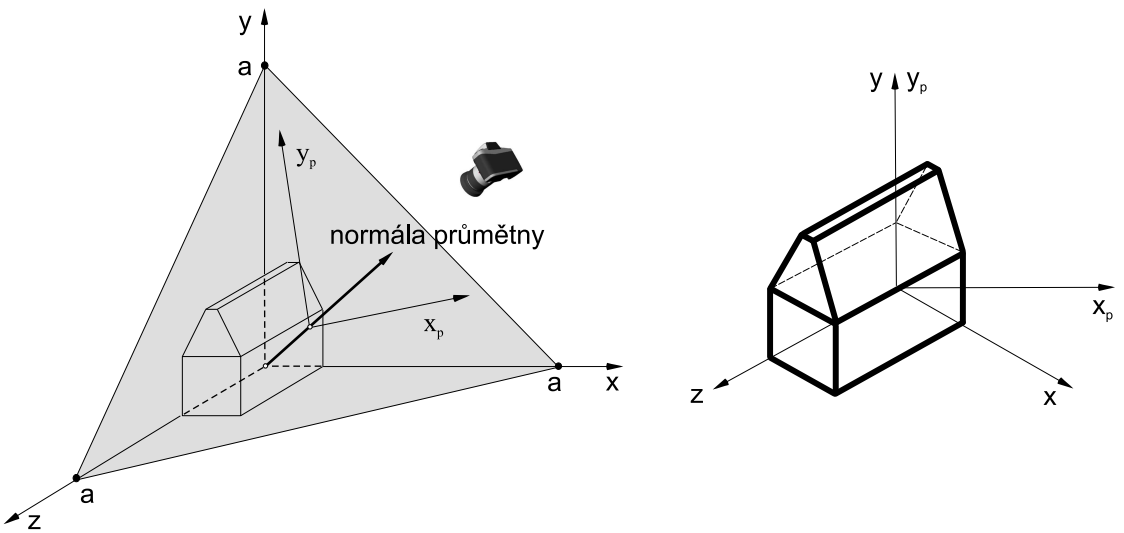


Obrázek 9.7: Zadání pravoúhlé axonometrie

Jestliže souřadnicový systém průmětny (osy x_p, y_p) zvolíme tak, aby průmět j_y ležel na ose y_p (viz obr.9.7)[1], pak axonometrii popíše matice

$$\mathbf{T}_{axon} = \begin{bmatrix} j_x cos\alpha & 0 & -j_z cos\beta & 0 \\ -j_x sin\alpha & 1 & 0 & 0 \\ 0 & 1 & -j_z sin\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Pokud průmětna protne hlavní osy ve stejné vzdálenosti od počátku WCS (svírá stejný úhel se všemi osami), lze v průmětu měřit a porovnávat vzdálenosti, zkreslení vzdáleností je totiž ve všech směrech promítnutých os stejné. Toto promítání se nazývá *izometrie*. Příklad je na obrázku 9.8.



Obrázek 9.8: Axonometrie – izometrické promítání

Mají–li shodnou vzdálenost od počátku pouze dva průsečíky, lze v průmětně měřit jen ve dvou směrech promítnutých os, ve třetím směru jsou zkrácené nebo prodloužené vzdálenosti. Tento případ se nazývá *dimetrie*. Při obecném sklonu průmětny vůči osám hovoříme někdy o *trimetrii*, v každé směru promítnuté osy musíme při porovnávání uplatnit odlišné měřítko. Pokud je směr promítání kolmý na průmětnu, jedná se o *pravoúhlou axonometrii*. Speciální případy axonometrií jsou definovány takto:

[1] V obecném případě musíme doplnit tranformace otočení a posunutí v rovině průmětny.

- isometrie: $j_x = j_y = j_z, \alpha = \beta$,
- dimetrie: $j_x = j_y, \alpha = \beta$,
- trimetrie: $j_x \neq j_y \neq j_z$,
- a technické kosoúhlé promítání: $j_y = j_z = 1, \beta = 0$.

Kosoúhlé promítání

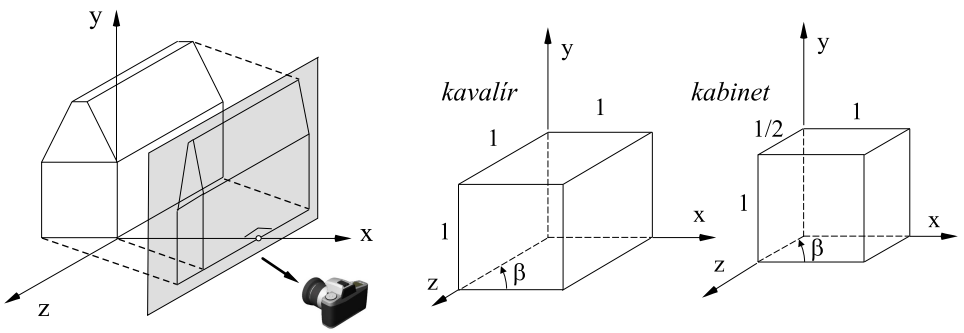
Obecné rovnoběžné promítání se označuje jako *kosoúhlé promítání*. Kosoúhlé promítání kombinuje vlastnosti Mongeova promítání (znázornění některého z hlavních průmětů) s axonometrickým promítáním. Průmětna je rovnoběžná s některou z hlavních rovin, směr rovnoběžného promítání není kolmý na průmětnu. Příklad kosoúhlého promítání je na obrázku 9.9 vlevo. Používá se v technické praxi, zejména v architektuře, protože se zvětšující se vzdáleností od pozorovatele nezkracuje vzdálenosti v rovinách rovnoběžných s průmětnou a poskytuje současně částečný boční pohled na promítaný objekt. Dva nejčastěji používané druhy kosoúhlého promítání se nazývají *kabinet* (obr. 9.9 uprostřed) a *kavalír* (obr. 9.9 vpravo). U kavalírního promítání svírá směr promítání s průmětnou úhel 45°. Výsledkem je, že úsečky rovnoběžné s průmětnou i kolmé na průmětnu se promítají se stejnou délkou. U kabinetního promítání svírá směr promítání s průmětnou úhel $arctan(2) = 63.4°$, úsečky kolmé na průmětnu se zkracují na polovinu. Toto promítání je realističtější než kavalírské, neboť poskytuje představu o zkracování vzdáleností při pohledu směrem do scény. Odpovídající transformace pro oba způsoby promítání jsou:

$$\mathbf{T}_{kavalir} = \begin{bmatrix} 1 & 0 & -cos\beta & 0 \\ 0 & 1 & -sin\beta & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{T}_{kabinet} = \begin{bmatrix} 1 & 0 & -\frac{cos\beta}{2} & 0 \\ 0 & 1 & -\frac{sin\beta}{2} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

9.3 Středové promítání

Středové (perspektivní) promítání odpovídá optickému modelu, který vyjadřuje lidské vidění reálného světa. Modeluje proporcionální zmenšování předmětů při vzrůstající vzdálenosti od pozorovatele. Poskytuje dobrý prostorový vjem na rovinné průmětně. Na obrázku 9.10 je ukázáno promítání bodu $P[x, y, z, 1]$ na bod $P_p[x_p, y_p, z_0, 1]$.

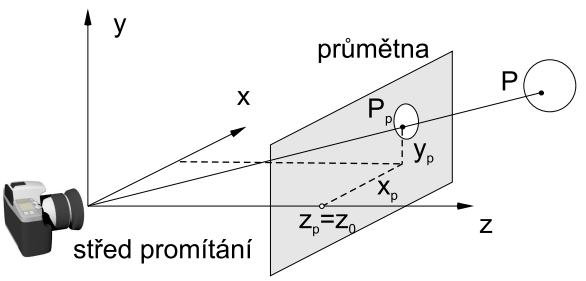
Pozorovatel je v daném případě v počátku souřadnicového systému a průmětna je kolmá na osu z. Na základě trojúhelníkové podobnosti snadno odvodíme, že



Obrázek 9.9: Kavalírské a kabinetní promítání

$$x_p = x.\frac{z_0}{z}\ , \quad y_p = y.\frac{z_0}{z}\ .$$

Vzdálené objekty za průmětnou jsou při zobrazení zmenšeny, objekty v průmětně se promítnou ve své velikosti a objekty před průmětnou se na obraze zvětší. Bod $P = [x, y, z, 1]$ bude transformován na bod $P_p = [x_p, y_p, z_p, w_p]$ pomocí



Obrázek 9.10: Perspektivní průmět bodu P

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ w_p \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{z_0} & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \mathbf{T}_{per}.P.$$

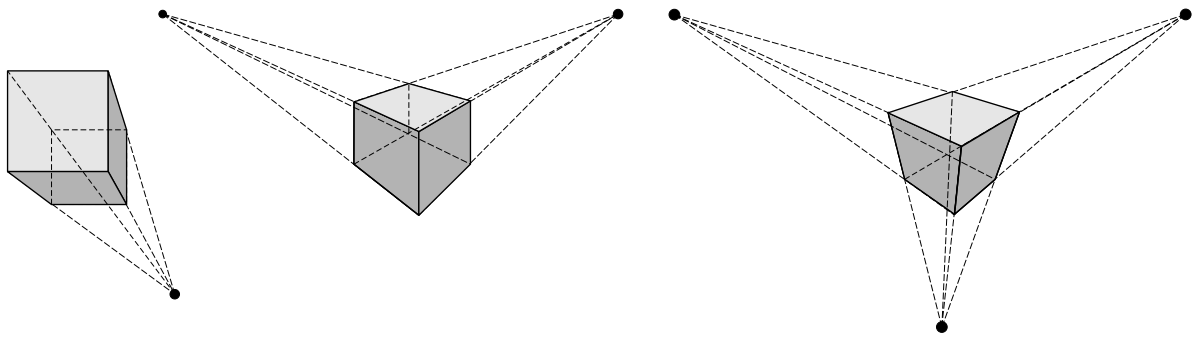
To znamená, že středové promítání vyjádříme v homogenních souřadnicích pomocí lineární transformace. Pokud uvažujeme umístění pozorovatele v bodě $z = z_0$ a průmětnu přemístíme do počátku souřadnicového systému (ztotožníme s rovinou xy), pak

$$x_p \ = \ x \ . \ \frac{z_0}{z_0 \ + \ z} \ = \ \frac{x}{1 \ + \ \frac{z}{z_0}}\ , \quad y_p \ = \ y \ . \ \frac{z_0}{z_0 \ + \ z} \ = \ \frac{y}{1 \ + \ \frac{z}{z_0}}$$

a v maticovém tvaru

$$\mathbf{T'}_{per} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{z_0} & 1 \end{bmatrix}.$$

Charakteristickým rysem středového promítání je, že nezachovává rovnoběžnost úseček. Průměty úseček rovnoběžných v trojrozměrném prostoru jsou obecně mimoběžné. Výjimkou jsou prostorové úsečky, ležící v rovině rovnoběžné s průmětnou.



Obrázek 9.11: Středové promítání jednobodové, dvoubodové a trojbodové

Ačkoliv může mít průmětna libovolnou polohu, z praktického hlediska se rozlišují tři případy, odpovídající orientaci průmětny vůči osám souřadnicového systému:

1. *jednobodová perspektiva* vzniká, když průmětna protíná jedinou souřadnicovou osu. To je případ na obrázku 9.11 vlevo. Všechny úsečky kolmé na průmětnu míří do jediného bodu, který se nazývá *hlavní úběžník*.

2. *dvoubodová perspektiva* vznikne, pokud průmětna protíná dvě ze souřadnicových os. Hrany osově orientovaných kvádrů směřují do dvou hlavních úběžníků, jak ukazuje obrázek 9.11 uprostřed.

3. *trojbodová perspektiva* je nejobecnější případ, který vzniká, pokud průmětna promítá všechny tři souřadnicové osy (obr. 9.11 vpravo). Protažením hran osově orientovaných kvádrů můžeme nalézt tři hlavní úběžníky.

Pokud protíná průmětna osy souřadnicového systému v úběžnících $[-x_0, 0, 0, 1]$, $[0, -y_0, 0, 1]$, $[0, 0, -z_0, 1]$, pak perspektivní transformaci popíšeme maticí

$$\mathbf{T}_{per3} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{1}{x_0} & \frac{1}{y_0} & \frac{1}{z_0} & 1 \end{bmatrix}.$$

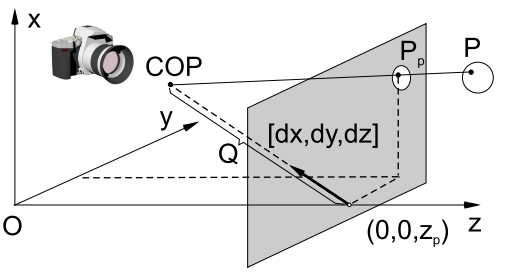
S trochou nadsázky lze říci, že úběžníky reprezentují „nekonečno, v němž se rovnoběžky stýkají“. Poznamenejme také, že úběžníků můžeme při středovém promítání nalézt velké množství.

Pouze ty, které odpovídají hranám osově orientovaných kvádrů, tedy úsečkám rovnoběžným se souřadnicovými osami, se nazývají hlavní. Mají význam v architektuře, protože v ní se používají objekty s hranami a plochami, které jsou rovnoběžné se souřadnicovými osami (podlahy, stěny).

9.4 Jednotné promítání

Každý druh rovinného promítání charakterizuje určitá transformační matice 4×4 pro transformaci bodů $[x, y, z, 1]$ ve scéně na body $[x_p, y_p, z_p, 1]$ v průmětně. Rovnoběžné a perspektivní promítání lze jednotně popsat dále uvedeným postupem, který formuloval N. Weingarten a který je popsán v [Fole90]. Promítání popíšeme následujícími parametry podle obrázku 9.12.

Průmětna je rovnoběžná s rovinou xy, protíná osu z v bodě $[0, 0, z_p]$. Střed promítání COP (*center of projection*) leží na polopřímce vycházející z bodu $[0, 0, z_p]$ ve směru (dx, dy, dz) ve vzdálenosti Q. S ohledem na dále odvozované vztahy předpokládáme, že směrový vektor je jednotkový. Bod P_p, který je průmětem bodu P, určíme z parametrické rovnice úsečky mezi body $COP-P$, tj.



Obrázek 9.12: Parametry obecné promítací úlohy

$$P_p = COP + t.(P - COP) \ .$$

Parametr t získáme z rovnice

$$z_p = (z_p + Q.dz) + [z - (z_p + Qdz)].t \ .$$

Po dosazení t do vztahů pro x_p a y_p a formálních úpravách sestavíme matici

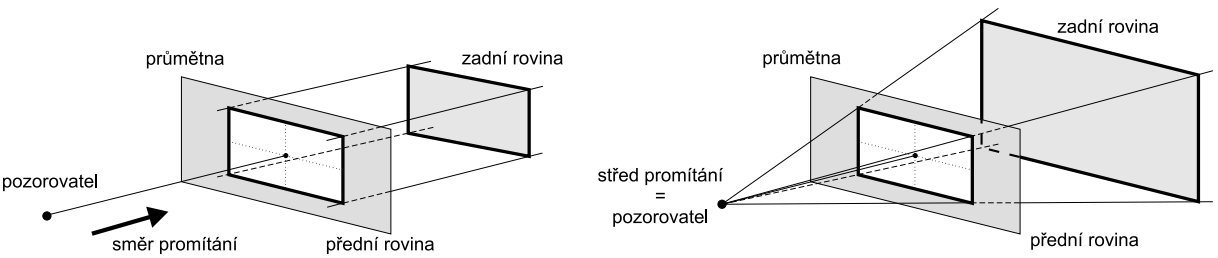
$$\mathbf{T}_{obecna} = \begin{bmatrix} 1 & 0 & -\frac{dx}{dz} & z_p.\frac{dx}{dz} \\ 0 & 1 & -\frac{dy}{dz} & z_p.\frac{dy}{dz} \\ 0 & 0 & -\frac{z_p}{Q.dz} & \frac{z_p^2}{Q.dz} + z_p \\ 0 & 0 & -\frac{1}{Q.dz} & \frac{z_p}{Q.dz} + 1 \end{bmatrix} .$$

Z matice $\mathbf{T}_{obecna}$ dostaneme různé druhy rovnoběžného a perspektivního promítání dosazením příslušných hodnot. Příklady jsou uvedeny v tabulce

	z_p	Q	dx	dy	dz
$\mathbf{T}_{xy}$	0	∞	0	0	-1
$\mathbf{T}_{per}$	d	d	0	0	-1
$\mathbf{T'}_{per}$	0	d	0	0	-1
$\mathbf{T}_{kavalir}$	0	∞	$cos\alpha$	$sin\alpha$	-1
$\mathbf{T}_{kabinet}$	0	∞	$cos\alpha/2$	$sin\alpha/2$	-1

9.5 Pohledový objem

Při promítání je vhodné potlačit zpracování těch objektů, které se nacházejí mimo oblast našeho zájmu. Nejenže tím urychlíme proces vykreslování, ale také tím vyřešíme kritické situace, které mohou vzniknout při středovém promítání. Je zřejmé, že střed promítání nesmí ležet v průmětně, protože při vyhodnocení vztahů pro středové promítání by došlo k dělení nulou. Podobně je třeba zamezit promítání objektů ležících za zády pozorovatele.



Obrázek 9.13: Vymezení pohledového objemu při promítání rovnoběžném (vlevo) a středovém (vpravo)

Proto se zavádí pojem *pohledový objem* (*viewing volume, viewing frustum*), někdy též *záběr*. Je to oblast prostoru, ohraničující ty objekty, které mají být podrobeny promítání. Veškeré ostatní objekty musí být před dalším zpracováním odstraněny, resp. ořezány. Pohledové objemy pro oba základní druhy promítání jsou znázorněny na obrázku 9.13. U středového promítání je objemem komolý jehlan, u rovnoběžného kvádr. Úhly při vrcholu jehlanu by měly odpovídat šíři záběru kamery. To je sice schopno vidět ostře jen ve velmi úzkém vrcholovém úhlu, ale pokud uvážíme i tzv. periferní vidění, lze za vhodný úhel při vrcholu jehlanu považovat $40° - 60°$.

Dvě významné stěny pohledového objemu se nazývají přední (*near*) a zadní (*far*) omezující rovina. Při ořezání pohledovým objemem zajišťují tyto roviny odstranění příliš blízkých objektů bránících ve výhledu a příliš vzdálených objektů, které jsou z hlediska pozorovatele nezajímavé a jejichž zpracování zpomaluje proces zobrazování. Na obrázku 9.13 je poloha přední ořezávací

roviny shodná s polohou průmětny. Průmětna však může být umístěna v libovolné vzdálenosti od pozorovatele, neboť po promítnutí v další fázi pohledového řetězce následuje úprava měřítka, která obraz promítnutých objektů převede do rozměrů určeného okna na obrazovce.

Někdy bývá prostor zobrazovaných objektů nejprve podroben transformaci změny měřítka tak, aby se celý pohledový objem vešel do jednotkové krychle. Tím se zjednoduší výpočty v algoritmech ořezávání.

Kapitola 10

Světlo

Pro počítačovou grafiku jsou fundamentální algoritmy simulující interakci světla s povrchy objektů a s opticky aktivním prostředím. Světlo je médiem vizuálního vnímání světa a pochopení jeho interakce s materiály a putování prostorem je základem pro tvorbu virtuálních scén a jejich zobrazování. Naneštěstí pro čtenáře, tyto jevy nejsou triviální a metody a algoritmy, jež je simulují, jsou založeny na komplikovaných fyzikálních jevech. V této části nejprve uvedeme základní fyzikální a radiometrické veličiny používané při popisu světla a od nich přejdeme k popisu interakce světla s povrchem. Odraz světla od povrchu, modely stínování povrchů a simulace světelných zdrojů nás připraví na jeden z nejdůležitějších pojmů počítačové grafiky, kterým je matematický formalismus zvaný zobrazovací rovnice, která je předmětem následující kapitoly.

Jedněmi z nejlépe propracovaných částí počítačové grafiky jsou oblasti týkající se modelování světla a jeho interakce s povrchy těles. Po dlouhém váhání jsme se rozhodli pro nepříliš důsledné respektování existujících českých pojmů pro některé, dnes již v počítačové grafice zlidovělé termíny. Místo toho budeme počešťovat termíny anglické. Není to proto, že bychom neměli náš jazyk rádi, jde o to, že naprostá většina literatury přichází ke čtenáři prostřednictvím Internetu v jazyce anglickém. Kde je to možné, české termíny samozřejmě uvádíme, ve vlastním textu je však příliš nedodržujeme. Nechceme totiž, aby se čtenář naší knihy musel například pracně pídit, co je míněno termínem hustota dopadajícího světelného výkonu na ploše, když se jedná o obyčejnou iradianci.

Světlo má dualistickou povahu, chová se jako vlny i jako částice. Vlnový popis je vhodný pro vysvětlení jevů jako je disperze světla, difrakce, interference aj. Částicovým popisem se naopak může vysvětlit odraz světla, interakce s drsným povrchem materiálu aj. Optika, nauka o světle, se rozděluje do následujících podoblastí [Jens01]:

- *Geometrická optika* modeluje světlo jako nezávislé paprsky, které putují prostorem a jejich trajektorie lze popsat geometrickými pravidly.
- *Vlnová optika* modeluje elektromagnetické vlny a umožňuje popsat většinu jevů, u kterých popisy geometrickou optikou selhávají, zejména difrakci a interferenci.
- *Elektromagnetická optika* zahrnuje vlnovou optiku a navíc popisuje polarizaci světla a disperzi na hranách.
- *Fotonová optika* (též zvaná kvantová optika) je základem pro vysvětlení interakce světla s materiálem.

V počítačové grafice se téměř výhradně používá geometrická optika a v některých případech i popis světla pomocí částic. Tyto modely umožňují popsat a modelovat naprostou většinu jevů, které jsou podstatné pro vizuální vnímání světa. Počítačová grafika se nezabývá jevy jako je polarizace či difrakce světla, ačkoli tato rozšíření jsou možná a poměrně snadná.

Při simulaci světla používáme v počítačové grafice následující předpoklady a zjednodušení:

1. Světlo se šíří přímočaře.
2. Rychlost šíření světla je nekonečná. Světlo je okamžitě po opuštění světelného zdroje přítomné na všech místech, což způsobuje, že je scéna v ustáleném stavu a nemusíme se zabývat dynamickou simulací pohybu světla. Veškeré odezvy jsou okamžité.
3. Světlo není ovlivněno gravitací, elektromagnetickým polem a neuplatňují se relativistické jevy.

Fotometrie (*photometry*) je nauka, která popisuje vnímání světla a zavádí veličiny, které ho kvantifikují. Lidský aparát pro vnímání světla je citlivý na světlo obsahující záření o vlnové délce v oblasti 380 - 720[1] nanometrů a tato citlivost byla ve fotometrii standardizována. Fotometrické veličiny je možné odvodit z radiometrických (*radiometry*) veličin, které jsou na člověku nezávislé a proto objektivnější. Z tohoto důvodu se v algoritmech globálního osvětlování nejčastěji používají radiometrické veličiny, které uvádíme v odstavci 10.1.2. Fotometrické veličiny jsou k vyhledání v knize [Cohe93].

10.1 Základní pojmy

Viditelným světlem nazýváme úzké frekvenční pásmo elektromagnetického spektra v oblasti 10^{14}Hz. Každá frekvence uvnitř viditelného pásma odpovídá určité barvě. Na spodním konci rozsahu je červená barva (o frekvenci přibližně 4.4 10^{14}Hz čili vlnové délce 720 nm) na horním

[1] Většina literatury se v těchto hodnotách liší. Nejčastěji je citován rozsah 380 - 720 nm, setkáme se i s hodnotami 300 - 780 nm.

je fialové světlo (frekvence 8.3 10^{14}Hz čili 380 nm). Mezi těmito krajními frekvencemi může lidské oko rozlišit až 400 000 různých barev, v jednom okamžiku však maximálně 10 000. Tyto barvy pokrývají barevný rozsah od červené přes oranžovou a žlutou pro nižší frekvence, až po zelenou, modrou a fialovou pro vyšší frekvence, jak ukazuje obrázek 1.1.

Světelné zdroje obyčejně vyzařují fotony na široké škále frekvencí, výjimkou jsou lasery či sodíkové výbojky, které mají frekvenční spektrum poměrně omezené. Přijímané světlo je pak v lidském mozku interpretováno jako výsledná barva světla.

Dopadne-li bílé světlo na objekt, jsou některé frekvence povrchem objektu odraženy, jiné jsou pohlceny a některé mohou projít a být vyzářeny druhou stranou. Kombinace frekvencí přítomných v odraženém světle vytváří to, co vnímáme jako barvu objektu. Převládají-li například v odraženém světle nízké frekvence, je objekt vnímán jako červený.

V počítačové grafice se světelný paprsek reprezentuje tak, aby byly zachyceny jeho geometrické a optické vlastnosti. Geometricky je paprsek shodný s polopřímkou a k jeho určení stačí zadat jeho počátek (bod) a směr (vektor) (detailněji viz kap. 10.6). Optické vlastnosti se popisují s ohledem na jejich další zpracování počítačem. Z praktických důvodů se často omezujeme jen na trojsložkovou barevnou reprezentaci světla, nejčastěji v modelu RGB. Přesnější vyjádření pracuje s detailnější reprezentací světla, například s 256 reprezentanty vlnových délek. Výsledný obraz je pak uložen buď ve formátu RGB, nebo v některém z formátů umožňujícím zaznamenat vysoký dynamický rozsah (viz část 4.2).

10.1.1 Prostorové úhly

V algoritmech globálního osvětlování hrají zásadní úlohu prostorové úhly. Výpočet osvětlení bodu se vypočítá jako světlo přicházející skrze polokouli, která tento bod obklopuje. Z tohoto důvodu se používají sférické integrály a vyjádření směrů pomocí jednotkových vektorů na povrchu koule či diferenciálních plošek.

Jednotkový vektor $\vec{\omega}$ může být popsán jako bod na povrchu jednotkové koule, lze ho tedy vyjádřit pomocí dvou úhlů (θ, ϕ) (viz obrázek 10.1). Úhel $\theta \in [0, 2\pi]$ určuje výšku bodu a $\phi \in [0, \pi/2]$ je jeho azimut.

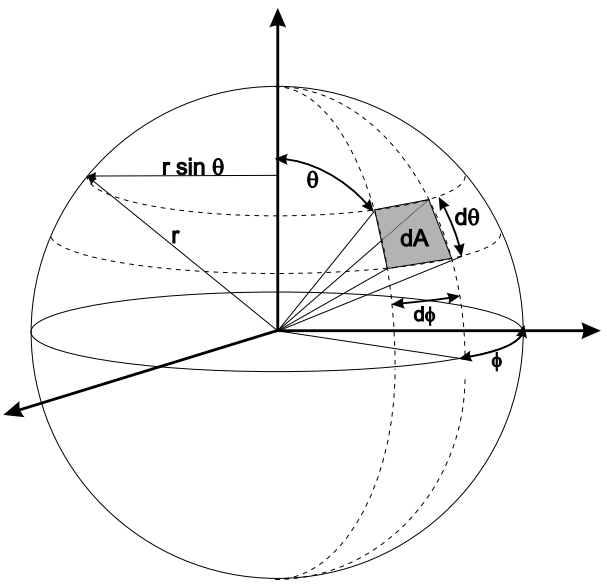
Uspořádaná trojice sférických souřadnic $[r, \theta, \phi]$ určuje jednoznačně polohu bodu na kouli o poloměru r

$$\begin{aligned} x &= r \cos\phi \ \sin\theta \\ y &= r \sin\phi \ \sin\theta \\ z &= r \cos\Theta. \end{aligned}$$

Rovnice pro převod kartézských souřadnic $[x, y, z]$ do sférických mají tvar:

$$r = \sqrt{x^2 + y^2 + z^2}$$

$$\tan\phi = \frac{x}{y}$$
$$\tan\theta = \frac{\sqrt{x^2+y^2}}{z}.$$



Obrázek 10.1: Sférická geometrie [Cohe93]

Prostorové úhly se nejčastěji orientují tak, aby směřovaly směrem od povrchu. Budeme tedy i dopadající světlo reprezentovat vektorem, který směřuje směrem od povrchu, tedy zdánlivě nelogicky, dopadající světlo je na povrchu objektu reprezentováno vektorem, směřujícím ke světelnému zdroji.

Směry mohou být jednotně popsány pomocí vektorů, ale také jako směry diferenciálních plošek na povrchu jednotkové koule. Uveďme nejprve obecný případ. Diferenciální plocha dA na povrchu koule o poloměru r vymezená dvěma úhly $d\theta$ a $d\phi$ je dána

$$dA = r^2 \sin\theta \; d\theta \; d\phi.$$

Velikost plošky se zmenšuje úměrně tomu, jak se přibližuje k pólu koule, a proto se v rovnici vyskytuje korekční člen $\sin\theta$. Úhel na kružnici je definován v radiánech, prostorový úhel ve steradiánech $[sr]$. Obvod kružnice o poloměru r je $2\pi r$ a plný úhel je 2π. Analogicky, celkový povrch koule je $4\pi r^2$, tedy úhel $4\pi[sr]$.

Diferenciální prostorový úhel $d\vec{\omega}$ je vymezen ploškou dA

$$d\vec{\omega} = \sin\theta \; d\theta \; d\phi.$$

Jeden prostorový úhel vymezuje na površích koulí o různém poloměru různě velké oblasti.

Obyčejně se volně zaměňuje diferenciální prostorový úhel, diferenciální ploška na jednotkové kouli a vektor, neboť všechny veličiny popisují směr. Směr $d\vec{\omega}$ je určen bodem na jednotkové kouli a velikost $d\vec{\omega}$ odpovídá velikosti prostorového úhlu v daném směru. Vektor $\vec{\omega} = (\Theta, \phi)$ určuje jednoznačně nějaký směr, stejně jako například vektor $\vec{n} = (n_x, n_y, n_z)$.

10.1.2 Základní radiometrické pojmy

Základní světelnou částicí je foton. Je to nejmenší kvantum záření, které může být vyzářeno. Energie fotonu na vlnové délce λ $[nm]$ je

$$e_\lambda = \frac{h.c}{\lambda},$$

kde $h \approx 6.63 \cdot 10^{-34} Js$ je Planckova konstanta, $c = 299\,792\,458\, ms^{-1}$ je rychlost světla ve vakuu, a jeho jednotkou je $[e_\lambda]$ =Joule.

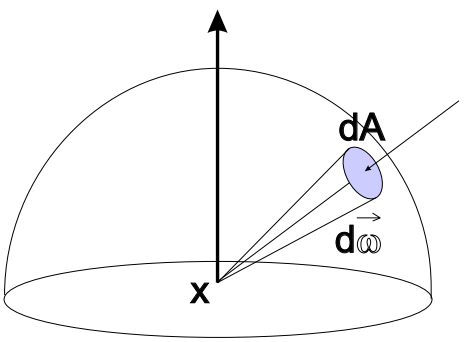
Radiantní energie Q $[J]$ je energie fotonů určité vlnové délky v určité oblasti. Tato veličina je nesměrová, jedná se pouze o součet energií všech fotonů v určitém místě. Předpokládejme n_λ fotonů vlnové délky λ s energií e_λ. Radiantní energie Q se určí jako integrál

$$Q = \int_0^\infty n_\lambda e_\lambda d_\lambda. \tag{10.1}$$

Veškeré níže uvedené veličiny se mohou uvádět pro konkrétní vlnovou délku nebo se mohou udávat na vlnové délce nezávislé. Ve druhém případě se taková veličina získá jako integrál na vlnové délce závislých příspěvků přes celé viditelné spektrum. Pro snažší vysvětlení budeme dále používat veličiny nezávislé na vlnové délce.

Světlo dopadající na povrch objektu či vyzářené do prostoru v určitém čase lze reprezentovat jako *zářivý výkon* (*radiant power*), též zvaný *zářivý tok* (*flux*). Označuje se Φ a udává se ve wattech $[W = Js^{-1}]$. Jedná se o množství energie vyzářené či přijaté za jednotku času. Světelný tok se vypočítá jako změna radiantní energie za jednotku času

$$\Phi = \frac{dQ}{dt}. \tag{10.2}$$



Obrázek 10.2: Prostorový úhel

V případě zářivého výkonu Φ se nerozlišuje, zda se jedná o výkon vyzářený či dopadající. V dalším textu ho budeme proto rozlišovat dolním indexem Φ_i pro dopadající a Φ_r pro odražený světelný tok. Indexy i a r se vztahují k významu – *incident*, *reflected*.

Světelný tok dopadající na jednotku plochy, též hustota světelného výkonu na ploše, se nazývá *iradiance* (*irradiance*), označuje se E, má jednotky $[Wm^{-2}]$ a

$$E = \frac{d\Phi_i}{dA}, \tag{10.3}$$

kde A je ozářená plocha v $[m^{-2}]$. Světelný tok vyzářený plochou, lhostejno, zda se jedná o vlastní emisi (svícení) či odražený tok, se nazývá *radiozita* (*radiosity*, *radiant exitance*). Označuje se M či B, má stejné jednotky jako irradiance $[Wm^{-2}]$ a vypočítá se

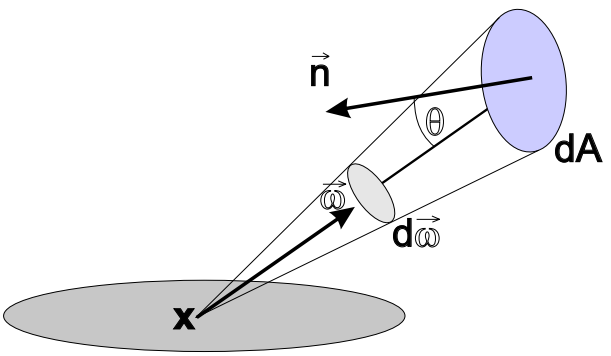
$$M = B = \frac{d\Phi_r}{dA}. \tag{10.4}$$

Zářivá intenzita (též zářivost) I $[Wsr^{-1}]$, též hustota výkonu v prostorovém úhlu, je dána jako světelný tok Φ proudící prostorovým úhlem $d\vec{\omega}$ (viz obrázek 10.2)

$$I = \frac{d\Phi}{d\vec{\omega}}. \tag{10.5}$$

10.1.3 Radiance

Nejdůležitější radiometrickou veličinou globálního osvětlování je *radiance*. Radiance udává přijímaný či vyzářený výkon na jednotkovém prostorovém úhlu na jednotku kolmo promítnuté plochy. Můžeme si ji představit jako veličinu udávající počet fotonů přicházejících či vyzářených v určitém směru za jednotku času a procházející průmětem diferenciální plošky dA, který je kolmý na tento směr. Radiance je to, co obyčejně označujeme barvu paprsku.



Obrázek 10.3: Radiance

Radiance (viz obrázek 10.3) závisí na poloze bodu x a na směru $\vec{\omega}$, označuje se $L(x, \vec{\omega})$, kde x je bod, ve kterém radianci vyšetřujeme, a $\vec{\omega}$ je daný směr. Jednotkou radiance je $[Wsr^{-1}\ m^{-2}]$ a určí se

$$L(x, \vec{\omega}) = \frac{d^2\Phi}{\cos\Theta dA d\vec{\omega}}.$$

Ve jmenovateli se vyskytuje člen $\cos\Theta$ neboť velikost promítnuté plochy závisí na kosinu úhlu, který svírá kolmice k promítané ploše a kolmice k ploše promítnuté (viz obrázek 10.4).

Důležitost radiance plyne i z faktu, že všechny výše uvedené radiometrické veličiny lze z ní vyjádřit. Radiantní energie (10.1) se vypočítá jako integrál radiance přes čas T, polokouli Ω nad bodem x a přes všechny body plochy $x \in A$

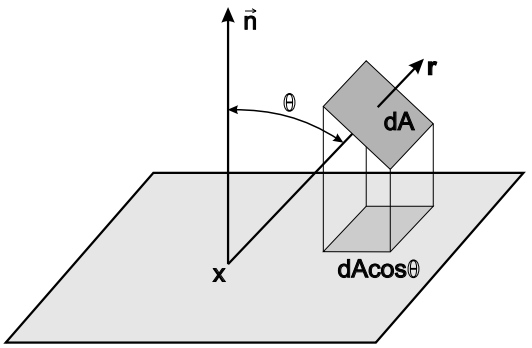
$$Q = \int_T \int_\Omega \int_{x \in A} L(x, d\vec{\omega}) \cos\Theta \; dA \; d\vec{\omega} dt.$$

Zářivý tok (10.2) je dán integrálem přes polokouli Ω a přes všechny body plochy A

$$\Phi = \int_\Omega \int_{x \in A} L(x, d\vec{\omega}) \cos\Theta \; dA \; d\vec{\omega}.$$

Iradiance (10.3), tedy přijímaná energie [analogicky by se získala vyzářená radiosita (10.4)], je integrálem vyzářené radiance přes polokouli

$$E = \int_\Omega L(x, d\vec{\omega}) \cos\Theta d\vec{\omega}.$$



Obrázek 10.4: Promítnutá plocha dA.

Intenzita (10.5) se získá integrací přes plochu A:

$$I = \int_{x \in A} L(x, d\vec{\omega}) \cos\Theta \; dA.$$

Ve vakuu je radiance na své dráze konstantní [Cohe93]. To je základní pravidlo, které je použito ve všech algoritmech globálního osvětlování s výjimkou tzv. opticky aktivních médií (*participating media*), tj. mlhy, kouře, částeček prachu, vodní páry, poloprůhledných materiálů atp. Tyto jevy jsou příčinou odrážení fotonů a tedy i změny jejich dráhy v prostoru.

Lidské oko, fotoaparáty, CCD aj. jsou citlivé na radianci. Jejich odezva je úměrná přijaté radianci, a proto se barva či kontrast objektu nemění se vzdáleností, ze které je pozorujeme či snímáme. Radiance je předmětem algoritmů globálního osvětlování a je to veličina, kterou obrazovka počítače interpretuje jako barvu.

10.2 Dvousměrová odrazová distribuční funkce – BRDF

Většina světla, které vnímáme, je světlo odražené od povrchů objektů. Světlo dopadající do lidského oka přímo ze světelného zdroje je spíše výjimkou. Barva objektů je dána spektrální charakteristikou světla, které na ně dopadá, ale především vlastnostmi povrchu, zejména tím,

jaké vlnové délky a v jakém směru odráží. Je patrné, že potřebujeme formální aparát, který nám umožní popsat schopnost materiálu odrážet či absorbovat světlo. K tomu musíme nejprve definovat určité předpoklady:

1. Světlo, které se od povrchu odrazí, se odrazí okamžitě. To je v souladu s požadavkem nekonečné rychlosti šíření světla. Tím však nebudeme moci simulovat jevy, jako je *fosforescence*, tj. „nabití“ materiálu světlem a jeho opožděné vyzáření.
2. Foton o vlnové délce λ bude vyzářen na téže vlnové délce. Změna vlnové délky po odrazu fotonu od povrchu se jmenuje *fluorescence* a nebudeme se jí rovněž zabývat.
3. Poslední předpoklad je složitější. Světlo, které přichází k povrchu materiálu z nějakého vstupního směru $\vec{\omega}_i$, se střetne s konkrétním bodem x. Budeme předpokládat, že světlo bude z téhož bodu odražené ve směru $\vec{\omega}_r$. Odraz z jiného bodu $x' \neq x$ znamená, že foton musí cestovat někde pod povrchem materiálu. Takový odraz je popsán pomocí dvousměrové rozptylovací odrazové distribuční funkceBSSRDF (*bidirectional scattering surface reflectance distribution function*) [Dutr03] a je zobecněním, které rovněž nebudeme uvažovat.

Dvousměrová odrazová distribuční funkce BRDF (*Bidirectional Reflectance Distribution Function*) charakterizuje odrazové schopnosti povrchu materiálu v určitém bodě. Označme tento bod x (viz obrázek 10.5). Světlo dopadá ze směru $\vec{\omega}_i$ a odráží se ve směru $\vec{\omega}_r$. BRDF se označuje $f_r(x, \vec{\omega}_r, \vec{\omega}_i)$ a definuje poměr odražené radiance v daném bodě x označené jako $dL_r(x, \vec{\omega}_r)$ ke vstupní diferenciální radianci $dL_i(x, \vec{\omega}_i)$ promítnuté na kolmou plochu

$$f_r(x, \vec{\omega}_r, \vec{\omega}_i) = \frac{dL_r(x, \vec{\omega}_r)}{dL_i(x, \vec{\omega}_i)(\vec{\omega}_i \cdot \vec{n})d\vec{\omega}_i}. \tag{10.6}$$

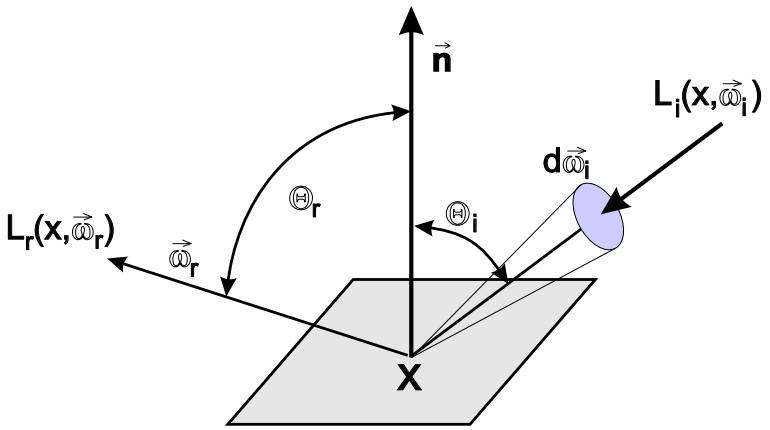
Pokud radiance dopadá kolmo na plochu, je přijatý světelný výkon největší.

10.2.1 Vlastnosti BRDF

Helmholtzův princip reciprocity říká, že hodnota BRDF v daném bodě zůstává stejná, i když zaměníme směr dopadu a odrazu světelného paprsku

$$f_r(x, \vec{\omega}_r, \vec{\omega}_i) = f_r(x, \vec{\omega}_i, \vec{\omega}_r).$$

Foton dopadající ze vstupního směru $\vec{\omega}_i$ je podroben na povrchu materiálu násobným odrazům, které vyústí v posledním odrazu ve výstupním směru $\vec{\omega}_r$. Pokud vypustíme foton z opačného směru musí projít pozpátku přesně stejnou cestou a být vyzářen z povrchu pod stejným úhlem. Tento princip je kriticky důležitý pro dvousměrové globální osvětlovací techniky, popsané dále



Obrázek 10.5: Dvousměrová odrazová distribuční funkce (BRDF)

v textu, neboť umožňuje vyšetřovat dráhu světla jak po drahách přicházejících, tak i po drahách odcházejících paprsků.

Pozitivita BRDF znamená, že tato funkce není nikdy záporná $f_r(x, \vec{\omega}_r, \vec{\omega}_i) \geq 0$.

Anizotropie je obecná vlastnost materiálu a BRDF je tedy obecně anizotropní. Anizotropie znamená, že odraz světla nezáleží jen na vstupním a výstupním směru a bodě x, ale také na natočení povrchu kolem normálového vektoru k tomuto povrchu. Příkladem anizotropního materiálu je kompaktní disk. Pokud ho z nějakého směru osvětlíme a budeme jeho povrch pozorovat z nějakého pevného bodu, při otočení disku kolem normálového vektoru se bude měnit barva odraženého světla. BRDF by tedy měla mít tvar $f_r(x, \vec{\omega}_r, \phi, \vec{\omega}_i)$, kde ϕ určuje úhel natočení bodu (odpovídá úhlu ϕ_i na obrázku 10.8). V počítačové grafice ve většině případů o anizotropii neuvažujeme, a proto pracujeme s BRDF ve tvaru (10.6).

Zákon zachování energie říká, že energie se nemůže vytvořit ani zaniknout, pouze se může měnit z jednoho druhu na jiný. V případě BRDF je vyjádřen vztahem:

$$\int_\Omega f_r(x, \vec{\omega}_r, \vec{\omega}_i) \cos \Theta_i d\vec{\omega}_i < 1, \forall \vec{\omega}_i.$$

Plocha nemůže odrazit více, nežli je celková přijatá energie. Zde je nutné připomenout, že BRDF vyjadřuje pouze odrazivost materiálu. To samozřejmě neznamená, že plocha sama není světelným zdrojem. Tento fakt však BRDF nereflektuje a je popsán dále v zobrazovací rovnici v části 15.1.

Linearita vyjadřuje fakt, že hodnota BRDF pro daný vstupní úhel $\vec{\omega}_i$ *nezávisí* na hodnotách BRDF pro jiné vstupní úhly, což znamená, že paprsek ze směru $\vec{\omega}_i$ je vyzářen ve výstupním

směru bez ohledu na to, co přichází ze směrů jiných. To vede k důležitému pojmu, zvanému lokální osvětlovací model, který je vysvětlen dále v části 10.3.

Odrazivost – reflektance Jednou z praktických nevýhod BRDF je neomezenost jejího oboru hodnot seshora. Z tohoto důvodu se pro vyjádření vlastností materiálů někdy používá *odrazivost*, též zvaná reflektance (*reflectance*). Odrazivost $\rho(x)$ je definována jako poměr odraženého světelného toku $\Phi_r(x)$ v bodě x k dopadajícímu světelnému toku $\Phi_i(x)$ v témže bodě:

$$\rho(x) = \frac{d\Phi_r(x)}{d\Phi_i(x)}. \tag{10.7}$$

Obor hodnost odrazivosti je interval $\langle 0, 1\rangle$. Pokud je hodnota $\rho(x) = 1$, dochází k odrazu veškerého dopadajícího světla. V ostatních případech je část světla pohlcena a přeměněna na teplo, nebo pouze projde materiálem.

10.3 Lokální osvětlovací model

Slovo lokální označuje fakt, že se vypočítává osvětlení jediného bodu na povrchu objektu. Rovnice lokálního osvětlovacího modelu zahrnuje dvousměrovou odrazovou distribuční funkci a vyjadřuje odraženou radianci v daném směru $L_r(x, \vec{\omega}_r)$ pro *všechny* vstupní směry $\vec{\omega}_i$:

$$L_r(x, \vec{\omega}_r) = \int_\Omega f(x, \vec{\omega}_r, \vec{\omega}_i) L_i(x, \vec{\omega}_i) \cos\Theta \; d\vec{\omega}_i. \tag{10.8}$$

Význam složitě vypadajícího integrálu je prostý. Abychom získali *celkovou* radianci, která opouští povrch ve směru $\vec{\omega}_r$, musíme vyčíslit integrál všech radiancí dopadajících na povrch, vynásobený dvousměrovou odrazovou distribuční funkcí. Barva bodu na povrchu tělesa se tedy získá jako odražená radiance v daném bodě. Lokální osvětlovací model je základem všech osvětlovacích algoritmů.

10.4 Odraz světla

BRDF může být reprezentována mnoha způsoby. Nejběžnější je její vyjádření pomocí jednoduché, často empirické, funkce. Složitější případy ji reprezentují pomocí čtyřrozměrné tabulky; pro dvojici vstupních a dvojici výstupních hodnot (dva směry vstupní a dva výstupní) máme vždy jednu hodnotu BRDF. V případě vlnově závislé BRDF musíme reprezentovat hodnot více. Hodnoty pro nedefinované směry se buď zkopírují z nejbližší známé hodnoty (interpolace nejbližším sousedem), nebo se interpolují z přilehlých hodnot. Tyto tabulky jsou přesné, avšak jsou obyčejně velmi rozsáhlé.

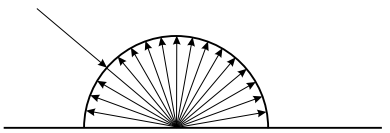
Nejčastěji jsou používány empirické modely BRDF a s tím je spojeno i jedno nebezpečí. Existují aproximace BRDF, které nesplňují některé z podmínek na BRDF kladených v odstavci 10.2. To může vést k nerealistickým výsledkům. Například dvousměrové metody globálního osvětlování jsou citlivé na splnění Helmholtzova principu reciprocity. Porušení zákona zachování energie může vést k tomu, že některé plochy budou odrážet více energie než pohlcují a tak dojde k nepřirozenému přesvětlování některých částí scény, atp.

10.4.1 Difúzní odraz

Difúzní odraz je speciálním případem obecného odrazu světla od povrchu materiálu.Povrch způsobující ideální difúzní odraz se také nazývá Lambertovský povrch (*Lambertian surface*). Dokonale difúzní povrch je matematická abstrakce.

Difúzní odraz rozptyluje vstupní radianci rovnoměrně do všech směrů (viz obrázek 10.6)

$$L_r(x, \vec{\omega}_r) = \int_\Omega f_d(x, \vec{\omega}_r, \vec{\omega}_i) L_i(x, \vec{\omega}_i) \cos\Theta \; d\vec{\omega}_i,$$



Obrázek 10.6: Difúzní odraz světla

kde $f_d(x, \vec{\omega}_r, \vec{\omega}_i)$ je BRDF difúzního materiálu. Protože $f_d(x) = konst$ pro všechny úhly, je odražená radiance rovna iradianci v daném bodě vynásobené hodnotou BRDF:

$$L_r(x, \vec{\omega}_r) = f_d(x) \cdot E_i(x).$$

Z toho plynou vlastnosti difúzního odrazu. Hodnota odražené radiance je úměrná vstupní radianci a nezávisí na výstupním úhlu. Budeme-li pozorovat difúzní povrch z jakéhokoli úhlu, bude vypadat stejně. Difúzně odražené světlo přináší informace o tom, co obyčejně nazýváme barvou povrchu.

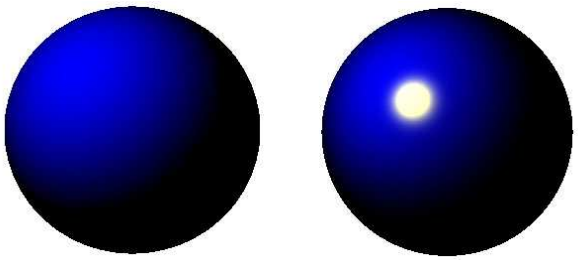
Odrazivost (reflektance) $\rho_d(x)$ difúzního povrchu v bodě x je konstantní a také nezávisí na výstupním úhlu:

$$\rho_d(x) = \frac{\Phi_r(x)}{\Phi_i(x)} = \pi \cdot f_d(x). \tag{10.9}$$

Vyjádření vlastností materiálu pomocí BRDF je v tomto případě obtížné, neboť BRDF může mít libovolnou kladnou hodnotu. Aby byla zachována její fyzikální korektnost, je snažší parametrizovat BRDF pomocí reflektance (10.9), neboť reflektance je omezena na interval $\langle 0, 1 \rangle$. Z toho plyne že BRDF lze vyjádřit z reflektance jako

$$f_d(x) = \frac{\rho_d(x)}{\pi}.$$

Z existujících materiálů se dokonale difúznímu odrazu nejvíce podobá křída, jak je vidět na obrázku 10.7 vlevo.

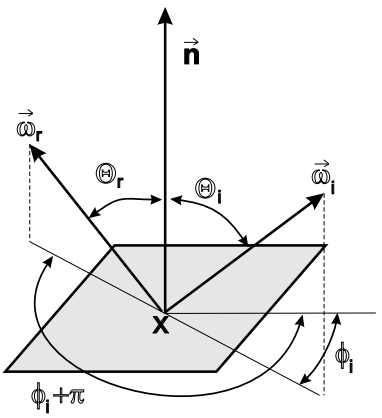


Obrázek 10.7: Difúzní a lesklý povrch.

V mnoha úlohách, například ve stochastickém renderingu či v radiozitě, je zapotřebí použít vztah pro rovnoměrné rozprostření paprsků na polokouli tak, jak odpovídá difúznímu odrazu [Jens01]. Předpokládejme, že máme dvě nezávislé náhodné proměnné $a, b \in \langle 0, 1\rangle$ s rovnoměrnou hustotou rozložení náhodných čísel. Z nich pak získáme požadovaný směr $\vec{\omega}$ jako $\vec{\omega} = (\Theta, \phi) = = (\arccos(\sqrt{a}), 2\pi b)$.

10.4.2 Zrcadlový odraz

Dalším důležitým druhem materiálu je zrcadlo, které odráží dopadající radianci pod zrcadlově souměrným úhlem ke směru dopadu. V praxi se této matematické abstrakci nejvíce podobají bezvadně vyleštěné povrchy kovů, voda a dielektrika, jako je například sklo. Zrcadlovému odrazu se také říká odraz *spekulární* (*specular reflection*).



Obrázek 10.8: Geometrie zrcadlového odrazu světla

Pro paprsek dopadající ze směru $\vec{\omega}_i = (\Theta_i, \phi_i)$ získáme odražený paprsek $\vec{\omega}_r$ (viz obrázek 10.8)

$$\vec{\omega}_r = (\Theta_i, \phi_i \pm \pi).$$

Výpočetně praktičtější je použít vztahy založené na skalárním součinu:

$$\vec{\omega}_r = 2(\vec{\omega}_i \cdot \vec{n})\vec{n} - \vec{\omega}_i.$$

Odražená radiance od zrcadlového povrchu je:

$$L_r(x, \vec{\omega}_r) = f_s(x, \vec{\omega}_r, \vec{\omega}_i) L_i(x, \vec{\omega}_i),$$

kde $f_s(x, \vec{\omega}_r, \vec{\omega}_i)$ je BRDF zrcadlového odrazu:

$$f_s(x, \vec{\omega}_i, \vec{\omega}_r) = \frac{1}{\cos\Theta_i} \delta(cos\Theta_i - \cos\Theta_r)\delta[\phi_i - (\phi_r \pm \pi)].$$

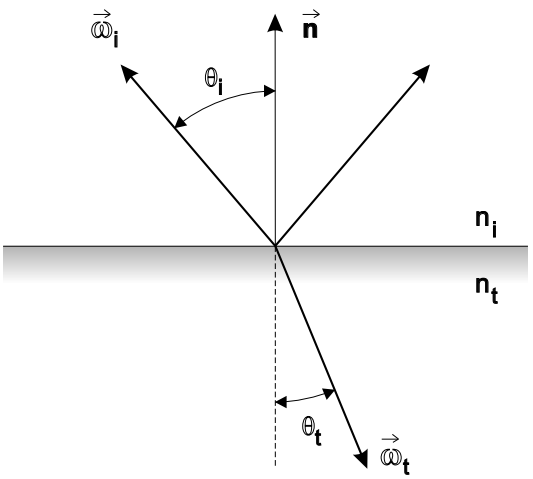
V tomto vztahu je δ tzv. Diracova funkce, se kterou jsme se již setkali při definici vzorkování na obrázku 2.6 na straně 47. Absolutně dokonalé zrcadlo bude odrážet jen v jednom jediném směru.

Zrcadlová složka je příčinou odlesků na zobrazovaných tělesech. Tyto odlesky mohou mít samozřejmě jinou barvu, než je barva povrchu tělesa. Příkladem je lesklý povrch černých brýlí odrážející všechny barvy.

Fresnelovy rovnice [Jens01] udávají množství odraženého světla pro lesklé hladké kovy a pro dielektrika. V těchto vztazích se uplatňují konstanty, které jsou pro některé materiály dostupné v tabulkách, a proto je možné je zahrnout v implementacích.

Lom světla

K lomu světla, refrakci (*refraction*), dochází na rozhraní dvou prostředí s různou optickou hustotou, a tedy i s různou rychlosti šíření světla. Příkladem je průchod světla průhledným objektem, například sklem či velmi tenkým kovem.



Obrázek 10.9: Lom paprsku na rozhraní dvou prostředí

Paprsek dopadající na rozhraní dvou prostředí je rozdělen na dvě části, na odraženou a lomenou. Na obrázku 10.9 je dopadající paprsek označen $\vec{\omega}_i$ a paprsek lomený $\vec{\omega}_t$ (*transmitted*). Společný povrch má jediný normálový vektor $\vec{n}$. Úhel dopadu, resp. úhel lomu je označen Θ_i, resp. Θ_t. Paprsky leží spolu s normálou k povrchu a bodem dopadu x v jedné rovině. Úhel lomu je možno vypočítat z vlastností materiálu a ze *Snellova zákona lomu*:

$$\frac{\sin\Theta_i}{\sin\Theta_t} = \frac{\eta_t}{\eta_i}, \tag{10.10}$$

kde η_i, resp. η_t je absolutní index lomu prostředí, ve kterém se šíří dopadající, resp. lomený paprsek.

Index lomu udává poměr rychlosti světla ve vakuu a v daném prostředí. Jeho hodnoty se pohybují v rozmezí od téměř 1,0 pro plyny, přes přibližně 1,6 pro sklo, až po více než 2,0 pro diamant. Index lomu závisí na vlnové délce a způsobuje například spektrální rozklad bezbarvého světla při průchodu hranolem. Tento jev, nazývaný *disperze*, obyčejně v počítačové grafice zanedbáváme.

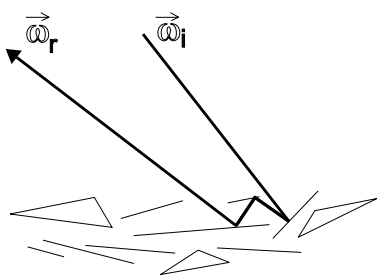
Šíří-li se paprsek z hustšího prostředí do řidšího ($\eta_i > \eta_t$), nabývá vztah (10.10) hodnoty menší než jedna a existuje úhel Θ_{iMax}, pro který již neexistuje úhel Θ_t takový, aby byla splněna rovnice (10.10). Tato situace odpovídá tzv. *totálnímu odrazu*, kdy žádné světlo již rozhraním neprojde a veškeré dopadající světlo se odrazí. K totálnímu odrazu dochází při kritickém úhlu dopadu, pro nějž platí $\sin\Theta_{iMax} = \eta_t/\eta_i$.

Praktickou implementační úlohou je ze zadaných indexů lomu η_i a η_t a směru dopadajícího paprsku $\vec{\omega}_i$ nalézt směr lomeného paprsku $\vec{\omega}_t$. Ten je určen vztahem [Jens01]:

$$\vec{\omega}_t = \frac{\eta_i}{\eta_r} - \left[\vec{\omega}_i - (\vec{\omega}_i \cdot \vec{n})\,\vec{n}\right] - \vec{n}\sqrt{1 - \left(\frac{\eta_i}{\eta_r}\right)^2 \left[1 - (\vec{\omega}_i \cdot \vec{n})^2\right]}.$$

10.4.3 Lesklý odraz

Lesklý, ne však dokonale zrcadlový odraz, se v anglické literatuře nazývá *glossy reflection* a je ukázán na obrázku 10.7 vpravo.



Obrázek 10.10: Odraz od povrchu složeného z mikroplošek

Lesklý odraz je výsledkem mnoha složitých jevů, které se navíc navzájem ovlivňují. Model takového odrazu předpokládá, že povrch je složen z mikroplošek (*microfacet*), které se vzájemně stíní a umožňují světlu proniknout do určité hloubky pod povrch materiálu (viz obrázek 10.10). Mnohonásobné podpovrchové stínění a rozptyl má vliv na směr odrazu světla. Analytická formule pro BRDF takového materiálu ve zcela obecné formě není známa, existují však její různé aproximace. Jedna z nejjednodušších je reprezentace BRDF ve formě:

$$f_r(x, \vec{\omega}_r, \vec{\omega}_i) = \frac{D\ G\ F}{4\cos\Theta_r \cos\Theta_i}.$$

Členy v čitateli zlomku mají následující význam [Cohe93]:

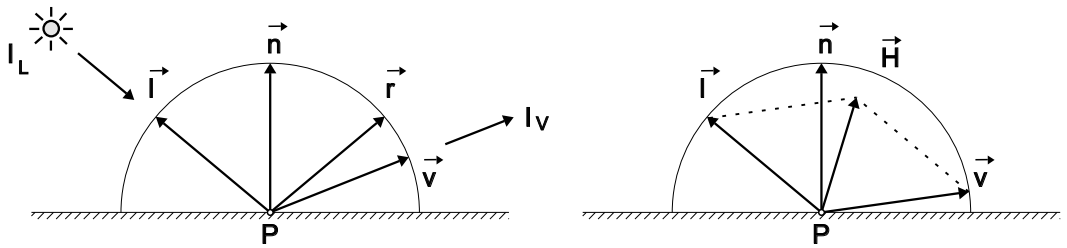
- D popisuje distribuci mikroplošek a v nejjednodušší podobě se používá člen (10.11) z Phongova empirického osvětlovacího modelu (viz část 10.5).
- G je geometrický člen, který simuluje vlastní stínění materiálu. Je to nejsložitěji modelovatelný člen z celé rovnice.

- F odpovídá Fresnelově koeficientu odrazivosti a je svázán s průhledností materiálu.

V dalším odstavci vysvětlíme nejčastěji používaný empirický osvětlovací model.

10.5 Phongův osvětlovací model

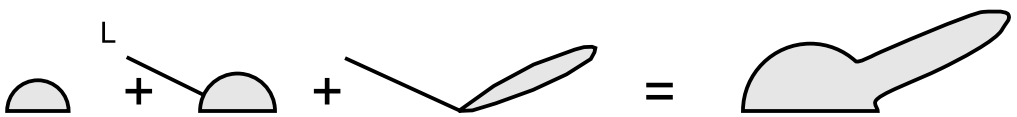
Empirický osvětlovací model pro výpočet odraženého světla navrhl v roce 1977 Bui–Tuong Phong[2] [Phon75]. Odraz na povrchu materiálu je určen směrem dopadajícího světla $\vec{l}$, směrem k pozorovateli bodem na povrchu P, normálovým vektorem v místě dopadu $\vec{n}$ a zrcadlově odraženým paprskem $\vec{r}$ tak, jak je uvedeno na obrázku 10.11.



Obrázek 10.11: (vlevo) Geometrie odrazu a zjednodušení výpočtu pomocí půlvektoru (vpravo)

Phongův osvětlovací model rozlišuje tři druhy odrazu světla od materiálu. Z těchto případů se pak skládá výsledný odraz. Odraz je rozdělený na zrcadlový (spekulární), difúzní a ambientní. Zrcadlový odraz není v pravém slova smyslu ideálním zrcadlovým odrazem, ale spíše odpovídá modelu lesklého povrchu z odstavce 10.4.3.

Na obr. 10.12 vidíme součet jednotlivých složek odrazu v Phongově osvětlovacím modelu. Oblasti znázorňují pravděpodobný rozptyl paprsků po odrazu, délky vektorů přibližně odpovídají intenzitě.



Obrázek 10.12: Odražené světlo v Phongově osvětlovacím modelu. Ambientní, difúzní a zrcadlová složka (vlevo) se sečtou do výsledného odrazu.

[2] Vyslov [fong].

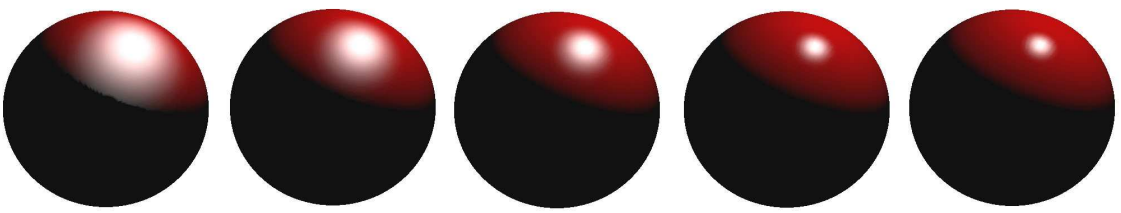
Zrcadlová složka je vyjádřena jako

$$\mathrm{I_s} = \mathrm{I_L}\ \mathrm{r_s}\ (\vec{v}.\vec{r})^h, \tag{10.11}$$

kde $\mathrm{I_L}$ reprezentuje barevné složení dopadajícího paprsku, $\vec{v}$ je jednotkový vektor pohledu (obr. 10.11). Vektor $\vec{r}$ vyjadřuje směr ideálního zrcadlového odrazu a je symetrický k vektoru $\vec{L}$ podle normály – lze jej vypočítat ze vztahu $\vec{r} = 2(\vec{l}.\vec{n})\vec{n} - \vec{l}$. Připomeňme, že skalární součin jednotkových vektorů je roven kosinu úhlu jimi svíraného.

Koeficient zrcadlového odrazu $0 \leq \mathrm{r_s} \leq 1$ určuje míru zastoupení odražené zrcadlové složky světla v celkově odraženém světle. Tento koeficient je trojsložkovým barevným vektorem. Násobení barevných vektorů se provádí po jednotlivých složkách na rozdíl od součinů geometrických vektorů.

Koeficient h je skalární a vyjadřuje ostrost zrcadlového odrazu. Udává se v rozmezí $\langle 1, \infty \rangle$. Pokud je $\vec{v}.\vec{r} < 0$, je pozorovatel vzhledem k zrcadlu ve stejné části prostoru jako zdroj světla, takže nemůže odraz vidět. Tehdy přiřadíme intenzitě $\mathrm{I_s}$ nulový barevný vektor vyjadřující fakt, že nedošlo k žádnému odrazu světla. Důležité je, že množství zrcadlově odraženého světla nabývá maxima tehdy, když směr odrazu je blízký směru pohledu. Toto maximum je tím výraznější, čím je parametr h větší. S rostoucím h se odlesky na zobrazovaném tělese stávají menšími a ostřejšími, dokonalé zrcadlo má $h = \infty$. V praxi vystačíme např. s hodnotou 100. Vliv koeficientu h na velikost odrazu demonstruje obrázek 10.13.



Obrázek 10.13: Vliv koeficientu h na velikost odlesku (zleva doprava: 10, 20, 40, 100 a 1 000)

Při výpočtu zrcadlové složky podle výše uvedeného vztahu lze použít zjednodušení, které navrhl Blinn a je k nalezení například v [Moll02]. Jeho podstatou je zamezení výpočtu odraženého paprsku a jeho nahrazení výpočtem skalárního součinu, což je podstatně jednodušší a urychlí tím výpočet. Namísto vztahu (10.11) se používá

$$\mathrm{I_s} = \mathrm{I_L}\ \mathrm{r_s}\ (\vec{h}.\vec{n})^{h_B},$$

kde $\vec{h}$ je tzv. *půlvektor*. Ten získáme v normalizované formě ze vztahu

$$\vec{h} = \frac{\vec{l} + \vec{v}}{\|\vec{l} + \vec{v}\|}.$$

Geometrie takto počítaného zrcadlového odrazu ukazuje obrázek 10.11 vpravo, kde je půlvektor zobrazen ještě v neznormalizované podobě. Skalární součin $\vec{n} \cdot \vec{h}$ vypočítává kosinus úhlu mezi normálovým vektorem a půlvektorem. Pokud směřuje odražený paprsek přímo k pozorovateli splyne půlvektor s normálovým vektorem a tento úhel je nulový.

Mocnitel h_B v upraveném vztahu neodpovídá mocniteli h ze vztahu původního. Platí přibližný vztah $(\vec{r} \cdot \vec{v})^h \approx (\vec{h} \cdot \vec{n})^{4h_b}$. Upravený model poskytuje odlišné výsledky, vizuální rozdíl je však nepostřehnutelný.

Difúzní složka odrazu se vypočítá podle vztahu

$$\mathrm{I_d} = \mathrm{I_L}\ \mathrm{r_d}\ (\vec{l}.\vec{n}), \tag{10.12}$$

kde $\mathrm{r_d}$ je koeficient difúzního odrazu, trojsložkový vektor. Udává zastoupení difúzní složky v celkově odraženém světle a do značné míry vyjadřuje to, co vnímáme jako barvu tělesa. Množství světla $\mathrm{I_d}$ je tím větší, čím je směr dopadu bližší normále. Což je Lambertův zákon difúzního odrazu $I_d = I \cdot \cos\alpha$. Vztah má smysl pouze pro $\vec{l}.\vec{n} > 0$, neboť v opačném případě je povrch odvrácen od světla a difúzní složka světla $\mathrm{I_d}$ je nulová.

Ambientní složka je odrazem blíže nespecifikovaného, ze všech směrů přicházejícího *okolního světla* (*ambient*) a značí se $\mathrm{I_a}$. Okolní rozptýlené světlo vzniklo mnohonásobnými odrazy od ostatních těles a rozptylem způsobeným molekulami vzduchu. Je proto většinou bílé. Odraz ambientního světla vyjádříme jednoduchým vztahem:

$$\mathrm{I_a} = \mathrm{I_A}\ \mathrm{r_a}, \tag{10.13}$$

kde $\mathrm{I_A}$ vyjadřuje množství okolního světla. Jde o veličinu, která bývá v empirických osvětlovacích modelech konstantní pro celou scénu, podílí se tedy stejným způsobem na osvětlení všech povrchových bodů. Barevný koeficient $\mathrm{r_a}$ vyjadřuje schopnost povrchu odrážet okolní světlo a bývá prakticky totožný s koeficientem $\mathrm{r_d}$ z rovnice (10.12). Zjednodušeně lze říci, že jeho velikost určuje, zda těleso je tmavé či světlé.

Přítomnost složky $\mathrm{I_a}$ zabraňuje tomu, aby povrchy odvrácené od všech světelných zdrojů byly zobrazovány jako zcela černé. Velikost $\mathrm{I_A}$ je nepřímo úměrná počtu světelných zdrojů ve scéně. Teoreticky by se naopak tato složka měla s počtem zdrojů zvyšovat, avšak to by vedlo k přesvětlení scény. Vzhledem k tomu, že s rostoucím počtem světelných zdrojů přibývá difúzních a zrcadlových složek odraženého světla, je celkové osvětlení scény uměle korigováno snížením ambientní složky. Tento logický rozpor je výsledkem empirického přístupu k osvětlení ve scéně.

Sečtením složky zrcadlové, difúzní a ambientní získáme vztah pro celkové světlo vnímané pozorovatelem na povrchu objektu:

$$\mathrm{I_V} = \mathrm{I_s} + \mathrm{I_d} + \mathrm{I_a}. \tag{10.14}$$

Je-li dán bod na povrchu, jehož osvětlení vyšetřujeme, přičteme pro něj složku $\mathrm{I_a}$ pouze jednou – na rozdíl od $\mathrm{I_s}$, $\mathrm{I_d}$, které se přičtou pro každý světelný zdroj L_k. Dopadají-li tedy do bodu P paprsky z M světelných zdrojů, bude platit

$$\mathrm{I_V} = \mathrm{I_A r_a} + \sum_{k=1}^{M} \mathrm{I_{L_k}} \left[\mathrm{r_s} (\vec{v}.\vec{r_k})^h + \mathrm{r_d} (\vec{l_k}.\vec{n}) \right]. \tag{10.15}$$

Tato rovnice se nazývá *Phongův osvětlovací model*[3]. Je třeba znovu zdůraznit, že jde o *empiricky* stanovený výraz, který nemá přímý vztah k fyzikální podstatě šíření a odrážení světla. Přesto je díky své jednoduchosti široce používán v počítačové grafice a stal se standardem v aplikacích reálného času. Prakticky všechny grafické procesory obsahují hardwarovou implementaci algoritmu modelu osvětlení. Další rozšíření Phongova osvětlovacího modelu uvedeme v kap. 15.9.1 v souvislosti s metodou sledování paprsku.

Modelování skutečných materiálů je nekončící zábavou mnoha počítačových grafiků. Existují desítky různých modelů jejichž základní dělení je možno provést na fyzikálně založené a na empirické. Z těch prvních jmenujme alespoň Straussův osvětlovací model [Stra90], Schlickův osvětlovací model, osvětlovací model Cook–Torrance, Torrance–Sparrow [Torr67]. Jejich popis nalezne čtenář v některé z knih o pokročilém globálním osvětlování [Cohe93, Dutr03, Jens01].

10.6 Světelné zdroje

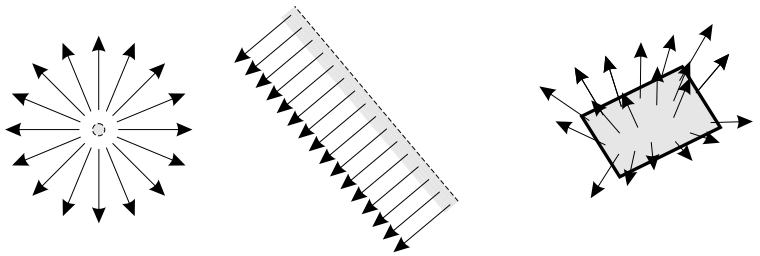
Světelný zdroj vyzařuje světelné záření. Světelný zdroj je obecně jakýkoli objekt, a proto může záření nejen emitovat, ale i odrážet. Vše, co bylo výše napsáno o odrazu světla od povrchu objektů, platí beze zbytku i pro světelné zdroje. Světelný zdroj je charakterizován emisním spektrem, funkcí, která udává světelný výkon pro každou vlnovou délku viditelného spektra. Tento údaj udává barvu světelného zdroje, směr vyzařování dané vlnové délky je specifikován příslušným vektorem. Nejjednodušším případem světelného zdroje je bodový světelný zdroj, který svítí konstantní barvou a intenzitou izotropně do všech stran. Nejsložitější světelné zdroje jsou zadány tabulkou a jejich popis je k nalezení v [Glas95].

Dále uvedeme nejčastěji používané světelné zdroje v počítačové grafice.

Bodový světelný zdroj (viz obrázek 10.14 vlevo) vyzařuje světlo rovnoměrně a se stejnou intenzitou do všech směrů. Bodový světelný zdroj je jednoznačně určen svou intenzitou a polohou.

Rovnoběžný světelný zdroj může být chápán jako bodový zdroj ležící v nekonečnu, nebo jako nekonečně velký rovinný zdroj v konečné vzdálenosti. Jeho paprsky dopadají rovnoběžně a obyčejně se jím aproximují velmi vzdálená světla, například sluneční svit.

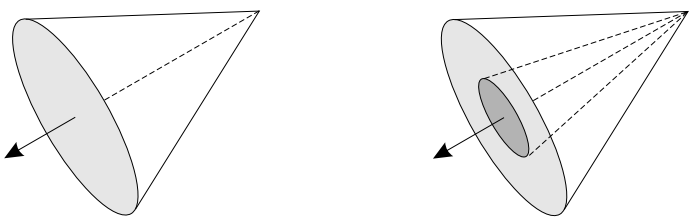
[3] Nepleťme si Phongův osvětlovací model s Phongovým stínováním, které bude uvedeno v kap. 10.7.3.



Obrázek 10.14: Zleva bodový zdroj světla, zdroj rovnoběžného světla a plošný zdroj

Plošný zdroj je obyčejně planární, má konečnou plochu a vyzařuje paprsky do předního poloprostoru všemi směry. Zatímco bodový zdroj nemůže nikdy způsobit stíny s neostrou hranicí (polostínem), plošné světelné zdroje tento jev způsobují.

Reflektor (*spot light*) je směrově závislý zdroj světla, který je určen polohou a směrem, kterým září (obr. 10.15 vlevo). Vyzařuje nejvíce ve směru osy vyzařování, kolmo k tomuto směru klesá intenzita exponenciálně. Geometricky je možno popsat reflektor jako kužel.

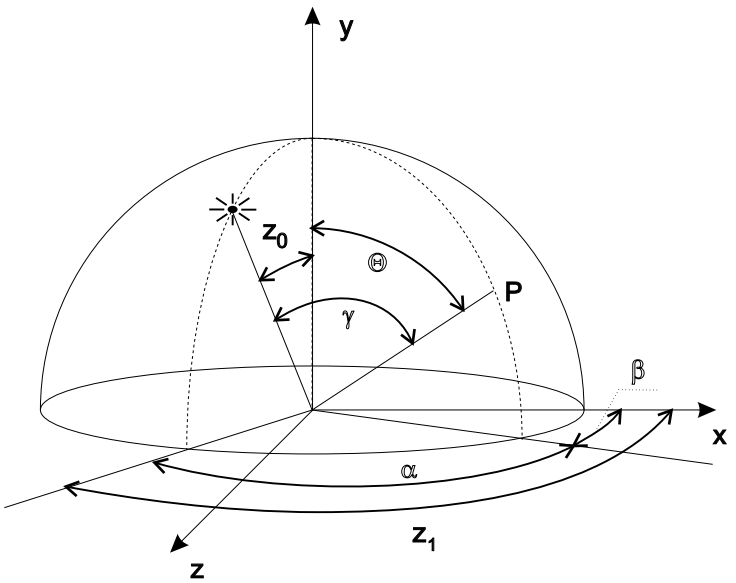


Obrázek 10.15: Jednoduchý reflektor (vlevo) a reflektor s jasným středovým svazkem (vpravo)

Reflektory mohou mít kolem své osy vyzařování jasnější oblast, kterou lze simulovat dvojicí souosých kuželů se shodným vrcholem (obr. 10.15 vpravo). Vnitřní kužel o menším poloměru představuje svazek jasného světla, jehož intenzita klesá pouze se vzdáleností od vrcholu, nikoliv se vzdáleností od osy kuželu. Vnější kužel určuje hranici, za kterou světlo z reflektoru vůbec nedopadá. V oblasti mezi vnitřním a vnějším kuželem je světlo tlumeno s ohledem na vzdálenost od osy.

Tabulka Dalším způsobem definice světla je jeho zadání tabulkou (*general luminaires*) nebo grafem. Takto lze definovat směrový bodový zdroj. Zápis pomocí tabulky či grafu určuje množství světla v závislosti na úhlu a vzdálenosti od světelného zdroje. Světelný zdroj je tedy zadán polohou a orientací a vyzařovací charakteristikou v tabulce.

Obloha je určena funkcí udávající radianci bodu P se souřadnicemi na obloze $[\Theta, \gamma]$ (viz obrázek 10.16). Obloha je popsána jako zdroj rovnoběžného světla ve tvaru polokoule s nekonečným poloměrem. Libovolný bod oblohy září tedy jako zdroj rovnoběžného světla.



Obrázek 10.16: Geometrie používaná při výpočtu zářivosti bodu na obloze

Standardizační komise CIE definovala dvě funkce pro výpočet zářivosti bodu na obloze, jednu pro oblohu zcela pokrytou mraky („zataženo“) a druhou pro čistou oblohu, na které je slunce („jasno“). Zářivost L bodu P na zcela zatažené obloze se vypočítá podle vztahu

$$L(\Theta) = L_z \frac{1 + 2\cos\Theta}{3}, \tag{10.16}$$

kde L_z je zářivost nejvyššího bodu oblohy (zenitu) a Θ je úhel mezi nadhlavníkem (zenitem) a bodem P. Je vidět, že intenzita je stejná na všech místech se stejnou úhlovou souřadnicí Θ, je tedy konstantní v kruzích rovnoběžných s obzorem.

Světlo z bodu P o souřadnicích $[\Theta, \gamma]$ na čisté obloze se sluncem se vypočítá podle vztahu

$$L(\Theta, \gamma) = L_z \frac{(0.91 + 10e^{-3\gamma} + 0.45\cos^2\gamma)(1 - e^{-0.32\sec\Theta})}{0.274(0.91 + 10e^{-3\gamma} + 0.45\cos^2 z_0)}. \tag{10.17}$$

V tomto vztahu je L_z zářivost zenitu, γ je úhel mezi sluncem a vyšetřovaným bodem P, Θ je úhel mezi zenitem a bodem P, z_0 je úhel mezi zenitem a sluncem a α je úhel γ promítnutý do roviny horizontu. Úhel γ může být vypočítán ze z_0, Θ, a α podle vztahu

$$\gamma = \arccos(\cos z_0 \cos\Theta + \sin z_0 \sin\Theta \sin\alpha). \tag{10.18}$$

10.7 Stínování

Vyhodnocování osvětlovacího modelu v každém bodě, který je vykreslován na obrazovce, je zdlouhavé, a proto bylo vyvinuto několik metod, které umožňují provést podrobný výpočet osvětlovacího modelu pouze pro několik bodů na povrchu tělesa a odvodit z nich barevné odstíny ostatních zobrazovaných bodů.

Tyto metody shrnujeme pod společný název *stínování* (*shading*). Pomocí stínování lze odlišit případné křivosti a zaoblení ploch, a docílit tak přirozeného vzhledu prostorových objektů přesto, že řada výpočtů týkajících se zpracování světla byla zjednodušena či vynechána. Některé druhy stínování dokonce umožňují opticky vyhladit povrchy, které jsou modelovány sítí rovinných plošek, takže přestanou být znatelné drobné hraniční zlomy.

Ačkoliv je stínování spjato se zpracováním světla, nezabývá se nalezením *vržených stínů* (*shadows*). Ty lze zpracovat speciálními metodami, popsanými dále v kap. 10.7, nebo globálními osvětlovacími technikami, které uvedeme v kap. 15.

10.7.1 Konstantní stínování

Tato metoda, známá též jako *flat shading*, je velmi rychlá, jednoduchá a používá se pro zobrazování rovinných ploch. Předpokládá, že každá plocha má jen jedinou normálu. Není-li normála implicitně obsažena v datech prostorového modelu, lze ji u konvexních rovinných plošek určit jako výsledek, který je správně zorientován tak, aby ukazoval do vnějšího poloprostoru. Podle normály je vypočítána jedna barva, která je při rasterizaci plochy přiřazena všem jejím pixelům. Konstantní stínování je používáno tam, kde je třeba docílit vysoké rychlosti kresby.

Pro kresby mnohostěnů je tento způsob stínování postačující a úspěšně znázorňuje umístění a natočení těles v prostoru. U obecnějších těles je konstantní odstín plošek negativním jevem, protože místo zkvalitnění obrázku zdůrazňuje, že oblý povrch je ve skutečnosti jen aproximován skupinou plošek (obr. 10.17 vlevo).

Pro obecná tělesa byly proto vyvíjeny metody spojitého barevného stínování. Jsou založeny na určování odstínu barvy bodu v ploše (obecně v prostorovém polygonu) ze znalosti hodnot ve vrcholech této plochy. Známe-li v každém vrcholu jeho barvu či normálu, dokážeme tyto hodnoty vypočítat bilineární interpolací (viz také kap. 22.6.4) pro každý vnitřní bod plochy. Při stínování pomocí bilineární interpolace vybarvujeme plochu po řádcích. Lineární interpolací mezi vrcholy určíme potřebné hodnoty na hranách, další lineární interpolací pak hodnoty uvnitř vybarvovaného řádku.

Obrázek 10.17: Zleva konstantní, Gouraudovo a Phongovo stínování téhož modelu definovaného pomocí rovinných plošek

10.7.2 Gouraudovo stínování

Metoda byla navržena H. Gouraudem[4] [Gour71] a je vhodná pro stínování těles, jejichž povrch je aproximován množinou rovinných plošek. Pro činnost algoritmu je důležitá znalost barev všech vrcholů zpracovávané plochy. Barvu vrcholu určíme vyhodnocením osvětlovacího modelu. Vzhledem k metodě výpočtu, který interpoluje barvy, nemá smysl zvažovat zrcadlovou složku světla. Počítá se tedy pouze ambientní a difúzní složka. Poté jsou vypočítány barevné odstíny vnitřních bodů dané plošky *bilineární interpolací* (viz kap. 22.6.4). Proto se Gouraudovo stínování také nazývá interpolací barvy.

Barvu v podobě trojsložkového vektoru (r, g, b) můžeme interpolovat po jednotlivých složkách. Každou složku v původním rozsahu $\langle 0, 1\rangle$ lze převést do celočíselného intervalu 0-255 a pro interpolaci použít celočíselnou aritmetiku. Celočíselná bilineární interpolace se snadno realizuje technickými prostředky. V současné době je proto Gouraudovo stínování standardní metodou používanou v grafických akcelerátorech.

Gouraudova metoda zajišťuje plynulé stínování křivých povrchů tak, že aproximace povrchů ploškami není zřetelná. Přesto ani tyto obrázky ještě neposkytují zcela věrný obraz reálných objektů – interpolace samotného odstínu barvy totiž nemůže způsobit místní zvýšení jasu na rovinné plošce, která nahrazuje zakřivenou plochu orientovanou kolmo na dopadající světelný paprsek. Tato metoda zahlazuje barevné rozdíly u místních nerovností povrchu.

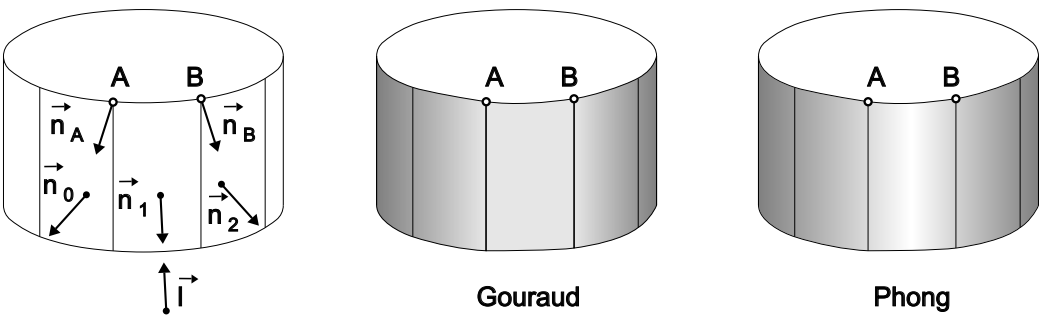
[4] Vyslov [guró, guródem]

10.7.3 Phongovo stínování

Také tato metoda je určena k plynulému stínování těles, jejichž povrch je tvořen množinou rovinných ploch. Jméno metody se spojeno s prací Bui-Tuong Phonga [Phon75]. Jeho jméno se tak objevuje v počítačové literatuře ve dvou souvislostech – jako jedna z variant empirického osvětlovacího modelu (viz předchozí část této kapitoly) a jako metoda stínování povrchů těles.

Metoda vychází ze znalosti normálových vektorů ve vrcholech stínované plochy. Z nich však nejsou pouze vypočítány barevné odstíny ve vrcholech, jako tomu bylo u Gouraudova stínování, nýbrž jsou použity k určení normálových vektorů ve vnitřních bodech plochy bilineární interpolací. Phongovo stínování je tedy založeno na *interpolaci normálových vektorů*. Normály jsou interpolovány současně při rasterizaci plochy a z nich je vyhodnocením, ve kterém bereme v úvahu všechny složky světla osvětlovacího modelu, určen odstín barvy každého pixelu.

Časové nároky algoritmu ve srovnání s Gouraudovým stínováním vzrostou, neboť osvětlovací model je vyhodnocován pro každý pixel.



Obrázek 10.18: Porovnání Gouraudova a Phongova stínování

Na obr. 10.18 je ukázán rozdíl mezi Gouraudovým a Phongovým stínováním. V situaci, kdy jsou světelné paprsky kolmé na určitou plošku, vzhled plošky se výrazně mění v závislosti na použité metodě stínování.

Na obr. 10.18 mají normály k povrchu v bodech A a B odlišné směry. Před normováním jsou

$$\begin{aligned} \vec{n}_A &= \vec{n}_0 + \vec{n}_1, \\ \vec{n}_B &= \vec{n}_1 + \vec{n}_2. \end{aligned}$$

U Gouraudova stínování jsou z těchto normál odvozeny tytéž barevné odstíny, protože úhly svírané normálami $\vec{n}_A, \vec{n}_B$ s vektorem $\vec{l}$ jsou totožné. Celá oblast mezi vrcholy A a B je pak vyplněna barvou téže intenzity a plocha vypadá jako rovinná.

U Phongova stínování dochází při interpolaci normál k jejich postupnému naklánění a tedy i ke správnému zvětšení velikosti jasu uprostřed oblasti mezi vrcholy A a B.

Gouraudovo stínování tedy vyniká rychlostí výpočtu, ale nedociluje tak kvalitního vzhledu obrázku jako stínování Phongovo. Zcela nevhodné je jeho použití v dynamických scénách, kdy dochází v průběhu natáčení plošek k viditelným jasovým změnám na povrchu. Pokud bychom například postupně otáčeli válec na obr. 10.18 kolem jeho osy, spatřili bychom na plášti výrazný světlý pruh vždy, když by se hrana pláště natočila kolmo na směr světla $\vec{l}$.

10.8 Opticky aktivní prostředí

Při šíření světla v prostoru jsme doposud předpokládali, že jeho trajektorie není ničím rušená. Prostor byl dokonale prázdný a foton vyzářený ze světelného zdroje, odražený od povrchu objektu, či prošlý průhledným objektem, putoval po přímce vstříc svému dalšímu osudu.

V této části se budeme zabývat případem, kdy šíření světla naopak něco v prostoru brání. Protože v prostoru je přítomné nějaké médium a aktivně se zúčastňuje transportu světla, říká se této úloze osvětlení s opticky aktivním prostředím (*participating media*).

Vstupem úlohy jsou objemová data, která můžeme reprezentovat například skalární funkcí $f(X)$ reprezentující hustotu v bodě $X[x, y, z]$. Tato funkce může být spojitá, či definována v diskrétních vzorcích, potom se aproximují hodnoty mezi vzorky interpolací (viz kap. 22.6). Při výpočtu osvětlení záleží na tom, zda je objem reprezentován jako voxely, či zda je v prostoru přítomen oblak částic (*particles*). Osvětlovací model, tj. předpis, jak stanovit barvu pixelu na obrazovce pro daný směr paprsku a danou cestu objemem, kombinuje:

- příspěvky světla, které prošlo celým objemem a bylo jím utlumeno (*absorption*);
- příspěvky vyzářené (*emission*) ve směru k pozorovateli každým mikroobjemem;
- a příspěvky rozptýlené (*scattering*) zrcadlovým odrazem a difúzí do směru k pozorovateli každým mikroobjemem. Vyzářené a rozptýlené příspěvky světla se cestou k pozorovateli tlumí stejně jako světlo z pozadí průchodem ostatními mikroobjemy. Navíc je v úplném osvětlovacím modelu nutné vypočítat zastínění světelných paprsků (od světelných zdrojů) jinými částmi objemu, tj. počítat stíny (*shadows*), a přidat příspěvky vzájemných odrazů mezi mikroobjemy (*multiple scattering*).

V této části uvedeme aproximaci osvětlovacího modelu, která zahrnuje útlum světla při průchodu objemem a záření jednotlivých mikroobjemů způsobené vlastním vyzařováním či odrazem světla přicházejícího ze světelných zdrojů. Vztahy budeme uvádět pro jednu vlnovou délku. Rozšířením této aproximace o vržené stíny (*shadows*) a vícenásobné odrazy (*multiple scattering*) se zabývat nebudeme. Podrobné odvození uvedených vztahů lze nalézt např. v [Max95, Niel93].

Objemové algoritmy zobrazující „skalární objem“ jsou založeny na myšlence, kterou publikoval J. Blinn [Blin82b]. Blinn zobrazuje objekty složené z velkého množství částic a simuluje tak mraky a prstenec kolem planety Saturn. Částice jsou všechny stejné, ale tak malé, že nejsou viditelné jednotlivě. Jas oblasti se projevuje až integrací jejich příspěvků. Model předpokládá konstantní hustotu částic v celém objemu a definuje osvětlovací model, který závisí na vzdálenosti, kterou v prostředí paprsek urazil, a na úhlu, pod kterým dopadá světlo od světelného zdroje. Model umožňuje simulovat osvětlení mraku z libovolného směru a pro libovolnou polohu pozorovatele.

Kajiya a von Herzen [Kaji84] rozšířili v roce 1984 Blinnův model o nehomogenní prostředí a zobrazují mraky metodou sledování paprsku. Přestože ne všechna objemová data mají podobu plynů a mraků, vychází řada pozdějších algoritmů z Blinnových a Kajiyových myšlenek.

V této části se seznámíme s aproximací objemového osvětlovacího modelu bez vícenásobných odrazů a bez vržených stínů. Důležitým předpokladem je, že částice, z nichž je objem dané hustoty modelován, mají malou odrazivost, a proto jen rozptylují a tlumí světlo, které jimi prochází k pozorovateli. Protože nedochází k vzájemným odrazům a my sledujeme pro každý pixel pouze jeden (primární) paprsek procházející okem pozorovatele, nazývá se tento typ algoritmů obvykle *vrháním paprsku* (*ray casting*). Paprsek prochází objemem, je jím utlumován a ztrácí tak svou energii. V každém bodě objemu současně přebírá energii, která přichází ze světelných zdrojů do daného místa a je opticky aktivním prostředím rozptýlena do směru pozorovatele. Výslednou hodnotu jasu získáme integrací těchto příspěvků podél dráhy paprsku.

10.8.1 Odvození integrálu pro zobrazování objemů

Integrál (10.27) odvodíme postupně, nejprve pro absorpční člen, dále pro emisní člen a nakonec pro jejich kombinaci. Při odvozování předpokládáme orientaci paprsku ve směru ze scény k oku pozorovatele, poté uvedeme vztah i pro opačný směr paprsku (10.28).

Pohlcení procházejícího světla

Předpokládejme vrstvu objemu (např. kvádr) o tloušťce Δs s průřezem podstavy E. Její objem je tedy $E\Delta s$. V tomto objemu se nacházejí částice o stejném průměru, z nichž každá má průmět o ploše A. Vrstva je tak tenká, že je pravděpodobnost překrytí průmětu částic zanedbatelná.

Označíme-li hustotu částic v objemu písmenem ϱ, nachází se v něm $N = \varrho E\Delta s$ částic a jejich průměty pokryjí plochu zhruba $NA = \varrho AE\Delta s$. Touto zakrytou částí světlo neproniká. Relativní část světla, kterou vrstva nepropustí, je tedy rovna $NA/E = \varrho A\Delta s$, což v limitě $\Delta s \to 0$ (zároveň pravděpodobnost překrytí jdoucí k nule) vede na diferenciální rovnici

$$\frac{dI}{ds} = -\varrho(s)AI(s) = -\tau(s)I(s), \tag{10.19}$$

vyjadřující pokles intenzity světla po absolvování dráhy s v prostředí, kde $I(s)$ je intenzita světla ve vzdálenosti s a $\tau(s) = -\varrho A$ se označuje jako *koeficient úbytku* (*extinction coefficient*). Řešením rovnice (10.19) je

$$I(s) = I_0 e^{\left(-\int_0^s \tau(t)dt\right)}, \tag{10.20}$$

v níž I_0 je intenzita světla v místě vstupu paprsku do objemu (kde $s = 0$) a člen

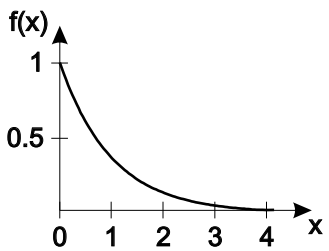
$$T(s) = e^{\left(-\int_0^s \tau(t)dt\right)} \tag{10.21}$$

bývá zvykem označovat jako *průhlednost média mezi* 0 *a* s, či jako *akumulovanou průhlednost* (*accumulated transparency*).

Písmenem α označujeme *neprůhlednost* (*opacity*) objemu o délce l, která je doplňkem průhlednosti $T(s)$ do jedničky

$$\alpha = 1 - T(l) = 1 - \exp\left(-\int_0^l \tau(t)dt\right). \tag{10.22}$$

Je vhodné upozornit na častou nepřesnost, kdy se v literatuře o objemovém zobrazování chybně pracuje s *neprůhledností* α jako by byla přímo rovna již uvedenému *koeficientu útlumu* τ. Chyba se projeví zejména při změně velikosti objemu a při procházení v jiném, než kolmém směru, to znamená pokud se délka cesty l' objemem liší od l (jako například u hranatých monolitických voxelů simulovaných jako objem částic hustoty ϱ).



Obrázek 10.19: Průběh funkce $1/e^x$

Příčina je jasně patrná z průběhu funkce $1/e^x$ na obrázku 10.19. Je-li hodnota útlumu τ ve voxelu konstantní, přechází rovnice (10.22) do tvaru

$$\alpha = 1 - e^{-\tau l} = \tau l - \frac{(\tau l)^2}{2} + \ldots \tag{10.23}$$

a α lze pro malé voxely aproximovat jako $\min(1, \tau l)$.

Funkce, která přiřazuje hodnotám skalární funkce $f(X)$ hodnotu τ, se označuje jako *přenosová funkce* (*transfer function*). Výpočetně nejjednodušší přenosovou funkcí je *prahování* (*thresholding*), kdy nastavíme $\tau = \infty$ (tj. $\alpha = 1$) pro $f(X) >$*práh* a $\tau = 0$ (tj. $\alpha = 0$) pro ostatní hodnoty.

Složitější optické modely zavádějí koeficient utlumení jako *spojitou funkci* $\tau(f(X))$ skalární proměnné $f(X)$. Výsledkem je obraz podobný rentgenovému snímku. Speciálně pro pole skalárních hodnot získaných rentgenovou počítačovou tomografií lze při identitě $\tau = f$ generovat simulované rentgenové snímky z libovolného směru pohledu. Obdobně lze kombinovat prahování a identitu a promítnout jen některé hodnoty ze zadaného intervalu.

Vlastní záření částic

Vzorec, popisující celkovou intenzitu při průchodu objemem se zářícími částicemi (či mikroobjemy), jímž můžeme popsat např. žhnoucí plyn, odvodíme následovně.

Předpokládejme již zmíněný plátek objemu tloušťky Δs, v němž jsou rozptýleny průhledné částice o hustotě ϱ, zářící do jednotkové plochy intenzitou C. Plochou jejich průmětu $\varrho AE\Delta s$ prochází tedy světelný tok $C\varrho AE\Delta s$ a jednotkou plochy proudí tok $C\varrho A\Delta s$. Pro I(s) platí diferenciální rovnice

$$\frac{dI}{ds} = C(s)\varrho(s)A = C(s)\tau(s) = g(s), \tag{10.24}$$

kde $g(s)$ označujeme jako *zdrojový člen* (*source term*), pomocí něhož lze modelovat jak vyzařování, tak odrazivost. Řešením rovnice je

$$I(s) = I_0 + \int_0^s g(t)dt. \tag{10.25}$$

Položíme-li g přímo úměrné f, lze opět generovat obdobu rentgenových snímků.

Pohlcování světla a vlastní záření částic

Realističtějších obrázků dosáhneme, když částice současně září a tlumí procházející světlo, přičemž není důležité, zda je původ záření ve vlastním vyzařování či v odrazech od zdrojů. Této situaci odpovídá diferenciální rovnice kombinující vztahy (10.19) a (10.24):

$$\frac{dI}{ds} = g(s) - \tau(s)I(s). \tag{10.26}$$

jejímž řešením je pro $I(s_1)$

$$I(s_1) = I_0 exp\left(-\int_{s_0}^{s_1} \tau(t)dt\right) + \int_{s_0}^{s_1} g(s)\exp\left(-\int_s^{s_1} \tau(t)dt\right) ds, \tag{10.27}$$

kde člen s_0 odpovídá hranici objemu, která je dál od pozorovatele, a s_1 hranici blíže k pozorovateli. Vzhledem k symetrii lze vztah pro obrácenou orientaci paprsku vyjádřit jako

$$I(s_0) = I_1 exp\left(-\int_{s_1}^{s_0} \tau(t)dt\right) + \int_{s_1}^{s_0} g(s)\exp\left(-\int_s^{s_0} \tau(t)dt\right) ds. \tag{10.28}$$

Uvedený vztah (10.27), (10.28) bývá označován jako *integrál pro zobrazování objemů* (*volume rendering integral*). Jeho první člen reprezentuje utlumené světlo z pozadí a druhý člen utlumené příspěvky jednotlivých mikroobjemů.

K výpočtu integrálu se obvykle používá odhad součtem. Rozdělíme-li interval na podintervaly šířky Δx, lze pro dostatečně malá Δx nahradit integrál $\int_a^b h(x)dx$ součtem $\sum_{i=1}^{n} h(x_i)\Delta x$, kde interval $\langle a, b\rangle$ je rozdělen na podintervaly, např. o šířce $\Delta x = (b-a)/n$. Pro $a = 0$ je $x_i \in \langle (i-1)\Delta x,\, i\Delta x\rangle$, tedy lze zvolit např. $x_i = i\Delta x$.

Integrál odhadneme součtem

$$exp\left(-\int_{s_0}^{s_1} \tau(x)dx\right) \approx exp\left(-\sum_{i=1}^{n} \tau(i\Delta x)\Delta x\right) = \prod_{i=1}^{n} exp\left(-\tau(i\Delta x)\Delta x\right) = \prod_{i=1}^{n} t_i,$$

kde $t_i = exp\left(-\tau(i\Delta x)\Delta x\right)$ [dle (10.23)] lze chápat jako průhlednost i-tého úseku podél paprsku. Je zřejmé, že t_i závisí nejen na $\tau(X)$, ale též na délce úseku Δx.

Obdobně pro $g_i = g(i\Delta t)$ upravíme vztah (10.27) do tvaru

$$\begin{aligned} I(s_1) &= I_0 \prod_{i=1}^{n} t_i + \sum_{i=1}^{n} g_i \prod_{j=i+1}^{n} t_j \\ &= g_n + t_n\left(g_{n-1} + t_{n-1}(g_{n-2} + \cdots (g_1 + t_1 I_0)\cdots)\right). \end{aligned} \tag{10.29}$$

Uvedený výraz lze prakticky realizovat jako algoritmus *back-to-front*, který při kolmém promítání zpracovává voxely podél paprsku v pořadí odzadu dopředu a akumuluje jejich příspěvky k intenzitě I. Algoritmem 10.1 postupně vyhodnotíme intenzity všech pixelů obrazu.

1. Nastav počáteční hodnotu I na I_0
2. pro j od 1 do n
   (a) $I = t_j.I + g_j$

Algoritmus 10.1: Algoritmus *back-to-front* pro jeden paprsek

Objem lze procházet i v opačném směru, tj. od bližších objemových elementů ke vzdálenějším. Při tomto směru postupu můžeme iteraci v neprůhledných částech objemu obvykle zastavit po několika málo krocích – v okamžiku, kdy hodnota akumulované průhlednosti T klesla pod zadaný práh a příspěvky dalších elementů se již stejně neuplatní. Pro malá t_i se cyklus ukončí dříve a provádění je rychlejší. Algoritmus 10.2 se nazývá *front-to-back*.

Použitý model chování částic

Ve vzorci (10.24) jsme předpokládali objem obsahující shodné částice a zdrojový člen $g(s)$ jsme definovali jako $g(s) = C(s)\tau(s)$. Je-li barva C uvnitř celého objemu (nebo jeho části podél

1. Vynuluj akumulátor intenzity I
2. Nastav počáteční hodnotu akumulované průhlednosti T na 1.
3. Nastav řídicí proměnnou cyklu j na n
4. Dokud $T >$ malý práh a zároveň $j \geq 1$ opakuj
   (a) $I = t_j.I + g_j$
   (b) $T = T.t_j$
   (c) Sniž proměnnou j o jedničku
5. $I = I + T.I_0$

Algoritmus 10.2: Algoritmus *front-to-back* pro jeden paprsek

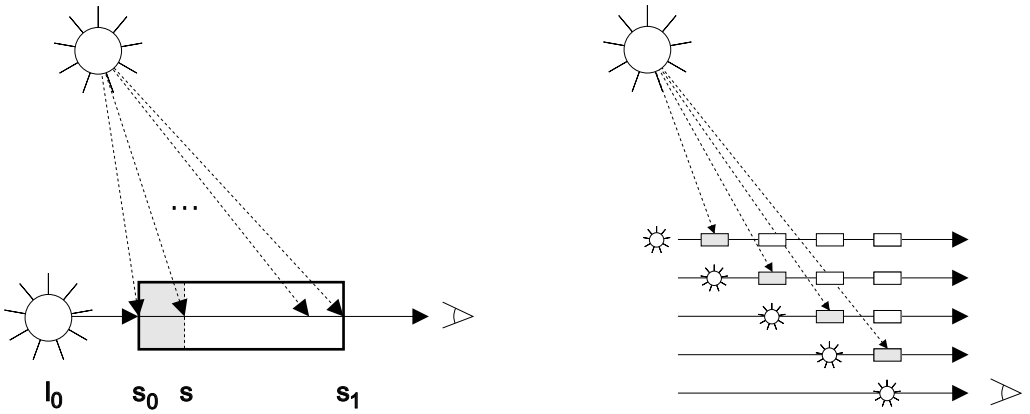
paprsku) konstantní (tj. objem je z jednoho materiálu), lze integrál zjednodušit na $I(s) = = I_0T(s) + C(1.0 - T(s))$. Neprůhlednost $\alpha = (1.0 - T(s))$ odpovídá pravděpodobnosti, že paprsek zasáhne částici a „uvidí“ barvu C. Příklady odlišných definic zdrojového členu $g(s)$ lze nalézt např. v [Max95].

Rozptylování světla přicházejícího ze světelných zdrojů

V předchozí části jsme uvedli, že obecně není důležité, co je zdrojem jasu $g(s)$ objemového elementu, a uvažovali jsme pouze vlastní vyzařování. Zde se soustředíme na případ, kdy je záření elementu objemu způsobeno též rozptylem světla, které přichází ze světelných zdrojů. I zde předpokládáme, že nedochází ke vzájemným odrazům mezi elementy a že světlo není na cestě od zdrojů k elementu nijak tlumeno, takže nevznikají stíny.

Situaci ilustruje obrázek 10.20. V levé části je naznačen průchod paprsku objemem od vstupního bodu s_0 po výstupní bod s_1 jako ve vzorci (10.27). Každý objem je zpracováván nezávisle na okolí, jako by byl přímo osvětlován světlem pozadí a světlem ze světelných zdrojů. V pravé části vidíme postup na úrovni mikroobjemů. Obrázek naznačuje, že každý objem je osvětlen ve směru paprsku (buď světlem z pozadí, nebo od předcházejících voxelů). Tato část světla je objemem částečně absorbována. Mikroobjem dále vyzařuje vlastní energii a rozptyluje či odráží světlo světelných zdrojů. Součet příspěvků předá jako osvětlení pozadí následujícímu mikroobjemu ve směru paprsku.

Postup lze chápat i jinak. Světlo z pozadí musí projít všemi mikroobjemy a každým je částečně utlumeno. Každý následující mikroobjem se chová jako světelný zdroj g v pozadí



Obrázek 10.20: Výpočet osvětlení při průchodu paprsku objemem (vlevo) se rozloží na součet příspěvků jednotlivých mikroobjemů utlumených cestou k pozorovateli

zbývajících mikroobjemů a jeho příspěvek je zbývajícími mikroobjemy utlumen. Takto se postupuje u všech mikroobjemů směrem k pozorovateli.

Intenzita světla, které je odraženo v bodě X od jednoho světelného zdroje ze směru označeném $\vec{\omega}'$ o intenzitě $I(X, \vec{\omega})$ do směru k pozorovateli $\vec{\omega}$, se obecně vyjádří jako

$$S(X, \vec{\omega}) = r(X, \vec{\omega}, \vec{\omega}')I(X, \vec{\omega}).$$

Pro světlo odražené shlukem částic o určité hustotě např. platí

$$r(X, \vec{\omega}, \vec{\omega}) = a(X)\tau(X)p(\vec{\omega}, \vec{\omega}'),$$

kde $a(X)$ je *odrazivost* (*albedo*) a $p(\vec{\omega}, \vec{\omega}')$ je *fázová funkce*, která je úměrná kosinu fázového úhlu mezi jednotkovými vektory $\vec{\omega}$ a $\vec{\omega}'$, tj. $x = cos\alpha = \vec{\omega}.\vec{\omega}'$. Příklady fázových funkcí jsou uvedeny např. v [Blin82b].

Kapitola 11

Řešení viditelnosti

Cílem algoritmů pro řešení viditelnosti je nalezení těch objektů a jejich částí, které jsou viditelné z určitého místa. V technických aplikacích se setkáváme s rozšířenou úlohou, při které je třeba umět vykreslit i zakryté části (*hidden parts*), např. zakryté hrany zobrazit čárkovanou čárou.

Algoritmy pro řešení viditelnosti jsou vždy svázány s konkrétní reprezentací prostorových objektů. Dobře se zpracovávají objekty popsané hraniční reprezentací, a to zejména ploškovou. Většina známých metod pro řešení viditelnosti pracuje právě s rovinnými ploškami. Pro jiné reprezentace je nutno použít specializované postupy. Často jsou také objekty z jiných reprezentací převáděny do ploškové hraniční reprezentace. Možná ztráta přesnosti při převedení dat je vyvážena rychlostí zobrazování. Algoritmy viditelnosti se dělí do dvou základních skupin podle toho, v jakém tvaru poskytují výsledná vyhodnocená data:

1. *Vektorové* algoritmy (též *liniové* algoritmy)
   Jejich výstupem je soubor geometrických prvků, např. úseček, představujících viditelné části zobrazovaných objektů. Úsečky, popsané koncovými body, lze plynule zvětšovat, takže jednou vyhodnocená data z hlediska viditelnosti lze opakovaně vykreslit v libovolném rozlišení na různých zařízeních. Dalším výstupem těchto algoritmů bývá soubor zakrytých částí objektů, se kterými se při případném vykreslování zachází obdobně. Tato třída algoritmů je používána zejména v technických aplikacích. Algoritmy, které vracejí pouze úsečky, se nazývají *HLE* (*Hidden Line Elimination*), pokud zpracovávají plochy, říkáme jim *HSE* (*Hidden Surface Elimination*).
2. *Rastrové* algoritmy
   Pracují pouze v rastru. Výsledkem je obraz, jehož jednotlivé pixely obsahují barvu odpovídajících viditelných ploch. Nevýhodou je pevný rozměr výsledného obrázku. Naprostá většina současných metod patří do této kategorie.

Zajímavé je posouzení vlivu možné numerické chyby při provádění algoritmů. Zatímco chyba při provádění rastrového algoritmu ovlivní většinou pouze několik pixelů, chyba vektorového algoritmu (např. označení jedné ze zakrytých hran za viditelnou) může způsobit výrazné porušení vzhledu celého obrazu.

Jiné dělení algoritmů bere do úvahy veličiny, které mají vliv na rychlost výpočtů. Těmito veličinami jsou jednak počet zpracovávaných objektů (n), resp. jejich ploch, jednak počet bodů výsledného rastrového obrazu (r). Při určování viditelnosti pak můžeme hledat řešení v různých prostorech:

1. Řešení v *prostoru objektů* (*object space*)
   Rychlost výpočtů odpovídá n^2. Základem je porovnávání vzájemných poloh těles, resp. jejich částí, jako jsou plochy a hrany. Algoritmy lze symbolicky vyjádřit takto:
   „Pro každý objekt zjisti, která jeho část je vidět.“
   Složitost $\mathcal{O}(n^2)$ je dána skutečností, že každý objekt musíme testovat vůči ostatním objektům. Pokud však prostor s objekty vhodným způsobem uspořádáme, např. pomocí hierarchických struktur popsaných v části 14.2, můžeme docílit složitosti $\mathcal{O}(n \log(n))$.

2. Řešení v *prostoru obrazu* (*image space*)
   Doba výpočtu je úměrná součinu $n.r$. Základem metod jsou testy určující pro každý pixel obrazovky, která část promítnutého objektu je v tomto pixelu nejblíže pozorovateli, a tudíž viditelná. Jinými slovy:
   „Pro každý pixel zjisti, který objekt je v něm vidět.“
   Pracuje se s promítnutými a poté rasterizovanými objekty. Tím je zajištěno, že testy viditelnosti se provádějí lokálně v určité oblasti rastru. Výsledkem je lineární výpočetní složitost $\mathcal{O}(n)$ pro danou velikost obrazu.

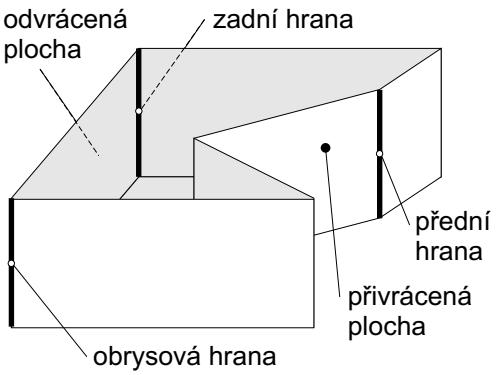
Některé algoritmy kombinují oba výše uvedené přístupy. Často jsou algoritmy urychlovány předzpracováním vstupních dat, zejména pro scény obsahující velké množství objektů. V této kapitole popíšeme pouze základní algoritmy řešení viditelnosti a pokročilým metodám pro zobrazování rozsáhlých scén v reálném čase věnujeme samostatnou kapitolu 19.

Dále budeme předpokládat, že na vstup algoritmů jsou dodávány objekty popsané hraniční reprezentací a že jsou již podrobeny pohledové transformaci tak, aby pozorovatel sledoval z kladné poloosy z jejich průmět do roviny xy. Osvětlovací model, použitý na výpočet barev viditelných částí objektů, budeme aplikovat *lokálně* na jeden objekt. Nebudeme tedy testovat vzájemné polohy objektů a zdrojů světla, na jejichž základě by bylo možno nalézt např. vržené stíny. Metody, zpracovávající objekty a světelné zdroje *globálně*, jsou složitější a budou jim věnovány samostatné kapitoly 12 a 15.

11.1 Vlastnosti zobrazovaných dat

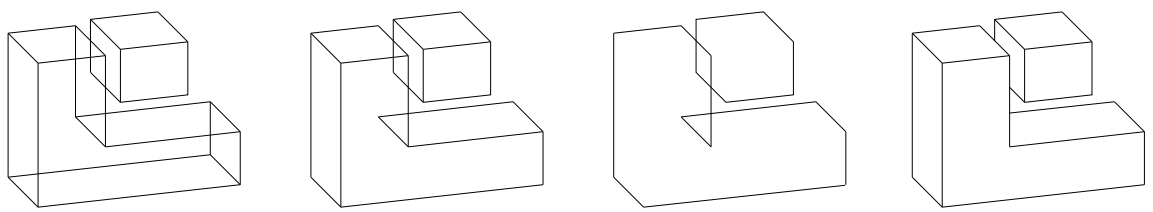
U těles popsaných ploškovou reprezentací je výhodné využít skutečnosti, že plošky jsou orientované – jejich normálový vektor míří vždy ven z tělesa. Podle toho, zda tento vektor směřuje k pozorovateli či od něj, dělíme plochy na *přivrácené* a *odvrácené*.

Úhel, který svírá normála odvrácené plochy se směrem pohledu, je vždy ostrý. Skalárním součinem uvedených dvou vektorů získáme kosinus tohoto úhlu. Je-li znaménko skalárního součinu kladné, úhel je ostrý a daná plocha je odvrácená. Pokud se pozorovatel dívá proti kladné poloose z, stačí namísto skalárního součinu ověřit znaménko složky z normálového vektoru plochy. Záporné znaménko určuje odvrácenou plochu. Předpokládáme, že plošky tvoří uzavřený plášť (povrch) tělesa a že tedy odvrácené plochy nejsou vidět, neboť jsou zakryty plochami přivrácenými. Můžeme je proto odstranit (*backface culling*), aniž bychom ovlivnili viditelnost dalších objektů.



Obrázek 11.1: Rozdělení hran na přední, zadní a obrysové (horní podstava tělesa je záměrně nevykreslena). Odvrácené plochy jsou tmavé.

Po rozdělení ploch na přivrácené a odvrácené je dále možno zavést *ohodnocení hran* (obr. 11.1). Hrana incidující se dvěma odvrácenými stěnami je *zadní* a tedy neviditelná, hrana sdílená přivrácenými stěnami je *přední*. Významnou hranou je ta, která je svírána přivrácenou a odvrácenou stěnou. Nazývá se *obrysová* (*contour edge, silhouette*) a v liniových algoritmech indikuje možnou změnu viditelnosti. Přední a obrysové hrany mohou být viditelné. Pokud zobrazujeme jediné konvexní těleso, jsou tyto jeho hrany viditelné všechny (viz malý kvádr na obr. 11.2). U nekonvexních těles a u skupin těles je třeba testovat vzájemné zákryty přivrácených stěn.



Obrázek 11.2: Při určování viditelnosti konvexního i nekonvexního tělesa je vhodné nejprve nalézt přivrácené stěny a obrysové hrany. Zleva doprava: objekty bez určení viditelnosti, vykreslení přivrácených stěn, vykreslení obrysových hran, objekty s vyřešenou viditelností.

11.2 Rastrové algoritmy viditelnosti

Na rozdíl od liniových algoritmů, které většinou v první fázi řeší samostatně úlohu viditelnosti a teprve ve druhé fázi rasterizují viditelné úsečky, u skupiny rastrově orientovaných algoritmů je rasterizace ploch prováděna současně s řešením viditelnosti.

11.2.1 Paměť hloubky

Tato metoda patří k nejznámějším a také nejefektivnějším. Klade sice určité nároky na paměť, ale současně dosahuje vysoké rychlosti zpracování. Je používána jako standardní prostředek pro řešení viditelnosti v naprosté většině grafických procesorů. Pouze některé specializované grafické procesory vypočítávají viditelnost odlišnými technikami, např. procesory pro zobrazování objemových (voxelových) dat.

Základem metody je použití *paměti hloubky* (*z-buffer, depth–buffer*), která tvoří dvourozměrné pole, jehož rozměry odpovídají velikosti obrazu. Každá položka paměti hloubky obsahuje souřadnici z toho bodu, který leží nejblíže k pozorovateli a jehož průmět leží v odpovídajícím pixelu v rastru.

Pokud se pozorovatel nachází před rovinou xy a objekty ve scéně leží za ní, je ze dvou bodů o stejných souřadnicích $[x, y]$ použit ten bod, jehož souřadnice z je větší. Zatímco souřadnice z bližšího bodu se ukládá do paměti hloubky, jeho barvu lze zapsat přímo do obrazové paměti (rastru). Metoda řešení viditelnosti pomocí paměti hloubky je popsána algoritmem 11.1.

1. Vyplň obrazovou paměť barvou pozadí
2. Vyplň paměť hloubky hodnotou $-\infty$
3. Pro každou plochu nalezni pixely, do nichž se plocha promítne. Pro každý takový pixel o souřadnicích $[x_i, y_i]$ stanov hloubku z_i
4. Má-li z_i větší hodnotu než položka $[x_i, y_i]$ v paměti hloubky, pak
   (a) obarvi pixel $[x_i, y_i]$ v obrazové paměti barvou této plochy
   (b) položku $[x_i, y_i]$ v paměti hloubky aktualizuj hodnotou z_i

Algoritmus 11.1: Řešení viditelnosti v paměti hloubky

Rasterizace plochy na pixely je prováděna obdobným způsobem jako při vyplňování rovinných ploch barvou. Pro vykreslení hranice ploch se používají pravidla uvedená v kap. 3.3

na str. 101. Oproti vyplňování v dvourozměrném prostoru jsou zpracovávány všechny tři souřadnice. Rozklad lze provádět v celočíselné aritmetice, s výjimkou hloubky z. Do paměti hloubky se ukládají hodnoty v pohyblivé řádkové čárce. I tato čísla jsou však v počítači uložena jen s určitou přesností a proto se může stát, že dvě plochy, jejichž hloubka se nepatrně liší, nebudou z hlediska viditelnosti vyhodnoceny správně. Tomuto jevu se říká *z-aliasing*.

Použití paměti hloubky je výhodné zejména proto, že každá plocha je zpracovávána pouze jednou. Doba zpracování tedy roste s počtem ploch lineárně, závisí ovšem i na jejich velikosti a rozměru obrazu. Metoda je typickým příkladem řešení úlohy viditelnosti v prostoru obrazu. Pokud rozdělíme obrazovou paměť a paměť hloubky na části (obvykle pruhy) můžeme plochy zpracovávat paralelně pomocí několika grafických procesorů.

Výhoda spočívá také v tom, že se jednotlivé prostorové plochy zpracovávají tak, jak přicházejí na vstup algoritmu, bez nutnosti ukládání do pomocných datových struktur a případného řazení. Při podrobnějším zkoumání chování algoritmu ovšem zjistíme, že pořadí zpracovávaných ploch má určitý vliv na rychlost algoritmu. Když na vstup algoritmu přicházejí plochy od nejvzdálenější k nejbližší, test v kroku 4 vrací vždy kladný výsledek, po němž následuje zápis do obrazové paměti i paměti hloubky pro všechny pixely každé plochy. Naopak při zpracování ploch odpředu dozadu je test v kroku 4 kladný pouze pro viditelné pixely ploch. Pro všechny ostatní (zakryté) pixely není nutno provádět kroky 4(a) a 4(b), čímž dojde k úspoře času.

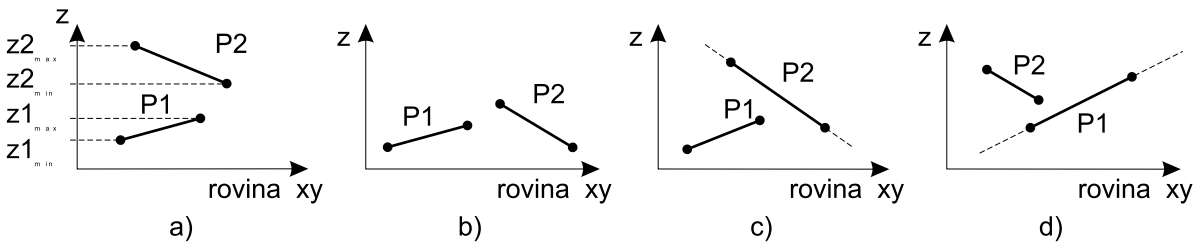
Z této úvahy vychází následující modifikace algoritmu. Předpokládáme, že počet zpracovávaných ploch je velký a že test v kroku 4 se tudíž provádí pro každý pixel mnohokrát (tento počet se nazývá hloubková složitost – *depth complexity*). Abychom se vyhnuli výpočetně náročnému řazení všech ploch odpředu dozadu, provedeme nejprve pro všechny plochy algoritmus 11.1, avšak s vynecháním kroku 4(a). Počet zápisů do paměti se tak sníží na polovinu a výsledkem je správně vyhodnocená paměť hloubky. Poté zpracujeme všechny plochy znovu, a to opět algoritmem 11.1, v němž tentokrát vynecháme krok 2, krok 4(b) a test v kroku 4 upravíme na rovnost. Výsledkem bude, že do každého pixelu obrazové paměti bude zapsána hodnota pouze jednou. Tato úprava má smysl jen v případě, že zápis do obrazové paměti je výpočetně „drahý“, tj. je spojen s náročnějšími výpočty barvy pixelu jako je mapování textury (viz část 13) nebo světelné a barevné efekty prováděné speciálními podprogramy (*shaders*).

11.2.2 Malířův algoritmus

Metoda je založena na myšlence vykreslování všech ploch postupně odzadu dopředu podobně, jako kdyby malíř nanášel na obraz nejprve barvy pozadí a přes ně kreslil objekty v popředí [Newe72]. Odtud plyne anglický název metody *painter's algorithm*, někdy též *priority list*. Plochy bližší k pozorovateli jsou vykreslovány později, takže překryjí zadní plochy. Řešení viditelnosti je tedy převedeno na úlohu seřazení ploch podle vzdálenosti od pozorovatele. Předpokládáme, že zpracováváme rovinné plochy, které se navzájem neprotínají (neprosekávají).

Vzhledem k tomu, že plochy jsou trojrozměrnými objekty, nelze je jednoznačně uspořádat podle jediné souřadnice. V malířově algoritmu proto nejprve nalezneme pro každou plochu její nejmenší souřadnici z_{min} a podle ní plochy seřadíme. Takto vzniklý seznam ještě nemůžeme vykreslovat, protože plochy mohou mít i takové polohy, při nichž plocha s menší souřadnicí z_{min} může zakrývat plochu s větší souřadnicí z_{min}.

První plocha v seznamu (nejvzdálenější od pozorovatele) je označena jako *aktivní* a je podrobována několika testům zakrývání vzhledem k ostatním plochám. Pokud lze rozhodnout, že aktivní plocha leží za všemi ostatními plochami, je vykreslena a vyřazena ze seznamu. V opačném případě dojde v seznamu k záměně aktivní plochy s tou plochou, u které dopadly všechny testy zakrývání negativně.



Obrázek 11.3: Polohy dvou ploch posuzované při testech zakrývání

Některé testy jsou založeny na porovnání dvou souřadnic, jiné vyžadují náročnější výpočty. V následujících textu budeme označovat aktivní plochu jako $P1$, její mezní souřadnice $z1_{min}$ a $z1_{max}$ (viz obr. 11.3a). Libovolnou další plochu ze seznamu označíme $P2$. Testy se provádějí postupně v dále uvedeném pořadí. Splnění libovolného z nich indikuje, že aktivní plocha $P1$ jednoznačně leží za plochou $P2$ a že je možno začít prověřovat polohu plochy $P1$ s další plochou. Obrázek 11.3 ukazuje možné polohy aktivní a testované plochy. Každá z poloh kreslených postupně zleva doprava představuje situaci, ve které předchozí testy neuspěly. Připomeňme, že stanoviště pozorovatele je umístěno na ose z a průmětnou je rovina xy.

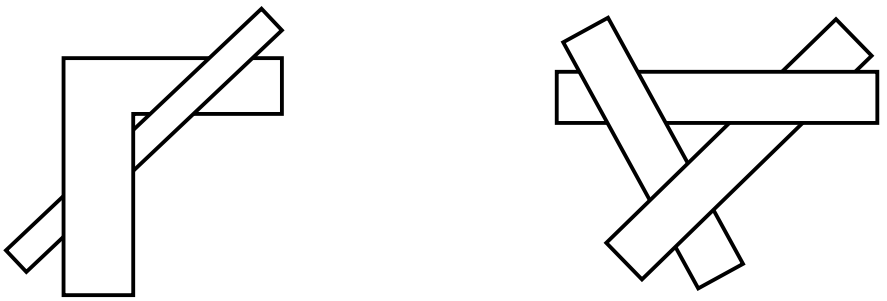
Čtyři nejjednodušší testy zakrývání u malířova algoritmu jsou následující.

1. Test je splněn, když $z1_{max} \leq z2_{min}$ (plocha $P1$ leží zcela za plochou $P2$, jako na obrázku 11.3a).
   Jakmile je tento test pro danou aktivní plochu $P1$ splněn, lze ji okamžitě vykreslit a vyřadit ze seznamu. Všechny další plochy leží blíže k pozorovateli, což je dáno iniciálním uspořádáním ploch v seznamu.
2. Test je splněn, pokud se nepřekrývají průměty ploch v rovině xy (obr. 11.3b).
   Máme-li u každé plochy zaznamenány mezní souřadnice x a y, lze překrývání průmětů ploch v mnoha případech rychle odhadnout jejich porovnáním.

3. Test je splněn, leží-li všechny vrcholy aktivní plochy $P1$ v poloprostoru, určeném testovanou plochou $P2$ a odvráceném od pozorovatele (obr. 11.3c).

4. Test je splněn, leží-li všechny vrcholy testované plochy $P2$ v poloprostoru, určeném aktivní plochou $P1$ a přivráceném k pozorovateli (obr. 11.3d).

Výše uvedené testy nejsou schopny odhalit veškeré možné případy vzájemné polohy dvou ploch. Pokud mají plochy nekonvexní tvary (obr. 11.4 vlevo), může dojít k jejich vzájemným zákrytům, které malířův algoritmus nedokáže správně zpracovat. Dílčím řešením je rozdělení všech ploch na konvexní části.



Obrázek 11.4: Nejednoznačné polohy ploch při malířově algoritmu

Ani existence pouze konvexních ploch nezaručuje korektní chování malířova algoritmu. Dokladem je obrázek 11.4 vpravo, na němž je znázorněna složitější (cyklická) poloha tří ploch. Tento stav můžeme odhalit při provádění algoritmu. Aktivní plocha přestane být aktivní, protože je zakryta jinou plochou, avšak po čase se opět aktivní stane, aniž by mezitím došlo k vykreslení jakékoliv plochy. Na situaci musíme reagovat buď rozdělením ploch (čímž klesá efektivita algoritmu) nebo vykreslením aktivní plochy s vědomím chybného výsledku.

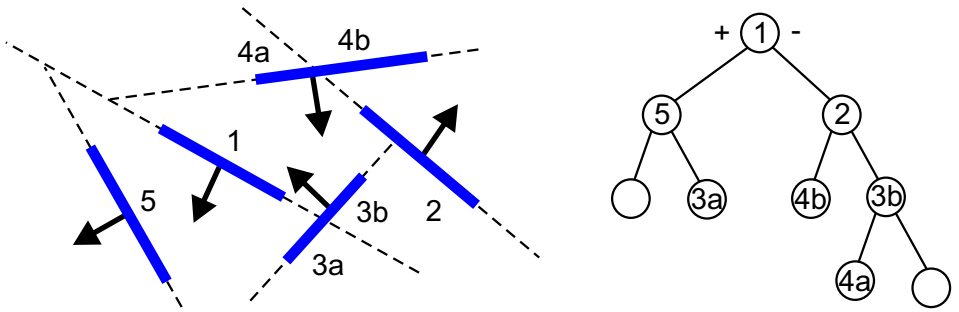
Mnoho implementací malířova algoritmu se omezuje na provádění jen těch nejjednodušších testů (např. testů 1 a 2). Takové zjednodušení vede k téměř lineární závislosti času na počtu ploch, protože časově nejnáročnějším se stává samotné vykreslení (rasterizace) ploch. Je-li scéna tvořena několika stovkami, či tisíci plošek, lze připustit i drobné chyby za cenu vysoké rychlosti řešení viditelnosti celé scény.

Malířův algoritmus se snadno implementuje a je proto mezi programátory oblíben. Nepotřebuje žádnou pomocnou paměť (srovnejme s požadavky na paměť hloubky). Na druhé straně však musí udržovat v paměti všechny (seřazené) plochy. Je příkladem algoritmu pracujícího v objektovém prostoru.

11.2.3 Malířův algoritmus se stromem BSP

Zajímavou variantou malířova algoritmu je jeho implementace za pomoci přídavné datové struktury nazývané *strom BSP* (*Binary Space Partitioning tree*). Cílem metody je uchovat popis objektů v prostoru v takovém tvaru, který je sice pohledově nezávislý, ale který je přitom pro libovolnou pozici pozorovatele schopen poskytnout údaje o správném pořadí vykreslování objektů.

Strom BSP se pro danou scénu nejprve jednorázově vytvoří a poté se opakovaně využívá pro různé pohledy. Je tedy vhodný pro statické scény, v nichž se pozice a velikosti objektů nemění. Strom BSP je binárním stromem, jehož každý vnitřní uzel představuje řez, rozdělující přidělený prostor na dvě části. Stromy BSP mají v počítačové grafice široké uplatnění, neboť hierarchické dělení prostoru na menší části pomáhá urychlit řadu algoritmů (viz též kapitola 14.2.2). Aplikace stromu BSP pro malířův algoritmus patří k nejznámějším.



Obrázek 11.5: Několik plošek (vlevo) a k nim vytvořený strom BSP (vpravo)

Při stavbě stromu postupujeme metodou shora dolů a rekurzívně dělíme trojrozměrný prostor obsahující zobrazované objekty. Metoda je vhodná pouze pro rovinné plošky, které před zahájením stavby stromu seskupíme do jediné množiny. Z ní vybereme jednu plošku a vedeme řez rovinou, v níž ploška leží. Tuto plošku odebereme z množiny plošek a množinu dále rozdělíme na dvě podmnožiny odpovídající dvěma poloprostorům určených řezem. Ty plošky, které byly řezem zasaženy, musíme rozdělit a jejich části, nazývané *fragmenty*, zapsat do odpovídajících podmnožin. Postup opakujeme, dokud je v některé množině více než jedna ploška.

Při výběru plošky, určující v každém kroku rekurzivní stavby stromu BSP řeznou rovinu, bychom měli dbát na to, aby řez neprotnul mnoho ostatních ploch a aby tak nevzniklo příliš mnoho fragmentů. Na obrázku 11.5 je ukázáno pět plošek a jeden z možných stromů BSP, při jehož stavbě vznikly čtyři fragmenty (z plošek 3 a 4). Pokud bychom začali prostor dělit rovinou koplanární s ploškou 3, dospěli bychom pouze ke dvěma fragmentům.

```
ZobrazBSP (S) {
   pokud (S je list) {
      Vykresli_plochy (S.kořen);
      return;
   }
   pokud (pozorovatel je v předním poloprostoru řezu S) {
      ZobrazBSP (S.zadní_větev);
      Vykresli_plochy (S.kořen);
      ZobrazBSP (S.přední_větev);
   }
   jinak {
      ZobrazBSP (S.přední_větev);
      Vykresli_plochy (S.kořen);
      ZobrazBSP (S.zadní_větev);
   }
}
```

Algoritmus 11.2: Řešení viditelnosti malířovým algoritmem za pomoci stromu BSP

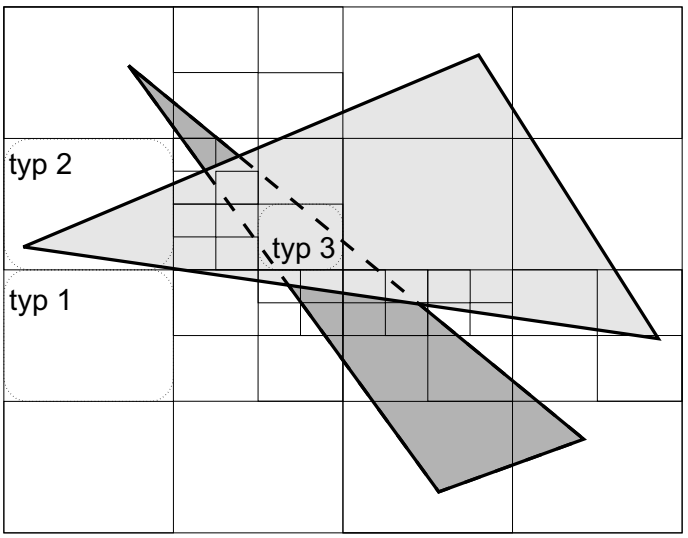
Při zobrazování procházíme strom BSP od kořene a v každém uzlu určujeme, který poloprostor je vůči pozorovateli vzdálenější a který bližší. Nejprve vykreslíme vzdálenější část prostoru, poté obsah aktuálního uzlu a nakonec oblast bližší k pozorovateli. Postup je uveden v algoritmu 11.2. V něm předpokládáme, že datová struktura S reprezentuje jeden uzel, který v položce *kořen* obsahuje údaje o plošce (případně několika ploškách), kterou prochází. Dále obsahuje ukazatele na levý a pravý podstrom (položky *přední_větev* a *zadní_větev*).

Strom BSP je možno stavět i nad celými tělesy. Pak není nutno prokládat řeznou rovinu plochou tělesa, ale stačí ji vést v jeho blízkosti. Odpadá tak omezení na rovinné plošky – tělesa mohou být modelována z libovolných elementů. Určení fragmentů pak ovšem bývá výpočetně náročnější. Jako řezy, které ve stromu BSP rozdělují prostor, bývají někdy voleny roviny kolmé na soustavu souřadnic (ortogonální strom BSP). To urychlí test, určující polohu pozorovatele vůči řezné rovině.

Strom BSP je výhodný pro uspořádání objektů, které dále nemění svoji polohu. Aktualizace stromu v případě dynamických objektů je časově náročná.

11.2.4 Dělení obrazovky

Další metoda pracující v prostoru objektů vychází z předpokladu, že složitý problém lze rozdělit na řadu menších, snáze řešitelných. Tento princip, nazývaný *rozděl a panuj* (*divide & conquer*), je ostatně velmi oblíben při návrhu paralelních algoritmů pro řešení složitých úloh na víceprocesorových počítačích. Algoritmus byl navržen J. Warnockem již v roce 1969 [Warn69] a v literatuře je označován jako *Warnock subdivision*. Algoritmus se snaží jednoduchými prostředky určit, jak má být vyplněno dané obdélníkové okno na obrazovce. Pokud je rozhodnutí složité, okno se rozdělí dvěma kolmými řezy v polovině na čtyři menší části a rekurzivním způsobem se algoritmus opakuje. Příklad postupného dělení okna je ukázán na obrázku 11.6.



Obrázek 11.6: Řešení viditelnosti rekurzivním dělením obrazovky

Při vyplňování okna mohou nastat následující možnosti:

1. Pokud do okna nezasahuje žádný objekt, zůstává vyplněné barvou pozadí.
2. Do okna zasahuje právě jedna plocha. Pak bude tato plocha uvnitř okna vyplněna, zbytek okna získá barvu pozadí.
3. Do okna zasahuje více ploch, ale plocha nejbližší k pozorovateli je zcela překrývá. Proto je okno vyplněno barvou této plochy.
4. Okno obsahuje komplikovanější část scény. Je proto rozděleno na čtyři menší okna a algoritmus je opakován pro každé z nich.

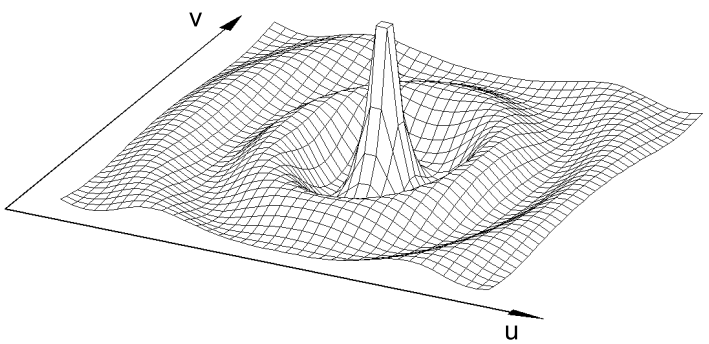
Na obrázku 11.6 jsou zvýrazněna některá dále nedělená okna, jejichž označení odpovídá pořadí výše uvedených situací. V případě, že velikost okna dosáhne velikosti pixelu, dělení je ukončeno. Pixel získá barvu té plochy, která je v tomto oknu nejblíže k pozorovateli.

Rychlost algoritmu závisí především na efektivitě testu průmětu plochy vůči oknu. V testu jsou použity obdobné výpočty, jako při ořezávání ploch v rovině (kap. 3.4). Při rekurzivním provádění algoritmu není třeba vyhodnocovat celý test polohy plochy vůči oknu znovu, protože vždy dvě strany okna zůstávají nezměněny. Díky dělení plošek je Warnockův algoritmus na rozdíl od malířova algoritmu schopen zpracovat i navzájem se prosekávající plochy.

Přestože v současné době již není tento algoritmus pro řešení viditelnosti využíván, princip plošného rozdělení složitého problému na lokálně jednodušší úlohy v něm použitý nalezneme v mnoha aplikacích počítačové grafiky.

11.2.5 Algoritmus plovoucího horizontu

Dosud uváděné algoritmy řešení viditelnosti byly univerzální v tom smyslu, že zpracovávaly množinu prostorových ploch bez využití případných dalších informací o jejich topologii. Algoritmus popsaný v této části je příkladem metody, která předpokládá speciální uspořádání vstupních dat.



Obrázek 11.7: Graf funkce dvou proměnných v podobě soustavy řezů

V některých aplikacích je třeba zobrazit plochu určenou explicitní funkcí dvou proměnných:

$$w = f(u,v), \quad \text{kde } u \in \langle u_{min}, u_{max}\rangle,\ v \in \langle v_{min}, v_{max}\rangle. \tag{11.1}$$

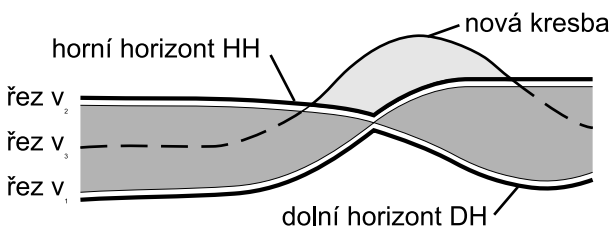
Takovou funkci nejsnáze zobrazíme soustavou řezů kolmých na osy u a v. Příklad na obrázku 11.7 znázorňuje funkci $w = 4\cos(\sqrt{2u^2+v^2})/\sqrt{u^2+v^2}$ v rozsahu $u, v \in \langle -10, 10\rangle$.

Algoritmus řešení viditelnosti je založen na předpokladu, že po provedení pohledové transformace kreslíme jednotlivé řezy od pozorovatele směrem dozadu. Uvažujme o situaci na ob-

rázku 11.7, kde vzdálenější řezy mají větší souřadnici v. Viditelnost budeme určovat metodou *plovoucího horizontu*, která vykresluje grafické prvky (úsečky) každého dalšího řezu jen tehdy, pokud se nacházejí buď nad, nebo pod již zaplněnou oblastí na obrazovce. Algoritmus využívá dvou pomocných polí – *horního horizontu* HH a *dolního horizontu* DH. Počet prvků v těchto polích je shodný s počtem pixelů obrazovky ve vodorovném směru. Horizonty obsahují pro každou souřadnici x jí odpovídající nejvyšší, resp. nejnižší souřadnici y dosud nakresleného pixelu na obrazovce. Při kreslení dalšího řezu jsou souřadnice bodů řezu porovnávány s oběma horizonty a nakresleny pouze v případě, že nepadnou do oblasti těmito horizonty omezené. Postup je popsán algoritmem 11.3. Na obrázku 11.8 je naznačena situace po provedení dvou řezů v_1 a v_2 při zpracování řezu v_3. Silně jsou nakresleny oba horizonty.

1. Inicializuj všechny prvky horizontu HH hodnotou $-\infty$ a všechny prvky horizontu DH hodnotou $+\infty$
2. Po krocích rostoucí $v_j \in \langle v_{min}, v_{max} \rangle$ určuje řezovou křivku $f(u, v_j)$
3. Pro všechna $u_i \in \langle u_{min}, u_{max} \rangle$ vypočítej body $w_{ij} = f(u_i, v_j)$
   (a) pohledovou transformací transformuj vypočítaný bod $[u_i, v_j, w_{ij}]$ na bod $[x, y]$ v souřadnicích obrazovky
   (b) je-li $y \leq HH[x]$ a zároveň $y \geq DH[x]$, pokračuj zpracováním dalšího u_i
   (c) nakresli bod $[x, y]$ a podle hodnoty y aktualizuj buď $HH[x]$, nebo $DH[x]$

Algoritmus 11.3: Řešení viditelnosti grafu funkce dvou proměnných metodou plovoucího horizontu



Obrázek 11.8: Plovoucí dolní a horní horizont

Uvedený algoritmus představuje pouze základní řešení. Rasterizace je při něm prováděna v trojrozměrném prostoru, takže krok měnící se souřadnice u_i (viz krok 3 v alg. 11.3) musí odpovídat velikosti jednoho pixelu (indexu plovoucího horizontu). Praktické rozšíření algoritmu spočívá ve vzorkování funkce ve větších intervalech a následné rasterizaci takto vzniklé posloupnosti úseček. Každý pixel rasterizované úsečky je podrobován krokům (3b) a (3c) v základním algoritmu. Je také zřejmé, že algoritmus kreslí pouze řezy kolmé na osu v, tj. s konstantní souřadnicí v. Abychom získali výsledek na obrázku 11.7, musíme do algoritmu zahrnout i zpracování příč-

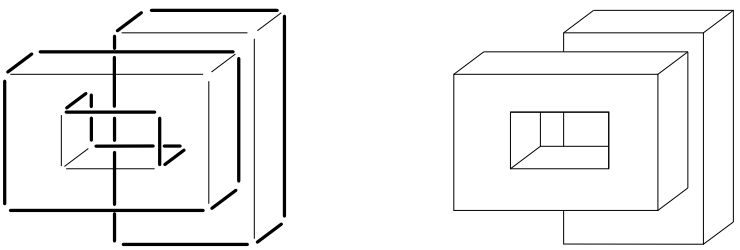
ných řezů – spojnic mezi dosud kreslenými řezy. Krátké křivky nebo úsečky, které jsou kolmé na osu u, lze kreslit obdobným postupem vždy mezi dvěma řezy kolmými na osu v.

Přestože byl algoritmus původně navržen pro zobrazování grafů funkcí dvou proměnných, lze jej použít i pro vykreslení dat zadaných odlišným způsobem. Známé jsou aplikace zobrazování terénu z naměřených prostorových souřadnic, existují i varianty pro kreslení vodorovných vrstevnicových řezů, algoritmus je použitelný také pro řešení viditelnosti plátů, např. Bézierových ploch. Při bližším zkoumání algoritmu plovoucího horizontu vidíme, že se vlastně jedná o „obrácený“ malířův algoritmus, tj. obraz scény je kreslen zepředu dozadu. Vidíme zde, jak jsou v počítačové grafice upravovány známé postupy pro různá speciální zadání a také to, že znalost kvalitních algoritmů může pomoci nalézt efektivní řešení v různých situacích.

11.3 Liniové algoritmy viditelnosti

Jako příklad uvedeme Robertsův algoritmus [Robe63], který reprezentuje nejstarší a programátorsky nejjednodušší metodu určení viditelnosti hran. Původní algoritmus byl určen pro řešení viditelnosti skupiny konvexních mnohostěnů, lze jej však rozšířit i na obecné mnohostěny.

Algoritmus 11.4 postupně zpracovává jednotlivé hrany těles. Pro každou hranu testuje, zda je zakrývána jiným tělesem. Pokud je zakryta jen částečně, dochází k rozdělení hrany na zakrytou a nezakrytou část. Každá nezakrytá část se stává novou, testovanou hranou.



Obrázek 11.9: Obrysové hrany rozdělí potenciálně viditelné hrany na elementární úseky s neměnnou viditelností

Robertsův algoritmus se stal základem mnoha dalších metod, lišících se např. seřazením hran, použitím urychlovacích testů při hledání zákrytů, apod. Většina úprav přitom zachovává hlavní myšlenku algoritmu, totiž rozdělení potenciálně viditelných hran na elementární úseky, na nichž se nemůže změnit viditelnost. K nalezení úseků s neměnnou viditelností se s výhodou použijí obrysové hrany. Pouze ty indikují případnou změnu viditelnosti, takže pouze ony mohou dělit hrany na menší úseky. Na obrázku 11.9 vlevo je vidět množinu elementárních úseků pro jednoduchou scénu. Obrysové hrany jsou nakresleny silnější čárou.

1. Rozděl hrany na zadní, obrysové a přední. Zadními hranami se dále nezabývej.
2. Vytvoř prázdný seznam V potenciálně viditelných hran
3. Pro všechny obrysové a přední hrany H_i dělej
   (a) Přidej hranu H_i do seznamu V potenciálně viditelných hran
   (b) Pro všechny mnohostěny M_j dělej:
      i. Pro všechny hrany H_k ze seznamu V dělej:
         A. Vyjmi hranu H_k ze seznamu V
         B. Testuj hranu H_k na zakrytí mnohostěnem M_j
         C. Pokud jsou na hraně nalezeny jeden nebo dva nezakryté úseky, zařaď úseky do seznamu V
         D. Pokud mnohostěn celou hranu zakrývá, dále se jí nezabývej
   (c) Vykresli všechny hrany ze seznamu V a vyprázdni jej

Algoritmus 11.4: Robertsův algoritmus pro určení viditelnosti hran konvexních mnohostěnů

Zkoumání zákrytu potenciálně viditelných elementárních úseků s nekonvexními mnohostěny je poměrně zdlouhavé. V literatuře nalezneme Appelův algoritmus [Appe67], který využívá obrysových hran ještě důsledněji. Potenciálně viditelné hrany se v něm opět testují vůči obrysovým. Průsečíky, ve kterých testovaná hrana „vstupuje“ pod obrysovou hranu (přechází tedy za jiné těleso), jsou zaznamenány do seznamu a ohodnoceny jako vstupní či výstupní. Seřazené průsečíky pak určují viditelnost jednotlivých úseků dané hrany.

Poměrně jednoduchý postup tohoto algoritmu je komplikován nutností ošetřování mnoha možných singulárních případů, kdy např. průmět vrcholu obrysové hrany leží v průmětu testované hrany a naopak. V literatuře [Blin88a] jsou uváděny různé možnosti ošetření těchto situací.

Z dalších algoritmů pro určování viditelnosti hran a ploch připomeňme ještě Weiler–Athertonův algoritmus [Weil77]. Ten je založen na rychlém ořezání (nekonvexních) ploch vybranou blízkou plochou. Výsledkem tohoto ořezání je rozdělení zbylých ploch do dvou seznamů. V prvním seznamu jsou ty plochy a ořezané části, jejichž průměty leží uvnitř vybrané plochy a které jsou tedy zakryté. Ve druhém seznamu jsou plochy ležící vně. Na ně je opakovaně použita tatáž metoda, tedy výběr nové blízké plochy a ořezání. Tento algoritmus je příkladem postupu, řešícího viditelnost ploch (HSE). Plochy nebo jejich ořezané části, které byly označeny

jako viditelné, lze dále vybarvit.

11.4 Zpracování poloprůhledných objektů

Přítomnost poloprůhledných objektů, resp. těles s poloprůhlednými plochami, způsobuje potíže v některých algoritmech řešení viditelnosti. Poloprůhledné plochy nezakrývají vzdálenější objekty, naopak je třeba barvu poloprůhledné plochy smíchat s barvou objektů v pozadí. Toto míchání barev bylo uvedeno v části 4.4.1 pod názvem *alfa míchání*. Připomeňme, že neprůhledná plocha má $\alpha = 1$, u zcela průhledné plochy je $\alpha = 0$.

Při zpracování poloprůhledných ploch je třeba dodržet zásadu jejich vykreslování (a s ním spojeného míchání barev) odzadu dopředu. Jiný způsob není možný, protože alfa míchání není komutativní operací. Máme-li například za sebou dvě poloprůhledné plochy, z nichž průhlednost přední je určena hodnotou α_N a průhlednost zadní hodnotou α_F, pak mícháním jejich barev C_N a C_F na barevném pozadí C_B ve správném pořadí získáme výslednou barvu jako

$$\alpha_N C_N + (1 - \alpha_N)[\alpha_F C_F + (1 - \alpha_F)C_B].$$

Pokud vykreslíme plochy v obráceném pořadí, získáme odlišný výsledek

$$\alpha_F C_F + (1 - \alpha_F)[\alpha_N C_N + (1 - \alpha_N)C_B],$$

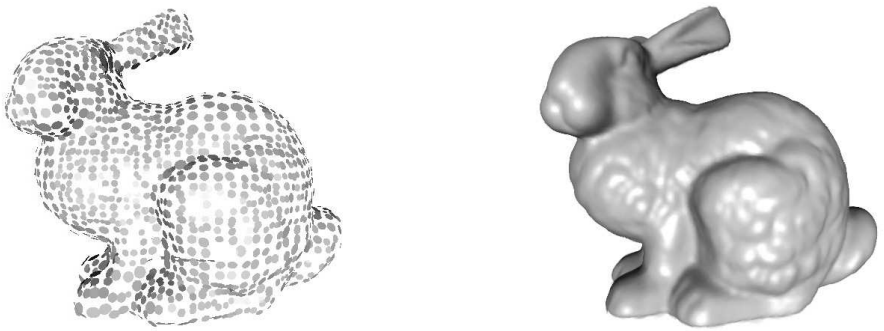
v němž subtraktivní člen $\alpha_N \alpha_F$ není aplikován na barvu C_F, nýbrž chybně na barvu C_N. Požadavek správného pořadí vykreslování poloprůhledných ploch přirozeně splňuje malířův algoritmus. Podívejme se, jaký vliv má přítomnost poloprůhledných ploch na nejčastěji používaný algoritmus řešení viditelnosti – paměť hloubky. Základní algoritmus je nutno modifikovat následujícím způsobem:

1. Inicializuj paměť hloubky a obrazovou paměť.
2. Postupně načítej jednotlivé plochy. Je-li plocha neprůhledná, okamžitě ji zpracuj jako v základním algoritmu. V opačném případě její zpracování odlož (plochu si zapamatuj).
3. Po zpracování všech neprůhledných ploch seřaď zapamatované poloprůhledné plochy podle vzdálenosti.
4. Postupně zpracuj poloprůhledné plochy od nejvzdálenější k nejbližší jako v základním algoritmus s tím, že namísto zápisu do obrazové paměti použiješ alfa míchání.

Je zřejmé, že přítomnost poloprůhledných ploch výrazně snižuje efektivitu algoritmu paměti hloubky. Do základní metody přidává nový prvek – řazení. Tím je výhoda původně sekvenčního algoritmu potlačena. Pokud chceme docílit co nejvyšší rychlosti zobrazování, měli bychom poloprůhledné plochy používat s rozmyslem, tedy v co nejmenším množství.

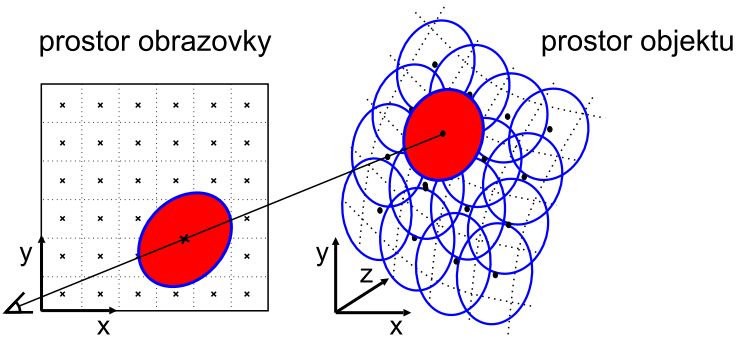
11.5 Zobrazování bodově reprezentovaných objektů

Při pohledu na běžnou scénu, například otevřenou krajinu, si můžeme všimnout, že většina objektů je vzdálená a tudíž velmi malá. V počítačové grafice pracujeme v diskrétním rastru a tak se mnoho objektů v takové scéně promítne do jediného pixelu. Je-li nějaký objekt reprezentován sítí trojúhelníků, bude častým případem promítnutí souřadnic trojúhelníku do jediného pixelu. Tato skutečnost vedla k zavedení *bodů* jako geometrických primitiv.



Obrázek 11.10: Bodová reprezentace králíčka (vlevo) zobrazená jako spojitá plocha (vpravo)

Bod je vzorek povrchu trojrozměrného objektu. Je určen polohou $[x, y, z]$ a atributy, nejčastěji normálovým vektorem $\vec{n} = (n_x, n_y, n_z)$, barvou $[r, g, b]$ a případně dalšími údaji např. exponentem h zrcadlového odrazu Phongova osvětlovacího modelu (10.11). Objekt je reprezentován jako množina takovýchto bodů na povrchu a úkolem je zobrazit ho tak, jako by byl reprezentován přesně.

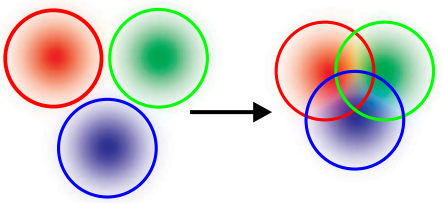


Obrázek 11.11: Kruhy reprezentující povrch objektu se promítnou na obrazovku jako elipsy

Problémem množin bodů je jejich topologie. Zatímco v případě sítí trojúhelníků víme, kde jeden trojúhelník začíná a druhý končí, u izolovaných bodů tomu tak není. Při zobrazování množin bodů mohou vzniknout nechtěné nespojitosti (díry) na povrchu výsledného objektu. Údaj, který máme k dispozici, je *lokální hustota bodů*. Z této znalosti můžeme zobrazit každý bod tak velký, aby se překrýval s nejbližšími body, čímž zamezíme vzniku děr.

Na této myšlence je založena nejjednodušší metoda pro zobrazování bodových objektů [Rusi00]. Každý bod je zobrazen jako bodové primitivum grafické zobrazovací knihovny a jeho velikost je nastavena tak, aby se nejbližší body vždy překrývaly. Tento přístup se stal základem systému QSplat, který slouží k interaktivnímu prohlížení vysoce detailních objektů. Objekty jsou reprezentovány hierarchickou datovou strukturou, jejíž uzly odpovídají bodům na povrchu. Počet bodů pro zobrazení každého snímku je určen dynamicky v závislosti na požadované snímkové frekvenci a rychlosti použitého počítače.

Nevýhodou zobrazování pomocí bodových primitiv je nízká kvalita výsledných obrázků při pohledu zblízka. Bod je zobrazen jako čtverec s konstantní barvou, neexistuje interpolace mezi sousedními body, obrázky trpí aliasingem. Řešení těchto problémů poskytuje *metoda vážených eliptických oblastí* (*EWA surface splatting*) [Zwic01], původně určená pro mapování textur. Její podstatu demonstruje obrázek 11.11. Body na povrchu objektu jsou považovány za kruhy (kolmé k normále bodu), které zcela pokrývají povrch objektu. Při zobrazování se každý kruh promítne a zobrazí na obrazovku jako elipsa. Barva c pixelu o souřadnicích $[i, j]$ se vypočítá jako vážený průměr barev bodů, jejichž promítnuté kruhy překrývají daný pixel:



Obrázek 11.12: Barva pixelů mezi středy promítnutých kruhů se určí jako vážený průměr barev bodů

$$c(i,j) = \frac{\sum_k c_k w_k(i,j)}{\sum_k w_k(i,j)},$$

kde c_k je barva $k-$tého příspěvku a $w_k(i,j)$ je váha bodu k vzhledem k pixelu (i,j) (viz obr. 11.12). Váha na povrchu kruhu je dána dvojrozměrným Gaussovým rozložením.

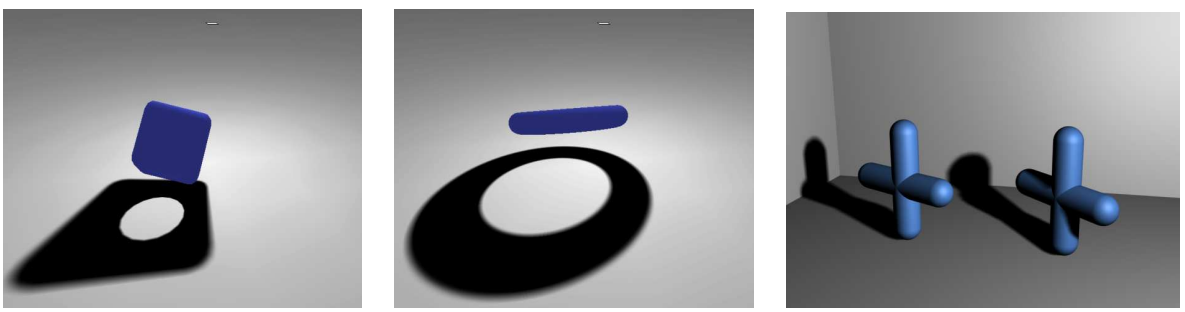
Bodová reprezentace není vhodná pro ploché nebo jednoduché objekty, stejně jako selhává při reprezentaci ostrých hran. Tyto problémy vedly k vývoji *hybridních technik* [Chen01, Wand02], které kombinují výhody bodů a trojúhelníků.

Metoda reprezentace objektů je živá a zasahuje i do jiných oblastí informatiky. V počítačovém vidění je důležitá úloha získávání bodů na povrchu reálných objektů, trojrozměrné skenování [Levo00, Matu02] či vzorkování syntetických objektů. Další oblastí je interaktivní editace tvaru bodově reprezentovaných objektů [Paul03], malování na takovéto objekty [Zwic02], nebo spektrální analýza geometrie [Paul01].

Kapitola 12

Stíny

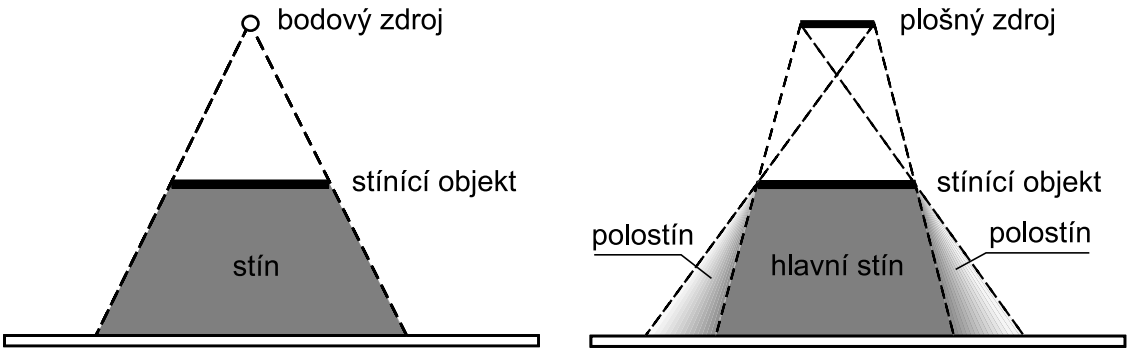
Stíny hrají důležitou roli při prostorovém vnímání člověka. Pomáhají pochopit vzájemné rozmístění objektů, jejich tvar a rozměry a poskytují dobrou informaci o vlastnostech a poloze zdrojů světla (viz obrázek 12.1). V počítačové grafice se proto techniky vytvářející stíny stávají důležitým prostředkem pro zvýšení věrohodnosti a realističnosti zobrazované scény. Globální zobrazovací metody jako metoda sledování paprsku a radiační metoda, které zobrazují scénu společně s jejími stíny, jsou výpočetně náročné. Proto se v aplikacích, kdy je potřeba zobrazovat scénu v reálném čase nebo s ní interaktivně pracovat, dává přednost podstatně rychlejším samostatným technikám pro generování stínů. Tyto metody většinou převádějí problém nalezení stínu na problém geometrický, nejčastěji na algoritmus řešení viditelnosti.



Obrázek 12.1: Přítomnost stínu informuje o tom, že v kostce (vlevo) je otvor a že plochý útvar (uprostřed) je ve skutečnosti prstenec. Obrázek vpravo ukazuje vržené stíny na podlaze a na stěně. Pravý křížek je přitom vykreslen s *vlastním* i *vrženým* stínem, zatímco u levého křížku *vlastní* stín chybí. Každý z křížků je reprezentován sítí trojúhelníků jako jediný objekt.

Tvar a velikost stínu závisejí na vzájemné poloze světelného zdroje, stínícího objektu

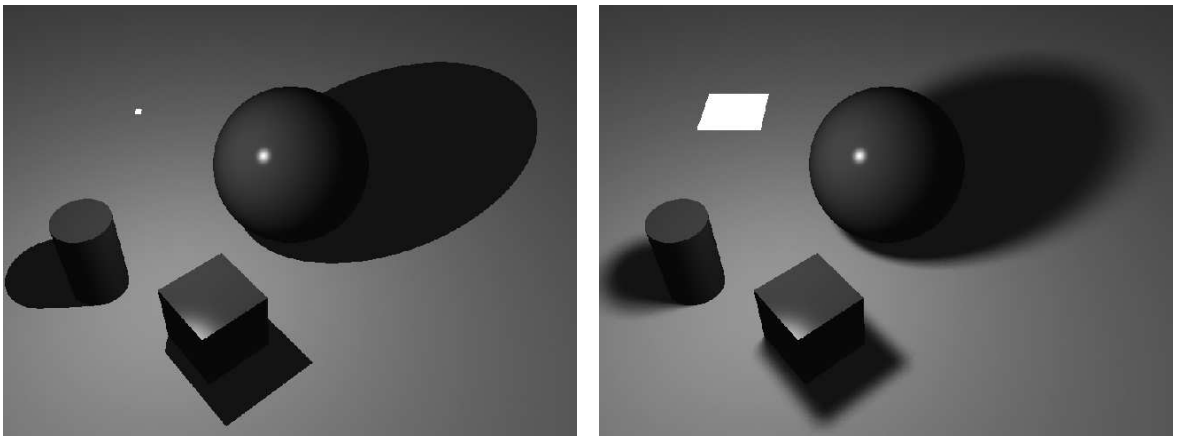
a objektu, na který stín dopadá. Tvar a velikost světelného zdroje pak ovlivňují charakter stínu. Bodové světelné zdroje, pro svoji jednoduchost oblíbené v počítačové grafice, vytvářejí *ostré stíny* (*hard shadows*) s přesně vymezenou hranicí (viz obrázek 12.2 a obrázek 12.3). V případě bodových zdrojů totiž můžeme pro každý bod scény jednoznačně určit, zda je z něj bodový zdroj vidět nebo zda je zastíněn jiným objektem. Bodové zdroje se však v reálném světě nevyskytují a stíny s ostrou hranicí proto nepůsobí věrohodně. Mnohem přirozenější jsou tzv. *měkké stíny* (*soft shadows*), které vznikají při osvětlení plošnými světelnými zdroji. Měkké stíny mají rozostřenou hranici přechodu mezi *hlavním stínem* (*umbra*) a plně osvětlenou plochou. Tato oblast se nazývá *polostín* (*penumbra*) a její velikost závisí na velikosti a vzdálenosti plošného zdroje a objektů ve scéně. Polostín se s rostoucí velikostí světelného zdroje zvětšuje, zatímco hlavní stín se naopak zmenšuje a může i zcela zmizet.



Obrázek 12.2: a) Ostrý stín způsobený bodovým zdrojem světla b) stín s polostínem ze zdroje plošného

V počítačové grafice se často rozlišují dva druhy stínů – *vlastní* (*self shadow*) a *vržený* (*cast shadow*). Vržený stín vnímáme nejčastěji. Je to stín, který vrhá jeden objekt na druhý a který tak pomáhá rozpoznat umístění těchto objektů v prostoru. Vlastním stínem se označuje stín, který objekt vrhá sám na sebe. Ve vlastním stínu pochopitelně leží všechny plochy tělesa odvrácené od světla (viz obrázek 12.3). U těchto ploch je však problém určení stínu vyřešen již přímo ve fázi stínování (kapitola 10.7), a proto si často ani neuvědomujeme, že se také jedná o stín. Mnohem důležitější jsou proto ty vlastní stíny, které vrhá jedna část tělesa na jinou. Některé algoritmy z této kapitoly jsou schopny nalézt pouze vržené, nikoliv vlastní stíny.

Generování stínů zcela neprůhledných objektů je poměrně dobře vyřešenou úlohou, byť se díky novým grafickým akcelerátorům stále objevují různá vylepšení a nové přístupy. Mnohem složitější je situace, kdy vyšetřujeme stín vržený částečně průhledným objektem. Procházející světlo je při průchodu poloprůhledným tělesem částečně pohlceno a stupeň tohoto pohlcení se



Obrázek 12.3: Ostrá hranice stínu vzniklá osvětlením z bodového zdroje (vlevo) a rozmazaná hranice jako důsledek osvětlení plošným světelným zdrojem (vpravo)

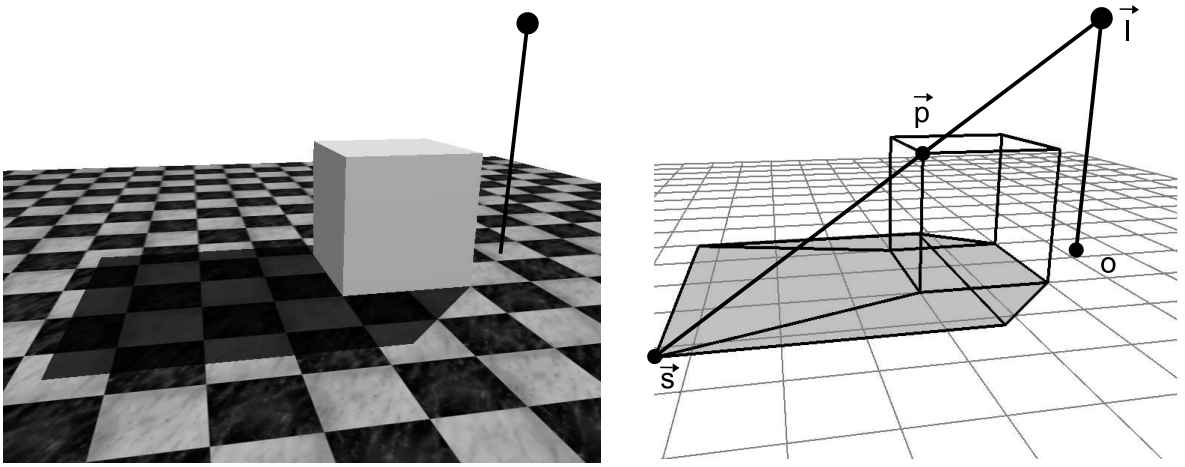
liší pro různé vlnové délky světla. Tato závislost se navíc liší v závislosti na materiálu, ze kterého se těleso skládá. Simulování absorbce by tedy mělo modelovat změnu barvy procházejícího světla. Druhý jev, který komplikuje výpočet vržených stínů průhledných objektů, je tzv. *disperze* (rozptyl) směru šíření světla. Dráha světla procházejícího materiálem je mnohonásobně měněna, navíc pro různé vlnové délky různě. Zjistit úplnou cestu světla není snadné. Algoritmy, které tyto jevy simulují, jsou časově velice náročné a obyčejně pracují spolehlivě jen s určitou třídou objektů. Tyto algoritmy patří do kategorie globálních zobrazovacích metod (kap. 15), a proto se jimi zde nebudeme zabývat.

V této kapitole nejprve uvedeme dvě metody pro výpočet stínů pracující s polygony a poté popíšeme metodu založenou na algoritmu Z–buffer (viz odstavec 11.2.1). Všechny algoritmy uvedené v této části knihy vycházejí z myšlenky, že to, co je vidět z místa světelného zdroje, je tímto zdrojem ovlivněné, zatímco zbytek scény je vůči tomuto zdroji ve stínu. Tímto způsobem je možno úlohu nalezení stínu převést na algoritmus řešení viditelnosti. Základní princip každé z metod vysvětlíme pro scény s jedním světelným zdrojem. Superpozicí světel pak lze metody snadno rozšířit na scény s více zdroji.

12.1 Projekční metody

Základní metoda [Blin88b], určená pro scény s polygonální reprezentací objektů, vypočítává stíny pomocí vhodně zvoleného zobrazení. S ohledem na polohu světelného zdroje se pro každou (cílovou) plochu, na kterou může dopadat stín, nalezne transformace zobrazující do roviny

této plochy libovolný objekt jako dvojrozměrný polygon. Tuto transformaci lze v homogenních souřadnicích (viz kapitola 21.1) popsat projekční maticí 4×4.



Obrázek 12.4: Projekční stíny pro bodový zdroj světla (znázorněný větším puntíkem)

Pro odvození transformační matice uvažujme situaci na obrázku 12.4 a pro jednoduchost nejprve předpokládejme, že rovina, na kterou stín dopadá, je totožná s rovinou $y = 0$. Bude-li poloha světelného zdroje vyjádřena radius vektorem $\vec{l} = (l_x, l_y, l_z)$, pak libovolný bod $\vec{p} = (p_x, p_y, p_z)$ tělesa bude vrhat stín do bodu $\vec{s} = (s_x, 0, s_z)$ podle vztahu

$$\vec{s} = \vec{l} + t(\vec{p} - \vec{l}). \tag{12.1}$$

Parametr t odvodíme z podmínky $s_y = 0$ jako

$$t = \frac{l_y}{l_y - p_y}.$$

Dosazením do vztahu 12.1 získáme souřadnice bodu $\vec{s}$. Výpočet lze vyjádřit jako transformaci ve tvaru:

$$[s_x\ 0\ s_z\ w] = [p_x\ p_y\ p_z\ 1] \begin{bmatrix} l_y & 0 & 0 & 0 \\ -l_x & 0 & -l_z & -1 \\ 0 & 0 & l_y & 0 \\ 0 & 0 & 0 & l_y \end{bmatrix}.$$

Při odvozování transformační matice pro libovolnou plochu ležící v rovině $\vec{n}.\vec{x} + d = 0$, kde $\vec{n}$ je normálový vektor roviny a $\vec{x}$ její libovolný bod, budeme postupovat obdobně. Dosazením

$\vec{x}$ za $\vec{s}$ ve vztahu 12.1 získáme parametr t

$$t = \frac{\vec{n}.\vec{l} + d}{\vec{n}.(\vec{l} - \vec{p})}.$$

Výpočet souřadnic bodu $\vec{s}$ můžeme opět vyjádřit pomocí transformační matice

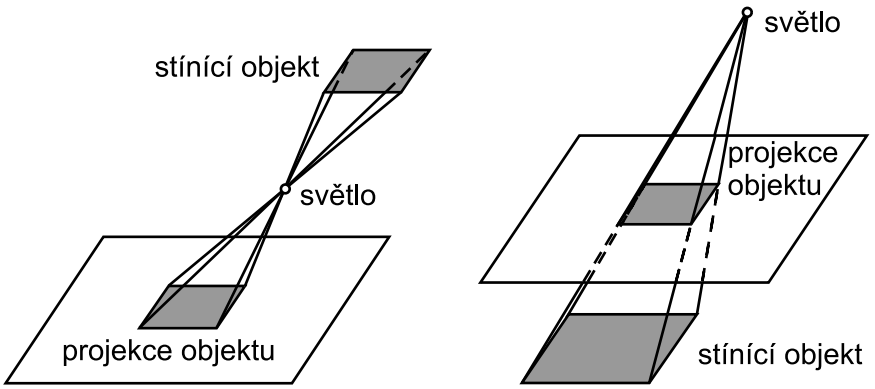
$$\mathbf{M} = \begin{bmatrix} \vec{n}.\vec{l} + d - l_x n_x & -l_y n_x & -l_z n_x & -n_x \\ -l_x n_y & \vec{n}.\vec{l} + d - l_y n_y & -l_z n_y & -n_y \\ -l_x n_z & -l_y n_z & \vec{n}.\vec{l} + d - l_z n_z & -n_z \\ -l_x d & -l_y d & -l_z d & \vec{n}.\vec{l} \end{bmatrix}.$$

Chceme-li zobrazit výslednou scénu, vykreslíme ji nejprve bez stínů a poté dokreslíme stínové polygony. Zde se setkáme s několika problémy. Stínový polygon může zasahovat i mimo cílovou plochu, na kterou má dopadnout. Tehdy je třeba jej ořezat. Dále, v důsledku numerických nepřesností, může stínový polygon dopadnout nepatrně nad nebo pod cílovou plochu a ztmavit tak jen některé z pixelů, které mají ležet ve stínu. Ani mírné posunutí stínového polygonu směrem ke světlu není dokonalým řešením, neboť stanovení velikosti posunu není jednoznačné.

Bezpečné a přitom elegantní řešení využívá kombinace z–bufferu a šablony (*stencil buffer*). Nejprve se vyřeší viditelnost scény běžným způsobem a pro každý výsledný pixel se do šablony zaznamená identifikátor viditelné plochy. Ve druhé části se postupně vytvářejí polygony vržených stínů a opět se zobrazují z pohledu kamery. Nyní se však již neřeší viditelnost, tj. neprovádí se hloubkový test, pouze se porovnává, zda v daném pixelu došlo ke shodě identifikátoru viditelné plochy a identifikátoru k ní příslušnému stínovému polygonu. Pokud ano, je tento pixel ve stínu a je třeba ztmavit jeho barvu. Současně je nutno si zapamatovat, že pixel byl již v rámci právě vyhodnocovaného světelného zdroje jednou ztmaven, aby nedošlo k chybnému *opakovanému ztmavení* (*double-blending*).

Jednoduchost projekční metody je zaplacena několika nedostatky. První nevýhodou je skutečnost, že algoritmus je určen pro scény obsahující jen objekty s ploškovou reprezentací – cílová plocha je částí roviny. Největším nedostatkem je však skutečnost, že transformace popsaná maticí **M** zobrazuje zvolený objekt do plochy bez ohledu na jeho umístění ve scéně. K tomu, aby těleso mohlo vrhat na plochu stín, však musí ležet v prostoru mezi touto plochou a světelným zdrojem. Jak ukazuje obrázek 12.5, metoda promítá objekt do plochy i v případě, kdy se objekt nachází v místech, odkud nemůže ovlivňovat její osvětlení. Výsledkem jsou pak falešné, nereálné stíny.

Existují techniky, u nichž falešné stíny nevznikají [Heck97]. Místo transformace ze 3D do 2D, která „slisuje“ stínící objekt do roviny, lze použít speciální transformaci ze 3D do 3D. Tato transformace zobrazí tělesa, způsobující falešné stíny, do prostoru mimo pohledový objem. Tím



Obrázek 12.5: Falešné stíny vytvářené projekční metodou. Situace vlevo, označovaná jako *anti-shadow*, nastává pro $t < 0$, zatímco situace na pravém obrázku, *false-shadow*, pro $0 \leq t < 1$.

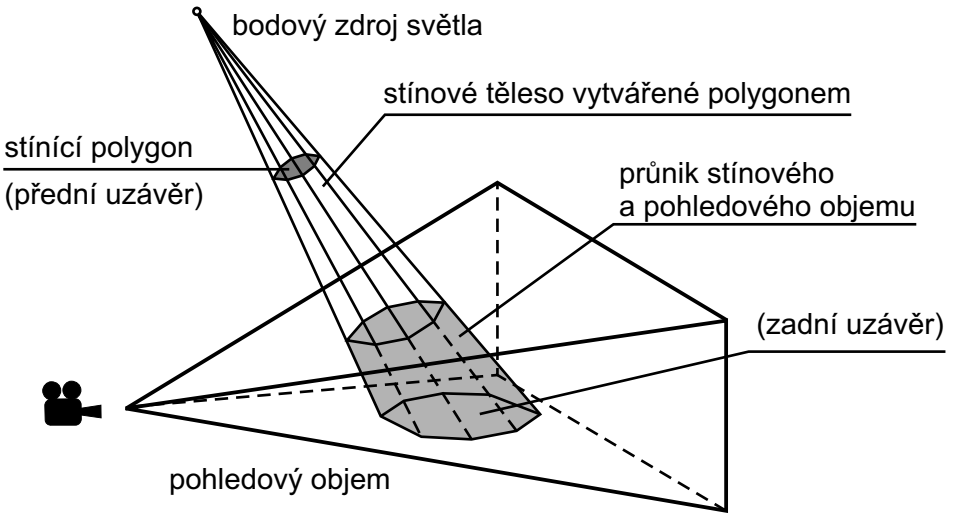
se vyloučí (tj. ořízne) objekty, které se nacházejí za plochou nebo za zdrojem osvětlení a které jsou původcem falešných stínů.

Stínové polygony lze chápat i jako texturu, která se promítne (mapuje) na cílovou plochu. Celou scénu můžeme tedy promítnout z pohledu světelného zdroje na cílovou plochu a výsledek uložit jako texturu, kterou posléze na plochu mapujeme běžným způsobem. Pokud obraz stínu v textuře rozostříme vhodným filtrem, získáme přibližné, avšak dostatečně věrohodné polostíny. Tento přístup je používán zejména v počítačových hrách a aplikacích, v nichž je třeba dosáhnout vysoké rychlosti zobrazování. Často se za cílové plochy volí jen vybrané objekty, jakými jsou podlaha a stěny. Extrémní zjednodušení pak spočívá ve vytvoření *pseudostínu* (*fake shadow*), který nemá nic společného s tvarem objektů vrhajících stín. Takový pseudostín mívá tvar elipsy, jejíž rozměry odpovídají maximálním rozměrům objektu (např. počítačové postavy), který vrhá stín. I takto triviální postup přináší zlepšení prostorového vjemu.

12.2 Stínové těleso

Další algoritmus, který pracuje v objektovém prostoru, se nazývá *stínové těleso* (*shadow volume*) [Crow77]. Společně s metodou *stínové paměti hloubky* je jedním z nejčastěji používaných algoritmů pro zjišťování stínů v reálném čase.

Algoritmus v základní podobě pracuje s polygony (jeho rozšíření na nonmanifoldy a nerovinné plochy navrhl Bergeron [Berg86]), bodovými světly a poskytuje tedy pouze ostré stíny. Základní myšlenkou je vytvoření tzv. *stínového tělesa* pro každý ze stínících objektů. Jak ukazuje obrázek 12.6, stínové těleso ohraničuje část prostoru ve scéně, ze kterého není přes



Obrázek 12.6: Stínové těleso a jeho průnik s pohledovým objemem podle [Watt92]

stínící objekt (v tomto případě jediný polygon) světelný zdroj vidět, a vymezuje tak zdrojem neosvětlený prostor. Stínové těleso je potřeba vytvořit pro každý ze stínících objektů. Známe-li všechna stínová tělesa, stačí během zobrazování testovat vzájemnou polohu zobrazovaných objektů, resp. jejich povrchových polygonů, s těmito tělesy. Výsledkem testu může být jedna z následujících situací:

- Polygon leží celý uvnitř stínového tělesa – je tedy ve stínu,
- polygon leží celý vně stínového tělesa – je osvětlený světelným zdrojem, a konečně
- polygon leží částečně ve stínovém tělese – je nutné provést rozdělení polygonu na osvětlenou a neosvětlenou část.

Určení stínového tělesa pro jeden samostatný stínící polygon je snadné. V případě obecného stínícího objektu však musíme nejprve nalézt jeho obrys – ten určuje hranici stínu. Obrys je tvořen obrysovými hranami, tzv. *silhouette edges* (viz kapitola 11). Každá obrysová hrana definuje jednu stěnu stínového tělesa. Vytváření stínových těles ze siluety není nutnou podmínkou. Je možno vytvořit stínové těleso pro každou (ke světlu přivrácenou) plochu objektu, což však zvyšuje počet stínových těles a prodlužuje výpočet stínů.

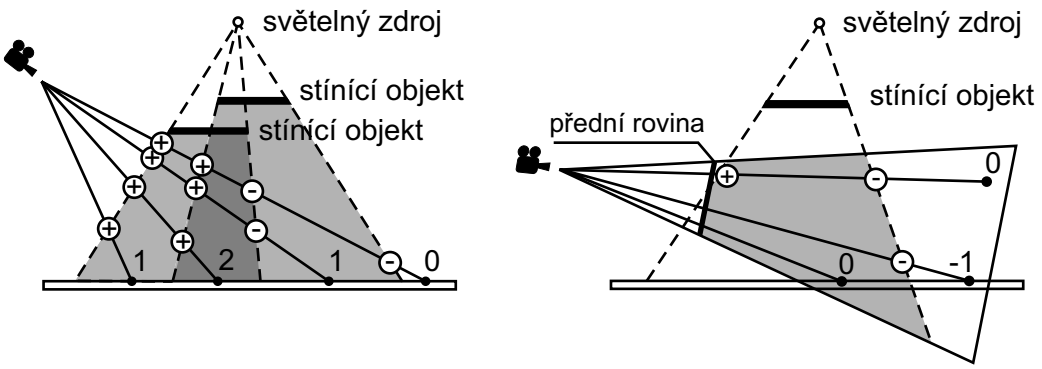
Testování vzájemné polohy polygonu a stínového tělesa se může zdát obtížné – zejména pokud polygon zasahuje do tělesa jen částečně. Většina algoritmů nerozděluje polygony geometrickými metodami na dílčí polygony, ale pracuje v prostoru rastru. Tehdy se vysílají testovací

paprsky ze středu promítání každým pixelem směrem k zobrazovanému povrchu ve scéně. Během cesty paprsku se počítá, kolikrát paprsek vstoupil do nějakého stínového tělesa a kolikrát je opustil (obrázek 12.7 vlevo). Pokud je rozdíl těchto hodnot nenulový, znamená to, že paprsek neopustil všechna stínová tělesa, do kterých vstoupil a bod na povrchu tělesa, ke kterému dorazil, musí ležet ve stínu. Můžeme si to také představit tak, že ke kameře přivrácené plochy stínového tělesa „přesouvají“ vše, co se nachází za nimi, do stínu, zatímco odvrácené plochy jejich účinek ruší. K tomu, aby těleso nebo jeho část ležely ve stínu, pak musí platit, že se nacházejí za přivrácenými a současně před odvráceným plochami stínových těles.

K rozlišení těchto situací můžeme využít hloubkový test. Nejprve vyřešíme viditelnost scény bez stínových těles za pomoci paměti hloubky a přitom výsledek v paměti zachováme. Každému pixelu přiřadíme čítač, jehož počáteční hodnota bude rovna počtu stínových těles, uvnitř kterých se nachází kamera. Nyní stačí provést hloubkový test polygonů stínových těles vůči hloubce tohoto pixelu. Je-li test pro daný polygon úspěšný, tj. polygon stínového tělesa se nachází před všemi objekty ve scéně, pak v případě přivrácené stínové plochy čítač inkrementujeme a v případě odvrácené stínové plochy jej dekrementujeme. Pokud po zpracování všech ploch stínových těles zůstane v čítači nenulová hodnota, pak pixel leží ve stínu. Poznamenejme, že iniciální určení počtu stínových těles, v nichž se nachází kamera, není triviální. Metoda je proto vhodná pro scény, v nichž je kamera umístěna mimo scénu.

Popsaný algoritmus implementoval Heidmann [Heid91] s pomocí šablony. V anglické literatuře se metoda označuje jako *depth-pass*. Při její konkrétní realizaci však musíme brát v úvahu, že některé stěny stínové tělesa mohou být ořezány přední nebo zadní rovinou pohledového tělesa. Ve stínovém tělese pak mohou vniknou trhliny, které budou mít za následek chybné určení stínů (obrázek 12.7 vpravo). Kritické je zejména ořezání přední rovinou. Řešení spočívá v přidání dodatečných ploch umístěných do přední roviny pohledového objemu tak, aby stínové těleso zůstalo uvnitř pohledového objemu uzavřené [Kilg01].

Problémům s přední rovinou pohledového tělesa se lze vyhnout, pokud obrátíme smysl algoritmu *depth-pass*, tj. pokud budeme měnit hodnotu čítače (na počátku nastaveného na nulu) jen v případě, že hloubkový test selže, tj. stěna stínového tělesa bude za testovaným pixelem. Budeme tak vlastně počítat navštívená a opuštěná stínová tělesa paprskem, který směřuje z nekonečné vzdálenosti k zobrazovanému (nejbližšímu) povrchu – tedy přesně opačným směrem, než v dosavadním algoritmu. Tento nový algoritmus, označovaný jako *depth-fail*, se do konfliktů s přední ořezávací rovinou nedostává a je navíc nezávislý na počáteční poloze kamery. Oproti metodě *depth-pass* je tentokrát potřeba hlídat ořezání zadní rovinou pohledového objemu a nepřipustit, aby ořezáním vznikly „otvory“ do stínového tělesa. Dalším požadavkem je uzavření stínového objemu (tělesa). K bočním stěnám, které určují stínové těleso, musíme přidat přední a zadní uzávěr (viz obrázek 12.6). Přední uzávěr je tvořen (ke světlu) přivrácenými plochami stínícího objektu, zatímco zadní uzávěr získáme projekcí odvrácených ploch do



Obrázek 12.7: Princip *depth-pass* algoritmu a jeho selhání při ořezání stínového tělesa přední rovinou pohledového tělesa (vpravo, oba dolní testovací paprsky)

vzdálenosti, o kterou jsme prodloužili boční stěny stínového tělesa. Problému s ořezáním zadní rovinou pohledového tělesa se lze vyhnout umístěním této roviny do nekonečné vzdálenosti od kamery [Kilg02]. Pokud zajistíme, že stínové těleso bude vždy uzavřené a nebude ořezáno zadní rovinou, pak společně s algoritmem *depth-fail* získáme univerzální metodu označovanou jako *robust stencil volumes*.

Obě metody, *depth-pass* i *depth-fail* se zpravidla implementují současně a teprve během zobrazování se rozhoduje o tom, zda je možné použít jednodušší a rychlejší *depth-pass* nebo univerzální a o něco složitější *depth-fail*. S metodami stínových těles je navíc spojena řada optimalizačních technik, blíže např. v [Ever03].

Myšlenku stínového objemu lze též použít pro generování měkkých stínů. Princip jednoho z možných rozšíření tohoto algoritmu [Nish85] tkví v tom, že se zkonstruují stínová tělesa pro rohy plošných světelných zdrojů tak, jako by v nich byly umístěny bodové zdroje. Výsledkem je tolik stínových těles, kolik je rohů plošného zdroje. Za kvalitnější obrázek pochopitelně platíme zpomalením výpočtu.

12.3 Stínová paměť hloubky

Posledním reprezentantem metod pro nalezení stínů je algoritmus, který se jmenuje *stínová paměť hloubky* (*shadow depth map*) [Will78]. Tato metoda, na rozdíl od předchozích dvou, pracuje v obrazovém prostoru. Z tohoto důvodu tento algoritmus trpí všemi nedostatky, které se při použití diskrétního obrazu objevují. Projevuje se při něm aliasing, není adaptivní, je nepřesný. Zásadní výhodou tohoto algoritmu je jednak velmi vysoká rychlost, jednak skutečnost, že pracuje s libovolnou reprezentací scény, tedy nejen ploškovou. Algoritmus je vhodný pro

implementaci v grafických akcelerátorech. Navíc lze k jeho realizaci využít prostředků pro mapování textur [Sega92]. Pracuje však pouze s bodovými zdroji světla.

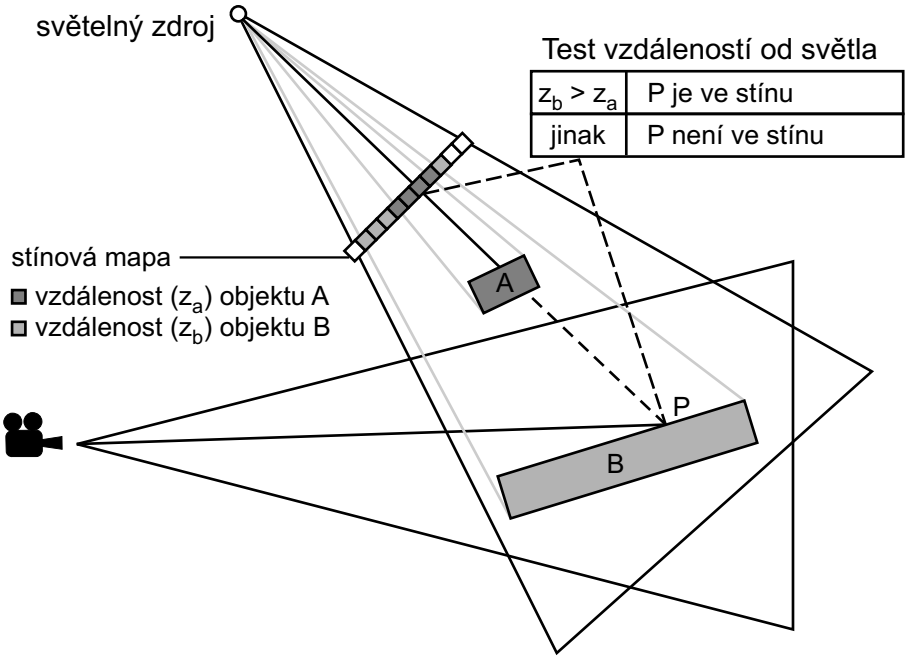
Algoritmus pracuje ve dvou krocích. V prvním kroku se zobrazí scéna z pohledu světla s řešením viditelnosti pomocí paměti hloubky (viz odstavec 11.2.1), jejíž obsah se uloží do tzv. *hloubkové mapy* (*depth map*). V této mapě pixely nenesou barevnou informaci, ale jejich hodnota reprezentuje vzdálenosti od kamery, v tomto případě od světelného zdroje. Ve druhém kroku se scéna zobrazuje z pohledu pozorovatele s řešením viditelnosti algoritmem Z–buffer. Pixely hloubkové mapy tohoto obrazu se transformují do pohledu ze světla a takto přepočtená vzdálenost od světla se porovná se souřadnicí z v paměti hloubky (viz obrázek 12.8 a algoritmus 12.1). Leží-li bod získaný z pohledu kamery po transformaci dále, nežli při pohledu ze světla, je ve stínu a jeho intenzita se sníží.

1. Zobraz scénu z pohledu světelného zdroje L_i; hodnoty z paměti hloubky (z–buffer) ulož do hloubkové mapy H_i.
2. Zobraz scénu z pohledu kamery pomocí paměti hloubky.
3. Pro všechny pixely $[u, v]$ (s hloubkou w) zobrazené scény dělej:
   (a) Převeď bod $[u, v, w]$ do soustavy souřadnic zdroje světla L_i a získej tak jeho nové souřadnice $[x, y, z]$.
   (b) $A = H_i[x, y]$.
   (c) $B = z$.
   (d) Pokud $(A < B)$, pak je pixel $[u, v]$ ve stínu, jinak je zdrojem L_i osvětlen.

Algoritmus 12.1: Algoritmus stínové paměti hloubky

Je-li zdrojů světla více, vytvoří se v 1. kroku algoritmu 12.1 hloubková mapa pro každý zdroj. Kroky (3a) až (3d) v algoritmu se pak opakují pro každou uloženou hloubkovou mapu. Vzhledem k tomu, že stíny jsou pohledově nezávislé, lze jednou spočítané hloubkové mapy využívat opakovaně, dokud se nezmění umístění světelného zdroje nebo poloha objektů ve scéně.

Nevýhodou algoritmu je, že kvalita vytvářených stínů závisí na rozlišení hloubkové mapy a také na přesnosti Z–bufferu. Citlivost algoritmu na vznik aliasingu je způsobena použitím diskrétní hloubkové mapy a přístupem do ní. Každý zobrazovaný pixel totiž reprezentuje určitou plochu z povrchu těles, jejíž projekce do hloubkové mapy může pokrýt hned několik jejích pixelů. Jedná se vlastně o stejný problém, se kterým se setkáváme při mapování textur, a jeho řešení je proto velmi podobné [Reev87]. Při porovnání vzdáleností v hlavním testu



Obrázek 12.8: Princip stínové paměti hloubky

algoritmu se neuplatní jen jedna položka z hloubkové mapy, ale i několik sousedních (například osm nejbližších). Vybrané položky se však nemohou zprůměrovat v jedinou zástupnou hodnotu jako v případě filtrování textur, ale pro každou z nich se musí provést samostatné srovnání s transformovanou souřadnicí z. Teprve výsledky srovnání se zprůměrují a určí konečnou intenzitu pixelu. Popsaným způsobem se nejen odstraňuje aliasing, ale původně ostré hrany stínů se rozmažou a vznikne iluze měkkých stínů, byť nereálných.

Aliasing lze výrazně snížit i metodou zvanou *perspective shadow map* [Stam02], jejíž podstatou je vytvářet stínové mapy v normalizovaném souřadnicovém systému, tedy až po perspektivním promítání. Použitím perspektivního promítání se zvýší rozlišení u bližších objektů a naopak sníží u těch, které se nacházejí ve větší vzdálenosti od středu promítání. Nejlepších výsledků dosahuje metoda pro směrové světelné zdroje, jejichž paprsky jsou rovnoběžné s průmětnou nebo pro bodové zdroje umístěné v rovině kamery (příkladem může být čelní svítilna).

Dalším problémem stínových map je tzv. *self-shadow aliasing*, při kterém polygony mohou být v důsledku numerických nepřesností polygony chybně vyhodnoceny tak, jako kdyby vrhaly stín samy na sebe. Hodnoty uložené v hloubkové mapě a transformované hodnoty z [vyhodnocené na řádku (3a) algoritmu 12.1] nemusejí vyjít totožné, i když je určujeme pro tentýž bod

ve scéně. Je-li hodnota uložená v hloubkové mapě nepatrně menší, dojde k chybnému vykreslení stínu. Nejjednodušší způsob, jak se nežádoucímu stínění vyhnout, spočívá v nepatrném přiblížení souřadnice z na řádku (3c) algoritmu 12.1. Variantou je i mírné oddálení hloubkové mapy v kroku (1) algoritmu. Tím se zajistí, že transformované vzdálenosti všech ze světla viditelných ploch budou vždy menší, než vzdálenosti uložené v hloubkové mapě, a plochy tak budou osvětleny. Velikost posunutí však není jednoznačná. Bude-li příliš velká, pak mohou být osvětleny i objekty, které ve skutečnosti mají ležet ve stínu. Lepší řešení navrhl Woo [Woo92]. Do hloubkové mapy se místo vzdálenosti (jediného) nejbližšího bodu ukládá průměr ze vzdáleností dvou nejbližších bodů promítnutých do dané pozice v hloubkové mapě. Tím se zajistí, že nejbližší povrch bude vždy osvětlen, zatímco všechny objekty za ním budou ležet ve stínu. Pokud v daném směru leží jen jeden povrch, pak se místo průměrné vzdálenosti uloží do hloubkové mapy dostatečně velká hodnota, aby tento povrch nemohl nikdy ležet ve stínu.

Problém *self-shadow aliasingu* lze řešit modifikací algoritmu stínové paměti navrženou Hourcadem [Hour85]. Někdy se tato technika označuje jako *priority buffer shadow*. Princip této metody je v podstatě stejný, avšak místo se vzdáleností od kamery se pracuje s identifikátory objektů. V první fázi algoritmu se každému objektu ve scéně přiřadí jednoznačná hodnota (identifikátor) v závislosti na jeho vzdálenosti od světelného zdroje. Čím blíže je objekt ke světlu, tím nižší hodnotu má jeho identifikátor. Největší hodnota je přiřazena pozadí. Sestavená stínová mapa pak místo vzdáleností obsahuje identifikátory nejbližších objektů. V části (3d) algoritmu 12.1 se porovnávají pouze identifikátory. Pokud je identifikátor nejbližšího objektu (pro daný pixel) větší než hodnota na odpovídajícím místě v hloubkové mapě, pak pixel leží ve stínu, v opačném případě je osvětlen. Nevýhodou metody je skutečnost, že přiřazením identifikátorů objektům ztratí objekty možnost vrhat vlastní stíny samy na sebe. Ty lze touto metodou vrhat jen v případě, že se všechny složitější (nekonvexní) objekty nejprve rozdělí na konvexní části, kterým se přiřadí vlastní identifikátory.

Technika stínových map je vhodná zejména pro směrové světelné zdroje, protože hloubková mapa obsáhne jen část scény vymezené pohledovým tělesem. Pokud však světelný zdroj ovlivňuje větší část scény, než je možno postihnout jednou hloubkovou mapou, nezbývá než vytvořit map více. Nejhorší případ nastane při použití všesměrového zdroje. Tehdy je třeba vytvořit až šest hloubkových map. Každá z nich odpovídá pohledu skrz jednu stěnu pomyslné krychle obklopující světelný zdroj. Vytvořit šest světelných map však znamená šestkrát vykreslit scénu z různých pohledů. Způsob, jak snížit počet stínových map (pro všesměrové zdroje na pouhé dvě) a zvýšit tak rychlost zobrazování scény, je založený na technikách mapování prostředí, blíže např. v [Brab02].

Kapitola 13

Textury

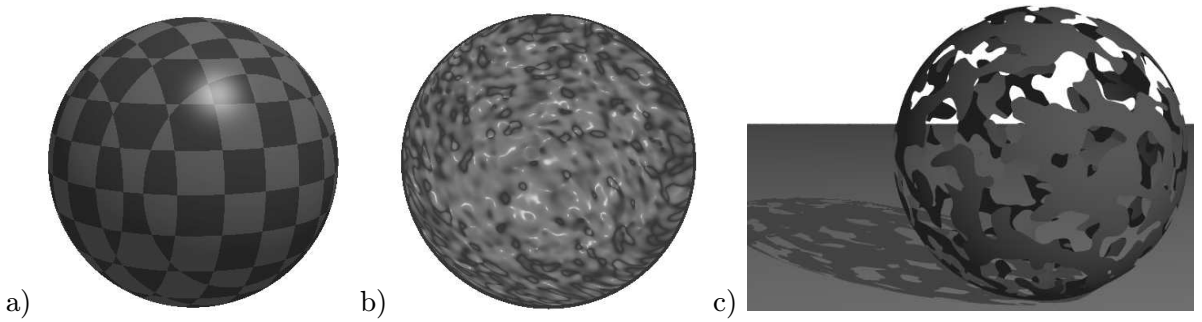
Textura je popisem vlastností povrchu a je důležitá pro vnímání struktury, barvy a kvality objektu. Textura je vzorek, který může být buď pravidelný, či nepravidelný. Prvek textury se jmenuje *texel*. Textura je úzce spjata s materiálem, který by měl povrch jednoznačně popisovat. Z historických důvodů, zejména však pro zjednodušení mnoha operací, se materiál a textura oddělují a aplikují se ve dvou krocích.

Aplikace textury vede k podstatnému zvýšení vizuální kvality objektu za cenu relativně malých nákladů, a proto je textura intenzivně používaná zejména v časově kritických aplikacích.

Často je efektivnější použít jednoduchou geometrii a složité textury, nežli definovat složité geometrické detaily [Moll02]. Výsledek, zejména pro objekty, které jsou zobrazeny jen krátce či z velké vzdálenosti, je od složitých objektů k nerozeznání. Typickým příkladem jsou billboardy, tento přístup ale nachází použití například při modelování příšer v počítačových hrách.

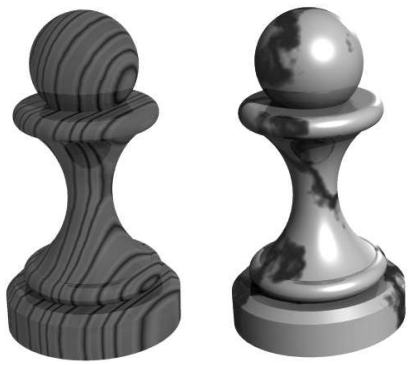
Heckbert [Heck86] navrhl rozdělení textur na základě toho, jakou vlastnost povrchu popisují. S postupem času bylo zapotřebí toto rozdělení rozšířit.

- *Barva povrchu* je určena koeficientem difúzního odrazu. Mapování difúzní složky materiálu je nejčastěji používaným způsobem aplikace textury (obr. 13.1a).
- *Odraz světla* se může měnit s místem povrchu a simuluje se jako změna zrcadlové složky materiálu. Projevem této vlastnosti je odrážející se okolí objektu na jeho povrchu.
- *Změna normálového vektoru* opticky mění tvar povrchu, aniž by měnila geometrii objektu. Výsledkem je povrch, který vypadá zprohýbaný, či jinak geometricky změněný. Typickým reprezentantem této techniky je hrbolatá textura (*bump mapping*) na obrázku 13.1b.
- Textura může určovat *průhlednost* povrchu a její aplikací se docílí dojem změny geometrie povrchu tak, jak je patrné na obrázku 13.1c.
- *Hypertextura* určuje optické vlastnosti nad povrchem objektu a hodí se pro modelování vlasů, ohně, trávy atp. (obrázek 13.16).



Obrázek 13.1: Příklady textur: a) difúzní odraz b) hrbolatá textura (*bump*) c) průhlednost

Jiná klasifikace textur je možná na základě jejich rozměru. Rozlišujeme tedy *jedno*, *dvou*, *troj* a *čtyřrozměrné* textury. Jednorozměrné textury se používají pro definici opakujících se podélných vzorků, například pruhů („přechod pro chodce“), ale také se mohou aplikovat na generování izokřivek na površích a mapách. Další aplikací jednorozměrných textur je vzorek pro generování přerušovaných křivek a čar. Poloprůhledná barevná textura se může použít pro animaci deště. Dvourozměrné textury jsou mapovány na povrch tělesa. Trojrozměrné textury definují hodnotu textury v prostoru a proto se jim také říká *objemové textury* (*solid texture*) a používají se pro simulaci objektů, které vypadají jako vyřezány z jediného bloku materiálu (obrázek 13.2). Čtyřrozměrné textury se mohou použít pro animaci trojrozměrných textur.



Obrázek 13.2: Procedurální trojrozměrné textury dřevo a mramor

Hodnota jednorozměrného a dvourozměrného texelu se obyčejně získá mapováním z tabulky pixelů z obrazu (při reprezentaci textury tabulkou) či voláním procedury. Podobně hodnota texelu se ve trojrozměrné textuře definované tabulkou získá mapováním odpovídajících voxelů či voláním procedury.

Poslední klasifikace textur je možná podle způsobu jejich reprezentace. Textura může být buďto uložena v jednorozměrné, dvourozměrné či trojrozměrné tabulce například jako obrazový soubor či pole voxelů. Další možností je definici textury pomocí procedury. Tomuto typu textur se říká *procedurální textury* a mají blízko k procedurálním modelům z kapitoly 8.

Aplikace samostatné textury není příliš častým případem. Obyčejně se aplikuje více textur, resp. více texturovacích technik najednou (*multitexturing*). Tyto kroky se mohou aplikovat v jednom, obyčejně však ve více krocích (*multipass texturing*). Tak lze docílit poměrně komplikovaných povrchů složených z více materiálů. Vícenásobná

aplikace textur nemusí být prováděna za pomoci software, ale stává se doménou grafických procesorů. Ty v sobě obyčejně zahrnují několik *texturovacích jednotek* (*texturing unit*), které mohou být programovány uživatelem [Moll02, Neid03]. Například ve hře Quake III je, pro docílení co nejvyšší kvality obrazu, na nejvýkonnějších počítačích každý obraz generován v deseti průchodech.

Aplikace textury se skládá ze dvou po sobě jdoucích kroků. V prvním z nich se textura definuje. Ve druhém se určí, na jaký objekt a kam přesně se položí. Druhému kroku se říká mapování textury (*texture mapping*) a je spojeno s filtrací (viz dále). Definici textury jsme uvedli výše, podívejme se nyní na její mapování.

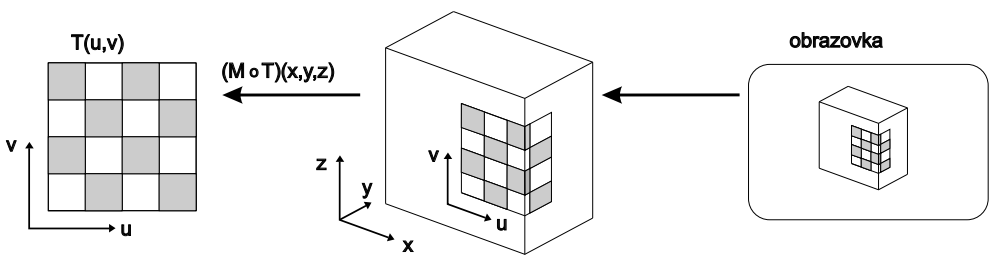
13.1 Mapování textur

Proces nanášení dvourozměrné textury si přiblížíme následujícím příkladem. Máme k dispozici láhev (povrch tělesa), na kterou chceme umístit nálepku (mapovanou texturu). Nalepit obrázek na válcový povrch je jednoduché. Potíže nebude činit ani hranatá láhev. Pokud ale bude mít obal složitější tvar, bude i proces mapování složitější.

Proces nanášení textury na povrch těles se nazývá *mapování textury* a je předurčen třemi důležitými faktory. *Definicí textury*, tj. kolika-rozměrná textura je a zda se jedná o definici tabulkou či o texturu procedurální. Dále tvarem tělesa, na který je textura nanášena a nakonec mapovanou veličinou. V následujících částech ukážeme nejprve nejjednodušší mapování rovinné textury, od kterého přejdeme ke složitějším způsobům.

Rovinnou texturu můžeme definovat jako funkci $T(u, v)$ přiřazující bodům $[u, v]$ v rovině hodnoty mapované veličiny, například barvu:

$$T : D_T \rightarrow H_T, \quad D_T \subset R^2$$



Obrázek 13.3: Inverzní mapování rovinné textury. Pro každý pixel obrazu se určí odpovídající objekt. Inverzní transformací se získají souřadnice bodu v objektovém prostoru a z nich se určí souřadnice textury.

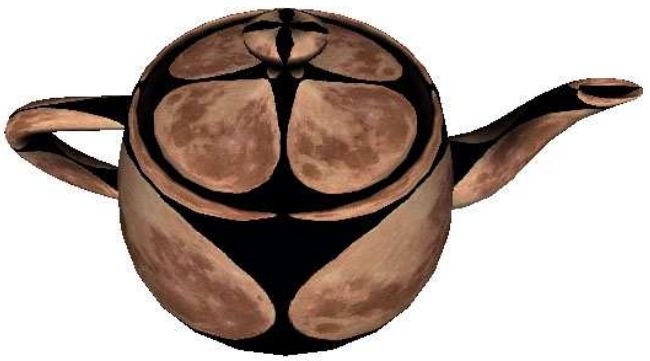
Abychom mohli texturu aplikovat, musíme definovat funkci $M(x, y, z)$, přiřazující každému bodu na povrchu tělesa bod z definičního oboru textury T. Funkce M je označována jako *inverzní mapování* (*inverse mapping*) a je využívána v procesu nanášení textury:

$$M : D_M \to D_T,$$

kde $D_M \subset R^3$ jsou body na povrchu tělesa.

Při vykreslování tělesa známe souřadnice $[x, y, z]$ bodů na povrchu a hledáme k nim informace o mapované (zobrazované) textuře. Toto zobrazení získáme složením funkcí $M \circ T$.

Celý proces je znázorněn na obr. 13.3. Použití textury si lze představit jako polepení tělesa papírem. Funkce T odpovídá tomu, co je na papíru nakresleno, funkce M způsobu nalepení.



Obrázek 13.4: Zkreslení textury na pokličce čajníku

Volba funkce M musí odpovídat tvaru tělesa, na které je textura nanášena. Tvoří-li povrch tělesa jediná analytická plocha, je možno M volit jako inverzní funkci (jestliže existuje) k parametrizaci plochy. U ploch rozvinutelných do roviny lze najít takovou funkci M, u které nedojde ke zkreslení textury. Příkladem je válcová plocha.

U zborcených ploch (takových, které nelze bez deformace rozvinout do roviny) dojde ke *zkreslení textury*. To je zřetelné například na kouli v okolí obou „pólů“, podobně jako na vrcholu čajníku, jak je vidět na obr. 13.4. Pokud je těleso ohraničeno více plochami, bývá problémem dosáhnout správného *navazování textury*.

13.1.1 Inverzní mapování válcové a kulové plochy

Jako příklad určení inverzní mapovací funkce uvedeme její odvození pro válcovou a kulovou plochu. Při určení inverzní mapovací funkce M umístíme válec tak, aby jeho osa byla rovnoběžná s osou z a střed dolní podstavy ležel v počátku soustavy souřadnic (viz obr. 13.5 vlevo). Mapovat budeme pouze na válcovou plochu tak, aby textura definovaná v oboru $\langle 0, 1\rangle \times \langle 0, 1\rangle$ pokryla celý plášť válce. Proto do výpočtu zahrneme poloměr r a výšku h válce. Vyjdeme z popisu válcové plochy v cylindrických souřadnicích, $[x, y] = [r\cos\alpha, r\sin\alpha]$, ze kterého odvodíme

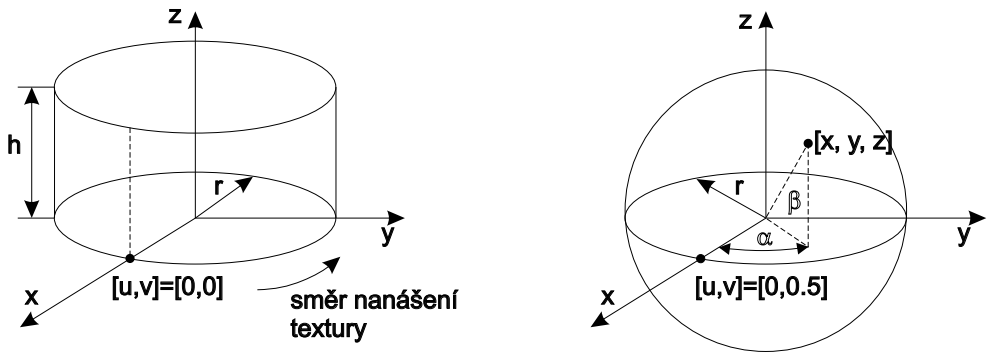
inverzní mapovací funkci $M(x, y, z)$:

$$u = \begin{cases} \frac{1}{2\pi} \arccos \frac{x}{r}, & \text{pro } y \leq 0 \\ 1- \frac{1}{2\pi} \arccos \frac{x}{r}, & \text{pro } y > 0 \end{cases} \qquad (13.1)$$

$$v = \frac{z}{h},$$

kde funkce *arccos* vrací hodnoty z intervalu $\langle 0, \pi \rangle$.

V uvedeném příkladu je čtvercová textura mapována na plášť válce, který má po rozvinutí do roviny tvar obdélníku. Tím dochází k neuniformní změně měřítka textury ve vodorovné a svislé ose. Vynásobením souřadnic u a v vhodnými měřítky lze nejen případná zkreslení napravit, ale s využitím funkce *modulo* je možno texturu nanášet opakovaně.



Obrázek 13.5: Inverzní mapování válcové plochy (vlevo) a kulové plochy (vpravo)

Při inverzním mapování kulové plochy postupujeme analogicky. Kouli o poloměru r umístíme tak, aby její střed ležel v počátku (obr. 13.5 vpravo). Body ležící na kulové ploše můžeme vyjádřit pomocí sférických souřadnic $[x, y, z] = [r \cos\alpha \cos\beta, r \sin\alpha \cos\beta, r \sin\beta]$ kde $\alpha \in \langle 0, 2\pi \rangle$ a $\beta \in \langle -\pi/2, \pi/2 \rangle$. Inverzní mapovací funkce M má pak tvar:

$$u = \begin{cases} \frac{1}{2\pi} \arccos \frac{x}{\sqrt{x^2+y^2}}, & \text{pro } y \leq 0 \\ 1- \frac{1}{2\pi} \arccos \frac{x}{\sqrt{x^2+y^2}}, & \text{pro } y > 0 \end{cases} \qquad (13.2)$$

$$v = 0.5 + \frac{1}{\pi} \arcsin \frac{z}{r},$$

a funkce *arccos* vrací hodnoty z intervalu $\langle 0, \pi \rangle$ a *arcsin* hodnoty z intervalu $\langle -\pi/2, \pi/2 \rangle$.

Textura se směrem k pólům $[0, 0, -r], [0, 0, r]$ deformuje a v nich dokonce degeneruje na jediný bod. Tomuto případu odpovídá dělení nulou v odmocnině ve vzorci (13.2).

Dvojrozměrná textura může být větší, nežli je mapovaný povrch, potom nedojde k zobrazení některých jejích částí. Je-li textura menší, buď se ukončí (*texture clamping*), nebo se v jednom či, obou směrech u a v opakuje (*texture repeat*) jak ukazuje obrázek 13.6. V některých systémech, například OpenGL, může mít textura definovaný okraj a poté se může opakovat hodnota okraje po zbytku plochy (*clamp to edge*).

Obrázek 13.6: Ukončení (vlevo) a opakování textury ve směru v a ve směrech u a v mapované na natočený polygon.

13.1.2 Mapování prostorové textury

Zatímco použití rovinné textury můžeme přirovnat k polepení tělesa pomalovaným papírem, prostorová textura odpovídá vyřezání tělesa z jednoho bloku materiálu, který má vnitřní strukturu popsanou texturou. Tak je možno modelovat například tělesa z mramoru, ze dřeva atp. (viz obr. 13.2).

Textura může být definována v nějaké omezené části trojrozměrného prostoru, například jako jednotková krychle. Texturovací funkce pak musí mapovat tento prostor do prostoru tělesa. Proces texturování spočívá v aplikaci souřadnic $[x, y, z]$ texturovaného bodu k nalezení odpovídajícího bodu $[u, v, w]$ v textuře.

Nevýhodou prostorové textury jsou její paměťové nároky. Pokud ji reprezentujeme tabulkou, má obyčejně velký rozsah a přístup k hodnotám je pomalý. Proto se hodnoty prostorových textur obyčejně zaznamenávají jen do nepříliš rozsáhlých mřížek a neznámé hodnoty se vypočítávají trilineární interpolací (viz 22.6.4). Běžnější a častěji používaná reprezentace prostorových textur je pomocí procedury, jak uvidíme v části 13.2.

13.1.3 Mapování prostředí

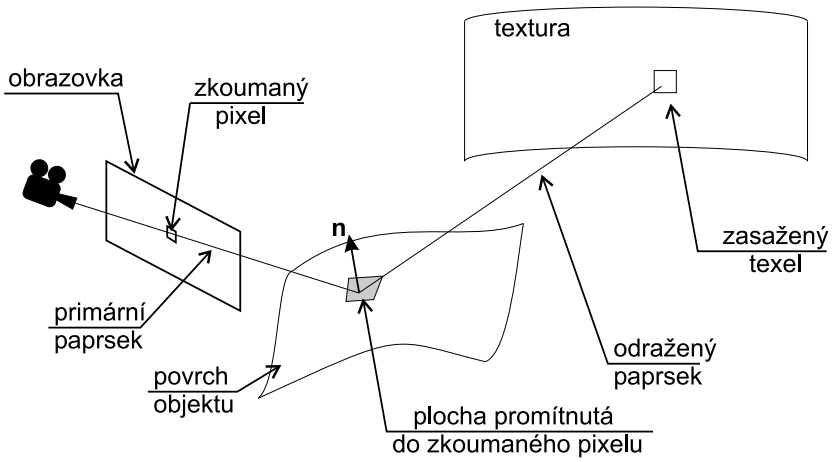
Mapování prostředí (*environment mapping*) se též nazývá *pohledově závislé mapování*. Modeluje zrcadlení textury na povrchu texturovaného objektu, čímž se aproximuje odraz okolního prostředí objektu na jeho povrchu. Protože takovýto odraz je typický pro zrcadlové materiály, je tato technika také nazývána *chromové mapování* (*chrome mapping*) a její příklad je na obrázku 13.7.

Odraz od povrchu tělesa v určitém bodě závisí na poloze tělesa, na poloze pozorovatele a na okolním prostředí. Pokud se bude některá z těchto složek pohybovat, dojde i ke změně textury, textura se bude posouvat po povrchu objektu, a proto je tato textura tzv. pohledově závislá. Reálný odraz skutečného prostředí by vypadal jinak, ale lidské oko si nevšimne, že se jedná o aproximaci prostředí jedinou texturou. Tato technika je efektní, a proto se často používá tam, kde je zapotřebí rychle počítat odrazy v rozsáhlých scénách. Často se s ní setkáváme ve filmech, byla například použita v Terminátoru II, setkáme se s ní v televizních reklamních šotech typu létající logo atp.

Obrázek 13.7: Příklad mapování prostředí

Princip mapování prostředí je znázorněn na obrázku 13.8. Jeho podstatou je uzavření texturovaného objektu do jiného objektu a aplikace textury na jeho vnitřek. Tento krok se simuluje aplikací mapovací funkce. Tato funkce mapuje souřadnice objektu do obdélníkové textury tak, jako by byla aplikována na obklopující objekt. Mapování znamená nalezení indexu texelu z odpovídající textury. Z polohy pozorovatele a pixelu na obrazovce se určí tzv. primární paprsek (*primary ray*). Pokud primární paprsek protne texturovaný objekt v nějakém bodě, určí se z tohoto průsečíku, primárního paprsku a normálového vektoru odražený paprsek. Protože texturovaný objekt je uzavřen do obklopujícího objektu, odražený paprsek musí někde zasáhnout jeho vnitřek. Z bodu, který je zasažen, se určí příslušný texel a ten je pak výslednou barvou pixelu na obrazovce.

Tato metoda se nejčastěji používá v časově kritických aplikacích, protože tam, kde je na to dostatek času, lze vypočítat odrazy ve scéně přesně. Pokud nezáleží na kvalitě výsledku lze použít jako obklopující těleso kouli, kdy zanedbáváme problémy zkreslení textury u pólů, nebo krychli, kdy přehlížíme problémy spojení textury ve švech.



Obrázek 13.8: Princip mapování prostředí

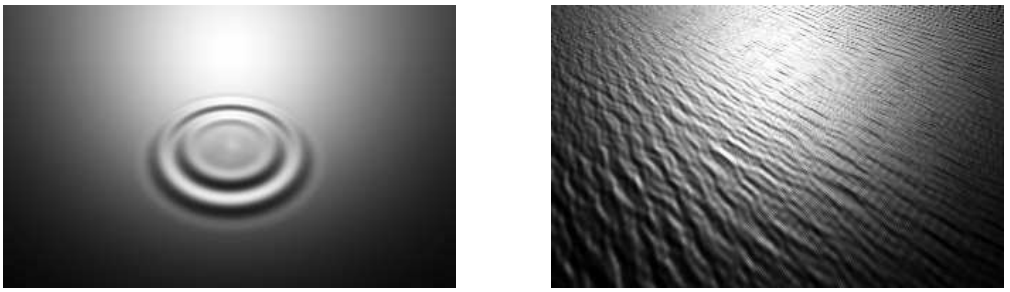
Původní metoda [Blin78] používá mapování textury na kouli a trpí problémy s degenerací textury na pólech. Metoda publikovaná Greenem [Gree86] používá mapování na krychli a je výhodná zejména pro snadné vytvoření takové textury a pro její rychlou aplikaci. Nevýhodou této metody je problém se švy u hran krychle, které se projeví na texturovaném objektu jako vizuální nespojitosti a hrany. Williams [Will78] publikoval metodu využívající projekci textury na kouli, která se, spolu s metodou mapování na krychli, stala standardní technikou mapování prostředí. Výhodou poslední zmíněné metody je snadné získání textury, kterou lze získat například fotografií odrazu ve stříbrné (vánoční) kouli. Takto získaná fotografie se také používá pro získání optických vlastností prostředí a říká se jí *light probe*. Detaily tvorby textury pro takováto prostředí jsou k nalezení [Moll02].

13.1.4 Hrbolaté textury

Metoda hrbolatých textur (*bump texture*) byla publikována Blinnem v roce 1978 [Blin78]. Způsobuje optický dojem hrbolatého povrchu, aniž by docházelo ke změnám geometrie tělesa. To je patrné na okraji koule z obrázku 13.1b, která má hladký obrys, zatímco v jiných místech vypadá jako silně zvrásněná.

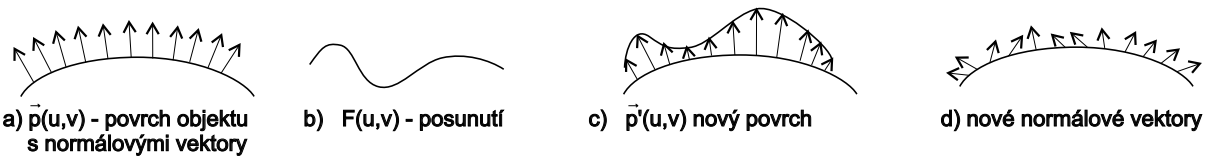
Hrbolaté textury vycházejí z osvětlovacího modelu, který využívá normálu k povrchu v místě, ve kterém jsou světelné poměry vyšetřovány. Normálový vektor je při mapování textury pozměněn tak, aby změnil směr odrazu světla způsobem, který odpovídá lokálnímu zvrásnění hrbolatého tělesa.

Předpokládejme bod na povrchu tělesa určen vektorovou funkcí $\vec{p}(u, v)$ (viz obrázek 13.10).



Obrázek 13.9: Obrázky získané metodou hrbolatých textur

Normálový vektor k původnímu povrchu označíme $\vec{n}(u,v)$ a budeme dále předpokládat, že jeho velikost je vždy rovna jedné. Funkce určující zvrásnění povrchu se označuje $F(u,v)$ a udává vzdálenost povrchu, který určuje texturu od původního hladkého povrchu. V dalším textu nebudeme pro zjednodušení zapisovat parametry (u,v).



Obrázek 13.10: Princip získání hrbolaté textury

Původní bod nejprve posuneme ve směru normálového vektoru o vzdálenost F a získáme bod $\vec{p}'$ na novém povrchu

$$\vec{p}' = \vec{p} + F\vec{n}.$$

V dalším kroku určíme nový normálový vektor $\vec{n}'(u,v)$. Nejprve vyjádříme tečné vektory k novému povrchu. Ty získáme parciální derivací nového povrchu v tečném směru u a v a budeme je označovat $\vec{p}'_u$ a $\vec{p}'_v$. Lze ukázat [Blin78] že pokud je změna normálového vektoru malá, lze tyto tečné vektory vyjádřit:

$$\begin{aligned} \vec{p}'_u &= \vec{p}_u + F_u\vec{n} \\ \vec{p}'_v &= \vec{p}_v + F_v\vec{n}, \end{aligned} \tag{13.3}$$

kde $\vec{p}_u$ je tečný vektor k původnímu povrchu a F_u je derivace mapy posunutí. Nový normálový vektor získáme jako vektorový součin tečných vektorů k novému povrchu:

$$\vec{n}' = \vec{p}'_u \times \vec{p}'_v = \vec{n} + \vec{d},$$

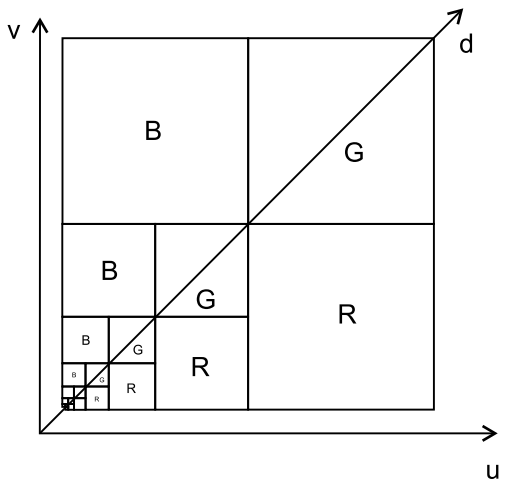
kde

$$\vec{d} = F_u[\vec{n} \times \vec{p}_u] - F_v[\vec{n} \times \vec{p}_v]. \tag{13.4}$$

Aplikace hrbolaté textury tedy spočívá v nalezení tečných vektorů k funkci určující posunutí (hrbolatost) a k původnímu povrchu a výpočtu normálového vektoru podle rovnice (13.4). Aplikovaná textura může mít libovolnou podobu. Pokud se pro její definici tabulkou použije dvojrozměrný obraz lze provést určení tečných vektorů jako detekci hran v zadaném směru. Procedurální textury mohou k vyjádření tečných vektorů využít výpočtu hodnoty funkce v sousedních bodech. Příklad aplikace takové textury je na obrázku 13.1.

Při manipulaci s normálovým vektorem musíme mít na zřeteli, že silně zvrásněný povrch by stínil sám sebe. Vzhledem k tomu, že tato technika zmíněný jev simulovat nedokáže, je nutné modifikovat směr normály „umírněně“.

13.1.5 MIP-mapping



Obrázek 13.11: Mip-mapping

Během mapování se textura musí filtrovat. Textura je obyčejně definována jako nespojitá a je aplikována v pixelech výsledného obrazu, do kterých se promítne povrch texturovaného objektu. Protože ne všechny body jsou definovány, musí se provádět interpolace hodnot textury mezi definovanými body, pro které se používají aplikace převzorkování algoritmy uvedenými v kapitole 2.

Jedním ze základních problémů při mapování textur zejména v časově kritických aplikacích je nutnost častého zvětšování a zmenšování textury, podle toho, zda je texturovaný objekt blízko či daleko. Zejména při aplikaci více textur může tato operace spotřebovávat podstatnou část výpočetního výkonu, resp. brzdit proudové zpracování grafických operací. Jednou z možností je předem spočítat a uložit texturu v různých velikostech (různém rozlišení). Tato technika se nazývá *mip-mapping* (mip – *multum in parvo*[1]). Při mapování se pak vybere nejvhodnější textura, případně se zmenší či zvětší, avšak již ne z jediného obrazu, ale z nejbližšího možného. Princip mip-mappingu tedy spočívá v tom, že namísto opakovaného zmenšování textury během zobrazování se provede tato

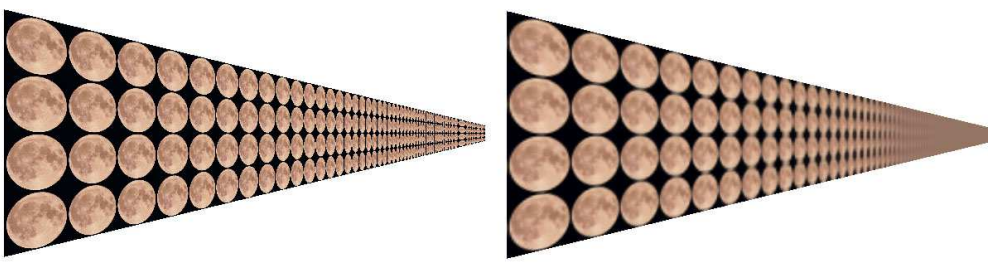
[1] *Lat.* mnohé v malém.

operace jednou před jejím použitím a výsledek se uloží. Za zrychlení výpočtu se samozřejmě platí zvětšením prostoru, který je nutný pro uložení textury.

Textura definovaná trojicí složek RGB o velikosti řekněme 512×512 pixelů je reprezentována v poli o rozměrech 1024×1024 bytů tak, jak naznačuje obrázek 13.11. Jednotlivé barevné roviny R, G a B jsou zapsány ve třech čtvercových oblastech o rozměru 512×512 bytů. Ve zbývajícím rohu je textura reprezentována stejně, pouze v polovičním rozlišení tj. 256×256, v jejím volném rohu je totéž v polovičním rozlišení, atd. Je tedy uloženo celkem devět reprezentací původní textury v různém rozlišení ($512 = 2^9$) a na tuto reprezentaci je možno pohlížet jako na devítistupňovou pyramidu od maximálního rozlišení až po reprezentaci textury jediným pixelem uloženým v trojici RGB bytů. Z předlohy v rozlišení 512×512 pixelů se zmenšeniny získávají filtrací, například průměrováním čtyř pixelů do jednoho. Jediný pixel v levém dolním rohu pak reprezentuje průměrnou hodnotu všech pixelů obrazu.

Bod textury je při mapování adresován třemi souřadnicemi u, v a d. První dvě souřadnice udávají běžné souřadnice určené pro mapování textury, reálné číslo d určuje vertikální souřadnici v pyramidě a odvozuje se ze vzdálenosti od texturovaného objektu. Zatímco textury jsou definovány v diskrétních velikostech rastru, při přibližování k objektu se vzdálenost mění spojitě. V praxi se tedy buď vybere ta vrstva, která je blíže, nebo se vypočítávají pixely pomocí lineární interpolace dvou sousedních vrstev v pyramidě. Odvození vztahu pro výběr příslušné textury z pyramidy ze souřadnice d je k nalezení v [Watt92, strana 143-145].

Tato metoda uložení a mapování textur je dnes standardním vybavením grafických procesorů. Jednou z jejích nevýhod je však výrazné rozmazání obrazu, zejména ve velkých vzdálenostech texturovaného povrchu od pozorovatele jak je patrné na obrázku 13.12.



Obrázek 13.12: Velmi dlouhý objekt zobrazen bez (vlevo) a s technikou mip-mappingu (vpravo).

13.2 Procedurální textury

Jedním z omezení textur reprezentovaných tabulkou je velký prostor, který zabírají. *Procedurální textury* tímto nedostatkem netrpí. Jedná se o programy, jejichž zavoláním získáme hodnotu textury v daném bodě. Procedurálními texturami snadno realizujeme různé opakující se vzory (viz obrázek 13.1a), například texturu cihlové zdi, plotu atp. Zajímavé možnosti nám poskytují především procedurální textury syntetizující *šum*. Z nich jsou v počítačové grafice nejčastěji používané procedurální textury založené na fraktálních šumových funkcích (viz část 8.1.5).

13.2.1 Perlinova šumová funkce

Optimální šumová funkce by měla poskytovat bílý šum, který má frekvenčně neomezené spektrum. Takový šum obsahuje detaily na všech možných úrovních, a proto může být libovolně zvětšován či zmenšován. To je nepraktické, pro jeho výpočet je zapotřebí velký výpočetní výkon, který navíc v mnoha případech není nutný, protože příliš velké detaily nemusí být zapotřebí, například při pohledu z velké vzdálenosti.

Perlin [Perl85] navrhl šumovou funkci, později pojmenovanou Perlinova funkce, kterou lze vypočítat rychle a navíc splňuje následující požadavky:

- Funkce je statisticky invariantní vzhledem k otáčení,
- je statisticky invariantní vzhledem k posunutí,
- je spojitá,
- má omezené frekvenční spektrum a
- je opakovatelná. Zavoláním funkce s parametrem x vrátí vždy stejnou hodnotu y.

Splnění prvních dvou požadavků zaručuje, že se funkce nebude statisticky měnit při posouvání a otáčení. Jinými slovy řečeno, nebude záležet na tom, kde začneme takovou texturu mapovat a jak bude natočená, textura bude statisticky soběpodobná. Splnění třetího požadavku zaručuje, že funkce bude mít určitou maximální frekvenci a teoreticky by mohla produkovat nežádoucí alias. Jak později uvidíme, tato nejvyšší frekvence se dá v algoritmu zvolit podle přesnosti vzorkování a tak lze tomuto jevu snadno zabránit. Především tato vlastnost má za následek rychlý výpočet hodnoty funkce. Funkce je náhodná, tato náhodnost je však řiditelná.

Podstatou Perlinova šumu je generování spojitého šumu, který je však vypočítáván v diskrétní mřížce. Perlinův šum může být definován v libovolné dimenzi. V následujícím textu se přidržíme popisu z [Eber03] a popíšeme trojrozměrný případ. Základem je šumová funkce $noise(float\ x, float\ y, float\ z)$, která pro určité hodnoty $[x, y, z]$ vrací vždy stejné náhodné číslo z intervalu $\langle -1, 1\rangle$. Voláním funkce určitým parametrem, dostaneme jako výsledek vždy stejné číslo, což je splnění páté podmínky z výše uvedeného seznamu. To je důležitá vlastnost,

která způsobuje, že volání funkce s určitou texturovací souřadnicí bude vždy na odpovídající místo mapovat stejnou barvu.

Perlinova funkce při volání s parametrem $noise(2x, 2y, 2z)$ vrací výsledek, který má dvojnásobné frekvence šumu. Později uvidíme, že tato vlastnost se využívá pro skládání šumových funkcí.

Základní myšlenkou Perlinovy šumové funkce je rozdělení prostoru do pravidelné mřížky, jejíž vrcholy budeme označovat $[i, j, k]$. V každém z vrcholů je definována pseudonáhodná trojrozměrná funkce, které se v původním textu říká vlnka (*wavelet*). Vlnka má tzv. poloměr, který určuje její rozsah. Hodnoty vlnky za touto hodnotou jsou nulové. Vlnka zároveň prochází počátkem, to znamená že pro parametry $[i, j, k]$ je její hodnota rovna nule. Při návrhu jejích hodnot se tak nemusíme zabývat hodnotou v počátku, ale definujeme pouze její gradient. Integrál přes obor hodnot vlnky by zároveň měl být nulový.

Postup výpočtu Perlinovy šumové funkce pro bod s reálnými souřadnicemi $[x, y, z]$ se skládá z následujících kroků.

1. Určíme ve které buňce se souřadnice $[x, y, z]$ nachází.
2. Pro každý z osmi sousedních vrcholů se vypočítá tvar vlnky.
3. Hodnoty vlnek odpovídající poloze $[x, y, z]$ se sečtou.

1. *Určení buňky* je prvním krokem algoritmu a provede se snadno. Zaokrouhlením reálných hodnot $[x, y, z]$ na jejich nejbližší dolní celočíselnou hodnotu získáme souřadnice levého dolního rohu krychle (rohu s minimálními souřadnicemi)

$$[i, j, k] = [\lfloor x \rfloor, \lfloor y \rfloor, \lfloor z \rfloor].$$

Ostatních sedm vrcholů krychle má souřadnice :

$$[i + 1, j, k], [i, j + 1, k], \ldots, [i + 1, j + 1, k + 1].$$

2. *Výpočet tvaru vlnky* je druhým krokem algoritmu. Vlnka má nulovou hodnotu v bodě $[i, j, k]$, a proto pro určení jejího tvaru stačí pouze hodnota jejího gradientu pro střed vlnky.

Šum nemá struktury, které by byly rozpoznatelné ve velkých měřítcích, a proto není příliš snadné pozorovat opakující se vzory. Perlin proto navrhuje použít tabulku pseudonáhodných vektorů označenou G o délce 256. Postup jejího generování je uveden v algoritmu 13.1.

Algoritmus generuje pro každý index tři náhodná čísla x, y a z z intervalu $\langle -1, 1 \rangle$. Tyto hodnoty odpovídají náhodnému vzorkování jednotkové krychle. Hodnoty, které jsou mimo rozsah jednotkové koule nejsou použity. Zbývající vhodné hodnoty se promítnou na jednotkovou kouli (znormalizují) a uloží do pole G, jak naznačuje předposlední řádek. Výsledné pole obsahuje 256 náhodných hodnot směrů, rovnoměrně pokrývající jednotkovou kouli.

proveď pro 256 vzorků i

1. vygeneruj tři pseudonáhodná čísla $-1 < x, y, z < 1$.
2. pokud jsou vně jednotkové koule zamítni je a opakuj předchozí krok
3. znormalizuj vektor daný souřadnicemi (x, y, z)
4. ulož (x, y, z) do pole $G[i]$

Algoritmus 13.1: Algoritmus výpočtu pole G

Přístup do pole G se neprovádí přímo, ale pomocí pole P, které obsahuje náhodné permutace indexů. Toto pole délky 256 se vypočítá předem „zamícháním“ indexů jak ukazuje program 13.2

zaplň pole P hodnotami od nuly do 255
pro i od nuly do 255

1. vygeneruj pseudonáhodné číslo $0 \leq k < 256$.
2. zaměň $P[i]$ a $P[k]$

Algoritmus 13.2: Výpočet pole permutací P

Pole náhodných směrů je jednorozměrné, hodnoty $[i, j, k]$, pro které tyto směry hledáme jsou trojrozměrné. K indexaci pole se provede tzv. *přeložení* souřadnic (*fold*). Mějme tři proměnné i, j, k. Hodnota $fold(i, j, k)$ se získá

$$fold(i, j, k) = P[(P[(P[i \ mod \ 256] + j) \ mod \ 256] + k) \ mod \ 256].$$

Tím libovolné trojici indexů $[i, j, k]$ přiřadíme náhodný směr, který splňuje podmínky výše uvedené – neopakuje se (příliš), je jednoznačný a rozložení těchto náhodných směrů v prostoru rovnoměrně vzorkuje jednotkovou kouli. Směr, gradient, v bodě $[i, j, k]$ se poté získá voláním $G[fold(i, j, k)]$.

Určení hodnoty vlnky v bodě $[i, j, k]$ je předposledním krokem výpočtu. Jejím výsledkem je hodnota vlnky v daném bodě změřítkovaná o relativní vzdálenost k bodu $[x, y, z]$. Vynásobením těchto dvou hodnot se získá hodnota, vlnky v daném rohu krychle, tedy příspěvek tohoto vrcholu k celkové hodnotě funkce.

Nejprve vypočítáme relativní vzdálenosti souřadnic $[x, y, z]$ od $[i, j, k]$. Označme tyto souřadnice $[u, v, w]$

$$[u, v, w] = [x, y, z] - [i, j, k], \quad -1 \leq u, v, w < 1.$$

Pokles hodnoty funkce se vzdáleností je označen $drop(t)$ a je dán kubickou funkcí

$$drop(t) = 1 - 3|t|^2 + |t|^3.$$

Celkový úbytek $\Omega(u, v, w)$ v bodě $[u, v, w]$ se určí z relativních příspěvků

$$\Omega(u, v, w) = drop(u) \cdot drop(v) \cdot drop(w).$$

Zbývá poslední krok, kterým je vynásobení relativního celkového úbytku náhodnou funkcí G

$$\Omega(u, v, w) \cdot G(i, j, k).$$

3. *Výsledná hodnota funkce v bodě* $[x, y, z]$ je určena součtem hodnot vlnek z vrcholů krychle.

Algoritmus výpočtu Perlinovy funkce je poměrně složitý, proto pro jeho implementaci přímo od jeho autora odkazujeme do přílohy knihy [Eber03].

Perlinova funkce poskytuje šum libovolné frekvence, který je získán za cenu poměrně nízkého výpočetního výkonu. Náhodné tabulky jsou vypočítány předem a výpočet hodnoty pro konkrétní souřadnice $[x, y, z]$ je proveden podle výše zmíněného algoritmu.

13.2.2 Skládání šumových funkcí

Perlinova šumová funkce je základem algoritmů pro skládání šumových funkcí, podobných fraktálním algoritmům z kapitoly 8. Připomeňme, že zavolání funkce $noise(x, y, z)$ s parametrem $noise(2x, 2y, 2z)$ generuje šum s dvojnásobnou frekvencí. Myšlenka skládání šumových funkcí je založena na následujícím postupu.

Složení šumových funkcí je součet funkcí $noise(x, y, z)$, každá se změněnou amplitudou a frekvencí. Takto získané funkci se v literatuře někdy říká Perlinova funkce, i když se ve skutečnosti jedná o součet Perlinovy funkce se změněnou velikostí a frekvencí. Obecně můžeme tímto způsobem skládat libovolné šumové funkce. Příspěvkům součtu se říká *oktávy*, též „šumové harmonické“ a jejich součet označíme *snoise*

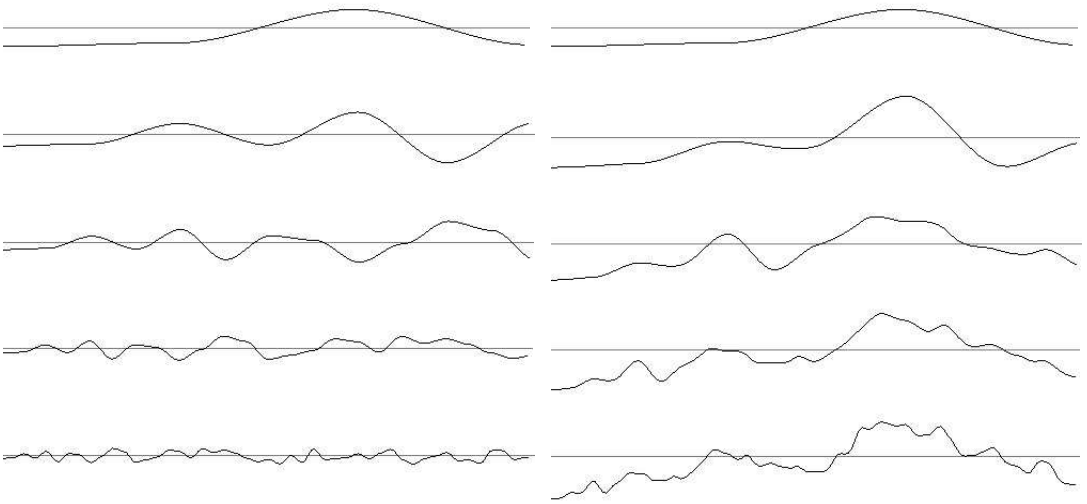
$$snoise(x, y, z, p, n) = \sum_{i=0}^{n-1} a_i \cdot noise(f_i\ x, f_i\ y, f_i\ z). \tag{13.5}$$

V tomto vztahu je n počet *oktáv* a $p \in (0, 1)$ je tzv. *persistence*, která určuje rychlost klesání vlivu každé oktávy na výsledný součet. To zajišťuje amplituda a_i $i-$té oktávy

$$a_i = p^i.$$

Frekvence f_i $i-$té oktávy se vypočítá (výpočet začíná vždy od hodnoty $i = 0$)

$$f_i = 2^i.$$



Obrázek 13.13: Skládání šumových oktáv (vlevo) do výsledného šumu

Například pro $p = 0.5$ získáme postupně následující oktávy

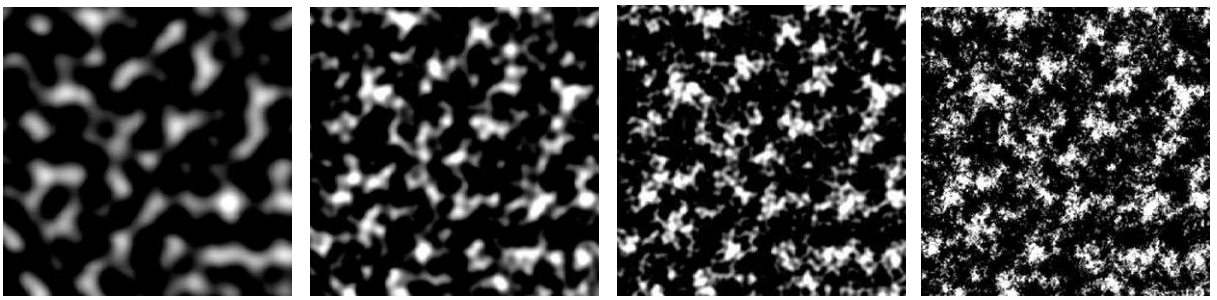
$$noise(x, y, z), \frac{noise(2x, 2y, 2z)}{2}, \frac{noise(4x, 4y, 4z)}{4}, \ldots, \frac{noise(2^i x, 2^i y, 2^i z)}{2^i}, \ \ldots$$

Amplituda jednotlivých členů klesá s $a_i = 1/2^i$.

Princip skládání šumových funkcí je ukázán na obrázku 13.13. Nalevo jsou zobrazeny jednotlivé oktávy a napravo jejich součet. Například třetí řádek má následující význam: levý sloupec zobrazuje třetí oktávu a pravý sloupec ukazuje součet prvních tří řádků z levého sloupce. Všimněme si, že amplituda funkcí v levém sloupci klesá, a funkce v pravém sloupci je obohacována o detaily. Podíl amplitud mezi dvěmi sousedními řádky levého sloupce je roven persistenci p, $a_i/a_{i-1} = p$. Vysoké oktávy udávají vliv vysokých frekvencí na výslednou funkci, neboli určují detaily. Nejnižší oktávy udávají základní charakteristiku výsledné funkce, její tvar. Obrázek 13.14 ukazuje formování šumové dvojrozměrné textury.

Součet šumových funkcí se také používá pro generování modelů krajin. Součet absolutních hodnot Perlinovy funkce poskytuje obrazy vizuálně podobné mrakům. Největší význam však tato funkce má jako tzv. *turbulence* aplikovaná v kombinaci s jinými technikami aplikace textur.

Perlinova funkce se používá pro simulaci textury mramoru a dřeva následujícím způsobem. Tzv. *barevná rampa*, na obrázku 13.15 vlevo, je modulována Perlinovou funkcí. Síla vlivu určuje „divokost“ textury. Barevná rampa se může určit mnoha způsoby, jedním z nejjednodušších je



Obrázek 13.14: Skládání šumu pomocí Perlinovy funkce (1,2,3 a 7 oktáv)

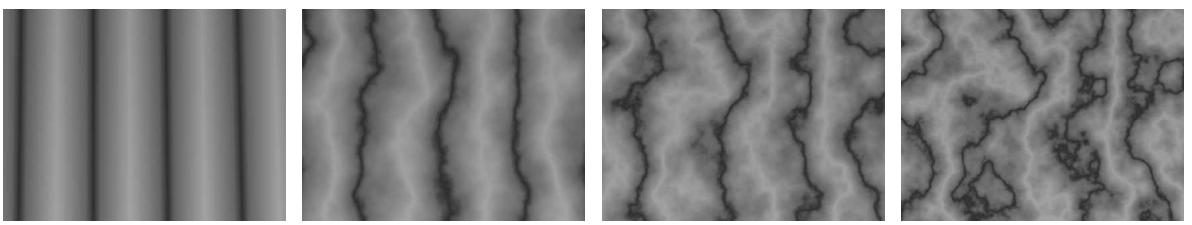
pomocí funkce:

$$rampa(x) = \frac{1}{2}\left(1 + \sin x\right). \tag{13.6}$$

Objemová textura mramoru se pak určí

$$mramor(x, y, z) = rampa(x + c \cdot snoise(x, y, z, p, a, n)).$$

Barva v konkrétním bodě je určena barevnou rampou a její parametr je modulován Perlinovým šumem. Síla vlivu je určena parametrem c, který je zadáván uživatelem a může být kladný i záporný. Zesilující vliv koeficientu c na barevnou rampu modulovanou Perlinovou funkcí demonstruje obrázek 13.15.



Obrázek 13.15: Spojitý barevný přechod (vlevo) je modulován postupně zesilující turbulencí

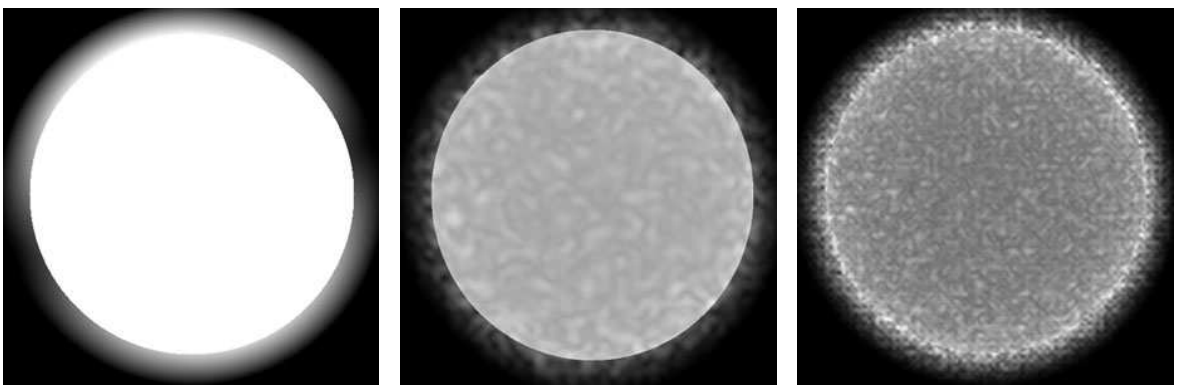
Trojrozměrnou texturu simulující strukturu dřeva získáme vztahem

$$wood(x, y, z) = perlin(x, y, z) - \lfloor perlin(x, y, z) \rfloor,$$

kde $\lfloor f(x) \rfloor$ vrací celočíselnou hodnotu funkce $f(x)$. Vhodnou volbou harmonických složek šumu ovlivníme vzhled struktury dřeva.

Hypertextura se hodí pro modelovaní objektů, které se podobají vlasům, textilu, trávě atd. Hypertextura udává změnu materiálu v blízkosti nad jeho povrchem. Hypertextura na obrázku 13.16 je posledním příkladem aplikace Perlinovy funkce. Předpokládejme kulový objekt, který má v poloměru $\langle 0, r_1 \rangle$ hustotu rovnu jedné, ve vrstvě $(r_1, r_2\rangle$ hustota lineárně klesá k nule. Pokud aplikujeme na tuto oblast součet Perlinovy šumové funkce, získáme objekt jehož příklad je na obrázku 13.16. Stejně jako v případě mramoru; rozdíl mezi obrázky je pouze v zesilujícím vlivu Perlinovy funkce.

Výhodou hypertextur je jejich nízká paměťová náročnost, nevýhodou je jejich zobrazování. Pro získání obrázku 13.16 bylo zapotřebí implementovat metodu vrhání paprsku, která ve voxelovém prostoru integruje příspěvky jednotlivých voxelů. Hodnoty voxelů jsou vypočítávány procedurálně a tak algoritmus potřebuje minimum paměti. Opakované volání šumové funkce je však pomalé.



Obrázek 13.16: Příklady hypertextur. Zleva doprava zesilující vliv Perlinovy funkce.

Kapitola 14

Reprezentace scény

V této kapitole nejprve popíšeme prvky, které se vkládají do prostorové scény, a uvedeme způsoby, jak je efektivně zorganizovat do grafu scény. Ve druhé části se seznámíme s pomocnými datovými strukturami (prostorovými hierarchiemi), které slouží k uspořádání trojrozměrného prostoru s cílem urychlit často prováděné operace, jako je nalezení (lokalizace) nejbližšího objektu v určité oblasti. Konec kapitoly je věnován detekci kolizí mezi prostorovými objekty.

Scénou nazýváme množinu prostorových objektů doplněnou dalšími informacemi potřebnými pro jejich zobrazení. Přestože se zdá, že vytvoření scény je poměrně jednoduchým závěrečným krokem po předchozím vymodelování individuálních prostorových objektů, je tvorbu prostorové scény možno chápat jako samostatnou úlohu. Zatímco systémy pro modelování těles zpracovávají geometrická data definující tvar jednotlivých objektů, systémy pro tvorbu scén přidávají k objektům transformace do jejich cílové polohy, určují informace potřebné pro zobrazování (světla, kamery) a především umožňují do scény opakovaně vkládat stejně nebo podobně vypadající objekty, tzv. *instance*. Scéna obvykle obsahuje:

- nezobrazované objekty – kamery, osvětlení scény;
- zobrazované objekty – jejich geometrie, barevné vlastnosti, textury;
- prvky definující logickou strukturu scény – definice skupin a jejich instancí;
- transformace – definované hierarchicky kvůli snadnější manipulaci s objekty.

Kamerou nazýváme množinu informací týkajících se jedné pohledové transformace. Pojem kamera byl uveden v kapitole 9, zde pouze připomeňme, že kamera není reprezentována žádným tělesem, neboť představuje virtuálního pozorovatele, který si scénu prohlíží. Z charakteristiky kamery tedy zobrazovací systém získá údaje o umístění pozorovatele, směru jeho pohledu a způsobu promítání. Tvůrce scény může do popisu scény zahrnout několik kamer nastavených

tak, aby poskytly pohledy na různá, významná místa ve scéně. Ke každé kameře lze doplnit údaje o animační křivce (blíže viz kap. 18.1.2), tedy o trajektorii, po které se kamera pohybuje, a o natáčení kamery.

Popis *osvětlení scény* zahrnuje údaje o světelných zdrojích. Jak bylo popsáno v kap. 10.6, světelné zdroje jsou buď abstraktní objekty určující pouze směr, odkud přichází světlo, nebo konkrétní plochy, které světlo emitují, ať již primárně či sekundárně. Abstraktní zdroje světla tedy nemají definován konkrétní geometrický tvar a pokud si je tvůrce scény přeje ukázat (žárovka, reflektor), musí je vymodelovat samostatně a zařadit mezi ostatní, běžná tělesa ve scéně.

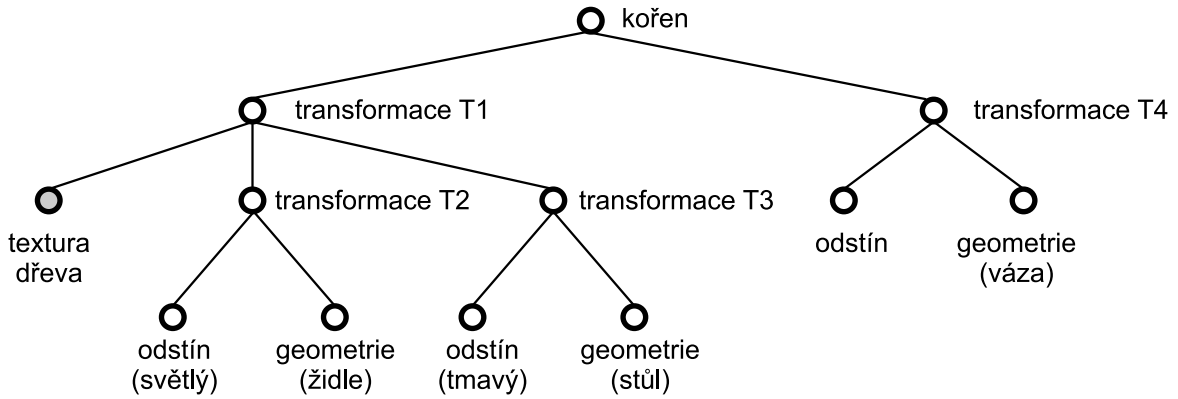
Tělesa a další zobrazované objekty je vhodné uspořádat do takové struktury, která umožňuje seskupovat logicky k sobě patřící části, efektivně je transformovat a jejich instance vkládat úsporným způsobem do prostoru scény. Tato struktura se obecně nazývá graf scény.

14.1 Graf scény

Graf scény (*scene graph*) je n-ární strom, tj. takový graf, v němž lze pro každý uzel nalézt právě jednoho předchůdce. Výjimku tvoří kořen stromu, který stojí na nejvyšší úrovni. Graf scény může obsahovat i několik stromů, tzv. les. Graf scény není ve všech systémech definován stejným způsobem, odlišnosti jsou v typech uzlů, v pravidlech pro stavbu stromu i pro interpretaci dat ve stromu uložených. V této části uvedeme principiální vlastnosti grafů scény a nebudeme se omezovat na konkrétní systém.

Důležitou vlastností stromu je schopnost vyjádřit vztahy mezi uzly. Jedním z těchto vztahů je *dědičnost*. Umožňuje v jednom místě stromu definovat vlastnost, která bude platná pro řadu dalších uzlů. Rozsah platnosti je dán vzájemnou polohou uzlů v rámci stromu. Lze například stanovit, že vlastnost definovaná v uzlu je platná pro všechny následníky tohoto uzlu. Některé systémy využívají graf scény pro zadání pevného pořadí následníků uzlu a předpokládají jejich systematické zpracování, například zleva doprava. To umožňuje rozšířit dědění, resp. sdílení určité vlastnosti i na uzly na stejné úrovni, umístěné v grafu scény vpravo od daného uzlu. Pravidla pro dědění mohou být definována různými způsoby, příklad je uveden na obrázku 14.1. Strom popisuje scénu se židlí, stolem a vázou. Židle a stůl mají totožnou texturu dřeva, avšak jiný odstín. Židle je světlá, stůl tmavý. Textura je tedy zděděna oběma tělesy, zatímco parametry odrazu světla jsou odlišné. Popis geometrie vázy je umístěn v jiné větvi stromu, mimo rozsah platnosti textury dřeva.

Na obrázku 14.1 si současně všimneme uzlů, označených jako transformace. Je zřejmé, že každé těleso může být pevně umístěno do své cílové pozice, tj. veškeré souřadnice tělesa mohou být předem transformovány pomocí posunutí, natočení, případně změny velikosti. Pro manipulaci se scénou je však mnohem vhodnější ponechat tělesa v jejich základních polohách



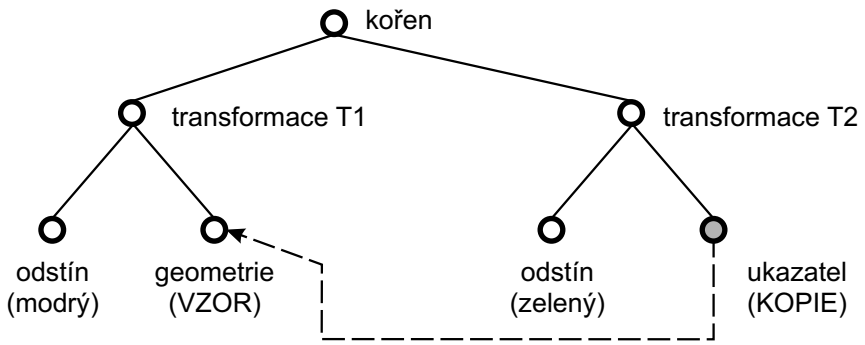
Obrázek 14.1: Dědění vlastností ve scéně popsané stromem

(lokálních souřadnicových systémech) a potřebné transformace zapsat do grafu scény, například v podobě transformační matice (viz též část 21.3). Hierarchické uspořádání scény umožní transformace skládat a změnou jedné transformace ovlivnit celý podstrom.

Skládání transformací se projeví při *interpretaci* grafu scény. Před vykreslením scény systematicky projdeme celý strom shora dolů a zleva doprava (*pre-order*). Při sestupu do nižší úrovně nalezené transformace ukládáme do zásobníku. Naopak při přesunu do vyšší úrovně transformace odebíráme. Jakmile dojdeme k uzlu, který uchovává geometrické údaje, aplikujeme na něj transformaci složenou ze všech transformací mezi ním a kořenem stromu. V příkladu na obrázku 14.1 je na židli aplikována složená transformace $T1.T2$, na stůl $T1.T3$ a na vázu pouze jediná transformace $T4$. Pokud bychom chtěli např. změnit polohu všech tří předmětů stejným způsobem, stačí do kořene přidat jedinou transformaci. Umístění transformací do samostatných uzlů je výhodné také pro animace, neboť při nich se v každém snímku aktualizují hodnoty právě v transformačních uzlech.

Při systematickém procházení grafu scény můžeme obdobně zpracovávat i další údaje potřebné k zobrazení objektů, například textury a materiálové vlastnosti (koeficienty odrazu světla). Tyto prvky se však na rozdíl od transformací neskládají, vždy je aplikována naposledy nalezená hodnota.

Graf scény může ovlivnit rozsah působnosti světelných zdrojů a napomoci k rychlejšímu vyhodnocení osvětlovacího modelu. Pokud tvůrce scény ví, že nějaký zdroj světla neovlivní vzhled určité části scény (například v modelu domu nepronikne světlo z chodby do obývacího pokoje), umístí zdroj světla do podstromu reprezentujícího model chodby. Díky stromové struktuře je působnost světla logicky omezena a při zobrazování předmětů v obývacím pokoji není nutno tento světelný zdroj vyhodnocovat.



Obrázek 14.2: Vzor a jeho instance ve scéně popsané stromem

Pokud se některé objekty ve scéně opakují, můžeme je definovat pouze jednou a jejich další instance popsat úsporným způsobem. K tomu slouží uzel obsahující ukazatel na jinou část stromu. Obrázek 14.2 znázorňuje část scény, v níž se nacházejí dva objekty se stejným tvarem (shodnou geometrií). První z nich má modrou barvu a je vzorem pro druhý, zelený objekt. Namísto potenciálně rozsáhlého popisu geometrie je v místě zeleného objektu použit ukazatel. Ten ukazuje na jediný uzel, obecně však může ukazovat na libovolně rozsáhlý podstrom, který pak bude celý sehrávat roli vzoru. Zajímavé je vyhodnocení složených transformací pro tento graf. Přestože je na vzorový objekt aplikována transformace $T1$, jeho další instance (kopie) bude transformována pouze transformací $T2$. To odpovídá výše uvedenému systematickému procházení a akumulování transformací. Pokud bychom z nějakého důvodu chtěli na kopii objektu použít složenou transformaci $T1.T2$, musel by ukazatel směřovat o úroveň výše, například na transformaci $T1$.

Poznamenejme, že grafová struktura obsahující takto zavedené ukazatele se odborně nazývá *zhroucený strom* (*collapsed tree*). Uzel, který je vzorem pro další kopie, má totiž několik předchůdců. To však není překážkou pro systematické procházení grafu scény při jejím vykreslování. Mezi praktická omezení při práci s ukazateli v grafu scény patří požadavek, aby ukazatel směřoval do levé části stromu a neukazoval na některého ze svých předchůdců (tím se zamezí vzniku cyklů).

Graf scény nemusí vždy odpovídat stromu z obrázku 14.1. Scénu je možno rozdělit na několik logických sekcí s informacemi stejného typu, např. na skupinu definic textur, skupinu koeficientů odrazivosti, skupinu souřadnic normál apod. Objekty potom obsahují kromě své geometrické reprezentace i sadu ukazatelů do výše uvedených skupin. Z hlediska konkrétního uložení scény je praktické používat více souborů, ať již pro textury či pro definice vzorů. Výhodou je nejen použití různých kompresních metod příslušných určitému typu dat, ale též možnost snadné změny vzhledu scény záměnou obsahu datových souborů. Některé formáty pro uložení scény

(např. VRML) dovolují do scény vkládat i odkazy na vzdálené soubory umístěné kdekoliv na Internetu. Rozdělení grafu scény do více souborů ovšem zvyšuje nebezpečí neúplnosti dat (nekonzistence), např. při archivaci.

Bez ohledu na konkrétní tvar grafu scény je důležité mít objekty scény organizované do logických struktur. Opakem hierarchicky uspořádaného grafu scény je množina elementárních geometrických prvků, např. polygonů, bez jakýchkoliv vzájemných vazeb. Takové množině se říká *polygonální polévka* (*polygon soup*) a pro efektivní editaci a další zpracování je nevhodná.

14.2 Pomocné datové struktury

Jedním z požadavků při zobrazování scén a při manipulaci s nimi, je určení objektů, nacházejících se v zadané prostorové oblasti, případně nalezení objektů, které mají neprázdný průnik se zadanou přímkou či polopřímkou (paprskem). Tyto úlohy musejí být řešeny zejména v pokročilých zobrazovacích metodách, jakými jsou sledování paprsku a radiosita (viz kap. 15). Mají význam také v oblasti virtuální reality, kde je třeba v reálném čase vybírat ze scény pouze objekty blízké pozorovateli a snížit tak požadavky na výkon zobrazovacího systému. Další úlohou je nalezení blízkých objektů při vyhodnocování možných kolizí.

Vhodné uspořádání informací v prostoru výrazně snižuje časové nároky při zpracování scény, zároveň však přináší nutnost zavedení dalších, pomocných datových struktur. Tyto datové struktury nebývají součástí popisu scény, ale jsou většinou vytvářeny při načtení scény. Struktury mají hierarchický charakter a zachycují trojrozměrný prostor, proto se jim někdy říká *prostorové hierarchie*. Při práci s pomocnými datovými strukturami rozlišujeme dvě fáze:

1. Počáteční jednorázové vytvoření datových struktur a jejich naplnění informacemi (odkazy) o objektech ve scéně.
2. Využívání pomocných datových struktur jako primárního prostředku pro vyhledávání informací o objektech scény.

Objevují se i přístupy, kdy jsou pomocné datové struktury vytvářeny dynamicky až při zobrazování scény s ohledem na frekvenci přístupu ke konkrétním objektům [Arvo89].

Prostorové hierarchie dělíme do dvou základních kategorií. *Hierarchie obálek* (*BVH, Bounding Volume Hierarchies*) vznikají postupným shlukováním objektů a obálek, které je obklopují. *Hierarchie dělení prostoru* (*space subdivision hierarchies*) se vytvářejí postupným rozdělováním trojrozměrného prostoru, nejčastěji s ohledem na objekty, které jsou v tomto prostoru umístěny. U obou kategorií je výsledkem stavby prostorové hierarchie stromová struktura. Kořen reprezentuje všechny objekty scény, resp. prostor, v němž se scéna nachází. V následujících obrázcích budeme prostorové hierarchie kreslit ve zjednodušené, rovinné podobě.

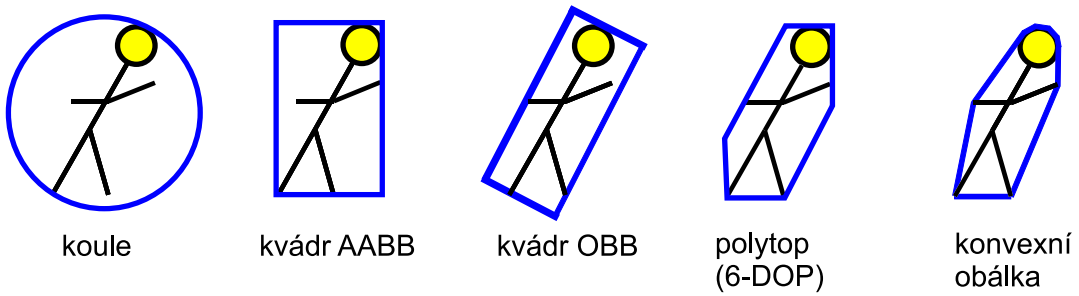
14.2.1 Hierarchie obálek

Ohraničení objektu obálkou (*bounding volume*), tedy tělesem s velmi jednoduchou geometrií, patří k základním urychlovacím postupům v grafice vůbec. Vychází z myšlenky, že uvažovaný test (test polohy objektu, test průsečíku objektu s paprskem apod.) lze realizovat mnohem rychleji s obálkou než s objektem, který obálka obklopuje. Tento test se obecně nazývá *předběžný test* (*hit/miss test*) a poskytuje výsledek TRUE nebo FALSE. V případě, že výsledek testu je negativní, není třeba přistupovat k testování vlastního objektu.

Při optimalizaci algoritmů se používá několik druhů obálek, které jsou ukázány na obrázku 14.3. Nelze jednoznačně říci, která obálka je lepší. Jejich volba záleží na řešené úloze. Obálky s jednoduchými tvary (koule, osově orientovaný kvádr) se snadno vytvářejí a aktualizují při transformacích obklopovaných objektů, ale hůře se těmto objektům přizpůsobují – často obklopí společně s objektem i větší prázdný prostor a předběžný test poskytne pozitivní výsledek, přestože test s vlastním objektem dopadne negativně. Složitější obálky se hůře aktualizují, ale lépe se přizpůsobí objektům. Vlastnosti a definice obálek uvádí následující tabulka.

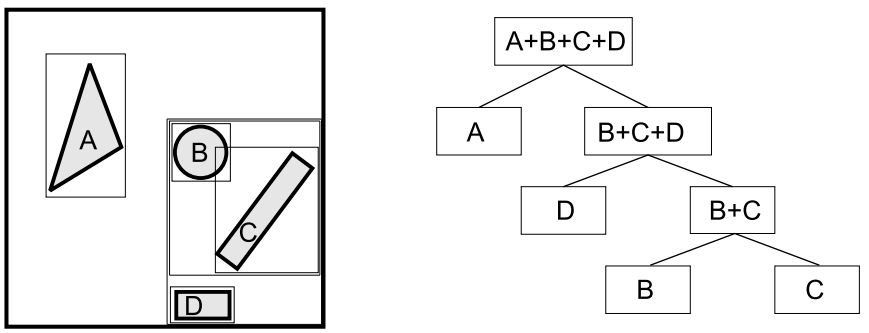
Typ obálky	*Zkratka*	*Popis a vlastnosti (anglický termín)*
koule	—	je invariantní vůči rotaci objektu
osově zarovnaný kvádr	AABB	stěny kvádru jsou kolmé na souřadnicové osy, je to současně min-max obálka všech bodů objektu (*Axis-Aligned Bounding Box*)
orientovaný kvádr	OBB	obklopující kvádr je orientován tak, aby jeho objem byl co nejmenší (*Oriented Bounding Box*)
orientovaný rovnoběžnostěn	k-DOP	průnik několika pásů v prostoru určuje tzv. polytop (*k-Discrete Orientation Polytop*). Protilehlé roviny obálky jsou tedy rovnoběžné a jejich normály jsou definovány v k diskrétních směrech. Nejčastější varianty jsou: 6-dop – kvádr (typu AABB) 14-dop – kvádr s oseknutými rohy 18-dop – kvádr s oseknutými hranami 24-dop – kvádr s oseknutými rohy a hranami

Samotné využívání obálek přináší jen mírné zrychlení výpočtů, neboť testy se musejí provést s tolika obálkami, kolik je ve scéně objektů. K výraznějšímu zrychlení dochází teprve tehdy, když obálky seskupíme do hierarchické stromové struktury, nazývané *hierarchie obálek*. Tvorba hierarchie obálek je naznačena na obrázku 14.4 ukazujícím strom obálek ve tvaru osově zarovnaných kvádrů (*AABB tree*). Pomocnou datovou strukturou je v tomto případě strom, v jehož listech jsou obálky ohraničující individuální objekty scény. Obálky, které leží blízko sebe,



Obrázek 14.3: Čtyři druhy obálek používané v pomocných datových strukturách a pro srovnání konvexní obálka (vpravo) (obrázek poskytla M. C. Lin, University of North Carolina, USA).

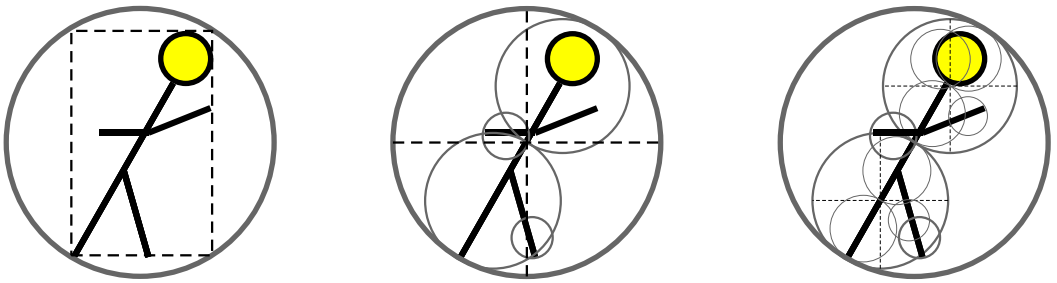
nebo které se překrývají, se postupně shlukují do větších celků a zapisují se do uzlů, ležících ve vyšších vrstvách stromu. Při shlukování je strom obálek stavěn zdola nahoru.



Obrázek 14.4: Scéna se čtyřmi tělesy a hierarchií obálek typu AABB

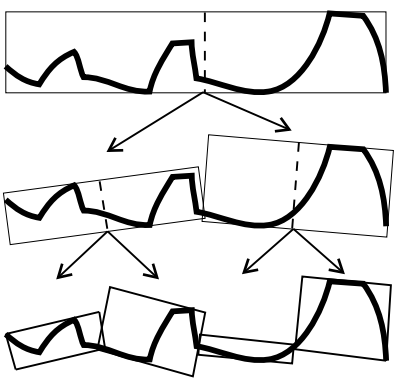
Při úloze zjišťování polohy bodu vzhledem k objektům ve scéně procházíme strom obálek směrem od kořene a testujeme jeho polohu vůči obálce obsažené v uzlu. Pokud je bod uvnitř obálky, jsou dále rekurzívně procházeni následníci uzlu. Na obrázku 14.4 je vidět výhodu hierarchické struktury oproti jednoduchému seznamu obálek. Pokud testovaný bod leží v blízkosti tělesa A, stačí provést dva předběžné testy se dvěma uzly, následníky kořene (je-li jisté, že bod leží uvnitř scény, nemusí být kořen testován). Bez použití stromu bychom museli provést čtyři předběžné testy pro obálky individuálních těles. Lineární složitost prohledávání neuspořádaného prostoru se díky hierarchii změní na složitost logaritmickou. To však pouze v případě, že strom je vyvážený, tj. má co nejmenší hloubku. Jak ukazuje obrázek 14.4, ne vždy lze takový strom postavit.

Efektivitu hierarchie výrazně ovlivňuje schopnost obálky přizpůsobit se tvaru objektu. Všimněme si například, že při testování bodu v blízkosti tělesa B na obr. 14.4 je nutno prověřit všechny uzly stromu, tedy provést více testů než bez použití hierarchie obálek.



Obrázek 14.5: Stavba stromu koulí shora dolů (podle [Palm95]).

Hierarchie obálek lze budovat i shora dolů. Pro některé úlohy je výhodné, když je strom obálek přiřazen individuálním objektům se složitější vnitřní strukturou, tj. větším počtem ploch. Postup stavby hierarchie koulí (*sphere tree*) dle [Palm95] je znázorněn na obrázku 14.5. Těleso nejprve obklopíme obálkou typu AABB, pro kterou vytvoříme obklopující kouli. Ta představuje nejhrubší náhradu tělesa a je kořenem hierarchie. Dále rozdělíme obklopující kvádr třemi navzájem kolmými řezy v polovině na tzv. oktanty a pro každý neprázdný oktant provedeme rekurzívně předcházející krok. Dělení ukončíme buď v neobsazeném oktantu, nebo v okamžiku, když je v oktantu méně plošek, nežli je určitý, pevně stanovený počet. Takto vytvořený strom koulí je nepravidelný – vnitřní uzly mohou mít různý počet následníků (dva až osm).



Obrázek 14.6: Stavba stromu OBB

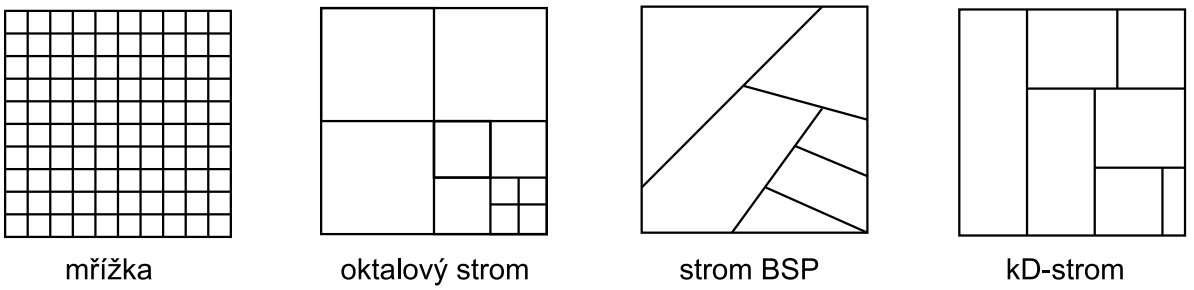
Podobným způsobem lze postavit hierarchii orientovaných kvádrů (*OBB tree*) [Gott96]. Obrázek 14.6 ukazuje na dvourozměrném případě, jak je strom OBB vytvářen. Objekt je nejprve jako celek obklopen obálkou (první řádek) a tato obálka představuje kořen hierarchie. Poté je těleso rozděleno na dvě části (nejčastěji ve svém těžišti). Ke dvěma takto získaným objektům jsou vytvořeny jejich OBB. Tyto dva OBB jsou zároveň následníky kořene. Algoritmus se aplikuje rekurzívně na nově získané objekty. Rekurzi ukončíme po dosažení určité hloubky stromu nebo snížení počtu plošek v obálce pod daný limit. Takto vytvořený strom OBB je binárním stromem.

Individuální hierarchie obálek nacházejí uplatnění v aplikacích, v nichž potřebujeme porov-

nat vzájemné prostorové vztahy dvou (nebo více) objektů. S tím se setkáváme při zpracování scén ve virtuální realitě (viz kap. 20), kde je např. nutno zabezpečit, aby návštěvník virtuálního světa neprocházel zdmi a jinými pevnými objekty. Další typickou úlohou je detekce kolizí, která je stručně popsána na konci této kapitoly.

14.2.2 Dělení prostoru

Odlišným přístupem oproti hierarchii obálek je systematické rozdělení prostoru scény (*scene partitioning*). Způsobů, jak trojrozměrný prostor dělit na menší části, existuje celá řada. Nejznámější techniky jsou znázorněny na obrázku 14.7.



Obrázek 14.7: Čtyři nejčastěji používané způsoby dělení prostoru (obrázek poskytla M. C. Lin, University of North Carolina, USA)

Bližší popis základních technik dělení prostoru je uveden v následující tabulce. S výjimkou pravidelné mřížky se výsledek dělení prostoru zaznamenává do hierarchické datové struktury – stromu. Dělení je prováděno rovinným řezem.

Typ dělení	*Popis a vlastnosti (anglický termín)*
pravidelná mřížka	není hierarchií, pouze pravidelně dělí prostor rovinami kolmými na souřadnicové osy (*grid*)
oktalový strom	dělení třemi řezy kolmými na souřadnicové osy a umístěnými v polovině daného prostoru (*octree*). Dělením vzniká vždy osm oktantů.
strom BSP	binární strom s obecně umístěnými řezy (*Binary Space Partitioning tree*)
kD-strom	speciální případ stromu BSP určený k popisu libovolného k-rozměrného (*k-Dimensional*) prostoru. Řezné roviny jsou vždy kolmé na některou ze souřadnicových os a orientace řezů se pravidelně střídají. Za k se kdysi dosazoval rozměr zpracovávaného prostoru (2D, 3D apod.), dnes se užívá jen univerzální zkratka kD (*kD-tree*).

Kromě těchto typů existuje několik dalších variant. Jednou z nich je *hierarchie mřížek*, která iniciální, hrubé rozdělení prostoru zjemňuje pomocí hustších mřížek obklopujících objekty nebo jejich skupiny. Variantou stromu BSP je *ortogonální strom BSP*, jehož řezné roviny jsou kolmé na souřadnicové osy. Na rozdíl od kD-stromu se jejich směr nemusí pravidelně střídat.

Při stavbě stromu uchovávajícího výsledky postupného dělení prostoru se používá algoritmus 14.1. Prostor obsahující scénu je rekurzívně dělen na menší části (buňky). Do vnitřních uzlů stromu zaznamenáváme informace o dělení (např. polohu a orientaci řezu). Listy obsahují seznam odkazů na tělesa, která svým objemem zasahují do odpovídající prostorové oblasti (buňky). Pokud nějaký objekt zasahuje do více buněk, zvyšuje se počet odkazů na něj a tedy i paměťová náročnost pomocné datové struktury. Často se používají obálky, a to jak pro objekty scény, tak pro vnitřní uzly hierarchie.

PostavStrom (buňka B, hloubka H)

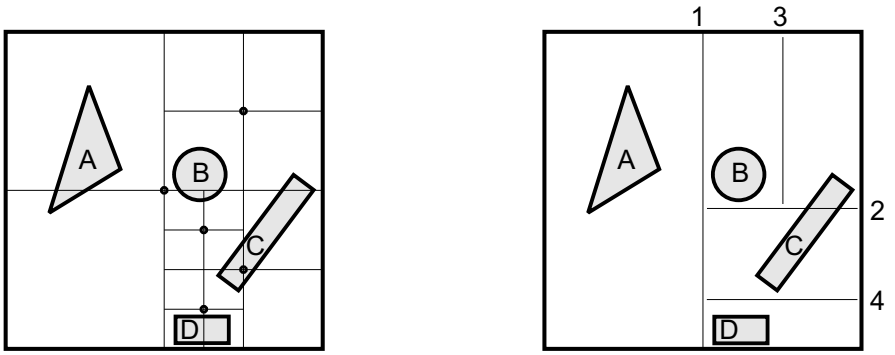
1. Pokud je hloubka stromu H velká nebo buňka B obsahuje malý počet těles, skonči
2. Rozděl buňku B na části B_i a učiň z nich následníky uzlu (buňky) B
3. Pro každou z buněk B_i dělej
   (a) Do buňky B_i zařaď odkazy na ta tělesa, která do ní alespoň částečně zasahují
   (b) PostavStrom (B_i, $H + 1$)
4. Zruš odkazy na všechna tělesa v buňce B

Algoritmus 14.1: Rekurzivní stavba stromu při dělení prostoru

Rekurzivní algoritmus 14.1 se zahájí nad jedinou buňkou (uzlem), která obsahuje všechny objekty scény. Volání metody má pak tvar PostavStrom (scéna, 0). V kroku (2) vznikají nové uzly stromu, jejichž počet závisí na typu dělení (obvykle dva nebo osm). Podmínku v kroku (1) pro ukončení rekurze lze rozšířit o požadavek, aby dělení bylo zastaveno i v případě, kdy další dělení již nepřinese žádoucí zjednodušení (typicky snížení počtu těles v uzlu).

Volba hierarchie dělení prostoru je ovlivněna různými kritérii, která zahrnují jak paměťové nároky datových struktur, tak rychlost algoritmů a testů prováděných nad výsledným stromem. Uveďme, že zatímco pro hierarchie obálek je příznačné obklopování nadbytečného prostoru (*space redundancy*), hierarchie dělení prostoru jsou charakteristické svým nadbytečným zaznamenáváním objektů (*object redundancy*). Při pravidelném dělení (mřížka, oktalový strom) vzniká jednak mnoho prázdných buněk, jednak jsou tělesa často protnuta řezy, což vede na opakování odkazů na tělesa v listech. Tvar a rozmístění buněk ovšem umožňuje rychle prohledávat

tyto hierarchie. Strom BSP je pro většinu scén úspornější. Vhodnou volbou řezů zamezíme vzniku prázdných buněk a můžeme také minimalizovat počet těles řezem protnutých. Obecný tvar buněk vyžaduje složitější výpočty při prohledávání stromu BSP.



Obrázek 14.8: Porovnání výsledku dělení pomocí oktalového stromu (vlevo) a binárního stromu.

Srovnání dvou hierarchií je na obrázku 14.8. Oktalový strom má dvanáct neprázdných listů a čtyři prázdné listy. Celá čtvrtina listů nenese žádnou informaci o objektech scény. Všimněme si také častého protnutí těles řeznými rovinami. Hierarchie vpravo je ortogonální strom BSP. Toto dělení prostoru se může daleko víc přizpůsobit konkrétnímu rozložení objektů ve scéně. V uvedeném příkladu obsahuje strom BSP pouze čtyři dělicí roviny (na obrázku postupně číslovány) a tedy jen pět listů. Žádný z listů není prázdný. Těleso C je jednou protnuto.

Dále uvedeme algoritmus procházení stromu pro jednu konkrétní úlohu – nalezení nejbližšího tělesa protnutého polopřímkou (paprskem) [Sung92]. Je to úloha prováděná v metodě sledování paprsku (viz část 15.9). Na rozdíl od poměrně jednoduchého testu, zda se v nějaké části prostoru nachází objekt scény, je v tomto případě algoritmus o něco složitější. Prochází hierarchii tak, aby postupně poskytoval buňky na dráze paprsku ve správném pořadí. Algoritmus je navržen pro binární stromy. Využívá zásobník a zahajuje procházení hierarchie v kořeni. Při sestupu odkládá na zásobník ty uzly, které by mohly být prozkoumány později (jsou na vzdálenější straně řezných rovin). Pokud paprsek mine všechna tělesa obsažená v jednom listu, další kandidáti jsou na vrcholu zásobníku.

Postup je popsán v algoritmu 14.2. Předpokládá se, že ve vnitřních uzlech jsou uloženy informace o řezné rovině a obálka typu AABB odpovídající buňky. Rozhodnutí o zařazení uzlu do zásobníku je určeno polohou průsečíku paprsku s řeznou rovinou. Při ohodnocení bodů P_N a P_F v kroku 4 se vzdálenost od počátku paprsku bere jako orientovaná, neboť počátek paprsku může být umístěn i dovnitř scény a tedy mezi body P_N a P_F. Algoritmus se inicializuje po vložení kořene stromu BSP do zásobníku.

Dokud není zásobník prázdný (tj. paprsek míří mimo scénu) nebo dokud není nalezen hledaný průsečík paprsku s tělesem, dělej:

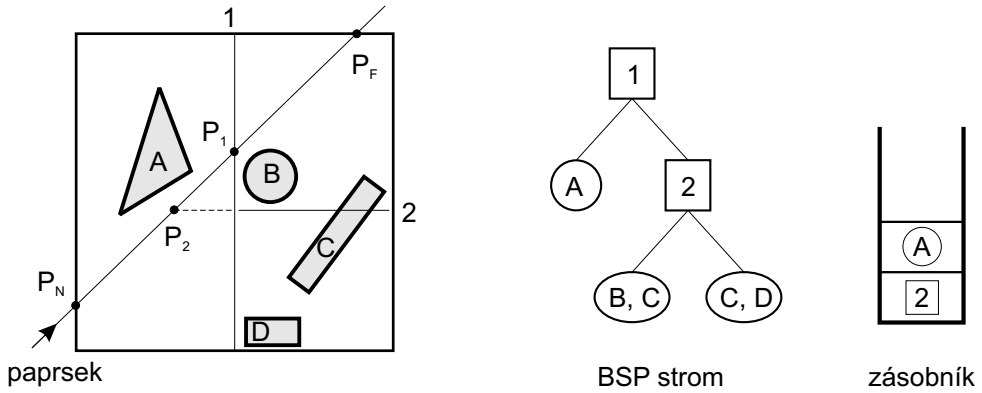
1. Vyber ze zásobníku uzel U
2. Je-li U listem, pak
   (a) Pro všechna tělesa v listu zjisti, zda je paprsek protne, a zapamatuj si nejbližší z protnutých těles
   (b) Byl-li nalezen průsečík, skonči, jinak jdi na krok 1
3. Uzel U je vnitřní. Vypočítej průsečík P paprsku a řezné roviny příslušné tomuto uzlu
4. Urči průsečíky paprsku s obálkou uzlu U. Průsečík bližší k počátku paprsku označ jako P_N, vzdálenější jako P_F (viz též obr. 14.9)
5. Potomka uzlu U, reprezentujícího poloprostor určený řeznou rovinou a obsahující počátek paprsku, označ jako *bližší uzel*. Druhého potomka uzlu U označ jako *vzdálenější uzel*
6. Podle polohy bodu P vzhledem k P_N a P_F ulož na zásobník:
   (a) *bližší uzel*, pokud P leží za P_F
   (b) *vzdálenější uzel*, pokud P leží před P_N
   (c) nejprve *vzdálenější uzel* a potom *bližší uzel* v ostatních případech

Algoritmus 14.2: Procházení binárního stromu při sledování paprsku

Procházení stromu BSP ukazuje příklad na obr. 14.9. Oba uzly, odpovídající podprostorům na obou stranách řezné roviny 1, jsou podle kroku 6 (c) uloženy do zásobníku, protože průsečík P_1 leží mezi body P_N a P_F obálky uzlu č. 1 ve stromu BSP. Při zpracování uzlu č. 2 dojde podle pravidla 6 (b) k vynechání uzlu s objekty C, D. Průsečík P_2 paprsku s řezem 2 leží před bodem P_1, který je pro uzel č. 2 chápán jako bod P_N.

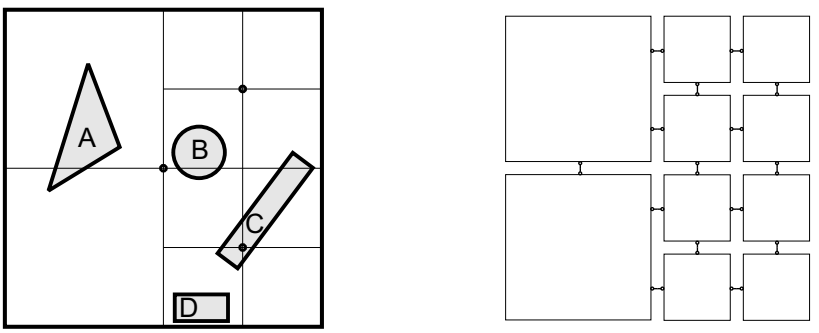
Při implementaci je možno algoritmus doplnit o různá praktická vylepšení. Je-li odkaz na totéž těleso uložen v několika různých listech stromu (viz těleso C na obr. 14.9), je zbytečné opakovat pro toto těleso test výpočtu průsečíku s daným paprskem. Řešením je zavedení tzv. *poštovní schránky* (*mail box*), což je paměťová buňka přiřazená každému tělesu. Jakmile je jednou proveden test paprsku s tělesem, je do schránky tohoto tělesa zapsáno identifikační číslo, které zamezí opakovanému testování.

Procházení stromu v uvedeném algoritmu je zahájeno vždy od kořene, a to i v případě, kdy je z předchozích výpočtů známo, ve kterém listu stromu bylo sledování paprsku ukončeno.



Obrázek 14.9: Použití zásobníku při procházení BSP stromu

Vhodnější by tedy bylo zahájit procházení stromu v naposledy zpracovávaném listu. Takové systematické procházení je algoritmicky náročnější. Namísto relativně komplikovaných přechodů do vyšších úrovní hierarchie a následných sestupů je výhodné přecházet mezi uzly s ohledem na jejich sousednost nikoliv ve stromu, ale v prostoru. K tomu je nutno doplnit datové struktury dalšími odkazy, neboť buňky, které spolu sousedí v trojrozměrném prostoru, mohou být v pomocných stromových strukturách umístěny do vzdálených větví. Odkazy, umožňující rychlý přeskok v rámci stromové struktury na prostorově sousední uzel, se nazývají *provazy* (*rope*). Příklad jejich použití je na obrázku 14.10. Má-li určitá buňka na jedné straně více sousedů, může mít v daném směru více provazů nebo jen jediný, směřující na vnitřní uzel stromu nadřazený všem sousedním buňkám. Vzhledem k obtížnější údržbě datových struktur není uspořádání s provazy vhodné pro dynamicky se měnící scény.



Obrázek 14.10: Scéna s oktalovým stromem (vlevo) obohaceným o provazy (vpravo)

14.3 Detekce kolizí

Algoritmy výpočtu kolizí se uplatňují ve virtuální realitě, v počítačových animacích a simulacích. Požadavky na detekci kolizí přitom mohou být odlišné. V případě počítačové animace jde především o přesnost výpočtu a nezáleží příliš na čase, který tomuto výpočtu věnujeme. Výpočty kolizí se uplatňují v inverzní kinematice, kde můžeme například polohu koncového efektoru – chodidlo syntetické figurky – určit jako výsledek kolize s libovolně nakloněným a hrbolatým terénem. Ve výsledné animaci je chůze postavy přizpůsobena terénu. Jiným příkladem je výpočet kolizí při animaci oděvu syntetického herce. Látka bývá reprezentována jako hustá síť trojúhelníků a vypočítávají se kolize každého trojúhelníku s každým a navíc i s tělem syntetického herce tak, aby byla výsledná animace co nejpřesvědčivější.

Ve virtuální realitě jsou požadavky jiné. Spokojíme se i s drobnými nepřesnostmi ve výpočtu, jen když je rychlost výpočtu co nejvyšší. Při vyhodnocení kolizí nám nezáleží, zda postava avatara naprosto přesně stojí na podložce (může stát kousek nad ní, či dokonce v ní), podstatné je, aby nedocházelo vlivem výpočetní náročnosti k výraznému snížení snímkové frekvence a tím i k nežádoucímu „loudání“ (*lagging*) zobrazovaného děje.

Výpočet kolize lze rozdělit do dvou kroků. Prvním je *detekce kolize* a druhým je *generování odezvy* na kolizi. Druhou částí se zde nebudeme zabývat. V dalším textu uvedeme dva algoritmy pro detekci kolizí. Obě metody jsou založeny na hierarchické struktuře obklopujících těles – obálek. V prvním případě jsou těmito obálkami koule, ve druhém případě kvádry. U obou těchto algoritmů není podstatná reprezentace těles, jejichž kolizi detekujeme. V dalším textu budeme předpokládat výpočet kolizí pouze mezi dvěma objekty.

První metoda využívá hierarchie kulových obálek postavené nad každým z testovaných objektů [Palm95]. Připomeňme, že výhodou koule je její invariance k otáčení, nevýhodou je, že nemusí těleso obklopovat příliš těsně.

Detekce kolizí (viz algoritmus 14.3) probíhá mezi dvěma objekty, které mají každý svou hierarchii obklopujících obálek (viz obrázek 14.11). V kroku (1) algoritmu 14.11 se nejprve detekuje kolize koulí, které jsou v kořenu obou hierarchií. Tento test vyžaduje porovnání vzdáleností středů a poloměrů obou koulí. Pokud ke kolizi došlo, zjistí se v kroku (3), zda jedna z koulí není listem stromu. Pokud tomu tak je, algoritmus prochází druhou hierarchii až do případné úrovně listu, v níž následuje přesný výpočet kolize na úrovni trojúhelníků v kroku (2). Pokud algoritmus nedosáhl listu jednoho ze dvou stromů, postoupí se v jednom ze stromů o jednu úroveň níže a provede se detekce kolize všech koulí v této úrovni s vyšetřovaným uzlem stromu hierarchie druhého objektu. Toho je docíleno v kroku (4) záměnou parametrů u rekurzivního volání procedury pro výpočet kolize.

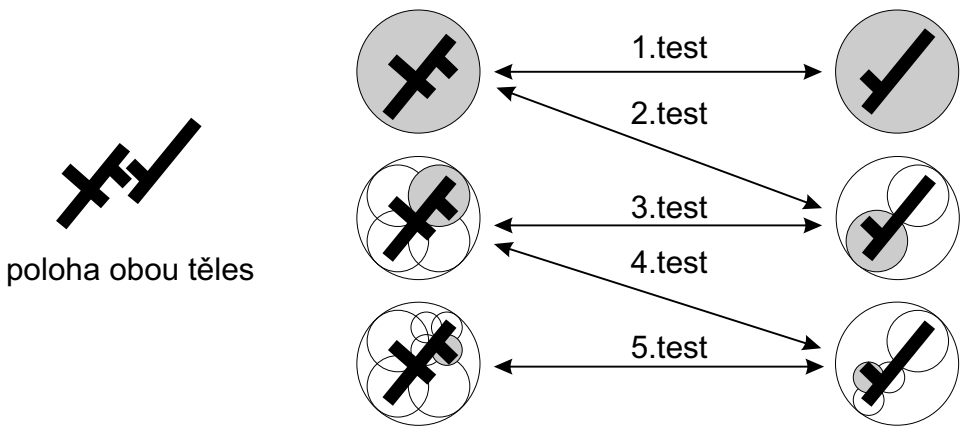
Algoritmus skončí buď vyloučením kolize, nebo nalezením první dvojice kolizních objektů. Neprovádí úplnou detekci kolize v rámci hierarchií objektů. Pro řešení korektních odezev

DetekujKolizi ($k1, k2$)
Vstup: uzly dvou stromů obalových koulí $k1$ a $k2$ testovaných objektů
Výstup: informace o tom, zda došlo ke kolizi

1. pokud nedošlo ke kolizi mezi obalovými koulemi uzlů $k1$ a $k2$, skonči s negativním výsledkem
2. pokud jsou $k1$ a $k2$ listy stromu, vypočti kolizi přesně a skonči
3. pokud je jeden z $k1$ a $k2$ listem stromu,
   DetekujKolizi mezi ním a všemi následníky druhého uzlu
4. jinak pro všechny následníky uzlu $k1$ označené $k1_i$
   DetekujKolizi ($k2$, $k1_i$)

Algoritmus 14.3: Detekce kolizí

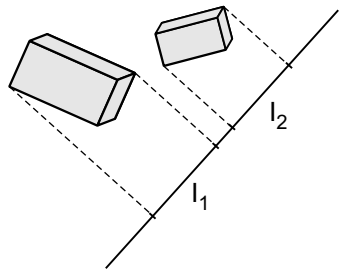
na kolize je třeba algoritmus příslušně upravit. Vyloučení kolize nastává v praktických aplikacích ve více než 90 % případů, a proto je tento algoritmus velice efektivní.



Obrázek 14.11: Testování dvou stromů koulí

Jiná varianta tohoto algoritmu se používá pro generování časově závislých kolizí ve virtuální realitě. Podstatou tohoto problému je, že na výpočet kolize máme pouze omezený čas. Algoritmus prochází oba stromy tak dlouho, dokud mu nedojde předem určený čas. Potom je použit dosavadní výsledek operace. V praktických aplikacích se takto násilně pozastavený výpočet může projevit jako zastavení objektu ještě před dosažením překážky, nebo naopak částečný

průchod překážkou. Ukazuje se však, že pro získání kvalitního vjemu ve virtuální realitě nejsou tyto chyby tak závažné oproti zpomalení snímkové frekvence při úplném provedení testu.



Obrázek 14.12: Promítnutí OBB na přímku

Poslední algoritmus zmíněný v této části se od předcházejícího liší pouze v tom, že využívá jiné obklopující obálky. Díky použití obálek typu OBB je algoritmus rychlejší, protože orientované kvádry obklopují těleso lépe než koule. Klíčem k efektivitě tohoto algoritmu je test průniku (kolize) dvou obálek typu OBB. V prvním kroku promítneme oba kvádry OBB na přímku tak, jak ukazuje obrázek 14.12. Promítnutí realizujeme například tak, že přímku i OBB otočíme takovým způsobem, aby přímka byla totožná s osou x souřadnicového systému a poté nalezneme největší a nejmenší x-ovou souřadnici OBB. Provedením tohoto výpočtu pro oba testované OBB získáme na této přímce dva intervaly (na obrázku 14.12 označené I_1 a I_2). Pokud nedošlo k jejich překrytí, nemohou se oba OBB protínat a test končí. Pokud došlo k jejich překrytí, musí se provést další test.

Přímka, na které se nepřekrývají intervaly vzniklé projekcí OBB, se nazývá *oddělující osa* (*separating axis*). V [Gott96] je ukázáno, že pro úplný test kolize dvou OBB stačí provést tento test pouze s patnácti přesně určenými přímkami v prostoru. Pokud je jedna z těchto přímek oddělující osou, ke kolizi nedošlo a test končí. V nejhorším případě je nutno provést test se všemi patnácti osami. V praktické implementaci se provádí v průměru sto instrukcí procesoru na testování průsečíků dvou OBB a tento algoritmus je tedy velmi rychlý.

Efektivita algoritmů pro výpočty kolizí je výrazně ovlivňována tím, jak obálky a jejich hierarchie těsně obklopují objekty ve scéně. Z tohoto důvodu jsou často využívány i hierarchie obálek typu k-DOP [Klos98].

Kapitola 15

Globální zobrazovací metody

Dosud uvedené zobrazovací metody se vyznačovaly tím, že objekty ve scéně se navzájem neovlivňovaly z hlediska osvětlení. Každý objekt měl individuálně (lokálně) vyhodnocen osvětlovací model tak, jako kdyby byl ve scéně osamocen a světlo na něj dopadalo ze všech světelných zdrojů. Jediný okamžik, kdy jsme byli nuceni vzít do úvahy vzájemné polohy objektů, nastal při určení viditelnosti. I tehdy bylo možno říci, že objekty neovlivňovaly osvětlení jiných objektů, ale že se „pouze“ zakrývaly. Výhodou výpočtu lokálního osvětlení je vysoká rychlost výpočtu zobrazování a snadná implementace v hardwaru.

V reálném světě se existence objektů v prostoru výrazně projevuje při interakci se světlem. Pokud jsou mezi jedním objektem a zdrojem světla umístěny další objekty, leží tento objekt ve stínu či polostínu. Naopak odražené světlo od velkého světlého tělesa může zvýšit osvětlení stěn těles od přímých zdrojů světla odvrácených. Jiným případem vzájemného ovlivňování je odraz obrazu objektů na lesklém povrchu jiných objektů, kaustiky (viz část 15.3) způsobené vodní hladinou, mlha ve vzduchu aj.

Cílem globálního osvětlování, a možná i celé počítačové grafiky, je nalezení metody, která bude simulovat všechny optické jevy v reálném čase. Cílem osvětlovacích technik je samozřejmě výpočet osvětlení a nepřímo tedy generování obrazu. Výsledkem metod globálního osvětlování jsou tak kvalitní obrazy prostorových scén, že je obtížné odlišit je od fotografií skutečného světa. Proto se jim také říká metody pro *fotorealistické zobrazování* (*photorealistic rendering*).

Globální charakter vzájemných vztahů mezi objekty formálně popisuje zobrazovací rovnice, se kterou se seznámíme v této kapitole. Řešení osvětlovací rovnice v obecném případě je analyticky nemožné. Zobrazovací metody jsou aproximací řešení analytického.

Dále uvedeme zobrazovací rovnici a poté základní jevy vznikající při přenosu světla, které je nutné simulovat, aby byl výsledek realistický. V další části se seznámíme s moderními technikami globálního osvětlování. Popíšeme metody založené na náhodném vzorkování a na

závěr podrobně vysvětlíme dvě nejčastěji používané metody globálního osvětlování – metodu sledování paprsku a radiační metodu.

Před čtením této kapitoly doporučujeme důkladné prostudování kapitoly 10. Na mnoho pojmů z této části, především na radiometrické veličiny, se budeme odkazovat.

15.1 Zobrazovací rovnice

Zobrazovací rovnice, se kterou se seznámíme v této části, byla formulována v [Kaji86] a je matematickou definicí problému zobrazení scény. Její řešení udává pro každý bod povrchu každé plochy a pro každý směr vycházející radianci. Umístěním virtuální kamery do scény pak snadno určíme radianci jež prochází každým pixelem výsledného obrazu a tedy i jeho barvu. Výchozím kritériem zobrazovací rovnice je zákon zachování energie. Radiance opouštějící bod povrchu musí být někde odražena či absorbována. Řešení zobrazovací rovnice pak popisuje ustálený stav ve scéně.

Zobrazovací rovnice předpokládá že jsou splněny veškeré podmínky uvedené v úvodu kapitoly 10 a z nich plyne jedno důležité omezení. Zobrazovací rovnice *nepopisuje* případ s opticky aktivním prostředím (*participating media*).

Matematická formulace osvětlovacího problému umožňuje nasazení matematického aparátu na jeho řešení. Získané řešení potom zbývá jen zobrazit. Naneštěstí zobrazovací rovnice je složitým integrálním vztahem, jehož analytické řešení je v obecném případě nemožné a musí se proto hledat aproximační řešení. Dále v textu uvidíme, že jak empirické modely, tak globální osvětlovací metody je vždy možno popsat, jako částečné řešení zobrazovací rovnice.

Předpokládejme bod x na povrchu nějakého objektu. Naším cílem bude vyjádřit celkovou radianci $L_o(x, \omega)$ opouštějící tento bod ve směru ω (dolní index je z anglického *outgoing*).

Bod x může být světelným zdrojem a vyzařovat v daném směru radianci $L_e(x, \vec{\omega})$. Dalším příspěvkem k opouštějící radianci je celková odražená radiance odpovídající lokálnímu osvětlovacímu modelu (10.8)

$$L_r(x, \vec{\omega}) = \int_\Omega f(x, \vec{\omega}, \vec{\omega}_i) L_i(x, \vec{\omega}_i) \cos\Theta \; d\vec{\omega}_i.$$

V tomto vztahu je $f(x, \vec{\omega}, \vec{\omega}_i)$ dvousměrová odrazová distribuční funkce – BRDF a Θ je úhel sevřený směrem $\vec{\omega}_i$ a normálovým vektorem k povrchu. Připomeňme, že $\cos\Theta = \vec{n} \cdot \vec{\omega}_i$. Radiance opouštějící bod x ve směru $\vec{\omega}$ je potom dána součtem vlastního záření a odražené radiance

$$L_o(x, \vec{\omega}) = L_e(x, \vec{\omega}) + L_r(x, \vec{\omega}) = L_e(x, \vec{\omega}) + \int_\Omega f(x, \vec{\omega}, \vec{\omega}_i) L_i(x, \vec{\omega}_i) \cos\Theta \; d\vec{\omega}_i. \qquad (15.1)$$

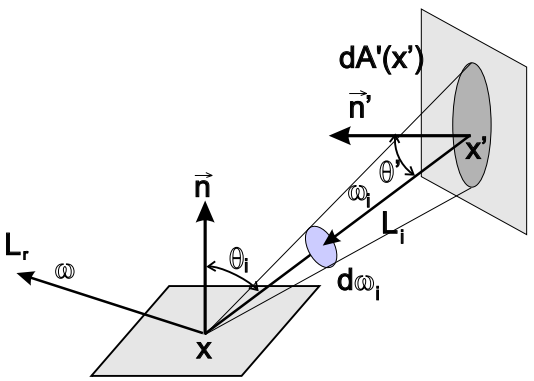
Tato rovnice se také označuje VTIGRE (*vacuum, time/invariant, gray radiance equation*). Originální rovnice [Kaji86] používá pouze vyzářenou radianci, označuje se OVTIGRE (*outgoing,*

vacuum, time/invariant, gray radiance equation) a má tvar

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + \int_\Omega f(x,\vec{\omega},\vec{\omega}_i) L_o(x,-\vec{\omega}_i)\cos\Theta \; d\vec{\omega}_i. \tag{15.2}$$

Tato rovnice se patří mezi tzv. Fredholmovy rovnice druhého řádu a její komplikací je, že neznámá (radiance) se vyskytuje na levé straně i na pravé straně uvnitř integrálu.

Zobrazovací rovnice (15.1) obsahuje na pravé straně člen, který zahrnuje integrál přes polokouli Ω. V praxi je výhodnější formulace, která udává odraženou radianci jako integrál přes přispívající plochy. Předpokládejme rozložení z obrázku 15.1 a následující význam. Bod x je vyšetřovaný bod a $L_r(x,\vec{\omega})$ je odražená radiance. Úhel Θ_i je úhel mezi dopadající radiancí $L_i(x,\vec{\omega}_i)$ ze směru $\vec{\omega}_i$ a normálovým vektorem v bodě x. Výpočet dopadající radiance je dán integrálem pro všechny body x' plochy A'. Úhel Θ' je úhel mezi normálovým vektorem $\vec{n}'$ k této ploše a směrem $\vec{\omega}_i$ k vyšetřovanému bodu x. Odražená radiance se vyjádří jako integrál přes *všechny* plochy S ve scéně



Obrázek 15.1: Geometrie pro zobrazovací rovnici vyjádřenou pomocí plošek

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + \int_S f(x,x'\to x,\vec{\omega}) L_i(x'\to x) V(x,x') G(x,x') dA'. \tag{15.3}$$

V tomto vztahu jsme změnili notaci pro směry odrazu radiance, již nepoužíváme zápis pro polokoule, ale snažší $x \to x'$ jehož význam je zřejmý. V rovnici (15.3) se vyskytují dva dosud nepopsané členy. $V(x,x')$ určuje viditelnost (*visibility term*) bodu x z bodu x' a naopak:

$$V(x,x') = \begin{cases} 1 & \text{pokud je bod } x \text{ viditelný z bodu } x' \\ 0 & \text{v opačném případě} \end{cases}.$$

Tzv. geometrický člen (*geometry term*) $G(x,x')$ v sobě zahrnuje koeficienty vyjadřující promítnutí radiance po vyzáření z bodu x' a po dopadu do bodu x:

$$G(x,x') = \frac{(\vec{\omega}\cdot\vec{n}')(\vec{\omega}'\cdot\vec{n})}{\|x'-x\|^2}.$$

Pokusme se shrnout co rovnice (15.3) říká. Pro výpočet radiance odcházející ve směru $\vec{\omega}$ z bodu x na nějaké ploše musíme vzít v úvahu všechny ostatní plochy ve scéně. Pro všechny

jejich body x' vypočteme viditelnost bodu x. Je-li bod viditelný, vyjádříme geometrický člen a provedeme vynásobení odcházející radiance tímto členem a BRDF v bodě x. Výsledek, integrál přes všechny plochy ve scéně, spolu s vlastní vyzářenou radiancí udává radianci vyzářenou daným směrem.

Přejděme na chvilku zpět k vyjádření (15.1) a zapišme integrál pomocí tzv. integrálního operátoru T, který zjednoduší zápis a umožní nám zbavit se v dalším textu integrálů.

$$< Tg > (x, \vec{\omega}) = \int_{\Omega} f(x, \vec{\omega}, \vec{\omega}_i) G(x, x') \cos\Theta \; d\vec{\omega}_i.$$

S použitím tohoto operátoru můžeme zobrazovací rovnici přepsat na vztah:

$$L = L_e + TL. \tag{15.4}$$

Tato rovnice je ze své podstaty rekurentní. K tomu, abychom vyjádřili radianci jdoucí do bodu x potřebujeme znát radiance z bodů x', jenže ty se musí vypočítat stejným způsobem. Tato rovnice má neznámou radianci na levé a na pravé straně; jedná se o rekurentní vztah a může se rozepsat pomocí Neumannovy řady na:

$$L = L_e + TL_e + T^2 L_e + \ldots = \sum_{i=0}^{\infty} T^i L_e,$$

kde T^0 označuje prázdnou aplikaci operátoru. Tento zápis nám sděluje to, co již víme: odražená radiance z bodu je dána vlastním zářením L_e a vlastním zářením všech ploch po prvním odrazu, tj. aplikaci operátoru TL_e, dále vlastním zářením po dvou odrazech $T^2 L_e$ atd. Důležité však je, že jsme získali popis začínající u světelných zdrojů vedoucí k výsledné radianci.

15.2 Notace transportu světla

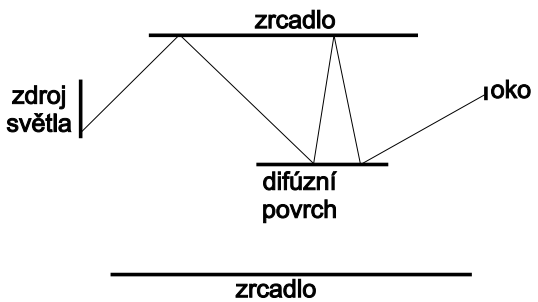
Heckbert [Heck90] popsal formální notaci, která je v počítačové grafice používána pro popis způsobu transportu a odrazu světla. Používá následující symboly (viz část 10.4):

- L – světelný zdroj (*light*),
- D – odraz od difúzního povrchu (*diffuse*),
- S – odraz od zrcadlového povrchu (*specular*),
- G – odraz od lesklého povrchu (*glossy*),
- E – dopad paprsku do oka, či do kamery (*eye*).

Kombinace různých odrazů a jejich varianty jsou popsány regulárními výrazy:

- (k)? žádný či jeden odraz,
- (k)+ jeden či více odrazů,
- (k)* žádný či více odrazů,
- (k|q) cesta k nebo q.

Paprsek jdoucí ze světelného zdroje přímo do oka se označí LE. Jednou zrcadlově odražený paprsek LSE, paprsek jdoucí ze světelného zdroje jednou difúzně odražený a nekončící v oku se označí LD, paprsek odražený od lesklého povrchu a poté několikrát, ale alespoň jednou odražený difúzně se označí LGD+E atp. Uvažme příklad na obrázku 15.2. Paprsek se odrazí nejprve od zrcadla, poté od difúzního povrchu, znovu do zrcadla, difúzního povrchu a nakonec padne do oka. Cesta paprsku se označí LSDSDE.



Obrázek 15.2: LSDSDE

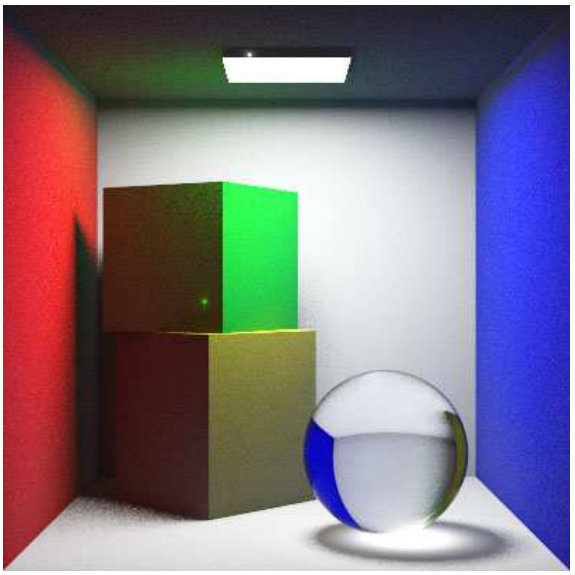
15.3 Základní optické jevy

Snahou počítačové grafiky je simulace co nejširšího spektra optických jevů. Některé z nich však jsou více důležité pro věrnost zobrazení scény a některé méně. Dále popíšeme ty nejdůležitější a u jednotlivých metod globálního osvětlování později uvedeme, zda a s jakou pracností, tyto jevy simulují.

Přímé osvětlení (*direct illumination*), též zvané lokální osvětlovací model, simuluje světlo dopadající k pozorovateli po jediném odrazu. Přímé osvětlení simuluje dráhy světla LDE, LSE či LGE. K jeho výpočtu je zapotřebí znalost charakteristiky světelného zdroje a BRDF na povrchu objektu. Přímé osvětlení je důležité pro vnímání barvy objektu. Asi nejtypičtějším příkladem takového modelu je empirický Phongův osvětlovací model popsaný v části 10.5.

Stín je důležitý pro vnímání objektu ve třech dimenzích. Z polohy stínu člověk intuitivně určuje uspořádání objektů ve scéně. K výpočtu stínu je zapotřebí uvážit rozložení ostatních objektů ve scéně, nelze ho tedy zjistit pouze z polohy jediného objektu. Tvar stínu je dán umístěním a charakteristikou světelného zdroje a stínícího objektu (viz část 12).

Násobné odrazy jsou způsobeny povrchy objektů, které odráží světlo. Ač se jedná o jediný jev, jeho projevy lze dále rozdělit na zrcadlový odraz, difúzní nepřímý odraz, difúzní přenos barvy a kaustiky.



Obrázek 15.3: Typický výsledek globálního osvětlování. V rozích obrázku získaného fotonovými mapami je patrný příspěvek násobných difúzních odrazů, stíny mají polostíny, protože scéna je osvětlena plošným zdrojem, a na podlaze je kaustika (obrázek poskytl R. C. Moreno, ITESM Campus Ciudad de México).

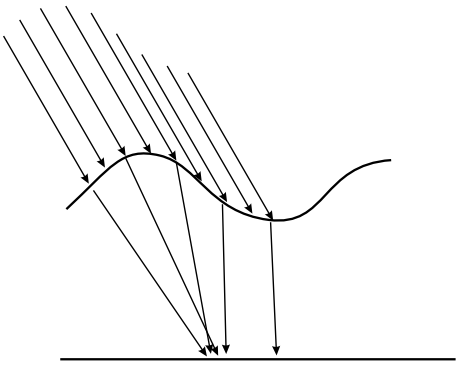
Zrcadlový odraz se projevuje jako odlesky na plochách, jasné odrazy světla a lesklé odrazy (*highlights*). Je způsoben hladkými materiály s vysokou odrazivostí. Trajektorii zrcadlově odraženého světla lze popsat jako L(S+|G+)*E.

Difúzní nepřímý odraz je jedním z časově nejnáročnějších jevů z hlediska simulace. Tento druh odrazu se v realitě projevuje například jako pomalá změna intenzity odrazu světla v rozích místností. Zdi jsou nejčastěji natřeny vysoce difúzní barvou a násobný odraz světla na nich se projevuje právě tak, jak je vidět na obrázku 15.3. V Phongově osvětlovacím modelu je násobný difúzní odraz aproximován konstantním ambientním členem, jeho přesnější řešení poskytuje například radiační metoda, poměrně dobré aproximace poskytují metody vzorkování Monte Carlo.

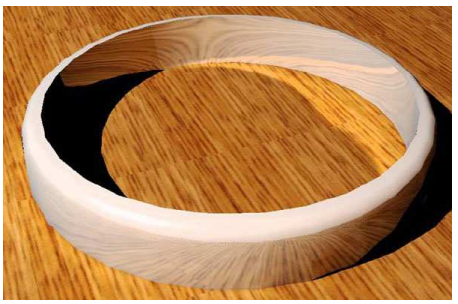
Difúzní přenos barvy (*color bleeding*) je subtilní jev, který je důsledkem difúzního odrazu světla a vzniká při těsné blízkosti dvou difúzních povrchů různé barvy. Barva z jednoho je emitována na druhý a naopak. Příkladem je bílá zeď sousedící se zdí červenou. V silném bílém světle budou mít přilehlé části bílé zdi načervenalý nádech.

Kaustiky („prasátka“) vznikají při koncentraci paprsků odražených od zrcadlového materiálu či prošlých průhledným materiálem. Obrázek 15.4 a) demonstruje princip vzniku tohoto jevu

a 15.4 b) je příklad získaný metodou mapování fotonů. Kaustiky jsou dobře patrné například na dně bazénu, kde vznikají průchodem a lomem paprsků na vodní hladině. Z důvodu lomu paprsku dochází ke koncentraci dopadajících paprsků v některých oblastech, které jsou pak světlejší. Cesta paprsku může mít libovolný průběh, podstatné je, že na některém místě musí dojít ke koncentraci zrcadlových příspěvků a poslední odraz před tím, než paprsek dorazí do oka, musí nastat na difúzním nebo hrubě lesklém povrchu.



a)



b)

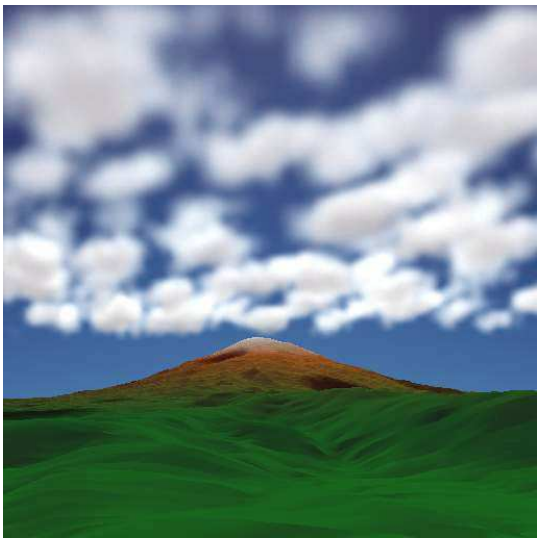
Obrázek 15.4: a) Princip vzniku kaustiky při lomu paprsku b) příklad kaustiky vzniklé koncentrací odrazem (obrázek poskytl J. Křivánek, ČVUT v Praze)

Opticky aktivní média (*participating media*) jsou nejsložitějším případem transportu světla, který navíc není popsán zobrazovací rovnicí. Radiance je rozptylována na částečkách prachu, vody, dýmu či kouře, uvnitř poloprůhledných materiálů, aj. Výsledkem je poměrně složitý přenos světla. Metody simulující tento způsob přenosu světla jsou v posledních letech v popředí zájmu počítačových grafiků. Příklad výsledku takové simulace je na obrázku 15.5.

15.4 Globální osvětlovací techniky

Globální osvětlovací techniky hledají řešení zobrazovací rovnice. Jejich cílem je buď výpočet osvětlení všech ploch ve scéně, tomu se říká pohledově nezávislé řešení, nebo výpočet osvětlení pro určitý směr, tj. pohledově závislé řešení. Pohledově nezávislé řešení poskytuje například radiační metoda, příkladem druhého přístupu je metoda sledování paprsku.

Metody globálního osvětlování se liší podle způsobu přístupu ke scéně. Některé z nich musí scénu nejprve rozdělit na menší části, typicky na sítě trojúhelníků. Výsledkem výpočtu algoritmu globálního osvětlování je potom osvětlení vrcholů v těchto sítích. Dělení je nejčastěji prováděno metodou konečných prvků, odtud i název těchto metod (*finite element methods*). Z jejich povahy plyne, že musí mít k dispozici celou scénu, kterou poté zpracovávají. Proto



Obrázek 15.5: Příklad obrazu získaného simulací šíření světla v opticky aktivním prostředí (obrázek poskytl M. Poneš, ČVUT v Praze)

se také někdy říká, že se ke scénám chovají `r/w` (*read/write*). Metody, které ke scéně pouze přistupují a nijak ji nemění, se nazývají `r/o` (*read/only*) metody.

15.4.1 Monte Carlo metody

Podstatou globálních osvětlovacích metod je nalezení řešení zobrazovací rovnice pro konkrétní scénu. Analytické řešení je ve většině případů nemožné, a proto se používají metody Monte Carlo. Jejich podstatou je aproximace řešení nějakého problému stochastickým vzorkováním[1]. Hledá se náhodná proměnná, jejíž střední hodnota je řešením problému. Metody Monte Carlo se používají pro řešení široké škály zejména fyzikálních problémů. Základní výhody metod Monte Carlo (podle [Jens01]) jsou:

- Používají bodové vzorkování, tj. sledují paprsek, který putuje modelovanou scénou. Je tedy možné použít celý existující aparát geometrické optiky, který byl původně vyvinut pro metodu sledování paprsku.

[1] Princip Monte Carlo metod popíšeme na jednoduchém příkladě. Předpokládejme že chceme metodou Monte Carlo vypočítat integrál $\int_a^b f(x)dx$. Aproximaci řešení poskytuje průměr z n rovnoměrně rozmístěných náhodných vzorků $\xi_i, i = 0, 1, \ldots, n$. Vzorec $(b-a)/n \sum_i f(\xi_i)$ pro $n \to \infty$ konverguje k řešení.

- Geometrie scény může být procedurální. Nemusíme mít k dispozici trojúhelníky aproximující původní povrch.
- Není zapotřebí rozdělení objektů na menší části, můžeme pracovat s objekty jako s celky.
- Metody Monte Carlo mohou pracovat s jakoukoli BRDF.
- Zrcadlový odraz lze snadno vypočítat pro libovolnou geometrii.
- Mají nízkou paměťovou náročnost.
- Přesnost řešení je dána počtem pixelů. To je dáno pohledovou závislostí řešení. Vypočítáváme obraz, jeho přesnost závisí v první řadě na jeho rozlišení.
- Empirická složitost těchto metod je $\mathcal{O}(\log n)$, kde n je počet objektů ve scéně. Nejrychlejší metody založené na konečných prvcích mají empirickou složitost $\mathcal{O}(n \log n)$.
- Řešení není zatíženo systematickou chybou (*unbiased*). Jediná chyba těchto metod je šum, který však může být silný, a pro některé případy těžko odstranitelný.

Nejdůležitější rozdělení Monte Carlo metod globálního osvětlování je dáno způsobem vyšetřování trajektorie světla. To je prováděno buď od pozorovatele, od zdrojů světla, či v obou směrech najednou.

Jednou z nevýhod metod Monte Carlo je jejich pomalá konvergence. Přesnost řešení roste s $\sqrt{n}$, kde n je počet vzorků. Na zdvojnásobení přesnosti tedy potřebujeme čtyřikrát více vzorků. V počítačové grafice můžeme za další omezení považovat fakt, že Monte Carlo metody poskytují pohledově závislé řešení.

15.5 Metody vycházející od pozorovatele

První třída metod vychází ze sledování trajektorie světla od pozorovatele. Vzhledem k tomu, že světlo se pohybuje od svého zdroje k cíli, je možné tyto metody označit jako zpětné sledování trajektorie světla. Protože tyto metody „sbírají“ energii, kterou paprsek na své dráze akumuluje, říká se jim *gathering methods*. Nejstarší z těchto metod je metoda zpětného sledování paprsku [Whit80], popsaná detailně v části 15.9.

Motivace pro tento postup je nasnadě. Větší část světelných paprsků vyzářených světelným zdrojem nedorazí ani po mnoha odrazech k pozorovateli. Sledování světla bude tedy náročné a mnoho úsilí bude vynaloženo na paprsky, které se k pozorovateli nikdy nedostanou. Z tohoto důvodu je jednodušší sledovat trajektorii světla v opačném směru. Základní algoritmus metod, jež vycházejí od pozorovatele, 15.1 vede z každého pixelu několik paprsků, zjistí jejich příspěvky a vypočte z nich výslednou barvu. Jednotlivé metody se liší v implementaci procedury Sleduj(paprsek) z řádku 2.

Pro všechny pixely v obraze
  barva=pozadí
Pro n vzorků
  1. Vypočti paprsek jdoucí od pozorovatele pixelem do scény.
  2. barvaPaprsku=Sleduj(paprsek)
  3. barva+=barvaPaprsku/n

Algoritmus 15.1: Generický algoritmus metod začínajících od pozorovatele

15.5.1 Sledování paprsku

Metoda rekurzivního sledování paprsku [Whit80] (*ray tracing*) je klasickou metodou globálního osvětlování a jejímu popisu se věnujeme detailně v části 15.9. Zde pouze popíšeme její princip v kontextu ostatních metod globálního osvětlování.

Základní algoritmus sledování paprsku bez dalších rozšíření pracuje s bodovými zdroji světla a proto poskytuje pouze ostré stíny bez polostínů. Základní algoritmus sledování paprsku bez dalších rozšíření pracuje s bodovými zdroji světla a transport světla je omezen na zrcadlové odrazy.

Sleduj(paprsek)
  1. (objekt,x)=NejbližšíPrůsečík(paprsek)
  2. není-li zasažen žádný objekt vrať barvu pozadí
  3. barva=$L_e(x,\omega)$+Osvětlení(x)
  4. je-li (koeficient průhlednosti materiálu>0) barva+=koef_tSleduj(paprsek_t)
  5. je-li (koeficient zrcadlového odrazu>0) barva+=koef_zSleduj(paprsek_r)
  6. vrať vypočtenou barvu

Algoritmus 15.2: Procedura realizující sledování paprsku

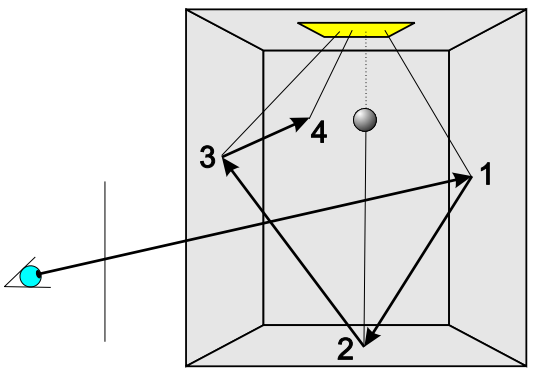
Program 15.2 schématicky popisuje vyšetřování trajektorie světla metodou sledování paprsku. Nejprve se vypočítá první průsečík paprsku se scénou. Není-li zasažen žádný objekt, vrátí se barva pozadí a výpočet se ukončí. V případě zásahu objektu se vypočítá barva jako příspěvek vlastní vyzářené radiance a lokálního osvětlovacího modelu (řádek 3 programu). Je-li materiál průsvitný, vypočítá se lomený paprsek, který jím prochází, a rekurzívně se sleduje jeho dráha. Pátý řádek programu je rekurzivní volání sledování paprsku, které se uplatní pouze pokud má povrch zrcadlovou složku nenulovou.

Metoda sledování paprsku vypočítává trajektorie L(D|G)?S*E, tj. libovolný počet zrcadlových odrazů a lokální osvětlovací model s difúzní a lesklou složkou osvětlení. Metoda sledování paprsku není schopna vypočítat kompletní řešení zobrazovací rovnice, konkrétně kaustiky, plošné a světelné zdroje a difúzně–difúzní odrazy, tedy ani difúzní přenos barvy. Existují různá rozšíření, z nichž asi nejdůležitější je tzv. distribuovaná metoda sledování paprsku (*distributed ray-tracing*), která vypočítává kaustiky a pracuje s plošnými zdroji světla [Watt92].

15.5.2 Sledování cesty

Metoda označovaná jako sledování cesty (*path tracing*) byla poprvé popsána ve stejném článku jako zobrazovací rovnice [Kaji86]. Protože se jedná o sledování cesty vzorkováním metodou Monte Carlo, bývá také nazývána Monte Carlo sledování cesty (*Monte Carlo path tracing*).

Tato metoda poskytuje kompletní řešení zobrazovací rovnice, i když za cenu velkého výpočetního výkonu. Jedná se, stejně jako v předcházejícím případě, o metodu sledující dráhu světla od pozorovatele. Základním rozdílem oproti metodě sledování paprsku je, že vypočítává příspěvky od nepřímých difúzních obrazů, pracuje s libovolnými světelnými zdroji (tedy i s plošnými) a je schopna vypočítat kaustiky a přenos barvy difúzním odrazem. Metoda Monte Carlo sledování cesty zpracovává dráhy světla typu L(S|D)+E.



Obrázek 15.6: Monte Carlo sledování cesty

Principem metody je sledování zcela náhodného paprsku od pozorovatele tak, jak popisuje algoritmus 15.3. Každým pixelem se vyšle velké množství paprsků a sledují se jejich cesty scénou. V každém průsečíku paprsku se scénou se vypočítá lokální osvětlovací model včetně detekce stínů. Na obrázku 15.6 je schematicky naznačeno, že se v každém odrazu vyšle stínový paprsek do světelného zdroje. Plošný světelný zdroj se může vzorkovat i více paprsky. V případě průsečíku číslo dva je dráha paprsku přerušena stínícím objektem. V dalším kroku (čtvrtý řádek programu 15.3) se náhodným vzorkováním BRDF v průsečíku určí směr dalšího sledování. Tento bod je nejdůležitějším rozdílem oproti metodě sledování paprsku. Zatímco při sledování paprsku se berou v úvahu pouze zrcadlové odrazy, zde se vypočítává náhodný odraz. Při použití metody sledování paprsku bude tedy trajektorie vždy stejná, v případě sledování cesty bude trajektorie pro každý vzorek různá.

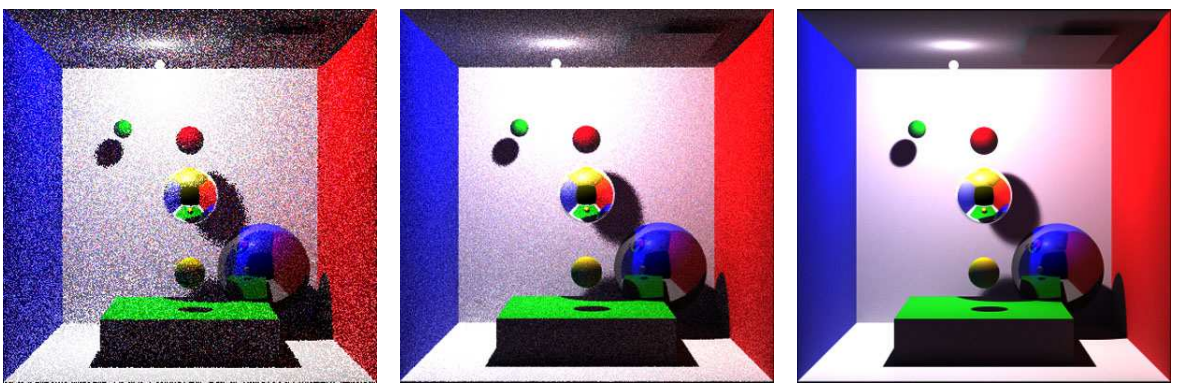
Algoritmus sledování cesty se realizuje integrací pomocí metody Monte Carlo. Neznámá, jejíž hodnota se hledá, je distribuce světla ve scéně. Hledá se stochastickým sledováním všech

Sleduj(paprsek)

1. (objekt,x)=PrvníPrůsečík(paprsek)
2. není-li zasažen žádný objekt vrať barvu pozadí
3. $barva$=$L_e(x,\omega)$+Osvětlení(x)
4. Vypočti odražený paprsek vzorkováním BRDF
5. barva+=Sleduj(paprsek)
6. Je-li materiál průhledný, vypočti paprsek_t a barva+=Sleduj(paprsek_t)
7. vrať barvu

Algoritmus 15.3: Procedura realizující Monte Carlo sledování cesty

možných světelných trajektorií. Průměrem ze vzorků se pak získá odhad integrálu ze všech cest jdoucích vyšetřovaným pixelem. Metoda používá jeden odražený paprsek a zvýšené vzorkování světelných zdrojů. Pokud bychom v každém průsečíku vyšetřovali více paprsků, došlo by k exponenciálnímu nárůstu paprsků ve scéně. Empirickým poznatkem je, že je výhodnější přednostně sledovat trajektorie, které mají menší počet odrazů, neboť jejich příspěvek bývá vyšší.

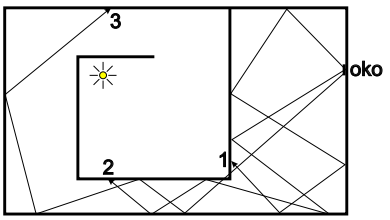


Obrázek 15.7: Šum metody Monte Carlo sledování cesty. Obrázky byly vypočteny s hloubkou 100 odrazů a s následujícím počtem paprsků procházejících každým pixelem: 1, 10 a 1 000.

Protože vzorkování je náhodné, problémem této metody je šum. Změna hloubky sledování paprsku, zvýšení množství odrazů, nemá pro kvalitu výsledku zásadní význam, pro většinu

scén postačuje čtyři až pět, maximálně pak deset odrazů. Naopak je tomu s počtem vyšetřovaných paprsků každým pixelem. Obrázek 15.7 ukazuje, jak se mění kvalita scény s počtem vyšetřovaných paprsků na každý pixel. V praxi tisíc až deset tisíc paprsků pro každý pixel postačuje. Tato metoda konverguje rychle k řešení především pro scény, ve kterých je změna nepřímého osvětlení pomalá, například scéna osvětlená zataženou oblohou popsané rovnicí (10.16). V takové scéně nevznikají složité optické jevy, jako kaustiky a její celková světelná dynamika je nízká. V těchto případech postačí i pouhých sto paprsků na pixel.

Zcela nevhodné pro tuto metodu jsou scény se složitými světelnými poměry, jak je naznačeno na obrázku 15.8. Paprsky vycházející od pozorovatele jsou zastaveny po pěti odrazech. K tomu, aby bylo dosaženo světla, či jasných ploch, je však zapotřebí více odrazů. Naprostá většina paprsků vycházejících od pozorovatele bude mít po několika prvních odrazech nulový světelný příspěvek, protože světlo je „za rohem“. K dosažení osvětlených ploch je zapotřebí více odrazů. V realitě však podobná scéna bude osvětlená. Monte Carlo sledování cesty bude potřebovat pro přesný výpočet takovéto scény obrovský počet paprsků.



Obrázek 15.8: Obtížné světelné podmínky pro výpočet osvětlení metodou Monte Carlo sledování cesty

Dalším problematickým jevem u této metody jsou kaustiky. Metoda je schopna vypočítat kaustiky přesně, ale za cenu obrovského výpočetního výkonu. Důvod je nasnadě. Pravděpodobnost, že zasáhne paprsek místo kde vzniká kaustika, je poměrně malá, daleko menší je však pravděpodobnost, že paprsek bude odražen do zrcadlového povrchu a pak do zdroje světla, případně do transparentního povrchu a skrz něj poputuje do zdroje světla. V případě bodového světla je tato pravděpodobnost nulová, a proto nelze metodu sledování cesty použít pro výpočet kaustik způsobených bodovými zdroji, ale pouze plošnými.

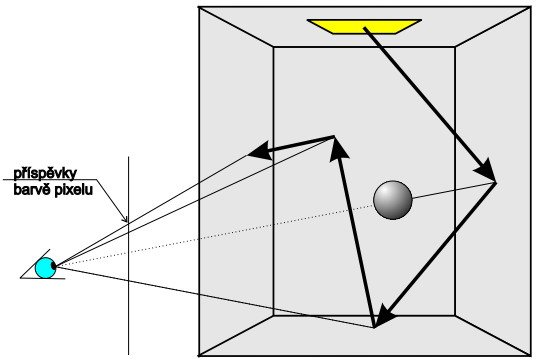
15.6 Metody vycházející od světelného zdroje

Metody, které začínají sledování trajektorie světla u pozorovatele, nejsou vhodné pro práci s malými, či skrytými světelnými zdroji. Rovněž je pro ně obtížný výpočet kaustik. Metody vycházející od světelných zdrojů tyto problémy odstraňují, jsou však zatíženy silnějším šumem. Metody vycházející od světelného zdroje se také označují jako metody vystřelující energii (*shooting methods*).

Tyto algoritmy v obecném případě hledají řešení osvětlovací rovnice náhodným sledováním všech drah vycházejících ze světelného zdroje (viz obrázek 15.9).

Generický algoritmus 15.4 všech metod vycházejících od světelného zdroje začíná vysláním

paprsku jdoucího od světelného zdroje směrem do scény. Nejprve se určí výkon odcházejího paprsku a poté se rekurzívně sleduje jeho trajektorie ve scéně. V každém průsečíku se vypočítá osvětlení bodu a zároveň se určí množství světla jdoucí směrem k pozorovateli.



Obrázek 15.9: Podstata metody Monte Carlo vycházející od zdroje světla

Poslední zmíněný krok je důležitý, neboť z tohoto paprsku se určí příspěvek k barvě pixelu. Vzhledem k tomu, že radiance se šíří po přímce, určí se průsečík paprsku s promítací rovinou, tedy s pixelem na obrazovce a tedy i příspěvek k jeho barvě. Každý pixel má čítač hodnot příchozí radiance a v konečné fázi se vyčíslí průměr z těchto příspěvků, jako součet hodnot dělený počtem příspěvků. Po výpočtu lokálního osvětlení se vypočítá odražený a (případně) prostupující paprsek a pokračuje se ve sledování. Monte Carlo sledování cesty se vyjadřuje v termínech radiance, metody vycházející od světelných zdrojů používají světelný výkon Φ (10.2). Způsob šíření světla je popsán geometrickou optikou a je v podstatě identický.

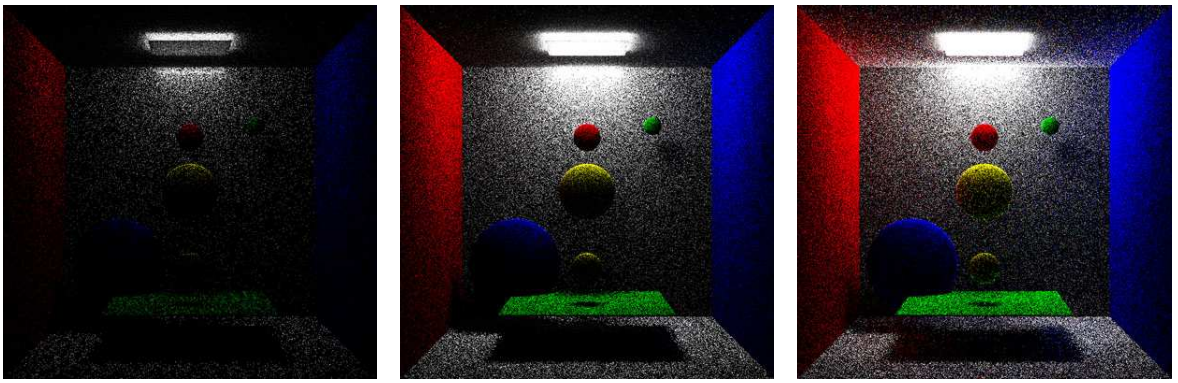
Jedním z problémů těchto metod je potenciální vyčíslení velkého množství paprsků, které se možná ani nevyužijí, neboť nedorazí k pozorovateli. Příkladem je opět scéna z obrázku 15.8. Pro tyto druhy scén jsou však metody vycházející od zdroje světla výhodnější.

Stejně jako v předcházejícím případě je otázkou, jak je implementována rekurzivní metoda, v případě těchto metod se jmenuje Sleduj(paprsek,výkon).

1. Smaž obraz
2. Pro i od nuly do počtu vzorků n
   (a) Vypočti náhodný paprsek jdoucí ze zdroje světla a urči jeho výkon
   (b) Sleduj(paprsek,výkon)

Algoritmus 15.4: Generický algoritmus metod vycházejících od zdroje světla

Obecným problémem všech metod vycházejících od světelných zdrojů je velký šum. Metoda totiž ze své podstaty neovlivňuje počet vzorků, které přispívají pixelu. Nezbývá než generovat velké množství paprsků a čekat, zda jejich příspěvky ovlivní pixel či ne. Obrázek 15.10 ukazuje, jak konverguje řešení osvětlovací rovnice metodou sledování světla.



Obrázek 15.10: Šum metody sledování světla. Obrázky byly vypočteny s následujícím počtem paprsků emitovaných ze zdroje světla: 10k, 100k, 1M vzorků. Šum metody je výrazný i pro velký počet vzorků.

15.6.1 Sledování fotonů

Metoda zvaná sledování fotonů (*photon tracing*) je duální metodou k metodě sledování paprsku. Používá stejné druhy odrazů, vyšetřování se však provádí v opačném směru. Procedura Sleduj(paprsek,výkon) je implementována pouze pro zrcadlové odrazy a průhledné objekty. Tato metoda nesimuluje difúzní odrazy a tak vypočítává trajektorie LS*DE tj. libovolný počet zrcadlových odrazů a lokální osvětlovací model v každém průsečíku. Metoda sledování fotonů *není* metodou fotonových map, kterou zmiňujeme v odstavci 15.7.2. Význam metody sledování fotonů je malý a uvádíme ji zde pouze pro úplnost.

15.6.2 Monte Carlo sledování světla

Jako je předcházející metoda duální k metodě sledování paprsku, je metoda Monte Carlo sledování světla (*light tracing*) komplementární k Monte Carlo sledování cesty. Princip popisuje algoritmus 15.5.

Paprsek vycházející ze světelného zdroje či odražený od objektu a nesoucí určitý výkon dopadne na nějaký bod na povrchu tělesa. Na řádku 3 se vypočítá, zda je tento bod viditelný z polohy pozorovatele. V kladném případě se určí pixel, který bude zasažen úsečkou definovanou bodem x a průsečíkem s objektem. K barvě pixelu se přičte hodnota barvy paprsku odraženého směrem k pozorovateli vynásobená hodnotou $1/n$, kde n je počet příspěvků k tomuto pixelu. Následně se vzorkuje BRDF v bodě odrazu a určí se útlum výkonu v nově vypočteném směru. Je-li příspěvek paprsku malý, sledování se ukončí (řádek 5), pokud má paprsek pro další výpočet

Sleduj(paprsek,výkon)

1. (objekt,x)=PrvníPrůsečík(paprsek)
2. není-li zasažen objekt skonči
3. je-li průsečík x viditelný, najdi pixel p ke kterému přispívá a uprav hodnotu jeho barvy barva[p]+=výkon(paprsek)/n
4. Vypočti odražený paprsek_r vzorkováním BRDF
5. je-li intenzita paprsku malá skonči
6. výkon paprsku=výkon/pravděpodobnost
7. Sleduj(paprsek,výkon)
8. je-li materiál průhledný vypočti zalomený paprsek_t a jeho výkon
9. Sleduj(paprsek_t,výkon)

Algoritmus 15.5: Procedura realizující Monte Carlo sledování světla

nějaký význam, sleduje se jeho dráha scénou dál. Poslední dva řádky programu sledují cestu paprsku po jeho lomu na průhledných tělesech.

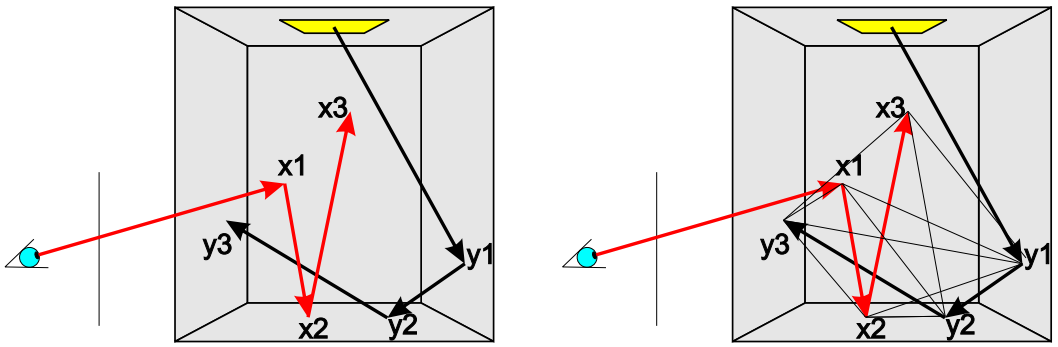
15.7 Dvousměrové metody

Dvousměrové metody (*bidirectional methods*) odstraňují nevýhody předchozích metod a spojují v sobě jejich výhody. Tyto metody sledují najednou dvě cesty, jednu od světelného zdroje a druhou od pozorovatele. Cesta jdoucí od světelného zdroje je výhodná pro výpočet kaustik a zaručuje rychlejší konvergenci k řešení ve světelně obtížných scénách. Generování paprsku od pozorovatele zase snižuje šum a umožňuje snažší řízení přesnosti metody generováním požadovaného počtu paprsků jdoucích pixelem.

15.7.1 Dvousměrové sledování cesty

Dvousměrové sledování cesty (*bidirectional path tracing*) [Lafo93] je jednou z metod, které generují v dostatečně krátkém čase kvalitní výsledky prakticky všech požadovaných optických jevů i ve stížených optických podmínkách. Její princip je naznačen na obrázku 15.11. Metoda generuje najednou dvě cesty, dráha směrem od pozorovatele je označena průsečíky $x1$, $x2$ a $x3$. Dráha generovaná od zdroje světla je v obrázku označena průsečíky $y1$, $y2$ a $y3$. Poté co jsou obě cesty určeny, spojí se tzv. deterministickým krokem poslední průsečíky obou cest $x3$ a $y3$. Následně se určí viditelnost a vzájemné příspěvky mezi všemi kombinacemi xi a yj

jak naznačuje obrázek 15.11 vpravo (každý z průsečíků xi se musí spojit se všemi yj). Pro zjednodušení jsme do obrázku nezakreslili stínovací paprsky spojující zdroj světla a průsečíky dráhy od pozorovatele.



Obrázek 15.11: Dvousměrové sledování cesty. Nejprve se sleduje cesta od zdroje světla a od pozorovatele (vlevo), poté se spojí obě cesty deterministickými kroky (vpravo).

Vzájemné energetické příspěvky se musí převést na odpovídající veličiny. Směrem od pozorovatele vyšetřujeme radiance, od světelného zdroje výkon. Poté je k dispozici radiance pro pixel, kterým prochází paprsek jdoucí od pozorovatele a zároveň všechny pixely, které jsou viditelné ze všech průsečíků paprsků jdoucích od světelného zdroje. Tím se spojují výhody obou metod.

Jednou z nevýhod této metody je obtížný výpočet kaustik, které, zvláště v obtížných světelných podmínkách, můžou být zašuměné.

15.7.2 Fotonové mapy

Fotonové mapy [Jens01] jsou kompromisem mezi metodami Monte Carlo a mezi metodami konečných prvků. Není zapotřebí žádné rozdělení scény na části, a proto se nejedná o metodu konečných prvků, na druhou stranu tato metoda používá pomocnou datovou strukturu, a tak jsou její paměťové nároky vyšší, než jsou nároky ostatních metod Monte Carlo.

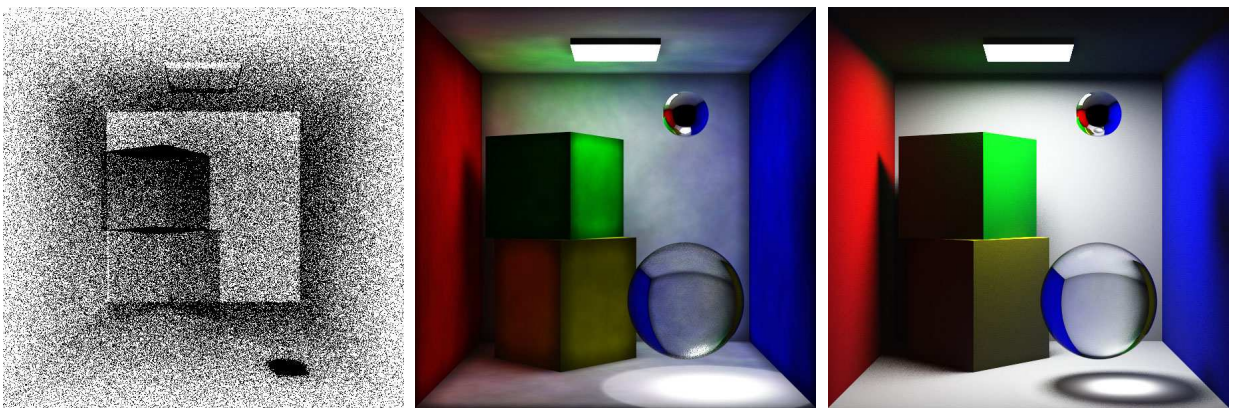
Výpočet globálního osvětlení pomocí fotonových map je dvousměrný a pracuje ve dvou krocích. V prvním kroku, vystřelovacím (*shooting*), se vyzáří ze všech světelných zdrojů velké množství fotonů. Každý foton nese část energie světelného zdroje a může se několikrát ve scéně odrazit. Fotony jsou vyzářeny z různých světelných zdrojů a ukládají se do trojrozměrné datové struktury, fotonové mapy, pro pozdější výpočet.

Ve druhém kroku se provede obyčejné sledování paprsku metodou Monte Carlo s tím, že se využívají uložené fotonové mapy. Při každém průsečíku s objektem se naleznou nejbližší

fotony a z jejich hodnot se vypočítá příspěvek osvětlení od všech světelných zdrojů. Teoreticky je možné použít jakoukoli Monte Carlo metodu sledování dráhy.

Fotonové mapy se také mohou zobrazit přímo tak, jak ukazuje obrázek 15.12 uprostřed. Přímé zobrazení však zanáší spoustu barevných chyb z nichž nejdůležitější je falešný difúzní přenos barvy.

Zvláštní péče věnuje tato metoda kaustikám. Fotony přispívající ke kaustikám jsou uloženy ve zvlášťní datové struktuře. Při zobrazování kaustik se pro ně používá algoritmus, který vzorkuje oblast, kde se projevují kaustiky, s vyšší přesností [Jens01].



Obrázek 15.12: Vlevo jsou zobrazeny přímo fotony zachycené ve scéně, uprostřed je scéna zobrazená pouze pomocí uložených fotonů a vpravo je scéna zobrazena metodou fotonových map

15.8 Zrychlení stochastických metod vzorkování

Zvýšení kvality výsledného obrazu roste s funkcí $\sqrt{n}$, kde n je počet paprsků jdoucích pixelem. Existuje mnoho cest jak stochastické metody urychlit, jedním z nejjednodušších, ale velice efektivních, je tzv. *ruská ruleta*. Podstatou této metody je, že se odstraní nepodstatné paprsky a výpočet se přenese na ty, které přispívají k výsledku více.

V dalším textu budeme používat zápis z [Jens01]. Algoritmus 15.6 popisuje slovně uvedený postup. Předpokládejme, že pravděpodobnost odrazu světla do vyšetřovaného směru, tedy reflektance (10.7) v bodě, je ρ a příchozí radiance je L_i. Vygenerujeme náhodné číslo ξ s rovnoměrným rozložením $\xi \in \langle 0, 1 \rangle$ a pokud je toto číslo menší nežli ρ, odrazíme radianci s výstupní radiancí rovnou L_i a ne zmenšenou vynásobením koeficientem ρ. Jinými slovy řečeno,

odrazivost se neuplatní při zmenšení intenzity radiance, ale při rozhodnutí o dalším osudu paprsku. V opačném případě paprsek zanikne.

Jednoduché je vysvětlení příkladem [Jens01]. Předpokládejme zeď s odrazivostí 1/2 na kterou vyzáříme sto paprsků. Můžeme buď vypočítat sto odražených paprsků s poloviční intenzitou, nebo, v případě aplikace ruské rulety, odrazit pouze polovinu, avšak s plnou intenzitou. Celkový počet paprsků se tak podstatně sníží, zejména po odrazech od objektů s nízkou odrazivostí.

1. vygeneruj náhodné číslo $\xi \in \langle 0, 1 \rangle$ s rovnoměrným rozložením
2. je-li $\xi < \rho$ vypočti odražený paprsek a přiřaď mu energii L_i
3. jinak je paprsek zrušen

Algoritmus 15.6: Ruská ruleta

Jiné metody zrychlení stochastického vzorkování scény, zejména vzorkování podle důležitosti (*importance sampling*) a vzorkování po vrstvách (*stratified sampling*) jsou k nalezení v [Dutr03, Glas95, Jens01].

15.9 Sledování paprsku

Pojmy *sledování paprsku* (*ray tracing*) nebo *vržení paprsku* (*ray casting*) pocházejí z optiky, kde se takto nazývané techniky používají pro určení vlastností optických systémů s čočkami. Jak plyne z fyzikální podstaty světla, paprsky se šíří od světelných zdrojů různými směry. Některé z paprsků opustí prostor scény, jiné zasáhnou povrch objektů ve scéně. Podle optických vlastností těchto objektů se paprsky odrážejí, případně lomí, a ovlivňují osvětlení v dalších místech scény.

V počítačové grafice často modelujeme proces šíření světla obráceným postupem. Paprsek promítneme ze stanoviště pozorovatele skrz pomyslný pixel v rovině obrazovky (průmětny) směrem do scény. Každý takový paprsek považujeme za sondu, která prochází scénou a hledá odpověď na otázku: „Co je vidět v tomto pixelu?“ či lépe: „Jaké informace o světelné energii tento paprsek přináší?“ Proto se metoda také někdy nazývá *zpětné sledování paprsku*. V následujícím textu budeme místo pojmu *objekt* užívat raději *těleso*, abychom zdůraznili, že paprsek může procházet různým optickým prostředím, tedy i vnitřkem (poloprůhledných) těles. Rozlišujeme dvě základní varianty algoritmu sledování paprsku:

1. Vržení paprsku (Sledování paprsku prvního řádu) (*ray casting*)
   Při něm zobrazujeme pouze bod na povrchu nejbližšího tělesa zasaženého paprskem. V tomto bodě stanovíme barvu pomocí jednoduchého osvětlovacího modelu, např. Phongova (viz kap. 10.5). Vzhled obrázků vzniklých sledováním paprsku prvního řádu se příliš neliší od obrázků vytvořených dříve uvedenými, jednoduššími zobrazovacími metodami. Má však význam pro zobrazování modelů v CSG reprezentaci (viz kap. 6.3 a 15.9.2).
2. Sledování paprsku vyššího řádu
   Metodu poprvé uvedl Whitted v roce 1980 [Whit80]. Sledování paprsku nekončí po nalezení nejbližšího tělesa, ale pokračuje sledováním dalších paprsků, odvozených podle odrazivosti a průhlednosti nalezeného tělesa. V tomto případě opět vidíme povrch nejbližšího tělesa, ale na jeho barvě se podílejí i světelné příspěvky zjištěné ostatními paprsky. Postup popisuje rekurzivní algoritmus 15.7, který se spouští s parametrem hloubky rekurze rovným nule.

SledujPaprsek (paprsek R, hloubka rekurze H)

1. Nalezni průsečík P paprsku R s nejbližším tělesem ve scéně
2. Pokud průsečík P neexistuje (paprsek opustil prostor scény), přiřaď paprsku R barvu pozadí a skonči
3. Ke každému světelnému zdroji vyšli z bodu P stínový paprsek a pokud k němu paprsek dorazí, označ světelný zdroj jako nezakrytý
4. Vyhodnoť příspěvky osvětlení v bodě P od všech nezakrytých světelných zdrojů
5. Pokud hloubka H nepřekročila maximální hloubku sledování, vyšli
   (a) odražený paprsek R_R voláním SledujPaprsek (R_R, $H + 1$)
   (b) lomený paprsek R_T voláním SledujPaprsek (R_T, $H + 1$)
6. Paprsku R přiřaď výslednou barvu jako součet příspěvků osvětlení, barvy odraženého paprsku R_R a barvy lomeného paprsku R_T

Algoritmus 15.7: Sledování paprsku vyššího řádu

Metoda sledování paprsku vyššího řádu dokáže zobrazit na povrchu tělesa zrcadlové obrazy jiných těles. Je také schopna zpracovat vržené stíny. Pro jednotlivé paprsky, pohybující se scénou, se zavedlo následující označení.

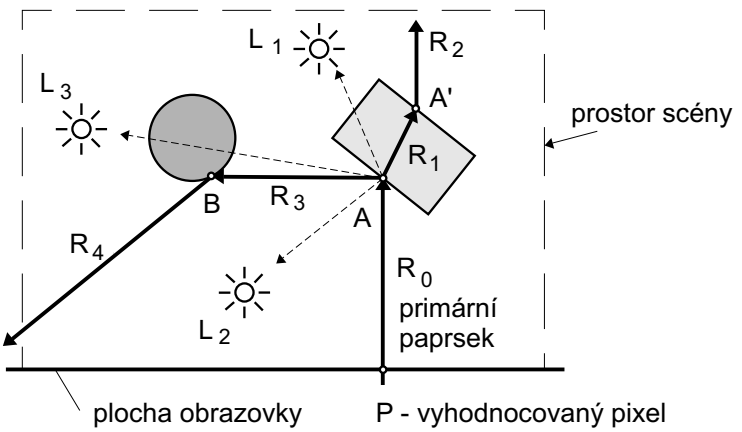
Primární paprsek (*primary ray*) je vysílán z místa pozorovatele bodem obrazu (pixelem).

Sekundární paprsek (*secondary ray*) je vytvořen po dopadu primárního nebo sekundárního paprsku na těleso. Reprezentuje stav, kdy se předchozí paprsek na povrchu tělesa odrazil zpět

do prostoru scény, nebo kdy pronikl do vnitřku poloprůhledného tělesa. V daném bodě tedy můžeme tedy počítat až se dvěma sekundárními paprsky, *odraženým* (*reflected*) a *lomeným* (*transmitted*). Počet sekundárních paprsků je při větší hloubce rekurze mnohem vyšší, než počet primárních, protože každý sekundární paprsek může po dopadu na další těleso způsobit vznik nových dvojic sekundárních paprsků.

Stínový paprsek (*shadow ray*) je vysílán z bodu, kam dopadl primární nebo sekundární paprsek ke světelnému zdroji. Jeho úkolem je zjistit, zda mezi tímto bodem a konkrétním světelným zdrojem není nějaká překážka zastiňující těleso. Pokud není žádná překážka nalezena, je tento světelný zdroj zahrnut do vyhodnocení osvětlovacího modelu v daném bodě. Oproti primárním a sekundárním paprskům jsou výpočty spojené se stínovými paprsky jednodušší, protože není třeba hledat nejbližší těleso na dráze paprsku. Výpočet je možné přerušit po nalezení libovolného tělesa na spojnici bod–zdroj světla. Z každého místa dopadu primárního a sekundárního paprsku je vysláno tolik stínových paprsků, kolik je zdrojů světla.

Na obr. 15.13 je naznačen typický postup určení barvy jednoho primárního paprsku. Paprsek R_0 prochází pixelem obrazovky a míří do prostoru scény. V ní nalezneme nejbližší průsečík na tělese A. Z tohoto místa vyšleme tři stínové paprsky, abychom mohli vyhodnotit osvětlení, a dva sekundární paprsky ve směru odrazu a lomu paprsku primárního.



Obrázek 15.13: Sledování jednoho primárního paprsku má za následek vznik mnoha sekundárních a stínových paprsků

Ve směru lomeného paprsku R_1 projdeme poloprůhledným tělesem A a v místě opuštění tělesa opět vyhodnotíme osvětlení. Sledování podruhé lomeného paprsku R_2 je ukončeno při opuštění scény. Na dráze v odraženém směru R_3 nalezneme neprůhledné těleso B, na němž také vyhodnotíme osvětlení. Od tohoto tělesa sledujeme paprsek R_4, na jehož dráze nenalezneme žádné objekty a sledování ukončíme.

Pokud by u některého ze sekundárních paprsků docházelo k mnohokrát opakovanému odrazu od těles, je výpočet ukončován podmínkou povolené hloubky rekurze. Ta bývá nastavena na 3 až 5. Větší hloubka příliš zpomaluje výpočty a navíc většinou nepřináší znatelné vylepšení výsledného obrazu.

Po vyhodnocení stínových paprsků zjistíme, že bod na tělese A bude ležet v částečném

stínu, přímo osvětlen pouze zdrojem světla L_2 a částečně zdrojem L_1. Osvětlení ze zdroje L_1 je sporné, protože ve skutečnosti tento zdroj neosvětluje daný bod přímo, ale světlo je při průchodu tělesem A lámáno. Většina implementací metody sledování paprsku problém lomu stínového paprsku není schopna řešit, a proto jej zanedbává. Zde se projevuje zásadní nedostatek algoritmu – zpětným sledováním paprsků nelze z principu nalézt všechny paprsky přispívající k osvětlení určitého bodu. Také pokud by ve scéně bylo zrcadlo odrážející světlo na nějaký předmět (vržení „prasátka“), museli bychom vést zvláštní stínový paprsek i k zrcadlu coby k sekundárnímu zdroji světla.

Pomocí stínových paprsků nalezneme vržené stíny a pomocí sekundárních paprsků jsou vykresleny odražené obrazy na povrchu těles. Paprsky mají různé počátky a ve scéně se pohybují různými směry. To je hlavním důvodem, proč je obtížné efektivně implementovat metodu sledování paprsku a proč nelze využít jednoduché postupy známé z úloh řešení viditelnosti, jako např. vyloučení zadních stěn těles či omezení rozsahu scény ořezáním pohledovým objemem. Hledání průsečíku paprsku s nejbližším tělesem přitom zabírá v případě základního algoritmu více než 90 % času.

15.9.1 Rozšíření Phongova osvětlovacího modelu

Osvětlovací model, používaný pro výpočet barvy paprsku dopadlého do určitého bodu na povrchu tělesa (část 10.5), je v algoritmu sledování paprsku upraven na následující tvar:

$$I_V = I_s + I_d + I_a + I_r + I_t. \tag{15.5}$$

V něm jsou obsaženy známé složky pro zrcadlové (I_s), difúzní (I_d) a okolní (I_a) osvětlení, týkající se pouze světla ze světelných zdrojů (viz vzorec [10.14)]. Připomeňme, že v metodě sledování paprsku se konkrétní zdroj světla do výpočtu zahrnuje pouze tehdy, dorazí-li k němu stínový paprsek.

Další složky ve vzorci (15.5) reprezentují paprsky odražené a lomené. Barva I_R paprsku přicházejícího ze směru odrazu je ovlivňována koeficientem zrcadlového odrazu r_s tělesa, který se také objevuje ve vzorci (10.11) pro výpočet zrcadlové složky osvětlení. Platí tedy

$$I_r = r_s I_R. \tag{15.6}$$

Podobně je ovlivňována barva I_T paprsku přicházejícího ze směru lomu, která navíc může být snižována dalším koeficientem, charakterizujícím útlum a závisejícím na vzdálenosti, kterou paprsek uvnitř tělesa urazí (blíže viz kap. 10.4.2).

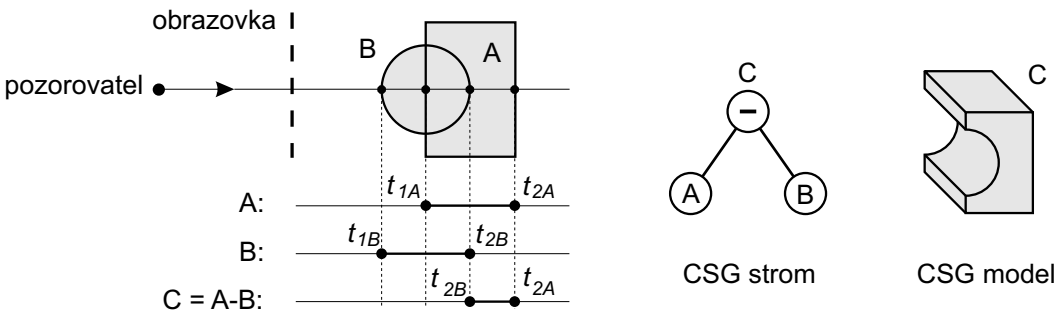
Z uvedených principů metody sledování paprsku vyplývají nedostatky, které lze odstranit jen velmi obtížně, a to za cenu vysokých výpočetních nároků:

- vržené stíny jsou ostré;
- osvětlovací model předpokládá pouze bodové zdroje světla, ne však plošné,
- zrcadla umístěná do scény sice odrážejí obraz okolních těles, nejsou však využita pro odraz světla, tedy pro určení nepřímého osvětlení,
- při změně polohy pozorovatele nebo při přidání nového tělesa do scény se musí provádět znovu celý náročný výpočet obrázku,
- i na velké nelesknoucí se plochy je používána stále stejná metoda, ačkoliv by takové plochy mohly být stejně kvalitně zobrazovány jednoduššími stínovacími algoritmy.

15.9.2 Sledování paprsku a CSG reprezentace

Metoda sledování paprsku je schopna zobrazit objekty definované v různých reprezentacích. Základním požadavkem je nalezení průsečíku polopřímky (paprsku) s objektem. Je-li objekt reprezentován jako CSG model, je určení průsečíku složitější.

Paprsek vedený pixelem není v případě CSG modelu určen pouze k nalezení nejbližšího (primitivního) tělesa, ale všechny jeho průsečíky s primitivními tělesy jsou uchovávány k dalšímu vyhodnocení. Průnik paprsku s konvexním primitivním CSG tělesem leží v intervalu parametru t polopřímky určené paprskem. Intervaly vymezují úseky paprsku, které jsou dále podrobovány těm množinovým operacím, které jsou obsaženy v uzlech CSG stromu. Výsledný úsek určuje viditelnou část CSG modelu.



Obrázek 15.14: Vyhodnocení úseků na paprsku, který protíná CSG strom

Uvedený postup je ukázán na obr. 15.14, kde je zjišťována viditelnost CSG modelu vytvořeného rozdílem dvou základních těles. Od kvádru A je odečten válec B. Kořen CSG stromu definuje tvar výsledného tělesa C. Na obrázku je naznačen jeden z vržených paprsků, který protíná obě základní tělesa. Výsledkem průniku paprsku se základními tělesy jsou dva intervaly parametru t: $\langle t_{1A}, t_{2A}\rangle$ a $\langle t_{1B}, t_{2B}\rangle$. Tyto intervaly vstupují do algoritmu vyhodnocování CSG

stromu zdola nahoru. Po provedení operace rozdílu v kořeni stromu docházíme k výslednému intervalu $\langle t_{2B}, t_{2A} \rangle$, který definuje úsek, ve kterém paprsek prochází skutečným tělesem z kořene CSG stromu. Blíže k pozorovateli je bod určený parametrem t_{2B}, jehož barvu po vyhodnocení osvětlovacího modelu vykreslíme. Protože víme, že bod patří tělesu B, můžeme stanovit i úhel, pod kterým paprsek v tomto bodě dopadá na povrch tělesa B, resp. můžeme vypočítat normálu povrchové plochy v tomto místě a případně pokračovat ve sledování paprsku vyššího řádu.

Je výhodné rozdělit prostor obsahující CSG model pomocí oktalového stromu, jak bylo ukázáno na obrázku 6.12 v části 6.3, a v jednotlivých podprostorech CSG model zjednodušit (prořezat). Jiný způsob snižování objemu výpočtů je založen na procházení CSG stromu od kořene zleva doprava s ohledem na množinové operace ve vnitřních uzlech. Vyhodnocujeme-li například operaci rozdíl a v levém podstromu neprotnul paprsek žádné těleso, není již třeba testovat žádný objekt v pravém podstromu.

15.9.3 Urychlování metody sledování paprsku

Způsobům urychlení sledování paprsku je věnováno mnoho úsilí, neboť výpočty v původní, nezměněné podobě algoritmu jsou časově velmi náročné. Pomocí urychlovacích technik lze přitom výpočty urychlit o dva až tři řády.

Arvo a Kirk [Arvo89] rozdělili urychlovací metody do několika tříd:

1. Urychlení výpočtu průsečíků
   (a) Rychlejší výpočet průsečíků paprsek–objekt
   (speciální výpočty pro jednotlivé objekty, jednoduché obálky)
   (b) Méně výpočtů průsečíků paprsek–objekt
   (hierarchie obálek, dělení scény, paměť překážek, koherence paprsků)
2. Snížení počtu paprsků
   (adaptivní vyhlazování, adaptivní řízení hloubky rekurze)
3. Sledování více paprsků naráz
   (svazky paprsků)

Tyto urychlovací techniky se často kombinují. Následující odstavce je zběžně představí.

Speciální výpočty pro jednotlivé objekty

Pro metodu sledování paprsku bylo vyvinuto mnoho tzv. *vyhodnocovačů průsečíků* (*intersectors*), což jsou rychlé podprogramy pro výpočet průsečíku paprsku s určitou třídou objektů. Jejich důležitou částí jsou tzv. zásah/mimo (*hit/miss*) testy, které vyloučí objekt z dalšího výpočtu dříve, než dojde ke skutečnému analytickému vyhodnocování možného průsečíku. To je obzvláště důležité pro komplexní objekty, jako jsou parametrické plochy či fraktály. Ovšem

i pro tak jednoduché těleso, jakým je koule, lze nalézt zrychlující test – dříve, než začneme řešit kvadratickou rovnici pro výpočet průsečíků přímky a koule, lze pomocí několika násobení a jednoho porovnání zjistit, zda paprsek kouli zcela nemine (viz část 22.4.7).

Do této třídy urychlovacích metod řadíme i jednoduché „triky“ založené na transformaci objektů do základní polohy. Příkladem je hledání průsečíku paprsku s mnohoúhelníkem. Namísto řešení v trojrozměrném prostoru je mnohoúhelník otočen do roviny rovnoběžné s rovinou xy. Pro průsečík analogicky otočeného paprsku s touto rovinou snadno určíme metodami rovinné grafiky (geometrie), zda leží uvnitř nebo vně mnohoúhelníku. Rychlé podprogramy pro rozličná primitivní tělesa lze nalézt v [Glas89, Glas90, Arvo91].

Obálky a prostorové hierarchie

Obálky, které jsou podrobněji popsány v kapitole 14, jsou jednou ze základních urychlovacích technik. Připomeňme, že tvar obálky může být rozmanitý (kvádr, koule, rovnoběžnostěn), princip použití však zůstává stejný – provedení testu s obálkou je výrazně rychlejší, než provedení testu se skutečným tělesem. Pokud je výsledek testu negativní (paprsek obálku neprotnul) není třeba se tělesem dále zabývat.

Jsou-li obálky uspořádány hierarchicky (část 14.2.1), úspora času se výrazněji projeví při vyhodnocování sekundárních a stínových paprsků. Zatímco u primárních paprsků je třeba prohledávat celou hierarchickou strukturu od kořene stromu, u ostatních paprsků známe bod uvnitř scény, odkud vycházejí. Jejich sledování tedy může začít u naposledy zpracovaného uzlu stromu obálek a teprve postupně se rozšiřuje do vyšších úrovní stromu. Předpokladem pro tento postup je, že uzly obsahují odkazy na své předchůdce.

Mnohem častěji než hierarchie obálek se pro urychlení sledování paprsku používají shora budované hierarchie, zejména kD-stromy a oktalové stromy. Jejich stavba a vlastnosti jsou popsány v části 14, zde jen uvedeme některé informace specifické pro metodu sledování paprsku.

Oktalový strom pro tuto metodu poprvé použil A. Fujimoto [Fuji85, Fuji86] spolu se speciálním traverzačním algoritmem nazvaným 3DDDA (*3-Dimensional Digital Difference Analyzer*). Tento algoritmus je schopen nalézt všechny prostorové buňky, kterými paprsek prochází, a to s ohledem na jejich různé velikosti. Je zobecněním původního algoritmu DDA používaného při rasterizaci úsečky (kap. 3.1.1) s tím, že délka kroku je proměnlivá a aktualizuje se podle velikosti voxelu.

Podobně jako u hierarchie obálek se výhoda oktalového stromu projeví u sekundárních a stínových paprsků, u nichž není třeba traverzovat celý oktalový strom od kořene až k listům, ale postup je možno zahájit v naposledy zpracovávaném listu. Z hlediska implementace není nutno oktalový strom reprezentovat jako běžnou stromovou strukturu, tvořenou uzly s osmi ukazateli na následníky. Takové řešení totiž nezajistí snadné nalezení sousedních oktantů, pokud tyto oktanty nemají stejného předchůdce. Jedna z možností je reprezentovat strom jako

tabulku, do které se přistupuje podle klíče, určeného přepočtem ze souřadnic libovolného bodu v prostoru scény [Fuji85]. Prázdná položka tabulky indikuje prázdný oktant. Dalším řešením je provázání sousedních oktantů pomocnými ukazateli, zvanými provazy (*rope*), jak ukazuje obr. 14.10 v části 14.2. Tak je umožněno kromě vertikálního i horizontální procházení stromu.

Značné výzkumné úsilí bylo věnováno použití kD-stromů a BSP stromů pro zrychlení metody sledování paprsku [Havr03]. Zejména BSP stromy vykazují řadu dobrých vlastností, např. dělicí rovinu lze velmi dobře umístit s ohledem na rozmístění objektů ve scéně, rovinné poloprostory se jednoduše testují, apod. Snaha nalézt optimální dělení prostoru, které by pro danou pozici pozorovatele a polohu průmětny minimalizovalo jak celkový počet traverzačních kroků v BSP stromu, tak počet zbytečných testů na průsečík (např. s tělesy, před nimiž je jiné těleso – překážka, o kterou se paprsek zarazí), však dosud nebyla korunována úspěchem. Důvodem je jak vysoká složitost obecných scén, tak obtížnost odhadu nejvýznamnějších směrů paprsků. Přesto byla tato myšlenka podnětem pro vývoj řady zlepšení týkajících se především fáze stavby BSP stromu. Pro volbu dělicí roviny se používají heuristiky zohledňující pravděpodobnost, že objekt (resp. jeho obálka) dané velikosti bude zasažen paprskem z určitého směru [Havr00]. Některé metody chápou prázdný prostor ve scéně jako speciální těleso, které je třeba odstranit (odříznout) při stavbě stromu co nejdříve. Tím způsobem se při procházení stromu docílí rychlého „přeskoku“ volného prostoru a paprsek se dostane ke skutečným tělesům rychleji, než kdyby byl volný prostor rozdělen mnoha rovinami, které byly určeny polohou jiných těles.

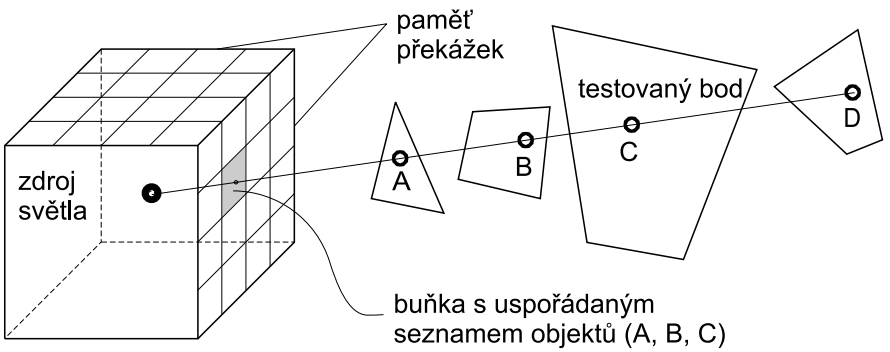
Paměť překážek

Vysoké množství stínových paprsků se stalo podnětem pro zavedení *paměti překážek*, která je v anglické literatuře známa jako *light buffer* [Hain86]. Každý bodový zdroj světla se obklopí krychlí pokrytou pravidelnou (obr. 15.15) nebo adaptivně dělenou sítí. Tato kostka se nazývá paměť překážek.

Při předzpracování scény se do každé buňky této sítě uloží seznam objektů, jejichž perspektivní průmět na stěnu krychle alespoň částečně pokryje danou buňku. Seznam je vzestupně uspořádán dle vzdálenosti ploch od světelného zdroje a je dále optimalizován. Nachází-li se v seznamu objekt, jehož průmět pokrývá celý povrch buňky, můžeme odstranit ze seznamu všechny vzdálenější objekty. To je případ plochy C na obr. 15.15, která pokryje celou buňku a způsobí vyřazení objektu D ze seznamu. Je-li naopak seznam tvořen pouze jediným objektem, lze seznam zcela zrušit, neboť daný objekt jistě nebude stát v cestě žádnému stínovému paprsku vyslanému z jiného objektu.

Při vyhodnocování stínového paprsku, mířícího z libovolného tělesa T ke zdroji světla, je nejprve nalezena buňka v paměti překážek, kterou paprsek protíná. Dále pak platí:

- je-li buňka prázdná, těleso T je tímto zdrojem osvětleno,



Obrázek 15.15: Paměť překážek a jedna z buněk obsahující seznam promítnutých ploch

- není-li těleso T uvedeno v seznamu překážek dané buňky, pak těleso T leží ve stínu, tj. za všemi překážkami,
- je-li těleso T v seznamu překážek, je třeba ověřit průsečíky stínového paprsku s těmi překážkami, které se nacházejí v seznamu před tělesem T.

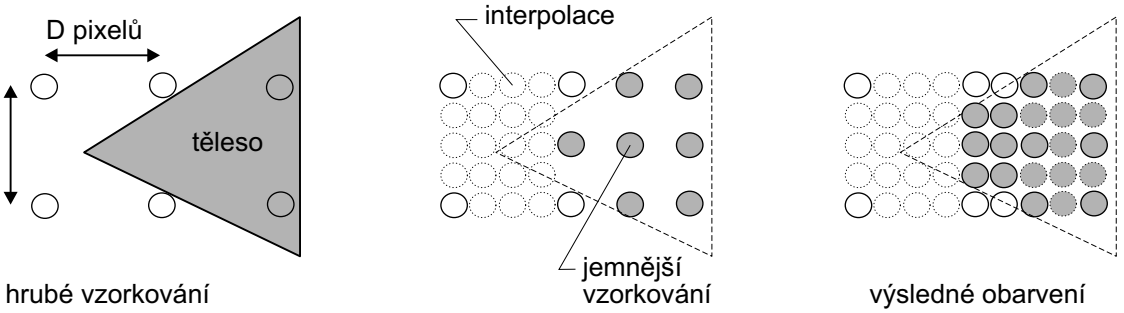
Předzpracování scény a vytvoření paměti překážek je značně náročné. Mnoho programů proto nahrazuje paměť překážek jedinou proměnnou pro každý zdroj světla, do které se průběžně ukládá odkaz na to těleso, které bylo překážkou naposledy vyhodnocovanému stínovému paprsku k tomuto zdroji světla. I tato jednoduchá myšlenka přináší výraznou úsporu času neboť je pravděpodobné, že vyšetřovaný stínový paprsek se nebude příliš lišit od předchozího a zasáhne ve většině případů stejnou překážku.

Koherence paprsků

Stejně jako v jiných oblastech počítačové grafiky, také při sledování paprsku je výhodné využívat koherence, tedy informací získaných při předchozích výpočtech. V literatuře můžeme nalézt různé přístupy využívající koherenci paprsků [Spee85, Hanr86]. Některé z nich ukládají a znovu využívají informace o všech tělesech nalezených při rekurzivním vyhodnocování jednoho primárního paprsku (tzv. strom rekurzivního sledování).

Koherence lze využít i pro adaptivní antialiasing. Předpokládáme, že v obrazu existují oblasti, ve kterých sousední pixely mají stejnou nebo podobnou barvu. Můžeme proto vyslat do scény paprsky jen některými pixely a podle výsledku vybarvit oblast mezi těmito pixely přímo, bez sledování paprsku.

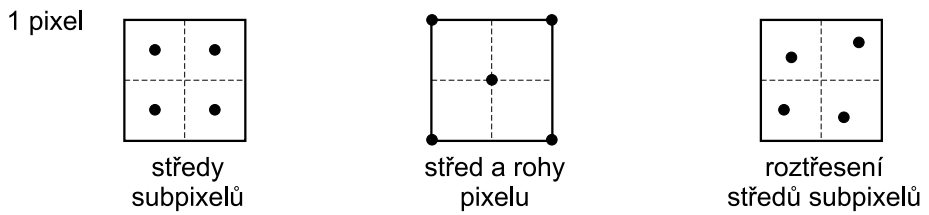
Postup je ukázán na obrázku 15.16. Zpočátku se vysílají paprsky velmi řídce, pouze do rohových pixelů oblastí o rozměrech $D \times D$ pixelů. Pamatujeme si přitom nejen výsledné



Obrázek 15.16: Sledování paprsku ve vybraných pixelech

barvy pixelů, ale i stromy určené vyhodnocením rekurzívně sledovaných paprsků v těchto pixelech. Mezilehlé pixely jsou dále vyhodnocovány sledováním paprsku pouze v případě, že barva a stromy jejich vyhodnocených sousedů se liší o určitou mez. Jinak jsou hodnoty pixelů stanoveny pomocí interpolace barvy. Tato metoda je nazývána *Pixel Selected Ray Tracing* (PSRT) a je založena na záměrném snížení vzorkovací frekvence.

Ačkoliv je výsledný obrázek po interpolaci barev zatížen chybou a také ztrátou detailů, jak je vidět na obrázku 15.16, metoda je použitelná např. při hledání správného pohledu na scénu, či nastavování vhodných koeficientů světelného modelu. Tehdy zajímá uživatele celkový vzhled scény a dává přednost rychlosti vytvoření náhledu obrazu před dlouho trvajícími výpočty dokonalého zobrazení. Po vytvoření náhledu je možné nechat výpočet dále pokračovat a postupně zpřesňovat zobrazení.



Obrázek 15.17: Různé možnosti vzorkování pixelu při sledování paprsku

Adaptivní chování metody lze využít i v opačném smyslu – ke zvýšení kvality obrazu vyšším vzorkováním výrazně odlišných sousedních pixelů (*adaptive antialiasing*). Pixel je považován za čtvercovou plochu, skrze kterou je vedeno několik primárních paprsků. Výsledná barva pixelu se získá jako průměr, případně vážený průměr z vypočítaných dílčích barev (viz též část 2.4.2).

Na obrázku 15.17 je uvedeno několik typických způsobů rozmístění paprsků vedených

vnitřkem pixelu. Při postupu naznačeném vlevo je pixel vzorkován čtyřmi paprsky. Při postupu uprostřed je jejich počet sice vyšší, ale zato se všechny rohové dají využít ve výpočtech sousedních pixelů. Je vhodné dodat středovému paprsku vyšší váhu. Pravý obrázek pak ukazuje náhodně *roztřesené* vzorkování, které obecně poskytuje nejlepší výsledky (část 2.3).

Řízení hloubky rekurze

Pokud scéna obsahuje mnoho matných, zrcadlově neodrážejících těles, je zbytečné i pro ně vypočítávat příspěvky odražených paprsků, a to např. až do hloubky rekurze 5. Namísto pevně dané maximální hloubky sledování je vhodné zavést limitní hodnotu příspěvku intenzity (barvy), při které je sledování zastaveno.

Do značné míry můžeme tento postup připodobnit k situaci, ve které paprsek postupně ztrácí informační hodnotu. Primární paprsek přináší plnou informaci o barvě a je proto ohodnocen informační intenzitou $[1, 1, 1]$ (ve smyslu barevných složek). Při každém odrazu je tato intenzita snižována vynásobením koeficientem zrcadlového odrazu r_s daného povrchu. Obdobně se zpracovává lom, kde lze vzít do úvahy i útlum prostředí, kterým paprsek prochází. Sekundární paprsky tak nesou nižší informační hodnotu a při jejím poklesu pod určitý práh se generování dalších sekundárních paprsků zastaví. Příspěvek jejich barvy k barvě primárního paprsku se považuje za zanedbatelný.

Takto řízená hloubka rekurze není vhodná pro scény, které obsahují mnoho zrcadlových povrchů. Tehdy průběžná aktualizace informační intenzity paprsku může způsobit i celkové zpomalení výpočtu.

Jak vyplývá z uvedeného přehledu metod, původně jednoduchý algoritmus sledování paprsku lze doplnit kombinací rozličných urychlovacích technik a vytvořit tak poměrně složitý, ale zároveň i velmi rychlý program. Obecně přinášejí největší urychlení metody založené na dělení scény. Je však dobré si uvědomit, že zvyšují nároky na paměť. Jejich efektivita roste se složitostí scény, rozsah pomocných datových struktur ovšem také.

Jako extrémně paměťově náročný uvedeme ještě jeden přístup, byť jeho cílem není pouze urychlení výpočtu jediného obrázku, ale především zajištění interaktivních změn barevných parametrů, textur a osvětlení scény. V této metodě je jednou vytvořený obraz doplněn úplnými informacemi o tom, která tělesa jsou v daném pohledu vidět a která další tělesa se na jejich povrchu odrážejí. Skupiny blízkých pixelů tak nesou společné údaje o celém stromu vzniklém při rekurzivním sledování paprsku. Při změně barvy či odrazivosti těles nebo při změně jasu světelných zdrojů se již neurčují žádné průsečíky s paprsky, pouze se vyhodnocují osvětlovací modely pro ty pixely, kterých se změna dotkla.

V předchozích odstavcích jsme se zabývali pouze geometrickými problémy (hledání průsečíků). Skutečné programy pro zobrazení scény metodou sledování paprsku zahrnují mnoho

dalších algoritmů pro manipulaci s texturami, realistické světelné zdroje, tvorbu měkkých stínů, efekty rozmazání části obrazu při pohybu objektu v čase (*blur*) a další vylepšení vedoucí ke zvýšení realističnosti výsledného obrazu. Všechny tyto prvky zvyšují složitost programu, jeho výpočetní a paměťové nároky. Metoda sledování paprsku stále zůstává aktuálním tématem výzkumu v oblasti prostorové grafiky

15.10 Radiozita

Metoda sledování paprsku, kterou jsme se zabývali v předchozí části, poskytuje kvalitní výsledky při modelování zrcadlových odrazů a při zobrazování průhledných objektů, avšak výsledné obrázky mají ještě daleko k fyzikálně věrné simulaci reálného světa. Klasická metoda sledování paprsku povoluje pouze bodové zdroje světla, k vyjádření nepřímého difúzního osvětlení používá pro scénu neměnný ambientní člen a celý mechanismus je empiricky odvozený. Pro dosažení fotorealistické věrnosti počítaných obrázků potřebujeme metodu, která dovolí z fyzikálního hlediska korektně simulovat šíření světla scénou.

Jeden z prvních pokusů o takovou simulaci podnikli Goralová, Torrance, Greenberg a další v polovině osmdesátých let [Gora84, Nish85]. Jimi navržená *metoda radiozity* aplikuje poznatky z oblasti výpočtů tepelného záření na problém výpočtu světelného záření. Základní radiozitní algoritmus vychází ze zákona zachování energie a předpokládá, že přenos světelného záření mezi objekty probíhá v energeticky uzavřené scéně. Přenos není ovlivněn prostředím (scéna se nalézá ve vakuu, resp. v prostředí, které netlumí procházející světlo), a všechny objekty jsou zcela neprůhledné a světlo se od nich odráží pouze difúzně (viz kap. 10.2). Objekty jsou popsány ploškovou hraniční reprezentací.

Postup při zobrazování scény metodou radiozity lze rozdělit na dvě části. Nejprve je vyhodnoceno šíření světla ze světelných zdrojů (v tomto případě plošných) a jeho odrazy na povrchu těles. Výsledkem tohoto výpočtu je ohodnocení ploch hodnotami osvětlení, které vyjadřují množství difúzně odraženého světla pro každou plochu. Tyto hodnoty nezávisí na poloze pozorovatele, jsou pouze vlastností scény. Proto lze ve druhé části metody použít libovolný zobrazovací algoritmus řešící viditelnost scény a zobrazovat scénu z různých pohledů bez nutnosti nových výpočtů difúzního odrazu světla.

15.10.1 Podstata metody

V dalším textu se budeme odvolávat na pojmy, zavedené v souvislosti se základní zobrazovací rovnicí. Připomeneme některé dříve uvedené veličiny:

- Radiozita $B(x)$ je světelný výkon vyzářený v bodě x vztažená na jednotkovou plochu.

Hodnoty radiozity jsou vždy kladné, jejich velikost není ale nijak omezena a záleží na osvětlení scény.

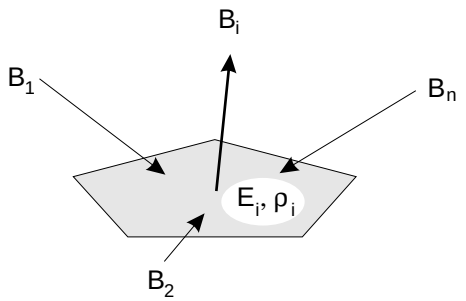
- Difúzní odrazivost $\rho(x)$ plochy v daném bodě udává, jaká část ze světelné energie, která na plochu dopadla, je opět vyzářena do prostoru. Difúzní odrazivost $\rho(x)$ nabývá hodnot mezi 0 (povrch veškerou dopadlou energii pohlcuje) a 1 (povrch veškerou dopadlou energii odráží zpět do scény).

Radiozitní rovnice

Radiozitní metoda je ve své podstatě speciální postup pro řešení základní zobrazovací rovnice v prostředí, kde všechny povrchy odráží světlo pouze difúzně. Pro takové plochy platí, že při změně stanoviště pozorovatele zůstává jejich vzhled stále stejný – hodnota osvětlovacího modelu je v každém bodu plochy konstantní, nezávislá na úhlu pohledu, a množství odražené světelné energie je stejné pro všechny směry. Lze dokázat, viz [Cohe93], že tyto podmínky umožňují zobrazovací rovnici zjednodušit a převést na tzv. *radiozitní rovnici*:

$$B(x) = E(x) + \rho(x) \int_S B(x')\ G(x, x')\ dx'. \tag{15.7}$$

Tato rovnice vyjadřuje, že radiozita $B(x)$ vyzařovaná povrchem v bodě x je součtem vlastní vyzářené radiozity $E(x)$ v tomto bodě a radiozity dopadlé v bodě x na povrch a odražené zpět do scény. Geometrický člen $G(x, x')$ zahrnuje geometrické informace týkající se vždy dvojice povrchů: jejich vzájemnou viditelnost, odchylky od normál obou povrchů a vzdálenost obou bodů. S značí množinu všech ploch scény.



Obrázek 15.18: Radiozita přijatá a vyzářená ploškou i

Diskrétní radiozitní rovnice

Analytické řešení radiozitní rovnice 15.7 je značně obtížné a pro netriviální scény nemožné. Z tohoto důvodu je nutné rovnici dále zjednodušit. Nahradíme-li povrch celé scény aproximací složenou z rovinných plošek a předpokládáme-li, že hodnota radiozity $B(x)$ každé plošky je všude na jejím povrchu konstantní, můžeme rovnici 15.7 přeformulovat na její diskrétní verzi, ve které je, za předpokladu $\sum B_j F_{ij} = \sum B_i F_{ji}$ integrál nahrazen součtem:

$$B_i = E_i + \rho_i \sum_{j=1}^{n} B_j F_{ij}, \tag{15.8}$$

F_{ij} značí tzv. *konfigurační faktor* (*form factor*), který v diskrétním tvaru radiozitní rovnice nahrazuje geometrický člen $G(x, x')$ a určuje vzájemnou viditelnost dvojice plošek. Pro plošky i a j o obsahu A_i, resp. A_j, udává konfigurační faktor F_{ij}, kolik z celkového výkonu vyzářeného ploškou j je přímo přijato ploškou i. Jeho hodnotu lze spočítat jako plošný průměr diferenciálních geometrických členů mezi všemi body obou testovaných plošek:

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} G(x, x') \; dA_i \; dA'_j. \tag{15.9}$$

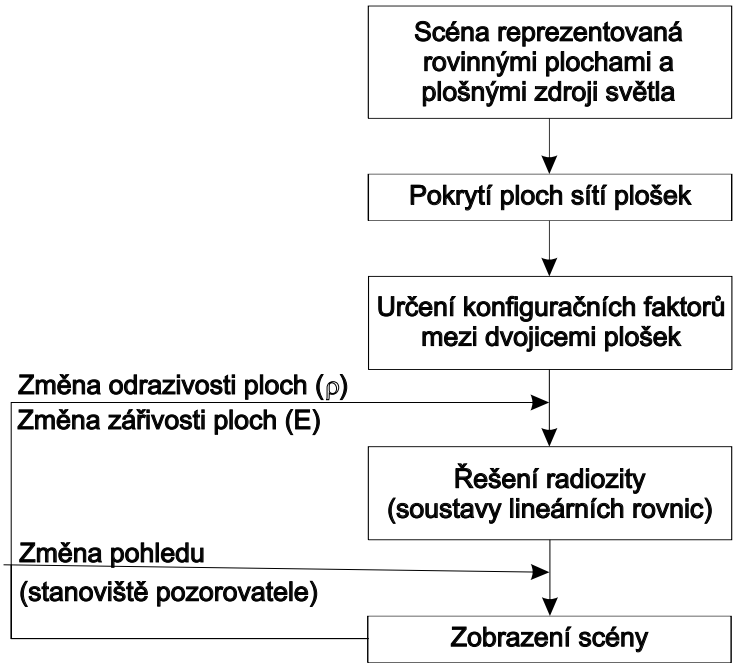
Diskrétní radiozitní rovnice 15.8 umožňuje vyjádřit radiozitu $i$té plošky scény pomocí radiozit všech ostatních plošek (viz. obrázek 15.18). Pro každou plochu scény získáváme jednu rovnici toho typu, což vede na soustavu rovnic, jejíž řešení určuje množství vyzařovaného světla pro každou plošku ve scéně.

15.10.2 Řešení radiozitní rovnice

Postup řešení radiozitní metody je schematicky znázorněn na obrázku 15.19. Vstupními daty metody je geometrie scény, tedy soustava ploch, doplněná informacemi o jejich difúzní odrazivosti ρ_i a zářivosti E_i. Rozdělení povrchu scény na síť rovinných plošek následují dva hlavní kroky výpočtu, které spočívají v určení konfiguračních faktorů a v řešení systému rovnic popisujícího rozdělení energie ve scéně. Výsledkem výpočtu je nalezení hodnoty radiozity pro každou plochu scény, které odpovídají rovnovážnému (ustálenému) stavu předávání světelné energie mezi tělesy v uzavřené scéně. V závěrečném kroku je získané řešení zobrazeno.

Rozdělení povrchu scény

Kvalita výsledných obrázků úzce souvisí s jemností a přesností konstrukce sítě plošek reprezentujících geometrii scény. Při jednoduché aproximaci povrchu velkými plochami se ve výsledném řešení osvětlení objevují chyby. Naopak velmi jemné sítě plošek sice zajišťují kvalitní řešení,



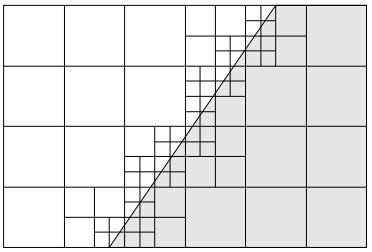
Obrázek 15.19: Schéma postupu při zobrazování pomocí radiozitní metody

ale neúměrným způsobem roste výpočetní čas a paměťové nároky. Cílem je tedy rozdělit scénu na co nejméně plošek a zároveň dosáhnou co nejvyšší přesnosti.

Uniformní metody rozdělují povrchy ve scény na pravidelné čtvercové nebo trojúhelníkové sítě plošek stejné velikosti. Výhodou tohoto přístupu je rychlost a jednoduchost, aproximace povrchů ale často není dostatečná a ve výsledcích se objevuje řada vizuálně nepříjemných jevů, například jako hrubé a vroubkované hranice stínů nebo chybějící stíny.

Neuniformní metody jsou vhodným způsobem, jak zkvalitnit výsledné řešení bez nutnosti rozčlenit celou scénu na velké množství malých plošek. Pro aproximaci povrchu jsou použity plošky různých velikostí, případně tvarů, a při konstrukci sítě je zohledněna geometrie scény a pozice světelných zářičů. Metody dosahují velmi přesných výsledků a i přes vyšší výpočetní náročnost jsou v praxi nejpoužívanější.

Dělení scény může být provedeno v prvním kroku algoritmu, ale lze ho také vytvářet dynamicky, zároveň s postupným řešením osvětlení. Tento přístup používá metoda známá jako *adaptivní dělení plošek* (*adaptive refinement*), viz. [Cohe86], kdy jsou v průběhu výpočtu osvětlení určována místa s velkými rozdíly osvětlení sousedních ploch a síť plošek je zde postupně zjemňována. Pro reprezentaci rozdělené plochy se většinou používá *kvadrantový strom*,



Obrázek 15.20: Adaptivní dělení plošek podle ostré hranice stínu

který je dvojrozměrnou verzí oktalového stromu uvedeného v části 14.2.2. Příklad adaptivního dělení podle hranice stínu je na obrázku 15.20.

Výpočet konfiguračních faktorů

Určení konfiguračních faktorů hraje v radiozitní metodě klíčovou roli, neboť se jedná o výpočetně nejnáročnější krok řešení. Velikost konfiguračního faktoru je ovlivněna třemi faktory:

- Velikostí plošek – čím bude zářící ploška j menší, tím bude menší i konfigurační faktor F_{ij}. Stejně tak klesá velikost konfiguračního faktoru se čtvercem vzdálenosti obou plošek.
- Vzájemnou polohou plošek – pokud bude zářící ploška j natočena „z profilu“, bude výsledný konfigurační faktor menší, než v případě, kdy je tato ploška natočena „čelem“.
- Vzájemnou viditelností plošek – pokud budou mezi ploškami ležet stínící plošky či objekty, bude konfigurační faktor menší, nežli v případě úplně vzájemné viditelnosti.

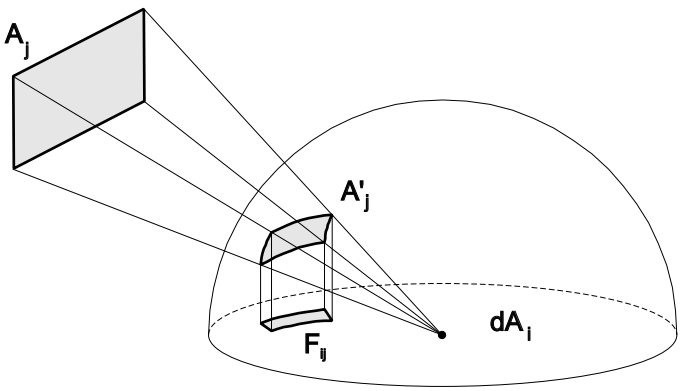
Hodnoty konfiguračních faktorů tedy závisí pouze na geometrii scény, což lze při výpočtu osvětlení vhodně využít. Pokud se ve scéně mění pouze osvětlovací podmínky (například se změnou denní doby nebo při zapnutí/vypnutí světel v místnosti), lze pro nové řešení radiozity využít stejné konfigurační faktory a podstatně tím ušetřit výpočetní čas.

Analytické metody řešení vztahu 15.9 je možné použít pouze u nejjednodušších případů vzájemné polohy a tvarů ploch, které navíc musí být vzájemně zcela viditelné. Vzhledem k tomu, že tato situace nastává ve scénách pouze vyjímečně, tyto metody se v praxi nepoužívají. Mohou však posloužit pro stanovení výchozích podmínek pro numerické metody, které konfigurační hodnoty více či méně aproximují.

Projekční metody jsou nejstarší a nejznámější používané aproximační metody výpočtu konfiguračních faktorů. První přístupy předpokládaly, že pokud jsou obě plošky navzájem dostatečně vzdálené, lze pro výpočet konfiguračních faktorů využít tzv. *Nusseltovy analogie*:

Konfigurační faktor diferenciální plošky dA_i a plošky A_j odpovídá ploše útvaru, jenž vznikne projekcí této plošky přes jednotkovou polokouli obklopující plošku dA_i do roviny této diferenciální plošky, samozřejmě s respektováním viditelnosti.

Dvojitá projekce plošky A_j nejprve na polokouli, obklopující uvažovanou plošku A_i, a poté do její roviny, naznačená na obrázku 15.21, je však stále příliš výpočetně složitá – efektivnější je nahradit polokouli polokrychlí a plošky promítat na její stěny (obr. 15.22).



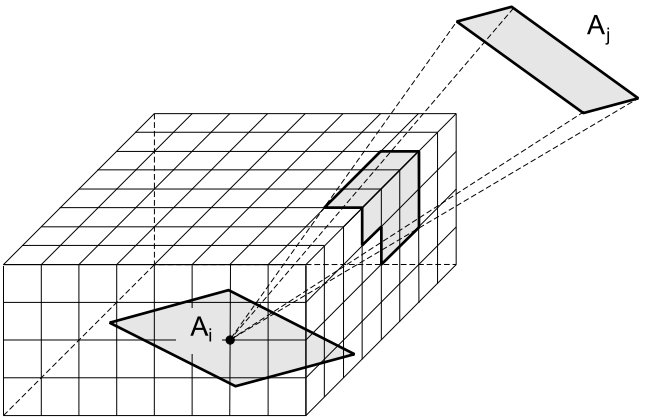
Obrázek 15.21: Nusseltova analogie

Abychom při změně tvaru polokoule na polokrychli zachovali korektní hodnoty konfiguračních faktorů, povrch polokrychle pokryjeme sítí elementů a každému z těchto elementů přiřadíme předem spočítaný tzv. *delta konfigurační faktor* ΔF_{ij} (zkráceně delta faktor). Hodnoty delta faktorů budou největší uprostřed horní podstavy polokrychle a nejmenší na bocích. Po promítnutí plochy A_j na stěny polokrychle určíme celkový konfigurační faktor F_{ij} jako součet delta faktorů jednotlivých elementů průmětu. Tento postup platí pouze pro dvojici ploch, které jsou vzájemně zcela viditelné, proto je nutné promítání plošek na stěny krychle doplnit řešením viditelnosti. Nejčastěji se používá paměť hloubky (kap. 11) umístěná na stěny polokrychle a vyhodnocovaná tedy pro každou plošku pětkrát.

Vzhledem k tomu, že polokrychle je vlastně jen rastr obalující testovanou plochu, dochází k aliasingu. Obdržené konfigurační faktory jsou u menších ploch zatíženy značnou chybou a často dochází k tomu, že malé plochy se na rastru polokrychle vůbec neprojeví, protože jejich průmět je menší než velikost elementu rastru. V případě že právě tyto malé plochy jsou světelnými zdroji či výrazně odráží záření, obdrží daná plocha znatelně méně světelné energie.

Pro odstranění tohoto problému lze využít faktu, že příspěvky ze směrů blízkých horizontu jsou poměrně malé. Polokrychli je proto možné nahradit jedinou plochou a viditelnost na ni počítat například algoritmem paměti hloubky. Jinou možností je zcela opustit Nusseltovu analogii a hledat konfigurační faktory pomocí vrhání paprsků.

Metody vrhání paprsků vycházejí ze stochastických numerických metod pro výpočet integrálů. Jejich princip je založen na náhodném vystřelování paprsků z dané plochy do okolí pro zjištění viditelných ploch a odhad výpočtu konfiguračních faktorů.



Obrázek 15.22: Promítání na polokrychli

Při určování paprsků vzorkováním povrchu [Wall89] se na dané plošce i náhodně vybere určitý počet bodů, stejně tak na plošce j. Každá úsečka s počátečním bodem vybraným z plochy i a koncovým bodem vybraným z plochy j určuje paprsek, který prochází pomyslnou jednotkovou polokoulí nad ploškou i a reprezentuje tak určitý delta faktor. Pokud paprsek nenarazí na překážku, je jemu odpovídající delta faktor přičten k hledanému celkovému konfiguračnímu faktoru F_{ij}. Metoda vzorkování směrů [Mall88] nám umožňuje určit najednou konfigurační faktory z jedné plochy ke všem ostatním plochám. Paprsky jsou vysílané do všech směrů nad zvolenou ploškou a při zásahu některé okolní plošky je k jejich vzájemnému konfiguračními faktoru přičten delta faktor.

Výhodou metod vrhání paprsků je možnost jejich aplikace i na složité geometrie scény. Problémem je nutnost výpočtu průsečíku paprsku se scénou a relativně velký počet paprsků nutný pro dosažení potřebné přesnosti. Přesto se tyto metody v praxi ukazují jako velice efektivní.

Řešení systému lineárních rovnic

V úvodu této části byl odvozen postup, ve kterém jsme integrální radiozitní rovnici 15.7 aproximovali pomocí systému lineárních rovnic 15.8. V této části uvedeme přehled metod, kterými lze tento systém efektivně řešit.

Pokud hledané neznámé radiozity B_i v rovnici 15.8 sdružíme na levé straně, dostaneme

$$B_i - \rho_i \sum_{j=1}^{n} B_j F_{ij} = E_i. \tag{15.10}$$

Tyto rovnice vyjádřené pro všech n ploch scény tvoří soustavu n lineárních rovnic pro n neznámých $B_1 \cdots B_n$:

$$\begin{pmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1-\rho_n F_{nn} \end{pmatrix} \cdot \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{pmatrix}, \tag{15.11}$$

či v maticovém zápisu

$$\mathbf{KB} = \mathbf{E} \tag{15.12}$$

Pro řešení soustavy lze použít běžné relaxační metody známé z numerické matematiky, například Gauss-Seidelovu či Jakobiho iterační metodu (viz [Rekt95]). Výpočet radiozitního řešení těmito metodami odpovídá postupnému shromažďování radiozit na jednotlivých ploškách scény. Klasický postup při řešení radiozity využívající Gauss-Seidelovu metodu je popsán v algoritmu 15.8.

1. Vypočti všechny prvky matice **K**
2. Přiřaď všem B_i počáteční odhad
3. Dokud výpočet nezkonvergoval, opakuj:
   (a) Přiřaď všem B_i novou hodnotu, spočítanou jako
   $$B_i = E_i - \sum_{j=1, j\neq i}^{n} \frac{B_j K_{ij}}{K_{ii}}.$$
4. Zobraz scénu podle hodnot v matici **B**

Algoritmus 15.8: Výpočet radiozity Gauss-Seidelovou iterační metodou

Řešení radiozitní soustavy Gauss-Seidelovou metodou má bohužel pro uplatnění v grafice mnoho nedostatků. Vzhledem k principu shromažďování radiozity na jednotlivých ploškách je k dispozici až celkové řešení, i když by bylo mnohdy výhodné mít možnost zobrazit dílčí řešení, které se postupně zlepšuje.

Jedno z možných řešení představili Cohen, Wallace a Greenberg v [Cohe88]. Snahou autorů bylo navrhnout algoritmus, jenž kromě možnosti shlédnout radiozitní simulaci po každém iteračním kroku konverguje co nejrychleji na samém počátku výpočtu. Jejich metoda *progresivní radiozity* (*progressive refinement*) spočívá nikoli ve shromažďování radiozity na jednotlivých

1. Všem ploškám přiřaď $B_i = E_i$ a $\Delta B_i = E_i$
2. Dokud výpočet nezkonvergoval, opakuj:
   (a) Vyber plošku i s největší hodnotou $B_i A_i$
   (b) Spočti konfigurační faktory F_{ji} pro ostatní plochy j
   (c) Přiřaď všem plochám j nové radiozity spočítané jako

$$\begin{aligned} B_j &= B_j + \Delta B_i \rho_j F_{ji} \\ \Delta B_j &= \Delta B_j + \Delta B_i \rho_j F_{ji} \end{aligned}$$

   (d) Přiřaď $\Delta B_i = 0$
3. Zobraz výsledek podle aktuálních hodnot v matici **B**

Algoritmus 15.9: Progresivní radiozita

ploškách scény, ale naopak ve „vystřelování“ radiozity z plošky, která má nejvíce nevyzářené energie, jak ukazuje alg. 15.9.

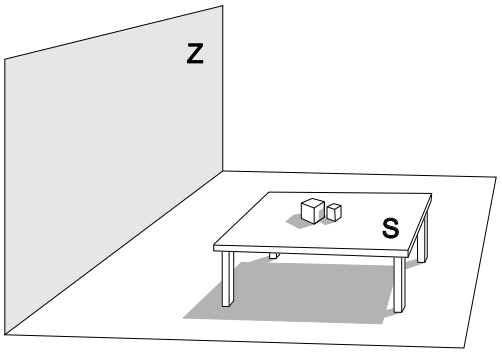
Zjednodušeně lze říci, že tato metoda postupně sleduje radiozitu od světelných zdrojů k ostatním plochám scény, jež se tak stávají sekundárními zdroji světla. Šíření radiozity je od těchto sekundárních zdrojů sledováno dále, dokud nevyzářená radiozita neklesne pod zadanou úroveň. Důvod, proč se metoda progresivní radiozity přibližuje zpočátku k výslednému řešení rychleji než klasická radiozita, spočívá v tom, že největší vliv na rychlost konvergence při radiozitním výpočtu má určení radiozity v místech osvětlených silnými světelnými zdroji. Při klasickém výpočtu radiozity Gauss-Seidelovou iterační metodou víme pouze, že silné světelné zdroje budou zcela jistě do výpočtu zahrnuty, nemůžeme ovšem ovlivnit, v kolikátém iteračním cyklu se tak stane.

15.10.3 Hierarchická radiozita

Metoda hierarchické radiozity [Hanr91], se zaměřuje na zjednodušení výpočtu konfiguračních faktorů F_{ij} mezi jednotlivými ploškami a na vytvoření hierarchie adaptivního dělení plošek.

Postupné zjemňování dělení plošek přináší zlepšení kvality výsledného obrazu, ale současně zvyšuje počet konfiguračních faktorů, které musíme počítat. Hierarchická radiozitní metoda je založena na myšlence, že vliv radiozity nově vzniklých malých individuálních plošek na vzdálené plochy je ve většině případů zanedbatelný.

Příkladem je scéna na obrázku 15.23. Při adaptivním dělení bude deska stolu S v okolí krabic, které na ní leží, rozdělena velmi jemně, protože algoritmus adaptivního dělení plošek se



Obrázek 15.23: Vztahy desky stolu vůči vzdálené zdi a blízkým krabicím jsou odlišné

bude snažit co možná nejvěrněji vystihnout hranice stínu krabic. Bude-li vzdálenost desky stolu S a zdi Z dostatečně velká, není třeba při přenosu radiozity mezi S a Z počítat konfigurační faktory pro všechny jemně dělené plošky, ale lze je nahradit jediným „skupinovým“ faktorem. Toto zjednodušení neplatí v opačném směru, protože radiozita zdi Z silně ovlivňuje vzhled stínů a tedy radiozitu individuálních plošek na stole S.

Dělení ploch

V algoritmu hierarchické radiozity jsou jednotlivé plochy scény uspořádány do několikaúrovňových kvadrantových stromů, stejně jako v metodě adaptivního dělení plošek. Výpočet začíná se scénou rozdělenou hrubě na přibližně stejně velké plošky, které představují kořeny kvadrantových stromů. Srdcem algoritmu je funkce nazývaná *orákulum* (*link oracle*), ohodnocující vzájemný vztah libovolných dvou uzlů A_i a A_j v kvadrantovém stromu z hlediska radiozity. Pokud orákulum určí, že přesnost přenosu energie z uzlu A_i do uzlu A_j vyhovuje daným kritériím, je vytvořena jednosměrná vazba propojující uzel A_i s uzlem A_j. Pokud chyba překročí danou mez a plošky uložené v daných dvou uzlech zatím nebyly jemněji rozděleny, orákulum určí, zda má být rozdělena jen jedna z nich či obě dvě. Pro následníky uzlů A_i a A_j je pak orákulum vyvoláno rekurzívně znovu. Postup je zapsán v dílčím algoritmu 15.10, v němž rekurzivní funkce VytvořVazbu prozkoumává s pomocí orákula vztah dvou uzlů.

Podle pokynů orákula vznikne v první části výpočtu sada vazeb, po kterých se šíří radiozita mezi plochami. Všimněme si, že ačkoliv uzly A_i i A_j mohou být v průběhu výpočtu i několikanásobně jemněji rozděleny, pro danou dvojici A_i, A_j může orákulum rozhodnout, že postačuje přenos radiozity jen na vyšší úrovni kvadrantového stromu. Radiozita vyzářená celým podstromem uzlu A_i se v takovém případě přenese jako jediná radiozitní hodnota do uzlu A_j, kde je rozdělena jednotlivým ploškám v jeho podstromu.

VytvořVazbu (A_i, A_j)

1. Vyhodnoť orákulum pro uspořádanou dvojici uzlů $[A_i, A_j]$
2. Pokud je velikost obou ploch uspokojivá, vytvoř orientované propojení $A_i \rightarrow A_j$
3. Pokud je plocha A_i příliš velká, pak
   (a) Je-li A_i listem kvadrantového stromu, rozděl ji na čtyři menší plošky
   (b) Pro všechny potomky A'_i uzlu A_i VytvořVazbu (A'_i, A_j)
4. Pokud je plocha A_j příliš velká, pak
   (a) Je-li A_j listem kvadrantového stromu, rozděl ji na čtyři menší plošky
   (b) Pro všechny potomky A'_i uzlu A_i VytvořVazbu (A_i, A'_j)

Algoritmus 15.10: Vytváření vazeb mezi uzly stromu při hierarchické radiozitě

V další části algoritmu je radiozita distribuována mezi ploškami podle výše uvedených vazeb. Pro všechny existující vazby orákulum zkontroluje chybu a podle ní jsou vazby případně zjemněny. Celý výpočet distribuce radiozity a zjemňování vazeb se opakuje, dokud není dosažena požadovaná přesnost, jak ukazuje alg. 15.11.

Metoda hierarchické radiozity je v dnešní době asi nejpopulárnější variantou radiozitní metody. V průběhu let od jejího publikování byla objevena řada nových vylepšení, které dále zvyšují efektivnost této metody. Mezi nejdůležitější patří seskupování ploch, vlnková radiozita a v neposlední řadě také začlenění vlivu důležitosti.

Seskupování ploch

Hierarchická metoda umožňuje dělit velké plochy na menší, ale neumí využívat seskupování ploch do větších skupin. Vrcholy hierarchie proto tvoří jednotlivé vstupní plochy a počáteční fáze metody musí vytvořit vazbu pro přenos osvětlení pro každou jejich dvojici. Z tohoto důvodu je klasický hierarchický přístup vhodný pro scény s malým množstvím základních ploch, ale pro rozsáhlé scény se stává z důvodu výpočetních nároků nepraktický.

Pro řešení tohoto problémů byla vyvinuta metoda *seskupování* (*clustering*) [Chri97], která spočívá ve sdružování sousedních objektů do skupin a přenosem energie přímo mezi těmito skupinami. Nad vstupními plochami je tak postupně vytvářena hierarchie, která je završena jedinou skupinou zahrnující celou scénu. První fáze vytváření vazeb pro přenos osvětlení je nyní zredukována na jeden počáteční spoj na vrcholu hierarchie, který je dále rekurzivně zjemňován.

1. Všem ploškám přiřaď $B_i = E_i$
2. VytvořVazbu (A_i, A_j) pro všechny dvojice plošek A_i a A_j
3. Opakuj
   (a) Dokud výpočet nezkonvergoval, opakuj:
      i. Pro každý uzel kvadrantového stromu urči radiozitu přicházející po všech vazbách na tento uzel
      ii. Pro každý uzel kvadrantového stromu (rekurzívně) rozděl přijatou radiozitu potomkům a naopak přičti k jeho vyzářené radiozitě hodnoty vyzářené jeho potomky
   (b) Pro každou dvojici plošek A_i, A_j proveď:
      i. Vyhodnoť orákulum pro uspořádanou dvojici $[A_i, A_j]$
      ii. Pokud je A_i nebo A_j příliš velká, ZjemniVazbu (A_i, A_j)
   (c) Pokud nebylo třeba zjemnit žádnou z vazeb, skonči
4. Zobraz výsledek

Algoritmus 15.11: Hierarchická radiozita

Vlnková radiozita

Jedním ze závažných problémů radiozitní metody je výskyt chyb ve vypočteném osvětlení, které vznikají v důsledku rozdělení scény do rovinných plošek, na kterých je osvětlení aproximováno konstantní hodnotou. Plošky by měly být dostatečně malé, aby tato jednoduchá aproximace byla postačující. Jinak se objevují různé nepřesnosti na hranicích plošek a okrajích stínů, které jsou vizuálně velmi nepříjemné a použití Gouraudovy interpolace je odstraňuje pouze částečně.

Mezi navrhovanými postupy řešení dosáhla největšího úspěchu tzv. *vlnková radiozita* publikovaná v [Stol96], která výskyt vizuálních chyb minimalizuje a poskytuje vysoce přesný odhad ve všech místech s jemnými přechody osvětlení. Základní myšlenkou je vyjádřit radianci na ploše jako lineární kombinaci funkcí a místo jedné konstantní hodnoty osvětlení hledat koeficienty této kombinace. Kvalita výsledků se při aproximaci osvětlení pomocí funkcí podstatným způsobem zvyšuje a přesný odhad získáváme už při hrubším rozdělení scény.

Vliv důležitosti

Dosud uvedené metody radiozity počítají osvětlení se stejnou přesností ve všech místech scény. Podstatného urychlení lze dosáhnout, pokud využijeme pozorování, že v mnoha případech není nutné celé pohledově nezávislé řešení, ale postačuje získání částečného řešení pouze pro aktuální

pohled na scénu. Výhodné je proto vzít při výpočtu v úvahu důležitost osvětlení v různých částech scény pro právě generovaný obrázek. Pokud dovolíme nižší přesnost v méně závažných místech, můžeme se soustředit na viditelné části scény a řešit zde osvětlení s větší přesností.

Požadovaného cíle lze dosáhnout, pokud v průběhu řešení osvětlení počítáme tzv. *důležitost* (*importance*) [Patt93] a tuto veličinu použijeme pro řízení celého výpočtu. V hierarchické radiozitní metodě byl průkopníkem tohoto přístupu Smits [Smit92], který prvně prezentoval nový postup dělení plošek, když jsou přednostně zjemňovány plošky důležité pro výsledný obrázek.

15.10.4 Stochastické metody řešení

Jak bylo ukázáno v předchozí části, při řešení radiozitní rovnice je scéna rozdělena na síť ploch, na kterých je osvětlení aproximováno konstantními hodnotami a radiozitní rovnice je redukována na soustavu lineárních rovnic. Tato soustava je následně řešena pomocí numerických iteračních metod, které postupně v jednotlivých krocích zpřesňují řešení osvětlení. Nutnost uložení konfiguračních faktorů však způsobuje vysokou paměťovou náročnost.

Výpočet konfiguračních faktorů je nejkomplikovanější krok řešení, jehož hlavní problémy se dají shrnout do následujících bodů:

- Složitost výpočtu – určení přesné hodnoty jednoho konfiguračního faktoru vyžaduje nalezení řešení netriviálního integrálu. Mezi navrženými aproximačními algoritmy dosáhly velkého úspěchu metody vrhání paprsků, bohužel stále platí, že řešení je velmi časově náročné.
- Množství konfiguračních faktorů – výpočet je nejen komplikovaný, ale je nutné jej provádět pro každou dvojici plošek. Postupy jako adaptivní dělení, hierarchická radiozita, použití seskupování a zohlednění důležitosti dokáží jejich množství redukovat, ale u složitých scén zůstává tento počet nepříjemně vysoký.
- Vysoké paměťové nároky – matice konfiguračních faktorů bývá ve většině případů značně rozsáhlá a závažným problémem je její uložení do paměti.

Elegantní řešení těchto problémů poskytují *stochastické metody radiozity*.

Monte Carlo odhad konfiguračních faktorů

Monte Carlo metody výpočtu konfiguračních faktorů jsou založeny na náhodném vrhání paprsků. Základní typ těchto metod vychází z pozorování, že konfigurační faktor F_{ij} mezi dvojicí ploch může být interpretován jako pravděpodobnost, že paprsek opouštějící plochu i zasáhne plochu j. Z plochy i jsou proto náhodně vystřelované paprsky do prostoru a hodnota konfiguračního

faktoru je odhadnuta jako podíl počtu paprsků dopadlých na plochu j a počtu všech paprsků vyslaných z plochy i [Shir90].

Existují i přístupy, které řeší všechny konfigurační faktory ve scéně zároveň. Jejich princip spočívá v generování paprsků procházejících celou scénou a stanovením odhadu na základě průsečíků paprsků s plochami scény. Jedná se buď o svazky rovnoběžných paprsků [SlllK98] nebo náhodně generované paprsky [Sber93]. Po určení všech průsečíků je konfigurační faktor pro každou dvojici ploch odhadnut jako poměr počtu paprsků procházejících skrz obě plochy a počtem paprsků, které protínají pouze jednu z ploch.

Stochastické metody radiozity

Patří mezi ně Monte Carlo a kvazi Monte Carlo metody, které při řešení soustavy rovnic používají místo přesné matice konfiguračních faktorů pouze její celkovou aproximaci, která má v průměrném případě stejné vlastnosti. Z tohoto důvodu odpadá nutnost explicitního výpočtu a uložení jednotlivých konfiguračních faktorů a přístup je vhodný pro výpočet distribuce osvětlení i ve složitých scénách. Tento způsob řešení vychází přímo z numerických Monte Carlo metod pro řešení soustavy lineárních rovnic.

Hlavním typem stochastických radiačních metod jsou metody *stochastické iterace* [Neum94]. Tyto metody řeší soustavu rovnic pomocí klasických iteračních metod jako je Jacobiho nebo Gauss-Seidelova metoda, ale matice konfiguračních faktorů a celkové řešení osvětlení je v každém kroku aproximováno s použitím výše uvedených Monte Carlo odhadů.

Nejpoužívanější metodou je stochastická Jacobiho iterace [Neum95, Sber95]. V jednom kroku iterace se z každé plošky vystřelí do scény v náhodně vybraných směrech tolik paprsků, kolik odpovídá nezpracované energii dané plošky. Tyto paprsky slouží pro odhad konfiguračních faktorů plošek i pro přenos osvětlení. Celkový počet paprsků vystřelených v jedné iteraci je předem zvolen.

Pokud si paprsek představíme jako část cesty světelné částice, můžeme postup výpočtu interpretovat jako sledování šíření světelných částic ve scéně, kdy je generována řada světelných cest zároveň. Každý krok výpočtu spočívající ve vystřelení množiny paprsků znamená prodloužení jednotlivých světelných cest. Pokud budeme sledovat cestu každé částice světla samostatně, získáváme další způsob řešení soustavy radiozitních rovnic, které je známé jako *diskrétní náhodná procházka* [Sber96, Sber97].

Kapitola 16

Vizualizace objemových dat

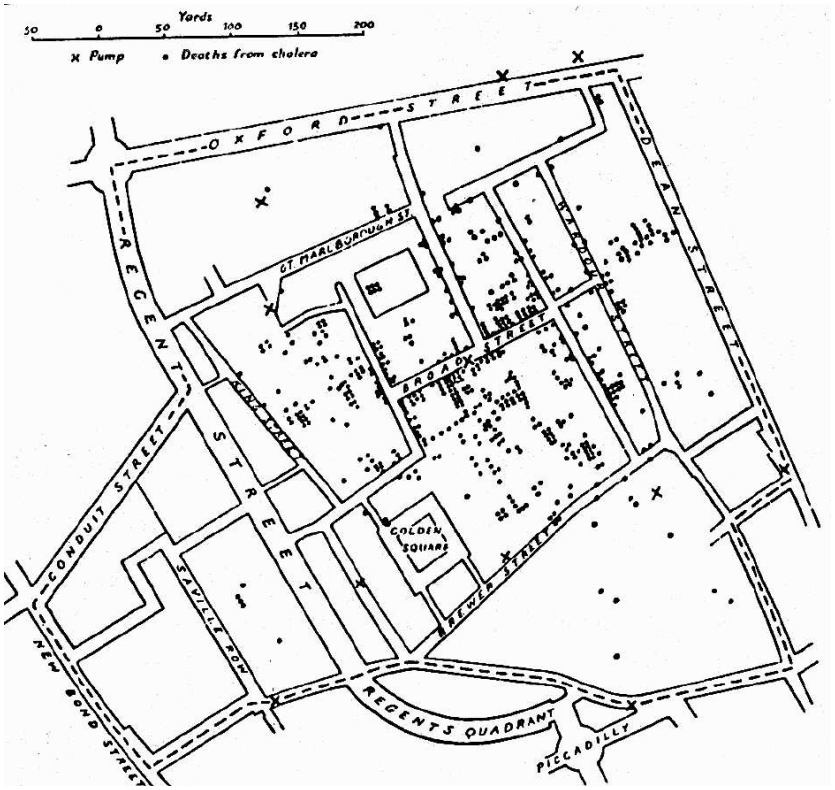
Pojmem *vizualizace* označujeme jakýkoliv postup, při němž vyjadřujeme nějaké hodnoty nebo vztahy pomocí obrázků. Jde o formu sdělení, která je názorná. Příkladem je meteorologická mapa, kterou vídáme v televizi při předpovědi počasí, či vysvětlující schéma dráhy planet při zatmění Slunce.

V užším slova smyslu chápeme vizualizaci jako sadu nástrojů a postupů, sloužících k *vizuální analýze dat*, tedy o celý proces zkoumání dat a informací po jejich převedení do grafické podoby. Cílem vizualizace je pochopení zkoumaných jevů a jejich vnitřních vztahů. Prostředkem však nejsou tabulky čísel, jako u numerické analýzy, ale zobrazení, v maximální míře interaktivní.

Za první použití vizualizace k vědeckým účelům je považován kartografický plánek doktora Johna Snowa z roku 1855 (viz obr. 16.1). Zabýval se výzkumem epidemie cholery, která postihla okolí londýnské ulice Broad Street v září roku 1854. Protože nemohl přijít na to, kde se nacházel zdroj nákazy, zaznamenával si do mapky města místa, kde někdo zemřel, a zjistil tak, že všechny oběti braly pitnou vodu z jedné společné studny. Po bližším ohledání se potvrdilo, že zdrojem nákazy je právě voda z této studny, znečištěná kalem prosakujícím z nedaleké žumpy. Studny jsou na obrázku označeny křížky, oběti tečkami.

O vznik vizualizace jako samostatné disciplíny se zasloužil hlavně rozvoj algoritmů umožňujících prostorové zobrazení velkých souborů skalárních prostorových dat. Pro vizualizaci je typické zejména velké množství často vícerozměrných dat. Z toho vyplývají vysoké nároky na výkon algoritmů i výpočetních systémů.

Vizualizace kromě vytváření vlastních postupů a metod přebírá metody a postupy z jiných oborů. Kromě počítačové grafiky jsou jimi například zpracování obrazu, počítačové vidění a umělá inteligence. V následující kapitolách se seznámíme se základními algoritmy vizualizace objemových skalárních dat.



Obrázek 16.1: Mapka případů úmrtí při epidemii cholery v Broad Street v Londýně v září 1854, kterou v roce 1855 zhotovil Dr. John Snow

16.1 Vizualizovaná data

Pro metody zkoumání dat je důležité, zda zpracovávaná data byla získána měřením reálných jevů, nebo zda jde o výstup simulace matematického modelu, či voxelizaci geometrického tělesa.

V prvním případě máme k dispozici *pevný počet vzorků*. Zdrojem tohoto typu dat jsou např. konfokální mikroskopy či rentgenové a magneticko-rezonanční tomografy. Při zobrazování a jiném zpracování obvykle musíme použít techniky převzorkování, interpolace a filtrování, které se používají při práci s 2D obrazy (viz části 4.6 a 4.3.1). Přitom mohou snadno vznikat *artefakty*, tj. nepřesnosti, kdy některé detaily nejsou zobrazeny, nebo se naopak objeví tam, kde v datech nejsou.

V případě *simulace* a *voxelizace* známe postup, jak vypočítat přesné hodnoty v libovolném místě, známe model simulovaného děje či geometrický popis voxelizovaného objektu. Pokud chceme znát nějaký detail, můžeme vybrat místo, které nás zajímá, provést výpočet znovu a získat přesné hodnoty. Pojmem voxelizace označujeme převod geometricky či stochasticky definovaných objektů do objemové reprezentace konečnou množinou vzorků. Simulace se používá v těch případech, kdy zkoumáme události a jevy, které buď nedokážeme, nebo nechceme přímo sledovat. Některé děje jsou buď příliš malé, nebo naopak rozlehlé, velmi rychlé, nebo příliš pomalé, aby se daly přímo sledovat a proto se jejich chování simuluje.

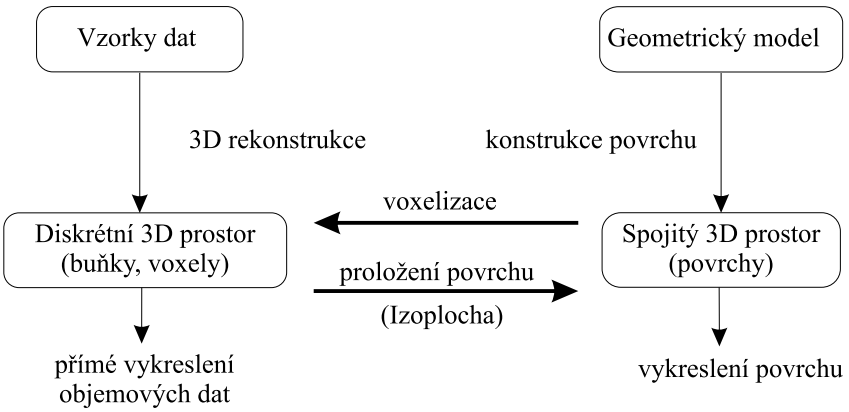
Vizualizační techniky lze také klasifikovat podle *prostorového uspořádání vzorků*. To je obvykle silně svázáno s aplikační oblastí a postupem, jakým data získáváme. Základními případy uspořádání vzorků jsou pravidelné a nepravidelné mřížky a rozptýlená data v dvoj-, troj-, i vícerozměrném prostoru. Hodnotami vzorků bývají nejčastěji skaláry a vektory skalárních hodnot. V dalším textu budeme předpokládat pravidelně rozmístěné vzorky dat v ekvidistantních mřížkách či voxelech. Odlišné schéma rozmístění vzorků bude do pravidelné mřížky převedeno pomocí převzorkování.

Cíle analýzy dat, a tudíž i způsob interpretace vytvářeného obrazu se samozřejmě liší podle typu zkoumaného jevu. Vizualizační algoritmy, které dále popíšeme, jsou určeny k zobrazení skalárních objemových dat (*volume rendering*), neboť ty patří k nejvíce používaným.

16.2 Skalární objemové algoritmy

Algoritmy vizualizující skalární prostorové mřížky lze rozdělit na *algoritmy zobrazující povrchy* a na *přímé objemové algoritmy*. Srovnání obou přístupů je znázorněno na obrázku 16.2. Algoritmy zobrazující povrchy (*SF, surface-fitting algorithms*) vytvářejí nejprve geometrickou reprezentaci povrchu. Pracují tedy s objemovými daty nepřímo, neboť je nejprve proloží povrchem, který reprezentují geometrickými primitivy, nejčastěji sítí trojúhelníků. Až následně tato primitiva zobrazují klasickými metodami počítačové grafiky. Výhodou je, že se přechodem k ploškové reprezentaci značně sníží množství dat. Místo celé mřížky vzorků zůstanou jen povrchové elementy popsané výrazně nižším počtem hodnot, s menším množstvím dat se rychleji pracuje a navíc pro zobrazování geometrických primitiv lze využít grafické procesory.

Objemové algoritmy (*DVR, direct volume rendering*) zobrazují skalární data přímo, bez nutnosti převodu do povrchové reprezentace (viz obr. 16.2 vlevo). Využívají plnou prostorovou informaci a před zobrazováním nemusíme jednoznačně vědět, zda vzorek (voxel či vrchol buňky) patří nebo nepatří k zobrazovanému objektu, respektive k jeho povrchu. Metody lze rozšířit i na zobrazování poloprůhledných materiálů a struktur uvnitř jiných struktur. Objemové techniky umožňují současně vizualizovat jak rozhraní mezi materiály, tak jejich vnitřek.



Obrázek 16.2: Metody zobrazování objemových dat (podle [Kauf91])

K převodu spojité reprezentace na objemovou lze použít voxelizaci, tj. navzorkování spojitého povrchu (geometrického popisu) v zadaném rozlišení. Často se spolu s hodnotou vzorku v daném místě mřížky ukládá i hodnota normály povrchu, která je potřebná při vyhodnocování osvětlovacího modelu. K převodu z objemové reprezentace na povrchovou poslouží např. *nalezení izoplochy*, tj. spojení míst v datech o stejné hodnotě, či komplikovanější metody *segmentace* (viz dále v části 16.3.6).

16.2.1 Algoritmy zobrazující povrchy

Algoritmy, které při vizualizaci objemových dat zobrazují povrchy, pracují nepřímo, neboť zobrazování předchází generování (pomocné) povrchové reprezentace.

Pokud zobrazujeme průběhy nějaké fyzikální veličiny (např. teploty v různých místech reaktorové nádoby) pomocí ploch, určíme zobrazovaný povrch jako místo, kudy probíhá plocha s konstantní hodnotou veličiny.

Při zobrazování skutečných útvarů, například lidských orgánů z dat naměřených počítačovým tomografem, se jedná o obraz existujících objektů. O příslušnosti voxelu k objektu (respektive k jeho hranici) rozhodneme klasifikací. Máme v principu dvě možnosti. Buď zvolíme nějaké jednoduché kritérium, pomocí něhož poznáme hranici během zobrazování dat (např. kritérium pro výběr izoplochy), nebo předřadíme před zobrazování krok, jehož výsledkem je označení, který voxel náleží a který nenáleží oplášťovanému objektu (viz část 16.3.6). Vzniklý „binární“ objem slouží jako prostorová maska, která vymezuje části prostorových dat, které jsou předmětem zobrazování nebo jiného zpracování. V jednoduchých případech lze binární objem vytvořit pomocí prahování, v kombinaci s morfologickými operacemi eroze a dilatace, složitější případy řešíme komplikovanějšími algoritmy v samostatné fázi zpracování.

Způsobů generování povrchové reprezentace v místě izoplochy je několik:

- *propojování kontur* (*contour connecting*) – Data se zpracovávají postupně po vrstvách (dvojicích řezů). V jednotlivých řezech se nejprve označí obrysy hranice, poté se v každé dvojici řezů stanoví topologie kontur (jednoznačné přiřazení, co s čím se bude spojovat) a provede se opláštění (*tessellation*). Podrobně v části 7.3.1.
- *povrchové kostky* (*opaque cubes, cuberille*) – Algoritmus chápe objemové elementy jako plné kostičky (viz část 7.2.1). Po nalezení povrchových kostiček (sledováním hranice okolo zadané hodnoty) se každá z nich převede na šestici čtyřúhelníků, které se zobrazují. Výsledkem je obraz se zřetelně viditelnou „kostkovou“ strukturou, která se příliš nevyhladí ani pokud odhadujeme směr normál ve vrcholech z původních dat. Tento algoritmus se pro hrubost výsledné reprezentace používá jen pro rychlé znázornění objemové informace při malých nárocích na kvalitu zobrazení.
- *pochodující kostky* (*marching cubes*) – Jeden z nejrozšířenějších algoritmů pro tvorbu sítě trojúhelníků, která aproximuje povrch procházející jednotlivými objemovými elementy (kostkami). Je popsán v části 7.3.2.
- *rozdělení kostek* (*dividing cubes*) – Výsledkem tohoto algoritmu nejsou plošky, ale povrchové body s normálou (*smart points*). Popis viz část 7.3.2.

Povrchové sítě trojúhelníků lze zobrazovat běžnými algoritmy s viditelností a hladkým stínováním (viz část 11) či metodou sledování paprsku (část 15.9). Postupem jejich generování se zabývá část 7.3.1.

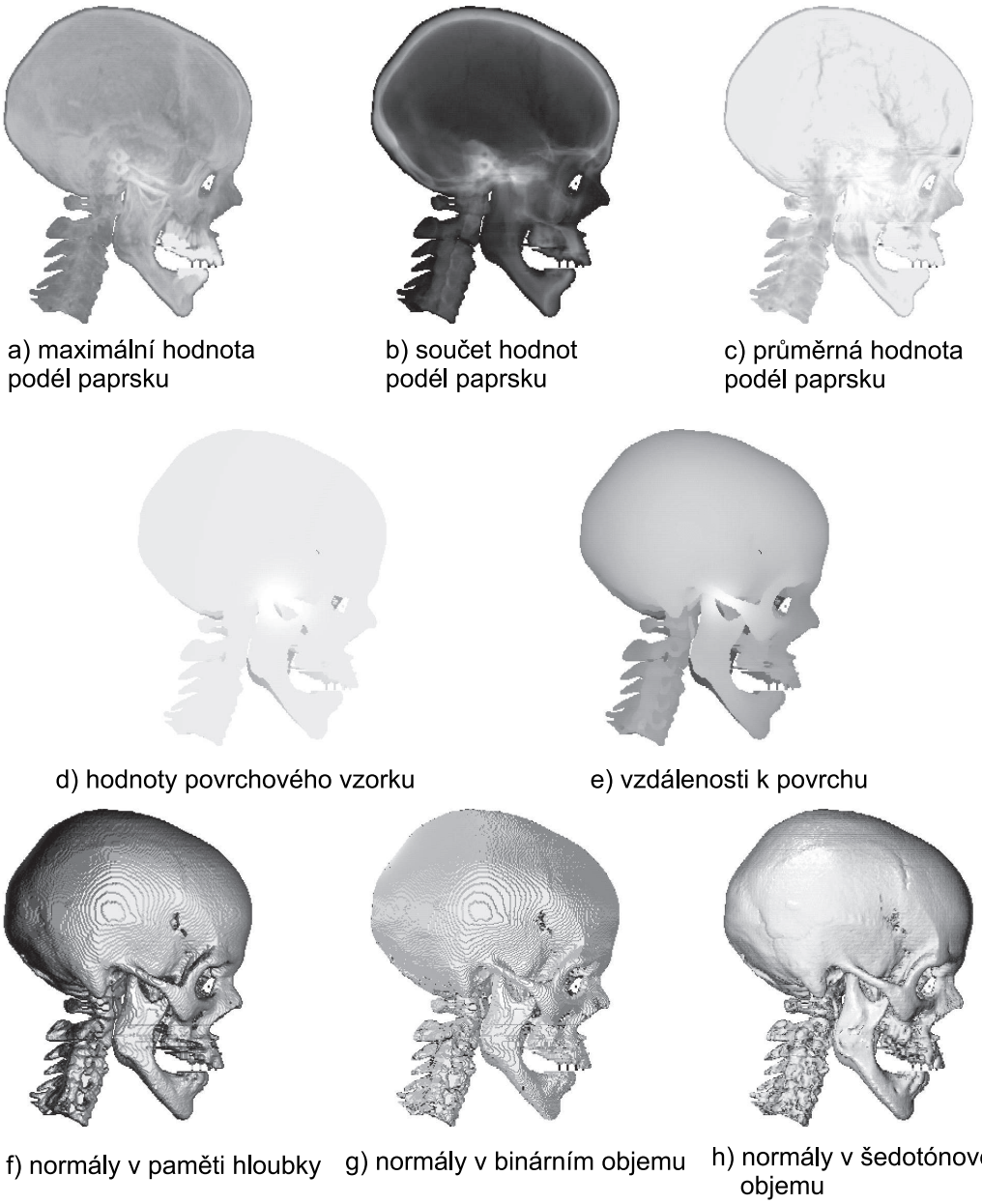
16.3 Přímé zobrazování objemů

Cílem tohoto odstavce je ukázat některé metody, které slouží k zobrazování trojrozměrných skalárních dat objemovými metodami (*direct volume rendering – DVR*), konkrétně metodou vrhání paprsku (*ray casting*), jejíž podrobný popis je uveden v části 15.9.

Zobrazování objemových dat metodou vrhání paprsku lze dále členit podle toho, s jakými daty pracuje a co z nich zobrazuje, na

1. metody pracující s daty bez hledání povrchu (tj. bez segmentace),
2. metody hledající povrch, které však nezjišťují normálu a
3. metody, které hledají povrch a snaží se odhadnout jeho normály.

Pro ilustraci jsou na obrázku 16.3 uvedeny ukázky získané aplikací uvedených metod na stejná data. Jde o prostorová data lidské lebky z počítačového tomografu, čítající 341 řezů v rozlišení 226×272 vzorků.



Obrázek 16.3: Příklady objemového zobrazování (Obrázky, vygenerované systémem Medicus, poskytl B. Ježek, VLA JEP Hradec Králové.)

U jednotlivých metod uvádíme postup výpočtu intenzity jasu pixelu I ve výsledném obraze pro jediný vržený paprsek. Při zobrazování je pak samozřejmě nutno tuto hodnotu převést do rozsahu jasů použitého displeje (viz odstavec 2.1.1). Data, se kterými pracujeme, jsou uspořádána v pravidelné trojrozměrné mřížce. Jako I_i či $f(X(i))$ označíme skalární hodnotu intenzity i-tého vzorku podél paprsku, resp. vzorku v rastru o souřadnicích $X(i) = = (x_0(i), x_1(i), x_2(i))$. Tyto skalární hodnoty se obvykle označují názvy jako jas, hustota (*density*) či intenzita vzorku. Písmenem J značíme množinu započítaných vzorků. Započítané vzorky jsou ty vzorky, které jsou zasaženy sledovaným paprskem, vyhoví případnému dalšímu kritériu a jsou použity ve výpočtech.

Jinou možností reprezentace dat jsou tzv. *binární objemy*, které mají stejnou strukturu, avšak každý vzorek nabývá hodnoty 0 nebo 1, podle toho zda přísluší ke hledanému objektu či nikoliv. V některých případech je výhodné používat obě reprezentace, případně omezit počet hodnot jasu a využít nějaký bit jako bit příznaku.

16.3.1 Metody nehledající povrch

Metody pracující s daty bez hledání povrchu nevyžadují žádné předzpracování dat, jsou rychlé, a proto se hodí zejména pro vytváření náhledu. Při postupném zobrazování objemu využívají metodu vrhání paprsku jednotlivými pixely obrazovky.

První metoda zobrazuje pouze nejjasnější struktury podél paprsku (*maximum intensity projection, MIP*) (viz obr. 16.3a) dle vztahu

$$I = \max_{i \in J}(I_i).$$

Jinou možností (*summed intensity projection*) je vypočítat jas pixelu jako součet těch intenzit podél paprsku, které leží v zadaném intervalu (viz obr. 16.3b)

$$I = \sum_{i \in J}(I_i).$$

Modifikací předchozí metody získáme metodu, která normalizuje příspěvky různého počtu stejných intenzit (*average intensity projection, additive reprojection*). Jas pixelu získáme jako průměr z intenzit podél paprsku, které leží v zadaném intervalu (viz obr. 16.3c)

$$I = \frac{\sum_{i \in J}(I_i)}{|J|}.$$

16.3.2 Jednoduché zobrazení povrchu

Předpokládejme, že máme binární objem b, ve kterém jsou vzorky náležející povrchu objektu označeny 1. Současně pracujeme s objemem, ve kterém máme rovněž k dispozici příslušné

intenzity jasu. Intenzitu jasu těch vzorků, které jsou na povrchu a mají tedy nastavenu hodnotu bitu v binárním objemu na 1, označíme I_s.

První metodou, která se používá pro zobrazení povrchu, je prosté nastavení pixelu, do kterého se promítne vzorek I_s na intenzitu I_s (*voxel value projection*). Protože těchto vzorků může být více, nastaví se jas pixelu na hodnotu intenzity prvního povrchového vzorku na dráze paprsku, tj.

$$I = I_s.$$

Tato metoda nezobrazuje tvar, ale pouze barvu objektu, což je obdobou konstantního stínování u těles zadaných hraniční reprezentací (viz obr. 16.3d).

Druhá metoda nastavuje jas pixelu na obrazovce podle hodnoty vzdálenosti nejbližšího vzorku na dráze paprsku (*depth/distance-only shading, Z-buffer shading*), tj. nastavuje intenzitu jasu pixelu $P = [p_0, p_1]$ na obrazovce podle hodnoty z v paměti hloubky, což zapisujeme

$$I = Z(I_s) = Z(p_0, p_1).$$

Tato metoda potlačuje šum vzorkování, ale spolu s ním i hrany a nespojitosti, které jsou důležité při vnímání tvarů člověkem (viz obr. 16.3e).

16.3.3 Zobrazení povrchu s normálou

Kvalitnějšího znázornění povrchu, oproti metodám z předchozích odstavců, dosáhneme metodami, které se pokoušejí odhadnout orientaci povrchu v místě dopadu paprsku. V tomto bodě potom můžeme vypočítat osvětlení např. dle Phongova osvětlovacího modelu z odstavce 10.5. Dále se zmíníme o třech metodách, které směr normály odhadují různým způsobem. Ve všech případech je však třeba vypočítaný vektor normalizovat. Tyto metody pracují s binárním objemem b a s objemem vzorků I.

První metoda (*Z-buffer gradient shading*), aproximuje normálu k povrchovému vzorku vektorem, jehož kolmým průmětem do plochy obrazovky je vektor gradientu v paměti hloubky, tj. vektor gradientu v dvojrozměrné mapě vzdáleností k povrchu (viz obr. 16.3f). Pro paprsek procházející pixelem o souřadnicích $[p_0, p_1]$ lze složky normály $\vec{n} = (n_0, n_1, n_2)$ vypočítat dvojrozměrnou symetrickou diferencí

$$\begin{aligned} n_0 &= Z(p_0 + 1,\ p_1) - Z(p_0 - 1,\ p_1), \\ n_1 &= Z(p_0,\ p_1 + 1) - Z(p_0,\ p_1 - 1), \\ n_2 &= 1. \end{aligned}$$

Tato metoda zachycuje tvar tělesa, ale na zaoblených površích vytváří artefakty v podobě vrstevnic.

Další metoda (*voxel gradient shading*) odhaduje orientaci povrchu podle gradientu v binárním objemu b klasifikovaných vzorků. Složky tohoto gradientu nabývají pouze jednu z hodnot $\{-1, 0, 1\}$ a výsledné normály mohou nabývat pouze jednu z 27 hodnot (viz obr. 16.3g). Normálu vypočítáme podle vztahu

$$\begin{aligned} n_0 &= b(s_0+1,\ s_1,\ s_2) - b(s_0-1,\ s_1,\ s_2), \\ n_1 &= b(s_0,\ s_1+1,\ s_2) - b(s_0,\ s_1-1,\ s_2), \\ n_2 &= b(s_0,\ s_1,\ s_2+1) - b(s_0,\ s_1,\ s_2-1), \end{aligned}$$

kde $[s_0, s_1, s_2]$ jsou souřadnice povrchového vzorku I_s a hodnota příznaku $b = (s_0, s_1, s_2)$ příslušnosti k povrchu nabývá hodnot 0 nebo 1. Metoda zachycuje tvar tělesa, ale na zaoblených površích opět vytváří artefakty v podobě vrstevnic.

Metoda gradientního stínování (*gray-level gradient shading*) předpokládá, že na povrchu dochází k největší změně hodnot vzorků, a proto opět předpokládá normálu k povrchu ve směru gradientu (tj. směru největší změny) hodnot intenzity v původním trojrozměrném objemu. Metoda vytváří díky velkému rozsahu intenzit hladké povrchy (viz obr. 16.3h). Gradient se odhaduje symetrickou diferencí hodnot v prostorové mřížce:

$$\begin{aligned} n_0 &= f(s_0+1,\ s_1,\ s_2) - f(s_0-1,\ s_1,\ s_2), \\ n_1 &= f(s_0,\ s_1+1,\ s_2) - f(s_0,\ s_1-1,\ s_2), \\ n_2 &= f(s_0,\ s_1,\ s_2+1) - f(s_0,\ s_1,\ s_2-1). \end{aligned}$$

Některé další metody, stejně jako detaily k výše uvedeným metodám, je možné vyhledat například v [Lich98]).

16.3.4 Integrace světla na dráze paprsku

Pokročilejší metody přímého zobrazování objemových dat kombinují metodu vrhání paprsku s osvětlením objemových dat. K určení barvy pixelu na obrazovce přispívá osvětlení všech objemových elementů na dráze příslušného paprsku. Společným rysem zde uvedených metod je zjednodušení spočívající v tom, že světlo není na cestě od světelných zdrojů k jednotlivým místům v objemu zastiňováno ostatními částmi objemu.

V následujícím textu se seznámíme s trojicí metod (Drebinova, Levoyova a Sabellova). První dvě umožňují zobrazovat v objemu poloprůhledné struktury i povrchy, třetí se soustřeďuje na zobrazování zářících průsvitných struktur.

Levoyova metoda

Levoy zobrazoval data z počítačového tomografu (CT), tj. objem skalárních hodnot, reprezentujících pohltivost (absorpci) rentgenových paprsků tkáněmi [Levo88]. Metoda chápe objem

jako pole voxelů a pracuje ve dvou krocích. Symbolem X budeme v této části označovat voxel o souřadnicích $[x, y, z]$.

V prvním kroku jsou veškerá data předzpracována ze dvou hledisek – z hlediska barvy a průhlednosti. Barva voxelů se určí dle Phongova osvětlovacího modelu, přičemž normála se získá odhadem gradientu symetrickou diferencí. Barvy se uloží do pomocného *pole barev* $C(X)$, resp. do trojice polí pro složky R, G a B.

Hodnoty neprůhlednosti voxelů (*opacity*) se určí klasifikací a uloží se do druhého pomocného pole – *pole neprůhledností* $\alpha(X)$. Postup klasifikace je značně závislý na aplikační oblasti a na tom, co konkrétně se má zobrazit.

Má-li být zobrazena prostá *izoplocha* o hodnotě h_c, vystačíme s jednoduchým porovnáním hodnoty voxelů se zadanou konstantou h_c. Voxelům, jejichž hodnota $h(x, y, z)$ je rovna h_c, přiřadíme neprůhlednost α_c, ostatním nulu. Šum a nelinearity při vzorkování lze potlačit tím, že voxelům s hodnotou $h(X)$ blízkou k h_c se přiřadí neprůhlednost $\alpha(X)$ blízká k α_c v závislosti na velikosti odchylky hodnot h a h_c.

Při zobrazování několika *objektů* (struktur), jimž odpovídají intervaly hodnot se středy $h_c(1)$ až $h_c(n)$, lze vyjít z obdobného principu a hodnotám h z intervalu $\langle h_c(i), h_c(i+1)\rangle$ přiřadit neprůhlednost z intervalu $\langle \alpha_c(i), \alpha_c(i+1)\rangle$ lineární interpolací (viz část 22.6.2). Tento způsob odhadu je vhodný tehdy, pokud se intervaly příslušející hodnotám $h_c(i)$ překrývají jen se sousedními intervaly $h_c(i-1)$ a $h_c(i+1)$.

Protože člověk vnímá obrysy a hrany intenzivněji než pozvolné přechody, je při vizualizaci výhodné zdůraznit rozhraní mezi objekty a potlačit vliv vnitřních voxelů. Toho lze dosáhnout vynásobením výsledné neprůhlednosti velikostí vektoru gradientu

$$\alpha'(X) = |\nabla h(X)|\ \alpha(X). \tag{16.1}$$

Ve druhém kroku se při rovnoběžném promítání vytváří výsledný dvojrozměrný obraz kombinací obou pomocných objemů $C(X)$ a $\alpha(X)$. Pro každý pixel je do objemu vyslán jeden paprsek. Ten prochází pole $C(X)$ a $\alpha(X)$ a akumuluje hodnoty intenzity (jasu) dle vzorce (16.2). Paprsek vstupuje do objemu voxelu s intenzitou (jasem) C_{in}, je částečně utlumen průchodem jeho objemem a získá část jeho barvy (obrázek 16.4). Vztah pro výpočet jedné barevné složky na úrovni voxelu má tvar:

$$C_{out} = C_{in}(1 - \alpha(X)) + C(X)\alpha(X). \tag{16.2}$$

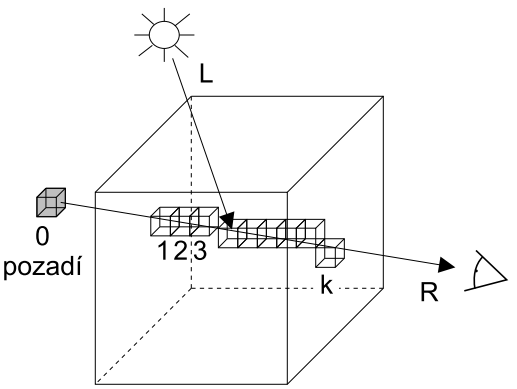
Není-li objem natočen, prochází paprsek středy voxelů kolmo na přední stěnu. Ve všech voxelech projde stejnou vzdálenost (rovnou délce jejich stěny), a proto lze počítat přímo s hodnotami C a α. Při pohledu z jiného úhlu nebo při jiné hustotě rastru obrazovky a pole voxelů procházejí paprsky každým voxelem jinak. Je nutné změnit způsob akumulace hodnot tak, aby zohlednil různou míru zasažení voxelu paprskem, a tedy i skutečnou délku dráhy.

Nejméně přesným řešením je přičítání hodnot všech voxelů zasažených paprskem, přesnější je objem převzorkovat. Paprsek při převzorkovávání prochází objemem s konstantním krokem a hodnoty $h(X')$ se dopočítávají např. trilineární interpolací z původních hodnot sousedních voxelů $h(X)$ (v 6okolí až 26okolí). Dále je třeba vyjádřit delší vzdálenost, kterou paprsek prochází voxelem, změnou hodnot neprůhlednosti.

Vzorec pro výpočet jedné barevné složky celkové intenzity po průchodu paprsku řadou k voxelů vznikne poskládáním k rovnic (16.2). Po úpravě se dá vyjádřit takto:

$$C(R) = \sum_{i=0}^{k} \left\{ C(R,i)\alpha(R,i) \prod_{j=i+1}^{k} (1 - \alpha(R,j)) \right\} \tag{16.3}$$

kde (R,i) je i-tý voxel podél paprsku R. Pozadí je reprezentováno nultým voxelem, který je neprůhledný, takže $C(R,0)$ je rovno barvě pozadí a neprůhlednost $\alpha(R,0) = 1$. Jednoduchý zápis vztahu 16.3 vychází z předpokladu, že $\alpha(R,j) = 0, j \geq k$.



Obrázek 16.4: Princip metody vrhání paprsku objemem složeným z voxelů

Porovnáme-li rovnici (16.3) s integrálem pro zobrazování objemů (10.27) v části 10.8, zjistíme, že obě obsahují dva hlavní členy. Jeden vyjadřuje barvu v místě vzorku, druhý popisuje utlumení paprsku při průchodu objemem mezi pozorovatelem a vzorkem. Vztah (16.3) tedy odpovídá odhadu integrálu pro zobrazování objemů. Zeslabení působí jen na cestě paprsku objemem od vzorku (voxelu) do oka pozorovatele, ale ne na cestě od zdroje světla k osvětlenému voxelu (dle předpokladu v úvodu kapitoly), světelný zdroj je vidět ze všech voxelů stejně, jako by okolí voxelu bylo dokonale průhledné.

Levoyův algoritmus je zaměřen na objemové vykreslování povrchů určených zvoleným kritériem, bez předchozí konstrukce těchto povrchů. Proto se barva C počítá gradientním stínováním myšlených povrchů ve voxelech.

Drebinova metoda

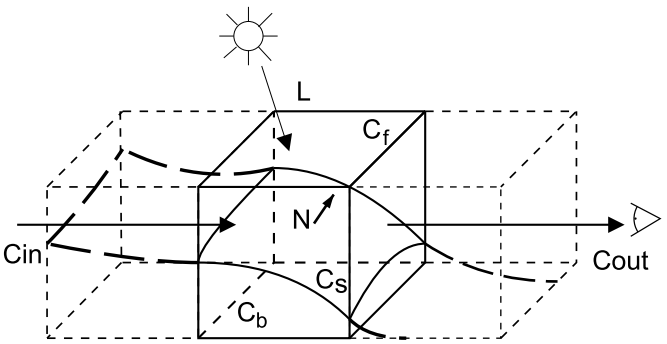
Drebinova metoda připouští, aby voxel obsahoval více druhů materiálů současně [Dreb88]. K tomu dochází tehdy, když voxelem prochází rozhraní mezi strukturami. Metoda pracuje dvoufázově. Ve fázi předzpracování se připraví několik pomocných polí atributů obdobně jako

u předchozí metody. Těmito poli jsou: pole procentních zastoupení materiálů, pole barev, pole hustot a pole velikostí gradientu.

Polí procentních zastoupení materiálů P (*material percentage volumes*) je tolik, kolik struktur je schopen rozlišit použitý klasifikační algoritmus. Přitom se rovněž předpokládá, že se intervaly hodnot příslušných k materiálům [tj. okolí hodnot $h_c(i)$] překrývají maximálně pro dvojice materiálů. Pomocná pole barev a neprůhledností jsou sdružena v jediném *poli barev*, kde každá barva C voxelu má čtyři složky $C = [\alpha R, \alpha G, \alpha B, \alpha]$. Barva voxelu, který obsahuje směs materiálů, se vypočte dle vzorce

$$C = \sum_{i=1}^{n} p_i C_i \ ,$$

kde n je počet materiálů ve voxelu, p_i je procentní zastoupení materiálu ve voxelu a C_i je barva materiálu vynásobená neprůhledností, tj. $C_i = [\alpha_i R_i, \alpha_i G_i, \alpha_i B_i, \alpha_i]$.



Obrázek 16.5: Drebinův „trojdílný“ stínovací model voxelu

Barevné složky jsou vynásobeny neprůhledností, což slouží k urychlení výpočtu [viz vztah (16.4] dále), zároveň se pro jednoduchost předpokládá nezávislost α na vlnové délce.

K detekci povrchů a výpočtům normál slouží *pole hustot*, které uchovává pro každý voxel hodnotu hustoty D. Pokud je ve voxelu obsaženo více materiálů s různými hustotami, spočítá se průměrná hustota D voxelu podle vztahu

$$D = \sum_{i=1}^{n} p_i \rho_i,$$

kde ρ_i je hustota přiřazená i-tému materiálu. Proces skládání barevných příspěvků se vyjadřuje pomocí tzv. „kompozičního operátoru *over*“, který je definován takto:

$$\begin{aligned} C \textbf{ over } C_{in} &= C + (1 - \alpha_C) C_{in}, \text{ kde} \\ C &= C(X)\alpha(X). \end{aligned} \tag{16.4}$$

Standardní vztah pro výpočet jedné barevné složky (16.2) pro jeden voxel přejde do tvaru:

$$C_{out} = C \textbf{ over } C_{in}. \tag{16.5}$$

Výsledný příspěvek voxelu k barvě paprsku se skládá ze tří „barevných oblastí“ (viz obr. 16.5): oblasti před předpokládaným povrchem, označené jako C_f (*front*), tenké povrchové oblasti C_s (*surface*) a oblasti za povrchem C_b (*back*).

$$C_{out} = (C_f \textbf{ over}(C_s \textbf{ over}(C_b \textbf{ over} C_{in}))) \tag{16.6}$$

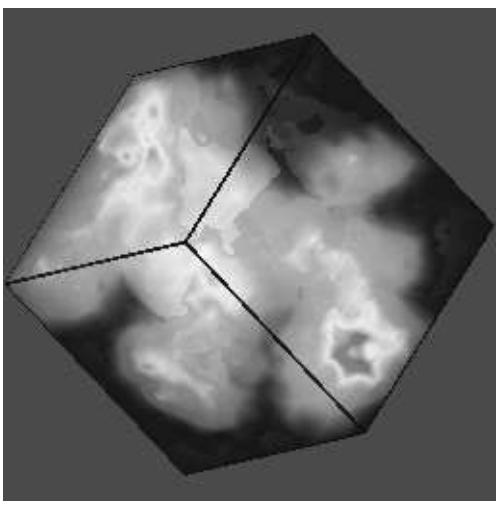
Velikost gradientu $|\nabla h(X)|$, uložená v *poli vlivu povrchu* (*surface strength*) $S = |\nabla h(X)|$, se používá jako váha parametru C_s. Při zpracovávání rovinných obrazů v počítačovém vidění se používá obdobný pojem pro velikost gradientu hran, tzv. *velikost hrany* [Hlav92].

V místech s malou velikostí gradientu S se proto povrch neuplatní. Takový voxel zrcadlově neodráží světlo přicházející ze světelných zdrojů a chová se jako homogenní poloprůhledný gel, který pouze moduluje světlo přicházející ze sousedního voxelu. Pro velké S naopak převáží vliv světla odraženého od předpokládaného povrchu. Levoy naproti tomu jednoduše používá jedinou barvu $C = C_s$.

Sabelova metoda zářících částic

Sabela [Sabe88] zobecnil osvětlovací model tak, aby umožnil simulovat i světlo přicházející z poloprůhledných objektů bez definovaného povrchu.

V Sabelově přístupu je voxelové pole chápáno jako pole částic (*varying density emitter*), vyzařujících a rozptylujících světlo. Prostorová hustota částic ϱ, která je přímo úměrná vstupní skalární hodnotě v daném místě, je použita ve verzi Kajiyovy a Herzenovy jasové rovnice [Kaji84] k výpočtu jasu podél paprsku.



Obrázek 16.6: Obraz objemu zářících částic vytvořený Sabelovou metodou

Metoda se soustřeďuje na modelování zakrytí vzdálenějších částí objemu bližšími částmi, ale opět záměrně zanedbává výpočet stínů a odlišné rozptylování pro různé vlnové délky, neboť předpokládá, že by tyto jevy komplikovaly vnímání změn hustoty.

Při vržení paprsku se akumuluje čtveřice hodnot: kromě celkové intenzity světla [zeslabované absorbcí stejně jako v (10.27)] i maximální hodnota intenzity na dráze paprsku, vzdálenost tohoto maxima podél paprsku a těžiště hodnot voxelů (hustot) podél paprsku. Výsledná barva se vyjadřuje v modelu HSV (část 1.2.1). Barevný tón H se nastavuje podle hodnoty maxima světla podél paprsku (např. tabulkou, která přiřazuje určité

velikosti maxima hodnotu barevného tónu), sytost S, která slouží jako prostředek zvyšování vjemu vzdálenosti, se nastaví buď uměrně vzdálenosti maxima nebo úměrně vzdálenosti těžiště a jasová hodnota V se rovná intenzitě světla po průchodu paprsku objemem. Sabelova metoda dává kvalitní výsledky simulace zářících objemů (viz obr. 16.6).

16.3.5 Projekční metody

Uvedeme zástupce zobrazování objemů pracující na principu *projekce*. Tyto metody sestavují obraz z promítnutých hodnot jednotlivých voxelů či buněk.

Naplácávání

Westover uveřejnil metodu, kterou podle podobnosti s házením sněhových koulí na skleněnou plochu nazval naplácávání (*splatting*)[West90]. Všechny voxely se postupně promítnou na obrazovku. Příspěvek voxelu se rozdělí mezi pixely tak, že nejvíce se připočte v místě středu zásahu, a směrem od něj klesá.

Algoritmus v první fázi nalezne nejbližší stěnu objemu, tj. takový ortogonální řez, který je nejblíže k průmětně, a v něm nejbližší voxel. Poté se ve druhé fázi postupně zpracovávají voxely počínaje nejbližším k nejvzdálenějšímu („po řádcích“), počínaje nejbližší stěnou k nejvzdálenější. Vždy se vyhodnotí jejich barva C a neprůhlednost α dle lokální hodnoty a gradientu a výsledek se promítne na obrazovku a připočítá k zasaženým pixelům. Protože se pracuje po vrstvách, vzniká výsledný obraz postupně.

Průmět voxelu na obrazovku, který je nazván *otiskem* (*footprint*), je při rovnoběžném promítání stejný pro všechny voxely a má tvar kruhu. Rozložení příspěvků pixelům uvnitř otisku má obvykle charakter dvojrozměrného Gaussova normálního rozložení (viz část 22.5.2) a lze ho předpočítat do vyhledávací tabulky. Průměr otisku, tj. i počet zasažených pixelů jedním voxelem, se určí tak, aby průmět objemu pokryl celou obrazovku. Proto může vzniknout velký obraz i z malého objemu.

Jednotlivé voxely jsou postupně (nejprve bližší, poté vzdálenější) promítány na obrazovku a části jejich příspěvků se podle tabulky otisku připočítávají k zasaženým pixelům. Dosáhne-li celková neprůhlednost v pixelu hodnoty 1, příspěvky dalších voxelů se v něm již nepřičítají.

Při středovém promítání se musí otisk, který má tvar elipsy, počítat pro každý voxel zvlášť.

16.3.6 Zlepšení interpretace dat

Interpretaci dat lze zkvalitnit důsledným využíváním prostředků zvyšujících prostorovost vjemu (*depth cues*), např. poklesu jasu se vzdáleností či použitím stereovidění. Podstatným příspěvkem ke zlepšení interpretace je možnost interaktivně manipulovat s daty.

Další možností je snížení množství dat, kterými při vizualizaci uživatele zahrnujeme. Množství nadbytečných dat lze zmenšit *segmentací*, při níž odstraníme nezajímavé části objemu a zpracujeme pouze oblast, která uživatele zajímá. Postup odstraňování nadbytečných dat úzce souvisí s technikami klasifikace dat a je těsně svázán s aplikační oblastí.

Mezi nejjednodušší metody segmentace patří *prahování*, kterým izolujeme struktury, jejichž hodnota přesahuje hodnotu prahu (nebo leží v daném intervalu) a ty promítáme ze zvoleného směru pohledu v kombinaci s morfologickými operacemi. Při zobrazování tenkých výrazných struktur, jakými jsou například tenké cévy, jejichž průměr je srovnatelný s velikostí voxelu, a tudíž příliš malý pro algoritmus Marching cubes, se používá jednoduché zobrazení maximální hodnoty podél paprsku.

Dále lze používat tzv. *maskovací objemy* (*matte volumes*), pomocí nichž lze jednoduše implementovat operace typu spojování objemů, pokles jasu se vzdáleností (*depth cueing*) a odstraňování nezajímavých částí, tj. vyloučením některých voxelů z dalšího zpracování.

Zmíněné postupy a techniky byly pouze příkladem různorodých přístupů, které závisejí na aplikační oblasti a na konkrétní řešené úloze. Počítačová grafika neposkytuje pro vizualizaci jedno univerzální řešení, ale nabízí široké spektrum technik, ze kterých si uživatelé vybírají podle potřeby.

Kapitola 17

Nefotorealistické zobrazování

V počítačové grafice se termínem *fotorealistické zobrazování* označují metody, které se snaží zobrazit virtuální trojrozměrnou scénu nebo model tak, aby se výsledek podobal fotografii scény skutečné. *Nefotorealistické zobrazování* zahrnuje metody, které o dosažení fotorealismu záměrně neusilují. Je důležité, že tento odklon od fotorealismu je záměrný a dobrovolný – jinak by bylo možno za nefotorealistické považovat i metody, které se sice snaží fotorealismu blížit, ale z nějakého důvodu jej nedosahují. V anglické literatuře se pro fotorealistické zobrazování používá zkratka *PR* (*Photorealistic Rendering*) a pro nefotorealistické zkratka *NPR* (*Non-Photorealistic Rendering*).

Problematika *NPR* je jednou z novějších oblastí počítačové grafiky a rychle se vyvíjí. Zde se omezíme na stručný, motivační přehled. Více informací lze nalézt v [Gooc01a, Stro02].

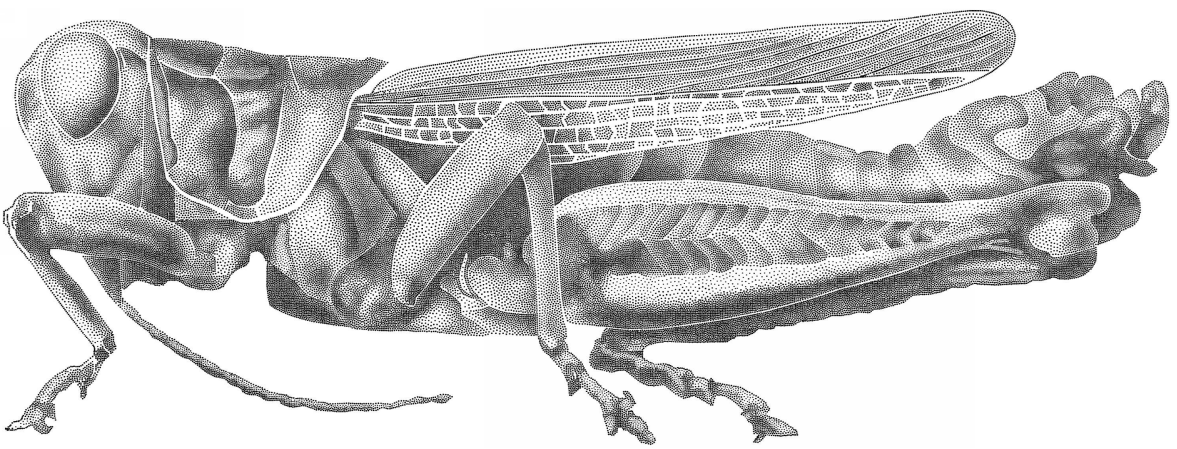
17.1 Výhody NPR

Metody *PR* usilují o zobrazení složitých a detailních scén v kvalitě takřka nerozeznatelné od fotografie. Přesto existuje řada situací, ve kterých je fotorealismus nedosažitelný, nepotřebný, či dokonce nežádoucí. Uveďme několik důvodů pro použití metod *NPR*.

Fotorealismus je nepřehledný. Scény jsou často příliš složité a je obtížné se v nich vyznat. Nefotorealistické zobrazování umožňuje zanedbat nepodstatné a nechat důležité části vyniknout. Názorné návody na údržbu, orientační mapy nebo anatomické atlasy vyžadují přesně tento druh zobrazení. Jindy je naopak potřeba detaily nepotlačovat, ale naopak zvýraznit. Například pomocí šrafování lze lépe vystihnout zakřivení povrchů, které by ani při osvětlení různými virtuálními zdroji světla nemuselo být dostatečně patrné. Jiným případem jsou technické ilustrace, které používají speciální techniky pro zobrazení lesklých ploch, aby mohl jejich tvar lépe vyniknout [Gooc98].

Vizualizace informací. Scéna může obsahovat kromě informací o vzhledu objektů i další údaje, které je třeba zobrazit. Těmito údaji může být například teplota, tlak, rychlost pohybu, nebo třeba informace o přesnosti dat, času jejich pořízení apod. Tyto dodatečné údaje je možno zobrazit pomocí široké škály stylů, které *NPR* nabízí. Techniky *NPR* se proto úspěšně používají pro přehlednou vizualizaci vědeckých dat (viz též část 16). Takové zobrazení může být efektivní právě díky schopnosti zvýraznit důležité informace na úkor nedůležitých.

Přizpůsobení se stylu. *NPR* lze výhodně kombinovat s ruční prací. Například při tvorbě animovaných filmů lze na počítači vytvořit složité scény (krajinu v pozadí, města apod.), které se pak kombinují s ručně kreslenými objekty, např. postavami hrdinů příběhu. Protože by realisticky zobrazené prostředí působilo vedle kreslených částí rušivě, algoritmy počítačové grafiky se upravují tak, aby jimi vytvořené obrazy odpovídaly co nejvíce stylu používanému animátory.



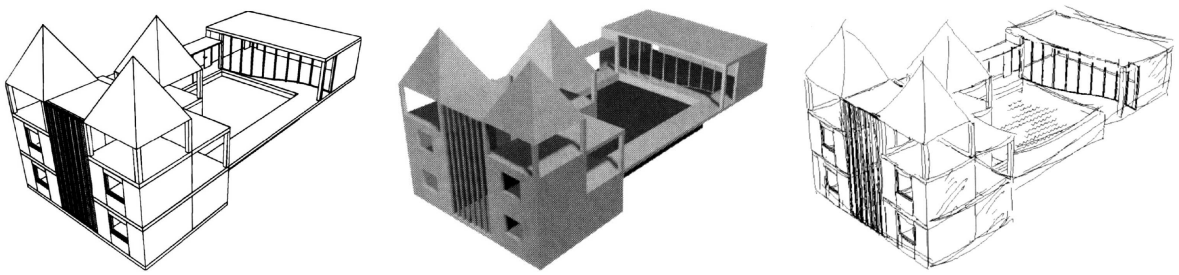
Obrázek 17.1: Simulovaná technika tečkování (*stippling*), používaná zejména v encyklopediích (obrázek poskytl O. Deussen, University of Konstanz)

Média nedovolující fotorealistické zobrazení. Použití technik *NPR* může být vynuceno přímo médiem. Některé knihy jsou z ekonomických důvodů omezeny na používání černobílých ilustrací. Ručně tečkované nebo šrafované obrázky vypadají lépe, než fotografie zobrazené polotónováním (viz obrázek 17.1).

Nedostatek dat pro PR. Často není dostupný dostatečně detailní popis scény. To je případ scén, které jsou teprve vytvářeny a navrhovány, takže je pro ně hrubé, skicovité zobrazení dostatečné. Jiným příkladem může být zobrazování složitých modelů, jako jsou travnaté povrchy, vlasy

nebo srst [Kowa99]. Modelovat a následně vykreslovat každé stéblo trávy je velmi náročné. Přitom může stačit znázornit trávník pomocí několika vhodně umístěných čar.

Příznivé psychologické působení a atraktivnost NPR. Pokud je potřeba zobrazovat nedokončené návrhy a koncepty, je *PR* nevhodné z psychologického hlediska. Fotorealisticky zobrazená scéna totiž bývá považována za konečný, definitivní stav a díky tomu může vyvolat nepříznivý dojem. Skicovité zobrazení dává jasně najevo, že scéna je nehotová (viz obrázek 17.2). Náčrty navíc motivují ke spolupráci na tvorbě díla, protože je možno do nich dále kreslit. Někdy je nefotorealistický styl zobrazení považován za atraktivnější, než možné fotorealistické zobrazení. *NPR* může do jisté míry napodobit práci umělce a zvýšit tak estetický dojem z generované scény. Obrázek navíc často zaujme více, než pouhá fotografie, což umožňuje použít *NPR* například v reklamě.



Obrázek 17.2: Skicovité zobrazení (vpravo) může být vhodnější než strohý počítačový model, ať již v podobě vektorové kresby nebo stínovaný (obrázky poskytli T. Strothotte a S. Schlechtweg, University of Magdeburg.)

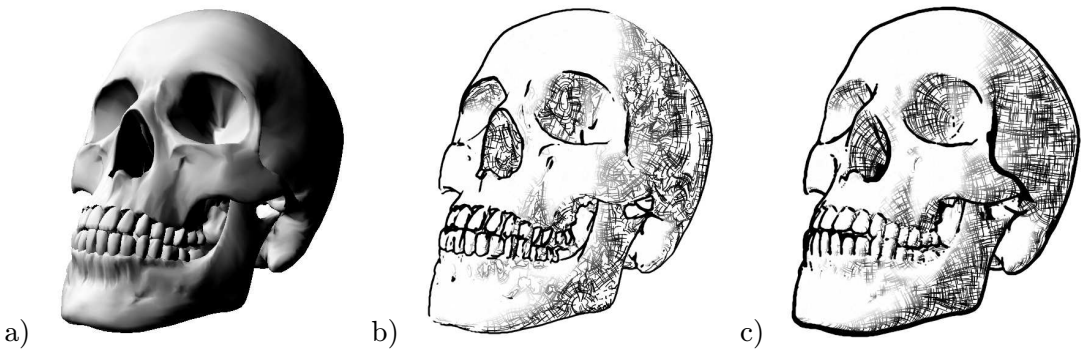
17.2 Rozdělení metod NPR

Protože motivace pro použití *NPR* je velmi různorodá, metody řeší řadu velmi rozdílných problémů. V tomto textu se omezíme na základní členění podle druhu vstupních a výstupních dat a dle rozsahu interakcí s uživatelem.

Druhy vstupních dat. Typickým vstupem pro metody *NPR* jsou klasická 3D data, nejčastěji zadaná povrchovou reprezentací nebo objemově. Kromě 3D vstupu je možno použít i dvojrozměrná data, která jsou vhodná pro zpracování pomocí obrazových filtrů (viz kapitola 4.6.4).

Na půli cesty mezi 3D a 2D vstupy jsou tzv. 2,5D data. Jsou to většinou 2D obrazy s dodatečnou informací o hloubce viditelných povrchových bodů. Taková data jsou výhodná, protože se s nimi snadno manipuluje a umožňují dosáhnout mnoha efektů, které jsou jinak doménou 3D.

Rozdíl mezi 3D a 2D vstupními daty ukazuje obrázek 17.3. Vstupní 3D model je využit různými způsoby. Na obrázku a) je nejprve zobrazen běžným stínováním, tedy technikou *PR*. Dva další obrázky vznikly pomocí metod *NPR* a navzájem se liší v určení směru šrafovacích čar. V případě b) je směr určen jako derivace intenzity obrázku a), vstupem jsou tedy pouze 2D data. V případě c) je směr šrafování odvozen z 3D modelu a výsledný obraz proto lépe zachycuje tvar objektu a obsahuje méně artefaktů vzniklých šrafováním.



Obrázek 17.3: Srovnání 2D a 3D vstupních dat: a) obraz *PR* vzniklý pomocí stínování, b) šrafování *NPR* zpracovávající jako vstup 2D obraz vlevo, c) stejný typ šrafování *NPR* s využitím dat původního 3D modelu (obrázky poskytl R. Ženka, ČVUT v Praze)

Kromě statických dat je možno používat i animace, resp. video. Tento druh vstupu lze použít i v případě, kdy je na výstupu očekáván jeden statický obrázek – *NPR* umožňuje použít tzv. *motion lines* pro vizualizaci směru a rychlosti pohybu [Masu99]. Specifickým druhem vstupu jsou také data získaná interakcí s uživatelem.

Rozsah interakcí s uživatelem. Škála aplikací *NPR* sahá od plně interaktivních programů s minimální inteligencí (uživatel určuje jednotlivé tahy sám), přes poloautomatické (uživatel napovídá, kde a jak má program kreslit), až k plně automatickým přístupům nevyžadujícím interaktivní zásahy uživatele. Interaktivní kreslicí programy používají většinou tradiční vstupní zařízení, jako jsou myši nebo tablety. Mezi méně obvyklá zařízení pak patří trojrozměrné snímače pohybu, obohacené případně o zpětnou silovou vazbu.

Algoritmy s nízkým rozsahem interakcí nechávají uživatele řídit proces zobrazování na vyšší úrovni, než jsou jednotlivé tahy daného nástroje. Uživatel si může vybírat mezi různými

styly, případně nakreslením části obrázku dodá programu informaci o tom, jak si představuje výsledek. Program nechává uživatele soustředit se na otázky estetiky a sám automatizuje únavné, opakující se činnosti, jako je například šrafování.

Metody, které nevyužívají interakcí s uživatelem, bývají vybaveny heuristikami, s jejichž pomocí dokáží odhadnout zajímavé oblasti scény, nastavit správný kontrast, šrafovat a stínovat na podobných místech jako uživatel, atd. Tyto postupy lze odvodit z rozboru určitého uměleckého stylu, ať se soustředíme na konečný vzhled kresby či způsob jejího vytvoření.

Generované výstupy. Výstupem metod *NPR* je většinou dvojrozměrný rastrový, případně vektorový obrázek. Podobně jako v úlohách řešení viditelnosti (část 11), i zde můžeme nalézt metody pracující v *prostoru objektů*, tj. metody pracující přímo s geometrickým popisem objektů, a tedy přesné a nezávislé na měřítku, a metody pracující v *prostoru obrazu*, které provádějí výpočty v mřížce pixelů či subpixelů. Možné jsou i metody hybridní, které používají bitmapu, nicméně generují vektorové výstupy. Obrazové a hybridní metody jsou v současné době nejoblíbenější, protože umožňují využít grafických akcelerátorů.

Metody pro generování animací si nevystačí jen s opakovaným použitím statické metody, zejména kvůli zachování návaznosti mezi jednotlivými snímky. Problematické je zejména umísťování šrafovacích čar nebo teček, případně jiných artefaktů, které souvisejí pouze se zobrazením a nemají podklad ve vstupních datech modelu. Při naivní implementaci dochází k rušivému náhodnému pohybu, objevování a mizení těchto artefaktů.

Časová koherence se v praxi řeší buď svázáním čar s konkrétním místem na obrázku (pak se často projeví tzv. *shower door* efekt – animace vypadá, jako by byla sledována přes hrbolaté sklo v koupelně, neboť čáry na popředí se nepohybují a pouze mění své atributy) nebo čáry sledují pohyb ve scéně, případně se jednotlivé tahy nástrojem svážou přímo s povrchem zobrazovaného modelu (tehdy připomínají texturu přilepenou na povrch). Žádné řešení není ideální – artefakty se buď pohybují příliš chaoticky, nebo naopak příliš uspořádaně, což z nich činí samostatné entity a ruší vjem vlastního objektu.

17.3 Aplikace NPR

Fyzikální simulace nástrojů

Škála nástrojů, jejichž činnost je možno věrně napodobit pomocí počítače, je relativně široká – sahá od tužek, pastelů a uhlů přes olejové a vodové barvy až k rydlům pro simulaci mědirytin a dřevorytů. Kvalitní simulace přitom musí vzít v úvahu nejen vlastní nástroj a pigment, ale i vlastnosti média. Obtížná je zejména simulace difúze kapalin, ke které dochází při malbě vodovými barvami a která vyžaduje objemový model struktury papíru.

Simulace tradičních malířských technik

Cílem těchto metod není napodobování činnosti konkrétního nástroje, ale napodobení práce malíře, který daný nástroj používá. Malířské techniky používají široké, relativně nepřesné nástroje, kterými se vyplňují celé plochy (viz obrázek 17.4). Hlavním cílem je estetické působení výsledného díla a vytvoření dojmu ruční práce. Vzhledem k nepřesnostem malby většinou není potřeba mít kvalitní vstupní data, často stačí i hrubě segmentované 2D obrázky.

Pro určení pohybu štětce se používají buď informace získané přímo parametrizací 3D modelu nebo se využívá gradientů a nespojitostí v podkladovém obrázku. Byly popsány i pokročilejší metody využívající principů strojového vidění [DeCa02].

Obrázek 17.4: Fotografie a podle ní vygenerovaná malba. Hlava vpravo byla pomocí snímače směru pohledu automaticky označena jako místo, na které se zejména soustřeďuje pozornost, a proto je zdůrazněna jemnější kresbou, zatímco rušivé pozadí je potlačeno (obrázky poskytli A. Santella a D. DeCarlo, Rutgers University, New Brunswick)

Složitější metody napodobují i zpětnou vazbu – po nakreslení části obrázku jsou schopné se na něj „podívat“ a opravit jej [Curt97]. Pomocí zpětné vazby lze automaticky upravovat barvy pro vyšší kontrast obrázku; byly popsány i metody pro automatickou kompozici, zajišťující vyvážené umísťování modelů do obrázku [Gooc01b].

Simulace kreseb a rytin

Při této simulaci je výstupem síť relativně tenkých čar, na rozdíl od vyplněných ploch získaných malířskými technikami. Čáry je možno popsat vektorově nebo je vykreslit do bitmapy. Vhodné nástroje používané v této skupině jsou zejména ty, které zanechávají úzkou, ostře ohraničenou stopu – jako je tužka, uhel, pero, případně rydlo. Protože takto nelze jednolitě vyplňovat souvislé plochy stejně jako štětcem, je používáno šrafování, případně tečkování.

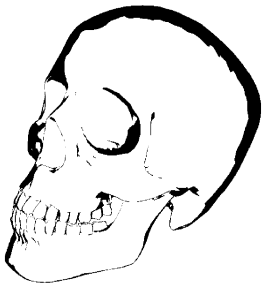
Obrázek 17.5: Různé styly kresby (obrázky poskytl C. Curtis, University of Washington)

Dobře voleným umístěním šrafovacích čar je možné zvýraznit tvar objektu. Jedním z často řešených problémů je proto správné umístění šrafovacích čar pro co nejnázornější vyjádření tvaru. Čáry mohou naznačit i materiál, případně druh povrchu. Složité povrchy, jako je tráva nebo srst, je možno tímto způsobem zobrazovat velice efektivně. Jiné techniky se naopak zaměřují na dosažení estetického dojmu – na obrázku 17.5 byly použity různé styly čar pro naznačení duševního rozpoložení postavy.

Při vykreslování je snaha minimalizovat počet použitých čar. Důvodem je kromě urychlení zobrazování také vyšší přehlednost výsledného obrázku. Zavádí se pojem *ekonomie čáry*. Čáry nesoucí nejvíce informace jsou zobrazovány nejvýrazněji, méně důležité jsou zanedbány. Vzhledem k tomu, že v obrázcích jsou většinou nejdůležitější siluety objektů, je jejich detekci věnována zvláštní pozornost [Sous03].

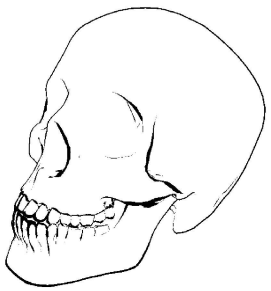
Detekce siluet

Pojmem *silueta* se v počítačové grafice rozumí křivka ležící na povrchu modelu, která tvoří hranici mezi plochou přivrácenou k pozorovateli a plochou odvrácenou. Pozice siluet je tedy závislá na směru, ze kterého je model pozorován. Tato definice popisuje něco jiného, než je běžně chápaný význam slova silueta jakožto vnější obrys objektu bez jakýchkoliv detailů. Silueta v počítačové grafice je tvořena obrysovými hranami, uvedenými v části 11.1. Siluety jsou důležité pro zobrazování a proto byla vyvinuta celá řada algoritmů pro jejich detekci. Tyto algoritmy řeší dva problémy – výpočet pozic siluet a řešení jejich viditelnosti. Tři z nejjednodušších algoritmů, pracující v prostoru obrazu, jsou uvedeny na konci této kapitoly. Ukázkové obrázky poskytl R. Ženka, ČVUT v Praze.



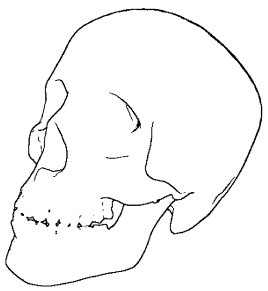
Algoritmus 17.1: NPR s pomocí prahování

1. Umísti bodový zdroj světla L do polohy kamery.
2. Nastav materiál všech těles na bílý, čistě difúzní.
3. Zobraz scénu osvětlenou pouze pomocí zdroje L.
4. Převeď výsledný obraz prahováním na černobílý.



Algoritmus 17.2: NPR s pomocí přivrácených a odvrácených ploch

1. Rozděl plochy scény na přivrácené a odvrácené.
2. Vykresli přivrácené plochy bíle.
3. Posuň model blíže k rovině obrazovky.
4. Vykresli odvrácené plochy černě (pomocí paměti hloubky).



Algoritmus 17.3: NPR s vyhledáváním změn hloubky

1. Vyhodnoť scénu pouze pomocí paměti hloubky.
2. Načti mapu hloubek.
3. Pomocí technik hledání hran (viz kapitola 4.6.2) nalezni na mapě místa s výraznou změnou hloubky a vykresli je.

Část D

ANIMACE A VIRTUÁLNÍ REALITA

Předposlední oddíl knihy se skládá ze tří kapitol. První z nich se zabývá technikami a metodami, které se používají v počítačové animaci. Seznámíme se zde s definicí animačních křivek a se zpětnou a přímou kinematikou. Na tyto algoritmy navazují metody pro animace postav.

Další kapitola je věnována algoritmům pro zobrazování rozsáhlých scén. Důraz je kladen na techniky umožňující zpracování v reálném čase.

Poslední kapitola tohoto oddílu je zaměřena na virtuální realitu. Popíšeme rozdělení aplikací virtuální reality a speciálními postupy, které se v ní uplatňují. Tato část také krátce pojednává o VRML a X3D, formátech pro trojrozměrnou virtuální realitu na webu. Prostorový zvuk, který je nedílnou součástí především počítačových her, tento oddíl uzavírá.

Kapitola 18

Počítačová animace

V této kapitole se seznámíme se základy trojrozměrné počítačová animace. Jejím nejjednodušším příkladem je zobrazování dynamické scény, jehož výsledkem je posloupnost obrazů. Objekty, které jsou součástí scény se mohou pohybovat, stejně jako kamera, která scénu snímá. V diskrétních časových okamžicích sestavíme model scény, umístíme do něj kameru a model zobrazíme. Tento postup je podobný klasické animaci, kdy je scéna taktéž snímána kamerou v diskrétních časových krocích.

Výhodou počítačové animace je možnost aplikace algoritmů simulujících fyzikální jevy, jako je detekce kolizí, větru, proudění vody, dále algoritmy simulace davů i virtuálních jedinců, aj. Počítačový animátor tak nemusí například simulaci pohybujícího se míče provádět ručně, nastavením jeho polohy a orientace v každém snímku. Místo toho pouze nastaví počáteční podmínky a spustí simulaci. Mezi nové techniky patří simulace autonomního pohybu davů, hejn ptáků či stád zvířat. Zadáním počátečních podmínek a spuštěním simulace získá uživatel realistickou animaci. Jedním z nejužitečnějších pomocníků jsou algoritmy přímé a inverzní kinematiky, které se používají k simulaci pohybu vzájemně spojených nepružných struktur.

Pokud hovoříme o počítačové animaci máme na mysli většinou pohyb. Pohyb však může mít mnoho odlišných forem; od posouvání hrníčku po stole, přes třepetání vlajky ve větru, až po tekoucí vodu. V počítačové animaci nemáme k dispozici jednotný popis všech forem pohybu, a proto existuje mnoho algoritmů, které (každý po svém) řeší dílčí úlohy počítačové animace.

Počítačovou animaci můžeme rozdělit na nízkoúrovňovou a vysokoúrovňovou. Nízkoúrovňová počítačová animace má blízko k teorii křivek (kapitola 5), neboť se v ní zabýváme reprezentací pohybu objektu po spojité dráze, jeho rychlostí, orientací, směrem atd. Vysokoúrovňová počítačová animace pracuje s pojmy jako je kolize, „jde po“, „kouká na“, „jede směrem“, apod. Tyto operace je možno skládat z nízkoúrovňových operací. Například chůzi můžeme popsat jako posloupnost kroků, pro simulaci pohybu koulí na kulečníkovém stole stačí popis několika

základních kolizí aj. Výhodou rozdělení operací na vyšší a nižší je možnost vytvářet knihovny pohybů, které slouží jako stavební kameny vyšší úrovně animace. Můžeme například vytvořit knihovnu gest, kroků či kolizí. Tyto základní funkce mohou být parametrizovatelné. Potom můžeme používat funkce jako „utíká rychle“, „utíká ladně“, apod. V této kapitole budeme nejprve hovořit o klíčování a uvedeme nejčastěji používané animační křivky. Ve druhé části se zmíníme o přímé a inverzní kinematice a o detekci kolizí.

18.1 Nízkoúrovňová počítačová animace

18.1.1 Klíčování

Pojem klíčování (*keyframing*) pochází z dílen Walta Disneye, kde označoval následující pracovní postup. Při tvorbě animovaného filmu měl nejdůležitější úlohu hlavní animátor, neboli choreograf. Ten udával tón celému filmu tím, že vymyslel postavičky a maloval jejich pohyb. Protože malování celé sekvence je zdlouhavé a jedná se o rutinní práci, bylo úkolem choreografa vytvořit nejdůležitější (klíčové) snímky. Zbývající mezisnímky (*in-betweens*) malovali animátoři určení speciálně pro tuto práci. Přirozeně, že první snaha tvůrců programů pro dvourozměrnou počítačovou animaci vedla právě k odstranění ručního malování mezisnímků.

Postup, kdy animátor zadává klíčové snímky a program dodělává zbývající práci, se začal používat i ve trojrozměrné počítačové animaci. Animátor zadává klíčové pozice v podstatě čehokoliv: poloh objektů, úhlů, barev, textur, průhlednosti, aj. Úkolem programu je automatické generování mezipoloh. Výsledkem je popis scény, který slouží jako vstup nějakého rendereru.

18.1.2 Animační křivky

Křivky se v počítačové animaci používají především pro definici dráhy a orientace objektů. Z pohledu počítačové grafiky spočívá úloha klíčování pohybu v nalezení interpolační křivky (viz obrázek 5.6 na straně 181). Určení pohybu se skládá z následujících kroků, nejprve se definuje dráha objektu, určí se změny rychlosti a nakonec se určí orientace objektu.

Poloha objektu v konkrétním snímku se získá interpolací zadaných klíčových poloh a rychlost se určí pomocí parametrizace křivky, která těmito body prochází. Orientace objektu se určí z lokálního souřadnicového systému, který se natáčí a pohybuje po dráze určené křivkou.

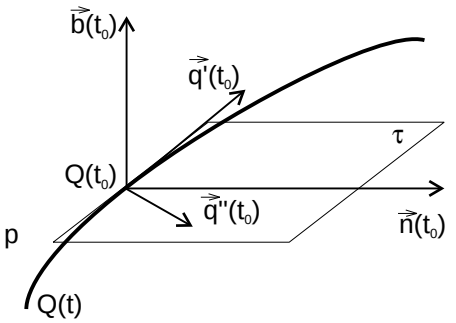
Změna parametrizace

Změnou parametrizace křivky $Q(t) = [x(t), y(t), z(t)]$ rozumíme nahrazení parametru t jiným parametrem, který je zadán jako funkce $t^* = t^*(t)$. Funkce $t^*(t)$ musí být *přípustná*, to jest ryze monotónní na definičním oboru t. Nové vyjádření $Q(t^*) = [x(t^*), y(t^*), z(t^*)]$ popisuje

křivku, která má stejný tvar, ale tečný vektor (vektor okamžité rychlosti) v bodě $Q(t^*)$ má jinou velikost. Tečný vektor je závislý na parametrizaci, tečna na parametrizaci závislá není.

Určení orientace

Orientaci objektu, jehož poloha je v čase t dána křivkou $Q(t)$, můžeme určit z vektorů rychlosti a zrychlení. Tečný vektor $\vec{q'}(t)$ k parametricky vyjádřené křivce získáme derivací složek křivky $Q(t)$ podle parametru t. Podobně vektor druhé derivace (zrychlení) získáme druhou derivací složek křivky $Q(t)$ po částech a tento vektor označíme $\vec{q''}(t)$. Oskulační rovinou křivky v bodě $Q(t_0)$ rozumíme rovinu označenou τ určenou bodem $Q(t_0)$, tečným vektorem $\vec{q'}(t_0)$ a vektorem druhé derivace $\vec{q''}(t_0)$ (viz obrázek 18.1). Binormála je vektor $\vec{b}(t_0)$, který prochází bodem $Q(t_0)$ a je kolmý k oskulační rovině τ. Určíme ji z tečného vektoru a zrychlení



Obrázek 18.1: Tečna $q'(t)$, binormála $b(t)$, hlavní normála $n(t)$ a oskulační rovina τ ke křivce $Q(t)$

$$\vec{b}(t_0) = \vec{q'}(t_0) \times \vec{q''}(t_0).$$

Hlavní normála $\vec{n}(t_0)$ je kolmá k tečnému vektoru $\vec{q'}(t_0)$ a je rovnoběžná s oskulační rovinou τ.

$$\vec{n}(t_0) = \vec{b}(t_0) \times \vec{q'}(t_0).$$

Tečný vektor $\vec{q'}(t_0)$, normála $\vec{n}(t_0)$ a binormála $\vec{b}(t_0)$ jednoznačně určují orientaci pohybujícího se objektu. Orientaci nelze určit v inflexním bodě křivky, kde platí $\vec{q'}(t_0) = k\ \vec{q''}(t_0); k\ \neq 0$. Obyčejně se použije orientace z nejbližšího bodu, kde je definována. V inflexním bodě také dochází k přesunu vektoru zrychlení na opačnou stranu křivky, což při použití výše uvedených vztahů, způsobí překlopení lokálního souřadnicového systému.

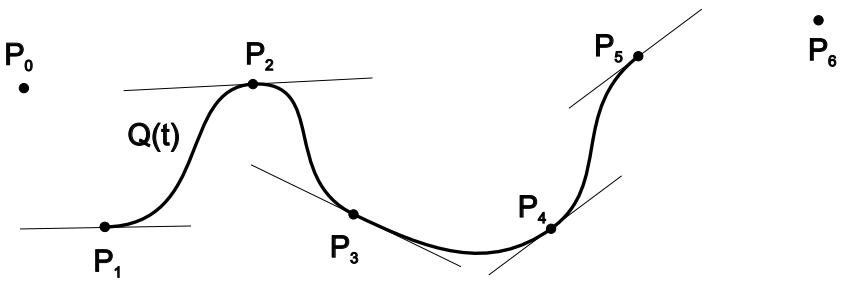
Křivky Kochanek-Bartels

Základní úlohou počítačové animace je interpolace bodů spojitou křivkou. Systém, který poskytuje interpolační křivku, by navíc měl poskytovat určitou úroveň řízení v zadaných bodech. Vhodným interpolačním schématem jsou Hermitovské kubiky (5.12), jejichž segment je definován dvěma body a dvěma tečnými vektory v nich. Z těchto křivek vycházejí animační křivky pojmenované podle svých autorů křivky *Kochanek-Bartels* [Koch84]. Oproti Hermitovským křivkám tyto křivky navíc poskytují možnost řízení průběhu interpolovanými body, konkrétně

umožňují definovat napětí (*tension*), spojitost (*continuity*) a šikmost (*bias*). Proto se jim také někdy říká *TCB křivky*.

Nejprve uvedeme vztah pro výpočet křivky, poté vysvětlíme význam jednotlivých koeficientů. Křivka je zadána posloupností bodů $P_0, P_1, \ldots, P_n$ a řídicími koeficienty a_i, b_i a c_i v každém z tzv. vnitřních bodů $P_1, P_2, \ldots, P_{n-1}$. Krajní body se používají pro definici průběhu křivky, ale ta jimi neprochází. Pro výpočet křivky se použijí rovnice uvedené v odstavci 5.3.1. Pro jejich určení je důležitá velikost tečného vektoru v každém řídicím bodě. Při výpočtu TCB křivek se pro každý řídicí bod definují dva vektory. Jeden pro segment nalevo (ve smyslu rostoucího parametru t) a druhý pro segment napravo od uzlu. Těmto vektorům se říká vstupní a výstupní tečný vektor a označujeme $\vec{l}_i$ vstupní vektor a $\vec{r}_i$ vektor výstupní.

$$\begin{aligned}\vec{l}_i &= \frac{(1-a)(1+b)(1-c)}{2}(P_i - P_{i-1}) + \frac{(1-a)(1-b)(1+c)}{2}(P_{i+1} - P_i)\\ \vec{r}_i &= \frac{(1-a)(1+b)(1+c)}{2}(P_i - P_{i-1}) + \frac{(1-a)(1-b)(1-c)}{2}(P_{i+1} - P_i).\end{aligned}$$



Obrázek 18.2: Catmul-Rom spline s vyznačenými tečnami v řídicích bodech

Obrázek 18.3: Vliv parametru a_i na křivku Catmul-Rom.

Implicitní hodnoty parametrů jsou $a = b = c = 0$. Pro tento případ jsou hodnoty vstupního a výstupního vektoru identické a jsou dány pouze polohou řídicích bodů. Hodnota vektoru pro vnitřní body P_i, $i = 1, 2, \ldots, n$ je pak rovna polovině vektoru daného body P_{i+1} a P_{i-1}, jak naznačuje obrázek 18.2. Tento speciální případ se jmenuje křivky

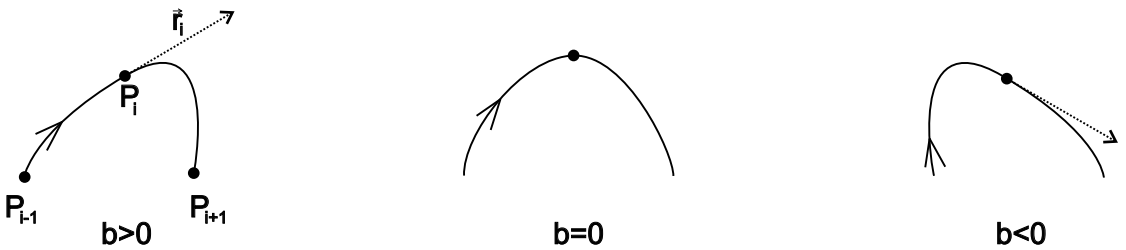
Parametr napětí a_i určuje velikost tečny v $i-$tém bodě. Předpokládejme ostatní parametry v daném bodě rovny nule. Je-li hodnota parametru $-1 < a < 1$, mění se velikost tečny a s ní i tvar křivky tak, jak je naznačeno na obrázku 18.3.

Parametr šikmosti b_i mění směr a délku tečného vektoru (obrázek 18.4). Předpokládejme znovu hodnoty ostatních parametrů rovny nule. Pro $b > 0$ se křivka bude přimykat k vektoru $P_i - P_{i-1}$ a pro $b < 0$ k vektoru $P_{i+1} - P_i$.

Koeficient c_i (*continuity*) řídí spojitost v příslušném bodě. Je-li $c = -1$ je $\vec{l_i} = P_i - P_{i-1}$ a $\vec{r_i} = P_{i+1} - P_i$. Křivka ztratila spojitost prvního řádu a je pouze C^0. Pohybující se objekt ostře a náhle změní svojí dráhu. Pro $c = 1$ je $\vec{l_i} = P_{i+1} - P_i$ a $\vec{r_i} = P_i - P_{i-1}$ a opět dojde k prudké změně směru pohybu.

Jednou z nevýhod TCB křivek je, že předpokládají konstantní interval $t_{i+1} - t_i = 1$ mezi dvěma uzly. Pro jinou změnu parametru $\Delta_i = t_{i+1} - t_i$ se upraví hodnoty vektorů $\vec{l_i}$ a $\vec{r_i}$

$$\vec{l_i'} = \vec{l_i}\frac{2\Delta_i}{\Delta_{i-1} + \Delta_i} \qquad \vec{r_i'} = \vec{r_i}\frac{2\Delta_{i-1}}{\Delta_{i-1} + \Delta_i}.$$



Obrázek 18.4: Vliv parametru b_i na křivku Kochanek–Bartels

18.2 Vysokoúrovňová počítačová animace

Z vysokoúrovňové animace uvedeme stručný úvod do přímé a inverzní kinematiky. Algoritmy, které se používají pro detekci kolizí, jsou uvedeny v části 14.3.

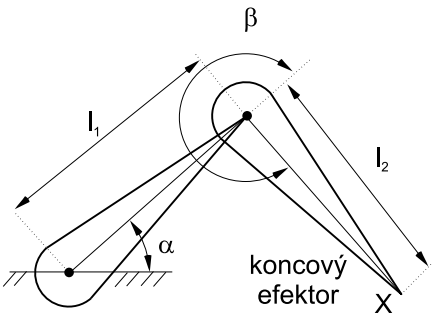
Část počítačové grafiky, o které budeme dále hovořit, patří do oblasti *kinematiky*. Kinematika je část fyziky studující pohyb nezávisle na silách, které ho způsobují. V kinematice se tedy zabýváme pouze polohou, rychlostí a zrychlením. Naproti tomu *dynamika* se zabývá studiem vzájemného působení sil a objektů. V této kapitole budeme hovořit o kinematice, z českých knih se dynamikou zabývá například [Noži91].

18.2.1 Segmentová struktura a stavový prostor

Jak jsme uvedli v úvodu této kapitoly, nejsilnější proud vysokoúrovňové animace směřuje k animaci syntetických herců. Postava člověka je dobrým příkladem toho, čemu budeme

říkat *segmentová struktura* (*articulated structure*). Takovéto objekty se skládají z posloupnosti pevných částí, které jsou mezi sebou spojeny a v každém spojení (*link*) je možno s oběma segmenty otáčet. Posuvné (prismatické) spojení nebývá v počítačové grafice obyčejně uvažováno.

Segmentová struktura bývá na jednom konci pevně zakotvená. Je-li její druhý konec volný, říká se takové struktuře otevřená segmentová struktura a bodu, který je na jejím volném konci, se říká *koncový efektor* (*end effector*).



Obrázek 18.5: Jednoduchá otevřená segmentová struktura se dvěma segmenty

Příkladem segmentové struktury je lidská paže. Na jednom konci je paže pevně svázána s tělem, koncovými efektory jsou prsty. Schematický příklad otevřené segmentové struktury je uveden na obrázku 18.5. Struktura je pevně zakotvena na podložce a skládá se ze dvou částí. Jak v zakotvení, tak ve spojovacím uzlu se může struktura otáčet.

Pokud chceme jednoznačně určit polohu tuhého tělesa v prostoru, potřebujeme k tomu zvolený souřadnicový systém a celkem šest čísel. Tři hodnoty určují souřadnice tělesa v prostoru a tři jeho natočení vzhledem k souřadnicovým osám. Tyto veličiny se jmenují *stupně volnosti* (*DOF – Degree of freedom*) a jednoznačně charakterizují libovolný systém. Počet stupňů volnosti závisí na tom, kolik ze stavových veličin lze při změně polohy tělesa měnit. Přidáme-li do systému nové těleso, zvýší se počet stupňů volnosti. Pokud budeme tělesa svazovat pevnými linkami a budeme vytvářet segmentovou strukturu, bude se naopak počet stupňů volnosti snižovat. Segmentová struktura na obrázku 18.5 se skládá ze dvou prvků. První je pevně zakotven na podložce, ale může se otáčet kolem jedné osy, druhý se může otáčet ve spojovacím kloubu opět pouze kolem jedné osy. Tato struktura má dva stupně volnosti, poloha je plně charakterizována pomocí dvou úhlů α a β.

Všechny možné stavy, ve kterých může zvolená segmentová struktura být, tvoří *stavový prostor* této struktury. Okamžitý stav může být jednoznačně popsán tzv. *stavovým vektorem* (*state vector*). Délka stavového vektoru je stejná jako je počet stupňů volnosti segmentové struktury (taková je i dimenze stavového prostoru). Stavový vektor budeme označovat

$$\mathbf{\Theta} = (\theta_0, \theta_1, \ldots, \theta_n),$$

kde θ_i jsou možné parametry modelu. Struktura na obrázku 18.5 je popsána stavovým vektorem

$$\mathbf{\Theta} = (\alpha, \beta) \ .$$

Tímto stavovým vektorem je poloha koncového efektoru X určena jednoznačně. Připomeňme, že kdyby segmentová struktura nebyla na začátku pevně ukotvena, museli bychom stavový vektor zvětšit o tři souřadnice udávající polohu prvního segmentu v prostoru. Vzhledem k zavedené notaci stavového prostoru můžeme říci, že animace pohybu segmentové struktury odpovídá hledání cesty (postupnému procházení jednotlivých stavů) v jejím stavovém prostoru.

18.2.2 Reprezentace animovaného objektu

Zatímco při modelování například pomocí ploch, objemů, či v procedurálním modelování, se zabýváme tím, jaký má těleso tvar, v této části knihy nás bude zajímat, jak je těleso poskládáno z pevných částí, které se mohou v kloubech otáčet. Snahou je používat takovou reprezentaci, která uchovává co možná nejméně parametrů a která umožňuje snadnou manipulaci s nimi. K tomu se v počítačové animaci používá tzv. *DH-notace*, která je pojmenovaná podle Denavita a Hartenberga [Dena55], kteří ji pro používání v robotice popsali v roce 1955.

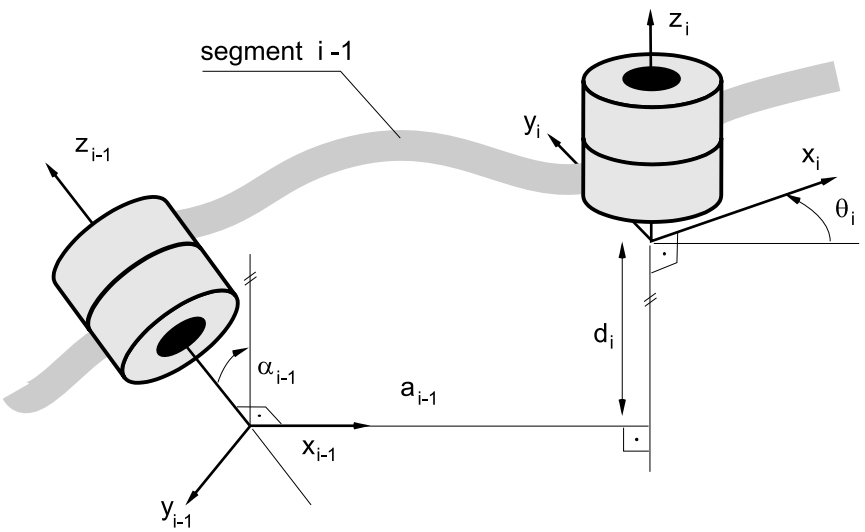
V DH-notaci je každý kloub reprezentován svým souřadnicovým systémem a čtyřmi parametry, které určují, jakou transformací přejdeme k následujícímu segmentu. Souřadnicový systém prvního segmentu je vyjádřen ve světových souřadnicích a souřadnicové systémy následujících segmentů jsou vyjádřeny v souřadnicových systémech segmentů předchozích. K určení souřadnic koncového efektoru musíme tedy postupně projít všechny segmenty.

Význam jednotlivých parametrů, které se v DH-notaci používají, je následující (viz obr. 18.6). Segmenty jsou číslovány v přirozeném pořadí, index $i-1$ označuje předchůdce, index i následníka.

- a_{i-1} označuje vzdálenost mezi osami z_{i-1} a z_i sousedních segmentů měřenou v ose x_{i-1},
- d_i označuje vzdálenost mezi osami x_{i-1} a x_i měřenou v ose z_i,
- α_{i-1} je úhel určující natočení souřadnicového systému následníka okolo osy x_{i-1}, tj. odchylku mezi mezi osami z_{i-1} a z_i,
- θ_i je úhel určující odchylku mezi osami x_{i-1} a x_i měřený vzhledem k ose z_i.

Z těchto čísel můžeme sestavit transformační matici, která určuje, jak se přejde od souřadnicového systému jednoho segmentu k souřadnicovému systému segmentu následujícího. Tyto matice bývá zvykem označovat ${}^{(i-1)}\mathbf{T}_i$. Indexy označují, od kterého segmentu přecházíme ke kterému.

Mezičlánková transformace je dána složením čtyř transformačních matic vyjadřujících postupně rotaci $\mathbf{R}_{z\theta}$ o úhel θ_i vzhledem k ose z_i, posun $\mathbf{T}_{zd}$ o vzdálenost d_i podél osy z_i, posun $\mathbf{T}_{xa}$ o vzdálenost a_{i-1} podél osy x_{i-1}, a konečně rotaci $\mathbf{R}_{x\alpha}$ o úhel α_{i-1} podle osy x_{i-1}.



Obrázek 18.6: Vztah mezi souřadnicovými systémy dvou sousedních segmentů (podle [Watt:92])

Výsledná matice ${}^{(i-1)}\mathbf{T}_i$ má pak tvar

$$ {}^{(i-1)}\mathbf{T}_i = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & a_{i-1} \\ \sin(\theta_i)\cos(\alpha_{i-1}) & \cos(\theta_i)\cos(\alpha_{i-1}) & -\sin(\alpha_{i-1}) & -d_i\sin(\alpha_{i-1}) \\ \sin(\theta_i)\sin(\alpha_{i-1}) & \cos(\theta_i)\sin(\alpha_{i-1}) & \cos(\alpha_{i-1}) & d_i\cos(\alpha_{i-1}) \\ 0 & 0 & 0 & 1 \end{bmatrix}. \qquad (18.1) $$

Transformace ${}^{0}\mathbf{T}_n$ přechodu od světového souřadnicového systému k lokálnímu souřadnicovému systému segmentu koncového efektoru X se získá vynásobením matic pro dílčí transformace mezi sousedními segmenty:

$$ {}^{0}\mathbf{T}_n = {}^{(n-1)}\mathbf{T}_n \,.\, {}^{(n-2)}\mathbf{T}_{n-1} \ldots {}^{(0)}\mathbf{T}_1 \qquad (18.2) $$

18.2.3 Přímá a inverzní kinematika

Naším cílem je nějakým způsobem určit polohu koncového efektoru a hodnoty stavového vektoru segmentové struktury, tj. úhly natočení všech segmentů, případně polohu zakotvení prvního segmentu. K tomu se v počítačové animaci obyčejně používají dvě metody, *přímá kinematika* a dnes častěji *inverzní kinematika*.

Přímá kinematika

Výpočet polohy koncového efektoru pomocí přímé kinematiky spočívá v postupném určení jednotlivých stavů stavového vektoru pro všechny segmenty struktury. Představme si virtuální figurku, která má za úkol položit skleničku na stůl. Nejprve stanovíme úhel natočení v rameni, poté v lokti a nakonec v zápěstí. Výsledkem tohoto postupu bude poloha skleničky (koncového efektoru) na stole. Pokud bude figurka působit nepřirozeně a budeme muset například změnit úhel natočení ramene, změníme tím i tvar celé ruky a musíme postupovat znovu od posledního změněného místa. To je největší nevýhodou přímé kinematiky. Formálně můžeme slovně vyjádřený postup napsat jako

$$X = f(\mathbf{\Theta}). \tag{18.3}$$

Poloha koncového efektoru X je určena pomocí transformace f, která je sestavena na základě nové hodnoty stavového vektoru $\mathbf{\Theta}$, v němž byly promítnuty všechny změny v jednotlivých stupních volnosti animované struktury. Do jednotlivých matic (vztahy 18.1 a 18.2) jsou dosazeny konkrétní hodnoty parametrů DH-notace a určena výsledná poloha efektoru ve světových souřadnicích.

Algoritmy využívající přímé kinematiky jsou jednodušší z hlediska implementace. Využívání tohoto způsobu řízení modelu je však neintuitivní a pro animátora je zadávání jednotlivých parametrů pro každý segment často zdlouhavé. Z těchto důvodů je přímá kinematika používána spíše v případech, kdy je animace řízena nějakým vnějším popisem (viz část 18.4.3). Pro interaktivní vytváření pohybu je výhodnější tzv. inverzní kinematika.

Inverzní kinematika

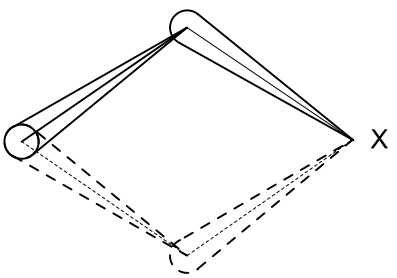
Při výpočtu parametrů jednotlivých spojů pomocí inverzní kinematiky jde o opačný postup nežli v předcházejícím případě. Cílem je nalézt stavový vektor $\mathbf{\Theta}$ na základě informace o poloze koncového efektoru P. Často uváděným příkladem takového postupu je stanovení úhlů natočení kloubů při jízdě na kole. Koncový efektor (poloha chodidla) stejně jako „zakotvení“ jezdce na sedačce jsou známy a algoritmus inverzní kinematiky určí stavový vektor, tj. úhly ohnutí kloubů. Formálně můžeme tento krok zapsat jako

$$\mathbf{\Theta} = f^{-1}(X). \tag{18.4}$$

Poloha koncového efektoru X je známa, pomocí inverzní funkce f^{-1} určíme stavový vektor $\mathbf{\Theta}$. Inverzní kinematice se také říká *cílem řízený pohyb* (*goal directed motion*).

Při podrobnějším rozboru řešení inverzní kinematiky zjistíme, že může nastat několik nepříjemných stavů. Inverzní funkce f^{-1} nemusí pro nějaké polohy koncového efektoru X vůbec existovat. Můžeme například požadovat natažení struktury tak, že náš požadavek překročí

celkovou délku segmentů. Jiný případ je uveden na obrázku 18.7. Zde je poloha koncového efektoru zvolena tak, že existují dvě možná řešení pro danou segmentovou strukturu, existují tedy dva stavy ve stavovém prostoru, které této polohy dosáhnou. Takovéto případy musí samozřejmě algoritmus umět včas odhalit a systém, který inverzní kinematiku používá, by na ně měl přiměřeně reagovat.



Obrázek 18.7: Nejednoznačné řešení pro danou strukturu

Jedním z řešení problému nejednoznačnosti je omezení stavového prostoru. Snížení počtu přípustných řešení můžeme docílit tak, že do systému zavedeme určitá omezení (*constraints*). Příkladem takové podmínky je zákaz otáčení kloubu za určitou mez. Tímto způsobem však nevyloučíme všechny nejednoznačnosti řešení.

Největším problémem inverzní kinematiky ale je, že funkce f je nelineární a analytický výpočet její inverze je pro jen trochu složitější hierarchie prakticky nemožný. Všeobecně používaným řešením je proto linearizace problému, kdy provádíme řešení pouze v okolí aktuální pozice. K tomu potřebujeme použít *jakobián*, který je vícerozměrným rozšířením diference jedné proměnné. Mějme dánu funkci:

$$X = f(\mathbf{\Theta})$$

kde X má dimenzi m (v prostoru je obvykle $m = 3$, může se však lišit podle počtu stupňů volnosti koncového efektoru) a Θ má dimenzi n. Velikost n závisí na počtu volitelných parametrů a u jednoduché kloubové soustavy může být např. $n = 6.k$, kde k je počet kloubů. Jakobián je matice o rozměrech $m \times n$, která mapuje změny $\mathbf{\Delta\Theta}$ na změny ΔX.

$$\Delta X = \mathbf{J}(\mathbf{\Theta})\mathbf{\Delta\Theta} \tag{18.5}$$

Přitom platí:

$$\mathbf{J}_{m\times n} = \begin{bmatrix} \frac{\partial f_1}{\partial \theta_1} & \frac{\partial f_1}{\partial \theta_2} & \cdots & \frac{\partial f_1}{\partial \theta_n} \\ \vdots & \ddots & \cdots & \vdots \\ \frac{\partial f_m}{\partial \theta_1} & \cdots & \cdots & \frac{\partial f_m}{\partial \theta_n} \end{bmatrix}.$$

Zde je důležité připomenout, že funkce f, a tedy i jakobián $\mathbf{J}$ závisí na konkrétním stavu hierarchie těles. To je ve vzorci zdůrazněno zápisem $\mathbf{J}(\mathbf{\Theta})$ místo pouhého $\mathbf{J}$. Pokud zderivujeme rovnici 18.5 změnou času dt, dostaneme

$$\dot{X} = \mathbf{J}(\mathbf{\Theta})\dot{\mathbf{\Theta}}$$

kde $\dot{X}$ je rychlost koncového efektoru, v nejobecnějším případě dimenze 6, obsahující jak lineární, tak úhlovou rychlost soustavy objektů připojených v tomto bodě. $\dot{\mathbf{\Theta}}$ je derivace stavového vektoru podle času. Jakobián mapuje rychlosti ve stavovém prostoru na rychlosti a natočení těles připojených ke koncovému kloubu.

18.2.4 Inverze jakobiánu

Inverze jakobiánu je pravděpodobně nejčastěji zmiňovaná metoda pro řešení inverzní kinematiky. Základní myšlenka je poměrně jednoduchá. Pokud předpokládáme, že dokážeme vypočítat inverzi jakobiánu, dostaneme po úpravě 18.5 vztah:

$$\mathbf{\Delta\Theta} = \mathbf{J}^{-1}(\mathbf{\Theta})\Delta X \tag{18.6}$$

Teď už můžeme dosadit za ΔX změnu souřadnic koncového bodu a s využitím vztahu 18.6 dostaneme požadovanou změnu stavu hierarchie $\mathbf{\Delta\Theta}$. Výpočet charakterizuje algoritmus 18.1 a postup je naznačen na obrázku 18.2.4.

Vstup: stavový vektor hierarchie $\mathbf{\Theta}$
poloha koncového efektoru v daném stavu, tj. $X = f(\mathbf{\Theta})$
Opakuj

$\mathbf{J}(\mathbf{\Theta})$	*vypočítej jakobián* $f(\mathbf{\Theta})$
$\mathbf{J}^{-1}(\mathbf{\Theta})$	*invertuj jakobián*
$\Delta X = k.(X_{cil} - X),\ 0 < k < 1$	*zvol malý pohyb směrem k cíli*
$\mathbf{\Theta} = \mathbf{\Theta} + \mathbf{J}^{-1}(\mathbf{\Theta})\Delta X$	*odhadni změnu stavového vektoru*
$X = f(\mathbf{\Theta})$	*efektor do polohy odpovídající změně stavu*

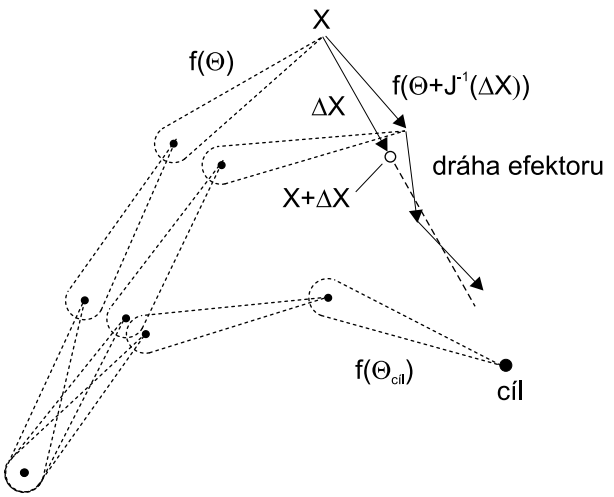
dokud není cíl dosažen.

Algoritmus 18.1: Výpočet pohybu pomocí inverzní kinematiky

Samozřejmě zde narazíme na několik problémů. V typickém případě bude matice $\mathbf{J}$ obdélníková a její inverze tedy neexistuje. Za této situace se používají metody pro výpočet tzv. pseudoinverze obdélníkové matice.

Dalším problémem je to, že jakobián $\mathbf{J}(\mathbf{\Theta})$ závisí na aktuálním stavu hierarchie – to znamená, že výpočet má smysl jen pro malé $\mathbf{\Delta\Theta}$ a tudíž i malé ΔX. Je tedy potřeba vhodným způsobem omezit maximální velikost $\mathbf{\Delta\Theta}$. To lze udělat například tak, že pokud dostaneme příliš velké $\mathbf{\Delta\Theta}$, zmenšíme ΔX na polovinu a výpočet opakujeme tak dlouho, dokud nedostaneme přijatelné hodnoty. Samozřejmě, zmenšením ΔX posuneme cíl, do kterého se má koncový bod

hierarchie dostat. Je tedy třeba výpočet opakovat – takovýto výpočet pak nenajde požadovanou konfiguraci hierarchie v jednom kroku, ale v několika, jedná se o iterační výpočet. Výhodou tohoto postupu je, že se dá jednoduše rozšířit o splnění dodatečných kritérií. Podrobný popis algoritmu inverzní kinematiky, stejně jako další detaily a rozšíření této metody jsou uvedeny v literatuře [Watt92].



Obrázek 18.8: Postupná iterace při použití pseudoinverze

18.3 Skeletální animace

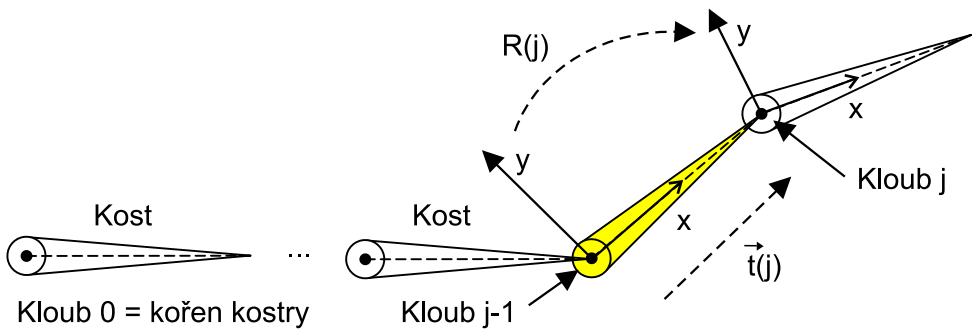
Skeletální animace slouží pro deformaci složitých objektů, které by bylo obtížné simulovat přesně. Typickou aplikací je vizualizace pohybujícího se lidského těla, odkud také vychází používaná terminologie.

Model je zadán svým tvarem vzhledem ke zvolené základní (referenční) poloze. Tvar lze reprezentovat sítí trojúhelníků, která se obvykle doplňuje o texturu a kvůli stínování i o normály vrcholů. Z hlediska skeletální animace přitom není podstatné, zda použijeme jedinou trojúhelníkovou síť pro celý model, tzv. *deformovatelnou pokožku* (*skinned body geometry*), nebo pro každou část těla samostatnou síť, zvanou *pevný segment* (*skeletal body geometry*).

Nejjednodušší animační technika, známá jako *animace vrcholů*, pracuje přímo s vrcholy trojúhelníků a interpoluje jejich polohu. Tento přístup je náročný na paměť, protože každý

klíčový snímek obsahuje údaje o poloze každého vrcholu. Navíc je obtížné tato pohybová data získat a upravovat.

Výhodnější je použití techniky skeletální animace, která využívá pomocnou strukturu zvanou *kostra*. Kostru lze definovat jako strom. Jeho uzly odpovídají kloubům a hrany kostem. V každém uzlu je uložena homogenní matice, která popisuje afinní transformaci od jednoho kloubu k druhému. Pro popis většiny deformovatelných objektů přitom vystačíme s rotací a posunutím. V dalším textu vysvětlíme, jak souřadnice vrcholů, zadané v souřadnicové soustavě kořene kostry, dočasně převést do lokálních souřadnicových soustav uzlů, v nich provést potřebné animační transformace a výslednou polohu vrcholů opět vyjádřit v souřadnicové soustavě kostry. Pojem *kartézská souřadnicová soustava* (někdy též označovaný jako *repér*) budeme zkráceně nazývat *soustava*.



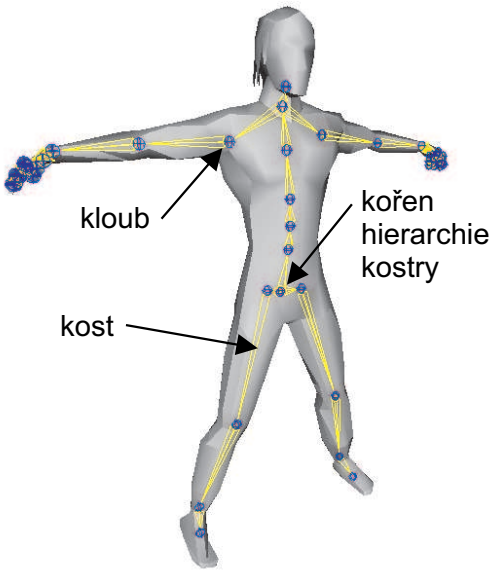
Obrázek 18.9: Změna souřadnicové soustavy mezi klouby $j-1$ a j

Postupy, používané při skeletální animaci, vysvětlíme bez újmy na obecnosti na zjednodušeném podstromu, který neobsahuje větvení. Je tedy tvořen řadou uzlů (kloubů) mezi kořenem a listem stromu, jak ukazuje obrázek 18.9. Klouby označíme nezápornými celými čísly podle vzdálenosti od kořene. Kořenu stromu přiřadíme nulu, index předchůdce kloubu j bude $j-1$. Homogenní matice ${}^{(j-1)}\mathbf{M}_j$ příslušná kloubu j sestává z matice rotace $\mathbf{R}(j)$ a vektoru posunutí $\vec{t}(j)$:

$$ {}^{(j-1)}\mathbf{M}_j = \begin{bmatrix} \mathbf{R}(j) & \vec{t}(j) \\ 0 & 1 \end{bmatrix}. $$

Připomeňme, že stejně jako v části 18.2 tato matice určuje přechod od soustavy kloubu $j-1$ k soustavě kloubu j. Pokud souřadnice vrcholu vyjádřené v soustavě kloubu $j-1$ vynásobíme maticí ${}^{(j-1)}\mathbf{M}_j$, získáme souřadnice téhož vrcholu vyjádřené v soustavě j. Celkovou transformaci světových souřadnic do souřadnic příslušných j-tému kloubu lze psát jako

$$ {}^{0}\mathbf{M}_j = {}^{(j-1)}\mathbf{M}_j \,.\, {}^{(j-2)}\mathbf{M}_{j-1} \ldots {}^{0}\mathbf{M}_1 $$



Obrázek 18.10: Pokožka a kostra virtuálního humanoida

Množina všech transformací ${}^{(j-1)}\mathbf{M}_j$ určuje základní (referenční) polohu kostry. Příklad kostry v její referenční poloze je na obrázku 18.10. Každý kloub j je umístěn do polohy dané posuvnou částí homogenní matice ${}^{0}\mathbf{M}_j^{-1}$. Při popisu referenční polohy kostry by bylo samozřejmě možné vystačit pouze s posunem, tj. předpokládat všechny matice rotace $\mathbf{R}(j)$ rovné identitě. Nicméně obecná rotace $\mathbf{R}(j)$ umožňuje pracovat se souřadnicovými systémy jednotlivých kloubů a nastavovat tak referenční polohu kostry podle požadavků různých aplikací (např. stoj, leh, sed).

Při animaci kostry aplikujeme na každý kloub individuální transformační matici $\mathbf{A}_j$. Pro všechny klouby s výjimkou kořene se tyto animační matice $\mathbf{A}_j$ pokládají za rotace, protože klouby nebývají posuvné. Pouze posun v kořeni má dobrý smysl – jedná se o transformaci celého modelu, která může zahrnovat rotaci i posun.

Předpokládejme, že máme vrchol v_j, který je zadán v soustavě (koncového) kloubu j. Jeho novou, animovanou polohu v'_j vypočítáme jako

$$v'_j = \mathbf{A}_j \; v_j.$$

Souřadnice v'_j vyjádříme v soustavě kloubu $j-1$ pomocí složené matice ${}^{j}\mathbf{F}_{j-1}$. Vzhledem k tomu, že postupujeme směrem ke kořeni, získáme potřebnou matici pro transformaci souřadnic inverzí matice ${}^{(j-1)}\mathbf{M}_j$

$${}^{j}\mathbf{F}_{j-1} \;=\; {}^{j}M_{(j-1)} \cdot \mathbf{A}_j \;=\; {}^{(j-1)}\mathbf{M}_j^{-1} \cdot \mathbf{A}_j.$$

V soustavě kloubu $j-1$ aplikujeme animační transformaci $\mathbf{A}_{j-1}$ a takto postupujeme až ke kořeni kostry. Jednotlivé matice $\mathbf{F}$ mají tvar

$$\begin{aligned}
{}^{j}\mathbf{F}_{j-1} &= {}^{(j-1)}\mathbf{M}_j^{-1} \cdot \mathbf{A}_j \\
{}^{j}\mathbf{F}_{j-2} &= {}^{(j-2)}\mathbf{M}_{j-1}^{-1} \cdot \mathbf{A}_{j-1} \cdot {}^{(j-1)}\mathbf{M}_j^{-1} \cdot \mathbf{A}_j \\
&\dots \\
{}^{j}\mathbf{F}_{0} &= {}^{0}\mathbf{M}_1^{-1} \cdot \mathbf{A}_1 \dots {}^{(j-1)}\mathbf{M}_j^{-1} \cdot \mathbf{A}_j.
\end{aligned}$$

Matice ${}^{j}\mathbf{F}_0$ vyjadřuje transformaci souřadnic animovaného vrcholu ze soustavy j do soustavy kořene kostry. Všimněme si, že rotace kloubů se skládají přirozeným způsobem, např. rotace prstu je následována rotací v zápěstí, lokti a rameni. Protože je kostra stromem, můžeme všechny matice ${}^{j}\mathbf{F}_0$, resp. ${}^{0}\mathbf{M}_j$ spočítat jedním systematickým průchodem stromu.

Kostra významně zjednodušuje úlohu animace deformovatelného objektu. Místo manipulace s potenciálně velkou skupinou vrcholů vystačíme s manipulací kostry. Zbývá vyjasnit, jak se změní povrch modelu při změně postavení kostry. Toto je ve skutečnosti klíčový problém skeletální animace, protože se snažíme dosáhnout přirozené, hladké deformace pokožky pouze pomocí informace o natočení kostry. Přesné řešení by v případě lidské postavy vyžadovalo reprezentaci vnitřních orgánů, svalů a tukových vrstev. To se v některých profesionálních systémech skutečně používá, avšak pro aplikace pracující v reálném čase postačí jednodušší modely.

Abychom zařídili automatickou deformaci pokožky pomocí kostry, musíme nějakým způsobem definovat vazbu pokožky na kostru. Zde se nabízí řada možností. Nejjednodušší z nich, známá jako *základní skeletální animace*, přiřazuje každý vrchol sítě reprezentující pokožku právě jednomu kloubu. Vrcholy se v této metodě transformují tak, jako kdyby byly pevně spojeny s kloubem. Formálně řečeno, je-li vrchol o referenční poloze v_0 zadaný v souřadné soustavě kořene kostry přiřazen kloubu j, polohu v_0' po transformaci spočítáme jako

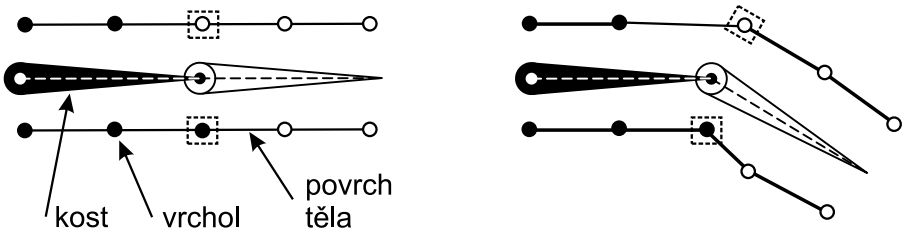
$$v_0' = \mathbf{A}_0 \cdot {}^{j}\mathbf{F}_0 \cdot {}^{0}\mathbf{M}_j \; v_0.$$

Interpretace tohoto vzorce je přímočará: matice ${}^{0}\mathbf{M}_j$ převede souřadnice vrcholu do souřadnicové soustavy kloubu j. Matice ${}^{j}\mathbf{F}_0$ provede všechny dílčí transformace v kloubech a současně souřadnice vrcholu vrátí zpět do soustavy kořene kostry. Animační transformace $\mathbf{A}_0$ pak mění celkovou pozici kostry. Všimněme si, že pokud jsou všechny $\mathbf{A}_j$ identity, tj. okamžitá poloha animované kostry je shodná s referenční polohou, vychází přirozeně $v_0' = v_0$.

Základní skeletální animace je vhodná především pro model tvořený pevnými segmenty. Takový model vypadá přirozeně např. pro robota, ale při pokusu o modelování lidské postavy se jeví, jako kdyby to byla dětská panenka tvořená z jednotlivých pevných částí. Je samozřejmě možné použít základní skeletální animaci i na deformovatelnou pokožku, ovšem s nepříliš uspokojivým výsledkem, viz obrázek 18.11 vpravo. Tvar deformované pokožky není hladký, deformace se projeví pouze na těch ploškách sítě, jejichž vrcholy jsou přiřazené k různým kloubům.

18.3.1 Míchání vrcholů

Hladkého tvaru deformované pokožky lze dosáhnout technikou známou jako míchání vrcholů (*vertex blending*). Jedná se vlastně pouze o jinou, obecnější vazbu pokožky na kostru. Vrchol



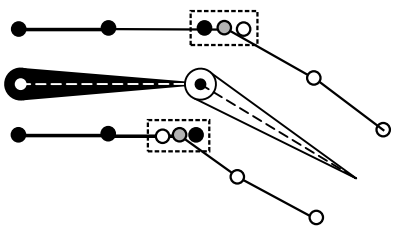
Obrázek 18.11: Základní skeletální animace aplikovaná na deformovatelnou pokožku. Vlevo dvojice kostí v referenční poloze, vpravo po rotaci kloubu. Vrcholy pokožky přiřazené první kosti jsou označeny černě. Na vrcholech zvýrazněných rámečky je vidět nepřirozená deformace pokožky způsobená pevným přiřazením k jednotlivým kostem.

pokožky je tentokrát přiřazen několika kloubům, nikoliv pouze jednomu jako v základní skeletální animaci. Výsledná poloha vrcholu se spočítá jako konvexní kombinace transformací tohoto vrcholu příslušnými klouby. Přesněji řečeno, předpokládejme, že vrchol o referenční poloze v je přiřazen kloubům $j_0, \ldots, j_n$. Potom jeho výsledná poloha v' se spočítá jako

$$v' = \mathbf{A}_0 \sum_{i=0}^{n} w_i \, {}^{j_i}\mathbf{F}_0 \cdot {}^{0}\mathbf{M}_{j_i} \, v,$$

kde w_i jsou váhy splňující podmínku

$$\sum_{i=0}^{n} w_i = 1, \; w_i \geq 0 \; \forall i \; \in \{0, \ldots, n\}.$$



Obrázek 18.12: Pokožka deformovaná mícháním vrcholů

Váhy w_i lze interpretovat jako míru vlivu kloubu j_i na transformaci vrcholu v. Proces míchání vrcholů je znázorněn na obrázku 18.12, kde jsou v označené oblasti nakresleny černě a bíle polohy vrcholů po transformaci způsobené první a druhou kostí. Šedé kroužky představují výsledné vrcholy vzniklé mícháním.

Váhy lze buď ručně nastavit v animačním programu, nebo spočítat podle nějakého pravidla (typicky se uvažuje vzdálenost vrcholu od kosti). Množina kloubů, ke kterým je vrchol přiřazen, bývá zpravidla malá – dva až čtyři klouby. Mnoho vrcholů dokonce vystačí s připojením pouze k jedné kosti, protože míchání vrcholů má význam jen v okolí kloubů.

Ačkoliv je míchání vrcholů velmi oblíbené pro svoji jednoduchost a rychlost, vykazuje při některých polohách kostry značné nedostatky. Jev, ke kterému dochází při velkých rotacích, bývá označován jako *problém papíru od bonbonů* (*candy-wrapper artifact*), protože míchání vrcholů pro úhel rotace blízký 180° pokožku překroutí, jako kdyby byla z papíru. O možnostech odstranění této vady pojednává řada článků [Bloo02, Kava03, Wang02].

18.4 Virtuální humanoid

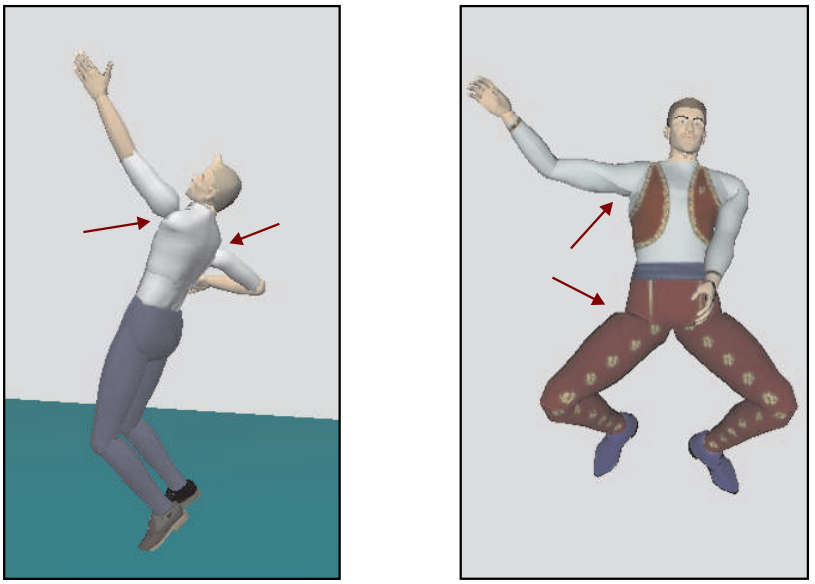
S rozvojem virtuálních prostředí se zcela přirozeně rozvíjí i snaha modelovat a zobrazovat v tomto prostředí člověka, resp. lidskou postavu. Počítačové modely člověka se liší v mnoha aspektech podle zamýšleného použití. Na jednom konci spektra stojí prosté projekce uživatele do víceuživatelského virtuální prostředí, jejichž jediným účelem je dát uživateli pozorovatelnou podobu a jejich úroveň realismu vizuálního i funkčního je tak často velmi nízká. Opačný konec tvoří modely funkčně i vizuálně velmi pokročilé, sloužící k nejrůznějším simulacím.

Model (jakkoliv realistický) sloužící ke zviditelnění návštěvníka virtuálního prostředí se nazývá *avatar*. Tento pojem pochází ze sanskrtu, kde původně označoval hmotné vtělení boha. Avatar ve virtuální realitě je v podstatě komunikačním nástrojem a pro svou funkci vlastně nemusí mít ani lidskou podobu. Naproti tomu pojmem *virtuální humanoid* označujeme modely, které mají skutečně lidské tvary. Využití virtuálních humanoidů přesahuje oblast víceuživatelské virtuální reality, mnohem větší význam mají ve filmovém a herním průmyslu, v různých simulacích fyzických možností i chování, při ověřování průmyslových návrhů a pracovních postupů a v dalších oblastech.

18.4.1 Struktura humanoida

Virtuální humanoid je obecně hierarchická artikulovaná struktura, kterou představuje vrstva kostry – acyklický graf, jehož uzly (klouby) nesou informace o transformacích, jak bylo uvedeno v předchozí části 18.3. Vrstva kostry tedy tvoří funkční, resp. řídicí část modelu. Počet kloubů, které tato vrstva obsahuje, určuje počet stupňů volnosti celé struktury a tak i míru její pohyblivosti, kterou nazýváme *úroveň artikulace*. Úroveň artikulace se pohybuje od několika kloubů pro jednoduché modely až po desítky kloubů u propracovaných humanoidů.

Další vrstvou virtuálního humanoida je vrstva zodpovědná za vzhled modelu – pokožka. Ta může být implementována jak deformovatelnou sítí plošek (mřížek) či množinou pevných segmentů. *Pevné segmenty* platí za svou jednoduchost nižší vizuální věrohodností, protože při animaci dochází v okolí kloubů k tvorbě nežádoucích artefaktů a „vykloubení“, jak znázorňuje obrázek 18.13 vlevo. Výpočetně náročnější, avšak vizuálně kvalitnější modely používají *deformovatelnou pokožku*, viz obrázek 18.13 vpravo.



Obrázek 18.13: Příklad geometrie pevných segmentů (vlevo) a geometrie deformovatelné pokožky (vpravo). Šipky ukazují na artefakty „vykloubení“ u pevných segmentů, resp. překroucení u deformované pokožky (obrázek poskytl V. Štěpán z ČVUT v Praze, humanoid a animační data pocházejí z laboratoře VRLab v Lausanne, Švýcarsko).

Zajímavý přístup kombinující oba předchozí byl použit laboratoří VRLab ve Švýcarsku [Fua00]. Mezi kostru a pokožku byla umístěna vrstva aproximující svalstvo pomocí elipsoidů o neměnné velikosti. Tato vrstva je pak v každém snímku vzorkována, čímž jsou získány řídicí body pro spline plochu, která se zobrazuje jako pokožka. Tento třívrstvý model poskytuje vizuálně velmi dobré výsledky za cenu vyšších výpočetních nároků.

Virtuální humanoidi se mohou lišit v úrovni artikulace i ve vizuální realističnosti. Modely s vysokou úrovní realismu se používají ve filmech a počítačových hrách. Pokud však aplikace nevyžadují realistický vzhled, můžeme namísto pokožky použít jednoduché geometrické objekty. Někdy dokonce postačuje i prostý drátový model. Zejména při zpracování dat získaných snímáním pohybů skutečného člověka (viz 18.4.3) je základním modelem jen hierarchie transformací. Často se realisticky zobrazuje pouze kostra (kostlivec), na níž není třeba implementovat deformace při pohybu.

18.4.2 Norma H-Anim

Rozvoj programů pro vytváření a animování virtuální lidské postavy, spolu s vývojem systémů pro sledování a záznam pohybů skutečných lidí v reálném světě, přinesl potřebu standardizovaného modelu humanoida. Takový model byl navržen koncem devadesátých let a později byl přijat jako mezinárodní norma s názvem *H-Anim* (*Humanoid Animation*) [ISO03a]. Norma definuje hierarchickou strukturu virtuálního humanoida založenou na studiu lidské anatomie; systém názvosloví a datové struktury pro konstrukci modelu. Pamatuje též na to, že různé aplikace mají různé nároky na realismus, a hierarchická struktura je popsána v několika úrovních artikulace, které se liší počtem zastoupených kloubů. Dalším přínosem normy je souhrn rozměrů jednotlivých segmentů (částí těla) průměrného jedince.

Postavy H-Anim jsou popsány pomocí pěti typů objektů, z nichž první tři se objevují ve většině modelů H-Anim:

- *Humanoid*. Uchovává globální údaje o postavě. Obsahuje autorské údaje o modelu, vytváří rozhraní pro přístup k vlastnímu modelu a umísťuje model v globálním souřadnicovém systému. Objekt Humanoid je kořenem stromu, který obsahuje ostatní části modelu. Umožňuje použít oba výše uvedené způsoby reprezentace geometrie modelu.
- *Joint*. Kloub je základní stavební jednotkou hierarchie humanoida. Každý kloub rotačně transformuje podřízenou část hierarchie, nese tedy data popisující tuto transformaci. Uchovává odkazy na další, podřízené klouby. Pro použití v animačních aplikacích má kloub speciální parametry určující možnosti jeho pohyblivosti (mezní polohy, tuhost). Středy otáčení jednotlivých kloubů jsou definovány v jednotné souřadnicové soustavě. Délku kosti (segmentu) tak lze určit jako vzdálenost středů kloubů, které s kostí incidují.
- *Segment*. Je to objekt popisující fyzickou část lidského těla. Obsahuje převážně informace o (zobrazované) geometrii. Z hlediska animace je segment na rozdíl od kloubu hmotný, proto obsahuje informace o hmotnosti, těžišti a setrvačných momentech příslušné tělesné části. V hierarchii modelu jsou objekty typu Segment ukládány na pozici potomka objektu Joint.
- *Site*. Určuje umístění významných bodů v modelu, resp. na povrchu zobrazované části humanoida. Tyto body mohou být koncovými efektory pro potřeby inverzní kinematiky, mohou to být body pro připojení oblečení, šperků, nesených předmětů, či jiných objektů nějak podřízených hierarchii modelu. Objekt tohoto typu je potomkem objektu typu Segment.
- *Displacer*. Smyslem tohoto objektu je definovat podmnožinu bodů v geometrii Segmentu, které mohou být podrobeny samostatné transformaci. Typické použití objektu Displacer

je při animaci pohyblivých částí těla, které se nepopisují pouze základní hierarchií transformací (pohyby v obličeji, např. zdvižení obočí).

Norma H-Anim také předepisuje nulovou polohu modelovaného humanoida, tj. polohu, při níž mají všechny klouby nulovou rotaci. Tato poloha je vzpřímená, tváří ve směru kladné poloosy $z+$, směr $y+$ je nahoru a směr $x+$ po levici humanoida. Počátek lokální soustavy souřadnic humanoida je v úrovni země mezi chodidly, která jsou položena na rovině $y = 0$. Paže spočívají rovně dolů podél boků s dlaněmi obrácenými ke stehnům. Ve vojenské terminologii bychom řekli, že humanoid stojí v základním postoji – v pozoru.

18.4.3 Data pro animaci virtuálních humanoidů

V kapitole 18.1.1 o nízkoúrovňové počítačové animaci jsou popsány animační křivky. Vzorkujeme-li tyto křivky s krokem, který odpovídá určitému kroku Δt, získáme posloupnost souřadnic bodů ležících na křivce. Obdobně data popisující animaci humanoidů (artikulovaných struktur obecně) bývají seřazena do posloupnosti, vyjadřující polohu humanoida v určitém čase. Informaci o změnách rychlosti představuje vektor klíčových časových okamžiků. Posloupnost obsahuje buď prostorové souřadnice vybraných bodů (kloubů) nebo hodnoty rotací v těchto kloubech. Rotace se vyjadřují různými způsoby: od transformační matice přes trojici Eulerových úhlů (viz dále) až po kvaterniony (viz část 21.4).

Animační data mohou být získávána různými způsoby. Přímá a inverzní kinematika patří mezi nejběžnější techniky (viz část 18.2). Je ovšem možno také nahrávat pohyby skutečného člověka. Tím se zabývá skupina technik souhrnně nazývaných *snímání pohybu* (*motion capture*).

Snímání pohybu

Technologie zvaná *snímání pohybu* (*motion capture*) umožňuje sledovat v čase pozice (a/nebo orientace) senzorů strategicky rozmístěných na objektech ve skutečném světě. Tyto objekty mohou být jakéhokoliv typu, nejčastěji je však tato technologie používána pro snímání pohybů člověka. Záznam pohybu skutečného člověka je v některých případech nejsnazší a nejrychlejší cestou, jak získat věrohodnou animaci. Další aplikací je přímý přenos – rozhýbání virtuální postavy v reálném čase. Nejčastější použití je v různých zábavných televizních pořadech, ale často jsou tyto efekty využívány i umělci v nejrůznějších představeních.

Potřeba realistických animací vychází nejčastěji z oblasti zábavního průmyslu, velká část vývoje probíhá v tomto silně konkurenčním prostředí. Výsledkem je značná variabilita v přístupech k technologii snímání pohybu. Existuje množství výstupních datových formátů a také množství různých implementací snímacích systémů na různých fyzikálních principech.

Zařízení pro snímání pohybu můžeme dělit podle určení na zařízení pracující v reálném čase (*on-line devices*) a zařízení, která ukládají data pro pozdější zpracování (*off-line devices*). Jinou

možností je rozdělení podle použitého fyzikálního principu. Každý z nejčastěji používaných principů má přitom své výhody a nevýhody:

- *Mechanická (elektromechanická) zařízení*
  Tyto snímače získávají data rychle, ale tuto výhodu značně zastiňují výrazným omezením herce (osoby, jejíž pohyby jsou snímány) nesenými senzory. Tyto senzory jsou tvořeny jakýmsi pohyblivým exoskeletem. Důvod, proč se mechanická zařízení používají, tkví v jejich další výhodě – jsou použitelná i ve venkovním prostředí, kde další druhy systémů selhávají. Také netrpí problémy s vnějším rušením a zákryty senzorů.
- *Magnetická zařízení*
  Využívají senzorů umístěných na těle herce, které měří nízkofrekvenční magnetické pole generované vnějším zdrojem. Senzory a emitory musejí být propojeny stíněnými (poměrně tlustými) kabely s řídicí jednotkou, která koreluje jejich data a vyhodnocuje polohu senzorů v poli, což výrazně omezuje herce. Senzorů bývá typicky použito 6 až 11, což znamená stejný počet kabelů vedoucích k herci. Také to ovšem znamená nutnost dopočítávat rotace kloubů algoritmy inverzní kinematiky. V tomto případě jsou některé sledované body používány jako báze dílčích kinematických řetězců, zatímco jiné představují koncové efektory a jsou dopočítávány rotace v kloubech mezi nimi. Další nevýhodou magnetických systémů je citlivost na kovy v okolí. Na druhou stranu tyto systémy nejsou citlivé vůči zákrytům. Moderní systémy jsou také poměrně rychlé i přes nutnost filtrování šumu, které výrazně snižuje potenciálně vysokou rychlost vzorkování.
- *Optické snímací systémy*
  Tyto systémy zaznamenávají pohyb herce větším počtem kamer rozmístěných na přesně určených místech okolo scény. Technikami počítačového vidění se potom v záznamech jednotlivých kamer sledují (*tracking*) vybrané body, což jsou většinou kontrastní značky (*marker*) rozmístěné na těle herce nebo jiných sledovaných objektech. Dalším krokem bývá nalezení korespondencí mezi sledovanými body na záznamech různých kamer a výpočet trojrozměrných souřadnic sledovaných bodů. Nakonec se provede výpočet rotací v kloubech.

Optické snímání pohybu se zatím nepoužívá pro tvorbu animací v reálném čase, kvůli náročnému zpracování sekvencí z jednotlivých kamer. Má však jiné výhody. Především nevyžaduje žádný drahý speciální hardware, vystačí si s několika kalibrovanými kamerami. Herce také v podstatě neomezuje v pohybu, jediné požadavky na něj se omezují na přiléhavé jednobarevné oblečení (tmavé, matné) na němž vyniknou kontrastní (světlé, lesklé) značky. Možnost zákrytu značek při pohybu je jistou nevýhodou, kterou lze omezit použitím většího počtu kamer nebo predikcí při sledování bodu v záznamu kamery.

Do kategorie optického snímání pohybu patří řada metod v současné době zkoumaných v oboru počítačového vidění. Zaměřují se převážně na rekonstrukci bez použití značek (*markerless*) a na odstranění jiných omezení, například požadavku na speciální oblečení. Právě oblečení je schopno do značné míry měnit tvar lidské postavy, což ztěžuje detekci jednotlivých tělesných částí potřebných pro rekonstrukci pohybu. Tento výzkum se ale již vymyká z rámce snímání pohybu jako nástroje pro vytváření animací. Zamýšlené aplikace míří spíše do oblasti rozpoznávání pohybových aktivit například automatickými sledovacími systémy.

Rozvíjejí se také metody trojrozměrné rekonstrukce pohybu ze záznamu jedné kamery (*monocular motion reconstruction*), jejichž aplikace mohou stále sloužit účelům tvorby animací. Pohled jedné kamery neposkytuje dostatek informace o hloubce, data jsou jen dvojrozměrná. Tento nedostatek se kompenzuje použitím realistického modelu člověka a jeho lícováním s detekovaným průmětem do roviny (*skeleton fitting*).

Pohybová data pro jednotlivé stupně volnosti jsou většinou hodnoty rotace okolo příslušné souřadnicové osy. Jedná se tedy o *Eulerovy úhly*. Ty jsou vyjádřením rotace v trojrozměrném prostoru založeném na Eulerově teorému, podle nějž lze každou rotaci v prostoru vyjádřit třemi rotacemi okolo jednotlivých souřadnicových os. Rozkladů obecné rotace na Eulerovy úhly může být více, záleží na použité konvenci, interpretaci Eulerových úhlů. Nejčastěji se používají dvě varianty:

- *NASA standard aeroplane*: $\mathbf{R} = R_x\ R_y\ R_z$
- *NASA standard aerospace*: $\mathbf{R} = R_z^1\ R_y\ R_z^2$

Symbol $\mathbf{R}$ značí obecnou matici rotace, R_x, R_y, R_z matice rotací kolem os x, y a z (viz část 21.3.2). Reprezentace pomocí Eulerových úhlů je vhodná pro popis stavu kloubu, avšak pro účely zobrazení je nutno ji převést do tvaru transformační matice. Tu získáme vynásobením matic rotací kolem jednotlivých souřadnicových os. Bližší výklad zajímavé tématiky reprezentace rotací v trojrozměrném prostoru podává například kniha [Eber01].

Nasnímané či jinak vytvořené animace jsou záznamy konkrétního pohybu konkrétní osoby. Často potřebujeme animovat jinou (např. vyšší) postavu, nebo potřebujeme animaci jen málo odlišnou od nějaké již hotové (chůze s delším krokem). V takových případech se zdá zbytečné procházet celým procesem vytváření animace nebo snímání pohybu znovu. Výzkum, související především s filmovým a herním průmyslem, je tedy zaměřen na problémy přizpůsobování jedné animace více modelům (*motion retargetting*) [Glei98], nebo zobecnění animace pro nějaký proměnlivý parametr (*motion parameterization*) [Liu02].

Kapitola 19

Zobrazování rozsáhlých scén

V současné době není model o velikosti několika desítek miliónů polygonů ničím neobvyklým. Poměrně detailní model letadla Boeing 737 se skládá z 500M polygonů, model městské zástavby s relativně málo detaily obsahuje 8M polygonů. Pro urychlení zobrazování takto rozsáhlých scén využíváme řadu metod, které jsou popsány v této kapitole. Metody lze rozdělit do dvou základních kategorií:

- výpočty viditelnosti,
- zjednodušování scény.

V první skupině je cílem metod rychlé určení viditelných částí scény. Z pozice pozorovatele, který se nachází uvnitř scény, je zřejmě vidět jen část objektů. Velkou množinu objektů není třeba zpracovávat, ať již proto, že se nacházejí mimo zorný úhel či proto, že jsou zakryty (zastíněny) objekty v popředí. Druhá skupina metod vychází z úvahy, že vzdálené části scény není třeba zobrazovat se všemi detaily, neboť tyto detaily buď při dané velikosti obrazu nejsou rozlišitelné nebo jim uživatel nevěnuje pozornost, protože vnímá především objekty v popředí.

Při zobrazování v reálném čase se můžeme soustředit na tři základní kvalitativní kritéria: frekvence zobrazování, rozlišení, a realističnost scény. Rychlost generování snímků by v ideálním případě měla odpovídat obnovovací frekvenci monitoru, tedy okolo $60 - 100$Hz. Vyšší rychlost již nepřináší další zvýšení kvality vjemu. Rozlišení by mělo odpovídat velikosti zobrazovacího zařízení, 1600×1200 je považováno za dostatečné pro většinu displejů. Na rozdíl od těchto dvou kritérií neexistuje žádná přirozená míra, která by vyjádřila realističnost zobrazení scény. Představy uživatelů o kvalitě zobrazení se přitom stále zvyšují – počítačem generované obrazy jsou srovnávány s obrazy skutečného světa kolem nás, dalším faktorem je fantazie uživatelů a tvůrců počítačových modelů. Realističnost scény je ovlivňována jak jejím detailnějším popisem, tak kvalitou simulace osvětlení ve scéně.

19.1 Výpočty viditelnosti

Pro rychlé zobrazení scény s mnoha objekty (plochami) s ohledem na viditelnost je třeba rozhodnout, které objekty lze z dalšího zpracování v zobrazovacím řetězci vyřadit a které objekty naopak podrobit detailnímu (klasickému) vyhodnocení viditelnosti. Algoritmy dělíme na základě kvality poskytovaného výsledku do následujících kategorií (dle [Nire02]):

- přesné,
- konzervativní,
- agresivní,
- přibližné.

Přesné algoritmy poskytují exaktní řešení daného problému – naleznou právě tu množinu objektů, kterou je nutno zobrazit. Konzervativní algoritmy „nadhodnocují“ viditelnost, tzn. jako výsledek poskytují nadmnožinu skutečně viditelných bodů, objektů či ploch. Agresivní algoritmy naopak „podhodnocují“ viditelnost, tzn. jejich výsledkem je pouze podmnožina viditelných objektů. Přibližné algoritmy pak poskytují pouze aproximaci exaktního výsledku, u níž nelze s jistotou určit, zda je podmnožinou či nadmnožinou exaktního řešení.

Dalším důležitým kritériem dělení algoritmů viditelnosti je forma jejich využití v dané aplikaci. Na základě tohoto kritéria dělíme algoritmy na:

- algoritmy pracující v reálném čase,
- algoritmy předzpracování.

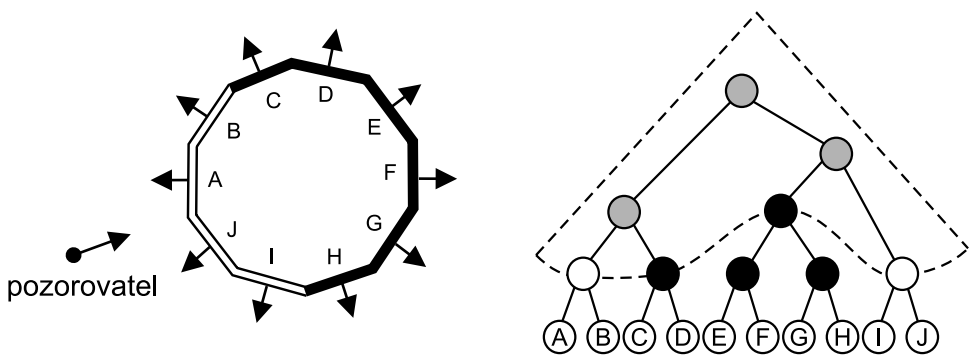
U algoritmů pracujících v reálném čase je stěžejním kritériem doba výpočtu. Typickými představiteli takových algoritmů jsou metody odstraňování zastíněných objektů. Jedná se většinou o konzervativní algoritmy, kde kvalita výsledku je podřízena rychlosti zpracování. Naproti tomu u algoritmů předzpracování není doba výpočtu klíčovým faktorem, ale je žádoucí, aby tyto algoritmy poskytly co nejpřesnější výsledky.

V této části popíšeme převážně konzervativní algoritmy, které slouží jako urychlovací techniky pro klasickou paměť hloubky. Následující odstavec pojednává o základních metodách odstraňování neviditelných polygonů. Část 19.1.2 diskutuje algoritmy odstraňování zastíněných polygonů v reálném čase. Závěr je věnován algoritmům předzpracování viditelnosti. Téma určování viditelnosti v rozsáhlých scénách je v počítačové grafice stále aktuálním předmětem výzkumu a vhodné algoritmy jsou přesouvány do grafických procesorů. Vzhledem k omezenému rozsahu knihy jsou algoritmy popsány jen přehledově, bez implementačních detailů. K dalšímu studiu doporučujeme například knihu [Moll02].

19.1.1 Základní techniky odstraňování neviditelných polygonů

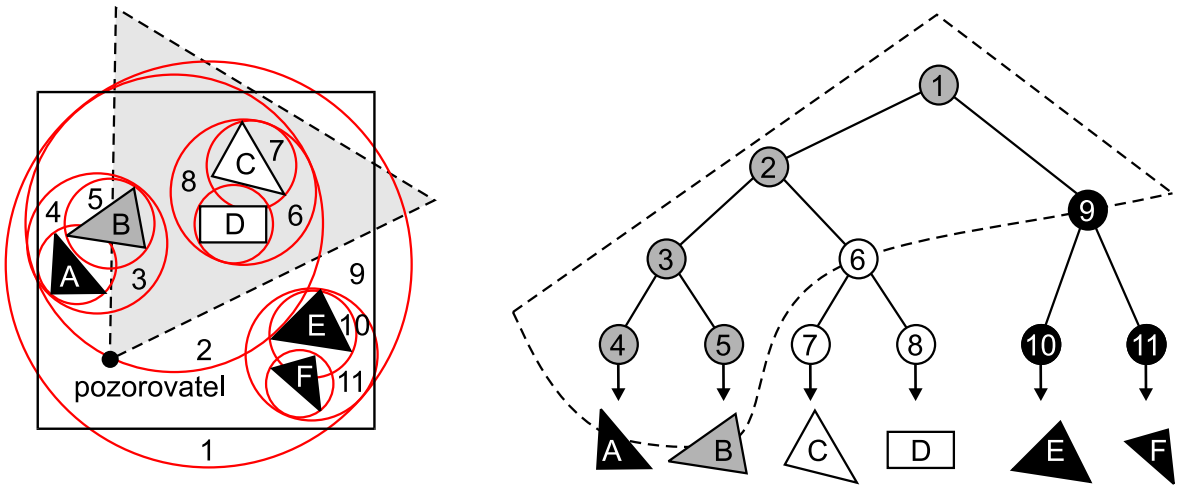
Základními technikami pro určování viditelných objektů jsou *odstraňování odvrácených stěn* (*backface culling*) (viz část 11.1) a *odstraňování objektů pohledovým jehlanem* (*viewing frustum culling*). Obě tyto techniky jsou dnes většinou již implementovány na úrovni polygonů přímo v grafickém procesoru. Problémem však zůstává fakt, že polygony musíme posílat do grafického procesoru, který provádí test jejich viditelnosti poměrně pozdě v rámci zobrazovacího řetězce. Většina moderních zobrazovacích systémů proto aplikuje testy pro odstranění neviditelných polygonů na úrovni modelovací nebo prostorové hierarchie scény a dokáže tak předejít zbytečnému zpracování rozsáhlých bloků polygonů.

Odstraňování odvrácených stěn eliminuje v průměru 50 % polygonů. Klíčem k efektivnímu odstraňování odvrácených polygonů je jejich shlukování na základě podobnosti normál, které umožňuje pomocí relativně malého počtu testů eliminovat velké množství polygonů [Kuma96]. Polygony, jejichž normály jsou orientovány podobným směrem, seskupíme a nad těmito skupinami postavíme binární strom. Vnitřní uzly tohoto stromu reprezentují skupiny polygonů, jejichž normály se nacházejí v omezeném prostorovém úhlu. Tento úhel je tím větší, čím blíže je uzel ke kořeni. Při zobrazování procházíme strom od kořene a podle směru pohledu pozorovatele vybíráme skupiny uzlů představující přivrácené polygony. Tato technika je ilustrována na obrázku 19.1, na němž jsou černé uzly označeny jako neviditelné a bílé jako viditelné. Křivka vpravo označuje řez, v němž bylo další procházení hierarchie ukončeno.



Obrázek 19.1: Odstraňování odvrácených polygonů pomocí shlukování (zjednodušený nákres v rovině). Prořezáváním hierarchie (vpravo) odstraníme celé skupiny odvrácených polygonů.

Další základní technikou je odstraňování polygonů pohledovým objemem. Tato technika je vlastně jednodušší variantou ořezávání podle pohledového objemu (viz část 9.5), kde namísto ořízntého polygonu je výsledkem pouze informace, zda polygon alespoň částečně leží v pohledovém objemu. Tento výpočet musí být proveden velmi rychle a aplikován pokud možno na větší



Obrázek 19.2: Odstraňování polygonů pohledovým jehlanem s pomocí hierarchie obálek

skupiny polygonů, aby se jeho aplikací dosáhlo významného urychlení. Pro efektivní realizaci této techniky se využívá buď graf scény nebo hierarchické datové struktury, typicky hierarchie obálek, oktalové stromy, BSP stromy, či kD-stromy (viz část 14.2). Při procházení stromu zjišťujeme, zda pohledový objem obsahuje (protíná) příslušnou část prostoru či scény. Pokud ne, je celý podstrom neviditelný. V opačném případě algoritmus rekurzívně pokračuje testováním následníků uzlu. Použití této metody na hierarchii obalových koulí ilustruje obrázek 19.2. Černě označené uzly leží zcela vně pohledového jehlanu a jsou tedy neviditelné. Bílé uzly leží celé uvnitř jehlanu a šedivé jehlan protínají – obě skupiny jsou potenciálně viditelné.

Tento základní algoritmus je možno dále optimalizovat při využití skutečnosti, že při malé změně pozice nebo směru pohledu pozorovatele se množina viditelných objektů při zobrazení následujícího snímku změní jen minimálně.

19.1.2 Odstraňování zastíněných objektů

Základní techniky odstraňování neviditelných polygonů nejsou schopny vyloučit veškeré polygony, které pozorovatel nemůže vidět, protože jsou zakryté jinými objekty. To znamená, že i zastíněné polygony jsou zaslány do grafického procesoru, kde jsou pomocí paměti hloubky eliminovány až při zápisu do obrazové paměti.

Představme si situaci, kdy zobrazujeme model města, ze kterého jsou z dané pozice pozorovatele viditelné jen nejbližší přilehlé ulice tvořící pouze 1 % celého modelu. Pokud nepoužijeme detekci zastíněných polygonů, je velká část celého modelu, která se nachází v pohledovém

jehlanu, zaslána ke zpracování do grafického procesoru. Celých 99 % polygonů bude eliminováno až na konci zobrazovacího řetězce a tudíž veškeré předchozí kroky k jejich zpracování byly nadbytečné. Současné grafické procesory sice obsahují optimalizace posouvající hloubkový test do popředí zobrazovacího řetězce, to však v daném případě vede jen k malému zrychlení.

V této části se seznámíme s několika metodami, které řeší uvedený problém tak, že vyloučí jednotlivé polygony nebo celé skupiny polygonů ještě dříve, než jsou podrobeny dalšímu zpracování. Všechny tyto algoritmy si nejprve připraví strukturu, která zachycuje možné zaclonění scény jednotlivými polygony nebo skupinami polygonů. Ta se nazývá *struktura zastínění* a je buď závislá na pozici pozorovatele, nebo může být řešena obecně pro libovolná místa pozorování. Zároveň se využívá prostorové třídění scény v podobě *prostorové hierarchie* (viz část 14.2). Při vyhodnocování viditelnosti je nutno procházet jak prostorovou hierarchii scény, tak strukturu zastínění, která také často mívá hierarchický charakter. Obecné schéma tohoto kombinovaného procházení je uvedeno v algoritmu 19.1, který představuje společnou kostru všech následujících metod pro odstraňování zastíněných polygonů. Konkrétní algoritmy se od uvedeného schématu mohou drobně lišit – jejich zásadní odlišnosti se projevují zejména ve způsobu reprezentace struktury zastínění a jejím využití pro test viditelnosti (bod 2 algoritmu).

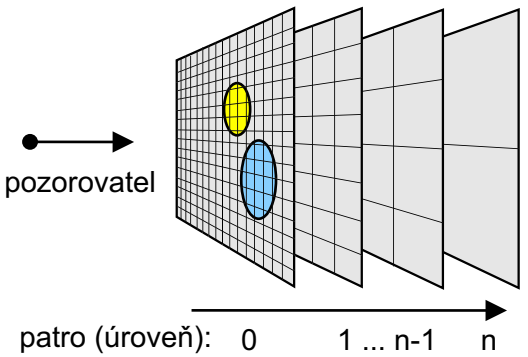
Vstupy: Prostorová hierarchie scény + Struktura zastínění
Výstup: Obraz viditelné části scény

1. Procházej prostorovou hierarchii směrem od kořene dolů a současně od pozorovatele směrem do scény.
2. Pro aktuální uzel hierarchie aplikuj *test viditelnosti* uzlu vzhledem k aktuální struktuře zastínění. Tento test ohodnotí uzel jako neviditelný, či viditelný.
3. Pokud je uzel neviditelný, ukonči procházení aktuální větve hierarchie.
4. Pokud je uzel viditelný:
   (a) Pokud uzel není listem, rekurzívně pokračuj bodem 1 v sestupu.
   (b) Pokud uzel je listem:
      i. Označ všechny polygony obsažené v daném uzlu za viditelné a zobraz je klasickou technikou s řešením viditelnosti (z-buffer).
      ii. Aktualizuj strukturu zastínění polygony obsaženými v daném uzlu. Tyto viditelné polygony představují stínící objekty pro ostatní části scény.

Algoritmus 19.1: Obecný algoritmus odstraňování objektů s využitím prostorové hierarchie scény.

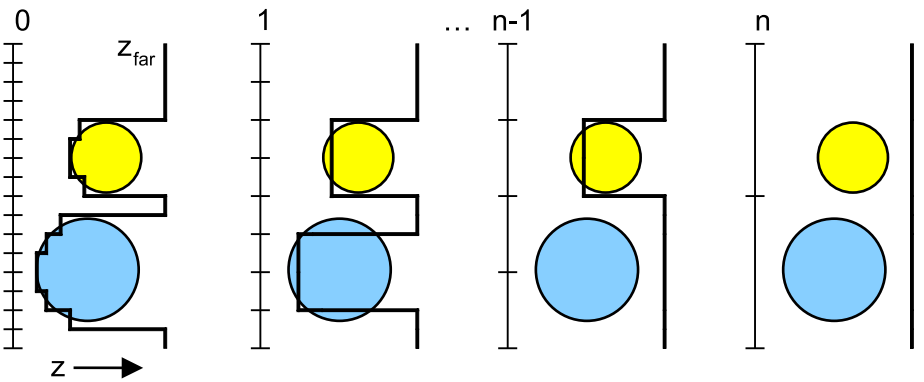
Hierarchická paměť hloubky

V této metodě kombinujeme prostorovou hierarchii scény a hierarchickou paměť hloubky (*hierarchický z-buffer*), abychom nahrubo zjistili, zda jsou velké části prostoru zastíněny [Gree93].



Obrázek 19.3: Hierarchická paměť hloubky tvoří z-pyramidu. Její podstava je reprezentována nejlevější pamětí hloubky.

K tomu využíváme *z-pyramidu*, jejíž nejnižší (nulté) patro odpovídá funkci i rozlišením klasické paměti hloubky (obr. 19.3). Vyšší patra pyramidy jsou vždy převzorkována do poloviční rozlišení tak, aby každý nový záznam odpovídal nejvzdálenějšímu záznamu příslušné čtveřice z nižšího patra (obr. 19.4). Z-pyramida reprezentuje v algoritmu 19.1 strukturu zastínění. Nejčastěji se využívá ve spojitosti s oktalovým stromem uchovávajícím scénu (viz část 14.2.2). Uzly tohoto stromu jsou rekurzívně testovány na viditelnost s použitím odpovídajících obálek. Hloubka obálky je nejprve porovnána s kořenem (vrcholem) z-pyramidy. Pokud je obálka hlouběji, je neviditelná (bod 3 algoritmu 19.1). Pokud ne, test pokračuje v nižších patrech z-pyramidy v buňkách, které obsahují průmět dané obálky. Hloubky nově zapsaných pixelů v nejnižším patře z-pyramidy jsou poté použity k aktualizaci hodnot ve vyšších patrech (bod ii algoritmu 19.1).



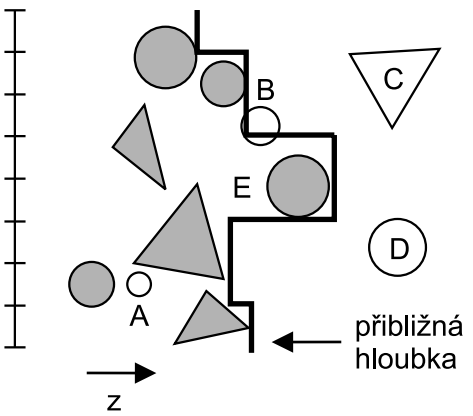
Obrázek 19.4: Vztah mezi hodnotami hloubek v sousedních patrech z-pyramidy. Každý záznam určité úrovně obsahuje nejvzdálenější z odpovídajících záznamů předchozí úrovně.

Pokud není hierarchická paměť hloubky podporována grafickým procesorem, je metoda málo efektivní. Můžeme ji částečně vylepšit dvouprůchodovým algoritmem, který využívá časové koherence viditelnosti. Nejprve vykreslíme všechny objekty viditelné v předchozím snímku. Tím získáme odhad neprůhlednosti při základním rozlišení. Z údajů v paměti hloubky sestavíme z-pyramidu v hlavní paměti. S ní provádíme testy uzlů prostorové hierarchie pro další snímek.

Hierarchické mapy zastínění

Hierarchické mapy zastínění [Zhan97] využívají podobné myšlenky jako hierarchický z-buffer, jsou však navrženy pro použití standardního grafického procesoru, který neobsahuje podporu z-pyramidy. Algoritmus ke své činnosti vyžaduje určení množiny polygonů, které mají výrazný stínící účinek. Nalezení těchto stínících polygonů je samostatným problémem, který se řeší buď rychlými heuristikami využívajícími velikosti průmětu ploch nebo předzpracováním a vytvořením databáze stínících polygonů.

Stínící polygony jsou použity k vytvoření nejnižšího patra mapy zastínění (patra s nejvyšším rozlišením). Každý záznam v mapě obsahuje úroveň zastínění v intervalu 0 až 1, kde 0 znamená žádné zastínění, 1 plné zastínění. Pixely zakryté alespoň jedním polygonem obsahují hodnotu jedna, ostatní nula. Vyšší úrovně mapy jsou vypočteny převzorkováním a filtrováním: každý pixel je průměrem čtyř odpovídajících pixelů nižší úrovně. K tomuto převzorkování můžeme využít texturovací jednotky, které jsou součástí moderních grafických procesorů. Výsledkem je hierarchická mapa, jejíž záznamy (pixely) reprezentují relativní podíl pixelů, do kterých se v odpovídající oblasti obrázku promítá alespoň jeden stínící polygon. Tato hierarchie představuje v algoritmu 19.1 strukturu zastínění.



Obrázek 19.5: Paměť přibližné hloubky. Stínící objekty (*occluder*) jsou tmavé, zastíněné (*occludee*) bílé.

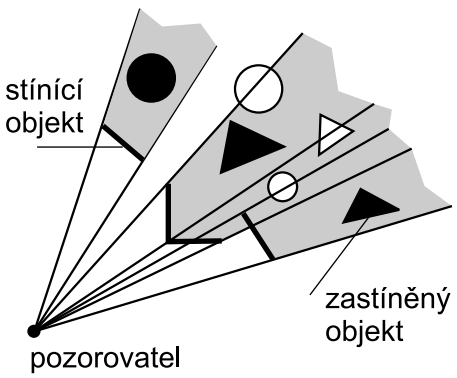
V testu viditelnosti je pak tato mapa využívána k rychlému určení, zda průmět testovaného polygonu nebo obálky je celý obsažen v oblasti, která je potenciálně zastíněna. Zde je nutno zdůraznit, že mapa zastínění neobsahuje žádnou informaci o hloubce. Hloubky stínících polygonů jsou reprezentovány odděleně v *paměti přibližné hloubky* (*depth estimation buffer*) s malým rozlišením, typicky 64×64. Tato paměť je vystavěna nad obálkami stínících polygonů tak, že její záznamy jsou konzervativní reprezentací hloubky.

Test viditelnosti obálky (bod 2 algoritmu 19.1) je rekurzívně aplikován nejprve v nejvyšší úrovni mapy zastínění. Pokud aktuální záznam v mapě zastínění obsahuje hodnotu jedna (plné zastínění), ukončíme procházení aktuální větve prostorové hierarchie scény. V opačném případě test rekurzívně pokračuje v buňkách nižšího patra mapy, které obsahují průmět dané obálky. Pokud dospějeme alespoň k jedné buňce mapy zastínění, která není zastíněna (hodnota < 1) je testovaná obálka označena za viditelnou a test je ukončen. Pokud je procházení ukončeno výlučně v plně zastíněných buňkách mapy, přistoupíme k hloubkovému testu. Hloubky pixelů promítnuté obálky porovnáme se záznamy v paměti přibližné hloubky. Pouze v případě, že obálka leží za všemi porovnávanými záznamy, je označena za neviditelnou.

Uvedený algoritmus můžeme vylepšit dvěmi jednoduchými úpravami. První metoda ukončuje testování v řídce zastíněných částech mapy (např. hodnoty < 0.1), jejichž velikost je v průmětu srovnatelná s průmětem testované obálky. Zde je zřejmé, že pravděpodobnost, že testovaná obálka bude zastíněná po sestupu do nižší úrovně mapy, je velmi malá. Druhá metoda ukončuje testování v hustě zastíněných částech mapy (např. hodnoty > 0.9) a předpokládá, že testovaná obálka je zastíněna v celé odpovídající oblasti. Nevýhodou této modifikace je však skutečnost, že výsledný algoritmus již není konzervativní, nýbrž pouze přibližný.

Stínové jehlany

Jedním z nejjednodušších algoritmů pro testování zastínění je metoda stínových jehlanů [Huds97].



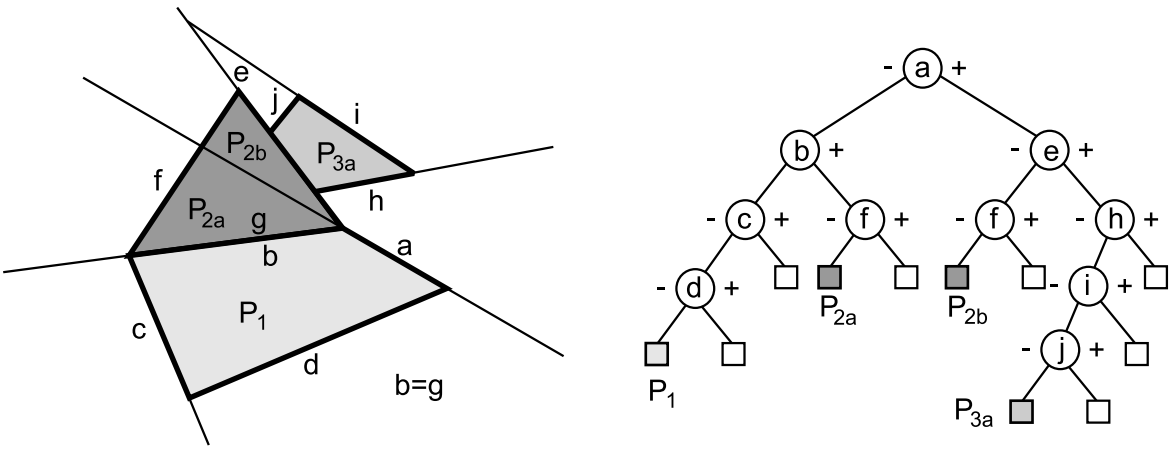
Obrázek 19.6: Stínové jehlany zkonstruované pro čtyři stínící polygony

Podobně jako pro mapy zastínění určíme vhodné stínící polygony a pro každý polygon vytvoříme stínový jehlan. Následně opět testujeme jednotlivá tělesa nebo jejich skupiny uspořádané v prostorové hierarchii. Pokud je celý uzel hierarchie obsažen v libovolném jehlanu, je označen za neviditelný.

Nevýhodou metody je, že neodhalíme zastínění způsobené kumulativním efektem několika stínících polygonů, protože stínové jehlany jsou použity pro určení neviditelných objektů izolovaně. To je dobře vidět na obrázku 19.6, v němž pouze tmavé objekty jsou určeny jako neviditelné, zatímco několik dalších zjevně zastíněných objektů detekováno nebylo. Metoda navíc vykazuje lineární nárůst složitosti testu s počtem stínových jehlanů. Výhodnějšího, logaritmického růstu složitosti lze docílit vhodným uspořádáním stínových jehlanů, které popisují následující techniky.

Stromy zastínění

Nevýhody metody stínových jehlanů lze odstranit s využitím stromů zastínění [Bitt98]. Strom zastínění (*occlusion tree*) je hierarchickou reprezentací stínového tělesa [Chin89], viz též kapitola 12.2. Jedná se o BSP strom, jehož uzly odpovídají rovinám procházejícím hranami vybraných stínících polygonů a listy odpovídají fragmentům těchto polygonů (obr. 19.7). Fragmentem nazýváme část původního polygonu, která vznikla rozříznutím dělicí rovinou. Test viditelnosti s využitím stromů zastínění je přesnější než u stínových jehlanů, jelikož postihuje i kumulativní zastínění. Navíc díky hierarchické struktuře roste časová náročnost pro větší počet stínících polygonů sublineárně.



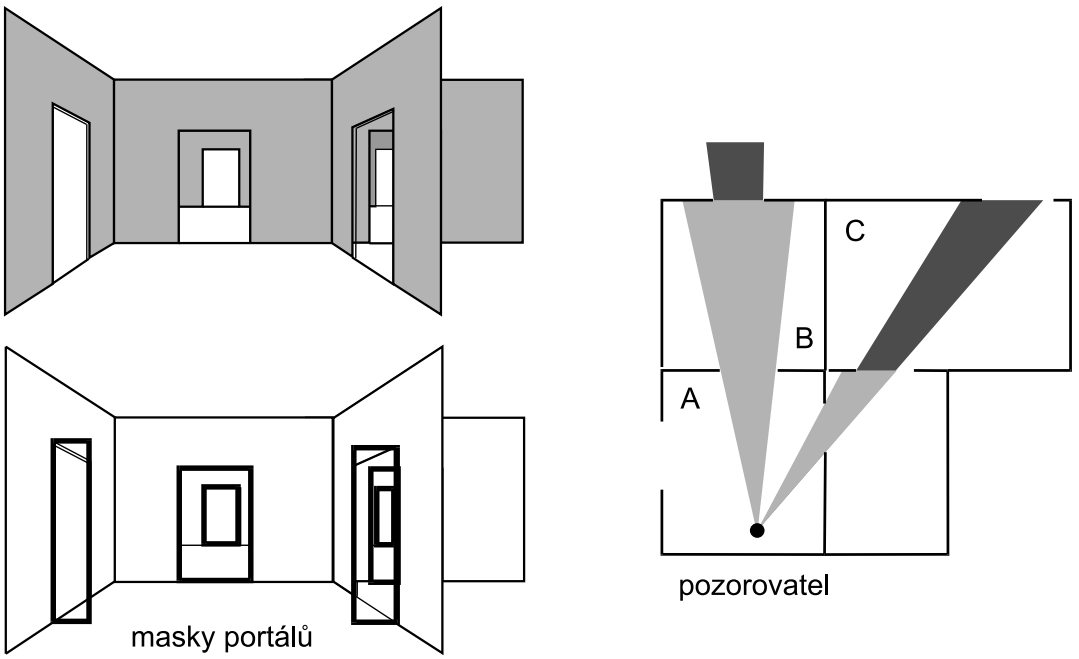
Obrázek 19.7: Příklad stromu zastínění pro tři stínící polygony. Prázdné čtverečky odpovídají nezastíněným oblastem prostoru, vyplněné čtverečky fragmentům stínících polygonů.

Všimněme si podobnosti s BSP stromem, použitým pro řešení viditelnosti v kapitole 11. Zatímco v části 11.2.3 byl strom konstruován pomocí řezů proložených rovinami všech polygonů, zde jsou řezy vedeny hranami polygonů stínících.

Buňky a portály

Příkladem scén, v nichž algoritmy určování zastínění mohou dosáhnout řádového urychlení, jsou modely interiérů budov. Tyto modely lze prostorově rozdělit na *buňky* zhruba odpovídající jednotlivým místnostem. Buňky jsou pak spojeny *portály*, které zhruba odpovídají dveřím a oknům. V takto rozdělené scéně lze aplikovat poměrně jednoduchý algoritmus, který však v praxi dosahuje velmi dobrých výsledků [Lueb95]. Scéna je procházena od buňky obsahující pozorovatele. Pro každou navštívenou buňku zobrazíme všechny objekty, které obsahuje.

Následně otestujeme, zda skrze portály navštívené buňky vidíme do dalších doposud nenavštívených buněk. Tento test je prováděn pomocí portálových masek, které jsou průnikem doposud zpracovaných portálů (obr. 19.8).



Obrázek 19.8: Detekce zastínění využívající buňky a portály. Portály vytvářejí masky (zvýrazněné vlevo dole pomocí obdélníků), jejichž průnikem řídíme procházení grafu buněk.

Horizont zastínění

Zvláštním případem rozsáhlých scén jsou modely městské zástavby. Pro ně je vhodný algoritmus, který reprezentuje aktuální zastínění pomocí po částech konstantního horizontu [Down01]. Takto aproximovaný horizont je uchováván ve vyváženém binárním stromu. Scénu procházíme odpředu dozadu a postupně obnovujeme horizont přidáváním zpracovávaných polygonů. Pro každý uzel hierarchie scény testujeme, zda obálka uzlu je pod aktuálním horizontem. V takovém případě je uzel neviditelný. V opačném případě sestupujeme v hierarchii níže a aktualizujeme horizont pomocí polygonů v příslušných listech. Tento postup je vlastně zobecněním metody plovoucího horizontu z části 11.2.5. Namísto jednoduché rastrové reprezentace horizontu je zde použita univerzálnější stromová struktura.

19.1.3 Předzpracování viditelnosti

Při použití výše popsaných metod předpokládáme, že po odstranění neviditelných objektů následuje okamžité vykreslení viditelných částí scény. Pokud však nasazení těchto metod nepostačuje k dosažení potřebné rychlosti zobrazování, musíme využít metod předpočítání viditelnosti. Výpočet viditelnosti je v takovém případě proveden předem a může trvat i poměrně dlouhou dobu. Jeho výsledky uložíme a při následném (rychlém) zobrazování scény pouze identifikujeme množinu předpočítaných objektů viditelnou z místa pozorovatele a jeho okolí.

V době předzpracování nejsou známy informace o pohybu pozorovatele ve scéně. Viditelnost proto musí být vyhodnocena „globálně“ s ohledem na různé situace, které mohou nastat ve fázi zobrazování. Klíčem k efektivnímu předpočítání viditelnosti je znalost vlastností scény. Metody rozdělujeme podle typu scény do několika skupin. Uvedeme techniky vhodné pro interiérové scény, městskou zástavbu a pro scény obecného charakteru.

Interiérové scény

V interiérových scénách opět využíváme dělení na buňky a portály. Pro každou buňku vypočítáme *potenciálně viditelnou množinu objektů* (*PVS, Potentially Visible Set*), tj. množinu objektů, které jsou vidět z některé pozice v rámci buňky [Aire90]. Tento výpočet lze realizovat vrháním paprsků skrz sekvence portálů navštívených během procházení grafu buněk. Přesnější metody pak využívají analytických geometrických testů pro určení existence přímky protínající danou sekvenci portálů [Tell91].

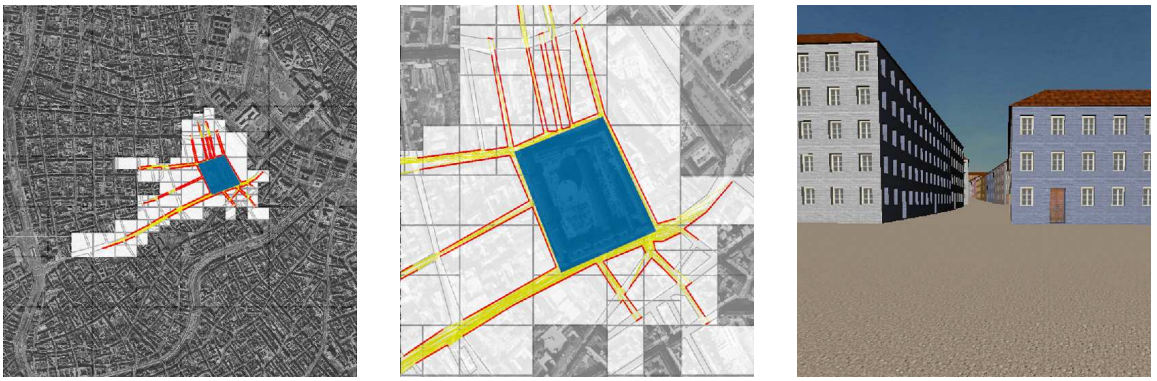
Pro každou buňku vypočítáme buď seznam buněk potenciálně viditelných nebo přímo seznam potenciálně viditelných objektů. Při procházení scény tedy stačí určit buňku, ve které se pozorovatel právě nachází a zobrazovat pouze potenciálně viditelné objekty. Přesné řešení viditelnosti z dané pozice se ponechává na paměti hloubky, podobně jako u metod odstraňování neviditelných objektů.

Zmíněné metody jsou efektivní jen pro hustě zastíněné scény, kde existuje přirozené dělení na buňky a portály. V opačném případě však může předvýpočet trvat neúnosně dlouho, neboť složitost algoritmu roste exponenciálně s délkou sekvencí portálů.

Městská zástavba

Pro scény městské zástavby lze s výhodou využít jejich převážně 2.5D charakter, neboť většinu ploch v modelech měst představují vertikálně orientované stěny domů. Jeden z algoritmů předpočítání viditelnosti v takových scénách využívá tzv. *objektové stíny* (*occluder shadows*) [Wonk00]. Metoda shromažďuje pro každou buňku půdorysné projekce stínů vybraných stínících objektů pomocí paměti hloubky. Výsledný obsah paměti hloubky je následně využit pro testování viditelnosti všech objektů ve scéně. Jinou možností je využití geometrického

popisu zastíněného objemu k přesnějšímu a mnohdy i rychlejšímu určení potenciálně viditelné množiny [Bitt01].



(a) (b) (c)

Obrázek 19.9: Potenciálně viditelné množiny v modelu městské zástavby. (a) Pohled na 8km^2 města Vídně, a s označenou buňkou o rozloze 0.1km^2 (uprostřed). Potenciálně viditelná množina vzhledem k dané buňce (náměstí) je tvořena světlými oblastmi s tmavě orámovanými ulicemi. Tmavé oblasti jsou neviditelné. (b) Bližší pohled na danou oblast. (c) Pohled z pozice pozorovatele nacházejícího se uvnitř dané buňky. (Obrázky poskytl J. Bittner, ČVUT v Praze.)

Obecné scény

Efektivní výpočet potenciální viditelnosti v obecných scénách je náročným problémem a popis odpovídajících metod přesahuje rozsah této knihy. Teprve velice nedávno byly publikovány metody pro přesné řešení toho problému [Bitt02, Nire02]. Tyto metody využívají principu duality a množinových operací ve vícerozměrném prostoru k určení existence přímek, skrze které je možno vidět objekty patřící do potenciálně viditelné množiny dané buňky.

Jednodušší metody využívají diskrétní reprezentaci scény [Scha00] nebo stínového objemu [Dura00], k určení konzervativní nadmnožiny potenciálně viditelných objektů.

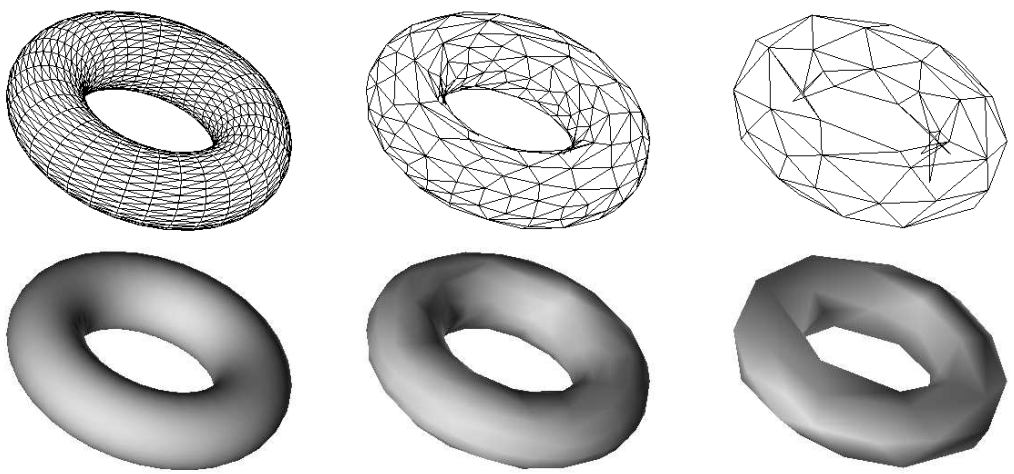
19.2 Zjednodušování scény

Zjednodušování scény se může odehrávat na úrovni sítí trojúhelníků, objektů, skupin objektů či ještě větších částí scény. Některé metody zachovávají reprezentaci zjednodušovaného objektu

(například při tzv. decimaci sítě trojúhelníků na obrázku 19.10 je výsledkem opět síť trojúhelníků), jiné přecházejí ze složitější datové reprezentace do jednodušší, například nahrazují geometrické prvky jedním nebo několika obrázky.

Společným pojmem používaným při zjednodušování scény je zkratka *LOD* (*Level Of Detail*), kterou bychom mohli slovy nazvat *stupeň* či *úroveň detailu*. Tento bezrozměrný údaj je většinou chápán intuitivně jako možnost odebírat a přidávat určitému objektu jeho detaily. Nejvyšší LOD reprezentuje objekt se všemi detaily, nejnižší LOD pak co nejvíce zjednodušenou reprezentaci, kterou jsme ochotni akceptovat jako náhradu původního objektu.

Stupeň detailu se snižuje či zvyšuje v závislosti na měnících se podmínkách při zobrazování scény. Nejčastěji je kritériem pro změnu LOD vzdálenost mezi pozorovatelem a objektem. Mezi další používaná kritéria patří velikost průmětu objektu na plochu obrazovky (tj. počet pixelů, v nichž je daný objekt vidět), velikost prostorového úhlu, v němž je objekt či jeho obálka vidět z pozice pozorovatele, či celkový počet polygonů v pohledovém objemu. Je možno brát zřetel i na způsob lidského vnímání a detailněji zobrazovat objekty ve směru pohledu (ve středu obrazovky) a směrem ke stranám kvalitu modelu snižovat. Jinou možností je měřit a uvažovat chybu ve výsledném obraze, kterou změna LOD způsobí [Ciam97].



Obrázek 19.10: Postupně zjednodušovaná síť trojúhelníků pokrývající povrch toroidu a její vystínovaný obraz. Počty trojúhelníků zleva doprava: 2 048, 512 a 128.

Obrázek 19.10 dokumentuje, že díky stínování může být zjednodušený objekt vizuálně velmi podobný obrazu původního objektu. Výraznějších rozdílů si všimneme nejprve na obrysu, teprve později na ostatních částech sítě. Je také zřejmé, že dříve reagujeme na odlišnosti u známých

tvarů, jakým je například toroid. Obecně zakřivené tvary je proto možno zjednodušovat více, než sítě reprezentující pravidelné objekty.

Při použití LOD se setkáváme s jevem nazývaným *vyskakování* částí objektu (*popping effect*). Projevuje se tím, že při zvýšení či snížení stupně detailu se objeví nebo zmizí skupina trojúhelníků. Toto nežádoucí chování lze omezit pomocí obrazových technik – v předstihu vytvoříme snímek po změně LOD a poté interpolujeme mezi aktuálním a tímto snímkem. Tuto operaci navíc můžeme aplikovat lokálně jen pro obraz objektu se změněným stupněm LOD. Někdy se též provádí morfing geometrie jednoho diskrétního stupně detailu na druhý.

Z hlediska reprezentace scény a jejích částí je vhodné rozdělit metody LOD do dvou tříd. První z nich je zaměřena na zpracování geometrických stupňů detailu. Druhá skupina algoritmů využívá obrazové, tedy především dvojrozměrné informace.

19.2.1 Geometrické stupně detailu

Do této kategorie řadíme především metody, které pracují s polygonální reprezentací objektů, například s tělesy či terénem.

Spojité stupně LOD

Stupeň detailu se u těchto metod mění v jemných krocích, které provedou malou změnu datové struktury, typicky přidají či odeberou jeden trojúhelník. Vychází se z decimačních algoritmů, které popíšeme v dalším textu. Algoritmus zjednodušování sítě je výhodné provést již ve fázi předzpracování a jeho výsledky uložit do vhodné datové struktury tak, aby při vykreslování nebylo nutno používat složité geometrické výpočty, ale pouze přidávat nebo ubírat trojúhelníky. Protože uchováváme různé reprezentace, vyžaduje datová struktura, umožňující pro zadaný stupeň LOD poskytnout potřebný model, větší paměťové nároky než statická síť trojúhelníků.

Metoda, známá pod názvem *progressive meshes* [Hopp96], zachycuje všechny jednotlivé kroky zjednodušení sítě. Z dané úrovně zjednodušení lze tedy postupovat oběma směry, tj. zkvalitnit i zjednodušit popis. Další možností reprezentace LOD je uložení všech trojúhelníků ze všech úrovní popisu modelu najednou spolu s ohodnocením [Ciam97], které udává velikost chyby způsobené postupným zjednodušováním. Uspořádání trojúhelníků do stromové struktury umožní rychle vyhledat trojúhelníky potřebné k zobrazení požadovaného stupně detailu.

Diskrétní stupně LOD

Jemné, dílčí změny LOD vyžadují jednak použití pokročilejších datových struktur, jednak průběžné vyhodnocování kvality obrazu a s ním spojenou aktualizaci zobrazované sítě trojúhelníků. Pro jednodušší aplikace se proto používají diskrétní stupně LOD, které nejsou tak výpočetně

náročné. Pracuje se na úrovni celých objektů, pro které je třeba předem připravit několik (typicky 3 až 5) samostatných reprezentací s klesajícím množstvím detailů. Pro přípravu lze sice použít automatizované decimační postupy, avšak častěji tyto méně detailní modely připravuje sám návrhář, který je schopen posoudit vizuální významnost detailů a vytvořit jednotlivé stupně LOD podle sémantiky modelu. Je dobré brát v potaz i další fakta ovlivňující výpočetní náročnost, například velikost a počet textur objektu či použití poloprůhledných částí.

Příkladem modelu s diskrétním stupněm LOD může být model automobilu. Nejvyšší stupeň LOD obsahuje veškeré detaily vozidla, včetně textur na sedadlech a poloprůhledných kouřových skel. Následující stupeň obsahuje pouze největší části automobilu (kapota, kola), neprůhledné a bez textur. Další stupeň mohou představovat dva barevné kvádry na sobě a nejjednodušší reprezentací je jediný kvádr či dokonce prázdný model, který se použije v situaci, kdy automobil je od pozorovatele vzdálen například více než 1 km.

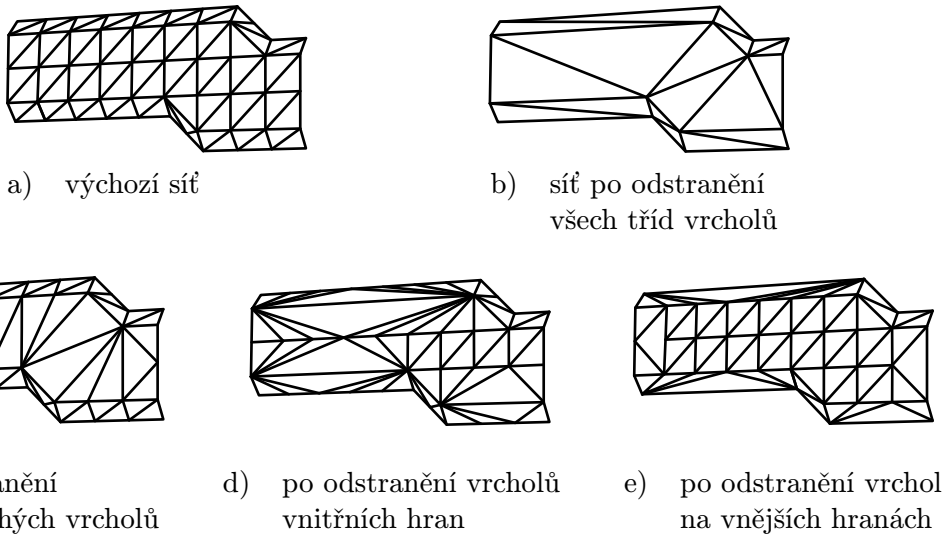
Každé diskrétní úrovni je přiřazeno číslo, které představuje vzdálenost, u níž při zobrazování dochází k přepnutí na další reprezentaci. Nevýhodou popsaného přístupu je redundance dat a skokové přecházení z jedné reprezentace do druhé, způsobená nepřítomností informací o vztahu jednotlivých částí modelu. Výhodou pak snadnost implementace a jednoduchost rozhodování o změně LOD, která je založena pouze na průběžném měření vzdálenosti mezi uživatelem a objektem. Tento přístup je používán v jednodušších aplikacích virtuální reality.

19.2.2 Zjednodušování sítě trojúhelníků

Požadavek na zjednodušení sítě trojúhelníků se objevuje v řadě úloh, například při zjednodušování modelů vzniklých převodem objemové reprezentace do povrchové (viz část 7.3.2) či vzniklých volným tvarováním těles (viz část 6.4.2), při zpracování sítě získané trojrozměrným snímáním objektů nebo při generování reprezentací s různými stupni LOD. Při snižování počtu trojúhelníků sítě dochází většinou ke zhoršení kvality reprezentace modelu, neboť algoritmy postupně nacházejí a vynechávají méně důležité detaily. Metody, které zachovávají přesnost modelu, triangulují rovinné plošky menším množstvím trojúhelníků.

Metody zjednodušování povrchu lze dále dělit podle toho, zda zachovávají či nezachovávají topologii nebo podle toho, zda zachovávají podmnožinu původních vrcholů sítě či zda vytvoří novou, zjednodušenou síť převzorkováním. Na obrázku 19.11 je příklad metody, která zachovává topologii i podmnožinu původních vrcholů [Schr92] a je známa pod názvem *decimace sítí trojúhelníků*. Metoda pracuje lokálně. Postupně prochází vrcholy sítě a pro každý zjišťuje, jak je důležitý pro zachování tvaru v okolí vrcholu. Pokud je vrchol vyhodnocen jako nedůležitý, odstraní jej a vzniklou díru vyplní triangulací.

Kromě lokálních metod zachovávajících topologii se lze setkat i s globálními metodami, které ohodnocují kvalitu sítě jako celku (např. pomocí hodnoty energie sítě) a problém *optimalizace sítě* převádějí na optimalizaci této funkce [Hopp96]. Energie sítě se stanovuje např. v závislosti



a) výchozí síť

b) síť po odstranění všech tříd vrcholů

c) po odstranění jednoduchých vrcholů

d) po odstranění vrcholů vnitřních hran

e) po odstranění vrcholů na vnějších hranách

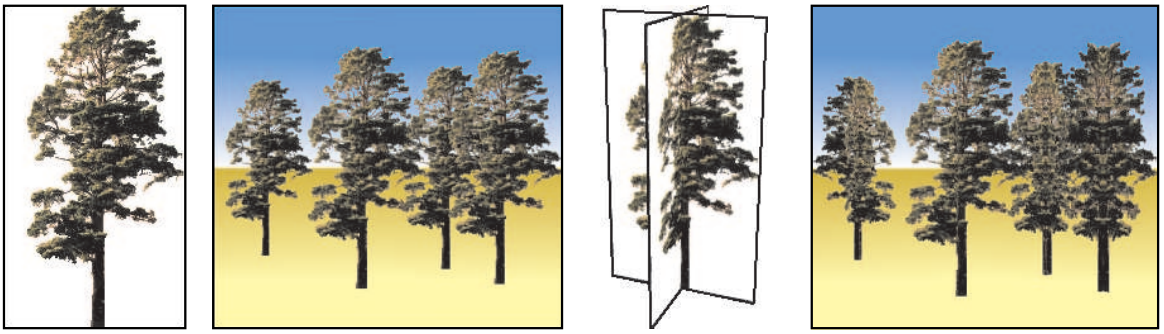
Obrázek 19.11: Příklad odstraňování různých tříd vrcholů. Druhá řada ukazuje dílčí kroky.

na čtverci vzdáleností původních bodů od upravené sítě, na celkovém počtu vrcholů a na vzdálenosti vrcholů sítě navzájem.

19.2.3 Zjednodušená reprezentace objektů pomocí obrázků a bodů

Přírodní objekty z reálného světa, jakými jsou např. květiny a stromy, se nejenom velmi obtížně modelují, ale především se kvůli vysokému počtu polygonů obtížně zobrazují ve větším množství (louka, les). Pro takové objekty je vhodné použít náhradní reprezentaci v podobě dvojrozměrného obrázku, který je aplikován jako textura na jednoduchý polygon, obvykle obdélník. Tento nosný polygon si můžeme představit jako skleněnou (tj. zcela průhlednou) desku, na níž je namalován obraz prostorového objektu. Pozadí obrázku s texturou tedy musí mít nastavenu plnou průhlednost. Metoda přináší vysoké zrychlení zobrazování zejména u přírodních scén. Jediný průhledný obdélník s obrázkem nahradí stovky malých polygonů složitého prostorového modelu (obr. 19.12). Pro takto vytvořené objekty se používá několik názvů – *impostor*, *billboard* a *sprite*. Liší se podle toho, zda jsou tvořeny jediným nosným obdélníkem, zda je obraz objektu neměnný či dynamicky aktualizovaný, zda nosný polygon nese obrazy jednoho nebo více objektů či zda je pevně umístěn do scény.

Impostor. Na nosném obdélníku se může zobrazovat i několik objektů současně. Pokud se pozorovatel k impostoru přiblíží, je impostor nahrazen skutečnými geometrickými objekty.



Obrázek 19.12: Obraz stromu použitý jako textura pro čtyři billboardy stromů (vlevo). Tentýž obraz mapovaný na dva navzájem kolmé obdélníky vytváří sprite, jehož čtyři různě natočené instance jsou vloženy do scény (vpravo).

Příkladem použití impostoru je v městské zástavbě obraz domů na konci ulice či mrakodrapů na obzoru v určitém směru od pozorovatele [Deco99]. Impostor zde zastupuje množinu vzdálených domů, jejichž přesnější modely však existují a použijí se jako vyšší stupeň LOD.

Impostory se ve většině aplikací nepřipravují předem – problémem je určení pozice pozorovatele, pro níž má být impostor předpočítán. Nahrazuje-li například impostor výhled dlouhou ulicí, jsou na něm zachyceny různé domy v částečných zákrytech. Když uživatel ulicí prochází, na základě zkušenosti z reálného života očekává, že se zákryty domů budou s ohledem na jeho polohu měnit. Statický impostor snižuje iluzi hloubky prostoru, působí jako plochá fotografie na pozadí. Proto se impostory vytvářejí dynamicky z obrazu prostorových objektů, které již byly zobrazeny v předchozích snímcích.

Impostory lze úspěšně využívat také při animacích objektů, dokonce i pro tak složitý objekt, jakým je lidská postava. V laboratoři VRLab v Lausanne ve Švýcarsku bylo v reálném čase úspěšně simulováno a zobrazováno chování skupiny čítající 120 osob [Aube00]. Také v tomto případě je impostor pro každou z postav vypočítáván dynamicky. Trojrozměrný model postavy je zobrazen pomocí paměti hloubky, avšak mimo část viditelnou na obrazovce (*off-screen buffer*). Tento obraz je uchován jako textura příslušného impostoru a používán v dalších snímcích až do té doby, než vlivem animace postavy dojde k výraznější změně její polohy. Tehdy se opět použije plný model transformovaný do aktuální animované pozice a z jeho obrazu vznikne nový impostor. Dynamické vytváření impostorů je v současné době jedinou rozumnou možností – předpočítání statických impostorů pro všechny možné animace a směry pohledu pozorovatele by znamenalo extrémní paměťové nároky.

Billboard. Je vhodný pro reprezentaci jednoho objektu, například stromu. Billboard je tvořen jedním polygonem, který je pevně umístěn do konkrétního místa ve scéně, avšak při zobrazování se vždy natočí tak, aby rovina polygonu byla pokud možno kolmá na pohled uživatele. Tím se

zamezí tomu, aby uživatel spatřil nosný polygon z boku, se silně zkreslenou texturou. Nejčastěji je billboard otáčen kolem svislé osy y, není to však pravidlem.

Billboard tedy zachycuje právě jeden objekt, nemění svoji polohu, ale mění svoji orientaci. Obraz na billboardu je statický, předem připravený. Pokud grafický procesor nepodporuje mipmapping (viz část 13.1.5), je možno při práci se stupni LOD připravit několik billboardů s různým rozlišením příslušného obrazu. Billboardy jsou často využívány v aplikacích virtuální reality. Jsou vhodné pro útvary symetrické podle svislé osy, jakými jsou stromy (obr. 19.12 vlevo).

Sprite. Tento druh obrazové náhrady prostorového útvaru je s oblibou používán v počítačových hrách. V základní variantě je tvořen jedním nosným polygonem s obrazem jednoho objektu. Na rozdíl od billboardu se jeho orientace při změně polohy pozorovatele nemění. Může se proto stát, že uživatel při procházení scénou spatří sprite z boku, promítnutý do úsečky. Proto se častěji používá rozšířená varianta, která obsahuje kombinaci několika navzájem se protínajících nosných polygonů. Polygony jsou různě natočeny a na každém z nich je nanesen obraz téhož prostorového objektu pořízený z odpovídajícího pohledu. Příkladem je strom na obrázku 19.12 vpravo, který je reprezentován dvěma navzájem kolmými svislými obdélníky. Takový sprite již může být uživatelem pozorován z různých stran a prostorový vjem zůstane zachován.

Sprite a billboard jsou pro svoji jednoduchost velmi oblíbené náhradní reprezentace komplexních útvarů. Obrazy pro nosné polygony jsou připraveny předem, takže výpočetní náročnost v době zobrazování je nízká a srovnatelná – u billboardu je nutno vypočítat a aplikovat správné natočení vůči pozorovateli, zobrazit jeden sprite pak znamená vykreslit několik nosných polygonů. Je-li ve scéně mnoho totožných objektů, poskytuje jejich reprezentace typu sprite vizuálně lepší výsledky než billboardy. Obrazy na billboardech totiž vypadají ze všech stran stejně, jak dokumentuje obrázek 19.12.

Při rychlém vykreslování rozsáhlých scén lze též využívat bodově reprezentované objekty. Základní techniky jejich zobrazování byly uvedeny v části 11.5.

Reprezentace složitých objektů a scén pomocí obrazů se objevují v nejrůznějších aplikacích a v různých kombinacích. Například v geografických systémech, které zachycují zemský povrch, je běžné (a především kvůli rozsahu dat i nutné), že různé oblasti jsou reprezentovány hierarchií družicových a leteckých snímků, která je dále kombinována s 2,5D a nakonec 3D modely budov a dalších objektů. Často je touto hierarchií *kvadrantový strom* (*quadtree*). Čtverec, reprezentující velkou část terénu, např. v řádu tisíců km^2, je pokryt jedinou texturou, vnitřně je však rozdělen na další čtyři čtverce. Tento postup lze rekurzívně opakovat až do dosažení libovolné velikosti detailu. V situaci, kdy pozorovatel klesá z atmosféry k povrchu, je hierarchie modelu procházena od kořene a podle vzdálenosti od pozorovatele jsou vybírány textury (fotografie terénu) v příslušných uzlech kvadrantového stromu. V hlubších patrech stromu jsou do terénu přidávány prostorové objekty.

Kapitola 20

Virtuální realita

Cílem systémů pro *virtuální realitu* (*VR, Virtual Reality*) je poskytnout uživateli či skupině spolupracujících uživatelů iluzi, že se nacházejí v umělém prostředí, nazývaném virtuální svět, virtuální scéna či virtuální prostředí. V dalším textu budeme pro virtuální realitu používat obecnou zkratku VR. Prostředí VR je nejčastěji vytvořeno v paměti počítače, může však existovat i jako kombinace skutečného světa a počítačem doplněných objektů. Dosažení pocitu přítomnosti uživatele ve virtuálním světě se dociluje ovlivněním lidských smyslů, nejčastěji zraku a sluchu, vzácněji pak hmatu a v simulátorech i rovnováhy. Svoji roli zde hraje i představivost a „ochota“ uživatele přijmout představu světa prezentovanou počítačem. Tato ochota nemusí být nutně vědomá – smysly uživatelů se často podvědomě brání uvěřit informacím, které jsou jim z počítače prezentovány, neboť oproti skutečnosti je tato prezentace ne zcela dokonalá, případně je v (částečném) rozporu s dosavadními zkušenostmi z reálného světa. Chování jednotlivých součástí virtuálního prostředí by proto mělo být v souladu s fyzikálními zákony. Je například žádoucí, aby se těžké virtuální předměty pohybovaly pomalu nebo aby se rychlost a orientace pohybu uživatele ve VR neměnily skokově.

Z hlediska počítačové grafiky jsou základem virtuální reality postupy, se kterými jsme se seznámili v předchozích kapitolách – tvorba prostorových modelů a scén, manipulace s nimi, pohyb v trojrozměrném prostoru, detekce kolizí a zobrazování v reálném čase. Tyto metody jsou umocněny použitím periferií, které zajišťují obrazovou, zvukovou a hmatovou interakci. Jde zejména o helmy se zabudovanými displeji, stereoskopické projekční plochy, snímače polohy v prostoru, hmatová zařízení, simulační kabiny, apod.

Virtuální realitu lze realizovat také bez speciálních zařízení, ovšem za cenu nižší kvality vjemu. I obrazovka osobního počítače se může stát „průzorem“ do virtuálního světa, v němž se pohybujeme pomocí klávesnice či myši.

Aplikací z oblasti VR nazveme takový program, u něhož převládají následující vlastnosti.

- *Reálný čas*
  Zobrazování a interakce s uživatelem se provádějí s takovou rychlostí, při níž se pohyb na obrazovce jeví jako plynulý. Za minimální rychlost se považuje 25 snímků za sekundu (*fps, frame per second*).
- *3D prostor*
  Scéna a objekty mají trojrozměrný charakter nebo alespoň vytvářejí jeho iluzi. To, co jsme v předchozích kapitolách nazývali scénou, je nazýváno virtuálním světem či prostředím.
- *Navigace*
  Uživatel neprohlíží scénu jen zvenčí, ale vstupuje do ní a prochází v ní po rozličných drahách (chodí, létá, skáče a mžikem se přesouvá – teleportuje se). Při pohybu může na uživatele působit gravitace a jsou vyhodnocovány kolize při nárazu do objektů.
- *Interaktivita*
  Scéna obsahuje interaktivní objekty – s některými uživatel přímo manipuluje, jiné jsou animovány podle předem daných scénářů či s ohledem na aktivitu uživatele.
- *Multimedia*
  Je využíván zvuk, ať již doplňující na pozadí celkovou atmosféru scény či přinášející konkrétní informaci o vlastnostech objektů, s nimiž uživatel interaguje (cinkání, skřípání, zvonění apod.). Virtuální prostředí zahrnuje i použití videosekvencí jako textur.

Z výše uvedených požadavků je pro systémy VR nejdůležitější ten první. Plynulé zobrazování scény má přednost před zajištěním realistického vzhledu scény. Tuto skutečnost nám pomůže vysvětlit analogie s televizním vysíláním – lidé spíše přijmou horší kvalitu obrazu nežli trhavý a přerušovaný děj.

20.1 Druhy aplikací VR

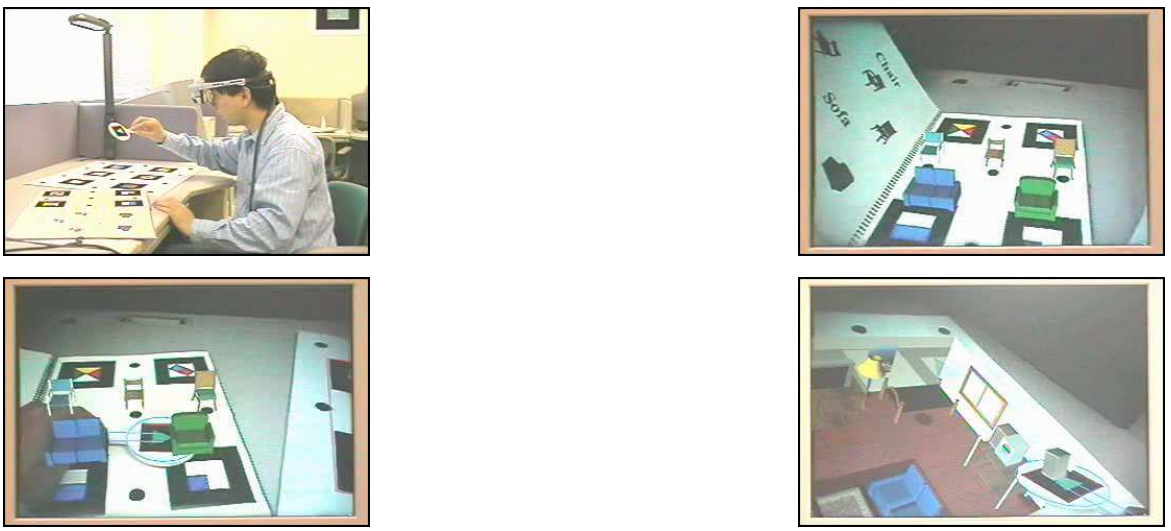
Rozlišujeme několik typů aplikací, které používají společný název VR.

Pohlcující virtuální realita (*immersive VR*). Je vždy spjata se speciálními technickými zařízeními, která mají v co největší míře oprostit (odříznout) uživatele od vjemů skutečného světa a dodat mu zdání, že je zcela ponořen do světa virtuálního. Mezi typická periferní zařízení patří helma se stereoskopickými brýlemi a sluchátky, snímače detekující prostorovou polohu uživatele nebo datová rukavice nahrazující jednodušší vstupní zařízení. Často je uživatel umístěn v simulátoru – kabině, která se naklání a vyvolává pocit pádu či odstředivé síly. Mezi zajímavé technické součásti patří dotyková (*tactile, force feedback*) zařízení, která jsou schopna měnit

odpor či tlak vyvíjený proti ruce uživatele, který tak fyzicky pociťuje mechanické vlastnosti virtuálního materiálu.

Pohlcující VR nachází uplatnění nejen v herních centrech a rozličných trenažérech, ale i jako prostředek terapeutický. Známé je například úspěšné léčení různých fóbií – z výšek, volného prostoru, uzavřeného prostoru nebo i jen z pavouků.

Rozšiřující virtuální realita (*augmented VR*). Informace ze skutečného, okolního světa jsou kombinovány s doplněnými prvky virtuální reality. Součástí systému bývá kamera, jejíž pozice a orientace je synchronizována s pohybem uživatele. Uživatelovy aktivity jsou snímány různými senzory.



Obrázek 20.1: Ukázka rozšiřující VR. Uživatel má na hlavě kameru a polopropustné brýle (vlevo nahoře). Systém rozpoznává ikony obrázků na papíře a do brýlí promítá na jejich místa 3D modely nábytku (vpravo nahoře), občas dojde ke kolizi mezi skutečným světem (obrazem uživatelovy ruky) a virtuálním (vlevo dole). K editaci interiéru virtuálního pokoje (vpravo dole) slouží obyčejná papírová pálka (*magic paddle*), podle níž byl tento systém nazván [Kawa01].

Příkladem je použití ve vojenských letadlech a vozidlech, kde je venkovní obraz snímaný kamerou přenášen na obrazovku a okamžitě doplněn výraznými symboly pro nepřátelské a spřátelené objekty. Z civilní oblasti uveďme použití rozšiřující VR při instalaci elektrických rozvodů o celkové délce mnoha desítek kilometrů v letadlech Boeing. Dělníci mají nasazeny brýle, skrze které normálně vidí, ale do nichž jsou jim současně promítány doplňující značky,

jednoznačně určující místa propojení či rozdělení kabelů podle toho, kam se dělník brýlemi dívá. V oblasti kultury se objevují projekty, při nichž uživatel prochází historicky zajímavými místy (Pompeje [Scag01], hrad v Heidelbergu [Kret01]) a skrze speciální brýle vidí počítačem doplněné stavby, předměty, ba dokonce i postavy z minulosti, které mu předvádějí scény z dávné historie a se kterými může rozmlouvat.

Jednoduchá virtuální realita (*desktop VR, low-end VR*). Do této skupiny řadíme aplikace, které nevyužívají speciální technická zařízení. K iluzi práce nebo pohybu ve virtuálním světě poslouží obyčejná obrazovka, představující „kukátko“ do jiného prostoru. Stereofonní reproduktory dodají zdání prostorového zvuku, myš nahradí složité ukazovací a uchopovací zařízení. Aplikace jednoduché VR lze provozovat na většině běžných počítačů, a proto je jejich škála skutečně rozmanitá – od zábavy (počítačové hry) přes vzdělávání (např. simulace pohybu planet ve vesmíru) až po profesionální aplikace (výzkum, trénink, simulace).

Uvedené rozdělení aplikací je založeno na technických prvcích používaných určitým systémem VR. Jiným hlediskem je množství skutečných a umělých prvků prezentovaných uživateli. Klasifikace je pak dána postupným přechodem od skutečného světa do plně počítačového prostředí. Jedno z takových členění, nazývané *kontinuum RV* (*reality-virtuality continuum*) používá stupnici: reálné prostředí ⇒ rozšířená realita ⇒ rozšířená virtualita ⇒ virtuální prostředí. Rozšířenou *virtualitou* je chápán počítačový svět obohacený o prvky z reálného světa, např. o fotografie. Aplikace kombinující prvky rozšířené reality a rozšířené virtuality se nazývá *smíšená realita* (*mixed reality*). Toto členění určitým způsobem koresponduje s filozofickými úvahami o vnímání světa. V dalším textu se omezíme na technické pojetí VR a jejích součástí.

Na aplikace VR můžeme nahlížet i z hlediska počtu uživatelů současně přítomných ve virtuálním prostředí. Jednouživatelské aplikace lze snadno zpřístupnit více uživatelům, kteří společně sledují zobrazovací zařízení nebo sdílejí stejný prostor, např. kabinu simulátoru. Ve většině případů jsou však jen v pozici diváků a interaktivní akce ve VR provádí pouze jeden z nich. Teprve pokročilé aplikace VR, které dovolují všem uživatelům aktivně se zúčastnit dění ve virtuálním prostředí, se správně nazývají *víceuživatelská virtuální realita* (*multi-user VR*). Uživatelé v takových systémech mohou být fyzicky vzdáleni, iluze společného pobytu ve VR je zajištěna propojením jejich počítačů do sítě. VR se tak stává prostředkem pro komunikaci mezi lidmi. Je možné hrát hry ve virtuálním světě se spoluhráči sedícími v danou chvíli na opačných koncích planety, stejně tak je možno posuzovat v distribuovaném lékařském týmu např. nádor reprezentovaný virtuálním modelem získaným z naměřených dat pacienta. Systémy pro víceuživatelskou VR mohou být postaveny nad libovolnou z výše uvedených aplikací (pohlcující, rozšiřující, jednoduchou). Jsou nadstavbou, která především zajišťuje synchronizaci aktivit uživatelů. S těmito aplikacemi se můžeme setkat i pod názvem *distribuovaná virtuální realita* (*distributed VR*).

Z uvedených příkladů aplikací VR zřetelně vystupují do popředí počítačové hry. Je pravdou, že mnoho vysoce efektivních algoritmů se zrodilo právě při programování her – v tomto smyslu mají počítačové hry průkopnickou roli, a to bez ohledu na jejich často nízkou společenskou úroveň a návykovost her.

20.2 Speciální postupy ve virtuální realitě

Systémy pro VR využívají mnoha metod popsaných v předchozích kapitolách, především technik zaměřených na dosažení co největší rychlosti zobrazování. Velmi časté je používání objektů s několika stupni detailu (viz část 19.2). Kromě těchto běžně používaných postupů se aplikace VR vyznačují i speciálními metodami, které popíšeme v následujících částech. Některé techniky slouží k urychlení výpočtů při zobrazování, jiné k prohloubení iluze existence virtuálního prostředí.

20.2.1 Pozadí scény

K doplnění iluze pobytu ve virtuálním světě je vhodné zahrnout do scény podlahu (*ground*) a pozadí (*background*). Bez těchto prvků je virtuální svět jen směsicí objektů vznášejících se v blíže neurčeném, nekonečném vesmíru. Podlaha umožní nalézt uživateli „pevné místo“ a usnadní jinak poměrně obtížnou orientaci v trojrozměrném prostředí.

Praktickým zjednodušením scény je vykreslit vzdálené objekty jako obraz v pozadí. Tento obraz namapujeme jako texturu na vnitřní stěny nějakého jednoduchého geometrického útvaru (kvádru, válce či koule), do něhož umístíme celou virtuální scénu, čímž pozadí obklopí celý prostor. Uživatel nesmí k pozadí nikdy dojít, řešením je průběžné posouvání objektu s namapovaným obrazem pozadí tak, aby jeho střed byl shodný s pozicí kamery.

Nenáročnou a přitom dostatečně působivou technikou je vybarvení pozadí postupně se měnícími barvami. Horní část pozadí může být světle modrá až bílá (obloha), střední zelená (les, tráva) a dolní hnědá či šedivá (půda, chodník). V tomto případě se virtuální svět uzavře do pomyslné koule, na níž jsou barevné pruhy ve směru rovnoběžek. Barevné pruhy se definují jako posloupnost barev doplněná informacemi o případné interpolaci mezi nimi.

20.2.2 Avatar a navigace

V systémech VR je uživatel reprezentován virtuální postavou, nazývanou *avatar*. Tato postava je jeho dvojníkem, pomyslnou bytostí, která se pohybuje ve virtuálním světě a její chování představuje akce uživatele. To, co uživatel vidí na obrazovce, je vlastně to, co vidí avatar ve virtuálním světě.

Zavedení avatara má praktický význam. Pomyslný dvojník má určité rozměry, které mu například brání v průchodu úzkým či nízkým prostorem. V jednouživatelských systémech nebývá postava avatara vidět – když uživatel změní pohled směrem dolů, málokdy spatří avatarovy boty, neboť pro systém VR není jejich obraz důležitý. Výjimkou jsou aplikace, v nichž uživatel ovládá program či interaguje s virtuálním prostředím pomocí tzv. datové rukavice (*data glove*) či hmatového zařízení. Tehdy je na obrazovce znázorněna ruka avatara jako forma vizuální zpětné vazby.

V systémech pro víceuživatelskou VR se avataři setkávají, a proto je třeba jim dodat skutečný vzhled. Pro potřeby sociálního chování se avatar doplňuje řadou připravených animací, jimiž dává najevo své pocity (souhlas, nesouhlas, nezájem, nadšení, upoutání pozornosti). Řeč těla vyjádřená pomocí tzv. gest se programuje lépe než výrazy obličeje. Je také zřetelná z větší vzdálenosti.

Aby byla iluze virtuálního světa co nejpřesvědčivější, jsou na avatara uplatňovány některé fyzikální zákony. Projevují se především při pohybu, tzv. *navigaci*. V aplikacích VR se rozlišují tři základní druhy navigace.

- *Chůze* (*walk*)
  Na avatara působí zemská přitažlivost. Směr dolů bývá shodný se směrem záporné poloosy y. V tomto směru systém vyhodnocuje přítomnost objektů a přesouvá na ně avatara tak, aby vždy stál na podlaze, schodišti či jakémkoliv odpovídajícím virtuálním předmětu. Příjemným důsledkem tohoto postupu je automatické sledování terénu. Avatar se může procházet po libovolně zvlněném povrchu či po šikmé ploše a vždy je jeho vertikální poloha aktualizována podle příslušné podložky. Uživatel se tedy nemusí starat o zvedání nohou svého avatara, když kráčí po schodišti.

  Součástí režimu chůze je testování kolizí s překážkami. Virtuální předměty jsou uvažovány jako neprůchozí, není-li určeno jinak. Pokud se avatar pokusí takovým objektem projít, aplikace VR detekuje kolizi (viz část 14.3) a další postup avatara je zastaven. Vzhledem k náročnosti úlohy detekce kolizí bývá vyhodnocována pouze vzájemná poloha avatara vůči prvkům virtuálního prostředí. Jednoduché aplikace VR neuvažují dynamiku nárazu a avatara pouze zastaví. V pokročilých simulátorech je naopak reakce na náraz vyhodnocena důsledně včetně výpočtu směru pohybu avatara a tělesa po nárazu. Objekty mezi sebou obecně testovány nejsou, existuje však možnost u vybraných (animovaných) objektů detekci kolizí zapnout. Simulace fyzikálních zákonů – přitažlivost a hmotný charakter objektů – zvyšují výpočetní náročnost.

- *Létání* (*fly*)
  Při navigaci pomocí létání je vypnuta funkce přitažlivosti. Avatar se volně pohybuje prostorem, typicky ve směru svého pohledu. Detekce kolizí je zapnuta.

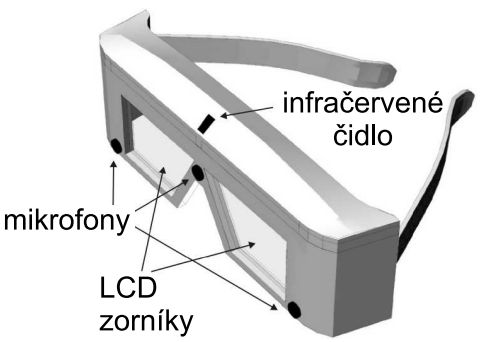
- *Zkoumání* (*examine*)
  V režimu zkoumání může avatar procházet virtuálními předměty, není prováděna detekce kolizí. Je též vypnuta přitažlivost. Tento režim je vhodný k prohlížení jednotlivých virtuálních objektů ze všech stran. Z praktického hlediska bývá pohled avatara fixován na střed vybraného objektu a pohyb kamery je povolen po povrchu pomyslné koule obklopující objekt. Tato interaktivní technika manipulace s kamerou se nazývá *virtuální kulový ovladač* (*virtual trackball*).

20.2.3 Stereoskopické pohledy

Pocit, že se člověk nachází v trojrozměrném prostoru, je nejvíce ovlivněn zrakem. Zornice lidských očí jsou od sebe vzdáleny přibližně 7 cm a oči proto do mozku dodávají dva mírně rozdílné pohledy. Ty jsou skládány z jemných rozdílů a mezi nimi je odvozena hloubka (vzdálenost) objektů. Přesvědčivost systémů pro VR je často založena právě na využití tzv. *stereoskopických* pohledů. Každý stereoskopický pohled je tvořen dvojicí obrazů vzniklých perspektivním promítáním ze dvou bodů ležících vedle sebe ve vzdálenosti cca 7 cm. Znamená to, že pokud chceme zachovat celkovou rychlost zobrazování 25 snímků za sekundu, musíme připravit za jednu sekundu 50 obrazů. Hlavním problémem u stereoskopických pohledů však není zdvojnásobení potřebné snímkové rychlosti, ale způsob, jak výsledné obrazy „dopravit“ do uživatelových očí. Buď jsou oba obrazy prezentovány odděleně (v čase nebo prostoru) nebo jsou zobrazeny ve stejném okamžiku na stejném zobrazovacím zařízení a je nutno je nějakým způsobem oddělit.

Do první kategorie patří použití helem pro VR (*HMD, Head Mounted Display*). Obraz pro levé i pravé oko je samostatně přenesen do helmy, která má před každým okem jednu malou obrazovku. Obrazy jsou tedy odděleny v prostoru a mohou být zobrazeny ve stejném čase.

Jiný způsob je založen na rozdělení obrazů v čase, resp. na rychlém, pravidelném střídání obrazů na stejném zobrazovacím zařízení. Tím může být jak obrazovka běžného počítače, tak větší projekční plátno. Podmínkou je použití tzv. *zatmívacích brýlí* (*shutter glasses, flicker lens*). Brýle mají polopropustné zorníky vyrobené na bázi tekutých krystalů (viz obr. 20.2). Jsou synchronizovány se zobrazovacím zařízením (např. pomocí infračervených paprsků) a střídavě mění optickou propustnost levého a pravého zorníku tak, jak je vykreslován obraz pro levé, resp. pravé oko. V okamžiku, kdy je zobrazen obraz pro levé oko, dojde k zneprůhlednění pravého zorníku a naopak. Některé brýle jsou navíc vybaveny i čidly pro snímání polohy na bázi ultrazvuku.



Obrázek 20.2: Zatmívací brýle

Do druhé kategorie patří systémy, které prezentují oba obrazy ve stejném čase na stejné zobrazovací ploše. K optickému oddělení obou obrazů se používají filtry. Nejjednodušší je použití barevných filtrů. Obraz pro levé oko vykreslíme např. v odstínech červené, obraz pro pravé oko v odstínech modré barvy. Oba obrazy můžeme smíchat a prezentovat najednou (takto barevně smíchaný stereoskopický obraz se nazývá *anaglyf*). Brýle mají barevné filtry ve stejných odstínech. Levý, červený zorník nepropustí modré odstíny a do levého oka se tak dostane pouze obraz jemu určený. Nevýhodou tohoto jednoduchého a tedy laciného řešení je omezení na odstíny dvou barev použité pro rozlišení levého a pravého obrazu, byť scéna v pozadí, která je vidět oběma očima shodně, může být vykreslena v libovolných barvách. Dokonalého oddělení dvou smíchaných obrazů lze docílit pomocí polarizovaného světla. Obraz pro levé oko je promítnut na projekční plátno s vodorovnou polarizací světelných paprsků, obraz pro pravé oko se svislou. Uživatelovy brýle mají v zornících polarizační filtry, které propustí jen shodně orientované světelné vlnění. Tohoto principu se používá také v síti stereoskopických kin IMAX 3D.

Se stereoskopickými pohledy na obrazovce počítače je spojen problém zabezpečení vhodné polohy uživatele. Systém vypočítává pohledy pro typicky umístěného pozorovatele, tedy sedícího přibližně 40 - 60 cm před obrazovkou. Případní další pozorovatelé v místnosti se dívají z jiné vzdálenosti a pod jinými úhly a kvalita jejich vjemu je výrazně nižší.

20.3 Formáty VRML a X3D

Pro popis virtuálního prostředí a jeho interaktivních vlastností byla vyvinuta mezinárodní norma. V roce 1997 byla publikována pod názvem VRML (*Virtual Reality Modeling Language*) [ISO97]. Norma definuje jazyk pro popis virtuálních světů a současně formát souborů, v nichž se tyto světy ukládají a přenášejí po síti. Zde uvedeme pouze základní charakteristiku VRML, podrobný popis jazyka včetně příkladů jeho použití je uveden v české publikaci [Žára99].

- Scéna je organizována do stromové struktury, která umožňuje skládat transformace a snadno vytvářet kopie dříve definovaných objektů (podstromů).
- Tělesa a další objekty jsou popsány pomocí hraniční reprezentace a jsou tvořeny rovinnými ploškami. Na plošky lze mapovat textury statické (ve formátech JPEG, GIF a PNG) nebo pohyblivé (MPEG).
- Scéna obsahuje většinu běžně používaných prvků jako jsou zdroje světla (bodové, směrové a reflektor – viz část 10.6), předdefinované polohy a charakteristiky kamer či barevná mlha.
- Přestože VRML není programovací jazyk (neobsahuje cykly, výrazy a další programovací konstrukce), je možno modifikovat existující struktury a definovat nad nimi vlastní,

parametrizovatelné objekty. Funkční vlastnosti nových objektů lze popsat jazyky Java a JavaScript.

- Jazyk je zaměřen na prezentaci VR na webu. Scénu lze kombinovat ze souborů lokálních i přístupných přes web. Po síti tak lze sdílet 3D objekty, textury a zvuky a automaticky je vkládat do virtuálního prostředí.
- Jazyk obsahuje prostředky pro popis animace objektů a interakce s uživatelem. Animace jsou založeny na lineární interpolaci klíčových hodnot. Pro interakce slouží různé senzory, reagující nejen na ukazovací vstupní zařízení (myš), ale též na aktivity avatara (jeho pohyb, nárazy, směr pohledu). Interakce a další dynamické akce využívají pro synchronizaci principu tzv. událostí.
- Scéna popsaná pomocí VRML se ukládá v textovém formátu, soubory mají příponu WRL. Před uložením do souboru je povoleno komprimovat data programem `gzip`.

X3D

Norma VRML a především její orientace na web přinesla zájem o VR v mnoha aplikačních oblastech. Původní rozsah schopností jazyka a množina podporovaných geometrických prvků postupně přestaly dostačovat novým a náročnějším požadavkům. Ty se snaží uspokojit nový formát s názvem X3D (*eXtensible 3D*) [ISO03b]. Mezi jeho hlavní vlastnosti patří:

- Využití křivek a ploch typu NURBS (viz část 5.10). Animace pomocí křivek NURBS, modelování pomocí FFD (viz část 6.4.2).
- Obohacení o geografická data a postupy. To zahrnuje jak zpracování objektů v jiné, než ortogonální souřadnicové soustavě (např. geocentrické), tak rozšíření přesnosti zápisu souřadnic.
- Na rozdíl od VRML je množina prvků (objektů) jazyka členěna do logických tříd, zvaných *komponenty*. Ty mají určitou vazbu na náročnost zpracování a zobrazení scén X3D. Programy, které čtou a interpretují scény X3D, mohou být koncipovány modulárně v souladu s komponentami X3D. Jednodušší komponenty lze využívat v mobilních zařízeních (např. telefonech), pokročilé komponenty jsou vhodné pro velké systémy VR. Napříč komponentami jsou definovány tzv. *profily* určující číselné meze prvků scény, např. maximální rozměr textur nebo počet vrcholů v jedné trojúhelníkové síti.
- Soubory X3D mohou být kódovány v podobě textové, binární nebo ve formátu XML. Textová verze je nadmnožinou VRML, je tedy zabezpečena zpětná kompatibilita.

20.4 Prostorový zvuk

Další z cest, jak zvýšit dojem prostorovosti virtuální reality, je modelovat prostorový zvuk virtuální scény. Je to souhrnný zvuk pocházející z různých virtuálních zdrojů ve scéně a zachycený virtuálním přijímačem v místě avatara, na který během jeho přenosu působilo prostředí scény (odrazy, ohyby, zkreslení atd.).

Počítačová akustika (*computational acoustics*), která se těmito problémy zabývá, sdílí s počítačovou grafikou některá paragidmata popisu scén, a proto je praktické zobrazovací a zvukový systém sloučit. Jednou z vyplývajících výhod je například jediný společný popis scén charakterizující jak světelné, tak i zvukové vlastnosti objektů ve scéně. Do jisté míry je možno sloučit též i datové struktury urychlující výpočet scén, i když toto sloučení je omezeno rozdílnými vlastnostmi obou vlnění.

V této části se seznámíme s principy vnímání prostorového zvuku, ukážeme několik metod, jak ve virtuálním prostředí šíření zvuku simulovat, a pak popíšeme, jak je možno prostorový zvuk za pomoci současných technických prostředků reprodukovat.

20.4.1 Vnímání zvuku

Pod pojmem prostorový zvuk si pravděpodobně představíme běžnou stereo reprodukci pomocí dvou reproduktorů či klasických sluchátek. Při pozornějším poslechu stereofonní nahrávky však zjistíme, že se prostorový vjem omezuje na spojnici zdrojů zvuku a tak například hudba poslouchaná ze sluchátek zní stále jakoby uvnitř hlavy. Poslech prostorového zvuku by však měl vyvolávat dojem, že se zdroj zvuku nachází i někde mimo posluchače, že přichází z libovolného konkrétního místa v okolním prostoru.

K tomu, abychom byli schopni vytvářet věrohodný zvukový výstup, je třeba znát způsob, jakým lidský sluch získává informace o směru, ze kterého zvuk přichází. Zvuk je jako mechanické vlnění převáděn na nervové vzruchy ve vnitřním uchu. Až sem se šíří mechanicky a je ovlivněn každou částí lidského těla, se kterou se na své cestě do vnitřního ucha setká. Právě změny způsobené tělem, hlavou a uchem spolu s rozdíly a zpožděním zvuku v každém z uší umožňují mozku zjistit směr, ze kterého zvuk přichází. Následující jevy umožňují určit směr příchodu zvuku v horizontální rovině (*lateralizaci*):

Meziušní zpoždění (*ITD, interaural time difference*). Zvuk přicházející z jednoho místa v prostoru urazí k jednomu z uší delší cestu než ke druhému a tudíž není zachycen oběma ušima ve stejný okamžik.

Meziušní rozdíl intenzity (*IID, interaural intensity difference*), též *zvukový stín*. Zvuk přicházející z jedné strany do protilehlého ucha je zastíněn hlavou a proto je jedním uchem vnímán jako hlasitější než uchem druhým.

Tyto dva jevy se v celém frekvenčním spektru doplňují: ITD funguje, dokud je vlnová délka příchozího zvuku větší než velikost hlavy, a není účinný ve vyšších frekvencích, u IID je to právě naopak. Pro určení směru příchodu zvuku též ve vertikální rovině (*elevaci*) využívá člověk zejména těchto faktorů:

Frekvenční charakteristika ušního boltce. Svým tvarem pracuje ušní boltec jako složitý frekvenční filtr, jehož vlastnosti jsou závislé na směru příchozího zvuku. Sluchový systém na základě porovnání přijímaného zvuku a předchozí zkušenosti je schopen extrahovat elevaci příchozího zvuku.

Odraz zvuku od ramen a hrudi. Lidské tělo je schopné odrážet zvuky o frekvencích přibližně 1 až 3 kHz. Určení elevace je tedy ještě zpřesněno detekcí takových odrazů od ramen a hrudi.

Následující principy přinášejí posluchači informaci o uspořádání prostoru kolem něj:

Identifikace brzkých ozvěn. Jsou-li v blízkosti posluchače rozmístěny větší překážky, je pravděpodobné, že se od nich bude zvuk odrážet. Posluchač pak zachytí kromě základního zvuku vyslaného ze zdroje ještě jeho ozvěnu. Sluchový systém člověka je schopen identifikovat jednotlivé ozvěny během první desetiny vteřiny poté, co je zachycen základní zvuk, a podle směru příchodu a frekvenčního zkreslení těchto ozvěn určit rozmístění a charakter blízkých překážek.

Délka dozvuku. Dozvuk je vlastností každého uzavřeného akustického prostoru. Je to soubor všech ozvěn, které vzniknou po vyslání impulsu ze zvukového zdroje. Čím větší je prostor a čím méně jeho stěny pohlcují zvuk, tím je jeho dozvuk delší. Podle jeho délky a barvy jsou lidé schopni přibližně určit velikost a povahu prostředí, v němž se nacházejí.

Stejně jako v případě prostorové grafiky, kdy nejprve vytváříme prostorový model a poté ho zobrazujeme, je možno rozdělit proces tvorby prostorového zvuku na dva kroky. V první fázi se pomocí simulace v počítačovém modelu určí parametry zvuku např. jeho směr, intenzita nebo zpoždění. Ve druhé fázi se pak zvuk generuje.

20.4.2 Simulace zvukového pole

Zvuk se šíří prostředím ve formě zvukového vlnění, odráží se a láme na rozhraní dvou prostředí. Zdálo by se, že vzhledem k vlnové podstatě světla bude možno použít některou z vizualizačních metod pro simulaci šíření světla ve virtuálním prostoru. Všechny vizualizační postupy však používají řadu zjednodušení vycházejících z vlastností šíření světla. Protože se zvuk od světla z hlediska virtuální reality významně liší, nelze přímo tyto metody pro simulaci šíření zvuku použít.

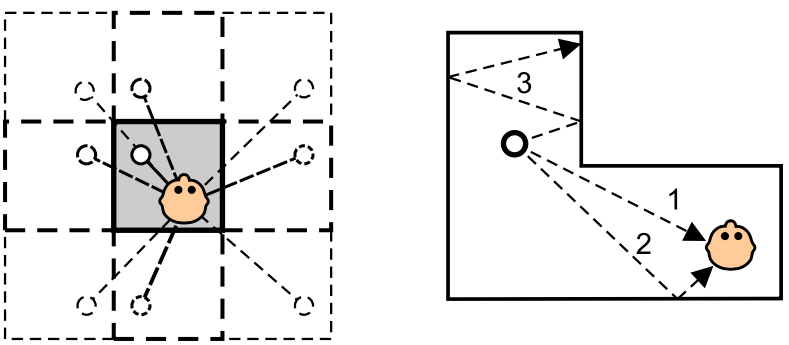
- Zvuk se odráží od všech povrchů v prostředí (jako kdyby byly difúzními zrcadly).
- Zvuk se šíří všemi látkami (jako kdyby byly „průsvitné“).
- Vlnová délka zvuku je srovnatelná s velikostí předmětů ve virtuálním světě, dochází tedy k ohybu zvuku.
- Rychlost zvuku není řádově vyšší než rychlosti pohybu různých objektů ve virtuálním prostředí a proto je třeba uvažovat Dopplerův jevu.

Míra projevu většiny výše uvedených vlastností je navíc závislá na frekvenci zvuku. Tato závislost je patrná nejvíce u ohybu zvuku. Hodnota frekvence (a tedy i vlnová délka) slyšitelného zvuku sahá přes čtyři řády. Zatímco u basových tónů je vlnová délka v řádu metrů, u vysokých frekvencí se měří na centimetry. Zvukové zdroje vysokých frekvencí lze zastínit i drobnými předměty, zatímco hluboké tóny nezastaví ani větší kusy nábytku.

Metody modelování je možno rozdělit na několik základních skupin: geometrické, fyzikální a statistické.

Geometrické metody

Tato skupina metod předpokládá, že je možno popsat šíření zvuku pomocí geometrické akustiky. V ní je postupující zvukové vlnění reprezentováno nezávislými paprsky kolmými na vlnoplochu modelovaného vlnění. Sledováním těchto paprsků lze získat informaci o celkovém zvukovém poli. Cílem těchto metod je sestavit výčet všech zvukových cest (posloupností odrazů jednotlivých paprsků), po kterých je dopravována zvuková energie ze zdroje k posluchači, a tak modelovat celkový dozvuk (odezvu) prostoru.

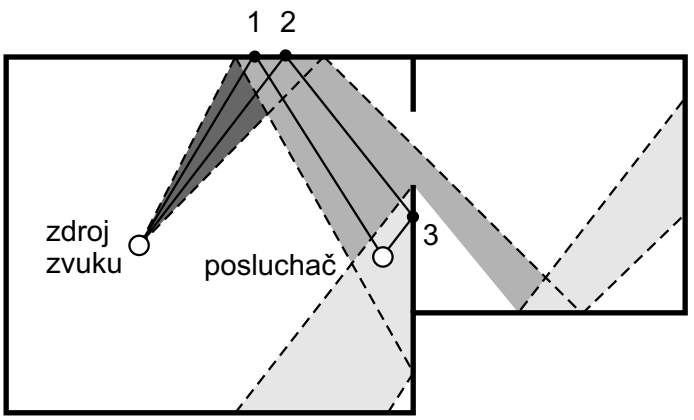


Obrázek 20.3: Metoda obrazových zdrojů (vlevo) – přerušovanými čárami jsou znázorněny zrcadlové odrazy původního prostoru a zdroje zvuku. Metoda sledování částic (vpravo) – první dvě zvukové cesty dorazí k posluchači.

Metoda obrazových zdrojů (*image source*) je založena na předpokladu, že zrcadlové odrazy zvuku ovlivňují výsledný sluchový vjem nejvíce. Původní scéna je doplněna o další prostory vzniklé zrcadlením původního prostoru podle jeho stěn. Součástí těchto fiktivních prostorů jsou i fiktivní zdroje zvuku (viz obr. 20.3 vlevo). V takto upravené scéně se potom pracuje pouze s přímou viditelností zdrojů zvuku (původních i nově vytvořených). Intenzita zdrojů vzniklých zrcadlením je snížena tak, aby odpovídala útlumu způsobenému odrazem od stěn. Složitost scény roste geometricky s počtem odrazů, které bereme v úvahu. U složitějších scén je možné tuto složitost snížit odstraněním zdrojů zvuku, které nemohou posluchače ovlivnit.

Metoda sledování částic (*particle tracing*) – nazývaná někdy též *sledování akustických paprsků* (*acoustic ray tracing*) – spočívá ve sledování částic, které jsou vypuštěny ze zdroje zvuku a šíří se přímočaře. Částice se pohybují scénou, jejich intenzita klesá se vzdáleností od zdroje a s počtem odrazů. Celková intenzita v místě pozorovatele se počítá jako součet intenzity všech částic, které dosáhly jistého okolí pozorovatele (viz obr. 20.3 vpravo). Tato metoda umožňuje vzít v úvahu i vzájemný pohyb zdroje a pozorovatele. V tom případě s sebou částice musí nést i informaci o rychlosti a směru, ze kterého byly vyzářeny. Z informací o rychlosti a směru částic a pozorovatele můžeme určit hodnotu Dopplerova posuvu frekvence zvuku. Tato metoda, stejně jako metoda z předchozího odstavce, však nepočítá s ohybem zvuku. Není proto vhodná pro simulaci šíření zvuku nízkých frekvencí.

Nevýhodou metody sledování částic je její malá robustnost. Při nevhodně zvolené výchozí množině částic může dojít k podvzorkování prostoru a nemusejí být proto nalezeny všechny důležité zvukové cesty. Je-li paprsek nahrazen *svazkem paprsků* (*beam*), je možno efektivněji procházet spojité úseky prostoru.



Obrázek 20.4: Metoda sledování svazku paprsků. Šedivé plochy představují výchozí a odražené svazky, plné čáry pak nalezené zvukové cesty, procházející místy odrazů 1, 2 a 3.

Na obrázku 20.4 je naznačeno, jak takové procházení probíhá a jak jsou nacházeny jednotlivé zvukové cesty. Tvar svazku je po každém dalším odrazu (a dělení na překážkách) složitější, a proto není možno použít tuto metodu pro modelování odrazů vysokých řádů. Často je tato metoda užívána pouze pro vyšetření základní slyšitelnosti zdrojů a doplněna některou ze statistických metod.

Výhodou této metody je možnost rozšíření modelu šíření zvuku též o schopnost modelovat ohyb zvukového vlnění kolem konkávních rohů ve scéně zavedením vztahů z jednotné teorie ohybu (*uniform theory of diffraction*), podle které se rohy stávají druhotnými zdroji zvuku. Více například v [Funk02].

Fyzikální metody

Fyzikální metody jsou založené na aproximaci řešení akustické vlnové rovnice[1], jejíž analytické řešení pro obecnou scénu neexistuje [Kutt00]. Je však možné použít klasické metody konečných prvků, které převádějí vlnovou rovnici na diferenční schéma. Výpočet pak probíhá v diskretizovaném prostoru scény. Tato metoda je výpočetně náročná, a proto se pro interaktivní simulace nehodí. Její použití navíc předpokládá statickou scénu.

Zajímavou a relativně snadno implementovatelnou alternativou jsou metody *vlnovodných mřížek* (*waveguide meshes*), které modelují akustické prostředí jako prostorovou mřížku uzlů propojených pružnými vazbami (o určené přenosové charakteristice), po kterých se zvukové vlnění šíří ze zdroje do okolí. Překážky ve scéně je možno modelovat vhodným nastavením přenosových charakteristik mezi příslušnými uzly sítě. Hustota sítě by měla odpovídat frekvenci zvuku (resp. jeho vlnové délce), jehož šíření simulujeme. Výhodou této metody je její schopnost modelovat ohyb zvuku.

Statistické metody

Na rozdíl od geometrických metod nepracuje statistická akustika s jednotlivými zvukovými cestami, ale zabývá se celkovým rozložením akustické energie v prostoru a čase. Statistické metody jsou vhodné zejména pro určení *délky dozvuku* a jeho *barvy* (*timbre*).

Zatímco předchozí metody popisující vlastnosti zvuku musejí být aplikovány před vlastní simulací, délku dozvuku lze určit v reálném čase podle Sabinova vztahu (20.1). Délka dozvuku je nezávislá na poloze pozorovatele a zdroje zvuku, takže je možné ji považovat za charakteristiku prostoru. Pro její výpočet se používá vzorec

[1] Vlnová rovnice je diferenciální rovnice, která popisuje harmonické vlnění procházející určitým prostředím

$$t = \frac{0.164.V}{\alpha S}, \tag{20.1}$$

kde S je celková plocha odrážejícího povrchu, V je objem prostoru a α je koeficient absorbce povrchu. Pokud není α pro všechny povrchy S_i konstantní, pak tuto hodnotu vypočítáme jako

$$\alpha = \frac{\alpha_1 S_1 + \alpha_2 S_2 + ... + \alpha_n S_n}{S}.$$

V případě, že se hodnota α blíží jedné, a povrch tedy zcela absorbuje zvuk, získali bychom ze vztahu (20.1) hodnotu $t = 0.164V/S$ místo očekávané nuly. V takovém případě se v tomto vztahu používá modifikovaná hodnota:

$$\alpha' = -ln(1 - \alpha).$$

20.4.3 Výstup prostorového zvuku

Metody používané pro výstup prostorového zvuku se liší podle použitého výstupního zařízení. Dnes se nejčastěji používají sluchátka nebo reproduktorové soustavy. Pro výstup do sluchátek se nejčastěji používá digitálních filtrů simulujících průchod zvuku lidským uchem (*HRTF, head related transfer function*). Pro výstup pomocí sady reproduktorů obklopujících posluchače se využívá skládání (superpozice) zvukových signálů z několika reproduktorů.

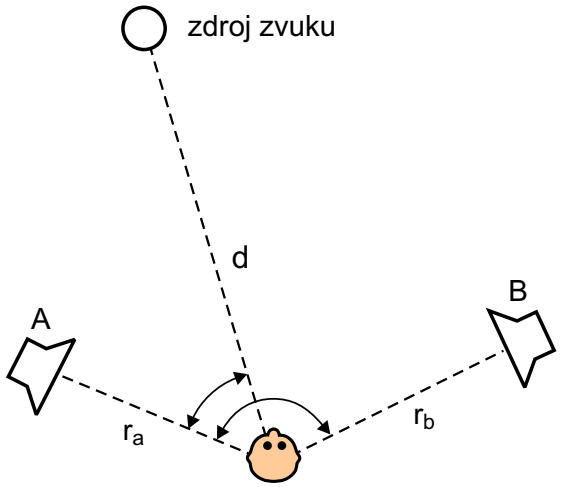
Výstup do sluchátek

Vzhledem k tomu, že sluchátka zakrývají uši posluchače a znemožňují tím působení posluchačova těla na zvuk, je nutné vliv posluchačova těla na přicházející zvuk simulovat. HRTF je sada digitálních filtrů, které v závislosti na poloze zdroje zvuku mění zvukový signál tak, jako by byl ovlivněn tělem posluchače. HRTF je možné získat měřením odezvy na jednotkový impuls uvnitř zvukovodu. Obyčejně se odezva zjišťuje pro určité klíčové polohy a HRTF pro nezměřené hodnoty se vypočítávají lineární interpolací z nejbližších naměřených hodnot. Pro každého posluchače se nejlepších výsledků dosahuje s HRTF měřenou přímo na jeho uších. To je prakticky nerealizovatelné, a proto se používají průměrované hodnoty, které vyhovují asi 80 % populace. Čím více se použitá HRTF liší od posluchačovy, tím slabší je výsledek. Jako první se ztrácí *externalizace*, tj. dojem, že zvuk přichází zvenku a nezní „uvnitř hlavy“.

Výstup pomocí reproduktorů

Pomocí sady vhodně rozmístěných reproduktorů je možno vytvořit virtuální prostor, ve kterém můžeme simulovat libovolně rozmístěné zdroje zvuku. Omezením této metody je, že se posluchač

musí pohybovat v dostatečné vzdálenosti od jednotlivých reproduktorů. S rostoucí vzdáleností od zdrojů zvuku prostorový vjem klesá.



Obrázek 20.5: Výstup pomocí reproduktorů

Dojem zdroje zvuku umístěného v prostoru mezi reproduktory (viz obrázek 20.5) vzniká superpozicí zvukových vln generovaných nejbližšími reproduktory. Označme Θ úhel mezi zdrojem zvuku a levým reproduktorem, Ψ úhel mezi oběma reproduktory, r_A a r_B jsou vzdálenosti mezi posluchačem a příslušnými reproduktory, d vzdálenost mezi simulovaným zdrojem zvuku a posluchačem, a amplitudu zvuku z tohoto zdroje zvuku. Hledané hodnoty amplitudy signálu z obou reproduktorů, označené A a B, získáme jako

$$A = \frac{1}{d} a.r_A \cos\left(\frac{\pi\Theta}{2\Psi}\right)$$
$$B = \frac{1}{d} a.r_B \sin\left(\frac{\pi\Theta}{2\Psi}\right).$$

Akustické modely pro virtuální realitu

Chceme-li zapojit realistickou simulaci prostorového zvuku do prostředí virtuální reality, musíme počítat s tím, že budeme moci pro akustické výpočty využít pouze část výpočetního výkonu počítače, a proto se budeme muset spokojit s nižší přesností simulace zvuku.

Vzhledem k tomu, že ve většině herních systémů je vizuální vjem primární (představuje největší množství vnímaných informací) a zvuku je přisuzována pouze doprovodná role, bývá často modelován jen hluk pozadí (*background noise*), dozvuk prostředí, směr a útlum hlasitosti zvukových zdrojů. Scéna je pak rozdělena na oblasti s konstatním dozvukem a hlukem pozadí. Hluk pozadí je zpravidla modelován jako smyčka zvukového signálu přiřazená určitým zvukovým zdrojům ve scéně, které plní zejména ilustrativní roli, např. šumící vodopád, bzučící včelí úl, zpívající ptáčci na stromech, atd. Samotné zvukové objekty jsou většinou chápány jako bodové zdroje zvuku s všesměrovou charakteristikou. Na jejich akustický signál jsou aplikovány filtry umožňující uživateli jejich lokalizaci a dozvuk prostředí. Nachází-li se zdroj zvuku nebo posluchač na rozhraní dvou akustických oblastí, jsou tyto vlastnosti interpolovány.

Tento přístup je velmi snadno implementovatelný pomocí hardware zvukových karet, které jsou v současnosti v podstatě již velmi vyspělými vícekanálovými signálovými procesory, schopnými počítat v reálném čase nejrůznější zkreslení akustického signálu, jako např. ozvěnu, dozvuk, změny frekvenčních charakteristik, atd.

Část E

MATEMATIKA PRO POČÍTAČOVOU GRAFIKU

Cílem tohoto oddílu, tvořeného dvěma kapitolami, je shrnout na jednom místě nejčastěji používané vzorce a elementární výpočetní postupy v počítačové grafice.

V kapitole 21 uvádíme popis běžně používaných lineárních geometrických transformací v rovině a v prostoru. Kapitola 22 pak obsahuje vzorce, na které lze v počítačové grafice narazit téměř na každém kroku. Naším cílem bylo v první řadě ušetřit čtenáři práci při jejich vyhledávaní v knihách o algebře či analytické geometrii. Současně jsme chtěli zamezit jejich opakování v různých částech knihy a raději se na ně do této kapitoly odkazujeme. V závěru této kapitoly je popsána diskrétní Fourierova transformace, která je základem zpracování obrazu, ale hraje i důležitou roli při spektrální syntéze používané v modelování.

Kapitola 21

Transformace

Geometrické transformacejsou jedněmi z nejčastěji používaných operací v počítačové grafice. Transformace je možno klasifikovat jako lineární a nelineární. Mezi lineární transformace patří otáčení, posunutí, změna měřítka, zkosení a operace vzniklé jejich skládáním. S nelineárními transformacemi se v počítačové grafice setkáváme při složitějších změnách tvaru grafických objektů, např. u deformací prostorových modelů nebo u warpingu obrazu (viz část 4.4). Zvláštní transformací je projekce, která převádí objekty z vícerozměrného prostoru do prostoru o méně rozměrech. Nejčastěji se s projekcemi budeme setkávat při převodu trojrozměrné scény do roviny obrazu. O těchto transformacích pojednává kapitola 9.

Objekty jsou popsány svými souřadnicemi, které jsou vztaženy ke zvolenému souřadnicovému systému. Geometrické transformace mohou být aplikovány na jednotlivé *souřadnice objektu*, který tak mění svou polohu. S touto operací se setkáváme například v počítačové trojrozměrné animaci, při pohybu objektu po určité dráze, při psaní textu na křivku v DTP, atp. Další možností je podrobit transformaci *souřadnicový systém*. To obyčejně činíme za účelem získání výhodnější reprezentace objektu pro jeho další zpracování. Typickým příkladem je umístění objektu do scény s více objekty. Při porovnání polohy různých objektů je nutné přepočítat (transformovat) jejich geometrické údaje buď mezi jejich souřadnicovými systémy, nebo lépe z lokálních souřadnicových systémů do světového souřadnicového systému společného pro všechny objekty.

Dále budeme hovořit o transformaci bodu P, který má kartézské souřadnice $[X, Y, Z]$ ve třech rozměrech, resp. $[X, Y]$ v rozměrech dvou. Transformací bodu P získáme bod P' o souřadnicích $[X', Y', Z']$, resp. $[X', Y']$. Transformací *objektu* budeme rozumět aplikaci transformace na všechny body, ze kterých se objekt skládá nebo, pokud to transformace a současně reprezentace objektu umožňují, aplikaci transformace na parametry, které objekt jednoznačně určují. Např. posunutí koule reprezentované čtyřmi údaji (středem a poloměrem) nebudeme

řešit transformací každého povrchového bodu, stačí pouze transformovat středový bod.

21.1 Homogenní souřadnice

Pro zjednodušení výpočtů transformací se s výhodou používá reprezentace bodů pomocí *homogenních souřadnic*. Tato reprezentace se používá z několika důvodů. Homogenní souřadnice umožňují vyjádření nejčastěji používaných lineárních transformací pomocí jediné matice, což v nehomogenních kartézských souřadnicích není možné. Další často používanou transformací vyjádřitelnou pomocí matice v homogenních souřadnicích je perspektivní promítání. Tento způsob vyjádření transformací je obzvlášť výhodný zejména proto, že pro jejich implementaci lze využít existujících knihoven pro práci s maticemi. Skládání transformací se totiž v tomto kontextu realizuje jako násobení matic, inverzní transformace je reprezentována inverzní maticí, atd. Moderní grafické procesory efektivně realizují výše uvedené maticové operace a rychlost zpracování scény se díky specializovaným grafickým kartám neustále zvyšuje.

Uspořádaná čtveřice čísel $[x, y, z, w]$ představuje homogenní souřadnice bodu P s kartézskými souřadnicemi $[X, Y, Z]$ ve třech rozměrech, platí-li:

$$X = \frac{x}{w},\ Y = \frac{y}{w},\ Z = \frac{z}{w},\ w \neq 0.$$

Bod P je svými homogenními souřadnicemi určen jednoznačně. Souřadnici w též nazýváme *váhou* bodu. Homogenní souřadnice transformovaného bodu P' s kartézskými souřadnicemi $[X', Y', Z']$ budeme označovat $[x', y', z', w']$. Často se volí $w = 1$, potom jsou homogenní souřadnice bodu $[X, Y, Z, 1]$. Rozdíl dvou bodů $A = [a_0, a_1, a_2, 1]$ a $B = [b_0, b_1, b_2, 1]$ určí vektor $\vec{p} = (a_0 - b_0, a_1 - b_1, a_2 - b_2, 0)$, sečtením bodu a vektoru dostaneme bod. Řešení úlohy nalezení průsečíku dvou rovnoběžných přímek lze v homogenních souřadnicích vyjádřit pomocí tvz. nevlastních bodů roviny, které mají váhu rovnou nule.

Obecnou matici 4×4 reprezentující lineární transformaci bodu $P = [x, y, z, w]$ na bod $P' = [x', y', z', w']$ budeme označovat $\mathbf{A}$, její speciální případy pak podle druhu transformace, např. $\mathbf{T}$ (translace), $\mathbf{R}$ (rotace). Transformaci souřadnic zapíšeme[1]

$$P' = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \mathbf{A_{4\times 4}} \cdot P = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{24} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}. \tag{21.1}$$

[1]Protože v dalším textu budeme používat násobení transformovaného bodu maticí transformace zleva, použijeme uspořádání souřadnic $P = [x, y, z, w]$ do sloupcového vektoru.

Matice reprezentující transformaci bodu $P = [x, y, w]$ na bod $P' = [x', y', w']$ ve dvou rozměrech budeme označovat stejným způsobem a pro převod souřadnic platí opět

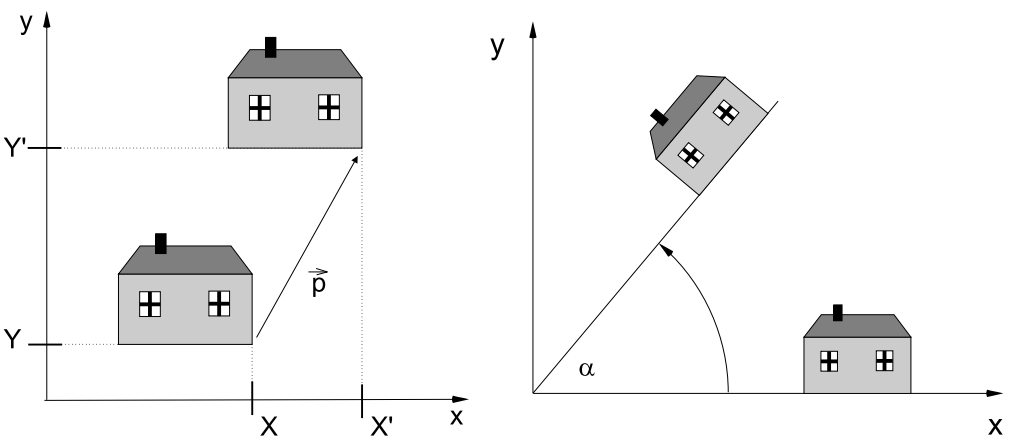
$$P' = \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \mathbf{A_{3\times 3}} \cdot P = \mathbf{A} \cdot \begin{bmatrix} x \\ y \\ w \end{bmatrix} .$$

21.2 Dvourozměrné geometrické transformace

21.2.1 Posunutí

Transformace posunutí nebo také translace (*translation, move*) bodu P (viz obrázek 21.1 vlevo) je určena vektorem posunutí $\vec{p} = (X_t, Y_t)$. Matice transformace posunutí $\mathbf{T}$ a inverzní matice $\mathbf{T}^{-1}$ mají tvar

$$\mathbf{T}(X_t, Y_t) = \begin{bmatrix} 1 & 0 & X_t \\ 0 & 1 & Y_t \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{T}^{-1}(X_t, Y_t) = \mathbf{T}(-X_t, -Y_t) = \begin{bmatrix} 1 & 0 & -X_t \\ 0 & 1 & -Y_t \\ 0 & 0 & 1 \end{bmatrix}.$$



Obrázek 21.1: Posunutí a otáčení

21.2.2 Otáčení

Otáčením (*rotation*) bodu P kolem počátku soustavy souřadnic O= $[0, 0]$ o úhel α (viz obrázek 21.1 vpravo) získáme bod P' o souřadnicích

$$\begin{aligned} X' &= X\cos\alpha - Y\sin\alpha \\ Y' &= X\sin\alpha + Y\cos\alpha. \end{aligned}$$

Matice transformace otáčení $\mathbf{R}$ a inverzí matice $\mathbf{R}^{-1}$ mají tvar

$$\mathbf{R}(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{R}^{-1}(\alpha) = \mathbf{R}(-\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

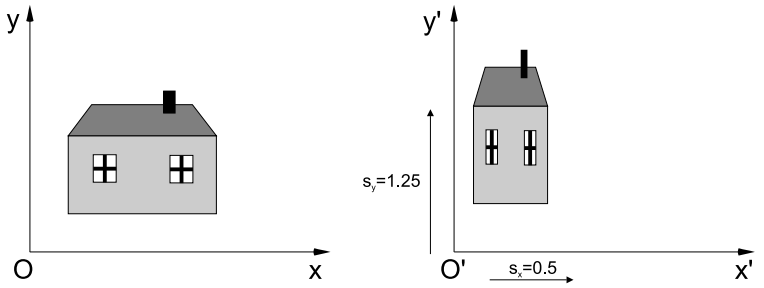
21.2.3 Změna měřítka

Změna měřítka (*scale*) ovlivňuje současně polohu i velikost transformovaného objektu ve směru souřadnicových os. Pokud je absolutní hodnota koeficientu měřítkování v intervalu $(0, 1)$, dochází ke zmenšení a přiblížení transformovaného objektu k počátku souřadnic. Je-li absolutní hodnota koeficientu větší než jedna, dojde k prodloužení, je-li znaménko koeficientu záporné, dochází k prodloužení či zmenšení v opačném směru. Příslušné transformační a inverzní matice jsou

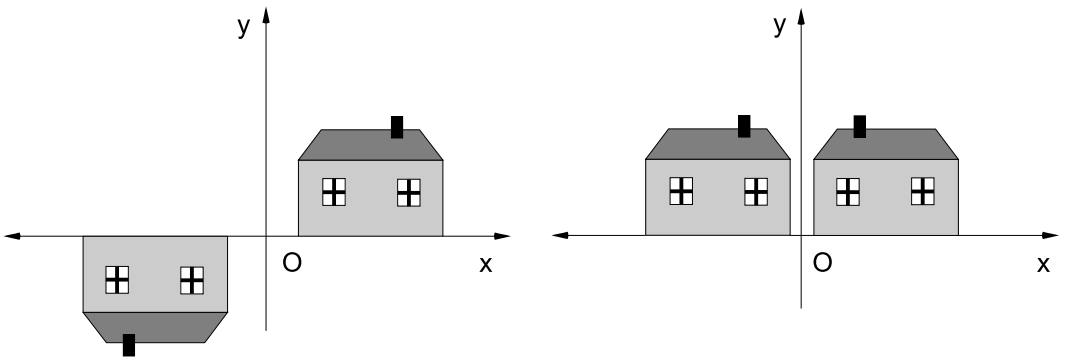
$$\mathbf{S}(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{S}^{-1}(s_x, s_y) = \mathbf{S}(1/s_x, 1/s_y) = \begin{bmatrix} 1/s_x & 0 & 0 \\ 0 & 1/s_y & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

kde s_x je koeficient změny měřítka ve směru souřadnicové osy x a s_y je koeficient změny měřítka ve směru souřadnicové osy y. Obrázek 21.2 demonstruje změnu měřítka ve směru obou souřadnicových os.

Souměrnost (*symmetry*) je zvláštním případem změny měřítka, kde absolutní hodnota koeficientu měřítka je rovna jedné. Souměrnost je možno ve dvourozměrném případě rozdělit na souměrnost *středovou* a *osovou* (viz obrázek 21.3). Středová souměrnost podle počátku souřadnicové soustavy je otáčením o 180° resp. změnou měřítka $s_x = -1$ a $s_y = -1$, zatímco osové souměrnosti získáme překlopením bodu podle jedné ze souřadnicových os. Souměrnost podle osy x má koeficienty $s_x = -1$ a $s_y = 1$, obdobně pro osu y je $s_x = 1$ a $s_y = -1$.



Obrázek 21.2: Změna měřítka



Obrázek 21.3: Středová souměrnost (vlevo) a souměrnost podle osy y (vpravo)

21.2.4 Zkosení

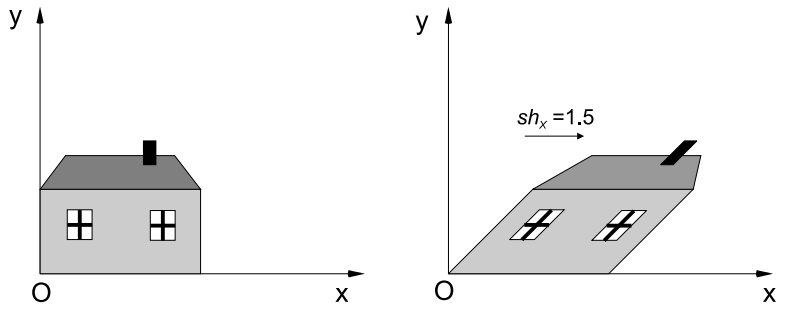
Na obrázku 21.4 je ukázáno použití transformace zkosení s koeficientem sh_x (*shear*) ve směru osy x. Analogicky se provádí zkosení ve směru osy y s koeficientem sh_y. Transformační matice mají tvar

$$\mathbf{Sh}_x(sh_x) = \begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{Sh}_y(sh_y) = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

a pro inverzní matice platí $\mathbf{Sh}_x^{-1}(sh_x) = \mathbf{Sh}_x(-sh_x)$ a $\mathbf{Sh}_y^{-1}(sh_y) = \mathbf{Sh}_y(-sh_y)$.

21.2.5 Skládání transformací

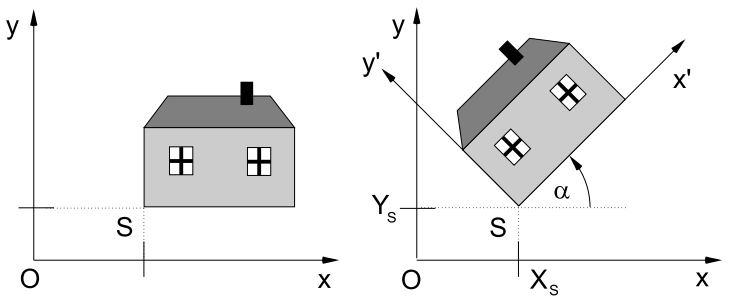
Při postupném aplikování transformací na body objektu záleží na pořadí, v jakém se transformace provádějí. Je rozdíl, jestliže bod posuneme a poté otočíme okolo počátku souřadnicového



Obrázek 21.4: Zkosení ve směru osy x

systému, nebo zda bod nejprve otočíme a poté provedeme transformaci posunutí. Transformaci vzniklou složením z více transformací lze vyjádřit jedinou maticí, již získáme postupným násobením matic, reprezentujících dílčí transformace. Protože záleží na pořadí transformací, záleží také na pořadí násobení matic. Jelikož používáme zápis $P' = \mathbf{A}.P$, musíme matice reprezentující pozdější transformace přidávat do složené transformace zleva. Pokud jsou tedy aplikovány transformace v pořadí $\mathbf{A}_1, \mathbf{A}_2$, je bod P transformován vztahem $P' = \mathbf{A}_2.\mathbf{A}_1.P$.

Například pokud chceme provést otočení kolem bodu $S = [X_s, Y_s]$ o úhel α (viz obrázek 21.5), použijeme složenou transformaci $P' = \mathbf{A}.P = \mathbf{T}(X_s, Y_s).\mathbf{R}(\alpha).\mathbf{T}(-X_s, -Y_s).P$. Objekt posuneme tak, aby střed otáčení byl v počátku, otočíme o úhel α a posuneme otočený objekt zpět.



Obrázek 21.5: Otáčení okolo bodu S

Často používané složené transformace je výhodné předem sestavit, neboť opakované násobení matic by bylo při zpracování většího objemu dat neefektivní. Grafické knihovny, jako např. OpenGL, tuto možnost poskytují.

21.3 Trojrozměrné geometrické transformace

Lineární transformace v prostoru jsou zobecněním rovinných transformací. V počítačové grafice se pro ně používají matice typu 4×4 ze vztahu (21.1).

21.3.1 Posunutí

Posunutí ve 3D je určeno vektorem posunutí $\vec{p} = (X_t, Y_t, Z_t)$. Transformační matice posunutí $\mathbf{T}$ a inverzní matice $\mathbf{T}^{-1}$ jsou

$$\mathbf{T} = \mathbf{T}(X_t, Y_t, Z_t) = \begin{bmatrix} 1 & 0 & 0 & X_t \\ 0 & 1 & 0 & Y_t \\ 0 & 0 & 1 & Z_t \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{T}^{-1} = \mathbf{T}(-X_t, -Y_t, -Z_t) = \begin{bmatrix} 1 & 0 & 0 & -X_t \\ 0 & 1 & 0 & -Y_t \\ 0 & 0 & 1 & -Z_t \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

21.3.2 Otáčení

Otáčení ve třech rozměrech může být jedním z podpřípadů otáčení kolem jednotlivých souřadnicových os. Matice $\mathbf{R}_x$ reprezentující otáčení okolo osy x o úhel α a odpovídající inverzní matice $\mathbf{R}_x^{-1}$ jsou

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{R}_x^{-1}(\alpha) = \mathbf{R}_x(-\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{21.2}$$

Obdobně jsou sestaveny matice pro otáčení kolem osy y, resp. z:

$$\mathbf{R}_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{R}_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

21.3.3 Otáčení kolem obecné osy

Otáčení kolem obecné osy v prostoru lze realizovat složením několika dílčích transformací kolem os x, y, z, nalezení příslušných transformací (úhlů otočení) však není jednoduché. Lze však využít Rodriguesovy formule [Murr94], která převádí rotační úlohu na promítání a skládání několika vektorů. Výchozí situace je znázorněna na obr.21.6. Předpokládejme, že osa otáčení

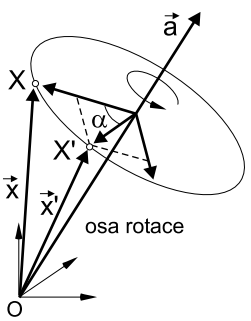
je určena počátkem souřadnicového systému O a jednotkovým vektorem $\vec{a}$. Polohový vektor $\vec{x}$ transformovaného bodu X je kolem této osy otočen o úhel α a výsledný polohový vektor je označen $\vec{x}'$. Za těchto předpokladů lze rotaci popsat vztahem

$$\vec{x}' = \cos\alpha \cdot \vec{x} + (1 - \cos\alpha)(\vec{a}.\vec{x})\vec{a} + \sin\alpha(\vec{a} \times \vec{x}).$$

Rotaci bodu X lze přepsat do maticového tvaru s využitím definic vektorového a skalárního součinu, matice $\mathbf{I}$ je jednotková matice (identita).

$$X' = \mathbf{R}(\vec{a}, \alpha).X.$$

$$\mathbf{R}(\vec{a}, \alpha) = \cos\alpha.\mathbf{I} + (1 - \cos\alpha). \begin{bmatrix} a_x^2 & a_x a_y & a_x a_z & 0 \\ a_x a_y & a_y^2 & a_y a_z & 0 \\ a_x a_z & a_y a_z & a_z^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \sin\alpha. \begin{bmatrix} 0 & -a_z & a_y & 0 \\ a_z & 0 & -a_x & 0 \\ -a_y & a_x & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{21.3}$$



Obrázek 21.6: Otáčení vektoru $\vec{x}$ kolem obecné osy $\vec{a}$

Zvolíme-li např. osu rotace shodnou s osou x, tj. $\vec{a} = [1, 0, 0]$, po dosazení do vztahu (21.3) dostaneme matici (21.2). Obecnou rotaci řešíme obdobně, jako u obecné rotace v rovině, tj. na ose rotace zvolíme bod $P = [P_x, P_y, P_z, 1]$, vypočteme jednotkový vektor ve směru osy $\vec{a}$, zvolený bod posuneme do počátku souřadnicového systému, otočíme transformované objekty o daný úhel a zpětným posunutím vrátíme výsledek do výchozí pozice. Transformační matice $\mathbf{A}$ bude složena ze tří základních transformací

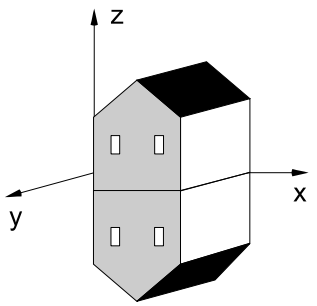
$$\mathbf{A} = \mathbf{T}(P_x, P_y, P_z).\mathbf{R}(\vec{a}, \alpha).\mathbf{T}(-P_x, -P_y, -P_z) \ .$$

21.3.4 Změna měřítka

Změnu měřítka (*scale*) v prostoru popisují transformační matice:

$$\mathbf{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \ , \quad \mathbf{S}^{-1}(s_x, s_y, s_z) = \mathbf{S}(1/s_x, 1/s_y, 1/s_z) \ ,$$

v níž koeficienty $s_x \neq 0$,$s_y \neq 0$ a $s_z \neq 0$ určují změnu ve směru příslušné souřadnicové osy. Pomocí měřítkových koeficientů můžeme realizovat některou z transformací souměrnosti



Obrázek 21.7: Souměrnost podle roviny xy

v prostoru (středovou souměrnost, souměrnost podle roviny a osovou souměrnost). Souměrnost podle roviny xy na obrázku 21.7 byla realizována pomocí koeficientů $s_x = 1, s_y = 1, s_z = -1$. Poznamenejme, že například souměrnost podle osy z je možno získat složením souměrnosti podle roviny yz a xz a středovou souměrnost složením souměrností podle všech tří rovin.

21.3.5 Zkosení

Operaci zkosení (*shear*) ve třech rozměrech můžeme rozdělit na tři případy zkosení ve směru jednotlivých rovin yz, xz a yz. Ve všech třech případech určují koeficienty sh_x, sh_y a sh_z míru zkosení v odpovídajícím směru. Matice transformace zkosení ve směru roviny yz, xz a xy mají tvar:

$$\mathbf{Sh}_{yz} = \begin{bmatrix} 1 & sh_y & sh_z & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \ \mathbf{Sh}_{xz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ sh_x & 1 & sh_z & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \ \mathbf{Sh}_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ sh_x & sh_y & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

21.4 Kvaterniony

Pro reprezentaci rotací nejsou matice vždy nejvýhodnější. Jednak obsahují nadbytečné údaje (devět čísel místo tří hodnot úhlů natočení) a hlavně jsou obtížně interpolovatelné. Přechod z jedné obecné polohy do jiné pomocí interpolovaného otáčení ve třech směrech nelze pomocí matic jednoduše vyřešit. Kvaterniony byly navrženy irským matematikem W.R.Hamiltonem v 19.století jako analogie komplexních čísel v prostoru. Praktický význam teorie kvaternionů byl rozpoznán nejen v kvantové mechanice, ale i při řešení animačních úloh v počítačové grafice.

Pro snazší vysvětlení podstaty kvaternionů začneme nejprve studiem komplexních čísel. Ukážeme, že je lze použít pro reprezentaci 2D rotací, analogicky jako lze kvaterniony použít pro reprezentaci 3D rotací. Dále popíšeme základní vlastnosti kvaternionů a jejich aritmetiku a uvedeme vzájemný převod mezi otáčením vyjádřeným pomocí matice a pomocí kvaternionu. Na závěr popíšeme praktické využití kvaternionů při interpolaci rotací.

21.4.1 Komplexní čísla a rotace v rovině

2D rotace o úhel α je lineární zobrazení, které vektoru $[x, y]$ přiřadí vektor

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} . \begin{bmatrix} x \\ y \end{bmatrix}. \tag{21.4}$$

Tento vzorec lze zapsat pomocí komplexních čísel. Označíme-li $c = x + iy$ a $c' = x' + iy'$, můžeme vztah (21.4) přepsat na

$$c' = ce^{i\alpha}.$$

Ekvivalence obou vyjádření vyplývá z Eulerovy rovnosti

$$e^{i\alpha} = \cos\alpha + i\sin\alpha. \tag{21.5}$$

2D rotace odpovídají jednotkovým komplexním číslům: každé 2D rotaci odpovídá jednotkové komplexní číslo $e^{i\alpha}$ a naopak.

21.4.2 Definice kvaternionů a základní vlastnosti

Kvaternion $\mathbf{q}$ je reprezentován čtveřicí $\mathbf{q} = w + xi + yj + zk$, kde w, x, y, z jsou reálná čísla a i, j, k jsou kvaternionové jednotky (i odpovídá komplexní jednotce). Sčítání a odčítání kvaternionů se provádí po složkách. Násobení kvaternionu skalárním číslem představuje vynásobení jednotlivých složek tímto číslem. Násobení kvaternionových jednotek se řídí několika rovnostmi:

$$ij = -ji = k,\ jk = -kj = i,\ ki = -ik = j,$$

$$i^2 = j^2 = k^2 = ijk = -1.$$

Součin dvou kvaternionů $\mathbf{q_0} = w_0 + x_0 i + y_0 j + z_0 k$ a $\mathbf{q_1} = w_1 + x_1 i + y_1 j + z_1 k$ je:

$$\begin{aligned} \mathbf{q_0 q_1} \quad = \quad & (w_0 w_1 - x_0 x_1 - y_0 y_1 - z_0 z_1) + \\ & (x_0 w_1 + w_0 x_1 + y_0 z_1 - z_0 y_1) i + \\ & (y_0 w_1 + w_0 y_1 + z_0 x_1 - x_0 z_1) j + \\ & (z_0 w_1 + w_0 z_1 + x_0 y_1 - y_0 x_1) k. \end{aligned}$$

Kvaternion si také můžeme představit jako dvojici složenou ze *skalární* (s) a *vektorové* ($\vec{v}$) části. Tento pohled vede k jednoduché notaci

$$\mathbf{q} = (s, \vec{v})$$

a násobení kvaternionů $\mathbf{q_0} = (s_0, \vec{v_0}), \mathbf{q_1} = (s_1, \vec{v_1})$ je dáno vztahem

$$\mathbf{q_0 q_1} = (s_0 s_1 - \vec{v}_0 \cdot \vec{v}_1, s_0\vec{v}_1 + s_1\vec{v}_0 + \vec{v}_0 \times \vec{v}_1). \tag{21.6}$$

Součin kvaternionů není komutativní, stejně jako složení 3D rotací kolem různých os. Kvaterniony můžeme chápat jako vektory čtyřrozměrného euklidovského prostoru. Díky tomu lze zavést skalární součin kvaternionů $\mathbf{q_0}, \mathbf{q_1}$ jako

$$\mathbf{q_0} \cdot \mathbf{q_1} = w_0 w_1 + x_0 x_1 + y_0 y_1 + z_0 z_1.$$

Kvaternion sdružený ke kvaternionu $\mathbf{q}$ definujeme jako $\mathbf{q}^* = w - xi - yj - zk$. Velikost kvaternionu $\mathbf{q}$ je $|\mathbf{q}| = \sqrt{w^2 + x^2 + y^2 + z^2}$. Kvaternion, jehož velikost je jedna nazýváme *jednotkový kvaternion*. Pro jednotkový kvaternion $\mathbf{q}$ platí $\mathbf{q}^*\mathbf{q} = \mathbf{q}\mathbf{q}^* = 1$.

Libovolnou 3D rotaci lze popsat úhlem α a jednotkovým 3D vektorem $\vec{a} = (a_0, a_1, a_2)$, který reprezentuje osu otáčení. Taková rotace odpovídá jednotkovému kvaternionu

$$\mathbf{q} = \cos(\alpha/2) + a_0 \sin(\alpha/2)i \ + a_1 \sin(\alpha/2)j + a_2 \sin(\alpha/2)k.$$

Tento zápis se obvykle zkracuje na

$$\mathbf{q} = \cos(\alpha/2) + \mathbf{a} \sin(\alpha/2), \tag{21.7}$$

přičemž $\mathbf{a} = (0, \vec{a})$ chápeme jako kvaternion se skalární částí s=0, tj. $\mathbf{a} = a_0 i \ + a_1 j + a_2 k$. Přiřazení jednotkového kvaternionu k rotaci není jednoznačné, neboť $-\mathbf{q}$ odpovídá téže rotaci jako $\mathbf{q}$. Kvaternion $1 + 0i + 0j + 0k$ představuje identitu (rotaci s nulovým úhlem). Sdružený kvaternion $\mathbf{q}^*$ reprezentuje inverzní rotaci.

Eulerovu rovnost (21.5) lze snadno zobecnit pro kvaterniony. Rotaci kolem osy procházející počátkem souřadnicového systému ve směru $\vec{a}$ o úhel 2α můžeme totiž zapsat jako

$$e^{\mathbf{a}\alpha} = \cos\alpha + \mathbf{a}\sin\alpha, \tag{21.8}$$

kde $\mathbf{a}$ chápeme stejně jako ve vztahu (21.7). Tento vzorec dostaneme z Eulerovy rovnosti (21.5) záměnou komplexní jednotky i za kvaternion $\mathbf{a}$. Z vyjádření (21.8) plyne také definice mocniny jednotkového kvaternionu $\mathbf{q}$

$$\mathbf{q}^t = e^{\mathbf{a}\alpha t} = \cos(\alpha t) + \mathbf{a}\sin(\alpha t),$$

21.4.3 Rotace pomocí kvaternionů

Vektor $\vec{v} = (v_0, v_1, v_2)$ můžeme chápat jako kvaternion $\mathbf{v}$ s nulovou reálnou částí, tedy $\mathbf{v} = = v_0 i \ + v_1 j + v_2 k$. Otočení vektoru $\vec{v}$ jednotkovým kvaternionem $\mathbf{q} = \cos(\alpha/2) + \mathbf{a}\sin(\alpha/2)$ kolem osy $\mathbf{a}$ o úhel α spočítáme pomocí kvaternionového násobení

$$\mathbf{v}' = \mathbf{q}\mathbf{v}\mathbf{q}^*. \tag{21.9}$$

Platí, že reálná část kvaternionu $\mathbf{v}'$ vyjde vždy nulová, tedy $\mathbf{v}' = v_0' i \ + v_1' j + v_2' k$. To nás opravňuje tvrdit, že vektor (v_0', v_1', v_2') představuje rotaci vektoru $\vec{v}$ zadanou kvaternionem $\mathbf{q}$.

Vzorec (21.9) má důležitý důsledek pro skládání rotací. Mějme další rotaci reprezentovanou jednotkovým kvaternionem $\mathbf{r}$ a otočme jím vektor $\vec{v}'$ reprezentovaný kvaternionem $\mathbf{v}'$. Výsledek otáčení lze napsat jako

$$\mathbf{v}'' = \mathbf{r}\mathbf{v}'\mathbf{r}^* = \mathbf{r}\mathbf{q}\mathbf{v}\mathbf{q}^*\mathbf{r}^*.$$

Vidíme že je to totéž, jako kdybychom vektor $\vec{v}$ otočili pomocí jednotkového kvaternionu $\mathbf{rq}$. Můžeme tedy shrnout: složení rotací odpovídá násobení kvaternionů. Násobení kvaternionů vyžaduje méně aritmetických operací než násobení rotačních matic.

Výpočet otočení vektoru podle rovnice (21.9) je pomalejší, než násobení vektoru rotační maticí. Potřebujeme tedy převod mezi kvaterniony a rotačními maticemi. Je důležitý i proto, že některé knihovny a grafický hardware používají rotace v maticovém tvaru. Převod *jednotkového* kvaternionu $\mathbf{q} = w + xi + yj + zk$ na rotační matici $\mathbf{R}$ je snadný:

$$\mathbf{R(q)} = \begin{bmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy & 0 \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx & 0 \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Při opačném převodu, z rotační matice $\mathbf{R}$ na kvaternion $\mathbf{q}$, musíme být opatrnější. Označme prvky této matice

$$\mathbf{R} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & 0 \\ r_{10} & r_{11} & r_{12} & 0 \\ r_{20} & r_{21} & r_{22} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Stopa matice je součet diagonálních prvků, v našem případě $st(\mathbf{R}) = r_{00} + r_{11} + r_{22} + 1$. Abychom se vyhnuli dělení nulou, či číslem blízkým nule, rozlišíme několik případů. Je-li $\mathbf{R} > 1$, obdržíme koeficienty kvaternionu $\mathbf{q}$ takto

$$w = \frac{\sqrt{st(\mathbf{R})}}{2},\ x = \frac{r_{21}-r_{12}}{4w},\ y = \frac{r_{02}-r_{20}}{4w},\ z\ = \frac{r_{10}-r_{01}}{4w}.$$

V případě $st(\mathbf{R}) \leq 1$ rozlišíme další tři možnosti

- pokud $r_{00} = max(r_{00}, r_{11}, r_{22})$, spočítáme

$$x = \frac{\sqrt{r_{00} - r_{11} - r_{22} + 1}}{2},\ w = \frac{r_{21} - r_{12}}{4x},\ y = \frac{r_{10} + r_{01}}{4x},\ z\ = \frac{r_{20} + r_{02}}{4x}$$

- pokud $r_{11} = max(r_{00}, r_{11}, r_{22})$, spočítáme

$$y = \frac{\sqrt{r_{11} - r_{00} - r_{22} + 1}}{2},\ w = \frac{r_{02} - r_{20}}{4y},\ x = \frac{r_{10} + r_{01}}{4y},\ z\ = \frac{r_{21} + r_{12}}{4y}$$

- pokud $r_{22} = max(r_{00}, r_{11}, r_{22})$, spočítáme

$$z\ = \frac{\sqrt{r_{22} - r_{00} - r_{11} + 1}}{2},\ w = \frac{r_{10} - r_{01}}{4z},\ x = \frac{r_{20} + r_{02}}{4z},\ y = \frac{r_{21} + r_{12}}{4z}$$

Porovnáním rotace vyjádřené pomocí kvaternionů se vztahem 21.3.3 zjistíme, že výsledek je totožný s Rodriguesovou formulí [Murr94]. Pro úplnost a pohodlí čtenářů připojujeme matici rotace rozepsanou pro jednotkový kvaternion $\mathbf{q} = (\cos(\alpha/2), \sin(\alpha/2)\vec{a})$ vytvořený na základě požadované osy rotace $\vec{a} = (a_x, a_y, a_z)$ (jednotkový vektor) a úhlu otočení α.

$$\mathbf{R}(\vec{a}, \alpha) =$$

$$\begin{bmatrix} a_x^2(1-\cos\alpha)+\cos\alpha & a_xa_y(1-\cos\alpha)-a_z\sin\alpha & a_xa_z(1-\cos\alpha)+a_y\sin\alpha & 0 \\ a_xa_y(1-\cos\alpha)+a_z\sin\alpha & a_y^2(1-\cos\alpha)+\cos\alpha & a_ya_z(1-\cos\alpha)-a_x\sin\alpha & 0 \\ a_xa_z(1-\cos\alpha)-a_y\sin\alpha & a_ya_z(1-\cos\alpha)+a_x\sin\alpha & a_z^2(1-\cos\alpha)+\cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

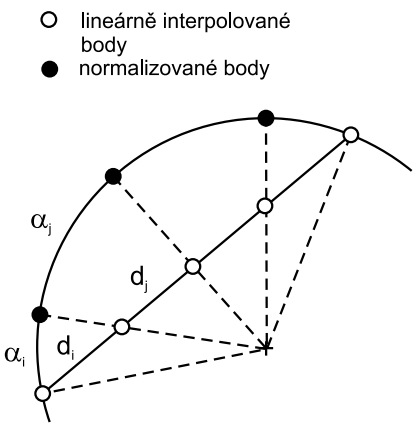
21.4.4 Sférická lineární interpolace

Jednou z nejznámějších aplikací kvaternionů je sférická lineární interpolace mezi dvěma 3D rotacemi, často označovaná zkratkou *SLERP (Spherical Linear intERPolation)*. Lineární interpolace 3D rotací (z nichž každá může být prováděna kolem jiné osy) není triviální, pokud chceme, aby interpolované rotace měly rozumné vlastnosti. Pokus o interpolaci rotační matice prvek po prvku selže, neboť výsledkem nemusí být matice rotace. Ani interpolace kvaternionů prvek po prvku nedává požadované výsledky, a to i v případě, že spočítaný kvaternion normalizujeme na jednotkovou velikost. Ukažme si nejprve, v čem je problém ve 2D případě.

Víme, že dvě 2D rotace můžeme zadat dvěma komplexními čísly r_0, r_1. Naivní lineární interpolace by vypadala takto

$$r_t = (1-t)r_0 + tr_1,\ t \in \langle 0, 1\rangle.$$

Výsledná velikost $|r_t|$ nemusí být jedna, ale můžeme ji snadno normalizovat $r_t/|r_t|$. Konstantním krokem neinterpolujeme rotaci, ale úsečku danou body r_0, r_1. Kdybychom interpretovali parametr t jako čas, vidíme, že na začátku se úhel mění pomaleji než uprostřed, jak je znázorněno na obrázku 21.8.



Obrázek 21.8: Rovnoměrná interpolace po úsečce vede na nerovnoměrnou interpolaci úhlu. Přestože $d_i = d_j$, ve výsledku je obecně $\alpha_i \neq \alpha_j$.

Je snadné navrhnout lepší schéma pro interpolaci 2D rotací, stačí jednoduše interpolovat úhel. Situace je trochu komplikovanější ve 3D, protože rotace nemusejí mít společnou osu. S pomocí kvaternionů se však dá navrhnout i v tomto případě kvalitní interpolace [Shoe85]. Uvažme dvě 3D rotace zadané jednotkovými kvaterniony $\mathbf{p}, \mathbf{q}$. Výsledek sférické lineární interpolace v bodě $t \in \langle 0, 1 \rangle$ dostaneme podle předpisu

$$\mathbf{s_t} = \mathbf{p}(\mathbf{p}^*\mathbf{q})^t. \tag{21.10}$$

$\mathbf{s_t}$ je jednotkový kvaternion. Když označíme $\mathbf{r} = \mathbf{p}^*\mathbf{q}$ a rozepíšeme jako $\mathbf{r} = e^{\mathbf{h}\sigma}$, můžeme přepsat (21.10) na

$$\mathbf{s_t} = \mathbf{p}\mathbf{r}^t = \mathbf{p}e^{\mathbf{h}\sigma t},$$

což říká, že $\mathbf{s_t}$ je složením konstantní rotace $\mathbf{p}$ a rotace kolem osy $\mathbf{h}$ o úhel $2\sigma t$. Z toho plyne, že interpolace je skutečně rovnoměrná.

Na první pohled k jinému výsledku dojdeme, když si prostor jednotkových kvaternionů představíme jako povrch jednotkové koule v 4D prostoru. Každé rotaci odpovídá bod na této kouli, takže přirozenou interpolací dvou rotací (bodů na povrchu 4D koule) je nejkratší oblouk, který je spojuje. Tímto způsobem lze odvodit vzorec (viz [Eber01])

$$\mathbf{s_t} = \frac{\sin((1-t)\sigma)\mathbf{p} + \sin(t\sigma)\mathbf{q}}{\sin\sigma}, \tag{21.11}$$

Tento předpis je výpočetně výhodnější než (21.10), přičemž oba vzorce jsou ekvivalentní.

Pozorný čtenář si mohl všimnout, že ačkoliv jednotkové kvaterniony $\mathbf{p}$ a $-\mathbf{p}$ reprezentují tutéž rotaci, výsledek sférické lineární interpolace mezi $\mathbf{p}, \mathbf{q}$ a mezi $-\mathbf{p}, \mathbf{q}$ se může lišit. Chceme-li minimalizovat otočení v průběhu interpolace, musíme zabezpečit, aby skalární součin $\mathbf{p}\cdot\mathbf{q} \geq 0$. To lze naštěstí snadno zařídit: pokud $\mathbf{p}\cdot\mathbf{q} < 0$, použijeme namísto $\mathbf{p}$ kvaternion $-\mathbf{p}$.

Kapitola 22

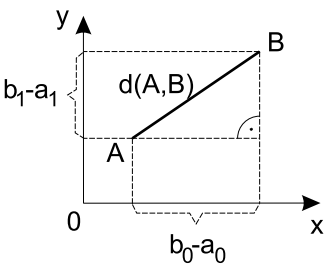
Často používané vzorce

Jak bylo uvedeno již v úvodu této části, snažíme se na jedno místo soustředit často používané vzorce. Nejde však o nijak soustavný přehled, jakých existuje celá řada (namátkou [Bart96, Rekt95]). Snažili jsme se uvést jen ty, které může čtenář v oboru počítačové grafiky přímo využít.

22.1 Pojmy a značení

Vzdálenost je zde vždy chápána jako euklidovská. O jiných metrikách se zmiňujeme při úvahách o spojitosti v částech 7.2.2 a 3.1.2. Zároveň předpokládáme, že pracujeme v kartézské soustavě souřadnic (její bázové vektory jsou jednotkové a navzájem kolmé).

Bod je dán uspořádanou n-ticí souřadnic v hranatých závorkách, např. v rovině jako bod $P = [p_0, p_1]$, v prostoru jako $Q = [q_0, q_1, q_2]$. Body označujeme velkými písmeny A, B, C, P, souřadnice malými. Vrcholy polygonu označíme čísly v závorce $P(8)$ či indexy P_8, s orientací proti smyslu pohybu hodinových ručiček.



Obrázek 22.1: Značení souřadnic

Toto značení je v souladu s radou O'Rourkeho [O'Ro94], která říká, že při programování je vhodnější uchovávat souřadnice nikoliv jako záznam se složkami $x, y, \ldots$, ale jako pole s indexy $0, 1, \ldots, n$. Při označování souřadnic čísly se snadněji provádí cyklus přes všechny souřadnice a snadněji se přechází do vyšších rozměrů.

Vektor budeme označovat šipkou a souřadnice psát do kulatých závorek $\vec{s} = (s_0, s_1, s_2)$. *Přímku* procházející dvěma body A, B a *úsečku* s koncovými body A, B pojmenujeme dvojicí písmen či malým písmenem, např. $q = AB$, a v textu vždy uvedeme slovo přímka či úsečka.

22.2 Základy práce s vektory

22.2.1 Velikost vektoru a vzdálenost dvojice bodů

Velikost vektoru $\vec{u}$ umístěného do bodů $A = [a_0, a_1, a_2]$ a $B = [b_0, b_1, b_2]$, která je rovna vzdálenosti bodů A, B, se spočítá jako

$$d = d(A, B) = \sqrt{(b_0 - a_0)^2 + (b_1 - a_1)^2 + (b_2 - a_2)^2}. \tag{22.1}$$

Na pořadí souřadnic bodů v závorkách nezáleží, protože jejich rozdíly jsou celé ve druhé mocnině. Jde spíše o zvyklost, neboť souřadnice vektoru $\vec{u}$ z bodu A do bodu B jsou určeny jako rozdíly souřadnic bodů, $\vec{u} = (b_0 - a_0, b_1 - a_1, b_2 - a_2)$. Úsečka AB tak získává svoji orientaci a hovoříme o *orientované úsečce*. Přímka p, která prochází body A a B, je bodem A rozdělena na dvě části. Polopřímka z bodu A procházející bodem B je kladně orientovaná a označujeme ji p_+, polopřímka z bodu A na opačnou stranu je orientována záporně a značí se p_-. *Paprsek* je *orientovaná polopřímka* vedoucí např. od pozorovatele do scény.
Speciálně vzdálenost bodu $A[a_0, a_1, a_2]$ od *počátku soustavy souřadnic* $O = [0, 0, 0]$, a tedy i délka (velikost) vektoru $\vec{a} = (a_0, a_1, a_2)$ je

$$d = d(O, A) = |\vec{a}| = \sqrt{a_0^2 + a_1^2 + a_2^2}. \tag{22.2}$$

Vektor $\vec{a}$ bývá zvykem označovat jako *rádiusvektor bodu* A. Délka vektoru lze vyjádřit též skalárním součinem $|\vec{a}| = \sqrt{\vec{a}.\vec{a}} = \sqrt{\vec{a}^2}$. Vektor, jehož délka $|\vec{u}| = 1$, označujeme jako *jednotkový vektor*, vektor o souřadnicích $(0, 0, 0)$ má délku 0 a nazývá se *nulový vektor*. Dále platí, že $k\vec{u} = (ku_0, ku_1, ku_2)$ a pro $k \geq 0$ je $|k\vec{u}| = k|\vec{u}|$ (násobení konstantou k). Vektory $\vec{u}$ a $\vec{v} = k\vec{u}, k \neq 0$, jsou *lineárně závislé* a pro $k > 0$ *souhlasně kolineární*, tj. po umístění do jednoho počátečního bodu A leží v jedné přímce a jejich koncové body leží na stejné polopřímce vycházející z bodu A. Značíme $\vec{u}\uparrow\uparrow\vec{v}$. Pro $k < 0$ jsou *nesouhlasně kolineární* ($\vec{u}\uparrow\downarrow\vec{v}$). Jednotkový vektor získáme pomocí *normalizace vektoru* vydělením souřadnic nenulového vektoru jeho délkou $\vec{u}' = (u_0/|\vec{u}|, u_1/|\vec{u}|, u_2/|\vec{u}|)$.

22.2.2 Součet a rozdíl vektorů, opačný vektor

Vektory se sčítají a odečítají po jednotlivých souřadnicích, výsledkem je vektor

$$\vec{w} = \vec{u} \pm \vec{v} = (u_0 \pm v_0, u_1 \pm v_1, u_2 \pm v_2). \tag{22.3}$$

Součet vektorů je komutativní ($\vec{u}+\vec{v}=\vec{v}+\vec{u}$) a asociativní ($\vec{u}+(\vec{v}+\vec{w})=(\vec{u}+\vec{v})+\vec{w}$). Rozdíl vektorů není komutativní ($\vec{u}-\vec{v}=-(\vec{v}-\vec{u})$). Součet i rozdíl vektorů je distributivní vzhledem k násobení konstantou ($k(\vec{u}\pm\vec{v})=k\vec{u}\pm k\vec{v}$). *Opačný vektor* $-\vec{u}=(-u_0,-u_1,-u_2)$.

22.2.3 Skalární součin vektorů

Skalárním součinem dvou vektorů $\vec{u}$ a $\vec{v}$ je skalár (tj. číslo) a pro vektory ve 3D je definován jako

$$\vec{u}.\vec{v}=u_0v_0+u_1v_1+u_2v_2 \tag{22.4}$$

Označme $\varphi=(\widehat{\vec{u},\vec{v}})$ *odchylku vektorů* (tj. úhel, který svírají), pak platí

$$\vec{u}.\vec{v}=|u||v|\cos\varphi. \tag{22.5}$$

Z tohoto vzorce vychází i vztah pro výpočet kosinu úhlu φ dvou nenulových vektorů

$$\cos\varphi=\cos(\widehat{\vec{u},\vec{v}})=\frac{\vec{u}.\vec{v}}{|\vec{u}||\vec{v}|}=\frac{u_0v_0+u_1v_1+u_2v_2}{\sqrt{(u_0^2+u_1^2+u_2^2)(v_0^2+v_1^2+v_2^2)}}, \tag{22.6}$$

kde $\varphi\in\langle 0,\pi\rangle$. Konkrétně pro souhlasně kolineární vektory $\vec{u}\uparrow\uparrow\vec{v}$ je $\varphi=0$ a pro nesouhlasně kolineární $\vec{u}\uparrow\downarrow\vec{v}$ klademe $\varphi=\pi$. V rovině můžeme zavést orientaci úhlu a úhel měřit ve smyslu otáčení hodinových ručiček. V rovině je tedy úhel $\varphi\in\langle 0,2\pi\rangle$.

Připomeňme, že skalární součin dvou vektorů je nulový tehdy, pokud je některý z nich nulový, nebo pokud jsou oba nenulové a na sebe kolmé. Proto, potřebujeme-li k nenulovému vektoru $\vec{u}$ získat kolmý vektor $\vec{v}\perp\vec{u}$, lze z (22.4) v rovině odvodit např. $\vec{v}=(-u_1,u_0)$ nebo $\vec{v}'=(u_1,-u_0)$, přičemž první je pootočen o 90° vlevo, druhý vpravo. V prostoru je kolmých vektorů nekonečně mnoho, tedy např. $\vec{v}=(1,1,-(u_0+u_1)/u_2)$.

Skalární součin je komutativní ($\vec{u}.\vec{v}=\vec{v}.\vec{u}$), distributivní ($(\vec{u}+\vec{v}).\vec{w}=\vec{u}.\vec{w}+\vec{v}.\vec{w}$) a asociativní vůči násobení skalárem ($k(\vec{u}.\vec{v})=(k\vec{u}).\vec{v}=\vec{u}.(k\vec{v})$). Pro trojici vektorů však $\vec{u}(\vec{v}.\vec{w})\neq(\vec{u}.\vec{v})\vec{w}$.

Průmět vektoru do vektoru Kolmý (pravoúhlý) průmět vektoru $\vec{u}$ do jednotkového vektoru $\vec{s}$, jinými slovy složku vektoru $\vec{u}$ ve směru jednotkového vektoru $\vec{s}$, označujeme jako $\vec{u}_s$. Určí se dle vztahu

$$\vec{u}_s=|u|cos(\widehat{u,s})=(\vec{u}.\vec{s})\vec{s}. \tag{22.7}$$

Pokud není vektor $\vec{s}$ jednotkový, pak doplníme předchozí vztah o dělení délkou $|\vec{s}|$

$$\vec{u}_s=\frac{(\vec{u}.\vec{s})\vec{s}}{|\vec{s}|^2}. \tag{22.8}$$

Průměty vektoru $\vec{u}$ do bázových vektorů kartézské soustavy souřadnic $\vec{\imath}, \vec{\jmath}$ a $\vec{k}$ nazýváme *složkami vektoru* $\vec{u}$, pro něž platí

$$\vec{u}_x = (\vec{u}.\vec{\imath})\vec{\imath} = (u_0, 0, 0), \qquad \vec{u}_y = (\vec{u}.\vec{\jmath})\vec{\jmath} = (0, u_1, 0), \qquad \vec{u}_z = (\vec{u}.\vec{k})\vec{k} = (0, 0, u_2),$$

a skaláry

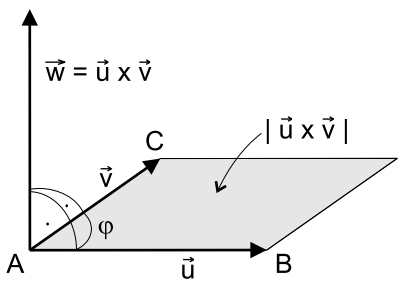
$$u_0 = u_x = \vec{u}.\vec{\imath}, \qquad u_1 = u_y = \vec{u}.\vec{\jmath}, \qquad u_2 = u_z = \vec{u}.\vec{k}$$

nazýváme *souřadnicemi vektoru* $\vec{u}$.

22.2.4 Vektorový součin

Vektorový součin $\vec{w}$ vektorů $\vec{u}$ a $\vec{v}$ lze pomocí bázových vektorů kartézské soustavy souřadnic $\vec{\imath}, \vec{\jmath}$ a $\vec{k}$ zapsat ve tvaru

$$\vec{w} = \vec{u} \times \vec{v} = \begin{vmatrix} \vec{\imath} & \vec{\jmath} & \vec{k} \\ u_0 & u_1 & u_2 \\ v_0 & v_1 & v_2 \end{vmatrix} = (u_1v_2 - u_2v_1)\vec{\imath} + (u_2v_0 - u_0v_2)\vec{\jmath} + (u_0v_1 - u_1v_0)\vec{k}. \qquad (22.9)$$



Obrázek 22.2: Geometrický význam vektorového součinu (podle [Budi83])

Výsledkem vektorového součinu je vektor $\vec{w}$ kolmý na rovinu určenou vektory $\vec{u}$ a $\vec{v}$, jehož velikost $|\vec{w}|$ je číselně rovna *obsahu P rovnoběžníku* sestrojeného nad oběma vektory umístěnými do společného bodu (viz obr. 22.2). Tento obsah se obvykle vyjadřuje jako

$$P = |\vec{u} \times \vec{v}| = |\vec{u}||\vec{v}| \sin \varphi. \qquad (22.10)$$

Obsah trojúhelníka sestrojeného nad oběma vektory (ABC na obrázku 22.2) je roven $\frac{1}{2}P$.

Protože je výsledný vektor $\vec{w}$ kolmý na rovinu daných vektorů $\vec{u}$ a $\vec{v}$, platí $\vec{u}.\vec{w} = 0$ a $\vec{v}.\vec{w} = 0$. Vektorový součin je antikomutativní ($\vec{u} \times \vec{v} = -(\vec{v} \times \vec{u})$), distributivní ($\vec{u} \times (\vec{v} + \vec{w}) = \vec{u} \times \vec{v} + \vec{u} \times \vec{w}$) a asociativní vzhledem k násobení skalárem ($n(\vec{u} \times \vec{v}) = (n\vec{u}) \times \vec{v} = \vec{u} \times (n\vec{v})$). Vektorové násobení trojice vektorů (tzv. *dvojný součin*) však asociativní není ($\vec{u} \times (\vec{v} \times \vec{w}) \neq (\vec{u} \times \vec{v}) \times \vec{w}$). Výsledkem vektorového součinu dvou lineárně závislých vektorů (kolineárních, rovnoběžných) je nulový vektor.

Velikost vektorového součinu v rovině Omezíme-li se na vektory, které leží v rovině, tj. $\vec{u} = (u_0, u_1, 0)$ a $\vec{v} = (v_0, v_1, 0)$, pak po dosazení do vzorce (22.9) vyjdou dle první dvě souřadnice výsledného vektoru nulové ($\vec{w} = (0, 0, w_2)$). Délka vektoru $\vec{w}$ je daná souřadnicí w_2

a současně určuje *orientovaný obsah P rovnoběžníku* zadaného v rovině třemi body $A[a_0, a_1, a_2]$, $B[b_0, b_1, b_2]$ a $C[c_0, c_1, c_2]$ či dvěma vektory $\vec{u} = \overrightarrow{AB}$ a $\vec{v} = \overrightarrow{AC}$ umístěnými do vrcholu A.

$$\begin{aligned} P &= u_0 v_1 - v_0 u_1 && (22.11)\\ &= (b_0 - a_0)(c_1 - a_1) - (c_0 - a_0)(b_1 - a_1) && (22.12)\\ &= a_0 b_1 + b_0 c_1 + c_0 a_1 - a_0 c_1 - b_0 a_1 - c_0 b_1. \end{aligned}$$

Orientovaný obsah je kladný v případě, že výsledný vektor směřuje ve směru osy z, tj. při pohledu z kladné poloosy z_+ na tento rovnoběžník máme vektor $\vec{u}$ po pravici a $\vec{v}$ po levici. *Orientovaný obsah trojúhelníka ABC* je roven $\frac{1}{2}P$, neorientovaný obsah $\frac{1}{2}|P|$.

22.3 Bod

V této části se budeme zabývat určováním vzdálenosti a vzájemné polohy bodu a geometrických primitiv, zejména úsečky, kružnice, koule a mnohoúhelníka.

22.3.1 Vzdálenost dvou bodů

Vzdálenost dvou bodů je rovna délce vektoru, který je do nich umístěn a spočítá se dle vztahu (22.1).

22.3.2 Vzdálenost bodu od přímky v rovině

V rovině je dána přímka, jejíž *obecná rovnice* je $ax + by + c = 0$, kde a, b a c jsou reálné konstanty a alespoň $a \neq 0$ nebo $b \neq 0$. Vzdálenost bodu $P = [p_0, p_1]$ od této přímky je:

$$d = \left| \frac{ap_0 + bp_1 + c}{\sqrt{a^2 + b^2}} \right|. \qquad (22.13)$$

Po vydělení obecné rovnice přímky členem $\sqrt{a^2 + b^2}$ získáme *normálový tvar rovnice přímky* $x \cos\alpha + y \sin\alpha - c' = 0$, kde $\cos\alpha = a/\sqrt{a^2 + b^2}$, $\sin\alpha = b/\sqrt{a^2 + b^2}$. Vektor $(\cos\alpha, \sin\alpha)$ je jednotkovým normálovým vektorem, tj. vektorem k přímce kolmým, a člen c' je roven vzdálenosti přímky od počátku. Vzdálenost bodu $P = [p_0, p_1]$ od přímky v normálovém tvaru vypočteme dosazením:

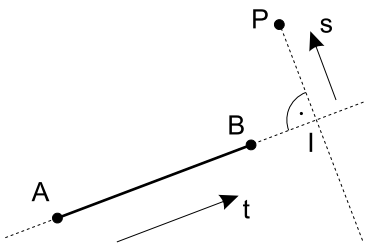
$$d = \left| p_0 \cos\alpha + p_1 \sin\alpha - c' \right|. \qquad (22.14)$$

Často je přímka v rovině určena dvěma body $A[a_0, a_1]$ a $B[b_0, b_1]$, $A \neq B$, kterými prochází. K výpočtům se potom používá *parametrické vyjádření* přímky $Q(t)$

$$Q(t) = A + (B - A)t, \text{ vektorově } Q(t) = A + \vec{q}t, \tag{22.15}$$

kde parametr t je reálné číslo z intervalu $(-\infty, \infty)$ a rozdíl $\vec{q} = (B - A)$ je *směrovým vektorem přímky* orientovaným z A do B. Uvedený zápis vyjadřuje dvojici rovnic:

$$q_0 = (1 - t)a_0 + b_0 t \qquad q_1 = (1 - t)a_1 + b_1 t. \tag{22.16}$$



Obrázek 22.3: Vzdálenost bodu od úsečky a význam parametrů t a s

Orientovaná vzdálenost bodu P od přímky AB v rovině, tj. vzdálenost opatřená znaménkem „+“, resp. „−“ podle toho, zda je bod nalevo či napravo od přímky AB, se vypočítá jako orientovaná vzdálenost bodu P a jeho kolmého průmětu I na úsečku AB. Situace je naznačena na obrázku 22.3.

Označíme-li jako L délku úsečky, která je rovna vzdálenosti bodů $d(A, B)$, zapíšeme vztahy pro výpočet parametrů t_I a s_P parametrických rovnic úseček AB a IP jako:

$$t_I = \frac{(b_0 - a_0)(p_0 - a_0) + (b_1 - a_1)(p_1 - a_1)}{L^2}, \tag{22.17}$$

$$s_P = \frac{(b_0 - a_0)(p_1 - a_1) - (b_1 - a_1)(p_0 - a_0)}{L^2}. \tag{22.18}$$

Souřadnice bodu I se určí z parametrické rovnice (22.15):

$$\begin{aligned} i_0 &= a_0 + t_I(b_0 - a_0), \\ i_1 &= a_1 + t_I(b_1 - a_1). \end{aligned} \tag{22.19}$$

Orientovaná vzdálenost od bodu A k bodu I, tj. vzdálenost dvou bodů na orientované přímce AB, je pak

$$d(A, I) = t_I L \tag{22.20}$$

a orientovaná vzdálenost bodu P od přímky AB je

$$d(P, AB) = d(P, I) = s_P L. \tag{22.21}$$

Parametr t_I má stejný význam jako v parametrické rovnici přímky (22.15) a vyjadřuje relativní vzdálenost kolmého průmětu I (průmětu bodu P na přímku AB) od bodu A. Proto

$t_I < 0$ označuje bod na přímce před body A, B, hodnota $t_I \in \langle 0, 1\rangle$ bod úsečky AB a $t_I > 0$ bod na přímce ve směru protažení úsečky AB.

Znaménko parametru s_P určuje, zda bod leží nalevo či napravo od přímky (úsečky) AB (v praxi stačí ověřit znaménko čitatele). Pro $s_P > 0$ leží P nalevo od přímky, pro $s_P < 0$ leží P napravo a pro $s_P = 0$ leží bod P na přímce AB. Čitatel ve vzorci (22.18) je roven obsahu rovnoběžníku jako v případě (22.12) a uvedený postup využívá faktu, že poměr parametru s_P k délce L je roven poměru obsahů rovnoběžníku a čtverce nad úsečkou AB.

Zajímá-li nás pouze vzdálenost $d(P, AB)$ a ne poloha bodu I, lze použít vztah (22.25) z další části.

22.3.3 Vzdálenost bodu od přímky v prostoru

V prostoru lze přímku zadat dvojicí obecných rovnic rovin a vznikne jako jejich průsečnice. K popisu přímky zadané dvojicí různých bodů $A[a_0, a_1, a_2]$ a $B[b_0, b_1, b_2]$, kterými prochází, používá častěji *parametrické vyjádření*, které je formálně shodné se vztahem 22.15. Ve 3D se jedná o trojici rovnic:

$$p_0 = (1-t)a_0 + b_0 t \qquad p_1 = (1-t)a_1 + b_1 t \qquad p_2 = (1-t)a_2 + b_2 t \tag{22.22}$$

Vzdálenost bodu $P[p_0, p_1, p_2]$ od přímky procházející body $A = [a_0, a_1, a_2], B[b_0, b_1, b_2]$ v prostoru se vypočítá následujícím postupem. Nejprve se určí poloha pravoúhlého průmětu I bodu P na orientovanou přímku AB. K tomu je třeba znát parametr t_I:

$$t_I = \frac{(b_0 - a_0)(p_0 - a_0) + (b_1 - a_1)(p_1 - a_1) + (b_2 - a_2)(p_2 - a_2)}{L^2}, \tag{22.23}$$

kde L je délka úsečky AB.

Dosazením parametru t_I do (22.22) se vypočítá poloha hledaného pravoúhlého průmětu bodu P. Dosazením získaných souřadnic do vzorce (22.1) se spočítá absolutní (neorientovaná) vzdálenost bodu P od úsečky, tj. $(d(P, AB) = d(P, I))$. Vzdálenost je neorientovaná proto, že v prostoru neexistují samostatné pojmy nalevo, napravo. Museli bychom doplnit informaci o tom, ve které z dvojice rovin body A, B a C leží, neboť těmito body procházejí dvě orientované roviny lišící se orientací normálového vektoru $(\vec{n_1} = -\vec{n_2})$.

Celý postup lze stručně zapsat vzorcem (dle [Bart96, str. 275]):

$$d = \left| -(P - A) + \frac{(P - A).\vec{q}}{\vec{q}.\vec{q}}\vec{q} \right| \tag{22.24}$$

kde $\vec{q}.\vec{q}$ lze zapsat též jako $|\vec{q}|^2$ a výsledkem je hodnota parametru t jako v (22.23).

Vzdálenost bodu $P[p_0, p_1, p_2]$ od přímky procházející body $A[a_0, a_1, a_2], B[b_0, b_1, b_2]$ v prostoru, pokud nepotřebujeme znát polohu kolmého průmětu, lze vypočítat i dle vzorce:

$$d = \frac{|\ \vec{AP} \times \vec{q}|}{|\vec{q}|}. \tag{22.25}$$

Vzdálenost bodu P od přímky AB lze chápat jako výšku rovnoběžníku určeného trojicí bodů A, B, C. Obsah rovnoběžníku lze vyjádřit jednak jako součin délky podstavy krát hledaná výška a jednak jako velikost vektorového součinu $|\ \vec{AB} \times \vec{AP}\ |$. Jednoduchou úpravou získáme uvedený vztah (22.25).

22.3.4 Vzdálenost bodu od úsečky

Leží-li bod v pásu vymezeném dvěma kolmicemi na úsečku v jejích koncových bodech, stává se vzdálenost bodu od úsečky vzdáleností bodu od přímky (viz část 22.3.2 a 22.3.3). Leží-li mimo tento pás, je výsledkem menší ze vzdáleností ke koncovým bodům úsečky (viz část 22.2.1). K testu lze s výhodou využít parametru t_I ze vztahů (22.17) a (22.23).

22.3.5 Poloha bodu vůči přímce a úsečce

Polohou bodu vůči přímce a úsečce rozumíme, zda se bod od ní nachází nalevo, napravo nebo na ní leží. Úloha o poloze nalevo či napravo má smysl pouze v rovině, neboť v prostoru tato relace samostatně neexistuje a je ji nutno určovat vzhledem k orientované rovině.

Poloha bodu P vůči přímce či úsečce AB se dá vypočítat dle znaménka čitatele vzorce (22.18), tj.:

$$(b_0 - a_0)(p_1 - a_1) - (b_1 - a_1)(p_0 - a_0) \begin{cases} < 0 & \text{bod leží napravo od úsečky } AB \\ = 0 & \text{bod leží na úsečce } AB \\ > 0 & \text{bod leží nalevo od úsečky } AB. \end{cases} \tag{22.26}$$

Bod P leží na úsečce, pokud po dosazení jeho souřadnic do parametrické rovnice přímky (22.15) vyjde pro všechny souřadnice stejný parametr $t \in \langle 0, 1 \rangle$.

22.3.6 Poloha bodu vůči kružnici a kouli

Test spočívá v dosazení souřadnic bodu do rovnice kružnice či koule. Pro kružnici se středem $S = [s_0, s_1]$ a poloměrem r v kartézských souřadnicích a pro daný bod $P = [p_0, p_1]$ platí:

$$(p_0 - s_0)^2 + (p_1 - s_1)^2 - r^2 \begin{cases} < 0 & \text{bod leží uvnitř kružnice} \\ = 0 & \text{bod leží na kružnici} \\ > 0 & \text{bod leží vně.} \end{cases}$$

Pro kouli se středem $S = [s_0, s_1, s_2]$ a poloměrem r v kartézských souřadnicích a pro daný bod $P = [p_0, p_1, p_2]$ platí:

$$(p_0 - s_0)^2 + (p_1 - s_1)^2 + (p_2 - s_2)^2 - r^2 \begin{cases} < 0 & \text{bod leží uvnitř koule} \\ = 0 & \text{bod leží na povrchu koule} \\ > 0 & \text{bod leží vně.} \end{cases}$$

22.3.7 Vzdálenost bodu od roviny

Vzdálenost bodu $P = [p_0, p_1, p_2]$ od roviny zadané *obecnou rovnicí* $ax + by + cz + e = 0$, kde a, b, c a e jsou reálné konstanty a alespoň jedna z a, b nebo $c \neq 0$, se spočítá dosazením souřadnic bodu do levé strany obecné rovnice roviny

$$d = \left| \frac{ap_0 + bp_1 + cp_2 + e}{\sqrt{a^2 + b^2 + c^2}} \right| . \tag{22.27}$$

Po vydělení obecné rovnice roviny členem $\sqrt{a^2 + b^2 + c^2}$ získáme *normálový tvar rovnice roviny* $x \cos\alpha + y \cos\beta + z \cos\gamma - e = 0$, kde vektor $(\cos\alpha, \cos\beta, \cos\gamma)$ je jednotkovým normálovým vektorem, tj. vektorem k rovině kolmým, člen e je roven vzdálenosti roviny od počátku a α, β a γ jsou velikosti orientovaných úhlů, které svírá normála roviny (orientovaná od počátku souřadnic k rovině) s osami souřadnic. Pro $e = 0$ není orientace normály jednoznačně určena. Vzdálenost bodu $P = [p_0, p_1, p_2]$ od roviny v normálovém tvaru vypočteme dosazením:

$$d = |p_0 \cos\alpha + p_1 \cos\beta + p_2 \cos\gamma - e| . \tag{22.28}$$

Rovnice roviny kolmé k vektoru $\vec{a} = (a_0, a_1, a_2)$ procházející bodem $P = [p_0, p_1, p_2]$ je

$$a_0(x - p_0) + a_1(y - p_1) + a_2(z - p_2) = 0 \tag{22.29}$$

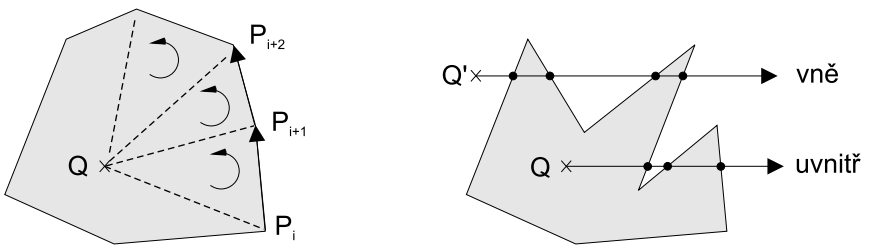
a rovnice roviny zadané trojicí bodů $A = [a_0, a_1, a_2]$, $B = [b_0, b_1, b_2]$ a $C = [c_0, c_1, c_2]$, které jsou vzájemně různé a neleží na přímce, lze zapsat pomocí determinantu ve tvaru

$$\begin{vmatrix} x & y & z & 1 \\ a_0 & a_1 & a_2 & 1 \\ b_0 & b_1 & b_2 & 1 \\ c_0 & c_1 & c_2 & 1 \end{vmatrix} = 0 . \tag{22.30}$$

22.3.8 Poloha bodu vůči mnohoúhelníku (polygonu)

K testování, zda je daný bod Q vnitřním bodem rovinného polygonu se používají dva přístupy ukázané na obrázku 22.4.

Pro *konvexní polygon* (obr. 22.4 vlevo) lze přímo využít jeho definice, tj. faktu, že vznikne jako průnik polorovin definovaných shodně nalevo (napravo) od jeho orientovaných hran. Zvolíme–li shodnou orientaci vlevo, pak vnitřní bod musí pro všechny hrany ležet v levých polorovinách vymezených hranami. Použijeme test velikosti orientovaného obsahu trojúhelníka $P_iP_{i+1}Q$, který musí být pro všechny vrcholy P_i vždy $<= 0$. V rovnici vektorového součinu (22.11) v části 22.2.4 tedy za body A, B, C dosazujeme P_i, P_{i+1} a Q.



Obrázek 22.4: Zjišťování polohy bodu vůči rovinnému mnohoúhelníku

Pro *nekonvexní polygon* se používá následující postup. Z testovaného bodu Q vyšleme polopřímku (paprsek) libovolným směrem a počítáme, kolikrát protne hranici mnohoúhelníka. Protne-li hranici v lichém počtu případů, je bod uvnitř, v sudém počtu je vně mnohoúhelníka (obr. 22.4 vpravo).

Jednoduchý algoritmus je komplikován řadou mezních případů, neboť z testu musíme vyloučit hrany, které (celé) leží na polopřímce a část hran, které mají na polopřímce koncový bod. Řešení těchto případů je součástí v praxi používaných algoritmů řádkového a paritního vyplňování oblastí zadaných geometrickou hranicí, viz část 3.3.1.

22.4 Přímka a paprsek

V této části se budeme zabývat určováním polohy paprsku (polopřímky) vůči geometrickým primitivům, a to jak rovinným, tak prostorovým. Při řešení těchto úloh je praktické používat popis přímky v prostoru parametrickou rovnicí (22.15), neboť umožňuje setřídit body na (polo)přímce podle vrůstající hodnoty parametru, kterým je průsečík určen.

22.4.1 Průsečík paprsku a přímky v rovině

Průsečík $P[p_0, p_1]$ paprsku ${}^1p$ a přímky ${}^2p$ v rovině se vypočítá jako průsečík dvojice přímek. Jsou-li přímky v zadány parametrickými rovnicemi ${}^1p = A + \vec{q}r$ a ${}^2p = B + \vec{s}t$, spočítá se jejich průsečík, položíme-li sobě rovny jejich pravé strany. Pro paprsek musí navíc platit, že parametr $r >= 0$.

22.4.2 Odchylka paprsku a přímky v prostoru

Odchylku paprsku a *orientované přímky*, lze chápat jako odchylku dvojice polopřímek zadaných směrovými vektory $\vec{q}$ a $\vec{s}$. Spočítá se stejně jako odchylka dvojice vektorů (22.6), tj.

$$(\widehat{\vec{q}, \vec{s}}) = \arccos \frac{\vec{q}.\vec{s}}{|\vec{q}||\vec{s}|}. \tag{22.31}$$

Výsledkem je konvexní úhel $\varphi \in \langle 0, \pi \rangle$, $\varphi = 0$ pro souhlasně kolineární a π pro nesouhlasně kolineární vektory. Pokud bychom chápali paprsek a přímku jako *neorientované přímky*, je výsledkem ostrý úhel $\varphi \in \langle 0, \frac{\pi}{2} \rangle$, a ve vzorci (22.31) přibude v čitateli absolutní hodnota.

22.4.3 Vzdálenost dvou mimoběžek v prostoru

Vzdálenost dvou mimoběžek definovaných parametrickými rovnicemi ${}^1p = A + \vec{q}r$ a ${}^2p = B + \vec{s}t$ je

$$d = \frac{|\,(B - A).(\vec{q} \times \vec{s})\,|}{|\vec{q} \times \vec{s}|}. \tag{22.32}$$

Výraz ve jmenovateli je v případě mimoběžek $|\vec{q} \times \vec{s}| \neq 0$. Je-li čitatel $(B - A).(\vec{q} \times \vec{s}) = 0$, leží přímky v jedné rovině, protínají se a vzdálenost je nulová.

Průsečík různoběžek se spočítá, položíme-li sobě rovny pravé strany vektorových rovnic přímek ${}^1p$ a ${}^2p$. Hodnoty parametrů r a t lze sice získat z dvojice rovnic pro libovolné dvě souřadnice, tyto parametry však musí vyhovovat i třetí rovnici, jinak se úsečky neprotínají.

22.4.4 Poloha paprsku vůči rovině

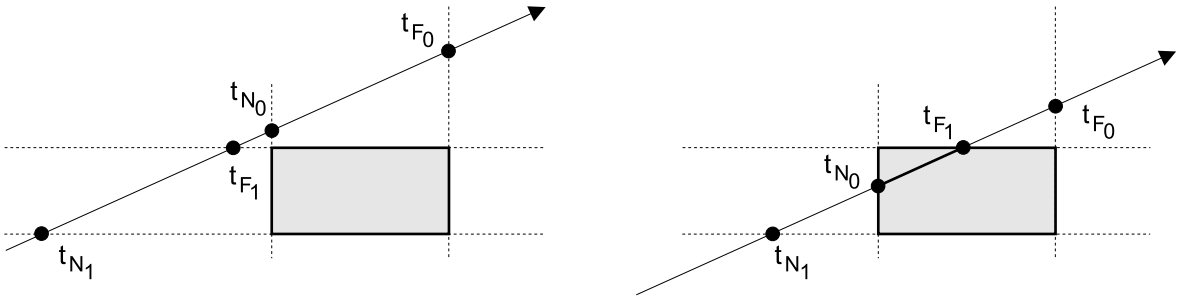
Paprsek může být s rovinou buď rovnoběžný, nebo ji protíná. Průsečík paprsku vycházejícího z bodu A ve směru $\vec{q}$, tj. definovaného parametrickou rovnicí (22.15), s rovinou zadanou obecnou rovnicí $ax + by + cz + e = 0$ se spočítá prostým dosazením souřadnic paprsku do rovnice roviny. Úpravou získáme vztah pro parametr t:

$$t = \frac{-(aa_o + ba_1 + ca_2 + e)}{aq_0 + bq_1 + cq_2}. \tag{22.33}$$

Pokud je jmenovatel roven nule a zároveň čitatel nenulový, je paprsek s rovinou rovnoběžný. Pro nulový čitatel i jmenovatel leží paprsek v rovině.

22.4.5 Průsečík paprsku s osově orientovaným kvádrem

Osově orientovaný kvádr (*bounding box*) tvoří často obálku prostorových objektů. Bližší z jeho možných dvou průsečíků s paprskem se určí dle algoritmu 22.1 [Watt92]. Situaci ilustruje obrázek 22.5.



Obrázek 22.5: Průsečík paprsku s ohraničujícím kvádrem (bboxem)

1. $t_N = 0,\ t_F = \infty$
2. Pro každý pár ohraničujících rovin s indexem souřadnice i ($i = 0, 1, 2$) dělej:
   (a) Vypočítej parametr t_{Ni} pro průsečík s bližší rovinou a parametr t_{Fi} pro průsečík se vzdálenější rovinou
   (b) $t_N = \max(t_N, t_{Ni})$ a $t_F = \min(t_F, t_{Fi})$
   (c) Pokud je hodnota $t_N > t_F$, paprsek kvádr míjí a cyklus je ukončen
3. Pokud nebyl algoritmus v kroku (c) ukončen, je průsečík určen hodnotou t_N (je-li počátek paprsku uvnitř kvádru, pak t_F).

Algoritmus 22.1: Algoritmus výpočtu průsečíku s ohraničujícím kvádrem

Ohraničující kvádr má souřadnice mezních vrcholů $[b_0, b_1, b_2]$ a $[v_0, v_1, v_2]$. Paprsek s počátkem v bodě $A = [a_0, a_1, a_2]$ je dán parametrickou rovnicí $R = A + t\vec{q}$. Výpočet parametrů t_{Ni}

a t_{Fi} pro roviny kolmé na i-tou souřadnicovou osu probíhá podle následujících vztahů:

$$t_{Ni} = \frac{b_i - a_i}{q_i}, \qquad t_{Fi} = \frac{v_i - a_i}{q_i}, \tag{22.34}$$

kde t_{Ni} je parametr průsečíku paprsku s bližší rovinou a t_{Fi} parametr průsečíku paprsku se vzdálenější rovinou.

22.4.6 Průsečík paprsku a mnohoúhelníka

Průsečík paprsku s mnohoúhelníkem se převádí na vyřešení těchto úloh:

1. Nalezení rovnice roviny, v níž mnohoúhelník leží (vzorec 22.30).
2. Nalezení průsečíku paprsku s rovinou (část 22.4.4).
3. Test, zda průsečík leží v mnohoúhelníku (část 22.3.8)

Je výhodné mnohoúhelník položit (otočit) do roviny xy a třetí úlohu řešit jen ve dvojrozměrném prostoru. Problémem, se kterým se zde setkáme, je situace, kdy vrcholy mnohoúhelníku neleží v rovině (např. následkem numerické nepřesnosti výpočtu při transformacích). Klasickým řešením je rozdělení mnohoúhelníku na trojúhelníky a testování průsečíku jako vnitřního bodu jednotlivých trojúhelníků v rovině.

22.4.7 Průsečík paprsku s kulovou plochou

Je dán paprsek R s počátkem $A[a_0, a_1, a_2]$ a směrovým vektorem $\vec{q}$ a koule se středem $S[s_0, s_1, s_2]$ a poloměrem r. Při zjišťování jejich průsečíku mohou nastat tyto případy:

a) paprsek kouli mine (žádný průsečík),
b) paprsek se kulové plochy dotkne (jeden průsečík),
c) paprsek kouli protne (dva průsečíky).

V případě b) nebo c) je nutno rozlišit, zda nalezený průsečík leží v kladném směru paprsku. V případě c) je výsledkem průsečík, který je blíže k pozorovateli ve směru paprsku.

Jednou z možností, jak tuto úlohu řešit, je dosazení souřadnice paprsku z jeho parametrické rovnice $R(t) = A + t\vec{q}$ do rovnice kulové plochy

$$(a_0 + t.q_0 - s_0)^2 + (a_1 + t.q_1 - s_1)^2 + (a_2 + t.q_2 - s_2)^2 - r^2 = 0\,.$$

Získáme kvadratickou rovnici, která má podle hodnoty diskriminantu 0,1 nebo 2 řešení pro parametr t. Výhodnější je však řešení založit na rozboru geometrické situace, uvedené na obrázku **??**.

Předpokládáme, že vektor $\vec{q}$ je jednotkový. Délku t_0 vektoru $(T - A)$ získáme pomocí průmětu vektoru $(S - A)$ na jednotkový vektor $\vec{q}$:

$$t_0 = |(T - A)| = |(S - A).\vec{q}|$$

Podle Pythagorovy věty je vzdálenost $d = (S - A).(S - A) - t_0^2$ a vzdálenost od bodu T k průsečíkům paprsku s koulí $t_d^2 = r^2 - d^2$. Výsledkem je jednoduché zhodnocení situace:

- Pro $t_d^2 = 0$ je paprsek tečnou koule v bodě $A + t_0\vec{q}$;
- pro $t_d^2 > 0$ existují dva průsečíky v bodech $A + (t_0 \pm t_d)\vec{q}$;
- pro $t_d^2 < 0$ paprsek kouli míjí.

Připomeňme, že průsečík určený hodnotou $t <= 0$ vidíme zevnitř, resp. z povrchu kulové plochy. Normálový vektor $\vec{n}$ je jednotkový vektor ve směru ze středu kulové plochy do nalezeného bodu P.

$$\vec{n} = \frac{P - S}{r}.$$

22.5 Užitečné drobnosti

22.5.1 Plocha mnohoúhelníka

Obsah mnohoúhelníka s vrcholy $A[a_0, a_1], B[b_0, b_1], \ldots, Z[z_0, z_1]$ se vyjádří jako

$$P = \frac{1}{2}\left|\begin{vmatrix} a_0 & b_0 \\ a_1 & b_1 \end{vmatrix} + \begin{vmatrix} b_0 & c_0 \\ b_1 & c_1 \end{vmatrix} + \ldots + \begin{vmatrix} z_0 & a_0 \\ z_1 & a_1 \end{vmatrix}\right|$$

Speciálně pro trojúhelník jako $\frac{1}{2}|P|$ ve vzorci (22.11) či (22.12).

22.5.2 Gaussovo rozložení

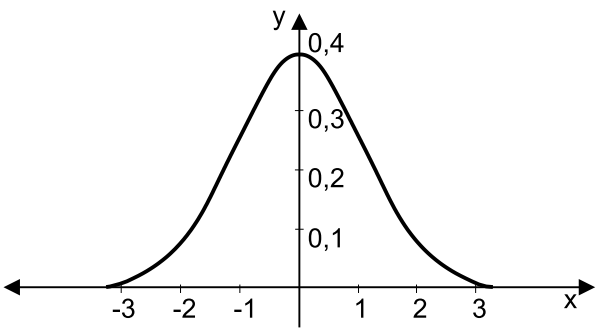
Hustota pravděpodobnosti $f(x)$ Gaussova rozložení neboli normálního rozložení (viz obrázek 22.7) odpovídá vztahu

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{22.35}$$

kde μ je střední hodnota tohoto rozložení a σ^2 je rozptyl. Normální rozložení označujeme většinou symbolem $N(\mu, \sigma^2)$.

Normované Gaussovo rozložení $N(0, 1)$ získáme dosazením do (22.35) za $\mu = 0$ a $\sigma^2 = 1$. Jeho hustota pravděpodobnosti $f(x)$ odpovídá vztahu

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}, \tag{22.36}$$



Obrázek 22.7: Gaussovo normální rozložení

Normované Gaussovo rozložení $N(0, 1)$ můžeme získat součtem náhodných čísel g_i, které mají rovnoměrné rozložení pravděpodobnosti

$$f(x) = -0.5 + \frac{1}{n} \sum_{i=1}^{n} g_i.$$

Čísla g_i poskytuje například generátor pseudonáhodných čísel `rand()` v jazyce C.

22.6 Interpolace

Interpolaci používáme v případě, že potřebujeme znát hodnotu v místě, kde není známa. Proto se ji snažíme odhadnout na základě známých hodnot v jejím bezprostředním okolí.

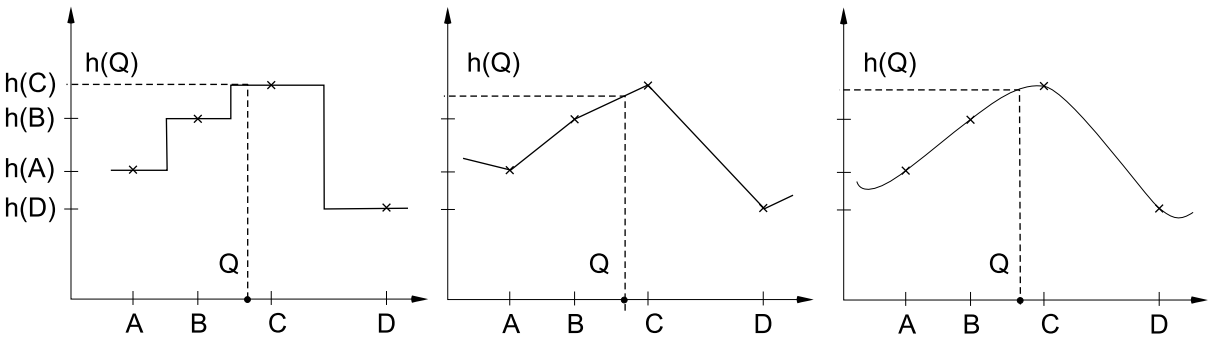
Vstupem jsou body $A, B, C, D, \ldots$ a v nich naměřené hodnoty $h(A), h(B), h(C), h(D), \ldots$. Dále je dán bod Q, jehož hodnotu $h(Q)$ chceme znát. Výstupem interpolace je odhad hodnoty $h(Q)$ v požadovaném místě, tj. v bodě Q.

Seznámíme se s interpolací hodnotou nejbližšího souseda, lineární, bilineární a trilineární interpolací a s příkladem specializované kubické interpolace. Informace o jiných interpolačních metodách lze nalézt např. v [Niel94].

22.6.1 Interpolace hodnotou nejbližšího souseda

Nejjednodušším případem interpolace je interpolace nultého řádu – interpolace hodnotou, která je k požadované nejblíže. Jako odhad hodnoty h(Q) vezmeme hodnotu bodu $X \in \{A, B, C, D\}$, pro nějž je $d(Q, X)$ minimální. Na obrázku 22.8 vlevo je nejbližším sousedem bodu Q bod C, interpolovaná hodnota je tedy $h(C)$.

Je zřejmé, že takový odhad je velmi hrubý. Lepší metodou by bylo proložení hodnot nějakou křivkou (plochou, objemem, ... podle rozměru úlohy) a odhadovat hodnotu mezi vzorky podle hodnoty proložené křivky. Nejčastěji se používá prokládání po částech lineární interpolací.



Obrázek 22.8: Interpolace hodnotou nejbližšího souseda (vlevo), lineární (uprostřed) a kubická interpolace (vpravo)

22.6.2 Lineární interpolace

Princip *lineární interpolace* hodnoty $h(Q)$ bodu Q pomocí hodnot $h(A), h(B)$ v sousedních bodech A, B je patrný z obrázku 22.8 uprostřed. Vyjádří se dle vztahu (22.37):

$$h(Q) = h(A) + (h(B) - h(A))\frac{|Q - A|}{|B - A|}. \tag{22.37}$$

Obdobně bychom postupovali při opačné úloze – nalezení souřadnice vzorku Q se zadanou hodnotou $h(Q)$

$$Q = A \ + (B - A)\frac{h(Q) - h(A)}{h(B) - h(A)} \ , \ h(B) \neq h(A). \tag{22.38}$$

22.6.3 Kubická interpolace

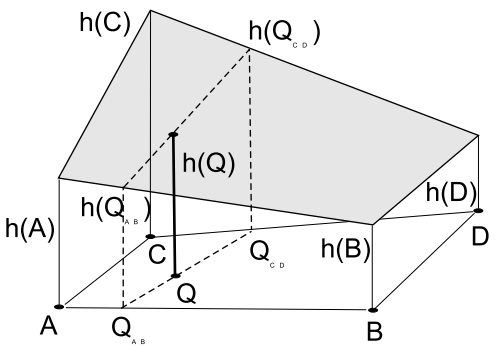
Kubická interpolace je nejjednoduší interpolační metoda, která s použitím kubických polynomů zajišťuje hladký průběh interpolovaných hodnot v určitém rozsahu. Pro určení kubického polynomu potřebujeme zadat čtyři podmínky, např. čtyři body, kterými interpolační křivka prochází. Dalšími podmínkami může být např. požadavek na to, aby délky křivkových úseků mezi jednotlivými body odpovídaly euklidovským vzdálenostem mezi těmito body, nebo aby parametrická křivka procházela body v pevně stanovených hodnotách parametru. Možností je mnoho, zde uvedeme příklad jednoduché kubické interpolace. Hledáme křivku – kubický polynom – která v bodech A, B, C, D nabývá hodnot $h(A), h(B), h(C), h(D)$, obrázek 22.8 vpravo.

Pro úsek mezi body B, C můžeme použít interpolaci pomocí B-spline. Nejprve vypočteme parametr $t = (Q - B)/(C - B); t \in < 0..1 >$, který určuje relativní pozici bodu Q mezi body B, C. Interpolovaná hodnota je vypočtena jako součet hodnot $h(A), h(B), h(C), h(D)$ násobených polynomy $C_0(t)$, $C_1(t)$,$C_2(t)$, $C_3(t)$.

$$\begin{aligned} C_0(t) &= (1 - t^3) \\ C_1(t) &= 3t^3 - 6t^2 + 4 \\ C_2(t) &= -3t^3 + 3t^2 + 3t + 1 \\ C_3(t) &= t^3 \\ h(Q) &= C_0(t)h(A) + C_1(t)h(B) + C_2(t)h(C) + C_3(t)h(D) \end{aligned} \tag{22.39}$$

22.6.4 Bilineární interpolace

Bilineární interpolace nahrazuje izolované body (vzorky) ve dvojrozměrném či trojrozměrném prostoru po částech lomenou plochou. Interpolovaná hodnota v bodě, který leží mezi zadanými body, se určuje interpolací čtyř nejbližších sousedů, jak je ukázáno na obrázku 22.9. Souřadnice bodů A, B, C, D, stejně jako hodnoty funkce $h(A)$, $h(B)$, $h(C)$, $h(D)$ v těchto bodech jsou známy. Hledáme hodnotu v bodě Q, která se spočítá z hodnot sousedních bodů podle vztahu 22.40. Nejprve spočteme lineární interpolací s využitím (22.37) hodnoty v bodech Q_{AB}

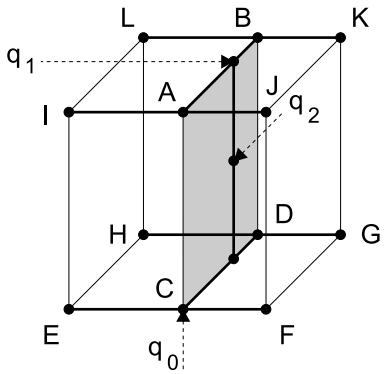


Obrázek 22.9: Bilineární interpolace

a Q_{CD}, poté s využitím stejného vztahu určíme pomocí těchto hodnot hledanou hodnotu $h(Q)$.

$$\begin{aligned} h(Q_{AB}) &= h(A) + (h(B) - h(A))\frac{|Q_{AB} - A|}{|B - A|} \\ h(Q_{CD}) &= h(C) + (h(D) - h(C))\frac{|Q_{CD} - C|}{|D - C|} \\ h(Q) &= h(Q_{AB}) + (h(Q_{CD}) - h(Q_{AB}))\frac{|Q - Q_{AB}|}{|Q_{CD} - Q_{AB}|} \end{aligned} \tag{22.40}$$

Na obrázku 22.9 tvoří zadané body rohy lichoběžníku, obecně mohou tyto body tvořit vrcholy libovolného konvexního mnohoúhelníku. Určení bodů Q_{AB} a Q_{CD}, jejichž spojnice prochází bodem Q, záleží na kontrétní úloze.



Obrázek 22.10: Trilineární interpolace

V počítačové grafice často prováděná úloha rasterizace mnohoúhelníku se provádí po vodorovných řádcích a libovolná hodnota pixelu (barva, hloubka – souřadnice z) se získá interpolací. Mnohoúhelník je rozdělen na vodorovné pásy omezené hraničními úsečkami a v jejich prostoru probíhá interpolace.

Bilineární interpolace je rychlá, neboť se pro výpočet každého bodu používají pouze čtyři body z jeho okolí a z tohoto důvodu bývá implementována i v technickém vybavení počítačů. Používá se zejména pro stínování a při práci s texturami (viz část 10.7).

Trilineární interpolace

Za *trilineární interpolaci* označujeme postupné, trojí uplatnění lineární interpolace. Setkáme se s ní obvykle u prostorových pravoúhlých mřížek, kde čtveřice interpolovaných vzorků vždy leží v jedné rovině.

Jsou známy souřadnice bodů E, F, G, H, I, J, K, L v pravidelné prostorové mřížce a hodnoty funkce $f(i, j)$ v těchto bodech. Hledáme hodnotu $f(Q)$ v bodě $Q[q_0, q_1, q_2]$. Čtveřicí lineárních interpolací jedné souřadnice (např. x) vypočteme hodnoty v bodech A, B, C a D z hodnot zadaných bodů (viz obr. 22.10). Příslušná souřadnice je vždy shodná se souřadnicí bodu Q (zde q_0). Bilineární interpolací (viz předchozí odstavec 22.6.4) pak vypočítáme hodnotu v bodě Q.

22.6.5 Interpolace vyššího řádu

Zatímco v předchozím případě byla úloha interpolace zadána jednoznačně, při interpolaci množiny bodů křivkami, či plochami vyššího stupně musíme zadávat kromě polohy bodů ještě nějaké další podmínky, například tečné vektory na okrajích.

Pravděpodobně nejlépe je tato teorie zpracována v případě převzorkování diskrétního obrazu (viz odstavec 4.3.1). Zde bývají interpolační metody nejčastěji popsány pomocí konvoluce. Při interpolaci množiny izolovaných bodů za účelem získání nějaké hladké plochy, obvykle používáme B-spline plochy (viz odstavec 5.5).

22.7 Diskrétní Fourierova transformace

Obraz můžeme považovat za diskrétní a periodický signál. Zpracování obrazu – např. při změně jeho frekvenčních charakteristik nebo při konvoluci – nelze řešit pomocí spojité Fourierovy transformace. Místo toho použijeme diskrétní Fourierovu transformaci, která se označuje jako DFT (*Discrete Fourier Transform*). Předpokládejme, že signál (obraz) I je dán jako posloupnost shodně vzdálených vzorků $I = I_0, I_1, \ldots, I_{N-1}, \ldots$, které se dále opakují s periodou N ($I_N = I_0, I_{N+1} = I_1, \ldots$). DFT posloupnosti vzorků I (komplexních čísel) označujeme F_n a vypočteme ji pomocí

$$\begin{aligned} F_n &= \sum_{k=0}^{N-1} I_k W_N^{nk} \qquad 0 \le n \le N-1 \\ W_N &= e^{-i2\pi/N} = cos(-2\pi/N) + i\, sin(-2\pi/N) \end{aligned}$$

W_N^k pro $k = 0 \ldots N-1$ se nazývají *N-kořeny jednotky*. Název pochází ze skutečnosti, že pro tato čísla platí v komplexní aritmetice $(W_N^k)^N = 1$ pro všechna k. Jsou to vrcholy pravidelného polygonu vepsaného do jednotkové kružnice v komplexní rovině, s jedním vrcholem v $(1 + i0)$.

Sekvence $F_0, F_1, \ldots$ je diskrétní Fourierovou transformací $I_0, I_1, \ldots$, obě duální reprezentace diskrétního signálu obsahují N komplexních čísel. Inverzní transformaci IDFT vypočteme ze vztahu

$$I_n = \frac{1}{N} \sum_{k=0}^{N-1} F_k W_N^{-nk} \qquad 0 \le n \le N-1$$

Výpočet IDFT je až na měřítko a znaménka koeficitů shodný s výpočtem DFT. Přímý výpočet DFT/IDFT je časově náročný. Podle definice vyžaduje N^2 komplexních násobení a $N.(N-1)$ komplexních sčítání ($4N^2$ reálných násobení a $4N^2$ reálných sčítání). Pro praktické použití je tento výpočet příliš nákladný. Naštěstí existuje možnost výpočet značně urychlit.

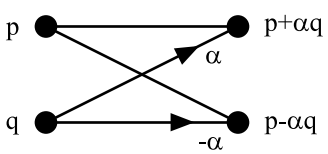
22.7.1 Rekurzivní rozklad DFT

V roce 1942 odvodili Danielson a Lanczos rekurzivní vztah pro řešení DFT [Dani42]. Dokázali, že DFT délky N může být nahrazena součtem dvou DFT o délkách $N/2$, kde N je celé číslo, které je mocninou dvou. Z původní posloupnosti N bodů transformuje první DFT vzorky s lichým indexem, druhá vzorky se sudým indexem. Rozklad vychází z postupné úpravy původního vztahu pro DFT.

$$\begin{aligned} F_n &= \sum_{k=0}^{N-1} I_k e^{-i2\pi nk\ /N} \\ &= \sum_{k=0}^{(N/2)-1} I_{2k} e^{-i2\pi n(2k)/N} + \sum_{k=0}^{(N/2)-1} I_{2k+1} e^{-i2\pi n(2k+1)/N} \\ &= \sum_{k=0}^{(N/2)-1} I_{2k} e^{-i2\pi nk\ /(N/2)} + W^n \sum_{k=0}^{(N/2)-1} I_{2k+1} e^{-i2\pi nk\ /(N/2)} \\ &= F_n^e + W^n F_n^o \end{aligned} \tag{22.41}$$

F_n^e označuje n-tý člen DFT délky $N/2$ posloupnosti složené ze sudých členů a F_n^o obdobně označují DFT posloupnosti lichých členů. Z definice DFT vyplývá, že $F_{n+N} = F_n$, tj. F_n je také periodická s periodou N. Z toho plyne, že

$$\begin{aligned} F^e_{n+N/2} &= F^e_n \\ F^o_{n+N/2} &= F^o_n \\ W^{n+N/2} &= -W^n \qquad 0 \le n \le N/2. \end{aligned}$$



Obrázek 22.11: Operace „motýlek“

Po dosazení dostaneme tzv. „motýlkové“ uspořádání výpočtu vyjádřené rovnicemi

$$\begin{aligned} F_n &= F_n^e + W^n F_n^o \qquad 0 \le n \le N/2 \\ F_{n+N/2} &= F_n^e - W^n F_n^o \qquad 0 \le n \le N/2. \end{aligned}$$

Každý motýlek vezme dvě komplexní čísla p a q a vypočte z nich jiná dvě čísla $p + \alpha q$ a $p - \alpha q$, kde α je komplexní číslo. Na obrázku 22.11 je diagram motýlkové operace.

22.7.2 Rychlá Fourierova transformace

Na základě rozkladu diskrétní Fourierovy transformace na jednodušší kroky, které obsahují společné podvýrazy, byl navržen algoritmus FFT (*Fast Fourier Transform*). Tento algoritmus, který vypočte DFT s použitím $O(N \log N)$ násobení a sčítání, má mnoho variant. Rekurzivní FFT algoritmus, který přímo realizuje postup podle rovnice (22.41), je zkráceně zapsán jako algoritmus 22.2.

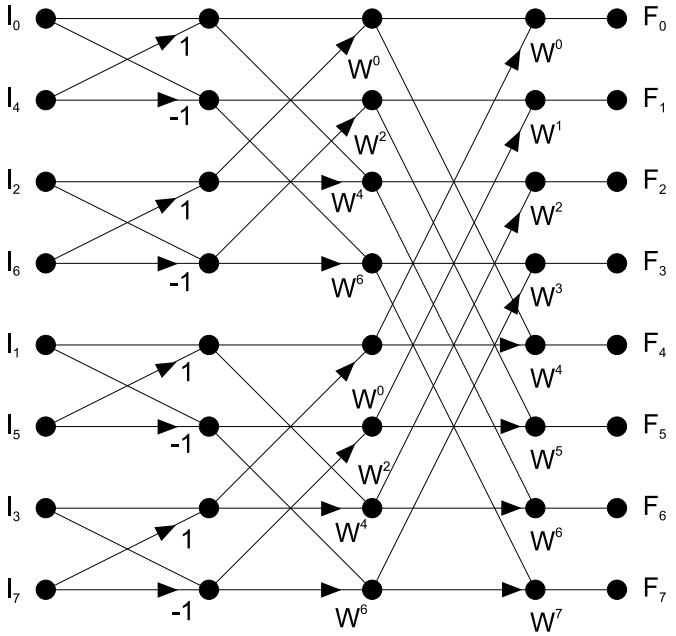
Procedura $FFT(N, f)$
/* N – počet hodnot v poli f */
Pokud $N = 2$
  pak
    Nahraď f_0 hodnotou $f_0 + f_1$ a f_1 hodnotou $f_0 - f_1$
  jinak
    Vytvoř g – seznam všech bodů z f, které mají sudý index;
        h – seznam všech bodů z f, které mají lichý index.
    $FFT(N/2,\ g)$
    $FFT(N/2,\ h)$
    Nahraď f_n hodnotou $g_n + W^n.h_n$ pro $n = 0, \ldots, N-1$.

Algoritmus 22.2: Rekurzivní FFT

Pro praktické implementace se však více hodí nerekurzivní vyjádření FFT. Jako příklad uvedeme algoritmus Cooley-Tukey, nazývaný *decimace v čase*. Na obr.22.12 je diagram 8-bodové FFT. Výpočet rozkládá DFT na $\log_2 N$ stupňů, každý z nich je složen z $N/2$ *motýlkových* výpočtů. Pro výpočet transformovaných vzorků v přirozeném pořadí $F_0, F_1, F_2, F_3, F_4, F_5, F_6, F_7$ jsou ze vstupu odebírány vzorky po dvojících $(I_0, I_4), (I_2, I_6), (I_1, I_5), (I_3, I_7)$. Uvedená změna pořadí na vstupu vypadá na první pohled složitě, indexy však v sobě skrývají jednoduchou zákonitost. Následující tabulka obsahuje index j určující pořadí výstupního vzorku a index k určující pořadí vstupního vzorku.

pořadí na výstupu – j	0	1	2	3	4	5	6	7
pořadí na vstupu – k	0	4	2	6	1	5	3	7
j binárně	000	001	010	011	100	101	110	111
k binárně	000	100	010	110	001	101	011	111

Pokud indexy j a k zapíšeme binárně (poslední dva řádky tabulky), zjistíme, že každé k je *bitově převrácené j* (první bit zaměněn s posledním, druhý s předposledním atd.). Ukazuje se,



Obrázek 22.12: 8-bodová FFT

že FFT algoritmus se zjednoduší, jestliže vstupní pole je přeuspořádáno do bitově převráceného pořadí. Výpočet je zahájen změnou pořadí ve vstupním poli hodnot, není dodrženo přirozené pořadí v čase (odtud název algoritmu). V jednotlivých krocích výpočtu, který je naznačen na obrázku 22.12, jsou motýlkové výpočty aplikovány vždy na dvojice hodnot z předchozí úrovně. Po ukončení výpočtu jsou transformované hodnoty uspořádány v přirozeném pořadí, viz obrázek 22.12. Podobně lze organizovat výpočet, nazývaný FFT *decimace ve frekvenci*. Jedná se o algoritmus Cooley-Sande, který ponechává vstupní hodnoty v daném pořadí, na výstup pak předává hodnoty v pozměněném pořadí (není dodrženo pořadí jednotlivých frekvenčních členů podle vzrůstající frekvence). Nerekurzivní výpočty s konstantním počtem kroků lze efektivně paralelizovat a realizovat pomocí specializovaného HW.

22.7.3 Použití algoritmu FFT

Výpočet DFT posloupnosti N vzorků podle definice vyžaduje N^2 komplexních násobení a sčítání, což představuje $4N^2$ reálných násobení a $4N^2$ reálných sčítání. Základním výpočetním

krokem algoritmu FFT je motýlek. Vyžaduje jedno komplexní násobení a dvě komplexní sčítání, což představuje 4 reálná násobení a 6 reálných sčítání. Každý stupeň výpočtu obsahuje $N/2$ motýlků a celkem je $\log_2 N$ stupňů, což představuje okolo $4 \cdot N/2 \cdot \log_2 N = 2N \log_2 N$ reálných násobení a $3N \log_2 N$ reálných sčítání. Ze srovnání je zřejmé, že algoritmus FFT je *mnohem* rychlejší pro velká N. Pro N, které není mocninou dvou, existují optimalizované algoritmy. Obecně jsou tyto algoritmy rychlejší, pokud N má hodně prvočinitelů. Další podrobnosti lze nalézt např. v [Brig74].

Algoritmus FFT se používá pro rychlou konvoluci při filtrování (nejen) obrazové informace pomocí lineárních filtrů. Konvoluce diskrétních signálů f a g je dána součtem

$$h_j = \sum_{k=-\infty}^{\infty} f_k g_{j-k}.$$

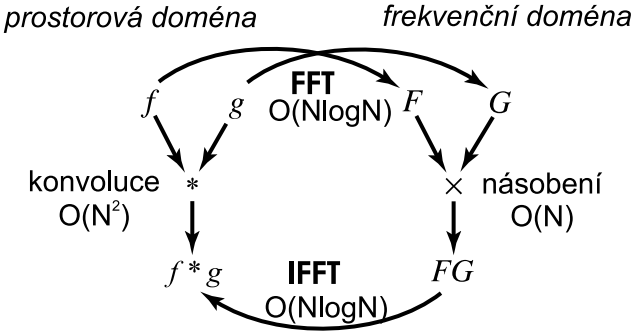
f můžeme považovat za signál a g za filtr, h je výsledek konvoluce, tj. signál po aplikaci filtru. Při práci s konečnými posloupnostmi se definice konvoluce zjednoduší za předpokladu, že f a g mají stejnou délku N a signály chápeme periodicky, tj. f a g se cyklicky opakují. Pak dostaneme *cyklickou konvoluci*:

$$h_j = \sum_{k=0}^{N-1} f_k \cdot g_{(j-k) \bmod N} \qquad j = 0 \ldots N-1$$

Z konvolučního teorému plyne, že Fourierova transformace konvoluce dvou signálů je součinem jejich Fourierových transformací: $f \star g \leftrightarrow FG$. Odpovídající teorém pro diskrétní signály uvádí, že DFT cyklické konvoluce dvou signálů je součinem DFT těchto signálů. Výpočet konvoluce podle definice vyžaduje N^2 (reálných) násobení a sčítání, což je velmi nákladné. S využitím algoritmu FFT však stejný výpočet realizujeme rychleji. Nejprve vypočteme FFT posloupnosti f a FFT posloupnosti g, poté vynásobíme transformované posloupnosti bod po bodu a výsledek transformujeme zpět pomocí algoritmu IFFT. Pak obdržíme výsledek shodný s přímou konvolucí, a to pomocí tzv. *Fourierovy konvoluce* realizované ve frekvenční oblasti násobením Fourierových obrazů obou signálů.

Při použití FFT algoritmu počítáme pro dvě DFT a jednu inverzní DFT celkem $6N \log_2 N$ reálných násobení pro transformace; násobení transformovaných posloupností ve frekvenční doméně má zanedbatelnou cenu $4N$ reálných násobení. Přímý algoritmus výpočtu konvoluce naproti tomu vyžadoval N^2 reálných násobení. Fourierova konvoluce jednoznačně vítězí pro velká N. Pokud nechceme řešit cyklickou konvoluci, můžeme algoritmus modifikovat pro řešení standardní lineární konvoluce přidáním příslušných sekvencí nul.

Při výpočtu Fourierovy transformace obrazu využijeme s výhodou separabilitu 2-dimenzionální DFT na dvě 1-dimenzionální DFT. Výpočet provedeme ve 2 krocích:



Obrázek 22.13: Konvoluce a násobení pomocí algoritmů FFT a IFFT

- vypočteme DFT každé řádky obrazu a uložíme jako mezivýsledek.
- vypočteme DFT každého sloupce mezivýsledku a uložíme do výsledku.

Někdy je vhodné cyklicky posunout obraz tak, aby se pixel (0,0), který po transformaci obsahuje koeficient frekvenční složky $(\omega_x, \omega_y) = (0, 0)$, přemístil do středu obrazu. Názornější představu o charakteristice obrazu může také poskytnout zobrazení pixelových hodnot úměrně $log(velikost)$ pro každé komplexní číslo. U barevných obrazů se popsané transformace provádějí nezávisle pro každý z barevných kanálů [R,G,B]. Pro obraz o rozměrech $N \times N$ je cena DFT a IDFT přímo úměrná $N^2 \log N$. Přímý algoritmus má cenu úměrnou N^4.

Na závěr odstavce připomeneme, že diskrétní Fourierova transformace implementovaná jako FFT má velmi širokou oblast využití. Je základem frekvenční analýzy obrazu, ale používá se také při syntéze fraktálových textur a pro vytvoření obrazů s daným spektrem.

Literatura

[Aire90] J. M. Airey, J. H. Rohlf, and F. P. Brooks, Jr. Towards image realism with interactive update rates in complex virtual building environments. In *Proceedings of Symposium on Interactive 3D Graphics*, pages 41–50. ACM SIGGRAPH, 1990.

[Appe67] A. Appel. The Notion of Quantitative Invisibility and the Machine Rendering of Solids. In *Proceedings of the ACM National Conference*, volume 14, pages 387–393. ACM Inc., New York, N.Y., 1967.

[Arvo88] J. Arvo and D. Kirk. Modeling Plants with Environment-Sensitive Automata. In *Proceedings of Ausgraph'88*, pages 27–33, 1988.

[Arvo89] J. Arvo and D. Kirk. A Survey of Ray Tracing Acceleration Techniques. In Andrew S. Glassner ed.: *An Introduction to Ray Tracing*. Academic Press, San Diego, 1989.

[Arvo91] J. Arvo. *Graphics Gems II.* Academic Press, 1991.

[Aube00] A. Aubel, R. Boulic, and D. Thalmann. Real-time display of virtual humans: Levels of detail and impostors. *IEEE Transactions on Circuits and Systems for Video Technology, Special Issue on 3D Video Technology*, 10(2):207–217, 2000.

[Barr84] A. H. Barr. Global and local deformations of solid primitives. *Computer Graphics*, 18(3), 1984.

[Bart96] H. J. Bartsch. *Matematické vzorce.* Mladá fronta, 1996.

[Baum75] B. G. Baumgart. A Polyhedron Representation for Computer Vision. In *AFIPS Conf. Proc.*, volume 44, May 1975.

[Bene01] B. Beneš and R. Forsbach. Layered Data Structure for Visual Simulation of Terrain Erosion. In *IEEE Proceedings of the Spring Conference on Computer Graphics*, pages 80–86, 2001.

[Bene02] B. Beneš and E. Millán. Virtual Climbing Plants Competing for Space. In *IEEE Proceedings of the Computer Animation 2002*, pages 33–42. IEEE Computer Society, 2002.

[Bene03] B. Beneš, J. A. Cordóba, and J. M. Soto. Modeling Virtual Gardens by Autonomous Procedural Agents. In *IEEE Proceedings of the Theory and Practice of Computer Graphics 2003*, pages 73–85. IEEE Computer Society, 2003.

[Bene04] B. Beneš and T. Roa. Simulating Desert Scenery. *Journal of WSCG*, 11(I):21–29, 2004.

[Berg86] P. Bergeron. A General Version of Crow's Shadow Volumes. *IEEE Computer Graphics & Applications*, 6(9):17–28, September 1986.

[Bitt98] J. Bittner, V. Havran, and P. Slavík. Hierarchical visibility culling with occlusion trees. In *Proceedings of Computer Graphics International '98 (CGI'98)*, pages 207–219. IEEE, 1998.

[Bitt01] J. Bittner, P. Wonka, and M. Wimmer. Visibility preprocessing for urban scenes using line space subdivision. In *Proceedings of Pacific Graphics (PG'01)*, pages 276–284, Tokyo, Japan, 2001. IEEE Computer Society.

[Bitt02] J. Bittner. *Hierarchical Techniques for Visibility Computations*. PhD thesis, ČVUT v Praze, October 2002.

[Blin78] J. F. Blinn. Simulation of wrinkled surfaces. In *SIGGRAPH '78 Conference Proceedings*, pages 286–292. ACM Press, 1978.

[Blin82a] J. Blinn. A Generalization of Algebraic Surface Drawing. *ACM Transaction on Graphics*, 1:232, 1982.

[Blin82b] J. F. Blinn. Light Reflection Functions for Simulation of Clouds and Dusty Surfaces. *Computer Graphics*, 16(3):21–29, 1982.

[Blin88a] J. F. Blinn. Fractional Invisibility. *Computer Graphics & Applications*, November 1988.

[Blin88b] J. F. Blinn. Me and My (fake) Shadow. *IEEE Computer Graphics & Applications*, 8(1):82–86, January 1988.

[Bloo88] J. Bloomenthal. Polygonization of Implicit Surfaces. *Computer Aided Geometric Design*, 4(5):341–355, 1988.

[Bloo02] J. Bloomenthal. Medial-based vertex deformation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 147–151. ACM Press, 2002.

[Bour02] D. M. Bourg. *Physics for Game Developers*. O'Reilly & Associates, Inc., 2002.

[Brab02] S. Brabec, T. Annen, and H. P. Seidel. Shadow mapping for hemispherical and omnidirectional light sources. In *Advances in Modelling, Animation and Rendering, Proceedings of the Computer Graphics International Conference (CGI)*, 2002.

[Bres65] J. E. Bresenham. Algorithm for Computer Control of a Digital Plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[Bres77] J. E. Bresenham. A Linear Algorithm for Incremental Digital Display of Circular Arcs. *Communications of the ACM*, 20(2):100–106, 1977.

[Brig74] E. O. Brigham. *The Fast Fourier Transform*. Prentice-Hall, Englewood Cliffs, NJ, 1974.

[Brod92] K. W. Brodlie, R. A. Carpenter, R. A. Earnshow, J. R. Gallop, R. J. Hubbold, A. M. Mumford, C. D. Osland, and P. Quarendon, editors. *Scientific Visualisation, Techniques and Applications.* Springer-Verlag, 1992.

[Bron90] W. F. Bronsvoort. *Direct Display Algorithm for Solid Modelling.* Delft University Press, 1990.

[Chen01] B. Chen and M. X. Nguyen. POP: A Hybrid Point and Polygon Rendering System for Large Data. In *Proceedings of the conference on Visualization '01*, pages 45–52. IEEE Computer Society, 2001.

[Chin89] N. Chin and S. Feiner. Near real-time shadow generation using BSP trees. *Computer Graphics (Proceedings of SIGGRAPH '89)*, 23(3):99–106, 1989.

[Chri97] P. H. Christensen, D. Lischinski, E. J. Stollnitz, and D. H. Salesin. Clustering for Gossy Global Illumination. *ACM Transactions on Graphics*, 16(1):3–33, January 1997.

[Ciam97] A Ciampalini, P. Cignoni, C. Montani, and R. Scopino. Multiresolution Decimation Based on Global Error. *The Visual Computer*, 13(5):228–246, 1997.

[Clin88] H. E. Cline, W. E. Lorensen, S. Ludke, C. R. Crawford, and B. C. Teeter. Two Algorithms for the Three-Dimensional Reconstruction of Tomograms. *Medical Physics*, 15(3):320–327, May/June 1988.

[Cohe86] M. F. Cohen, D. P. Greenberg, D. S. Immel, and P. J. Brock. An Efficient Radiosity Approach for Realistic Image Synthesis. *IEEE Computer Graphics and Applications*, 6(3):26–35, March 1986.

[Cohe88] M. F. Cohen, S. E. Chen, J. R. Wallace, and D. P. Greenberg. A Progressive Refinement Approach to Fast Radiosity Image Generation. In *SIGGRAPH '88 Conference Proceedings*, pages 75–84, August 1988.

[Cohe93] M. F. Cohen and J. R. Wallace. *Radiosity and Realistic Image Synthesis.* Academic Press, 1993.

[Cook86] R. L. Cook. Stochastic Sampling in Computer Graphics. *ACM Transaction on Graphics*, 5(1):51–72, 1986.

[Crow77] F. C. Crow. Shadow Algorithms for Computer Graphics. In *SIGGRAPH '77 Conference Proceedings*, pages 242–248, 1977.

[Curt97] C. J. Curtis, S. E. Anderson, J. E. Seims, K. W. Fleischer, and D. H. Salesin. Computer-generated watercolor. In *SIGGRAPH '97 Conference Proceedings*, pages 421–430, 1997.

[Cyru78] Martin Cyrus and Jeff Beck. Generalized two- and three dimensional clipping. *Computers and Graphics*, 12(1):23–28, 1978.

[Dani42] G. C. Danielson and C. Lanczos. Some Improvements in Practical Fourier Analysis and Their Application to X-ray Scattering from Liquids. *J. Franklin Institute*, pages 365–380, 435–452, 1942.

[Debe97] P. E. Debevec and J. Malik. Recovering high dynamic range radiance maps from photographs. In *SIGGRAPH 97 Conference Proceedings*, pages 369–378. Addison Wesley, August 1997.

[Debe98] P. Debevec. Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *SIGGRAPH '98 Conference Proceedings*, pages 189–198. ACM Press, 1998.

[DeCa02] D. DeCarlo and A. Santella. Stylization and abstraction of photographs. In *SIGGRAPH '02 Conference Proceedings*, pages 769–776. ACM Press, 2002.

[Deco99] X. Decoret, G. Schaufler, F. Sillion, and J. Dorsey. Multi-layered impostors for accelerated rendering. *Computer Graphics Forum (Proceedings of Eurographics '99)*, 18(3):61–73, 1999.

[Dena55] J. Denavit and R.S. Hartenberg. A Kinematic Notation for Lower-pair Mechanisms Based on Matrices. *Applied Mechanics*, 1955.

[Deus98] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. In *Proceedings of SIGGRAPH'98*, Annual Conference Series 1998, pages 275–286. ACM Press, 1998.

[Devl02] K. Devlin, A. Chalmers, A. Wilkie, and W. Purgathofer. Star: Tone reproduction and physically based spectral rendering. In Dieter Fellner and Roberto Scopignio, editors, *State of the Art Reports, Eurographics 2002*, pages 101–123. The Eurographics Association, September 2002.

[Doba00] Y. Dobashi, K. Kaneda, H. Yamashita, T. Okita, and T. Nishita. A simple, efficient method for realistic animation of clouds. In *SIGGRAPH '00 Conference Proceedings*, pages 19–28. ACM Press/Addison-Wesley Publishing Co., 2000.

[Down01] L. Downs, T. Möller, and C. H. Séquin. Occlusion horizons for driving through urban scenes. In *Symposium on Interactive 3D Graphics*, pages 121–124. ACM SIGGRAPH, 2001.

[Dreb88] R. A. Drebin, L. Carpenter, and P. Hanrahan. Volume Rendering. *Computer Graphics*, 22(4):51–58, August 1988. Proceedings of SIGGRAPH'88.

[Duch97] M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, Ch. Aldrich, and M. B. Mineev-Weinstein. Roaming terrain: real-time optimally adapting meshes. In *Proceedings of the conference on Visualization '97*, pages 81–88. ACM Press, 1997.

[Dura00] F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative visibility preprocessing using extended projections. In *SIGGRAPH '00 Conference Proceedings*, pages 239–248, 2000.

[Dutr03] P. Dutré, P. Bekaert, and K. Bala. *Advanced Global Illumination.* A. K. Peters, 2003.

[Eber01] D. Eberly. *3D game engine design: a practical approach to real-time computer graphics.* Morgan Kaufmann Publishers Inc., 2001.

[Eber03] S. D. Ebert, F. K. Musgrave, D. R. Peachey, K. Perlin, and S Worley. *Texturing & Modeling A Procedural Approach, third edition.* Morgan Kaufmann Publishers, 2003.

[Ekou91] A. B. Ekoule, F. C. Peyrin, and C. L. Odet. A Triangulation Algorithm from Arbitrary Shaped Multiple Planar Contours. *ACM Transaction on Graphics*, 10(2):182–199, April 1991.

[Ever03] C. Everitt and M. J. Kilgard. Optimized Stencil Shadow Volumes. Game Developer Conference Presentation, San Jose.
http://developer.nvidia.com/, March 2003.

[Fari93] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design a practical – guide third edition.* Academic Press, INC, 1993.

[Fatt02] R. Fattal, D. Lischinski, and M. Werman. Gradient domain high dynamic range compression. In *SIGGRAPH '02 Conference Proceedings*, pages 249–256. ACM Press, 2002.

[Fell93] D. W. Fellner and C. Helmberg. Robust Rendering of General Ellipses and Elliptical Arcs. *ACM Transactions on Graphics*, 12(3):251–276, July 1993.

[Fole90] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics – Principles and Practice.* 2nd ed., Addison–Wesley, Reading, Massachusetts, 1990.

[Fua00] P. Fua, L. Herda, R. Plänkers, and R. Boulic. Human shape and motion recovery using animation models. In *Proceedings of the 19th Congress, International Society for Photogrammetry and Remote Sensing, Amsterdam, Netherlands*, July 2000.

[Fuji85] A. Fujimoto and K. Iwata. Accelerated Ray–Tracing. In Tosiyasu L. Kunii ed.: *Computer Graphics – Visual Technology and Art*, pages 41–65. Springer–Verlag, 1985.

[Fuji86] A. Fujimoto, T. Tanaka, and K. Iwata. Accelerated Ray–Tracing System. *IEEE CG&A*, 6(4):16–26, April 1986.

[Funk02] T. Funkhouser, J. M. Jot, and N. Tsingos. „Sounds Good to Me!“ Computational Sound for Graphics, Virtual Reality, and Interactive Systems. *SIGGRAPH 2002 Course Notes*, 2002.

[Glas89] A. S. Glassner. *An Introduction to Ray Tracing.* Academic Press, San Diego, 1989.

[Glas90] A. S. Glassner. *Graphics Gems I.* Academic Press, 1990.

[Glas95] Andrew Glassner. *Principles of Digital Image Synthesis volume I and II.* Morgan Kaufman Publishers, Inc, 1995.

[Glei98] M. Gleicher. Retargetting motion to new characters. In *SIGGRAPH '98 Conference Proceedings*, pages 33–42. ACM Press, 1998.

[Gome97] J. Gomes and L. Velho. *Image Processing for Computer Graphics.* Springer-Verlag, 1997.

[Gonz87] R. C. Gonzales and P. Wintz. *Digital Image processing, 2nd edition.* Addison-Wesley, 1987.

[Gooc98] A. Gooch, B. Gooch, P. Shirley, and E. Cohen. A non-photorealistic lighting model for automatic technical illustration. In *SIGGRAPH '98 Conference Proceedings*, pages 447–452. ACM Press, 1998.

[Gooc01a] B. Gooch and A. Gooch. *Non-Photorealistic Rendering.* AK Peters Ltd, July 2001. ISBN: 1-56881-133-0.

[Gooc01b] B. Gooch, E. Reinhard, C. Moulding, and P. Shirley. Artistic composition for image creation. *Eurographics Workshop on Rendering*, pages 83–88, 2001.

[Good03] N. Goodnight, R. Wang, C. Woolley, and G. Humphreys. Interactive time-dependent tone mapping using programmable graphics hardware. In *Proceedings of the 14th Eurographics workshop on Rendering*, pages 26–37. Eurographics Association, 2003.

[Gora84] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile. Modelling the Interaction of Light Between Diffuse Surfaces. In *SIGGRAPH '84 Conference Proceedings*, pages 212–222, July 1984.

[Gott96] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. In *SIGGRAPH '96 Conference Proceedings*, pages 171–180, 1996.

[Gour71] H. Gouraud. Continuous Shading of Curved Surfaces. *IEEE Transactions on Computers*, 20(6), June 1971.

[Gree86] N. Greene. Environment mapping and other applications of world projections. *IEEE CG&A*, 6(11):21–29, November 1986.

[Gree89] N. Greene. Voxel Space Automata: Modeling with Stochastic Growth Processes in Voxel Space. In *SIGGRAPH '89 Conference Proceedings*, pages 175–184, 1989.

[Gree93] N. Greene, M. Kass, and G. Miller. Hierarchical Z-buffer visibility. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 231–238, 1993.

[Haeb90] P. Haeberli. Paint by numbers: abstract image representations. In *SIGGRAPH '90 Conference Proceedings*, pages 207–214. ACM Press, 1990.

[Hain86] E. A. Haines and D. P. Greenberg. The Light Buffer: A Shadow-Testing Accelerator. *IEEE CG&A*, 6(9):6–16, September 1986.

[Hanr86] P. Hanrahan. Using Caching and Breadth-First Search to Speed Up Ray Tracing. In *Proceedings of Graphics Interface '86*, pages 56–61, May 1986.

[Hanr91] P. Hanrahan, D. Salzman, and L. Aupperle. A Rapid Hierarchical Radiosity Algorithm. In *SIGGRAPH '91 Conference Proceedings*, pages 197–206, July 1991.

[Havr00] V. Havran. *Heuristic Ray Shooting Algorithms*. PhD thesis, ČVUT v Praze, November 2000.

[Havr03] V. Havran and W. Purgathofer. On Comparing Ray Shooting Algorithms. *Computer and Graphics*, 27(4):593–604, August 2003.

[Heck82] P. S. Heckbert. Color Image Quantization for Frame Buffer Displays. In *SIGGRAPH '82 Conference Proceedings*, pages 297–307, 1982.

[Heck86] P. S. Heckbert. Survey of Texture Mapping. *IEEE Computer Graphics and Applications*, 6(11):56–67, 1986.

[Heck90] P. S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. In *SIGGRAPH '90 Conference Proceedings*, pages 145–154. ACM Press, 1990.

[Heck97] P. S. Heckbert and M. Herf. *Simulating Soft Shadows with Graphics Hardware*. Technical Report CMU-CS-97-104, Carnegie Mellon University, January 1997.

[Heid91] T. Heidmann. Real Shadows, Real Time. *Iris Universe, Silicon Graphics Inc.*, 18:28–31, 1991.

[Hlav92] V. Hlaváč and M Šonka. *Počítačové vidění*. Grada, 1992.

[Höhn86] K. H. Höhne and R. Bernstein. Shading 3D-Images from CT Using Gray-Level Gradients. *IEEE Transactions on Medical Imaging*, 5(1):45–47, March 1986.

[Hopp96] H. Hoppe. Progressive Meshes. In *ACM Computer Graphics Proceedings Annual Conference Series (SIGGRAPH'96)*, pages 99–108. ACM Press, 1996.

[Hour85] J. C. Hourcade and A. Nicolas. Algorithms for Antialiased Cast Shadows. *Computer and Graphics*, 9(3):259–265, 1985.

[Huds97] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated occlusion culling using shadow frusta. In *ACM Symposium on Computational Geometry*, pages 1–10, 1997.

[Inte93] V. Interrante, W. Oliver, S. Pizer, and H. Fuchs. Chapter 11. – rendering. In *3D Confocal Microscopy: Volume Investigation of Biological Specimens*. Academic Press, 1993.

[ISO93] ISO. *11172-1: Information technology – Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1,5 Mbit/s – Part 1: Systems*. International standard ISO/IEC, 1993.

[ISO94a] ISO. *10918-1: Information Technology – Digital Compression and Coding of Continuous-tone Still Images, Part 1: Requirements and Guidelines*. International standard ISO/IEC, 1994.

[ISO94b] ISO. *13818-1: Information technology – Generic coding of moving pictures and associated audio information – Part 1: Systems*. International standard ISO/IEC, 1994.

[ISO96] ISO. *12639: Prepress Digital Data Exchange – Tag Image File Format for Image Technology (TIFF/IT)*. International standard ISO/IEC, 1996.

[ISO97] ISO. *14772-1: Information Technology – Virtual Reality Modelling Language (VRML)*. International standard ISO/IEC, 1997.

[ISO00] ISO. *15444-1: Information Technology – JPEG 2000 image coding system – Part 1: Core coding system*. International standard ISO/IEC, 2000.

[ISO01a] ISO. *14496-1: Information technology – Coding of audio-visual objects – Part 1: Systems*. International standard ISO/IEC, 2001.

[ISO01b] ISO. *TR 21000-1: Information technology – Multimedia framework (MPEG-21) – Part 1: Vision, Technologies and Strategy*. International standard ISO/IEC, 2001.

[ISO02] ISO. *15938-1: Information technology – Multimedia content description interface – Part 1: Systems*. International standard ISO/IEC, 2002.

[ISO03a] ISO. *FCD 19774: Information Technology – Humanoid Animation (H-Anim)*. Committee Draft – International Standard ISO/IEC, 2003.

[ISO03b] ISO. *FDIS 19775: Information Technology – Extensible 3D (X3D).* Draft International Standard ISO/IEC, 2003.

[ISO04] ISO. *15948 Information technology – Portable Network Graphics (PNG).* International Standard ISO/IEC, 2004.

[Jens01] H. W. Jensen. *Realistic image synthesis using photon mapping.* A. K. Peters, Ltd., 2001.

[Jone00] H. Jones. Modelling of Growing Natural Forms. In *Eurographics'00 Tutorials.* Springer–Verlag, 2000.

[Kaji84] J. T. Kajiya and B. P. von Herzen. Ray Tracing Volume Densities. *Computer Graphics*, 18(3):165–174, 1984.

[Kaji86] J. T. Kajiya. The Rendering Equation. In *SIGGRAPH '86 Conference Proceedings*, pages 143–150, August 1986.

[Kang03] S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High dynamic range video. *ACM Trans. Graph.*, 22(3):319–325, 2003.

[Kauf91] A. Kaufman, editor. *Volume Visualisation.* Compute Society Press, Los Alamitos, CA, 1991.

[Kava03] L. Kavan and J. Žára. Real-time skin deformation with bones blending. In *Proceedings of WSCG – Short Papers*, pages 69–74. Pilsen: University of West Bohemia, 2003.

[Kawa01] T. Kawashima, K. Imamoto, H. Kato, K. Tachibana, and M. Billinghurst. Magic Paddle: A Tangible Augmented Reality Interface for Object Manipulation. In *Proceedings on the Second International Symposium on Mixed Reality (ISMR2001)*, pages 194–195, 2001.

[Kell88] A.D. Kelley, M.C. Malin, and G.M. Nielson. Terrain Simulation Using a Model of Stream Errosion. *Computer Graphics*, 22(4):263–268, 1988.

[Kepp75] E. Keppel. Approximating Complex Surfaces by Triangulation of Contour Lines. *"IBM Journal of Research and Development"*, 19:2–11, 1975.

[Kilg01] M. J. Kilgard. Robust Stencil Shadow Volumes. CEDEC 2001 presentation, Tokyo. *http://developer.nvidia.com/object/cedec_stencil.html*, September 2001.

[Kilg02] M. J. Kilgard and C. Everitt. Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering. *http://developer.nvidia.com/*, 2002.

[Klos98] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[Knut87] D. Knut. Digital Halftones by Dot Diffusion. *ACM Transactions on Graphics*, 6(4):245–273, October 1987.

[Koch84] D. H. U. Kochanek and R. H. Bartels. Interpolating splines with local tension, continuity, and bias control. In *SIGGRAPH '84 Conference Proceedings*, pages 33–41. ACM Press, 1984.

[Kole93] Kolektiv autorů. *Encyklopedický slovník*. Encyklopedický dům, 1993.

[Koll03] T. Kollig and A. Keller. Efficient illumination by high dynamic range images. In *Proceedings of the 14th Eurographics workshop on Rendering*, pages 45–50. Eurographics Association, 2003.

[Kowa99] M. A. Kowalski, L. Markosian, J. D. Northrup, L. Bourdev, R. Barzel, L. S. Holden, and J. Hughes. Art-based rendering of fur, grass, and trees. In *SIGGRAPH '99 Conference Proceedings*, pages 433–438. Addison Wesley Longman, 1999.

[Kret01] U. Kretschmer, V. Coors, U. Spierling, D. Grasbon, K. Schneider, I. Rojas, and R. Malaka. Meeting the spirit of history. In *Proceedings of the 2001 conference on Virtual reality, archeology, and cultural heritage*, pages 141–152. ACM Press, 2001.

[Kuma96] S. Kumar, D. Manocha, W. Garrett, and M. Lin. Hierarchical back-face computation. In *Rendering Techniques (Proccedings of Eurographics Workshop on Rendering '96)*, pages 235–244. Springer Wein, June 1996.

[Kutt00] H. Kuttruff. *Room Acoustics*. Spon Press, New York, 4th edition, 2000.

[Lafo93] Eric P. Lafortune and Yves D. Willems. Bi-directional Path Tracing. In H. P. Santo, editor, *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*, pages 145–153, Alvor, Portugal, 1993.

[Lamo94] H. J. Lamousin and W. N. Waggenspack. NURBS-Based Free-Form Deformation. *Computer Graphics and Applications*, 1994.

[Lars97] G. W. Larson, H. Rushmeier, and C. Piatko. A visibility matching tone reproduction operator for high dynamic range scenes. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):291–306, 1997.

[Levo88] M. Levoy. Display of Surfaces from Volume Data. *IEEE Computer Graphics and Applications*, 8(5):29–37, May 1988.

[Levo00] M. Levoy, K. Pulli, K. Curless, S. Rusinkiewitz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The Digital Michelangelo project: 3D scanning of large statues. In *SIGGRAPH '00 Conference Proceedings*, pages 131–144, 2000.

[Lewi90] R. Lewis. *Practical Digital Image Processing*. Ellis Horwood, 1990.

[Lian83] You-Dong Liang and Brian A. Barsky. An analysis and algorithm for polygon clipping. *Commun. ACM*, 26(11):868–877, 1983.

[Lich98] B. Lichtenbelt, R. Crane, and S. Naqvi. *Introduction to Volume Rendering*. Prentice Hall PTR, 1998.

[Liu02] C. K. Liu and Z. Popović. Synthesis of complex dynamic character motion from simple animations. In *SIGGRAPH '02 Conference Proceedings*, pages 408–416. ACM Press, 2002.

[Lore87] W. Lorensen and H. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):163–169, July 1987.

[Lueb95] D. Luebke and C. Georges. Portals and mirrors: Simple, fast evaluation of potentially visible sets. In *Proceedings of Symposium on Interactive 3D Graphics '95*, pages 105–106, 1995.

[Mall88] T. J. Malley. A shading method for computer generated images. *Master´s thesis, Dept. of Computer Science, University of Utah*, June 1988.

[Mand82] B. B. Mandelbrot. *The Fractal Geometry of Nature.* W. H. Freeman, San Francisco, 1982.

[Masu99] M. Masuch, S. Schlechtweg, and R. Schulz. Speedlines: Depicting Motion in Motionless Pictures. In *SIGGRAPH'99 Conference Abstracts and Applications*, page 277, New York, 1999. ACM SIGGRAPH.

[Matu02] W. Matusik, H. Pfister, R. Ziegler, A. Ngan, and L. McMillan. Acquisition and Rendering of Transparent and Refractive Objects. In *Proceedings of the 13th Eurographics workshop on Rendering*, pages 267–278. Eurographics Association, 2002.

[Max95] N. Max. Optical Models for Direct Volume Rendering. *IEEE Transaction on Visualization and Computer Graphics*, 1(2):99–108, June 1995.

[Měch96] R. Měch and P. Prusinkiewicz. Visual Models of Plants Interacting With Their Environment. In *SIGGRAPH '96 Conference Proceedings*, pages 397–410, 1996.

[Meye92] D. Meyers, S. Skinner, and K. Sloan. Surfaces from Contours. *ACM Transactions on Graphics*, 11(3):228–258, July 1992.

[Moll02] T. A. Moller and E. Haines. *Real-Time Rendering.* A K Peters, 2002.

[Mort97] Michael E. Mortenson. *Geometric Modeling.* John Wiley & Sons, Inc., 1997.

[Murc87] G. M. Murch. Color displays and color science. In H. J. Durrett, editor, *Color and the Computer*, pages 1–26. Academic Press, Orlando, FL, 1987.

[Murr94] R. M. Murray, Z. Li, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation.* CRC Press, 1994.

[Murr95] J. D. Murray and W. van Ryper. *Encyklopedie grafických formátů.* Computer Press, 1995.

[Neid03] J. Neider, T. Davis, and M. Woo. *The OpenGL Programming Guide – 4th Edition The Official Guide to Learning OpenGL Version 1.4.* Addison-Wesley Publishing Company, 2003.

[Neum94] L. Neumann, M. Feda, M. Kopp, and W. Purgathofer. A new stochastic radiosity method for highly complex scenes. In *(Proceedings of the Fifth Eurographics Workshop on Rendering)*, pages 195–206, Darmstadt, Germany, 1994.

[Neum95] L. Neumann, W. Purgathofer, R. F. Tobler, A. Neumann, P. Elias, M. Feda, and X. Pueyo. The Stochastic Ray Method for Radiosity. In *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pages 206–218. Springer-Verlag, 1995.

[Newe72] M. E. Newell, R. G. Newell, and T. L. Sancha. A Solution to the Hidden Surface Problem. In *Proceedings of the ACM National Conference*, 1972.

[Niel93] G. M. Nielson. The Volume Rendering Equations. In G. M. Nielson, editor, *Scientific Data Visualization*. Eurographics'93 Tutorial, September 1993.

[Niel94] G.M. Nielson and J. Tvedt. Comparing Methods of Interpolation for Scattered Volumetric Data. In *State of the Art in Computer Graphics – Aspects of Visualization*, pages 67–86. Springer, 1994.

[Ning93] P. Ning and J. Bloomenthal. An Evaluation of Implicit Surfaces Tilers. *IEEE Computer Graphics and Applications*, 13(6):33–41, 1993.

[Nire02] S. Nirenstein, E. Blake, and J. Gain. Exact From-Region visibility culling. In *Proceedings of Eurographics Workshop on Rendering '02*, pages 199–210, 2002.

[Nish85] T. Nishita and E. Nakamae. Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection. In *SIGGRAPH '85 Conference Proceedings*, pages 23–30, July 1985.

[Nožiˇ91] J. Nožička. *Mechanika a termodynamika*. ČVUT, 1991.

[Oliv95] J.-M. Oliva. *Reconstruction Tridimentionnelle d'Objets Complexes à l'aide de Diagrammes de Voronoï Simplifiés*. No 130 gd, Thése de l'Université et de l'École des Mines de Saint-Etienne, October 1995.

[Onou00] K. Onoue and T. Nishita. A Method for Modeling and Rendering Dunes with Wind-ripples. In *Proceedings of Pacific Graphics'00*, pages 427–428, 2000.

[Onou03] K. Onoue and T. Nishita. Virtual sandbox. In *Proceedings of Pacific Graphics'03*, pages 252–260. IEEE Computer Society, 2003.

[O'Ro94] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1994.

[Over93] K. van Overveld and B. Wyvill. Potentials, Polygons and Penguins. An Efficient Adaptive Algorithm for Triangulating an Equipotential Surface. *Implicit Sufraces'95*, pages 31–62, 1993.

[Paet91] A. W. Paeth. Image File Compression Made Easy. In James Arvo ed.: *Graphics Gems II*. Academic Press, San Diego, 1991.

[Palm95] I. J. Palmer and R. L. Grimsdale. Collision Detection for Animation using Sphere-Trees. *Computer Graphics Forum*, 14(5):105–116, 1995.

[Patt93] S. N. Pattanaik and S. P. Mudur. The potential equation and importance in illumination computations. In *Computer Graphics Forum*, volume 12, pages 131–136, 1993.

[Paul01] M. Pauly and M. Gross. Spectral processing of point sampled geometry. In *SIGGRAPH '01 Conference Proceedings*, pages 379–386. ACM Press, 2001.

[Paul03] M. Pauly, R. Keiser, L. P. Kobbelt, and M. Gross. Shape Modeling with Point-Sampled Geometry. *ACM Transactions on Graphics*, 22(3):641–650, 2003.

[Payn90] B. A. Payne and A. W. Toga. Surface Mapping Brain Function on 3D Models. *Computer Graphics and Applications*, pages 33–41, September 1990.

[Peac86] D. R. Peachey. Modeling Waves and Surf. In *SIGGRAPH '86 Conference Proceedings*, pages 65–74, 1986.

[Peit92] H. O. Peitgen, H. Jurgens, and D. Saupe. *Chaos and Fractals – New Frontiers of Science*. Springer–Verlag, New York, 1992.

[Perl85] K. Perlin. An Image Synthetizer. In *SIGGRAPH '85 Conference Proceedings*, pages 287–296, 1985.

[Phon75] B. T. Phong. Illumination for Computer Generated Images. *Communications with the ACM*, 18(6), June 1975.

[Pieg91] L. Piegl. On NURBS: A Survey. *IEEE Computer Graphics and Applications*, January 1991.

[Pieg95] L. Piegl and W. Tiller. *The NURBS Book*. Springer Verlag, 1995.

[Prus90] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer–Verlag, 1990.

[Prus94] P. Prusinkiewicz, M. James, and R. Měch. Synthetic Topiary. In *Proceedings of SIGGRAPH '94*, volume I, pages 351–358, 1994.

[Reev83] W. Reeves. Particle Systems – A Technique for Modeling a Class of Fuzzy Objects. *ACM Transaction on Graphics*, 2(2):12–22, 1983.

[Reev87] W. Reeves, D. Salesin, and R. Cook. Rendering Antialiased Shadows with Depth Maps. In *SIGGRAPH '87 Conference Proceedings*, pages 283–291, 1987.

[Rein02] E. Reinhard, M. Stark, P. Shirley, and J. Ferwerda. Photographic tone reproduction for digital images. In *SIGGRAPH '02 Conference Proceedings*, pages 267–276. ACM Press, 2002.

[Rekt95] K. Rektorys. *Přehled užité matematiky II*. Prometheus, Praha, šesté vydání, 1995.

[Robe63] L. G. Roberts. *Machine Perception of Three Dimensional Solids*. MIT Lincoln Lab.Rep., TR 315, May 1963.

[Roge01] D. F. Rogers. *An Introduction to NURBS*. Morgan Kaufmann Publishers, 2001.

[Rusi00] S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In *SIGGRAPH '00 Conference Proceedings*, pages 343–352, 2000.

[Sabe88] P. Sabella. A Rendering Algorithm for Visualising 3D Scalar Fields. *Computer Graphics*, 22(4):51–58, August 1988.

[Salo99] David Salomon. *Computer Graphics & Geometric Modeling*. Springer, NY, 1999.

[Sber93] M. Sbert. An integral geometry based method for fast form-factor computation. In *(Proceedings of Eurographics '93)*, pages 409–420. North-Holland, 1993.

[Sber95] M. Sbert, F. Perez, and X. Pueyo. Global Monte Carlo: A Progressive Solution. In *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pages 231–239. Springer-Verlag, 1995.

[Sber96] M. Sbert. The use of global random directions to compute radiosity - global monte carlo techniques. *PhD Dissertation*, 1996.

[Sber97] M. Sbert. Error and complexity of random walk monte carlo radiosity. In *IEEE Transactions on Visualization and Computer Graphics*, volume 3, pages 23–38, 1997.

[Scag01] D. Scagliarini, A. Coralini, E. Vecchietti, T. S. Cinotti, L. Roffia, S. Galasso, M. Malavasi, M. Pigozzi, E. Romagnoli, and F. Sforza. Exciting understanding in Pompeii through on-site parallel interaction with dual time virtual models. In *Proceedings of the International Symposium on Virtual Reality, Archaeology and Cultural Heritage (VAST 2001)*, pages 97–104, 2001.

[Scha00] G. Schaufler, J. Dorsey, X. Decoret, and F. X. Sillion. Conservative volumetric visibility with occluder fusion. In *Computer Graphics (SIGGRAPH '00 Proceedings)*, pages 229–238, 2000.

[Schr92] W. Schroeder, J. Zarge, and W. Lorensen. Decimation of Triangle Meshes. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):65–70, July 1992.

[Sede86] T. W. Sederberg and S. R. Parry. Free-Form Deformation of Solid Models. *ACM Computer Graphics*, 20(4), 1986.

[Seet04] H. Seetzen, W. Heidrich, W. Stuerzlinger, G. Ward, L. Whitehead, M. Trentacoste, A. Ghosh, and A. Vorozcovs. High dynamic range display systems. In *SIGGRAPH '04 Conference Proceedings*. Addison Wesley, 2004.

[Sega92] M. Segal, C. Korobkin, van Rolf Widenfelt, J. Foran, and P. Haeberli. Fast shadows and lighting effects using texture mapping. In *SIGGRAPH '92 Conference Proceedings*, pages 249–252, July 1992.

[Shir90] P. Shirley. A ray tracing method for illumination calculation in diffuse-specular scenes. In *Graphics Interface '90*, pages 205–212, May 1990.

[Shoe85] Ken Shoemake. Animating rotation with quaternion curves. In *SIGGRAPH '85 Conference Proceedings*, pages 245–254. ACM Press, 1985.

[Skal93] V. Skala. *Světlo, barvy a barevné systémy v počítačové grafice*. Academia, Praha, 1993.

[SlllK98] L. Szirmay-Kalos and W. Purgathofer. Global ray-bundle tracing with hardware acceleration. In *Ninth Eurographics Workshop on Rendering*, Austria, June 1998.

[Smit92] B. E. Smits, J. R. Arvo, and D. P. Greenberg. An importance-driven radiosity algorithm. In *Computer Graphics (ACM SIGGRAPH '92 Proceedings)*, July 1992.

[Smyt90] D. Smythe. A Two–Pass Mesh Warping Algorithm for Object Transforming and Image Interpolation. *ILM Technical memo 1030, Computer Graphics Dept., Lucasfilm Ltd.*, 1990.

[Soro81] B. I. Soroka. Generalized Cones from Serial Sections. *Comput. Graph. Image. Proc.*, 15:154–166, 1981.

[Sous03] M. C. Sousa and P. Rusinkiewicz. A few good lines: Suggestive drawing of 3d models. *Computer Graphics Forum (Proc. of Eurographics '03)*, 22(3):381–390, 2003.

[Spee85] L. R. Speer, T. D. DeRose, and B. A. Barsky. A Theoretical and Empirical Analysis of Coherent Ray–Tracing. In *Proceedings of Graphics Interface '85*, pages 11–25, May 1985.

[Sper90] D. Speray and S. Kennon. Volume Probes: Interactive Data Exploration on Arbitrary Grids. *Computer Graphics*, 24(5):1–12, 1990.

[Spro68] R.F. Sproull and I.E. Sutherland. A clipping divider. In *AFIPS Conference Proceedings*, volume 33, pages 765–776, 1968.

[Stam02] M. Stamminger and G. Drettakis. Perspective shadow maps. In *SIGGRAPH '02 Conference Proceedings)*, pages 557–562, 2002.

[Stol96] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgann Kaufmann, San Francisco, CA, 1996.

[Stra90] P. S. Strauss. A Realistic Lighting Model for Computer Animators. *IEEE Computer Graphics and Applications*, 10(6):56–64, November 1990.

[Stro02] T. Strothotte and S. Schlechtweg. *Non-Photorealistic Computer Graphics. Modeling, Rendering, and Animation*. Morgan Kaufmann, San Francisco, 2002.

[Sung92] K. Sung and P. Shirley. Ray Tracing with the BSP Tree. In David Kirk ed.: *Graphics Gems III.*, pages 271–274, 538–546. Academic Press, 1992.

[Suth74] Ivan E. Sutherland and Gary W. Hodgman. Reentrant polygon clipping. *Commun. ACM*, 17(1):32–42, 1974.

[Tell91] S. J. Teller and C. H. Séquin. Visibility preprocessing for interactive walkthroughs. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, pages 61–69, 1991.

[Torr67] K. E. Torrance and E. M. Sparrow. Theory for Off-Specular Reflection from Roughened Surfaces. *J. Opt. Soc. Am.*, 57(9):1105–1114, September 1967.

[Tumb93] J. Tumblin and H. Rushmeier. Tone reproduction for realistic images. *IEEE Comput. Graph. Appl.*, 13(6):42–48, 1993.

[Tumb97] J. Tumblin and G. Turk. Low curvature image simplifiers (LCIS): A boundary hierarchy for detail-preserving contrast reduction. In *SIGGRAPH 99 Conference Proceedings*, pages 83–90. Addison Wesley, 1997.

[Wall89] J. R. Wallace, K. A. Elmquist, and E. A. Haines. A Ray Tracing Algorithm for Progressive Radiosity. In *SIGGRAPH '89 Conference Proceedings*, pages 315–324, July 1989.

[Wand02] M. Wand and W. Straßer. Multi-Resolution Rendering of Complex Animated Scenes. *Computer Graphics Forum*, 21(3), 2002.

[Wang02] X. C. Wang and C. Phillips. Multi-weight enveloping: least-squares approximation techniques for skin animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 129–138. ACM Press, 2002.

[Ward94] G. Ward. A contrast-based scalefactor for luminance display. *Graphics Gems IV*, pages 415–421, 1994.

[Warn69] J. Warnock. *A Hidden-Surface Algorithm for Computer-Generated Half-Tone Pictures.* [TR4-15, NTIS AD-753 671] Salt Lake City (Utah). University of Utah, Comput. Sci. Dept., 1969.

[Watt92] A. Watt and M. Watt. *Advanced Animation and Rendering Techniques: Theory and Practice.* Addison-Wesley, Reading, 1992.

[Weil77] K. Weiler and P. Atherton. Hidden Surface Removal Using Polygon Area Sorting. In *SIGGRAPH '77 Conference Proceedings*, pages 214–222, August 1977.

[Welc84] T. A. Welch. A Technique for High Performance Data Compression. *IEEE Computer*, 17(6), June 1984.

[West90] L. Westover. Footprint Evaluation for Volume Rendering. *Computer Graphics*, 24(4):367–376, August 1990.

[Whit80] T. Whitted. An Improved Illumination Model for Shaded Display. *ACM Communication on Graphics*, 26(6):342–349, 1980.

[Will78] L. Williams. Casting curved shadows on curved surfaces. *Computer Graphics*, 12(3):270–274, 1978.

[Wonk00] P. Wonka, M. Wimmer, and D. Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. In *Rendering Techniques (Proceedings of Eurographics Workshop on Rendering '00)*, pages 71–82, 2000.

[Woo92] A. Woo. The Shadow Depth Map Revisited. *Graphics Gems III, AP Professional*, pages 338–342, 1992.

[Wool90] G. Woolberg. *Digital Image Warping.* IEEE Computer Society Press Monograph, Los Alamitos, California, 1990.

[Wyvi86] G. Wyvill, C. McPheeters, and B. Wyvill. Data Structures for Soft Objects. *The Visual Computer*, 2(4):227–234, 1986.

[Yang99] D. X. D. Yang, A. E. Gamal, B. Fowler, and H. Tian. CMOS image sensor with ultra wide dynamic range floating-point pixellevel ADC. In *IEEE International Solid-State Circuits Conference*, San Francisco, CA, USA, 1999.

[Yell82] T. Yellot. Spectral Analysis of Spatial Sampling by Phototreceptors: Topological Disorder Preventing Alias. *Vision Research*, 22:1205–1210, 1982.

[Zhan97] H. Zhang, D. Manocha, T. Hudson, and E. Kenneth. Visibility culling using hierarchical occlusion maps. In *SIGGRAPH '97 Conference Proceedings*, pages 77–88, 1997.

[Zori00] D. Zorin and P. Schröder. Subdivision for Modeling and Animation. SIGGRAPH Course Notes, 8 2000.

[Zwic01] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface splatting. In *SIGGRAPH '01 Conference Proceedings*, pages 371–378. ACM Press, 2001.

[Zwic02] M. Zwicker, M. Pauly, O. Knoll, and M. Gross. Pointshop 3D: An interactive system for point-based surface editing. In *SIGGRAPH '02 Conference Proceedings*, pages 322–329, 2002.

[Žára99] J. Žára. *VRML 97 — Laskavý průvodce virtuálními světy.* Computer Press, Brno, 1999.

Rejstřík