ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA ELEKTROTECHNICKÁ

KATEDRA POČÍTAČOVÉ GRAFIKY A INTERAKCE

PODPORA VÝUKY OPENGL

DIPLOMOVÁ PRÁCE

BC. MARTIN VAŇKO

Vedoucí práce Ing. Petr Felkel PhD.

2010

Poďakovanie

Chcem poďakovať najmä mojej snúbenici Márii Michnovej za jej veľkú pomoc, podporu a energiu, ktorú mi dodávala počas celej tvorby tejto práce. Ďakujem aj mojim rodičom, ktorí ma vždy podporovali v mojom úsilí a dodávali mi silu na tejto ceste plnej prekážok. Na záver by som chcel poďakovať aj vedúcemu mojej diplomovej práce Ing. Petrovi Felklovi PhD., za jeho prejavenú ochotu kedykoľvek pomôcť a podnetné pripomienky.

Declaration

I hereby declare that I have completed this diploma thesis independently and that I have listed all the publications used.

In Prague 13.12.2010

Čestné prehlásenie

Týmto prehlasujem, že som svoju diplomovú prácu vypracoval samostatne a že som v záverečnom zozname uviedol všetky použité podklady.

V Prahe 13.12.2010

Abstract

There are many online examples of OpenGL applications. Finding a well-prepared programs is difficult. Students starting with an OpenGL lack of clear and targeted examples. Therefore, I created a set of applications with an emphasis on simplicity. I established a new and modern criteria, which I implemented. Examples are designed for both fixed and programmable pipelines. For the creation were used less known, but new frameworks. Thus, there incept programs that are applicable to modern OpenGL. Architecture allows applications to be extended into larger projects. It provides operations such as loading textures or 3D models.

Abstrakt

Príkladov aplikácií OpenGL je na internete mnoho. Nájsť dobre spracované programy je obtiažne. Študenti začínajúci s OpenGL majú nedostatok prehľadných, a cielene vypracovaných príkladov. Preto som vytvoril sadu aplikácií s dôrazom na jednotnosť a prehľadnosť. Tým sa docielila pomoc hlavne začiatočníkom. Príklady sú zamerané na fixnú aj programovateľnú pipeline. K vytvoreniu boli použité menej známe, ale nové frameworky. Vznikli tak programy, ktoré sú použiteľné s novodobým OpenGL. Architektúra aplikácií umožňuje ich rozšírenie do rozsiahlejších projektov. Zabezpečuje totiž operácie ako načítavanie textúr, či 3D modelov.

OBSAH

ZOZNAM OBRÁZKOV

1 Úvod

Z osobnej skúsenosti, akademického, ale aj vedeckého prostredia so zameraním na počítačovú grafiku pozorujem stále častejšie požiadavky na znalosť grafickej technológie OpenGL. Ako už z jej názvu vyplýva je táto technológia otvorená a určená pre voľné použitie a distribúciu. Je taktiež nezávislá na architektúre a je široko podporovaná všetkými veľkými výrobcami súčasného grafického hardvéru.

Pre uvedené dôvody sa stala mimoriadne rozšírenou v súkromnom, akademickom, vedeckom ale najmä komerčnom a open-source sektore. Jej hlavnou prednosťou sú akcelerácia zobrazovania prevažne RT počítačovej grafiky na finančne dostupných GPU. Jedná sa o najrozšírenejšiu zobrazovaciu technológiu v súčasnosti vo všetkých odvetviach s výnimkou komerčného sektoru. Súčasné použitie v tomto prostredí je bohužiaľ menšie ako použitie konkurenčnej a spoplatnenej technológie Direct3D od firmy Microsoft. Hlavným dôvodom je existencia zákazníckej podpory u tohto rozhrania.

Hlavnou prekážkou v správnom pochopení OpenGL je nedostupnosť systematicky vytvorených a jednotných príkladov. V zásade sa vždy jedná o projekty podobné môjmu. Autor v nich však neprikladá váhu ani výuke ani názornosti, naopak prezentuje len ním dosiahnuté výsledky. Zároveň sa v týchto projektoch často prelína niekoľko vecí súčasne. Jedná sa o komplexné projekty. Navyše sú tieto projekty vždy zastaralé a neaktuálne. Obvykle ide o zastaralosť troch zo súčasných štyroch generácií OpenGL. Takýchto projektov je na internete dostupných mnoho. Pokročilému užívateľovi môžu aj tieto projekty poskytnúť prínosné informácie. Pre začiatočníka však obvykle predstavujú zmätok a slúžia len ako odstrašujúce príklady.

2 Ciele projektu

Myšlienka vytvorenia tejto práce vznikla počas môjho štúdia. Podnetom mi boli neustále problémy pri práci s OpenGL API v mnohých projektoch a to iba z dôvodu jeho nedostatočnej znalosti. Preto som sa rozhodol vytvoriť prácu, ktorá pomôže lepšie pochopiť toto rozhranie.

Filozofiou projektu Taygeta je za pomoci moderných prostriedkov demonštrovať názorné použitie súčasného i moderného OpenGL. Základným krokom je návrh projektu s dobre definovaným a metodickým cieľom.

Posledný vývoj grafických nástrojov, ako aj samotnej grafickej knižnice OpenGL, predurčil na vytvorenie projektu použitie súčasných novodobých nástrojov. Prispôsobenie sa moderným potrebám znamená mimo iného objektovú orientáciu kódu, paralelné spracovanie či možnosť vytvárania viacerých okien súčasne. Splnenie týchto požiadaviek prináša ďalšie pozitívum v podobe načrtnutia iných možností programovania v OpenGL, než len programovania prostredníctvom knižnice GLUT.

Práca si kladie za cieľ vytvorenie sady vzorových programov znázorňujúcich základy, ale aj pokročilé techniky použitia knižnice OpenGL. Na dosiahnutie celkovej jednoduchosti a dobrej orientácie v projekte je zamýšľaný postup, v ktorom bude vždy jeden program zameraný na jeden konkrétny problém. Tým sa zaistí vysoká prehľadnosť a rýchlejšie pochopenie problematiky. Cieľom každého programu bude demonštrácia vybranej techniky s použitím minimálneho množstva kódu a s prihliadnutím na užívateľské experimentovanie. Vždy bude predpokladom použitie programov ako základ väčšej užívateľskej aplikácie a pre tento účel bude projekt disponovať knižnicou implementujúcou základné požiadavky jednoduchého grafického programu. Vedľajším cieľom projektu je osvojenie si práce s modernými prostriedkami a dobrých programovacích návykov užívateľom.

Práca si nekladie za cieľ vytvoriť knihu o OpenGL. Nejedná sa ani o výukový kurz. Zámerom nieje obsiahnuť všetky možnosti knižnice ani poskytnúť sled na seba nadväzujúcich krokov vytvárajúcich vyšší, komplexný cieľ. Účelom je poskytnúť stručné príklady k vybraným technikám. Príklady budú vyberané s ohľadom na ich význam v praktickom použití ako aj na výskyt v publikáciách venovaných tématike OpenGL. Pri ich vytváraní sa bude postupovať spôsobom, ktorým tieto informácie spracuje ako začiatočník, tak aj pokročilý užívateľ.

Vedľajším efektom práce je prezentovanie knižnice OpenGL ako nástroja, nie cieľa. Toto ostáva stále nepochopené prevažne amatérskymi užívateľmi. Je dobré názorne predviesť možnosti a použitie tohto API, pritom ale poukázať na jeho určenie pre integráciu do komplexných systémov, akými sú herné stroje a modelovacie nástroje. Ukázať jeho jednoduchosť a široké možnosti pri vizualizácií rôznych efektov. Zároveň ale odhaliť chýbajúce vlastnosti a pomernú zložitosť malých jednoúčelových programov pri jeho použití. Skrátene, cieľom nieje tvorba OpenGL aplikácií samotných, ale aplikácií využívajúcich API knižnice pre akceleráciu svojho grafického podsystému a výpočtov s tým spojených.

3 Existujúce implementácie

V súčasnej dobe je OpenGL jedným z najrozšírenejších grafických rozhraní, ktorého vznik sa datuje približne do roku 1991. Vzhľadom na svoju dlhotrvajúcu existenciu a prepracovanosť, expandovala táto knižnica do množstva dnes bežne používaných systémov. S rozširovaním a zlepšovaním sa rozrastala aj komunita prívržencov tohto API. Preto dnes nájdeme na internete nespočetne mnoho návodov, tipov, trikov a príkladov ako toto grafické rozhranie použiť.

Táto práca má podobné zameranie ako množstvo iných, už existujúcich projektov. Je teda nutné analyzovať určité množstvo existujúcich implementácií. Len tak sa overí, či má zmysel v práci naďalej pokračovať, prípadne o ktoré prvky by sa tento projekt mohol vylepšiť. V nasledujúcom texte bude uvedený zoznam niekoľkých väčších, ľahko vyhľadateľných internetových projektov spolu s ich zistenými vlastnosťami. Pokiaľ nebude uvedené inak, sú príklady na stránkach vždy dostupné pre programovací jazyk C++.

Bonzai Software

Stránky zamerané komerčne na projekty autora. Až na dve výnimky neposkytujú žiadne použiteľné návody. Dostupné sú shadery z projektov, ktoré sú zastaralé. Taktiež jednotnosť zdrojových kódov je slabá. Návody sú síce pokročilejšieho charakteru avšak pomerne dobre vysvetlené. Stránka okrem nich ale neposkytuje žiadny relevantný zdroj informácií.

URL: http://bonzaisoftware.com

Clockwork coders

Portál zastrešený pod hlavičkou OpenGL je dobrým začiatkom do jazyka GLSL. Zaoberá sa výhradne shadermi. Poskytuje približne desať základných programov. Je dostatočne komentovaný avšak odkazy na zdrojové kódy príkladov sú nefunkčné. Zároveň sú príklady

zastaralé a orientované na OpenGL verziu 1.5 využívajúcu GLSL shadery iba ako ARB extenziu. Jednotnosť kódov nebola zistená nakoľko sa nedajú stiahnuť. Podľa kontextu, obrázkov k návodom a prezentovaných techník, odhadujem na použitie knižnice GLUT ako základu všetkých programov. Preto si dovolím tvrdiť že zdrojové kódy budú pomerne jednotné aj keď procedurálne a zastaralé.

URL: http://www.clockworkcoders.com

DHPO ware

Celkové množstvo návodov tohto projektu je dobré. Nezameriava sa však len na OpenGL, ale aj na DirectX a v ňom implementovaný framework XNA. Celkový počet návodov pre OpenGL nieje nijak výrazný. Príkladom chýbajú kvalitnejšie a podrobnejšie popisy a taktiež nie sú postavené na jednotnej kostre. Ich závislosti taktiež nie sú popísané. Pri problémoch s kompiláciou je ťažké doplniť závislosti a zdrojové kódy opraviť. Pre začiatočníka sú príklady prakticky nepoužiteľné kvôli ich rozsahu a komplexnosti, nejednotnosti a občasnému zlyhaniu prekladu. Aj napriek týmto nedostatkom projekt poskytuje ukážky programov vo verzii OpenGL 3.0, čo je pri dnešnom nedostatku aktuálnych programov jednoznačné plus.

URL: http://www.dhpoware.com

Game-Dev

Zameraním tohto projektu je vytvorenie komunity užívateľov zaoberajúcich sa programovaním počítačovej grafiky a výmenou informácií. Užívateľ tu nájde množstvo návodov, rád a trikov k tvorbe herných svetov ako aj k niektorým pokročilým grafickým technikám. Väčšinou ale bez preložiteľných zdrojových kódov. O jednotnosti príkladov nemožno hovoriť, keďže tie sú poskytované vždy iným užívateľom. Wiki stránky tohto portálu obsahujú takisto zaujímavú zbierku informácií a odkazov na rôzne zdroje, ale iba so zameraním na vytváranie hier.

URL: http://www.gamedev.net

URL: http://gpwiki.org

Game-Dev NEHE

V súčasnosti jeden z najlepších projektov. Je zameraný didakticky. Poskytuje množstvo návodov, ktoré boli navyše portované do rôznych programovacích jazykov a prostredí. Pre tento projekt dokonca existuje celkové prenesenie do českého jazyka. Vo všetkých uvedených príkladoch využíva OpenGL a je dobre dokumentovaný. Nevýhodou je jeho zameranie na postupné vytvorenie herného prostredia z pohľadu prvej osoby. Našťastie to začne užívateľ pociťovať až vo vyšších lekciách. Je škoda, že jeho primárnym cieľom nieje poskytovať návody ani príklady OpenGL. Zásadným nedostatkom je značná zastaralosť kódov a používaných knižníc. Ďalej je faktom, že s rastúcim číslom lekcie rastie aj množstvo nabaleného kódu na základnú kostru programu. Tým jeho prehľadnosť postupne degraduje. Momentálne sa však jedná o najlepší online zdroj o použití OpenGL pre začínajúcich užívateľov.

URL: http://nehe.gamedev.net

Gamerendering

Pomerne dobre spracovaná stránka s niekoľkými modernými príkladmi pokročilých grafických techník využívajúcich OpenGL. Ďalej sa na stránke vyskytujú zoskupené bezplatné odkazy na rôzne známe knižné publikácie. Z výberu spomeniem napr. knihu GPU Gems, alebo známu ShaderX. Stránka je bohužiaľ koncipovaná iba ako prehľad s miernym opisom a s odkazmi na rôzne techniky. Zdrojové kódy nie sú skoro nikdy dostupné v plnom rozsahu. Po väčšine sa jedná o fragmenty shaderov, zverejnené priamo v stránke. Kódy nie sú nijak didakticky ani systematicky vedené a pre začiatočníka sa jedná o prakticky nepoužiteľné informácie. Naopak pre pokročilého grafika predstavuje stránka zdroj mnohých nových informácií.

URL: http://www.gamerendering.com

GameTutorials

Zameranie tohto projektu je hlavne na vytvorenie herného sveta. Poskytuje podrobnejší a lepšie spracovaný popis k príkladom ako iné stránky. Príklady sú vedené pomerne dobre didakticky a sú vhodné aj pre začiatočníka. Dôsledkom toho je však skutočnosť, že väčšina príkladov je platená a tým pádom nepoužiteľná. Z pohľadu mojej práce sa príklady navyše zameriavajú na iný účel, aj keď k tomu používajú rovnakú technológiu. Príklady sú technologicky zastaralé, písané v jazyku C a tým aj menej prehľadné.

URL: http://www.gametutorials.com

Jerome Jouvieje

Stránka s početným množstvom návodov. K zverejneným príkladom je vždy uvedený podrobný popis. Zdrojové kódy sú k dispozícií. Prvé čo ale typického užívateľa odradí je použitý programovací jazyk a framework JOGL – Java for OpenGL. Ten nieje natoľko rozšírený a nemá takú podporu v komunite ako jazyk C++. Ďalšou nevýhodou nieje ani tak závislosť zdrojových kódov na externých knižniciach ako to, že tieto závislosti nie sú nikde popísané. Stránka samotná je dosť neprehľadná a pôsobí archaickým dojmom. Jednotnosť zdrojových kódov nebola zistená. S prihliadnutím k iným existujúcim zdrojom na internete venujúcim sa rovnakej tématike sa táto stránka môže radiť na popredné miesta.

URL: http://jerome.jouvie.free.fr

LightHouse3D

Podobne ako Clockwork coders, aj tieto návody sú zastrešené priamo pod hlavičkou OpenGL. Zameriavajú sa na základy GLSL. Príkladov nieje veľa. Sú ale podrobnejšie popísané a s názornými ilustráciami. Nanešťastie sú orientované viac teoreticky a uvedené ukážky kódov nieje možné stiahnuť a vyskúšať. Aj napriek tejto okolnosti poskytuje stránka dobrý základ pre priblíženie jazyka GLSL. Uvádzané kódy sú novšie ako v prípade projektu Clockwork coders a to vo verzii OpenGL 2.

URL: http://www.lighthouse3d.com

ZANIR – Marek Mižanin

Stránka s množstvom programov zameraná nielen na OpenGL a jazyk C++, ale aj Javu a DirectX 8. Obecne sú programy dobre spracované, vždy s popisom. Stránka je vedená ako blog a preto sa na nej ťažšie orientuje. Vzhľadom na použitý slovenský jazyk je výborná pre slovenských a českých užívateľov. Programy sú vhodnejšie pre pokročilejšieho užívateľa nakoľko sa takmer vždy jedná o náročnejšie techniky. Príklady nie sú didakticky smerované ani nieje braný ohľad na ich náväznosť. Zároveň sú zverejnené programy už technologicky prekonané. Aj napriek uvedeným nedostatkom sa tento internetový zdroj môže s prehľadom zaradiť na popredné miesta vo svete príkladov k OpenGL.

URL: http://zanir.wz.cz

OpenGL Code

Portál samotného OpenGL orientovaný pre vývojárov. Predstavuje jednu z najhoršie vytvorených stránok pre podporu knižnice. Ide o fórum na ktorom rôzny užívatelia s rôznou predstavou a zameraním uverejňujú odkazy na svoje stránky s obvykle jedným až tromi príkladmi, ktoré niekedy v minulosti vytvorili. Uvedené odkazy často vedú na zabudnuté, zastaralé a dlho neaktualizované projekty. Je možné, že existujú aj odkazy, ktoré vedú na weby udržované a aktualizované. Takýto odkaz som však v rámci hľadania našiel iba jeden. Z pohľadu programátora OpenGL túto stránku nemožno vôbec odporučiť.

URL: http://www.opengl.org/code

Ozone3D

Portál známy hlavne vďaka svojmu prepracovanému nástroju na zisťovanie vlastností grafických kariet – GPU CapsViewer. Na stránke sa nachádza aj niekoľko návodov na OpenGL a jazyk GLSL. Príklady sú však vždy od iného užívateľa. Nemajú jednotnú kostru a zoznam príkladov je vyberaný čisto náhodne. Programov je taktiež malé množstvo. Pre akéhokoľvek užívateľa začínajúceho v oblasti používania OpenGL sú návody na stránkach nevhodné. Niektoré techniky ale obsahujú dobrý teoretický popis a zdrojové kódy GLSL shaderov sú zobrazené priamo na stránke. To sa môže hodiť pokročilému užívateľovi. Celkovo by som však stránku zhodnotil ako nevhodnú.

URL: http://www.ozone3d.net

Paul's Project

Projekt sa zameriava prevážne na OpenGL. Obsahuje zhruba dve desiatky pokročilých príkladov. Tie nie sú smerované didakticky. Ide o príklady, ktoré autor zhotovil v rámci osobnej potreby. Programy sú vhodné pre pokročilého užívateľa, pretože nie sú zamerané priamo na OpenGL ako také. Používajú toto API na dosiahnutie zložitejších techník a efektov. Zdrojové kódy sú pomerne zložité a z časti zastaralé, majú však jednotnú štruktúru. Užívateľ sa preto po krátkom skúmaní dobre zorientuje. Ako zdroj OpenGL príkladov považujem tento projekt za veľmi dobrý, aj keď nevhodný pre začiatočníkov.

URL: http://www.paulsprojects.net

Root.cz – Grafická knihovna OpenGL

Seriál na serveri Root.cz prináša skutočný návod pre všetkých užívateľov rozhodnutých naučiť sa pracovať s OpenGL knižnicou. Návod je síce určený pre staršie verzie knižnice, ale ako základ postačí. Sú v ňom didakticky a postupne preberané hlavné črty OpenGL. Každý z cca. 30 návodov je podrobným popisom niektorej vlastnosti API. Ku každej časti seriálu sú priložené zdrojové kódy. Seriál je navyše v českom jazyku. Nevýhodou je zastaralosť kódov a fakt, že niektoré zdrojové kódy sú nedostupné. Navyše dostupné zdrojové kódy neobsahujú žiadny projektový súbor. Užívateľ musí byť sám dostatočne znalý, aby ich dokázal preložiť. Seriál je založený na teórií s pridruženými zdrojovými kódmi. Zdrojové kódy nie sú nikdy priamo vysvetľované a sú nejednotné. Povaha kódu je demonštrácia uvedenej teórie na komplexnejšom príklade, kde je možné ju uplatniť. Príklady nie sú synteticky vytvárané presne na preberanú látku. V každom prípade sa ale jedná o veľmi dobrý didaktický, aj keď hlavne teoretický, zdroj informácií.

URL: http://www.root.cz/serialy/graficka-knihovna-opengl

Song Ho Ahn

Stránka sa nevenuje jedine problematike OpenGL. Z nej však obsahuje zhruba desiatku podrobne a dobre napísaných návodov. Zdrojové kódy sú k dispozícií v jazyku C++ a ako okenné rozhranie je použitý GLUT. Príklady sú teda procedurálne. Napriek tomu, že sú príklady dobre spracované je ich jednotlivé zameranie vyberané náhodne bez ohľadu na akúkoľvek náväznosť či systém. Uvedené programy sú tak vhodnejšie skôr pre pokročilého užívateľa. Ten tu môže nájsť vysvetlených niekoľko zaujímavých techník. Na rozdiel od iných projektov sú tieto príklady novšie, súčasnejšie.

URL: http://www.songho.ca

Swiftless

Nový a technologicky moderný návod, ktorý vznikol v priebehu písania tejto práce. Poskytuje základy OpenGL od verzie 2.0. Návod sa vyznačuje poskytovaním celkového zdrojového kódu priamo v rámci textu stránok. Jednotlivé príklady sa snažia zachovať si maximálnu jednoduchosť. Príkladov je zverejnené veľké množstvo. So zvyšujúcim sa číslom lekcie sa ale stále viac vyhýbajú vysvetľovaniu technológie samotnej. Napriek tomu má každý príklad sprievodný text a je dobre komentovaný. Tutoriál je moderný. Začína na OpenGL 2.0 a siaha až na verziu 4.0. Poskytuje aj niekoľko základných príkladov do GLSL. Tento web je odporučenia hodný ako pre začiatočníkov tak aj pokročilejších užívateľov. Projekt presne zasahuje do oblasti, ktorej sa chce venovať táto práca. Príklady sú navyše jednotné a priamočiare. Tento projekt výrazne vedie nad všetkými tu uvedenými návodmi. Splňuje taktiež skoro všetko čo by som očakával od vlastnej práce. Poskytuje však kódy v príliš jednoduchej forme. To typický užívateľ pochopí až v momente, keď bude potrebovať vytvoriť vlastnú aplikáciu, ktorá bude určite zložitejšia. Z môjho pohľadu je to asi jediná nevýhoda. Projekt Swiftless je najlepším projektom so zameraním na OpenGL s akým som sa kedy stretol.

URL: http://www.swiftless.com

Video tutorials rock

Veľmi vydarený a originálny projekt pre podporu výuky OpenGL. Ako už z názvu vyplýva ide o video tutoriály, kde autor svoje počínanie vždy komentuje. Zameranie je na OpenGL samotné. Príklady sú vyberané veľmi rozumne od základných po zložitejšie. Zároveň na seba postupne nadväzujú. Keďže sa jedná o video-návod je celý proces od založenia projektu v MSVC cez písanie aplikácie až po jej preloženie vždy komentovaný. Techniky sú síce zastaralé, ale je to doposiaľ jediný návod zameraný priamo na výuku OpenGL ako takého. Pre pokročilého užívateľa tento projekt neprináša nič. Je určený výhradne začiatočníkom. Na záver ešte pripomeniem, že tutoriál neobsahuje žiadne použitie shaderov GLSL.

URL: http://www.videotutorialsrock.com

Zhodnotenie

Až na dve či tri výnimky sa žiaden návod nezameriava priamo na podporu výuky OpenGL. Bežne dostupné kódy sú vždy zastaralé s výnimkou projektu Swiftless. Ten však trpí prílišným zjednodušením problematiky a používaním zastaralej knižnice GLUT. Na druhú stranu sú príklady ostatných projektov niekedy až príliš zložité. Taktiež existuje len málo projektov s príkladmi pre dnes už vyžadované shadery GLSL v spracovaní pod OpenGL 2. V prípade dostupnosti zdrojových kódov nie sú tieto takmer nikdy jednotné. Dokonca aj v rámci odporúčaných projektov každý trpí niektorým z uvedených nedostatkov, najmä však použitou verziou knižnice OpenGL.

Vzhľadom ktomu, že výsledky tejto práce majú za cieľ zlepšiť úroveň užívateľa v používaní API OpenGL, je nutné aby prezentované príklady boli dostatočne technologicky aktuálne. Zároveň je nutné smerovať užívateľa k dobrým programovacím návykom a osvojeniu si použitia zložitejších externých knižníc pre vytvorenie požadovanej aplikácie. Tie sú v praxi nenahraditeľné. Nakoniec je veľmi dobré vytvoriť určitú oporu pri začiatkoch programovania, a to poskytnutím knižnice doplňujúcej najzákladnejšie úkony používané spolu s týmto rozhraním.

4 Technológia

Technológiou sa rozumie grafické API a programovací jazyk s prekladačom. Ten musí byť schopný zaistiť podporu požiadavkov vyplývajúcich z vytvárania nízkoúrovňových aplikácií pri použití tohto API a to minimálne prostredníctvom doplnkových knižníc a frameworkov.

Použitím programovacieho jazyka resp. prekladača, ktorý by bol úzko prepojený s OpenGL, by sa dosiahla vysoká efektivita pri tvorbe a užívaní vytvorených programov. Úzke prepojenie by sa najideálnejšie dosiahlo pomocou dostupnosti grafických príkazov priamo v rámci jazyka resp. prekladača, bez nutnosti použitia externých knižníc zaisťujúcich toto spracovanie. Jediným dnes dostupným jazykom spĺňajúcim tento požiadavok je skriptovací jazyk WebGL. Ostatné vyššie jazyky ako Java, C#, C++ nadobudnú túto funkcionalitu iba prostredníctvom externých knižníc. WebGL bohužiaľ podporuje iba podmnožinu funkcií OpenGL API a iba do verzie 2.0. Preto bol do zadania práce vybraný jazyk C++. Ten môže prostredníctvom dostupných externých knižníc získať prístup k plnej funkcionalite aj najnovšej verzie OpenGL.

Istou formou zaistenia dlhšieho časového horizontu na použiteľnosť vytváraného projektu je použitie technológie podporujúcej novodobú verziu OpenGL 3, alebo OpenGL 4. API samotné je však od svojho vzniku konštruované tak aby plne zaisťovalo spätnú kompatibilitu. Je preto rozumnejšie podporovať skôr najrozšírenejšiu, než najnovšiu verziu. Nové verzie poskytujú vždy len novú funkcionalitu. Nezlepšujú ani výkon ani nezjednodušujú použitie pôvodného rozhrania. Dokonca ani nikdy žiadnu funkcionalitu nenahradzujú. Taká je filozofia OpenGL. Verzia knižnice vyplývajúca zo zadania práce by tak mohla byť postačujúcou. Za posledný rok však OpenGL rapídne expandovalo vo funkcionalite. Počet vykonaných zmien bol najvyšší za celú jeho existenciu. Už len preto je v tomto ohľade lepšie analyzovať súčasný stav a prípadne verziu použitú pre implementáciu prehodnotiť.

Na začiatok je dôležité stanoviť požiadavky nutné pre smerovanie ďalšej analýzy. Tie budú vychádzať hlavne z osobnej skúsenosti s tvorbou nízkoúrovňových OpenGL aplikácií, ako aj z vlastností získaných analyzovaním existujúcich implementácií.

4.1 Požiadavky

Technológia musí kvôli umožneniu vytvárania požadovaných aplikácií zaisťovať:

- vytvorenie kontextu a okna OpenGL,
- správu užívateľských vstupov,
- načítanie obrázkov so zaisteným prístupom k ich pixelom, čo zabezpečí možnosť aplikácie textúr na objekty,
- načítanie modelov s prístupom k jednotlivým vrcholom a ich atribútom, umožní to využitie programovateľnej pipeline pre vytváranie pokročilých GLSL shaderov,
- načítanie a inicializácia shaderov GLSL,
- matematiku pre OpenGL, tzn. prácu s maticami, vektormi a trigonometrickými funkciami,
- prenositeľnosť,
- objektovo-orientovaný prístup.

Najvhodnejšie je, aby tieto vlastnosti zaisťoval programovací jazyk. Preto musia byť dostupné tieto vlastnosti či už pomocou komplexných externých nástrojov – frameworkov, alebo použitím jednoúčelových externých knižníc. Pri použití takýchto zdrojov by mala byť minimalizovaná ich závislosť vo vznikajúcich programoch.

Kvôli dosiahnutiu širokej použiteľnosti produktov, by nemali byť vybrané nástroje závislé ani na architektúre ani na platforme. Objektovo orientovaný prístup môže zase zaručiť lepšiu prehľadnosť a ľahšiu rozšíriteľnosť kódu.

4.2 Jazyk C++

Jazyk C++ vznikol z najpoužívanejšieho jazyka na svete C. Oproti jazyku C prináša zapracované inovatívne myšlienky objektového paradigma a zjednodušenie programovania. Ďalej má programátor možnosť voľby procedurálneho, alebo objektovo-orientovaného modelu. Procedurálny model bol zachovaný v dôsledku naviazanosti na materský jazyk C.

Jedná sa o jediný vyšší nízko-úrovňový jazyk. Týmto kladie na programátora vysokú prácnosť. Poskytuje iba základné programovacie konštrukty. K rozšíreniu podpory, vo forme zjednodušenia často používaných úkonov, nikdy nedošlo. Takúto podporu, najmä prostú prácu so súbormi a reťazcami, ponúkajú dnes už bežne rôzne moderné programovacie jazyky. Najznámejším takýmto predstaviteľom je Java. Rozšírenie C++ v podobe knižnice STL je síce výborné, pre prácu s nízko-úrovňovým grafickým API však nepoužiteľné. K jazyku C++ existuje veľký počet prekladačov pre rôzne platformy a vývojové prostredia. Jazyk samotný má týmto spôsobom zaistenú veľmi dobrú prenositeľnosť medzi rôznymi architektúrami. Kvalita veľkej väčšiny prekladačov je bohužiaľ na nízkej úrovni a nieje zaručená ani v prípade

komerčných verzií. Najlepším dostupným je unixový prekladač GCC, ktorý je veľmi dobre prepracovaný.

Architektúra rozhrania OpenGL bola navrhovaná pre prácu s jazykom C. Preto je C/C++ najpoužívanejším a momentálne aj najrozšírenejším jazykom pre prácu s týmto API. Aj napriek jeho nespočetným nedostatkom je práve tento dôsledok rozhodujúcim faktorom pre jeho nasadenie v mojich OpenGL aplikáciách.

Programovanie je priamočiare. Priama podpora OpenGL síce existuje, ale je slabá a dostupná iba vo forme štandardnej externej knižnice. Podporovaná je iba dostupnosť verzie OpenGL 1.1 a to takmer vo všetkých prekladačoch. Aj najpoužívanejšie netriviálne operácie ako napr. načítanie súborov sú však v tomto jazyku zložité a zabezpečujú sa obvykle prostredníctvom pridaných externých knižníc. Správa týchto zdrojov a ostatných prídavných prvkov je necentralizovaná, čo spôsobuje ďalšie problémy s používaním vytvorených aplikácií. Knižníc a frameworkov existuje na internete k voľnému používaniu celá rada. Tieto produkty sú často-krát nekvalitné a bez poriadnej selekcie väčšinou nepoužiteľné.

Z požiadavkov uvedených v kapitole 4.1 splňuje samotný jazyk len prenositeľnosť a objektové paradigma. Síce obsahuje v rámci väčšiny prekladačov podporu OpenGL 1.1, tá ale nemusí byť dostupná vždy. Navyše je nepoužiteľná pre akékoľvek dnešné aplikácie. Žiadnou ďalšou funkcionalitou nedisponuje C++ ani priamo v rámci jazyka ani v rámci štandardne distribuovaných knižníc. Vzhľadom na jeho rozšírenosť a dlhú tradíciu však existuje veľká pravdepodobnosť, že požadované vlastnosti už niekto implementoval a budú voľne dostupné v podobe rozširujúcej knižnice, alebo frameworku.

4.3 OpenGL

Knižnica je vo všetkých smeroch sebestačná. Je nezávislá ako na iných knižniciach, tak aj na architektúre či platforme. Z tejto nezávislosti však plynú rôzne obmedzenia na podporované vlastnosti. Nech by sa to zdalo pri takejto knižnici akokoľvek praktické, nedisponuje táto ani jednou požiadavkou definovanou v kapitole 4.1 s výnimkou prenositeľnosti. Knižnica nedisponuje žiadnym načítaním obrázkov pre textúry, ani načítaním modelov pre geometrické údaje. Jediné vlastnosti ktorými disponuje sa týkajú samotného vykresľovania a nastavovania stavu kontextu. Bohužiaľ sú tieto obmedzenia nutné pre použiteľnosť knižnice nielen v rámci rôznych platforiem, ale hlavne v rámci rôznych hardvérových architektúr. Aj napriek nesplneniu žiadnej z uvedených technologických požiadaviek ju nemožno nijak vylúčiť či nahradiť, pretože tvorí cieľ práce. Všetká táto funkčnosť tak musí byť prenesená na bedrá programovacieho jazyka a jeho doplnkových knižníc.

Ako už bolo povedané, neexistujú žiadne závislosti na iných produktoch, ktoré by bolo možné analyzovať. V súčastnosti existujú štyri verzie OpenGL. Celý ďalší rozbor tak zahŕňa preskúmanie vlastností a poskytovaných možností jednotlivých verzií, ich rozšírenosť medzi užívateľmi, dostupnosť pre vývojárov a použiteľnosť na súčasnom hardvéri.

4.3.1 OpenGL 1.x

OpenGL 1 je najstaršia verzia knižnice OpenGL. Hoci vznikla pred dvadsiatimi rokmi, hardvérová podpora od významných výrobcov je zaisťovaná posledných pätnásť rokov. Ešte v súčastnosti je táto verzia stále veľmi rozšírená a používaná. Zohľadnením súčastného trendu je možné očakávať, že sa bude ešte aspoň 5 rokov aktívne v širokej miere používať.

Táto verzia používa na vykresľovanie tzv. fixnú pipeline. V dôsledku toho, že OpenGL 1.x poskytuje len fixnú pipeline nieje možné ľubovolne meniť zobrazovacie vlastnosti prostredia, ale ich len v poskytovanej miere modifikovať. Nieje teda možné zasahovať do jej procesov spracovania. Často sa toto označuje výstižne ako čierna skrinka. Editácia je však dostatočne široká, takže rozsah a kvalita efektov, ktoré je možné vytvoriť je aj v súčasnej dobe postačujúca.

Samozrejme, že v rámci revízií vychádzali zlepšenia tejto verzie. Postupne prišla možnosť zasahovať do procesu spracovania vrcholov a fragmentov prostredníctvom assembleru od revízie 1.4 a prostredníctvom vyššieho jazyka GLSL od revízie 1.5, a tak sa vízia čiernej skrinky postupne vytrácala. Takisto od revízie 1.5 je možné uchovávanie geometrie v pamäti GPU. Prinieslo to obrovské pozitívum v rýchlosti spracovania. Odľahčil sa prenos dát medzi CPU a GPU. Tieto zásahy do procesu spracovania boli možné zatiaľ iba prostredníctvom ARB extenzií.

Keďže táto verzia je prvá, je najjednoduchšia. Pre nového programátora je preto najvhodnejším nástrojom na začiatok používania. Poskytuje širšiu užívateľskú podporu pri práci s grafikou. Túto verziu je možné v plnej miere používať v rámci kompatibiného módu na každom dnešnom relevantnom hardvéri a verzii OpenGL. To jej zabezpečuje použiteľnosť na dnešných všetkých bežne používaných grafických kartách ako v stolných, tak aj v prenosných počítačoch. Vytvorené programy je možné používať na všetkých súčasných podporovaných architektúrach a platformách. Verzia 1.x je najlepšie dostupná zo všetkých verzií OpenGL pre programovacie jazyky, prekladače, samotných programátorov, ale hlavne užívateľov.

4.3.2 OpenGL 2.x

Hlavný prínos novej verzie OpenGL spočíval v tom, že sa potlačila fixná pipeline, a do procesov spracovania celých primitív už bolo možné pristupovať pomocou GLSL 1.1 ustanovenej v jadre knižnice. V OpenGL 2 sú zachované rovnaké funkcie a kontext ako v prechádzajúcej verzii. Pri použití, aplikácia nepozná rozdiel medzi týmito dvomi verziami. Zásadne sa zmenili niektoré názvy príkazov pracujúcich so shadermi.

Dnes je najrozšírenejšia a najpoužívanejšia. Trochu nižšia je použiteľnosť v aplikáciách na grafických kartách v bežných prenosných počítačoch v porovnaní s verziou OpenGL 1. Rovnaká je aj programovacia a finančná dostupnosť pre užívateľov a programátorov ako v prípade verzie 1.

4.3.3 OpenGL 3.x

Ďalšia verzia priniesla veľa inovatívnych zmien, ale zostala zachovaná kompatibilita všetkých príkazov z predchádzajúcich verzií. Plná kompatibilita a nové príkazy sú prístupné pre OpenGL a GLSL v profile – Compatibility. Popri tomto ešte existuje druhý profil – Core. Pri použití profilu Core sú zachované iba vybrané funkcie serveru OpenGL, hlavne nastavenia stavu a príkazy hromadného vykresľovania. Používa rozdielny kontext okna ako predchádzajúce verzie OpenGL 1 a OpenGL 2. Spracovanie vrcholov, fragmentov aj primitív je programovateľné. Výpočty aj vykresľovanie je plne závislé na programátorovi, čo mu dovoľuje vytvárať akékoľvek spracovanie objektov scény.

Došlo k prečisteniu OpenGL od redundantných príkazov vytvorených v rámci OpenGL 1 k užívateľskému pohodliu. Prečistením došlo k optimalizácii výkonu. API slúži už iba na nastavenie grafického kontextu, nahranie geometrie do grafickej karty a preposlanie údajov na spracovanie do shaderov. Došlo k nárastu zložitosti používania a vytvárania shaderov. Aj keď zavedením profilov malo dôjsť k zvýšeniu užívateľskej prehľadnosti, praktickým dôsledkom bolo jej rapídne klesnutie.

V praxi sa táto verzia zatiaľ často nepoužíva. Je využívaná hlavne vo vývoji a testovaní. V krátkom čase, ale dôjde k jej masovému rozšíreniu. Na grafických kartách bežne používaných prenosných počítačov je takmer nepoužiteľná. V poslednom čase sa podstatne zvýšila použiteľnosť na grafických kartách stolných počítačov. V súčastnosti je dobre dostupná pre programovanie, ale slabšie dostupná pre normálnych užívateľov. Finančná náročnosť je v podstate porovnateľná s OpenGL 2.x.

4.3.4 OpenGL 4.x

V najnovšej, posledne vydanej verzii vznikli nové programovateľné jednotky. Hlavný prínos má Teselačný shader, ktorý je distribuovaný v niekoľkých hardvérových jednotkách. Celkové spracovanie geometrie je už plne závislé na programátorovi. Zostala zachovaná kompatibilita všetkých predchádzajúcich verzií OpenGL i GLSL v rámci špeciálnych profilov.

OpenGL 4 ponúka nové objekty - Pipeline, ktoré simulujú grafickú pipeline. Pozostávajú z programovateľných shaderov a sú schopné nahradiť celú pipeline grafickej karty. Z predchádzajúcich verzií zostáva zachované iba poradie spracovania. V prípade vynechania Geometrického, alebo Teselačného shaderu sa pracuje s ich fixným variantom.

V prenosných počítačoch je použitie OpenGL 4 raritou. Grafické karty podporujúce túto verziu sú vzácne aj v prípade ich použitia v stolných počítačoch. Programátorská aj finančná dostupnosť je slabá, z dôvodu nutnosti zakúpenia najnovšej grafickej karty.

5 Frameworky a knižnice

Z dôvodu nesplnenia žiadnej technologickej požiadavky samotným jazykom C++ je potrebné analyzovať k nemu dostupné frameworky a knižnice pre doplnenie chýbajúcej podpory. Tých je samozrejme veľké množstvo. Nieje možné bez podrobnejšieho preskúmania viacerých možností určiť priamu voľbu. Existujú ako v komerčnej tak v podobe OpenSource. Z komerčných produktov sú najvýznamnejšie Unigine a Cry Engine. Cena zdrojových kódov týchto grafických strojov sa pohybuje okolo $100 000. Práca ale v nasledujúcich kapitolách analyzuje súčasné nekomerčné možnosti. Postupne budú rozobrané frameworky od veľkých komplexných herných a grafických strojov, cez multimediálne, po špecificky zamerané na správu okien, spracovanie rastrových obrázkov, 3D geometriu a matematiku. Popis sa zameriava na oblasti vytýčené v požiadavkách v kapitole 4.1.

5.1 Herné a grafické stroje

Slovo stroje je tu používané vo význame „engine“. Jedná sa o komlexné programy zabezpečujúce rozsiahlu funkčnosť používanú pri práci s grafikou ako napr. načítanie modelov, načítanie textúr, správu shaderov, správu scény. Z možných variantov som vybral jeden herný a jeden grafický stroj, tzv. scenegraph. Konkrétne bude analyzovaný open-source herný stroj OGRE a open-source grafický stroj OpenSceneGraph.

5.1.1 OGRE

Komplexný a významný herný stroj OGRE je jeden z najkvalitnejších a najpoužívanejších voľne dostupných strojov. Používa objektovo orientované programovanie. Je použiteľný pre platformy Windows, Linux a MacOS. Z programátorského hľadiska je OGRE nezávislé na externých knižniciach. Nástroj vznikol pred desiatimi rokmi, a od vzniku je pravidelne

vylepšovaný. Momentálne je v širokej miere rozšírený, preto je možné predpokladať ešte jeho dlhú aktívnu existenciu.

Engine je postavený na podpore OpenGL vo verzii 2. Podporuje navyše D3D a software rendering. Jeho použitím sa zabezpečia prerekvizity ako:

- vytvorenie vykresľovacieho okna
- správa užívateľských vstupov
- načítanie textúr vo formátoch JPG, PNG, TGA a špeciálnych textúr DDS
- načítanie modelov prostredníctvom pluginu, možno poľahky nájsť i plugin načítavajúci najnovší formát modelov – Colladu
- funkcie pre matematiku a fyziku
- správa scény – geometrie a osvetlenia

Stroj nepodporuje všetky typy shaderov. Medzi podporované patria iba vertex a fragment shadere, nepodporované sú geometrické a teselačné shadere. Vertex a fragment shader je podporovaný v rôznych programovacích jazykoch ako je GLSL, HLSL, Assembler. Je umožnený priamy prístup k nastaveniam fixnej pipeline OpenGL. Stroj disponuje nezávislým rendererom, označovaným ako Render-Wrapper, ktorý môže byť eventuálne urýchľovaný niektorým z grafických API (D3D, OGL).

Najmarkantnejšou nevýhodou stroja je zabránenie zásahu do procesu vykresľovania. Nie je umožnený žiadny priamy prístup k funkcionalite OpenGL API. Táto vlastnosť ho predurčuje na vytváranie scén a virtuálnych svetov, avšak pre tvorbu návodov ilustrujúcich použitie OpenGL príkazov je úplne nepoužiteľný.

URL: http://www.ogre3d.org

5.1.2 OpenSceneGraph

Jedná sa o komplexný a často využívaný objektovo orientovaný scene-manager. Na vykreslovanie využíva OpenGL vo verzii 2, konkrétne ale jeho fixnú pipeline. Funguje na platformách Windows, Linux a MacOS. Projekt sa už roky vylepšuje a dá sa predpokladať jeho využitie ešte v dlhom časovom horizonte. Zabezpečuje a umožňuje nasledujúce funkcie:

- vytvorenie vykresľovacieho okna,
- správu užívateľských vstupov,
- renderovanie do textúry,
- načítanie obrázkov textúr vo formátoch JPG, PNG, TIFF a špeciálnych DDS,
- načítanie modelov vo formáte OBJ, 3DS, VRML a MD2,
- fyzikálne výpočty a podpora animácií.

Scene-manager obsahuje množstvo funkcií, ktoré zjednodušujú prácu so scénou a jej vykreslením. Hoci OSG ponúka viacero možností vo svojom jadre ako OGRE, chýba k nim adekvátna dokumentácia, ktorá by popisovala význam a spôsob využitia týchto možností. Vykresľovacia platforma je OpenGL, neumožňuje však zásah do procesu vykresľovania. Umožnená je editácia prístupných nastavení a ovplyvňovanie výsledku vykreslenia pomocou shaderov. Podporované sú len vertex a fragment shadere v jazykoch CG a GLSL. Po preskúmaní vlastností scene-managera je nutné konštatovať, že jeho použitie v procese tvorby tutoriálov je nevhodné.

URL: http://www.openscenegraph.org

Po úvodných dvoch komplexných frameworkoch bola ďalšia pozornosť venovaná, menším knižniciam. Tieto poskytujú v prevažnej miere vždy jednu konkrétnu funkcionalitu použiteľnú v programoch OpenGL. Žiadna z týchto knižníc neposkytuje podobnú funkcionalitu, akú majú vyššie uvedené frameworky. Je ale možné očakávať, že vďaka neposkytovaniu toľkej funkcionality nebudú zasahovať do procesu vykresľovania. Z tohto dôvodu sa môže stať sada menších knižníc prospešnejšia na dosiahnutie môjho cieľa ako jeden komplexný framework.

Na druhú stranu sa použitím sady knižníc zvyšuje závislosť, znižuje prenositeľnosť. Pre začiatočníka môžu vzniknúť aj problémy s prekladom takéhoto projektu. Preto je pri ich výbere potrebné zamerať sa na prenositeľnosť jednotlivých knižníc a ich funkcionalitu. Čím bude širšia funkcionalita použitej knižnice, tým menej knižníc bude potrebné použiť. Nemenej dôležité je brať v úvahu aj ich potenciálny vývoj v budúcnosti.

5.2 Multimediálne frameworky

Pojmom multimediálne frameworky sú myslené knižnice zabezpečujúce širšiu sadu funkcií. Nie sú špecificky zamerané na jednu funkcionalitu. Popri podpore správy okien, poskytujú často načítanie textúr, matematiku, prácu so sieťou a správu vstupov. Neposkytujú však ucelené rozhranie pre správu scény, ako je to v prípade herných strojov. Zo širokej ponuky ďalej uvádzam len analýzu produktov SFML, Clanlib, GLT a GLFW.

5.2.1 SFML – Simple and Fast Multimedia Library

SFML je multimediálna knižnica podobne ako SDL, ktorá obsahuje množstvo podporných funkcií nielen pre prácu s grafikou, ale aj so zvukom a sieťou. Od SDL sa však úplne odlišuje. Hlavný rozdiel je, že SFLM je prehľadná a objektovo-orientovaná knižnica, zatiaľčo SDL je staršia procedurálne-orientovaná knižnica, čo ju zneprehľadňuje.

Knižnica podporuje vytváranie okien s kontextom OpenGL 3 a vyšším, avšak iba v COMPATIBILITY profile. Podpora CORE profile neexistuje. Vytváranie CORE aplikácie je aj napriek tomu možné pri striktnom dodržiavaní špecifikácie. Samotné OpenGL programátorovi neošetruje používanie nepovolených príkazov. Je teda možné použiť akékoľvek príkazy hoci aj z verzie OpenGL 1. Objektovo-orientovaná knižnica podporuje systémové časovače nezávisle na platforme a architektúre. Podporuje zároveň aj multi-threading a multi-windowing spracovanie.

Medzi základné funkcie používané pri práci s OpenGL, ktoré knižnica ponúka patrí:

- správa okien a vstupov,
- spracovanie udalostí, aj RT kontrola tlačidiel klávesnice a pohybov myši, podpora joysticku
- načítavanie obrázkov v najpoužívanejších formátoch – BMP, JPG, PNG, TGA, PSD a dokonca DDS,
- správa fragment shaderov ako post-process efektov,
- podporuje písanie textu na obrazovku,
- načítavanie audio stopy, načítavanie množstva hudobných formátov vrátane RAW, WAW, MP3 a OGG,
- práca so sieťou a sieťovými paketmi.

Ako nedostatok možno knižnici pripísať absenciu načítavania modelov a podporu pre správu ostatných typov shaderov. Knižnica tiež neobsahuje žiadne funkcie, ktoré by uľahčili prácu s matematikou.

Vďaka objektovému spracovaniu, je knižnica prehľadná a tým pádom dobre použiteľná pri vytváraní jednoduchého a prehľadného kódu. SFML má veľmi precízne

vypracovaný návrh a vynikajúcu implementáciu. Nemenej dôležitou súčasťou každej knižnice je jej dokumentácia. V prípade SFML je dokumentácia stručná a prehľadná, nezachádza do zbytočných podrobností. K podrobnému skúmaniu funkcionality sú dostupné zdrojové kódy.

S knižnicou je príjemná a jednoduchá práca ako pre začiatočníka, tak aj pre pokročilého programátora grafiky. Zaisťuje prístup od nízko-úrovňových funkcií OpenGL až po vysoko-úrovňové funkcie ako napr. možnosť vkladania textov na obrazovku a prácu so spritami.

URL: http://www.sfml-dev.org

5.2.2 CLANLIB

Podobne ako SFML aj CLANLIB je multimediálna multiplatformná knižnica, ktorá vykresľovanie zaisťuje pomocou OpenGL 1, OpenGL 3, Direct3D 9, Direct3D 10, alebo pomocou software renderingu. Hoci je knižnica objektovo-orientovaná, nepodporuje menné priestory. Všetky príkazy vyžadujú zadávanie prefixu, čo zneprehľadňuje použitie.

Podporuje načítavanie obrázkov vo formátoch JPG a PNG, načítavanie textových súborov, XML, CSS, zdrojov (resources) a prácu s maticami, vektormi. Ďalej podporuje výpočty nad základnými geometrickými primitívami ako úsečka, štvorec, kruh, elipsa a iné. Dokáže pracovať so zvukovými stopami a sieťou. Počas jej štúdia nebola zistená podpora načítavania modelov a priama podpora a správa shaderov. Nepodporuje ani priamu prácu s príkazmi OpenGL.

Knižnica je stále aktualizovaná a vyvíjaná do súčasnosti. Niektoré jej prirodzené funkcie sú na vytvorenie demonštračných príkladov na podporu výuky OpenGL nevhodné. Jedná sa hlavne o to, že jednotlivé grafické rozhrania zapúzdruje do špeciálnych tried, čím priamo špecifikuje ich možnú funkcionalitu. Preskúmaním dokumentácie a odskúšaním niekoľkých príkladov vyšlo najavo, že knižnica vytvára pomerne neprehľadný kód a je komplikovaná pre začiatočníka.

URL: http://clanlib.org

5.2.3 GLT – OpenGL C++ Toolkit

Jedná sa objektovo-orientovaný koncept, ktorý poskytuje správu okien a vstupov cez GLUT. Použitie GLUTu však nieje možné vo verzii OpenGL 3 a vyššej. Jeho pomocou je možné načítavať textúry vo formátoch PNG, TGA, PPM a BMP, ale len prostredníctvom „wrapper objektov“. Nieje možné načítanie samotných obrázkov, čím sa znemožňuje vytváranie textúr s ľubovolnými nastaveniami. Funkcionalita knižnice je rozšírená aj na prácu s matematikou, konkrétne prácu s maticami a vektormi. Poskytuje implicitné modely.

Podpora primaych vlastností knižnice OpenGL je zaistená len do revízie OpenGL 1.3. Knižnicu je však možné použiť súčasne s OpenGL do verzie 2, prípadne vyššej verzie v COMPATIBILITY profile. V súčasnosti sa knižnica ďalej nevyvíja. Posledná revízia je z novembra 2002. Knižnicu je možné ľahko začleniť do projektu a používať vďaka jej výbornému spracovaniu v C++. Dnes je však už zastaralá a môže slúžiť len ako zdroj informácií pre vytváranie podobnej knižnice štúdiom jej zdrojového kódu.

URL: http://www.nigels.com/glt

5.2.4 GLFW – GL Framework

Jedná sa o multiplatformnú procedurálnu knižnicu, ktorá umožňuje správu okien a užívateľských vstupov. V súčasnej verzii podporuje načítavanie obrázkov vo formáte TGA 1. Knižnica je pravidelne aktualizovaná. V posledných rozšíreniach sa podporili kontexty OpenGL verzií 3 a vyšších, a to dokonca s možnosťou výberu medzi profilom CORE, alebo COMPATIBILITY. Pri vytváraní kontextu okna je možné špecifikovať bitovú hĺbku vybraných bufferov. Nepodporuje načítavanie modelov ani neobsahuje žiadne implicitné modely. Podpora spracovania matematických operácií a spáva shaderov taktiež chýba. Pomocou podpory multi-threadingu je možné využiť paralelné viac-vláknové spracovanie.

Neustále vyvíjanie knižnice zabezpečuje jej stálu aktuálnosť. Na internetových stránkach knižnice je dokonca k nahliadnutiu plán jej budúceho vývoja – Roadmap. V ňom sa dá zistiť, že niektoré dôležité funkcie knižnice budú v ďalších verziách odstránené. Jedná sa hlavne o odstránenie podpory načítavania obrázkov a multi-threadingu. Spočiatku veľmi dobré spracovanie, aj keď procedurálne, stála aktualizácia a funkcie knižnice ju stavali do pozície adepta k použitiu v chystanom projekte. Ale práve odstránenie pre projekt dôležitých funkcií v budúcich verziách ju z tejto pozície zosadilo.

URL: http://www.glfw.org

5.3 Správa okien

Ide o špecificky zamerané knižnice k podpore správy okenného systému OpenGL. Tieto zavše obsahujú aj pridanú funkčnosť v podobe poskytovania implicitných modelov alebo načítavanie jedného typu obrázkov. Existuje veľké množstvo týchto knižníc. Takmer všetky sú open-source. Často-krát sú nekvalitné. Venoval som sa analýze notoricky známych knižníc ako GLUT a FREEGLUT, ale aj menej známych knižníc ako OGLWFW a GLOW.

5.3.1 OGLWFW – OpenGL Window Framework

Zastaralý framework, ktorého vývoj bol zastavený už v roku 2005. K podpore OpegGL využíva knižnicu GLEE, ktorá je z obdobia vývoja projektu. Použitá verzia GLEE podporuje iba staršie verzie OpenGL, maximálne do verzie 2. OGLWFW je multiplatformný, objektovo-orientovaný framework, podporujúci správu okien a užívateľských vstupov. Okrem tejto správy neposkytuje ďalšie funkcie. Chýba v ňom akákoľvek podpora shaderov, matematických operácií, načítavanie obrázkov prípadne modelov. Podporuje však multi-window, zabezpečujúci súčasné vytváranie viacerých okien a multi-threading.

URL: http://oglwfw.sourceforge.net

5.3.2 GLUT – OpenGL Utility Toolkit

Na podporu vytvárania jednoduchších aplikácií s využitím OpenGL je najznámejšia a najpoužívanejšia procedurálne orientovaná knižnica GLUT. Podporuje vytváranie kontextu OpenGL do verzie 2. Pre vyššiu verziu nedokáže vytvoriť kontext. Návrh knižnice je prispôsobený pre použitie v OpenGL.

Pomocou knižnice je možné spravovanie okien a vstupov, neumožňuje však správu shaderov, matematické operácie, ani načítavanie obrázkov. Obsahuje v sebe implicitné modely. Je multiplatformná a prenositeľná. Knižnica nebola aktualizovaná už vyše desať rokov. Posledná dostupná revízia je z roku 2000. V tom období bola k dispozícii ešte len verzia OpenGL 1.3. Zachovaním architektúry OpenGL do verzie 2, sa predĺžila jej použiteľnosť až do tejto verzie. Novodobá aktualizácia knižnice sa našťastie odohráva v podobe projektu FreeGLUT.

URL: http://www.opengl.org/resources/libraries/glut

5.3.3 FREEGLUT

Procedurálna knižnica, ktorá je rozšírením a upravením pôvodnej knižnice GLUT. Podporuje kontext OpenGL 3 a vyššej verzie. Umožňuje výber z profilov CORE a COMPATIBILITY. Podporuje mutiplatformnú správu okien a užívateľských vstupov. Programátorovi dovoľuje úplnú špecifikáciu bufferov, s ktorými má byť kontext okna vytvorený, ako napr. hĺbkový, akumulačný, stencil, double/single, multisample buffer atp.

Je na škodu, že FREEGLUT nerozširuje GLUT o ďalšie funkcie, ako napr. spracovanie vyšších matematických operácií, správu shaderov, načítanie obrázkov. Knižnica je len akýmsi updatom pôvodného GLUTu. Nieje zabezpečený, ani multi-window, ani multi-threading. V súčasnosti je knižnica aktívne a neustále vyvíjaná. Vďaka silným základom, ktoré jej zabezpečila materská knižnica GLUT je možné predpokladať jej dlhú aktívnu existenciu a podporu aj v budúcnosti. Knižnica je veľmi dobre spracovaná a kvalitným a prepracovaným spôsobom podporuje novodobé žiadané vlastnosti kladené na okenný systém OpenGL.

URL: http://freeglut.sourceforge.net

5.3.4 GLOW

Framework je zastaralý, ale objektovo orientovaný. Podporuje OpenGL iba do verzie 1.3. Zameriava sa na vytváranie kontextu a okien pre OpenGL, a hlavne na podporu GUI. Nepodporuje načítavanie obrázkov, modelov ani matematické operácie. GLOW je multi-platformný a prenositeľný. Posledná revízia je z roku 2000. Stojí na báze GLUTu, preto je nevhodný pre použitie vyššej verzie OpenGL ako 2.1.

URL: http://glow.sourceforge.net

5.4 Spracovanie rastrových obrázkov

V ďalšom texte budú uvedené a rozobrané knižnice, ktoré poskytujú načítavanie obrázkov. Bude kladený dôraz na jednoduchosť ich používania, aktuálnosť, množstvo a typ podporovaných obrázkových formátov a použiteľnosť na rôznych platformách. Analyzované budú rozsiahle knižnice zabezpečujúce najširšiu podporu ako DevIL, FreeImage, GTL a SOIL.

5.4.1 DevIL – Developer's Image Library

Moderná a udržiavaná knižnica DevIL je špeciálne zameraná na načítavanie obrázkov pre rôzne grafické platformy – OpenGL, DirectX, Allegro, Windows GDI. Syntax príkazov je plne v duchu syntaxe OpenGL. Obdobne ako u OpenGL je potrebné pri práci najprv vytvorenie objektu, jeho naviazanie, nastavenie parametrov prostredia a objektu a až potom je ho možné používať opätovným nasadzovaním. Pri súčasnom vytvorení viacerých objektov je vždy aktívny iba posledne naviazaný objekt, a teda všetky operácie sa vykonávajú nad týmto jedným či už obrázkovým, alebo texturovacím objektom.

Knižnica podporuje načítavanie obrovského množstva rastrových i vektorových obrázkových formátov. Sú medzi nimi samozrejme najpoužívanejšie formáty ako JPG, PNG, TGA, i najoptimalizovanejšie ako DDS. Konkrétne podporované vlastnosti v rámci DDS neboli zistené. Podporuje HDR obrázky formátov HDR a OpenEXR. Okrem načítania prostých obrázkov podporuje aj načítavanie vrstiev, generovanie mipmáp, zobrazovanie animácií a načítavanie kubických máp z jedného obrázku v preddefinovanom formáte.

Medzi jednotlivými platformami je prenositeľná. Knižnica je rozsiahla a má zložitejší kód. Funkcionalita je zaisťovaná pomocou niekoľkých externých knižníc. DevIL sa stále aktívne vyvíja a aktualizuje. Vďaka jej prepracovaniu a poskytovanej funkcionalite je pravdepodobné, že sa bude aktívne využívať aj v budúcnosti.

URL: http://gpwiki.org/index.php/DevIL

5.4.2 FreeImage

FreeImage predstavuje modernú a aktuálnu knižnicu zameranú na nezávislé načítavanie obrázkov. Jej použitie je možné ako v kombinácii s OpenGL, tak i v kombinácii s DirectX API. Je veľmi podobná prechádzajúcej knižnici DevIL. Rovnako ako DevIL poskytuje veľké množstvo podporovaných formátov od najpoužívanejších – JPG, PNG a TGA po vysoko-optimalizované DDS. Takisto ako v prípade knižnice DevIL, ani v prípade FreeImage nebola zistená podpora vlastností formátu DDS. Podporuje HDR obrázky formátov HDR a OpenEXR. Nepodporuje tvorbu mipmáp, zobrazovania animácií ani načítavania kubických máp. Avšak podporuje užívateľské rozširovanie prostredníctvom pluginov.

Funkčnosť knižnice je zaistená integráciou niekoľkých externých knižníc. FreeImage je medzi jednotlivými platformami prenositeľná. Existuje k nej vynikajúca dokumentácia. V súčasnosti je pravidelne aktualizovaná a ďalej vyvíjaná. Rozhranie a použitie je prirodzenejšie ako v prípade knižnice DevIL.

URL: http://freeimage.sourceforge.net

5.4.3 GTL - The Game Texture Loader

Ako už sám názov napovedá,knižnica je priamo zameraná na zjednodušenie tvorby textúr. Jej integrácia je možná do rôznych grafických API. Jedná sa o objektovo-orientovanú knižnicu, ktorá podporuje formáty TGA, BMP, PNG, JPG a DDS. Vrámci DDS podporuje kompresiu DXT 1 - 5 a 3Dc.

Knižnica je malá a jednoduchá s niekoľkými presne cielenými užitočnými funkciami. V podstate umožňuje iba načítanie obrázku zo súboru, vytvorenie triedy nad načítanými dátami a poskytnutie základných informácií o obrázku spolu s ukazovateľom na pixely. Knižnicu vytvoril autor frameworku OGLWFW.

Hoci bol vývoj knižnice zastavený v roku 2007, pre obrázky základných formátov ako JPG, PNG a TGA to nieje prekážkou aktuálnosti, vzhľadom k tomu, že špecifikácia týchto formátov sa nezmenila aspoň 15 rokov. Knižnica je nezávislá a multiplatformná. Vďaka jej jednoduchosti je ľahko použiteľná.

URL: http://tgtl.sourceforge.net

5.4.4 SOIL - Simple OpenGL Image Library

Simple OpenGL Image Library je procedurálna knižnica určená na jednoduché načítavanie základných typov obrázkov. Podporuje načítavanie formátov JPG, PNG, BMP, TGA, PSD a optimalizovaného DDS. Načítava komprimované i nekomprimované DDS a to v kompresiách DXT 1 – 5. Nepodporuje iba formát 3Dc. Podporuje načítavanie HDR a kubických máp v jednom obrázku. Je to malá knižnica, dobre integrovateľná do projektov. Ovládanie je tiež jednoduché. I napriek tomu, že je vývoj bol zastavený v roku 2008, je pre bežné formáty obrázkov JPG, PNG a TGA plne použiteľná a stále aktuálna.

URL: http://www.lonesock.net/soil.html

5.5 3D geometria

Počas prípravnej fáze projektu bolo odskúšaných niekoľko model-loaderov. Popis všetkých skúmaných loaderov presahuje rámec tohto textu. Súhrnne je možné konštatovať, že vo valnej väčšine sa jednalo o loadery vytvorené k použitiu v nízko-úrovňových OpenGL aplikáciách, podporujúce vždy práve jeden formát a to buď 3DS, alebo OBJ. Model-loadery načítajúce formát OBJ vykazovali všeobecne vyššiu spoľahlivosť ako 3DS. To je zrejme spôsobené každoročnou zmenou špecifikácie tohto formátu. OBJ loader GLM je dobre spracovaný, ale je procedurálny a má problém s korektnou prácou s textúrovacími súradnicami. Zároveň používa pre vykreslovanie OpenGL immediate mode, čo je nežiadúce. OBJ loader z portálu robthebloke týmito nedostatkami netrpí. Pri testovaní, ale opäť vykazoval problémy s textúrovacími súradnicami. Žiaden ďalší odskúšaný model-loader nebol v praxi použiteľný. Bolo by bezúčelné uvádzať v texte analýzy týchto malých chybných model-loaderov, miesto toho je uvedená analýza knižnice Assimp. Nejedná sa o model-loader, ale iba o načítavač zostáv, teda údajov z rôznych 3D formátov. Zabezpečuje podporu pre vytvorenie vlastného model-loaderu.

5.5.1 Assimp – Open Asset Import Library

Jedná sa o open-source projekt, ktorý sa zaoberá načítavaním 3D geometrie z rôznych formátov. Podporuje načítavanie takmer všetkých dostupných formátov uchovávajúcich 3D geometriu. Podporuje formáty ako 3DS vo všetkých verziách, OBJ, MD2 – 5, MilkShape a dokonca aj najnovší formát – Colladu.

Už sám názov napovedá, že sa nejedná o plnohodnotný model-loader, zabezpečujúci načítanie a správu modelu. Je to „iba“ importér zostáv – súborov s 3D geometriou. Načítava rôzne formáty a prevádza ich do jednotnej vnútornej reprezentácie. Prostredníctvom toho môže programátor pristupovať priamo ku geometrii prípadne reprezentáciu ďalej upraviť do podoby optimálnej pre použitie vo vytváranej aplikácii. Samozrejme, že je možné využívať v aplikácii reprezentáciu vytvorenú priamo importérom, ale táto možnosť nieje odporúčaná a v rade prípadov ani optimálna. V rámci Assimp je možná aplikácia tzv. post-process funkcií na načítanú zostavu. Tieto funkcie sú napr. triangulácia a optimalizácia geometrickej siete, optimalizácia hierarchie, dopočítanie tangent a bitangent a mnoho ďalších.

Importér je nezávislý na použitej architektúre. Je ho možno používať ako s OpenGL, tak aj s D3D prípadne vlastným softvérovým rendererom. Keďže sa nejedná o model-loader, neposkytuje žiadne funkcie vykreslenia ani manipulácie s modelom. Poskytuje len údaje zostavy – načítanú a prípadne optimalizovanú geometriu, svetlá a hierarchiu scény. Jeho použitie je možné na všetkých platformách. Vznikol len pred pár rokmi a v súčasnosti sa aktívne pracuje na jeho rozširovaní, čo mu zaisťuje v budúcnosti podporu.

URL: http://assimp.sourceforge.net

5.6 Matematika

Existuje nepreberné množstvo matematických knižníc, kvôli ich jednoduchej realizácii. Len málo z nich je použiteľných v praxi. Ešte menej z nich je použiteľných pre nízko-úrovňové aplikácie OpenGL. Od takýchto knižníc sa vyžaduje podpora hlavne pre vektory, matice a generovanie transformačných matíc. Zároveň je vhodné ak implementujú trigonometriu. Z takýchto knižníc som vybral k analýze práve GMTL a GLM.

5.6.1 GMTL - Graphics Math Template Library

Využiteľná ako podpora matematiky potrebnej ku grafickým výpočtom vyššej úrovne, pri spracovaní raytracingu, BVH, foton-mappingu a iných techník, je knižnica GMTL. Je to generická matematická knižnica, ktorá je nezávislá na type použitého grafického API. Obzvlášť vhodná je pre softvérový rendering. Je multiplatformná, bez závislostí a je aktuálna. Podporuje prácu s hraničnými obálkami objektov a detekciu ich kolízií. Je zameraná hlavne na parametrizovateľnosť a výkon. Nevhodná je pre použitie s nízko-úrovňovými grafickými API akým je aj OpenGL. V týchto nástrojoch je jej použitie ťažkopádne. Zároveň k nej neexistuje kvalitná literatúra.

URL: http://ggt.sourceforge.net

5.6.2 GLM – OpenGL Mathematics

OpenGL Mathematics je matematická knižnica určená výhradne k použitiu v OpenGL. Knižnica je multiplatformná, bez závislostí a aktuálna. Vďaka tomu, že knižnica je hlavičková, je jej inštalácia a použitie jednoduché.

Pri výbere matematických funkcií sa riadi priamo špecifikáciou GLSL. To zaručuje, že obsahuje všetku potrebnú funkcionalitu. Okrem funkcií daných špecifikáciou je knižnica doplnená o pomocné výpočty, týkajúce sa hlavne výpočtov s transformačnými a projekčnými maticami. Podporuje trigonometrické výpočty a všeobecne prácu s vektormi a maticami. Len obmedzene je vhodná na použitie pri výpočtoch vysokoúrovňových grafických metód ako je raytracing, radiozita, BVH a ďalšie. Knižnica je výborne zdokumentovaná.

URL: http://glm.g-truc.net

5.6.3 Alternatívy pre GLM

Ďalšie, avšak menej preskúmané alternatívy sú knižnice CML a Eigen. Sú rozsiahlejšie ako GLM a ich podpora pre OpenGL sa javí taktiež veľmi dobrá. Pre prácu s OpenGL API sú jednoznačne lepšou voľbou ako knižnica GMTL.

URL: http://cmldev.net

URL: http://eigen.tuxfamily.org

5.7 OpenGL API

K použitiu OpenGL príkazov je nutné vkladať do zdrojových hlavičkový súbor *gl.h*. Tento je v mnohých prekladačoch dostupný len do verzie OpenGL 1.1. Z tohto dôvodu je potrebné využiť knižnice so zabudovanou širokou podporou OpenGL príkazov a extenzií ako sú GLEE a GLEW. Navyše pre OpenGL v CORE profile je použitie týchto knižníc nevyhnutné, kvôli nedostupnosti hlavičkového súboru *gl.h*, ani zo samotného portálu opengl. Tieto knižnice predstavujú nástroje sledujúce vývoj špecifikácie OpenGL a poskytujú pre ňu aktualizovanú podporu. Pre programy do verzie OpenGL 1.5 však stačí použitie hlavičkového súboru *gl.h* z portálu opengl a pre podporu extenzií hlavičkový súbor *glext.h*.

5.7.1 GLEE – OpenGL Easy Extension library

Knižnica GLEE zaisťuje podporu príkazov OpenGL a mnohých dostupných extenzií. Podporuje OpenGL príkazy do verzie 3.0 a úctyhodných 398 extenzií. Jedná sa o knižnicu, ktorá je výborne spracovaná, veľmi intuitívna a jednoduchá na použitie. Priamo počas behu programu podporuje dotazovacie príkazy k zisteniu podpory príkazu, alebo extenzie na príslušnej verzii OpenGL. Pre načítanie funkcií používa tzv. lazy-loading metodiku. Tá umožňuje kompilovať príslušné funkcie až v čase, keď sú skutočne použité v programe. Tým je odstránená potreba inicializácie.

Knižnica je multiplatformná, ale je závislá na existencii hlavičkového súboru *gl.h* v prekladači. Tým sa obmedzuje použitie v niektorých minoritných prekladačoch. Knižnica je vynikajúca k integrácii do iných knižníc. V čase tvorby projektu bol jej ďalší vývoj zastavený. Posledná revízia je z roku 2009.

URL: http://elf-stone.com/glee.php

5.7.2 GLEW – OpenGL Extension Wrangler library

Obdobne ako predchádzajúca knižnica GLEE, aj knižnica GLEW zaisťuje podporu OpenGL príkazov a načítavanie OpenGL extenzií. Knižnica je dobre spracovaná. Podporuje najnovšiu verziu OpenGL 4.1, spolu so všetkými dostupnými extenziami. Obsahuje príkazy na dotazovanie podpory a existencie vybraných OpenGL funkcií na aktuálnej OpenGL platforme počas behu aplikácie, podobne ako je tomu v GLEE. Oproti knižnici GLEE je nezávislá na existencii hlavičkového súboru *gl.h*. Základné funkcie OpenGL tak implementuje sama.

Knižnica je menej intuitívna v porovnaní s GLEE, ale aj napriek tomu je jednoduchá na použitie. Bohužiaľ je nutné ju inicializovať. To je značne obmedzujúci faktor pre integráciu do iných knižníc. GLEW je multiplatformná a aktuálna knižnica s neustálou aktualizáciou a vývojom.

URL: http://glew.sourceforge.net

6 Vývojové prostredia

6.1 NetBeans

NetBeans je kvalitne spracované a zdarma dostupné moderné vývojové prostredie, ktoré sa podobá prostrediu Eclipse. Podporuje viacero programovacích jazykov. Najčastejšie sa používa pre vývoj aplikácií v jazyku Java. Okrem neho podporuje aj C++, PHP, JavaFX, Ruby a niektoré ďalšie málo používané programovacie jazyky. V maximálnej verzii, v ktorej obsahuje všetky podporované programovacie jazyky a prekladače, je prostredie ťažkopádne. Je však dostupné aj menších v balíkoch, ktoré sú špeciálne určené pre konkrétny jazyk. V týchto verziách je práca s prostredím nepomerne svižnejšia. Prostredie je multiplatformné. V maximálnej verzii zaberá na disku približne 700MB. Vo verzii určenej pre C++ má veľkosť približne 150MB. To predstavuje výhodu oproti konkurenčnému MSVC. Neustále sa vyvíja a rozširuje. Má vybudovanú širokú komunitu.

V prípravnej fáze projektu nebola zistená žiadna priama podpora OpenGL. Ani pre programovací jazyk Java, ani pre C++. K vytvoreniu aplikácie používajúcej OpenGL je potrebná implementácie externých knižníc príslušného jazyka, napr. JOGL pre Javu alebo GLEW pre C++.

URL: http://netbeans.org

6.2 MSVC – Microsoft Visual Studio

Najpoužívanejšie prostredie na vývoj aplikácií v jazyku C++ je Microsoft Visual Studio. Je to komerčné prostredie, ktorého študentská verzia je dostupná zdarma. Je závislé na platforme Windows. Okrem jazyka C++ podporuje aj ostatné jazyky pod správou M$, a teda C# a VisualBasic. Prostredie je nekvalitne spracované. Ťažko sa ovláda, poskytuje minimálne množstvo vlastností podporujúcich rýchlejší vývoj aplikácií. Projekty vytvorené prostredníctvom tohto prostredia sú spätne nekompatibilné v rámci verzií produktu. Vyššie verzie umožňujú načítavanie projektov vytvorených vo verziách nižších ale iba do verzie MSVC 6. Aj napriek malému množstvu poskytovaných funkcií zaberá vo verzii 2008, pri inštalácii iba jazyka C++, až 2,2GB. Úplná inštalácia zaberie približne 4GB.

Prekladač obsahuje natívne OpenGL vo verzii 1.1. Vzhľadom na rozšírenosť používania tohto prostredia pri vývoji OpenGL aplikácií, existuje výborná podpora v komunite. Množstvo existujúcich projektov, ktoré sa venujú OpenGL, poskytuje príklady práve pre prostredie MSVC.

MSVC stále aktívne vyvíjané. Rozšírenosť jeho používania mu zaručuje istú podporu v budúcnosti. Prostredie je závislé na platforme Windows. Nedá sa však nepodotknúť, že aj napriek svojej desaťročnej zaostalosti oproti vývojovým nástrojom určeným pre iné jazyky, sa v súčastnosti jedná o jedno z najrozšírenejších prostredí pre vývoj C++ aplikácií.

6.3 Code::Blocks

Jedná sa o kvalitne spracované prostredie, ktoré je dostupné zdarma. Je pomerne rozšírené pre vývoj aplikácií v C++. Multiplatformnosť mu zaručuje široké uplatnenie. Jeho výhodou je to, že je malé a určené výhradne pre jazyk C++. Vďaka tomu je flexibilné a ľahko rozšíriteľné. Napriek malej veľkosti (jeho veľkosť je približne 150MB vrátane vstavaného prekladača pre platformu Windows – MinGW), poskytuje množstvo funkcií na podporenie rýchleho vývoja aplikácií. Okrem prekladača MinGW, čo je variant špičkového unixového kompilátoru GCC, podporuje množstvo rôznych druhov prekladačov pre jazyk C++.

V rámci prostredia je vstavaná podpora pre rýchle vytváranie OpenGL projektov, založených na rôznych frameworkoch. Sú dostupné šablóny pre aplikácie ako napr. štandardný nízko-úrovňový OpenGL, GLUT, GLFW, Irrlicht, SDL, SFML, GTK+ a mnohé iné. Prostredie sa aktívne vyvíja a má širokú komunitnú podporu. Vzhľadom na prepracovanosť prostredia a ľahkosť ovládania je možné usudzovať na jeho dlhú existenciu.

URL: http://www.codeblocks.org

7 Reprezentácia povrchovej informácie

Priestorové telesá sa v počítačovej grafike najčastejšie reprezentujú pomocou informácií o povrchu. Povrch objektu je vždy základom reprezentácie. Na povrchových detailoch závisí kvalita vizualizácie, a tým pádom aj celkový dojem. Dnes, v dobe programovateľných grafických kariet, je možné vďaka informáciám o povrchu dosiahnuť zaujímavé efekty.

Povrchové informácie modelu sa najčastejšie uchovávajú vo forme 2D rastrových obrázkov. Okrem základných druhov povrchovej informácie, kde sa radí uchovanie farby povrchu a normály, v nedávnej dobe pribudlo uchovávanie AO informácií, informácií o svetelných podmienkach prostredia pre povrch objektu a o odrazivosti povrchu. Tieto informácie sa spracovávajú hlavne pri aplikácii shaderov.

7.1 Farebné textúry – Diffuse/Color textures

Farebné textúry sa vo svojej podstate nijako neodlišujú od obyčajných obrázkov. Existujú ale v troch základných variantoch: smerové textúry, dlaždicové textúry a textúry objektu.

Dlaždicové textúry sú symetrické, spravidla podľa viacerých ôs, a nadväzujú na seba vo všetkých smeroch. Taktiež ale nemusia byť symetrické. Postačujúcou podmienkou je aby mali symetricky definované okraje. Tento typ textúry sa využíva v príkladoch, v ktorých sa pomocou opakovaného kopírovania malej textúry pokrývať väčšia plocha objektu.

Smerové textúry sú dôležité pre znázornenie mapovania textúry na objekt. Nie sú symetrické podľa žiadnej osi a nenadväzujú na seba. Užívateľ je po vykreslení scény schopný ľahko rozpoznať kam sa ktorá časť textúry namapovala.

Objektové textúry sú špecificky vytvorené textúry pre pokrytie celého komplexného objektu zapracované do jediného obrázku. Model do týchto textúr pristupuje podľa presne definovaných textúrovacích súradníc. Ich výhodou je korektne definované mapovanie na objekt, úspora textúrovacej pamäte, ale najmä jednoduchá manipulácia.

Obrázok 1: Symetrická dlaždicová textúra

Obrázok 2: Nesymetrická dlaždicová textúra

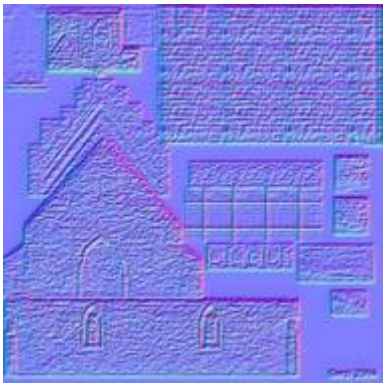
Obrázok 3: Smerová textúra

Obrázok 4: Objektová textúra

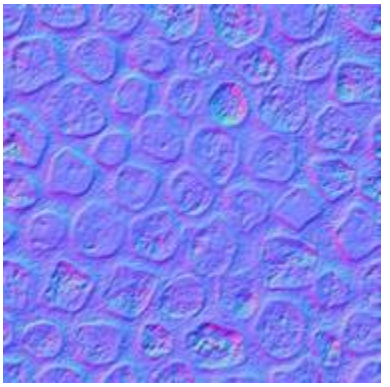
7.2 Normálové textúry – Normal textures

Aj v prípade normálových textúr, podobne ako pri farebných textúrach, existujú dva základné varianty. Prvý typ udáva výchylku normály v tangentovom priestore – perturbation textures. Naopak druhý typ definuje priamo hodnotu normály – normal textures. Medzi týmito textúrami nebýva spravidla rozdiel. Toto názvoslovie je zavedené a slúži len pre programátora na identifikáciu aplikovanej metódy použitia. Normálové textúry sa používajú hlavne ku zvýšeniu efektov na povrchu modelu.

Farba tohto typu textúr je namodralá. Je to z dôvodu, že tieto textúry udávajú výchylku, prípadne priamo normálu, v smere XYZ mapovanú na farebné kanály RGB. Vo farebnom priestore RGB to tak reprezentuje postupne červenú, zelenú a modrú farbu. Normála sa v smere Z, ktorý smeruje von z obrázku, mení len veľmi málo. Hodnota Z je teda blízka 1, čo predstavuje v priestore RGB sýto modrú farbu. Preto celá textúra pôsobí modrým dojmom.



Obrázok 5: Normálová textúra definovaná pre komplexný objekt

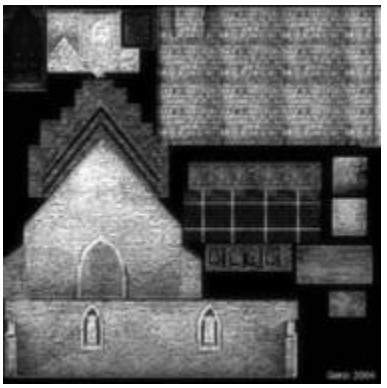


Obrázok 6: Normálová textúra kameniva

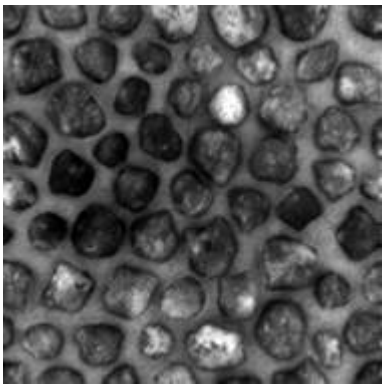
7.3 Výškové textúry – Height textures

Výškové textúry v niektorom svojom farebnom kanáli obsahujú informáciu o relatívnej výške povrchu. Tieto obrázky sú obyčajne v odtieňoch šedej farby, teda v každom z RGBA kanálov je prenášaná tá istá hodnota určujúca relatívnu výšku povrchu objektu. Ak je textúra farebná, je možné do každého zo štyroch farebných kanálov uložiť rôzne výšky. Pokiaľ sú tieto textúry v bežne dostupnom LDR formáte, majú nízku presnosť kvôli nemožnosti uchovať viac ako 255 rozdielnych hodnôt (8 bitov) v rámci celého obrázku. Dnes však existuje formát označovaný ako HDR, ktorý umožňuje uchovať rozsah hodnôt odpovedajúci až 128 bitom. Táto presnosť je dostačujúca aj na zobrazovanie rozmerných objektov s malými detailami. Pri obrázkoch LDR to nieje možné.

Výškové textúry sú obvykle dostupné s ich normálovým a farebným variantom. Dajú sa však použiť aj samostatne. Bežne sa výšková textúra kóduje do jedného obrázku spolu s normálovou textúrou. RGB kanály slúžia k prenosu XYZ hodnôt normály a kanál A obsahuje výškovú mapu. To sa často využíva v shaderoch, nakoľko tie umožňujú individuálny prístup k týmto informáciám. Zníži sa tým celkové zaťaženie systému vzniknuté presunom dát do GPU. Zároveň sa ušetrí ako pamäť CPU tak aj GPU.



Obrázok 7: Výšková textúra definovaná pre komplexný objekt



Obrázok 8: Výšková textúra kameniva

7.4 Mapy prostredia – Environmetal maps

Svetelné podmienky prostredia je možné určiť prostredníctvom environmentálnych máp. Použite týchto obrázkov môže nahradiť množstvo, rádovo až tisícky, svetelných zdrojov definovaných v scéne. Mapy prostredia sú v podstate obyčajné farebné obrázky, ktoré určujú svetelnú energiu dopadajúcu na objekt z určitého smeru. Špecifickým spôsobom však dochádza ku generovaniu súradníc do týchto textúr, potrebných k získaniu svetelnej energie dopadajúcej na povrch objektu z určitého smeru. Obrázky majú špecifický tvar, ktorý závisí od typu. Tieto mapy môžu byť sférické, kubické, prípadne lat-long mapy (lattitude-longitude).

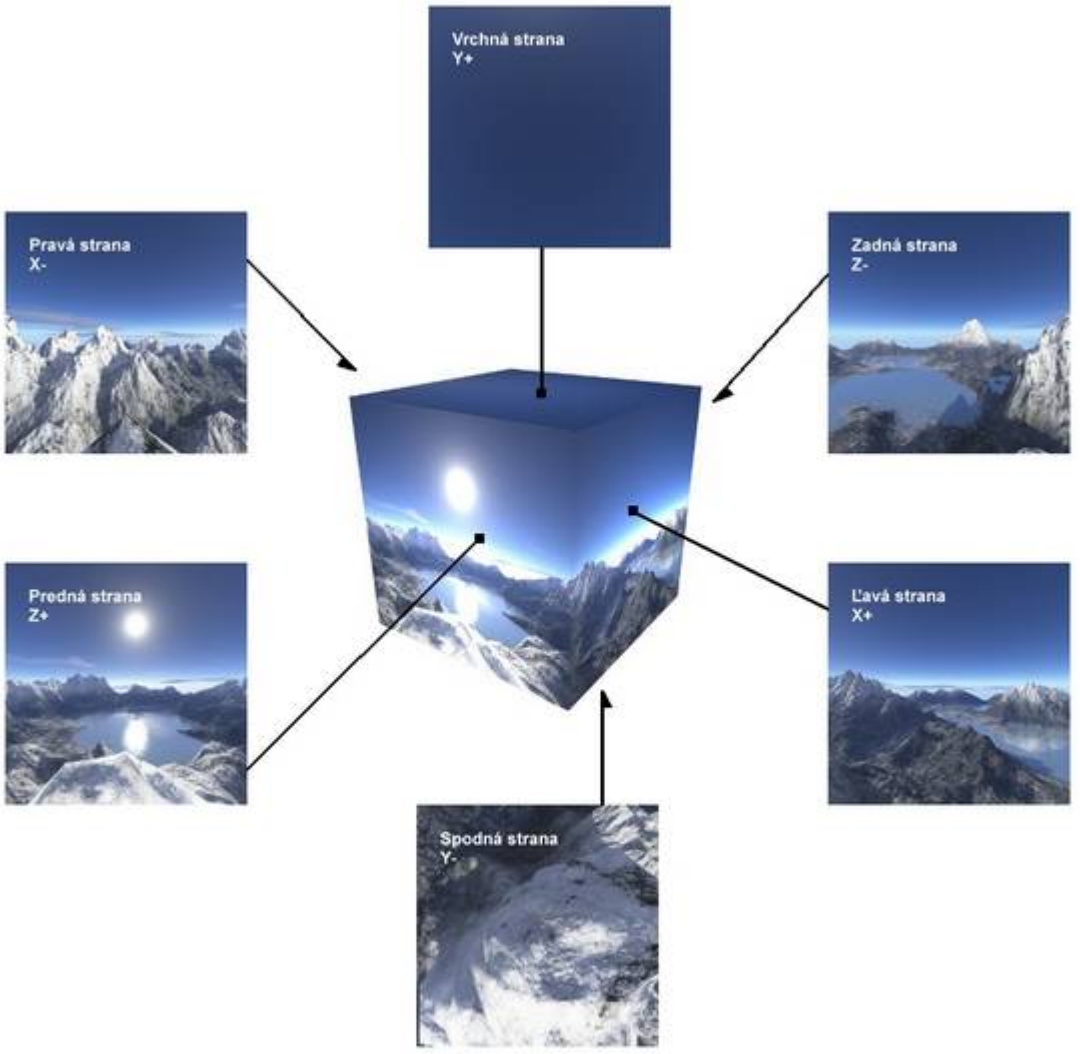
7.4.1 Sférické mapy – Spherical maps

Ich špecifický kruhový tvar sa získava buď synteticky, alebo odfotografovaním reálneho prostredia. Pri fotografovaní sa postupuje umiestnením plne odrazovej, napr. leštenej chrómovej, gule do prostredia. Tá sa potom vyfotografuje z dostatočnej vzdialenosti – v ideálnom prípade z nekonečnej vzdialenosti s objektívom s nekonečným fókusom. Nevýhoda týchto textúr je ich exponenciálne klesajúca kvalita v smere od stredu ku krajom kruhu.

Obrázok 9: Sférická mapa

7.4.2 Kubické mapy – Cubemaps

Kubická environmentálna mapa pozostáva zo sady šiestich 2D obrázkov zobrazujúcich prostredie zo všetkých smerov: +X,-X,+Y, -Y, +Z, -Z. Pri ich spojení v priestore dostaneme kocku, ktorej strany sú jednotlivé časti kubickej mapy. Tiež ich je možné vyrobiť synteticky, alebo získať odfotografovaním. Výhoda kubických máp spočíva vo vysokej kvalite a jednoduchosti zhotovenia. Nevýhodou je práca so šiestimi oddelenými textúrami, namiesto jednej, čo obecne znižuje manipulovateľnosť. Napriek existujúcim vylepšeniam v podobe uloženia všetkých šiestich textúr do jedného obrázku (vertikálny/horizontálny kríž, uloženie za sebou s definovaným poradím), podporuje OpenGL kubické mapy iba spôsobom načítania šiestich samostatných textúr.



Obrázok 10: Kubická mapa
Popisky sú zorientované tak, že užívateľ stojí v strede kocky a jeho pohľad smeruje v kladom smere osy Z.

7.4.3 Lat-long mapy – Lat-long maps

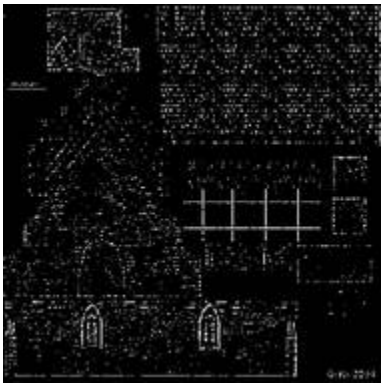
Lattitude-longitude mapy sú hybridom medzi sférickými a kubickými mapami. Dosahujú asi 90%-tnú kvalitu kubických máp. Sú však obsiahnuté v jednej textúre. Znižuje sa zložitosť pri importe textúry do OpenGL a zvyšuje sa pracovná efektivita. Mapy je možné získať synteticky pomocou konverzie z kubických, alebo sférických máp za pomoci špecializovaných programov ako napr. HDR Shop. Mapu je možné použiť aj k zjednodušeniu efektov vyžadujúcich kvalitu kubických máp.

Obrázok 11: Lat-long mapa

7.5 Mapy odleskov – Gloss/Specular maps

Mapy odleskov sú najčastejšie čiernobiele obrázky, kde obyčajne biele plochy určujú miesta kde dochádza k počítaniu odrazovej zložky svetla na povrchu objektu. Záleží však na programátorovi, ktorú z farieb biela-čierna zvolí pre počítanie odrazovej zložky. V prípade, že mapy sú v odtieňoch šedej farby, odtieň udáva faktor odrazu materiálu. Využíva sa to na ďalšie zvýšenie realistickosti odleskov svetla na povrchu objektu. Objektívne tým dochádza k pocitu nehomogénnosti materiálu. Jej použitím sa dosahuje kvalitný a pôsobivý efekt.

Obrázok 12: Gloss mapa v odtieňoch sivej



Obrázok 13: Čierno-biela gloss mapa

8 Reprezentácia geometrickej informácie

Geometrické objekty sa v počítačovej grafike reprezentujú pomocou súradníc vrcholov. Tieto k plnohodnotnej reprezentácii nestačia, pretože pre typické grafické výpočty sú nevyhnutné normály, textúrovacie súradnice ako aj ďalšie atribúty. V ďalšom texte budú popísané najpoužívanejšie atribúty vrcholov a formáty pre ich uchovávanie.

8.1 Atribúty vrcholov

V tejto časti sa zameriam na základné atribúty akými sú súradnice vrcholov, normály a textúrovacie súradnice. Zároveň popíšem v praxi najčastejšie používané doplňujúce atribúty ako tangenty, binormály a faktory zatieneia. Tieto sa používajú hlavne v pokročilých technikách sú bump mapping, SSAO či SSDO.

8.1.1 Vrcholy a normály

V rámci geometrie za uchovávajú informácie o súradniciach vrcholov. Tieto sú v OpenGL reprezentované v afinnej reprezentácii pomocou 4-zložkových vektorov. Vrcholy sa potom spájajú do trojuholníkových alebo kvadriliterálnych primitív. To sú základné geometrické objekty OpenGL.

Ku každému vrcholu musí existovať normála. Táto vlastnosť je dôležitá hlavne pri výpočte osvetlenia. Nedá sa bez nej v súčasnosti zaobísť ani pri vytváraní takmer všetkých GLSL shaderov, najmä efektov, ktoré vizuálne zlepšujú povrch objektu.

Normály musia byť normalizované. Fixná pipeline OpenGL, ako aj bežné implementácie v shaderoch vykonávajú napr. výpočet miery osvetlenia pomocou skalárneho súčinu a nie trigonometricky, teda na základe výpočtu s uhlami. Ten vyžaduje normalizované vektory, pretože veľkosť vektora sa premieta do výsledku skalárneho súčinu. Ten má rádovo menšie nároky na výkon ako trigonometrický. Všetky normály musia byť zhodne orientované, aby bolo možné identifikovať vonkajšiu resp. vnútornú stranu objektu. Typicky sa orientácia normál uvádza smerom von z povrchu. Ďalším významom normál je možnosť za ich pomoci dopočítať tangenty a bitangenty. Tieto atribúty modelu sú veľmi dôležité pre množstvo efektov, ktoré pracujú na per-pixel báze, a takmer všetkých efektov zvyšujúcich detaily povrchu vo fragment shaderoch.

8.1.2 Textúrovacie súradnice

Textúrovacie súradnice musia byť určené pre každý bod. Sú dôležité pre správne „nasadenie“ textúry na objekt, a tým vytvorenie správneho vizuálneho vnemu zobrazovaného objektu. Dôležité je ich rovnomerné rozloženie v textúre. Minimalizuje sa tým interpolácia a zvyšuje sa tak výsledná kvalita. Rovnomerné rozloženie textúrovacích súradníc ovplyvňuje aj výsledky niektorých GLSL efektov.

8.1.3 Tangenty a bitangenty

V každom vrchole by mali byť určené tangenty a bitangenty. Jedná sa o vzájomne kolmé dotyčnice k povrchu v danom bode (pixeli), kolmé na normálu v tomto bode. Tieto súradnice môžu byť súčasťou atribútov vrcholov modelu, alebo je možné ich dopočítať. Pri výpočte je pevne určená ich orientácia. To nemusí byť pre špecifický model výhodné. Preto je lepšie, aby tangenty a bitangenty obsahoval samotný model explicitne stanovené jeho autorom.

Tangenty a bitangenty musia byť jednotkové. Spolu s jednotkovou normálou tak vytvárajú ortonormálny súradný systém s počiatkom v danom bode na povrchu geometrie. Tieto atribúty sa využívajú hlavne v shaderoch pri práci s efektami založenými na báze per-pixel. Najčastejšie sú to efekty zvyšujúce detailnosť povrchu. V prípade, že model neobsahuje tangenty, a je potrebný ich dodatočný výpočet, sú pre správne výsledky nutné textúrovacie súradnice. Pri tomto výpočte sa musí dodržať zhodná orientácia vzhľadom k povrchu, obdobne ako je tomu u normál.

8.1.4 Faktory zatienenia

Faktor zatienenia (occlusion factor) je desatinné číslo z intervalu [0,1]. Udáva percentuálnu viditeľnosť prostredia v danom vrchole z hemisféry centrovanej podľa normály. Ich využitie je v implementáciách osvetľovacích modelov pomocou shaderov. Najmä pri vytváraní efektov typu Ambient Occlusion (AO). Ich využitím sa zvyšuje realistickosť zobrazenia modelov. Túto informáciu je ďalej možné využiť aj v iných efektoch.

8.2 Formáty pre uchovávanie 3D geometrie

Z veľkého množstva formátov slúžiacich k uchovávaniu 3D geometrie sú najvýznamnejšie 3DS, OBJ a MD2. 3DS je známy kvôli rozšíreniu aplikácie 3D studio Max, OBJ je významný svojou stabilnou špecifikáciou a jednoduchosťou spracovania a MD2 sa stal významným vďaka veľkému rozšíreniu a sprístupneniu herného enginu Quake2. Formát Collada, ktorý je popísaný na záver, nieje v súčasnosti vôbec rozšírený, avšak predstavuje otvorený formát vypracovaný na základe skúseností získaných z predchádzajúcich omylov. Preto je možné predpokladať jeho dlhú budúcnosť a široké uplatnenie.

8.2.1 3DS

3D modely dostupné na internete sa najčastejšie vyskytujú vo formáte 3DS. Jedná sa o zastaralý formát, ktorý bol vytvorený primárne pre použitie v aplikácii 3D Studio Max, udržiavaný korporáciou Autodesk. Stratégia Autodesku vychádza v súčasnosti z toho, že každý rok je potrebné vydanie novej verzie programov. To má za následok, že často dochádza k zmene špecifikácie formátu. Hoci je 3DS schopný reprezentovať okrem statických modelov i modely animované, tento formát nieje vhodný, ak je zamýšľaná jeho dlhodobá použiteľnosť. Navyše špecifikácia tohto formátu pochádza z čias systému DOS a niektoré atribúty sú len 16-bitové. Už pri trochu zložitejších modeloch sa toto obmedzenie začne neblaho prejavovať.

Ďalšou prekážkou jeho použitia je neexistencia moderného a spoľahlivého loaderu. Na internete je dostupné množstvo model-loaderov pre 3DS formát vo verziách pre 3D Studio Max 2 až 4. Niektoré sú dokonca určené až pre verziu 2009. Typicky ale obsahujú množstvo chýb, týkajúcich sa hlavne nesprávneho načítavania materiálov a textúrovacích súradníc. Najčastejšie sú vytvorené v jazyku ANSI C, teda sú neobjektové a nie sú nijako optimalizované. Na vykresľovanie používajú skoro všetky existujúce realizácie Immediate mode. Ten je podporovaný len do verzie OpenGL 2. Z vyššej verzie OpenGL 3 je odstránený. Dostupné model-loadery obvykle zapúzdrujú svoju funkčnosť a neposkytujú priamy prístup k vrcholom a ich atribútom, čo je často vyžadované. Ani vytvorenie vlastného loaderu by nezaručovalo jeho použiteľnosť počas najbližších dvoch rokov, kvôli častej zmene špecifikácie formátu.

Medzi najčastejšie nedostatky voľne dostupných modelov v tomto formáte patrí absencia textúr a normály niektorých častí modelu bývajú prevrátené. Ďalej sú normály skoro vždy nejednotkové a geometrická sieť nieje trojuholníková. V prípade, že k modelu existujú textúry sú špatne nastavené textúrovacie súradnice modelu, alebo sa v modeli vôbec žiadne nevyskytujú. Bez správnych súradníc bohužiaľ nieje možné model korektne otextúrovať a zároveň je znemožnené správen vypočítanie tangent a bitangent, ktoré sú potrebné pre niektoré často používané efekty ako napr. Bump-mapping.

8.2.2 OBJ

Formát OBJ je najpoužívanejším pre modely v nízko-úrovňových OpenGL aplikáciách. Dostupnosť a početnosť modelov tohto formátu na internete je nižšia ako v prípade formátu 3DS. Existujú však kvalitné konvertery, ktoré dokážu formát 3DS previesť do formátu OBJ. Neopravia však špatnú reprezentáciu samotného modelu.

Presne špecifikovaný, otvorený a stabilný základ formátu ho odlišuje od formátu 3DS. Je však vhodný, iba pre statické modely. Má textovú podobu súborov. Formát je jednoduchý na spracovanie. Uchováva si najdôležitejšie atribúty geometrických vrcholov, potrebné pre interaktívnu a RT počítačovú grafiku. Uchováva hlavne najdôležitejšie časti 3D geometrie, ktorými sú súradnice vrcholov, normály, textúrovacie súradnice a materiály povrchov. Ostatné údaje týkajúce sa modelu ako napr. tangenty a bitangenty je možné k modelu dopočítať. Uchovávané údaje týkajúce sa materiálov povrchov sú vhodné len pre aplikáciu primitívneho Phongovho osvetľovacieho modelu.

Jednoduchý model-loader je možné ľahko vytvoriť parsovaním textového súboru. Dostupnosť kvalitných loaderov je bohužiaľ slabá, avšak sú aspoň stabilné pri načítavaní ako starších, tak i nových modelov. Zabezpečuje to stabilná špecifikácia. Aj napriek jednoduchosti formátu sa úplne spoľahlivo pracujúce loadery ťažko zháňajú. Obvykle majú problémy so spracovaním textúr a materiálov, čo nemusí byť striktne spôsobené samotným loaderom, ale môže to spôsobovať chybný model vygenerovaný nekvalitným exportérom.

Súčasné model-loadery sú zastaralé a nevhodné pri použití OpenGL verzie 3 a 4 v profile CORE. Je to z dôvodu neexistujúcej podpory VBO v nich. Takmer všetky vykresľujú geometriu pomocou Immediate mode, v lepšom prípade pomocou Display listov. Tieto spôsoby vykresľovania sú v nových verziách OpenGL v profile CORE zakázané. Implementácie obvykle neposkytujú prístup k načítaným vrcholom a ich atribútom. Poskytujú len načítanie a vykreslenie modelu a z pohľadu užívateľa sa správajú ako čierne skrinky. Pri tvorbe môjho projektu, je ale takáto vlastnosť nežiadúca.

Aj napriek tomu, že je formát zastaralý a už vyše 10 rokov sa nevyvíja, bol navrhnutý nadčasovo, a je stále aktívne používaný. V prípade spracovávania statických modelov je formát OBJ jednoznačne najlepšia voľba zo súčasných existujúcich formátov.

8.2.3 MD2

Formát MD2, ktorý vznikol pri tvorbe hry Quake 2, je uzavretý binárny fromát s presne danou a stabilnou špecifikáciou. Jeho stabilita vyvažuje i zložitosť spracovania v porovnaní s formátom OBJ. Podporuje uchovanie a rozšírené vlastnosti i animovaných modelov.

Model-loadery MD2 sú ťažko dostupné, aj keď návody v C++ na načítanie formátu sa dajú obstarať poľahky. Na internete sa modely vo formáte MD2 takmer nevyskytujú. Je to

spôsobené z veľkej miery nedostupnosťou akéhokoľvek modelára, či editora alebo konvertera do tohto formátu. MD2 je zastaralý formát, ktorý sa už vyše 10 rokov nevyvíja. Jeho dnešným nástupcom je formát MD5.

8.2.4 Collada

Jeden z najnovších 3D formátov predstavuje Collada. Je to moderný a perspektívny formát, navrhnutý špeciálne pre uchovanie 3D geometrie s množstvom priradených vlastností. Collada je postavená na XML, ktorý zabezpečuje ľahkú spracovateľnosť akýmkoľvek XML parserom. Knižnice spracovávajúce XML sú dnes dostupné prakticky pre každý v súčasnosti používaný programovací jazyk. Vzhľadom k tomu, že Collada dokáže obsiahnuť akúkoľvek požiadavku na uchovanie akýchkoľvek atribútov geometrie, je XML špecifikácia veľmi rozsiahla a nie vždy úplne jednoduchá a logická.

Formát Collada umožňuje uchovávanie statických i animovaných modelov. Okrem samostatných objektov je schopná uchovávať aj celú scénu a jej hierarchiu. Podporuje uchovávanie textúr i materiálov a to pre rôzne druhy osvetľovacích modelov – phong, ward, blinn, lafortune, atď. Dokonca umožňuje uloženie shaderových efektov priamo v súbore s geometriou modelu.

Z dôvodu, že sa jedná o nový formát, existuje na internete iba málo modelov k voľnému stiahnutiu. S model-loadermi je to obdobné. Nie sú dostupné žiadne ani pre najrozšírenejšie jazyky ako C++ a Java. Avšak pre jazyk WebGL sa ich dá nájsť hneď niekoľko. Myšlienka vytvorenia vlastného loaderu stroskotala na absencii kvalitnej literatúry potrebnej k priblíženiu základných čŕt formátu, potrebných na vytvorenie, čo i len, jednoduchého statického loaderu. Algoritmy je však možné získať z rozsiahlych herných strojov resp. scene-graphov, ktorých zdrojové kódy sú k dispozícii, a ktoré po väčšine načítavaním geometrie v tomto formáte disponujú.

Collada je nový a otvorený formát, ktorý sa v bude v praxi určite dlhú dobu používať. Dokáže obsiahnuť požiadavky na akékoľvek atribúty geometrie a je vďaka XML jednoducho rozšíriteľný. Formát je umožnené rýchlo adaptovať na nové špecifické požiadavky v prípade ich vzniku v budúcnosti.

9 Návrh

Cieľom tejto práce je optimálny návrh a vytvorenie sady výukových programov. Z predchádzajúcej analýzy je zrejmé, že aktuálne dostupné riešenia sú prevážne zamerané na iný cieľ. Zároveň ani tento sa vo väčšine prípadov nedarí úspešne dosiahnuť.

Grafická knižnica OpenGL je navrhnutá pre podporu vykresľovania 3D reprezentácií. Zároveň je táto knižnica úplne nezávislá na architektúre a je projektovaná ako prenositeľná. To je kľúčové si uvedomiť. V knižnici tak nenájdeme okrem vykresľovacích príkazov a príkazov nastavenia stavov žiadne iné príkazy. Nieje podporované žiadne načítanie textúr ani geometrie. Neexistuje žiadna podpora pre matematiku ani prácu s vektormi. Dokonca nieje zaistené ani vytvorenie kontextu či okna. Všetky tieto vlastnosti sú tak plne v réžii programátora. Z tohto dôvodu nieje možné vytvoriť program s jednoduchým a prehľadným kódom, ktorý by bol navyše nezávislý na externých knižniciach.

Optimálny výukový program by mal obsahovať iba kód špecifický pre daný problém a zároveň by nemal byť závislý na externých knižniciach. Mal by poskytovať všetky potrebné funkcie, nesúvisiace priamo s grafickým API, volaním jedného príkazu. To však v skutočnosti nieje možné dosiahnuť. Týmto môže čitateľ dôjsť k záveru, že vhodným riešením pre vytváranie grafických programov je použitie niektorého, už existujúceho riešenia v podobe herného stroja, alebo manažéra scén tzv. scenegraph. Toto je veľmi správna úvaha, ak cieľom nieje samotné grafické API, ale len jeho použitie v praxi. Užívateľovi tak odpadá množstvo rutinnej (nie však triviálnej) práce so správou svetiel, načítaním geometrie, obrázkov a textúr, správou rozloženia scény a vzájomnou interakciou objektov v nej. Čoskoro však zistí, že aj samotné vykresľovanie geometrie a prípadných efektov má plne v rukách vybraný systém. Tieto komplexné frameworky síce umožňujú čiastočné ovplyvňovanie zobrazovacieho reťazca, avšak nie do plnej miery. Keďže mojim cieľom je práve demonštrácia týchto možností, nieje možné využitie žiadneho podobného systému. Z uvedených dôvodov je nutné využiť buďto existujúce externé knižnice, alebo si celú chýbajúcu funkčnosť naprogramovať samostatne.

Prvá možnosť môže zvýšiť prehľadnosť a jednoduchosť kódu. Zároveň použitie existujúceho kódu šetrí čas a zvyšuje stabilitu, pretože predpokladám aspoň jeho základné otestovanie. Naopak táto možnosť prináša závislosti, ktoré môžu časom technologicky zostarnúť a komplikuje sa preklad kódu v rôznych prekladačoch a na rôznych platformách. Druhá varianta naopak zvyšuje prenositeľnosť a udržiavateľnosť kódu naprieč platformami a v čase. Na druhú stranu neúnosne zvyšuje nároky na znalosti a čas práce. Zároveň môže znižovať stabilitu pri menej dôkladnom testovaní všetkých súčastí a nezaručí, že tento kód nezastará.

V nasledujúcom texte bude predložený návrh na vytvorenie potrebných súčastí použiteľný v rámci všetkých výukových programov, ako aj návrh programov samotných. Na základe vykonanej analýzy a skúseností bude stanovený postup pre vytvorenie základovej kostry programov. Ďalej bude vytýčené použitie externých knižníc popísaných analýzou spolu s návrhom chýbajúcich súčastí, ktoré bude nutné naprogramovať. Pre tento účel bude nevyhnutné určiť model maximalistického programu využívajúci väčšinu aspektov grafického API.

Po vytýčení vlastností modelu výukového programu musíme určiť výukové programy samotné. Tie musia byť navrhnuté s ohľadom na náväznosť a príbuznosť jednotlivých tém. Zároveň je dôležité vziať v úvahu súčasný stav OpenGL a to hlavne rozdelenie na fixnú a programovateľnú pipeline.

Typický OpenGL program je možné rozdeliť na dve základné časti. Prvá, obecná časť, zabezpečuje spustenie aplikácie a jej inicializáciu, vytvorenie okna s kontextom, spustenie hlavného cyklu programu a odchytávanie užívateľských vstupov a udalostí. Druhá, výkonná časť, zaisťuje vykonávanie príkazov spojených s vykresľovaním grafiky cez rozhranie OpenGL. Zatiaľ čo výkonná časť sa bude zásadne meniť naprieč jednotlivými programami, obecná časť bude prakticky stále rovnaká.

9.1 Požiadavky na program

Najzákladnejšími požiadavkami na vytváraný projekt bolo použitie moderných a ekonomicky dostupných nástrojov na jeho realizáciu. Počas analýzy, keď boli existujúce nástroje skúmané, sa vykryštalizovali požiadavky na architektúru programov medzi ktoré patrí:

- použitie objektovo-orientovaného prístupu,
- zachovanie jednoduchosti a prehľadnosti kódu,
- zapuzdrenie kódu, ktorý priamo nesúvisí s OpenGL, alebo sa opakuje vo viacerých programoch do samostatných tried,
- použitie moderného ale použiteľného OpenGL.

Zapuzdrením kódu do tried sa zabráni duplicite kódu, zníži sa chybovosť a zvýši sa efektivita pri vytváraní programov. Pre užívateľa to bude znamenať jednak sprehľadnenie kódu a jednak sa nebude zaťažovať skúmaním kódu, ktorý s danou problematikou nesúvisí. Na splnenie účelu projektu je ďalej potrebné k funkciám OpenGL pristupovať priamo. Len tak je možné ilustrovať prácu s týmto API.

K vytvoreniu akéhokoľvek programu, ktorý predvedie prácu s určitou funkciou OpenGL, bude potrebné najprv zabezpečiť funkčnosť nasledujúcich požiadaviek:

- správa okien a kontextu OpenGL – vytvorenie vykresľovacieho okna, ideálne vo variante Multi-window,
- správa užívateľských vstupov – ovládanie pohybu, natáčanie scény, RT zmena parametrov aplikácie
- umožnenie načítania obrázkov resp. textúr s prístupom k pixelom,
- umožnenie načítania geometrie s prístupom k vrcholom a ich atribútom.

V zložitejších programoch, v ktorých budú aplikované shadere, bude potrebné zabezpečiť ich správu. Všetky tieto úseky kódu bude vhodné zorganizovať po skupinách do samostatných súborov. Vzhľadom k tomu, že tieto funkcie sa budú používať opakovane v každom programe, zabezpečí sa tým prehľadnosť, ale aj jednoduchosť ich používania.

9.2 Štruktúra kódu

Základom k dosiahnutiu jednoduchosti a prehľadnosti zdrojového kódu, ktorá je požadovaná pri tvorbe akýchkoľvek príkladov pre začiatočníkov, je jeho dobrá organizácia a zapúzdrenie opätovne používaných a logicky zviazaných funkcií.

Samotná organizácia môže prebiehať v niekoľkých úrovniach. Najnižšia je organizácia na úrovni funkcií. Ďalej je možno organizovať kód na úrovni tried, potom súborov a následne celých modulov. Na začiatku je potrebné rozdelenie kódu do dobre definovaných úsekov – funkcií. Každý nízko-úrovňový OpenGL program musí obsahovať určité konkrétne bloky funkcií. Je potrebné rozvrhnúť celkový kód do týchto skupín. Základnými blokmi funkcií, ktoré bude projekt obsahovať budú:

- základná inicializácia premenných a spustenie aplikácie,
- vytvorenie okna,
- správa vstupov – spracovanie udalostí okna, spracovanie reálnych vstupov z klávesnice, myši, joysticku, prípadne z ďalších periférií,
- inicializácia OpenGL, jedná sa o hlavnú inicializačnú funkciu, ktorá nastaví všeobecný stav OpenGL, a niekoľko menších inicializačných funkcií, ktoré budú zamerané na nastavenie špecifických vlastností ako napr. nastavenie svetla, hmly a pod.
- rendering, obdobne ako v prípade inicializácie bude jedna hlavná funkcia volať niekoľko podriadených špecifických funkcií.

Pravdepodobne sa ukáže ako žiadúce rozdelenie takto definovaných funkcií do tried prípadne do blokov. Toto rozdelenie závisí na konkrétnych možnostiach používaných systémov. Preto bližšie rozdelenie prebehne po zvolení konkrétnych technológií, teda v rámci implementácie. Je však isté, že v prípade použitia objektovo-orientovaného prístupu bude efektívne zachovanie všetkých týchto funkcií v rámci jednej triedy – triedy aplikácie. Dôvod je prostý. Vždy sa bude jednať o programy malého rozsahu, s presne definovaným cieľom. Následne bude vhodné tento blok funkcií rozdeliť do viacerých podsúborov podľa špecifika funkcií. Tým sa dosiahne ďalšieho zvýšenia prehľadnosti projektu. Človek totiž často ľahšie organizuje viacero štruktúr s menším počtom položiek.

9.2.1 Štruktúra súborov

Štruktúru súborov je potrebné vytvárať s ohľadom na umožnenie jednoduchej a rýchlej zmeny opakujúcich sa úsekov zdrojového kódu, pri vyvíjaní projektu. Hlavne pri zistení nedostatku určitého opakujúceho sa úseku. V dobe, keď bude už vytvorené podstatné množstvo programov s použitím toho úseku kódu, môže nastať zmätok pri jeho vylepšovaní. Pri nesprávnej štruktúre súborov môže zase ľahko dochádzať k vynechaniu opravy tohto úseku zdrojového kódu na niektorom mieste, kde sa používa. Preto je žiadúce rozdelenie zdrojového kódu programu do permanentných a v čase variabilných častí.

Jeden samostatný súbor bude tvoriť inicializácia aplikácie, vytvorenie okna a správa vstupov. Tento súbor by sa nemal prakticky vôbec meniť. Zmeny v tejto časti budú mať obvykle iba charakter zmeny reakcie na užívateľské vstupy iniciované napr. stlačením klávesy. Obvykle sa to bude diať jednoduchými zmenami hodnôt premenných nastavujúcich nejaký stav aplikácie ako napr. hustota hmly, faktor útlmu osvetlenia, faktor odrazu a iné.

Ďalší súbor budú vytvárať metódy inicializácií. Bude obsahovať všetky procedúry zabezpečujúce jednak inicializáciu vykresľovania, ale aj načítanie a inicializáciu textúr a rôznych bufferov, načítanie shaderov i načítanie 3D modelov. Tieto funkcie budú závisieť na konkrétnom programe, budú sa teda meniť vždy podľa požiadaviek.

Rendering bude tiež vhodné separovať do zvláštneho súboru. Budú sem spadať všetky procedúry kresliace na obrazovku. Jedná sa o reprezentáciu výsledného vykreslenia, preto je zrejmé, že sa táto sada funkcií bude musieť radikálne meniť s každým programom.

V rámci inicializácie a renderingu sa vzhľadom k množstvu programov budú často opakovať určité časti zdrojového kódu nesúvisiace priamo s problematikou. Jedná sa napríklad o kód pre načítavanie textúr, geometrie modelov, alebo na určovanie pohľadu do scény. Tento kód nebude vhodné jednoducho separovať a distribuovať s každým jednotlivým programom. Vhodnejšie bude vytvorenie knižnice, ktorá poskytne zapúzdrenie tohto kódu. Dosiahne sa tým zvýšenie prehľadnosti, zjednodušenie a hlavne sa zvýši stabilita a prispôsobenie požiadavkám projektu. Opakujúca sa časť kódu sa totiž bude fyzicky nachádzať len na jednom mieste.

9.3 Podporná knižnica

Vytváraný projekt bude pozostávať zo sady programov. V rámci jednotlivých programov bude ukázané riešenie vždy jedného konkrétneho problému počítačovej grafiky pomocou OpenGL. K fungovaniu týchto programov bude potrebné aj vytvorenie funkcií, ktoré so žiadnou s riešených problematík nebudú priamo súvisieť. Práve zdrojový kód definujúci túto funkčnosť bude obsiahnutý v podpornej knižnici. Sprehľadní sa tým celá štruktúra. Zároveň sa zabráni zbytočnému kopírovaniu rovnakého kódu naprieč programami.

Knižnica musí poskytovať pre každú svoju súčasť jednoduché rozhranie, ktoré zachováva prehľadný a taktiež jednoduchý kód. Vhodnou realizáciou sa javí jej naprogramovanie objektovo-orientovaným prístupom, pri užití menných priestorov. To však bude záležať na zvolenej implementačnej technológii.

Prehľadnosť rozdelenia knižnice musí byť prvoradá. Rozdelenie sa bude realizovať do samostatných nezávislých modulov. Bude vhodné, keď sa každý modul umiestni do samostatného súboru. Docieli sa tým okrem prehľadnosti aj lepšia znovupoužiteľnosť obsiahnutého zdrojového kódu.

Z analýzy vyplynulo, že žiadna zo skúmaných knižníc neposkytuje bezproblémovo funkčné a spoľahlivé načítavanie modelov, kameru a správu shaderov. Tieto vlastnosti bude nutné implementovať vlastné. Vzhľadom na povahu OpenGL, ktoré neupozorňuje na chyby v zdrojovom kóde, bude vhodné implementovať aj vlastnú podporu pre zobrazovanie chýb vzniknutých priamo v OpenGL. Návrh jednotlivých celkov podpornej knižnice bude popísaný v nasledujúcich podkapitolách.

9.3.1 Kamera

Vytvorenie kamery – definovanie dynamického, alebo statického pohľadu na scénu je nevyhnutná záležitosť. Je potrebná v každom grafickom programe. Samotná knižnica OpenGL poskytuje v rámci ním distribuovanej knižnice utilít GLU, riešenie tohto problému v podobe funkcie *gluLookAt()*. Parametre tejto funkcie sú bohužiaľ nevhodne navrhnuté pre vytváranie dynamických pohľadov.

Kamera vytvorená v rámci tohto projektu musí splňovať požiadavky na vytváranie interaktívnych a dynamických pohľadov ovládaných užívateľom. K ovládaniu pohybu pomocou myši bude najvhodnejšie poskytnutie tzv. trackballu. Parametre nastavenia smeru pohľadu je vhodné zvoliť na určenie azimutu a zenitu. K stanoveniu pozície budú jednoznačne použité klasické 3D kartézske súradnice, ktorých orientácia bude súhlasiť s reprezentáciou súradníc v OpenGL.

Je možné vytvoriť dva základné druhy pohľadov. Prvým typom je vytvorenie pozorovacej kamery. Druhým typom je FPS kamera, teda kamera z pohľadu prvej osoby. Pozorovacou kamerou je umožnené len oddiaľovanie resp. približovanie a otáčanie okolo zvoleného počiatku. Tento počiatok je pevne zviazaný so smerom vektoru pohľadu. Táto kamera je vhodná pre bližšie analyzovanie detailov typicky jedného modelu a použitie s trackball. Na druhú stranu predstavuje FPS kamera akéhosi avatara v scéne. Naprieč celou scénou sa dá ľubovoľne pohybovať a meniť vektor pohľadu. Tým je možné voľne a podrobne analyzovať všetky objekty v scéne. Pre analýzu detailov objektu je však toto ovládanie ťažkopádne.

K plnohodnotnému dosiahnutiu cieľa môjho projektu, bude postačovať pozorovacia kamera. V rámci programov projektu sa totiž bude demonštrovať iba presne definovaný efekt na obvykle jednom objekte, do ktorého bude vždy smerovať vektor pohľadu.

9.3.2 Načítanie 3D geometrie

V rámci analýzy bol zistený nedostatok bezproblémovo fungujúcich model-loaderov. Žiadny zo skúmaných nevyhovoval účelom tohto projektu. Bude teda potrebné zhotoviť si vlastný. Najideálnejšie bude prostredníctvom vytvoreného loadera zabezpečiť načítavanie moderného formátu Collady. V prípade neúspechu, bude možné zamerať pozornosť na implementáciu jednoduhšieho model-loaderu zabezpečujúceho načítavanie formátu OBJ. Tento formát je aj napriek svojmu veku stále použiteľný a stabilný.

Loader musí zabezpečiť jednoduché načítanie geometrie, a k nej patriacich atribútov, zo súboru. Medzi načítavané atribribúty musia patriť minimálne geometria, normály, textúrovacie súradnice. Navyše by bolo vhodné implementovať aj načítavanie materiálov a prípadne ciest k textúram. Poskytnutie jednoduchého rozhrania na vykreslenie modelu s pomocou OpenGL v kombinácii s použitím modulu kamery je ďalšou z funkcií, ktorú musí tento program poskytnúť. Bolo by vhodné, aby bol loader schopný umožniť počiatočné nastavenie pozície a natočenie modelu vzhľadom ku scéne. Kompletnú funkčnosť načítania

geometrie, ku ktorej patrí aj načítanie a vytvorenie textúr, musí byť tento model-loader schopný zabezpečiť sám.

9.3.3 Správa Shaderov

V rámci projektu bude vytvorená aj sada príkladov zameraná na vytváranie a používanie GLSL shaderov. Preto bude vhodné túto funkcionalitu integrovať do knižnice. Bude tak ľahšie prístupná vo všetkých relevantných programoch. Vytvorenie demonštračných programov na tvorbu a použitie shaderov sa javí ako rozumná myšlienka z dôvodu, že súčasné OpenGL v profile CORE nutne vyžaduje ich použitie. Vo svojom jadre už totiž neobsahuje žiadne príkazy kreslenia. Je umožnené iba nastavovanie stavu OpenGL, čiže grafickej karty.

Manažér musí obsahovať jednoduché a priamočiare rozhranie umožňujúce efektívnu prácu so shadermi. Musí umožniť prácu so všetkými druhmi shaderov, od základných vertex a fragment shaderov, cez geometry a tesselation control shadery až po tesselation evaluation shadery. Tie sa musia dať prostredníctvom shader manažéra ľahko načítať a musí byť zabezpečená funkcionalita všetkých dostupných funkcií nastavujúcich parametre shaderov. Teda použitie obdobných funkcií ako `glUniformMatrix2x3()`. V rámci práce so shadermi je súbor týchto funkcií najčastejšie sa rozširujúci naprieč verziami OpenGL. Pre túto funkcionalitu musí modul poskytnúť vhodné rozhranie.

Je žiadúce, aby manažér zastrešoval čo najrozsiahlejšiu funkcionalitu práce so shadermi, pričom by nekomplikoval zdrojový kód aplikácií. Musí obsahovať kompiláciu a linkovanie načítaných shaderov a výpis prípadne vzniknutých chýb pri tomto procese. Jednou z jeho najzákladnejších funkcií bude jednoduché zavedenie a odstránenie shaderu z GPU. Modul musí zároveň umožňovať kombináciu a spoločné použitie všetkých druhov shaderov. Jedna inštancia manažéru musí umožňovať vytvorenie simulácie pre kompletnú grafickú pipeline. Použite tohto modulu bude predpokladané až od verzie OpenGL 2.0 a preto bude možné tomuto faktu prispôsobiť implementáciu.

9.3.4 Ladenie chýb OpenGL

OpenGL štandardne ignoruje všetky vzniknuté chyby. Pri výskyte chybného príkazu sa tento ignoruje a pokračuje sa vykonávaním nasledujúceho príkazu. To má za následok vznik neočakávaných výsledkov nielen pre programátora začiatočníka.

Vytvorením funkcionality, ktorá by informovala vývojára o vzniknutej chybe, jej polohe a odkazovala by na riadok chybného kódu, by sa eliminoval zbytočný zmätok a optimalizoval čas nielen pri vytváraní projektu. Implementácia takéhoto „debuggeru“ musí povoľovať jeho deaktivovanie. Deaktivácia by mala prebiehať ideálne bez akýchkoľvek zásahov do existujúceho kódu. Prakticky je však možné tieto zásahy iba minimalizovať. Po deaktivácii nesmie mať „debugger“ žiadny vplyv na výkon aplikácie. Úplne postačí, aby bol tento modul nastaviteľný iba počas kompilácie programu, teda neimplementovaný pre runtime prostredie. Jeho integrácia by zároveň nemala zneprehľadňovať zdrojový kód.

9.4 Textúry

V súčasnosti neexistujú grafické karty, ktoré by boli schopné zobrazovať detaily na povrchu modelu definované čistou geometriou a materiálmi. Je to dôsledok nedostatočnej rýchlosti a kapacity pamätí, priepustnosti zberníc ako aj rýchlosti dnešných CPU. Aj dnes sa modeluje model jednoduchší a realistickosť, či detaily povrchu, sú zobrazované prostredníctvom textúr.

Textúry sú obvykle 2D obrázky, na ktoré sa odkazujú textúrovacie súradnice vrcholov modelu. Ku každej súradnici je z 2D obrázku priradená farba texelu pre daný pixel na obrazovke výsledného zobrazovaného modelu. Okrem povrchových 2D textúr, existujú aj iné druhy ako 1D či 3D textúry.

Pre potreby tohto projektu postačí používanie 2D textúr v rôznych variantoch. Nebudú implementované žiadne efekty, či modely obsahujúce 1D, alebo 3D textúry. Nebude potrebné pracovať ani s HDR obrázkami. V plnej miere postačia klasické a ľahko dostupné LDR obrázky. DDS textúry taktiež nebudú používané aj napriek svojej vysokej optimalizácií. Cieľom práce je vytvorenie jednoduchých, prehľadných a názorných programov využívajúcich OpenGL, nie vytvorenie vysoko-optimalizovanej aplikácie. Hotové textúry DDS sa navyše ťažko zháňajú a ich ručné vytvorenie je zložité a zdĺhavé, kvôli nedostatku príslušných aplikácií.

Preferovaný formát obrázkov textúr bude JPG. V tomto formáte je dostupných množstvo obrázkov k voľnému stiahnutiu online. Obrázky vo formáte JPG majú malú veľkosť vhodnú pre distribúciu a existuje mnoho editorov poskytujúcich ich modifikáciu a vytváranie.

Vytvorenie všetkých obrázkov pre textúry presahuje rozsah tejto práce. Preto obrázky, ktoré poslúžia ako textúry budú získané z internetových zdrojov, ktoré ich budú poskytovať s licenciou umožňujúcou ich ďalšie používanie a šírenie. Pravdepodobne bude potrebné niektoré textúry modifikovať, alebo na ich základe zhotoviť nové. Je tak možné že takýto zásah bude vyžadovať povolenia od autora. Preto bude lepšie vyberať voľnejšie licencie, ktoré toto priamo umožnia.

Bude potrebné získať ako smerové tak i dlaždicové farebné textúry pre príklady vysvetľujúce mapovanie textúr. Pri textúrovaní modelov sa počíta aj s prítomnosťou farebných textúr určených pre výhradné použitie na danom modeli.

Pre normálové textúry bude v práci stačiť použitie textúry prvého typu, teda vychyľovacie normálové mapy. Budú použité hlavne v efektoch pre zvýšeniu detailov na povrchu modelu. Textúry je potrebné obstarať aj s ich farebnými a výškovými variantmi. Normálové textúry sa vo väčšine prípadov nedajú použiť samostatne.

Na výškové mapy v tejto práci bude stačiť použitie obrázkov vo formáte LDR – teda 8 bitov na farebný kanál. Tieto textúry sú lepšie dostupné na internete a v prípade potreby sa

dajú ľahšie vytvoriť. Používanie HDR textúr by vyžadovalo integráciu pokročilejšieho načítavania obrázkov, čo by zrejme zneprehľadnilo a skomplikovalo zdrojový kód. Taktiež podstatný fakt je ten, že dnešné rozšírené prenosné počítače nie sú schopné s HDR textúrami pracovať. Pravdepodobne sa podarí výškové mapy obstarať pri získavaní ich odpovedajúcich normálových textúr, a teda ich samostatné vytvorenie nebude potrebné. Niekedy sa výšková textúra kóduje do jedného obrázku spolu s normálovou textúrou. Pre použitie v shaderoch by bolo lepšie obstarať tento typ textúry. To ale nieje vyžadované, nakoľko sa jedná len o miernu optimalizáciu.

Aj napriek dnešnej pomerne dobrej dostupnosti HDR sférických máp na internete bude postačovať pre príklady použitie klasických LDR. Podobne ako pre výškové mapy sa tieto v prípade potreby dajú ľahšie vytvoriť a ich používanie nevyžaduje integráciu pokročilejšieho načítavania obrázkov. Kubické mapy sa pre zlepšenie manipulovateľnosti a zníženie počtu prenosov bežne kódujú do jedného obrázku. Tým by vznikla potreba definovať korektné textúrovacie súradnice, prípadne by bolo nutné implementovať špeciálny texture-loader, ktorý vie takúto mapu automaticky rozdeliť na šesť samostatných textúr. Kvôli OpenGL je však vždy potrebné mať textúry k dispozícií oddelene. Pre zachovanie jednoduchosti kódu implementácie sa v tomto projekte budú používať kubické mapy vo forme šiestich samostatných textúr, bez ďalších optimalizácií. Navyše na demonštráciu použitia rôznych máp prostredia bude od programov vyžadované použitie aspoň jednej Lat-long environmentálnej mapy.

Keďže použitím mapy odleskov sa dosahuje kvalitný a pôsobivý vizuálny dojem, bude vhodné ju v programoch využiť k vytvoreniu aspoň jedného efektu. Pre použitie bude stačiť aj základná čierno-biela mapa i keď použitie mapy v odtieňoch sivej je výpočtovo rovnaké.

9.5 Model

K predvedeniu výsledku implementovaného efektu je potrebné použiť nejaký model. Vzhľadom k rôznorodosti efektov, ktoré bude potrebné vytvoriť je veľmi pravdepodobné, že nebude jeden model postačovať. Neexistuje totiž model, na ktorom by bolo možné dobre demonštrovať všetky efekty.

9.5.1 Obecné vlastnosti

Používané modely musia byť inšpirujúce. Musia byť príjemné na pohľad, nie úplne jednoduché a najlepšie, aby zobrazovali nejaký reálny predmet. Vo vyšších programoch tak potom bude užívateľ inšpirovaný výsledkom, ktorý bude vedieť dosiahnuť sám ihneď po naštudovaní implementácie programu. To pozitívne podporí ďalšie učenie. Výlučné používanie základných 3D objektov by mohlo pôsobiť kontraproduktívne. Užívateľ by mohol byť sklamaný jednoduchosťou scény, ktorú po preštudovaní všetkých programov bude schopný vytvoriť. Kvôli motivácii užívateľov, bude vhodnejšie vytvorenie zložitejšieho prostredia, ktoré bude obsahovať používanie externých knižníc, prácu s model-loaderom, načítavanie textúr a používanie zložitejšieho kódu. Výsledný efekt tak bude prepracovanejší a bude sa viac približovať realite. Zároveň sa užívateľ oboznámi s reálnym použitím OpenGL.

Získanie modelov zdarma z internetu s právami pre voľné používanie je ľahké. Avšak tieto dostupné modely sú v mnohých prípadoch škaredé, nekvalitne spracované a s mnohými ďalšími nedostatkami. Navyše nikdy neobsahujú všetky potrebné atribúty, ktoré by mal každý model obsahovať. Vytvorenie modelov vlastnoručne je tiež jednou z možností, avšak vyžadujúcou si mnoho času, ktoré bude zrejme vhodnejšie venovať tvorbe samotných programov. Navyše je k tomu potrebná pokročilá znalosť modelovacieho prostredia ako 3D Studio Max, alebo Maya. So získaním niektorého z týchto softvérov v študentskej verzii by nemal byť žiadny problém. Samotná firma Autodesk poskytuje na svojich internetových stránkach registrovaným študentom svoje programy. Možnosť obstarania hotového modelu z internetu a následne jeho oprava v študentskej verzii niektorého z uvedených programov sa javí ako najschodnejší kompromis.

9.5.2 Technické požiadavky na model

Každý model musí okrem určenia polohy vrcholov v priestore, čo je splnené u všetkých dostupných modelov, obsahovať aj množstvo iných atribútov. Umožní sa tým vytvorenie pokročilejších efektov. Samotná definícia polohy vrcholov v priestore by stačila na vytvorenie počítačovej grafiky odpovedajúcej úrovni zhruba spred dvadsiatich rokov.

Základnými atribútami, ktoré musí model bezpodmienečne obsahovať, sú:

- definícia geometrie,
- normály k vrcholom,
- textúrovacie súradnice.

Špeciálne požiadavky budú kladené na definíciu geometrie. Geometrická sieť modelu musí byť trojuholníková. Tým sa stanoví presná definícia použitých primitív pre model-loader. Jeho činnosť sa tým zjednoduší a zrýchli. Zvýši sa celkový výkon aplikácie, pretože dnešné grafické karty sú optimalizované práve pre prácu s trojuholníkmi. Pozícia vrcholov v priestore by nemala byť rozložená jednoducho. Objekt by nemal byť úplne symetrický. Mal by obsahovať zložitejšiu geometriu, aby sa dosiahol plnohodnotný vizuálny efekt. Nakoniec musia byť správne nastavené normály aj textúrovacie súradnice.

Popri základných atribútoch existujú aj ďalšie často používané, ktoré by bolo vhodné, aby model obsahoval:

- tangenty a bitangenty,
- faktory zatienenia – occlusion factors,
- prípadne iné.

9.6 Úlohy

Programy budú implementované v oboch pipeline OpenGL. Verzia OpenGL 1 bude používaná pri vytváraní programov použitím fixnej pipeline. Táto sada programov bude demonštrovať použitie vstavanej funkcionality OpenGL. Vo verzii 2 budú implementované príklady využívajúce shadere. Zároveň budú niektoré príklady s rovnakým účelom vytvorené pomocou oboch prístupov kvôli možnosti porovnania spracovania. Vzhľadom na rozšírenosť prenosných počítačov je v súčasnosti použitie vyšších verzií OpenGL kontraproduktívne. Jeden príklad by však mohol byť vytvorený pre novú verziu OpenGL 3, alebo OpenGL 4.

Fixná	**Programovateľná**	**Oba prístupy**
– Okno	– Integrácia shaderov	– Osvetlenie
– Projekcia	– Osvetlenie	– Materiály
– Transformácie	– Image processing	– Textúry
– Vytváranie telies	– Detaily povrchu	– Hmla
– Definovanie farieb	– RTT	
– Normály	– Mapovanie prostredia	
– Nastavenie textúrovacích jednotiek	– Pokročilejšie efekty	
– Tieňovanie		
– Blending		
– Antialiasing		

Niektoré oblasti budú vytvárané ako pre fixnú, tak aj pre programovateľnú pipeline. Zamýšľané je to cielene, z dôvodu demonštrácie tvorby rovnakých efektov pomocou oboch metód.

10 Implementácia

Projekt je implementovaný v jazyku C++. Projekt je zmýšľaný ako podpora výučby OpenGL hlavne na elektrotechnickej fakulte ČVUT. Z vlastnej skúsenosti viem, že študenti študujúci odbor počítačová grafika, majú zväčša najrozsiahlejšie skúsenosti práve v tomto programovacom jazyku a medzi počítačovými grafikmi má C++ celosvetovú širokú podporu. Sú preň dostupné rôzne frameworky a hotové riešenia ako kvalitný framework pre matematiku – GLM, ktorý je nutný pre použitie OpenGL v profile CORE. Ďalej ostatné kvalitné frameworky podporujúce vývoj OpenGL aplikácií, či už určené ku správe okien a aplikácie, načítavania obrázkov, alebo načítavania 3D modelov vrátane modelov vo formáte Collada, sú dostupné prevažne v tomto jazyku.

Projekt bol vyvíjaný vo vývojovom prostredí Microsoft Visual Studio 2008 Professional. Toto prostredie je dostupné zdarma pre študentov ČVUT v rámci školského programu MSDN eAcademy. Práve MSVC je prostredie, ktoré sa v rámci odbornej komunity najčastejšie používa. Preto je množstvo príkladov pracujúcich s OpenGL dostupné práve v projektových súboroch MSVC. Tým sa uľahčí preklad a využitie týchto príkladov najmä začínajúcimi užívateľmi.

Základom všetkých implementovaných aplikácií je nová knižnica SFML. Jej vlastnosti boli už uvedené v kapitole 5.1. Na tomto mieste ju len stručne pripomeniem a uvediem dôvody jej výberu. Jedná sa o multiplatformnú multimediálnu knižnicu, ktorá zaisťuje súčasne niekoľko funkcionalít. Zabezpečuje prístup k OpenGL príkazom, vytvorenie systému okien, spracovanie vstupov, prácu so sieťou a mnohé iné. Knižnica je voľne distribuovateľná a má výbornú dokumentáciu. Hoci je objektovo-orientová, je možné ju použiť i pre procedurálny návrh aplikácie. Podporuje oba typy. Je intuitívna a jednoduchá na naučenie i používanie.

SFML zaisťuje splnenie hneď niekoľkých požiadavkov potrebných pre vytvorenie aplikácie:

- sprístupňuje príkazy OpenGL až do verzie 3,
- zaistením programového cyklu tvorí základ aplikácií,
- spravuje okenný systém,
- spracováva a reaguje na užívateľské vstupy,
- načítava obrázky určené pre použitie v textúrach.

Zabezpečením všetkých týchto funkcionalít, vrámci jednej knižnice, sa znižuje počet závislostí nutných k fungovaniu aplikácií. Nezabezpečuje však celú potrebnú funkcionalitu. Chýbajú ešte funkcie:

- vytvorenie a správa pohľadu do scény – kamera,
- podpora matematických výpočtov,
- načítavanie modelov,
- komplexná správa shaderov, knižnica SFML poskytuje spravovanie iba fragment shaderov ako efektov post-processingu.

Táto funkcionalita bola doprogramovaná, pričom boli využívané iné knižnice. Popis tvorby týchto funkcií je popísaný v ďalšom texte.

10.1 Základ

Ako základ aplikácií bola zvolená už spomínaná knižnica SFML. Aj napriek procedurálnemu spracovaniu OpenGL a možnosti knižnice SFML vytvoriť čisto procedurálnu aplikáciu bol vytvorený čiste objektovo orientovaný variant aplikácie. Je vhodné, aby boli nový používatelia od začiatku práce s OpenGL naučený pracovať s objektovo orientovanou aplikáciou. V súčasnej dobe sú študenti oboznamovaný v rámci výučby zväčša len s používaním zastaralého procedurálneho návrhu aplikácií, používajúc GLUT.

Celá základná aplikácia pozostáva z jednej triedy nazvanej príznačne *Application*. Spustenie aplikácie nastáva po inicializácii metódy Run, a nie ihneď po vytvorení inštancie triedy *Application*. Táto vlastnosť je vytvorená zámerne. SFML totiž podporuje multi-threading dedením od triedy *Thread*. Jedinou podmienkou je implementácia metódy *Run*. Takto je rozhranie jednoducho a už od samého začiatku pripravené k použitiu vo viacvláknovom prostredí.

Aplikácia je vytvorená ako konzolový typ so vstupným bodom prostredníctvom funkcie *main*. Táto funkcia obsahuje len zdrojový kód potrebný k vytvoreniu inštancie triedy *Application* a zavolanie metódy *Run*. Tým aplikácia započne inicializačný proces nasledovaný vstupom do programovacieho cyklu.

Kód každej aplikácie je rozdelený do štyroch súborov, ktorými sú: *Application.hpp Application.cpp Init.cpp* a *Render.cpp*. Súbor *Application.hpp* je zdieľaným hlavičkovým súborom, ostatné tri sú výkonné.

10.1.1 Application.cpp

Súbor *Application.cpp* obsahuje zdrojový kód, ktorý sa mení len veľmi zriedka. Patria sem príkazy zabezpečujúce:

- deklaráciu a definíciu triedy aplikácie,
- vytvorenie aplikácie,
- základnú inicializáciu premenných,
- vytvorenie kontextu a okna OpenGL,
- odchytávanie udalostí a správu vstupov.

10.1.2 Init.cpp

Oddelenou skupinou funkcií triedy *Application* je súbor *Init.cpp*. Obsahuje funkcie zamerané na inicializáciu súvisiacu priamo s nastavením OpenGL, poprípade súvisiacu s riešeným problémom. Okrem inicializácii stavu OpenGL sa touto skupinou funkcií zabezpečuje načítavanie modelov, textúr, shaderov a vytvorenie a inicializácia rôznych bufferov ako FBO či PBO.

10.1.3 Render.cpp

Podobne ako Init.cpp je aj Render.cpp oddelenou skupinou triedy Application. Avšak obsahuje funkcie súvisiace s vykresľovaním scény a efektov v scéne. Uvedená štruktúra sa osvedčila počas celej implementácie. Nebolo ju potrebné nijako meniť. Súčasne stačila na splnenie rôznych požiadavkov vyplývajúcich z rôznorodosti programov. Tým sa splnil jej zamýšľaný cieľ. V každom programe sa zvýraznilo jadro problému a v pozadí sa izolovali spoločné funkcionality súvisiace s ovládaním a behom programu.

10.1.4 Hlavný cyklus programu

Hlavný cyklus programu je implementovaný v rámci metódy *Run* ako nekonečný cyklus s podmienkou ukončenia na začiatku. Cyklus sa vykonáva v nasledujúcom postupe. Scéna sa vykreslí do *back-bufferu*. Potom sa výsledok zobrazí v okne. Dôjde k presunutiu scény z *back-bufferu* do *front-bufferu*. Túto funkciu zaobstaráva SFLM automaticky. Po zobrazení scény v okne sa skontrolujú RT vstupy klávesnice a myši. Nakoniec sú skontrolované vzniknuté udalosti.

K ukončeniu cyklu dochádza v momente uzatvorenia okna aplikácie. Je to zabezpečené funkciou kontroly udalostí – *handleEvents*. Počas vytváraní aplikácií sa vyskytol problém vyplývajúci z nesprávneho poradia funkcií hlavného cyklu. Je potrebné zaistiť, aby k došlo k uzatvoreniu okna (čiže k volaniu funkcie *handleEvents)* až po vykonaní vykresľovania. Teda až po volaní funkcie *Render*. Pri opačnom poradí dôjde k zavolaniu príkazov OpenGL, vo funkcii *Render*, nad neexistujúcim kontextom. Vyvolá sa tým neplatný prístup do pamäte s následným pádom celej aplikácie bez možnosti odhalenia chyby.

10.1.5 Prístup k funkciám OpenGL

Použité vývojové prostredie MSVC 2008 podporuje priamy prístup k funkciám OpenGL iba do verzie 1.1. Pri použití SFML je možné volať funkcie OpenGL po jej verziu 3. Interne sa totiž používa staršia verzia knižnice GLEW. Kvôli súčasnému rýchlemu rozvoj funkcionality OpenGL tak možnosti SFML v tomto smere rýchlo zastarávajú. Preto bude vhodnejšie priamo do programov integrovať podporu pre najnovšiu verziu OpenGL.

Pri integrácii externej podpory je na výber použitie knižníc GLEW a GLEE. Pri rozhodovaní, ktorá z týchto špecifických knižníc bude používaná, zohrala najväčšiu rolu aktuálnosť poslednej revízie. Vzhľadom na toto kritérium bola zvolená knižnica GLEW, aj napriek tomu, že má špatne vyriešený návrh v porovnaní s knižnicou GLEE. U GLEW je totiž potrebné zaistiť inicializáciu, čo znamená problém pri integrácii do knižníc. GLEE zase podporuje OpenGL „iba“ do verzie 3.1, zatiaľ čo GLEW do verzie 4.1.

Použitím poslednej verzie GLEW sa otvára možnosť využívania OpenGL do v súčasnej dobe najnovšej revízie OpenGL 4.1. V rámci tvorby aplikácií došlo iba vo výnimočných prípadoch k využitiu tohto potenciálu. Obyčajne vytvorené aplikácie používajú OpenGL len do verzie 2. Sú však pripravené na modifikáciu budúcimi užívateľmi na prácu s najnovšími verziami OpenGL. Použitím tohto návrhu je súčasne zabezpečený upgrade jednoduchou zámenou knižnice za jej vyššiu verziu. Aj v prípade príkladov s fixnou pipeline, ktorým postačuje prekladačom poskytovaná revízia OpenGL 1.1, je v nich integrovaná knižnica GLEW, kvôli zachovaniu jednotnosti.

10.1.6 Správa okien a kontextu OpenGL

Správa okien a kontextu OpenGL je vykonávaná prostredníctvom implementácie knižnice SFML. Knižnica podporuje ľubovolný kontext plynúci z OpenGL, ale iba v profile COMPATIBILITY. Nebráni to však vytváraniu aplikácií v profile CORE. Rozdiel spočíva v tom, že v prípade programovania pre tento profil musí programátor s uvážením vyberať používané príkazy. OpenGL sa nestará o to, či ich je možné použiť. Podporou najnovšieho kontextu je preto zaistená možnosť tvorby aplikácií v novších verziách OpenGL. V budúcnosti pravdepodobne dôjde k rozšíreniu aplikácií, pričom však nebude potrebné meniť túto knižnicu.

V programoch je zväčša používaný kontext OpenGL 2, ktorý je rovnaký ako kontext staršej verzie OpenGL 1. Jedná sa o základný kontext bez možnosti výberu medzi profilmi CORE a COMPATIBILITY. V dôsledku používania knižnice, nieje možné špecificky zvoliť buffery, ktoré sa vytvoria v rámci kontextu. Vždy dochádza k vytvoreniu maximalistického kontextu, teda kontextu obsahujúceho všetky potrebné buffery.

Okno sa vždy vytvorí automaticky v móde RGBA s *double-bufferom*. Dochádza k automatickému vytvoreniu *depth*, *stencil*, *akumulačného*, *alpha* ako aj iných bufferov. Tieto vystačujú pre všetky dnes vytvárané grafické programy. V programoch ale nikdy nie sú použité všetky buffery naraz. Dochádza tak zbytočne k plýtvaniu pamäte CPU i GPU. V iných knižniciach ako napr. GLUT je možné požadované buffery presne špecifikovať. Pri vytváraní podobných programov ako v celom projekte (hlavne s podobným relatívne nízkym rozlíšením) sa tým ušetrí pamäť na grafických kartách nanajvýš v rádoch jednotkách megabajtov. Keďže dnešné grafické karty bežne disponujú pamäťou v rádoch stoviek megabajtov je to pre účel tohto projektu zanedbateľné. Bohužiaľ v praxi pri použití veľkého rozlíšenia v kombinácii s multi-okennou aplikáciou môže selekcia bufferov ušetriť pamäť aj v stovkách megabajtov. Samotné množstvo iniciovaných bufferov nemá vplyv na výkon aplikácie. Knižnica neumožňuje vytvorenie *multisample* bufferu na úrovni kontextu OpenGL, teda pre použitie s *GL_MULTISAMPLE*. Knižnica toto nahradzuje možnosťou vyžiadania si multisample okna priamo od OS. Nieje ho tak možné ovplyvniť príkazmi OpenGL. Počas

existencie okna je multisample trvale zapnutý bez možnosti jeho deaktivácie. Vo vytvorených aplikáciách je tento typ okna použitý iba raz, pri demonštrácii antialiasingu metódou FSAA.

Automatické buffery s rezervou postačujú pre všetky aplikácie, ktoré boli vytvorené v rámci tohto projektu. Vytvorené aplikácie sú v rozlíšení 800x600. Hoci je umožnené špecifikovať bitové hĺbky vybraných bufferov, vo všetkých príkladoch je použité automatické nastavenie. Aj keď SFML disponuje možnosťou vytvárania viacerých kontextov a okien OpenGL v rámci jednej aplikácie, ostáva táto vlastnosť mojim projektom nevyužitá.

10.1.7 Odchytávanie systémových udalostí a užívateľských vstupov

Implementáciou SFML je zabezpečené aj odchytávanie systémových udalostí a užívateľských vstupov. Knižnica umožňuje vytvorenie RT spracovania vstupov ako aj spracovanie udalostí.

RT spracovanie je vhodnejšie pri vytváraní animovaných scén. Prakticky sa jedná o nekonečný renderovací cyklus s kontrolou udalostí. Nevýhodou je neustále vyťaženie CPU. Výhodnejšie sa zdá byť interaktívne spracovanie. To vykoná jeden renderovací cyklus, následne uspí vlákno aplikácie a čaká na užívateľský vstup. Tým dochádza k šetreniu systémového času ako GPU, tak aj CPU.

Pre všetky programy v projekte by pôvodne postačovalo interaktívne spracovanie udalostí. Nakoniec bolo ale použité RT spracovanie. K tomuto rozhodnutiu došlo po zistení chýb vyplývajúcich z použitia interaktívneho spracovania. Neprirodzené správanie a trhanie obrazu vznikajúce pri vstupe z myši, bolo spôsobené nepravidelným zasielaním udalostí myši do aplikácie. Toto zasielanie zabezpečuje operačný systém a je plne v jeho kompetencii. Nieje ho možné nijak ovplyvniť. RT spracovanie tak poskytuje najkvalitnejšiu interakciu.

Odchytávanie udalostí a užívateľských vstupov pracuje v dvoch metódach: *handleEvents* a *handleInputs*. V metóde *handleEvents* dochádza k odchytávaniu a spracovaniu udalostí systému a vstupných zariadení, a tiež k zabezpečeniu ukončenia aplikácie. V metóde *handleInputs* prebieha RT kontrola vstupných zariadení – konkrétne myši a klávesnice.

10.1.8 Načítavanie obrázkov

Načítavanie obrázkov je posledná funkcionalita, ktorú zastrešuje knižnica SFML. Funkcionalita je využívaná v rámci súboru *Init.cpp*. Načítavanie sa často používa v príkladoch, kde je potrebné dodatočné načítanie textúr. Jedná sa obvykle o príklady demonštrujúce prácu s textúrami a pokročilé efekty GLSL.

10.2 Knižnica TGL

Ostatná funkcionalita vyžadovaná pre beh programov bola implementovaná vlastným zdrojovým kódom v rámci knižnice TGL za použitia ďalších knižníc. Knižnicu TGL sa podarilo implementovať presne k splneniu všetkých ostatných, v návrhu požadovaných vlastností. Je rozdelená do štyroch modulov: kamera, „debugger“, model-loader, manažér shaderov. Implementovaná je ako hlavičková knižnica, čiže je možné ju používať bez nutnosti pridružovať jej zdrojový kód k výkonnej časti projektu. Stačí vykonať iba jednoduché načítanie hlavičiek.

10.2.1 Kamera

Z možnosti diskutovaných v návrhu, viď kapitola 9.3.1, bola implementovaná pozorovacia kamera. Jedná sa o kameru, ktorej počiatok pozorovacieho vektoru sa pohybuje po guli s pevne definovaným stredom do ktorého vždy smeruje. Okolo tohto stredu je kamera schopná rotovať a približovať resp. odďalovať sa od neho. Tento typ kamery plne vyhovuje požiadavkom všetkých implementovaných programov. Modul kamery nespolieha na dostupnosť OpenGL príkazov v prekladači, ale k funkcionalite spojenej s OpenGL API využíva knižnicu GLEW.

Kamera je vytvorená čo možno najjednoduchšie. K vytvoreniu pohľadu do scény využíva priamo funkcie OpenGL. Tieto funkcie parametrizuje svojim aktuálnym vnútorným stavom, teda súradnicami počiatku, azimutom a zenitom. K jednoduchému ovládaniu pohľadu pomocou myši kamera podporuje tzv. trackball. Trackball je implementovaný v zjednodušenej forme, ale pre potreby projektu to plne postačuje.

Získanie aktuálneho pohľadu v poli typu `float`, umožňuje kamera prostredníctvom vygenerovania 4x4 matice. Pohľad vo formáte odpovedajúcom matici 4x4, je potrebný hlavne v prípade používania akejkoľvek verzie OpenGL v profile CORE. Zároveň je ju možné s úspechom využiť aj v shaderoch vyžadujúcich k funkčnosti maticu pohľadu. Získanie matice pohľadu zabezpečuje externá knižnice pre matematiku – GLM.

10.2.2 Model loader

Model-loader využíva k načítavaniu geometrie knižnicu Assimp. Jej pomocou bolo možné vytvoriť stabilný a robustný nástroj načítavajúci modely rôznych formátov, vrátane Collady. Podobne ako modul kamery, ani model-loader nespolieha na dostupnosť OpenGL v prekladači. Funkcionalitu OpenGL využíva cez knižnicu GLEW. Jedná sa o najzložitejšiu časť knižnice TGL s najvyšším počtom závislostí.

Modul pozostáva z troch tried: `MeshObject`, `MaterialObject` a `Model`. `Model` obsahuje pole objektov typu `MeshObject` reprezentujúcich geometriu častí modelu. Pre každý `MeshObject` existuje práve jeden `MaterialObjekt`, ktorý uchováva vlastnosti materiálu

odpovedajúce danej časti geometrie. Je obvyklé, že celý model pozostáva z jedného *MeshObjektu* uchovávajúceho celú geometriu, a k nej priradený jeden materiál resp. textúru. Hlavné rozhranie modulu tvorí trieda *Model*, ktorá je zároveň najjednoduchšou časťou, nakoľko poskytuje iba parametrizovanie vykresľovania modelu.

Pomocou knižnice Assimp je načítaná geometria modelu s hierarchiou scény. S geometriou sú zároveň načítané všetky materiály a interne k nej priradené názvy použitých textúr. Následne sú aplikované funkcie pre optimalizáciu načítanej geometrie a grafu scény. Vrámci optimalizácie sú vykonávané tieto akcie:

- odstránenie animácií, kostí, svetiel, kamier a explicitne zadaných farieb z grafu scény,
- vygenerovanie jednotkových normál ku každému vrcholu, pokiaľ je to potrebné,
- normalizácia existujúcich normál,
- výpočet tangent a bitangent v prípade, že ich model neobsahuje,
- triangulácia geometrickej siete, pokiaľ sieť nieje trojuholníková,
- optimalizácia geometrickej siete odstránením duplikovaných vrcholov,
- kolaps celého grafu scény/hierarchie, do stromu obsahujúceho iba koreňový uzol a listy, samozrejme sa zachovávajú transformačné vzťahy.

Po optimalizácii je celý model dostupný po častiach ako jednoduché pole. Z tejto štruktúry je kód prevedený do internej reprezentácie triedy *Model*. Všetky vrcholy, indexy a normály modelu sú reprezentované poliami typu *float*. Tým sa zjednoduší použitie týchto informácií v príkazoch OpenGL. V poslednom kroku je, za pomoci pred-pripravenej internej štruktúry, vytvorený Display List potrebný pre okamžité vykresľovanie. Po načítaní objektu je interná štruktúra Assimp odstránená. Vnútorná štruktúra model-loaderu však zostáva zachovaná počas celej existencie objektu, aby sa umožnil prístup externých aplikácií k vrcholom a ich atribútom.

Trieda *Model* disponuje funkcionalitou potrebnou na vykresľovania objektov bez načítaných materiálov. V tomto prípade musí príslušná aplikácia zaistiť, aby sa materiály načítali a aplikovali pred vykreslením samotnej geometrie modelu. Vhodné použitie tejto vlastnosti je v prípade programov zobrazujúcich používanie materiálov resp. textúr. Ďalej trieda umožňuje vykreslenie konkrétnej časti modelu, pokiaľ sa sám skladá z viacerých častí. To je vhodné na použitie v prípadoch, kedy je nutné iba na jednu časť modelu aplikovať shader. Nakoniec je umožnené vykreslenie celého modelu pomocou polí vrcholov, tzv. *vertex-arrays*. Táto funkcia sa hodí v prípadoch, kedy nieje umožnené použitie DL, ale je ich použitie nežiadúce.

Načítavanie textúr modelov realizuje trieda *Model* pomocou knižnice SFML. Vzhľadom k tomu, že sa využíva vo všetkých programoch, sa týmto znížila závislosť o jednu knižnicu, ktorá by bola potrebná pre samostatné načítavanie textúr. Model-loader je tak schopný načítať textúry pre model v bežných LDR formátoch.

10.2.3 Manažér shaderov

Správa shaderov je zabezpečená centralizovane, prostredníctvom manažéra. Manažér k svojej funkcionalite využíva iba knižnicu GLEW, ktorá mu sprístupňuje príkazy OpenGL nutné pre prácu so shadermi. V OpenGL existujú dve verzie príkazov umožňujúce prácu so shadermi. Verzia príkazov použitá v OpenGL 1.5 využíva extenzie ARB. Od verzie OpenGL 2 sú príkazy umožňujúce prácu so shadermi implementované priamo do jeho jadra.

V tomto projekte prebehla implementácia manažéra shaderov pomocou funkcií OpenGL 2, podľa požiadaviek uvedených v návrhu – kapitola 9.3.3. Jednoduchým spôsobom manažér spracováva všetky použité druhy shaderov a dokáže ich vzájomne kombinovať do výsledného programu. Zároveň je umožnené načítavanie jedného typu shaderu z viacerých súborov. V rámci týchto súborov môže existovať iba jedna funkcia `main`. Manažér poskytuje rozhranie k lokalizácii premenných. Vďaka takejto implementácii je použiteľný v akejkoľvek súčasnej i budúcej verzii OpenGL. Nesnaží sa vyčerpávajúcim spôsobom zastrešiť funkčnosť všetkých súčasných požiadaviek. Naopak, poskytuje jednoduché rozhranie dovoľujúce ho použiť v parametroch príkazov shaderov OpenGL. Manažér umožňuje pridávanie shaderov a aktiváciu resp. inaktiváciu programu. V rámci modulu boli implementované informačné a chybové výpisy z procesu kompilácie shaderov vypisované na štandardný výstup konzole.

10.2.4 Ladenie chýb OpenGL

Posledným, najmenším a najjednoduchším modulom knižnice TGL je modul umožňujúci odhaľovanie chýb, tzv. debugger. Kvôli jeho jednoduchosti je implementovaný procedurálnym spôsobom. Pre malú implementáciu je tento spôsob oveľa výhodnejší a flexibilnejší. Použitie jazyka C++ zabezpečilo možnosť splnenia všetkých požiadaviek uvedených v návrhu – kapitola 9.3.4. Všetky požiadavky sa podarilo splniť implementáciou makier jazyka C++. Detekcia chýb prebieha v štandardnej funkcii, ktorá získa aktuálnu chybu OpenGL a na základe typovej tabuľky zobrazí chybovú správu.

V rámci menného priestoru `tgl::debug` sú dostupné funkcie umožňujúce kontrolu chýb. Celé čaro systému spočíva v definícii makra `checkErr` podmieneného definíciou premennej `TGL_DEBUG` v programe kdekoľvek pred vložením hlavičky `Debug.hpp`. Môžu nastať dve situácie vyplývajúce z toho, či je premenná `TGL_DEBUG` definovaná, alebo definovaná nieje. V prvom prípade je makro `checkErr` definované tak, aby sa zavolala výkonná funkcia OpenGL automaticky nasledovaná volaním funkcie pre kontrolu chyby OpenGL. V prípade, že premenná nieje definovaná, makro `checkErr` je definované tak, aby sa nahradilo volaním samotnej funkcie. Pri nedefinovaní `TGL_DEBUG` nemá „debugger“ žiadny vplyv na výkon aplikácie.

Každé volanie funkcie OpenGL, u ktorého chceme zistiť chybu volaním `checkErr(funkcia_OpenGL)`, je možné vďaka tomuto spôsobu obaliť. Nastavením premennej `TGL_DEBUG` je možné určiť, či sa pri preklade programu všetky volania makra `checkErr` premenia na volanie priradenej výkonnej funkcie nasledovanej volaním kontroly chyby, alebo iba na volania samotnej výkonnej funkcie. Podobná implementácia bola uvedená v rámci knižnice SFML, odkiaľ bola prevzatá a vylepšená.

10.3 Implementované príklady

Podľa návrhu v kapitole 9.6 bola približne polovica programov implementovaná v prvej verzii OpenGL. Zvyšná časť programov bola implementovaná v druhej verzii OpenGL, tak aby využívali shadery GLSL. Vyššie verzie OpenGL by bolo vhodné implementovať v profile CORE. Bohužiaľ knižnica SFML nedisponuje funkciami pre vynútenie tohto profilu. Zároveň z analýzy vyplýva, že použitie týchto verzií by bolo kontraproduktívne, nakoľko nieje rozšírený hardvér s touto podporou.

Fixná pipeline

Okno

– vytvorenie kontextu a okna, vymazávanie okna

Projekcia

– ortogonálna, perspektíva

Transformácie

– posun, rotácia, mierka

Vytváranie telies

– 2D trojuholník, 3D tetrahedron, použitie komplexného modelu cez TGL

Definovanie farieb

– nastavenie farby vrcholov

Normály

– definícia normál, normalizácia

Nastavenie textúrovacích jednotiek

- priamy výber texelu, bilineárne filtrovanie

Tieňovanie

– flat shading, smooth (gourad) shading

Blending

– transparentnosť telies

Antialiasing

– čiary, FSAA

Programovateľná pipeline

Integrácia shaderov

- integrácia shaderov do aplikácie OpenGL

Osvetlenie

– hemisphere lighting, image based lighting, spherical harmonics

Materiály

– základne použitie materiálov

Image processing

– blur, edges, greyscale, sephia

Zvyšovanie detailnosti povrchu

– emboss, normal, bump, paralax, steep paralax mapping

Render to texture (RTT)

– depth-map, normal-map, depth of field

Mapovanie prostredia

– sphere mapping, cube mapping, equirect mapping

Špeciálne efekty

– cartoon shader, fur, carpet

Oba prístupy

Osvetlenie

– per-vertex, per-fragment osvetlenie

Materiály

– ambient, diffuse, specular, emmision, ADS, color material

Textúry

– načítanie a mapovanie textúr

Hmla

– lineárna, exponenciálna, súradnice hmly, per vertex/fragment hmla

Všetky implementovaní programy sú dostupné na DVD v prílohe D. Pre demonštráciu vytvorených programov je v nasledujúcej kapitole uvedený vzorový príklad – Cartoon shading.

10.4 Vzorový program – Cartoon shading

Cartoon shading, často tiež známy ako Toon, či Cel shading, patrí do rodiny nefotorealistických efektov. Cieľom NPR nieje vytvárať realistické vizualizácie. Naopak je zámerom generovať štylizovanú grafiku. Napodobujú sa v ňom najčastejšie „rukou kreslené" štylizácie ako kresba, šrafovanie, či maľba. Teda grafika používaná v kreslených filmoch, či komiksoch. V nasledujúcom príklade bude popísaný shader pre vytvorenie tohto efektu.

Kresleného efektu je možné docieliť rôznymi i zložitými cestami. Existujú spôsoby využívajúce dokonca až tri rendrovacie priechody, navyše za použitia pokročilých techník. V tomto príklade bude efekt implementovaný v jednoduchej podobe. Zároveň predvedená implementácia dosahuje vizuálne veľmi dobrých výsledkov.

Celá technika je realizovaná počas jedného rendrovacieho priechodu prakticky ako post-proces efekt. Výkonný kód sa tak nachádza v jedinom fragment shaderi. Efekt bohužiaľ vyžaduje súčasné použitie vertex shaderu, ktorý musí do FS predať informácie o normále a aktuálnom vektore pohľadu. Z uvedeného tak vyplýva požiadavka na prítomnosť normál v modeli. Cartoon-shading je založený na vytváraní konštantných farebných plôch s ostrými prechodmi. Mnou implementovaný efekt poskytuje takéto prechody tri – zvýraznenie v odrazivej časti, difúznej časti a plochy tvoriace siluetu.

Hlavná aplikácia iba iniciuje vykreslenie samotného objektu a nastavuje parametre shaderu. Zároveň s tým musí pre shader identifikovať textúrovaciu jednotku, na ktorej sa nachádza difúzna textúra modelu. Keďže je objekt vykresľovaný za pomoci model-loaderu knižnice TGL, nachádza sa táto na nultej pozícií. Ďalej aplikácia nastavuje parametre ako pozíciu svetla, či farbu siluety predmetu.

Výpis 1: Render.cpp

```
shader.bind();
glUniform1i(...("colorMap"),0);
glUniform3f(...("lightPos"), 0.f, 0.f, 1.f);
glUniform1f(...("diffuseTreshold"), 0.5f);
glUniform1f(...("specularTreshold"), 0.2f);
glUniform4f(...("silhouetteColor"), 0.f, 0.f, 0.f, 1.f);
glUniform1f(...("silhouetteTreshold"), 0.35f);
model.Draw();
shader.unbind();
```

Vertex shader implementuje základné transformácie príchodzieho vrcholu a textúrovacích súradníc pre fragment shader. Zároveň zapisuje na výstup normálu a vrchol transformované do súradníc kamery. Za ich pomoci sú potom vo FS vypočítané farebné prechody a silueta objektu.

Výpis 2: Cartoon.vert

```
varying vec3 normal;
varying vec3 vertex;

void main()
{
      vertex           = vec3(gl_ModelViewMatrix * gl_Vertex);
      normal           = gl_NormalMatrix * gl_Normal;
      gl_TexCoord[0]   = gl_TextureMatrix[0] * gl_MultiTexCoord0;
      gl_Position      = ftransform();
}
```

Výkonný kód je obsiahnutý vo fragment shaderi. Z aplikácie a z vertex shaderu sú k dispozícií všetky potrebné parametre. Hlavne sa jedná o interpolovanú normálu a pozíciu bodu v priestore, pozíciu svetla, textúru objektu a farbu siluety.

Výpis 3: Cartoon.frag - Deklarácia premenných

```
varying vec3 normal;
varying vec3 vertex;

uniform vec3  lightPos;
uniform float diffuseTreshold;
uniform float specularTreshold;
uniform vec4  silhouetteColor;
uniform float silhouetteTreshold;
uniform sampler2D colorMap;
```

Na začiatku programu sú získané základné dynamické informácie ako difúzna a odrazivá farba povrchu a predpočítané vektory potrebné pre výpočty siluety a farebných prechodov.

Výpis 4: Cartoon.frag - Predvýpočet vektorov a farby fragmentu

```
vec4 diffuseColor  = texture2D(colorMap, gl_TexCoord[0].st);
vec4 specularColor = pow(diffuseColor, vec4(2.0));

vec3 vectorToEye   = normalize(-vertex);
vec3 vectorToLight = normalize(lightPos - vertex);
vec3 halfVector    = normalize(vectorToEye + vectorToLight);
```

V ďalšom kroku je spočítaný faktor pre určenie siluety na základe výpočtu skalárneho súčinu normály a obráteného vektoru pohľadu. Použitá normála nieje normalizovaná. Tým sa dosiahne nepresnej a nerovnomernej siluety a dôjde tak k zvýšeniu vizuálnej kvality. Skalárny súčin preto, lebo jeho hodnota vždy udáva kosínus uhlu vektorov. Čím je uhol medzi týmito vektormi menší, tým väčší je jeho kosínus a teda uhol pohľadu na povrch je ostrejší. Stanovením medznej hodnoty z intervalu [0, 1] určíme od akého uhlu už nebude vykresľovaný povrch objektu, ale iba silueta.

Výpis 5: Cartoon.frag - Výpočet siluety

```
float silhouetteFactor = max(dot(normal, vectorToEye), 0.0);
if (silhouetteFactor < silhouetteTreshold) {
      gl_FragColor = silhouetteColor;
```

V prípade, že je uhol medzi vektorom pohľadu a normálou menší než stanovená medzná hodnota, je potrebné vyhodnotiť farbu povrchu v tomto bode. Na začiatku sa vypočíta, či sa práve spracovávaný pixel nachádza v oblasti, kde sa počíta odrazivá zložka materálu. To sa vyhodnotí opäť na základe uhlu normály a vektoru pohľadu. Hodnota je samozrejme umocnená empirickým faktorom určujúcim ostrosť odrazu a tým okolie v ktorom sa bude zložka počítať.

Výpis 6: Cartoon.frag - Výpočet odrazivej časti

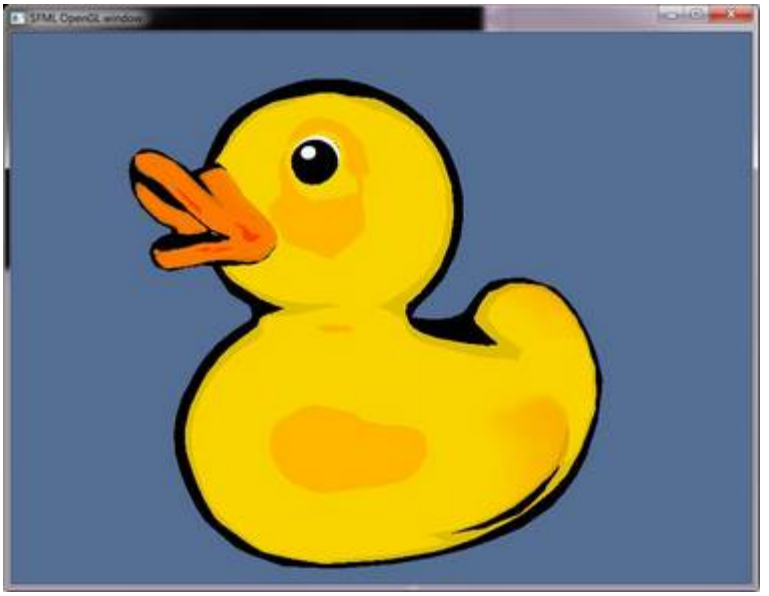
```
} else {
float specularFactor = pow(max(dot(normal, vectorToEye), 0.0), 20.0);
```

Na záver sa vyhodnotí podmienka, či sa spracovávaný pixel nachádza v oblasti pre ktorú sa vyhodnocuje odrazivá, alebo difúzna zložka. Na tomto základe sa nastaví výsledná farba príslušného fragmentu.

Výpis 7: Cartoon.frag - Nastavenie výslednej farby pixelu na základe pozície

```
if (specularFactor < specularTreshold)
	gl_FragColor = diffuseColor * 0.98;
else
	gl_FragColor = specularColor;
```

Predchádzajúcimi výpočtami sa vždy najprv určilo miesto na povrchu pre ktoré je potrebné vypočítať farbu – teda či sa pixel nachádza v odrazivej oblasti, difúznej oblasti alebo na hranici objektu. Následne sa pre túto lokalitu vždy pevným spôsobom vypočítala farba povrchu.



Obrázok 14: Príklad programu na Cartoon shading

Tento príklad demonštruje jeden z implementovaných programov. Podobne ako tento sú vedené aj všetky ostatné príklady vytvorené v rámci tejto práce. Zdrojové kódy programov sú dostupné na priloženom DVD.

11 Testovanie

Testovanie použiteľnosti, prehľadnosti a intuitívnosti v používaní prebehne v niektorom ďalšom semestri študentmi Fakulty elektrotechnickej ČVUT, v rámci predmetov Grafické Systémy, alebo Základy počítačovej grafiky pod vedením Ing. Petra Felkla, PhD.

12 Záver

Cieľom diplomovej práce bol návrh a implementácia projektu umožňujúceho rýchlejšie a kvalitnejšie pochopenie technológie OpenGL. Tento zámer mal byť realizovaný za pomoci vytvorenia sady názorných a jednoduchých programov implementovaných v jazyku C++. Ako grafický základ programov bola stanovená technológia OpenGL vo verzii 2 a novšia. Programy mali za úlohu demonštrovať základnú, ale aj pokročilú funkcionalitu dostupnú vo fixnej aj programovateľnej pipeline. Implementované mali byť programy ako napr. vytvorenie okna, prenos geometrie, použitie svetiel, či zobrazovanie hmly. Ďalej boli k implementácii stanovené aj pokročilé techniky ako napr. bump či parallax mapping, rôzne druhy mapovania prostredia, alebo efekty typu cartoon shading.

V rámci práce sa podarilo stanoviť jednotný a robustný návrh pre všetky implementované aplikácie. Zaistila sa tak ich podobnosť v rámci projektu a tým aj rýchlejšie štúdium realizovaných zdrojových kódov. Podľa zadania mali byť všetky programy implementované vo verzii OpenGL 2 a vyššej. Avšak analýzou rozšírenosti a finančnej dostupnosti v kapitole 4.3 bol tento cieľ pozmenený a zameraný práve do OpenGL 2, rozšírený o fixnú pipeline OpenGL 1. Implementácie programov prebehli podľa zadania v jazyku C++ za asistencie podporných knižníc identifikovaných v kapitole 10. Realizovali sa programy demonštrujúce fixnú i programovateľnú pipeline. Navyše sa niektoré princípy naprogramovali v oboch pipeline pre možné porovnávanie. Programy tak obsahujú príklady základného typu ako vytvorenie okna, vytvorenie jednoduchého objektu a použitie materiálov a efekty ako cartoon rendering či per-pixel lighting. Zároveň boli zrealizované aj pokročilé techniky hlavne za pomoci shaderov. Medzi príklady patrí per-fragment hmla, paralax a bump mapping, hemisphere lighting a spherical harmonics, či depth of field. Podrobnejší zoznam je uvedený v kapitole 10.3. Ako vedľajší produkt tejto práce bola vytvorená knižnica vytvárajúca podporu pre základné operácie v príkladoch. Medzi implementovanú funkčnosť patrí hlavne jednoduchá kamera, manažér shaderov či načítavanie rôznych 3D modelov.

Pre realizáciu programov boli využité najmodernejšie nástroje, u ktorých sa predpokladá bezproblémová použiteľnosť i v budúcnosti. Architektúra aplikácií a použité frameworky sú schopné pracovať až s OpenGL verziou 4. Vďaka rozsiahlej analýze sa podarilo

nájsť robustné nástroje ako multimediálnu knižnicu SFML či importér grafických zostáv Assimp, ktorý umožňuje načítavanie 3D modelov v rôznych formátoch, vrátane najnovšieho formátu – Collada. Projekt je vďaka týmto použitým knižniciam a návrhu z kapitoly 9 jednoducho rozšíriteľný. Preto bude vhodné ho časom rozšíriť aj o príklady pre najnovšie OpenGL a uzavrieť tak kapitolu so zastaralými špecifikáciami. K dosiahnutiu tohto cieľa bude potrebné prepracovať knižnicu TGL tak, aby nepoužívala funkcie zakázané v profile CORE vybraného OpenGL. Jediným slabším článkom projektu je totiž knižnica samotná. Tá musela byť implementovaná v rámci požiadaviek OpenGL 2. Z tohto dôvodu ju nieje možné použiť pre vyššie verzie OpenGL v profile CORE.

13 Diskusia

V tejto kapitole sa zameriam na diskusiu určitých rozhodnutí vykonaných behom tejto práce. Bude sa jednať hlavne o vysvetlenie výberu knižnice SFML a niektorých ňou realizovaných riešení. Ďalej spomeniem aj okolnosti, ktoré ma viedli k implementácií vlastného model-loaderu pomocou knižnice Assmip, a to aj napriek množstvu dostupných hotových implemetácií. V závere predkladám myšlienky, ktoré ma viedli k výberu a použitiu zastaralej verzie OpenGL 1, teda fixnej pipeline. Zároveň s tým sú uvedené aj dôvody, ktoré diskvalifikovali verzie OpenGL 3 a 4 z výberu.

13.1 Výber knižnice SFML ako základu aplikácií

V prvopočiatkoch projektu sa javila ako vhodný základ aplikácie knižnica GLUT. Nijako mi neprekážala ani jej procedurálna povaha nakoľko som nevedel či nejaká objektová vôbec existuje. Zároveň som vedel s týmto okenným systémom dobre pracovať. Čoskoro som však zistil, že pre budúce potreby projektu je obmedzená nielen na kontext do verzie OpenGL 2. Začal som teda hľadať podobné alternatívy.

Ako prvú alternatívu som objavil knižnicu FreeGLUT. Má rovnakú stavbu a zameranie ako GLUT. Značne ho rozširuje, ale zachováva plnú spätnú kompatibilitu. Podporuje vytváranie kontextov v ľubovoľnej verzií OpenGL. Zároveň sa v ňom dá stanoviť používaný profil OpenGL a jeho ďalšou výhodou je, podobne ako u GLUTu, že umožnuje presne špecifikovať používané buffery. Preklad zo zdrojových kódov bol taktiež bezproblémový. Na jeho základe tak vznikla prvá verzia projektu. Pomerne rýchlo sa však ukázalo, že nakoľko je tento systém ideálny pre moju prácu je potrebné implementovať ešte mnoho podpornej funkcionality ako načítavanie obrázkov, vykresľovanie textu na obrazovku atd. Preto som začal hľadať framework podobnej kvality, ale s rozšírenou funkčnosťou.

Po niekoľkých bezvýznamných knižniciach som objavil GLFW – OpenGL Framework. Táto bola taktiež procedurálna. Navyše však mala výborne spracovanú dokumentáciu. V tej dobe bol ešte podporovaný iba základný kontext OpenGL. Verzie 3 a 4 mala implementovať najbližšia verzia. Preto táto skutočnosť nepredstavovala prekážku nakoľko bol vývoj aktívny a implementované príklady mali byť len do verzie OpenGL 2.0. Zároveň GLFW podporovala viacvláknové spracovanie a hlavne dôležité načítavanie obrázkov a textúr. Tie síce podporovala iba vo formáte TGA 1, to však stačilo pre potreby projektu. Po dlhšej a podrobnejšej analýze som však zistil, že ako podpora pre multithreading, tak aj načítavanie obrázkov je ďalej nepodporované a v budúcich verziách bude odstránené. Z tohto dôvodu a z dôvodu občasnej problémovej kompilácie som aj túto variantu zavrhol.

Ako ďalší logický krok mi po predchádzajúcich skúsenostiach vyvstalo vyhľadanie práve multimediálne zameranej knižnice SDL. Pri jej analyzovaní som objavil referenciu na alternatívu v podobe SFML. Táto knižnica predstavovala popri mnohých preskúmaných akúsi revolúciu. Bola plne objektovo orientovaná, ale podporovala aj procedurálny návrh. Mala vstavanú podporu pre ľubovoľný kontext OpenGL v COMPATIBILITY profile. Zároveň podporovala načítavanie niekoľkých druhov obrázkov vrátane JPG, PNG, TGA a dokonca DDS. Ďalej sa ukázalo, že podporuje ako vytváranie offscreen kontextov, tak aj viac okenných systémov. K tomu ďalej podporovala multithreading, prácu so zvukom a sieťou. Bola výborne dokumentovaná a na stránkach mala stručné a kvalitné tutoriály. Čo ma ale veľmi prekvapilo bolo samotné spracovanie zdrojových kódov. Tie sú stručne a výborne okomentované, objektovo orientované so zachovaním jednotného návrhu a používajú menné priestory pre lepšiu segmentáciu kódu. Po preštudovaní kódov môžem zodpovedne prehlásiť, že s takto kvalitne spracovaným projektom som sa ešte nestretol. Dôležité na SFML je, že celá poskytuje stručné a prepracované rozhranie. Vďaka tomu sa v knižnici ľahko orientuje a jej API je možné naštudovať za menej ako jeden deň. Stručné rozhranie nijako knižnicu neochudobňuje. Tá efektívne podporuje presne tie typy operácií, ktoré programátor po 90% svojho času potrebuje. SFML predstavuje API, ktoré za cenu miernych obmedzení programátora výborne smeruje k jeho cieľu.

Aj napriek vysokej strate času pri hľadaní a testovaní rôznych frameworkov som rád, že sa mi podarilo objaviť niečo tak dobré ako SFML. Hlavne pre začiatočníkov, ale aj pokročilých užívateľov bude znalosť tejto multimediálnej knižnice viesť k lepšej efektivite práce a možnosti lepšie sa zamerať na riešený problém, namiesto venovania väčšiny času príprave základu aplikácie a podporných knižníc pre textúry, zvuk, sieť tak ako to v praxi býva. Na záver ešte dodám, že s knižnicou SFML odpadlo množstvo závislostí, ktoré vyvstali nielen v programoch v tejto práci, ale ktoré vyvstanú v užívateľských aplikáciách a rôznych semestrálnych projektoch.

13.2 Assimp ako základ Model-loaderu

Za niekoľko rokov aktívneho používania OpenGL som nezaznamenal žiadny kvalitný a zároveň voľne dostupný model-loader. Po rozsiahlom a dôkladnom hľadaní som však natrafil na knižnicu Assimp. Nejedná sa priamo o model loader, ale o tzv. asset-importer, teda načítavač geometrických zostáv. Vzhľadom na skutočnosť, že nepodporuje žiadne grafické operácie nad načítaným modelom, bolo možné ho vytvoriť stabilnejší a prenositeľnejší. Zároveň sa tak autori mohli zamerať na lepšie pochopenie podporovaných 3D formátov a ich

kvalitnejšiu implementáciu. Knižnica disponuje vysokým výkonom a veľkým množstvom funkcií pre tzv. post-processing na načítanej geometrii. Ako príklad môžem uviesť napr. re-triangulácia geometrickej siete, výpočet normál, tangent, bitangent, či korekcia geometrickej siete ako i mnohé iné. Uvedené funkcie predstavujú neoceniteľné vlastnosti pre použitie v praxi, pretože drvivá väčšina dostupných modelov obsahuje množstvo chýb. Tie ani nemusia byť vytvorené neschopnosťou autora modelu. Totiž aj komerčné a astronomicky drahé aplikácie ako 3D Studio MAX, či Maya nedisponujú kvalitnými exportérmi.

Aj keď použitie Assimp vedie k nutnosti implementácie vlastného model-laoderu, jedná sa o efektívnejšie investovanie času, než do opravy niektorého z dostupných hotových produktov. Zároveň sa mi jej použitie osvedčilo ako bezproblémové. Za celý čas strávený nad implementáciou a neustálym používaním knižnice som nenarazil na žiadnu chybu, či nedeterministickosť. Použitím Assimp ako základu model-loaderu navyše získavam možnosť načítavania 3D informácie z približne 20 rôznych formátov.

13.3 Načítavanie obrázkov cez SFML

Nakoľko knižnica disponuje načítavaním obrázkov aj vo formátoch JPG, PNG a TGA nebolo žiadnym spôsobom potrebné vytvárať podporu pre túto funkcionalitu. Z uvedených sú podporované ako stratové tak i bezstratové kompresie. Navyše tento typ obrázkov je najbežnejšie použitý pre voľne dostupné textúry. Pri podpore uvedených formátov je ďalej umožnené aby si užívateľ mohol vytvárať takéto textúry v domácom prostredí za pomoci digitálneho fotoaparátu, alebo si ich generoval z akéhokoľvek dnešného softvéru synteticky. Súčasne každý konverzný softvér umožňuje prevádzanie do uvedených formátov. Týmto krokom sa celá práca stala stabilnejšou a jednoduchšou pre používateľa.

13.4 Diskvalifikácia formátu obrázkov DDS

Nakoľko SFML umožňuje načítavanie obrázkov aj v tomto formáte, nebol tento použitý z dôvodu jeho ťažkej dostupnosti. DDS je síce vysoko optimalizovaný formát pre ukladanie textúr. Dokonca podporuje aj pred-generovanie mip-máp, či optimalizované ukladanie normálových textúr. Bohužiaľ existuje len veľmi málo a k tomu zväčša iba komerčných programov, ktoré s týmto formátom dokážu pracovať v zmysle syntetického generovania obrazu, či konverzie a optimalizácie. Ani len programov na prosté prezeranie obrázkov v tomto formáte nieje mnoho. Z hľadiska použiteľnosti v mojich príkladoch, ktoré dbajú iba na prehľadnosť a nijako na výkon, tak jeho použitie postráda zmysel. Prednosť majú aj keď menej kvalitné ale zato široko rozšírené formáty.

13.5 Použitie OpenGL verzie 1 a 2

Aj napriek zadaniu práce, implementovať príklady v klasickom OpenGL, som uvažoval o ich vytvorení pre OpenGL 3 a 4 v profile CORE. Po miernom naštudovaní špecifikácie týchto verzií sa to však ukázalo ako kontraproduktívny krok. Pre OpenGL verziu 4 je totiž nutné zakúpiť najnovší hardvér. V špecifikácií totiž vyžaduje integráciu špeciálnych výpočtových hardvérových jednotiek. Kvôli tomu nebude ani v budúcnosti možné toto obmedzenie prekročiť inštaláciou nových driverov GPU. Zároveň tento hardvér nieje bežne dostupný ani v stolových počítačoch. Verzia OpenGL 3.3, ktorá bola vydaná až po špecifikácií 4.0, vytvára akúsi záplatu na tento nedostatok poskytovaním čo najväčšej hardvérovo nezávislej funkcionality z verzie 4 už vo verzii 3. V rámci školy, ako sektoru do ktorého sú moje programy smerované v súčasnej dobe neexituje ani stabilná podpora pre OpenGL 3.2. S tým padol konečný verdikt pre zamietnutie verzie 4.

Naďalej však ostala možnosť použitia verzie 3 v profile CORE. Zavedenie tohto profilu malo „vyčistiť“ OpenGL od nastavení rôznych stavov GPU a redundantnej funkcionality, ktorá sa nabaľovala časom na jadro OpenGL 1 pre zvýšenie jeho flexibilnosti a výkonu. Dôsledkom bolo prenesenie vykresľovania a správy stavu do shaderov (a teda na programátora) a zrušenie akejkoľvek fixnej podpornej funkcionality. Pozitívum tohto bolo neopísateľné sprehľadnenie správy stavu OpenGL. Už tak nedochádza k situáciám, kde použitie nekvalitnej knižnice nastaví iný stav OGL ako programátor pred jej použitím definoval, a tým dôjde k nesprávnemu spracovávaniu inak dobre napísaného kódu. Toto predstavovalo vždy jeden z najväčších nedostatkov tohto rozhrania. Na druhú stranu ale profil CORE vniesol veľkú zložitosť do malých aplikácií, akými sú aj tie, ktoré som implementoval. Je nutné opakovane implementovať kód, ktorý je v každej aplikácií rovnaký a ktorý fixná pipeline vykonávala prakticky bez vedomia užívateľa. Tým je napríklad nutné opakované nastavovanie a naväzovanie premenných predávajúcich informácie o súradniciach vrcholu, jeho normály, transformačných maticiach objektu, pohľadu, textúr. Ďalej je nutné predávanie všetkých informácií o polohe a smere svetla, jeho farebných zložkách, intenzity ako aj o rôznych vlastnostiach materiálov povrchu objektu. To sprehľadneniu kódu veľmi nepridalo a zároveň vnieslo do aplikácií ďalšiu závislosť v podobe nutnosti implementácie matematickej knižnice spolu s rôznymi štruktúrami uchovávajúcimi uvedené informácie. Taktiež nieje umožnené kreslenie pomocou immediate módu, čo sa často využívalo pri viac-priechodovom vykresľovaní, ktoré potrebovalo iba vykreslenie fullscreen quad objektu. Aj vykreslenie jediného vrcholu je odteraz nutné implementovať skrze rozhranie VBO.

Uvedené skutočnosti nepredstavujú praktické obmedzenie pre skúseného programátora OpenGL. Pre začiatočníka ale predstavujú generovanie veľkého množstva aplikačného kódu pre dosiahnutie malých výsledkov. Opísané hardvérové, ako aj softvérové nároky tak stanovili konečný verdikt pre použitie klasického OpenGL.

Vo verzii 2 je možné pracovať so shadermi, zároveň je ale dostupná funkcionalita fixnej pipeline v podobe transformačných matíc, či implicitného predávania vrcholov so základnými atribútmi do shaderov. Naučenie verzie OpenGL 1 (teda fixnej pipeline) je dnes stále dôležité z dvoch dôvodov. Za prvé fakt, že v súčasnosti rozšírené modely prenosných počítačov stále nepodporujú ani plný potenciál funkcionality OpenGL 2. Za druhé je možné použitie týchto príkazov v tzv. COMPATIBILITY profile. Ten je dostupný aj pre najnovší OpenGL 4.1 a veľký výrobcovia grafického hardvéru ako nVidia, či AMD sa vyjadrili o jeho dlhodobej podpore aj napriek striktným špecifikáciám OpenGL konzorcia.

Prílohy

A Zoznam použitej literatúry

[1] Řeháček D.: *3D Studio MAX 2: Uživatelská příručka*. Computer Press. Praha, 1998. 80-7226-104-5.

[2] Liberty J.: *Naučte se C++ za 21 dní*. Computer Press. Praha, 2002. 80-7226-774-4.

[3] Shreiner D., Woo M., Neider J., Davis T.: *OpenGL, Průvodce programátora*. Computer Press. Brno, 2006. 80-251-1275-6.

[4] Žára J., Beneš B., Sochor J., Felkel P.: *Moderní počítačová grafika*. Computer Press. Brno, 2004. 80-251-0454-0.

[5] Richard W. S. Jr., Haemel N., Sellers G., Lipchak B.: *OpenGL SuperBible: Comprehensive tutorial and reference*. Addison-Wesley. Upper Saddle River, NJ, 2010. 0-32-171261-7.

[6] The Khronos Group: *OpenGL 4.1 API Quick reference card*. 2010

[7] Rost R. J.: *OpenGL Shading Language*. Addison-Wesley. Upper Saddle River, NJ, 2004. 0-321-19789-5.

[8] Rost R. J., Licea-Kane B.: *OpenGL Shading Language*. Addison-Wesley. Upper Saddle River, NJ, 2009. 0-321-63763-1.

[9] Shreiner D.: *OpenGL Programming guide*. Addison-Wesley. Upper Saddle River, NJ, 2009. 0-321-55262-8.

[10] McReynolds T., Blythe D.: *Advanced graphics programming using OpenGL*. Elsevier. Amsterdam, 2005. 1-55860-659-9.

[11] The Khronos Group: *The OpenGL Graphics system: A specification Version 4.1 (Core profile)*. 2010

[12] The Khronos Group: *The OpenGL Shading Language: A specification Version 4.10*. 2010

[13] Rákos D., *RasterGrid: An introduction to OpenGL 4.1* [online]. 2010. http://rastergrid.com/blog/2010/08/an-introduction-to-opengl-4-1

[14] Clark R., *The Game Programming Wiki: Game Engines* [online]. 2004. http://gpwiki.org/index.php/Game_Engines

[15] Clark R., *The Game Programming Wiki: Libraries* [online]. 2004. http://gpwiki.org/index.php/Libraries

[16] McGuire M. a kol., *G3D Innovation Engine* [online]. http://g3d.sourceforge.net/

[17] Cloward B., *Extra texture: Texture archive* [online]. http://www.bencloward.com/textures_extra.shtml

[18] *Institute for creative technologies: High-Resolution Light Probe Image Gallery* [online]. 2008. http://gl.ict.usc.edu/Data/HighResProbes/

[19] Debevec P., *Diffuse and Specular Convolution* [online]. http://ict.debevec.org/~debevec/HDRShop/tutorial/tutorial6.html

[20] D' Angelo A., *Fusion Industries: HDR Stage, Relighting buildings with high dynamic range images* [online]. 2004. http://www.fusionindustries.com/default.asp?page=thesis

[21] Whorley H., *Hazel.h: Textures* [online]. http://www.hazelwhorley.com/skyboxtex3_bitmaps.html

[22] *3DHeaven: 3D Modelle* [online]. 2003. http://www.3dheaven.net/start.htm

[23] *Digital Dream Designs: 3D Model Source* [online]. http://www.digitaldreamdesigns.com/3DModels.htm

[24] *3D arts: Downloads* [online]. http://www.3drt.com/downloads.htm

[25] *Archive 3D* [online]. http://www.archive3d.net/

[26] Mutel A. a kol., *NShader - HLSL - GLSL - CG - Shader Syntax Highlighter: AddIn for Visual Studio* [online]. 2009. http://nshader.codeplex.com/

[27] *Sourceforge: The OpenGL Extension Wrangler Library* [online]. 2004. http://glew.sourceforge.net/

[28] *G-Truc Creation: OpenGL Matematics* [online]. 2005. http://glm.g-truc.net/

[29] *Assimp: Open Asset Import Library* [online]. 2008. http://assimp.sourceforge.net/index.html

[30] *SFML: Simple and Fast Multimedia Library* [online]. http://www.sfml-dev.org/

[31] Bateman R., *robtheblog.org* [online]. 2005. http://nccastaff.bournemouth.ac.uk/jmacey/RobTheBloke/www/

[32] Turek M., *CZ NeHe OpenGL: Download* [online]. 2002-2008. http://nehe.ceske-hry.cz/download.php

[33] *GameDev.net: FAQ - OpenGL* [online]. http://www.gamedev.net/community/forums/showfaq.asp?forum_id=25#extlibs

[34] Astle D., *Game Programming Tricks of the Trade: Moving Beyond OpenGL 1.1 for Windows* [online]. 2003. http://www.gamedev.net/reference/articles/article1929.asp

[35] *NeHe: Latest NeHe News* [online]. 2008-2010. http://nehe.gamedev.net/

[36] Ramey D.,Rose L.,Tyerman L., *File Formats: MTL material format (Lightwave, OBJ)* [online]. 1995. http://local.wasp.uwa.edu.au/~pbourke/dataformats/mtl/

[37] *FileFormat.info: Wavefront OBJ File Format Summary* [online]. http://www.fileformat.info/format/wavefrontobj/egff.htm

[38] *Object Files (.obj)* [online]. http://local.wasp.uwa.edu.au/~pbourke/dataformats/obj/

[39] *Collada: COLLADA - Digital Asset and FX Exchange Schema* [online]. 2009. https://collada.org/mediawiki/index.php/COLLADA-_Digital_Asset_and_FX_Exchange_Schema

[40] Guinot J., *oZone3D: Bump Mapping using GLSL* [online]. 2005. http://www.ozone3d.net/tutorials/bump_mapping.php

[41] *Relief Texture Mapping* [online]. http://www.inf.ufrgs.br/~oliveira/RTM.html

[42] Risser E., Shah M.A., Pattanaik S., *Interval Mapping: Quick Per-pixel Displacement Mapping* [online]. http://graphics.cs.ucf.edu/IntervalMapping/

[43] *Wikipedia.org: Bump mapping* [online]. 2003. http://en.wikipedia.org/wiki/Bump_mapping

[44] *Wikipedia.org: Normal mapping* [online]. 2003. http://en.wikipedia.org/wiki/Normal_mapping

[45] McGuire M., McGuire M., *Brown Technical Report: Steep Parallax Mapping* [online]. 2005. http://graphics.cs.brown.edu/games/SteepParallax/index.html

[46] *OpenGL: OpenGL Software Development Kit* [online]. 2007-2010. http://www.opengl.org/sdk/docs/

[47] Rideout P., *The Little Grasshopper* [online]. 2010. http://prideout.net/blog/?p=48

[48] *Swiftless Tutorials: OpenGL Drivers* [online]. 2010. http://www.swiftless.com/tutorials/misc/gldrivers.html

[49] *OpenGL: OpenGL & OpenGL Utility Specifications* [online]. http://www.opengl.org/documentation/specs/#

[50] *OpenCOLLADA* [online]. 2009. http://www.opencollada.org/

[51] *ShaderMap™ Pro / CL: Create Height and Normal Maps from Photos* [online]. 2007-2010. http://shadermap.com/

[52] *3d how i see it: Treasure chest - How to bake correctly normal and ambient occlusion maps* [online]. 2010.

http://3dhowiseeit.blogspot.com/2010/04/treasure-chest-how-to-bake-correctly.html

[53] *CMake* [online]. 2008. http://www.cmake.org/

[54] Lajzer B., Nottingham D.: *Combining Screen-Space Ambient Occlusion andCartoon Rendering on Graphics Hardware* [online]. Available: http://www.brettlajzer.com/pub/graphics/final/nprssao_final_presentation.pdf

[55] Marroquim R.,Maximo A.: *Introduction to GPU Programming with GLSL* [online]. CNR, COPPE – UFRJ. Available:

http://glsl-intro-shaders.googlecode.com/files/glsl-intro-survey.pdf

[56] Risser E., Shah M., Pattanaik S.: *Interval Mapping* [online]. University of Central Florida. Available: http://graphics.cs.ucf.edu/IntervalMapping/images/IntervalMapping.pdf

[57] Oliveira M., Bishop G., McAllister D.: *Relief Texture Mapping* [online]. University of North Carolina at Chapel Hill. Available:

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.43.4626&rep=rep1&type=pdf

[58] McGuire M., McGuire M.: *Steep Parallax Mapping* [online]. Brown University, Iron Lore Entertainment. Available:

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.126.8956&rep=rep1&type=pdf

[59] Villar J.R.: *OpenGL Shading Language Course:Chapter 4 – Advanced Shaders* [online]. TyphoonLabs. Available: http://www.opengl.org/sdk/docs/tutorials/TyphoonLabs/Chapter_4.pdf

[60] Filion D., McNaughton R.: *StarCraft II: Chapter 5: Effects & Techniques* [online]. Blizzard Entertaiment. Available: http://developer.amd.com/documentation/presentations/legacy/Chapter05-Filion-StarCraftII.pdf

[61] Gold M.I.: *Emboss Bump Mapping* [online]. NVIDIA Corporation. Available: http://nehe.gamedev.net/data/lessons/extras/lesson22/EmbossBumpMapping.ppt

[62] *2D/3D graphics and games programming using OpenGL, Direct3D, and XNA* [online]. 2005-2010. http://www.dhpoware.com/

[63] Gregor L.: *Jazyky pro programování shaderů* [online]. Bakalářská práce, Masarykova Univerzita, Fakulta Informatiky. Available:

http://is.muni.cz/th/99007/fi_b/Shaders-text.pdf?lang=en

B Zoznam použitých skratiek

2D	- Dvoj-dimenzionálny
3D	- Troj-dimenzionálny
3Dc	- kompresný algoritmus pre normálové mapy založený na DXT5
3DS	- 3D Studio file
ANSI	- American National Standards Institute
AO	- Ambient Occlusion
API	- Application Programming Interface, aplikačné programové rozhranie
ARB	- Architecture Review Board, spoločnosti podieľajúce sa na vývoji OpenGL
Assimp	- Open Asset Import Library
BMP	- Bitmap, rastrový formát obrázkov
C	- Procedurálny programovací jazyk
C#	- Čisto objektová varianta jazyka C++ s podporou NET Framework
C++	- Objektové a funkčné rozšírenie jazyka C
CG	- C for Graphics, jazyk pre programovanie shaderov od firmy nVidia
CML	- Configurable Math Library
CPU	- Central Processing Unit, hlavný procesor počítača
CSS	- Cascading Style Sheets
ČVUT	- České Vysoké Učení Technické v Praze
D3D	- Direct3D
DDS	- Direct Draw Surface, formát grafických textúr
DevIL	- Developer's Image Library
Direct3D	- Konkurenčný systém pre OpenGL, súčasť multimediálnej knižnice DirectX
DirectX	- Multimediálna knižnica firmy Microsoft
DL	- Display List
DOS	- Disk Operating System
DXT	- S3 Texture Compression (S3TC), nazývaná aj DXTn, alebo DXTC
DXTC	- DirectX Texture Compression
FBO	- Frame Buffer Object
FPS	- First Person Shooter
FS	- Fragment shader
FSAA	- Full Screen Anti-Aliasing
GCC	- GNU C Compiler, unixový prekladač jazyka C/C++
GL	- OpenGL

GLEE - OpenGL Easy Extension library
GLEW - OpenGL Extension Wrangler library
GLFW - OpenGL Framework
GLM - OpenGL Mathematics, alebo model-loader OBJ formát
GLOW - *nezistené*
GLSL - GL Shading Language
GLT - OpenGL C++ Toolkit
GLU - GL Utility library
GLUT - OpenGL Utility Toolkit
GMTL - Graphics Math Template Library
GPU - Graphics Processing Unit, hlavný procesor grafickej karty
GS - Geometry shader
GTL - The Game Texture Loader
GUI - Graphical User Interface
HDR - High Dynamic Range, obrázky s viac ako 8 bitmi na kanál
HLSL - High Level Shading Language
JOGL - Java for OpenGL, framework pre sprístupnenie OpenGL v jazyku Java
JPG - Joint Photographic Experts Group, rastrový formát obrázkov
Lat-long - Lattitude Logitude, typ environmentálnej textúry
LDR - Low Dynamic Range, klasické 32 bitové obrázky, max. 8 bitov na kanál
LIB - Library, statická knižnica
MinGW - Minimalist GNU for Windows
MSVC - Microsoft Visual Studio
NPR - Non-photorealistic rendering
OBJ - Wavefront Object file
OGL - OpenGL
OGLWFW - OpenGL Window Framework
OGRE - Object-Oriented Graphics Rendering Engine
OpenGL - Open Graphics Library
OS - Operating System
OSG - Open Scene Graph
PBO - Pixel Buffer Object
PCX - Personal Computer eXchange, rastrový formát obrázkov
PNG - Portable Network Graphics, formát obrázkov
PPM - Portable Pixel Map, formát obrázkov
PSD - PhotoShop Document, vektorový formát obrázkov

RGB	- Red Green Blue, farebný priestor/kanály
RGBA	- Reg Green Blue Alpha, farebný priestor RGB rozšírený o alfa kanál
RT	- Real Time, spracovanie v reálnom čase
RTT	- Render To Texture
SDL	- Simple Direct media Layer
SFML	- Simple and Fast Multimedia Library
SOIL	- Simple Open Image Library
SSAO	- Screen Space Ambient Occlucion
SSDO	- Screen Space Directional Occlucion
STL	- Standard Template Library, knižnica štandardných rozšírení jazyka C++
TCS	- Tesselation Control Shader
TES	- Tesselation Evaluation Shader
TGA	- Targa image format, rastrový formát obrázkov
TGL	- Taygeta Graphics Library
TIFF	- Tagged Image File Format, rastrový formát obrázkov
VBO	- Vertex Buffer Object
VRML	- Virtual Reality Modeling Language
VS	- Vertex shader
WebGL	- Javascriptový jazyk s priamou podporou OpenGL ES 2
XML	- eXtensible Markup Language
XNA	- Xbox/DirectX New generation Architecture

C Inštalácia knižníc

Všetky inštalácie sú popísané pre vývojové prostredie Microsoft Visual Studio 2008 a všetky postupy popisujú statické linkovanie – teda vytvorenie LIB súborov z uvedených knižníc. Pre správne fungovanie všetkých programov je nutné nastaviť v závislostiach linkeru cestu k všetkým týmto prekompilovaným knižniciam.

Navyše sa v rámci práce zachováva konvencia označovania názvov vytvorených knižníc pridávaním prípony za názov nasledovne:

- statické knižnice: *-s*
- dynamické knižnice: *bez prípony*
- *debug* knižnice: -d
- *release* knižnice: *bez prípony*

V prípade potreby sú tieto prípony kombinované. Napr. pre statickú knižnicu s *debug* symbolmi je výsledná prípona *-s-d*. Pre dynamickú *release* knižnicu je názov uvádzaný bez prípon.

GLEW

1. Stiahnite zdrojové kódy knižnice z *http://glew.sourceforge.net*, sú dostupné v komprimovanom archíve ZIP.
2. Rozbaľte zdrojové kódy do adresára s ľubovolným názvom. Ďalej bude jeho názov označovaný ako *GLEWDIR*.
3. Otvorte v MSVC sollution súbor *glew.dsw*. Nájdete ho v *GLEWDIR/build/vc6/glew.dsw*. Pre MSVC novšie ako verzia 6 sa objaví otázka na konvertovanie projektu. Zvoľte odpoveď *YES TO ALL*.
4. Z menu MSVC zvoľte: *Build → Batch build*.
5. V okne *Batch build* vyberte všetky položky, ktoré v stĺpci *Project* obsahujú *glew_static* a stlačte tlačidlo *REBUILD*.
6. Z adresára *GLEWDIR/lib* prekopírujte všetky LIB súbory do adresára, kde ich chcete mať uložené pre ďalšie použitie. Napríklad: *cesta/k/projektu/libs/*
7. Z adresára *GLEWDIR/include/GL* prekopírujte všetky súbory do adresára s hlavičkovými súbormi ostatných externých knižníc, používaných v projekte. Napríklad: *cesta/k/projektu/include/glew*

SFML

1. Stiahnite *CMake* z *http://www.cmake.org/cmake/resources/software.html* pre platformu Windows v archíve ZIP a rozbaľte ju.
2. Stiahnite zdrojové kódy knižnice SFML 2.0 dev snapshot z *http://www.sfml-dev.org/download.php*
3. Rozbaľte zdrojové kódy knižnice SFML do adresára s ľubovolnými názvom. Ďalej bude jeho názov označovaný ako *SFMLDIR*.
4. Spustite *CMake*.
5. Do pola *Where is the source code* zadajte cestu k hlavnému adresáru *SFMLDIR*.
6. Do pola *Where to build the binaries* zadajte cestu kam sa bude ukladať výstup, teda vygenerovaný projekt MSVC 2008. Napríklad: *SFMLDIR/build*
7. Po nastavení ciest stlačte tlačidlo *Configure* a zvoľte typ projektu *MSVC 2008*. Následne program *CMake* skontroluje existenciu MSVC 2008 v systéme a iné závislosti nutné pre vytvorenie *MSVC 2008 solution*. Po skončení sa zobrazia v hlavnom okne nastavenia. Pokiaľ nie sú potvrdené nastavenie ďalším stlačením tlačidla *Configure*, sú tieto zvýraznené na červeno. Zaujímajú nás len niektoré položky, ktoré je nutné nastaviť.
8. Položku *BUILD_SHARED_LIBS* odškrtnite pre vytvorenie statických knižníc
9. V položke *CMAKE_CXXFLAGS_DEBUG* zmente defalutnú hodnotu *Zi* na *Z7* kvôli integrovaniu Debug-symbolov priamo do LIB knižníc. Inak ich MSVC v užívateľských projektoch nevie nájsť a hlási upozornenia.
10. Po nastavení týchto dvoch hodnôt stlačte opäť tlačidlo *Configure* a následne tlačidlo *Generate*. Tým sa vytvorí v adresári *SFMLDIR/build* nakonfigurovaný *MSVC solution* pripravený na použitie.
11. Otvorte v MSVC tento nakofigurovaný *MSVC solution*.
12. Z menu MSVC zvoľte: *Build → Batch build*.
13. Označte všetky *Debug* a *Release* konfigurácie, kde sa v stĺpci *Project* vyskytuje *sfml-xxx*. *xxx* je *audio*, *graphics*, *main*, *network*, *system* a *window*. Po označení všetkých stlačte *REBUILD*.
14. Po skončení sa v adresári *SFMLDIR/build/lib* objaví *Debug* a *Release* variant knižnice SFML. Súhrnne prekopírujte všetky LIB súbory do adresára, kde budete mať uložené knižnice. Napríklad: *cesta/k/projektu/libs/*
15. Nakoniec prekopírujte všetky súbory z *SFMLDIR/include/SFML* do adresára s hlavičkovými súbormi ostatných externých knižníc. Napríklad: *cesta/k/projektu/include/SFML*
16. Pri použití statických knižníc (majú príponu *-s*) najnovšej verzie SFML je v projekte nutné preprocesoru nastaviť premennú *SFML_STATIC*.

GLM

1. Stiahnite zdrojové kódy v archíve ZIP z *http://glm.g-truc.net*
2. Rozbaľte zdrojové kódy do adresára s ľubovolným názvom. Ďalej bude jeho názov označovaný ako *GLMDIR*.
3. Prekopírujte všetky súbory z *GLMDIR/glm* do adresára s hlavičkovými súbormi ostatných externých knižníc. Napríklad: *cesta/k/projektu/include/glm*

Assimp

Assimp podporuje kompiláciu s a bez knižnice Boost. Tento postup sa venuje druhej z uvedených možností.

1. Stiahnite zdrojové kódy knižnice Assimp zo stránok *http://assimp.sourceforge.net/main_downloads.html*
2. Rozbaľte zdrojové kódy do adresára s ľubovoľným názvom. Ďalej bude jeho názov označovaný ako *ASSIMPDIR*.
3. Otvorte v MSVC sollution súbor s názvom *assimp.sln*. Nájdete ho v *<ASSIMPDIR>/workspaces/vc9/assimp.sln*.
4. Z menu zvoľte: *Build → Batch build*.
5. V okne *Batch build* vyberte všetky položky, ktoré v stĺpci *Project* obsahujú *assimp,* zároveň v stĺpci *Configuration* obsahujú *debug-noboost-st* a *release-noboost-st* a v stĺpci *Platform* Vami preferovanú platformu. Po tomto kroku by ste mali mať označené práve dve položky. Prvú pre Debug a druhú pre Release konfiguráciu. Po označení stlačte *REBUILD*.
6. Z adresárov *<ASSIMPDIR>/lib/assimp_[debug|release]_noboost-st_[platforma]* prekopírujte všetky LIB súbory do adresára s knižnicami. Napr: *cesta/ku/knižniciam/libs*
7. Z *<ASSIMPDIR>/include* prekopírujte všetky súbory do adresára, kde chcete mať uložené hlavičkové súbory. Napr: *cesta/ku/knižniciam/include/assimp*

D Obsah priloženého DVD

/text

- text práce v rôznych formátoch s priloženými ilustráciami

/specifications

- špecifikácie OpenGL a GLSL

/resources

- vybrané zdroje/programy/knižnice použité pri tvorbe práce

/resources/plugins

- pluginy ako zvýrazňovač syntaxe GLSL do MSVC, či exportér Collady do 3DS MAX

/resources/utils

- pomocné programy

/resources/libs

- zdrojové kódy použitých knižníc

/binaries

- spustiteľné súbory príkladov

/binaries/models

- modely použité v príkladoch

/binaries/textures

- textúry použité v príkladoch

/taygeta

- zdrojové kódy príkladov

/taygeta/.project/include

- hlavičkové súbory použitých knižníc na asociáciu s projektami

/taygeta/.project/include/tgl1

- hlavičkové súbory a zdrojové kódy knižnice TGL

/taygeta/.project/binaries

- adresár pre výstup preložených zdrojových kódov

/taygeta/.project/msvc2008

- projektový adresár MSVC 2008 s projektovými súbormi a použitými knižnicami

/taygeta/.project/msvc2008/libs

- statické LIB knižnice preložené pre MSVC 2008, používané v projekte

/taygeta/.project/msvc2008/properties

- súbory nastavení kompilátoru a linkeru pre projekty MSVC 2003 a viac