Na tomto místě bude oficiální zadání vaší práce

- Toto zadání je podepsané děkanem a vedoucím katedry,
- musíte si ho vyzvednout na studiijním oddělení Katedry počítačů na Karlově náměstí,
- v jedné odevzdané práci bude originál tohoto zadání (originál zůstává po obhajobě na katedře),
- ve druhé bude na stejném místě neověřená kopie tohoto dokumentu (tato se vám vrátí po obhajobě).

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Bakalářská práce

Genesis - jádro herního enginu

Michal Červenka

Vedoucí práce: Ing. Jiří Bittner Ph.D.

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Web a multimedia

12. června 2009

Poděkování

Chtěl bych poděkovat Dagmar Tkadlecové, Aleně Bendové, Kateřině Štemberové, Michaele Zbytovské, Ladislavu Hrbáčkovi, Aleši Zavadskému a dalším, kteří se jakkoliv podíleli na bakalářském projektu Genesis.

Upřímné díky patří také Ing. Jiřímu Bittnerovi Ph.D. za vedení mé práce, Prof. Ing. Jiřímu Žárovi CSc. a Ing. Romanu Berkovi za podporu a směrování při práci i za cenné rady, Ing. Zdeňku Trávníčkovi za pomoc při organizaci a firmě A|W Graph, která našemu týmu zajistila legální software.

Velmi specielní poděkování patří Evě Pechové za korekturu této práce.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 1. 6. 2009 .....

Abstract

Behind the wonderful graphics, animations and storytelling of today's games, there lies the powerful game engine, which is the key to the ultimate game experience. The game engine, which can be described as an optimized graphic application which is able to change the input data into operational scenes, is the subject of this work.

In the first part, the synopsis of the game technologies development in recent years is discussed; the profesional ones will be studied and described minutely. Second chapter is dedicated to selected graphic effects which are going to be implemented in the final product. The thesis specification was extended with a part treating the subject of team leading and game design. The rest of the work is dedicated to design and implementation of the respective game engine and related applications.

The work doesn't treat the details of the implementation process, but concentrates on the main thoughts and problems encountered during the work process. Besides the resulting application, it is possible to find, on the cd enclosed, even the prototypes that originated during the process of implementation and are used to test selected parts of the final application.

Abstrakt

To, co za dnešními hrami stojí, není jen úžasná grafika, příběh a animace, ale i výkonné herní jádro, bez kterého by si to hráč nevychutnal. Herní jádro je optimalizovaná grafická aplikace, která dokáže vstupní data proměnit ve funkční scény, a právě jeho implementací se zabývám ve své práci.

V první části práce shrnu vývoj herních technologií za posledních několik let a uvedu sepsané vlastnosti některých profesionálních technologií. Druhá kapitola bude věnována vybraným grafickým efektům, které mám v plánu ve výsledné aplikaci implementovat. Oproti zadání jsem přidal do práce část věnující se řízení týmu studentů a návrhu hry, pro kterou je technologie připravována. Zbytek práce je věnován návrhu a realizaci herního jádra a doprovodných aplikací.

Práce nezabíhá do detailů implementace, ale popisuje myšlenky a hlavní problémy, se kterými jsem se při psaní aplikace setkal. Na přiloženém CD si lze prohlédnout kromě výsledné aplikace i prototypy, které vznikly v průběhu příprav této práce a testují vybrané části finální implementace.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Počítačové hry, respektive počítačová grafika, jsou poměrně mladý obor, avšak během posledního desetiletí jsou také jedním z nejrychleji se vyvíjejících počítačovědních disciplín, a nejen to, jsou hlavní hnací kolo i pro další odvětví, primárně pro vývoj stále novějšího hardwaru, především grafických karet. Čím více se ale úroveň her zvyšuje, tím jsou stále dražší, dokonce v posledních letech náklady vybraných titulů předběhly i filmový průmysl. Před patnácti lety na jedné hře pracovaly maximálně desítky lidí, dnes jsou tyto počty v řádu stovek.

Česká republika má na poli herní scény vybudované poměrně silné postavení díky firmám jako je 2K Czech[1][14], Bohemia Interactive[2][10], Mindware Studios[3][18] a dalších. V nemalém zastoupení je u nás i celá řada firem zaměřujících se na mobilní, webové a jednoduché hry pro příležitostné hráče (Casual games).

Tato práce je součástí komplexního bakalářského projektu Genesis, který si vzal za úkol maximálně připravit tým studentů do praxe v oboru počítačových her a 3D grafiky a naučit je pracovat v koordinovaném týmu. Vzhledem k rychle se zvyšující kvalitě a požadavkům v dnešní hře, je pro herní firmy stále větší problém sehnat kvalifikované zaměstnance, od grafiků, přes programátory až po game designéry a projektové manažery. Na realizaci se pracovalo od září roku 2008 a za tu dobu všichni zúčastnění rozšířili své znalosti v tomto oboru. Na projektu se dále bude pracovat i po odevzdání bakalářských prací, podle plánů až do září 2009, kdy by roční práce měla být zakončena technickým demem[4] 3D Fantasy RPG hry odehrávající se v paralelním světě plném magie a tajemství. Zkušenosti a demo budou výbornou základní referencí pro všechny zúčastněné, aby po ukončení studia byli připraveni nastoupit do libovolného tuzemského, nebo i zahraničního herního studia. Zkušenosti jsou v tomto oboru neocenitelné. ČVUT FEL, Institut intermédií a sponzor projektu, firma A|W Graph[16] nám umožnili přístup nejen k profesionálnímu softwaru, ale i k technologiím, ke kterým se normální člověk nedostane a které jsou důležité pro tvorbu kvalitních AAA titulů.[5]

[1] 2K Czech je nový název pro u nás asi nejznámější Illusion Softworks, které v minulém roce koupilo Take 2[15], autoři připravované Mafia 2 a vydaných her Mafia, Hidden & Dangerous, Hidden & Dangerous 2, Vietcong, Vietcong 2

[2] Operation Flashpoint, Arma, Arma II

[3] Cold War, Painkiller Overdose

[4] Několikaminutová hratelná prezentace hry ukazující technické a umělecké dovednosti týmu.

[5] AAA, neboli "Velké hry", zpravidla obsahují video, zvuk, psaný text, interactivitu. V dnešní době jsou třídy AAA ve srovnání s filmy velmi finančně nákladné.

V České republice existují desítky nezávislých týmů. To, v čem se tým projektu Genesis odlišuje, jsou právě ony technologie, ke kterým se nám povedlo získat přístup, přednostně tedy optický systém Vicon[6][20], díky kterému jsme dokázali nasnímat pohyby reálných postav, které by se jinak musely ručně dlouze a složitě animovat.

V této práci jako autor a projektový manažer bakalářského projektu Genesis rozeberu kromě technologie, na které hra běží, i řízení týmu, pracovní plán, vývoj a celkový koncept hry. Zbytek práce bude o doprovodných nástrojích a technickém popisu herního jádra (Game engine). Rozsah této práce by dokázal pokrýt diplomovou závěrečnou práci, proto některé části jsou zkrácené a v textu se často odkazuji na již existující literaturu, které v posledních letech začíná být velké množství. I přes to ale stále existují části herního vývoje, které zdokumentované nejsou. Naše technické demo bude řazeno mezi tzv. filmové hry[7] a příběh bude vyprávět pomocí In-Game CUT sekvencí[8] a FMV sekvencí[9], které jsou pro tento typ žánru příznačné. Jedním z příkladů absence literatury jsou například metody visuálních optimalizací, které používá skriptér pro maximální rychlost, výkon a visuální dojem v těchto sekvencích. Bakalářské práce v projektu jsou naštěstí pokryty tak, aby výsledkem některých prací byly dokumenty, které pokryjí právě tuto scházející literaturu a posbírali důležité informace do jedné práce.

[6] Optický systém pro zachycení pohybu od společnosti Vicon, který zakoupil Institut Intermédií na ČVUT FEL.

[7] Primárním cílem filmových her je příběh a jeho prezentace. Standardními nástroji jsou In-Game CUT sekvence a FMV sekvence.

[8] Předskriptované sekvence, běžící v reálném čase na herním enginu. Zpravidla se odlišují od herních dialogů skriptovanou kamerou a složitějším děním na scéně.

[9] Full Motion Video sekvence - předrenderované videosekvence v přijatelném rozlišení.

Kapitola 2

Historie a vývoj herních enginů

Jedním z důležitých rozhodnutí při návrhu hry je, zda se pro hru napíše nová technologie, nebo zda se bude licencovat nějaká již existující. Již od počátku 3D her se technologie psaly tak, aby se mohly použít několikrát, jedním ze starších příkladů a asi nejznámějším je id Tech 1[1], který je znám pod názvem Doom Engine. I když jádro vzniklo v roce 1993, hry na této technologii se dělaly až do roku 1997 a vzniklo jich více jak 25 a má ji licencovanou více jak 12 firem. V dnešní době se opět začíná iD Tech 1 technologie používat, a to na mobilních telefonech, které již mají dostatečný výkon, aby mohly zobrazovat 3D hry – některé i již dokonce s hardwarovou akcelerací. Herní enginy se v dřívější době dělily na dva základní typy – optimalizované pro outdoor[2] a indoor[3]. V dnešní době velká řada herních enginů je psána univerzálně, aby hra mohla kvůli výkonu optimálně vykreslovat interiéry i exteriéry. Zásadní rozdíl je v použité optimalizační technologii pro vykreslování scény a v dnešní době i pro osvětlení a renderování stínů[17]. Indoorové enginy využívaly primárně binární dělení prostoru (BSP)[4] a portálovou technologii[4] a outdoorové byly založeny na oktanových stromech (Octree, Quadtree)[4]. Příklad enginu založeném na BSP je například již zmíněný iD Tech 1 a portálovou technologii využívá například Tomb Raider engine od firmy Core Design z roku 1996.

Dnes je primární rozdíl mezi outdoorem a indoorem nejen v dělení scény, zpravidla outdoorové technologie v kombinaci s Occlusion Cullingem[4][19] postačují i na indoorové scény, ale hlavně v použitém typu osvětlení a referování stínů. Příkladem může být následující situace: Máme hru, která umožňuje jezdit po městě s obrovskou rozlohou autem (příklad indooru), ale můžeme s autem také zajet do podzemních garáží (příklad outdooru). Když jsme venku, je renderer[5] nastaven do režimu outdoor, ve scéně je jedno globální světlo, které se používá k vrhání stínů metodou stínových map[6]. Jinou variantou je venku vůbec nepočítat stíny a vše mít předpočítané technologií lightmapping[7]. V okamžik, když s autíčkem zajedeme

[1] 1993, iD Software, Inc. [8]

[2] Hry odehrávající se převážně v exteriérech

[3] Hry odehrávající se převážně v interiérech

[4] Algoritmus pro vyřazení neviditelných ploch z renderovací pipeline.

[5] Část herního enginu, která se stará o vykreslování a grafické efekty. Zpravidla je to rozhraní mezi grafickou knihovnou a samotným enginem.

[6] Metoda založená na offscreen renderingu z pohledu světla, jejíž výsledek se použije pro výpočet zastínění jednotlivých fragmentů.

[7] Při exportu scény z editoru se vypočítá pro každý polygon tzv. textura světla, která určuje zastínění

do podzemních garáží, renderer se přepne do režimu indoor, kde nepoužíváme globální světlo, ve scéně může být N světel a každé světlo vrhá stín například metodou Shadow volume[8][19], aby se dosáhlo ostrých stínů, které jsou v místnostech bez denního světla v pořádku.

Poměrně velký vývoj také zaznamenaly modely ve scéně, které v roce 1992 byly pouze na úrovni 2D Spritů[9]. Kolem roku 1994 se začaly ve hrách objevovat první 3D modely, které začaly postupně nahrazovat 2D Sprity, hlavní důvod, proč se ale 2D Sprity používaly ve hrách pro charaktery, jsou animace. Kolem roku 1993, kdy ještě hry nebyly akcelerované grafickou kartou, nebyl výkon a paměť pro uchování potřebných informací pro vykreslení a animaci modelu a 2D Sprite, kde se dala pouze prohazovat textura byl ideálním řešením. V roce 1994 se začaly objevovat první 3D modely, například ve vesmírné střílečce Star Wars: TIE Fighter od týmu Totally Games[7] a vydavatele LucasArtsLUCAS. Stále ale převládaly sprity. Animované sprity jako herní charaktery a NPC[10] začaly nahrazovat modely s animacemi až na přelomu roku 1996, kdy se ve hrách začala poprvé používat obdoba skeletálních animací[11][21] v kombinaci s několikanásobně náročnější per-vertex animací[12]. Jedna z prvních her, která pro animované modely začala používat technologii skinning[13], tj. animuje se kostra a vrcholy přebírají transformace od kostry, byl v roce 1996 Crash Bandicoot od týmu Naughty Dog[5] pro konzoli Sony Playstation.

S počátkem používání 3D modelů se do her začaly implementovat i úrovně detailů LOD (Level Of Detail)[13], tj. modely měly pro každou vzdálenost od pozorovatele jinou geometrii, kterou mezi sebou měnily. Metody výměny mezi jednotlivým LOD modely také prošly menším vývojem. Z počátku se používalo skokové přehazování a následně se začaly používat přechody pomocí průhlednosti.

Optimalizační technologie ale není jediné, co se v herních enginech vyvíjí. Značného vývoje dosahují i grafické efekty, které dodávají scéně na realističnosti. Velmi důležitý vývoj, z pohledu grafických detailů a snížení počtu polygonů při zachování maximálních detailů na modelu je bezesporu transformace staré technologie bump mappingu[14] do normal mappingu[15] a do dnešního parallax mappingu[16]. Všechny tři uvedené technologie mají za cíl pomocí dodatečných informací v textuře posouvat a zabarvovat (podle toho, zda implementaci využívají technologie stínění (self-shadowing)) texturové souřadnice na renderovaných fragmentech, čímž lze docílit efektu, že v každém bodě se láme světlo podle textury, díky čemuž se může

každého texelu. Tato technologie se zpravidla používá pro statické zdroje světla.

[8]Technologie stínových těles vytváří dynamicky pomocnou geometrii, kde podle průniku s fragmenty se počítá, zda je zastíněn, nebo ne.

[9]Polygon, který je vždy orientovaný ke kameře, tedy působí jako 2D

[10]Non-player character - herní postava ovládána umělou inteligencí hry.

[11]Z počátku byl model rozdělen na jednotlivé části, které měly hierarchickou strukturu, stejně jako dnešní kostry a každá node tohoto stromu si ukládala pro klíčový snímek svou transformaci, které se mezi snímky interpolovali. V dnešní době je skeletální animací myšleno použití kostry, která je za pomoci technologie skinning přes váhy svázána s jednotlivými vrcholy modelu. To umožňuje, aby model byl jeden objekt a nemusel být rozdělen na malé části - tj. lepší visuální dojem.

[12]Pro každý klíčový snímek je uloženo rozložení všech vrcholů a v čase se mezi snímky interpoluje.

[13]Metoda výpočtu pozice vrcholu podle animace kostry za pomoci tabulky vah. Sérii výborných prací o Skinningu napsal například Mgr. Ladislav Kavan[11] z Trinity College v Dublinu.

[14]Technika využívající výškové mapy k posunu texturových koordinátů

[15]Metoda je založená na bump mapování, ale využívající texture, kde v každém texelu je uložena informace o směru normály v jeho bodě.

[16]Technologie rozšiřující normal mapping o informaci o délce posunu ve směru normály za pomoci výškové mapy

použít i model s relativně nízkým počtem polygonů a bude působit jako model s vysokým počtem polygonů. Tato technologie rapidně při vysoké úrovni snižuje počet polygonů ve scéně a v dnešní době se využívá i ve filmovém průmyslu, kde není nutné pracovat se scénou složenou z mnoha milionů polygonů, což poměrně zpomalovalo vývoj.

Dalšími efekty, kterými jsou obohaceny díky modernímu hardwaru dnešní hry, jsou například HDR, rozmazání pohybu, hloubka ostrosti, volumetrické osvětlení, volumetrická mlha, dynamické měkké stíny, ambient occlusion, subsurface scattering a další. Více o některých efektech je sepsáno v následující kapitole.

Vývoj ukáži tabulkově na několika vybraných, chronologicky seřazených, herních enginech. V tabulkách 2.1 až 2.8 si můžeme všimnout již zmíněného vývoje z čistě indoor, nebo outdoor enginů do obecných. Většina firem, která primárně používá OpenGL přešla s příchodem Shaderů na DirectX. Ale poslední enginy se zase kvůli Sony PlayStation 3 začaly k OpenGL vracet, přesněji k OpenGL ES 2.0. V tabulkách je možné zaznamenat i postupný zánik indoorového algoritmu BSP, který se primárně používal na výpočet kolizí, osvětlení a viditelnosti. Dnes, pokud se použije, tak již pouze na kolize, osvětlení a viditelnost a jsou počítány dynamicky. Algoritmy viditelnosti se zabývá například vedoucí mé práce, Ing. Jiří Bittner Ph.D.[2]

Název	iD Tech 1
Výrobce	iD Software
Rok vydání	1993
Licence	GNU General Public Licence (od roku 1999)
Cena	zdarma
Podporované 3D API	Software
Primární určení	Indoor
Optimalizační technologie	BSP
Animační technologie	Animované sprity
Grafické efekty	Sprity, Animované sprity, Animované textury
Skriptování	Triger systém
Hry	Doom (1993), Doom II (1994), Heretic (1994), D!Zone 2 (1995) Hexen: Beyond Heretic (1996), Towers of Darkness (1997)...

Tabulka 2.1: Tabulka vlastností enginu iD Tech 1

Název	Tomb Raider engine
Výrobce	Core Design
Rok vydání	1996
Licence	?
Cena	neznámá
Podporované 3D API	Software, 3Dfx's Glide
Primární určení	Indoor
Optimalizační technologie	Portals
Animační technologie	Skeletální animace
Grafické efekty	Sprity, Animované sprity, Animované textury, FMV Sekvence, CUT Sekvence
Skriptování	vlastní scriptovací jazyk
Hry	Tomb Raider (1996)

Tabulka 2.2: Tabulka vlastností Tomb Raider enginu

Název	iD Tech 2
Výrobce	iD Software
Rok vydání	1997
Licence	GNU General Public Licence (od roku 2001)
Cena	zdarma
Podporované 3D API	Software, OpenGL
Primární určení	Indoor
Optimalizační technologie	BSP
Animační technologie	Per-vertex animace
Grafické efekty	Sprity, Animované sprity, Animované textury, Skinovatelné modely, Light mapping
Skriptování	Triger systém, Texture shadery, SDK
Hry	Quake II (1997), Heretic II (1998), Kingpin: Life of Crime (1999), Solider of Fortune (2000), John Romero's Daikatana (2000) Anachronox (2001), ...

Tabulka 2.3: Tabulka vlastností enginu iD Tech 2

Název	iD Tech 3
Výrobce	iD Software
Rok vydání	1999
Licence	GNU General Public Licence (od roku 2005)
Cena	zdarma
Podporované 3D API	OpenGL
Primární určení	Indoor
Optimalizační technologie	BSP
Animační technologie	Per-vertex animace
Grafické efekty	Sprity, Animované sprity, Animované textury, Skinovatelné modely, Light mapping, Dynamická světla, Stíny metodou stínových těles, Částicový systém
Skriptování	Triger systém, Texture shadery, SDK
Hry	Quake III Arena (1999), Star Trek: Voyager - Elite Force (2000), Heavy Metal: F.A.K.K. 2 (2000), 007 Agent Under Fire (2001),...

Tabulka 2.4: Tabulka vlastností enginu iD Tech 3

Název	iD Tech 4
Výrobce	iD Software
Rok vydání	2004
Licence	Plná práva má pouze výrobce a vydavatel
Cena	$250.000 + 5% z každého prodaného titulu
Podporované 3D API	DirectX 9
Primární určení	Indoor
Optimalizační technologie	BSP
Animační technologie	Skeletalní animace + skinning
Grafické efekty	Sprity, Animované sprity, Animované textury,
	Skinovatelné modely, Light mapping, Phongovo stínování, Dynamická světla, Stíny metodou stínových těles, Bump mapping, Normal mapping, Specular highlights Částicový systém, Plnohodnotný fyzikální systém
Hry	Doom 3 (2004), Quake 4 (2005), Prey (2006), Enemy Territory: Quake Wars (2007), ...

Tabulka 2.5: Tabulka vlastností enginu iD Tech 4

Název	Source
Výrobce	Valve Corporation
Rok vydání	2004
Licence	Plná práva má pouze výrobce a vydavatel
Cena	Neveřejná informace
Podporované 3D API	DirectX 8, DirectX 9, DirectX 10
Primární určení	Indoor / Outdoor
Optimalizační technologie	BSP, Portály, Occlussion Culling, LOD
Animační technologie	Skeletalní animace + skinning, Blednování animací, Morphing, Obličejové animace
Grafické efekty	Sprity, Animované sprity, Animované textury, Light mapping, Phongovo stínování, Radiosita, Gloss mapy, Dynamická světla, Stíny metodou stínových map, Enviromental mapping, Normal mapping, Parallax mapping, HDR, Light Bloom, Hloubka ostrosti, Motion blur, Film Grain, Barevné korekce, Částicový systém, Plnohodnotný fyzikální systém
Skriptování	Triger systém, SDK
Hry	Half-Life 2 (2004), Vampire: The Masquerade-Bloodlines (2004), Dark Messiah of Might and Magic (2006), SiN Episodes: Emergence (2006), Left 4 Dead (2008), Zeno Clash (2009), ...

Tabulka 2.6: Tabulka vlastností Source enginu

Název	Unreal Engine 3
Výrobce	Epic Games Inc.
Rok vydání	2006
Licence	Plná práva má pouze výrobce a vydavatel
Cena	$700.000
Podporované 3D API	OpenGL, DirectX 9, DirectX 10
Primární určení	Primárně Indoor / Outdoor
Optimalizační technologie	BSP, Portály, LOD
Animační technologie	Skeletalní animace + skinning, Inverzní kinematika, Keyframe animace, Blednování animací, Obličejové animace
Grafické efekty	Sprity, Animované sprity, Animované textury, Light mapping, Phongovo stínování, Radiosita, Gloss mapy, Volumetrická světla, Dynamická světla, Anisotropní osvětlení, Stíny metodou stínových map, Stíny metodou stínových těles, Projekční stíny, Enviromental mapping, Lens Flare, Normal mapping, Emboss bump, Env-mapped bump, Parallax mapping, Offset mapping, Relief mapping, Photonic mapping, Horizon mapping, Per-pixel displacement mapping, Virtual displacement mapping, Z-Bias korekce, HDR, Light Bloom, Emmisive, Hloubka ostrosti, Motion blur, Film Grain, Barevné korekce, Částicový systém, Plnohodnotný fyzikální systém
Skriptování	Vizuální skriptovací jazyk Unreal Script
Hry	Tom Clancy's Rainbow Six: Vegas(2004), Gears of War (2006) John Woo presents Stranglehold (2007), Mass Effect (2007) BioShock (2007), Unreal Tournament III (2007), Medal of Honor: Airborne (2007), Tom Clancy's EndWar (2008), Mortal Kombat vs. DC Universe (2008), Mirror's Edge (2008), Gears of War 2 (2008), Turok (2008), Army of Two (2008), X-Men Origins: Wolverine (2009), ...

Tabulka 2.7: Tabulka vlastností Unreal 3 enginu

Název	CryEngine 2
Výrobce	Crytek
Rok vydání	2007
Licence	Plná práva má pouze výrobce a vydavatel
Cena	Neveřejná informace
Podporované 3D API	DirectX 9, DirectX 10, OpenGL
Primární určení	Indoor / Outdoor
Optimalizační technologie	Dynamic LOD, Occlusion Culling, LOD
Animační technologie	Skeletalní animace + skinning, Blednování animací, Inverzní kinematika, motion warping
Grafické efekty	Sprity, Animované sprity, Animované textury, Light mapping, Phongovo stínování, Dynamická světla, Stíny metodou stínových těles, projekční stíny, Multi-samplet SS, Normal mapping, Emboss Bump, Env-mapped Bump, Parallax mapping, Specular highlights, Realtime ambient occlusion, HDR, Light Bloom, Hloubka ostrosti, Motion blur, Film Grain, Barevné korekce, Částicový systém, Soft particles, Volumetric particles, Volumetrické mraky, Volumetrický kouř, Volumetrická mlha, Plnohodnotný fyzikální systém
Skriptování	Vizuální skriptovací jazyk
Hry	Crysis (2007), Crysis Warhead (2008), Merchants of Brooklyn (2009), ...

Tabulka 2.8: Tabulka vlastností enginu CryEngine 2

Kapitola 3

Moderní grafické efekty

3.1 Per-Pixel osvětlení

Moderní grafické enginy, v dnešní době již téměř neobsahují Per-Vertex osvětlení, které je implementované do standartních grafických knihoven, zpravidla interpretování Gouraudovým stínováním. Díky shaderům, tj. programovatelným částím grafické karty, máme možnost vypočítat barvu pro každý fragment a použít například Phongovo stínování[17].

3.2 Normal mapping

Normálové mapování[17] je nástupcem Emboss bump mapování z roku 1978, metoda je určena pro per-pixel osvětlení a nelze ji použít na Per-Vertex, tedy klasické Gouraudovo stínování, které je implementované v OpenGL a DirectX. K docílení výsledku využívá dodatečné informace v doprovodné normálové texuře. První engine, který normal mapping implementoval byl iD Tech 4 od společnosti iD Software. Normal mapping pracuje na úrovni fragmentů, kde podle texturovacích souřadnic dostane z normalové textury informace o normále v každém bodě modelu. Získaný normálový vektor fragmentu se pak použije pro Per-pixel výpočet osvětlení pro daný fragment. Normálové textury mohou být uloženy buď v prostoru tangent nebo v prostoru objektu (například modelář Autodek 3DS Max navíc podporuje i obrazový prostor, které jsou ale pro účely „pohyblivé“ grafiky zbytečné.)

Normálová textura se dá vypočítat buď z difúzní textury, nebo výškové mapy, výsledek ale není příliš uspokojivý, nebo rozdílovou projekcí modelu s nízkým počtem polygonů a modelu s vysokým počtem polygonů. Druhým zmíněným výpočtem se docílí velmi uspokojivých výsledků, protože lze pro každý bod na modelu uložit, jakou by měl normálu, kdyby na daném bodě existoval vrchol. Častou chybou grafiků zde bývá fakt, že používají při texturování jednu část textury na několik různých částí modelu, v takovémto případě vzniknou na modelu chyby ve formě špatně nasvícených míst.

3.3 Parallax mapping

Efekt je též znám jako Virtual Displacement mapping[17], popřípadě jako Off-set mapping a vychází z klasického bump mappingu a normal mappingu. K docílení efektu vystouplých

detailů na modelu s nízkým počtem polygonů se používá normálová mapa v kombinaci s výškovou mapou, která značí, jak moc se mají texturové souřadnice posunout. Aby nedocházelo k deformacím v hraničních úhlech, doplňuje se algoritmus o tzv. offset limit, kterým zamezíme přílišným posunům koorinátů. tato modifikace se nazývá Parallax mapping with offset limiting.

3.4 Lightmapping

Technologie[4] používaná primárně pro simulaci osvětlení v realtime aplikacích. Myšlenka je poměrně jednoduchá, pro každý polygon ve scéně se předem vypočte, například pomocí radiozity tzv. mapa světla, která se ve scéně pro-násobí s difuzní texturou. Výsledkem je nasvícená scéna, bez použití realtime počítaných světel.

3.5 Shadowmapping

Výpočet stínů pomocí stínových map[19] je rychlejší varianta výpočtu realtime stínů. Scéna se vyrenderuje z pohledu světla pouze do hloubkového bufferu (stínová mapa), při renderu z pohledu kamery se pak kontroluje podle stínové mapy, zda je pixel zastíněn, podle porovnání přepočítaných hodnot hloubky.

3.6 Enviromental mapping

Mapování prostředí[19] je pohledově závislá metoda mapování textury na model tak, aby odraz od povrchu tělesa v určitém bodě závisel na poloze modelu a na poloze pozorovatele. Ve fotorealistickém renderingu by odrazem bylo skutečné okolí modelu, v herní grafice se používá optimalizace ve formě před-renderovaných textur pro dané prostředí, většinou cube map.

3.7 Water rendering

Metoda generování na první pohled realistické vodní hladiny[19] je jednou z nejvíce obměňovaných a vyvíjejících se technik herní grafiky. Zpočátku se používala pouze průhledná textura, resp. průhledná animovaná textura na plochém polygonu, která znázorňovala vodní hladinu. Textury pod vodní hladinou byly předem obarveny více do modra, což vzbuzovalo efekt „být pod vodou“. Později se na vodní hladinu začal používat Enviromental mapping, který dával vodě větší fotorealističnost díky lesku, bohužel lesk byl falešný. Nejlépe vypadající technikou je moderní vodní hladina založená na kombinaci odrazu a lomu okolního prostředí, společně s animovanou normálovou mapou. Pro výpočet refraxe a reflexe je již potřeba používat víceprůchodový rendering, protože je nutné vyrenderovat zvlášť scénu pod vodou a nad vodou s otočenou kamerou podél úrovně vody. Oba offscreen rendery jsou pak modifikovány pomocí normal mapy a dalších rovnic, používaných pro výpočet apod. a sjednotí se do jedné textury pomocí technologie multitexturing.

Kapitola 4

Projekt Genesis

Na dílčích částech projektu se celkem podílí kolem 40 lidí. Bakalřské práce na projektu založené pokrývají jen malou část celku. Popsat veškerou práci, která se na projektu zpracovala by bylo na dlouhé hodiny, proto zde vyzdvihnu jen 2 nejdůležitější, tj. Návrh hry, Řízení týmu.

4.1 Návrh hry

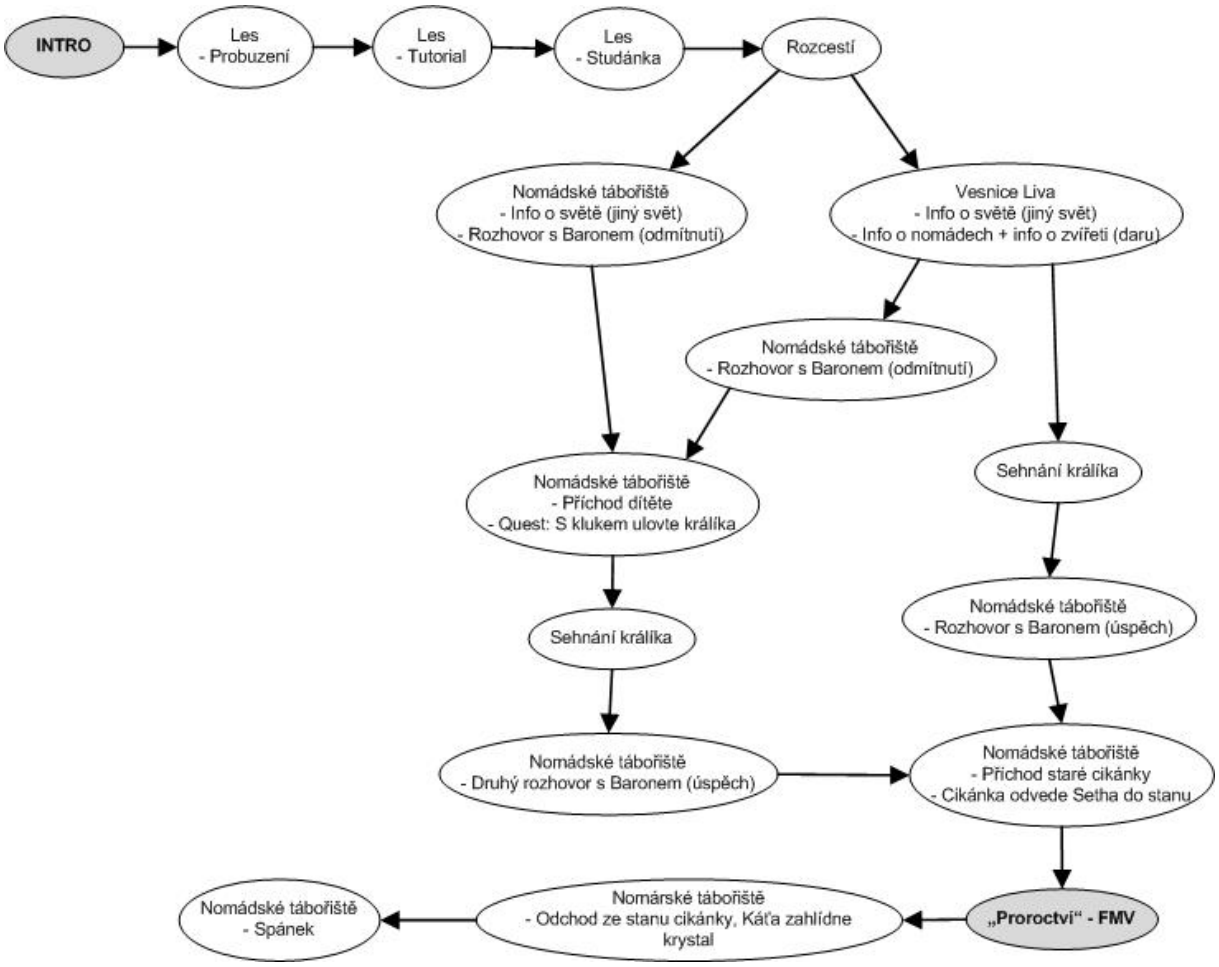
Návrh hry (Game Design) je jednou z prvotních procedur projektu, která se ale zároveň táhne po celé delce a průběžně se mění. Návrh hry musí obsahovat popis základního konceptu hry, příběh, scénář hry a sekvencí, dialogy, popis aktérů, herního prostředí, lokací, předmětů, questů[1], miniher a další. Nedílnou součástí je i návrh herního jádra a doprovodných editorů. Všechny uvedené části a další nezmíněné je nutné sepsat do jednoho dokumentu, kde k aktuální verzi bude mít vždy každý člen projektu přístup, tomuto dokumentu se v praxi říká Design dokument. Pro naše účely jsme se rozhodli použít internetový systém Media Wiki, který umožní kromě nastavení práv dynamickou tvorbu stránek každému uživateli. Systém navíc ukládá kompletní historii změň, takže vše lze přehledně dohledat.

Herní ukázka, která by v září tohoto roku měla vzniknout bude 3D RPG[2]. Hra by měla obsahovat vyprávění příběhu pomocí předskriptovaných herních CUT sekvencí a předrenderovaných FMV sekvencí, herní dialogy, volný pohyb po 3D světě, minihry, sbírání a manipulaci předmětů a souboje s počítačem řízenými protivníky.

Příběh je dobré pro přehlednost a jeho větvení popsat kromě scénáře ještě tzv. příběhovým diagramem (někdy také nazýván bublinkový diagram). Hlavní důvod, proč se příběhový diagram používá je, že lehce ukáže všechny možnosti, kterých lze docílit a jak lze výsledek ovlivnit rozhodování uprostřed hry a hlavně rychle odhalí chyby a slepá místa. Na obrázku 4.1 můžeme vidět příběhový diagram pro technické demo projektu. Kompletní scénář, svázaný s tímto diagramem lze nalézt v příloze na straně 45.

[1] Úkol zadaný herní postavou, po jehož splnění je hráč zpravida odměněn hernímy penězi, nebo předměty

[2] Role Playing Game - hra na hrdiny. Svět ve hře je většinou zcela vymyšlen a hlavní postava (postavy) se vylepšují podle daných pravidel. Ve hře je velmi důležitý příběh.



Obrázek 4.1: Příběhový diagram pro technické demo hry.

Strukturu Design dokumentu v době psané této bakalářské práce lze nalést v příloze na straně 51.

4.2 Řízení týmu

Součástí mé bakalářské práce sice nebylo řízení projektu, ale byla to má úloha v rámci projektu jako celku. Šest studentů mělo jednotlivé části zadané jako bakalářský projekt a dalších zhruba čtyřicet lidí na projektu spolupracovalo externě. Jak studenti, tak i externisti mají jiný přístup k práci a jinou úroveň odpovědnosti, proto jsem zvolil i jiný přístup k jejich řízení.

Koordinovat tým studentů bylo obtížnější, na rozdíl od externistů, protože brali z velké části práci jako povinnost, která musí být řádně zdokumentována doprovodnou bakalářskou prací. Někteří věnovali studiu a psaní práce tolik času, že nezbyl potřebný čas na implementaci jim vytyčeného problému. Proto jsme pro zvládnutí implementace klíčovou část problému

přesunuli již do zimního semestrální projektu. Pokud by se zvládlo vše podle původního plánu a semestrální projekt by byl u všech splněn na sto procent, k dnešnímu dni by byl projekt v první spustitelné a hratelné fázi. Bohužel klíčový student neměl v únoru letošního roku dokončenou část, kterou měl mít již v prosinci 2008 a i přes veškerou snahu nestihl do začátku června dokončit to, co mělo být na přelomu února a března. Jedná se o editor scén, na kterém nejpozději od začátku června měl vytvářet a skriptovat scény jiný student.

Řešení časového skluzu, který tímto problémem vznikl bylo, že já, jako programátor jádra, jsem musel vytvořit další sadu nástrojů, která zastoupila v té době nedokončený editor scén. Bohužel vzhledem k nutnosti přerušit práci na jádru kvůli editoru, na kterém byla závislá jiná práce, se zdržel kompletní vývoj a v době odevzdání této práce není herní jádro připraveno.

Externě na projektu spolupracovali mimo jiné vybraní studenti ČVUT FEL v rámci předmětu Y36MM1 - Multimédia 1, pod záštitou Ing. Romana Berky Ph.D., kteří pomáhali při zpracování některých motion capture animací, kde jejich konzultant a zadavatel jsem byl já, díky čemuž jsem měl plnou kontrolu nad měřením a zpracováním výsledku. Další externisté byli konzultanti z dvou herních studií, zvukař, hudebník, správce webového serveru, správce serveru s SVN repositářem, modely pro fototextury, scénárista a samozřejmě sponzor projektu firma A|W Graph, která nám zajistila legální software.

Kapitola 5

Návrh implementace

Před návrhem jádra, bylo nutné zvolit použité technologie od programovacího jazyku po seznam plánovaných algoritmů k implementování. Pro psaní her se dá použít mnoho jazyků, záleží ale na typu a zaměření hry. Pokud bychom chtěli psát hru pro web, máme na výběr z PHP, ASP, Flash společně s Action Scriptem, AJAX, SilverLight a další. Pokud by cílová platforma byl mobilní telefon, můžeme si zvolit mezi Javou, kterou lze použít s MIDP 2.0 pro většinu dnešních i starších mobilních telefonů, .NET pro mobilní telefony s operačním systémem Microsoft Windows Mobile, nebo C++ pro většinu telefonů s operačním systémem Microsoft Windows Mobile, Nokia Symbian, Google Android, Apple iPhone OS, PalmOS nebo dalších. Microsoft nám s technologií XNA[3] a .NET umožňuje programovat pro konzoli Microsoft X-Box 360. Co nás ale nejvíce zajímá, je programování pro osobní počítače.

Jedno ze základních rozhodnutí u her na pc je, zda bude 2D nebo 3D, jak hra bude složitá a graficky propracovaná a pro jaký operační systém bude určena. Z návrhu víme, že naše hra bude 3D Fantasy RPG , takže budeme potřebovat silný a výkonný jazyk. I přes to, že hru určujeme primárně pro Microsoft Windows, rádi bychom ji v budoucnu zkusili implementovat například na linux. Multiplatformnost je poměrně problemová část, která značně vyřazuje velkou část programovacích jazyků, přesněji ty, které jsme ještě nevyřadili kvůli požadovanému výkonu náročné 3D Aplikace. Jaké jsou tedy možnosti? Asi jako jedna z pvních variant může být jazyk Java. Java je velmi silný jazyk s v dnešní době již dobrou podporou 3D. Máme na výběr mezi OpenGL přes celou řadu knihoven, kde z mé zkušenosti nejkvalitnější je knihovna LWJGL [1], která podporuje dokonce i nové OpenGL 3. V Javě máme možnost použít i Microsoft DirectX přes knihovnu Java 3D, která má ale rozhraní a trochu jiný přístup. Programovací jazyk Java má bezesporu výhody co se týče rychlosti psaní kódu a multiplatformnosti, bohužel při zatížení, které dokáží 3D aplikace, začne Javu výrazně spomalovat Garbage Collector[1], který se pokouší mazat data, které již nepotřebujeme.

Kvůli budoucí podpoře Linuxu přichází jako další jazyky, které jsou velmi podobné Javě a umožňují díky nástrojům od Microsoftu ještě rychlejší vývoj, jedná se o sadu jazyků pro .NET, přesněji C#.NET, Visual Basic.NET nebo Managed C++. Pod linuxem se dá sice .NET s poměrně uspokojujícími výsledky spustit přes interpreter MONO [12], bohužel OpenGL není pod .NET téměř použitelný jazyk, o něco méně než Javu zpomaluje Garbage

[1] metoda automatické správy paměti

Collector, a ani podpora DirectX pod .NETem není nejideálnější, vzhledem k faktu, že před 2 lety byl její vývoj ukončen. Primární grafická knihovna pro .NET je XNA, což je nástavba nad DirectX. Bohužel, opět na Linuxu nepodporovaná.

Jediný použitelný jazyk, který sice není přímo multiplatformní a je nutné jej překompilovat pro každý operační systém je jazyk C++. Rychlost C++ je při práci s pamětí několikanásobně rychlejší a programátor má absolutní kontrolu nad jejím uvolňováním. Absolutní moc C++ je tedy zároveň i jeho nevýhodou, protože v něm lze lehce docílit úniku paměti (Memmory Leak). C++ podporuje jak OpenGL tak DirectX a v něm lze navrhnout tak, aby šlo použít obě knihovny. Volba, který jazyk použít, tedy byla poměrně snadá a rovnou jsme při návrhu aplikace počítali s programovým rozhraním C++.

V první fázi jsem se do jádra rozhodl implementovat pouze OpenGL, do budoucna je však jádro lehce rozšířitelné o další knihovny, typycky asi Microsoft DirectX 9, 10.1 nebo 11.

V druhé řadě je potřeba vybrat algoritmickou cestu, kterou se chceme vydat. Vytvořil jsem tabulku vlastností (feature list), které by měl do budoucna engine implementovat. Každé vlastnosti jsem určil prioritu, tj. hodnotu od jedné do pěti. Položky s prioritou 1 by se měli stihnout implementovat do září. Další vlastnosti podle času a možnostech pokračování práce na jádru. V tabulkách 5.1 a 5.2 je vše patřičně sepsáno a rozděleno do jednotlivých kategorií.

Kategorie	**Vlastnosti**	**Priorita**
Obecné	Objektově orientovaný návrh	1
	Podpora Windows XP, Windows Vista, Windows 7	1
	3D Renderer používající OpenGL	1
	3D Renderer používající DirectX 9	5
	3D Renderer používající DirectX 10.1	5
	3D Renderer používající DirectX 11	5
Skriptování	Skriptování pomocí jazyku LUA	1
Osvětlení	Per-pixel osvětlení (Phongovo stínování)	1
	Lightmaping	4
Stíny	Shadow mapping	3
Textury	Základní	1
	Multi-texturing	1
	Animované textury	1
	Enviromental mapping	1
	Normal mapping	1
	Parallax mapping	1
	Projektivní textury	2
	Steep Parallax mapping	3

Tabulka 5.1: Tabulka plánovaných vlastností Midas Enginu.

Kategorie	Vlastnosti	Priorita
Optimalizace	Quadtree	1
	LOD	2
	Frustum Culling	1
	Occlusion Culling	2
	Geometry streaming	3
Animace	Skeletal animation	1
	Skinning	1
	Blendování animací	2
	Obličejové animace	2
Specielní efekty	SkyBox	1
	Voda	1
	Osově-orientovaný billboarding	2
	Systém částicových efektů	2
	Mlha - volumetric fog	2
	Oheň	2
	Přechodové efekty (minimálně - in/out black fading)	2
	Déšť	2
	Lans Flare	2
	Volumetric lighting	3
	HDR	3
	Zrcadlo	3
	Střídání dne a noci	4
	Volumetrické mraky	4
	Sníh	4
Zvuk	2D zvuk	1
	3D zvuk	2
Umělá inteligence	Algoritmus hledání nejkratší cesty	2
	Konečný automat	2
	Rozhodování	3
	Předskriptované chování	3
Renderer	GLSL	1
	Přehrávání FMV sekvencí	1
	2D GUI	2
	HUD	2
	CG/HLSL	5

Tabulka 5.2: Tabulka plánovaných vlastností Midas Enginu.

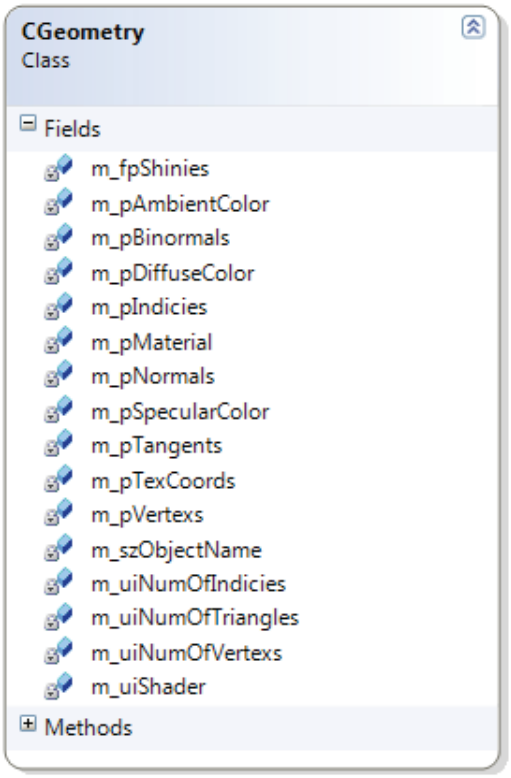
5.1 Návrh Midas Enginu

Jedna ze základních otázek implementace herního jádra bylo, jak to vše propojím. Navíc jsem chtěl vypracovat systém, kde půjde rychle cokoliv dodělat. Nejde mi ani o systém pluginů, jako spíš o přehledné SDK aplikace. Při návrhu jsem se snažil inspirovat architekturou známých a používaných enginů, kde největší inspirací pro mě byla publikace [13], která popisuje architekturu multiplatformního Building Block 3D enginu.

Návrh jádra jsem rozdělil do několika částí - Renderer, který se stará o vykreslování a komunikaci se zvoleným grafickým aplikačním rozhraním, manager zdrojů (Resource manager), který zajišťuje, že v paměti nebude načteno nic vícekrát a instancuje již načtené zdroje (textury, zvuky, hudbu, modely, shadery), manažer scén, jehož úkol kromě načítání scén a sestavování grafu scény je i řešit přechody mezi scénami a optimalizační technologie, navíc ovládá správce shaderů, správce hudby, správce zvuků, správce skriptů, správce AI a správce hráče, kde jsou veškeré herní tabulky a rpg systém. Většina jádra by měla být psaná bez vláken, které budou použity až na umělou inteligenci a na paralelní načítání zdrojů s

vykreslováním.

Základní třída, která spojuje všechny části jádra do jednoho se bude chovat jako stavový automat, abychom mohli jádra snadno ovládat a měli kontrolu nad aktuálním děním. Základní vstupy do jádra budou stavěny na vlastních formátech. Důvod, proč jsem nepoužil již existující datové struktury, například pro modely, je fakt, že jsem nedokázal najít formát, který exportuje z modeláře Autodesk 3DS Max korektně kosti, potřebné materiály, binormály a tangenty. Každá geometrie v jádru má, jak je vidět na uml na obrázku 5.1, informace o materiálech (lesk, odraz ambientní složky, odraz difuzní složky, odrazovou složku), použitý shader a materiál, informace o geometrii, tj. indexy, vrcholy, jednu sadu texturových koordinátů a normály, tangenty a binormály pro jednotlivé vrcholy a jméno, které se používá pro identifikaci ve skriptovacím jazyce.

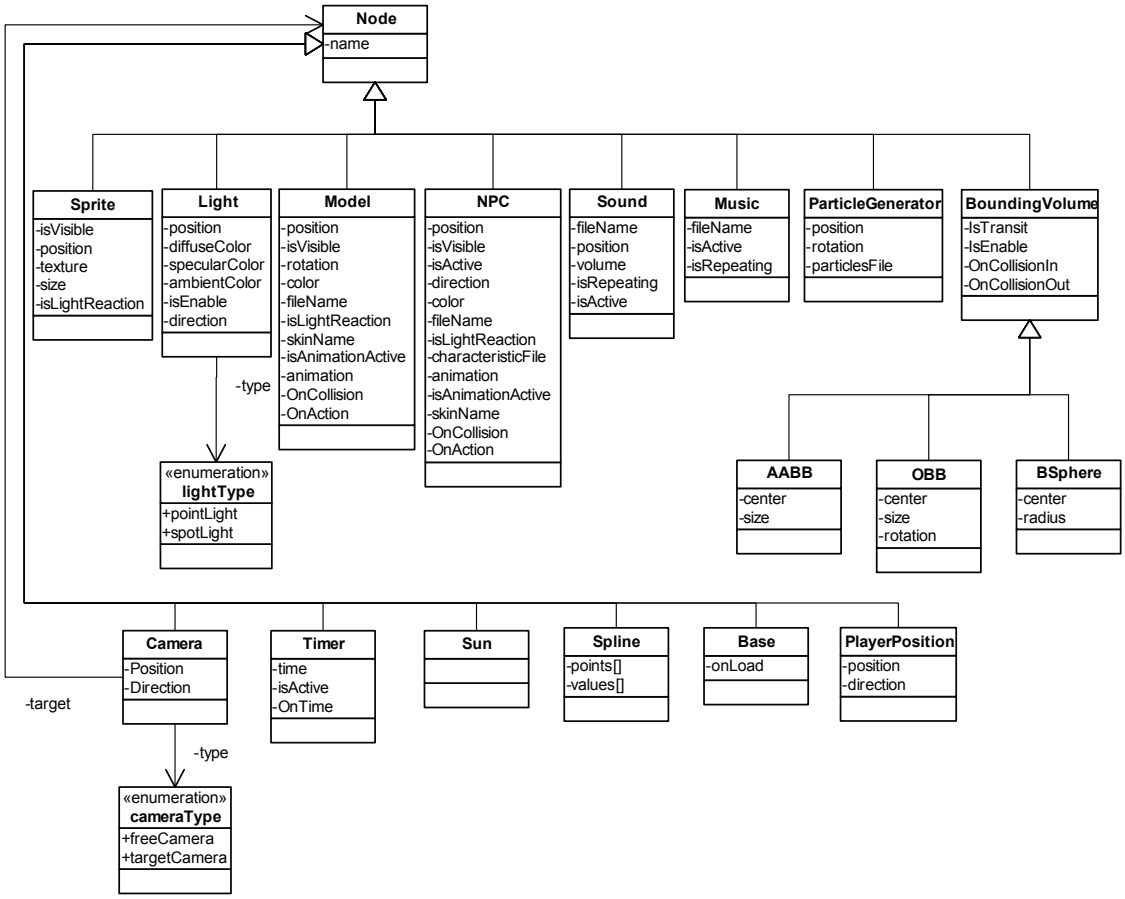


Obrázek 5.1: UML model struktury pro uchování geometrie.

Geometrie je oddělena od modelu z jednoho zásadního důvodu, a to je možnost mít v jednom modelu více geometrických sad s více texturami, a zároveň použitelnost struktury geometrie ve scéně u terénu, s kterým se pracuje jinak. Model také obsahuje informace o kostech a skinningu, který je tabulkou vah svázán s geometrií.

Scéna bude v jádru uložena ve stromové struktuře, ke které budou existovat dva přístupy. Samotná stromová struktura bude řazena podle závislosti transformací, tzn. potomek vždy bude dědit transformace od rodiče a obohacovat je vlastníma. Tento typ stromu je určen primárně pro update scény. Druhý přístup bude pro render, kde budou jednotlivé instance

uloženy v jiné formě stromu, přesněji v QuadTree, což je algoritmus dělení prostoru do menších částí. Test viditelnosti se pak provádí na úrovni jednotlivých uzlů stromu a né na samotné geometrii. Algoritmus quadtree vyžaduje při načtení grafu scény rozdělit geometrii na jednotlivé části. V tomto případě mám volbu mezi řezáním geometrie, nebo mezi sdílením přesahujících polygonů. Vzhledem k faktu, že herní jádro by mělo podle průměrných dnešních grafických karet zvládat renderovat statisíce až milión polygonů, troufám si tuto část zjednodušit na sdílení přesahujících polygonů, které budou dostatečně malé, podle plánované grafiky, pro kterou je jádro určeno. Další optimalizací, která není závyslá na jádru, navíc je, že geometrie se do pravidelných bloků uloží nebo rozřeže již při exportu. Elemety, které by podle prvního návrhu měl graf scény obsahovat, lze vidět na obrázu 5.2.



Obrázek 5.2: Návrh první verze elementů grafu scény.

Pokud bychom do budoucna uvažovali o podpoře konzolích, nebo jen o podpoře jakéhokoliv ovladače, je dobré vytvořit zvlášť komunikační rozhraní, neboli control wrapper, který se bude starat o události klávesnice, myši a ostatních handlet zařízeních. Jádro by také mělo obsahovat správce chyb, abychom mohli veškeré nečekané události, chyby a další informační hodnoty ukládát do logu. To, zda aplikace nezapisuje do zakázané paměti a zda má dostatek prostředků se dá v jednoduché verzi ohlídat globálním přetížením operátlorů new, new[],

delete a delete[].

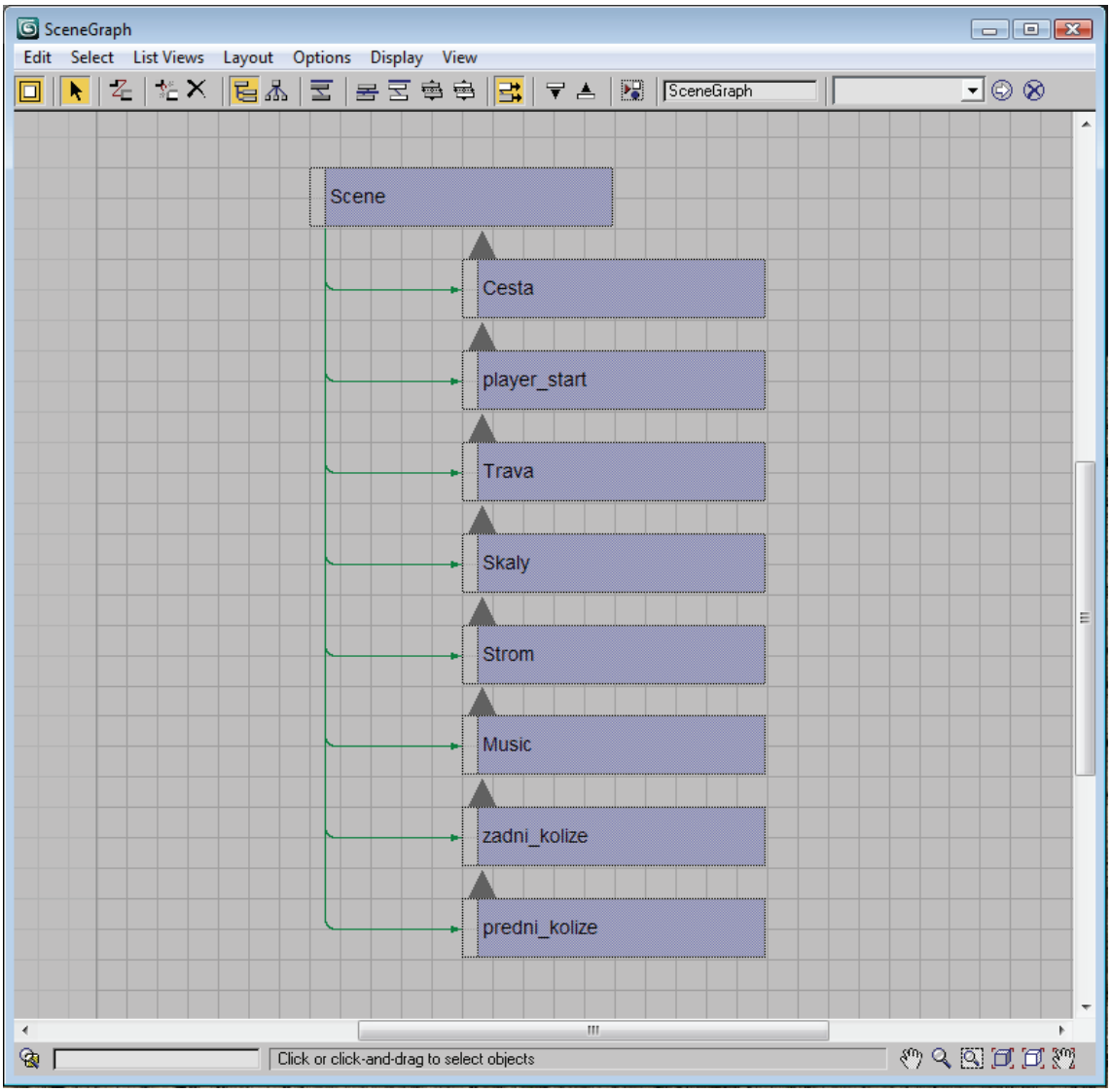
5.2 Návrh doprovodných aplikací a pluginů

Abychom mohli do jádra dostat potřebnou geometrii ve vlastním formátu, je nutné napsat konvertor ze stávající geometrie, nebo exporter do existujících modelářů. Cesta, kterou jsem si zvolil byla náročnější, ale výsledkem je maximální volnost při přístupu k datům při exportu. Jako modelář máme zakoupené sponzorem licence na Autodesk 3DS Max 2009, který obsahuje i tzv. Max SDK, což jsou zdrojové kódy potřebné pro psaní a kompilaci pluginů ve formě dynamických knihoven. Psaní pluginů do 3DS max má jednu zásadní nevýhodu a to je, že z důvodu historického vývoje SDK obsahuje více metod a postupů, přes které se dá dopracovat ke stejnému i lepšímu výsledku.

3DS Max umožnuje psaní pěti druhů pluginů - Modelovacích, Animačních, Materiálových, Renderovacích a Export/Importních. Z počátku práce na projektu jsem potřeboval vypracovat pouze plugin pro export geometrie a materiálů, nakonec však kvůli již zmíněným problémům se zpožděním dokončené práce jednoho člena týmu jsem byl nucen vypracovat pluginů víc, a to modelovacích. Co rozhodně stojí za zmínku v 3DSMax SDK, je třída IGame, která je od verze 8 navržena pro herní vývojáře, kteří mají rychlý přístup pro export ke všemu, co potřebují. Navíc třída jako jediná umožňuje identifikovat biped[2] od kosti. Max rozlišuje normální kost a bipeda, což je objekt integrovaného pluginu Character Studio, v podstatě se jedná o humanoidní kostru s již předem vytvořenou hierarchií a rigingem[**?**, **?**, **?**]. Hlavní výhoda bipedu je kromě již zmíněného hotového rigingu hlavně jednoduchá správa a blednování již hotových animací, což nám kosti jednoduše neumožňují. Ještě výhodnější by bylo použít animační systém (plugin) CAT, který je původně z modeláře Softimage | XSI[9], ale je placený a pro studenty špatně dostupný.

Je tedy nutné vytvořit do 3DS Maxe vlastní materiál, který kromě přiřazení diffuzní mapy, normal mapy a height mapy umožní i nastavit shader ve formátu enginu. Druhým pluginem je export geometrie a již vytvořeného materiálu do formátu modelu enginu. Třetí plugin je zástupný level editor, který do 3DS Maxe přidá objekty grafu scény, který jádro využívá. Při prostudování obejtků a vlastnosí 3DS Maxe, dojdeme k závěru, že musíme podle grafu scény doimplementovat strukturu pro model, npc, hudbu, zvuk, počáteční pozici hráče, triger[3], pomocnou značku pro určování pozice skriptéra a informace o scéně. K vytvoření grafu scény lze využít Schematic View (Graf scény v 3DS Max), což není nic jiného než graf dědění transformací, jak je znázorněno na obrázku 5.3.

[2] Druh humanoidní kostry v modeláři 3DS Max, který je automaticky narigován.
[3] Virtuální geometrie používající se jako generátor událostí v závislosti na interakci.



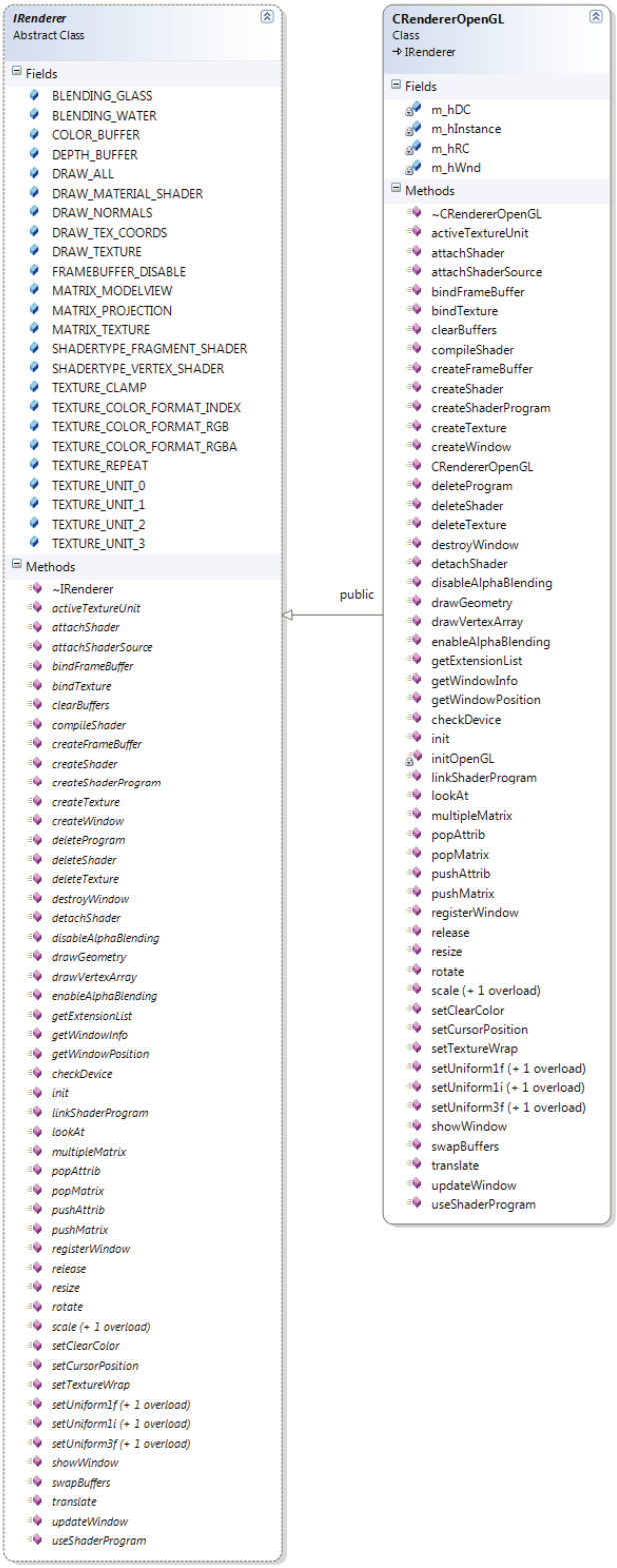
Obrázek 5.3: Ukázka Schematic View modeláře Autodesk 3DS Max.

Výsledný graf scény lze přesně tak, jak je určen v modeláři, exportovat pomocí dalšího pluginu, který je nutné připravit. Kromě pluginů do 3DS Max je také dobré připravit pro grafiky program, který jim umožní model zobrazit dřív, než jej načtou do scény. K tomuto účelu se vytvoří ještě zobrazovač modelů, který bude využívat herní zobrazovací jádro. To, co v podstatě bude mít navíc zobrazovač modelů, bude primárně trackball algoritmus založený na quaternionech, který umožní bez gimbal locku[4] prohlížet načtený model.

5.3 Renderer

Renderer je klíčová část enginu, co se týče multiplaformnosti. Pro každý operační systém a každou grafickou knihovnu je napsán vlastní potomek třídy IRenderer, který implementuje veškeré metody a definuje veškeré konstanty. Za předpokladu, že je zbytek jádra korektně napsán, neměl by být problém po napsání nového "pluginu" do rendereru spustit engine na operačním systému Windows, nebo třeba na Sony Playstation. Při návrhu metod Třídy IRenderer jsem se snažil dodržet jak zásady DirectX, tak i OpenGL a najít rozumný kompromis. Například vykreslování modelů je kompletně v rukou rendereru, kvůli rozdílným uložištím pro data a různým postupům při vykreslování. Vše je psáno tedy tak, abychom mohli využít jak extenzí OpenGL, výhod nového OpenGL 3 a low-api výhod Direct3D. UML abstraktní třídy IRenderer a implementace OpenGL 2.0 pro Microsoft Windows je možné vidět na obrázku 5.4

[4] Chyba, při které, při použití Eulerových úhlů, ztratíme jeden stupeň volnosti.



Obrázek 5.4: UML model rendereru a OpenGL implementace.

Zásadní otázkou při návrhu bylo použití Shaderů. Nejvíce logický postup, z hlediska podpory pro DirectX i OpenGL by bylo použít CG Shadery od společnosti NVidia, bohužel jsem neměl dostatek času přejít z pro mě již dobře známého GLSL. Pokud bych přecházel například na Sony Playstation 3, bylo by potřeba implementovat právě CG, pro XBox 360 je to HLSL od Microsoftu, na Windowsech, Linuxu a dalších pc operačních systémech běží jak pod OpenGL tak pod DirectX také CG. Přehledněji je vše vidět v tabulce 5.3. Volba GLSL do začátku z pohledu budoucího vývoje nebyla nejmoudřejší, ale díky rozšířitelnosti rendereru není problém přejít na jiné shadery.

Shadery	**Grafické API**	**Platforma**
GLSL	OpenGL	Microsoft Windows Linux MacOS
GLESSL	OpenGL ES 2.0	Pandora
CG	OpenGL ES 1.0	Sony PlayStation 3
CG	OpenGL	Microsoft Windows Linux MacOS
CG	DirectX	Microsoft Windows
HLSL	DirectX	Microsoft Windows Microsoft X-Box 360
HLSL	XNA	Microsoft Windows Microsoft X-Box 360

Tabulka 5.3: Shadery a jejich použití na různých GAPI a platformách.

Při návrhu Shader systému vznikla otázka, jak implementovat shadery tak, aby mohl mít každý materiál jiné vlastnosti a aby mohl být výsledek víceprůchodový. Vyrešit problém rozdílnosti lze několika způsoby. Je možné vytvořit databázi shaderů, které se budou dynamicky linkovat a kombinovat tak, aby vznikl požadovaný výsledek, nevýhoda tohoto řešení je čas a ztráta přehlednosti nad výsledkem, protože mohou začít vznikat stovky kombinací. Přeci jen, pokud bychom implementovali 3 druhy světla (bodové, směrové, reflektorové), 20 druhů materiálů (sklo, železo, bronz, umělá hmota, dřevo, chrom, ...), 5 grafických efektů (diffuse, normal mapping, parallax mapping, specular mapping, enviromental mapping), tak se dostaneme na 300 různých kombinací a nepočítáme možnost kombinace různých grafických efektů, třeba normal mappingu a specular mappingu. S kombinacemi efektů se dostaneme na 1500 kombinací. Nemluvě o tom, že systém dynamického kombinování shaderů by bylo poměrně obtížné a časově náročné implementovat a časté prohazování shaderů na grafické kartě není nejlevnější operace. Druhou možností, která je jednodušší, je vytvořit jeden velký parametrizovatelný shader. Ušetřili bychom čas při přehazování shaderů v grafické pipeline.

Řešení, které jsem navrhl kombinuje výhody obou, již zmíněných, a přidává několik dalších rozšíření. Jedná se v podstatě o balíčky shaderů, které jsou propojeny jednou materiálovou funkcí. K implementaci myšlenky jsem využil interpretovaný jazyk Lua, který mi kromě programu shaderů umožňuje volat dynamicky i další funkce jádra. Lua pouze navazuje své funkce přímo na funkce jazyka C++, což umožňuje poměrně snadno ovládat jádro aplikace ze skriptovacího jazyka. Základní struktura shaderu je vidět v následujícím kódu:

```
[0]  number_of_pass = 3;
[1]  shader_type = "GLSL";
```

```
[2]  base_shader = false;
[3]  --base_shader = "baseshader.mes";
[4]
[5]  function pass_1()
[6]     -- Prostor pro nastavení stavů průchodu, viz dále
[7]
[8]     VertexShader("ZDE JE ZDROJOVÝ KÓD VERTEX SHADERU PRO PRVNÍ PRŮCHOD");
[9]     FragmentShader("ZDE JE ZDROJOVÝ KÓD FRAGMENT SHADERU PRO PRVNÍ PRŮCHOD");
[10] end
[11]
[12] function pass_2()
[13]    -- Prostor pro nastavení stavů průchodu, viz dále
[14]
[15]    VertexShader("ZDE JE ZDROJOVÝ KÓD VERTEX SHADERU PRO PRVNÍ PRŮCHOD");
[16]    FragmentShader("ZDE JE ZDROJOVÝ KÓD FRAGMENT SHADERU PRO PRVNÍ PRŮCHOD");
[17] end
[18]
[19] function pass_3()
[20]    -- Prostor pro nastavení stavů průchodu, viz dále
[21]
[22]    VertexShader("ZDE JE ZDROJOVÝ KÓD VERTEX SHADERU PRO PRVNÍ PRŮCHOD");
[23]    FragmentShader("ZDE JE ZDROJOVÝ KÓD FRAGMENT SHADERU PRO PRVNÍ PRŮCHOD");
[24] end
```

Na začátku kódu si nastavíme vlastnosti shaderu. Hodnota `number_of_pass` je počet průchodů, které shader umožňí, `shader_type` jazyk, kterým je zrojový kód napsán a hodnota `base_shader` smí nabývat buď hodnoty false, což znamená, že se jedná o bázový shader, nebo String s cestou k bázovému shaderu, pak se jedná o materiálový shader. Bázový shader obsahuje osvětlovací model a grafické efekty, jedná se o větší parametrizovaný shader. Vertex shader na konci svého provádění volá funkci vec4 objectMaterialMain(inout vec4 vertexPosition), jejíž výsledek vrátí novou hodnotu vrcholu. Fragment Shader volá na konci stejnou funkci vec4 objectMaterialMain(inout vec4 fragColor), kde výsledkem je nová barva fragmentu. Definice těchto funkcí jsou obsahem materiálových shaderů a mohou se různě měnit. Tato myšlenka je založena na dynamickém linkování shaderů, kde jsem ale výsledek zpřehlednil kombinováním s parametrizovaným shaderem pro osvětlovací model a grafické efekty.

Má implementace shaderů jde ale ještě dál. Na počátku každého průchodu umožní herní jádro překlápět do různých, předem definovaných stavů, čímž lze nepříklad vytvořit efekt vodní hladiny bez zásahu do kódu aplikace. Pro překlápění stavů enginu využívám Luou nalinkované funkce do jádra. Tabulka 5.4 ukazuje nastavení vstupů a výstupů texturovacích jednotek pro jednotlivé průchody a další nastavení stavů jádra.

Funkce	Stavy	Výchozí hodnota
RenderAll	true false	Ano Ne
RenderShadowModels	true false	Ne Ano
RenderClipYPlaneUp	true false	Ne Ano
RenderClipYPlaneDown	true false	Ne Ano
SetY	0.0	Ano
RenderFrom	"light0" "camera" "y_minus_camera"	Ne Ano Ne
OutBuffer	"default" "depth" "color"	Ano Ne Ne
OutUnit	"standard" "renderOut1" "renderOut2" "renderOut3" "renderOut4"	Ano Ne Ne Ne Ne
TextureUnit1	"diffuse+alpha"	Ano
TextureUnit2	"disable" "normal+specular" "normal+height" "normal"	Ano Ne Ne Ne
TextureUnit3	"disable" "renderOut1+height" "renderOut1+renderOut2" "renderOut1"	Ano Ne Ne Ne
TextureUnit4	"disable" "renderOut3+renderOut4" "renderOut3"	Ano Ne Ne

Tabulka 5.4: Tabulka nastavení stavu enginu pro každý průchod.

Funkce RenderAll ignoruje předchozí stavy a pošle na výstup veškerou geometrii, která se standardně má renderovat. RenderShadowModels nechá renderovat pouze optimalizované verze geometrie pro rychlé renderování stínů (je zbytečné, aby při dvouprůchodovém algoritmu pro stínové mapy jsme v prvním průchodu, když se vypočítává pouze stínová paměť hloubky, renderovali modely, které nevrhají stín, nebo modely, které mají vysoký počet polygon.). Funkce SetY nastaví výšku úrovně ořezové roviny rovnouběžnou s osou x a osou z.

RenderClipYPlaneUp ořízne dolní část scény pod ořezovou rovinou, tj. renderuje se vše nad hodnotou nastavenou ve funkci SetY. RenderClipYPlaneDown renderuje vše pod hodnotou z funkce SetY. Funkce RenderFrom nastavuje, odkud se má scéna renderovat, na výběr máme od světla (vzhledem k tomu, že jsme definovali, že stíny bude vrhat pouze jedno světlo ve scéně, postačí hodnota light0), nebo z aktuální kamery scény, nebo z aktuální kamery scény ozrcadlenou podél nastavené úrovně y. Zbytek funkcí se vztahuje čistě ke vstupům a výstpům z aktuálně renderovaného průchodu. Funkce OutBuffer nastaví, zda chceme renderovat mimo klasického postupu (default), kdy se renderuje jak do paměti barev, tak i do hloubkové paměti, pouze do paměti barev, nebo jen do paměti hloubky. OutUnit nastaví, zda se má renderovat do klasického frame bufferu (standard), nebo zda provádíme offscreen rendering do textury, kterou v dalším průchodu dostaneme v nastavené texturovací jednotce. V jádru jsem se omezil pouze na 4 texturovací jednotky, i přes to, že

většina grafických karet na dnešním trhu využívá desítky a dokonce i stovky texturovacích jednotek. Důvod, proč využít pouze 4, je podpora starších grafických karet, například NVidia řady 7. Pokud by bylo do budoucna potřeba rozšířit jádro o podporu dalších texturovacích jednotek, například kvůli autostereoskopickým obrazovkám, kde se výsledný obraz zkládá z 8 pohledů, je nutné přidat do renderu další konstanty a navázat je na jazyk Lua. Další funkce TextureUnit1 až TextureUnit4 nastavují, které informace chceme na vstupních 4 texturovacích jednotkách. Úmyslně jsem neumožnil všem jednotkám nastavit vše, protože některé kombinace nikdy v jádru nenastanou. 4 texturovací jednotky je poměrně málo, proto pro napříkad efekt parallax mappingu a efekt vody s reflexí a refraxí můsíme využít všech 4 složek jednotek, tzn. například volání TextureUnit3("renderOut1+height"); znamená, že na třetí texturovací jednotce dostaneme ve složkách RGB první výstup z předchozích průchodů a ve složce alpha hodnotu výškové mapy renderované geometrie. RenderOut1 až renderOut4 lze také pochopit jako úložiště, které můžeme využít pro přenos informací mezi průchody.

5.4 Manažer zdrojů

Veškeré zroje v herním jádru, tj. textury, zvuky, hudbu, modely, shadery a skripty je potřeba mít pro ušetření paměti nahrané v ramce pouze jednou, to zajišťuje právě manager zdrojů, který podle relativní cesty od hlavní složky aplikace, která je unikátní v rámci projektu, vytvoří identifikátor pro každý načtený zdroj a uloží jej i s odkazem k paměti do množiny. Pokud při dalším načtení tato množina již cestu obsahuje, pouze se předá ukazatel do paměti, kde se příslušný zdroj nachází. Aby vždy při přístupu k datům nedošlo ke zpomalení, kvůli vyhledávání zdroje podle jména v množině, používají se identifikátory cesty pouze pro načítání zroje a jádro následňe pracuje s přiděleným číselným identifikátorem, který není nic jiného, než adresa v paměti, kde se zdroj nachází. Pokud tedy víme, o jaký zdroj se jedná, jsme schopni bez prohledávání seznamu vrátit odkaz na potřebnou datovou strukturu.

Manažer zdrojů, se stará i o načítání grafických formátů, skriptů a shaderů. V prnví plánované verzi bude jádro využívat jako grafické formáty pouze BMP a TGA ve většině verzích.

5.5 Manažer scén, kolize a základy AI

Podle předpokladů a průzkumu náročnosti dnešních AAA titulů se ve scéně vykresluje statisíce, až milion polygonů a je nutné posílat do grafické pipeline jen potřebné minimum. V první části této kapitoly jsem již popsal základy manažera scén a napsal něco o plánovaných optimalizačních algoritmech. To vše je nutné ale rozšířit i dál. Aby herní postava mohla po virtuálním světě chodit a vrážet do stěn, je nutné implementovat kolizní systém a základy fyziky, přesněji gravitaci a reakci na nárazy. Pro omezení se na výpočet kolizí v okolí hráče můžeme použít vytvořeného zobrazovacího stromu s quadtree, který nám umožní omezit výpočty z celého světa jen na nejbližší okolí. V jádru budou existovat tři zástupné kolizní objekty, tzv. obalová tělesa, která budou simulovat tvar objektů, s kterými se kolize bude počítat. Jedinou vyjímkou bude terén, který je i v grafu scény uložen jinak a u kterého se bude počítat s ostatními objekty přímo přes trojuhelníky. Jinými slovy veškeré modely a npc

budou zastoupeny při výpočtu zástupnou geometrií, k dispozici budou koule, která je pro výpočet kolizí nejrychlejší, osově zarovnaný kvádr (AABB) a osově orientovaný kvádr (OBB), u které se s terénem bude počítat "per triangle". Je tedy nutné implementovat výpočet kolizí mezi všemi kombinacemi sféra, trojuhelník, AABB a OBB. Výborná literatura na řešení kolizí je [6].

Jedním z velkých problémů, na které jsem narazil a nepodařilo se mi najít potřebnou literaturu, implementace vyhledávacího algoritmu umělé inteligence A* pro 3D scény, nakonec jsem problém vyřešil tak, že po celé scéně se při exportu rozmístí označené markery, které budou jen tam, kam se dá jít a pomocí těchto markerů, kde bude každý vědět o svých sousedech, se dá již vytvořit graf, na který lze efektivně aplikovat A* algoritmus.

5.6 Skriptovací jazyk

Jako skriptovací jazyk jsem vybral interpretovaný jazyk Lua, jazyk bude založen na systému událostí, kde v případě vzniku události bude volána jménem pevně daná funkce v přiřazeném skriptu. Skriptovací jazyk bude pomocí jmen přistupovat ke všem objektům grafu scény a bude moci pomocí sady funkcí, které navrhla Kateřina Štemberová v paralelní bakalářské práci "Genesis - skriptování animací pro herní engine" objekty ovládat.

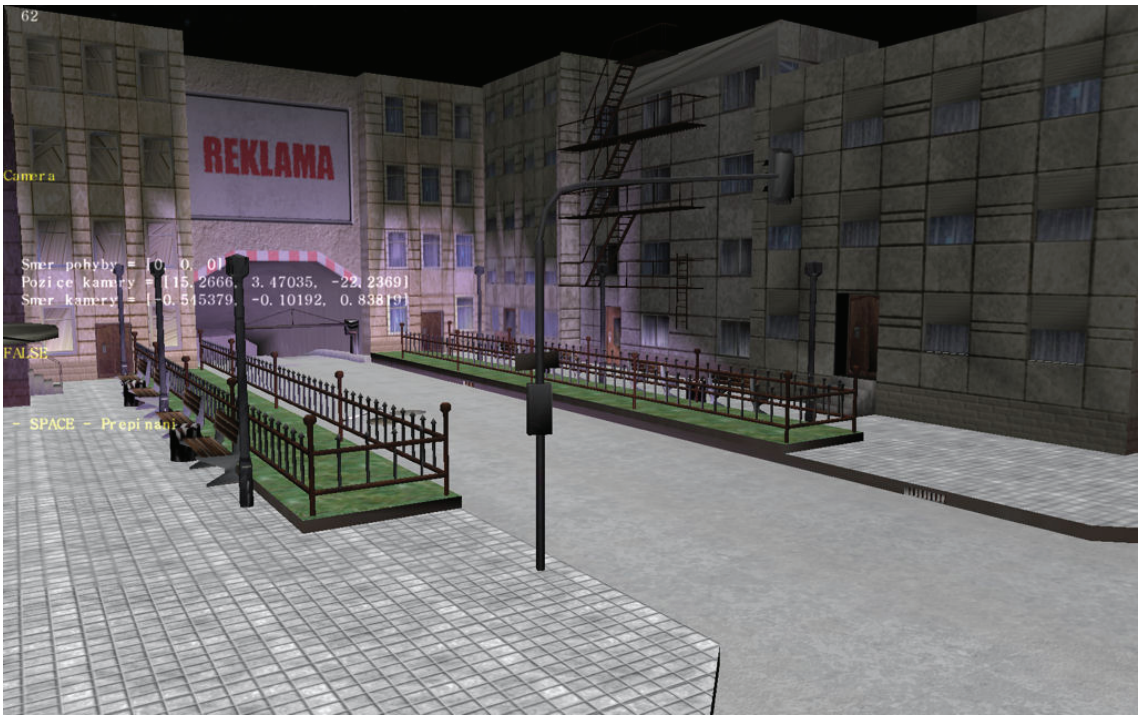
Kapitola 6

Výsledky

6.1 Realizace herního jádra

Od návrhu aplikace k její realizaci je dlouhá cesta a pro plnohodnotné otestování výsledku je nutné vypracovat doprovodné nástroje, které dokáží jádro naplnit potřebnými daty. Mohl bych zde dlouze rozebírat detaily implementace jádra a okolních nástrojů, ale bohužel k tomu je v této práci málo prostoru, zůstanu tedy jen u popisu vybraných tříd a některých implementačních detailů. Bohužel jsem nestihl naprogramovat kolem poloviny vlastností, které jsou potřeba pro první hratelnou scénu, převážně kvůli tvorbě okolních nástrojů, které původně nebyly v popisu práce. Z jiného pohledu, část jádra, která je implementována je z částí otestována a z velké části i optimalizována. V průběhu implementace, na které je odhadem více jak 1500 hodin práce, vznikli 3 verze jádra. První verze testovala technologie jako je Octree, základní implementace shaderů, ovládání kamery, Frustum Culling, systém kolizí, načítání vlastní geometrie, export dat z 3DS Max a další. Ukázku z tohoto prototypu můžeme vydět na obrázkách 6.1, 6.2 a 6.3.

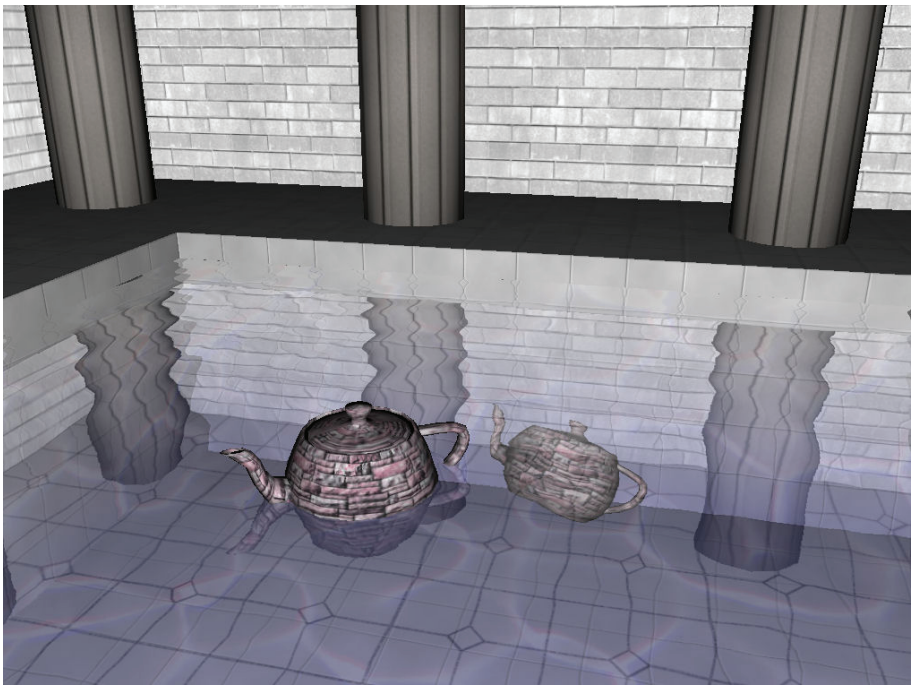
Vzhledem k faktu, že se jednalo o mou první implementaci, bylo vše pomalé a výsledek nebyl uspokojivý. První prototyp byl tedy pojat jako hřiště, na kterém jsem si dané technologie mohl nejprve ozkoušet a najít lepší implementační cestu. Druhá implementace (druhý prototyp) již sloužila jako základ pro dnešní verzi, jejím úkolem bylo zaměřit se hlavně na systém Shaderů a na grafické efekty. Druhá implementace obsahuje dvě testovací scény - vnější a vnitřní. Obě scény testují efekt reflexe a refraxe vodní hladiny a Phongovo stínování. Z Druhého prototypu se zachovalo kolem 80%. Ukázka je k nahlédnutí na obrázkách 6.4, 6.5 a 6.6.



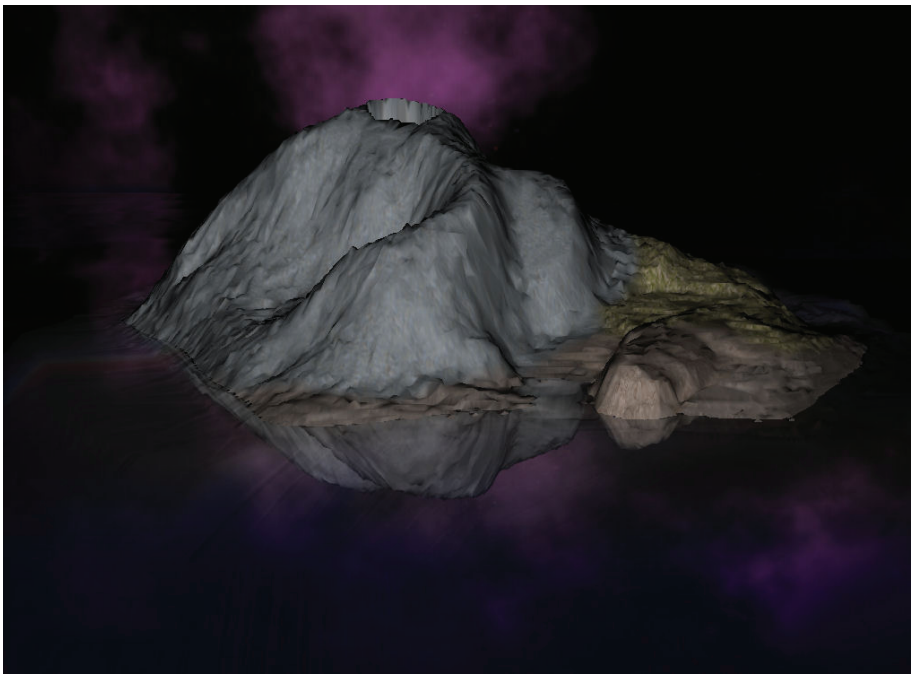
Obrázek 6.1: První prototyp - outdoorová scéna s vypnutými shadery.

Obrázek 6.2: První prototyp - Pohled na načtený model.

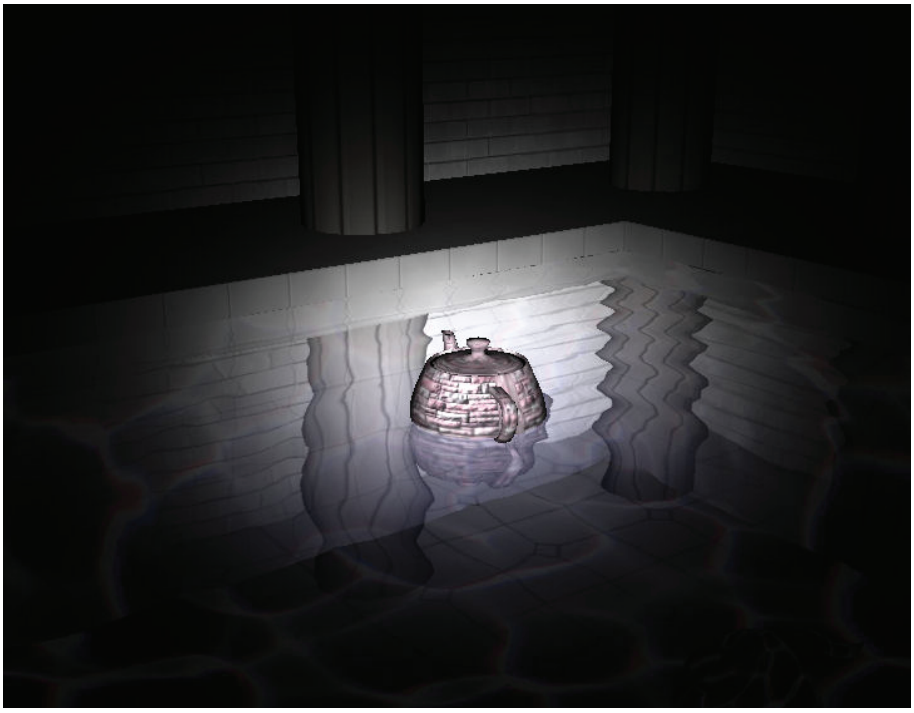
Obrázek 6.3: První prototyp - outdoorová scéna se zapnutými shadery



Obrázek 6.4: Druhý prototyp a základ dnešního jádra - Ukázka odrazu a odlesku.



Obrázek 6.5: Druhý prototyp a základ dnešního jádra - Outdoorová scéna.



Obrázek 6.6: Druhý prototyp a základ dnešního jádra - Reflektorové světlo.

Graf scény jsem byl nucen zjednodušit na minimum, které je potřeba pro hratelnou verzi, tak jak ukazuje výsledný UML diagram na obrázku 6.7. Všechny důležité části se však zachovaly.

Obrázek 6.7: UML finální implementace grafu scény.

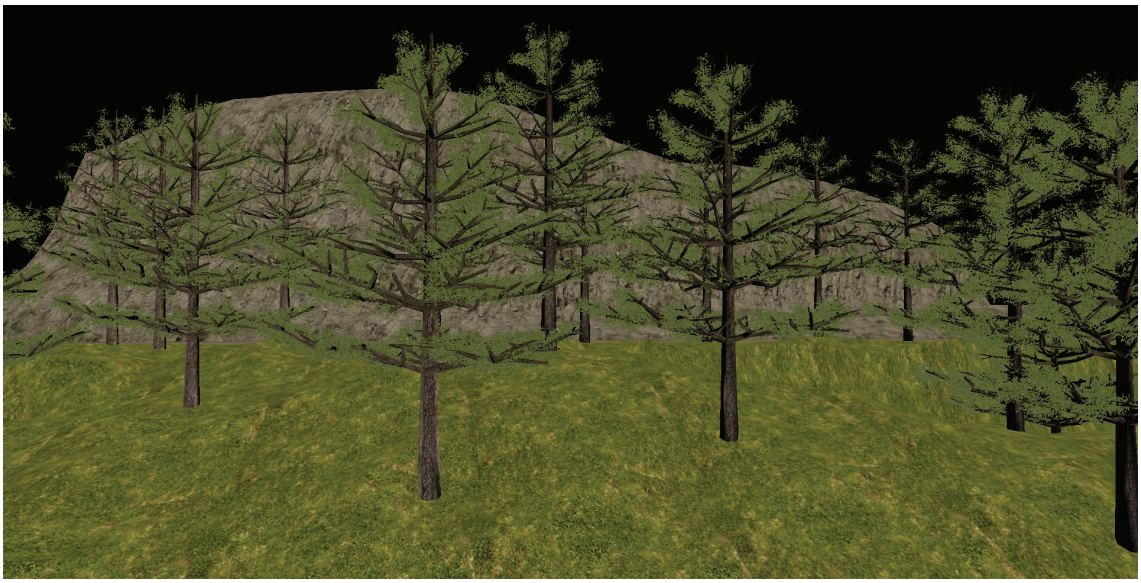
V aktuální implementaci, v době psaní této práce, je graf scény implementován pro update, tzn. je hierarchicky řazen pro dědění transformací a vykreslování a zároveň je z části implementován scriptovací jazyk Lua, který s některými objekty dokáže manipulovat v závislosti na události onLoad. V tabulce 6.1 lze najít plánovaný seznam událostí a v tabulce 6.2 podporovaný seznam funkcí pro manipulaci s objekty scény. Do budoucna se seznam znatelně rozšíří.

Událost	**Podporované objekty**	**Popis**
`on_collision`	Triger	Začátek průniku
`on_collision_out`	Triger	Ukončení průniku
`on_action`	Triger NPC	Akce hráče
`on_loop`	Triger	Události během průniku
`on_load`	Scéna Model NPC	Inicializační událost

Tabulka 6.1: Události pro skriptování.

Funkce	**Objekt**	**Popis**
SetPosition(String name, String nodeName)	Node	Nastavení pozice
SetRotation(String name, Real x, Real y, Real z)	Node	Nastavení rotace
AddPosition(String name, Real x, Real y, Real z)	Node	Připočtení pozice
AddRotation(String name, Real x, Real y, Real z)	Node	Připočtení rotace
SetActive(String name, Bool value, Bool branch)	Node	Aktivace nody
MoveNode(String name, String newParent)	Node	Přesune větev pod jinou node
SetScript(String name, String script)	Triger Model NPC Scéna	Nastaví script objektu
Play(String name)	Music Sound	Spustí zvuk resp. hudbu
Stop(String name) Switch(String name, String newMusic, String transition)	Music Sound Music	Zastaví zvuk resp. hudbu Přechod mezi hudbou
GoToScene(String scene, String playerPosition)	-	Přechod do další scény
SetCamera(String name, String transition)	Camera	Přechod na jinou kameru
StartSequence(IntArray timeLine, String function)	-	Začátek sekvence
EndSequence()	-	Konec sekvence

Tabulka 6.2: Funkce pro manipulaci s objekty scény.



Obrázek 6.8: Ukázka výsledné implementace - 500 tisíc polygonů, 60fps, 2 druhy shaderů.

Jedním z problémů, na které jsem narazil bylo vytváření událostí v čase. První myšlenkou bylo zavedení funkce Timer, která by pozastavila provádění skriptu na požadovanou dobu, touto metodou bych ale neuhlídal návaznost jednotlivých skriptů a vše by se špatně synchronizovalo. V řešení, které jsem použil jsem se inspiroval softwarem Adobe Premiere, kde se na časovou osu umísťují jednotlivé akce.

Funkce StartSequence dostane jako první parametr pole celých čísel, v kterém je sudý počet dat určující události na časové ose. Data jsou ve formátu dle vzorce 6.1. Id určuje číselný identifikátor události a time je čas v milisekundách od začátku sekvence. Druhým parametrem funkce StartSequence je název funkce v Lue, která bude volána.

$$array \quad = [id_1, time_1, id_2, time_2, ..., id_n, time_n] \tag{6.1}$$

Funkce bude volána vždy, když nastane událost podle předaného pole, a bude obsahovat jeden parametr, do kterého aplikace předá identifikátor aktuální události.

```
[0]  function udalost()
[1]     timeline = {1, 0,
[2]                 2, 1000,
[3]                 3, 1500,
[4]                 4, 5000}
[5]     StartSequence(timeline, "moje_osa")
[6]  end
[7]
[8]  function moje_osa(time)
[9]     if time == 1 then
[10]       -- udalosti v case 0ms
[11]    elseif time == 2 then
[12]       -- udalosti v case 1000ms
[13]    elseif time == 3 then
[14]       -- udalosti v case 1500ms
[15]    elseif time == 4 then
[16]       -- udalosti v case 5000ms
[17]    end
[18] end
```

Výhoda tohoto řešení, do budoucna, je možnost vytvořit nástavbovou aplikaci, podobnou softwaru Adobe Premiere, kde budeme mít časovou osu, na kterou budeme umísťovat jednotlivé scripty. Pokud bych to schrnul, aktuálně lze tedy vytvářet modelářem 3DS Max scény a modely, lze je exportovat do vlastního formátu a výslednou kompozici si můžeme proletět v Jádru. Fungují první průchody shaderů a lze scény jednoduše scriptovat pomoci události onLoad v jazyce Lua. Výsledek implementace je vidět na obrázku 6.8.

6.2 Realizace doprovodných aplikací

6.2.1 Model Viewer

Aplikace, která usnadní grafikům kontrolu exportovaných modelů. Aplikace využívá renderovací jádro hry, díky kterému je výsledek téměř identický s výslednou podobou. Zobrazovač modelů v aktuální verzi dokáže u vrcholů zobrazit orientaci normál, tangent a binormál. Pro ovládání aplikace je implementován trackball, rozšířen o pohyb vzhledem k souřadnicovému systému pozorovatele. Celá aplikace je psána v aplikačním rozhraní systému windows pro C++.

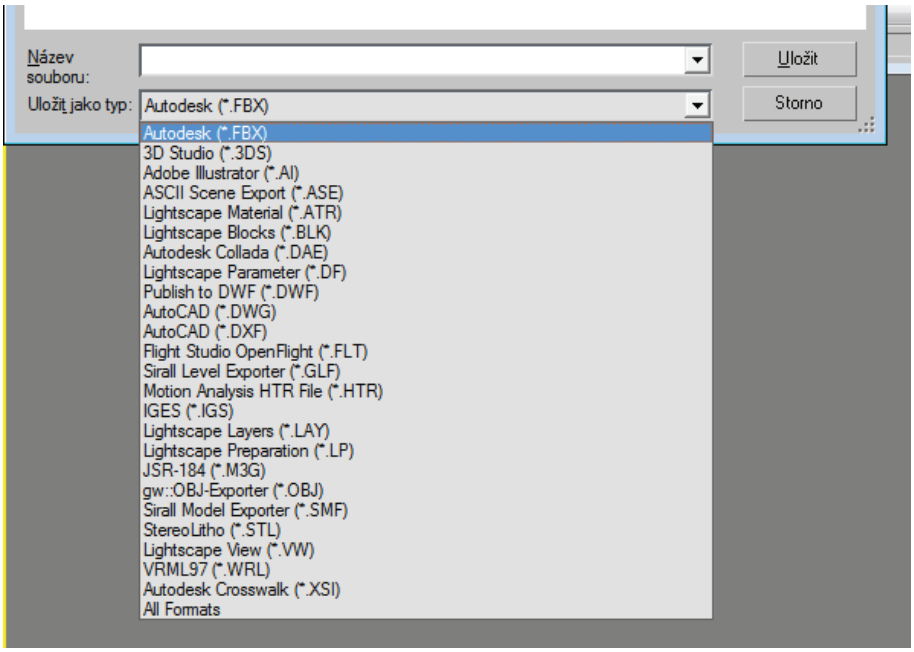
Obrázek 6.9: Program na zobrazování modelů ve formátu SMF.

6.2.2 SMF a GLF exporer (3DS Max pluginy)

Všechny pluginy pro software 3DS Max vyžadovaly studii zrojových příkladů a spoustu testů, vzhledem k nemožnosti výslednou dynamickou knihovnu debugovat. Problém také byl, že pluginy jsou závislé na verzi modeláře, tudíž nikdo z týmu nemůže v průběhu přejít na jinou verzi. Minimálně nemůže, pokud nemáme SDK dané verze. Všechny pluginy pro 3DS Max jsem byl nucen kompilovat pro 32 i 64 bitovou verzi, protože né všichni v týmu můžou používat rychlejší 64bitovou verzi Maxe. Při implementaci mám dvě možnosti, které použít při získání datových informací z modeláře. První je použít klasické 3DS Max objekty.

Zásadní nevýhodou výchozích objektů je nutnost většinu informací přepočítat a dohledat, nic není pohromadě na jednom místě. Například osy, 3DS max používá pravotočivý souřadnicový systém s osou Z nahoru, avšak OpenGL používá levotočivý systém s osou Y nahoru a je nutné všechna data přepočítat. Tuto a některé další nevýhody řeší druhá varianta přístupu k objektům Maxe, a to použít rozhraní IGame, které se pokouší vše nahromadit na jedno míst a čitelně propojit. Díky kompletní konfiguraci si lze přesně říct, jaký souřadnicový systém budeme používat. Jediná věc, kterou IGame neumí vyřešit, je přepočítání jednotek, tzn. aby 1 metr ve scéně v modeláři byla jedna jednotka. Konstantu, kterou všechny hodnoty při exportu dělím a která mi požadovaný přepočet zařídí jsem získal exportem jednotkové krychle v metrickém systému.

Pluginy v 3DS Max jsou potomky obecného objektu a jsou identifikovány unikátním class id, které se generuje pomoci externího softwaru genuid v SDK Maxe. Class id se využívá pro identifikaci objektu v rámci komunikace mezi pluginy, využíval jsem jej pro identifikaci materiálů Midas Materiál a objektů MaxEditu.

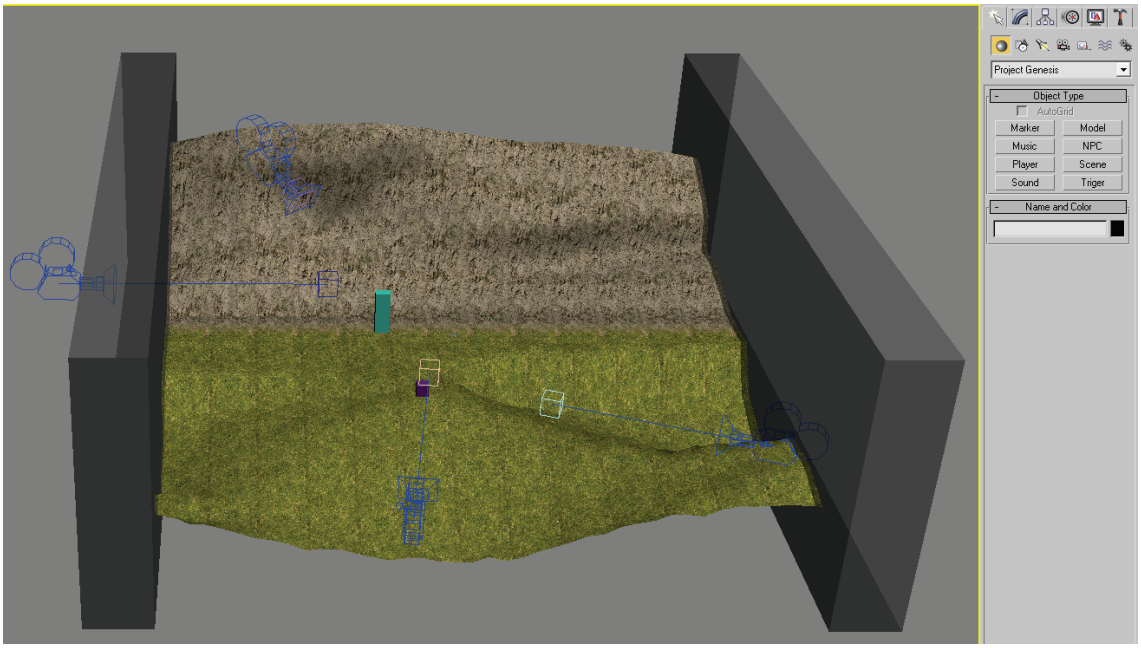


Obrázek 6.10: Plugin SMF Exporter a GLF exporter v 3DS Max.

6.2.3 MaxEdit (3DS Max plugin)

Plugin MaxEdit představuje prozatímní náhradu Editoru scén, který má vyjít z bakalářské práce "Genesis - Level Editor". Plugin krom kamery a světla, kde používá objekty modeláře, přidává objekt Marker, Model, Music, NPC, Player, Scene, Sound a Triger.

Objekt Marker představuje pomocnou pozici v prostoru, kterou můžeme využívat například při skriptování k získání informací o prostoru, například pro targetování kamery. Model je reprezentace SMF modelu, který na určenou pozici doplní herní jádro. NPC a Player jsou značky, kde se příslušný herní charakter nebo hráč může objevit. Triger je neviditelná obalová geometrie, která se používá dvěma způsoby. První je jako neviditelný kolizní objekt a druhý způsob je jako virtuální spušťeč, který při událostech onCollision (vniknutí do objektu), onCollisionOut (východ z objektu) a onLoop (pobyt v objektu) generuje příslušné události v každém updatu scény. Ukázka rozhraní je k nahlédnutí na obrázku 6.11.



Obrázek 6.11: Ukázka prostředí pluginu MaxEdit pro 3DS Max.

6.3 Testování

Vzhledem k faktu, že v době psaní této práce není do enginu implementována většina optimalizačních technologií, využiji provedené testování pro budoucí srovnání rychlosti aplikace. Aplikace je testována na notebooku s procesorem Intel Core 2 Duo 2.33GHz, grafickou kartou NVidia Quatro FX 1500M 512MB a pamětí 4GB-RAM. Při rozlišení 1920x1200 běží aplikace v průměrných 13 fps při scéně s půl milionem polygonů. Při rozlišení 800x600 stoupne fps na přijatelných 25. Pokud do vykreslovací pipeline posílám scénu se 100 tisíci polygony, při plném rozlišení aplikace běží v maximální obnovovací frekvenci monitoru, tj. v 60fps.

SMF plugin optimalizuje při exportu geometrii tak, že odstraní veškeré duplicity[1]. Výsledná rychlost exportu je závislá na počtu duplicit ve scéně. Model o velikosti 10 tisíc polygonů bez duplicit se exportuje 1.5s. Stejný model s duplicitami se exportuje 5 sekund.

[1] Dva vrcholy jsou duplicitní, pokud mají stejnou pozici, normálu, binormálu, tangentu a texturovací souřadnice

Kapitola 7

Závěr

Za dobu, kterou jsem této práci věnoval jsem zvládl nastudovat většinou technologií používaných v dnešních hrách a část jsem implementoval, ať už do prototypů, nebo do výsledné implementace. Fakt, že během příprav vznikly dva funkční prototypy byl pro mě velmi přínosný, protože jsem si mohl vyzkoušet záludnosti některých algoritmů a mohl jsem při další iteraci vývoje vypracovat koncept, založený na předešlích zkušenostech. Čas, který jsem věnoval této práci řádově přesahuje 1500 hodin. Další čas, kolem 500 hodin zabrali další aktivity spojené z projektem - snímání Motion Capture dat v Institutu intermédií, týmové schůzky, organizace obsahu na týmové Wikipedii a SVN, domluva a řízení extenistů a hlavně návrh hry, na kterém jsme společně pracovali.

V době dokončení této práce existuje spustitelná, bohužel nedokončená verze herního jádra. Velkou část vlastností si ale lze vyzkoušet v prototypech, které jsou funkční. Na přiloženém CD lze najít Zdrojové kódy, instalační spustitelné soubory pro jednotlivé prototypy i finální implementaci a dokumentaci, generovanou systémem Doxygen.

Na herním jádru budu pracovat i po ukončení bakalářské etapy studia a rád bych jej nejpozději v magisterském studiu dovedl do zdárného a plně funkčního a optimalizovaného celku. Výkonostní test, které jsem uvedl na straně 38 v budoucnu poslouží pro porovnání jádra bez optimalizačních technologií a s nima. V další fázy implementace se dokončí graf scény, kompletní propojení se skriptovacím jazykem Lua, přidá se kolizní systém, práce se zvukem a hudbou a částicový systém se základní fyzikou.

Literatura

[1] Lwjgl - lightweight java game library. http://www.lwjgl.org.

[2] J. Bittner. Homepage. http://www.cgg.cvut.cz/members/bittner.

[3] M. Corporation. Xna developer center. http://msdn.microsoft.com/en-us/xna/default.aspx.

[4] D. S.-C. Dalmau. *Core Techniques and Algorithms in Game Programming*, volume 1. New Riders Publishing, 201 West 103rd Street, Indianapolis, U.S., 1th edition, 2003.

[5] N. Dog. Naughty dog. http://www.naughtydog.com.

[6] I. P. Fletcher Dunn. *3D Math Primer for Graphics and Game Development*, volume 1. Jones & Bartlett Publishers, 40 Tall Pine Drive, Sudbury, U.S., 1th edition, 2002.

[7] T. Games. Totally games. http://www.totallygames.com.

[8] id Software. id software. http://www.idsoftware.com.

[9] A. Inc. Autodesk softimage. http://www.softimage.com/.

[10] B. Interactive. Bohemia interactive. http://www.bistudio.com.

[11] L. Kavan. Homepage. http://www.jarmilakavanova.cz/ladislav.

[12] Novell. Mono - open source .net development framework. http://mono-project.com.

[13] A. Sherrod. *Ultimate 3D Game Engine Design & Architecture*, volume 1. Charles River Media, 25 Thomson Place, Boston, U.S., 1th edition, 2007.

[14] T.-T. I. Software. 2k czech. http://www.2kczech.com.

[15] T.-T. I. Software. Annual report - purchased 2k czech, formerly known as illusion softworks. http://ir.take2games.com/secfiling.cfm?filingID=1047469-08-13277.

[16] A. G. s.r.o. A|w graph. http://www.awgraph.cz.

[17] S. St-Laurent. *Shaders for Game Programmers and Artist*, volume 1. Thomson Course Technology PTR, 25 Thomson Place, Boston, U.S., 1th edition, 2004.

[18] M. Studios. Mindware studios. http://www.mindwarestudios.com.

[19] D. B. Tom McReynolds. *Advanced Graphics Programming Using OpenGL*, volume 1. Morgan Kaufmann Publishers, 500 Sansome Street, San Francisco, U.S., 1th edition, 2005.

[20] Vicon. Motion capture systems from vicon. `http://www.vicon.com`.

[21] S. Zerbst. *3D Game Engine Programming*, volume 1. Thomson Course Technology PTR, 25 Thomson Place, Boston, U.S., 1th edition, 2004.

Dodatek A

Scénář hry

A.1 Intro

Úvodem do příběhu je krátké intro. Jde o předzvěst velké katastrofy, před kterou hlavní hrdina Jonathan uprchne, díky své potkanici Cori, náhodně otevřeným portálem. Po prostoupení portálem se Jonathan přenese ze svého světa do cizího neznámého světa, což zatím netuší.

A.2 Blind Forest - tutoriál

Po několika hodinách se Jonathan probouzí společně se svou ochočenou potkanicí vprostřed neznámého lesa. Potkanice dotírá a snaží se ho probudit. Jonathan pomalu procitá. Zatím netuší kde je, jak se sem dostal ani co se s ním stalo. Je zmatený.

Myšlenky: Zde se ozve několik trefných hlášek na téma „Kde to jsem?“ „Co se to děje?“

Jonathan se pomalu zvedá ze země a rozhlíží se kolem (TUTORIAL ovládání kamery) Po krátkém rozkoukání si všimne vyšlapané cesty.

Myšlenky: Postava upozorní na vyšlapanou pěšinu a navrhne vydat se po ní.

Jonathan se vydá po právě nalezené cestě ke studánce. (TUTORIAL chození) Na půl cesty Jonathanovi přeletí okolo hlavy něco na způsob netopýra. Jonathan upadne vyděšeně na zem. Když se rozhlédne uvidí, že několik netopýrů letí přímo na něj. Jonathan začne (v sedět) ustupovat a rukou šátrat po nějaké zbrani. Postava sebere klacek a odpálí prvního netopýra, začne souboj s třema dalšíma (TUTORIAL boje).

A.3 Blind Forest - studánka

U studánky si vyprahlý Jonathan vzpomene, že má žízeň a chce se trochu opláchnout.

Myšlenky: Postava upozorní na studánku a pronese nějakou vtipnou hlášku o studánkách uprostřed lesa, pouštních oázách a fatamorgáně...

Jonathan jde ke studánce a nabírá první doušek vody. Místo vody se mu ale v dlani zformuje krystal. Chvíli ho zkoumá...

Jonathan si krystal pověsí na krk, na řetízek který nosí. (Krystal má v sobě přirozený otvor, kterým lze šňurku prostrčit.) Nyní se může konečně opláchnout a napít.

U studánky je zapíchnutá směrovka (pokud ji hráč zkusí číst bez krystalu, nepochodí). Dá se přečíst až s krystalem. Jediná šipka ukazuje "K rozcestí". Hráč tak dovede Jonathana až k prvnímu rozcestí...

A.4 Rozcestí

Jonathan dojde na rozcestí. Zde stojí další směrovky. Jedna ukazuje zpět ke studánce, druhá do nedaleké vesnice Shade (možná je vidět na obzoru), třetí směrovka ukazuje k městu Nápis na třetí směrovce je přeškrtnutý a pod ním je vyritá značka nomádů. Postava se může volně rozhodnout, kterým směrem se vidá. Vesnici může dokonce úplně minout.

A.5 Vesnice Shade

Jonathan přichází do vesnice. Je den a po vsi se potulují její obyvatelé. Hned první budovou, kterou hráč míjí je velký statek.

A.6 Vesnice Shade - statek

Na dvoře statku postává starý sedlák. Už z dálky je slyšet jak nadává na svého ztraceného čeledína. Jakmile se k němu Jonathan přiblíží, začne dialog.

Dialog: Sedlákovi se ztratil čeledín. Nejspíš se někam zašil, protože je líný pracovat. Sedlák Jonathana žádá jestli by nemohl jeho práci zastat, potřebuje urychleně obrátit seno, jinak shnije. Sedlák je starý a na podobnou práci už nestačí.

Quest - Hledání čeledína: Jonathan může nabídnout pomoc s hledáním čeledína. Sedlák mu pak prozradí několik bonusových informací o čeledínovi a řekne mu, že se nejspíš zašívá někde ve vesnici. Tenhle quest má spojitost s mrtvolou na konci louky, právě to je sedlákův čeledín. Pokud hráč mrtvolu prohledá, měl by najít něco z čeho si odvodí o koho jde a sedlákovi dát vědět co se stalo.

Jonathan se může vyptávat na telefon a další podivnosti, ale starému sedlákovi to nic neříká. Sedlák může prozradit jméno vesnice, nasměrovat Jonathana k nomádům a do města. Dá se zeptat i na nocleh. Statkář ale odmítá nechat přespat někoho, koho vůbec nezná.

Jonathan může příjmout práci a vydělat si tak první peníze, nebo i něco jiného. Statkář dá Jonathanovi kosu a pošle ho na pole. Na poli proběhne minihra práce na poli.

Minihra - Práce na poli: Za splnění úkolu nabídne statkář menší peníz, nebo čerstvě zabitého králíka.

A.7 Vesnice Shade - vetešnictví

Vetešník má malý obchůdek naproti kovárně. Je to starší fousatý pán, který vykupuje a rozprodává všechno co může. Jeho dům je plný harampádí. Vesničanům prodává jídlo,

šaty, nástroje a vůbec všechno co neobstará kovář nebo vlastní výroba. Zvláštní zálibu našel vetešník ve starých a podivných věcech. Je za ně ochotný nabídnout poměrně příznivou cenu, takže u něj hráč z počátku bude moct výhodně prodávat veškeré podivnosti, které se mu nebudou hodit.

Dialog: S vetešníkem se dá především obchodovat. Jonathan se může vyptávat na telefon a na to kde se octl. Vetešník mu udělá přednášku o podivných mocných předmětech, kterým se říká artefakty. Mají prý moc o které se nikomu nesnilo. Vetešník by takovéto předměty rád vykoupil.

Vetešník znáširoké okolí díky tomu, že cestuje. Může hráči povědět větší podrobnosti než kdokoli ostatní. Upozorní ho na lesní bandity a poví o několika dalších okolních lokací, taky nabídne kontakt na několik lidí v přístavním městě Lipber.

Quest - Nový dodavatel: Vetešník už hledá nové dodavatele a odbytiště pro svůj obchod. Nabídne dobrou cenu za nějaký ten typ.

Quest - Výkup artefaktů (Stálý quest): Jonathan může vetešníkovi nabídnout některé ze svých věcí (třeba hodinky) Vetešník je vykoupí. Navíc nabídne za jakoukoli podivnost, kterou nikdo jiný nevykoupí dobrou cenu.

A.8 Vesnice Shade - kovárna

Kovárna se nachází na kraji vesnice. Pracují v ní dva kováři – mistr a učeň. Veškeré svoje věci přímo prodávají zájemcům. Oba kováři jsou spíš mlčenlivý, mohli by se nějak vzájemně doplňovat.

Dialog: Na vyptávání kováři odpovídají ve smyslu, že to není jejich starost atd.

S kováři je možné obchodovat. Prodávají základní zbraně jako dýky a nože.

A.9 Vesnice Shade - ostatní

Na náměstíčku se prochází několik dalších lidí. Jonathan se jich může ptát na základní otázky ohledně telefonu, místa, kde se nachází... Dozví se jen minimum, rozhodně nic víc než od hlavních postav vesnice.

A.10 Cesta přes louku

Několik kroků za rozcestím se začíná stmívat. Postava pokračuje po cestě a nebe se pozvolna zatemňuje (časovaný trigger na počasí). Padá noc a začíná pršet (Mění se celkový nádech atmosféry, ale i hudba a zvuky) Asi vprostřed louky se potuluje smečka divokých zvířat (max 5). Zvířata zatím o příchozího nejeví zájem, ale jakmile se přiblíží, měli by zaútočit.

Souboj: Příšery se jmenují Nibblies (okusovači), jsou to poměrně zbabělí tvorové útočící pouze v přesile. Nejčastěji se živí jako mrchožrouti. Jonathan Nibblies vyrušil a ty na něj vzápětí zaútočí. Jejich morálka je slabá, po několika úspěšných zásazích zbaběle utečou pryč. V místě, kde se smečka potulovala leží mrtvola. Pokud Jonathan mrtvolu prozkoumá, najde

čeledínovu píšťalu. Pokud má info od statkáře, dokáže mrtvolu identifikovat. Od mrtvoly už je jen pár kroků k říčce, která se dá přejít po úzkém mostě.

A.11 Nomádské tábořiště

Jakmile Jonathan překročí řeku uvidí před sebou na obzoru plápolat oheň. Je to nomádský tábor o kterém se hráč mohl dozvědět už ve vesnici. (na pár sekund se kamera zafixuje na pohled na tábořiště, abychom hráče správně navedli. Postava může i něco prohodit.)

Jonathan je vyčerpaný, tma v níž pořádně nevidí mu taky není dvakrát příjemná. Pokud se hráč rozhodne za nomády nejít, postava začne postupně připomínat, že je ospalá, později nadávat, možná i kýchat. Pokud hráč postavu dovede příliš daleko spustí se tma jako v pytly ve které nebude vidět vůbec nic a budou se ozývat děsivé zvuky, takže postava bude odmítat jít dál.

Tábor představuje několik chatrčí sestavených kolem malého jezírka. V přítmí chladné a sychravé noci se nomádi tetelí kolem ohně. Jeden z nich pohrává na píšťalu smutnou písničku. Ostatní posedávají, postávají, možná popíjejí a baví se.

A.12 Nomádské tábořiště - Rozhovor s Baronem

U ohně postává i baron. Mělo by být na první pohled poznat, že jde o důležitou postavu. Jednak bude lépe oblečený a jednak by měl stát osamoceně někde v „čele“. Pokud Jonathan barona neosloví, měl by si ho baron jakožto neznámého cizince vyzvat sám.

Dialog: Baron cizince přivítá, tak jak se u nomádů sluší a zeptá se co požaduje. Jonathan se opět může ptát na obecné otázky ohledně telefonu, toho kde se nachází, jak se sem dostal a kdo by ho mohl dostat zpět domů. Do těchto hovorů se mohou částečně zapojit i ostatní nomádi. Postava musí barona nutně požádat o nocleh. Tehdy by měl baron znejistět a napoprvé Jonathana odmítnout se slovy, že nomádi jsou sice společenským národem, ale nepřijímají každého cizince na potkání. V průběhu celého dialogu s baronem si Jonathana měří temná postava stojící v pozadí. Dost možná, že si ji hráči ani nevšimnou. Je to věštkyně.

A.13 Nomádské tábořiště - Noční dobrodružství

Odmítnutý Jonathan odchází z tábora. Stále nemá kde přespat. Někde na cestě by ho měl chytit malý kluk (Farcio) s pochodní. Je to jedno z nomádských dětí z tábora. Farciovi se zalíbila potkanice, kterou Jonathan nosi na rameni.

Dialog: Dítě nabídne Jonathanovi pomoc. Poví mu o starém cikánském zvyku: „Tomu kdo přinese něco na hostinu, nemůže být odmítnuta pohostinost“. Pokud má sebou Jonathan králíka, může se vidat rovnou zpět do tábora. Pokud králíka ještě nemá, nezbývá mu než vyrazit s chlapcem na lov.

A.14 Varianta A - Noční lov

Jonathan se s Farciem vydává podél jezera k lesu, kde společně uloví králíka.

A.15 Varianta B - Noční vesnice Shade

V noci je vesnice prázdná. Všichni jsou zalezlí doma a dveře mají zamčené na petlici. Pokud se pokusí klepat, můžou se ozývat podrážděné hlasy probuzených vesničanů. Selka má na zahrádce králíkárnu, kde se dají krást králíci

Minihra - Krádež králíka

A.16 Nomádské tábořiště - Přijetí

Dialog: Poté co Jonathan sežene králíka, baron už ho nemůže odmítnout, aby nezneuctil nomádské zvyklosti. Příjme ho a nabídne nocleh. Taky mu řekne v kterém stanu bude spát a dá mu možnost doptat se na věci, které ho zajímají. Jonathan se může bavit s nomády. Měli by umět vyprávět nějaké historky, nebo vtipy. Dá se od nich taky vyzvědět spousta informací o okolí. Mimo to může poprvé potkat Káťu. Káťa evidentně není nomádka i když se na oko snaží zapadnout.

Dialog: Káťa Jonathanovi poví nějakou vymyšlenou historku, nejlépe o tom jak jí opustil přítel a jak se bezmocná musela přidat k nomádům. Taky řekne, že dřív bydlela ve městě a může mu povědět pár bližších informací. Unavený Jonathan nakonec zamíří ke stanu. Ještě před vstupem ho zastihne věštkyně.

A.17 Nomádské tábořiště - Slova vědmy

Sibyla zavede Jonathana do své chýše. Je to malá kruhová místnůstka. Vprostřed kulatý stůl okolo něj dvě židle a na něm křišťálová koule. Po straně pokoje je něco na způsob kuchyňské linky, na ní leží rozházené králičí vnitřnosti. U druhé stěny je křeslo na kterém vědma spává. Celou místnost ozařují dvě oranžové svíčky po stranách stolu (tam kde nejsou židle) Vědma rozhrne hromádku vnitřností a na chvíli se do nich zakouká, potom se posadí na jednu z židlí a čeká až se usadí i Jonathan.

Věštkyně odvypráví Jonathanovi příběh (FMV Sekvence - Proroctví).

Dialog: Jonathan bude mít nejspíš spoustu otázek. Vědma je jediná, kdo může Jonathana nasměrovat, postava jí tedy musí povědět o svých problémech a o tom, že sem nepatří (Ne že by to vědma už dávno nevěděla). Sibyla mu naoplátku prozradí, že se má shánět po Salderovi, který by měl brzo dorazit do města na slavnosti. Hráč se nejspíš bude chtít zeptat jakéže poselství z jejího příběhu vlastně plyne. Vědma mu poví, že to pozná v praví čas... Stejně tajemné budou i odpovědi na dotazy ohledně vědmy samotné a ohledně krystalu, který Jonathan Sibyle na závěr ukáže. Cestou ze stanu bude Jonathan schovávat svůj krystal, který z pozadí zahlídne Káťa. Noc končí tím, že se Jonathan podruhé přiblíží ke svému stanu.

A.18 Nomádské tábořiště - Slova vědmy

Ráno je čas vyrazit do města. Většina nomádů už v táboře není. Jonathan se může poptat na události včerejšího večera, může se i vrátit zpět do vesnice. Pak by měl vyrazit dál směrem k městu.

Jakmile vykročí směrem do města, objeví se Káťa, která se po krátkém dialogu přidává do party.

Dodatek B

Struktura Design dokumentu

- Koncept dokument
- Příběh
- Příběhový diagram
- Knihovna
  - Frakce
    - Vodní nájezdníci
    - Odvrácení
    - Salderova legie
  - Historie světa Cipra
- Charaktery
  - Hlavní příběhové charaktery
    - Jonathan Seth
    - Cori
    - Sibyla (věštkyně)
    - Farcio
    - Kateřina Tylerová
    - Lord Salder
    - Mofis
  - Příběhové charaktery
    - Starý Sedlák
    - Vetešník
    - Kovářský mistr
    - Nomádský baron
    - Elli Seth
    - Daniel Seth

  - Ostatní charaktery
    * Mrtvý Čeledín
    * Lesní Banditi
    * Kovářský učeň
    * Vesničani
    * Nomádi
    * Selka
- Monstra
  - Temný Netopýr
  - Nibblies
  - Králík
- Lokace
  - Technical World
  - World of Cipra (Spiritual World)
  - Blind Forest
  - Vesnice Shade
  - Nomádské tábořiště
  - Lipber
- Herní předměty
- Minihry
  - 1. Práce na poli
  - 2. Krádež králíka
- Questy
  - 1. Vesnice Shade - Hledání čeledína
  - 2. Vesnice Shade - Nový dodavatel
  - 3. Vesnice Shade - Výkup artefaktů
- FMV Sekvence
  - Intro
  - Proroctví
- Titulky

Dodatek C

Specifikace datových formátů

C.1 SMF - Static Model File

Velikost (`byte`)	**Název**	**Popis**
4 (`int`)	počet znaků hlavičky	N_h = počet znaků hlavičkového textu
$N_h * 1$ (`char`)	hlavička	Hlavička obsahující verzi souboru o velikosti N_h znaků
4 (`int`)	počet objektů	N_o = Počet geometrických sad, které soubor obsahuje
4 (`int`)	počet koster	B_o = Počet kosterních sad, které soubor obsahuje
-	**OBJECTS**	Postupně za sebou N_o objektů
4 (`int`)	počet znaků patičky	N_p = počet znaků patičkového textur
$N_p * 1$ (`char`)	patička	Patička obsahující copyright o velikosti N_p znaků

Tabulka C.1: Formát souboru SMF

Velikost (`byte`)	**Název**	**Popis**
4 (`int`)	počet znaků jména	N_j = počet znaků jména geometrické sady
$N_j * 1$ (`char`)	jméno	Jméno dané sady geometrie o velikosti N_j znaků
4 (`int`)	počet vrcholů	N_v = Počet vrcholů v jedné sadě geometrie
4 (`int`)	počet indicií	N_i = Počet indicií v jedné sadě geometrie
-	**VECTORS**	Pole vrcholů o velikosti N_v
-	**TRIANGLES**	Indicie v poly o velikosti N_i, tj. $N_i/3$ trojuhelníků
-	**NORMALS**	Pole normál o velikosti Nv, tj. jedna normála na vrchol.
-	**TEXTURE COORDINATES**	Pole koordinátů pro textury o velikosti Nv, tj. jedna sada koordinátů na vrchol.
4 (`float`)	r - diffuse	Materiál, červená složka odrazivosti diffuzního světla
4 (`float`)	g - diffuse	Materiál, zelená složka odrazivosti diffuzního světla
4 (`float`)	b - diffuse	Materiál, modrá složka odrazivosti diffuzního světla
4 (`float`)	r - ambient	Materiál, červená složka odrazivosti ambientního světla
4 (`float`)	g - ambient	Materiál, zelená složka odrazivosti ambientního světla
4 (`float`)	b - ambient	Materiál, modrá složka odrazivosti ambientního světla
4 (`float`)	r - specular	Materiál, červená zrcadlová složka
4 (`float`)	g - specular	Materiál, zelená zrcadlová složka
4 (`float`)	b - specular	Materiál, modrá zrcadlová složka
4 (`float`)	glossiness	Materiál, lesklost
4 (`int`)	počet znaků diffuzní textury	N_{td} = počet znaků názvu souboru diffuzní textury + ukončovací znak. Pokud $N_{td} == 0$, textura neexistuje.
$N_{td} * 1$ (`char`)	diffuzní textura	Název souboru s diffuzní texturou velikosti N_{td} znaků
4 (`int`)	počet znaků specular textury	N_ts = počet znaků názvu souboru specular textury + ukončovací znak. Pokud $N_{ts} == 0$, textura neexistuje.
$N_{ts} * 1$ (`char`)	specular textura	Název souboru se specular texturou velikosti N_{ts} znaků
4 (`int`)	počet znaků normal textury	N_{tn} = počet znaků názvu souboru normal textury + ukončovací znak. Pokud $N_{tn} == 0$, textura neexistuje.
$N_{tn} * 1$ (`char`)	normal textura	Název souboru s normal texturou velikosti N_{tn} znaků
4 (`int`)	počet znaků height textury	N_{th} = počet znaků názvu souboru height textury + ukončovací znak. Pokud $N_{th} == 0$, textura neexistuje.
$N_{th} * 1$ (`char`)	height textura	Název souboru s height texturou velikosti N_{th} znaků
4 (`int`)	počet znaků shaderu	N_m = počet znaků názvu použitého shaderu
$N_m * 1$ (`char`)	shader	Název souboru použitého shaderu

Tabulka C.2: Formát souboru SMF

C.2 GLF - Genesis Level File

Velikost (`byte`)	**Název**	**Popis**
4 (`int`)	počet znaků hlavičky	N_h = počet znaků hlavičkového textu
$N_h * 1$ (`char`)	hlavička	Hlavička obsahující verzi souboru o velikosti N_h znaků
4 (`int`)	počet materiálů	N_m = počet materiálů
-	**MATERIALS**	Pole materilů o velikosti N_m
-	**NODE**	Kořen stromu scény
4 (`int`)	počet znaků patičky	N_p = počet znaků patičkového textur
$N_p * 1$ (`char`)	patička	Patička obsahující copyright o velikosti Np znaků

Tabulka C.3: Formát souboru GLF

Velikost (`byte`)	**Název**	**Popis**
-	**COLOR**	Diffuzní barva
-	**COLOR**	Ambientní barva
-	**COLOR**	Specularni barva
4 (`float`)	glossiness	Lesklost
4 (`int`)	počet znaků diffuzní textury	N_d = počet znaků názvu diffuzní textury
N_d*1 (`char`)	diffuzní textura	Název souboru diffuzní textury
4 (`int`)	počet znaků specularní textury	N_s = počet znaků názvu speculární textury
$N_s * 1$ (`char`)	speculární textura	Název souboru speculární textury
4 (`int`)	počet znaků normálové textury	N_n = počet znaků názvu normálové textury
$N_n * 1$ (`char`)	normálová textura	Název souboru normálové textury
4 (`int`)	počet znaků výškové textury	N_h = počet znaků názvu výškové textury
$N_h * 1$ (`char`)	výškový textura	Název souboru výškové textury
4 (`int`)	počet znaků shaderu	N_m = počet znaků názvu použitého shaderu
$N_m * 1$ (`char`)	shader	Název souboru použitého shaderu

Tabulka C.4: Material

Velikost (`byte`)	**Název**	**Popis**
1 (`unsigned char`)	Typ nody	ID udávající typ nody
4 (`int`)	počet znaků názvu	N_h = počet znaků názvu nody
$N_n * 1$ (`char`)	název	Název aktuální nody
-	**VEKTOR**	Pozice Node
-	**QUATERNION**	Rotace Node
1 (`unsigned char`)	aktivni Node	1 - noda je aktivní, 0 - noda není aktivní
4 (`int`)	Počet potomků	N_c = počet potomku aktuální nody
-	**DATA**	Data Aktuální nody
-	**NODES**	Postupně za sebou Nc nod

Tabulka C.5: Node

Velikost (`byte`)	**Název**	**Popis**
-	**VECTOR**	Směr dopadajícího světla od slunce
-	**COLOR**	Barva dopadajícího světla
4 (`int`)	počet znaků SkyBoxUp	N_u = počet znaků cesty k textuře
$N_u * 1$ (`char`)	textura SkyBoxUp	Cesta k textuře pro SkyBoxUp
4 (`int`)	počet znaků SkyBoxDown	N_d = počet znaků cesty k textuře
$N_d * 1$ (`char`)	textura SkyBoxDown	Cesta k textuře pro SkyBoxDown
4 (`int`)	počet znaků SkyBoxLeft	N_l = počet znaků cesty k textuře
$N_l * 1$ (`char`)	textura SkyBoxLeft	Cesta k textuře pro SkyBoxLeft
4 (`int`)	počet znaků SkyBoxRight	N_r = počet znaků cesty k textuře
$N_r * 1$ (`char`)	textura SkyBoxRight	Cesta k textuře pro SkyBoxRight
4 (`int`)	počet znaků SkyBoxFront	N_f = počet znaků cesty k textuře
$N_f * 1$ (`char`)	textura SkyBoxFront	Cesta k textuře pro SkyBoxFront
4 (`int`)	počet znaků SkyBoxBack	N_b = počet znaků cesty k textuře
$N_b * 1$ (`char`)	textura SkyBoxBack	Cesta k textuře pro SkyBoxBack

Tabulka C.6: DATA ID 0 - Sun

Velikost (`byte`)	**Název**	**Popis**
4 (`int`)	material id	Id materiálu, který geometrie používá
4 (`int`)	počet vrcholů	N_v = Počet vrcholů v jedné sadě geometrie
4 (`int`)	počet indicií	N_i = Počet indicií v jedné sadě geometrie
-	**VERTICES**	Pole vertexů o velikosti N_v
-	**TRIANGLES**	Indicie v poly o velikosti N_i, tj. $Ni/3$ trojuhelníků

Tabulka C.7: DATA ID 1 - Geometry

Velikost (`byte`)	**Název**	**Popis**
4 (`int`)	počet znaků cesty	N_f = počet znaků cesty k modelu
$N_f * 1$ (`char`)	cesta k modelu	Relativní cesta k modelu
4 (`int`)	počet znaků cesty	N_s = počet znaků cesty ke scriptu
$N_s * 1$ (`char`)	cesta ke scriptu	Relativní cesta ke scriptu
1 (`usnigned char`)	spuštěná animace	1 - animace je spuštěná, 0 - animace je vypnutá

Tabulka C.8: DATA ID 2 - Model

Velikost (`byte`)	**Název**	**Popis**
4 (`int`)	počet znaků cesty	N_f = počet znaků cesty k modelu
$N_f * 1$ (`char`)	cesta k modelu	Relativní cesta k modelu
4 (`int`)	počet znaků cesty	N_s = počet znaků cesty ke scriptu
$N_s * 1$ (`char`)	cesta ke scriptu	Relativní cesta ke scriptu

Tabulka C.9: DATA ID 3 - NPC

Velikost (`byte`)	**Název**	**Popis**
4 (`int`)	počet znaků cesty	N_f = počet znaků cesty ke zvuku
$N_f * 1$ (`char`)	cesta ke zvuku	Relativní cesta ke zvuku
1 (`usnigned char`)	opakování	1 - zvuk se do nekonečna opakuje, 0 - zvuk se přehraje pouze jednou
1 (`unsigned char`)	hlasitost	Hlasitost zvuku v procentech 0 - 100

Tabulka C.10: DATA ID 4 - Sound

Velikost (`byte`)	**Název**	**Popis**
4 (`int`)	počet znaků cesty	N_f = počet znaků cesty k hudbě
$N_f * 1$ (`char`)	cesta k hudbe	Relativní cesta k hudbe
1 (`usnigned char`)	opakování	1 - hudba se do nekonečna opakuje, 0 - hudba se přehraje pouze jednou

Tabulka C.11: DATA ID 5 - Music

Velikost (`byte`)	**Název**	**Popis**
-	**COLOR**	Barva světla

Tabulka C.12: DATA ID 6 - Light

Velikost (`byte`)	**Název**	**Popis**
-	**VEKTOR**	LookAt, tj. pozice, na kterou se kamera dívá

Tabulka C.13: DATA ID 7 - Camera

Velikost (`byte`)	**Název**	**Popis**

Tabulka C.14: DATA ID 8 - Player position

Velikost (`byte`)	**Název**	**Popis**

Tabulka C.15: DATA ID 9 - Mark

Velikost (`byte`)	**Název**	**Popis**
1 (`unsigned char`)	typ trigeru	Id udávající typ trigeru
-	**POPIS TĚLESA**	popis tělesa, reprezentující triger podle typu
1 (`usnigned char`)	kolize	1 - Triger je použit jako kolizní objekt, 0 - Triger neni použit jako kolizní objekt
4 (`int`)	počet znaků cesty	N_s = počet znaků cesty ke scriptu
$N_s * 1$ (`char`)	cesta ke scriptu	Relativní cesta ke scriptu onCollision

Tabulka C.16: DATA ID 10 - Trigger

Velikost (`byte`)	**Název**	**Popis**
-	**QUATERNION**	Orientace orientovaného boxu
4 (`float`)	šířka	Velikost orientovaného boxu podél jeho osy x
4 (`float`)	výška	Velikost orientovaného boxu podél jeho osy y
4 (`float`)	tloušťka	Velikost orientovaného boxu podél jeho osy -z

Tabulka C.17: POPIS TĚLESA ID 0 - Typ OBB

Velikost (`byte`)	**Název**	**Popis**
4 (`float`)	šířka	Velikost osově zarovnaného boxu podél osy x
4 (`float`)	výška	Velikost osově zarovnaného boxu podél osy y
4 (`float`)	tloušťka	Velikost osově zarovnaného boxu podél osy -z

Tabulka C.18: POPIS TĚLESA ID 1 - Typ AABB

Velikost (`byte`)	**Název**	**Popis**
4 (`float`)	poloměr	Poloměr koule

Tabulka C.19: POPIS TĚLESA ID 2 - Typ Koule (Sphere)

Velikost (`byte`)	**Název**	**Popis**
-	**VECTOR**	Pozice vrcholu
-	**VECTOR**	Normála vrcholu
-	**VECTOR**	Binormála vrcholu
-	**VECTOR**	Tangenta vrcholu
-	**TEXTURE COORDINATE**	Texturové souřadnice vrcholu

Tabulka C.20: VERTEX

Velikost (`byte`)	**Název**	**Popis**
4 (`float`)	x	X-ová složka vektoru
4 (`float`)	y	Y-ová složka vektoru
4 (`float`)	z	Z-ová složka vektoru

Tabulka C.21: VECTOR

Velikost (`byte`)	**Název**	**Popis**
4 (`float`)	r	červená barevná složka, hodnota 0.0f - 1.0f
4 (`float`)	g	zelená barevná složka, hodnota 0.0f - 1.0f
4 (`float`)	b	modrá barevná složka, hodnota 0.0f - 1.0f

Tabulka C.22: COLOR

Velikost (`byte`)	**Název**	**Popis**
4 (`uint`)	i1	První vrchol trojuhelníků
4 (`uint`)	i2	Druhý vrchol trojuhelníků
4 (`uint`)	i3	Třetí vrchol trojuhelníků

Tabulka C.23: TRIANGLE

Velikost (`byte`)	**Název**	**Popis**
4 (`float`)	u	U složka texturového koordinátu
4 (`float`)	v	V složka texturového koordinátu

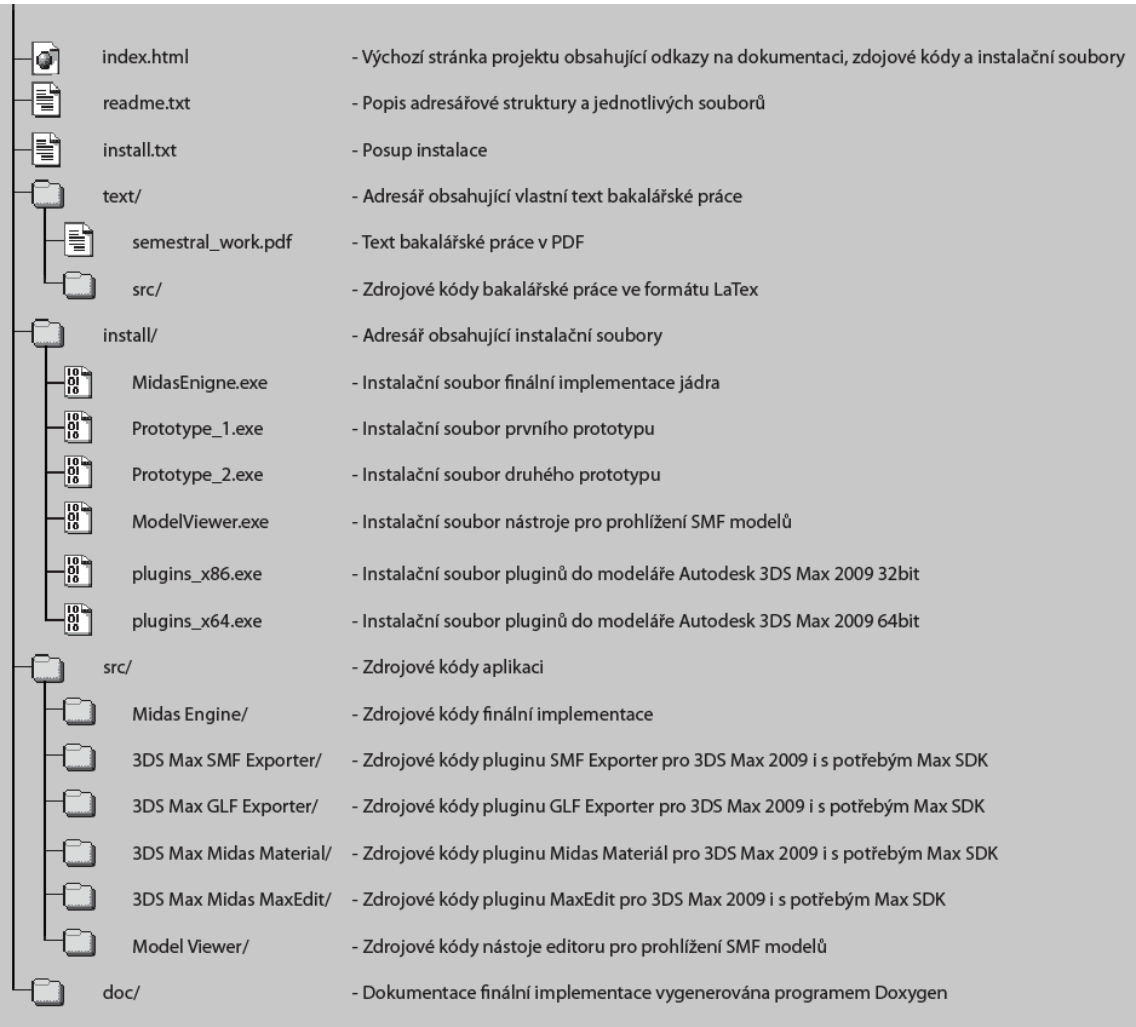
Tabulka C.24: TEXTURE COORDINATE

Velikost (`byte`)	**Název**	**Popis**
4 (`float`)	x	x-ová složka quaternionu
4 (`float`)	y	y-ová složka quaternionu
4 (`float`)	z	z-ová složka quaternionu
4 (`float`)	w	w-ová složka quaternionu

Tabulka C.25: QUATERNION

Dodatek D

Obsah přiloženého CD



Obrázek D.1: Adresářová struktura přiloženého CD