Ogre engine a jeho využití

Ogre engine and its use

Eva Petřvalská

Bakalářská práce
2011

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2010/2011

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Eva PETŘVALSKÁ**
Osobní číslo:	**A07084**
Studijní program:	**B 3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Téma práce:	**Ogre engine a jeho využití**

Zásady pro vypracování:

1. **Seznamte se s grafickým open source enginem Ogre, založeným na technologiích DirectX a OpenGL.**
2. **Tento engine popište v jeho aktuální verzi a shrňte jeho vlastnosti.**
3. **Porovnejte Ogre s jinými grafickými enginy.**
4. **Navrhněte 3D aplikaci za použití grafického enginu Ogre.**
5. **Takto navrženou aplikaci realizujte za pomoci programovacího jazyka C++.**
6. **Vytvořte podrobnou elektronickou příručku pro práci s tímto enginem určenou začínajícím programátorům.**

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **KERGER, Felix. Ogre 3D 1.7 Beginners Guide. Birmingham : Published by Packt Publishing Ltd., 2010. 300 s. ISBN 978-1-849512-48-0.**
2. **JUNKER, Gregory. Pro OGRE 3D Programming. New York : APRESS ACADEMIC, 2006. 312 s. ISBN 978-1-59059-710-1.**
3. **PRATA, Stephen. Mistrovství v C++ 3. vydání. Praha : Computer Press, 2007. 1120 s. ISBN 978-80-251-1749-1.**
4. **KRUGLINSKI, David J. Mistrovství ve Visual C++. Praha : Computer Press, 1999. 854 s. ISBN 80?7226?132?0.**
5. **Ogre [online]. 2010 [cit. 2011-01-27]. OGRE Manual v1.7. Dostupné z WWW: [http://www.ogre3d.org/docs/manual/].**
6. **Ogre [online]. 2010 [cit. 2011-01-27]. OGRE Wiki. Dostupné z WWW: [http://www.ogre3d.org/tikiwiki/tiki-index.php].**

Vedoucí bakalářské práce: **Ing. Pavel Pokorný, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **25. února 2011**

Termín odevzdání bakalářské práce: **7. června 2011**

Ve Zlíně dne 25. února 2011

L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

ABSTRAKT

Cílem této bakalářské práce je podrobně se seznámit s vizualizačním enginem Ogre. Práce je rozdělena do několika částí. Teoretická část je zaměřena na popis základních vlastností, struktury, jednotlivých tříd, funkcí a využití tohoto enginu. Dále obsahuje jeho srovnání s několika dalšími enginy. Praktická část pak zahrnuje vytvoření příručky určené začínajícím uživatelům ogre. Příručka obsahuje několik tutorialů včetně popisu základních funkcí. Další částí této práce je návrh a realizace aplikace založené na tomto enginu. Aplikace je vytvořena v programovacím jazyku C++.

Klíčová slova: Ogre engine, C++, objektově orientované programování, vykreslování, entita

ABSTRACT

The goal of this bachelor thesis is closely execute with the Ogre rendering engine. The thesis is divided into several parts. The theoretical part is aimed at describing the basic characteristics, structures, individual classes, functions and use of this engine. It also contains a comparison with several other engines. The practical part includes a manual for novice users Ogre. This manual includes several tutorials, including a description of basic functions. Another part of this thesis is design and implementation of applications based on this engine. Application is developed in programming language C++.

Keywords: Ogre engine, C++, Object-oriented programming, rendering, entity

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně …….…….

podpis diplomanta

OBSAH

ÚVOD **9**
I TEORETICKÁ ČÁST **10**
1 OGRE ENGINE **11**
1.1 CO JE TO OGRE 11
1.2 HISTORIE 11
1.3 HARDWAROVÉ POŽADAVKY 12
1.4 SOFTWAROVÉ POŽADAVKY 12
1.5 PODPOROVANÉ FORMÁTY 13
1.5.1 Běžné formáty: 13
1.5.2 Specifické formáty pro OGRE: 13
1.6 SPECIÁLNÍ EFEKTY 14
1.7 PROGRAMOVACÍ JAZYK C++ 15
1.7.1 Objektově orientované programování 16
2 STRUKTURA **18**
2.1 OBECNÝ PŘEHLED 18
2.2 JMENNÝ PROSTOR 20
2.3 OBJEKT ROOT 20
2.4 OBJEKTY SPRÁVY SCÉNY 21
2.4.1 SceneManager 21
2.4.2 Pohyblivé objekty scény 21
2.4.2.1 Entita 22
2.4.2.2 Kamera 23
2.4.2.3 Světla 23
2.4.3 SceneNode 24
2.5 OBJEKTY ŘÍZENÍ ZDROJŮ 24
2.5.1 ResourceGroupManager 24
2.6 MESH 25
2.6.1 Materiály 25
2.7 OBJEKTY VYKRESLOVÁNÍ 27
2.7.1 RenderSystem 27
3 DALŠÍ DŮLEŽITÉ OBJEKTY A EFEKTY OGRE **28**
3.1 ANIMACE 28
3.1.1 Skeletální animace 28
3.1.2 Vertexová animace 29
3.1.3 Animace sceneNode 30
3.1.4 Animace numerické hodnoty 30

3.2 STÍNY 30
3.3 TŘÍDA OVERALY 31
3.4 FRAMELISTENER 32
3.5 OBLOHA 32
3.6 MLHA 33
4 PLUGINY A SKRIPTY 35
4.1 SKRIPTY 35
4.1.1 Materiálové Skripty 35
4.1.2 Skripty kompozitoru 35
4.1.3 Částicové skripty 36
4.1.4 Skripty třídy overaly 36
5 POROVNÁNÍ OGRE S JINÝMI ENGINY 37
II PRAKTICKÁ ČÁST 41
6 3D APLIKACE 42
6.1 KOMPONENTY APLIKACE 42
6.1.1 Převod modelů do fomátu .mesh z programu blender 43
6.1.2 Nastavení materiálů 45
6.1.3 Tvorba terénu pro aplikaci 45
6.2 POPIS APLIKACE 46
6.2.1 Základní myšlenka 46
6.2.2 Jednotlivé funkce 46
6.3 VYTVOŘENÍ APLIKACE 47
6.3.1 Scéna 47
6.3.2 Pohyblivý objekt a kamerový systém. 47
6.3.3 Vstupy do aplikace 47
6.3.4 Detekce kolizí 48
6.3.4.1 S povrchem 48
6.3.4.2 Se stromy 49
7 PŘÍRUČKA 50
7.1 ZÁKLADNÍ MYŠLENKA 50
7.2 HIERARCHIE 50
7.2.1 Jednotlivé kapitoly 50
7.3 POPIS KAPITOL 51
ZÁVĚR 53
ZÁVĚR V ANGLIČTINĚ 54
SEZNAM POUŽITÉ LITERATURY 55
SEZNAM OBRÁZKŮ 56
SEZNAM TABULEK 57
SEZNAM PŘÍLOH 58

ÚVOD

Myšlenka, že počítač je možné využít i pro generování obrazu, pravděpodobně napadla již tvůrce prvních elektronických výpočetních systémů. První pokusy o její praktické naplnění totiž sahají až k samotným počátkům výpočetní techniky.

Vývoj specializovaných grafických prostředků počítačů však začal zhruba v první polovině šedesátých let minulého století. V té době se začaly počítače v poměrně velké míře využívat při zpracovávání vědeckých dat. Později pronikla počítačová grafika do filmu a televize, kde se osvědčila jako konkurenceschopná náhrada za tradiční speciální efekty a animační techniky, takže i komerční firmy začaly přispívat k jejímu rozvoji.

Vývoj grafických enginů jde ruku v ruce s vývojem grafických karet, počítačových her a počítačové grafiky všeobecně. První enginy vznikali na konci devadesátých let minulého století při vývoji počítačových her. V té době se ještě nejednalo o pravé 3D grafické enginy, i když svými efekty 3D grafiku velmi věrohodně napodobovali. První opravdové 3D enginy začali vznikat ve druhé polovině posledního desetiletí dvacátého století díky vzniku prvních grafických karet podporujících 3D grafiku. Za posledních 15 let pak toto odvětví počítačové grafiky prošlo obrovským vývojem a v dnešní době je základem každé aplikace pracující s prostorovou grafikou nějaký grafický engine.

Na poli současné počítačové grafiky se vyskytuje nepřeberné množství enginů a i začínající uživatel si mezi nimi může vybrat kvalitní open-source engine pro tvorbu svých aplikací. Všechny již nejsou primárně zaměřeny na komplexní tvorbu her, jak tomu bylo v minulosti. Některé se specializují na konkrétní oblasti tohoto odvětví, jako je fyzika nebo renderování. Se stálým zlepšováním grafických akcelerátotů vzrůstá i kvalita efektů a vznikají stále nové a lepší algoritmy i efekty. Proto je jednou ze základních vlastností enginu možnost jeho neustálého rozšiřování a k jeho vývoji je potřebný tým, který toto zajišťuje.

Cílem mé bakalářské práce bylo seznámit se s enginem OGRE, který patří mezi open-source enginy zaměřující se na vykreslování scény. Dále měla být vytvořena aplikace demonstrující vlastnosti OGRE a v neposlední řadě příručka. Ta má začínající programátory 3D grafiky seznámit s tímto enginem, ukázat jim jeho základní možnosti a díky tutoriálům pomoci ve vývoji vlastních aplikací.

I. TEORETICKÁ ČÁST

1 OGRE ENGINE

1.1 Co je to OGRE

Název OGRE je vlastně zkratkou anglického **O**bject-oriented **G**raphics **R**endering **E**ngine (v překladu objektově orientovaný grafický vykreslovací engine). Již z tohoto názvu vyplývá co vlastně OGRE je - flexibilní, multiplatformní a spolehlivý open-source grafický engine vytvořený pro tvorbu 3D scén. Můžeme říci, že OGRE je knihovna vytvořená v programovacím jazyku C++, která nám pomocí svých tříd umožňuje snadné využití systémových knihoven pro práci s 3D grafikou jako je Direct3D a OpenGL. Její třídy abstrahují všechny detaily použití těchto knihoven a sama knihovna nabízí programátorsky jednoduché rozhraní založené na světě objektů a dalších třídách vysoké úrovně. OGRE samotné je založeno na datové struktuře scene graph a má pluginovou architekturu. K dispozici je uživateli zdarma s licencí MIT. [1]

Obrázek 1 – Logo enginu OGRE

1.2 Historie

Přibližně v roce 1999 si Steve Streeting (známý komunitou OGRE jako sinbad**)** uvědomil, že se jeho projekt určený pro usnadnění ovládání Direct3D stal tak abstraktním, že už nemusí být založený striktně na Direct3D a začal plánovat vytvoření mnohem ambicióznější knihovny nezávislé na API a platformě. 25. 2. 2000 byl projekt zaregistrován a pojmenován OGRE. Postupně bylo vytvářeno jádro programu a další důležité třídy a o 5 let později vydána verze 1.0.0 Azathoth. Nejnovější verze ogre je 1.7.3 Cthugha, avšak v této bakalářské práci je použita verze 1.7.2 Cthugha, která byla při jejím zadávání aktuální. [2]

Obrázek 2 - Původní autor enginu Steve Streeting

1.3 Hardwarové požadavky

OGRE podporuje Direct3D(konkrétně DirectX 9 a 10) a OpenGL. Tento engine běží na velmi široké paletě hardwaru, který podporuje 3D grafiku, výkon se samozřejmě liší podle výkonnosti konkrétního počítače, hlavně grafické karty. Mezi nejnižší možné použitelné grafické karty patří u ATI Radeon 7500 a Radeon 9600, u GeForce GeForce2 nebo Geforce 4 (non-MX). Karty SiS, Intel S3 mohou, ale nemusí být podporovány. Samozřejmostí je použití nejnovějších ovladačů pro konkrétní grafickou kartu. [1] [2]

1.4 Softwarové požadavky

Tvůrci tohoto enginu se snaží o multiplatformnost. To ovšem neznamená, že naprosto nezáleží na operačním systému či programovacím jazyce, ve kterém je aplikace, využívající tento engine, vytvořena. Operační systémy, ve kterých lze OGRE použít jsou Windows (všechny majoritní verze), Linux a Mac OS. Hlavní programovací jazyk je C++, mezi další jazyky, ve kterých má uživatel možnost programovat svou aplikaci, patří python, .net a java, ty ovšem nelze použít s klasickým OGRE ale pouze při použití "obaleného" OGRE, které je k tomuto uzpůsobené. Nevýhodou však je, že tyto projekty nejsou podporované vývojářským týmem OGRE a tudíž za ně neručí. Podporované kompilátory pro C++ jsou Visual studio C++ (verze 2003, 2005, 2008 a 2010) a gcc 4+. [1] [2]

1.5 Podporované formáty

OGRE podporuje velké množství běžných formátů a také formátů pro něj specifické. V následujících podkapitolách je uveden jejich seznam.

1.5.1 Běžné formáty:

- **.bsp** Quake bsp soubor
- **.shader** Quake 3 shader soubor
- **.ttf** True type font soubor
- **.png, .tga, .jpg, .raw, .gif, .dds, .tif** některé z podporovaných souborů obrázků
- **.cg** Cg shader soubor
- **.asm** Assembly shader soubor
- **.zip** archiv zdrojů
- **.xml** XML soubor
- **.xsd** XML schéma souborů, které definují specifika Ogre souborových formátů, které jsou založeny na XML
- **.log** textový výstupní soubor používaný pro protokoly ladění, paměti a výstupy zpráv paměti

1.5.2 Specifické formáty pro OGRE:

- **.material** skripty pro definici komplexních materiálů
- **.compositor** skripty pro definici compositoru
- **.particle** skripty pro definici částicových systémů
- **.overlay** skripty po definici overlay
- **.fontdef** skripty pro definici písem

- **.skeleton** binární soubor obsahující skeleton animací
- **.mesh** binární soubor obsahující jednoduchou definici modelů
- **.scene** soubor založený na XML s mnoha modely dohromady
- **.cfg** konfigurační soubor pro různé účely
- **.inc** šablony oberlay
- **.scheme** soubor XML založený na GUI
- **.imageset** soubor XML založený na GUI
- **.layout** soubor XML založený na GUI
- **.font** soubor XML založený na GUI

[3]

1.6 Speciální efekty

OGRE obsahuje velkou škálu speciálních efektů. Patří mezi ně částicový systém, jež jde pomocí pluginu rozšířit o zářiče. Pro snadné nastavení je umožněna definice těchto systémů pomocí skriptů. Automaticky se využívá sdružování částic pro maximální výkon. Při definici krajiny uživatel jistě ocení jednoduchou implementaci oblohy pomocí jednoho z objektů *skybox*, *skyplane* nebo *skydome*. Všechny typy se velmi jednoduše používají. Další bezesporu užitečnou funkcí je systém objektů *overlay*. Díky němu lze vytvořit HUD (heads-up display) a menu v nichž jsou použity 2D a3D objekty. Mezi další efekty patří mlha a různé její modifikace vytvořené nastavením tohoto objektu v programu. OGRE také podporuje post-processing efekty. [3]

Obrázek 3 – Ukázka oblohy a overlay

1.7 Programovací jazyk C++

V tomto jazyce je celý engine vytvořen a také se v něm obvykle vytváří aplikace na OGRE založené. **C++** je objektově orientovaný programovací jazyk, který vyvinul Bjarne Stroustrup a další v Bellových laboratořích AT&T rozšířením jazyka C. C++ podporuje několik programovacích stylů (paradigmat) jako je procedurální programování, objektově orientované programování a generické programování, není tedy jazykem čistě objektovým. V současné době patří C++ mezi nejrozšířenější programovací jazyky.

Starší verze jazyka, společně označované jako „C with Classes“ (česky C s třídami), byly používány od roku 1980. Jméno „C++“ vymyslel Rick Mascitti v létě 1983. Toto jméno zdůrazňuje evoluční povahu změn oproti jazyku C; „++“ je totiž operátor inkrementace v C. Poněkud kratší jméno „C+“ je syntaktická chyba, bylo též použito jako jméno jiného nesouvisejícího jazyka.

Přestože byl jazyk vyvíjen již od počátku 80. let, první oficiální norma C++ byla přijata v roce 1998, další v roce 2003 (INCITS/ISO/IEC 14882-2003). V roce 2006 a 2007 byly přijaty některé aktualizace.

1.7.1 Objektově orientované programování

Objektově orientované programování (zkracováno na *OOP*, z anglického Object-oriented programming) je metodika vývoje softwaru, založená na následujících myšlenkách, koncepci:

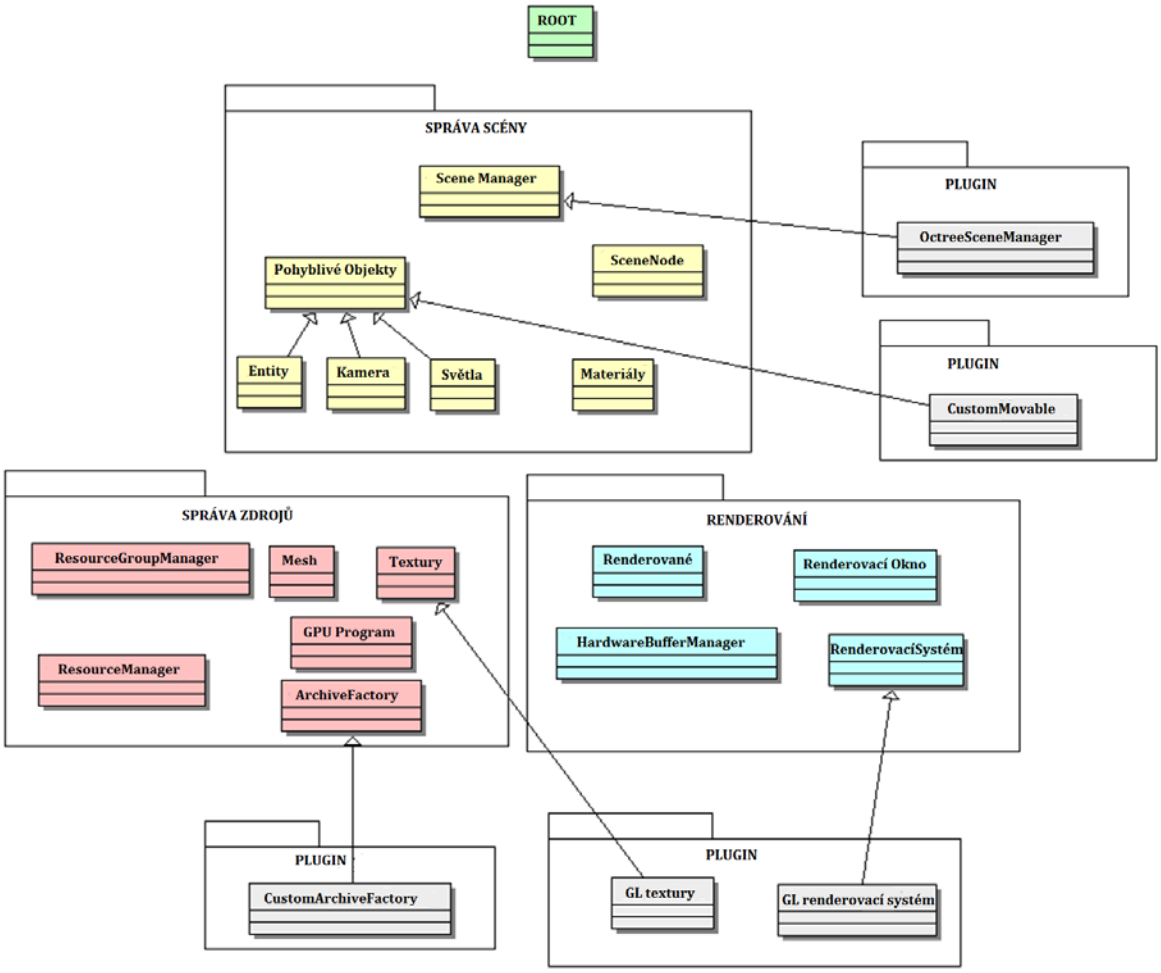
- *Objekty* – jednotlivé prvky modelované reality (jak data, tak související funkčnost) jsou v programu seskupeny do entit, nazývaných objekty. Objekty si pamatují svůj stav a navenek poskytují operace (přístupné jako metody pro volání).
- *Abstrakce* – programátor, potažmo program, který vytváří, může abstrahovat od některých detailů práce jednotlivých objektů. Každý objekt pracuje jako černá skříňka, která dokáže provádět určené činnosti a komunikovat s okolím, aniž by vyžadovala znalost způsobu, kterým vnitřně pracuje.
- *Zapouzdření* – zaručuje, že objekt nemůže přímo přistupovat k „vnitřnostem“ jiných objektů, což by mohlo vést k nekonzistenci. Každý objekt navenek zpřístupňuje rozhraní, pomocí kterého (a nijak jinak) se s objektem pracuje.
- *Skládání* – Objekt může obsahovat jiné objekty.
- *Delegování* – Objekt může využívat služeb jiných objektů tak, že je požádá o provedení operace.
- *Dědičnost* – objekty jsou organizovány stromovým způsobem, kdy objekty nějakého druhu mohou dědit z jiného druhu objektů, čímž přebírají jejich schopnosti, ke kterým pouze přidávají svoje vlastní rozšíření. Tato myšlenka se obvykle implementuje pomocí rozdělení objektů do tříd, přičemž každý objekt je instancí nějaké třídy. Každá třída pak může dědit od jiné třídy (v některých programovacích jazycích i z několika jiných tříd).
- *Polymorfismus* – odkazovaný objekt se chová podle toho, jaké třídy je instancí. Pokud několik objektů poskytuje stejné rozhraní, pracuje se s nimi stejným způsobem, ale jejich konkrétní chování se liší podle implementace. U polymorfismu podmíněného dědičností to znamená, že na místo, kde je očekávána instance nějaké třídy, můžeme dosadit i instanci libovolné její podtřídy, neboť rozhraní třídy je podmnožinou rozhraní podtřídy. U polymorfismu nepodmíněného

dědičností je dostačující, jestliže se rozhraní (nebo jejich požadované části) u různých tříd shodují, pak jsou vzájemně polymorfní. [6] [7]

2 STRUKTURA

2.1 Obecný přehled

Na obrázku číslo 3 je uvedeno schéma hlavních objektů OGRE a jejich konkrétního uspořádání. Nejsou zde všechny třídy, které obsahuje, ale pouze několik konkrétních nejvýznamnějších tříd.



Obrázek 4 – Struktura OGRE

Na vrcholu schématu je základní objekt (*Root*). Ten je ‘cestou’ do systému OGRE a místem, v němž lze vytvořit objekty vyšších úrovní, které je potřeba sdílet, jako jsou *SceneManager*, *RenderSystem*, *RenderWindow* a jiné základní objekty.

Většina zbylých tříd OGRE spadá do jedné ze 3 funkcí:

a) Řízení scény

Třídy této kategorie uchovávají obsah scény, informaci o její struktuře, jak je vidět z kamer, atd. Objekty v této oblasti jsou zodpovědné za poskytování přirozeného deklaračního rozhraní do světa, který jste vytvořily, to znamená, že např. nemusíme OGRE říkat nastav tyto renderovací stavy a pak renderuj 3 polynomy, pouze řekneme, že chceme objekt tady, tady a tady, s těmito materiály, renderované z tohoto pohledu, a necháme to engin udělat.

b) Řízení zdrojů

Při vykreslování jakékoliv scény jsou potřeba zdroje, které můžeme vykreslit. Patří mezi ně například geometrie, textury, fonty nebo objekty. Je důležité opatrně spravovat načítání těchto zdrojů, jejich znovu použití a uvolnění. Právě tuto funkci zastávají třídy v této oblasti.

c) Renderování (vizualizace)

Třídy této kategorie se starají o níže úrovňové funkce celého renderovacího systému, specifické renderovacímu systému API jako jsou např. buffery, renderovací stavy.

Můžete si všimnout, že kolem okraje enginu je rozptýleno mnoho pluginů. OGRE je navrženo tak, aby se mohlo dále rozrůstat a pluginy jsou obvyklou cestou jak toho dosáhnout. Mnoho tříd v OGRE může být rozděleno nebo rozšiřováno. Pluginy mohou sloužit např. pro změnu organizace scény, přidávat nové renderovací systémové implementace (Direct3D nebo OpenLG) nebo dodávat způsob jak načíst zdroje z jiných pramenů (řekněme z webu nebo databáze). Toto je jen malý příklad toho co mohou pluginy dělat. Ovšem i z tohoto příkladu lze poznat, že je lze pluginy zapojit do každého aspektu systému. V tomto ohledu OGRE není jen řešením jednoho přesně definovaného problému, ale lze jej použít na hodně úkolů, které chcete udělat. [3]

2.2 Jmenný prostor

OGRE využívá vlastnost C++ jménem jmenný prostor. Ta uživateli dovoluje vkládat třídy, struktury, proměnné a další dovnitř 'jmenného prostoru' a díky tomu lze lehce předejít střetu jmen. V praxi to znamená, že každá třída, typ atd. by měla být nadepsána 'Ogre::', popřípadě, 'Ogre::Camera', 'Ogre::Vector3' atd. A pokud kdekoliv ve své aplikaci použijete například název Vektor3 nedojde ke kolizi jmen.

2.3 Objekt Root

Root objekt je základní bod v systému OGRE. Tento objekt musí být vytvořen jako první a je posledním, který je uvolněn. Což znamená, že pokud si vytvoříme aplikaci, kde *Root* bude členem aplikačních objektů, pak *Root* je vytvořen těsně před prvním aplikačním objektem a smazán až po smazání posledního aplikačního objektu.

Objekty *Root*u dovolují uživateli konfigurovat systém, např. pomocí metody `showConfigDialog()`, která může upravovat všechny renderovací systémové nastavení a vytváří dialogové okno, ve kterém lze měnit nastavení rozlišení, barevné hloubky, nastavení velikosti obrazovky atd.

Pomocí tříd *Root*u můžete získat ukazatele na jiné objekty v systému, jako je *SceneManager*, *RenderSystem* a různé jiné správce zdrojů.

Nakonec, pokud zapnete OGRE v nepřetržitém režimu vykreslování, tj. chcete vždy obnovit všechny vykreslované cíle co nejrychleji (norma pro hry a dema, ale ne pro utility v oknech), objekt *Root* má metodu nazvanou `startRendering`, po jejímž zavolání otevře průběžnou vykreslovací smyčku končící pouze při zavření všech renderovacích oken nebo pokud *FrameListener* objekty signalizují zastavení cyklu. [3] [4]

2.4 Objekty správy scény

2.4.1 SceneManager

Na rozdíl od *Root*u je *SceneManager* pravděpodobně nejdůležitější část systému z aplikačního hlediska. Určitě je to objekt, který je nejpoužívanější v aplikacích. *SceneManager* má na starosti obsah scény, která má být enginem renderována. Je odpovědný za organizaci jejího obsahu s využitím jakékoliv techniky, kterou považuje za nejlepší, za vytváření a správu všech kamer, pohybujících se objektů, světel, materiálů (povrchové vlastnosti objektů) a pro správu 'geometrie světa', což je rozlehlá statická geometrie obvykle používána pro reprezentaci nemovité části scény.

Využívá se například k vytváření kamery pro scénu, pro načítání nebo mazání světel, k uchovávání seznamu objektů použitelných na scéně atd. Taktéž posílá scénu *RenderSystem*u pokud je potřeba ji vyzualizovat. Metodu `sceneManager::_renderScene` nelze použít přímo v programu - je volána automaticky kdykoliv při updatu vykreslovaného objektu.

Protože různé typy scén potřebují rozdílné algoritmy pro přístup a rozhodování o tom, kdy bude který objekt poslán *RenderSystem*u, aby bylo dosaženo dobrého vykreslovacího výkonu, je třída *SceneManager* vytvořena tak, aby mohla být rozdělena na dílčí třídy pro různé typy scén. Výchozí *SceneManager* renderuje scénu, ale jen málo, nebo vůbec ji neorganizuje a nemůžeme tedy očekávat vysoký výkon v oblasti velkých scén. Záměrem tvůrců OGRE je proto vytvoření podtřídy pro každý typ scény. Ty pod povrchem optimalizují organizaci scény pro předpokládaný nejlepší výkon kterého je možno dosáhnout konkrétním typem scény. [3] [5]

2.4.2 Pohyblivé objekty scény

Jsou to objekty, kterými lze na scéně pohybovat, patří mezi ně entita, kamera a světla.

2.4.2.1 Entita

Entita je instance pohyblivého objektu na scéně. Může jí být auto, osoba, pes, hvězdice nebo jakýkoliv hýbající se objekt. Jediným předpokladem je, že nemusí mít nutně své pevné místo na scéně, ale může se pohybovat.

Tyto objekty jsou založeny na diskrétních *mesh*ích. Více entit může být založeno na stejných *mesh*ích, neboť často chceme vytvořit na jedné scéně více kopií toho samého objektu.

Entita se vytváří voláním funkce `SceneManager::createEntity`, které přiřadíme jméno a specifické jméno *mesh*e, na němž je založena (např. "OgreHead.mesh"). O načtení *mesh*e se stará *SceneManager*, který za uživatele zavolá *MeshManager* (ten patří mezi *ResourceGroupManager*y – viz kapitola 2.5.1), tento manažer už zajistí načtení pouze jedné kopie.

Sami o sobě nejsou entity považovány za část scény. Tou se stanou až po připojení k *SceneNode* (viz kapitola 2.4.2.3). Jejich připojením pak můžeme vytvořit komplexní hierarchii vztahů mezi pozicemi a směry entit.

Každá entita, vytvořená z konkrétního *mesh*e, má defaultně nastaveny textury načítané společně s tímto *mesh*em. Nicméně OGRE dovoluje změnu těchto textur, takže lze vytvořit mnoho entit závislých na stejném *mesh*i, ale s různými texturami. Změny materiálů se provádí na základě dílčích *mesh*ů. Vždy, když vytvoříme entitu založenou na konkrétním *mesh*i, je složena z dílčích entit, jejichž počet odpovídá počtu dílčích *mesh*ů. Každá jedna dílčí entita je založena na jednom dílčím *mesh*i. Ke konkrétním dílčím entitám přistupujeme pomocí metody `Entity::getSubEntity`. Jakmile máme odkaz na dílčí entitu, můžeme změnit její texturu pomocí příkazu `setMaterialName`. [3] [4]

Obrázek 5 – Ukázka entity

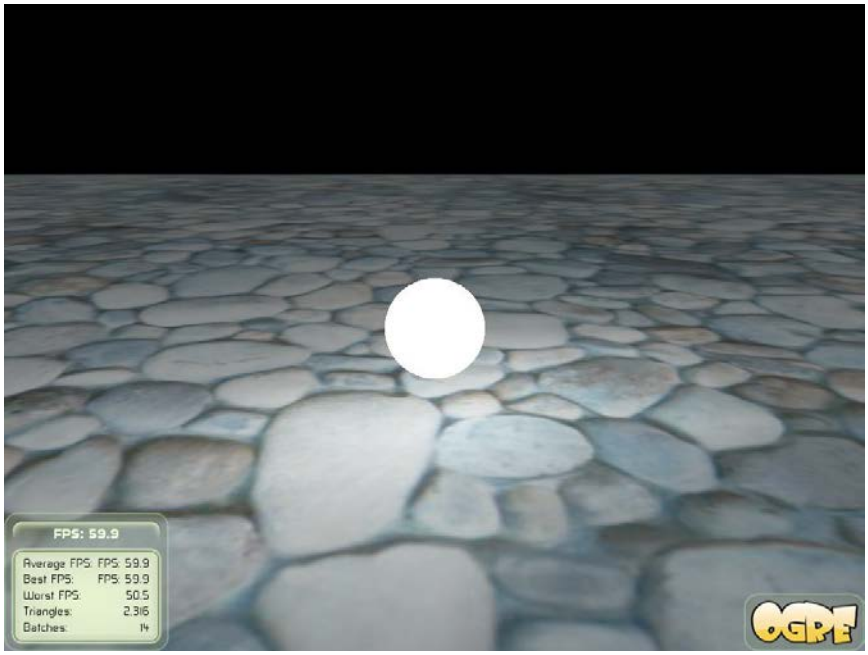
2.4.2.2 Kamera

Kamera je objekt, který používáme k prohlížení scény, dá se říct, že je místem, odkud je scéna renderována. Tento pohled je pak zobrazen v nějakém bufferu, obvykle oknu nebo textuře. Kamery OGRE podporují obě perspektivní projekce – defaultní (objekty se zmenšují se zvětšující vzdáleností od kamery) a ortografickou (objekty se nezmenšují se zvětšující vzdáleností). Každá kamera v sobě nese styl vykreslování, pole viditelnosti, vzdálenosti vykreslování atd. a více kamer může být zaměřeno na stejný cíl vykreslování. Scéna se pak rozdělí na více částí podle počtu kamer. Kamera pracuje podobným způsobem jako *sceneNote*. Také má funkce pro nastavení pozice, otáčení a lze ji připojit k jakékoliv *sceneNote* na scéně. Stejně jako u *sceneNote* je pozice kamery relativní ke svým předkům.

2.4.2.3 Světla

OGRE poskytuje uživateli 3 typy světel, jsou jimi bodové světlo, světelný kužel a směrové světlo. Bodové světlo je světelný zdroj, který vyzařuje světlo do všech směrů. Světelný kužel funguje stejně jako svítilna. Má pozici, kde světlo začíná a pak směřuje určitým směrem. Lze u něj také nastavit velikost úhlu vnitřního a vnějšího kruhu (svítilna totiž svítí

uvnitř více než v okolí). Směrové světla simulují vzdálená světla osvětlující vše, co je na scéně v jejich směru. Mohou například simulovat měsíční světlo.



Obrázek 6 – Ukázka bodového světla

2.4.3 SceneNode

Třída *SceneNode* je logický element hierarchie scény, který slouží k její organizaci. Mohou na ni být upoutány dceřiné *SceneNode*, pohyblivé objekty jako jsou entity, kamery atd. To se provádí funkcí `attachObject`. Pomocí *sceneNote* lze měnit pozici objektů, posouvat je, měnit jejich velikost a otáčet je. [2] [4]

2.5 Objekty řízení zdrojů

2.5.1 ResourceGroupManager

Tato třída spravuje zdroje, což jsou soubory dat, které musí být odněkud načteny, aby mohly být poskytnuty ogre. *ResourceManager* je vlastně 'hub' pro načtení a znovupoužití zdrojů jako jsou textury a *mesh*e. V něm se definují skupiny pro zdroje, které pak mohou být znovu načteny a vyjmuty kdykoliv je potřeba. Existuje mnoho *ResourceManager*ů, z nichž každý spravuje individuální typ zdroje. Jsou to např. *TextureManager* (manažer pro správu textur) nebo *MeshManager* (manažer pro správu meshů) a jiné.

*ResourceManager*y zajistí, aby zdroje byly načteny pouze jednou a poté byly v enginu sdíleny. Také spravují požadavky zdrojů na paměť. Zdroje vyhledají v několika umístěních a mohou je načítat i z komprimovaných zip souborů.

S tímto managerem se většinou nepracuje přímo. Je volán jinými částmi OGRE, např. pokud je potřeba přidat texturu k materiálu, je zavolán *TextureManager*. Samozřejmě je možné zavolat manažer zdrojů přímo pro před načtení zdrojů, většinou je ale lepší nechat ogre rozhodnout kdy má být konkrétní manažer zavolán.

Pro přidání cesty ke zdrojům se používá příkaz:

```
Root::getSingleton().addResourceLocation
```

[3]

2.6 Mesh

Mesh představuje diskrétní model souboru geometrie, který je samostatný a oproti celé scéně relativně malý. U *mesh* objektů se předpokládá, že představují pohyblivé objekty a nejsou používány pro rozlehlou geometrii, která je typicky použita pro pozadí.

Tyto objekty jsou jedním z typů zdrojů a jsou spravovány *MeshManagerem*. Většinou jsou načteny z vlastního objektového formátu .mesh, který OGRE podporuje. Tento formát lze vytvořit exportem z různých modelů, se kterými může být manipulováno pomocí různých nástrojů pro to určených. *Mesh*e jsou také základem pro pohyblivé objekty nazývající se Entity a mohou být animovány.

Mesh lze také vytvořit manuálně pomocí metody:

```
MeshManager::createManual
```

2.6.1 Materiály

Materiály řídí povrchové vlastnosti objektů vykreslených na scéně. Jsou jimi například odrazivost barvy, světlost, kolik vrstev textur je zobrazeno, jaké obrázky na nich jsou a jak jsou sloučeny dohromady, které speciální efekty jsou použity (jako např. mapování prostředí), jak jsou textury filtrovány atd. V podstatě všechny vlastnosti týkající se vhledu objektu mimo jeho tvar jsou řízeny třídou materiálů.

Jsou nastaveny buď programově zavoláním metody `SceneManager::createMaterial` a nastavením vlastností nebo je můžeme specifikovat ve skriptu který je načten v průběhu programu.

Seznam materiálů použitelných pro scénu spravuje *SceneManager*. Nový materiál se do tohoto seznamu přidává buď již zmíněnou metodou `SceneManager::createMaterial`, nebo je načten společně s *mesh*em, ke kterému se vždy načítají vlastnosti materiálů. Pokaždé když je materiál přidán do *SceneManager*u, začíná s defautně nastavenými vlastnostmi, které jsou definovány OGREm. Zde jsou některé uvedeny:

- Odrazivost světla z okolí (odraženého světla) je nastavena na bílou (vše odráží)
- Odrazivost světla produkovaného světlenými objekty je nastavena na bílou (odráží vše)
- Odrazivost světelných ploch nastavena na černou (není)
- osvětlení sama sebe je nastaveno na černou (není)
- odlesky nastaveny na 0 (není lesklý)
- žádné vrstvy textur (to znamená žádné textury)
- okolní osvětlení na scéně nastaveno na středně šedou
- povoleno dynamické osvětlení
- bilineární filtrování textur

Tyto a další vlastnosti můžete později přenastavit pomocí funkce `SceneManager::getDefaultMaterialSettings()`. [10] [3]

2.7 Objekty vykreslování

2.7.1 RenderSystem

RenderSystem je abstraktní třída, která definuje interface pro podporovanou 3D API. Je zodpovědná za odeslání renderovací operace do API a nastavení všech různých vizualizačních možností. Tato třída je abstraktní, neboť všechny implementace jsou pro každou renderovací API specifické - existují proto API specifické dílčí třídy pro každou vykreslovací API (např. *D3DRenderSystem* pro Direct3D). Potom, co je systém inicializován pomocí `Root::initialise`, jsou objekty *RenderSystem*u pro vybranou renderovací API dostupné metodou `Root::getRenderSystem()`.

Nicméně typická aplikace by neměla mít za normálních podmínek potřebu manipulovat s objekty *RenderSystem*u přímo - vše potřebné pro vizualizaci objektů a přizpůsobení nastavení by mělo být dostupné ve *SceneManager*u, materiálech a jiných scénově-orientovaných třídách. *RenderSystem* bude potřebný pouze při vytváření více renderovacích oken (v tomto případě kompletně separované okno, ne mnohonásobné výřezy jako split-screen efekt který se provádí pomocí třídy *RenderWindow*) nebo při použití jiných pokročilých funkcí, které tato třída obsahuje. [3]

3 DALŠÍ DŮLEŽITÉ OBJEKTY A EFEKTY OGRE

3.1 Animace

OGRE podporuje relativně flexibilní animační systém, který uživateli dovoluje použít vytvořené animace k různým účelům. Tento systém je založen na čtyřech typech animace, jsou jimi skeletální animace, vertexová animace, animace *sceneNode* a animace numerické hodnoty. Pokud je vytvořena entita obsahující animace jakéhokoliv typu, jsou jí přiřazeny objekty '*animation state*' (pro každou animaci jeden). Ty dovolují uživateli specifikovat animační stavy konkrétní entity. OGRE dovoluje použití stejné animace na více objeků, její opětovné použití si pak třídí interně.

Načítání ukazatele na objekt *AnimationState* se provádí pomocí metody `Entity::getAnimationState`. Na takto vrácený objekt pak může uživatel použít funkce pro update animace. Aby měla animace nějaký efekt, musí být *AnimationState* povolený (nastavením funkce `setEnabled` na hodnotu *true*) a může se pro něj v případě potřeby nastavit váha a čas pozic mající vliv na správnou funkci aplikace při použití metod korelace. Další důležitou funkcí je `addTime`, která dovoluje uživateli měnit polohy animace postupně, takže se bude automaticky opakovat. `AddTime` dovoluje nastavení i záporných hodnot, takže animační smyčka může běžet i obráceně.

3.1.1 Skeletální animace

Je to proces animace *mesh*ů, kdy se využívá jejich skeletální struktura. Pohybují se pomocí souboru hierarchických kostí obsažených v *mesh*i, kde vertexy se pohybují posunem kosti, k níž patří. Tato metoda se také nazývá *skinning*. Obvykle se tyto animace vytváří pomocí modelovacích programů, jako jsou např. Blender, Maya nebo 3D studio. OGRE pak nabízí exportery, které převedou data z těchto programů do enginu. Podporuje ovšem pouze tyto funkce animace:

- Každý *mesh* může být propojený pouze s jednou kostrou
- Neomezený počet kostí každé kostry
- Více animací pro jednu kostru (např. 'Walk', 'Run', 'Jump', 'Shoot' atd)

- Neomezený počet snímku na jednu animaci
- vertex může být přidělen k více kostem
- na mesh může být aplikováno více animací ve stejný čas
-

Kostry a animace jsou uloženy v souborech *.skeleton*, které jsou vytvářeny exportery. Tyto soubory jsou automaticky načteny při vytvoření entity založené na *mesh*i, který je k nim připojen. Skeletální animace se dá kombinovat s animací vetexovou. Přičemž první se na *mesh* uplatňuje vertexová a na výsledek se pak uplatní skeletální.

Obrázek 7 – Ukázka animace

3.1.2 Vertexová animace

Je dalším způsobem animace *mesh*ů. Používá k ní přímo informace o pohybu vrcholů a je uložena v *.mesh* souboru, kde je přímo napojena na jeho vertexovou strukturu. Vertexová animace má ještě dva podtypy. Jsou jimi morfální animace a poziční animace. Jejich rozdíl je v tom, že zatímco poziční umožňuje zamíchat dohromady více potenciálních pozic vertextů, kdy každá má na konečný stav vertexu různý vliv, morfální animace ukládá při každém snímku apsolutní pozici vertexu a pak mezi nimi interpoluje. Poziční animace se

využívá hlavně pro animaci tváří, morfální pak pro objekty které při animaci radikálně mění strukturu a skeletální animace by pro ně byla nepoužitelná.

3.1.3 Animace sceneNode

Automatické animování *sceneNote* se využívá pro docílení efektů, jako je zahýbání kamery a objekty pohybující se pro předdefinovaných cestách. V zásadě je podobná vnitřní kostře skeletální animace. Po tom, co uživatel vytvoří hlavní animaci pomocí `SceneManager::createAnimation`, vytváří pro každou *SceneNode* *NodeAnimationTrack* a klíčové snímky, které ovládají pozici, orientaci a měřítko, které může být lineární nebo pomocí drážek. Animační stavy se pak používají stejně jako u předešlých animací, jediný rozdíl je poskytování stavů *SceneManager*em místo entity.

3.1.4 Animace numerické hodnoty

Využívá rozšiřitelnou strukturu OGRE k animaci jakékoliv hodnoty. Kromě specifických typů animace, které mohou zahrnovat i nejčastější využití rámců animace, může uživatel také využít animaci podle jakékoliv hodnoty která je poskytována prostřednictvím rozhraní animovatelných objektů. [3]

3.2 Stíny

Stíny jsou důležitou částí chování scény při vykreslování. Dělají scénu více realistickou a pomáhají uživateli pochopit prostorové umístění objektů. Nicméně jsou také nejnáročnějším aspektem 3D vykreslování a velká část této oblasti je stále zkoumána. Existuje mnoho technik pro vykreslování stínů, žádná není dokonalá a každá má své výhody a nevýhody. Z tohoto důvodu poskytuje OGRE mnohonásobnou implementaci stínů s velkými možnostmi konfigurace, takže si uživatel může vybrat pro svou scénu nejvhodnější řešení.

Základně spadá implementace stínů do dvou kategorií – šablonové stíny a stíny založené na texturách. Jejich názvy vlastně popisují metodu vytváření stínů. Kromě toho jsou ještě další způsoby jak vykreslit stíny na scéně – modulativní stíny u kterých scéna v místě stínu

tmavne a aditivní světelná maska, která naopak hromadí světlo v místech mimo stín. Další možností je použití integrovaných stínů textur, ty dávají uživateli naprostou kontrolu při aplikaci stínů na textury. OGRE podporuje kombinaci všech těchto způsobů. [3] [4]

Obrázek 8 – Ukázka aditivních šablonových stínů

3.3 Třída overaly

Tato třída dovoluje uživateli vykreslit 2D a 3D elementy na vrcholu normální scény a vytváří tak efekty jako jsou menu systémy, status panely a jiné. Příkladem může být panel zobrazující statistiky vykreslování nebo panel s názvem OGRE, oba jsou aktivní hned po spuštění standartní OGRE aplikace a patří mezi 2D elementy. 3D elementem je například kokpit letadla v leteckém simulátoru.

Overlay se vytváří buď metodou `SceneManager::createOverlay` nebo mohou být definovány ve skriptu uloženém v souboru *.overlay*. Skripty jsou ovšem jako u většiny objektů výhodnější, neboť se dají jednoduše měnit bez potřeby rekompilace celé aplikace.

Na scéně může být definovaný velký počet *overlay*, všechny se totiž inicializují jako neviditelné a zobrazují se pomocí metody `show()`. Zobrazeno jich samozřejmě může být v jeden okamžik více a jejich pořádek (tzn. jak se překrývají) je určen pomocí metody `Overlay::setZOrder()`. [3]

3.4 FrameListener

V OGRE existuje třída, která může přijímat zprávy před a po každém renderování snímku. Nazývá se *FrameListener*. V tomto rozhraní jsou definovány tři funkce pro přijímání událostí snímků, jsou jimi:

```
virtual bool frameStarted(const FrameEvent& evt);
virtual bool frameRenderingQueued(const FrameEvent& evt);
virtual bool frameEnded(const FrameEvent& evt);
```

Funkce `frameStarted` je volána před vykreslením snímku, funkce `frameRenderingQueued` pak po vykreslení všech objektů ale před smazáním bufferů vykreslovacího okna. Poslední funkce `frameEnded` je spuštěna těsně po vykreslení snímku. Všechny funkce jsou binárního typu, a pokud vrací hodnotu *true*, ta v podstatě znamená stále vykreslovat, pokud alespoň jedna z funkcí vrátí hodnotu *false*, program skončí.

3.5 Obloha

V ogre si uživatel může zvolit mezi třemi typy nebe, jsou jimi *Skybox*, *SkyPlane* a *SkyDomes*. *SkyBox* je v zásadě obrovská krychle obklopující scénu. Lze u něj nastavit použitý materiál, vzdálenost od kamery a to zdali je vykreslován jako první nebo ne. Nevýhodou tohoto typu oblohy je možnost špatného vykreslení scény při špatně zadaných parametrech. Vytváří se pomocí metody `setSkyBox`. Dalším způsobem zobrazení nebe je použití *SkyDomes*. Ten je podobný předchozímu typu oblohy. Také jej tvoří obrovská krychle okolo scény, zásadní rozdíl je ale v tom, že textura je projektována na *SkyBox* sférického charakteru. I když je základem stále krychle, vypadá textura jako by byla obalena okolo povrchu koule. Největším záporem u tohoto typu oblohy je absence textury

na spodní části krychle. Takže se hodí spíše pro scény obsahující nějaký terén, díky němuž se volná plocha zakryje. Vytváří se voláním metody `setSkyDome`. Posledním typem oblohy je *SkyPlane*, ten je naprosto odlišný od obou předešlých. Místo renderování oblohy na krychli totiž využívá obyčejnou rovinu. Taktéž se vytváří jiným způsobem, nejprve uživatel musí vytvořit rovinu, v níž si určí vykreslovací vzdálenost od kamery, posléze vytvoří samotnou oblohu použitím metody `setSkyPlane`, ve které si určí její texturu a velikost, popřípadě zahnutí. Protože jej tvoří pouze zahnutá plocha a nezakrývá dolní polovinu scény je tento typ vhodný zejména pro scény, které obsahují nějaký terén a nejde přes něj vidět za horizont. [2]

3.6 Mlha

Nejdůležitější informací o mlze je že ji nelze vytvořit v prázdném prostoru, protože je pouze filtrem použitým na objekty, na něž směřuje kamera. Důsledkem toho je, že když se díváte do prázdna, nevidíte mlhu ale pouze barvu pozadí. Proto je důležité mít nastavenou na pozadí barvu podobnou mlze. Pokud ovšem barvu pozadí i mlhy nastavíme na velmi tmavou, můžeme ji použít jako tmu. Existují dva typy mlhy, lineární a exponenciální. Lineární houstne lineárně a exponenciální exponenciálně.

Obrázek 9 – Ukázka lineární mlhy

4 PLUGINY A SKRIPTY

Pluginy a skripty jsou nedílnou součástí OGRE. Rozšiřují ho a zjednodušují práci s ním.

4.1 Skripty

OGRE řídí mnoho svých vlastností pomocí skriptů kvůli zjednodušení nastavení. Skripty jsou jednoduché textové soubory, které mohou být upravovány v mnoha standartních textových editorech a jejich editace má okamžitý vliv na aplikace založené na OGRE bez potřeby rekompilace. Skripty jsou napsány jazykem založeným na C++, kde oddíly ohraničují složené závorky a poznámky jsou uváděny v řádcích začínajících znaky '//'.

4.1.1 Materiálové Skripty

I když lze všechny materiály pro konkrétní scénu popsat v kódu aplikace použitím metod tříd materiálů a vrstev textur, je to docela nepraktické. Výhodnější je použít skripty, jejichž vytvoření je relativně jednoduché a navíc se dají snadno znovu použít.

Tyto skripty jsou načteny při inicializaci zdrojů, kdy OGRE vyhledává ve všech místech spojených s touto skupinou všechny soubory *.material*. Lze je načíst i ručně pomocí metody `MaterialSerializer::parseScript`. V tuto chvíli materiály nejsou načteny kompletně, ale bez textur a jiných zdrojů. Bylo by totiž velmi náročné načítat všechny zdroje, zvláště když s velkou pravděpodobností nebudou všechny využity. Konkrétní textury se pak načítají při použití konkrétního materiálu, nebo je lze načíst manuálně. Každý materiál musí mít samozřejmě unikátní jméno.

4.1.2 Skripty kompozitoru

Kompozitor umožňuje vytváření efektů ovlivňujících celou scénu, které jsou aplikovány na uživatelův pohled. *Kompozitory* totiž ovlivňují scénu až po jejím vykreslení a umožňují tak použití efektů na celou scénu, např. změna barevnosti. Tyto skripty stejně jako skripty materiálů načítají při inicialiaci zdrojů. Jejich přidání do aplikace se pak provádí metodou `CompositorManager::getSingleton().addCompositor(viewport, compositorName)`

.

4.1.3 Částicové skripty

Umožňují uživateli definovat částicové systémy, které budou vloženy k jeho kódu bez jejich zdlouhavého nastavování uvnitř kódu, což dovoluje velmi rychlou odezvu na jakoukoliv změnu, která je ve skriptu provedena. Částicové systémy definované ve skriptech jsou používány jako šablony a může podle nich být při běhu vytvořeno mnoho jiných.

Částicové systémy jsou načteny při inicializaci. Poté, co byli skripty načteny lze vytvořit systémy na nich založené metodou `SceneManager:: createParticleSystem ()`. Ta může obsahovat buď název nové šablony, nebo šablony, na níž bude systém založen.

4.1.4 Skripty třídy overaly

V kódu programu lze nastavit *Overlay* pomocí metod tříd *SceneManager*u, *Overlay* a *Overlayelement*. To je ovšem relativně neohrabané a v praxi je lepší použití skriptu. Ty jsou načítány při inicializaci systému. [3]

5 POROVNÁNÍ OGRE S JINÝMI ENGINY

Obrázek 10 – Loga jednotlivých enginů

Hlavní rozdíl OGRE od většiny enginů tkví v jeho základní vlastnosti – je primárně určen pouze pro vykreslování scény - a všechny ostatní funkce (jako fyzika, detekce kolizí, zvuk, atd.) se k němu přidávají pomocí pluginů. Pokud tedy budeme porovnávat samotné OGRE s ostatními enginy, většinou nám budou některé funkce chybět. Pro porovnání jsem si zvolila 2 enginy, jedním z nich je Irrlicht a druhým Panda3D. Irlicht je 3D engine, známý svou malou velikostí a kompatibilitou. Panda3D je herní engine, který obsahuje vše, co je potřebné pro tvorbu her (jako grafika, audio, I/O, detekce kolizí a jiné). Všechny tyto enginy jsou napsané v programovacím jazyku C++, Panda3D ale jako jazyk pro vývoj aplikací využívá python. Ostatní enginy mohou v různých modifikacích jiné jazyky než C++ využívat také. Všechny jsou založeny jak na Direct3D tak na OpenGL. Nejvíce multiplatformním je z nich Irrlicht, který běží na velkém množství operačních systémů. Všechny enginy mají početnou komunitu a Ogre s Irrlichtem i dobrou dokumentaci, Panda3D za nimi v tomto ohledu zaostává, dokumentaci sice má, ale nepopisuje dostatečně její vlastnosti.

Všechny tři enginy jsou objektově orientované. Při srovnávání vlastností jsou si OGRE a Irrlicht velice blízké, co se týče vlastností patřících k renderování Panda3D zaostává. Je však uceleným herním enginem. V tabulce 1 jsou uvedeny podrobnější vlastnosti jednotlivých enginů. [1] [8] [9]

Tabulka 1 - Srovnání vlastností jednotlivých enginů.

engine	OGRE	Irrlicht	Panda3D
autor	Steve Streeting	Nikolaus Gebhardt	více
grafické api	OpenGL, DirectX	OpenGL, DirectX	OpenGL, DirectX
podporovaný programovací jazyk	C/C++	C/C++, C#, VB.NET	Python, lze i C++
podporovaný operační systém	Windows, Linux, MacOS	Windows, Linux, MacOS, Solaris, FreeBSD, Xbox	Windows, Linux, MacOS, SunOS
dokumentace	ano	ano	ano
poslední stabilní verze	1.7.3 [Cthugha]	1.7.2	1.7.2
datum jejího vydání	8.5.2011	15.11.2010	7.3.2011
obecné rysy	objektově orientovaný deging, Plug-in Architektura, Save/Load Systém	objektově orientovaný deging, Plug-in Architektura	objektově orientovaný deging, široká škála funkcí pro tvorbu her
skriptovací jazyk	pseudo C++		python
fyzika	Minimální detekce kolizí	detekce kolizí	základní fyzika, detekce kolizí, tuhá tělesa, fyzika prostředků
osvětlení	podle vertexů, podle pixelů,	podle vertexů, podle pixelů,	podle vertexů, podle pixelů

	podporuje lightmapy	podporuje lightmapy	
stíny - použité metody	Shadow Mapping, Shadow Volume	Shadow Volume	neuvedeno
texturování	základní textury, multi-texturing, Bumpmapping, Mipmapping, volumetrické materiály, projektivní mapování textur	základní textury, Multi-texturing, Bumpmapping, Mipmapping	základní textury
shadery	Vertex, Pixel, High Level	Vertex, Pixel, High Level	High Level
řízení scény	základní, BSP, octree struktura, Occlusion Culling, LOD	základní, BSP, octree struktura	neuvedeno
animace	inverzní kinematika, skeletální animace, morphální animace, animace prolnutí	skeletální animace, morphální animace, animace prolnutí	skeletální animace
meshe	načítání meshů, Skinning, progresivní meshe	načítání meshů	načítání meshů, Skinning
speciální efekty	mapování prostředí, Lens	mapování prostředí,	částicový systém, motion blur, mlha

	Flare efekt, Billboarding, částicový systém, Motion Blur, obloha, voda, mlha	Billboarding , obloha, voda, mlha, částicový systém	
video & zvuk			2D zvuk, 3D zvuk, streamovaný zvuk
vykreslování	fixní funkce, vykreslování do textury, Fonty, GUI	fixní funkce, vykreslování do textury, Fonty, GUI	fixní funkce, stereo vykreslování, GUI

II. PRAKTICKÁ ČÁST

6 3D APLIKACE

Jedním z úkolů teoretické práce bylo vytvořit 3D aplikaci demonstrující vlastnosti OGRE. Aplikace byla vytvořena ve Visual studiu 2010 pomocí programovacího jazyku C++.

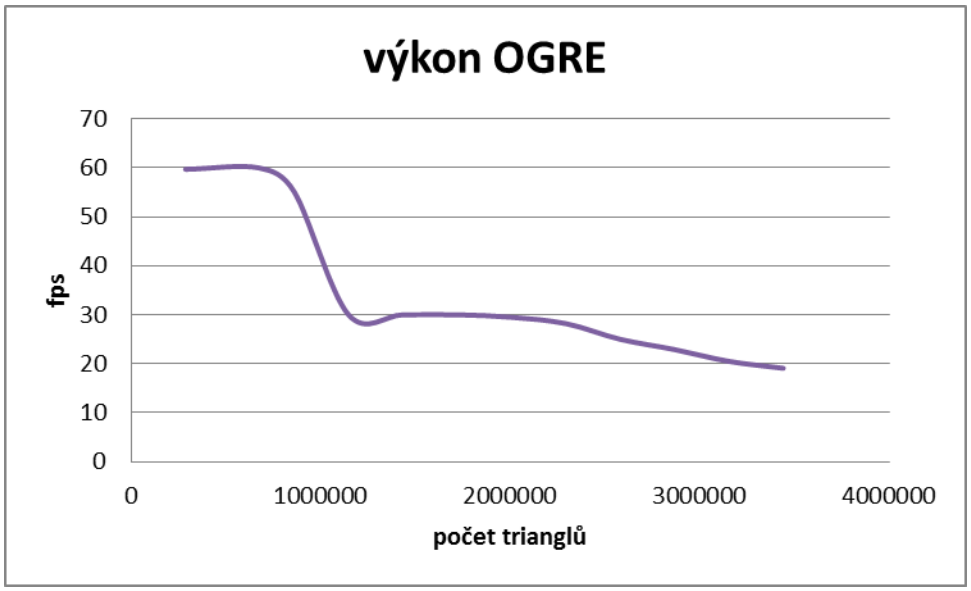
6.1 Komponenty aplikace

Před samotným vytvořením aplikace jsem musela zjistit výkon OGRE, abych věděla, jak velké modely mohu v aplikaci použít, aniž bych razantně zpomalila její chod.

Pro tuto práci jsem si vytvořila jednoduchou scénu obsahující pouze jeden velký model (necelých 300000 trianglů) a postupně jsem k němu přidávala další modely o stejné velikosti. V tabulce číslo dva jsou pak uvedeny moje poznatky.

Tabulka 2 – výkon enginu OGRE

počet trianglů	Fps
287027	59,7
673812	60
860601	55,1
1147392	30
1434177	30
1720964	30
2007753	29,5
2294540	28,2
2581325	25
2868112	22,9
3154901	20,5
3441688	19,1



Obrázek 11 – Graf znázorňující výkon OGRE

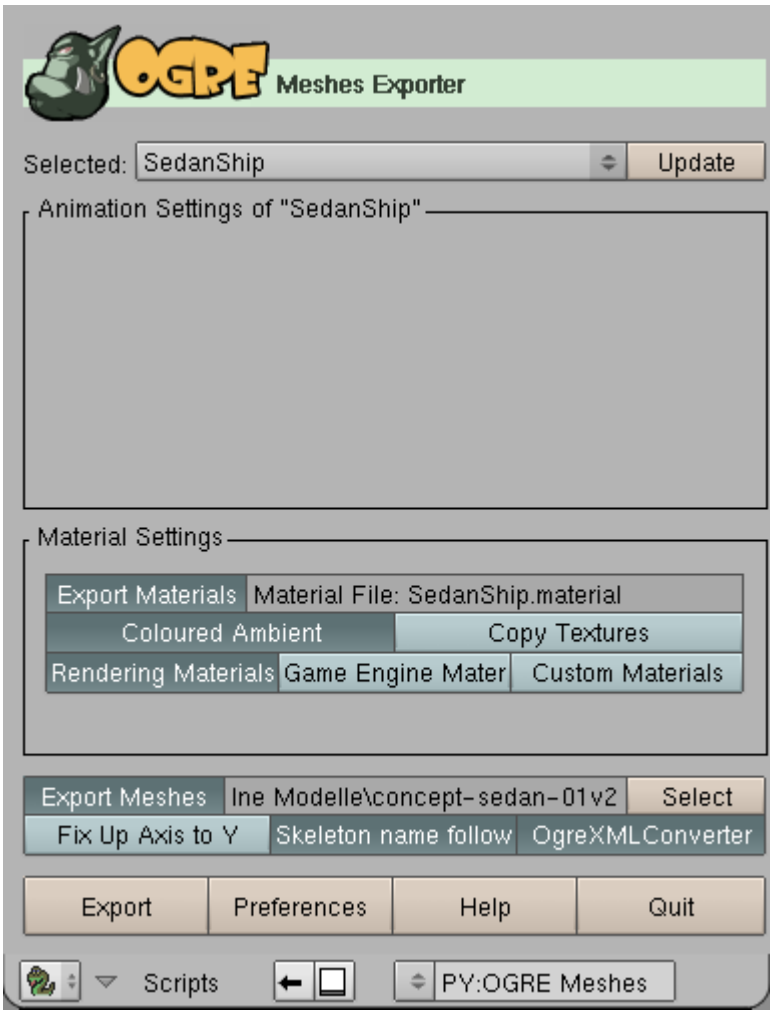
Z uvedeného grafu vyplívá, že OGRE zvládne použití až 3500000 trianglů při plynulém chodu aplikace. Samozřejmě tato hodnota závisí na konfiguraci počítače. Tento test byl proveden na laptopu Asus M51Va s touto konfigurací:

Procesor: Intel Core 2 Duo P8600 s frekvencí 2,4 GHz, 3 MB L2 cache, FSB 1066 MHz
Čipset: Intel PM 45 + ICH9M
Grafická karta: ATI Mobility Radeon HD 3650 s 512 MB vlastní paměti
Operační paměť: 2 x 2 GB DDR2 800 MHz
Pevný disk: 320 GB, 5400 RPM, Rozhraní Serial ATA
Operační systém: Windows 7 Professional 64bitový

6.1.1 Převod modelů do fomátu .mesh z programu blender

Všechny modely použité v OGRE mají formát *.mesh*, což je základní formát pro *mesh*e používaný OGREm. Pro převod objektů do tohoto souboru OGRE nabízí celou řadu nástrojů. Lze je rozdělit do třech kategorií na exportery, XmlConverter a MeshUpgrater. Pro převod modelu z modelovacího programu se vždy použije příslušný exporter. Pro Blender je exportérem Python script. Ten převádí model do formátu .mesh.xml a vytváří

také materiálový skript pro tento model. Případně v něm lze nastavit, zda mají být kopírovány textury a další možnosti.



Obrázek 12 – Blender Exporter

Nyní už model můžeme převést do formátu .mesh. To se provádí pomocí XmlConverteru, což je aplikace určená pro převod .mesh a .skeleton souborů a naopak. Nyní už máme vytvořený .mesh soubor a .material skript a stačí je pouze spolu s texturami umístit do příslušných složek pro zdroje v OGRE. [2]

Meshe jsou umístěny ve složce *OGRE SDK\media\models*

Materiálové skripty v *OGRE SDK\media\materials*

Textury v *OGRE SDK\media\materials\textures*

6.1.2 Nastavení materiálů

Blender Exproter ovšem nepřesně převádí některé materiálové skripty, proto jsem všechny kontrolovala, popřípadě přepisovala. Zde je uveden jeden vzorový:

```
material AG02_Branch2
{     receive_shadows on
      technique
      {     pass
            {     ambient 1.000000 1.000000 1.000000 1.000000
                  diffuse 0.800000 0.800000 0.800000 1.000000
                  specular   0.249020   0.249020   0.249020   1.000000
12.500000
                  emissive 0.000000 0.000000 0.000000 1.000000
                  texture_unit
                  {     texture AG02brn2.tif
                        tex_address_mode wrap
                        filtering trilinear
                        colour_op alpha_blend
                  }}}}
```

Skript vždy obsahuje název materiálu a použité techniky, `receive_shadows on` povoluje vykreslení stínu, druhá technika nastavuje konkrétní texturu a její vlastnosti.

6.1.3 Tvorba terénu pro aplikaci

Terén jsem si předem vytvořila ve speciální aplikaci pomocí funkcí TerrainSystému. Jeho vlastnosti se uložili do souboru "*terrain.cfg*", který je potom v aplikaci volán funkcí `mSceneMgr->setWorldGeometry("terrain.cfg");`

Mezi nastavované vlastnosti patřila velikost terénu, jeho textury nebo nastavení členitosti terénu.

Obrázek 13 – Vytvořený povrch

6.2 Popis aplikace

6.2.1 Základní myšlenka

Protože má být aplikace postavená na enginu OGRE, který popisuji v teoretické práci, a má demonstrovat jeho vlastnosti, snažila jsem se využít pouze tento engine bez dalších přídavných knihoven. V praxi to znamená, že k spuštění stačí mít v aplikaci přidanou pouze knihovnu OGRE a žádné další speciální knihovny nejsou potřeba.

6.2.2 Jednotlivé funkce

Aplikace představuje širokou paletu funkcí, které lze v OGRE nastavit. Kromě základní scény obsahující světla, kameru, povrch a celkovou geometrii, obsahuje i specifický kamerový systém, hlavní postavu, kterou lze ovládat klávesnicí a další funkce nastavované pomocí tlačítek.

Seznam použitých tlačítek:

A, S, D, W – pohyb postavy

R – nastavení typu vykreslované scény

O – přepínání různých typů oblohy

C – přepínání mezi různými typy kompozitoru

K – zapnutí / vypnutí detekce kolizí

B, N, M, Shift – změna barvy světla

Pohyb myši – otáčí kamerou, pouze pokud je tato funkce zapnuta

Pravé tlačítko myši, levé tlačítko myši – nastavení přiblížení a oddálení kamery

Prostřední tlačítko myší – nastavení zapnutí / vypnutí otáčení kamery

6.3 Vytvoření aplikace

6.3.1 Scéna

Základní scéna obsahuje travnatý povrch se stromy, květinami a oblohu. Není moc velký, slouží totiž pouze k demonstraci vlastností.

6.3.2 Pohyblivý objekt a kamerový systém.

Pohyblivou postavou aplikace je ork. Je vytvořen jako základní entita a připojen na vlastní *sceneNode*. Je na něj navázána kamera. Ta se může otáčet okolo vlastní osy. Pohyb entity je realizován při zmáčknutí tlačítek k tomu určených. Při pohybu je entita animována.

6.3.3 Vstupy do aplikace

Vstupy do aplikace jsou realizovány pomocí OIS, což je knihovna implementovaná v OGRE, která podporuje a umožňuje využití mnoha různých zařízení standartu HID. Tři základní kategorie zařízení jsou myš, klávesnice a joystick.

Pro vstupy je využit interface *listener*ů, ty informují tom, ke které události došlo. Například, pokud je zmáčknuto tlačítko, je využita událost `KeyListener::keyPressed` a pokud je toto tlačítko puštěno, je využita událost

KeyListener::keyReleased pro všechny registrované třídy *KeyListeneru*. Díky tomu se nemusíme starat, zda bylo v minulém framu stisknuto tlačítko či nikoliv.

Dalším důležitým poznatkem o OISi systému *listener*ů je, že může být pro každé zařízení použit pouze jeden listener.

6.3.4 Detekce kolizí

Protože je OGRE pouze renderovací engine, neobsahuje klasickou detekci kolizí. Ovšem aplikace bez použití detekce kolizí by nevypadala nejlépe, proto je zde využita alespoň minimální detekce kolizí, kterou samotné OGRE dovoluje. Jejím základem je třída `RaySceneQuery`, která vždy podle zadaného směru vyšetřuje objekty v něm obsažené.

6.3.4.1 S povrchem

Detekce kolizí s povrchem funguje tak, že pokud je poblíž orka povrch, jeho souřadnice y se vždy nastaví na hodnotu povrchu. Na následujícím kódu podrobněji vysvětlím její princip.

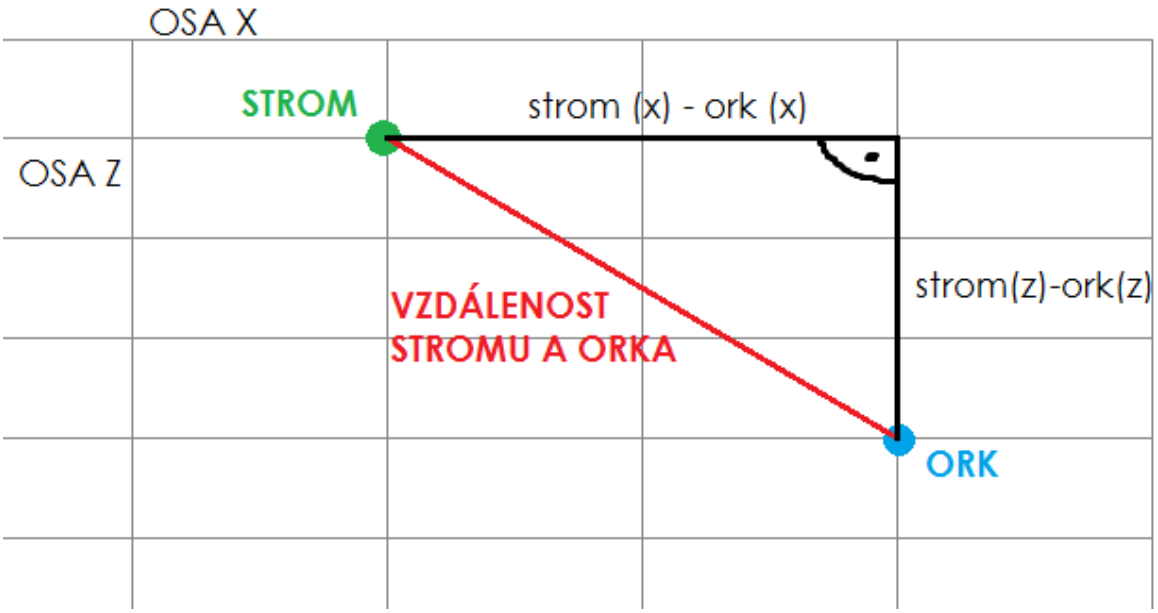
```
Ogre::Vector3 camPos = mSceneMgr->getSceneNode("NinjaNode")-
>getPosition();
          Ogre::Ray cameraRay(Ogre::Vector3(camPos.x,
camPos.y + 10, camPos.z), Ogre::Vector3::NEGATIVE_UNIT_Y);
          mRaySceneQuery->setRay(cameraRay);
          mRaySceneQuery->setSortByDistance(false);

          // Perform the scene query
          Ogre::RaySceneQueryResult &result = mRaySceneQuery-
>execute();
          Ogre::RaySceneQueryResult::iterator itr =
result.begin();
          // Get the results, set the camera height
          if (itr != result.end() && itr->worldFragment)
               {
                    Ogre::Real terrainHeight = itr-
>worldFragment->singleIntersection.y;
                    mSceneMgr->getSceneNode("NinjaNode")-
>setPosition( camPos.x, terrainHeight+5, camPos.z);
               }
```

Nejprve jsem si vytvořila objekt *Ogre::Ray*, ten se dá nazvat paprskem, jež má určenou pozici a směr (10 nad okrem, směr -y). Tento paprsek pak využíváme třída

Ogre::RaySceneQuery, ta nám uchovává informace o geometrii, na kterou tento paprsek narazil. Její výsledek se pak zapisuje do proměnné result. S tímto výsledkem už můžeme pracovat. V tomto případě si najdeme *worldFragment* (náš povrch) a orka na něj postavíme.

6.3.4.2 Se stromy

Základní princip detekce je stejný jako u povrchu. Rozdíl tkví v konečné realizaci a směru objektu *Ogre::Ray*. Směr je vždy nastaven stejný jako u orka. Konečná detekce funguje podle obrázku číslo 14. Vždy se zjistí pozice orka a modelu, na nějž detekci uplatňujeme. Pak pomocí Pythagorovy věty vypočteme jejich přímou vzdálenost (zobrazeno na obrázku). Pokud je větší než minimální určená vzdálenost, nic se neděje. V opačném případě se ork nemůže posunout dál.



Obrázek 14 – princip detekce kolizí s objektem

7 PŘÍRUČKA

Dalším úkolem praktické části této práce bylo vytvoření příručky pro začínající uživatele OGRE.

7.1 Základní myšlenka

Engine OGRE má v angličtině velmi dobrou základní dokumentaci. Byli vydány 2 knihy zabývající se tímto enginem. Také na portálu OgreWiki, jež spadá pod záštitu tvůrců OGRE, najde začínající uživatel mnoho rad a tutoriálů. Inspirovala jsem se tedy těmito zdroji a vytvořila příručku obsahující základní práci s enginem. Všechny funkce tohoto enginu využité v příručce jsou podrobně popsány a vysvětleny na příkladech. Po přečtení příručky a absolvování základních tutoriálů získá uživatel základní znalosti tohoto enginu. Další důležitou vlastností příručky je její použití pouze se základní knihovnou. Z toho vyplývá, že modely v ní využité jsou součástí knihovny již po jejím nainstalování.

7.2 Hierarchie

Příručka k enginu OGRE obsahuje 8 kapitol, 6 z nich je pojato formou tutoriálů. Na začátku každé kapitoly je vždy shrnutí jejího obsahu. Na konci několik otázek. Správné odpovědi jsou uvedeny na konci příručky. Praktické kapitoly vždy obsahují základní teorii, kterou uživatele uvedou do problému řešeného v kapitole. Za teorií vždy následuje šablona, do níž pak v průběhu kapitoly vkládáme vysvětlovaný kód. U většiny kapitol je po vysvětlení probrané látky samostatný úkol, na kterém si uživatel může své znalosti ověřit. Jeho kód je uveden na konci příručky. Konečné programy a všechny samostatné úkoly jsou pak přiloženy k příručce jako přílohy a rozděleny do samostatných složek.

7.2.1 Jednotlivé kapitoly

Zde jsou uvedeny názvy jednotlivých kapitol.

- Než začneme aneb co je důležité vědět
- Instalace a spuštění OGRE
- První program a práce s entitami
- Kamera, světla a stíny

- Obloha a mlha
- FrameListener a vstup do programu
- Spouštěcí sekvence OGRE
- Animace

7.3 Popis kapitol

- **Než začneme aneb co je důležité vědět** – První kapitola příručky je čistě teoretická a poskytuje uživateli základní znalosti o OGRE, jako jsou jeho hardwarové a softwarové požadavky. Také je v ní uvedeno jaké základní znalosti jsou od uživatele očekávány
- **Instalace a spuštění OGRE** – V této kapitole je podrobně popsána instalace SDK OGRE a kompilace pomocí Visual Studia 2010.
- **První program a práce s entitami** – Ačkoliv v minulé kapitole byl úplně první program spuštěn, zde je znovu popsán. Také je zde vytvořen tutoriál pro práci s entitami.
- **Kamera, světla a stíny** – Kamera a světla jsou základní objekty scény. Ve vytvořených tutoriálech je popsána jejich základní funkce. I přesto že jsou stíny velmi náročné náročnou částí 3D vykreslování, jejich implementace není zase tak složitá. Navíc úzce souvisí se světlem, zařadila jsem je tedy do této kapitoly také.
- **Obloha a mlha** – Mlha a obloha jsou speciální efekty OGRE, jež jistě ocení každý uživatel. Jejich implementace a použití jsou jednoduché, proto jsou vysvětleny hned ve třetí praktické kapitole.
- **FrameListener a vstup do programu** – V této kapitole je pomocí jednoduchého tutoriálů vysvětlena základní funkce objektu *FrameListener* a jeho využití při vytváření uživatelského vstupu do programu.
- **Spouštěcí sekvence OGRE** - Tato kapitola je už náročnější na pochopení a předpokládá alespoň základní zkušenosti s OGRE. Ty uživatel získal při poctivém

studiu předešlých kapitol. Při procházení tutoriálů se uživatel naučí vytvořit aplikaci využívající OGRE od základů.

- **Animace** – Ačkoliv je tato kapitola jednodušší než ta předešlá, je uvedena až zde protože už nepatří mezi ty nejzákladnější znalosti. Na vytvořeném tutoriálů je provedena demonstrace použití animace v OGRE.

ZÁVĚR

Cílem této bakalářské práce bylo podrobně se seznámit s vizualizačním enginem Ogre a jeho vlastnostmi. Tento engine popsat v jeho aktuální verzi a srovnat jej s jinými dostupnými enginy. Dále pak navrhnout a vytvořit 3D aplikaci demonstrující vlastnosti enginu a příručku tyto vlastnosti popisující.

S uvedeným enginem jsem se pozorně seznámila a popsala jej v teoretické části práce. V druhé kapitole jsem popsala základní vlastnosti tohoto enginu, jako jsou jeho hardwarové a softwarové požadavky, podporované formáty souborů nebo efekty. Protože je OGRE engine vytvořen v programovacím jazyce C++ a pro tvorbu aplikací na něm závislých se tento jazyk používá, popsala jsem v této kapitole i základní vlastnosti jazyka C++ a objektově orientovaného programování.

Popis struktury enginu a některých jeho tříd a objektů je uveden ve třetí kapitole. Zde jsem vycházela z popisu uvedeného v oficiálním manuálu k OGRE pro verzi 1.7.2, který se mi zdál nejvhodnější. V následujících dvou kapitolách jsou pak uvedeny další objekty, třídy a funkce OGRE, které toto schéma nezahrnuje, ovšem já jsem jejich popis pokládala za důležitý.

V kapitole číslo pět byl engine OGRE porovnán s enginy Irrlicht a Panda3D. V tomto srovnání jsem se nesnažila rozhodnutí který z enginů je nejlepší, to opravdu ani nejde. Pouze jsem srovnala jejich vlastnosti, celkový výčet je uveden v tabulce číslo jedna.

V praktické části bakalářské práce jsem se zabývala návrhem a tvorbou 3D aplikace a příručky. Nejprve jsem se zaměřila na 3D aplikaci. Její popis jsem uvedla v kapitole šest. Ten zahrnuje hlavní myšlenku aplikace, popis a převod zdrojů, popis scény aplikace a popis vytvořené aplikace z programového hlediska.

Příručku popisuji v kapitole sedm. Hlavní myšlenkou příručky bylo vytvoření jednoduchého manuálu a dostatečnou obrazovou dokumentací. Koncepce celé příručky zahrnuje základní použití enginu OGRE a jeho jednotlivé funkce jsou popsány pomocí podrobně vysvětlených příkladů.

ZÁVĚR V ANGLIČTINĚ

The aim of this work was closely acquainted with the Ogre rendering engine and its properties. The engine described in the current version and compare it with other available engines. Furthermore, to design and create 3D applications that demonstrate features engine and manual describing these properties.

With that engine, I have carefully read and describe it in the theoretical part. In the second chapter, I described the basic features of this engine, as its hardware and software requirements, supported file formats and effects. Since the OGRE engine developed in C++ and for creating applications dependent on it, this language is used, I have described in this chapter the basic properties of C + + and object-oriented programming.

Description of the structure of the engine and some of his classes and objects is given in the third chapter. Here, I drew from the description contained in the official manual for OGRE version 1.7.2, which seemed most appropriate. The next two chapters are then given additional objects, classes and functions OGRE that this scheme does not, however, the description I found it important.

In chapter five was compared with the Ogre and Irrlicht Panda3D engines. In this comparison, I sought a decision from the engine is the best, no not really. I just compared their properties, the total are listed in table number one.
The practical part of the thesis, I discussed the design and creation of 3D applications and manuals. First, I focused on 3D. The description I have indicated in chapter six. This includes the main idea of the application, description and transfer of resources, scene description and a description of the application created from the application programming point of view.

The manual describes in chapter seven. The main idea was to create a simple guide and manual adequate illustrations. The concept of the whole manual includes the basic use Ogre and its various functions are described in detail explained with examples.

SEZNAM POUŽITÉ LITERATURY

[1] *Ogre* [online]. 2010 [cit. 2011-05-27]. OGRE Web. Dostupné z WWW: <http://www.ogre3d.org >.

[2] *Ogre* [online]. 2010 [cit. 2011-01-27]. OGRE Wiki. Dostupné z WWW: <http://www.ogre3d.org/tikiwiki/tiki-index.php>.

[3] *Ogre* [online]. 2010 [cit. 2011-01-27]. OGRE Manual v1.7. Dostupné z WWW: <http://www.ogre3d.org/docs/manual/>.

[4] KERGER, Felix. *Ogre 3D 1.7 Beginner's Guide*. Birmingham : Published by Packt Publishing Ltd., 2010. 300 s. ISBN 978-1-849512-48-0.

[5] JUNKER, Gregory. *Pro OGRE 3D Programming*. New York : APRESS ACADEMIC, 2006. 312 s. ISBN 978-1-59059-710-1.

[6] KRUGLINSKI, David J. Mistrovství ve Visual C++. Praha : Computer Press, 1999. 854 s. ISBN 80–7226–132–0.

[7] PRATA, Stephen. *Mistrovství v C++ 3. vydání*. Praha : Computer Press, 2007. 1120 s. ISBN 978-80-251-1749-1.

[8] GEBHARDT, Nikolaus, et al. Irrlicht [online]. 2011 [cit. 2011-06-07]. Dostupné z WWW: <http://irrlicht.sourceforge.net/index.html>.

[9] Panda3D [online]. 2011 [cit. 2011-06-07]. Dostupné z WWW: <http://www.panda3d.org/>.

[10] Torus Knot Software Ltd. Ogre [online]. 2008 [cit. 2011-06-07]. Dostupné z WWW: <http://www.ogre3d.org/docs/api/html/ >.

SEZNAM OBRÁZKŮ

Obrázek 1 – Logo enginu OGRE 11
Obrázek 2 - Původní autor enginu Steve Streeting 12
Obrázek 3 – Ukázka oblohy a overlay 15
Obrázek 4 – Struktura OGRE 18
Obrázek 5 – Ukázka entity 23
Obrázek 6 – Ukázka bodového světla 24
Obrázek 7 – Ukázka animace 29
Obrázek 8 – Ukázka aditivních šablonových stínů 31
Obrázek 9 – Ukázka lineární mlhy 34
Obrázek 10 – Loga jednotlivých enginů 37
Obrázek 11 – Graf znázorňující výkon OGRE 43
Obrázek 12 – Blender Exporter 44
Obrázek 13 – Vytvořený povrch 46
Obrázek 14 – princip detekce kolizí s objektem 49

SEZNAM TABULEK

Tabulka 1 - Srovnání vlastností jednotlivých enginů. 38
Tabulka 2 – výkon enginu OGRE 42

SEZNAM PŘÍLOH

PŘÍLOHA P I: DVD

PŘÍLOHA P I: DVD

- /prace: text bakalářské práce
- /prirucka: příručka s vlastními přílohami
- /zdrojove_kody: všechny zdrojové kódy práce
- /aplikace: výsledná aplikace v přiložené knihovně