VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

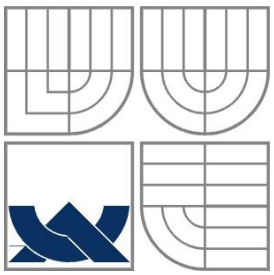
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ALGORITMY ZMĚNY ÚROVNĚ DETAILU (LOD) V KNIHOVNĚ OPENSCENEGRAPH

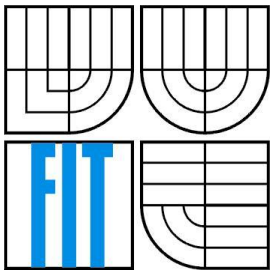
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE ONDŘEJ KOUKOLÍČEK
AUTHOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ALGORITMY ZMĚNY ÚROVNĚ DETAILU (LOD) V KNIHOVNĚ OPENSCENEGRAPH

LEVEL OF DETAIL (LOD) ALGORITHMS IN OPENSCENEGRAPH LIBRARY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ONDŘEJ KOUKOLÍČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN PEČIVA, Ph.D.

BRNO 2012

Abstrakt

Tato práce se zabývá knihovnou OpenSceneGraph (OSG) a jejími možnostmi v oblasti algoritmů level of detail (LOD). Obsahuje seznámení s problematikou snižování úrovně detailů a podrobný popis metod, které se v souvislosti s touto technologií využívají. Dále rozebírá nástroje poskytované knihovnou OSG, které je možné použít při tvorbě aplikací využívající LOD. Hlavní část práce se zabývá popisem návrhu a implementace aplikace, která využívá LOD algoritmy dostupné v knihovně OSG zkombinované s vlastním rozšířením. V závěru je provedeno měření výkonu implementované aplikace a zhodnocení LOD funkcionality knihovny OpenSceneGraph.

Abstract

The aim of this thesis is to explore and evaluate capabilities of the OpenSceneGraph library in the field of level of detail (LOD) algorithms. It contains introduction to the LOD issues and detailed description of techniques that are used in conjunction with this technology. Furthermore it analyses those tools available in OSG, which can be used to create an application utilizing LOD. The main part of the thesis is focused on description of design and implementation of application, which uses the LOD capabilities of OSG as well as own extension to these methods. In the conclusion, the resulting application is subjugated to a benchmark and the OpenSceneGraph library's LOD capabilities are evaluated.

Klíčová slova

OpenSceneGraph, level of detail, LOD , redukce detailů, alpha blending

Keywords

OpenSceneGraph, level of detail, LOD, detail reduction, alpha blending

Citace

Ondřej Koukolíček: Algoritmy změny úrovně detailu (LOD) v knihovně OpenSceneGraph, bakalářská práce, Brno, FIT VUT v Brně, 2012

Algoritmy změny úrovně detailu (LOD) v knihovně OpenSceneGraph

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jana Pečivy, Ph.D.

.....
Ondřej Koukolíček
16. května 2012

Poděkování

Tímto bych chtěl poděkovat vedoucímu mé práce panu Ing. Janu Pečivovi, Ph.D. za cenné rady a ochotu, kterou projevoval při konzultacích. Dále bych rád poděkoval své rodině za podporu při tvorbě této práce.

Obsah

1 Úvod

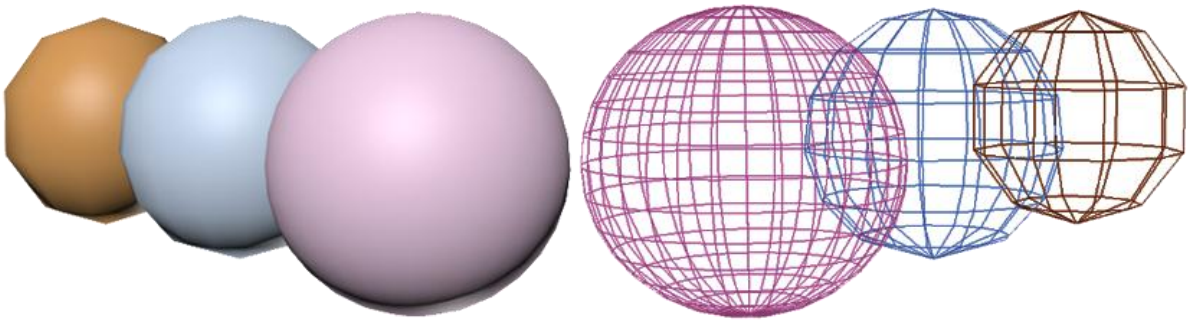
Jednou z úloh oboru počítačové grafiky je vytváření věrně vypadajícího obrazu reality. Nároky na věrnost tohoto zobrazení se v průběhu času stále zvyšují, stejně jako se zvyšuje výkon počítačů, které tyto obrazy vytváří. Mezi těmito dvěma faktory je zachován vyrovnaný poměr a zdá se, že tomu tak bude i do budoucna. V honbě za co nejdokonalejší grafikou je hardware využíván na maximum svého potenciálu. Technologie *level of detail* umožňuje tento potenciál distribuovat mezi objekty grafické scény tak, že redukuje prostředky používané pro vykreslení méně podstatných objektů a umožňuje jejich soustředění na důležitější objekty. Princip této technologie se v nezměněné podobě používá již desítky let, což dokazuje, že problematika zjednodušování grafických scén stále zůstává aktuálním a důležitým tématem.

Tato je práce je zaměřena na technologii level of detail a to konkrétně na její implementaci v knihovně OpenSceneGraph. V práci jsou prozkoumány a popsány všechny metody, které jsou v knihovně implementovány a souvisí se změnami úrovně detailů. Na základě zjištěných informací byla navrhnuta a implementována interaktivní grafická aplikace, která slouží jako demonstrace schopností knihovny OpenSceneGraph v oblasti level of detail. Tato aplikace byla testována a byly vyhodnoceny zkušeností s funkcionalitou ve zkoumané knihovně.

Práce začíná vysvětlením základních principů a metod v kapitole 2. Následuje stručný popis knihovny OpenSceneGraph spolu s podrobným průzkumem všech dostupných nástrojů pro práci s úrovněmi detailů v kapitole 3. Kapitola 4 se zabývá návrhem a implementací grafické aplikace. Podrobně jsou rozebrány všechny významnější prvky aplikace, jako jsou částicové systémy, použité modely a hlavně implementace přepínání detailů. V poslední kapitole je výpis měření provedených na vytvořené aplikaci, porovnání implementovaných režimů level of detail a zhodnocení zkušeností s knihovnou OpenSceneGraph.

2 Technologie level of detail

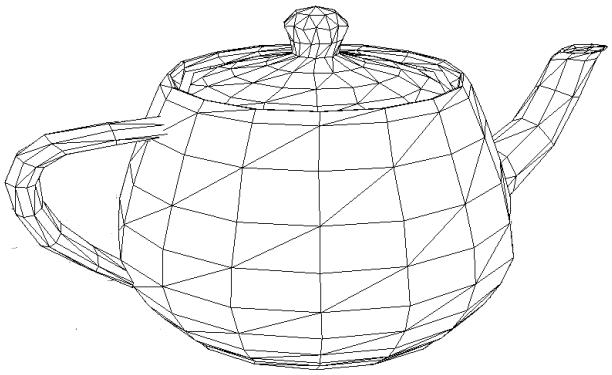
Princip *level of detail* (dále jen LOD) je v zásadě velmi jednoduchý a byl popsán již v roce 1976 Jamesem H. Clarkem v práci „Hierarchical Geometric Models for Visible Surface Algorithms“. Clark poukázal na to, že máme-li ve scéně složité objekty, které chceme vykreslit ve velké vzdálenosti, můžeme je vykreslit ve zjednodušené formě a tím ušetřit systémové prostředky [1]. Z pohledu uživatele se přitom scéna příliš nezmění, protože zjednodušený objekt zabírá jen několik pixelů celého obrazu. Tento princip umožňuje dnešním grafickým aplikacím rychle zobrazovat i velmi náročné scény při zachování přijatelných systémových nároků.



Obrázek 2.1: Základní princip LOD – zjednodušování vzdálených objektů.

V ideálním případě by tedy LOD mělo spravovat všechny objekty scény a na základě určitých kritérií objekty vykreslovat v co nejmenší úrovni detailů, přičemž uživatel by neměl nic poznat. Tento přístup vyžaduje plynulou změnu kvality vykreslovaných objektů, což je výpočetně velmi náročné. Různé typy LOD, které tento problém řeší, jsou popsány v následujících kapitolách.

Důležitým faktem je, že se dnes v počítačové grafice používají převážně polygonální modely. Jde o modely složené z navazujících polygonů, typicky trojúhelníků, které dohromady tvoří celý objekt. Celkové množství polygonů určuje úroveň detailů modelu a z pohledu LOD je podstatné, že je možné toto množství algoritmicky snižovat.



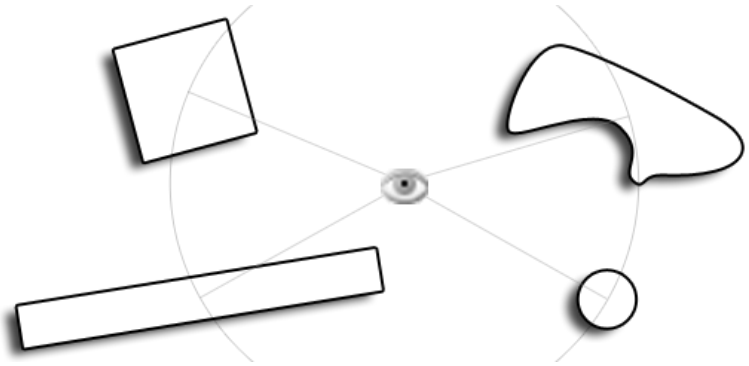
Obrázek 2.2: Typický polygonální model.

Pod pojmem LOD jsou někdy chápány i další techniky používané k šetření systémových prostředků. Svou funkcí nejbližší klasickému LOD je *mipmapping*, který zajišťuje dynamickou změnu kvality textur. Dalším příkladem může být skupina technik zaměřených na šetření paměti na úrovni datových typů použitých pro skladování informací o modelu či celé scéně. Tyto metody jsou uvedeny pouze pro zajímavost a práce se jimi nebude dále zabývat.

2.1 Způsoby výběru úrovně detailů

K rozhodování o výběru úrovně detailů zobrazovaného modelu lze využít dvě základní metody [2], které lze případně doplnit o další rozhodující kritéria.

Prvním přístupem je prosté počítání vzdálenosti objektu od kamery scény, přičemž s rostoucí vzdáleností se snižuje detail objektu. Nevýhodou této metody je fakt, že vždy počítá vzdálenost od jediného bodu objektu, nejčastěji středu, což nemusí u nepravidelných objektů reprezentovat vzdálenost vhodným způsobem. S tímto problémem souvisí i fakt, že metoda nerozlišuje mezi malými a velkými objekty. Celý problém je znázorněn na obrázku 2.3 níže.



Obrázek 2.3: Všechny zobrazené objekty mají stejnou vzdálenost svého středu k oku pozorovatele.

Problémy první metody se snaží napravit další, která určuje zobrazenou kvalitu pomocí počtu pixelů, jež objekt zabírá na obrazovce. Blízké objekty zabírají větší část obrazovky a jsou proto vykresleny detailněji. Velikost na obrazovce je nejčastěji určena pomocí obsahu nejmenšího možného čtverce či kružnice, do kterého se objekt vejde (*bounding box/sphere*). Tato metoda podává lepší výsledky než první zmíněná, což ale vykupuje svou vyšší výpočetní náročností a složitější implementací. Problémem této metody je horší odhad velikost nepravidelných objektů, což je znázorněno na obrázku 2.4.

Obrázek 2.4: Objekt může mít ve stejné vzdálenosti různě velký *bounding box*.

V kombinaci s jednou ze zmíněných metod je možné použít další kritéria rozhodující o LOD objektu:

- Priorita objektu – nezjednodušují se objekty nezbytné pro scénu.
- Důležitost objektu – objekty, na které se soustředí pozorovatel, se nezjednodušují.
- Globální počet polygonů – aplikace limituje celkový počet polygonů.

2.2 Klasifikace LOD

Od vzniku LOD se vyvinuly tři metody jeho použití. Mezi jednotlivými metodami jsou zásadní rozdíly, které je předurčují pro využití v odlišných typech grafických aplikací. Popis těchto metod vychází z [2].

2.2.1 Diskrétní LOD

Ačkoliv je diskrétní LOD ze všech přístupů nejstarší, stále se používá ve většině dnešních grafických aplikací. Algoritmus má nastavené mezní hodnoty, při jejichž překročení zamění zobrazovaný objekt za jiný v příslušné úrovni detailů. Zjednodušené objekty musí být předem nachystány v požadovaném počtu úrovní, což je omezením, ale zároveň i výhodou této metody. Složité modely, jejichž zjednodušení trvá delší čas, je možné přichystat předem a ušetřit tak prostředky v době běhu aplikace. Je také možné vytvořit různý detail modelů manuálně, což je přístup, který se využíval před rozšířením algoritmů pro zjednodušení modelů.

Hlavní nevýhodou metody je dobře viditelná skoková změna kvality modelu a s tím spojený problém, kdy se pozorovatel pohybuje v okolí hranice přepnutí a model se opakovaně mění. Tento problém je možné redukovat zabudováním určitého zpoždění výměny modelu při překročení hranice záměny modelu.

Pro zakrytí skokového přechodu se používá *alpha blending*, což je metoda, která mezi modely vytvoří plynulý přechod pomocí průhlednosti. V době přechodu jsou průhledně vykresleny oba modely zároveň, takže se dočasně zvyšuje náročnost grafické scény.

Obrázek 2.5: *Alpha blending* umožňuje plynulou záměnu modelů.

Obdobou *alpha blendingu* je *geomorphing* popsaný v [3]. Jedná se o složitou metodu, která dokáže z dvojice modelů plynule aproximovat model odvozený z jejich sloučení.

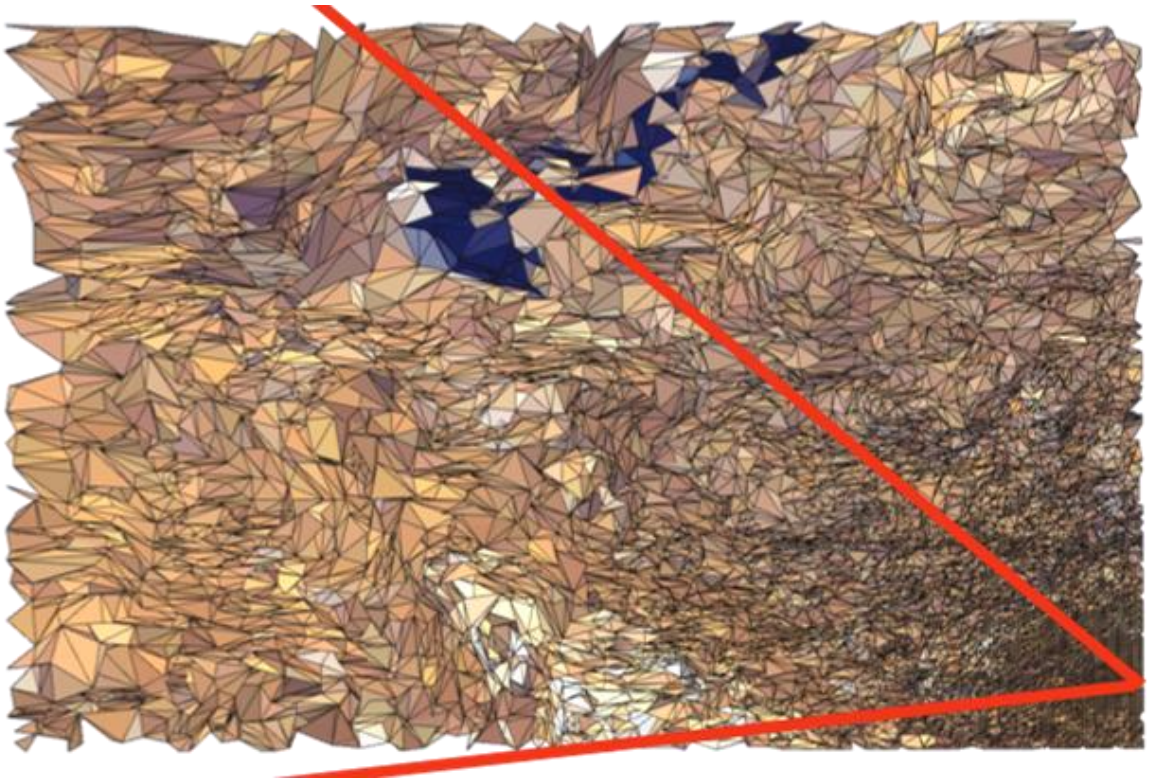
2.2.2 Spojitý LOD

Spojitý LOD, jak již název napovídá, nevyužívá žádné hraniční hodnoty pro změnu detailu. Namísto toho si vytváří v reálném čase model „na míru“ z datové struktury, ve které je model načten. Tento přístup eliminuje skokové přechody diskrétního LOD, ale cenou za to je vyšší výpočetní náročnost a složitost práce s modelem.

Jeden ze způsobů uložení modelu popisují autoři [4]. Využívají stromové struktury, která ukládá jednotlivé shluky blízkých polygonů do svých listových uzlů. Tyto skupiny poté mohou být nahrazeny svým rodičovským uzlem, čímž se dosáhne zjednodušení modelu. Některé z výhod tohoto přístupu jsou:

- Hladké přechody mezi různými úrovněmi detailu.
- Inkrementální přidávání detailu.
- Velikost datové struktury je stejná, nebo menší než původní model.
- Je možné zobrazit jen určité části modelu ve vyšší úrovni detailu.

Rozšířením spojitého LOD je *view dependent* LOD, který využívá výše zmíněného zobrazení určitých částí modelu ve vyšší kvalitě. Obecně řeší problém vykreslení modelů, které jsou příliš velké, aby mohly být celé vykresleny, a zároveň nejdou rozdělit. Často řešený problém je např. zobrazení modelů reprezentujících topografii terénu.



Obrázek 2.6: Ukázka *view dependent* LOD (obrázek převzat z [2])

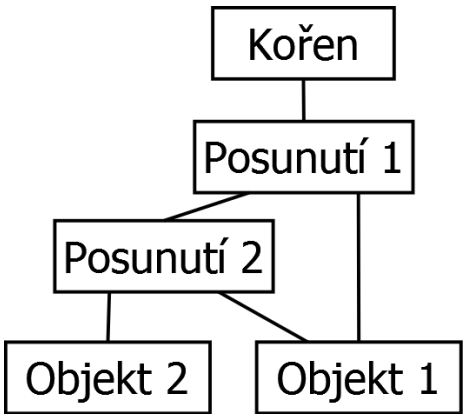
Jak je patrné z obrázku 2.4, účelem je vykreslit poblíž kamery model terénu ve vysoké kvalitě a zároveň zjednodušit jeho vzdálenější části. Podrobné informace o této metodě je možné najít v [5].

3 OpenSceneGraph

OpenSceneGraph (dále jen OSG) je sada grafických nástrojů a knihoven, které fungují jako nástavba nad knihovnou OpenGL. Svým uživatelům zprostředkovává potřebné funkce v jednodušším rozhraní a se zajištěnou správou paměti, což značně zjednodušuje vývoj aplikace.

Dle svých autorů najde OSG využití hlavně v oblasti vizuální simulace, virtuální reality, modelování a tvorby her [6]. Známé projekty, které využívají OSG jsou například letecký simulátor Flight Gear [7], nebo Virtual Terrain Project [8].

Význačnou vlastností OSG je způsob, jakým ukládá veškerá data a nastavení grafické scény. Pro uložení využívá tzv. *scene graph*, což je grafová struktura s hierarchickým řazením vnitřních prvků. Obecně je v grafu vždy kořenový uzel, pod který je dále možné přidávat další uzly různých typů, jako jsou pohybové transformace, skupiny, nebo uzly obsahující načtené modely. Potomci vždy dědí všechny transformace a jiná nastavení provedená jejich rodičovským uzlům, což umožňuje ovlivňovat všechny uzly v dané větvi grafu.



Obrázek 3.1: Příklad grafu scény.

Na obrázku 3.1 můžeme pozorovat zjednodušený graf scény, na kterém bude vysvětlen princip vykreslování. Objekt 2 je potomkem dvou transformací, takže je vykreslen na pozici dané jejich výsledkem. Objekt 1 má v grafu dva rodičovské uzly, což znamená, že bude během vykreslování navštíven dvakrát. Díky tomu bude dvakrát i ve výsledné scéně, jednou s jedinou transformací a podruhé s oběma. Ačkoliv je objekt ve scéně dvakrát, zabírá v tomto případě v paměti prostor pouze jednou.

Součástí OSG je množství tříd, které zajišťují funkcionalitu, kterou je běžně třeba implementovat manuálně. V následující části této kapitoly budou popsány ty z nich, které lze využít při implementaci LOD grafické scény.

3.1 Třída osg::LOD

Tato třída je základem pro veškerou funkcionalitu OSG v oblasti přepínání LOD objektů ve scéně. Implementuje diskrétní LOD, tedy přepíná úroveň detailů objektů v předem daném počtu kroků.

Metody výběru úrovně detailů

Třída podporuje dva způsoby rozhodování o volbě úrovně detailů objektu:

- vzdálenost objektu od kamery;
- počet viditelných pixelů objektu.

Výchozí způsob je vzdálenost od kamery. Metodu je možno změnit voláním:

```
lodNode->setRangeMode(osg::LOD:: DISTANCE_FROM_EYE_POINT);
lodNode->setRangeMode(osg::LOD::PIXEL_SIZE_ON_SCREEN);
```

Vzdálenost se ve výchozím nastavení počítá od středu objektu k oku kamery. V případě, že výchozí střed nevyhovuje, je možné ho manuálně změnit:

```
lodNode->setCenterMode(USER_DEFINED_CENTER);
lodNode->setCenter(osg::Vec3(x, y, z));
```

Počet viditelných pixelů se odhaduje na základě *bounding-sphere* objektu, jejíž velikost je nastavena automaticky, případně ji lze manuálně upravit:

```
lodNode->setRadius(x);
```

Přidání uzlů a nastavení rozsahů

Třída *osg::LOD* je odvozena z třídy *osg::Group*, což napovídá, že umožňuje spravovat skupiny uzlů, které jsou do ní vloženy. Vložený uzel může být objekt třídy *osg::Node*, nebo třídy z ní odvozené, takže ve výsledku téměř libovolný objekt používaný v OSG. Na pořadí vložených uzlů nezáleží.

Pro každý vložený uzel si třída udržuje hodnotu minima a maxima vzdálenosti od kamery či počtu pixelů, při kterých jsou zobrazeny. Tyto hranice je třeba nastavit při vložení uzlu, pomocí přetížené funkce, nebo zpětně:

```
lodNode->addChild(model, rangeMin, rangeMax);

lod->addChild(model2);
lod->setRange(1, rangeMin, rangeMax);
```

Hodnoty minima a maxima jsou sdíleny pro obě metody výběru LOD, takže je třeba je po přepnutí vždy změnit. Jejich význam je totiž přesně opačný. Větší hodnota pro metodu pixelů znamená větší objekt, zatímco větší vzdálenost znamená objekt menší.

V případě, že nejsou rozsahy pro určitý uzel nastaveny, je ignorován a nikdy se nezobrazí. Rozsahy se také mohou překrývat, takže se objevuje více objektů zároveň. Toto lze využít např. v situaci, kdy máme k dispozici oddělený model a přídavné detaily ve více souborech. Hlavní model je zobrazen neustále, zatímco detaily se mohou postupně skrývat či zobrazovat.

LODScale

V OSG je zavedena proměnná *LODScale*, která umožňuje globálně ovlivňovat všechny potomky objektu, kterému je nastavena. Nastavené meze zobrazení objektů jsou násobeny touto hodnotou, takže změníme-li například *LODScale* hlavní kamery na konstantu 2, zvětší se vzdálenosti nastavených intervalů na dvojnásobek. Nastavením hodnoty na číslo menší než jedna je tedy možné snížit množství modelů vykreslovaných ve vyšší kvalitě a zvýšit tak snímkovou frekvenci.

LODScale se automaticky projevuje i u transformací aplikovaných na objekty. V případě zmenšení objektů se *scaling* nastaví tak, aby byly úměrně zmenšeny i rozsahy LOD.

3.2 Třída osg::PagedLOD

PagedLOD je třída, která dědí všechny vlastnosti *osg::LOD*, a rozšiřuje její možnosti o správu načítání modelů. Umožňuje dynamicky načítat modely v aktuálně potřebné úrovni detailů do paměti a uvolnit ji, v době kdy model již potřeba není. Její použití se liší jen funkcí pro přidání modelu do *PagedLOD* uzlu:

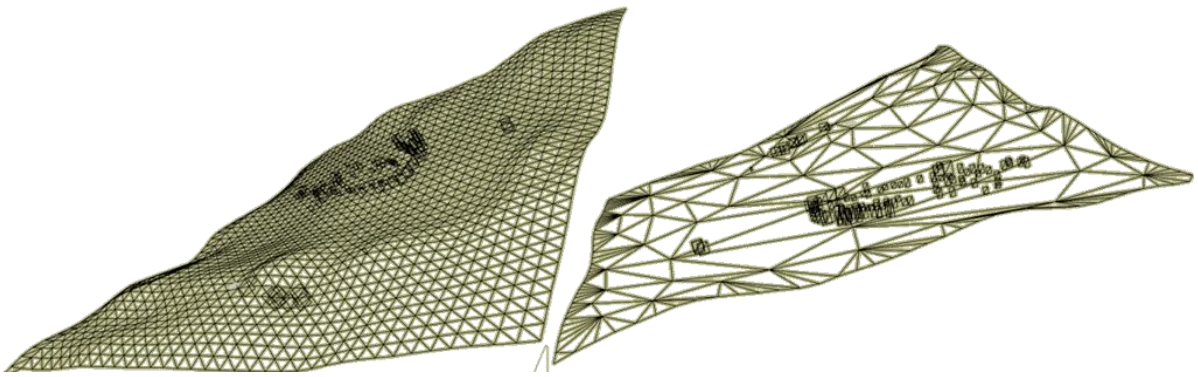
```
pagedLOD->setFileName(0, "model.3ds");
pagedLOD->setRange(0, rangeMin, rangeMax);
```

Příklad výše vytvoří nultý uzel, uloží do něj zvolené meze a cestu k souboru s modelem. Model je načten do paměti až tehdy, kdy jsou splněny podmínky pro jeho zobrazení. Tento způsob je vhodné kombinovat s načtením nízké kvality modelu funkcí *addChild()*, která načítá modely okamžitě a nikdy je neuvolní z paměti. Zabrání se tak problémům v případě, že je třeba vykreslit model a jeho načtení trvá delší dobu.

PagedLOD je lepší alternativou základní třídy *osg::LOD*. Průběžné načítání podstatně zrychluje spuštění aplikace a uvolňování modelů šetří paměť počítače. Tuto třídu je vhodné použít v případě, že aplikace pracuje s větším množstvím modelů, které se ve scéně střídají.

3.3 Třída osgUtil::Simplifier

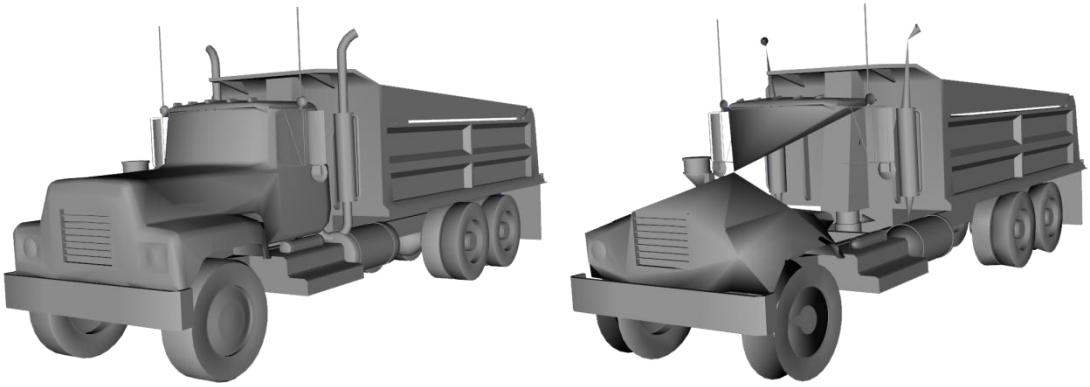
OSG nabízí tuto třídu jako nástroj pro redukci počtu trojúhelníků geometrických objektů a modelů. Do OSG byl přidán především kvůli zjednodušování geometrie terénu, který vytváří třída *osgTerrain*. Pro zjednodušování využívá jen jednoduchý algoritmus eliminace hran, takže v případě použití na složitější modely neposkytuje příliš kvalitní výsledky.



Obrázek 3.2: Redukce polygonů jednoduchého modelu terénu.

Na obrázku 3.2 je velmi jednoduchý model krajiny zmenšený na 10% původního počtu polygonů. Zjednodušení v tomto případě proběhlo v pořádku, je zachován správný tvar modelu a nedošlo k deformacím.

Složitější model nákladního vozu, který vidíme na obrázku 3.3, se naopak správně zjednodušit nepodařilo. Některé části modelu nejsou zjednodušeny vůbec a model je příliš zdeformovaný. Je vidět, že implementovaný algoritmus skutečně není zamýšlen pro použití se složitými modely a tudíž nenajde moc velké využití při implementaci LOD scény.



Obrázek 3.3: Redukce počtu polygonů složitějšího modelu.

Modely lze zjednodušit předem, pomocí přiloženého nástroje *osgconv.exe*, nebo za běhu programu následujícím postupem:

```
osg::Node *modelNode = osgDB::readNodeFile("model.osg");
osgUtil::Simplifier simplifier;
simplifier.setSampleRatio(0.1);
modelNode->accept(simplifier);
```

Nejdříve je do uzlu načten model, poté se vytvoří instance třídy *Simplifier* a nastaví se úroveň zjednodušení na 0.1, což odpovídá devadesáti procentnímu úbytku detailů. Zavoláním metody *accept()* se zjednodušení aplikuje na uzel s modelem.

4 Návrh a implementace aplikace

V této kapitole budou rozebrány vlastnosti, které by výsledná aplikace měla mít a způsob, jakým jsem se rozhodl požadovaných vlastností dosáhnout. Poté budou podrobně rozebrány klíčové prvky aplikace a nástroje, které jsem použil k jejich implementaci. Po celou dobu implementace bylo čerpáno z [9], jakožto příručky pro použití OSG.

4.1 Návrh a požadované vlastnosti aplikace

Cílem práce je vytvořit grafickou aplikaci demonstrující LOD v grafické scéně. Aplikace by k tomu měla využít nástroje, které jsou k dispozici v OSG a ukázat tak jejich schopnosti. Předpokládá se možnost volného pohybu po scéně a množství objektů s aktivním LOD. Jelikož OSG umožňuje použít pouze diskrétního LOD, výsledná aplikace bude rovněž využívat jen této metody.

Na základě daných požadavků jsem se rozhodl pro vytvoření scény s vesmírnou tématikou. Ve vesmírném prostoru je možné rozmístit velké množství objektů a létat mezi nimi, čímž budou dobře viditelné změny LOD jednotlivých objektů. Rozmístění objektů bude generováno náhodně v určitém rozsahu, ve kterém se očekává pohyb pozorovatele.

Jelikož je třeba vyhodnotit a provést měření nad funkcionalitou OSG LOD, aplikace využije všechny metody, které OSG umožňuje. Dále jsem se pro porovnání rozhodl přidat vlastní implementaci manuálního LOD objektů, na kterou bude možné za běhu aplikace přepnout.

4.2 Výsledná aplikace

Popis výsledné aplikace je v práci umístěn před popisem implementace, aby čtenáři usnadnil pochopení popisované části programu.

Výsledná aplikace implementuje uživatelem ovládanou loď, která se nachází ve vesmírném prostoru. Celý prostor je zaplněn řádově tisíci asteroidy a dalšími objekty, které se nezávisle pohybují v prostoru. Tyto objekty dynamicky mění svou úroveň detailů dle nastaveného režimu LOD.

Loď je možno natáčet ve všech třech osách a je možné přidávat a ubírat výkon motoru. Lze přepínat mezi dvěma modely pohybu – základní a realistický. Pomocí částicových efektů jsou vytvořeny trysky motoru a lodní kanón, který ale jinak neslouží žádnému účelu. Kameru, která sleduje loď z pohledu třetí osoby, je možné přepnout do režimu volného rozhlížení.

Aplikace umožňuje za chodu měnit režim LOD, nebo ho zcela vypnout. Dostupné režimy jsou:

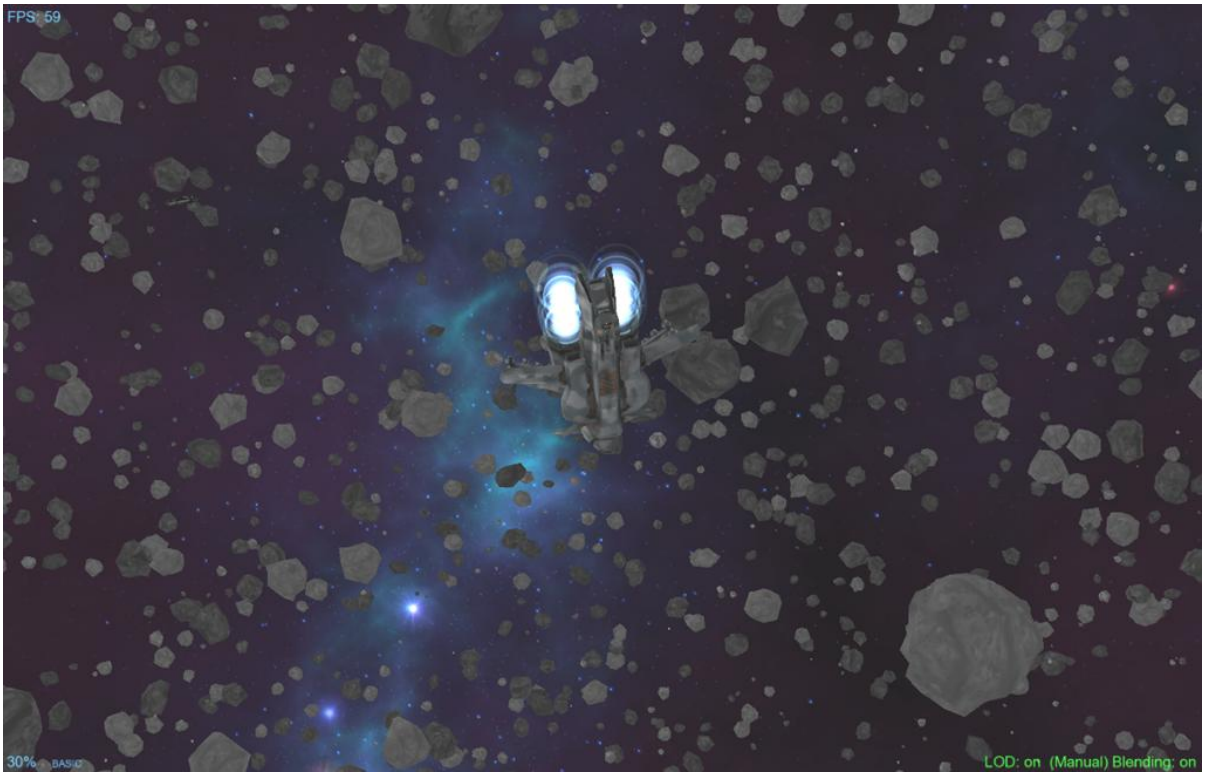
- LOD vypnutý,
- OSG LOD metodou vzdálenosti,
- OSG LOD metodou velikosti objektu,
- manuální LOD,
- manuální LOD s *alpha blendingem*.

Aplikace byla otestována na třech počítačích. Na jednom z počítačů se vyskytl problém s texturami a grafickými artefakty, jehož příčina se nepodařila odhalit. Na zbývajících dvou počítačích aplikace fungovala v pořádku.

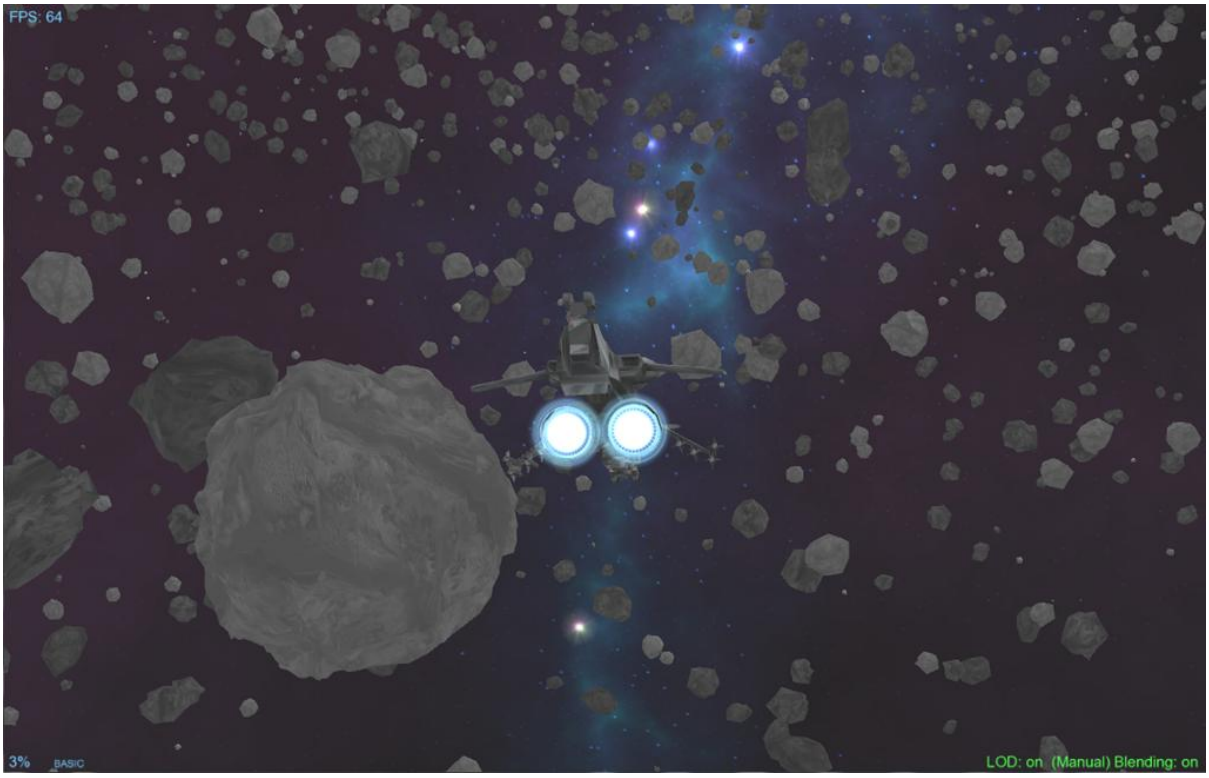
Práce byla zveřejněna na *wiki* školního serveru Merlin v sekci Graphics Projects 2012:

https://merlin.fit.vutbr.cz/wiki/index.php/Graphics_Projects_2012

Na obrázcích 4.1 a 4.2 jsou vidět snímky z dokončené demonstrační aplikace.



Obrázek 4.1: Výsledná aplikace.



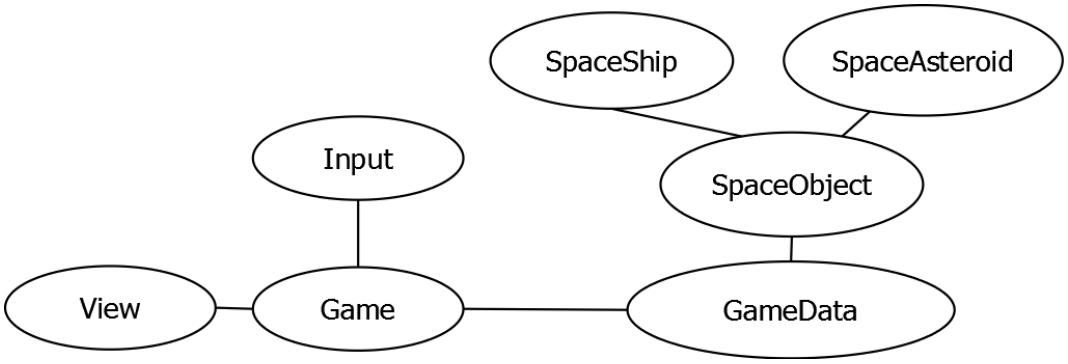
Obrázek 4.2: Výsledná aplikace.

4.3 Implementace aplikace

Aplikace je implementována pod systémem Microsoft Windows 7 64b. K vývoji bylo využito Microsoft Visual Studio 2010 a OpenSceneGraph ve verzi 3.0.1 32b, což je poslední stabilní verze. Výsledná aplikace by měla po přiložení potřebných knihoven fungovat pod všemi aktuálně používanými verzemi Windows.

4.3.1 Stručný popis tříd v aplikaci

Aplikace je rozdělena do několika tříd, z nichž každá zajišťuje určitou část její funkcionality. Závislosti tříd je možné vidět na následujícím obrázku.



Obrázek 4.3: Diagram závislostí tříd.

Třída Game

Jde o řídící třídu, která kontroluje běh programu. Po spuštění programu vytvoří instance ostatních tříd a poté přejde do nekonečného cyklu běhu programu. Třída dále provádí obsluhu akcí vyvolaných klávesnicí či myší a udržuje aktuální nastavení aplikace.

Třída GameData

Tato třída vytváří a spravuje všechny objekty scény. Udržuje pole vytvořených objektů a umožňuje měnit nastavení jejich LOD.

Třída Input

Třída funguje jako správce uživatelského vstupu. Je odvozena z třídy *osgGA::GUIEventHandler*, která umožňuje detekci zmáčknutých kláves pohybu myši.

Třída View

Obsahuje implementaci vytvoření okna aplikace, správu kamer a HUD.

Třída SpaceObject

Základní třída každého objektu, který se vyskytuje ve scéně. Obsahuje funkce pro nastavení pozic objektů a režimu LOD daného objektu.

Třída SpaceShip

Je rozšířením třídy *SpaceObject* pro uživatelem ovládanou loď. Přidává navíc funkce pro vytváření a nastavení částicových systémů a ovládání natočení a pohonu lodi.

Třída SpaceAsteroid

Je také odvozena ze základní třídy *SpaceObject*, kterou ale přizpůsobuje potřebě tvořit velké množství objektů. Dále implementuje samostatný pohyb objektů v předem zvoleném směru a jednoduchou detekci kolizí.

4.3.2 Level of detail

V práci je implementován diskrétní LOD aplikovaný na všechny objekty grafické scény. Je zde ve dvou implementacích, mezi kterými lze za chodu aplikace přepínat, nebo je zcela vypnout. Modely jsou předem přichystány pro každou ze tří úrovní detailů, mezi kterými program přepíná.

Implementace využívající osg::LOD

První implementovaný LOD využívá ke své činnosti připravenou třídu *osg::LOD*. Bylo možné využít i *PagedLOD*, ale rozhodl jsem se, že to není nutné. Vzhledem k tematickému zaměření scény do otevřeného prostoru jsou všechny modely vidět prakticky neustále a nevyplatí se je tedy uvolňovat z paměti. Start aplikace by se pravděpodobně urychlil, ale mohlo by se stát, že nebudou včas načteny modely pro několik prvních okamžiků po spuštění.

Úroveň detailu	Vzdálenost	Počet pixelů
Maximální	0–500	500+
Střední	500–1500	170–500
Nízká	1500+	0–170

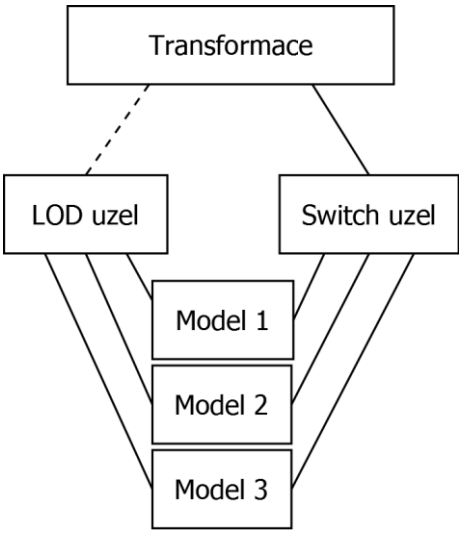
Tabulka 4.1: Hranice pro přepnutí detailu.

Po startu aplikace se pro výběr zobrazeného modelu využívá metoda vzdálenosti objektu od kamery. Na druhou podporovanou metodu je možné přepnout vyhrazenou klávesou. Rozmezí vzdáleností pro přepínání modelů (tabulka 4.1) je nastaveno tak, aby obě metody přepínaly v podobný čas. Nastavené hranice jsou určeny tak, aby byla změna úrovně detailů pro uživatele jasně viditelná.

Vlastní implementace LOD

Do aplikace jsem se rozhodl přidat metodu *alpha blending* pro skrytí přechodu mezi dvěma modely. Bohužel jsem zjistil, že třída *osg::LOD* neumožňuje zjistit aktuální vzdálenost od nastavených přechodů. Tato vzdálenost je pro *alpha blending* nezbytná a bylo by třeba ji zvlášť počítat. Vzhledem k velkému množství objektů ve scéně by počítání této hodnoty dvakrát v každém snímku nebylo vhodné, takže jsem se rozhodl deaktivovat *osg::LOD* a přepínat modely sám (dále pod názvem manuální LOD).

Při přepnutí na manuální LOD se z grafu scény vyřadí uzel s LOD a nahradí se uzlem typu *osg::Switch*. Ten funguje jako skupinový uzel, který umožňuje pro každý přidaný uzel nastavit, zda bude zobrazen nebo ne. Jednotlivé uzly s modely jsou sdíleny mezi *Switch* a LOD uzlem.

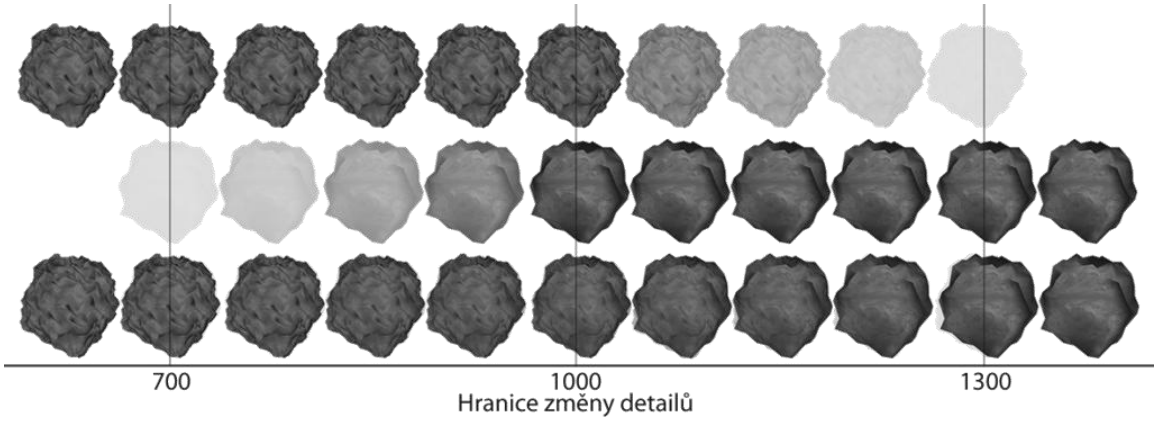


Obrázek 4.4: Přepnutí na manuální LOD.

V každém cyklu aplikace se poté počítají vzdálenosti všech objektů od kamery a aktivuje se příslušná úroveň detailu v uzlu *Switch*.

Alpha blending

Blending se pokouší zajistit plynulý přechod mezi dvěma úrovněmi detailů pomocí postupného zakrývání nahrazovaného modelu jiným, průběžně se „zhmotňujícím“. V aplikaci jsou nastaveny hranice vzdálenosti, při kterých se má přepnout model. Z hranice je vypočítán rozsah o velikosti 30% této hodnoty, při kterém bude prolínání probíhat. Při přechodu se nejdříve postupně zobrazuje nový model až do okamžiku přechodu, poté začne mizet původní model. Celý postup lze vidět níže na obrázku 4.4.



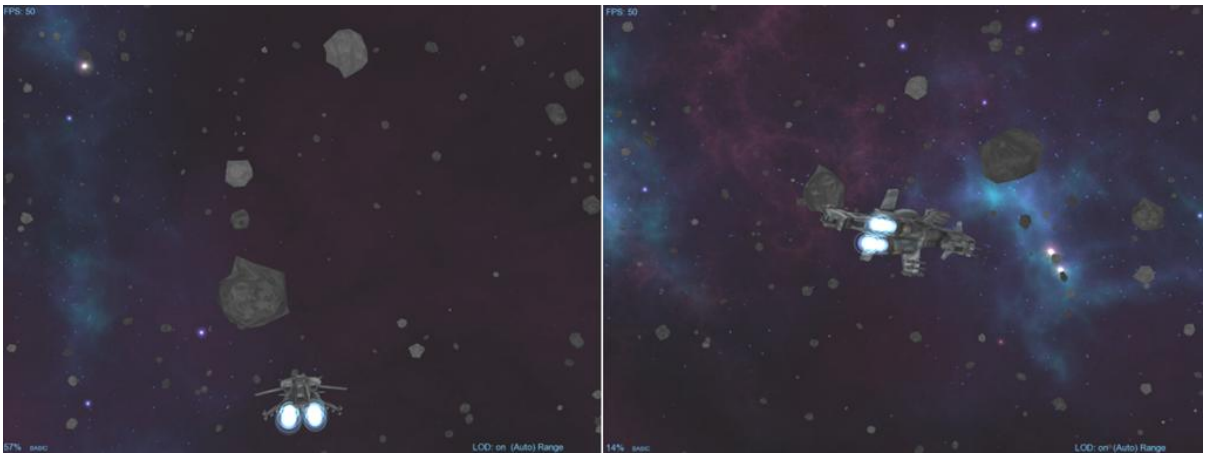
Obrázek 4.5: Průběh implementovaného *alpha blendingu*.

Na obrázku vidíme přepínání dvou modelů asteroidů. V prvních dvou řádcích stav jednotlivých modelů v různých vzdálenostech a v řádku třetím poté jejich kombinaci tak, jak je vidět v aplikaci. V celém intervalu, kdy probíhá *blending*, jsou vykresleny oba dva modely, takže by interval neměl být příliš velký kvůli zvýšené náročnosti.

Při implementaci se vyskytl neočekávaný problém způsobující nekorektní ukládání nastavení stavu jednotlivých objektů (*StateSet*). Následkem toho jsem byl nucen povolit *blending* jen při přechodu z nejvyšší úrovně detailů na střední a naopak. Za tohoto omezení se chyba projevuje méně, nicméně stále lze poměrně často zahlédnout objekty se špatně nastavenou průhledností.

4.3.3 Kamera

V aplikaci jsou implementovány dva typy kamery, klasická kamera z pohledu třetí osoby a režim volného rozhlížení, do kterého se uživatel přepne držením pravého tlačítka myši. Předpokládá se spíše používání volné kamery, protože poskytuje lepší přehled nad celým okolím lodi. Obě kamery dále podporují oddálení či přiblížení pohledu pomocí kolečka myši.



Obrázek 4.6: Kamera z pohledu třetí osoby a volná kamera.

Kamera z pohledu třetí osoby není implementována pomocí maticových transformací, jak je u těchto kamer běžně zvykem. Namísto toho se v každém snímku počítá pozice kamery manuálně na základě aktuální pozice a natočení lodi.

Pohled kamery je poté nastaven funkcí *setViewMatrixAsLookAt*(). Tato funkce nastaví kameru na základě zadané pozice oka kamery, cíle pohledu kamery a vektoru udávající směr nahoru. Tento vektor je vždy nastaven tak, aby kamera udržovala konzistentní směr vzhůru. Při otočení lodi „vzhůru nohama“ se tedy neotočí kamera, ale loď. Tento přístup jsem se rozhodl použít na základě zkušeností s kamerou použitou v různých herních projektech. Otočení kamery je věc, které se vývojáři většinou snaží vyhnout, protože působí ztrátu orientace a dočasné zmatení hráče.

Režim volného rozhlížení počítá pozici kamery stejným způsobem jako výše zmíněný, ale přidává možnost kamerou rotovat kolem lodi. Po zapnutí tohoto režimu se skryje ukazatel myši a umístí se do středu obrazu. V každém snímku se poté detekuje, zda uživatel pohnul myší, směr pohybu a uražená vzdálenost. Na základě zjištěného pohybu se poté změní úhel natočení kamery a spočítá se její nová pozice.

Mezi kamery lze zařadit i *head-up display* (HUD), který je implementován jako kamera, která svým obsahem překrývá předtím vykreslenou scénu. HUD zobrazuje přes scénu užitečné informace o aktuální snímkové frekvenci, nastavení pohonu, nebo o zvoleném režimu LOD.

4.3.4 Pohyb

Pohyb lodi ve scéně byl nejdříve implementován, tak aby napodoboval reálné chování objektů v nulové gravitaci. Během testování se zjistilo, že je tento způsob velmi omezující a nepohodlný, takže byla doplněna druhá metoda ovládání, která jej zjednodušuje. Mezi oběma modely pohybu lze volně přepínat za běhu aplikace.

Třída *SpaceShip* má definovaný vektor rychlosti, jehož jednotlivé složky udávají, jakou rychlostí se loď pohybuje v příslušné ose. V každém snímku je tento vektor přepočítáván na základě nastaveného výkonu motoru, případně aktivního brzdění.

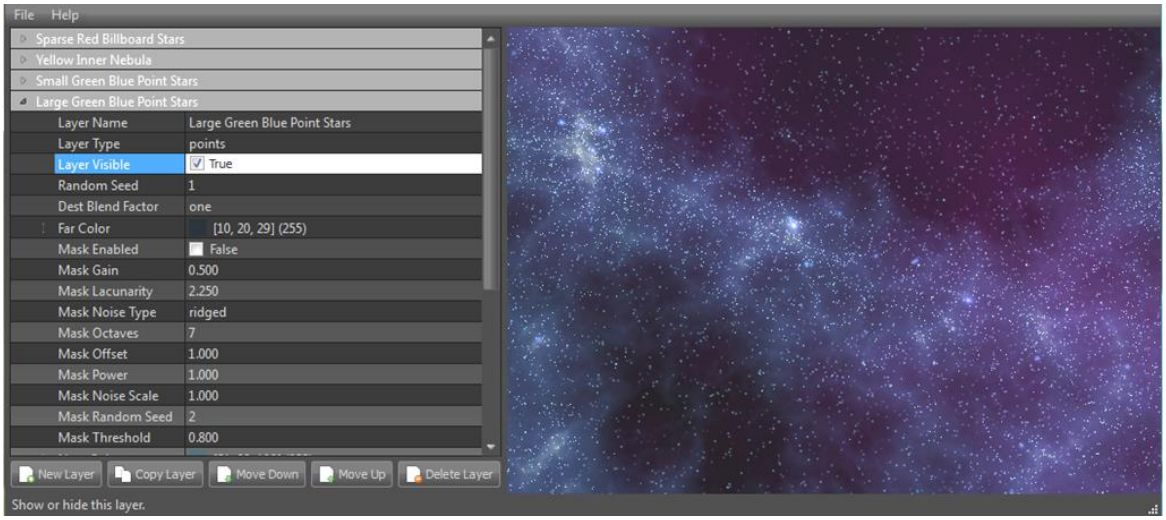
V případě zapnutého pohonu se nejdříve na základě současného natočení lodě vypočítá velikost zrychlení v jednotlivých osách. Toto zrychlení je poté násobeno hodnotou *dt*, což je čas, který uplynul od posledního přepočtu. Získáme přírůstky rychlosti v jednotlivých osách, které jsou poté přičteny příslušným složkám vektoru rychlosti lodi.

V režimu zjednodušeného pohybu je před přičtením původní vektor rychlosti zmenšen na 98% své velikosti, což zajišťuje, že se nově nastavený směr projeví více. Ačkoliv se zmenšení původního vektoru o pouhé dvě procenta zdá málo, změna se projevuje velmi razantně díky častému provádění přepočtu rychlosti.

Získaný vektor rychlosti je nakonec po jednotlivých složkách přičten k aktuální pozici lodi, což zajistí její posunutí při nejbližší aktualizaci transformace lodi.

4.3.5 Hvězdné pozadí

Pro vykreslení pozadí je využita metoda zvaná *skydome*. Umožňuje vytvořit iluzi detailního pozadí scény při minimálním navýšení výpočetní náročnosti. Vychází z metody *skybox*, která pracuje na principu umístění scény do krychle, na jejichž vnitřní strany jsou mapovány navazující textury pozadí. *Skydome* pracuje stejným způsobem, ale namísto krychle využívá kouli, což zlepšuje výsledný efekt. Tato koule se poté pohybuje s kamerou, takže je vždy ve stejné vzdálenosti a vytváří efekt nehybného pozadí.



Obrázek 4.7: Program Spacescape pro tvorbu hvězdných pozadí

Mapovaná textura se skládá ze šesti vzájemně navazujících čtvercových ploch, takže není snadné ji vytvořit. V případě hvězdných pozadí ale naštěstí existuje vynikající volně dostupný program Spacescape [10], který tvorbu výrazně usnadňuje.

Obrázek 4.8: Náhled textury použité pro hvězdné pozadí.

V OSG je k vytvoření pozadí k dispozici třída *osg::TextureCubeMap*, která umí načíst a nastavit čtvercovou texturu. Ta je mapována na kouli, jež se bude pohybovat spolu s kamerou. Koule nemusí být tak velká, aby obalovala celou scénu. Stačí ji vytvořit malou a vypnout pro ni vlastnost GL_DEPTH_TEST, což zajistí, že se vykreslí i objekty, které koule překrývá. Poté je třeba zajistit, aby se koule vykreslila jako první, což se provede přidáním do nejnižší skupiny *RenderBin*. Nakonec se výsledná koule umístí do pohybové transformace, která ji bude udržovat v pozici kolem kamery.

4.3.6 Částicové efekty

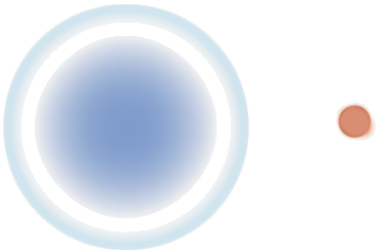
V aplikaci jsou částicové efekty použity pro vykreslení trysky pohonu lodi a také pro lodní kanón. Jelikož se oba jmenované prvky nachází na lodi v párech, jsou použity dva páry částicových systémů.

Každý částicový systém (dále jen PS – *particle system*) v OSG se skládá z následujících součástí:

- **Particle** představuje šablonu částice, která se poté použije pro všechny částice PS. OSG nabízí široké možnosti nastavení vlastností těchto částic a není také problém doimplementovat si vlastní funkcionalitu. Nejdůležitější vlastnosti částice jsou tvar, velikost, doba života a použitá textura.
- **Counter** je jakési počítadlo, které řídí intenzitu vystřelování částic za časovou jednotku. V OSG se nejčastěji využívá *RandomRateCounter*, který vytváří náhodný počet částic dle zadaného rozsahu hodnot.
- **Shooter** umožňuje nastavení vystřelování částic. Jediným implementovaným v OSG je *RadialShooter*, který dovoluje nastavit vertikální a horizontální úhel směru ve kterém bude částice vypuštěna a také rychlost, která bude částici udělena.
- **Emitter** v sobě drží ukazatele na všechny předchozí zmiňované prvky. To značně usnadňuje zpětný přístup k jejich nastavení. OSG obsahuje předdefinovaný *ModularEmitter*, který automaticky tvoří všechny potřebné součásti PS.
- **Program** dává možnost ovlivnit chování částic poté, co byly vypuštěny. V OSG je připraveno několik operátorů, které lze programu přiřadit, nebo je možné vytvořit vlastní. Mezi připravené patří například gravitace, působení větru na částice, či pohyb částic ve vodě.
- **Updater** si udržuje seznam všech částic, které jsou aktuálně vypuštěny, a stará se o jejich pravidelnou aktualizaci.

Všechny v aplikaci použité PS využívají výše zmíněných prvků, s různým nastavením. Efekt pohonu je tvořen zvětšující se částicí, s krátkou dobou života a velmi nízkou rychlostí vypuštění. Při nízké rychlosti pohybu je tvořen jen krátký plamen trysky, zatímco při vyšší se částice za lodí více roztáhnou a plamen je delší. Částice mají povolený *blending*, což znamená, že se vzájemně prolínají a jejich barvy skládají. Je vypuštěno v průměru 50 částic za sekundu.

Systém použitý pro lodní kanón využívá menší částice s velmi vysokou rychlostí vystřelení a delší životností. Vystřelováno je průměrně 90 částic za sekundu.



Obrázek 4.9: Textury použité pro částice. Vlevo tryska pohonu, vpravo projektil zbraně.

Během implementace se vyskytl problém pravděpodobně způsobený omezením ze strany OSG. V případě, že se vytvořené částice pohybují příliš rychle, neproběhne jejich první aktualizace dostatečně rychle a objevují se na špatném místě. Proto nebylo možné zajistit správné chování částic i v případě použití realistického modelu pohybu. Problém je vidět v době, kdy se loď pohybuje vysokou rychlostí a je aktivována střelba. Místo vzniku částic se posouvá dozadu s přibývající rychlostí. Problém se vyskytuje více při snížené snímkové frekvenci a lze předpokládat, že na dostatečně výkonném počítači nebude tak výrazný.

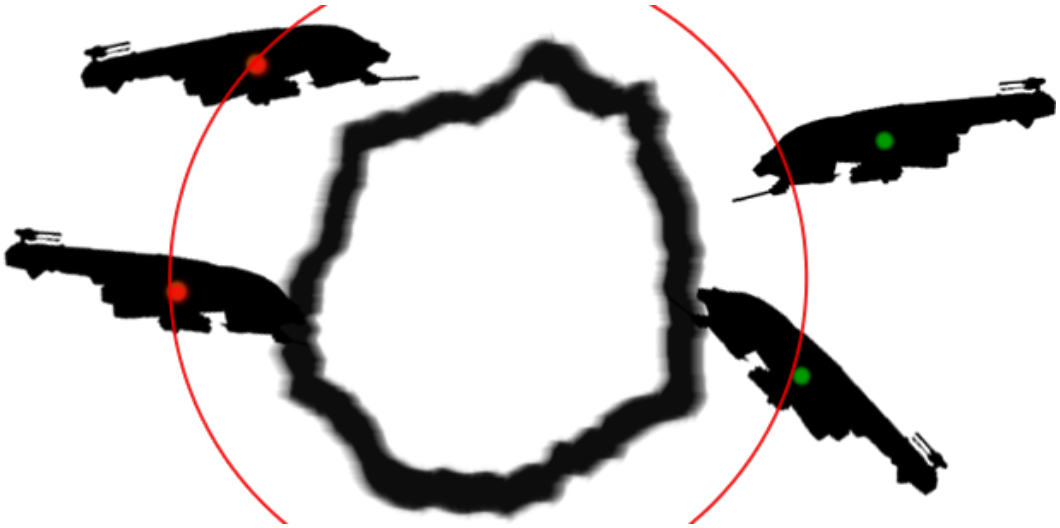
Obrázek 4.10: Výsledné částicové efekty.

Použití částicových systémů může být v závislosti na počtu generovaných částic velmi náročné na výkon počítače. V této práci se to projevuje nutností použití menšího počtu částic pro trysky pohonu. Se zvoleným menším počtem částic zaniká při pohledu z boku efekt spojitého plamene, který by byl s více částicemi zachován.

4.3.7 Kolize

V aplikaci jsou implementovány velmi jednoduché kolize s ovladatelnou lodí. Pro zjednodušení jsou implementovány pouze pro objekty třídy *SpaceAsteroid*, protože použitý způsob by nešlo aplikovat na modely nepravidelné velikosti.

Každý asteroid má při svém vytvoření nadefinovaný poloměr koule, která představuje oblast jeho rozsahu. V každém snímku aplikace je poté kontrolováno, zda středový bod uživatelovy lodě do této koule nezasahuje. Pokud ano, vektor rychlosti lodi je rozpůlen a znegován, čímž dojde k odražení lodi od asteroidu. Zbývající polovina vektoru rychlosti je přičtena k vektoru pohybu zasaženého asteroidu, což způsobí jeho odražení na opačnou stranu.



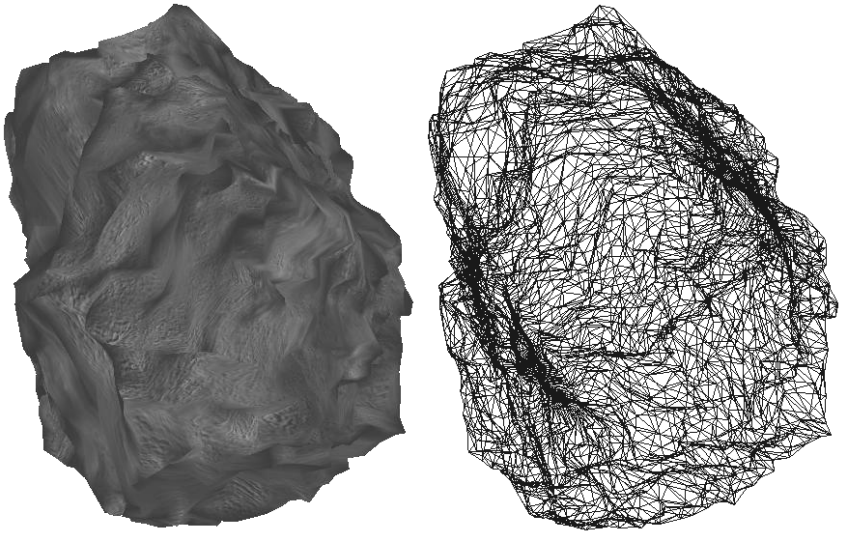
Obrázek 4.11: Jednoduchý model kolizí. Červeně označené lodě kolidují s asteroidem uprostřed.

Tento způsob implementace je samozřejmě jen velmi základní a nepřesný. Na obrázku 4.4 je kupříkladu patrné, že loď vlevo nahoře do asteroidu nijak nezasahuje, ale je i přesto označena za kolidující.

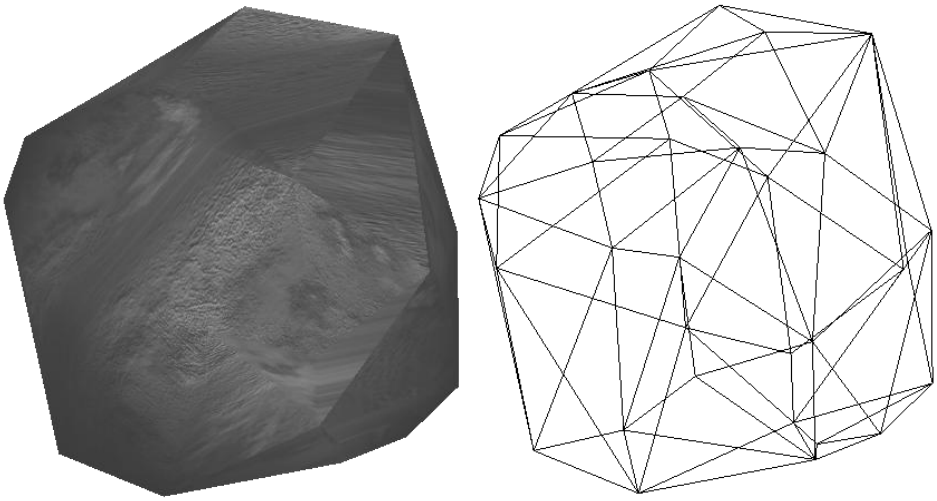
4.3.8 Modely a jejich zjednodušení

Modely použité ve scéně jsou kombinací vlastní tvorby a volně dostupných modelů. K jejich zjednodušení byl použit program Autodesk 3ds Max 2012, konkrétněji jeho nástroj ProOptimizer. Zjednodušení je aplikováno na všechny modely, kromě uživatelem ovládané lodi, která je středem scény a jako taková by se zjednodušovat neměla.

Nejvýznamnější modely scény jsou asteroidy. Tyto objekty jsem vytvořil pomocí již zmíněného programu 3ds Max. Vychází z koule, na kterou byl aplikován náhodný šum. Výsledné modely používají textury dostupné z [11]. Bylo vytvořeno celkem 5 různých modifikací asteroidů.



Obrázek 4.12: Asteroid v nejvyšší úrovni detailů (6240 polygonů).



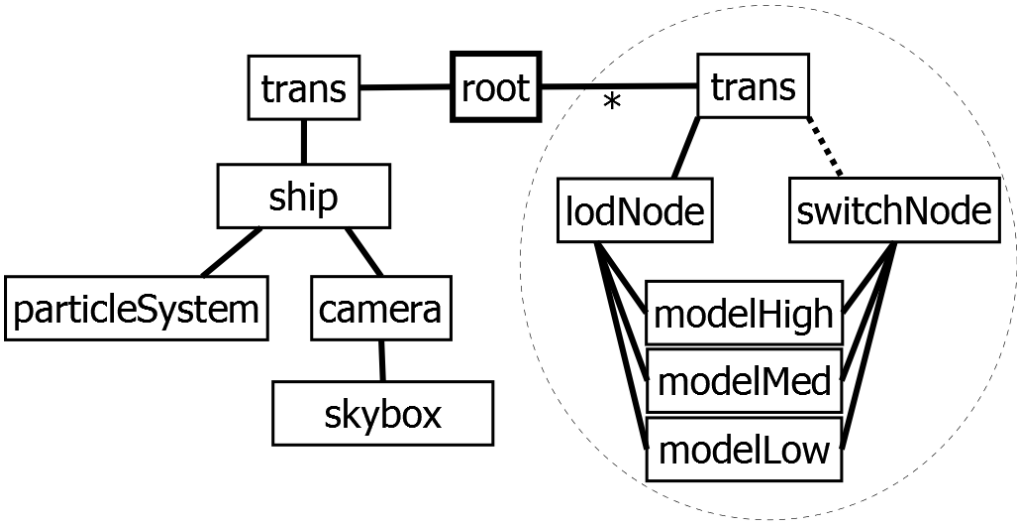
Obrázek 4.13: Asteroid v nejnižší úrovni detailů (58 polygonů).

Do scény dále měly být přidány další modely vesmírných lodí o velmi vysokém počtu polygonů. Toto se bohužel nepodařilo, protože nebylo možné je zjednodušit. Ukázalo se, že nelze použít nástroje pro zjednodušení na velmi složité modely s aplikovanými texturami. Model zjednodušený touto metodou buď nezachovává správně textury, nebo je zjednodušen jen zanedbatelně. V modelech se vyskytoval problém známý jako *z-fighting*. Ten vzniká při prolínání více ploch a způsobuje problikávání textur při pohybu kamery. V praxi se složité modely pro účely LOD zjednodušují manuálně a žádný takto připravený se mezi volně dostupnými modely nepodařilo najít.

I přes výše zmíněné problémy je do scény vložen alespoň jeden menší model lodi, který se podařilo dostatečně kvalitně zjednodušit.

4.3.9 Výsledná scéna

Celou vytvořenou scénu je možné popsat velmi zjednodušeným grafem na obrázku 4.13, jenž zobrazuje objekty ve scéně a jejich závislosti. Na levé straně od kořene scény je ovladatelná loď a části na ní závislé. Vpravo je zakroužkovaná část grafu, která představuje asteroidy vyskytující se ve scéně ve velkém množství.



Obrázek 4.14: Graf objektů ve scéně.

Počet asteroidů je nastaven na 6000. Asteroidy sdílí své modely, takže paměťová náročnost není příliš velká. Při jejich tvorbě je náhodně zvoleno jejich umístění, velikost i směr pohybu a rotace. Každému je přiřazen jeden z pěti vytvořených modelů asteroidů. Jsou generovány v poli tvaru kvádru okolo počáteční pozice uživatelovi lodi.

4.3.10 Běh aplikace

Aplikace je implementována tak, aby byla co nejméně závislá na počtu snímků za sekundu. Všechny operace jako, jsou rotace či pohyby objektů, se tedy odvozují od doby jejich provádění namísto počtu snímků, které operace trvá.

V každém snímku se provádí následující operace:

- obsluha uživatelského vstupu,
- aktualizace HUD,
- aktualizace ovladatelné lodi,
- detekce kolizí a nastavení manuálního LOD (je-li povolen) všech asteroidů,
- aktualizace pohybu asteroidů,
- aktualizace kamery závislé na pozici lodi

Přístup ke všem asteroidům v každém snímku se poměrně výrazně projevuje na snímkové frekvenci aplikace, nicméně v případě manuálního LOD je nutný.

5 Měření a zhodnocení funkcionality

V této kapitole proběhne měření a porovnání výkonnosti aplikace při použití různých metod LOD. Dále bude provedeno zhodnocení nástrojů pro LOD, které knihovna OSG nabízí.

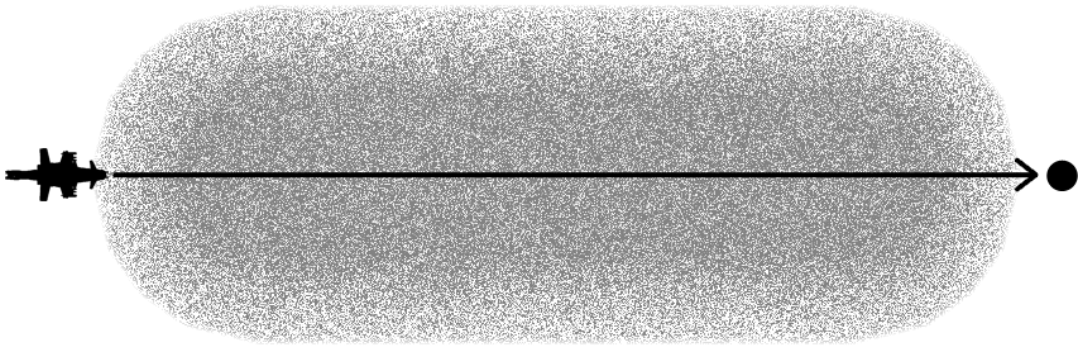
Měření je provedeno pro všechny typy LOD ve shodných podmínkách. Je provedeno na počítači v konfiguraci:

Core 2 Duo T9400 2.53GHz
4 GB ram
NVIDIA GeForce 9650M GT
Windows 7 64b

Výkon aplikace je určen pomocí snímkové frekvence (dále FPS). K měření FPS je použit program Fraps. Doba testování je 70 sekund a probíhá dle následujícího postupu:

- aplikace je spuštěna,
- nastaví se příslušný testovaný režim LOD,
- pohon lodi se nastaví na 100 % výkonu,
- spustí se měření testu.

Pro účely měření je vždy použit stejný *seed* generátoru náhodných čísel při generování asteroidů. Loď je ve výchozí pozici nastavena dle obrázku 5.1, tedy přímo před celým vytvořeným polem asteroidů. Čas je zvolen tak, aby loď stihla proletět skrz pole až do koncového bodu.



Obrázek 5.1: Průběh měření výkonu aplikace – průlet polem asteroidů.

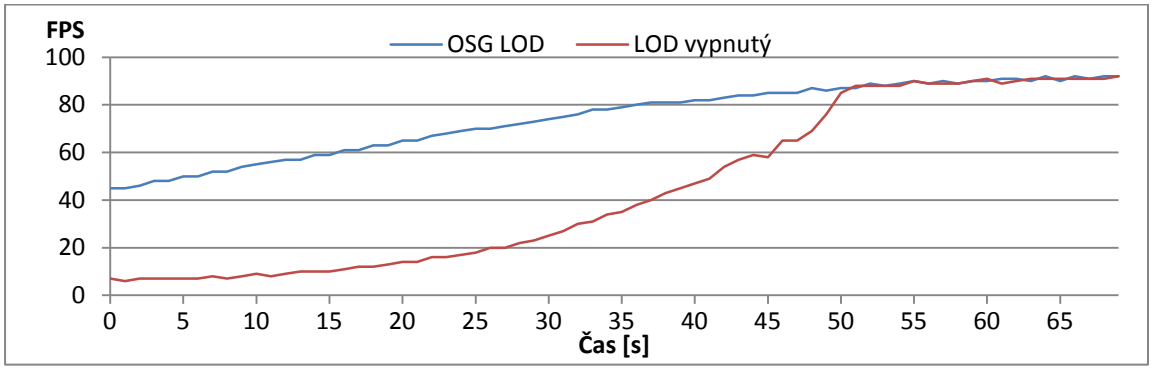
Vzhledem k počáteční pozici lodi se předpokládá postupné zvyšování FPS s ubývajícím počtem objektů, které je třeba vykreslovat v zorném poli před lodí.

Měření proběhlo třikrát pro každý implementovaný režim LOD. Průměr jejich výsledků je možné vidět v tabulce 5.1.

Režim LOD	**Min. FPS**	**Max. FPS**	**Průměrné FPS**
OSG LOD - velikost	43,66	93,33	74,49
OSG LOD - vzdálenost	42,67	90,67	72,67
Manuální LOD	42,67	90	71,98
Manuální LOD - *blending*	41,67	85,33	68,83
LOD vypnutý	5,67	92,67	43,68

Tabulka 5.1: Naměřené hodnoty.

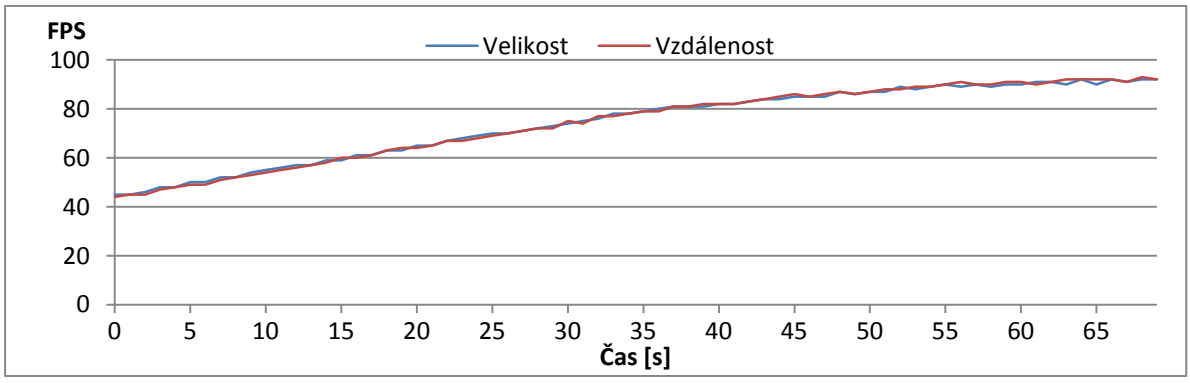
Srovnání výkonu s vypnutým LOD a zapnutým OSG LOD



Obrázek 5.2: Graf srovnání výkonu aplikace se zapnutým a vypnutým LOD.

Se snižujícím se počtem vykreslovaných objektů se zmenšuje rozdíl ve výkonu. Na konci, kdy nejsou vykreslovány žádné objekty, by aplikace s vypnutým LOD mohla mít vyšší FPS, ale zdá se, že režie na přepínání modelů není tak velká, aby se to projevilo.

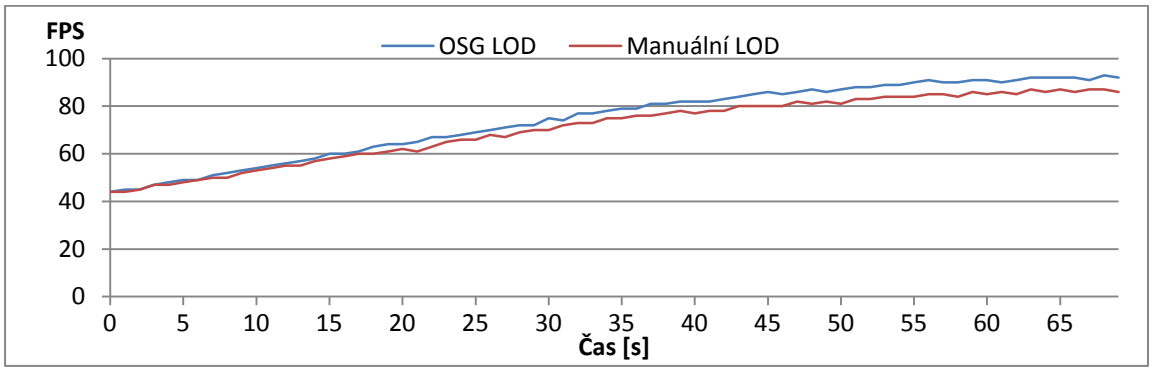
Srovnání metod vzdálenosti a velikosti objektů v OSG LOD



Obrázek 5.3: Graf porovnání metod vzdálenosti a velikosti objektů v OSG LOD.

Z grafu je patrné, že obě metody podávají prakticky stejné výsledky. To je vzhledem k nastavení těchto metod tak, aby fungovaly co možná nejvíc stejně, očekávaný výsledek.

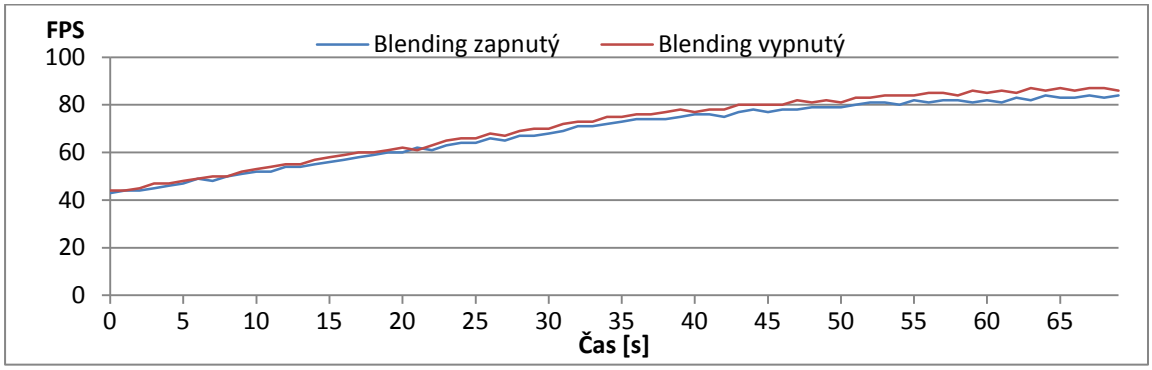
Srovnání OSG LOD a manuální LOD bez blendingu



Obrázek 5.4: Graf porovnání osg LOD s manuálním LOD bez blendingu.

Na začátku testu, kdy je vykreslováno největší množství objektů, je výkon metod srovnatelný. Později při menším počtu objektů, kdy FPS neomezuje grafická karta, ale spíše procesor, je manuální LOD pomalejší. To může být způsobeno horší optimalizací zdrojového kódu.

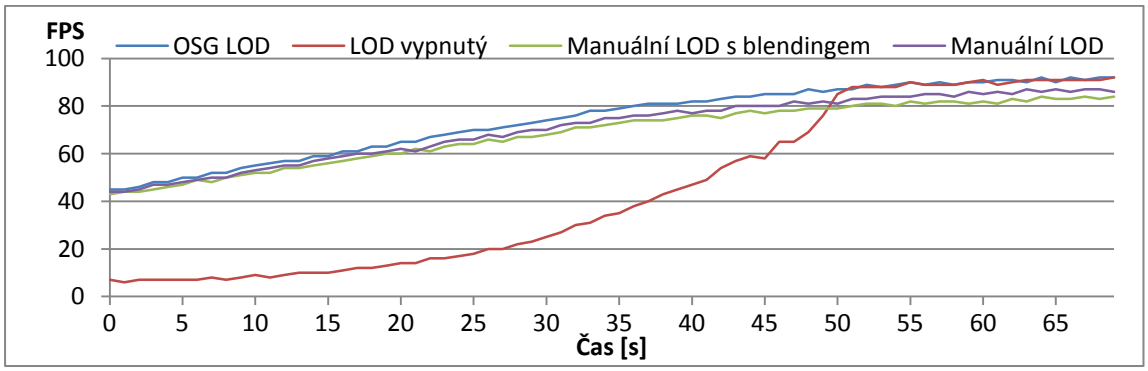
Srovnání manuálního LOD se zapnutým a vypnutým blendingem



Obrázek 5.5: Graf srovnání manuálního LOD se zapnutým a vypnutým blendingem.

Z testu vyplývá, že povolený *blending* se na výkonu prakticky neprojeví. To je způsobeno tím, že v prostoru, kde nejsou asteroidy ve shlucích, se *blending* provádí vždy jen pro malé množství asteroidů a to nepředstavuje příliš velkou zátěž.

Srovnání všech implementovaných metod

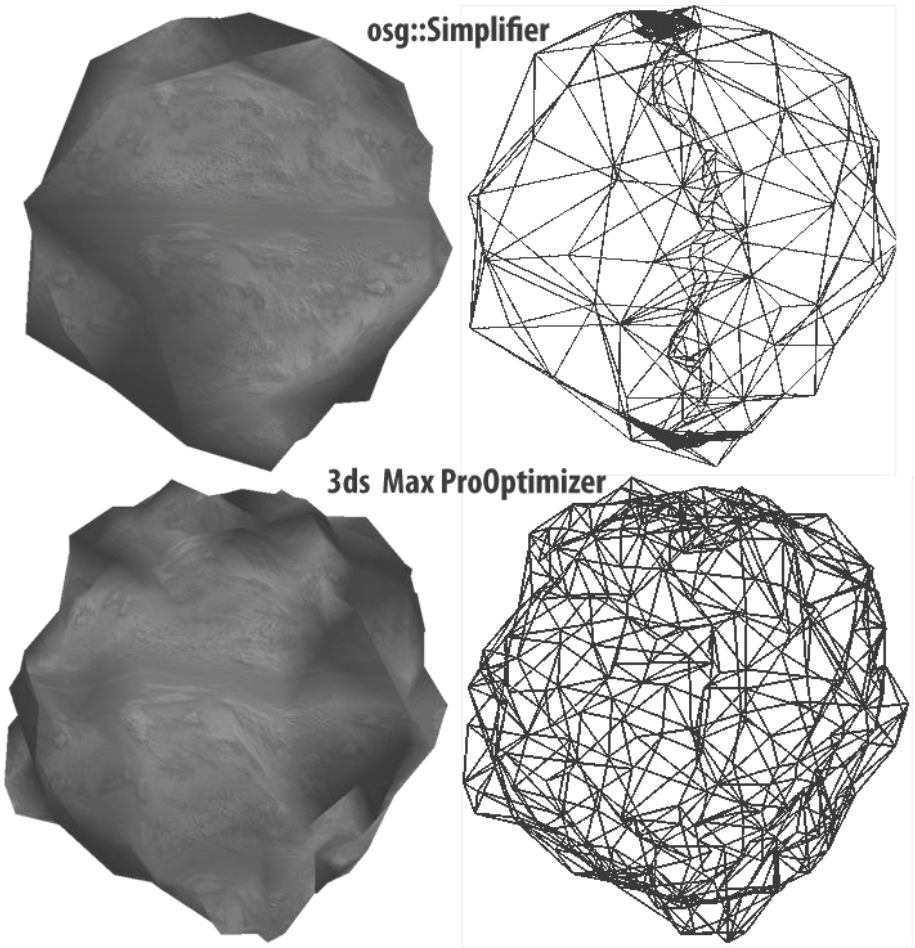


Obrázek 5.6: Graf srovnání všech implementovaných metod.

5.1 Testování třídy osg::Simplifier

Tato třída, která slouží k redukci počtu polygonů modelů, je srovnána s její profesionální alternativou *ProOptimizer*, která je dostupná v rámci programu Autodesk 3ds Max 2012.

V prvním testu je zjednodušení aplikováno na jeden z modelů asteroidů využívaných v aplikaci. Původní model o počtu 6240 polygonů (obr. 4.12) je zjednodušen na 10% tohoto počtu, tedy 624 polygonů.

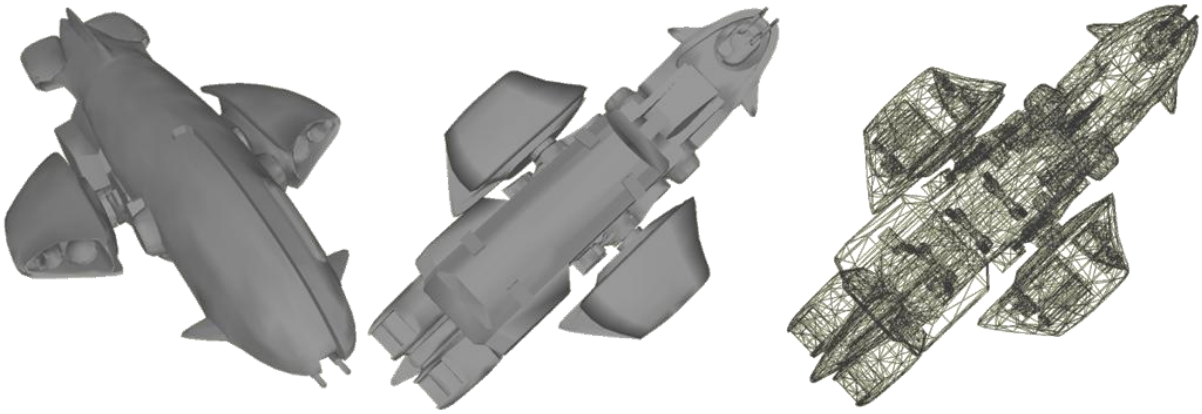


Obrázek 5.7: Zjednodušení modelu v podání OSG a 3ds Max.

Z obrázku 5.7 je patrné, že v případě *ProOptimizeru* se podařilo při stejném počtu polygonů zachovat vyšší úroveň detailu. Počet polygonů ve všech částech modelu je poměrně rovnoměrný. Naproti tomu v případě *Simplifieru* je vidět, že naprostá většina polygonů je soustředěna do vrcholů koule, ze které je asteroid odvozen, což je sice očekávatelné, ale v tomto případě nevhodné.

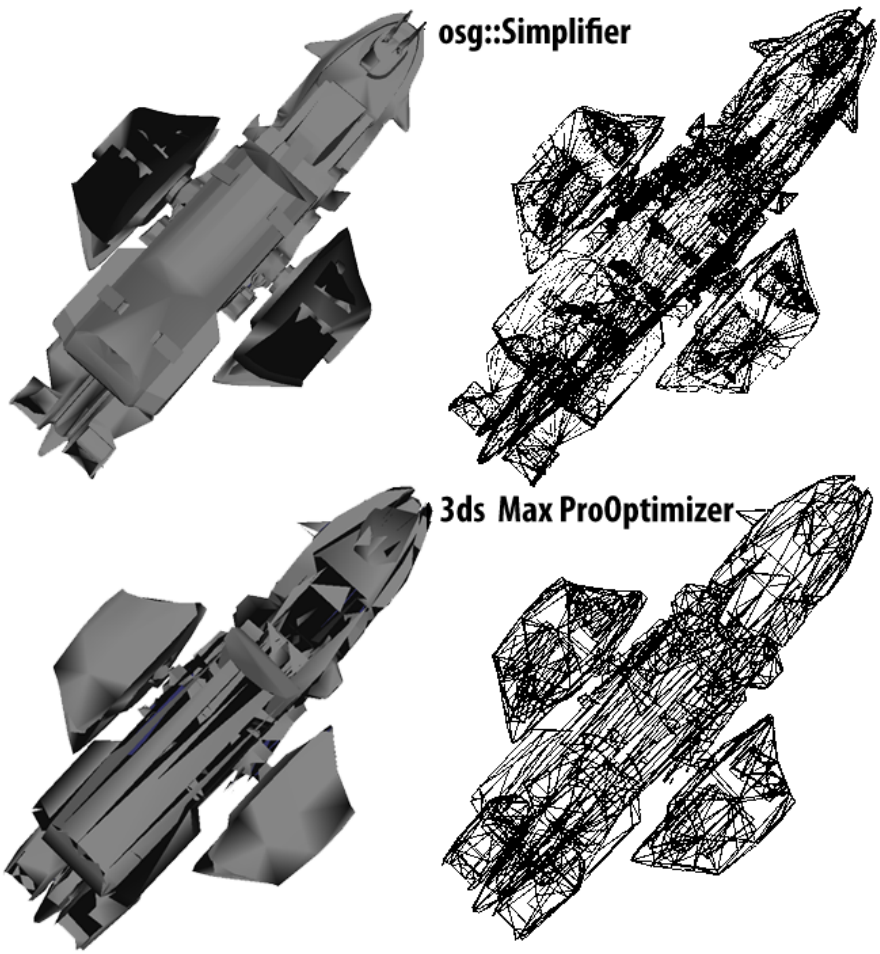
V plášti obou zjednodušených modelů asteroidů vznikla „trhlina“, která byla v případě *Simplifieru* uzavřena (svislá spojnice polygonů ve středu modelu).

Druhý test je proveden na složitějším modelu vesmírné lodi, který můžeme vidět na obrázku 5.8. Celkový počet polygonů modelu je v tomto modelu 12398.



Obrázek 5.8: Model použitý pro druhý test zjednodušení.

V tomto případě již nedokázal *osg::Simplifier* redukovat počet polygonů na požadovaných 10%. Jeho výsledný model měl 8788 polygonů, což je 71% z celkového počtu.



Obrázek 5.9: Druhý test zjednodušení.

V testu se ukazuje, že *osg::Simplifier* může ukončit zjednodušování objektu dřív, než dosáhne požadovaného počtu polygonů. Toto chování vykazoval u většiny modelů a je to také důvod, proč není v implementované aplikaci použit. Problém je pravděpodobně způsoben tím, že je používaný algoritmus nastaven tak, aby nikdy nevytvářel v plášti modelu mezery. Algoritmus je zamýšlen pro zjednodušení modelů terénu, kde by tyto mezery nebyly žádoucí.

5.2 Zhodnocení LOD nástrojů v OSG

Knihovna *OpenSceneGraph* nabízí rozhraní, které umožňuje rychle a jednoduše používat LOD v grafické scéně. K dispozici je implementace diskrétního LOD a dvou metod rozhodování o volbě detailu.

V případě, že uživatel potřebuje funkcionalitu, která přesahuje tyto možnosti, musí si daný problém naimplementovat sám. Může buď využít stávající třídy pro LOD a rozšířit jejich možnosti, nebo si vytvořit implementaci zcela vlastní. Toto může být pro nezkušeného uživatele knihovny neřešitelný problém, ale je to v souladu s myšlenkou knihovny OSG, která není vyvíjena jako všemocný framework typu *Nokia Qt*.

V mém případě jsem tedy kvůli nemožnosti zjistit z LOD uzlu, který z jeho potomků je momentálně aktivní a v jaké vzdálenosti se nachází, musel vytvořit vlastní implementaci LOD. Implementuje rozhodování na základě vzdálenosti objektu od kamery a funguje pro objekty třídy *SpaceObject*. Z hlediska výkonu aplikace dosahuje o něco slabších výsledků než třída *osg::LOD*.

Pro redukci počtu polygonů modelů je k dispozici třída *osg::Simplifier*. Její primární použití je zjednodušování modelů terénu, ale dá se použít i pro jiné modely, které nejsou příliš komplexní. Pro účely LOD je použití této třídy ve většině případů nevhodné, protože není schopna redukovat složité modely, což jsou právě ty, které jsou třeba. Pro zjednodušení je tedy většinou třeba použít externí programy, nebo v ideálním případě modely již vytvářet ve více úrovních detailů.

Funkcionalita *level of detail* poskytovaná knihovnou *OpenSceneGraph* lze vyhodnotit jako dostatečná pro využití v jednoduchých aplikacích. Složité aplikace, jako jsou například počítačové hry, budou pravděpodobně vyžadovat vlastní, rozšířenou implementaci LOD.

6 Závěr

Knihovna *OpenSceneGraph* (OSG) je nástroj, který umožňuje vytvářet grafické aplikace na nižší úrovni abstrakce, než jiné současné alternativy, jako je např. OGRE. Jako taková dovoluje preciznější kontrolu nad vykreslováním grafické scény za cenu zvýšených nároků na vědomosti programátora.

Pro úspěšné využívání OSG je vhodné mít předchozí zkušenosti s knihovnou *OpenGL*, protože dokumentace, která je k dispozici, je velmi stručná a bez těchto znalostí obtížně pochopitelná. Knihovna není příliš rozšířená, takže její uživatelská komunita není příliš velká a množství dostupných návodů ještě menší.

V rámci práce byla podrobně prozkoumána a zhodnocena *level of detail* funkcionalita poskytovaná touto knihovnou. Byla vyhodnocena jako dostatečná pro použití v jednoduchých aplikacích. V případě složitějších projektů, pravděpodobně bude nutné implementovat vlastní algoritmy LOD.

Pro praktické ověření dostupné LOD funkcionality byla navrhnuta a implementována demonstrační aplikace. Jde o složitou scénu s vesmírnou tématikou, ve které uživatel zaujme roli pilota vesmírné lodi v otevřeném prostoru. Ten je vyplněn tisíci pohyblivými asteroidy, které na základě zvoleného režimu LOD přepínají svou úroveň detailů. V aplikaci je kromě všech režimů LOD dostupných v OSG přidána i vlastní implementace LOD algoritmů, která podporuje metodu postupného prolínání dvou úrovní detailů (*alpha blending*). Mezi všemi implementovanými režimy LOD lze za chodu aplikace přepínat a pozorovat tak vliv nastavení na okolní objekty.

Na demonstrační aplikaci bylo provedeno měření výkonu pro všechny režimy LOD. Naměřené hodnoty odpovídaly očekávaným výsledkům. Ukázalo se, že povolení LOD výrazně zvýší snímkovou frekvenci velmi náročných scén, zatímco v případě jednoduché scény je jeho vliv zanedbatelný. Srovnání původní a vlastní implementace LOD dopadla ve prospěch té původní. Rozdíl výkonu je ale velmi malý. Použití *alpha blendingu* má jen minimální negativní vliv na výkon aplikace, což je ale způsobeno jeho povolením jen pro velmi blízké objekty, kterých nikdy není větší množství.

Při ohlédnutí za vývojem aplikace docházím k závěru, že jsem věnoval příliš velké množství času poznávání základních principů, jako je vykreslování jednoduchých tvarů, nebo texturování objektů. Tyto vědomosti se nakonec ve vlastní práci vůbec nevyužily, protože všechny objekty jsou tvořeny načtenými modely.

Kdybych aplikaci vyvíjel se současnými vědomostmi, vyřešil bych jinak způsob, jakým dochází k přepínání mezi režimy LOD. Původně jsem nepředpokládal přidání vlastních algoritmů LOD a výsledná implementace díky tomu není příliš efektivní.

Možným rozšířením projektu by byla implementace spojitého LOD, který by dokázal objekty zjednodušovat plynule. Bylo by třeba implementovat datovou strukturu, která by dokázala načíst modely a v reálném čase poskytovat jejich zjednodušené verze. Jednodušším rozšířením by mohlo být nahrazování nejvzdálenějších objektů pouhou texturou (tzv. *billboard*) nebo implementace mlhy omezující rozhled.

Literatura

[1] CLARK, James H. Hierarchical Geometric Models for Visible-surface Algorithms. In: *Communications of the ACM: a monthly publication of the ACM Publications Office*. New York: Association for Computing Machinery, říjen 1976, s. 547-554. ISSN 0001-0782. Dostupné z: http://accad.osu.edu/~waynec/history/PDFs/clark-vis-surface.pdf

[2] LUEBKE, D, et al. Level of detail for 3D graphics. Vyd. 1. Boston: Morgan Kaufmann, 2003, 390 s. ISBN 15-586-0838-9.

[3] TURK, Greg. Re-tiling polygonal surfaces. In: *ACM SIGGRAPH Computer Graphics*. červenec 1992, s. 55-64. Dostupné z: www.cs.unc.edu/techreports/92-006.pdf

[4] SCHMALSTIEG, Dieter a Gernot SCHAUFLER. Incremental Encoding of Polygonal Models. In: *HICSS '97 Proceedings of the 30th Hawaii International Conference on System Sciences: Software Technology and Architecture*. Washington, DC, USA: IEEE Computer Society Washington, 1997, s. 638-646. ISBN ISBN 0-8186-7862-3/97.

[5] HOPPE, Hugues. Smooth view-dependent level-of-detail control and its application to terrain rendering. In: EBERT, David, H HAGEN a Holly E RUSHMEIER. *Visualization '98*. New York: ACM Press, 1998, s. 33-42. ISBN 1-58113-106-2.

[6] OpenSceneGraph [online]. [cit. 2012-05-06]. Dostupné z: http://www.openscenegraph.org

[7] FlightGear [online]. [cit. 2012-05-06]. Dostupné z: http://www.flightgear.org/

[8] Virtual Terrain Project [online]. [cit. 2012-05-06]. Dostupné z: http://vterrain.org/

[9] WANG, Rui a Xuelei QIAN. OpenSceneGraph 3.0 beginner's guide: create high-performance virtual reality applications with OpenSceneGraph, one of the best 3D graphics engines. 1st pub. Mumbai: Packt Publishing, 2010, 385 s. ISBN 978-184-9512-824.

[10] PETERSON, Alex. Spacescape. [online]. [cit. 2012-05-08]. Dostupné z: http://alexcpeterson.com/spacescape

[11] The Celestia Motherlode: The addon repository for the 3D space simulator Celestia [online]. [cit. 2012-05-09]. Dostupné z: http://www.celestiamotherlode.net/catalog/asteroids.php

Příloha A: Obsah CD

- Zdrojové kódy aplikace
- Zdrojové soubory technické zprávy
- Výsledná aplikace
- Manuál k aplikaci
- Plakát
- Video demonstrace