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 PRO ZJEDNODUŠOVÁNÍ MODELŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE VÁCLAV MILATA
AUTHOR

BRNO 2013

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 PRO ZJEDNODUŠOVÁNÍ MODELŮ

ALGORITHMS FOR MESH SIMPLIFICATION

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — VÁCLAV MILATA

AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN PEČIVA, Ph.D.

SUPERVISOR

BRNO 2013

Abstrakt

Práce se zabývá použitím OpenSceneGraphu, nástroji pro LOD a algoritmy pro zjednodušování modelů. Výsledkem je prototyp hry a aplikace pro demonstraci algoritmů.

Abstract

The topic of the thesis is using of OpenSceneGraph, tools for LOD and algorithms for mesh simplification. The result is a game prototype and a program for demonstrating algorithms.

Klíčová slova

zjednodušování modelů, úroveň detailu, LOD, OpenSceneGraph

Keywords

mesh simplification, level of detail, LOD, OpenSceneGraph

Citace

Václav Milata: Algoritmy pro zjednodušování modelů, bakalářská práce, Brno, FIT VUT v Brně, 2013

Algoritmy pro zjednodušování modelů

Prohlášení

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

. . . . . .
Václav Milata
17. května 2013

Poděkování

Děkuji svému vedoucímu práce, Ing. Janu Pečivovi Ph.D. za rady při řešení projektu a rodině za podporu.

© Václav Milata, 2013.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **3**

2 Level of detail **4**
2.1 Diskrétní LOD . . . 4
2.2 Spojitý LOD . . . 4
2.3 View-dependent LOD . . . 5

3 Zjednodušovací algoritmy **6**
3.1 Lokální operátory . . . 6
3.1.1 Edge collapse . . . 6
3.1.2 Vertex-pair collapse . . . 7
3.1.3 Triangle collapse . . . 7
3.1.4 Cell collapse . . . 8
3.1.5 Vertex removal . . . 9
3.1.6 Polygon merging . . . 9
3.1.7 Obecné nahrazení geometrie . . . 9
3.2 Gobální operátory . . . 9
3.3 Aplikace operátorů . . . 9

4 OpenSceneGraph **10**
4.1 osg::LOD . . . 10
4.2 osgUtil::Simplifier . . . 10

5 Návrh aplikace **12**
5.1 Zjednodušovací algoritmy . . . 12
5.1.1 Uniform grid vertex clustering . . . 12
5.1.2 Floating-cell clustering . . . 13
5.2 Návrh prototypu hry . . . 14
5.3 LOD Maker . . . 14

6 Implementace **15**
6.1 Algoritmy . . . 15
6.1.1 modify() . . . 15
6.1.2 simplify1() – uniform grid vertex clustering . . . 15
6.1.3 simplify2() – floating-cell vertex clustering . . . 16
6.2 Hra . . . 16
6.2.1 Hlavní třídy . . . 17
6.2.2 Terén . . . 17

6.2.3 Modely . . . 17
6.2.4 Letadla . . . 18
6.2.5 Střely . . . 18
6.2.6 Fyzika . . . 18
6.2.7 Detekce kolizí . . . 18
6.2.8 Částicové efekty . . . 19
6.2.9 Plynulost hry . . . 20
6.2.10 Kamera . . . 20
6.3 LOD Maker . . . 21
6.3.1 Ovládání . . . 21

7 Zhodnocení algoritmů **23**
7.1 Rychlost zjednodušení . . . 23

8 Závěr **26**

A Obsah CD **28**

Kapitola 1

Úvod

Trojrozměrné modely v počítačové grafice bývají nejčastěji reprezentovány pomocí mnohoúhelníků (polygonů), především trojúhelníků. Dnešní běžné grafické karty dokáží v reálném čase zobrazovat i stovky tisíc polygonů. Při vykreslování složitých grafických scén nebo modelů však ani to nemusí stačit. Pro rychleší vykreslování je potom třeba snížit počet vykreslovaných polygonů, pokudmožno tak, aby byl výsledný obraz co nejméně odlišný od obrazu s plným počtem polygonů. Toho lze docílit například tím, že grafické kartě nebudou zasílány polygony nacházející se mimo zorný úhel kamery nebo zobrazováním jednotlivých modelů v různých úrovních detailu, například v závislosti na vzdálenosti modelu od kamery. Právě úrovněmi detailu a jejich automatickým vytvářením zjednodušováním původního modelu se zabývá tato práce.

Kromě toho je práce zaměřena na rozhraní OpenSceneGraph a vytvoření aplikace v jazyce C++ s demonstrací jeho nástrojů, použití techniky level of detail a různých algoritmů pro zjednodušování modelů. OpenSceneGraph sám nabízí nástroj pro zjednodušování modelů a kromě něj byly vrámci projektu implementována další dvě řešení.

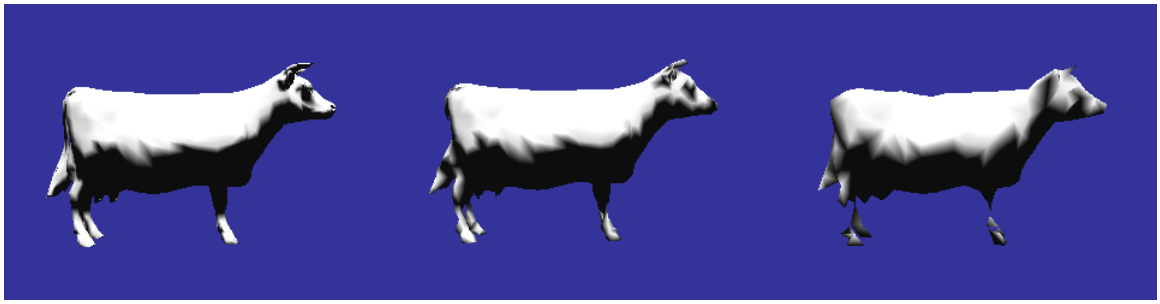
Výsledkem projektu je pak prototyp jednoduché hry s leteckou tématikou, kde hráč ovládá letadlo s možností ničit jiné vzdušné či pozemní objekty. Dále jako součát projektu vznikla aplikace čistě pro zobrazení 3D modelů a demonstraci zjednodušovacích algoritmů na nich, s možností exportu modelu s různými úrovněmi detailu do souboru. Kromě rozhraní OpenSceneGraph využívá také framework Qt, zejména kvůli ovládacím prvkům (widgetům), které OpenSceneGraph nabízí pouze velmi omezeně.

Práce je tedy komplexní ukázkou použití rozhraní OSG, Qt a implementace algoritmů pro zjednodušování modelů.

Kapitola 2

Level of detail

Základním konceptem techniky *level of detail* (LOD) je zobrazování modelu s různou úrovní detailů v závislosti na faktorech, jako jsou například vzdálenost objektu od kamery nebo velikost vyrenderovaného objektu na obrazovce [5]. Když je model blízko kamery a zabírá skoro celou vykreslovací plochu, je příhodné, aby byl velmi detailní. Když je však model daleko od kamery a vykreslí se pouze na několik desítek pixelů, je vykreslování modelu v plném detailu neefektivní. Při správném použití techniky by mělo být vykreslování rychlejší a uživatel by neměl změnu detailu nijak poznat.



Obrázek 2.1: Model krávy ve třech různých úrovních detailu: 5804, 2184 a 730 trojúhelníků

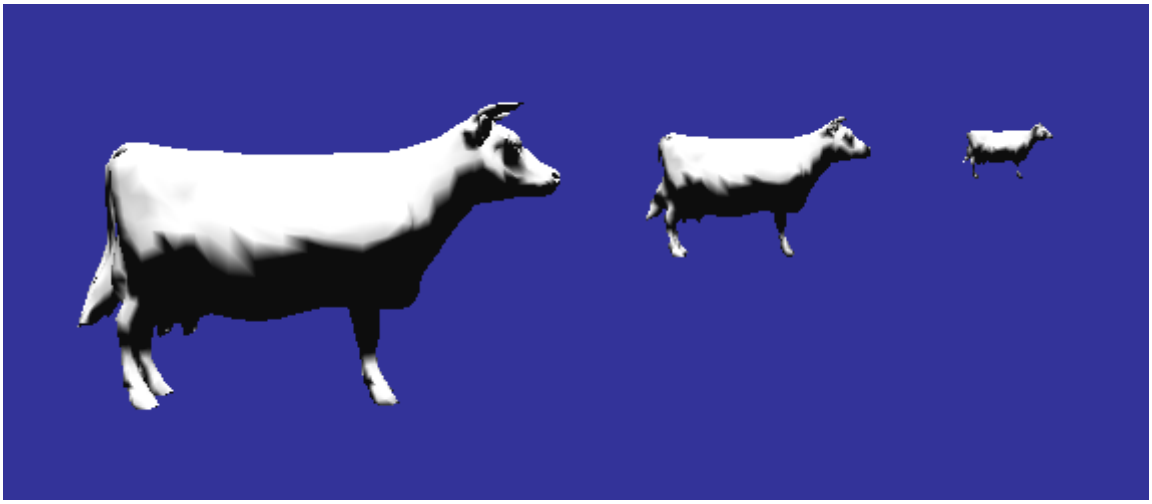
Techniku LOD lze aplikovat třemi základními způsoby [5]:

2.1 Diskrétní LOD

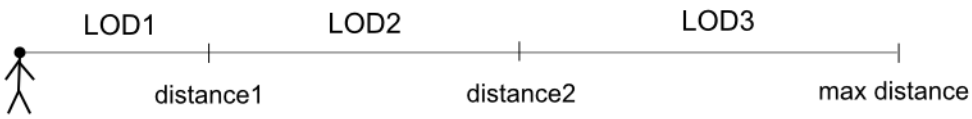
Tento přístup je nejstarší a i v dnešní době stále nejpoužívanější. Princip spočívá ve vytvoření několika verzí modelu, každého s jinou úrovní detailu ještě před samotným během aplikace. Všechny části modelu by měly být zjednodušeny stejnou mírou. Jednotlivé úrovně detailu modelu se pak přepínají dynamicky za běhu aplikace.

2.2 Spojitý LOD

Narozdíl od diskrétního lodu se požadovaná úroveň detailu generuje za běhu programu z datové struktury k tomu určené. Výhodou tohoto přístupu je velmi plynulá změna úrovně detailu a fakt, že před samotným během aplikace není potřeba nic generovat. Metoda je však náročnější na výpočetní výkon a implementaci.



Obrázek 2.2: Modely z obrázku 2.1 v různých vzdálenostech.



Obrázek 2.3: Diskrétní LOD v závislosti na vzdálenosti od pozorovatele

2.3 View-dependent LOD

Přístup rozšiřuje spojitý LOD, kdy je dynamicky vybírána nejvhodnější úroveň detailu v závislosti na umístění a pohledu kamery. Různé části jednoho objektu pak mají různou úroveň detailu. Toho se využívá například u terénu, kdy část modelu bližší pozorovateli má lepší rozlišení než vzdálené části.

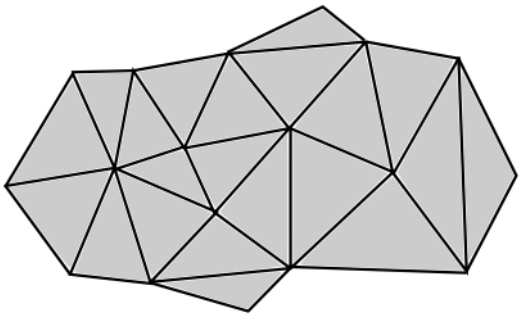
Kapitola 3

Zjednodušovací algoritmy

Polygonální modely jsou určeny dvěma složkami: geometrií, reprezentovanou vrcholy a propojením, reprezentovaným hranami. Hrany spojují vrcholy a vytvářejí stěny. Propojení určuje topologii sítě, tedy počet děr, tunelů a dutin modelu. [6]

Přestože je řeč o polygonálních modelech, pro zjednodušení budeme předpokládat, že je model tvořen trojúhelníky. Ty jsou nejpoužívanější primitivou v 3D grafice a jakýkoliv jiný polygon je možné na trojúhelníky rozdělit.

Většina zjednodušovacích algoritmů navíc předpokládá, že se model sestává ze sítě navazujících trojúhelníků, přičemž každá hrana je sdílena dvěma trojúhelníky (tzv. *manifold mesh*). To však u zjednodušování některých modelů může činit potíže, jak si ukážeme později.



Obrázek 3.1: Manifold mesh – síť navazujícít trojúhelníků

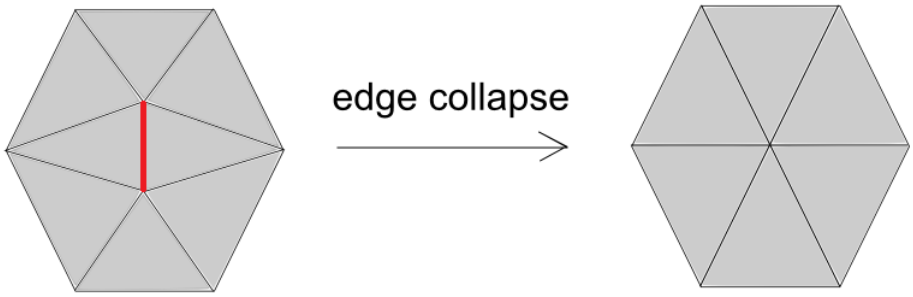
3.1 Lokální operátory

Lokální operátory jsou nízkoúrovňové operátory, které zjednoduší trojúhelníkovou síť o malé množství. Tyto operátory jsou pak součástí komplexních zjednodušovacíh algoritmů. [5]

3.1.1 Edge collapse

Edge collapse (zhroucení hrany) je operátor, který vybranou hranu v síti nahradí jedním vrcholem. Hrana se odstraní, stejně jako trojúhelníky, jichž byla součástí.

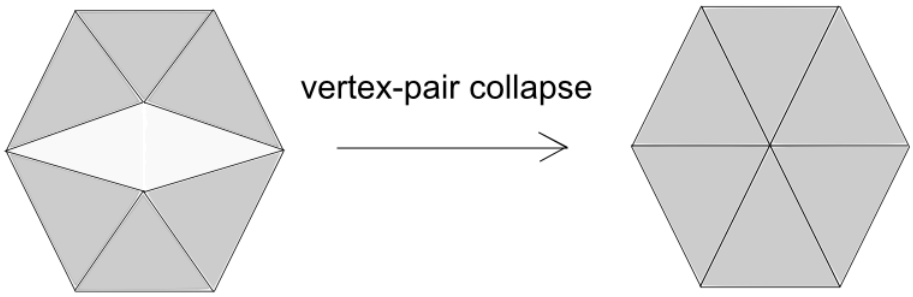
Operátor můžeme dále rozdělit na *half-edge collapse* a *full-edge collapse*. V prvním případě je novým vrcholem jeden z vrcholů odstraněné hrany. V druhém případě se poloha nového vrcholu vypočítá.



Obrázek 3.2: Full-edge collapse

3.1.2 Vertex-pair collapse

Tento operátor podobně jako v předchozím případě nahradí dva vrcholy jedním. Vrcholy však nejsou spojeny hranou. Nezanikne tedy žádná hrana ani žádné trojúhelníky. Je však umožněno spojení jiných trojúhelníků, uzavírají se díry v modelu, a tím vzniká prostor pro další zjednodušování.

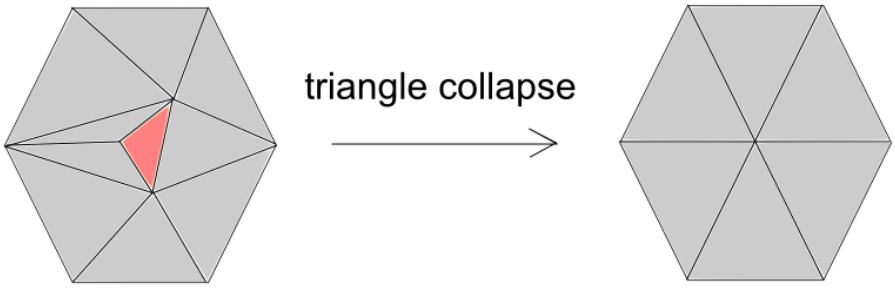


Obrázek 3.3: Vertex-pair collapse

3.1.3 Triangle collapse

V tomto případě dojde k nahrazení celého jednoho trojúhelníku vrcholem. Spolu se zaniknuvším trojúhelníkem se odstraní i tři přilehlé trojúhelníky. Operace je ekvivalentem dvojitého

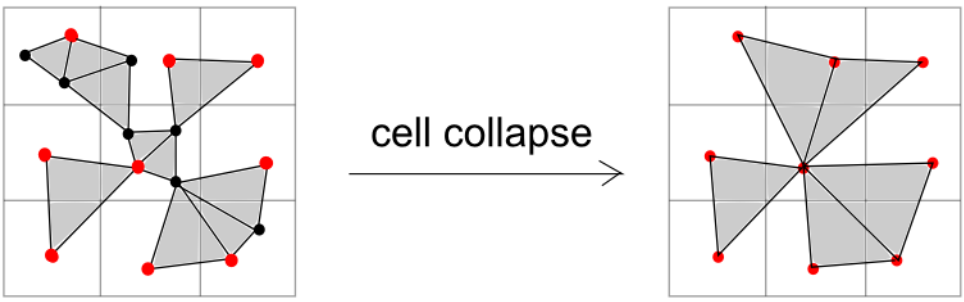
edge collapse.



Obrázek 3.4: Triangle collapse

3.1.4 Cell collapse

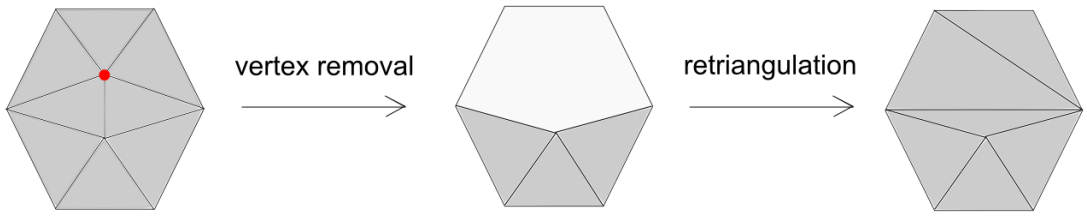
Operátor zjednoduší síť trojúhelníků tak, že vrcholy v určitém objemu (buňce) spojí do jediného vrcholu. Buňka může být například součástí 3D mřížky. Nový vrchol buňky může být jedním ze starých vrcholů (vybraný např. nějakou hodnotící funkcí) nebo může vzniknout nový (průměr pozice původních vrcholů, střed buňky). Míra zjednodušení závisí na rozlišení mřížky. Nevýhodou také je, že různé natočení a posunutí shlukovací mřížky nad modelem bude produkovat odlišné výsledky.



Obrázek 3.5: Cell collapse – červeně jsou vyznačeny vrcholy s nejvyšším ohodnocením vrámci buňky (pouze názorně, nemusí odpovídat reálnému ohodnocení)

3.1.5 Vertex removal

Operátor odstraní vrchol společně s hranami, kterých je součástí a vzniklá díra je následně zaplněna trojúhelníky. Pod tuto metodu de facto spadá i *half-edge collapse*.



Obrázek 3.6: Vertex removal

3.1.6 Polygon merging

Operátor spojí několik polygonů, které jsou posléze nahrazeny sítí trojúhelníků. Metoda je podobná jako v předchozím případě, je však obecnější. Spojením několika polygonů může dojít k odstranění více vrcholů. Může být aplikována obecně na mnohoúhelníky.

3.1.7 Obecné nahrazení geometrie

Operátor spočívá v nahrazení sousedících trojúhelníků zjednodušenou sadou trojúhelníků se stejným ohraničením. Je velmi obecný a spadají pod něj jak *edge collapse*, tak i *vertex removal* nebo *polygon merging*.

3.2 Gobální operátory

Globální operátory se zabývají modifikováním topologie modelu. Topologií jsou myšleny díry, tunely a dutiny modelu a globální operátory rozhodují o tom, zda je zanechávat nebo zjednodušovat. To může být důležitá otázka například u modelů ze strojírenského nebo lékařského prostředí. V případě této práce, která se zaměřuje na zjednodušování modelů ve hře však topologie modelu nehraje výraznější roli a její zjednodušování tedy není dále rozebíráno.

3.3 Aplikace operátorů

Na vyšší úrovni je třeba vyřešit, jakým způsobem operátory aplikovat. Většinou se používá pouze jeden operátor, ačkoliv je možné jich vrámci jednoho algoritmu použít více. Algoritmus může například operace aplikovat v libovolném pořadí (neoptimalizující). Další možností je spočítat cenu jednotlivých operací (např. podle chyby, která operací vznikne) a nesledně postupovat od nejmenší. Podobně může cenu pouze odhadnout a snížit tím nároky na výpočetní výkon. Každou operací se navíc zpravidla změní cena okolních operací a je vhodné ji nějakým způsobem přepočítat.

Kapitola 4

OpenSceneGraph

OpenSceneGraph (OSG) je multiplatformní open-source 3D grafické API (application programming interface). Je napsaný v C++ a slouží jako abstraktní nadstavba nad nízkoúrovňovým OpenGL [7][1].

Scene graph je obecná datová struktura definující prostorové a logické uspořádání pro efektivní správu a rendering grafických dat. Typicky je reprezentována hierarchickým grafem, který obsahuje sbírku uzlů.

Na vrcholu grafu se nachází kořenový uzel, pod ním zpravidla skupinové uzly (*osg::Group*) obsahující libovolný počet dceřinných uzlů a na nejspodnější vrstvě se nacházejí listové uzly, které již žádné potomky nemají. Každý uzel v grafu dědí vlastnosti svého rodiče. Skupinový uzel a operace na něm tedy mají vliv na všechny své potomky.

Na listech grafu se nacházejí geody (*osg::Geode*), které obsahují grafická data. Nás pak zajímá především třída *osg::Geometry*, spravovaná geodami, která obsahuje geometrická data. Jsou to jednak vrcholy a informace o nich (uchovávané v polích *osg::Array*) a sady primitiv (*osg::PrimitiveSet*), uchovávající informace o mnohoúhelnících, čarách a bodech.

OSG také nabízí nástroje pro LOD a zjednodušování modelů.

4.1 osg::LOD

Správu úrovně detailu modelů v OSG poskytuje třída *osg::LOD*. Je odvozena od skupinového uzlu (*osg::Group*) a každý dceřinný uzel má určen rozsah, kdy je viditelný. Jedná se tedy o diskrétní LOD. Rozsah může být určen buď vzdáleností středu objektu od kamery nebo velikostí (počtem pixelů), kterou vyrenderovaný model zabírá na obrazovce.

Z *osg::LOD* pak také dědí třída *osg::PagedLOD*, která jednotlivé úrovně detailu načítá za běhu programu. V tomto projektu jejího použití nebylo potřeba.

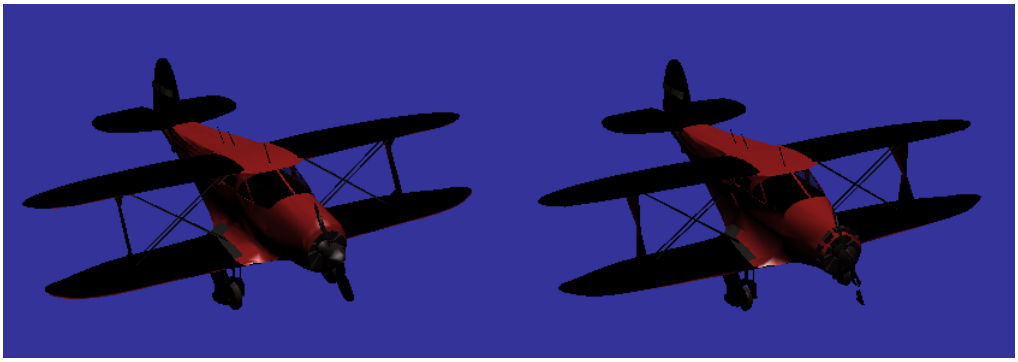
Třída sama však neposkytuje funkcionalitu pro zjednodušení modelu.

4.2 osgUtil::Simplifier

Knihovna osgUtil obsahuje třídu Simplifier pro zjednodušování modelů. Je možné nastavit míru zjednodušení (v rozsahu 0.0 – 1.0 znamená vyšší číslo vyšší úroveň detailu) a maximální chybu, kterou toto zjednodušení nepřekročí. Jako zjednodušovací operátor používá edge collapse.

Zjednodušení funguje dobře u modelů s validní topologií, resp. částí modelu, kde na sebe polygony modelu navazují (viz obrázek 3.1). U jiných modelů pak nemusí být dosaženo

požadované míry zjednodušní, neboť operátor v určitém momentě již nelze aplikovat (obrázek 4.1).



Obrázek 4.1: Model letadla lze nástrojem *osgUtil::Simplifier* z původních 9 180 trojúhelníků zjednodušit maximálně na 5 689

Kapitola 5

Návrh aplikace

Cílem projektu bylo vytvořit prototyp hry s použitím algoritmů pro zjednodušování modelů a implementovat alespoň dva tyto algoritmy. Jednou z možností je použít výše zmíněný simplifikátor implementovaný v OSG. Ten používá operátor edge collapse popsaný v sekci 3.1.1 a funguje relativně dobře pro modely s validní topologií a dobře navazující sítí trojúhelníků. U jiných modelů však může mít jen velmi malý efekt. Proto byl vrámci projektu implementován algoritmus *Vertex clustering* ve dvou různých variantách. Algoritmus lze aplikovat na libovolný model, nezávisle na topologii. Dále v kapitole je popsán návrh prototypu hry a aplikace pro genorování LOD modelů.

5.1 Zjednodušovací algoritmy

5.1.1 Uniform grid vertex clustering

Algoritmus používá lokální operátor *cell clustering* popsaný v sekci 3.1.4. Nejdříve dojde k ohodnocení každého vrcholu v modelu podle důležitosti na základě hodnotící funkce. Následně je přes model rozprostřena pomyslná 3D mřížka a všechny vrcholy vrámci každé buňky jsou sjednoceny do jediného, toho s nejvyšším ohodnocením v buňce. Pokud se alespoň dva vrcholy trojúhelníku nacházejí ve stejné buňce, je trojúhelník vyfiltrován. Rozlišení mřížky ovlivňuje detail výsledného modelu. Čím větší jsou buňky, tím menší je detail výsledného modelu.

Ohodnocení vrcholu

Pro hodnocení vrcholu existují různé způsoby. V tomto případě bylo implementováno hodnocení podle délky nejdelší hrany, která z vrcholu vede. Myšlenka je taková, že čím delší hrana z vrcholu vede, tím větší trojúhelníky se u něj nacházejí a tím je vrchol důležitější. Je možné, že se v buňce najdou dva vrcholy se stejným, nejvyšším ohodnocením a vybrán je pouze jeden z nich.

Výhody a nevýhody

Výhodou algoritmu je, že je relativně rychlý a oproti zjednodušení implementovaném v *osgUtil::Simplifier* nevyžaduje návaznost polygonů v modelu pro zjednodušení.

Nevýhodou algoritmu je, že nelze určit míru zjednodušení modelu jinak, než experimentálně. Dalším nedostatkem je fakt, že stejný model s jiným natočením nebo posunutím

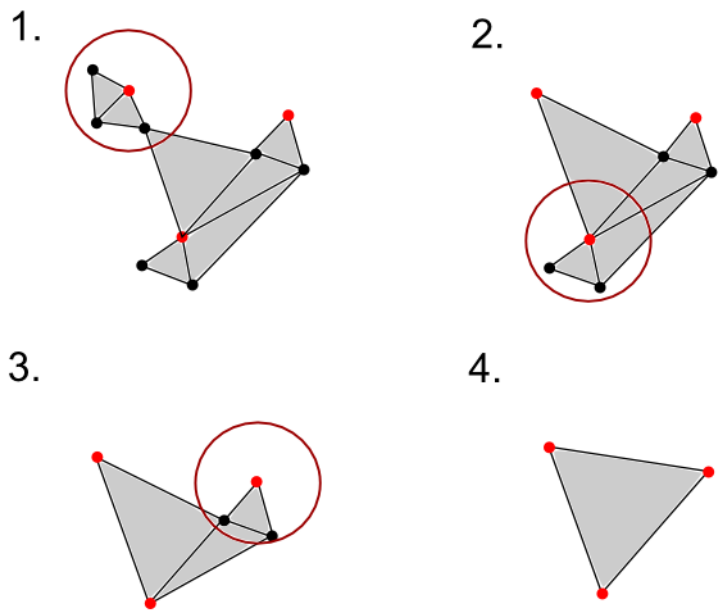
může produkovat odlišné výsledky. Také nelze určit žádnou maximální odchylku od původního modelu a výsledek pak může být méně vizuálně uspokojiný.

5.1.2 Floating-cell clustering

Stejně jako u *uniform grid* dojde nejdříve k ohodnocení všech vrcholů a poté se vrcholy seřadí v seznamu podle důležitosti. Následuje posloupnost:

1. Vyjmi vrchol ze začátku seznamu (nejvyšší ohodnocení)
2. Umísti střed shlukovací buňky na tento vrchol
3. Sjednoť všechny vrcholy v buňce do jediného a odstraň je ze seznamu
4. Pokud není seznam prázdný, vrať se k 1.

Buňka v tomto případě nemá tvar kostky, ale koule. Tento způsob, narozdíl od uniform grid, produkuje konzistení výsledek nezávisle na poloze a natočení modelu.



Obrázek 5.1: Floating cell vertex clustering – příklad

Hodnocení vrcholů

Kromě způsobu popsaného u *uniform grid vertex clustering* byl implementován ještě další. Ten vezme největší úhel mezi přilehlými hranami a vrcholu dá tím větší ohodnocení, čím je daný úhel menší. Vrchol s velkým ohodnocením pak leží v oblasti s velkým zakřivením a je pravděpodobné, že se nachází na siluetě modelu [4].

5.2 Návrh prototypu hry

Hra je koncipována jako jednoduchý letecký simulátor. Cílem bylo, aby mohl hráč ovládat letadlo nad krajinou, střílet a případně ničit jiná letadla nebo pozemní objekty. Letadlo by se mělo chovat reálně, ale nějaká přesnější implementace fyzikálních zákonů nebyla cílem. Důraz byl kladen spíše na snadné ovládání, plynulost pohybu a kamery. Návrh počítal i s nepřátelskými letadly, která by se snažila zamířit a zničit uživatel, to se však nakonec podařilo jen částečně.

Hlavní ovšem bylo, aby modely ve scéně používaly technologii LOD, a aby úrovně detailu byly vytvořeny pomocí některého z výše popsaných algoritmů. Z prvu bylo zamýšleno, že se jednotlivé úrovně detailu vytvoří na začátku po spuštění aplikace. Nebyl to nicméně nejlepší způsob demonstrace zjednodušovacích algoritmů.

Nakonec tedy vrámci projektu vznikla ještě další aplikace *LOD Maker*, která pouze zobrazí daný model, zjednoduší ho podle kritérií zadaných uživatelem a umožní vytvořit *osg::LOD* uzel, který je možné exportovat do souboru. Tento soubor se následně použije ve hře. Pro hru samotnou to znamená, že načítání modelů trvá kratší dobu (již nemusí probíhat zjednodušování). Modely použité ve hře pak ale zabírají více místa na disku.

5.3 LOD Maker

Cílem bylo, aby aplikace zobrazila model a umožnila výše popsanou funkcionalitu. K tomu bylo potřeba přidat nějaké ovládací prvky jako textová pole a tlačítka. V OSG ovládací prvky sice jsou, avšak poměrně omezené, pročež byl k tomuto účelu použit framework Qt [3][2].

Kapitola 6

Implementace

6.1 Algoritmy

Zjednodušovací algoritmy byly implementovány vrámci třídy *mySimplifier*. Ta obsahuje metody *simplify1()* a *simplify2()* pro *uniform grid* a *floating cell vertex clustering*.

Dále je součástí metoda *modify()*, která upraví sady primitiv (*osg::PrimitiveSet*) v případě, že se sestávají z pásů trojúhelníků (*GL_TRIANGLE_STRIP*) na trojúhelníky (*GL_TRIANGLES*). Metoda *modify()* je volána před oběma zjednodušovacími metodami, neboť ty s pásy trojúhelníků zacházet neumějí a pro zjednodušování je třeba pásy rozdělit na jednotlivé trojúhelníky.

Po zjednodušení jsou pásy opět vytvořeny pomocí třídy *osgUtil::TriStripVisitor*. Jejich výhodou je, že zabírají méně místa a i jejich vykreslování je rychlejší. Pás o N trojúhelnících odkazuje pouze na N+2 vrcholů, kdežto samostatné trojúhelníky odkazují na 2N vrcholů.

Třída *mySimplifier* dále obsahuje privátní proměnnou *ratio*, která určuje délku hrany shlukovací buňky v případě *simplify1()*, případně poloměr kulové shlukovací buňky v případě *simplify2()*. Pak také booleovskou proměnnou *gradMode*, kterou se přepíná systém ohodnocení hran u druhého algoritmu. Obě tyto proměnné je možné nastavit parametry v konstruktoru, případně změnit metodami *setRatio()* a *setMode()*.

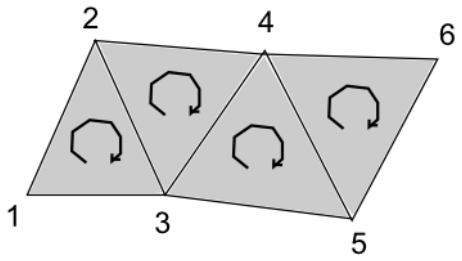
6.1.1 modify()

Metoda prochází primitive sety s pásy trojúhelníků a vytváří podle nich nové primitiv sety se samostatnými trojúhelníky. K tomu stačí jednoduchý průchod vrcholy pásu. Pozor se však musí dát na orientaci výsledných trojúhelníků, neboť v pásu jsou vrcholy trojúhelníku poskládány na přeskáčku po a proti směru hodinových ručiček (obr. 6.1). Aby tedy každý druhý trojúhelník nebyl orientován opačně, je třeba pořadí jejich vrcholů zaměnit.

6.1.2 simplify1() – uniform grid vertex clustering

Metodě je v jediném parametru předán ukazatel na uzel, který obsahuje model pro zjednodušení. Počítá se s tím, že uzlem může být i skupina (*osg::Group*). V tom případě se metoda zavolá rekurzívně pro všechny její potomky. Dále se v podstromu uzlu hledají geody, které obsahují geometrie (*osg::Geometry*). Vytvoří se pole pro vrcholy nového modelu, jejich atributy a pole pro jejich ohodnocení.

V geometrii pak probíhá úprava jednotlivých primitive setů, u nichž se v cyklu projdou všechny trojúhelníky. Nejdříve se zjistí, zda-li se alespoň dva body trojúhelníku nenacházejí



Obrázek 6.1: Orientace trojúhelníků v triangle stripu

ve stejné buňce (clusteru). Pokud ano, trojúhelník je vynechán, neboť by zdegeneroval na úsečku nebo bod.

Následně se u každého vrcholu trojúhelníku zjistí, zda-li už má nový model v daném clusteru nějaký vrchol. Pokud ne, přidá se vrchol trojúhelníku na konec pole pro vrcholy nového modelu. Pokud ano, spočítá se ohodnocení vrcholu a v případě, že bude vyšší než u vrcholu v dané buňce, je vrchol nahrazen i s jeho hodnocením. Následně se trojúhelník přidá na konec primitive setu pro nový model.

Hodnocení se počítá jako nejdelší hrana vrcholu, tedy jako největší vzdálenost k vrcholu některého z přilehlých trojúhelníků.

6.1.3 simplify2() – floating-cell vertex clustering

Podobně jako u předchozí metody se v hierarchii uzlů vyhledají geometrie a jejich sady primitiv. V prvním průchodu trojúhelníky v primitive setu dojde k ohodnocení, které se uloží do paralelního vektoru.

Kromě hodnocení délky nejdelší přilehlé hrany je zde implementováno i druhá možnost hodnocení vrcholů, popsané v sekci 5.1.2. Důvod, proč není implementováno i v *simplify1()* je hlavně ten, že první metoda hodnocení vybírá z možných ohodnocení vrcholu to největší a stejně tak mezi vrcholy dominuje ten s největším ohodnocením. V případě druhé metody se u jednoho vrcholu vybírá ohodnocení nejmenší, přestože mezi jednotlivými vrcholy je cennější ten s vyšším ohodnocením. V *simplify1()* probíhá vrámci snahy o rychlost hodnocení i shlukování vrcholů vrámci jednoho průchodu primitive setem a implementovat tam i druhý typ hodnocení by znamenalo jeden průchod navíc.

Po ohodnocení vrcholů proběhne jejich seřazení podle důležitosti. Následuje posloupnost popsaná v sekci 5.1.2, kdy se vždy vrchol s nejvyšším ohodnocením vyjme a vrcholy v zadané blízkosti se do něj sjednotí, vyjmou se z původního pole a nový vrchol je zařazen na konec pole s novými vrcholy. Zároveň se do pomocného vektoru *unifiction* mapují indexy, který vrchol v poli s novými vrcholy nahradí vrchol z pole s původními vrcholy.

Toho se využije v posledním kroku, při modifikaci primitive setu, který odkazuje na původní vrcholy. U každého trojúhelníku se zjistí, jaké jsou jeho nové vrcholy a pokud trojúhelník nezdegeneruje na úsečku nebo bod, uloží se na konec nového primitive setu.

6.2 Hra

Hra byla implementována v jazyce C++ s použitím OSG verze 3.0.1 v prostředí Microsoft Visual Studio 2010.

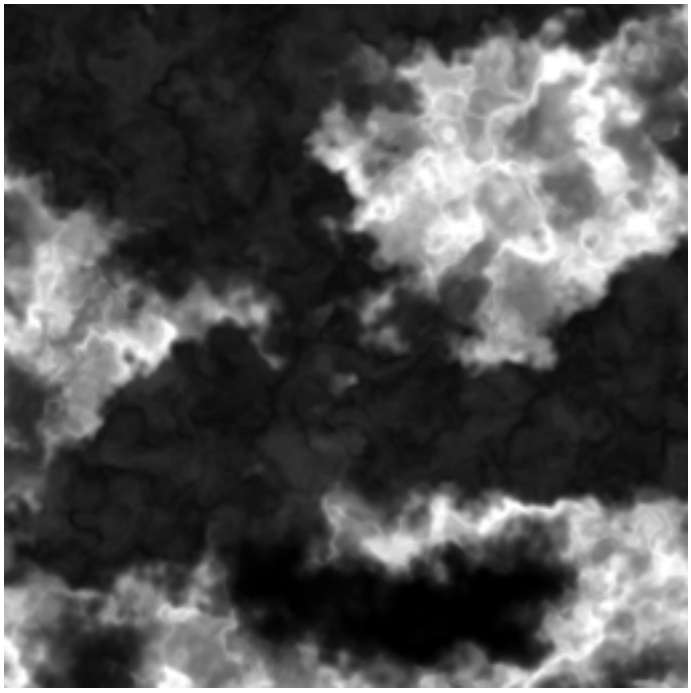
6.2.1 Hlavní třídy

Třídou, která spravuje herní data je *gameClass*. Třída *sceneClass* spravuje veškerá grafická data. Vstup z klávesnice obstarává třída *myKeyboardEventHandler*. Letadla, střely a pozemní objekty jsou instancemi tříd *planeClass*, *landObject* a *shotClass*.

6.2.2 Terén

K načtení terénu slouží funkce *createHeightField()*. Terén se generuje z bitmapového obrázku, který slouží jako výšková mapa. Každý pixel určuje výšku terénu v daném bodě podle své tmavosti. Čím je pixel světlejší, tím určuje vyšší místo.

Pro uložení výškové mapy slouží třída *osg::HeightField*. Kvůli lepší optimalizaci není celý terén načten do jedné geody, ale je rozkouskován. Tím se zaručí, že části terénu mimo záběr kamery nebudou vykreslovány. Zároveň je nastaveno, aby se kousky terénu vykreslovaly jen do určité vzdálenosti.



Obrázek 6.2: Výšková mapa pro generování terénu ve hře

6.2.3 Modely

Původní myšlenkou bylo objekty načíst po spuštění programu a rovnou vygenerovat několik úrovní detailu. Od toho nakonec bylo upuštěno a hra načítá rovnou modely s několika úrovněmi detailu. Kvůli toho vrámci projektu vznikla aplikace *LOD Maker*, popsaná v další kapitole, která vygenerování takového souboru umožňuje. Hra tímto způsobem ušetří čas, který by strávila zjednodušováním modelů, za cenu větší velikosti souborů s modely.

6.2.4 Letadla

Letadla jsou instancemi třídy *planeClass*. Pozice v souřadném systému hry je určena trojrozměrným vektorem třídy *osg::Vec3*, natočení letadla je udává kvaternion třídy *osg::Quat*. Pro změnu natočení je třeba jej vynásobit příslušným kvaternionem.

Letadlo je možné otáčet ve všech jeho osách. Třída obsahuje tři proměnné, které určují o kolik stupňů se na dané ose zrovna letadlo otáčí. V případě, že je příslušné tlačítko ve stisknuté poloze, k hodnotě se v každém snímku přičítá (odečítá) konstanta až do určité krajní meze maximální rotace. Když jsou obě klávesy dané osy uvolňeny, konstanta se přičítá/odečítá, dokud není rotace na ose nulová. Tak je zaručena plynulost pohybu, narozdíl od případu, kdy by stiknutí dané klávesy přímo způsobilo rotaci letala o konstantu.

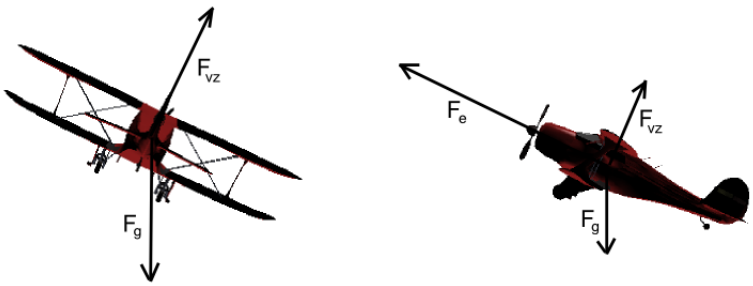
6.2.5 Střely

Model střely je jednoduchý hranol vytvořený třídou *osg::Box*. Letadlo dokáže najednou vystřelit dvě střely (z každého křídla jednu) a do dalšího výstřelu čeká určitou dobu.

6.2.6 Fyzika

Ve hře je také aplikován jednoduchý fyzikální model. Kromě síly, která žene letadlo kupředu na na něj také působí gravitace vždy směrem kolmo dolů v souřadném systému světa. Dále pak vztlaková síla křídel, která působí směrem nahoru v souřadném systému letadla. Pokud letadlo letí vodorovně, působí tyto dvě síly proti sobě a navzájem se vyrovnají.

Pro snažší hratelnost nepůsobí gravitace na střely. Letí tedy přímo rovně a pokud nezasáhnou cíl, po několika sekundách se deaktivují.



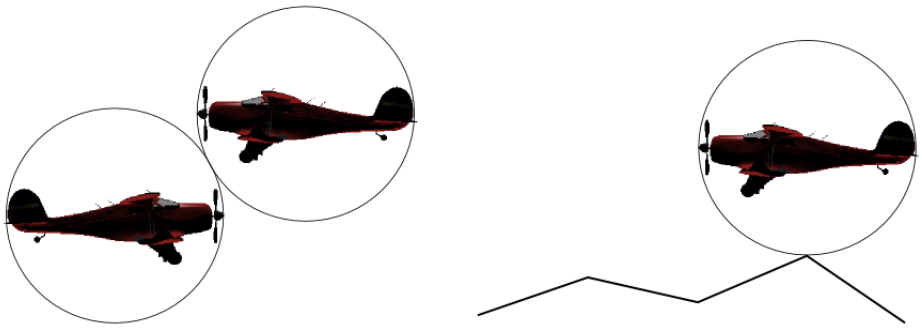
Obrázek 6.3: Fyzika letícího letadla

6.2.7 Detekce kolizí

Kolize se sledují mezi vzdušnými objekty (letadla a střely) navzájem, mezi vzdušnými objekty a pozemními objekty a mezi letadly a terénem. Jako kolize je brán okamžik, kdy se k sobě dva objekty přiblíží na určitou vzdálenost. Dá se tedy říct, že každý objekt je obalen pomyslnou koulí a při dotyku těchto koulí nastává kolize. Koule je určena polohou středu objektu a poloměrem.

Mezi každými dvěma objekty se tedy vypočítá vzájemná vzdálenost jejich středů v prostoru a zkontroluje se, zda-li není menší než součet poloměrů jejich "koulí".

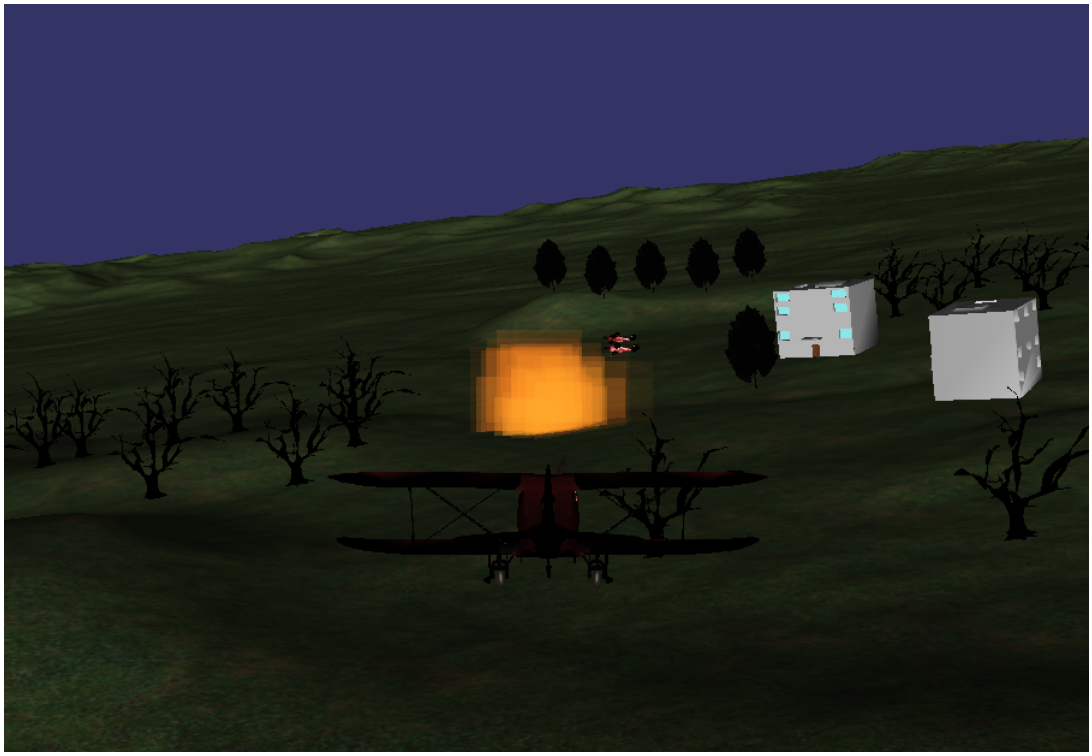
To neplatí pro terén. Z výškové mapy se jednoduše zjistí, jakou má na souřadnicích X a Y letadla terén výšku. Ta se porovná s výškou, ve které se nachází letadlo a jednoduše se tak spočítá případná kolize.



Obrázek 6.4: Detekce kolizí mezi letadly navzájem a mezi letadlem a terénem

6.2.8 Částicové efekty

Pro implementaci částicových efektů slouží knihovna *osgParticle*. Ve hře pak byl implementován efekt výbuchu pomocí třídy *osgParticle::ExplosionEffect*.



Obrázek 6.5: Částicový efekt výbuchu

6.2.9 Plynulost hry

Všechny herní prvky sdružuje instance třídy *gameClass*. Důležité je, aby se plynulost hry neodvíjela od aktuálního počtu snímků za sekundu. K aktualizaci herních dat je použit časovač třídy *osg::Timer*.

6.2.10 Kamera

Pro manipulaci s kamerou byla použita třída *osgGA::NodeTrackerManipulator*, která sleduje uzel, jenž je umístěn kousek nad letadlem v souřadném systému letadla. Tak je vidět uživatelovo letadlo, které zároveň nebrání ve výhledu.

Obrázek 6.6: Kamera

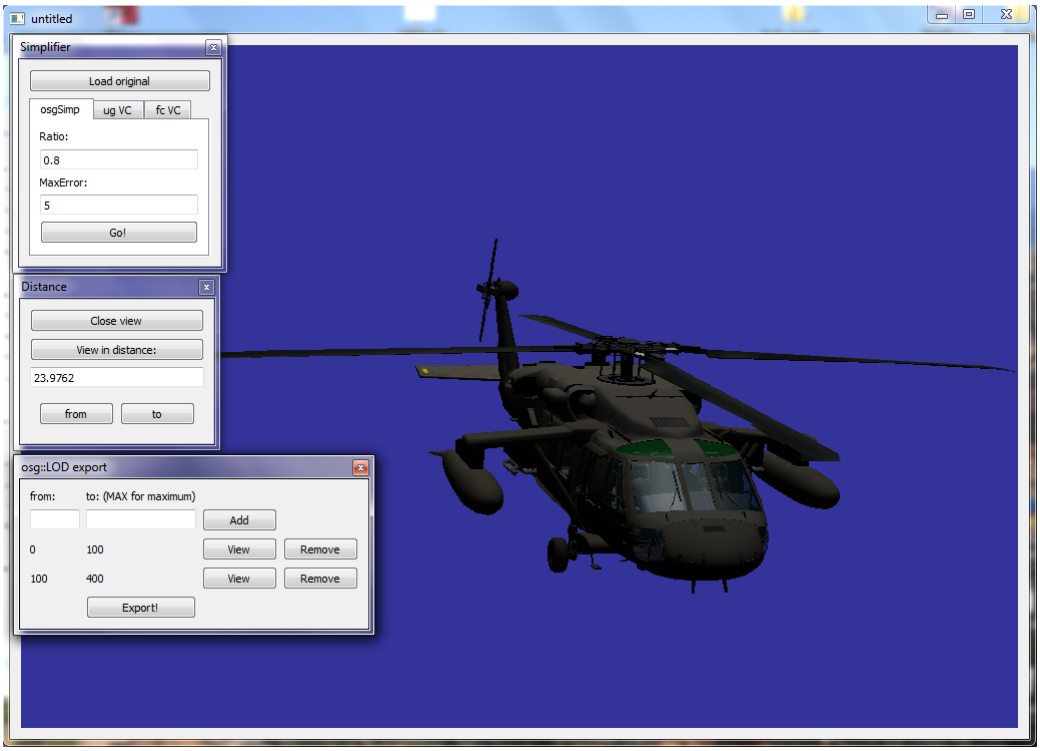
Obrázek 6.7: Výsledná hra

6.3 LOD Maker

Vrámci projektu vznikla ještě další aplikace s názvem LOD Maker. Slouží zejména k prohlížení výsledků zjednodušovacích algoritmů, jelikož hra k tomuto účelu není příliš vhodná.

Zároveň aplikace umožňuje vytvořit osg::LOD uzel s několika úrovněmi detailu a exportovat jej do souboru. Toho je pak využito ve hře, kde místo zjednodušování modelu se načte rovnou model i s jeho zjednodušenými variantami.

Kvůli ovládacím prvků (widgetům) byla v projektu použita knihovna Qt verze 4.8.4. I OSG obsahuje třídu pro widgety, nejsou však příliš propracované. Přestože je možné Qt používat i v MS Visual Studiu, byla nakonec aplikace napsána jako Qt projekt v Qt Creatoru. Výhodou je snažší použití Qt a také relativně snadný port projektu na jiné platformy.



Obrázek 6.8: Aplikace LOD Maker

6.3.1 Ovládání

Aplikace se sestává z hlavního okna s modelem a třech oken s ovládacími prvky, tzv. *dock widgets*. Vstupní model se zadává jako první parametr programu, název výstupního jako druhý parametr. Výstupní soubor by měl mít koncovku '.osg'. Pokud však chybí, přidá se automaticky. Pokud druhý parametr chybí, výstupní soubor bude pojmenován 'exportLOD.osg'. Pokud chybí i první parametr, pokusí se program načíst model 'cow.osg'.

Zjednodušení

První *dock widget* je věnován zjednodušovacím algoritmům, kde je možno originální model zjednodušit jedním ze tří způsobů (*osg::Simplifier*, *uniform grid vertex clustering* a *floating-cell vertex clustering*).

U prvního způsobu zadá uživatel míru zjednodušení a maximální odchylku, kterou zjednodušení na modelu způsobí (max error). Míra zjednodušení by měla být mezi 0.0 a 1.0, kdy větší číslo znamená více detailu. Například míra 0.5 by měla odpovídat polovičnímu detailu.

U druhého způsobu zadá uživatel velikost strany kostky (buňky), ve které se budou shlukovat vrcholy modelu. Míra zjednodušení se nedá dopředu určit příliš přesně. Zůstáva na uživateli, aby to experimentálně vyzkoušel.

Třetí způsob je podobný jako předchozí, uživatel zde však určí poloměr shlukovací buňky tvaru koule.

Třída *osgViewer::StatsHandler* spravuje statistiky právě zobrazovaného modelu včetně počtu polygonů a jejich zobrazení je možné přepínat klávesou 'S'.

Vzdálenost modelu

Dalším *dock widgetem* se určuje vzdálenost, v jaké je model v hlavním okně zobrazen. Je možné zobrazit model zblízka, zobrazit model v zadané vzdálenosti, případně posouvat vzdálenost kolečkem myši. Tak může uživatel získat přehled o tom, jak model s daným zjednodušením vypadá z dálky. Myší je rovněž možné v hlavním okně model otáčet nebo posouvat stisknutím kolečka myši.

Součástí docku jsou také tlačítka *from* a *to*, která danou vzdálenost zkopírují do políček from nebo to ve třetím *dock widgetu*.

LOD uzel

Třetí *dock widget* slouží pro vytvoření výsledného LOD uzlu. Model, který je zrovna v hlavním okně je možné do LODu přidat vyplněním políček *from* a *to*, tedy vzdálenosti od/do, kdy bude dotyčný model zobrazen, a stisknutím tlačítka *add*. Následně je model přidán do seznamu níže, kde je možné jej zobrazit nebo smazat tlačítky *view* nebo *remove*.

Posledním důležitým tlačítkem je *export*, které model v úrovních detailu exportuje do souboru ve formátu OSG.

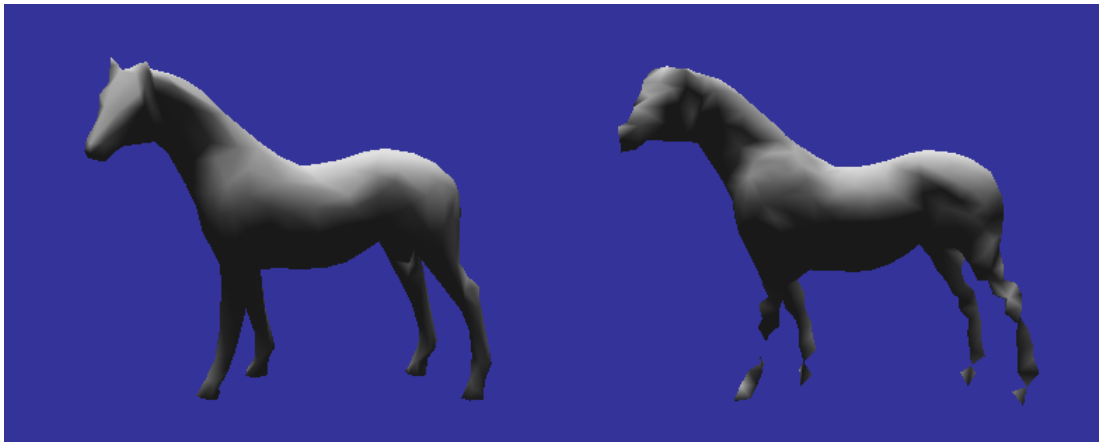
Kapitola 7

Zhodnocení algoritmů

Jedním z výsledků je, že se povedlo implementovat zjednodušení modelů, nezávisle na tom, zda-li je složen ze sítě navazujících trojúhelníků. Nativní simplifikátor v OSG má s takovými modely problém.

Na druhou stranu *osg::Simplifier* dává možnost kontroly nad maximální odchylkou výsledného modelu, a ten je pak zpravidla vizuálně uspokojivější.

Jelikož simplifikátory založené na *cell collapse* tuto možnost nemají, je poměrně obtížné jejich výsledky exaktně porovnat s OSG.

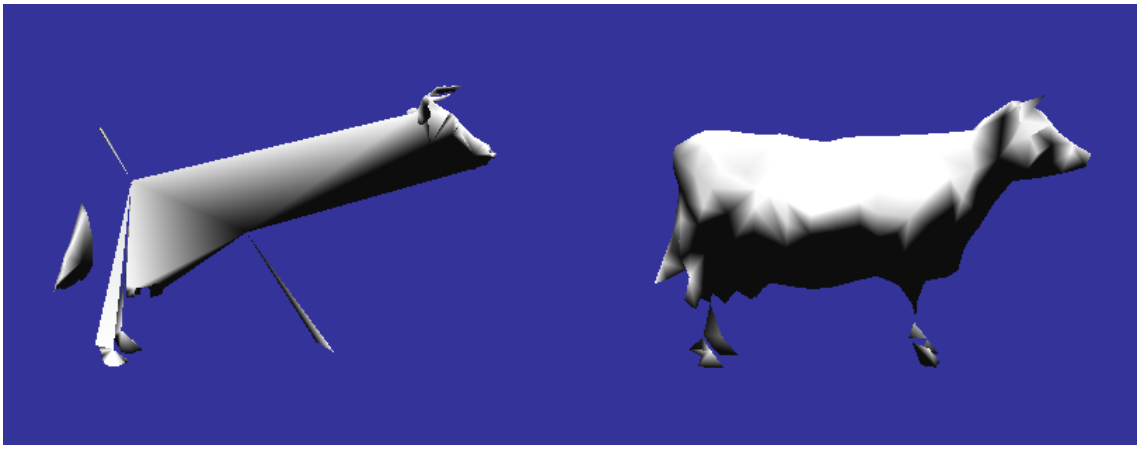


Obrázek 7.1: Model koně – vlevo byl použit *osgUtil::Simplifier* a vpravo *mySimplifier::simplify2*. Oba modely jsou zjednodušeny přibližně na 1000 trojúhelníků.

7.1 Rychlost zjednodušení

Co se však do jisté míry dá porovnat, je rychlost provedení zjednodušení. Problém je pouze v tom, že simplifikátory založené na *cell collapse* nedokáží předem určit míru zjednodušení, a tudíž je prakticky nemožné, aby produkovaly výsledky s nějakým konkrétním počtem trojúhelníků. I tak si ale lze udělat představu, jak jsou na tom jednotlivé algoritmy s rychlostí (viz tabulka 7.1).

Zjednodušovaným modelem byl kůň (obrázek 7.1), který má v plném detailu 96 966 trojúhelníků. Testování bylo provedeno na počítači s procesorem Intel Core 2 Duo CPU P8700 @ 2.53GHz × 2, 2 GB RAM. Výsledné hodnoty jsou aritmetickým průměrem časů



Obrázek 7.2: Model krávy – vlevo byl použit *osgUtil::Simplifier* a vpravo *mySimplifier::simplify2*. Oba modely jsou zjednodušeny přibližně na 730 trojúhelníků

tří zjednodušení modelu na danou úroveň. Algoritmus *floating-cell vertex clustering* byl testován s oběma možnostostmi hodnocení vrcholů – podle délky nejdelší přidružené hrany (hodnocení 1) a podle velikosti největšího úhlu mezi přidruženými hranami (hodnocení 2).

osgUtil::Simplifier					
výsledný počet trojúhelníků	87 268	67 873	48 476	29 083	9 683
doba zjednodušení (s)	7,967	14,081	19,523	25,476	30,482

Floating-cell vertex clustering (hodnocení 1)					
výsledný počet trojúhelníků	89 424	78 654	49 390	17 744	11 606
doba zjednodušení (s)	37,720	33,477	22,425	9,866	7,703

Floating-cell vertex clustering (hodnocení 2)					
výsledný počet trojúhelníků	86 632	53 168	42 106	20 546	13 326
doba zjednodušení (s)	25,520	16,715	13,576	8,142	6,321

Uniform grid vertex clustering					
výsledný počet trojúhelníků	67 728	35 956	21 676	10 382	4 010
doba zjednodušení (s)	162,131	48,340	17,134	4,012	0,751

Tabulka 7.1: Průměrná doba aplikace zjednodušovacích algoritmů na model koně (96 966 trojúhelníků v plném detailu)

Zjistíme, že simplifikátory založené na *cell collapse* mají poměrně dobré výsledky při velké míře zjednodušení, která OSG trvá relativně dlouho. Při menším zjednodušení je tomu přesně naopak.

Výsledek není překvapivý. *osg::Simplifier* založený na *edge collapse* model zjednodušuje postupným rušením hran v modelu. Čím větší míra zjednodušení, tím více hran musí být odebráno a proces tím pádem trvá logicky déle.

Algoritmy *vertex clustering* shlukují vrcholy modelu vrámci buňek. Větší míra zjednodušení znamená méně buněk o větším objemu, a tedy poměrně rychlé zjednodušení. Malé zjednodušení naopak potřebuje velké množství buněk o malém objemu a proces tak trvá

déle.

Z nově implementovaných algoritmů má nejlepší výsledky *floating-cell vertex clustering*. O něco lépe je na tom varianta s hodnocením vrcholů podle přidružených úhlů. *Uniform grid vertex clustering* má relativně dobré výsledky při velmi malém výsledném detailu, ale se zvyšujícím se rozlišením doba zjednodušení výrazně roste, což je důvod, proč nebyl algoritmus testován na více než 67 728 výsledných trojúhelníků. Agoritmus byl implementován naivním způsobem a pro lepší výsledky by bylo třeba jej optimalizovat.

Obrázek 7.3: Model z obrázku 4.1 zjednodušený na 1033 trojúhelníků metodou floating-cell

Kapitola 8

Závěr

Vrámci projektu vznikl prototyp hry s použitím rozhraní OpenSceneGraph. Pro plynulejší běh hry je demonstrována technika *level of detail*, konkrétně její diskrétní verze, která k vytvoření modelů používá zjednodušovací algoritmy.

Kromě nativního simplifikátoru OSG, který se pro některé modely nehodí byly implementovány další dva algoritmy založené na operátoru *cell collapse*, které eliminují nedostatky zjednodušovací metody OSG, avšak za cenu jiných nedostatků. Je na programátorovi, aby zvážil výhody a nevýhody použití jednotlivých algoritmů.

Implementované simplifikátory jsou použitelné především na menších modelech (do 100 000 trojúhelníků). Pro zjednodušování větších modelů by bylo potřeba algoritmy dále optimalizovat.

Aplikace LOD Maker napsaná s použitím frameworku Qt slouží pro demonstraci implementovaných algoritmů a nabízí export modelu s různými úrovněmi detailu do souboru pro další použití rozhraním OSG.

Literatura

[1] openscenegraph.org. online.
URL http://www.openscenegraph.org

[2] qt-project.org. online.
URL http://www.qt-project.org

[3] Blanchette, J.; Summerfield, M.: *C++ GUI Programming with Qt4*. Pearson Education, 2008, ISBN 9780132703000.

[4] Low, K.-L.; Tan, T.-S.: Model simplification using vertex-clustering. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, ACM, 1997, s. 75–ff.

[5] Luebke, D.: *Level of Detail for 3d Graphpics*. The Morgan Kaufmann Series in Computer Graphics Series, Morgan Kaufmann Publishers, 2003, ISBN 9781558608382.

[6] Luebke, D. P.: A developer's survey of polygonal simplification algorithms. *Computer Graphics and Applications, IEEE*, ročník 21, č. 3, 2001: s. 24–35.

[7] Wang, R.; Qian, X.: *Openscenegraph 3.0: Beginner's Guide*. Packt Publishing, Limited, 2010, ISBN 9781849512831.

Příloha A

Obsah CD

- zdrojové kódy hry
- spustitelná hra
- zdrojové kódy LOD Maker
- spustitelný LOD Maker se sbírkou modelů
- návod k použití
- zdrojový kód technické zprávy
- technická zpráva
- plakát