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

TECHNIKY „LEVEL OF DETAIL“ V KNIŽNICI OPENSCENEGRAPH

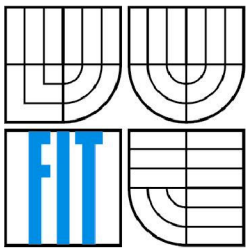
DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

AUTOR PRÁCE
AUTHOR

BC. DUŠAN HUPKA

BRNO 2014

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

TECHNIKY „LEVEL OF DETAIL“ V KNIHOVNE OPENSCENEGRAPH

ALGORITHMS OF LEVEL OF DETAIL IN OPENSCENEGRAPH

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. DUŠAN HUPKA

VEDOUCÍ PRÁCE
SUPERVISOR

ING. JAN PEČIVA, PHD.

BRNO 2014

Abstrakt

Dnešní grafika se neobejde bez neustálé optimalizace technik a výpočtů. Je to způsobeno tím, že nároky na zobrazování scény jsou pořád vyšší. Jednou z technik, která napomáhá optimalizovat scénu jako takovou, je Level of detail. Tahle práce je zaměřena na jednotlivé metody, často používané v LOD a v knihovně OpenSceneGraph a OpenGL. Podrobně popíše, podle čeho se ve scéně určí, jaká úroveň detailu se má vybrat, a jak se 3D modely zjednodušují. Představené techniky budou následně implementovány do konverzní utility a do demonstrační aplikace. Metody pro zjednodušení modelu budou rychlostně i kvalitativně měřeny a vyhodnoceny.

Abstract

Present graphic requires a lot of optimizations of rendering techniques and mathematical calculations. It is caused by increased requirements of scene's visualization. One of scene's optimizing techniques is the Level of detail. This thesis is focused on methods used by LOD in OpenSceneGraph and OpenGL library. Next it will be described how to choose the right level of detail in a scene. Later it will be explained how to simplify 3D models. These techniques will be implemented in converting tool and demonstrating application. Methods for simplify 3D models will be tested for their speed and quality.

Klíčová slova

Zjednodušování 3D modelů, úroveň detailu, metriky, redukce trojuhelníků, zjednocení hrany, zjednocení trojuhelníku, zmazání vertexu, zjednocení buňky, OpenSceneGraph, LOD, PagedLOD, Impostor

Keywords

3D model simplify, level of detail, metrics, triangles reduction, edge collapse, triangle collapse, remove vertex, cell collapse, OpenSceneGraph, LOD, PagedLOD, Impostor

Citace

Hupka Dušan: Techniky „level of detail“ v knižnici OpenSceneGraph, diplomová práce, Brno, FIT VUT v Brně, 2014

Techniky „level of detail“ v knižnici OpenSceneGraph

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jana Pečivi Phd.
Další informace mi poskytol Ing. Tomáš Starka.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Dušan Hupka
28.5.2014

Poděkování

Chcel by som poďakovať svojmu vedúcemu práce Janovi Pečivovi za odborné rady a pomoc pri písaní tejto práce. Tak isto by som rád poďakoval za časté konzultácie Tomášovi Starkovi.

© Dušan Hupka, 2014
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

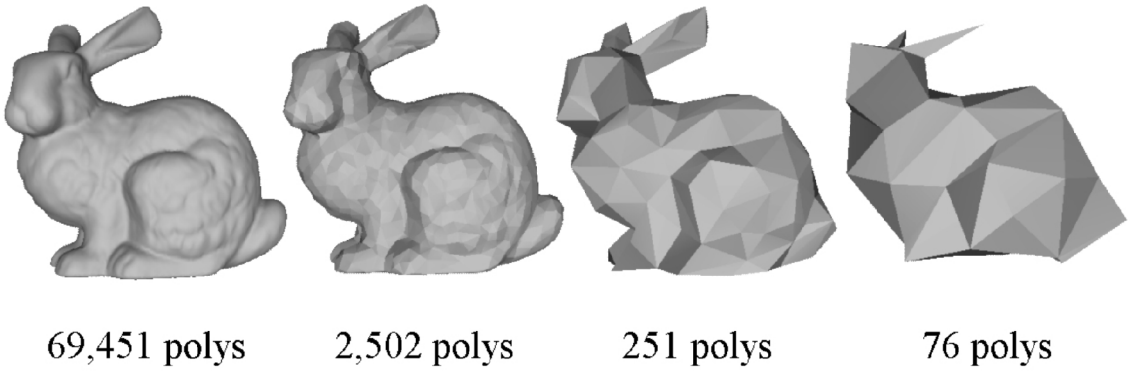
1 Úvod

V dnešnom svete grafického a herného priemyslu sa kladie veľký dôraz na rýchlosť a optimalizovanosť výslednej implementácie. Výskumníci z celého sveta sa nezaoberajú len novými objavmi, ale aj zrýchlením a vylepšením už vymyslených produktov. Niekedy sa podarí článkom začať úplne nový smer v danom odvetví. Takýto článok sa podarilo napísať pánovi Jamesovi Clarkovi [1]. Jeho publikácia narážala na neoptimálnosť pri vykresľovaní modelov s veľkým počtom polygónov, ktoré vo výsledku zaberajú na obrazovke len pár pixelov. Naznačil, že takýto prístup nie je najlepší a ďalej načrtol inovatívnu hierarchiu scény a kompletne popísal základnú myšlienku takzvaného statického Level of detail, ktorý je dokonca jedným z najpoužívanejších aj v dnešnej dobe. Okrem základov LOD, pán Clark ako prvý vo svojom článku popísal aj dodnes používané view-frustum culling. Myšlienky LOD priniesli pre grafiku veľký pokrok, pretože je možné s rovnakým hardvérom vykresliť scénu omnoho kvalitnejšie za použití rovnakého množstva polygónov. Práca si kladie za cieľ informovať čitateľa o problematike LOD a algoritmoch, ktoré sa v tejto tematike považujú za kľúčové.

V tejto práci budú ako prvé podrobne popísané princípy Level of Detail. Čitateľ sa dozvie podrobnosti o statickom LOD a základy dynamického LOD. Práca je zameraná na statický LOD, ku ktorému patria algoritmy zjednodušovania modelov a chybové metriky pre tieto algoritmy. Tie budú predstavené v ďalšej podkapitole. Následne sa práca venuje aj technikám Level of Detail v bežne používaných knižniciach ako je OpenGL alebo OpenSceneGraph. V tejto kapitole budú predstavené základné prostriedky a možnosti týchto knižníc pre obsluhu LOD technológií. V rámci práce prebehla implementácia dvoch aplikácií, ktorým je venovaná štvrtá kapitola. V priebehu implementácie boli naprogramované aj vybrané metriky. Meraniam a hodnoteniam týchto metrík a použitých technológií LOD patrí predposledná kapitola.

2 Princípy Level of detail

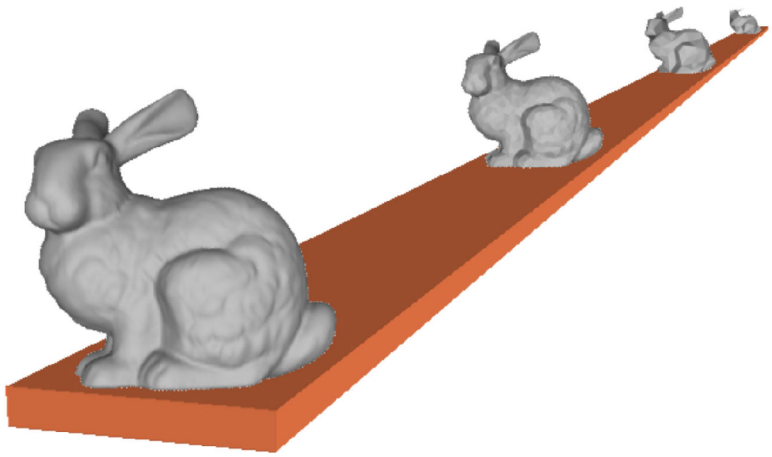
Hlavný princíp Level of detail je vedieť vyprodukovať viac detailných úrovní jedného modelu, pri zachovaní čo najpodobnejších proporcií a vizuálnej kvality s originálnym modelom. Na obrázku 2.1 je možné vidieť rôzne úrovne detailu typického modelu pre demonštráciu LOD a pod každou demonštrovanou ukážkou je uvedený počet polygónov pre danú úroveň modelu.



Obrázok 2.1: Ukážka úrovní modelu od najzložitejšieho po najjednoduchší [12]

Podľa obrázku je možné pozorovať postupné klesanie kvality modelu. Tento postup takzvaného postupného zjednodušovania modelu je základným princípom LOD.

Ako pán Clark [1] uviedol, je zbytočné vykresľovať veľké množstvo polygónov pri objektoch, ktoré sú vo výslednom obrázku malé. Preto vymyslel ideu prepínania úrovní detailu modelu na základe vzdialenosti od pozorovateľa. Čím je model ďalej od zobrazovacej roviny, tým sa vykreslí model s menším počtom polygónov. Na obrázku 2.2 je demonštrácia LOD zobrazenia príkladného modelu z obrázka 2.1.



Obrázok 2.2: Ukážka LOD zobrazenia. Najjednoduchší model je najvzdialenejší. [12]

Historicky prvé implementácie level of detail boli použité v leteckých simulátoroch v 80. rokoch minulého storočia [2]. V týchto počiatkoch sa rôzne detaily modelov robili ručne a museli sa investovať veľké peniaze do dizajnérov a modelárov. Preto následne v 90. rokoch sa začali výskumníci zaoberať automatizáciou týchto procesov. Vzniklo mnoho štúdií a článkov o algoritmoch pre automatické zjednodušovanie modelov s veľkým množstvom polygónov - pre príklad [3] [4].

2.1 Statický Level of detail

Hlavná myšlienka statického alebo takzvaného diskrétneho LODu je založená na Clarkovej práci. Idea je v tom, že ešte pri vývoji alebo pred spustením grafickej aplikácie využívajúcej LOD, sa predvypočítajú viaceré úrovne detailu modelu a uložia sa do pamäte. Počas behu aplikácie sa už len na základe kritérií rozhodne, ktorá úroveň detailu sa z pamäti vybere a následne zobrazí. Ideálny výsledný efekt je, že užívateľ si nevšimne zhoršenej kvality modelov pri menších alebo vzdialenejších objektoch a zároveň scéna obsahuje radikálne nižší počet polygónov k vykresľovaniu.

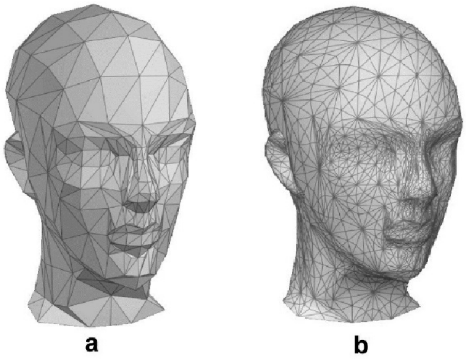
Takýto prístup ma viacero výhod ale i nevýhod. Medzi tie najpodstatnejšie výhody patrí, že úrovne detailu sa vypočítavajú pred spustením aplikácie, a teda prakticky nezáleží na dĺžke trvania výpočtov. Zároveň sa môže pri výpočte úrovní detailu model zoptimalizovať alebo inak upraviť aby viac vyhovoval požiadavkám na LOD. Vo väčšine prípadov každá takto predvypočítaná úroveň detailu je skontrolovaná ľudským faktorom a teda prípadné vizuálne nezrovnalosti sa môžu bez problémov opraviť. Ďalšia veľká výhoda je, že v spustenej aplikácii je už samotná správa prepínania úrovní detailov výpočetne pomerne jednoduchá a rýchla.

Avšak statický level of detail prináša aj problémy a to hlavne zvýšenú pamäťovú náročnosť. Ideálne by bolo mať čo najviac úrovní detailu pre každý model aby sa zobrazoval v scéne čo najrealistickejšie a zároveň neobsahoval mnoho polygónov. Takéto riešenie je však nemožné, vzhľadom na obmedzenú kapacitu pamäte a teda model máva predvypočítaných len zopár úrovní detailu. Tým sa dostávame k ďalšiemu problému a to takzvané „preblikávanie“ modelu. Keď sa za behu programu splnia kritéria pre zmenu úrovne detailu, tak sa z pamäti načíta nový model s inou úrovňou detailu a vtedy sa pôvodný model nahradí novým modelom. Tento okamih nahradenia jedného modelu za druhý môže byť viditeľný pre užívateľa ako „prebliknutie“, čo je nežiadúci vedľajší efekt. Ako ďalšiu nevýhodu je potrebné uviesť, že statický prístup je neadaptívny, a teda sa nemôžu prispôsobovať predvypočítané úrovne detailu vonkajším vplyvom. Pre príklad si uvedieme adaptívny LOD, ktorý by menil len istú časť modelu na detailnejšiu úroveň podľa toho, či je daná časť modelu viditeľná užívateľovi alebo nie. Neadaptívny LOD dokáže zmeniť úroveň detailu modelu len ako celku.

2.1.1 Tvorba predpočítaných modelov

Samotná tvorba modelov sa dá uskutočniť dvoma spôsobmi. Buď zo zložitejšieho modelu vyprodukovať jednoduchší, alebo z jednoduchšieho naopak zložitý. Druhý spôsob je uskutočňovaný technikou subdivision, ktorá na hrany povrchu modelu pridáva ďalšie vertexy tak, aby ostré hrany boli jemnejšie. Táto technika má však jedno základné obmedzenie a to také, že použitím subdivision sa vytvára model obsahujúci viac informácií z modelu, ktorý ich obsahuje menej. To znamená, že detailné informácie o povrchu sa dajú dopočítať len z už vytvorených informácií. Napríklad z kompletne rovnej steny nie je možné spraviť detailnú kamenistú stenu (bez ďalšieho použitia inej techniky, algoritmu alebo nejakého iného zdroja informácií). Preto na tvorbu LOD modelov sa v praxi najčastejšie používa produkcia zjednodušených modelov z ich zložitejších predlôh.

Procesu zjednodušovania je v tejto práci venovaná samostatná kapitola 3.2. Subdivision však má veľmi dôležité miesto v DCC[1] programoch pri tvorbe najzložitejšieho modelu (z ktorého sa neskôr robia jednoduchšie LOD modely).



Obrázok 2.3: Demonštrácia pridania polygónov do modelu *a* za pomoci funkcie Subdivision [webreference.com]

2.1.2 Kritériá na výber úrovne modelu

Vo vyššie uvedenom texte boli spomenuté takzvané kritériá pre zmenu úrovne detailu. Tieto kritériá slúžia výslednému programu k určeniu, či má dôjsť k zmene modelu na inú úroveň detailu. Najčastejšie sa používajú kritériá určené z týchto princípov:

- Vzdialenosť
- Veľkosť
- Priorita
- Hysterézia a okolné podmienky
- Vnemové faktory

Vzdialenosť

Asi najčastejšie používané kritérium pre zmenu úrovne detailu. Idea spočíva vo vzdialenosti od pozorovateľa. Čím je táto vzdialenosť väčšia, tým stačí menšia úroveň detailu modelu. Kvôli pamäťovej náročnosti je nutné určiť adekvátny počet úrovní a podľa vizuálnej stránky nastaviť vzdialenostné hranice, kedy sa ktorá úroveň použije. Pre príklad najlepšia úroveň bude použitá do 5 metrov od pozorovateľa, stredná úroveň od 5 do 20 metrov a najhoršia úroveň sa aplikuje pri vzdialenosti nad 20 metrov. Podľa tohto kritéria si každý snímok testuje vzdialenosť od pozorovateľa a keď sa prekročí nejaká hranica, tak sa model prepne do patričnej úrovne. Základný výpočet vzdialenosti od pozorovateľa berie v úvahu práve jeden bod v modelu, najčastejšie v strede modelu, a od neho sa počíta vzdialenosť k pozorovateľovi. Výhodou takto zvoleného bodu je veľká rýchlosť určenia správnej úrovne modelu. Na druhú stranu tento prístup má ale aj radu nevýhod. Pri veľmi dlhých modeloch sa môže vyskytnúť situácia, keď stred modelu je ďaleko od pozorovateľa, ale zároveň nejaká časť modelu je blízko a teda pozorovateľ uvidí detailne nízku úroveň modelu. Preto je vhodné zvoliť bod pre výpočet úrovne taký, aby bol najbližšie k pozorovateľovi. Určenie takéhoto bodu však žiada ďalšiu réžiu vo výpočte. Taktiež je potrebné počítať s tým, že ak model zmení svoju

[1] DCC - Digital Content Creator

veľkosť, tak hranice s najväčšou pravdepodobnosťou nebudú korektné. Napríklad výšková budova by mala mať najväčšiu úroveň detailu aj nad 20 metrov od pozorovateľa, pričom ak by sa jednalo o malý model budovy, tak hore uvedené hranice sú vskutku reálne. Ďalej pri hraniciach môže prísť ku častému vyššie spomínanému preblikávaniu. Ak sa model pohybuje, môže pri hranici nastať situácia, kedy sa model behom krátkych časových okamžikov vyskytuje v jednom alebo druhom pásme a teda dochádza k veľmi častému prepínaniu úrovne detailu.

Veľkosť

Toto kritérium je myslené ako veľkosť výsledného zobrazenia modelu po rasterizácií. Čím je objekt zobrazený na menšej ploche, tým je aj požadovaná úroveň menšia. Zmena úrovne nastane opäť ako veľkosť modelu prekročí istú hranicu. Tento prístup má ako svoje výhody tak i nevýhody. Medzi hlavnú nevýhodu patrí väčšia výpočetná náročnosť oproti vzdialenosti. Pre výpočet veľkosti je nutné premietnuť vrcholy modelu do 2D plochy a zrátať veľkosť. Na druhú stranu sa tu nevyskytuje problém pri zmene veľkosti modelu ako tomu je pri kritériu vzdialenosti. Bohužiaľ artefakt preblikávania ostáva aj v tejto metóde nevyriešený.

Priorita

Pri väčšine programov a ich jednotlivých scén je nutné riešiť aj prioritu objektov v scéne. Podľa tejto priority sa následne môžu určiť zjednodušené alebo naopak zložitejšie modely pre objekty v scéne. Napríklad pri simulácii jazdy v automobile sú dôležitejšie detaily na volante a na palubovej doske ako detaily na autorádiu.

Hysterézia

Hysterézia rieši problém, ktorý bol vyššie spomenutý ako neustále preblikávanie. Pridaním istej tolerancie ku hraniciam sa docieli eliminovanie efektu neustáleho prepínania LOD pri tejto hranici. Pre vyššie uvedený príklad by sa stredná úroveň detailu zapla až po prekročení 5,5 metra od pozorovateľa ale spätný prechod do najväčšej úrovne by nastal pri 4,5 metroch.

Ľudský vnem

Rôzna úroveň detailu modelov v scéne môže závisieť aj od ľudských vnemov. Myšlienka je taká, že človek vníma v daný okamih niektoré časti scény menej a iné naopak viac. Typické pre zhoršené vnímanie sú objekty v periférii alebo objekty mimo hlavné dianie. Napríklad, ak sa v scéne kamera pozerá von oknom, kde na ulici sa dejú podstatné činnosti pre užívateľa, je zbytočné aby rám okna v daný okamih bol v najlepšej možnej úrovni kvality. Podobne je nepotrebné pre užívateľa vidieť v najvyššej kvalite modely, ktoré sa pohybujú nezanedbateľnou rýchlosťou. Kvalitu takýchto objektov jeho oko a mozog aj tak nedokážu správne zachytiť a spracovať.

2.2 Dynamický Level of detail

Dynamický, alebo takzvaný spojitý LOD, má opačnú ideu ako statický LOD. V statickom LOD je kľúčová myšlienka predvypočítať si rôzne úrovne modelu dlho pred tým než sú potreba. Naopak v Dynamickom LOD sa model neustále mení podľa kritérií (napríklad vzdialenosť) a nejaké hranice sa nepoužívajú. Typickým reprezentantom sú progressive meshes [5].

Po štarte aplikácie sa načíta model do špeciálnej štruktúry, v ktorej je pre model určené ako ho postupne po malých krokoch zjednodušovať až do naprosto minimálnej úrovne. Za jeden krok zjednodušenia sa považuje aplikovanie takzvaného operátora. Tie budú podrobnejšie popísané

v ďalšej kapitole, avšak je nutné zdôrazniť, že operátory sa používajú aj pri tvorbe modelov pre statický LOD.

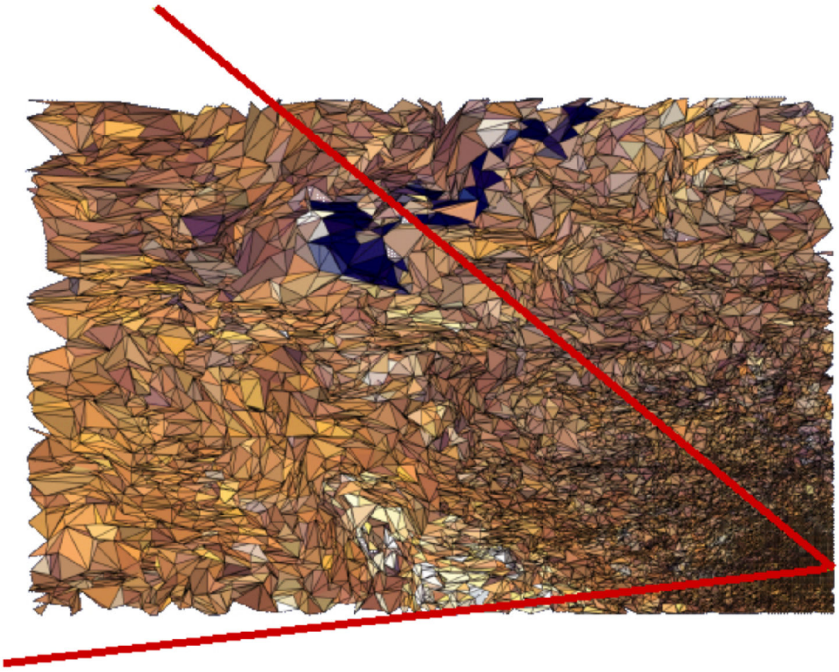
Následne po štartu scény sa v každom snímku pri behu aplikácie na základe kritéria určí ako veľmi sa má pôvodný model zjednodušiť. Po tomto určení prebehne samotné zjednodušovanie modelu postupnou aplikáciou operátorov, kým sa kvalita modelu nedostane na požadovanú úroveň. Tento proces sa dá zrýchliť tým, že nezjednodušujeme po každé pôvodný model, ale pracujeme s modelom, ktorý bol v minulom snímku. Pri prechode na zložitejšiu úroveň detailu je pri veľa operátoroch možné ho aplikovať obrátene a teda zozložitiť model na požadovanú kvalitu.

Dynamický LOD má mnoho výhod i nevýhod. Podstatná nevýhoda je časová a pamäťová náročnosť. Štruktúra s modelom a s informáciami o postupnom zjednodušovaní zaberá väčšiu pamäť, ako by tomu bolo pri statickom LOD. Ďalej samotné vykonanie operátorov berie značnú časť výpočetného času pre jeden snímok.

Medzi hlavnú výhodu patrí úspora výsledných polygónov. Pre model je vypočítaná presná hodnota výslednej kvality podľa kritéria a nie približná ako je tomu pri statickom LOD. Ďalej je veľmi priaznivé, že spojitý LOD odstraňuje preblikávanie. Zmeny na modelu medzi jednotlivými snímkami sú zväčša tak malé, že užívateľ si zmenu kvality nepovšimne prebliknutím.

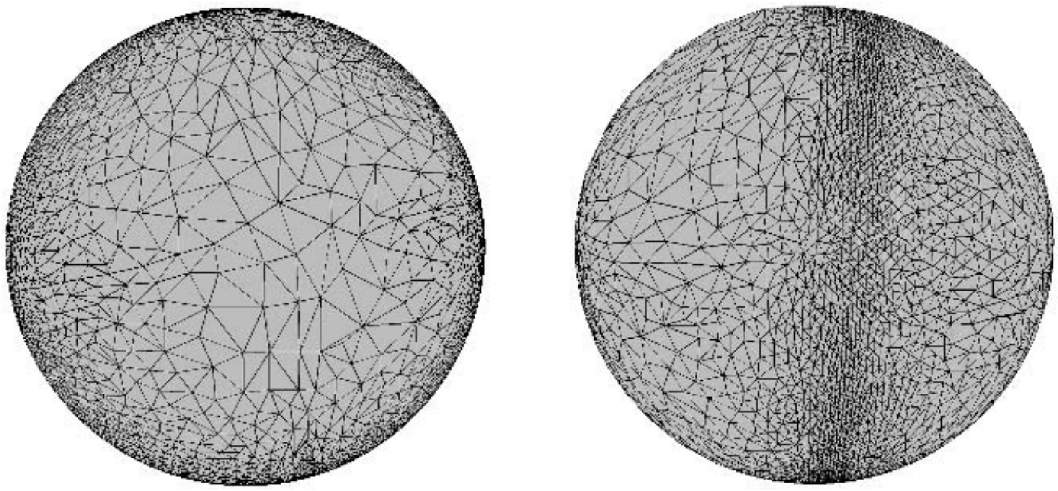
2.2.1 Adaptívny dynamický LOD

Keďže k výpočtu výsledného modelu dochádza počas behu programu, je možné v tomto výpočte brať do úvahy rôzne ďalšie aspekty scény. Najpoužívanejšími sú informácie o užívateľovi, a to pozícia a smer pohľadu. Ak sa pri výpočte výsledného modelu berú tieto informácie v úvahu, hovorí sa o adaptívnom LOD.



Obrázok 2.4: Ukážka adaptívneho LOD z vtáčej perspektívy. Pohľad pozorovateľa je naznačený červenými čiarami [12]

Na obrázku 2.4 je možné vidieť typický príklad adaptívneho LOD nad terénom. Napravo dole je zhustená a teda viac detailnejšia trojuholníková sieť. Tam sa nachádza pozorovateľ, ktorého pohľad je naznačený dvoma červenými čiarami. Ako je vidno, čím je terén ďalej od pozorovateľa, tým je menej kvalitnejší.



Obrázok 2.5: Ukážka siluety modelu. Vľavo je ukázané, čo vidí pozorovateľ a vpravo je takýto model ukázaný z boku. Pre pozorovateľa je dôležitá silueta. [12]

Obrázok 2.5 hovorí o ďalšej možnosti využitia informácie o pozorovateľovi a to o zachovaní tvaru modelu po jeho obvodu. Ponechanie takzvanej siluety podľa modelu v najvyššej kvalite vedie k tomu, že pozorovateľ menej vníma zmenu kvality modelu voči prostrediu. Tým pádom je možné ubrať kvalitu vo vnútornej časti modelu a na odvrátených častiach modelu, ktoré nie sú vôbec vidno.

2.2.2 Teselácia

O technológii s názvom teselácia sa úpenlivo začalo hovoriť až v posledných rokoch. Príčinou je hardvérová podpora na grafických kartách. Technológia sa často využíva na dynamický LOD s použitím algoritmov zo subdivide a displacement mapingu. Keďže výpočty sú prevedené až na grafickej karte, je metóda veľmi rýchla. Idea je v poslaní jednoduchého modelu do grafickej karty a za pomoci teselácie z neho spraviť zložitý model. Teselácia v jednoduchosti znamená rozdelenie polygóna na väčšie množstvo spojených polygónov podľa určitého algoritmu. Na grafickej karte teseláciu obsluhujú tieto 3 prvky:

- Control shader – spracúva dáta z vertex shaderu a riadi stupeň teselácie
- Generátor primitív – generuje nové vertexy
- Evaluation shader – počíta súradnice a atribúty vertexov novovytvorených polygónov

Ako je naznačené, tak do teselácie vstupujú dáta z vertex shaderu (v tomto prípade sa jedná o primitívum Patch) a na výstupe je novovytvorená geometria, ktorá pokračuje grafickou kartou ďalej do geometry shaderu.

2.3 Zjednodušovanie modelu

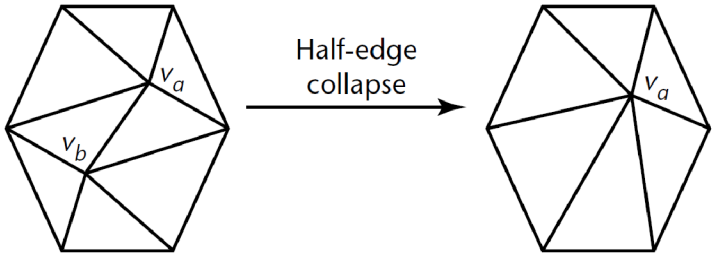
Ako bolo uvedené v predchádzajúcej kapitole, na samotné zjednodušovanie modelu sa používajú takzvané operátory. Tento pojem reprezentuje algoritmus, ktorý popisuje metódu ako zjednodušiť celý model alebo časť modelu. Podľa toho sa operátory delia na globálne alebo lokálne. Táto práca sa zameriava na lokálne operátory, ktoré svojou činnosťou ovplyvňujú malý vybraný kúsok na povrchu modelu. Vstupná časť povrchu, ktorá je operátorom spracovaná sa líši od každého operátora. Najčastejšie to býva hrana, vrchol alebo trojuholník. O výber najvhodnejšieho vstupu pre operátor sa starajú takzvané chybové metriky, ktorým je venovaná zvlášť kapitola. Nasledujúci text vysvetľuje princípy operátorov za predpokladaného použitia na manifold telesá.

2.3.1 Zjednotenie hrany

Zjednotenie hrany je najčastejšie používaný operátor. Vstup do tohto operátora je práve jedna zvolená hrana na povrchu modelu. V princíp spočíva v tom, že zvolená hrana sa zjednotí do práve jedného vrcholu. Tým sa zároveň odstránia aj dva polygóny, čím sa dosiahne zjednodušenie modelu. Metódu publikoval Hoppe v roku 1993 [6].

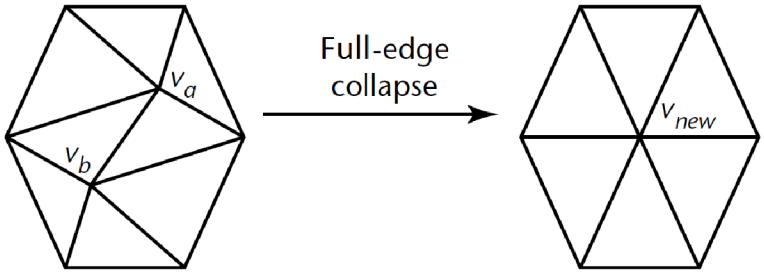
Algoritmus

Vybraná hrana sa skladá z dvoch vrcholov a najčastejšie z dvoch priľahlých polygónov. Operátor zmazaním hrany tieto dva polygóny odstráni a zároveň hranu zjednotí do jedného vrcholu. Metóda má dve základné formy, ktoré sa líšia pozíciou výsledného vertex po zlúčení hrany.



Obrázok 2.6: Princíp half-edge collapse. Vrchol V_b sa zjednotí do vrcholu V_a [12]

Pri základnom type, takzvanom half-edge collapse (obrázok 2.6), sa za výsledný bod vyberie jeden z dvoch vrcholov, ktoré sa už na hrane nachádzali. Na obrázku je to vrchol V_a. Do tohto bodu sa presmerujú všetky hrany vedené z vrcholu V_b.



Obrázok 2.7: Princíp full-edge collapse. Vrcholy V_a a V_b sa zjednotí do nového V_{new} [12]

Druhý typ sa volá full-edge colapse (obrázok 2.7), ktorý sa líši od half-edge collapse tým, že výsledný vrchol je nový bod nachádzajúci sa najčastejšie niekde na zjednocovanej hrane. Tento bod sa často určí v polovičke hrany, ale existujú zložité výpočty ako vrchol určiť tak aby operátor spôsobil čo najmenší úbytok kvality.

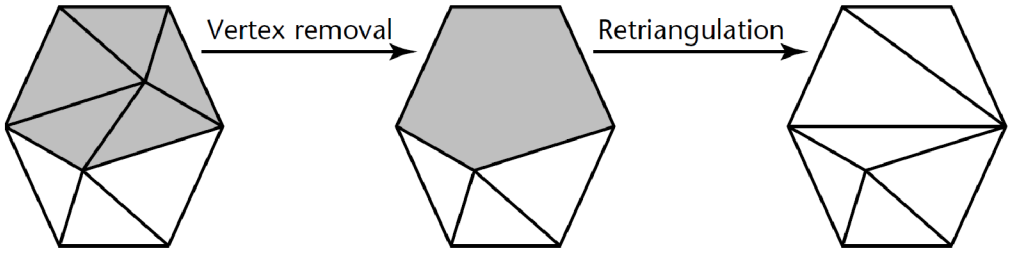
Hlavnou výhodou half-edge collapse je jednoduchosť a rýchlosť. Ďalšia nemenej dôležitá výhoda je korektné určenie atribútov vertexu. To je zapríčinené tým, že výsledný vrchol je jeden z pôvodných vrcholov a teda atribúty nie je nutné počítať. Na rozdiel pri full-edge collapse je nutné atribúty vypočítať z dvoch zlučovaných vertexov, čo v niektorých prípadoch môže byť zložité[2].

2.3.2 Odstránenie vertexu

Pre tento operátor je vstupom práve jeden vrchol na povrchu modelu a jeho úlohou, ako názov napovedá, je odstrániť tento vrchol. Každým aplikovaním sa z povrchu vymažú opäť dva polygóny. Metódu popísal ako prvý vo svojom článku Schroeder v roku 1992 [3]. Podrobný výskum o zacelení tejto diery vydali v roku 1997 Klein and Krämer [7], v ktorom dokázali, že tento operátor sa dá uskutočniť aj ako half-edge collapse, vysvetlený v kapitole 2.3.1.

Algoritmus

Po výbere vrcholu metrikou, je nutné tento vrchol odstrániť. Spoločne s ním sa odstránia aj všetky hrany a polygóny, na ktoré bol vrchol naviazaný. Po tomto úkone vznikne na povrchu modelu diera, ktorú treba zaceliť. Tomuto procesu sa hovorí retriangulácia a na zacelenie je nutné použiť práve o 2 polygóny menej oproti pôvodnému množstvu, ktoré priliehalo k odstraňovanému vertexu. Celý algoritmus odstránenia vrcholu je ukázaný na obrázk.



Obrázok 2.8: Princíp odstránenia vrcholu. Po zmazaní vrcholu sa diera retrianguluje [12]

[2] Niektoré algoritmy počítajú výsledný vrchol aj mimo zjednocovanú hranu a teda atribúty sa nedajú vypočítať lineárnou interpoláciou.

Vzniknutú dieru je možné zaceliť mnoho spôsobmi [7]. Je nutné si povšimnúť, že jedna z možností je rovnaká ako pri odstraňovaní hrany pomocou half-edge collapse. Z istého uhľa pohľadu sa dá považovať tento operátor za zovšeobecnenie half-edge collapse. Počet všetkých možností, ako vzniknutú dieru retriangulovať je daný vzťahom [7]:

$$C(i) = \frac{1}{i+1} * \binom{2i}{i} = \frac{1}{i+1} * \frac{(2i)!}{i!\,(2i-i)} = \frac{1}{i+1} * \frac{(2i)!}{i!\,i!} = \frac{(2i)!}{(i+1)!\,i!}$$

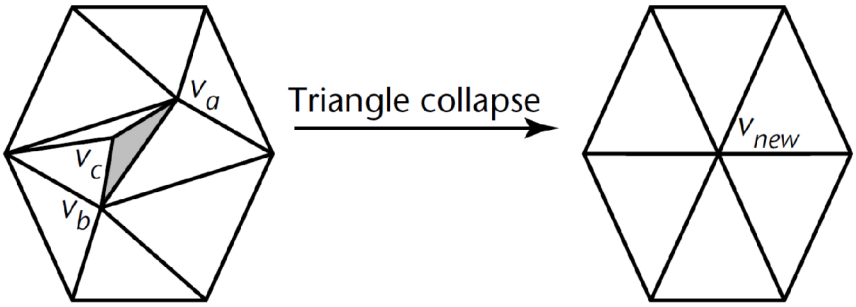
V rovnici C(i) značí výsledný počet možností retriangulácie pre dieru, ktorá má i + 2 strán. Vybrať z tohto množstva ten najsprávnejší postup môže byť pomerne zložité a zdĺhavé.

2.3.3 Zjednotenie trojuholníka

Vstupom pre tento operátor je trojuholník, ktorý sa zjednotí do jedného vrcholu. Táto metóda je oproti dvom predchádzajúcim zaujímavá hlavne v tom, že po aplikovaní operátora zmiznú až štyri trojuholníky. Vďaka tomuto faktu tento operátor dosahuje rýchlejšie zjednodušovanie ako jeho predchodci. S touto metódou prišiel prvý krát Hamann v roku 1994 [9] .

Algoritmus

Až sa za pomoci metriky určí najvhodnejší trojuholník, začne proces zlučovania. Tento algoritmus najprv vymaže štyri polygóny. Jeden, ktorý udala metrika a 3 ďalšie, ktoré zdieľajú nejakú hranu s vybraným trojuholníkom.



Obrázok 2.9: Princíp zjednotenia trojuholníku V_a, V_b, V_c do jedného vrcholu V_{new} [12]

Z pôvodných vertexov zlučovaného trojuholníka sa vypočíta jeden nový vrchol, do ktorého budú prevedené všetky hrany ostatných trojuholníkov, ktoré boli napojené na niektorý z pôvodných bodov. Možností umiestnenia nového bodu je opäť niekoľko, napríklad sa používa stred zjednocovaného trojuholníka. Tento proces je znázornený na obrázku 2.9.

Za povšimnutie stojí fakt, že táto metóda je opäť veľmi podobná metóde zjednotenia hrany. Jeden trojuholník je možné zjednotiť dvojitým aplikovaním tohto operátora.

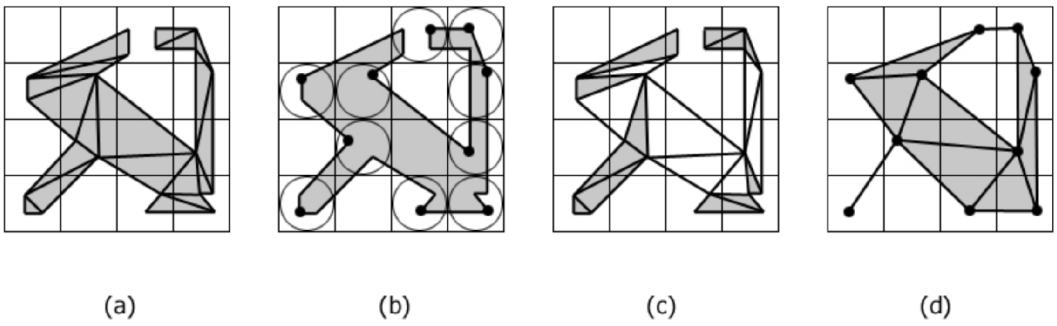
2.3.4 Ostatné operátory

Existujú a pomenej sa aj používajú operátory, ktoré sú svojím prístupom pomerne špecifické. Takéto operátory prinášajú trochu iný pohľad na zjednodušovanie a každý z nich má svoje klady a zápory.

Zjednotenie bunky

Táto metóda je zvláštna v tom, že jej vstupom je celý model. Avšak aj napriek tomu je považovaná za lokálny operátor. Keďže vstupom je celý model, nepoužíva sa žiadna metrika. Algoritmus je zo všetkých uvedených operátorov najrýchlejší. Bohužiaľ, na úkor rýchlosti sú však zjednodušené modely v mnohých prípadoch kvalitatívne nedostačujúce. Metóda bola publikovaná autorom Rossignac v 1993 [4] a neskôr rozšírená od Luebke a Low v 1997 [10][11].

Idea spočíva v rozdelení priestoru okolo modelu do buniek. Tieto bunky môžu, ale nemusia byť v tvare kocky. Veľkosť bunky sa môže určiť podľa veľkosti modelu. Napríklad ako percentový pomer z niektorej strany bounding boxu.

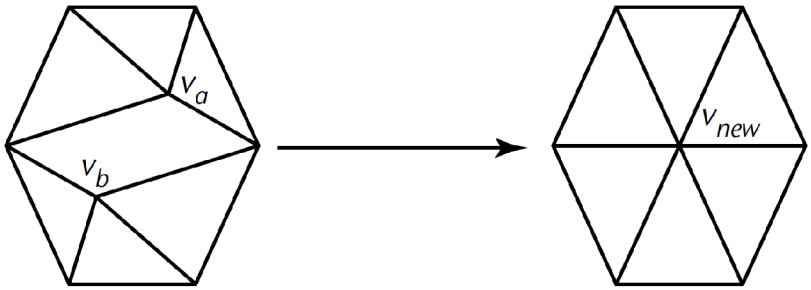


Obrázok 2.10: Princíp zjednotenia bunky [12]

Ako je na obrázku 2.10 vidno, ak sa v nejakej bunke nachádza viac vrcholov, budú nahradené jedným vertexom a všetky ostatné budú zmazané. Tak isto aj hrany vedené zo zmazaných vrcholov budú napokon presmerované do výsledného vertexu alebo budú odstránené. Zostávajúci vertex bunky môže byť zvolený mnoho spôsobmi. Pre príklad to môže byť jeden z vrcholov (ako je tomu na obrázku), ktoré sa v bunke nachádzali alebo ich priemer. Prípadne sa môže určiť ako výsledný vrchol stred bunky alebo jej nejaký krajný bod. Obrázok ďalej naznačuje hlavný problém tejto metódy, a to výslednú kvalitu. Operátor môže pospájať časti povrchu, ktoré pred tým vôbec spojené neboli. Prípadne naopak, veľmi ľahko môže odstrániť niektorú užšiu časť modelu.

Zjednotenie vrcholov

Operátor zjednotenie vrcholov je podobný ako zjednotenie hrany, len v tom rozdielu, že za vstup sú považované nespojené vrcholy. Keďže tieto vertexy nemajú spoločnú žiadnu hranu, nie je zmazaný po aplikácii ani žiadny polygón. Avšak okolité hrany boli aktualizované ako kebyže spojovacia hrana tam bola a vykonala sa klasická operácia zjednotenie hrany. Pre tento dôvod, býva často zjednotenie vrcholov volané aj zjednotenie virtuálnej hrany, alebo anglicky virtual-edge collapse [12].



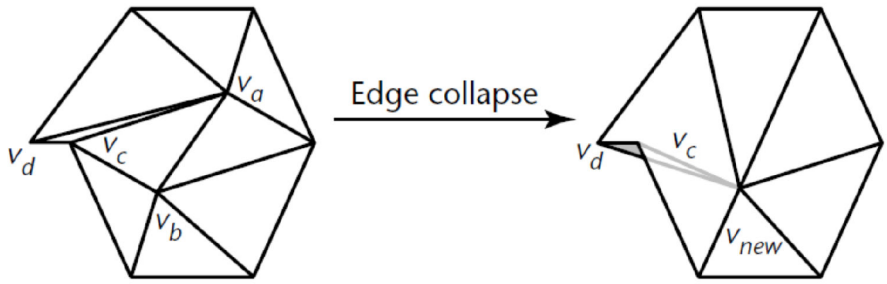
Obrázok 2.11: Princíp zacelenia diery pri operátoru zjednotenie vrcholov [12]

2.3.5 Problémy pri simplifikácii

Činnosť operátorov nie je jednoduchý proces a môže ho sprevádzať množstvo špecifických situácií, pri ktorých táto ich činnosť môže byť nekompletná alebo môže viesť k veľkému poškodeniu modelu. Tieto situácie môžu vznikať pri všetkých operátoroch a ich výskyt nie je zďaleka ojedinelý.

Prekrytie polygónov a otočenie normály

Tento problém je jedným z najčastejších javov pri simplifikácii a obzvlášť pri operátore zjednotenia hrany. Za istých podmienok sa môže stať, že po aplikovaní operátora sa trojuholníku inverzne zmení normála a zároveň sa prekrýva navzájom s iným.

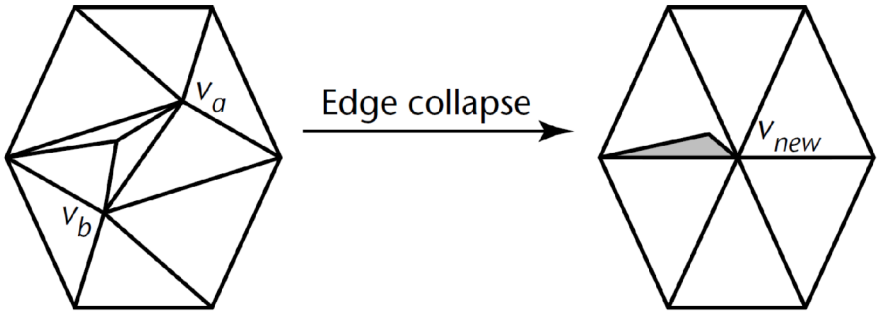


Obrázok 2.12: Princíp otočenia normály pri operácii zjednotenie hrany [12]

Tomuto javu sa dá zabrániť tak, že operátor dopredu vypočíta normály pre každý priľahlý trojuholník. Následne pre tieto trojuholníky vypočíta uhly medzi pôvodnými normálami a potencionálnymi novými normálami. Ak pri niektorom trojuholníku vznikne tento uhol väčší ako 90°, operátor nie je možné na danom mieste vykonať [12].

Tvorba non-manifold polygónov

Pri operátoru zjednotenie hrany vzniká ďalší problém. Ak vrcholy hrany, ktorá je pripravená na mazanie, majú aspoň 3 spoločné susedné vrcholy, tak po aplikovaní operátora vznikne minimálne jeden non-manifold polygón. Takýto model neskôr môže robiť problémy ako pri simplifikácii, tak aj pri iných činnostiach s modelom.



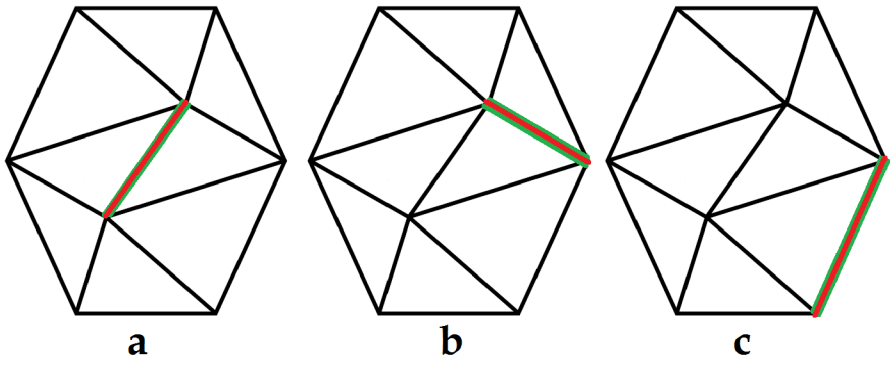
Obrázok 2.13: Princíp vytvorenia non-manifold trojuholníka ako nežiadaný efekt operátoru zjednotenie hrany [12]

Zachovanie hraníc modelu

V praxi sa bohužiaľ vo veľa prípadoch používajú non-manifold modely. Pri simplifikácii je s takýmito modelmi problém. V non-manifold modeloch je možné rozdeliť hrany do troch skupín:

- Vnútorné hrany
- Hraničnо-susedné hrany
- Hraničné hrany

Význam jednotlivých kategórií je znázornený na nasledujúcom obrázku.



Obrázok 2.14: a) Vnútorná hrana b) Hraničnо-susedná hrana c) Hraničná hrana

Pre zachovanie tvaru modelu je dôležité aby sa hraničné hrany na modelu nesimplifikovali. Pre hraničnо-susedné hrany sa doporučuje simplifikáciu tiež vynechať alebo prípadne použiť len half-edge variantu zjednotenia hrany. Pri tomto úkone sa hrana zjednotí do hraničného vertexu aby sa neporušil tvar modelu. Vnútorné hrany je možné simplifikovať svojvoľne.

2.4 Chybové metriky

Ako bolo uvedené v predchádzajúcej kapitole, pred aplikovaním operátora je nutné najprv nájsť najvhodnejšie miesto na modely, kde tento operátor bude aplikovaný. Pri hľadaní tohto miesta sa používa takzvaná chybová metrika. Tá slúži na ohodnotenie miesta na modely podľa konkrétnych kritérií pre danú metriku. Takéto ohodnotenie najčastejšie vypovedá o spôsobenej chybe, keď na dané miesto bude aplikovaný operátor. Následne sa z takto ohodnotených miest vyberie to s najmenšou hodnotou (chybou) a na zvolené miesto sa aplikuje konkrétny operátor.

2.4.1 Dĺžkové metriky

Dĺžkové metriky patria medzi najzakladanejšie metriky. Ich kvalita nie je vždy dostatočná, ale v praxi sa vyskytujú situácie kedy použitie týchto metrík je veľmi vhodné, dokonca potrebné.

Ohodnotenie hrany

Jedna z metrík pre operátor zjednotenie hrany je ohodnotiť hrany podľa ich dĺžky.

$$cost\ (A,B) = \|A - B\| = \sqrt{\left((A_x - B_x)^2 + \left(A_y - B_y\right)^2 + (A_z - B_z)^2\right)}$$

Pri tomto prístupe ale nie je zaručený najlepší výsledok, lebo odstránenie najkratšej hrany nutne nemusí znamenať spôsobenie najmenších škôd na modely. Táto metrika je však v istých situáciách nenahraditeľná. Napríklad pri implementácii funkcie zmazania všetkých hrán, ktoré sú kratšie ako „jeden pixel“. V tomto prípade sa nedá použiť žiadna iná metrika, len táto metrika.

Ohodnotenie vertexu

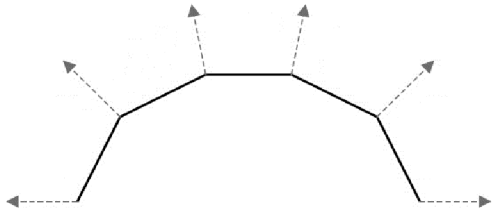
Na samotnom vertexu sa nedá nič základného zmerať, ako tomu bolo pri hrane. Preto dĺžkové metriky na odstránenie vertexu pracujú hlavne s okolím vertexu. Pre príklad je možné merať najdlhšiu hranu vedenú z vrcholov a tým určiť pre jednotlivé vertexy najväčšiu chybu. Ďalšou zaujímavou metrikou môže byť najkratšia vzdialenosť medzi vzniknutou dierou a samotným vrcholom.

Ohodnotenie trojuholníka

Možná metrika pre operátor zjednotenie trojuholníka je jeho obvod. Je však nutné zdôrazniť, že pri zjednotení trojuholníka dochádza aj k odstráneniu priľahlých troch ďalších trojuholníkov. Preto je vhodné tieto trojuholníky započítať do ohodnotenia. Napríklad je možné spočítať všetky strany všetkých priľahlých trojuholníkov.

2.4.2 Dĺžkové metriky s pridaním normál

Veľký nedostatok samotných dĺžkových metrík je, že neberú v úvahu zakrivenie povrchu. Tento faktor pri zjednodušovaní je dôležitý a preto sa vyššie uvedené metriky dajú rozšíriť. Rozšírenie spočíva v započítaní normálových vektorov príslušných vertexov alebo trojuholníkov do výpočtu ohodnotenia.



Obrázok 2.15: Ukážka normálových vektorov z vrcholov modelu

Z normálových vektorov sa dá zistiť zakrivenie okolitého povrchu. Pre príklad zjednotenia hrany a metriky uvedenej v minulej kapitole, môžeme výpočet upraviť nasledovne:

$$cost\ (A,B) = \|A - B\| * abs(1 - \cos\alpha)$$

V rovnici symbol α znamená uhol medzi normálami vrcholov A a B. Z rovnice vyplýva, že čím je uhol α menší, tým je dôraz na veľkosť hrany menší. Naopak, pri uhle väčšom ako 90° je možné hranu hodnotiť až do dvojnásobku jej dĺžky. Podobným započítaním normál je možné upraviť všetky metriky z predchádzajúcej kapitoly.

2.4.3 Quadric Error Metric

Táto metrika bola predstavená v roku 1997 na konferencii Siggraph pánmi Garland a Heckbert [18]. Metóda, často označovaná svojou skratkou QEM, sa teší veľkej popularite. Býva často implementovaná a aj vylepšená rôznymi modifikáciami iných autorov. Metrika je navrhnutá ako pre operátor zjednotenie hrany tak aj pre operátor zjednotenie vrcholov. Táto kapitola je však zameraná na QEM pre zjednotenie hrany.

Algoritmus

Ako názov napovedá, metóda pracuje s kvadrikou. V prvom kroku je nutné vypočítať kvadriku pre každý polygón ako 4x4 maticu Q_p. Polygón je vlastne výrez 3D plochy, ktorá sa vyjadruje nasledovne:

$$ax + by + cz + d = 0$$

Pričom platí:

$$a^2 + b^2 + c^2 = 1$$

Vektor [a, b, c] označuje normálu roviny a hodnota *d* je posun plochy od bodu [0, 0, 0]. Z týchto štyroch hodnôt je definovaný vektor *p* = [a, b, c ,d]. Následne je možné vyjadriť vyššie spomínanú maticu Q_p ako kvadriku:

$$Q_p = pp^T = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}$$

Keď je pre všetky polygóny matica Q_p vypočítaná, je nutné vypočítať maticu Q_v pre každý vertex. Matica Q_v sa vypočíta ako súčet matíc Q_p všetkých polygónov obsahujúci vrchol *v*:

$$Q_v = \sum_{p \in planes(v)} Q_p$$

Po vypočítaní všetkých Q_v pre každý vertex, je model pripravený pre samotné zjednodušovanie. Ako je vyššie zmienené, QEM používa operátor zjednotenie hrany. Forma operátora je full-edge collapse a QEM obsahuje matematiku aj na výpočet presnejšej pozície nového vrcholu. Ohodnotenie hrany je závislé na novom vrchole po zjednotení hrany. Čiže je nutné najprv vypočítať výsledný vertex pre každú hranu a až za pomoci tohto vrcholu sa hrana ohodnotí. Výpočet nového vrcholu *V* z hrany obsahujúcej vrcholy *A* a *B* je nasledovný:

$$Q = Q_A + Q_B$$

$$V = \begin{bmatrix} Q_{11} & Q_{12} & Q_{13} & Q_{14} \\ Q_{12} & Q_{22} & Q_{23} & Q_{24} \\ Q_{13} & Q_{23} & Q_{33} & Q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Z vypočítanej potencionálnej pozície nového vertexu je možné ohodnotiť hranu. Ohodnotenie je chyba, ktorá bude spôsobená zjednotením hrany:

$$cost(A, B) = V^T Q\, V$$

QEM je pomerne rýchla a kvalitná metrika. Jej základná verzia má však jeden nedostatok a to, že nerieši výpočet atribútov pre nové vrcholy. Tento nedostatok je riešený v jej modifikáciach ako je napríklad New QEM od pána Hoppe [19].

2.4.4 Stan Melax

Názov kapitoly nezodpovedá v tomto prípade názvu metriky ale menu autorovi metriky. Stan Melax túto metriku publikoval v roku 1998 v Game Developer Magazine [13]. Metóda je half-edge collapse a pri ohodnotení hrany pracuje hlavne s normálami povrchu:

$$cost(A, B) = |A - B| * \max_{i \in T_A}\left(\min_{j \in T_{AB}}\left(\frac{1 - dot(N_i, N_j)}{2}\right)\right)$$

Kde T_A je množina trojuholníkov, ktoré obsahujú vertex A, následne množina T_{AB} obsahuje trojuholníky, ktoré majú vrcholy A i B zároveň. Operácia *dot* znamená skalárny súčin a N_i je vyjadrenie normály pre trojuholník *i*. Z povahy výpočtu je vidieť, že ohodnotenie *cost(A, B)* nie je rovnaké ako *cost(B, A)*. Preto pre ohodnotenie hrany je nutné vypočítať obe hodnoty a následne vybrať menšiu hodnotu a podľa toho aj výsledný vrchol po zjednotení. Hodnota funkcie *cost(A, B)* zodpovedá ohodnotení ak sa vrchol A zjednotí do vrcholu B.

3 Podpora LOD v knižniciach

Techniky Level of Detail sú v dnešnej dobe neoddeliteľnou súčasťou návrhu 3D scén. Preto sa vývojári knižníc pre 3D vykresľovanie snažia nielen poskytnúť množstvo funkcií a nástrojov na obsluhu týchto techník ale snažia sa docieliť aj ich jednoduché a intuitívne použitie.

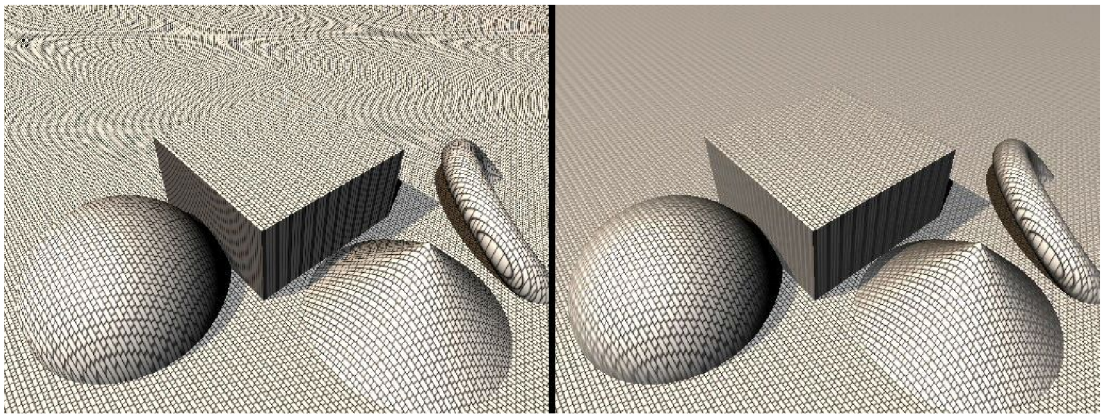
3.1 LOD v OpenGL

Multiplatformná knižnica OpenGL slúži pre vykreslovanie 2D a 3D vektorovej grafiky. Primárne slúži aj ako prostredník medzi aplikáciou a grafickou kartou. V zásade samotné OpenGL hlavne rieši ako zobraziť výslednú scénu a nerieši rozpoloženie scény, ovplyvňovanie objektov navzájom alebo vyššiu logiku konkrétnej aplikácie. Preto v OpenGL je ponuka LOD techník pomerne obmedzená. Obsahuje napríklad obslužné funkcie pre vyššie spomínanú teseláciu, ale pre túto prácu je dôležitejšie zmieniť technológiu MipMapping.

3.1.1 MipMapping

Techniku MipMapping ako prvý prezentoval vo svojej práci pán Lance Williams [17] a to už v roku 1983. Účel tejto techniky napovedá aj samotné slovo „MIP“, ktoré je skratka od latinského výrazu „multum in parvo“ – voľne preložené ako „veľa vecí na malom mieste“ [17].

Na otextúrované objekty je možné sa pozerať rovnako ako na všetky ostatné objekty v scéne a to z rôznych vzdialeností od kamery. Ak sa v dynamickej scéne otextúrovaný objekt vzďaľuje, je vhodné aby textúra tiež zmenila veľkosť, a to úmerne so znižovanou veľkosťou vykresleného obrazu. Na dosiahnutie tohto cieľa je nutné aby OpenGL zmenšilo textúry na vhodné veľkosti pre mapovanie na vzdialenejšie objekty. Takto vopred zmenšené textúry sa volajú mipmapy a hlavný účel je odstránenie aliasingu. [16]



Obrázok 3.1: Ukážka odstránenia antialiasingu pomocou MipMappingu [povplace.com]

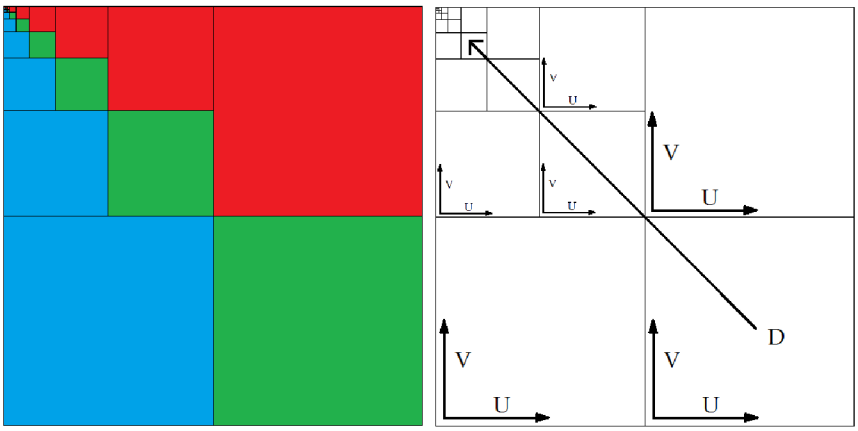
Spomínané zmenšovanie textúr je v OpenGL predpočítané z dodanej textúry v maximálnom rozlíšení. Ak sa chce MipMapping použiť, je nutné aby sa predpočítali všetky zmenšené textúry o veľkosti násobku 2. Napríklad, ak má najväčšia textúra rozmery 64 x 16 texelov, bude vypočítaných

6 ďalších textúr o rozmeroch 32 x 8, 16 x 4, 8 x 2, 4 x 1, 2 x 1 a 1 x 1. Menšie textúry sú zväčša filtrované a hodnota každého jedného texelu je zodpovedajúca priemeru štyroch texelov z väčšej textúry (ak väčšia textúra poskytuje dostatočný počet texelov). [16]

Obrázok 3.2: Ukážka zmenšených textúr a jednotlivých úrovní detailu [tomshardware.com]

Pri použití MipMappingu OpenGL automaticky určí na základe veľkosti vykreslovaného objektu v pixeloch, ktorú zmenšenú textúru (alebo originál) použije na mapovanie objektu. MipMapping vyžaduje extra výpočty a aj pamäť pre uloženie rôznych veľkostí textúr. Avšak je doporučené MipMapping používať, kvôli odstráneniu vyššie spomínaného aliasingu.[16]

Pán Williams [17] okrem myšlienky MipMappingu navrhol aj veľmi efektívne uloženie zmenšených textúr v pamäti. Vo výsledku všetky textúry (vrátane originálu) dokopy zaberajú len o tretinu viac než ich originálna textúra a sú uložené v pamäti pokope na jednom mieste. Táto optimalizácia sa dosiahla tým, že sa vytvorí nová textúra o dvojnásobných rozmeroch ako je originál, ale na rozdiel od troch kanálov (kanály pre červenú, zelenú a modrú farbu) je v textúre použitý iba jeden kanál. Táto jednokanálová textúra je potom pomyselne rozdelená na štyri časti o veľkosti originálnej textúry. Následne sú kompletné farebné informácie o originálnej textúre uložené do troch z týchto častí, pričom každá časť reprezentuje jeden farebný kanál. Posledná štvrtá časť slúži na uloženie zmenšených textúr rovnakým spôsobom ako bola uložená originálna textúra. MipMapy sú indexované pomocou troch koordinátov: U, V a D. U a V sú priestorové koordináty textúry. D je premenná používaná na index a interpoláciu medzi rôznymi veľkosťami textúr. [17]



Obrázok 3.3: Ukážka ako sú usporiadané kanály v MipMape (prekreslený do OpenGL z [17])

V OpenGL je MipMapping možné obsluhovať cez nastavenia konkrétnej textúry. Tie je možné meniť za pomoci funkcie *glTexParameteri*. Na zapnutie a základné nastavenie MipMappingu slúžia tieto parametre textúry:

- TEXTURE_MIN_FILTER
  - Určuje akým spôsobom bude vybraná výsledná hodnota fragmentu
  - Hodnota NEAREST
    - mapuje najbližší texel (bez MipMappingu)
  - Hodnota LINEAR
    - lineárne mapuje najbližšie texely (bez MipMappingu)
  - Hodnota NEAREST_MIPMAP_NEAREST
    - mapuje najbližší texel z najbližšej MipMapy
  - Hodnota NEAREST_MIPMAP_LINEAR
    - lineárne mapuje najbližšie texely z najbližšej MipMapy
  - Hodnota LINEAR_MIPMAP_NEAREST
    - mapuje najbližší texel z lineárnej interpolácie medzi MipMapami
  - Hodnota LINEAR_MIPMAP_LINEAR
    - lineárne mapuje najbližšie texely z lineárnej interpolácie medzi MipMapami
- TEXTURE_MAG_FILTER
  - Určuje ako bude získaná hodnota pre fragment, ak je nutné zväčšovanie textúry
  - Hodnota NEAREST
    - Rovnaký význam ako pri TEXTURE_MIN_FILTER
  - Hodnota LINEAR
    - Rovnaký význam ako pri TEXTURE_MIN_FILTER
- TEXTURE_BASE_LEVEL
  - Štartovací level mipmáp
  - Hodnota INT
- TEXTURE_MAX_LEVEL
  - Maximálny level mipmáp
  - Hodnota INT

3.2 LOD v OpenSceneGraph

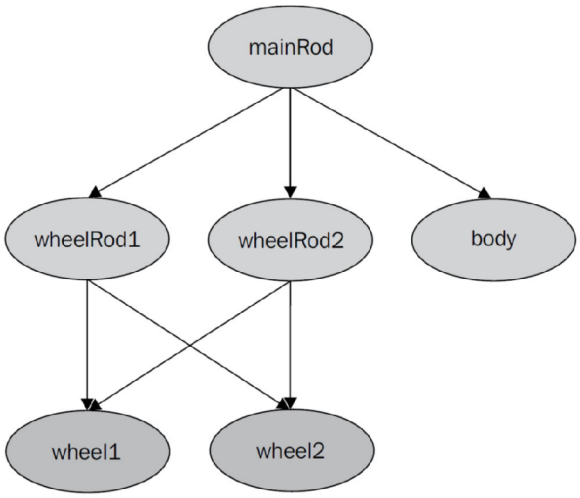
Ako bolo povedané v úvode predchádzajúcej kapitoly, OpenGL nerieši logiku v scéne. Pre tento úkon slúžia nadstavbové knižnice a systémy na vyššej abstraktnej úrovni. Poskytujú nástroje na prípravu a optimalizovanie scény ako celok. Medzi časté optimalizačné techniky patria aj techniky LOD a v knižnici OpenSceneGraph tomu nie je inak.

3.2.1 Knižnica OpenSceneGraph

Keďže sa táto práca prevažne venuje technikám LOD v knižnici OpenSceneGraph (OSG), tak je potrebné stručne popísať ako samotné OSG funguje. OSG je kvalitne navrhnutá knižnica na renderovanie so systémom založeným na teórii grafu scény. Ten zaznamenáva vykresľovacie príkazy a dáta do pamäti pre ich neskoršie použitie. To umožňuje aby systém vykonával rôzne optimalizácie pred vykreslením. Zároveň môže pre riešenie komplexných scén použiť viac vlákien. [15]

Graf scény je všeobecná dátová štruktúra, ktorá definuje priestorové a logické vzťahy pre efektívnu správu a zobrazovanie grafických dát. Tá je typicky reprezentovaná ako hierarchický graf, ktorý obsahuje kolekciu grafických uzlov, vrátane koreňového uzla najvyššej úrovni. Graf obsahuje množinu skupinových uzlov, ktoré môžu mať ľubovoľný počet potomkov a ďalej množinu listových uzlov, ktoré nemajú žiadnych potomkov. Tieto listové uzly slúžia spoločne ako spodná vrstva stromu. Prirodzené chovanie skupinových uzlov je, že šíria svoje pôsobenie aj na všetkých ich potomkov. Typický graf scény však nedovolí orientovaný cyklus (ak je skupina uzlov uzavretých do orientovanej smičky) alebo izolovaný uzol (nemá žiadneho potomka ani rodiča). [15]

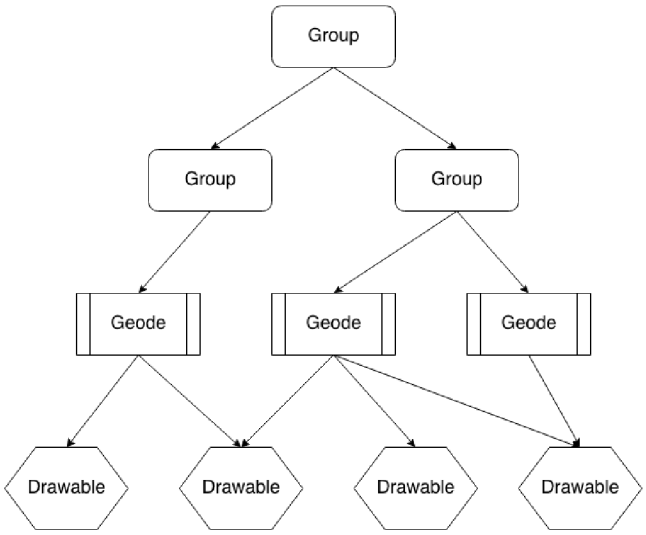
Bežne sa používa, že niektoré uzly majú viac ako jeden rodičovský uzol. V takom prípade je uzol považovaný za inštanciu a graf scény môže byť definovaný ako orientovaný acyklický graf. Použitie inštancií môže slúžiť na vytvorenie veľa zaujímavých efektov, vrátane zdieľania dát a multi-pass vykresľovania. [15]



Obrázok 3.4: Ukážka logiky grafu scény. [14]

V knižnici sa nachádzajú 3 základné druhy uzlov a to Group, Geode a Drawable. Z týchto uzlov je možné naďalej dediť a vytvárať iné špecifickejšie uzly. Pre uzol typu Group je typická možnosť vlastniť rôzny počet potomkov. Títo potomkovia bývajú najčastejšie ďalšie uzly typu Group

alebo uzly typu Geode. Uzol Geode slúži ako posledný uzol pred listami stromu. Geode môže obsahovať množstvo potomkov typu Drawable, pričom do uzla Group nie je možné pridať Drawable bez použitia uzlu Geode. Uzol typu Drawable je vlastne list stromu a slúži pre reprezentáciu geometrie.



Obrázok 3.5: Ukážka grafu scény v OSG. Na grafu sú znázornené základné typy uzlov.

Pred vykresľovaním je uskutočňovaný takzvaný priechod grafom, ktorý postupuje cez uzly od koreňa ku listom. Týmto priechodom sa vykonáva príprava scény a nastavovanie premenných pre OpenGL. Napríklad v špecifických uzloch podedených z Group môže byť nastavenie pozície a rotácie. Po priechode cez tento uzol budú všetci potomkovia posunutí na isté miesto a otočení podľa nastavenia rodičovského uzla. Problematika priechodov grafom a všeobecne vykresľovania je ďaleko rozsiahlejšia, ale pre potreby tejto práce takýto výklad je dostatočný.

Keďže je knižnica OpenSceneGraph nadstavbou nad OpenGL, je možné obsluhovať aj MipMapping. Ten je možné nastavovať spôsobom podobným, ako bolo uvedené v predchádzajúcej kapitole. Rozdiel je len v reprezentácii textúry. V OSG je každá textúra reprezentovaná zvlášť inštanciou objektu z triedy *Texture* (alebo z nej podedených tried). V tomto objekte sú obslužné funkcie na nastavovanie a prácu s MipMappingom.

3.2.2 Uzol Lod

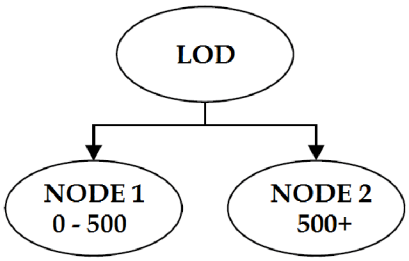
Uzol LOD je uzol v grafe scény, ktorý je podedený z Group a teda môže mať potomkov. Je typický reprezentant pre diskrétny LOD. Títo potomkovia sa používajú práve na reprezentáciu rovnakého objektu v rôznych detailoch, zoradených od najzložitejšieho po najjednoduchší. Tieto levely detailu potrebujú mať nastavené hraničné kritéria pre zmenu detailu. Kritéria sa nastavujú pre každého potomka formou rozsahu od - do a reprezentujú kedy daný detail má byť aktívny. Nastavenie týchto rozsahov je možné za pomoci funkcie *setRange()* alebo je možné ho nastaviť pri pridávaní potomka.

Pri štandardnom priechode grafom sa uzol rozhodne, ktorá časť stromu bude aktívna na základe dostupných dát o pozorovateľovi. Uzol môže byť nastavený v dvoch štandardných módoch. Prvý mód je, že detail sa prepína na základe vzdialenosti od pozorovateľa, ktorá sa nastavuje za pomoci konštanty *DISTANCE_FROM_EYE_POINT*. Druhý mód je, že rozsahy prepnutia sú udané vo

veľkosti pixelov na obrazovke, pričom sa meria bounding sphere okolo objektu. Tento mód je možné nastaviť za pomoci konštanty *PIXEL_SIZE_ON_SCREEN*. [15]

```
//Jednoduchá demonštrácia LOD uzlu
osg::LOD *lodNode = new osg::LOD;
lodNode->setRangeMode(DISTANCE_FROM_EYE_POINT);
lodNode->addChild( node1 );
lodNode->addChild( node2, 500.0f, FLT_MAX );
lodNode->setRange( 0, 0.0f, 500.0f );
```

V uvedenom kóde je najprv vytvorený uzol LOD a nastavený mód rozsahov na vzdialenosť od pozorovateľa. Následne sú pridaní dvaja potomkovia, avšak pri druhom je nastavený hneď aj rozsah viditeľnosti (vyššie spomínané kritériá). Pre prvého potomka je rozsah nastavený dodatočne. Z kódu vyplýva, že od vzdialenosti 0 až po vzdialenosť 500 bude zobrazený potomok „node1“ a vo väčšej vzdialenosti bude zobrazený potomok „node2“. [15]



Obrázok 3.6: Ukážka použitia LOD uzlu. NODE1 bude zobrazené od vzdialenosti 0 po 500 a nad 500 bude zobrazený NODE2

3.2.3 Uzol PagedLod

OSG obsahuje aj rozšírenie uzlu LOD, pod názvom PagedLOD. Klasický LOD uzol vyžaduje aby všetci jeho potomkovia, čiže všetky úrovne detailu boli vždy prítomné v pamäti. Toto chovanie je podedené z Group. Z toho vyplýva, že pre veľmi veľké scény s veľkým množstvom LOD modelov môže byť klasický uzol LOD nepoužiteľný. V takýchto scénach by bola spotreba pamäti neúnosne veľká aj keď je v jednom čase obvykle vykresľovaný iba jeden potomok z každého LOD uzla.

Odpoveďou na tento problém je uzol PagedLOD. Tento uzol dedí z LOD uzla a dodáva schopnosť načítavať potomkov neskôr, až keď sú skutočne potrebné. PagedLOD môže mať potomka klasicky v sebe, ako je tomu pri LOD uzle, avšak potomok môže byť definovaný aj ako cesta k súboru, z ktorého sa má načítať.

```
//Jednoduchá demonštrácia PagedLOD uzlu
osg::PagedLOD *plodNode = new osg::PagedLOD;
plodNode->setRangeMode(DISTANCE_FROM_EYE_POINT);
plodNode->addChild( node1, 500.0f, FLT_MAX );
plodNode->setFileName( 1, “node_2.osg” );
plodNode->setRange( 1, 0.0f, 500.0f );
```

V uvedenom kóde je použité klasické pridanie potomka „node1“. Zmena nastáva pri pridávaní druhého potomka. Na plné využitie PagedLOD uzlu stačí nastaviť pre daného potomka

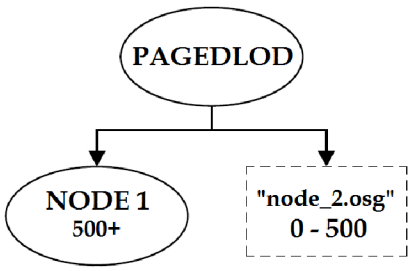
s indexom 1 („node1“ má index 0) názov súboru a rozsah. Je treba si povšimnúť jeden rozdiel voči príkladu na LOD uzol. Pri PagedLOD je nutné na nižších indexoch mať nižšie úrovne detailu. Preto prvý potomok má pri tomto príklade nastavený rozsah od 500 do nekonečna.

Keď je potomok potrebný, spúšťa sa načítavanie zo súboru. Tento proces je vykonávaný asynchrónne, takže pri načítaní veľkých modelov sa nenarušuje priebeh vykresľovania. O načítavanie vo zvlášť vlákne sa stará objekt DatabasePager. Vždy, keď sa vykonáva priechod PagedLOD uzlom, je určený potomok, ktorý by sa mal vykresliť. Určenie potomka je obdobné ako pri LOD uzle, avšak keď nastane situácia, že potrebný potomok nie je aktuálne načítaný, pripraví sa požiadavka pre DatabasePager o jeho načítanie. Táto požiadavka obsahuje prioritu načítavania, ktorá je určená na základe konfigurácie užívateľa, scale faktoru a vzdialenosti, prípadne veľkosti v pixeloch.

Napríklad ak vzdialenosť užívateľa od modelu je práve tesne okolo hranice prepnutia, tak detailnejší potomok, ktorý by mal byť prepnutý, nemusí byť okamžite načítaný. V tomto prípade sa nastaví nízka priorita požiadavku lebo je predpoklad, že pri hraniciach nie sú až tak veľké vizuálne rozdiely medzi jednotlivými úrovňami modelu. Ale v prípade, že užívateľ dôjde bližšie k modelu a ten stále nemá načítaného patričného potomka, PagedLOD uzol vygeneruje požiadavku novú s už vyššou prioritou.

Proces znovu posielania požiadavku sa deje pri každom priechode PagedLOD uzlu. Totižto je predpoklad, že podmienky načítania sa neustále menia. Takáto forma aktualizácie priority načítavania sa deje až kým DatabasePager kompletne nenačíta požadovaného potomka a nevloží ho do PagedLOD uzlu.

Takáto forma načítavania na pozadí však platí len pre prechod z nižšieho detailu na vyšší. Je dôležité povedať, že prechod na nižší detail je možný uskutočniť hneď. Je to spôsobené tým, že PagedLOD nikdy nemaže nižšie detaily od aktuálne potrebného. Naopak vyššie detaily bývajú odstránené. Nie sú však odstraňované hneď, keď nie sú potrebné. Zaznamenáva sa moment posledného zobrazenia a model sa odstráni až nejaký čas po poslednom použití a pri zahltení pamäti. Je totiž pravdepodobné, že táto úroveň detailu bude v bezprostredne blízkej dobe po poslednom zobrazení znova potrebná.



Obrázok 3.7: Ukážka použitia PagedLOD uzlu. Súbor node_2.osg bude načítaný a zobrazené od vzdialenosti 0 po 500 a nad 500 bude zobrazený NODE1

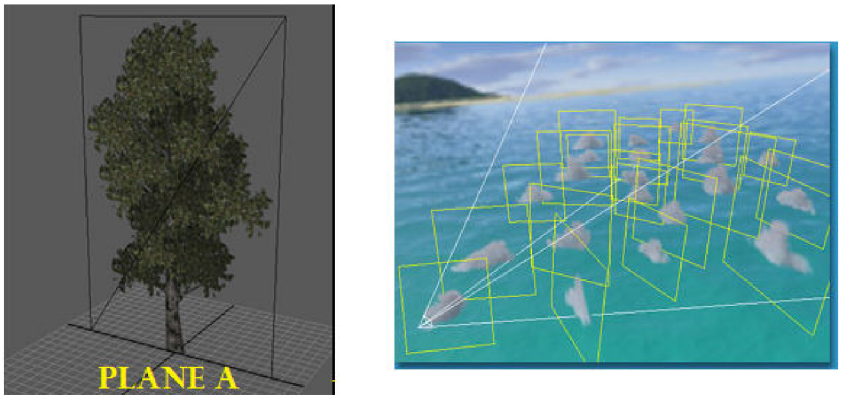
3.2.4 Uzol Billboard

Myšlienky Level of Detail sú zamerané na čo najväčšiu redukciu vykresľovaných dát. Jednou z najlepších redukcií sú práve techniky ako je Billboard alebo Impostor, ktoré prinášajú na redukciu nový pohľad.

V 3D scéne Billboard reprezentuje objekt, ktorý je zobrazovaný najčastejšie ako obrázok. V OSG je Billboard dedený z triedy Geode a teda môže v ňom byť akákoľvek geometria. Avšak najbežnejšie použitie je práve ako jedna obdĺžniková alebo štvorcová plocha zkonštruovaná z dvoch

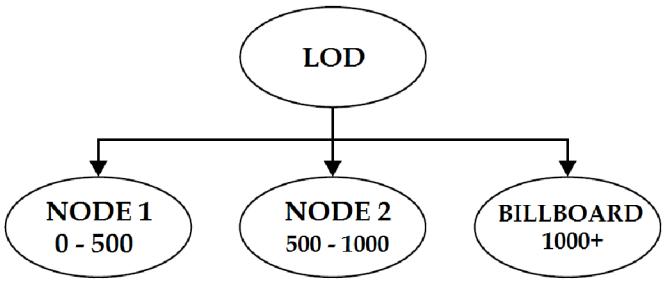
trojuholníkov a následne patrične otextúrovaná. Dôležitá vlastnosť Billboardu je, že objekt je vždy natočený jedným smerom. Tento smer spravidla súvisí s pozorovateľom.

Napríklad dymový mrak je možné pekne spraviť za pomoci Billboardu. V tomto prípade sa otextúrovaná plocha neustále otáča k pozorovateľovi. Druhý typický príklad je simulácia stromov. Teraz nie je vhodné aby sa plocha otáčala iba priamo k pozorovateľovi ale je nutné k tejto rotácii pridať dodatočnú os, okolo ktorej sa Billboard bude otáčať. Pre uvedený príklad stromu je tá os daná kolmo nahor.



Obrázok 3.8: Praktická ukážka použitia Billboardu. Naľavo je zobrazený strom ako Billboard a napravo je zobrazená scéna s oblakmi, ktoré sú natočene k pozorovateľovi (vľavo dole)

Pochopiteľne každý objekt v scéne môže mať svoju Billboard reprezentáciu, ktorá z blízkeho pohľadu nepôsobí kvalitne, avšak z veľkej diaľky je úplne dostatočná. Preto realizácia Billboardov je vskutku jednou z najpopulárnejších techník. Široké využitie majú v počítačových hrách a v real-time grafických aplikáciách, v ktorých sa používajú okrem zobrazovania efektov aj ako práve najnižšia úroveň detailu pri použití technológie LOD [15].



Obrázok 3.9: Ukážka použitia Billboardu v uzle LOD, kde bude zobrazený nad vzdialenosť 1000

3.2.5 Uzol impostor

Uzol Impostor predstavuje podobnú technológiu Billboardu. Na prvý pohľad sa môže zdať jednoznačne lepšia, avšak ako Impostor tak aj Bilboard má svoje klady i zápory. Impostor v OSG dedí od uzlu LOD a líši sa od tohto uzlu v zásade jedným hlavným nastavením a tým je hodnota prahu. Tento prah predstavuje istú vzdialenosť modelu od pozorovateľa. Ak je aktuálna vzdialenosť

od pozorovateľa menšia ako uvedený prah, Impostor sa chová ako štandardný uzol LOD. Rozdiel nastáva ak sa pozorovateľ nachádza za týmto prahom. Vtedy uzol začne fungovať tak, že aktuálnu úroveň detailu nevykreslí štandardne ako polygonálny model ale predkreslí si takýto model z pozície pozorovateľa do textúry. Následne je na pozíciu modelu vložený Billboard, ktorému je nastavená predkreslená textúra a ten sa používa až do kedy pozorovateľ nepríde bližšie k modelu a nedostane sa do vzdialenosti menšej ako je prah. Je nutné zdôrazniť, že textúra na Billboarde sa prekresľuje z polygonálneho modelu vždy, keď sa takzvaná chyba pixelov zväčší nad určitú hodnotu. Chyba pixelov sa dá popísať takto:

- Pre Billboard je uložená prvotná pozícia všetkých 4 vrcholov
- Billboard sa v čase rotuje podľa pozorovateľa a teda mení pozíciu svojich vertexov
- Pri renderovaní je skontrolovaná chyba pixelov nasledovne:
    1. Prvotné a aj aktuálne pozície vrcholov sú projektované s MVP maticou na 2D plochu (obrazovka)
    2. Po projekcii sú zmerané 2D vzdialenosti medzi vyprojektovanými aktuálnymi a pôvodnými vrcholmi
    3. Chyba pixelov je najväčšia z týchto vzdialeností

Ak je chyba väčšia ako určitá hodnota, je nutné vykresliť aktuálny LOD model do novej textúry. Uzol Impostor je navrhnutý tak, že staré snímky nezahadzuje ale uchováva si ich. Pri nutnej zmene textúry sa najprv pozrie, či vhodnú textúru nevykreslil už v minulosti. Ak áno, tak ju použije a nevykresľuje novú.

Tento uzol je veľmi nápomocný pri scénach, kde sa modely častejšie opakujú a zdieľajú rovnakú geometriu s textúrami. Napríklad radové zástavby v mestách. Tu je použitie Impostoru veľmi vhodné. Ak by však Impostor bol použitý pre každý rozdielny model samostatne, hrozilo by neúnosné narastanie pamäti.

3.2.6 Trieda Simplifier

Simplifier nie je uzol v grafe ako tomu bolo pri predchádzajúcich kapitolách. Knižnica OpenSceneGraph je navrhnutá podľa množstva návrhových vzorov a jeden z nich je aj vzor vizitor. Tento vzor je použitý v triede Simplifier.

Vizitor

Ako už bolo povedané, grafom scény sa uskutočňujú takzvané priechody. Tieto priechody sú sprevádzané práve týmito vizitormi. Niektoré priechody sú pravidelné a iné sú iba na podnet programátora. Každý vizitor je uspôsobený na nejakú konkrétnu úlohu, ktorú má vykonať v uzloch pri ich „návšteve“ (preto je vzor nazvaný vizitor). Doposiaľ bol spomínaný priechod grafom, ktorý mal na starosti vykresľovanie. Simplifier je vizitor, ktorý na podnet programátora má schopnosť simplifikovať geometriu modelu. Postupným priechodom grafom doputuje až k uzlom Geode a simplifikuje každý Drawable.

Operátor a metrika

Simplifier obsahuje jeden operátor a jednu metriku. Operátor je ten najznámejší a najpoužívanejší a to zjednotenie hrany. Konkrétne je implementovaná forma full-edge collapse. Je nutné zmieniť, že autor tejto triedy, Robert Osfield, implementoval do tejto triedy aj možnosť subdivide, ale táto práca sa ňou

nezaoberá. Autor pri implementácii metriky vychádzal z viacerých vedeckých článkov, preto nie je možné sa odkázať na jeden konkrétny [8]. Zameranie metriky je však na simplifikáciu terénov [8].

Metrika je založená na ohodnotení hrany za pomoci nového vrcholu po zjednotení. Podobne tomu bolo aj pri metrike Quadric Error Metric. Nový vrchol sa v tomto prípade nachádza v strede zjednocovanej hrany. Následne sa pripraví zoznam trojuholníkov, ktoré susedia s touto hranou. To znamená, že do zoznamu sa pridá každý trojuholník, ktorý obsahuje aspoň jeden vrchol zo zjednocovanej hrany. Pre každý trojuholník zo zoznamu sa vypočíta vzdialenosť tohto trojuholníka od nového vrcholu. Výsledné ohodnotenie hrany je priemer týchto vzdialeností.

Možnosti Simplifieru

Simplifier pri zjednocovaní uvažuje možnosť non-manifold modelov. Preto hraničnо-susedné a hraničné hrany ohodnotí s nekonečne veľkou chybou. Následne ak po zjednotení hrany by pri niektorom trojuholníku vznikol problém otočenia normály, tak rovnako túto hranu nezjednocuje a ohodnotí ju nekonečnou chybou. Simplifieru sa dá nastaviť aj množina vertexov, ktoré sú určené ako nezmazateľné. To znamená, že hrany obsahujúci aspoň jeden vertex z tejto množiny nie je možné zjednotiť. Ďalšie podstatné nastavenia sú koľko % polygónov má byť vymazaných a je možné nastaviť aj maximálnu chybu hrany. Táto chyba je reálne číslo a všetky hrany, ktoré budú mať väčšie ohodnotenie nebudú zjednocované.

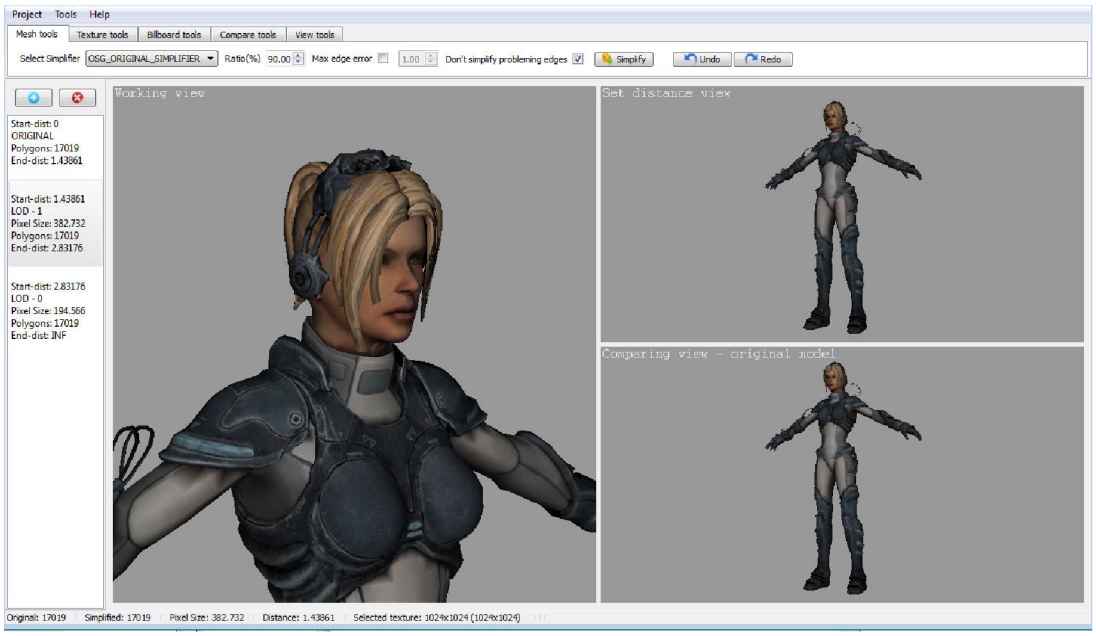
```
//Jednoduchá demonštrácia Simplifieru
osgUtil::Simplifier *simplifier = new osgUtil::Simplifier;
//Percento simplifikácie je udané od 0 do 1
simplifier->setSampleRatio( 0.5f );
//Maximálna chyba hrany
simplifier->setMaximumError( 0.1f );
//Poslanie simplifieru na prechod grafom modelu
model->accept( &simplifier );
```

4 Návrh a implementácia

Účelom tejto práce je implementovať konverznú utilitu na tvorbu modelov s úrovňami detailov a demonštračnú aplikáciu pre technológiu PagedLOD. Vývoj prebehol pod operačným systémom Windows 7 a bolo použité vývojové prostredie Visual Studio 2010 s jazykom C++. Verzia OSG knižnice je 3.3.1 a je modifikovaná firmou Cadwork.

4.1 Aplikácie Exportér

Knižnica OpenSceneGraph obsahuje množstvo nástrojov a externých programov, avšak nástroj pre tvorbu LOD modelov v nej chýba. Aplikácia Exportér vznikla za účelom odstránenia tohto nedostatku. Exportér je navrhnutý s dbaním na komfort pre užívateľa a preto používa prehľadné GUI spravené v knižnici Qt 4.8.



Obrázok 4.1: Ukážka rozrobeného projektu v aplikácii Exportér

Exportér disponuje radou užitočných nástrojov na tvorbu a obsluhu LOD modelov. Mimo tieto funkcie sa snaží poskytnúť pre užívateľa radu ďalších, ktoré slúžia na analýzu modelov alebo len poskytujú komfortnejšie rozhranie.

4.1.1 Projekt a jeho export

Prácu na modely v programe Exportér zapuzdruje abstrakcia menom Projekt. Je možné vytvoriť nový projekt, uložiť ho alebo znova otvoriť. Nový projekt sa vytvára pomocou selektovania originálneho modelu zo súboru, ktorý bude rázom načítaný a použitý ako najvyššia úroveň detailu.

Následne sa použijú funkcie Exportéru (tomu budú venované ďalšie kapitoly) a po vytvorení všetkých detailných úrovní je možný export projektu. Export je najlepší pre OSG formáty, lebo pri

týchto formátoch je možné zachovať v súboroch aj kritéria pre zmenu úrovne detailu, ako sú vzdialenosť alebo veľkosť v pixeloch. Pre tento účel je používaný súbor s koncovkou osgt. Týmto je možný export s výberom medzi viacerými druhmi koreňových uzlov. Je možné exportovať projekt ako LOD, PagedLOD alebo Impostor, pričom do všetkých týchto exportov sa ukladajú aj informácie o kritériách (osgt súborový formát). Pri každom exporte vzniká pri súbore zložka s textúrami o všetkých potrebných veľkostiach.

Po exporte do LOD formátu vznikne jeden hlavný súbor, v ktorom sú uložené všetky úrovne modelov. Podobným spôsobom je implementované aj exportovanie ako Impostor s tým rozdielom, že Impostor je možné exportovať len so vzdialenostným kritériom. Je to implementované kvôli tomu, že pre uzol Impostor sa hranica zobrazovania Billboardu určuje ako vzdialenosť a nie ako veľkosť v pixeloch. Táto hranica je pri exporte automaticky nastavená podľa vzdialenosti najnižšej úrovne detailu.

Pri použití funkcie na export do PagedLOD formátu, je vytvorené množstvo súborov a jeden hlavný súbor. Hlavný súbor obsahuje informácie o kritériách a zároveň aj najnižšiu úroveň detailu. Ďalšie súbory reprezentujú ostatné úrovne detailu. Ak by najnižšia úroveň modelu mala zvlášť súbor a nebola by v hlavnom súbore, tak by sa zbytočne robilo jedno čítanie z disku naviac. Pri použití modelu by totiž nebola žiadna úroveň k dispozícii a mohlo by sa stať, že by užívateľ chvíľku videl prázdnu scénu a až potom by sa mu zobrazovali jednotlivé úrovne. Takto po použití modelu je zaručené, že bude načítaná aspoň najnižšia úroveň modelu.

Predchádzajúce exporty sú určené výhradne pre OSG aplikácie, pretože v aplikáciách nepoužívajúce OSG nie je zaručená automatická funkcionalita zmeny úrovne modelov. Ak exportovaný model je otvorený v OSG aplikácii, a tá nemá explicitne zakázané nejakým spôsobom zmeny úrovne detailu, tak by mal model automaticky prepínať úrovne podľa nastavení v Exportéru. Pre užívateľov, ktorí nepoužívajú OSG a potrebujú modely do iných svojich aplikácií, Exportér disponuje funkciou pre export ako množstvo OBJ súborov. Opäť platí, že jeden súbor reprezentuje jednu úroveň modelu. Spolu s OBJ súbormi je vygenerovaný jeden TXT súbor s informáciami o kritériách.

4.1.2 Tvorba úrovne detailu

Po vytvorení nového projektu je načítaný originálny model, ktorý slúži ako model najväčšej kvality. Vzápätí sa automaticky vytvorila jedna úroveň detailu, ktorá je však zatiaľ zhodná ako originálny model. Pre prehľad a základné nástroje na editáciu jednotlivých úrovní slúži postranný panel vľavo v aplikácii. Kliknutím na LOD v zozname sa úroveň stáva selektovanou. Nad zoznamom sú dve tlačidlá slúžiace na pridanie a zmazanie selektovaného LODu. Pri pridávaní novej úrovne vzniká nový záznam v zozname hneď za práve selektovaným LODom. Kvalita a nastavenia novej úrovne sa skopírujú zo selektovaného LODu. Zoznam začína originálom a nasledujúce LODy sú vždy číslované od najvyššieho po najnižší.

Najväčšiu plochu aplikácie tvoria tri pohľady, v ktorých nastavenie kamery sa ovláda štandardne myškou. Prvý najväčší pohľad slúži ako pracovný pohľad, kde je detailne zobrazený selektovaný LOD. Menší pohľad vpravo hore vždy obsahuje simplifikovaný model daného LODu a je určený na nastavovanie kritéria pre tento LOD. Ak v tomto pohľade bude model vzďaľovaný od pozorovateľa, automaticky sa podľa neho nastaví kritérium vzdialenosti LODu a aj veľkosť v pixeloch. Inú formu nastavenia kritérií Exportér zatiaľ neobsahuje. Takáto forma nastavovania bola zvolená z dôvodu pohodlnosti pre užívateľa a uprednostňovanie vizuálneho nastavovania pred manuálnym číselným nastavovaním. V pravej spodnej časti je možné vidieť posledný pohľad. Tento pohľad ako jediný nie je možné ovládať manuálne. Pohľad obsahuje vždy originálny model na rovnakej pozícii a rovnako natočený ako simplifikovaný model v pohľade nad ním. Jeho úloha je

poskytnúť užívateľovi možnosť vizuálneho porovnania medzi simplifikovaným modelom a originálom na presne tej istej pozícii a rovnakej vzdialenosti.

Editácia geometrie a textúr

V záložke *Mesh tools* je možné nájsť nástroje pre zjednodušovanie selektovanej úrovne detailu. Nachádzajú sa tu nastavenia pre simplifikátor a operácie *Undo* a *Redo*. Po každom zjednodušení sa ukladá história pre prípad, že by užívateľ chcel vykonanú operáciu zrušiť alebo znova vykonať. Táto história je obsluhovaná práve funkciami *Undo* a *Redo* a každý LOD má svoju vlastnú históriu. Nastavenia Simplifieru sú dokopy 4. Prvé je výber simplifikačnej metriky. Nasleduje voľba *Ratio* udávajúca počet percent polygónov, ktoré má model obsahovať po simplifikácii. Po zaškrknutí voľby *Max edge error* sa zapne možnosť zadať maximálne ohodnotenie hrán, ktoré ešte môžu byť simplifikované. Posledné nastavenie aktivuje alebo deaktivuje ignorovanie problémových hrán pri simplifikácii. Jedná sa o hrany hraničné, hranično-susedné alebo hrany, ktoré po odstránení spôsobia otočenie normály niektorému z priľahlých trojuholníkov. Simplifikátor podľa nastavení je aplikovaný na model za pomoci tlačidla *Simplify*.

V rámci tejto práce boli implementované z predchádzajúceho textu všetky metriky zjednotenia hrany. Ako bolo zmienené, algoritmus zjednotenia hrany je najpopulárnejší a obsahuje ho aj Simplifier, ktorý sa nachádza v OSG knižnici. Preto boli uvedené metriky doimplementované priamo do OSG Simplifieru. Počas implementácie sa kládol veľký dôraz, aby nedošlo k zmene vstávajúcej funkcionality Simplifieru. Programátor môže Simplifier používať presne podľa oficiálnych návodov a iba ak chce, môže používať nové metriky a nové nastavenia, ktoré boli pridané. Trieda je pre jednoduchšie testovanie zatiaľ iba v aplikácii Exportér ale je pripravená na vloženie do OSG knižnice. Autor tejto práce dúfa, že tieto zmeny Simplifiera sa zapracujú aj do oficiálnej verzie OSG.

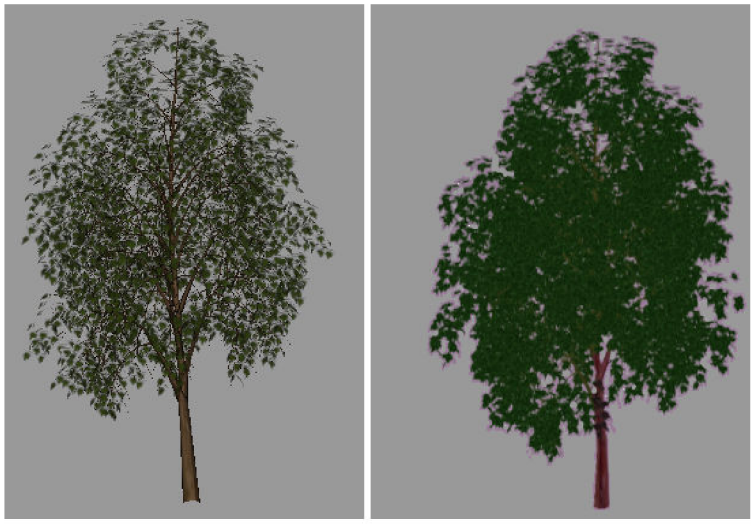
Nastavenie pre Simplifikáciu teda obsahuje metriku pôvodnú z OSG, ďalej *QEM*, *Stan Melax,* odstránenie najkratšej hrany a odstránenie najkratšej hrany s normálovým rozšírením. Okrem uvedených, je dorobená aj jedna zaujímavá funkcia, a to odstránenie všetkých hrán, ktoré sú kratšie ako jeden pixel. Táto „metrika“ nemohla byť pochopiteľne implementovaná v rámci triedy Simplifier, lebo k svojej práci potrebuje aj pohľad kamery a nejakým spôsobom zmerať pixel. Preto táto funkcia je úzko spätá s Exportérom. Funkcia pri bežnom počítaní veľkosti modelu v pixeloch zároveň zmeria aj priestorovú veľkosť jedného pixelu vo vzdialenosti, kde sa model nachádza. Pri použití funkcie, je táto hodnota prečítaná a podľa nej je nastavená maximálna chyba hrany pre metriku najkratšej hrany (maximálna dĺžka hrany). V zapätí je pustený Simplifier s touto metrikou a uvedenou maximálnou chybou. *Ratio* je nastavené na čo najmenšie, lebo Simplifier sa zastaví automaticky pri zjednotení všetkých hrán menších ako veľkosť pixela.

Nástroj na editáciu textúr je možné nájsť v záložke *Texture tools*. Po načítaní originálneho modelu, je celý model prehľadaný a nájdu sa všetky textúry. Tieto textúry je možné pre jednotlivé úrovne zmenšovať. Stačí v nástroji vybrať textúru podľa názvu súboru, zvoliť výsledný rozmer odvodený od originálnej textúry a stlačením tlačidla *Resize* sa textúra zmenší. Efekt zmenšenia je len v rámci selektovaného LODu.

Obrázok 4.2: Ukážka výsledku simplifikacie geometrie a aj zmenšenie textúry. Vľavo je 80 000 polygónový model a vedľa je jeho približne 1 000 polygónová redukcia so zmenšenou textúrou.

Billboard

Zo selektovanej úrovne detailu je možné spraviť Billboard reprezentáciu. Exportér obsahuje množstvo nástrojov v záložke *Billboard tools* na nastavenie tejto technológie. Je možné prestaviť selektovaný LOD na Billboard pomocou tlačidla *Set as Billboard* alebo vrátiť nastavenie späť na geometrickú reprezentáciu funkciou *Unset Billboard*. Po nastavení úrovne detailu ako Billboard sa zjaví jeho klasická podoba a to malý štvorec. Billboardu v tomto stave je potrebné určiť textúru, ktorá ho bude vypĺňať. Na výber textúry zo súboru slúži funkcia *Load Image*. Textúru je následne možné doupravovať, aby potrebne sedela funkciami *Rotate* a *Flip*, ktoré textúru otočia alebo preklopia. Takto správne otextúrovaný Billboard je pravdepodobne nutné zväčšiť alebo zmenšiť na potrebnú veľkosť, aby sa rozmerovo podobal originálu. Na to poslúži funkcia *Scale Billboard* s nastaveniami *Width* a *Height factor*. Po použití funkcie je Billboardu zmenená veľkosť tak, že šírka je vynásobená hodnotou *Width* faktorom a výška *Height* faktorom. Niekedy je potrebné šírku a výšku dať do pôvodného stavu. Exportér má na tento úkon funkciu *Reset scale*. Posledný dôležitý parameter pre Billboard je jeho mód, ktorý je v Exportéry dvojaký. Prvý je mód *Point*, ktorý nastaví Billboard tak, aby vždy bol celý otočený na kameru. Druhý mód otáča Billboard rovnako k pozorovateľovi ale okolo niektorej osi. Preto je rozdelený na 3 možnosti a to podľa osi otáčania ako *AxisX, AxisY* a *AxisZ*



Obrázok 4.3: Ukážka modelu stromu a jeho Billboard reprezentácie

4.1.3 Porovnanie modelov

Pre užívateľa je potrebné aby mohol vytvorenú úroveň detailu porovnať s originálom. Na vlastné vizuálne porovnanie poslúži tomu určený pohľad *Comparing view*. Do Exportéru boli však implementované algoritmy pre automatické porovnávanie simplifikovaného modelu a jeho originálu. Tieto funkcie je možné nájsť v záložke *Compare Tools*.

Obrazové porovnanie

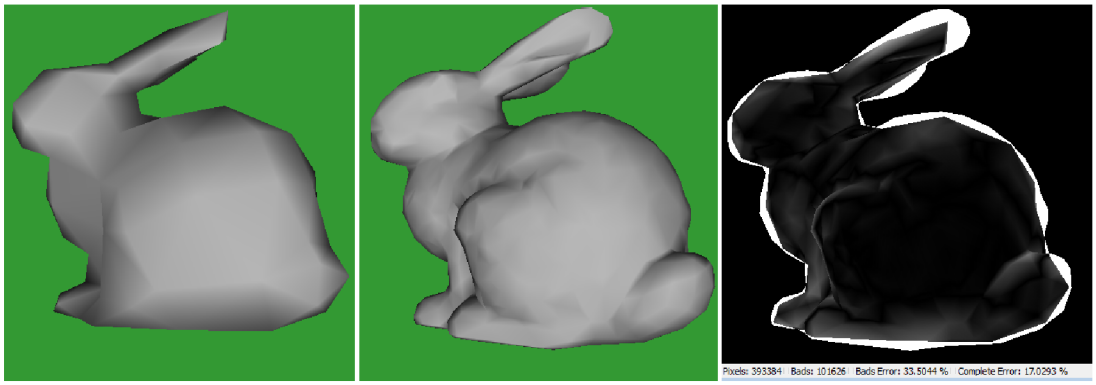
Obrazové porovnanie je postavené na porovnaní dvoch vyrenderovaných obrázkov. V pohľadu na nastavovanie vzdialenosti sa vykreslí simplifikovaný model a následne aj originálny model na tej istej pozícii. Tieto dve snímky sa navzájom porovnajú podľa známej metódy RMSE[3] so vzorcom:

$$err\ (A,B) = \sqrt{\frac{1}{w * h}(\sum_{i=1}^{w}\sum_{j=1}^{h}(A_{ij} - B_{ij})^2)}$$

Kde A a B sú porovnávané snímky prevedené do ich jasových zložiek podľa klasického vzorca:

$$I = 0.2126 * R + 0.7152 * G + 0.0722 * B$$

Pre užívateľa je výsledok porovnania okno s čierno-bielym obrázkom, kde sú zobrazené chybné pixely. Veľkosť chyby v jednom pixely je znázornený jeho intenzitou. Okno ďalej obsahuje lištu statusov, kde sa nachádzajú namerané hodnoty. Obsahuje celkový počet pixelov a koľko z toho pixelov obsahovalo chybu. Následne je udaná percentová chyba v rámci iba chybných pixelov (*bads error*) alebo v rámci celého obrázka (*complete error*).



Obrázok 4.4: Zľava: Simplifikovaný model, jeho originál a výsledok obrazového porovnania. Čím je farba viac biela, tým je v danom pixely väčšia chyba.

Je potrebné si uvedomiť, že čím je vzdialenosť modelu od pozorovateľa väčšia, tak tým je chyba menej vidno a teda aj táto funkcia bude udávať menšie chybové hodnoty. Funkcia je pre užívateľa nápomocná hlavne vtedy, keď určuje vzdialenosť úrovne detailu. Praktické použitie je aj na zvýraznenie problémových častí modelu.

[3] RMSE – Root Mean Square Error

Hranové porovnanie

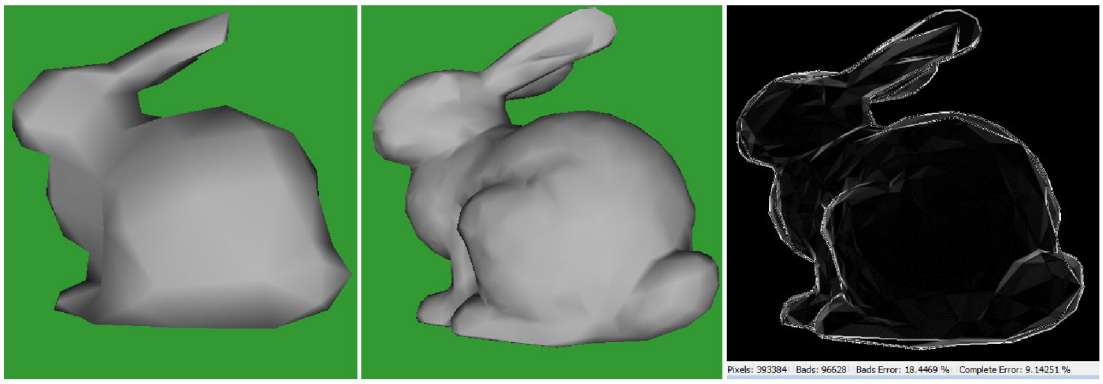
Ľudské vnímanie je veľmi citlivé na hrany. Po simplifikácii sa môže stať, že hrany modelu budú pozmenené. Preto do Exportéru bola implementovaná funkcia na vizuálne porovnanie hrán. Nástroj funguje podobným spôsobom ako *Obrazové porovnanie* len s tým rozdielom, že pred porovnaním obrazov prebehne v oboch detekcia hrán. Detekcia hrán v obraze je spravená pomocou základného Sobelovho algoritmu. Ten používa konvolúciu s maticou 3x3 ako jadro. Detekcia prebehne v dvoch etapách a to najprv po ose X a potom po ose Y. Použité sú následné matice:

$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Po vytvorení detekcií v oboch osách bude vytvorený obraz s výslednými detekovanými hranami za použitia vzorca:

$$E = \sqrt{{E_x}^2 + {E_y}^2}$$

Kde E reprezentuje výsledný obraz so zvýraznenými hranami. E_x a E_y reprezentujú obrazy vzniknuté za pomoci vyššie spomínanej konvolúcie. Takýmto spôsobom vzniknú dva obrazy s detekovanými hranami z vyrenderovaných obrazov zo simplifikovaného a originálneho modelu. Na záver nad týmito dvoma obrazmi prebehne *Obrazové porovnanie*.



Obrázok 4.5: Zľava: Simplifikovaný model, jeho originál a výsledok hranového porovnania. Čím je farba viac biela, tým je v danom pixely väčšia chyba medzi hranami.

Objemové porovnanie

Pre užívateľa môže byť prínosné vedieť porovnať objem dvoch modelov. Funkcia *Capacity compare* ponúka výpočet percenta odchýlky objemov medzi simplifikovaným a originálnym modelom. Aby funkcia správne vypočítala objemy modelov, je nutné aby modely boli uzavreté.

Algoritmus funguje totiž podobne ako tečúca voda. Model je umiestnený do virtuálneho kvádra, ktorý je rozdelený na veľké množstvo bodov. Zistiť, či sa bod nachádza v modely alebo mimo neho, nie je také jednoduché. Preto algoritmus funguje spôsobom hľadania cesty medzi uvedenými bodmi. Začína v rohu na bode [0, 0, 0] a snaží sa pohybovať čo najďalej vo všetkých smeroch. Keď prejde cez niektorý bod, uloží si ho a neskôr bude hľadať cestu z neho. Ak algoritmus pri hľadaní

narazí na stenu modelu, tak je to preňho pokyn nepokračovať z daného bodu uvedeným smerom ďalej a začne prehľadávať iný bod, ktorý si pred tým uložil. Keď už algoritmus nemá uložený ďalší bod na prehľadávanie, tak výpočet skončil. Výsledný objem modelu je teda vypočítaný nasledovne:

$$V = \frac{C - P - \frac{P_F}{2}}{C} * V_k$$

Kde C je počet všetkých bodov v kvádre, P znamená počet prejdených bodov a P_F je počet bodov označených ako kolíznych. Takýto bod bude vysvetlený ďalej v texte. V_k reprezentuje objem celého kvádru v 3D priestore.

Pri implementácii sa muselo myslieť na spotrebu pamäti. V Exportéri sa používa kváder rozdelený na 50x50x50 bodov, čo je dokopy 125 000 bodov. Takéto množstvo bodov dosahuje rozumné výsledky (odchýlka od reálneho objemu maximálne 1%). Algoritmus si musí pamätať o každom bode v kvádre nejaké informácie a tých nie je až tak málo. Je nutné si zaznamenávať dáta, napríklad, ktorými smermi sa z bodu už išlo alebo či bod je už kompletne prehľadaný alebo sa k nemu algoritmus má ešte vrátiť. Preto tieto dáta boli čo najviac zredukované a všetky informácie o jednom bode sa zmestili do jedného bajtu. Za využitia jednotlivých bitov bolo možné zaznamenať, či sa prešlo 6 smermi, či je bod čakajúci vo fronte na prehľadávanie a napokon, či je bod kolízny. Posledný údaj hovorí o tom, že do uvedeného bodu nebolo možné sa z niektorej strany dostať a teda je považovaný za bod na povrchu modelu. Spomínaná fronta čakajúcich bodov na prehľadanie je tiež štruktúra, ktorá obsahuje veľa dát. Bod je v nej uložený podľa jeho súradníc a jeden záznam zaberá 3 bajty - jeden bajt pre každú hodnotu súradnice.

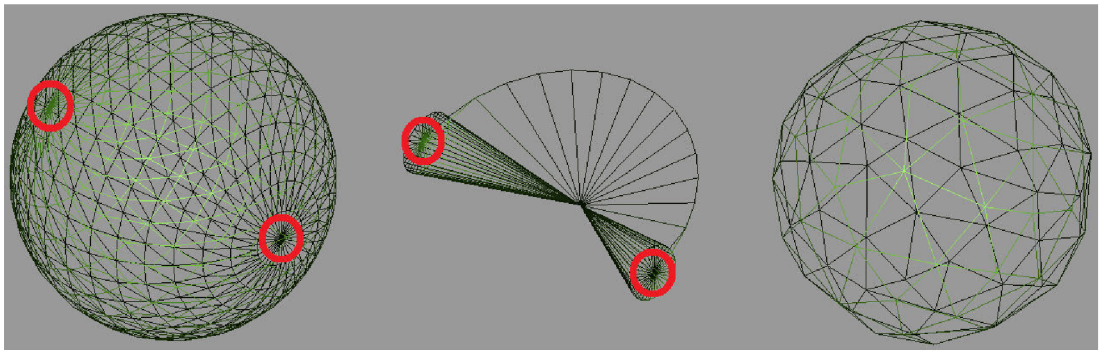
Riešenie samotnej detekcie steny modelu je implementované za pomoci OSG vizitoru *IntersectVisitor,* ktorý pri priechode grafom hľadá podľa zadaného lúča kolíziu s modelom. Pri takom veľkom množstve testovaných lúčov, ktoré sa musia vykonať v tomto algoritme, je vhodné využiť plnú funkcionalitu *IntersectVisitoru* a to možnosť zadania pre jeden prechod grafom až 32 lúčov.

4.1.4 Dodatočné funkcie

Okrem funkcií na simplifikáciu, Billboardy a porovnávanie modelov disponuje Exportér aj ďalšími užitočnými funkciami. Napríklad funkcie ako zmena farby pozadia alebo zobrazenie wireframe je možné nájsť v záložke *View tools*. Naspodku aplikácie je zobrazená lišta statusov, kde sú údaje o počtoch polygónov, aktuálnej vzdialenosti alebo veľkosti selektovanej textúry. Ďalšia užitočná funkcia má názov *Preview* a s pomocou nej si môže užívateľ prehliadnuť výslednú podobu projektu s automatickým menením úrovní detailu na základe vzdialenosti od pozorovateľa. Okrem funkcie *Preview* je možné v menu *Tools* nájsť aj ďalšie dva nástroje, a to funkcie *Repair* a *Auto Convert.*

Poškodené modely a funkcia Repair

Pri aktívnom používaní Exportéru sa v mnohých prípadoch stávalo, že model je non-manifold, pričom opticky pôsobil ako manifold. Bolo to spôsobené špatným exportom z DCC programov alebo priamo nesprávnym modelovaním pôvodného modelu. Model obsahoval totižto vertexy, ktoré boli na rovnakej pozícii ale inak boli rozdielne. Pre vizuálny vnem tieto vertexy pôsobili ako jeden jediný, avšak robili z modelu non-manifold teleso. Z tohto dôvodu sa tieto vertexy pri simplifikácii neodstraňovali, pričom opticky tomu tak nemalo byť. Kvôli tomu mohli vznikať modely, ktoré sa absolútne nepodobali pôvodným.

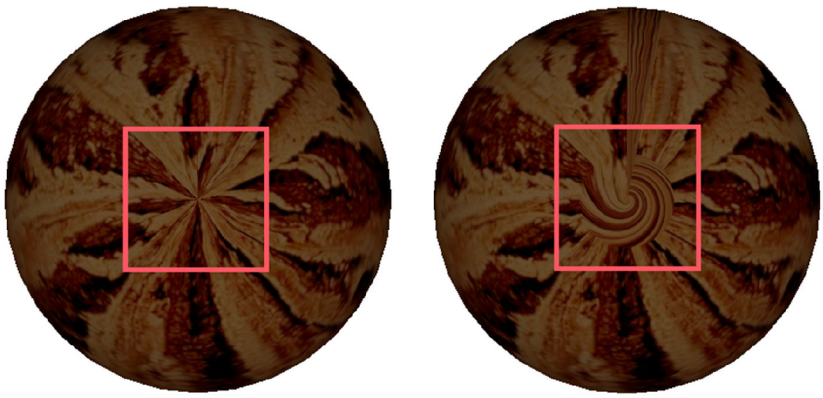


Obrázok 4.6: Ukážka nesprávnej simplifikácie (v strede), sposobenej non-manifold guľou (vľavo). Obrázok vpravo demonštruje správnu simplifikáciu po použití funkcie Repair.

Obrázok demonštruje simplifikáciu 90% polygónov z modelu guli (vľavo). Červeným krúžkom označený vrch a spodok gule nie je, ako sa môže zdať, jeden vertex. Je to zhluk 32 vrcholov na jednom mieste. Opticky je model v poriadku, avšak po zjednodušení, ako je vidno na obrázku v strede, množstvo hrán sa kvôli tomu vyhodnotí ako hraničné alebo hranično-susedné. Preto nie sú simplifikované a vzniká z guli takýto absurdný model. V prípade, že by na vrchu a na spodku gule bol iba jeden vertex, model by bol manifold a bol by korektne simplifikovaný ako je znázornené na obrázku vpravo. Preto je nutné tieto vrcholy zjednotiť do jedného jediného. Funkcia *Repair* bola do Exportéru implementovaná práve za týmto účelom.

Pre nájdenie zoskupenia vrcholov je potrebné zadať maximálnu toleranciu, kedy sa vrcholy budú považovať, že sú ešte stále na rovnakom mieste. V tejto funkcii je to vzdialenosť tisíciny jedného percenta z dĺžky polomeru bounding gule okolo modelu. Vertexy, ktoré majú medzi sebou menšiu vzdialenosť ako je táto, sú zlúčené do jedného.

Funkcia má dobrý účinok na modely, ktoré nemajú textúry. Pri otextúrovanom modely vzniká problém určenia správnych textúrovacích súradníc pri zjednotení vertexov. Je pravdepodobné, že vertexy, ktoré sú na jednom mieste majú úplne iné textúrovacie súradnice alebo dokonca aj úplne inou textúrou sú mapované.



Obrázok 4.7: Ukážka nesprávnych textúrovacích súradníc po aplikácii funkcie Repair

Automatická konverzia

Niekedy môže užívateľ chcieť vytvoriť LOD model čo najrýchlejšie a v zároveň dostatočnej kvalite. Kvôli tomu bola v Exportéry implementovaná funkcia Auto Convert. Táto funkcia zahodí všetky

doposiaľ vytvorené úrovne detailu a v zapätí za použitia Quadric Error Metric vytvorí 3 nové úrovne detailu redukované o 51%, 76% a 88%. Následne určí kritérium vzdialenosti pre každú úroveň. Uvedené čísla redukcie a výsledná vzdialenosť sú výsledkom týchto rovníc:

$$Ratio = b^{2*t}$$

$$Dist_{first} = \frac{k_1 * radius}{\cos(45°)}$$

$$Dist_t = Dist_{first} * {k_2}^t * (1 + err)$$

Kde *t* je index aktuálnej úrovne detailu a je v rozmedzí <1, 3>. Prvou rovnicou je vypočítané *Ratio* pre simplifikátor (percento simplifikácie). Ďalej *radius* je hodnota polomeru bounding gule okolo modelu. Premenná *err* je vypočítaná ako priemerné ohodnotenie zvyšných hrán metrikou QEM, ktoré ostali na modely po jeho zjednodušení (za použitia *Ratio*). Tým sa uvažuje do výpočtov vzdialenosti aj chyba simplifikácie. Konštanty *b*, k_1 a k_2 udávajú variabilitu výpočtov a v Exportéry sú nastavené nasledovne:

$$b = 0.7$$
$$k_1 = 1.5$$
$$k_2 = 2.5$$

Takto nastavené konštanty produkujú pomerne rozumné výsledky. Ale predsa len ide o automaticky vytvorené úrovne detailu a teda funkcia je s ľudským vnemom neporovnateľná. Užívateľ po automatickej konverzii môže naďalej upravovať projekt a jednotlivé úrovne detailu. Ak sa geometria niektorej úrovne nepodarila dostatočne kvalitne zjednodušiť automaticky, užívateľ má možnosť použiť funkciu *Undo* na konkrétnu úroveň a tá bude navrátená do originálnej kvality. Následne môže model zjednodušiť podľa štandardných nástrojov. Prípadne ďalšia možnosť je len upraviť vzdialenosti jednotlivým nevyhovujúcim úrovniam.

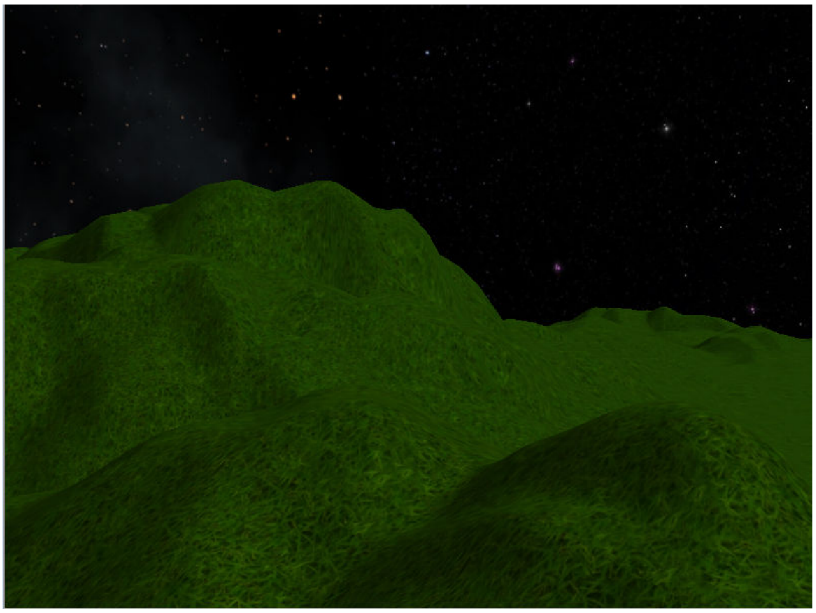
4.2 Demonštračná aplikácia

V tejto práci bolo predstavených množstvo technológií a okrem Exportéru práca disponuje aj aplikáciou, ktorá radu týchto technológií demonštruje. Všetky modely v tejto aplikácii sú skonvertované cez Exportér. Demonštračná aplikácia obsahuje tri scény a každá scéna je zameraná na ukážku niečoho iného. Jednotlivé scény sa púšťajú za pomoci parametru programu. V každej scéne je rovnaké ovládanie, a to za pomoci šípok sa užívateľ pohybuje dopredu, dozadu a do strán. S podržaním klávesy *Shift* sa pohyb kamery zrýchli. Rotácia pohľadu sa uskutočňuje pohybom myši s podržaním ľavého tlačidla. Ako je štandardom v OSG aplikáciách, klávesa S a jej viacnásobné kliknutie spustí štatistiku scény a klávesa W zapne wireframe zobrazenie.

4.2.1 Prvá scéna

Táto scéna sa spúšťa parametrom *–scene1* a je zameraná na meranie načítavania dát. V scéne je možné vidieť nočnú oblohu a terén skonštruovaný zo 16 častí. Tieto časti sú modely používajúce metodiky Level of Detail. Kvôli účelom merania je scéna spravená vo viacerých variantoch, medzi

ktorými sa opäť prepína parametrom. Každá varianta je odlišná len čo sa týka reprezentácie dát. Scénu je možné pustiť vo variante s uzlami LOD alebo PagedLOD. Následne je možné určiť, či má byť aktívny MipMapping alebo nie. A posledná varianta je, či terén má mať fyzicky na disku len jednu textúru zdieľanú medzi 16 modelmi, alebo každý model má disponovať vlastnou osobitnou textúrou. Defaultné pustenie scény je s aktívnymi uzlami LOD, bez použitia MipMappingu a so zapnutou zdieľanou textúrou. Pre zmenu LOD uzlov na PagedLOD slúži parameter *-p*. Parameter *-m* zapína MipMapping a posledný parameter *-t* nastavuje osobitnú textúru každému modelu.



Obrázok 4.8: Ukážka prvej scény

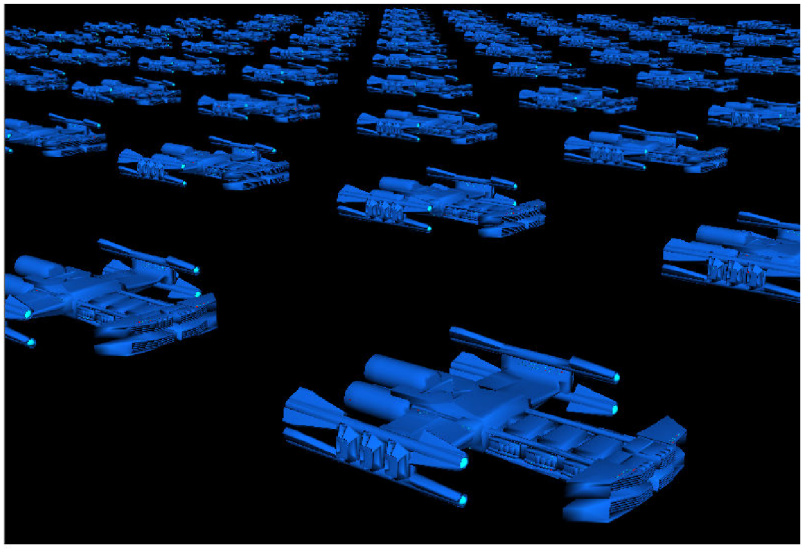
4.2.2 Druhá scéna

Spustenie druhej scény je uskutočnené parametrom *-scene2*. Scéna je zameraná na veľké množstvo LOD modelov a je v nej vytvorená stredoveká dedinka. Scéna obsahuje opäť variantu s LOD uzlami alebo s PagedLOD uzlami a prepínanie medzi nimi je uskutočnené rovnakým spôsobom ako pri prvej scéne. V dedinke boli použité aj technológie ako je Billboard. Kompletná scéna s najvyššou kvalitou modelov je pomerne zložitá. Dokopy obsahuje cez milión polygónov. Modely sú vo väčšine stiahnuté z internetu (http://turbosquid.com) a prerobené alebo upravené v 3D Studio Max. Bohužiaľ, nie ku všetkým modelom sa dala zohnať High-poly reprezentácia a teda niektoré modely majú aj v najvyššom detaile pomerne malý počet polygónov. Niektorým modelom boli umelo navýšené počty polygónov cez algoritmy Subdivision v 3D Studio Max, ale na hranatých domoch to vizuálnu kvalitu nezlepšilo. Preto väčšina modelov s umelým navýšením nakoniec nebola použitá.

Obrázok 4.9: Ukážka druhej scény

4.2.3 Tretia scéna

Parameter pre spustenie tretej scény je *-scene3*. Táto scéna ako jediná neobsahuje PagedLOD a je zameraná čisto len na demonštráciu technológie *Impostor*. Scéna si kladie za cieľ upozorniť na najlepšie využitie tejto technológie, preto obsahuje 100 rovnakých modelov vesmírnej lodi. Dohromady má scéna cez 2 milióny polygónov bez použitia Impostorov. Tie je možné zapnúť parametrom *-i*. Scéna taktiež neobsahuje žiadnu hviezdnu oblohu aj keď ide o vesmírnu scénu. Tá bola odstránená z dôvodu značného ovplyvňovania FPS, čo pri meraní v tejto scéne je najdôležitejšia meraná veličina.



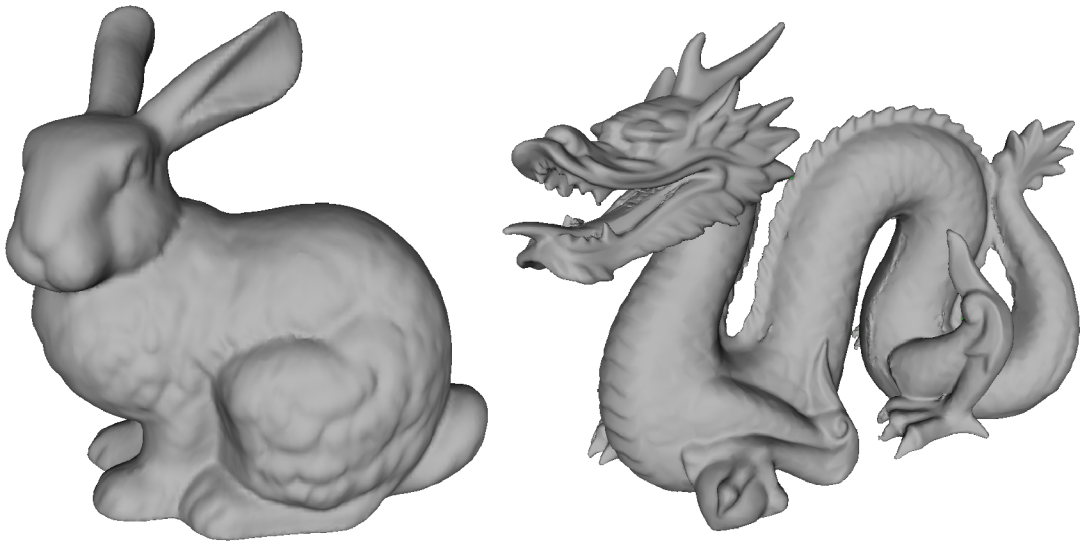
Obrázok 4.10: Ukážka tretej scény

5 Merania a výsledky

Táto kapitola si kladie za cieľ informovať čitateľa o uskutočnených meraniach a dosiahnutých výsledkoch v rámci tejto práce. Merania boli dvojitého typu. Prvý typ sa zameral na vyhodnotenie metrík v aplikácii Exportér a druhé merania sa uskutočnili v Demonštračnej aplikácii.

5.1 Porovnanie metrík

V rámci tejto práce prebehli merania výkonu a kvality implementovaných a vstávajúcich metrík na zjednotenie hrany. Všetkých metrík dohromady je 5 a to jedna natívna v OSG a 4 implementované. V ďalšom texte sú tieto metriky označované ako *OSG* pre natívnu metriku Simplifiera, *QEM* pre metriku Quadric Error Metric a následne metrika pána Stan Melaxa je označená rovnako podľa jeho mena. Metrika najkratšej hrany je označovaná ako *short* a jej rozšírenie pomocou normál je značená ako *short_normal*. Merania sú vykreslené v prehľadných grafoch a boli uskutočnené na dvoch známych modeloch zo Stanfordu. Jedná sa o modely Bunny a Dragon. Oba modely sú považované za jedny z najznámejších na univerzitných pôdach vzhľadom na testovanie. Králik je model jednoduchší s počtom polygónov 69 666 a model čínskeho draka má presne 100 000 polygónov.

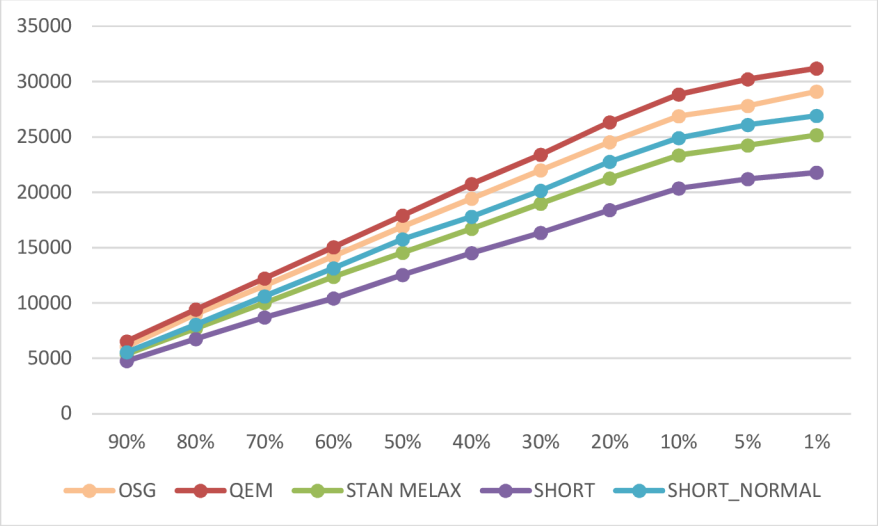


Obrázok 5.1: Modely Bunny a Dragon určené na testovanie

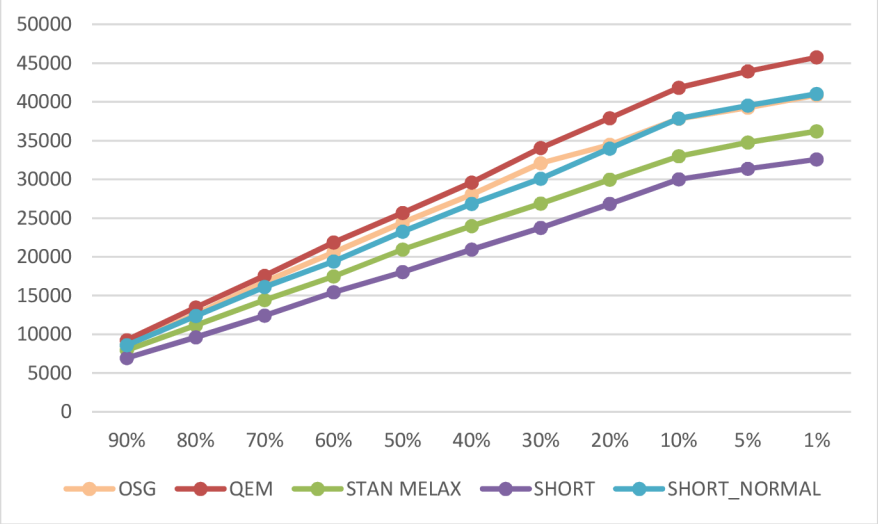
Na modeloch boli merané štyri hodnoty. Ako prvé bol meraný čas potrebný na simplifikáciu modelu do požadovanej percentuálnej úrovne. Následné tri merania boli zamerané na kvalitu vykonanej simplifikácie. Jednalo sa o už spomínané obrazové porovnanie, hranové porovnanie a objemové porovnanie. Vodorovná os všetkých nasledujúcich grafov znamená výsledné percento polygónov modelu po simplifikácii.

5.1.1 Rýchlosť metrík

V rámci výkonnosti jednotlivých metrík bol čas meraný v milisekundách a od aplikovania vizitoru na model až po jeho dokončenie priechodu. V tomto meraní sú teda započítané aj vedľajšie práce mimo simplifikácie ako je manažment priechodu grafom a podobné. Avšak časy na tieto úkony sú veľmi malé a teda nenarušujú výpovednú hodnotu merania.



Graf 5.1: Porovnanie rýchlostí metrík na modelu Bunny

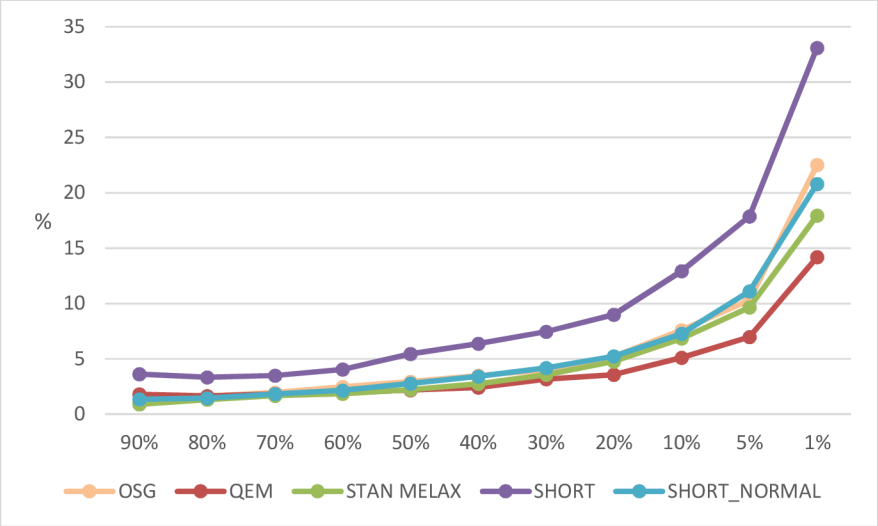


Graf 5.2: Porovnanie rýchlostí metrík na modelu Dragon

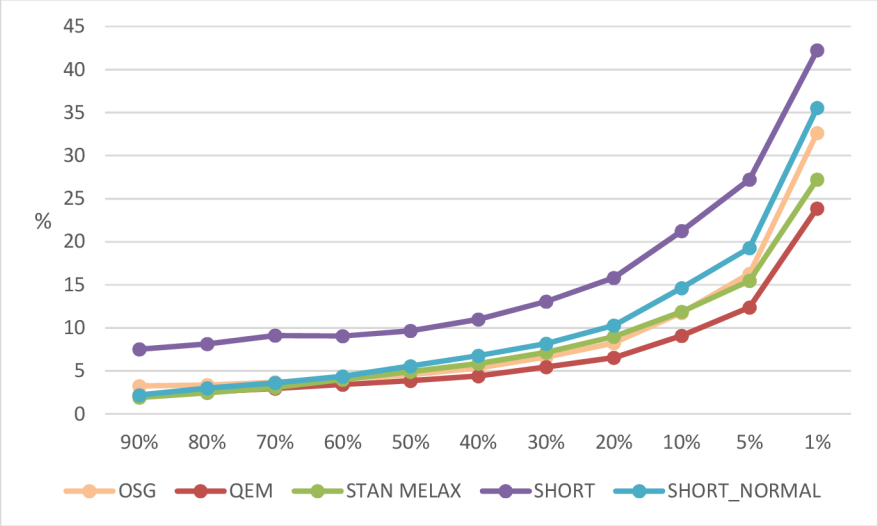
Z grafov je jasne poznať najpomalšiu metriku a to QEM. Najrýchlejšia metrika bola zjednotenie hrany podľa najkratšej dĺžky. Metriky STAN MELAX a OSG si udržujú svoje pozície v oboch grafoch rovnaké. Jedine metrika SHORT_NORMAL sa v jednom prípade viac približuje k STAN MELAX a v druhom prípade k metrike OSG.

5.1.2 Obrazové porovnanie metrík

Na porovnanie kvality je jeden z možných prostriedkov *obrazové porovnanie*. Táto metóda je hodne závislá od pohľadu užívateľa. Preto bol zvolený detailný pohľad a pre všetky testy úplne totožný. Na zvislej osi je znázornená percentuálna chyba chybných pixelov (*Bads Error*).



Graf 5.3: Obrazové porovnanie metrík na modelu Bunny

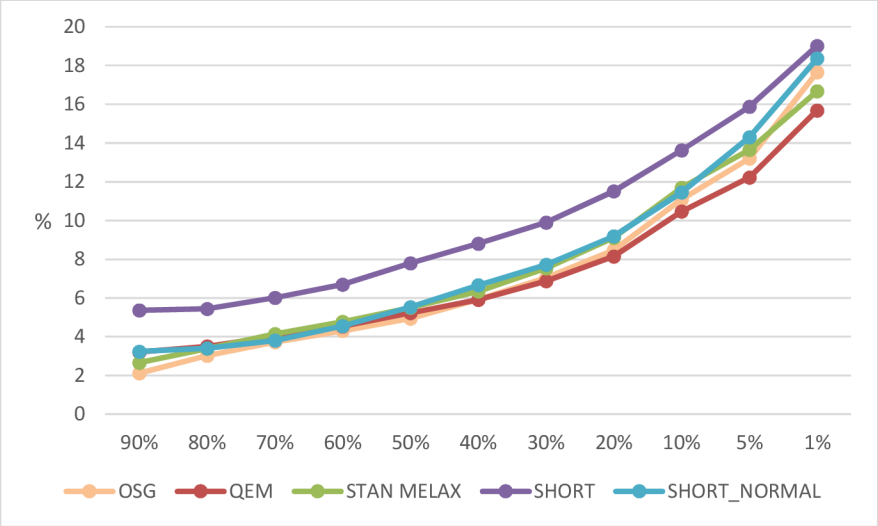


Graf 5.4: Obrazové porovnanie metrík na modelu Dragon

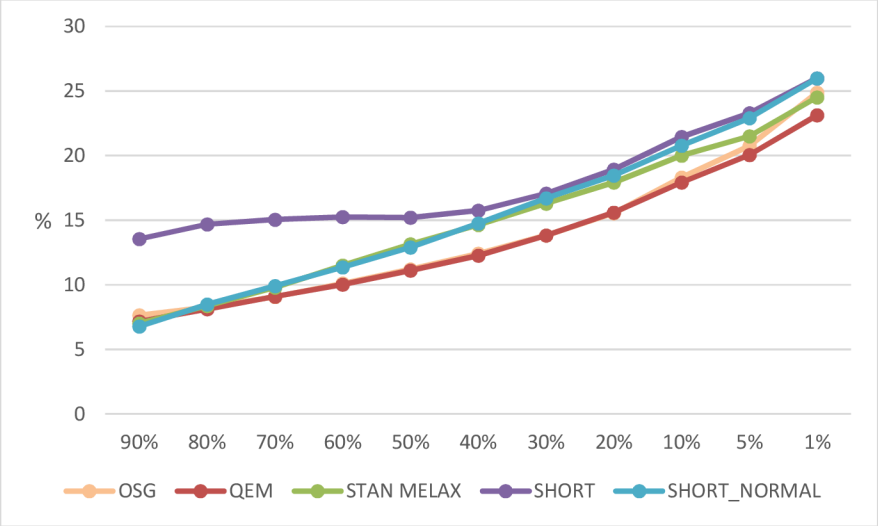
Výsledky z grafov ukazujú jasný vizuálny úpadok kvality pri metrike SHORT. A to dokonca aj pri redukcii len o 10%, kde všetky ostatné metriky dosahujú porovnateľných výsledkov. Najkvalitnejšie vizuálne výsledky produkuje QEM. Zvyšné tri metriky sú zhruba na rovnakej úrovni a rozdiely začínajú až pri redukcii veľkého množstva polygónov, kde STAN MELAX by nasledoval hneď sa QEM. Metriky OSG a SHORT_NORMAL sa opäť pomerne prelínajú.

5.1.3 Hranové porovnanie metrík

Nasledovné zrovnanie používa funkciu *Hranové porovnanie*, ktorá je podobná ako *Obrazové porovnanie*. Tak ako v predchádzajúcej kapitole je použitá hodnota percentuálnej chyby chybných pixelov (*Bads Error*) a pohľad na model je totožný ako pri predchádzajúcom meraní.



Graf 5.5: Hranové porovnanie metrík na modelu Bunny

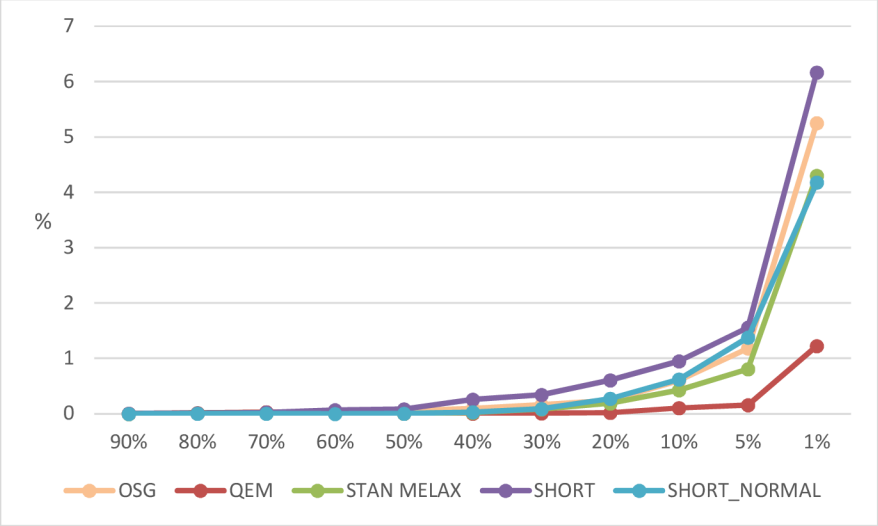


Graf 5.6: Hranové porovnanie metrík na modelu Dragon

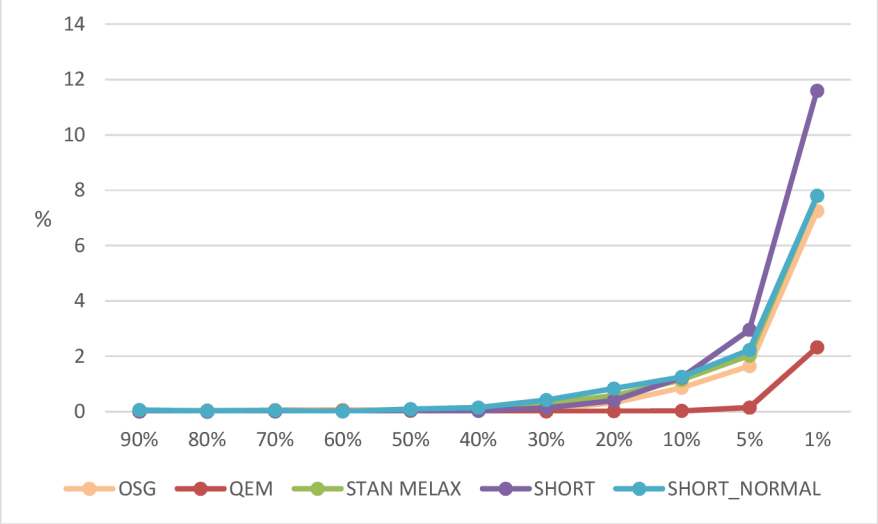
Meranie poukázalo opäť na horšiu kvalitu metriky SHORT, avšak pri meraní modelu Dragon metrika pri cca 30% pomerne slušne dobehla metriky STAN MELAX a SHORT_NORMAL. Z grafov je možné určiť najkvalitnejšiu metriku a to opäť QEM. Ale v tomto prípade má prvenstvo veľmi tesné, lebo ostatné metriky (hlavne OSG) sú v tomto meraní veľmi podobné.

5.1.4 Objemové porovnanie metrík

Posledné meranie bolo uskutočnené za pomoci funkcie *Objemové porovnanie*. Táto funkcia pri výpočtoch môže vrátiť aj záporné percentuálne hodnoty, čo však znamená, že simplifikovaný model je väčší ako jeho originál. Nestáva sa to často ale pri meraní také prípady vznikli. Boli však veľmi blízke 0 a preto grafy boli skonštruované v absolútnej odchýlke objemov.



Graf 5.7: Objemové porovnanie metrík na modelu Bunny



Graf 5.8: Objemové porovnanie metrík na modelu Dragon

V tejto kategórii testov je jednoznačný víťaz QEM so svojimi sofistikovanými výpočtami pre určenie nového vrcholu po zjednotení hrany. Naopak najväčšie objemové rozdiely boli namerané opäť pri metrike SHORT. V tomto teste ostatné metriky mali veľmi podobné chyby. Je potrebné ale zdôrazniť veľmi veľký rozdiel medzi QEM a SHORT. Najväčšie namerané chyby pri QEM sú približne 5 krát menšie ako pri metrike SHORT.

5.2 Merania v demonštračných scénach

Merania v rámci demonštračnej aplikácie prebehli vo všetkých troch scénach a sú zhrnuté v nasledujúcich dvoch tabuľkách. Prvý stĺpec v tabuľkách udáva aké parametre aplikácie boli zadané pri spustení. Podrobný popis parametrov je možné nájsť v kapitole 4.2. Ostatné stĺpce udávajú namerané hodnoty.

	Čas spustenia (s)	Pamäť po štartu (MB)	Pamäť na konci (MB)
-scene1	13.066	1005	1020
-scene1 -t	15.817	1199	1212
-scene1 -m	13.514	1014	1018
-scene1 -m -t	16.509	1190	1196
-scene1 -p	2.205	172	1016
-scene1 -p -t	2.581	174	1193
-scene1 -p -m	2.179	172	1011
-scene1 -p -m -t	2.662	174	1195

Tabuľka 5.1: Testovanie 1.scény – Čas potrebný na spustenie, obsadená pamäť po spustení a obsadená pamäť pred vypnutím aplikácie. Rozdiel medzi LOD a PagedLOD je jednoznačný.

Scéna 1 je zameraná na využitie pamäti a čas spustenia. Z Tabuľky 1 je možné pozorovať značný rozdiel času spustenia a využitia pamäti po štartu aplikácie medzi spustením v režimu LOD a PagedLOD. Sú tým pekne demonštrované prednosti PagedLOD technológie. Za povšimnutie stoja aj drobnejšie časové a pamäťové rozdiely pri spustení v móde s viacerými textúrami, kde pri pustení v móde PagedLOD sú rozdiely zanedbateľnejšie. Ako naznačujú posledné hodnoty v tabuľke, rozdiely v pamäti obom technológiám klesnú na minimum po prejdení celej scény.

Počas testovania 1. scény bol zistený problém pri použití Mipmap na modeloch. Vždy keď sa prvý krát začala vykresľovať časť grafu doposiaľ nevykreslená, OSG až v tomto momente začalo prepočítavať Mipmap levely pre použité textúry v tejto časti grafu. To spôsobilo veľmi citeľné narušenie plynulosti aplikácie. Výpočet Mipmap trval aj niekoľko sekúnd, počas ktorých aplikácia nereagovala. Pri bližšom skúmaní tohto javu bolo zistené, že výpočet Mipmap je na počítačoch so staršími grafickými kartami (testované na Mobile Intel® 4 Series Express Chipset Family integrovanej karte) vykonávaný na CPU a nie je akcelerovaný grafickou kartou. Test na modernejších kartách (Intel® HD Graphics 4000) potvrdil, že na novších kartách sa Mipmapy počítajú na grafickej karte a k takému spomaleniu aplikácie nedochádza. Treba však zmieniť, že pri PagedLOD nastáva ešte aj druhé pozastavenie aplikácie. To je spôsobené pridaním nového kúska stromu do grafu scény po načítaní úrovne zo súboru a následným vytváraním a posielaním dát do grafickej karty. Avšak toto zdržanie nie je tak značne citeľné. Následne v 2.scéne boli modely produkované, tak aby problém s Mipmapami nevznikal. Využil sa formát *dds* pre textúry, ktorý môže mať v sebe predpočítané Mipmapy a OSG ich bez problémov z tohto formátu načíta.

	Čas spustenia (s)	Pamäť po štartu (MB)	Pamäť na konci (MB)	Priemerné FPS
-scene2	22.428	1173	1425	31
-scene2 -p	3.215	227	1420	26
-scene3	0.558	12	15	5
-scene3 -i	1.121	15	72	32

Tabuľka 5.2: Testovanie posledných dvoch scén - Čas potrebný na spustenie, obsadená pamäť po spustení, obsadená pamäť pred vypnutím aplikácie a priemerné FPS počas behu

Z tabuľky je vidno opäť veľký rozdiel po štarte aplikácie medzi LOD a PagedLOD. Tabuľka však obsahuje aj priemerné FPS, ktoré pri PagedLOD je o niečo nižšie. To je spôsobené vyššie spomínaným zdržaním po načítaní úrovni z disku a jej zapojením do scény.

Scéna číslo 3 predstavovala technológiu Impostor. Z tabuľky vyplývajú aj klady ale i zápory tejto technológie. Priemerné FPS je podstatne lepšie s použitím Impostorov avšak toto zrýchlenie je na úkor väčšej spotreby pamäti. Avšak pri rozumnom nastavení Impostora je to veľmi užitočná technológia a spotreba pamäti nemusí byť kritická.

6 Záver

Zadaním práce bolo naštudovať si technológie Level of Detail v knižnici OpenSceneGraph a OpenGL. V rámci práce vznikli dve aplikácie. Jedna obsahujúca demonštračné scény pre naštudované technológie a druhá Exportér pre tvorbu LOD modelov z obyčajných modelov. Následne bolo doimplementovaných množstvo simplifikačných metrík do vstávajúcej OpenSceneGraph triedy určenej na zjednodušovanie modelov.

Z analýzy porovnávania metrík najlepšie vyšla metrika Quadric Error Metric. Aj keď jej čas potrebný na simplifikáciu je o niečo väčší ako majú ostatné metriky, nie je tento rozdiel priveľký. Vývoj scén do tejto práce však ukázal, že každá metrika má svoju dôležitosť. Obzvlášť pri vytváraní modelov do druhej scény vznikalo množstvo problémov. Modely v mnoho prípadoch bývali non-manifold alebo inak nesprávne vymodelované. Boli prípady kedy modely mali veľmi ťažko simplifikovateľné časti pre niektoré metriky. V takýchto prípadoch boli ocenené aj metriky, ktoré v uvedených testoch nedopadli najlepšie. Metrika zjednotenia najkratšej hrany sa ukázala nakoniec pre svoju jednoduchosť veľmi nápomocná a obzvlášť s kombináciou kedy sa zjednodušujú všetky hrany menšie ako pixel. Ani jedna metrika nie je úplne univerzálna a dokonalá pre všetky možné typy modelov. V zásade je možné prehlásiť, že žiadna metrika kvalitou nenahradí ľudský vnem a čím viac metrík má grafik k dispozícii, tým je väčšia pravdepodobnosť, že spraví pekne vypadajúci LOD model.

Z testovania jednotlivých scén vyplýva, že PagedLOD je výborný na zobrazenie rozsiahlych scén, ktoré by sa inak celé nemohli zmestiť do pamäti. Ďalšie veľké plus je doba od štartu programu k prvému zobrazeniu scény, ktorá môže byť pre užívateľa niekedy až zanedbateľne krátka. Avšak všetko niečo stojí a v prípade PagedLOD sú tieto výhody kompenzované aj nevýhodami. Tým, že množstvo načítavania a výpočtov sa necháva až na čas kedy je model skutočne potrebný, môže dôjsť k spomaleniu aplikácie. Pre užívateľa to nemusí ale môže byť viditeľné. Ďalšia nevýhoda je riziko neskorého načítania modelu. Užívateľ dôjde blízko k objektu skôr ako sa stihne načítať model a teda uvidí veľmi podrobne nízku úroveň detailu. Tieto dva problémy by pri LOD uzloch nevznikali, ale na druhú stranu LOD uzli ani nedisponujú výhodami PagedLODu. Ani jedna technológia nie je na prosto dokonalá a opäť je nutný ľudský faktor na posúdenie kedy je správne použiť jednu alebo druhú.

Počas práce na projektu bolo vymyslených množstvo potencionálnych vylepšení. Pre príklad by bolo veľmi užitočné riešiť simplifikáciu modelov na grafickej karte. Nie je tým myslená teselácia ale paralelizácia uvedených výpočtov. Aplikácia Exportér má potencionálne využitie aj v praxi, preto by bolo vhodné Exportér podrobiť Beta-testingu a doimplementovať množstvo ďalších funkcií. Pre príklad uvediem vyriešenie textúrovacích súradníc pri metrike Quadric Error Metric alebo pri funkcii Repair. Pre reálne využitie Exportéru by mohlo byť implementované aj väčšie množstvo štandardne používaných formátov.

Práca bola prezentovaná v rámci článku publikovanom na študentskej konferencii EEICT 2014, kde bola ocenená 2. miestom v kategórii Grafika a multimédia. Exportér bol vyvinutý za spolupráce s firmou Cadwork.

Literatúra

[1] CLARK, James H. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*. 1976, vol. 19, issue 10, s. 547-554.

[2] COSMAN, A. a R. SCHUMACKER. System Strategies to Optimize CIG Image Content. *Proceedings of 1981 Image II Conference*. 1981, s. 463–480.

[3] SCHROEDER, William J., Jonathan A. ZARGE a William E. LORENSEN. Decimation of triangle meshes. *ACM SIGGRAPH Computer Graphics*. 1992, vol. 26, issue 2, s. 65-70.

[4] ROSSIGNAC, J. a P. BORREL. Multi-Resolution 3D Approximations for Rendering Complex Scenes. *The IBM Research Division*. 1992.

[5] HOPPE, H. Progressive meshes. *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH '96*. New York, New York, USA: ACM Press, 1996, s. 99-108.

[6] HOPPE, H., T. DEROSE, T. DUCHAMP, J. MCDONALD a W. STUETZLE. Mesh optimization. *Proceedings of the 20th annual conference on Computer graphics and interactive techniques - SIGGRAPH '93*. New York, New York, USA: ACM Press, 1993, s. 19-26.

[7] KLEIN, R. a J. KRÄMER. Multiresolution Representations for SurfaceMeshes. *Proceedings of Spring Conference on Computer Graphics*. 1997, s. 57–66.

[8] OSFIELD, R. Simplifier algorithm. In: *OpenSceneGraph Forum* [online]. Dostupné z: http://forum.openscenegraph.org/viewtopic.php?t=149

[9] HAMANN, B. A data reduction scheme for triangulated surfaces. *Computer Aided Geometric Design*. 1994, vol. 11, issue 2, s. 197-214.

[10] LUEBKE, D. a C. ERIKSON. View-dependent simplification of arbitrary polygonal environments. *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97*. 1997.

[11] LOW, K. a T. TAN. Model simplification using vertex-clustering. *Proceedings of the 1997 symposium on Interactive 3D graphics - SI3D '97*. 1997.

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

[13] MELAX, S. A Simple, Fast, and Effective Polygon Reduction Algorithm. *Game Developer Magazine*. November 1998, s. 44-49.

[14] WANG, R. a X. QIAN. *OpenSceneGraph 3 cookbook: over 80 recipes to show advanced 3D programming techniques with the OpenScenaGraph API*. 1st pub. Mumbai: Packt Publishing, 2012, iv, 410 s. ISBN 978-184-9516-884.

[15] WANG, R. a X. 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.

[16] WOO, M., J. NEIDER a T. DAVIS. *OpenGL programming guide: the official guide to learning OpenGL, version 1.1*. 2nd ed. Reading, Mass.: Addison Wesley, c1997, xxxviii, 650 p. ISBN 02-014-6138-2.

[17] WILLIAMS, L. Pyramidal parametrics. *ACM SIGGRAPH Computer Graphics*. 1983, vol. 17, issue 3, s. 1-11. DOI: http://dx.doi.org/10.1145/964967.801126.

[18] GARLAND, M. a P. S. HECKBERT. Surface simplification using quadric error metrics. *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97*. 1997.

[19] HOPPE, H. New quadric metric for simplifying meshes with appearance attributes. *Proceedings Visualization '99 (Cat. No.99CB37067)*. 1999.