VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

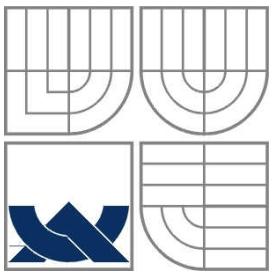
VIZUALIZACE ROZSÁHLÝCH MODELŮ

DIPLOMOVÁ PRÁCE

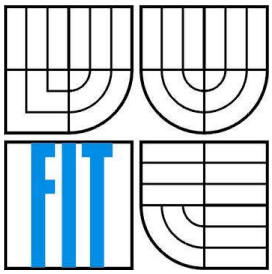
AUTOR PRÁCE
AUTHOR

Bc. PETR MOKROŠ

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VIZUALIZACE ROZSÁHLÝCH MODELŮ

VISUALISATION OF LARGE DATA SETS

DIPLOMOVÁ PRÁCE

AUTOR PRÁCE
AUTHOR

Bc. PETR MOKROŠ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAROSLAV PŘIBYL

BRNO 2009

Abstrakt

Práce je zaměřena na zobrazování rozsáhlých modelů, především modelů terénů ve vysokém rozlišení. Zabývá se návrhem a implementací aplikace, která by umožňovala dělení rozsáhlých modelů terénů na menší části a teoretický rozbor toho, jak tuto lze výsledky práce využít pro plynulé zobrazování rozsáhlých scén v reálném čase. Implementace je provedena pomocí knihovny Open Inventor.

Klíčová slova

Vizualizace, model, terén, textura, level of detail (LOD), trojúhelník, Open Inventor (OI), OpenGL.

Abstract

Work is focused on visualization of large data sets, especially on high resolution terrain models. Thus the goal of this work is to design a library which can divide large models to small regular parts for fast render of whole scene using level of detail techniques. Further goal is theoretical analysis how the library for fast and smooth visualization of large scenes in real time can be used.

Keywords

Visualization, model, terrain, texture, level of detail (LOD), triangle, Open Inventor (OI), OpenGL.

Citace

Petr Mokroš: Vizualizace rozsáhlých modelů, diplomová práce, Brno, FIT VUT v Brně, 2009

Vizualizace rozsáhlých modelů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jaroslava Přibyla. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Petr Mokroš
26. 5. 2009

Poděkování

Rád bych poděkoval panu Ing. Jaroslavu Přibylovi za vedení a podporu při tvorbě diplomové práce. Dále bych chtěl poděkovat Michaele Kocmanové, DiS. a Ing. Jiřímu Vysloužilovi za podporu a trpělivost.

Obsah

1 Úvod

V dnešní době, stejně jako před lety, mnohonásobně převyšují vize a nároky návrhářů a tvůrců digitálních modelů nebo terénů výkon běžných počítačů. Snaha o co nejdokonalejší popis, co nejvíce se blížící realitě, vede k obrovským datovým souborům a jejich zobrazování je nedílnou součástí nespočtu komerčních i nekomerčních aplikací. Ať už se jedná o herní průmysl, různé simulátory, procházky po virtuálních budovách nebo prohlížení uměleckých děl zachycených v obrovském rozlišení, tam všude je potřeba přenést data na zobrazovací zařízení. A nejen to, je také důležité mít možnost se v tomto umělém světě pohybovat, či jiným způsobem do něj zasahovat a to nejlépe v reálném čase. Většina zmíněných aplikací se snaží zaujmout co největší počet uživatelů, a to znamená najít vhodný kompromis mezi vysokou kvalitou obrazu a vysokou mírou interakce v reálném čase.

Tato práce se hledáním takového kompromisu zbývá. V druhé kapitole naznačím obecné postupy při zobrazování scén a přiblížím pojmy *Level Of Detail* (dále LOD) a řešení viditelnosti. Dále uvedu dva nejpoužívanější algoritmy na poli zobrazování rozsáhlých scén algoritmy *ROAM* a *Chunked LOD*. Závěrem kapitoly se ještě zmíním o nástroji, který budu v rámci práce používat – knihovnu Open Inventor.

Ve třetí kapitole přiblížím návrh svého řešení, popíši některé části blíže, a doplním základní postupy použité při vytváření metody umožňující dělení modelů na menší části.

V kapitole čtyři detailně provedu hlavními fázemi implementace. Seznámím s úkoly, které bylo nutné vyřešit a stavem v jakém se nachází.

Testováním aplikace a porovnání výkonu různých konfigurací programu se zabývá pátá kapitola.

S dalším možným vývojem projektu nás seznámí kapitola číslo šest. Především zde uvedu různé nápady pro možné rozšíření algoritmu a cesty, které by bylo zajímavé projít a věnovat se jim v rámci pokračování diplomové práce.

V závěru uvádím zhodnocení dosažených výsledků a zhodnocení splnění zadání.

1.1 Analýza zadání

Vizualizací rozsáhlých modelů se rozumí zobrazování modelů terénů v reálném čase. V rámci diplomové práce je mým úkolem seznámit se s algoritmy, které takové zobrazování umožňují. Dalším úkolem je navrhnout a vytvořit jednoduchou aplikaci, která umožňuje rozdělení rozsáhlé scény na menší díly. V rámci této diplomové práce se chci zaměřit na modely s nepravidelnou sítí trojúhelníků. V závěru chci vzniklou aplikaci otestovat, změřit její výkonnost a zhodnotit, zda se získané řešení hodí pro reálné použití.

2 Úvod do problematiky

V této kapitole se budeme zabývat různými teoretickými přístupy k zjednodušení zobrazování rozsáhlých scén. K základním algoritmům patří zjednodušování geometrie s rostoucí vzdáleností tzv. *Level Of Detail* a ořezání těch částí modelu, které uživatel ze svého pohledu nevidí, nebo vidět nemůže, pomocí algoritmů řešícím výpočty viditelnosti ve scéně. Tato kapitola čerpá z [1].

2.1 Renderování scén

V současné době nejsou modely skládající se z několika set milionů polygonů ničím výjimečným. Pro urychlení jejich zobrazování používáme několik metod, které můžeme rozdělit do dvou základních skupin:

- výpočty viditelnosti (viz 2.1.1),
- zjednodušování scény (viz kapitola 2.1.2 Level Of Detail (LOD)).

První skupina má za cíl rychlé určení viditelných částí scény. Pozorovatel může z místa, kde se ve scéně nachází, vidět jen část objektů. Mnoho objektů není potřeba zobrazovat, jelikož jsou mimo zorný úhel nebo proto, že jsou zakryty objekty v popředí.

Druhá skupina metod je založena na úvaze, že vzdálené objekty není potřeba zobrazovat v maximálním rozlišení, neboť mnoho detailů není při dané velikosti objektu rozlišitelných nebo je pozorovatel více zaujat objekty v popředí a vzdálenějším nevěnuje tolik pozornosti.

Při zobrazování v reálném čase je nutné soustředit se na tři základní kritéria:

- frekvence zobrazování,
- rozlišení,
- realističnost scény.

Frekvence zobrazování by měla být přibližně shodná s obnovovací frekvencí monitoru (60 - 100Hz) - vyšší rychlosti nevedou ke zkvalitnění dojmu z obrazu. Rozlišení by se mělo pohybovat kolem hodnot 1600 x 1200, které je pro většinu současných monitorů považováno za dostatečné. Co se realističnosti scény týče, tato veličina není nijak exaktní a každý uživatel ji vnímá různě. Detailnější popis modelu je jednou z hlavních částí, k realističtějšímu vjemu scény také značně přispívá kvalita simulace osvětlení ve scéně.

2.1.1 Výpočty viditelnosti

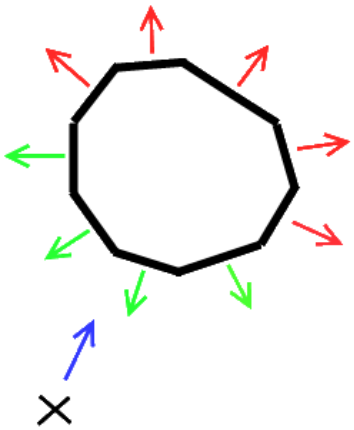
Tento druh optimalizací výpočtů nad scénou má za cíl rychlé rozhodnutí, které objekty lze z dalšího zpracování, s ohledem na jejich viditelnost, vypustit. Existuje několik kategorií těchto algoritmů, které se dělí z hlediska kvality zobrazovaného výsledku na

- přesné – poskytují exaktní řešení, jejich výsledkem je přesná množina objektů, které se mají zobrazit.
- konzervativní – metody tohoto typu tuto množinu lehce “nadhodnocují”
- agresivní – agresivní produkují pouze podmnožinu viditelných objektů
- přibližné – u těchto metod se nedá přesně specifikovat, zda výsledek je nadmnožinou nebo podmnožinou přesného řešení.

Dále se dají tyto algoritmy rozdělit podle toho, zda pracují v reálném čase či scénu předzpracovávají. U algoritmů pracujících v reálném čase se spíše uplatňují konzervativní metody, kde je stěžejním kritériem doba výpočtu. Typické se odstraňují zastíněné objekty. U algoritmů předzpracovávajících scénu není doba výpočtu tak důležitá, hlavní důraz je kladen na kvalitu výsledku.

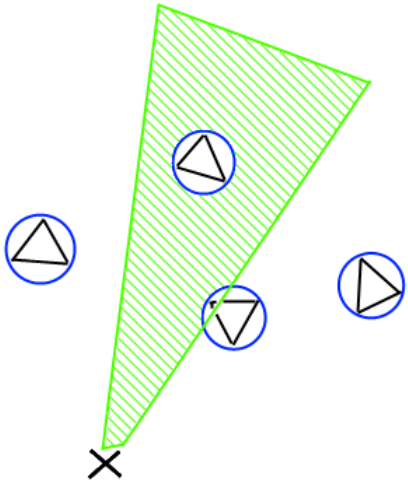
Základními technikami pro odstraňování neviditelných ploch jsou odstraňování odvracených polygonů a odstraňování objektů pohledovým jehlanem. Jedná se z hlediska předchozího dělení o metody konzervativní. Obě tyto metody jsou dnes většinou implementovány přímo v grafických procesorech. Problémem zůstává, že do grafické jednotky musíme poslat všechny polygony, které jsou následně „ořezány“ v grafickém čipu. Proto některé zobrazovací systémy provádí odstranění neviditelných polygonů ve své režii před tím, než postoupí data grafické kartě.

- **Odstraňování odvrácených polygonů**
  Odstraní v průměru 50% ploch. Princip spočívá ve shlukování polygonů na základě podobnosti směru jejich normál.



Obrázek 2.1: Odstraňování odvrácených polygonů. Pozorovatel ze své pozice může vidět pouze určité polygony (naznačeny zelenými šipkami), plochy označené červenou šipkou viditelné nejsou a budou odstraněny.

- **Odstraňování objektů pohledovým jehlanem (Viewing frustum culling)**
  Algoritmus je analogií ořezávání scény pohledovým objemem. Zde získáváme namísto ořízntého polygonu informaci, zda je polygon alespoň částečně viditelný. Pro rychlejší výpočty se používá technika hierarchie obálek, oktalové stromy, BSP stromy či kD-stromy.



Obrázek 2.2: Odstraňování objektů pohledovým jehlanem. Zelená plocha představuje pohledový jehlan, černé trojúhelníky polygony (v obálkách). Odstraněny z dalšího zpracování jsou ty, které nepadnou do pohledového jehlanu.

2.1.2 Level Of Detail (LOD)

Level Of Detail je souhrnné pojmenování pro algoritmy, které spojuje myšlenka zjednodušování scény. To se může odehrávat na úrovni sítě trojúhelníků, objektů, skupin objektů, nebo ještě větších částí scény. Některé metody zachovávají reprezentaci, jiné převádějí složitou reprezentaci do jednodušší formy např. nahrazení geometrických prvků jedním nebo několika obrázky. V překladu LOD znamená přibližně „úroveň detailu“. Úroveň detailu je vnímána tak, že v nejvyšším stupni je objekt reprezentován se všemi detaily, naopak na nejnižším stupni je co nejvíce zjednodušená varianta objektu. Stupeň detailu se zvyšuje nebo snižuje v závislosti na měnících se podmínkách, přičemž obvykle je hlavní podmínkou vzdálenost pozorovatele od objektu. Existuje i další řádka jiných přístupů – velikost průměru objektu na plochu obrazovky, celkový počet polygonů v pohledovém objemu a další. Lze také počítat s tím, že se člověk více zaměřuje na střed obrazu a směrem k okrajům se kvalita jeho vnímání snižuje. Dále je možné měřit a uvažovat chybu, kterou změna LOD ve výsledném obraze způsobí.

Při použití LOD se setkáváme s jevem, který působí jako vyskakování částí objektu (*popping effect*). Je to způsobeno tím, že při snížení či zvýšení stupně detailu zmizí, nebo se objeví, skupina trojúhelníků. Tento efekt se dá omezit pomocí obrazových technik – v předstihu vytvoříme snímek po změně LOD a poté interpolujeme mezi tímto a původním obrazem. Dá se také provádět *morfing* geometrie jednoho diskrétního stupně na druhý.

Obrázek 2.3: Různé stupně detailu, první 69 400 polygonů, druhý 2500 polygonů a třetí 251 polygonů. Převzato z [11].

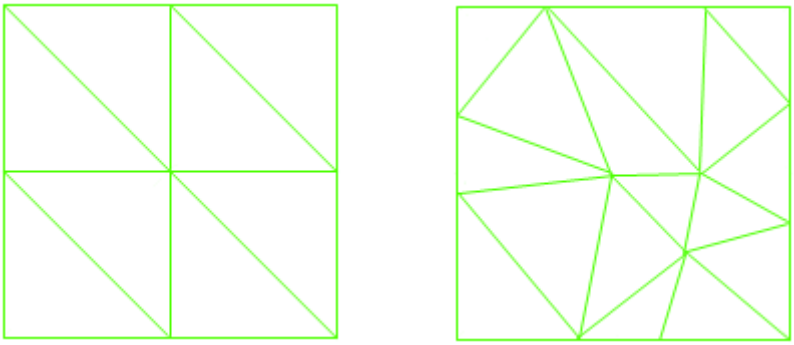
2.2 Druhy scén

Scény můžeme rozdělit podle způsobu rozložení jejich geometrie na scény s pravidelným rozložením a scény s nepravidelným uspořádáním sítě trojúhelníků.

Scény s pravidelným rozložením získáváme například zpracováním tzv. výškových map. Výšková mapa je ve své podstatě matice hodnot, kde každá hodnota nese informaci o výšce terénu v daném bodě. Výškové mapy se dají získat například přímým pozemním měřením existujících terénů nebo zpracováním satelitních snímků. Pokud nezáleží na tom, aby daný model odpovídal přesně některému reálnému místu, je možné si výškovou mapu vygenerovat. Pro umělé generování výškových map existují různé funkce proto, aby výsledná mapa působila realisticky. Často používanou je například *Perlinova šumová funkce*.

Modely s nepravidelným rozložením získáme většinou po ručním vytváření modelu, nebo převedením modelů s pravidelným rozložením. Výhodou je, že se nepravidelné sítě vyznačují paměťovou úsporou oproti pravidelným sítím, které se vyznačují vysokou redundancí. Naproti tomu se s nimi hůře manipuluje.

U dělení modelu vycházející z výškové mapy můžeme postupovat téměř libovolně a pro dělení modelu a navazování segmentů existuje několik jednoduchých principů. Při dělení nepravidelného modelu máme na výběr dvě možnosti. První je dělit podle dělící roviny a to tak, že model jakoby dělící rovinou rozřízneme. Vzniknou tedy dva modely s rovnou hranou ale s vyšším počtem trojúhelníků. Druhý způsob je, že dělící rovinu bereme jako vodítko, kde by mělo k oddělení modelu dojít, a model dělíme po hranách trojúhelníků, kterými dělící rovina prochází. Tak nám vzniknou dva modely, s nerovným okrajem, nedojde však k nárůstu počtu trojúhelníků.



Obrázek 2.4: Pravidelný a nepravidelný model

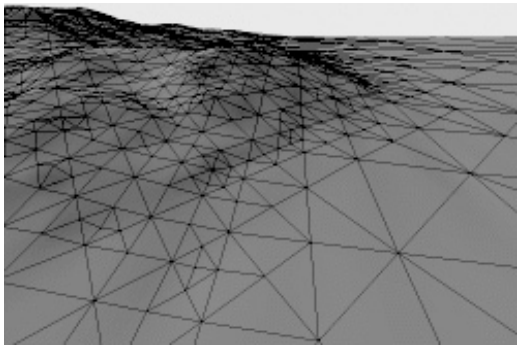
2.3 Renderování modelů terénu

Tato kapitola uvádí stručný přehled základních algoritmů, které se pro rychlé zobrazovaní modelů terénů využívají. Tyto algoritmy vznikly na základě zpracování modelů vytvořených z výškových map a pracují s modely s pravidelnou strukturou a jejich účelem je snížit počet zobrazovaných trojúhelníků.

2.3.1 ROAM

ROAM je zkratka, která znamená *Real-time Optimally Adapting Meshes*. Algoritmus používá strukturu *BTT* (*Binary Triangle Tree*), pracuje ve dvou fázích části statické a části dynamické. Ve statické části mimo jiné plní strukturu *BTT* a tzv. pole rozdílů. V poli rozdílů je uložena míra chyby pro každý bod, podle které se v dynamické fázi určuje, zda daný bod zobrazit, či nikoli (kromě krajních bodů, které zaniknout nesmí). V dynamické fázi, která už musí probíhat rychle, se pak těchto struktur využívá a na základě „rozhodovacího poměru“ se posoudí, zda daný bod bude zobrazen a jaká bude úroveň detailu v jeho okolí (např. zda se kolem něj zobrazí dva nebo čtyři trojúhelníky). Aby se zamezilo vzniku *T-junction* (T-spojení) používá se technika *Force-split*, tedy pokud bod který se má zobrazit není středem diamantu, musí se rozdělit nejprve sousední trojúhelník.

Metoda se dá ještě optimalizovat použitím ořezávání pohledovým jehlanem. Více lze nalézt v [6].

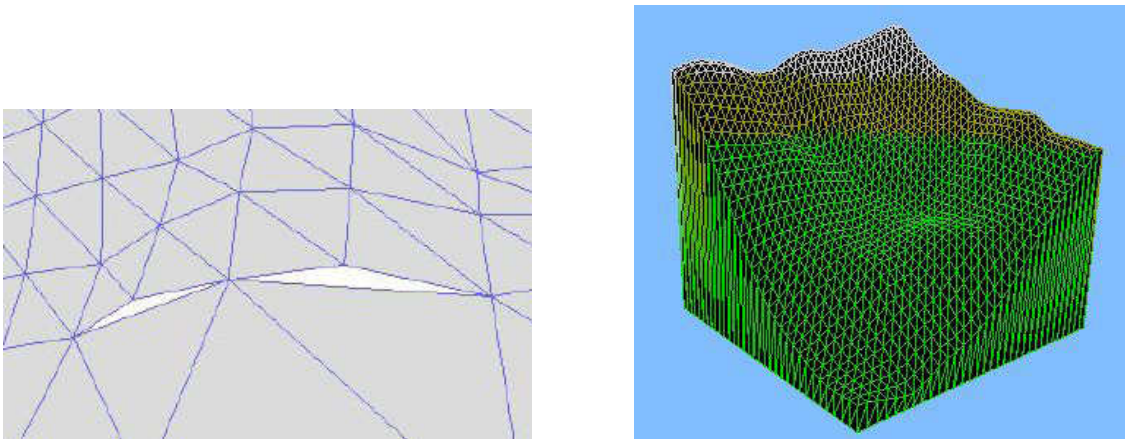


Obrázek 2.5: Ukázka výstupu algoritmu ROAM. Převzato z [6].

2.3.2 Chunked LOD

Podobně jako klasický LOD je i tento algoritmus založen na myšlence, že s rostoucí vzdáleností objektu od pozorovatele nemusí být tento objekt vykreslován v maximálním detailu. Metoda spočívá v rozdělení modelu na podsegmenty. Využívá přitom strukturu *Quad Tree*. V kořeni tohoto stromu je celá scéna v nejnižším stupni detailu (využívá pouze každou x-tou hodnotu z výškové mapy. Proměnná „x“ závisí na tom, jakou zvolíme velikost segmentu na nejdetailnější úrovni, respektive kolik kroků *Chunked LOD* nad modelem chceme provést). V dalším kroku je scéna rozdělena na 4 segmenty a ty mají úroveň detailu vyšší než předchozí krok. Každý tento segment je v dalším kroku znovu rozdělen na 4 podsegmenty s vyšší úrovní detailu. Takto se postupuje, až se model rozdělí na nejmenší segmenty s původním (nejdetailnějším) rozlišením modelu. Při zobrazování modelu se pak postupuje tak, že na nejbližší okolí pozorovatele jsou použity menší segmenty s detailnějším popisem krajiny a se vzrůstající vzdáleností od pozorovatele se použijí segmenty s menší mírou detailu.

Protože se výpočet trojúhelníkové sítě provádí pro každý krok samostatně a úroveň detailu se zvyšuje, může při zobrazování docházet k tomu, že segment s vyšším detailem přesně nenavazuje na segment s detailem nižším. Ve výsledném modelu se tak objeví díry (*cracks*). Díky nim pak má uživatel možnost jakoby nahlédnout pod krajinu. Jejich odstraňování se provádí pomocí tzv. sukýnek (*skirts*), svislých stěn na okrajích segmentů, které tyto nedostatky opticky zakryjí.



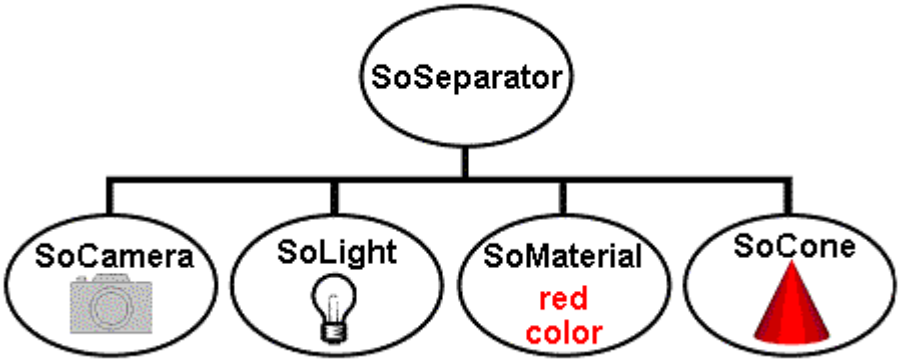
Obrázek 2.6: „Cracks“ v modelu a jejich řešení v podobě „skirts“. Převzato z [10].

2.4 Open Inventor

Tato kapitola čerpá z [2]. Open Inventor je knihovna napsaná v C++ a postavená nad OpenGL, která posunuje programátora od základního OpenGL rozhraní na vyšší úroveň a nabízí mu rozsáhlou množinu C++ tříd. Ta podstatně zjednodušuje práci programátora a dokonce často poskytuje vyšší výkon než přímá implementace v OpenGL. Vyšší výkon je možný díky jistým optimalizacím, které Open Inventor může provádět nad daty scény.

Pro svou práci jsem si vybral distribuci firmy System In Motion, knihovnu Coin3D, která je k dispozici pod GPL licencí. Kromě ní existují ještě další dvě hlavní větve Open Invetoru – komerční balík firmy TSG, která od SGI odkoupila práva na další vývoj jejich projektu a další verze přímo od firmy SGI, která v roce 2000 zveřejnila svou poslední verzi Open Inventoru 2.1 jako open-source. Tento balík je však poměrně zastaralý a to díky tomu, že ve vývoji již dále firma nepokračovala.

Design Open Inventoru vychází z konceptu grafu scény. Tedy, scéna je složena z uzlů (*nodes*). Ty mohou být různých typů. Jedny nesou informace o geometrii těles (krychle, kužel, model tělesa), další různé atributy (barva, textury, souřadnice objektu) a také existují speciální uzly, které obsahují seznam jiných uzlů, anglicky zvané *groups*. A právě tyto *groups* umožňují organizovat ostatní nody do hierarchických struktur zvaných grafy. Takovýto graf nám pak reprezentuje naši scénu. Celou problematiku můžeme vidět na obrázku 2.7.



Obrázek 2.7: Ukázka části Grafu scény v Open Inventoru. Zdroj [2].

Tato reprezentace je velice výhodná, protože lze velmi jednoduše nastavovat transformace pouze pro určitou část scény, bez ovlivnění zbytku zobrazovaného světa. Open Inventor navíc umožňuje uchovávat v uzlech definujících geometrii objektu také několik stupňů LOD.

3 Návrh systému

Rychlé zobrazování pravidelných modelů terénů v reálném čase je dnes díky algoritmům jako *ROAM* nebo *Chunked LOD* poměrně běžnou záležitostí. V rámci diplomové práce bych se proto rád zaměřil na zobrazování modelů, které jsou tvořeny nepravidelnou sítí trojúhelníků. Ani jeden z výše popsaných algoritmů s nepravidelnou sítí ve své základní podobě nepracuje. Rád bych se proto inspiroval algoritmem *Chunked LOD*.

Protože nemám výškovou mapu terénu, ale již hotový nepravidelný model, bude základním úkolem tento model rozdělit tak, aby se mohly segmenty uložit do struktury podobné *Quad Tree*. V první fázi bych model dělil na 4 stejné obdélníkové podsegmety. V místě dělení zákonitě vznikne řada nových trojúhelníků. Nad segmenty bych provedl decimaci trojúhelníkové sítě, tak aby se toto řešení podobalo původnímu algoritmu *Chunked LOD*. Tedy větší segmenty budou obsahovat méně trojúhelníků, potažmo detailů a se snižující se velikostí segmentu se bude síť trojúhelníků více přibližovat původnímu modelu.

Dalším úkolem pak bude vytvořit aplikaci, která bude umožňovat demonstrovat funkci algoritmu a měřit dosažené výsledky.

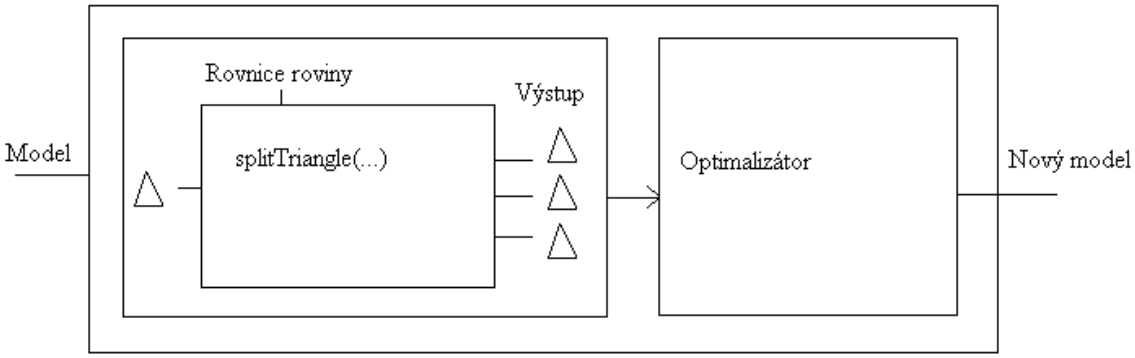
Základním motivačním prvkem celé práce byla diplomová práce Petra Vrby [1]. Autor zde také dělil model s nepravidelnou sítí trojúhelníků. Rozhodl se pro dělení modelu tak, že určí hranice, kde se má model rozdělit. Podle této hranice je poté rozhodnuto, které trojúhelníky náleží do jedné části modelu, a které do druhé. Vzniklé rozdělené modely mají tedy „zubatý“ okraj. Výhodou tohoto typu řešení je, že při dělení nevzniknou vůbec žádné trojúhelníky. Další výhodou jsou okrajové trojúhelníky, které se nezapojí do procesu snižování počtu trojúhelníků, a proto není potřeba řešit navazování jednotlivých segmentů – vždy na sebe navazují stejné trojúhelníky. Nevýhodou je právě zubatý okraj jednotlivých segmentů. Díky němu se například na tyto segmenty velice špatně mapují textury.

Proto jsem se začal zabývat myšlenkou jak tento postup vylepšit tak, aby zůstaly zachovány výše zmíněné výhody a nevýhody byly eliminovány. Základem je zachovat pravoúhlý tvar segmentů pro jejich snadnější pokrývání texturou. Pokud budu ovšem zachovávat pravoúhlý tvar, dojde zákonitě na dělení modelu a tedy vnik nových trojúhelníků. Toho jsem se chtěl ale vyvarovat. Bude tedy nutné najít způsob, kterým bych počet nových trojúhelníků snížil na minimum.

Pro zhodnocení zda je toto řešení v praxi použitelné je také potřeba zkontrolovat jak vypadá vizuální stránka projektu a jak výstup působí na uživatele. Pro zobrazování výsledku by bylo dobré vyzkoušet, jak vypadá toto řešení ve spojení s metodou LOD. Zobrazování LOD by se dalo provést tak, že zobrazím rozdělenou scénu a každý jednotlivý díl bude mít řekněme tři varianty zobrazení (úrovně detailu), které se budou přepínat podle vzdálenosti pozorovatele od daného dílu scény.

3.1 Schéma dělení modelu

Na obrázku 3.1 je znázorněné zjednodušené schéma zakomponování metody `splitTriangle()` do plánovaného projektu. Vstupem je celý model, který je do metody posílán po jednotlivých trojúhelnících. Ty metoda vyhodnotí, a podle jejich umístění v prostoru je rozdělí zadanou rovinou. Výstupy této třídy pak mohou projít jistou optimalizací.



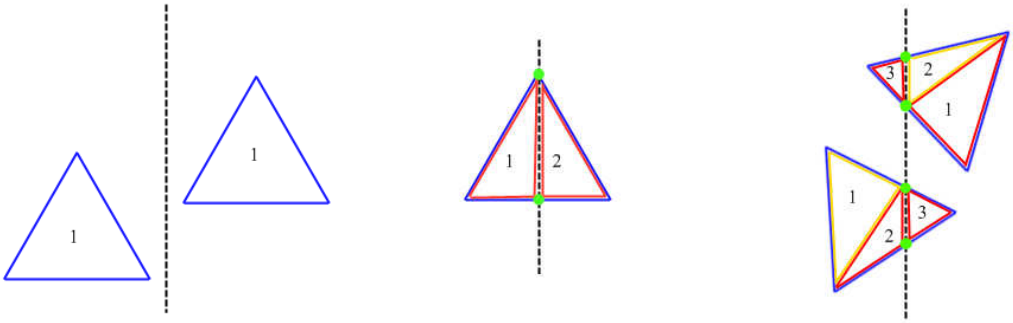
Obrázek 3.1 Zjednodušené schéma budoucího jádra projektu

Metoda bude zabudována do komponenty, která umožní dělit model na vstupu na dva modely na výstupu. Opakovaným voláním této komponenty se dosáhne potřebného nadělení celého modelu.

3.2 Dělení trojúhelníku

Ačkoli by se zdálo, že je dělení trojúhelníku v počítačové grafice rutinní záležitost, nepodařilo se mi najít v použitých knihovnách příslušnou funkci. Implementoval jsem proto své řešení.

Základem je zjistit, zda trojúhelníkem prochází dělící rovina. Toho dosáhneme tak, že si pro každý vrchol vypočítáme vzdálenost bodu od roviny. Výsledky mohou být, podle toho jaké vyjdou hodnoty, celkem tři.

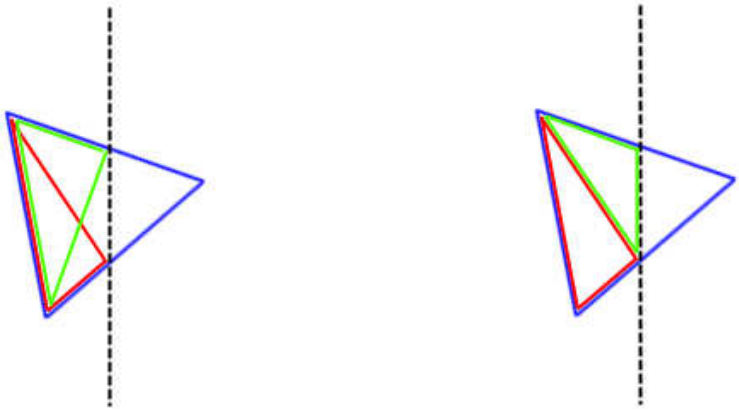


Obrázek 3.2: Varianty rozmístění trojúhelníků vůči dělící rovině a označené nově vzniklé

1. ***Trojúhelník leží celý na jedné nebo druhé straně*** – poznáme to tak, že vzdálenost všech vrcholů trojúhelníka od roviny je kladná případně záporná (nebo se jedná o variantu, kdy jeden vrchol leží přímo na řezací rovině, a zbylé dva jsou orientovány do stejného poloprostoru, nebo dva body leží přímo na dělící rovině a jeden bod je orientován na jednu nebo druhou stranu). Tento trojúhelník není potřeba dělit a jen označíme, zda patří do kladného nebo záporného (ten je určen směrovým vektorem řezací roviny) poloprostoru, abychom ho mohli posléze zařadit do správného podmodelu.

2. ***Jeden vrchol trojúhelníku leží přímo na dělící rovině a zbylé dva jsou každý na jiné straně roviny*** – v tomto případě je již nutné dělit. Vzniknou tak dva nové trojúhelníky.

3. ***Jeden vrchol je na jedné straně, zbylé dva na straně druhé*** – i v tomto případě je nutné dělit a to tak, že na straně, kde je jeden vrchol, vznikne jeden nový trojúhelník a na straně druhé vytvoříme trojúhelníky dva. Právě na této straně je nutné dát pozor, aby vznikly dva správné trojúhelníky a nedošlo k situaci na obrázku 3.3 vlevo.



Obrázek 3.3: Vlevo – špatně definované nové trojúhelníky, vpravo - správně

Při vytváření trojúhelníku je také třeba vzít v potaz orientaci jejich normály, tedy to, zda jsou jejich vrcholy definovány po směru nebo protisměru hodinových ručiček. Nedodržení stejné definice u nově vzniklých trojúhelníků pak má za následek například chybně počítané osvětlení.

3.3 Matematické pozadí

3.3.1 Trojúhelník základ modelu

Modely se skládají z grafických primitiv. Základními primitivy v počítačové grafice jsou bod, úsečka, trojúhelník a polygon. Z hlediska tvorby 3D modelů se stal nejpoužívanějším primitivem trojúhelník. Splňuje totiž zásadní podmínku – všechny jeho tři body leží vždy v jedné rovině, čímž se velmi liší například od čtyř a více – úhelníků, kde bychom tuto vlastnost museli pracně kontrolovat. Trojúhelník se díky této vlastnosti stal základním kamenem pro tvorbu modelů.

3.3.2 Průnik přímky a roviny

3.3.2.1 Parametrická rovnice přímky

Přímku v prostoru určuje tzv. směrový vektor ***u*** a bod *A* ležící na přímce. Vyjádříme-li to matematicky, obdržíme parametrické vyjádření přímky.

$$X = A + t\vec{u} \quad t \in R \qquad [3.1]$$

Kde *A* jsou souřadnice bodu, ***u*** směrového vektoru a *t* se nazývá parametr. Změnou parametru získáme různé body ležící na přímce. Parametrické vyjádření můžeme psát i v souřadnicích:

$$x = a_1 + tu_1,$$
$$y = a_2 + tu_2,$$
$$z = a_3 + tu_3, \quad t \in R. \qquad [3.2]$$

3.3.2.2 Obecná rovnice roviny

Pro určení obecné rovnice roviny musíme znát bod ležící v hledané rovině a vektor, který je kolmý na všechny vektory ležící v rovině, tzv. normálový vektor.

$$ax + by + cz + d = 0 \qquad [3.3]$$

Kde *a*, *b*, *c* jsou hodnoty souřadnic normálového vektoru.

3.3.2.3 Průnik

Pokud máme parametrickou rovnici přímky a obecnou rovnici roviny, je již určení průniku snadné. Z parametrické rovnice dosadíme hodnoty jednotlivých souřadnic do obecné rovnice roviny.

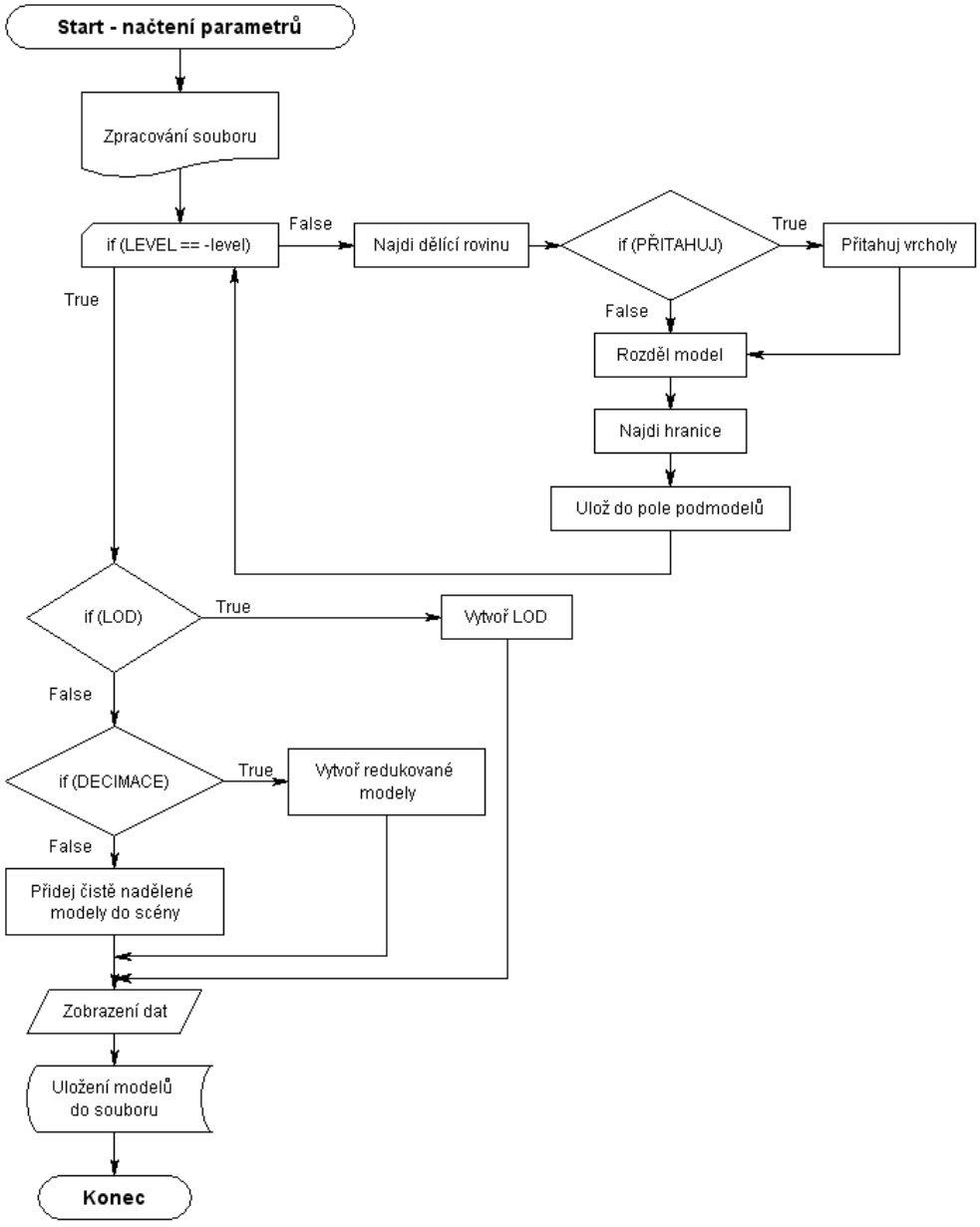
$$a(x + tu_1) + b(y + tu_2) + c(z + tu_3) + d = 0 \qquad [3.4]$$

Odtud nyní vypočítáme parametr *t*. Získaný parametr dosadíme zpět do parametrické rovnice přímky. Spočtením pravých stran jednotlivých rovnic získáme souřadnice průsečíku.

4 Implementace

4.1 Aplikace TriangleDivision

Aplikace TriangleDivision byla vytvořena jako základní nástroj pro dělení rozsáhlého modelu na menší části. Finální řešení je realizováno pomocí třídy `MyClassSubModel`. Třída obsahuje metody, které řeší veškerou logiku kolem dělení modelu. Tato třída je poté využívána v hlavní smyčce programu pro uchování jednotlivých nadělených modelů. Celý postup fungování programu je zachycen na následujícím vývojovém diagramu.



Obrázek 4.1: Základní vývojový digram aplikace TriangleDivision

Nyní si probereme jednotlivé části. Načtení modelu zajišťuje callback funkce `triangle_cb()`, která projde celou geometrii uloženou v uzlu typu `SoSeparator` a vrátí vektor bodů a vektor indexů. Dostaneme tedy přístup ke všem trojúhelníkům. Tento postup je nutný, neboť vstupní model může být složen z různých geometrických tvarů (koule, kužel, kvádr…). Budeme tak pracovat přímo s jednotlivými trojúhelníky.

Získaný vektor bodů a indexů použijeme pro vytvoření první instance třídy `MyClassSubModel`. Jedná se ve své podstatě o nultý, základní model scény v nejvyšším rozlišení a nijak nezměněné formě.

Při dělení modelu postupujeme tak, že si vytvoříme dvě nové instance třídy `MyClassSubModel`. Ty budou reprezentovat scénu o úroveň níže (tedy scénu s vyšším počtem dělení). Vypočteme souřadnice dělící roviny, pomocí které budeme model dělit. Podle toho, zda je parametrem určeno, že se má přitahovat, se zkontrolují vrcholy v pásmu přitahování a upraví se geometrie. Poté modely dělíme. Pokud chceme zobrazit ve výsledné scéně LOD, zjednodušené modely nebo pouze zobrazit hraniční trojůhelníky, je nutné ještě provést výpočet hraničních trojúhelníků metodou `findBoundaryTriangles()`. Nadělené modely uložíme do pole modelů. Zvýšíme jim index `level`, který indikuje, k jaké úrovni dělení modely patří. Podle toho, zda chceme zobrazovat LOD, zjednodušené modely, nebo pouze plný výsledek přitahování a dělení, je do kořene scény uložena odpovídající kombinace výstupu a ta je posléze zobrazena. Scéna je vykreslena do standardního okna vytvořeného knihovnou `SoWin`. Okno je navíc opatřeno ovládacími prvky, které umožňují se ve zobrazené scéně pohybovat a měnit parametry zobrazení jako je to, zda bude model zobrazován jako síť trojúhelníků nebo bude stínovaný apod. Aplikaci ukončíme zavřením okna. Nyní se vrátím k jednotlivým krokům a podrobněji je popíši.

4.2 Výpočet souřadnic dělící roviny

Výpočet souřadnic provádíme tak, že vyhledáme v původním modelu jeho nejdelší hranu. Tu rozdělíme vždy v půli. Při prvním dělení navíc zjistíme, jak je model orientován a ve které ose je umístěna výška terénu. Tyto hodnoty jsou uloženy a při dalším dělení se předávájí dalším podmodelům jako referenční. Použití takové techniky je založeno na faktu, že je program určen pro dělení rozsáhlych modelů terénů a v takových lze předpokládat, že počáteční rozměry plochy terénu mnohonásobně převyšují hodnoty vrcholků nejvyšších pohoří a nejhlubších údolí v modelu. Důvodem k takovému kroku byla chyba, která se vyskytla v průběhu imlementace. Pokud byla výška v modelu udávána například v ose „y“, byla plocha modelu, která měla být rozdělena umístěna v souřadnicích na osách „x“ a „z“. Během dělení docházelo k tomu, že se rozměry „x“ a „z“ zmenšily natolik, že se výška modelu stala nejdelší stranou modelu. Dělící rovina pak byla vypočítána chybně a model byl rozdělen v polovině výškového rozdílu.

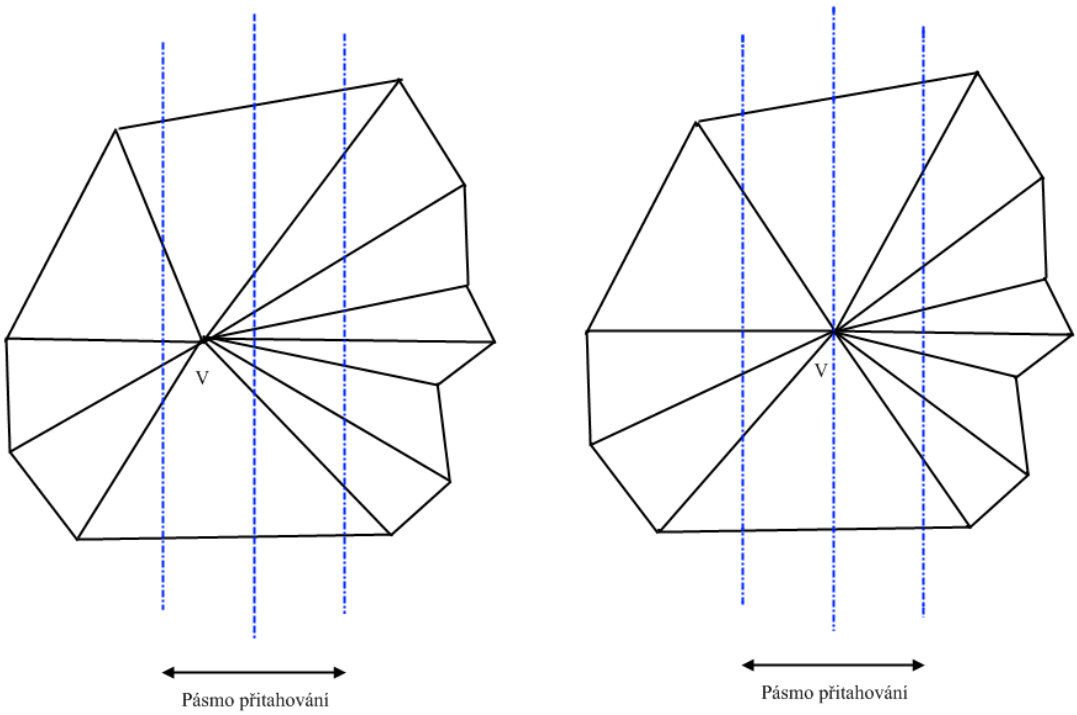
4.3 Dělení modelu

Původní model tedy dělíme podle vypočítané dělící roviny pomocí metody `splitTriangle(…)`. Postupuji tedy tak, že každý trojúhelník v původním modelu nechám projít touto funkcí. Princip jejího chování byl podrobně popsán v kapitole 3.3. Zjednodušený princip postupu je takový, že trojúhelníky leží buď na jedné nebo druhé straně dělící roviny celé nebo jimi dělící rovina prochází. Ty, kterými prochází, jsou rozděleny tak, že výsledkem je opět několik trojúhelníků, které celé leží na jedné nebo druhé straně roviny.

4.4 Přitahování vrcholů

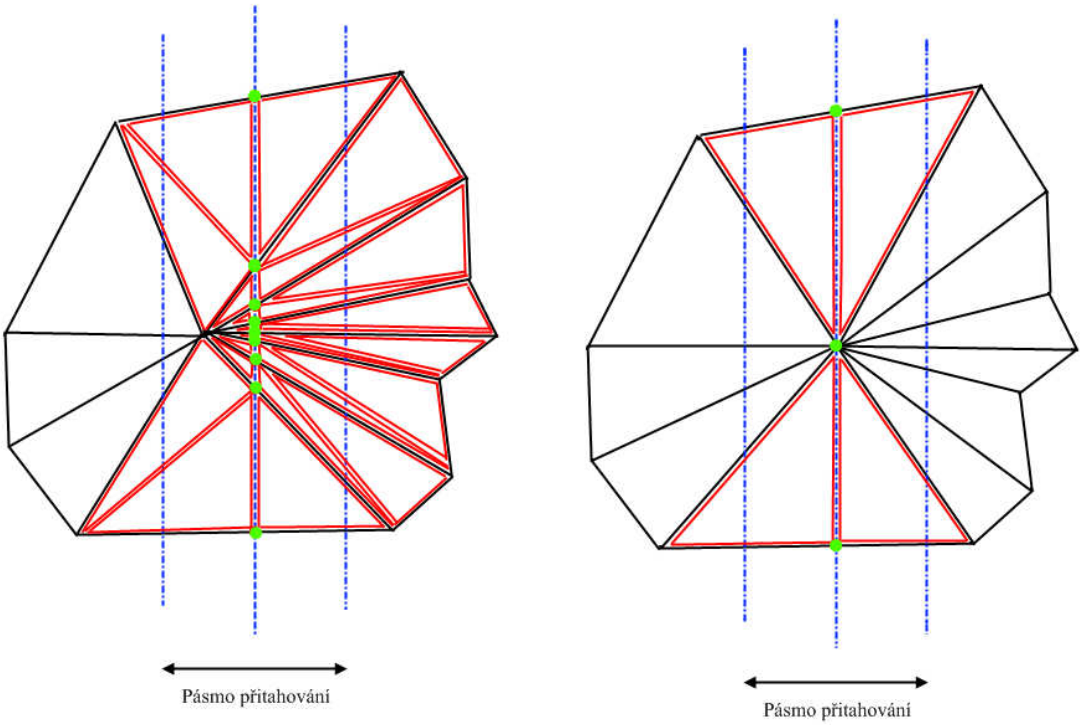
Původní úvahy o tom, jak snížit počet trojúhelníků vzniklých na hranách modelů při jejich dělení, počítaly s myšlenkou, že dělící rovina nebude procházet pouze středem plochy modelu, ale bude existovat určité rozmezí. V něm se nejprve zjistí, kde vznikne umístěním dělící roviny nejméně trojúhelníků, a tam poté dělící rovina skutečně povede. Pokusy o implementaci ukázaly, že takový postup není příliš efektivní, neboť v sobě zahrnuje velký počet hledání ideální polohy roviny.

Jako daleko zajímavější se jevila myšlenka, že dělící rovina bude procházet středem plochy modelu a bude existovat pásmo, ve kterém se vrcholy jakoby přitáhnou k dělící rovině tak, jak můžeme vidět na obrázku 4.2.



Obrázek 4.2: Základní myšlenka a ukázka myšlenky přitahování vrcholů k dělící rovině

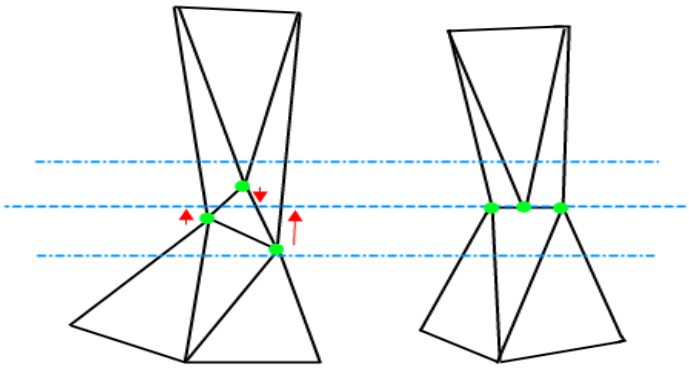
Tím dojde výraznému úbytku dělení trojúhelníků. V obecném modelu může být do jednoho vrcholu vztažen velký počet hran. Pokud takovýto vrchol, kde by došlo k vytvoření ohromného počtu nových trojúhelníků, přitáhneme k dělící rovině, je úspora jednoznačná. Modelový případ je znázorněn na obrázku 4.3. V levé části se model pouze dělí, to má za následek vznik nových 16 trojúhelníků, přičemž do dělení jich bylo zapojeno 8. Na pravém obrázku vidíme, že posunutím jednoho vrcholu přímo na dělící rovinu sice lehce pozměníme ráz krajiny, avšak při následném dělení vzniknou pouze 2 nové trojúhelníky a to pouze z 2 trojúhelníků, které je potřeba dělit.



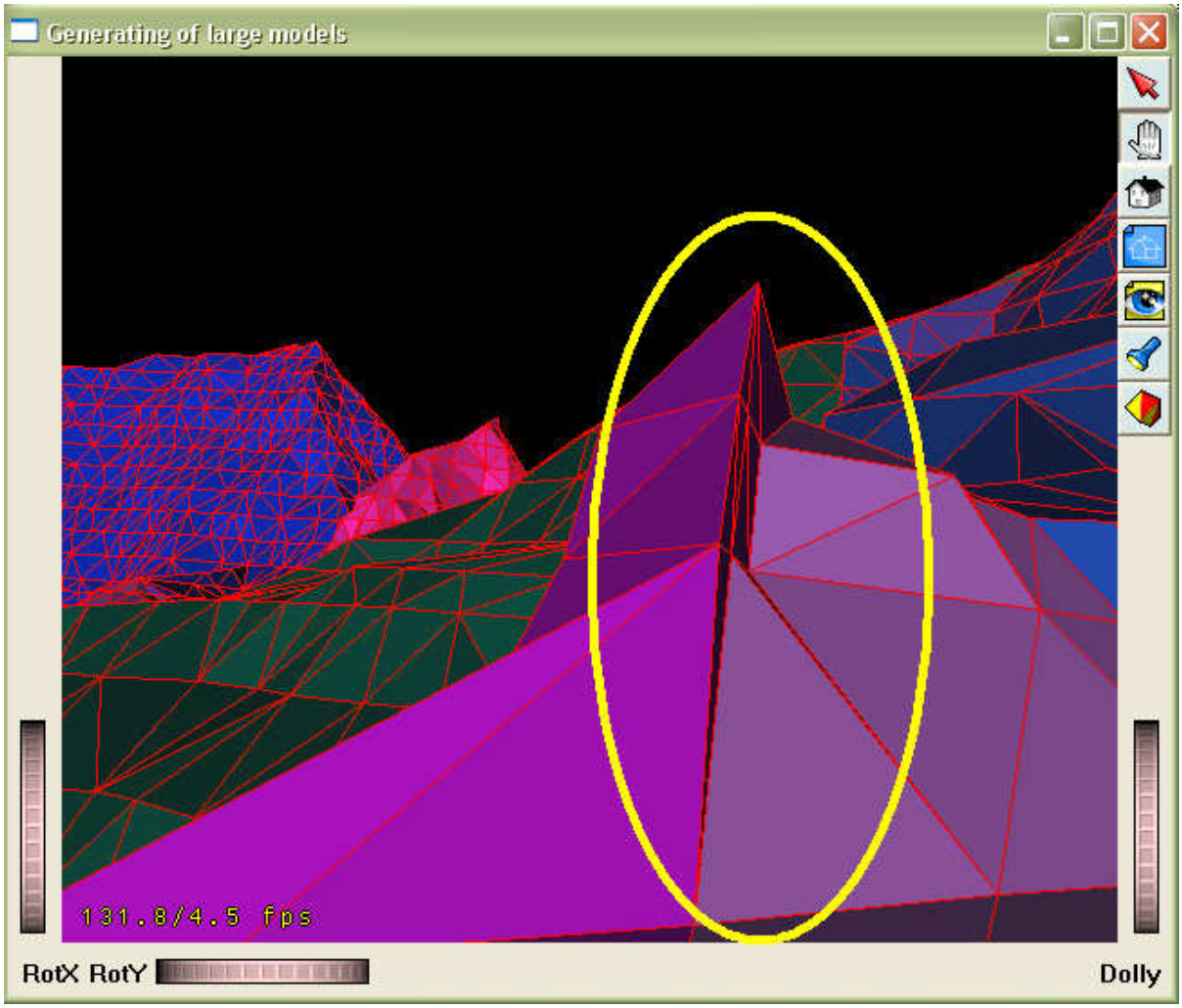
Obrázek 4.3: Demonstrace snížení počtu nových trojúhelníků vzniklých dělením při použití metody přitahování

Přitahování vrcholů k dělící rovině bylo původně realizováno metodou `pullPointsToPlane(...)`. Pracovala tak, že procházela všechny jednotlivé body modelu a ty, které byly v pásmu určeném pro přitahování, tedy jejich vzdálenost od dělící roviny byla menší než stanovený práh, byly přitaženy. Samotné přitažení znamená, že je vrchol přemístěn na dělící rovinu, a to po nejkratší možné trase, tedy kolmo. Při návrhu jsem uvažoval i o jiném než kolmém promítnutí, nakonec jsem však dospěl k závěru, že je nejvýhodnější a dochází při něm k nejmenším změnám v profilu krajiny.

Výše popsaná metoda však trpěla jistou chybou. Pokud v pásmu přitahování ležely všechny tři vrcholy, které tvořily trojúhelník, docházelo k tomu, že v modelu vznikaly artefakty. Pracovně jsem je označil jako „kolmé stěny“.



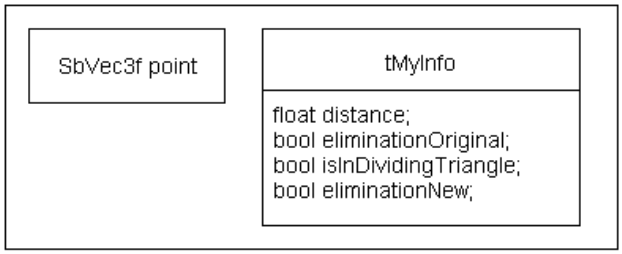
Obrázek 4.4 : Důvod vzniku kolmých stěn



Obrázek 4.5: Syndrom kolmých stěn rušící celkový ráz krajiny

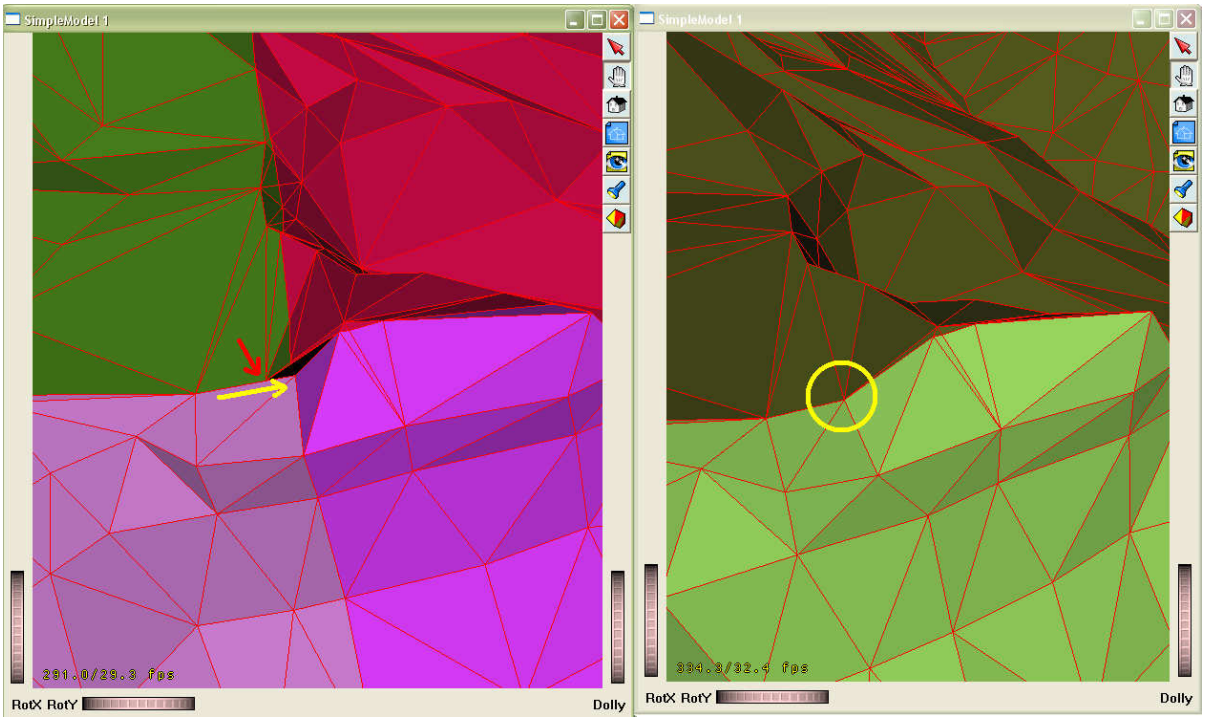
Byly důsledkem toho, že se všechny vrcholy ležící v pásmu přesunuly na dělící rovinu. Proto vznikla metoda `pullPointsToPlaneTriangleUpdate(...)`, kde byla přidána kontrola, zda všechny tři vertexy tvoří trojúhelník. Takové pak byly z procesu přitahovaní odstraněny. To ovšem nestačilo, a proto byla navíc optimalizována tak, aby řešila pouze trojúhelníky,

které spadají do pásma. Upravená metoda tedy nejprve projde pomocí metody `addBSPMyInfoAboutElimination(...)`všechny vertexy a každému z nich přidá informaci o jeho vzdálenosti od dělící roviny a příznaky určující, zda by měl být tento bod přitažen (tedy leží v pásmu). Protože může být jeden bod součástí více trojúhelníků, tak obsahuje příznak, zda je součástí nějakého trojúhelníku, který se má dělit, a poslední příznak určuje, zda se skutečně bod bude přitahovat. Každý bod tak projde sérií testů, které určí výsledek v posledním příznaku. Nepřitažené body patří trojúhelníkům, které je nutné rozdělit, aby nedošlo k výrazným deformacím původního terénu. Na obrázku 4.6 je struktura s přídavnými informacemi.



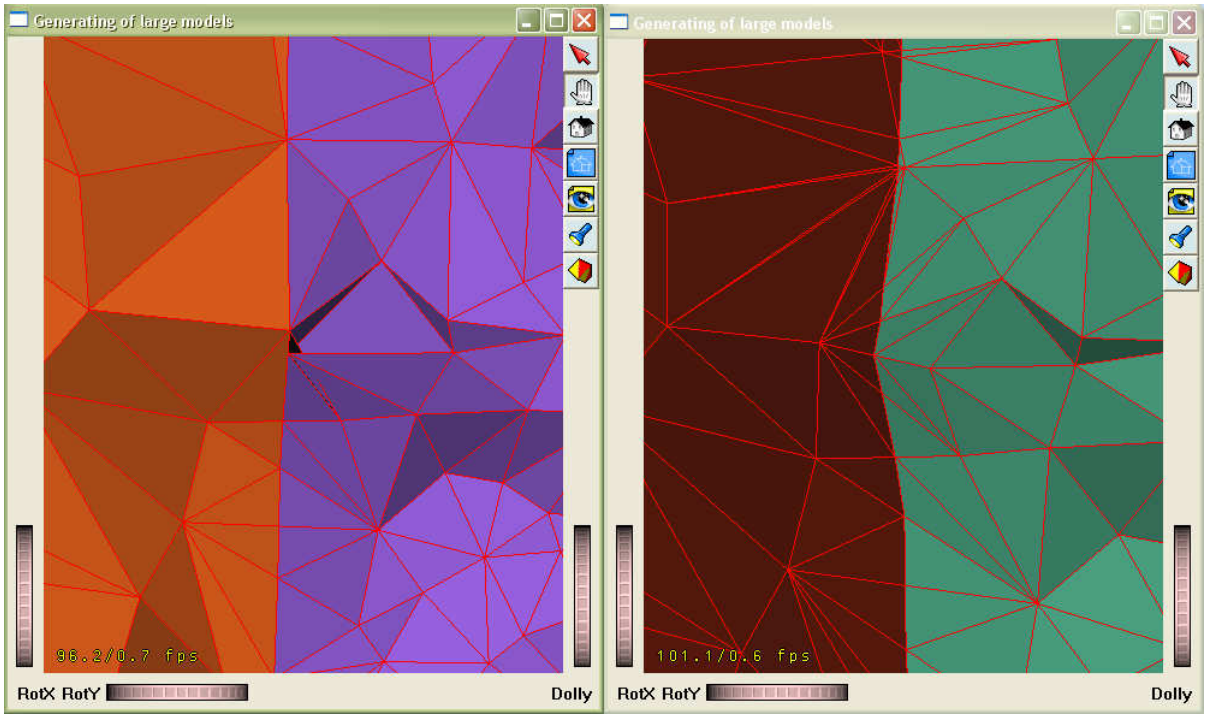
Obrázek 4.6: Struktura obsahující příznaky. Každý bod obsahuje informaci o vzdálenosti od dělící roviny, eliminationOriginal je true v případě, že spadá do pásma přitahování. IsInDividigTriangle je nastaveno pokud vrchol patří do nějakého trojúhelníku, který má být rozdělen.Pokud vrchol projde všemi testy a má být přitažen, je tato informace uložena v eliminationNew.

Testováním tohoto řešení jsem však zjistil, že to nebyla jediná chyba. K další docházelo na okrajích nově vznikajících modelů, respektive v místech, kde se stýkaly 4 modely – obrázek 4.7. Při přitahování a dělení prvních dvou (např. dělení na úrovni 1) bylo vše v pořádku. Vznikala ovšem tak, že při přitahování a dělení v další úrovni (modely na levelu 2) už každý nový model bral svou stranu hranice čistě za svou. Tedy bod ležící na hranici, vzniklý předchozím dělením, byl v obsažen v jednom i druhém modelu, a proto v některých případech na jedné straně k přitažení došlo a na druhé ne. Řešením bylo zavedení pole bodů, které vzniky dělením na hranicích a jejich porovnáváním s právě přitahovaným bodem.



Obrázek 4.7: Vlevo je znázorněna chyba. Zatím co bod označený červenou šipkou zůstal na svém původím místě, bod označený žlutou šipkou se přitáhl k nové dělící rovině a v modelu tak vznikla díra. Na pravém obrázku je situace těsně po dělení první rovinou. To co vypadá jako jeden bod jsou dva samostatné body (každý je definovaný ve svém modelu).

Zatím poslední neduh, objevený při prohlížení výsledku, je na obázku 4.8 vlevo. Příčinou je jisté rozložení vrcholů trojúhelníka v prostoru tak, jak je na obrázku 4.8 vpravo. Tato chyba byla objevena bohužel v nedávné době, proto se jí již žádného řešení nedostalo. Je však zajímavou úlohou pro rozšíření práce do budoucna.



Obrázek 4.8: Chyba - při přitahování dojde k otočení trojúhelníka (trojúhelníků)

4.5 Zjednodušení sítě trojúhelníků

Algoritmus zjednodušení sítě trojúhelníků je převzat z [4], kde je také podrobně popsán postup, jak pracuje. Jedná se ve své podstatě o variaci na decimaci stylem *edge collapse*. Tento algoritmus je však sestaven tak, aby zachovával hraniční trojúhelníky a nedocházelo tak ke špatnému navazování sousedních segmentů. Pracuje v několika krocích:

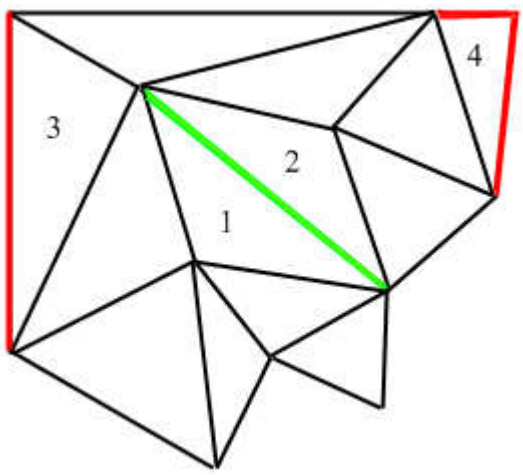
1. nalezení a seřazení hran podle vzdálenosti,
2. určení hran, které se mají odstranit,
3. mazání trojúhelníků,
4. mazání vrcholů.

Tato metoda poskytuje dostatečný kompromis mezi rychlostí a vizuální kvalitou. Je ovšem náchylná k chybě - pokud se v modelu vyskytuje *T-přechod*, může v modelu vzniknout díra.

4.6 Hledání hranic modelu

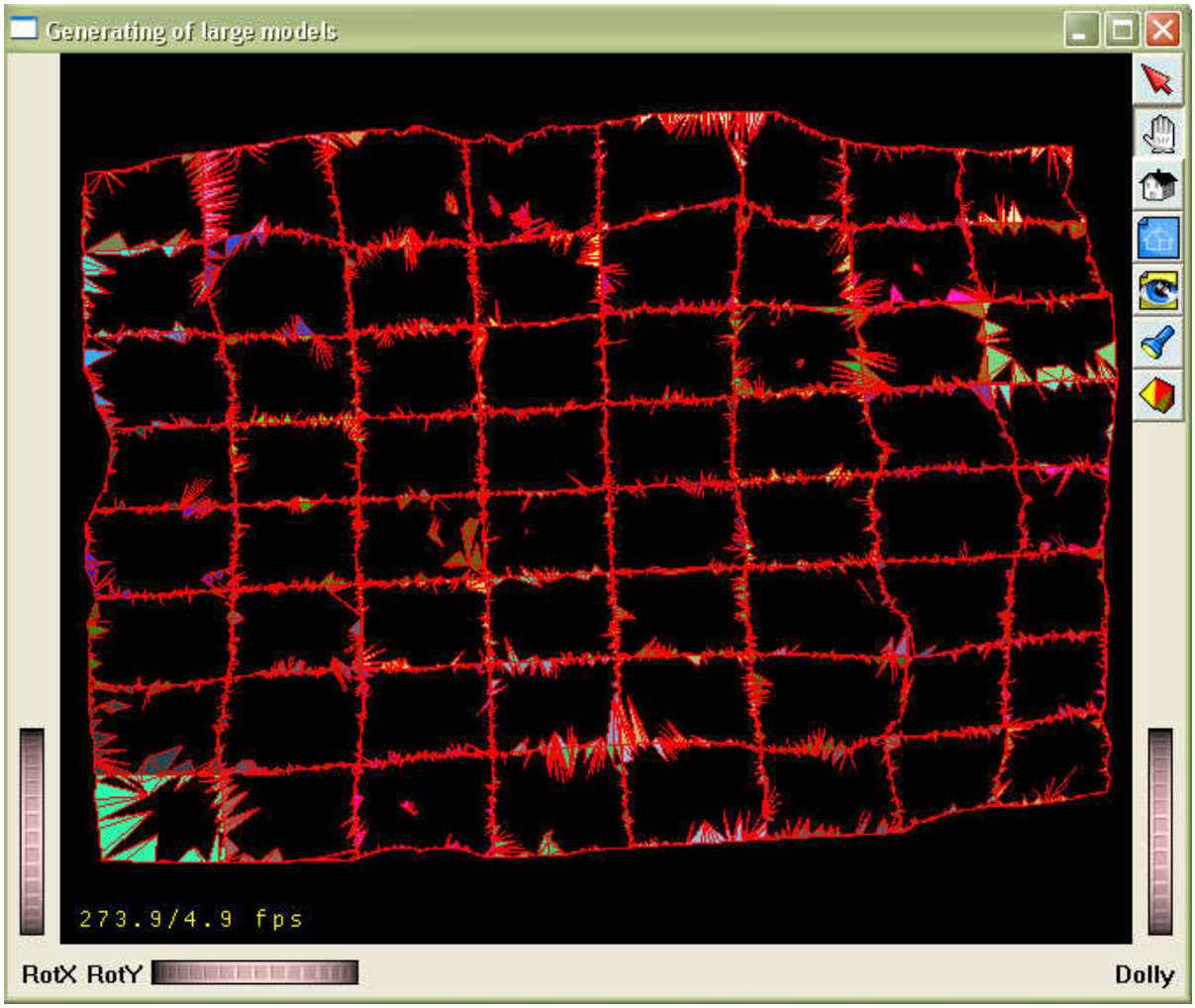
Pro segmentační algoritmus, který pracuje tak, že zachovává okraje, bylo nutné zjistit, které trojúhelníky jsou hraniční. Při přitahování bodů a dělení modelu jsem sice zjistil hranici, ale pouze tam, kde procházela dělící rovina. Myšlenku hledání hraničních trojúhelníků zachycuje obrázek 4.9 a vychází z úvahy, že pokud trojúhelník leží na hranici, pak se jeho jedna (hrana trojúhelníku č. 3) nebo dvě hrany (trojúhelník č. 4) vyskytují v modelu pouze jednou (červené hrany).

Zelená hrana se v modelu musí objevit dvakrát, neboť je sdílena trojúhelníky č. 1 a 2. Stručně řečeno, hraniční trojúhelník postrádá jednoho nebo dva sousední trojúhelníky.



Obrázek 4.9: Hledání hranic podle počtu trojúhelníků sdílejících jednu hranu

Metoda `findBoundaryTriangles(…)` pracuje tak, že prochází pole indexů, kde jsou definované jednotlivé trojúhelníky, každou hranu uloží do mapy a její počitadlo nastaví na jedničku. Při jejím dalším výskytu zvýší stav počitadla. Hrany, kterým po skončení algoritmu zůstane počitadlo na jedničce, jsou hraniční. V rámci výsledného programu lze takto nalezené hranice zobrazit (více v kapitole Ovládání programu). Díky tomu lze zjistit i chybně definované trojúhelníky v modelu, které se, ač nejsou na hranicích, přesto ve výsledném grafickém výstupu zobrazí. Takový případ je vidět na obrázku 4.10.



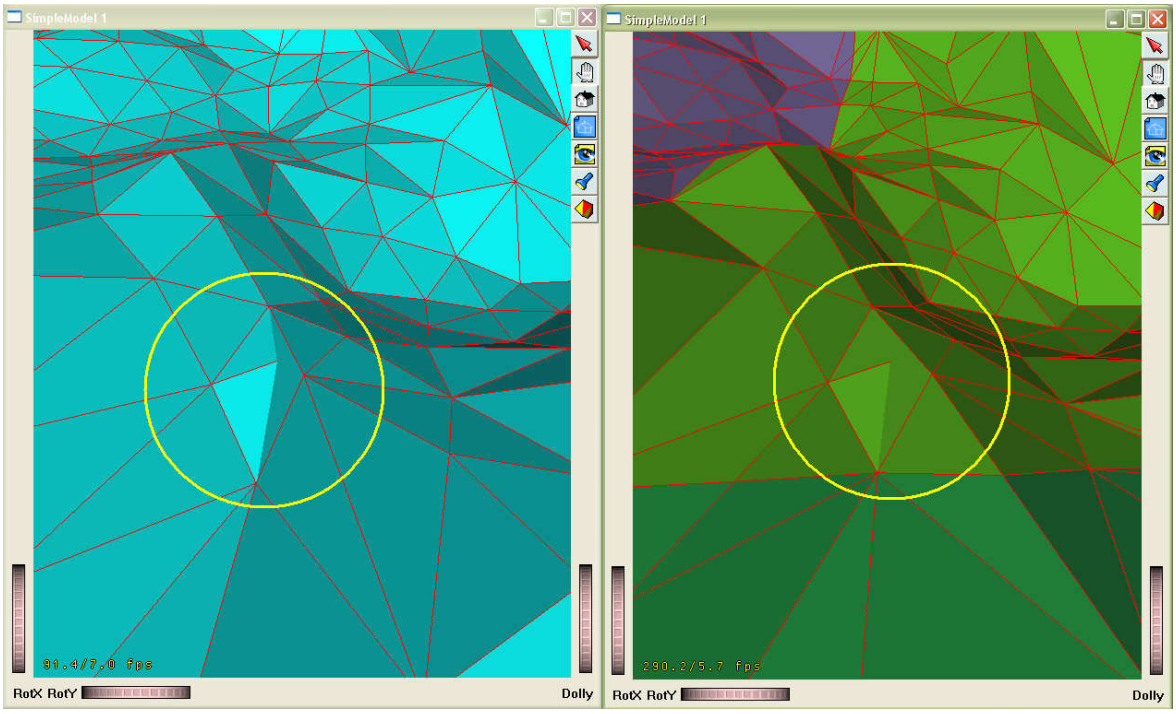
Obrázek 4.10: Vykreslení pouze hraničních trojúhelníků. Uvnitř některých z nich lze nalézt chybně definované trojúhelníky.

4.7 Chyby v původním modelu

Při tvorbě algoritmu na dělení segmentů a přitahování vrcholů jsem čas od času, při testování jednotlivých etap, narazil na podivné chování programu. Jednalo se především o nejrůznější artefakty v trojúhelníkové síti modelu. Modely jsem používal tři. Dva z nich mi vytvořil a poskytl vedoucí práce. Pro základní testování programu postačovaly.

Pro pokročilejší otestování funkčnosti algoritmu jsem si vypůjčil model cilaos.iv, na kterém byly prováděny testy i ve vlastní práci [4]. Tvůrcem toho modelu je firma CadWork. Model znázorňuje část obydleného kráteru na ostrově Reunion, který leží poblíž ostrova Madagaskar. Právě při testování na tomto modelu jsem narazil na několik chyb, které, jak se ukázalo, nebyly produktem algoritmu, ale byly obsaženy již v původním modelu. Na obrázcích je vždy vlevo zobrazen model bez jakýchkoliv úprav a vpravo po použití algoritmu na dělení segmentů a přitahování vrcholů. Více obrázků s chybami v původním modelu je možné najít v příloze A. Chybně definované trojúhelníky v původním modelu lze také přehledně zobrazit použitím programu TriangleDivision.

Zadáním volby „zobrazovat pouze hranice“ v grafickém zavaděči, dostaneme trojúhelníkovou síť, kde se kromě skutečných hranic modelu, či modelů, zobrazí také trojúhelníky, které nejsou v modelu správně navázány na okolí.



Obrázek 4.11: Chyby v původním modelu (vlevo původní model, vpravo zpracovaný)

4.8 Optimalizace

Během výpočtu hraničních trojúhelníků se navíc ukázalo, že trojúhelníky nejsou pouze špatně definované – což vyplynulo z faktu, že existují i uvnitř modelu hrany, které nebyly navázány na žádnou další geometrii, ale také z toho, že v modelu existuje mnoho hran, které jsou sdíleny ne pouze dvěma ale často třemi a více trojúhelníky. Tento fakt byl zajímavý i z toho hlediska, že nebyl vizuálně pozorovatelný, a proto jsem na něj narazil až ve chvíli, kdy jsem zobrazoval pouze hranice modelu a mnoho hranic nebylo kompletních a byly v nich díry. Podle předpokladu, že hraniční trojúhelníky jsou jen ty, které mají alespoň jednu hranu takovou, že se vyskytuje v modelu pouze jednou, mělo být vše v pořádku. Kontrolou algoritmu jsem zjistil, že ač jsem předpokládal pouze hrany s četností výskytu jedna a dva, v modelu je mnoho hran, které mají četnost vyšší. Jednalo se tak především o duplicitní (či více násobnou) definici stejných trojúhelníků. První optimalizace se tedy zabývala odstraněním těchto anomálií.

Další optimalizace, která s předchozí také trochu souvisela, byl vznik nových vrcholů při dělení modelu. Trojúhelníky jsem dělil jednotlivě, pokud tedy dva trojúhelníky sousedily a byly děleny, pak na jejich společné hraně vznikly ve stejném místě dva body se stejnými souřadnicemi,

náležící však do dvou různých trojúhelníků. Protože jsem v práci s knihovnou COIN využíval pro definici vertexu strukturu `SbVec3f`, měl jsem k dispozici přesnost výpočtu pouze v rámci rozsahu typu `float`, což někdy způsobilo, že se, ač měly být dva body na hranici dvou dělených trojúhelníků naprosto stejné, v řádu desítitisícin lišily. Proto jsem se rozhodl vytvořit pro optimalizaci metodou `optimalizeCoord()`, která vždy po přitahování a dělení modelu upravila pole vrcholů tak, že zachovala pouze jednu reprezentaci každého bodu (se zaokrouhlením na tři desetinná místa) a adekvátně k tomu změnila pole indexů.

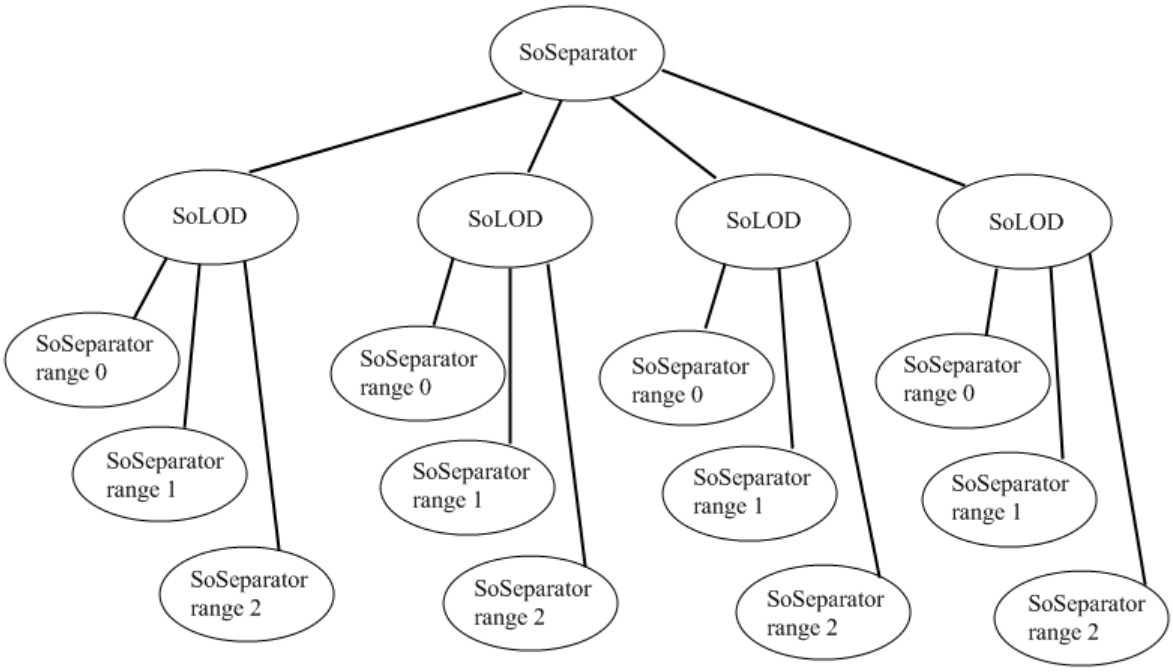
Nutné bylo hlídat ještě jednu věc, a to, že při přitahování vrcholů, dělení segmentů i decimaci trojúhelníků někdy vznikají chybné trojúhelníky, které mají všechny vrcholy v jedné přímce. Na takové Open Inventor reaguje varovným hlášením a proto je nutné je odstranit. V rámci metody `isTriangleRightDefined()`, která je jedním z testů v `optimalizeCoord()`, nejprve zjišťuji, zda se některé ze tří bodů sobě přímo nerovnají, a poté zkoumám pro každý vrchol, jestli neleží na úsečce tvořené zbylými dvěma body.

4.9 Řešení LOD pomocí Open Inventoru

Práce je zaměřena na zobrazování rozsáhlých scén. Proto, že máme k dispozici model rozdělený na menší části, bylo zajímavé vyzkoušet, jak bude, z hlediska výkonnosti a rychlého zobrazování terénu, fungovat technika *Level of detail*. Nabízí se několik variant řešení. Pravděpodobně nejnáročnější variantu představuje vytvoření vlastního systému, který by se o zobrazování jednotlivých úrovní LOD staral. Vzhledem k tomu, že práce není primárně zaměřena na precizní práci s úrovní detailu, se o této variantě vůbec neuvažovalo. Nicméně se tato možnost jeví jako zajímavá práce do budoucna. Dal by se pomocí ní například částečně ovlivnit tzv. *popping efekt*.

Další varianta byla využít možnosti Open Inventoru. Samotná knihovna obsahuje třídu, která je schopna starost o přepínání a správu LOD sama zajistit. Ačkoliv jsem v základním návrhu v kapitole 3 počítal s vytvořením struktury *Quad Tree* a paralelou s algoritmem *Chuned LOD*, zvolil jsem nakonec přístup poněkud odlišný, a to takový, že jsem pro práci s *Level Of Detail* využil pouze poslední (nejdrobněji nadělené) části modelu.

Třída `SoLOD` je potomkem třídy `SoNode` a může být tedy použita jako klasický prvek scény.



Obrázek 4.12: Typické použití uzlů třídy SoLOD a jejich začlenění do scény

Jeho použití vidíme na obrázku 4.12. Každý uzel typu `SoLOD` může obsahovat několik dalších uzlů scény, které mohou mít definovány vlastní texturu, geometrii a další položky. Typicky se do uzlu `SoLOD` ukládá 3 – 5 různých reprezentací úrovně detailu scény.

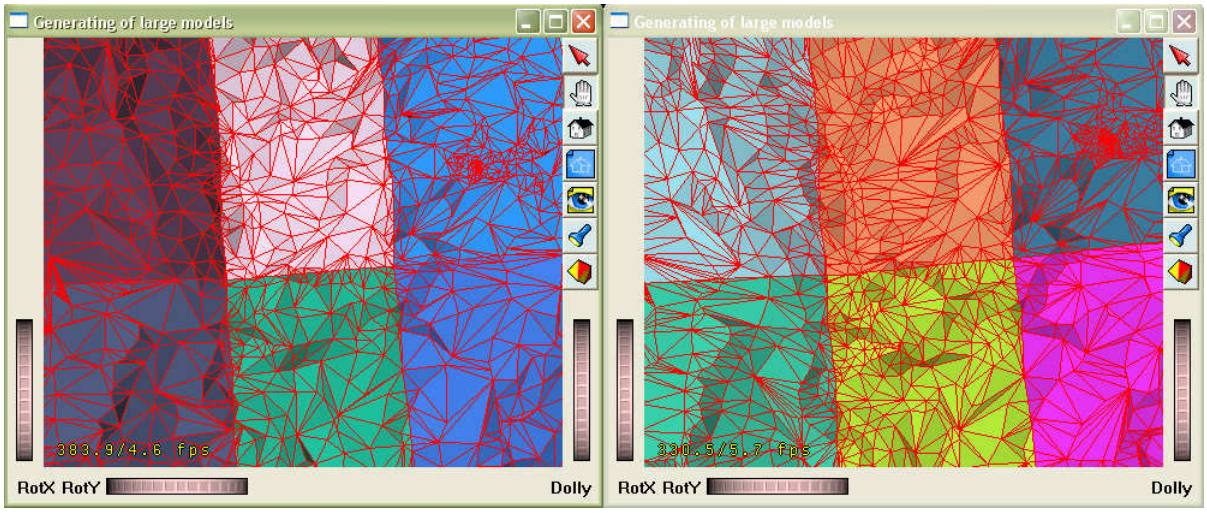
Uzel `SoLOD` obsahuje dva důležité parametry – `float range[]` a `SbVec3f center`. Pak dále obsahuje uzly obsahující geometrii jednotlivých úrovní detailu. `Center` udává střed objektu, který je použit jako referenční bod, od kterého se pak odečítají hodnoty vzdálenosti uložené v parametru `range[]`, který udává, v jaké vzdálenosti od kamery se bude používat ten který uzel.

Pro svou potřebu jsem pro každý díl modelu vždy vytvořil jeden uzel `SoLOD`, do kterého jsem uložil tři různě zjednodušené verze modelu. K redukci počtu trojúhelníků jsem použil postup uvedený v kapitole 4.4. Parametr `center` jsem nastavil na střed pomyslného bounding boxu každého modelu a vzdálenosti v poli `range[]` se nastavují na rozmezí 0 – dvojnásobek nejdelší strany modelu, dvojnásobek až čtyřnásobek, a více než čtyřnásobek nejdelší strany modelu. Parametry redukce jsou nastaveny tak, že v nejvyšším rozlišení je použit originální model, dále pak je provedena redukce o 30% a o 90% trojúhelníků.

4.9.1 Vizuální vlastnosti tohoto řešení

Protože existují 3 konkrétní úrovně a přechod mezi nimi není nijak speciálně ošetřen, je při přechodu z jedné úrovně ke druhé vidět patrný úbytek respektive nárůst trojúhelníků v krajině. Tento jev popisovaný jako *popping efekt* by se dal odstranit případným vlastním algoritmem nebo použitím

specializované knihovny, která se danou problematikou hlouběji zabývá. Pro demonstrační účely této aplikace je použití `SoLOD` vyhovující.



Obrázek 4.13: Rozdíl mezi dvěma úrovněmi LOD, vlevo menší detail, vpravo detailnější stupeň

Na obrázku 4.13 je vidět ukázka z aplikace. Nastaveno bylo dělení do 6 úrovně, šířka pásma přitahování byla 20 pixelů. Je zřetelně vidět, že některé vrcholy jsou díky decimaci vynechány, ale celkový dojem nepůsobí rušivě.

4.10 Ovládání programu

Základní program je koncipován jako konzolová aplikace s možností nastavení parametrů přes příkazovou řádku. Takové ovládání však není příliš intuitivní ani uživatelsky příjemné. Vytvořil jsem proto pro aplikaci *TriangleDivision.exe* zavaděč, ve kterém je možné všechny parametry programu nastavit v grafickém režimu. Jeho velkou výhodou také je, že umožňuje spustit několik instancí programu s různým nastavením za sebou, a tak dává možnost přímo porovnat výsledky dvou různých nastavení. Obrázek zavaděče s popisy jednotlivých voleb je uveden v příloze B.

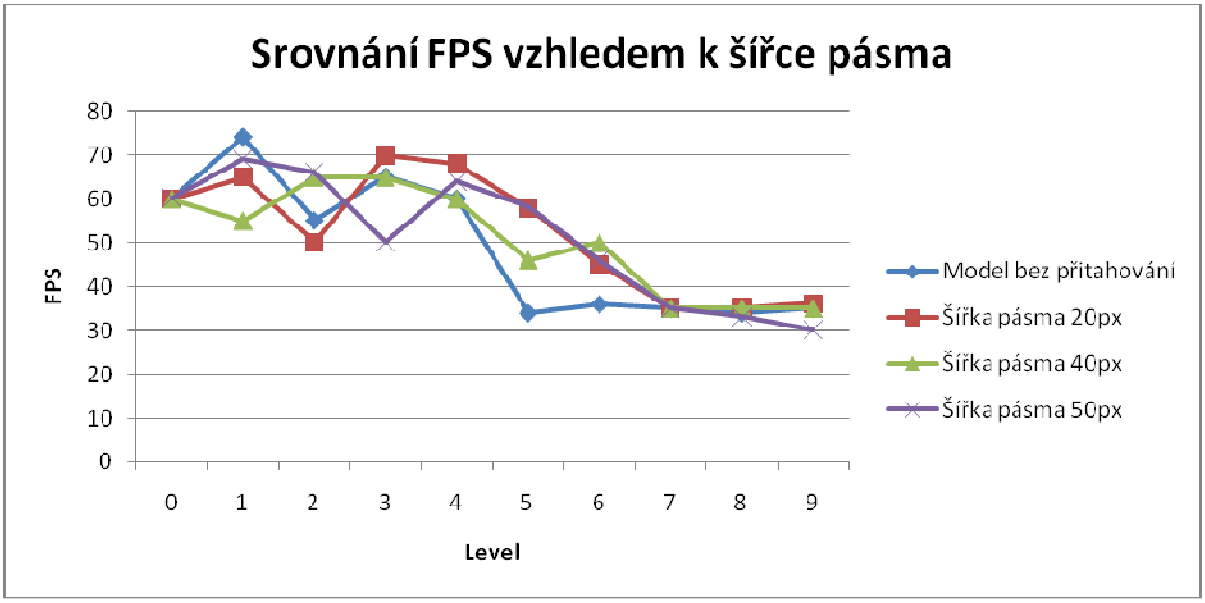
Všechny parametry se dají nastavit pomocí přiložené aplikace *Launcher* v grafickém uživatelském rozhraní. Po spuštění programu TriangleDivision.exe se objeví standardní okno knihovny *SoWIN*. To umožňuje s modelem pohybovat, přibližovat se k němu a otáčet jím. Navíc také obsahuje paletu nástrojů, které například umožňují vrátit model do výchozí polohy, či jej efektivněji přiblížit. Pod pravým tlačítkem myši je kontextové menu, kde se dá měnit pohled na model. Je možné zobrazit jej jen stínovaný, jako drátový model, jejich kombinaci, atd.

5 Testování

Významným bodem zadání bylo také testování a zhodnocení výsledků navržené aplikace. V této kapitole se tedy zaměřím na výkonnostní testy a zhodnocení dosažených výsledků. Všechny testy jsou provedeny na notebooku značky DELL, rozlišení obrazovky 1280x800 pixelů, grafická karta NVIDIA GeForce 8600M GT - 256MB, procesor Intel Core 2 Duo T5470 (1.6 GHz), 2 GB RAM, operační systém Windows XP (SP2) a modelu cilaos.iv, kromě posledního testu, kde je použit model abom2.iv, poskytnutý Ing. Jaroslavem Přibylem.

Zkratka FPS vychází s anglického originálu *Frames Per Second* (snímky za sekundu). Udává počet snímků, které je zobrazovací zařízení schopno vykreslit na obrazovku za jednu sekundu. K zjištění tohoto čísla využívám funkce knihovny COIN, která zadáním parametru COIN_SHOW_FPS_COUNTER zobrazuje do levého spodního rohu okna aplikace dvě čísla. Levé udává maximální *frame rate*, jaký by byl v dané scéně možný, pravé ukazuje aktuální *frame rate*. Při zjišťování hodnot FPS provádím testování v režimu zobrazování přes celou obrazovku.

5.1 Srovnání FPS nadělené scény s přitahováním a bez něj

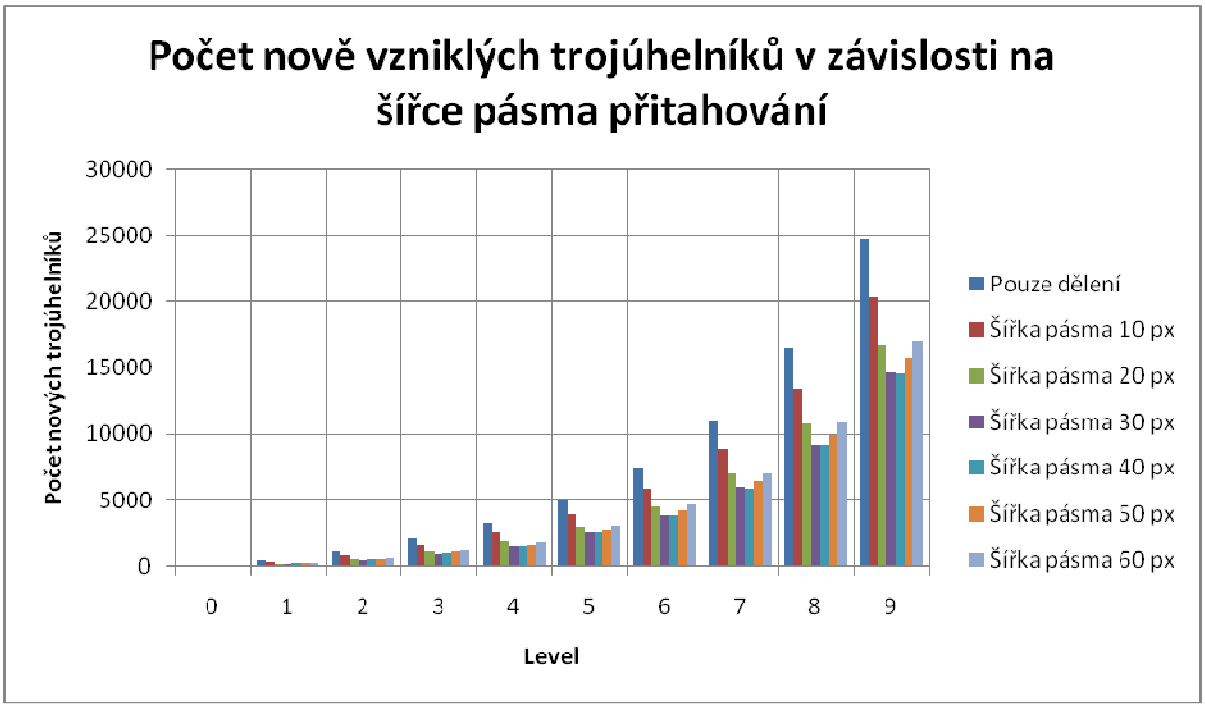


Obrázek 5.1: Srovnání FPS nadělené scény s různou hodnotou šířky pásma přitahování

Graf na obrázku 5.1 zachycuje srovnání FPS scény, která byla pouze nadělena, se scénami, které byly zpracovány algoritmem pro přitahování vrcholů. Jak můžeme vidět, nenadělená scéna zprvu vede, poté se však její snímkovací frekvence propadne a scény s menším počtem trojúhelníků jsou na tom o

něco lépe. Nicméně nejlepší průměrný výsledek vykazuje šířka pásma 20px. Parametr Level znamená, kolikrát byl původní model dělený – celkový počet nově vzniklých modelů je pak 2^{LEVEL}.

5.2 Závislost šířky pásma na počtu nově vzniklých trojúhelníků



Obrázek 5.2: Srovnání počtu nových trojúhelníků při dělení scény s různou hodnotou šířky pásma přitahování

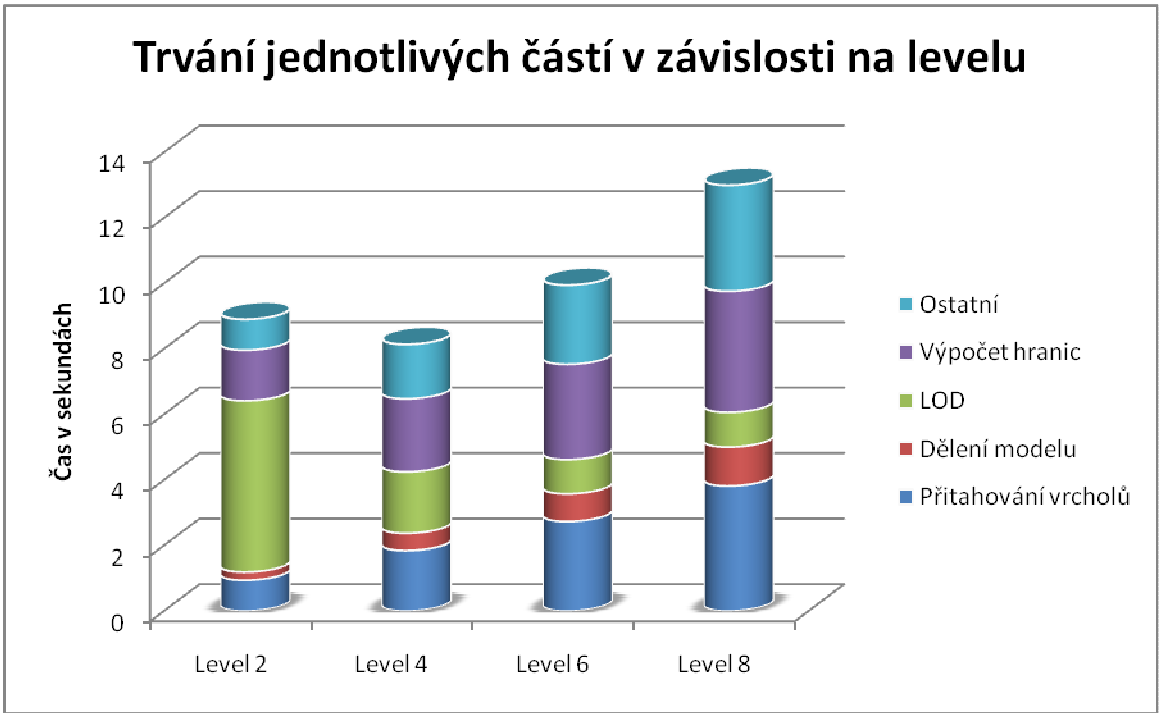
Protože bylo hlavním úkolem najít řešení, které sníží počet vznikajících trojúhelníků, rozhodl jsem se pro tento typ testu. V něm jsem sledoval jak, se zvyšující se úrovní dělení modelu, roste počet nově vzniklých trojúhelníků, a jak je metoda přitahování vrcholů účinná. Zajímavý výsledek je pak na obrázku 5.2, kde je zřetelně vidět, že metoda je funkční, použitelná, a při správném nastavení je možné s její pomocí zabránit vzniku až 40% nových trojúhelníků. Je to sice za cenu drobné změny v geometrii modelu, pokud ovšem nelpíme na důkladné přesnosti (například GIS aplikace) a uvažujeme o opravdu velkých modelech terénů s délkou hrany řádově tisíce pixelů, je tato změna posunu o několik desítek pixelů stěží postřehnutelná.

Za zmínku také stojí fakt, že je opravdu nutné správně volit šířku pásma. Z testu vyplývá, že nejideálnější je pásmo kolem 40 pixelů. Je také vidět, že ani nastavení větší šířky pásma nepřispívá k větší úspoře vznikajících trojúhelníků, právě naopak. Tento fakt je pravděpodobně způsoben podmínkou, podle které se musejí trojúhelníky, mající všechny vrcholy uvnitř pásma, dělit.

Podmínka je ochranou proti vzniku kolmých stěn na okrajích modelu. Zvětšením šířky pásma tak dojde k tomu, že bude také více trojúhelníků, které budou celé ležet v pásmu, a tedy se dělit.

5.3 Časová náročnost jednotlivých komponent

Z hlediska např. pozdějších optimalizací je vhodné zjistit, které komponenty systému pracují nejdéle nebo jsou používány nejčastěji, abychom byli schopni zaměřit se na nejvíce vytížené partie a program tak cíleně zefektivnit.



Obrázek 5.3: Srovnání časové náročnosti jednotlivých částí algoritmu při zvyšování hloubky dělení

Provedl jsem proto test sledující časové vytížení. Na obrázku 5.3 je grafické znázornění tohoto testu. Při dělení modelu do několika málo částí, má jednoznačně nejnákladnější režii vytvoření struktury s modely v různých úrovních detailu. Tato režie je nižší, se stále se zmenšující se plochou nově vznikajících podmodelů. Při dělení na levelu 8 pak zabírá vůbec nejméně výpočetního času. To metoda starající se přitahování vrcholů má tendenci zcela opačnou. Pokud model dělíme na více částí, používá se mnohem více a výpočet se prodlužuje. Stejně je na tom také část, která dělí modely. Její nárůst je však velice pozvolný a dalo by se říci předvídatelný – čím více modelů má vzniknout, tím více musí být také v permanenci kód, který to zajišťuje. Poměrně velkou část doby stráví aplikace hledáním hranic jednotlivých modelů. Tato pasáž také s hloubkou levelu roste. Kromě těchto hlavních částí pak aplikace tráví čas v jiných metodách. Zajímavostí tohoto testu je, že mezi dělením na levelu 2 a levelu 4 aplikace zrychlila a to poměrně výrazně. Testování jsem prováděl několikrát a vždy se

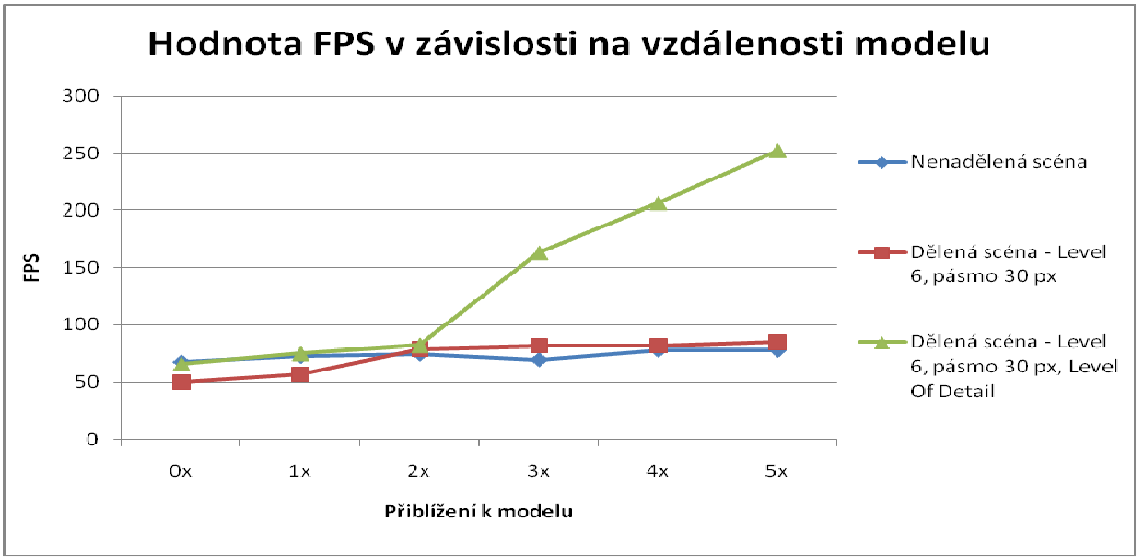
stejným výsledkem. Je možné, že režie kolem tvorby LOD je při dělení na levelu 2 tak významná, že, ač se neprovádí tolik dělení jako na levelu 4, dochází tím k velkému zpozdění. Tuto úvahu podporuje také fakt, že je doba tvorby LOD na levelu 4 oproti levelu 2 poloviční.

5.4 Srovnání FPS nadělené a nenadělené scény

Předmětem zkoumání bylo také, jak velký vliv má použití technologie LOD na FPS zobrazované scény. Porovnával jsem snímkovací frekvenci tří modelů – nedělené scény, modelu děleného na level 6 s pásmem přitahování 30 pixelů a stejně dělenou scénu s vykreslováním pomocí LOD. Test probíhal tak, že jsem si vždy scénu zobrazil, zjistil FPS a plynulým přiblížením o definovanou vzdálenost (funkce přiblížení viz ovládání programu příloha B) jsem se posunul o jeden krok blíže ke scéně. Přibližování jsem opakoval 5x. Předpokládal jsem, že nedělená scéna bude mít nejhorší výsledky, lepší bude scéna dělená a test dopadne nejlépe pro model s LOD.

Hypotézy se potvrdily jen částečně. Při pohledu na kompletní model měla nejlepší výsledky, byť velice těsně, právě nedělená scéna před scénou s LOD a děleným modelem. Řešení s LOD jasně dominovalo až zhruba v polovině testu – jeho výkon byl oproti ostatním zobrazením dvojnásobný. Druhá byla scéna dělená a nedělený model byl těsně poslední.

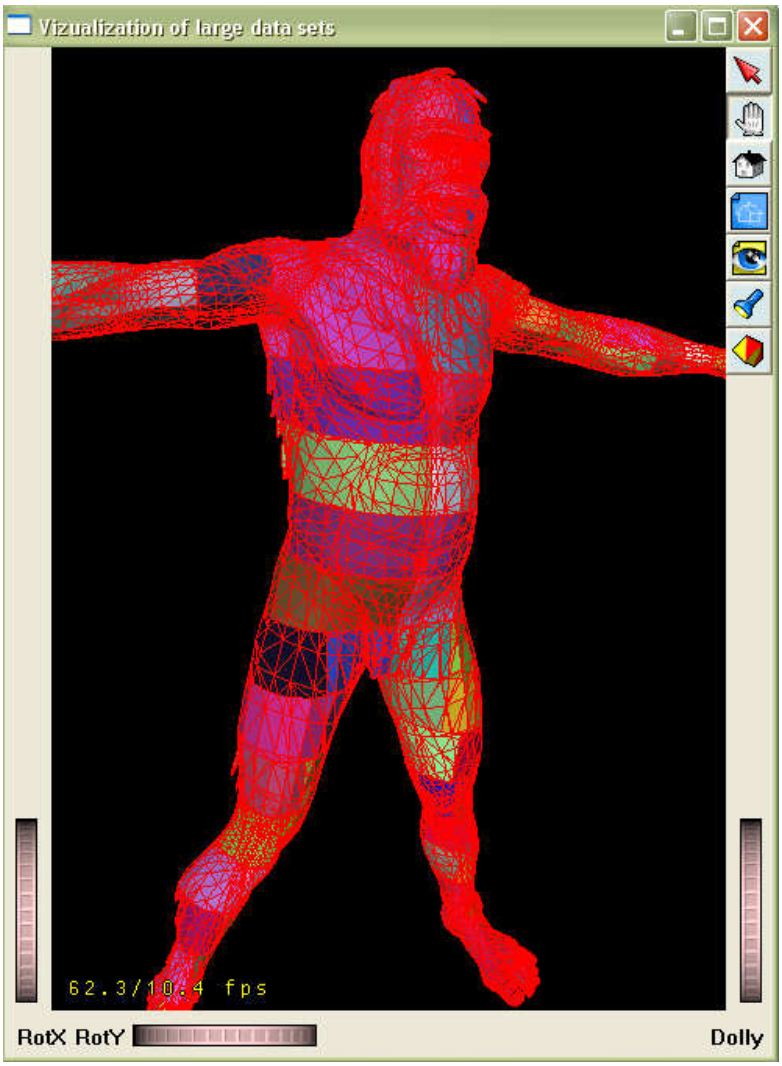
Příčinu slabšího začátku dělené scény vidím ve vyšší režii i vyšším počtu jednak zobrazovaných částí, tak trojúhelníků. Do druhého přiblížení, tak nedělená scéna držela krok i s řešením LOD. To ovšem své přednosti prokázalo, jakmile jsem se modelu dostal na takovou vzdálenost, že byl přes celou obrazovku. Projevilo se zde ořezání pohledovým tělesem, které je možné díky implementovanému dělení modelu na části. Průběh testu je vynesen v grafu na obrázku 5.4.



Obrázek 5.4: Srovnání FPS nedělené scény se scénou pouze nadělenou a scénou nadělenou využívající LOD

5.5 Dělení trojrozměrného modelu

Cílem tohoto testu nebylo zjišťování výkonu aplikace, ale spíše ověření toho, že postup, jakým je řešení provedeno, může být použit nejen na plošné modely terénů, ale také na uzavřené 3D modely. Na modelu opičího muže jsem vyzkoušel dělení, přitahování vrcholů, decimaci i LOD. Protože aplikace nebyla původně na uzavřené 3D modely určena, stávalo se, že některé segmenty byly větší než jiné. Což sice z vizuálního hlediska příliš nevadí, ale někdy v místech, kde s větším segmentem sousedily dva menší, docházelo ke vzniku *T-přechodů*. LOD funguje také dobře, pouze se sem tam, při vzdálenějším pohledu, vyskytují chybějící trojúhelníky. To je způsobeno použitým decimacím algoritmem, který byl také stavěný spíše na plošné modely terénů. Na obrázku 5.5 je rozdělený uzavřený 3D model.



Obrázek 5.5: Rozdělení uzavřeného 3D modelu

6 Další vývoj projektu

Z hlediska dalšího vývoje projektu je implementovaná aplikace vhodná pro různé způsoby rozšiřování. V této kapitole se budu zabývat tím, jak by se dal stávající systém vylepšit. Postup, který zatím implementovaná aplikace nabízí, je jen jedním z několika směrů kterými se dá jít. Proto v této kapitole uvedu několik možných optimalizací (i když to, zda se jedná o skutečně lepší řešení, bude také předmětem další práce. Také zde promyslím možnosti rychlého načítání částí modelu z hlediska práce s pamětí.

6.1 Možná rozšíření

6.1.1 Odstranění „popping“ efektu pomocí Shaderů

Při přechodu mezi jednotlivými stupni LOD může docházet k nepříjemnému efektu probliknutí scény v situaci, kdy je vyšší detail krajiny nahrazen sníženým počtem trojúhelníků například na obzoru a dojde jakoby ke skokovému snížení, respektive zvýšení horizontu. K pozvolnějšímu řešení těchto situací by se daly využít *Shadery*, konkrétně *Vertex Shader*.

6.1.2 Práce s texturou

Díky pravoúhlému rozdělení modelu, se navíc velmi zjednodušuje mapování textury na model. Díky OI a jeho struktuře SoLod bychom dokonce v jednotlivých uzlech *Level Of Detail* mohli mít uloženou, nejen několika násobnou a zjednodušenou verzi modelu, ale také několik stupňů kvality textury, která by se přepínala zároveň s geometrií. Případně vytvořit celý systém, který by se staral o načítání resp. odkládání textur z resp. na pevný disk v případě velmi rozsáhlých modelů a velkého množství kvalitních textur.

Level of detail pro textury je metoda používaná pro úsporu paměti a pro zrychlení renderování v rámci grafického akcelerátoru. Abychom však mohli šetřit paměť, je nutné vytvářet a mazat opakovaně texturovací objekty vytvořené pro konkrétní úroveň detailu dané části scény. Inicializace každého texturovacího objektu při jeho prvním volání je však časově velmi náročná operace a může významně narušit plynulost referování. Tento problém se projevuje především při vytváření texturovacích objektů o velikosti 2048x2048 a to i na nových grafických akcelerátorech. Částečným řešením tohoto nepříjemného stavu je vytváření prázdných texturovacích objektů a jejich postupné naplnění daty, čímž lze dosáhnout kratšího celkového času inicializace texturovacích objektů při prvním volání.

Úplné odstranění potřeby inicializace texturovacího objektu je možné zavedením statických referenčních texturovacích objektů o určitém počtu a ty pak naplňovat daty podle potřeby.

Nevýhodou tohoto přístupu je fakt, že vzroste statická obsazenost paměti takto vytvořenými referenčními texturovacími objekty. V případě např. pěti referenčních texturovacích objektů o velikosti 2048x2048 pro texturu o bitové hloubce 24bpp bude velikost permanentně obsazené paměti na GA rovna cca 63MB. V případě textur o rozlišení 4096x4096 už to bude 252MB. V případě nepoužívání těchto statických texturovacích objektů je možné jim nastavit nižší prioritu uchování v paměti GA a v případě delší časové nepotřebnosti tak budou přesunuty z paměti GA do operační paměti.

6.1.3 Realistické pojetí, síťová aplikace

Aby scéna působila více realisticky, mohla by se celá umístit například do *skyboxu*, přidat oblohu, zvuk. Případně by se mohla dodělat podpora pro ukládání rozdělených modelů do různých formátů. Zajímavé by také mohlo být zkoumání využití výstupů pro zobrazování rozsáhlých modelů a jejich přenosy po síti. Model rozdělený na menší části by mohl být ideální cestou.

7 Závěr

Zadáním diplomové práce bylo seznámit se s algoritmy, které souvisí se zobrazováním rozsáhlých scén a navrhnout a vytvořit jednoduchou aplikaci pro rozdělení rozsáhlé scény na menší díly. V rámci práce byly oba body splněny, byť implementace postrádá práci s texturami, kterou jsem původně zamýšlel. Aplikace přináší netradiční řešení snížení velkého počtu vznikajících trojúhelníků při dělení, a to zavedením algoritmu pro přitahování vrcholů. V kapitole 6 jsem navrhl několik cest, kterými by se mohl projekt dále ubírat.

Výsledky práce se dají využít všude tam, kde potřebujeme zobrazovat rozsáhlé modely terénů, u kterých nám nevadí jisté nepřesnosti. Například tedy v leteckých simulátorech, či počítačových hrách. Využít se například příliš nedá pro GIS aplikace, kde je nutné zachovávat vysokou přesnost. Díky rozdělení rozlehlého modelu na malé části, by se dal (a přímo se to nabízí) také výstup z programu využít v síťových aplikacích, kde by velký model nemusel být načítán najednou, ale právě po jednotlivých kusech a úrovních detailu například s využitím *progressive mesh*. Velký potenciál má také dělení uzavřených 3D modelů.

V rámci práce na aplikaci TriangleDivision jsem se seznámil s knihovnou Open Inventor, se kterou jsem do té doby ještě nepracoval.

Literatura

[1] Žára J., Beneš B., Felkel P. *Moderní počítačová grafika*. 2. vydání, Brno, Computer press 2004.

[2] Pečiva J. *Seriál Open Inventor*. http://www.root.cz/clanky/open-inventor/, 2003.

[3] Wernecke J. *The Inventor Mentor*. Addison-Wesley Longman Publishing Co., Inc., 1993.

[4] Vrba P. *Zobrazování krajiny a jeho optimalizace se zaměřením na LOD algoritmy.* FIT VUT, 2006. http://merlin.fit.vutbr.cz/upload/IvProjects/2005/chunkedLOD/chunkedLOD.pdf

[5] WWW stránky. *Dokumentace knihovny Coin3D.* http://www.coin3d.org/

[6] WWW stránky. *ROAM algoritmus*, https://graphics.llnl.gov/ROAM/roam.pdf

[7] Bartoň R. *ROAM algoritmus a analýza jeho výkonové náročnosti*, FIT VUT, 2006, http://merlin.fit.vutbr.cz/upload/IvProjects/2006/SoTerrain/SoTerrain.pdf

[8] Černý P. *Zobrazování krajiny a jeho optimalizace pomocí ROAM algoritmu*, FIT VUT, 2006, http://merlin.fit.vutbr.cz/upload/IvProjects/2006/ROAM/ROAM.pdf

[9] Ulrich T. *Rendering Massive Terrains using Chunked Level of Detail Control*, Oddworld Inhabitants, 2002, http://tulrich.com/geekstuff/sig-notes.pdf

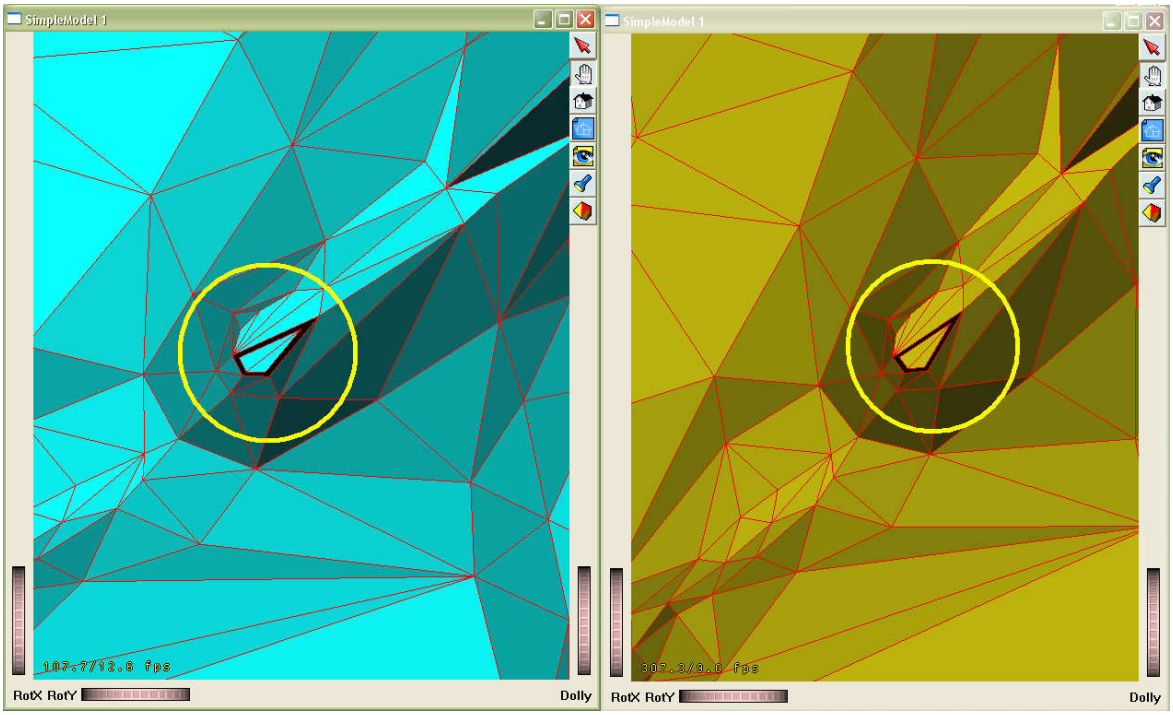
[10] Přibyl J. *Přednáška k předmětu PGP*, FIT VUT, https://www.fit.vutbr.cz/study/courses/PGP/private/lectures/prendaska_PGP_renderovani_krajiny.pdf

[11] Kršek P. *Přednáška k předmětu PGP*, Level of Detail, FIT VUT, 2006.

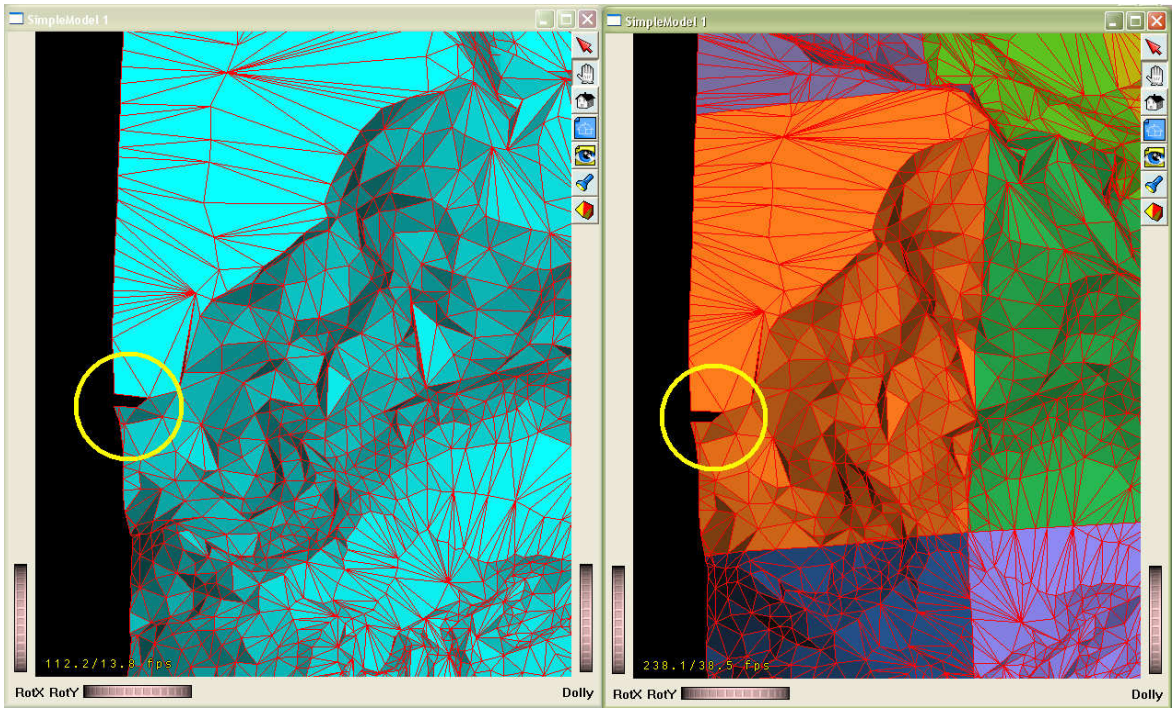
Seznam obrázků

Příloha A

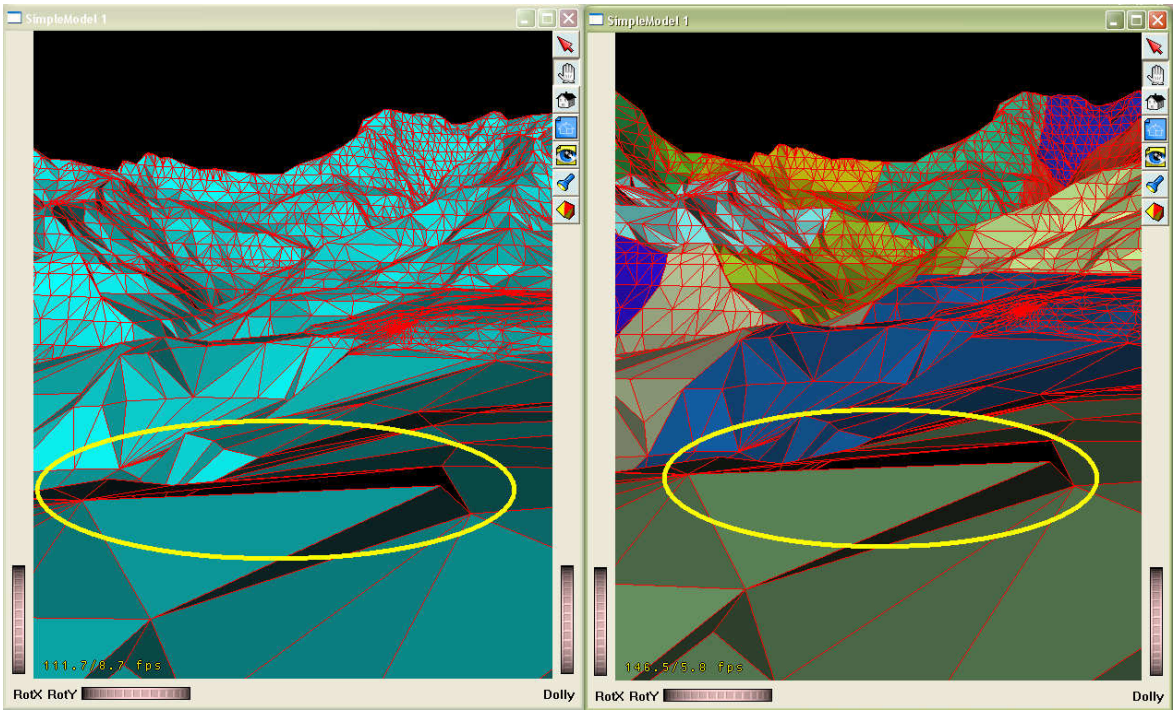
Chyby v původním modelu



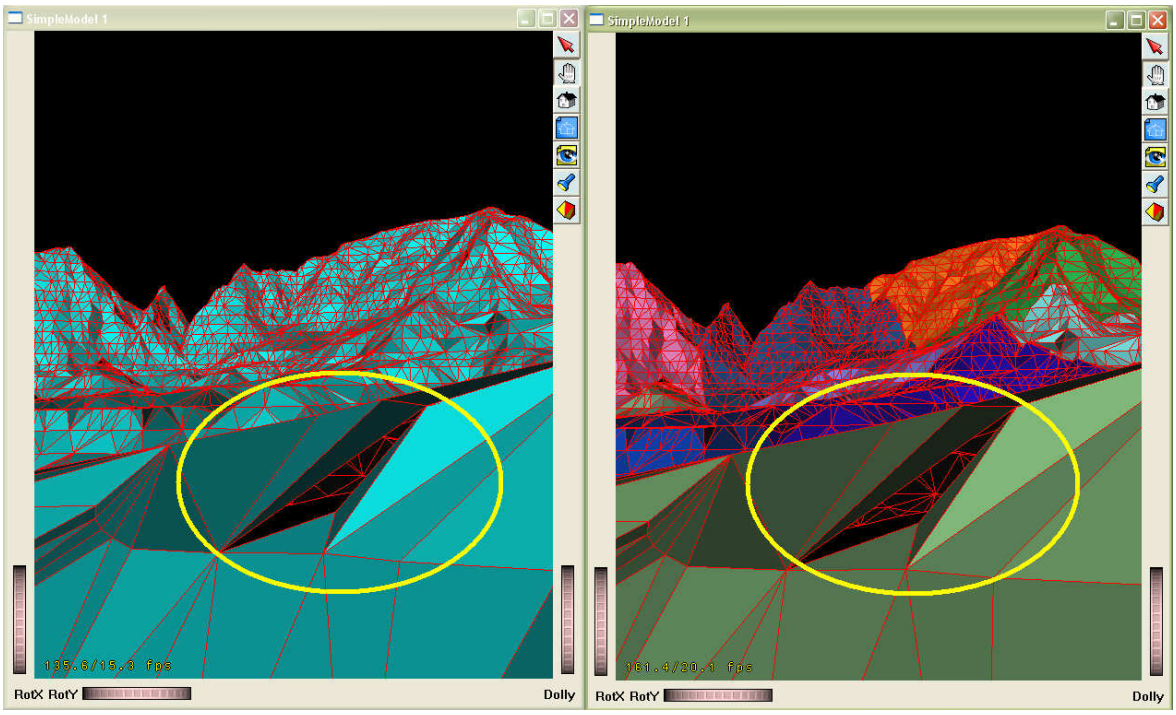
Obrázek A. 1: Chyby v původním modelu (vlevo původní model, vpravo zpracovaný)



Obrázek A. 2: Chyby v původním modelu (vlevo původní model, vpravo zpracovaný)



Obrázek A. 3: Chyby v původním modelu (vlevo původní model, vpravo zpracovaný)



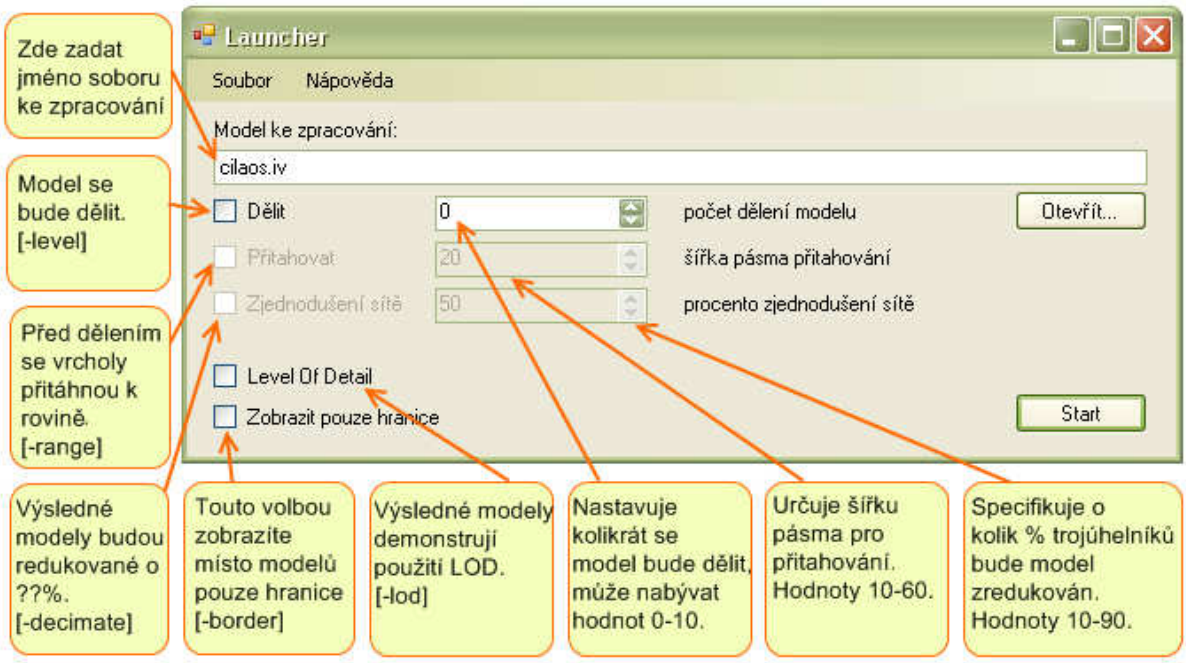
Obrázek A. 4: Chyby v původním modelu (vlevo původní model, vpravo zpracovaný)

Příloha B

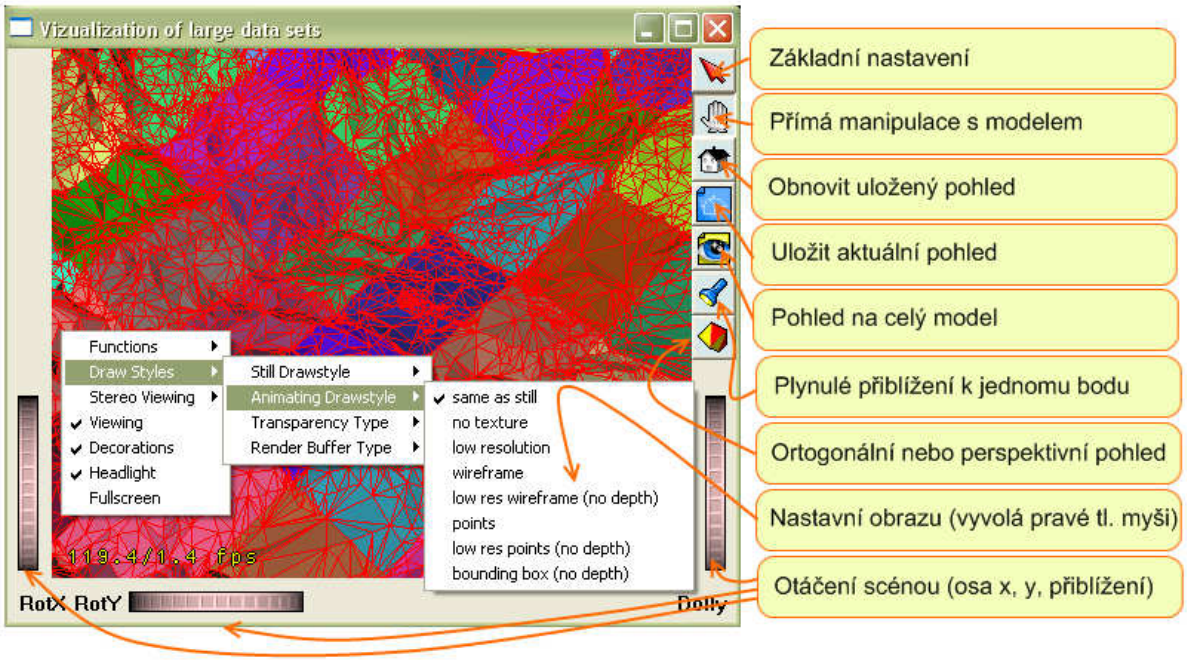
Popis parametrů příkazové řádky

Parametr	Implicitní hodnota	Význam
-file	cilaos.iv	Určuje, který *.iv soubor se má zpracovat.
-level	0	Udává do jaké hlouby se bude model dělit. Pracuje v rozsahu 0-10. Počet nově vzniklých modelů se pak rovná 2level.
-range	-1	Nastavuje šířku pásma pro přitahování vrcholů, kolem dělící roviny. Může nabývat hodnot 10.0-60.0.
-decimate	-1	Znamená, že se výsledné modely mají pouze zdecimovat a zobrazit. Rozsah hodnot 10-90.
-lod	---	Zadání tohoto parametru vytvoří a zobrazí z nadělených modelů Level Of Detail.
-border	---	Alternativa k parametrům –decimate a –lod. Umožňuje zobrazit pouze hraniční trojúhelníky.
		Pokud se nezadá žádný parametr příkazové řádky, vypíše se na obrazovku nápověda.

Ukázky aplikace



Obrázek B. 1: Ukázka zaváděcí aplikace s popisy jednotlivých ovládacích prvků



Obrázek B. 2: Ukázka vlastní aplikace s vysvětlením základních ovládacích prvků

Seznam příloh

Příloha 1. CD se zdrojovými kódy programu, programem zkompilovaným pro platformu Windows a textem diplomové práce.