Vysoké učení technické v Brně
Fakulta informačních technologií
Bakalářská práce

ROAM algoritmus a analýza jeho výkonové náročnosti

Radek Bartoň (xbarto33)

V Brně dne 23. dubna 2006

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod odborným vedením Ing. Jana Pečivy. Tímto bych mu chtěl poděkovat za nesčetné konzultace problémů souvisejících s projektem i za elektronickou korespondenci.

V závěru této práce jsem uvedl všechny literární prameny, publikace a internetové stránky, ze kterých jsem čerpal podklady pro vypracování této práce.

Podpis: .....

Abstrakt a klíčová slova

Abstrakt

Tento dokument pojednává o možnostech vizualizace terénu reprezentovaného pravidelnými výškovými mapami, výhodách a nevýhodách tohoto druhu reprezentace dat. Předkládá a vysvětluje některé známé algoritmy pro tuto vizualizaci, diskutuje jejich výkonnostní a vizuální vlastnosti z teoretického hlediska a přináší praktické poznatky z implementace algoritmů ROAM a Geo Mip–Mapping. Dále se snaží o zobecnění některých prvků napříč všemi algoritmy. Jsou zde vysvětleny termíny z oblasti vizualizace terénu, v úvodu je zahrnut stručný slovník těchto pojmů. Závěrečná část se zabývá testováním výkonnosti algoritmů a jejich součástí na reálné implementaci a diskuzí naměřených výsledků společně s jejich znázorněním v přehledných grafech. V úplném závěru dokumentu jsou nastíněny možnosti rozšíření dosavadní práce do budoucích projektů.

Klíčová slova

vizualizace terénu, úroveň detailů, Geo Mip–Mapping, ROAM, Chunked LoD, SOAR, CLoD, Diamond, výšková mapa, triangulace, binární strom, kvadrantový strom, dlaždice, prioritní fronta, profilování, optimalizace, OpenInventor

Abstract and Key Words

Abstract

This document deals with possibilities of the visualisation of regular grid height map terrains, advantages and disadvantages of this data representation approach. Offers and explains some well–known algorithms for terrain visualization. From theoretical aspect debate their performance and visual properties and brings practical experiences from ROAM and Geo Mip–Mapping implementation. Further it aims on generalization of some elements across all algorithms. There are terms from terrain visualisation domain explained here and there is dictionary of these terms in the beginning of document. End part occupies with algorithms and their parts performance testing on real implementation and with discussion of measured values together with their visualisation on easy to take in graphs. Possibilities of present work extension and options of future work are covered up in the absolute end of this document.

Key Words

terrain visualisation, level of detail, Geo Mip–Mapping, ROAM, Chunked LoD, SOAR, CLoD, Diamond, height map, triangulation, binary tree, quad tree, tile, priority queue, profiling, optimization, OpenInventor

Obsah

1 Úvod **6**
- 1.1 Slovník pojmů . . . 6
- 1.2 Slovo autora . . . 6

2 Algoritmy vizualizace terénu **8**
- 2.1 Real-time Optimally Adapting Meshes (ROAM) . . . 8
  - 2.1.1 Binární strom trojúhelníků . . . 8
  - 2.1.2 Fronta trojúhelníků na rozdělení . . . 10
  - 2.1.3 Fronta diamantů na spojení . . . 11
  - 2.1.4 Ořezávání pohledovým tělesem . . . 13
  - 2.1.5 Inkrementální generování trojúhelníkových pásů . . . 13
  - 2.1.6 Inkrementální výpočet chybové metriky . . . 13
  - 2.1.7 Shluky trojúhelníků . . . 13
  - 2.1.8 Nekonečný a editovatelný terén . . . 14
  - 2.1.9 Shrnutí . . . 14
- 2.2 Geo Mip–Mapping . . . 14
  - 2.2.1 Vytváření a organizace dlaždic . . . 14
  - 2.2.2 Kvadrantový strom dlaždic a ořezávání pohledovým tělesem . . . 16
  - 2.2.3 Napojování dlaždic . . . 17
  - 2.2.4 Nekonečný a editovatelný terén . . . 19
  - 2.2.5 Shrnutí . . . 19
- 2.3 Chunked LoD . . . 19
- 2.4 Diamond . . . 21
  - 2.4.1 Kvadrantový strom trojúhelníků . . . 21
  - 2.4.2 LIFO fronty . . . 22
  - 2.4.3 Napojování trojúhelníků . . . 22
  - 2.4.4 Shrnutí . . . 22
- 2.5 Další . . . 24

3 Společné vlastnosti algoritmů **25**
- 3.1 Chybové metriky . . . 25
- 3.2 Ořezávání pohledovým tělesem . . . 26
- 3.3 Test potenciální viditelnosti . . . 27
- 3.4 Geomorphing . . . 27
- 3.5 Dynamické osvětlení terénu . . . 29

4 Implementace algoritmů ROAM a Geo Mip–Mapping **30**
4.1 Algoritmus ROAM 30
4.2 Algoritmus Geo Mip–Mapping 31
4.3 Profiler 31

5 Výsledky testování **35**
5.1 Jak se testovalo 35
5.2 Algoritmus ROAM 36
5.2.1 Výkonové nároky 36
5.2.2 Paměťové nároky 36
5.3 Algoritmus Geo Mip–Mapping 37
5.3.1 Výkonové nároky 37
5.3.2 Paměťové nároky 37
5.4 Shrnutí 39

6 Závěr **41**

A Slovník pojmů **42**

B Ovládání testovací aplikace **45**

C Obsah přiloženého CD **47**

D Grafy výsledků testování **49**

Kapitola 1

Úvod

1.1 Slovník pojmů

Již v úvodu tohoto dokumentu se čtenář setká s celou řadou výrazů a termínů z oblasti vizualizace terénu, se kterými nemusí být obeznámen. Pro tento případ je v dodatku A uveden slovník pojmů, který se snaží o stručné vysvětlení a případně anglický překlad termínů uváděných v textu. Vzhledem k tomu, že většina anglických slov nebo slovních spojení z této oblasti nemá žádný ustálený český ekvivalent, jsou některé překlady neoficiální a mají pouze informativní charakter.

1.2 Slovo autora

V mnoha implementacích virtuální reality hraje terén jednu z klíčových úloh. Ať už na něm svádí virtuální pěšáci líté boje v některé z 3D akčních her, prohánějí se po něm tanky a jiná bojová technika v bitevních simulátorech nebo skýtá nádherné pohledy z výšky v leteckých simulátorech a geografických informačních systémech. Pokud jde v těchto aplikacích o dostatečnou realističnost, je třeba zajistit zobrazování co největšího území s co největší vizuální přesností. Na druhou stranu je také více než důležité dodržet interaktivní počet snímků za sekundu, tedy dostatečně plynulé zobrazování.

Oba tyto požadavky si však vzájemně protiřečí a zlepšování jednoho aspektu je vždy na úkor druhého. Ačkoliv toto platí bezesporu pro vykreslování terénu hrubou silou, existuje celá řada algoritmů, které se snaží tuto nepřímou závislost porušit.

Další zvýšení realističnosti lze také docílit použitím výškových dat získaných buď pozemním měřením nebo satelitním snímkováním ze skutečné krajiny. Tyto data bývají zpravidla uloženy ve formě výškových map, tj. pravidelných matic hodnot udávajících nadmořskou výšku v každém bodě. Také uměle vytvořené terény se velmi často generují ve formě pravidelných sítí bodů, a tak se tento dokument bude zabývat výhradně těmi algoritmy, které pracují právě s pravidelnými výškovými mapami.

Kromě pravidelných výškových map se pro popis terénu používají i nepravidelné trojúhelníkové sítě tzv. TIN, které lze získat buď ručním modelováním v některém z modelovacích nástrojů nebo konverzí z pravidelných výškových map. Pro zobrazování těchto terénů se používají speciální algoritmy, o nichž nebude v tomto dokumentu řeč. Stručný soupis výhod a nevýhod obou těchto přístupů shrnují následující odstavce.

Výhody pravidelných výškových map

- Snadné získávání - Jak již bylo řečeno výše, lze výškové mapy získávat automaticky za pomocí satelitního snímkování či radaru.
- Snadné ukládání a načítání - Data jsou jednoduše uložena jako matice bodů, tedy stačí ukládat pouze informaci o nadmořské výšce. Informace o poloze bodu s touto nadmořskou výškou je dána topologií. Dále je možné data ukládat jako obyčejné obrázky ve formátu bmp, png nebo jiném, který nevyužívá ztrátovou kompresi, a využít pro jejich načítaní některou již napsanou knihovnu.
- Jednoduchost zobrazování - Díky pravidelnosti načtené výškové mapy ji lze pro potřeby vizualizačního algoritmu hierarchicky rozdělit do bloků čtvercového, trojúhelníkového nebo jiného tvaru, což sebou přináší možnosti optimalizace vykreslování.

Nevýhody pravidelných výškových map

- Ohromné množství dat - Výšková mapa pokrývající větší území při dostatečné hustotě výškových bodů zabírá v paměti počítače stovky megabytů až jednotky gigabytů, což nese značné nároky na systémové prostředky počítače.
- Převisy a jeskyně v terénu - Vzhledem k tomu, že terén je reprezentován pouze maticí bodů, nelze pracovat s terénem, který obsahuje různé převisy a jeskyně.

Výhody nepravidelných trojúhelníkových sítí

- Kompaktnost uložení - Nepravidelné trojúhelníkové sítě zabírají mnohem méně místa externí v paměti počítače, neboť obsahují pouze tolik informací, kolik je třeba pro zobrazení daného území s danou úrovní detailů.
- Minimální triangulace - Některé algoritmy pro zobrazování pravidelných výškových map se snaží o zobrazení takové triangulace, která by reflektovala míru členitosti terénu. Toto je u nepravidelných trojúhelníkových sítí většinou dosazeno už po načtení dat.

Nevýhody nepravidelných trojúhelníkových sítí

- Vektorový popis - Nepravidelné trojúhelníkové sítě jsou uloženy ve formě obecných trojúhelníků, bodů a hran, a tudíž je s touto reprezentací poněkud složitější manipulace než s pouhou maticí bodů.

Kapitola 2

Algoritmy vizualizace terénu

V této kapitole se zaměříme na popis jednotlivých algoritmů pro vizualizaci terénu pracujících s pravidelnými výškovými mapami. Původní popis těchto algoritmů byl publikován v příslušných dokumentech uvedených v seznamu použité literatury, na něž je podle potřeby odkazováno. Zde je tento popis prezentován pokud možno co nejsrozumitelnější formou a rozšířen o některé praktické zkušenosti z implementace.

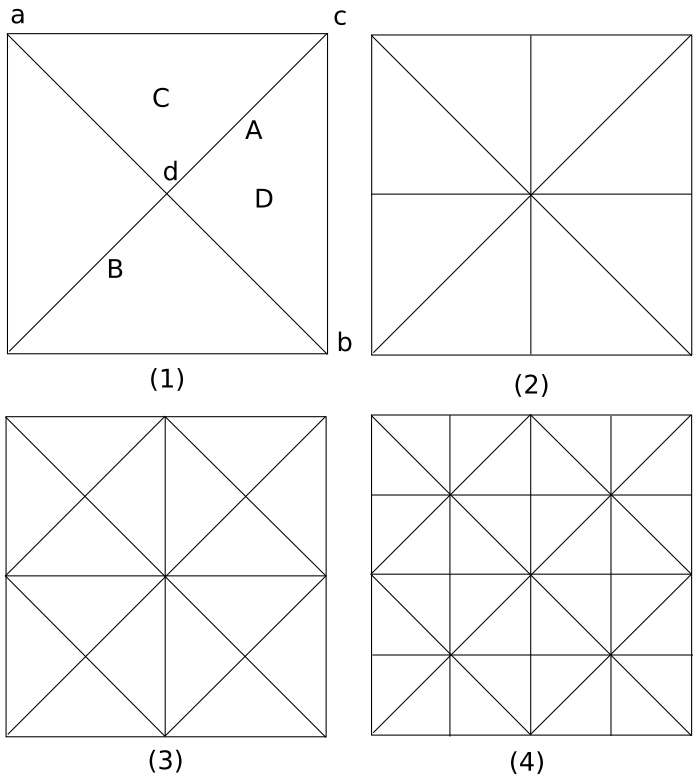
2.1 Real-time Optimally Adapting Meshes (ROAM)

V roce 1997 byl týmem kolem Marka Duchaineau vyvinut algoritmus nazvaný Real-time Optimally Adapting Meshes. Ačkoliv je možno ho využít i při zobrazování obecných manifold objektů, byl primárně navržen pro vizualizaci terénu s důrazem na možnost dodržování striktního počtu snímků za sekundu a minimální triangulaci. Souhrn této práce byl uveden v dokumentu [1].

Po dlouhou dobu se stal standardem v této oblasti a spousta vývojářů her jej použila ve svých dílech. Dnes však začíná být vytlačován jednoduššími algoritmy jako je Chunked LoD a Geo Mip–Mapping, neboť ty dosahují na dnešním hardware lepších výsledků. Nicméně v roce 2000 rozšířil Alex A. Pomeranz ve své disertační práci [2] algoritmus ROAM o zajímavou myšlenku a nový algoritmus nazval jménem RUSTiC. Ve fázi rozpracování je implementace ROAM 2.0, která by mohla vrátit algoritmus ROAM zpátky do popředí.

2.1.1 Binární strom trojúhelníků

Základní reprezentací výškové mapy pro tento algoritmus je binární strom trojúhelníků $\mathbf{B}$, který se vytvoří ve fázi načítání dat. Rozdělíme-li celou výškovou mapu na dva pravoúhlé trojúhelníky A a B, vznikne nám první úroveň tohoto binárního stromu. Kořen stromu zadefinujeme jako virtuální trojúhelník R na nulté úrovni, který má za levého a pravého potomka oba trojúhelníky z první úrovně. Další úrovně nám pak vzniknou postupným rekurzivním dělením trojúhelníků spojením vrcholu s pravým úhlem (vrchol c trojúhelníku A) a bodu v polovině přepony (bod d trojúhelníku A) novou hranou na levé a pravé potomky C a D. Několik kroků tohoto dělení lze vidět na obrázku 2.1.



Obrázek 2.1: Vytvoření binárního stromu trojúhelníků. (1) – Dva potomci kořenového trojúhelníku R tvoří první úroveň binárního stromu trojúhelníků. (2) – Trojúhelníky na druhé úrovni. (3) – Trojúhelníky na třetí úrovni. (4) – Trojúhelníky na čtvrté úrovni.

Z tohoto přístupu plyne několik závěrů:

1. Vstupní výšková mapa musí být čtvercových rozměrů o velikosti strany:

$$h = 2^n + 1 \tag{2.1}$$

   kde n je celé kladné číslo nebo 0.

2. Počet úrovní binárního stromu trojúhelníků je:

$$l = 2 \cdot \log_2(h - 1) \tag{2.2}$$

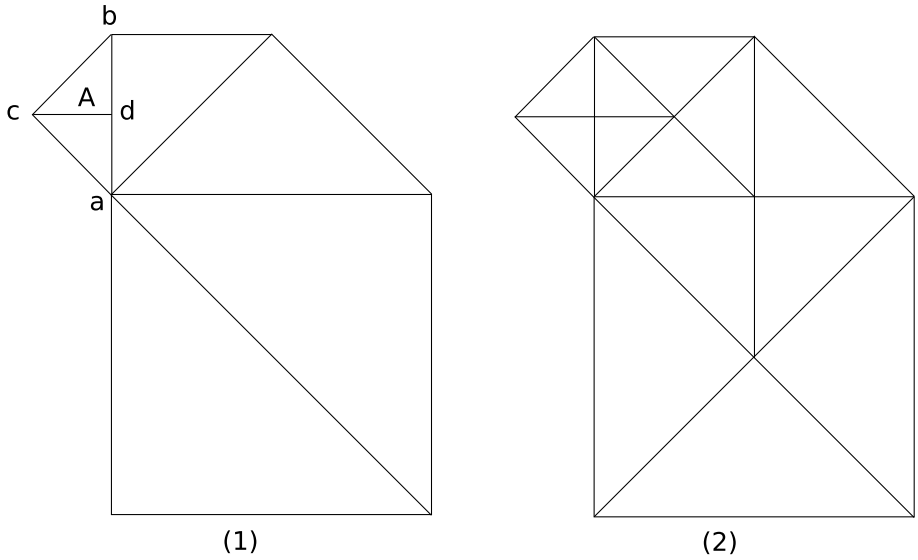
   kde h je velikost strany výškové mapy.

3. Počet trojúhelníků v binárním stromu je

$$|\mathbf{B}| = 2^l - 1 \tag{2.3}$$

   kde l je počet úrovní binárního stromu trojúhelníků (včetně nulté úrovně).

Při této inicializaci se zároveň vypočítá i statická část chybové metriky. Vzhledem k tomu, že algoritmus ROAM je navržen pro použití libovolné monotónní metriky, budou možnosti tohoto výpočtu diskutovány ve společné sekci pro všechny algoritmy v kapitole 3.1.



Obrázek 2.2: Vznik rekurzivního dělení při dělení rodičovského trojúhelníku A (1). Abychom mohli tento trojúhelník rozdělit, musíme nejdříve provést dělení všech ostatních trojúhelníků (2).

2.1.2 Fronta trojúhelníků na rozdělení

V dynamické fázi, tedy v průběhu vykreslování každého snímku hraje hlavní úlohu prioritní fronta trojúhelníků na rozdělení. Do ní se vkládají ty trojúhelníky, které se mají ve výsledné triangulaci zobrazit. Jsou seřazeny podle chyby $p(T)$, které se dopouštíme tím, že nezobrazíme místo nich jejich dva potomky. Možností jak vypočítat prioritu je spousta, jediné, co je třeba dodržet aby byla výsledná triangulace minimální, je její monotónnost, tzn. že musí platit vztah:

$$p(T_p) \geq max\{p(T_a), p(T_b)\} \; \forall \; T \in \mathbf{B} \tag{2.4}$$

kde T_p je rodičovský trojúhelník, T_a a T_b jsou jeho potomci a $\mathbf{B}$ je binární strom trojúhelníků.

Použije-li se jenom fronta trojúhelníků na spojení, tak se na začátku každého snímku vyprázdní a vloží se do ní pouze potomci kořenového trojúhelníku R, trojúhelníky A a B. Vložením trojúhelníku do fronty se myslí jeho vložení na správné místo podle jeho priority vypočtené pomocí dynamické části chybové metriky v závislosti na natočení a vzdálenosti kamery vůči vkládanému trojúhelníku. Následně se z fronty vybírají ty trojúhelníky, jejichž priorita a tedy i chyba je největší, rozdělí se a zpět se vloží jejich potomci. Při tomto dělení je nutno ošetřit případ, kdy rozdělením trojúhelníku A vznikne trhlina v triangulaci. V tom případě je nutno provést další rekurzivní dělení. Příklad takovéto situace je znázorněn na obrázku 2.2. Aby to bylo možno provést, je třeba mít uloženu informaci o sousednosti. V praxi lze tohoto docílit tím, že trojúhelníky ve frontě mají navíc oproti trojúhelníkům v binárním stromu tři položky s ukazateli na bázového, levého a pravého souseda. Obecně platí, že v triangulaci spolu sousedí pouze trojúhelníky stejné, o jedna menší nebo o jedna větší úrovně.

Rozdělování trojúhelníků probíhá dokud není dosaženo požadovaného počtu trojúhelníků, nevyprší čas vyhrazený na tuto činnost nebo dokud nejvyšší priorita ve frontě není pod ur-

```
split_queue.clear();
split_queue.add(root->left);
split_queue.add(root->right);

/* Opakuj, dokud není dosaženo požadovaného počtu trojúhelníku, nevypršel čas
vyhrazený na generování triangulace nebo není dosaženo minimální triangulace
podle nastavené priority. */
while ((split_queue.size() < count_max) && (time < time_max) &&
  split_queue.get_max().priority > min_priority))
{
  force_split(split_queue.get_max());
}
```

Kód 2.3: Vykreslovací smyčka při použití fronty trojúhelníků na rozdělení.

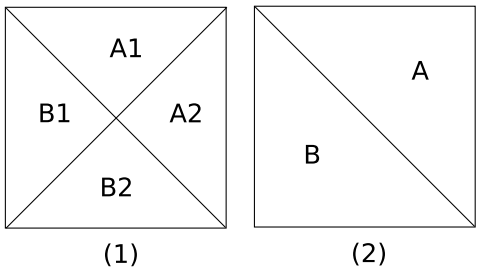
čitou hranicí, což znamená, že triangulace má požadovanou přesnost. Vše by se dalo shrnout pseudo C++ kódem 2.3, který se bude opakovat pro každý snímek. Funkce `force_split` provede vyjmutí trojúhelníku z fronty, případné rekurzivní rozdělení sousedů s jejich vyjmutím a vložením všech nově vzniklých potomků. Při použití fronty diamantů na spojení, o níž bude řeč v následující sekci, se toto děje pouze u prvního snímku.

2.1.3 Fronta diamantů na spojení

Zavedeme-li další frontu, do níž budeme tentokrát vkládat všechny diamanty, které se nám v triangulaci vytvoří, dostaneme možnost využívat výsledky výpočtu triangulace z předchozích snímků. Tím docílíme, že časová náročnost algoritmu nebude závislá na velikosti vstupní výškové mapy, ale na počtu zobrazovaných trojúhelníků a velikosti změny triangulace, tzn. rychlosti pohybu a otáčení kamery. Příklad toho, jak takový diamant v triangulaci vypadá, lze vidět na obrázku 2.4. V zobrazovací s myčce se v prvním snímku provede naplnění obou front obdobně jako v kódu 2.3, ale další snímky se budou zpracovávat již inkrementálně. Při vkládání diamantů do fronty se vypočte priorita diamantu jako maximum z priorit všech jeho čtyř trojúhelníků. Zavede se nová funkce `merge`, která spojí čtyři trojúhelníky diamantu do dvou rodičovských a zjistí a případně vloží do fronty na spojení nově vzniklé diamanty. Funkce `force_split` se rozšíří o detekci vzniku diamantů při rozdělování trojúhelníků a jejich případné vložení do fronty na spojení.

Ve vlastní smyčce se nejprve přepočítají priority trojúhelníků a diamantů ve frontách s ohledem na polohu kamery v novém snímku a následně se opakuje proces spojování a rozdělování trojúhelníků, dokud není splněna některá z podmínek uvedených v předchozí sekci a zároveň je dosaženo minimální triangulace. Toto poznáme tehdy, když minimální priorita ve frontě diamantů na spojení dosáhne hodnoty maximální priority ve frontě trojúhelníků na rozdělení. O tom, zda se má přistoupit ke spojování trojúhelníků nebo jejich rozdělování, rozhodne některé z kritérií, jakým je např. aktuální počet trojúhelníků v triangulaci nebo nejvyšší hodnota priority ve frontě trojúhelníků na rozdělení. Zápis tohoto postupu v pseudo kódu lze vidět v kódu 2.5.

Počet rozdělení a spojení trojúhelníků mezi dvěma snímky je roven počtu rozdílných trojúhelníků v obou minimálních triangulacích. Dá se dokázat (viz. [1]), že maximum tohoto



Obrázek 2.4: Sloučení diamantu v triangulaci na dva trojúhelníky. (1) – Čtyři trojúhelníky diamantu. (2) – Dva rodičovské trojúhelníky po sloučení.

```
if (first_run)
{
  split_queue.clear();
  merge_queue.clear();
  split_queue.add(root->left);
  split_queue.add(root->right);
}
else
{
  split_queue.recompute();
  merge_queue.recompute();
}

/* Opakuj, dokud není dosaženo požadovaného počtu trojúhelníku, nevypršel čas
vyhrazený na generování triangulace nebo není dosaženo minimální triangulace
podle nastavené priority. */
while ((split_queue.get_max().priority > merge_queue.get_min().priority) ||
  ((split_queue.size() < count_max) && (time < time_max) &&
  (split_queue.get_max().priority > min_priority)))
{
  if (split_queue.size() < count_max)
  {
    force_split(split_queue.get_max());
  }
  else
  {
    merge(merge_queue.get_min());
  }
}
```

Kód 2.5: Vykreslovací smyčka při použití fronty trojúhelníků na rozdělení a fronty diamantů na spojení.

počtu je rovno součtu počtu trojúhelníků v triangulacích v obou snímcích. V praxi je však počet rozdílných trojúhelníků ve dvou po sobě jdoucích snímcích velmi malý. Situace, kdy tomu tak není, např. v případě náhlého skoku kamery na jiné místo, lze snadno detekovat a vrátit algoritmus do stejného stavu jako v prvním snímku.

2.1.4 Ořezávání pohledovým tělesem

Tak jako v jiných algoritmech, můžeme docílit velkého nárůstu výkonu tím, že nebudeme vykreslovat ty části terénu, které jsou mimo obrazovku. Ačkoliv v algoritmu ROAM nemusíme vykreslovat trojúhelníky mimo záběr kamery vůbec, stejného ba dokonce lepšího výsledku docílíme, když v průběhu výpočtu dynamické části chybové metriky nastavíme trojúhelníkům mimo pohledové těleso imaginární nulovou hodnotu. Tyto trojúhelníky se ve fázi dělení a spojování nikdy nerozdělí, pokud k tomu nebudou donuceny sousedy, a nebudou mít vliv na spojování diamantů, jichž budou členem. Za pohledem kamery budou tedy trojúhelníky vždy o maximální velikosti vzhledem k vynucenému dělení zabraňujícímu vzniku trhlin. Metody detekce trojúhelníků mimo pohledové těleso jsou uvedeny v sekci 3.2.

2.1.5 Inkrementální generování trojúhelníkových pásů

Další možností, jak docílit vyššího počtu snímků za sekundu, je zobrazování triangulace za pomocí trojúhelníkových pásů. Během fáze rozdělování a spojování trojúhelníků vytváříme datové struktury reprezentující jednotlivé pásy. Při rozdělení trojúhelníku se pás rozdělí na dva nebo prodlouží, při spojení se snažíme nově vložený trojúhelník napojit na některý ze sousedních pásů nebo pás obsahující potomky spojeného trojúhelníku zkrátit. Toto se děje pouze při vlastním rozdělení nebo spojení trojúhelníků, a tudíž inkrementálně, což nese jen minimální požadavky na výpočet.

2.1.6 Inkrementální výpočet chybové metriky

Nyní je výpočetně nejnáročnější částí algoritmu přepočet priorit trojúhelníků a diamantů ve frontách. Pokud však zadefinujeme prioritu minimální triangulace p_T jako hodnotu maximální priority v prioritní frontě v době dokončení výpočtu triangulace a zjistíme jak rychle, vzhledem k rychlosti pohybu kamery, se tato hodnota mění, můžeme vytvořit seznam trojúhelníků, u nichž můžeme výpočet priority odložit na dobu (snímek), kdy by měla tato priorita překročit očekávanou hodnotu priority minimální triangulace p_T. Pak pouze přepočítáváme priority těch trojúhelníků, které jsou v seznamu a mají očekávanou dobu překročení shodnou s dobou aktuálního snímku.

2.1.7 Shluky trojúhelníků

V době, kdy vznikl algoritmus ROAM, poskytoval grafický hardware mnohem menších výkonů. Chceme-li dnes do grafické karty poslat dostatečné množství trojúhelníků, které ještě zvládne vykreslit, naroste zátěž CPU nad snesitelnou hranici. Tento neduh se snaží kompenzovat algoritmus RUSTiC (viz. [2]), který nahrazuje každý trojúhelník algoritmu ROAM shlukem o definovaném počtu trojúhelníků. Tím se přibližuje modernějším algoritmům jako je např. Geo Mip–Mapping (viz. 2.2), které provádějí výběr bloků na určité úrovni detailů, ale ty pak vykreslují hrubou silou. Hlavním problémem tohoto rozšíření je nutnost zajištění návaznosti shluků a to i patřících do sousedních úrovní binárního stromu trojúhelníků.

2.1.8 Nekonečný a editovatelný terén

Pokud potřebujeme zobrazovat terén obrovských rozměrů, na který nestačí paměť počítače, musíme zajistit plynulé načítání a zobrazování jeho viditelných částí. Jeden kořen binárního stromu trojúhelníků proto nahradíme více kořeny, které budeme zpracovávat samostatně, a do sousedních stromů pouze propagujeme vynucené dělení trojúhelníků. V případě načtení nového bloku buď vrátíme algoritmus do stavu v prvním snímku a vytvoříme triangulaci dělením kořenů nebo projdeme hranici nově přidaného bloku a provedeme jeho dělení dokud nebudou hranice pasovat.

Při editaci terénu, např. výbuchem nebo zbrázděním koly nějakého vozidla, musíme přepočítat statickou část priority těch trojúhelníků, které byly úpravou ovlivněny, a tuto změnu propagovat směrem ke kořeni stromu. Přitom se může tato propagace někde v průběhu cesty zastavit v závislosti na použité metrice. Tato změna je tedy lokální, a tudíž není příliš výpočetně náročná.

2.1.9 Shrnutí

Přestože je algoritmus ROAM ve své původní podobě uvedené v [1] již zastaralý, poskytuje vlastnosti, které jsou důležité ve speciálních aplikacích, a tudíž u nich najde uplatnění i dnes. Po implementaci všech zde uvedených optimalizací, však konkuruje i modernějším algoritmům.

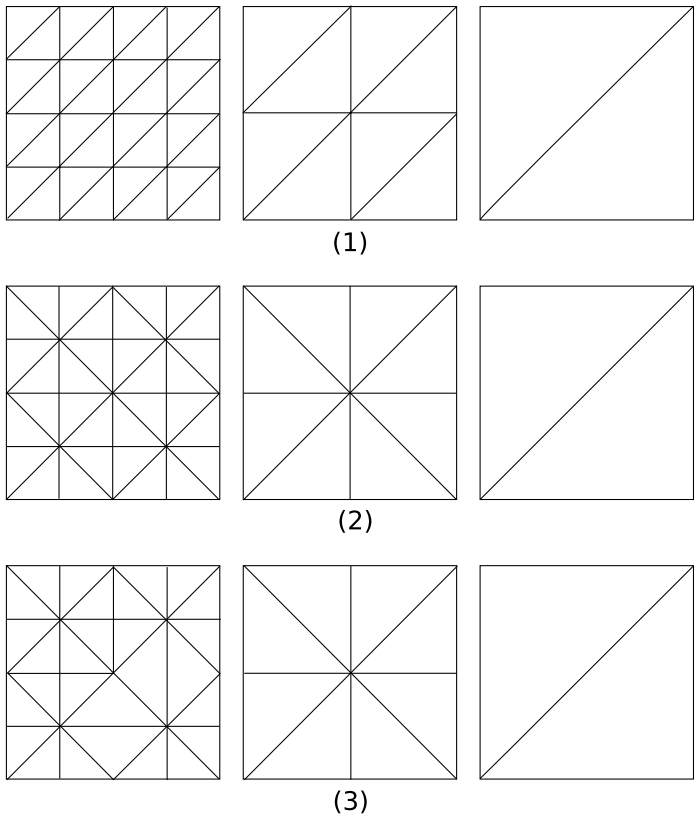
2.2 Geo Mip–Mapping

Dokumenty [3], [4] a [5] ukazují přístup, kdy terén je rozdělen na sadu pravidelných dlaždic uložených v paměti ve více úrovních detailů. Název Geo Mip–Mapping tento algoritmus dostal proto, že představuje jakousi geometrickou analogii k texturovému Mip–Mappingu a jeho geomorphing analogii k bilineárnímu filtrování textur. Tento způsob vizualizace terénu čerpá ze schopnosti dnešních grafických karet zobrazit relativně velké množství trojúhelníků v krátkém čase jen s minimální režií počítanou v CPU. Použití technik jakými jsou display-listy, pole vrcholů a Vertex Buffer Object tento počet ještě zvyšují.

2.2.1 Vytváření a organizace dlaždic

Při načítání dat se ze vstupní výškové mapy vytvoří dlaždice různých úrovní detailů, ze kterých pak budeme podle dané metriky při zobrazování terénu vybírat. Trojúhelníky v dlaždici mohou být organizovány dvěma způsoby, jak ukazuje obrázek 2.6. Buď vytvoříme triangulaci dlaždice ze čtverců dělených stejnolehlými úhlopříčkami (1) nebo tzv. dělením nejdelší hrany (2). První způsob je jednodušší a umožňuje vykreslování dlaždic za pomocí čtyřúhelníkových pásů, nicméně v neotexturovaném terénu může Gourandovo stínování generovat hranaté přechody stínů. U druhého způsobu není tento efekt tak markantní a triangulaci lze adaptivně zjednodušit při zachování dělení okrajových hran (3). Zvolené dělení trojúhelníků má také vliv na způsob napojování dlaždic různých úrovní detailů vedle sebe, jak bude uvedeno dále.

Během vytváření dlaždic, je také vhodné vypočítat jejich ohraničení, z důvodu výpočtu ořezávání pohledovým tělesem, a statickou část chybové metriky. O možnostech výpočtu chybové metriky bude pojednáno v sekci 3.1. Zde se pouze zmíníme o tom, že pracovat



Obrázek 2.6: Několik úrovní detailů dlaždice a jejich dělení. (1) – Dělení pomocí čtverců se stejnolehlými úhlopříčkami. (2) – Dělení půlením nejdelší hrany trojúhelníků. (3) – Dělení půlením nejdelší hrany trojúhelníků s adaptivním zjednodušením.

s chybou každého trojúhelníku během dynamického výpočtu chybové metriky by bylo časově velmi náročné, a tak se i statická část počítá pro celou dlaždici jako maximum nebo průměr z chyb jednotlivých trojúhelníků.

2.2.2 Kvadrantový strom dlaždic a ořezávání pohledovým tělesem

Z dlaždic vytvoříme nejlépe rekurzivním způsobem kvadrantový strom **T**, přičemž každé dlaždici na vyšší úrovní stromu přiřadíme ohraničení vzniklé sloučením ohraničení jejích potomků. Tato organizace slouží čistě pro zajištění snadného vyloučení dlaždic mimo pohled kamery z dalšího zpracovávání s logaritmickou složitostí. Za běhu to vypadá následovně: Začneme v kořeni stromu, kde provedeme test na viditelnost s použitím jeho ohraničení. Je-li otec v pohledu kamery pokračujeme v rekurzi pro všechny potomky, dokud se nedostaneme ke dlaždici na nejnižší úrovni stromu. Zde se podle pozice kamery a použité metriky rozhodneme, kterou z úrovní detailů dlaždice použijeme pro vykreslování.

O kvadrantovém stromu a jeho dlaždicích platí:

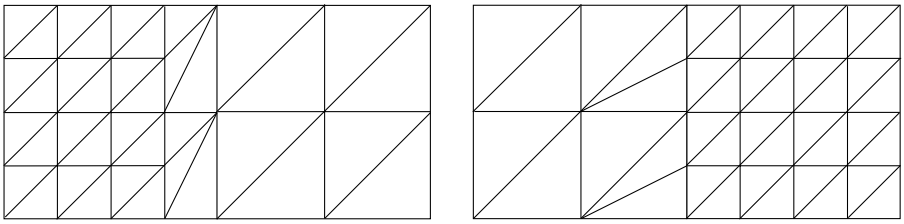
1. Je dobré, nicméně to není podmínkou, aby vstupní výšková mapa měla čtvercové rozměry o velikosti strany:
$$h = 2^n + 1 \tag{2.5}$$
kde n je celé kladné číslo nebo 0.

2. Samotné dlaždice však už z důvodu jejich dělení musí mít čtvercové rozměry o velikosti strany:
$$h_t = 2^m + 1 \tag{2.6}$$
kde m je celé kladné číslo nebo 0.

3. Počet úrovní detailů jedné dlaždice je pak:
$$l = m + 1 \tag{2.7}$$
kde m je číslo použité při výpočtu rozměru dlaždice.

4. Splňuje-li výšková mapa podmínku z rovnice (2.5), dá se počet dlaždic v kvadrantovém stromu dlaždic určit vztahem:
$$|\mathbf{T}| = (\frac{h}{h_t})^2 \tag{2.8}$$
kde h je velikost strany vstupní výškové mapy a t_t je velikost strany jedné dlaždice.

5. Výška stromu vybudovaného nad tímto počtem dlaždic pak bude:
$$l_t = \log_4 |\mathbf{T}| + 1 \tag{2.9}$$
kde $|\mathbf{T}|$ je počet dlaždic v zobrazovaném terénu.



Obrázek 2.7: Napojení dlaždic pomocí trojúhelníkových vějířů. (1) – Změna geometrie dlaždice s vyšší úrovní detailů. (2) – Změna geometrie dlaždice s nižší úrovní detailů.

2.2.3 Napojování dlaždic

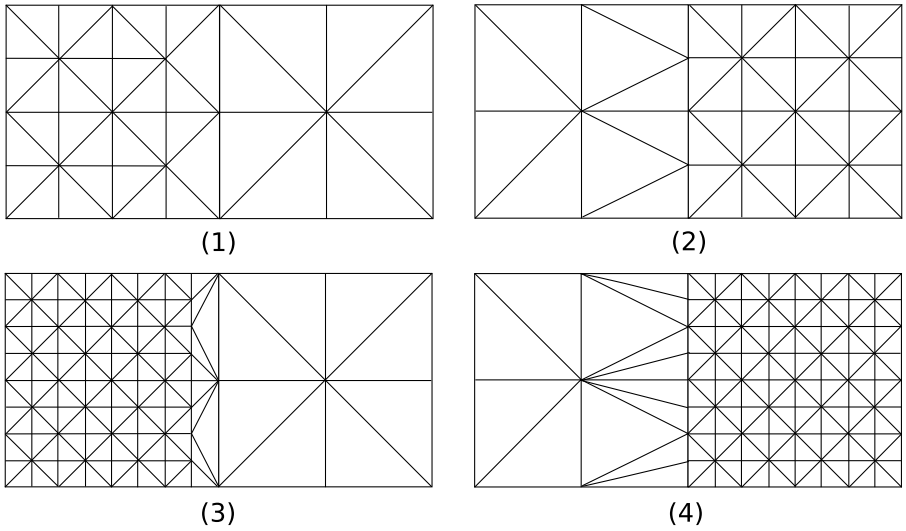
Celkově by byl algoritmus velice jednoduchý, kdyby nevznikaly napojováním sousedních dlaždic různých úrovní detailů trhliny v terénu. Tato nepříjemná situace se musí oproti algoritmu ROAM explicitně řešit ve fázi vykreslování trojúhelníků dlaždice. Obecně lze spojit dlaždice, jejichž úroveň detailů se liší o více než jednu, nicméně mnoho implementací algoritmu spojování dlaždic omezuje právě na podmínku, že rozdíl úrovní dlaždic je maximálně jedna. Způsobů, jak dlaždice spojit, je vícero a my si zde ukážeme několik z nich.

Obrázek 2.7 ukazuje první způsob, který se hodí převážně pro dlaždice ze čtverců rozdělených stejnolehlými úhlopříčkami. Za použití trojúhelníkových vějířů se jedna z dlaždic přizpůsobí druhé, přitom je to buď ta s vyšší úrovní detailů (1) nebo ta s nižší úrovní detailů (2). Navíc zde existují dva způsoby určení, která dlaždice se má přizpůsobit které. Trojúhelníkové vějíře můžeme generovat vždy v rámci např. dolního a pravého okraje každé dlaždice, která na těchto okrajích sousedí s dlaždicí jiné úrovně detailů, nebo řekneme, že přizpůsobovat se bude vždy ta dlaždice, jejíž úroveň je např. větší.

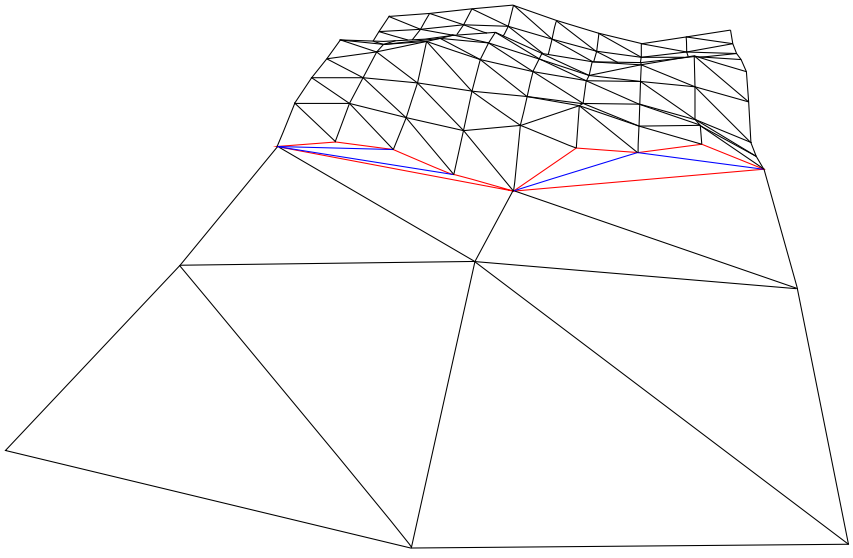
Druhý způsob vhodný pro dlaždice rozdělené půlením nejdelší hrany trojúhelníků je vidět na obrázku 2.8. Zde se dlaždice spojí segmenty trojúhelníků připomínajících písmeno V, přičemž z hlediska implementace je jednodušší, omezí-li se napojování pouze na úpravu dlaždic s vyšší úrovní detailů (2, 4), neboť v opačném případě (1, 3) vniknou komplikace tam, kde dlaždice s nižší úrovní detailů sousedí s více dlaždicemi s vyšší úrovní.

Dalším způsobem je prosté vyplnění trhlin v terénu polygonem nebo několika trojúhelníky, jak ukazuje obrázek 2.9. Červené hrany na tomto obrázku představují okraje vzniklé trhliny a modré hrany patří výplňkovým trojúhelníkům. Zde je třeba dávat pozor na konvexní tvar okraje trhliny, který potřebuje zvláštní ošetření vyplnění (pravá část obrázku). Konkávní okraje lze vyplnit za pomocí trojúhelníkových vějířů. Tak jako tak při tomto způsobu vyplňování vznikají v terénu velice úzké trojúhelníky, což kvůli zaokrouhlovacím chybám způsobuje vizuální nepřesnosti. Navíc ani Gourandovo stínování v takto doplněné triangulaci nevypadá dobře.

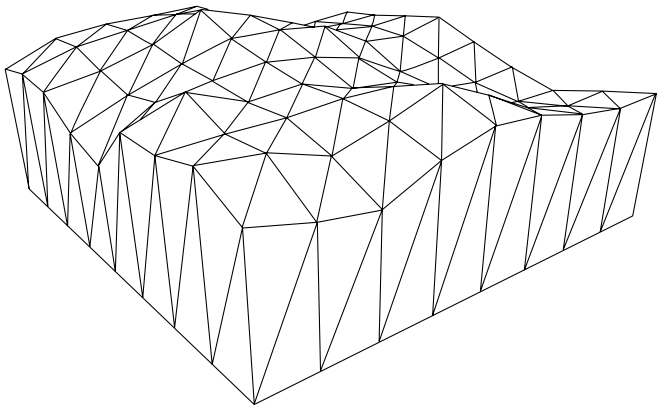
Posledním způsobem uvedeným v tomto dokumentu je poněkud bizarní řešení uvedené v práci [6]. Jak je vidět na obrázku 2.10 vytvoříme z okrajů dlaždice svislé stěny a při těsném sousedství spojovaných dlaždic nám toto zajistí, že nebude vidět pod terén. Výhodou je, že vše lze připravit již v průběhu načítání dlaždic, nevýhodou jsou opět chyby při výpočtu Gourandova stínování.



Obrázek 2.8: Spojení dlaždic segmenty trojúhelníků připomínajících písmeno V. (1) – Změna geometrie dlaždice s úrovní detailů o jedna větší. (2) – Změna geometrie dlaždice s úrovní detailů o jedna menší. (3) – Změna geometrie dlaždice s úrovní detailů o dvě větší. (4) – Změna geometrie dlaždice s úrovní detailů o dvě menší.



Obrázek 2.9: Spojení dlaždic vyplněním trhliny v terénu. Vlevo je konkávní trhlina vyplněna trojúhelníkovým vějířem, vpravo je konvexní trhlina vyplněna třemi trojúhelníky.



Obrázek 2.10: Vytvoření stěn z okrajů dlaždice. Toto lze provést už v době načítání dlaždic.

2.2.4 Nekonečný a editovatelný terén

U tohoto algoritmu je zajištění zobrazení terénu velkých rozměrů poněkud snazší, než je tomu u algoritmu ROAM. Při inicializaci kvadrantového stromu dlaždic postupujeme stejně jako u malého terénu, ale do paměti nahráváme geometrii pouze těch dlaždic, které budeme potřebovat pro vykreslení prvního snímku. V dalších snímcích načítáme ty dlaždice, které je nutno zobrazit a ještě nejsou v paměti, a nepotřebné dlaždice z paměti uvolňujeme. Aby se docílilo nejkratší možné doby pro načtení dlaždice, je vhodné mít předpočítány a uloženy na disku počítače všechny potřebné informace o dlaždici, jakými jsou její ohraničení a statická část chybové metriky.

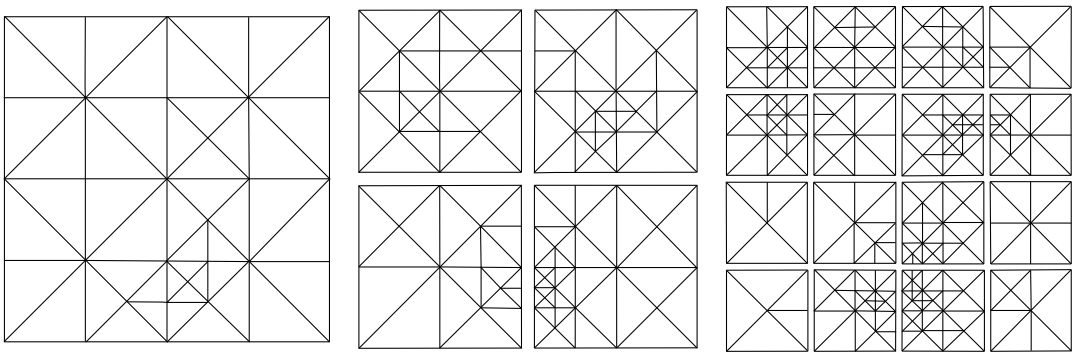
Editace terénu je také snadná, neboť změnou geometrie dlaždice je ovlivněna pouze chyba dlaždice samotné a není nutno tuto změnu nikam propagovat. Pokud je zvolena metrika počítající maximální hodnotu z chyb všech vrcholů dlaždice (viz. 3.1), stačí ji přepočítat jenom u těch vrcholů, které změnily svou polohu.

2.2.5 Shrnutí

Jako konečný verdikt k tomuto algoritmu můžeme konstatovat, že s relativně malým úsilím získáme dostatečné výkonný způsob zobrazování i obrovských terénů, za což musíme zaplatit jistou ztrátou vizuální kvality, protože při změně úrovně detailů celé dlaždice dochází k viditelnému skoku v triangulaci. Tento neduh však lze částečně eliminovat za pomocí geomorphingu (viz. 3.4). Také požadavek na striktně konstantní počet snímků za sekundu je hůře realizovatelný a konstantní počet trojúhelníků zhola nedosažitelný, neboť tento algoritmus negeneruje minimální triangulaci.

2.3 Chunked LoD

Velice podobný algoritmu Geo Mip–Mapping je algoritmus Chunked LoD, který také využívá dlaždice s pevně předpočítanou triangulací. Jeho popis byl uveden v dokumentu [7]. Rozdíl zde však je v tom, že velikost dlaždic není konstantní, ale zmenšuje se s rostoucí úrovní detailů. Dlaždice na nejnižší úrovni detailů pokrývá území celé vstupní výškové mapy,



Obrázek 2.11: První tři úrovně kvadrantového stromu dlaždic algoritmu Chunked LoD. První úroveň obsahuje celou zjednodušenou výškovou mapu, druhá úroveň čtyři čtvrtiny a třetí úroveň šestnáct šestnáctin výškové mapy terénu.

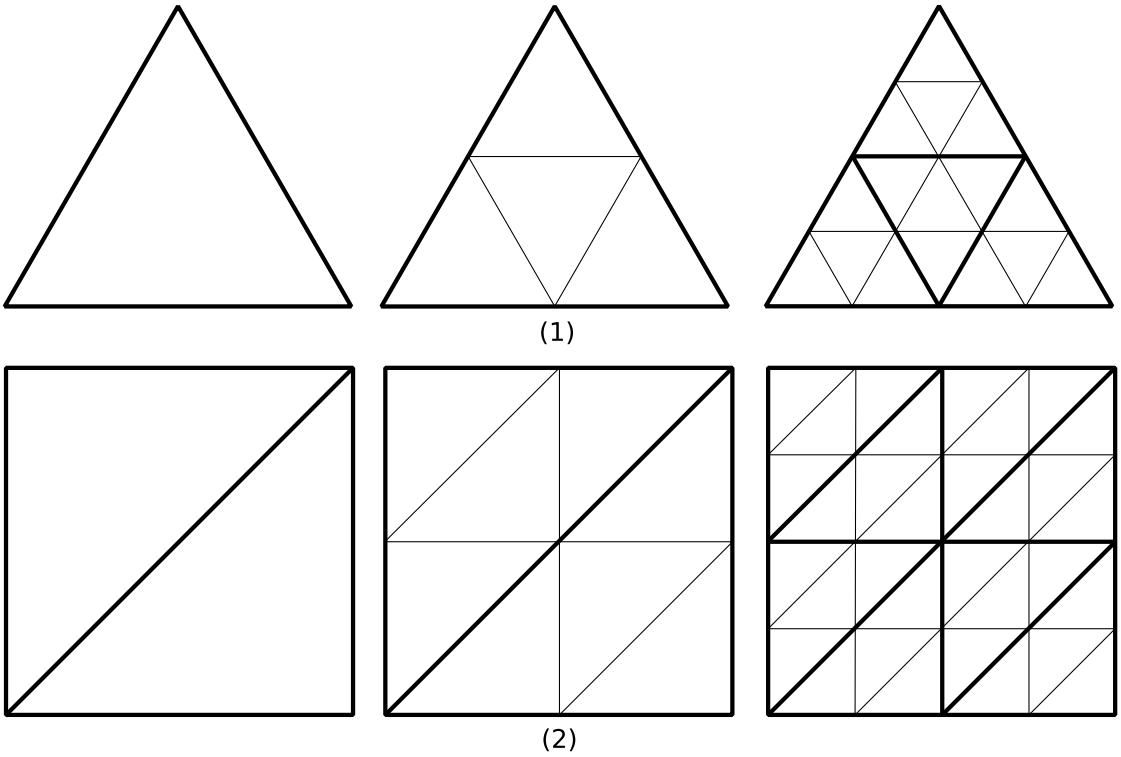
```
void render_tree(node_t node, viewport_t viewport)
{
  if (node.should_render(viewport))
  {
    node.render();
  }
  else
  {
    render_tree(node.get_child(node_t::FIRST));
    render_tree(node.get_child(node_t::SECOND));
    render_tree(node.get_child(node_t::THIRD));
    render_tree(node.get_child(node_t::FOURTH));
  }
}
```

Kód 2.12: Průchod stromem v algoritmu Chunked LoD.

dlaždice na druhé úrovní čtvrtinu území, dlaždice na třetí šestnáctinu, atd. Příklad takového dělení lze vidět na obrázku 2.11.

Podobný, ne však stejný, je také způsob vykreslování. Na rozdíl od Geo Mip–Mappingu, kde průchod kvadrantovým stromem dlaždic slouží pouze k realizaci ořezávání pohledovým tělesem a kde se geometrie dlaždic nachází až na spodní úrovní tohoto stromu, zde se prochází po jednotlivých uzlech stromu, které geometrii obsahují všechny. Na základě zvolené metriky se rozhodně, zda se má vykreslit dlaždice v aktuálním uzlu nebo se má proces opakovat pro jeho potomky. Přehledněji by to mělo být vyjádřeno v kódu 2.12.

Při spojování dlaždic lze opět použít podobné postupy, které byly uvedeny u popisu algoritmu Geo Mip–Mapping, nicméně vzhledem k situaci, že spojujeme dlaždice různých velikostí, nehodí se postupy upravující geometrii okraje dlaždice. Vhodnější jsou proto způsoby, které vyplňují vzniklou mezeru dodatečnými trojúhelníky. Nakonec se můžeme ještě zmínit, že jistou nevýhodou tohoto algoritmu je, že při vizualizaci obrovských terénu je nutno mít již



Obrázek 2.13: První tři úrovně kvadrantového stromu trojúhelníků algoritmu Diamond. (1) – Strom výškové mapy s topologií rovnostranných trojúhelníků. (2) – Dva stromy výškové mapy s topologií pravoúhlých trojúhelníků.

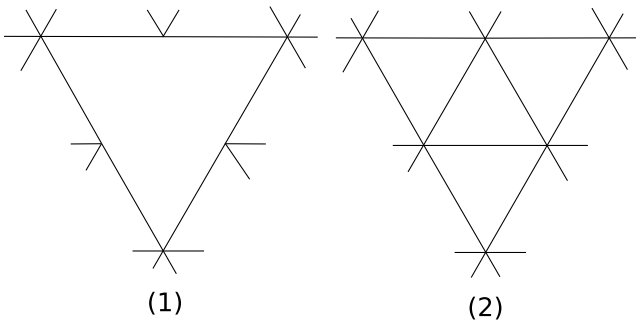
celý strom dlaždic na disku předpočítaný, aby jej bylo možno v průběhu zobrazování podle potřeby načítat.

2.4 Diamond

Variantu algoritmu ROAM uvedl v dokumentu [8] Henri Hakl. Nahradil binární strom trojúhelníků stromem kvadrantovým, a umožnil tak snadnější vytváření trojúhelníkových pásů. Navíc z optimalizačních důvodů nahradil obě prioritní fronty čtyřmi LIFO frontami, které mají konstantní dobu vkládání a odebírání trojúhelníků, jak bude uvedeno dále.

2.4.1 Kvadrantový strom trojúhelníků

Použijeme-li výškovou mapu s topologií rovnostranných trojúhelníků, můžeme binární strom nahradit za kvadrantový. Příklad takového stromu je vidět na obrázku 2.13 části (1). Jelikož však většina výškových map má topologii pravoúhlých trojúhelníků a ne vždy je vhodné výškovou mapu do této topologie přepočítat, můžeme vytvořit kvadrantový strom i z pravoúhlých výškových map, část (2) obrázku 2.13. Jak probíhá spojování a rozdělování trojúhelníku v takovém stromu, je patrné z obrázku 2.14.



Obrázek 2.14: Spojování a rozdělování trojúhelníků v algoritmu Diamond. (1) – Jeden trojúhelník. (2) – Diamant tvořený čtyřmi trojúhelníky.

2.4.2 LIFO fronty

Algoritmus ROAM používá pro řízení spojování a rozdělování trojúhelníků dvě prioritní fronty. Ty bývají implementovány zpravidla pomocí haldy, která má logaritmickou složitost vkládání a odebírání svých prvků. Aby bylo možno tuto složitost zjednodušit na konstantní, byly místo dvou prioritních front zavedeny čtyři LIFO fronty: `split_above`, `split_bellow`, `merge_above` a `merge_bellow`. První část názvu napovídá, zda fronta slouží ke vkládání trojúhelníků, které lze rozdělit, nebo diamantů, které lze sloučit. Druhá část nám říká, zda priorita těchto prvků je nad nebo pod hranicí priority pro aktuální minimální triangulaci, jíž chceme dosáhnout.

V průběhu generování triangulace to pak vypadá tak, že se nejprve přepočítají priority trojúhelníků ve frontě `split_bellow` resp. diamantů ve frontě `merge_above` a ty, jejichž nová priorita překročí mezní prioritu pro aktuální snímek, jsou přesunuty do fronty `merge_bellow` resp. `merge_above`. Poté se vybírají trojúhelníky z fronty `split_above` a diamanty z fronty `merge_bellow`, rozdělují nebo spojují a jejich potomci nebo rodiče se podle priority vkládají do příslušné fronty. Při rozdělování i spojování je nutno ošetřit vznik a vložení nových diamantů a zánik a odstraňování starých. Celý proces opakujeme dokud nejsou fronty `split_above` a `merge_bellow` prázdné, tedy je dosaženo minimální triangulace. Pseudokód k tomuto postupu je uveden v kódu 2.15.

2.4.3 Napojování trojúhelníků

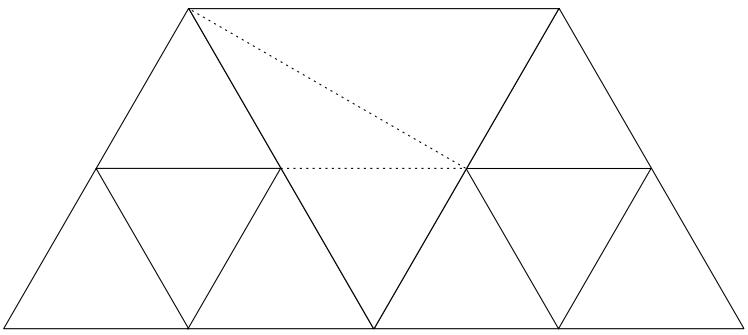
Oproti svému předchůdci nezamezuje algoritmus Diamond vzniku trhlin v terénu v průběhu generování triangulace, ale odkládá řešení tohoto problému až do fáze vlastního vykreslování. Zde se přechody mezi trojúhelníky různých úrovní detailů opraví způsobem zobrazeným na obrázku 2.16. Aby bylo toto napojování jednodušší je doporučeno zajistit, aby spolu sousedili trojúhelníky s rozdílem úrovní detailů maximálně jedna.

2.4.4 Shrnutí

Ačkoliv je tento algoritmus pouze další variantou algoritmu ROAM, zavádí některé zajímavé optimalizace a je důkazem, že se ROAM stále dá v současných implementacích využít.

```
if (first_run)
{
  split_above.clear();
  split_bellow.clear();
  merge_above.clear();
  merge_bellow.clear();
  split_above.add(root);
}
else
{
  // Prvky, jejichž priorita překročí mezní prioritu, jsou přesunuty
  // do split_above respektive merge_bellow.
  split_bellow.recompute(split_above);
  merge_above.recompute(merge_bellow);
}
while (!merge_bellow.is_empty())
{
  merge(merge_bellow.get())
}
while (!split_above.is_empty())
{
  split(split_above.get())
}
```

Kód 2.15: Vykreslovací smyčka algoritmu Diamond.



Obrázek 2.16: Způsob zamezení trhlin v terénu u sousedících trojúhelníků s různou úrovní detailů.

2.5 Další

Kromě doposud zmíněných algoritmů vizualizace terénu existuje ještě celá řada dalších, které by si zasloužili stejný prostor. Nicméně se zde omezíme pouze na vyjmenování některých z nich s odkazem na literaturu, kde jsou tyto algoritmy popsány:

- Stateless One-pass Adaptive Refinement (SOAR) [9] a [10]
- Continuos LoD (CLoD) [11] a [12]
- Geometry Clipmaps [13]
- a další...

Kapitola 3

Společné vlastnosti algoritmů

Některé aspekty algoritmů vizualizace terénu lze zobecnit nezávisle na vysvětlovaném algoritmu a dají se uplatnit ve většině z uvedených algoritmů. Ty, jenž tuto podmínku splňují, jsou uvedeny v této kapitole.

3.1 Chybové metriky

Abychom určili, které trojúhelníky v triangulaci zobrazit nebo ne, musíme zavést nějaké ohodnocení míry vizuální chyby způsobené jejich nepřítomností. Taková ohodnocení nazýváme chybovými metrikami. Většina algoritmů umožňuje využití libovolné metriky za splnění předpokladu monotónnosti. Tedy že rodičovská část triangulace (trojúhelník, dlaždice) má vždy větší nebo alespoň stejnou velikost chyby zobrazení než příslušné synovské části triangulace. Výpočet chybové metriky tvoří prakticky vždy značnou část času spotřebovaného na výpočet triangulace, a tak se chybová metrika rozděluje na statickou část vypočtenou během načítání výškové mapy a dynamickou část počítanou pro každý snímek.

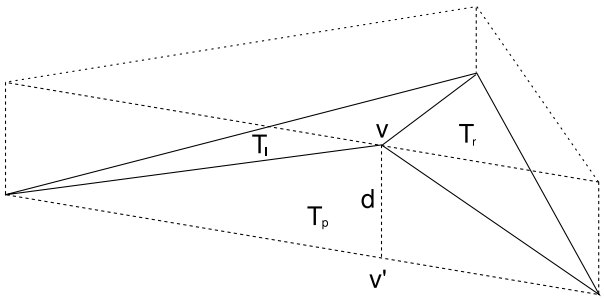
Nejednodušší a nejčastěji používanou metrikou je výpočet výšky hranolu obalujícího rodičovský trojúhelník a jeho dva potomky a projekce této výšky na odpovídající počet pixelů na obrazovce. Tuto metriku ilustruje obrázek 3.1 a rovnice 3.1 nebo 3.2 a 3.3, kde d_l, d_r resp. d_p jsou statické části chybové metriky obou synovských trojúhelníků resp. rodičovského trojúhelníku, e_p projekce rodičovské chyby do prostoru obrazovky, w výška obrazovky v pixelech, n vzdálenost projekční roviny od pozice kamery $\mathbf{p}$, h výška projekční roviny, φ úhel, který svírá horní a dolní rovina pohledového tělesa a $\mathbf{v}$ vrchol, pro který chybu počítáme. Stejná metrika se může využít i u shlukových či dlaždicových algoritmů, ale statická část se počítá jako maximum nebo průměr přes všechny trojúhelníky ve shluku nebo dlaždici.

$$d_p = max\{d_l, d_r\} + |v_z - v'_z| \tag{3.1}$$

$$d_p = max\{d_l, d_r, |v_z - v'_z|\} \tag{3.2}$$

$$e_p = w \cdot \frac{n}{h} \cdot \frac{d_p}{\|\mathbf{p} - \mathbf{v}\|} = \frac{w}{2 \cdot tan\frac{\varphi}{2}} \cdot \frac{d_p}{\|\mathbf{p} - \mathbf{v}\|} \doteq \frac{w}{\varphi} \cdot \frac{d_p}{\|\mathbf{p} - \mathbf{v}\|} \tag{3.3}$$

Rozšíření této metriky za účelem optimalizace zavádí algoritmus SOAR [9]. Kromě monotonní statické části chybové metriky přiřazuje každému trojúhelníku v kvadrantovém stromu



Obrázek 3.1: Hranol obalující levý a pravý synovský trojúhelník. Počítáme jeho výšku $v_z - v'_z$.

trojúhelníků poloměr kuloplochy se středem umístěným ve vrcholu s pravým úhlem, která zahrnuje všechny kuloplochy synovských trojúhelníků. Vizuálně by se to dalo ukázat tak, jak je to zobrazeno na obrázku 3.2. V dynamické části pak vzdálenost trojúhelníku od kamery $\|\mathbf{p} - \mathbf{v}\|$ ve vzorci 3.3 zmenšíme o předpočítaný rádius. Vyjde-li vzdálenost záporná, trojúhelník vykreslíme vždy.

Dále se dá na uvedenou metriku aplikovat poznatek, že díváme-li se na terén kolmo shora, ztrácí projektovaná chyba svůj význam. Proto můžeme zavést do rovnice 3.3 úhel θ, který svírá spojnice trojúhelníku a pozice kamery s rovinou terénu. Tak nám vznikne vzorec 3.4.

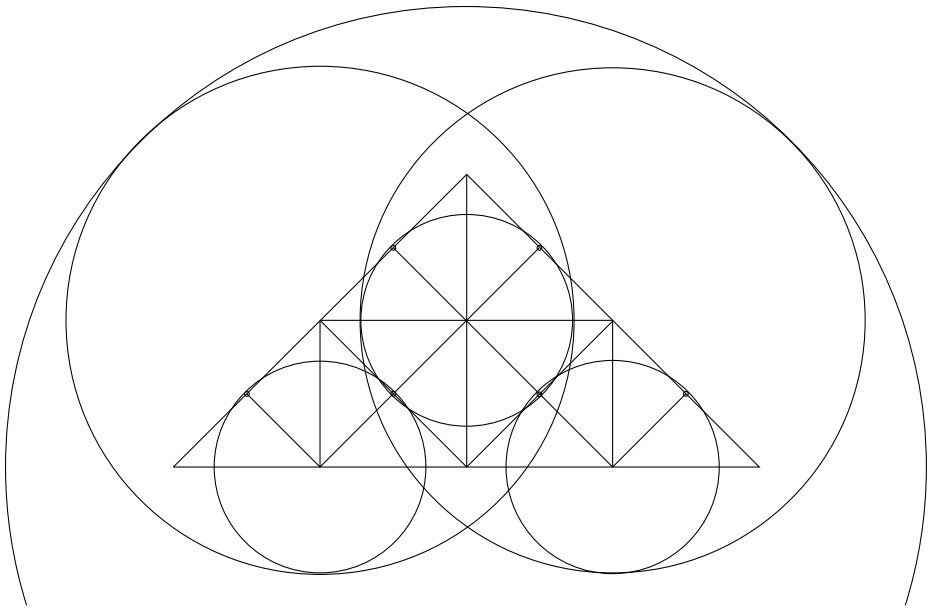
$$e_p = \frac{w}{\varphi} \cdot \frac{d_p}{\|\mathbf{p} - \mathbf{v}\|} \cdot cos\theta \tag{3.4}$$

Existuje mnoho dalších způsobů, jak chybovou metriku vypočítat ať už principiálně nebo s ohledem na některé speciální aplikace. Jako příklad uveďme započítání vlivu mlhy na viditelnost rostoucí se vzdáleností, požadavek na test přímé viditelnosti ve středu obrazovky nebo požadavek zvýšené úrovně detailů v místech umístění objektů na povrchu terénu.

3.2 Ořezávání pohledovým tělesem

Součástí dynamické části chybových metrik je také rozhodování, zda některá část terénu zasahuje do prostoru pohledového tělesa a má tedy smysl ji vykreslovat. Obecně lze říci, že pohledové těleso je tvořeno šesti rovinami, z nichž dvě jsou kolmé na směr pohledu a zbylé čtyři splývají s okraji obrazovky. Pro perspektivní projekci má těmito rovinami ohraničený prostor tvar komolého jehlanu, pro ortogonální projekci je to pak kvádr.

V praxi se realizace ořezávání pohledovým tělesem zúží na test přítomnosti bodů v poloprostorech tvořených šesti rovinami pohledového tělesa. Otestovat jediný bod je triviální záležitostí, problém nastává, chceme-li otestovat přítomnost celého trojúhelníku. Spočívá v situaci, kdy ani jeden z vrcholů trojúhelníku v pohledovém tělese není, ale trojúhelník do pohledového tělesa přesto zasahuje. Zde lze uplatnit analogii Cohen-Shuterland algoritmu, který slouží pro ořezávání úseček přesahujících okraj okna v dvojrozměrném prostoru. Bez dalšího komentáře je tento algoritmus uveden v kódu 3.3, poznamenejme však, že funkce `is_in_view_volume` může ve velmi speciálním případě vrátit `true`, i když trojúhelník v pohledovém tělese není.



Obrázek 3.2: Hierarchie zanořených kuloploch se středem ve vrcholech s pravým úhlem trojúhelníku a poloměrem ohraničujícím kuloplochy potomků.

3.3 Test potenciální viditelnosti

Kromě ořezávání pohledovým tělesem můžeme do dynamické části chybové metriky započítat také výsledek testu potenciální viditelnosti trojúhelníků či dlaždic. Ten provedeme tak, že v době načítání výškové mapy rozdělíme prostor nad terénem na určité množství krychlí či kvádrů. Poté zjistíme přímou viditelnost každé krychle s každou a výsledek této viditelnosti uložíme do bitového pole. V době běhu algoritmu pak vezmeme krychli, do které patří bod, kde je zrovna umístěna kamera, a krychli, do níž patří trojúhelník, pro který test provádíme. Tím docílíme, že se nebudou vykreslovat ty části terénu, které jsou skryty za kopcem vzhledem k aktuální pozici kamery. Podrobně je tento algoritmus popsán v dokumentu [14].

3.4 Geomorphing

Při náhlé změně úrovně detailů dlaždice či nahrazení rodičovského trojúhelníku za synovské a naopak dochází k viditelnému nežádoucímu efektu anglicky zvanému popping. Náhlá změna geometrie je viditelná tím víc, čím je nastavena větší chyba zobrazení. Čistě teoreticky by tento efekt neměl být postřehnutelný vůbec za předpokladu, že chyba zobrazení není větší než jeden pixel. Tehdy by ale v triangulaci bylo tolik trojúhelníku, že nebylo absolutně možné dosáhnout interaktivních počtů snímků za sekundu.

Máme-li tedy chybu zobrazení nastavenu na větší hodnotu, musíme nějakým způsobem zajistit eliminaci viditelnosti prudké změny v triangulaci. K tomu nám slouží metoda zvaná geomorphing. Jedná se o prostou interpolaci polohy bodu výškové mapy ve vertikální ose v závislosti na parametru t, který může nabývat hodnot v rozsahu $\langle 0.0, 1.0 \rangle$. Může být odvozen jak podle doby uplynulé od snímku, ve kterém jsme se rozhodli změnu geometrie provést, nebo podle změny polohy kamery vůči bodu, jehož polohu měníme. Dosáhne-li parametr t krajní

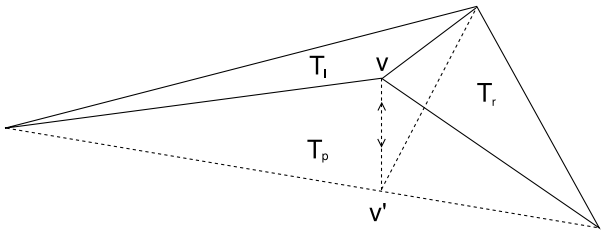
```
bool is_in_view_volume(triangle_t triangle)
{
  int first_flags = 0x00; // Příznak polorovin pro první vrchol trojúhelníku.
  int second_flags = 0x00; // Příznak polorovin pro druhý vrchol trojúhelníku.
  int apex_flags = 0x00; // Příznak polorovin pro třetí vrcholů trojúhelníku.

  int mask = 0x01; // Maska pro aktuální testovanou polorovinu.

  /* Pro všech šest ořezávacích rovin otestujeme každý vrchol trojúhelníku
  na přítomnost v daném poloprostoru. */
  for (int I = 0; I < 6; ++I, mask <<= 1)
  {
    if (!planes[I].isInHalfSpace(triangle.first))
    {
      first_flags |= mask;
    }
    if (!planes[I].isInHalfSpace(triangle.second))
    {
      second_flags |= mask;
    }
    if (!planes[I].isInHalfSpace(triangle.apex))
    {
      apex_flags |= mask;
    }
  }

  /* Je-li některý z vrcholů přímo v pohledu kamery, vykreslíme
  trojúhelník vždy. */
  if ((first_flags & second_flags & apex_flags) == 0x00)
  {
    return true;
  }
  else
  {
    /* Jsou-li všechny vrcholy mimo, vrátíme výsledek potenciální
    viditelnosti trojúhelníku. */
    return !((first_flags & second_flags) && (second_flags & apex_flags)
      && (apex_flags & first_flags));
  }
}
```

Kód 3.3: Analogie Cohen–Shuterland algoritmu pro test přítomnosti trojúhelníku v trojrozměrném pohledovém tělese.



Obrázek 3.4: Geomorphing náhrady synovských trojúhelníků T_l a T_r za rodičovský trojúhelník T_p v algoritmu ROAM.

polohy 0.0, nahradíme ovlivněné trojúhelníky resp. dlaždici jejich rodičem resp. dlaždicí s nižší úrovní detailů. Dosáhne-li parametr t krajní hodnotu 1.0, počítání geomorphingu ukončíme a použijeme geometrii na vyšší úrovni detailů.

Na obrázku 3.4 lze vidět, jak toto probíhá u nahrazování rodiče za potomky v algoritmu ROAM, a rovnice 3.5 popisuje tuto změnu matematicky, kde v_z'' je počítaná výšková pozice vrcholu v, v_z' je výšková pozice středu přepony rodičovského trojúhelníku a v_z je výšková pozice společného vrcholu obou synovských trojúhelníků.

$$v_z'' = v_z' + t(v_z - v_z') \tag{3.5}$$

3.5 Dynamické osvětlení terénu

Zajištění dostatečně reálného osvětlení a stínování terénu vykreslovaného jedním z uvedených algoritmů je velkým problémem. Jak se nám mění geometrie, mění se také normály povrchu terénu. Neustálý přepočet normál trojúhelníků v triangulaci nepřipadá díky své náročnosti v úvahu a používání předpočítaných normál zvyšuje už tak nepříjemný vizuální efekt poppingu. Mnoho implementací proto volí pro simulaci stínování světelné mapy, tj. textury s předpočítanou barvou terénu s ohledem na osvětlení. Vytváříme-li však aplikaci, kde se mění osvětlení terénu v průběhu času např. změnou denní doby, výbuchem, apod., máme potíže i s tímto řešením. Zde nám nezbude než použit řešení s předpočítanými normálami nebo mít předpočítány světelné mapy pro více denních dob a plynule přecházet mezi nimi v průběhu času trvání scény.

Kapitola 4

Implementace algoritmů ROAM a Geo Mip–Mapping

Pro realizaci projektu byla požadována implementace s využitím knihovny Coin dostupné z `http://www.coin3d.org`. Pro zajištění multiplatformní správy okna aplikace byla zvolena knihovna SoQt a pro umožnění načítání výškových map a textur z obrázků ve formátu PNG knihovna simage.

Aby byla zajištěna co největší univerzálnost implementací jednotlivých algoritmů v rámci knihovny Coin, byl upřednostněn přístup vytvoření sady uzlů grafu scény odvozených od třídy `SoShape`. Před ně je pak možno předřadit uzly tříd `SoCoorinate3`, `SoTextureCoordinate2` a `SoTexture`, z nichž si uzly algoritmů za pomocí elementů knihovny Coin berou informace o vstupní výškové mapě a její textuře. Jediné, co je třeba nastavit, je velikost strany výškové mapy tedy druhé odmocniny počtu koordinátů v uzlu třídy `SoCoordinate3`.

Snímky z implementace algoritmu ROAM lze vidět na obrázcích 4.1, 4.2 a 4.3, z implementace algoritmu Geo Mip–Mapping na obrázcích 4.4, 4.5 a 4.6.

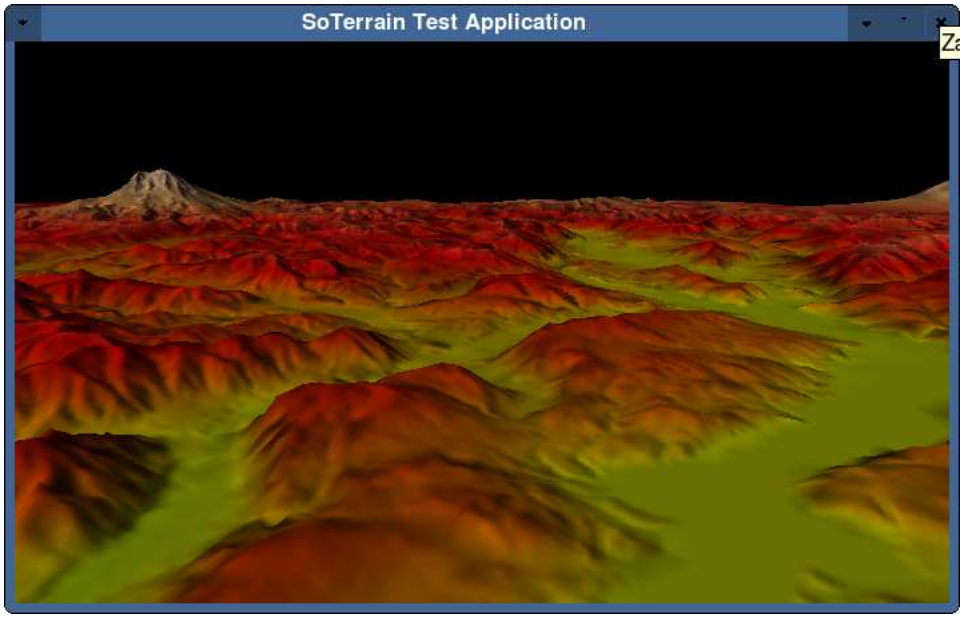
Dále následují některé postřehy z implementací algoritmů, zájemce o seznam implementovaných tříd a jejich popis lze odkázat na dokumentaci generovanou systémem Doxygen umístěnou v adresáři `doc/html/` na přiloženém CD.

4.1 Algoritmus ROAM

Jelikož knihovna Coin skýtá již hotovou implementaci haldy ve třídě `SbHeap`, byla tato třída v projektu využita jako bázová třída pro obě prioritní fronty ve třídách `SbROAMSplitQueue` a `SbROAMMergeQueue`. Pouze bylo třeba zajistit otočení významu priority pro prioritní frontu `SbROAMSplitQueue`.

Binární strom trojúhelníků má konstantní velikost, která se určí již během inicializace, a tak je z důvodů rychlosti a úspory paměti implementován jako pole, kde na indexu 0 leží kořen stromu a indexy potomků každého uzlu se vypočtou podle vzorce $i_l = i_p \cdot 2 + 1$ resp. $i_r = i_p \cdot 2 + 2$, kde i_p je index rodiče, i_l je index levého a i_r je index pravého potomka.

Jako chybová metrika je zvolena jednodušší varianta metriky uvedené společně s algoritmem SOAR [9], kde se ve statické části počítá maximum z rozdílů výšky středu přepony trojúhelníku a výšky bodu výškové mapy v tomto místě přes všechny potomky daného trojúhelníku. V dynamické části se otestuje přítomnost kamery v kouli ohraničující daný trojúhelník. Je-li kamera v této izoploše, trojúhelník se vykreslí, není-li, provede se ještě test, zda



Obrázek 4.1: Terén `ps_height_2k` vykreslený algoritmem ROAM plnými trojúhelníky při maximálním počtu trojúhelníku 10 000 a chybě zobrazení 4 pixely.

projekce statické chyby do vykreslovací roviny nepřesáhne nastavenou hranici chyby zobrazení. Podrobnější vysvětlení této metriky je uvedeno v sekci 3.1.

4.2 Algoritmus Geo Mip–Mapping

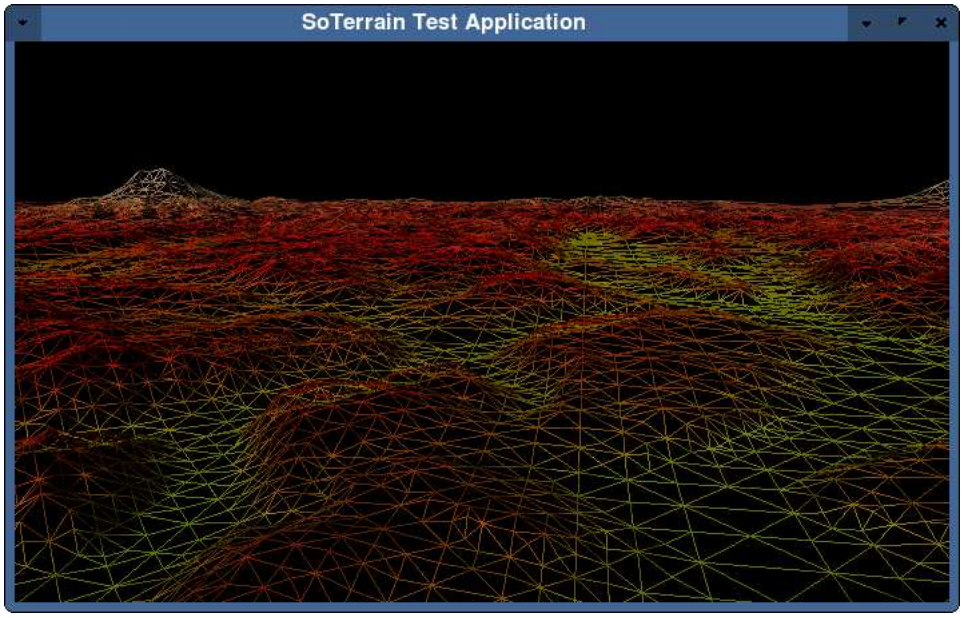
Jedinou zvláštností implementace algoritmu Geo Mip–Mapping je fakt, že geometrie jednotlivých úrovní detailů dlaždice je v paměti uložena zvlášť. Toto sice zabírá více operační paměti, nicméně vykreslování je o něco rychlejší. Hlavním důvodem pro toto řešení je však příprava na vykreslování dlaždic pomocí Vertex Bufer Object rozšíření knihovny OpenGL či za pomocí display–listů, i když to zatím není realizováno.

Chybová metrika je použita původní, představená s popisem algoritmu (viz. dokument [3] a sekce 3.1).

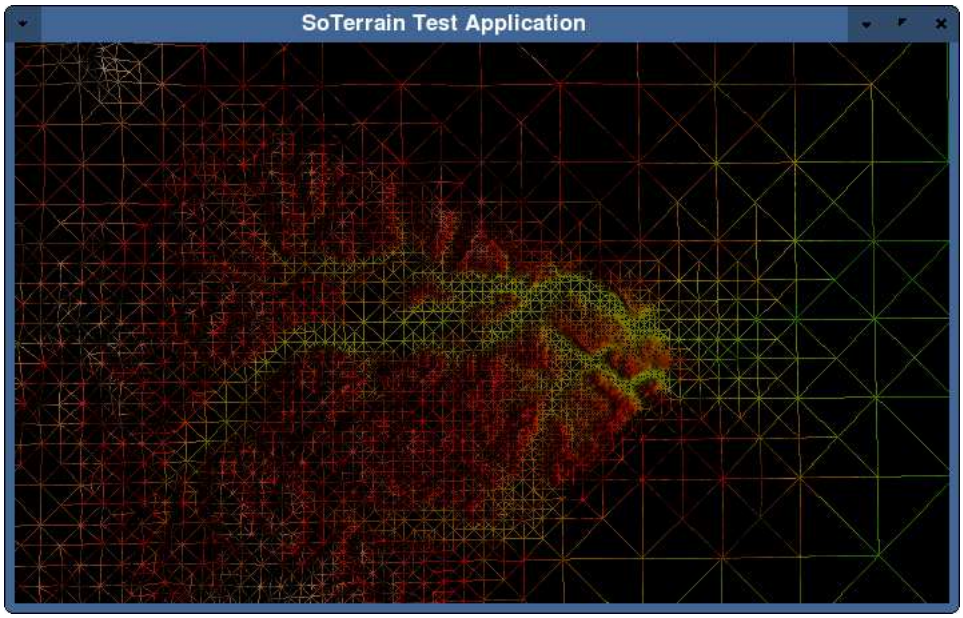
4.3 Profiler

Pro zajištění co nejpřesnějšího způsobu měření doby strávené výpočtem úseku kódu, používá profiler implementovaný ve třídě `PrProfiler` instrukci `RDTSC`, která přečte obsah volného čítače taktů procesoru. Tímto je implementace omezena pouze na platformu x86 a kompatibilní, ale na jiných platformách se může doba zjišťovat standardní funkcí `gettimeofday()`, která ovšem nezajistí takovou přesnost.

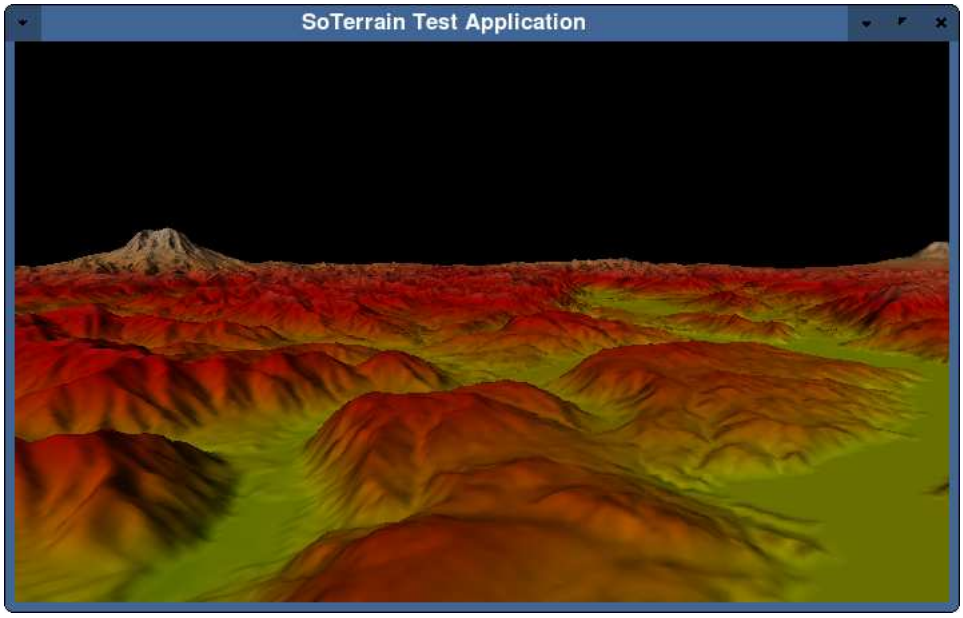
Aby bylo lze získat hodnoty zjištěné doby v sekundách či jiných násobcích této standardní veličiny, musí se v průběhu inicializace profileru zjistit počet taktů procesoru za nějaký referenční čas. Toto se děje ve smyčce, která funkcí `gettimeofday` zjišťuje uplynulý čas a po dovršení nastavené doby odečte počet taktů procesoru za tuto dobu. Pokusy bylo zjištěno, že



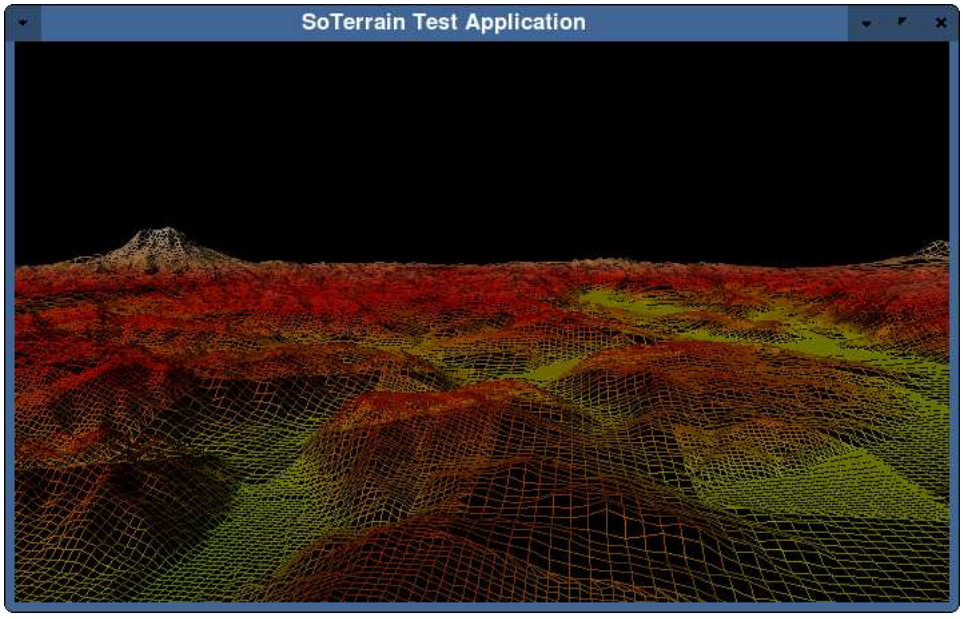
Obrázek 4.2: Terén `ps_height_2k` vykreslený algoritmem ROAM pomocí úseček při maximálním počtu trojúhelníku 10 000 a chybě zobrazení 4 pixely.



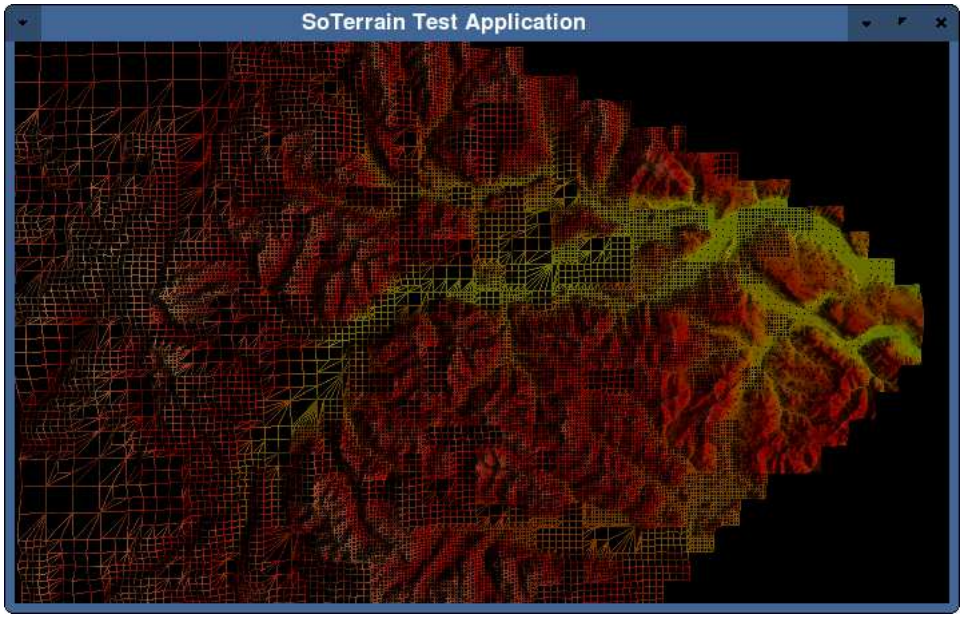
Obrázek 4.3: Terén `ps_height_2k` vykreslený algoritmem ROAM pomocí úseček při maximálním počtu trojúhelníku 10 000 a chybě zobrazení 4 pixely. Pohled na efekt ořezávání pohledovým tělesem.



Obrázek 4.4: Terén `ps_height_2k` vykreslený algoritmem Geo Mip–Mapping plnými trojúhelníky při velikosti dlaždice 33 bodů a chybě zobrazení 4 pixely.



Obrázek 4.5: Terén `ps_height_2k` vykreslený algoritmem Geo Mip–Mapping pomocí úseček při velikosti dlaždice 33 bodů a chybě zobrazení 4 pixely.



Obrázek 4.6: Terén `ps_height_2k` vykreslený algoritmem Geo Mip–Mapping pomocí úseček při velikosti dlaždice 33 bodů a chybě zobrazení 4 pixely. Pohled na efekt ořezávání pohledovým tělesem.

je-li referenční čas alespoň sto milisekund, poskytuje profiler dostatečné přesné výsledky.

Za účelem zvýšení přesnosti měření se odečítá doba strávená odečtem hodnoty volně běžícího čítače a doba potřebná na zpracování a uložení této hodnoty.

Kapitola 5

Výsledky testování

5.1 Jak se testovalo

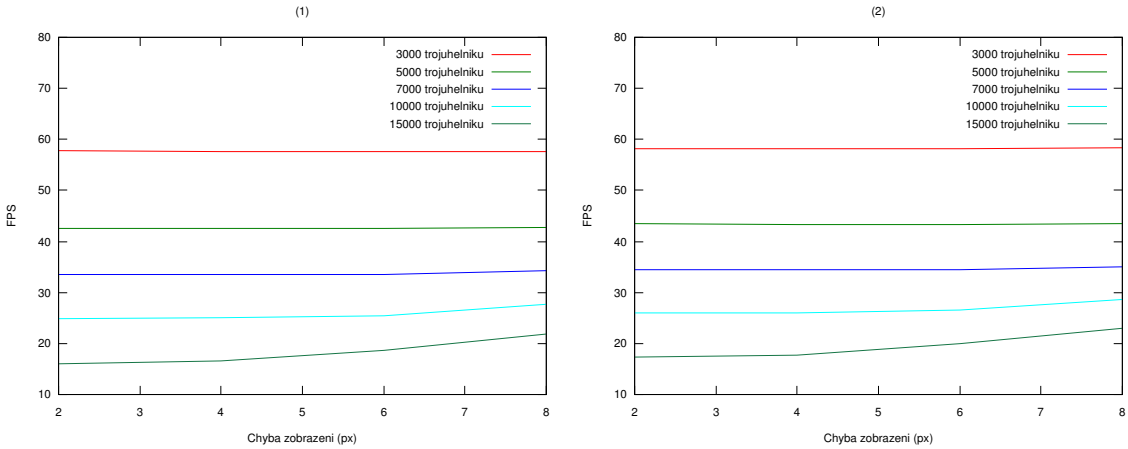
Všechny testy probíhaly na sestavě uvedené v tabulce 5.1. Přitom byly vypnuty všechny procesy nepotřebné k běhu systému a nastavena frekvence procesoru na konstantní maximální hodnotu, aby se eliminoval vliv změny frekvence mobilních procesorů. Testovalo se vykreslováním scény na celou obrazovku při poměrně vysokém rozlišení.

Pro testování výkonové náročnosti byla napsána statická třída `PrProfiler`, jejíž popis je uveden v dokumentaci generované systémem Doxygen na CD-ROM v adresáři `doc/html/`. V adresáři `scripts/` je skript `performance_test.py` napsaný v jazyce Python, který sloužil k provedení vlastních testů. Výsledky testování vyprodukované profilovací knihovnou a uložené v adresáři `results/`, byly skripty `prstat.py` a `performance_graph.py` převedeny do grafů pro tento dokument. Hodnoty v grafech vznikly zprůměrováním výsledků desíti měření se stejnými parametry.

Hodnoty paměťových nároků byly získány programem `top`, nicméně toto měření ukazuje pouze celkovou paměť procesu testovací aplikace. Dále bylo provedeno profilování paměti za pomocí modulu `massif` programu `valgrind`. Grafy z tohoto profilování včetně jejich popisu lze nalézt v adresáři `doc/massif/` na CD-ROM.

Procesor	Intel Pentium M 725 (1.6 Ghz)
RAM	512 MB
Swap	512 MB
Grafická karta	ATI Mobility Radeon 9700
Video RAM	128 MB
Operační systém	Linux (Gentoo, 2.6.16)
Ovladače grafické karty	ATI Proprietary Driver 8.23.7
X Server	X.org 7.0
Rozlišení obrazovky	1400x1050

Tabulka 5.1: Sestava, na které probíhalo testování výkonových a paměťových nároků algoritmů ROAM a Geo Mip-Mapping.



Obrázek 5.1: Závislost počtu snímků za sekundu na chybě zobrazení v pixelech pro výškovou mapu `ps_height_1k` u algoritmu ROAM. (1) – Výsledek pro animaci trvající 10 sekund. (2) – Výsledek pro animaci trvající 100 sekund.

5.2 Algoritmus ROAM

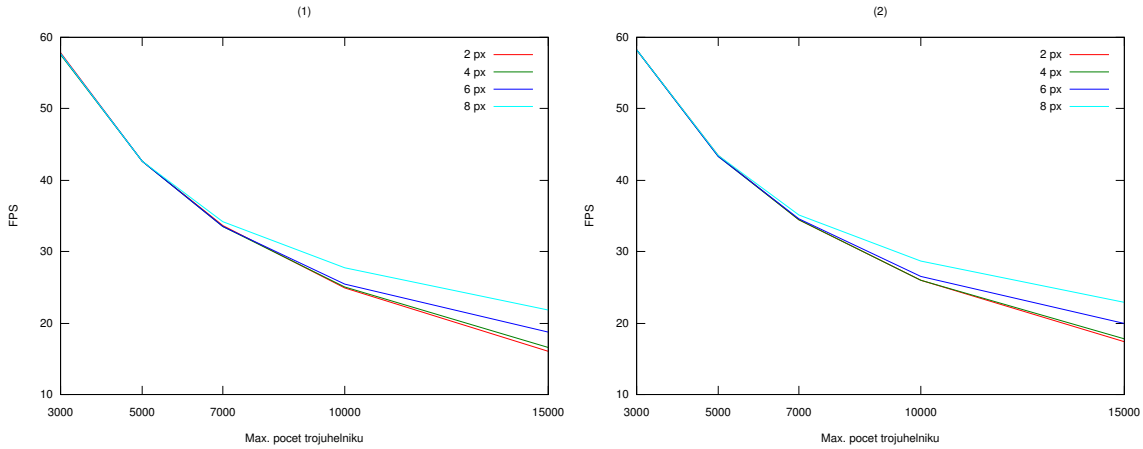
5.2.1 Výkonové nároky

Nejdůležitějším testováním algoritmu ROAM bylo zjištění závislosti výsledného počtu snímků za sekundu v průběhu animace na nastavené chybě zobrazení v pixelech a maximálním počtu trojúhelníků v triangulaci. Výsledek tohoto měření pro výškovou mapu `ps_height_1k.png` je zobrazen na obrázcích 5.1 a 5.2. Zde je jasně vidět, že při hodnotách do 7 000 trojúhelníků v triangulaci se neuplatní nastavená chyba zobrazení do 8 pixelů. U menších výškových map je tato hranice nižší u větších vyšší, jak je vidět na stejných grafech pro jiné výškové mapy v příloze D.

Na obrázku 5.2 je jasně vidět zřejmá nepřímá závislost počtu zobrazovaných trojúhelníků na počet snímků za sekundu. Nicméně na obou obrázcích si lze všimnout, že počet snímků za sekundu pro rychlou scénu (grafy (1) na obrázcích 5.1 a 5.2) zhruba odpovídá počtu snímků za sekundu pro pomalou scénu (grafy (2) na obrázcích 5.1 a 5.2). Toto odporuje teoretickému předpokladu závislosti počtu snímků za sekundu na velikosti změny triangulací mezi dvěma snímky. Důvodem tohoto rozporu je přílišná náročnost přepočtu dynamické části chybové metriky. I profilováním bylo zjištěno, že program tráví nejvíce času tímto výpočtem, a tak by první krok optimalizace dosavadní implementace měl směřovat k zavedení inkrementálního výpočtu dynamické části chybové metriky.

5.2.2 Paměťové nároky

Algoritmus ROAM má oproti algoritmu Geo Mip–Mapping větší nároky na operační paměť počítače. Jaké tyto nároky jsou, je ukázáno v tabulce 5.2. Význam druhého sloupce je hodnota teoreticky spočtených paměťových nároků pro danou výškovou mapu a 10 000 trojúhelníků v triangulaci. První číslo udává paměťové požadavky na uložení vrcholů, textury, texturových souřadnic a normál výškové mapy, druhé číslo je velikost režijních dat algoritmu ROAM pro tuto výškovou mapu. Třetí sloupec tabulky je hodnota změřená programem `top`. Značný



Obrázek 5.2: Závislost počtu snímků za sekundu na maximálním počtu trojúhelníků v triangulaci pro výškovou mapu `ps_height_1k` u algoritmu ROAM. (1) – Výsledek pro animaci trvající 10 sekund. (2) – Výsledek pro animaci trvající 100 sekund.

Výšková mapa	Teoretické	Reálné
ps_height_513	9 + 12 MB	343 MB
ps_height_1k	35 + 48 MB	422 MB
ps_height_2k	140 + 192 MB	725 MB

Tabulka 5.2: Tabulka paměťových nároků algoritmu ROAM.

rozdíl v hodnotách druhého a třetího sloupce tabulky je způsoben režií knihovny Coin.

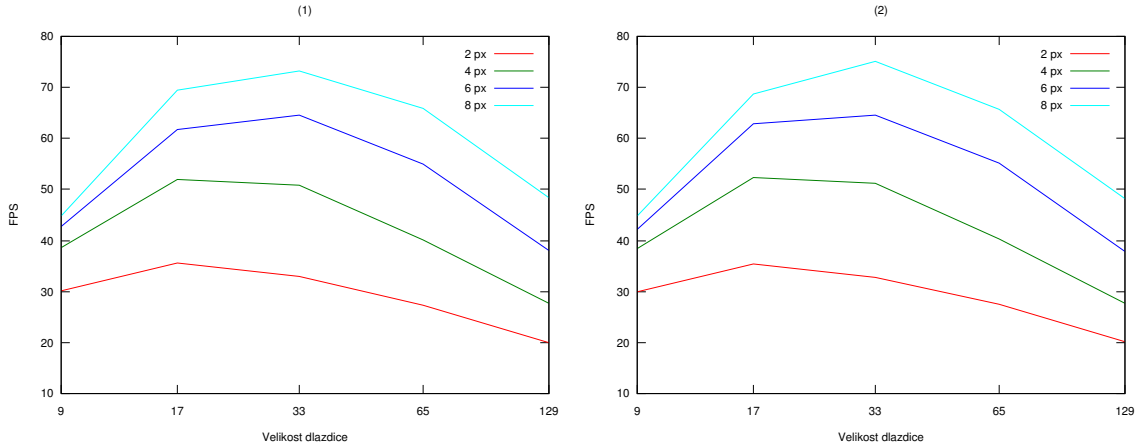
5.3 Algoritmus Geo Mip–Mapping

5.3.1 Výkonové nároky

Výkon algoritmu Geo Mip–Mapping závisí, samozřejmě kromě nastavené chyby zobrazení, také na velikosti dlaždic. Graf této závislosti pro výškovou mapu `ps_height_1k` lze vidět na obrázku. 5.3. Z něj je patrné, že optimální velikost dlaždice pro tuto výškovou mapu je 17 nebo 33 bodů, podle nastavené chyby zobrazení. Dá se očekávat a testy bylo i potvrzeno, že s rostoucí velikostí výškové mapy tato hodnota také stoupá.

5.3.2 Paměťové nároky

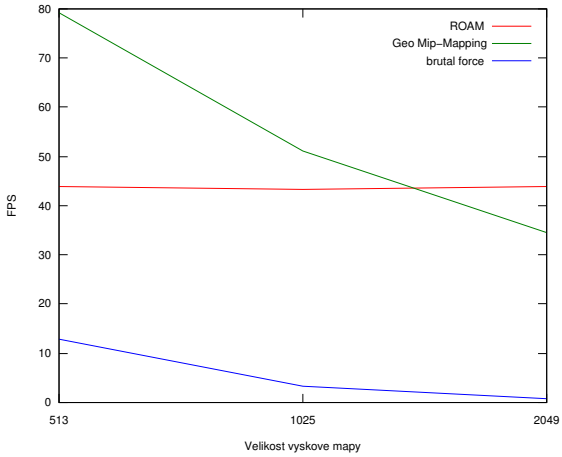
Paměťové nároky algoritmu Geo Mip–Mapping jsou v tabulce 5.3. Význam sloupců je stejný jako v tabulce 5.2 (viz. sekce 5.2.2). Hodnoty jsou platné pro nastavenou velikost strany dlaždice 33 vrcholů. Zde je opět vidět značná režie knihovny Coin.



Obrázek 5.3: Závislost počtu snímků za sekundu na velikosti dlaždice pro výškovou mapu `ps_height_1k` u algoritmu Geo Mip–Mapping. (1) – Výsledek pro animaci trvající 10 sekund. (2) – Výsledek pro animaci trvající 100 sekund.

Výšková mapa	Teoretické	Reálné
ps_height_513	9 + 1,5 MB	332 MB
ps_height_1k	35 + 6 MB	380 MB
ps_height_2k	140 + 25 MB	557 MB

Tabulka 5.3: Tabulka paměťových nároků algoritmu Geo Mip–Mapping.



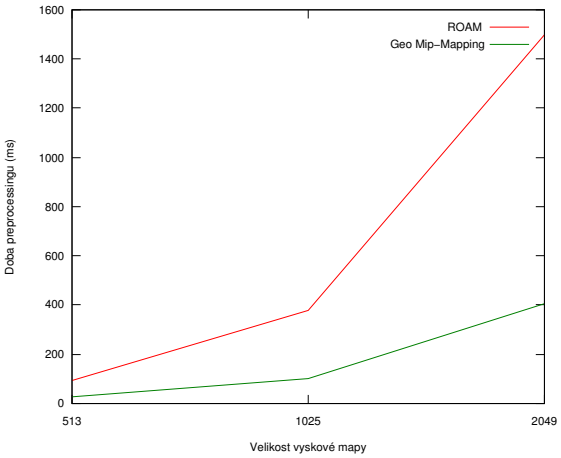
Obrázek 5.4: Závislost počtu snímků za sekundu na velikosti vstupní výškové mapy u algoritmů ROAM a Geo Mip–Mapping.

5.4 Shrnutí

Podle předpokladů zmíněných v popisu jednotlivých algoritmů se ukázalo, že algoritmus Geo Mip–Mapping podává vyšší výkony oproti algoritmu ROAM ve formě v jaké byly implementovány. Nicméně kdyby se algoritmus ROAM rozšířil o uvedené optimalizace (inkrementální generování trojúhelníkových pásů, trojúhelníkové shluky, ...), jistě by dosahoval srovnatelných ba dokonce lepších výsledků. Kvůli rozsahu a časovým nárokům této práce však toto nebylo uskutečněno. Zde je tedy příslib budoucí práce pro některé další projekty.

Zajímavá je také závislost výkonu na velikosti vstupní výškové mapy na obrázku 5.4. Zde je vidět předpoklad algoritmu ROAM, že výkon není závislý na velkosti výškové mapy. Toto je důkazem tvrzení, že algoritmus ROAM je vhodnější pro letecké simulátory, kde je zobrazena velká část terénu z výšky a celkový pohyb kamery je relativně pomalý, kdežto algoritmus Geo Mip–Mapping se hodí spíše tam, kde je zobrazena pouze menší část terénu a pohyb kamery je rychlý. Na tomto grafu je také znázorněno, jakých výkonů dosahuje vykreslování terénu hrubou silou oproti algoritmům ROAM a Geo Mip–Mapping.

Na závěr ještě uveďme graf závislosti doby preprocessingu na velikosti vstupní výškové mapy na obrázku 5.5. Zde opět algoritmus ROAM prohrává, nicméně doba přípravy dat do 2 sekund pro výškovou mapu 2049x2049 bodů je stále přijatelná.



Obrázek 5.5: Závislost doby preprocessingu na velikosti vstupní výškové mapy u algoritmů ROAM a Geo Mip–Mapping.

Kapitola 6

Závěr

Ačkoliv se tento dokument snažil přednést obecný přehled z oblasti algoritmů vizualizace terénu, nepostačuje jeho rozsah (a nikdy ani postačovat nemůže) k tomu, aby toto téma plně vyčerpal. Byly zde uvedeny výsledky a zkušenosti z reálné implementace některých zmiňovaných algoritmů, ale přesto i zde by bylo stále co dohánět.

Kromě zde uvedených optimalizací, které nebyly dosud implementovány, jako je inkrementální generování trojúhelníkových pásů, inkrementální počítání chybové metriky, apod., bude v budoucnu snaha implementovat podporu nekonečného terénu, která si ovšem vynutí poněkud komplexnější přístup k návrhu architektury knihovny. Dále je uvažováno experimentování s podporou chybových metrik volitelných a měnitelných v době běhu programu. Mimo optimalizací výkonu dosavadních algoritmů za použití technologií Vertex Buffer Object či display-listů, mohou být implementovány i další algoritmy, ať už zde uvedené či nikoliv. Nejdříve však bude aplikován dosavadní stav knihovny pro implementaci vizualizačního nástroje pro geografický informační systém GRASS.

Autor doufá, že dovršením této práce nebude jeho studium této oblasti uzavřeno a v budoucnu přinese další výsledky. V případě nejasností, výtek, podmětů, ale i chuti se aktivně připojit, ho můžete kontaktovat na e-mailových adresách `xbarto33@stud.fit.vutbr.cz` nebo `blackhex@post.cz`.

Příloha A

Slovník pojmů

CLoD – viz. Continuos LoD

Continuos LoD (CLoD) – Algoritmus vizualizace terénu využívající kvadrantový strom středů oblastí výškové mapy ke generování triangulace způsobem shora dolů. Autory jsou Stefan Rottger, Wolfgang Heidrich, Philipp Slusallek a Hans-Peter Seidel.

diamant (diamond) – Čtveřice pravoúhlých trojúhelníků sdílejících vzájemně všechny odvěsny. Tyto trojúhelníky lze v triangulaci nahradit za dva rodičovské, aniž by tím vznikla trhlina.

diamond – viz. diamant

Diamond – Algoritmus vizualizace terénu. Alternativa ROAM algoritmu zavádějící trojúhelníkovou topologii výškové mapy a nahrazující dvě prioritní fronty čtyřmi LIFO frontami.

display-list – Technologie grafické knihovny OpenGL umožňující uložit sekvenci příkazů v paměti grafické karty, a urychlit tak jejich provádění.

frustrum – viz. pohledové těleso

geomorphing – Plynulá změna geometrie triangulace výškové mapy v závislosti na čase nebo pohybu kamery za účelem eliminace poppingu.

Geo Mip–Mapping – Algoritmus vizualizace terénu využívající geometrické dlaždice jako analogii k texturovému Mip–Mappingu. Autorem je Willem H. De Boer.

Gourandovo stínování – Způsob stínování těles v počítačové grafice, kdy barva trojúhelníků je podle světelného modelu vypočtena pouze v jeho vrcholech a vnitřek trojúhelníku je vykreslen interpolací vypočtených barev.

height map – viz. výšková mapa

heightmap – viz. výšková mapa

Chunked LoD – Algoritmus vizualizace terénu, který využívá kvadrantový strom obsahující triangulace různých částí výškové mapy na různé úrovni detailů. Kořen obsahuje celou výškovou mapu na nejnižší úrovní detailů, jeho potomci čtvrtinu výškové mapy, atd. Autorem je Thatcher Urlich.

konkávní – Je-li nějaký útvar na výškové mapě konkávní, je zahnutý směrem dolů, tj, má tvar kopce.

konvexní – Je-li nějaký útvar na výškové mapě konvexní, je zahnutý směrem nahoru, tj, má tvar údolí.

Level of Detail – viz. úroveň detailů

LoD – viz. úroveň detailů

manifold – Přesnou matematickou definici tohoto pojmu lze nalézt v příslušné literatuře, pro potřeby tohoto dokumentu stačí vědět, že manifold objekty jsou takové, u nichž okolí každého bodu povrchu je topologicky ekvivalentní s otevřeným kruhem v E_3. Příklad manifold objektu je např. terén definovaný výškovou mapou. Ne manifold objekty jsou např. takové, které jsou složené ze dvou objektů dotýkajících se vzájemně pouze hranou.

mesh (síť trojúhelníků) – Zpravidla uzavřený objekt tvořený vrcholy v prostoru spojenými hranami do trojúhelníků.

nepravidelné trojúhelníkové sítě (TIN, triangulated irregular network) – Jak již napovídá sám název, představuje nepravidelná trojúhelníková síť ucelený povrch tvořený trojúhelníky sdílejícími své hrany. Rozmístění vrcholů těchto trojúhelníků je zcela libovolné.

pohledové těleso (frustrum) – Objekt v souřadnicích zobrazované scény, který odpovídá prostoru, který se může zobrazit na obrazovce při vykreslování. U perspektivní projekce má pohledové těleso tvar komolého jehlanu, u ortogonální projekce má tvar kvádru.

popping – Náhlá a viditelná změna geometrie terénu, která působí nežádoucí vizuální efekt.

preprocessing (předzpracování, příprava) – Počáteční část algoritmu, kdy se připraví datové struktury a spočítají konstantní hodnoty nutné pro co nejrychlejší běh zbytku algoritmu.

Real-time Optimally Adapting Meshes (ROAM) – Algoritmus vizualizace terénu pracující se dvěma prioritními frontami a binárním stromem trojúhelníků. Autory jsou Mark A. Duchaineau, Murray Wolinky, David E. Sigeti, Mark C. Miller, Charles Aldrich a Mark B. Mineev-Weinstein.

ROAM – viz. Real-time Optimally Adapting Meshes

ROAM Using Surface Triangle Clusters – Algoritmus vizualizace terénu rozšiřující algoritmus ROAM o shluky trojúhelníků umístěných místo samostatných trojúhelníků v triangulaci. Autorem je Alex A. Pomeranz.

RUSTiC – viz. ROAM Using Surface Triangle Clusters

SOAR – viz. Stateless One-pass Adaptive Refinement

Stateless One-pass Adaptive Refinement (SOAR) – Algoritmus vizualizace terénu využívající kvadrantový strom trojúhelníků a optimalizované uložení dat terénu na disku či paměti. Autory jsou Peter Lindstrom a Valerio Pascucci.

světelná mapa – Textura umístěna na trojúhelníky terénu simulující předpočítané barvy osvětlení a stínování terénu.

T-vertex – viz. T-vrchol

T-vrchol (T-vertex) – Vrchol trojúhelníku v triangulaci, který těsně přiléhá k hraně jiného trojúhelníku jinde než na jejím okraji. Z důvodů zaokrouhlovacích chyb se může v místě tohoto vrcholu vykreslit trhlina.

TIN – viz. nepravidelné trojúhelníkové sítě

triangulace – Obecný pojem pro pravidelné i nepravidelné trojúhelníkové sítě. V kontextu vizualizace terénu jde o množinu trojúhelníků tvořících souvislý povrch terénu bez trhlin.

triangulated irregular network – viz. nepravidelné trojúhelníkové sítě

úroveň detailů (LoD, level of detail) – Technika zajištění rychlejšího vykreslovaní grafických objektů na obrazovce. V závislosti na vzdálenosti objektu od kamery se zobrazuje jeho zjednodušená podoba.

Vertex Buffer Object – Technologie grafické knihovny OpenGL umožňující uložit geometrii objektů v paměti grafické karty, a urychlit tak jejich vykreslování.

výšková mapa (heightmap, height map) – Matice bodů udávajících výšku terénu v daném místě. Poloha tohoto bodu je dána implicitně topologií. Ta bývá většinou čtvercová.

Příloha B

Ovládání testovací aplikace

Pro demonstraci implementovaných algoritmů vznikla jednoduchá aplikace `SoTerrainTest`, kterou lze přeložit příkazem `make` v hlavním adresáři projektu. Příkaz `make debug` a `make gnu_profile` přeloží aplikaci s debugovacími informacemi resp. debugovacími a profilovacími informacemi. Pro zakompilování knihovny `PrProfiler` použijte příkaz `make profile`. Spuštění aplikace lze provést příkazem `make run`. Dokumentace generovaná systémem Doxygen se vytvoří příkazem `make doc`. Příkaz `make clean` smaže výsledky kompilování.

Parametry a použití testovací aplikace je následující:

```
Použití: SoTerrainTest -h heightmap [-t texture] [-p profile_file]
[-a algorithm] [-A animation_time] [-F frame_time] [-e pixel_error]
[-r triangle_count] [-g tile_size] [-f] [-c] [-v] [-s]
  -h heightmap            Soubor se vstupní výškovou mapou.
  -t texture              Soubor s texturou terénu.
  -p profile_file         Soubor pro uložení výsledku profilování.
  -a algorithm            Výběr algoritmu pro zobrazení terénu.
       brutalforce             Vykreslení terénu hrubou silou.
       roam                    Vykreslení terénu algoritmem ROAM.
       geomipmapping           Vykreslení terénu algoritmem Geo Mip-Mapping.
  -A animation_time       Délka animace v milisekundách.
  -F frame_time           Doba trvání jednoho snímku v milisekundách.
  -e pixel_error          Nastavení chyby zobrazení algoritmu.
  -r triangle_count       Nastavení počtu zobrazovaných trojúhelníků.
  -g tile_size            Nastavení velikosti dlaždice.
  -f                      Spuštění programu na celé obrazovce.
  -c                      Zapnutí ořezávání pohledovým tělesem.
  -v                      Spuštění animace po startu programu.
  -s                      Zapnutí synchronizace snímků animace.
```

Parametr `-r` funguje, pouze je-li zvolen algoritmus ROAM, a parametr `-g` funguje jenom u algoritmu Geo Mip–Mapping. Je-li zvolen parametr `-s`, trvá animace přesně dobu nastavenou parametrem `-A`, v opačném případě může dojít ke zpožďování vykreslování snímků a animace se prodlouží. Ukládání výsledků profilování probíhá pouze tehdy, je-li program přeložen příkazem `make profile`. Není-li zadán parametr `-v`, je zobrazování terénu interaktivní s následujícím významem kláves klávesnice (Pro názvy kláves byly použity názvy běžné

pro knihovnu Coin):

W – posun kamery kupředu

S – posun kamery dozadu

A – posun kamery doleva

D – posun kamery doprava

Q – posun kamery nahoru

E – posun kamery dolů

LEFT – natočení kamery doleva

RIGHT – natočení kamery doprava

UP – natočení kamery nahoru

DOWN – natočení kamery dolů

PGUP – naklonění kamery doleva

PGDOWN – naklonění kamery doprava

PAD_ADD – zvýšení chyby zobrazení v pixelech

PAD_SUBTRACT – snížení chyby zobrazení v pixelech

F – zapnutí/vypnutí ořezávání pohledovým tělesem

L – zapnutí/vypnutí zobrazení terénu pomocí úseček

LEFT_ALT+ENTER – přepnutí celoobrazovkového režimu prohlížeče

ESC – ukončení aplikace

Příloha C

Obsah přiloženého CD

Na CD přiloženém k tomuto dokumentu lze nalézt tyto soubory a adresáře:

`bin/` – Spustitelné soubory pro operační systémy Linux a Windows.

`bin/linux/` – Spustitelné soubory testovací aplikace pro operační systém Linux.

`bin/windows` – Spustitelné soubory testovací aplikace pro operační systém Windows.

`doc/` – Různé dokumentace k projektu SoTerrain.

- `html/` – Dokumentace knihovny SoTerrain generovaná systémem Doxygen.
- `massif/` – Grafy a popisy grafů profilování paměti generované nástrojem `massif` programu `valgrind`.
- `prezentace/` – Slidy použité pro prezentaci semestrálního projektu.
- `zprava/` – Zdrojové soubory tohoto dokumentu.
  - `images/` – Obrázky použité v tomto dokumentu.

`help/` – Dokumenty použité literatury i jiné dokumenty s tématikou vizualizace terénu.

`images/` – Obrázky a obrázková data.

- `doc/` – Zdroje obrázků použitých v tomto dokumentu.
- `graphs/` – Grafy použité v tomto dokumentu.
- `heightmaps/` – Výškové mapy terénů.
- `textures/` – Textury terénů.

`include/` – Hlavičkové soubory.

- `geomipmapping/` – Hlavičkové soubory algoritmu Geo Mip–Mapping.
- `profiler/` – Hlavičkové soubory profileru.
- `roam/` – Hlavičkové soubory algoritmu ROAM.

`obj/` – Adresář pro odkládání zkompilovaných objektů.

`plots/` – Zdrojové soubory grafů pro program `gnuplot`.

`results/` – Komprimované výsledky testování, před použitím nutno dekomprimovat.

`scripts/` – Skripty v jazyce Python pro provedení testování a generování grafů.

`scr/` – Zdrojové soubory.

- `geomipmapping/` – Zdrojové soubory algoritmu Geo Mip–Mapping.
- `profiler/` – Zdrojové soubory profileru.
- `roam/` – Zdrojové soubory algoritmu ROAM.

`doxyfile` – Soubor pro generování dokumentace systémem Doxygen.

`makefile` – Soubor pro překlad projektu programem `make`.

`run.bat` – Skript pro spuštění testovací aplikace pod operačním systémem Windows.

`run.sh` – Skript pro spuštění testovací aplikace pod operačním systémem Linux.

`README` – Stručný popis projektu a ovládání testovací aplikace.

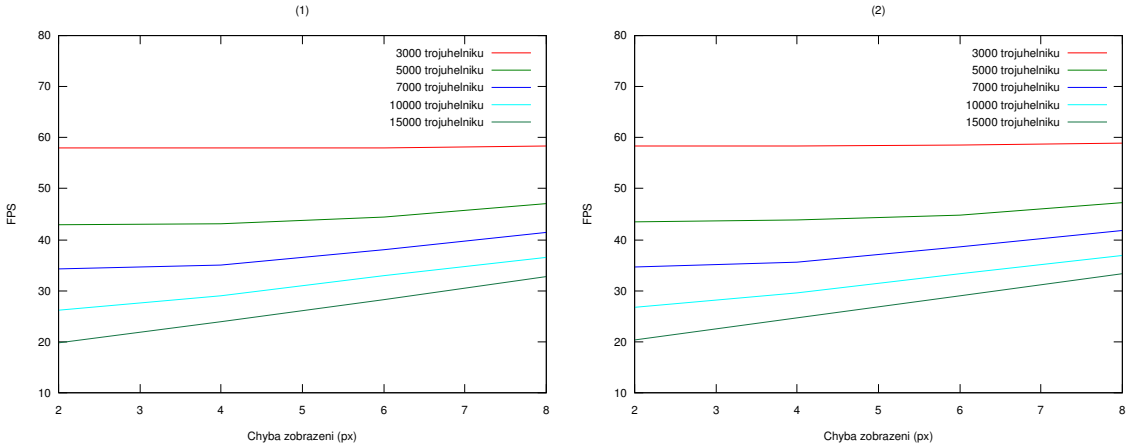
`SoTerrain.dsp` – Projektový soubor Microsoft Visual Studio 6.0.

`SoTerrain.dsw` – Workspace soubor Microsoft Visual Studio 6.0.

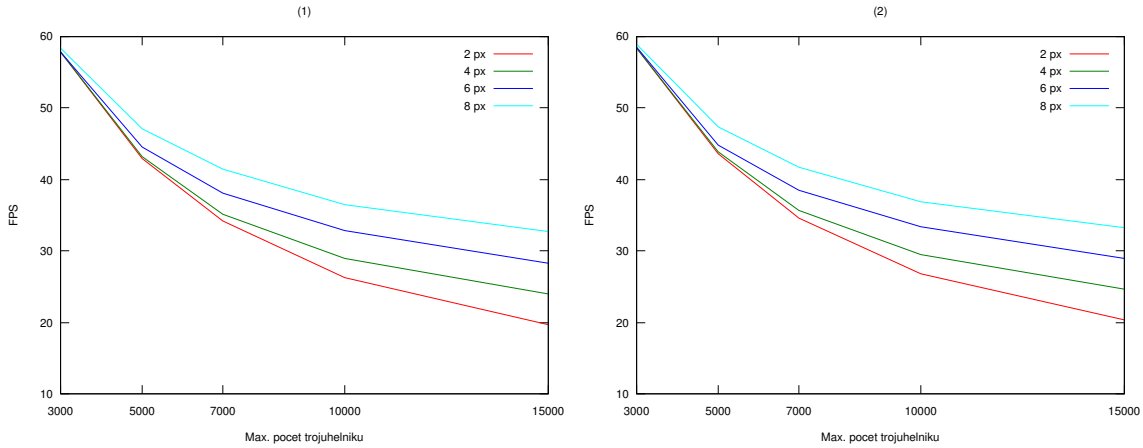
`SoTerrain.kateproject` – Projektový soubor pro program Kate.

Příloha D

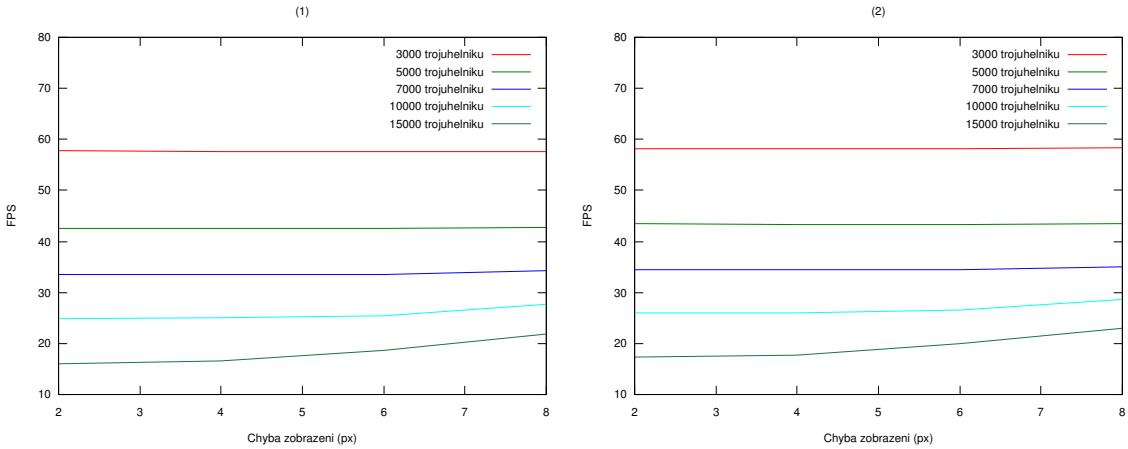
Grafy výsledků testování



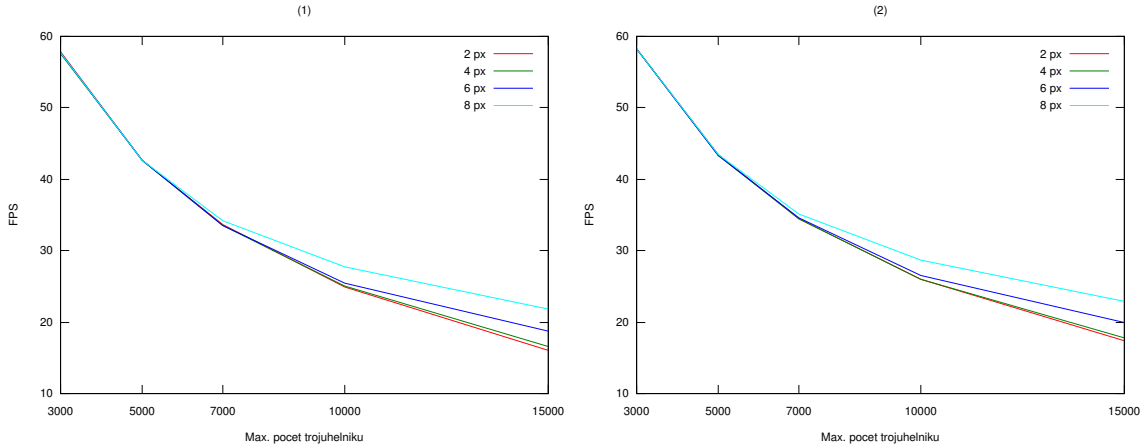
Závislost počtu snímků za sekundu na chybě zobrazení v pixelech pro výškovou mapu `ps_height_513` u algoritmu ROAM. (1) – Výsledek pro animaci trvající 10 sekund. (2) – Výsledek pro animaci trvající 100 sekund.



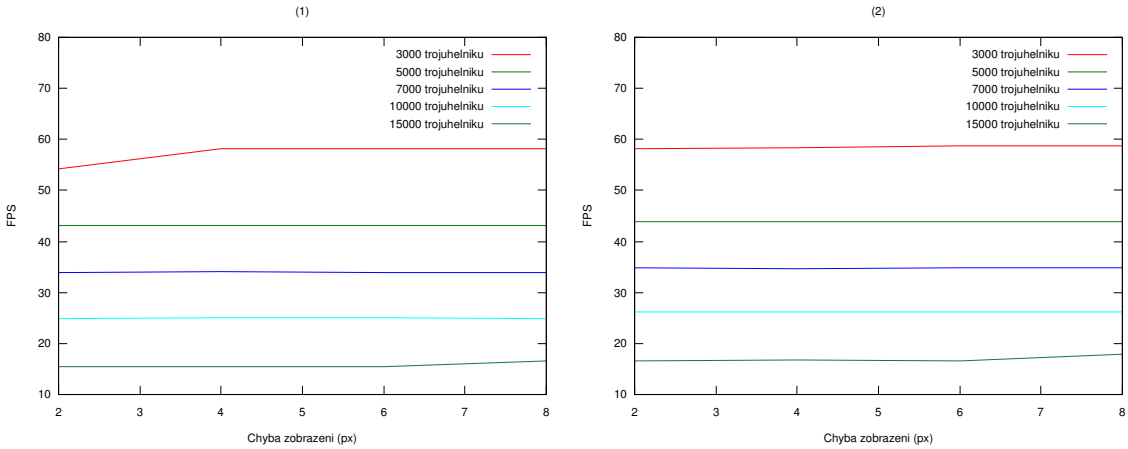
Závislost počtu snímků za sekundu na maximálním počtu trojúhelníků v triangulaci pro výškovou mapu `ps_height_513` u algoritmu ROAM. (1) – Výsledek pro animaci trvající 10 sekund. (2) – Výsledek pro animaci trvající 100 sekund.



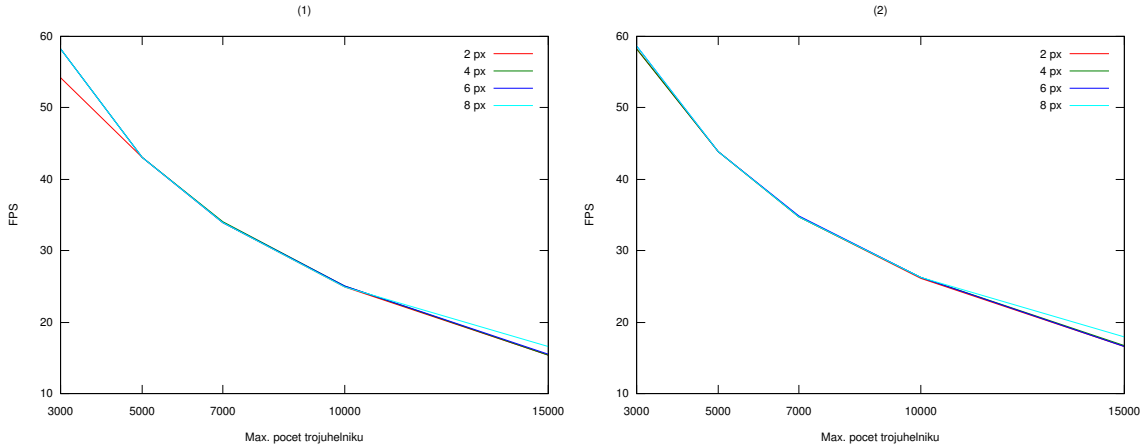
Závislost počtu snímků za sekundu na chybě zobrazení v pixelech pro výškovou mapu `ps_height_1k` u algoritmu ROAM. (1) – Výsledek pro animaci trvající 10 sekund. (2) – Výsledek pro animaci trvající 100 sekund.



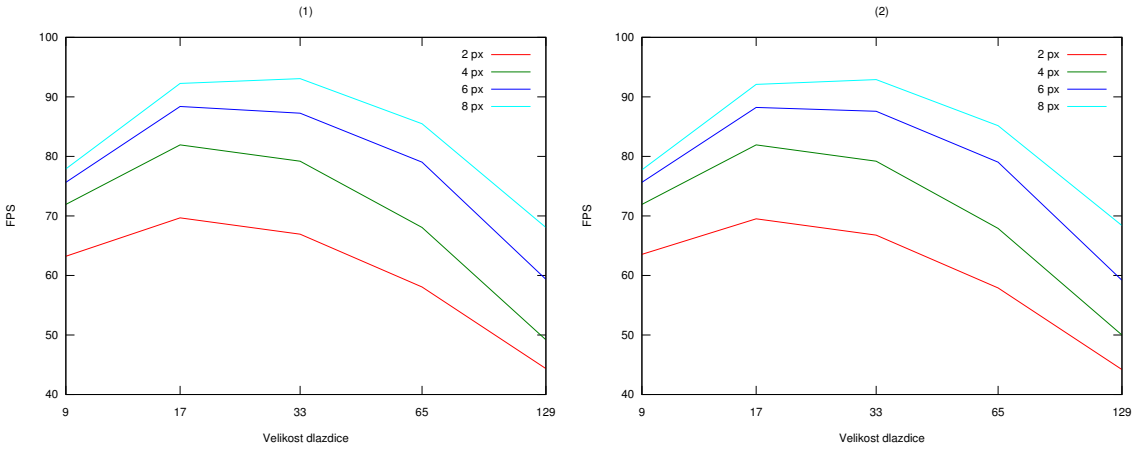
Závislost počtu snímků za sekundu na maximálním počtu trojúhelníků v triangulaci pro výškovou mapu `ps_height_1k` u algoritmu ROAM. (1) – Výsledek pro animaci trvající 10 sekund. (2) – Výsledek pro animaci trvající 100 sekund.



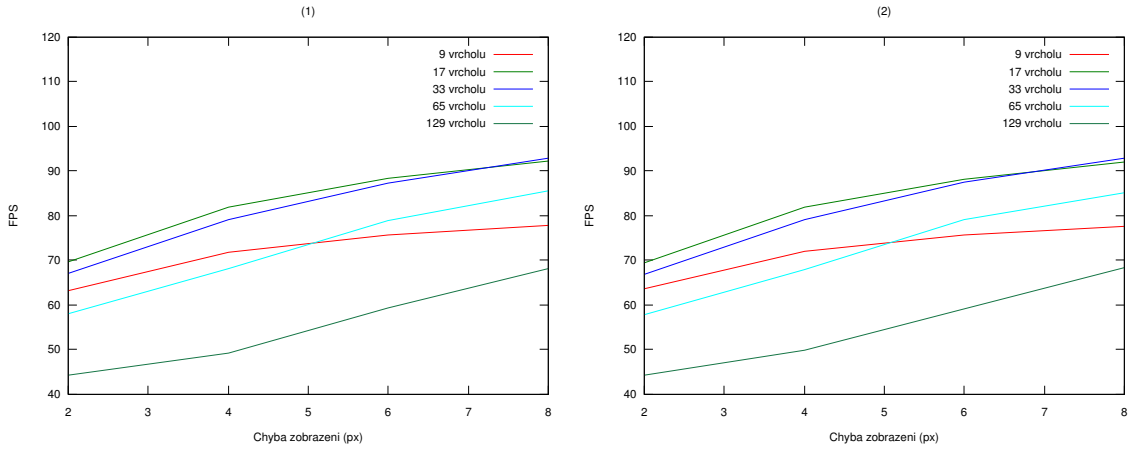
Závislost počtu snímků za sekundu na chybě zobrazení v pixelech pro výškovou mapu `ps_height_2k` u algoritmu ROAM. (1) – Výsledek pro animaci trvající 10 sekund. (2) – Výsledek pro animaci trvající 100 sekund.



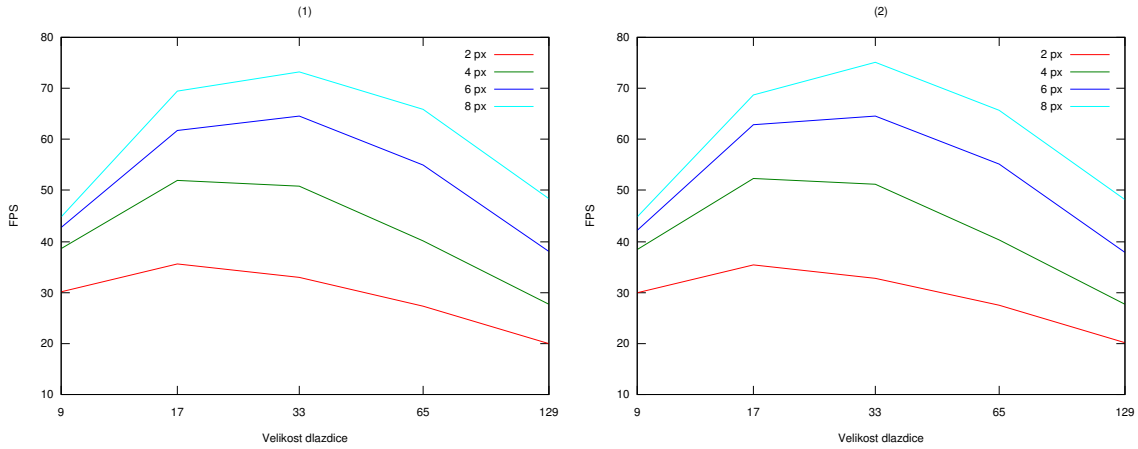
Závislost počtu snímků za sekundu na maximálním počtu trojúhelníků v triangulaci pro výškovou mapu `ps_height_2k` u algoritmu ROAM. (1) – Výsledek pro animaci trvající 10 sekund. (2) – Výsledek pro animaci trvající 100 sekund



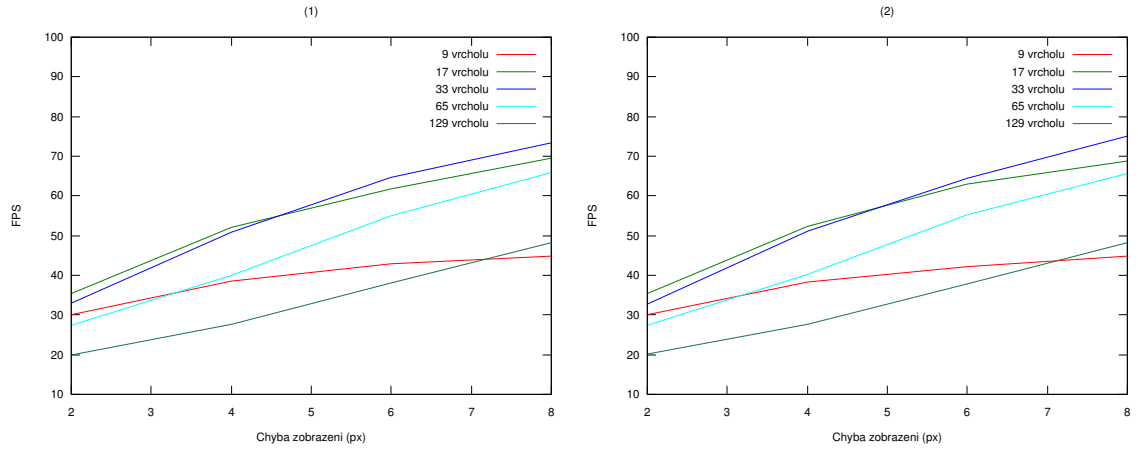
Závislost počtu snímků za sekundu na velikosti dlaždice pro výškovou mapu `ps_height_513` u algoritmu Geo Mip–Mapping. (1) – Výsledek pro animaci trvající 10 sekund. (2) – Výsledek pro animaci trvající 100 sekund.



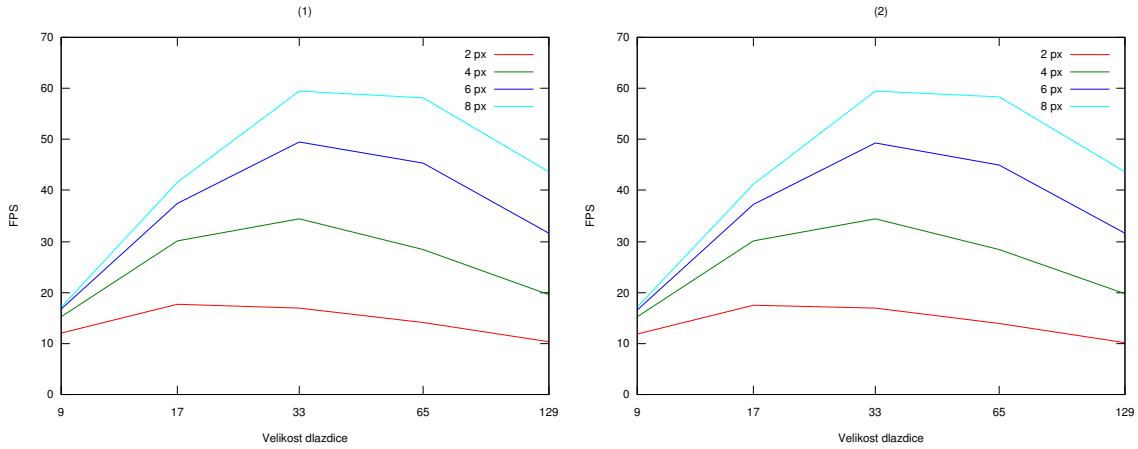
Závislost počtu snímků za sekundu na chybě zobrazení v pixelech pro výškovou mapu `ps_height_513` u algoritmu Geo Mip–Mapping. (1) – Výsledek pro animaci trvající 10 sekund. (2) – Výsledek pro animaci trvající 100 sekund.



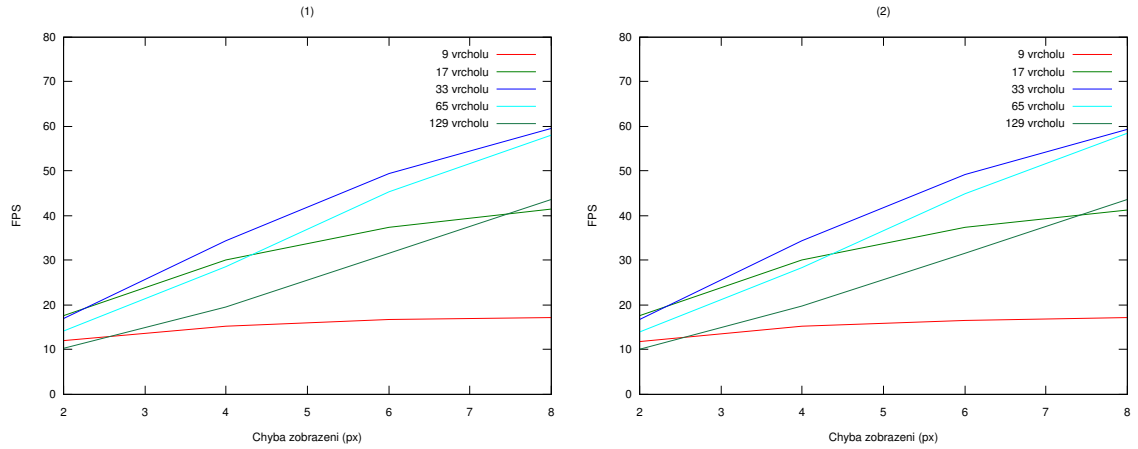
Závislost počtu snímků za sekundu na velikosti dlaždice pro výškovou mapu `ps_height_1k` u algoritmu Geo Mip–Mapping. (1) – Výsledek pro animaci trvající 10 sekund. (2) – Výsledek pro animaci trvající 100 sekund.



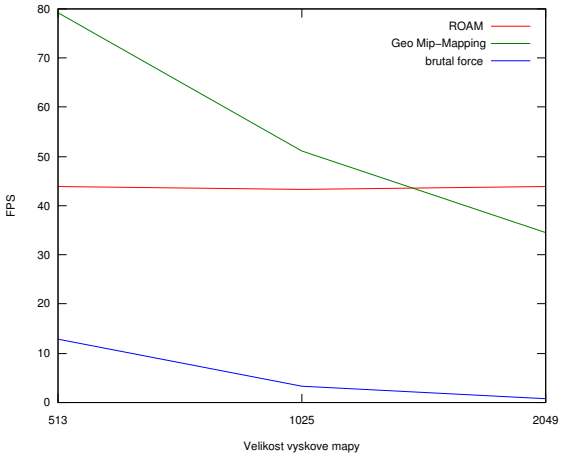
Závislost počtu snímků za sekundu na chybě zobrazení v pixelech pro výškovou mapu `ps_height_1k` u algoritmu Geo Mip–Mapping. (1) – Výsledek pro animaci trvající 10 sekund. (2) – Výsledek pro animaci trvající 100 sekund.



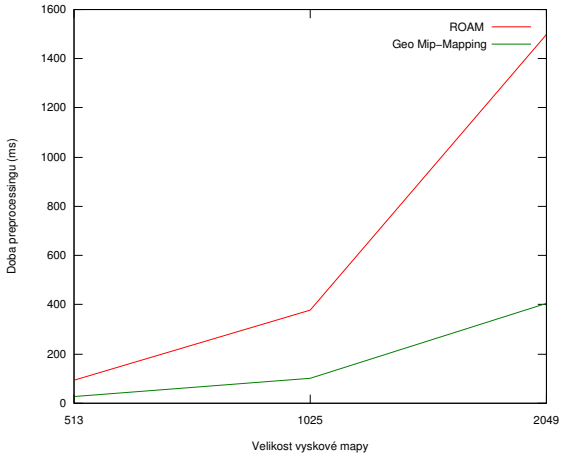
Závislost počtu snímků za sekundu na velikosti dlaždice pro výškovou mapu `ps_height_2k` u algoritmu Geo Mip–Mapping. (1) – Výsledek pro animaci trvající 10 sekund. (2) – Výsledek pro animaci trvající 100 sekund.



Závislost počtu snímků za sekundu na chybě zobrazení v pixelech pro výškovou mapu `ps_height_2k` u algoritmu Geo Mip–Mapping. (1) – Výsledek pro animaci trvající 10 sekund. (2) – Výsledek pro animaci trvající 100 sekund.



Závislost počtu snímků za sekundu na velikosti vstupní výškové mapy u algoritmů ROAM a Geo Mip–Mapping.



Závislost doby preprocessingu na velikosti vstupní výškové mapy u algoritmů ROAM a Geo Mip–Mapping.

Literatura

[1] DUCHAINEAU, Mark A., WOLINKSY, Murray, SIGETI, David E. MILLER, Mark C., ALDRICH, Charles a MINEEV-WEINSTEIN, Mark B. ROAMing terrain: Real–time optimally adapting meshes. *IEEE Visualization '97*. 1997, strany 81—88. Dostupný z: `http://www.llnl.gov/graphics/ROAM/roam.pdf`

[2] POMERANZ, Alex A. *ROAM Using Surface Triangle Clusters (RUSTiC)*. Davis, California: University of California, 2000. 46 stran. Dostupný z: `http://www.cognigraph.com/ROAM_homepage/PomeranzThesis.pdf`

[3] DE BOER, Willem H. *Fast Terrain Rendering Using Geometrical MipMapping*. 2000. 7 stran. Dostupný z: `http://www.flipcode.com/articles/article_geomipmaps.pdf`

[4] LARSEN, Bent Salgaard a CHRISTENSEN, Niels Jørgen. *Real–time Terrain Rendering using Smooth Hardware Optimized Level of Detail*. WSCG, 2003. 8 stran. Dostupný z: `http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/1425/pdf/imm1425.pdf`

[5] LINDSTROM, Peter, KOLLER, David, HODGES, Larry F., RIBARSKY, William, FAUST, Nick a TURNER, Gregory. *Level–of–Detail Management for Realtime Rendering of Phototextured Terrain*. Georgia: Georgia Institute of Technology, GIT-GVU-95-06, 1995. 16 stran. Dostupný z: `ftp://ftp.gvu.gatech.edu/pub/gvu/tr/1995/95-06.pdf`

[6] VRBA, Petr. *Perlin Generator & Chunked-LOD*. Brno: Fakulta informačních technologií, Vysoké učení technické, 2005. 23 stran. Dostupný z: `http://merlin.fit.vutbr.cz/upload/IvProjects/2005/chunkedLOD/chunkedLOD.pdf`

[7] URLICH, Thatcher. *Rendering Massive Terrains using Chunked Level of Detail Control*. Oddworld Inhabitants, 2002. 14 stran. Dostupný z: `http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/*checkout*/tu-testbed/tu-testbed/docs/sig-notes.pdf?rev=HEAD`

[8] HAKL, Henri. *Diamond Terrain Algorithm: Continuos Levels of Detail for Height Fields*. University of Stellenbosh, 2001. 17 stran. Dostupný z: `http://www.cs.sun.ac.za/~henri/DiamondPaper.ps`

[9] LINDSTROM Peter a PASCUCCI, Valerio. Visualisation of Large Terrains Made Easy. *IEEE Visualization '01*. 2001, strany 363-370, 574. Dostupný z: `http://www.pascucci.org/pdf-papers/vis2001.pdf`

[10] LINDSTROM Peter a PASCUCCI, Valerio. Terrain Simplification Simplified: A General Framework for View-Dependent Out–of–Core Visualization. *IEEE Transactions on*

Visualization and Computer Graphics. 2002, roč. 8, č. 3, strany 239–254. Dostupný z: http://www.pascucci.org/pdf-papers/IEEE-TVG-2002.pdf

[11] LINDSTROM, Peter, KOLLER, David, RIBARSKY, William, FAUST, Nick, TURNER, Gregory A. a HODGES, Larry F. Real–Time Continuous Level of Detail Rendering of Height Fields. *Proceedings of SIGGRAPH'96*. 1996, strany 109–118. Dostupný z: http://www.gvu.gatech.edu/people/peter.lindstrom/papers/siggraph96/siggraph96.pdf

[12] ROTTGER, Stefan, HEIDRICH, Wolfgang, SLUSALLEK, Philipp a SIEDEL, Hans–Peter. *Real–Time Generation of Continuous Levels of Detail for Height Fields*. Graphische Datenverarbeitung (IMMD9), Universitat Erlangen–Nurnberg, 1998. 8 stran. Dostupný z: http://www.vis.uni-stuttgart.de/~roettger/data/Papers/TERRAIN.PDF

[13] LOSSASO, Frank a HOPPE, Hugues. *Geometry Clipmaps: Terrain Rendering Using Nested Regular Grids*. 2004. 8 stran. Dostupný z: http://research.microsoft.com/copyright/accept.asp?path=http://research.microsoft.com/~hoppe/geomclipmap.pdf&pub=acm

[14] WAGNER, Daniel. *Terrain Geomorphing in the Vertex Shader*. Vienna University of Technology, 2003. 12 stran. Dostupný z: http://www.ims.tuwien.ac.at/media/documents/publications/Terrain_Geomorphing_in_the_Vertex_Shader.pdf

[15] *Coin Documentation 2.4.5a*. Oslo, Norsko, System in Motion, 2006. Průběžně aktualizovaný. Dostupný z: http://doc.coin3d.org/Coin/

[15] *SoQt Documentation 1.3.1a*. Oslo, Norsko, System in Motion, 2006. Průběžně aktualizovaný. Dostupný z: http://doc.coin3d.org/SoQt/