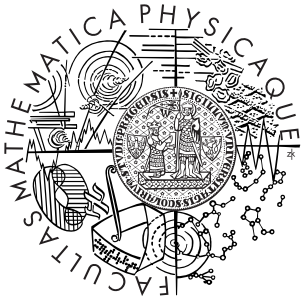
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Ondřej Šrámek

Zobrazování terénních dat v reálném čase

Katedra software a výuky informatiky

Vedoucí bakalářské práce: Mgr. Lukáš Maršálek
Studijní program: Informatika, Programování

2007

Na tomto místě bych rád poděkoval vedoucímu práce Mgr. Lukáši Maršálkovi za odborné rady a vstřícný přístup. Dále děkuji své přítelkyni a rodině, sice za podporu neodbornou, ale neméně důležitou.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 30. 5. 2007 Ondřej Šrámek

Obsah

Název práce: Zobrazování terénních dat v reálném čase
Autor: Ondřej Šrámek
Katedra: Katedra software a výuky informatiky
Vedoucí bakalářské práce: Mgr. Lukáš Maršálek
E-mail vedoucího: marsalek@cgg.ms.mff.cuni.cz

Abstrakt: V současné době je zobrazování rozsáhlých terénních dat aktuálním problémem v mnoha aplikacích, například interaktivních hrách a virtuálních prostředích. Pro jejich interaktivní zobrazení bylo potřeba implementovat výpočetně náročné techniky úrovně detailu, neboť grafické systémy nebyly schopné vykreslit dostatečně velké množství geometrie pro kvalitní reprezentaci celého terénu. Výkon současných systémů s ohledem na rasterizaci narostl natolik, že je možné během jedné sekundy vykreslovat stovky milionů trojúhelníků. Díky tomu se začínají objevovat nové práce založené na pozorování, že tento výkon je dostatečný k vykreslování trojúhelníků o velikosti pixelu při zachování rozumného výstupního rozlišení. Cílem práce bylo vytvořit praktickou implementaci založenou na některém takovém přístupu. Jako výchozí technika byl vybrán algoritmus Geometry Clipmaps a byla realizována jeho modifikovaná implementace. Implementované řešení je schopné na běžném stolním počítači zobrazit terén o velikosti téměř 300 milionů vzorků v reálném čase rychlostí průměrně 56 snímků za vteřinu.
Klíčová slova: 3D grafika, zobrazování terénu, Geometry Clipmaps

Title: Real-time visualization of terrain data
Author: Ondřej Šrámek
Department: Department of software and computer science education
Supervisor: Mgr. Lukáš Maršálek
Supervisor's e-mail address: marsalek@cgg.ms.mff.cuni.cz

Abstract: The rendering of huge terrain data is an actual problem in many application areas like interactive games or virtual environments. To keep rendering interactive it was necessary to implement computationally intensive LOD techniques, because the graphical systems were incapable of rendering enough geometry to represent the whole terrain accurately. Due to the raw processing power of current systems (with respect to rasterization) which enables to render hundreds of millions of triangles per second, new methods for rendering these data emerge. They are based on observation that this is enough to render pixel-sized triangles while keeping reasonable output resolution. The aim of this work was to develop a practical implementation based on such approach. The main algorithm chosen was Geometry Clipmaps and its modified version was created. The solution devoloped is capable of rendering almost 300M samples terrain in the real-time with avarage 56 frames per seconds.
Keywords: 3D graphics, terrain rendering, Geometry Clipmaps

Kapitola 1

Úvod

Jednou ze snah počítačové 3D grafiky je co nejvěrněji modelovat svět, jenž nás obklopuje. Mluvíme o tzv. *fotorealistickém* zobrazování. Simulujeme perspektivu, světlo, povrchy a další přirozené věci tak, aby nám náš mozek dvourozměrný obraz na monitoru interpretoval jako trojrozměrnou scénu. Cílem této práce je nahlédnout do podoblasti, která navozuje pocit pohledu na nějaký reálný, trojrozměrný terén.

1.1 Interaktivní a neinteraktivní zobrazování

Podle požadavků kladených na rychlost vykreslení rozlišujeme dva přístupy. Uveďme příklad: pokud automobilka navrhne nový automobil a chce zveřejnit jeho "fotografie", může si dovolit jeho velice kvalitní vykreslení, trvající i několik hodin. Jedná se o *neinteraktivní* zobrazování. Na druhou stranu, vývojáři her si do své závodní hry taktéž vytvoří model auta. Ti, kromě navození dojmu pohledu na 3D scénu, musí ale ještě navodit pocit, že se scéna pohybuje. Toho se dosahuje známým principem animace, tj. rychlým střídáním mírně pozměněných obrázků. Požadavek vývojářů na rychlé vykreslení je tedy velmi striktní. Aby se dalo mluvit o *interaktivitě*, či jinak taky vykreslování v *reálném čase*, musí být model vykreslen alespoň 24-krát[1] za vteřinu[2]. Nejenže tedy musí být ve hře jednodušší geometrický model, ale stejně tak musí být zjednodušeny i modely výpočetní (osvětlení, stíny, ...). Jak vypovídá samotný název práce, zaměříme se na zobrazování v reálném čase.

[1] literatura se v této hodnotě liší, měla by být ale v rozmezí 15-25

[2] tato jednotka se označuje jako počet snímků za sekundu a je pro ni zažitá anglická zkratka *FPS* (frames per second)

1.2 Motivace zobrazování výškových dat

Zobrazování výškových dat je poměrně dlouho známou[3] a přitom stále aktuální oblastí počítačové 3D grafiky. Moderní počítačové hry si žádají zobrazování stále větších detailnějších terénů a podporují tak další vývoj. Nelze ale chápat prosperující herní průmysl jako jediný důvod neustálého zájmu o toto téma. Možnosti využití jsou daleko širší. Vizualizace terénů jsou široce používány armádou pro plánování misí či trénink vojáků. Stejně tak piloti (a nejen ti vojenští) jsou zaučováni na simulátorech, které musí poskytovat co největší věrohodnost, a to i po stránce grafické. Dále lze zmínit filmový průmysl a jeho oblast speciálních efektů. Velký význam zobrazování trojrozměrných výškových dat pro GIS[4] zdůrazňuje např. práce [5].

1.3 Cíle práce

Pro vypracování této práce jsem si vymezil několik cílů.

1. Prostudovat některé starší techniky zobrazování výškových dat za účelem hlubšího proniknutí do tématu.
2. Implementovat moderní metodu pro zobrazování terénů.
3. Implementaci algoritmu provést nezávisle na aplikaci, tj. umožnit použití algoritmu třetí stranou.

[3] ze starších prací lze např. zmínit [4]

[4] grafické informační systémy

Kapitola 2

Zobrazování terénu

2.1 Reprezentace vstupních dat

Přirozeným a velice často používaným přístupem jak charakterizovat vstupní terén je tzv. *výškové pole*[1]. Jedná se o dvourozměrné pole výškových hodnot (souřadnic z), přičemž souřadnice x a y jsou reprezentovány indexy do tohoto pole. Samotná výškové pole nám ale nestačí pro přesné zobrazení terénu. Je potřeba jej doplnit o měřítko, a to jak měřítko samotných výškových hodnot, tak i měřítko zachycující vztah mezi indexem a reálnou souřadnicí x, resp. y. Případně lze volbu těchto měřítek ponechat plně v kompetenci aplikace.

Představme si toto výškové pole jako rastrový obrázek, kde jednotlivé výšky jsou zakódovány jako barvy (či pouze některé kanály barev) jednotlivých pixelů. Pokud se omezíme pouze na jeden kanál, můžeme si barvy představit jako stupně šedi. Nejníže položený bod bude mít tedy v příslušné bitmapě černou barvu a s rostoucí nadmořskou výškou bude plynule přecházet do bílé. Pokud je opravdu terén uložen v nějakém grafickém formátu, používá se často namísto označení výškové pole, označení *výšková mapa*[2]. Příklad takové mapy je uveden na Obrázku 2.1.

Alternativou k pravidelnému výškovému poli jsou *nepravidelné trojúhelníkové sítě*[3], neboli TIN. TIN umožňují reprezentovat terén pomocí menšího počtu vrcholů, které jsou po povrchu rozmístněny zcela libovolně. Každý je tedy zadán explicitně všemi třemi souřadnicemi (x, y, z), oznamujícími, že terén v bodě (x, y) dosahuje výšky z. Pro naprosto rovný povrch (nebo velmi málo zvlněný) se jedná o efektivní reprezentaci, TIN si u takového povrchu vystačí s velmi málo vrcholy.

[1] angl. *height field*
[2] angl. *height map*
[3] angl. *triangulated irregular networks*

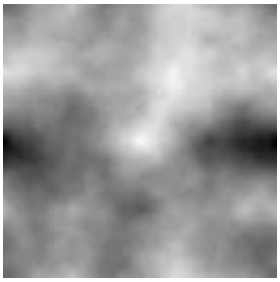
Je několik důvodů proč jsou výšková pole často upřednostňována před TIN [8]:

1. Pro uchování stejného počtu vrcholů je pravidelná mřížka paměťove výhodnější. Ve výškové mapě jsou pro každý vrchol uloženy pouze souřadnice z, kdežto u TIN je třeba uchování všech tří souřadnic.

2. Jednoduchost výškového pole umožňuje primitivní zjištění výšky v daném bodě. Díky tomu lze lehce zavést *ořezávání pohledovým jehlanem* (viz. Podkapitola 2.3) či řešení kolizí s terénem. U TIN jsou tyto problémy složitější.

3. Geografická data jsou šířena ve formátu pravidelné mřížky. Pro použití TIN je většinou nutná předzpracovávací fáze.

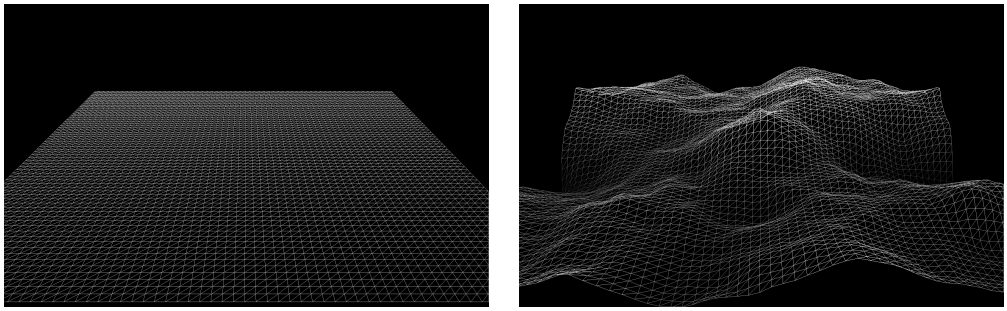
Na druhou stranu nespornou výhodou TIN je fakt, že lze pomocí nich reprezentovat kolmé stěny, převisy či jeskyně. Z povahy výškového pole je patrné, že takové přírodní zvláštnosti zachytit nemůže.



Obrázek 2.1: Výšková mapa s rozměry 64×64 pixelů

2.2 Triangulace a naivní zobrazování

Předpokládejme tedy, že mámě vstupní terén zadán pomocí výškového pole. V případě malého terénu je jeho zobrazení celkem nasnadě. V trojrozměrném prostoru se vytvoří pravidelná mřížka trojúhelníků ležící např. v rovině x, y (viz. Obrázek 2.2(a)). Poté jsou jednotlivým vrcholům jejich souřadnice z nastaveny dle hodnot přečtených z výškové mapy (Obrázek 2.2(b)). Oba kroky samozřejmě berou v potaz měřítko dané mapy. Říkáme, že jsme provedli *triangluaci* zadaného výškového pole, tedy charakterizování povrchu terénu pomocí množiny trojúhelníků. Zobrazit tuto množinu pomocí nějakého 3D API je již triviální. Tím, že při vytváření triangulace jsme použili opravdu všechny hodnoty výškového pole, vytvořili jsme nejpřesnější možné zobrazení vstupního terénu. Tento přístup se označuje jako zobrazení *"bru-*



(a) (b)

Obrázek 2.2: Vykreslení terénu zadaného výškovou mapou z Obrázku 2.1

tální silou"[4].

Jak bylo ale zmíněno, takový postup je možný pouze v případě malého terénu. Je zřejmé, že výškovou mapu o rozměrech $m \times n$ takto zobrazíme pomocí $2(m-1)(n-1)$ trojúhelníků. Představme si vstupní mřížku $10\,000 \times 10\,000$. Reprezentujeme ji pomocí přibližně 200 miliónů trojúhelníků. Takové množství nelze v reálném čase zobrazit ani na stávajících, velice výkonných grafických kartách.

2.3 Obecné optimalizační techniky

Základním primitivem moderní 3D grafiky jsou trojúhelníky. Ty lze, jak v OpenGL tak v Direct3D, zadat několika způsoby. Nejprimitivnějším a tedy nejméně efektivním způsobem je *seznam trojúhelníků* - každý jednotlivý trojúhelník je zadán 3-mi vrcholy. Lepší je využít společných hran a provést vykreslování pomocí *pásu* či *vějíře trojúhelníků*. Díky oběma docílíme určení n trojúhelníků pouze pomocí $n+2$ vrcholů[5]. Kromě přesouvání menšího množství dat je zvýšena i rychlost vykreslení - kromě prvního trojúhelníku jsou všechny další zadány již pomocí jediného vrcholu. Výpočet osvětlení nebo interpolace mohou tedy použít již dříve určených hodnoty pro oba vrcholy ležící na společné hraně. Je tudíž velice žádoucí, aby algoritmy zobrazující terén generovaly ne seznamy, ale buď pásy, nebo vějíře trojúhelníků.

Je zřejmé, že při konkrétním pohledu na terén nebude velké množství polygonů vůbec viditelné. Přesto polygony ale posíláme na grafickou kartu,

[4]angl. *brute force*

[5]podrobnější popis a názorné obrázky lze najít v každé OpenGL (resp. Direct3D) učebnici pro začátečníky

prochází celým vykreslovacím procesem a až např. v ořezávací fázi vůči obrazovce se zjistí, že vidět nejsou. Tento problém řeší tzv. *ořezávání pohledovým jehlanem*[6]. Pokud bychom ale testovali pozici každého polygonu vůči pohledovému jehlanu nic neušetříme, pouze přesuneme výpočet z GPU na CPU. Rozumným kompromisem je navrhnout systém nějakým způsobem dělící terén do větších celků (bloků), jež se poté vyplatí vůči pohledovému jehlanu testovat.

2.4 Techniky úrovně detailu

I přesto nám ale nezbývá, než se pro rozlehlé terény smířit s částečnou aproximací. Logickým požadavkem je, aby tato aproximace byla co nejméně viditelná. Přesně k tomuto účelu jsou určeny *techniky úrovně detailu*, které se i v našich krajích označují anglickou zkratkou LOD[7]. Hlavní motivací jsou tedy omezené možnosti grafických karet. Tyto techniky ale řeší i další problém. Při velké vzdálenosti zobrazovaných plošek se může stát, že se jich několik zobrazí do jednoho pixelu na obrazovce. To může při rasterizaci způsobit nežádoucí artefakty. Je tedy záhodno, aby úroveň detailu byla *pohledově závislá*, tedy aby více vzdálený terén byl více aproximován pomocí větších plošek. Tím je naplněn i náš požadavek, že aproximace by měla být pro pozorovatele co nejméně patrná.

Rozeznáváme dva typy technik úrovně detailu. *Spojitý* LOD je charakteristický tím, že zjednodušování je počítáno za běhu a úrovní detailu je teoreticky neomezeně. Opakem je *diskrétní* LOD, jež má model předpočítán v několika (konečně mnoha) různých úrovních detailu a algoritmus se pouze rozhoduje, kterou úroveň zobrazí. Diskrétní LOD nachází uplatnění v zobrazování obecných 3D modelů, ale pro terén je obvykle požadován LOD spojitý.

2.4.1 Praskliny a T-spoje

Se zavedením technik úrovně detailu často vyvstávají problémy se zajištěním spojitosti zjednodušeného terénu. Přesněji se tak děje na hraně, na níž se stýkají dvě oblasti s různou úrovní detailu. Je pravděpodobné, že úroveň s větší úrovní detailu má podél společné hrany větší počet vrcholů. Některé z nich na společné hraně neleží a způsobují tak zmiňovanou nespojitost. To označujeme jako *praskliny*. I kdyby tyto vrcholy na společné hraně ležely dochází k druhému artefaktu a tím jsou *T-spoje*. Jsou způsobeny tím, že tyto

[6] přesněji se jedná o komolý jehlan, jehož přesné rozměry a umístění jsou závislé na nastavení, místě a směru pohledu na scénu. Podrobnější popis je opět nad rámec této práce, lze jej nalézt např. v [11]

[7] level of detail

vrcholy nejsou sdíleny v hrubší úrovni. To může kvůli zaokrouhlovacím chybám vést k mírným rozdílům v osvětlení či interpolaci, a tyto spoje jsou při vykreslení viditelné. Různé LOD techniky se s těmito artefakty vypořádávají různě.

2.4.2 Přehled algoritmů

Při otázce zjednodušování terénů se často používají dvě struktury. Jsou jimy *binární trojúhelníký strom*[8] a *kvadrantový strom*[9]. Binární trojúhelníkový strom je strom, jehož uzly jsou tvořeny rovnoramennými pravoúhlými trojúhelníky. Potomky jsou trojúhelníky, jež vzniknou rozdělením uzlu kolmicí vztyčenou ve středu přepony. Vrcholy jednotlivých trojúhelníků jsou přímo mapovány na vzorky v pravidelné mřížce. Rozdělení uzlu tak vlastně značí přidání odpovídajícího vrcholu mřížky do výsledné zjednodušené triangulace. Kvadrantový strom je poté variantou binárního trojúhelníkového stromu s tím, že vrcholy jsou čtverce a každý uzel může mít až čtyři syny.

V oblasti úrovně detailu terénů bylo vynaloženo poměrně velké množství výzkumné práce. Následující velice krátký seznam několika technik v žádném případě nelze chápat jako výčet všech prací. Slouží pouze pro ilustraci jak se s průběhem doby problém zobrazování terénů řešil. Všechny uvedené algoritmy požadují vstupní terén ve formě výškového pole.

CLOD - spojitý LOD pro výšková pole Dle [8] se jedná o jednu z prvních pohledově závislých spojitých LOD technik pracujících opravdu v reálném čase. Byl představen v roce 1996 [6] a pro svou funkci využívá mimo jiné binárního trojúhelníkového stromu. Algoritmus pracuje stylem *zdola-nahoru*, tedy vstupní terén v nejvyším možném rozlišení je postupně zjednodošován spojováním sousedních trojúhelníků. Kritériem pro zjednodušování je přitom chyba, která by vznikla při zjednodušení. Je to rozdíl výšek vrcholu a místem v nižší úrovni detailu, kde právě onen vrchol chybí. Chyba je změřena v souřadném systému obrazovky, tzn. transformována a promítnuta do obrazovky (její jednotkou jsou tedy poté pixely). Míru zjednodušení může snadno uživatel ovlivnit pomocí nastavení *prahu*. Jedině pokud je chyba menší než práh může dojít ke spojení trojúhelníků. Polygony byly dále formovány do větších bloků provázaných v podobě kvadrantového stromu. Každý blok měl taktéž svou chybovou hodnotu, podle které se bloky buď ještě spojovaly nebo rozpadávaly.

ROAM [10] Tato technika [7] byla o rok později než CLOD, tj. v roce 1997.

[8] angl. *binary triangle tree*, zkracováno na *bintree*

[9] angl. *quadtree*

[10] real-time optimally adapting meshes

Algoritmus je založen na binárním trojúhelníkovém stromu a dvou prioritních frontách. Jednou, pro řízení zjednodušování terénu - operaci *split*(rozdělení), druhou pro přidávání detailu - operaci *merge* (spojení). Použití front umožnilo tzv. *snímkovou koherenci*, tedy možnost algoritmu využít triangulaci z minulého snímku a tu pouze pomocí několika operací poupravit. Priorita ve frontách byla převážně udávána velikostí chyby, kterou by operace způsobila (chyba byla stejně jako u CLOD měřené v souřadnicích obrazovky). ROAM se stal velmi oblíbeným algoritmem, a to snad i díky volně šiřitelné implementaci. Vzniklo několik modifikací z nichž nejznámější je tzv. *split-only* ROAM, který se dle [8] začal hojně používat v tehdejších hrách.

Geomipmapping Jedná se o metodu navrhnutou W. Boerem [9] v roce 2000. Je na ní patrný pokrok v rozvoji grafického hardware a tím pádem i používání více pravidelných struktur a omezení velkého množství testů. Algoritmus si vstupní terén rozděluje do bloků o velikosti $2^n + 1$, jež jsou hiearchicky uloženy ve formě kvadrantového stromu. Každý blok může být zobrazen v několika úrovních detailu. Nejdatailnější úroveň je tvořena blokem vykresleným s plnou triangulací (tedy jakoby brutální silou). Pokud triangulaci vytvoříme pouze na vrcholech na lichých pozicích dostaneme následující hrubší úroveň detailu (o velikosti $2^{(n-1)} + 1$). Stejným způsobem jsou vytvořeny i další nižší úrovně. Úroveň detailu, v níž nakonec bude daný blok zobrazen, je opět závislá na geometrické chybě v souřadnicích obrazovky.

SOAR [11] Algoritmus SOAR [10] (z roku 2001) opět s výhodou používá binární trojúhelníkový strom, jež je stejně jako u CLOD mapován na vrcholy výškového pole. Hledání optimální jednodušší varianty terénu se ale děje metodou *shora-dolů*, tedy celý terén je nejprve reprezentován dvěma trojúhelníky, každý z nich je kořenem jednoho binárního trojúhelníkového stromu. Postupným dělením jsou stavěny oba stromy a terén tak zpřesňován do doby, než jsou splněny požadavky zadané prahovou hodnotou chyby (měřena podobně jako u předešlých technik). V algoritmu je ořezávání pohledovým jehlanem použito již v této fázi, a proto jsou zpřesňovány pouze části terénu ležící ve směru pohledu. Algoritmus si taktéž dává záležet na generovaní trojúhelníkových pásů a ve výsledku lze celý terén zobrazit pomocí jediného pásu trojúhelníků.

Geometry clipmaps Vzhledem k předchozím pracím je Geometry Clipmaps poněkud odlišně koncipována. Na tuto metodu, pocházející z roku 2004, se zaměříme a je tedy blíže probrána v následující kapitole.

[11] stateless one-pass adaptive refinement

Kapitola 3

Geometry Clipmaps

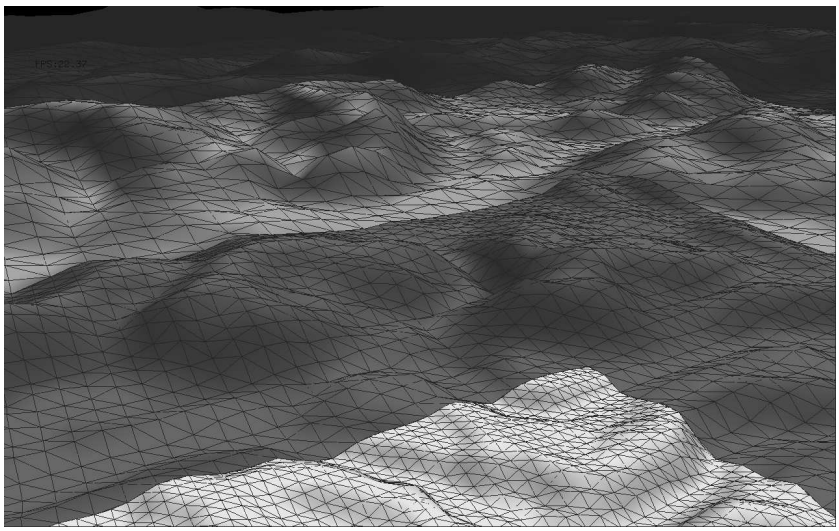
Tuto poměrně novou LOD techniku uvedli Hugues Hoppe a Frank Lossaso [1] v roce 2004. Autoři již v době vzniku předpokládali, že metodu bude možné v průběhu několika let implementovat na GPU. Předpoklad se částečně splnil již o rok později a byla publikována moderní GPU-implementace [2]. Přesunem výpočtů je dosaženo ještě vyššího výkonu a zároveň je ubrána zátěž z CPU, jež se může např. v počítačových hrách více věnovat herní logice, umělé inteligenci, atd.

Je třeba ale zmínit, že pojem *clipmap* a tedy i některé myšlenky tohoto přístupu, byly zavedeny již dříve v [3]. Tento článek pojednává o zobrazování rozsáhlých textur, ale jak již bylo nastíněno v části 2.1, na terén lze v jistém smyslu pohlížet jako na rastrový obrázek a tedy i texturu.

3.1 Základní popis algoritmu

Geometry clipmaps využívá velké propustnosti grafických karet. Předpokládá, že si lze již i v reálném čase dovolit vykreslení terénu v takové podobě, kdy trojúhelníky po provedení všech transformací a promítnutí na obrazovku, budou mít přibližně velikost jednoho pixelu. Cílem bylo tedy navrhnout systém, pracující s terénem jako s texturami (nutné pro následnou GPU implementaci), a který by díky pohledové závislosti dosahoval zhruba 1-pixelových trojúhelníků. Úroveň detailu je závislá čistě na vzdálenosti od místa pohledu, není vůbec závislá na geometrii terénu jak jsme mohli vidět u některých dřívějších technik.

Pro lepší pochopení následujících odstavců je na Obrázku 3.1 zachycen terén zobrazený touto metodou. Jak napovídá samotný název metody, algoritmus je založen na struktuře zvané clipmapa. Její krátkou definice lze nalézt v [3]:



Obrázek 3.1: Terén zobrazený metodou geometry clipmaps

"A clipmap is an updatable representation of a partial mipmap, in which each level has been clipped to a specified maximum size."

„Clipmapa je lehce aktualizovatelná reprezentace části mipmapy, v níž je každá úroveň ořezaná na danou maximální velikost."

Aktuálně zobrazovaný terén je reprezentován pomocí jednotlivých vnořených úrovní clipmapy centrovaných okolo pozorovatele. Úrovně tedy odpovídají pravidelným (čtvercovým) mřížkám, přičemž všechny mají stejný počet vrcholů, ale liší se rozestupem mřížky a tedy i rozlohou terénu, který reprezentují. Počet úrovní clipmapy označme pro další potřeby m, rozměr mřížky n, úrovně se číslují tak, že nejhrubší označujeme pořadovým číslem 0, nejdetailnější číslem $m-1$. Pro $n = 5, m = 4$ situaci zachycuje Obrázek 3.2.

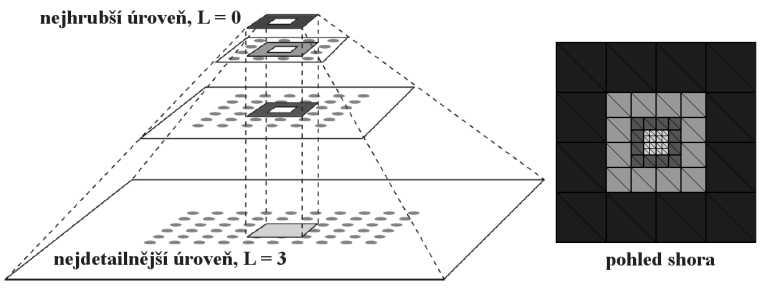
Během pohybu pozorovatele jsou mřížky přesouvány tak, aby byl pozorovatel stále uprostřed. Ve směru pohybu jsou tudíž objeveny části terénu ve větším detailu, proti směru pohybu části upadají do nižšího. Z toho pramení i nutnost realizovat část definice *"lehce aktualizovatelná reprezentace"*. Toho je docíleno pomocí *toroidního přístupu* k hodnotám mřížky[1]. Ten umožňuje přesně to co potřebujeme - bez jakéhokoli přesouvání stále viditelných dat, pouhé přepsání dále nepotřebných, daty nově objevenými. Nová data jsou načítána z mipmapy terénu. Tu si vypočítáme v předzpracovávací fázi běžným postupným zmenšováním vstupní výškové mapy na poloviční velikost.

[1] lehce implementovatelný aplikováním operace *mod* na indexy mřížky

Při tomto úkonu také určíme číslo m, tedy počet úrovní mipmapy a clipmapy. Úrovní musí být přinejmenším tolik, aby nejhrubší úroveň clipmapy obsahovala celou úroveň mipmapy, jinak také, aby mipmapa měla na dané úrovni rozměry menší než $n \times n$.

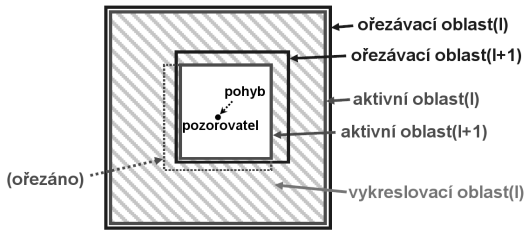
3.2 Aktivní, ořezávací a vykreslovací oblasti

Pro každou úroveň, přesně dle původního článku, definujeme několik obdélníkových oblastí. *Ořezávací oblast* je rozsah oblasti (měřeno ve světových souřadnicích), pro níž máme v dané úrovni momentálně načtená data. *Aktivní oblast* má stejné rozměry a je neustále centrována okolo místa pohledu. Značí část terénu, kterou bychom v ideálním případě chtěli mít pro úroveň načtenou. Snahou tedy je načítat nová data tak, aby ořezávací oblast byla totožná s aktivní. Při rychlém pohybu to někdy může být problém. Proto má Geometry Clipmaps na každou aktualizaci terénu vymezen určitý maximální počet, značící kolik nejvíce nových vrcholů je možné během jedné aktualizace načíst. V článku je tato mez stanovena na n^2. Aktualizace probíhají od nejhrubší úrovně k detailnějším. Ve smyslu oblastí vlastně necháme ořezávací oblast jakoby "pozastávat" za aktivní. Protože ale poté nemůžeme chtít zobrazit data, jež nemáme v dané úrovni načtená, aktivní oblast je přenastavena na její průnik s ořezávací oblastí (dochází k tzv. *ořezu*). Toto jednoduše zajistitelné omezení vytváří užitečnou vlastnost algoritmu - i při rychlém pohybu je udržována přibližně konstantní rychlost vykreslování. Další oblast, taktéž definovaná pro každou úroveň l, je označována jako *vykreslovací*. Má tvar proděravělého obdélníku, přičemž vnější obvod je tvořen aktivní oblastí l a vnitřní obvod je tvořen aktivní oblastí úrovně $l + 1$ (detailnější). Pro nejdetailnější úroveň má vykreslovací oblast tvar klasického obdélníku. Oblasti včetně příkladu ořezu jsou názorně ukázány na Obrázku 3.3. Během jakéhokoliv posunu některé z oblastí je potřeba zajistit platnost čtyř omezení přesně definovaných v [1]. Omezení zajistí návaznost jednotlivých



Obrázek 3.2: Mipmapa terénu a výsledné zobrazení pomocí jednotlivých vnořených úrovní. Překresleno z [2].

úrovní, dále funkčnost predikce a přechodových vrstev (probírány dále).



Obrázek 3.3: Oblasti definované v rámci úrovní Geometry Clipmaps. Překresleno z [1].

Zobrazením všech vykreslovacích oblastí docílíme vykreslení celého terénu pomocí pravidelných vnořených mřížek, jak naznačuje samotný název práce [1].

Pokud bychom nad terén stoupali kolmo vzhůru, stane se, že trojúhelníky v nejdetailnějších úrovních budou tak malé, že po rasterizaci jich několik bude určovat barvu jediného pixelu. Jak bylo již řečeno v Podkapitole 2.4, LOD techniky by měly zajistit aby k takovým jevům nedocházelo. Řešením Geometry Clipmaps je, pro každou úroveň clipmapy kontrolovat vzdálenost jejího nejvýše položeného bodu od z-ové souřadnice kamery. Pokud je tato vzdálenost větší než $(0,4)ng_l$ přestává být nadále viditelnou. Hned, jak podmínka přestane platit, úroveň se stává znovu viditelnou.

3.3 Zajištění spojitosti

Stejně jako většina technik využívajících pravidelných mřížek v různých detailech, tak i Geometry Clipmaps se potýká s problémy jako praskliny a T-spoje. Praskliny jsou řešeny pomocí *přechodových oblastí*, jež se nacházejí v každé úrovni podél vnějšího obvodu vykreslovací oblasti. Výšková data v této oblasti lineárně přecházejí mezi výškovými datami úrovně l do výškových dat úrovně hrubší, tedy $l-1$. Výšky jsou upraveny tak, že přímo na rozhraní mezi dvěma úrovněmi se data z obou úrovní rovnají, čímž dochází k eliminaci prasklin.

Aby bylo počítání přechodů efektivní, je mu přizpůsobena samotná struktura vrcholů, kterými jsou tvořeny mřížky. Každý totiž kromě klasických souřadnic (x,y,z) v sobě nese i čtvrtou hodnotu z_c, jež značí výšku vrcholu o souřadnicích (x,y) v následné hrubší úrovni. Díky tomuto lze problematiku přechodů snadno implementovat ve *vertex shaderu*. Podrobnější popis

a přesný vzorec výpočtu lze opět najít v [1]. T-spoje jsou řešeny pásem degenerovaných trojúhelníků na vnějším obvodu každé vykreslovací oblasti.

3.4 Syntéza a komprese

Hierarchická struktura Geometry Clipmaps umožňuje ze znalosti úrovně l *predikování* detailnější úrovně $l + 1$. Díky tomu nemusí být v clipmapě uloženy samotné výšky, ale pouze *rezidua*, tedy rozdíly výšek mezi predikovanou hodnotou a výškou samotnou. Neboť rezidua jsou v daleko menším rozmezí než samotné výšky, lze rezidua uložit do menších datových typů. Ještě většího ušetření paměti lze dosáhnout pomocí *komprese* reziduí[2]. Požadavkem na použitou kompresi je podpora tzv. *oblastí zájmu*[3], což je vlastně možnost *dekomprese* pouze zadané oblasti obrázku. To umožňují např. formáty JPEG2000 či PTC, který použili autoři původního článku. Právě díky kompresi je možné si uložit i terény značně velkých rozměrů čistě v operační paměti počítače.

Dále metoda představuje možnost zavedení *syntézy*. Může být aplikována na detailnější úrovně a znamená, že rezidua nejsou čerpána ze samotné předpočítané mipmapy terénu, nýbrž jsou náhodně generována. To se může zdát jako zbytečné znepřesnění vstupního terénu, ale výsledky [1] vypovídají o opaku. Základní vzhled terénu je dobře zadán pomocí hrubších úrovní a drobné nerovnosti náhodný (ale pro stejný terén deterministický) šum dobře nahradí.

3.5 Implementace v GPU

Již rok po vydání původního článku byl publikován popis implementace Geometry Clipmaps, hojně využívající programovatelnou grafickou jednotku [2]. Na trh byly totiž tou dobou uvedeny grafické karty podporující shadery modelu 3.0. Ty jako jednu z novinek uvedly možnost přístupu k texturám i z vertex shaderů[4], což umožnilo přesun výpočtů do vertex shaderů. Jedinou z hlavních částí algoritmu, která zůstala nadále počítána pomocí hlavního procesoru, byla dekomprese.

[2] výškové mapy poskytují díky své koherenci, ještě větší prostor pro kompresi než běžné obrázky [1]

[3] angl. *regions of interest* - ROI

[4] označováno jako *vertex textures*

Kapitola 4

Implementace

4.1 Cíle implementace

Cílem projektu je vypracování softwaru zobrazující reálná výšková data v trojrozměrném prostoru pomocí metody Geometry Clipmaps. Vstupem je výškové pole, které může být zadáno v podobě výškové mapy v běžném grafickém formátu (.raw, .png). Je také podporován formát .hgt, v němž jsou distribuována data získaná během mise SRTM[1]. Po zobrazení terénu má uživatel absolutní volnost pohybu a možnost náhledu na terén z jakéhokoli úhlu a vzdálenosti. Výsledná aplikace je určena pro běh v operačním systému MS Windows XP. Při implementaci jsem se snažil, aby algoritmus zobrazující terén nebyl v aplikaci pevně zakomponován, a bylo možné jeho co nejsnadnější použití i jinou aplikací.

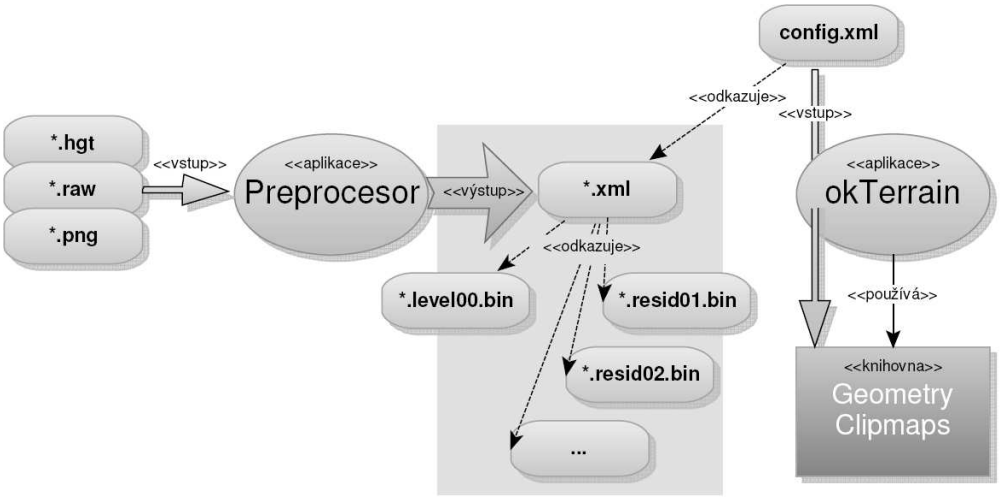
4.2 Analýza

Jako u většiny grafických aplikací je třeba i u tohoto projektu důsledně dbát na rychlost výsledné aplikace. Zato složitost uživatelského rozhraní je značně omezena. Vstupní data a nastavení výsledné aplikace mohou být lehce charakterizována pomocí parametrů programu, nebo pomocí jednoduchého konfiguračního souboru. Pohyb během zobrazování terénu lze vyřešit jako reakci na stisk klávesy nebo pohyb myši. Z důvodu rychlosti a nevyžadování GUI se tedy jako vhodné programovací jazyky pro implementaci nabízejí C nebo C++. Byla předpokládána přece již větší rozsáhlost projektu, pro pozdější možnost snadného rozšíření bylo zvoleno C++. Koncovou platformou je systém Windows - jako 3D API přichází tedy v úvahu

[1] cílem bylo zmapování převážné části zemského povrchu s přesností 1", to odpovídá rozestupu mřížky přibližně 30 metrů, v době psaní práce byly data volně přístupná na ftp://e0srp01u.ecs.nasa.gov/srtm

jak OpenGL, tak i Direct3D. Abychom nezatracovali možnost budoucího přenosu na jiný operační systém, bylo použito OpenGL.

Algoritmus je založen na metodě Geometry Clipmaps popsaný v Kapitole 3. Jelikož požadavkem je možné použití i v jiné aplikaci, bude umístěn v samostatné staticky linkované knihovně. Pro metodu Geometry Clipmaps je nutná předzpracovávací fáze, při níž je vstupní terén podvzorkováván a jsou počítána rezidua. Tato fáze je poměrně časově náročná. Z toho důvodu je výhodnější předzpracování oddělit od zbytku algoritmu a vytvořit pro tento účel samostatnou aplikaci. Konečně posledním produktem je demonstrační aplikace okTerrain, využívající knihovny Geometry Clipmaps. Tento hrubý návrh, včetně detailů probíraných dále, je zachycen na diagramu v Obrázku 4.1.



Obrázek 4.1: Moduly, soubory a jejich interakce

4.3 Použité knihovny

Při implementaci bylo mimo OpenGL použito několik knihoven:

FreeGLUT - Volně šiřitelná verze knihovny GLUT[2]. Je to nadstavba OpenGL, která obstarává vytvoření okna, reakci na pohyb myši nebo stisk klávesy, časovač, atd.

FreeImage - Podpora vstupu a výstupu obrázků v rozličných grafických formátech (TGA, PNG, JPEG, ...)

[2]OpenGL Utility Toolkit

GLEW - Usnadňuje kontrolu podpory jednotlivých rozšíření OpenGL a umožňuje jejich snadné použití.

MathGL++ - Zapouzdřuje vektory a matice v 3D prostoru. Je přímo určena pro spolupráci s OpenGL.

TCLAP - Definuje rozhraní pro snadnou definici argumentů programu a přístup k nim.

TinyXML - Čtení/vytváření XML dokumentů.

Licence k jednotlivým knihovnám jsou k nalezení v souboru `licences.txt` na přiloženém disku CD-ROM. Všechny knihovny jsou platformově nezávislé.

4.4 Preprocesor

Je realizován konzolovou aplikací okPreproc provádějící předzpracování dat pro algoritmus Geometry Clipmaps. Pomocí parametrů (viz. Dodatek B.3) je preprocesoru předána cesta ke vstupnímu souboru a atributy zpracování. Jedním z povinných parametrů je například n, tj. rozměr clipmapy. Dle postupu v 3.1 aplikace určí počet úrovní a sestrojí mipmapovou pyramidu zadaného terénu. Tu poté procházíme od nejhrubší úrovně a pomocí filtru popsaného v [1] se snažíme predikovat úrovně detailnější. Rezidua, tedy rozdíly mezi predikovanou hodnotou a skutečnou hodnotou, ukládáme do souborů. Pokud k těmto souborům přidáme výšková data v nejhrubší úrovni, dostáváme reprezentaci terénu ve všech úrovních. Všechny nastavení a odkazy na jednotlivé soubory jsou charakterizovány vygenerovaným výstupním XML souborem. Ten později poslouží jako inicializační soubor pro modul Geometry Clipmaps.

4.5 Geometry Clipmaps

Snahou práce nebylo implementovat tuto metodu *naprosto* přesně podle [1]. Cílem bylo vyvinout algoritmus vhodný pro naše dané zadání a tedy zvážit použití jednotlivých částí Geometry Clipmaps, příp. navrhnout přístupy jiné. Většina odlišností je motivována menšími rozměry zobrazovaného terénu.

První odlišností je nepoužití komprese a ponechání reziduí v nezkomprimované podobě. Hrozí nám, že rezidua přesáhnou velikost operační paměti. Vypočtěme tedy horní odhad paměťové náročnosti pro daný počet vzorků

c (nechť jednotkou jsou stovky milionů vzorků). Znovu připomeňme, že rezidua díky malému rozsahu mohou být uložena v datovém typu o velikosti pouze 1B. Použijeme taky tvrzení známé u práce s texturami - při vygenerování všech úrovní mipmapy pro danou texturu, zvětší se její paměťová náročnost o 33%[3] S převodem na MB dostáváme:

$$\frac{c \cdot 10^8}{2^{20}} \cdot \frac{4}{3} \doteq 127c \; MB \tag{4.1}$$

Například zobrazování (již poměrně rozsáhlého) terénu o rozměrech 20 000× 20 000 (tedy $c = 4$), si vyžádá 508MB operační paměti. Takové množství paměti je při dnešním hardware akceptovatelné. Pro úplnost bychom ještě měli určit paměť nutnou k uchování samotné geometrie. Ta je pro terény námi uvažovaných rozměrů v řádu pouze jednotek megabajtů. Data jsou navíc často uloženy v paměti samotné grafické karty. Z výpočtů vyplývá, že komprese tedy není pro naše účely nezbytně nutná.

Změna oproti [1] byla taktéž učiněna ve změně mřížkování a měřítku. Článek navrhuje rozestup mřížky l-té úrovně (označme g_l) nastavit roven 2^{-l}. Jedná se o velikost ve světových souřadnicích, v nichž poté probíhají veškeré výpočty a ořezávání jednotlivých oblastí (viz. Podkapitola 3.2). V naší implementaci za účelem převedení výpočtu na celá čísla je nastaveno $g_l = 2^{m-l-1}$, přičemž m opět značí počet úrovní, nejhrubší je značena $l = 0$, nejdetailnější potom $l = m - 1$.

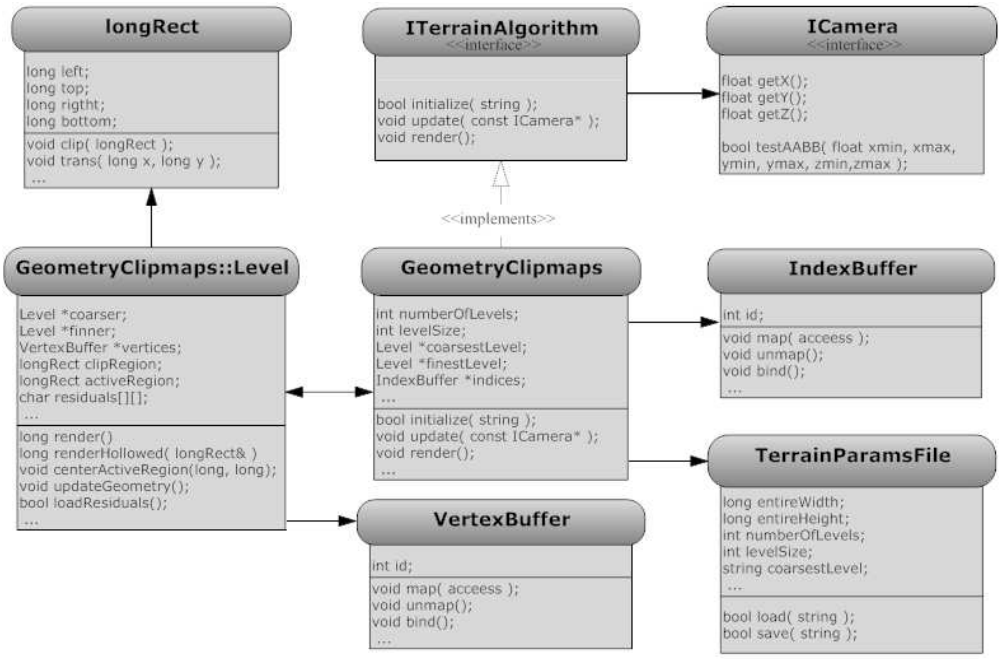
Jiným přístupem je řešeno stínování a texturování terénu. V původní práci je terén obarvený pouze na základě výšky. Za běhu je poté počítána normálová mapa[4]. Našim řešením je potažení terénu reálnou texturou ze vstupu. Grafické karty mají rozměry textur omezeny, běžnými jsou maximální rozměry např. 4 096 × 4 096. Při našich rozměrech geometrie terénu okolo $16k \times 16k$, dostáváme přibližně jeden *texel* na čtyři vrcholy. To není pro dobrý vzhled terénu dostačující, a proto je terén pokryt druhou texturou, tzv. *detailní mapou*. Ta může mít i menší rozměr a na terén je kladena vícekrát za sebou. Zatímco hlavní textura udává obarvení celého terénu, detailní mapa odlišuje sousední plošky, jimž byla přiřazena stejná barva.

Algoritmus je zapouzdřen ve třídě `GeometryClipmaps` a implementuje jednoduché rozhraní pro použití v konkrétní aplikaci. Na Obrázku 4.2 je uveden

[3] ve skutečnosti pyramida v našem případě nebude zcela kompletní (bude mít pouze m úrovní - viz. 3.1) a náročnost bude tedy o něco menší. Jelikož se jedná o horní odhad toto malé zaokrouhlení (v řádu několika MB) si můžeme dovolit.

[4] tzn. při načítání nových vrcholů do clipmapy jsou zároveň počítány jejich normály a jsou kódovány do textury

zjednodušený diagram tříd, podrobnější informace lze vyčíst v referenční příručce na přiloženém CD.



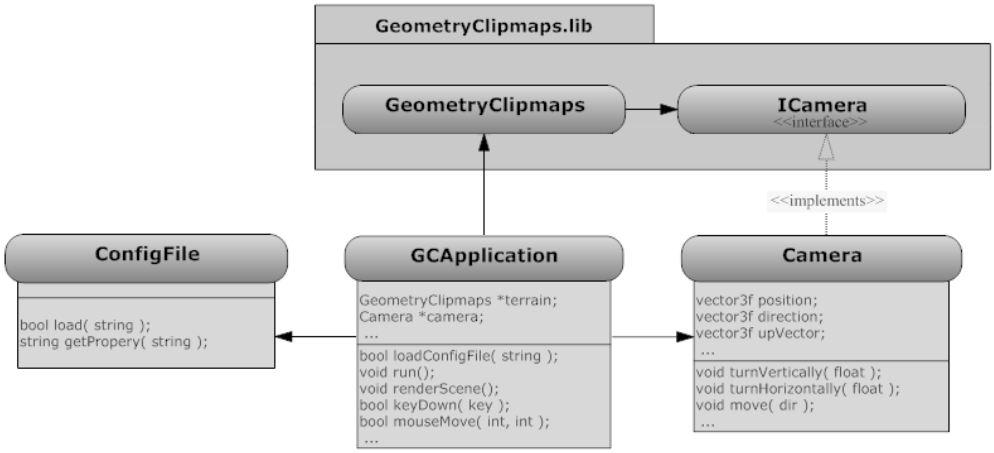
Obrázek 4.2: Struktura modulu Geometry Clipmaps

Konečnost terénu I když je tato LOD technika v původním článku, z hlediska teorie dobře popsána, neposkytuje příliš mnoho rad pro efektivní implementaci metody. V části 3.2 jsme zmínili nutnost zachovávat čtyři omezení, jež musí platit pro jednotlivé oblasti definované pro každou úroveň. To se ale na okraji terénu stává problémem. Článek se o problému vůbec nezmiňuje. Naskýtají se dvě možnosti řešení. Buď porušíme některá omezení, čímž si způsobíme problémy při predikování dat a narušíme funkčnost přechodových vrstev. Tyto problémy by v tom případě bylo nutné eliminovat pomocí různých ošetření, která by ale mohla mít vliv na rychlost algoritmu. Druhou variantou je omezení pevně dodržet a nevyžadovat na okrajích přesné centrování aktivních oblastí okolo místa pohledu. To způsobí, že terén na okrajích bude v nižším detailu než by si vyžadovala pozice kamery. Pro naši implementaci byla zvolena tato druhá možnost.

4.6 Demonstrační aplikace

Jako ukázka použití modulu Geometry Clipmaps byla vytvořena aplikace okTerrain. Její hlavní třída `GCApplication` inicializuje grafický režim, vytváří instanci třídy `GometryClipmaps` a předává jí vstupní soubor.

Pomocí funkcí knihovny GLUT třída reaguje na vstup uživatele a zajišťuje tak pohyb kamerou (instance třídy `Camera`). V hlavní smyčce jsou poté opakovaně volány metody `render` a `update` dané instance třídy `Geometry Clipmaps`. Metodě `update` je nutné jako parametr předávat ukazatel na instanci kamery. Názorný diagram tříd je uveden na Obrázku 4.3.



Obrázek 4.3: Struktura demonstrační aplikace okTerrain

4.7 Výsledky

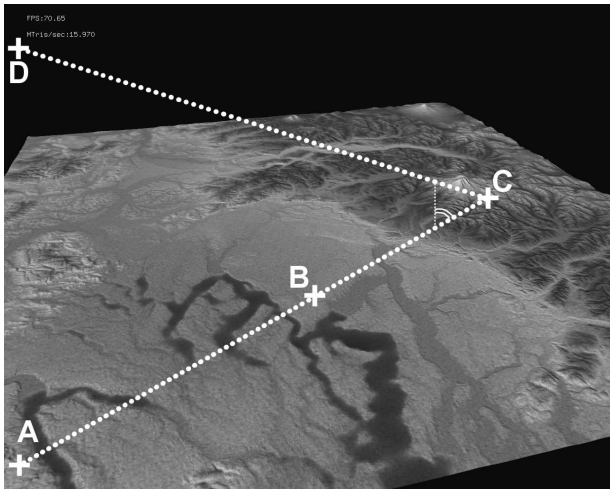
Implementovaný algoritmus byl analyzován na vstupních datech o třech různých rozměrech. Ve všech případech se jednalo o výškovou mapu oblasti *Puget Sound*[5]. Jednotlivé rozměry jsou uvedeny v následující tabulce:

Terén	Rozměry výškové mapy	Paměťová náročnost
T_1	$1\,025 \times 1\,025$	1,6MB
T_2	$4\,097 \times 4\,097$	22,4MB
T_3	$16\,385 \times 16\,385$	358MB

Pohled na data byl řízen automatickou kamerou, trajektorie pohybu je znázorněna na Obrázku 4.4. Pohyb terénem lze rozdělit do šesti, plynule na sebe navazujících, úseků:

1. Statický pohled na terén z bodu A.
2. Průlet konstantní rychlostí relativně nízko nad terénem do bodu B. Terén spíše rovinného charakteru.

[5]oblast v okolí Washingtonu, v době psaní práce byla data volně přístupná na http://www-static.cc.gatech.edu/projects/large_models/ps.html



Obrázek 4.4: Trajektorie automatické kamery

3. V bodě B otočení o 360° v horizontálním směru.

4. Pohyb konstantní rychlostí do bodu C, terén více zvlněný.

5. V bodě C natočení kamery asi o 30° dolů a poté rychlejší pohyb do bodu D. Kamera stále zaměřena na bod C.

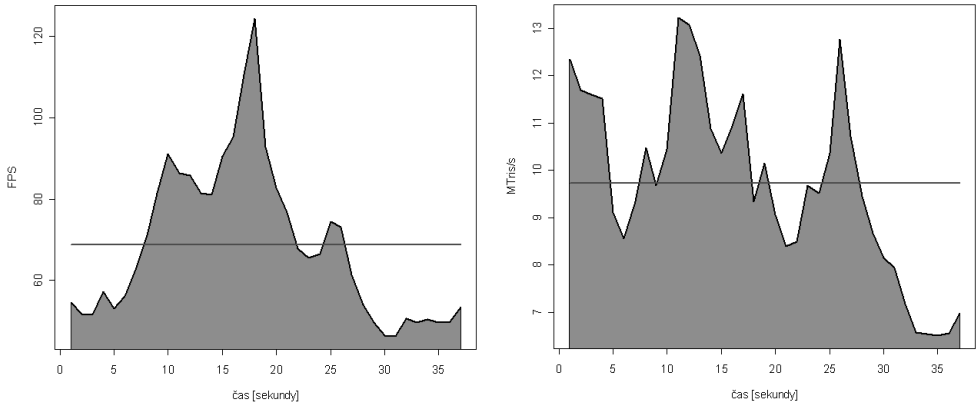
6. Dosažením bodu D měření končí. V tomto bodě byl viditelný téměř kompletní terén.

Testování probíhalo na stanici s 64-bitovým procesorem AMD Opteron 144, frekvence 1,81 GHz, cache: L1 - 128kB, L2 - 1MB, s operační pamětí 960 MB a grafickou kartou nVidia GeForce 6150. Všechny testy probíhaly při rozlišení 800×600. Výsledky jsou zachyceny v následujících Obrázcích 4.5–4.7[6]

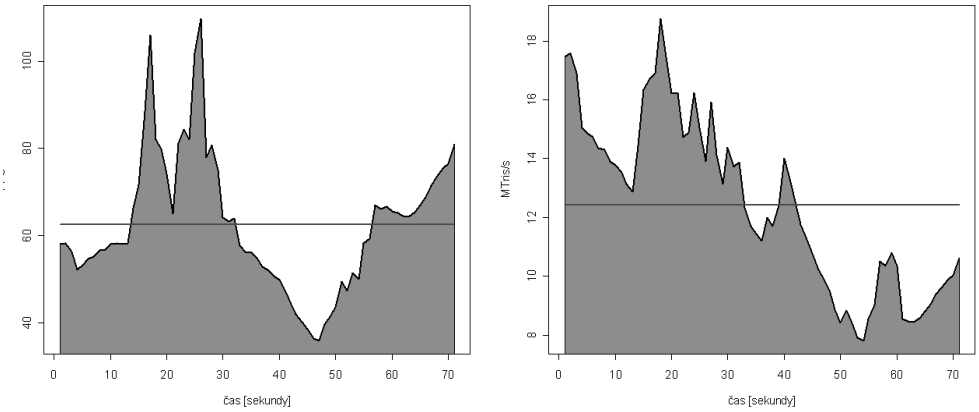
Průměrné hodnoty jsou zvýrazněny v grafech vodorovnou čárou, jejich přesné hodnoty jsou přehledně zachyceny v tabulce:

Terén	Průměr FPS	Průměr M△/s
T_1	67,49	9,56
T_2	62,03	12,32
T_3	56,16	13,38

[6]na obrázcích vlevo je zachycena rychlost snímků za vteřinu, na obrázcích vpravo poté počet vykreslených trojúhelníků za vteřinu (v milionech). Obě veličiny jsou měřeny v závislosti na čase od spuštění aplikace.



Obrázek 4.5: Výsledky měření, terén T_1, $n = 257$, $m = 3$

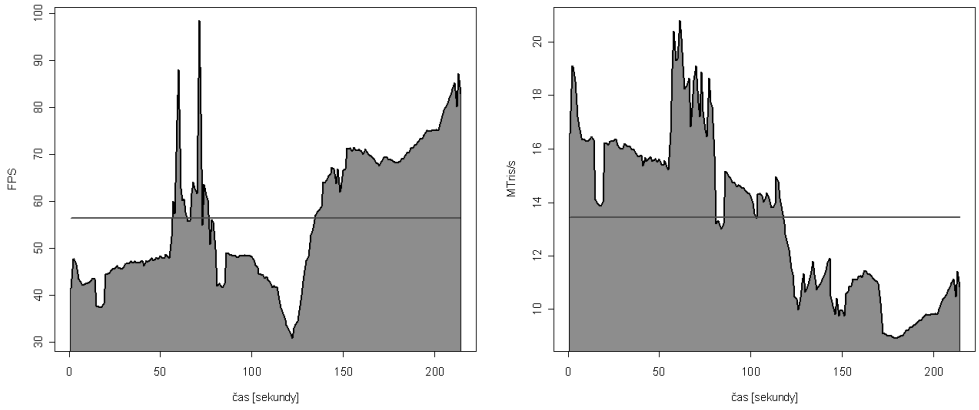


Obrázek 4.6: Výsledky měření, terén T_2, $n = 257$, $m = 5$

Z grafů lze vypozorovat několik zajímavostí:

1. Asi ve třetině času dochází u všech konfigurací ke značnému nárůstu vykreslovací rychlosti. Tento časový úsek odpovídá otáčení kamery v bodě B. Metoda Geometry Clipmaps aktualizuje geometrii terénu pouze při pohybu, a proto, při pouhé změně směru pohledu podává vynikající výsledky.

2. Zvláštní výsledky vykazují hodnoty v závěrečné fázi měření. Při rychlém vzdalování se od terénu, prokazuje třetí test až dvakrát lepší výsledky než test první. Toto chování je způsobeno podmínkou testující vzdálenost místa pohledu od jednotlivých úrovní (viz. 3.2). Dalo by se očekávat, že výsledky budou pro oba terény přibližně stejné. Větší rychlost při vykreslování T_3 je překvapující.

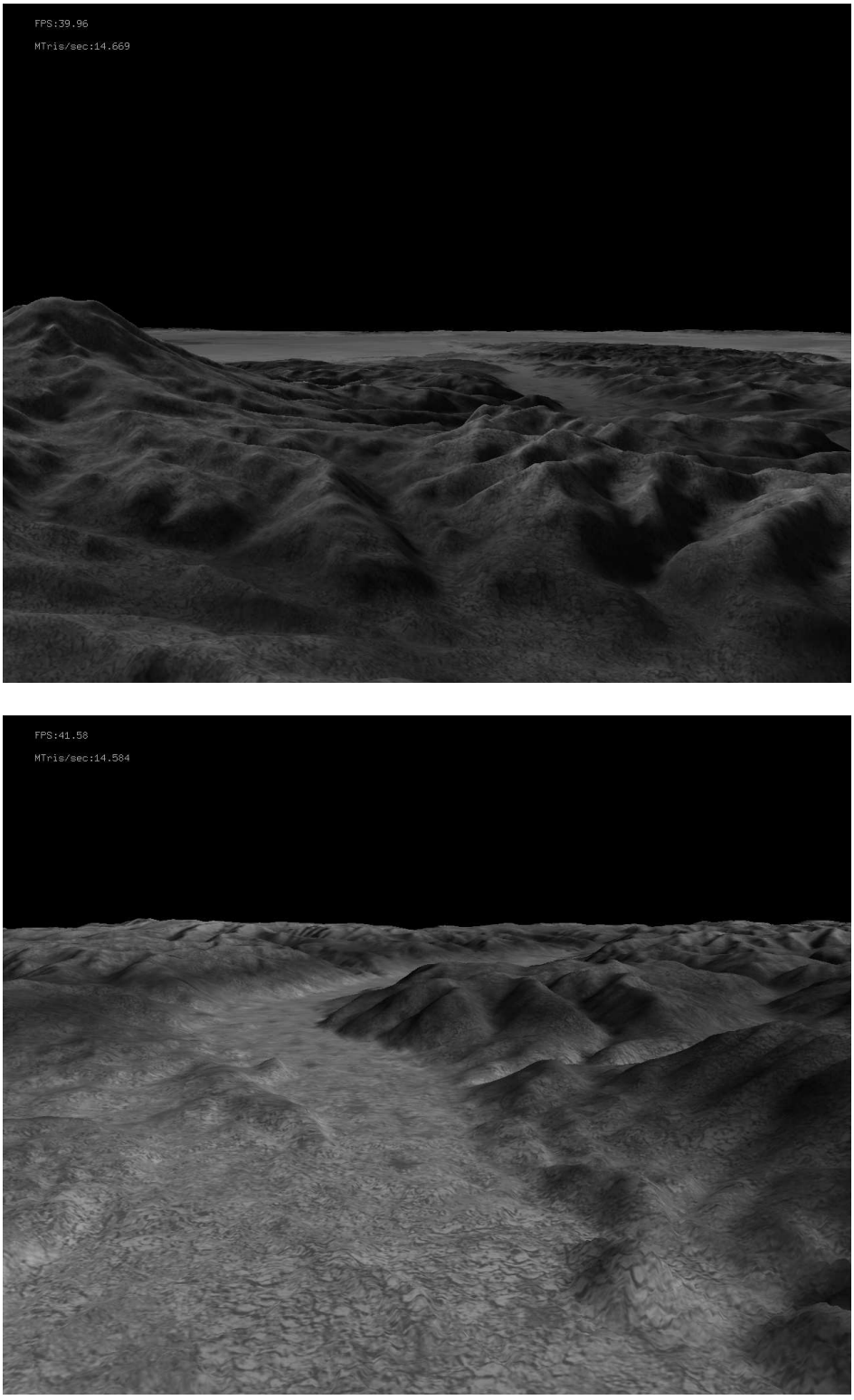


Obrázek 4.7: Výsledky měření, terén T_3, $n = 257$, $m = 7$

3. Pův. článek [1] uvádí, že metoda Geometry Clipmaps zajišťuje „téměř konstantní rychlost vykreslování“. Pro naši konkrétní implementaci, data a pohyb se toto tvrzení jeví jako neplatné.

4. Test T_1 je vzhledem k malé rozloze terénu hodně pomalý a pravděpodobně i "brute-force" algoritmus by poskytl lepší výsledky. Nezmínili jsme se totiž o *bottle necku*, metody Geometry Clipmaps programované pomocí CPU. Při každém snímku jsou totiž znovu přepočítávány indexy všech trojúhelníků. Předpokládám, že výsledky T_1 jsou zapříčiněny právě jím.

5. Na druhou stranu se ale podívejme na rozdíly mezi po sobě jdoucími testy - počet vzorků v terénu se zvětšil 16-krát, zatímco průměrné FPS kleslo o pouhých 6 FPS. Tento výsledek považuji za velmi překvapující.



Obrázek 4.8: Ukázka pohledu na terén T_2

Kapitola 5

Závěr

5.1 Shrnutí

V první kapitole jsme se povrchně seznámili s problematikou a motivací zobrazování terénních dat. Definovali jsme interaktivní a neinteraktivní zobrazování a uvedli, kterým směrem se práce vydává. Byly stanoveny cíle práce. Druhá kapitola nás uvedla hlouběji do tématu. Charakterizovali jsme nejpoužívanější vstupní reprezentace terénů a osvětlili základní přístup k jejich zobrazení. Vysvětlili jsme, proč nám tento naivní přístup nedostačuje, a plynule jsme překročili k tématu LOD technik. Po podrobném popisu jsme, v rámci splnění jednoho z cílů práce, uvedli chronologický přehled několika oblíbených technik. Jedné konkrétní z nich, metodě Geometry Clipmaps, jsme věnovali celou Kapitolu 3. Stejnou metodu jsme v rámci práce implementovali a podrobně probrali v Kapitole 4. V rámci ní jsme provedli analýzu a představili finální strukturu implementační části práce. Nakonec byly výsledné aplikaci předkládána různá data a měřena rychlost jejich zobrazování. Naměřené výsledky jsme také okomentovali.

5.2 Splnění cílů

V rámci práce byl implementován algoritmus založený na moderní metodě Geometry Clipmaps. Povedlo se výborně eliminovat praskliny na přechodu různých úrovní detailu, čímž byla zajištěna důležitá spojitost a vizuální návaznost terénu. Dle původního plánu je samotný zobrazovací algoritmus oddělen od demonstrační aplikace, což jej činí snadno použitelným aplikacemi jinými. Pomocí série testů byla prokázána jeho zaměřenost na rozlehlé terény a naopak značná neefektivita při zobrazování terénů malých. Jako obzvláště potěšující považuji výsledky z předchozí kapitoly pro testovací data T_3. Zdá se, že průměrný počet snímků za sekundu při rostoucí velikosti terénu klesá velmi nepatrně. Jako omezení může působit množství potřebné operační paměti. Při stavu dnešního hardware je však již na běžných stanicích možné

načíst dosti rozsáhlé terény.

Nabízí se také srovnání se zmiňovanou GPU-implementací [2]. Ta může být sice rychlejší, ale to na úkor flexibily a rozšiřitelnosti. Současné vícejádrové systému mohou navíc rozdíl v rychlosti minimalizovat.

5.3 Diskuze

Během vypracovávání práce jsem na dospěl k několika vlastním závěrům. Překvapilo mě, že problematika zobrazování terénů je mnohem komplexnější téma, než jsem si kdykoliv před psaním práce myslel. Stala se mi příkladem zajímavého řešení některých problémů (viz. toroidní pole a přechodové vrtsvy v Geometry Clipmaps algoritmu). Uvědomil jsem si taktéž velkou výhodu, pokud je předem definováno cílové nasazení zobrazovacího algoritmu. Důkazem mohou být např. terény menších rozměrů - nemá již příliš smysl zabývat se složitou LOD technikou, dostačující se stává jakákoliv hierarchická struktura podporující ořezávání pohledovým jehlanem.

5.4 Směr další práce

Další vývoj by se měl především zaměřit na implementaci komprese reziduí. Tím bychom se zbavili závislosti na množství operační paměti počítače (přesněji bychom ji spíše razantně omezili). Následně by mohla být přidána podpora rozsáhlých textur, např. dle práce [3]. V neposlední řadě by bylo vhodné provést netriviální optimalizace kódu, na základě profilování a četných testů.

Literatura

[1] Lossaso, F. and Hoppe, H. 2004. *Geometry clipmaps: terrain rendering using nested regular grids*, ACM SIGGRAPH 2004, 769–776.

[2] Pharr, M. and Fernando, R. *GPU Gems 2*, eds., Addison-Wesley, March 2005

[3] Tanner, C., Migdal, C. and Jones, M. 1998. *The clipmap: A virtual mipmap*, ACM SIGGRAPH 1998, 151–158.

[4] Fowler, R. J. and Little, J. J. 1979 *Automatic extraction of Irregular Network Digital Terrain Models*, SIGGRAPH 79, 199–207.

[5] Vaněk, J. 2003 *Zobrazování terénu*, Univerzita Hradec Králové

[6] Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L., Faust, N., and Turner, G. 1996. *Real-time, continuous level of detail rendering of height fields*, ACM SIGGRAPH 1996, 109–118.

[7] Duchaineau, M., Wolinsky, M., Sigeti, D., Miller, M., Aldrich, C., and Mineev-Weinstein, M. 1997. *ROAMing terrain: Real-time op-timally adapting meshes*, IEEE Visualization 1997, 81–88.

[8] Luebke, D., Reddy, M., Cohen, J. D., Varshney, A., Watson, B., and Huebner, R. *Level of detail for 3D graphics*, Morgan Kaufman Publishers, 2001, 185–228.

[9] Boer, W. H. 2000. *Fast Terrain Rendering Using Geometrical Mip-Mapping*

[10] Lindstrom, P., and Pascucci, V. 2001. *Visualization of Large Terrains Made Easy*, IEEE Visualization 2001

[11] Snook, G. *Real-Time 3D Terrain Engines Using C++ and DirectX 9*, Charles River Media, 2003, 100–117

Příloha A

Obsah CD

Součástí práce je disk CD-ROM obsahující výslednou implementaci projektu. Strom adresářů je organizován takto:

- `/app` - spustitelná aplikace okTerrain s jednou předzpracovanou sadou dat
- `/data` - několik výškových map určených k předzpracování
- `/install` - distibuční *.zip* archiv projektu. Obsahuje:
  - aplikace okTerrain a okPreproc
  - všechny *.dll* knihovny nezbytné pro běh obou aplikací
  - sadu ukázkových předzpracovaných terénních dat
- `/docs/api` - programátorská dokumentace vytvořená systémem *Doxygen*
- `/docs/thesis` - text práce ve formátu *.pdf* a zdrojový text v jazyce *csLaTeX*
- `/sreens` - několik sejmutých obrazovek z běhu aplikace okTerrain
- `/src` - zdrojové kódy celého projektu
- `/tests` - textové soubory s daty naměřenými v rámci Podkapitoly 4.7

Příloha B

Uživatelská příručka

B.1 Požadavky a instalace

Pro běh aplikace musí cílový systém splňovat následující požadavky:

- Grafická karta podporující OpenGL 1.4. nebo vyšší verze, tento ovladač nainstalovaný a funkční, podporován *multitexturing* a rozšíření ARB_vertex_buffer_object
- Monitor umožňující rozlišení alespoň 800×600
- 512MB RAM a více
- Dostatečný volný prostor na pevném disku, minimálně 70 MB pro základní instalaci
- Operační systém Microsoft 2000/XP

Instalaci provedeme rozbalením archivu `install/okterrain.zip` na libovolné místo na pevném disku.

B.2 Aplikace okTerrain

Aplikace se spouští souborem `bin/okTerrain.exe`. Nejprve je inicializován grafický režim, načítána konfigurace a poté vstupní terén. Pokud v libovolné fázi dojde k chybě, program zobrazí okno s popisem chyby a je ukončen. Při úspěchu se program přepne do celoobrazovkového režimu a vykreslí počáteční pohled na terén. V prostoru je možné se libovolně pohybovat. Základní ovládání zajišťují:

- Klávesy $\uparrow, \downarrow, \leftarrow, \rightarrow$: pohyb dopředu, dozadu, vlevo, vpravo
- Pohyb myší : otáčení pohledu

- Klávesy 1–9 : změna rychlosti pohybu (nejpomalejší - 1)
- Klávesa H : zobrazení/skrytí nápovědy
- Klávesa Escape : ukončení programu

Vzhled terénu je možné dále ovlivnit klávesami uvedenými v následující tabulce:

Klávesa	Zapíná/vypíná
D	detailní mapa[1]
J	eliminace T-spojů
M	podpora přechodových vrstev
T	potažení terénu texturou[2]
W	zvýraznění hran trojúhelníků

Pro zobrazení jiného terénu je potřeba změnit konfiguraci aplikace - k tomu účelu slouží soubor `config/config.xml`. Soubor může být ručně editován např. v poznámkovém bloku. Důležitým nastavením je vlastnost *input_file*. Stanovuje cestu k souboru, který popisuje vstupní terén pro aplikaci okTerrain[3]. Dále jdou změnit vlastnosti *screen_width* a *screen_height*, které určují rozlišení v jakém má být aplikace příště spuštěna. Před takovou změnou se doporučuje zkontrolovat jestli nové rozlišení podporuje monitor a grafická karta cílového stroje.

B.3 Aplikace okPreproc

Každý terén musí být před použitím v aplikaci okTerrain předzpracován aplikací okPreproc (`bin/okPreproc.exe`). Předzpracování je jednorázové a *není* jej třeba mezi opakovaným pouštěním aplikace okTerrain provádět znovu.

Funkce aplikace je ovládána pomocí parametrů předaných při spuštění. Jako jediné dva *povinné* parametry jsou cesty k vstupními a výstupnímu souboru[4]. Nejdůležitější *nepovinné* parametry zachycuje následující tabulka:

[3] jeho struktura byla zběžně popsána v Podkapitole 4.4, soubor je generován aplikací okPreproc viz. dále

[4] je vhodné, aby výstupní soubor měl příponu *.xml*, není to však nutné

Parametr (argument)	Popis	Přednastaveno
-x (celé číslo)	šířka výškové mapy	čteno ze vstupního souboru
-y (celé číslo)	výška výškové mapy	čteno ze vstupního souboru
-n (celé číslo)	rozměr mřížky	257
-t (cesta)	textura terénu	textura se nepoužije
-c	udává zda se má výstupní soubor nastavit jako vstupní pro aplikaci okTerrain	ne
-e (reálné číslo)	koeficient, kterým jsou upraveny výškové hodnoty	šířka vstupního souboru/256

Pokud je vstupní soubor ve formátu, který sám o sobě nenese informaci o svých rozměrech, stávají se povinnými i parametry `-x` a `-y`. Do této kategorie spadají i formáty *.hgt*[5] a *.raw*. Po spuštění `okPreproc.exe -h` vypíše program všechny podporované parametry a jejich popis.

Příklad zobrazení vlastního terénu

1. V adresáři `/bin` zadáme

   ```
   okPreproc ../data/sample01.png ../pdata/sample01.xml
     -t ../data/sample01t.png -n 183
   ```

2. V poznámkovém bloku otevřeme soubor `config/config.xml` a nahradíme řádek obsahující `name="input_file"` za následující:

   ```
   <property name="input_file" value="../pdata/sample01.xml" />
   ```

3. Z adresáře `/bin` spustíme aplikaci okTerrain.exe

Pokud je vstupní soubor ve formátu *.raw*, je třeba zadat jeho přesné rozměry. Případně se, přidáním přepínače `-c`, můžeme vyhnout druhému kroku v předešlém příkladu. Zobrazení by se tedy uskutečnilo:

1. V adresáři `/bin` zadáme

   ```
   okPreproc ../data/sample02.raw ../pdata/sample02.xml
     -x 1025 -y 1025 -t ../data/sample02t.png -c
   ```

2. Z adresáře `/bin` spustíme aplikaci okTerrain.exe

[5]tyto soubory mívají zpravidla rozměry 1 201 × 1 201 nebo 3601 × 3601