VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Diplomová práce

Petr Vrba

2006

Prohlášení

Prohlašuji, že jsem tuto Diplomovou práci vypracoval samostatně pod vedením Ing. Jana Pečivy. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Petr Vrba
15.5.2006

Poděkování

Rád bych tímto poděkoval mému vedoucímu ing. Janu Pečivovi za hodnotné odborné rady a přátelský přístup při řešení Diplomové práce.

Abstrakt

Projekt je z oblasti počítačové grafiky, konkrétně zobrazování krajiny. Základním požadavkem bylo zobrazit krajinu pomocí Chunked-LOD algoritmu a vyřešit problémy plynulého navazování částí krajiny. Projekt obsahuje dvě aplikace pro zobrazování krajin v podobě výškových map a v podobě obecných modelů o vysokém rozlišení v reálném čase. V dokumentu jsou popsány požadované principy a algoritmy. Implementace je realizována pomocí knihovny Open Inventor.

Klíčová slova

virtuální krajina, Perlinova funkce, chunked, LOD, level of detail, model, terén, reálný čas, Open Inventor

Abstract

This project is focused on visualization of landscape. Terrain is rendered by chunked-LOD algorithm. Main effort was focused on perfect merge between terrain's segments. This report describe required principles. Project contains two applications. The first works with landscape in heightmap form.The second application works with terrain as universal model. Both programs show high resolution terrain in real time. For implementation is used Open Inventor library.

Key words

virtual landscape, Perlin noise, chunked, LOD, level of detail, terrain, real time, model, Open Inventor

Obsah

Kapitola 1

Úvod

V dnešní době řeší počítačová grafika několik závažných problémů. Jedním z nich je zobrazování krajiny pomocí soudobého hardwaru. Ať již zobrazujeme krajinu v počítačové hře nebo v geodetickém simulátoru, řešíme stále stejný problém. Při vykreslování terénu je hlavním důvodem malého výkonu obrovské množství dat, které musí grafická karta a procesor zpracovat. Proto se využívá nejrůznějších postupů, jak množství dat snížit. Jedná se například o ROAM algoritmus, chunked-LOD a další.

Požadované vlastnosti zobrazované krajiny jsou různé. V simulačních aplikacích vyžadujeme co nejvěrnější ztvárnění krajiny a na rychlosti simulace nám prioritně nezáleží. Naproti tomu v jiných případech (např. počítačové hry) usilujeme především o dokonalou real-timovou aplikaci a některé aspekty detailu jsme ochotni oželet.

V rámci Ročníkového projektu jsem se zabýval generováním krajiny pomocí Perlinovy funkce a použitím algoritmu Chunked-LOD pro její následovné zobrazení. Algoritmus se mi podařilo implementovat, ovšem krajina vykazovala nepřesnosti u sousedících segmentů, které neměly stejný rozměr. Problém jsem řešil svislými stěnami na okraji každého segmentu. Toto řešení ovšem nebylo ideální. V Semestrálním projektu jsem se proto důkladněji zabýval vyřešením problému navazování jednotlivých segmentů terénu. Problém jsem vyřešil použitím švů. Mezi jednotlivé segmenty jsem umístil speciální úzké části terénu, které realizovali přechody mezi segmenty různých velikostí.

V následujících kapitolách postupně objasním zobrazování terénu, který je reprezentován jak v podobě výškové mapy, tak v podobě obecného modelu.

Ke každému typu terénu rozvedu téma o implementaci zadaného problému pomocí knihovny Open Inventor. Nastíním také různé překážky, které při implementaci nastaly a následně i jejich řešení.

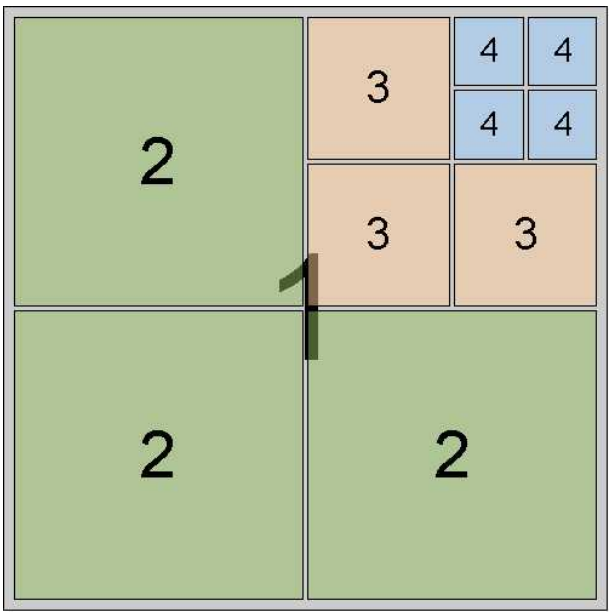
V závěru zhodnotím výkonové a vizuální vlastnosti algoritmu a také uvedu možnosti budoucího rozvoje tohoto projektu.

Kapitola 2

Chunked-LOD algoritmus

Algoritmus Chunked-LOD představil v roce 2002 Thatcher Ulrich (více viz. [6]). Jak již z názvu metody vyplývá (chunk = kus, LOD = Level of detail = úroveň detailu), princip optimalizace spočívá v myšlence, že vzdálenější části krajiny nemusí být vykreslovány v maximálním detailu. To znamená, že s rostoucí vzdáleností od kamery redukujeme detail krajiny.

Narozdíl od jiných algoritmů, kde máme krajinu jako jednolitý celek, je terén rozdělen na kusy nebo-li segmenty. Celá krajina je definována několikrát v různém detailu. Podle pohybu ve scéně a vzdálenosti od jednotlivých částí krajiny se pak zobrazují segmenty v příslušném detailu. Obecně platí, že čím dále je kamera, tím větší část krajiny se muže zobrazit s menším detailem.



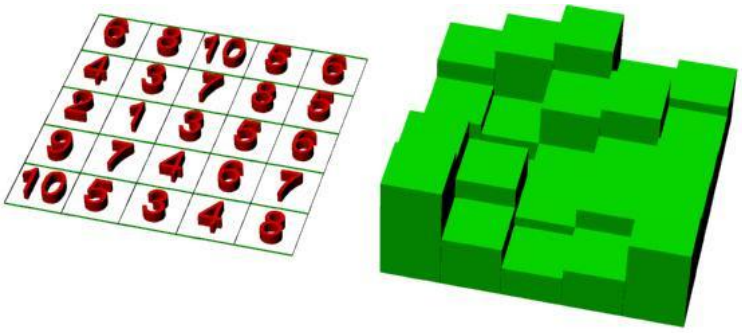
Obrázek 2.1: Chunked-LOD: Dělení segmentů

Na obrázku 2.1 je nastíněn způsob dělení segmentů. Každý segment je označem číslem, které určuje jeho úroveň detailu. Pokud si na obrázku odmyslíme segment s číslem 1, který definuje celou krajinu v nejnižším detailu, získame situaci, kdy je kamera v pravém horním rohu. To znamená, že segmenty v blízkosti kamery se musejí zobrazit ve vyšším detailu.

Kapitola 3

Výšková mapa

Pokud chceme zobrazit zemský povrch, musíme si určitým způsobem uchovat informace o jeho tvaru. K tomuto účelu se hojně využívají výškové mapy. Výškovou mapou rozumíme matici hodnot, které nám definují výšku terénu v místě daném souřadnicemi (viz. obrázek 3.1). Z takto připravených dat lze již snadno vytvořit trojrozměrný model krajiny.



Obrázek 3.1: Příklad výškové mapy a 3D modelu

V případě, že zobrazujeme existující povrch, výškovou mapu máme již k dispozici (například z družicových snímků). Pokud je však požadavkem zobrazení neexistující virtuální krajiny, musíme si data do výškové mapy vhodným způsobem vygenerovat. K tomuto účelu se používají nejrůznější metody. Požadavkem, na každý postup při generování výškové mapy, je naplnit toto dvourozměrné pole vhodnými daty tak, aby po převedení na model, co nejvíce připomínálo reálný povrch. V aplikaci jsem implementoval generování krajiny Perlinovou funkcí (více viz. [3]).

3.1 Princip

Při generování modelu krajiny o rozměrech například 512x512 postupujeme následovně. Vygenerujeme nebo načteme si výškovou mapu celého terénu. Zvolíme si, že rozměr jednoho segmentu v paměti bude například 32x32. Vytvoříme si první segment, který bude popisovat celou krajinu tedy o souřadnicích 0,0 – 512,512. Do tohoto segmentu přeneseme z výškové mapy každou 16tou ($512/32 = 16$) hodnotu. Nyní nám tento segment popisuje krajinu v minimálním detailu. Vytvoříme 4 podsegmenty. První z nich bude popisovat krajinu o souřadnicích 0,0 – 256,256. Druhý o souřadnicích 256,0 – 512,256 atd. Každý z těchto

4 podsegmentů opět naplníme daty z výškové mapy s tím rozdílem, že budeme přenášet každou osmou ($256/32 = 8$) hodnotu. Ke každému segmentu vytváříme další 4 podsegmenty, které definují stejný prostor krajiny s dvojnásobným detailem. Až se při generování dostaneme do stavu, že segment na mapě popisuje území o rozměrech stejných jako je jeho velikost (tedy 32x32), nebudeme pro něj již vytvářet podsegmenty, neboť úroveň detailu je maximální.

Po tomto postupu nám vznikne quad-tree nebo-li strom, kde každý uzel má 4 poduzly. Pro každý segment vygenerujeme příslušný model krajiny. Při zobrazování krajiny procházíme quad-tree od kořene a testujeme, zda aktuální prvek stromu vyhovuje našim požadavkům detailu. Vypočteme si vzdálenost segmentu d od kamery a míru chyby zobrazení $chunk\_error$ aktuálního segmentu podle následujících vztahů.

$$d_{2D} = \sqrt{(cam_x - segment_x)^2 + (cam_y - segment_y)^2}$$

$$d = \sqrt{{d_{2D}}^2 + (cam_z - segment_z)^2}$$

$$chunk\_error = (x_2 - x_1)/(segment_{width} - 1)$$

a dosadíme do vztahu

$$show = (chunk\_error * LODdistance)/d$$

kde $LODdistance$ určuje vzdálenost od kamery, od které se může snižovat úroveň detailu a x_1 a x_2 jsou souřadnice okraje segmentu v krajině. Pokud je výsledek $show <= 1$, zobrazí se daný segment, protože vzhledem ke vzdálenosti od kamery je úroveň detailu dostačující. Pokud je ovšem $show > 1$ pro aktuální segment, musíme opakovat stejný postup pro jeho 4 podsegmenty.

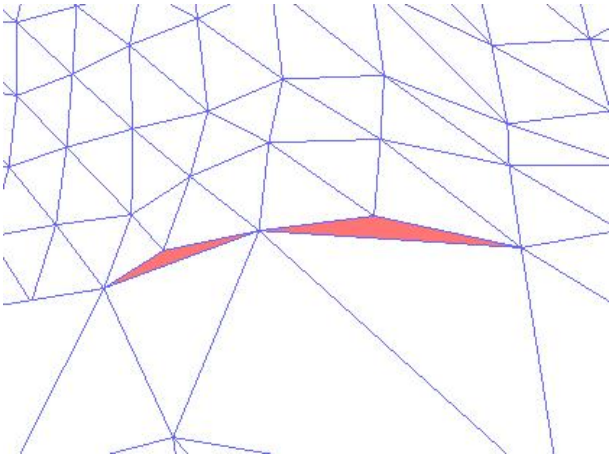
V aplikaci pak v určitých časových intervalech (např. 5x za sekundu) kontrolujeme krajinu, zda stále vyhovuje našim požadavkům detailu. Pokud tomu tak není, přeuspořádáme model krajiny podle aktuální potřeby. Pro testování krajiny před každým snímkem většinou nemáme dostatečný výkon hardware. Takto časté testování by bylo zbytečné, neboť většinou se kamera ve scéně pohybuje dostatečně pomalu na to, aby frekvence testování krajiny stíhala adekvátně reagovat.

3.2 Navazování segmentů

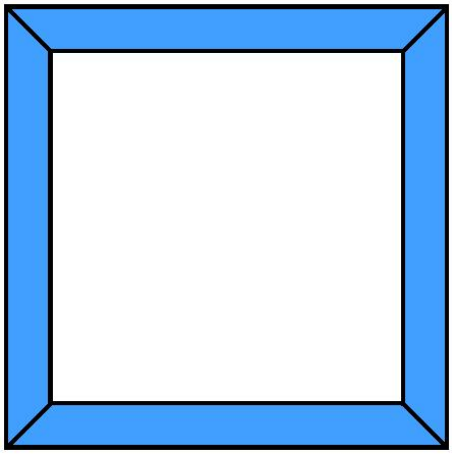
Jelikož je terén sestavován z částí a ke každé je přistupováno jako k samostatnému modelu, nutně narážíme na problém vizuálního spojení segmentů tak, aby krajina vypadala jako celistvý útvar. Velké potíže nám budou činit nestejně velké segmenty, které budou sousedit ve výsledné vykreslované krajině. V následujících odstavcích se seznámíme s některými nedostatky a ukážeme si, jak je řešit.

Prvním problémem, se kterým se musíme vypořádat, jsou mezery v krajině v místech, kde se dotýkají segmenty různých úrovní detailu. Tuto situaci zmiňuje již [4]. K tomuto problému dochází v důsledku rozdílné členitosti terénu těchto segmentů. Tento jev je dobře patrný na obrázku 3.2. Červené trojúhelníky jsou chybějící plochy v krajině. Výsledkem je, že pozorovatel vidí pod krajinu.

Abychom vyřešili tento problém, je nutné na každém okraji každého segmentu vytvořit takzvaný šev (viz. obrázek 3.3).



Obrázek 3.2: Nespojené segmenty



Obrázek 3.3: Švy segmentu

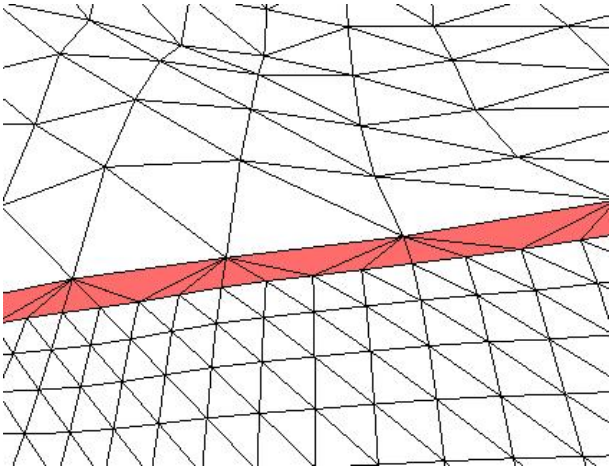
Při zobrazování segmentu zjistíme jaký segment se nachází na straně švu, který právě generujeme. Spočítáme rozdíl úrovně detailu (LOD) zpracovávaného segmentu a sousedícího segmentu.

$$rozdil_{LOD} = LOD_{generovany} - LOD_{sousedni}$$

Pokud bude tato hodnota 0, sousedící segment je stejné úrovně detailu. Proto zobrazíme předpřipravený šev pro rozdíl úrovně 0. Pokud bude rozdíl LOD záporný, znamená to, že sousedící segment má vyšší úroveň detailu. V tomto případě zobrazíme také šev pro rozdíl LOD 0, protože nesrovnalosti mezi segmenty opravuje vždy segment s vyšší úrovní detailu.

Kladný rozdíl LOD znamená, že přilehlý segment má nižší úroveň detailu, a proto se právě zobrazovaný segment musí svým švem přizpůsobit. Vygenerujeme si šev, který bude korigovat rozdíl úrovní detailu. Tento šev si uložíme a budeme jej k tomuto segmentu zobrazovat do doby, než se změní požadavky na švy.

Přizpůsobený šev musí spojit n trojúhelníků na jedné straně a $n*rozdil^2_{LOD}$ trojúhelníků na straně druhé. Z tohoto požadavku vyplývá uspořádání trojúhelníků švu, které je patrné na obrázku 3.4.



Obrázek 3.4: Přizpůsobený šev (rozdíl LOD = 2)

Jak zjistit úroveň detailu sousedního segmentu? Výpočet vzdálenosti kamery od každého sousedního segmentu je výpočetně náročný. Proto je v aplikaci vytvořeno dvourozměrné pole. Jeho rozměr je $rozmer_{mapy}/rozmer_{segmentu}$. Do tohoto pole zapisují zobrazované segmenty svoji úroveň detailu vždy tak, aby postihly poměrnou část krajiny, kterou reprezentují. Například segment s nejnižší úrovní detailu vyplní celé pole hodnotou 1, protože zobrazuje celou krajinu s úrovní detailu 1. Naopak segment s nejvyšší úrovní detailu, zapíše do pole jedinou hodnotu na místo, které odpovídá jeho pozici v krajině. Příklad vyplněného pole LOD ukazuje obrázek 3.5.

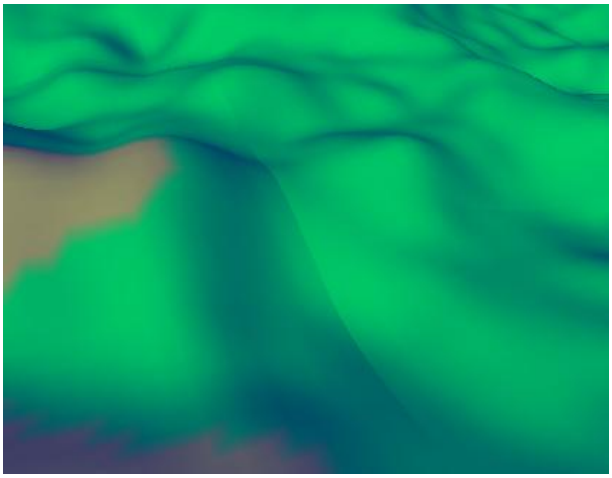
2	2	2	2	3	3	3	3
2	2	2	2	3	3	3	3
2	2	2	2	3	3	3	3
2	2	2	2	3	3	3	3
3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3
3	3	3	3	3	3	4	4
3	3	3	3	3	3	4	5

Obrázek 3.5: Příklad vyplněného pole

Další problém, na který narazíme, jsou viditelné nesrovnalosti na hranicích mezi segmenty (viz. obrázek 3.6). Tento jev vzniká v důsledku chybně vypočtených normálových vektorů na okrajích těchto segmentů.

Pro výpočet normály místa v krajině jsou nutné informace o výšce sousedních bodů. Pokud tedy počítáme normálový vektor pro okrajový bod, tyto informace nám nutně chybí. Řešení je velmi prosté. Nejprve si celou krajinu vygenerujeme do jediného velkého pole.

Při vytváření modelů pro jednotlivé dílčí segmenty krajiny, jsme pak schopni počítat i s hodnotami mimo rozsah aktuálního segmentu, neboť máme potřebná data k dispozici v podobě pomocného pole. Výsledkem tohoto přístupu je dokonalé navázání stejně velkých segmentů.



Obrázek 3.6: Chybně vypočtené normály

Pokud ovšem při vykreslování krajiny sousedí segmenty různých velikostí, problém nekorektního navázání terénu přetrvává. Chybné spojení segmentů je způsobeno faktem, že pro výpočet normálového vektoru jsou potřebné nejen sousední výšky terénu, ale také jejich vzájemná vzdálenost a ta je v případě nestejně velkých segmentů rozdílná. Tento problém nám pomohou vyřešit výše zmiňované švy. Normálové vektory na hranici švu, která se dotýká segmentu s nižší úrovní detailu, musíme počítat tak, jako by náležely do sousedního segmentu. Tímto postupem vzniknou hraniční trojúhelníky, které na jedné straně mají normálový vektor zobrazovaného segmentu a na druhé straně normálový vektor sousedního segmentu. Plynule se tak propojí normály obou segmentů a krajina zůstává dokonale hladká.

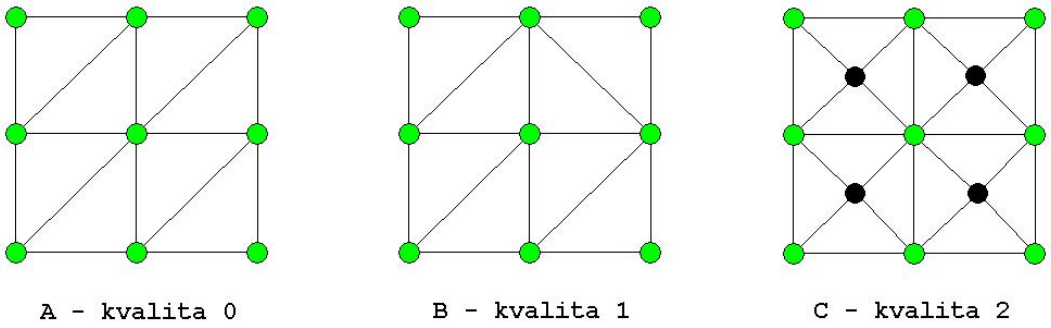
3.3 Implementace

Aplikace je implementována v prostředí Open Inventor od firmy SGI. Toto prostředí je nadstavbou nad knihovnou OpenGL. Přináší vyšší stupeň abstrakce při programování grafických aplikací. Konkrétní balík, který je použit v tomto projektu je Coin od firmy Systems in Motion pod GPL licencí. Více k historii a používání Open Inventoru viz. [2].

Tesselace

Při převádění výškové mapy na 3D model, lze při spuštění aplikace (parametrem -hq) určit jednu ze 3 možných voleb tesselace. Rozdíly mezi jednotlivými způsoby osvětluje obrázek 3.7.

Zelené body na obrázku jsou hodnoty z výškové mapy a černé body jsou dopočítány pomocí interpolace. Možnost A je v aplikaci výchozí. Tento způsob tesselace určuje, že každý čtverec v krajině bude reprezentován dvěma trojúhelníky a rozdělení čtverce bude vždy ve stejném směru. Naproti tomu metoda B určuje, že se čtverec zobrazí také dvěma



Obrázek 3.7: Různé metody tesselace krajiny

trojúhelníky, ovšem směr rozdělení čtverce se určí podle výšek jeho rohů v krajině tak, aby oba trojúhelníky lépe postihly sklon krajiny. A konečně metoda C modeluje čtverec v krajině jako 4 trojúhelníky. Černý bod je doplněn pomocí lineární interpolace čtyř krajních bodů. Tento způsob tesselace přináší lepší výsledky při osvětlování terénu. Klade ovšem dvojnásobné nároky na hardware počítače (dvojnásobný počet trojúhelníků).

Povrch krajiny

Jak již bylo z dřívějších obrázků vidět, v krajině se vyskytují 4 druhy povrchu. Konkrétně vodní hladina, tráva, hory a sněhové čepičky hor. V aplikaci je tohoto efektu docíleno pomocí materiálů, kdy každému vertexu přidělíme materiál podle jeho výšky v krajině. Voda je realizována jako vodorovný čtverec, který protíná krajinu a tím vytváří dojem hladiny.

3.4 Výkonové vlastnosti

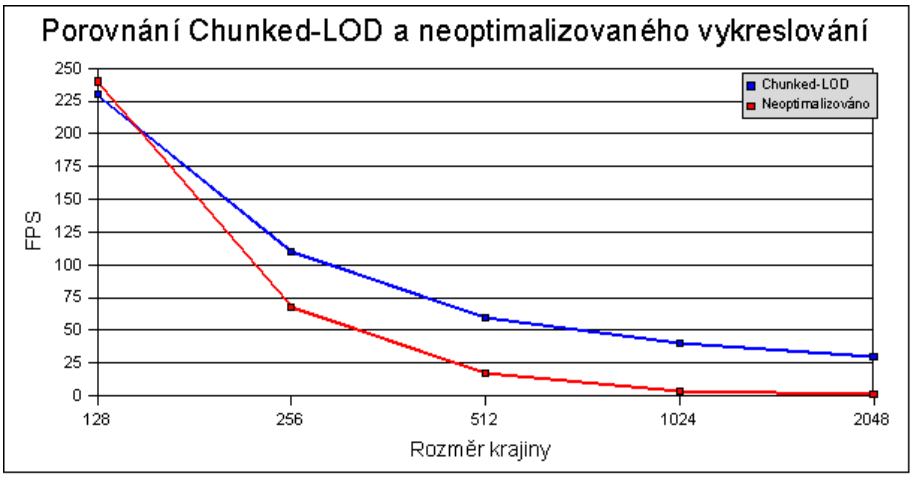
Výkon

Domnívám se, že po výkonové stránce aplikace splnila hlavní požadavek. A tím je zobrazení rozsáhlé krajiny s přijatelnou úrovní detailu v reálném čase. V následujícím grafu je dobře patrný nárůst výkonu, především při zobrazování rozsáhlých krajin.

Hodnoty byly získány na počítači: Athlon64 3000+, 1024 MB RAM, ATI Radeon 9600XT.

Rozměr krajiny	128	256	512	1024	2048
Chunked-LOD	230	110	60	40	30
Neoptimalizováno	240	68	17	4	1

Tabulka 3.1: Porovnání Chunked-LOD a neoptimalizovaného vykreslování



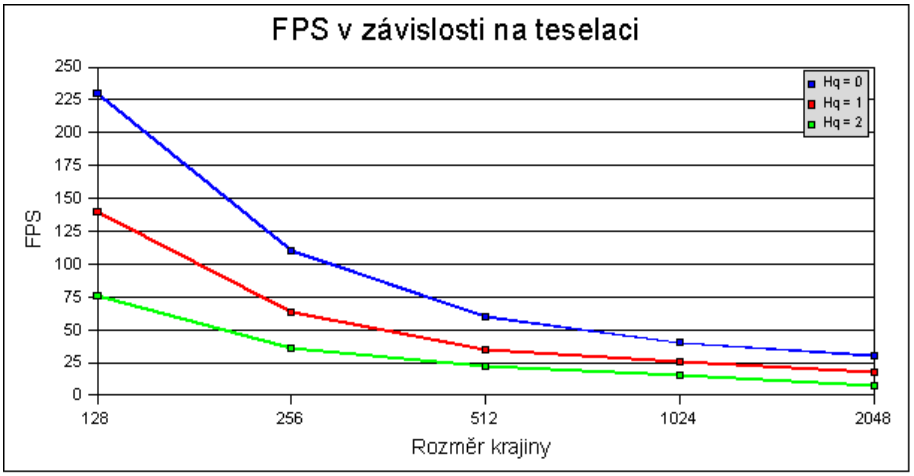
Obrázek 3.8: Graf porovnání výkonu

Jak již z grafu 3.8 vyplývá, síla chunked-LOD algoritmu se naplno projeví při renderování krajiny s vysokým rozlišením. V případě terénu o rozměrech 1024x1024 byl výkon chunked-LOD algoritmu 10krát vyšší a při rozlišení 2048x2048 dokonce 30krát vyšší. Naopak při rozlišení 128x128 můžeme pozorovat drobný pokles výkonu, který přisuzuji režii chunked-LOD algoritmu.

Rozměr krajiny	128	256	512	1024	2048
Hq = 0	230	110	60	40	30
Hq = 1	140	63	35	26	18
Hq = 2	76	36	22	15	7

Tabulka 3.2: FPS v závislosti na teselaci ($size = 512, distance = 100, dimension = 33$)

Podle výsledků z grafu 3.9 můžeme usoudit, že způsob teselace výrazně ovlivňuje výkon aplikace. Při zobrazování krajiny se tedy musíme rozhodnout, zda požadujeme vyšší výkon



Obrázek 3.9: FPS v závislosti na teselaci ($size = 512, distance = 100, dimension = 33$)

nebo kvalitnější model krajiny.

Rozměr segmentu	9	17	33	65	129	257
FPS	6	20	60	73	58	22

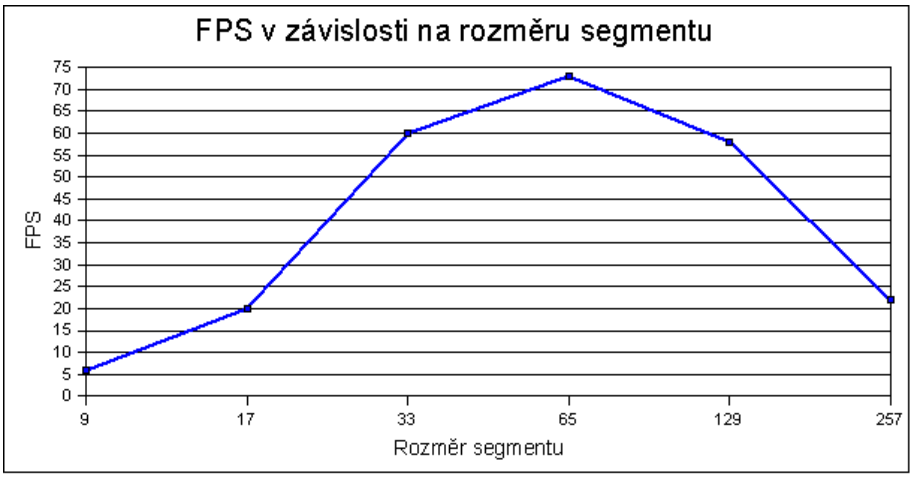
Tabulka 3.3: FPS v závislosti na rozměrech segmentu ($size = 512, distance = 100, hq = 0$)

Rozměr segmentu se ukázal jako zajímavý nástroj při ladění výkonu aplikace. Při velmi malých rozměrech segmentu došlo k drastické ztrátě výkonu. Tento fakt přisuzuji velkému počtu segmentů a tudíš náročné režii Chunked-LOD algoritmu při kontrole detailu ve scéně. Naopak při velkých rozměrech segmentu byla sice režie algoritmu malá, ale nezůstal žádný prostor pro uplatnění optimalizací. Proto ani při tomto nastavení nevykazovala aplikace uspokojivé výsledky. Správným řešením se tedy ukázalo volit středně velké segmenty. Režie algoritmu není velká a přesto je quadtree modelu dostatečně vysoký na to, aby se při renderování mohla krajina zobrazit v nižším detailu.

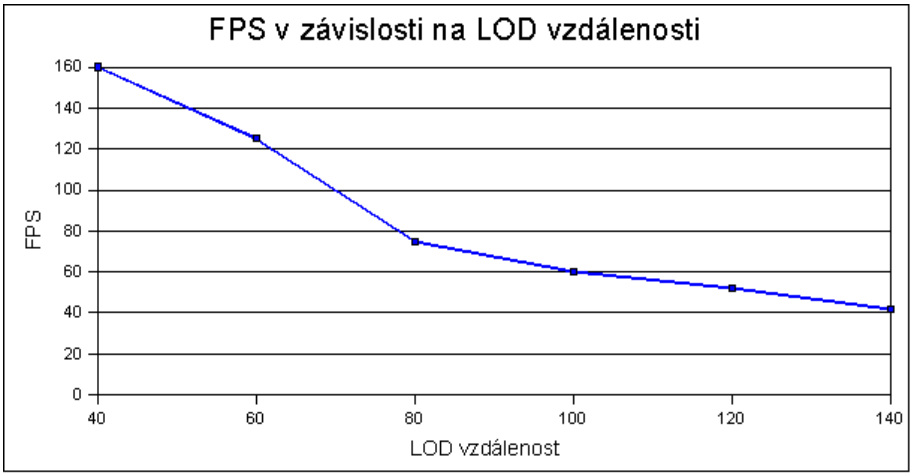
LOD vzdálenost	40	60	80	100	120	140
FPS	160	125	75	60	52	42

Tabulka 3.4: FPS v závislosti na LOD vzdálenosti ($size = 512, dimension = 33, hq = 0$)

Výsledky v grafu 3.11 nás ničím nepřekvapily. Protože LOD vzdálenost určuje, kdy se již může model přepnout do nižšího detailu, s rostoucí LOD vzdáleností klesá výkon. Opět zde platí staré známé dilema grafických aplikací a to, že je potřeba kompromisně volit mezi kvalitou a rychlostí.



Obrázek 3.10: FPS v závislosti na rozměrech segmentu ($size = 512, distance = 100, hq = 0$)



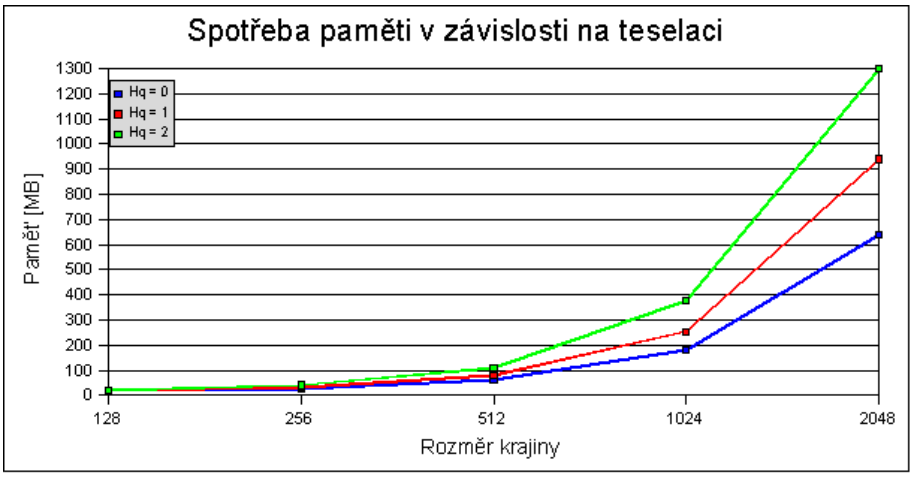
Obrázek 3.11: FPS v závislosti na LOD vzdálenosti ($size = 512, dimension = 33, hq = 0$)

Paměť

Jednou z nevýhod chunked-LOD algoritmu je poměrně vysoká paměťová náročnost. Jak vyplývá z obrázku 3.12, požadavek paměti roste s rozměrem krajiny přibližně exponenciálně. Řešením tohoto problému by u velkých krajin mohlo být průběžné odkládání nepotřebných částí krajiny na pevný disk.

Rozměr krajiny	128	256	512	1024	2048
Hq = 0	18	25	60	180	640
Hq = 1	20	32	80	255	940
Hq = 2	21	41	111	377	1300

Tabulka 3.5: Paměťová náročnost algoritmu chunked-LOD ($distance = 100, dimension = 33$)



Obrázek 3.12: Paměťová náročnost algoritmu chunked-LOD ($distance = 100, dimension = 33$)

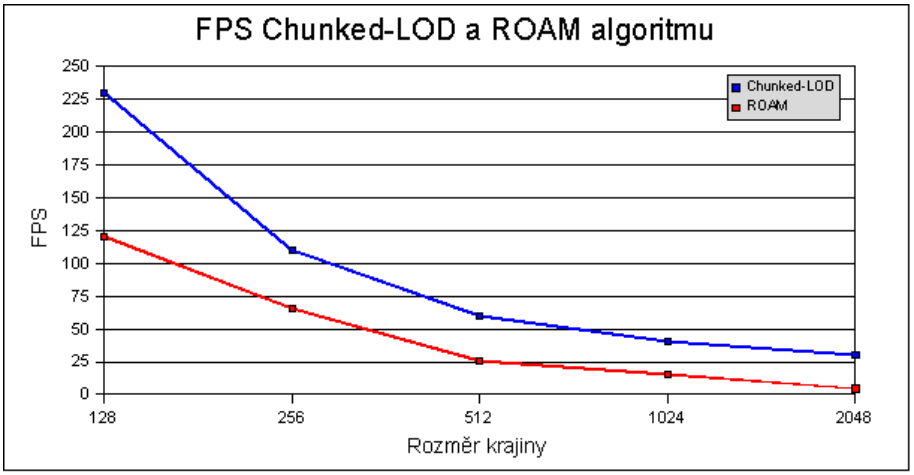
Porovnání s ROAM algoritmem

Zkratka ROAM znamená Real-time Optimally Adapting Meshes. Tento algoritmus si před zahájemín renderování vytvoří binární strom trojúhelníků. Při renderování krajiny je pak schopen slučovat nebo rozdělovat plochy dle vzdálenosti od kamery a tím dynamicky regulovat množství trojúhelníků ve scéně. Více viz. [5].

Porovnání výsledků jsem provedl vzhledem k práci Pavla Černého, který implementoval algoritmus (více vi. [1]).

Rozměr krajiny	128	256	512	1024	2048
Chunked-LOD	230	110	60	40	30
ROAM	120	66	26	15	4.5

Tabulka 3.6: Srovnání FPS dosažených Chunked-LOD a ROAM algoritmem

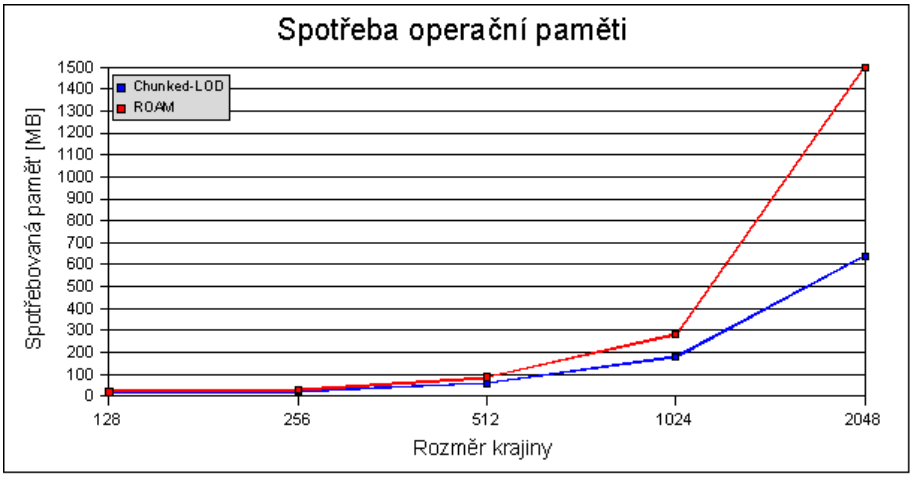


Obrázek 3.13: Srovnání FPS dosažených Chunked-LOD a ROAM algoritmem

Rozměr krajiny	128	256	512	1024	2048
Chunked-LOD	18	25	60	180	640
ROAM	20	32	87	280	1500

Tabulka 3.7: Paměťová náročnost Chunked-LOD a ROAM algoritmu v MB

Z grafů 3.13 a 3.14 vyplývá, že algoritmus ROAM je při zobrazování krajiny méně výkony a více náročný. Jeho výhodou je krátká doba preprocesingu, která je téměř zanedbatelná oproti době předpřipravování scény u chunked-LOD algoritmu.



Obrázek 3.14: Paměťová náročnost Chunked-LOD a ROAM algoritmu

Závěry testů

Výkon a vzhled aplikace můžeme ovlivňovat několika parametry. Vždy musíme mít na paměti, že výkon je vykoupen kvalitou zobrazovaného modelu. Všechny měřené parametry se ukázaly jako účinné regulátory výkonu. Jejich souběžné a vyvážené nastavení může vést k přijatelnému vizuálnímu výsledku s dostačujícím výkonem. Při testech se ale také projevila skutečnost, že jeden chybně zvolený parametr může pokazit výsledek celé aplikace.

Velkou zátěží pro počítač je jistě vysoká paměťová náročnost algoritmu. Když si představíme, že bychom k modelu ještě přidali textury s rozměry řádově v tisících, spotřeba paměti by se vyhoupla k již těžko akceptovatelným hodnotám.

3.5 Vizuální vlastnosti

Navazování segmentů

Na rozdíl od výsledku Ročníkového projektu, vzhled krajiny již vypadá přirozeně a neruší jej žádné nepřesnosti a nežádoucí předěly mezi segmenty. Použití švů se ukázalo jako obecné řešení navazování segmentů různé úrovně detailu.

Přepínání detailů

Přepínání detailu jednotlivých segmentů je dosti patrné. Zvláště při teselaci 0 je změna detailu do očí bijící, protože rozdíl v modelu zvýrazňuje nedokonalé stínování. Tento nedostatek by mohlo zmírnit zavedení textury, která by zakryla některé změny.

Kapitola 4

Obecný model

Oproti krajině uložené ve výškové mapě, bude zpracování a zobrazení obecného modelu, tedy modelu složeného z libovolných trojúhelníků, vyžadovat naprosto odlišný přístup. Hlavními problémy bude rozdělení modelu na části a snížení počtu trojúhelníků segmentů, které mají být zobrazeny v nižším detailu.

4.1 Dělení segmentů

Dělení segmentů budeme provádět po nepřímých hranicích. To znamená, že vzniklé rozdělené segmenty nebudou mít pravoúhlý tvar. Oproti klasickému pravoúhlému "sekání" segmentů, bude mít metoda několik výhod. Především nebudou vznikat redundantní trojúhelníky, které by zvyšovaly složitost modelu a tím zároveň nároky na grafický hardware. Další výhodou bude, že ve výsledné krajině budou hranice mezi segmenty hůře postřehnutelné.

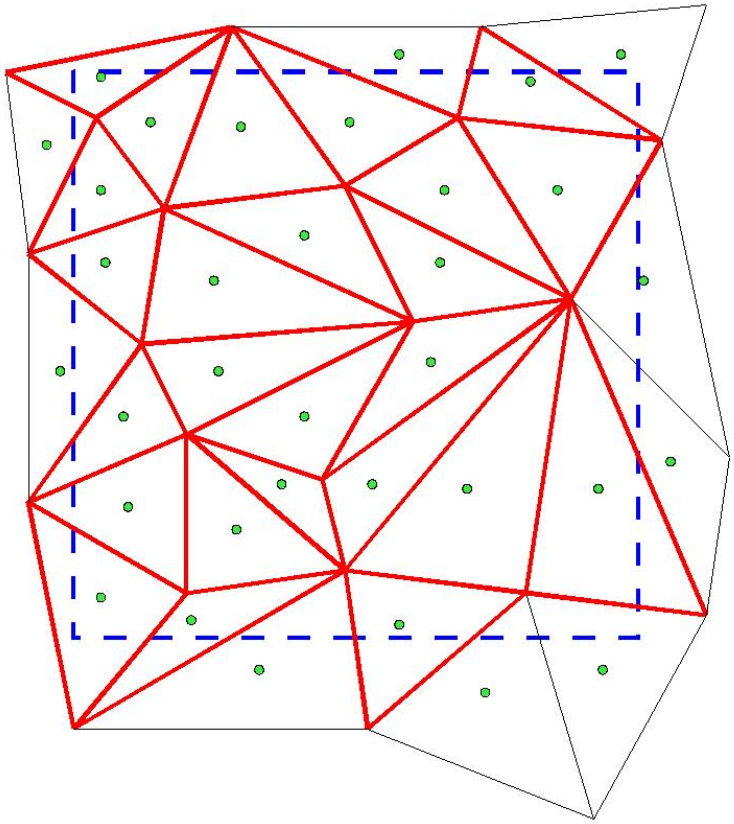
Základní princip dělení segmentů bude následovný. Z celkové krajiny budeme chtít například získat segment o souřadnicích [-100,-100] až [100,100]. U každého trojúhelníku si určíme jeho střed podle následujícího vztahu:

$$x_s = \frac{x_1 + x_2 + x_3}{3}$$

$$z_s = \frac{z_1 + z_2 + z_3}{3}$$

Proměnné s indexem s označují souřadnice středu trojúhelníku a proměnné s číselným indexem označují souřadnice jednotlivých vrcholů trojúhelníku. Pokud se střed trojúhelníku bude nacházet v oblasti vymezené souřadnicemi segmentu, pak bude tento trojúhelník náležet do daného segmentu. Tímto způsobem překontrolujeme všechny trojúhelníky v modelu a určíme, které patří do daného segmentu. Když tedy model rozdělíme, vzniknou segmenty, které sice nebudou provoúhlé, ale budou na sebe navazovat. Na obrázku 4.1 je vidět, jak bude vypadat segment. Hranice požadovaného segmentu jsou zobrazeny modře. Zelené body určují středy trojúhelníků. Červené trojúhelníky budou patřit do požadovaného segmentu a černé nikoliv.

Při použití klasického pravoúhlého řezání segmentů by vzniklo přibližně 20 nových trojúhelníků, které by v podstatě zdvojnásobily složitost vzniklého modelu.



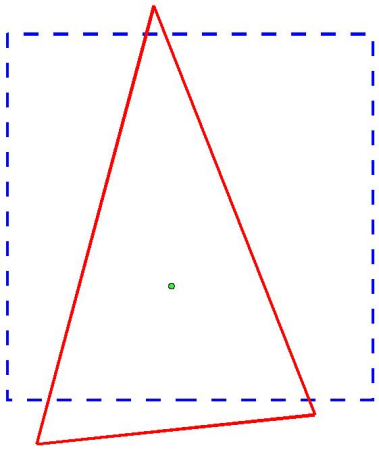
Obrázek 4.1: Dělení segmentu

4.2 Navazování segmentů

Dříve než přistoupíme k zjednodušování segmentů, musíme si uvědomit, že prostým a bezhlavým redukováním trojúhelníků vzniknou segmenty, které ve výsledné krajině nebudou navazovat a v terénu tak vzniknou díry. Při zjednodušování budeme tedy muset zajistit to, aby hraniční trojúhelníky nezměnily svou hranu, kterou sousedí s přilehlým segmentem. Budeme tedy muset určit hraniční body segmentu, se kterými se za žádnou cenu nesmí při redukci hýbat.

Tyto hraniční body lze nejlépe získat již při dělení segmentů, kdy určujeme, zda se trojúhelník nachází v daném segmentu. Při výpočtu těchto trojúhelníků si zároveň pro všechny tři jeho vrcholy zjistíme, zda leží v daném segmentu. Podle počtu vrcholů trojúhelníků v segmentu se pak zpracování rozpadne do těchto 3 možností:

1. **žádný z vrcholů trojúhelníku neleží v segmentu**
   V tomto případě by se na první pohled mohlo zdát, že trojúhelník neleží v segmentu a ani neprotíná hranice segmentu, a proto není potřeba se zabývat jeho vrcholy. Problém by ovšem nastal v případě, kdy by analyzovaný trojúhelník byl tak velký, že by svou plochou pohltil celý nebo téměř celý segment, jeho střed by patřil do segmentu a přesto by všechny jeho vrcholy ležely mimo segment. Vzniklou situaci nastiňuje obrázek 4.2. V takovémto případě pak musíme všechny vrcholy trojúhelníku zařadit do hraničních bodů segmentu.

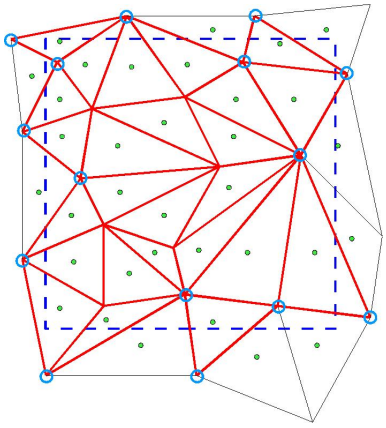


Obrázek 4.2: Zvláštní případ trojúhelníku v segmentu

2. **1 nebo 2 vrcholy trojúhelníku leží v segmentu**
   Pokud leží 1 nebo 2 vrcholy trojúhelníku v segmentu, pak leží 2 nebo 1 vrchol mimo segment. Z tohoto faktu dále vyplývá, že trojúhelník protíná hranice segmentu. Dále bude záležet na skutečnosti, zda patří trojúhelník do segmentu nebo ne. Pokud do něj náleží, musíme mezi hranice zařadit všechny vrcholy trojúhelníku, které leží mimo segment. Naopak pokud trojúhelník do segmentu nepatří, zařadíme do hraničních bodů všechny vrcholy, které leží v segmentu.

3. **všechny 3 vrcholy trojúhelníku leží v segmentu**
   V tomto případě leží všechny vrcholy trojúhelníku v segmentu, a proto není možné, aby trojúhelník protínal některou z hranic. Proto nebudeme do hraničních bodů přidávát žádný vrchol.

Situaci po určení hraničních bodů zachycuje obrázek 4.3.



Obrázek 4.3: Určené hraniční body segmentu

4.3 Redukce trojúhelníků

Nyní, když máme vyextrahovaný segment a známe jeho hraniční body, můžeme přistoupit ke snížení počtu trojúhelníků. Pro redukci trojúhelníků je mnoho algoritmů. Protože jsou však v tomto projektu na redukční algoritmus kladeny speciální požadavky (zachování hraničních bodů, rychlost redukce), vytvořil jsem vlastní postup.

Princip

Základní hledisko pro určení, která hrana má být odstraněna, bude vzdálenost jejích vrcholů. Z hlediska optimalizace modelu krajiny tento způsob možná není naprosto ideální, ale podle mého názoru poskytuje dobrý poměr mezi rychlostí a vizuálním výsledkem. Postup pro redukci bude následující:

1. Nalezení a seřazení hran podle vzdálenosti
   V každém trojúhelníku si určíme nejkratší hranu. Uložíme si její délku a vrcholy, které ji tvoří. Jeden z vrcholů si označíme ke smazání a druhý jako cílový vrchol, kam se přesunou všechny hrany, které původně vedly do smazaného bodu. Délku hrany a její dva body postupně ukládáme do pole. Když projdeme celý model, vzniklé pole seřadíme podle délky hran vzestupně. V programu je použit algoritmus Quick-sort.

2. Určení hran, které se mohou odstranit
   Ke každému vrcholu v modelu si vytvoříme 3 pomocné hodnoty. Budou uchovávat informaci o tom, zda byl vrchol smazán, kam se mají přesunout hrany vedoucí do vrcholu a zda do tohoto vrcholu byl přesunut jiný vrchol. Nejdříve tyto hodnoty nastavíme na výchozí hodnotu -1. Nyní postupně procházíme pole hran a pro každou hranu určujeme následující podmínku. O bodu k vymazání musí platit, že nesmí být již smazán a zároveň do něj nesmí být vztažen jiný bod a zároveň nesmí patřit do hraničních bodů. O bodu, do kterého se mazaný bod má přesunout, musí platit, že nesmí být již smazán. Pokud výše uvedené podmínky platí, může být právě zpracovávaná hrana vymazána. To znamená, že nastavíme, že bod k vymazání je smazán a hrany jsou vztaženy do druhého bodu dané hrany. Dále nastavíme, že do druhého bodu již byl vztažen nějaký vrchol. Toto děláme proto, abychom jej později nesmazali. Když tímto postupem projdeme všechny hrany z pole, označí se nám vrcholy, které se mají smazat a zároveň i vrcholy, do kterých se mají navázat hrany jdoucí do smazaného vrcholu.

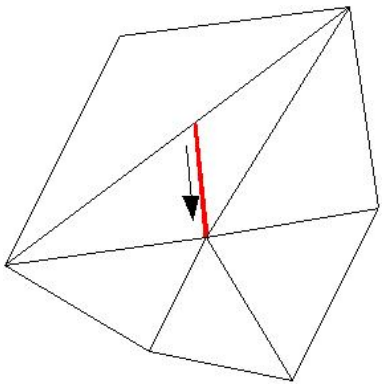
3. Mazání trojúhelníků
   Nyní postupně procházíme všechny trojúhelníky v modelu a zjišťujeme, zda obsahují vrchol pro vymazání a zároveň vrchol, do kterého se mazaný vrchol má vztáhnout. Pokud trojúhelník tuto podmínku splňuje, můžeme ho vymazat. Pokud danou podmínku nesplňuje, poupravíme pouze navázání hran.

4. Mazání vrcholů
   Nakonec vymažeme vrcholy, které jsou označeny k vymazání. Trojúhelníky, které tyto vrcholy využívaly jsou již smazány.

Mezi výhody této metody patří i fakt, že se trojúhelníky redukují rovnoměrně v celém modelu. Uživatel proto hůře postřehne změny terénu, které by se jinak mohly udát kumulovaně v jedné části modelu.

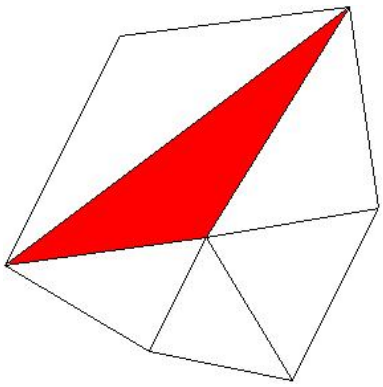
Chyby v modelu

Výše popsaná metoda je náchylná na jeden druh chyby. Pokud se bude v modelu vyskytovat podobné uspořádání trojúhelníků, jako je znázorněno na obrázku 4.4, dojde při redukci za určitých podmínek k porušení celistvosti modelu.



Obrázek 4.4: Situace před chybnou redukcí

Pokud se bude v obrázku 4.4 redukovat červená hrana ve směru šipky, vznikne v modelu „díra“. Výsledek po redukování je zobrazen na obrázku 4.5. Červeně vybarvenou oblast nedefinuje žádný trojúhelník, a proto se v modelu projeví jako chyba, která velice ruší vizuální dojem krajiny.



Obrázek 4.5: Situace po chybné redukci

Jak se podobná chyba projeví ve výsledném terénu, zobrazuje obrázek 4.6 (označeno červeně). Zatím jsem nenalezl jiný způsob, jak tyto chyby odstranit než tak, že upravíme model, aby se v něm nevyskytovaly podobné situace jako na obrázku 4.4.

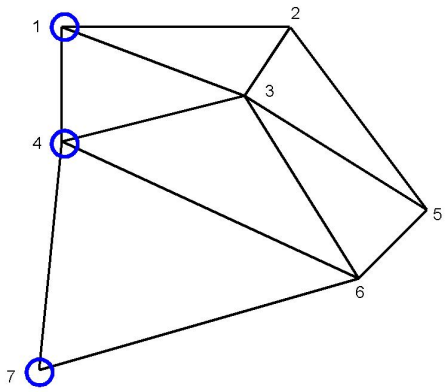
Další chyby, které se v modelu mohou po redukci vyskytnout, jsou poněkud jiného druhu. Za určitých podmínek může dojít k tomu, že model po redukci obsahuje trojúhelníky, které mají všechny 3 vrcholy v jedné přímce. Takovéto trojúhelníky pak není nutné zobrazovat. Prostředí Open Inventoru na tyto trojúhelníky upozorňuje varovným dialogovým oknem. K odstranění této chyby je v programu definována funkce, která překontroluje všechny trojúhelníky a určí, zda se nejedná o chybový případ. Kontrola se provádí tak, že se určí

libovolné 2 hrany trojúhelníku a vypočte se, zda mají stejný směrový vektor. V případě nutnosti je pak vadný trojúhelník z modelu odstraněn.

Obrázek 4.6: Chyba vzniklá redukcí ve skutečném modelu

Příklad redukce na jednoduchém modelu

Nyní si ukážeme konkrétní příklad redukce trojúhelníků na jednoduchém modelu. Výchozí model je zobrazen na obrázku 4.7. V modelu se vyskytujjí i hraniční body, se kterými nesmíme při redukci hýbat.



Obrázek 4.7: Příklad - Výchozí model před redukcí

Vrcholy jsou označeny číslem. Nejprve projdeme všechny trojúhelníky a nalezneme v nich nejkratší hranu a její dva vrcholy, které ji definují a tyto hodnoty si uložíme do pole.

Délka hrany	2.1	2.1	2.5	3.8	5.2	8.9
Vrchol ke smazání	3	2	6	1	3	4
Vrchol ke sloučení	2	3	5	4	4	7

Tabulka 4.1: Příklad - Seřazené pole hran v modelu

Číslo vrcholu	1	2	3	4	5	6	7
Smazán	-1	-1	-1	-1	-1	-1	-1
Sloučit do	-1	-1	-1	-1	-1	-1	-1
Sloučeno do mě	-1	-1	-1	-1	-1	-1	-1

Tabulka 4.2: Příklad - Inicializovaná tabulka vrcholů

Výsledné pole seřadíme podle délky hrany. Vyplněné a seřazené pole je zobrazeno v tabulce 4.1.

Nyní si ke každému vrcholu vytvoříme pomocné proměnné a inicializujeme je. Opět se bude jednat o pole. Jeho příklad zachycuje tabulka 4.2.

Nyní budeme postupně procházet tabulku hran a vyplňovat tabulku vrcholů. První sloupec udává, že máme vymazat vrchol 3 a hrany přenést do vrcholu 2. Tuto operaci je možné provést, protože vrchol 3 není hraniční bod a ani doposud nebyl označen jako smazaný. Vrchol 2 také není smazaný. V tabulce vrcholu označíme vrchol 3 jako smazaný. Dále označíme, že hrany se mají přesunout do vrcholu 2. A nakonec si u bodu 2 poznamenáme, že do něj byl vztažen nějaký jiný vrchol. Jak bude tabulka vypadat po prvním kroku ukazuje tabulka 4.3.

Tímto způsobem projdeme celou tabulku hran. Druhý záznam nebudeme moci zpracovat, protože vrchol 3 je označen k vymazání, a proto do něj nemůžeme vztahnout vrchol 2. Třetí záznam můžeme bez problému provést. Vrchol 6 označíme jako smazaný a hrany převedeme do vrcholu 5. Zbývající záznamy nebudeme moci provést, protože v nich figurují hraniční body a s těmi nesmíme hýbat. Kompletně vyplněnou tabulku vrcholu ukazuje tabulka 4.4.

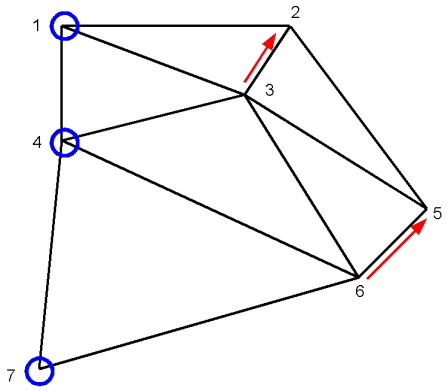
Číslo vrcholu	1	2	3	4	5	6	7
Smazán	-1	-1	1	-1	-1	-1	-1
Sloučit do	-1	-1	2	-1	-1	-1	-1
Sloučeno do mě	-1	1	-1	-1	-1	-1	-1

Tabulka 4.3: Příklad - Tabulka vrcholů po prvním kroku

Číslo vrcholu	1	2	3	4	5	6	7
Smazán	-1	-1	1	-1	-1	1	-1
Sloučit do	-1	-1	2	-1	-1	5	-1
Sloučeno do mě	-1	1	-1	-1	1	-1	-1

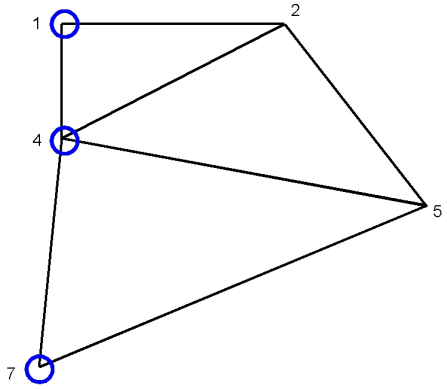
Tabulka 4.4: Příklad - Vyplněná tabulka vrcholů

Na obrázku 4.8 je zachycena situace před finálním odstraněním vrcholů. Červené šipky označují, jak se bude redukce provádět.



Obrázek 4.8: Příklad - Model s naznačenými změnami před redukcí

Na obrázku 4.9 je model po provedení redukce. Počet trojúhelníků se v našem ukázkovém příkladu snížil ze 6 na 3.



Obrázek 4.9: Příklad - Model po redukci

4.4 Vytvoření scény

Nyní již máme připraveny všechny potřebné nástroje, abychom mohli z původního modelu sestavit výslednou scénu, která bude zobrazovat krajinu v různých detailech podle

vzdálenosti od kamery, tak jak to specifikuje chunked-LOD algoritmus.

Program při vytváření scény postupuje následovně:

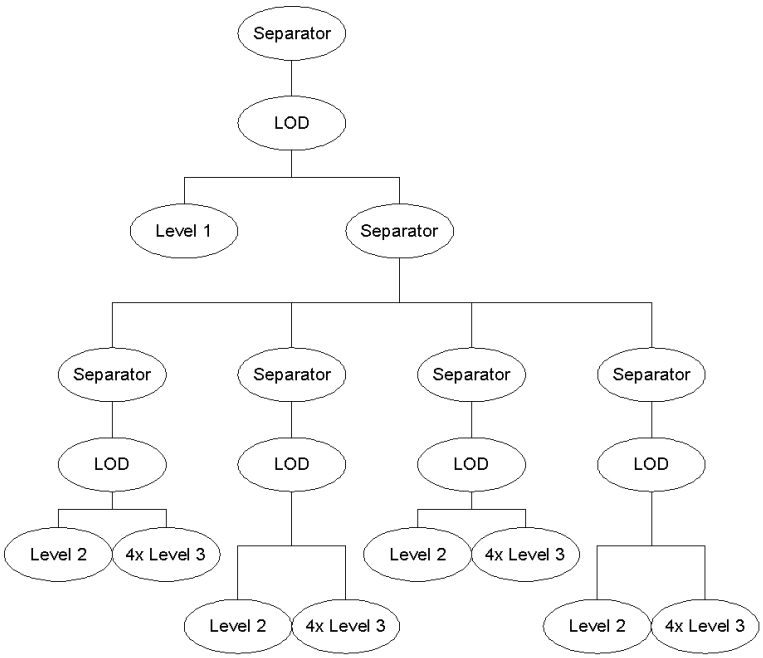
1. Načtení krajiny z externího souboru
   Název souboru je programu předán jako první parametr. Program si tento soubor načte standardním způsobem. Poté na načtený model aplikuje callback funkci přes všechny trojúhelníky. Tento přístup je použit, neboť soubory Open Inventoru mohou obsahovat předdefinované tvary (např. kvádr, koule atd.) a program musí mít přístup přímo k jednotlivým trojúhelníkům.

2. Určení rozměrů krajiny
   Nyní program projde všechny vrcholy a určí minimální a maximální rozměr krajiny. Takto zjištěné rozměry budou později použity pro prvotní spuštění dělení krajiny.

3. Výtvoření grafu scény
   Jako poslední krok program spustí rekursivní generování Chunked-LOD stromu. Nejdříve se z původní krajiny vyextrahuje segment o požadovaných souřadnicích. Pokud se nejedná o segment, který má být v maximálním detailu, aplikuje na něj redukční algoritmus. Poté vytvoří jeho 4 podsegmenty a operace pro ně rekursivně opakuje. Postupně se tak skládá výsledný graf scény.

   Narozdíl od aplikace pro zobrazování výškové mapy není potřeba, aby bylo nutné kontrolovat model „ručně“, neboť se za běhu aplikace nedogenerovávají žádné švy. Proto je možné použít předdefinovaný nástroj Open Inventoru v podobě třídy SoLOD, která potřebnou kontrolu zajistí za nás.

Výsledný graf scény pro 3 úrovně detailu zachycuje obrázek 4.10.



Obrázek 4.10: Schéma grafu scény

4.5 Výkonové vlastnosti

Všechny následující testy byly pořízeny na sestavách, které měly tyto parametry:

	Počítač A	**Počítač B**	**Počítač C**
Procesor	Athlon 64 3000+	Athlon XP 2000+	Sempron 2600+
RAM [MB]	1024	512	768
Grafická karta	Radeon 9600XT	GeForce 5700	GeForce 6600GT
Paměť GK [MB]	128	128	256
Verze ovladačů GK	6.3	83.10	84.20
Operační systém	WinXP SP2	WinXP SP2	WinXP SP2
Barva v grafech	Modrá	Červená	Zelená

Tabulka 4.5: Testované sestavy

Jako referenční byl použit model cilaos.iv, který mi laskavě poskytla firma CadWork. Tento model zachycuje „cirque de Cilaos“, což je kráter nacházející se na ostrově Reunion, který leží východně od Madagaskaru.

Chování programu lze ovlivnit dvěma parametry. První z nich je *-level*. Tento parametr určuje, jakou výšku bude mít výsledný quadtree. Druhým parametrem je *-reduce*. Tato hodnota učuje kolik procent trojúhelníků si přejeme odstranit ze segmentu při jednom průchodu redukční funkce. Následující testy budou postupně měnit tyto dva parametry. Budeme sledovat změny výkonu a náročnosti programu.

Doba generování

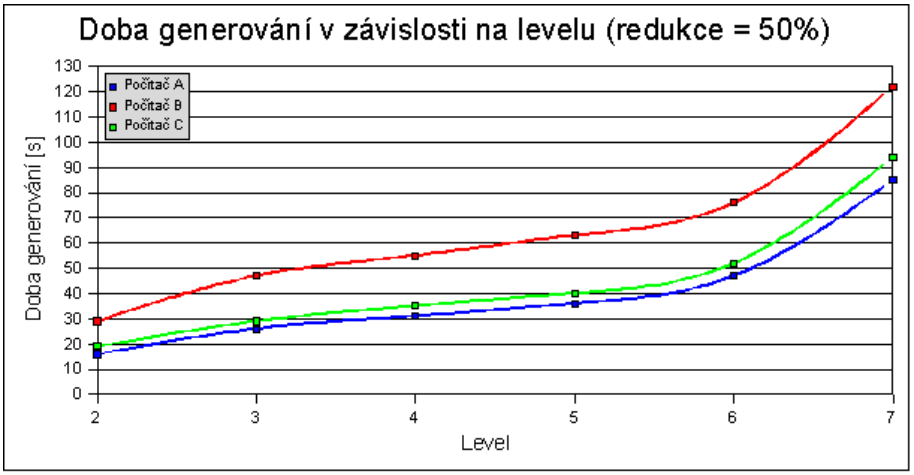
Nejdříve se zaměříme na dobu generování celé scény. Protože doba prvotního načtení modelu ze vstupního souboru je konstantní, měřena bude pouze doba generování chunked-LOD stromu. Z této doby je většina spotřebována na redukci trojúhelníků.

	Level					
Doba generování [s]	2	3	4	5	6	7
Počítač A	16	26	31	36	47	85
Počítač B	29	47	55	63	76	122
Počítač C	19	29	35	40	52	94

Tabulka 4.6: Závislost doby generování na levelu (redukce = 50%)

Z tabulky a grafu je vidět, že level zásadně ovlivňuje dobu generování. Křivky v poslední části grafu připomínají exponenciální charakteristiku.

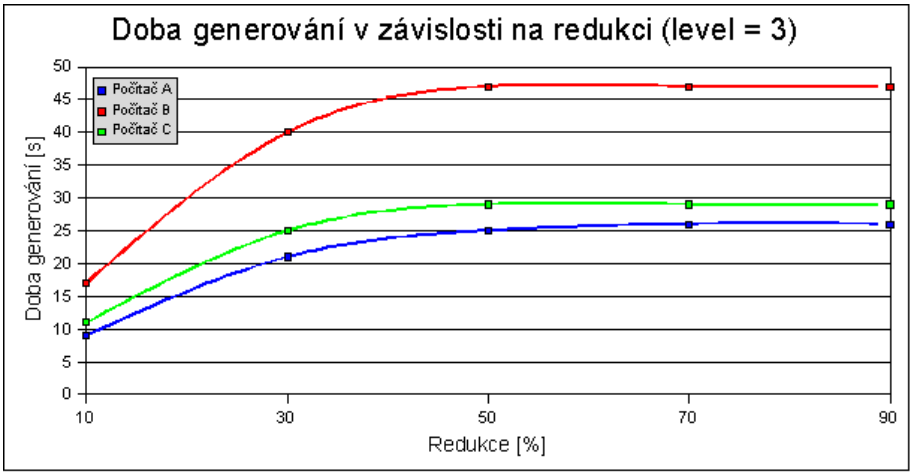
Jak nám graf 4.12 napovídá, doba generování se od hodnoty redukce 50 již téměř nemění. Tento jev je způsoben faktem, že při redukci trojúhelníků jsou před smazáním chráněny všechny vrcholy, do kterých byly vztaženy nějaké hrany. To znamená, že maximálně přibližně polovina vrcholů může být smazána, protože zbytek bodů jsou buď vztažné nebo hraniční body. Doba generování je závislá na výkonu procesoru. Proto Počítač B vykázal výrazně horší výsledky, než zbylé dvě sestavy.



Obrázek 4.11: Závislost doby generování na levelu (redukce = 50%)

	Redukce[%]				
Doba generování [s]	10	30	50	70	90
Počítač A	9	21	25	26	26
Počítač B	17	40	47	47	47
Počítač C	11	25	29	29	29

Tabulka 4.7: Závislost doby generování na redukci (level = 3)



Obrázek 4.12: Závislost doby generování na redukci (level = 3)

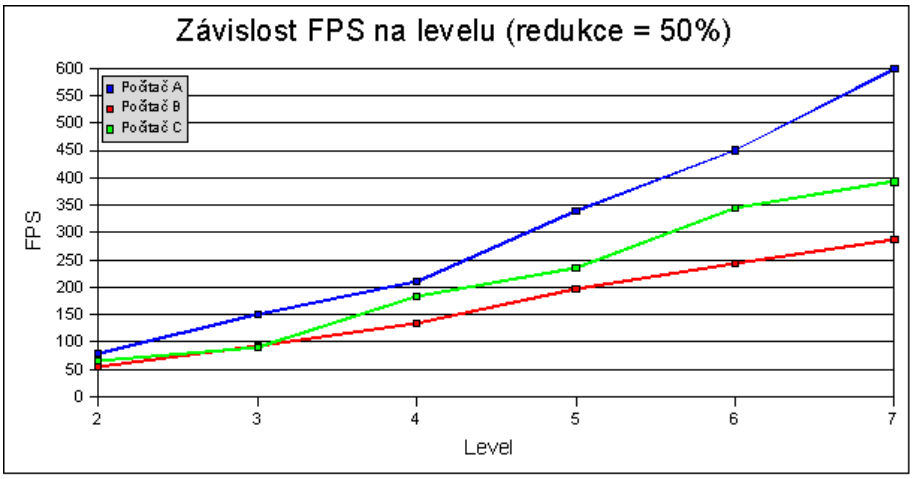
Počet snímků za sekundu

Dalším důležitým měřítkem při hodnocení výkonu grafické aplikace je počet snímků, které aplikace dokáže vyrenderovat za jednu sekundu. Častěji se tato hodnota označuje jako FPS, což je zkratka anglického *„Frames per second“*.

Protože tato aplikace dynamicky přepíná úrovně detailu a tím zásadně ovlivňuje hodnotu FPS, snažil jsem se vždy najít na krajinu takový pohled, při kterém se dostal počet snímků za sekundu na minimum. Intuitivně by se mohlo zdát, že nejnižší počet FPS bude, když se s kamerou přiblížíme co nejvíce k terénu, protože se terén přepne do maximálního detailu. Ve výsledku se ovšem stane to, že krajina bude v maximálním detailu pouze v nejbližším okolí kamery a zbytek se zobrazí s nižším detailem. Dále se uplatní optimalizace OpenGL, kdy modely za kamerou nejsou renderovány, protože nemohou být vidět. Paradoxně tak aplikace vyvine větší výkon, než by se na první pohled mohlo zdát. Nejnižší počet FPS docílíme ze střední vzdálenosti, kdy v zorném poli vidíme celou krajinu a ta je již přepnuta do vyššího detailu.

	Level					
FPS	2	3	4	5	6	7
Počítač A	80	150	210	340	450	600
Počítač B	55	92	135	197	243	288
Počítač C	66	91	184	235	344	393

Tabulka 4.8: Závislost FPS na levelu (redukce = 50%)



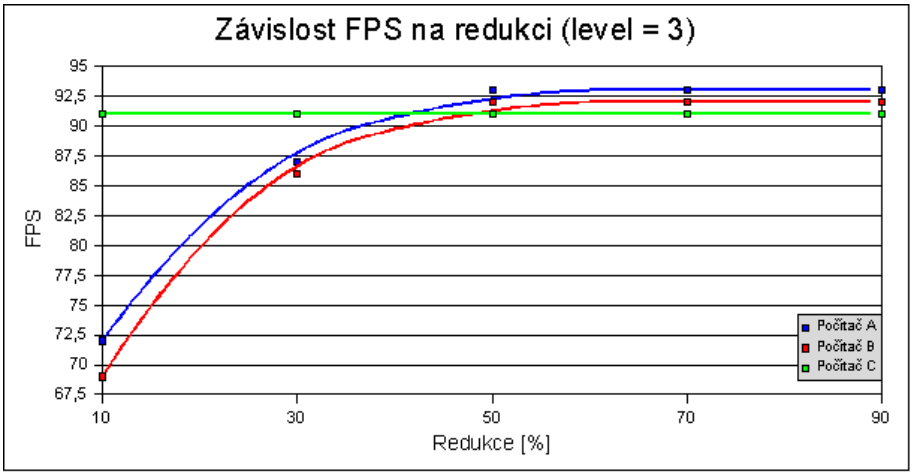
Obrázek 4.13: Závislost FPS na levelu (redukce = 50%)

Z grafu 4.13 je patrné, že změna levelu má zásadní vliv na FPS. Charakteristiky mají téměř lineární průběh. Proto je vhodné volit level, pokud možno, co nejvyšší. Negativní vlastností vzrůstající hodnoty levelu je prodlužování doby generování grafu scény, která se zvláště u levelu kolem hodnoty 10 stává neúnosnou. Dalším negativním důsledkem vysokého levelu jsou vzrůstající paměťové nároky. Více viz. graf 4.16.

Stejně jako u doby generování grafu scény se projevil fakt, že redukční algoritmus dokáže v jediném kroku odstranit maximálně zhruba polovinu trojúhelníků. Proto se při konstantní

	Redukce[%]				
FPS	10	30	50	70	90
Počítač A	72	87	93	93	93
Počítač B	69	86	92	92	92
Počítač C	91	91	91	91	91

Tabulka 4.9: Závislost FPS na redukci (level = 3)



Obrázek 4.14: Závislost FPS na redukci (level = 3)

hodnotě levelu již nezvyšuje počet FPS pro redukci větší než 50 (viz. graf 4.14). Zajímavým úkazem je také naprosto konstantní průběh křivky pro počítač C. Projevila se zde hrubá síla grafické karty, která dokázala zobrazit velký počet trojúhelníků bez poklesu FPS.

Obrázek 4.15 ukazuje graf procentuálního nárůstu výkonu v závislosti na levelu. Především při vyšších hodnotách levelu se křivky rozcházejí podle výkonosti jednotlivých sestav.

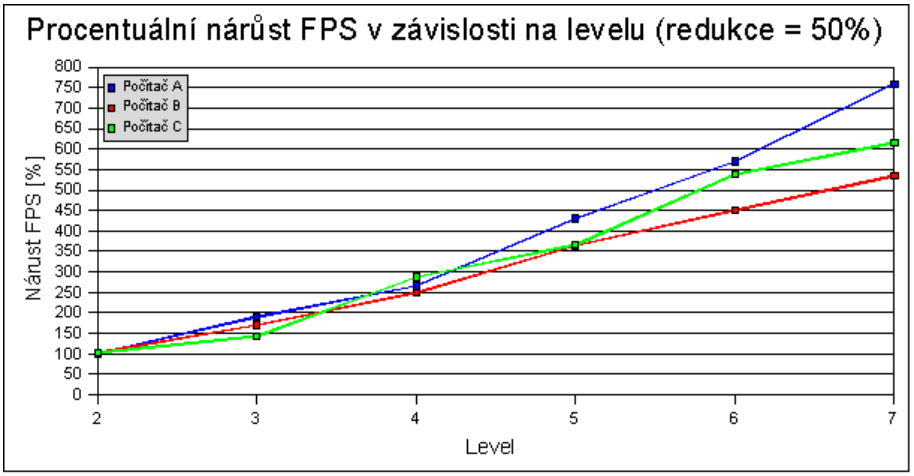
Hlavním nástrojem pro korekci FPS se nám tedy stává nastavení levelu. Procento redukovaných trojúhelníků je jako ovlivňující parametr aplikace odsunuto na druhé místo.

Paměťová náročnost

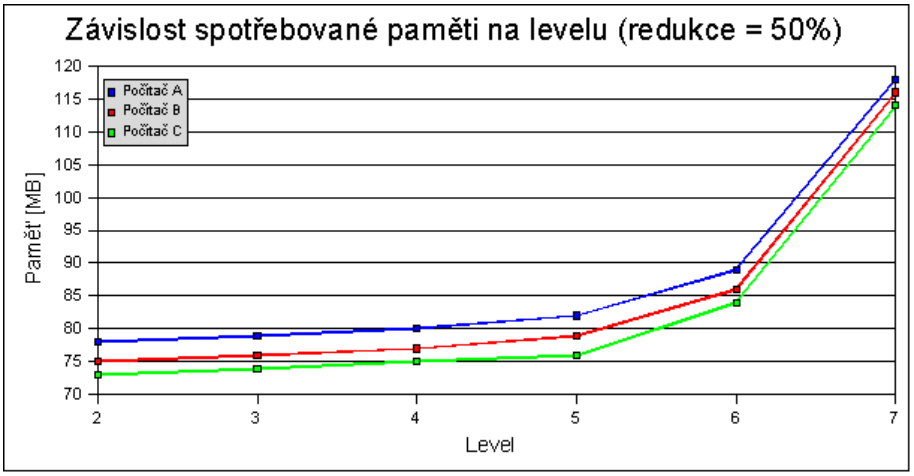
Neméně důležitým ukazatelem výkonu aplikace je paměťová náročnost. V následujících testech budeme opět postupně nastavovat level a poměr redukce trojúhelníků a budeme sledovat, kolik aplikace spotřebuje operační paměti počítače.

	Level					
Paměť [MB]	2	3	4	5	6	7
Počítač A	78	79	80	82	89	118
Počítač B	75	76	77	79	86	116
Počítač C	73	74	75	76	84	114

Tabulka 4.10: Závislost využité paměti na levelu (redukce = 50%)



Obrázek 4.15: Nárůst FPS v závislosti na levelu (redukce = 50%)



Obrázek 4.16: Závislost využité paměti na levelu (redukce = 50%)

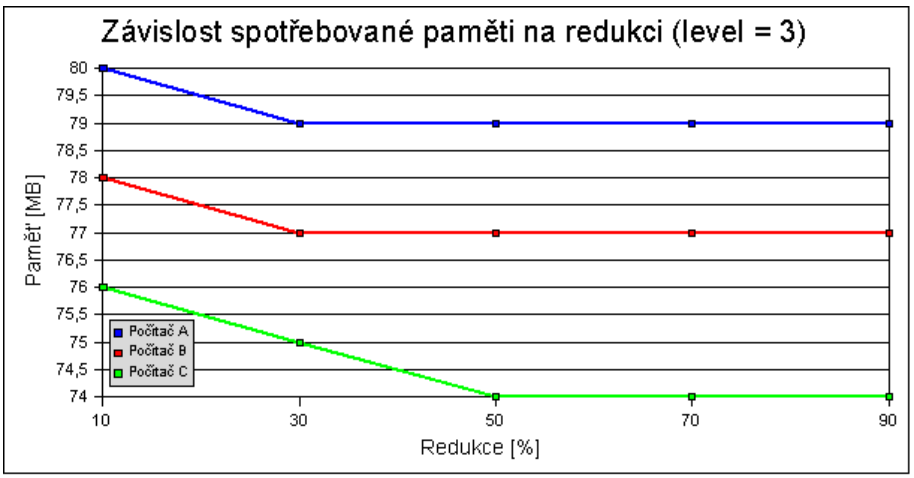
Graf 4.16 zachycuje závislost využité paměti na levelu. Pro nižší hodnoty levelu (2 - 5) se paměťové nároky téměř nemění. Drastičtější nárust požadavků je patrný až při hodnotě levelu 7, kdy se velikost požadované paměti oproti předchozí hodnotě levelu zvýšila o desítky MB. U nižších levelů tomu bylo pouze o jednotky MB.

Protože z celkové paměti, kterou aplikace vyžaduje, je převážná část potřeba pro uložení poměrně rozsáhlé textury a podíl paměti pro model je minoritní, nepřineslo zjednodušení modelu výraznější úsporu paměti.

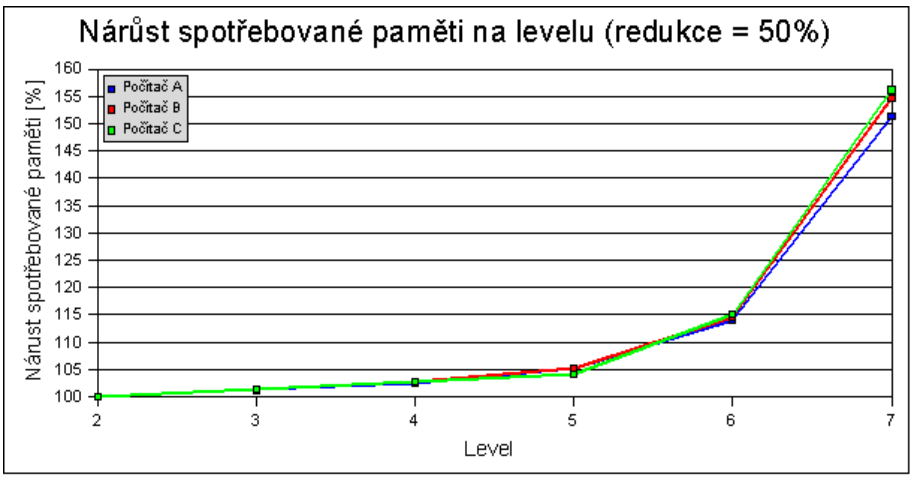
Graf 4.18 zobrazuje nárůst spotřebované paměti v procentech. Křivky pro jednotlivé sestavy se v podstatě kopírují. Z tohoto vyplývá, že množství spotřebované operační paměti nezáleží na velikosti paměti grafické karty, neboť počítač C má na grafické kartě 256 MB paměti a počítače A a B pouze 128 MB.

	Redukce[%]				
Paměť [MB]	10	30	50	70	90
Počítač A	80	79	79	79	79
Počítač B	78	77	77	77	77
Počítač C	76	75	74	74	74

Tabulka 4.11: Závislost využité paměti na redukci (level = 3)



Obrázek 4.17: Závislost využité paměti na redukci (level = 3)



Obrázek 4.18: Nárůst spotřebované paměti na levelu (redukce = 50%)

Závěry testů

Z výše provedených testů je patrné, že největší podíl na aktivním ovlivňování výkonu aplikace má především parametr level. S jeho pomocí jsme schopni radikálně ovlivňovat dobu generování grafu scény, množství zobrazených FPS a také paměťové nároky. Nastavení redukce trojúhelníků hraje při ovlivňování výkonu pouze „druhé housle“.

Osobně se domnívám, že ideálním nastavením pro testovaný model cilaos.iv je $level = 5$ a $redukce = 50\%$. Redukci nemá cenu nastavovat na vyšší hodnotu, neboť algoritmus stejně nedokáže odstranit požadovaný počet trojúhelníků. Nižší hodnota redukce nám naopak nepřinese výraznější snížení doby generování nebo paměťových nároků. Co se týče parametru level, myslím si, že hodnota 5 je optimální vzhledem k počtu FPS, doby generování a paměťových nároků. V grafech 4.11 a 4.16 můžeme sledovat, že u levelu většího jak 5 se závislost stává exponenciální, zatímco v grafu 4.13 pokračuje stále lineárně. Z toho vyplývá, že lineární nárust FPS bychom byli nuceni vykoupit exponenciálním nárustem doby generování a paměťových nároků a to podle mne rozhodně není dobrý poměr $cena/vykon$.

4.6 Vizuální vlastnosti

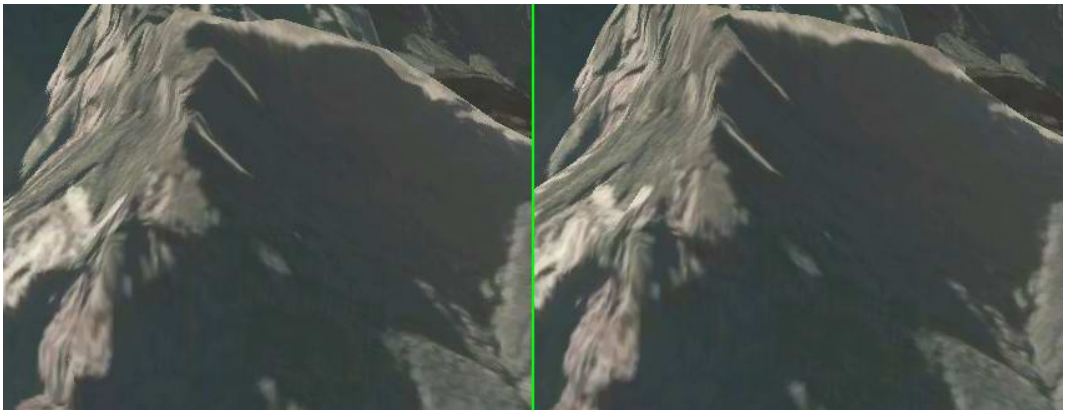
Redukční algoritmus

Při pohledu na krajinu a přibližování a oddalování pohledu, oku neuniknou občasné „díry“ v krajině. Jak jsem již psal výše, redukční algoritmus si nedokáže poradit se situacemi podobnými, jako je na obrázku 4.4. Tento nedostatek vidím jako hlavní narušení vizuálního dojmu z aplikace.

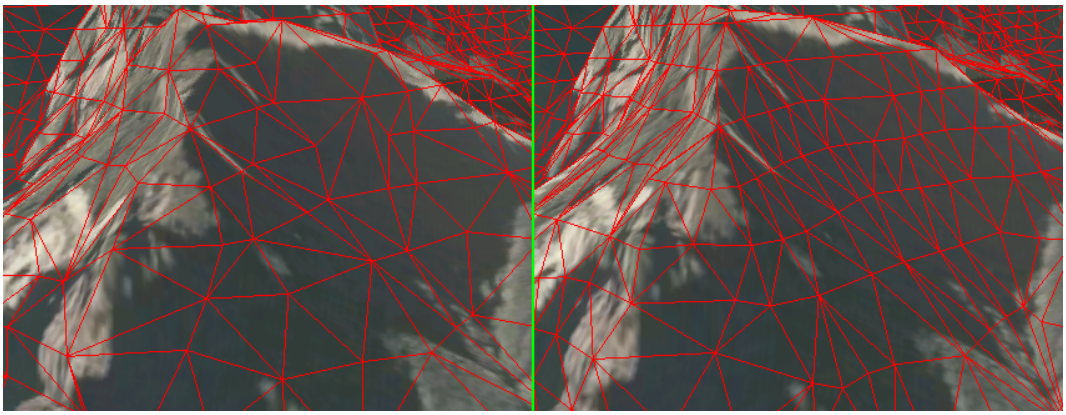
Dalším nedostatkem z vizuálního hlediska může být fakt, že redukční algoritmus vybírá hrany k odstranění podle jejich délky. Ne vždy je dobré v krajině odstranit nejkratší hranu. Jako optimálnější variantu bych např. viděl, hledání hrany podle objemové změny, kterou její odstranění přinese.

Přepínání detailu v modelu

Dalším důležitým aspektem vizuálního dojmu z krajiny jsou přechody mezi jednotlivými detaily modelu. Hlavní požadavek bude, aby změny detailu byly pokud možno co nejméně postřehnutelné.



Obrázek 4.19: Změna detailu v modelu



Obrázek 4.20: Změna detailu v modelu (drátěný model)

Na obrázku 4.19 je zachycena krajina těsně před (vlevo) a těsně po (vpravo) změně detailu. Na první pohled jsou si krajiny velmi podobné a téměř se zdá, že neobsahují žádnou změnu. Obrázek 4.20 zobrazuje stejné pohledy na krajinu, ovšem se zapnutým zobrazením triangulace. Zde je již patrné, že v modelu došlo ke změně detailu. Při používání aplikace jsou změny krajiny více patrné, neboť přepnutí detailu probíhá za běhu a uživatel v pohybu vidí, jak některé části krajiny jakoby poskočí. Osobně si ale myslím, že drobný pohyb při změně detailu krajiny není drastický a příliš neruší celkový vizuální dojem.

Kapitola 5

Závěr

U obou typů krajiny se podařilo implementovat zobrazování rozsáhlé krajiny v reálném čase pomocí chunked-LOD algoritmu tak, aby vzrostl výkon aplikace. V případě výškové mapy bylo oproti Ročníkovému projektu dosaženo dokonalého navázání segmentů krajiny. Nad rámec Semestrálního projektu bylo implementováno zobrazování obecného modelu a byly provedeny výkonové testy obou aplikací.

Tento projekt a jemu podobné jsou vystaveny v systému Wiki na internetové adrese http://merlin.fit.vutbr.cz/wiki/index.php?title=Inventor_Projects_2006.

5.1 Kde lze projekt využít?

Možné využití projektu vidím všude tam, kde je nutné najednou a v reálném čase zobrazovat velký kus krajiny. Například v leteckých simulátorech, kde je hlavním požadavkem komplexní pohled na krajinu, přičemž na detailu terénu nám prioritně nezáleží.

5.2 Další možná pokračování projektu

Aplikaci zobrazující výškovou mapu by bylo dobré rozšířit o zobrazování textur. Dalším zajímavým rozšířením funkčnosti by mohla být detekce kolizí s krajinou. Výšková mapa má k takovémuto úkolu lepší dispozice než obecný model, protože lehce dokážeme vypočítat přesnou výšku každého bodu v krajině. V případě obecného modelu by již toto rozšíření nebylo zcela triviální.

Jako hlavní nedostatek aplikace pracující s obecným modelem vidím občasné nekonzistence v modelu, které jsou způsobeny redukčním algoritmem v kombinaci s nedokonalým vstupním modelem. Vizuální dojem obou aplikací by také výrazně vylepšilo umístění krajin do skyboxu a přidání některých částicových efektů jako je například déšť, prach nebo mraky.

Další slabina aplikace pro obecný model je v textůře. Rozměr textury pro model cilaos.iv je 3000x2000 pixelů. Některé grafické karty umožňují zobrazení textur o rozměrech pouze 2048x2048 pixelů. Tento nedostatek by bylo dobré odstranit rozdělením textury na několik menších. V tomto případě, by se již nemohlo provádět dělení segmentů po nerovných souřadnicích, protože nově vzniklé textury by byly pravoúhlé. Možným řešením by mohlo být pravoúhlé dělení segmetů ve vyšších patrech quadtree.

Literatura

[1] Pavel Černý. *Zobrazování krajiny a jeho optimalizace pomocí ROAM algoritmu*. 2006.

[2] Jan Pečiva. Open inventor tutorial. `http://www.root.cz/serialy/open-inventor/`.

[3] Ken Perlin. Making noise. `http://www.noisemachine.com/talk1`.

[4] Sánchez-Crespo. *Core Techniques and Algorithms in Game Programming*. 2003.

[5] WWW Stránky. Roaming terrain: Real-time optimally adapting meshes. `http://www.llnl.gov/graphics/ROAM/roam.pdf`.

[6] Thatcher Ulrich. *Rendering Massive Terrains using Chunked Level of Detail Control*. 2002.

Seznam obrázků

Seznam tabulek

Příloha A

Uživatelský manuál

Chování obou aplikací lze ovlivnit pouze parametry, se kterými jsou spuštěny. Jejich seznam, výchozí hodnoty a význam shrnují následující tabulky.

Parametr	**Implicitně**	**Význam**
-seed	0	Inicializuje generátor náhodných čísel
-size	512	Velikost celé krajiny
-dimension	33	Velikost segmentu
-distance	100	LOD vzdálenost
-filterQ	0.5	Parametr filtrovací funkce $(0 - 1)$
-hq	0	Režim tesselace krajiny $(0, 1, 2)$

Tabulka A.1: Parametry programu - Aplikace pro výškovou mapu

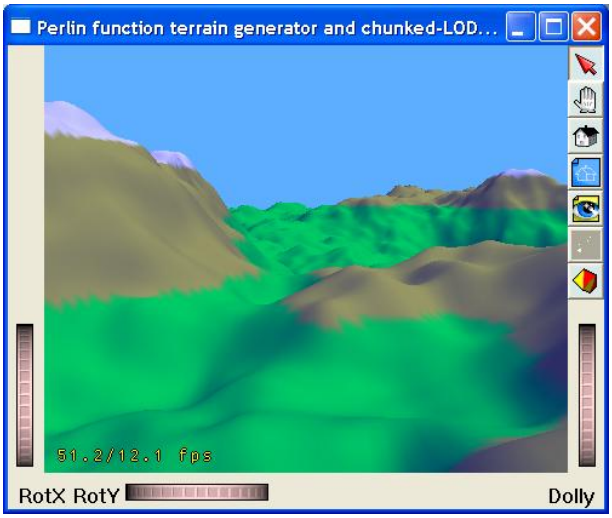
Parametr	**Implicitně**	**Význam**
-operation	0	Druh operace s modelem
-level	5	Výška LOD stromu
-reduce	50	Procento trojúhelníků k odstranění
-x1	0.0	Souřadnice segmentu pro extrakci
-y1	0.0	Souřadnice segmentu pro extrakci
-x2	0.0	Souřadnice segmentu pro extrakci
-y2	0.0	Souřadnice segmentu pro extrakci

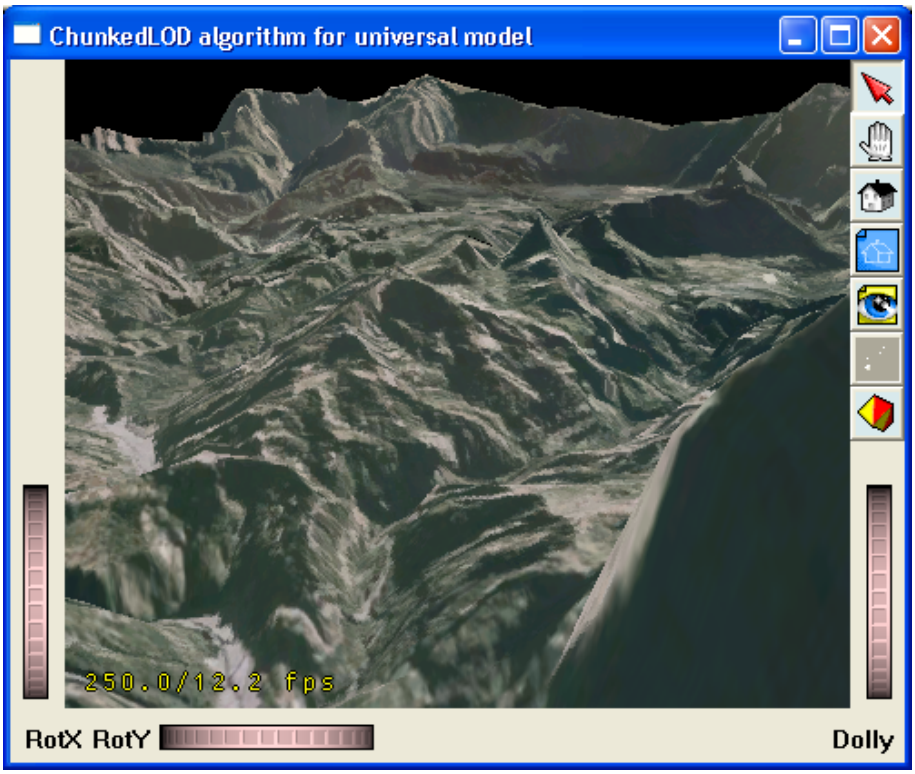
Tabulka A.2: Parametry programu - Aplikace pro obecný model

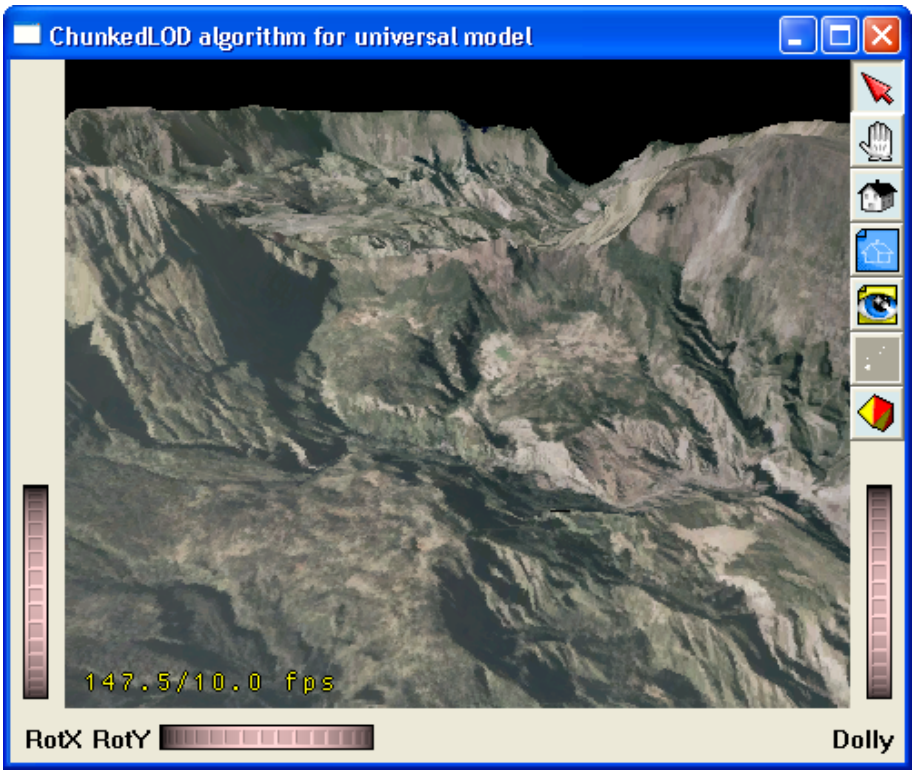
Všechny parametry obou programů lze přehledně nastavit v přiložené spouštěcí aplikaci.

Příloha B

Obrazová příloha

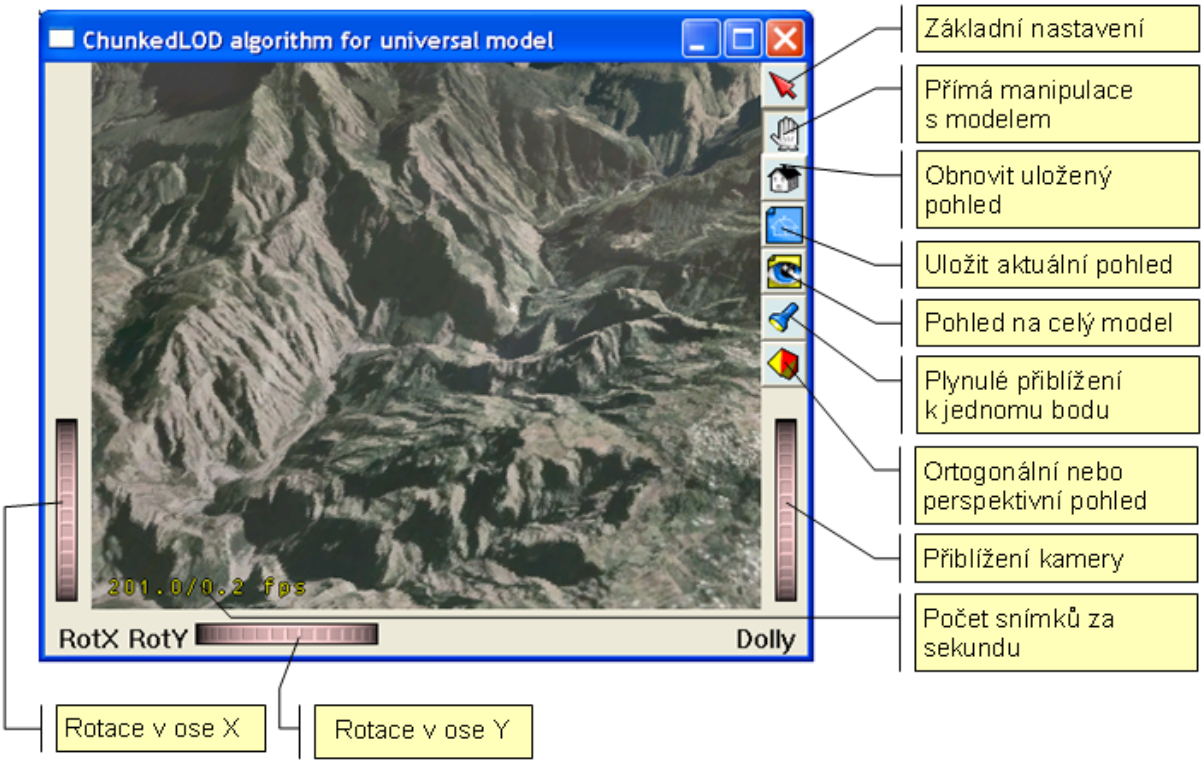



ChunkedLOD algorithm for universal model
250.0/12.2 fps
RotX RotY
Dolly


ChunkedLOD algorithm for universal model
147.5/10.0 fps
RotX RotY
Dolly

Příloha C

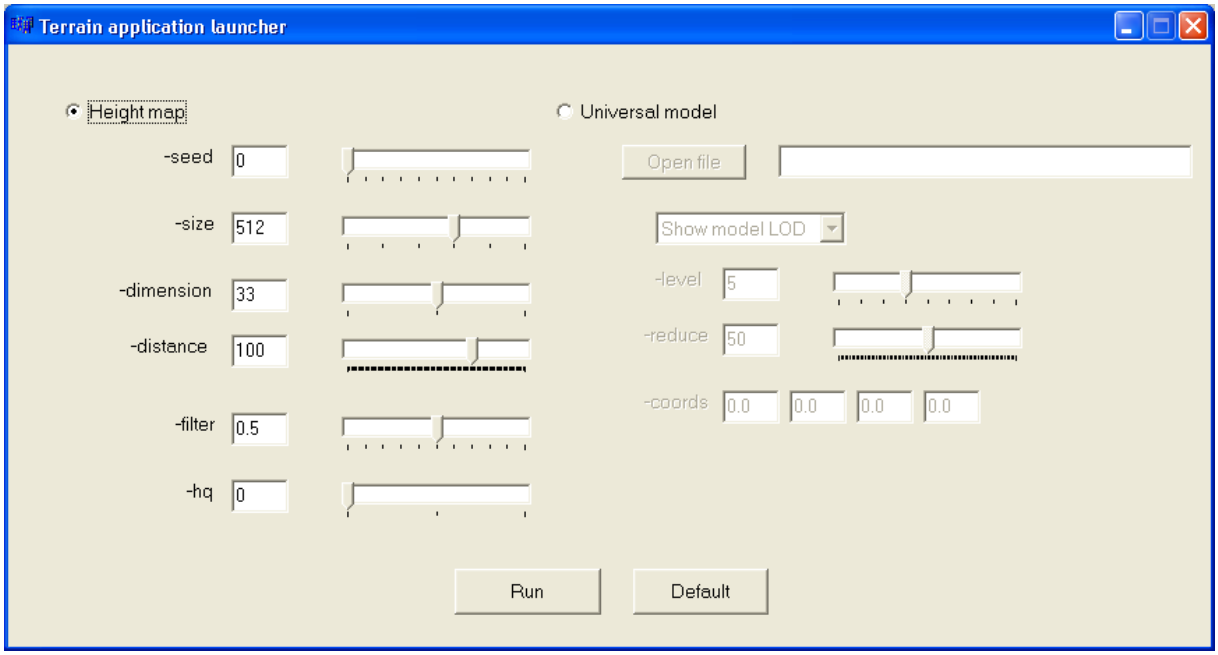
Vzhled aplikace



Obrázek C.1: Vzhled aplikace s vysvětlivkami

Příloha D

Spouštěcí aplikace



Obrázek D.1: Vzhled spouštěcí aplikace