České vysoké učení technické v Praze
Fakulta elektrotechnická

BAKALÁŘSKÁ PRÁCE

Efektivní zobrazení terénu pro glass-cockpit

Praha, 2013 **Autor: Martin Kerhart**

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 23. 5. 2013

M. Kurhan

podpis

Poděkování

Zde bych rád poděkoval vedoucímu práce Ing. Tomáši Levorovi za čas věnovaný kontrole textu a řešení různých problému, jež se vyskytly během vypracovávání práce. Dále mé díky patří rodině za poskytnutou podporu a klid ke psaní.

Abstrakt

Level of detail algoritmy umožňují rychlé a efektivní zobrazení komplexních 3D modelů. Jejich typickým použitím je vykreslování terénu. Náplní této práce bylo studium level of detail algoritmů a jejich využití při zobrazení terénu prostřednictvím 3D grafiky. Po teoretickém rozboru následuje popis dvou implementovaných algoritmů a jejich důkladné porovnání. Efektivnější algoritmus byl použit ve finální aplikaci k zobrazení rozlehlé krajiny. Povrch terénu je tvořen kombinací výškových dat z projektu The Shuttle Radar Topography Mission a leteckých snímků ze serveru Mapy.cz.

Aplikace slouží jako prototyp navigace v glass cockpitu pro ultralehká letadela. Program lze napojit na FlightGear Flight Simulator, získaná simulovaná data jsou následně zobrazena na grafických leteckých přístrojích.

Abstract

Level of detail algorithms allow a fast and effective visualization of complex 3D models. Their typical usage is a terrain rendering. The main goal of this thesis is the study of level of detail algorithms and their application for displaying a terrain in 3D graphics. After the theoretical elaboration follows the description of two algorithms and their comparison. The more effective algorithm was used in the final application abble to render a vast terrain. The surface of the terrain is made by combination of altitude data from The Shuttle Radar Topography Mission and aerial photographs obtained from the server Mapy.cz.

The application serves as a prototype of navigation in glass cockpit for ultralight airplanes. The program can be connected to FlightGear Flight Simulator and then gathered data are visualized on the graphical aerial instruments.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Martin Kerhart**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Efektivní zobrazení terénu pro glass-cockpit**

Pokyny pro vypracování:

1. Objektivně srovnejte skupinu algoritmů pro zobrazení terénu.
2. Implementujte vybraný Level of Detail algoritmus pro rychlé zobrazení terénu na displeji glass-cockpit zařízení.
3. Implementovaný kód rozšiřte o zobrazení letadlových přístrojů.
4. Demonstrujte realizovaný program zobrazením terénu a letových dat ve spojení s leteckým simulátorem.

Seznam odborné literatury:

[1] ROAM algoritmus (http://www.cognigraph.com/ROAM_homepage)
[2] SOAR algoritmus (https://computation.llnl.gov/casc/SOAR/SOAR.html)
[3] Geometrical MipMapping (http://www.flipcode.com/archives/article_geomipmaps.pdf)
[4] SRTM podklady (http://www2.jpl.nasa.gov/srtm)

Vedoucí: Ing. Tomáš Levora

Platnost zadání: do konce zimního semestru 2013/2014

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 11. 2. 2013

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Přestože se počítače neustále zdokonalují a výpočetní výkon roste, v mnohých případech je zobrazení grafických dat stále problém. Typické je to pro 3D počítačovou grafiku. Ať už ve hrách, medicíně nebo ve vědě, velké množství objektů a jejich vysoké detaily představují pro procesor a grafickou kartu značnou zátěž. U velmi detailních modelů nebo komplexních scén čítajících tisíce objektů může počet trojúhelníků snadno dosáhnout i několika milionů. [1] Vykreslení jednoho snímku pak může trvat až stovky milisekund. Přitom pro zajištění plynulého vykreslování obrazu je třeba alespoň 20 – 30 snímků za sekundu. Když se k tomu přidají textury, případně osvětlení, stínování a další efekty, náročnost zobrazení značně stoupá. Ovšem ne vždy jsou všechny zobrazované detaily jasně patrné – v tu chvíli je na místě použití level of detail algoritmu.

Správně navržený level of detail algoritmus (dále LOD) dokáže nahradit detailní část modelu hrubším ekvivalentem, aniž by to bylo patrné. Ku příkladu na vzdálených objektech zabírajících na obrazovce jen pár pixelů není poznat, zda se skládají z 10 či 10000 trojúhelníků. Vedle samotných objektů, které lze poměrně jednoduše staticky nahradit méně podrobným modelem, hraje významnou roli terén.

Terén je obvykle souvislý – nedá se celý, ani jeho část, jen tak nahradit jiným modelem. LOD je zde však nutností, protože čím je terén členitější a podrobnější, tím více trojúhelníků je třeba pro jeho reálné zachycení. Je-li navíc nutno zobrazit krajinu ve velké vzdálenosti, stává se počet trojúhelníků neúnosný. Různé metody, jak lze přesto terén efektně zobrazit, jsou popsány v následujících kapitolách.

Pro svou práci jsem si vybral dva LOD algoritmy – Stateless, One-pass Adaptive Refinement (SOAR) a Geometrical MipMapping (GMM). Oba jsem je implementoval a otestoval na zkušebních datech. GMM z porovnání vyšel podstatně lépe, proto jsem na něm založil jádro aplikace navigace ultralehkých letadel. Struktura terénu je gene-

rována z výškových dat projektu The Shuttle Radar Topography Mission (SRTM), model terénu je následně potažen texturou z leteckých snímků získaných od společnosti Mapy.cz. Výšková data jsou v podobě pravidelné mřížky bodů vzorkované po třech vteřinách v zeměpisných souřadnicích viz. [2]. Vedle samotného terénu je program schopen zobrazit různé letecké přístroje jako výškoměr a gyroskop. V aplikaci se dá „proletět“ ručně pomocí klávesnice a myši nebo skrze FlightGear Flight Simulator (komunikace je zajištěna UDP pakety).

Tato práce shrnuje problematiku level of detail algoritmů a obšírněji pojednává o dvou z nich (SOAR a GMM). Dále popisuje kompletní implementaci programu zobrazujícího terén s GMM algoritmem a tento základní program rozšiřuje o vyobrazení leteckých přístrojů.

1.1 Level of detail algoritmy ve 3D grafice

Využití level of detail algoritmů je zřejmé – snížit počet polygonů, zátěž procesoru a grafické karty a zvýšit rychlost zobrazení a počet vykreslených snímků za sekundu. Objekty i terén vyžadují zvláštní přístup, obecně však lze LOD algoritmy rozdělit na statické a dynamické.

Statické za běhu programu nevytvářejí novou strukturu, jen detailnější model nahradí hrubším, načteným z paměti. Ideální využití statického LOD je ve scénách s velkým množstvím menších objektů. Dynamické LOD oproti tomu stávající detailní model dle parametrů scény přetvoří na model s nižším počtem trojúhelníků. Tohoto přístupu se využívá především u velkých objektů a terénu.

1.1.1 Statický LOD

Principem statických level of datail algoritmů je načítání předem vytvořených modelů z paměti. To znamená nízké zatížení procesoru – není třeba složitě vytvářet novou strukturu objektu. Na druhou stranu to vede k vyšším paměťovým nárokům. Z diskrétních vlastností tohoto přístupu plyne nemožnost plynulého přechodu mezi jednotlivými stupni detailů, což vede k tzv. poppingu - hrubý model se najednou změní na podrobný a naopak.

Procesor zde pouze musí vyřešit, jaký konkrétní model pro danou polohu kamery a stav scény zvolit. Adekvátní stupeň detailů se dá lehce vybrat na základě vzdálenosti objektu od kamery a jeho velikosti, tedy podle plochy, kterou zabírá na obrazovce.

Generování modelů s různou úrovní detailů je možno provádět ručně nebo automaticky s využitím technik dynamického LOD. Je třeba vyvarovat se zásadních deformací objektů – lehce můžou zmizet tenké části nebo průzory v objektu.

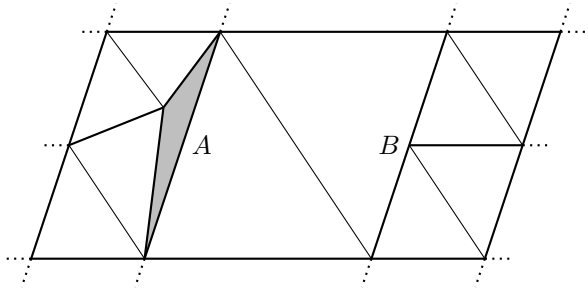
Hlavní využití statického LOD je v grafických enginech zobrazujících velké množství objektů na rozsáhlé ploše, typicky to jsou mapy s vymodelovanými budovami, stromy apod.

1.1.2 Dynamický LOD

Dynamické level of detail algoritmy v reálném čase generují zcela novou strukturu objektů. To znamená zvýšení výpočetních nároků při plynulých změnách úrovní detailů, které splývají v jeden celek. Tyto algoritmy se také označují jako view dependable: vždy je zobrazeno jen aktuálně potřebné množství polygonů rozložených v objektu podle polohy kamery.

Při pohybu kolem dynamicky se měnícího modelu je potřeba na jedné straně detaily zvyšovat a na druhé naopak ubírat. Méně podrobné struktury lze dosáhnout slučováním (collapsing) hran, trojúhelníků nebo celých oblastí modelu v jeden vertex. Opačným postupem je rozdělení (spliting) jednoho vertexu ve více. Pokud oba procesy předchází plynulé přiblížení či oddálení vertexů, jedná se o tzv. morphing – objekt se téměř nepozorovatelně mění bez nežádoucího poppingu.

Struktura 3D modelů se skládá z přesně navazujících trojúhelníků a při neopatrné modifikaci modelu za chodu může lehce dojít k jeho narušení. U přechodu mezi dvěma stupni úrovní detailů se můžou vyskytnou tzv. t-junction a cracks. Crack v modelu vzniká, jestliže jeden vertex není umístěn do roviny všech sousedních trojúhelníků – v modelu je mezera, viz. obr. 1.1. Pokud někde sousedí dva trojúhelníky s jednou hranou třetího trojúhelníku, jedná se o t-junction. Kvůli zaokrouhlování reálných čísel nemusí být společný vertex dvou trojúhelníků přesně umístěn na hraně třetího trojúhelníku. Při renderování je pak v onom místě patrná mezera. Oběma těmto poruchám se dá předejít pečlivou distribucí změn mezi sousedními trojúhelníky. Bohužel tím většinou stoupá složitost a klesá efektivita LOD algoritmů, jelikož se musí navíc zobrazit i trojúhelníky umožňující tento přechod.



Obrázek 1.1: Chyby v povrchu modelu. Vertex B označuje t-junction, A označuje crack, vzniklá díra v modelu je zvýrazněna.

Vedle problémů s konzistencí povrchu modelů je třeba dodržet požadavky na přesnost zobrazení objektů v dané aplikaci. Například počet polygonů u modelu lanového mostu by se dal lehce snížit vypuštěním lan, ale v architektonické aplikaci je jejich zobrazení zásadní, byť by měly šířku jen jednoho pixelu. Některé aplikace vyžadují zachování základního tvaru, jiné dodržení objemu či obrysu objektu.

V případě terénu lze mezi dynamickými LOD algoritmy rozlišit dvě hlavní skupiny podle uspořádání dat, z kterých je tvořen model. První skupina se vyznačuje pravidelným rozmístěním vertexů v předem stanoveném 2D rastru, pouze třetí souřadnice je libovolná. Snižuje se tak paměťová náročnost a zjednodušují se některé výpočty jako detekce kolizí apod. Druhá skupina využívá nepravidelnou síťovinu, což umožňuje v oblastech s vyšším členěním terénu soustředit větší množství trojúhelníků než v plochých částech. Pro popis stejného terénu nepravidelnou strukturou je většinou potřeba méně trojúhelníků s lepším vizuálním výsledkem než u pravidelné sítě. Kvůli značné komplexnosti nepravidelné síťoviny je její algoritmické zpracování pro LOD poněkud náročnější. Volba LOD z hlediska struktury modelu závisí na reprezentaci výškových dat terénu.

1.2 Příklady LOD

Technika zobrazení rozsáhlých terénů v počítačové grafice je předmětem častého zkoumání. Existuje proto celá řada prací popisujících různé algoritmy, které tento problém řeší z různých úhlů a pro různé aplikace.

Vedle mnou implementovaných algoritmů (Stateless, One-pass Adaptive Refinement a Geometrical MipMapping), zde uvádím ještě dva další algoritmy, které významně ovlivnily vývoj v této oblasti.

1.2.1 Continuous LOD for Height Fields

Jeden z prvních algoritmů zobrazujících rozlehlý spojitý terén [1]. Popsal ho Lindstrom aj. v [3]. Algoritmus pracuje s pravidelnou strukturou terénu, ten je rozdělen do bloků a jednotlivé bloky pak reprezentuje binární strom pravoúhlých trojúhelníků. Listy stromu obsahují nejdetailnější strukturu terénu, každý vyšší uzel pak jedním větším trojúhelníkem nahrazuje všechny menší z uzlů pod ním.

Proces LOD spočívá v hrubém zjednodušení celých bloků a následné úpravě geometrie v rámci bloků. Postupným slučováním trojúhelníků od nejnižší pozice ve stromu se dosahuje nejnižšího povoleného stupně detailů v dané části terénu. Rozhodnutí, zda dva trojúhelníky sloučit a postoupit na vyšší úroveň, spočívá v porovnání stávajícího a nového povrchu. Vypuštěním společného vertexu dvou sousedních trojúhelníků vzniká jeden větší a tím i chyba vůči detailnějšímu modelu. Je-li chyba nižší než nastavená prahová hodnota, dojde ke snížení detailů.

1.2.2 Real-time Optimally Adapting Meshes

Zřejmě nejznámější LOD algoritmus je Real-time Optimally Adapting Meshes (ROAM), autorem je Duchaineau aj. [4]. Podobně jako v [3] je pravidelná struktura terénu rozdělena do binárního stromu, terén však není dělen do bloků, ale je modifikován jako celek. LOD postupuje od kořene stromu k listům rozdělováním velkých trojúhelníků na menší. V paměti se uchovává aktuální stav modelu a LOD zpracovává jen ty části terénu, kde je to aktuálně třeba - ROAM tak těží z koherence dvou následných snímků. Pořadí zpracování trojúhelníků řídí dvě prioritní fronty, jedna pro dělení a druhá pro slučování trojúhelníků. Prioritní pořadí rozdělování polygonů umožňuje udržovat určitý maximální počet trojúhelníků.

1.2.3 Statle-less, one-pass adaptive refinement

Autory toho LOD algoritmu jsou P. Lindstrom a V. Pascucci [5], je jedním ze zde implementovaných algoritmů. Podobě jako ROAM vytváří model terénu postupným dělením velkých trojúhelníků na menší, místo binárního stromu však využívá quad tree topologii (jeden vertex má více rodičů). To společně se speciálními předem vypočtenými parametry zajišťuje bezchybný povrch modelu.

1.2.4 Geometrical Mipmapping

Druhým z porovnávaných algoritmů v této práci je Geometrical Mipmapping, Navrhl ho W. de Boer pro projekt eMersion [6]. Oproti výše zmíněným algoritmům se výrazně odlišuje vynecháním per vertex kontroly tolerance. Místo modifikace struktury terénu na úrovni vertexů mění celé bloky terénu. Autor tento postup přirovnává k mipmappingu textur. Všechny bloky mají stejnou velikost a volba aktuální úrovně detailů je nezávislá na sousedech, návaznost mezi bloky je řešena explicitně. Ve výsledku blokový přístup znamená méně plynulý gradient detailů v rámci celého terénu, ale zobecněnou volbou detailů na bloky terénu je dosaženo značného zrychlení. GMM je podrobně popsán a otestován v příslušné části práce.

Kapitola 2

Implementace LOD

V aplikaci jako je navigace pro letadla zobrazující 3D terén, je použití level of detail algoritmu přímo zásadní. S jeho pomocí lze vykreslit 3D mapu do mnohem větší vzdálenosti než běžným způsobem. Pokud se navíc použijí letecké snímky, jako textura zobrazeného modelu terénu, pilot získá možnost prohlédnout si krajinu desítky kilometrů před sebou bez ohledu na počasí nebo denní dobu.

V historii LOD zpracovávající terén se objevily dva zásadní přístupy. V době, kdy nebyly k dispozici moderní a výkonné grafické karty jako dnes, bylo zapotřebí co nejvíc omezit počet trojúhelníků výpočty na procesoru. Grafické kartě se pak k renderování zasílal co nejlépe optimalizovaný set vertexů. Takovým algoritmem je zmíněný ROAM nebo SOAR. Oba se snaží ušetřit každý trojúhelník, potažmo vertex. Vyhodnocení, kterou úroveň detailů použít, probíhá na úrovni jednotlivých vertexů, tím se dosáhne minimálního počtu polygonů pro aktuálně zobrazenou scénu.

S nástupem grafických karet s velkou pamětí a vysokým výpočetním výkonem, schopným zobrazit milióny trojúhelníků za sekundu, přestal být celkový počet polygonů do jisté míry zásadní problém. Trendem se stal postup nahrání co největšího množství geometrie do paměti grafické karty a následné opakované vykreslování bez zvláštní zátěže procesoru. Dynamicky měnící terén se bohužel delší dobu v paměti uchovávat nedá, ale díky vysoké rychlosti zobrazení grafických čipů odpadá nutnost šetřit každý trojúhelník. Tohoto přesně využívá LOD algoritmus Geometrical MipMapping. Místo jednotlivých vertexů zkoumá celé bloky terénu čítající stovky i tisíce trojúhelníků. Vyhodnocení použité úrovně detailů se neliší, pouze probíhá v mnohem hrubším měřítku, což obvykle znamená nutnost použít více trojúhelníků, než vyžaduje per vertex metoda. Tento nedostatek je však bohatě vyvážen podstatně kratším časem nutným pro generováním celého modelu terénu, tedy i nižší zátěží procesoru.

Programovací jazyk pro implementaci jsem zvolil Java v kombinaci s JOGL (Java OpenGL). Java mi umožňuje implicitní podporu mnohých platforem, JOGL pak pohodlný přístup k metodám OpenGL. Prvně jsem si připravil jednoduchou grafickou aplikaci s pohybující se kamerou a různými ovládacími prvky. V této aplikaci jsem pak testoval implementované algoritmy.

2.1 Statle-less, one-pass adaptive refinement

Level of detail algoritmus Statle-less, one-pass adaptive refinement (SOAR) popsal P. Lindstrom společně s V. Pascucci v práci nazvané Visualization of Large Terrains Made Easy. Popsaný algoritmus byl uveden jako jednoduše implementovatelný, zdrojově nenáročný a výkonný [5]. O rok později byla autory vydána druhá publikace Terrain Simplification Simplified: A General Framework for View-Dependent Out-of-Core Visualization, rozšířená o detailnější popis vylepšeného LOD algoritmu.

Hlavní specifikací SOARu je práce s pravidelně vzorkovanými výškovými daty. Pro každý snímek je generován nový model terénu rekurzivním dělením přepon trojúhelníků od velkých po nejmenší (top-down přístup). Při návrhu SOARu se podařilo rozdělit jednotlivé LOD operace do nezávislých funkčních bloků, díky čemuž je možno jednoduše přepínat a testovat různá řešení jednotlivých fází. Lze tak zkoumat vlastnosti různých metrik, na nichž závisí volba aktuální úrovně detailů. Do jádra algoritmu, jež prochází vertexy terénu a vyhodnocuje, které zahrnout do vykresleného modelu, se dá jednoduše přidat funkce view-frustum culling a snížit tak počet testovaných vertexů. Proces generování souřadnic vertexů pro renderování grafickou kartou je možno obohatit o morfing a minimalizovat tím viditelné změny struktury terénu. Autory je každá tato část pečlivě rozebrána a je navrženo vhodné řešení.

Vedle algoritmu pro redefinování geometrie terénu je v [7] popsána metoda uspořádání dat na disku pro zajištění co nejmenšího počtu chyb při stránkování paměti. Samotné stránkování je přitom ponecháno ve správě operačního systému. Uvedený postup by měl umožnit spolehlivě zpracovávat data o velikosti mnoha GB a tedy zobrazit terén tzv. out-of-core, kdy fyzická paměť je příliš malá na pojmutí celého setu dat.

2.1.1 Implementace

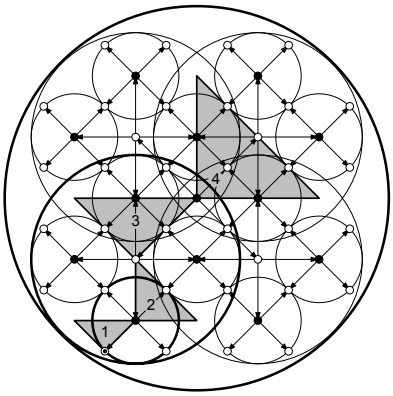
V této části je uveden postup, jak je SOAR implementován na základě informací získaných z [7]. Funkce algoritmu ve zkratce je následující: SOAR pro každý snímek animace generuje nový model terénu s hustotou trojúhelníků rozloženou tak, aby celkový počet zobrazených trojúhelníků byl co nejnižší a vizuální informace o struktuře terénu co nejvíce zachována. Vzniká tak objekt tvořený místy hustší, místy řidší sítí trojúhelníků. Přechody mezi dvěma úrovněmi detailů je třeba ošetřit kvůli možnému vzniku spár, toho je docíleno rozložením celého terénu do pravidelného stromu o čtyřech větvích (quad tree). Úroveň detailů se hierarchicky šíří z potomka na rodiče, to spolu se speciálními parametry popsanými níže zaručuje vždy souvislý povrch terénu.

2.1.1.1 Metrika

Termínem metrika se označuje funkce která pro aktuální polohu kamery udá zobrazovací chybu jednoho vertexu. Volba aktuálně vhodné úrovně detailů pak spočívá v porovnání této chyby s uživatelem nastavenou tolerancí. Terén ve scéně je tvořen menším počtem vertexů než originál, je tedy do jisté míry deformován. V místech kde jsou vertexy vynechány je povrch pouhou aproximací původního podrobného modelu. Do jaké míry se zobrazený terén liší od originálu v místech vynechaných vertexů (na souřadnicích x, y) udává objektově prostorová chyba (object space error). Je to výškový rozdíl mezi vynechaným vertexem a polygonem tvořeným okolními vertexy. Tato hodnota se však z různých úhlů pohledu a hlavně vzdálenosti kamery zdá jiná, je jí proto třeba převést na chybu v prostoru obrazovky (screen space error) užitím perspektivní projekce na plochu. Tím je získána chyba v pixelech obrazovky, stejně jako zadaná tolerance. Z porovnání obou hodnot pak plyne, zda vertex zahrnout do zobrazovaného terénu či nikoliv.

Kvůli zachování kontinuity povrchu terénu se nestačí při konstrukci geometrie spoléhat jen na zobrazovací chybu jednotlivých vertexů, ta totiž sama o sobě nenese informaci o stavu okolních vertexů. Snadno by se mohlo stát, že mezi různými úrovněmi detailů není žádný přechod, vzniká tedy trhlina v modelu. Místo vyžadující vysokou úroveň detailů je třeba obklopit méně podrobnou geometrií (vytvořit přechod) a až poté se lze vrátit k původní nízké úrovni detailů. V reprezentaci terénu quad tree strukturou to znamená, že žádný vertex-potomek nesmí být zobrazen, aniž by nebyl v terénu zahrnut i jeho vertex-rodič. Tuto podmínku lze zajistit explicitně, avšak jeden vertex má dva rodiče a čtyři potomky (není-li kořenem nebo na okraji terénu) a neustálé propagování aktivního zobrazení v rámci stromu by bylo výpočetně nevýhodné.

SOAR tento problém efektně řeší vnořením zobrazovacích chyb dle hierarchického rozložení terénu ve stromové struktuře. Každému vertexu je jako chyba zobrazení přiřazeno maximum e_i z jeho vlastní chyby a chyb všech jeho potomků (rovnice 2.1). Vedle samotných zobrazovacích chyb je třeba uvážit vztah mezi polohou potomka a rodiče, to je vyřešeno zavedením ohraničujících vnořených sfér. Každá sféra má střed umístěn na souřadnicích vertexu $\boldsymbol{p}_i$, ke kterému náleží a poloměr r_i odpovídající vzdálenosti nejvzdálenějšího (i nepřímého) potomka. Sféra z vyší úrovně tedy obaluje i vertexy ze sféry na nižší úrovni, uspořádání ohraničujících sfér pro 2D případ je znázorněn na obr. 2.1. Zavedením těchto dvou parametrů se dosáhne uvedené podmínky pořadí zobrazení vertexů.



Obrázek 2.1: Vnořené sféry pro 2D případ, každý kruh obsahuje všechny potomky vertexu, k němuž náleží. Převzato z [7].

Oba parametry r_i i e_i jsou maxima pro daný vertex a nejsou proto závislé na poloze kamery, dají se tudíž s výhodou vypočítat předem. K tomu slouží přípravná fáze, parametry jsou vypočteny podle následujících vzorců převzatých z [7]. Aktuální vertex je označen indexem i z množiny všech vertexů V, jeho potomek má index j nabývající hodnot z množiny C_i všech potomků vertexu i. Maximum objektově prostorové chyby je zjištěno jako

$$e_i = \begin{cases} \hat{e}_i & \text{vertex } i \text{ je list,} \\ \max\{\hat{e}_i, \max\limits_{j \in C_i}\{e_j\}\} & \text{jinak.} \end{cases} \tag{2.1}$$

Kde $\hat{e}_i$ představuje vlastní objektově prostorovou chybu vertexu i

$$\hat{e}_i = \left\| \boldsymbol{p}_i - \frac{\boldsymbol{p}_{i0} + \boldsymbol{p}_{i0}}{2} \right\| . \tag{2.2}$$

Je to výškový rozdíl mezi vertexem i a hranou, která vznikne po jeho vypuštění mezi jeho dvěma sousedníma vertexy i_0 a i_1. Dále je třeba vypočítat poloměr ohraničující koule

$$r_i = \begin{cases} 0 & \text{vertex } i \text{ je list,} \\ \max\limits_{j \in C_i}\{\|\boldsymbol{p}_i - \boldsymbol{p}_j\| + r_j\} & \text{jinak.} \end{cases} \tag{2.3}$$

Za běhu programu je nutno zjistit pouze vzdálenost vertexu od kamery

$$d = \|\boldsymbol{e} - \boldsymbol{p}\| \tag{2.4}$$

kde $\boldsymbol{e}$ je poloha kamery a $\boldsymbol{p}$ souřadnice vertexu. Vertex s indexem i se v modelu použije právě tehdy, jestliže platí následující nerovnost:

$$(\nu e_i + r_i)^2 > d_i^2 \tag{2.5}$$

Kde ν zahrnuje nastavenou toleranci τ v pixelech a perspektivní projekci kamery λ na plochu okna aplikace. Porovnávané členy jsou v kvadrátu z důvodu úspory výpočetního času. Nezáleží zda jsou dvě porovnávané hodnoty představující vzdálenost umocněny na druhou či nikoliv.

$$\nu = \frac{\lambda}{\tau}, \ \lambda = \frac{w}{\varphi} \tag{2.6}$$

Perspektivní projekce je definovaná zorným úhlem kamery φ a šířkou okna w v pixelech, v němž je scéna zobrazena.

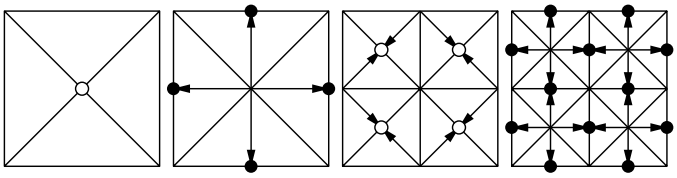
2.1.1.2 Příprava dat

SOAR pracuje s terénem v podobě pravidelné mřížky výškových dat, algoritmus vybere jen nutné vertexy a jejich pospojováním vytvoří síť trojúhelníků reprezentující terén ve scéně. Pro samotné zobrazení je třeba znát 3D souřadnice každého vertexu a k tomu dva parametry e a r vypočtené podle použité metriky. Příprava souboru vstupních dat probíhá v přípravné fázi, kde se k poli samotných výškových dat (např. v podobě černobílé bitmapy) přidají i polohové souřadnice x, y a vypočtené zmíněné parametry. Výstupem je binární soubor, kde je každý vertex definován pěti hodnotami, tento soubor poté slouží jako přímý vstup pro LOD algoritmus.

2.1.1.3 Funkce algoritmu

Jednou z možných metod změny detailů 3D modelu terénu je půlení nejdelší hrany (longest edge bisection). Čtvercový terén je vykreslen pomocí sítě pravoúhlých trojúhelníků,

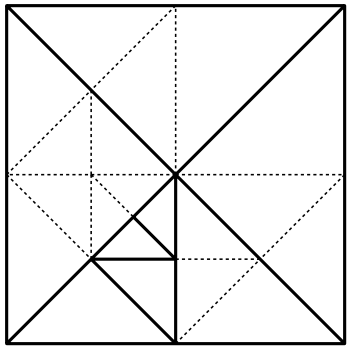
na nejnižší úrovni detailů to jsou pouze čtyři trojúhelníky se společným vrcholem ve středu terénu a jejich přepony tvoří okraj. Vertexy jsou do quad tree struktury namapovány tak, že středový vertex je kořen stromu terénu a jeho čtyři potomci jsou vertexy na středu přepon čtyř základních trojúhelníků. Pokud se tyto vertexy také zobrazí, je terén popsán již osmi trojúhelníky. Nové vertexy jsou na okraji terénu, mají proto pouze jednoho rodiče a dva potomky místo čtyř. Další generace potomků jsou opět vertexy umístěné na středech přepon nových trojúhelníků. Prvních pár kroků v quad tree hierarchii je uvedeno na obr. 2.2. Zanořením v různých místech stromu různě hluboko se postupně vygeneruje celý 3D model terénu. Příklad možné struktury sítě trojúhelníků je na obr. 2.3a.



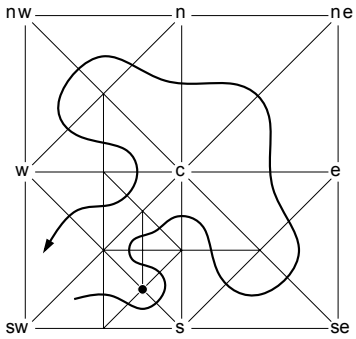
Obrázek 2.2: Quad tree topologie terénu. Podrobnější úroveň vzniká dělením přepon větších trojúhelníků. Převzato z [7].

Podstatou LOD algoritmu není jen určit, z kterých vertexů bude terén vytvořen, ale i jakým způsobem se z nich sestaví celkový 3D model. Možnou variantou je zobrazení objektu trojúhelník po trojúhelníku zasíláním grafické kartě vždy sady tří bodů o třech souřadnicích, tento postup však pro každý snímek produkuje značný objem dat nutný k dopravení do grafické karty. Mnohem lepší variantou je použití tzv. pásu trojúhelníků (triangle strip). Při využití tohoto způsobu zobrazení se počet vertexů na jeden trojúhelník v ideálním případě (dlouhý pás) blíží jedné. První trojúhelník je definován třemi vertexy a každý další je tvořen předchozí hranou a jediným novým vertexem, viz. obr. 2.4. Podle [8] je použití pásu trojúhelníků ideální jednak pro nízký objem dat, jednak je preferován i grafickým hardwarem. Výhodou pásu trojúhelníků je možnost přeskočit trojúhelník v posloupnosti vertexů a pás tak zatočit (ve smyslu mřížky trojúhelníků) požadovaným směrem, díky tomu odpadá nutnost použití více pásů pro jeden model, což je opět hardwarově přívětivější.

Pokud by byla na celý terén použita jen jedna úroveň detailů, stačilo by ho vykreslit jediným pásem trojúhelníků po řádcích sem a tam. Úroveň detailů se však v různých místech terénu liší a je třeba použít sofistikovanější metody sestavení posloupnosti vertexů pro pás trojúhelníků. Proces generování je nutno skloubit přímo s vyhodnocováním,



(a) Možná struktura terénu, tučně je vyznačeno maximální možné zjednodušení terénu. Tečkované hrany je nutno přidat kvůli zachování neporušeného povrchu modelu.



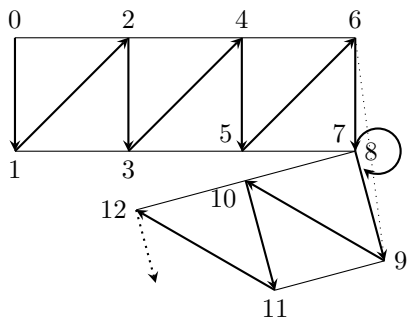
(b) Křivka znázorňuje postupný průchod terénem a vytváření pásu trojúhelníků. Tečka označuje jeden z vertexů, kde je nutno pás pootočit.

Obrázek 2.3: Nová struktura terénu a jeho vykreslení. Převzato z [7].

zda vertex zobrazit, či ne. Vyhodnocení probíhá rekurzivně, začíná úplným zanořením k listům stromu (nemá-li nižší úroveň šanci na zobrazení, hlouběji se nesestoupí) a při postupném vynořování se přidávají vertexy do pásu trojúhelníků. Průchod polem vertexů a generování pásu je naznačeno křivkou na obr. 2.3b, při bližším pohledu je patrné, že je nutno pás zatáčet podle aktuální hloubky v rekurzi. Po skončení algoritmu je získaná posloupnost dat předána grafické kartě k renderování.

Algoritmus zpracovává terén po trojúhelnících, je tedy postupně volán na čtyři základní trojúhelníky z nichž se skládá nejméně podrobná síť terénu. Každý trojúhelník je v quad tree struktuře reprezentován vertexem i (pravoúhlý vrchol). Pokud vertex-potomek j (na pozici středu přepony) splňuje toleranci pro zobrazení, dojde k rekurzivnímu volání algoritmu na nižší úroveň stromu. Velký trojúhelník se dělí na dva menší: oba s pravoúhlým vrcholem tvořeným vertexem j. Nyní se z vertexu j stává i' a jeho potomek j' bude případně umístěn na střed odvěsny původního trojúhelníka. Takto se postupuje, dokud j splňuje toleranci a není listem.

Celý proces vyhodnocení lze urychlit použitím view-frustum cullingu, tedy oříznutím pro kameru neviditelné části terénu. Doposud bylo zobrazením vertexu myšleno jeho použití v modelu terénu bez ohledu na to, která část modelu je zrovna vidět. Algoritmus tak strávil většinu času testováním vertexů, které kamera vůbec nezabírá. Největší počet trojúhelníků se vždy nachází v blízkém okolí kamery, ale ta z nich zobrazuje jen poměrně



Obrázek 2.4: Posloupnost vertexů v pásu trojúhelníků. Po přidání druhého vertexu je každý trojúhelník určen jediným vertexem. Opakováním dvou bodů za sebou dojde k resetování aktuální hrany a pootočení pásu.

úzký výsek (pokud nesměřuje přímo dolů). View culling vyhodnocuje viditelnost vertexů a neviděné části terénu z testování metrikou vyřadí již na vysoké úrovni stromu. Celkový počet testovaných vertexů se tak rapidně sníží, je ovšem nutno upravit dosavadní postup generovaní modelu: Pokud je view-frustum culling aktivní, je vertex j před sestoupením na nižší úroveň otestován na viditelnost v kameře, jestliže je vidět, pokračuje se jako dříve a zkoumání viditelnosti na nižších úrovních již není třeba. Pokud vidět ještě není, probíhá test viditelnosti i na dalších úrovních.

Jakmile je dokončen LOD proces nad posledním základním trojúhelníkem, je celý model terénu pro aktuální polohu kamery připraven na renderování v aktuálním snímku animace. V následujícím snímku celý proces začíná znovu. Výsledné chování algoritmu znázorňují obrázky A.2 až A.4.

2.1.2 Test výkonu

Dokončený algoritmus byl testován na zkušebních datech s rozměry 2049×2049 bodů. Testovací sekvence se lišily použitou počítačovou sestavou a hodnotou tolerance. Každá sekvence proběhla v rozlišení 1024×768 pixelů s i bez view-frustum cullingu. Měřily se počet zobrazených trojúhelníků a okamžitý počet snímků za sekundu při průletu nad terénem. Průlet se sestával z obkroužení terénu s kamerou směřující do jeho středu. Jedna sekvence (průlet) trvala 4096 snímků, v následujících tabulkách jsou uvedeny průměrné hodnoty z celé sekvence.

Tabulka 2.1: Použité počítačové sestavy

Název	Komponenty
ASUS (notebook)	CPU: Intel Core i7 2670QM GPU: NVIDIA GeForce GT 555M 2GB DDR3 RAM: 6GB DDR3 OS: Windows 7 64x
Acer (tablet)	CPU: AMD Dual-core C-60 GPU: AMD Radeon HD 6290 384MB RAM: 2GB DDR3 OS: Windows 7 86x
PC (stolní PC)	CPU: AMD Athlon 4200+ GPU: NVIDIA GeForce 8600 GT 512MB RAM: 2GB OS: Windows 7 64x

Testy proběhly na sestavách uvedených v tab. 2.1. Více použitých sestav umožňuje do jisté míry odhadnout chování SOARu i na jiném hardware, platí ovšem, že čím výkonnější sestava, tím rychlejší zobrazení.

Tabulka 2.2: Rychlost zobrazení, SOAR bez view culling

τ [px]	1	2	3	4	5
troj. [tis.]	196,11	67,72	34,73	21,28	14,42
ASUS	19,24	55,38	111,36	192,43	294,01
Acer	2,90	8,57	16,85	25,69	36,60
PC	9,37	27,19	54,02	88,00	129,28

Různé hodnoty tolerance τ umožňují sledovat schopnost SOARu redukovat počet trojúhelníků. Použitý terén se v základní podobě sestává z $8,39$ mil. trojúhelníků, tabulka 2.2 ukazuje, že i při toleranci chyby zobrazení jeden pixel je počet trojúhelníků snížen přibližně $40\times$ a s vyšším τ množství trojúhelníků ještě klesá.

Tabulka 2.3: Rychlost zobrazení, SOAR s view culling

τ [px]	1	2	3	4	5
troj. [tis.]	80,28	27,25	14,03	8,75	6,08
ASUS	44,79	135,26	270,42	415,49	532,02
Acer	6,87	20,32	35,34	53,22	72,73
PC	22,06	64,20	120,39	183,72	246,34

Z tabulky naměřených hodnot 2.2 vyplývá, že SOAR dosahuje uspokojivého FPS spíše při vyšších úrovních tolerance, kdy je třeba zobrazit (a propočítat) menší počet trojúhelníků. Vyhodnocení každého vertexu, zvlášť při snaze zobrazit několik set tisíc polygonů, je evidentně výpočetně příliš náročné. Aktivace view-frustum cullingu (data uvedená v tab. 2.3) umožňuje zobrazit i při nízkých hodnotách tolerance malý výřez (kamera směřuje pod horizont) terénu s vysokými detaily, ale jakmile kamera zabere větší část terénu, opět se dostaví problém přílišného počtu vertexů k vyhodnocení. View-frustum culling tedy do jisté míry zvýší výkonost SOARu, ale při zobrazení velkého množství trojúhelníků jeho vliv upadá.

2.2 Geometrical MipMapping

Algoritmus Geometrical MipMapping (GMM) sepsal v práci Fast Terrain Rendering Using Geometrical MipMapping Willem H. de Boer. Jak jsem předeslal v úvodu kapitoly, GMM mění úroveň detailu terénu po blocích, de Boer tuto metodu přirovnává k mipmappingu textur [6]. Princip je opravdu obdobný, stejně jako je pro texturu vygenerována sada mipmap vždy o poloviční velikosti než předchozí, je pro blok terénu vytvořena sada trojúhelníkové sítě vždy s poloviční hustotou vertexů. Grafická karta volí různé úrovně mipmap textur v závislosti na vzdálenosti modelu od kamery, stejně tak GMM vybírá pro blok terénu adekvátní úroveň detailů podle zobrazovacího kritéria. Kritérium je uživatelem zvolená tolerance deformace struktury terénu, ta je porovnána s hodnotou vypočtenou metrikou pro každý blok zvlášť.

Blokové rozdělení terénu má smysl jen pro terén reprezentovaný pravidelnou mřížkou výškových dat. Všechny bloky jsou uloženy v quad tree struktuře na pozici listů, nadřazené uzly stromu obsahují vždy čtyřnásobek bloků než nižší uzel, až kořen zahrnuje celý terén.

Stromová topologie je využita pro efektivní view-frustum culling [6]. Blok terénu je vykreslen, pouze pokud se alespoň malá část bloku nalézá v záběru kamery.

Nedílnou součástí LOD algoritmu je zabránění tvorbě defektů v povrchu modelu popsaných v sekci 1.1.2. Pro GMM to znamená odstranění švů mezi bloky s rozdílnou úrovní detailů.

2.2.1 Implementace

Zde je uveden způsob, jímž jsem GMM algoritmus implementoval. Vycházel jsem především z [6] v chování jádra algoritmu a použité metriky. Dále jsem použil pozměněnou metodu navázání bloků terénu a základní algoritmus jsem rozšířil o out-of-core funkci, jejíž možná realizace je v [6] pouze naznačena. Jelikož se GMM ukázal jako vhodný pro použití ve finální aplikaci, bylo nutné zahrnout do renderovacího procesu i texturu terénu.

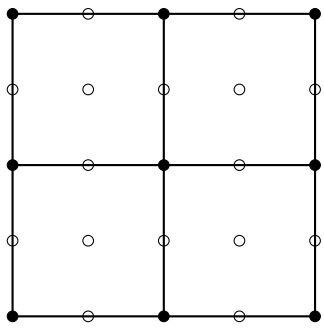
2.2.1.1 Volba úrovně detailu

Stejně jako u SOARu je LOD volen skrze metriku, která z objektově prostorové chyby bloku vypočte zobrazovací chybu v prostoru obrazovky. Objektově prostorová chyba se tentokrát nezjišťuje pro jeden vertex, ale pro celý blok terénu. Přesto je třeba uvážit zobrazovací chybu každého vertexu zvlášť a teprve podle maximální chyby ze všech vertexů se rozhodnout pro vhodný stupeň detailů. Volba maxima zaručí, že se vždy zobrazí terén se stupněm detailů vyšším nebo přesně takovým, jaký vyžaduje nastavená tolerance. Objektově prostorová chyba se nemění, proto lze blok terénu reprezentovat jejím nalezeným maximem po celou dobu běhu aplikace [6]. Dále je třeba zvolit bod v prostoru, od něhož se bude určovat vzdálenost ke kameře, přímo se k tomu nabízí střed terénu. Poloha středu $\boldsymbol{p}$ v rovině x, y je jednoduše geometrický střed terénu a souřadnice z je učena jako průměrná výška všech vertexů.

Přestože je bloků terénu podstatně méně než vertexů, počítat zobrazovací chybu v prostoru obrazovky a porovnávat ji s tolerancí pro každý snímek animace je výpočetně stále značně náročné. Nabízí se proto možnost vypočítat předem jakousi hraniční vzdálenost, od které by bylo možno použít méně detailní strukturu bloku. Zobrazovací chyba v prostoru obrazovky však závisí vedle vzdálenosti bloku terénu od kamery i a na jejím směru pohledu. To technicky znemožňuje vypočítat a uložit chybu pro všechny případy. Dá se ovšem předpokládat, že v aplikaci zobrazující terén směřuje kamera převážně do oblasti horizontu, nachází se tedy téměř ve vodorovné poloze. Kamera potom sleduje terén

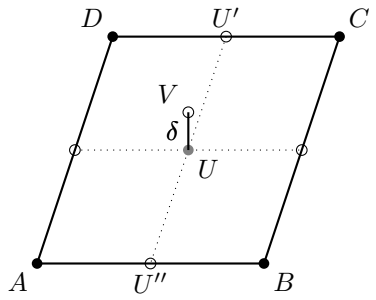
přesně z boku a chyba v prostoru obrazovky je tudíž maximální, takže tento předpoklad nezpůsobí za žádných okolností zobrazení méně detailního terénu než je třeba. Naopak při poloze kamery okolo vertikální pozice bude renderován zbytečně detailní terén. S pevně zvolenou polohou kamery ve vodorovné pozici je již možno pro nastavenou toleranci vypočítat minimální vzdálenost D_i, od které lze použít i-tý stupeň detailů [6].

Všechny bloky terénu se na základní úrovni skládají ze stejného počtu vertexů, ty jsou uspořádány do čtvercové mřížky. Na nejpodrobnější (nulté) úrovni má mřížka hranu o velikosti $2^n + 1$ vertexů. Vyšší úroveň vznikne z předchozí vynecháním každého druhého vertexu v lichých řádcích a sloupcích a vypuštěním celých sudých řádků a sloupců, jak ilustruje obr. 2.5. Mřížka úrovně m tak má hranu o délce $2^{(n-m)} + 1$ vertexů. Celkový počet stupňů detailů je $n + 1$. Objektově prostorová chyba bloku při přechodu z i-té úrovně na $i + 1$, je maximum objektově prostorových chyb ze všech vypuštěných vertexů ve stupni $i + 1$. Limitní vzdálenost, od níž je možno tento přechod uskutečnit, je počítána z maxima zobrazovací chyby úrovně $i + 1$.



Obrázek 2.5: Změna úrovně detailů bloku terénu. Blok na obrázku má v základní úrovni hranu s 5 body, vyšší úroveň vznikne vynecháním bílých vertexů. Případný další stupeň by tvořily jen rohové body.

Objektově prostorová chyba jednoho vertexu je výškový rozdíl mezi ním a povrchem modelu zkoumaného stupně detailů. V nulté úrovni je rozestup použitých vertexů v základní mřížce $h = 1$, ve vyšší úrovni je rozestup vždy dvojnásobný. Na i-tém stupni je pak $h = 2^i$. Vertex na bodu V umístěný v mřížce na řádku r a sloupci s, takových že se oba nerovnají násobku h, je aproximovaný polygonem s levým horním rohem na řádku $\frac{r}{h}$ a sloupci $\frac{s}{h}$. Tento polygon je v mřížce vertexů tvořen čtvercem o hraně h s rohovými vertexy v bodech A, B, C a D viz. obr. 2.6. Za předpokladu že vrcholy polygonu



Obrázek 2.6: Objektově prostorová chyba vertexu V je vzdálenost mezi ním a polygonem, jenž ho nahradí (bod U).

nesplývají, lze bod U ležící v polygonu na souřadnicích V_x, V_y vypočítat následovně:

$$\begin{aligned}
&U'_x = V_x, \quad U'_y = \frac{D_y - C_y}{D_x - C_x}(U'_x - C_x) + C_y, \quad U'_z = \frac{D_z - C_z}{D_x - C_x}(U'_x - C_x) + C_z \\
&U''_x = V_x, \quad U''_y = \frac{A_y - B_y}{A_x - B_x}(U''_x - B_x) + B_y, \quad U''_z = \frac{A_z - B_z}{A_x - B_x}(U''_x - B_x) + B_z \\
&U_x = V_x, \quad U_y = V_y, \quad U_z = \frac{U'_z - U''_z}{U'_y - U''_y}(U_y - U''_y) + U''_z
\end{aligned} \tag{2.7}$$

Kde U' je bod ležící na hraně CD a U'' náleží protilehlé hraně AB aproximačního polygonu. Je-li $D_x - C_x$ či $A_x - B_x$ rovno nule, je třeba ve všech rovnicích 2.7 vzájemně prohodit indexy x a y. Objektově prostorová chyba δ_v vertexu j na souřadnicích V je potom:

$$\delta_{vj} = |V_z - U_z| \tag{2.8}$$

Objektově prostorová chyba δ bloku na stupni i je

$$\delta_i = \max_{j \in M_i}\{\delta_{vj}\}, \quad i \in \{1,\ 2,\ 3, ...n+1\} \tag{2.9}$$

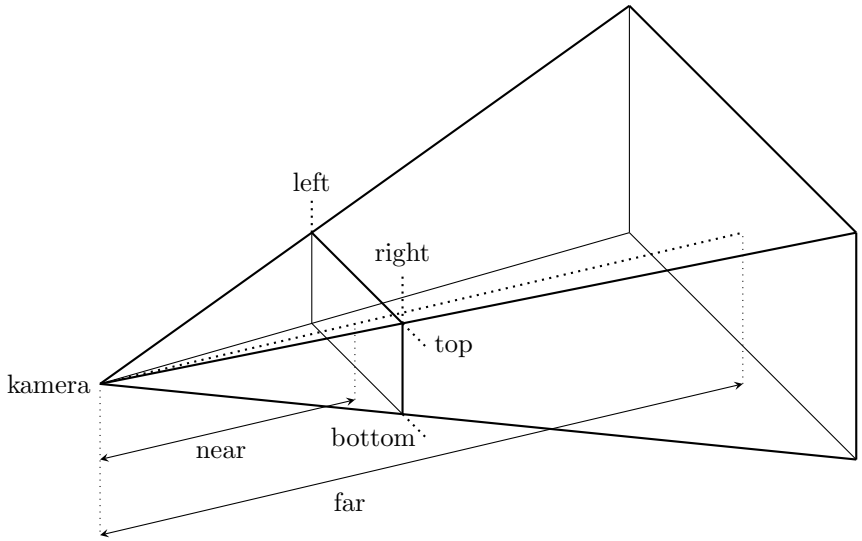
Pro $i = 0$ platí $\delta_i = 0$, j nabývá hodnot z množiny M_i všech vynechaných vertexů na stupni i. Minimální vzdálenost bloku terénu od kamery D_i, od které je povoleno použít model terénu se stupněm detailů i je

$$D_i = \delta_i C \tag{2.10}$$

Kde C je konstanta zahrnující nastavenou toleranci τ (v pixelech) a parametry perspektivní projekce kamery.

$$C = \frac{nh}{2\tau\,|t|} \tag{2.11}$$

Perspektivní projekce je definovaná parametry left, right, top, bottom, near a far určující view frustum kamery v jednotkách 3D souřadnic aplikace. Význam parametrů je patrný z obr. 2.7. V rovnici 2.11 n představuje near, t top a h výšku okna aplikace v pixelech.



Obrázek 2.7: View frustum kamery definuje výřez prostoru, jenž se zobrazí v okně aplikace. Výřez ohraničuje šest rovin, jejichž polohu určují parametry left, right, top, bottom, near a far.

Pomocí vzorce 2.10 se hned po načtení dat terénu z disku každému bloku vytvoří tabulka minimálních vzdáleností s n hodnotami. V průběhu renderovaní aplikace se pro každý snímek již jen vypočte vzdálenost d mezi kamerou a blokem terénu a porovná se s připravenou tabulkou. Největší hodnota, která splňuje nerovnost 2.12 určuje úroveň detailů, jež se v aktuálním snímku pro daný blok terénu použije.

$$d > D_i \tag{2.12}$$

Vzdálenost d je vypočtena podle:

$$d = \|\boldsymbol{e} - \boldsymbol{p}\| \tag{2.13}$$

Kde $\boldsymbol{e}$ je poloha kamery a $\boldsymbol{p}$ střed bloku terénu.

2.2.1.2 Jádro algoritmu

Celý proces GMM LOD algoritmu lze shrnout do čtyř bodů:

- načtení terénu, rozdělení do bloků, výpočet limitních vzdáleností

- nalezení viditelných bloků
- přiřazení nové úrovně detailů viditelným blokům
- vygenerování modelu z viditelných bloků, renderování

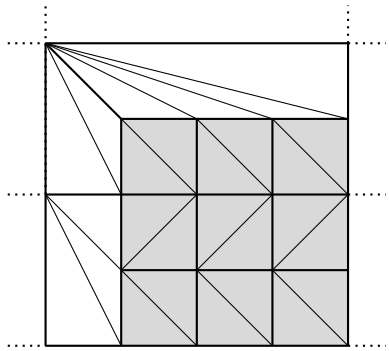
Data terénu jsou načtena v podobě mřížky vertexů o rozměrech $(2^k+1)\times(2^l+1)$, přičemž l a k jsou celočíselné násobky n viz. 2.2.1.1. Terén lze tedy beze zbytku rozdělit do pole bloků s rozměry k, l každý s hranou 2^n+1 vertexů a $n+1$ úrovněmi detailů. Sousední bloky sdílejí vertexy na okrajích bloků, čímž je zaručena přesná návaznost povrchu vykreslených bloků na stejných úrovních. Řešení švů mezi terény s rozdílnými stupni je uvedeno níže. Po načtení terénu jsou každému bloku vypočteny jeho limitní vzdálenosti pro různé stupně detailů.

Bloky terénu jsou namapovány do quad tree struktury. Kořen zahrnuje celý terén (všechny bloky) a každý z jeho čtyř potomků dědí jednu čtvrtinu terénu. Potomek vždy reprezentuje čtvrtinu terénu svého rodiče. Terén je takto dělen a přidáván do uzlů, dokud nezbudou samotné bloky, ty se uloží do listů stromu. Pokud je k rovno l, pak je strom pravidelný. Stromová reprezentace dat umožňuje rychlé nalezení viditelných bloků. Je-li alespoň část terénu náležící uzlu u viditelná, jsou na viditelnost otestováni i jeho potomci. Není-li ani kousek u zabrán kamerou, pak nemá cenu jeho potomky testovat. Takto se prochází celý strom od kořene až k listům a všechny viditelné listy jsou označeny.

Dále má smysl zabývat se pouze viditelnými bloky terénu, každému z nich je třeba určit správný stupeň detailů. Vypočte se vzdálenost bloků od kamery (vztah 2.13) a porovná se s předem připravenou tabulkou podle 2.10. Tím je všem viditelným blokům určen LOD.

Nastavený stupeň detailů určuje vzorkovací krok, s kterým se z původní mřížky (0. stupeň) vybírají vertexy do modelu dané úrovně. Ze stejných důvodů jako v SOARu (2.1.1.3) je celý terén renderován z jednoho pásu trojúhelníků. Mezi jednotlivými bloky se přeskakuje tzn. degenerativními trojúhelníky (opakování posledního vertexu jednoho bloku a prvního vertexu v druhém bloku), který grafická karta nevykreslí a lze tak najednou zobrazit dva separátní modely. Bez ohledu na sousední bloky obsahuje každý blok obdélníkové jádro, jehož geometrie se z mřížky vertexů generuje do pásu trojúhelníků po řádcích sem a tam. Možná podoba jádra je obr. 2.8.

Sousedí-li blok b s blokem d na vyšším stupni detailů, má b na jejich společné hraně více vertexů než d. Mezi bloky tudíž vznikají cracky a t-junctions. Napravení chyb v povrchu je docíleno vynecháním vertexů, které má b na společné hraně oproti d navíc,



Obrázek 2.8: Triangulace bloku terénu. Blok na obrázku sousedí se dvěma bloky s nižší úrovní detailů, přechod mezi nimi zajišťuje vějířovitá struktura trojúhelníků na místě vynechaného řádku a sloupce. Jádro bloku je zvýrazněno šedivou barvou.

jak je naznačeno na obr. 2.8. Rozdíl v počtu vertexů hrany se tak přesune do bloku b a zde se vyrovná speciálním vykreslením narušeného řádku či sloupce. Vertexy, které jsou v podrobnější hraně navíc, utvoří trojúhelníky spojením s nejbližším předchozím vertexem méně podrobné hrany. Vznikají tak vějířům podobné obrazce (obr. 2.8), jež se dají poměrně jednoduše renderovat pomocí vějíře trojúhelníků (triangle fan). Já jsem vhodnou volbou pořadí vertexů dokázal i tyto obrazce popsat pásem trojúhelníku a díky tomu je celý terén možno renderovat jediným voláním grafické karty. Kombinace geometrie jádra a případných švů dává dohromady model bloku přesně navazující na všechny sousední bloky. Příklad terénu generovaného GMM je na obr. A.5.

2.2.1.3 Out-of-core

Schopnost algoritmu zobrazit efektně terén z dat, jež jsou větší než fyzická paměť počítače, se nazývá out-of-core. Jednou z možností jak toho dosáhnout je operovat pouze s výřezem z celkového terénu, zhruba odpovídajícím velikosti paměti.

Podpory out-of-core je u GMM dosaženo rozdělením celkového terénu (rozsahu např. ČR) na mnoho menších, na sebe navazujících segmentů. Doposavad byl princip GMM algoritmu popisován na jednom (segmentu) terénu. Nyní je segmentů více, ale každý je opět rozdělen na bloky a dále zpracován GMM algoritmem podle uvedeného postupu. LOD je pro segmenty řešen nezávisle, ale stejně jako jsou provázané bloky v rámci jednoho segmentu, je třeba navázat i sousední bloky na rozhraní dvou segmentů.

Načítání dílčích terénů z disku je dáno polohou kamery. Jakmile se kamera přiblíží na určitou vzdálenost ke středu terénu, jenž ještě nebyl načten, dojde k jeho nahrání do paměti a následnému zobrazení. Podobně, pokud se kamera od některého již načteného terénu příliš vzdálí, je tento z paměti smazán. Minimální vzdálenost pro načtení segmentu je vhodné volit jako celistvý násobek šířky terénu, to ovšem závisí na velikosti fyzické paměti. Maximální vzdálenost pro ponechání segmentu v paměti je nutno nastavit větší než její protiklad, jinak by se segmenty mazaly ihned po načtení. Může být větší např. o polovinu šířky segmentu. Vzniká tak cache několika málo částí terénů a v případě, že by kamera prudce změnila směr, není nutno znovu načítat zrovna smazaná data.

Proces nahrávání segmentu z disku může trvat relativně dlouhou dobu a působit tak krátkodobé „zamrzání“ aplikace. Tomuto problému se dá do jisté míry vyhnout vytvořením zvláštního vlákna, jež bude načítat a mazat segmenty nezávisle. Vlákno cyklicky kontroluje již načtené a pokouší se načíst nové segmenty terénu. Samotné načítání tak probíhá na pozadí a v ideálním případě není uživatelem vůbec pozorováno.

2.2.2 Test výkonu

Hotová implementace GMM algoritmu byla podrobena stejným testům jako SOAR, jejich detailní srovnání je uvedeno v následující sekci, tato část je zaměřena na samotný GMM. Zajímavým parametrem GMM je velikost bloku terénu. Malé bloky umožňují přesnější selekci částí terénu vyžadující vyšší detaily, ale na druhou stranu klesá počet stupňů detailů, takže terén nelze tolik zjednodušit. Menší bloky také znamenají více bloků a více zabrané paměti. Proto byla nejprve provedena série testů s velikostmi bloků v rozsahu 17 až 129 bodů.

Tabulka 2.4: Test velikostí bloků, tis. troj. (ASUS)

	τ [px]				
blok [body]	1	2	3	4	5
17	627,94	229,57	128,77	89,54	69,72
33	863,23	311,41	162,98	102,04	70,38
65	1164,90	414,07	217,73	137,83	92,27
129	1568,70	565,72	293,46	186,74	125,91

Tabulka 2.5: Test velikostí bloků, FPS (ASUS)

	τ [px]				
blok [body]	1	2	3	4	5
17	65,09	93,49	104,04	109,13	112,92
33	82,59	181,10	263,04	342,66	371,44
65	77,54	201,37	352,15	481,02	588,40
129	72,03	194,29	340,43	475,47	586,95

Tabulky 2.5 a 2.4 ukazují rychlost vykreslování (FPS) a průměrný počet trojúhelníků při průletu nad terénem bez view cullingu. Použitá sestava ani aktivace view cullingu na výsledné porovnání neměly vliv. Navzdory menšímu počtu trojúhelníků je vykreslení s bloky po 17 bodech pomalejší než při 33 a to je zas pomalejší než s bloky po 65 bodech. Zlom nastává mezi 129 a 65, kdy je počet trojúhelníků zřejmě již příliš vysoký. Zajímavé je, že při $\tau = 1\,\text{px}$ jsou nejrychlejší bloky s 33 body, rozdíl oproti 65 však není nijak zásadní a patrně je to způsobeno optimálnější strukturou modelu pro daný průlet nad terénem.

Rozpor v pomalejším zobrazení menšího počtu trojúhelníků je způsoben delším časem stráveným při vytváření modelu terénu. Je totiž třeba vyhodnotit 4× více bloků oproti předchozí velikosti a mezi bloky vzniká také více kratších švů, které je nutno speciálně ošetřit. S menšími bloky GMM ztrácí výhodu ve zpracování bloků a přibližuje se per vertex metodě.

Dále testy probíhaly obdobně jako u SOARu, byly použity stejné sestavy (2.1) a parametry tolerance, ve všech následujících testech měly bloky velikost 65 bodů.

Tabulka 2.6: Rychlost zobrazení, GMM bez view culling

τ [px]	1	2	3	4	5
troj. [tis.]	1164,90	414,07	217,73	137,83	92,27
ASUS	77,54	201,37	352,15	481,02	588,40
Acer	12,82	31,13	50,57	73,44	93,78
PC	28,14	70,78	115,50	184,78	238,44

Z tabulek 2.6 a 2.7 je jasná závislost počtu renderovaných trojúhelníků na použité toleranci, při jednotkovém růstu τ, klesá množství trojúhelníků přibližně na polovinu předchozí hodnoty. Počet zobrazených snímků naopak stoupá, ne však tak prudce.

Tabulka 2.7: Rychlost zobrazení, GMM s view culling

τ [px]	1	2	3	4	5
troj. [tis.]	559,01	215,02	121,80	81,90	57,02
ASUS	149,12	349,18	519,24	619,93	696,89
Acer	23,84	49,93	81,65	102,69	124,47
PC	54,25	130,07	178,54	263,71	316,42

View culling opět ukázal svou schopnost oříznout kamerou neviděné trojúhelníky a zrychlit tak zobrazení, úspora je přibližně poloviční. Při toleranci nastavené na $\tau = 1$ dosahuje počet trojúhelníků ve scéně jednoho milionu (aktivní view culling) a FPS poněkud klesá. Grafické karty by přitom měly být schopny rychlejšího zobrazení, ukázalo se, že při takovém množství dat, je časově náročné sestavit v každém snímku odpovídající pás trojúhelníků. Jistého zrychlení jsem dosáhl uložením stavu každého bloku, takže pokud se kamera zastaví, GMM přestane generovat nové modely terénu a pouze se kopírují dříve vygenerované sekvence vertexů.

Zběžný pohled na výsledky SOARu (tab. 2.3) stačí k tomu, aby bylo jasné, že GMM zvládá zobrazení terénu mnohem lépe (viz. 2.3). GMM byl proto zvolen jako LOD algoritmus pro finální aplikaci a byl rozšířen o vykreslování textur. Nová verze byla opět podrobena testům.

Tabulka 2.8: Rychlost zobrazení, GMM s texturou

τ [px]	1	2	3	4	5
troj. [tis.]	559,01	215,02	121,80	81,90	57,02
ASUS	97,77	244,99	386,49	469,85	524,81
Acer	15,01	34,48	60,32	80,29	99,57
PC	32,09	87,77	148,88	197,71	246,00

Tabulka 2.8 obsahuje naměřené hodnoty ze zobrazení stejného terénu jako dříve, nyní s texturou 4096×4096 pixelů. Na první pohled je patrné jisté zpomalení, to je způsobeno

vedle samotného vykreslení textury grafickou kartou i nárůstem generovaných dat o dvě třetiny. Ke třem prostorovým souřadnicím přibyly každému vertexu dvě souřadnice pozice v textuře. Podoba testovaných dat s texturou je na obr. A.1.

2.3 Porovnání algoritmů

V předchozí části práce jsem uvedl implementace dvou level of detail algoritmů – SOAR a GMM. V této části se budu zabývat jejich srovnáním a objasním, co vedlo k výběru GMM do finální aplikace.

V úvodu jsem popsal důvody a výhody použití LOD algoritmů. Nyní, před jejich vyhodnocením, je vhodné se nejprve podívat na zobrazení terénu bez LOD. Každá grafická karta je schopna vykreslit jen určitý počet trojúhelníků za sekundu. LOD algoritmus představuje zátěž navíc, proto ho nemá smysl použít na terén, jenž se ve své základní formě do limitu karty vejde. Ovšem taková aplikace by byla buď omezena pro úzký výběr velmi výkonných karet, nebo by mohla zobrazit jen malou oblast terénu. Změnou tolerance LOD algoritmů lze adaptivně nastavit počet vykreslovaných trojúhelníků, a tak aplikaci zrychlit i na slabším hardwaru.

Tabulka 2.9: Rychlost zobrazení bez LOD algoritmu

	Asus	Acer	PC
FPS	29,07	3,71	2,26

Tabulka 2.9 ukazuje zobrazení čtvercového terénu o hraně 2049 vertexů bez pomoci LOD algoritmu na dříve zmíněných sestavách. Test proběhl za stejných podmínek jako v předchozích sekcích (stejný terén a průlet). Z porovnání s tabulkou 2.7 i 2.3 je evidentní ohromný nárůst výkonu při použití LOD algoritmu, přesto má jejich nasazení i své nevýhody.

Zatímco GMM aktivně mění úroveň detailů, značně tím snižuje počet renderovaných trojúhelníků a v závislosti na něm se mění i FPS, v případě bez LOD je množství trojúhelníků konstantní a FPS víceméně taktéž. GMM z porovnání vychází lépe, v průběhu celého testu dosahuje vyššího FPS, ale počet vykreslených trojúhelníků za sekunduje je přibližně 3× nižší. Hlavním důvodem tohoto zpomalení je nutnost neustále kopírovat nová data do poměti grafické karty, zatímco při zobrazení bez LOD stačí celý terén nahrát jen

jednou a následně ho jen renderovat. Další snížení rychlosti způsobuje LOD proces samotný – nalezení správné úrovně detailů, vyhodnocení viditelnosti atd. Zpomalení se však bohatě vynahradí mnohem menším počtem trojúhelníků k vykreslení, význam LOD algoritmů při zobrazení rozsáhlého terénu je tedy jasně patrný.

Následující porovnání obou algoritmů vychází z testů v předchozích sekcích 2.1.2 a 2.2.2. Testy byly provedeny na algoritmech v jejich základní podobě, tedy při zobrazení jednoho bloku terénu bez textur a dalších grafických vylepšení. Algoritmy běžely na stejných sestavách se shodnými sety dat.

Zásadním srovnávacím kritériem je rychlost zobrazení, tedy FPS a počet trojúhelníků za sekundu. Jak je vidět při pohledu do tabulek 2.3 a 2.7, GMM je ve všech ohledech rychlejší než SOAR. Další měřenou veličinou je počet trojúhelníku, ten by měl být naopak co nejnižší, aby se přiblížil optimálnímu množství doopravdy potřebných polygonů k vykreslení aktuální scény. Toto množství se nedá zcela přesně určit, jelikož výpočet zobrazovací chyby vertexů závisí na triangulaci méně detailního (neoptimálního) terénu. Je však jasné, že optimální množství bude zpravidla nižší než zrovna použité, protože se musí ošetřit přechody mezi různými stupni detailů. Ze základní charakteristiky SOARu a GMM je patrné, že GMM bude trojúhelníků potřebovat podstatně více než SOAR, jelikož SOAR je schopen upravit strukturu terénu již na úrovni vertexů. V tomto ohledu je dělení terénu na bloky nevýhodou. I v přídě, kdy v celém bloku převyšuje toleranci jen jediný vertex, musí být detailně vykreslen celý blok. Oproti tomu SOAR podrobně zobrazí jen jeho okolí a pokud možno se vrátí k nižší úrovni detailů.

GMM tedy zobrazuje více trojúhelníků než je třeba, mnoho jich je nepatrně malých nebo splývají s ostatními do roviny, ale je to jeho výhodou, neboť neztrácí zbytečně čas zkoumáním každého jednotlivého vertexu jako SOAR. Důležité je podotknout, že tento postup bude fungovat jen s patřičně výkonnou grafickou kartou. Pokud bude množství trojúhelníků stále příliš vysoké, je třeba zvýšit toleranci a to může vést ke snížení vizuální kvality terénu.

Oba algoritmy potřebují ke své práci data terénu a navíc si vytvářejí set parametrů umožňující zrychlení LOD procesu. Zatímco u GMM dodatečný objem dat závisí na velikosti bloků (na každý blok šířky $2^n + 1$ bodů to je n hodnot), SOAR vyžaduje dva parametry pro každý vertex. Navíc je výpočet parametrů pro SOAR relativně zdlouhavý, je proto nutno data připravit předem a uložit na disk.

Gemetrical MipMapping algoritmus potřebuje k zobrazení terénu méně paměti, rychleji vygeneruje data nového modelu a i když používá více trojúhelníků než je nezbytně

nutné, ve výsledku je vždy rychlejší než SOAR. Z uvedených důvodů jsem proto GMM level of detail algoritmus vybral pro finální aplikaci.

Kapitola 3

Aplikace

V předchozí části byl vybrán Geometrical MipMapping jako vhodný LOD algoritmus pro finální aplikaci. Dalším úkolem bylo tuto aplikaci, jež bude pomocí GMM efektivně zobrazovat terén, vytvořit. Aplikace má posloužit při vývoji glass-cockpitu pro ultralehká letadla, jednou z jeho funkcí má být vizuální navigace prostřednictvím 3D terénu.

Kompletní aplikace nezobrazuje jen terén, ale i textury a grafické letové přístroje. V porovnání s testovanými verzemi obou algoritmů je vykreslení o něco náročnější, v závěru kapitoly je proto uveden test celé aplikace.

Veškeré zdrojové kódy, spustitelný Java archiv a dokumentaci aplikace je možno nalézt na CD příloze práce.

3.1 Funkce

Základní funkcí aplikace je vykreslování 3D terénu. Struktura 3D modelu je generována z výškových dat a pro zvýšení vizuálního dojmu je potažena texturou sestavenou z leteckých snímků krajiny.

Zobrazení terénu by se bez LOD algoritmů neobešlo, vedle rozšířeného GMM obsahuje aplikace i testovací verze SOARu a GMM. Mód aplikace lze vybrat v menu po spuštění. Úvodní okno obsahuje tři karty: Zobrazení terénu, Test LOD a Příprava dat, každá koresponduje s následujícími oddíly textu.

3.1.1 Zobrazení terénu

Princip GMM algoritmu byl již popsán, nyní se naplno využije jeho out-of-core funkce. Nejprve je třeba připravit vstupní data jednotlivých segmentů terénu, viz. 3.1.3. Za běhu programu se budou načítat malé části terénu a zobrazí se pomocí LOD algoritmu.

Aplikaci je možno spustit ve dvou režimech: buď samostatně a nebo ve spojení s FlightGear Flight Simulator. Oba režimy umožňují volné pohybování kamerou, režim se simulátorem pak v závislosti na zasílaných datech zobrazuje zjednodušený model letadla a různé letové přístroje. Ty jsou dostupné i v samostatném modu, ale udávají hodnoty jen pro kameru. Ukázka spolupráce aplikace a simulátoru je na obr. A.6, kde je vidět umělý horizont znázorňující náklon letadla a kompas, další hodnoty jako poloha, rychlost a výška jsou udány v číselné podobě.

3.1.2 Test LOD

Tato část aplikace umožňuje zobrazit terén LOD algoritmy ve formě v jaké byly testovány. Na CD je k tomu účelu přiloženo několik obrázků jako zdrojových dat. Obrázky by měly být černobílé, ideálně se 16 bity na pixel. Dále je očekáván čtvercový formát o hraně $2^n + 1$ pixelů, případně dojde k oříznutí obrázku na nejbližší nižší rozměr v této podobě. GMM je schopen číst rovnou obrázky, pro SOAR je třeba nejprve vygenerovat vstupní data.

3.1.3 Příprava terénu

Renderovaný terén se skládá ze dvou částí: textur a modelu. Geometrie modelu je síť trojúhelníků, vzniklá spojením vertexů, jejichž pozice je definovaná výškovými daty. V testovacím modu jsou data čerpána z černobílých obrázků, hlavní část aplikace pak používá data ze SRTM projektu. Textury v testovacím modu není možno zobrazit, na SRTM terén je možno aplikovat texturu sestavenou z fotomapy získané ze serveru Mapy.cz.

SRTM (The Shuttle Radar Topography Mission) projekt vznikl za účelem vytvořit globální výškový model Země. Raketoplánem byla pořízena data povrchu mezi 54° j.š. a 60° s.š. souřadnicového systému WGS 84, data jsou dostupná v blocích 1×1 stupeň. Oblast Severní Ameriky je k dispozici v rozlišení jedné vteřiny, zbytek světa pak ve třech vteřinách [2]. Dle licenčních podmínek společnost Seznam.cz (Mapy.cz) je možno volně

přístupné mapy použít pro osobní potřeby [9], k jejich získání je však nutný externí downloader.

Mapy na serveru Mapy.cz využívají UTM projekci, aby se textura nezobrazila zkresleně, je nutno SRTM data také převést do UTM souřadnic. Konverzi lze uskutečnit pomocí vzorců popsaných např. v [10], ve výsledku vykreslený terén nebude složen ze čtverců (dva trojúhelníky), ale z obdélníků, které se navíc směrem k severu zužují a protahují. Důsledkem toho je celý segment terénu obdélník s poněkud zaoblenými hranami a textura tudíž musí místy oblast terénu přesahovat, aby pokryla i toto zaoblení.

O konverzi souřadnic a generování dat se stará aplikace sama, pouze je nutno zadat umístění obou datových podkladů a velikost jednoho segmentu terénu v bodech. Výstupem je mřížka začínající v nejjihozápadnějším rohu všech nalezených SRTM souborů a pokrývající celou oblast až do nejseverovýchodnějšího rohu, včetně případného přesahu. Vedle dvou souborů pro každý segment terénu je vygenerován i textový soubor identifikující právě vytvořenou mapu a sloužící jako vstup pro hlavní aplikaci.

3.2 Test výkonu

Testy tentokrát proběhly na celých sadách segmentů terénů místo jednoho staticky načteného před začátkem měření. Jednotlivé segmenty se do paměti načítaly podle pozice kamery. Sady se lišily velikostí hrany segmentu a rozlišením použité textury. Nastavení GMM bylo ponecháno na šířce bloku 65 bodů, $\tau = 2$ a zapnutý view culling. Vedle standardně měřeného počtu trojúhelníků a FPS, přibyla doba načtení segmentů: čas načítání textury do RAM t_1, její zkopírování do paměti grafické karty t_2 a čas načítání souřadnic vertexů terénu t_3. Poslední novou veličinou je průměrný počet načtených segmentů. Testy byly uskutečněny na stejných sestavách jako v 2.1.2.

Tabulka 3.1: Velikosti segmentů terénu a textur

	terén		textura			
	[body]	[MB]	[px]	[MB]	[m/px]	dohled
A	257	0,77	1940 × 2980	16,60	16	2
B	257	0,77	490 × 750	1,00	32	5
C	513	3,08	980 × 1490	4,10	32	2

Celkem proběhly tři testy nad terénem s jihozápadním rohem na 49° s.š. a 14° v.d. a severovýchodním rohem na 51° s.š. a 16° v.d., tedy terén s hranou 2401 bodů a celkově zabírající 75 MB. Tabulka 3.1 popisuje parametry testů A, B a C. Uvedené rozměry a velikost v MB je přibližná, jelikož každý segment má kvůli zakřivení trochu jiné rozměry. Poslední sloupec udává vzdálenost dohledu v počtu segmentů.

Tabulka 3.2: Zobrazení mapy A

	tis. troj.	FPS	t_1 [ms]	t_2 [ms]	t_3 [ms]	segmenty
ASUS	71,25	498,50	459,59	21,59	27,21	7,72
Acer	71,34	86,84	2774,60	149,96	203,62	7,66
PC	71,50	182,48	642,95	42,22	90,03	7,77

Tabulka 3.3: Zobrazení mapy B

	tis. troj.	FPS	t_1 [ms]	t_2 [ms]	t_3 [ms]	segmenty
ASUS	76,61	477,91	34,54	1,56	30,09	28,20
Acer	76,61	80,72	186,84	14,35	244,26	28,02
PC	76,61	166,25	47,20	3,42	86,59	28,24

Tabulka 3.4: Zobrazení mapy C

	tis. troj.	FPS	t_1 [ms]	t_2 [ms]	t_3 [ms]	segmenty
ASUS	76,31	510,90	138,00	5,54	119,79	6,02
Acer	76,33	92,47	765,71	39,54	852,17	6,03
PC	76,34	181,72	182,54	10,79	347,71	6,05

Zvětšení textury či dat terénu přibližně odpovídá prodloužení času načítání, počet trojúhelníků a FPS je pak ovlivněno především dohledností a tedy aktuálním počtem načtených segmentů. Zatímco čtení souřadnic vertexů může plně běžet na pozadí v jiném vláknu, texturu je takto možno načíst pouze do paměti RAM. Do paměti grafické karty je texturu nutno nahrát mezi dvěma následnými snímky, čas t_2 tak představuje dobu „zamrznutí“ aplikace.

S vysokým rozlišením textur stoupá i paměťová náročnost, objem dat terénu v případě A je téměř zanedbatelný oproti textuře a přestože je průměrné FPS poměrně vysoké, načítání segmentů způsobuje nepříjemné trhání obrazu. Řešením je snížit čas t_2, čehož lze dosáhnout menšími segmenty terénu (častější kratší načítání) nebo snížením rozlišení textur. Nižší rozlišení umožňuje použití větších segmentů, což podobně jako v případě samotných bloků terénu vede k vyšší rychlosti zobrazení.

Při pevně zvoleném rozlišení textur je třeba volit velikost segmentů a dohlednost především podle použitého hardwaru. Z testů vyplývá, že výsledná aplikace je schopna efektně zobrazit terén do vzdálenosti větší než 90 km s texturou 32 px/m.

Kapitola 4

Závěr

Náplní práce bylo prozkoumat algoritmická řešení zobrazení rozlehlých terénů, vybrat vhodné algoritmy, implementovat je a porovnat. Následně ten, který se ukáže jako nejschopnější rozšít na finální aplikaci. Tedy na program, zobrazující rozsáhlý 3D terén včetně textur a grafických letových přístrojů.

LOD algoritmů je celá řada. Každý se snaží vhodnou úpravou geometrie modelu rychle zobrazit i velmi rozsáhlý terén se zachováním vizuální kvality originálu. Použité metody se přitom značně liší. Bez předchozí implementace je možno algoritmy porovnat jen na základě výsledků uveřejněných v dokumentech spolu s nimi. Takto jsem zvolil SOAR (Statle-less, one-pass adaptive refinement) a GMM (Geometrical MipMapping) jako vhodné algoritmy na implementaci.

Funkční verze obou algoritmů jsem objektivně porovnal na stejných datech a počítačových sestavách (tab. 2.1). Již z letmého nahlédnutí do tab. 2.7 (GMM) a 2.3 (SOAR), je jasně patrná rychlostní převaha GMM nad SOARem. Projevila se totiž výhoda blokového přístupu oproti per vertex metodě. Dále se nabízí možnost srovnání se situací bez LOD algoritmu v tab. 2.9.

GMM je schopen plynule zobrazit i terén o hraně 4097 vertexů, což při použití SRTM výškových dat znamená rozhled zhruba 130 až 200 km. V takové vzdálenosti je úroveň zobrazených detailů minimální a celkový počet trojúhelníků roste jen mírně, výsledná dohlednost je tedy fakticky omezena velikostí fyzické paměti počítače.

Dalším krokem bylo rozšířit vybraný LOD algoritmus o schopnost zobrazit SRTM data společně s texturami z leteckých snímků. Za tímto účelem byla vytvořena aplikace připravující terén k přímému zobrazení. Příprava spočívá v rozdělování výškových dat na segmenty zadané velikosti, to samé je provedeno s texturou. Segmentace celého terénu na menší díly umožňuje vykreslovat a mít v paměti pouze potřebné části krajiny. Vzniká

tak prostor např. pro aplikaci textury ve vysokém rozlišení, nebo renderování přídavných grafických objektů.

Vlastní aplikaci zobrazující terén byla přidána možnost komunikace s FlightGear Flight Simulator. Komunikace je prozatím jednostranná, simulátor pouze zasílá údaje o stavu letounu a aplikace podle nich vykresluje zjednodušený model letadla, nebo jen vybrané letové přístroje. Lze tak bezpečně otestovat chování aplikace v různých situacích.

V průběhu vypracovávání práce se podařilo splnit veškeré vytyčené cíle. Algoritmy pracují podle očekávání a skutečně zrychlují zobrazení terénu. Výsledný dojem z 3D krajiny je velice efektní.

Literatura

[1] Luebke, D., Reddy, M., Cohen, J. D., Varshney, A., Watson, B. a Huebner, R., *Level of Detail for 3D Graphics*. San Francisco: Morgan Kaufmann Publishers, 2003, isbn: 1-55860-838-9.

[2] *SRTM Topography*, ⟨http://www2.jpl.nasa.gov/srtm/⟩.

[3] Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L. F. a Faust, N. and Turner, G. A., *Real-Time, Continuous Level of Detail Rendering of Height Fields*. Georgia Institute of Technology, SAIC, 1996.

[4] Duchaineau, M., Wolinsky, M., Sigeti, D. E., Miller, M. C., Aldrich, C. a Mineev-Weinstein, M. B., *ROAMing Terrain, Real-time Optimally Adapting Meshes*. Los Alamos National Laboratory, Lawrence Livermore National Laboratory, 1997.

[5] Lindstrom, P. a Pascucci, V., *Visualization of Large Terrains Made Easy*. Lawrence Livermore National Laboratory, 2001.

[6] de Boer, W. H., *Fast Terrain Rendering Using Geometrical MipMapping*. 2000.

[7] Lindstrom, P. a Pascucci, V., *Terrain Simplification Simplified, A General Framework for View-Dependent Out-of-Core Visualization*. Lawrence Livermore National Laboratory, 2002.

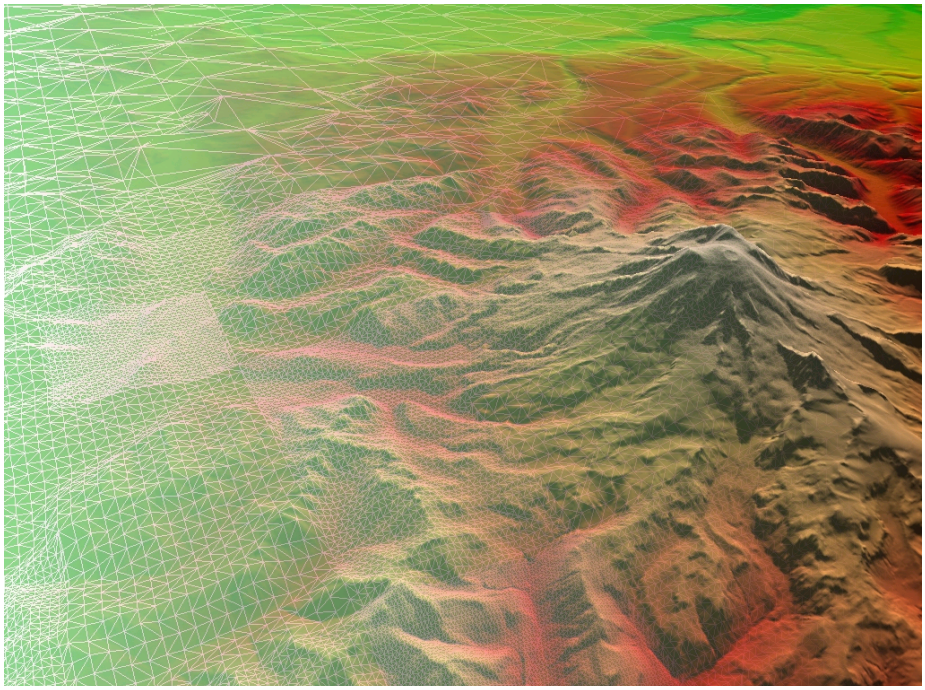
[8] Evans, F., Skiena, S. a Varshney, A., *Optimizing Triangle Strips for Fast Rendering*. State University of New York at Stony Brook, 2001.

[9] Seznam.cz, a.s., "Licenční podmínky mapových podkladů", [cit. 2013-05-20], ⟨http://napoveda.seznam.cz/cz/mapy/mapy-licencni-podminky/⟩, 2013.

[10] Defense Mapping Agency, *The Universal Grids, Universal Traverse Mercator (UTM) and Universal Polar Stereographic (UPS)*. Defence Mapping Agency Hydrographic Topographic, 1986.

Příloha A

Obrazová příloha



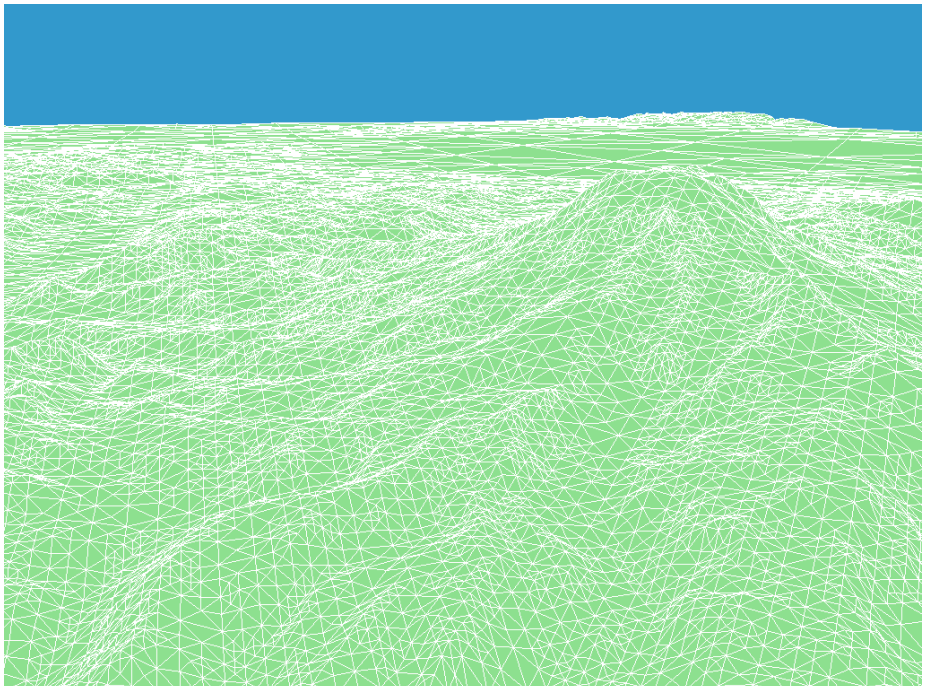
Obrázek A.1: GMM s texturou a náznakem geometrie

A.1 Chování algoritmů

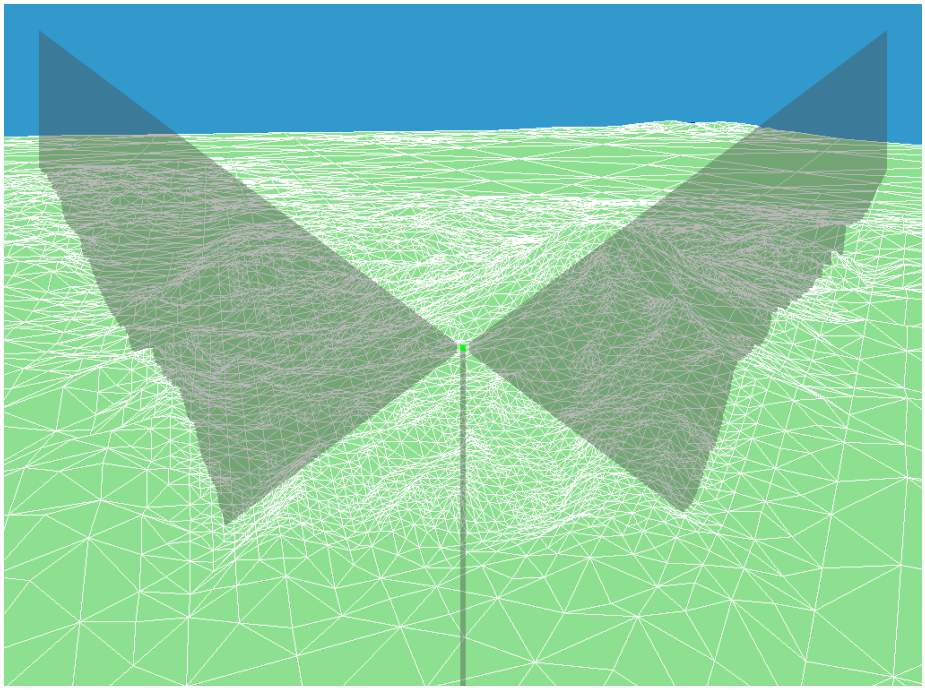
Schopnost LOD algoritmů omezit počet trojúhelníků bez větší ztráty vizuální kvality nejlépe předvedou obrázky.

Částečné oddálení kamery na obr. A.3 z pozice na obr. A.2, umožňuje sledovat vývoj geometrie terénu mimo rámec pohledu kamery, ten je ohraničen šedivými trojúhelníky.

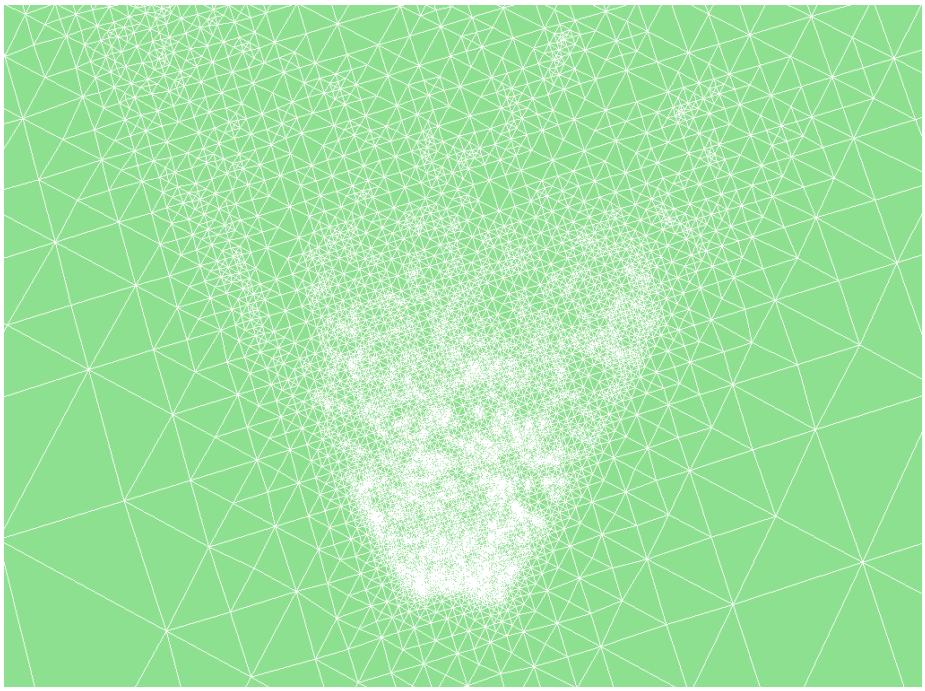
Při pohledu z vrchu na model terénu (obr. A.4), vytvořený podle polohy kamery na obr. A.2, jsou krásně vidět postupně se snižující detaily geometrie s rostoucí vzdáleností od kamery. Podobně vystupuje hranice mezi terénem v záběru kamery a mimo ni.



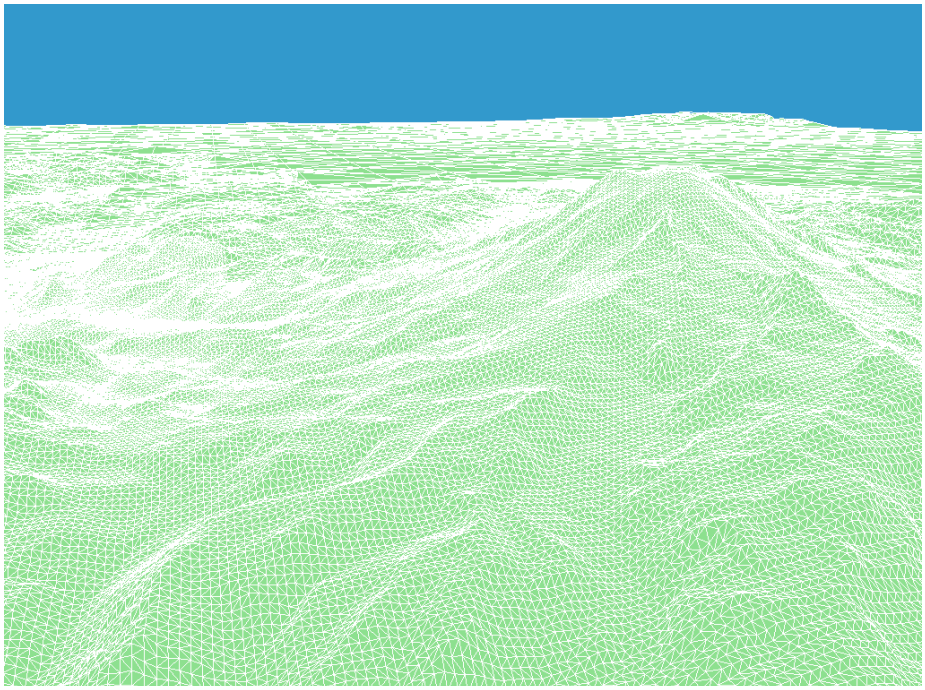
Obrázek A.2: Terén vykreslen SOARem, jedná se o data použitá v testech, $\tau = 3$, $18,5$ tis. trojúhelníků.



Obrázek A.3: Terén vykreslen SOARem, view frustum

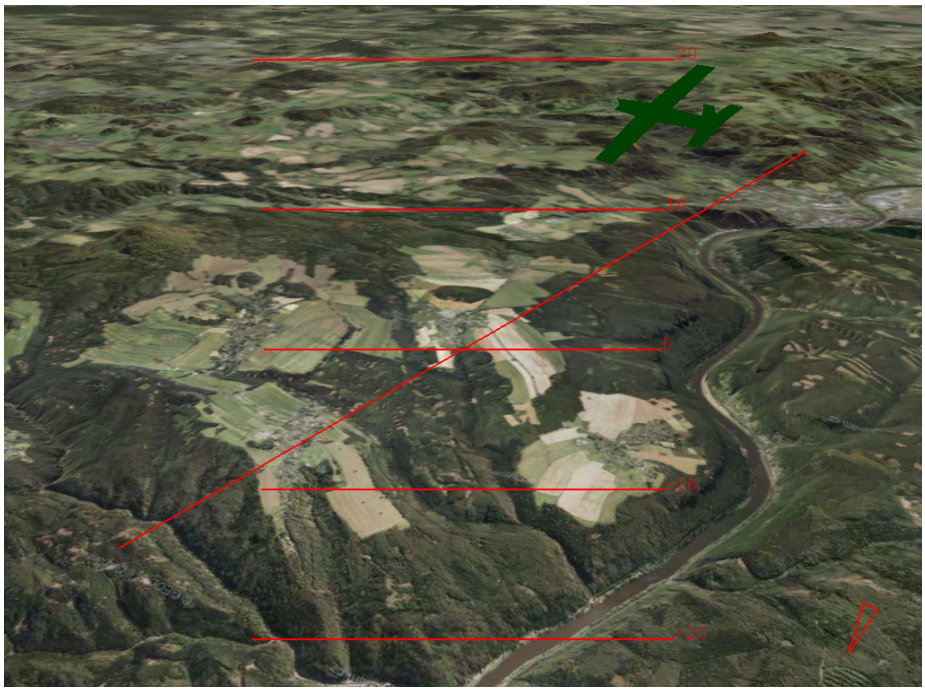


Obrázek A.4: Terén vykreslen SOARem, view culling



Obrázek A.5: Terén vykreslen GMM, jedná se o data použitá v testech a stejný pohled jako na obr. A.2, $\tau = 3$, 205 tis. trojúhelníků.

A.2 Aplikace



Obrázek A.6: Snímek z aplikace s FGS, letadlo se nachází poblíž 51° s.š. a 14° v.d., textura v rozlišení 16 m/px.

Příloha B

Obsah přiloženého CD

Na přiloženém CD se nachází především spustitelný program, zdrojové kódy a dokumentace, dále pak data terénů umožňující reprodukci testů a rozšířená obrazová příloha.

- Data z testů
- Obrázky
- Vstupní data
- Zobrazení terénu.pdf
- Zobrazení terénu
  - bin
  - doc
  - src

Adresář Data z testů obsahuje veškeré naměřené hodnoty v průběhu testování algoritmů. Ve složce Obrázky je umístěna rozšířená obrazová příloha ilustrující chování a vlastnosti implementovaných algoritmů. Po spuštění aplikace je z adresáře Vstupní data možno zobrazit některý z přiložených terénů. Kořenový adresář CD dále obsahuje tuto práci v souboru Zobrazení terénu.pdf a adresář programu Zobrazení terénu. V něm se nachází spustitelný java archiv společně s potřebnými knihovnami (adresář bin), vyexportovaná dokumentace v html formátu (složka doc) a zdrojové kódy programu (adresář src).