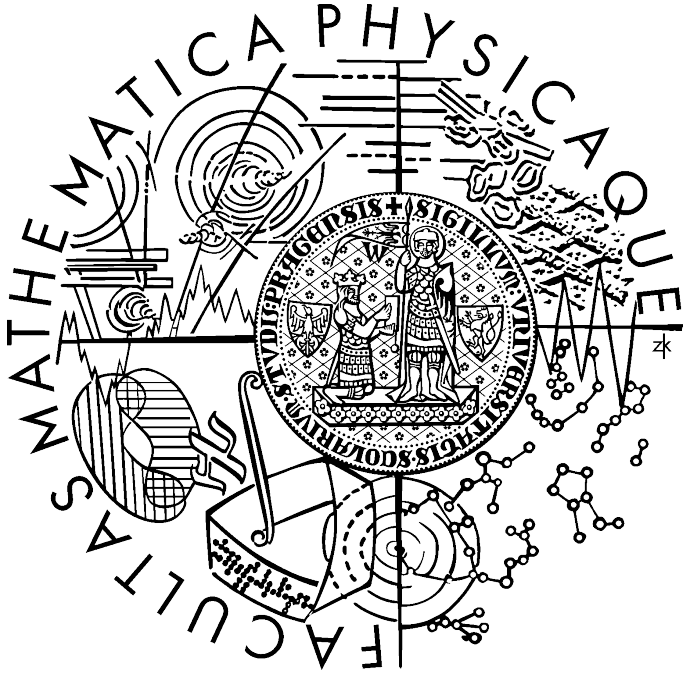
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Stanislav Kašný

Modelování a zobrazování terénu

Katedra software a výuky informatiky
Vedoucí diplomové práce: **Doc.Ing. Jiří Žára, CSc.**
Studijní program: **Informatika**
Studijní obor: **Softwarové systémy**

Poděkování

Na tomto místě bych chtěl poděkovat především Doc. Ing. Jiřímu Žárovi, CSc. za vedení mé diplomové práce. Dále bych chtěl poděkovat všem, kteří mě při psaní této práce podporovali.

Prohlášení

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

V Praze dne 12.4.2004 Stanislav Kašný

Obsah

1 Úvod....7

2 Popis přírodního prostředí....8

2.1 Více pohledů na jedno prostředí....9

3 Datové struktury....10

3.1 Povrch terénu – výšková mapa....10

3.1.1 Nepravidelná trojúhelníková síť....10

3.1.2 Pravidelná mřížka výškových bodů....10

3.1.3 Vrstevnicová výšková mapa....10

3.1.4 Rozdělení terénu do bloků....11

3.1.5 Úroveň detailů – LOD (Level of Detail)....11

3.1.6 Hierarchické datové struktury....11

3.1.7 Shrnutí....13

3.2 Mapování textur na terén....13

3.2.1 Letecké snímky....14

3.2.2 Detailní opakující se textury....15

3.2.3 Mipmapping....15

3.3 Popis prvků přírodního prostředí....15

3.3.1 Neměnné objekty, vyskytující se v krajině....16

3.3.2 Stromy....16

3.3.3 Distribuce různých prvků v terénu....18

3.3.4 Sítě řek, silnic....18

3.3.5 Obloha....18

4 Sedris.....19

4.1 Model reprezentace dat – DRM.....20

4.2 Specifikace kódování dat přírodního prostředí – EDCS.....26

4.3 Sedris – shrnutí.....26

5 Jiné datové formáty.....28

6 Techniky modelování povrchu terénu.....30

6.1 Generování povrchu virtuálního terénu.....30

6.2 Vytvoření nepravidelné trojúhelníkové sítě.....32

6.2.1 Výběr reprezentativních bodů terénu.....32

6.2.2 Triangulace bodů terénu.....34

7 Techniky zobrazování.....38

7.1 Optimalizace.....38

7.1.1 Poloha objektu v zorném jehlanu - view frustum culling.....38

7.1.2 Odstranění odvrácených ploch – backface culling.....39

7.1.3 Úroveň detailů – LOD (Level of Detail).....39

7.2 Zobrazování nepravidelné trojúhelníkové sítě pomocí LOD.....40

7.3 Zobrazení pravidelné mřížky výškových bodů pomocí LOD.....44

7.4 Vytváření sítě pomocí LOD - shrnutí.....49

8 Prohlížeč terénů.....50

8.1 Vstupní data ve formátu Sedris.....50

8.2 Předzpracovaná data.....50

8.3 Algoritmy použité v programu.....53

8.4 Rychlost zobrazování dat.....55

8.5 Objem dat v paměti.....60

8.6 Srovnání s programem Side-by-Side Viewer.....61

9 Závěr.....63

10 Seznam literatury.....64

11 Přílohy.....66

A. Uživatelská dokumentace 66

B. Struktura CD.....71

Název práce: Modelování a zobrazování terénu

Autor: Stanislav Kašný

Katedra: Katedra software a výuky informatiky

Vedoucí diplomové práce: Doc. Ing. Jiří Žára, CSc.

e-mail vedoucího: zara@fel.cvut.cz

Abstrakt: V této práci se zabýváme zobrazováním rozsáhlých terénů v reálném čase. Podáváme přehled o různých datových strukturách pro popis povrchu terénu a pro popis prvků vyskytujících se na něm. Seznámíme se s projektem Sedris, který slouží k uchování dat přírodního prostředí. Dále popisujeme různé typy metod automatického generování výškových map terénů. Podáváme přehled základních algoritmů a optimalizací používaných při zobrazování rozsáhlých terénů. Zaměřujeme se zejména na dynamické generování LOD pro výškové mapy v reálném čase. Druhá část práce popisuje vytvořený prohlížeč, který umožňuje v reálném čase procházet po virtuálním terénu. Vstupní data jsou ve formátu Sedris. Aplikace používá knihovnu OpenGL.

Klíčová slova: modelování, zobrazování, terén, Sedris

Title: Modeling and visualization of terrain

Author: Stanislav Kašný

Department: Department of Software and Computer Science Education

Supervisor: Doc. Ing. Jiří Žára, CSc.

Supervisor's e-mail address: zara@fel.cvut.cz

Abstract: This thesis investigates the real-time visualization of large areas of terrain. We process various data structures describing surface terrain and its features. We describe several methods to automatically generate terrain topography maps, and focus on the real-time generation of continuous levels of detail for the height fields. We introduce the Sedris Project, which stores data about natural environments. The second part of this thesis concerns the Terrain Viewer application, which is used to 'fly' over the virtual terrain. Input data is stored in the Sedris format. The application uses the OpenGL library.

Keywords: modeling, visualization, terrain, Sedris

1 Úvod

V této práci se budeme zabývat zobrazováním rozsáhlých terénů v reálném čase. Podáme přehled o různých datových strukturách pro popis povrchu terénu a pro popis prvků vyskytujících se na něm. Popíšeme základní algoritmy a optimalizace používané při zobrazování. Budeme se zabývat pouze těmi algoritmy, jejichž zobrazovací primitiva jsou trojúhelníky. Představíme různé typy metod pro generování virtuálního povrchu terénu. Dalším z cílů je vytvořit prohlížeč, který umožňuje procházet po virtuálním terénu, který je zadán ve formátu Sedris. V následujícím odstavci stručně popíšeme, co v které kapitole budeme rozebírat.

Ve druhé kapitole si ujasníme, co všechno může tvořit databázi terénu. Třetí kapitola se bude věnovat datovým strukturám. Nejprve rozebereme geometrii povrchu terénu, coby výškovou mapu, a způsoby jejího uložení. Dále se budeme věnovat texturám nanášeným na povrch terénu. Ve zbývající části této kapitoly se budeme zabývat popisem prvků vyskytujících se v krajině. Ve čtvrté kapitole se seznámíme s projektem Sedris a jeho datovým formátem pro ukládání dat přírodního prostředí. Původně jsme chtěli zkoumat možnosti použití projektu Sedris při zpracování dat v reálném čase. Tento formát je ale primárně určen pro sdílení dat. Pokud chceme data v tomto formátu zobrazovat v reálném čase, je nutné tato data předzpracovat, například transformovat do jiného formátu. Z tohoto důvodu věnujeme projektu Sedris pouze tuto kapitolu. V páté kapitole popíšeme jiné datové formáty, které lze použít pro uchování dat přírodního prostředí. Šestá kapitola se bude věnovat technikám modelování povrchu terénu. Shrneme typy metod pro vytváření počítačem generovaného náhodného virtuálního terénu, jejich výhody a nevýhody. Dále představíme algoritmy na vytvoření nepravidelné trojúhelníkové sítě z husté pravidelné mřížky výškových bodů. V sedmé kapitole se budeme zabývat zobrazováním terénu v reálném čase. Popíšeme základní optimalizační techniky na redukci počtu zobrazovaných trojúhelníků. Dále se budeme věnovat generování trojúhelníkových sítí pomocí různých úrovní detailů v reálném čase. Jedním z našich cílů bylo vytvořit prohlížeč terénů, který umožňuje procházet po virtuálním terénu v reálném čase. Vstupní data jsou zadána ve formátu Sedris. Tuto aplikaci představíme v osmé kapitole.

2 Popis přírodního prostředí

V této kapitole popíšeme různé prvky, ze kterých se může skládat databáze popisující terén.

Základní povrch terénu

Popisuje základní povrch terénu, jeho geometrii. Jedná se o podloží, na kterém dále popisujeme různé jiné objekty. Jde o výškový model povrchu.

Charakter povrchu terénu

Popisuje hlavní rysy terénu. Například zdali je terén přírodní, vytvořený člověkem. Zda se jedná o vegetaci, hydrologii. Popisuje útvary jako silnice, řeky, aglomeraci, nějaké překážky. Tyto údaje můžeme získat z jiných zdrojů než z jakých jsme získali geometrii terénu. Proto se může stát, že na sebe data přesně nepasují. Dochází pak k různým paradoxům. Vidíme pak například řeku tekoucí do kopce, silnici vedoucí v jezeře, apod. Je proto nutné při začlenění data vždy řádně upravit.

3D Modely

V přírodním prostředí se většinou vyskytují trojrozměrné modely různých struktur. Jedná se většinou o budovy, stromy, auta. Umístění těchto modelů můžeme získat například z leteckých snímků.

Vlastnosti objektů a hlavních rysů terénu

Popisují další přídavné informace týkající se prostředí, případně prvků v prostředí. Může se jednat o věci jako teplota, vítr v daném místě. Informace o provozu na silnici, různé jiné vlastnosti prostředí. Například popisuje specifické znaky různých objektů.

Model prostředí

Popisuje úkazy jako déšť, vítr, mlhu, smog.

Další data

Zahrnuje další informace o terénu. Může se jednat například o topologii. Případně zde mohou být popsány různé události, například výbuch sopky.

2.1 Více pohledů na jedno prostředí

Každá aplikace se může dívat na terén jiným způsobem, každá potřebuje trochu jiná data. Představme se aplikaci, která simuluje pohyb po terénu v nějakém vozítku, a poté jinou aplikaci, která simuluje let letadla tisíc metrů nad povrchem. V první aplikaci potřebujeme detailně znát všechny překážky, vědět, kde se nachází silnice, potřebujeme přesně vykreslit terén. Na druhé straně v druhé aplikaci nám stačí letecký snímek dané krajiny, případně nasnímaná výšková data. V tomto případě by nám byly detailní data spíše na obtíž, neboť bychom je nedokázali v reálném čase zpracovávat.

3 Datové struktury

3.1 Povrch terénu – výšková mapa

3.1.1 Nepravidelná trojúhelníková síť

Mějme reprezentativní výškové body terénu. Tyto body jsou rozložené po celém terénu. Měly by být hustěji rozmístěné v místech hrubšího povrchu terénu. Pro reprezentaci takto zadaného terénu se používá trojúhelníková nepravidelná síť, kde vrcholy trojúhelníků jsou tvořeny danými body. Hrany a plochy trojúhelníků říkají, jak interpolovat plochu mezi těmito vybranými body. Takto vytvořená síť může popisovat různé přírodní útvary jako jsou například jeskyně, převisy, apod. Tuto síť je velice obtížné modifikovat při změně terénu. Musíme vždy část sítě přepočítat.

3.1.2 Pravidelná mřížka výškových bodů

Pokud měříme terén z ptačí perspektivy, dostaneme výškové body uspořádané do pravidelné mřížky nezávisle na tom, k jak členitému terénu přistupujeme. Takto reprezentovaná data můžeme jednoduše uložit ve dvourozměrném poli. S touto reprezentací však nepopíšeme složité útvary – jeskyně, apod. Tato struktura se nijak nepřizpůsobuje komplikovanějšímu terénu. Na druhou stranu, změna výškových dat je v takovéto reprezentaci jednoduchá. Před zobrazením musíme tyto data triangulovat.

Výšková mapa jako obrázek

Výškovou mapu můžeme reprezentovat pomocí nějaké bitmapy. Mějme obrázek tvořený pouze pomocí odstínu šedi. Každý bod obrázku reprezentuje jeden bod v terénu. Barva bodu reprezentuje výšku. Obrázek je tvořený pomocí odstínu šedi, kde černá barva reprezentuje nejnižší výšku, bílá barva představuje nejvyšší výšku.

3.1.3 Vrstevnicová výšková mapa

Výškovou mapu můžeme mít zadanou také pomocí vrstevnic. Tento formát je ale pro zobrazování nevhodný a proto se převádí buď na pravidelnou mřížku výškových bodů, nebo na nepravidelnou trojúhelníkovou síť.

3.1.4 Rozdělení terénu do bloků

Rozsáhlé terény jsou většinou příliš veliké na to, aby se vešly celé do paměti, nebo aby se celé zobrazovaly. Proto v paměti udržujeme pouze určitou část této scény. Scéna se proto rozděluje do čtvercových bloků stejné velikosti, které se mohou dynamicky nahrávat do paměti.

3.1.5 Úroveň detailů – LOD (Level of Detail)

Úroveň detailů je důležitý mechanismum, pomocí nějž můžeme velice urychlit zobrazování scény. Tento mechanismus určuje kvalitu zobrazování v závislosti na vzdálenosti od pozorovatele. Pokud je nějaký objekt od nás vzdálený tak, že na obrazovce zabírá pouze pár pixelů, není nutné vykreslovat všechny jeho detaily. Naopak, pokud daný objekt je blízko pozorovatele a zabere velkou část obrazovky, vykreslíme tento objekt se všemi detaily. Často se proto různé modely vyskytující se v terénu i terén samotný uchovávají v různých úrovních kvality a při zobrazení vybereme model či terén v kvalitě v jaké právě potřebujeme. Nadále budeme značit úroveň detailů jako LOD.

3.1.6 Hierarchické datové struktury

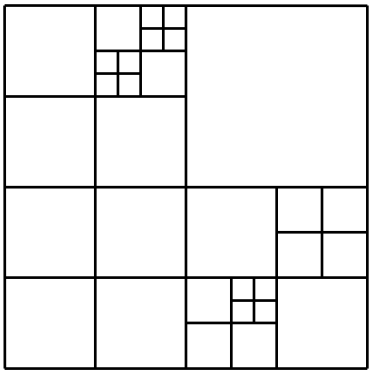
Hierarchické datové struktury se používají pro uchování LOD terénu. Terén je totiž nutné zobrazovat v různých místech pomocí různého LOD v závislosti na pozici pozorovatele a v závislosti na geometrii, či důležitosti terénu. Hierarchické datové struktury umožňují triangulaci terénu v různých místech pomocí různé úrovně. Tyto struktury můžeme rozdělit do dvou hlavních kategorií, podle toho, zda potřebují ke své reprezentaci pravidelnou mřížku výškových bodů či nikoliv. Tyto modely jsou podrobně popsány v [3].

Vrstvený model

Tento model dává přímočarý způsob, jak reprezentovat terén pomocí LOD. Obsahuje seznam nezávislých modelů terénu, kde každý takový model představuje původní terén v jiné úrovni detailu. Takovýto způsob má ovšem pár nevýhod. Největší nevýhodou je, že každá úroveň reprezentuje celý povrch dané oblasti a v této oblasti již nemůžeme rozhodovat, které části mají být vykresleny s většími detaily, a které s menšími. Další nevýhodou je, že nemůžeme plynule přecházet mezi jednotlivými úrovněmi.

Reprezentace pomocí kvadrantového stromu

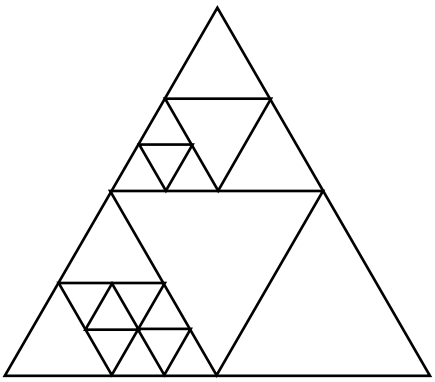
Model založený na kvadrantovém stromě potřebuje jako data pravidelnou mřížku výškových bodů. Poskytuje rozdělení terénu do obdélníků různých velikostí (viz obr. 3-1). Model je založen na rekurzivním dělení obdélníků zvaných kvadranty. Kdykoliv není aproximace nějakého obdélníku příliš dobrá, je tento obdélník rekurzivně rozdělen do čtyř subobdélníku, tvořících jednotlivé subkvadranty. Souhrn všech těchto obdélníků můžeme uspořádat do kvadrantového stromu. Nevýhodou tohoto modelu je obtížné udržování souvislosti terénu mezi jednotlivými bloky. Před zobrazením musíme tuto datovou strukturu triangulovat.



Obrázek 3-1 - Struktura kvadrantového stromu.

Reprezentace pomocí kvartérní triangulace

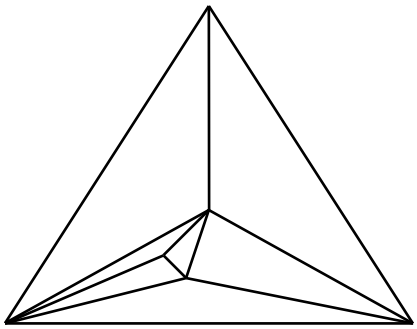
Tato reprezentace je konceptuálně podobná reprezentaci pomocí kvadrantového stromu. Povrch je nejprve aproximován plochou tvořenou iniciálním trojúhelníkem. Tento povrch je dále rekurzivně dělen do čtyř trojúhelníků spojením středů hran původního trojúhelníku (viz obr. 3-2). Nevýhodou tohoto modelu je opět obtížné udržování souvislosti terénu mezi jednotlivými bloky. Tento model potřebuje pravidelně nasnímaná data.



Obrázek 3-2 - Struktura kvartérní triangulace.

Reprezentace pomocí ternární triangulace

V tomto modelu je počáteční trojúhelník rekurzivně rozdělen na tři trojúhelníky pomocí nějakého vnitřního bodu (viz obr. 3-3). Ternární triangulace nepotřebuje ke své reprezentaci pravidelně nasnímaná data. Má ale jednu vážnou nevýhodu. Produkuje příliš úzké a dlouhé trojúhelníky, které vedou k nepřesnosti při zobrazování.



Obrázek 3-3 - Struktura ternární triangulace.

3.1.7 Shrnutí

Vyjmenovali jsme zde pouze nejznámější datové struktury. Existují i jiné datové struktury, které umožňují reprezentovat terén. Mezi ně patří například Delaunayho hierarchie, která je ovšem dost složitá. Další kapitolou, kterou jsme zde nijak neprozkoumali jsou hybridní datové struktury, které jsou kombinací dříve zmíněných struktur.

Každá datová struktura má své výhody, své nevýhody. Nikde jsme nenašli informace o tom, zda jsou nějaké struktury lepší pro daný typ terénu, zda například pro hornaté terény jsou lepší jiné struktury než pro rovinné terény. Při volbě datových struktur se tedy rozhodujeme podle jednoduchosti implementace, časové náročnosti při zobrazování a reálnosti zobrazené scény.

3.2 Mapování textur na terén

V předchozí kapitole jsme se zabývali tím, jakými strukturami popsat výškovou mapu. Povrch terénu je však nutné také otexturovat, abychom docílili reálnosti terénu. Existují dva základní druhy textur, které nanášíme na terén. Prvním z nich jsou letecké, případně satelitní snímky. Druhým typem jsou textury typické pro určitý druh terénu. Jedná se o malé detailní opakující

se textury obsahující typickou část terénu, například trávu, asfalt, kamení. Nejprve se budeme zabývat leteckými snímky – velkými texturami.

3.2.1 Letecké snímky

Při práci s velkými texturami, které chceme mapovat na terén, můžeme narazit na velikost paměti grafické karty. Je proto třeba pracovat pouze s částí dané textury. Jedním z možných přístupů je přerozdělení veliké textury do menších bloků a poté při zobrazování budeme pracovat pouze s těmi bloky, které jsou potřeba k otexturování scény. Druhou možností je vyříznout z veliké textury vždy tu část, která je potřeba k otexturování viditelné části scény. Pro detailnější vysvětlení viz [9, 20]

Rozdělení textury do bloků

Při rozdělování textur do bloků se objeví problémy při zobrazování scény na hranách bloků (viz [20]). Polygony, které leží zároveň ve více blocích je nutné rozdělit podél hranic a teprve poté je možné je správně otexturovat. Dále je třeba, aby textura obsahovala redundantní pixel na hranách bloků, aby grafický akcelerátor věděl, jak interpolovat body na hranicích. Jinak dochází k ostrým hranicím na přechodu bloků. Textury jednotlivých bloků je vhodné udržovat v různých úrovních detailů.

Výhodné je rozdělit textury stejně jako rozdělujeme výškovou mapu, usnadní nám to mnoho práce. Následují příklad možného rozdělení: mějme scénu zadanou pomocí kvadrantového stromu. Pro každý uzel stromu definujeme texturu stejné velikosti (např. 256x256). Kořen stromu bude mít k dispozici pouze hrubou texturu bez detailů. Uzel definující jemnější rozdělení terénu bude mít tedy k dispozici detailnější texturu.

Vyříznutí části velké textury podle viditelnosti scény – viz [9]

Omezme velikost viditelné části scény tak, aby nám stačil pouze vyříznutý blok z celé textury o nějaké předem určené velikosti. Poté při zobrazení scény předáme grafickému akcelerátoru danou část textury. Při pohybu pozorovatele (při změně viditelné části scény) musíme daný blok textury aktualizovat. Při aktualizaci nemusíme přehrávat celou texturu, stačí pouze část. Využijeme toho, že při změně viditelné části scény se většinou zobrazovaná část textury pouze posune o nějaký malý krok ve velké textuře. To znamená, že část textury nám zmizí a na druhé straně nám část přibude. Onu novou část můžeme nahrát na místo staré nepoužívané

části. Tato technika funguje díky možnosti opakujícímu se mapování textur i mimo velikost textury.

3.2.2 Detailní opakující se textury

Pokud je pozorovatel blízko povrchu, jeden texel textury může pokrýt více pixelů na obrazovce. Zobrazovaný terén se tak stává nerealistický. Jedno z řešení tohoto problému spočívá v použití jemnější textury, která ale zároveň zabírá více místa v paměti. Zobrazovaný terén může být z velké části velice podobný. Představme si například povrch asfaltové silnice. Ten vypadá ve všech místech stejně. Je proto záhodno udržovat si v paměti pouze malý vzorek daného povrchu a ten pak opakovat po celé ploše. Pokud se pozorovatel vyskytne blízko daného povrchu zkombinujeme detailní malé textury spolu s velkou texturou celého povrchu, abychom docílili zjemnění. Tyto detailní textury však používáme pouze v blízkosti pozorovatele od daného povrchu.

Detailní textury (textury typické pro specifický typ terénu – geotypické textury) můžeme použít i samostatně. Pokud určíme, kde se jaký typ terénu nachází, můžeme na něj pak opakovaně nanášet jednotlivé geotypické textury. Pro každý specifický typ terénu máme dvourozměrnou masku, která určuje, kde se daný typ nachází a v jakém množství. Při zobrazení je nutné smísit adekvátní textury a ty poté nanést na povrch terénu. Metody detailních textur jsou popsány v [8, 20].

3.2.3 Mipmapping

Mipmapping je technika používaná pro odstranění vizuálních chyb vzniklých při mapování textury na vzdálené objekty. Mapovanou texturu si uchováváme v různých předfiltrovaných rozlišeních. Při vykreslování otexturovaného polygonu nejdříve zjistíme velikost povrchu vůči nanášené textuře, a poté vybereme vhodné rozlišení textury. Tato technika je podporována přímo pomocí OpenGL.

3.3 Popis prvků přírodního prostředí

3.3.1 Neměnné objekty, vyskytující se v krajině

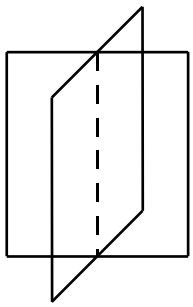
V tomto případě máme na mysli objekty, které se v prostředí mnohokrát opakují, přičemž jejich vzhled je stále stejný – např. budovy, (viz [20]). Je nesmyslné definovat pro každou instanci modelu novou síť trojúhelníků. Místo toho máme jakousi knihovnu modelů a poté již jen v terénu určíme umístění modelu a typ modelu z knihovny. Je vhodné udržovat modely v různých úrovních detailů a zobrazovat tyto úrovně v závislosti na pozici pozorovatele.

3.3.2 Stromy

V této kapitole rozebereme několik přístupů, jak reprezentovat stromy – viz [20].

Billboardová technika

Toto je nejčastější technika používaná pro reprezentaci stromů. Obrázek stromu je nanesen na dva na sebe kolmé polygony (viz obr. 3-4).



Obrázek 3-4 - Dva na sebe kolmé polygony, na které nanášíme texturu stromu.

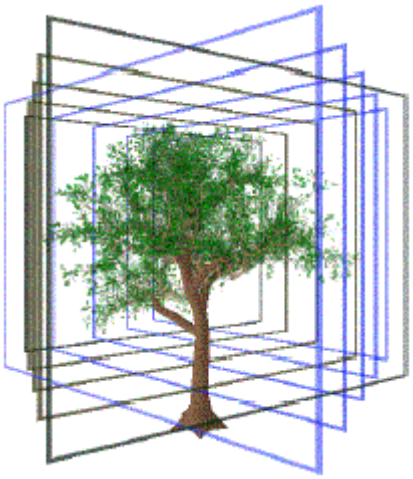
Případně můžeme použít jeden polygon a otáčet jej tak, aby byl vždy natočený směrem k pozorovateli. Stín, který strom vrhá na zem reprezentujeme černým poloprůhledným polygonem.

Směrové billboardy

Pro reprezentaci stromu touto technikou použijeme jeden polygon, který budeme opět otáčet směrem k pozorovateli (otáčíme pouze kolem osy z). Myšlenka této metody je nanášet různé textury v závislosti na úhlu pohledu pozorovatele. Při této metodě bychom se na stromy měli dívat pouze ze strany. Metoda není vhodná pro průlet nad stromy.

Metoda řezů

Tato metoda rozšiřuje metodu billboardů. Listoví stromu je vykreslováno pomocí řady řezů – billboardů - tak, jak ukazuje obrázek 3-5. Každý z těchto řezů obsahuje část listoví. Pokud předpokládáme pohledy shora na daný objekt, je třeba vést navíc řezy ve vodorovné rovině. Vše je podrobněji vysvětleno v [6].



Obrázek 3-5 - Strom vykreslený pomocí řady řezů (obrázek viz [6]).

Ručně modelované stromy

Toto je mezistupeň mezi billboardovou technikou a stromem, definovaným pomocí velkého množství polygonů. Strom má většinou kmen tvořený z polygonů a každá část větví je tvořena billboardem, případně menším množstvím polygonů. Každý strom je třeba namodelovat zvlášť.

Dále se používají různé algoritmy pro generování stromů. Ty generují realistické pravděpodobnostní polygonální modely, jejichž výhodou je, že každý strom vypadá trochu jinak. Tyto algoritmy jsou často založeny na fraktálech. Příkladem mohou být právě L-systémy.

L-systémy

Tento algoritmus generování stromů je založen na Chomského hierarchii. Při zvolení vhodné gramatiky vytváříme řetězce reprezentující strom. Výsledný objekt se chová jako fraktál. Pomocí jednoduchých pravidel tak můžeme generovat složité geometrické útvary. Tato metoda tvoří velice realistické stromy, nicméně na úkor rychlosti.

Samozřejmě na stromy generované pomocí množství polygonů je záhodno použít LOD.

3.3.3 Distribuce různých prvků v terénu

Existují dvě cesty, jak popsat distribuci (rozšíření, pokrytí) prvků v terénu (viz [20]). První z nich popisuje rozšíření pomocí bitmapy pokrytí. Definuje tak konkrétní informace pro každý element výškové mapy. Vytvoření této reprezentace je poměrně jednoduché, lehce určíme hustotu prvků v různých místech. Na druhou stranu pomocí této reprezentace se složitě určují hranice daného rozšíření prvků. Druhý přístup popisuje ono rozšíření pomocí ploch polygonů. K vytvoření této datové struktury jsou již třeba programové nástroje. Tento způsob lépe popisuje hranice pokrytí.

3.3.4 Sítě řek, silnic

Topologie sítě obecně reprezentuje spojení silnic, řek, jako uzly, které se odkazují na jednotlivé segmenty této sítě (viz [20]). Jednotlivé segmenty mohou být reprezentovány různými způsoby:

- Povrch úseků reprezentovaný pomocí polygonů.
- Úseky definované pomocí jednoduchých křivek. Pro vykreslení musíme mít další informace definující šířku a další vlastnosti segmentů.
- Definice pomocí hranic daných objektů. (hranic řek, silnic)

3.3.5 Obloha

Obloha se aproximuje velkou kopulí nad terénem, kterou pokryjeme texturou, případně obarvíme (viz [20]). Otexturování kopule vypadá realisticky, ale má několik nevýhod. Těžko měníme výraz oblohy během dne a pak tato metoda umožňuje zobrazovat pouze značně vzdálené mraky. Při zobrazení bližších mraků bychom při průletu nemohli simulovat pohyb těchto mraků. Při obarvování oblohy pozvolna měníme barvu oblohy od horizontu k zenitu. Při zvolení vhodných barev můžeme simulovat přechod mezi různými fázemi dne.

4 Sedris

Práce s daty hraje klíčovou roli v aplikacích zabývajících se přírodním prostředím. Mnoho aplikací si definuje vlastní formát na popis či práci s těmito daty. Sedris je open source projekt, zabývající se popisem a výměnou (sdílením) dat přírodního prostředí. Projekt Sedris byl zahájen v roce 1994. Sedris popisuje přírodní prostředí ze všech možných stránek. Všechny informace o Sedrisu jsou k nalezení v [18].

K reprezentaci dat Sedris poskytuje:

- model reprezentace dat (Data Representation Model - DRM)
- specifikace kódování dat přírodního prostředí (Environmental Data Coding Specification - EDCS)
- prostorový model (Spatial Reference Model - SRM)

Pro práci s těmito daty nabízí Sedris:

- své vlastní API (Sedris Interface Specification - API)
- formát dat (Sedris Transmittal Format - STF)

Navíc Sedris nabízí různé nástroje pro práci s těmito daty.

Model reprezentace dat - DRM

Model reprezentace dat - DRM - je objektově založený datový model obsahující v sobě abstraktní třídy a jejich vztahy, atributy. DRM není implementace databáze, ale je to popis typů dat, které se mohou vyskytovat v databázi. DRM obsahuje třídy potřebné k reprezentaci přírodního prostředí, ale neobsahuje abstrakce objektů v prostředí. Například DRM neobsahuje třídy pro strom, budovu, ale obsahuje třídy nezbytné pro popis těchto objektů. K popisu hierarchie se používají různé typy vztahů. DRM model obsahuje třídy pro geometrii, topologii, textury, vlastnosti, tabulky, knihovny a mnoho dalších tříd.

Specifikace kódování dat přírodního prostředí – EDCS

EDCS poskytuje klasifikaci objektu jako například pojmenování, identifikaci, případně charakter těchto dat. EDCS se nezabývá tím, jak jsou dané objekty representovány. Říká pouze co daný objekt je, popisuje jeho vlastnosti a jednotky, které se používají na měření

těchto vlastností. Říká například, že nějaký objekt je strom, nezávisle na tom, jak je definovaný.

Prostorový vztahový model – SRM

SRM unifikuje prostorové modely používané Sedrisem. Tyto modely zahrnují například inerciální, neinerciální soustavy, kartézské, polární souřadnice. SRM zároveň obsahuje různé konverzní knihovny, pomocí kterých je možné se dívat na různá data pomocí různých prostorových modelů.

API Sedrisu a formát dat - API & STF

API pro přístup a práci k datum přírodního prostředí je psané v C++, běžících na systémových platformách Unix, Windows, Linux. Formát dat je souborově založený a je navržený pro maximální úsporu místa.

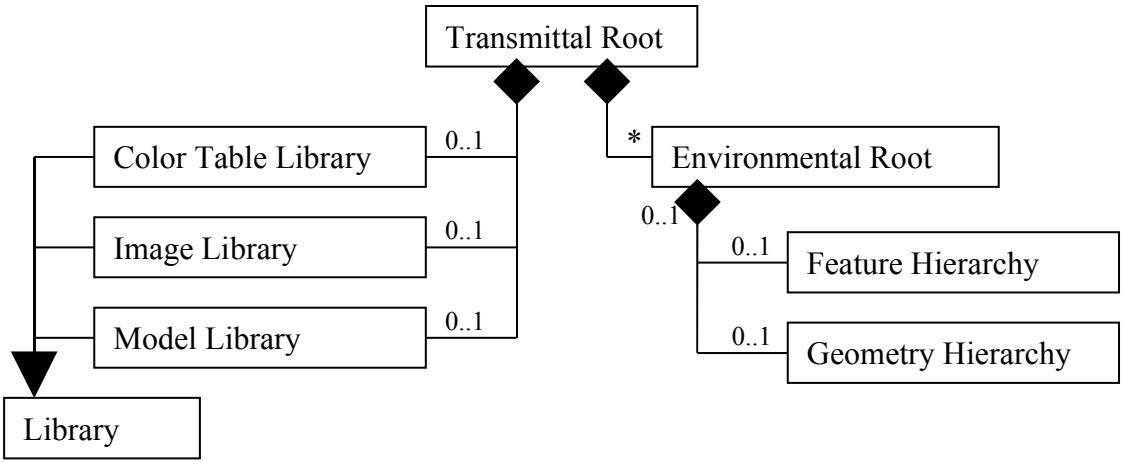
4.1 Model reprezentace dat – DRM

Jak již bylo popsáno dříve, model reprezentace dat nám definuje různé typy objektů, jejich atributy a vztahy mezi těmito objekty. Příkladem objektu může být bod prostoru, mezi jeho atributy patří jeho trojrozměrné souřadnice. Jsou definovány tři základní vztahy mezi objekty. Dědičnost, asociace a agregace.

- **Dědičnost** – Objekt může být definován jako potomek nějakého objektu. To znamená, dědí všechny jeho atributy a navíc si definuje sám nějaké vlastní.
- **Asociace** – Dva objekty jsou asociované, to znamená, ví o sobě navzájem.
- **Agregace** – Jeden objekt může vlastnit více jiných objektů jakožto své atributy.

Podívejme se nyní na základní typy objektů. Všechny typy objektů budu popisovat pomocí UML. Nebudeme zde kvůli velkému počtu zobrazovat všechny typy a vztahy, ale jen ty základní.

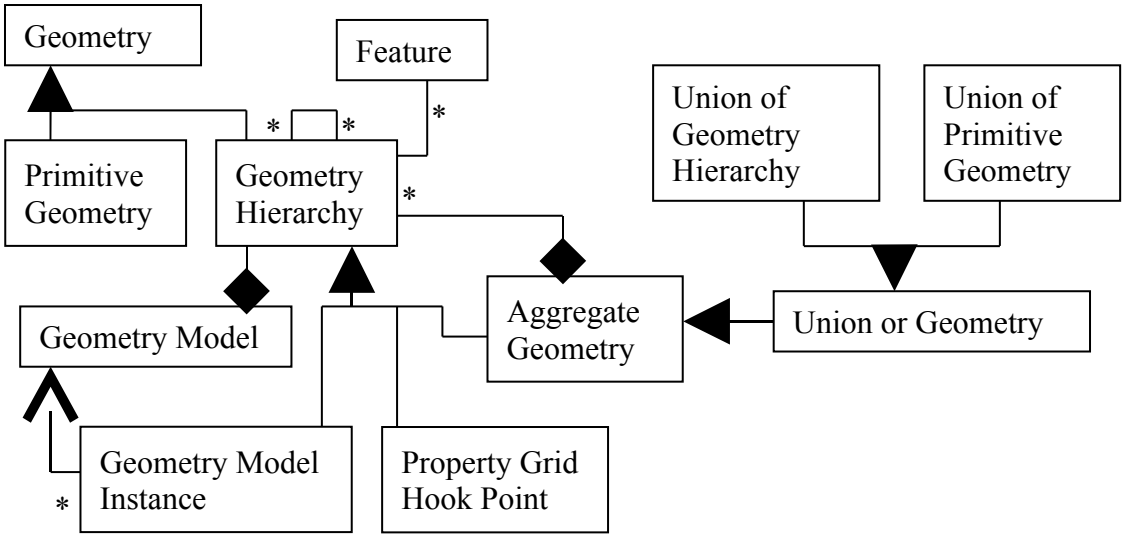
K jakýmkoliv datům přírodního prostředí se dostaneme přes základní objekt *Transmittal Root* (viz obr. 4-1), který tvoří kořenový objekt celého prostředí. Přes něj se můžeme dostat ke knihovnám modelů, barev, textur, k popisu samotné scény – objektu *Environmental Root*.



Obrázek 4-6 - Kořenový uzel sedrisovské datové struktury.

Geometry Hierarchy je nejdůležitější uzel, co se týče vizualizace scény. Toto je uzel, pomocí kterého se dostaneme k polygonům scény (pokud je scéna tvořena pomocí polygonů). Pomocí objektu *Feature Hierarchy* můžeme vytyčit ty části scény, které mají nějaký speciální význam. Můžeme zde například určit hranice silnice, co ze scény představuje řeku, co představuje překážku.

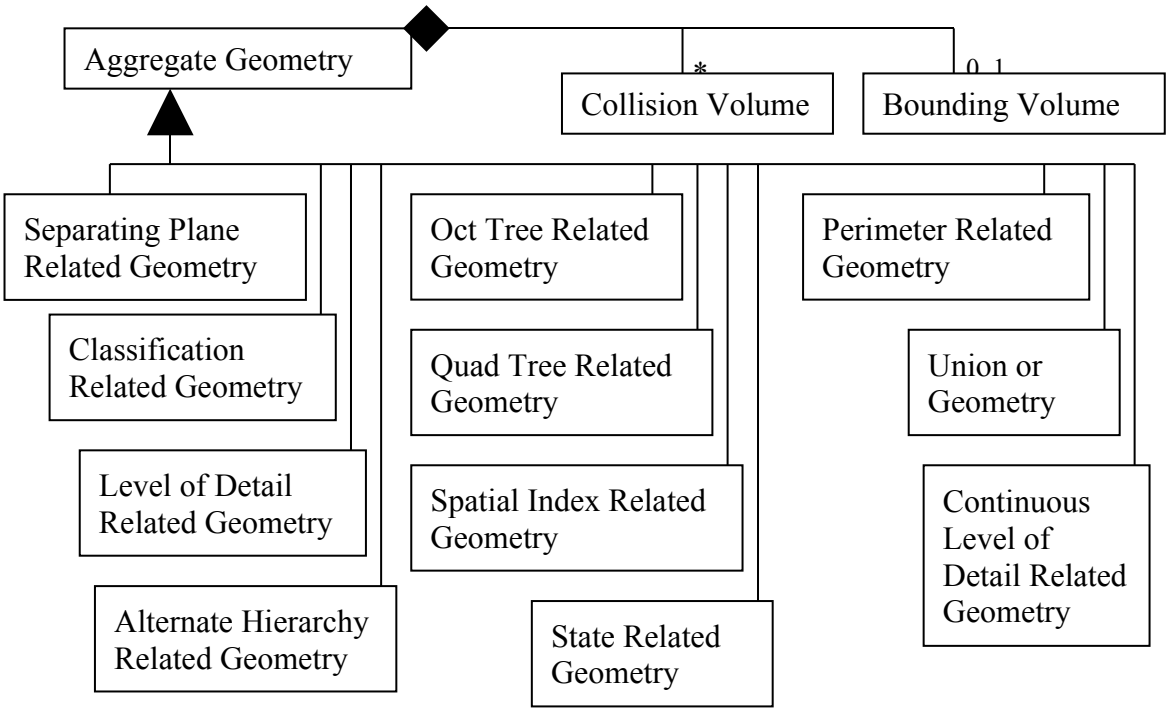
Schéma na obrázku 4-2 popisuje strukturu uzlu geometrie scény – *Geometry Hierarchy*. Tento uzel může reprezentovat buď popis nějakého modelu, část scény zadanou pomocí výškové mapy, seznam nějakých primitivních geometrických útvarů, případně rekurzivně rozvětvenou hierarchickou strukturu geometrických objektů.



Obrázek 4-7 - Struktura objektů zabývajících se geometrií.

Každá skupina geometrických objektů může být propojena s uzlem *Feature*, který popisuje určité vlastnosti daného objektu.

Popis definice scény můžeme oddělit do různých částí a poté si mezi těmito částmi vybírat. Diagram na obrázku 4-3 ukazuje možná rozdělení.



Obrázek 4-8 - Možná rozdělení geometrie scény.

Popíšeme jednotlivé geometrické rozdělení.

Separating Plane Related Geometry – Tento uzel rozděluje celou scénu pomocí množiny oddělovacích rovin.

Classification Related Geometry – Uzel rozdělující scénu pomocí nějaké klasifikace. Tato klasifikace je založena na specifikaci kódování dat přírodního prostředí – EDCS. Představme si scénu tvořenou lesem a jezerem. Pak zde můžeme použít tento uzel, který obsahuje dvě hierarchie objektů. První obsahující množinu polygonů lesa v sobě bude mít edcs značku *EDCS_CC_FOREST*, zatímco druhý bude obsahovat značku *EDCS_CC_WATER*.

Level of Detail Related Geometry – Tento uzel obsahuje alternativní reprezentace téhož prostředí, přičemž každou tuto reprezentaci použijeme při jiných příležitostech. Může se

jednat například o rozdělení v závislosti na vzdálenosti od pozorovatele, na velikosti měřítku mapy, na velikosti prostorového rozlišení.

Alternate Hierarchy Related Geometry – Jednotlivé uzly tohoto rozdělení reprezentují totéž prostředí. Každá z těchto reprezentací ukazuje na objekt, který určuje pomocí čeho vybíráme různé reprezentace při práci. Představme si, že k nějakému prostředí potřebujeme přistupovat jak podle umístění, tak podle klasifikace daného prostředí. Abychom tohoto dosáhli, použijeme tento objekt, se dvěmi komponentami. První komponenta bude obsahovat *Spatial Index Related Geometry*, zatímco druhá komponenta bude obsahovat *Classification Related Geometry*. Oba dva uzly budou reprezentovat to samé prostředí.

Oct Tree Related Geometry – Tento uzel rozděluje trojrozměrnou scénu podél tří os, které definují osy kartézských souřadnic.

Quad Tree Related Geometry – Tento uzel rozděluje scénu na 4 kvadranty pomocí dvou os, zadaných kartézskými souřadnicemi.

Spatial Index Related Geometry – Celou scénu rozsekáme na velké obdélníkové bloky. Každý blok si můžeme představit jako jednu část dvojrozměrné mřížky. Tento uzel obsahuje jednotlivé části mřížky. Poskládáním těchto částí dostaneme celou mřížku. Tento objekt se hojně využívá při popisu rozsáhlých terénů.

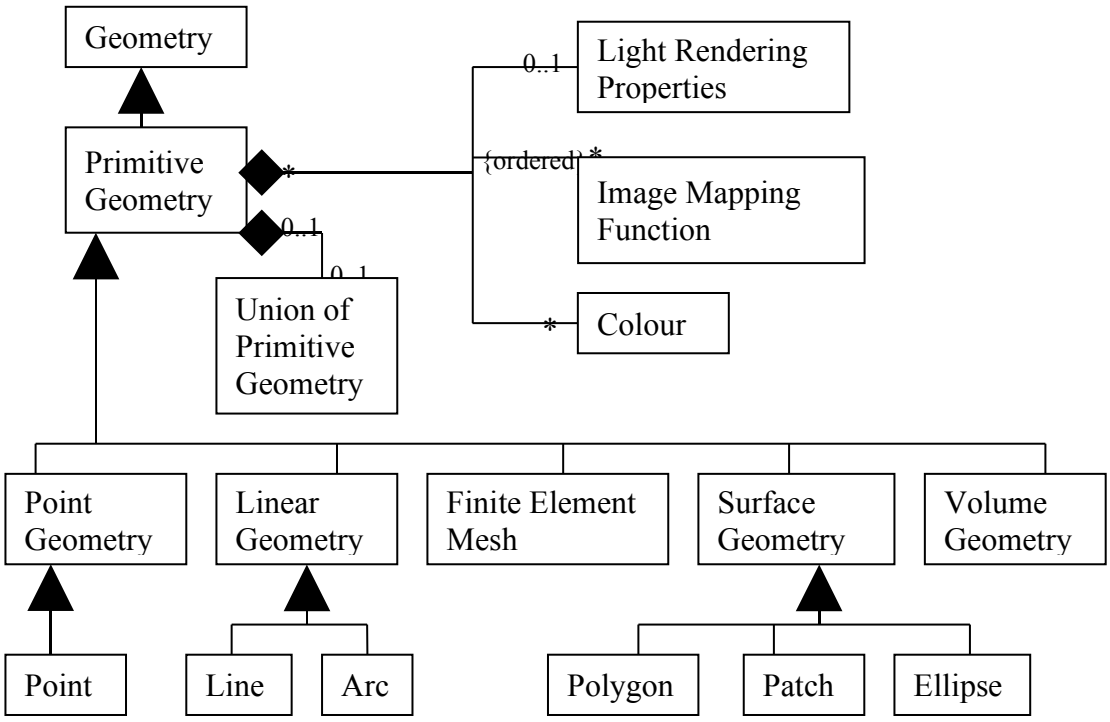
State Related Geometry – Popisuje objekt v závislosti na nějakém stavu určité položky. Podle této položky vybereme vhodnou reprezentaci. Představme si nějakou budovu, která může mít 4 stupně poškození. Kód *EDCS_AC_DAMAGE_GENERAL* zde bude použit jako položka určující, která reprezentace objektu se má použít.

Perimeter Related Geometry – Pomocí tohoto uzlu můžeme scénu rozdělit do různých dvourozměrných oblastí.

Union or Geometry – Tento uzel je základním uzlem pro definici geometrie scény.

Continues Level of Detail Related Geometry – Tento uzel reprezentuje souvislý, přizpůsobivý terén, kdy každý uzel nahrazuje polygon předchozího uzlu množinou polygonů, které reprezentují ten samý útvar z bližšího pohledu.

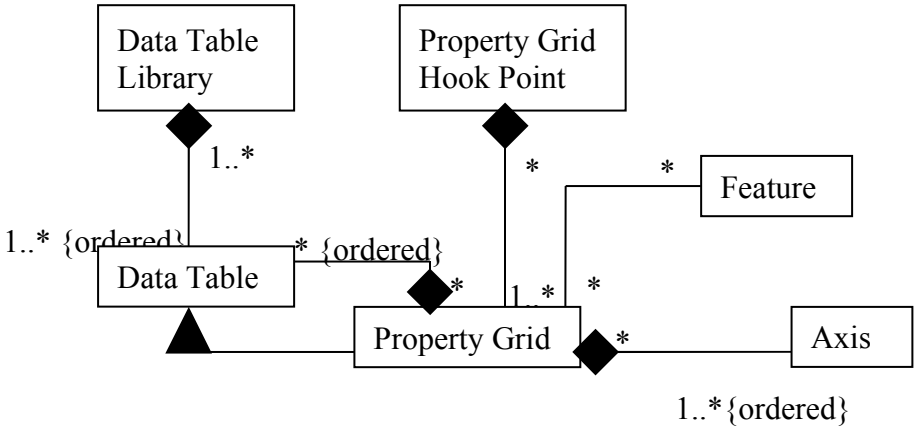
Obrázek 4-4 nám dává představu o tom, jaká geometrická primitiva máme k dispozici - z jakých základních elementů se scéna může skládat.



Obrázek 4-9 - Geometrická primitiva.

Vidíme, že všechna primitiva můžeme rozdělit na jednorozměrné, dvourozměrné, a trojrozměrné útvary.

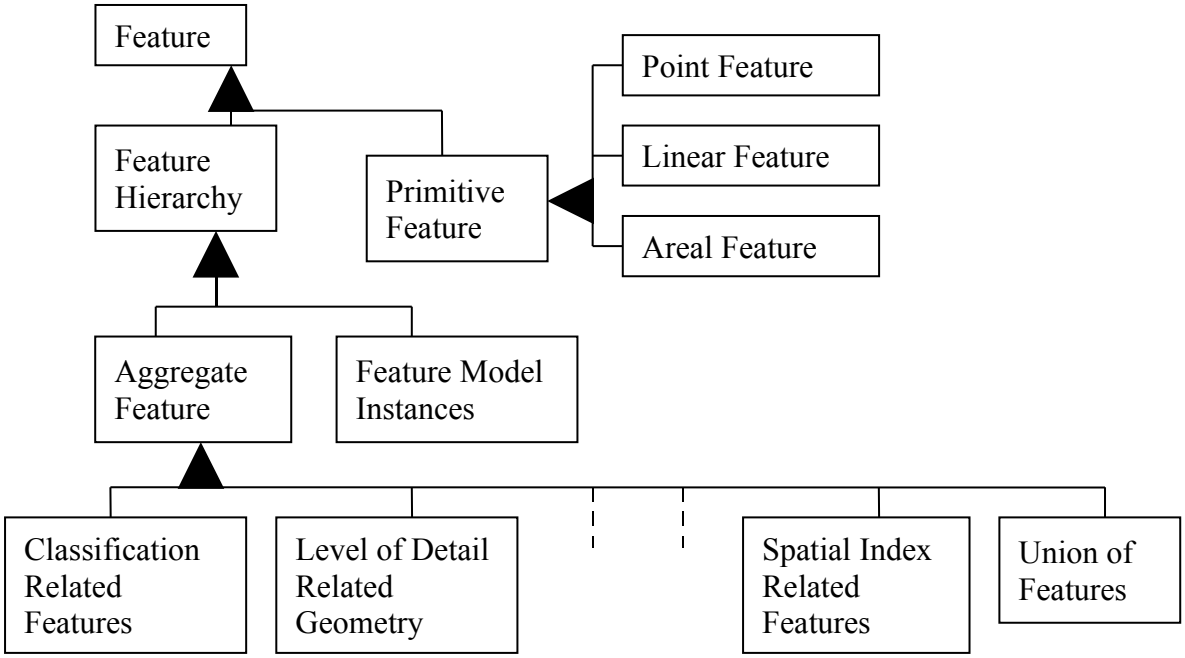
Pokud chceme využít k reprezentaci tabulku hodnot, například nasnímaná výšková data v určitých přesně daných intervalech, využijeme objekty zobrazené v diagramu 4-5.



Obrázek 4-10 - Výšková mapa zadaná pomocí pravidelné mřížky.

Property Grid – Tento objekt reprezentuje tabulku hodnot, která má nejméně jednu a maximálně tři prostorové souřadnice. Každá souřadnice je zadána pomocí objektu *Axis*. Tabulka udává hodnoty v jednotlivých prostorových bodech.

Další důležitým objektem je *Feature* (viz obr. 4-6). Tento objekt reprezentuje entitu prostředí. Definuje například hranici lesa, silnice, různé překážky. Nedefinuje význam, ale spíše strukturu daného objektu. Jeho struktura připomíná strukturu objektu *Geometry*. Často jsou také tyto objekty vzájemně propojeny.



Obrázek 4-11 - Objekt Feature a jeho reprezentace.

Nastínili jsme hlavní objekty a vztahy mezi těmito objekty. Toto je pouze výčet těch nejdůležitějších. V aktuální verzi Sedrisu (verze 3.1.2) existuje 352 typů objektů. Mezi ty, o kterých jsme se nezmínili, patří například topologie, transformace, barva, pozice, kontrolní bod, textura, čas, světlo.

4.2 Specifikace kódování dat přírodního prostředí – EDCS

Specifikace kódování dat zajišťuje klasifikaci objektů a vlastností všech možných částí přírodního prostředí. Návrh tohoto kódování je složen ze tří částí.

- Popisuje **klasifikaci**, která definuje typ objektů, vyskytujících se v prostředí. Definuje objekty jako jsou například mosty, budovy, mraky, stromy, auta.
- Popisuje **atributy**, které definují stav daných objektů, vyskytujících se v prostředí. Popisují vlastnosti jako je například velikost, barva, teplota, slanost, vlhkost,
- Definuje **výčet** různých hodnot daných vlastností objektu a jednotky různých **fyzikálních veličin**. Definuje například různé hodnoty barev – červenou, modrou, černou, atd. Definuje jednotky délky, teploty, tlaku, apod.

Specifikace těchto objektů je tvořena 9 slovníky. Jednotlivé záznamy těchto slovníků reprezentují vždy jednu určitou klasifikaci, popř. atribut, výčet, či fyzikální veličinu. Každý tento záznam je tvořen z názvu, kódu, definice vlastností a odkazů na jiné související záznamy. Každý slovník obsahuje vlastnosti, které jsou si svým způsobem blízké. Mezi slovníky patří například slovník klasifikací, slovník atributů, slovník metadat, slovník výčtů typů a vlastností, slovník jednotek měřítek, apod.

4.3 Sedris – shrnutí

Sedris je datový formát, ve kterém můžeme popsat terén prakticky jakýmkoliv způsobem. Ač se to zdá divné, při použitelnosti může být jeho slabostí právě veliká obecnost přístupu k věci. Používá se zde sice přehledná, ale poněkud robustní hierarchie objektů. Tento datový formát je proto pro specializované programy až příliš složitý.

V dokumentaci je sice napsáno, že je kladen důraz na rychlost čtených dat ze souboru, nicméně podle našich zkušeností tento formát nemůžeme použít v real-time aplikacích, kde potřebujeme získávat rozsáhlá data za běhu programu. Pro tyto aplikace je třeba data transformovat do jiného datového formátu.

Při práci s API zjistíme, že se zde vyskytují určité drobné odlišnosti od věcí uvedených v dokumentaci, což poněkud znepříjemňuje programování. Podpora uživatele – programátora není nijak vysoká. Chybí tutoriály, k dispozici máme jen referenční příručku a strohý manuál, kde se žádné příklady nevyskytují. Naštěstí jsou od všech sedrisovských programů k dispozici zdrojové kódy, a tak můžeme nahlédnout přímo do nich. Funkce pro práci s daty jsou navrženy poměrně kvalitně. Jednoduchým způsobem se zde můžeme přesouvat po objektech a jejich vazbách. Existuje zde systém maker, pomocí nichž můžeme vyhledat jakékoliv objekty, aniž bychom znali jejich pozici v hierarchii.

Tento formát dokáže uchovávat všechny datové struktury zmíněné v předchozí kapitole. Nicméně vzhledem k jeho nepoužitelnosti v reálném čase se jím nadále nebudeme podrobněji zabývat.

5 Jiné datové formáty

OpenFlight – viz [16, 17]

Tento formát byl vytvořen společností MultiGen speciálně pro účely zobrazování scény. Je to nejvíce používaný formát pro vizuální simulaci. K tomuto formátu je vyvinuto speciální API, pomocí něhož pracujeme s daty. Zatímco jiné formáty umožňují uchovávat charakter objektů, či jiné informace, OpenFlight se zaměřuje pouze na informace důležité pro zobrazení. Nabízí věci jako úroveň detailů, obalová tělesa, priority vykreslování částí scény, bsp stromy, hierarchické uložení scény.

CTDB – Compact Terrain Database – viz [7]

Tento formát pro popis terénu je používaný vojenskými simulačními programy (ModSAF, OneSAF, JointSAF). Je navržen pro používání v reálném čase. Je tedy optimalizován na rychlost čtených dat. Téměř pro vše je zde daný předpis, podle kterého můžeme daná data reprezentovat. Například při popisu výškové mapy pomocí pravidelné mřížky máme předepsáno v jakých rozestupech budou data uložena, do jak velkých bloků se budou data ukládat. Jsou popsané atributy, které se o daném povrchu ukládají. Vše je zaměřeno na použití ve vojenství. Velký důraz je daný na popis sítě cest, silnic a vodních toků. CTDB unifikuje popis všech modelů. Tento formát nám nedává příliš velkou volnost při popisování.

VRML – Virtual Reality Modelling Language – viz [13]

Tento jazyk byl vytvořen pro popis 3D světů na internetu. VRML nabízí pro popis všechny standardní věci. Geometrická primitiva, výškové mapy, mapování textur, osvětlení, úroveň detailů, hierarchické struktury. VRML nabízí další věci, kterou mnoho formátů nepodporuje – interakce scény s uživatelem, animace ve scéně, skriptování různých složitých úkonů v java skriptu, vytváření vizuálních efektů. Nabízí prohlížení různých světů, jejichž části mohou být umístěny kdekoliv na internetu. Scéna je zadána v textovém formátu. Pro vytvoření scény není potřeba žádných modelovacích programů. Tento jazyk je poměrně snadno pochopitelný. Proti VRML je Sedris určen čistě ke sdílení dat, ne k interakci s uživatelem. Sedris také nabízí daleko více typů objektů pro definici scény.

GeoVRML – viz [20]

GeoVRML je jazyk vytvořený pro reprezentaci a vizualizaci geografických dat. Tento jazyk je založen na standartu VRML 97. Definuje nové objekty pro popis geografických dat.

Souřadnice objektů mohou být zadané pomocí zeměpisných šířek a délek. Pozice ve scéně mohou být udány s větší přesností než jeden milimetr. Poskytuje větší kontrolu nad přenosem rozsáhlých geografických objektů po síti k cílovému uživateli. Je plně podporována práce s metadaty objektů. Existuje zde možnost dotazovat se na umístění jednotlivých referenčních bodů. Poskytuje základní podporu navigace specifické pro geografické aplikace.

Existuje mnoho jiných formátů, které popisují terén a objekty s ním spojené. Většina z nich nabízí omezené prostředky pro popis. Jsou většinou šité na míru nějaké aplikaci, případně určené pro určitý druh použití. Sedris je unikátní tím, že popisuje data ze všech možných pohledů, nabízí široké spektrum možností, jak k daným datům přistupovat.

6 Techniky modelování povrchu terénu

V této kapitole se budeme věnovat různým metodám automatického modelování náhodného virtuálního povrchu. Dále představíme algoritmy na vytváření nepravidelné trojúhelníkové sítě.

6.1 Generování povrchu virtuálního terénu

V této kapitole se budeme zabývat tím, jak vygenerovat výškovou mapu náhodného virtuálního terénu. Existuje více způsobů, jak se zhostit tohoto problému. Nebudeme zde popisovat jednotlivé algoritmy, ale pouze typy těchto algoritmů. Metody jsou popsány v [2].

Výšková mapa jako obrázek

Výškovou mapu můžeme reprezentovat pomocí nějaké bitmapy. Tuto bitmapu můžeme ručně vyrobit v nějakém kreslícím nástroji. V takto generovaném terénu jsme limitování tím, co dokážeme nakreslit.

Rekurzivní dělení výškové mapy

Nejpopulárnější metodou generování povrchu terénu je právě rekurzivní dělení. Myšlenka této metody je následující: vezmeme mřížku, kterou chceme vygenerovat (nemusí se jednat přímo o čtvercovou mřížku) a nastavíme výšku v rohových bodech na nějakou náhodnou hodnotu. Poté vypočítáme výšku ve středu každé strany a v centru - pomocí lineární interpolace dříve nastavených bodů a přidáním nebo ubráním nějaké náhodné veličiny. Dále rozdělíme mřížku podle centrálního bodu a algoritmus dále iterujeme. V každé úrovni přidáváme náhodné hodnoty vždy z menšího intervalu. Existuje mnoho algoritmů, které si tento způsob lehce modifikují, nicméně myšlenka zůstává stejná.

Pokud bychom iterativním procesem postupovali do nekonečna, získali bychom fraktál. Algoritmy založené na této metodě jsou většinou extrémně rychlé, zvláště pokud je velikost mřížky mocnina dvou. Tyto metody jsou založené na pravidelném dělení, a proto se zde mohou vyskytovat nepříjemné artefakty, zvláště podél diagonál a podél os mřížky. Můžeme zde mít problémy při spojování více bloků, které jsou tímto způsobem vygenerovány. Takto vytvořený terén není hladký, vyskytují se zde ostré nerovnosti. Doporučuje se proto terén na závěr vyhladit.

Složení funkcí

V tomto případě je terén popsán dvourozměrnou spojitou funkcí $f(x, y) = z$, kde x, y jsou body na povrchu terénu a z určuje výšku. Používají se různé funkce. Kvadratické, kubické, spline funkce, složení goniometrických funkcí, apod.

Jaké jsou výhody a nevýhody této metody? Můžeme kdykoliv získat výšku kteréhokoliv bodu nezávisle na ostatních bodech. Se správnou funkcí můžeme rozšířit velikost terénu do nekonečna. Terén je většinou hladký, není třeba dále vyhlazovat. Máme větší kontrolu nad různými jevy vyskytujícími se v krajině. Hlavní nevýhodou této metody je rychlost, neboť zabere daleko více času než ostatní metody. Můžeme nechtěně docílit opakování vzorku terénu.

Rozdělení a zvýšení

Mějme na začátku rovnou plochu. Vezmeme nějakou křivku, podle které rozdělíme plochu na dvě části, a poté jedné z těchto částí zvedneme nebo ubereme výšku. Tento krok mnohonásobně zopakujeme (například tisíckrát). Jako rozdělovací hranice většinou používáme přímky, nicméně můžeme použít kružnice, obdélníky, či jiné útvary. Tato metoda je časově velice náročná, nicméně generuje atraktivní terény.

Fyzikálně založené metody generování

Povrch Země vytvářejí dva základní procesy. Tektonické pochody, které posouvají litosférické desky, a eroze způsobená vodou a větrem. Pomocí simulace těchto procesů můžeme vytvořit realistický terén. Tato simulace nicméně trvá velice dlouho, neboť musíme simulovat procesy trvající bilióny let. Výhodou této metody je možnost generování řek. Předchozí metody nebraly v úvahu říční toky. Jsou zde dvě základní metody, jak generovat řeky: simulovat erozi terénu, nebo navrhnout říční síť a poté generovat terén podél této sítě.

Filtrování dat

Po vygenerování mřížky nějakou předchozí metodou zpravidla data vyhladíme digitálním filtrem, abychom se zbavili ostrých přechodů. Tímto způsobem můžeme také generovat terén přímo tím, že naplníme mřížku náhodnými výškami a poté vyhladíme nerovnosti filtrem.

6.2 Vytvoření nepravidelné trojúhelníkové sítě

Předpokládejme, že máme hustou síť bodů. Z této sítě chceme vytvořit nepravidelnou trojúhelníkovou síť. Nejprve musíme vybrat určitou část těchto bodů, které budou tvořit reprezentativní vzorek. Poté tyto body spojíme do trojúhelníků a tím vytvoříme síť.

6.2.1 Výběr reprezentativních bodů terénu

Následující algoritmy vybírají body v důležitých zlomových místech terénu. (Tyto útvary jsou v normálním terénu běžné, chybějí ale v terénech popsaných hladkými matematickými funkcemi.)

Fowlerův a Littlův algoritmus (viz [10])

Tento algoritmus je založen na bodech, které hrají významnou roli v povrchu terénu. Jde o lokálně nejvyšší vrcholky, nejnižší body, body tvořící hřebeny a údolí. Algoritmus potřebuje na vstupu hustou síť bodů terénu tvořící pravidelnou mřížku.

Postup:

- Prozkoumáme každou submatici velikosti 3x3. V každém kroku se tedy díváme na 9 bodů uspořádaných do pravidelné mřížky.
    - Označíme 8 sousedů centrálního bodu znaménkem + nebo – podle toho, zda jsou vyšší nebo nižší než centrální bod.
    - Daný centrální bod označíme jako vrchol, pokud všichni jeho sousedé jsou níže.
    - Daný centrální bod označíme jako prohlubeň, pokud všichni jeho sousedé jsou výše.
- Nyní použijeme matici velikosti 2x2, abychom rozpoznali hřebeny, případně žlaby.
    - Bod může tvořit hřeben, pokud je v submatici 2x2 nejvyšším bodem.
    - Bod může tvořit žlab, pokud je v submatici 2x2 nejnižším bodem.
- Algoritmus prochází od nejvyšších (nejnižších) bodů po hřebenech (žlabech) tím, že spojuje potencionální sousedy hřebenových (žlabových) bodů. Prochází body tak dlouho, dokud nedojde od vrcholu k prohlubni, případně naopak.

Výsledkem tohoto procesu je propojená síť vrcholů, prohlubní, hřebenů a žlabů. Doporučuje se množinu bodů tvořící dané hřebeny (žlaby) zredukovat, tak aby výsledná množina reprezentovala rozumnou křivku. Dále je vhodné přidat body, které sice neleží na žádných výše zmiňovaných křivkách, ale které mohou zredukovat rozdíly mezi reálnou krajinou a námi modelovanou krajinou. Tyto body by měly být na nějakých zlomových místech svahů.

Tento algoritmus pracuje nejlépe na terénech, kde jsou dobře odlišené hřebeny a údolí. Často je třeba závěrečné ruční doladění výsledných bodů.

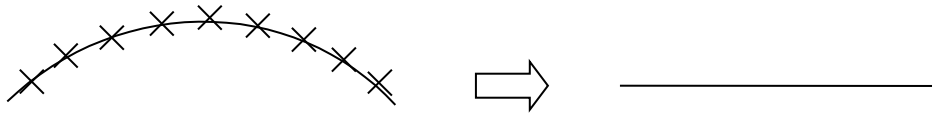
VIP (Very Important Points) algoritmus (viz [10])

Vstupem algoritmu je též hustá síť bodů tvořící pravidelnou mřížku. Tento algoritmus eliminuje nevýznamné body povrchu. Opět budeme zkoumat submatice velikosti 3x3.

Postup:

- V každé submatici velikosti 3x3 zjistíme důležitost centrálního bodu následujícím způsobem:
    - Vezmeme 8 sousedů centrálního bodu a spárujeme dvojice ležící proti sobě. Spárujeme tedy horní a dolní bod, levý a pravý, atd.
    - Tyto páry spojíme úsečkou a spočítáme kolmou vzdálenost centrálního bodu od této úsečky.
    - Zprůměrujeme výsledné 4 vzdálenosti, a tím získáme významnost tohoto bodu.
- Mažeme body mřížky v pořadí od nejnižší významnosti k nejvyšší, dokud nedosáhneme předurčeného počtu bodů, případně dokud minimální významnost nedosáhne určité kritické hranice.

Tento algoritmus využívá rovných čar a ploch vyskytujících se v terénu. Díky tomu je méně úspěšný na lehce zakřivené křivky (plochy), neboť z nich vyrábí rovné útvary (jak je znázorněno na obrázku 6-1).



Obrázek 6-12 - Lehce zakřivené plochy přechází do rovin.

6.2.2 Triangulace bodů terénu

Mějme množinu bodů reprezentující terén. V této kapitole představíme dva algoritmy na vytvoření nepravidelné trojúhelníkové sítě z těchto bodů. Budeme vždy chtít, aby výsledné trojúhelníky byly co „nejtlustší“, tj. trojúhelníky, jejichž každý úhel se blíží velikosti 60 stupňů. To zajišťuje, že zde nebudou existovat body příliš vzdálené vrcholům trojúhelníku, ve kterém daný bod leží. Vhodným kritériem může být například poměr opsané a vepsané kružnice trojúhelníka.

Stavění podle vzdálenosti (viz [10])

Postup:

- Spočítáme vzdálenost všech dvojic bodů a uspořádáme tyto dvojice podle jejich vzdálenosti od nejmenší k největší.
- Spojíme nejbližší dva body, jejichž spojnice neprotíná již dříve definované spojnice.
- Tento postup opakujeme, dokud můžeme vybrat další spojnici.

Po aplikaci tohoto postupu budou všechny body spojeny do trojúhelníků. Tento algoritmus produkuje mnoho „hubených“ trojúhelníků, a proto většina aplikací používá druhý algoritmus.

Delaunayho triangulace (viz [1])

Síť trojúhelníků tvoří Delaunayho triangulaci právě tehdy, když pro každý trojúhelník platí:

- Kružnice opsaná trojúhelníku neobsahuje žádné další body.

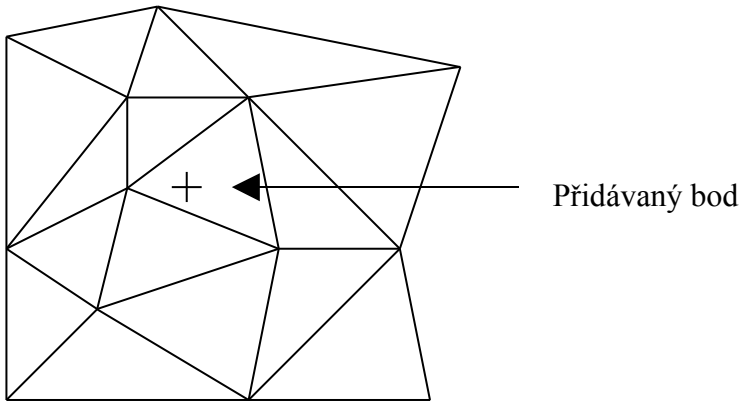
Jiný způsob, jak definovat Delaunayho triangulaci, je následující:

- Rozdělíme celou plochu na části tak, že každá část bude obsahovat jeden reprezentativní bod prostoru a k tomu všechny body plochy, které mají k danému bodu nejblíže ze všech.
- Hranice vytvořené v tomto procesu tvoří Voronoiho diagram.
- Dva vrcholy v Delaunayho triangulaci jsou spojené, když jejich Voronoiho polygony jsou spojené hranicí.

Voronoiho diagram a Delaunayho triangulace jsou navzájem duální grafy.

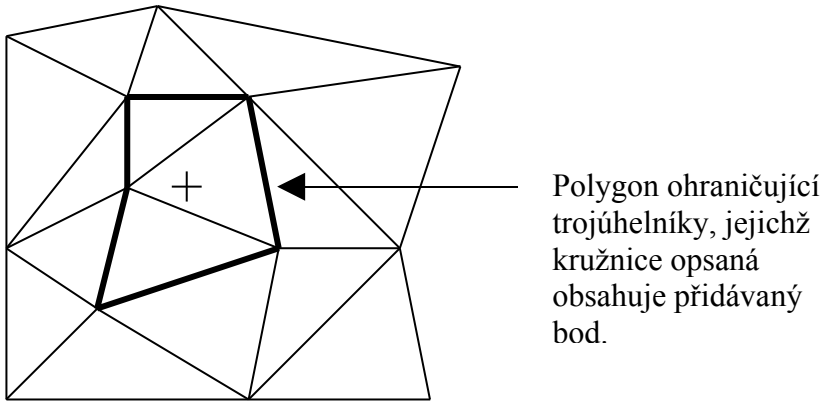
Popíšeme algoritmus na vytvoření Delaunayho triangulace.

- Vytvoříme velký fiktivní trojúhelník obsahující všechny body terénu. Tento trojúhelník bude tvořit počáteční triangulaci terénu.
- Vezmeme jakýkoliv bod (viz obr. 6-2) a přidáme ho do stávající trojúhelníkové sítě následujícím způsobem:



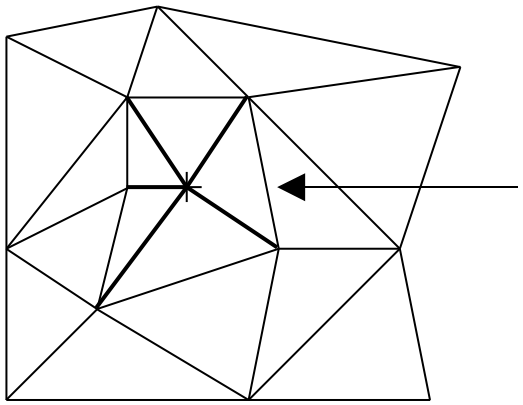
Obrázek 6-13 - Bod, přidávaný do triangulované sítě (obrázek viz [1]).

- Označíme všechny trojúhelníky, jejichž kružnice opsaná obsahuje námi přidávaný bod (viz obr. 6-3).
- Vnější strany těchto trojúhelníků (ty, které nesousedí z žádným z takto ohraničených trojúhelníků) tvoří tzv. ohraničující polygon.



Obrázek 6-14 - Ukazuje trojúhelníky, které změní topologii sítě (obrázek viz [1]).

- Smažeme všechny trojúhelníky v ohraničujícím polygonu a přidáme trojúhelníky, kde každý je tvořen jednou hranou polygonu a nově přidávaným bodem (viz obr. 6-4).



Obrázek 6-15 - Nově ztriangulovaná část sítě (obrázek viz [1]).

- Předchozí krok opakujeme tak dlouho, dokud nám zbývají volné body – body nezařazené do trojúhelníkové sítě.
- Na závěr smažeme počáteční velký fiktivní trojúhelník a s ním všechny trojúhelníky, které jsou tvořeny pomocí bodů počátečního trojúhelníku.

Tento algoritmus relativně rychle vytváří „hezké“ trojúhelníkové sítě.

7 Techniky zobrazování

Zobrazovací algoritmy jsou hlavní částí celého programu. Na nich nejvíce záleží, zda bude program dostatečně rychlý. Většina algoritmů používá jako vykreslovací primitiva trojúhelníky. Tyto trojúhelníky můžeme mít zadány jako nějakou množinu, případně v nějaké hierarchické struktuře. Tuto strukturu se pak snažíme co nejvíce zredukovat, abychom zobrazovali pouze to, co je vidět. Valná část algoritmů má na vstupu pravidelnou mřížku výškových bodů. Zde je pak hlavním úkolem triangulovat scénu v reálném čase, správně aproximovat povrch terénu. Existují algoritmy, jejichž primitiva nejsou tvořena trojúhelníky, ale zde je nebudeme popisovat. Zabývejme se nyní tím, jak správně zredukovat množinu polygonů, kterou chceme zobrazit. V další části se budeme věnovat dynamickému generování LOD trojúhelníkových sítí v reálném čase.

7.1 Optimalizace

Zabývejme se nyní technikami, jak optimalizovat zobrazování scény. Mějme rozsáhlý terén. Pokud se v něm pohybujeme, neuvidíme nikdy celý terén najednou, a je proto zbytečné vykreslovat všechny trojúhelníky scény. Otázkou je, jak zjistit, které trojúhelníky nepotřebujeme zobrazit. Tento problém je popsán v [11].

7.1.1 Poloha objektu v zorném jehlanu - view frustum culling

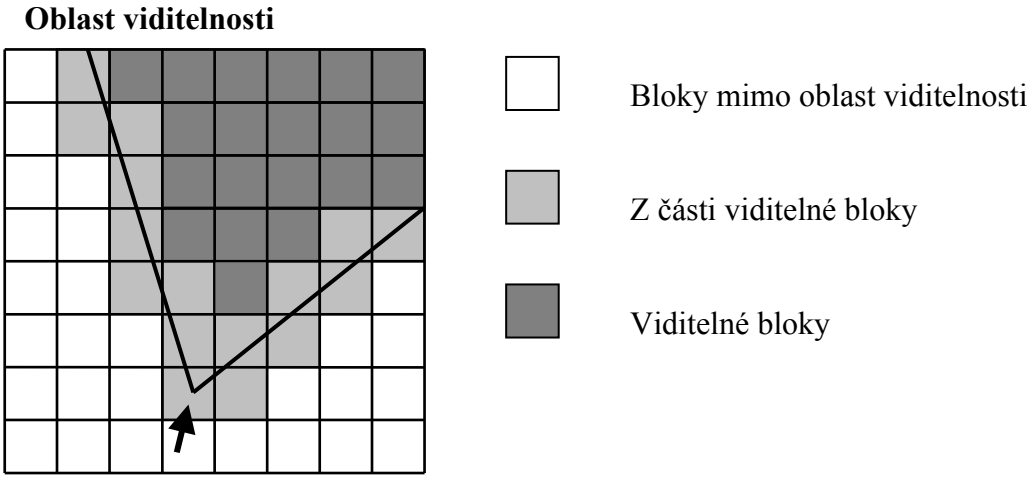
Jaký je geometrický útvar, který reprezentuje prostor našeho vidění? Pokud si představíme plochu monitoru jako okno, kterým se díváme na terén, je tento geometrický útvar reprezentovaný čtyřhranným nekonečným jehlanem, který má počátek v našem oku, a jeho stěny jsou definovány hranami obrazovky. Je tedy třeba otestovat, jaká část scény spadá do tohoto objektu a jaká část je mimo.

Obalová tělesa

Jednoduchým testem bychom mohli otestovat každý polygon ve scéně, zda spadá do našeho zobrazovaného prostoru. Testování každého polygonu je však procesorově náročné. Daleko lepší je rozdělit scénu do určitých částí, kde každou část můžeme ohraničit jednoduchým geometrickým objektem – kvádrem, koulí. Dejme tomu, že máme část scény ohraničenou koulí. Poté při zobrazování stačí zjistit, zda vidíme kouli celou, její část, nebo je tato koule

celá mimo prostor zobrazování. Pouze pokud vidíme část koule, je nutné dále testovat všechny polygony na viditelnost.

Při zobrazování rozsáhlých polygonů máme často terén rozdělený do bloků čtvercové velikosti. Každý blok můžeme jednoduše ohraničit kvádrem. Způsob ořezávání daných kvádrů je vidět na obrázku 7-1.



Obrázek 7-16 - Způsob ořezávání pomocí zorného jehlanu.

Je vhodné uspořádat obalová tělesa do hierarchie podle inkluze. Pak můžeme jednoduše při průseku tohoto tělesa s naším geometrickým útvarem viditelnosti přejít v hierarchickém stromě o úroveň níže a znovu zkoumat viditelnost.

7.1.2 Odstranění odvrácených ploch – backface culling

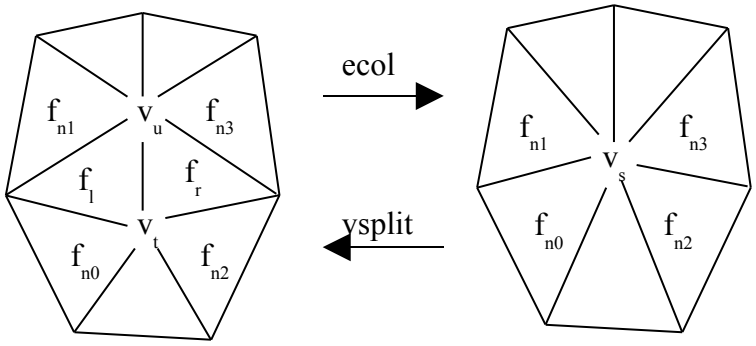
Mějme nějaký polygon určující část povrchu terénu. Pozorovatel vždy může vidět pouze jednu (přední) stranu tohoto polygonu. Druhá strana polygonu směřuje směrem do země. Nabízí se tedy otestovat každý polygon, případně části scény, zda můžeme vidět jeho přední stranu. Podíváme se na normálu daného polygonu. Pokud směřuje ve směru od kamery, pak polygon nevidíme a nevykreslujeme jej.

7.1.3 Úroveň detailů – LOD (Level of Detail)

Zobrazování v různých úrovních detailů je důležitý mechanismus pro rychlé zobrazování. Princip tohoto mechanismu byl již popsán v kapitole zabývajícími se datovými strukturami (viz 3.1.5) a bude dále rozebrán v následujících kapitolách.

7.2 Zobrazování nepravidelné trojúhelníkové sítě pomocí LOD

Tento algoritmus je podrobně popsán v [5]. Mějme nepravidelnou trojúhelníkovou síť $\widehat{M}$. Nejprve musíme vytvořit jednotlivé úrovně LOD pro zobrazení nezávisle na pozici pozorovatele. Síť postupně zjednodušujeme odebíráním hran a nahrazováním koncových vrcholů těchto hran jedním vrcholem tak, jak ukazuje obrázek 7-2.

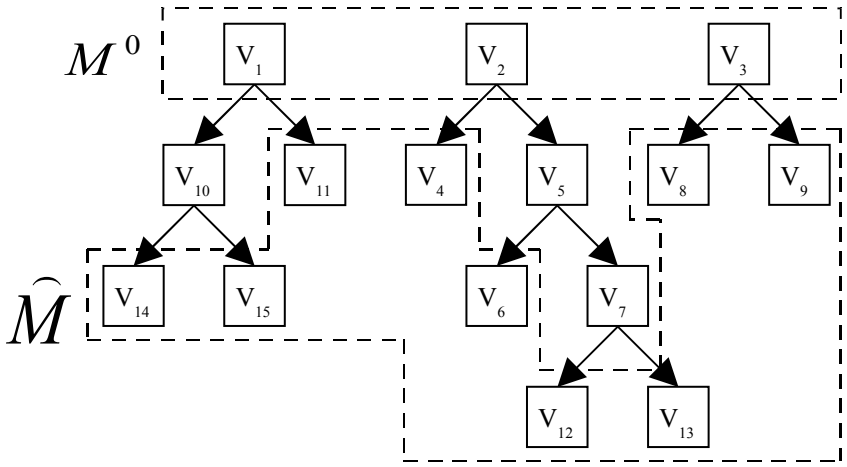


Obrázek 7-17 - Odebírání a přidávání vrcholu do sítě (obrázek viz [5]).

Tím, jak odebírat hrany, abychom zachovali geometrii terénu, se v tomto textu nebudeme zabývat. Vše je podrobněji vysvětleno v [4]. Postupným odebíráním *n* hran dostáváme posloupnost sítí, kde každá síť je zjednodušením předchozí sítě. Takto dostaneme různé stupně LOD, které bychom mohli použít, pokud bychom chtěli mít scénu zjemněnou ve všech bodech stejně.

$$\widehat{M} = M^n \xrightarrow{ecol_{n-1}} M^{n-1} \ldots \xrightarrow{ecol_1} M^1 \xrightarrow{ecol_0} M^0$$

M^0 je maximálně zjednodušená síť. $ecol_i$ značí odebrání i-té hrany tzv. edge collapse. Odebrání hrany $ecol(v_s, v_t, v_u, f_l, f_r, f_{n0}, f_{n1}, f_{n2}, f_{n3})$ zruší vrcholy v_u a v_t a přidá vrchol v_s. Navíc odebere dva trojúhelníky f_l a f_r. Ke každému odebrání hrany můžeme definovat inverzní transformaci rozdělení vrcholu (*vsplit – vertex splitting*). Rozdělení vrcholu $vsplit(v_s, v_t, v_u \ldots)$ nahradí vrchol v_s dvěma vrcholy v_u a v_t. Tímto vzniknou dva nové trojúhelníky f_l a f_r.



Obrázek 7-18 - Hierarchie vrcholů (obrázek viz [5]).

Při pohledu na obrázek 7-3 nás může napadnout, zda při zjednodušování síťě musíme vždy zachovávat pořadí odebírání hran. Při dodržování určitých pravidel nemusíme toto pořadí dodržovat. Označme vrchol nebo trojúhelník jako aktivní, pokud se vyskytuje v síti, se kterou právě pracujeme. Definujme nyní pravidla, při jejichž dodržování nemusíme zachovávat pořadí odebírání hran, příp. rozdělování vrcholů.

- Rozdělení vrcholu - $vsplit(v_s, v_t, v_u ...)$ - je povoleno, pokud je vrchol v_s aktivní a pokud trojúhelníky f_{n0}, f_{n1}, f_{n2}, f_{n3} jsou aktivní.
- Odebrání hrany - $ecol(v_s, v_t, v_u ...)$ - je povoleno, pokud jsou oba vrcholy v_u a v_t aktivní a pokud trojúhelníky sousedící s f_l a f_r jsou f_{n0}, f_{n1}, f_{n2}, f_{n3} tak jak ukazuje obrázek 7-2.

Zjemňovací kritéria

V tomto odstavci popíšeme funkci $qrefine(v_s)$, která rozhodne, zda daný vrchol rozdělit na základě aktuálního pohledu.

```
function qrefine(vs)
      // rozděl pouze pokud vrchol zasahuje do viditelné části
      if outside_view_frustum(vs) return false
      // rozděl pouze pokud horní strana ovlivněných trojúhelníků bude
      // vidět pozorovatelem
      if oriented away(vs) return false
      // rozděl pouze pokud by scéna byla pro zobrazení příliš hrubá
      if screen_space_error(vs) < τ  return false
      return true
```

Kritéria:

- **Viditelnost vrcholu**: pro každý vrchol spočteme sféru jeho vlivu. Jde o oblast, kde daný vrchol ještě ovlivňuje geometrii scény, a je proto nutné s ním počítat. Jedná se vlastně o prostor všech přilehlých trojúhelníků k danému bodu. Tuto oblast aproximujeme koulí o poloměru r. Nejprve spočteme tento poloměr pro vrcholy, které již dále nerozdělujeme (listové vrcholy v hierarchii uzlů). Poté rekurzivně počítáme poloměry ohraničujících koulí pro ostatní uzly jako nejmenší poloměry koulí, které zahrnují sféry vlivu potomků i samotného zkoumaného vrcholu.
- **Orientace povrchu vzhledem k pozorovateli:** toto kritérium říká, abychom nezjemňovali povrch, který je k nám obrácen „zády“. Toto přiblížení je analogické předchozímu, jen zde jako sféru vlivu uvažujeme prostor normál daného povrchu. Tento prostor budeme ohraničovat kuželem, definovaným pomocí normály a úhlu, o který můžeme normálu odchýlit. Konstrukce je analogická jako v předchozím kritériu. Nejprve spočteme dané kužele pro listové vrcholy naší hierarchie. Spočítáme normály všech přilehlých trojúhelníků a z nich spočítáme obalový kužel. Poté stejným způsobem spočteme kužele uzlových vrcholů s tím, že musí zahrnovat i kužele svých synů.
- **Velikost chyby při projekci**: cílem třetího kritéria je přizpůsobit síť tak, aby projekce této sítě vypadala přibližně stejně, jako projekce plné sítě. Pro každý vrchol tedy zkoumejme, jak se jeho okolí liší od stejného okolí v plně rozvinuté síti. Používají se dvě hlavní měřítka: První měřítko hledá nejmenší r takové, že každý bod v okolí zkoumaného vrcholu je vzdálen maximálně o r od plné sítě. Druhé měřítko předpokládá, že chyba povrchu se projeví nejvíce ve směru kolmém na povrch. Toto druhé měřítko udává pro každý vrchol hodnotu δ , což je hodnota, o kterou se výsledná hustá síť může lišit ve směru kolmém na povrch. Spojením těchto dvou měřítek získáme formuli, pomocí které můžeme jednoduše určit chybu pro každý vrchol.

  $$err = \max(r, \delta \left\| \hat{n} \times \vec{v} \right\|)$$

  $\hat{n}$ je normála povrchu daného vrcholu, $\vec{v}$ je směr ve kterém se díváme.

Hodnoty všech tří kritérií můžeme předpočítat ihned po nahrání scény.

Algoritmus pro přizpůsobení trojúhelníkové sítě k vykreslení

```
procedure adapt_refinement()
      for each v ∈ V
            if v ≠ leaf and qrefine(v)
                  force_vsplit(v)
            else if v ≠ root    and ecol_legal(v.parent)
                                and not qrefine(v.parent)
                  ecol(v.parent)

procedure force_vsplit(v')
      stack ← v'
      while v ← stack.top()
            if v ≠ leaf and v.fl ∈ F  // F - množina trojúhelníků, viz obr. 7-2
                  stack.pop() // vrchol byl rozdělen v předchozích krocích
            else if v ∉ V  // nejprve musíme rozdělit otce
                  stack.push(v.parent)
            else if vsplit_legal(v)
                  stack.pop()
                  vsplit(v)
            else for i ∈ {0..3}
                  if v.fn[i] ∉ F
                        // rozdělíme vrcholy, které vytvářejí
                        // trojúhelníky f0..f3, viz obr. 7-2
                        stack.push(v.fn[i].vertices[0].parent)
```

Algoritmus prochází všemi aktivními vrcholy (množina V). Pro každý takový vrchol spočítá rozhodovací funkci pro zjemnění. Pokud se má vrchol rozdělit, ale toto rozdělení není povoleno, musí se vykonat rozdělení jiných vrcholů tak, abychom mohli provést rozdělení našeho vrcholu povoleným způsobem. To zaručuje funkce `force_split`. Pokud zjemňovací funkce nedoporučí rozdělit vrchol, zjistíme, zda nemáme povrch zjednodušit. Podíváme se, co nám vrátí rozhodovací funkce pro otce našeho vrcholu. Pokud nedoporučí otce rozdělit, musíme náš vrchol a jeho souseda nahradit otcem. Odstraníme tedy jednu hranu, pokud je to povoleno.

Po odstranění hran, či přidání vrcholů, se nové vrcholy mohou dále transformovat. Proto je důležité přidat je do seznamu aktivních vrcholů a podívat se na ně pomocí zjemňovacího kritéria.

Vykreslování

Abychom urychlili vykreslování trojúhelníků, je třeba je skládat do pásů. Algoritmus prochází seznamem trojúhelníků a v každém ještě nevykresleném definuje nový pás. Vykreslí trojúhelník, podívá se zda má nějakého souseda a pokud ano, pokračuje pás tímto sousedem. Abychom se vyhnuli velké fragmentaci, vybíráme sousedy vždy v levotočivém, nebo v pravotočivém stylu.

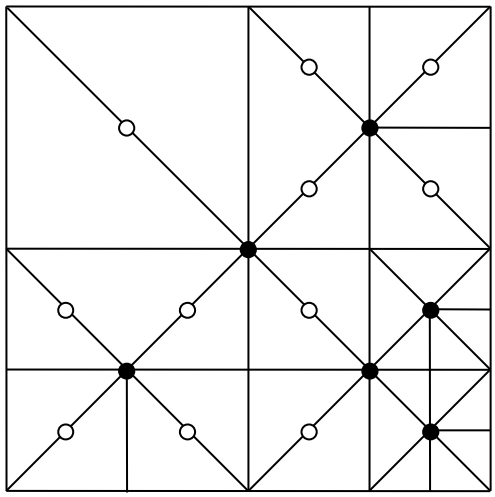
7.3 Zobrazení pravidelné mřížky výškových bodů pomocí LOD

Mějme scénu zadanou pomocí pravidelné mřížky výškových bodů. Zabývejme se nyní triangulací této mřížky v reálném čase (podrobněji popsáno v [12]). Pokud bychom k triangulaci použili všechny body mřížky, dostali bychom zbytečně vysoký počet trojúhelníků. Přitom stejného efektu zobrazení dosáhneme aproximací různých míst terénu. Budeme brát v úvahu hrubost terénu s tím, že hrubší terén je aproximován větším počtem polygonů než hladší terén. Dále vezmeme v úvahu, že vzdálenější terén bude zobrazen pomocí menšího počtu polygonů než blízký terén.

Pro jednoduchost předpokládejme, že mřížka má velikost stran $2^k + 1$. Pokud bude mít jinou velikost můžeme doplnit mřížku do požadované velikosti fiktivními body. Algoritmus bude používat pomocnou datovou strukturu – kvadrantový strom. Tento strom bude reprezentován dvourozměrnou maticí stejné velikosti jako má vstupní mřížka výškových bodů. Strom bude udávat, jak daný terén triangulovat. Obrázky 7-4 a 7-5 uvádí reprezentaci kvadrantového stromu a k němu způsob, jak pomocí ní triangulovat scénu.

$$\begin{pmatrix} ? & ? & ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & 0 & ? & 0 & ? \\ ? & ? & 0 & ? & ? & ? & 1 & ? & ? \\ ? & ? & ? & ? & ? & 0 & ? & 0 & ? \\ ? & ? & ? & ? & 1 & ? & ? & ? & ? \\ ? & 0 & ? & 0 & ? & 0 & ? & 1 & ? \\ ? & ? & 1 & ? & ? & ? & 1 & ? & ? \\ ? & 0 & ? & 0 & ? & 0 & ? & 1 & ? \\ ? & ? & ? & ? & ? & ? & ? & ? & ? \end{pmatrix}$$

Obrázek 7-19 - Datová struktura pro kvadrantový strom (obrázek viz [12]).

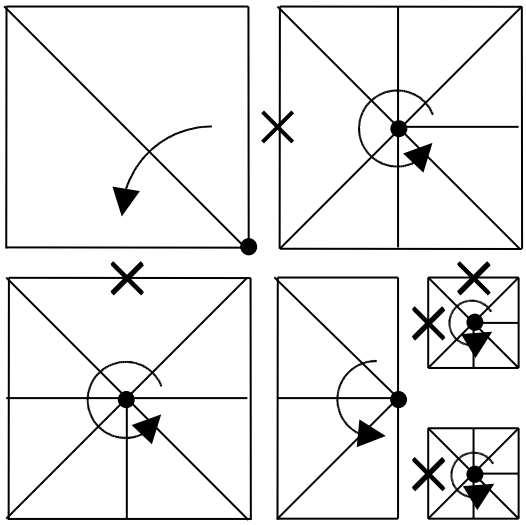


Obrázek 7-5 - Vykreslení stromu pomocí trojúhelníků (obrázek viz [12]).

Střed dvourozměrné matice (středový bod terénu) reprezentuje kořen kvadrantového stromu. Synové kořenu jsou reprezentovány středy submatic tvořených rozdělením matice pomocí kořenového uzlu. Nelistový uzel je reprezentován jedničkou, listový uzel nulou.

Zobrazení výškového pole pomocí kvadrantového stromu

Mějme již vygenerovaný kvadrantový strom (jak toho dosáhnout popíšeme v následujících odstavcích). Nyní budeme procházet daný strom do hloubky a vykreslovat pomocí něho scénu. Kdykoliv dojdeme k listu, vykreslíme trs trojúhelníků tak, jak ukazuje obrázek 7-6.



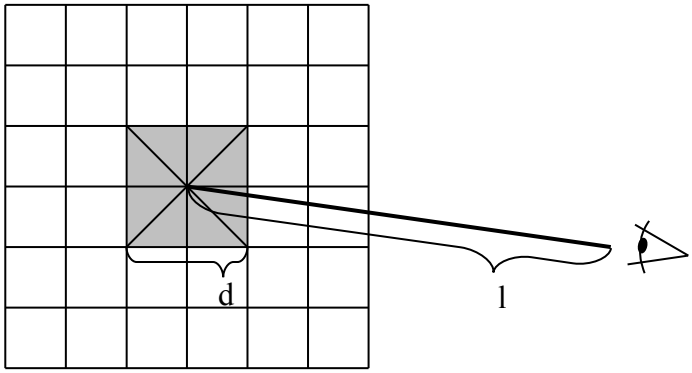
Obrázek 7-6 – Rekurzivně generované trsy trojúhelníků (obrázek viz [12]).

Abychom se vyvarovali T-vrcholů v místech, kde se setkávají sousední bloky různé úrovně, musíme přeskakovat centrální vrcholy – na obrázku označené křížkem. Tato metoda ovšem pracuje jen pokud je rozdíl úrovní jedna. Při vykreslování trsu trojúhelníků musíme znát úroveň sousedního bloku.

Vytvoření kvadrantového stromu

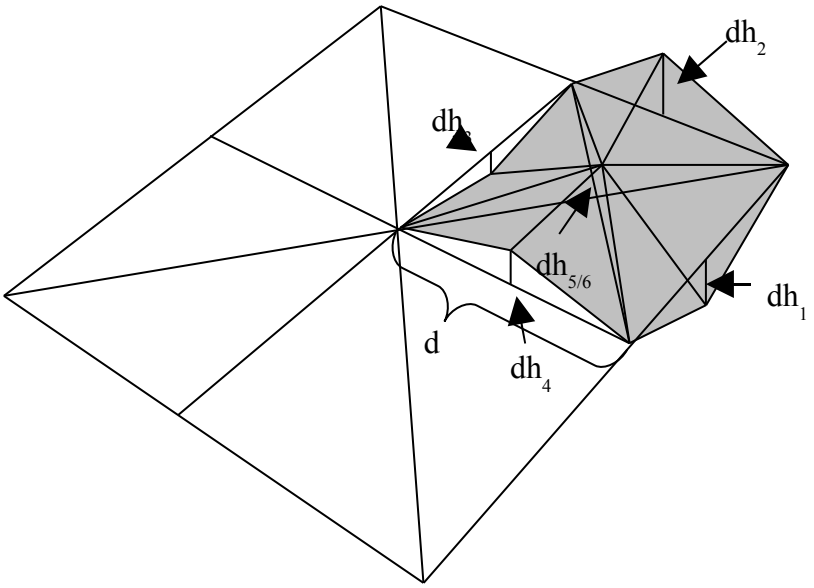
Zabývejme se nyní tím, jak vytvořit pomocnou matici kvadrantového stromu, jak určit triangulaci terénu. Budeme procházet strom do hloubky a v každém uzlu spočítáme, zda nám stačí reprezentace části terénu pomocí tohoto uzlu, nebo zda chceme terén polygonalizovat jemněji.

Mějme uzel reprezentující nějaký čtverec v terénu. Nyní nás zajímá, podle jakých kritérií se rozhodneme, zda-li blok dále dělit či nikoliv. Jako první kritérium vezmeme vzdálenost bloku od pozorovatele. Pokud bude vzdálenost příliš malá, daný blok reprezentovaný uzlem rozdělíme. Nechť d je šířka bloku a l je vzdálenost daného bloku od pozorovatele. Poté, pokud je splněna podmínka $l/d < C$, pak daný blok rozdělíme (viz obr. 7-7). Konstanta C reprezentuje minimální rozlišení terénu – to, které se použije, pokud bude terén plochý.



Obrázek 7-7 - Rozlišovací kritérium - vzdálenost versus velikost buňky (obrázek viz [12]).

Druhé kritérium bude založené na hrubosti terénu. Při přechodu bloku do nižší úrovně odstraníme 5 potencionálních chyb, jak ukazuje obrázek 7-8.



Obrázek 7-8 - Definice hrubosti terénu (obrázek viz [12]).

Druhé kritérium se zabývá právě těmito chybami. V každém podbloku rozdělovaného bloku spočteme maximální chybu, která se při nerozdělení může vyskytnout. Následně spočítáme tzv. povrchovou hrubost daného bloku *d2*.

$$d2 = \frac{1}{d}\max_{i=1..6}\left|dh_i\right|$$

Pokud bude povrchová hrubost příliš veliká, terén rozdělíme. Výsledné kritérium spojující obě dvě předchozí zní takto:

$$f = \frac{1}{d \cdot C \cdot \max(c \cdot d2,1)}$$ rozdělíme pokud f<1

Konstanta *C* opět určuje minimální rozlišení terénu, konstanta *c* určuje požadované rozlišení terénu. Pomocí změny požadovaného rozlišení můžeme při zobrazování v reálném čase dosáhnout konstantního počtu snímků za sekundu. Při rozdělování bloků je nutné dávat pozor na to, aby rozdíl úrovní sousedních bloků byl maximálně jedna. Pokud by toto nebylo splněno, je třeba sousední bloky dále rozdělit.

Vyskakování vrcholů

Pokud prolétáme nad terénem a terén zobrazujeme pomocí tohoto algoritmu, vyskytuje se nám zde nepříjemný efekt – tzv. vyskakování vrcholů (angl. vertex popping effect). Zaměřme se nyní na rozhodovací faktor *f* z předchozího odstavce. Pokud bude mít tento faktor pro nějaký uzel hodnotu v intervalu $[1/2,1]$, pak synové tohoto uzlu budou s velkou pravděpodobností listy (synové mají totiž 2x menší velikost buňky, a tím pádem při zachování stejné hrubosti terénu bude rozhodovací faktor 2x větší). Vyskakování vrcholů dochází právě v okamžiku, kdy se daný uzel mění z listového na nelistový a obráceně. Pro každý uzel si proto budeme pamatovat tzv. *splývající faktor* $b = 2(1 - f)$ v mezích $[0,1]$. Tento faktor nám říká, v jakém stádiu se vyskytuje uzel, u něhož pravděpodobně dojde k vyskakování vrcholů. Říká nám, jak interpolovat daný uzel při přechodu z listového uzlu na nelistový. V každém uzlu se může vyskytnout až 5 vrcholů, kde může dojít k tzv. „vyskočení“. Tyto vrcholy jsou aproximovány lineárně pomocí faktoru *b* mezi výškou jemnějšího bloku a výškou hrubšího bloku (který je průměrem dvou odpovídajících rohových vrcholů) .

Problémy se mohou vyskytnout v bodech na přechodech sousedních bloků, neboť jejich splývající faktor může být odlišný. V takovém výškovém bodě budeme brát v úvahu vždy menší hodnotu *splývajícího faktoru*.

Shrnutí

Tento algoritmus pracuje metodou shora dolů. Část terénu aproximujeme pomocí nějakého bloku. Pokud zjistíme, že tento blok nezobrazuje terén v dostatečném rozlišení, blok rozdělíme. Tímto způsobem se může samozřejmě stát, že při zobrazování scény se na některé

hodnoty mřížky vůbec nepodíváme i přesto, že mohou obsahovat hodnoty hodně vybočující z řady.

7.4 Vytváření sítě pomocí LOD - shrnutí

Existuje mnoho algoritmů, které generují trojúhelníkové sítě v reálném čase pomocí LOD. Myšlenka je však vždy stále stejná. Co nejvíce snížit počet polygonů. Některé algoritmy pracují metodou shora dolů. To znamená, že vezmou jednoduchou síť a tu podle potřeby zjemňují. Nevýhodou této metody je, že na některé vrcholy se při zobrazování scény vůbec nepodíváme. Přitom právě v těchto vrcholech mohou být lokální extrémy. Jiné algoritmy pracují metodou zdola nahoru. Vychází z velice jemné sítě, kterou postupně zjednodušují. Tato metoda je přesnější než předchozí. Další možností je pamatovat si aktuální síť a tu při pohybu pozorovatele zjemňovat, případně zjednodušovat. Algoritmus použití LOD závisí především na datové struktuře. Pokud budeme mít terén rozdělený do bloků a pro každý blok předpočítanou trojúhelníkovou síť v různých úrovních detailů, nemusíme složitě počítat LOD za běhu. Pro generování LOD v reálném čase se používají hierarchické datové struktury.

8 Prohlížeč terénů

Implementovali jsme prohlížeč, který umožňuje v reálném čase procházet po rozsáhlých virtuálních terénech a létat nad nimi ve vybraných výškách. Vstupní data měla být ve formátu Sedris. Nicméně v předchozí části jsme zjistili, že tento formát není vhodný pro zobrazování v reálném čase a proto si sedrisovská data transformujeme do našeho vlastního formátu.

Tato aplikace byla vytvořena pro platformu Windows pomocí vývojového nástroje Microsoft Visual C++ 6.0 a jeho knihoven MFC. Pro grafické zpracování bylo použito rozhraní OpenGL. Komunikaci s datovými soubory ve formátu Sedris zajišťoval soubor knihoven Sedris SDK Release 3.1.2.

8.1 Vstupní data ve formátu Sedris

Prohlížeč terénů podporuje **dva způsoby reprezentace vstupních dat:** nepravidelnou trojúhelníkovou síť a pravidelnou mřížku výškových bodů. Následuje přesnější vymezení:

- **Nepravidelná trojúhelníková síť:** terén je rozdělen do bloků (viz 3.1.4) a jednotlivé bloky jsou tvořeny nepravidelnou trojúhelníkovou sítí (viz 3.1.1). Podporovány jsou i jiné polygony než trojúhelníky. Ve scéně se mohou vyskytovat modely, které mohou být zadány v různých stupních LOD (viz 3.1.5). Modely jsou opět zadány pouze pomocí polygonů. Všechny polygony ve scéně mohou být buď obarveny nebo otexturovány. Barva může být definována buď pro celý polygon, nebo pro každý vrchol zvlášť.
- **Pravidelná mřížka výškových bodů:** terén je reprezentován pouze pomocí pravidelné mřížky výškových bodů (viz 3.1.2). Nejsou podporovány textury, neboť nebyly k dispozici v žádných dostupných vstupních datech. Terén není v sedrisovském souboru rozdělen na bloky ani nejsou definovány žádné modely.

Ukázkové terény jsou k dispozici přímo na stránkách projektu Sedris (viz [19]). Všechny tyto scény jsou v jednom z formátů popsaných v předchozím odstavci.

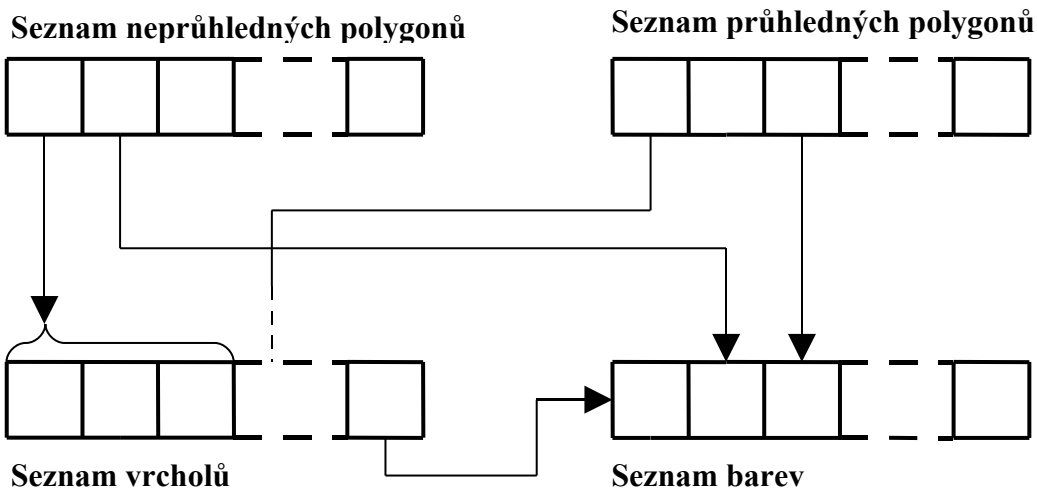
8.2 Předzpracovaná data

Data ze sedrisovského formátu transformujeme do námi definovaného formátu, který je již určen pro práci v reálném čase. Při návrhu bylo hlavním cílem, abychom byli schopni data ze

souboru co nejrychleji načíst a připravit pro práci s prohlížečem. Data proto ukládáme do souboru ve stejném formátu, v jakém s nimi pracujeme v paměti při zobrazování. Celá scéna je vždy rozdělena na bloky. Práci s těmito bloky i uchování dat v paměti zajišťuje třída *CScene* a její odvozené třídy *CPolygonScene* a *CHeightMapScene*. Každý blok je definován pomocí abstraktní třídy CTile a odvozených tříd *CPolygonTile* a *CHeightMapTile*. Dané bloky jsou v datovém souboru uloženy za sebou. Předzpracování dat se liší v závislosti na tom, zda je scéna tvořena nepravidelnou trojúhelníkovou sítí, či pravidelnou mřížkou výškových dat.

Nepravidelná trojúhelníková síť

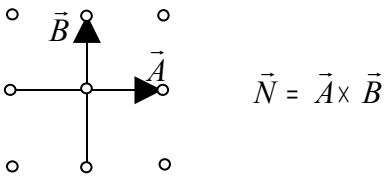
O konverzi ze sedrisovského formátu do našeho formátu se stará soubor funkcí v *Transmittal.cpp*. Konverzi spustí funkce *LoadTransmittal*. Ta načte uzel obsahující jednotlivé bloky scény (fce *ProcessSpatialIndexRelatedGeometry*) a zpracuje každý blok samostatně (fce *ProcessSpatialRelatedTiles* a *ProcessSingleTile*). Načteme všechny polygony tvořící daný blok (fce *LoadPolygons*, *GetPolygon*, *GetVertex*, *GetObjectColor*, *GetVertexTextureCoordinates*). Po tomto nahrání z formátu Sedris máme v paměti seznam neprůhledných polygonů, seznam průhledných polygonů, seznam vrcholů všech polygonů a seznam barev (viz obr. 8-1). Podrobnější popis těchto struktur nalezneme v kapitole „Objem dat v paměti“ – viz 8.5. Dále nahrajeme instance všech modelů vyskytujících se v daném bloku (fce *SelectModelInstances*). Tyto seznamy jsou uloženy pomocí šablony *CAppList*. Šablona je navržena pro uchovávání typově stejných položek v poli. Dokáže ukládat tento seznam na disk ve stejném formátu jako jej drží v paměti. Do jednotlivých seznamů se odkazujeme pomocí indexů, ne pomocí ukazatelů, neboť ty bychom nemohli jednoduše uložit na disk. Všechny tyto seznamy a tím pádem celou definici scény udržuje třída *CPolygonTile*, pomocí jejíž metod *SaveToFile* a *LoadFromFile* ukládáme a nahráváme data z našeho formátu. Samotné modely ani textury nejsou přeneseny a jsou načítány přímo ze sedrisovského souboru.



Obrázek 8-1 – Možné závislosti seznamů polygonů, vrcholů a barev.

Pravidelná mřížka výškových bodů

Konverzi pravidelné mřížky výškových bodů z formátu Sedris do našeho formátu zajišťuje funkce *ProcessHeightMap*. Ta rozdělí mřížku na bloky velikosti 257x257 bodů a po těchto blocích uloží do souboru. Bloky se navzájem překrývají vždy o jeden řádek, případně sloupec. Každý blok ukládáme nekomprimovaně po řádcích. Každý bod je definován pomocí třídy CHMPoint (podrobněji viz „Objem dat v paměti“, kapitola 8-5), ve které si uchováváme výšku bodu v terénu a dále předpočítaný normálový vektor povrchu v daném bodě. Ten spočítáme s využitím vlastností vektorového součinu pomocí čtyř okolních bodů, jak je vidět na obrázku 8-2.

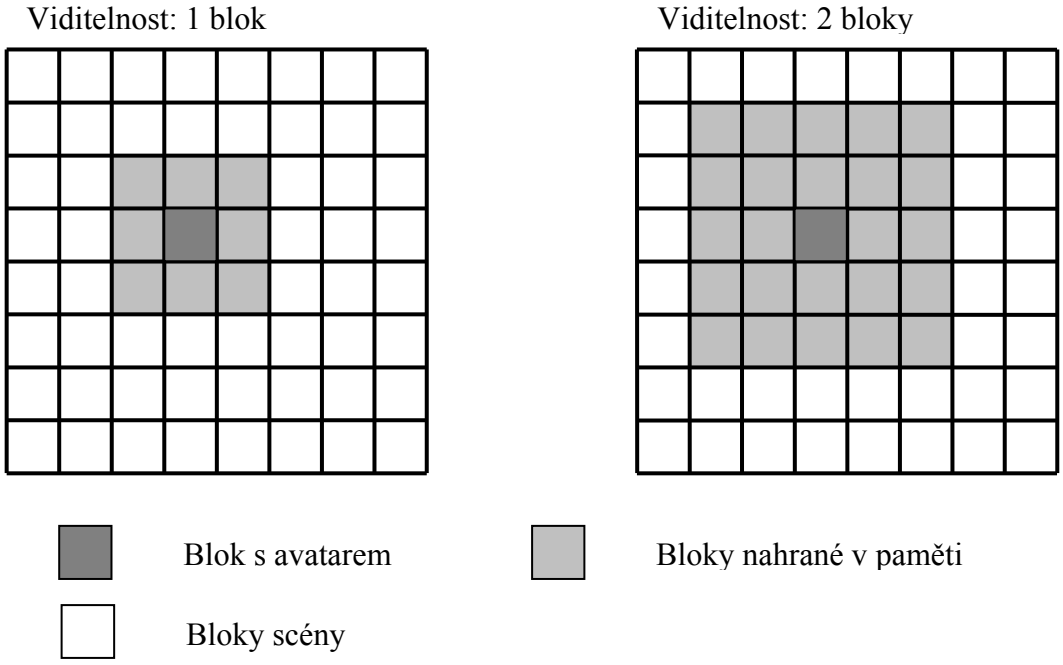


Obrázek 8-2 – Výpočet normálového vektoru pro bod mřížky.

Každou zkonvertovanou scénu uchováváme ve dvou souborech. Hlavní soubor, ten který otevíráme v prohlížeči (*.mof), obsahuje základní informace o scéně – její velikost, počet bloků. Dále je zde pro každý blok uložena pozice, která definuje místo uložení daného bloku ve druhém datovém souboru (*0000.mof).

8.3 Algoritmy použité v programu

S jednotlivými bloky scény pracujeme pomocí tříd *CPolygonScene* a *CHeightMapScene* odvozených od třídy *CScene*. Bloky nahráváme dynamicky do paměti podle pozice pozorovatele (metoda CScene::*UpdateActualTiles*). Aktuálně používané bloky tvoří čtvercový útvar s pozorovatelem v prostředním bloku. Podle minimální viditelnosti avatara[1] určíme, jakou má mít tento čtvercový útvar velikost. Vždy při přechodu avatara z jednoho bloku do druhého aktualizujeme seznam viditelných bloků (viz obr. 8-3).



Obrázek 8-3 – Nahrané bloky scény v paměti podle pozice avatara.

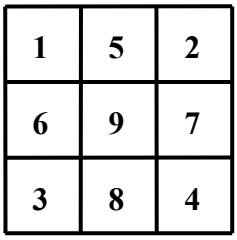
Poté ořízneme aktuálně nahrané bloky podle zorného jehlanu (*view frustum culling*), tak jak je popsáno v kapitole 7.1.1. To zajišťuje metoda *CScene::SetTileVisibility*. Dále postupuje algoritmus pro každý blok zvlášť podle toho, zda se jedná o nepravidelnou trojúhelníkovou síť, či pravidelnou mřížku výškových bodů.

Nepravidelná trojúhelníková síť

Práci s nepravidelnou trojúhelníkovou sítí zajišťují třídy *CPolygonScene* a *CPolygonTile*. Pro každý blok máme definovaný seznam neprůhledných polygonů a seznam průhledných polygonů. Neprůhledné polygony můžeme díky Z-bufferu zobrazit v jakém pořadí chceme, ale průhledné polygony musíme zobrazovat tak, aby vzdálenější polygony od pozorovatele

[1] avatar je pozorovatel pohybující se po virtuální scéně.

byly vykresleny dříve než bližší polygony. Z toho důvodu je třeba brát ohled na pořadí zpracování jednotlivých bloků. Musíme je vykreslit v pořadí podle vzdálenosti od pozorovatele (viz obrázek 8-4). Zpracování bloků zajišťuje metoda *CPolygonScne::Process*.



Obrázek 8-4 – Pořadí zpracování jednotlivých bloků.
Pozorovatel je v prostředním bloku.

V každém bloku nejprve vykreslíme neprůhledné polygony, poté průhledné polygony uspořádané podle vzdálenosti od pozorovatele. Dále vykreslíme všechny modely vyskytující se v daném bloku. Pokud je pro daný model definováno LOD, vykreslíme jej v odpovídající úrovni detailů. Pro mapování textur používáme techniku mipmappingu.

Pro každý vykreslený blok si knihovna OpenGL pamatuje seznam neprůhledných polygonů v předzpracovaném bloku příkazů. Pokud je celý blok viditelný, zavoláme pro vykreslení těchto polygonů pouze předzpracovaný blok příkazů knihovny OpenGL definovaný metodou *CPolygonTile::Process*. Pokud je blok viditelný pouze z části, zobrazujeme polygony a modely pouze v případě, že jsou vidět (metoda *CPolygonTile::ProcessPartVisible*). Při načtení knihovny modelů ze sedrisovského souboru si pro případy zjišťování viditelnosti pro každý model předpočteme obalovou plochu tvořenou koulí.

Pravidelná mřížka výškových bodů

Práci s pravidelnou mřížkou výškových bodů zajišťují třídy *CHeightMapScene* a *CHeightMapTile*. Pro zobrazení takto zadané výškové mapy používáme algoritmus pro dynamické generování LOD v reálném čase, popsaný v kapitole 7.3. Každý blok scény je pomocí tohoto algoritmu triangulován (metoda *CHeightMapTile::Precompute*) a zobrazen (metoda *CHeightMapTile::Process*). Výškovou mapu uchováváme v paměti v poli *CHeightMapTile::Matrix*, tak jak je popsáno v kapitole 7.3. Odstranění tzv. T-vrcholů (viz kapitola 7.3.) zajišťuje funkce *CHeightMapScene::SolveTVertices*. Nemáme k dispozici žádné

textury povrchu terénu, a proto terén alespoň osvítíme difůsním světlem. Pro každý bod v mřížce máme předpočítanou normálu, kterou použijeme pro výpočet osvětlení.

Problém nastal při takzvaném „vyskakování vrcholů“. Při pohybu pozorovatele ve scéně přibývají (případně ubývají) vrcholy, které přispívají k triangulaci scény. To vytváří nepříjemný efekt - tzv. vyskakování vrcholů. Algoritmus popsaný v kapitole 7.3 se tímto problémem zabývá tak, že postupně mění výšku v přidaném (ubraném) bodě. Nám ovšem k vyskakování vrcholů také značně přispívá změna normály v daném bodě. Abychom předešli vyskakování vrcholů, museli bychom spojitě měnit normálu mezi původní aproximovanou hodnotou a předpočítanou normálou v daném bodě. Kvůli časové náročnosti výpočtu se proto „vyskakováním vrcholů“ nezabýváme. Pomocí změny požadovaného rozlišení terénu se snažíme při průletu udržet konstantní počet snímků za sekundu.

Avatar

Avatar je definován pomocí třídy *CAvatar*. Ta definuje metody pro pohyb avatara (*CAvatar::Move*) a udržování konstantní výšky nad terénem (*CAvatar::SetHeight*).

Hlavní běh programu a vykreslování scény zajišťuje třída *COpenGLView* a její metoda *OnStep*. Ta vždy zpracuje pohyb avatara a vykreslí danou scénu do okna prohlížeče.

Předdefinované procházky terénem

Procházku po terénu si můžeme uložit a později znovu prohlédnout. O uložení procházky se stará třída *CWalking*. Do souboru (*.wlk) si ukládáme seznam míst, které avatar prošel, včetně odpovídajících časů a natočení pohledu avatara. Poté při rekonstrukci procházky (viz metoda *CWalking::Process*) zjistíme dvě časově nejbližší uložená místa a lineárně aproximujeme aktuální umístění avatara. Danou procházku si neudržujeme celou v paměti, ale načítáme ze souboru vždy pouze adekvátní údaje.

8.4 Rychlost zobrazování dat

Program byl testován na počítači s procesorem AMD Athlon 1800+, s grafickou kartou ATI RadeOn 9200, pamětí 512 MB a operačním systémem Windows XP. Rozlišení zobrazované scény bylo 640x480 s barevnou hloubkou 32 bitů. K testování byly použity terény dostupné na stránkách projektu Sedris (viz [19]).

Scény tvořené pravidelnou mřížkou výškových dat i nepravidelnou trojúhelníkovou sítí:

- Atlantis
- Lake Tahoe, Nevada, USA
- Twentynine Palms, California, USA
- Camp Pendleton, California, USA
- Oceanside, California, USA

Scény tvořené pouze nepravidelnou trojúhelníkovou sítí:

- Auto-generated Terrain
- Bellevue, Washington, USA
- Town Square, Anywhere

Pro každou scénu jsme vytvořili procházku, pomocí které jsme měřili rychlost zobrazování. Všechny tyto scény a odpovídající procházky jsou uloženy na přiložených CD.

Nepravidelná trojúhelníková síť

Cílem testování bylo změřit průměrný počet trojúhelníků v paměti, počet průměrně zobrazovaných trojúhelníků v jednom snímku a průměrný počet snímků za sekundu v závislosti na počtu bloků v paměti. Tabulky 8-5 až 8-12 ukazují výsledky měření. Redukce terénu udává poměr zobrazených trojúhelníků vůči počtu trojúhelníků v paměti prohlížeče.

Atlantis

Scéna pokrývá plochu 14400 km^2, obsahuje 737686 polygonů a je rozdělena do 145x101 bloků.

Počet bloků v paměti	Průměrný počet trojúhelníků v paměti	Počet průměrně zobrazovaných trojúhelníků	Průměrný počet snímků za sekundu	Redukce terénu
7x7=49	2697	750	468,5	28%
9x9=81	4313	1252	426,6	29%
11x11=121	6359	1864	362,6	29%

tabulka 8-5 - Scéna Atlantis.

Lake Tahoe, Nevada, USA

Scéna pokrývá plochu 2656 km^2, obsahuje 2034846 polygonů a je rozdělena do 101x61 bloků.

Počet bloků v paměti	Průměrný počet trojúhelníků v paměti	Počet průměrně zobrazovaných trojúhelníků	Průměrný počet snímků za sekundu	Redukce terénu
7x7=49	13112	3149	125,4	24%
9x9=81	19882	4549	88,2	22,8%
11x11=121	27767	5843	68,4	21%

tabulka 8-6 - Scéna Lake Tahoe, Nevada, USA.

Twentynine Palms, California, USA

Scéna pokrývá plochu 5,5 km^2, obsahuje 119673 polygonů a je rozdělena do 29x19 bloků.

Počet bloků v paměti	Průměrný počet trojúhelníků v paměti	Počet průměrně zobrazovaných trojúhelníků	Průměrný počet snímků za sekundu	Redukce terénu
7x7=49	10385	2316	270	22,3%
9x9=81	16709	3808	183,1	22,8%
11x11=121	24132	5435	136,8	22,5%

tabulka 8-7 Scéna Twentynine Palms, California, USA.

Auto-generated Terrain

Scéna pokrývá plochu 100 km^2, obsahuje 100352 polygonů a je rozdělena do 10x10 bloků. Tato scéna neobsahuje textury, všechny polygony jsou pouze obarveny.

Počet bloků v paměti	Průměrný počet trojúhelníků v paměti	Počet průměrně zobrazovaných trojúhelníků	Průměrný počet snímků za sekundu	Redukce terénu
7x7=49	96398	21768	37,6	22,6%
9x9=81	141063	34634	27,2	24,5%
11x11=121	177183	44093	23	24,9%

tabulka 8-8 - Scéna Auto-generated Terrain.

Camp Pendleton, California, USA

Scéna pokrývá plochu 625 km^2, obsahuje 185132 polygonů a je rozdělena do 50x50 bloků.

Počet bloků v paměti	Průměrný počet trojúhelníků v paměti	Počet průměrně zobrazovaných trojúhelníků	Průměrný počet snímků za sekundu	Redukce terénu
7x7=49	8958	1935	337.9	21,6%
9x9=81	14345	3154	255.6	22%
11x11=121	19780	4549	207.7	23%

tabulka 8-9 - Scéna Camp Pendleton, California, USA.

Oceanside, California, USA

Scéna pokrývá plochu 9523 km^2, obsahuje 362782 polygonů a je rozdělena do 107x89 bloků.

Počet bloků v paměti	Průměrný počet trojúhelníků v paměti	Počet průměrně zobrazovaných trojúhelníků	Průměrný počet snímků za sekundu	Redukce terénu
7x7=49	2057	523	405.8	25,4%
9x9=81	3374	889	380.6	26,3%
11x11=121	5015	1353	359,5	27%

tabulka 8-10 - Scéna OceanSide, California, USA.

Bellevue, Washington, USA

Scéna pokrývá plochu 100 km^2, povrch je tvořen 13734 polygony a je rozdělen do 20x20 bloků. Obsahuje 19339 modelů, které jsou dohromady tvořeny 23886 polygony.

Počet bloků v paměti	7x7=49	9x9=81	11x11=121
Průměrný počet trojúhelníků v paměti	1819	2788	3765
Počet průměrně zobrazovaných trojúhelníků (pouze povrch)	378	605	820
Průměrný počet modelů ve scéně	135	199	282
Počet průměrně zobrazovaných trojúhelníků (pouze modely)	187	272	371
Průměrný počet snímků za sekundu	385,9	311	253
Redukce terénu	20,15	21,7%	21,8%

tabulka 8-11 - Scéna Bellevue, Washington, USA.

Town Square, Anywhere

Scéna pokrývá plochu 0.25 km^2, povrch je tvořen 217276 polygony a je rozdělen do 6x4 bloků. Obsahuje 341 modelů, které jsou dohromady tvořeny 53181 polygony.

Počet bloků v paměti	6x4=24
Průměrný počet trojúhelníků v paměti	10523
Počet průměrně zobrazovaných trojúhelníků (pouze povrch)	1826
Průměrný počet modelů ve scéně	40

Počet průměrně zobrazovaných trojúhelníků (pouze modely)	4860
Průměrný počet snímků za sekundu	135,4%
Redukce terénu	17,4%

tabulka 8-12 - Scéna Town Square, Anywhere.

Pravidelná mřížka výškových bodů

Cílem testování bylo změřit průměrný počet trojúhelníků vykreslených v jednom snímku v závislosti na velikosti mřížky udržované v paměti prohlížeče. Udržovali jsme konstantní rychlost 60 snímků za sekundu. Tabulky 8-13 až 8-16 ukazují výsledky měření. Redukce terénu udává poměr zobrazených trojúhelníků vůči počtu trojúhelníků tvořících plně triangulovanou část scény v paměti prohlížeče.

Twentynine Palms, California, USA

Scéna pokrývá plochu 6,7 km^2 a je popsána maticí velikosti 3276x2164 bodů. Vzdálenost sousedních bodů je 1 metr.

Počet bloků v paměti	**Velikost mřížky v paměti**	**Počet průměrně zobrazovaných trojúhelníků**	**Průměrný počet snímků za sekundu**	**Redukce terénu**
3*3=9	769x769	8547	58,9	0,72%
5*5=25	1281x1281	7005	60	0,21%
7*7=49	1793x1793	5636	60,1	0,09%

Tabulka 8-13 – Scéna Twentynine Palms, California, USA.

Lake Tahoe, Nevada, USA

Scéna pokrývá plochu 2940 km^2 a je popsána maticí velikosti 8430x3516 bodů. Vzdálenost sousedních bodů je 10 metrů. Tato scéna je velice hornatá a tudíž i náročná na vykreslování.

Počet bloků v paměti	**Velikost mřížky v paměti**	**Počet průměrně zobrazovaných trojúhelníků**	**Průměrný počet snímků za sekundu**	**Redukce terénu**
3*3=9	769x769	7885	58.6	0,67%
5*5=25	1281x1281	6677	56,6	0,20%
7*7=49	1793x1793	5337	59,4	0.08%

Tabulka 8-14 – Scéna Lake Tahoe, Nevada, Usa.

Camp Pendleton, California, Usa

Scéna pokrývá plochu 630 km^2 a je popsána maticí velikosti 251x251 bodů. Vzdálenost sousedních bodů je 100 metrů.

Velikost mřížky v paměti	Počet průměrně zobrazovaných trojúhelníků	Průměrný počet snímků za sekundu	Redukce terénu
257x257	4041	150	15,7%

Tabulka 8-15 – Scéna Camp Pendleton, California, USA.

Oceanside, California, USA

Scéna pokrývá plochu 10167 km^2 a je popsána maticí velikosti 4405x3693 bodů. Vzdálenost sousedních bodů je 25 metrů.

Počet bloků v paměti	Velikost mřížky v paměti	Počet průměrně zobrazovaných trojúhelníků	Průměrný počet snímků za sekundu	Redukce terénu
3*3=9	769x769	6142	60,5	0,52%
5*5=25	1281x1281	5400	60.6	0,16%
7*7=49	1793x1793	4658	60,5	0,07%

Tabulka 8-16 – Scéna Oceanside, California, USA.

Čím větší byla velikost mřížky v paměti, tím více času zabrala triangulace scény. Z toho důvodu jsou paradoxně větší scény tvořeny menším počtem trojúhelníků.

8.5 Objem dat v paměti

Nepravidelná trojúhelníková síť

V souboru a následně i v paměti prohlížeče uchováváme seznam polygonů, seznam vrcholů, seznam barev a seznam instancí modelů. Následuje přehled velikostí jednotlivých struktur:

- Datová struktura pro **polygon** (*CPolygon*) obsahuje: počet vrcholů (int), index do seznamu vrcholů na první vrchol polygonu (vrcholy polygonu jsou uloženy těsně za sebou) (int), index do seznamu textur (int), bitovou mapu, která určuje, čím je polygon definovaný (int). Každý polygon tedy zabírá v souboru i v paměti prohlížeče 20 bytů.
- Datová struktura pro **vrchol** (*CVertex*) obsahuje: trojrozměrnou pozici ve scéně (3x float), mapování do textury (2x float), index do seznamu barev (int), bitovou mapu určující, jak je daný vrchol definovaný (int). Dohromady zabírá tato struktura 28 bytů.
- Datová struktura pro **barvu** (*CColor*) obsahuje čtyři složky RGBA, to znamená 4 byty.

- Datová struktura pro **instanci modelu** (*CModelInstance*) obsahuje: polohu objektu ve scéně (3x float), matici transformace modelu (9x float). Každá instance modelu zabírá v paměti 48 bytů.

Po načtení scény se ze sedrisovského souboru načtou modely, kde každý model si můžeme představit jako samostatnou scénu. Na každý model se opět můžeme dívat jako na seznam polygonů, vrcholů, barev a instancí modelů.

Uveďme příklad: mějme scénu obsahující 10000 trojůhelníků, kde každý polygon bude otexturován. Poté seznam polygonů bude mít 10000x20 bytů, což je 200kB. Seznam vrcholů je veliký 10000x3x28 bytů, což je 840 kB. Základní datové struktury tedy budou obsahovat na disku i v paměti přibližně 1MB. Dále v paměti udržujeme textury a další pomocné informace o scéně, jako je např. rozdělení do bloků. Tyto pomocné informace jsou však svojí velikostí vůči předchozím datům zanedbatelná.

Pravidelná mřížka výškových bodů

Pro každý bod pravidelné mřížky si v datovém souboru uchováváme informace o výšce (float) a dále předpočítaný normálový vektor v daném bodě (3x byte) (třída *CHMPoint*). Po nahrání této struktury do paměti si dále udržujeme pomocné stavové informace (int). Datová struktura pro bod zabírá po zarovnání 8 bytů v souboru a 12 bytů v paměti.

Uveďme příklad: mějme scénu velikosti 1000x1000 bodů. Tu rozdělíme do bloků velikosti 257x257, které se vzájemně překrývají o jeden pixel. Dostaneme tak 16 bloků, kde každý blok zabírá v souboru 257x257x8 bytů, což je přibližně 0,5 MB. Po načtení do paměti má daný blok velikost 257x257x12 bytů, čili přibližně 0,75 MB. Celá scéna na disku bude mít 8 MB, přídavné informace můžeme vzhledem k této velikosti zanedbat. Pokud bychom v paměti udržovali scénu o velikosti 3x3 bloky (v paměti většinou neudržujeme celou scénu), zabírala by necelých 7MB. Opět můžeme vzhledem k této velikosti zanedbat další pomocná data týkající se většinou celých bloků.

8.6 Srovnání s programem Side-by-Side Viewer

Side-by-Side Viewer (viz [14]) je program vyvinutý společností AcuSoft. Je určen pro zobrazování scén v Sedris formátu. Scény tvořené pravidelnou mřížkou výškových bodů se

nám nepodařilo zobrazit. Program dokázal nahrát pouze scénu Camp Pendleton, California, USA, nicméně ji zobrazoval jen jako rovnou plochu bez jakýchkoliv vyvýšenin. Program se zdá být použitelný pouze pro scény tvořené nepravidelnou trojúhelníkovou sítí. Při pohybu po scéně prohlížeč nedokáže dynamicky nahrávat aktuální části scény. Pokud bychom tedy chtěli prolétávat nad celým terénem, museli bychom ho celý nahrát do paměti, což je u větších terénů problém, neboť program končí s chybovými hláškami. Menší scény, které můžeme nahrát celé (Bellevue, Washington, USA, Auto-generated Terrain, Twentynine Palms, California USA, Town Square, Anywhere), nemůžeme dost dobře kvůli malé zobrazovací frekvenci procházet (přibližně několik snímků za sekundu). Nad menšími částmi scény program prolétává plynule. Bohužel zde nemůžeme zjistit zobrazovací frekvenci, a proto nemůžeme z hlediska rychlosti porovnat Side-by-side Viewer s Terrain Viewerem.

Side-by-side Viewer má daleko více možností v nastavení vlastností prohlížené scény. Můžeme nastavit osvětlení, barvu oblohy, mlhu. Můžeme zapínat/vypínat algoritmy, které se v programu použijí – backface culling, view frustum culling, lod (viz 7.1). Program dokáže zobrazovat scénu v různých módech: jako drátěný model, neotexturovanou. Zobrazuje strukturu sedrisovského souboru. Dokáže zobrazit jakákoliv data v Sedris formátu, ne jenom terén.

Program byl testován na stejné konfiguraci jako při měření rychlosti dat – viz kapitola 8.4.

Nenašli jsme žádné jiné prohlížeče, které by umožnily zobrazit scénu v Sedris formátu.

9 Závěr

V této práci jsme se nejprve zabývali datovými strukturami pro popis povrchu terénu. Popsali jsme hierarchické struktury, které se dají použít při zobrazování pomocí LOD. Věnovali jsme se mapování textur na povrch terénu. Zjistili jsme, kde se mohou vyskytovat problémy i způsob jejich řešení. Dále jsme rozebrali možnosti popisu prvků vyskytujících se v přírodním prostředí.

Popisem dat přírodního prostředí se zabývá projekt Sedris, který pro tyto účely vyvinul svůj vlastní formát. Tento formát se ale nehodí pro zpracování v reálném čase, a proto jsme pouze popsali jeho rozsáhlé možnosti a dále jsme se jím podrobněji nezabývali.

Probrali jsme možnosti automatického generování terénů. Popsali jsme šest způsobů, jak generovat výškovou mapu terénu. Každý způsob měl své výhody a své nevýhody. Dále jsme se zabývali vytvářením nepravidelné trojúhelníkové sítě z husté mřížky výškových bodů. Popsali jsme dva algoritmy na výběr reprezentativních bodů trojúhelníkové sítě a dále pak dva algoritmy na tvorbu sítě z těchto reprezentativních bodů. Obzvlášť dobrá síť byla tvořena pomocí Delaunayho triangulace.

V následující části jsme se zabývali zobrazovacími algoritmy. Nejprve jsme představili základní optimalizační techniky na redukci počtu zobrazovaných trojúhelníků. Věnovali jsme se dynamickému generování trojúhelníkové sítě v reálném čase pomocí LOD. Popsali jsme dvě reprezentativní techniky, jednu pro práci s nepravidelnou trojúhelníkovou sítí, druhou pro práci s pravidelnou mřížkou výškových bodů. Ostatní algoritmy pro generování LOD v reálném čase jsou založené na podobném mechanismu.

V praktické části této práce jsme implementovali prohlížeč terénů, který prochází nad rozsáhlými virtuálními terény v reálném čase. Vstupní data jsou ve formátu Sedris, program si je transformuje do svého vlastního formátu. Kromě aplikace Side-by-Side Viewer jsme nenašli žádný jiný prohlížeč tohoto formátu. Tato aplikace se ale nedokázala vyrovnat s velkými scénami, program končil s chybovými hláškami.

10 Seznam literatury

[1] Paul Bourke (Leden 1989): Efficient Triangulation Algorithm Suitable for Terrain Modelling. *Pan Pacific Computer Conference, Beijing, China, 1989.* Dostupný na adrese: http://astronomy.swin.edu.au/~pbourke/terrain/triangulate/

[2] C. Burke: Generating terrain, http://www.geocities.com/Area51/6902/terrain.html

[3] Urs Enzler (2001): Multiresolution Terrain Triangulation using a Delaunay Hierarchy. Dostupný na adrese: http://www.inf.ethz.ch/personal/stamm/sada/sa_enzler.pdf

[4] H. Hoppe (1996): Progressive Meshes. *Computer Graphics (SIGGRAPH '96 Proceedings), stránky 99-108.*

[5] Hugues Hoppe (1997): View Dependent Refinement of Progressive Meshes. *Computer Graphics (SIGGRAPH '97 Proceedings), stránky 189-198.*

[6] Aleks Jakulin (2000): Interactive Vegetation Rendering with Slicing and Blending. *Conference Proceedings Eurographics 2000.* Dostupný na adrese: http://ai.fri.uni-lj.si/~aleks/slicing-and-blending/trees-electronic.pdf

[7] F. Mamaghani, K. Wertman, A. Tosh (Červen 2001): Sedris to CTDB and CTDB to Sedris Conversions Tutorial. *Sedris Technology Conference 2001.* Dostupný na adrese: http://www.sedris.org/stc/2001/tu/ctdb/sld005.htm

[8] T. McReynolds, D. Blythe (1998): Advanced Graphics Programming Techniques Using OpenGL, kapitola Detail Textures. *SIGGRAPH Course Notes, 1998.* Dostupný na adrese: http://www.sgi.com/software/opengl/advanced98/notes/node64.html

[9] T. McReynolds, D. Blythe (1998): Advanced Graphics Programming Techniques Using OpenGL, kapitola Paging Textures. *SIGGRAPH Course Notes, 1998.* Dostupný na adrese: http://www.sgi.com/software/opengl/advanced98/notes/node38.html

[10] Thomas K. Poiker (1990): The TIN model. *Unit 39 of the NCGIA Core Curriculum in GIScience.*

[11] Morné Pistorius: Optimization techniques in terrain rendering. *Dept. of Computer Science University of Stellenbosch.* Dostupný na adrese: http://www.cs.sun.ac.za/~henri/Optimization%20techniques%20in%20Terrain%20Rendering.pdf

[12] S. Röttger, W. Heidrich, P. Slusallek, H.P. Seidel (1998): Real-time Generation of Continuous Levels of Detail for Height Fields. *Conference Proceedings of WSCG '98, stránky 315-322.*

[13] Jiří Žára (1999): VRML 97 Laskavý průvodce virtuálními světy. *Computer Press.*

[14] AcuSoft, Inc.: Side-by-Side Viewer. Dostupný na adrese: http://www.acusoft.com/products/sbs/

[15] GeoVRML, Dostupný na adrese: http://www.geovrml.org/

[16] MultiGen-Paradigm: Open Flight, http://www.multigen-paradigm.com/products/standards/openflight/index.shtml

[17] Okino Computer Graphics: Open Flight Importer, http://www.okino.com/conv/imp_flt.htm

[18] Projekt SEDRIS, http://www.sedris.org/

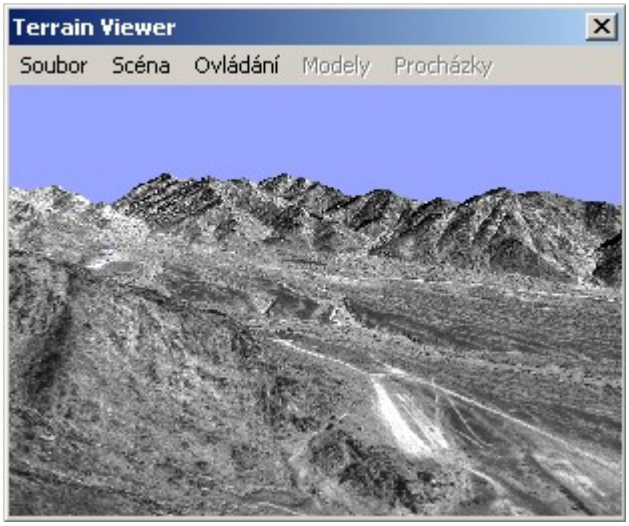
[19] SEDRIS Data, http://data.sedris.org

[20] Virtual Terrain Project, http://www.vterrain.org/

11 Přílohy

A. Uživatelská dokumentace

V této kapitole se seznámíme s ovládáním programu „Terrain Viewer“. Po spuštění se nám objeví ovládací panel (viz obrázek A-1), ze kterého spouštíme všechny příkazy.



Obrázek A-20 - Ovládací panel programu Terrain Viewer.

Otevření scény:

- **Otevření nepředzpracované scény** - Z menu vybereme položku *Soubor->Otevřít nezpracovanou scénu*. V následujícím dialogu vybereme soubor se scénou, kterou chceme předzpracovat a otevřít. Dále určíme soubor, kde bude uložena předzpracovaná scéna. Scéna se převede do našeho formátu a zobrazí se.
- **Otevření předzpracované scény** – Z menu vybereme položku *Soubor->Otevřít předzpracovanou scénu*. Pokud otvíráme scénu s pravidelnou mřížkou výškových bodů, stačí zadat pouze scénu v našem formátu (*.mof). Jinak, pokud je scéna tvořena nepravidelnou trojúhelníkovou sítí, musíme zadat i odpovídající scénu ve formátu Sedris. Poté se scéna načte do paměti a zobrazí.

Soubory mají následující značení:

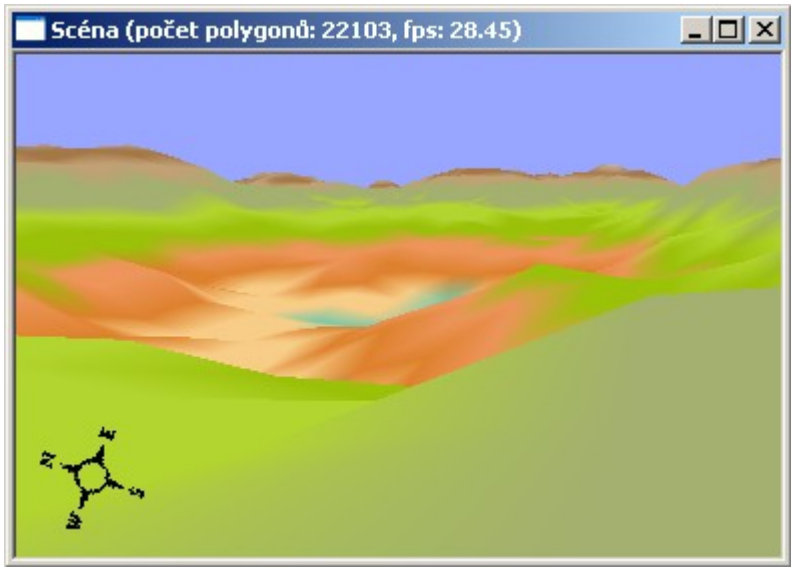
- *.stf – hlavní soubor scény ve formátu Sedris
- *00000.stf, *00001.stf, atd. – pomocné soubory scény ve formátu Sedris
- *.mof – hlavní soubor předpočítané scény
- *0000.mof – pomocný soubor předpočítané scény

- *.wlk – soubor obsahující zaznamenanou procházku po scéně

V dialogovém okně pro otevření scény vybíráme pouze hlavní soubory scén.

Okno se zobrazovanou scénou:

Po otevření a zpracování souboru se automaticky otevře okno se zobrazovanou scénou (viz obrázek A-2). Pokud je scéna načtena a nemáme zobrazeno okno se scénou, můžeme toto okno otevřít pomocí menu *Scéna->Zobrazit.*



Obrázek A-21 - Okno pro vizualizaci scény.

Po terénu se můžeme procházet (pohled je umístěn do uživatelem zadané výšky nad terénem), případně nad ním létat. Při pohybu po terénu se v titulku okna se scénou zobrazuje aktuální počet zobrazovaných polygonů a počet snímků vykreslovaných za sekundu. Do módu procházení po scéně se dostaneme kliknutím myši na zobrazený terén. Tento mód opustíme opětovným kliknutím na tlačítko myši. Ovládání scény a avatara popisuje tabulka A-3.

Ovládání avatara:

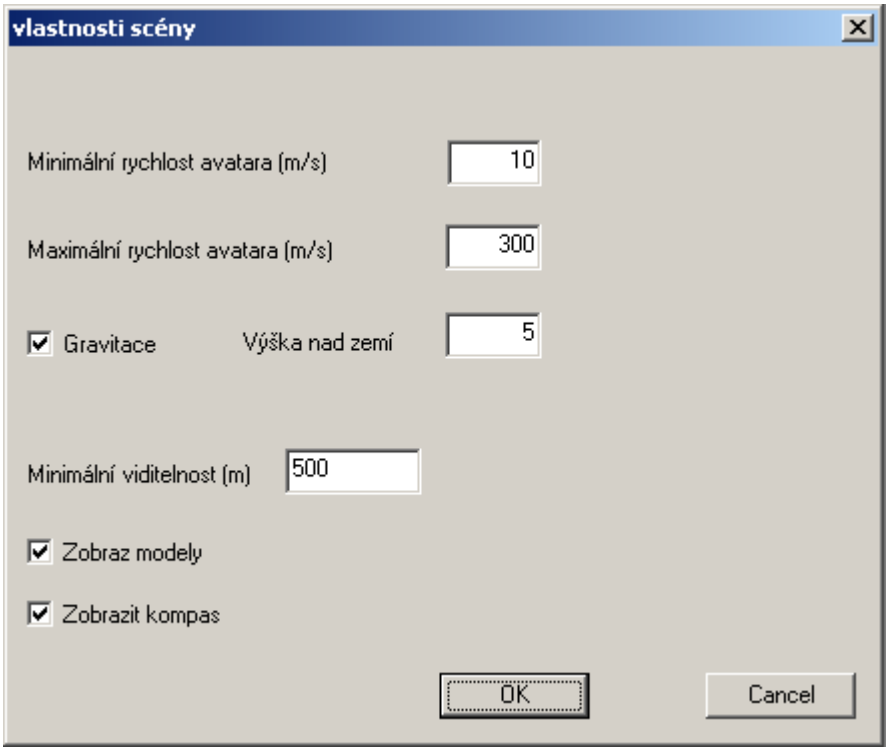
Pohyb myši	**Ovládání pohledu avatara**
Klávesové šipky	**Pohyb dopředu, dozadu a do stran**
Klávesy „E“, „S“, „D“, „F“	**Alternativní možnost pohybu dopředu, dozadu a do stran**
Klávesa „G“	**Zapnutí/vypnutí gravitace**

Klávesa „M“	**Zapnutí/vypnutí zobrazování modelů**
Klávesa „C“	**Zapnutí/vypnutí zobrazování kompasu**
Klávesa „T“	**Zapnutí/vypnutí zobrazování drátěného modelu (pouze při zobrazování pravidelné mřížky výškových bodů)**
Klávesy „1“ – „9“	**Změna rychlosti pohybu avatara. „1“ – nejmenší rychlost avatara „9“ – největší rychlost avatara**

Tabulka A-3 – Ovládání avatara.

Vlastnosti scény:

Vlastnosti zobrazované scény můžeme měnit v panelu „vlastnosti scény“ (viz obrázek A-4), který zobrazíme pomocí menu *Scéna -> Vlastnosti*. Jednotlivé vlastnosti jsou zřejmé přímo z dialogového okna.

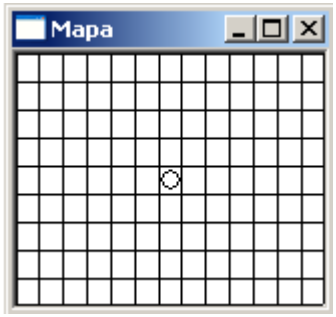


Obrázek A-4 - Definice vlastností zobrazované scény.

Mapa scény:

Pro lepší orientaci v terénu si můžeme otevřít mapu dané scény pomocí menu *Scéna->Mapa* (viz obrázek A-5). Mapa zobrazuje rozdělení scény na jednotlivé bloky a pozici avatara

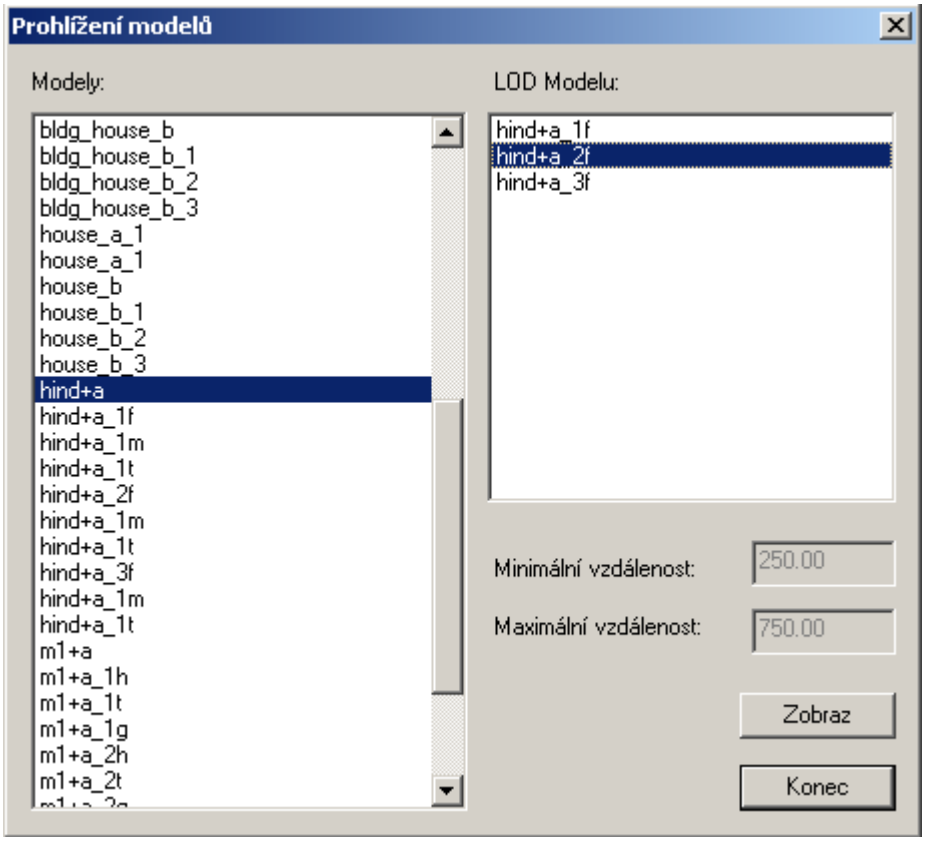
v těchto blocích. Avatara můžeme ve scéně přemístit pomocí kliknutí na požadované místo v mapě.



Obrázek A-5 - Mapa zobrazované scény.

Prohlížeč modelů:

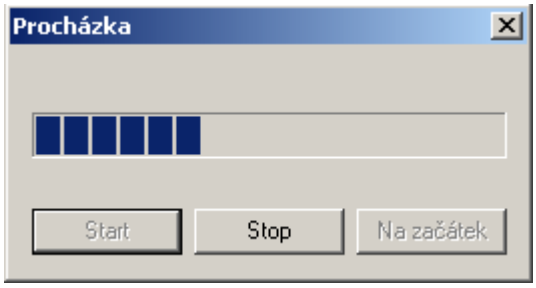
Pokud jsou ve scéně definované modely, prohlédneme je pomocí dialogového okna *„Prohlížeč modelů“* (viz obrázek A-6) které aktivujeme v menu *Modely*. Vybereme si model, který chceme zobrazit a jeho LOD, pokud existuje. Model se nám zobrazí v okně pro vizualizaci scény.



Obrázek A-6 - Prohlížeč modelů.

Zaznamenaná procházka po scéně:

Program Terrain Viewer dokáže uchovávat jednotlivé procházky po scéně. Pokud si chceme uložit procházku, vybereme z menu položku *Procházky -> Nová pocházka*. Poté vybereme soubor, do kterého chceme danou procházku po terénu uložit (soubor má značení *.wlk). Následující pohyb po scéně bude uložen do vybraného souboru. Po dokončení se nám objeví dialog (viz A-7), pomocí něhož si můžeme uloženou procházku znovu přehrát.



Obrázek A-7 Přehrání uložené procházky.

Uloženou procházku můžeme nahrát ze souboru pomocí položky menu *Procházky -> Otevřít procházku*. Poté vybereme soubor, ze kterého chceme danou procházku načíst. Objeví se dialog pro přehrání uložené procházky. Pokud necháme přehrát celou procházku, objeví se nám v okně pro výpis zpráv údaje o zobrazované procházce – počet průměrně zobrazovaných trojúhelníků a průměrný počet snímků za sekundu. Pro scénu tvořenou nepravidelnou trojúhelníkovou sítí se navíc zobrazí průměrný počet trojúhelníků v paměti během procházky.

B. Struktura CD

První CD obsahuje následující adresáře:

- **TerrainViewer** – adresář obsahující aplikaci Terrain Viewer (TerrainViewer.exe)
  - **src** – zdrojové kódy programu
- **Data** - adresář obsahující scény
  - **Tin** - scény tvořené nepravidelnou trojúhelníkovou sítí
    - **Atlantis**
    - **Bellevue, Washington, USA**
    - **Lake Tahoe, Nevada, USA**
    - **Town Square, Anywhere**
- **Text** – adresář obsahující text diplomové práce

Druhé CD obsahuje následující adresáře:

- **Data** - adresář obsahující scény
  - **Tin** - scény tvořené nepravidelnou trojúhelníkovou sítí
    - **Auto-generated Terrain**
    - **Camp Pendleton, California, USA**
    - **Oceanside, California, USA**
    - **Twentynine Palms, California, USA**
  - **HeightMap** – scény tvořené pravidelnou mřížkou výškových bodů
    - **Atlantis**
    - **Camp Pendleton, California, USA**
    - **Lake Tahoe, Nevada, USA**
    - **Oceanside, California, USA**
    - **Twentynine Palms, California, USA**

Každý adresář se scénou obsahuje scénu ve formátu Sedris, v předpočítaném formátu a procházku po dané scéně.

- *.stf – hlavní soubor scény ve formátu Sedris
- *00000.stf, *00001.stf, atd. – pomocné soubory scény ve formátu Sedris
- *.mof – hlavní soubor předpočítané scény
- *0000.mof – pomocný soubor předpočítané scény
- *.wlk – soubor s procházkou po scéně