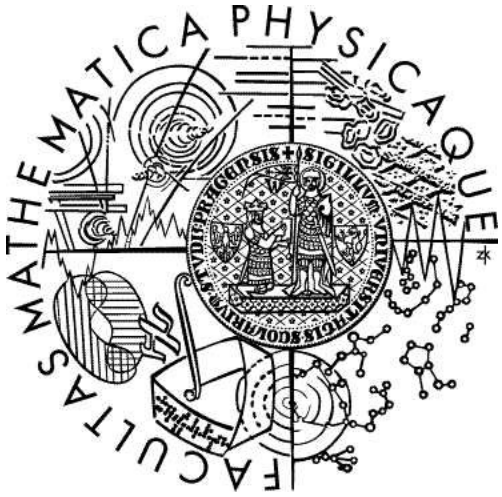
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Radek Křižka

Modelování a procházení terénu

Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Josef Pelikán
Studijní program: informatika, obecná informatika

2007

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu práce panu RNDr. Josefu Pelikánovi za podnětné připomínky, průběžnou kontrolu mých výsledků a celkové vedení práce.

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

V Praze 25. července 2007 Radek Křižka

Obsah

Název práce: Modelování a procházení terénu
Autor: Radek Křižka
Katedra (ústav): Kabinet software a výuky informatiky
Vedoucí bakalářské práce: RNDr. Josef Pelikán
e-mail vedoucího práce: josef.pelikan@mff.cuni.cz
Abstrakt:

Práce se zabývá problémem modelování a zobrazování scény. Předmětem modelování je jak terén, tak také objekty, které se na něm mohou nacházet. Vstupní terén může být libovolný, proto nepřipadá v úvahu, že by vstupní data terénu obsahovaly údaje o každé pozici stromu či keře na mapě. To je právě úkolem našeho modelování. Na základě vstupních parametrů musíme být schopni vygenerovat takové pozice objektů na mapě, aby se co nejvíce blížily skutečnosti. Poslední částí práce je zobrazení takto vymodelované scény a její procházení. Seznámíme se s různými typy objektů, možnostmi optimalizace zobrazování a technikami Level of detail. Výsledkem je aplikace, která nám demonstruje popsané techniky na příkladě scény s několika typy porostů, kterou můžeme interaktivně procházet.

Klíčová slova: modelování terénu, ekosystém, 3D grafické objekty, level of detail, OpenGL.

Title: Terrain modelling and walk-through
Author: Radek Křižka
Department: Department of Software and Computer Science Education
Supervisor: RNDr. Josef Pelikán
Supervisor's e-mail address: josef.pelikan@mff.cuni.cz
Abstract:

My thesis deals with terrain modelling and scene displaying. The main subject of modelling is the terrain as well as objects, that can be found on it. Terrain data may be random so they cannot include information about the position of each object (e.g. a tree or abush). This is the aim of our modelling; given the entry data, we must be able to generate objects at positions that would be as close to reality as possible. The last, but certainly not least part of the project is displaying the model-scene with the possibility of a walk-through. We will be introduced to various object types, possibilities of displaying optimization and we'll explore the "Level of detail" techniques. The result is an application displaying the described techniques on an example scene with a few types of brushwood including the possibity of an interactive walk-through.

Keywords: terrain modelling, ecosystem, 3D graphic objects, level of detail, OpenGL.

Kapitola 1

Úvod

V posledních letech dochází k velkému rozvoji a rozmachu virtuální reality. Virtuální realita se nám snaží nabídnout něco reálného, skutečného za pomocí různých prostředků. Má za úkol navodit člověku takovou atmosféru, jako by se pohyboval ve skutečném prostředí. Pojem virtuální realita můžeme velice snadno propojit s pojmem simulace. Virtuální realita nám může simulovat určitý jev, prostředí nebo věc. V dnešní době se bez simulací téměř neobejdeme, spousta věcí se nedá odzkoušet nebo ověřit. Ať je to již například z časových důvodů, kdy reálné odzkoušení určité věci by zabralo několik lidských generací, nebo neméně důležitých finančních důvodů. Virtuální realita, simulace a modelování různých věcí mají skutečné uplatnění v mnoha oborech, jako jsou lékařství, průmysl - strojírenství, stavebnictví, různá sportovní odvětví, vojenství a v neposlední řadě je to klimatologie a vývoj podnebí na Zemi.

1.1 Cíl projektu

Tato práce se zabývá modelováním a procházením terénu. Podle vstupních výškových dat terénu, které mohou být samozřejmě založeny na skutečném pokladu, se vygeneruje 3D terén. Vstupní data obsahují další atributy, které nám dále podrobněji popisují scénu. Ať se již jedná o typ přírodního povrchu, jako je tráva, pole, louka, voda atd., nebo o druh porostu, jeho hustotu a jiné. Hlavním úkolem této práce je vymodelovat "přírodní ekosystém", nebo-li takové prostředí, které má odpovídat skutečnému reálnému prostředí. Vstupní terén může být samozřejmě libovolný, proto nepřipadá v úvahu, že by vstupní data terénu obsahovaly údaje o každé pozici stromu či keře na mapě. To je právě úkolem našeho modelování. Na základě vstupních parametrů musíme být schopni vygenerovat takové pozice objektů na mapě, aby se co nejvíce blížily skutečnosti. Skutečnost může být pro každého člověka jiná, každý má jinou představu o jehličnatém lese. Ale všechny tyto modelace by měly mít jednu věc společnou, a to, že by neměly být porušovány základní pravidla a zákony přírody, tedy "zákony" našeho ekosystému.

Největším nepřítelem modelování a simulací jsou prostředky, které můžeme k je-

jich realizaci použít. Výpočetní kapacita již dosahuje velmi dobrých výsledků, ale k zobrazení obrovských vstupních dat terénu a vykreslování statisíců stromů a keřů v reálné kvalitě nám stále nedostačuje. Proto musíme použít vhodné techniky optimalizace a takové metody, které nám zjednoduší a ulehčí modelování. Tyto techniky musí být takové, abychom stále zachovávali důvěryhodnost modelované scény. Vycházíme ze základních fyzikálních zákonů, předměty v naší blízkosti vidíme jinak než předměty vzdálené několik stovek metrů či kilometrů. Blízký předmět pozorujeme ve velkých detailech, kdežto u předmětu vzdáleného několik set metrů jsme schopni pouze rozeznat jeho obrys. Tohoto jevu právě využíváme u modelování, kdy k vykreslení vzdáleného předmětu si vystačíme s mnohem jednodušším modelem, tzn. mnohem menšími výpočetními prostředky než u předmětu blízkého.

1.2 Popis přírodního ekosystému

Našim úkolem je modelovat přírodní ekosystém. Pod tímto pojmem si lze představit mnoho různých krajin a terénů, ale všechny by měly mít něco společného. Je to alespoň částečná podoba v nějaké skutečné přírodní scéně.

Každý člověk má jistě svou představu o přírodní scéně, ale všude by měly být dodržovány základní "přírodní zákony a pravidla". Samozřejmě tyto a další omezující podmínky si při modelování může každý zvolit sám. Pro příklad uvedeme několik pravidel, které můžeme používat při modelování ekosystému.

- Rozmístění stromů v přírodě není pravidelné. To znamená, že stromy v lese nejsou rozmístěny do nějakých pravidelných geometrických útvarů (jako například čtverce, trojúhelníky, šestiúhelníky atd.).
- Stromy v lese nevytváří shluky. Tzn. že v jednom lese nejsou místa s obrovskou koncentrací stromů a oblasti téměř bez jediného stromu. Počty stromů ve stejně velkých oblastech lesa jsou přibližně shodné.
- Vedle sebe se v lese nenacházejí dva velké stromy. Navzájem by si stínily a zhoršovaly si životní podmínky, proto v těsné blízkosti velkého stromu nejsou žádné jiné velké stromy.

Kapitola 2

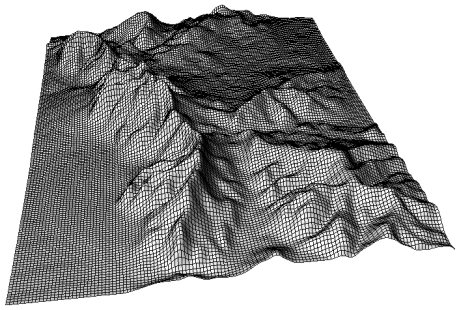
Modelování terénu

V této kapitole se budeme zabývat možnostmi, jak lze modelovat terén. V první řadě se budeme zabývat reprezentací vstupních dat a jejich zpracováním. Dále to budou různé možnosti optimalizace následného zobrazení.

2.1 Reprezentace vstupních dat terénu

Našim úkolem je modelovat reálný terén, což znamená třírozměrné prostředí. Každý bod v takovém prostředí je tedy charakterizován třemi souřadnicemi.

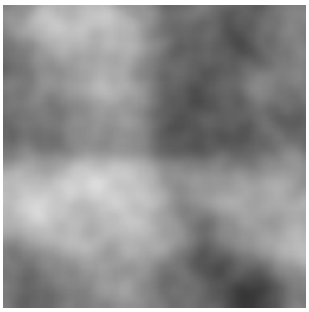
Jednou ze základních možností jak reprezentovat takový terén jsou tzv. **výškové pole**[1]. Jedná se obvykle o dvourozměrné pole, které indexujeme pomocí souřadnic X-Y. Každý bod v tomto poli nese informaci o své výšce. Takto zadaná mapa nám představuje speciální druh pravidelné trojúhelníkové sítě TRN (anglická zkratka od Triangulated Regular Network - TRN). Hlavní nevýhodou výškových map je nemožnost modelovat terén s přesahy (tj. terén se skalními převisy, díry ve skalách, tunely, apod.). Jestliže si označíme osu z jako osu kolmou k našemu povrchu, pak nám může protnout terén v každém místě pouze jednou.



Obrázek 2.1: Příklad pravidelné trojúhelníkové sítě

[1] v anglických textech se označuje jako **heightfield** nebo **height-field**

Jednoduchá reprezentace takového výškového pole je například pravidelná rastrová mřížka (bitmapa), tzv. **výšková mapa**[2]. Příklad takové mapy je na obrázku 2.2. Jednotlivé pixely na pozici X, Y neobsahují údaj o barvě, ale o výšce bodu na pozici X, Y. Jestliže tedy máme obrázek ve stupních šedi, pak černá barva má význam nejnižší výšky, naopak bílá barva nám značí bod s nejvyšší výškou. Po té stačí pouze vhodně nastavit měřítko výšky vůči barvě a můžeme reprezentovat libovolný terén.



Obrázek 2.2: Příklad rastrového obrázku ve stupních šedi jako výškové mapy

V minulém odstavci jsme se zabývali terénem, kde jednotlivé body terénu jsou rozmístěny pravidelně. Další možností jak uložit informace o terénu je použití nepravidelných trojúhelníkových sítí **TIN**[3] (anglická zkratka od Triangulated Irregular Network - TIN), popsány v [1]. TIN nám reprezentuje povrch jako soubor trojúhelníků, které jsou tvořeny libovolnými třemi body v prostoru a pro tyto trojúhelníky jsou uchovávány informace o jejich vzájemné poloze - jejich topologická struktura. Tato struktura s sebou přináší jak řadu výhod, tak také nevýhody. Příklad terénu reprezentovaného pomocí TIN je na obrázku 2.3.

Dle [13] přináší TIN řadu výhod oproti pravidelným rastrovým reprezentacím:

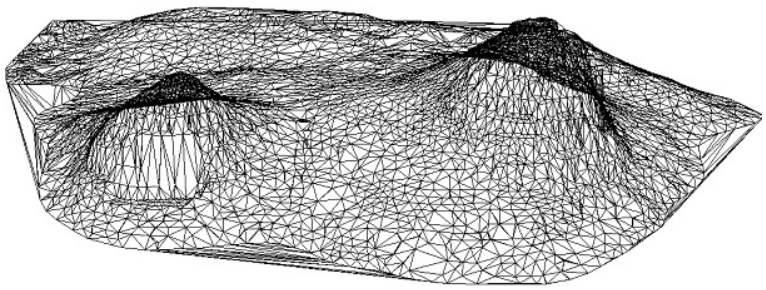
- zmenšení objemu uložených údajů při reprezentaci nehomogenních povrchů
- větší přesnost a věrnost pro nehomogenní povrchy
- kompatibilita s moderními grafickými kartami (které podporují 3D zobrazení v reálném čase)

Ale samozřejmě také nevýhody:

- velká složitost datové struktury a algoritmů s ní pracujících

[2] v anglických textech označováno jako **heighmap**

[3] v naší práci budeme používat zkratku TIN místo termínu nepravidelná trojúhelníková síť



Obrázek 2.3: Příklad terénu reprezentovaného pomocí TIN

2.2 Příprava dat - triangulace

V dnešní době grafické karty vykreslují objekty pomocí jednoduchých primitiv, a to nejčastěji pomocí trojúhelníků. Zjednodušeně můžeme říci, že nejdůležitější operací pro grafickou kartu je tedy operace vykreslení trojúhelníku spolu s výpočtem jeho viditelnosti vůči ostatním objektům vzhledem k aktuálnímu pohledu - kameře.

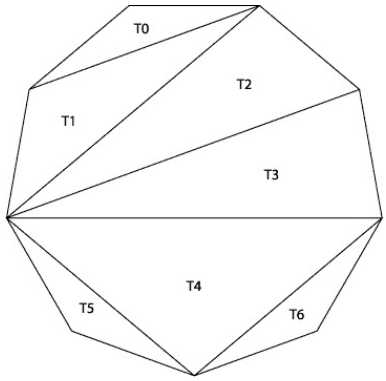
Jak vyplývá ze zadání, vstupem naší práce není pouze výšková mapa terénu, ale je to vektorově zadaný vstup terénu spolu s jeho atributy - typ povrchu, typ porostu, hustota atd. Původně byly na vstupu vektorově zadané polygony spolu s atributy. Protože je ale obtížné vytvořit dostatečně velkou mapu tvořenou polygony pro testování a zobrazení, použili jsme na vstup čtvercovou síť. Tuto vektorovou podobu sítě jsme vygenerovali z rastrového obrázku viz sekce 2.1. Při generování byla tato síť doplněna atributy o typu terénu. Samotným generováním vstupních dat se budeme dále zabývat v kapitole 4.

Triangulace je proces, při kterém dochází k vytvoření množiny trojúhelníků z většího geometrické obrazce. Obecně to může být libovolný polygon, v našem případě je to čtvercová síť, což můžeme brát jako speciální případ polygonu.

Použijeme jednoduchý algoritmus triangulace zveřejněn v [2] a v [3]. Na vstupu máme dánu množinu V ($|V| = N$) bodů, dále pak množinu E vzájemně se neprotínajících hran. Triangulace je proces, při kterém jsou do planárního[4] grafu reprezentujícího polygon přidávány vzájemně se neprotínající hrany. Tento krok opakujeme do té doby, než se nám graf rozpadne na konečný počet trojúhelníkových oblastí.

V naší práci si k vůli rychlosti a celkové složitosti zpracování vstupních dat dovolíme počítat s tím, že na vstup tohoto algoritmu dostáváme polygon monotónní. Nebo-li polygon je monotónní vzhledem k přímce l, pokud je jednoduchý a lze ho rozdělit na 2 souvislé řetězce hran, z nichž každý je monotónní vzhledem k l. Za přímku l budeme pro naše účely považovat přímku y.

[4] planární graf = rovinný graf, pro který existuje takové rovinné nakreslení, že se žádné hrany nekříží



Obrázek 2.4: Příklad triangulace polygonu

Protože je polygon monotónní vzhledem k ose y souřadného systému, lze vrcholy polygonu v lineárním čase seřadit podle y-ové souřadnice. Vrcholy jsou tedy seřazeny do posloupnosti v_1, v_2.. v_n.

Triangulační algoritmus zpracovává v daném čase vždy jeden vrchol postupně podle klesající y-ové souřadnice a jsou jím generovány diagonály. Každá diagonála ohraničuje (uzavírá) jeden z trojúhelníků výsledné triangulace. Ten tvoří vždy daná diagonála spolu s minimálně jednou hranou polygonu P. Z toho vyplývá, že triangulací nevznikají uvnitř planárního polygonu další uzly.

Algoritmus používá jako hlavní datovou strukturu zásobník, do kterého ukládá již zpracovávané body avšak ještě nespojené diagonálou. Přístupný je jak vrchol, tak dno zásobníku. Jeho obsah $[v_1, v_2..v_i]$ se sestává z seřazené posloupnosti vrcholů polygonu, pro něž platí, že $y(v_1) > y(v_2) > ... > y(v_i)$ a pokud $i \geq 3$, pak úhel $(v_j, v_{j+1}, v_{j+2}) \geq 180°$ pro všechna j=1...i-2.

Algoritmus:

1. Dva vrcholy s nejvyšší y-ovou souřadnicí uložíme na zásobník.
2. Označme u právě zpracovávaný vrchol.
3. Postupně dosazujeme za u všechny vrcholy které ještě nebyly zpracovávány. Začínáme třetím nejvyšším vzhledem k ose y.
   (a) Pokud u hranově sousedí s v_1 (dno zásobníku) a nesousedí s v_i (vrchol zásobníku) pak:
      i. Přidáme diagonály [u, v_2] až [u, v_i].
      ii. Odstraníme obsah zásobníku a uložíme na něj u a v_i.
   (b) Pokud u hranově sousedí s v_i (vrchol zásobníku) a nesousedí s v_1 (dno zásobníku) pak:

i. Dokud (i>1) a (úhel (u, v_j, v_{j-1}) < 180°), přidáme diagonálu [u, v_{i-1}] a vyjmeme v_i ze zásobníku.

ii. Poté přidáme u do zásobníku.

(c) Pokud u hranově sousedí jak s v_1 (dno zásobníku) tak i s v_i (vrchol zásobníku) pak:

i. Přidáme diagonály [u, v_2] až [u, v_{i-1}].

ii. Ukončíme výpočet.

4. Ukončíme algoritmus.

Dále může docházet k dalším úpravám trojúhelníkové sítě. Například velká rovná plocha nemusí být zbytečně složena z mnoha trojúhelníků, může dojít k jejímu zjednodušení, neboli zmenšení počtu trojúhelníků, aniž by došlo ke kvalitativním ztrátám.

2.3 Možnosti zobrazování terénů

Jak jsme se již zmínili v předchozí části, v dnešní době jsou nejpoužívanější grafické karty (akcelerátory), které umožňují rychlé vykreslování těles popsaných pomocí hraniční reprezentace. To je taková reprezentace, kdy je těleso popsáno svým povrchem, který může mít kromě geometrických vlastností zadány například i další vlastnosti povrchu (barvu, texturu, průhlednost, odrazivost, apod.). Tyto tělesa jsou vykreslovány pomocí jednodušších primitiv. Odtud nám tedy zjednodušeně vyplývá, že nejdůležitější činností grafické karty je zobrazování trojúhelníků s řešením vzájemného překrývání podle umístění kamery - podle pohledu na scénu.

S jednoduchým zobrazováním trojúhelníků bychom si ovšem nevystačili. Každý trojúhelník může být kreslen jinou barvou, osvětlován jedním nebo dokonce více světelnými zdroji, což má za následek další výpočty a operace, a tudíž zvýšené požadavky na grafické karty. Současné grafické karty jsou schopné samozřejmě zobrazovat složité objekty popsané obecnými polygony. Při jejich zobrazování ale musí grafická karta provést teselaci[5], triangulaci a další operace, než je schopna požadovaný objekt vykreslit. Tyto operace jsou časově náročné, vykreslení takového objektu trvá delší dobu. Proto při zobrazování rozsáhlých terénu je nutné předkládat grafické kartě data, která jsou již téměř připravena pro okamžité kreslení.

V našem projektu se může vyskytovat několik typů terénu jako například pole, tráva, cesta, atd. Proto nejprve dochází k triangulaci a následnému zpracování vstupních dat (dále se tímto problémem budeme zabývat v kapitole o implementaci). Jedná se o to, že soubor trojúhelníků s atributy, které dohromady tvoří popis celého terénu, jsou uloženy v jednom poli. Tedy při vykreslování je nutné stéle přepínat typ textury pro každý trojúhelník a vykreslování trvá neúnosně dlouho. Pro náš úkol, tedy nejenom modelování, ale i procházení terénu, je nutné co nejvíce urychlit vykreslování. Proto provedeme sjednocení trojúhelníků se shodným typem textury do

[5]teselace je jednoduché rozdělení povrchu objektu na síť (trojúhelníků, čtyřúhelníků, apod.)

stejných polí. Po tomto kroku stačí grafické kartě předat jednotlivá pole trojúhelníků s typem textury. Výsledné vykreslení je mnohem rychlejší.

Při kreslení terénu si nevystačíme pouze s jednoduchou operací vykreslení trojúhelníku s určitou texturou, ale je třeba provádět i složitější operace s vykreslovanými tělesy. Je to například kreslení průhledných nebo poloprůhledných objektů (tzv. blending). Tyto operace jsou potřebné pro kreslení vody v naší scéně. Problematikou průhlednosti textur (blendingem) se budeme zabývat v sekci 3.4.

Kapitola 3

Modelování a zobrazování objektů v ekosystému

V minulé kapitole jsme se zabývali modelováním terénu a jeho zobrazováním. Protože našim úkolem je modelovat celý ekosystém, musíme se věnovat také objektům, které se ve scéně mohou nacházet.

3.1 Typy objektů v terénu

Nejenom našim cílem, ale cílem všech aplikací zabývajících se modelováním, je, aby výsledná zobrazovaná scéna vypadala co nejpěkněji, nejdůvěryhodněji. Proto musí použité modely pro reprezentaci stromů, keřů a dalších porostů vypadat co nejrealističtěji. Musíme řešit rozpor mezi co nejpěkněji vypadající scénou, kde jeden stromu může obsahovat potencionálně až několik desítek tisíc vertexů[1], a rychlostí zobrazování scény. Bohužel nejen v naší simulaci, ale i dalších projektech, které jsou založeny na interaktivní spolupráci s uživatelem, takové hodně složité objekty použít nelze. Je to možné pouze v případech, kdy klientovi - uživateli ukazujeme výslednou zobrazenou scénu (např. vymodelovanou zahradu, model rodinného domu). V tomto případě si můžeme dovolit, aby se výsledný obrázek generoval i několik sekund, na čase nám v tomto případě nezáleží.

Snahou je udělat náš projekt "univerzální", ve smyslu frameworku[2]. Jde nám o takovou univerzálnost, abychom byli schopni v projektu načítat jak hotové 3DS modely, tak vytvářet jednoduché objekty pomocí billboardů. Struktura programu má být připravena na rozšíření o další typy porostů, typy textur atd. Změna 3D modelu musí být otázkou pouze jednoduché úpravy konfiguračního souboru. V našem projektu používáme 2 typy objektů:

[1] vertexem je označován v počítačové grafice jeden bod

[2] framework je softwarová struktura, která slouží jako podpora při programování a vývoji jiných projektů

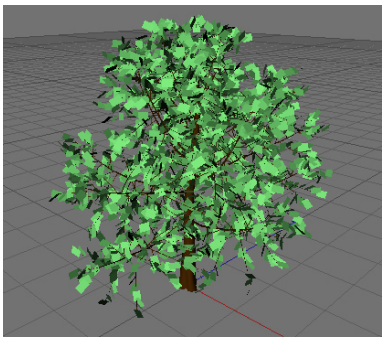
- objekty ve formátu 3DS (hotový 3D objekt uložen v jednom souboru)
- jednoduché objekty vykreslované pomocí billboardů (billboardingu)

3.1.1 Objekty ve formátu 3DS

Projekt umožňuje načítání a zobrazování objektů ve formátu 3DS. Jedná se o formát komerčního programu 3D Studio Max. Vybrali jsme si právě tento formát, protože se na webu nachází větší množství volně přístupných modelů. Samozřejmě není problém provést konverzi do jiného grafického formátu. 3DS je grafický formát pro uložení popisu kompletního 3D modelu, tzn. nejenom popisu tvaru, ale také dalších vlastností jako je pokrytí texturou, nastavení vlastností materiálu atd., viz obrázek 3.1.

Každý 3DS soubor se skládá dle [16] z jednotlivých bloků (chunks). Tyto bloky v sobě uchovávají veškeré informace o modelu. Blok vždy obsahuje identifikátor, svoji délku a vlastní data. Některé bloky slouží pouze jako kontejnery a obsahují větší či menší počet jiných bloků. Takové bloky je možné díky znalosti jejich délky přeskočit.

Výhodou používání 3DS formátu je celková jednoduchost práce s ním. Model můžeme jednoduše načíst, nastavit některé jeho parametry a zobrazit (viz dále v kapitole implementace 4). Nevýhodou pro účely naší práce je dosti velká složitost modelů pro zobrazování. Například strom nelze jednoduše popsat pomocí několika vertexů, obvykle ho tvoří stovky až tisíce vertexů. Tyto objekty jsou pak hodně náročné na vykreslování.



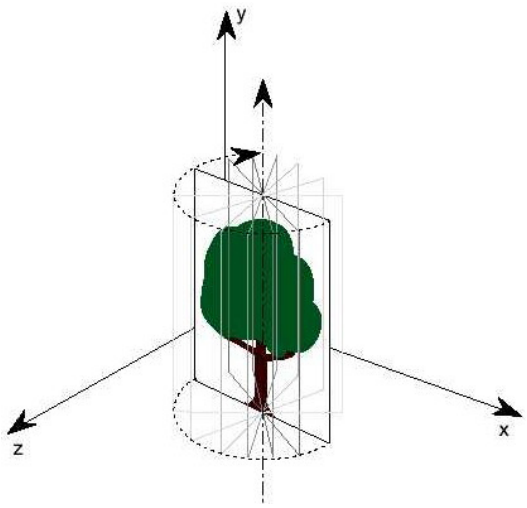
Obrázek 3.1: 3DS model stromu zobrazen v programu Cinema 4D

3.1.2 Objekty vykreslované pomocí billboardingu

Jedním ze způsobů, jak realizovat objekty ve scéně, kde je důležitá rychlost vykreslování, je použití billboardů. Billboard může být jednoduchý geometrický útvar (nejčastěji obdélník), na který je namapována fotorealistická textura. Takto vykreslený objekt vypadá ve scéně mnohdy lépe než jednoduchý 3D model. Billboard

stromu tvořený obdélníkem obsahuje pouze 4 vertexy. Ale vytvoření 3D modelu stromu s použitím několika málo vertexů je téměř nemožné.

Libovolné procházení scény s sebou přináší ale problém. Jestliže bychom kreslili billboard na pevné souřadnice v terénu, může se stát, že při procházení scény narazíme místo na čelní plochu billboardu na tenkou linku profilu. Tento problém řeší tzv. **billboarding**, neboli natáčení čelních stran polygonů ke kameře. Při každém pohybu avatara (vizuální reprezentace uživatele ve virtuálním světě) musí dojít k výpočtu nových souřadnic billboardu tak, aby byl opět správně natočen ke kameře. Tento problém je podrobně popsán v [18]. Příklad billboardingu je na obrázku 3.2.



Obrázek 3.2: Ukázka billboardingu - natáčení stromu ke kameře

Nyní si podrobněji popíšeme postup billboardingu - výpočtu nových souřadnic objektu. Detailnější popis k tomuto výpočtu nalezneme na obrázku 3.3.

Označme si následovně (všechny body jsou v 3D prostoru, tzn. mají 3 souřadnice):

- width - šířka billboardu
- height - výška billboardu
- cam - bod, umístění kamery
- pos - bod, umístění aktuálního objektu, který natáčíme
- akt - vektor směřující od kamery k aktuálnímu objektu
- up - standartní up vektor
- mul - pomocný vektor

Nejprve si vypočítáme vektor akt - vektor směřující od kamery k aktuálnímu objektu.

$$\begin{aligned} \text{akt.x} &= \text{cam.x - pos.x} \\ \text{akt.y} &= \text{cam.y - pos.y} \\ \text{akt.z} &= \text{cam.z - pos.z} \end{aligned}$$

Standartní vektor up již máme normalizovaný.

$$\begin{aligned} \text{up.x} &= 0 \\ \text{up.y} &= 1 \\ \text{up.z} &= 0 \end{aligned}$$

Nyní provedeme vektorový součin vektorů akt a up - výsledný vektor je kolmý k oběma vstupním.

$$\begin{aligned} \text{mul.x} &= akt.y \cdot up.z - akt.z \cdot up.y \\ \text{mul.y} &= akt.z \cdot up.x - akt.x \cdot up.z \\ \text{mul.z} &= akt.x \cdot up.y - akt.y \cdot up.x \end{aligned}$$

Po té tento vektor vynásobíme poloviční šířkou billboardu.

$$\begin{aligned} \text{mul.x} &= mul.x \cdot (width/2) \\ \text{mul.y} &= mul.y \cdot (width/2) \\ \text{mul.z} &= mul.z \cdot (width/2) \end{aligned}$$

Standartní up vektor vynásobíme poloviční výškou billboardu.

$$\text{up.y} = up.y \cdot (height/2)$$

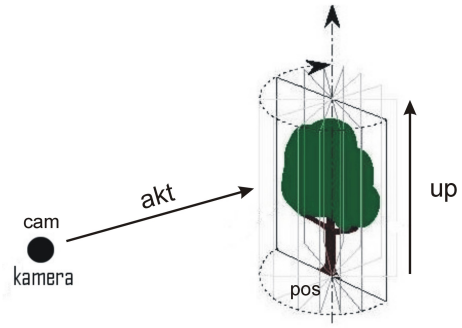
Výsledná pozice billboardu je tedy rovna následujícím souřadnicím:
Souřadnice dolního levého rohu billboardu pak jsou:

$$\begin{aligned} x_1 &= \text{pos.x} + (\text{- mul.x - up.x}) \\ y_1 &= \text{pos.y} + (\text{- mul.y - up.y}) \\ z_1 &= \text{pos.z} + (\text{- mul.z - up.z}) \end{aligned}$$

Souřadnice dolního pravého rohu billboardu pak jsou:

$$\begin{aligned} x_2 &= \text{pos.x} + (+ \text{ mul.x - up.x}) \\ &\ldots \end{aligned}$$

Stejným postupem bychom získali i y-ové a z-ové souřadnice. Souřadnice horních rohů obdélníkového billboardu získáme opětovným postupem, pouze bychom zaměnili znaménko u vektoru up.



Obrázek 3.3: Vysvětlení výpočtu billboardingu

3.2 Modelování rozmístění objektů

Jak již bylo nastíněno v úvodu práce, je nereálné, aby vstupní data terénu obsahovaly pozice každého stromu, keře či jiného objektu. Byly by ohromně velké a zpracování by trvalo neúnosně dlouho. Generování pozic objektů je právě úkolem naší simulace.

Ve vstupních datech má každý polygon (v naší práci je to čtvercová síť) vlastnosti - atributy.

Tabulka 3.1: Význam atributů terénu ve vstupním souboru

atribut	**význam**
TEXTURE	určuje typ textury, která se bude mapovat
TYPE	udává typ porostu (smrk, borovice, buk, křoví, atd.)
DENSITY	udává hustota porostu
SIZE	určuje velikost porostu (stáří)

Návrh algoritmu pro rozmísťování objektů po terénu byl jednou z nejdůležitějších částí projektu. Hlavním problémem je opravdu termín reálného ekosystému, tzn. implementovat takový algoritmus, aby byl výsledek přijatelný nejen pro autora, ale i ostatní uživatele - pozorovatele. Dalším omezením je samotný terén, který dostáváme na vstup. Obecně nemusí mít žádný předem známý tvar, může to být opravdu libovolná trojúhelníková síť.

Na vstup algoritmu tedy dostáváme triangulované polygony s parametrem hustota. Našim úkolem je vypočítat souřadnice objektů, které budeme později vykreslovat. Celý úkol můžeme rozdělit na jednotlivé části:

1. Na vstup dostáváme triangulovaný polygon, takže výpočet obsahu celého polygonu je snadnou záležitostí. Stačí nám sečíst obsahy jednotlivých trojúhelníků,

které vypočteme podle Heronova vzorce

$$S = \sqrt{s(s-a)(s-b)(s-c)} \tag{3.1}$$

kde pro malé s ve vzorci platí:

$$s = \frac{a+b+c}{2} \tag{3.2}$$

2. Každý trojúhelník v síti obsahuje své atributy. Kromě samozřejmých 3 bodů, kterými je trojúhelník definován, je to pseudonáhodné číslo. Potřebuje vnést do simulace určitý náhodný prvek tak, abychom zamezili pravidelným výsledkům. K tomuto účelu by nám stačilo vygenerovat úplně náhodné číslo a po té ho zakomponovat do našeho výpočtu. My ale potřebujeme, aby rozmístění stromů bylo při každém spuštění simulace na stejný vstupní terén shodné. Jestliže se při procházce lesem vrátíme po hodině cesty nato samé místo, očekáváme přirozeně, že stromy budou růst na těch samých místech. K tomu účelu nám právě slouží pseudonáhodné číslo. My jsme k jeho výpočtu zvolili metodu, kdy sečteme všechny souřadnice příslušného trojúhelníku a výsledek modulíme určitou konstantou. Tuto konstantu zvolíme podle toho, z jakého intervalu pseudonáhodné číslo očekáváme.

3. Podle atributu DENSITY - hustota porostu (význam dalších atributů uveden v tabulce 3.1) a obsahu polygonu vypočteme příslušný počet stromů, které se v něm mají nacházet. Obsah polygonu si převedeme vhodným měřítkem na číslo, které vynásobíme příslušnou hodnotou hustoty.

4. Dále podle pseudonáhodného čísla rozdělíme stromy do jednotlivých trojúhelníků polygonu. Tímto způsobem je řešena i možnost, že by počet sázených stromů byl menší než je počet trojúhelníků. Jestliže máme například vysázet 4 stromy do polygonu tvořeného 10 trojúhelníky, jsou pseudonáhodně zvoleny 4 trojúhelníky, kam budou stromy zasazeny.

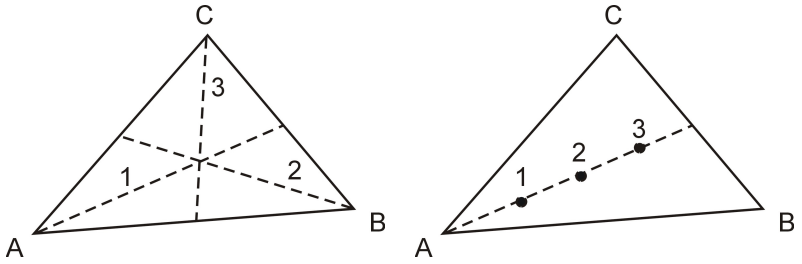
5. Nyní provedeme sesortování - seřazení trojúhelníku podle z-ové souřadnice. V případě rovnosti těchto souřadnic bude brán jako druhý pomocný parametr x-ová souřadnice. Jestliže si představíme terén jako obdélník, budeme objekty "sázet" odzadu směrem doprava. Význam tohoto sortování je zabránění vzniku shluků - stromy nerostou přímo vedle sebe. Podrobnější vysvětlení podáme v následujících odstavcích.

6. Postupně procházíme všechny trojúhelníky v síti a rozmísťujeme stromy:

   - Pokud je počet stromů v trojúhelníku roven 0, žádný strom nesadíme, pokračujeme dalším trojúhelníkem.
   - Pokud máme vysadit pouze jeden objekt v trojúhelníku, můžeme zvolit jednu z 9 možností, kam strom zasadit. Máme na výběr ze 3 těžnic trojúhelníka, viz obrázek 3.4(a), těžnice je volena podle pseudonáhodného

čísla. Abychom ještě více zvětšili možnosti rozmísťování stromů a různorodost krajiny, můžeme si dále opět podle pseudonáhodného čísla vybrat 3 místa na těžnici, kde už bude konečná pozice našeho objektu (viz obrázek 3.4(b)).

Nyní přicházíme k důvodu, proč jsme na začátku algoritmu sortovali trojúhelníky. Již jsme sice vypočítali novou pozici pro umístění objektu, ale ten se může nacházet blízko jiného, který jsme zasadili již předtím - tím pádem by došlo ke vzniku shluku. Proto se stačí podívat do struktury, kam ukládáme nové pozice stromů, a zkontrolovat vzdálenosti od několika posledních stromů. Slovo několik má význam konstanty, kde si předem stanovíme, kolik vzdáleností od objektů budeme kontrolovat. Vzdálenost počítáme klasickým způsobem jako vzdálenost 2 bodů v prostoru a porovnáváme ji s konstantou VZDALENOST-SHLUK. Pokud jsou všechny naše kontrolované vzdálenosti větší než tato konstanta, pak je to v pořádku a můžeme směle přidat nový strom do struktury. Pokud je ale alespoň jedna ze vzdáleností menší než konstanta, musíme provést určitá opatření. Buď můžeme objekt úplně vynechat nebo nastavíme jeho atribut velikost na malý (atribut SIZE).



(a) Možnost výběru jedné z těžnic, na kterou budeme objekt sázet

(b) Další možnosti umístění objektu na těžnici

Obrázek 3.4: Možnosti rozmístění objektů v trojúhelníku

- Pokud je počet stromů, které máme rozmístit v trojúhelníku větší než 1, zavoláme tuto metodu sázení rekurzivně na 2 trojúhelníky (na každý s poloviční počtem stromů). Tyto 2 trojúhelníky získáme rozdělením trojúhelníka podle těžnice (obrázek 3.4(a)). Těžnici získáme opět podle pseudonáhodného čísla.

3.3 Techniky LOD

Náš projekt má za cíl nejen modelování terénu, ale také jeho procházení v reálném čase. To s sebou přináší zvýšené nároky na výpočetní kapacitu. Proto je naší snahou se také zabývat technikami optimalizace. Jednou z nich je technika LOD (Level Of Detail - úrovně detailů) popsána v [1]. Tato metoda je založena na poznatku, že ne

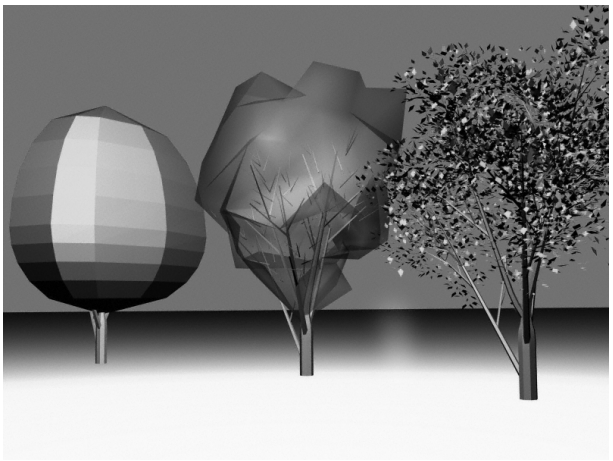
všechny objekty musí být kresleny v co nelepší kvalitě, co nejpěkněji. Při vykreslování platí samozřejmě fakt, že čím je model složitější, tedy obvykle pěknější, tím déle trvá jeho vykreslení. Pomocí techniky LOD vykreslujeme blízké předměty v co nejlepší kvalitě, naopak ty nejvzdálenější v co nejhorší. Nemusíme používat pouze 2 úrovně detailů, to už záleží na konkrétní implementaci techniky LOD.

V našem projektu používáme 3 úrovně detailů. Tuto techniku jsme realizovali následujícím způsobem. Při každém pohybu avatara dojde k výpočtu vzdáleností objektů od kamery (avatara). Tuto hodnotu vypočítáme jako vzdálenost dvou bodů (bod A, bod B) v prostoru dle vzorce 3.3.

$$|AB| = \sqrt{(A_x - B_x)^2 + (A_y - B_y)^2 + (A_z - B_z)^2} \tag{3.3}$$

Předem jsme si již stanovili vzdálenostní konstanty pro jednotlivé úrovně detailů LODu. Po té stačí pouze porovnávat vypočtenou vzdálenost s našimi konstantami a vykreslovat modely objektů příslušné úrovně. Můžeme použít i omezení, že pokud je vzdálenost větší než námi zvolená konstanta viditelnosti, je úplně zbytečné objekt kreslit. Uživatel by ho stejně viděl jako malou tečku a jeho vykreslení by zabralo zbytečně příliš mnoho výpočetní kapacity.

Realizace jednotlivých stupňů úrovní modelu může být různá, opět záleží na konkrétní implementaci této techniky. U 3DS modelů jsme úrovně řešili tak, že například u modelu jehličnatého stromu jsme odebrali celou jednu úroveň větví. Tento postup je naznačen na obrázku 3.5.



Obrázek 3.5: LOD aplikovaný na model stromu - 3 úrovně detailů

Obdobným způsobem můžeme úrovně řešit LOD i u objektů vytvářených pomocí billboardů.

3.4 Zobrazování objektů

Nyní již máme rozmístěny všechny objekty a jejich pozice uloženy v datové struktuře. Jestliže ale pro zobrazování objektů používáme billboardy, tzn. jednoduché geometrické obrazce potažené texturou, musíme pro reálný vzhled použít alfa - blending nebo-li alfa průhlednost.

Alfa kanál[3] nám definuje průhlednost textury. Nejlépe lze alfa průhlednost demonstrovat na barevném modelu RGBA, kde R(red) B(green) B(blue) jsou barevné kanály obrázku a A je kanál nesoucí informaci o průhlednosti. Alfa kanál může být jedno bitový, pak je (ne)průhlednost absolutní, tzn. že je pixel buď 100% průhledný nebo neprůhledný. Pokud nám tento alfa kanál nedostačuje, můžeme pro něj použít větší bitové rozlišení. V našem případě používáme jako alfa kanál obrázek ve stupních šedi. V místech, kde je tento obrázek černý, je textura průhledná. Naopak v bílých oblastech obrázku alfa kanálu je textura zcela neprůhledná. My používáme obrázky s alfa kanálem jako textury billboardových stromů. Ukázku textury stromu pro billboarding nalezneme na obrázku 3.6(a), příslušný alfa kanál na obrázku 3.6(b).

(a) Textura stromu, kterou budeme mapovat na billboard

(b) Příslušný alfa kanál textury stromu

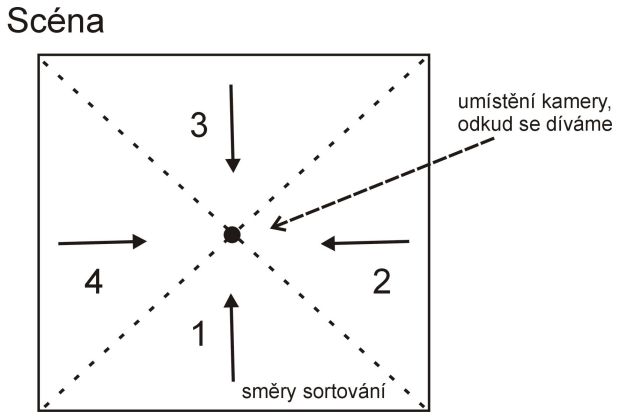
Obrázek 3.6: Textura a její alfa kanál

Díky používání blendingu - průhlednosti při vykreslování objektů musíme řešit další významný problém. Tím je pořadí vykreslovaných objektů. Jak jsme se již zmínili dříve v kapitole o kreslení terénu, nejdůležitější činností grafické karty je zobrazování trojúhelníků s řešením vzájemného překrývání dle umístění kamery - podle pohledu na scénu. Pokud bychom kreslili objekty od nejbližšího k nejvzdálenějšímu, karta by začala kreslit blízký předmět, jehož některé části mohou být průhledné. Ovšem za ním se ještě nenachází žádný jiný předmět, tudíž v průhledných místech bude vykreslen správně terén, který se za objektem nachází. Dále jsou kresleny objekty za ním, které ale již nemusí být vidět (již jsme nakreslili nejbližší objekty s jejich

[3] v anglických textech označováno jako alpha channel nebo alpha transparency channel

průhlednými místy). Proto je nutné objekty vykreslovat ve správném pořadí, a to od nejvzdálenějších po nejbližší.

Díky tomuto problému musíme v našem projektu používat sortování objektů podle vzdálenosti. Toto řazení je ale relativní. Jestliže projdeme scénou určitou vzdálenost jedním směrem, kde budou objekty vykreslovány ve správném pořadí, a nyní se otočíme a půjdeme zpět, dojde k porušení zásady vykreslování. Z tohoto důvodu používáme v našem projektu sortování ve 4 základních směrech.



Obrázek 3.7: Sortování objektů ve scéně vůči pohledu kamery - 4 základní směry

Obrázek 3.7 nám názorně ukazuje směry sortování objektů. Naše kamera je umístěna ve středu scény. Podle směru pohledu kamery do scény (my rozlišujeme 4 základní směry) se objekty řadí ve správném pořadí. K vůli rychlosti vykreslování objektů máme v projektu již přichystané předsortované 4 struktury. Při procházení scény nám tedy stačí zjistit, jakým směrem se díváme a vykreslit objekty ve správném pořadí.

Kapitola 4

Implementace

Našim úkolem je vytvořit grafickou aplikaci, která nám na základě vstupních dat vygeneruje scénu, kterou pak bude možné libovolně procházet. Vstupní data jsou zadána ve vektorové podobě v upraveném formátu X3D. Díky dobré znalosti programovacího jazyku C++ a jeho velmi častému použití v grafických aplikací jsme zvolili právě tento jazyk. Jako grafické rozhraní (API)[1] pro zobrazení výsledné 3D scény jsme zvolili OpenGL.

4.1 Zpracování vstupních dat

Jednou z důležitých částí aplikace je zpracování vstupních dat. Data jsou zadána ve vektorové podobě ve formátu X3D, viz dále [4]. Jedná se o formát pro popis prostorové scény. Je to standard pro definici interaktivní, animované 3D grafiky zahrnující další média jako zvuk, video a hypertext. Je vyvíjen konsorciem Web3D, vznikl postupem času z formátu VRML, X3D můžeme nazvat jeho následovníkem. Data v tomto formátu jsou zapisována formou XML dokumentu.

V našem úkolu se projevila ale také stinná stránka tohoto formátu. Při popisu velkých scén rychle stoupá velikost souboru. Je to především díky velkému množství nevyužitých dat, což jsou hlavně XML tagy. Jestliže jsme si nechali vygenerovat "velkou" krajinu, velikost tohoto souboru již nebyla únosná (10-tky MB). Proto jsme sice zachovali způsob ukládání dat ve formátu X3D, ale zvolili jsme úspornější řešení. Vynechali jsme nadbytečné tagy a souřadnice jsme nechali zapisovat do souboru přímo bez dalších komentářů. Současná podoba vstupní sítě má velikost "pouze" 2 MB. Příklad obou variant uložení dat je na obrázcích 4.1 a 4.2.

Nyní se konečně dostáváme k samotným vstupním datům. K vůli dostatečnému otestování projektu a testu všech možností typů krajin a různých hustot porostů jsme si museli generovat data sami. Jako součást projektu tedy vznikl i **generátor vstupních dat**. Ten na základě 2 vstupních obrázků vygeneruje úsporné uložení čtvercové sítě do formátu X3D. Na vstup dostáváme tedy 2 obrázky. Zaprvé se jedná o výškovou mapu reprezentovanou obrázkem ve stupních šedi (viz kapitola 2.1). Druhý obrázek

[1] angl. zkratka pro application programming interface - rozhraní pro programování aplikací

```
<Shape>
        <Appearance>
                <Material texture='0' type='0' density='0' />
        </Appearance>
        <IndexedLineSet coordIndex='0 1 -1 1 2 -1 2 3 -1 3 0 -1'>
                <Coordinate point='
                        0 -2110 0
                        0 -2250 3000
                        3000 -3090 3000
                        3000 -3020 0
                '/>
        </IndexedLineSet>
</Shape>
<Shape>
        <Appearance>
                <Material texture='1' type='0' density='2' />
        </Appearance>
        <IndexedLineSet coordIndex='0 1 -1 1 2 -1 2 3 -1 3 0 -1'>
                <Coordinate point='
                        0 -2250 3000
                        0 -1060 6000
                        3000 -430 6000
                        3000 -3090 3000
                '/>
        </IndexedLineSet>
</Shape>
```

Obrázek 4.1: Uložení dat v klasickém x3d souboru

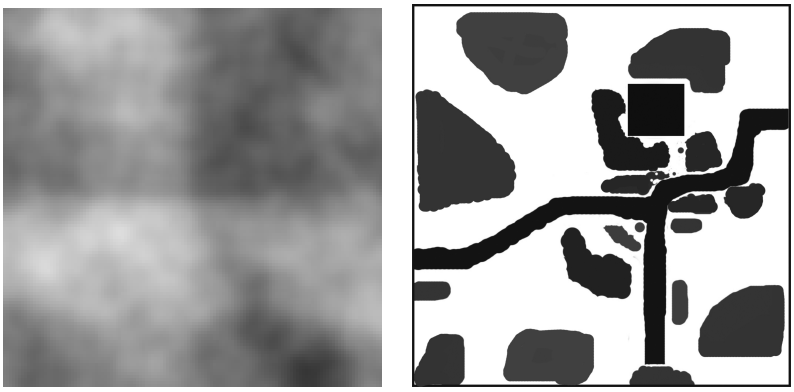
```
<Shape>
        <Squares all='
                texture= 0  type= 1  density= 1
                Coordinate_point=
                        0 -11660 0
                        0 -12080 15000
                        15000 -14180 15000
                        15000 -13760 0
                texture= 0  type= 0  density= 0
                Coordinate_point=
                        0 -12080 15000
                        0 -12500 30000
                        15000 -14600 30000
                        15000 -14180 15000
        ' />
</Shape>
```

Obrázek 4.2: Uložení dat v úsporném x3d souboru

je také ve stupních šedi a reprezentuje nám porostovou mapu - obrázek 4.3(b). Jednotlivé odstíny šedi nám reprezentují jiný typ porostu. Generátor tedy funguje na takovém principu, že současně procházíme obě vstupní mapy. Na základě získaných dat (výška bodů z prvního obrázku a typ porostu z druhého obrázku) generujeme vektorovou podobu dat spolu s atributy.

4.2 Moduly programu

Celou aplikaci jsme se snažili psát univerzálně, ve smyslu frameworku. Program by měl být jednoduše rozšiřitelný. Přidání dalšího typu porostu nesmí znamenat velké zásahy do projektu. Výměna modelu použitého ve scéně je otázkou pouze nastavení cesty k souboru v konfiguračním souboru. Celý program můžeme rozdělit



(a) Výšková mapa terénu (b) Porostová mapa terénu

Obrázek 4.3: Reprezentace vstupních map

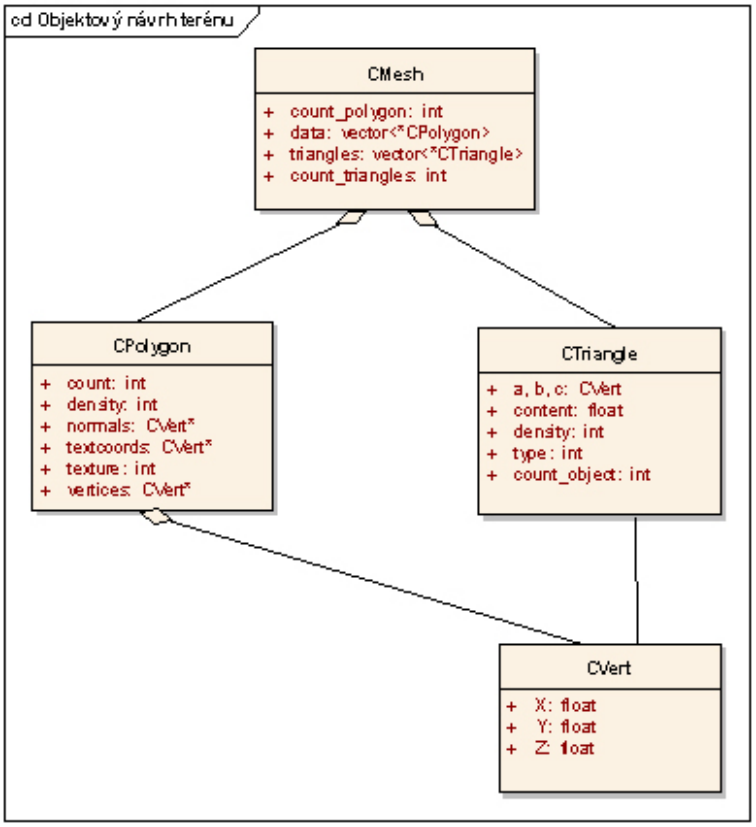
na jednotlivé komponenty (třídy).

Třída CMesh

Třída CMesh nám reprezentuje celý terén. Obsahuje veškerá data terénu, to jak zpracovanou trojúhelníkovou síť s atributy, tak i data připravená rovnou pro vykreslování. Mezi nejdůležitější metody této třídy patří *LoadX3D(...)*, která načte vstupní data z X3D souboru. Dále uvnitř této metody pro načítání X3D souboru voláme metodu *Triangulation(...)*, která nám provede triangulaci polygonů a následně metodu *CreateData(...)*, která vytvoří strukturu trojúhelníků s atributy. Jak jsme se již zmínili, formát tohoto souboru vychází z ukládání dat ve formátu XML. Proto v této metodě využíváme XML parser.
V projektu jsme použili volně stažitelný XML parser popsán v [7]. Podle licence se smí volně používat vyjma komerčních účelů. Dle [7] se jedná o jednoduchý XML parser napsaný v jazyce C++. Tento parser nejprve načte celý soubor do paměti a až po té vytváří stromovou strukturu reprezentující (XML) soubor. Má implementovány základní operace pro práci se XML strukturou - jak pro čtení, tak pro editaci i vytváření nových struktur.
Zavoláním metody *RenderQuickly(...)* můžeme celý výsledný terén zobrazit. Objektový návrh terénní sítě nalezneme na obrázku 4.4.

Třída CAvatar

Jedná se o samostatnou nezávislou třídu, která řídí veškerý pohyb avatara ve scéně. Tzn. zajišťování veškerého pohybu a natáčení kamery. V projektu jsme umožnili avatarovi - uživateli neomezený pohyb ve scéně. Třída CAvatar zpracovává veškeré vstupní reakce uživatele (stisk klávesy, pohyb myši) a reaguje na ně pohybem



Obrázek 4.4: Objektový návrh terénní sítě

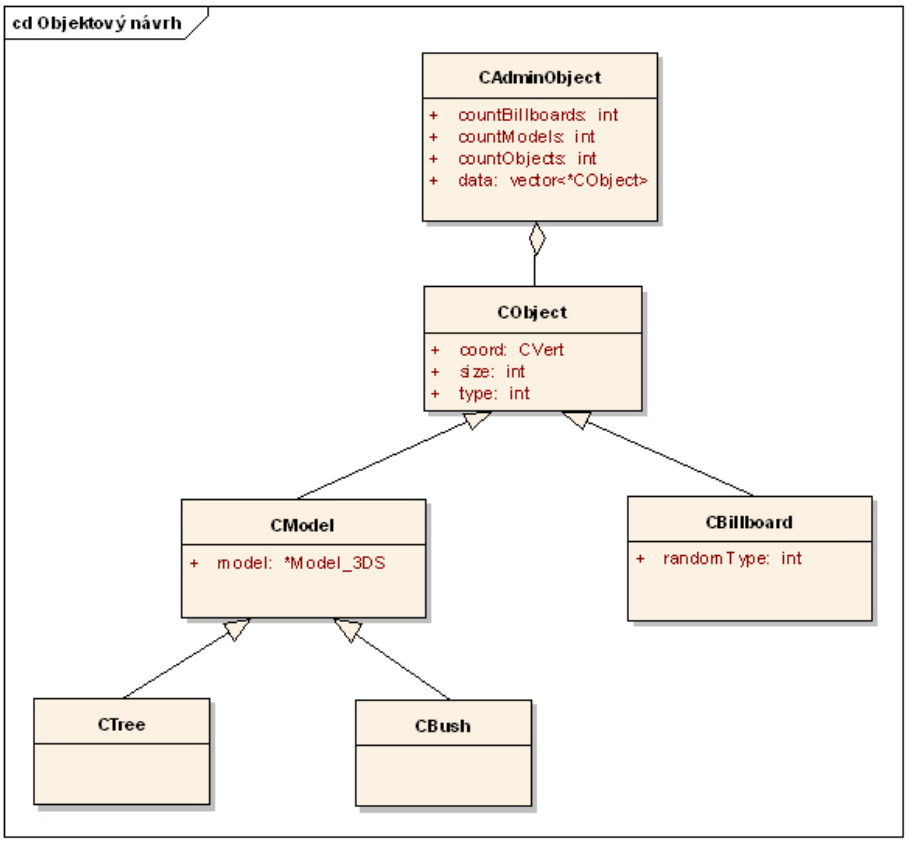
po scéně nebo natočením kamery. V této třídě máme uchovány informace o pozici a aktuálním stavu avatara. Důležité metody pro pohyb avatara:
- *avatarMoveForward(...)* - pohyb dopředu
- *avatarMoveBackward(...)* - pohyb dozadu
- *avatarMoveLeft(...)* - pohyb doleva
- *avatarMoveRight(...)* - pohyb doprava

Poslední metodou je *AvatarMoveView(...)*. Ta slouží k výpočtu pohledu avatara při pohybu myší. Styl pohybu avatara, tzn. reakce na stisk kurzorových šipek nebo pohyb myši je podobný jako v 3D akčních hrách. Jedná se o tzv. typ FPS (first person shooter).

Třída CAdminObject

Tuto třídu můžeme nazvat ekvivalentem třídy CMesh pro objekty. Řídí všechna načítání modelů, přípravu a samotné vykreslování modelů. Hlavními položkami jsou data 3DS modelů a data billboardů. Tuto třídu starající se o všechny operace s objekty se snažíme naprogramovat nezávislou na terénu (na třídě CMesh). Jedna z nejdůležitějších metod této třídy je *PlantingObjects(...)*, která provede vlastní rozmís-

tění objektů po terénu. Tato metoda dostává na vstup trojúhelníkovou síť s atributy, podle které počítá výsledné souřadnice objektů. Další metodou volanou při spouštění aplikace - inicializaci je metoda *InitDraw(...)*, která provede samotné načtení modelů, sesortuje objekty podle potřeby v datových strukturách a připravuje data pro konečné vykreslení. To je zajištěno metodou *Render(...)*, která se stará jak o vykreslování modelů, tak billboardů. V této metodě používáme důležitou metodu pro zjišťování směru vykreslování objektů *CalcQuadrant(...)*, která nám určuje, jakým směrem mají být objekty sesortovány, aby došlo k jejich správnému vykreslení. Problémem sortování objektů a pořadím vykreslování jsme se zabývali v kapitole 3.4. Vlastní objektový návrh třídy CAdminObject a jednotlivých tříd pro objekty nalezneme na obrázku 4.5.



Obrázek 4.5: Objektový návrh správy objektů

4.3 Zobrazování scény - použité knihovny

V naší aplikaci budeme realizovat API mezi programem a grafickým hardwarem pomocí **OpenGL**. Dle [8], [9] a [10] se jedná o nízkoúrovňovou knihovnu pro práci s trojrozměrnou grafikou. Mezi základní znaky OpenGL patří nezávislost na plat-

formě a použitém programovacím jazyku. Knihovna OpenGL vznikla v roce 1992 u společnosti Silicon Graphics Inc. (SGI). Primárně je k dispozici hlavičkový soubor pro jazyky C a C++. OpenGL poskytuje pouze základní rozhraní pro přístup ke grafickým akcelerátorům. V naší aplikaci budeme používat i další rozšiřující knihovny:

Knihovna **GLU** (OpenGL Utility Library), která dle [12] umožňuje využívat tesselátory (rozložení nekonvexních polygonů na trojúhelníky), evaluátory (výpočet souřadnic bodů ležících na parametrických plochách) a vykreslovat kvadriky (koule, válce, kužely a disky).

Knihovna **GLUT** (OpenGL Utility Toolkit) - zatímco knihovna OpenGL je určena především k zobrazování dvourozměrných a trojrozměrných modelů těles i celých scén, knihovna GLUT dle [12] definuje a implementuje aplikační rozhraní pro tvorbu oken a jednoduchého grafického uživatelského rozhraní, přičemž je systémově nezávislá, tj. pro práci s okny se na všech systémech používají vždy stejné funkce, které mají stejné parametry. Pomocí GLUTu ovládáme také vstupní zařízení jako jsou klávesnice a myš.

Dále v projektu používáme volně šiřitelný (dle licence, která je obsažena na CD) **XML parser**. Jedná se o jednoduchý multiplatformní XML parser napsaný v jazyce C++. Tento parser nejprve vytvoří stromovitou strukturu, kterou můžeme následně "procházet" a editovat. Veškeré pársování probíhá až v paměti, samozřejmostí je také vytváření XML struktur. Podrobnější informace nalezneme v [7].

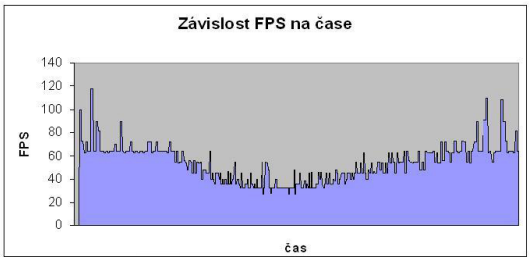
4.4 Výsledky implementace

Naši aplikaci jsme pouštěli a testovali na vstupním terénu, který obsahoval několik typů porostu. Byly použity jak objekty přímo načtené z 3DS souborů, tak i objekty vytvořené pomocí billboardingu, jak je vidět na obrázcích 4.7 a 4.8. Výslednou scénu můžeme hodnotit jak po stránce grafického zpracování (vzhled objektů, celkový vzhled), tak i po stránce výkonnostní. To znamená náročnosti vykreslování celé scény, tudíž i možnostech procházení.

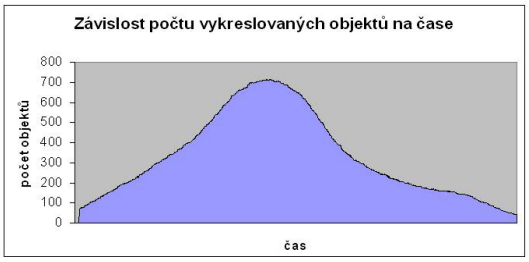
Program jsme testovali na počítači s procesorem AMD Athlon(tm) XP 2500+, 512 MB RAM a grafickou kartou Radeon 9600 PRO. Jedním z důležitých údajů o výkonu aplikace je také hodnota FPS[2]. Tato hodnota nám udává, kolikrát za sekundu je grafickou kartou vykreslen obrázek scény (snímek neboli frame). FPS je jedním z důležitých pojmů hlavně u počítačových her, kde nízká hodnota FPS znamená "trhání" hry. Díky nedokonalosti lidského oka stačí, aby počet snímků byl alespoň 24 za 1 sekundu, a člověk už vnímá výsledný obraz jako plynulý. Tato hodnota není stálá, ale může kolísat vzhledem k vykreslované scéně. Provedli jsme test, kdy jsme nechali avatara projít napříč celou scénou, tzn. že se zde nacházejí jak oblasti bez objektů

[2]angl. Frames per Second

(voda, tráva, atd.), tak i oblasti řídce nebo hustě osázené. Naměřené výsledky jsme vyjádřili následujícími grafy na obrázku 4.6.



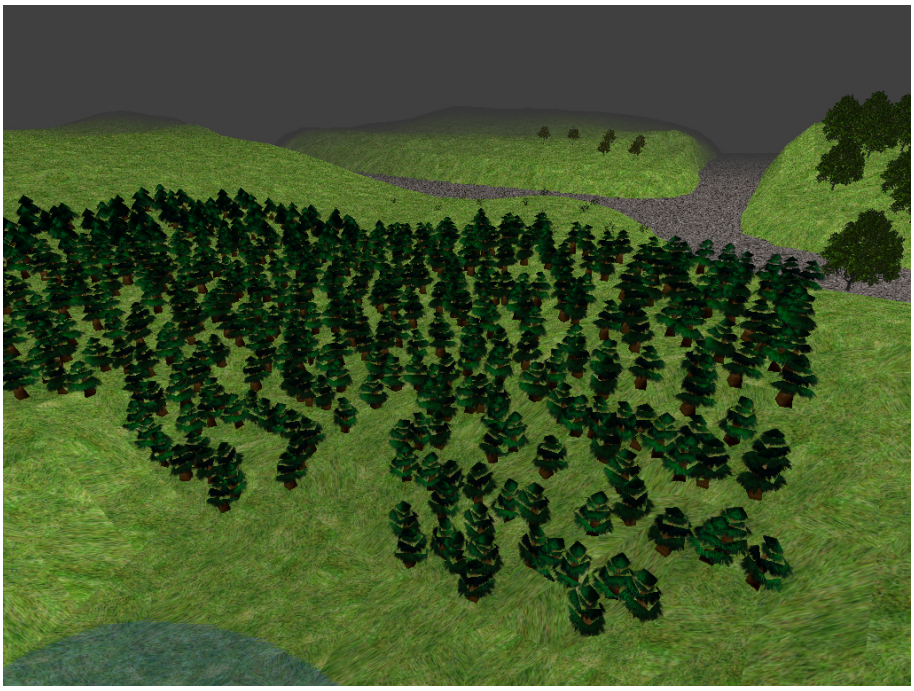
(a) Závislost FSP na čase



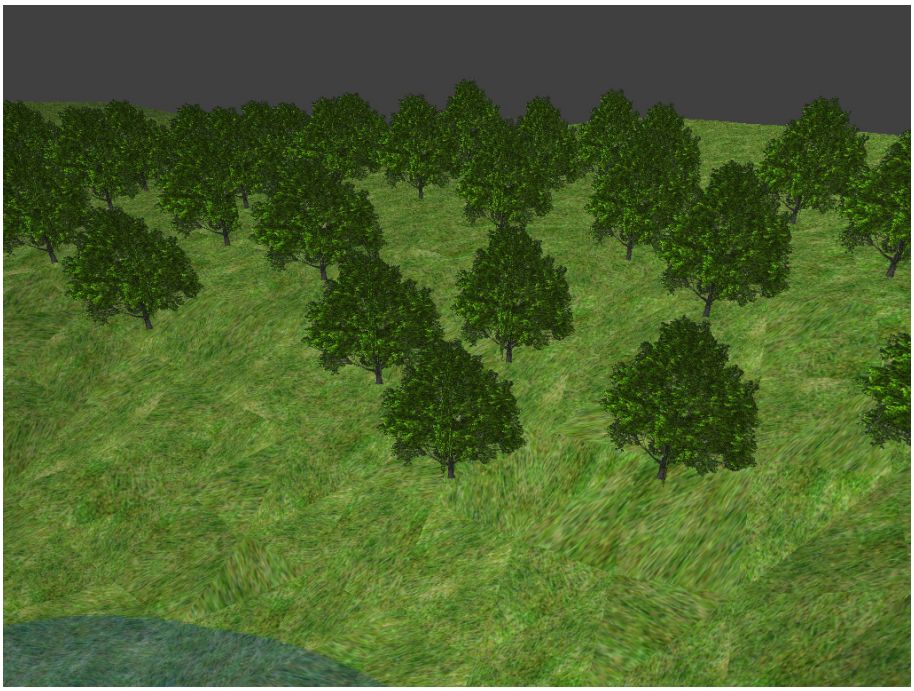
(b) Závislost počtu vykreslovaných objektů na čase

Obrázek 4.6: Grafy závislostí veličin na čase

Jak je vidět z průběhů grafů, hodnota FPS je nepřímo úměrná počtu vykreslovaných objektů. K největšímu poklesu FPS dochází při vykreslování 3DS objektů. Doba potřebná k vykreslení 3DS modelu stromu je mnohem větší než doba k vykreslení stromu pomocí billboardu. Přitom výsledný dojem z jednoduchého modelu není lepší než z billboardu. K reprezentaci vzdálených předmětů je jednoznačně lepší použití billboardů. Při použití kvalitních textur a alfa blendingu je výsledný dojem z objektů vytvořených pomocí billboardů velmi dobrý, a to jak po stránce grafické, tak i po stránce rychlosti vykreslování.



Obrázek 4.7: Ukázka aplikace 1 - rozmístění smrkového lesa, který byl vytvořen pomocí 3DS modelů



Obrázek 4.8: Ukázka aplikace 2 - listnatý les vytvoření pomocí billboardů

Kapitola 5

Závěr

5.1 Shrnutí

Tato práce se zabývala modelováním a procházením terénu. Podle vstupních výškových dat terénu, které mohou být samozřejmě založeny na skutečném pokladu, se vygeneruje 3D terén. Vstupní data obsahují další atributy, které nám dále podrobněji popisují scénu. Jedná o typ přírodního povrchu, jako je tráva, pole, louka, voda, dále je to druh porostu, jeho hustota a další vlastnosti. Popis vstupních dat najdeme v části 2.1 a 4.1. Hlavním úkolem této práce je vymodelovat "přírodní ekosystém", neboli takové prostředí, které má odpovídat skutečnému reálnému prostředí. Vstupní terén může být samozřejmě libovolný, proto nepřipadá v úvahu, že by vstupní data terénu obsahovaly údaje o každé pozici stromu či keře na mapě. To je právě úkolem našeho modelování. Na základě vstupních parametrů musíme být schopni vygenerovat takové pozice objektů na mapě, aby se co nejvíce blížily skutečnosti. Námi navržený algoritmus je popsán v části 3.2. Na jeho vstup dostáváme libovolnou trojúhelníkovou síť s atributy, výstupem našeho algoritmu jsou výsledné pozice objektů v terénu. Poslední částí práce je také zobrazení takto vymodelované scény a její procházení. Protože dovolujeme uživateli libovolné procházení scény v reálném čase, musíme tomuto požadavku také přizpůsobit rychlost vykreslování objektů. Tímto problémem se zabýváme v celé kapitole 3, kde se seznamuje s různými typy objektů, možnostmi jejich optimalizace a technikami Level of detail. Konkrétními možnostmi zobrazování dále pak v části 3.4. Výsledkem je aplikace napsaná v jazyce C++, grafická část je založena na technologii OpenGL. Tato aplikace nám demonstruje všechny popsané techniky a možnosti zobrazování terénů a objektů na příkladě scény s několika typy porostů, kterou můžeme interaktivně procházet.

5.2 Zhodnocení výsledků a diskuze

Při práci jsme zjistili, že modelování a procházení terénu je mnohem komplexnější a rozsáhlejší problém, celou práci lze rozdělit do několika větších oblastí. Jedna část práce by se mohla zabývat pouze modelováním terénu. Jednalo by se o vhodnou

volbu reprezentace dat terénu, spousta možností optimalizace, počínaje jednoduchými pravidelnými trojúhelníkovými sítěmi až po složité techniky Level of Detail. Nemůžeme se věnovat úplně všem směrům modelování, to už je otázkou nějakého herního enginu, na kterém pracuje několik lidí. Naše práce by se měla hlavně zabývat modelováním ekosystému, tzn. modelováním rozmístění stromů, keřů a dalších objektů v terénu. Myslíme si, že jsme navrhli v jistém smyslu univerzální algoritmus pro rozmísťování objektů, který není závislý na reprezentaci vstupních dat. Jeho vstupem je pouze trojúhelníková síť s atributem hustota. Výstupem algoritmu je datová struktura s výslednými pozicemi rozmístěných objektů. Náš algoritmus může být dále upraven či jenom pozměněn podle požadavků uživatele. Stačí přidat další kritéria pro rozmísťování. Je to například minimální vzdálenost mezi sousedními stromy. Nebo podmínka, zda požadujeme les jedné generace (v takovém lese jsou všechny stromy stejně vzrostlé) nebo dovolíme i vícegenerační les.

5.3 Směry dalšího vyvíjení

Jak jsme již uvedli v předcházejících odstavcích, je zde několik směrů, kterými se můžeme dále ubírat. Ať je to již samotný terén či modelování a zobrazování různých objektů. Zůstaňme ale u našeho problému, kterým je modelování ekosystému. Podle našeho názoru, tento problém není nějak přesně definován, nijak striktně vymezen. Jistě lze postupovat i jinými směry. Za úvahu stojí i myšlenka "vzniku například deštného pralesa". Úvaha stojí na základě, že bychom zaseli do terénu semínka porostů zcela náhodně. Přirozeným výběrem by půlka stromů vůbec nevyrostla, stromy by si navzájem překážely. Museli bychom provést simulaci růstu lesa pro získání finální podoby. Dalším směrem vývoje takových modelací by mohlo být porovnání výsledků takto vysetého lesa s naším, tedy takovým lesem, který vznikne přímým použitím algoritmu.

Literatura

[1] David Luebke, Benjamin Watson, Jonathan D. Cohen, Martin Reddy, Amitabh Varshney: *Level of Detail for 3D Graphics*, Elsevier Science Inc 2002, str. 185 - 228.

[2] F.P.Preparata: *Computational geometry*, 1978.

[3] Martin Fuchs: *Triangulace monotónního polygonu*, http://moon.felk.cvut.cz/∼pjv/Jak/_info/i452/triangulace.doc, 1998.

[4] Web3D: Základní informace a specifikace formátu X3D, http://www.web3d.org

[5] Bruce Eckel: *Myslíme v jazyku C++*, Grada Publishing, 2000.

[6] Pavel Herout: *Učebnice jazyka C*, Kopp, 2004.

[7] Frank Vanden Berghen: *Small, simple, cross-platform, free and fast C++ XML Parser*, http://iridia.ulb.ac.be/∼fvandenb/tools/xmlParser.html

[8] Jackie Neider, Mason Woo, Tom Davis: *OpenGL Programming Guide*, Addison Wesley, 1997.

[9] OpenGL.org: Základní specifikace a materiály o programování OpenGL http://www.opengl.org

[10] CZ NeHe OpenGL: Vše o programování 3D grafiky pod knihovnou OpenGL, http://nehe.ceske-hry.cz/

[11] Pavel Tišnovský: *Trojrozměrné modely terénu*, http://www.root.cz/clanky/trojrozmerne-modely-terenu

[12] Pavel Tišnovský: *Grafické karty a grafické akcelerátory. Grafická knihovna OpenGL.* http://www.root.cz/serialy/graficke-karty-a-graficke-akceleratory http://www.root.cz/clanky/graficka-knihovna-opengl-1

[13] Martin Břehovský, Karel Jedlicka: *Úvod do geografických informačních systémů*, http://gis.zcu.cz/kam/studium/ugi/e-skripta/ugi_k2b-datove_modely.pdf

[14] Josef Pelikán: *Datové struktury pro prostorové vyhledávání*, http://cgg.ms.mff.cuni.cz/ pepca/lectures/pdf/datasurvey.pdf

[15] Vladislav Šindelář: *Digitální model terénu*, http://www.grafika.cz/art/3d/clanek1033198934.html

[16] Martin van Velsen, Robin Fercoq, Jim Pitts, Albert Szilvasy: *3D Studio File Format (3ds) - Document Revision 0.93 - January 1997.*

[17] Samir Emdanat: Zdroje modelů stromů, http://www-personal.umich.edu/∼emdanat/Tutorials/FormZ/Vrml/Examples/Trees/index.html

[18] Michal Turek: *Billboarding (přiklápění polygonů ke kameře)*, http://nehe.ceske-hry.cz/cl_gl_billboard.php

Dodatek A

Obsah CD

Součástí bakalářské práce je CD, na kterém se nachází vlastní aplikace a také ostatní důležité dokumenty v elektronické formě. Vše je popsáno v této kapitole.

Vlastní obsah CD:

- adresář **Application** - aplikace nachystaná pro okamžité spuštění z CD, podrobnější popis spustění v části B.2
- adresář **Bachelor thesis** - obsahuje elektronickou podobu této práce
- adresář **Documentation** - zde se nachází programátorská dokumentace k programu, dále licence k používaným knihovnám
- adresář **Install** - obsahuje archiv ModTer.zip pro instalaci aplikace, více v části B.1
- adresář **Project** - obsahuje adresář Source, kde jsou veškeré zdrojové kódy programu, a adresář Data se všemi potřebnými daty

Dodatek B

Uživatelská příručka

B.1 Úvod a instalace

Program **ModTer** slouží k zobrazování a procházení terénu uloženého ve vektorové podobě v upraveném formátu X3D. Na základě vstupních dat dojde k vygenerování trojúhelníkové sítě s příslušnými parametry, jako je hustota porostu, textura atd. Dále podle parametrů vstupních dat proběhne výpočet rozmístění vegetace v terénu (stromy, keře atd.). Takto vygenerovaná je scéna je následně graficky zobrazena s možností interaktivního procházení.

Požadavky na výpočetní prostředky:

- procesor s taktovací frekvencí 2.0 GHz nebo vyšší
- operační paměť alespoň o velikosti 512 MB
- 100 MB volného místa na pevném disku
- operační systém Windows 2000/XP

Samotnou instalaci provedeme rozbalením archivu ModTer.zip z přiloženého CD (CD:\install\ModTer.zip) na pevný disk.

B.2 Spuštění a struktura aplikace

Adresářová struktura aplikace po instalaci:

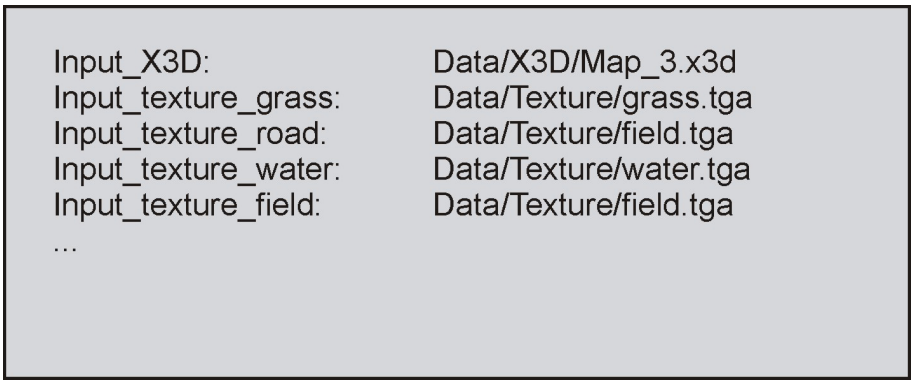
Aplikace ModTer

- EXEC\
  - ModTer.exe
  - Conf.mtr

- Data\
  - X3D\
  - 3DS\
  - Texture\
  - Wallpaper\

Samotné spuštění aplikace se provede souborem **ModTer.exe**, který se nachází v adresáři EXEC. Dále se v tomto adresáři nachází konfigurační soubor **Conf.mtr**, kde lze nastavit cesty k jednotlivým vstupním datům, uloženým v adresáři Data. Tento konfigurační soubor můžeme jednoduše upravovat v libovolném textovém editoru. Ukázka konfiguračního souboru je na obrázku B.1. V tomto souboru stačí přepsat cestu ke konkrétním datům. Například první řádek tohoto souboru nám nastavuje cestu ke vstupní vektorové mapě X3D. Ta se nachází podle konfiguračního souboru v adresáři Data\X3D\Map_3.x3d.

Po spuštění aplikace se objeví konzole, kde jsou vypisovány údaje o průběhu, načítaní a zpracování vstupních dat. Po dokončení inicializace programu dojde ke spuštění grafického prostředí - okna, do kterého je vykreslována výsledná scéna. Případné chyby při inicializaci jsou vypsány do konzole.



Obrázek B.1: Část konfiguračního souboru Conf.mtr

V adresáři Data máme uloženy všechny vstupní soubory. Data jsou dále dělena do jednotlivých adresářů:

- adresář **X3D** - vstupní vektorové data terénu
- adresář **3DS** - 3D modely objektů zobrazované v aplikaci, tj. modely stromů atd.
- adresář **Texture** - textury použité v aplikaci, textury povrchu (tráva, kamení, skála, atd.)
- adresář **Wallpaper** - uloženy obrázky okolí v případě používání tapet okolí

B.3 Ovládání aplikace

- Po zobrazení okna s vykreslenou scénou může uživatel rovnou začít s procházením terénu.
- Ve středu aplikace se po spuštění zobrazí kurzorová šipka, kterou lze ovládat myší.
- Pomocí myši lze měnit směr pohledu na scénu.
- Po nastavení směru pohledu na scénu se lze avatarem v tomto směru pohybovat pomocí kurzorových šipek na klávesnici.
- Přepínání režimu myši z procházení terénu do režimu ovládání okna se provádí mezerníkem = klávesa Space.

Základní ovládací prvky jsou popsány v tabulce B.1.

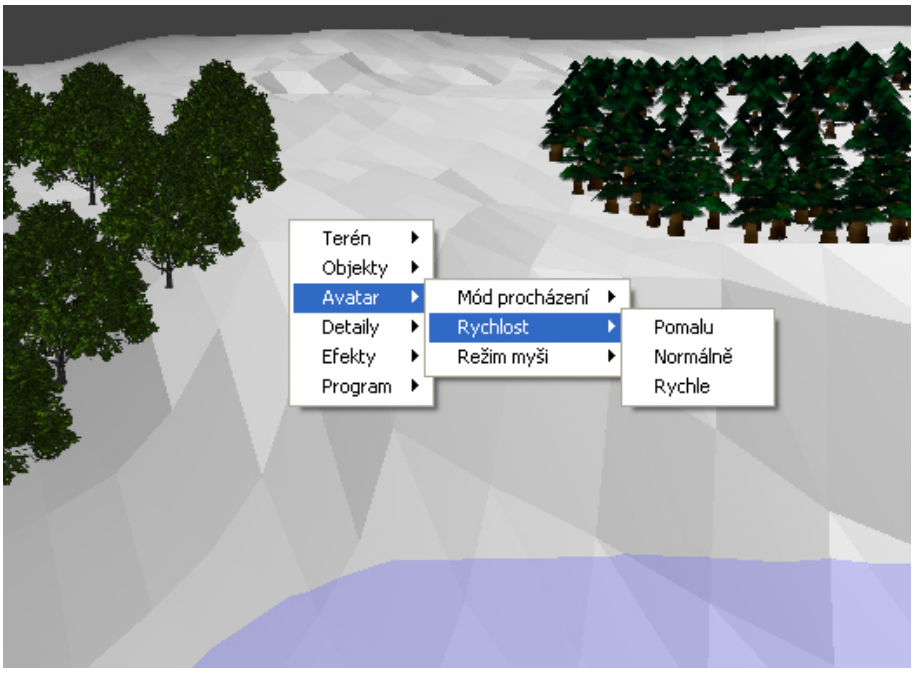
Tabulka B.1: Základní ovládací prvky na klávesnici

klávesa	**význam**
ESC, Q	ukončení celé aplikace
kurzorové šipky	pohyb avatara
F	přepnutí do celoobrazovkového režimu = fullscrean
space = mezerník	přepínaní kurzoru myši z režimu procházení do režimu ovládání okna

Pokročilejší ovládání, další nastavení a různé možnosti nastavení procházené scény se nacházejí v uživatelském menu přímo v aplikaci. Toto menu vyvoláme stiskem pravého tlačítka myši přímo v aplikaci. Příklad menu nalezneme na obrázku B.2.

Základní položky uživatelského menu jsou:

- Terén - vykreslování, textura, atd.
- Objetky - vykreslování
- Avatar - mód procházení, rychlost, režim myši
- Detaily - velikost dohledu, úroveň LODu, atd.
- Efekty - světlo, mlha
- Program - ukončení, okno, fullscrean



Obrázek B.2: Ukázka aplikace 3 - uživatelské menu vyvolané pravým tlačítkem myši, nastavena možnost vypnutí textury u terénu