Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Modelování a vykreslování trávy pro herní aplikace

Plzeň, 2007 Vladimír Geršl

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky
Akademický rok: **2006/2007**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Vladimír GERŠL**

Studijní program: **B3902 Inženýrská informatika**

Studijní obor: **Informatika**

Název tématu: **Modelování a vykreslování trávy pro herní aplikace**

Zásady pro vypracování:

1. Seznamte se s existujícími řešeními daného problému a shrňte jejich výhody a nevýhody
2. Na základě tohoto souhrnu a dalších požadavků ze strany vedoucí navrhněte vlastní řešení.
3. Po schválení vedoucí řešení implementujte a vyzkoušejte na požadovaném typu dat.
4. Dosažené výsledky zhodnoťte.
5. K vytvořenému programu sepište kvalitní uživatelskou i programátorskou dokumentaci.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

Vladimír Geršl

V Plzni dne 17.5.2007

Poděkování

Je mojí milou povinností poděkovat všem, kteří svým dílem přispěli ke vzniku této práce. Rád bych poděkoval zejména vedoucí mé bakalářské práce Doc. Dr. Ing. Ivaně Kolingerové za její obětavý přístup, cenné rady a pochopení pro můj zájem o nevšední herní tématiku. Dále bych rád poděkoval mému externímu konzultantovi Ing. Michalovi Varnuškovi, PhD. a počítačovému odborníkovi Tomášovi Lamrovi za poskytnutí odborné pomoci a potřebných studijních materiálů. Díky patří v neposlední řadě též mé rodině a přítelkyni za to, že mi vytvářeli příjemné zázemí a poskytovali maximální podporu v mém studiu.

Abstrakt

Trendy grafiky v dnešních herních aplikacích vyžadují po travnatém povrchu mnohem více, než jen jednoduchou texturou potažený terén. Bohužel však ani nejmodernější počítače nemají takový výkon, aby mohl být v reálném čase vykreslen realistický trávník tvořený miliony stébel. Práce byla tudíž zaměřena jednak na uspokojení náročných požadavků herního trhu o vytvoření realisticky vypadajícího travního povrchu, zároveň však též na záruku rychlého vykreslování v reálném čase na hardwaru, který je v současné době k dispozici. Je zde tedy popsána jak tvorba textur a podkladových materiálů, tak samotný princip algoritmu řešení. Základem algoritmu jsou osově zarovnané billboardy uložené ve velkých celcích (Vertex Bufferech). Součástí tohoto dokumentu je též vysvětlení doplňujících technik pro zrychlení vykreslování (úroveň detailů, QuadTree). Dále práce předkládá řešení, jak efektně a efektivně aplikovat na trávník vítr. Pomocí těchto vlastností lze vykreslit rozsáhlé a vizuálně atraktivní zatravněné plochy (miliony stébel) v reálném čase.

Klíčová slova: tvorba textur, billboardy, Vertex Buffer, vykreslování trávy, úroveň detailů, alfa průhlednost, vítr, QuadTree

Graphic trends in nowadays game applications involve more complex grassy surfaces than a simple grass texture on a terrain only. However, today's modern computers do not have sufficient enforcement to render a realistic-looking grass plot (with millions of grass blades) in a real-time. Therefore, this report is aimed to both above mentioned relevant questions, i.e., both to satisfy of demanding game market's requirements (realistic-looking grass) and also on rendering on a hardware which is latterly available on the market. Thus, the making-process of textures and other background materials as well as the whole principle of algorithm of the solution are described. Basic elements of this algorithm are fixed aligned layers of quadrilaterals, which are inserted into big units (Vertex Buffers). Moreover, an explanation of supplementary methods for acceleration of render-time (level of detail, QuadTree) is a part of this document, too. Thus, a solution how effectively and impressively apply the wind on a grass plot is given in this report. Using these features and approaches is therefore possible to render visually attractive grassy surfaces (millions of blades) in interactive frame rates.

Key-words: texture creation, billboards, Vertex Buffer, grass rendering, level of detail, alpha transparency, wind, QuadTree

Obsah

1. Úvod

Obr 1.1: *Ukázka konečného výstupu našeho programu s pohledem na celkovou plochu.*

Obr. 1.2: *Ukázka konečného výstupu našeho programu s detailním pohledem na stébla v popředí.*

Doby, kdy počítačového hráče-konzumenta zaujala hra pouze svojí hratelností, jsou již dávno pryč. Nyní - se strhujícím nárůstem výkonu počítačového hardwaru - rostou stejně i nároky na realismus v počítačových hrách. K realistickému pocitu ze hry přispívá (spolu s fyzikálním modelem a ozvučením) hlavně její grafická stránka. Na tvůrce her jsou kladeny stále větší nároky a pokud chtějí uspokojit poptávku a prosadit se na zahlceném trhu se svým dílem, musí přinášet stále nové prvky, přibližující hru reálu.

Jednou z těchto věcí je travnatý povrch. Dřív se tráva řešila jednou texturou, kterou se potáhl terén a pouze větší rostliny (keře, stromy) se modelovaly formou billboardů, natočených stále směrem k hráčovi. Dnešní trendy však vyžadují po travnatém povrchu mnohem víc. Tráva musí být detailní, musí být vizuálně atraktivní. Nesmí to být jen 2D textura – naopak, stébla mají vystupovat z terénu do prostoru. A to není všechno. Měla by mít i další vlastnosti opravdové trávy – například vlnit se ve větru.

Jak vidíme, nároky jsou obrovské, avšak – i když je hardware moderních počítačů velmi výkonný – ještě zdaleka není na takové úrovni, aby dokázal vykreslovat miliony stébel trávy v reálném čase. A přitom právě vysoká rychlost vykreslování v reálném čase je jedna z elementárních vlastností, kterou by měla každá metoda (spojená s vykreslováním) v herním enginu mít.

A to byl také cíl této práce. Vytvořit metodu, která atraktivně zobrazí zatravněnou plochu v reálném čase. Musíme si proto uvědomit, co vlastně chceme. Málokterý hráč se například uprostřed líté přestřelky zastaví a začne zkoumat stébla trávy. Tráva tedy musí vypadat reálně, ale jen do té míry, aby hra – jako celek – působila skutečně. V samotné hře nebude mít určitě dominantní postavení. Naopak – je to „jen“ doplňující prvek. Detail, který po zasazení do celku pomůže navodit reálnou atmosféru. Proto je třeba najít zjednodušení trávy do té míry, aby „vypadala dobře“ s tím, že hlavní důraz je kladen na rychlost vykreslování.

Úvodní část práce tvoří originál zadání, prohlášení, poděkování, český i anglický abstrakt a obsah. Po těchto úvodních částech je pět číslovaných kapitol, další součástí práce je přehled zkratek, literatura a dvě přílohy.

První kapitola je *Úvod* a čtenář je v ní uveden do řešené problematiky. Potom následuje *Teoretická část*, ve které jsou probírány různé možné postupy řešení a ty, které jsou vybrány jako nejlepší pro tuto práci, jsou implementovány. Postup a principy implementace jsou zachyceny v kapitole třetí, nazvané *Realizační část*. Nakonec následují *Experimenty a výsledky*, kde je čeho se vyvarovat a k čemu jsme dospěli. V poslední číslované, páté kapitole *Závěr*, najdeme poté jak celkovou rekapitulaci, tak dále zhodnocení, nakolik bylo docíleno vytyčených cílů, jakož i na další vylepšení naší práce.

K tomuto dokumentu je také přiložen disk DVD, který obsahuje zdrojová data, spustitelný soubor s naším řešením a různé doprovodné materiály (animovaný průlet nad námi vytvořenou trávou, záznam z autorovy přednášky [11] o této práci atd.).

2. Teoretická část

Na začátku této kapitoly je třeba zmínit, že uceleně publikovaných a veřejně přístupných řešení tohoto problému není mnoho. Většina vývojářských studií, když nějaké řešení vymyslí a zahrne ho do svého herního enginu, drží toto svoje know-how pod pokličkou. Některé popsané postupy však přece jenom existují. Z tohoto důvodu jsme se rozhodli rozdělit tuto kapitolu a každou sekci – tj. jak vypozorovaných vlastností, tak vyčtených vlastností a postupů – si představíme zvlášť.

2.1 Vlastnosti vypozorované

Bylo třeba vyzkoušet moderní hry a zjistit, jak travnaté plochy vypadají a jak se chovají. Z toho se poté dá odvodit řada zajímavých vlastností. Jelikož ve specifikacích externího konzultanta této práce (a zároveň zadavatele) bylo, že řešení má být navrhnuto pro hry s velmi velkou hrací plochou (cca 400 km^2), bylo třeba vyzkoušet nejnovější hry převážně z řad tzv. otevřených RPG (např. *The Elder Scrolls IV: Oblivion*, *Gothic 3*), ve kterých jsou také velmi rozsáhlé travnaté terény. Ze všech her, které jsme testovali, jsou (dle mého názoru) jako ideální předlohy pro možné řešení hry *The Elder Scrolls IV: Oblivion* a *Far Cry*. Obě hry jsou zahaleny v moderním grafickém kabátu a řadí se mezi současnou špičku. Soustředíme se na aspekty, které jsou pro náš účel zásadní a které se dají vypozorovat. Těmito aspekty jsou vzhled trsu, vzhled celkové zatravněné plochy, pohyb ve větru, distribuce trsu na terénu a úroveň detailů (Level of Detail - LoD), neboli způsob mizení trsů v dálce. Poznatky z těchto dvou (i některých dalších) her jsou shrnuty v Tab. 2.1. Z Obr. 2.1 a Obr. 2.2 je vidět, že ačkoli každá hra přistupuje k dílčím řešením jinak, přesto obě dávají hezké výsledky.

Obr. 2.1: *Obrázek travnatého terénu ze hry Far Cry.*

Obr. 2.2: *Obrázek travnatého terénu ze hry TES IV: Oblivion*

	FAR CRY	*THE ELDER SCROLLS IV: OBLIVION*	*ARMED ASSAULT*	*COBRA 11*
Vzhled trsu	Trs je tvořen z jedné velké plochy a další plochy, která na ni navazuje pod jiným úhlem a ruší tak „placatý dojem“.	Trs je tvořen z více (cca 10) různě nakloněných ploch.	Podobný Far Cry, jen tvořen z více ploch a s horší texturou.	Jedna plocha, zarovnaná stále čelně ke kameře.
Vzhled celkové plochy	Různě velké trsy, prokládané vždy nějakou kontrastní barevnou rostlinou.	Vždy soudržné celky. Někdy vřesoviště, jindy klasické trsy. Trsy v jednotlivých celcích rozlišeny jen velikostí.	Reálně vyhlížející tráva, čas od času proložená barevnou květinou.	Přibližně 4 druhy textur, které se střídají. Celkově zde vypadá tráva velice nepřitažlivě.
Pohyb	Vše se houpe ze strany na stranu stejně.	Trsy se různě pohupují - v různých vzdálenostech se houpají jinak.	Při větru se různě pohupuje. Hýbe se i při poryvu z helikoptéry.	Nepohybuje se.
Rozmístění po krajině	Libovolné.	Zřejmě podle textury a náklonu terénu.	Zřejmě dle podkladové textury.	Libovolné.
Způsob mizení v dálce	Ty co jsou daleko se zprůhledňují a poté (v určité vzdálenosti) se všechny najednou přestanou vykreslovat.	Vzdálené se zprůhledňují až do ztracena.	Postupně se zvětšují a přidávají se další pláty i detaily textury.	Trsy se postupně v dálce objevují (zřejmě se zvětšují celé plochy).

Tab. 2.1: *Vypozorované vlastnosti travnatých povrchů her Far Cry a TES IV: Oblivion.*

Následující zbytek kapitoly 2.1 *Vlastnosti vypozorované* je o velmi subjektivních pocitech, proto je zde nutné upozornit, že všechny vlastnosti budou hodnoceny čistě z pohledu autora práce.

Jako kandidáty na výběr vhodné trávy jsme si již výše zvolili *Far Cry* a *Oblivion*. Rozhodnout, který z těchto dvou přístupů je lepší, však nelze. Co se týká vzhledu jednotlivého trsu, je ideál někde mezi. Dva čtverce na trs jsou málo a naopak 10 čtverců (z hlediska poměru o jaký se zvedne estetický dojem z trsu a o jaký se zvýší náročnost jeho vykreslení) zbytečně mnoho. Ideální by tedy měl být kompromis, tzn. 4-6 čtverců na trs.

Jako vzhled celkové zatravněné plochy lépe působí trávník hry *Far Cry*. Ta má výhodu, že se odehrává v tropech, a proto si autoři mohli dovolit přidat velmi barevné a vizuálně atraktivní rostliny. I když si však odmyslíme tropickou flóru, zdá se být koncept barevně stejnorodého trávníku s občasným ozvláštněním nějakou rostlinou lepší.

Naopak pohyb trávy vypadá zajímavější u hry *Oblivion*, kde se tráva nehoupe na všech místech stejně. To dodává na realističnosti (i v reálu, když se díváme na louku, tak vítr také dělá „vlnění“ trávy – neohne všechna stébla stejně a na všech místech stejně ani nefouká).

Musím tedy konstatovat, že nemám vyhraněný postoj k tomu, který přístup k rozmístění trsů po krajině, je lepší. Nějaká distribuce ale třeba je, to je zjevné. Minimálně je nezbytné provést rozmístění trsů na terénu. Jinak řečeno, umístit je do správné nadmořské výšky.

Způsob mizení trsů v dálce se mi nelíbí ani u jedné hry. Zprůhledňování trsu v dálce mi přijde nepřirozené a působí na mě rušivě. Na tomto místě bych tedy spíše uvedl zmínil hru *Armed Assault* (viz Tab. 2.1), která má tuto vlastnost zvládnutou (z testovaných her) zřejmě nejlépe.

2.2 Vlastnosti vyčtené

Metody vykreslování travnatých povrchů (nebo povrchů s podobnými vlastnostmi – např. chlupy) se dají rozdělit do hlavních dvou skupin. Jsou to metody založené na geometrickém vykreslování, tzv. *Geometry-Based Rendering* (GBR) [1] a metody založené na vykreslování obrázků, tzv. *Image-Based Rendering* (IBR) [2].

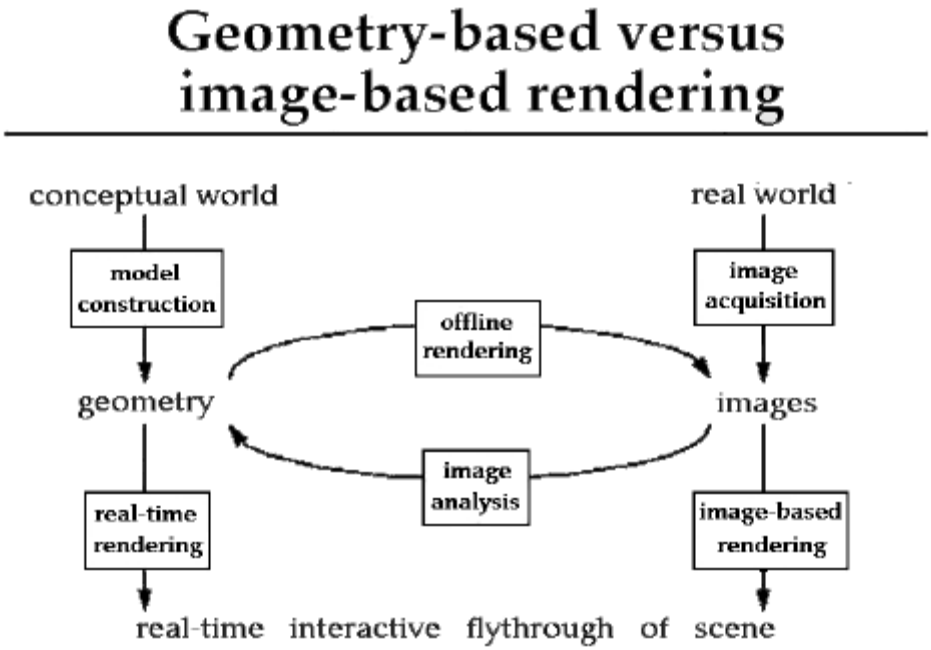
Vztah mezi metodami GBR a IBR je zachycen na Obr. 2.3.

2.2.1 Geometry-Based Rendering

Tělesa tvořená geometrií jsou zřejmě tou nejintuitivnější formou tvorby těles. Každý objekt (v našem případě například stéblo trávy) je tvořen vlastní geometrií, potažen vlastní texturou, apod. (viz Obr. 3.5). Metody geometrického vykreslování vykreslují tento druh těles (viz Obr 2.3). Proto dávají samozřejmě vizuálně nejlepší výsledky. Objekty kompletně tvořené geometrií, s materiály a texturami poskytují všechny informace pro osvětlovací model, ze všech pozorovací úhlů vypadají (samozřejmě pokud je model správně vytvořen) přirozeně a dají se na ně daleko lépe aplikovat různé realistické fyzikální modely.

Jelikož je ale každé těleso tvořeno z mnoha trojúhelníků, jsou také zároveň tyto metody velmi náročné na výkon hardwaru [2]. Proto je jejich použití

v herních aplikacích ještě hudbou relativně vzdálené budoucnosti. Nyní je jejich hlavní použití v aplikacích, kde není třeba vykreslovat v reálném čase (tzv. offline rendering, jeho využití je např. tvorba efektů pro filmová studia nebo vykreslování pro statické obrázky).



Obr. 2.3: *Vztah mezi GBR a IBR. Je zde naznačená cesta, kdy se model (zkonstruovaný nejprve detailní geometrií) vykreslí offline. Potom se tento obrázek nanese opět na geometrii (tentokrát již značně zjednodušenou) a vykresluje se v reálném čase. (Obrázek převzatý z [2])*

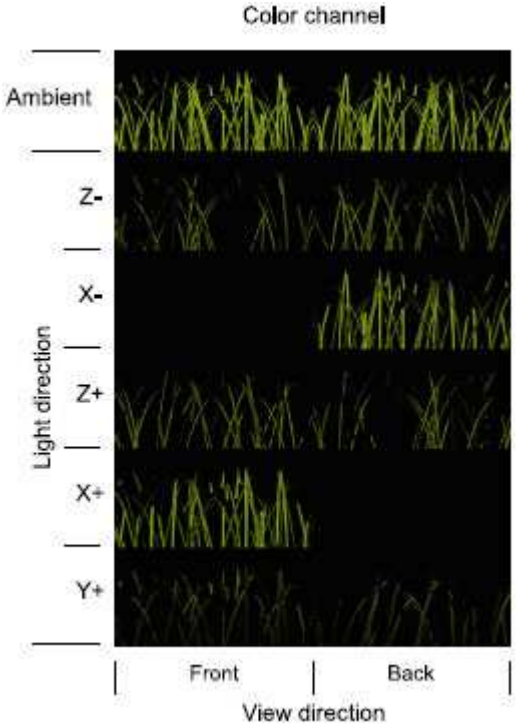
2.2.2 Image-Based Rendering

Úvodem citace z přednášky Marc Levoy [2]: „Studium modelování a vykreslování založeného na obrázcích je studiem o tom, jak zjednodušit reprezentaci geometrie.“. Problém je tedy v tom, že většinou není možné GBR vykreslovat v reálném čase. Snažíme se je tedy zjednodušit, čímž se právě zabývají IBR. V těchto metodách jde především o rychlost zobrazování na úkor vizuální atraktivity. Základní formou IBR jsou tzv. billboardy. Jedná se o nahrazení geometrie (většinou) částečně průhlednou texturou, nanesenou na jednoduchý objekt (zpravidla trojúhelníkový či čtvercový plát). Jednou z hlavních nevýhod této metody je nemožnost aplikace kompletního osvětlovacího modelu, jelikož se převedením 3D objektu do textury ztratí například informace o jeho hloubce a reflexních vlastnostech.

Existuje více přístupů k billboardům. Snad úplně první metodou, která se začala používat ve hrách před více než deseti lety (např. *open RPG Might and Magic VI.*, 1997) a používá se dodnes pro nejvzdálenější objekty, jsou billboardy stále čelně otočené ke kameře (myšleno k pozorovateli). Tato metoda však nevypadá přirozeně a pro blízké objekty se již v současnosti nepoužívá. Další

možností jsou např. billboardy zarovnané s osami. To již dává znatelně lepší výsledky a proto se také objevuje tento přístup v mnoha současných hrách.

Místo klasických, částečně-průhledných textur, můžeme použít například tzv. *Bidirectional Texture Function* (BTF). BTF spočítá v tom, že pro každý plát máme více textur - pro různé úhly pohledu a pro směry světelného zdroje (viz Obr. 2.4). Kombinace osově zarovnaných billboardů s BTF můžeme najít v různých modifikacích například v [1, 3, 4].



Obr. 2.4: *Ukázka textur potřebných pro potažení jednoho plátu pomocí techniky BTF. Zde jsou vidět textury pro jeden osově zarovnaný plát. Jelikož má tento plát nulovou šířku, můžeme ho pozorovat jedině zepředu a zezadu. V jednotlivých řádcích je potom plát osvětlen z různých směrů. Směr osvětlení zespodu chybí, jelikož předpokládáme, že se světelný zdroj nemůže nacházet pod zemí. (Obrázek převzatý z [1])*

2.2.3 Další metody

Další metodou je například kombinace předchozích dvou metod [1]. V tomto přístupu je užíván třístupňový *Level of Detail* (LoD), kde nejbližší trsy trávy jsou tvořeny reálnou geometrií, středně vzdálené trsy jsou tvořeny několika osově zarovnanými billboardy a nejvzdálenější jsou tvořeny pouze horizontálním billboardem. Tento způsob dává velmi hezké výsledky (viz Obr. 2.5) s reálnými stíny a možností využití kompletního osvětlovacího modelu. Bohužel je zřejmě ale stále výpočetně příliš náročný (i když už není problém ho vykreslovat v reálném čase), což pravděpodobně zamezuje jeho využití v hernách aplikacích.

Obr. 2.5: *Ukázka travního povrchu vytvořeného programem využívající kombinace IBR a GBR metody. (Obrázek převzatý z [5])*

Objevují se také metody vycházející z nového *Shader Modelu 3.0* [6]. Ty používají takzvaný *Geometry Instancing*. Ten funguje v principu tak, že se daný trs pošle pouze jednou na grafickou kartu a tam se z něj vytvoří instance, kterými poté pokryji povrch. Je to velmi rychlé, jelikož se oprostíme od jinak velmi časově náročné komunikace mezi CPU a GPU. Nevýhodou však je, že k této metodě potřebujeme nejnovější grafické karty, které ještě nejsou mezi veřejností (ani tou hráčskou) příliš rozšířené.

2.3 Sumarizace poznatků

Žádná z výše uvedených metod a postupů pro náš účel není použitelná jako celek, ale lze využít mnoho dílčích řešení.

Z dříve uvedených faktů víme, že GBR je náročné na hardwarový výkon počítače a používá se téměř výlučně pro offline vykreslování. Naše aplikace však klade hlavní důraz na rychlost vykreslování. Z toho vyplývá, že není možné používat reálnou geometrii a je zapotřebí trs tvořit pomocí billboardů. Jako přiměřený počet billboardů, které budou tvořit jeden trs, jsme zvolili 4-6 čtvercových osově zarovnaných billboardů.

Kombinovaná metoda, která využívala pro nejbližší trsy reálnou geometrii (viz 2.2.3 *Další metody*), je velmi zajímavá (a to jak principem, tak především grafickými výsledky – viz Obr. 2.5). V návrhu programu necháme tedy tuto cestu otevřenou pro budoucí pokusy. To v důsledku znamená, že bude potřeba vymodelovat trs ve 3D grafickém editoru (viz 3.1.2 *Použitelný trs*).

Pro texturu billboardu je ideální textura s alfa kanálem (viz 3.2.5. *Alfa kanál*). Ten určuje, která místa budou průhledná a která nikoli (tzn. klasická částečně-průhledná textura).

BTF je zajímavá technika, pokud chceme mít trávník osvětlený. Tento požadavek však nebyl na naší metodu kladen. Proto tato vlastnost nebude implementována. Mimoto je BTF poměrně pracná a také paměťově i výpočetně náročnější technika, takže by se snížila rychlost vykreslování (každý plát by měl minimálně 12 různých textur místo jedné, viz Obr. 2.4).

Jako koncept celkového trávníku, je podle mého názoru vizuálně nejatraktivnější použít rozsáhlý trávník s homogenními trsy, ozvláštněnými čas od času nějakou barevnou rostlinou (viz Obr. 2.1, Obr. 2.5).

V aplikaci bude využit LoD podobný jako v [1], avšak nejbližší trsy nebudou vytvářen reálnou geometrií, jelikož je příliš výpočetně náročná.

Vzdálené trsy se nebudou zprůhledňovat (tvoří to nepřirozený efekt), místo toho se budou s rostoucí vzdáleností od pozorovatele zmenšovat.

Pohyb trávy ve větru bude pro různá místa trávníku různý a k jeho vlnění využijeme goniometrické funkce sinus a cosinus, které jsou hladké, plynulé a pro potřebu naší práce ideální.

Geometry instancing využít nelze, jelikož je žádoucí, aby vytvořený algoritmus mohli používat i uživatelé se starším grafickým hardwarem.

3. Realizační část

V této části si pokusíme nastínit postup vývoje travnatého povrchu „krok za krokem“, včetně „slepých uliček“ (aby případní další tvůrci podobných aplikací věděli, čeho se vyvarovat).

Realizace by se dala rozdělit na dvě hlavní části. První z nich je tvorba podkladů, fotografování a skenování textur, jejich úprava ve 2D grafickém editoru a tvorba modelů ve 3D grafickém editoru. Druhá je potom samotná programová realizace a věci s ní spojené.

3.1 Tvorba podkladů

Nejprve bylo třeba nafotografovat textury několika trsů trávy. Autor používal digitální fotoaparát *Olympus FE150*, který má slušné parametry pro fotografování na blízkou vzdálenost, což je pro tvorbu textur tohoto charakteru velmi důležité (nejkratší zaostřitelná vzdálenost v režimu *Super Macro* je 6 cm).

3.1.1 Nepoužitelný trs - slepá ulička

V improvizovaném ateliéru jsme vyskládali dvě rovné plochy bílým papírem, aby vzniklo jednolité pozadí, které je žádoucí pro budoucí úpravu textury. Jelikož nebyly k dispozici profesionální fotografické reflektory a nasvětlení běžnými světelnými zdroji je komplikované, umístili jsme tento ateliér na volné venkovní prostranství a vyčkali, až nebude ostrý sluneční svit, abychom se zbavili nežádoucích stínů. V takto připraveném prostředí jsme vyfotografovali řadu rozdílných trsů trávy a další rostliny, to vše z různých vzdáleností a stran. Ukázka jednoho takto nafoceného trsu je na Obr. 3.1.

Dalším krokem bylo vytvoření low-poly trsu v 3d editoru *3D Studiu Max8* (3Ds Max). Tento výkonný grafický editor jsme si vybrali proto, že je to nejvíce používaný editor tvůrci počítačových her a je pro tuto činnost přímo stavěn. Je například možné z něj bez problémů exportovat jakýkoli otexturovaný model do formátu *.X, který se dá velmi snadno načíst v *Managed DirectX* (které jsme používali k programové realizaci viz 3.2 Programová realizace). Mimoto má autor práce s tímto grafickým editorem dlouholeté zkušenosti, a tudíž se mu v něm dobře pracuje. Trs jsme zatím zvolili velmi jednoduchý – pouze dvě na sebe kolmé obdélníkové plochy. Na takto připravený low-poly trs jsme namapovali dvě fotografie, vytvořené z jednoho trsu (fotografováno zepředu a zboku).

Takto zhotovený trs si můžeme prohlédnout na Obr. 3.2. Zároveň se jedná o první slepou uličku. Výsledek nebyl podle očekávání. Nepůsobil ani vzdáleně přirozeně a nedokázali jsme si ho představit jako trs trávy v nějaké hře na komerční úrovni. Další důvod, proč jsme od tohoto způsobu tvorby trsu upustili byl fakt, který uvádíme v kapitole 2.3 *Sumarizace poznatků*. Rozhodli jsme se nechat si otevřené dveře pro možné vyzkoušení metody, kdy jsou nejbližší trsy tvořeny reálnou geometrií. S trsem, který jsme však nyní vytvořili, by byly v programu přechody mezi GBR a IBR velmi patrné. V rámci této práce jsme se sice rozhodli tuto metodu nepoužívat, ale přesto nás velmi zaujala a chtěli jsme si nechat možnost připojit ji později do tohoto programu (viz 2.2.3. *Další metody*). Pro tento přístup je ale tedy třeba zhotovit trs trávy od začátku jinak.

Obr. 3.1: *Trs trávy vyfotografovaný v improvizovaném ateliéru.*

Obr. 3.2: *Low-poly model vytvořený v 3Ds Max a otexturovaný fotografií trsu.*

3.1.2 Použitelný trs

Poučeni z předchozího nezdaru jsme se nechali inspirovat prací [1]. Textura tedy bude vytvořena tak, že naskenujeme stéblo trávy a to poté upravíme ve 2D grafickém editoru *Adobe Photoshop 7 CE*.

Jako skener jsme používali multifunkční zařízení *Lexmark X2470*, které pro tyto účely bohatě dostačuje. Editor Photoshop si autor práce zvolil z toho důvodu, že ho precizně ovládá, a proto je v něm schopen docílit daleko lepších výsledků než v jiných editorech.

Úprava naskenovaného stébla spočívala ve vyretušování fleků na stéble a výměně bílého pozadí za pozadí trávově zelené. Potom pro toto stéblo vytvoříme alfa kanál (viz 3.2.5. *Alfa kanál*), který definuje jeho průhlednost. Na Obr. 3.3 a 3.4 je vidět stéblo s jeho alfa kanálem. Alfa kanál definuje průhlednost obrázku takto: tam, kde je bílá barva, je obrázek neprůhledný, a čím je barva tmavší, tím více se zprůhledňuje. Černá je tedy úplně průhledná. Proto je dobré nedělat ostré černo-bílé přechody, ale udělat na okrajích stébla pár barevných mezistupňů přechodu. Tím docílíme vyhlazených hran stébla. Musíme si dát ovšem pozor na prosvítání barvy pozadí – to je právě důvod, proč je dobré jako barvu pozadí stébla zvolit trávovou zeleň (viz Obr. 3.3).

Tímto postupem jsme vytvořili několik stébel trávy s jejich alfa kanály.

Obr. 3.3: *Textura stébla trávy s trávovou zelení, jako barvou pozadí*

Obr. 3.4: *Alfa kanál k textuře na Obr. 3.3. Bílá barva určuje neprůhlednou a černá barva naopak zcela průhlednou část textury*

Dalším krokem bylo vytvořit vysoký úzký obdélníkový plát v 3Ds Max, na který se později nanese textura stébla. Plátu nastavit rozdělení na 4 segmenty tak, aby se v těchto segmentech dal ohnout. Mnohokrát ho rozkopírovat po čtvercové

ploše a potom každý vzniklý plát ručně transformovat tak, aby celek co nejvíce připomínal nesourodá stébla trávy. Plát je třeba transformovat pomocí modifikátoru *Editable Mesh*, který dovoluje pohybovat s každým vrcholem jednotlivého plátu zvlášť. Každé stéblo potom různě ohýbáme, natáčíme podle různých os, měníme jeho velikost apod. Takto zaplníme celou čtvercovou plochu, kterou jsme si vyhradili, jako malou část země pro zatravnění. Na každý plát poté naneseme jednu z námi dříve vytvořených textur stébla i s alfa kanálem, aby bylo okolí stébla průhledné. Výsledek tohoto snažení naleznete na Obr. 3.5 a Obr. 3.6.

Obr. 3.5: *High-poly model trsu, vytvořený v 3Ds Max.*

Obr. 3.6: *High-poly model v editoru 3Ds Max. Na tomto obrázku je vidět, že je každé stéblo rozděleno na 4 části.*

Takto vytvořený trs by bylo již možné využít, pokud bychom chtěli v našem programu používat reálnou geometrii. Každé stéblo je tvořeno samostatně. Proto by se na tento model výborně aplikovalo osvětlení, tvorba dynamických stínů či vítr. Také přibližování a oddalování modelu od pozorovatele by v programu nemělo výraznější vliv na vzhled trsu (narozdíl od toho, když je trs tvořen jenom pomocí billboardů).

My ovšem chceme nyní tento model zjednodušit. Proto provedeme 4 vertikální a 1 horizontální řez tohoto trsu a tyto vzniklé řezy naneseme jako textury na low-poly model v 3Ds Max, skládající se z pěti ploch (viz Obr. 3.7). Přitom máme uložené v jednom obrázku velkém 256×1024 pixelů vedle sebe všechny čtyři textury pro vertikální pláty a v samostatném obrázku texturu pro jeden horizontální plát (viz Obr. 3.8 a 3.9).

0br. 3.7: *Low-poly model trsu, který je tvořen 5ti texturovanými pláty (jednotlivé pláty jsou označeny červenou linkou).*

Obr. 3.8: *Textura horizontálního plátu.*

Tento model (Obr. 3.7) poté můžeme zkusit exportovat do formátu *.X a načíst jako trs ve svém programu. I tento poslední krok se nakonec ukáže jako slepá ulička a to z důvodů rychlosti vykreslování. Byli jsme tedy nakonec nuceni vytvořit si plochy ručně v programu a uvedené textury (Obr. 3.8 a 3.9) na ně poté nanést (více o tomto problému viz 3.2.1).

Obr. 3.9: 4 t*extury vertikálních plátů, dané do jednoho obrázku. Jedna textura je velká 256 x 256px. Odleva jsou poskládány takto: čelní pohled přední plát, čelní pohled zadní plát, boční pohled přední plát, boční pohled zadní plát.*

3.1.3 Další modely

Dalším modelem je květina, která oživuje a probarvuje travní plochu. U květiny však musíme použít postup, který jsme popisovali v kapitole 3.1.1. Nepovedený trs. Je to z toho důvodu, že květina se špatně skenuje a je tedy nutné jí nafotit. Také, narozdíl od trávy samotné, nehraje v naší aplikaci hlavní roli a

tudíž ji není třeba dělat tak detailní. Vyfotografoval jsem tedy rostlinu ze dvou stran, fotky upravil a vytvořil texturu (Obr. 3.10). Tato textura je potom v programu nanášena na dva navzájem kolmé čtvercové billboardy, jejichž rozměry jsou násobně větší než rozměry trsů trávy.

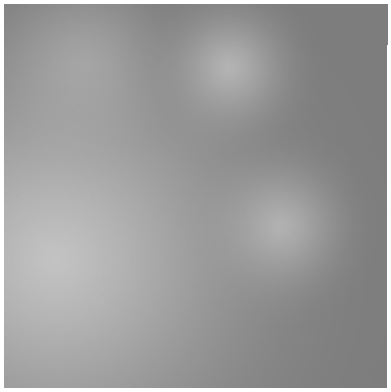
Obr. 3.10: *2 textury pro květinu, vytvořené z fotografií, focených ze dvou různých stran, a poté upravené ve Photoshopu.*

Dalším modelem, který není sice cílem našeho vývoje, ale nesmí chybět, je terén. Po diskuzi s vedoucí a s konzultantem této bakalářské práce jsme se usnesli, že terén bude tvořen pravidelnou sítí bodů. To znamená, že při pohledu shora na body terénu, vidíme pravidelnou mřížku. Jinak řečeno, rozestupy mezi body se v rovině xz nemění. Různá může být pouze souřadnice y každého bodu. Pro jeho vytvoření je nejprve třeba výšková mapa (height map), ve formě obrázku ve stupních šedi kde barva pixelu určuje nadmořskou výšku (v naší terminologii souřadnici y) jednotlivých bodů terénu. K vytvoření této výškové mapy, neboli height mapy (viz Obr. 3.11) byl opět použit výše již několikrát zmiňovaný Photoshop.

Poté, co máme vytvořenou výškovou mapu, přicházejí v podstatě dvě (z našeho pohledu) přijatelné možnosti, jak terén vygenerovat. První z nich je vygenerovat terén přímo v programu. Druhá je vygenerovat terén v 3Ds Max a do programu ho importovat již jako hotový otexturovaný model (Obr. 3.12). To nám na počátku vývoje programu přišlo jako snadnější řešení, takže jsme zvolili tuto variantu. V pozdějších fázích vývoje se ukázalo, že fakt, že dostaneme již jenom model bez výškové mapy, potom značně zkomplikuje určování nadmořské výšky libovolného bodu (například při rozmísťování trsů na terén – více viz 3.2.3 *Rozmisťování trsů a květin na terén*).

Terén se ve 3Ds Max vytváří jednoduše. Na plát s dostatečně velkým počtem dělení (aby byl terén hladký) aplikujeme modifikátor *Diplace*, které dáme jako parametr *Bitmap* naší výškovou mapu. Poté již jenom pomocí zvětšování parametru *Strength* (*Displacement -> Strength*) zvyšujeme perforaci terénu.

Textura terénu se od ostatních textur, které jsou použity v tomto programu, liší tím, že se opakuje na terénu, a tudíž na sebe musí navazovat. Postup, jak dosáhnout efektu navazujících textur, je popsán například v [7]. Jde toho ve Photoshopu dosáhnout poměrně jednoduše tak, že nenavazující texturu, kterou máme k dispozici, necháme pomocí efektu *Posun* (*Filtr -> Jiné -> Posun*) přetočit dokola o polovinu šířky naší textury vodorovně a o polovinu výšky svisle. Uprostřed textury nám vzniknou nenavazující spoje. Ty poté pomocí nástroje *Klonovací razítko* vyhladíme. Nakonec, pokud jsou na textuře nějaká příliš světlá či tmavá místa, upravíme jas, protože jinak je hodně znatelný opakující se vzor.

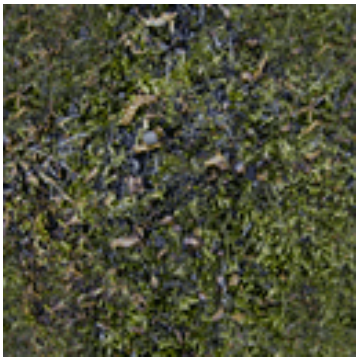


Obr. 3.11: *Výšková mapa – tzn. obrázek ve stupních šedi, který se používá na vygenerování terénu. Čím je barva světlejší, tím má příslušný bod vyšší nadmořskou výšku (viz Obr. 3.12).*

Obr. 3.12: *Terén z 3Ds Max, vygenerovaný pomocí výškové mapy z Obr. 3.11. Textura použita z Obr. 3.14.*

Nejprve jsme se pokusili vytvořit texturu terénu pouze ve Photoshopu, pomocí jeho nástrojů. Textura však nevypadala příliš přirozeně (Obr. 3.13). Proto jsme se rozhodli (jako druhou variantu) vyfotografovat terén a poté tuto fotografii upravit podle [7]. Tato textura je již přirozenější – viz Obr. 3.14. Jako podloží pod trávu můžeme (a také to později děláme) využívat dle potřeb obě varianty.

Obr. 3.13: *Textura terénu, vytvořená kompletně ve Photoshopu.*



Obr. 3.14: *Textura vzniklá vyfotografováním reálného terénu.*

Další doprovodné modely v programu (např. *SkyBox* – neboli krabice s texturou mraků, v které je vše „obaleno“), jsou dělány standardním způsobem v 3Ds Max a poté jsou vloženy ve formátu *.X do našeho programu.

3.2 Programová realizace

Celý program je psaný v programovací jazyce C# a platformě pro vývoj hardwarově akcelerovaných multimediálních aplikací DirectX. DirectX autor zvolil proto, že měl již možnost vyzkoušet i jeho hlavního konkurenta (nebo spíše konkurenta Direct3D) – OpenGL, a na základě předchozích zkušeností mu z těchto dvou vyhovovalo DirectX více. Je více uživatelsky přívětivé, obzvláště pro projekty z oblasti počítačových her a jiných multimediálních projektů.

Rychlostní rozdíly mezi aplikacemi psanými v C# a aplikacemi psanými v jeho předchůdci C++ jsou - s rostoucím výkonem hardwaru moderních počítačů - stále více zanedbatelné (C++ je sice stále o něco rychlejší, ale v řádu procent). Jazyk C# však nabízí oproti svému předchůdci větší rychlost a efektivitu vývoje, je robustnější, odpovídá moderním trendům a v neposlední řadě je nyní již více podporován ze strany Microsoftu.

To jsou důvody, proč stále více herních vývojářských studií přechází z C++ na vývoj v jazyce C#. A proto i pro naše potřeby zvolíme jako vývojový prostředek *Managed DirectX* (tzn. DirectX spolu s jazykem C#).

3.2.1 Komplikace s vykreslováním hodně malých dat

První myšlenka byla taková, že do programu importujeme trs trávy z formátu *.X, který vždy umístíme a terénu a vložíme do pole trsů. Při vykreslování potom pro všechny prvky pole zavoláme jejich vykreslovací metodu (viz Výpis 3.1).

```
foreach (Trs t in trsy)
{
    if (t != null)
        t.Draw(device);
}
```

Výpis 3.1: *Myšlenka uložit všechny trsy do pole a poté u všech volat metodu Draw(). Tento způsob je však neúnosně pomalý.*

Tento způsob však dával extrémně špatné výsledky. Výkon prudce klesal již při vykreslování pár desítek trsů. Důvody jsou, myslím, vcelku zřejmé. Trs je tvořen sice jen pěti pláty, takže samotné vykreslení proběhne rychle. Pro GPU je ale „smrtící“ neustálé přepínání různých doplňkových věcí (např. textury) mezi samotným vykreslováním.

Musíme se tedy snažit přijít na způsob, jak sjednotit trsy do nějakého většího celku tak, aby stačilo zavolat jen jednou metodu pro vykreslení na GPU a vše se (bez přepínání) vykreslilo najednou. Model importovaný z formátu *.X je načtený jako objekt typu *Mesh* (třída *Mesh* je odvozená od třídy *Object* a využívá se pro ukládání těles v programu; jedná se o trojúhelníkovou síť). Ani po

důkladném prozkoumání dokumentace k *Managed DirectX* i dalších zdrojů však nebyl objeven způsob, který by s *Meshem* něco podobného umožňoval provést.

Mesh je vnitřně v podstatě *Vertex Buffer* (VB – datová struktura pro uchovávání vrcholů, nejvíce podobná poli). Existovala zde tedy možnost otevřít VB trsu a všechny vrcholy zkopírovat do nějakého hromadného VB, ve kterém by byly uloženy všechny trsy. Jelikož informace o tom, jak mají být jednotlivé vrcholy ve VB pospojovány, aby tvořily kýžený objekt, je ukryta v tzv. *Index Bufferu* (IB), bylo by nutné vytvořit také celkový IB a zkopírovat do něj *Index Buffery* jednotlivých trsů.

Výše zmiňované řešení se však zdálo být příliš krkolomné. Rozhodli jsme se tedy, že trs je tak jednoduchý objekt, že si ho vytvoříme v programu sami, sami si ho uložíme do celkového VB a budeme nad ním mít poté úplnou kontrolu.

3.2.2 Uložení trsů do větších celků

V předešlé kapitole byl nastíněn směr, kterým jsme se rozhodli ubírat. Tato cesta spočívá ve vytvoření velkých polí s vrcholy (*Vertex Bufferů*), pro které by se vždy zavolalo vykreslení jenom jednou a toto pole by se celé odeslalo na grafickou kartu a tam vykreslilo.

Pro tato pole by ještě musela existovat pole indexů (*Index Buffery*), která by určovala, jak mají být pospojovány dané vrcholy.

Udělali jsme si tedy 2 pole *Vertex Bufferů*. Jedno pro blízké trsy a druhé pro jednodušší horizontální trsy, které se vykreslují po celé ploše (více viz kapitola 3.2.4 Úroveň detailů).

Z pole *umisteni* jsem vždy načetl pozici středu trsu a kolem této pozice jsem udělal v příslušné vzdálenosti příslušný počet vrcholů (viz 3.2.3. Rozmístění trsů). Pro každý trojúhelník jsou třeba tři vrcholy. Oba trojúhelníky tvořící čtverec mají však vždy dva vrcholy společné – tzn. v našem případě vzdálený trs je tvořen pouze jedním čtvercem = 4 vrcholy, blízký trs je tvořen čtyřmi čtverci = 16 vrcholů. Tohle jsem pak vždy vložil do celkového *Vertex Bufferu* (ukázka jednoduchého VB pro květiny viz Výpis 3.2) určeného pro daný druh dat.

Poslední důležitou informací, kterou musí každý mnou používaný vrchol nést, je souřadnice textury. Je to nezbytná informace, pokud chceme nanášet na námi vytvořené trsy později texturu. Jedná se v podstatě o to, jak velká část textury bude zobrazena. Pohybujeme se na intervalu 0 až 1 a máme dvě souřadnice (většinou označované u a v). Souřadnice u nám určuje horizontální rozsah (0 je na textuře vlevo a 1 je vpravo) a souřadnice v určuje vertikální rozsah (0 je nahoře a 1 dole). Když tedy například chceme nanést na plát celou texturu, nastavíme u horních vrcholů plátu souřadnici v = 0 a u spodních v = 1. A u vrcholů vlevo nastavíme u = 1 a vpravo u = 0. Pokud ale máme třeba v jednom obrázku uloženy vedle sebe 4 textury (viz Obr. 3.9) a chceme na plát nanést tu druhou zleva, změní se nám souřadnice u tak, že na vrcholech vlevo bude u = 0.25 a na vrcholech vpravo u = 0.5.

```
 // vytvořím VB se správnými parametry
celkovyVbKytky = new VertexBuffer(...)
// jakým způsobem je VB zpravován, před prací s VB je třeba ho
vždy uzamknout - Lock
CustomVertex.PositionTextured[] vertex =
(CustomVertex.PositionTextured[])celkovyVbKytky.Lock(0, 0);
     for (int i = 0; i < pocetKytek; i++)
     {
     vertex[(i * 8) + 0] =  new
CustomVertex.PositionTextured(umisteniKytky[i].X – velikostKytky /
2, umisteniKytky[i].Y - velikostKytky / 2, umisteniKytky[i].Z, 0f,
1f);
     vertex[(i * 8) + 1] =  new
CustomVertex.PositionTextured(umisteniKytky[i].X - velikostKytky /
2, umisteniKytky[i].Y + velikostKytky / 2, umisteniKytky[i].Z, 0f,
0f);

      //..............atd...
      }
      //po práci můžeme VB odemknout
      celkovyVbKytky.Unlock();
```

Výpis 3.2: *Ukázka části metody, která naplňuje VB květin. Jak je vidět, je třeba opravdu „ručně“ umístit každý vrchol.*

Pro každý druh VB (myšleno Květiny, Blízký trs, Vzdálený trs) existuje jeho vlastní IB, který sice vypadá pro všechny VB stejně, ale mění se jeho velikost (ta se odvozuje od počtu vrcholů ve VB). Tvar IB viz Výpis 3.3.

```
           // vytvořím IB se správnými parametry
       celkovyIbKytky = new IndexBuffer(...);
      // před prací s IB je třeba ho zamknout a zpřístupnit ho,
přes pole integerů
       int[] index = (int[])celkovyIbKytky.Lock(0, 0);
           //v tomto cyklu vždy říkám, jak je spojen čtverec –
z těch se skládají všechny „ručně“ tvořené modely v tomto progr.
       for (int i = 0; i < pocetKytek * 2; i++)
       {
           index[(i * 6) + 0] = (i * 4) + 0;
           index[(i * 6) + 1] = (i * 4) + 1;
           index[(i * 6) + 2] = (i * 4) + 2;
           index[(i * 6) + 3] = (i * 4) + 2;
           index[(i * 6) + 4] = (i * 4) + 3;
           index[(i * 6) + 5] = (i * 4) + 0;
       }
       celkovyIbKytky.Unlock();
```

Výpis 3.3: *Index Buffer, který říká, jak mají být vrcholy ve VB květin pospojovány. Stejný tvar IB mají všechny mnou vytvářené modely a jediné, co se u něj mění, je jeho rozsah.*

Ve Výpisu 3.2 stojí za povšimnutí, jakého formátu jsou jednotlivé vrcholy (vertexy). Jejich formát je *PositionTextured*. Tento formát je z jednoduchého důvodu. U všech vrcholů samozřejmě potřebujeme znát jejich pozici. Dále také na všech našich modelech je nanesena textura. Normálu u vrcholů neurčujeme, jelikož ta by byla využitelná snad jedině na osvětlení. Po domluvě s konzultantem práce jsme však dospěli k závěru, že osvětlení u trávy je zbytečně výkonově náročné. Bohatě stačí osvětlovat terén pod trávou, který stejně prosvítá. Je však ale tedy nutné mít na paměti, že před vykreslováním trávy musíme vždy vypnout světla. Jinak budou všechny trsy černé.

Nyní jsme tedy již v situaci, že umíme sjednocovat data do větších celků a ty potom vykreslit. Jak velké však mají být ty celky? Kolik trojúhelníků je ideální posílat na grafickou kartu, aby nebyla zahlcena komunikační sběrnice, ale abychom zbytečně neztráceli čas stálým přepínáním mezi CPU a GPU? Z Grafu 4.1 je vidět, že maximum je mezi dvěmi až osmi tisíci trojúhelníků na jeden Vertex Buffer.

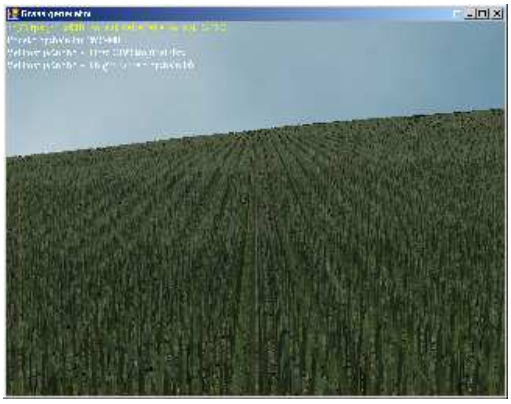
Hlavní předností tohoto algoritmu je to, že se všechno předpočítá v předzpracování (preprocessingu). Tudíž se po načtení úrovně již nemusí nic zásadního počítat, čím se zvyšuje rychlost zobrazování.

3.2.3 Rozmístění trsů a květin po terénu

Při volání konstruktoru třídy *Uroven* se volá metoda *spoctiUmisteniGrass*, která naplní 2D pole *umisteni*. Každá hodnota v tomto poli je pak při stavbě trsu používána jako střed trsu. Metoda ji spočítá ze znalosti levého dolního rohu (odkud začíná růst tráva – zadává se jako parametr při vytváření úrovně), velikosti trsu a ze znalosti maximálního počtu trsů ve VB. Respektive – tato metoda nejprve spočítá souřadnici x a z. Potom se využije jeden ze základních přístupů k tak zvanému neperiodickému dláždění (nonperiodic tiling - [8]), a to randomizace. V každém cyklu vytvoří dvě náhodná čísla v intervalu nula až velikost trsu a jedno potom přičte k x-ové a druhé k z-ové souřadnici. Tento krok je velmi zásadní pro vzhled trávy. Když nepoužijeme metodu neperiodického dláždění, bude tráva vypadat nepřirozeně a bude v ní silně patrné to, že je vlastně celá tvořena z jednoho trsu (viz Obr. 3.15). Naopak i s použitím tohoto jednoduchého neperiodického dláždění se stane vzhled trávy daleko uvěřitelnější (viz Obr. 3.16).

(Pozn.: Metodu randomizace poté využíváme ještě jednou, když posouváme vrchní části trsů o malou kladnou či zápornou hodnotu po y-ové souřadnici. Tím docílíme efektu, že nejsou všechny trsy trávy stejně vysoké.)

Když máme spočítané a posunuté obě výše zmiňované souřadnice, potřebujeme zjistit ještě poslední y-ovou složku. Zde autor narazil na výše zmiňovaný problém (kapitola 3.1.3 *Další modely*), a sice ten, že jelikož se rozhodl vytvořit terén z výškové mapy v 3Ds Max a poté importovat do svého programu již vytvořený model terénu, nemá k dispozici tuto mapu, ze které je velmi snadné zjistit výšku z pixelu x, z. Prostě stačí se jenom podívat na příslušný pixel a hned známe výšku, dle jeho barvy. Bylo tedy nutné přistoupit tomuto problému jinak. Z *Vertex Bufferu*, který získáme z *Meshe* terénu, si „vytáhneme“ všechny trojúhelníkové plošky, z kterých je terén tvořen.



Obr 3.15: *V této scéně není použito neperiodické dláždění a je na ní značně patrné, že se jedná o jeden druh strojově rozmístěného trsu. Takovéto dláždění můžeme najít například v nové hře Kobra 11, kde to také vizuálně velmi ruší.*

Obr. 3.16: *Zde je již využitý náhodný posun každého trsu a trávník se tím jak vidno stává daleko vizuálně přitažlivější. Již není například na první pohled vůbec patrné, že je tráva tvořena jedním trsem.*

Bylo třeba experimentálně vyzkoušet, jak jdou výše zmiňované plošky za sebou. Zjistili jsme, že plošky začínají v levém horním rohu a jdou po řádcích směrem zleva doprava a dolů. Jelikož již víme, že je terén tvořený pravidelnou (regulární) mřížkou, není řešení tohoto problému až tak složité [9]. Ze souřadnic x a –z (z je obráceno, právě z důvodu, že ploška s nejnižším indexem se nachází v levém horním rohu) rovnou zjistíme, v jakém čtverci se nacházíme, a jsme již schopni mu přidělit nějakou souřadnici ze souřadnic definujících plošku. To nám ale nestačí (alespoň ne u terénu, který není rovina) a proto si potřebujeme nejprve rozhodnout, ve kterém trojúhelníku z těch dvou, které tvoří čtvercovou oblast, daný bod leží. Na to potřebujeme znát *offsetX* a *offsetZ* (tzn. *umístění hledaného bodu – umístění v0*; neboli vzdálenosti, které přebývají od levého horního rohu směrem do čtverce) a potom víme, že platí (jelikož jsme na regulárním povrchu) vztah (3.1). Pokud

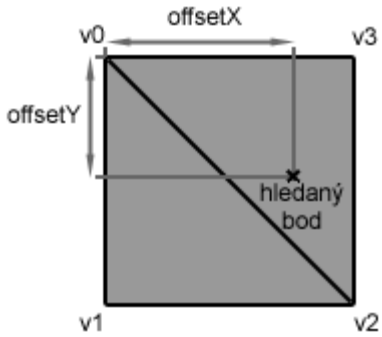
$$offsetX > offsetZ \quad (3.1),$$

bod se nachází v pravém trojúhelníku (viz Obr. 3.17). Naopak, pokud tato nerovnost neplatí, pak je bod v levém trojúhelníku.

Nyní již známe přesný trojúhelník, ve kterém se bod nachází, a přesnou pozici spočítáme podle vzorce (3.2).

$$y = vrchol0.y - \frac{(N.x \cdot offsetX + N.z \cdot offsetZ)}{N.y} \quad (3.2),$$

kde *vrchol0* viz Obr. 3.17.



Obr. 3.17: *Zde je vidět případ, když je offsetX > offsetZ a bod se opravdu nachází v pravém trojúhelníku.*

Pokud neznáme normálu N (to je náš případ), musíme ji spočítat dle vzorce:

$$N = (vrchol1 - vrchol0) \times (vrchol2 - vrchol0) \quad (3.3).$$

Když vykonáme výše zmíněné matematické operace, zjistíme konečně námi požadovanou souřadnici y.

Květiny využívají stejnou metodu pro zjištění y-ové souřadnice, avšak jinak získávají ostatní dvě složky svého vektoru umístění. Jelikož květin není na hrací ploše zdaleka tolik, kolik trsů (je jich maximálně několik tisíc a každá květina = 8 trojúhelníků), můžeme všechny uložit do jediného VB. To nám dovolí použít jednodušší metodu na zatravnění (viz Výpis 3.4), která prostě v rozsahu, ve kterém mají být květiny (čtverec se středem v bodě nula, jehož levý dolní roh definujeme konstruktorem třídy) náhodně vygeneruje tolik hodnot x a z, kolik má být trsů květin.

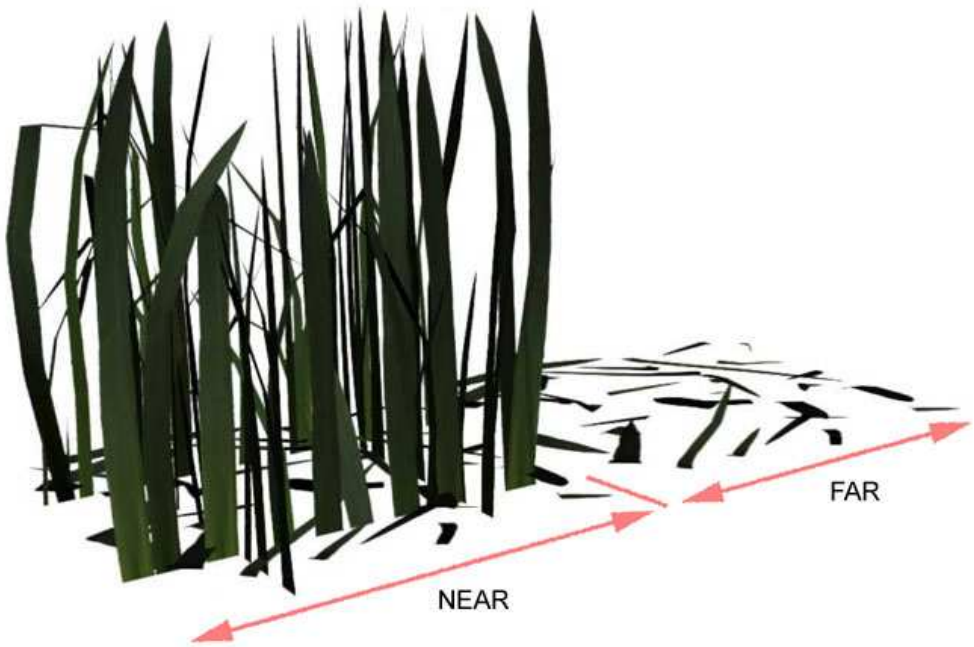
```
      //vytvoříme vektor umisteniKytky, který je tak velký, kolik
má být kytek
      umisteniKytky = new Vector3[pocetKytek];
      //pro kazdou kytku vygeneruji v zadaném rozsahu x a z
            for (int i = 0; i < pocetKytek; i++)
            {
                float x = (float)rand.Next((int)levyDolni.X, -
(int)levyDolni.X) + velikostKytky / 2;
                float z = (float)rand.Next((int)levyDolni.Y, -
(int)levyDolni.Y) + velikostKytky / 2;
             //a pro vygenerované souřadnice zjistím jejich reálné
umístění na terénu
                float y = teren.ZjistiVysku(x, z) + velikostKytky
/ 2;
                umisteniKytky[i] = new Vector3(x, y, z);
            }
```

Výpis 3.4: *Tělo metody, která generuje umístění květin.*

3.2.4 Úroveň detailů

Aby program fungoval rychleji, je žádoucí řídit počet detailů (LoD), které se budou vykreslovat. Druhy detailů, které budeme řídit, jsou dva: velikost textury a počet trojúhelníků. Jejich řízení se dá udělat několika způsoby.

Nejprve si řekneme o řízení počtu trojúhelníků. Již v kapitole 3.2.2. Uložení trsů do větších celků jsme si popisovali, že máme dva druhy Vertex Bufferů. Jeden, kde jsou uloženy trsy tvořené čtyřmi vertikálními pláty (v programu se toto pole jmenuje *VBHighGrass*) a druhý, kde je každý trs tvořený pouze jedním horizontálním plátem (*VBLowGrass*). Důvod tohoto rozdělení do dvou celků byl právě řízení počtu detailů. Princip je jednoduchý. Do nějaké vzdálenosti od uživatele se vykreslují oba dva druhy Vertex Bufferů. Potom se přestane vykreslovat *VBHighGrass* a vykresluje se jen *VBLowGrass* (viz Obr.3.17). A jelikož má být program schopen zatravňovat velmi rozsáhlé plochy, tak musíme předpokládat, že po nějaké vzdálenosti stejně uživatel trávu neuvidí (bude už tak daleko, že tam zkrátka nedohlédne), takže po určité velké vzdálenosti se přestane vykreslovat i *VBLowGrass*.



Obr. 3.17: *2 úrovně detailů. Blízké trsy jsou tvořeny jak vertikálními, tak i horizontálními pláty. Vzdálené jsou potom tvořeny pouze horizontálními pláty.*

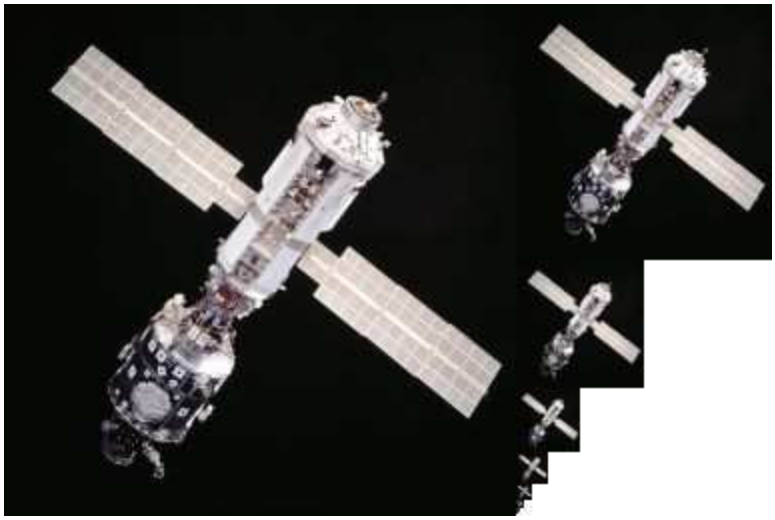
Samotné testování, zdali se má ještě vykreslovat geometrie, zohledňuje vzdálenost středu daného *Vertex Bufferu* od umístění pozorovatele. Vždy, když zavoláme vykreslování *Vertex Bufferu*, uvedeme i nějakou hodnotu *LoDmez* a zkontrolujeme, jestli je výše zmíněná vzdálenost menší. Pokud je menší, trsy se vykreslí (viz Výpis 3.5). Jak velká by měla být vzdálenost *LoDmez* obecně nejde určit. Záleží to na tvaru terénu (když je hodně vlnitý, může být *LoDmez* menší, jelikož člověk stejně nevidí za nejbližší kopec), dále na tom, jestli používáme

v dálce nějaký mlžný opar (za kterým již stejně není vidět nic) a také na velikosti jednotlivých VB. Samozřejmě platí, že je čím menší *LoDmez*, tím je méně vykreslovaných trojúhelníků a tím vyšší rychlost (viz 4.2 *Výsledná měření našeho programu*).

```
 //testuje jestli je vzdálenost středu od pozorovatele menší než
LoDmez
if (Math.Pow(Math.Pow(vbLowStred[i].X - kdeJsme.X, 2) +
Math.Pow(vbLowStred[i].Y - kdeJsme.Y, 2) +
Math.Pow(vbLowStred[i].Z - kdeJsme.Z, 2), 0.5) < LoDmez)
                {//vykresli trsy}
```

Výpis 3.5: *Test, jestli se má vykreslovat daný VBLowGrass.*

Další věcí, která se stará o úroveň detailů, je tak zvaný mipmapping [10]. Tato technika však neredukuje počet trojúhelníků, ale redukuje velikost textur na ně nanášených. *MIP mapy* jsou vlastně předpočítaná kolekce textur, které vycházejí z textury původní. Podmínkou pro tvorbu *MIP map* je, že původní textura musí mít délku stran rovnou *k*-té mocnině čísla 2 (přičemž *k* je libovolné). *MIP mapu* potom tvoří kolekce textur, které mají vždy poloviční délku strany než předchozí textura. Takto se dělí do té doby, dokud není délka alespoň jedné strany textury rovna jedné. Když je velikost původní textury tedy například 256×256 pixelů, vytvoří se z něj dalších 8 textur o velikostech 128×128, 64×64, 32×32, 16×16, 8×8, 4×4, 2×2, 1×1. Potom se všech devět textur uloží do jednoho obrázku (viz Obr. 3.18), ze kterého se v programu potom načítají vždy nejbližší textury, mezi nimiž se interpoluje.



Obr. 3.17: *Ukázka MIP map pro texturu o velikosti 256x256 pixelů.(Obrázek převzatý z [10])*

Uveďme si příklad. V programu je potřeba potáhnout nějaký vzdálenější plát, jehož textura by měla být velká 90×90 pixelů. Z kolekce *MIP map* se potom vezmou textury o velikosti 128×128 a 64×64 a mezi nimi se interpoluje.

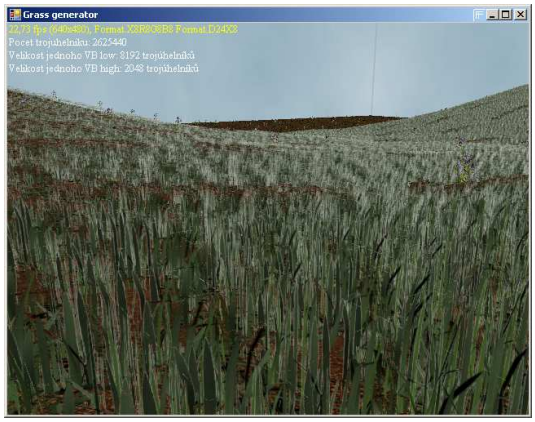
To jednak zlepší vzhled textur potažených na plátech (odstraní se artefakty, které předtím vznikaly nanášením velké textury na menší pláty) a také to zrychlí běh programu, protože není třeba na grafickou kartu mnohdy posílat tak velké textury (viz 4.2 *Výsledná měření našeho programu*).

3.2.5 Alfa průhlednost

Jelikož jsou trsy v programu dělané pomocí billboardů potažených částečně průhlednými texturami, je třeba využít v programu alfa průhlednost a nastavit jí správnou prahovou hodnotu.

V kapitole 3.2.1 *Použitelný trs* byla již vysvětlena otázka tvorby alfa kanálu a jeho podoba. Alfa kanál definuje průhlednost textury. Jedná se v podstatě o další obrázek, který je ve stupních šedi. Tam, kde je černý, je textura, ke které se alfa kanál vztahuje, zcela průhledná. Tam, kde je bílý, je zcela neprůhledná. Na hodnotách barvy mezi těmito dvěmi ohraničujícími barvami je textura potom více či méně průhledná. Prahová hodnota v programu zmenšuje tento interval. Přesněji řečeno, specifikuje referenční hodnotu alfa barvy, proti které jsou testovány pixely na alfa průhlednost. Implicitně je prahová hodnota nastavena na nulu. To znamená, že jenom úplně černá barva pixelu v alfa kanálu představuje úplně průhlednou barvu pixelu v textuře. Problém je však v tom, že pokud nepoužíváme řazení vykreslovaných objektů, prolíná se v místech, kde jsou pixely poloprůhledné, do této textury textura pozadí – v našem případě *skyboxu*.

Pokud nechceme snižovat výkon naší aplikace řazením, musíme nastavit prahovou hodnotu tak, aby chyby na okrajích neprůhledných částí textury nevznikaly. Experimentálním postupem bylo tvůrcem programu zjištěno, že ideální prahová hodnota je 180 až 200. To znamená, že již úplně průhledná je textura tam, kde už má alfa kanál hodnotu 180 až 200 (tzn. světle šedivá barva). Tím sice přestanou být okraje neprůhledných částí textury tolik vyhlazené, ale nevznikají na nich zobrazovací chyby (viz Obr. 3.18 a Obr. 3.19).



Obr. 3.18: *Pohled na scénu s prahovou hodnotou nastavenou na 1. Jak vidíte, chyby vzniklé na okrajích jsou velmi markantní. Celá scéna je tak zbarvena do bílé barvy.*

Obr. 3.19: *Zde je vidět ta samá scéna jako na Obr. 3.18, ale s prahovou hodnotou nastavenou na 190. Scéna má hezkou přirozenou barvu zeleně.*

3.2.6 Vítr

Udělat vlnící se trávu ve větru je jeden z cílů, které jsme si na počátku dali. V kapitole 2.3 *Sumarizace poznatků* bylo uvedeno, že pohyb bude proveden pomocí goniometrické funkce. Tohoto záměru se tedy autor bude držet.

Princip tvorby větru je velmi primitivní. Jediné, co by mohlo pro někoho možná představovat bariéru, je, že pohyb bude dělaný pomocí *High-Level Shading Language* (HLSL). Nejprve si tedy řekneme něco málo o tomto jazyku.

HLSL je jazyk, pomocí něhož můžeme naprogramovat grafickou kartu. To v podstatě znamená, že můžeme programovat rovnou na GPU, což umožňuje podstatně zrychlit práci. Pomocí shaderu můžeme dělat různé efekty (fx), například i pohyb ve větru. Efekt se skládá ze 2 kompaktních celků (v podstatě již ze tří – zcela nově totiž přibyl ještě *Geometry Shader*), a to z *Vertex Shaderu* (VS) a *Pixel Shaderu* (PS). *Pixel Shader* nahrazuje v našem případě pouze texturování. Tzn. jeho vstupem je *Vertex Shaderem* předaná souřadnice textury. Z této souřadnice a z textury potom vypočte pro každý pixel správnou barvu, která je jeho výstupem.

Daleko důležitější pro náš efekt je však *Vertex Shader*. Jeho vstupem je pozice daného vrcholu (vertexu), který do něj posíláme, a také jeho texturová souřadnice. Ještě je potřeba říci, že do tohoto efektu se z našeho programu předává proměnná *appTime*, která obsahuje čas aktuálně uplynulý od startu aplikace.

Souřadnici vrcholu musíme vynásobit maticí, která vznikne násobkem všech matic v programu ($\text{World} \times \text{View} \times \text{Projection}$), abychom dostali vrchol ve správné pozici. Mimoto si zavedeme proměnnou *x*:

$$x = délkaPoziceVrcholu \cdot \sin(appTime) \cdot 20 \quad (3.4).$$

Nakonec si do proměnných *fCos* a *fSin* uložíme $\cos(x)$ a $\sin(x)$ a tyto proměnné přičítáme k x-ové a y-ové složce každého vrcholu, jehož texturová souřadnice $v < 0.5$ (viz Výpis 3.6). Tím zajistíme, že se budou pohybovat jenom vrchní vrcholy a tím pádem jenom vrchní části trsů, stejně jako v reálu se při větru taky hýbají jenom stébla a ne celé trsy.

```
//pokud je souřadnice v nižší než 0.5 (vrchní vrcholy)
if (uv.y < 0.5)
{
      // změň x-ovou a y-ovou souřadnici vrcholu o hodnotu
      založenou na čase a transformuj vrchol do projekčního
      prostoru
      Output.Position = mul(float4(vPosition.x + (fCos *
0.2f),vPosition.y, vPosition.z + (fSin * 0.1f), 1.0f),
worldViewProjection);
}
```

Výpis 3.6: *Část kódu VS, která se stará o simulaci pohybu vrcholu ve větru*

Jak jsme si již uváděli, operace prováděné na GPU jsou velmi rychlé. Pokud však je v nějakém programu (např. ve hře) použit efekt, který je výkonově náročný, tak většinu výkonu sebere PS. Operace prováděné ve VS na GPU jsou oproti tomu v podstatě zdarma. Proto i námi implementovaný efekt větru, tvořený pomocí *Vertex Shaderu*, neovlivňuje nijak znatelně rychlost aplikace.

3.2.7 QuadTree

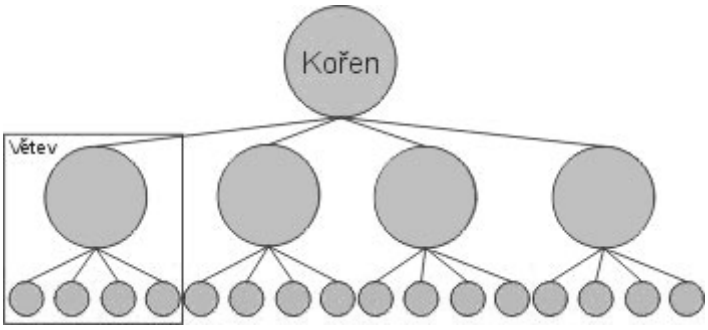
Je samozřejmé, že nejrychlejší polygon je ten, který se vůbec nekreslí. V náročných aplikacích pracujících s velkým množstvím polygonů (např. počítačové hry, generátory terénů...) je důležité co nejefektivněji využívat všech dostupných hardwarových prostředků a tím zajistit maximální výkon aplikace.

Jako příklad si vezměme, že v naší aplikaci máme nastavenu *LoDmez* (viz kapitola 3.2.4 Úroveň detailů), po které nejsou vykreslovány blízké trsy (tedy např. *LoDmez = 48*). Každý trs je tvořen ze čtyř plátů, což je 8 trojúhelníků. To může znamenat, že budeme zobrazovat např. 110 000 trojúhelníků. Námi viděný výsek je však zpravidla daleko menší (např. čtvrtinový). Je samozřejmě možné vykreslit všechny trsy, ale to je opravdu velmi neefektivní způsob! Bohatě nám většinou stačí vykreslovat jenom malou část celého terénu, a to tu, na kterou se právě díváme (tzv. *Viewing Frustum* – pohledový jehlan, nebo také výhled kamery).

Jako první řešení, které napadne řadu lidí je projíždět všechny VB a u každého zjišťovat, jestli je viditelný nebo ne. Tento způsob je však zbytečně výkonově náročný. Proto je třeba metoda, která zjišťuje viditelnost nějakým jiným, efektivnějším postupem.

Existuje řada metod, které se zabývají rychlým rozhodováním, co vykreslovat a co ne. Jednou z nich je právě metoda *QuadTree*, kterou využívá náš program.

QuadTree je vyvážený strom, jehož rodič má vždy 4 potomky. Tato metoda vezme ohraničující krychli (*Bounding Box*), která ohraničuje všechny trsy a uloží ji do kořene *QuadTree*. Potom vezme postupně všechny 4 polygony, které vzniknou rozdělením ohraničující krychle na čtvrtiny (viz Obr. 3.21), a uloží je na místo jeho potomků (Obr. 3.20). Tímto způsobem zaplní celý strom, který bude mít hloubku rovnou počtu dělení detailnější trávy. Bylo by možné jít i níž, ale jelikož je detailní tráva uložena v nedělitelných *Vertex Bufferech*, je zbytečné dělit strom více, než je právě velikost těchto *Vertex Bufferů*.

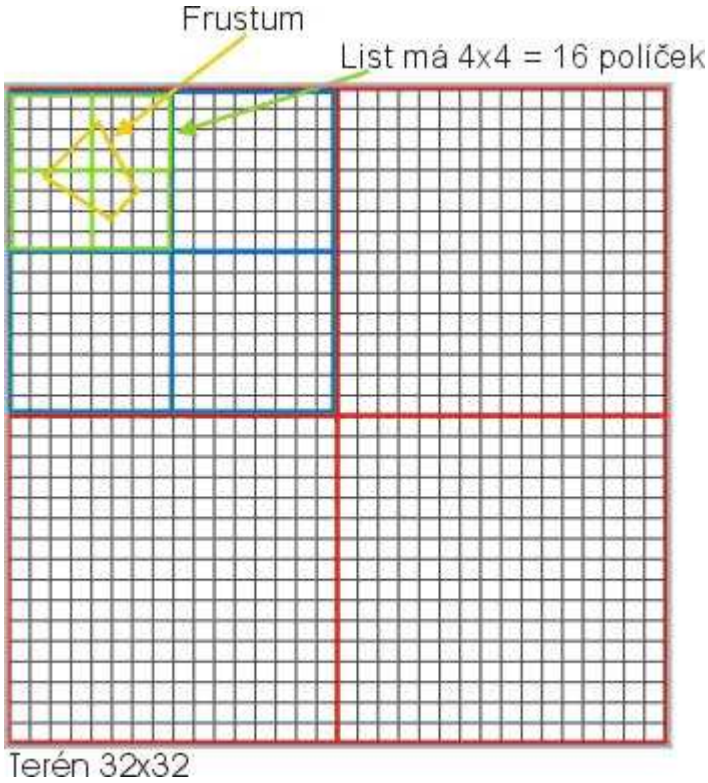


Obr. 3.20: *Ukázka QuadTree s hloubkou 3.*

Při zjišťování viditelnosti začne metoda u kořene *QuadTree*. Ten není třeba kontrolovat, přejde tedy na jeho potomky a zjistí, na kterém z nich se nachází výhled kamery. Ten, kde se nachází, vezme a vnoří se do něj (ostatní zahodí) a postup opakuje. Postup končí, pokud se výhled nachází na více jak jednom potomku. Potom vezme rodičovský uzel těchto potomků (viz barevné čtverce – ohraničující krychle (*Bounding Boxy* - BB) na Obr. 3.21) a vrátí ho. S tímto rodičovským uzlem je poté porovnáván každý *Vertex Buffer*. Pokud se daný VB alespoň z části nachází v oblasti, kterou uzel ohraničuje, je vykreslen. Pokud se nachází mimo tuto oblast, není vidět, a tudíž není vykreslován

Na Obr. 3.21 si ještě jednou ukážeme, jak bude postupovat program při zjišťování viditelnosti polygonů. Vezme největší BB (šedivé ohraničení) a otestuje jeho 4 červené potomky. Zjistí, že pohledový jehlan (na Obr. 3.21 *Frustum*) se nachází v levém horním rohu – do toho se zanoří. Otestuje modré ohraničující krychle a zanoří se opět do levého horního. Otestuje zelené a zjistí, že pohledový jehlan se nachází na všech čtyřech. Proto metoda vrátí levý horní modrý BB, proti kterému budeme později zkoušet viditelnost jednotlivých VB.

Tato metoda značně urychluje vykreslování programu, jelikož nekreslíme, co není vidět (více o urychlení v kapitole 4.2 *Výsledná měření našeho programu*).



***Obr. 3.21**: Konkrétní příklad QuadTree, kdy se frustum nachází vlevo nahoře. Metoda QuadTree se zanoří až na zelené Bounding Boxy, u kterých zjistí, že se již neumí rozhodnout, kde se nachází pohledový jehlan. Proto vrátí levý horní modrý Bounding Box.*

4. Experimenty a výsledky

Než provedeme závěrečnou sumarizaci, uvedeme ještě některá měření a pozorování, ze kterých později budeme moci lépe odvodit míru úspěšnosti našeho řešení.

Parametry hardwarového i softwarového vybavení počítače, na kterém byla měření prováděna je uveden v Tab. 4.1.

Počítač	CPU: Intel Core2 T5500 1.66 GHz RAM: 1 GB HDD: 100GB (5400 ot./s) Grafická karta: Ati Mobility Radeon X1450, 128 MB fyzické paměti
Systém	Microsoft Windows XP Professional Verze 2002 Service Pack 2
Vývojové nástroje	Microsoft DirectX SDK (December 2005) Microsoft Visual C# 2005 Express Edition .NET framework 2.0
Výkon	3D Mark 2001 (nastavení „Default“): 7925 bodů

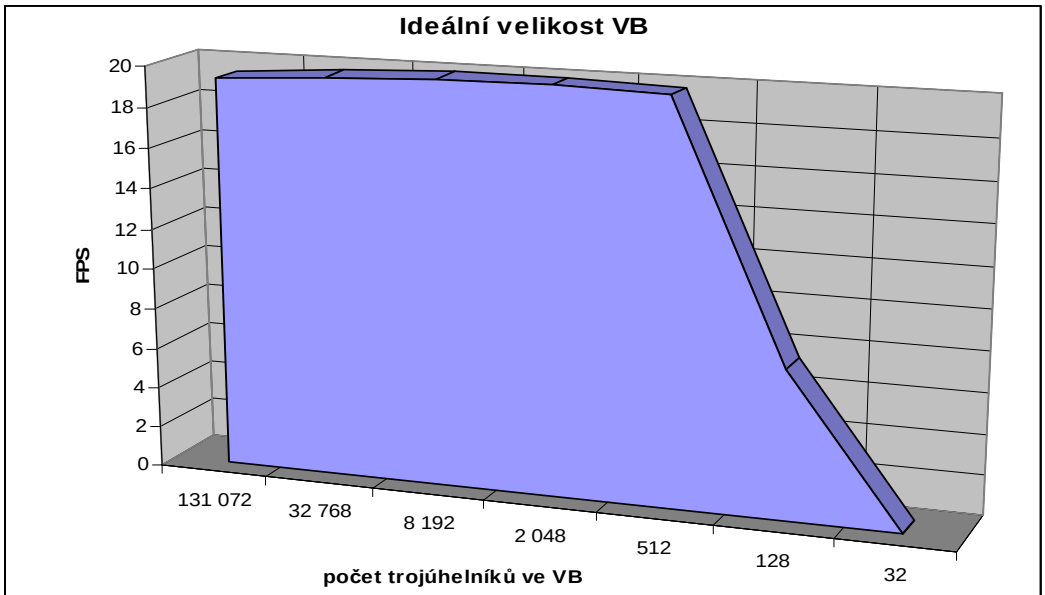
Tab. 4.1: *Tabulka s parametry HW a SW vybavení testovacího počítače*

4.1 Experimenty a výběr vhodného řešení

Náš první experiment se týká ideálního počtu trojúhelníků ve VB. Při tomto experimentu vykreslovala aplikace konstantních cca 10,5 milionu trojúhelníků. Parametr, který jsme měnili, byl počet trojúhelníků v jednom *Vertex Bufferu*, a hodnota, kterou jsme měřili, byla počet vykreslených snímků za sekundu (FPS). Celé měření je zaznamenáno v Grafu s tabulkou 4.1.

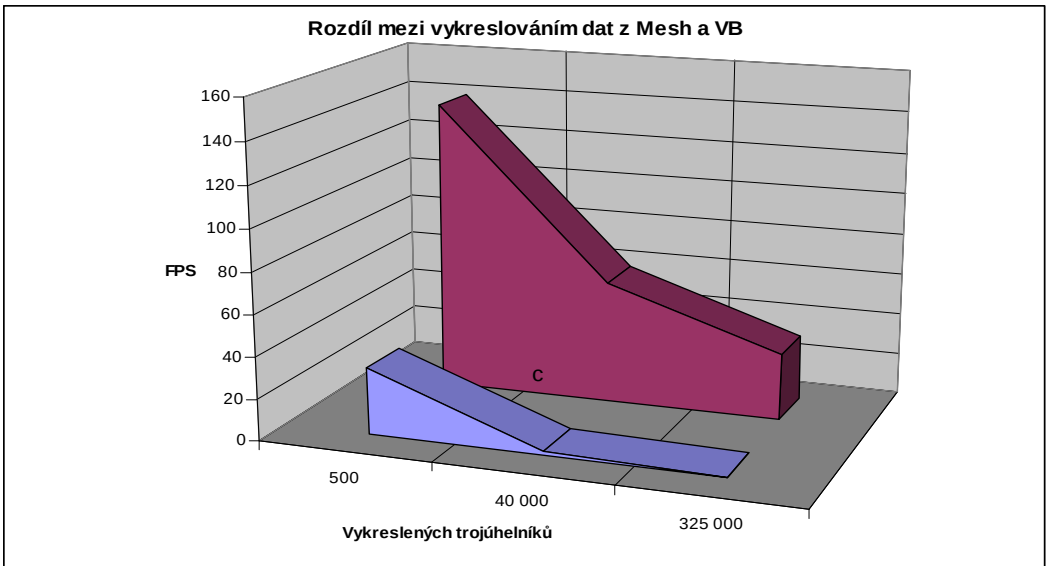
Měření pro tento graf byla dělána při hardwarovém režimu zobrazování, byly použity dvě textury o velikosti 256×256 a 1024×256 pixelů. Nebyly zapnuty žádné optimalizace. Pro orientační meze *Vertex Bufferů* tento odhad velmi dobře postačuje.

Z Grafu s tabulkou 4.1 vidíme, že ideální velikost bufferu je v rozmezí 2 až 8 tisíc trojúhelníků. Je také vidět, že zatímco při zvyšování počtu trojúhelníků výkon téměř neklesá, tak při snižování po určité mezi (512 trojúhelníků) klesá velmi dramaticky.



	131 072	32 768	8 192	2 048	512	128	32
FPS (cca 10 500 000 zobrazených trojúh.)	19,35	19,66	19,89	19,89	19,77	7,33	0,0004

Graf s tabulkou 4.1*: Z tohoto grafu je vidět, že ideální je posílat v jednom Vertex Bufferu něco mezi dvěmi až osmi tisíci trojúhelníků.*



	500	40 000	325 000
X - průměrné FPS (bez optimalizací)	32,01	1,84	0
VB - průměrné FPS (bez optimalizací)	140,04	59,01	31,82

Graf s tabulkou 4.2: *Na tomto grafu je vidět výrazně vyšší počet vykreslených snímků za sekundu pokud vykreslujeme trsy uložené ve VB, oproti tomu, když každý trs načítáme ze samostatného souboru .X.*

Na dalším experimentu si ukážeme, o kolik je výhodnější využít při tvorbě trávy velké *Vertex Buffery*, obsahující stovky a více trsů, než načítat každý trs samostatně ze souboru *.X.

Následující Graf s tabulkou 4.2 ukazuje velmi výrazný rozdíl v rychlosti vykreslování takovýchto scén. Jak vidíme, čím je počet vykreslovaných trojúhelníků vyšší, tím se rozdíl ještě více zvětšuje. Při 325 tisících trojúhelníků již nebylo možné scénu s trsy načtenými ze souboru .X vůbec vykreslit.

Vykreslování bylo děláno při vypnutých optimalizacích s texturami o velikosti 256×256 pixelů.

4.2 Výsledná měření našeho programu

Následná měření mají za úkol změřit průměrný počet snímků za sekundu (*Frame Per Second* - FPS) v naší aplikaci. Je třeba si uvědomit, že obnovovací frekvence je závislá na pozici kamery. Měření tedy probíhala tak, že jsme se snažili dostatečně dlouhou dobu prolétávat v různých výškách i pozicích k trávníku a po celou dobu jsme počítali programově průměrnou obnovovací frekvenci.

V následující Tab. 4.1 je vidět, jak roste výkon naší aplikace s použitím různých technik pro urychlení vykreslování. Měření byla dělána při hardwarovém režimu zobrazování, byly použity dvě textury o velikosti 256×256 a 1024×256 pixelů. Vykreslovaná scéna byla tvořena z cca 170 tisíc trojúhelníků, které tvořili trávu. Další přibližně 2 tisíce trojúhelníků tvořil terén.

cca 170 tisíc trojúhelníků (bez terénu ~ 2000 troj.)	bez LoDmez, QuadTree a MIP map	bez LoDmez a QuadTree	bez QuadTree	se vším
průměrné FPS	20,2	89,5	110,7	127,2

Tab. 4.1: *Porovnání rychlosti vykreslování s vypnutými a zapnutými optimalizacemi. Jak vidíme nárůst výkonu se zapnutými optimalizacemi je více než 6ti násobný.*

Jak vidíme, zatímco se všemi vypnutými optimalizacemi dosahovala průměrná rychlost vykreslování pouhých 20 snímků za sekundu (fps) se zapnutými *MIP mapami* již bylo vykreslování více než 4 násobně rychlejší. Potom jsme nastavili *LoDmez* (tzn. vzdálenost od pozorovatele, kdy se již trsy nekreslí) na hodnotu 32 pro blízké trsy a 80 pro vzdálené (více viz 3.2.4 Úroveň detailů). To nám zvedlo výkon o dalších 20 snímků. Nakonec jsme zapnuli *QuadTree* a s těmito všemi technikami naše aplikace dosahovala průměrně téměř 130 snímků za sekundu, což je oproti počátečním 20ti více jak šestinásobný nárůst.

V Tab. 4.2. máme změřen počet snímků, kterých dosáhne naše aplikace se všemi zapnutými optimalizacemi, při rozdílném počtu trojúhelníků, které nám tvoří travnatou plochu. Měření byla dělána při hardwarovém režimu zobrazování, byly použity stejné textury, jako v měření výše.

počet trojúhelníků (bez terénu ~ 2000 troj.)	průměrné FPS (se vším)
0 (tzn. jen terén)	302,3
cca 170 000	127,2
cca 660 000	99, 8
cca 2 630 000	69, 7
cca 16 390 000	29,3

Tab. 4.2: *Měření rychlosti aplikace při rozdílné velikosti travnaté plochy. Je vidět, že náš program si bez problémů poradí v reálném čase i s travnatými plochami tvořenými několika miliony trojúhelníků.*

Je tady zřejmé, že ani se zobrazováním velkých ploch, tvořených z několika milionů trojúhelníků, nemá náš program potíže. Naopak, dokáže je vykreslit v reálném čase s velmi dobrou průměrnou obnovovací frekvencí (na našem testovacím stroji desítky FPS).

Podívejme se dále na srovnání naší metody s hrou, kterou jsme na začátku uváděli. Rychlost zobrazování velkého počtu stébel má náš program určitě na stejné úrovni, jako metody využívané v dnešních komerčních hrách. Jak je na tom po stránce vizuální? Na následujících Obr. 4.1 a 4.2 vidíme srovnání s *Far Cry*.

Obr. 4.1: *Ukázky travnatého povrchu ze hry Far Cry. Jak vidíme stébla trávy jsou nedetailní a oproti naší metodě jsou to spíše barevné fleky. Hezká je barevnost a sytost textur.*

Obr. 4.2: *Ukázky travnatého povrchu, vytvořeného naším programem. Textury trávy jsou oproti Obr. 4.1 detailnější.*

Tráva vypadá v obou případech naprosto odlišně a přitom je uvěřitelná a realistická u obou dvou. V našem přístupu se ale tráva lépe chová ve větru, což bych viděl jako jednu z hlavních výhod oproti *Far Cry*. Další výhodou jsou potom naše daleko detailnější textury pro stébla, takže když stojíme blízko stébla, vidíme jeho ostré hrany. Oproti tomu ve hře *Far Cry*, když jsme v blízké vzdálenosti od stébla, vidíme už jenom neurčitý barevný flek.

Ovšem výhody má i konkurent. Jeho tvůrci například zvolili sytější barvy (což si mohli dovolit, hlavně kvůli zasazení hry do prostředí tropického ostrova), které více lahodí oku.

Jako zhodnocení vizuální stránky však musím říct, že naše metoda se může s přístupy ve kvalitních komerčních hrách bez problémů srovnávat, což je, myslím, dostatečná známka kvality.

Pokud chceme zkoumat další vzhledové vlastnosti naší metody, tak obrázky z naší metody jsou také přiloženy v Příloze (Obr. Příl.1 a Příl.2). Doporučuji však vyzkoušet si program z přiloženého CD, popř. spustit si krátký film s průletem nad námi vytvořeným travním povrchem, protože ze statických obrázků není viditelný (dle mého názoru) povedený efekt větru.

5. Závěr

Předmětem této práce bylo modelování a vykreslování trávy pro herní aplikace. Jako hlavní priority jsme si stanovili rychlé vykreslování rozsáhlých zatravněných ploch, s aplikací větru a atraktivním vzhledem. Naše řešení využívá osově-zarovnaných billboardů, na které jsou naneseny kvalitní částečně-průhledné textury, vytvořené ve 3Ds Max a Adobe Photoshopu. Tyto billboardy uchováváme ve velkých celcích (*Vertex Bufferech*), abychom odstranili režii vzniklou při kreslení velkého množství malých objektů. Pro řízení počtu detailů textur využívá program *MIP mapping*. Řízení počtu vykreslovaných trojúhelníků je zajištěno pomocí dvou různě detailních trsů. Jednoduché horizontální trsy se vykreslují ve všech vzdálenostech od pozorovatele (kromě natolik vzdálených trsů, jejichž vykreslování již ztrácí smysl) a druhé, detailnější, tvořené čtyřmi vertikálními billboardy, se vykreslují pouze v blízké vzdálenosti od pozorovatele. Z důvodu zamezení vykreslování objektů, které nejsou vidět, využíváme techniku *QuadTree*. Do trávníku jsou pro vizuální oživení zasazeny rostliny. Na ně a na detailnější trsy je poté volitelně aplikován vítr.

Použité metody se osvědčily, výsledek je vizuálně příjemný a dostatečně rychlý pro herní aplikace, takže se lze domnívat, že cíle práce byly splněny.

Autor prezentoval postupy a výsledky této práce ve své přednášce [11] v rámci Game Developers Session 2007. Přednáška měla kladné ohlasy, které se týkaly zejména grafického vzhledu zatravněných ploch a vysoké rychlosti jejich zobrazování.

Již v průběhu vývoje a hlavně také v diskuzi po výše zmíněné přednášce [11] se také objevilo několik zajímavých tipů a nápadů, jak práci vylepšit a kudy se ubírat dál. Například využít BTF k osvětlení travnatých ploch nebo vytvořit nejbližší trsy pomocí reálné geometrie. Zajímavým nápadem také bylo uložit všechno potřebné pro vykreslování na grafickou kartu a poté pouze „tahat za nitky“. Další možností je využít *Geometry instancing*. Tyto dvě metody by zřejmě ještě znatelně zvýšily rychlost vykreslování.

Přehled zkratek

2D – Dvoj-rozměrné (rovina)

3D – Troj-rozměrné (prostor)

BB – Bounding Box (krychle ohraničující nějaké objekty)

BTF - Bidirectional Texture Funtions (technika tvorby osvětlení)

C# - Objektový programovací jazyk vycházející z jazyka C++ a Java

C++ - Objektový programovací jazyk vycházející z jazyka C

CPU – Central Processor Unit (centrální čip počítače)

GBR - Geometry Based Rendering (vykreslování reálně geometrie)

GPU - Graphical Processor Unit (centrální čip grafické karty)

HLSL - High-Level Shader Language (jazyk pro programování grafických karet)

HW - Hardware (fyzické vybavení počítače)

IB - Index Buffer (pole indexů, které říkají, jak pospojovat vrcholy uložené ve Vertex Bufferu)

IBR - Image-Based Rendering (vykreslování pomocí obrázků namísto geometrie)

LoD - Level of Detail (úroveň detailů, pro zrychlování vykreslení scény)

PS - Pixel Shader (rasterizační program)

QT – QuadTree (strom, kde má uzel vždy 4 potomky – používá se obvykle ke zjištění viditelnosti částí scény)

RPG - Role Playing Game (hra na hrdiny, každý se stará o svého avatara, vylepšuje mu vlastnosti apod.)

SW - Software (programové vybavení počítače)

VB - Vertex Buffer (pole vrcholů)

VS - Vertex Shader (program na grafické kartě, který se stará o práci s vrcholy)

Literatura

[1] K. Boulanger, S. Pattanaik and K. Bouatouch. Rendering Grass in Real-Time with Dynamic Light Sources and Shadows. INRIA research report RR-5960, July 2006.

[2] M . Levoy. Expanding the Horizons of Image-Based Modeling and Rendering. http://www.graphics.stanford.edu/talks/imbased/slides/walk001.html, August 1997.

[3] E. Galin, P. Poulin. Real-time Rendering of Realistic-looking Grass. Eurographic Workshop on Natural Phenomena, 2005.

[4] A. Meyer, F. Neyret and P. Poulin. Interactive Rendering of Trees with Shading and Shadows. Eurographic Workshop on Rendering, July 2001.

[5] K. Boulanger, S. Pattanaik and K. Bouatouch. Rendering Grass in Real-Time with Dynamic Light Sources and Shadows. Web-sites of the „GRDLS“ Project , May 2007.

[6] R. Habel, M. Wimmer, S. Jeschke. Instant Animated Grass. Journal of WSCG, 15(1-3):123-128, January 2007

[7] L. Roleček. Tvorba opakovacího pozadí ve Photoshopu. http://interval.cz/clanky/tvorba-opakovaciho-pozadi-ve-photoshopu/, June 2000.

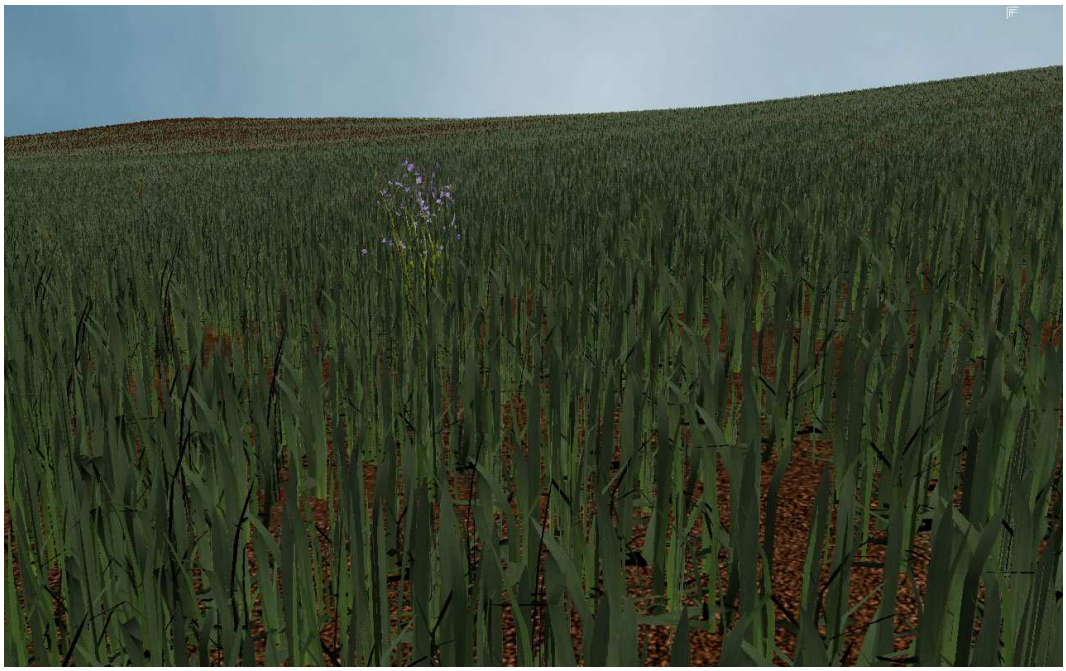
[8] T. Harrison et al. Aperiodic Tiling. Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Aperiodic_tiling, September 2006.

[9] K. Ditchburn. Terrain. Toymaker. http://www.toymaker.info/Games/html/terrain.html, 2004 - 2007.

[10] R.J.Hall et al. Mipmap. Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Mipmap, June 2005.

[11] V.Geršl, M. Novosad. Moderní grafické efekty. Game developers session. http://www.ceske-hry.cz/gds/gds2007.html, June 2007.

Příloha Grafické výstupy



Obr. Příl.1: *Dva pohledy na terén s texturou z obrázku Obr. 3.13.*

Obr. Příl.2: *Dva různé pohledy na terén pokrytý texturou z Obr. 3.14.*