VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

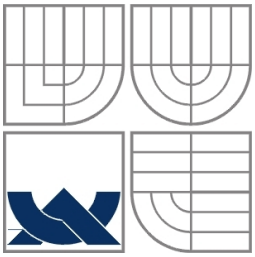
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZOBRAZOVÁNÍ „NÍZKÉ“ VEGETACE VE 3D PROSTORU

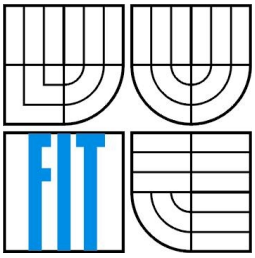
DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

AUTOR PRÁCE Bc. Jiří Řehánek
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZOBRAZOVÁNÍ „NÍZKÉ“ VEGETACE VE 3D PROSTORU

LOW VEGETATION RENDERING IN 3D SPACE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Jiří Řehánek

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Adam Herout, Ph.D.

BRNO 2008

Abstrakt

Tato diplomová práce se zabývá zobrazováním nízké vegetace ve 3D prostoru v reálném čase. Nízkou vegetací se myslí tráva a keře. Nejdříve se podívá na již existující implementace vegetace ve 3D hrách, kde se prozkoumají vlastnosti pozorovatelné při samotném hraní a dojde k hodnocení vizuální kvality. Poté následuje návrh postupu při řešení této problematiky a popis implementace. V závěrečné části se shrnou dosažené výsledky.

Klíčová slova

Terén, vegetace, tráva, keře, počítačová grafika, OpenGL, GLSL, průhledná textura, míchání textur, alfa test, test viditelnosti, správa paměti

Abstract

This masters thesis deals with a real-time low vegetation rendering in 3D space. The low vegetation means grass and shrub. At first it looks on a implementation of vegetation in a 3D games that already exist and there investigate characteristics notable while playing and after that it comes to a evaluation of visual qualities. As next will follow design of solution of this problems and description of implementation. In final part will be summary of achieved product.

Keywords

Terrain, vegetation, grass, shrub, computer graphic, OpenGL, GLSL, texture blending, alpha test, visibility test, memory management

Zobrazování „nízké“ vegetace ve 3D prostoru

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením
Ing. Adama Herouta, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jméno Příjmení
Datum

Poděkování

Rád bych poděkoval vedoucímu Ing. Adamovi Heroutovi, Ph.D. za usměrňování při řešení této práce a některé zajímavé nápady. Rovněž bych rád poděkoval rodině za podporu během celé té dlouhé doby studia.

Obsah

1 Úvod

Při tvorbě mnoha programů zobrazujících rozsáhlé exteriéry se většinou nejdřív řeší zobrazování krajiny, její textura a poté různé velké objekty, které na ní leží. Až jako poslední obyčejně přijde na řadu vykreslování vegetace. A právě tato součást grafiky exteriérů dodává větší pocit reality. Tato práce si bere za cíl prozkoumat způsoby, jakými je možné na současných počítačích v reálném čase na velké ploše zobrazovat tzv. „nízkou“ vegetaci.

Touto „nízkou“ vegetací se myslí převážně tráva, rostliny a keře, což jsou objekty, které se mohou, bez většího dopadu na výsledný obraz, často opakovat. Například ke stromům by už bylo nutné přistupovat poněkud jinak, než bude popisováno v tomto dokumentu.

Ve většině knižních publikací a i na internetu se až na výjimky [1] této tématice věnuje jen málo pozornosti, proto bylo potřeba využít postupů aplikovaných na jiné situace a popřípadě vymyslet vlastní metody řešení problémů, které se během vývoje naskytly. Hlavní inspirace je popsána ve druhé kapitole, kde budou zkoumáni zástupci z řad počítačových her, na kterých lze vidět největší pokrok v této oblasti. Výrobci her si však své postupy tají, proto bude nutné přistoupit k jisté formě reverzního inženýrství. U každé hry budou popsány vlastnosti zobrazovaní již výsledné vegetace a zhodnocen její celkový visuální dojem. V závěru kapitoly se pak shrnou všechny vypozorované vlastnosti.

Třetí kapitola, na základě vypozorovaných vlastností, navrhne postupy, jakými se bude vegetace vytvářet a zobrazovat. Nejdříve se popíší data, které budou používána. Poté následuje uložení v paměti a metody optimalizace vykreslování. Jako poslední přijdou na řadu efekty.

Nejrozsáhlejší čtvrtá kapitola detailně popisuje implementaci projektu. Zmíní se nástroje použité při tvorbě, podrobně se popíše vznik použitých modelů. Pokračuje se výčtem struktur a jejich vzájemné propojení. Kapitola je uzavřena popisem vykreslování, použití shaderů a správou paměti.

A konečně pátá kapitola popisuje výsledek tohoto projektu a jeho použití v již existujících systémech zobrazující 3D krajinu.

2 Vegetace v počítačových hrách

Jako hlavním inspiračním zdrojem pro tuto práci byly počítačové hry. V posledních letech rychlost hardwaru umožnila přidat další geometrii, kterou bylo před tím nutné vynechat. Kde dřív musela stačit zelená textura naznačující trávu, tam je dnes vyžadována hezky se vlnící, pokud možno co nejvíce realistická, vegetace.

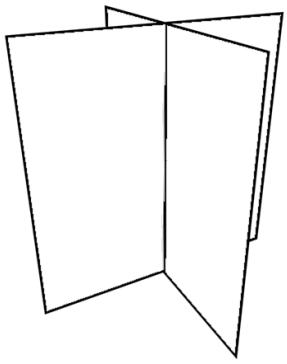
Z mnoha žánrů využívajících 3D grafiku patří FPS (first person shooter – střílečka z pohledu první osoby) právě mezi ty, kde lze vývoj v oblasti zobrazování vegetace nejvíce vidět. Pro studium této problematiky byly vybrány čtyři hry. Každá má trochu jiný přístup k zobrazování trávy a keřů. V těchto vybraných hrách bude sledován způsob, jakým je nízká vegetace vytvořena, její chování v klidném stavu, chování při přibližování, vzdalování a jakým způsobem mizí. U každé hry je na konci zhodnocen celkový dojem. U všech her jsou vlastnosti zkoumány pokud možno při nastavení maximálních detailů, což znemožnilo zkoušet například nejnovější Crysis, kde je vegetace zase o stupeň lépe zpracována. Bohužel v době tvorby této práce nebyl k dispozici počítač, který by byl schopen spustit hru na maximálních detailech.

2.1 Serious Sam: The Second Encounter

V roce 2002 vyšlo pokračování počítačové hry Serious Sam. Jednalo se o poměrně prostou FSP, kde hlavním úkolem bylo prostřílet se na konec úrovně. Co je však z pohledu této práce důležité, obsahuje velké exteriéry, na kterých se určitým způsobem vykresluje vegetace.

2.1.1 Zpracování vegetace

K zobrazování nízké vegetace se v Serious Sam: The Second Encounter [2] přistupuje poměrně jednoduchým způsobem. Tráva je zobrazována jako vysoké trsy (do poloviny výšky postavy). Velikosti jednotlivých kusů se lehce liší. Tyto trsy jsou vytvořeny, jak je znázorněno na obrázku 1.



Obrázek 1: Model trsu trávy

Jde vlastně o dva čtverce v prostoru, které jsou postaveny do kříže a je na nich textura. Na tuto texturu je použita plynulá průhlednost (blending). Takto vytvořená tráva je rovnoměrně rozmístěna po předem určené ploše, protože existují místa, kde například v pravidelném kruhu nic neroste. Takovéto okraje nejsou nijak zvlášť upraveny (například menší trávou).

Je potřeba se zmínit o tom, že kromě menší trávy se ve hře ještě vyskytují větší porosty, které dosahují velikosti postavy. Většinou se jedná o velké kapradiny, nebo rostliny nápadně připomínající lopuchy. Ty jsou oproti malým trsům vytvořeny složitějším modelem, který také používá průhlednou texturu, ale místo blendingu je použit alfa test. Což znamená, že jsou pixely buď absolutně průhledné, nebo absolutně neprůhledné.

Další rozdíly je možné naleznout v chování. Menší trsy se nehýbají, zatímco tyto velké „keře“ se lehce pohybují jakoby ve větru. Znatelný rozdíl je i v objevování a mizení v dálce. Malá tráva má bližší horizont než keře. U keřů lze navíc zpozorovat používání metody LOD (Level Of Detail). Co mají oba druhy společného je způsob objevení/zmizení – změna průhlednosti. Společné je i celkové osvětlení, pokud je zem ve stínu hory, tak je tráva taky tmavější. Ani tráva nebo velké keře nevrhají na podklad stíny.

Obrázek 2: Ukázka ze hry Serious Sam: The Second Encounter

2.1.2 Celkový dojem

Serious Sam: The Second Encounter se nesnaží o super realistickou grafiku. Ta je spíše comicsového charakteru hýřící barvami, tudíž tomu odpovídá i zpracování vegetace. Příběh se navíc odehrává v exotickém prostředí, takže je tráva nepřirozeně zelená a objevuje se zde rostlinstvo, které v našich podmínkách příliš neroste.

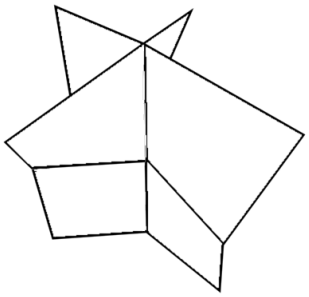
Hra na objektech (a převážně na zdech) používá výjimečně detailní textury, bohužel tomu tak není u trávy. Při bližším zkoumání trávy se ukáže, že textura má nedostatečné rozlišení, což působí rušivým dojmem. Stejně tak působí i jistá poloprůhlednost malých trsů a jejich jednoduchost. Mnohem lépe jsou na tom právě větší keře, hlavně díky větší složitosti a přesnějšímu alfa testování průhlednosti.

2.2 Far cry

O dva roky později na jaře 2004 vyšla hra, která již při svém ohlášení lákala na úchvatné zpracování vegetace tropické džungle. Far cry opravdu přineslo pokrok při využívání možností grafických karet a ze všech zde uváděných her je graficky nejvíce zaměřena právě na zobrazování husté flóry.

2.2.1 Zpracování vegetace

Far cry [3] je pravým opakem předchozí hry, co se týče přístupu k řešení zobrazování trávy a keřů. Zde je i ten nejmenší trs trávy vymodelován mnohem pečlivěji, příklad na obrázku 3. Ve hře se nachází daleko složitěji vytvořené rosliny.



Obrázek 3: Model trsu trávy

Hra obsahuje velké množství druhů rostlin. Všechno jsou to modely s velmi malým počtem trojúhelníků využívající textury s průhledností. U této hry je mnohem více využíván spíš alfa test, než blending. Blending používá jen ta nejmenší vegetace.

Rozmístění trávy a keřů je přizpůsobeno okolí. U Far cry už nejde jen o rozlehlé pláně, ale spíš o členitý terén plný kamenů, prudkého svahu a vyšlapaných stezek. Z toho vyplývá, že i podkladová textura je mnohem složitější a pozice vegetace na ní vložené tomu odpovídá. Přechod mezi například

pískem na pláži a džunglí je velmi pečlivě vytvořen přechodem textur země, poté nastupují menší trsy trávy a nakonec velké keře a stromy.

Podobně jako předešlá hra má i tato více druhů rostlinstva, které se chová trochu jinak. Ta nejmenší tráva (používající blending) má mnohem bližší horizont objevování, které se opět provádí jako postupná změna průhlednosti. Větší trsy nejen, že mají velmi daleký horizont, ale také používají techniku LOD. Oproti Serious Sam má však viditelně víc stupňů složitosti modelu a k jeho změně dochází skokově.

Téměř všechna vegetace se kymácí ve větru. Velikost tohoto pohybu se různě liší. Po chvíli bloudění je možné narazit na nehybné rostliny, ale těch je jen málo. Rostliny nevrhají stíny na zem. Jsou světlé pokud jsou umístěny na přímém slunci a pokud jsou ve stínu hory či stromů, tak jsou přiměřeně tmavé.

Obrázek 4: Ukázka ze hry Far Cry

2.2.2 Celkový dojem

Far cry se už o realistické zpracování vegetace snaží, na druhou stranu je pravda, že výběr prostředí tropické džungle opět nahrává jasným barvám a používání převážně zelené barvy. Ve výsledku však tato barevnost není přehnaná. Díky pečlivosti tvůrců při tvorbě prostředí působí vegetace velmi

přirozeně. Textury mají vyváženou kvalitu, kde se vedle sebe nenachází příliš detailní a málo detailní povrch. Rostliny sice nevrhají stíny na zem, ale velmi efektně stíny dopadají na zbraň.

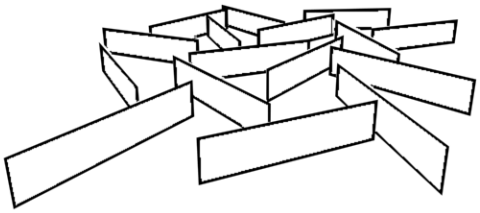
Celkově, i když má hra k dnešnímu datu již tři roky, je zde vegetace velmi dobře zpracována a vezme-li se v úvahu relativní nenáročnost na výkon počítače, tak se bude konkurence hledat jen těžko.

2.3 Battlefield 2

O rok později za Far cry v létě 2005 vyšlo pokračování čím dál tím populárnější série Battlefield. Oproti předchozím dvěma hrám se tato soustředí čistě na multiplayer. Boje se odehrávají v současnosti s moderními zbraněmi na rozlehlých bojištích jejichž velikost dosahuje několika kilometrů čtverečních. V této hře již není na vegetaci kladen takový důraz, ale stojí za to, se na ni podívat.

2.3.1 Zpracování vegetace

Vegetace v Battlefield 2 [4] se podobně jako v Serious Sam dělí na dvě skupiny. První je velmi nízká tráva dosahující výšky maximálně několika málo desítek centimetrů. A druhá skupina jsou keře. Každá skupina má poněkud jiné vlastnosti.



Obrázek 5: Rozmístění polygonů trávy

Malá tráva je tvořena náhodně rozmístěnými pásy, jak je ukázáno na obrázku 5. Tyto pásy jsou viditelné jen z jedné strany (back face culling), čehož si lze všimnout až teprve po detailním zkoumání. I když jsou pásy rozmístěny s náhodným natočením, tak kopírují terén v případě nerovností a jsou jen v předem daných oblastech. Ze všech uváděných her se jako jediná tato tráva objevuje tak, že v podstatě vyrůstá ze země. Děje se tak na samotné hranici viditelnosti, což jde zpozorovat jen při vyšších rozlišení nebo při nastavení minimálních detailů.

Jistý pokrok lze vysledovat i u osvětlení a stínování, které by se dalo rozdělit na statické a dynamické. Malá tráva sice nevrhá stíny, ale přijímá stíny stromů, budov, kamenů a hor. Tyto stíny jsou plynule rozmazané. Ostré stíny vrhají převážně postavy a dopravní prostředky.

Větší keře se od malé trávy liší převážně tím, že se opět jedná o modely s malým počtem trojúhelníků. Keře tentokrát vrhají měkké stíny, které dopadají i na malou trávu. Zvláštností je, že

tento typ vegetace ani ve velkých vzdálenostech nemizí. Ve chvíli, kdy se vykresluje země, tak se vykreslují i keře. Přirozeně se zde používá LOD, kde jsou přechody mezi jednotlivými úrovněmi složitosti modelu proloženy změnou průhledností.

Společnými vlastnostmi veškeré vegetace v Battlefield 2 je pak kymácení ve větru a použití alfa testu. Ve hře se blending používá jen na přechody mezi modely LODu.

Obrázek 6: Ukázka ze hry Battlefield 2

2.3.2 Celkový dojem

Jak bylo zmíněno v úvodu, Battlefield 2 se primárně na zobrazování vegetace nesoustředí. Kvůli méně exotickému prostředí bylo potřeba vytvořit téměř nenápadnou složku celé grafiky. Zobrazování velmi malé trávy (takové, jakou běžně vídáme v našem okolí) je zde vyřešeno kombinací vhodné textury podkladu a rozmístěním průhledných textur tak, že díky správně zvolené hustotě v podstatě vidíme jednotlivá stébla trávy. Při zalehnutí pak vegetace vypadá velmi realisticky.

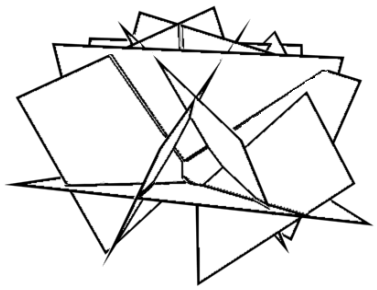
Bohužel u větších keřů lze objevit nedostatečně detailní texturu. Listy či větvičky jsou příliš tlusté a rozmazané, což trochu kazí jinak velmi dobrý dojem z grafiky hry. Celkově má Battlefield 2 velmi realistické zobrazování vegetace a v této práci se budou používat podobné postupy, jaké byly vypozorovány právě zde.

2.4 Oblivion

Poslední hra vybraná pro studii zobrazování vegetace už není FPS, ale RPG (Role-playing game – fantasy hra na hrdiny). Vyšla v roce 2006 a popis herního systému by byl nad rámec této práce. Pro čtenáře bude stačit informace, že se odehrává ve středověkém prostředí plné hradů, jeskyň a hlavně volné přírody. Oblivion opět patří ke hrám, které se prezentovaly jako grafický zázrak s pokročilým zobrazováním vegetace.

2.4.1 Zpracování vegetace

Oblivion [5] k zobrazování vegetace přistupuje velmi odlišným způsoben. I když existují vetší keře, tak ty mají spíše vlastnosti jakými jsou tvořeny koruny stromů – skupina stejných textur s průhledností, které se stále otáčí do pohledu (billboarding). Samotná tráva je tvořena jinak, viz obrázek 7.



Obrázek 7: Model trsu trávy

Takovéto trsy jsou vždy po skupinkách stejného druhu rozmístěny po krajině. Jejich výška zpravidla dosahuje ke kolenům postav, ale je možné narazit na mnohem nižší hloučky rostlin. Samozřejmostí je pohyb ve větru.

Trsy se objevují v závislosti na nastavené vzdálenosti. Toto objevování funguje jako LOD, který má velmi krátkou vzdálenost mezi jednotlivými modely. Z počátku je trs tvořen jen několika trojúhelníky, ale stačí krok vpřed a během toho krátkého okamžiku viditelně postupně přibude do plného modelu. Pokud si pozorovatel stoupne na správné místo, lze vidět, jak trojúhelníky přibývají a ubývají vlivem pohybu ve větru. Podle videí dostupných na internetu by se ale tráva měla objevovat podobně jako u Far cry – postupnou změnou průhlednosti. Těžko odhadovat, proč tomu tak není.

Trsy trávy na zem stíny nevrhají, ale přijímají stíny, které vrhají velké stromy. Tyto stíny jsou měkké a pohybují se, což je vidět nejen na zemi ale i na trsech trávy. Zde už se také blending nepoužívá, průhlednost je řešena alfa testem.

Obrázek 8: Ukázka ze hry Oblivion

2.4.2 Celkový dojem

Nízká vegetace v Oblivion má své nepopiratelné kouzlo. Hloučky trávy vypadají velmi efektně, ale je možné si velmi brzy všimnout, jak moc se modely opakují. V jejich struktuře není žádná obměna, takže i přes změnu textury vypadá v některých chvílích většina trávy až rušivě stejně.

Je ale potřeba dodat, že díky pohyblivým stínům korun stromů, pohybu ve větru, hustotě rozsazení a efektu rozmazaného obrazu (bloom efekt) zapadá tráva do celkové grafiky, aniž by výrazně vynikala, nebo naopak zaostávala.

2.5 Souhrn vypozorovaných vlastností vegetace ve hrách

Jako první je potřeba si všimnout jakým způsobem je samotná vegetace vytvořena. V žádné zde popsané hře není tráva vymodelována detailně stéblo po stéblu, keře nejsou modely s trojúhelníky použitými na každou větvičku a lísteček. Pokud by tomu tak bylo, tak by se tento postup nazýval Geometry based rendering (GRB - zobrazování založené na geometrii). Tímto by bylo potřeba čelit

dvěma problémům. Prvním je náročnost při tvorbě takto složité geometrie a druhým je výpočetní náročnost při vykreslování.

V konečném výsledku je ve hrách používána kombinace GBR a Image based rendering (zobrazování založené na obrázcích). Modely nejsou složité a velmi se využívá průhledných textur, které zastoupí složitější geometrii, protože u listů trávy příliš nevadí nulová tloušťka.

U průhledných textur se používá blending a alfa test. Další vypozorovanou vlastností bylo mizení trávy za určitou hranicí. Je potřeba si uvědomit, že zobrazování vegetace ubírá určitou část výkonu celého systému. Není momentálně v silách běžně dostupného hardwaru vykreslovat veškerou vegetaci nebo třeba jen tolik, kolik je vykreslována samotná krajina. Ve většině případů vegetace mizí dřív než viditelný horizont. Samotný přechod je možné buď udělat prostým zmizením, což v menší blízkosti nevypadá uspokojivě. Dalším způsobem je postupná změna průhlednosti, která je při správném vyvážení vzdálenosti a odpovídající barvě pozadí velmi nenápadná. Posledním způsobem je vyrůstání ze země, což vypadá také velmi dobře.

S mizením vegetace souvisí i její rozmístění. Na základě vypozorovaných vlastností trávy lze odhadovat, že určení, kde se má flora vyskytovat, lze provádět dvěma způsoby. Velmi pracný je způsob první. Návrhář prostředí hry určí umístění, velikost a druh trávy ručně model po modelu. Pokud je takto nutné vytvořit velké oblasti, tak je to práce na několik dní a možná i pro větší počet lidí. Druhým způsobem je určení větší oblasti, druhu vegetace, jaká na ní bude růst a dalších doplňujících informací (například hustota osazení). Na základě těchto dat si vše potřebné vygeneruje sám počítač. Tento způsob je jednodušší, ale uspokojivé výsledky nejsou tak jisté.

Velmi užitečné může být i rozdělení rostlinstva do více druhů. Každý takový druh má jiné vlastnosti, mezi které patří vzdálenost mizení, světelné vlastnosti, chování textur a zda používá LOD.

Pro úsporu výkonu se také používá technika LOD, kdy jsou pro určité vzdálenosti vytvořeny různě složité modely. Většinou se tak děje pro mohutnější vegetaci jako jsou například keře. Tyto modely ale mají už tak málo trojúhelníků, takže počet stupňů složitosti není nikdy moc velký.

Osvětlení vegetace může být taky velmi různorodé. Základem je, aby jasově odpovídala okolí a podkladu. Barvu je možné nastavit jak celému modelu, tak jeho vrcholům nebo pomocí multitextury ovlivňovat jednotlivé pixely.

Poslední vlastností je kymácení ve větru, které mnohem vylepší výsledný visuální dojem a dodá pocit živosti.

3 Návrh řešení

3.1 Zobrazované objekty

I to nejmenší stéblo je potřeba nějakým způsobem vytvořit. Jak už bylo zmíněno v předchozí kapitole, nejlepším způsobem se zdá vytvoření 3D modelu s co nejmenším počtem trojúhelníků, který používá texturu s průhledností.

Pro počítání průhlednosti je možné použít blending nebo alfa test. Používání blendingu má jednu velkou nevýhodu. Kvůli principu, jakým se provádí, je nutné seřadit trojúhelníky podle své hloubky ve scéně od nejvzdálenějšího po nejbližšího. Navíc by bylo nutné přizpůsobit modely, protože by se geometrie mohla křížit. U alfa testu tento někdy výpočetně náročný úkon řazení odpadá a výsledný efekt, při použití dostatečně detailní textury, není o mnoho horší než u blendingu.

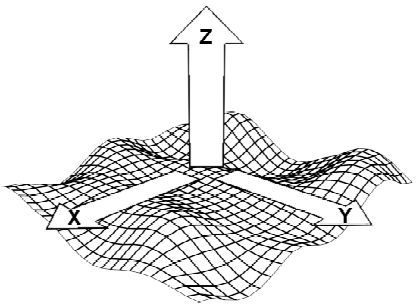
Vytvořený model by měl být natolik členitý, aby při pouhém natočení vypadal jinak. Tímto způsobem je možné omezit počet modelů, které se budou pro zobrazování vegetace používat. Při správném vymodelování budou na zobrazení obyčejné louky stačit dva až tři modely.

Pro snazší správu objektů a menší nároky na grafickou kartu bude výhodné používat jednu texturu na jeden objekt. V případě modelování nejmenší trávy by bylo možné použít jednu texturu na více modelů, ale v tomto případě by nastoupila nutnost složitější správy textur.

3.2 Způsob zobrazování

3.2.1 Krajina pro vegetaci

Základem je předpoklad, že se práce zabývá jednoúrovňovou krajinou. Pokud by se prostor označil tak, že šířka a délka je X a Y, pak výška bude Z (obrázek 9).



Obrázek 9: Orientace os v krajině

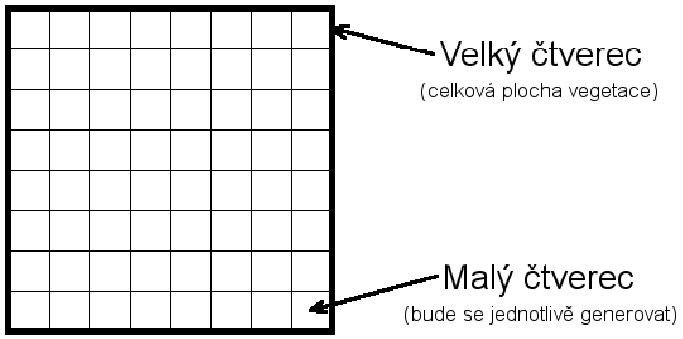
Jednoúrovňovou krajinou se myslí to, že pokud zadáme souřadnice X a Y dostaneme jediné Z. Generováním povrchu krajiny se tento dokument zabývat nebude, bude jen předpokládat, že je k dispozici funkce vracející výšku krajiny na určité pozici.

3.2.2 Uložení v paměti

Vytvořené modely se načtou do paměti, ale ještě nejsou připraveny pro vykreslování. Je potřeba nějakým způsobem určit, kde se budou vykreslovat. Z praktického hlediska bude plocha vykreslované krajiny omezená na čtverec o určité velikosti. Na této ploše se náhodně vygenerují pozice o zadaném počtu kusů. Postupné vykreslování modelů na vygenerované pozici by nebylo právě nejefektivnější. Stejně tak je málo jen pouhé náhodné určení místa. Vegetace by se až nápadně opakovala. Z toho důvodu se ještě bude generovat natočení kolem osy Z, naklonění od osy Z a velikost v určitém rozmezí.

Vzhledem k tomu, že se vegetace z místa moc daleko neposunuje, tak je možné si ji vygenerovat celou do paměti. Takto bude připravena pro vykreslování, které je mnohem rychlejší, než posunutí, natočení a změna měřítka jednotlivých modelů.

Zde však nastupuje problém, pokud se bude jednat o velmi rozlehlé plochy. Jednak paměť počítače nemusí stačit, a také by se určitě zobrazovala část vegetace zbytečně. Je potřeba použít alespoň nějakou základní správu paměti. Jako ideální se jeví rozdělení zadané plochy čtverce na mřížku menších čtverců. Tyto menší čtverce by se generovaly (případně mazaly) a zobrazovaly podle potřeby (obrázek 10).



Obrázek 10: Rozdělení prostoru vegetace

Výše popsaný postup generování je pak možné aplikovat na různé modely, kterým se určí rozdílné vlastnosti. Takovými hlavními vlastnostmi, které se nejvíce projeví na výsledku jsou rozmezí náhodné velikosti a počet kusů daného druhu ve čtverci.

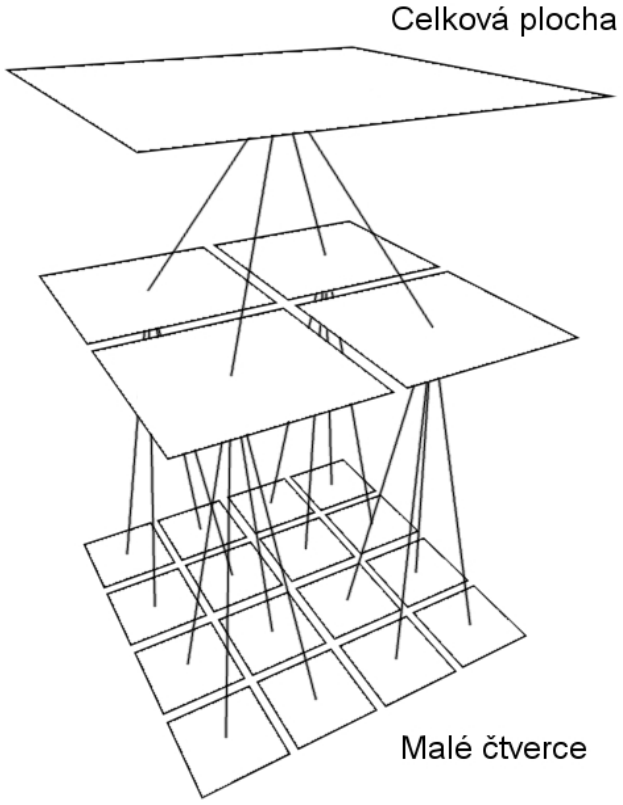
3.3 Metody zvyšující rychlost vykreslování

3.3.1 Ořezání komolým kuželem pohledu pomocí quadtree

Jedná se o základní metodu, která omezuje množství dat, které jsou posílány na grafickou kartu a tím zvyšuje rychlost vykreslování. Princip je prostý – na základě pohledu se vypočte šest ploch, které ohraničují oblast v prostoru [6]. Zda bod leží v pohledu se zjišťuje dosazením jeho pozice do obecné rovnice všech rovin. Pokud u všech vyjde kladné číslo, leží bod uvnitř (záleží na orientaci normál ploch). Kromě bodu se ještě velmi snadno zjišťuje, zda v pohledu leží koule, kterou je možné obalit celé objekty.

Objekty nacházející se v tomto prostoru budou vykresleny. Objekty mimo něj je možné vypustit. V případě této práce se budou testovat koule obalující jednotlivé uzly quadtree [7].

Quadtree je struktura znázorněná na obrázku 11. Pomocí ní je možné ve většině případů snížit množství prováděných kontrol, zda obalová koule leží v pohledu.



Obrázek 11: Princip quadtree

3.3.2 Určení hustoty výskytu vegetace

Tráva přirozeně neroste na celé zadané ploše rovnoměrně. Proto je potřeba nějakým způsobem určit hustotu růstu. Pokud se navíc objeví místa, kde neroste vůbec, naskytne se příležitost k úsporám při vykreslování. Nejideálnějším způsobem se jeví použití šedotónové bitmapy. Černá barva by znamenala žádný výskyt vegetace a bílá maximální zadaný počet. Pokud by se objevil malý čtverec, na kterém se při generování nevytvoří žádný model, pak se to u něj poznačí a vůbec se nevysílá k vykreslení.

3.4 Dodatečné efekty

Mezi dodatečné efekty by se dalo počítat kymácení ve větru a objevování vegetace v dálce. Toto jsou úkony, pro které je možné využít programovatelné procesory na grafických kartách (shadery [8]).

3.4.1 Pohyb ve větru

Jemné pohupování rostlin lze vytvořit pohybem vrcholů trojúhelníků. Pro tento druh pohybu je nejideálnější sinusová funkce, díky které se vrcholy budou přesunovat realisticky. Čím výš je vrchol od země, tím delší dráhu projede.

3.4.2 Mizení a objevování v dálce

Visuálně je přijatelnější, aby za určitou hranicí tráva mizela. Kdyby tomu tak nebylo, bylo by možné vidět, jak je tráva uspořádána do nepřirozených čtverců. Zjištění vzdálenosti vrcholu od pozice pohledu je v shaderech velmi jednoduché. Na základě této hodnoty je možné použít jak změnu průhlednosti, tak vyrůstání ze země.

V případě změny průhlednosti by se ovlivňovala alfa složka barev a v případě vyrůstání by se měnilo umístění vrcholu v ose Z.

3.4.3 Míchání barev textur

Vložený model nemusí vždy barevně ladit s prostředím nebo s podkladem, proto by bylo dobré umožnit nějaký způsob ovlivňování jeho barvy. Řešením může být další barevná textura, která se smíchá s barvou modelu. V dnešní době toto není problém s použitím multitexturingu.

4 Implementace

4.1 Nástroje

Práce na projektu by se dala rozdělit na dvě části. Tou menší je tvorba modelů vegetace, což je spíše kreativní činnost než programátorská. Pro vytváření modelů se používala aplikace Blender [9] a úpravu textur program Gimp [10].

Druhá část se zabývá implementací nástroje pro zobrazování vegetace. Program byl vyvíjen v prostředí Visual Studia 2005 za použití knihoven OpenGL zobrazující 3D grafiku, GLUT [11] starající se o komunikaci se systémem, GLEW [12] pro načítání extenzí a glpng [13] načítající obrázky ve formátu PNG. Jako programovací jazyk je použito čisté C a GLSL [14].

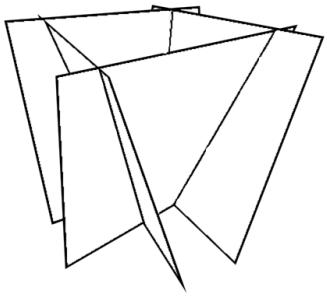
4.2 Tvorba modelu

V předchozích kapitolách bylo naznačeno, jak by měl takový model vegetace vypadat a kolik druhů je možné vytvořit. Jako ukázkové příklady zde budou uvedeny postupy tvorby u většího trsu trávy a použití fotografie pro vznik keře.

4.2.1 Kompletně uměle vytvořený model

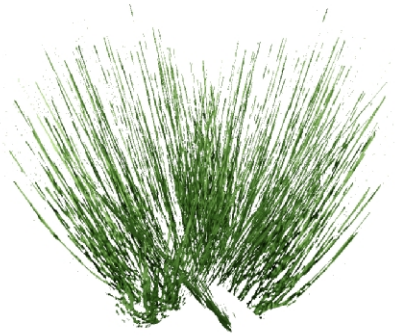
V Blenderu se vytvořil objekt, na který se použil částicový systém. Při správném nastavení je možné vytvořit stejně tlusté a dlouhé čáry náhodně protažené do prostoru v určitém směru. Obarvením těchto čar zelenou barvou vznikne trs trávy složený z několika stovek (podle zadaného parametru) stébel.

Takovýto trs se vyrenderoval do obrázku. Aby byla dosažena větší variabilita modelu a využita větší plocha textury, vytvořily se dva pohledy na trs. Tyto pohledy se v Gimpu vložily do jediné textury nad sebou. Jako formát pro textury byl použit PNG. Ten umožňuje ukládat alfa složku potřebnou pro určení průhlednosti. Posledním krokem zbývá vytvoření samotného modelu. Pro dobrý visuální dojem a co největší omezení počtu trojúhelníků se trs trávy vytvořil tak, jak je naznačeno na obrázku 12.



Obrázek 12: Model trsu trávy

Na jednotlivé čtyřúhelníky se nanesla textura (na každý polovina textury) a složily se dohromady. Výsledek lze vidět na obrázku 13.



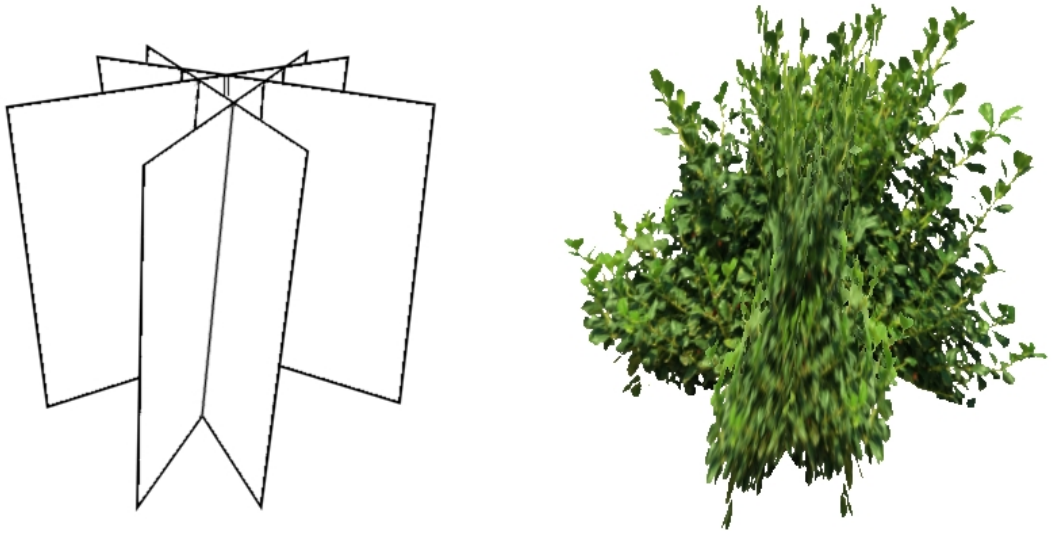
Obrázek 13: Otexturovaný model trsu trávy

4.2.2 Model vytvořený s použitím skutečné fotografie

Dalším způsobem vytvoření modelu bylo použití fotografie upravené jako textury. Na internetu je možné naleznout různé obrázky použitelné pro tuto práci. Příkladem může být například keř jako je na obrázku 14.

Obrázek 14: Fotografe keře použitá jako textura

Samotná textura je dostatečně složitá na to, aby stačilo vytvořit model, jako je na obrázku 15.



Obrázek 15: Příklad modelu keře

4.2.3 Barevná korekce

Ve chvíli, kdy bylo vytvořeno více modelů a došlo k jejich vykreslení v samotném programu, vyšlo najevo, že k sobě barevně neladí. Toto bylo potřeba na úrovni textur upravit v Gimpu. Obrázky s texturou se vedle sebe otevřely a podle visuálního dojmu se nastavily odpovídající tóny barev. Přitom bylo potřeba dávat pozor, aby barvy zůstaly stále přirozené.

4.3 Načítání modelu

4.3.1 Uložení v souboru

Když je model hotov, vyexportuje se ve formátu ASE do souboru. Tento formát byl vybrán ze dvou důvodů. Prvním důvodem je jeho snadná čitelnost a druhým je, že obsahuje všechny důležité informace pohromadě:

- Jméno souboru s texturou (formát png)
- Seznam vrcholů s jejich pozicemi v prostoru
- Indexy do seznamu vrcholů z nichž se složí trojúhelník
- Seznam mapovacích souřadnic
- Indexy do seznamu mapovacích souřadnic pro každý vrchol trojúhelníku

Načítá se jen první geometrie, případná další se ignoruje. Pokud některé z dat chybí, tak se přeskočí.

4.3.2 Uložení v paměti

Výše popsaný formát se převádí tak, aby se dalo využít vykreslování pomocí pole vrcholů (vertex array). Jeden vrchol má následující tvar:

- Mapovací souřadnice U,V
- Souřadnice polohy X,Y,Z,W

Z hodnot načtených ze souboru se zatím nevyužívají normály. Oproti tomu se využívá čtvrtá souřadnice W, což není obvyklé. Proč tomu tak je, bude vysvětleno v kapitole 4.9.1. Tato struktura je uspořádána do pole, kde jednotlivé její trojice znamenají trojúhelník.

4.4 Prototypy vegetace

Prototyp je pracovní označení pro model s určitými vlastnostmi, podle kterých se bude chovat celý jeden druh vegetace. Mimo model obsahuje odkazy do pole bitmap hustoty. Každý prototyp jich může mít libovolný počet.

4.4.1 Vlastnosti

Vlastnosti prototypu se dělí do dvou skupin. První se použije ve chvíli, kdy dochází ke generování vegetace na základě modelu prototypu (kapitola 4.7). Druhá se zadává pokaždé, když dochází k vykreslení vegetace určitého prototypu.

Vlastnosti použité při generování

- Počet kusů na malém čtverci
- Maximální naklonění (od osy Z)
- Maximální natočení (kolem osy Z)
- Nejmenší velikost
- Největší velikost

Vlastnosti použité při vykreslování

- Vzdálenost hranice mizení/objevování
- Šířka kývání
- Rychlost kývání
- Mlha (zapnuta/vypnuta)
- Mixovaní barev s globální texturou (zapnuto/vypnuto)

4.4.2 Uložení v paměti

Prototyp je reprezentován strukturou obsahující výše zmíněné hodnoty. Mimo to ještě uchovává název prototypu, jméno souboru s modelem a jméno obrázku s texturou. Tyto řetězce se dále používají pro úsporu na paměti.

Jednotlivé prototypy se rozlišují podle řetězce s názvem. Samotná struktura se vytvoří ve chvíli, kdy se zavolá libovolná funkce pro nastavení vlastností určitého prototypu.

Při přidávání nového modelu se zjišťuje, zda již nebyl před tím načítán. Pokud ano, použijí se načtené data. Pokud model načítán nebyl, je ještě šance, že používá již existující texturu (v tomto případě se jedná o významné ušetření paměti). Aby docházelo ke korektnímu uvolňování paměti, nastavují se příznaky značící, že se odkazující data nemají mazat.

4.5 Hustota vegetace

Hustota vegetace se uchovává podobně jako u prototypu – struktura identifikovatelná podle řetězce se jménem. Obsahuje následující informace:

- Jméno souboru s obrázkem
- Rozlišení bitmapy
- Velikost
- Pozice středu
- Bitmapa

Data s hustotou se dají vložit dvěma způsoby. První je načtení obrázku. Pokud se jedná o barevný obrázek, dojde k celočíselnému přepočtu na šedotónový obrázek podle vzorce (1). Y značí šeď (gra**y**), R červenou barvu (**r**ed), G zelenou barvu (**g**reen) a B modrou barvu (**b**lue).

$$Y = \frac{R*3}{10} + \frac{G*59}{100} + \frac{B*11}{100}1 \tag{1}$$

Druhým způsobem je přímé vložení dat. Jeden pixel je uložen v unsigned char, což znamená hodnoty od 0 do 255. Je třeba dodat, že se používá výhradně čtverec, takže pokud se načítá obrázek, jako rozlišení se použije jeho menší rozměr. V případě vkládání vlastních dat je potřeba dávat pozor, aby zadané rozlišení odpovídalo a nedošlo ke špatnému načtení, popřípadě snaze číst v paměti, která není alokována.

4.6 Celková datová struktura vegetace

Základní části, ze kterých se skládá vegetace, byly v minulých kapitolách vysvětleny. Nyní nastala chvíle pro popsání celkové struktury používající se pro vykreslování. V kapitole 3.2.2 byl navržen způsob, zde se přiblíží jeho implementace.

4.6.1 Quadtree

Celková čtvercová plocha vegetace je pravidelně rozdělena na menší čtverce. Ty jsou uloženy do jednorozměrného pole ukazatelů na strukturu obsahující vygenerované modely. I když je pole jednorozměrné, používá se jako dvourozměrné. K přepočtu souřadnic dochází podle vzorce (2). Zpočátku jsou ukazatele prázdné, ale podle potřeby se postupně vyplňují vegetací.

$$I = Y * velikost\ strany + X \tag{2}$$

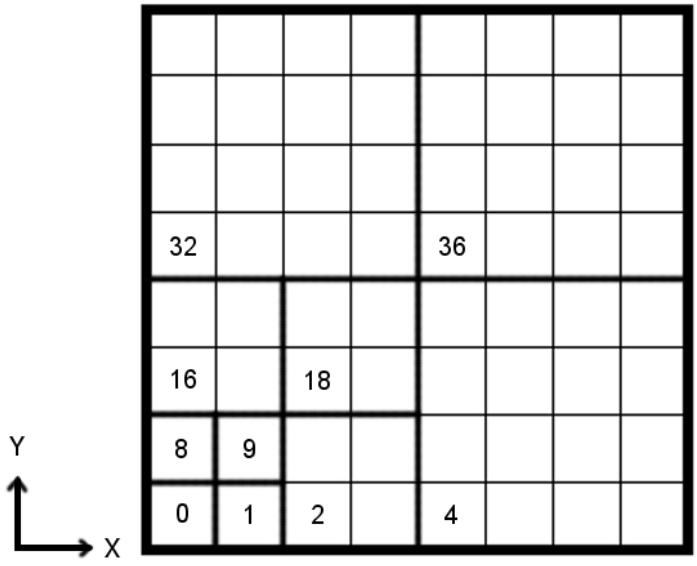
Aby se rychleji vyhledávaly čtverce ležící v pohledu, vytvořila se nad polem struktura quadtree. Ta se jak rekurzivně vytváří, tak i maže. Quadtree se při každé změně pozice a velikosti plochy vegetace nebo hloubky zanoření přepočítává.

Každá větev stromu v sobě uchovává tyto informace:

- Index do pole s malými čtverci
- Pozice středu obalové koule
- Poloměr obalové koule

Jak z tohoto výčtu vyplývá, pro zjišťování výskytu v pohledu se používá obalová koule. Tato obalová koule má průměr o délce úhlopříčky čtverce, kterého obaluje. Vždy větve o jeden stupeň hlouběji mají poloviční průměr.

Při rekurzivním vytváření se pro každou větev určí index tak, aby v nejhlubším nastaveném zanoření odpovídalo umístění malého čtverce, na který ukazuje. Obrázek 16 naznačuje indexování v případě zanoření s hodnotou 4.

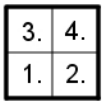


Obrázek 16: Indexování malých čtverců při tvorbě quadtree

Strana velkého čtverce má vždy $2^{hloubka-1}$ malých čtverců. Celé indexování a vykreslování počítá s natočením podle os, stejně jako lze vidět na obrázku. Indexy následující čtveřice uzlů quadtree se vypočtou následovně:

1. $uzel = index$
2. $uzel = index + 2^{hloubka-1}$
3. $uzel = index + strana * 2^{hloubka-1}$
4. $uzel = index + strana * 2^{hloubka-1} + 2^{hloubka-1}$

Proměnná ***index*** značí aktuální index uzlu, proměnná ***strana*** značí celkový počet malých čtverců na stranu velkého čtverce a proměnná ***hloubka*** značí aktuální zanoření uzlu. Pořadí uzlů je určeno stejně, jako v případě, kdy by šlo o pole 2x2 (obrázek 17).



Obrázek 17: Pořadí indexů na ploše

4.6.2 Globální vlastnosti

Nyní je potřeba shrnout zbývající vlastnosti:

- Velikost velkého čtverce
- Střed velkého čtverce
- Hloubka zanoření quadtree (ovlivňuje počet malých čtverců)
- Velikost čtverce se středem v pohledu, podle které se generuje
- Počet najednou vygenerovaných malých čtverců
- Ukazatel na funkci vracející výšku na určité pozici
- Cesta k modelům
- Cesta k texturám
- Cesta k shaderům
- Maximální velikost struktury

Toto jsou vlastnosti, které lze pomocí funkcí ovlivnit a jsou vesměs povinné (bez řádného nastavení by se nemuselo nic vykreslit). Další vlastností je globální obarvení. To je možné zadat jednak ze souboru s texturou nebo přímo indexem textury OpenGL. Obarvení má jen tyto vlastnosti:

- Velikost čtverce textury
- Pozice středu textury

4.6.3 Vnitřní pomocné proměnné

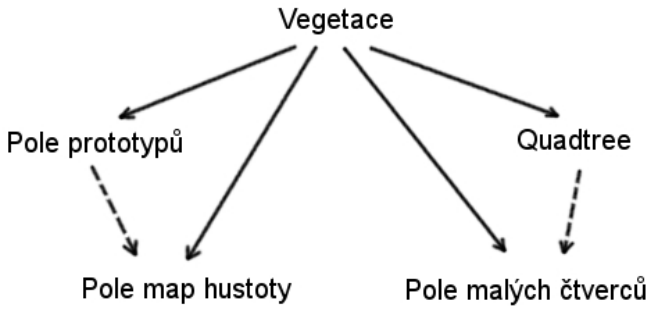
V průběhu vývoje vyvstala potřeba si některé informace ukládat. Mezi ty základní patří odkazy na proměnné v programu shaderů. Ty se při vytvoření struktury vegetace nastaví a už se jen používají k nastavování hodnot popsané v kapitole 4.4.1.

Další informační hodnoty jsou množství paměti, které struktura aktuálně zabírá a počet naposledy vykreslených trojúhelníků. Zatímco první se aktivně používá při promazávání paměti, druhá je opravdu spíše informačního charakteru.

Velmi důležité jsou proměnné značně urychlující vykreslování. Jedná se o pole s odkazy na malé čtverce, které se mají vykreslit (vysvětleno v kapitole 4.7.1) a jejich vzdáleností od pohledu. Další pole uchovává indexy na malé čtverce, které jsou prázdné a měly by se co nejdříve vygenerovat. Jako poslední je odkaz na jeden jediný čtverec, který neobsahuje žádnou vegetaci (kapitola 4.7.3).

4.6.4 Struktura

Jak jsou celkově struktury používány a uloženy je znázorněno na obrázku 18.



Obrázek 18: Vazby struktur v programu

Vegetace v sobě odděleně uchovává všechny čtyři části. Prototypy, mapy hustoty a malé čtverce jsou uloženy v polích, zatím co quadtree je stromová struktura (což ostatně plyne z názvu). Prototypy pomocí indexů odkazují na mapy hustoty, stejně tak quadtree do pole malých čtverců. Zatímco quadtree má indexy určeny pokaždé, prototyp nemusí mít ani jeden.

4.7 Generování vegetace k vykreslení

Generování patří k nejsložitější operaci v projektu. Jedná se v podstatě o kopírování modelu prototypu do určité oblasti, kdy dochází k jeho úpravám (umístění, natočení, velikost atd.).

4.7.1 Příprava

Aby se nevytvářely malé čtverce zbytečně tam, kde vlastně ani nejsou vidět, generují se jen ty, které jsou v obraze. V praxi to znamená, že se ve vykreslovací části, kde dochází ke zjišťování malých čtverců v pohledu (kapitola 4.6.1), zároveň zjišťuje, zda už má vytvořenou vegetaci. Pokud nemá, uloží se jeho index do pole. Toto pole se pak používá při určování, co vygenerovat.

Kolik malých čtverců se najednou vytvoří, ovlivňuje nastavitelná hodnota. Rovněž je třeba dodat, že se ani neberou v úvahu všechny uložené v poli pro vykreslování. Generují se jen ty ve čtverci kolem pozice pohledu (jehož velikost je rovněž nastavitelná), takže ty příliš vzdálené počkají na vhodnější příležitost.

Poslední částí přípravy je alokace paměti. Vzhledem k tomu, že v té chvíli ještě není určeno, kolik z nastaveného počtu modelů v malém čtverci bude skutečně vygenerováno, tak se zabere maximální možná velikost paměti. Toto řešení má nevýhodu v tom, že čím je více prototypů, tím je větší pravděpodobnost zbytečně zabrané paměti.

4.7.2 Generování

Generování probíhá opakovaně pro každý prototyp vegetace. Výsledek by se dal popsat jako vrstvy v paměti vytvořené z uložených modelů. Generování pseudonáhodných čísel obstarává standardní knihovní funkce jazyka C. Pro každý jeden model se provede následující:

1. Náhodné určení umístění v oblasti malého čtverce
2. Zjištění hodnoty hustoty v umístění
3. Náhodné vygenerování hodnoty od 0 do 254

Na tomto místě dojde k porovnání hodnoty hustoty a náhodně vygenerovaného čísla. Pokud je náhodné číslo menší, model se vytvoří. Generování náhodných hodnot je nastaveno takovým způsobem, aby při zadání hustoty 0 opravdu nic nerostlo a při zadání hodnoty 255 naopak vždy došlo k vytvoření nastaveného maximálního množství modelů.

Každý prototyp může mít libovolný počet map hustoty (kapitola 4.5). Při zjišťování hodnoty na určité pozici se mapy postupně prochází a výsledná hodnota se vypočítá z první nalezené, do které souřadnice zapadá. Protože se při určování map hustoty do pole přidávají indexy postupně od počátku, nejvyšší prioritu má nejdříve určená mapa hustoty. Jestliže souřadnice leží mimo všechny mapy, použije se nastavená hodnota. Tímto se může za použití malé mapy hustoty s malým rozlišením vytvořit osamocená oblast s vegetací uprostřed rozlehlé krajiny, kde jinak nic neroste.

Tvorba modelu dále pokračuje následovně:

1. Zjistí se výška umístění
2. Náhodné vygenerování natočení
3. Náhodné vygenerování naklonění
4. Náhodné vygenerování velikosti

5. Zkopírování modelu do předem alokované paměti

Pro každý vrchol trojúhelníku zkopírovaného modelu se provede:

1. Naklonění kolem osy X
2. Otočení kolem osy Z
3. Vynásobení velikostí a posun na místo
4. Nastavení 4. souřadnice vrcholu na výšku umístění (vysvětleno v 4.9.1)

K naklonění a otočení dochází kolem souřadnice 0,0,0, protože k posunu na pozici dochází až nakonec. To stejné platí i u změny velikosti modelu.

4.7.3 Paměť

Ve chvíli, kdy se používají mapy hustoty, může nastat situace, kdy na některém malém čtverci nevznikne ani jeden model vegetace. Tento stav se využívá pro ušetření paměti a také pro zrychlení vykreslování (čtverec není nutné vůbec vykreslovat). Na začátku generování se nastaví příznak značící, že se zatím žádný model nevytvořil. Pokud to tak zůstane do konce generování, nastaví se příznak nevykreslování a uloží se index malého čtverce. Při dalším generování nedochází k alokaci, ale použití paměti předchozího čtverce.

Aby se zbytečně nevykreslovaly modely, které nejsou vygenerovány, při kopírování se počítají, v paměti skládají za sebe a nakonec se vykreslí jen jejich správný počet.

4.8 Vykreslování

Vykreslování by se dalo rozdělit na dvě části – příprava a poslání dat do grafické karty. Tyto části neprobíhají po sobě, ale příprava probíhá uvnitř posílání dat.

4.8.1 Příprava

Než začne vykreslování trojúhelníků, nejdříve se zjistí, co má vlastně smysl vykreslovat. Rekurzivně se prochází quadtree a zkouší se, zda obalová koule zasahuje do pohledu. Pokud narazí na poslední větev (ukazatelé neukazují na další větev) a koule stále zasahuje do pohledu, tak se provede kontrola, zda je malý čtverec vygenerovaný. Pokud ano:

1. Poznačí se, že malý čtverec se nesmí smazat (kapitola 4.10)
2. Do pole se uloží index malého čtverce k vykreslení
3. Vypočte se vzdálenost od pohledu do středu obalové koule a hodnota se uloží

Pokud čtverec není vygenerován, pak:

1. Do pole se uloží index malého čtverce k vygenerování

Takto jsou připravena 3 různá pole. První dvě se použijí při vykreslování a poslední, když dojde na generování malých čtverců.

4.8.2 Poslání dat do grafické karty

Pro vykreslování vegetace se používá pole vektorů (vertex arrays). Existuje několik způsobů, jak data do grafické karty pomocí vertex arrays poslat. V tomto projektu mají podobu jednotlivých trojúhelníků, kdy pro vykreslení dvou je potřeba zaslat popis 6 vrcholů (existují varianty zřetězených trojúhelníků, kde pro dva stačí zadané 4 vrcholy, ale pro modely vegetace se příliš nehodí). Už při generování se data vhodně skládají tak, aby se jednou funkcí poslala k vykreslení.

Celkový postup při vykreslování je následující:

1. Zapnutí vertex arrays
2. Zapnutí shader programu
3. Výpočet kuželu pohledu (frustum)
4. Uložení aktuální pozice pohledu
5. Nastavení globálních proměnných v shaderu (pozice, globální mapa)
6. Provedení přípravy (předchozí kapitola 4.8.1)
7. Zapnutí multitexturingu

Aby se zbytečně nepřepínaly textury, následuje cyklus probíhající všechny prototypy a provádějící toto:

1. Nastavení vlastností vegetace v shaderu (kapitola 4.4.1.2)
2. Přiřazení textury

V této chvíli se v dalším cyklu prochází všechny malé čtverce připravené v poli a vykresluje vrstva obsahující modely příslušného prototypu (vrstvy modelů jsou uloženy ve stejném pořadí jako prototypy vegetace). Před zavoláním funkce pro zobrazení se kontrolují dvě věci – zda náhodou má vrstva nulový počet a zda není čtverec za hranicí viditelnosti. V obou případech se žádné data do grafické karty neposílají.

Nakonec následuje už jen:

1. Vypnutí mutlitexturingu
2. Vypnutí shader programu
3. Vypnutí vertex arrays

4.9 Efekty v shaderech

Vně zdrojového kódu psaného v jazyce C stojí program pro grafickou kartu v GLSL. Skládá se ze dvou souborů. První obsahuje vertex shader ovlivňující vrcholy a druhý fragnent shader pracující s pixely.

Název souborů byl napevno určen „veg.vert“ a „veg.frag“. Cestu k nim lze určit při vytváření velkého čtverce s vegetací.

4.9.1 Vertex shader

Ve vertex shaderu se nastavují každou vykreslovací smyčku tyto proměnné:

- Aktuální čas
- Hranice viditelnosti
- Šířka pohybu
- Rychlost pohybu
- Pozice pohledu v prostoru
- Střed mapy obarvení
- Velikost mapy obarvení
- Příznak zapnutí mlhy

Program jako první nastaví informace, které se dále předávají fragment shaderu:

1. Barva vrcholu
2. Informace o mlze
3. Mapovací souřadnice modelu
4. Mapovací souřadnice mapy obarvení

Následuje výpočet pozice vrcholu. Vzorec (3) ukazuje výpočet souřadnice X, což v praxi znamená, že se vegetace vlní jen v jedné ose. Aby se model přirozeně vlnil podle svého umístění, bylo potřeba v souřadnici W uchovat informaci o výšce. Od této hodnoty po počítá, jak moc se má vrchol posunout. Poslední sinus byl přidán až v pozdější fázi vývoje a způsobí, že se vegetace na všech místech nepohybuje stejně. Hodnota 0,05 měla být původně nastavitelná, ale postupným zkoušením se došlo k závěru, že to není potřeba.

$$X = \cos(\check{c}as * rychlost + Y) * \check{s}\acute{i}\check{r}ka * (Z - W) * \sin((\check{c}as + X) * 0{,}05) \quad (3)$$

Po výpočtu nového umístění vrcholu modelu se vypočítá vzdálenost od pohledu. Pokud je vzdálenost větší než hranice viditelnosti, sníží se pozice vrcholu. Napevno je nastaveno snižování o třetinu metru na vzdálenost jednoho metru (metr vysoký keř zmizí tři metry za hranicí viditelnosti).

Aby došlo ke korektnímu výpočtu při násobení maticí pohledu, změní se souřadnice W na hodnotu 1.0. Předposlední akcí je transformace vrcholu, po které se jako poslední získá hloubka vrcholu v obraze pro účely počítání mlhy ve fragment shaderu.

4.9.2 Fragment shader

Ve fragment shaderu docházelo v průběhu vývoje k razantním změnám. Největší změna byla provedena ve chvíli, kdy vyšlo najevo, že základní funkce větvení programu IF se velmi projevuje na výsledném počtu snímků za vteřinu. Proto jsou i příznaky zapnutí/vypnutí mlhy a zapnutí/vypnutí globálního obarvování přímo vloženy do výpočtu v podobě hodnot float (0.0 – vypnuto, 1.0 – zapnuto).

Jediné větvení, které zůstalo, je alfa test odstraňující na základě hodnoty alfa celý pixel. Tímto se fragment shader zmenšil na:

1. Získání barev z textur modelu a globální mapy
2. Alfa test
3. Výpočet barvy pixelu

Hodnota barvy pixelu je určena lineárním promícháním dvou barev. První je barva mlhy a druhá se vypočítá podle vzorce (4).

$$barva = \left(\left(\frac{(barva1 + barva2)}{2,0}\right) * barva\right) * obarvení\ zapnuto + (1,0 - obarvení\ zapnuto) * barva1 * barva \tag{4}$$

Zde následuje vysvětlení proměnných:

- barva – vrchol nastavený pomocí glColor3f
- barva1- globální textura
- barva2- textura modelu
- obarvení zapnuto – buď hodnota 1,0 nebo 0,0

Samotný poměr určující, která z barev převládne je popsán ve vzorci (5).

$$poměr = \frac{((vzdálenost - počátek\ mlhy) * mlha\ zapnuta)}{konec\ mlhy - počátek\ mlhy} \tag{5}$$

Zde již není potřeba vysvětlovat jednotlivé proměnné. Pokud je mlha vypnuta, hodnota 0,0 z celého poměru automaticky udělá nulu (převládne barva textury). Pokud je zapnuta, počítá se poměr vzdáleností. Funkce provádějící mixování se stará o ořezávání hodnoty. Překročení poměru 1 nemá žádný nežádoucí vliv na výsledek, pixel má stále barvu mlhy.

4.10 Správa paměti

Všechny dosavadně popsané postupy jen zabíraly čím dál tím větší paměť. Maximální velikost by byla omezena na základě nastavených vlastností (hlavně počet malých čtverců, prototypů a nastavení počtu modelů). Pro situace, kdy by bylo potřeba používat jen určitou velikost paměti, je nutné zavést její správu.

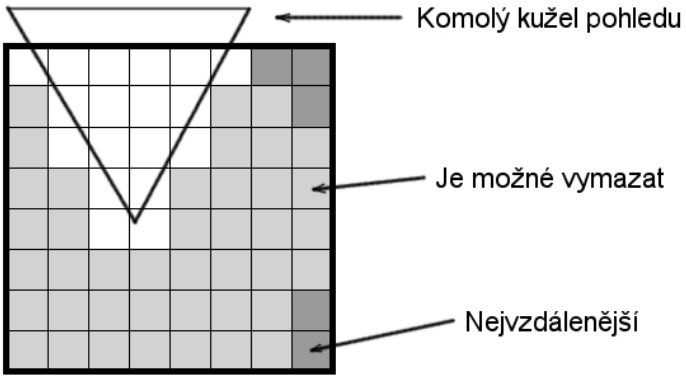
První způsob, jak snížit paměťové nároky, byl zmíněna v kapitole 4.7.3. Ve zkratce se jednalo o používání paměti, která by jinak byla stejně nevyužita.

Další způsob už je poněkud složitější – vymazávají se co nejvíce vzdálené malé čtverce, které nejsou v pohledu. Promazávání probíhá následovně:

1. Zjistí se, zda vegetace zabírá více, než je zadaná hodnota
2. Určí se počet malých čtverců k vymazání (dvojnásobek těch, co jsou navíc)
3. Vytvoří se pole s odkazy a vzdálenostmi o velikosti počtu malých čtverců k vymazání

Následuje procházení všech malých čtverců. Již při fázi přípravy dat k vykreslování (kapitola 4.8.1) se poznačily čtverce, které se mazat nemají, protože jsou v pohledu. Pokud tedy malé čtverce zabírají paměť a mohou být vymazány, uloží se jejich index a vypočtená vzdálenost od středu pohledu. Takto se naplní obě pole.

Ve chvíli, kdy jsou naplněna dochází k hledání indexu, kterému odpovídá nejmenší vzdálenost a na jeho místo se umístí nově nalezený malý čtverec. Během procházení se u všech nastaví příznak, že mohou být vymazány. Při dalším vykreslování se opět vypnou ty, které jsou v pohledu. Když to vše skončí, v polích zůstanou nejvzdálenější čtverce, které nejsou v pohledu. Názorný příklad lze vidět na obrázku 19.



Obrázek 19: Mazání přebytečných malých čtverců

Tato metoda má viditelně nevýhodu v tom, že jako nejvzdálenější se může určit malý čtverec, který je těsně vedle pohledu. Tím by teoreticky docházelo ke zbytečnému mazání a opětovnému generování, protože otáčení pohledem je daleko častější než jakýkoliv jiný posun s pohledem.

5 Výsledek

Výsledkem práce se staly zdrojové kódy, které je možné vložit do programu zobrazující krajinu a jednoduchým způsobem na ni vysázet vegetaci.

5.1 Systém používání vegetace

Jako základ pro používání vegetace je struktura *BigVegQuad*. Nejdříve se vytvoří ukazatel na tuto strukturu a poté se předá funkci jako parametr, která ji alokuje a nastaví základní vlastnosti (většinou jde o nulové hodnoty). V každé další funkci nastavující libovolný parametr se tento ukazatel předává jako identifikátor, o kterou vegetaci se jedná. Z toho vyplývá, že není problém mít vytvořených více velkých čtverců s vegetací a libovolně je používat.

Identifikace prototypu se provádí pomocí řetězce s názvem (pole *char*ů). Při nastavování vlastnosti se funkci jako první parametr předává ukazatel na vegetaci, jako druhý jméno prototypu a konečně jako poslední požadovaná hodnota. Pokud jméno prototypu bylo použito poprvé, vytvoří se nová struktura a uloží do pole prototypů s vynulovanými hodnotami.

Stejně jako vlastnosti prototypů jsou ovládány mapy hustoty. Jejich identifikace je taktéž podle řetězce. Při určování, kterému prototypu patří která hustota, se zadávávají jen jejich jména v řetězcové podobě.

Kvůli složité vnitřní struktuře odkazů a indexů se nedoporučuje mazat data jinak než pomocí dodaných funkcí. Celkový koncept je připraven spíše tak, že se vše nastaví a poté už jen vykresluje. Přidávání je možné, avšak mazaní je problematické.

Když je vše nastaveno, přichází na řadu tři důležité funkce – vykreslení, vygenerování nových malých čtverců a vymazání malých čtverců pokud přesahují určenou paměť. Doporučuje se používat následující pořadí volání:

1. Generování
2. Vykreslování
3. Promazávání

Důležité je, aby po fázi generování hned nenásledovala fáze mazání. Nejdříve se nově vytvořené čtverce musí vykreslit a tím označit, že se nemají mazat. V opačném případě by mohlo dojít k vymazání toho, co bylo právě vygenerováno. Tím by se došlo k situaci, kdy po překročení nastavené hranice paměti zůstanou některé malé čtverce prázdné a dokud se neotočí pohled, nic se nezmění.

Přehled všech funkcí a výčet jejich vlastností je uveden v příloze 4.

5.2 Vložení vegetace do vlastní aplikace

Vývoj tohoto projektu nejdříve probíhal ve vlastní zkušební aplikaci, do které se postupně přidávaly kousky kódu a testovalo se chování. Během tvorby byl veden stručný záznam průběhu, který je shrnut v následující kapitole.

5.2.1 Vývoj projektu

Nejdříve se v GLUT vytvořilo základní zobrazení okna a přidalo ovládání pozice pomocí myši. Dále následovalo vytvoření jednoduché textury trsu trávy v programu Blender. Těžší částí bylo vygenerování plochy na níž roste vegetace. Jako modely pro vykreslení se použily čtverce, na něž se nanesla vytvořená průhledná textura. Dále se musely naklonit a natočit, což byl výsledek matematiky s vektory. Když byla takto připravena jednoduchá scéna s trsy trávy, použil se jednoduchý shader na rozpohybování vrcholů modelů.

Programově vytvořené modely hrubě nestačily, proto se vložilo načítání modelů ve formátu ASE. Zde se objevil problém otočených souřadnic pro textury, na což naštěstí použitá knihovna glpng pamatovala. Tímto také vzniklo rozdělování vegetace na prototypy ukládající se do různých rovin. Následovala kreativní činnost - vytváření modelů a jejich barevná úprava, protože textury neměly stejné barevné ladění, což se projevilo až při vložení do programu.

Zatím se pracovalo jen s předpřipravenou geometrií na velmi omezeném prostoru. Požadavky však byly mnohem větší než 20x20m vegetace, proto se plocha velkého čtverce pravidelně rozdělila na menší čtverce. Jejich celkový počet se určoval libovolnou hodnotou množství na jedné straně (v praxi – 10 čtverců na stranu znamená dohromady rozdělení na 100 čtverců).

Tyto malé čtverce se ukládaly do jednorozměrného pole ukazatelů. Prázdný ukazatel značí, že v dané ploše ještě není vygenerována vegetace. Ta se tvořila vždy v určitém čtverci kolem pozice pohledu. Každý posun přes hranici malého čtverce tak znamenal značné cuknutí, protože se musel celý nový okraj vygenerovat. Kvůli tomu se omezil počet najednou vygenerovaných malých čtverců.

Nastal čas vložit nějakou optimalizační techniku. Z pohledových matic OpenGL se zjistil komolý kužel pohledu (frustum) a jednotlivé čtverce se kontrolovaly na okrajové body. Tento způsob ovšem v mnoha případech způsobil, že se nevykreslovaly některé části vegetace, protože byť byly uprostřed pohledu, ani jeden okrajový bod nebyl vidět. Toto se na chvíli neřešilo a pozornost byla věnována shaderům

Jako první se implementovalo mizení za hranicí viditelnosti. Ve vertex shaderu se vložil test na vzdálenost a za ní se lineárně zmenšovalo umístění v ose Z. Další bylo řešení mlhy, protože při používání shaderů se její počítání v OpenGL vypíná. Následovalo řešení největšího otazníku – jak uchovat informaci o výšce modelu, aby se vegetace vlnila ve všech úrovních stejně. Do této fáze se používala jen krajina s výškou 0, takže bylo potřeba nějakých způsobem vytvořit výškové rozdíly. Což bylo také nakonec zdárně vyřešeno použitím funkce sinus a vykreslováním malých čtverců s texturou trávy.

Jednou z nejnáročnějších úprav bylo přidání quadtree. Rozdělování plochy vegetace se muselo přizpůsobit způsobu dělení, takže počet malých čtverců na straně mohla být jen mocnina čísla 2. S quadtree přišel i jiný způsob hledání malých čtverců v pohledu – porovnávaly se obalující koule místo krajních bodů malých čtverců. Tím se však získal tolik potřebný větší výkon.

Dále již následovaly drobnější úpravy – pořadí vykreslování se změnilo tak, aby se co nejméně přepínaly textury a hodnoty v shaderu, omezil se počet vykreslovaných trojúhelníků za hranicí viditelnosti, nevygenerované malé čtverce v pohledu dostaly větší prioritu.

Velkým vývojovým skokem bylo přidání ovládání hustoty vegetace. Nejdříve se vložilo načítání textur a poté se na jejich základě generovaly modely. V rámci další úspory se označovaly ty malé čtverce, které v sobě po vygenerování nic neobsahovaly a poté se nemusely kreslit. Aby se při spuštěném programu měnila hustota, byla přidána funkce znovu načítající textury z disku a promazávající celou plochu.

Poněkud experimentální byla část implementace obarvování plochy vegetace. Zkoušely se různé způsoby míchání barev, až byl nakonec zvolen průměr mezi barvou textury modelu vegetace a barvou celkové obarvující textury. V té chvíli na povrch vyplynulo, jak náročné je ve fragment shaderu větvení programu, proto byl kód přepsán do podoby jednoho velkého vzorce.

Protože se schylovalo k závěrečné fázi, ve které šlo o šetření paměti, došlo ke kompletnímu prozkoumání kódu a opravování počítání paměti, kterou vytvořené struktury zabírají. Mezi tím docházelo k dalším drobným úpravám – přesnější nastavení počtu vrcholů posílaných do grafické karty, úpravy ve vykreslování, úplně prázdné malé čtverce se používaly pro další generování (úspora paměti).

První pokus o promazávání nepotřebné paměti se ukázal jako slepá ulička. U každého malého čtverce se počítalo kolikrát byl vykreslen a podle toho se vytvořila fronta kandidátů na vymazání. Bohužel docházelo k mazání nevhodných čtverců a problikávání, takže se muselo přistoupit na promazávání na základě vzdálenosti od pohledu, které se používá doteď – mazání nevzdálenějších malých čtverců.

Poslední úpravy kódu už více souvisí s vkládáním do cizího kódu a odstraňováním chyb, ale to hlavní již bylo z vývoje popsáno.

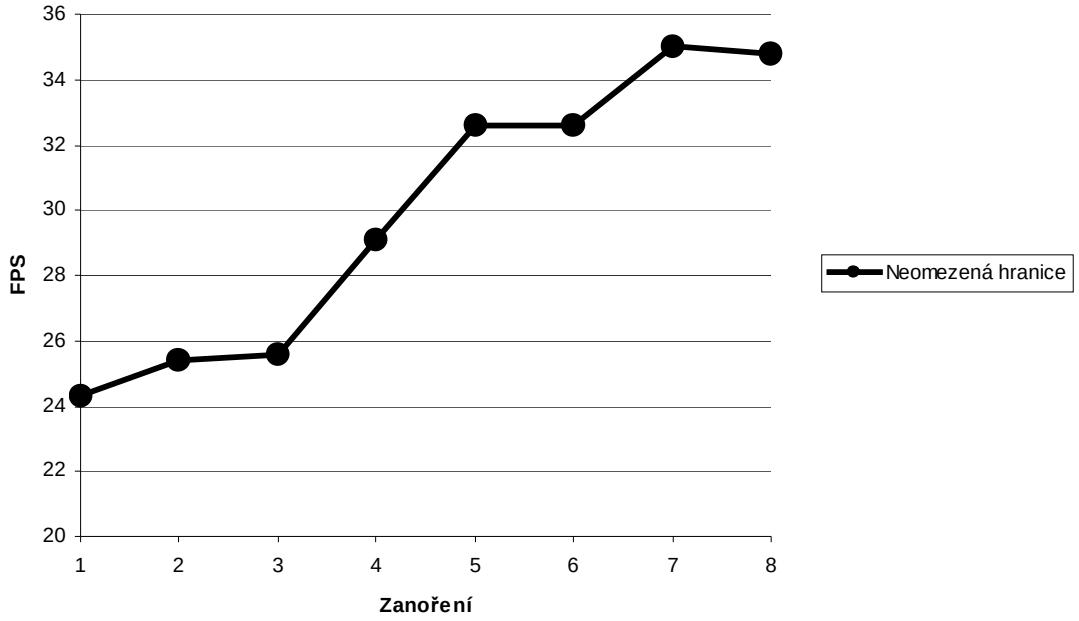
5.2.2 Testy

Při používání vytvořeného nástroje nelze postupovat bez rozmyslu. Pro získání, pokud možno, co největší hodnoty snímků za sekundu (dále jen FPS – frames per second) je potřeba vhodně volit nastavení vegetace. Je logické, že čím víc bude použito modelů ve scéně, tím bude vykreslování náročnější.

Jako hlavní parametr ovlivňující výslednou rychlost je hloubka zanoření quadtree. Touto hodnotou se ovlivňuje celkový počet malých čtverců. Pro zjištění efektu zvyšování tohoto čísla byl připraven terén o rozměrech 100x100 m, s nulovou výškou ve všech místech, obsahující jeden druh vegetace (na obrázku 12, což znamená 8 trojúhelníků na model) a celkový počet kusů na ploše byl 16384. Místo pohledu do scény bylo uprostřed velkého čtverce vegetace směrem do osy Y. Se zvětšováním hloubky quadtree se zmenšoval počet kusů prototypu na malém čtverci (dělí se číslem 4) a tím zůstával celkový počet na ploše zachován. Testováno bylo na notebooku s následující konfigurací:

- Intel Pentium M 1,73 GHz
- 1 MB RAM
- Nvidia GeForce 6600 go 128MB
- Windows XP pro SP2

V grafu 1 je možné vidět výsledek.

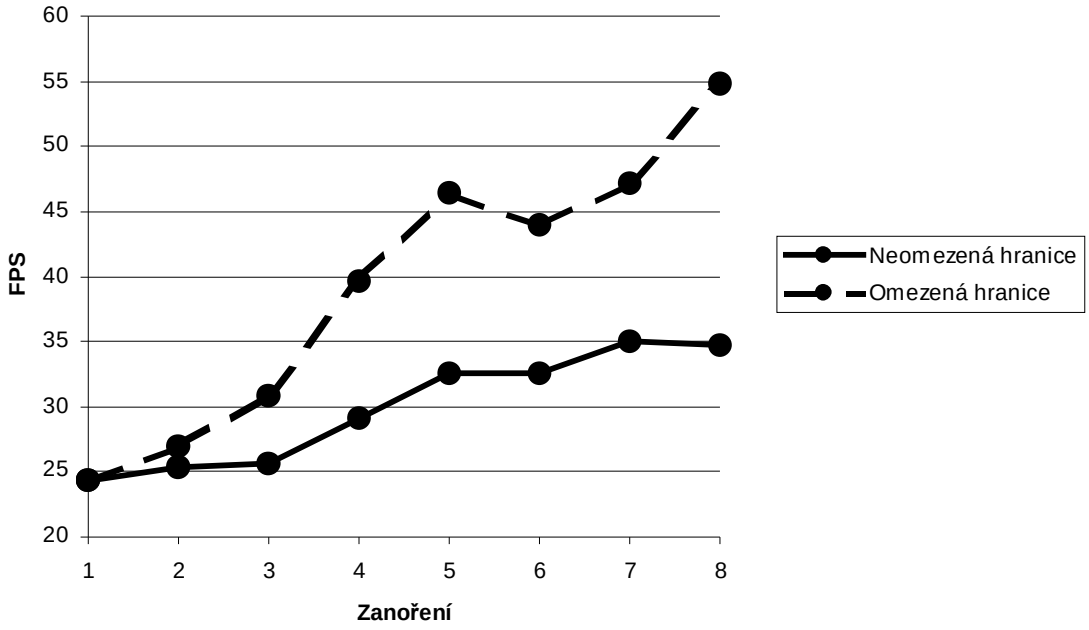


Graf 1: Závislost FPS na maximálním nastaveném zanoření quadtree

Není překvapující, že hodnota FPS roste společně s hodnotou hloubky stromu. Se zvyšujícím se počtem čtverců se zpřesňuje oblast ležící v komolém kuželu pohledu a odpadá čím dál tím víc trojúhelníků, které by se zbytečně posílaly do grafické karty.

I když by se zdálo, že je nejlepší nastavit co největší zanoření, již při nastavení hodnoty 6 (32x32 malých čtvcerců) dochází k nepříjemnému efektu dopočítávání chybějící vegetace. Toto se dá zmírnit nastavením většího čísla u počtu najednou vygenerovaných malých čtverců, je však třeba počítat s tím, že může docházet ke skokům v FPS. Při velké hodnotě může dokonce dojít k viditelnému zastavení programu. Tímto se dostáváme k prvnímu případu, kdy je potřeba se rozhodnout mezi visuální kvalitou a rychlostí vykreslování. K tomuto případu rozhodování však dochází málokdy, protože tu jsou ještě další možnosti optimalizace.

Další parametr, který významně ovlivňuje výkon, je nastavení vzdálenosti vykreslování. Každý prototyp vegetace může mít vlastní hodnotu a čím je menší tím méně se toho musí vykreslovat. Na následujícím grafu 2 (přerušovaná čára) byla pro stejnou scénu použita hodnota hranice vzdálenosti 25, což znamená, že trsy mizí na polovině vzdálenosti od okraje velkého čtverce.

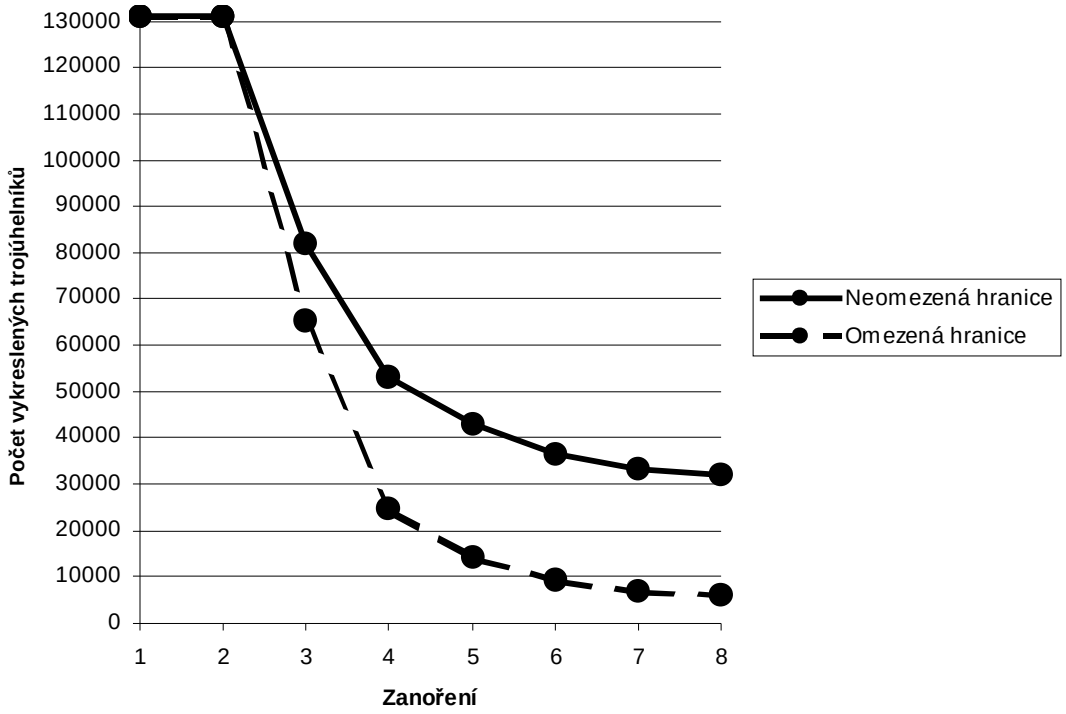


Graf 2: Porovnání závislostí FPS na maximálním nastaveném zanoření quadtree

Oproti předchozímu případu je nárůst FPS mnohem větší, protože se razantně sníží počet vykreslovaných trojúhelníků jejichž počet je naznačen v grafu 3 (maximální počet 8x16384 = 131072). Zajímavým zjištěním bylo, že zatímco počet trojúhelníků se pravidelně zmenšoval, zvyšování FPS je značně kolísavé.

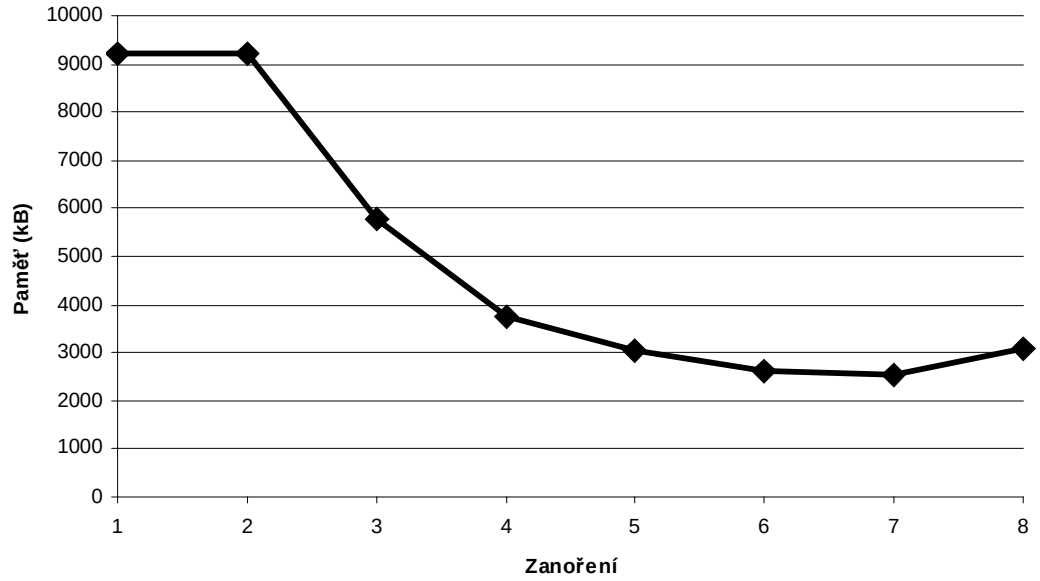
S jednoznačnou úsporou výkonu (a paměti) souvisí i určování hustoty vegetace, která vlastně funguje na stejném principu jako hranice viditelnosti. Čím větší je hloubka quadtree, tím více se toho nevykresluje, ale roste čas generování vegetace k vykreslení.

Tímto zkoušením je potřeba najít co nejlepší poměr mezi složitostí modelů prototypů, jejich počtem, hranicí viditelnosti, hloubkou quadtree a visuální kvalitou. Toto rozhodování velmi ovlivňuje velikost plochy, na kterou chceme vegetaci vysázet. Během vývoje se empiricky zjistilo, že ideální velikost malého čtverce se pohybuje něco mezi 20 až 2 metry. Přirozeně záleží na typu pohybu nad krajinou. Pokud se vegetace používá jako doplňující efekt při průletech vysoko nad krajinou, mohou být větší. Pokud se krajina prochází ve výšce očí dospělého člověka, vyplatí se naopak čtverce menší.



Graf 3: Závislost počtu vykreslených trojúhelníků na maximálním nastaveném zanoření quadtree

Je třeba myslet i na paměťovou náročnost jednoho malého čtverce. U většího počtu malých čtverců se lépe ovládá, ale je s tím spojená větší režie přidávání a mazání. Rovněž klesá množství paměti nutné pro uložení vegetace v pohledu, jako naznačuje graf 4.



Graf 4: Závislost potřebné paměti pro vykreslení na maximálním nastaveném zanoření quadtree

Průběh grafu by měl teoreticky přesně kopírovat křivku vykreslených trojúhelníků, ale je třeba brát v úvahu, že samotné struktury quadtree a malých čtverců zabírají čím dál tím větší paměť.

U všech testů (i těch následujících) je třeba brát v úvahu, že program běžel na téměř tři roky starém počítači. Na aktuálních grafických kartách budou výsledky úplně jiné.

5.3 Vložení vegetace do Tank game

Tank game je název projektu, jenž vzniká na fakultě FIT postupným přidáváním částí založených na modulárním principu. Pro tuto práci je důležité, že obsahuje terén, na který je možné vložit vegetaci.

5.3.1 Průběh vložení

Základem bylo vložit zdrojové kódy vegetace do Tank game a společně přeložit. To proběhlo relativně hladce, pokud se pomine problém s knihovnou glew, která vyžaduje, aby byla vložena dříve, než knihovna GL. Následovalo zjištění nejdůležitější části a to funkce výšky na určité souřadnici. Tank Game tuto funkci již implementovanou má, ale objektově vystavěná struktura programu nedovolila přímo předat ukazatel na potřebnou funkci. Proto se vytvořila nová funkce odpovídajícího formátu získávající hodnotu přes globální proměnnou.

Při vkládání prototypů se nikterak závažné problémy neobjevily. Jediný, který snad stojí za zmínku je obrácené vykreslování textur vegetace. Po delší době vyplulo na povrch, že byl problém v inicializaci vegetace, což se záhy opravilo.

Další problém nastal při získávání obarvovací textury, která byla rafinovaně skryta v objektech Tank game. Důležité bylo, že se na celou velkou plochu terénu používala jen jedna, tudíž se nemusela nijak spojovat, nebo podobně upravovat.

Pro urychlení zkoušení map hustoty se přidala reakce na klávesu provádějící jejich znovunačtení. Stejně tak se přidalo vypínání vykreslování vegetace (pro zjištění náročnosti). Zbytek už se sestával z přidávání prototypů a testování.

5.3.2 Výsledek

Tank game zobrazuje nekonečný terén s čímž tato vegetace nepočítá. Naštěstí je nekonečnost vyřešena opakujícími se čtverci, které na sebe navazují. Velikost těchto čtverců je 2048x2048 m, což je pro otestování zobrazování na velkých plochách ideální, byť za hranicí čtverce umístěného v počátku souřadnic vegetace náhle končí.

Samotná vegetace vypadá nejlépe při použití obarvování, kdy její barva ladí k podkladu terénu. Ten obsahuje velké barevné výkyvy (od hnědé po zelenou), takže bez obarvení by byly jednotlivé modely velmi nápadné.

Protože velká část povrchu jsou hory, mapy hustoty mohly určit růst jen na několika omezených místech, většinou kopírující globální texturu, kde je zelená barva.

Postupným zkoušením parametrů se přišlo na to, že ideální hloubkou quadtree je hodnota 9. Tímto vznikne velikost malého čtverce 8x8 m, což je možná vzhledem ke stylu pohybu pohledu v Tank Game zbytečně málo, ale pozitivní vliv na FPS to má.

Při zjišťování ideální hustoty trávy se vyzkoušel způsob, který svým způsobem připomíná metodu LOD (Level Of Detail). Vytvořily se tři prototypy se stejným modelem, ale s různými stupni hranice viditelnosti. Počet kusů modelů se mohl značně snížit a ve výsledku nejblíže pohledu rostla tráva velmi hustě, o kus dál méně a ještě dál velmi řídce. A protože se podle vzdálenosti zvolily vhodné hodnoty, není na první pohled nic vidět.

Dalším experimentem bylo použití nástroje pro zobrazování nízké vegetace na zobrazování jiných objektů než doposud. Místo modelu trsu trávy, či jiné vegetace se použil model kamene. Ve své podstatě takovýto druh modelu odpovídá charakteristikám vegetace. Jediné, co se nevyužívalo, byl pohyb ve větru. Nastavil se rozptyl velikosti, kde se mají vyskytovat a následovalo vykreslení. Přesně podle předpokladu nebyl v ničem problém a kameny krajinu velmi dobře doplnily.

Ve chvíli, kdy byla vložena vegetace s přijatelným visuálním vjemem, vyzkoušela se náročnost. Krajina bez vegetace měla v průměru 150 FPS, při zapnutí spadla až na 20 FPS, což je velmi špatný výsledek. FPS lehce ovlivňovalo rozlišení v jakém se celá scéna zobrazovala. Při menším se poměr FPS bez vegetace a s vegetací zlepšil, ale stále je příliš náročná.

Vzhledem k tomu, že není využívaná celá plocha vegetace, zkusila se zmenšit velikost velkého čtverce na polovinu (1024x1024 m). Stejně tak se zmenšilo zanoření. Výsledkem byl další nárůst FPS, z čehož vyplývá přirozený fakt, že se rozhodně vyplatí nastavit velikost, pokud možno, co nejmenší, těsně obalující prostor, kde bude vyžadována vegetace.

Ukázka vegetace vložené do Tank Game je vložena do přílohy 2.

5.4 Vložení vegetace do Terrain Engine

Terrain Engine je program, jehož zdrojové kódy jsou volně dostupné na internetu [15] a bez jakékoliv optimalizace zobrazuje krajinu o rozloze 1x1 km.

5.4.1 Průběh vložení

Oproti Tank Game, kde veškerá grafika byla vygenerována příslušnými algoritmy v průběhu spouštění programu, Terrain Engine používá na vše textury načtené ze souboru. Terén se vytvoří pomocí výškové textury, obarví se podle textury krajiny a vylepší se detailní texturou.

I když se zpočátku zdálo, že textury uložené na disku usnadní práci, problémy se vyskytly někde jinde. Jako první se projevil fakt, že Terrain Engine k vykreslování krajiny používá, stejně jako tento projekt, vertex array. To by v normálním případě nevadilo (ostatně Tank Game je používá také),

ale nastavení ukazatelů na pole s vrcholy se provádí jen při inicializaci programu, tudíž ve výsledku krajina zmizela. Muselo se proto přidat nastavování ukazatelů do vykreslovací smyčky.

Hned jako druhý problém byla situace, s níž vegetace nepočítá – prohození globálních os. Zatímco vegetace bere za osu výšky Z (kapitola 3.2.1), Terrain Engine používá osu Y. Řešením bylo vynásobení pohledové matice maticí na obrázku 20 před samotným vykreslením vegetace, jenž vlastně obě osy Z a Y prohodí. Kromě tohoto prohození se ještě v shaderu provádělo otočení osy Y v mapování globální obarvovací textury.

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

Obrázek 20: Matice prohazující osu Y za osu Z

Posledním větším problémem byla úplná absence funkce pro návrat výšky na pozici X, Y. Během inicializace se geometrie terénu vytvářela podle výškové mapy, ale poté se smazala. Nezbylo než mazání odstranit a podle části kódu generující terén vytvořit vlastní funkci vracející požadovanou hodnotu.

Během vkládání odpovídající vegetace se projevila chyba, jenž se ze záhadného důvodu před tím ani v Tank Game neprojevila. Šlo o chybu ve vertex shaderu, kdy se špatně počítala vzdálenost vrcholu od pozice pohledu. Důvodem byla složka vrcholu W, která se nastavovala na hodnotu 1,0 (kapitola 4.9.1) později, než docházelo k výpočtu vzdálenosti. Projevovala se vegetací umístěnou pod samotným povrchem terénu.

Když bylo toto vše vyřešeno, mohlo se začít s vkládáním prototypů odpovídající rázu krajiny.

5.4.2 Výsledek

Krajina Terrain Engine je mnohem barevnější než u Tank Game (příloha 3.), takže použití globálního obarvení bylo téměř nutností. Podle stejné textury, jaká se použila na obarvení, se vytvořila i mapa hustoty. Kde převládala zelená barva, tam tráva rostla nejhustěji.

Nejvýhodnější hodnota zanoření quadtree se ukázala jako 8, což opět odpovídá velikosti přibližně 8x8 m. Pro menší nároky se použil stejný postup jako v předchozím případě – tři prototypy s rozdílnou hranicí viditelnosti, jejichž počet kusů modelů zůstal po delším zkoušení na hodnotě 70.

Ve výsledku je původní krajina hodně visuálně vylepšená, bohužel za cenu velkých nároků. Propad FPS je sice o něco menší než u Tank Game, ale i tak chvíli trvalo, než se dosáhlo nejoptimálnějšího výsledku.

Závěr

Byla prostudována problematika zobrazení vegetace ve 3D prostoru. Popsaly se, implementovaly a zhodnotily se algoritmy použitelné pro tuto tématiku. Vzniklý kód se vložil do existujících systémů zobrazující 3D krajinu a výsledek se zhodnotil, čímž se splnilo zadání diplomové práce.

Během tvorby projektu vycházelo čím dál tím víc najevo, že vegetace vložená do terénu nepatří mezi visuální vylepšení, která stojí jen zlomek celkového výkonu. Ve skutečnosti se jedná o velmi náročnou záležitost, proto je použití potřeba zvážit. I když je implementace schopná vykreslovat vegetaci na větších plochách, například pro vizualizaci v geografických informačních systémech se zatím příliš nehodí. Své uplatnění najde spíše tam, kde je kamera pohledu umístěna blízko povrchu terénu. Může se jednat například o zobrazování architektury (procházení domu a přilehlého pozemku v reálném čase), vojenské simulátory či počítačové hry.

Samotné vložení kódu vegetace do již existujícího systému je bez problému, pokud má programátor přehled o způsobu vykreslování obou částí (na úrovni stavového stroje OpenGL). Ve chvíli, kdy je vegetace úspěšně zakomponována, začíná hledání optimálního vyvážení vzhledu a náročnosti. Na tomto poměru se velmi projevuje kvalita vytvořených modelů a textur. Jako nejvýhodnější se ukázaly málo složité modely s propracovanou texturou. Rovněž závisí na hardwaru, který se používá. Vývoj projektu probíhal na starších grafických kartách, takže ty dnešní karty zvládnout na větší ploše mnohem hustší vegetaci a výkon poroste dál.

S rostoucím výkonem rostou i rezervy pro přidávání efektů, což souvisí s dalším vylepšováním tohoto projektu. Bylo by možné se zaměřit na osvětlování a vrhání dynamických stínů, protože v současném stavu lze ovládat jen barevný odstín. Pokud by vznikla potřeba použití na opravdu velkých plochách (zmiňovaný GIS), pak by se vývoj mohl zaměřit na důmyslnější správu paměti. Otevřená je i oblast další optimalizace množství dat posílaných k vykreslení. Mimo programovací část diplomové práce by bylo vhodné zvětšit počet druhů vegetace, protože pro testování funkčnosti algoritmů jich stačilo jen několik a nekladl se na ně takový důraz.

Celkově projekt splnil vložená očekávání a přispěl do nepříliš zdokumentovaného tématu.

Literatura

[1] Rendering Grass Terrains in Real-Time with Dynamic Lighting. [online], [cit. 2008-05-05]. URL <http://www.irisa.fr/siames/GENS/kboulang/grass.html>

[2] Seriously! - Leading coverage of Serious Sam 2 and other Serious Engine Games. [online], [cit. 2008-05-05]. URL <http://www.seriouszone.com/>

[3] Far Cry - Wikipedia, the free encyclopedia. [online], [cit. 2008-05-05]. URL <http://en.wikipedia.org/wiki/Far_Cry>

[4] EA – Battlefield 2. [online], [cit. 2008-05-05]. URL <http://www.ea.com/official/battlefield/battlefield2/us/>

[5] The Elder Scrolls. [online], [cit. 2008-05-05]. URL <http://www.elderscrolls.com/home/home.php>

[6] MOLER, T; HAINES, E.: Real-Time Rendering. A K PETERS, 1999, ISBN 1-56881-101-2, s. 330-335.

[7] POLACK, T.: Focus on 3D Terrain Programming. Premier Press Books, Prosinec 2002, ISBN 1-59200-028-2, s. 105-126.

[8] ENDGEL, W.: ShaderX^4. Charles River Media, 2006, ISBN 1-58450-425-1.

[9] POKORNÝ, P.: Blender - naučte se 3D grafiku. BEN – technická literatura, první vydání, Říjen 2006, ISBN 80-7300-203-5.

[10] VYBÍRAL, J.: GIMP. Computer Press, Duben 2004, ISBN: 80-251-0158-4.

[11] GLUT API, version 3. [online], [cit. 2008-05-05]. URL <http://www.opengl.org/resources/libraries/glut/spec3/spec3.html>

[12] GLEW: The OpenGL Extension Wrangler Library. [online], [cit. 2008-05-05]. URL <http://glew.sourceforge.net/>

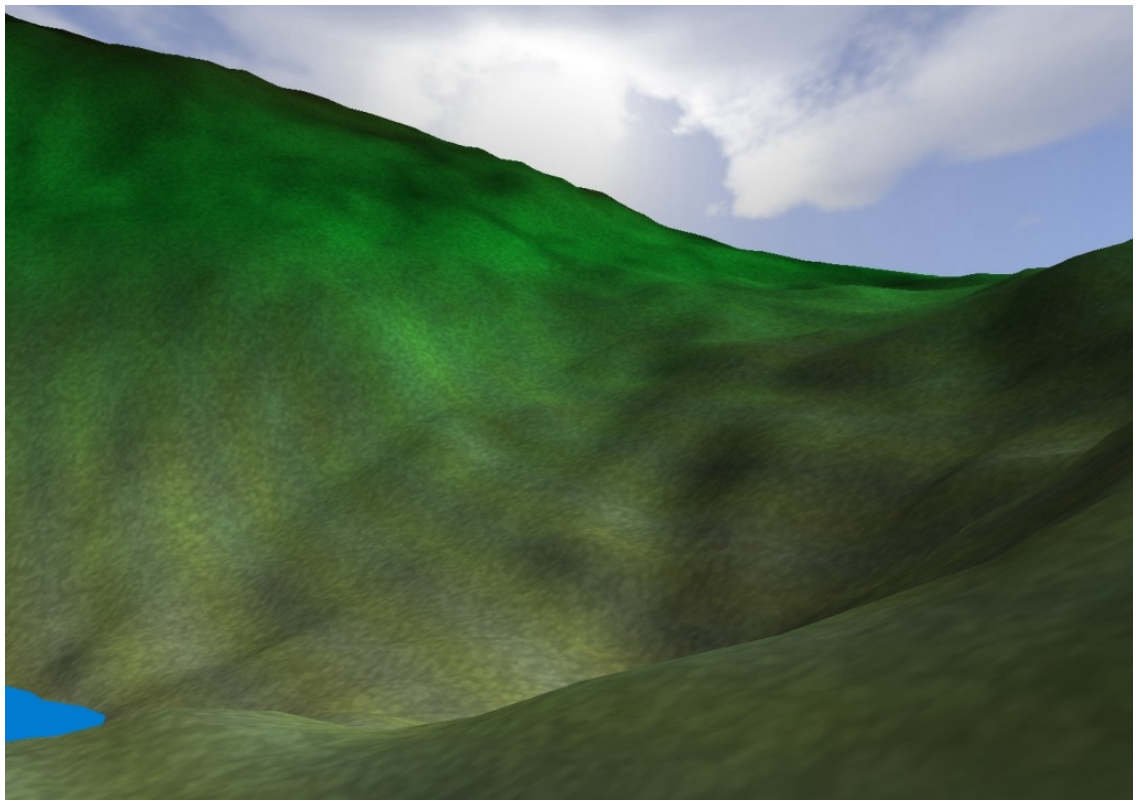
[13] glpng. [online], [cit. 2008-05-05]. URL <http://www.fifi.org/doc/libglpng-dev/glpng.html>

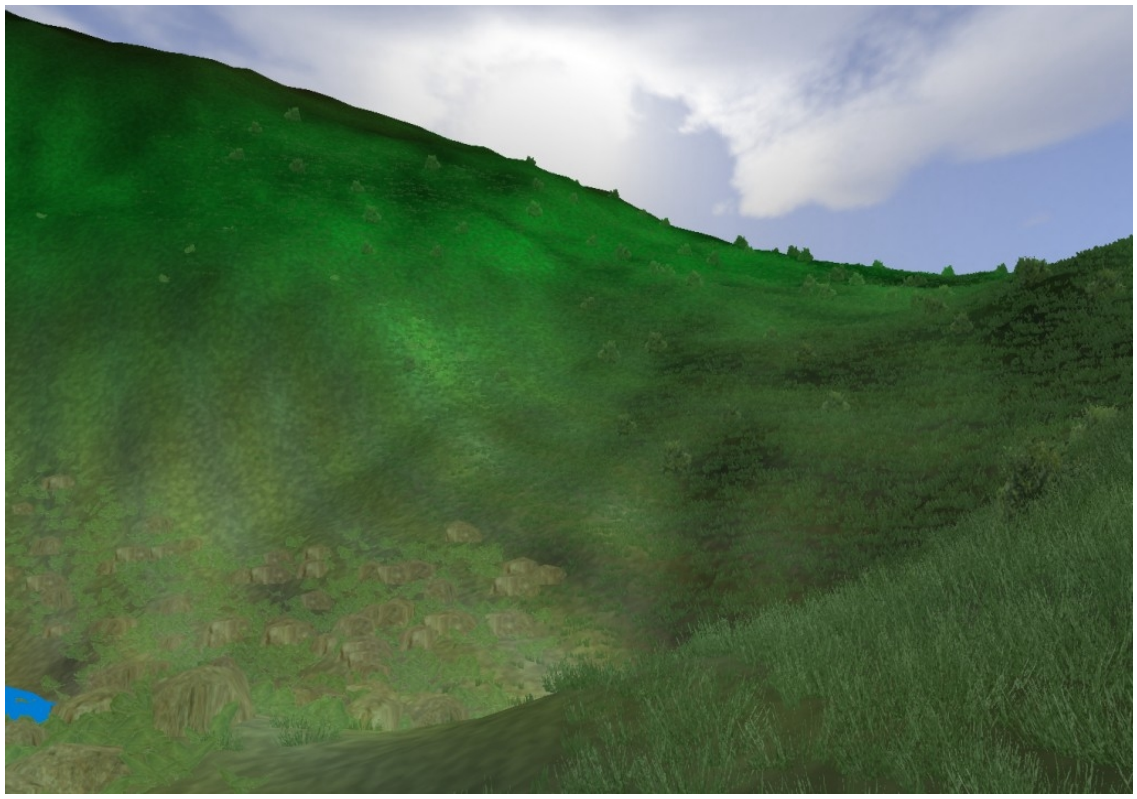
[14] ROST, J. R.: Opengl Shading Language. Addison-Wesley Publishing, druhé vydání, Leden 2006, ISBN 0-321-33489-2.

[15] Computer graphics experiments (OpenGL, DirectX, Terrains, GIS...). [online], [cit. 2008-05-05]. URL < http://ohad.visual-i.com/exper/exper.htm>

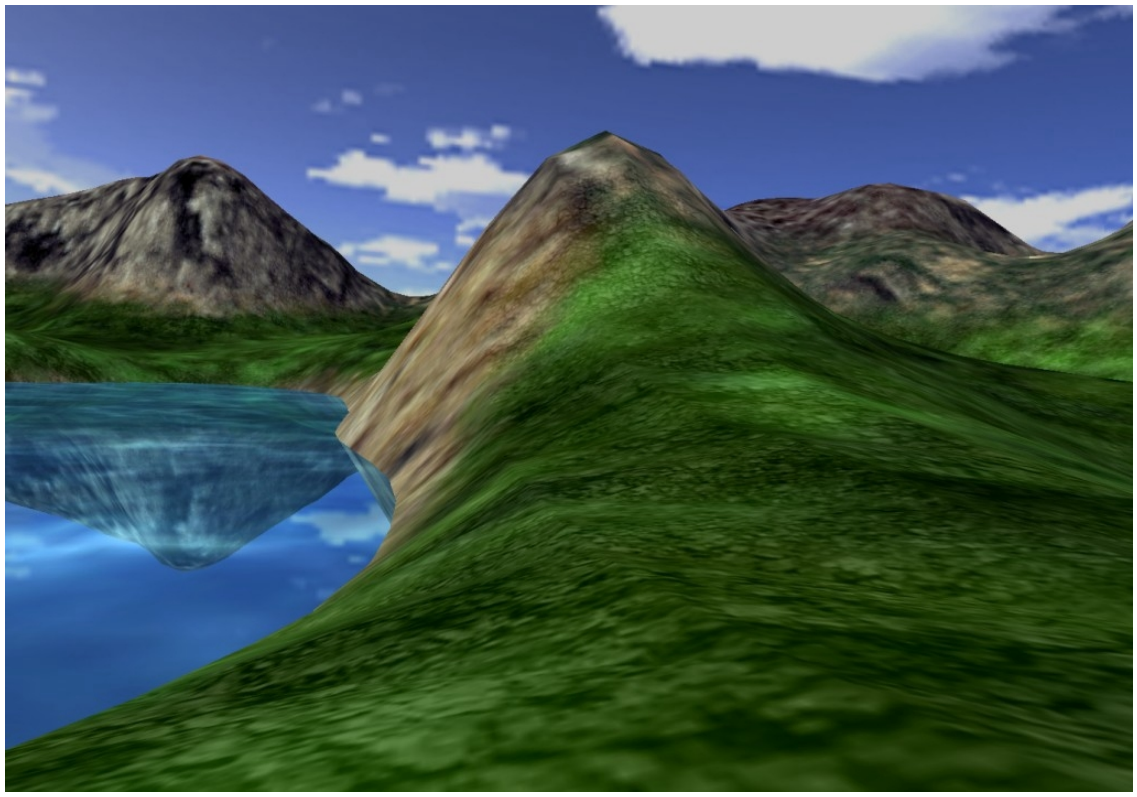
Seznam příloh

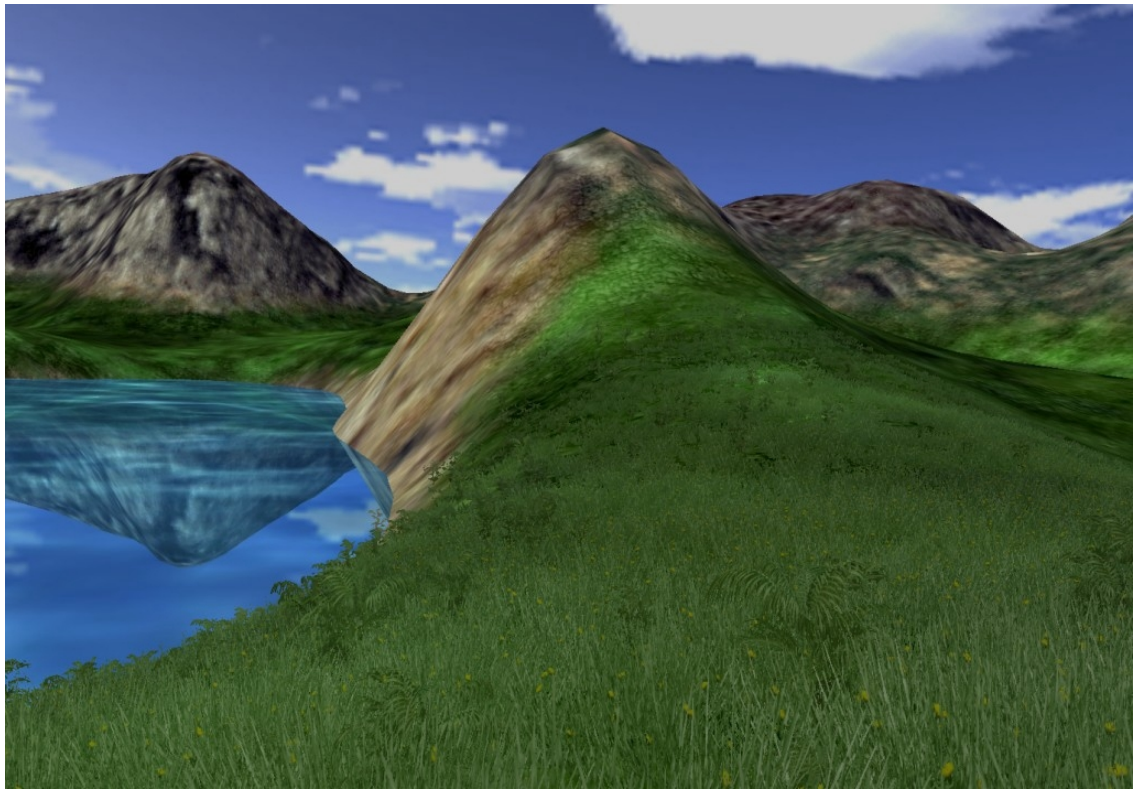
Příloha 1. Obrázky – Tank Game





Příloha 2. Obrázky – Terrain Engine





Příloha 3. Dokumentace funkcí

Následující seznam a popis není kompletní přehled všech funkcí vytvořených pro vegetaci. Obsahuje jen ty, které přímo ovládají její vlastnosti, nebo s ní pracují.

Funkce ovládající vegetaci

int generateSmallQuads(BigVegQuad * veg, float x, float y);

Vygeneruje malé čtverce. Vlastnosti generovaní jsou nastaveny ve *veg*.

veg – velký čtverec vegetace

x, y – pozice pohledu

void drawVegQuadtree(BigVegQuad * veg, float time, float x, float y, float z);

Vykreslí vegetaci podle aktuálně nastavených matic pohledu.

veg – velký čtverec vegetace

x, y, z – pozice pohledu

time – aktuální čas v sekundách

Funkce nastavující vlastnosti celé plochy vegetace

int createVegQuad(BigVegQuad ** veg, float (*getHeight)(float,float), char * path);

Inicializuje ukazatel na vegetaci.

veg – velký čtverec vegetace

getHeight – ukazatel na funkci vracející výšku na určité pozici

path – cesta k shaderům „veg.vert“ a „veg.frag“

void setVegQuadMaxMB(BigVegQuad * veg, unsigned int size);

Nastaví maximální hranici paměti, při jejíž překročení se bude promazávat pole s malými čtverci.

veg – velký čtverec vegetace

size – velikost paměti v MB

void setVegQuadTexturesPath(BigVegQuad * veg, char * path);

Nastaví cestu k adresáři, odkud se budou načítat textury.

veg – velký čtverec vegetace

path – cesta k adresáři

void setVegQuadModelsPath(BigVegQuad * veg, char * path);

Nastaví cestu k adresáři, odkud se budou načítat modely.

veg – velký čtverec vegetace

path – cesta k adresáři

void setVegQuadSize(BigVegQuad * veg, float size);

Nastaví velikost velkého čtverce vegetace.

veg – velký čtverec vegetace

size – velikost strany čtverce

void setVegQuadtreeDepth(BigVegQuad * veg, int depth);

Nastaví hloubku zanoření quadtree.

veg – velký čtverec vegetace

depth – velikost strany čtverce

void setVegSmallQuadsSideCountArea(BigVegQuad * veg, int count);

Nastaví počet malých čtverců na stranu čtverce, ohraničující ty, které je možné generovat.

veg – velký čtverec vegetace

count – počet malých čtverců na stranu čtverce

void setVegSmallQuadsAllocStep(BigVegQuad * veg, int count);

Nastaví počet malých čtverců, které najednou funkce *generateSmallQuads* vytvoří.

veg – velký čtverec vegetace

count – počet malých čtverců

void setVegQuadCenter(BigVegQuad * veg, float x, float y);

Nastaví střed velkého čtverce vegetace.

veg – velký čtverec vegetace

x, y – pozice středu

Funkce nastavující vlastnosti prototypů

void setPrototypeModel(BigVegQuad * veg, char * name, char * path);

Nastaví prototypu model vegetace.

veg – velký čtverec vegetace

name – jméno prototypu

path – cesta s souboru ASE

void setPrototypeCount(BigVegQuad * veg, char * name, int count);

Nastaví prototypu počet kusů modelů na jednom malém čtverci.

veg – velký čtverec vegetace

name – jméno prototypu

count – počet kusů modelů

void setPrototypeDistance(BigVegQuad * veg, char * name, float distance);

Nastaví prototypu vzdálenost, za kterou bude vegetace mizet.

veg – velký čtverec vegetace

name – jméno prototypu

distance – vzdálenost hranice viditelnosti

void setPrototypeScale(BigVegQuad * veg, char * name, float scale);

Nastaví prototypu šířku pohybu trávy.

veg – velký čtverec vegetace

name – jméno prototypu

scale – šířka pohybu trávy ve výšce 1 m

void setPrototypeSpeed(BigVegQuad * veg, char * name, float speed);

Nastaví prototypu rychlost pohybu trávy.

veg – velký čtverec vegetace

name – jméno prototypu

speed – dvojnásobek jednoho průběhu pohybu

void setPrototypeFog(BigVegQuad * veg, char * name, float fog);

Nastaví prototypu, zda bude zobrazován v mlze.

veg – velký čtverec vegetace

name – jméno prototypu

fog – 0.0 vypnuta, 1.0 zapnuta

void setPrototypeGlobalMap(BigVegQuad * veg, char * name, float global);

Nastaví prototypu, zda bude míchat barvu modelu s globální obarvovací texturou.

veg – velký čtverec vegetace

name – jméno prototypu

global – 0.0 vypnuto, 1.0 zapnuto

void setPrototypeAngle(BigVegQuad * veg, char * name, float angle);

Nastaví modelu prototypu maximální úhel natočení, ve kterém se bude generovat.

veg – velký čtverec vegetace

name – jméno prototypu

angle – úhel ve stupních

void setPrototypeBank(BigVegQuad * veg, char * name, float bank);

Nastaví modelu prototypu maximální úhel naklonění, ve kterém se bude generovat.

veg – velký čtverec vegetace

name – jméno prototypu

angle – úhel ve stupních

void setPrototypeMaxSize(BigVegQuad * veg, char * name, float max_size);

Nastaví modelu prototypu maximální velikost.

veg – velký čtverec vegetace

name – jméno prototypu

size – násobek velikosti (1.0 – model nezmění velikost)

void setPrototypeMinSize(BigVegQuad * veg, char * name, float min_size);

Nastaví modelu prototypu minimální velikost.

veg – velký čtverec vegetace

name – jméno prototypu

size – násobek velikosti (1.0 – model nezmění velikost)

void setPrototypeDensityMap(BigVegQuad * veg, char * name, char * d_name);

Přidá prototypu odkaz na mapu hustoty.

veg – velký čtverec vegetace

name – jméno prototypu

d_name – jméno mapy hustoty

void setPrototypeDefaultDensity(BigVegQuad * veg, char * name, unsigned char value);

Nastaví prototypu standardní hodnotu používanou v případě, že se nepoužije mapa hustoty.

veg – velký čtverec vegetace

name – jméno prototypu

value – hodnota v rozsahu 0..255

Funkce nastavující vlastnosti globální obarvovací textury

void setVegGlobalTexture(BigVegQuad * veg, char * name);

Načte a nastaví globální obarvovací texturu.

veg – velký čtverec vegetace

name – cesta k textuře

void setVegGlobalTexture(BigVegQuad * veg, GLuint tex);

Nastaví texturu globální obarvovací texturu.

veg – velký čtverec vegetace

tex – index na již načtenou texturu

void setVegGlobalTextureCenter(BigVegQuad * veg, float x, float y);

Nastaví střed globální obarvovací textury.

veg – velký čtverec vegetace

x, y – pozice středu

void setVegGlobalTextureSize(BigVegQuad * veg, float size);

Nastaví velikost globální obarvovací textury.

veg – velký čtverec vegetace

size – velikost strany čtverce textury

Funkce nastavující vlastnosti map hustoty

void setDensityMapTexture(BigVegQuad * veg, char * name, char * path);

Načte a nastaví mapu hustoty.

veg – velký čtverec vegetace

name – jméno mapy hustoty

path– cesta k souboru s obrázkem mapy hustoty

void setDensityMapData(BigVegQuad * veg, char * name, int res, unsigned char * data);

Nastaví zadané data jako mapu textury.

veg – velký čtverec vegetace

name – jméno mapy hustoty

data – cesta k souboru s obrázkem mapy hustoty

void setDensityMapSize(BigVegQuad * veg, char * name, float size);

Nastaví velikost mapy hustoty.

veg – velký čtverec vegetace

name – jméno mapy hustoty

size – velikost strany čtverce mapy hustoty

void setDensityMapCenter(BigVegQuad * veg, char * name, float x, float y);

Nastaví střed mapy hustoty.

veg – velký čtverec vegetace

x, y – pozice středu

Funkce pro mazání

void destroyVegQuad(BigVegQuad ** veg);

Kompletně vymaže vegetaci.

veg – velký čtverec vegetace

void clearAllSmallVegQuads(BigVegQuad * veg);

Vymaže všechny vygenerované malé čtverce.

veg – velký čtverec vegetace

void clearUnusedSmallVegQuads(BigVegQuad * veg);

Vymaže jen ty malé čtverce, které nejsou v pohledu a jsou daleko od jeho středu.

veg – velký čtverec vegetace

void clearPrototypes(BigVegQuad * veg);

Vymaže všechny prototypy.

veg – velký čtverec vegetace

void clearDensityMap(BigVegQuad * veg);

Vymaže všechny mapy hustoty.

veg – velký čtverec vegetace

void clearGlobalTexture(BigVegQuad * veg)

Vymaže globální obarvovací texturu.

veg – velký čtverec vegetace