VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

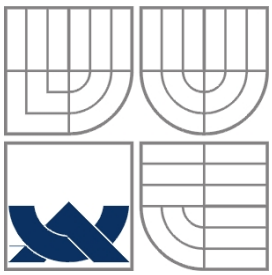
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ZOBRAZOVÁNÍ 3D KRAJINY SE SVĚTELNOU MAPOU

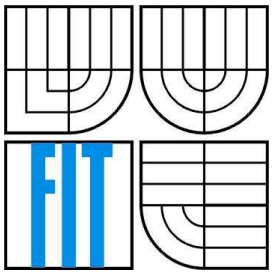
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE Pavel Fadrhonc
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ZOBRAZOVÁNÍ 3D KRAJINY SE SVĚTELNOU MAPOU

3D LANDSCAPE RENDERING WITH LIGHT-MAP

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

Pavel Fadrhonc

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Jiří Venera

BRNO 2007

Abstrakt

Cílem práce je nastudovat způsob zobrazování 3D krajiny v počítači v reálném čase a implementovat jednoduchý program, který toto demonstruje za použití vybraného algoritmu pro generování výškových a světelných map.

Klíčová slova

3D krajina, světelná mapa, výšková mapa, fault formation, procedurální generování textur, OpenGL, C++, kamera, detailní mapování

Abstract

Goal of this thesis is to study methods of realtime 3D landscape rendering and implement simple demonstration program which uses selected algorithm for heightmap and lightmap generation.

Keywords

3D landscape, lightmap, heightmap, fault formation, procedural texture generation, OpenGL, C++, camera, detail mapping

Citace

Pavel Fadrhonc: Zobrazování 3D krajiny se světelnou mapou. Brno, 2008, bakalářská práce, FIT VUT v Brně.

Zobrazování 3D krajiny se světelnou mapou

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Jiřího Venery, Ing. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jméno Příjmení
Datum

Poděkování

Mé poděkování za pomoc při vytváření této práce patří zejména mému vedoucímu Jiřímu Venerovi, Ing.

Obsah

Úvod

Tato práce má za cíl nastudovat algoritmy pro zobrazování 3D krajiny v reálném čase pomocí výškových map a metod pro zvýšení detailu zobrazovaného terénu, jako jsou procedurální generování textur, detailní mapování a osvětlování pomocí světelných map. Světelné mapy, stejně jako výškové jsou generované pomocí speciálních algoritmů, jako jsou fault formation a midpoint displacement. V práci bude detailně popsán jejich algoritmus, stejně tak metoda erozního filtru, která se na výslednou mapu aplikuje za účelem uhlazení terénu a tím pádem zvýšení realističnosti. Dále bude popsáno jaké výsledky je možné s různými parametry těchto algoritmů dosáhnout a také pro co je který algoritmus více vhodný.

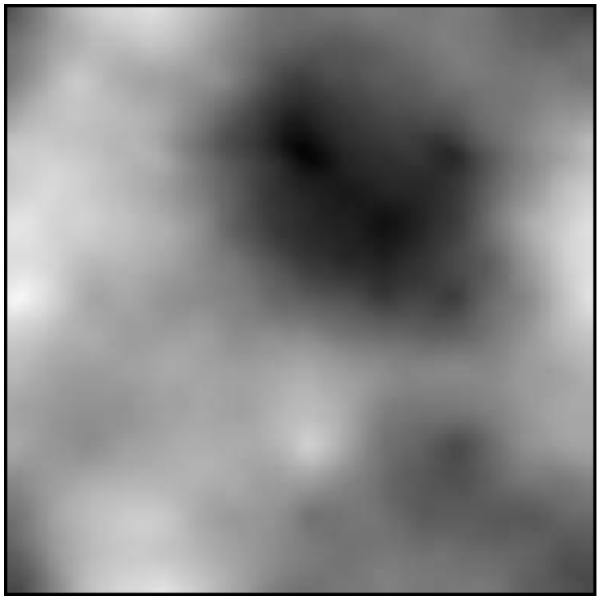
Bude popsána způsob procedurálního generování textur na základě výšky terénu. Tato technika texturování 3D krajiny využívá několik vzorů povrchu, které vytváří podle výšky terénu. Umožňuje plynulý přechod mezi jednotlivými povrchy (hlína, tráva) a poskytuje tak dynamický způsob realistického vytváření textur. Realističnost povrchu terénu dále umocňuje jednoduchá technika s výbornými výsledky, a tou je detailní mapování. Postup aplikace detailní mapy bude rovněž popsán.

1 Výškové a světelné mapy a jejich generování

1.1 Co jsou světelné a výškové mapy

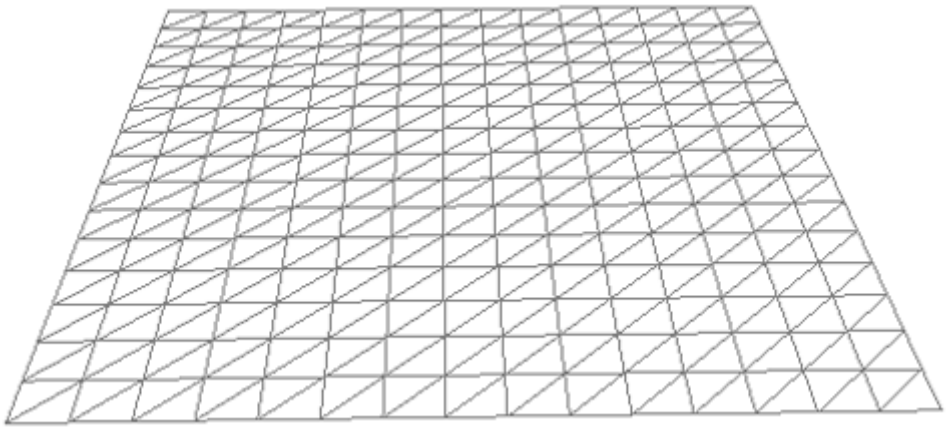
V počítačové grafice je termínem výšková mapa označována rastrová bitmapa hodnot, které určují výšku v daném bodě. Každý pixel v bitmapě o souřadnicích x a z značí výšku, neboli polohu y příslušného bodu ve 3D prostoru. Výšková mapa je obvykle reprezentována jako rast o rozměrech 2^n x 2^n kde každý bod je určitým stupněm šedi a určuje výšku. Světlejší hodnoty značí výšší místa v terénu a tmavší nižší. Tato reprezentace pro zobrazení 3D terénu je velice výhodná, neboť výšková mapa je vlastně obyčejný rastr, který lze editovat v libovolném editoru. Toho se využívá v nejrůznějších 3D grafických enginech, kde si uživatelé lehce můžou vytvořit vlastní terén například v grafickém editoru, nebo v některém programu, který výškové mapy generuje na základě algoritmů pro generování šumu, nebo speciálních algoritmů, jako jsou fault formation a midpoint displacement, které budu dále popisovat v této práci. Grafickému enginu potom již poskytnou pouze výškovou mapu a on si dokáže vše potřebné, jako je vykreslování, texturování, osvětlování a případně umisťování dalších objektů obstarat sám.

Z výškové mapy se vykresluje terén nejběžněji pomocí trojúhelníkové sítě, kdy právě vrcholy každého trohúhelníku v takové síti jsou posunuty do výšky na základě údajů ve výškové mapě. Obrázek 1 ukazuje příklad výškové mapy o rozměrech 128x128.



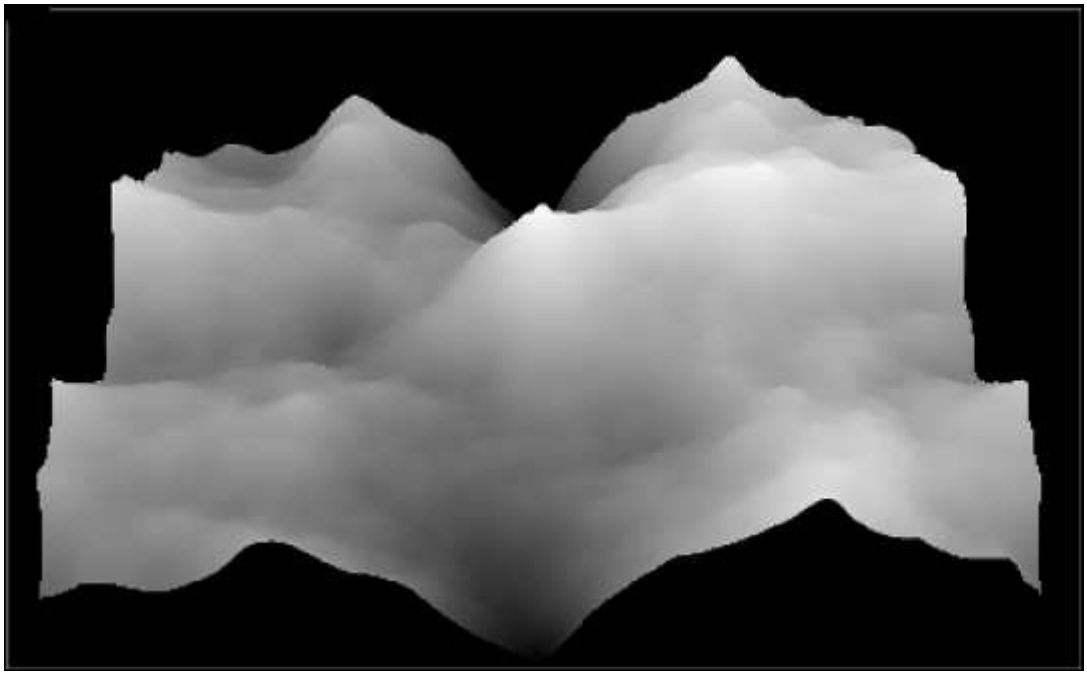
Obrázek 1

Obrázek 2 ukazuje trojúhelníkovou síť před aplikací výškové mapy



Obrázek 2

Obrázek 3 ukazuje jaký terén vznikne z aplikace výškové mapy z obrázku 1 na plochou trojúhelníkovou síť na obrázku 2. Všimněte si, jak přesně odpovídají vysoká a nízká místa v terénu výškové mapě včetně odstínů šedi.



Obrázek 3

Světelná mapa je postavena na naprosto stejném principu jako výšková mapa; je to tedy rastr 256 bitových hodnot reprezentovaných jako odstíny šedi, její použití je však jiné. Jak už název napovídá, světelná mapa se používá k osvětlování scény. Jednoznačně určuje světlost každého pixelu ve scéně. Pokud je tedy rozsah možných hodnot bodů ve světelné mapě 0-255, každý pixel změní svoji barvu následujícím způsobem:

$$R = R \times \frac{hodnota(x,z)}{255}$$

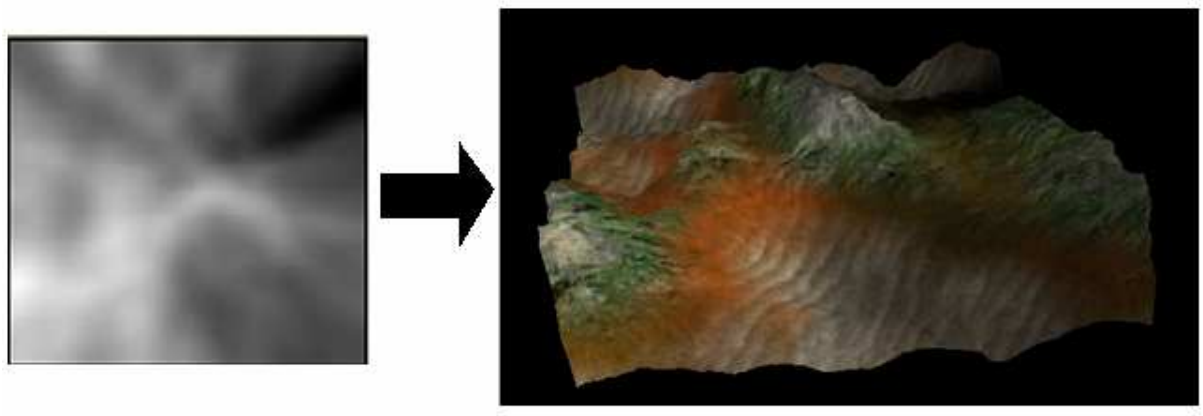
$$G = G \times \frac{hodnota(x,z)}{255}$$

$$B = B \times \frac{hodnota(x,z)}{255}$$

kde R, G a B jsou barevné složky daného pixelu a hodnota(x,z) je funkce vracející hodnoty výšky (neboli souřadnice y) v daném bodě.

Světelná mapa může být generování mnoha různými způsoby, například pomocí různých osvětlovacích modelů. Těmi nejznámnějšími a nejběžnějšími jsou například radiosita, která umí simulovat nepřímé osvětlení na difusních povrzích, nebo ray-traycing, který dokáže simulovat velké množství odrazů světelných fotonů ve scéně. Poté, co je světelná mapa vygenerována v době před-výpočtu scény, je v průběhu zobrazování aplikována na scénu a výrazně tak urychluje renderování.

Světelná mapa může být také generována pomocí stejných algoritmů, jako výšková mapa, tj např. midpoint displacement a simulovat tak např. efekt pronikání slunce přes mraky.
V mé práci budu světelnou mapu generovat jen pomocí algoritmů a aplikovat ji na scénu.
Obrázek 4 ukazuje aplikaci světelné mapy na trojrozměrnou scénu – krajinu. Opět si lze povšimnout, jak hodnota světelné mapy, která je zobrazena jako stupeň šedi, ovlivňuje osvětlení krajiny ve scéně.



Obrázek 4

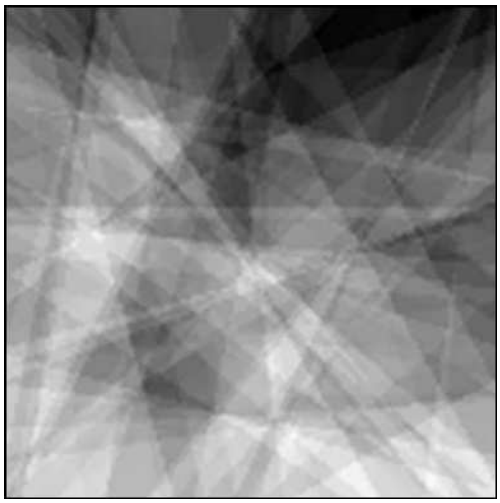
1.2 Algoritmus Fault Formation

V následujících kapitolách mluvím o algoritmech pro generování výškových map. Tyto algoritmy se však se stejným efektem dají použít pro generování map světelných, protože jsou obě založeny na stejném principu, liší se jen účelem použití.

Tento algoritmus je součástí algoritmů na fraktální generování terénu. Základní algoritmus vypadá následovně:

1) Vybereme dva náhodné body na okrajích výškové mapy.
2) Pokud těmito body vedeme pomyslnou přímku, rozdělíme výškovou mapu na dvě části.
3) Na jedné části výškové mapy přidáme hodnotu výšky všem bodům patřícím do této oblasti.

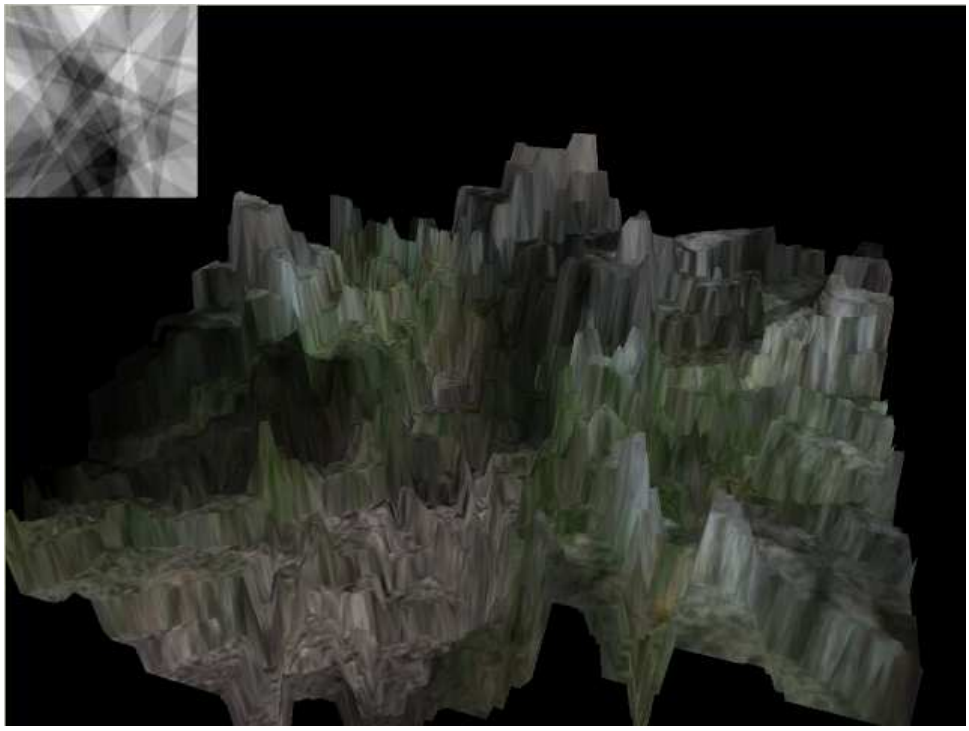
Takto provedeme několik iterací tohoto algoritmu. Obrázek 4 ukazuje aplikaci výškové mapy vygenerované právě algoritmem fault formation s 64 iteracemi a navíc s aplikací erozního filteru, který si popíšeme v další podkapitole.
Kromě tohoto jednoduchého postupu musíme uvažovat také další faktory. Jedním z nich je fakt, že pokud generujeme naprosto náhodné hodnoty výšky, světlá a tmavá místa nemají žádnou návaznost, žádný vzor, jsou jen náhodně rozprostřena v prostoru. Toto můžete vidět na obrázku 5. Aby fault formation generoval terén, který bude vypada více realisticky, provedeme jednoduše to, že hodnota, o kterou se bude výška jedné strany mapy každý krok iterace algoritmu zvyšovat se bude postupně snižovat. Konkrétní hodnota může být parametrizována čímž lze vytvořit různé druhy terénu pro různé použití.
Vzhledem k tomu, že jsem si tento algoritmus vybral na generování výškových a světelných map do svého programu, uvedu v kapitole Implementace implementační detaily, které nejsou při vysvětlování konceptu tolik důležité.



Obrázek 5

1.3 Erozní filter

Erozní filter je dalším faktorem, který musíme na výškovou mapu vygenerovanou algoritmem fault formation aplikovat. Na Obrázku 5 lze vidět, že výšková mapa vygenerována tímto algoritmem má jeden velký nedostatek: rozdíly mezi jednotlivými výškovými stupni, které vznikají jak mapu dělíme na dvě části a na jedné straně přidáváme výšku, jsou moc skokové. Na Obrázku 6 můžeme vidět jaký terén vytvořila výšková mapa nacházející se v rohu okna, která byla vygenerování algoritmem fault formation bez erozního filteru.

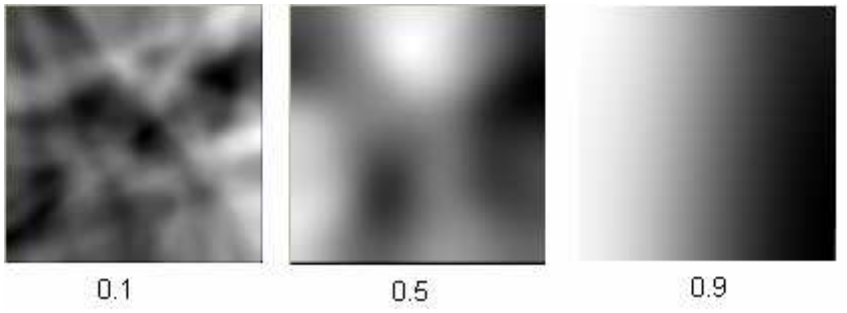


Obrázek 6

Tento nedostatek odstraňuje právě erozní filter, který „zahlazuje“ terén, aby vypadal více hladce a realističtěji. Erozní filter značně připomíná funkci „blur“ v programech pracujících s rastrovou grafikou.
Nejjednodušším a nejúčinějším způsobem, jak erozní filter implementovat je jako FIR filter. Ten převádí posloupnost x_1, x_2, x_3 … x_n na posloupnost y_1, y_2, y_3 … y_n podle následujícího vzorce:

$$y_i = ky_{i-1} + (1-k)x_i$$

kde k je filtrovací konstanta mezi 0 a 1. Menší k znamená menší erozi, větší znamená větší erozi. Na obrázku 7 můžete vidět jaký efekt mají různé hodnoty k na rozmazání terénu a vytvoření plynulosti přechodu mezi jednotlivými výškovými hodnotami.



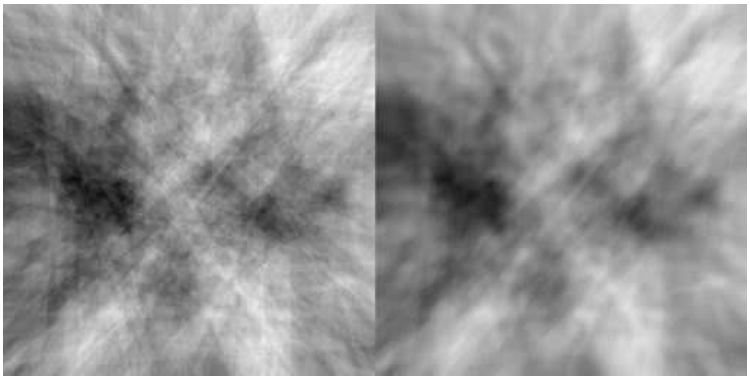
Obrázek 7

Rovnici uvedenou výše aplikujeme nejdříve na všechny řádky, které procházíme nejdříve zprava doleva a poté zleva doprava a poté na všechny sloupce, které procházíme nejdříve zeshora dolu a poté zdola nahoru.

Algoritmus fault formation se používá ke generování terénních úkazů jako jsou svahy, stolové hory a pobřežní útesy, které vzniká takovými přírodními efekty, jako jsou sesuvy půdy v důsledku gravitace, pohybu tektonických desek a pobřežní eroze.

1.4 Vylepšení Algoritmu Fault Formation

V [3] je uvedeno vylepšení algoritmu fault formation. Autor jednak zrychluje algoritmus a také opravuje jeden nedostatek: při zhruba 1000 iteracích již značně převažují světlá místa nad tmavými, jak ukazuje obrázek 8. Ve výškové mapě napravo je verze s aplikovaným erozním filterem, který ovšem situaci nijak nezlepšuje.



Obrázek 8

Autor tedy přichází s jednoduchým a účinným zlepšením: při provádění algoritmu se nejen hodnoty na jedné straně výškové mapy zvýší, ale také se hodnoty na druhé straně sníží a to podle následujícího kritéria:

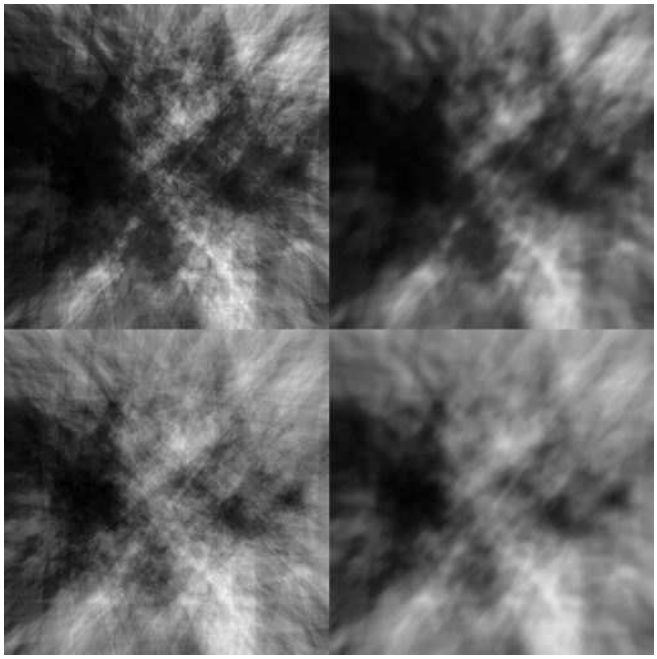
pokud je pixel na levé straně

zvyš jeho výškovou hodnotu o h

jinak pokud je na pravé straně

snižjeho výškovou hodnotu o h*dec

kde dec je konstanta, jejíž parametrizováním lze dosáhnout různé druhy výškových map, jak je ukázáno na Obrázku 9.



Obrázek 9

Mapa vlevo nahoře je vygenerována 1000 iteracemi a konstantou dec = 0.1. Mapa vlevo dole je také vygenerování 1000 iteracemi avšak konstantou dec = 0.01. V pravém sloupci jsou verze příslušných map s aplikovanými erozními filtry. Pokud se použije hodnota dec = 0.0, algoritmus se chová jako původní a žádnou hodnou pixelům na druhé straně rozdělující přímky neubírá. Toto generování je podle orignálního dokumentu [2], kde autor popisuje generování pomocí přidávání hodnot na jednu stranu a ubírání hodnot na druhé straně. Naprostá většina implementací algoritmu fault formation však vychází z [1], kde Jason Shankel pouze hodnoty na jedné straně přičítá.

1.5 Algoritmus Midpoint Displacement

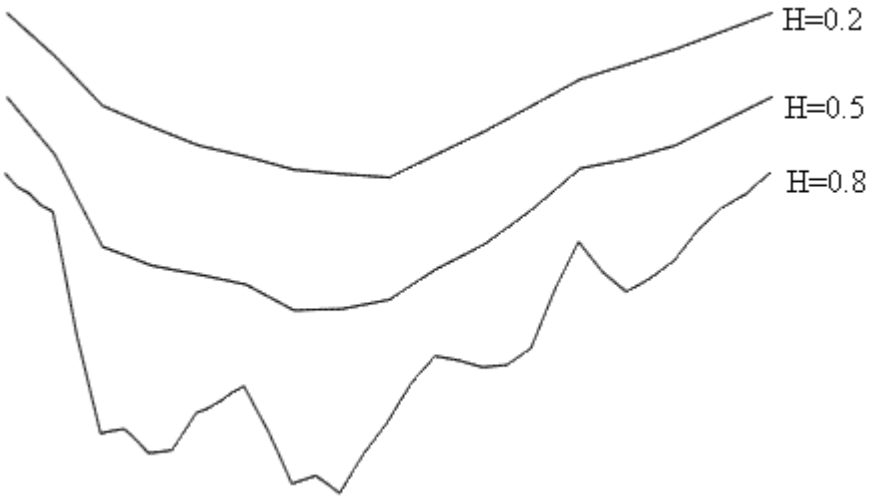
Tento algoritmus je znám také pod názvy jako diamond-square nebo plasma fractal algoritmus. Jeho základní myšlenka vyjádřená ve 2D prostoru je jednoduchá. Pro úsečku AB změním polohu bodu nacházejícím se v jejím středu o nějakou hodnotu. Většinou to bývá hodnota v rozsahu

-height/2 až height/2

kde height je délka úsečky AB. Nyní mám rozdělenou úsečku na dvě části. Opakuji postup pro obě nově vytvořené části, ale předtím snížím hodnotu height následujícím způsobem:

height = height $*2^{-r}$

kde r je hodnota hrubosti. Různé nastavení této hodnoty vedou k různým výsledkům v generování terénu. Obrázek 10 ukazuje vygenerované 2D „terény“ s různými hodnotami H, kde $H = 2^{-r}$.



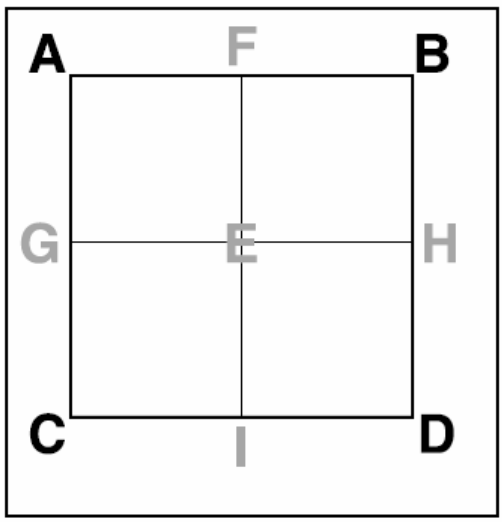
Obrázek 10

Je jasně vidět, že čím vyšší hodnoty r, tím nižší hodnoty H a tím více budeme s každým krokem snižovat rozsah hodnot ve kterých se střed úseček může posunout. Algoritmus pro generování hodnot do trojrozměrného světa a tedy výškové mapy je velmi podobný.
Základem je čtverec ABCD, neboli dvourozměrné pole bodů. Jedno z omezeních tohoto algoritmu je velikost mapy, ve které se generují hodnoty. Mělo by být vždy 2^n. Důvod, proč to tak je, vyplyne z následujícího výkladu principu generování výškových hodnot.

Tomuto algoritmu se mimojiné říká Diamond-Square algoritmus z toho důvodu, že každá iterace se skládá ze dvou hlavních kroků:
1) kroku Diamond, ve kterém se vezmou body vrcholů čtverce a v průsečíku diagonál se změní výška bodu následujícím způsobem:

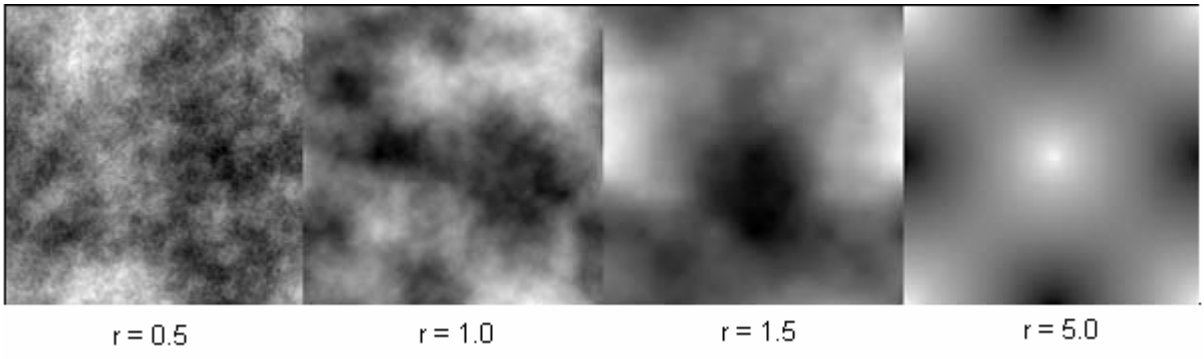
E = (A+B+C+D)/4 + random(-height/2,+height/2)

kde je E je bod v průsečíku diagonál, ABCD jsou vrcholy čtverce a random je funkce, která vrátí náhodné číslo v rozsahu -height/2 až height/2. Názorněji to je vidět na Obrázku 11:



Obrázek 11

2) Druhým krokem je krok Square. Ten slouží k vygenerování hodnot v bodech F, G, H a I. Vygeneruje se podobně jako bod E: vezmou se body z přilehlých vrcholů (AB) a „midpointů“ (E a v případě větší mapy hodnota na druhé straně) a z těch se udělá průměr. K tomu se přičte náhodná hodnota v rozsahu opět -height/2 až height/2. Hodnota height se musí stejně jako u 2D verze s každým krokem snižovat podle koeficientu r. Takto projdeme celou výškovou mapu, až se dostaneme na nejmenší elementární(tedy dále nerozdělitelný) čtverec, který nemá žádný střed. Pro tento algoritmus můžeme používat buď hodnoty float, nicméně potom zabírá taková výšková mapa v paměti 256kB za předpokladu, že proměnná float má velikost 32 bitů. Pokud chceme ušetřit místo, použijeme char a potom již je opravdu nutné, aby byla výšková mapa velikosti 2^n z důvodu omezeného rozsahu proměnné char (jen 256 hodnot).
Obrázek 12 ukazuje různé výškové mapy vygenerované s různými hodnotami hrubosti.



Obrázek 12

Horské masivy jako jsou například Himaláje nebo Grand Canyon byly utvořeny geologickým procesem zdvihu, který značí jev, kdy na sebe dvě desky zemské kůry tlačí a vytvářejí tak horské hřebeny. Algoritmus Midpoint Discplacement simuluje právě tyto přírodní jevy. Lze ho implementovak velmi elegantním způsobem rekurzivně, nicméně iterativní implementace je vhodnější, neboť při rekurzivní průchodu nebudu mít pro krok square všechny 4 hodnoty, které jsou

potřeba pro vytvoření bodů ve středu úseček stran čtverce. Je proto lepší vytvořit iterativní verzi algoritmu, ve které si nejdříve spočítám všechny „diamond“ kroky a teprve poté přistoupím k počítání „square“ kroků. Zde je implementace rekurzivní verze v pseudokódu:

Proveď krok „diamond“.
Proveď krok „sqaure“.
Zmenši rozsah pro náhodné generování čísel.
Zavolej sám sebe 4x.

A zde je implementace iterativní verze v pseudokódu:

Dokud je délka stran čtverce vetší nez 0{
Projdi pole a proveď krok „diamond“ pro každý obsažený čtverec.
Projdi pole a proveď krok „square“ pro každý obsažený čtverec.
Zmenši rozsah pro náhodné generování čísel.
}

2 Texturování terénu

Texturování terénu je jeden ze způsobů, jak dodat vykreslené krajině detail a silněji tak simulovat realitu. Jedná se v podstatě o nanášení určitého vzoru na nějaké těleso, v našem případě na vygenerovaný terén. Krajina, kterou jsme generovali doteď měla barvy určené podle světelné mapy a nevypadala zdaleka tak věrohodně a nepřibližovala se svým vzhledem realitě (Obrázek 3).

Dva základní druhy textur jsou obrázkové a procedurální. Obrázkové textury jsou nahrané ze souboru a mají tu výhodu, že lze např. pořídit fotografii nějakého reálného místa ve vysokém rozlišení a tu potom v počítači na daný objekt, popř. krajinu aplikovat pomocí grafického API tak, že objekt vykreslený v počítači působí velmi realistickým dojmem. Obrázkovém textury mají nevýhodu v tom, že pokud jsou ve velkém rozlišení, zabírají poměrně hodně místa v paměti. Naproti tomu procedurální textury jsou generované podle algoritmu a tak zabírají mnohem méně místa v paměti počítače. Takové texury ovšem nelze generovat na všechny objekty, někdy se prostě musí použít obrázkové textury. Kombinace těchto dvou postupů spočívá v nahrání texturového vzoru, neboli menší textury do počítače a potom jeho opakovaného nanášení, popřípadě kombinování s jinými vzory. Tento postup používám i ve svém programu a detailněji ho popíšu v podkapitole procedurální generování textur.

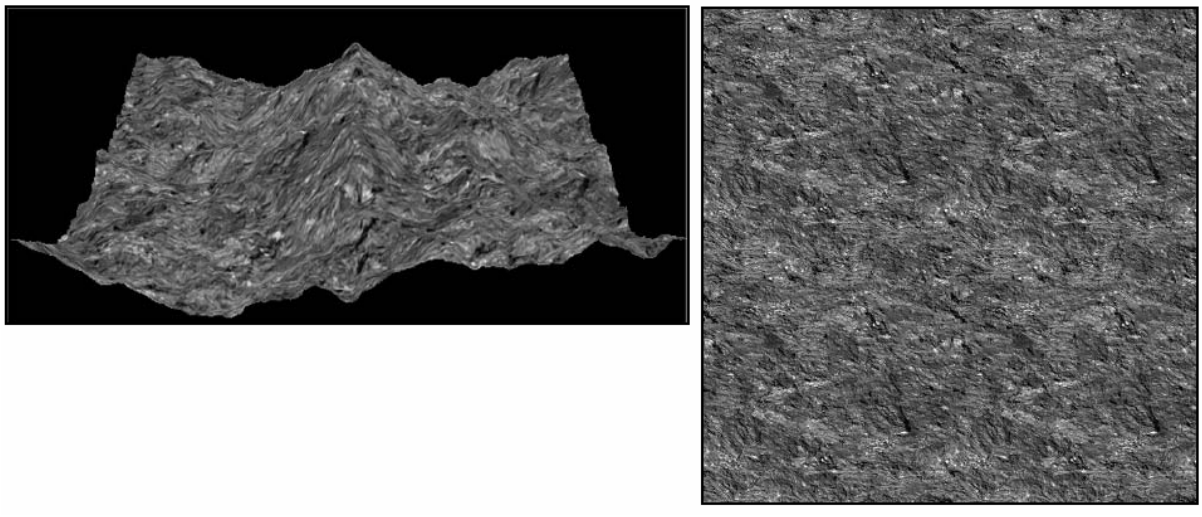
2.1 Druhy textur a způsoby texturování

Textur může být více druhů. Například se světelnou mapou popsanou v předchozí kapitole se zachází jako s texturou, aby obstarávala osvětlení scény. Jeden z dalších druhů textur se nazývá diffuse textura. Je to textura, která má kromě základních barevných kanálů RGB také alpha kanál A, který určuje její průhlednost. Occlusion textura určuje, jak moc povrch pohlcuje v jednotlivých bodech světlo. Používá se zlepšení vnímavosti nerovností povrchu. Je ještě mnoho dalších druhů textur a všechny se nanášejí formou multitexturingu, což je technika, kterou podporuje naprostá většina moderních grafických karet a tím pádem nezpomaluje vykreslování scény.

2.2 Procedurální generování textur

V mém demonstračním programu, který vykresluje krajinu používám aplikování jedné textury na celý terén. Tato technika většinou vytváří dosti špatné výsledky, jak je ostatně vidět na Obrázku 13. Vlevo je terén vygenerový s texturou vpravo. Je jasné, že takovýto terén je již sice lepší než ten úplně bez textury, nicméně pořád nevypadá moc realisticky. Bylo by dobré přidat do terénu reálné části vyskytující se v přírodě, jako skály, tráva, kameny, hlína, sníh a také nějak odlišit vysoké místa od

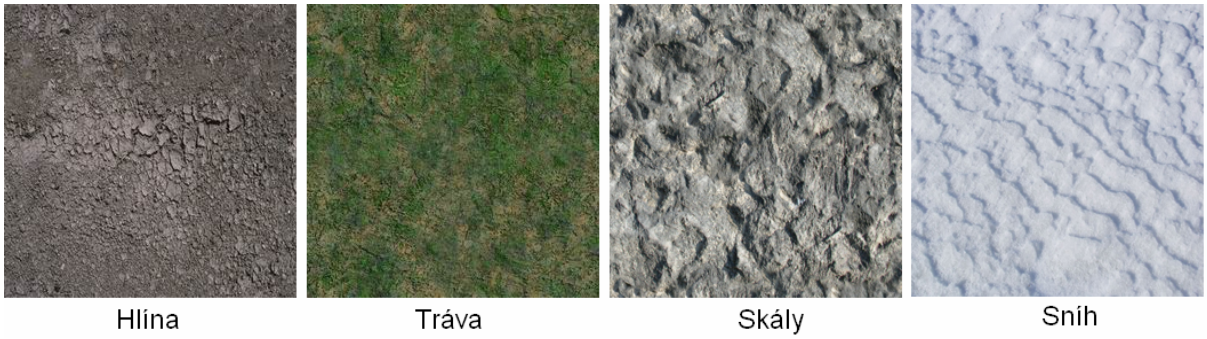
nízkých. Máme možnost nahrát do počítače už předem vytvořenou texturu se všemi těmito terénními atributy (například takovou, které je na obrázku 14) a tu na terén aplikovat.



Obrázek 13

Obrázek 14

Taková textura může být již velmi kvalitní, ale je tu jeden zásadní problém. Pokud použijeme tento postup, nemůžeme za běhu dynamicky generovat výškovou mapu a tím pádem samotný terén, protože místa na kterých se vyskutují jednotlivé povrchy už nebudou výškově odpovídat, a to je právě to, co my chceme. Proto musím použít procedurální vytváření textur za chodu. To funguje tak, že do počítače načteme několik základních texturových vzorů pro různé výškové úrovně a z nich poté složíme texturu pro celý terén. Vzory můžete vidět na Obrázku 15. Textura hlíny představuje texturu nanášenou v nejmenší výškové úrovni. Jak se výška zvyšuje, přecházíme plynule přes trávu, skály až ke sněhu. Právě plynulé přecházení je klíčovým prvkem celého algoritmu pro procedurální generování textur.

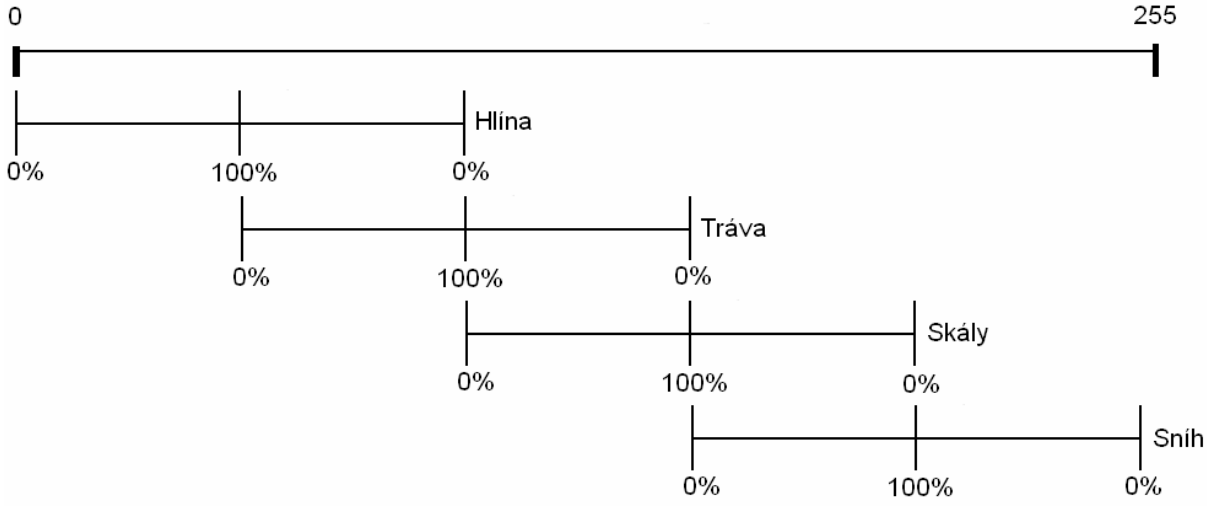


Obrázek 15

Prvním krokem pro plynulý přechod mezi texturami reprezentující různé výškové úrovně je určení míry výskytu vzoru v konkrétních výškách. Pro každou texturu se musí nastavit tři parametry:

1) Nejnižší výška - v této výšce se již textura nebude vyskytovat – míra výskytu 0%
2) Optimální výška – v této výšce se textura bude vyskytovat na 100%
3) Nejvyšší výška - v této výšce se již textura nebude vyskytovat – míra výskytu 0%

Hodnoty výšky se určují podle hodnot výškové mapy. Jsou to tedy hodnoty od 0 do 255.
Obrázek 16 ukazuje, jak se zastoupení pro jednotlivé vzory v tomto rozsahu určí. Máme-li zastoupení, můžeme přistoupit k samotnému generování textury pro náš terén. Postup je jasný: procházíme po řádcích výškovou mapu a pro každý bod spočteme RGB hodnoty podle procentuálního zastoupení která jsme právě určili.
Např: Narazíme na bod, který má výšku 80. Nyní projdeme všechny vzorky a pro každý zjistíme jeho barevnou složku. Postupujeme, jakobychom chtěli na celý terén aplikovat všechny 4 textury najednou. Tedy vezmeme bod, který zkoumame ve výškové mapě a podle jeho souřadnic spočítáme, nebo pokud máme textury stejně velké jako výškovou mapu přímo vezmeme, souřadnice pro texturu.

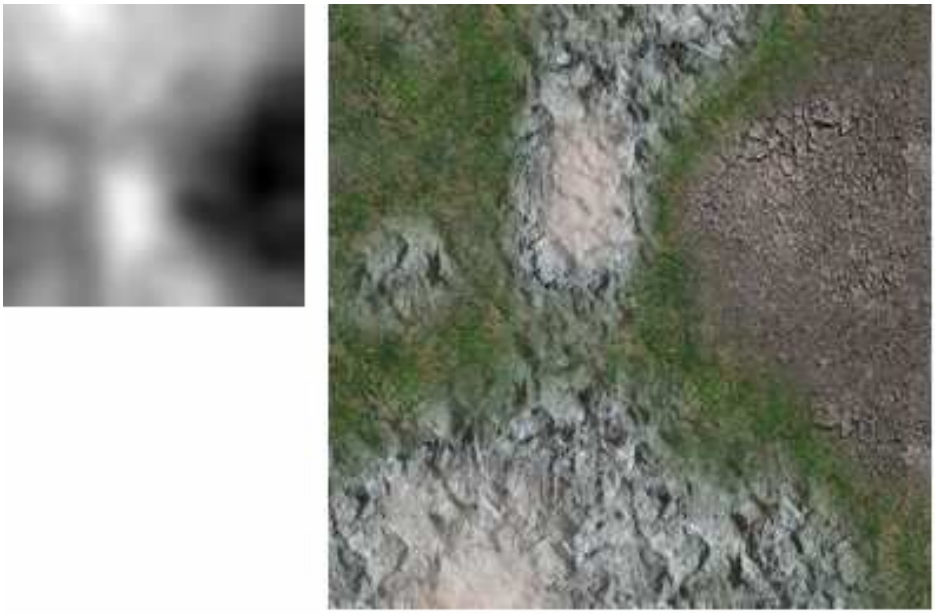


Obrázek 16

V bodě, který je určený těmito souřadnicemi vezmeme všechny barevné složky pixelů všech textur a podle procentuálního zastoupení určíme příspěvek výslednému pixelu. Tedy s bodem s výškou 80 máme zastoupení hlíny např. 65% a zastoupení trávy 35%. Barvu výsledného pixelu tedy spočítáme následovně:

$R_v = 0.65*R_h + 0.35*R_t$
$G_v = 0.65*G_h + 0.35*G_t$
$B_v = 0.65*B_h + 0.35*B_t$

Tímto způsobem projdeme všechny řádky výškové mapy a spočítáme pro ně barvu textury. Texturu uloženou v paměti poté převedeme pomocí OpenGL funkce na texturu a tu poté vyrenderujeme. Obrázek 17 ukazuje výškovou mapu vygenerovanou pomocí algoritmu fault formation a podle této výškové mapy procedurálně vygenerovanou texturu o rozlišení 512x512 pixelů.



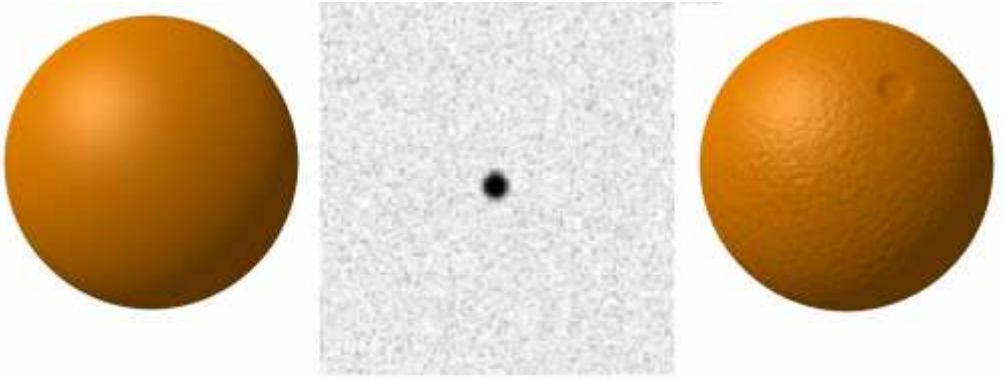
Obrázek 17

Obrázek 18

Za povšimnutí zejména stojí přechody mezi jednotlivými výškovými úrovněmi, které je detailně ukázáno na obrázku 18, kde můžeme vidět přechod mezi vzorkem hlíny a trávy.

2.3 Metody pro zlepšení detailu textur

Existuje několik metod, jak zlepšit realističnost nanášených textur. Jedním z nich je Bump-mapping. Bump-mapping je jedna z prvních zásadnějších metod používaných pro zvýšení detailu a přestože je již poměrně stará (výrazně s používá od roku 2001, kdy grafické karty začaly hardwarově podporovat její počítání) dodnes se s úspěchem používá. Je to metoda založená na aplikaci speciální mapy, která v každém pixelu mění hodnotu normálového vektoru a simuluje tak lehce vystouplý povrch pomocí změněného osvětlení a bez zvýšení počtu vykreslovaných polygonů, což by zapříčinilo zpomalení vykreslování v reálném čase. Na obrázku 19 můžete vidět úplně vlevo těleso bez bump-mapy, uprostřed aplikovanou bump-mapu (což je v rastr 8bitových hodnot, jako světelné a výškové mapy) a vpravo je osvětlené těleso s aplikovanou bump – mapou.

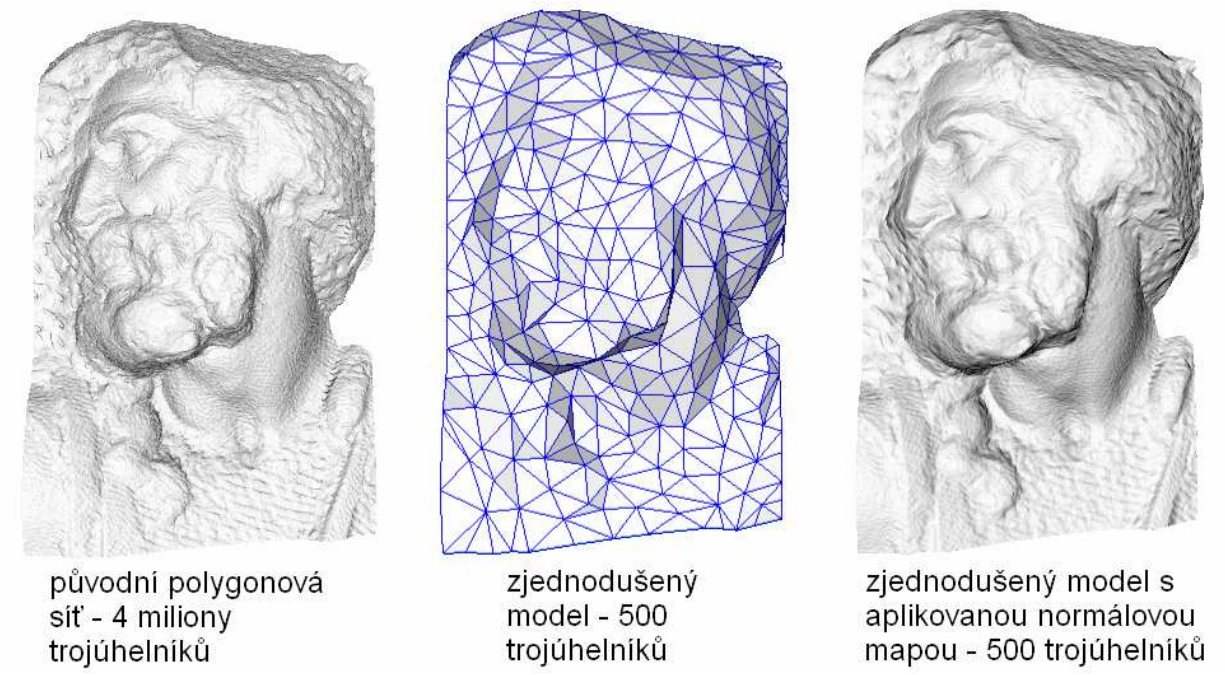


Obrázek 19

Tato technika má výhodu v tom, že je podporována na naprosté většině grafických karet a tak nijak nezpomaluje vykreslování a vytváří efektní výsledky. Nevýhoda této techniky tkví v tom, že pomocí ní nelze simulovat příliš velké „výčnělky“, protože nedeformuje geometrický model objektu a při velkých převýšeních nevrhá stíny sama na sebe, což ve výsledků působí nepříliš realistickým dojemem.

Dalšími technikami na zvýšením detailu jsou například normálové mapování, což je technika v podstatě stejná jako bump-mapping, jen s tím rozdílem, že normály pixelů neposouvá, ale určuje jejich absolutní hodnotu. Prakticky se používá tak, že je vytvořen vysokopolygonový model v některém z modelovacích programů (např. Blender), ze kterého je následné automaticky odvozen jak nízkopolygonový model, tak normálová mapa, která se na tento model bude ve vykreslování v reálném čase používat. Výsledný efekt je poté téměr stejný, jakoby se použil vysokopolygonový

model, ale vykreslování je několikanásobně rychlejší. Obrázek 20 ukazuje názorný příklad aplikace normálové mapy.



Obrázek 20

2.4 Detailní mapování

Právě popsané techniky přidávají dost na realističnosti, ale používájí se převážně na jednodušší a menší modely než je krajina. Existují sice enginy, které používají bump-mapping na přidání detailu terénu, ty ovšem vyžadují náročnější implementaci a mimojiné určení bumpmapy a tudíž i normály pro každý druh povrchu. Detailní mapování je však velmi jednoduchá technika, která je poměrně jednoduchá na implementaci a zajišťuje značný nárůst detailu krajiny. Její princip spočívá v tom, že se kromě standartní textury krajiny aplikuje také detailní textura o vyšším rozlišení simulující různé trhliny, nerovnosti a vystouplá místa v krajině. Obrázek 21 ukazuje příklad takové detailní mapy.

Obrázek 21

Při aplikaci lze použít více buď více texturovacích jednotek, pokud jime grafická karta disponuje, jinak je nutno provést více renderovacích cyklů a efekt tak simulovat softwarově. Implementační detaily budou popsány v další kapitole Implementace.

3 Implementace

Pro implementaci demostračního programu pro tuto bakalářskou práci jsem si vybral jazyk C++, protože je velmi rozšířený nejen na simulaci grafických efektů pro jeho rychlost, ale také obecně pro psaní jakýchkoliv grafických enginů. Jako API pro práci ve 3D jsem zvolil OpenGL.

3.1 OpenGL

OpenGL je multiplatformní nízkoúrovňová knihovna pro práci nejen s 3D, ale i s 2D grafikou. Jejímu vzniku v roce 1992 v Silicon Graphics předcházely výzkumy v oblasti grafického modelování a realistické syntézy obrazu. Do doby vzniku se stala široce uznávaným standartem na poli grafických a multimediálních aplikací, CAD systémů a tvorby her. Využívá jí značné množství komerčních programů, například modelovací systémy jako jsou 3D Studio MAX a Maya, CAD systémy jako Microstation, nebo počítačové hry od společnosti id Software jako série Quake, či Doom 3. Hlavní výhoda OpenGL jako je API je, že dokáže před programátorem skrýt konkrétní volání služeb hardwaru, o kterém tak programátor nemusí nic vědět a přesto ho plnohodnotně využívat. OpenGL se dá používat teoreticky v jakémkoliv programovacím jazyce, ovšem specifikace je psaná pro C/C++ a proto je nejlepší psát programy, které toto rozhraní využívají právě v tomto jazyce.

OpenGL se skláda asi z 250 funkcí, které jsou rozděleny do několika knihoven:

1) GL – základní knihovna OpenGL, která obsahuje volání funkcí na vykreslení základních útvarů jako jsou body, přímky, trojúhelníky, kvádry, koule a podobně.

2) GLU – v této knihovně jsou zapouzdřeny složitější postupy převáděné na jednotlivá volání funkcí. Mezi tyto postupy patří například nastavení ortogonální či perspektivní kamery, práce s kvadrikami, tesselátory, NURBS křivkami apod.
3) GLUT – GLUT je zkratka GL Utility Toolkit. Protože základní knihovna OpenGL je vytvořena tak, aby ji bylo možné provozovat nezávisle na hardwaru a operačním systému, nejsou v ní obsaženy žádné funkce pro práci s okny (rušení, změna velikosti, otevírání), pro zpracování událostí, nebo pro vytvoření grafického uživatelského rozhraní. Knihovna GLUT řeší právě tyto problémy. Definuje a implementuje aplikační rozhraní pro tvorbu oken a uživatelského rozhraní. Obsahuje také funkce pro vykreslování bitmapového a vektorového písma.

OpenGL nepodporuje objektové programování, obsahuje pouze funkce, které mění vnitřní proměnné, matice a zásobníky a ke kterým není možno přímo přistupovat. Připomína tak princip stavového automatu, mimojiné také z důvodu, že proměnné (například barva pixelu) si uchovávají svojí hodonotu, dokud je programátor nezmění. Pomocí tohoto přístupu lze realizovat architekturu klient-server, kdy klientem je uživatelská aplikace, která posílá požadavky na změny vnitřních datových struktur právě OpenGL. Tohoto lze využít i pro situaci, kdy náročný výpočet probíhá na jiném počítači a výstup se posílá na terminální počítač přes síť.

3.2 Hlavní soubor main.cpp

Demostrační program pro zobrazování krajiny využívá rozhraní WinAPI, pomocí kterého vytváří okna a zachycuje zprávy od klávesnice a myši. Program tedy začíná funkcí WinMain v souboru main.cpp, ve které proběhne vytvoření okna pomocí funkce CreateGLWindow.
V této funkci se kromě vytvoření okna nastavuje pixelformat, vytváří se GL device kontext, který slouží pro spojení okna vytvořeného ve WinAPI s OpenGL a zabezpečuje vykreslování do něj. Nakonec se v této funkci zjišťuje, zda grafická karta, se kterou bude OpenGL pracovat, podporuje multitexturing (kvůli detailnímu mapování – hardwarový mutlitexturing značně urychluje vykreslování). To se provádí prostřednictvním funkcí z knihovny glext.h, pomocí kterých je možné přistoupit k informacím o grafické kartě a zjisti tak různá rozšíření, mezi nimi i multitexturing.

Poté se v nekonečné smyčce nejdříve zpracovávají příchozí zprávy od okenního systému, které se posílají funkci WndProc. Dále se zavolá funkce DrawGLScene, ve které již probíhá kompletní vykreslování. V té se nejdříve pomocí funkce glClear vyčistí hloubkový a obrazovkový buffer. Poté se volá funkce na vykreslení minimap reprezentujících výškovou a světelnou mapu použitou v aktuální scéně. Tato funkce projde všemi body příslušných map a vykreslí je na příslušnou pozici v okně pomocí funkce glVertex2d. Dále se ve vykreslovací funkci nastaví kamera.

Poté se vstoupí do renderovací funkce, která ověří, zda je nastaveno detailní mapování, texturové mapování a zda je možné provádět multitexturing a podle toho již funkce začne vykreslovat scénu. Pokud je povolen multitexturing, je nutné nastavit klasickou texturu na jednu textorovací

jednotku a detailní texturu na druhou. Po vykreslení všech trojúhelníků je nutné všechny textury z texturovací jednotky zase uvolnit, jinak by mohly ovlivňovat další vykreslování, např. mininap výškové a světelné mapy.

3.3 Třída Map

Třída map je obecná třída pro práci s mapou, ať už světelnou, či výškovou. Obsahuje samotná data mapy, což je jednoduše pole proměnných typu char, velikost pole a dále metody pro obecnou práci s mapou, jako jsou:

1) faultFormationMapGen: Tato metoda provádí algoritmus fault formation jak je popsán v kapitole 2.2. Nejdříve si alokuje paměť pro vygenerovaná data a poté provádí standartní algoritmus této metody. Pomocí funkce rand z knihovny math.h vygeneruje náhodné body a určí jejich vektory. Poté v cyklu prochází všechny body a podle x a z pozice upravuje jejich hodnotu. Nakonec aplikuje erozní filter. Toto opakuje pro daný počet iteraci, v našem případě je to napevno nastavená hodnota 64. Po vygenerování všech hodnot je musíme převést do rozsahu 0-255, což jsou jediné možné hodnoty, které typ unsigned char přijímá. Nakonec hodnoty z dočasného bufferu nahrajeme do třídního pole určeného pro data mapy a uvolníme paměť dočasného bufferu.

2) erodeArray: Metoda projde dvourozměrné pole hodnot a aplikuje na ní erozní filtr. Za zmínku stojí proměnna stride, která určuje velikost kroku a pomocí které lze dosahovat různých variací tohoto filtru.

3) erodeMap: Aplikuje funkci erodeArray různými směry podle popisu algoritmu, tedy nejdříve zprava doleva, zleva doprava, zdola nahoru a nakonec shora dolu.

4) setTrueRangeValues: převede předaná data do rozsahu 0-255, tedy takového, se kterým se dále pracuje ve třídním poli hodnot mapy.

3.4 Třída Terrain

Třída Terrian obsahuje výčtový typ pro druhy povrchů, strukturu vzoru textury obsahující hodnoty výšek pro procentuální výskyt povrchu, dále dvě proměnné třídy Texture, která bude popsána dále, jednu pro texturu povrchu a druhou pro detailní texturu a tyto metody pracující s texturami:

1) getTextureCoords: tato metoda převádí souřadnice z textury krajiny do souřadnice v textuře vzorku. Nejdříve zjistí, zda je souřadnice větší než velikost vzorku a poté zjistí, kolikrát se vzor může opakovat, než získáme souřadnici v rámci textury vzorky. Potom tuto souřadnici nastaví do ukazatelů na typ int, které byly předány jako parametry.

2) RegionPercent: metoda přebírá typ textury a výšku a vrací číslo typu float v rozmezí 0.0 až 1.0, které značí procentuální zastoupení daného vzoru v dodané výšce. Metoda ověří několik krajních případů jako např. předaná výška je mimo, nebo výška přesně odpovídá optimální výšce vzoru (v tom

případě vrací 1.0) a pokud je výška někde v rozsahu, musí odpovídající procentuální zastoupení spočítat.

3) generateTerrainTexture: hlavní funkce, která zajišťuje procedurální generování textury krajiny. Na začátku ověříme, zda jsou všechny textury úspěšně nahrané. Dále spočítáme hranice výšek pro všechny vzorky povrchu a zjistíme poměr textura ku výšková mapa. Nyní procházíme celou texturu a pro každý bod zjistíme jednak výšku ve výškové mapě a jednak barevné složky bodů ze všech vzorků. Pro každý vzorek zjistíme pomocí funkce RegionPercent procentuální zastoupení v dané výšce, kterým potom zredukujeme příslušnou barevnou složku. Barevné složky všech vzorků se sčítají v proměnných sumRed, sumGreen a sumBlue a nakonci cyklu se vloží do pole hodnot textury. Poslední věc, co je potřeba v této metodě udělat, je vytvořit texturu tak, aby s ní mohlo OpenGL pracovat. To se provede pomocí následující sekvence příkazů:

- glGenTextures vygeneruje unikátní identifikátor textury.

- pomocí glBindTexture řekneme OpenGL, že právě vygenerovaný identifikátor bude 2D textura

- metoda glTexParameterf říká, co má OpenGl s texturou dělat, pokud ji bude zmenšovat nebo roztahovat v případě, že rozlišení textury bude jiné než rozlišení výškové mapy. Parametr GL_LINEAR říká, že pro spočítání barvy vytvořeného bodu při zmenšení nebo při zvětšení se vezmou 4 pixely které jsou danému pixelu nejblíže.

- glTexImage2D provede další nastavení textury a její vygenerování z dvourozměrného pole usigned char hodnot.

3.5 Třída Texture

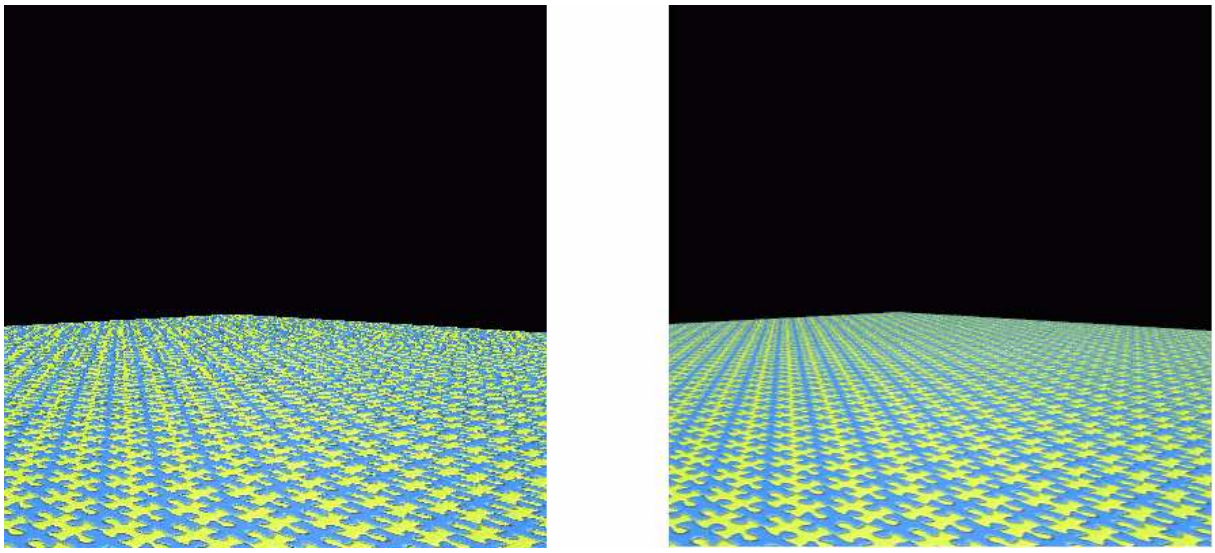
Třída texture slouží k nahrání obrázku typu BMP do paměti počítače a do třídního dvojrozměrného pole hodnot. Kromě toho obsahuje chráněné proměnné indikující rozměr textury, její barevnou hloubku, identifikační hodnotu typu int používánou při aplikaci textury pomocí OpenGL a metody pro získávání a nastavování těcto proměnných. Další metody, které tato třída obsahuje jsou:

1) Create: Alokuje paměť pro texturu předané velikosti a barevné hloubky.

2) LoadData: Tato metoda má za úkol nahrát obrázek z BMP souboru do paměti ovšem stará se jen o otevření souboru a nahrání dat do alokovaného třídního ukazatele. Jeho skutečné nahrání z formátu BMP zabezpečuje metoda LoadBMP.

3) LoadBMP: V této metodě se nahrávají data ve formátu BMP do třídního ukazatele. Nejdříve se extrahují informace o souboru, jako jsou jeho velikost a barevná hloubka a poté se zavolá funkce Create, která podle těchto informací alokuje paměť. Do té poté nahrajeme samotná data BMP souboru bez hlavičky a metainformací a na závěr projdeme všechny pixely a prohodíme barevné složky R a B, protože formát BMP zapisuje body v pořadí BGR.

4) Load: Tato funkce zahrnuje kompletní sestavení textury pro OpenGl. Nejprve se nahrají data do třídního pole RGB hodnot, poté sestavíme texturu způobem ne nepodobným tomu v třídě Terrain a metodě generateTerrainTexture. Kromě toho sestavíme také mipmapové textury, které se používají při velkém oddálení textury, kdy je nutné více pixelů sloučit do jednoho. K tomuto účelů se vytvoří více textur ve více rozlišení, aby se předešlo efektu aliasingu. Příklad aplikace antialiasingu je na obrázku 22. Vlevo je scéna bez antialisingu.



Obrázek 22

Vpravo je s aplikovaným antialisingem za použití textury s vyšším rozlišením.

4 Závěr

4.1 Shrnutí přínosů

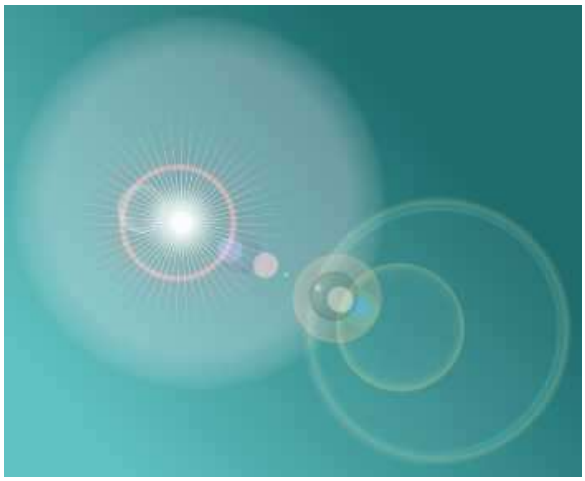
Práce na demostračním programu a této zprávě pro mě byla velmi přínosná. Zdokonalil jsem své programovací dovednosti v jazyce C++ a v rozhraní OpenGL, dozvěděl jsem se, jak vnitřně fungují a jak se implementují efekty, se kterými jsem se dříve setkával v počítačových hrách, jako jsou bump-mapping, antialiasing, dozvěděl jsem se, jak probíhá texturání a řešil jsem několik problémů s viditelností, rychlostí vykreslováním a natáčením scény, aby byl možné interaktivní průlet scénou pomocí otáčení myši. Taktéž jsem zdokonalil své schopnosti v rozhraní WinAPI. Zobrazování 3D grafiky v reálném čase mě programováním této aplikace opravdu začlo velmi bavit a tak v budoucnu určitě zkusím některý z efektů, jako je mlha, voda či déšť do programu přidat a vyzkoušet si tak jejich implementaci.

4.2 Budoucí návaznost

Demostrační program v současné chvíli používá pro zobrazení krajiny algoritmu hrubé síly, které nic neřeší a zobrazuje všechny trojúhelníky. Bylo by zajímavé a účelné implementovat některý z algoritmů, který zobrazuje jednak jen viditelné trojúhelníky a šetří tak výpočetní výkon, jednak přidává detail (více polygonů) na místa, která jsou blíže pozorovateli, nebo která jsou velmi členitá a naopak nezobrazuje přiliš trojúhelníků na místech, který to příliš nepotřebují, jako jsou místa vzdálená od pozorovatele nebo rovné části krajiny. Takové algoritmy jsou například geomipmapping, QuadTree nebo algoritmus ROAM, který umožňuje zobrazit krajinu při nižších detailech i na pomalejších počítačích.

Dalšími rozšířeními do zobrazování krajiny může být možnost generovat výškovu mapu nejenom podle algoritmu fault formation, ale také midpoint discplacement a nebo perlin noise, který tvoří především krajiny kde jsou převážně údolí a vrcholky hor.

Pro přidání realističnosti terénu by bylo zajímavé přidat možnost zobrazení vody, mlhy, oblohy, efekt odlesku (Obrázek 23), zobrazení dalších objektů, jako jsou stromy, keře, kameny, tráva a obecně nízká vegetace, efekty ohně či částicových systémů, nebo deště či sněhu. Toto může být téma například pro diplomovou práci.



Obrázek 23

Literatura

[1] - Shankel, Jason. “Fractal Terrain Generation—Fault Formation.” Game Programming Gems. Rockland, Massachusetts: Charles River Media, 2000. 499–502.

[2] - Robert Krten, Generating Realistic Terrain - http://www.ddj.com/architect/184409269

[3] - Davide Coppola, Fault Formation improved - http://www.m3xbox.com/GPU_blog/?tag=fault-formation

Paul Martz ,Generating Random Fractal Terrain - http://gameprogrammer.com/fractal.html#midpoint

Trent Polack, Game Development Series: Focus on 3D terrain programming, Premier Press, USA, Ohio, 2003, 239 stran, ISBN 1-59200-028-2

Encyklopedie Wikipedia - http://en.wikipedia.org, http://cs.wikipedia.org

Daniel Čech, OpenGL referát na praktikum z informatiky - http://nehe.ceske-hry.cz/cl_gl_referat.pdf

Seznam příloh

Příloha 1. Manuál

Příloha 2. CD se zdrojovými texty programu a projektem pro Microsoft Visual Studio