VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Ročníkový projekt

Petr Vrba

2005

Prohlášení

Prohlašuji, že jsem tento ročníkový projekt vypracoval samostatně pod vedením Ing. Jana Pečivy. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Petr Vrba
8.5.2005

Poděkování

Rád bych tímto poděkoval mému vedoucímu ing. Janu Pečivovi za hodnotné odborné rady a přátelský přístup při řešení Ročníkového projektu.

Abstrakt

Projekt je z oblasti počítačové grafiky, konkrétně generování a zobrazování virtuálních krajin. Základním požadavkem bylo vytvořit krajinu Perlinovou funkcí a zobrazit ji pomocí Chunked-LOD algoritmu. Přiložená aplikace zobrazuje krajinu o vysokém rozlišení v reálném čase. V dokumentu jsou popsány požadované principy a algoritmy. Implementace je realizována pomocí knihovny Open Inventor.

Klíčová slova

virtuální krajina, Perlinova funkce, chunked, LOD, level of detail, terén, reálný čas, Open Inventor

Abstract

This project is focused on generation and visualization of virtual landscape. Terrain is generated by Perlin noise and rendered by chunked-LOD algorithm. This report describe their principles. Aplication show high resolution terrain in real time. For implementation is used Open Inventor library.

Key words

virtual landscape, Perlin noise, chunked, LOD, level of detail, terrain, real time, Open Inventor

Obsah

1 Úvod **5**

2 Generování terénu **6**
2.1 Midpoint Displacement 7
2.2 Fault Formation 7

3 Perlinova funkce **8**
3.1 Princip 8
3.2 Implementace 9

4 Zobrazování terénu **11**
4.1 ROAM algoritmus 11
4.2 Triangle-strip method 11

5 Chunked-LOD **12**
5.1 Princip 12
5.2 Navazování segmentů 13
5.3 Implementace 15

6 Výkonové a vizuální vlastnosti aplikace **17**
6.1 Výkon 17
6.2 Vizuální vlastnosti 18

7 Závěr **19**
7.1 Kde lze projekt využít? 19
7.2 Další možná pokračování projektu 19

A Uživatelský manuál **21**

B Obrázová příloha **22**

Kapitola 1

Úvod

V dnešní době řeší počítačová grafika několik závažných problémů. Jedním z nich je zobrazování krajiny pomocí soudobého hardwaru. Ať již zobrazujeme krajinu v počítačové hře nebo v geodetickém simulátoru, řešíme stále stejný problém. Při vykreslování terénu je hlavním důvodem malého výkonu obrovské množství dat, které musí grafická karta a procesor zpracovat. Proto se využívá nejrůznějších postupů, jak množství dat snížit. Jedná se například o ROAM algoritmus, chunked-LOD a další.

Požadované vlastnosti zobrazované krajiny jsou různé. V simulačních aplikacích vyžadujeme co nejvěrnější ztvárnění krajiny a na rychlosti simulace nám prioritně nezáleží. Naproti tomu v jiných případech (např. počítačové hry) usilujeme především o dokonalou real-timovou aplikaci a některé aspekty detailu jsme ochotni oželet.

V následujících kapitolách postupně objasním způsob generování náhodné krajiny Perlinovou funkcí a zobrazování terénu chunked-LOD algoritmem.

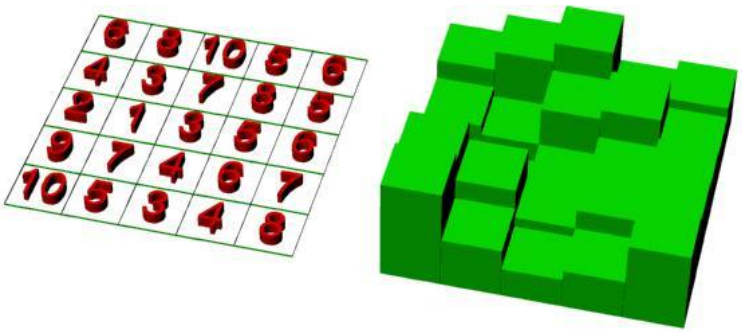
Poté rozvedu téma o implementaci zadaného problému pomocí knihovny Open Inventor. Nastíním také různé překážky, které při implementaci nastaly a následně i jejich řešení.

V závěru zprávy zhodnotím výkonové a vizuální vlastnosti obou algoritmů a také uvedu možnosti budoucího rozvoje tohoto projektu.

Kapitola 2

Generování terénu

Pokud chceme zobrazit zemský povrch, musíme si určitým způsobem uchovat informace o jeho tvaru. K tomuto účelu se hojně využívají výškové mapy. Výškovou mapou rozumíme matici hodnot, které nám definují výšku terénu v místě daném souřadnicemi (viz. obrázek 2.1). Z takto připravených dat lze již snadno vytvořit trojrozměrný model krajiny.

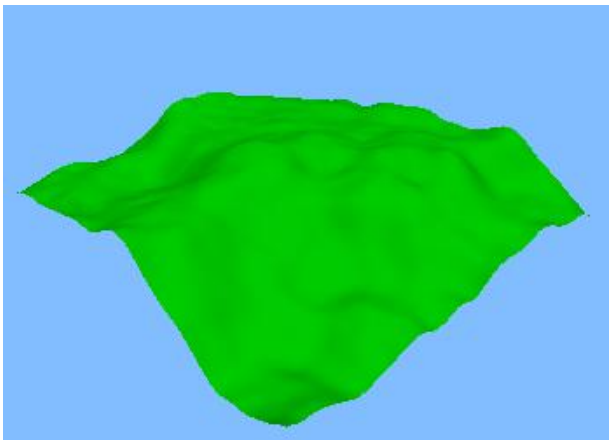


Obrázek 2.1: Příklad výškové mapy a 3D modelu

V případě, že zobrazujeme existující povrch, výškovou mapu máme již k dispozici (například z družicových snímků). Pokud je však požadavkem zobrazení neexistující virtuální krajiny, musíme si data do výškové mapy vhodným způsobem vygenerovat. K tomuto účelu se používají nejrůznější metody. Požadavkem, na každý postup při generování výškové mapy, je naplnit toto dvourozměrné pole vhodnými daty tak, aby po převedení na model, co nejvíce připomínálo reálný povrch. V následujících odstavcích stručně popíši běžně používané algoritmy pro výpočet výškových map.

2.1 Midpoint Displacement

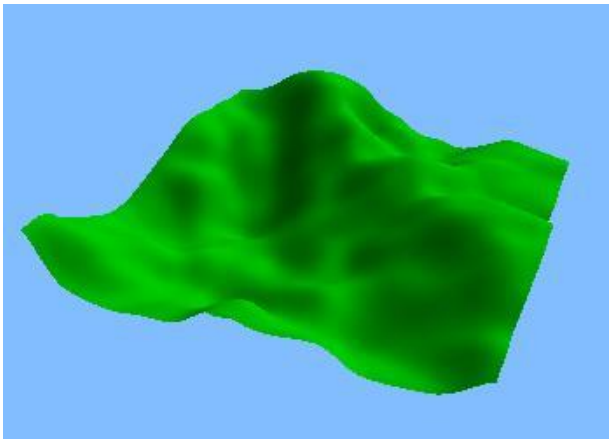
Tato metoda funguje na rekursivním principu. V každém kroku se náhodně zvýší nebo sníží výška středního bodu čtverce krajiny. Plocha mapy se pak rozdělí na čtvrtiny a postup se pro ně opakuje. Výsledkem je relativně přirozená krajina. Rozměry krajiny musí být mocninou dvou, kvůli postupnému dělení čtverců.



Obrázek 2.2: Příklad krajiny vygenerované metodou Midpoint

2.2 Fault Formation

Algoritmus Fault Formation pracuje na odlišném principu. Výchozím stavem je výšková mapa, ve které jsou veškeré hodnoty nulové. V krajině se náhodně určí dva body. Přímkou, kterou definují tyto body, pomyslně zlomíme krajinu. Veškeré body na jedné straně přímky zvýšíme o stejnou hodnotu. Tento postup opakujeme v cyklu podle potřeby. Výsledkem je mírně zvlněná krajina.



Obrázek 2.3: Příklad krajiny vygenerované metodou Fault formation

Kapitola 3

Perlinova funkce

3.1 Princip

Generování krajiny pomocí Perlinovy funkce nebo-li Perlinova šumu, přistupuje k problému naprosto odlišným způsobem. Principem není postupné upravování výškové mapy, ale vytvoření 2D funkce, jejíž hodnoty v požadovaných místech přeneseme do výškové mapy. Hledaná funkce musí být spojitá a vždy definovaná.

Funkci, která splňuje výše uvedené podmínky, vyvinul dle [2] v letech 1983–1990 profesor Ken Perlin z Newyorkské univerzity. Její princip spočívá ve složení výsledného šumu z několika dílčích funkcí (Noise) různé frekvence a amplitudy.

Řídícími parametry šumu jsou persistence p a množství opakování dílčích funkcí n. Tyto parametry definují aplitudu a frekvenci dílčí funkce následovně:

$$A = p^i$$

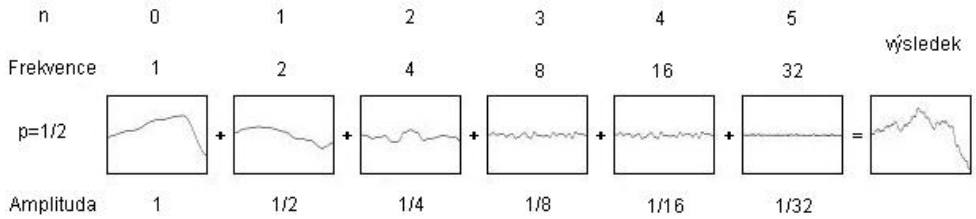
$$f = 2^i$$

kde

$$i = 0..n$$

Výslednou Perlinovu funkci s využitím výše uvedených vztahů tedy můžeme zapsat takto:

$$Perlin(x, y) = \sum_{i=0}^{n} A * Noise(x * f, y * f)$$



Obrázek 3.1: Princip Perlinovy funkce

3.2 Implementace

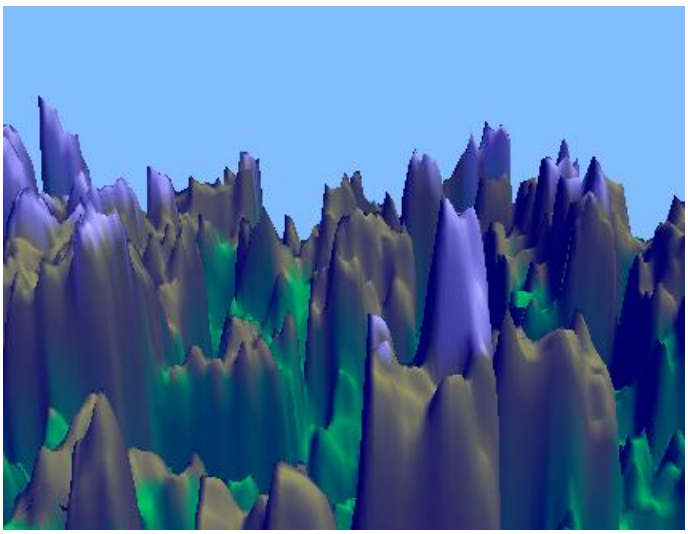
Výpočet dílčí funkce (pseudokód):

```
  const k1 = 15731 + rand(2500)
  const k2 = 789221 + rand(10000)
  const k3 = 1376312589 + rand(100000000)

func Noise(x,y)
{
  n = x + y * 57
  n = (n<<13) ^ n
  Noise = 1 - ((n * (n * n * k1 + k2) + k3) &
          7fffffff) / 1073741824
}
```

Pokud bychom použili pouze tuto funkci jako dílčí do výpočtu Perlinova šumu, výsledná krajina by byla značně kostrbatá (viz. obrázek 3.2) a naprosto by nevyhovovala našim požadavkům. Proto je třeba provést vyhlazení a interpolaci pomocí sousedních bodů. Detailní popis postupu viz. [4].

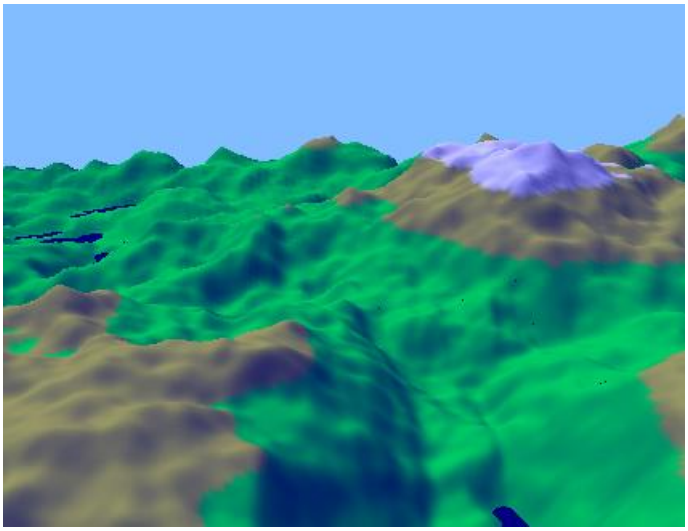
Všimněme si konstant $k1$,$k2$ a $k3$. Jejich volbou ovlivňujeme tvar krajiny. Tyto konstanty jsou incializovány ve funkci *PerlinInit*.



Obrázek 3.2: Neupravená Perlinova funkce

Vyhlazení krajiny odstraní ostré špičaté vrcholky hor a interpolace zajistí plynulé přechody v krajině. Cenou nám ovšem bude několikanásobný nárust doby generování krajiny.

Při prostém generování krajiny o rozměrech 1024x1024 bude funkce *Noise* volána 6.291.456krát. Naproti tomu při použití vyhlazování a interpolace bude volána 226.492.416krát, což je 36krát více. Odměnou nám může být výsledek, jímž je relativně přirozená krajina (viz. obrázek 3.3).



Obrázek 3.3: Interpolovaná Perlinova funkce

Výpočet výsledné Perlinovy funkce vypadá tedy takto:

```
func PerlinNoise(x, y)
{
  sum = 0
  p = 0.5 //persistence
  n = 5 //pocet dilcich funkci

  for(i = 0; i <= n; i++)
  {
    frequency = 1
    amplitude = p

    for (j = 0; j < i; j++) frequency *= 2
    for (j = 0; j < i; j++) amplitude *= p

    sum = sum + InterpolatedNoise(x * frequency, y * frequency) * amplitude
  }

  PerlinNoise = sum
}
```

Kapitola 4

Zobrazování terénu

Jak jsem již v úvodu uvedl, hlavním problémem při zobrazování krajiny je neúměrné množství dat, které musí grafická karta zobrazit. V praxi se tedy snažíme tento počet dat snížit. Jedná se především o počet zobrazovaných ploch (trojúhelníků, polygonů).

Existuje mnoho způsobů, jak optimalizovat model krajiny. V následujících odstavcích stručně popíši principy několika metod, pro urychlení zobrazování terénu.

4.1 ROAM algoritmus

Zkratka ROAM znamená Real-time Optimally Adapting Meshes. Tento algoritmus si před zahájemín renderování vytvoří binární strom trojúhelníků. Při renderování krajiny je pak schopen slučovat nebo rozdělovat plochy dle vzdálenosti od kamery a tím dynamicky regulovat množství trojúhelníků ve scéně. Více viz. [6].

4.2 Triangle-strip method

V pozadí této metody je snaha převést maximum krajiny na triangle-strip namísto toho, aby byla zobrazována po jednotlivých trojúhelnících. Důvodem je fakt, že triangle-strip redukuje přebytečná data posílaná hardwaru a také skutečnost, že grafická karta si může uchovat transformaci předchozích dvou vertexů, když generuje další trojúhelník. Nevýhodou této metody je zvýšený počet trojúhelníků v modelu, neboť algoritmus generující triangle-strip musí v některých místech přidat přebytečné trojúhelníky, aby mohl pokračovat v převodu krajiny na triangle-strip. Více viz. [5].

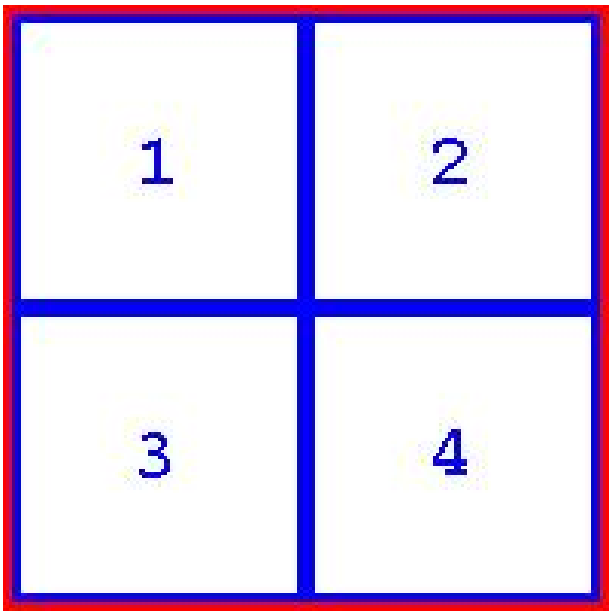
Kapitola 5

Chunked-LOD

5.1 Princip

Jak již z názvu metody vyplývá (chunk = kus, LOD = Level of detail = úroveň detailu), princip optimalizace spočívá v myšlence, že vzdálenější části krajiny nemusí být vykreslovány v maximálním detailu. To znamená, že s rostoucí vzdáleností od kamery redukujeme detail krajiny. Narozdíl od jiných algoritmu,kde máme krajinu jako jednolitý celek, je terén rozdělen na kusy nebo-li segmenty.

Při generování modelu krajiny o rozměrech například 512x512 postupujeme následovně. Vygenerujeme si výškovou mapu celého terénu. Zvolíme si, že rozměr jednoho segmentu v paměti bude například 32x32. Vytvoříme si první segment, který bude popisovat celou krajinu tedy o souřadnicích 0,0 – 512,512. Do tohoto segmentu přeneseme z výškové mapy každou 16tou (512/32 = 16) hodnotu. Nyní nám tento segment popisuje krajinu v minimálním detailu. Vytvoříme 4 podsegmenty (viz. obrázek 5.1).



Obrázek 5.1: Chunked-LOD: Dělení segmentů

První z nich bude popisovat krajinu o souřadnicích 0,0 – 256,256. Druhý o souřadnicích 256,0 – 512,256 atd. Každý z těchto 4 podsegmentů opět naplníme daty z výškové mapy s tím rozdílem, že budeme přenášet každou osmou ($256/8 = 8$) hodnotu. Ke každému segmentu vytváříme další 4 podsegmenty, které definují stejný prostor krajiny s dvojnásobným detailem. Až se při generování dostaneme do stavu, že segment na mapě popisuje území o rozměrech stejných jako je jeho velikost (tedy 32x32), nebudeme pro něj již vytvářet podsegmenty, neboť úroveň detailu je maximální.

Po tomto postupu nám vznikne quad-tree nebo-li strom, kde každý uzel má 4 poduzly. Pro každý segment vygenerujeme příslušný model krajiny. Při zobrazování krajiny procházíme quad-tree od kořene a testujeme, zda aktuální prvek stromu vyhovuje našim požadavkům detailu. Vypočteme si vzdálenost segmentu d od kamery a míru chyby zobrazení $chunk\_error$ aktuálního segmentu podle následujících vztahů.

$$d_{2D} = \sqrt{(cam_x - segment_x)^2 + (cam_y - segment_y)^2}$$

$$d = \sqrt{{d_{2D}}^2 + (cam_z - segment_z)^2}$$

$$chunk\_error = (x_2 - x_1)/(segment_{width} - 1)$$

a dosadíme do vztahu

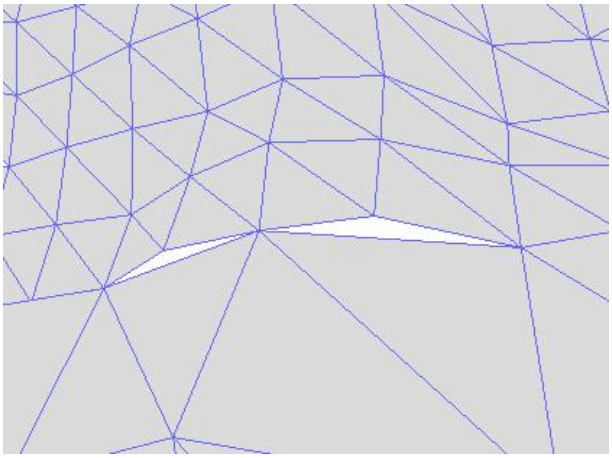
$$show = (chunk\_error * LODdistance)/d$$

kde $LODdistance$ určuje vzdálenost od kamery, od které se může snižovat úroveň detailu a x_1 a x_2 jsou souřadnice okraje segmentu v krajině. Pokud je výsledek $show <= 1$, zobrazí se daný segment, protože vhledem ke vzdálenosti od kamery je úroveň detailu dostačující. Pokud je ovšem $show > 1$ pro aktuální segment, musíme opakovat stejný postup pro jeho 4 podsegmenty.

5.2 Navazování segmentů

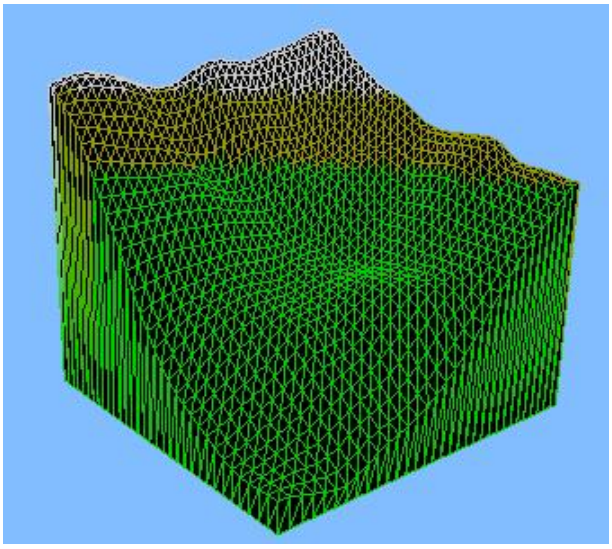
Jelikož je terén sestavován z částí a ke každé je přistupováno jako k samostatnému modelu, nutně narážíme na problém visuálního spojení segmentů tak, aby krajina vypadala jako celistvý útvar. Velké potíže nám budou činit nestejně velké segmenty, které budou sousedit ve výsledné vykreslované krajině. V následujících odstavcích se seznámíme s některými nedostatky a ukážeme si, jak je řešit.

Prvním problémem, se kterým se musíme vypořádat, jsou mezery v krajině v místech, kde se dotýkají segmenty různých úrovní detailu. Tuto situaci zmiňuje již [1]. K tomuto problému dochází v důsledku rozdílné členitosti terénu těchto segmentů. Tento jev je dobře patrný na obrázku 5.2. Bílé trojúhelníky jsou chybějící plochy v krajině. Výsledkem je, že pozorovatel vidí pod krajinu.



Obrázek 5.2: Nespojené segmenty

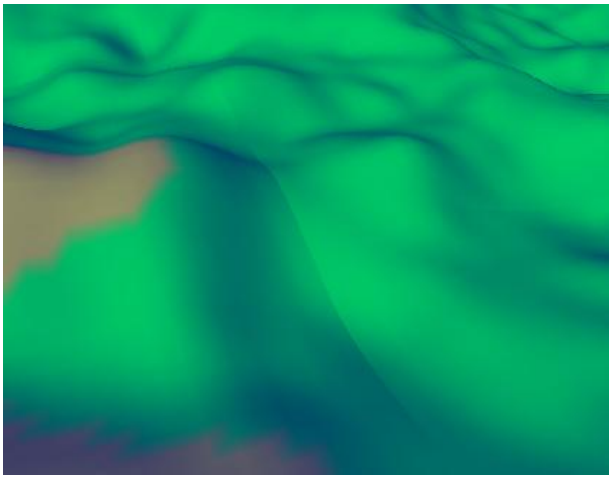
Abychom vyřešili tento problém, stačí na okrajích každého segmentu vytvořit svislé stěny (viz. obrázek 5.3). Při zobrazování dvou různě velkých segmentů, pak tyto stěny zaplní mezery a pozorovatel již nebude vidět pod krajinu.



Obrázek 5.3: Segment se svislými stěnami

Další problém, na který narazíme, jsou viditelné nesrovnalosti na hranicích mezi segmenty (viz. obrázek 5.4). Tento jev vzniká v důsledku chybně vypočtených normálových vektorů na okrajích těchto segmentů.

Pro výpočet normály místa v krajině jsou nutné informace o výšce sousedních bodů. Pokud tedy počítáme normálový vektor pro okrajový bod, tyto informace nám nutně chybí. Řešení je velmi prosté. Nejprve si celou krajinu vygenerujeme do jediného velkého pole. Při vytváření modelů pro jednotlivé dílčí segmenty krajiny, jsme pak schopni počítat i s hodnotami mimo rozsah aktuálního segmentu, neboť máme potřebná data k dispozici v podobě pomocného pole. Výsledkem tohoto přístupu je dokonalé navázání stejně velkých segmentů.



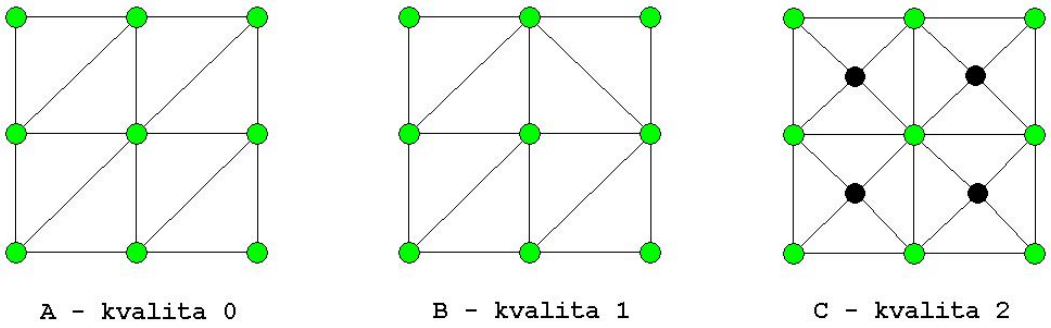
Obrázek 5.4: Chybně vypočtené normály

Pokud ovšem při vykreslování krajiny sousedí segmenty různých velikostí, problém nekorektního navázání terénu přetrvává. Chybné spojení segmentů je způsobeno faktem, že pro výpočet normálového vektoru jsou potřebné nejen sousední výšky terénu, ale také jejich vzájemná vzdálenost a tak je v případě nestejně velkých segmentů rozdílná. Tento problém ovšem není již tak palčivý, neboť vhodnou volbou LOD vzdálenosti můžeme dosáhnout toho, že detail krajiny začne klesat v dostatečné vzdálenosti od kamery a uživatel tedy nebude tak silně vnímat tyto nesrovnalosti. Jisté problémy tento jev bude pravděpodobně způsobovat v leteckých simulátorech, kde pozorovatel pohlíží na krajinu z velké vzdálenosti a rozdíly mezi segmenty jsou pro něj proto markantnější. Proto bych se v rámci diplomové práce rád zabýval, mimo jiné, dokonalým navázáním všech segmentů.

5.3 Implementace

Aplikace je implemtována v prostředí Open Inventor od firmy SGI. Toto prostředí je nadstavbou nad knihovnou OpenGL. Přináší vyšší stupeň abstrakce při programování grafických aplikací. Konkrétní balík, který je použit v tomto projektu je Coin od firmy Systems in Motion pod GPL licencí. Více k historii a používání Open Inventoru viz. [3].

Při převádění výškové mapy na 3D model, lze při spuštění aplikace (parametrem -hq) určit jednu ze 3 možných voleb teselace. Rozdíly mezi jednotlivými způsoby osvětluje obrázek 5.5.



Obrázek 5.5: Různé metody teselace krajiny

Zelené body na obrázku jsou hodnoty z výškové mapy a černé body jsou dopočítany pomocí interpoloce. Možnost A je v aplikaci výchozí. Tento způsob teselace určuje, že každý čtverec v krajině bude reprezentován dvěma trojúhelníky a rozdělení čtverce bude vždy ve stejném směru. Naproti tomu metoda B určuje, že se čtverec zobrazí také jako 2 trojúhelníky, ovšem směr rozdělení čtverce se určí podle výšek jeho rohů v krajině tak, aby oba trojúhelníky lépe postihly sklon krajiny. A konečně metoda C modeluje čtverec v krajině jako 4 trojúhelníky. Černý bod je doplněn pomocí lineární interpolace čtyř krajních bodů.

Jak již bylo z dřívějších obrázků vidět, v krajině se vyskytují 4 druhy povrchu. A to konkrétně vodní hladina, tráva, hory a sněhové čepičky hor. V aplikaci je tohoto efektu docíleno pomocí materiálů, kdy každému vertexu přidělíme materiál podle je výšky v krajině. Voda je realizována jako vodorovný čtverec, který protíná krajinu a tím vytváří dojem hladiny.

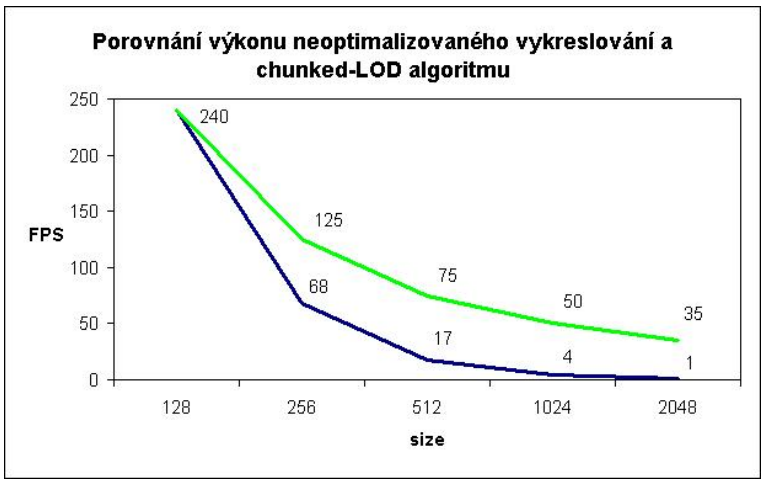
Kapitola 6

Výkonové a vizuální vlastnosti aplikace

6.1 Výkon

Domnívám se, že po výkonové stránce aplikace splnila hlavní požadavek. A tím je zobrazení rozsáhlé krajiny s přijatelnou úrovní detailu v reálném čase. V následujícím grafu je dobře patrný nárust výkonu, především při zobrazování rozsáhlých krajin.

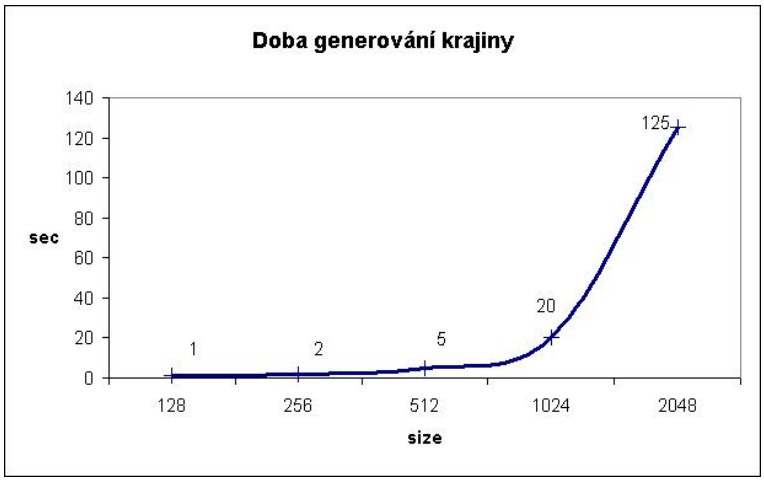
Hodnoty byly získány na počítači: Athlon64 3000+, 1024 MB RAM, ATI Radeon 9600XT.



Obrázek 6.1: Graf porovnání výkonu (*modrá* = neoptimalizovaný algoritmus, *zelená* = chunked-LOD)

Jak již z grafu vyplývá, síla chunked-LOD algoritmu se naplno projeví při renderování krajiny s vysokým rozlišením. V případě terénu o rozměrech 1024x1024 byl výkon chunked-LOD algoritmu 25krát vyšší a při rozlišení 2048x2048 dokonce 35krát vyšší.

Závislost času generování krajiny na jejích rozměrech je znázorněn na obrázku 6.2. Zde, jak vidíme, můžeme hovořit o menších nesnázích, v případě, že budeme chtít krajinu dogenerovávat dynamicky při běhu aplikace. Možnou eventualitou by mohlo být zvolení dostatečně malých částí terénu, které se budou generovat, aby uživatel nepostřehl pokles zobrazovaných FPS.



Obrázek 6.2: Graf doby generování v závisloti na velikosti krajiny

6.2 Vizuální vlastnosti

Při posuzování vizuálních vlastností aplikace, musíme mít na paměti, že neustále volíme kompromis mezi dokonalou grafikou a nenáročnou aplikací, která bude schopna pracovat v reálném čase s dostatečným výkonem.

Na první pohled se zdá být krajina téměř dokonalá. Při průletu se dynamicky mění a reaguje na pohyb kamery tak, aby pozorovatel viděl v nejbližším okolí krajinu v maximálním detailu. Ovšem při podrobnějším pohledu, oku kritika jistě neuniknou viditelné přechody mezi segmenty krajiny. K dokonalému efektu také nepřispívají skokové změny detailu segmentů. V ideálním případě by měl terén plynule přejít z jedné úrovně detailu do jiné.

Kapitola 7

Závěr

Podařilo se implementovat algoritmus pro generování náhodného terénu Perlinovou funkcí a zobrazování rozsáhlé krajiny v reálné čase pomocí chunked-LOD algoritmu.

Tento projekt a jemu podobné jsou vystaveny v systému Wiki na internetové adrese http://merlin.fit.vutbr.cz/wiki/index.php?title=Inventor_Projects_2005.

7.1 Kde lze projekt využít?

Možné využití projektu vidím všude tam, kde je nutné najednou a v reálném čase zobrazovat velký kus krajiny. Například v leteckých simulátorech, kde je hlavním požadavkem komplexní pohled na krajinu, přičemž na detailu terénu nám prioritně nezáleží.

7.2 Další možná pokračování projektu

Jak jsem již uvedl výše, jako prioritní rozšíření projektu považuji dokonalé navázání všech segmentů, neboť v současné podobě tento nedostatek působí příliž rušivě v celkovém dojmu krajiny. Dále by bylo dobré doplnit texturování terénu, což by celé aplikaci dodalo realističnosti. K lepšímu dojmu by také pomohlo umístění celé krajiny do sky boxu. Zajímavým rozšířením by mohlo být automatické dogenerovávání terénu, čímž by se vytvořilo zdání nekonečné krajiny.

Všechna uvedená a mnohá další rozšíření bych rád implementoval v rámci diplomové práce, aby se grafický výsledek co nejvíce podobal skutečné krajině.

Literatura

[1] Sánchez-Crespo. *Core Techniques and Algorithms in Game Programming.* 2003.

[2] WWW Stránky. Making noise. `http://www.noisemachine.com/talk1`.

[3] WWW Stránky. Open inventor tutorial.
`http://www.root.cz/serialy/open-inventor/`.

[4] WWW Stránky. Perlin noise.
`http://www.freespace.virfin.net/hugo.elias/models/m_perlin.htm`.

[5] WWW Stránky. Real-time rendering of realistic landscape.
`http://www.dcs.shef.ac.uk/teaching/eproj/ug2003/pdf/u0bm.pdf`.

[6] WWW Stránky. Roaming terrain: Real-time optimally adapting meshes.
`http://www.llnl.gov/graphics/ROAM/roam.pdf`.

Příloha A

Uživatelský manuál

Chování algoritmu lze ovlivnit pouze parametry, se kterými je program spuštěn. Jejich seznam, výchozí hodnoty a význam shrnuje následující tabulka.

Parametry programu:

Parametr	**Implicitně**	**Význam**
-seed	0	Inicializuje generátor náhodných čísel
-size	512	Velikost celé krajiny
-dimension	33	Velikost segmentu
-distance	100	LOD vzdálenost
-filterQ	0.5	Parametr filtrovací funkce $(0 - 1)$
-hq	0	Režim teselace krajiny $(0, 1, 2)$

Příloha B

Obrázová příloha