MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Procedurálne generovanie a zobrazovanie terénu v reálnom čase

BAKALÁRSKA PRÁCA

Michal Kuderjavý

Brno, jar 2010

Prehlásenie

Prehlasujem, že táto bakalárska práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Vedúci práce: RNDr. Vít Kovalčík, Ph.D.

Zhrnutie

Cieľom tejto bakalárskej práce je zoznámenie sa s technikami procedurálneho generovania terénu a porovnanie jednotlivých algoritmov generujúcich terén. Súčasťou práce je naprogramovanie aplikácie, schopnej v reálnom čase generovať a zobrazovať časť terénu.

Kľúčové slová

procedurálne techniky, procedurálne generovanie terénu, šum, fraktály, Perlinov šum, Simplexný šum, Fractional Brownian motion, Heterofractal, Hybrid multifractal, Ridged multifractal, Algoritmus Diamant-Štvorec, výškové mapy

Obsah

Kapitola 1

Úvod

Procedurálne techniky boli používané v celej histórii počítačovej grafiky. Ide o silný nástroj, ktorý sa používa na generovanie mnohých prírodných materiálov, či úkazov – napr. mramor, drevo, voda, oblaky, či terén [1]. Obľúbenosť a použitie týchto nástrojov začína v poslednej dobe stúpať, a to hlavne vďaka zvyšovaniu výkonu procesorov a grafických kariet.

Skutočná explózia nielen v oblasti procedurálneho generovania terénu, ale aj procedurálnych techník ako takých nastala, keď v roku 1985 profesor Ken Perlin predstavil tzv. šum. Tento šum (neskôr pomenovaný po ňom – Perlinov šum [2]) sa stal základnou funkciou využívanou nielen pri procedurálnom generovaní terénu, či realisticky vyzerajúcich textúr, ale bol aj úspešne použitý na produkovanie realistickej, komplexnej animácie (napr. ohňa a dymu). Procedurálne techniky sa tak stali oblasťou aktívneho výskumu v počítačovej grafike, vzhľadom k širokým možnostiam uplatnenia, ktoré ponúkajú.

Mnoho prírodných tvarov má zaujímavú, špecifickú vlastnosť – sebapodobnosť, ktorá je nezávislá na mierke. Objekt s touto vlastnosťou je presne alebo približne rovnaký ako nejaká jeho časť. Už v roku 1960 Dr. Benoit Mandelbrot našiel spojenie medzi prírodnými tvarmi, ktoré si zachovávajú istý stupeň sebapodobnosti, a matematickými konštrukciami – fraktálmi [3]. Tieto znalosti nám umožňujú generovať realisticky vyzerajúci terén s charakteristikami podľa zadaných parametrov.

Cieľom tejto práce je oboznámiť čitateľa so základnými algoritmami, ktoré sa používajú na procedurálne generovanie terénu. Všetky tieto algoritmy vychádzajú z fraktálnej podstaty reálneho terénu. Najväčší dôraz je kladený na algoritmy založené na šume, ktoré sú najčastejšie používané a sú schopné generovať rôzne druhy terénu.

Úvodná kapitola práce nás zoznamuje s procedurálnymi technikami ako takými, popisuje ich vlastnosti a uvádza do problematiky fraktálov, ktoré sú často využívané v tejto oblasti. Následne podrobnejšie opisujem primitivum procedurálnych techník – šumovú funkciu, pričom uvádzam dve najpoužívanejšie šumové funkcie. Popis samotných algoritmov generujúcich procedurálny terén sa nachádza v ďalšej kapitole.

Súčasťou práce je aplikácia, ktorá umožňuje generovať terén podľa zvoleného algoritmu na základe zadaných parametrov. Samostatná kapitola je vyhradená opisu tohto programu a použitých techník na zobrazovanie terénu. Ďalšia kapitola je venovaná porovnaniu uvedených algoritmov. Záverečná kapitola obsahuje zhodnotenie dosiahnutých výsledkov a načrtnutie možných vylepšení.

Kapitola 2

Procedurálne techniky

Procedurálne techniky spočívajú v algoritmickom generovaní požadovaného obsahu, či už ide o terén, modely alebo textúry. Namiesto manuálneho modelovania definujeme algoritmus, ktorý za nás tento objekt vytvorí. Napríklad algoritmus, ktorý procedurálne generuje textúru dreva nepotrebuje na vstupe skenovaný obrázok, aby definoval farbu dreva. Namiesto toho používa matematické funkcie, ktoré túto farbu určia samé.

2.1 Vlastnosti procedurálnych techník

Medzi základné vlastnosti patrí abstrakcia. Namiesto toho, aby sme ukladali zložité údaje o zobrazovanom objekte, abstrahujeme vlastnosti tejto scény do algoritmu, schopného vygenerovať požadovaný objekt podľa potreby. Podstatne sa tým znižujú nároky na úložný priestor, na druhej strane však tieto algoritmy môžu byť výpočtovo náročné.

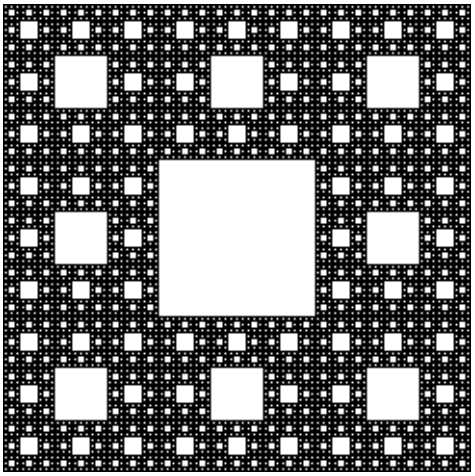
Taktiež získavame parametrickú kontrolu nad generovaným obsahom. Procedurálny algoritmus dostáva na vstupe rôzne parametre, definujúce jeho chovanie. Napríklad pri generovaní terénu môže existovať parameter, ktorý ovplyvňuje hrubosť, resp. jemnosť terénu. Vhodným nastavením vstupných parametrov dosiahneme požadovaný výsledok.

Poslednou hlavnou vlastnosťou procedurálnych techník je pseudonáhodnosť. Pri generovaní terénu vyžadujeme, aby výsledný terén bol náhodný, no zároveň aj reprodukovateľný zadaním rovnakých parametrov. Skutočná náhodná funkcia ale nemá žiadne parametre a jej chovanie nie je možné zreprodukovať. Namiesto toho potrebujeme funkciu, ktorá sa zdá byť náhodná, ale pre rovnaký vstup dáva rovnaký výstup. Takto je možné vygenerovať požadovaný terén náhodne, no zároveň je ho možné zreprodukovať zadaním rovnakých parametrov.

2.2 Fraktál

Fraktál je geometricky komplexný objekt, ktorý vzniká niekoľkonásobným opakovaním zvoleného tvaru. Jeho hlavná vlastnosť je sebapodobnosť, nezávislá na zvolenej mierke. Termín fraktál zaviedol matematik Mandelbrot a je odvodený z latinského fractus (zlomený, rozbitý). Pretože fraktály sú podobné pri akejkoľvek mierke, často sú považované za nekonečne komplexné. Medzi prírodné úkazy, ktoré je možné popísať fraktálmi, patria napríklad: oblaky, terén, blesky, tvar pobrežia, niektoré rastliny, vetvenie stromov, snehové vločky.

Fraktály majú dôležitú vlastnosť zvanú fraktálna dimenzia (nazývaná aj Hausdorfova dimenzia, vždy ostro väčšia než euklidovská). Klasická euklidovská dimenzia je dimenzia, ktorú vyjadrujeme prirodzenými číslami – nula predstavuje bod, jedna dimenzia predstavuje čiaru, dve dimenzie rovinu a tri dimenzie priestor. Euklidovská dimenzia však vykazuje isté nečakané správanie na fraktáloch. Napríklad Sierpinského koberec má euklidovskú dimenziu 1, ale fraktálnu dimenziu 1,8928. Môžeme povedať, že čím väčšia fraktálna dimenzia, tým viac priestoru daný fraktál zaberá. V prípade generovania terénu fraktálna dimenzia určuje drsnosť výsledného terénu.



Obr. 2.1: Sierpinského koberec

2.3 Fraktály a terén

Fraktály úzko súvisia s procedurálnymi technikami. Samotná konštrukcia fraktálu je iteratívny proces, ktorý je možné popísať matematickými postupmi. Fraktály majú dobrú vlastnosť v tom, že sú popísané veľmi jednoduchými pravidlami. V praxi dokážu fraktálne procedurálne techniky poskytnúť značnú vizuálnu komplexnosť z relatívne krátkeho kódu.

Procedurálne techniky s obľubou využívajú ako základnú primitívnu funkciu šum. Táto funkcia predstavuje základný tvar, ktorý aplikujeme opakovane, podobne ako pri konštrukcii fraktálu. Keďže je nutné túto funkciu často vyhodnocovať, mala by byť jednoduchá a mala by mať malú časovú zložitosť.

Procedurálne generovanie terénu sa opiera o základnú vlastnosť reálneho terénu – sebapodobnosť. Menšie časti terénu sa podobajú na väčšie časti terénu a naopak. Z matematického hľadiska je preto možné modelovať terén pomocou fraktálov. To, či reálny terén vykazuje znaky fraktálu, bolo predmetom mnohých štúdií [4]. Zistenia z týchto štúdií potvrdzujú myšlienku použitia fraktálov na modelovanie terénu, no len po niekoľko radov zväčšenia. Toto zistenie je však logické – terén nemôže vykazovať vlastnosť sebapodobnosti donekonečna.

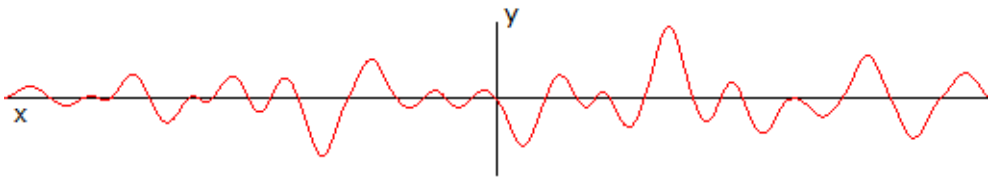
Kapitola 3

Šum

Ako už bolo povedané, fraktál vzniká periodickým opakovaním zvoleného tvaru. V procedurálnom generovaní terénu ide o základnú primitívnu funkciu, zvyčajne nazývanú šum (noise). Túto funkciu potom aplikujeme v rôznych mierkach, a týmto iteratívnym postupom vygenerujeme požadovaný fraktál. Táto funkcia je pseudonáhodná, a teda pre rovnaké vstupy generuje rovnaké výstupy.

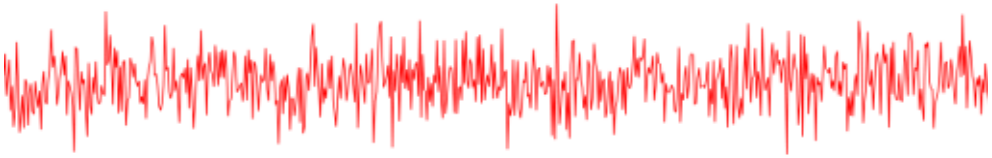
Prvou myšlienkou je použiť ako primitívnu funkciu biely šum[1]. Biely šum ale nie je to, čo skutočne chceme. Pri použití bieleho šumu ako základnej primitívnej funkcie sa generovaný terén stáva nestabilným a vznikajú problémy s aliasom[2] ([1], str. 67-68). Ideálna primitívna funkcia by mala byť koherentná (a samozrejme pseudonáhodná). Vlastnosti ideálnej primitívnej funkcie šum [1]:

- Šum je opakovateľná pseudonáhodná funkcia.
- Malá zmena vstupnej hodnoty spôsobí malú zmenu výstupnej hodnoty (koherencia).
- Veľká zmena vstupnej hodnoty spôsobí náhodnú zmenu výstupnej hodnoty.
- Šum má známy rozsah (najčastejšie od -1 do 1).
- Šum nevykazuje očividné periodicity alebo pravidelné vzory. Hoci každá pseudonáhodná funkcia je periodická, táto perióda môže byť veľmi dlhá.



Obr. 3.1: Koherentná pseudonáhodná šumová funkcia n(x)

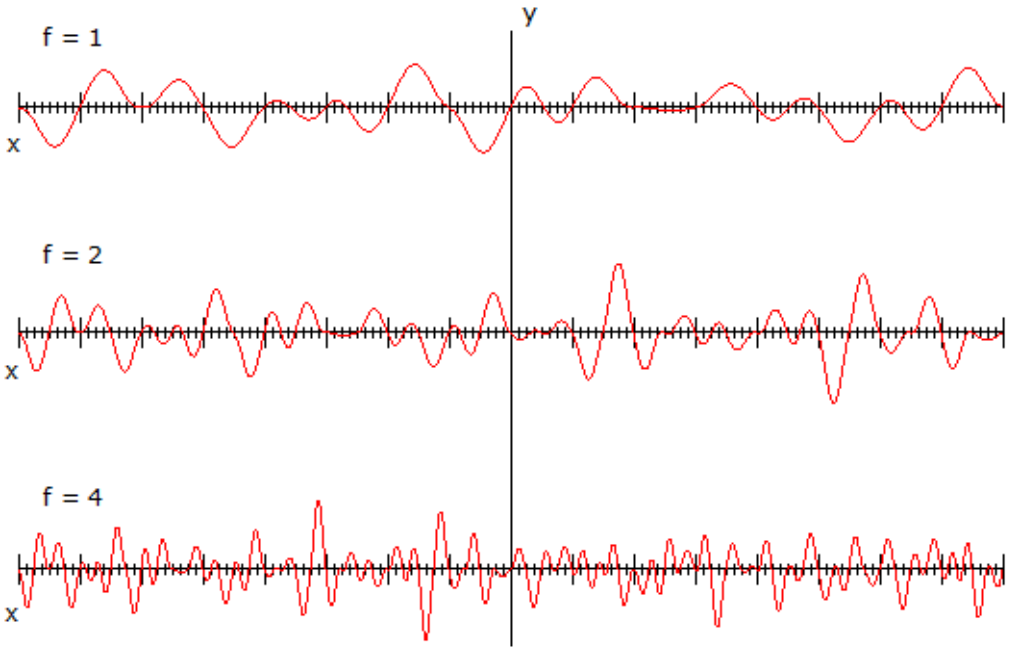
1. biely šum (white noise) – je náhodný signál, zdroj náhodných čísel, uniformne distribuovaných, so žiadnou koreláciou medzi nimi
2. alias – rušivé javy vznikajúce zobrazením v pravidelnej diskrétnej mriežke



Obr. 3.2: Funkcia náhodného bieleho šumu

Šumová funkcia má dva vstupy – seed (jadro) a frekvenciu. Parameter seed je vstupom do procedurálneho generátora náhodných čísel. Zmenou tohto parametru vygenerujeme iný (náhodný) šum. Pokiaľ ponecháme parameter seed rovnaký, vygenerujeme vždy rovnaký šum (pri zachovaní frekvencie). Tento parameter zaručuje pseudonáhodnosť výsledného šumu.

Frekvencia je počet cyklov za jednotku dĺžky. Šumová funkcia má podobné vlastnosti ako sínusoida. Má periodické cykly dĺžky 1/f, kde f je frekvencia, a na začiatku každého cyklu má hodnotu nula. Na rozdiel od sínusoidy, šumová funkcia môže, ale ani nemusí, prechádzať cez nulu uprostred cyklu.

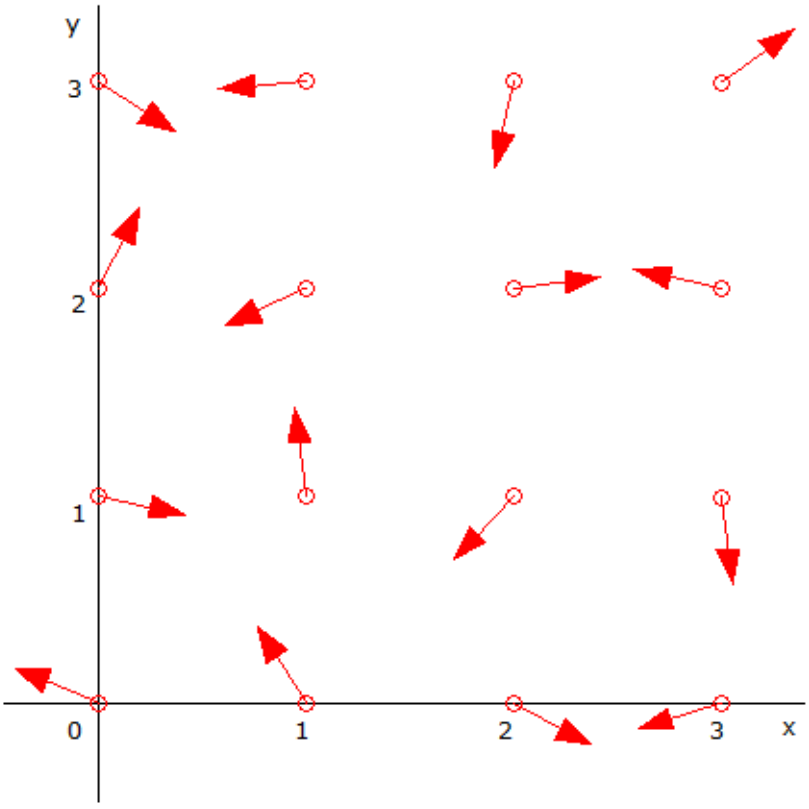


Obr. 3.3: Vplyv zmeny frekvencie na funkciu šumu

3.1 Perlinov šum

Najznámejšou šumovou primitívnou funkciou je Perlinov šum (Perlin noise). Túto funkciu vynašiel Ken Perlin v roku 1985 a v roku 1997 za ňu vyhral cenu Technical Achievement Award od Academy of Motion Picture Arts and Sciences.

Perlinov šum patrí do kategórie gradientového šumu [6], čo značí, že definuje pseudonáhodný gradient v pravidelne rozostúpených bodoch (určených súradnicami z prirodzených čísel) v priestore, medzi ktorými potom interpoluje. Hodnota gradientového šumu je nulová v týchto mriežkových bodoch. Pre určenie hodnoty požadovaného bodu X interpolujeme medzi 2^N (N – počet dimenzií) najbližšími mriežkovými bodmi.

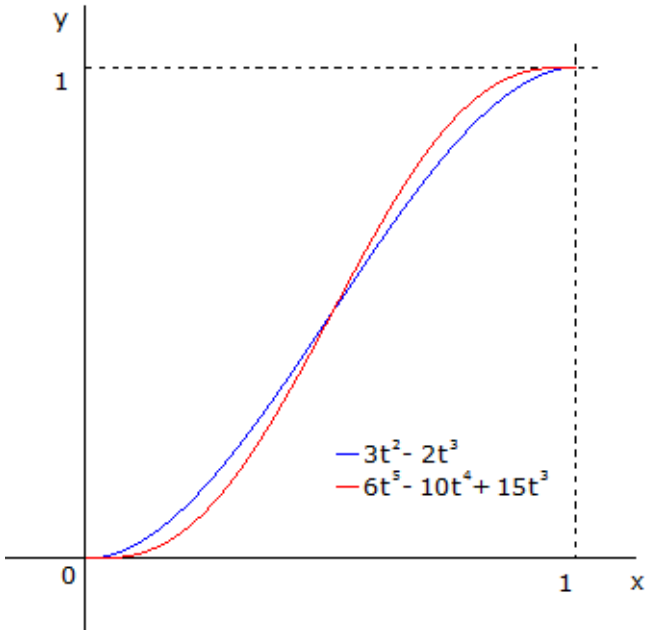


Obr. 3.4: Príklad gradientu dvojrozmerného Perlinovho šumu

Hodnotu bodu X určíme v tomto prípade interpolovaním medzi štyrmi najbližšími mriežkovými bodmi. Použitá interpolačná krivka je kubická Hermitova krivka $f(t) = 3t^2 - 2t^3$. V prípade trojrozmerného Perlinovho šumu interpolujeme medzi ôsmimi okolitými bodmi.

3.2 Vylepšený Perlinov šum

Pôvodná implementácia Perlinovho šumu má dva hlavné nedostatky, ktoré Perlin odstránil v roku 2002. Nový druh šumu nazval Vylepšený Perlinov šum (Improved Perlin Noise) [7]. Prvým vylepšením bola výmena pôvodnej interpolačnej krivky $f(t) = 3t^2 - 2t^3$, ktorej druhá derivácia je $6 - 12t$, a tá nie je nulová ani pre $t = 0$ alebo $t = 1$. Táto nenulovosť druhej derivácie vytvára nespojitosť, ktorá sa prejavuje pri nasvietení časti terénu generovaného pomocou Perlinovho šumu. Tento problém Perlin jednoducho vyriešil použitím novej interpolačnej krivky $f(t) = 6t^5 - 15t^4 + 10t^3$, ktorá je veľmi podobná ako pôvodná krivka, no zároveň má nulovú druhú deriváciu v oboch bodoch $t = 0$ a $t = 1$.



Obr. 3.5: Interpolačná krivka použitá pri výpočte šumu (modrá – pôvodná krivka, červená – nová krivka)

Druhým vylepšením bolo nahradenie pôvodnej, zbytočne výpočtovo náročnej metódy počítania gradientu novou, efektívnejšou metódou. Výsledný šum vyzerá vizuálne lepšie a je približne o 10% rýchlejší ako pôvodný šum [7].

3.3 Simplexný šum

Klasický Perlinov šum sa stal veľmi populárnym a často sa používa ako procedurálna primitívna funkcia. Napriek tomu obsahuje niekoľko ďalších nedostatkov, ktoré sa Perlin rozhodol odstrániť vytvorením nového šumu – Simplexného šumu [5]. Názov Simplexného šumu je odvodený od anglického slova simplex, čo značí jednoduchý. Simplexný šum má

niekoľko zásadných vylepšení oproti klasickému šumu [6]:

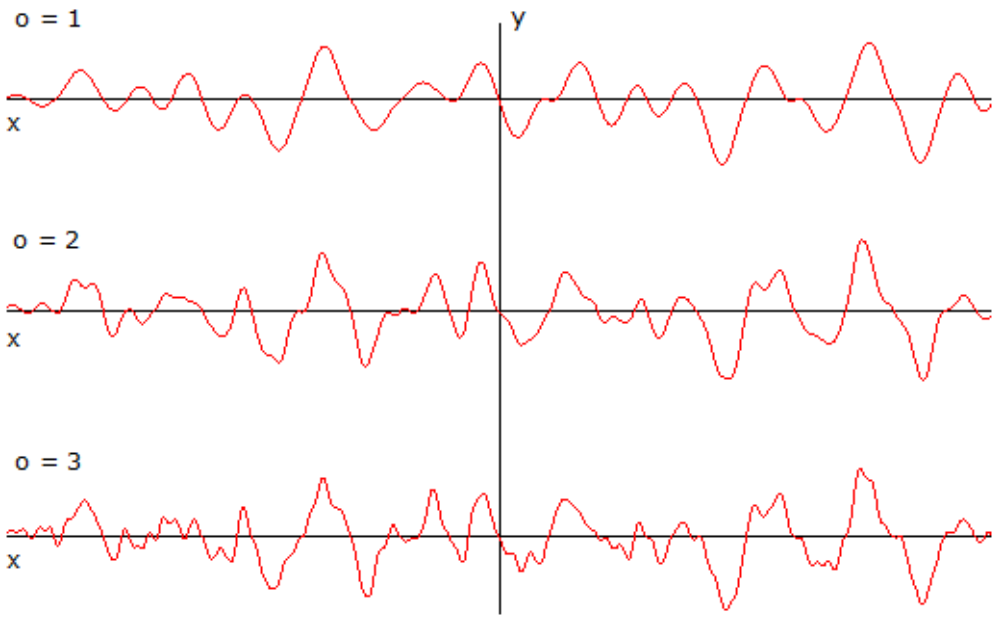
- Simplexný šum má menšiu časovú zložitosť vzhľadom na počet dimenzií. Časová zložitosť je O(N), kde N je počet dimenzií, namiesto $O(2^N)$, čo je časová zložitosť pôvodného Perlinovho šumu.
- Simplexný šum nemá žiadne viditeľné smerové artefakty.
- Simplexný šum má dobre definovaný a spojitý gradient, ktorý je možné vypočítať rýchlo.
- Simplexný šum je ľahko implementovateľný na hardware.

Zmenšenie časovej zložitosti spočíva vo výbere najvhodnejšieho a najkompaktnejšieho tvaru, ktorý je schopný vyplniť celý priestor. Simplexný šum používa vždy čo najjednoduchší tvar. Pre jednorozmerný šum najjednoduchší (a zároveň aj jediný) možný priestor vyplňujúci tvar sú intervaly rovnakej dĺžky, umiestnené jeden za druhým. V dvoch dimenziách je očividná voľba štvorec, no existuje aj jednoduchší útvar – rovnoramenný trojuholník, ktorý má menej vrcholov ako štvorec. Podobne pre trojrozmerné objekty nepoužijeme kocku s ôsmimi vrcholmi, ale štvorsten, ktorý má len štyri vrcholy. Namiesto interpolovania medzi 2^N vrcholmi (N je počet dimenzií), interpolujeme len medzi N + 1 vrcholmi, čo výrazne znižuje časovú zložitosť, hlavne pre vyššie dimenzie.

Kapitola 4

Šumové fraktálne algoritmy

Všetky algoritmy uvedené v tejto kapitole využívajú nejakú primitívnu šumovú funkciu. Zmena tejto funkcie samozrejme ovplyvňuje výsledný fraktál. Podobne ako pri fraktáloch, aj pri generovaní terénu využívame opakovanú aplikáciu primitívnej funkcie. Koľkokrát túto funkciu aplikujeme, určuje parameter octaves (oktávy). Tento parameter je spoločný pre všetky algoritmy v tejto kapitole. Pridávanie ďalších oktáv zvyšuje množstvo detailov výsledného fraktálu.



Obr. 4.1: Vplyv parametru octaves na šumovú funkciu

Pre každú ďalšiu oktávu zároveň zvyšujeme frekvenciu primitívnej funkcie. Koľkokrát túto frekvenciu zväčšíme, určuje parameter lacunarity. Najjednoduchšie je ponechať parameter lacunarity na 2, pri použití vyšších hodnôt začínajú byť evidentné jednotlivé frekvencie základnej šumovej funkcie, čo nevytvára vizuálne kvalitný terén. Pokiaľ chceme mať

terén čo najlepšej kvality, je dobré nastaviť parameter lacunarity na hodnotu blízku 2, napr. 1,9, alebo 2,1, ideálne nejaké iracionálne číslo ([1], str. 586).

4.1 Monofraktálne procedurálne algoritmy

4.1.1 Fractional Brownian motion

Fractional Brownian motion (fBm) [8] je generalizácia Brownovho pohybu, čo je náhodný pohyb molekúl v látke. Takisto býva často prirovnávaný k náhodnému pohybu opilca. fBm je zostavený tak, aby vyzeral rovnako vo všetkých miestach a vo všetkých smeroch – je štatistický monotónny, preto patrí medzi monofraktálne algoritmy.

Rovnako ako aj ďalšie prezentované algoritmy, fBm sa využíva v rôznych odvetviach procedurálneho generovania. Algoritmus je kontrolovaný jediným parametrom H (okrem klasických parametrov frekvencia, oktávy a lacunarity). Tento parameter, spoločne s lacunarity určuje fraktálnu dimenziu, a teda ovplyvňuje drsnosť terénu. Čím vyššia hodnota, tým hladší terén tento algoritmus produkuje.

```
float fBm(Vector point, float H, float lacunarity)
{
  float value = 0.0f;
  for (int i = 0; i < 8; ++i)
  {
    value += noise(point) * pow(lacunarity, -H * i);
    point *= lacunarity;
  }
  return value;
}
```

Povšimnime si, že konštrukcia tohto fraktálu je veľmi jednoduchá, jedná sa o jediný cyklus for, v ktorom sa volá základná primitívna funkcia šumu. Ako už bolo povedané, fBm je štatisticky monotónny. Skutočný terén je ale zriedka taký jednoduchý. Reálne pohoria majú rôznu drsnosť povrchu v rôznych výškach. Na generovanie takéhoto terénu potrebujeme komplexnejšie techniky – multifraktály.

4.2 Multifraktálne procedurálne algoritmy

Multifraktály sú heterogénne[1] fraktály, ktorých fraktálna dimenzia sa mení s pozíciou. Takéto fraktály už nie sú rovnaké vo všetkých miestach – napr. kopce a vrchy v teréne sú drsnejšie než údolia. Multifraktálne algoritmy lepšie zachytávajú vlastnosti reálneho povrchu – výsledný terén je rozmanitejší a realistickejší.

1. heterogénne – rôznorodé

4.2.1 Heterofractal

Pozorovanie, že v reálnom teréne zvyknú byť nízko umiestnené časti terénu jemnejšie, zatiaľ čo vyššie umiestnené časti terénu drsnejšie, viedlo k vytvoreniu tohto typu terénu ([1], str. 500-501). Jeho konštrukcia vychádza z fBm.

Výsledný terén už ale nie je monofraktálny, pretože jeho fraktálna dimenzia sa mení s narastajúcou výškou. Požadovaný efekt je dosiahnutý tým, že násobíme každú ďalšiu oktávu súčasnou hodnotou funkcie. To znamená, že blízko úrovne mora (nulová výška) budú ďalšie oktávy silno zoslabené, čo vedie k tomu, že nízko položený terén ostane hladký. Zvyšné frekvencie nebudú natoľko zoslabené, a teda terén vo vyšších výškach ostane členitý.

Heterofractal má rovnaké parametre ako fBm, ale pridáva aj parameter offset. Tento parameter určuje výšku terénu nad hladinou mora. Čím je terén vyššie nad morom, tým bude aj drsnejší (podľa definície tohto fraktálu), preto možno povedať, že tento parameter určuje aj heterogenitu výsledného povrchu.

4.2.2 Hybrid multifractal

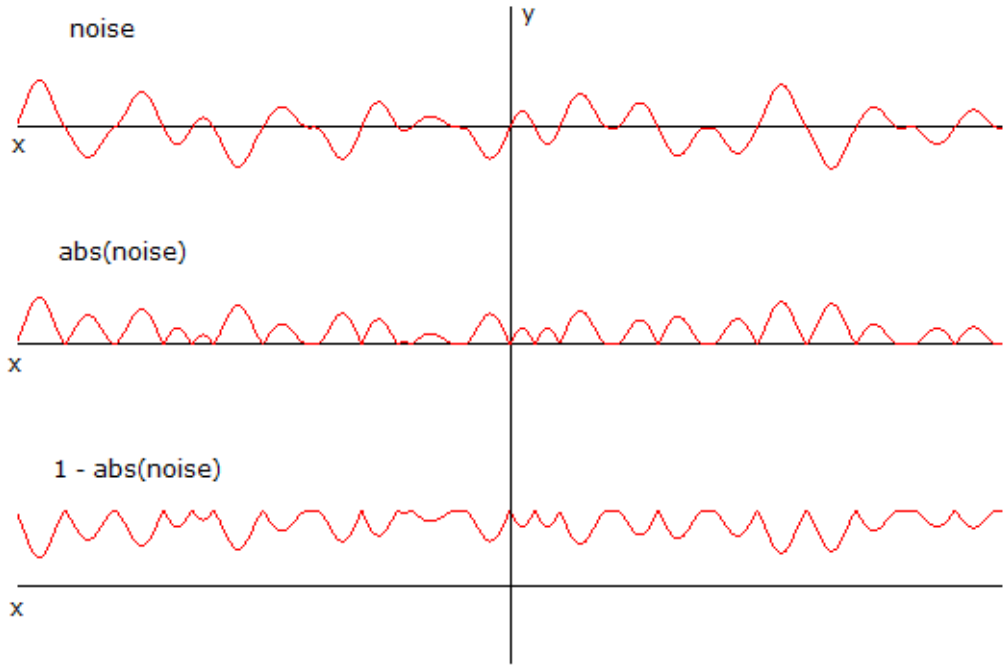
S podobným úmyslom bol navrhnutý aj tento typ terénu ([1], str. 502-503). Tentokrát vytvárame heterogénnosť povrchu tak, že násobíme vyššie frekvencie terénu lokálnou hodnotou predošlej frekvencie (a nie celej funkcie, ako v predošlom prípade).

Generovaný terén vykazuje silnejšie vyhladenie v oblasti nížin a ponecháva približne rovnakú zvrásnenosť terénu vo vyšších výškach, čo viac odpovedá realite. Vstupné parametre má zhodné ako terén heterofractal, no ich počiatočné hodnoty sú rozdielne. To je spôsobené odlišnou metódou výpočtu fraktálu.

4.2.3 Ridged multifractal

Ridged multifractal ([1], str. 504-505) je známym a často používaným šumovým algoritmom, pretože generuje vizuálne kvalitné a mohutné pohoria, vrchy, ale aj roviny a pláne. Ako základnú primitívnu funkciu používa absolútnu hodnotu šumovej funkcie, obrátenú hore nohami: $1 - \text{abs}(\text{noise})$. Takto vznikajú hrebene a horské vyvýšeniny.

Okrem klasických parametrov, ktoré má zhodné s fBm ovplyvňujeme správanie tohto šumu aj ďalšími parametrami gain a offset. Parameter gain určuje veľkosť prírastku novej oktávy oproti predošlej, kontrolujeme ním hlavne sekundárne detaily terénu. Parameter offset opäť ako v predošlých algoritmoch určuje výšku terénu nad hladinou mora.



Obr. 4.2: Konštrukcia prevrátenej absolútnej funkcie šumu. Takto vznikajú štíty a horské hrebene.

Kapitola 5

Ostatné fraktálne algoritmy

5.1 Algoritmus Diamant-Štvorec

Algoritmus Diamant-Štvorec (Diamond-Square algorithm) [9] patrí taktiež medzi fraktálne algoritmy, no nevyužíva žiadnu šumovú funkciu. Ide o veľmi populárny, rýchly, jednoduchý, monofraktálny algoritmus. Je známy aj pod názvami random midpoint displacement fractal (fraktál náhodného posunutia stredného bodu), alebo plasma fractal (plazmový fraktál).

Algoritmus dostáva na vstupe dvojrozmerné pole bodov, ideálne štvorec, s dĺžkou strany, ktorá je mocnina dvojky plus jedna (65, 129, 257). Na začiatku vygenerujeme výšku štyroch rohových bodov terénu. Následne je spustená iteratívna procedúra, ktorá má dva základné kroky:

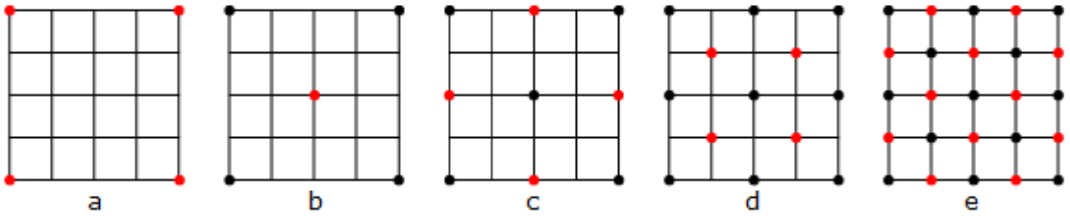
- Diamantový krok – vezmeme štyri body aktuálne spracovávaného štvorca a generujeme výšku prostredného bodu, ktorý leží na priesečníku oboch diagonál štvorca. Táto výška je vypočítaná ako priemer štyroch rohových bodov plus náhodná hodnota K. Takto získané body vytvárajú diamanty, ktoré sú vstupom do štvorcového kroku.

- Štvorcový krok – vezmeme štyri body aktuálne spracovávaného diamantu a generujeme výšku prostredného bodu, ktorý leží na priesečníku oboch diagonál diamantu. Táto výška je vypočítaná ako priemer štyroch rohových bodov plus náhodná hodnota K. Takto získané body vytvárajú štvorce, ktoré sú vstupom do diamantového kroku.

Algoritmus končí, keď vygenerujeme výšku všetkých bodov v poli. S bodmi na hranici poľa môžeme naložiť dvojako. Buď jednoducho použijeme na výpočet priemeru len body, ktoré sú k dispozícii, alebo namiesto chýbajúceho bodu použijeme bod na opačnej strane poľa, a tým vytvoríme opakovateľný (tileable) terén.

Na obrázku 5.1 je znázornený proces generovania terénu pomocou algoritmu Diamant-Štvorec pre dvojrozmerné pole veľkosti 5 krát 5. Čiernou farbou sú znázornené body, ktorých výška je známa (bola vygenerovaná v niektorých predošlých krokoch). Červenou farbou sú znázornené body, ktoré generujeme v tomto konkrétnom kroku.

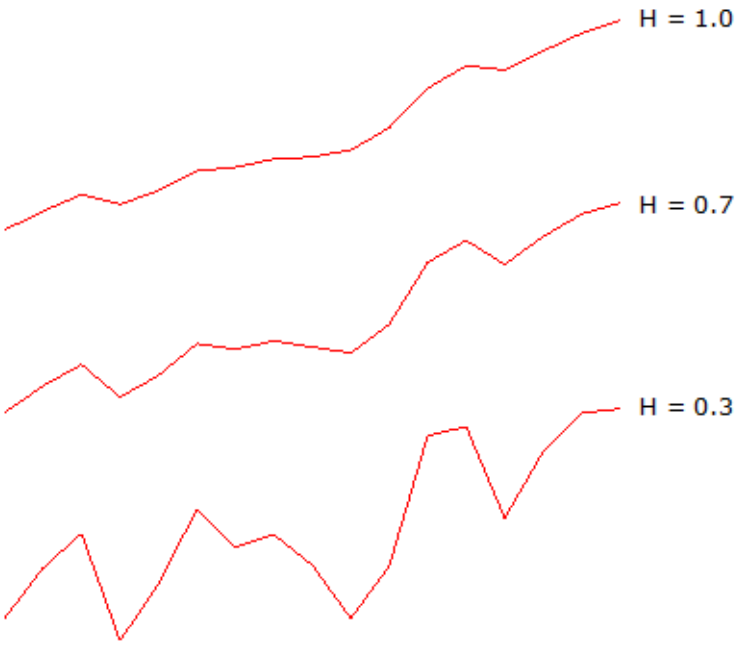
Začíname tým (5.1a), že vygenerujeme výšku štyroch rohových bodov, ktoré sú vyznačené červenou. Následne (5.1b) urobíme diamantový krok – vygenerujeme výšku prostredného bodu (vyznačeného na obrázku červenou) v štvorci, tvorenom týmito štyrmi bodmi.



Obr. 5.1: Jednotlivé kroky konštrukcie terénu pomocou algoritmu Diamant-Štvorec

V tomto kroku sme vytvorili štyri neúplné diamanty, ležiace na okrajoch celého poľa. Nasleduje štvorcový krok (5.1c), ktorý vygeneruje nové hodnoty pre body ležiace uprostred týchto štyroch diamantov. Týmto sme vytvorili štyri menšie štvorce. Opäť urobíme diamantový krok (5.1d), na týchto novo vygenerovaných štvorcoch a vygenerujeme ďalšie štyri body. Posledný štvorcový krok (5.1e) dogeneruje výšky zvyšných bodov poľa.

Výsledný fraktál je kontrolovaný jednak počiatočnou výškou štyroch rohových bodov vstupného poľa a jednak hodnotou parametru H, ktorý určuje, o koľko sa zmenší náhodná hodnota K pre každý ďalší krok. Čím vyšší parameter H, tým hladší terén generujeme. Tento parameter je ekvivalentom k rovnako nazvanému parametru pre šumové algoritmy.



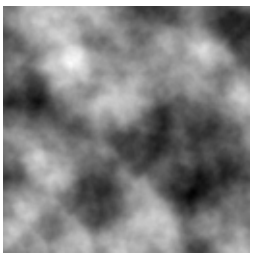
Obr. 5.2: Vplyv parametru H na generovaný terén

Kapitola 6

Zobrazovanie terénu

6.1 Výškové mapy

Výškové mapy sú najjednoduchším a najpraktickejším spôsobom reprezentácie terénu [10]. Výškové dáta terénu sú uložené v textúre (výškovej mape) ako mriežka bodov. Pre každý bod terénu je vo výškovej mape definovaná jeho výška. Výšková mapa býva uložená ako šedotónny[1] obrázok. Čím je farba svetlejšia, tým je v danom bode vyššia výška.



Obr. 6.1: Výšková mapa terénu, využívajúca len jeden farebný kanál

V tomto prípade je ale využívaný len jeden farebný kanál, čo umožňuje uloženie iba 256 rôznych stupňov výšky. V prípade komplikovanejšieho terénu je toto nedostačujúci počet a výsledný terén obsahuje rôzne neprirodzené „skoky“ vo výške. Pokiaľ využijeme všetky štyri farebné kanály, je možné zachytiť až 4 294 967 296 stupňov výšky (256^4). Taktiež je možné použiť 32-bitové textúry[2], ktoré poskytujú dostatočnú presnosť aj pri použití jediného kanálu.

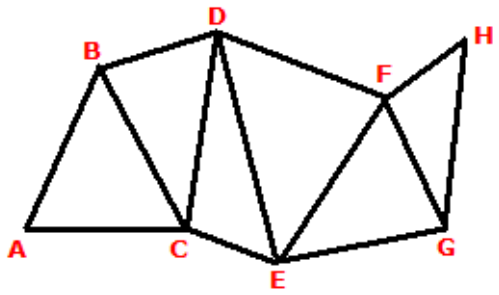
Medzi nevýhody výškových máp patrí najmä nemožnosť touto technikou zachytiť previsy v teréne, pretože v danom bode môžeme pomocou výškovej mapy definovať výšku len jedného bodu. Napriek tomu sú výškové mapy často používané kvôli svojej jednoduchosti a intuitívnej reprezentácii terénu.

---

1. šedotónny (gray-scale) obrázok – obrázok, ktorý obsahuje len stupne šedi
2. 32-bitové textúry (floating-point textures) – textúry, ktoré umožňujú uloženie farby každého kanálu v 32 bitoch (tradičné textúry používajú len 8 bitov na kanál)

6.2 Trojuholníkové pásy

Trojuholníkové pásy (triangle strips) [11] sú série spojených trojuholníkov, ktoré umožňujú zobrazovať objekty rýchlejšie a s použitím menšieho množstva pamäte. Tradične je trojuholník definovaný troma bodmi. Definovanie N trojuholníkov týmto spôsobom vyžaduje definovanie 3N bodov, usporiadaných do trojíc (ktoré tvoria trojuholníky). V prípade trojuholníkových pásov definujeme troma bodmi len prvý trojuholník, každé ďalšie definujeme jediným ďalším bodom. Zvyšné dva body zdieľa s predošlým zadaným trojuholníkom. Potrebný počet definovaných bodov je teda len N + 2.



Obr. 6.2: Trojuholníkový pás ABCDEFGH

Napríklad, šesť trojuholníkov na obrázku 6.2 môžeme pomocou trojuholníkových pásov zadať jednoducho ako sekvenciu bodov ABCDEFGH.

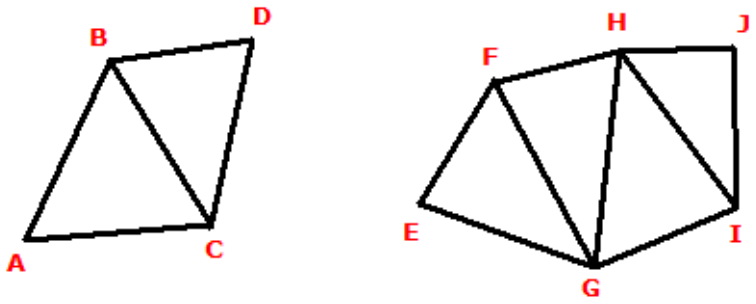
Viacero trojuholníkových pásov je možné spojiť za sebou. Výhoda spočíva v tom, že nemusíme opakovane volať príkaz pre renderovanie[3] každého pásu, čím opäť zvyšujeme rýchlosť renderovania. Spojenie dvoch trojuholníkových pásov realizujeme zopakovaním posledného vrcholu prvého pásu, začiatočného vrcholu druhého pásu a spojením oboch pásov. Tým vytvoríme dva degenerované trojuholníky[4], ktoré dokážu grafické karty veľmi rýchlo vyradiť z renderovania a výsledný spojený trojuholníkový pás bude zobrazený bez akýchkoľvek artefaktov.

Na obrázku 6.3 máme dva trojuholníkové pásy definované ako: ABCD, EFGHIJ. Ak chceme tieto pásy spojiť postupujeme nasledovne. Zopakujeme posledný vrchol prvého pásu (D) a prvý vrchol druhého pásu (E) a oba pásy spojíme. Výsledný trojuholníkový pás je teda: ABCDDEEFGHIJ.

6.3 Vertex Buffer Object

Vertex Buffer Object (VBO) je v súčasnosti najrýchlejší spôsob renderovania množiny trojuholníkov [12]. Použitím tohto spôsobu je možné odoslať na pamäť grafickej karty požadovanú množinu trojuholníkov, ktoré potom môžeme opakovane vykresliť.

3. renderovanie – zobrazenie, vypočítanie aktívneho pohľadu na scénu
4. degenerovaný trojuholník – trojuholník, ktorý má všetky tri vrcholy ležiace na jednej priamke



Obr. 6.3: Dva samostatné trojuholníkové pásy ABCD, EFGHIJ

Kapitola 7

Popis realizácie a implementácie programu

7.1 Použité technológie a minimálna konfigurácia

Aplikácia je naprogramovaná v C++, vo Visual Studio 2005, s použitím grafickej knižnice OpenGL, multimediálnej knižnice SDL, jazyka Cg, určeného na programovanie shaderov[1] a knižníc Boost (knižnice rozširujúce funkcionalitu jazyka C++). Podmienkou pre spustenie programu je grafická karta, ktorá podporuje aspoň shader model 3.0.

Implementácia šumových funkcií na GPU[2] je jemne upravená implementácia šumu od pána Gustavsona [6]. Textový výstup na obrazovku je implementovaný pomocou knižnice glFont[3]. Použitý generátor náhodných čísel je Mersenne Twister[4]. Aplikácia používa komprimované textúry vo formáte dds[5], načítané programom pána Jonssona[6].

7.2 Stručný popis aplikácie

Aplikácia využíva všetky techniky zobrazovania terénu, popísané v predošlej kapitole. Najprv vygeneruje výškovú mapu terénu podľa aktuálne nastavených parametrov. Samotný proces generovania terénu v prípade šumových algoritmov prebieha na GPU, pretože ide o ľahko paralelizovateľnú úlohu. Algoritmus Diamant-Štvorec je naprogramovaný štandardne na CPU[7]. Následne aplikácia vypočíta normálové vektory pre vygenerovaný terén.

Zobrazený terén je osvetlený jednoduchým svetelným modelom zloženým z ambientného[8] a difúzneho[9] svetla. Aplikácia umožňuje zobrazovanie terénu so základnými textúrami trávy a kameňa. Rozloženie týchto textúr je dané vlastnosťami povrchu, na ktorý sú nanášané – tráva sa nachádza na miestach, ktoré nemajú veľký sklon, naopak kameň sa nachádza na prudkých zrázoch a úpätiach vrchov.

---

1. shader – program určený pre spracovanie priamo na grafickej karte
2. GPU – graphics processing unit – grafická karta
3. glFont – knižnica na zobrazovanie textu pre OpenGL programy. Viac na <http://students.cs.byu.edu/~bfish/glfont.php>.
4. Mersenne Twister – generátor náhodných čísel. Vyznačuje sa vysokou rýchlosťou generovania a dlhou periódou. Viac na <www.agner.org/random>.
5. dds – DirectDraw Surface, formát umožňujúci ukladanie komprimovaných textúr
6. Viac na <http://www.codesampler.com/usersrc/usersrc_7.htm>.
7. CPU – central processing unit – procesor
8. ambientné svetlo – trvalé, všesmerové svetlo
9. difúzne svetlo – smerové svetlo, prichádzajúce z nejakého zdroja svetla

V prípade, že užívateľ zmení nejaké parametre generovaného terénu, aplikácia okamžite vygeneruje nový terén, zodpovedajúci týmto parametrom a tento terén zobrazí. Všetky zmeny sa dejú v reálnom čase, výpočet je rýchly, a je tak možné okamžite sledovať, ako parametre ovplyvňujú charakteristiku generovaného terénu.

Základné nastavenia rozmerov okna aplikácie a kvality terénu je možné nastaviť ešte pred spustením programu v externom súbore settings.txt.

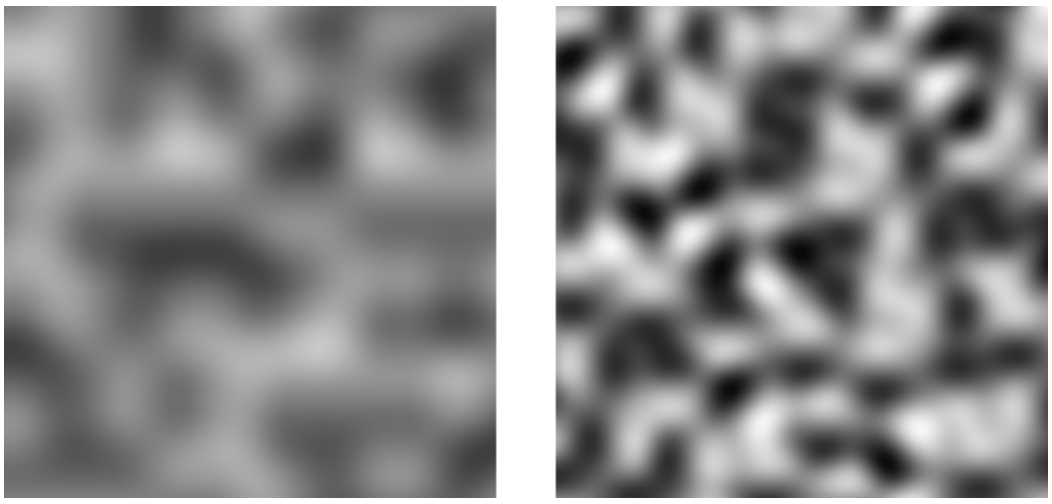
Kapitola 8

Porovnanie algoritmov na generovanie terénu

8.1 Porovnanie Perlinovho a Simplexného šumu

Obidva algoritmy sa používajú v súčasnej dobe ako základné procedurálne primitíva. Mohlo by sa zdať, že Simplexný šum prevažuje svojimi výhodami nad klasickým Perlinovým šumom, no nie je to celkom tak. Hoci časová zložitosť Simplexného šumu je menšia ako Perlinovho šumu, pre jednu alebo dve dimenzie je Perlinov šum stále rýchlejší. Pre tri dimenzie sú výsledky takmer totožné a vo všetkých vyšších dimenziách už dominuje Simplexný šum [6].

Na obrázku 8.1 je zobrazený dvojrozmerný Perlinov šum a dvojrozmerný Simplexný šum. Hoci sú obidva šumy vytvorené použitím rovnakého parametra seed, sú rozdielne, čo je spôsobené odlišnou metódou výpočtu šumu. Nie je ich teda možné priamo porovnať[1], ako ostatné fraktálne algoritmy.



Obr. 8.1: Perlinov šum (vľavo) a Simplexný šum (vpravo)

Napriek tomu jasne vidíme vizuálne rozdiely medzi obidvomi šumami. Hoci obidva šumy majú charakteristiky ideálnej šumovej funkcie, Simplexný šum je kontrastnejší než

1. Pod pojmom „priame porovnanie“ rozumieme vizuálne porovnanie výstupu dvoch veľmi podobných algoritmov, pracujúcich zväčša s rovnakými vstupnými dátami (rovnaká šumová funkcia, vstupné parametre), no produkujúcich rozdielne výstupy. Práve táto rozdielnosť výstupov je predmetom porovnania.

Perlinov. To nie je na škodu, no aplikácie, ktoré sú založené na presných štatistických charakteristikách pôvodného Perlinovho šumu, budú zrejme musieť byť upravené, ak chcú používať Simplexný šum.

Nemožno povedať, že jeden šum prevláda nad druhým. Perlinov šum je pre nízky počet dimenzií rýchlejší než Simplexný šum, a preto sa stále využíva. Širšiemu používaniu Simplexného šumu zabraňuje aj to, že vykazuje iné štatistické charakteristiky ako pôvodný Perlinov šum. Algoritmy využívajúce šumové funkcie sú väčšinou veľmi citlivé na akúkoľvek zmenu parametrov a vyžadujú úpravu. V aplikáciách, kde rozhoduje rýchlosť počítania šumu a zároveň je nutné použiť šum vo vyšších dimenziách, je ale Simplexný šum ideálnym riešením.

8.2 Porovnanie fraktálnych algoritmov

Každý popísaný algoritmus generuje terén iných charakteristík. Algoritmy fBm, Heterofractal a Hybrid multifractal je možné porovnávať priamo, pretože všetky využívajú rovnakú šumovú funkciu a výsledný terén generujú podobným spôsobom. Algoritmus Diamant-Štvorec generuje terén veľmi podobných charakteristík ako terén generovaný pomocou fBm, no nie je možné ich porovnať priamo, keďže ide o diametrálne odlišné algoritmy. Ridged multifractal používa prevrátenú absolútnu hodnotu šumovej funkcie, a preto má výsledný terén úplne iné charakteristiky ako terén generovaný ostatnými algoritmami.

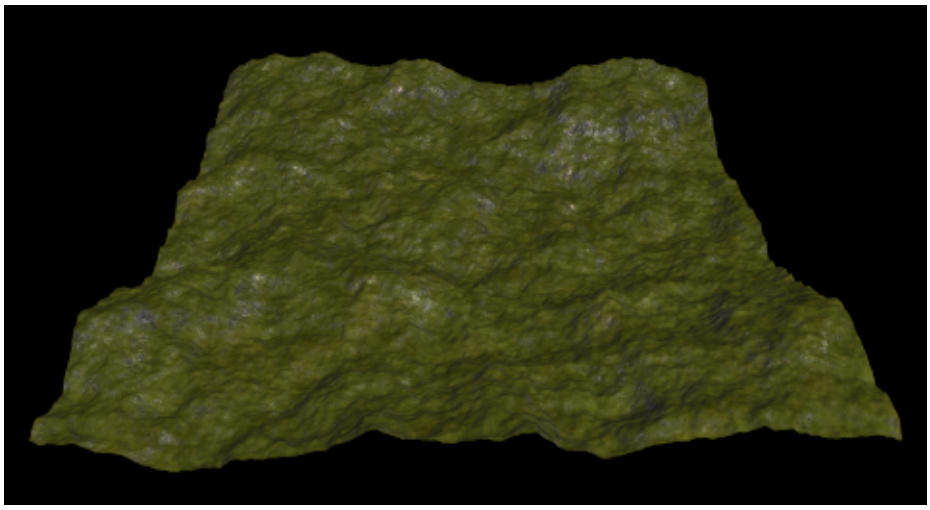
Všetky obrázky v tejto kapitole sú vytvorené pomocou priloženého programu. Pokiaľ nie je uvedené inak, vygenerovaný terén má nastavené počiatočné vstupné parametre, definované programom, okrem parametra seed. Ten je nastavený na hodnotu 3. Použitá šumová funkcia je Perlinov šum.

Začnime s najjednoduchším fraktálnym algoritmom – fBm. Terén generovaný pomocou fBm je homogénny. Jasne to môžeme vidieť na obrázku 8.2. Všimnime si, že terén je rovnako zvrásnený vo všetkých miestach, nezávisle na výške, či iných parametroch. V teréne sa nenachádzajú žiadne výrazné pohoria alebo údolia, nič, čo by sa nejakým výrazným spôsobom odlišovalo od okolitého povrchu. Podobne vyzerá aj terén generovaný algoritmom Diamant-Štvorec (obrázok 8.3).

Ani zmenou vstupných parametrov nedokážeme výrazne zmeniť vygenerovaný terén. Je to preto, lebo tieto parametre ovplyvňujú globálne vlastnosti terénu. Môžeme napríklad zväčšiť parameter H a dostaneme tak celkovo hladší terén, no tým sa vyhladia nielen údolia v teréne, ale aj pohoria a iné vyvýšené časti terénu.

Multifraktály nám umožňujú lepšie zachytiť reálne vlastnosti terénu. Terén generovaný algoritmom Heterofractal vychádza z algoritmu fBm, no zvrásnenie terénu je závislé od jeho výšky nad morom. Keďže algoritmy využívajú rovnakú šumovú funkciu, je možné ich priamo porovnať. To sa prejavuje rovnakým rozmiestnením veľkých terénnych útvarov ako sú pohoria a údolia, čo vidíme na obrázku 8.4. Líšia sa len lokálnou úrovňou zvrásnenia povrchu.

Všimnime si, že údolia sú teraz výrazne hladšie, ale vrchy drsnejšie. Výsledný terén vyzerá realistickejšie. Zatiaľ čo v prípade terénu generovaného pomocou fBm boli jednotlivé



Obr. 8.2: Terén generovaný pomocou algoritmu fBm

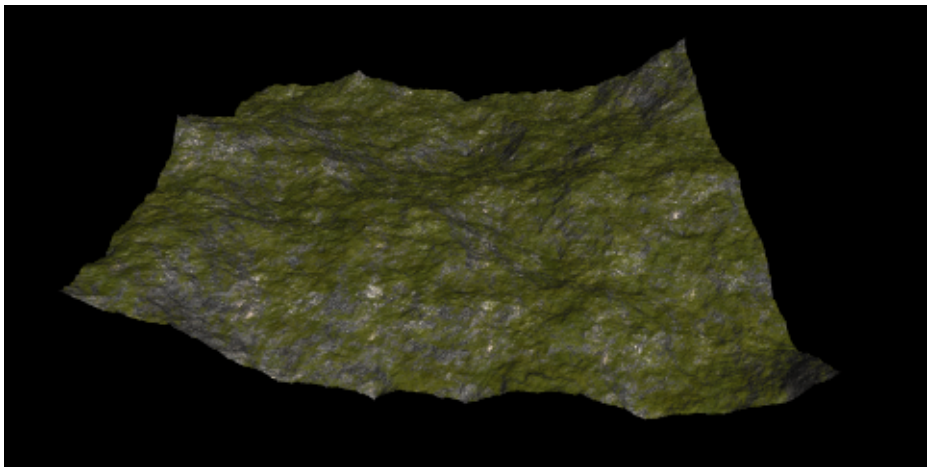
pohoria a údolia akoby „zliate“ do seba, v tomto prípade je možné identifikovať jednotlivé pohoria jednoduchšie a sú výrazne odlišné od okolitého terénu.

Ďalším algoritmom, ktorý dosahuje podobné výsledky, je Hybrid multifractal. Tento algoritmus je opäť odvodený z fBm, no požadovaný efekt vyhladenia údolí dosahuje iným spôsobom (obrázok 8.5).

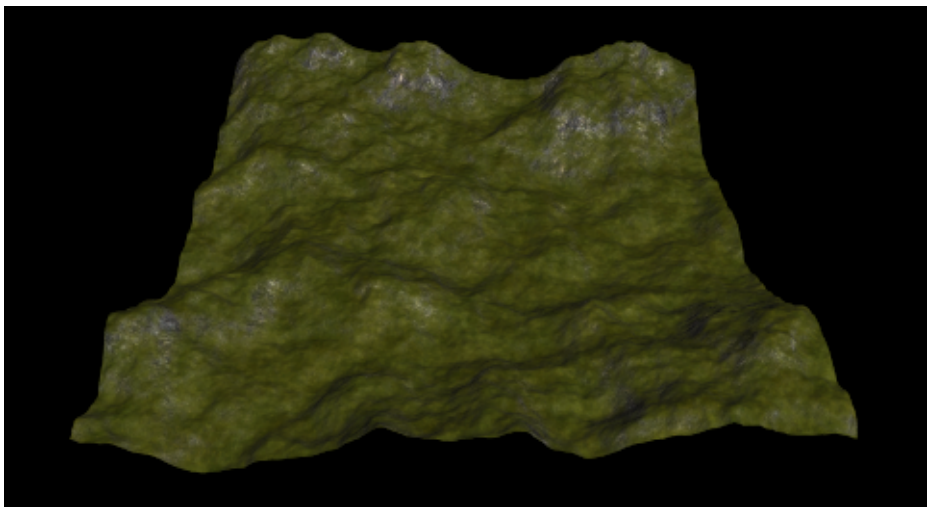
Výsledný efekt je ešte väčšie vyhladenie nízko položených častí terénu a približne rovnaké zvrásnenie terénu vo vyšších výškach. Formujú sa takto veľmi hladké údolia a pláne. Generovaný terén je opäť o niečo bližšie realite.

Všimnime si ale, že všetky doposiaľ vygenerované pohoria sú vlastne len osamelé vrchy a pahorkatiny. Reálny terén avšak obsahuje mohutné pohoria, rozpínajúce sa po celej krajine. Takéto pohoria majú niekoľko vrchov, štítov, horských hrebeňov a sediel. Ridged multifractal je šum, ktorý bol navrhnutý aby generoval práve takéto pohoria. To dosahuje tak, že využíva pozmenenú šumovú funkciu $1 - \text{abs}(\text{noise})$. Na obrázku 8.6 je zobrazená časť terénu generovaného algoritmom Ridged multifractal.

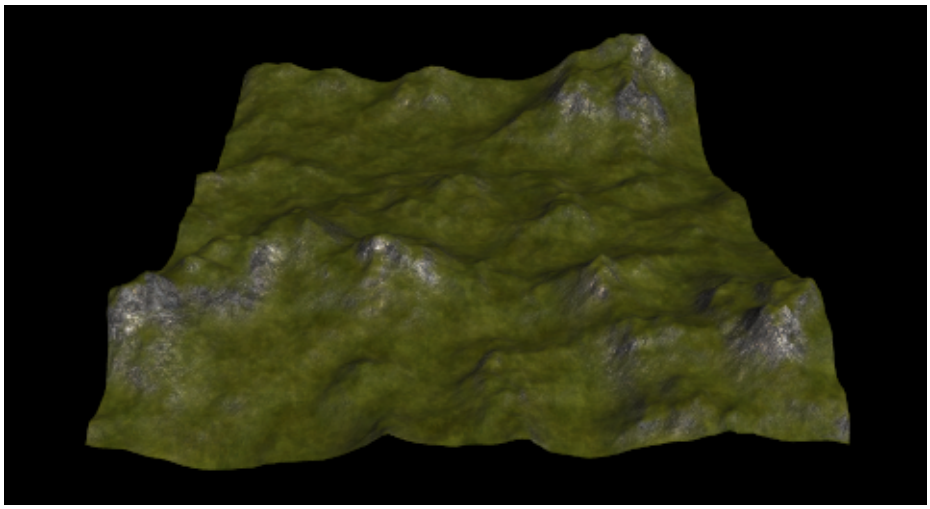
Ako vidíme, nielenže tento algoritmus generuje realistické pohoria, ale zachováva si aj multifraktalitu a generovaný terén obsahuje okrem vrchov a štítov aj horské pláne, nížiny a roviny. Detailnejšie to môžeme vidieť na obrázku 8.7, kde je zobrazený ten istý terén, no z väčšej blízkosti a sú vypnuté textúry, aby bol jasnejšie viditeľný tvar terénu.



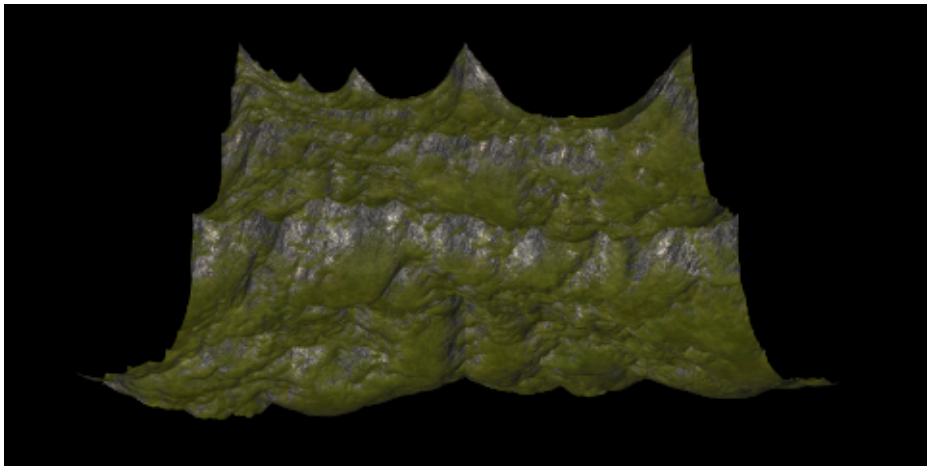
Obr. 8.3: Terén generovaný pomocou algoritmu Diamant-Štvorec



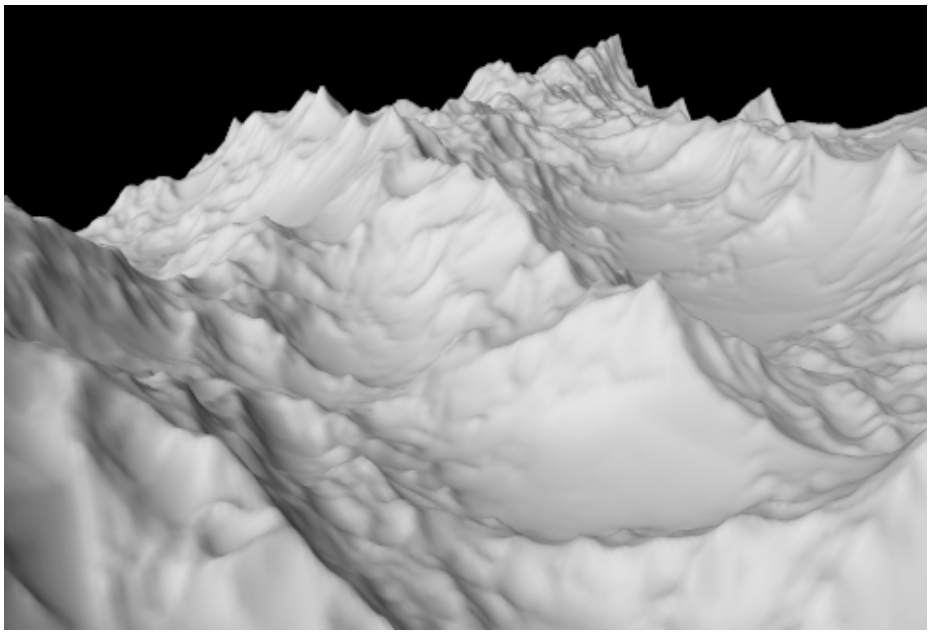
Obr. 8.4: Terén generovaný pomocou algoritmu Heterofractal



Obr. 8.5: Terén generovaný pomocou algoritmu Hybrid multifractal



Obr. 8.6: Terén generovaný pomocou algoritmu Ridged multifractal



Obr. 8.7: Časť terénu generovaného pomocou algoritmu Ridged multifractal s vypnutými textúrami

Kapitola 9

Záver

V práci sme sa zoznámili so súvislosťou medzi prírodnými materiálmi a útvarmi, kam patrí aj terén, a fraktálmi. Túto spojitosť sme následne využili pri generovaní terénu pomocou procedurálnych algoritmov. Veľká časť týchto algoritmov patrí do kategórie šumových fraktálnych algoritmov. Tieto algoritmy využívajú na generovanie terénu šumovú funkciu. V práci sme si popísali vlastnosti takejto funkcie a uviedli dve, v súčasnosti najpoužívanejšie.

Následne sme sa už venovali popisu jednotlivých algoritmov, procedurálne generujúcich terén. Väčšina priestoru bola venovaná algoritmom založených na šume, no predstavili sme si aj veľmi známy fraktálny algoritmus Diamant-Štvorec. Algoritmy sme porovnali, k čomu nám pomohol program, ktorý je súčasťou tejto práce.

Zistili sme, že monofraktály nedostatočne vystihujú vlastnosti reálneho terénu. Multifraktálne algoritmy produkujú reálnejší terén, no stále nedokážeme zachytiť všetky charakteristiky terénu v jednom fraktálnom procedurálnom modeli. Napríklad, algoritmus Hybrid multifractal generuje vierohodné vrchy, pahorkatiny a údolia, no algoritmus Ridged multifractal generuje realistické pohoria a horské štíty.

Algoritmy prezentované v tejto práci dokážu relatívne dobre zachytiť daný typ terénu, pre ktorý boli navrhnuté. Budúce vylepšenia týchto algoritmov môžu zahrňovať nové typy generovaného terénu, prípadne ich kombinácia do jediného algoritmu, schopného generovať značne heterogénny terén.

Literatúra

[1] EBERT, D.: *Texturing & Modeling - A Procedural Approach*, Morgan Kauffman Publishers, 2003, ISBN 1-55860-848-6, 687 s..

[2] PERLIN, K.: *An Image Synthetizer [online]*, ACM, 1985, Dostupné na: <http://doi.acm.org/10.1145/325165.325247> [cit. 2010-05-16].

[3] MANDELBROT, B.: *How Long Is the Coast of Britain [online]*, 1967, Dostupné na: <http://www.math.yale.edu/mandelbrot/web_pdfs/howLongIsTheCoastOfBritain.pdf> [cit. 2010-05-16].

[4] RICHARDSON, L.: *The Problem of Contiguity*, General systems: Yearbook of the Society for the Advancement of General Systems Theory, 1961.

[5] PERLIN, K.: *Noise Hardware [online]*, 2001, Dostupné na: <http://www.csee.umbc.edu/~olano/s2002c36/ch02.pdf> [cit. 2010-05-16].

[6] GUSTAVSON, S.: *Simplex Noise Demystified [online]*, 2005, Dostupné na: <http://webstaff.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf> [cit. 2010-05-16].

[7] PERLIN, K.: *Improving Noise [online]*, 2002, Dostupné na: <http://mrl.nyu.edu/~perlin/paper445.pdf> [cit. 2010-05-16].

[8] WARD, M.: *Fractional Brownian Motion [online]*, 1996, Dostupné na: <http://davis.wpi.edu/~matt/courses/fractals/brownian.html> [cit. 2010-05-16].

[9] MARTZ, P.: *Generating Random Fractal Terrain [online]*, 1997, Dostupné na: <http://www.gameprogrammer.com/fractal.html> [cit. 2010-05-16].

[10] Heightmap In Wikipedia: the free encyclopedia [online], Wikipedia Foundation, 2010, Dostupné na: <http://en.wikipedia.org/wiki/Heightmap> [cit. 2010-05-16].

[11] Triangle Strip In Wikipedia: the free encyclopedia [online], Wikipedia Foundation, 2010, Dostupné na: <http://en.wikipedia.org/wiki/Triangle_strip> [cit. 2010-05-16].

[12] BAVOIL, L.: *Rendering Huge Triangle Meshes with OpenGL [online]*, 2005, Dostupné na: <http://www.sci.utah.edu/~bavoil/opengl/bavoil_trimeshes_2005.pdf> [cit. 2010-05-16].

Dodatok A

Obsah CD

Súčasťou tejto práce je CD. Obsah:

- Elektronická verzia tejto práce vo formáte pdf.
- Zdrojové kódy aplikácie určenej na generovanie a zobrazovanie časti terénu v reálnom čase.
- Spustiteľný súbor vyššie uvedenej aplikácie.