Vysoké učení technické v Brně
Fakulta informačních technologií

Vybrané fraktálové algoritmy a jejich použití

Bakalářská práce

Bedřich Michálek

2006

Prohlášení

Prohlašuji, že tato práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Bedřich Michálek

Vedoucí projektu: Herout Adam, Ing., Ph.D.

Abstrakt

Cílem projektu bylo nastudovat problematiku fraktálových algoritmů, vytvořit jednoduchou demonstrační aplikaci nastudovaných algoritmů a navrhnout možnosti dalšího zkoumání.

Protože oblast fraktálů a fraktálových algoritmů je velice rozsáhlá, zaměřil jsem se pouze na tři konkrétní typy - Mandelbrotovu množinu a s ní související Juliovy množiny, systémy iterovaných funkcí za použití různých algoritmů pro generování a stochastické fraktály a jejich využití při generování modelů travin, keřů a plasmy.

Při studiu a implementaci algoritmů pro generování jsem se snažil o jistou míru jejich optimalizace, urychlení a srovnání z hlediska rychlosti a využití paměti, třebaže ne vždy to je z povahy daného typu fraktálu možné. Teoreticky také uvádím možnosti využití technologie SSE pro urychlení výpočtu fraktálových algoritmů.

Demonstrační aplikace je navržena velice jednoduše, protože cílem nebylo vytvořit nový propracovaný program na generování fraktálů, ale vyzkoušet implementovat prostudované algoritmy. Z toho důvodu jsem se spíše zaměřil na přehlednou implementaci, aby bylo možné tyto algoritmy kdykoliv jednoduše použít. Aplikace je napsána v jazyce C/C++ s použitím knihovny SDL (Simple Directmedia Layer) pro práci s grafikou.

Klíčová slova

fraktál, Mandelbrotova množina, systémy iterovaných funkcí, stochastické fraktály, celočíselná aritmetika, tesseral, solid-guessing, RWA, DIA, difůze, SSE, optimalizace

Abstract

To read up on the problems of fractal algorithms, programme a simple demo application of this algorithms and propose alternatives of a further study was the goal of my project.

Since the subject of fractals and fractal algorithms is very extensive, I concentrated on three specific types of them - the Mandelbrot set and the Julia sets related to it, iterated function systems with application of different algorithms for generating, and the stochastic fractals and their utilization for grass, shrub and plasma modelling.

I tried out for a certain rate of their optimalization, acceleration and the comparison of time and memory requirements in my study and application although it is not ever possible due to the characteristic of the given fractal type. I also theoretically present some utilization options of the SSE technology for the acceleration of fractal algorithm calculations.

The demo application is proposed in the very easy way because there was not the goal to programme a new carefully worked-out software, but rather to try to implement the read up algorithms. For this reason I focused on the clear implementation of these algorithms to be their utilization easily possible. The application is written in C/C++ language with SDL library used to handle the graphics output.

Key words

fractal, Mandelbrot set, iterated function systems, stochastic fractals, fixed-point arithmetic, tesseral, solid-guessing, RWA, DIA, diffusion, SSE, optimalization

Obsah

Úvod

Klasická euklidovská geometrie se zabývá studiem ideálních pravidelných útvarů jako čtverce, kružnice, pravidelné mnohoúhelníky a mnohostěny. Naproti tomu fraktální geometrie studuje tvary tak členité jako třeba hory, pobřeží, mraky, elektrické výboje, stromy, špinavé skvrny a podobně. Odtud plyne její praktické využití v počítačové grafice.

První kapitola obsahuje stručný úvod do fraktální geometrie a pojmů s tím souvisejících, jako je definice pojmu fraktál, rozdíl mezi topologickou a Hausdorffovou dimenzí, soběpodobnost fraktálních útvarů, co je to atraktor systému a rozdělení fraktálních objektů do základních kategorií.

Druhá kapitola popisuje některé metody, jak urychlit generování fraktálního obrazce. Metody jsem rozdělil do dvou kategorií, a to metody pro urychlení výpočtu, kde popisuji metodu detekce periodické posloupnosti, transformaci výpočtu do aritmetiky s pevnou řádovou čárkou a teoretické využití technologie SSE, a metody pro urychlení vykreslení, kde popisuji metodu solid-guessing a tesseral. Úmyslně tuto kapitolu uvádím před vlastním popisem vybraných fraktálů, protože se některé metody vztahují k více typům fraktálů. Zatímco například metoda detekce periodické posloupnosti, solid-guessing a tesseral je aplikovatelná pouze na Mandelbrotovu množinu, transformaci výpočtu do aritmetiky s pevnou řádovou čárkou lze aplikovat jak na Mandelbrotovu množinu, tak i na systémy iterovaných funkcí.

Další kapitoly podrobně popisují prostudované typy fraktálů - Mandelbrotovu množinu, systémy iterovaných funkcí a stochastické fraktály. U každého typu se snažím uvést stručný matematický popis a několik algoritmů, které lze využít pro generování.

Poslední kapitola popisuje experimentálně získané výsledky srovnání metod pro generovaní jednotlivých typů fraktálů a stručný popis demonstrační aplikace.

Kapitola 1

Stručný úvod do fraktální geometie

1.1 Co je to fraktál

Termín fraktál poprvé uveřejnil ve své práci v roce 1975 polský matematik Benoit Mandelbrot. Tento termín je vytvořen z latinského slova *fractus*, které znamená "rozlámaný" nebo "rozbitý".

Fraktál jako geometrický objekt splňuje následující vlastnosti:

- má jasnou strukturu při jakékoliv změně měřítka,
- je příliš nepravidlený, aby byl snadno popsán klasickou Euklidovskou geometrií,
- je soběpodobný (alespoň přibližně nebo náhodně),
- jeho Hausdorffova dimenze je větší jak topologická,
- má jednoduchou a rekurzivní definici.

Protože se fraktály jeví soběpodobné při jakékoliv míře zvětšení, jsou často považovány za objekty "nekonečně komplexní".

1.2 Dimenze

Důležitou součástí při zkoumání fraktálů je jejich dimenze, a to jak dimenze topologická, tak i fraktální (nazývaná také Hausdorffova dimenze).

Topologická dimenze

Geometricky hladké objekty, které je možné popsat klasickou Euklidovskou geometrií, mají celočíselnou dimenzi, nazývanou také topologická dimenze. Lze říci, že topologická

dimenze určuje počet parametrů (nezávislých proměnných), kterými je možné dané těleso (resp. každý bod na tělese) popsat. Například bod má nulovou dimenzi, jelikož je sám popsán vztahem $P = X$ (tj. konstantním vektorem), zatímco úsečka má dimenzi rovnu jedné, neboť ji lze popsat vztahem $y(t) = A + t\overrightarrow{u}$, kde t je jediný parametr (nezávislá proměnná). Pozici každého bodu ležícího na úsečce lze vyjádřit výše uvedeným vztahem.

Z výše uvedeného vyplývá, že úsečka, přímka či jiná křivka (například parabola, sinusoida či Bézierova křivka) má dimenzi rovnu 1. To znamená, že je jednorozměrná, a tudíž poloha bodu je na ní definována pouze jedním číslem - souřadnicí v parametrickém prostoru. Pro křivky s topologickou dimenzi rovnou jedné je definována jejich délka (která může být i nekonečná), ale jejich plocha je nulová (jsou nekonečně tenké).

Jakákoliv hladká plocha (kruh, trojúhelník, n-úhelník) má dimenzi rovnu dvěma, to znamená, že poloha bodu musí být určena pomocí dvou parametrů. Takto definované plochy mají určitý obsah, ale jejich objem je nulový, protože mají nulovou tloušťku.

Krychle, (vyplněná) koule, válec nebo celý běžný prostor kolem nás mají dimenzi rovnu třem, protože poloha jakéhokoli bodu je v nich jednoznačně určena třemi parametry.

Zjednodušeně lze říci, že topologická dimenze je taková dimenze, která je specifikována celým číslem.

Hausdorffova dimenze

K "měření" soběpodobných útvarů již nelze použít prostředky klasické geometrie. Proto se jako míra členitosti definuje tzv. Hausdorffova dimenze. Její definice je složitá, ale pro dosti velkou třídu soběpodobných množin lze Hausdorffovu dimenzi spočítat podle jednoduchého vzorce. V případě, že zkoumaný objekt obsahuje n kopií sebe sama zmenšených na jednu *k-tinu*, je Hausdorfova dimenze rovna:

$$dim_{Haus} = \frac{log\ n}{log\ k} \tag{1.1}$$

Jedním z nejjednodušších fraktálních útvarů je Cantorovo diskontinuum. Úsečka libovolné délky se rozdělí na třetiny a prostřední část se smaže. Získají se tak dvě úsečky, s nimiž se postup opakuje. Když se toto bude provádět až donekonečna, získá se množina nekonečně mnoha bodů, jejichž celková délka je 0. To je Cantorovo diskontinuum. Taková množina obsahuje dvě kopie sebe sama zmenšené na třetinu. Hausdorfova dimenze Cantorova diskontinua je tedy $dim_{Haus} = \frac{log\ 2}{log\ 3} \doteq 0.6309$.

1.3 Soběpodobnost

Soběpodobnost (matematicky se tato vlastnost nazývá invariance vůči změně měřítka) je taková vlastnost objektu, že objekt vypadá stejně, ať se na něj díváme v jakémkoliv zvětšení. Jinými slovy, kterákoliv část fraktálu je přesnou (nebo alespoň dosti podobnou) kopií původního motivu. Soběpodobnost je hlavním znakem fraktálních útvarů a

většinou je také považována za jejich definici. Tato vlastnost se vyskytuje jednak u čistě matematických struktur, ale v určité míře i u přírodních objektů (např. list kapradiny). V přírodě jsme jednak omezeni velikostí částic a jednak těžko v přírodě vznikne takto dokonalý fraktál.

Matematicky je soběpodobná množina definována takto: Soběpodobná množina A n-dimenzionálního Euklidovského prostoru $\mathbf{E^n}$ je taková množina, pro níž existuje konečně mnoho kontrahujících zobrazení $f_1 ... f_n$ takových, že A vznikne jako:

$$A = \bigcup_{i=1}^{n} f_i(A) \tag{1.2}$$

Obrázek 1.1: Příklad soběpodobného IFS fraktálu modelujícího list kapradiny

1.4 Atraktor

Při popisu dynamických systémů a systémů iterovaných funkcí (IFS) hraje důležitou roli pojem atraktor (*attractor*). Zjednodušeně lze atraktor dynamického systému definovat jako množinu stavů, do kterých systém směřuje. Jedná se o množinu hodnot, kterých může nabývat stavový vektor dynamického systému po dostatečně dlouhém časovém úseku od počátečního impulzu. Atraktory lze rozdělit do několika tříd:

- Atraktorem může být množina pevných bodů - systém se v nekonečném čase ustálí v nějakém stavu, který je možné předem vypočítat.
- Atraktorem může být množina periodických bodů - systém se po určité době ustálí tak, že osciluje mezi několika stavy, které mohou být buď spočitatelné nebo nespočitatelné.
- Atraktor je chaotický - výsledný stav systému nelze dopředu předpovědět.
- Atraktor je "podivný" (*strange attractor*) - z hlediska fraktální geometrie nejzajímavější případ atraktoru. Tento typ může vzniknout tehdy, je-li systém popsán

minimálně třemi navzájem souvisejícími diferenciálními rovnicemi. Takový systém může mít velmi komplikovaný atraktor, který sice bude vykazovat vlatnosti pravidelného, ale současně i chaotického atraktoru. Podivný atraktor není ještě přesně matematicky definován, ale považuje se za něj takový atraktor, který vykazuje vlastnosti do jisté míry shodné s těmi, jaké mají fraktály. Platí, že všechny chaotické atraktory jsou současně podivnými atraktory, opačná implikace však neplatí.

1.5 Typy fraktálních objektů

I ve fraktální geometrii se rozlišují jednotlivé typy fraktálů, přičemž fraktály stejného typu mají shodné své nejvýznamější charakteristiky. Toto rozdělení typů je zavedeno proto, že jednotlivé typy fraktálů jsou vhodné pro řešení určitého okruhu problémů, které se vyskytují v jednom nebo více oborech. Také způsob generování fraktálů se liší podle typu fraktálu, přičemž fraktály stejného typu se většinou generují s využitím poobných algoritmů.

Nejobecněji lze fraktální typy rozdělit následovně:

- dynamické systémy s fraktální strukturou,
- systémy iterovaných funkcí,
- stochastické fraktály (nepravidelné),
- L-systémy.

Protože je oblast fraktální geometrie neuvěřitelně rozsáhlá, zaměřil jsem se ve svém studiu pouze na první tři typy fraktálních objektů.

Dynamické systémy s fraktální strukturou

Dynamické systémy tvoří skupinu fraktálů, která má v technické praxi nejširší uplatnění. Dynamický systém je model, jehož stav je závislý na nějaké nezávislé veličině, například na čase. Dynamický systém je popsán pomocí dynamických podmínek, které popisují změnu systému v čase. Dynamické podmínky jsou většinou zadány soustavou diferenciálních rovnic, které popisují změnu stavového prostoru v čase.

Z hlediska počítačové grafiky je nejzajímavější vhodným způsobem vizualizovat nějakou vlastnost dynamického systému, například počet iterací systému v komplexní rovině.

Dynamických systémů existuje nepřeberné množství, proto jsem si jako zástupce této skupiny pro bližší studium zvolil pouze Mandelbrotovu množinu.

Systémy iterovaných funkcí

V systémech iterovaných funkcí se pro tvorbu fraktálních obrazců používá takzvaná generativní metoda. Tato metoda je vhodná jak pro generování fraktálů, tak i například pro kompresi dat. Zajímavé je, že metoda generování fraktálů pomocí systémů iterovaných funkcí může být jak plně deterministická, tak i stochastická (tj. při výpočtech se

využívají náhodné prvky). Oboje však paradoxně vede ke stejnému výslednému fraktálu, ovšem za použití dostatečného počtu iterací.

Stochastické fraktály

Zatímco předchozí skupiny fraktálů jsou v určitém smyslu symetrické, stochastické (nepravidelné) fraktály vnášejí do generování fraktálu náhodu. Tento typ také umožňuje nejlepší popis přírodních objektů, a proto se často používá pro generování různých modelů travin, keřů a stromů, stejně jako povrchů planet a krajin.

1.6 Využití fraktální geometrie

Objekty s fraktální strukturou lze objevit v živé i neživé přírodě, stejně jako se vyskytují například v geometrii, fyzice či chemii. Největší uplatnění však nacházejí v počítačové grafice, například při procedurálním generování pravidelných i nepravidelných těles, přírodních útvarů (stromy, rostliny, hory, mraky, ...) či fraktálové kompresi dat.

Kapitola 2

Optimalizace fraktálových algoritmů

Vzhledem k tomu, že fraktální obrazec se může skládat ze statisíců pixelů a pro každý pixel je třeba provést mnohdy až několik stovek iterací, může být vykreslení fraktálního obrazce velmi časově náročné. Proto se s rozvojem fraktální geometrie objevili i metody, jak urychlit vlastní výpočet nebo vykreslení obrazce. Urychlení se dá obecně provést ve třech oblastech:

- urychlení výpočtu barvy daného pixelu,
- urychlení vykreslení redukcí celkového počtu pixelů, jejichž barvu je nutné získat iterativním způsobem,
- redukce počtu pixelů při dynamické změně pohledu na fraktální obrazec.

Zmíním se pouze o prvních dvou oblastech, a to o metodách urychlení výpočtu a metodách urychlení vykreslení.

2.1 Metody urychlení výpočtu

Metody urychlení výpočtu se zaměřují na urychlení časově náročného iteračního procesu, ať už na úrovni algoritmu nebo na úrovni procesoru využitím celočíselné aritmetiky nebo speciálních SIMD instrukcí.

Detekce periodické posloupnosti

Nejjednodušší metodou pro urychlení výpočtu barvy jednoho pixelu je metoda detekce periodické posloupnosti. Pokud při výpočtu dojde k situaci, že se nějaká hodnota Z_i bude po několika iteracích opakovat, tj. bude platit $Z_i+\delta=Z_i$, je jasné, že daný bod musí ležet uvnitř fraktální množiny. Pokud budeme schopni tuto periodu detekovat, může výrazným způsobem dojít ke zrychlení iteračního procesu - o tom, zda bod leží uvnitř nebo vně fraktální množiny bude možno rozhodnout výrazně dříve než po dosažení

maximálního počtu iterací. Možností, jak detekovat periodickou posloupnost, je několik. Lze si zapamatovat všechny již spočítané honoty Z_i a novou hodnotu vždy porovnat s hodnotami předešlými. Kromě velké paměťové náročnosti by se v každém kroku iterace musel lineárně prohledávat seznam předchozích hodnot, což by vedlo spíše ke zpomalení výpočtu.

Více použitelný (rychlý a méně paměťově náročný) způsob je počítání dvou posloupností Z_i a Z'_i zároveň. Hodnota prvků z první posloupnosti je počítána v každém iteračním kroku, zatímco prvky druhé posloupnosti jsou počítány s mnohem nižší frekvencí. V každém iteračním kroku jsou obě hodnoty porovnány a v případě, že si jsou velmi blízké, znamená to, že jsme našli periodickou posloupnost. Velkou výhodou tohoto postupu je, že periodické posloupnosti jsou nalezeny v relativně malém počtu iterací (průměrně 20), což je podstatně méně než maximální počet iterací (např. 256, 1024 či více).

Aritmetika v pevné řádové čárce

Aritmetika v pohyblivé řádové čárce dovoluje pracovat s čísly ve velkém rozsahu, od čísel velmi malých až po čísla velmi velká. I v případě využití matematického koprocesoru může počítání s čísly v pohyblivé řádové čárce zabrat nezanedbatelnou dobu. Pokud jsme ochotni obětovat velký rozsah reálných čísel, lze výrazného zrychlení výpočtu docílit pevnou pozicí řádové čárky a použitím celočíselných operací. Tento způsob se nazývá aritmetika v pevné řádové čárce.

Numerické hodnota X zapsaná ve formátu pevné řádové binární čárky se chápe jako podmnožina racionálních čísel, jejíž hodnoty lze vyjádřit vztahem:

$$X_{FX} = a/b, a, b \in Z, b \neq 0, b = 2^k, k \in \mathbf{Z}^+ \tag{2.1}$$

Protože b je celočíselnou mocninou dvojky, určuje jeho hodnota polohu binární čárky v uloženém čísle. Jednou z implementačních podmínek je zachování stejného počtu binárních cifer v každém reprezentovaném čísle. To mimo jiné znamená, že všechna čísla mají řádovou binární čárku umístěnou na stejném místě. V dalším textu bude zápis U(a,b) představovat reprezentaci kladného čísla s pevnou řádovou binární čárkou, zatímco A(a,b) bude představovat dvojkový doplněk čísla s pevnou řádovou binární čárkou. Pro bližší vysvětlení těchto pojmů doporučuji prostudovat literaturu [6].

Základní vlastnosti čísel s pevnou řádovou binární čárkou:

- Počet bitů nutných pro uložení číselné hodnoty v reprezentaci U(a,b) je roven hodnotě a+b. V reprezentaci A(a,b) je počet bitů roven hodnotě a+b+1, protože jeden bit je nutné rezervovat pro uložení znaménka.
- Rozsah hodnot čísel ve formátu U(a,b) je vyjádřen vztahem $0 \leq x \leq 2^a - 2^{-b}$, pro reprezentaci A(a,b) vztahem $-2^a \leq x \leq 2^a - 2^{-b}$.
- Platnost operace sčítání či odčítání dvou hodnot $U(a_1,b_1)$ a $U(a_2,b_2)$ lze zaručit pouze tehdy, jestliže $a_1=a_2$ a současně $b_1=b_2$, tj. obě hodnoty jsou uloženy na stejném počtu bitů a poloha řádové čárky je konstantní. Pro hodnoty $A(a_1,b_1)$ a $A(a_2,b_2)$ platí stejné podmínky.

- Rozsah výsledků po operaci sčítání dvou hodnot ve formátu U(a,b) nebo A(a,b) je U(a,b)+U(a,b)=U(a+1,b), resp. A(a,b)+A(a,b)=A(a+1,b), tj. pro uložení výsledku je zapotřebí o jeden bit více.

- Rozsah výsledků po vynásobení dvou čísel $U(a_1,b_1)$ a $U(a_2,b_2)$ je možné vyjádřit jako $U(a_1,b_1)\times U(a_2,b_2)=U(a_1+a_2,b_1+b_2)$. Z tohoto vztahu je vidět, že při násobení se stejnou měrou zvyšuje počet bitů před i za binární čárkou. Násobení dvou čísel $A(a_1,b_1)$, $A(a_2,b_2)$ je rovno $A(a_1,b_1)\times A(a_2,b_2)=A(a_1+a_2+1,b_1+b_2)$. Je nutné ještě vyjádřit znaménko výsledku, tedy rozsah je o jedničku větší jak u předchozího vztahu.

- Při bitovém posunu doleva je výsledek určen jako $U(a,b)\ll n=U(a+n,b-n)$ nebo $A(a,b)\ll n=A(a+n,b-n)$. Při bitovém posunu doprava je formát výsledku obdobný, tj. $U(a,b)\gg n=U(a-n,b+n)$ nebo $A(a,b)\gg n=A(a-n,b+n)$.

Základní aritmetické operace nad čísly v pevné řádové binární čárce:

- Součet dvou čísel lze provést jen v případě, že oba operandy mají stejný formát (tj. shodný počet bitů před a za binární řádovou čárkou). Součet dvou čísel A a B lze zapsat jako $A \times 2^b + B \times 2^b = (A + b) \times 2^b$.

- Pro aritmetickou operaci rozdílu dvou čísel musí být splněny stejné podmínky jako pro součet. Výsledek operace je potom určen jako $A \times 2^b - B \times 2^b = (A - B) \times 2^b$.

- Aritmetická operace součin je poněkud složitější, protože počet bitů výsledku se velmi významně zvyšuje, což může vést k přetečení operace a zneplatnění výsledku. Z tohoto důvodu je možné před operací násobení uměle zmenšit přesnost operandů, například posunutím hodnot o n bitů doprava. Tím ale dojde k nárustu chyby výpočtu. Výsledek operace násobení lze vyjádřit jako $A\times 2^b\times B\times 2^b = A\times B\times 2^{2\times b}$.

- Převod čísla z formátu pohyblivé řádové čárky do formátu pevné řádové čárky je určen vztahem $X_{FX} = X_{FP}\times 2^b$.

- Převod čísla z formátu pevné řádové čárky do formátu pohyblivé řádové čárky je určen vztahem $X_{FP} = X_{FX}/2^b$.

Využití technologie SSE

Další z možností, jak urychlit výpočet fraktálních obrazců, je použití technologie SSE (Streaming SIMD Extensions) a programování algoritmu na úrovni asembleru. SSE technologie umožňuje zpracovat množinu dat během jedné instrukce, kdy množství zpracovaných dat závisí na použitém datovém typu (byte, word, doubleword, float, double). Protože fraktálové algoritmy většinou pracují nad reálnými čísly (32 bitový typ float) a SSE instrukce pracují se 128 bitovými registry, lze počítat 4 body fraktálního obrazce současně. Teoreticky by se mělo tedy dosáhnout čtyřnásobné rychlosti výpočtu. Psání kódu přímo v asembleru dovoluje optimalizovat pořadí SSE instrukcí tak, aby se

co nejvíce odstranily datové závislosti mezi instrucemi a maximálně se využilo paralelního zpracování instrukcí. Velmi detailní zpracování této problematiky je uvedeno v [7], kde jsou popsány a vzájemně porovnány tři algoritmy pro generování Mandelbrotovy množiny pomocí SSE instrukcí.

2.2 Metody urychlení vykreslení

Metody urychlení vykreslování se zaměřují na eliminaci počtu pixelů nutných k vykreslení fraktálního obrazce. Je totiž možné využít faktu, že většina fraktálů v komplexní rovině je spojitá a dokonce, že body o stejném počtu iterací tvoří souvislé izoplochy. Nemusí být tedy nutné složitě a především zdlouhavě počítat barvy všech bodů, ale pouze bodů, které fraktál nějakým význačným způsobem reprezentují.

Metoda solid-guessing

Jednou z metod, jak redukovat počet pixelů pro vykreslení, je metoda nazvaná solid-guessing. Tato metoda dělí obrázek na menší části, čtverce o velikosti 2x2 až 16x16 pixelů, a snaží se uhádnout, zda všechny pixely v dané oblasti mají stejnou barvu. Nejdříve zkontroluje zda barva (resp. počet iterací) všech pixelů ležících na hranici dané oblasti je stejná. Pokud ano, všechny pixely uvnitř oblasti jsou obarveny stejně a dále už se s touto oblastí nepočítá. V opačném případě je čtverec rozdělen na čtyři stejné části a proces se rekurzivně opakuje na každé z nich. Takto se postupuje v několika krocích až do chvíle, kdy mají zbylé čtverce velikost 1x1 pixel, u kterých se už musí provést celý iterační výpočet. Tato metoda je použitelná poute pro spojité fraktály, protože u nespojitých fraktálů by mohlo dojít k vynechání částí ležících uvnitř testované oblasti, kdy by všechny body této oblasti měli přiřazenu jednu hodnotu počtu iterací.

Metoda tesseral

Metoda tesseral je variací metody solid-guessing. Jediný rozdíl je ve velikosti testované oblasti. Metoda tesseral začne pracovat s celým obrazcem, který postupně dělí na čtvrtiny, osminy, atd. až dosáhne oblasti, kterou lze vyplnit, nebo limitu 1x1 pixel. Hlavní výhoda metody tesseral oproti metodě solid-guessing je ve zpracování velkých oblastí jedné barvy, jako například vnitřní část Mandelbrotovy množiny, jejíž výpočet při velkém počtu iterací trvá velmi dlouho. Metoda tesseral se potýká se stejným problémem jako solid-guessing v případech, kdy malé ostrovy spadnou celé do testované oblasti.

Kapitola 3

Mandelbrotova množina

Mandelbrotova množina, a s ní související Juliovy množiny, byla objevena při zkoumání některých zdánlivě jednoduchých zpětnovazebních dynamických systémů. Analýza dynamických systémů se zabývá otázkou budoucího stavu systému, pokud iterativně provádíme výpočet s určitou funkcí (případně skupinou funkcí). Studiem těchto sytémů se na počátku 20. století zabývali matematikové Pierre Fatou a Gaston Julia, kteří jako první definovali Mandelbrotovu množinu. Za použití počítačové grafiky Mandelbrotovu množinu poprvé zobrazil v roce 1979 matematik Benoit B. Mandelbrot, po kterém získala tato množina svůj název.

3.1 Mandelbrotova množina

Mandelbrotova množina je vytvořena pomocí jednoduchého dynmického systému se zpětnou vazbou, který je založen na iteraci funkce komplexní paraboly:

$$Z_{n+1} = Z_n^2 + C \tag{3.1}$$

kde proměnné Z a C leží v komplexní rovině. Během výpočtu se hodnota Z postupně mění, zatímco hodnota C zůstává konstantní. Iterační proces začíná s hodnotou Z_0, takže systém postupně generuje posloupnost hodnot Z_i, které se nazývají orbit. U takto popsaného systému je důležité určit, zda pro danou hodnotu Z_0 a konstantu C posloupnost Z_i konverguje nebo diverguje. Nejznámější Mandelbrotova množina je určena počáteční hodnotou $Z_0 = 0$ a pro každý počítaný bod se pouze mění konstanta C. Výše popsaným iterativním výpočtem vzniknou orbity nuly, které lze rozdělit do dvou kategorií:

- pro hodnotu C je orbit konečný, tzn. všechny hodnoty Z_i jsou konečné,
- pro hodnotu C je orbit nekonečný, tzn. po určité době rostou hodnoty Z_i nad všechny meze.

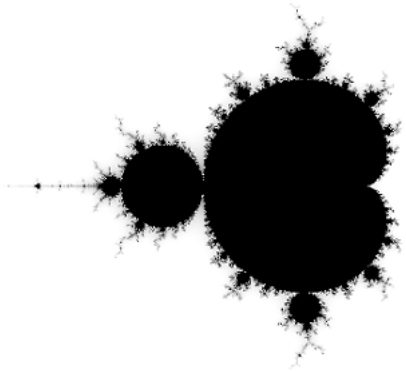
Matematicky lze Mandelbrotovu množinu definovat jako množinu všech komplexních čísel C, které produkují konečný orbit:

$$M = \{C | P_C^i(0) \neq \infty \forall n \rightarrow \infty; Z, C \in \mathbf{C}\} \tag{3.2}$$

kde $P_C^i(0)$ znamená hodnotu Z_i pro danou konstatnu C a hodnotu $Z_0 = 0$. Lze dokázat, že všechny body Mandelbrotovy množiny leží v komplexní rovině uvnitř kružnice o poloměru 2. Tohoto faktu lze využít při testování, zda posloupnost diverguje a počítaný bod tedy neleží v Mandelbrotově množině. Test na konvergenci posloupnosti Z_i nelze provést přesně, místo toho se použije metoda hádání. Zvolí se dostatečně velký počet iterací, přičemž v každém kroku se testuje, zda absolutní hodnota nepřekročí 2. Pokud se tak během výpočtu stane, bod určitě neleží uvnitř Mandelbrotovy množiny. Pokud proběhně výpočet všech iterací, bod je prohlášen za část Mandelbrotovy množiny. Nepřesnost spočívá v tom, že například posloupnost, která konverguje během 256 iterací, může už při 257 iteraci divergovat.

Algoritmus pro výpočet Mandelbrotovy množiny lze popsat následujícím způsobem:

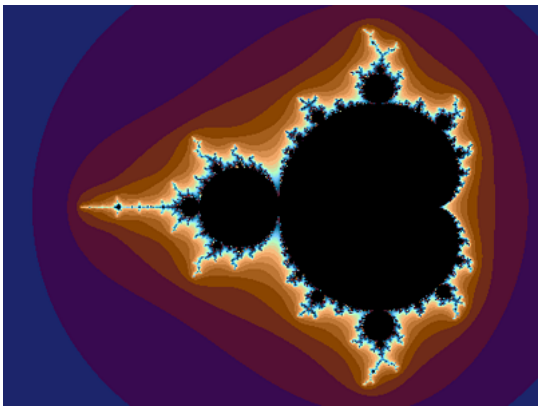
1. nastav $Z_0 = 0$
2. nastav $iter = 0$
3. pokud $iter < maxIter$ prováděj smyčku
   - 3.1. spočítej $Z = Z^2 + C$
   - 3.2. pokud $|Z| > 2$ konec
   - 3.3. $iter = iter + 1$
4. konec smyčky
5. bod leží uvnitř Mandelbrotovy množiny, konec



Obrázek 3.1: Mandelbrotova množina

Mandelbrotovu množinu lze vykreslit velmi jednoduše pomocí výše uvedeného algoritmu. Stačí provést převod pozice pixelu na obrazovce na hodnotu komplexní konstanty C. Pro každý pixel se tedy vypočte, zda jemu příslušející konstanta C způsobí divergenci či konvergenci posloupnosti Z_i. Mandelbrotova množina se v komplexní rovině většinou vykresluje ohraničena obdélníkem o souřadnicích -2+1.5i (levý horní roh) a 2-1.5i (pravý

dolní roh). Samozřejmě je možné vykreslit Mandelbrotovu množinu ve více barvách než jen černá a bíla. Metod, jakým způsobem určit barvu pixelu, existuje nepřeberné množství a jejich popis je nad rámec této práce. Například lze barvu zvolit na základě velikosti koncové hodnoty orbitu, na základě podílu reálné a imaginární složky orbitu, na základě úhlu koncové hodnoty orbitu, na počtu iterací nebo podle libovolně definované funkce a dosáhnout tak vizuálně velice zajímavého výsledku.



Obrázek 3.2: Obarvená Mandelbrotova množina

Algoritmus pro výpočet Mandelbrotovy množiny nabízí několik možností, jak dosáhnout jeho urychlení. Lze použít metodu detekce periodické posloupnosti, solid-guessing, tesseral nebo převést výpočet do celočíselné aritmetiky.

3.2 Juliovy množiny

S Mandelbrotovou množinou velmi úzce souvisí Juliovy množiny, a to tak, že Mandlbrotova množina tvoři mapu všech Juliových množin. Juliovy množiny jsou tvořeny stejnou funkcí komplexní paraboly jako Mandelbrotova množina, jediný rozdíl je v hodnotách parametrů Z_0 a C. V případě Juliových množin hodnota Z_0 reprezentuje pozici počítaného bodu v komplexní rovině (tzn. mění se pro každý bod, zatímco u Mandelbrotovy množiny byla hodnota konstantní pro všechny body) a hodnota C zůstává konstantní pro všechny body (u Mandelbrotovy množiny se měnila pro každý počítaný bod). Juliovy množiny jsou definovány jako množiny všech komplexních čísel Z_0, pro které posloupnost Z_i nediverguje (tj. posloupnost buď konverguje nebo osciluje):

$$J = \{Z | P_C^n(0) \neq \infty \forall n \rightarrow \infty; Z, C \in \mathbf{C}\} \tag{3.3}$$

kde $P_C^i(0)$ znamená hodnotu Z_i pro danou konstatnu C a hodnotu Z_0. V závislosti na hodnotě C lze Juliovy množiny rozdělit na tři typy:

1. Pro určité hodnoty C tvoří body ležící v Juliově množině spojitou oblast. To znamená, že každé dva body je možné navzájem spojit určitou křivkou tak, že

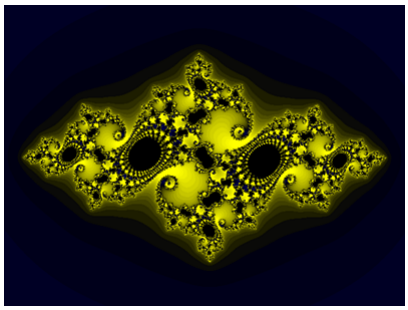
celá křivka leží uvnitř Juliovy množiny. Tato oblast je vždy jedna, neexistují tedy například navzájem izolované "ostrovy".

2. Pro určité hodnoty C jsou jednotlivé body tvořící Juliovu množinu zcela izolovány, tj. v jejich okolí neexistuje další bod, který by ležel v Juliově množině. Taková "rozložená" Juliova množina se nazývá Fatoův prach (*Fatou dust*) nebo také Cantorův prach (*Cantor dust*).

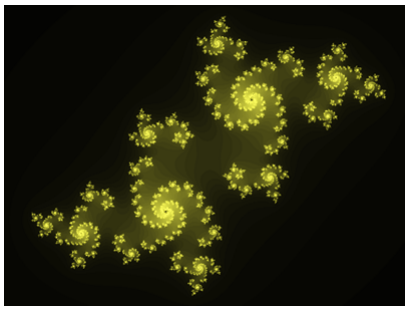
3. Třetí možnost leží na hranici obou předchozích. Body ležící uvnitř Juliovy množiny sice nejsou vzájemně izolovány, ale současně netvoří žádnou plochu (body na úsečce také nejsou izolovány, ale úsečka má nulovou plochu).

Algoritmus pro výpočet Juliovy množiny lze popsat následujícím způsobem:

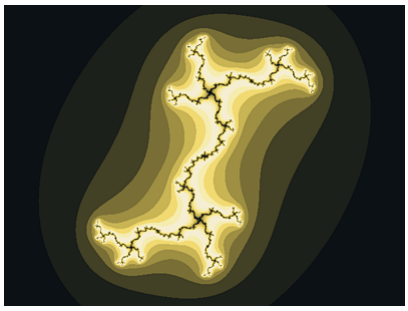
1. nastav Z_0="pozice bodu v komplexní rovině"
2. nastav $iter = 0$
3. pokud $iter < maxIter$ prováděj smyčku
   - 3.1. spočítej $Z = Z^2 + C$
   - 3.2. pokud $|Z| > 2$ bod neleží v Juliově množině, konec
   - 3.3. $iter = iter + 1$
4. konec smyčky
5. bod leží uvnitř Juliovy množiny, konec



Obrázek 3.3: Juliova množina typu 1



Obrázek 3.4: Juliova množina typu 2

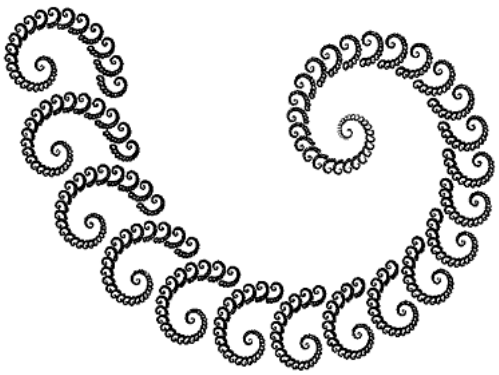


Obrázek 3.5: Juliova množina typu 3

Kapitola 4

Systémy iterovaných funkcí

Název této skupiny fraktálů je odvozen od anglického názvu Iterated Function System (IFS). Fraktály tohoto typu jsou obecně nepravidelné, protože do algoritmu generování je vnášena náhoda. Díky tomuto IFS fraktály umožňují výborný popis přírodních objektů. Metoda generování využívá soběpodobnosti fraktálů, kdy mohou vzniknout menší kopie původního objektu pomocí afinních transformací, lineárních i nelineárních. Tyto transformace jsou například libovolnou kombinací otočení, posunu, zkosení, zmenšení (lineární transformace) a ohybu, zborcení či zkroucení (nelineární transformace). IFS fraktály nachází velké uplatnění například v tvorbě procedurálních modelů těles nebo fraktální kompresi dat.



Obrázek 4.1: Příklad IFS fraktálu

4.1 Matematický popis

Pro zjednodušení uvažujme pouze afinní lineární transformace. Tyto transformace musí splňovat ještě jednu podmínku, a to, že musí být kontrakcemi. Kontrahující transformace znamená, že po aplikaci transformace na dva různé body v rovině či prostoru, bude nová vzdálenost bodů menší než vzdálenost bodů původních. Na základě předchozích

vlastností lze afinní transformace v Euklidově prostoru definovat jako:

$$d(\varphi_i(X) - \varphi_i(Y)) < s \times d(X - Y) \tag{4.1}$$

kde:

- d(X) je Euklidova metrika
- X a Y jsou dva libovolné body v rovině $\mathbf{E^2}$ nebo prostoru $\mathbf{E^3}$
- $\varphi_i(X)$ a $\varphi_i(Y)$ jsou body po aplikaci transformace φ_i

Podle velikosti koeficientu s lze určit typ transformace:

- $s < 1$ - φ_i je transformací zmenšení (kontrakce)
- $s = 1$ - φ_i je transformací symetrie
- $s > 1$ - φ_i je transformací zvětšení (expanze)

Transformace jsou popsány maticí U (v rovině $\mathbf{E^2}$ maticí 2x2, v prostoru $\mathbf{E^3}$ maticí 3x3) a vektorem posunutí V (v rovině $\mathbf{E^2}$ má dva prvky, v prostoru $\mathbf{E^3}$ prvky tři). Afinní transformaci lze tedy zapsat ve tvaru:

$$w(X) = UX + V \tag{4.2}$$

Koeficienty matice U se uplatňují při aplikaci transformace rotace, zkosení a změny měřítka, koeficienty vektoru V při posunu.

K množině zobrazení $\phi = \{\varphi_1, \varphi_2, ...\varphi_n\}$ přísluší množina pravděpodobností $\Pi = \{\pi_1, \pi_2, ...\pi_n\}$. Součet všech pravděpodobností je roven jedné.

Na základě uvedených vztahů lze systém iterovaných funkcí definovat jako uspořádanou dvojici:

$$IFS = (\phi, \Pi) = (\{\varphi_1, \varphi_2, ...\varphi_n\}, \{\pi_1, \pi_2, ...\pi_n\}) \tag{4.3}$$

4.2 Metody generování IFS

Z matematického hlediska jsou IFS fraktály tvořeny nekonečně mnoha body. Z toho důvodu by pro vygenerování a vykreslení celého fraktálního objektu bylo zapotřebí provést nekonečně mnoho iterací. Pro praktickou aplikaci je nutné, aby byl čas generování IFS fraktálu konečný. Proto se generuje pouze určitý počet bodů (řádově statisíce) a výsledkem je pouze přibližný tvar fraktálního obrazce.

Algoritmus náhodné procházky

Nejjednodušším algoritmem pro generování IFS fraktálů je tzv. algoritmus náhodné procházky (*Random Walk Algorithm - RWA*). Na libovolně zvolený bod P_{iter} ležící v rovině $\mathbf{R^2}$ nebo prostoru $\mathbf{R^3}$ jsou náhodně aplikovány jednotlivé transformace φ_i. Výsledkem je nová pozice bodu P_{iter+1}. IFS fraktál je poté zobrazen jako množina všech vygenerovaných bodů P_i pro $i \in (1..maxIter)$. Pozici počátečního bodu P_0 je možné volit libovolně z toho důvodu, že všechny transformace jsou kontrahující a tím pádem zajišťují směřování systému ke svému atraktoru.

Algoritmus lze popsat následovně:

1. náhodně vyber počáteční polohu bodu P_0
2. nastav čítač iterací $iter = 0$
3. pokud iter<maxIter prováděj smyčku
   - 3.1. vygeneruj náhodné číslo $n \in < 0..1)$
   - 3.2. podle čísla n a pravděpodobností transformací π zvol transformaci φ_i
   - 3.3. aplikuj transformaci φ_i na bod P_{iter} a vypočti nový bod P_{iter+1}
   - 3.4. bod P_{iter+1} je vstupním bodem další iterace
   - 3.5. $iter = iter + 1$
4. vykresli pixel na pozici bodu P_{iter}
5. konec

Výhodou tohoto algoritmu je poměrně velká jednoduchost a malá paměťová náročnost. Nevýhodou je nutnost generovat velké množství bodů, z nichž některé se mohou generovat vícekrát, zatímco jiné vůbec. Při malém počtu iterací se výsledný obrazec nemusí vykreslit.

Deterministický algoritmus

Na odlišném principu než algoritmus RWA pracuje deterministický algoritmus (*Deterministic Iteration Algorithm - DIA*). Tento algoritmus nepracuje s jedním bodem jako algoritmus RWA, ale s celou množinou bodů. Současně nemusí počítat s pravděpodobností jednotlivých transformací, protože v každém kroku se na množinu bodů aplikují všechny transformace. Na počátku generování se zvolí několik náhodných bodů v rovině nebo prostoru (poloha těchto bodů nemá stejně jako u RWA vliv na výsledný tvar fraktálu), na které se aplikují všechny transformace. Nově vzniklá množina bodů tvoří vstup další iterace. Transformace se aplikují deterministicky, odtud tedy vznikl název tohoto algoritmu.

Algoritmus lze popsat následovně:

1. náhodně vyber množinu počátečních bodů $P_0 = \{P_0^1, P_0^2, ...P_0^n\}$
2. nastav čítač iterací $iter = 0$
3. pokud $iter < maxIter$ prováděj smyčku
   3.1. nastav $index = 0$
   3.2. pokud $index <$ "$poettransformac$" prováděj smyčku
      3.1.1. pro každý bod $P \in P_{iter}$ proveď: $P_{iter+1} = P_{iter} \bigcup \varphi_{index}(P)$
4. vykresli všechny body uložené v množině P_{iter+1}
5. konec

Počáteční bod stačí vybrat jeden, například $P_0 = \{P_0^1\}$, $P_0^1 = [0, 0]$, protože s každou iterací se množina zvětšuje.

Výhodou algoritmu je měnší počet iterací, které jsou potřeba provést k vykreslení IFS fraktálu. Velkou nevýhodou je ale velká spotřeba paměti, protože s každou iterací se množina bodů exponenciálně zvětšuje.

4.3 Plynulý morfing mezi dvojicí IFS fraktálů

Pomocí velmi jednoduchého způsobu lze docílit plynulého morfingu z jednoho IFS fraktálu do druhého. Celý princip spočívá v postupné změně koeficientů transformační matice U jednoho systému do druhého. Nejjednodušší způsob je v provedení váženého součtu odpovídajících koeficientů $a_{i,j}$ obou systémů:

$$a_{i,j} = (1 - ratio) \times a_{i,j}^1 + ratio \times a_{i,j}^2 \tag{4.4}$$

kde parametr ratio nabývá hodnot z intervalu $< 0, 1 >$. Tento výpočet je nutné použít pro všechny transformace zadaných IFS systémů. V případě, že jeden systém bude mít jiný počet transformací než druhý, je potřeba chybějící transformace uměle přidat. Aby přidáním transformací nedošlo ke změně systému, musí být přidáváné transformace identitou, aby po aplikaci na nějaký bod vrátily původní souřadnice tohoto bodu.

Kapitola 5

Stochastické fraktály

Důležitou skupinou fraktálů jsou takzvané stochastické (náhodné, nepravidelné) fraktály. Narozdíl od ostatních typů fraktálů, které jsou většinou soběpodobné, stochastické fraktály vnáší při svém generování do algoritmu náhodu. Díky tomuto jsou pouze soběpříbuzné, nikoliv soběpodobné. Díky této nepravidelnosti a náhodnosti umožňují vůbec nejlepší popis přírodních objektů.

Stochastické fraktály lze generovat buď simulací Brownova pohybu, simulací difůze, metodou přesouvání prostředního bodu nebo spektrální syntézou. Zatímco první dvě metody lze úspěšně využít při generování travin a keřů, zbývající metody se používají pro generování obrázků plasmy, modelů terénů či mraků.

5.1 Metody generování travin a keřů

Simulace difůze

Pomocí této metody je možné vytvářet tvarově složité přírodní útvary, například modely trávy, keřů a stromů. Útvary vygenerované touto metodou se vyznačují částečnou soběpodobností, velkou tvarovou složitostí a fraktální strukturou.

Simulace difůze je založena na náhodném pohybu částic v omezeném či neomezeném prostoru. Protože výsledkem simulace nemá být zobrazení trajektorie částic hmoty, jak je tomu v reálném světě, ale plošný či prostorový model přírodního objektu, je nutné tuto metodu modifikovat. Modifikace spočívá v postupném vytváření pevných bodů v rovině či prostoru, na kterých postupně "vyrůstají" výčnělky tvořené z dalších bodů, které jsou, stejně jako už dříve vytvořené body, nepohyblivé. Pevné body lze ke vznikajícímu fraktálnímu obrazci přidávat dvěma způsoby:

- Částice se vytvoří na náhodném místě a dále se simuluje její pohyb v prostoru. Tento pohyb plně vychází z principu Brownova pohybu. Po dotyku se stávající fraktální strukturou je tato částice připojena a generuje se částice nová. Výsledkem jsou věrohodnější modely, ovšem za cenu větší časové náročnosti výpočtu.
- Částice se postupně vytváří na náhodných místech, ale nepohybují se. Pokud se vygenerovaná částice dotkne stávající struktury, je k ní připojena. V opačném

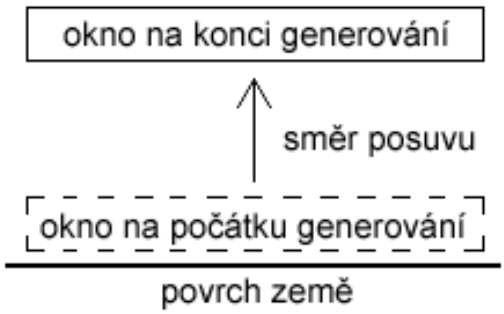
případě zanikne. Postupným vytvářením částic na náhodných místech se aproximuje trajektorie částice, která se pohybuje Brownovým pohybem. Není zaručeno, že se zjistí veškeré dotyky trajektorie částice se vznikajícím fraktálem, to je však kompenzováno větším množstvím generovaných částic.

Tvar vznikajícího fraktálního obrazce je velmi výrazně určen pozicí jedné či více částic, které představují základ pro další růst útvaru. Kvůli této vlastnosti se počáteční částice nazývá semínko (*seed*).

Algoritmus simulace difůze lze popsat následovně:

1. vytvoř množinu bodů (semínek) $S^{'} = (S_1, S_2, ...Sn)$
2. urči oblast Ω, ve které se může fraktální objekt generovat
3. urči typ okolí, ve kterém se hledají body příslušející již vytvořenému objektu (čtyřokolí, osmiokolí, ...)
4. nastav čítač iterací $iter = 0$
5. dokud $iter < maxIter$ prováděj smyčku
   - 5.1. vygeneruj náhodný bod P, jehož souřadnice musí ležet v oblasti Ω
   - 5.2. pokud se v okolí bodu P nachází už dříve vygenerovaný bod, přidej bod P ke vznikajícímu obrazci

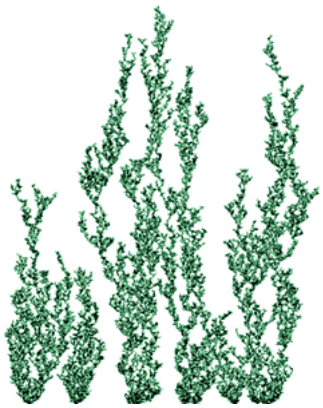
Tento základní algoritmus simulace difůze je možné různými způsoby modifikovat a nepřímo tím měnit i morfologii, fraktální dimenzi, popř. hustotu vytvářených objektů. Kromě modifikace výběru počáteční množiny semínek lze použít tzv. posuvné okno, ve kterém se částice mohou generovat. Posuvná oblast je na počátku generování umístěna na povrchu země, tj. v místech, ze kterých má model traviny či keře vyrůstat.



Obrázek 5.1: Princip posuvného okna

Generování fraktálního obrazce potom probíhá tím způsobem, že se po každém posuvu okna o předem daný krok $\Delta\Omega$, vygeneruje zadané množství bodů. Pokud se tyto body dotýkají již vytvořeného obrazce, jsou k němu připojeny. Vzhledem k tomu, že je množství náhodných bodů generovaných v oblasti Ω konstantní, bude postupně klesat či naopak růst množství bodů, které se při každém posuvu okna připojí k vytvářenému objektu.

Změnou velikosti oblasti Ω a počtu generovaných náhodných bodů lze docílit různých výsledků.



Obrázek 5.2: Simulace difůze, model mořských řas

Simulace difůze na základě Brownova pohybu

Modely vytvořené podle předcházejícího algoritmu nejsou zcela korektní, protože vzhledem k zavedení pevných částic se trajektorie aproximuje pouze hrubě a z toho důvodu nejsou nalezeny všechny dotyky částic se stávajícím fraktálním objektem. Na druhou stranu jsou v některých oblastech objektu částice velmi nahuštěny, takže obrázky spíše připomínají mořské řasy či chaluhy.

K zajištění tvorby korektních modelů je proto nutné zvolit přesnější simulaci difúze, která bude přímo založena na Brownově pohybu jednotlivých částeček hmoty. Princip práce této metody spočívá v postupném vytváření částic, které se pohybují po trajektorii dané Brownovým pohybem do té doby, než se dotknou stávajícího fraktálního objektu.

V prvním kroku algoritmu se opět inicializuje množina semínek a vygeneruje se náhodně umístěný (v určitém ohraničeném prostoru) počáteční bod. Následně se tento bod začne pohybovat, a to tak, že v každém kroku může změnit své souřadnice maximálně o jednotku. Nová souřadnice bodu se tedy nachází v osmiokolí souřadnice bodu z předchozího kroku. Pokud se bod na nové souřadnici dotkne stávajícího fraktálního obrazce, je k němu připojen. Pokračuje se vygenerováním souřadnic nového bodu a celý postup se opakuje. Generování lze například ukončit po vygenerování určitého počtu částic nebo ve chvíli, kdy se částice dostane na okraj oblasti, ve které se objekt generuje.

Obrázek 5.3: Simulace difůze na základě Brownova pohybu

5.2 Plasma a metoda přesouvání prostředního bodu

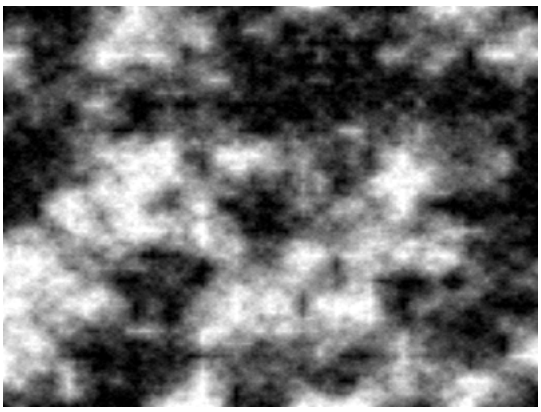
Jednou z velmi používaných metod pro generování stochastických fraktálů je metoda přesouvání prostředního bodu (*Midpoint Displacement Method - MDM*), kterou lze snadno použít pro generování objektů v rovině i prostoru. Často se tato metoda používá při generování přírodní krajiny nebo mraků.

Nejjednodušší aplikací metody přesouvání prostředního bodu je postup rekurzivního dělení horizontální úsečky tak, aby vznikl fraktální útvar:

1. dvojice bodů P_1^0 a P_2^0 určují úsečku u^0 o délce L^0 (horní index značí počet rekurzivního volání metody)
2. nastav $iter = 0$
3. pokud $iter < maxIter$ opakuj smyčku
   - 3.1. pro danou úsečku u^{iter} najdi její prostřední bod $P_S^{iter} = \frac{P_1^{iter}+P_2^{iter}}{2}$
   - 3.2. bod P_S^{iter} je vertikálně posunut o náhodnou hodnotu δ^{iter}, vznikne tak nový bod $P_{S'}^{iter}$
   - 3.3. úsečka u^{iter} je rozdělena na dvě poloviny: $u_1^{iter} = P_1^{iter} P_{S'}^{iter}$, $u_2^{iter} = P_{S'}^{iter} P_2^{iter}$
   - 3.4. koeficient δ se s každou iterací sníží tak, aby se limitně blížil k nule
   - 3.5. nastav $iter = iter + 1$ a postup od bodu 3. aplikuj na nově vzniklé úsečky u_1^{iter} a u_2^{iter}

Metodu půlení úsečky je možné snadno rozšířit na plošný útvar, čtverec. V nejjednodušším případě leží krajní body čtverce v rovině x-y a při rekurzivním dělení jsou posouvány v kladném či záporném směru osy z. Jednoduchou metodou, jak čtverec postupně rozdělovat je, že se v každém kroku rekurze vypočte výška bodů ležících v polovině hran čtverce. Ta je poté posunuta o náhodnou hodnotu. Čtverec je tím rozdělen na čtyři menší části, na které se rekurzivně aplikuje stejný postup.

Tuto metodu lze využít pro generování prostorových modelů terénu nebo obrázků, kterým se říká plasma (které lze například použít jako základ pro vytvoření výškové mapy terénu).



Obrázek 5.4: Šedotónní obrázek plasmy

Kapitola 6

Dosažené výsledky a demonstrační aplikace

6.1 Shrnutí výsledků

Porovnání metod RWA a DIA pro generování IFS fraktálů

Algoritmus náhodné procházky a deterministický algoritmus lze porovnávat ze dvou hledisek, a to podle časové a paměťové náročnosti.

Z hlediska paměťových nároků je méně náročný algoritmus RWA, protože pracuje přímo s prostorem (bitmapou) vygenerovaných bodů a jediná potřebná paměť je na uložení polohy iterovaného bodu. Algoritmus DIA je paměťově podstatně více náročnější, zejména kvůli nutnosti udržovat v paměti množinu už vygenerovaných bodů.

Po experimentování s různými parametry lze říci, že algoritmus DIA pracuje obecně rychleji jak algoritmus RWA. To je hlavně způsobeno tím, že k vygenerování obrazce pomocí algoritmu DIA stačí provést podstatně méně iterací než u algoritmu RWA. Oba algoritmy jsem vyzkoušel implementovat i pomocí celočíselné aritmetiky. Algoritmus RWA se implementací v celočíselné aritmetice zrychlí v průměru o 35%, je nutné ovšem počítat s dostatečným počtem iterací (řádově tisíce a více). Implementace algoritmu DIA pomocí celočíselné aritmetiky nepřinesla v podstatě žádné zrychlení výpočtu, spíše došlo ke zkoršení kvality vygenerovaného obrázku. To je způsobeno hlavně ztrátou přesnosti při převodu reálných čísel do celočíselného formátu, což se projeví tím, že se při výpočtu některé body "přeskočí" a ve výsledném obrázku potom chybí určité pixely.

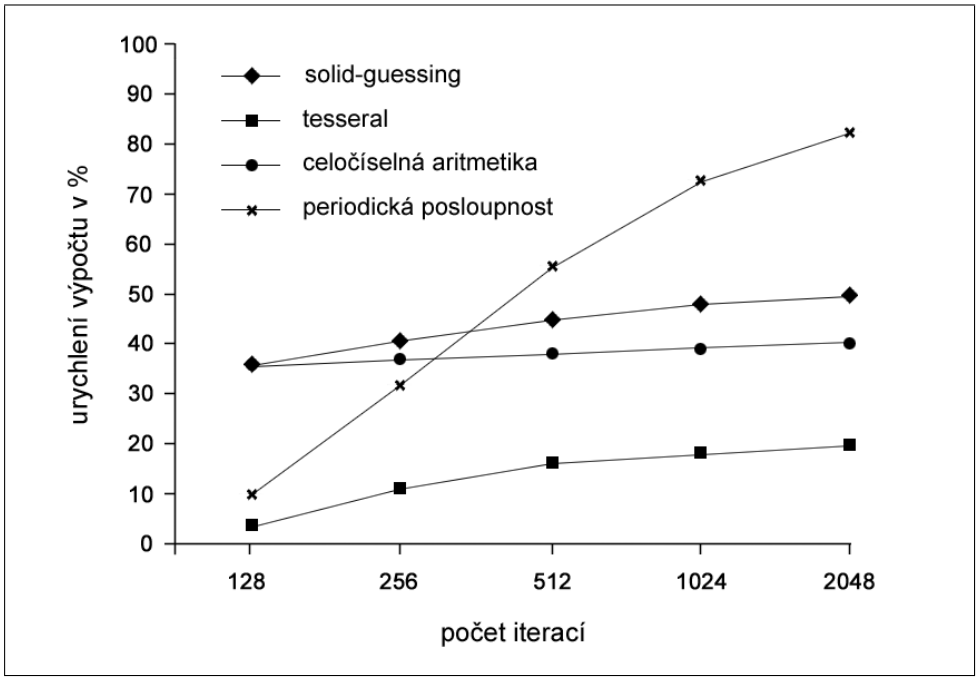
Zhodnocení metod pro generování stochastických fraktálů

Prostudované algoritmy (simulace difůze, simulace difůze na základě Brownova pohybu a metoda přesouvání prostředního bodu) pro generování stochastických fraktálů není možné vzájemně srovnávat, protože každý produkuje jiný výsledek. Po podrobném prostudování jsem zjistil, že z povahy těchto algoritmů není možné provést jakoukoliv formu jejich optimalizace a urychlit tak výpočet.

Porovnání metod pro urychlení výpočtu Mandelbrotovy množiny

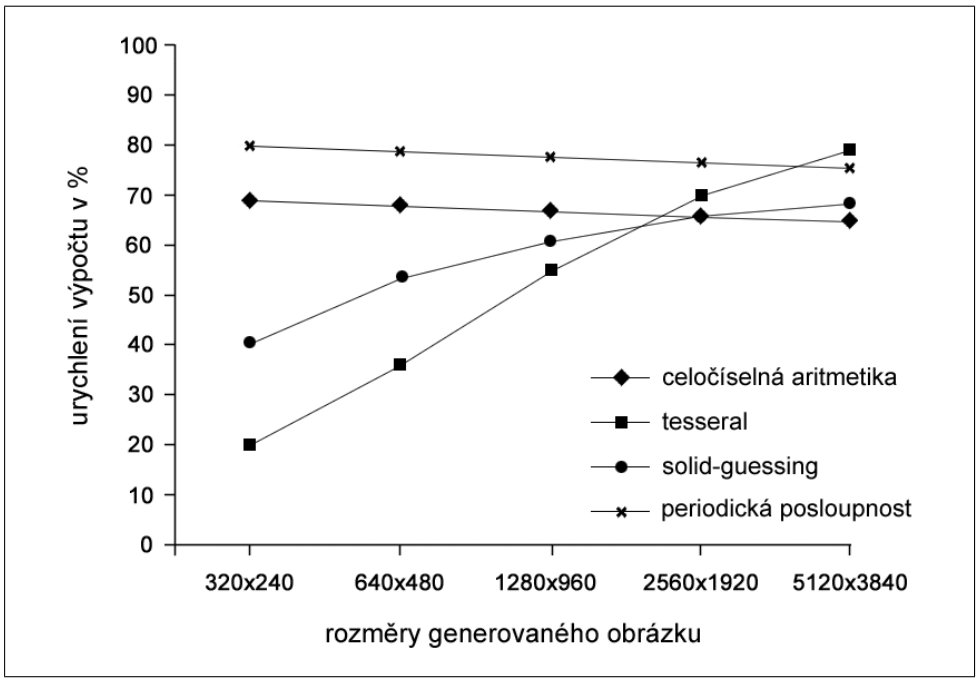
Algoritmus pro výpočet Mandelbrotovy množiny lze urychlit pomocí metody tesseral, solid-guessing, detekce periodické posloupnosti a transformací výpočtu do celočíselné aritmetiky. Uvedené metody se ukázaly být pro urychlení výpočtu velmi přínosné, ovšem většinou za cenu zhoršení kvality vygenerovaného obrázku. Tyto metody lze porovnávat z mnoha pohledů, navíc lze u některých metod modifikovat určité parametry, proto zde uvádím pouze dvě nejvýznamější srovnání na základě experimentálně získaných výsledků. Obě srovnání jsou vztažena vůči nemodifikovanému algoritmu pro výpočet Mandelbrotovy množiny.

Urychlení výpočtu v závisloti na počtu iterací ukazuje následující graf. Porovnával jsem metodu detekce periodické posloupnosti, tesseral, solid-guessing a výpočet v celočíselné aritmetice. Uvedené hodnoty jsou získané pro obrázek o rozměrech 640x480 pixelů.



Obrázek 6.1: Závislost urychlení výpočtu na počtu iterací

Urychlení výpočtu v závislosti na rozměrech generovaného obrázku ukazuje následující graf. Porovnával jsem metodu detekce periodické posloupnosti, tesseral, solid-guessing a výpočet v celočíselné aritmetice, Uvedené hodnoty jsou získané pro 512 iterací/pixel.



Obrázek 6.2: Závislost urychlení výpočtu na rozměrech generovaného obrázku

Na základě získaných výsledků uvedených v grafech lze velice snadno určit, jakou metodu je vhodné kdy použít. Všechny metody lze navíc navzájem vhodně kombinovat a dosáhnout tak velmi rychlého výpočtu Mandelbrotovy množiny.

6.2 Demostrační aplikace

Součástí práce bylo vyzkoušet si nastudované algoritmy implementovat. Cílem nebylo vytvořit propracovanou aplikaci pro generování fraktálů, z toho důvodu je celý demonstrační program navržen dosti jednoduše. Jako implementační jazyk jsem zvolil C/C++ za použití knihovny SDL (Simple Directmedia Layer) pro práci s grafikou a překladač Microsoft Visual C++. Celý kód je psán bez použití nepřenositelných konstrukcí, takže v případě potřeby by měl jít velice snadno použít i na jiné platformě. Knihovna SDL je použita pouze pro vytvoření jednoduchého GUI (Graphical User Interface - grafické uživatelské rozhraní) a zobrazení vygenerovaných obrázků, vlastní algoritmy pro generování fraktálů na ní nejsou závislé.

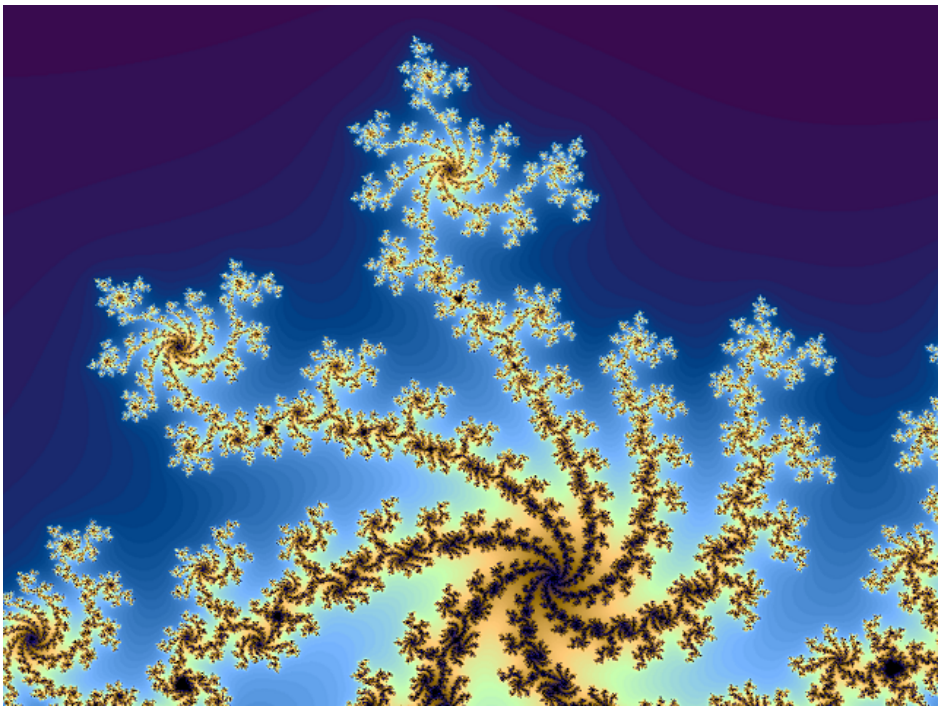
Pro každý z výše popsaných typů fraktálů jsem implementoval vlastní třídu, která zapouzdřuje popsané algoritmy. Každá třída poskytuje jednu nebo dvě metody pro na-

stavení potřebných parametrů a jednu metodu pro vygenerování fraktálu. Tato metoda poté podle svého parametru volá privátní funkce, které provádějí konkrétní algoritmus.

Aplikace umožňuje vygenerovat:

- Mandelbrotovu množinu pomocí metody detekce periodické posloupnosti, solid-guessing, tesseral a výpočtem v celočíselné aritmetice,
- Juliovu množinu podle interaktivně zvolené pozice v Mandelbrotově množině,
- IFS fraktál pomocí algoritmů RWA, DIA a jejich verzí v celočíselné aritmetice,
- plynulou animaci morfingu jednoho IFS fraktálu do druhého,
- model traviny pomocí metody simulace difůze a metody simulace difůze na základě Brownova pohybu,
- obrázek plasmy pomocí metody přesouvání prostředního bodu.

Ukázkou výstupu demonstrační aplikace jsou všechny obrázky uvedené v této práci.



Obrázek 6.3: Výřez Mandelbrotovy množiny

Kapitola 7

Závěr

Během studia fraktálových algoritmů jsem získal dostatek znalostí, abych v případě potřeby dokázal nějaký fraktálový algoritmus vhodně použít. Prakticky jsem se také seznámil s několika možnostmi, jak prakticky dosáhnout urychlení výpočtu daného algoritmu.

Další směřování práce bych zaměřil na využití technologie SSE, na implementaci uvedených algoritmů, jejich optimalizaci na úrovni instrukcí a vzájemné srovnání, případně i na porovnání efektivnosti kódu napsaného v asembleru a nějakém vyšším programovacím jzyce.

Literatura

[1] G. Schober J. H. Ewing. *The area of the Mandelbrot Set.* Numer. Math., 1992.

[2] Benoit B. Mandelbrot. *The Fractal Geometry of Nature.* W. H. Freeman, New York, 1982.

[3] Benoit B. Mandelbrot. *Fraktály - tvar, náhoda a dimenze.* Mladá fronta, 2003.

[4] Prof. Dr. P. H. Richter Prof. Dr. H. O. Peitgen. *The Beauty of Fractals.* Springer-Verlag Berlin Heidelberg, 1986.

[5] Fraktály v počítačové grafice. http://www.root.cz/clanky/fraktaly-v-pocitacove-grafice-i/.

[6] Fixed point arithmetic. http://www.root.cz/clanky/fixed-point-arithmetic/.

[7] Generating fractals with sse/sse2. http://www.codeproject.com/cpp/fractalssse.asp.