VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

FRAKTÁLY V POČÍTAČOVÉ GRAFICE

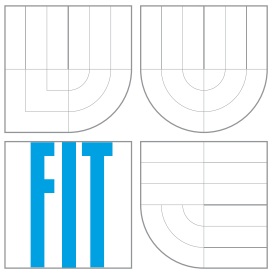
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

JAN HEINÍK

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

FRAKTÁLY V POČÍTAČOVÉ GRAFICE

FRACTALS IN COMPUTER GRAPHICS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

JAN HEINÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ADAM HEROUT, Ph.D.

BRNO 2007

Abstrakt

Diplomová práce se zabývá historií fraktální geometrie a popisuje vývoj nauky o fraktálech. Po počátečním seznámení se základními pojmy jsou popsány jednotlivé druhy fraktálů a jejich typické příklady. Dále jsou uvedeny oblasti, ve kterých je možno se s fraktály setkat mimo obor počítačové grafiky. Práce seznamuje s praktickým využitím fraktální geometrie. V textu jsou uvedeny v současné době známé programy a softwarové balíky vhodné pro zobrazování fraktálů a jsou popsány jejich možnosti. Praktickou část diplomové práce tvoří slajdy, demonstrační program a plakát. Elektronické slajdy představují osnovou využitelnou pro přednášky o problematice fraktální geometrie. Program slouží k demonstraci vybraných druhů fraktálů. Plakát je grafickým shrnutím výsledků práce.

Klíčová slova

Fraktál, Juliova množina, Mandelbrotova množina, Newtonův fraktál, L-systém, Systém iterovaných funkcí, Stochastické fraktály, Hausdorffova dimenze, atraktor, soběpodobnost, Sierpinského trojúhelník, africké fraktály, Cantorova množina, Kochova vločka, Hilbertova křivka, Fractal Flame

Abstract

This Master's thesis deals with history of Fractal geometry and describes the fractal science development. In the begining there are essential Fractal science terms explained. Then description of fractal types and typical or most known examples of them are mentioned. Fractal knowledge application besides computer graphics area is discussed. Thesis informs about fractal geometry practical usage. Few present software packages or more programs which can be used for making fractal pictures are described in this work. Some of theirs capabilities are described. Thesis' practical part consists of slides, demonstrational program and poster. Electronical slides represents brief scheme usable for fractal geometry realm lectures. Program generates selected fractal types. Thesis results are projected on poster.

Keywords

Fractal, Julia set, Mandelbrot set, Newton fractal, L-system, Iterated Function Set, Stochastic fractal, Hausdorff dimension, attractor, self-similarity, Sierpinski triangle, African fractals, Cantor set, Koch flake, Hilbert curve, Fractal Flame

Citace

Jan Heiník: Fraktály v počítačové grafice, diplomová práce, Brno, FIT VUT v Brně, 2007

Fraktály v počítačové grafice

Prohlášení

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

.....
Jan Heiník
21. května 2007

Poděkování

Rád bych poděkoval Ing. Adamu Heroutovi, Ph.D. za vstřícný přístup, odborné vedení a pomoc při návrhu struktury diplomové práce.

Zadání diplomové práce/5134/2006/xheini00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2006/2007

Zadání diplomové práce

Řešitel: **Heiník Jan**
Obor: Výpočetní technika a informatika
Téma: **Fraktály v počítačové grafice**
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte a popište historii a pozadí nauky o fraktálech.
2. Prostudujte a popište typické i méně známé algoritmy zobrazování fraktálních obrazců v počítačové grafice.
3. Prostudujte a popište použití fraktálů v praxi.
4. Prostudujte a popište existující a dostupné softwarové balíky pro výpočet fraktálů.
5. Navrhněte a implementujte program zobrazující vybrané fraktály.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; prezentujte projekt plakátkem.

Literatura:

- Barnsley Michael: *"Fractals Everywhere"*, Academic Press Inc., 1988, ISBN 0-12-079062-9
- Barnsley Michael, Devaney R. L., Mandelbrot Benoit B., Peitgenn Heinz-Otto, Saupe Dietmar, Voss Richard: *"The Science of Fractal Images"*, Springer-Verlag, New York, 1988
- Mandelbrot Benoit B.: *"The Fractal Geometry of Nature"*, W. H. Freeman, New York; San Francisco, 1982, ISBN 0-7167-1186-9
- Peitgen Heinz-Otto, Richter Peter: *"The Beauty of Fractals"*, Springer-Verlag, New York, 1986, ISBN 0-387-15851-0

Při obhajobě semestrální části diplomového projektu je požadováno:

- Dílčí řešení bodů 1 - 4.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Herout Adam, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 22. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Pavel Zemčík
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Jan Heiník**
Id studenta: 11787
Bytem: Lukavec čp. 59, 742 45 Fulnek
Narozen: 05. 03. 1981, Bílovec
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
diplomová práce

Název VŠKP: Fraktály v počítačové grafice
Vedoucí/školitel VŠKP: Herout Adam, Ing., Ph.D.
Ústav: Ústav počítačové grafiky a multimédií
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	.....
Nabyvatel	Autor

Obsah

Kapitola 1

Úvod

Poprvé je možné se s pojmem "fraktál" setkat v roce 1975, kdy jej ve své práci Fraktály: Tvar, náhoda a dimenze [11] (francouzsky *Les objets fractals, forme, hasard et dimension*) uvedl Benoit Mandelbrot. Mnoho takovýchto útvarů, jako např. Kochova vločka či Sierpinského koberec, bylo objeveno a popsáno již dříve, mnohé byly dokonce vytvořeny již před staletími - Africké fraktály. Pojem vychází z latinského slova "fractus" (rozbitý). Tyto složité struktury, vyhovující určitým matematickým požadavkům, můžeme nalézt také v přírodě. Představují je sněhové vločky, kapradí, stromy, blesky, mraky, atd. Často je jejich poznávacím znakem soběpodobnost, což znamená, že daný útvar se jeví jako stejný v jakémkoliv měřítku a stále se opakuje.

Práce je rozdělena na dvě hlavní části - teoretickou a praktickou. V teoretické části práce jsou vysvětleny základní pojmy jako je fraktál, soběpodobnost aj. a jsou přiblíženy souvislosti vývoje fraktální podstaty. Dále jsou uvedeny příklady fraktálních útvarů, jejich rozčlenění a druhy algoritmů, které lze ke generování obrazců použít. V podkapitolách jsou uvedeny vždy popisy jednotlivých typů - Cantorova množina, IFS fraktály, Mandelbrotova množina atd. Fraktály se vyskytují v mnoha oblastech techniky i každodenních jevech. Některým zajímavostem je věnována větší pozornost. Například v kapitole č. 4 jsou popsány fraktály, které se vyskytují ve světě kolem nás, ať už v přírodě, nebo v některémm z odvětví techniky.

Přechod mezi teoretickou a praktickou částí práce tvoří 5. kapitola, kde jsou uvedeny algoritmy pro generování fraktálních útvarů. V textu jsou uvedeny příklady, kdy tvarově podobné fraktály lze vytvořit odlišným způsobem. Pro zajímavost jsou u některých algoritmů uvedeny možnosti modifikací. Dále jsou popsány některé softwarové balíky pro generování fraktálů.

Praktickou část reprezentuje kapitola 6. Cílem diplomové práce bylo zpracovat informace ve formě elektronických slajdů využitelných jako výukový materiál předmětu Počítačová grafika na FIT VUT v Brně. Pro ilustraci jednotlivých skupin fraktálů je součástí práce také demonstrační program.

Kapitola 2

Základní pojmy

2.1 Fraktál

Fraktál je členitý objekt, jehož jednotlivé části vykazují tvarovou podobnost s celkem [16]. Těmito objekty se zabývá vědní disciplína zvaná fraktální geometrie, která se začala rozvíjet od 60. let minulého století díky Benoitu B. Mandelbrotovi. Mandelbrot v roce 1977 jako první použil pojem fraktál a uvedl jeho definici. Ačkoliv bylo mnoho takovýchto objektů známo a popsáno již dříve (Kochova vločka, Sierpinského trojúhelník, atd.), byly většinou považovány za abnormality, a jak uvádí [12, 3], matematici je řadili do tzv. "Galerie monster". Tyto útvary se zdály nové, ničemu v přírodě neodpovídající a odporovaly geometrické intuici. Mnohé z nich byly "soběpodobné". Právě tato vlastnost je umožňovala popsat nejlépe, proto je Mandelbrot vyčlenil pro své vědecké zkoumání a nazval je právě "fraktály". Ukázal, že fraktály nejsou vyjímkami, ale že v podstatě vše v přírodě má fraktální podstatu. Díky matematickému popisu fraktálu je Mandelbrotovi připisováno prvenství v této oblasti a podle něho je pojmenována také množina, která vychází z jednoduché komplexní definice (bude uvedena později). Vykreslenou Mandelbrotovu množinu můžeme vidět na obrázku 2.1.

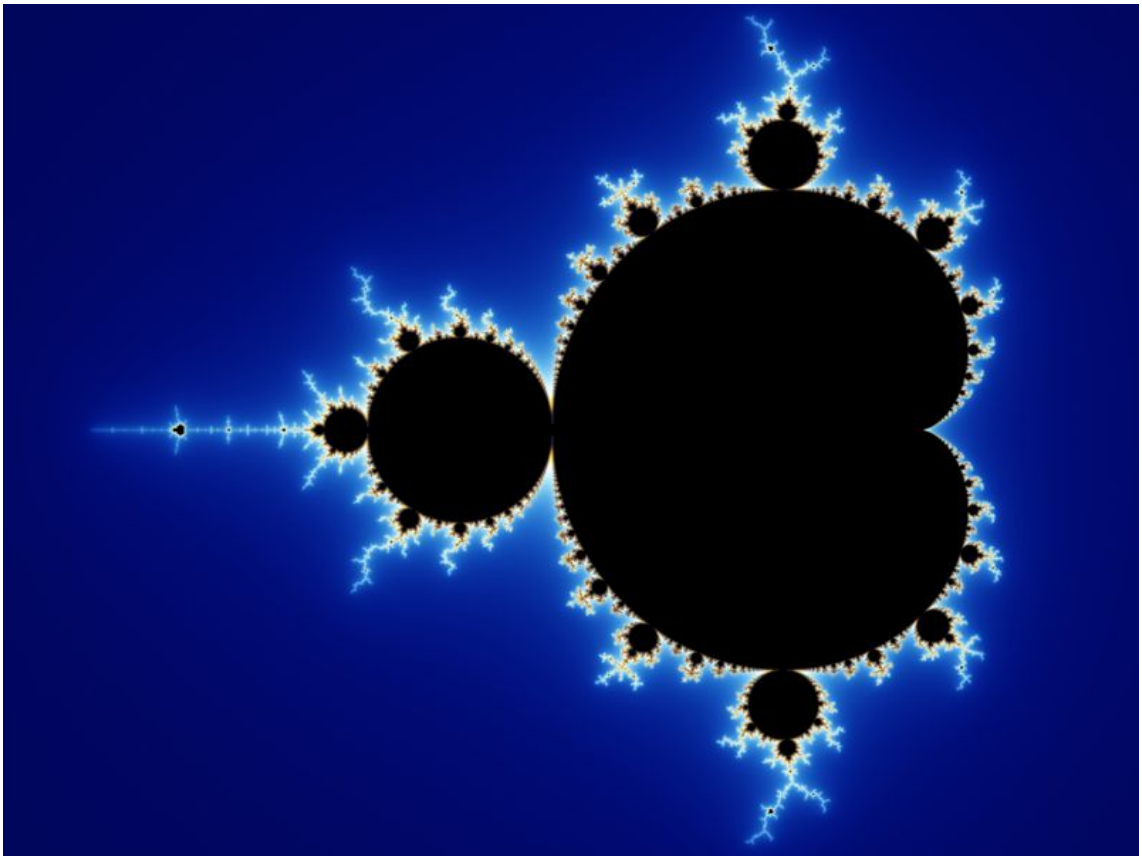
Fraktál tedy můžeme, z pohledu počítačové grafiky, popsat jako nekonečně členitý útvar, jehož část často vypadá podobně při jakémkoliv přiblížení či rotaci jako celý obrazec [10]. Další zajímavou vlastností může být např. nekonečně velký obvod, či nekonečně malý obsah.

Matematická definice fraktálu dosud nebyla uvedena. Nejvíce se jí blíží Mandelbrotova specifikace: **Fraktál je možina, jejíž hodnota Hausdorffovy-Besicovichovy dimenze přesahuje hodnotu dimenze topologické**, uvedená v jeho knihách [3, 2, 12, 11]. Topologická dimenze (D_T) určuje rozměr tělesa, jak jej známe z běžné geometrie. D_T bodu je rovna nule, přímka má $D_T = 1$, rovina $D_T = 2$ a krychle $D_T = 3$. Hausdorffova dimenze obvykle nebývá celé číslo. Více podrobností bude uvedeno v kapitole 2.2.2.

2.1.1 Historie fraktálů

Fraktální geometrie je mladou, avšak dnes již relativně rozsáhlou vědní disciplínou, která zasahuje do mnoha oborů. Největší rozvoj nastal od šedesátých let 20. století, kdy se začaly využívat první počítače [25]. Jak již bylo uvedeno, nejznámějším fraktálem je Mandelbrotova množina (obrázek 2.1), dynamický fraktál ležící v komplexní rovině. Mandelbrot nebyl první, kdo tento fraktál vykreslil, jako první ho však publikoval.

Příkladem objevení fraktální podstaty objektu dávno před Mandelbrotem jsou tzv. Africké fraktály [6]. Vesnice, paláce nebo šperky, které lze nalézt na území afrických států,



Obrázek 2.1: Mandelbrotova množina

splňují podmínky pro zařazení mezi fraktály svým tvarem či strukturou. V kapitole 4 jsou uvedeny půdorysné fotografie vesnice, jejíž obydlí jsou uspořádána do fraktálního obrazce.

S fraktály je možné se setkat také v oboru meteorologie a klimatologie. Vědci při zkoumání grafů předpovědí počasí zjistili, že vypočtené hodnoty se značně měnily i s jen minimální změnou vstupních údajů. Z toho vyplývá, že počasí se chová jako chaotický systém a nelze dosáhnout spolehlivé předpovědi chování pro delší časové úseky [9]. Tento problém popsal Edward Lorenz v roce 1963 při zkoumání tří dynamických rovnic, jak je uvedeno v kapitole 2.4.1.

Chaos byl v té době také předmětem zkoumání na mnoha vědeckých pracovištích. V laboratořích IBM se snažili vědci vyřešit problém s náhodným výskytem šumu na komunikačních linkách mezi počítači. Při snaze o jeho odstranění bylo vždy opraveno vedení, ovšem bez úspěchu. Šum se stále objevoval a mizel. V té době u IBM pracoval také Mandelbrot. Na vytištěných záznamech datových přenosů a chyb na vedení vypozoroval značnou podobnost s Cantorovou množinou (obrázek 3.5). Problém odstranili opakováním přenosu zašuměných dat [11, 25].

Konkrétní příklady jsou uvedeny v kapitole 4 na straně 34.

2.2 Dimenze

Dimenze je hlavní pojem při studiu fraktálů. Běžná geometrická tělesa lze popsat konečným počtem parametrů (např. délka, plocha, počet rozměrů). Většina běžných útvarů má hodnotu dimenze celé číslo, nazývá se topologická dimenze, které vyjadřuje kolik souřadnic potřebujeme k jednoznačnému určení polohy bodu v tomto útvaru [25, 24]. Takovéto útvary značíme jako geometricky hladké tělesa. Pro fraktální geometrii však není topologická dimenze dostatečná, proto se přidává dimenze fraktální - Hausdorffova (podle německého profesora Felixe Haussdorfa). Tato dimenze nabývá neceločíselných hodnot a udává míru členitosti útvaru, viz kapitola 2.2.2.

2.2.1 Topologická dimenze

Topologická dimenze (D_T) je známá z klasické geometrie. Jak již bylo řečeno dříve, udává počet parametrů, nutný pro popis polohy bodu v tělese [25, 24, 12]. D_T bodu je rovna nule, pro popis jeho polohy v něm samém není potřeba žádný údaj. Bod můžeme popsat vztahem $P = X$.

Úsečka má $D_T = 1$, protože potřebujeme jeden údaj pro určení polohy bodu na úsečce. Totéž platí i pro další křivky, kde poloha bodu na křivce je vyjádřena podobně jako pro např. parabolu :

$$x = x_0 + t^2 \tag{2.1}$$

kde t je parametrem, který jednoznačně definuje polohu bodu na křivce. Délka křivek může být nekonečná, avšak jejich plocha je nulová.

D_T roviny je rovna dvěma, pro určení polohy bodu na ploše potřebujeme dva parametry, které jej jednoznačně určují. Vyjádřením roviny ohraničené kružnící budiž:

$$x = x_0 + r * cos\ \alpha \tag{2.2}$$
$$y = y_0 + r * sin\ \alpha \tag{2.3}$$

kde r je parametrem určujícím poloměr kružnice, její střed se nachází v bodě $[x_0, y_0]$ a α je proměnný parametr $\alpha \in \langle 0, 2\pi)$.

Podobně pak má např. krychle $D_T = 3$.

Pro základní geometrické útvary známe vztahy, ze kterých můžeme vypočítat jejich vlastnosti, jako délku, plochu či objem. Tyto hodnoty jsou nezávislé na jednotkách, ve kterých jsou udány parametry těles. Detailněji viz [3, 24].

2.2.2 Hausdorffova dimenze

Fraktální dimenze či Hausdorffova dimenze, bude označována D, udává míru členitosti útvaru. Pro fraktální objekty je hodnota této dimenze větší než hodnota D_T.

Při měření vlastností Euklidovských útvarů dostáváme celočíselné hodnoty nezávisle na zvoleném měřítku, ať už zjišťujeme jejich délku, povrch, objem, dimenzi. Pro fraktální obrazce však není celočíselné vyjádření dimenze dostatečné.

Při měření délky pobřeží ostrova Korsika v roce 1961 zjistil L. Richardson zvláštnost - naměřená hodnota byla závislá na délce používaného měřidla. Čím více se zmenšovalo měřidlo, tím větší hodnota vyšla, tudíž při délce limitně se blížící nule, by délka rostla

do nekonečna [12, 25, 9]. Z toho vyplývá, že ostrov o konečné ploše má nekonečnou délku pobřeží. Ilustraci měření je možno vidět na obrázku 2.2.

Obrázek 2.2: Měření obvodu ostrova

Richardson při zkoumání hranic států a pobřeží ostrovů zjistil, že odměřená délka $L(G)$ je závislá na měřítku G. Po několika měřeních vyslovil domněnku, že $L(G)$ může být aproximováno funkcí:

$$L(G) \quad = \quad MG^{1-D} \tag{2.4}$$

Až Mandelbrot ukázal, že délka pobřeží a ostatní zeměpisné hranice jsou závislé na konstantě D, jež vyjadřuje Hausdorffovu dimenzi [22].

Stejný problém, jako s obvodem ostrova, by nastal při měření délky řeky, pokud by se změřila délka levého a pravého břehu a pak se zprůměrovala. Při přesném měření by se délka řek blížila nekonečnu. Právě proto, že vzdálenost od ústí k prameni je konečná, je nutno řeky měřit po daných krocích, či přímo v toku řeky.

Vzorec pro výpočet Hausdorffovy dimenze lze získat následujícím způsobem:

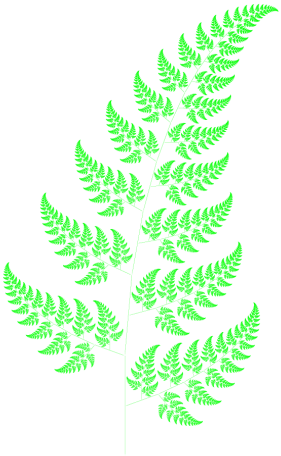
- mějme úsečku, kterou rozdělíme na N dílků, délka jednoho dílku bude $s = \frac{1}{N}$,
- pro Hausdorffovu dimenzi platí obecně, že $N * s^D = 1$,
- postupnými úpravami předchozího výrazu dostaneme vzorec 2.5, ze kterého po dosazení daných parametrů, dostaneme Hausdorffovu dimenzi objektu.

$$D = \frac{\log N}{\log \frac{1}{s}} \tag{2.5}$$

Hausdorffova dimenze je tedy číslo, které udává jakou rychlostí roste délka, obsah či objem do nekonečna. Liší-li se D a D_T málo, bude útvar málo členitý, kdežto bude-li $D_T > D$, bude útvar velmi členitý. Podrobněji viz [3, 24].

2.3 Soběpodobnost

Soběpodobnost je jednou z hlavních vlastností fraktálu. Dá se říci, že soběpodobnost znamená opakování sebe sama např. se změnou měřítka, posunutím, rotací nebo zkosením. V matematice se setkáváme s označením *invariance vůči změně měřítka*. Tato vlastnost také napomáhá identifikaci fraktálů v přírodě. Například kapradinu můžeme považovat za složeninu nekonečného počtu větviček či lístků, které po zvětšení vypadají opět jako kapradina, viz obrázek 2.3.



Obrázek 2.3: Fraktální kapradina

Definice soběpodobné množiny zní takto [25]: Soběpodobná množina A n-dimenzionálního Euklidovského prostoru E_n je taková množina, pro níž existuje konečně mnoho kontrahujících zobrazení $\phi_1, ..., \phi_n$ (zobrazení z E^n do E^n), které zmenší vzdálenost mezi body ležícími v E^n takových, že A vznikne jako:

$$A = \bigcup_{i=0}^{n} \Phi_i(A) \tag{2.6}$$

Pro E^2 a E^3 se může u zobrazení ϕ_i jednat o tzv. *lineární transformace*, kdy se mění i měřítko. Těchto transformací se pak využívá ke generování IFS fraktálů, které jsou popsány v kapitole 3.3.

Množina definovaná tímto způsobem má několik velmi zajímavých vlastností:

- vzniká opakováním sebe sama při určité transformaci (změna měřítka, rotace, posunutí nebo zkosení). Používají se i transformace nelineární, avšak je důležité, aby vždy šlo o transformace, které zkracují vzdálenost mezi dvěma různými body.
- je invariantní vůči změně měřítka. Při libovolném zvětšení, či zmenšení vypadají podobně. Vlastnost vyplývá z definice soběpodobné množiny.
- vzniká sama ze sebe, respektive vzniká opakováním téhož motivu.

Princip opakování podobných tvarů ve zmenšené podobě je vidět prakticky u jakékoliv složité struktury, která je generována i velmi jednoduchými pravidly. Větvení stromů či cév a žil v tělech živočichů, nebo hromadění baktérií a řas v koloniích, se dá matematicky uspokojivě popsat pouze fraktální geometrií. Při pokusech o popsání přírodních útvarů nástroji Euklidovské geometrie byly konstrukce příliš komplikované nebo neodpovídaly realitě. Fraktály však slouží i k modelování a pochopení složitých dějů, které se odehrávají v čase, tudíž se jedná o jevy dynamické.

Při používání pojmu soběpodobnosti je třeba si uvědomit, že se jedná o podobnost objektu při změně měřítka. Existuje mnoho geometrických útvarů, které jsou podobné (nebo shodné) při aplikaci jiné transformace. Například čtverec je invariantní vůči středovému zrcadlení, kruh je invariantní vůči stranovému zrcadlení a přímka je invariantní vůči posunu ve směru jejího vektoru. Tato invariance, pokud není doprovázena invariancí vůči změně měřítka, v žádném případě nedefinuje fraktál.

2.4 Atraktor

S pojmem "atraktor" se setkáváme nejčastěji při popisu dynamických systémů a IFS systémů. Atraktor dynamického systému je množina stavů, do kterých systém směřuje. Je to množina hodnot, kterých může nabývat stavový vektor dynamického systému po dostatečně dlouhém časovém úseku od inicializace systému. Z geometrického hlediska může být atraktorem bod, křivka nebo jakkoli rozmanitá množina fraktální podstaty.

Atraktory se dají rozdělit do několika tříd a může jimi být [25]:

- množina pevných bodů,
- množina periodických bodů,
- množina kvaziperiodickych bodů,
- atraktor je chaotický,
- atraktor je podivný.

Je-li atraktorem dynamického systému množina pevných bodů, jedná se o nejjednodušší případ - systém se v nekonečném čase ustálil v nějakém stabilním stavu, který lze dopředu vypočítat.

Je-li atraktorem množina periodických (resp. kvaziperiodických) bodů, jde také o elementární případ. Systém se po určité době ustálil tak, že osciluje mezi několika stavy.

Je-li atraktor chaotický, znamená to, že výsledný stav systému nelze nijak dopředu předpovědět. To může být způsobeno také tím, že je systém velmi citlivý na počáteční podmínky. Chaotičnost v tomto případě neznamená náhodnost, protože se stále pohybujeme v rámci deterministických systémů.

Podivný atraktor (anglicky strange attractor) je z hlediska fraktální geometrie případem nejzajímavějším. Tento typ atraktoru může vzniknout, je-li systém popsán minimálně třemi souvisejícími diferenciálními rovnicemi. Takový systém může mít velmi komplikovaný atraktor, který sice bude vykazovat vlastnosti pravidelného, ale současně i chaotického atraktoru. Termín podivný atraktor není přesně matematicky definován (definice sice existují, nezahrnují však všechny typy podivných atraktorů), ale považujeme za něj takový atraktor, který vykazuje stejné vlastnosti, jaké mají fraktály. Všechny chaotické atraktory jsou současně podivnými atraktory, opačně to však neplatí. Příklad podivného atraktoru máme na obrázku 2.4, nebo viz [12, 25].

2.4.1 Lorenzův atraktor

Lorenzův atraktor byl prvním zkoumaným a popsaným dynamickým systémem. Edward Lorenz jej objevil při zkoumání vlastností vodního kola, kdy do nádob umístěných po obvodu přitéká a odtéká voda. Proces se jevil jako jednoduchý, avšak jedná se o chaotický systém. Projevila se u něj vysoká citlivost na počáteční podmínky, tzv. "motýlí efekt" (malá změna počátečních hodnot způsobí velký rozdíl ve výsledku). Naplněním nádoby se kolo roztočilo a Lorenz zkoumal možnosti pohybu kola, které mohou nastat v případě, že se nádoba nestihne naplnit celá, nevyteče celá atd.

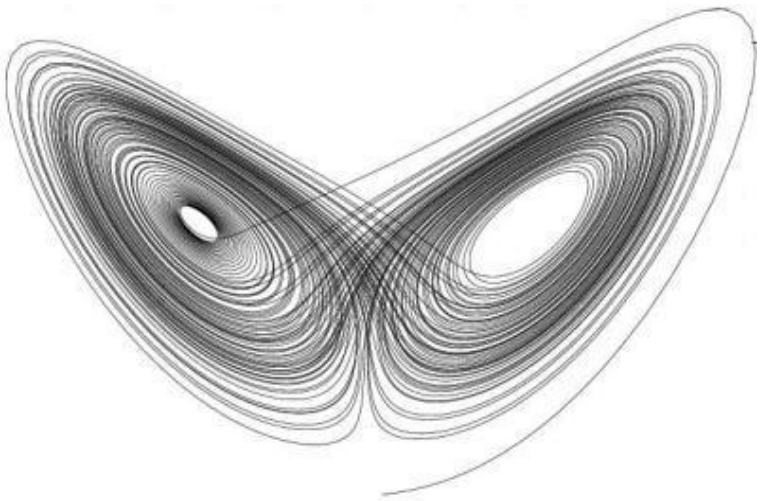
Pomocí tří rovnic [22]:

$$\frac{dx}{dy} = \sigma(y - x) \tag{2.7}$$

$$\frac{dy}{dt} = x(\rho - z) - y \tag{2.8}$$

$$\frac{dz}{dt} = xy - \beta y \tag{2.9}$$

vyjádřil Lorenz chování systému (σ se nazývá Prandtlovo číslo a ρ je Rayleightovo číslo). Když pak na počítači nechal zobrazit graf, systém se díky zvoleným hodnotám nechoval periodicky, ale chaoticky - nedalo se předpovědět v jakém stavu se bude systém nacházet v následujícím okamžiku. Výsledek je na obrázku 2.4. Dráhy se na simulaci překrývají z důvodu použití konečného počtu desetinných míst pro zobrazení čísel. Pokud by se použilo přesného modelu, tato situace by nikdy nenastala a dráhy by se křížily.



Obrázek 2.4: Lorenzův atraktor - graf

S podivnými atraktory a dynamickými systémy souvisí úloha pocházející z klasické fyziky - problém tří těles [25]. Ve vesmíru máme tři tělesa a zadané počáteční podmínky systému (poloha, hmotnost těles, počáteční rychlost a směr pohybu). Úkolem je analyticky vyjádřit poholu těles v libovolném okamžiku.

Byla dokázána neřešitelnost tohoto problému, přestože problém dvou těles je jednoduchý. Důvodem je pohyb těles po částečně chaotických drahách, kdy jsou přitahována k určitým bodům, kterými trajektorie procházejí. Tyto body byly nazvány atraktory kvůli své vlastnosti (přitažlivost, anglicky attraction).

Kapitola 3

Typy fraktálů

Jedněmi z prvních matematiků, kteří nalezli objekty vyhovující definici fraktálu, avšak neviděli mezi nimi ještě souvislost, byli Georg Cantor, Helge von Koch, Waclaw Sierpinski či Gaston Julia. I díky jejich práci mohl Mandelbrot později definovat pojem fraktál. V následujících kapitolách budou některé z prvních fraktálů uvedeny.

Postupem času byly fraktální útvary na základě společných charakteristik rozřazeny do skupin [25]. Takto členěné jednotlivé typy fraktálů jsou vhodné pro řešení určitých úloh v jednom nebo více oborech. V počítačové grafice je důležité, že algoritmy pro vytváření fraktálů jsou podobné pro jednotlivé skupiny, přestože se tvar vygenerovaných obrazců liší.

Fraktály dělíme na následující typy:

- Dynamické systémy s fraktální strukturou,
- L-systémy,
- Systémy iterovaných funkcí IFS,
- Stochastické fraktály (nepravidelné fraktály).

3.1 Dynamické systémy

Dynamické systémy tvoří tu skupinu (resp. kategorii) fraktálů, která má v technické praxi nejširší uplatnění. Dynamický systém je matematický model, jehož stav je závislý na nějaké proměnné (např. času). Dynamický systém vychází z počátečních podmínek a je jimi v čase determinován. Existují dynamické systémy, které se po určitém čase neustálí v pevném stavu, ale ani nedivergují. Tento případ, který připomíná iracionální čísla, má většinou fraktální dynamiku a označuje se termínem deterministický chaos. Více viz [3, 12, 24].

Dynamický systém je popsán pomocí dynamických podmínek, které popisují jeho změnu systému v čase. Dynamické podmínky jsou většinou zadány soustavou diferenciálních rovnic, které popisují změnu stavového vektoru v čase. Změna stavu dynamického systému se děje provedením těchto diferenciálních rovnic a nahrazením starého stavového vektoru novým.

Z hlediska počítačové grafiky je nejzajímavější vhodným způsobem vizualizovat nějakou vlastnost dynamického systému. Nejčastěji se u bodů v komplexní rovině sleduje únik od atraktoru v závislosti na počtu iterací. Dojde-li k úniku bodu, vybarví se tento bod úměrně počtu iterací. Z těchto systémů jsou v počítačové grafice asi nejvíce známé Juliovy množiny a Mandelbrotova množina.

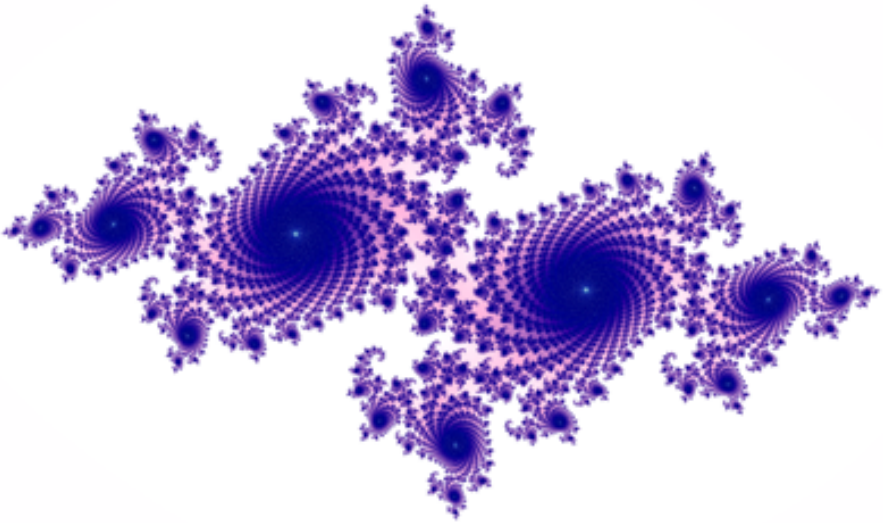
3.1.1 Juliovy množiny

Během druhé světové války dva francouzští matematikové Gaston Julia a Pierre Fatou objevili zvláštní útvary později nazýné Juliovy množiny. Některé se podobají keříkům, jiné mořským koníkům či králíkům.

Juliovy množiny jsou vytvářeny pomocí iterace funkce komplexní paraboly:

$$z_{n+1} = z_n^2 + c \tag{3.1}$$

kde proměnné z_n a c leží v komplexní rovině. Počáteční hodnota z_0 v případě Juliových množin reprezentuje pozici bodu v komplexní rovině. Komplexní hodnota c je zvolena libovolně a pro všechny počítané body v jednom obrazci zůstává konstantní. Příklad Juliovy množiny o souřadnicích (-0.726895347709114071439, 0.188887129043845954792) vidíme na obrázku 3.1.



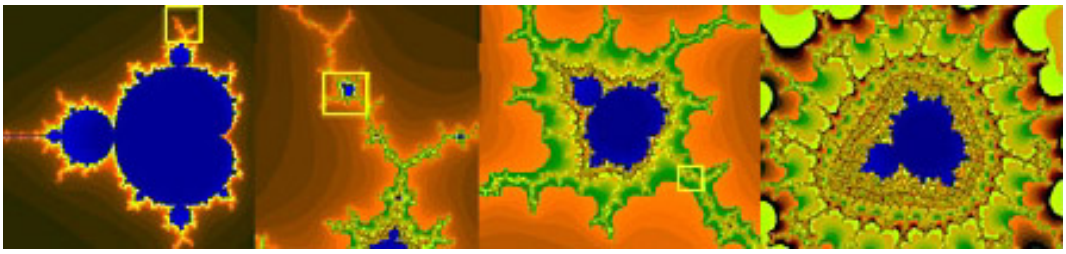
Obrázek 3.1: Juliova množina

Pro zvýraznění atritbutů se Juliovy množiny vykreslují barevně. Nejčastější způsob obarvení je ten, že barva bodu charakterizuje krok iterace, ve kterém bylo zjištěno, že bod nediverguje.

3.1.2 Mandelbrotova množina

Mandelbrotova množina (anglicky Mandelbrot Set nebo M-set) je nelineárním deterministickým fraktálem. Je považována za jeden z nejzajímavějších fraktálů (obrázek 2.1), který se používá v počítačové grafice, například ke generování textur nebo vytváření trojrozměrných modelů hor. Objevena byla při analýze dynamických systémů se zpětnou vazbou. Jako první zřejmě množinu vykreslili Robert Brooks a Peter Matelski v roce 1978. Obrazec byl černobílý a rozlišoval konvergenci a divergenci posloupnosti hodnot [12, 25]. S rozvojem počítačové grafiky došlo k rozšíření možností vybarvování na základě např. počtu iterací nutných k rozhodnutí divergence bodu, velikosti poslední vypočtené hodnoty $|z_n|$, podle hodnot reálné či imaginární složky atd.

Narozdíl od Juliových množin, kterých je více druhů, je Mandelbrotova množina jedna a je souvislá, což dokázali v roce 1982 Adrien Douady a John Hubbard. Právě Hubbard ji dal jméno, pod kterým je dnes všeobecně známa. Objekty stejného tvaru jako je samotná Mandelbrotova množina lze nalézt nejen po jejím obvodu, ale i "osamoceny" v okolí, jak lze vidět na obrázku 3.2. Vždy jsou spojeny s hlavní částí, z čehož vyplývá, že Mandelbrotova množina je souvislá.



Obrázek 3.2: Detaily v Mandelbrotově množině

Mandelbrotova množina je opět vytvářena pomocí iterace funkce komplexní paraboly:

$$z_{n+1} \quad = \quad z_n^2 + c \tag{3.2}$$

kde proměnné z_n a c leží v komplexní rovině. Mandelbrotovu množinu lze definovat jako množinu [16]

$$M \quad = \quad \{c \in C \mid c \to c^2 + c \to \cdots je \ \ omezená\} \tag{3.3}$$

což je původní Mandelbrotova definice z roku 1979. Lze ji ovšem také definovat jako množinu všech komplexních čísel c, kdy je Juliova množina J_c souvislá, tj.

$$M \quad = \quad \{c \in C \mid J_c \ \ je \ \ souvislá\} \tag{3.4}$$

Pokud je posloupnost iterací omezená, náleží bod C Mandelbrotově množině.

Zajímavostí je, že průsečík této množiny s reálnou osou tvoří proslulou Feigenbaumovu posloupnost bifurkací. Tato okolnost zavdala příčinu k písemné přestřelce mezi Mandelbrotem a Feigenbaumem, týkající se autorství některých základních idejí v teorii chaosu [16, 25].

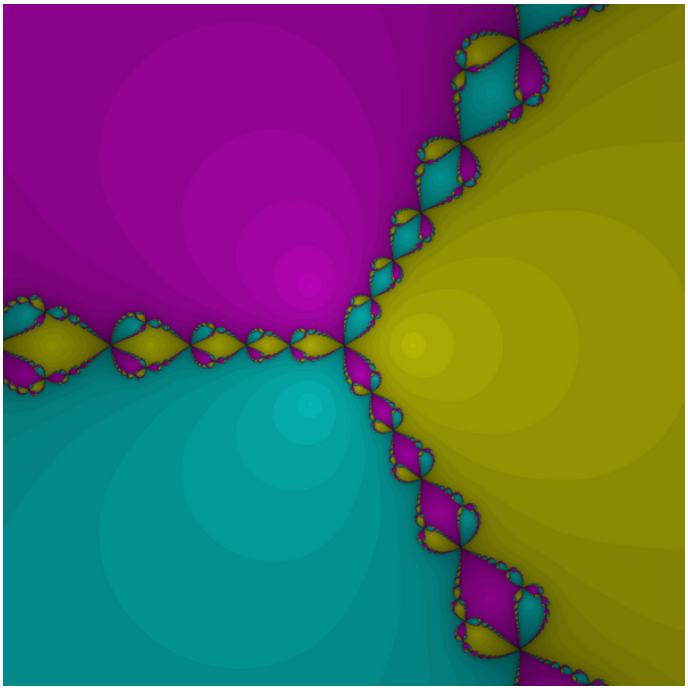
3.1.3 Newtonova fraktální množina

Newtonova fraktální množina byla objevena Johnem Hubbardem. Při přednášení Newtonovy iterační metody [9, 22], určené pro vyhledávání kořenů polynomů, zadal studentům úkol, aby zkusili pro rovnici 3.5 nalézt kořeny v komplexní rovině.

$$x^3 + 1 = 0 \tag{3.5}$$

Jeho představou bylo, že řešení rozdělí komplexní rovinu na několik oddělených ploch, ke kterým bude řešení konvergovat. Na rozhraní oblastí se však objevily zvláštní obrazce. Hubbard začal chování rovnice zkoumat a po barevném označení bodů dostal obrazec,

který můžeme vidět na obrázku 3.3. Na hranici tří velkých oblastí vídíme soběpodobný obrazec, který podle obarvení podoblastí určuje, ke kterému řešení konverguje. Zajímavé je, že výseče obsahují i barvy odpovídající nejvzdálenějšímu kořenu. Rovnice 3.5 je základní rovnicí pro vygenerování Newtonovy fraktální množiny a z obrázku je patrné, že mimo oblast konvergence je tato metoda citlivá na zvolené počáteční podmínky. Při změně stupně a tvaru polynomu dostáváme odlišný vzhled obrazce.



Obrázek 3.3: Newtonova fraktální množina

3.2 L-systémy

L-systémy, jež jsou v některé literatuře také označovány termínem Lindenmayerovy systémy, jsou skupinou fraktálů definovaných pomocí přepisovacích gramatik [24]. Název pochází z anglické zkratky LOGO-like turtle. LOGO je známý programovací jazyk určený pro výuku programování, jež je založený na Lispu. Pomocí tohoto jazyka se jednoduchými příkazy, ovládajícími virtuální želvu pohybující se po ploše, dají kreslit různé obrazce složené většinou z úseček.

Podstatou tvorby L-systémů je přepisování řetězců podle daných gramatik. Každému terminálnímu symbolu (znaku, jež se nepřepisuje) v řetězci je přiřazen jistý geometrický význam, který odpovídá příkazu pro želvu v jazyce LOGO. Mezi tyto terminální symboly patří například znaky uvedené v tabulce 3.1 [25].

Pomocí posledních dvou příkazů (push a pop) lze generovat obrazce, ve kterých probíhá větvení struktury. Zajímavé obrazce s fraktální strukturou se začnou tvořit, jestliže v jazyce LOGO použijeme rekurzi, což odpovídá iteraci v gramatice (na pravé straně přepisovacího pravidla gramatiky je nonterminální symbol shodný se symbolem na levé straně pravidla). Jednou z možností, jak klasické L-systémy rozšířit, je použití více želv a zavedení nových symbolů pro souběžné nebo samostatné řízení těchto želv.

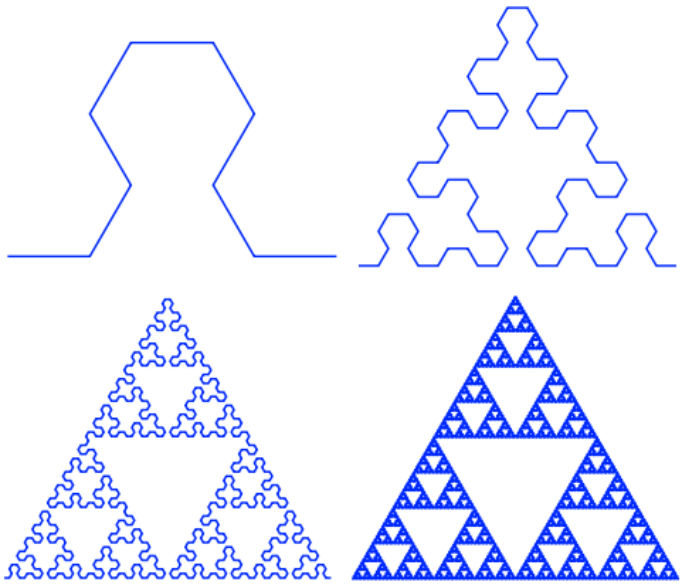
Symbol	Význam
F	forward - posun želvy dopředu
B	back - posun želvy dozadu
+	right - natočení želvy doprava
-	left - natočení želvy doleva
[	push - uložení pozice a orientace želvy na zásobník
]	pop - obnovení pozice a orientace želvy ze zásobníku

Tabulka 3.1: Terminální symboly

Za pomoci L-systémů lze generovat fraktály, které se podobají rostlinám, stromům a dalším přírodním útvarům. Složitější aplikace směřují k využití těchto fraktálů ke generování plošných i prostorových (tj. objemových) textur, využití nacházejí také v oblasti CAD systémů.

L-systémy se dá vytvořit také Sierpinského trojúhelník, obrázek 3.4, který generuje gramatika:

$$
\begin{aligned}
\textit{symboly}: &\quad A,\ B\ (\textit{kresli vpřed}) \\
\textit{konstanty}: &\quad +,\ -(\textit{rotace doprava},\ \textit{doleva o}\ \ \textit{úhel}) \\
\textit{start}: &\quad A \\
\textit{pravidla}: &\quad A \rightarrow B - A - B \qquad (3.6) \\
&\quad B \rightarrow A + B + A \qquad (3.7) \\
\textit{úhel}: &\quad 60^{\circ}
\end{aligned}
$$



Obrázek 3.4: Sierpinského trojúhelník

Mezi L-systémy patří i Fibonacciho čísla, jež jsou generovaná gramatikou se startovacím symbolem A a pravidly:

$$A \rightarrow AB \tag{3.8}$$
$$B \rightarrow B \tag{3.9}$$

což vygeneruje sekvenci:

$$\begin{aligned} n = 0 &: A \\ n = 1 &: B \\ n = 2 &: AB \\ n = 3 &: BAB \\ n = 4 &: ABBAB \\ n = 5 &: BABABBAB \\ &\vdots \end{aligned}$$

Pokud spočítáme délku všech řetězců, dostaneme Fibonacciho posloupnost čísel: 1 1 2 3 5 8 13 21 34 ....

3.2.1 Cantorova množina

Cantorova množina (někdy označovaná Cantorovo diskontinuum) je množina bodů z intervalu $\langle 0, 1\rangle$. Popsal ji německý matematik Georg Cantor při zkoumání množin bodů, ve spojitosti s konvergencí fourierových řad.

Zkonstruuje se tak, že se daný interval rozdělí na tři shodné itervaly, kdy je vypuštěn interval $\left(\frac{1}{3}, \frac{2}{3}\right)$. Takto jsou získány intervaly $\left\langle 0, \frac{1}{3}\right\rangle$ a $\left\langle \frac{2}{3}, 1\right\rangle$. V další iteraci se postupuje stejně - vyjme se vždy prostřední třetina intervalu. Intervaly, které v množině zůstaly jsou Cantorovou množinou. Sedm iterací lze vidět na obrázku 3.5.

Obrázek 3.5: Cantorova množina, 7 iterací

Do množiny náleží krajní body intervalů - $0, 1, \frac{1}{3}, \frac{2}{3}, \frac{1}{9}, \frac{2}{9}$ atd., kterých je spočetně mnoho. Cantorova množina je nespočetná [12, 16], existuje proto ještě nespočetně mnoho dalších bodů, které do množiny náleží.

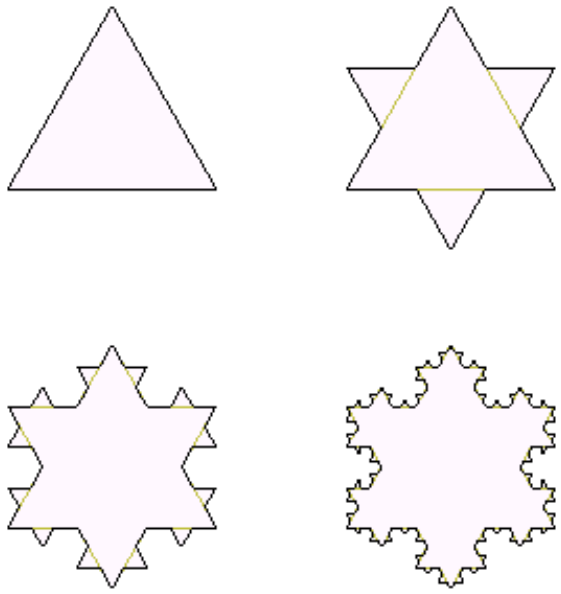
Gramatika pro vygenerování Cantorovy množiny v pravidlech želví grafiky (viz tabulka 3.1) má tvar:

$$\begin{aligned} \textit{symboly}: &\quad F,\ G\ (\textit{kresli vpřed},\ \ \textit{posun vpřed}) \\ \textit{start}: &\quad F \\ \textit{pravidla}: &\quad F \rightarrow FGF \qquad (3.10) \\ &\quad G \rightarrow GGG \qquad (3.11) \end{aligned}$$

Hausdorffova dimenze Cantorovy množiny je $D = \frac{\log 2}{\log 3} \approx 0,6309$. Zajímavostí je, že Cantorova množina je množina bodů na Kochově křivce, které protínají původní horizontální úsečku. Představuje také nejjednodušší fraktální objekt, na kterém lze vysvětlit pojmy fraktální geometrie (změna měřítka, soběpodobnost, ...). Cantorovu množinu získáme také řezem nespojitou Juliovou množinou.

3.2.2 Kochova vločka

Dalším známým L-systémem je Kochova vločka (křivka) pojmenovaná po Helge von Kochovi. Konstrukce, kterou lze vidět na obrázku 3.6, se provede tak, že se úsečku délky 1, rozdělí na tři části o délce $\frac{1}{3}$. Prostřední třetinu je poté nahrazena rovnostranným trojúhelníkem. Stejný postup je aplikován na všechny čtyři vzniklé úsečky. Takto se pokračuje až do nekonečna. První krok, v tomto případě úsečka délky 1, se nazývá iniciátor. Útvar v druhém kroku, kterým je úsečka nahrazována se nazývá generátor. Generování množiny spočívá v tom, že v každém kroku jsou nahrazeny všechny iniciátory generátorem. Detailněji viz [12].



Obrázek 3.6: Kochova vločka, konstrukce

Křivka má několik zajímavých vlastností, mnoho z nich je shodných s Cantorovou množinou. Je tvořena čtyřmi částmi, které jsou shodné s původní množinou, ale zmenšené v měřítku 1:3. Křivka neobsahuje žádné úsečky nebo hladké segmenty, rovněž v žádném bodě nemá derivaci. Výpočtem se dá zjistit, že délka k-té iterace při konstrukci množiny je rovna $(\frac{4}{3})^k$, pro $k \to \infty$ je tedy její délka rovna nekonečnu a křivka zabírá nulovou plochu. Hausdorffova dimenze křivky je $D = \frac{\log 4}{\log 3} \approx 1,2619$.

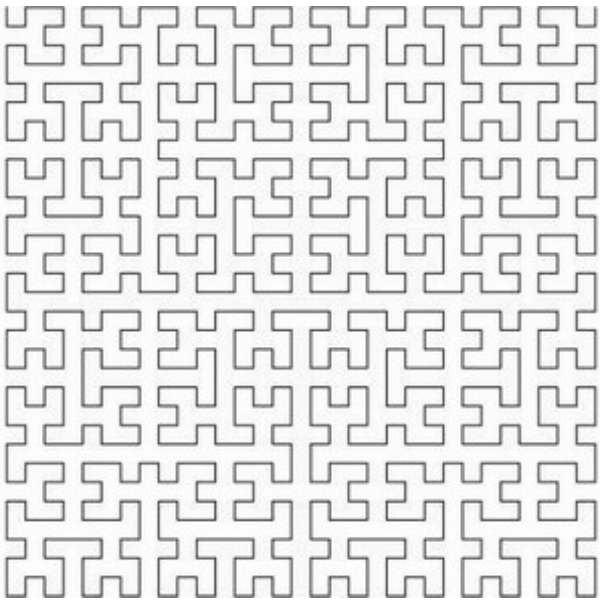
Kochovu vločku lze vygenerovat, za použití pravidel želví grafiky (viz tabulka 3.1), gramatikou s jediným přepisovacím pravidlem:

$$\begin{aligned} symboly: &\quad F\ (kresli\ \ vpřed) \\ konstanty: &\quad +,\ -\ (rotace\ \ doprava,\ \ doleva) \\ start: &\quad F--F--F \qquad (3.12) \\ pravidlo: &\quad F \rightarrow F+F--F+F \qquad (3.13) \end{aligned}$$

Kochova vločka byla pokládána za "matematické monstrum". V roce 1893 Charles Hermite v dopise napsal [12]: "odvracím se ve strachu a s hrůzou od tohoto bídného moru funkcí bez derivace. (anglicky *turning away in fear and horror from this lamentable plague of functions with no derivatives*)".

3.2.3 Hilbertova křivka

Hilbertova křivka je jedním z nejjednodušších fraktálů. Poprvé byla popsána v roce 1891 německým matematikem Davidem Hilbertem. Přestože je tvořena na sebe navazujícími úsečkami (jednodimenzionální objekt), vyplňuje celou plochu. Topologická dimenze Hilbertovy křivky $D_T = 1$, Hausdorffova dimenze $D = 2$, podobně jako u Mandelbrotovy množiny, která však nevyplňuje celou rovinu viz kapitola 3.1.2. Její délka je rovna $2^n - \frac{1}{2^n}$, což znamená, že roste exponenciálně s n (počet iterací).



Obrázek 3.7: Hilbertova křivka

Zkonstruování Hilbertovy křivky je složitější než v případě Kochovy vločky. Křivka vytvořená v n-té iteraci je složena ze čtyř křivek vzniklých v $n-1$ iteraci. Aby na sebe segmenty navazovaly, je nutno je otočit a zmenšit na polovinu. Slovní popis čtyř částí A, B, C, D Hilbertovy křivky nalezneme v tabulce 3.2 [25] a lze jej také jednodušše převést na algoritmus programovacího jazyka.

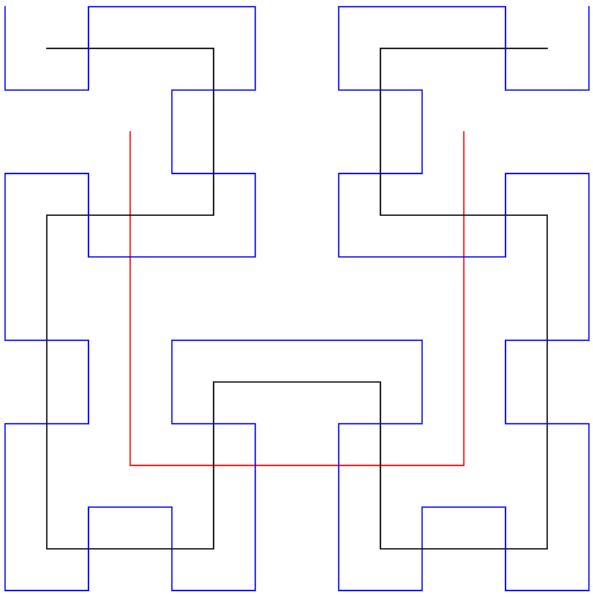
Symbol	Význam
A	D(doleva)A(dolů)A(doprava)B
B	C(nahoru)B(doleva)B(dolů)A
C	B(doprava)C(nahoru)C(doleva)
D	A(dolů)D(doleva)D(nahoru)C

Tabulka 3.2: Základní části Hilbertovy křivky

Gramatika pro vygenerování Hilbertovy křivky vyjádřená symboly želví grafiky (viz tabulka 3.1) má tvar:

$$
\begin{aligned}
\textit{symboly}: &\quad L,\ R \\
\textit{konstanty}: &\quad F,\ +,\ -\ (\textit{kresli vpřed},\ \textit{rotace doleva},\ \textit{doprava}) \\
\textit{start}: &\quad L \\
\textit{pravidla}: &\quad L \rightarrow +RF - LFL - FR + \qquad (3.14) \\
&\quad R \rightarrow -LF + RFR + FL - \qquad (3.15)
\end{aligned}
$$

Na obrázku 3.8 je vidět současné zobrazení první až třetí iterace Hilbertovy křivky. Jejich vlastností se využívá ve vícerozměrných databázích, kde je aplikována k mapování dat do jedné dimenze namísto z-křivky. Důvodem je, že takováto mapovací funkce lépe zachovává pozice prvků. Informace o aplikaci je možno najít v práci J. Lawdera a P. Kinga, např. "*Querying Multi-dimensional Data Indexed Using the Hilbert Space-Filling Curve*". Takovéto databázové algoritmy jsou chráněny patenty, podobně jako algoritmy pro fraktální analýzu a kompresi obrazu, která je zmíněna v kapitole 3.3.



Obrázek 3.8: Tři iterace Hilbertovy křivky

3.3 IFS fraktály

Název IFS systémů je odvozen z původního anglického označení *Iterated Function System*, česky lze tento název přeložit jako systém iterovaných funkcí. První publikace, které se týkaly IFS systémů, vydali v roce 1985 Demko a v roce 1987 Barnsley. IFS je generativní metoda vytváření fraktálů, kterou řadíme mezi deterministické. Algoritmus pro generování IFS fraktálů však může být jak deterministický, tak nedeterministický (tj. stochastický, viz kapitola 5.3.1, *algoritmus RWA*).

V systémech iterovaných funkcí IFS se pro tvorbu fraktálních obrazců používá takzvaná generativní metoda [3, 25]. Tato se používá také ke kompresi bitmapových obrazů, zavedeme-li ke generativní metodě takzvanou inverzní úlohu. Při generování fraktálů pomocí IFS systémů se při výpočtech používá náhodných prvků, tj. metoda je stochastická. Deterministické vytváření se používá také, oba přístupy však vedou ke stejnému výslednému fraktálu, použije-li se dostatečný počet iterací.

Systémy iterovaných funkcí tvoří velmi důležitou skupinu lineárních deterministických fraktálů, které mají v počítačové grafice více uplatnění. Praktický návod jak používat IFS pro generování rastrových textur je v Barnsleyho knize *Fractals Everywhere* [3]. IFS se skutečně pro tuto činnost používají i v běžných aplikacích, jako např. počítačové hry, nebo v aplikacích sloužících k tvorbě procedurálních těles. Pro tvorbu rastrových obrázků existuje plugin *IFS Composer*, pro editor GIMP. Detailní popis využití IFS viz je možno najít na webových stránkách Root.cz [25], díl XXX a dále a samozřejmě v knihách Michaela Barnsleyho [3, 2].

Ve skupině IFS fraktálů je možné nalézt útvary charakteristické pro jiné skupiny fraktálů např. Sierpinského kobereček, obrázek 3.10 nebo Mengerovu houbu, viz obrázek 3.11. IFS, které jsou generalizací *Rhamovy křivky* [22], můžeme rozdělit podle výsledného vzhledu, kdy dostáváme většinou vzhled blízký rostlinám, např. obrázek 3.9, nebo vizuálně atraktivní tzv. *Fractal Flames* popsané v kapitole 3.3.3.

Obrázek 3.9: Kapradina vygenerovaná IFS

Druhou možností jak IFS kategorizovat může být jejich rozdělení podle použitého generujícího algoritmu. Některé jsou popsány v kapitole 5, např.:

- Algoritmus náhodné procházky (*Random Walk Algorithm*),
- Deterministický algoritmus (*Deterministic Iteration Algorithm*),
- Algoritmus pro generováni minima pixelů (*Minimal Plotting Algorithm*).

Výhodnost uspořadání transformací do IFS systému vyústila ve vznik nového grafického formátu *FIF*, který využívá IFS komprimaci a dosahuje lepších výsledků než běžně používaný formát JPEG. FIF formát však není příliš rozšířen pro svou vysokou výpočení náročnost, zejména při komprimaci (až několik řádů proti JPEG). Není proto vhodný pro použití kompresního algoritmu ve spotřební elektronice. Dalším faktorem hovořícím proti širšímu využití formátu FIF jsou patenty chránící postup hledání transformací, které vlastní M. Barnsley a jeho spolupracovníci. Oproti tomu dekompresní algoritmy jsou volně šiřitelné.

Systémy iterovaných funkcí jsou množiny zobrazení v rovině či prostoru a IFS fraktál je pak popsán několika zobrazeními [25, 3] označovanými $\phi_1, \ldots, \phi_n$. Tyto zobrazení tvoří *množinu zobrazení* Φ, která obsahuje jednotlivá použitá zobrazení. Většinou jde o *lineární transformace* (posun, zkosení, změna měřítka atd.), někdy se však používají také *nelineární transformace* (např. ohyb, zkrut, ...).

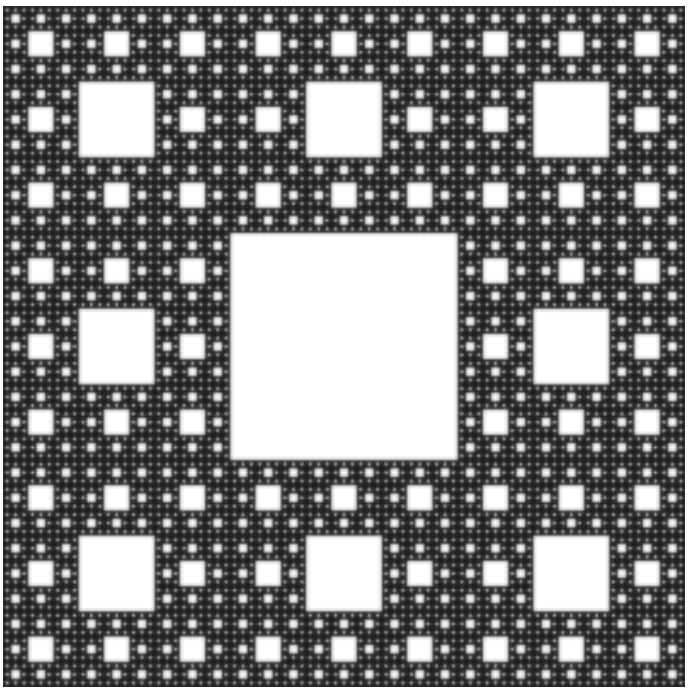
V IFS se objevuje pojem "pevného bodu" (tj. takový, který se aplikací nějaké funkce f mapuje sám na sebe), jež je výchozím bodem pro generování systému iterovaných funkcí. Základními transformacemi jsou kontrakce (zkracuje vzdálenosti mezi body) a expanze (zvětšení vzdálenosti mezi body), ačkoliv expanze je využívána méně. Pro danou transformaci existuje právě jeden pevný bod. **IFS mohou být definovany jakýmkoliv prostorem, který má definován pojem vzdálenosti** [25].

3.3.1 Sierpinského kobereček

Sierpinského kobereček, obrázek 3.10, je rovinný fraktál, který jako první popsal Waclav Sierpinski v roce 1916. Je to jedna z generalizací Cantorovy množiny do dvourozměrného prostoru. Sierpinsky ukázal, že je to univerzální křivka, ve které jakýkoliv 1D graf zobrazený do 2D plochy je homeomorfní s určitou podmnožinou koberečku.

Tento útvar je modifikací Sierpinského trojúhelníku, který generovaný L-systémem znázorňuje obrázek 3.4. Základním tvarem namísto trojúhelníku je použit čtverec.

Konstrukce se provádí tak, že se jednotkový čtverec rozdělí na devět stejných dílů a odstraní se prostřední z nich. Stejným způsobem se postupuje s osmi zbývajícími čtverci do nekonečna (v případě programu do maximálního počtu iteračních kroků). Hausdorffova dimenze (vzorec 2.5) útvaru $D = \frac{\log 8}{\log 3} \approx 1,8928$. Sierpinského kobereček má nekonečně velký obvod, avšak nekonečně malou plochu.

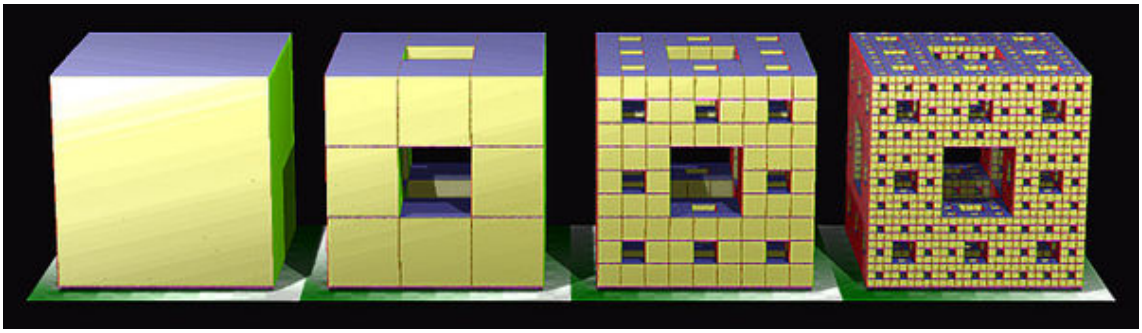


Obrázek 3.10: Sierpinského kobereček vygenerovaný IFS

3.3.2 Mengerova houba

Mengerova houba byla poprvé popsána Karlem Mengerem v roce 1926. Je zobecněním Sierpinského koberce v trojrozměrném prostoru a zachovává si jeho vlastnosti (je univerzálním objektem pro všechny křivky a grafy).

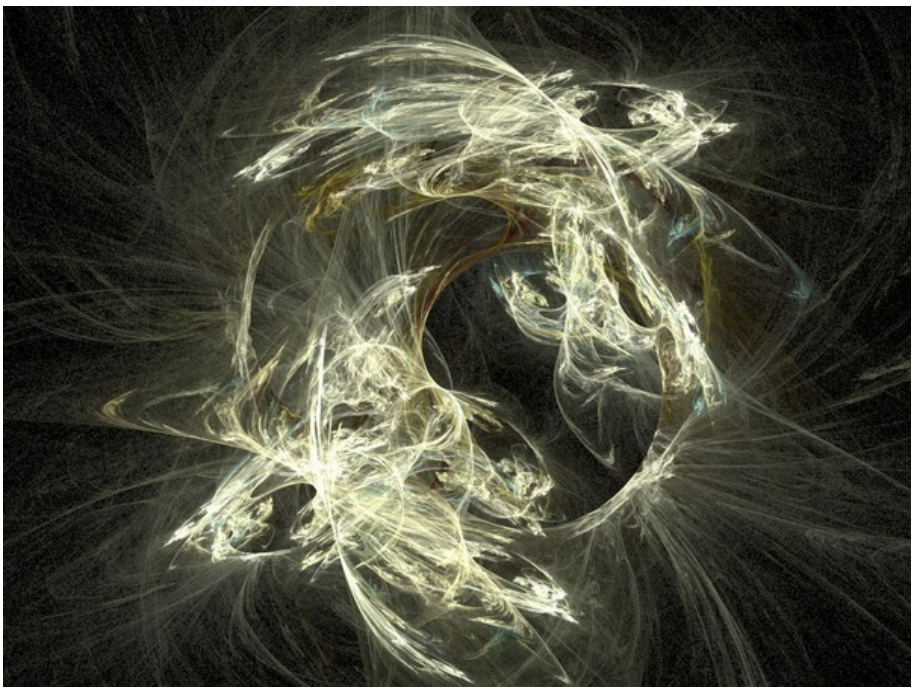
Mengerova houba vzniká postupným dělením krychle vždy na 27 menších krychlí, vždy se odebere 7 krychlí, které se nacházejí uprostřed stěn a uvnitř krychle. Tento postup se opakuje do nekonečna. První čtyři iterace znázorňuje obrázek 3.11. Počet krychlí narůstá s mocninou 20^n, kde n je počet iteračních kroků. Mengerova houba má nekonečně velký povrch, ale nekonečně malý objem. Její Hausdorffova dimenze (vzorec 2.5) $D = \frac{\log 20}{\log 3} \approx 2,726833$.



Obrázek 3.11: Mengerova houba

3.3.3 Fractal Flames

Tuto skupinu fraktálů nedávno pojmenoval Scott Draves [5]. Jedná se o generalizaci IFS, která byla vytvořena s ohledem na vizuální kvalitu výsledných obrázků, např. obrázek 3.12. Menší důraz je kladen na matematickou čistotu algoritmu [25].



Obrázek 3.12: Fractal Flame

Oproti klasickým IFS jsou používány pouze některé funkce, zejména nelineární. Závislost světlosti pixelu v obrazci při jeho "zásahu" je logaritmická a při vykreslování se využívá symetrie.

U tohoto druhu fraktálů je rozšířena skupina IFS o další funkce [25, 5]. Pro oba druhy (IFS i "flame") se ke generování používá algoritmus náhodné procházky (*RWA* viz kapitola 5.3.1), jímž se na jednotlivé body vybírají a aplikují transformace. Tyto transformace jsou však oproti IFS složené z jednoduché lineární a složitější nelineární funkce zvané *variace*.

Označení funkce	**Výraz**	**Původní název**
$V_0(x,\, y)$	$(x,\, y)$	linear
$V_1(x,\, y)$	$(\sin x,\, \sin y)$	sinusoidal
$V_2(x,\, y)$	$(\frac{x}{r^2},\, \frac{y}{r^2})$	spherical
$V_3(x,\, y)$	$(r\cos(\Theta + r),\, r\sin(\Theta + r))$	swirl
$V_4(x,\, y)$	$(r\cos(2\Theta),\, r\sin(2\Theta))$	horseshoe
$V_5(x,\, y)$	$(\frac{\Theta}{\pi},\, r - 1)$	polar
$V_6(x,\, y)$	$(r\sin(\Theta + r),\, r\cos(\Theta - r))$	handkerchief
$V_7(x,\, y)$	$(r\sin(\Theta r),\, -r\cos(\Theta r))$	heart
$V_8(x,\, y)$	$(\Theta\frac{\sin(\pi r)}{\pi},\, \Theta\frac{\cos(\pi r)}{\pi})$	disc
$V_9(x,\, y)$	$(\frac{\cos\Theta+\sin r}{r},\, \frac{\sin\Theta-\cos r}{r})$	spiral
$V_{10}(x,\, y)$	$(\frac{\sin\Theta}{r},\, r\cos\Theta)$	hyperbolic
$V_{11}(x,\, y)$	$((\sin\Theta)(\cos r),\, (\cos\Theta)(\sin r))$	diamond
$V_{12}(x,\, y)$	$(r\sin^3(\Theta + r), r\cos^3(\Theta - r))$	ex
$V_{13}(x,\, y)$	$(\sqrt{r}\cos(\frac{\Theta}{2} + \Omega),\, \sqrt{r}\sin(\frac{\Theta}{2} + \Omega))$	julia
$V_{16}(x,\, y)$	$(\frac{2r}{(r+1)}(x,\, y)$	fisheye
$V_{20}(x,\, y)$	$(\cos(\pi x)\cosh(y), -\sin(\pi x)\sinh(y))$	cosine

Tabulka 3.3: Variace pro Fractal Flame

Nelineární funkce způsobují, že se ve výsledném fraktálu často objevují části ve tvaru spirál, vln atd. Jako další se používá symetrie, ať už středová nebo osová.

Obarvování pixelů je oproti IFS pozměněno tak, aby byly zvýrazněny zajímavé části fraktálu. Mezi barevností a počtem "zásahů" pixelů je logaritmická závislost. Možné je i obarvení podle použité transformace. Prostor obrázku je rozšířen o třetí rozměr, představující index do barevné palety či odstín barvy.

V klasických IFS je transformační matice $\mathbf{A}$ tvořena šesti koeficienty $(a_i,\, b_i,\, c_i,\, d_i,\, e_i,\, f_i)$. U Fractal Flame je přidána nelineární funkce. Draves zavedl afinní funkci 3.16 [5], kde V_j je právě variací, která mění vzhled fraktálu specifickým způsobem.

$$F_i(x,\, y) \;=\; V_j(a_i x + b_i y + c_i,\, d_i x + e_i y + f_i) \tag{3.16}$$

Příklady variací V_j, jež navrhl Scott Draves, jsou uvedeny v tabulce 3.3. Jedná se o kombinace např. goniometrických funkcí a jsou označovány $V_n(x,\, y)$, kde n nabývá hodnot 0 až 20 [25, 5].

Význam symbolů v tabulce 3.3 je následující:
$r = \sqrt{x^2 + y^2}$
$\Theta = \arctan(\frac{y}{x})$
$\Omega = 0$ *nebo* π (na základě náhodného čísla).

Další modifikací zavedenou Scottem Dravesem je již výše zmíněná logaritmická závislost barvové intenzity pixelů na počtu jejich zásahů. Tímto je dosaženo zvýraznění i těch částí fraktálu, které by byly použitím lineární závislosti příliš tmavé. Také překrytí světlého motivu tmavším nezpůsobí jeho "zmizení" [25, 5]. Při počtu zásahů např. 1000 (světlejší) a 100 (tmavší) je při překrytí obrazců počet zásahů roven 1100, při logaritmování je však rozdíl v hodnotě mezi $\log 1000$ a $\log 1100$ malý a tudíž i výsledná barva se liší jen velmi málo. Logaritmické obarvování umožňuje odlišení nepříliš rozdílných intenzit v jedné části obrazce od řádově menších intenzit, avšak také si blízkých intenzit v části jiné. Tímto vzniká další efekt - obrázky se jeví trojrozměrné, přestože jsou tvořeny pouze plošnými variacemi. Pro ilustraci jsou uvedeny obrázky 3.12, 3.13, kde lze prostorovou iluzi pozorovat.

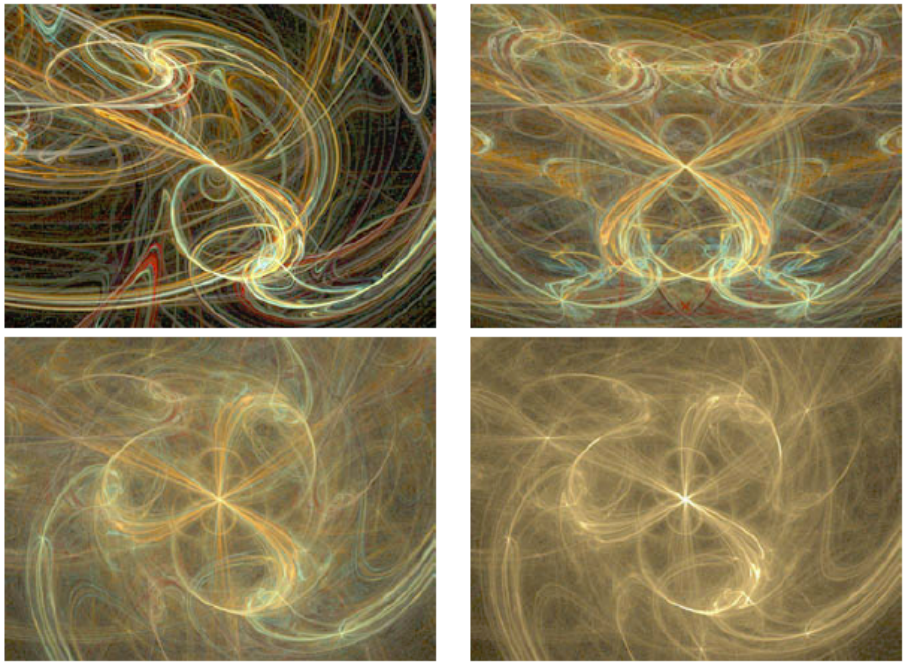
Obrázek 3.13: Fractal Flame, Heart

Technika obarvování pixelů je oproti IFS rozšířena. Stále částečně závisí na použité transformaci, přestože se používá počtu zásahů a hustoty pixelů [5, 25]. I přes zdánlivý 3D vzhled zůstává vnitřní struktura fraktálů neprůhledná. Tohoto je dosaženo přidáním třetí souřadnice k 2D transformaci. Třetí souřadnicí je barva pixelu (intenzita, případně index do barvové palety).

Výpočet barvy se provádí až v průběhu iteračního cyklu. Její počáteční intenzita, či index c, je zvolena náhodně v rozsahu 0 až 1. V každém kroku se pak provádí přepočet barvy nebo intenzity podle vztahu $c = \frac{c+c_i}{2}$, což má za následek, že nejvíce používané transformace nejvíce ovlivní barvu pixelu. Výsledná barva je součtem barvy nově vygenerovaného pixelu s barvou předchozí a vznikají takto postupné barevné přechody způsobené "pamětí systému".

Symetrie u tohoto druhu fraktálu je využívána poměrně často. Musí však být použito vhodných prostředků, jinak dochází ke "ztrátě" barevnosti výsledného obrazce, jak lze vidět na obrázku 3.14 [5]. Barvy jsou zprůměrovány a tímto ztrácí fraktál na své pestrosti. Řešením tohoto problému je ponechání původní barvy pixelu i po aplikaci funkcí symetrie.

Volba vhodné symetrie se provádí nejčastěji po výpočtu souřadnic pixelů [25], kdy jsou vykresleny kopie pixelu v pozicích osově či středově souměrných s původně vypočteným pixelem. Další možností je výběr vhodných funkcí při výpočtu fraktálu. Tato možnost se využívá častěji kvůli přirozenějšímu vzhledu, protože je v tomto případě symetrie méně dokonalá. Osová symetrie spočívá ve změně znaménka jedné ze souřadnic, středová symetrie se provádí aplikací funkce, která provede rotaci bodu kolem počátku souřadného systému. Pravděpodobnost aplikací symetrických funkcí by měla být rovna součtu pravděpodobností ostatních funkcí [5, 25].



Obrázek 3.14: Nevhodně ošetřená symetrie

Fractal Flame dosud nenašel své uplatnění v některém z technických oborů, využívá se především pro svou vizuální efektnost výhradně k dekorativním účelům.

3.4 Stochastické fraktály (nepravidelné fraktály)

Další velkou skupinou jsou fraktály stochastické. Zatímco všechny předchozí typy byly v určitém smyslu symetrické, nepravidelné fraktály vnášejí při generování fraktálu do algoritmu náhodu a jsou pouze tzv. **soběpříbuzné** [25]. Tento typ fraktálů také umožňuje nejlepší popis přírodních objektů. Při generování tvarů rostlin klasickými L-systémy nebo IFS systémy je výsledek perfektně symetrický, avšak ve skutečnosti je tomu zcela jinak. Strom rostoucí v přírodě má nepravidelné délky a tloušťky větví, úhel růstu také není vždy stejný, strom rostoucí v lese má jiný tvar než tentýž strom rostoucí osamoceně apod. Proto je vhodné při generování fraktálů uplatnit prvky náhody. Způsob, jakým se náhodnost bude podílet při generování fraktálů, bude vždy určovat tvar fraktálu i jeho Hausdorffovu dimenzi. Pro generování náhodných čísel se používá například Gaussovský generátor nebo generátor bílého šumu.

Náhodné fraktály lze vytvářet více způsoby. První způsob spočívá v simulaci Brownova pohybu v ploše nebo v prostoru. Druhý způsob spočívá v použití metody přesouvání středního bodu a třetí možností je využití spektrální syntézy.

3.4.1 Simulace Brownova pohybu

Simulace Brownova pohybu vytváří fraktální objekt, jehož Hausdorffova dimenze je úměrná absolutní velikosti změny při jednom kroku iterace. Tato metoda se používá například při generování toků řek, není však vhodná pro trojrozměrné objekty ze stejného důvodu, jaký vedl k problémům u trojrozměrných systémů iterovaných funkcí IFS [12, 25, 24]. Vytvořený model se skládá z jednotlivých bodů a dnešní grafické akcelerátory jsou spíše optimalizovány na vykreslování trojúhelníků a dalších plošných entit. Také vytváření obrazů difúze je založeno na Brownově pohybu.

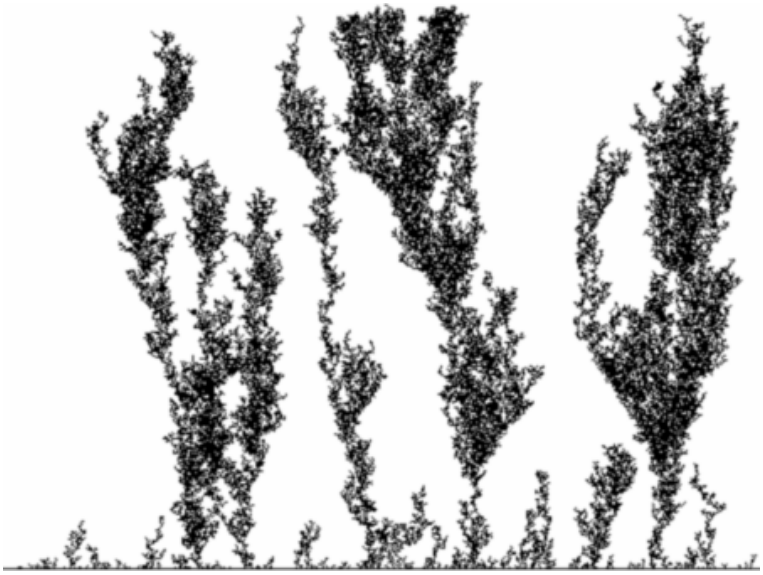
Simulace difúze

Simulace difúze je minimálně využívaná metoda [25], která je však schopna vytvářet složité přírodní útvary. Nejčastěji se jedná o modely stromů, keřů, trávy. Prostorové útvary generované touto metodou charakterizuje částečná soběpodobnost a velká tvarová složitost. Rostlinu vygenerovanou simulací difúze lze vidět na obrázku 3.15.

Schopnosti metody rozšiřuje nová metoda návrhu omezujících podmínek pro generování prostorových fraktálů, která je uvedena na Root.cz [25]. Omezující podmínky definují charakteristiku vytvářených útvarů, kdežto parametry algoritmu ovlivňují celkový vzhled modelu, jeho hustotu, ale také jeho tvar. Vygenerované útvary jsou složeny z orientovaných elementárních prvků plochy, či neorientovaných bodů, nacházejících se v prostoru E^3 (částice) nebo v ploše E^2 (body, po přepočtu pixely). Dimenze těchto fraktálů se může pohybovat od jedné do tří.

Metoda není využívána pro tvorbu přírodních těles z následujících důvodů [25]:

- Vytváření modelu je sice algoritmicky jednoduché, avšak časově velmi náročné. Časovou složitost lze minimalizovat omezením prostoru, v němž se nové body vytvářejí.
- Fraktál je tvořen množinou jednotlivých bodů nebo plošek bez vzájemných vazeb. Převod těchto primitiv je obtížný, jelikož vzniká velké množství polygonů, které je nutno vykreslit.



Obrázek 3.15: Rostlina vygenerovaná pomocí simulace difúze

Pomocí simulace difúze je možno vygenerovat mnoho zajímavých obrázků s fraktální strukturou. Metoda je založena na pohybu částic v omezeném nebo neomezeném prostoru [25]. Z pevných bodů v rovině či prostoru postupně "vyrůstají" další body, které lze k již vytvořenému útvaru přidávat dvěma způsoby:

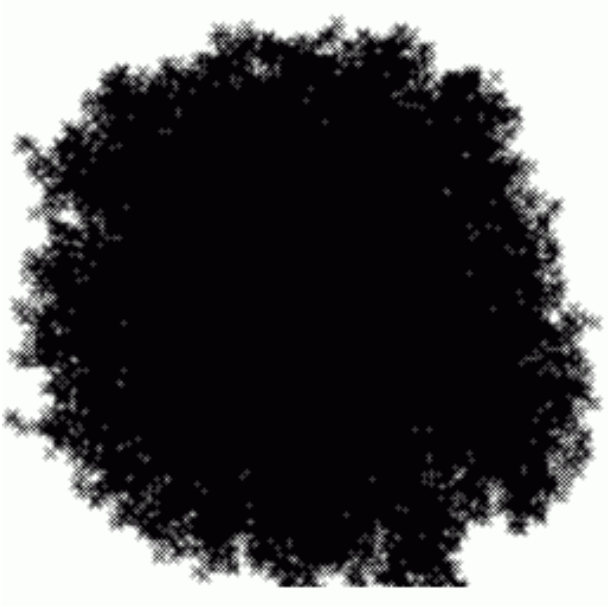
- Částice se vytvoří na náhodném místě a simuluje se její další pohyb v prostoru vyhrazeném pro fraktál. Pokud se částice dotkne již existující části útvaru, připojí se k němu a vygeneruje se další částice. Princip vychází z Brownova pohybu, viz obrázek 3.16.
- Na náhodných místech se postupně vytvářejí statické částice. Pokud se dotknou stávajícího útvaru, jsou k němu připojeny, jinak zaniknou. Tato technika není plně v souladu s Brownovým pohybem, vytvářením nových částic se aproximuje trajektorie jedné částice, avšak pouze v některých časových bodech pohybu.

Urychlení simulace difúze

Urychlení metody simulace difúze je vhodné kvůli nízké efektivnosti jejího provádění na počítačích. Zobrazení fraktálu metodou popsanou výše je časově náročné. Kratší doby procesu generování lze dosáhnout omezením prostoru, ve kterém jsou vytvářeny nové body. Důvodem je zvýšení pravděpodobnosti dotyku nové částice s existujícím obrazcem [25].

Pro vytváření stromů a keřů se omezení prostoru realizuje pomocí osově orientovaného kvádru. Jeho pozice se postupně mění, většinou se posouvá vzhůru od základny po vygenerování dostatečného počtu částic. Kvádr je dále možno dělit na podoblasti, které se mohou překrývat v závislosti na požadovaném výsledku.

Další možností urychlení urychlení výpočtu je diskretizace prostoru pro generování částic. Místo spojité plochy či prostoru je použita pevně daná mřížka. Souřadnice i, j, k pak plně určují pozici částice v mřížce. Vzdálenost dvou bodů přejde v určení nejmenšího počtu



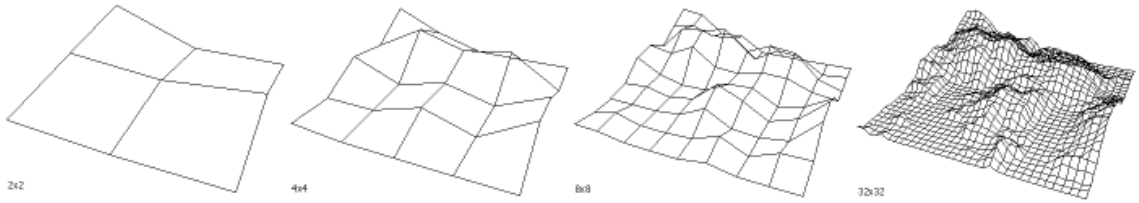
Obrázek 3.16: Difúze vygenerovaná z jednoho bodu

buněk mezi nimi, což je značně rychlejší než výpočet Euklidovské vzdálenosti ve spojitém prostoru.

Dotyk částic s vygenerovaným útvarem může být díky diskretizaci prostoru řešen pouze jako testování nejbližšího okolí bodu podle toho, zda se nacházíme v rovině či prostoru. V rovině se testuje čtyřokolí nebo osmiokolí bodu, v prostoru pak šestiokolí, osmnáctiokolí či 26 sousedních bodů, podle toho v kolika indexech se liší souřadnice nového bodu a testovaných buněk.

3.4.2 Metoda přesouvání prostředního bodu

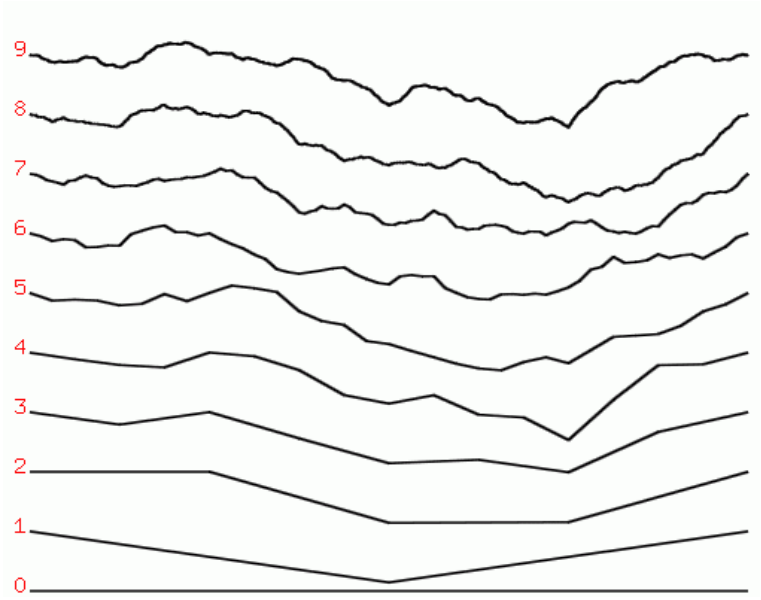
Metoda přesouvání prostředního bodu (*Midpoint Displacement Method, MDM*) patří mezi typické metody generování stochastických fraktálů plošných i prostorových. V počítačové grafice se velmi často používá k vygenerování takzvaného výškového pole (height field) a následné vizualizaci přírodní krajiny. Volbou maximální odchylky Δ, která určuje Hausdorffovu dimenzi vytvořené krajiny při posunu prostředního bodu, lze měnit celkový ráz povrchu od zvlněné krajiny až po velehory s rozeklanými vrcholky, viz obrázek 3.17. Rozšířením je možno touto medotou generovat reálně vypadající mraky [12, 25, 1], případně modifikací algoritmu na plošný lze generovat obrazce zvané plazma [20].



Obrázek 3.17: Krajina - přesouvání prostředního bodu

Nejjednodušší aplikací MDM je aplikace postupu na úsečku. Příklad dělení, ilustrovaný obrázkem 3.18, lze slovně popsat takto [25, 14]:

- mějme horizontální úsečku,
- pro zadaný počet iterací i_{max} opakujeme:
  - nalezneme prostřední bod P_s^i každé úsečky (segmentu) v útvaru,
  - posuneme prostřední bod P_s^i ve vertikálním směru o náhodnou hodnotu udanou koeficientem Δ^i,
  - zredukujeme koeficient Δ^i pro příští iteraci.



Obrázek 3.18: MDM - 9 iterací dělení úsečky

Změna Δ Hurstovým exponentem

Díky tomu, že se v každém kroku snižuje rozptyl hodnot koeficientu Δ, zmenšují se také rozdíly mezi posunutím prostředních bodů úseček, což ústí v relativně malou variabilitu

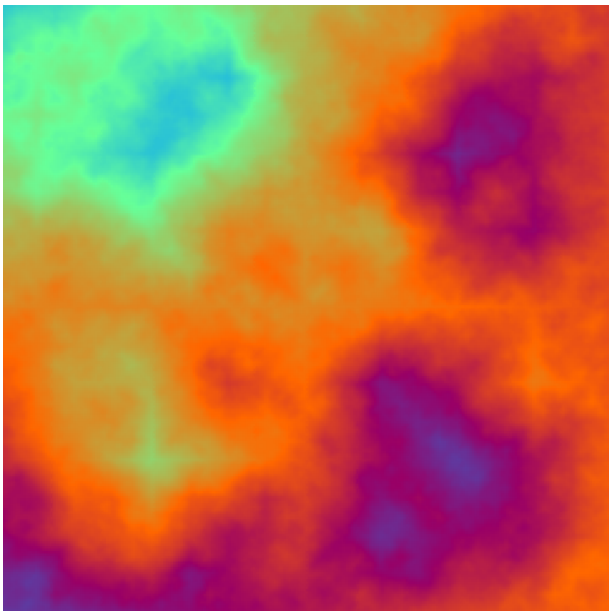
vygenerovaných křivek s Hausdorffovou dimenzí $D = 1,5$. Aby bylo možné nastavit D před výpočtem, je vhodné modifikovat MDM následujícím způsobem [25, 13]:

- generátor náhodných čísel je třeba změnit na generátor s Gaussovským rozložením, nejčastěji výpočtem průměru několika náhodných čísel,
- obrazy s různou D vyžadují změnu rozptylu v Gaussově rozložení. Vztah pro rozptyl i-té iterace je $\sigma_i^2 = \sigma^2/(2^{2H(i+1)})$, kde H je *Hurstův exponent* [12], který určuje Hausdorffovu dimenzi: $D = 2 - H$.

Rekurzivní dělení čtverce

Metoda přesouvání prostředního bodu může být použita také na plošný útvar, kdy se při rekurzivním dělení posouvají krajní body čtverce v kolmém směru. Nejčastější varianta využívaná v praxi spočívá v tom, že se v každém kroku vypočítají výšky bodů v polovině hran čtverce a z nich pak výška středu čtverce. Střed je poté posunut o náhodnou hodnotu a vznikají čtyři části, které jsou dále děleny [14, 20, 25].

Využití tato technika nachází při generování výškových map terénu. Také lze vytvářet obrázky, které jsou nazývány plazma, viz 3.19. Plazma se tvoří tak, že čtverec je nahrazen obrázkem, kde pozice každého pixelu odpovídá souřadnicím bodu v rovině xy a barva pixelu odpovídá souřadnici z. Generování postupně čtverec dělí na čtvrtiny a končí v okamžiku, kdy zbývají čtverce o velikosti jednoho pixelu.



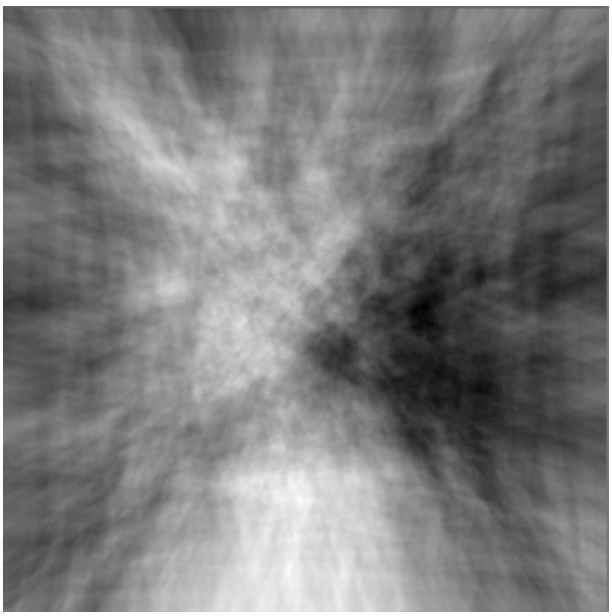
Obrázek 3.19: Plazma

Obrázky plazmy vytvořené MDM vypadají realisticky. Jak je patrné na obrázek 3.19, u prvních iterací dělení čtverce vznikají na hranicích svislé a vodorovné barevné přechody. Na nižších úrovních již nejsou patrné kvůli snižování hodnoty koeficientu Δ. Přechody vznikají z důvodu posouvání pouze prostředního bodu čtverce a ponecháním středů jeho hran na původní pozici kvůli chybějícím informacím o následujícím dělení sousedních čtverců.

Na odstranění nebo potlačení barevných přechodů existuje několik metod. Obvykle spočívají v odlišném dělení čtverce střídavě po úhlopříčkách a vodorovných nebo svislých hranách např. diamanto-čtvercový algoritmus (*diamond-square algorithm* [14]). Zajímavou metodou uvádí Pavel Tišnovský [25]. Jde o iterativní generování různě orientovaných přímek a další úpravu rozděleného obrazu následujícím způsobem:

- intenzitu všech pixelů ležicích nalevo od přímky **sniž** o jedničku,
- intenzitu všech pixelů ležících napravo od přímky **zvyš** o jedničku.

Pro dosažení uspokojivého výsledku je třeba generovat přímky a měnit intenzitu v mnoha cyklech, obvyklý počet opakování je 1000 - 10 000 iterací. Barevný přechod pak není viditelný, ale vygenerovaný obrázek je třeba normalizovat, aby nejméně intenzivní pixely byly černé a naopak nejintenzívnější pixely byly bílé. Plazma vytvořená touto metodou je na obrázku 3.20.



Obrázek 3.20: Plazma vytvořená generováním přímek

3.4.3 Spektrální syntéza

Spektrální syntéza je často používanou metodou, která vychází z principu výpočtu Fourierovy řady. Metoda spočívá v tom, že se náhodně vygenerují Fourierovy obrazy, které mají spektrální hustotu úměrnou zadané Hausdorffově dimenzi. Jestliže je známa spektrální hustota, lze nalézt koeficienty A_k a B_k a provést s nimi inverzní Fourierovu transformaci, viz 3.17 (převod z frekvenční oblasti do oblasti časové).

$$X(t) = \Sigma(A_k \cos(kt) + B_k \sin(kt)) \tag{3.17}$$

Spektrální hustotu je možno vypočítat pomocí vztahu 3.18:

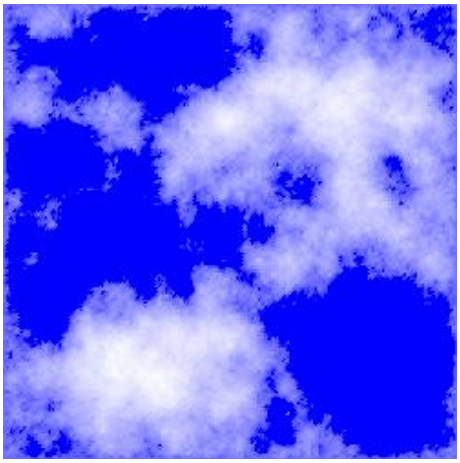
$$S(f) = \lim_{T \to \infty} \frac{1}{T} |X(u)|^2 \tag{3.18}$$

Spektrální hustota $S(f)$ pro zlomkovitý Brownův pohyb je úměrná hodnotě $1/f^{\beta}$. Koeficient β je ve vztahu s Hurstovým koeficientem [12] a sice $\beta = 2H + 1$. Pro požadovanou Hausdorffovu dimenzi D je možno vypočítat β a následně spektrální hustotu 3.18, která musí odpovídat hodnotě $1/f^{\beta}$ [25]. Fourierovy obrazy dané spektrální hustoty je možné získat generováním koeficientů A_k a B_k tak, aby vyhovovaly vztahu 3.19:

$$E(A_k^2 + B_k^2) \sim \frac{1}{k^{\beta}} \tag{3.19}$$

Rozšířením metody spektrální hustoty do dvourozměrného prostoru je možné opět generovat plazmu, případně výškové pole, čehož se využívá při generování modelů krajiny (obrázek 3.21) či povrchů planet. Dalším rozšířením o jednu dimenzi vzniká trojrozměrná mřížka, ve které jednotlivé voxely mohou vyjadřovat hustotu prostoru na jejich souřadnicích. Tímto způsobem je pak možno generovat modely reálně vypadajících mraků 3.22 [12, 25, 24, 14, 1].

Obrázek 3.21: Krajina - spektrální syntéza



Obrázek 3.22: Fraktální oblaka

Kapitola 4

Fraktály v praxi

Poznatky získané fraktální geometrií jsou postupně zaváděny do praxe, kde našly využití v biologii, chemii, medicíně a samozřejmě v počítačové grafice. V dnešním moderním světě se setkáváme s fraktály velmi často, ikdyž si nemusíme být vědomi skutečnosti, že se na fraktál díváme či poznatky získané fraktální geometrií využíváme. V dalším textu je uvedeno několik typických příkladu, kde je možné se s fraktály setkat, případně je prakticky využít. Mnoho dalších aplikací lze nalézt např. v literatuře [12, 11, 3, 2, 25, 22].

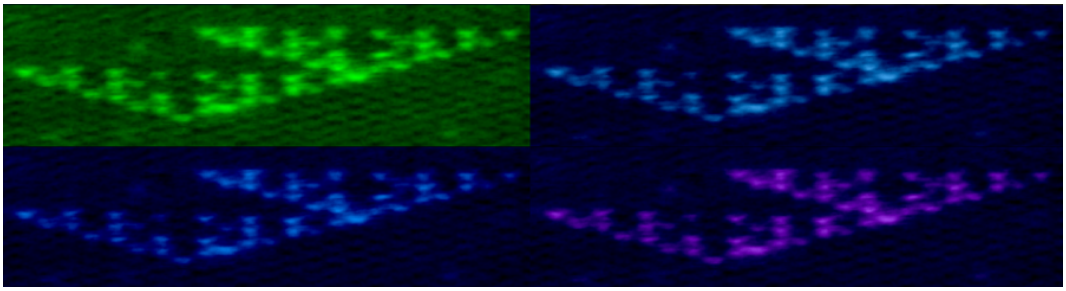
S rozvojem výpočetní techniky lze analyzovat mnohem větší objemy dat, které bylo nutno předtím zjednodušovat. Tak se fraktální geometrie promítla i do oborů jako je ekonomie, kde je využívána při analýze grafů popisujících např. ceny akcií na burze, výkonnost ekonomiky atd. Tyto křivky jsou v ekonomii nazývány *Elliottovy vlny* a používají se k předpovědi vývoje trhu. V devadesátých letech se objevila *Hypotéza fraktálového trhu*, která je takto označovaná kvůli předpokladu zachovávání stejných charakteristických rysů prvků operujících v tomto prostředí, ikdyž v jiném časovém měřítku [15].

4.1 Biologie a lékařství

V biologii a medicíně se fraktální geometrie využívá při zkoumání vlastností krve, kdy se měří Hausdorffova dimenze jejich složek. Vědci také zkoumají souvislost Hausdorffovy dimenze povrchu mozku a inteligence člověka.

Při zkoumáni signálu EKG byly zaznamenány nepravidelnosti srdečního rytmu typické pro nelineární dynamické systémy. Na základě poznatků fraktální geometrie byly vyvinuty chaotické filtry, které dokáží odlišit signál EKG plodu a matky od případného dalšího šumu. Srdeční rytmus reaguje na činnost mozku a nervové soustavy. Touto problematikou se zabývá relativně mladý vědní obor zvaný *Fraktální analýza variability srdečního rytmu*. Nové poznatky se diskutují především v odborných časopisech, na konferencích a na odborných fórech např. www.ams.org a stránkách Gerstnerovy laboratoře ČVUT.

Zajímavé chování je také možno vypozorovat v chování řetězce DNA. Na základě preferovaných vazeb mezi složkami DNA adenin-thymin a guanin-cytosin, lze "naprogramovat" růst krystalu definovaným způsobem. Molekuly DNA se pak na základě lokálních pravidel chovají jako stavební bloky pro větší globální strukturu [18, 19]. Pro vytvoření Sierpinského trojúhelník na obrázku 4.1 bylo použito DNA s dvojitým překřížením.



Obrázek 4.1: Fraktální vzor DNA

4.2 Počítačová grafika

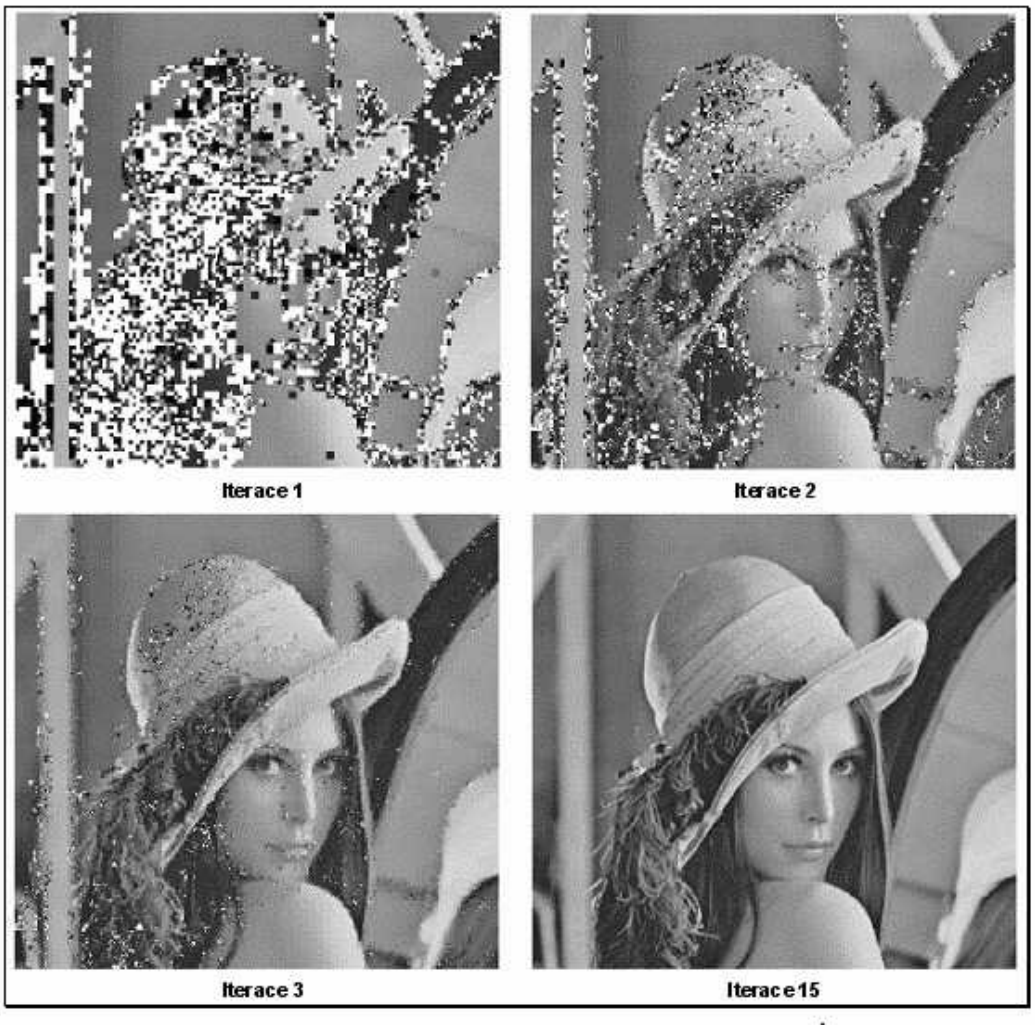
Největší zastoupení využití fraktální geometrie je v oboru počítačové grafiky. Fraktály zde našly uplatnění v mnoha oblastech potřebných pro práci s informacemi.

Komprese dat je typickou oblastí využití systémů iterovaných funkcí, viz kapitola 3.3 a literatura [3, 2, 17, 16]. Dále lze fraktálními technikami generovat textury, které jsou paměťově velmi nenáročné (mnohdy až 10x menší objem dat) a netrpí zkreslením při zvětšení jako bitmapové textury. IFS jsou vhodné i pro kompresi dat, jelikož jedna transformace je v tomto případě zapsána pomocí osmi čísel. Tuto metodu využívá grafický formát FIF, více informací viz literatura [3, 25, 17]. Fraktální komprese je ztrátová a dosahuje při vyšších stupních vizuálně lepších výsledků, ačkoliv porovnání je nesnadné z důvodu odlišných projevů ztráty kvality oproti např. JPEG.

Příklad fraktální rekonstrukce fotografie 4.2 pomocí černé plochy vidíme na obrázek 4.3. Původní obrázek byl analyzován a pomocí hierarchického členění byly vyhledány podobné podoblasti pro jeho komprimaci [17].

Obrázek 4.2: Původní Lenna

Generování pseudopřírodních útvarů bylo zmíněno u stochastických fraktálů v kapitole 3.4. Fraktály se užívají hlavně ke generování útvarů, jako jsou stromy, mraky, rostliny, kameny apod. Ikdyž jsou tyto metody úspornější na množství potřebných dat k popsání

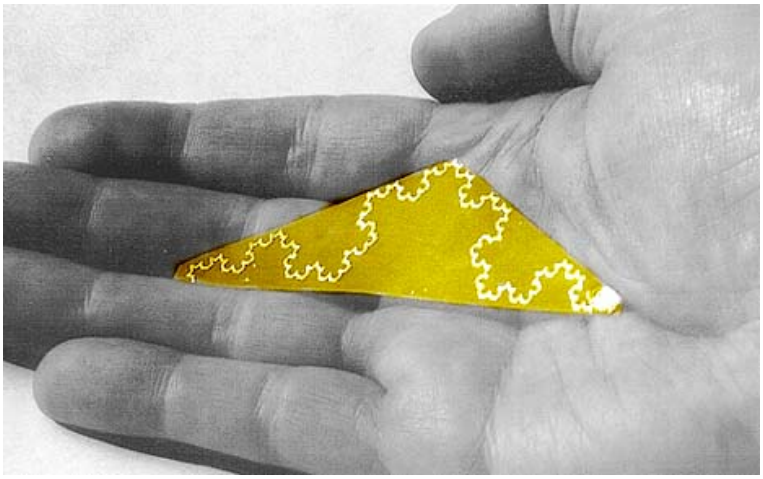


Obrázek 4.3: Rekonstrukce obrázku Lenna

útvarů, dosahují vizuálně kvalitních výstupů.

4.3 Fraktální antény

Poznatků z fraktální geometrie se využívá při konstrukci nejrůznějších anténních systémů (mobilní telefony, vysílačky, vesmírné sondy atd.). Při projektování antén se využívá soběpodobnosti k maximalizaci délky či obvodu materiálu, který je schopen vysílat nebo přijímat elektromagnetický signál v rámci vymezené plochy či objemu. Z důvodu několikerého opakování tvarů jsou fraktální antény velmi kompaktní, použitelné pro mnoho frekvencí a využívají se např. v mikrovlné komunikaci [22]. Od typu použití antény se odvíjí její tvar. Anténa například může vypadat jako část Kochovy křivky (obrázek 4.4), Sierpinského trojúhelník atd.



Obrázek 4.4: Fraktální anténa

4.4 Materiály

Fraktální struktura materiálu byla objevena při změnách skupenství látky, jako je tání ledu či vypařování vody. Z makroskopického hlediska existují tři základní látková skupenství (pevné, kapalné, plynné), v některých případech se za čtvrté skupenství považuje plazma. Z mikroskopického hlediska je ovšem těchto stavů mnohem více. Fraktální strukturu hmoty potom můžeme vidět při přechodu hmoty z jednoho stavu do druhého. Tento úkaz lze pozorovat např. při přechodu magnetického materiálu na materiál nemagnetický. V mikroskopickém měřítku je složen z elementárních magnetů, které v závislosti na teplotě určují, zda se celek bude chovat jako magnet. Za nízkých teplot je materiál magnetický, avšak vyskytují se v něm mikroskopické oblasti, kde elementární magnety uspořádány nejsou. Při kritické teplotě (*Curierova teplota*), kdy je materiál v bodě přechodu mezi stavy, vypadá jeho struktura ve všech měřítcích stejně a má fraktální strukturu [24].

Poznatky z fraktální geometrie nacházejí uplatnění i ve vojenské technice při maskování vozidel, letadel (obrázek 4.5), lodí atd. Fraktální vzory se objevují také na vojenských oděvech a zbraních. Další oblastí, kdy je vhodné objekt “ukrýt” před zraky okolí, jsou stavby v krajině. Po zabarvení fraktálním vzorem se sníží vizuální narušení okolní krajiny.

Obrázek 4.5: Fraktální kamufláž F16

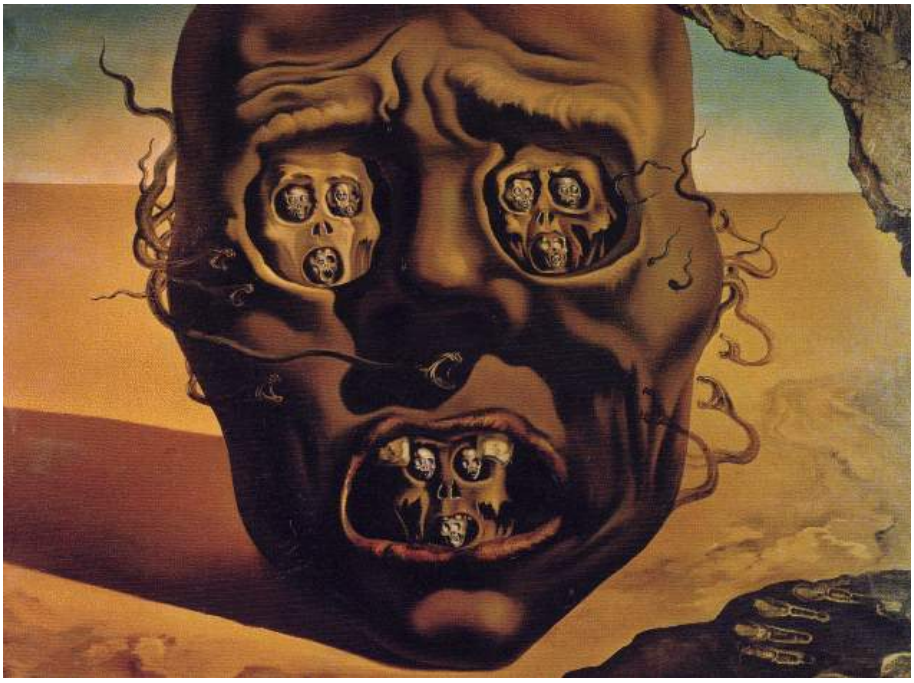
4.5 Umění a architektura

Mnoho lidí, Mandelbrota nevyjímaje, zaujala krása fraktálů natolik, že si vytvářeli fraktály pouze pro jejich vizuální zajímavost. Rozmach tohoto hledání nových podob fraktálů nastal s érou osobních počítačů. Vzniklo mnoho programů pro generování fraktálních obrazců, na kterých pracovali ať už jednotlivci nebo skupiny lidí z celého světa.

Přestože je fraktální umění spojováno s počítači, lze fraktální vzory pozorovat ve stavbách a obrazech mnoha autorů. Nejstarší známky fraktálních principů je možno pozorovat již v antice. Pravidlo krásy, tzv. zlatý řez, dělí délku na dvě části tak, že poměr celé délky k větší části je stejný jako poměr větší části k menší. Ve zlatém řezu lze spatřovat faktor soběpodobnosti právě kvůli zmíněnému poměru. Dalším příkladem z pozdější doby jsou podlahy italských bazilik, viz obrázek 4.6, kdy se v mozaikách objevují vzory podobné druhé či třetí iteraci Sierpinského trojúhelníku [4].

Obrázek 4.6: Santa Prassede - podlaha

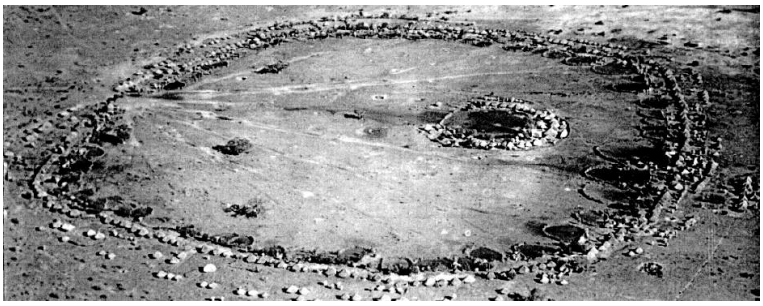
Příkladem využití fraktálního principu v moderním umění je obraz Salvadora Dalího "Tvář války" (*Visage de la guerre*). Opakující se motiv tvoří obličej a jeho zmenšeniny umístěné do očních důlků a úst, jak lze vidět na obrázku 4.7. Dále se s fraktály lze setkat v moderní architektuře při pohledu na organizaci měst či uspořádání staveb.



Obrázek 4.7: Salvador Dalí - Tvář války

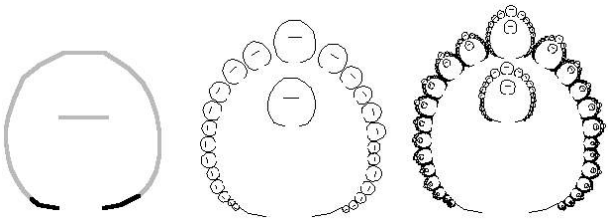
4.5.1 Africké fraktály

Příkladem, kdy byla objevena fraktální podstata dávno před definováním pojmu fraktál, jsou tzv. Africké fraktály [6]. Na obrázku 4.8 je letecký pohled na vesnici Ba-ila z období před rokem 1944. Tvar vesnice odpovídá fraktálu, jehož iterace jsou na obrázku 4.9.



Obrázek 4.8: Africká vesnice Ba-ila

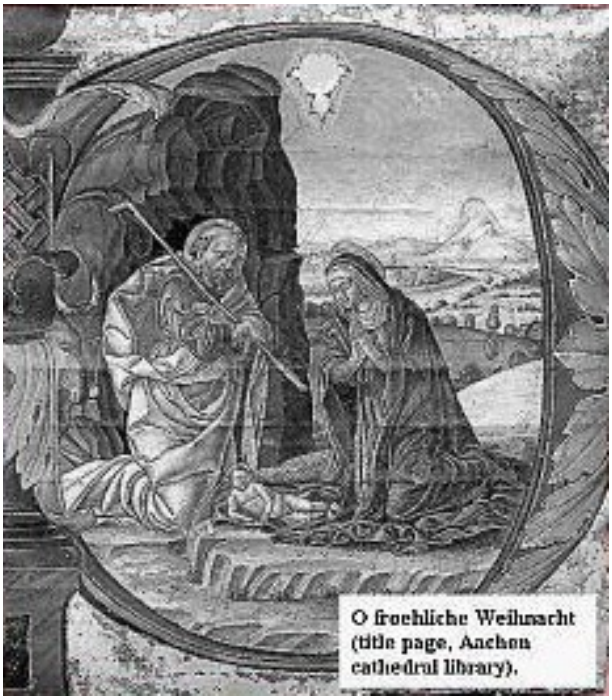
Dalšími oblastmi, kde byl objeven vliv fraktálů v Africe, jsou šperky a úprava vlasů. Fraktální vlivy je možno nalézt také v Egyptské kultuře.



Obrázek 4.9: Fraktál vesnice Ba-ila

4.5.2 Betlémská hvězda

Betlémská hvězdě je v oblasti fraktální geometrie raritou. Většina zmínek o ní odkazuje na publikaci Rona Eglashe [8], který zkoumal práci mnicha Uda z Aachenu.



Obrázek 4.10: Ilustrace z 13. století

Zvláštního vyobrazení Betlémské hvězdy si povšiml profesor matematiky Bob Schipke. Hvězda vypadala jako znázornění Mandelbrotovy množiny, jedné z ikon moderní doby počítačů. Její složitost se dá vyobrazit pouze pomocí počítačů, což se však díky kresbě Hvězdy ukázalo jako mylná doměnka. Více o této pozoruhodnosti uvádí Ray Girvan na svých stránkách [8]. V pracech Uda z Aachenu později vědci nalezli ještě několik zajímavostí.

Nejdůležitejší na článku o Udovi je však datum uvedení - 1. dubna 1999. Ray Girvan připravil aprílový žertík s takovou pečlivostí, že po přečtení mělo mnoho odborníků zájem o cestu do Aachenu, aby mohli spis středověkého mnicha studovat. Girvan docílil věrohodnosti práce uvedením málo známých informací a odkazy na mnoho seriózních prací, jež jsou v pořádku až na neexistující počin fiktivního profesora Boba Schipkeho.

4.6 SW pro generování fraktálů

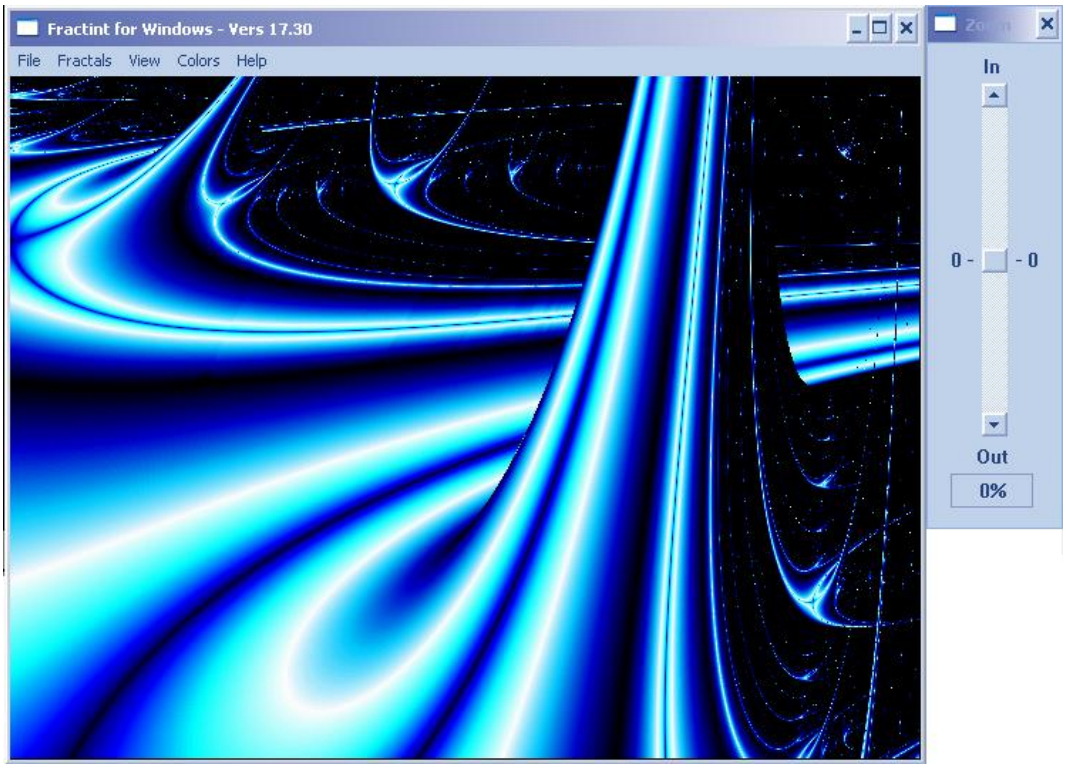
Na internetu je k dispozici velké množství programů, jejichž pomocí lze fraktály generovat. V následujících kapitolách je popsáno několik z nich, které byly vybrány na základě významu, kvalitním výstupům, či široké možnosti voleb ovlivňující výsledný obrazec. Nejčastěji se jedná o programy volně šiřitelné.

4.6.1 FractInt

FractInt je volně šiřitelný program vytvářený skupinou Stone Group, která má v současnosti šest členů podílejících se na dalším vývoji programu. Původním autorem je Bert Tyler. Jméno programu souvisí s tím, že velké množství jím generovaných obrázků využívá celočíselnou (*INTeger*) matematiku, namísto reálných čísel běžných u jiných programů.

První verze FractIntu vydaná v září roku 1988 byla napsána pro procesory 386, používala celočíselný algoritmus a nepotřebovala matematický koprocesor pro matematické operace. Na počátku byl program rychle vyvíjen a o rok později již byla k dispozici verze 9.3. V současnosti (květen 2007) je nejnovější FracInt dostupný ve verzi 20.0.

FracInt je ovládán především z příkazové řádky, umí však pracovat i s dávkovými soubory. Nejnovější grafická nadstavba WinFract pro Windows je ve verzi 18.21. Starší verzi GUI s vykresleným Ljapunovým fraktálem lze vidět na obrázek 4.11. Na ovládání programu myší při pohybu (zoomování) ve vykreslených fraktálech je třeba si zvykat z důvodu poněkud nestandardních reakcí oblasti výběru. Usnadněním je tzv. "Zoom bar" kterým lze lépe vybrat oblast pro zvětšení.



Obrázek 4.11: GUI programu WinFract

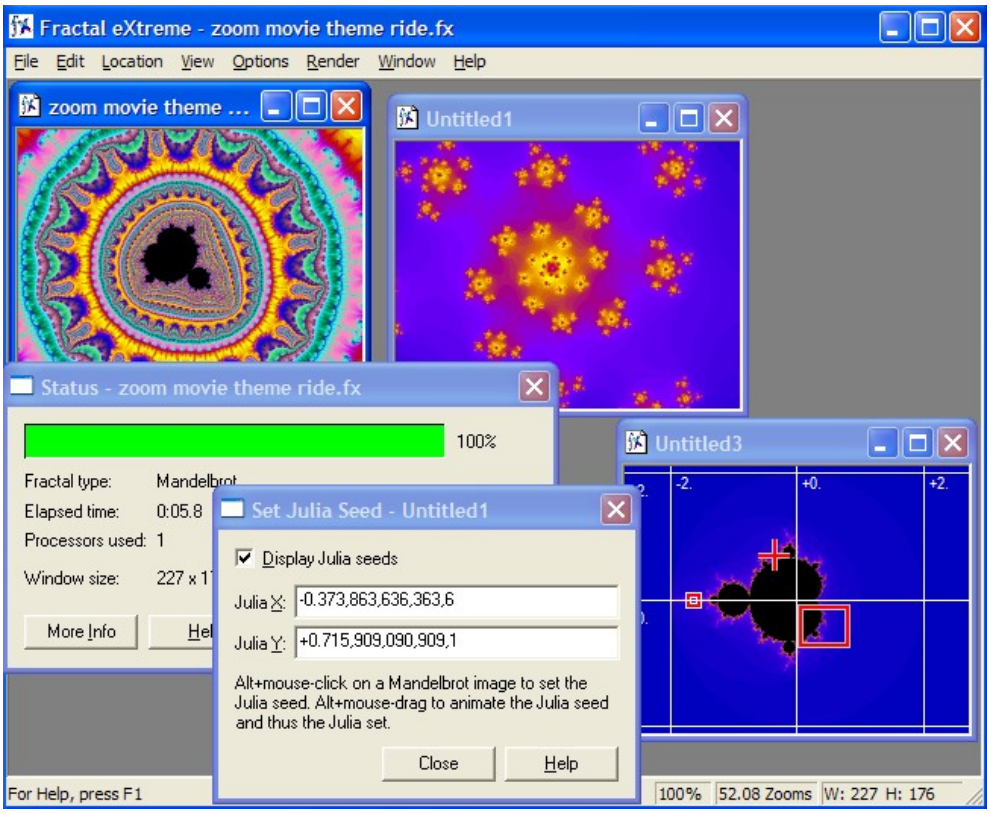
Nabídka “Fractals” obsahuje další položky s výběrem druhu vykreslovaného fraktálu, nastavení základních a rozšířených parametrů zobrazovací oblasti atd. Každý druh fraktálu má přednastaveny základní parametry zaručující vykreslení alespoň základního tvaru na obrazovku.

FracInt umí pracovat s velkým množstvím adaptérů a grafických režimů. Nejvyšší nabízené rozlišení je 1024*768, avšak je možno jej nastavit i ručně. Programem lze vygenerovat i obrázky s větším rozlišením, než má obrazovka. Tyto se pak ukládají do paměti nebo na disk. Počet barev je stále omezen na 256, což může být v dnešní době nevýhoda při vytváření dekorativních obrázků.

Veškeré informace o autorech a tutorialy ukazující práci s programem lze nalézt na domovských stránkách `http://spanky.triumf.ca/www/fractint/fractint.html`.

4.6.2 Fractal eXtreme

Fractal eXtreme je komerční software vyvíjený společností Cygnus Software založenou v roce 1985. Prvními produktem společnosti byl v roce 1986 program MandFXP. V roce 1997 pak byl uveden na trh Fractal eXtreme, který je naprogramován v C++ v kombinaci s assemblerem. Poslední aktualizace programu byla v roce 2003, kdy vyšla verze 1.902. Na webových stránkách výrobce `www.cygnus-software.com` je k dispozici ke stažení patnácti denní zkušební verze programu.



Obrázek 4.12: Fractal eXtreme

Fractal eXtreme nabízí náhled vybraného fraktálu ve snížené kvalitě před jeho finálním vykreslením, aby se mohl uživatel rozhodnou zda daný obrázek splňuje jeho požadavky a

chce jej zobrazit nebo uložit v plné kvalitě.

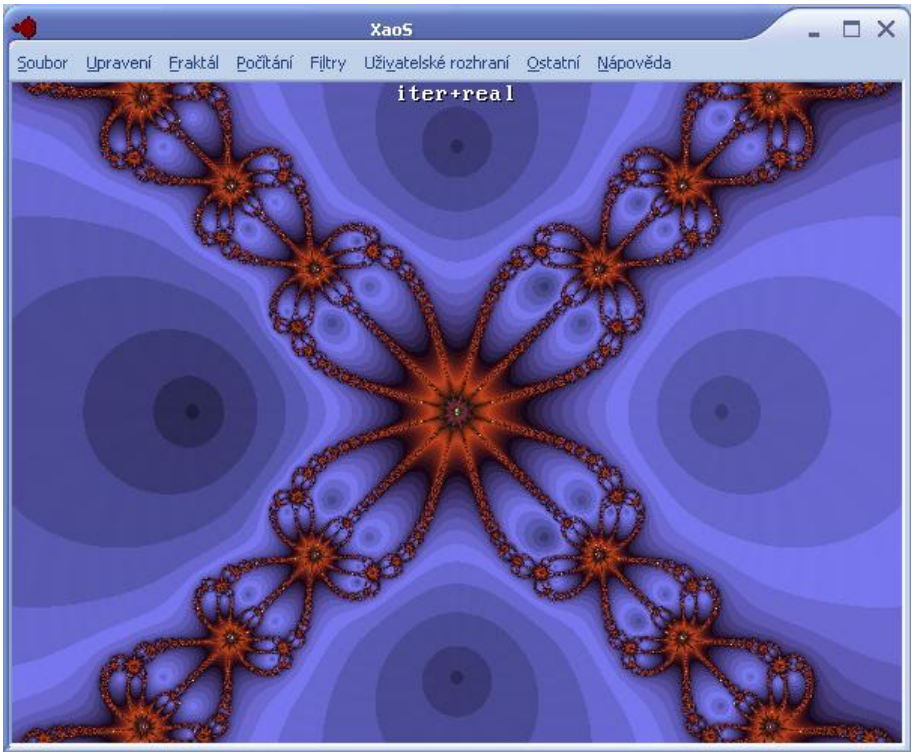
V okně programu, obrázek 4.12, je možno zobrazovat více fraktálů najednou. Zajímavou vlastností je, že v případě Mandelbrotovy množiny lze zobrazit jednotlivé Juliovy množiny odpovídající dané oblasti.

Další zajímavou možností je uložení animovaných průletů fraktálem, kdy si lze ve vykresleném vždy definovat oblast, která bude tvořit další přiblížení fraktálu. Program pak propočítá jednotlivé "snímky" výsledné animace. Pro uložení animací Fractal eXtreme používá vlastní formát, jež lze přehrát v programu *Zoom Movie Player*. Tento je zdarma a umí převést videa do avi sekvencí. Několik vzorových "filmů" je umístěno na webových stránkách programu.

4.6.3 XaoS

XaoS je program určený ke generování animací fraktálů v reálném čase. Původním autorem je Jan Hubička, ke kterému se přidali další vývojáři. První verze programu vznikla v roce 1998. V současnosti je nejaktuálnější verze 3.2.3.

Výhodou XaoSe je fakt, že je napsán nejen pro Windows, ale i další operační systémy jako je MacOS X, Linux a BSD, OS/2 aj. Program nabízí volbu několika jazykových verzích - anglické, české, španělské, maďarské atd.



Obrázek 4.13: XaoS

Generování fraktálů je rychlé i ve vysokém rozlišení. V programu je předvoleno vykreslení několika nejznámějších fraktálů (např. Mandelbrotova množina, fraktál Newton atd.). Vybarvení útvaru lze měnit zvlášť pro vnitřní a vnější oblast, k dispozici je několik variant. Okno XaoSu s Newtonovou fraktální množinou čtvrtého řádu je na obrázku 4.13. Zajímavou

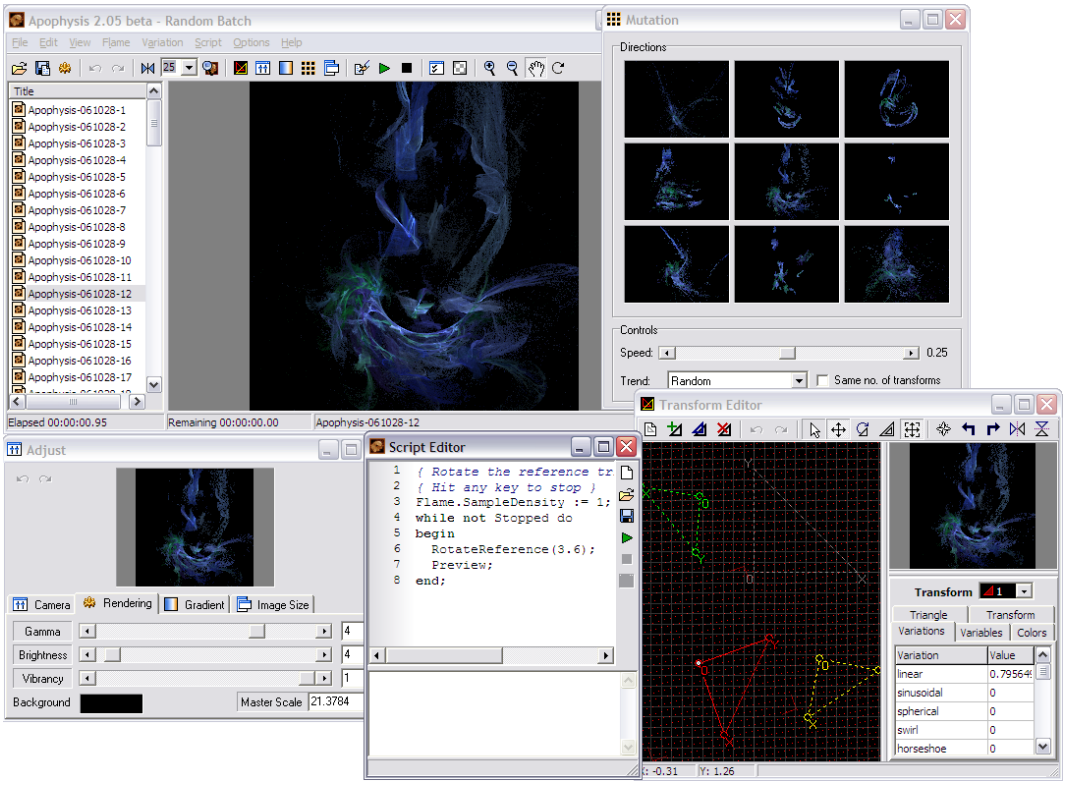
funkcí je “autopilot”, kdy program sám určí směr průletu fraktálem.

Na domovské stránce programu `http://wmi.math.u-szeged.hu/xaos/doku.php` lze nalézt informace o programu i galerie obrázků. Součástí archivu s programem jsou vícejazyčné tutorialy popisující nejen ovládání XaoSu, ale obsahují také informace o fraktálech a fraktální geometrii.

4.6.4 Apophysis

Apophysis je open source editor fraktálů typu Fractal Flame. Program vytvořil Mark Townsend pro platformu MS Windows a v současné době se na vývoji podílí několik dalších programátorů. Poslední dostupná verze programu je 2.02.

Program obsahuje spoustu možností pro vytváření a úpravu fraklálů. Součástí je i editor, ve kterém lze přímo měnit transformace, ze kterých se flame skládá. Okno mutací obsahuje zobrazení několika stádií výpočtu daného fraktálu. Mezi jednotlivými verzemi lze vybrat vizuálně nejpřijatelnější a nechat ji vygenerovat ve větším rozlišení. V nastavení lze zvolit barevnou paletu a pozici obrázku, případně přímo napsat skript který bude aplikován na fraktál.



Obrázek 4.14: Apophysis

Vytváření fraktálů v Apophysisu je převážně založeno na náhodě. Při každém spuštění se vygenerují nové předlohy, na které se náhodným způsobem aplikuje nějaká variace. Výběr variace lze ručně měnit, stejně jako definici předlohy, a získat tak pokaždé nový fraktál.

Na domovských stránkách `www.apophysis.org` je možno nalézt rozsáhlou sekci s tutorialy pro začátečníky i pokročilé uživatele.

Kapitola 5

Algoritmy pro generování fraktálů

V následující části diplomové práce jsou uvedeny algoritmy, jimiž lze generovat jednotlivé typy fraktálů. Nejkomplexnější informace a příklady zdrojových kódů uvádí Pavel Tišnovsky ve svém seriálu o fraktálech zveřejňovaného na stránkách Root.cz [25]. S ohledem na aktuálnost dat použitých pro tuto práci, jsou odtud převzaty úseky zdrojových kódů. Většina informací a kódů je průběžně upřesňována díky připomínkám dalších autorů.

5.1 Algoritmy dynamických systémů

5.1.1 Lorenzův atraktor

Popis Lorenzova atraktoru je uveden v kapitole 2.4.1 a je charakterizován třemi rovnicemi (2.7, 2.8, 2.9). Pro iterativní výpočet je třeba provést jejich úpravu, aby bylo možno vypočítat pozice bodů $P_s = [x_n, y_n, z_n]$ v prostoru [25, 3]:

$$x_{n+1} = x_n + (-a \times x_n \times dt) + (a \times y_n \times dt) \tag{5.1}$$

$$y_{n+1} = y_n + (b \times x_n \times dt) - (y_n \times dt) - (z_n \times x_n \times dt) \tag{5.2}$$

$$z_{n+1} = z_n + (-c \times z_n \times dt) + (x_n \times y_n \times dt) \tag{5.3}$$

Lorenzův atraktor je narozdíl od mnoha fraktálů vykreslován v trojrozměrném prostoru. Za pomoci OpenGL vypadá funkce realizující výpočet bodů následovně:

```
//------------------------------------------------------------
// Překreslení Lorenzova atraktoru (původní Lorenzův atraktor)
//------------------------------------------------------------
void recalcLorenz(double a,       // parametry fraktálu
                  double b,
                  double c,
                  double dt,      // použito při numerické integraci
                  int    maxiter, // maximální počet iterací
                  double scale)   // měřítko obrazce
{
  double x=0.1, y=0, z=0;           // pozice bodu v prostoru
  double x2, y2, z2;
  int iter=maxiter;
```

```
  glColor3f(1.0f, 1.0f, 1.0f);
  glBegin(GL_LINE_STRIP);           // začátek vykreslování polyčáry
  while (iter--) {                  // iterační smyčka
      glColor3f(x/10.0,y/10.0,z/10.0);
      x2 = x+(-a*x*dt)+(a*y*dt);
      y2 = y+(b*x*dt)-(y*dt)-(z*x*dt);
      z2 = z+(-c*z*dt)+(x*y*dt);
      x=x2;                         // přepis nových souřadnic
      y=y2;
      z=z2;
      glVertex3d(x*scale, y*scale, z*scale);  // vykreslení úsečky
  }
  glEnd();                          // konec vykreslování polyčáry
}
```

Uvedená verze vykreslí původní Lorenzův atraktor. Existují i další modifikace algoritmu, které lze nalézt v [25]. Jsou zde uvedeny kompletní kódy původního i modifikovaných programů.

5.1.2 Juliovy množiny

Seznámení s Juliovými množinami je v kapitole 3.1.1. Tento typ fraktálů je generován iteracemi funkce komplexní paraboly $z_{n+1} = z_n^2 + c$, kde z_n a c jsou komplexní proměnné. Iterační proces začíná startovní hodnotou z_0.

Realizace algoritmu spočívá ve dvou vnějších cyklech, které generují počáteční souřadnice bodů z_0, a vnitřní smyčce, kde jsou tyto body iterovány. Iterace probíhá postupným počítáním z_1 a z_2, než hodnota některé z nich překročí hodnotu 2, což je mez divergence. Pokud není rozhodnuto o divergenci posloupnosti do dosažení nastaveného počtu iteračních kroků, je posloupnost považována za konvergující. Podle počtu kroků je obarven pixel odpovídající souřadnicím počátečního bodu z_0 [12, 11, 25].

První příklad obsahuje cyklus výpočtu Juliovy množiny podle klasického postupu.

```
zy0=MINY;
for (y=0; y<pix->height; y++) {        // pro všechny řádky v pixmapě
  zx0=MINX;
  for (x=0; x<pix->width; x++) {       // pro všechny pixely na řádku
    zx=zx0; zy=zy0;                    // nastavit počáteční hodnotu Z(0)
    for (iter=0; iter<maxiter; iter++) {// iterační smyčka
      if (zx*zx+zy*zy>4.0) break;     // kontrola překročení meze divergence
      zxx=zx*zx-zy*zy+cx;              // výpočet Z(n+1)=Z(n)^2+C
      zy=2.0*zx*zy+cy;
      zx=zxx;
    }
    putpixel(pix, x, y, iter, iter, iter); // vykreslení pixelu
    zx0+=(MAXX-MINX)/pix->width;        // posun na další bod na řádku
  }
  zy0+=(MAXY-MINY)/pix->height;         // posun na další řádek
}
```

V druhém příkladu je provedena optimalizace počtu operací ve vnější smyčce dvěma pomocnými proměnnými zx2 a zy2, které obsahují mocninu reálné (zx) a imaginární (zy) složky proměnné z [25]. Z praxe vyplynulo, že oproti první uvedené metodě se na moderních CPU nemusí jednat o rychlejší metodu, z důvodu odlišné optimalizace provádění instrukcí.

```
zy0=MINY;
for (y=0; y<pix->height; y++) {      // pro všechny řádky v pixmapě
  zx0=MINX;
  for (x=0; x<pix->width; x++) {     // pro všechny pixely na řádku
    zx=zx0; zy=zy0;                  // nastavit počáteční hodnotu Z(0)
    for (iter=0; iter<maxiter; iter++) {// iterační smyčka
      zx2=zx*zx; zy2=zy*zy;          // zkrácený výpočet mocniny složek Z
      if (zx2+zy2>4.0) break;        // kontrola překročení meze divergence
      zy=2.0*zx*zy+cy;               // výpočet Z(n+1)
      zx=zx2-zy2+cx;
    }
    putpixel(pix, x, y, iter, iter, iter); // vykreslení pixelu
    zx0+=(MAXX-MINX)/pix->width;     // posun na další bod na řádku
  }
  zy0+=(MAXY-MINY)/pix->height;      // posun na další řádek
}
```

Více příkladů lze nalézt na [25] a detailnější popis všech druhů Juliových množin pak obsahují např. texty [3, 12, 2].

5.1.3 Mandelbrotova množina

Abychom mohli vykrestlit Mandelbrotovu množinu, je třeba zjistit, které body do množiny nepatří. Z důkazu demonstrovaného v [25] a na základě rovnice definující množinu vyplývá, že body splňující podmínku $|z| > 2$ do Mandelbrotovy množiny nenáleží. Posloupnost z_n pro tyto body diverguje.

Generování Mandelbrotovy množiny spočívá v testování, zda absolutní hodnota z nepřekročí hodnotu 2. Pokud se tak stane, bod v množině neleží. Jestliže proběhne výpočet všech iterací a absolutní hodnota z hodnotu 2 nepřesáhne, je bod považován za prvek Mandelbrotovy množiny. Přesnost výpočtu lze ovlivnit počtem iterací [25, 12, 22].

Slovní popis algoritmu výpočtu bodů náležejících do Mandelbrotovy množiny, se vstupními hodnotami c (komplexní konstanta) a MaxIter (maximální počet iterací), lze zapsat následovně:

1. Nastav iterační krok iter = 0.
2. Nastav z = 0.
3. Jestliže je počet iteračních kroků menší než MaxIter opakuj:
   - Nastav z = z2+c,
   - jestliže $|z| > 2$ bod neleží v Mandebrotově množině, konec cyklu.
   - nastav iter = iter+1.
4. Bod leží uvnitř Mandelbrotovy množiny.

Pro vlastní implementaci algoritmu je třeba upravit parametry výpočtu, kvůli absenci datového typu komplexního čísla ve většině jazyků a pomalému výpočtu odmocniny. Testovací podmínka divergence pak má tvar $zx * zx + zy * zy > 4$.

Příklad algoritmu na výpočet bodů Mandelbrotovy množiny může vypadat následovně [25]:

```
int MandelbrotTest(double cx, double cy)
{
  double      zx,zy,zx2,zy2,cx,cy;  // komplexní proměnná Z
  int         iter;                 // počet iterací

  zx=0;                             // vynulovat komplexní proměnnou C
  zy=0;

  iter=0;                           // nastavit počitadlo iterací

  do {                              // iterační smyčka
     zx2=zx*zx;                     // zx^2
     zy2=zy*zy;                     // zy^2
     zy=2.0*zx*zy+cy;
     zx=zx2-zy2+cx;                 // z:=z^2+c
     iter++;                        // zvýšit hodnotu počitadla iterací
  }while (iter<maxiter && (zx2+zy2)<4.0);// test na počet iterací a bailout

  if (iter==maxiter)                // bod leží uvnitř Mandelbrotovy množiny
     return LEZI_UVNITR;
  else                              // bod leží vně Mandelbrotovy množiny
     return LEZI_VNE;
}
```

Další možnosti algoritmů generujících fraktální obrazce spolu s postupy pro obarvení lze nalézt např. v [22, 25, 24]. Zajímavým fraktálem je také např. Newtonova fraktální množina, avšak z kapacitních důvodů je nutno odkázat na zdroje uvedené v literatuře.

5.2 Algoritmy L-systémů

Algoritmy L-systémů jsou často založeny na tzv. želví grafice. Pravidla pro generování jsou většinou pevně daná, využitím prvku náhody dostáváme Stochastické L-systémy. Želva je ovládána příkazy pro posun a natočení. Symboly, kterými je ovládána jsou uvedeny v tabulce 3.1 a v případě potřeby lze tabulku doplnit o další akce.

5.2.1 Kochova křivka

Informace o Kochově vločce jsou uvedeny v kapitole 3.2 spolu s přepisovacími pravidly gramatiky, která demonstrují postup vytváření křivky. Při vytváření algoritmem je nutno nastavit krok želvy tak, aby se vykreslený obrazec vlezl do zobrazované oblasti. Při znalosti počtu iterací zjistíme kolik kroků želva provede v horizontálním směru, na základě čehož vypočteme krok želvy [25]:

```
step = WINDOW_WIDTH/(pow(3, rulesCount))
```

WINDOW_WIDTH určuje šířku okna a *rulesCount* udává počet přepisů symbolu F. Algoritmus aplikace přepisovacích pravidel, pak vypadá následovně:

```
void applyRule(void)
{
  int i, j, k;                      // pocitadla smycek a indexy znaku
  char src[MAX_LENGTH];             // zdrojovy retezec
  char dest[MAX_LENGTH];            // cilovy retezec
  int fcount=0;                     // pocet prepisu
  strcpy(src, ret);
  puts(src);                        // kontrolni vypis pred prepisem
  for (i=0, j=0; src[i]; i++) {     // projit celym retezcem
    if (src[i]=='F') {              // tento symbol se ma prepsat
      for (k=0; prepstr[k]; k++)    // provest prepis
        dest[j++]=prepstr[k];
    }
    else {                          // ostatni symboly kopirovat
      dest[j]=src[i];
      j++;
    }
  }
  dest[j]=0;

  for (j=0; dest[j]; j++)
    if (dest[j]=='F') fcount++;
  puts(dest);                       // kontrolni vypis po prepisu
  printf(``fcount=%d\n'', fcount);
  strcpy(ret, dest);
}
```

Pro generování např. Sierpinského trojúhelníku, Hilbertovy křivky nebo Dračí křivky se dá využít paralelních přepisovacích algoritmů [12, 25], kdy nezávisí na pořadí aplikace přepisovacích pravidel a tyto se projeví vždy až v následujícím iteračním kroku. Algoritmu

jsou uvedeny v seriálu o fraktálech na Root.cz, stejně jako další typy technik zabývajících se vytvářením L-systémů.

5.3 Algoritmy IFS

Jak bylo uvedeno v kapitole 3.3, pro generování IFS fraktálů se využívají transformace. Nejvhodnější způsob jejich zápisu jsou *transformační matice*, případně *vektory posunu*, které při programování ušetří paměť a urychlí výpočty.

Transformace je v ploše E^2 popsána maticí $\mathbf{U}$ o velikosti 2×2 a vektorem posunu $\mathbf{V}$ s dvěma prvky [25, 24]. Analogicky pro prostor E^3 bude matice rozměru 3×3 a vektor bude mít tři prvky. Koeficienty matice $\mathbf{U}$ se uplatňují při aplikacích transformací rotace, zkosení, změny měřítka. Koeficienty vektoru $\mathbf{V}$ se používají k posunu.

Pro vlastní implementaci je však využíváno *transformační matice* $\mathbf{T}$, která je sloučením matice $\mathbf{U}$ a vektoru $\mathbf{V}$. V rovině bude mít $\mathbf{T}$ rozměr 3×3 a v prostoru 4×4.

Množinu zobrazení $\Phi = \{\phi_1, \ldots, \phi_n\}$ (viz kapitola 3.3, [2, 25]) doplňuje množina pravděpodobnostní $\Pi = \{\pi_1, \ldots, \pi_n\}$, $\pi_i > 0$, která určuje pravděpodobnost použití jednotlivých transformací při vytváření fraktálu. Součet jednotlivých pravděpodobností musí být roven jedné. Dále je možné říci, že čím větší pravděpodobnost transformace ϕ_i, tím větší musí být míra její kontrakce. Pokud vytvářený objekt nemá kontrahující transformaci, nejedná se o IFS (viz [3, 2, 25, 24]) Systém iterovaných funkcí je tedy určen vztahem 5.4.

$$IFS = (\Phi, \Pi) = (\{\phi_1, \ldots, \phi_n\}, \{\pi_1, \ldots, \pi_n\}) \tag{5.4}$$

5.3.1 Algoritmus náhodné procházky (RWA)

Algoritmus náhodné procházky je jedním z nejjednodušších algoritmů pro generování IFS fraktálních objektů. Jeho výhodou je jednoduchost a malé nároky na paměť. Nevýhodou je nutnost generování velkého počtu bodů pro zobrazení fraktálu, z čehož výplývá možná duplicita vykreslování některých částí. V opačném případě při nízkém počtu iterací mohou některé částí obrazce chybět [3, 25, 24].

Princip generování IFS pomocí algoritmu RWA je následující:

1. Zvolíme náhodný bod P_0 v rovině/prostoru.
2. Výběr transformace ϕ_i z množiny Φ na základě zvolené hodnoty pravděpodobnosti π_i z množiny Π. Zaručí se tím dodržení, počet použití transformace na základě pravděpodobnosti.
3. Do dosažení zadaného počtu iterací opakujeme:
   (a) aplikace vybrané transformace na bod - změna souřadnic,
   (b) výběr další transformace.

Bod P_0 nemusí ležet v atraktoru, proto je vhodné několik prvních iterací nezobrazovat. Vhodě zvolený počet iterací ovlivňuje kvalitu výstupu (duplicita/absence bodů). Zkrácený ukázkový algoritmus *RWA* pro vygenerování IFS koláže je převzat z Root.cz [25]. Uvedený příklad vygeneruje kapradinu podobnou obrázku 3.9. Transformační matice spolu s vektorem pravděpodobnosti transformací pro takovýto IFS systém je:

```
    float  t[4][2][3];                  // transformační matice
    float  p[4];                        // vektor pravděpodobnosti transf

// transformační matice
    t[0][0][0]= 0.000;   t[0][0][1]= 0.000;   t[0][0][2]= 0.000;
    t[0][1][0]= 0.000;   t[0][1][1]= 0.160;   t[0][1][2]= 0.000;

    t[1][0][0]= 0.200;   t[1][0][1]=-0.260;   t[1][0][2]= 0.000;
    t[1][1][0]= 0.230;   t[1][1][1]= 0.220;   t[1][1][2]= 1.600;

    t[2][0][0]=-0.150;   t[2][0][1]= 0.280;   t[2][0][2]= 0.000;
    t[2][1][0]= 0.260;   t[2][1][1]= 0.240;   t[2][1][2]= 0.440;

    t[3][0][0]= 0.850;   t[3][0][1]= 0.040;   t[3][0][2]= 0.000;
    t[3][1][0]=-0.040;   t[3][1][1]= 0.850;   t[3][1][2]= 1.600;

// vektor pravděpodobnosti transformace
    p[0]=0.01;
    p[1]=0.07;
    p[2]=0.08;
    p[3]=0.84;
```

Vlastní iterační cyklus pak vypadá následovně:

```
 while (iter++<maxiter*100) {          // iterační smyčka
     pp=(float) random (255) / 255.0;  // p leží v rozsahu 0.0-1.0
     // proměnná cyklu vyhledání transformace
     k=0;
     sum=0;
     while (sum<=pp) {              // podle hodnoty náhodného čísla
         sum=sum+p[k];              // vybrat transformaci
         k++;
     }                              // k určuje index transformace
     k--;                           // úprava hodnoty k ze smyčky while
     hlp=t[k][0][0]*x+t[k][0][1]*y+t[k][0][2]; // provedení vybrané
     y  =t[k][1][0]*x+t[k][1][1]*y+t[k][1][2]; // transformace
     x  =hlp;
     if (iter > threshold)          // je-li dosaženo hranice iterací
         // vykreslení pixelu
         putpixel(pix, x*scale+xpos, y*scale+ypos, 0xff, 0xff, 0xff);
 }
```

Algoritmus RWA nachází využití v ekonomice v modelech akciových trhů, v populační genetice, kde statisticky popisuje genetický směr. Ve fyzice je používán ke zjednodušení modelu Brownova pohybu a v teorii kvantového pole. Dalšími oblastmi použití RWA jsou kombinatorika, biologie (výzkum mozku, pohyb očí), psychologie atd. Některé ukázky lze nalézt např. na wikipedia.org (Random Walk) [23]. Většinou se jedná o ekonomické zaměření.

5.3.2 Deterministický algoritmus (DIA)

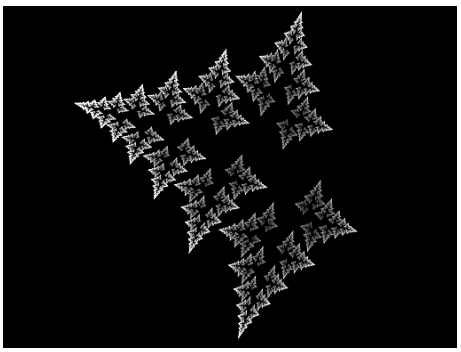
Druhým uvedeným algoritmem pro výpočet IFS fraktálů je deterministický algoritmus (*DIA - Deterministic Algorithm*). Od předchozího *RWA* se liší přístupem ke generování bodů.

RWA v každé iteraci vypočítá vždy jeden bod, zatímco DIA vyčíslí bodů více [3, 25]. Není nutno počítat ani s pravděpodobností jednotlivých transformací, protože na množinu bodů se vždy aplikují všechny transformace z daného IFS. RWA používá pro výpočet náhodná čísla a průběh generování je stochastický, zatímco DIA použitím všech transformací současně nemusí žádnou z nich vybírat.

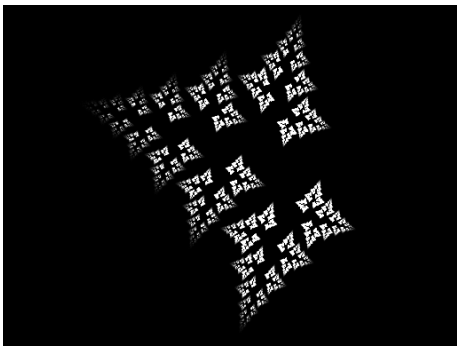
Postup generování IFS pomocí algoritmu DIA, jež se obvykle realizuje dvěmi vnořenými cykly, je:

1. zvolí se několik bodů v prostoru,
2. do dosažení zadaného počtu iterací opakujeme:
   (a) aplikují se všechny transformace tvořící IFS systém,
   (b) všechny vzniklé body jsou množinou bodů pro další iteraci.

IFS vzniklé generováním algoritmy RWA a DIA můžeme porovnat na obrázcích 5.1 respektive 5.2. Je patrné, že se výsledné fraktály liší.



Obrázek 5.1: IFS vytvořený RWA



Obrázek 5.2: IFS vytvořený DIA

Následující část zdrojového kódu znázorňuje implementaci algoritmu DIA.

```
//---------------------------------------------------------------
// Překreslení IFS pomocí rekurzivního algoritmu DIA
//---------------------------------------------------------------
void recalcIFSusingDIA(float x, float y, int depth)
{
  float  xn, yn;                      // poloha iterovaného bodu
  int    k;                           // číslo transformace

  if (depth) {
    // pro všechny transformace v~IFS
    for (k=0; k<transf; k++) {
      xn=(*t)[k][0]*x+(*t)[k][1]*y+(*t)[k][4];// provedení
      yn=(*t)[k][2]*x+(*t)[k][3]*y+(*t)[k][5];// transformace
      iter++;
      if (iter>threshold)         // je-li dosaženo hranice iterací
        switch (drawType) {     // výběr vykreslovacího režimu
          case IterPutPixel:
             putpixel(pixIFS, x*scale+xpos, y*scale+ypos, rf ?
                      0xff:0x00, gf ? 0xff:0x00, bf ? 0xff:0x00);
             break;
          case IterAddPixel:
             addpixel(pixIFS, x*scale+xpos, y*scale+ypos, red,
                      green, blue);
             break;
          case TransfPutPixel:
             putpixel(pixIFS, x*scale+xpos, y*scale+ypos,
                      pal[k&7][0], pal[k&7][1], pal[k&7][2]);
             break;
          case TransfAddPixel:
             addpixel(pixIFS, x*scale+xpos, y*scale+ypos,
                     (!!pal[k&7][0])*deltar, (!!pal[k&7][1])*deltag,
                     (!!pal[k&7][2])*deltab);
             break;
          default:
             break;
      } // konec switche
      // rekurze - výpočet dalších ‘‘transf’’ pixelů
      recalcIFSusingDIA(xn, yn, depth-1);
    } // konec for cyklu
  } // konec if (depth)
} // konec těla fuknce
```

Dalšími algoritmy generující IFS, je modifikovaný algoritmus náhodné procházky *M-RWA*, který spojuje výhody RWA a DIA, a algoritmus generování minima pixelů *MPA (Minimal Plotting Algorithm).*

5.3.3 Modifikovaný algoritmus náhodné procházky (M-RWA)

Modifikovaný algoritmus náhodné procházky *M-RWA* má pozměněn fuknění princip a pracuje tak, že na bod se v každé iteraci aplikují všechny transformace, ale pouze některé body (alespoň 1) jsou použity pro další iterace [25]. Výhodou této úpravy je přibližně stejná paměťová náročnost jako má klasický RWA a poměrně rychlé generování fraktálu. Lze jej využít pro generování výřezů obrazce. Transformace pro další iterace se mohou vybírat stejně jako u RWA pravděpodobnostní množinou Π, nebo zvolením hashovací funkce.

Algoritmus M-RWA, který je opět realizován dvěma vnořenými cykly, lze slovně popsat následujícím způsobem:

1. zvolíme náhodný bod P_0 v rovině/prostoru,
2. do dosažení zadaného maximálního počtu iterací opakujeme:
   (a) postupně vybíráme transformace ϕ_k,
      i. aplikace vybrané transformace na bod P_i a výpočet nového bodu P_{i+1},
      ii. pokud se nejedná o první iteraci, vykresli bod P_i.

Algoritmus stále není deterministický, protože dalšího generování se účastní pouze několik transformací, které byly aplikovány na vybrané body.

Kód algoritmu M-RWA [25], bez použití pravděpodobnosti transformací Π (výběr se provádí generátorem náhodných čísel), pak vypadá následovně:

```
// pomocná smyčka, ve které se zjistí počet transformací
for (sum=0.0, maxTransf=0; sum<1.0; sum+=(*t)[maxTransf][6], maxTransf++);

while (iter++<maxiter*100) {                    // iterační smyčka
  // smyčka, ve které se provedou všechny transformace
  for (k=0; k<maxTransf; k++) {
    float xn=(*t)[k][0]*x+(*t)[k][1]*y+(*t)[k][4];
    float yn=(*t)[k][2]*x+(*t)[k][3]*y+(*t)[k][5];
    if (iter>threshold) {          // je-li dosaženo hranice iterací
      switch (drawType) {        // výběr vykreslovacího režimu
        case IterPutPixel:
           putpixel(pix, xn*scale+xpos, yn*scale+ypos, rf ? 0xff:0x00,
                    gf ? 0xff:0x00, bf ? 0xff:0x00);
           break;
        case IterAddPixel:
           addpixel(pix, xn*scale+xpos, yn*scale+ypos, red, green, blue);
           break;
        case TransfPutPixel:
           putpixel(pix, xn*scale+xpos, yn*scale+ypos, pal[k&7][0],
                    pal[k&7][1], pal[k&7][2]);
           break;
        case TransfAddPixel:
           addpixel(pix, xn*scale+xpos, yn*scale+ypos, (!!pal[k&7][0])*dr,
                    (!!pal[k&7][1])*dg, (!!pal[k&7][2])*db);
           break;
```

```
        default:
           break;
      }
    }
  }
  k=rand()%maxTransf;     // výběr bodu x(n) na základě náhodného čísla
  hlp=(*t)[k][0]*x+(*t)[k][1]*y+(*t)[k][4];   // provedení
  y  =(*t)[k][2]*x+(*t)[k][3]*y+(*t)[k][5];   // transformace
  x  =hlp;
}
```

5.3.4 Algoritmus generování minima pixelů (MPA)

Algoritmus MPA (*Minimal Plotting Algorithm*) má proti předchozím algorirmům výhodu v tom, že se nesnaží zobrazit fraktál naprosto přesně, ale zobrazuje pouze tzv. reprezentativní část obrazce [25, 7].

Vizuálně reprezentativní část fraktálu je konečná, což znamená že je tvořena konečným počtem bodů, a proto může být zobrazena v konečném čase. MPA je efektivní, protože pracuje přímo s jednotlivými pixely v E^2 nebo E^3. Pixel se od bodu odlišuje tím, že má pevné souřadnice a pevnou velikost. Mezi dvěma pixely leží konečné množství pixelů, což pro dva body neplatí.

Pomocným datovým typem u tohoto algoritmu je fronta (queue), kam se ukládají adresy pixelů. Počáteční body se nevybírají náhodně, pro každou transformaci je vypočten pevný bod x_0 (kapitoly 2.43.3), jehož adresa je uložena do fronty. Postupně jsou z ní vybírány a jsou na ně aplikovány transformace. Nové pixely se porovnají na základě adres s pixely již vykreslenými. Neduplicitní pixely se vykreslí a jejich adresa se uloží na konec fronty, atributy ostatních pixelů se neukládají. Algoritmus nepočítá provedené iterační kroky, jelikož konec nastává vyprázdněním fronty pixelů, a ani není nutno počítat pravděpodobnost transformací. MPA je deterministický algoritmus [25, 7].

Algoritmus začíná inicializací:

- nalezení všech pevných bodů x_0 pro všechny transformace ϕ_i,
- vykreslení pixelů odpovídajících těmto bodům,
- inicializace fronty s adresami vykreslených pixelů.

Dokud není fronta prázdná, opakuj:

1. vyjmutí adresy pixelu z počátku fronty, převedení na souřadnice bodu P,
2. pro všechny transformace z množiny Φ opakuj:
   (a) aplikace transformace ϕ_k na bod P a výpočet nového bodu, P_k.
   (b) Zzokrouhlení souřadnice P_k, aby odpovídala pixelu,
   (c) pokud na dané souřadnici není vykreslen bod, vykresli bod P_k a vlož jeho adresu na konec fronty.

Výhody algoritmu MPA spočívají ve vykreslení každého bodu pouze jednou a nejsou na něj již prováděny žádné transformace. Algoritmus musí skončit v konečném čase,

protože v nejhorším případě se vygenerují všechny pixely na obrazovce. MPA je paměťově nenáročný, ukládá se pouze bitmapa obrazu a fronta s adresami pixelů.

Nevýhodou je hledání každého vypočteného pixelu ve frontě (komplikace výpočtů a zdržení při výpočtu obrazce) a také nepraktičnost při generování výřezů fraktálu (algoritmus počítá i body mimo viditelnou oblast).

5.3.5 Fractal flame

Fractal flames se generují stejně jako IFS algoritmem náhodné procházky (RWA) a obsahují rozšíření specifická pro tento druh fraktálů.

Jak již bylo uvedeno v kapitole 3.12, používají se při generování flame fraktálů nelineární transformace nazývané variace, jež jsou uvedeny v tabulce 3.3. Pro implementaci v C se vytvoří funkce vx() a vy() s třemi parametry. Zápis variací v programovacím jazyce je jednoduchý a je k nalezení v dílu nazvaném algoritmus Fractal Flame prakticky seriálu Pavla Tišnovského. Následující kód demonstruje práci s obarvováním fraktálu flame.

Pixely se přímo nezapisují do pixmapy pro vykreslování, ale ukládají se do pomocného pole, kde se mění hodnoty čítačů zavoláním funkce addfloat() [25].

```
//-------------------------------------------------------------
// Přírůstek barvy pixelu na zadaných souřadnicích ve floatové
// monochromatické pixmapě
//-------------------------------------------------------------
void addfloat(const unsigned int x, const unsigned int y)
{
    // kontrola, zda není překročena velikost pixmapy
    // (záporné hodnoty není díky převodu na unsigned int zapotřebí
    //  brát v potaz)
    if (x>=PIXMAP_WIDTH || y>=PIXMAP_HEIGHT) return;
    // zvýšit hodnotu čítače
    floatMap[y][x]++;
}
```

Z vygenerovaných hodnot se vybere hodnota maximální, zlogaritmuje se a provede se její převod do rozsahu 0..255. Nově získané hodnoty se zapíší do pixmapy, která se vykreslí na obrazovce.

```
//-------------------------------------------------------------
// Kopie pixelu z floatové pixmapy do pixmapy kompatibilní s
// OpenGL spolu s logaritmickým převodem měřítka
//-------------------------------------------------------------
void convertFloatMap(void)
{
  float maxp=0;
  float pixelFactor;                    // faktor konverze
  int   i, j;                           // čítače smyček
  // zjistit maximální hodnotu uloženou ve floatové pixmapě
  for (j=0; j<PIXMAP_HEIGHT; j++)
    for (i=0; i<PIXMAP_WIDTH; i++)
      if (maxp<floatMap[j][i]) maxp=floatMap[j][i];
```

```
  pixelFactor=255.0/log(maxp);       // vypočítat faktor konverze

  // konverze a změna na logaritmické měřítko
  // s tím, že výsledné barvové složky jsou v rozsahu 0..255
  for (j=0; j<PIXMAP_HEIGHT; j++) {
    for (i=0; i<PIXMAP_WIDTH; i++) {
      float pixel=log(floatMap[j][i])*pixelFactor;
      putpixel(pixFractalFlame, i, j, (int)pixel, (int)pixel, (int)pixel);
    }
  }
}
```

Pro vykreslení fraktálu je nutno vytvořit pole s koeficienty jednotlivých transformací, viz Root.cz [25]. Vlastní smyčka generování Fractal Flame, ve které se IFS popsaný polem obsahujícím v řádcích informace o transformacích přepočítá, může vypadat takto:

```
while (iter++<maxiter*100) {
  k=rand()%3;               // zvolit jednu ze tří transformací
  // aplikovat lineární transformaci
  xn=x*coefs[type][k][0]+y*coefs[type][k][2]+coefs[type][k][4];
  yn=x*coefs[type][k][1]+y*coefs[type][k][3]+coefs[type][k][5];
  // aplikovat nelineární transformaci
  x=vx((int)coefs[type][k][6], xn, yn);
  y=vy((int)coefs[type][k][6], xn, yn);
  if (iter>threshold) {       // je-li dosaženo hranice iterací
  double xx=x*scale+xpos;
    double yy=y*scale+ypos;
    switch (drawType) {       // výběr vykreslovacího režimu
      case IterPutPixel:
        putpixel(pix, xx, yy, rf?0xff:0x00, gf?0xff:0x00, bf?0xff:0x00);
        break;
      case IterAddPixel:
        addpixel(pix, xx, yy, red, green, blue);
        break;
      case IterLogPixel:
        addfloat(xx, yy);
        break;
      case TransfPutPixel:
        putpixel(pix, xx, yy, pal[k&7][0], pal[k&7][1], pal[k&7][2]);
        break;
      case TransfAddPixel:
        addpixel(pix, xx, yy, (!!pal[k&7][0])*dr, (!!pal[k&7][1])*dg,
                 (!!pal[k&7][2])*db);
        break;
      case TransfLogPixel:
        addfloatRGB(xx, yy, (!!pal[k&7][0]), (!!pal[k&7][1]),
                    (!!pal[k&7][2]));
        break;
```

```
      default:
        break;
    }
  }
  x=xn;
  y=yn;
}
```

5.4 Algoritmy stochastických fraktálů

5.4.1 Simulace difúze

Difúzi lze provést několika způsoby. Postup je možné popsat takto [25]:

- V určeném prostoru se vygeneruje 1 částice nebo skupina částic, tyto se nazývají semínka. Zvolené pozice mají vliv na tvar fraktálu.
- Další semínka se generují na náhodných místech.
  - Pokud se setkají s již vygenerovaným útvarem, stanou se součásti fraktálu a vygeneruje se nová částice na náhodném místě prostoru.
  - Semínka zaniknou, pokud se nesetkají s již existujícím obrazcem.
- Generování nových semínek se provádí dokud počet částic fraktálu je nižší než zvolená konstanta, nebo nějaká částice dosáhne hranice plochy pro vytváření fraktálu.

Dotyk vygenerovaného semínka s existujícím útvarem se rozlišuje podle vzdálenosti nově vytvořeného bodu od tohoto útvaru. Je zvolena limitní vzdálenost l_{min}, kdy se již považuje bod dostatečně blízký fraktálu, aby k němu mohl být připojen.

Podle tohoto postupu lze popsat algoritmus difúze pro generování stochastického fraktálu [25]. Hlavní smyčka programu opakuje vnitřní cyklus do dosažení maximálního počtu iterací:

1. vygeneruj dvojici náhodných čísel x_i, y_i,
2. pro fraktál v prostoru vygeneruj třetí souřadnici z_i,
3. z této dvojice/trojice vytvoř bod P,
4. pokud bod P leží v oblasti pro generování fraktálu a jeho vzdálenost od obrazce P_f je menši než l_{min}, přidej bod P do množiny P_f a ukonči vnitřní cyklus,
5. nové opakování vnitřního cyklu od 1.

Hlavní cyklus algoritmu simulace difúze převzatý ze seriálu na Root.cz [25] v jazyce Java pak vypadá následovně:

```
do {                                // vnější smyčka, která se zastaví
    doneInnerLoop=false;            // po nastavení ukončovací podmínky
    do {                            // vnitřní smyčka
        double x=Math.random()*(float)(width-2)+1;  // generovat bod
        double y=Math.random()*(float)(height-2)+1;
        xi=(int)x;
        yi=(int)y;
        // pokud se bod nachází v blízkosti obrazce
        if (neighboor(neighboorType, xi, yi)) {
            putPoint(xi, yi);       // přidat ho do fraktálu
            doneInnerLoop=true;     // a ukončit vnitřní smyčku
        }
    } while (!doneInnerLoop);
    counter++;
    if (counter==50) {          // průběžný výpis práce programu
        label.setText(``Points: ''+String.valueOf(i));
        counter=0;
    }
    i++;
  // test na ukončení vnější smyčky
} while (!checkStopCondition(i, xi, yi));
```

Urychlení simulace difúze

Pro uchovávání dat během generování fraktálu urychleným algoritmem simulace difúze se využívají dvě datové struktury: lineárně vázaný seznam, který obsahuje údaje o částicích, a dále pak rastrovou mřížku, plošnou nebo prostorovou, kde jsou rovněž uloženy informace o částicích, ale také příznak obsazenosti buňky. Díky tomu, že příznak lze uložit v jednom bitu, je datová struktura paměťově úsporná. Při výpočtu algoritmem simulace difúze se mění pouze příznak obsazenosti buňky a použití mřížky je proto výhodnější a rychlejší [25].

Slovní popis hlavní smyčky takto modifikovaného algoritmu simulace difúze je následující, cyklus opět opakuje do maximálního počtu iterací:

1. vygeneruj dvojici náhodných čísel x_i, y_i,
2. pro fraktál v prostoru vygeneruj třetí souřadnici z_i,
3. souřadnice $P = [x_i, y_i]$ ($P = [x_i, y_i, z_i]$) musí ležet v oblasti vymezené pro generování,
4. z čísel vytvoř dvojici/trojici tak, aby určovaly bod P,
5. pokud v okolí bodu P leží pixel/voxel s příznakem obsazenosti, nastaví se pixelu/voxelu odpovídajícímu bodu P, taktéž příznak obsazení,
6. nové opakování vnitřního cyklu od bodu 1.

5.4.2 Metoda přesouvání prostředního bodu

Jak bylo popsáno v kapitole 3.4.2, spočívá metoda přesouvání prostředního bodu v rekurzivním dělení úsečky či čtverce s následnou změnou pozice nalezeného bodu o předem definovaný koeficient Δ.

Nejjednodušším případem je dělení úsečky. Příklad hlavní funkce, jež je volána rekurzivně a realizuje rozdělení úsečky a následný posun nalezeného bodu [25], je:

```
//---------------------------------------------------------------
// Rekurzivně volaná metoda, která provede rozdělení úsečky a posun
// prostředního bodu
//---------------------------------------------------------------
void mdaRecursive1D(float x1, float y1, float x2, float y2,
                    float Delta, int iter)
{
    float x,y;
    if (!iter) {            // pokud jsme dosáhli max počtu iterací
        // vykreslení rozdělené úsečky na nejnižší úrovni
        glVertex2f(x1, y1);
        glVertex2f(x2, y2);
        return;             // a ukončení rekurze
    }
    x=(x1+x2)/2.0;          // výpočet polohy prostředního bodu
    y=(y1+y2)/2.0;
    y+=my_random()*Delta-Delta/2.0;  // posun prostředního bodu
    // rekurzivní volání na první polovinu úsečky
    mdaRecursive1D(x1, y1, x, y, Delta/2.0, iter-1);
    // rekurzivní volání na druhou polovinu úsečky
    mdaRecursive1D(x, y, x2, y2, Delta/2.0, iter-1);
}
```

Generování plazmy

Pro generování plazmy se používá rozšíření metody dělení úsečky, kdy rekurzivně dělíme čtverec. Funkci se předávají hodnoty čtyř hraničních bodů vymezující čtverec (x_1, y_1, x_2, y_2), čtyři hodnoty barev v těchto bodech (c_1, c_2, c_3, c_4) a koeficient Δ (delta) určující maximální odchylku posunu prostředního bodu [25]. Pixel se vykresluje v okamžiku, kdy velikost čtverce dosáhne rozměru 1×1 pixel. U větších čtverců se vypočtou barvy ve středech hran a středu čtverce. Poslední barva pixelu vypočtená v předchozím kroku je změněna o náhodnou hodnotu Δ a funkce je rekurzivně zavolána na čtyři podčtverce:

```
void plasma(int x1, int y1, int x2, int y2,
            int c1, int c2, int c3, int c4,
            int delta)
{
#define avg(x, y)  (((x)+(y))>>1)
#define step(x, y) (((x)-(y))>>1)
#define bound(x)   if (x>255) x=255; if (x<0) x=0;
#define displac(x, delta)  (x+(rand()%(delta*2)-delta))
```

```
  // pokud jsme nedosáhli jednopixelového kroku
  if (x2-x1>1) {
     int dc12=avg(c1, c2);      // výpočet průměru barev na hranách čtverce
     int dc13=avg(c1, c3);
     int dc24=avg(c2, c4);
     int dc34=avg(c3, c4);
     int dc  =avg(dc13, dc24);  // výpočet průměrné barvy ve středu čtverce
     int dx=step(x2, x1);       // krok - polovina velikosti čtverce
     int dy=step(y2, y1);
     if ((x2-x1>2) && (delta>0))
         dc=displac(dc, delta); // posun středního bodu o náhodnou hodnotu
     bound(dc);              // zajištění mezí barev v povoleném rozsahu
     bound(c1);
     bound(c2);
     bound(c3);
     bound(c4);
     delta>>=1;      // maximální náhodná hodnota se zmenšuje na polovinu
     // rekurzivní rozdělení všech čtyř podčtverců
     plasma(x1,    y1,    x1+dx, y1+dy, c1,   dc12, dc13, dc,   delta);
     plasma(x1+dx, y1,    x2,    y1+dy, dc12, c2,   dc,   dc24, delta);
     plasma(x1,    y1+dy, x1+dx, y2,    dc13, dc,   c3,   dc34, delta);
     plasma(x1+dx, y1+dy, x2,    y2,    dc,   dc24, dc34, c4,   delta);
  }
  else {                     // bylo dosaženo hranice rozlišení pixmapy
     putpixel(pix, x1, y1, (c1+c2+c3+c4)/4, (c1+c2+c3+c4)/4,
              (c1+c2+c3+c4)/4);
  }
#undef avg
#undef step
#undef bound
}
```

Další příklady algoritmů pro generování plazmy je možno najít v [25, 20, 14]

Kapitola 6

Výsledky práce

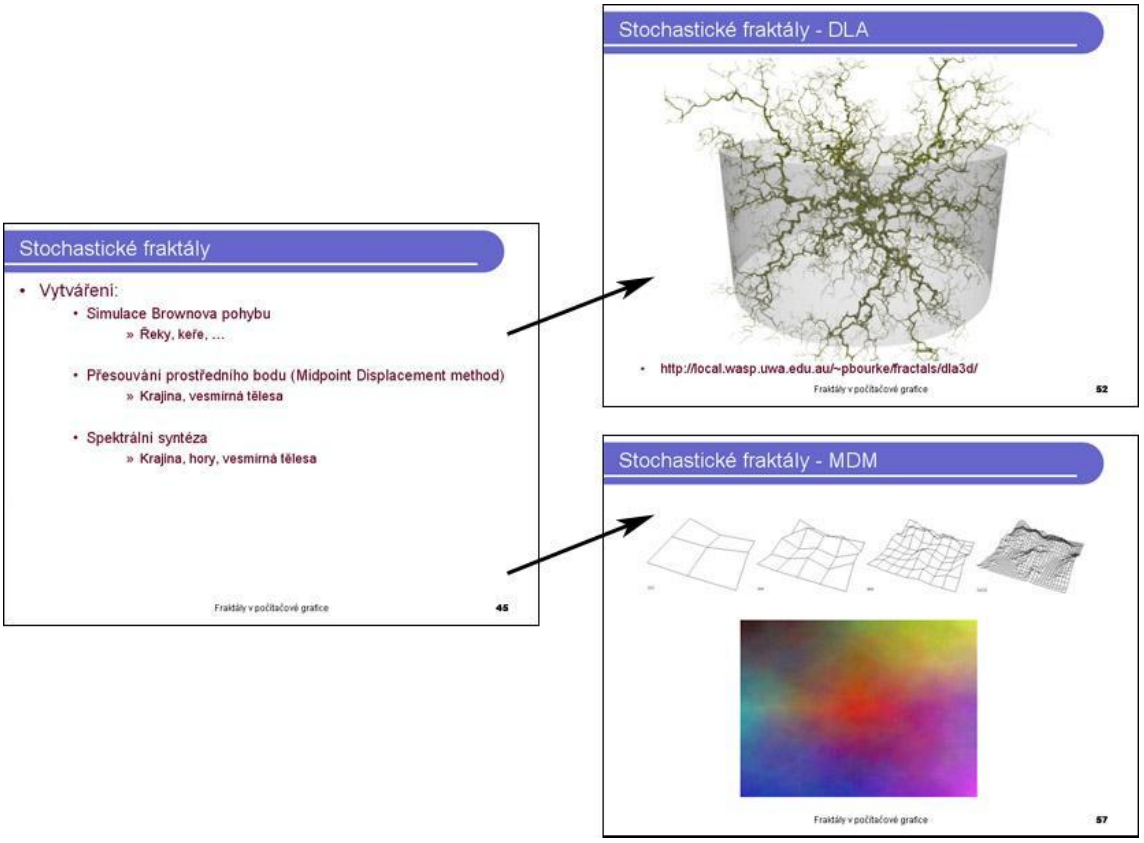
6.1 Slajdy

Jedním z cílů diplomové práce bylo jednoduchým a srozumitelným způsobem objasnit základní vývoj, dělení a využití fraktální geometrie. Součástí praktické části práce byla i tvorba slajdů pro možnost prezentace tématu fraktální geometrie studentům FIT VUT v Brně. Tato problematika by mohla být součástí předmětu Počítačová grafika a její grafické ztvárnění formou prezentace bylo průběžně konzultováno s přednášejícím, aby odpovídalo struktuře a časovým možnostem výukové hodiny. Slajdy jsou k dispozici ve formátech PDF a PPT o rozsahu 59 stran.

První slajd je pokryt přehledem dále uváděné látky. V úvodu je nastíněno základní dělení objektů a historie "zvláštních útvarů" před rokem 1975, kdy poprvé v jejich souvislosti zazněl pojem "fraktál". Jednotlivé etapy vývoje fraktálních obrazců provázejí významná jména, jako je Lévy, Hausdorf, Mandelbrot aj. Ještě před samotným výčtem jednotlivých typů fraktálů jsou ve slajdech uvedeny nejdůležitější pojmy a charakteristiky (obr 6.1), které v přednášce budou slovním doprovodem definovány.

Obrázek 6.1: Základní pojmy

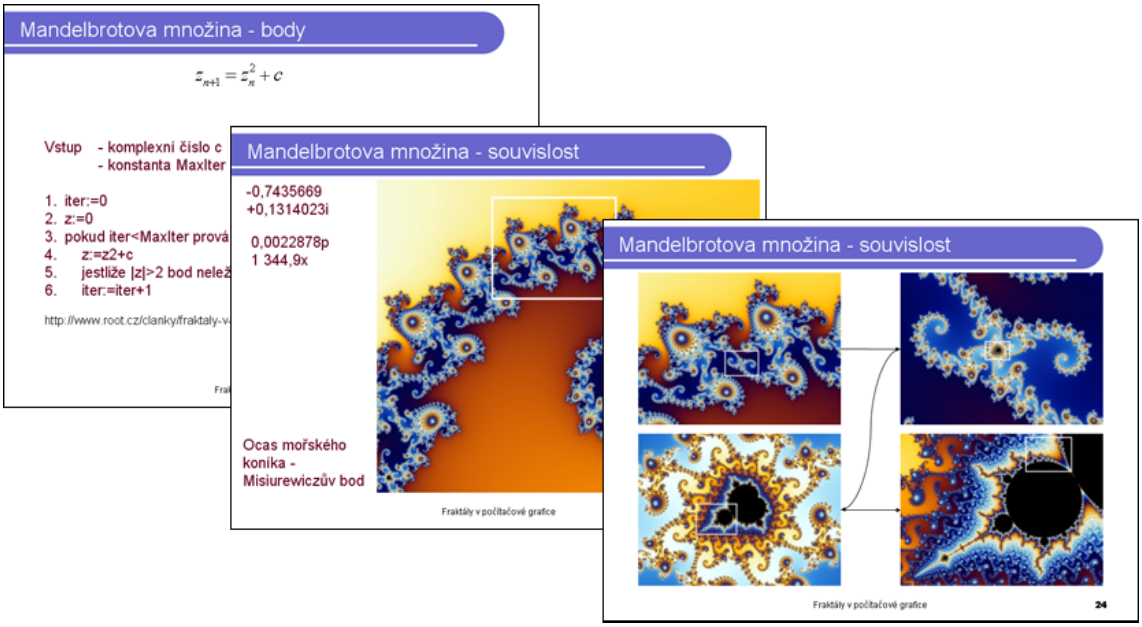
Tématicky jsou slajdy dále řazeny do kapitol, které pokrývají problematiku dělení fraktálních obrazců do skupin na základě podobných charakteristických vlastností, jak lze vidět na obrázku 6.2.



Obrázek 6.2: Charakteristiky skupin fraktálů

Vlastní prezentace poznatků fraktální geometrie by pak měla odpovídat struktuře a rozsahu uváděnému v diplomové práci. Informace nad rámec práce lze dohledat v literatuře.

Graficky jsou uvedeny typické příklady jednotlivých systémů a u některých pro názornost ukázány detaily nacházející se v jednotlivých segmentech fraktálů, viz obrázek 6.3. Tímto je demonstrována soběpodobnost a spojitost útvarů při změně měřítka.



Obrázek 6.3: Demonstrace souvislosti

Důraz je kladen na grafické ztvárnění zvolené problematiky a přehlednost, tedy na obsah prezentace nikoliv na způsob její tvorby. Z tohoto důvodu je voleno nevýrazné pozadí samotných slajdů.

Literatura k přednášce je zmíněna na posledním slajdu, přesnější seznam použité literatury je uveden v závěru této práce. Zdroje u některých typických příkladů fraktálů jsou pro jejich rychlejší dohledání uváděny přímo na stránce prezentace, která se jimi zabývá.

Celá diplomová práce je tedy přehledně obsažena na již zmiňovaných slajdech, které budou pro účely výuky studentům k dispozici. Pokud se student podívá na slajdy blíže, jistě zde nalezne zajímavé podněty k další diskuzi a studiu.

6.2 Demonstrační program

Dalším cílem této práce bylo vytvořit program zobrazující vybrané fraktály tak, aby svým ovládáním a funkčností byl snadno použitelný pro vykreslení základních typů fraktálních obrazců. Před vlastním vykreslením je možno upravit vlastnosti generovaných útvarů a uložit barevné palety sloužící k obarvení výsledného fraktálu.

Demonstrační program byl vytvořen v Borland Developer Studio 2006 a funguje v prostředí Windows. Jeho záměrem je ilustrovat souběžně s výkladem některé druhy fraktálů. Díky předvoleným oblastem zobrazení není nutné složité nastavování parametrů při každém generování. V případě Mandelbrotovy množiny je možno zobrazit předvolené body, v jejichž okolí lze pozorovat specifické oblasti dané množiny. Jedná se především o Fibonacciho posloupnost, posloupnost přirozených čísel atd.

6.2.1 Realizace

Kvůli přehlednosti a logickému oddělení jednotlivých částí kódu je program rozdělen do několika modulů. Moduly lze podle jejich funkce rozčlenit na tři celky. První částí je *fractalfunc*, obsahuje algoritmy pro generování fraktálů a definice jejich parametrů. Moduly *unit1* a *unit2* zajišťují zobrazování vypočítaných hodnot ve formě obrázků, zpracovávájí parametry zadané uživatelem. Soubory *PaletteManager* a *coltable* jsou poslední skupinou a obstarávají funkce správce palet. Jednotlivé moduly programu jsou:

- Coltable.[cpp/h] - obsahuje definice struktur palet a funkce pro výpočet barev palety z mezních bodů.
- Fractalfunc.[cpp/h] - modul s definicemi funkcí, kterými se fraktály generují. Zároveň obsahuje definice jejich parametrů, u IFS se jedná např. o různé druhy transformací.
- Fractals.cpp - centrální modul programu, inicializuje okna programu.
- PaletteManager.[cpp/h] - slouží k nadefinování barevné palety, obsahuje akce pro její obsluhu jako je uložení, načtení, úprava.
- Unit1.[cpp/h] - rozhraní hlavního okna programu DemFract, definice vzhledu.
- Unit2.[cpp/h] - okno pro vykreslení fraktálu ShowFract, výběr oblasti zobrazení.
- Unit3.[cpp/h] - zajištuje vykreslení progress baru.
- Utilz.[cpp/h] - obsahuje funkce pro práci se soubory a další pomocné funkce.

V celém programu byl namísto datové struktury pole většinou využit standardní kontejner *vector* z důvodu robustnosti. Soubory uchovávaných hodnot se během činnosti programu často mění a použití standardního pole vyžaduje velkou pozornost při správě paměti. *Vector* přidělování a uvolňování paměti obsluhuje sám a jeho použití je tak mnohem bezpečnější. Použití *vectoru* je sice časově náročnější, ale v časově kritických úsecích programu není nutno s *vectorem* pracovat, doba generování fraktálu se tak neprodlužuje.

Modul fractalfunc

Modul *fractalfunc* obsahuje definice algoritmů jednotlivých druhů fraktálů. Definicí algoritmu se rozumí jméno algoritmu, např. *IFS Fraktály*, ukazatel na prováděcí funkci a vektor definic potřebných parametrů. Tyto hodnoty jsou uchovány ve struktuře *tDEFDRAWPARAMS*. Příklad definic parametrů vidíme na ukázce funkce *IFSInit*.

Na počátku si nadefinujeme jméno funkce a ukazatel. Dalším krokem je nastavení souřadnic vykreslované oblasti:

```
tDEFDRAWPARAMS IFSInit() {
   tDEFDRAWPARAMS pDP; // struktura obsahujici definice vykreslovaci
                       // funkce a jejich parametru
   tPARAMINFO pPI;  // struktura obsahujici definici jednoho parametru

   pDP.FuncName = ''IFS Fraktály'';
   pDP.pDrawFunc = &IFSFunc;  // ukazatel na adresu vykreslovaci funkce
```

```
   // Defaultni hodnoty Max a Min souradnic vykreslovane oblasti
   pDP.ldDefMinX = -5;
   pDP.ldDefMaxX =  5;
   pDP.ldDefMinY =  0;
   pDP.ldDefMaxY =  10;
```

Specifické parametry pro každý fraktál představuje počet iterací, transformace použité pro výpočet, druh obarvení atd. Pro IFS fraktály jsou to např.:

```
   // definice prvniho parametru - pocet iteraci
   pPI.type = defINT;
   pPI.name = ‘‘Iterace’’;
   pPI.iMin = 0;
   pPI.iMax = 1000000000;
   pPI.iStep = 10000;
   pPI.defVal = 500000;
   pDP.params.push_back(pPI);

   // definice druheho parametru - typ transformace
   pPI.type = defENUM; // typ pozadovaneho parametru (vycet)
   pPI.name = ‘‘Transformace’’; // nazev parametru

   // seznam pojmenovani vyctovych hodnot (zde transformace)
   pPI.defEnumCaptions.clear();
   pPI.defEnumCaptions.push_back(‘‘Trojúhelník’’);
   // ... a dalsi
   pPI.defEnumCaptions.push_back(‘‘Barnsleyova kapradina’’);

   // pocatecni transformace
   pPI.defVal = 1;   // defaultni hodnota pocatecni transformace
   pDP.params.push_back(pPI); // vlozeni parametru do vektoru parametru
   // ... definice dalsich parametru
   return pDP;   // vraci se struktura definice algoritmu
}
```

Definici algoritu obsahuje struktura *tDEFDRAWPARAMS*. Těmito strukturami se naplní vektor *DefDrawParams*, který pak obsahuje definice všech algoritmů použitých v demonstračním programu.

Předání parametrů prováděcí funkci se pak provádí pomocí struktury *tDRAWFUNCPARAM*, kde se předává vektor ukazatelů na obecný typ *std::vector< void * > params*. Každý jeho prvek se později přetypovává zpět na typ uvedený při definici v inicializační funkci.

Nejdůležitější částí modulu *fractalfunc.cpp* je funkce *InitDrawFuncMenu*, která naplní vektor *std::vector<tDEFDRAWPARAMS> DefDrawParams;* definicemi algoritmů:

```
void InitDrawFuncMenu(void) {

/*Mandelbrot Initialize Part*/
   DefDrawParams.push_back(MandelInit());

/*IFS-Fract Initialize Part*/
   DefDrawParams.push_back(IFSInit());

};
```

MandelInit (*IFSInit*) vrací strukturu *tDEFDRAWPARAM*, která se uloží do vektoru *DefDrawParams*. Struktura *tDEFDRAWPARAM* obsahuje vždy definici jednoho algoritmu, viz výše.

V případě, že je do *fractalfunc.cpp* dopsán další druh algoritmu, je nutno zařadit jeho inicializaci do funkce *InitDrawFuncMenu* na konec. V opačném případě by bylo nutné přepsat indexy algoritmů v definičním soubor pro položku menu "Speciál", protože indexy položek jsou závislé na pořadí struktur *tDEFDRAWPARAMS* ve vektoru *DefDrawParams*.

Vzhledem k povaze problému nebyl použit objektový přístup pro implementaci algoritmů. Objektový přístup by sice fugoval, avšak nepřinesl by výrazné zefektivnění kódu a abstrakce by byla zcela samoúčelná.

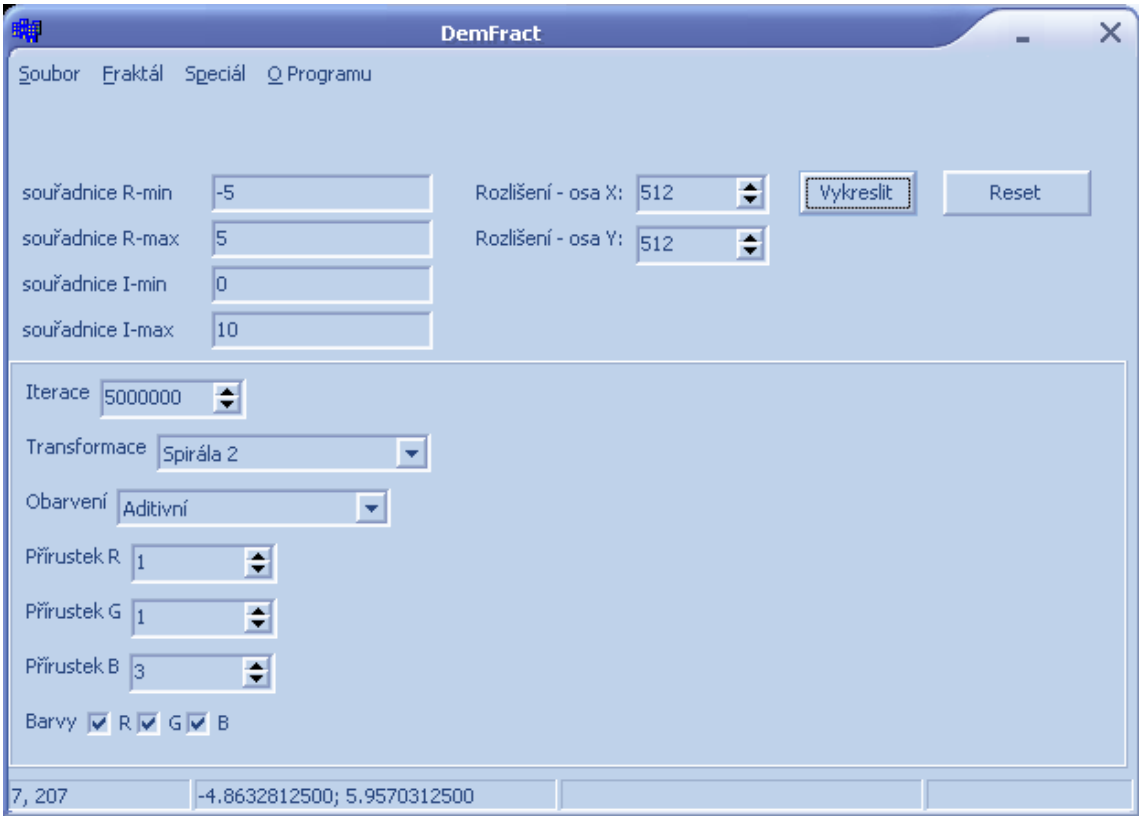
6.2.2 Ovládání programu

Hlavními ovládacími prvky programu jsou funkční tlačítka a boxy pro nastavení souřadnic okna, iterací a v případě IFS parametry obarvení.

Výběr skupiny fraktálů se volí v položce menu "Fraktál". Na výběr je Mandelbrotova množina a IFS fraktály. V hlavním okně jsou zobrazeny boxy se souřadnicemi oblasti, kam se bude fraktál vykreslovat, nastavení velikosti okna a tlačítka pro vygenerování fraktálu a reset souřadnic. Souřadnice fraktálu a rozlišení lze uložit do externího souboru pro pozdější použití.

Na obrázku 6.4 je zobrazeno okno programu s nastavenými parametry IFS fraktálu. V případě vykreslování Mandelbrotovy množiny je možno nastavit v okně *DemFract* počet iterací, zvolit typ obarvení, paletu barev a druh obarvovací funkce. Posledním nastavením je *check box* s volbou obarvení konvergujících bodů množiny (vnitřní body). Palety barev lze vybírat až po jejich vytvořeni nebo načtení ve správci palet. Správce palet se nachází pod položkou "Paleta barev" v menu "Fraktál". Vytvoření palety je popsáno dále.

Ve vykresleném fraktálu v okně *ShowFract* lze myší vybírat oblasti pro přiblížení. Výběr se provádí stiskem levého tlačítka a tažením myší, viz obr 6.5. V případě, že daná oblast nevyhovuje představám, lze současným stiskem pravého tlačítka výběr zrušit. Při nechtěném označení pouze jednoho bodu pro zvětšení (dvojklik), program vypíše upozornění a je možno vybrat korektní oblast. K orientaci a pro utvoření představy o vzdálenostech mezi útvary fraktálu slouží zobrazení souřadnic, nad kterými se kurzor pohybuje. Tyto jsou vypsány ve spodní části okna. V levém dolním rohu jsou souřadnice vztažené k oknu, vedle pak souřadnice samotného fraktálu.



Obrázek 6.4: Okno programu

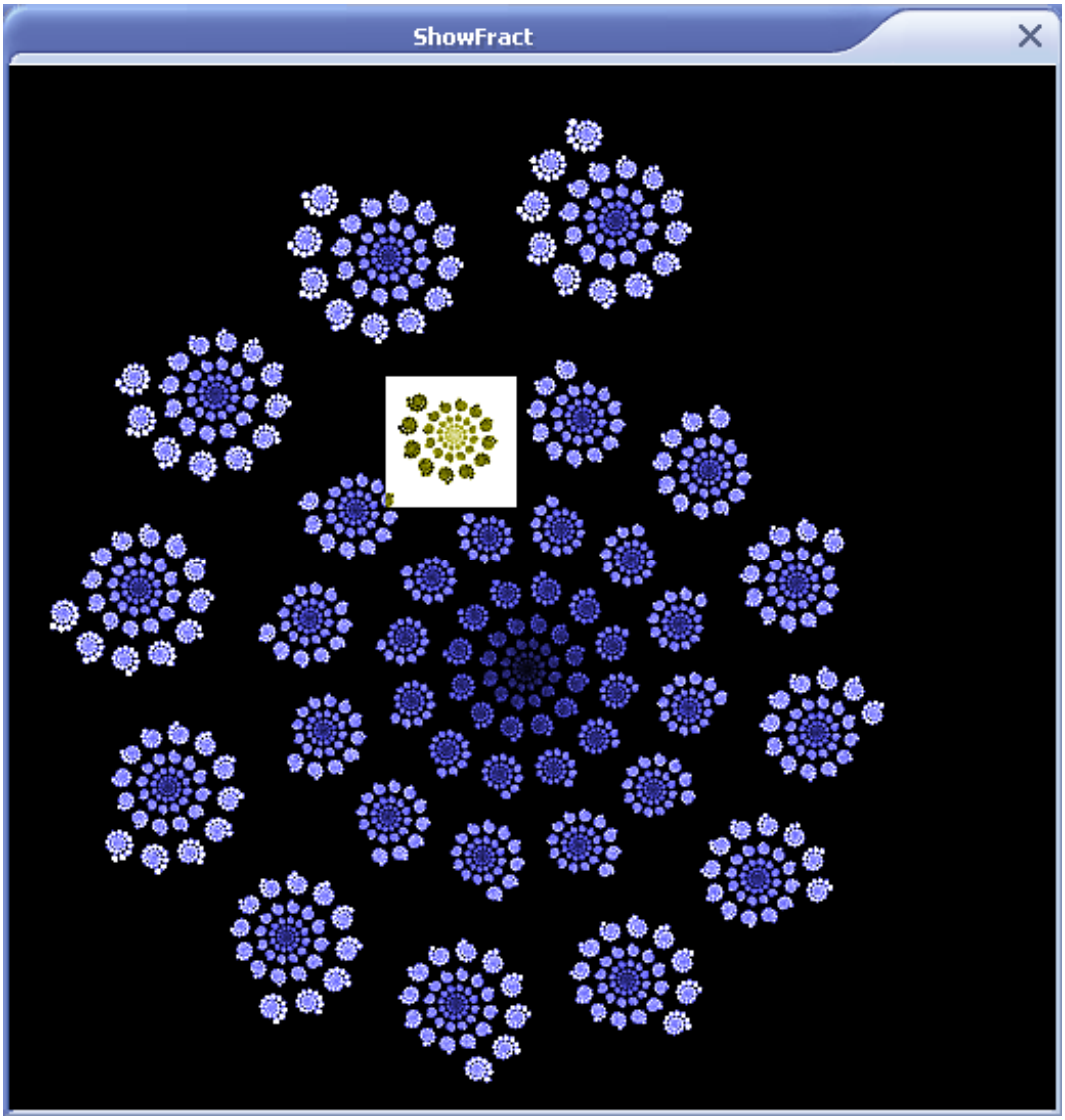
Menu Speciál

V této položce menu lze nalézt předvolené zajímavé oblasti Mandelbrotovy množiny. Nadefinovány jsou oblasti, kde lze vypočítat číslo Π, spirální útvary představující Fibonacciho posloupnost, posloupnost přirozených čísel aj.

Menu je načítáno ze souboru *specmenu.def*. Struktura vypadá následovně:

```
[Mandelbrot;0]
[Pi;0]
@Pi 1
!-0,7793984375;-0,7223828125;-0;0,0990234375
@Pi 2
!0,21484375;0,35546875;-0,064453125;0,076171875
[..]
```

V hranatých závorkách je na prvním místě název položky a za středníkem pak index určující pro který druh má být menu aktivní(0 - Mandelbrotova množina, 1 - IFS). Index je závislý na pořadí inicializace algoritmů v modulu *fractalfunc.cpp* ve funkci *InitDrawFuncMenu*. Zanořená nabídka se vytvoří další pojmenovanou položkou uzavřenou v hranatých závorkách. Jednotlivé položky menu jsou pak uvozeny znakem @, souřadnice pro vykreslení následují na dalším řádku za znakem "!" a jsou odděleny středníkem. Položky vnořeného menu jsou uzavřeny sekvencí znaků `[..]`.

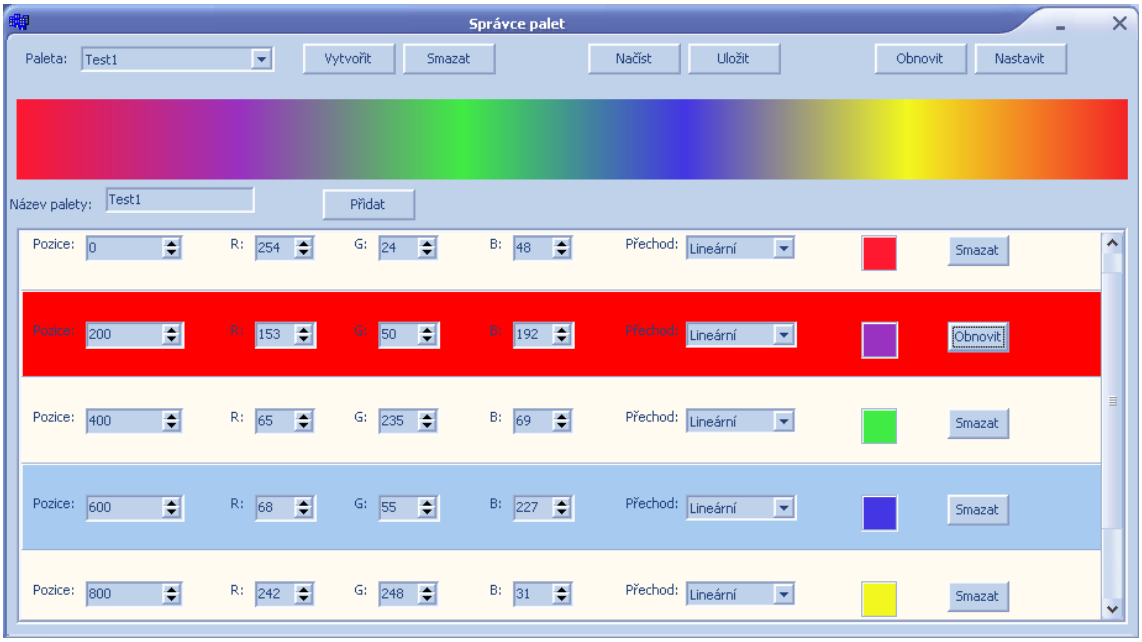


Obrázek 6.5: IFS spirála

6.2.3 Správce palet

Fraktály získávájí na vizuální atriktivitě především obarvením, které zdůrazní jejich komplexnost a složitost. Pro sestavení barevných palet slouží *Správce palet*, který lze vidět na obrázku 6.6. Přístup ke správci je přes položku "Paleta barev", která se nachází v nabídce "Fraktál".

Hlavními ovládacími prvky jsou funkční tlačítka v horní části okna, nabídka s výběrem načtených palet v levé části a tlačítko pro přidání dalšího bodu do palety. Defaultně je nastaven počáteční a koncový bod palety. Tlačítkem "Přidat" se přidá nový bod s implicitně nastavenou pozicí 0 a černou barvou. Pozice udává, kde se bude bod v posloupnosti nacházet, a určuje také počet barev, který je možno až po tento bod přidělit. Barvu lze zadat buď nastavením složek *RGB* nebo ji vybrat v paletě. Barevné přechody je možno počítat lineárně, logaritmicky nebo exponenciálně, se zvláštním nastavením pro každý bod.



Obrázek 6.6: Správce palet

Tlačítem “Smazat” se označí bod určený pro výmaz. Dalším stiskem tlačítka se příznak smazání zruší.

Pruh pod funkčními tlačítky ilustruje rozložení barev v paletě, avšak po změně parametrů bodu je nutno stisknout buď “Obnovit” pro aktualizaci barev, nebo “Nastavit” pro překreslení pruhu a současné seřazení bodů podle nastavené pozice v paletě.

Vytvořenou paletu lze uložit do externího souboru s příponou *pal*. Data jsou uložena jako text z důvodu snadnějšího importu či manuální editace barev. Program nabízí také možnost uloženou paletu smazat.

V případě, že počet iteračních kroků přesáhne počet barev nadefinovaných v paletě, celá paleta se cyklicky opakuje.

Kapitola 7

Závěr

Dle zadání je předkládána diplomová práce pojednávající o historii fraktální geometrie a jejím uplatnění nejen v počítačové grafice, ale také v dalších oblastech vědy a techniky.

Jelikož hlavním záměrem diplomové práce bylo na konkrétních příkladech obsáhnout problematiku fraktální geometrie, jsou historické souvislosti uvedené v práci pro ilustraci vývoje dostatečné. Matematické aspekty sestávající z popisů dimenzí a metrik prostorů jsou spíše předmětem pro teoretické zkoumání, proto jsou také v textu podány pouze ve zkrácené formě. Pro detailnější informace jsou uváděny odkazy na odbornou literaturu.

Dalším cílem práce bylo vytvořit materiály pro základní seznámení s fraktální problematikou v rámci přednášek předmětu Počítačová grafika. Byly zhotoveny slajdy tvořící kostru přednášky a demonstrační program zobrazující některé fraktální útvary. Program nabízí možnost zobrazení předvolených oblastí, např. posloupnost přirozených čísel, Fibonacciho posloupnost aj. zejména v Mandelbrotově množině. Počet příkladů a jejich složitost je volena tak, aby byly zachyceny nejtypičtější fraktály dané skupiny s ohledem na přijatelnou časovou náročnost vykreslení útvarů. Program lze jednoduše rozšířit o další typy fraktálů doplněním algoritmů a parametrů útvaru do příslušného modulu. Za účelem prezentace diplomové práce byl vytvořen plakát s náhledy programu a slajdů. Na plakátu jsou obsaženy základní informace o fraktální geometrii pro atraktivnost doplněné ukázkami fraktálů.

Potenciální možnost rozšíření práce spočívá v popisu zajímavostí u zvolených typů fraktálů. Další kapitolu mohou tvořit jiné algoritmy pro generování fraktálů, jejich popis a modifikace. Zajímavé by mohlo být porovnání těchto modifikací s původními algoritmy pro zdůraznění odlišné náročnosti na zpracování výstupů.

Přínosem této práce je shrnutí jak teoretických poznatků fraktální geometrie, tak jejich praktické aplikace. Rozsah tohoto souhrnu by měl odpovídat jednomu přednáškovému bloku doplněnému o možnost vyzkoušení programu a ověření vlastností fraktálů uváděných v kapitole 2 na vygenerovaných příkladech. Text je obsahově i strukturou přizpůsoben i "začátečníkům" v oblasti počítačové grafiky .

Všechny body zadání diplomové práce byly z pohledu autora splněny.

Literatura

[1] Kristopher Babic. Using fractals to simulate natural phenomena. `http://davis.wpi.edu/~matt/courses/fractals/index.htm`. Vyhledáno v dubnu 2007.

[2] Michael Barnsley, R. L. Devaney, Benoit Mandelbrot, Heinz-Otto Peitgenn, Dietmar Saupe, and Richard Voss. *The Science of Fractal Images*. Springer-Verlag, 1988.

[3] Michael F. Barnsley. *Fractals Everywhere*. Morgan Kaufmann, 2000. ISBN 0-12-079069-6.

[4] Markéta Breuerová. *Fraktály*. Praha. 2006. Vysoká škola Umělecko Průmyslová v Praze. Diplomová práce.

[5] Scott Draves. The fractal flame algorithm. `http://flam3.com/flame_draves.pdf`, 2005.

[6] Ron Eglash. *African Fractals: modern computing and indigenous design*. Rutgers University Press, 1999. ISBN 0-8135-2614-0.

[7] Bri Tish Technology Group Ltd. (London GB2), Donald Martin Monro, and Frank Dudbridge. *Fractal coding of data*. US Patent 5768437, 16. 6. 1998. Int G06K 009/36, G06K 009/46.

[8] Ray Girvan. The mandelbrot monk. `http://classes.yale.edu/fractals/MandelSet/MandelMonk/MandelMonk.html`. Vyhledáno v listopadu 2006.

[9] Martin Hinner. Jemný úvod do fraktálů. `http://martin.hinner.info/math/Fraktaly/`. Vyhledáno v listopadu 2006.

[10] Petra Konečná. *Vytvoření ukázkových příkladu pro předmět Základy počítačové grafiky*. Ostrava. 2006. FEI VŠB TU Ostrava. Diplomová práce.

[11] Benoit B. Mandelbrot. *Fraktály: Tvar, náhoda a dimenze*. Mladá Fronta, 2003. ISBN 80-204-1009-0.

[12] Benoit B. Mandelbrot. *The Fractal Geometry of Nature*. W. H. Freeman and Company, 2004. ISBN 0-7167-1186-9.

[13] Ivo Marak. On synthetic terrain erosion modeling:a survey. `http://www.cescg.org/CESCG97/marak/`. Vyhledáno v květnu 2007.

[14] Paul Martz. Generating random fractal terrain. http://www.gameprogrammer.com/fractal.html. Vyhledáno v květnu 2007.

[15] Martin Netuka. *Nelinearita výnosů cenných papírů*. Praha. 1997. Fakulta Sociálních Věd Univerzity Karlovy. Diplomová práce.

[16] Petr Pauš. Počítačové generování fraktálních množin. http://kmlinux.fjfi.cvut.cz/~pausp1/html/skola/fraktaly/reserse.htm. Vyhledáno v prosinci 2006.

[17] Martin Přikryl. Fraktální komprese. http://www.prikryl.cz/cze/htmlseminarka.php?id=fraktal. Vyhledáno v květnu 2007.

[18] Paul W. K. Rothemund, Nick Papadakis, and Erik Winfree. (2004) algorithmic self-assembly of dna sierpinski triangles. PLoS Biol 2(12): e424 doi:10.1371/journal.pbio.0020424. Vyhledáno v květnu 2007.

[19] Paul W. K. Rothemund, Nick Papadakis, and Erik Winfree. (2004) using biology to create complex patterns. PLoS Biol 2(12): e448 doi:10.1371/journal.pbio.0020448. Vyhledáno v květnu 2007.

[20] Justin Seyster. Plasma fractal. http://www.sinc.stonybrook.edu/Stu/jseyster/plasma/. Vyhledáno v květnu 2007.

[21] WWW stránky. Benoit mandelbrot - wikipedia. http://en.wikipedia.org/wiki/Benoît_Mandelbrot. Vyhledáno v listopadu 2006.

[22] WWW stránky. Fractal - wikipedia. http://en.wikipedia.org/wiki/Category:Fractals. Vyhledáno v dubnu 2007.

[23] WWW stránky. Stochastic processes - wikipedia. http://en.wikipedia.org/wiki/Category:Stochastic_processes. Vyhledáno v dubnu 2007.

[24] Pavel Tišnovský. Fraktální stránky. http://www.fit.vutbr.cz/~tisnovpa/fract/uvod.html. Vyhledáno v dubnu 2007.

[25] Pavel Tišnovský. Fraktály v počítačové grafice. http://www.root.cz/serialy/fraktaly-v-pocitacove-grafice/. Vyhledáno v květenu 2007.

Seznam obrázků