Rešeršní práce

Počítačové generování fraktálních množin

Petr Pauš

Školitel : Dr. Ing. Michal Beneš
Zaměření : Tvorba software
Katedra : KM
Akademický rok : 2003/2004
Rok studia : 3.

Obsah

1 Úvod

1.1 Fraktál

Fraktál je geometrický objekt, který po rozdělení na menší části vykazuje tvarovou podobnost s těmito částmi. Fraktálními objekty se zabývá samostatná vědní disciplína nazývaná fraktální geometrie. Tato disciplína je intenzivně rozvíjena zhruba od šedesátých let minulého století. Za jejího zakladatele je dnes považován matematik Benoit B. Mandelbrot, který jako první matematicky definoval pojem fraktál. I před zavedením pojmů fraktál a fraktální geometrie se vědci a umělci zabývali geometrickými útvary, které dnes nazýváme fraktály, jako například sněhovou vločku Kochovu (Koch snowflake) nebo Sierpinského trojúhelník (Sierpinsky triangle).

Protože velká část fraktálů je využívána v počítačové grafice a fraktály lze nejlépe popsat jako geometrické objekty, můžeme fraktál nejjednodušeji definovat jako nekonečně členitý útvar. Opakem nekonečně členitého útvaru je geometricky hladký útvar, který lze popsat klasickou Euklidovskou geometrií.

1.2 Soběpodobnost

Soběpodobnost (matematicky se tato vlastnost nazývá *invariance vůči změně měřítka*) je taková vlastnost objektu, že objekt vypadá stejně, ať se na něj díváme v jakémkoliv zvětšení.

Soběpodobnost je hlavním znakem fraktálních útvarů a většinou je také považována za jejich definici. To nám také pomáhá při vyhledávání fraktálních útvarů v přírodě. Soběpodobný je například kámen, hory, mraky, stromy, rostliny ale i krátery atd., tedy objekty živé i neživé přírody.

Matematicky je soběpodobná množina definována takto: Soběpodobná množina A n-dimenzionálního Euklidovského prostoru E_n je taková množina, pro níž existuje konečně mnoho kontrahujících zobrazení $\phi_1, ..., \phi_n$ takových, že A vznikne jako:

$$A = \bigcup_{i=1}^{n} \phi_i(A) \quad .$$

Takto definovaná množina má několik velmi zajímavých vlastností:

- Soběpodobná množina vzniká opakováním sebe sama při určité transformaci (změna měřítka, rotace, posunutí, zkosení).
- Soběpodobné množiny jsou invariantní vůči změně měřítka. Při libovolném zvětšení, či zmenšení vypadají podobně.
- Soběpodobná množina vzniká sama ze sebe, respektive vzniká opakováním téhož motivu.

Princip opakování podobných tvarů ve zmenšené podobě je vidět prakticky u jakékoliv komplexní, složité struktury, která je vytvářena i pomocí velmi jednoduchých pravidel. Způsob, jakým probíhá větvení stromů či cév a žil v tělech živočichů, nebo hromadění baktérií a řas v koloniích, se dá matematicky uspokojivě popsat pouze fraktální geometrií.

Fraktály však slouží i k modelování a pochopení složitých dějů, které se odehrávají v čase, jedná se tedy o jevy dynamické.

1.3 Atraktor

Atraktor (anglicky attractor) dynamického systému je množina stavů, do kterých systém směřuje. Je to tedy množina hodnot, kterých může nabývat stavový vektor dynamického systému po dostatečně dlouhém časovém úseku od počátečního impulsu.

Atraktory se dají rozdělit do několika tříd:

- atraktorem jsou pevné body
- atraktorem jsou periodické body
- atraktorem jsou kvaziperiodické body
- atraktor je chaotický, tzv. podivný atraktor

Jsou-li atraktorem dynamického systému pevné body, jde o nejjednodušší případ, protože se systém v nekonečném čase ustálil v nějakém stabilním stavu, který lze dopředu vypočítat.

Jsou-li atraktorem periodické (resp. kvaziperiodické) body, jde také o jednoduchý případ. Systém se ustálil tak, že osciluje mezi několika stavy.

Je-li atraktor chaotický, znamená to, že výsledný stav systému nelze v podstatě 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 bavíme o deterministických systémech.

Podivný atraktor (anglicky *strange attractor*) je z hlediska fraktální geometrie nejzajímavějším případem atraktoru. Tento typ atraktoru může vzniknout tehdy, je-li systém popsán minimálně třemi diferenciálními rovnicemi. Takový systém může mít velmi komplikovaný atraktor, který sice bude vykazovat vlastnosti pravidelného, ale současně i chaotického atraktoru. Termín *podivný atraktor* není ještě přesně matematicky definován (definice sice existují, ale nezahrnují všechny typy podivných atraktorů), ale považujeme za něj takový atraktor, který vykazuje stejné vlastnosti, jaké mají fraktály. Všechny chaotické atraktory jsou současně podivnými atraktory, opačná implikace však neplatí.

2 Klasické fraktály

Jedny z prvních fraktálů vznikly jako pokus o nalezení hranic matematických pojmů. Slavní matematici jako Georg Cantor, Giuseppe Peano, David Hilbert, Niels Fabian Helge von Koch, Waclaw Sierpinski, Gaston Julia či Felix Hausdorff vymysleli různé matematické objekty vyhovující definicím, ale svými vlastnostmi velmi podivné. Například Kochova křivka, která je spojitá, avšak nikde nemá ani první derivaci. Tyto objekty byly považovány spíše za výjimky, za „matematická monstra“. Uveďme některé z těchto monster.

2.1 Cantorova množina

Cantorova množina je množina bodů z uzavřeného intervalu <0; 1>. Nejjednodušší definicí této množiny je popis, jak ji získat. Interval <0; 1> rozdělíme na tři shodné a otevřený interval (1/3; 2/3) vyjmeme. Čísla 1/3 a 2/3 nám tedy zůstanou v množině. Takto jsme získali dva uzavřené intervaly <0; 1/3> a <2/3; 1> o délce 1/3. Nyní opakujeme tento postup na nové intervaly, tj. z obou intervalů vyjmeme prostředek. To provádíme až do nekonečna. Body které zbudou prohlásíme za Cantorovu množinu, viz obr. 1.

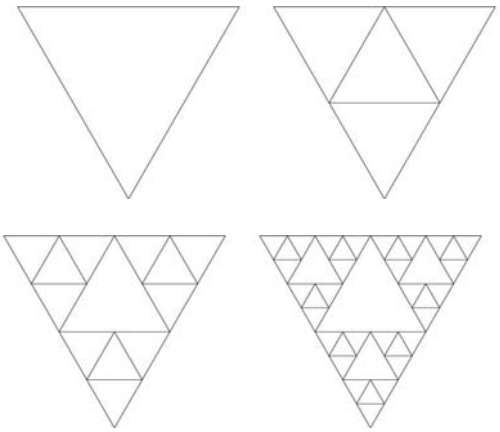
Obrázek 1 – Konstrukce Cantorovy množiny

Které body tedy patří do množiny? Jistě to jsou 0, 1, 1/3, 2/3, 1/9, 2/9, 7/9, 8/9..., tedy krajní body všech intervalů, kterých je spočetně mnoho. Nejsou to ale všechny body. Cantorova množina je nespočetná, zbývá tedy ještě nespočetně mnoho bodů, které náleží množině. Už toto je celkem zajímavá vlastnost.

2.2 Sierpinského trojúhelník a koberec

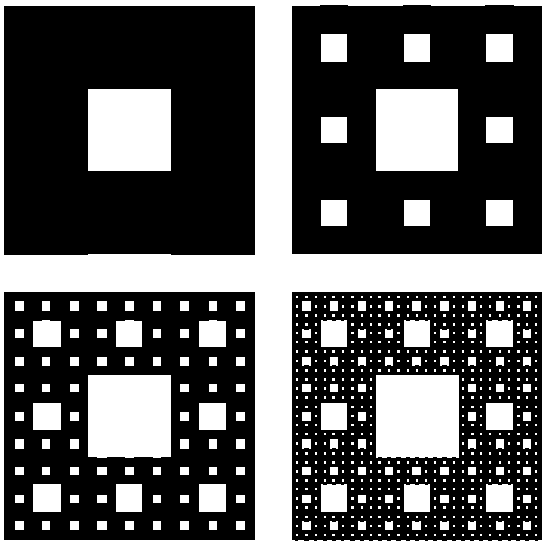
Další dva klasické fraktály vymyslel polský matematik Waclaw Sierpinski. Nejprve uveďme Sierpinského trojúhelník.

Konstrukce: začneme s rovnostranným trojúhelníkem o hraně 1, ten rozdělíme na 4 stejné s délkou hrany 1/3 a prostřední vyjmeme. Tento proces opakujeme nekonečněkrát, viz obr 2.



Obrázek 2 – Konstrukce Sierpinského trojúhelníku

Vylepšením toho objektu je Sierpinského koberec. Postup je velmi podobný, avšak místo trojúhelníku použijeme čtverec (obr. 3).

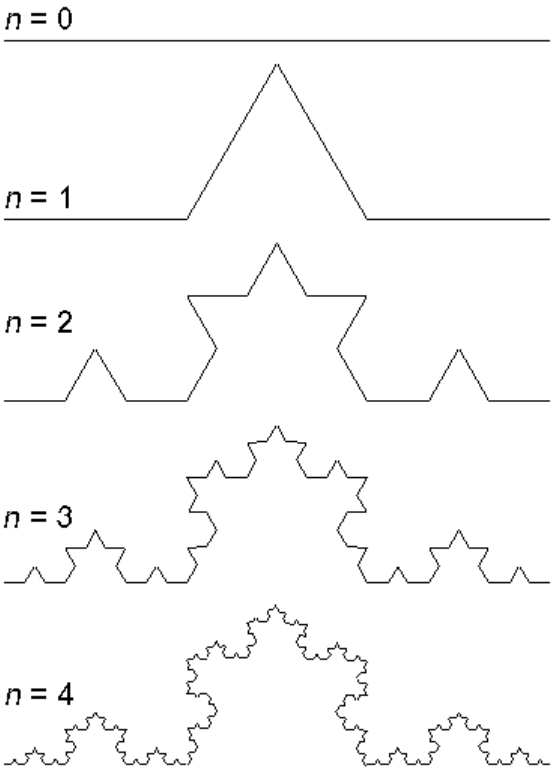


Obrázek 3 - Konstrukce Sierpinského koberce

2.3 Kochova křivka

Jako poslední příklad klasických fraktálů uvedeme Kochovu křivku, nazývanou též Kochova vločka nebo Kochův ostrov. Tato křivka má několik zajímavých vlastností, neobsahuje žádné úsečky nebo hladké segmenty, nemá derivaci v žádném bodě.

Konstrukce: začneme s úsečkou délky 1, rozdělíme ji na tři části o délce 1/3. Prostřední třetinu nahradíme rovnostranným trojúhelníkem. Stejný postup aplikujeme na všechny čtyři vzniklé úsečky. Postup znázorňuje obr. 4.



Obrázek 4 - Konstrukce Kochovy křivky

3 Fraktální dimenze

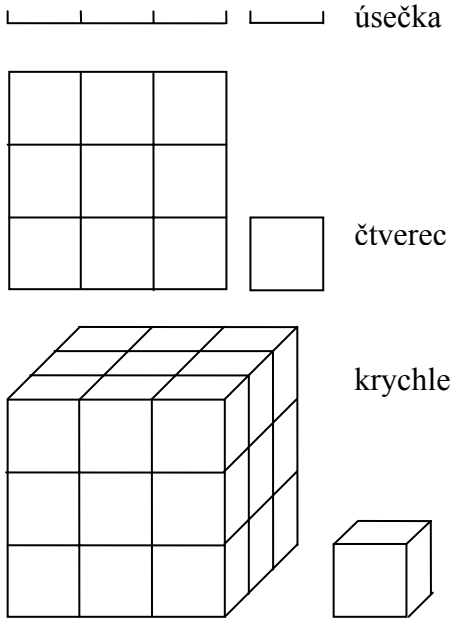
Pod pojmem dimenze si každý většinou představí počet informací (např. souřadnic) potřebných k jednoznačnému určení stavu (např. polohy) nějakého objektu. Křivky mají dimenzi 1, neboť stačí popisovat vzdálenost od nějakého pevného bodu. Pro určení polohy bodu na ploše je třeba dvou souřadnic, proto je dimenze plochy 2. Někdy mohou být tyto objekty (křivky, plochy, tělesa) natolik složité, že je dost dobře nelze takto popsat, a proto zde zavádíme pojem tzv. fraktální dimenze.

Fraktální dimenze umožňuje popsat stupeň složitosti objektu podle toho, jak rychle roste jeho délka, obsah či objem v závislosti na velikosti měřítka, při kterém měříme. Existuje několik forem této dimenze:

- soběpodobnostní dimenze (self-similarity dimension)
- mřížková dimenze (box-counting dimension)
- Hausdorffova dimenze (Hausdorff dimension)

3.1 Soběpodobnostní dimenze

Tento typ fraktální dimenze lze aplikovat pouze na struktury, které jsou soběpodobné. Řekneme, že nějaká struktura je ryze soběpodobná, pokud ji lze rozdělit na několik částí, kde každá z těchto částí je zmenšená kopie celku.



Objekt	Počet částí	Faktor zmenšení
Úsečka	3 6 173	1/3 1/6 1/173
Čtverec	$9 = 3^2$ $36 = 6^2$ $29929 = 173^2$	1/3 1/6 1/173
Krychle	$27 = 3^3$ $216 = 6^3$ $5177717 = 173^3$	1/3 1/6 1/173
Kochova křivka	4 16 4^k	1/3 1/9 $1/3^k$

Obrázek 5 - soběpodobnost úsečky, čtverce a krychle.

Z tabulky je vidět závislost mezi počtem částí a a faktorem zmenšení s daná vztahem $a = \frac{1}{s^D}$, kde D=1 pro úsečku, D=2 pro čtverec a D=3 pro krychli. Pro Kochovu křivku D vypočítáme ze vztahu $4^k = 3^D$, tzn. $D = \frac{k \log 4}{k \log 3} = \frac{\log 4}{\log 3} \approx 1.2619$.

Nyní můžeme definovat soběpodobnostní dimenzi pro libovolný soběpodobný objekt s a částmi a faktorem zmenšení s jako $D = \frac{\log a}{\log 1/s}$.

Objekt	Faktor zmenšení *s*	Počet částí *a*	Dimenze
Cantorova množina	$\frac{1}{3^k}$	2^k	$\frac{\log 2}{\log 3} \approx 0.6309$
Sierpinského trojúhelník	$\frac{1}{2^k}$	3^k	$\frac{\log 3}{\log 2} \approx 1.5850$
Sierpinského koberec	$\frac{1}{3^k}$	8^k	$\frac{\log 8}{\log 3} \approx 1.8928$

Obrázek 6– dimenze některých základních fraktálů

3.2 Mřížková dimenze (box-counting dimension)

Narozdíl od dimenze soběpodobnostní, která je definována pouze pro ryze soběpodobné útvary, lze box-counting dimenzi aplikovat na libovolný útvar.

Zjišťování dimenze: zadaný útvar se umístí na mřížku s velikostí buněk s a spočítá se, kolik buněk obsahuje alespoň část útvaru. Tím se získá číslo N, které samozřejmě závisí na volbě s. Dále se postupně volí menší s a počítá se $N(s)$. Potom se hodnoty vynesou do log/log diagramu (přesněji do $\log(N(s))/\log(1/s)$ digramu) a získané body se aproximují přímkou. Box-counting dimenze je směrnice této přímky.

Z praktického hlediska je vhodné volit mřížku dvakrát hustší než v předchozím kroku. Tím získáme posloupnost $N(2^{-k})$, $k = 0,1,2,...$. Směrnice z log/log diagramu je potom dána vzorcem

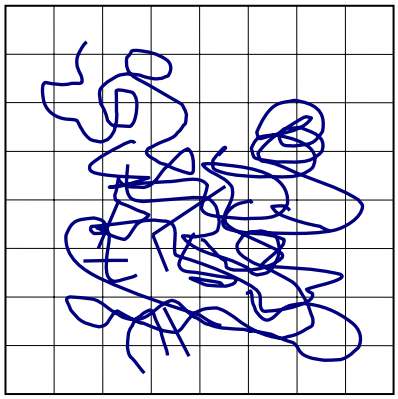
$$\frac{\log N(2^{-(k+1)}) - \log N(2^{-k})}{\log 2^{k+1} - \log 2^{k}},$$

který při volbě logaritmu o základu 2 přejde na

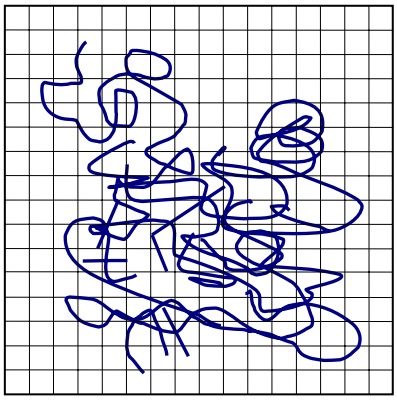
$$\log_2 \frac{N(2^{-(k+1)})}{N(2^{-k})}.$$

Takto získáme směrnici vždy mezi dvěma měřeními. Pokud se počet započítaných buněk $N(s)$ mezi dvěma po sobě jdoucími měřeními znásobí číslem 2^D, kde D je stále stejné, pak box-counting dimenze je D.

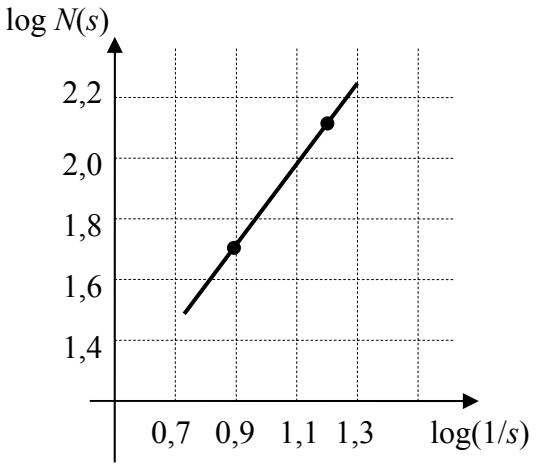
Příklad výpočtu box-counting dimenze:



$s = 1/8$ $N(s) = 48$



$s = 1/16$ $N(s) = 140$



Směrnice přímky je tedy:
$\frac{\log 140 - \log 48}{\log 16 - \log 8} \cong \frac{0.465}{0.3} = 1.55$,
což odpovídá box-counting dimenzi.

Obrázek 7 – Příklad výpočtu box-counting dimenze.

Výhodou box-counting dimenze je její snadná realizace na výpočetní technice, lze ji použít na útvary bez soběpodobnosti. Pro většinu soběpodobných objektů dává stejné hodnoty jako soběpodobnostní dimenze, nemůže však překročit hodnotu 2.

3.3 Hausdorffova dimenze

Pro definici této dimenze se omezíme pouze na množiny A, které jsou částí euklidovského prostoru

$$\mathbf{R}^n = \{ x \mid x = (x_1, \ldots, x_n),\ x_i \in \mathbf{R} \}$$

pro n přirozené. Dále je třeba zavést vzdálenost dvou bodů

$$d(x,y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \ .$$

Definujme ještě diametr množiny A jako

$$\mathrm{diam}A = \sup\{d(x,y) \mid x, y \in A\} .$$

Jako poslední pojem zavedeme spočetné otevřené pokrytí množiny $A \subset \mathbf{R}^n$. Spočetný systém $\{U_1, U_2, U_3, \ldots\}$ otevřených podmnožin $\mathbf{R}^n$ nazveme otevřené spočetné pokrytí množiny A, pokud $A \subset \bigcup_{i=1}^{\infty} U_i$.

Definice Hausdorffovy dimenze: Nechť s a ε jsou reálná kladná čísla. Potom definujeme

$$h_\varepsilon^s(A) = \inf\left\{ \sum_{i=1}^{\infty} \mathrm{diam}(U_i)^s \mid \{U_1, U_2, \ldots\} \text{ je otevřené pokrytí } A, \forall i \in \mathbb{N}(\mathrm{diam}(U_i) < \varepsilon \right\}.$$

Se snižujícím se ε se množina možných pokrytí zužuje, a proto se infimum zvyšuje. To nám dovoluje psát

$$h^s(A) = \lim_{\varepsilon \to 0} h_\varepsilon^s(A) \ .$$

Toto číslo se nazývá s-dimenzionální Hausdorffova míra množiny A. Hausdorff dokázal, že existuje číslo $D_H(A)$ takové, že

$$h^s(A) = \begin{cases} \infty \text{ pro } s < D_H(A) \\ 0 \text{ pro } s > D_H(A) \end{cases}$$

Toto číslo $D_H(A)$ se nazývá Hausdorffova dimenze množiny A. Lze ho získat pomocí předpisu

$$D_H(A) = \inf\{s \mid h^s(A) = 0\} = \sup\{s \mid h^s(A) = \infty\}$$

Příklady výpočtu Hausdorffovy míry $\overline{H}^s_\varepsilon(M)$

1. Přímka P jednotkové délky

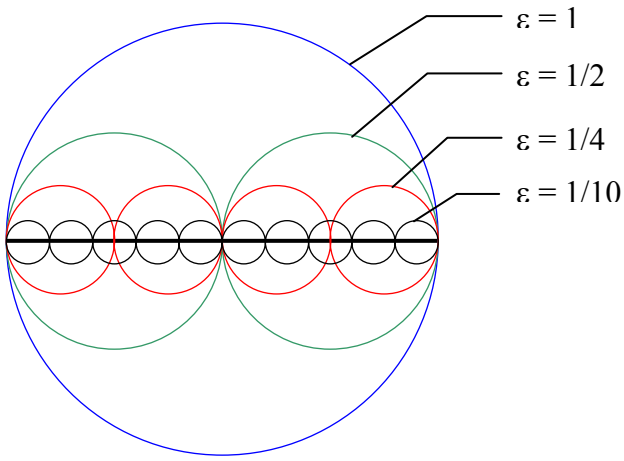
ε \ s	$\frac{1}{2}$	1	2	3
1	1	1	1	1
$\frac{1}{2}$	$\sqrt{2}$	1	$\frac{1}{2}$	$\frac{1}{4}$
$\frac{1}{4}$	2	1	$\frac{1}{4}$	$\frac{1}{16}$
$\frac{1}{10}$	$\sqrt{10}$	1	$\frac{1}{10}$	$\frac{1}{100}$

$\varepsilon=1: \quad \overline{H}^s_\varepsilon(P) = 1^s$

$\varepsilon=1/2: \quad \overline{H}^s_\varepsilon(P) = 2\cdot\left(\frac{1}{2}\right)^s$

$\varepsilon=1/4: \quad \overline{H}^s_\varepsilon(P) = 4\cdot\left(\frac{1}{4}\right)^s$

Obrázek 8 – Hausdorffova míra křivky



Obrázek 9 – Různá pokrytí přímky

2. Čtverec S jednotkového obsahu

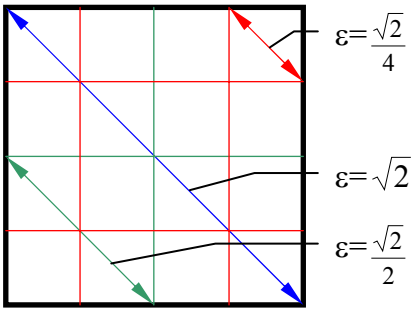
ε \ s	$\frac{1}{2}$	1	2	3
$\sqrt{2}$	$\sqrt[4]{2}$	$\sqrt{2}$	2	$2^{3/2}$
$\frac{\sqrt{2}}{2}$	$\frac{4}{\sqrt[4]{2}}$	$2\sqrt{2}$	2	$\frac{2^{3/2}}{2}$
$\frac{\sqrt{2}}{4}$	$8\cdot\sqrt[4]{2}$	$4\sqrt{2}$	2	$\frac{2^{3/2}}{4}$
$\frac{\sqrt{2}}{10}$	$\frac{100\cdot\sqrt[4]{2}}{\sqrt{10}}$	$10\sqrt{2}$	2	$\frac{2^{3/2}}{10}$

$\varepsilon=\sqrt{2}: \quad \overline{H}^s_\varepsilon(S) = \left(\sqrt{2}\right)^s$

$\varepsilon=\frac{\sqrt{2}}{2}: \quad \overline{H}^s_\varepsilon(S) = 4\cdot\left(\frac{\sqrt{2}}{2}\right)^s$

$\varepsilon=\frac{\sqrt{2}}{4}: \quad \overline{H}^s_\varepsilon(S) = 16\cdot\left(\frac{\sqrt{2}}{4}\right)^s$

Obrázek 10 – Hausdorffova míra čtverce



Obrázek 11 – Pokrytí čtverce

3. Kochova křivka K

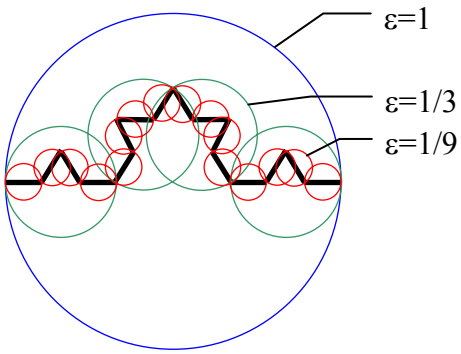
ε \ s	$\frac{1}{2}$	1	2	3	$\frac{\ln 4}{\ln 3}$
1	1	1	1	1	1
$\frac{1}{3}$	$\frac{4\sqrt{3}}{3}$	$\frac{4}{3}$	$\frac{4}{9}$	$\frac{4}{27}$	1
$\frac{1}{9}$	$\frac{16}{3}$	$\frac{16}{9}$	$\frac{16}{81}$	$\frac{16}{729}$	1

$\varepsilon=1:\quad \overline{H}_{\varepsilon}^{s}(K)=(1)^{s}$

$\varepsilon=1/3:\quad \overline{H}_{\varepsilon}^{s}(S)=4\cdot\left(\frac{1}{3}\right)^{s}$

$\varepsilon=1/9:\quad \overline{H}_{\varepsilon}^{s}(S)=16\cdot\left(\frac{1}{9}\right)^{s}$

Obrázek 12 – Hausdorffova míra Kochovy křivky



Obrázek 13 – Pokrytí Kochovy křivky

4 Juliovy množiny (Julia sets)

4.1 Definice množiny

Juliovy množiny jsou vytvářeny pomocí iterace funkce komplexní paraboly:

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

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.

Juliova množina J je definována jako množina všech komplexních čísel z_0, pro které posloupnost z_n nediverguje, tzn.:

$$J = \left\{ z_0 \in C \mid \lim_{n\to\infty} z_n \neq \infty \right\},$$

kde z_n je samozřejmě posloupnost $z_{n+1} = z_n^2 + c$.

4.2 Prahový poloměr divergence

Při počítačovém generování Juliových množin máme k dispozici pouze konečný počet iterací, z kterých musíme usoudit, zda posloupnost z_n konverguje či nikoliv. Existuje číslo $r(c)$ závislé na konstantě c takové, že pokud pro nějaké $n \in \mathbf{N}_0$ je $|z_n| > r(c)$, pak posloupnost diverguje. Platí, že

$$r(c) = \max\{|c|, 2\}.$$

Důkaz:

Předpokládejme tedy, že $|z| \geq |c|$ a $|z| > 2$. Potom existuje $\varepsilon > 0$ tak, že $|z| = |2 + \varepsilon|$. Dále využijeme trojúhelníkové nerovnosti pro komplexní čísla :

$$|z^2| = |z^2 + c - c| \leq |z^2 + c| + |c|.$$

Po převedení $|c|$ na druhou stranu získáme $|z^2 + c| = |z^2| - |c|$, což lze dále upravit

$$|z^2 + c| \geq |z^2| - |c| = |z|^2 - |c| \geq |z|^2 - |z| = |z|(|z| - 1) = (1 + \varepsilon) \geq |z|.$$

Z toho plyne, že při splnění předpokladů, vzroste v každé iteraci hodnota alespoň o faktor $1+\varepsilon$. V k-té iteraci tedy alespoň o $(1+\varepsilon)^k$ a z toho plyne, že absolutní hodnota jde k nekonečnu.

4.3 Vykreslení Juliových množin

Nejprve je třeba zvolit obdélníkovou část komplexní roviny, nejlépe zadanou dvěma protilehlými rohovými body. Tento obdélník se musí přepočítat na souřadnice obrazovky. Při přepočtu souřadnic musíme dát pozor na to, že v matematice je kladný směr imaginární osy směrem nahoru, kdežto ve výpočetní technice je to opačně.

Pro každý bod na obrazovce zjistíme odpovídající komplexní hodnotu z_0. Pro tuto hodnotu a pro hodnotu c rozhodneme, zda posloupnost z_n diverguje či nikoliv. To znamená, že postupně počítáme z_n a jakmile $|z_n|$ překročí hodnotu $\max\{2, |c|\}$, pak posloupnost diverguje. Pokud po určitém počtu iterací (zadaných uživatelem) $|z_n|$ nepřekročí hodnotu $\max\{2, |c|\}$, pak rozhodneme, že posloupnost konverguje. Pokud posloupnost diverguje, bod odpovídající

hodnotě z_0 neleží uvnitř Juliovy množiny. Pokud naopak posloupnost po zadaném počtu iterací nediverguje, prohlásíme počítaný bod za prvek dané Juliovy množiny.

Algoritmus pro zjištění, zda bod leží uvnitř Juliovy množiny, lze zapsat následovně:

nastav iter:=0
nastav z:=pozice_bodu_v_komplexní_rovině
pokud iter<MaxIter prováděj smyčku:
  nastav $z:=z^2+c$
  jestliže |z|>2 bod neleží v Juliově množině;konec;
  nastav iter:=iter+1
konec smyčky
bod leží uvnitř Juliovy množiny;
konec

Tento algoritmus lze velmi jednoduše implementovat s tím, že proměnné *z* a *c* jsou komplexní čísla. Protože ve většině programovacích jazyků nejsou komplexní čísla zavedena jako základní datové typy, pomůžeme si tím, že každé komplexní číslo reprezentujeme jeho reálnou a imaginární složkou. Proto např. *z* bude reprezentováno dvojicí *zx* a *zy* a *c* bude reprezentováno dvojicí *cx* a *cy*.

Absolutní hodnotu |*z*| lze rozepsat jako sqrt(*zx***zx*+*zy***zy*), kde sqrt() je funkce odmocniny. V reálných programech není použit přímo výpočet absolutní hodnoty, protože vyčíslení odmocniny je časově velmi náročné. Podmínku sqrt(*zx***zx*+*zy***zy*)>2 lze umocnit a použít novou podmínku, kde není potřeba provádět výpočet odmocniny: *zx***zx*+*zy***zy*>4. Hodnoty *zx***zx* a *zy***zy* je vhodné předpočítat do nových proměnných, protože se v jednou iteračním kroku používají na dvou místech a není nutné počítat stejný výraz dvakrát.

Výraz $z:=z^2+c$ lze rozepsat na reálnou a imaginární část:

zx'=*zx***zx*-*zy***zy*+*cx*
zy'=2**zx***zy*+*cy*

Aby nebylo nutné používat dvou nových proměnných *zx*' a *zy*', použijí se již předpočítané mocniny reálné a imaginární složky *z*. Také je vhodné prohodit oba výrazy, aby nebylo nutné zavádět nové pomocné proměnné:

*zx*2=*zx***zx*
*zy*2=*zy***zy*
zy=2**zx***zy*+*cy*
zx=*zx*2-*zy*2+*cx*

Program můžeme tedy napsat např. v jazyce C:

```
void CalcJulia (int    width,int height,      // velikost bitmapy
                double xmin,double xmax,     // mezni pozice v komplexni rovine
                double ymin,double ymax,
                double cx,  double cy,       // pocatecni pozice v komplexni rovine
                int    maxiter, byte* data)  // max. pocet iteraci a vysledna bitmapa int x,y;
{
    int         i,j,iter;                    // pocitadla smycek
    double      zx,zy,zx2,zy2;               // komplexni promenna "z"
    double      x,y;                         // pozice v komplexni rovine
    byte*       p=data;                      // ukazatel na zapisovany pixel
    double      max_r;                       // polomer pro divergenci

    if (cx*cx+cy*cy>4) max_r=cx*cx+cy*cy;    // max_r = max{|c|,2}
    else max_r = 4.0;

    y = ymin;
    for (j=0; j<height; j++) {               // pro vsechny radky v bitmape
        x=xmin;
```

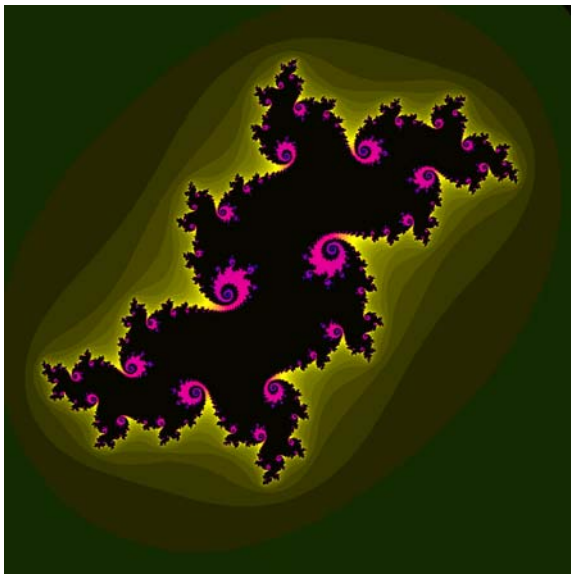
```
        for (i=0; i<width; i++) {           // pro vsechny sloupce v bitmape
            zx=x;  zy=y;                    // nastavit komplexni promennou "z"
            iter=0;                         // vynulovat pocet iteraci


            do {                                    // iteracni smycka
                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++;                             // zvysit pocet iteraci
            } while (iter<maxiter && (zx2+zy2)<max_r);     // test na poc. iteraci a bailout

            if (iter==maxiter) {                    // bod je uvnitr mnoziny
                *p=0;p++;                           // -> cerny pixel
                *p=0;p++;
                *p=0;p++;                           // kazdy pixel je ulozen ve 3 bytech - RGB
            }
            else {                                  // bod je vne mnoziny
                *p=(byte)(iter);p++;                // tzn, je to obarveny pixel
                *p=(byte)(iter);p++;                // zde se pixel obarvi odstinem sede
                *p=(byte)(iter);p++;
            }
            x+=(xmax-xmin)/width;                   // dalsi bod v komplexni rovine
        }
        y+=(ymax-ymin)/height;
    }
}
```

4.4 Typy Juliových množin

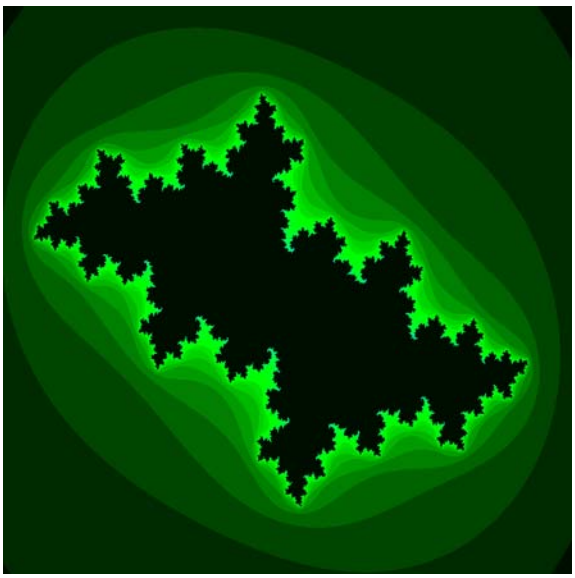
Juliovy množiny lze rozdělit na dva druhy, na souvislé a na nesouvislé. Množinu nazveme souvislou právě tehdy, když ji nelze rozdělit na dvě disjunktní otevřené množiny. Jednoduše řečeno, když je množina jako jeden kus, pak je souvislá. Souvislost množiny nám dále poslouží pro definici Mandelbrotovy množiny.

Na následujících obrázcích jsou příklady Juliových množin vygenerovaných v mém programu. Jsou ověřeny pomocí programu WinFract verze 18.21.

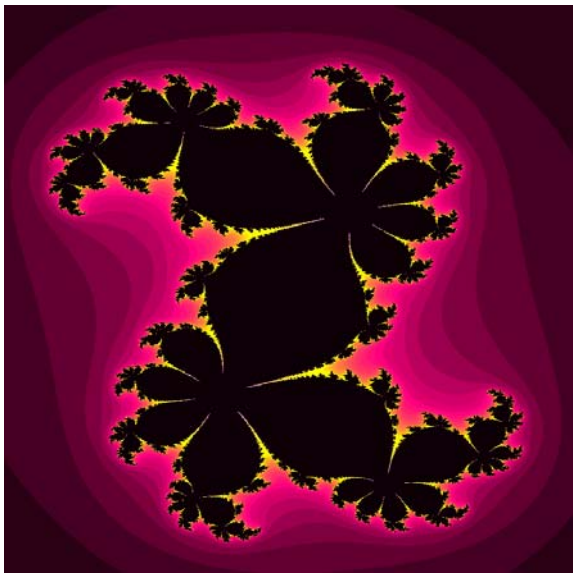
Juliova množina je vždy zobrazena černou barvou, ostatní barvy určují, kolik iterací je třeba vypočítat, abychom rozhodli, zda posloupnost jde k nekonečnu. Čím světlejší barva, tím je třeba více iterací.



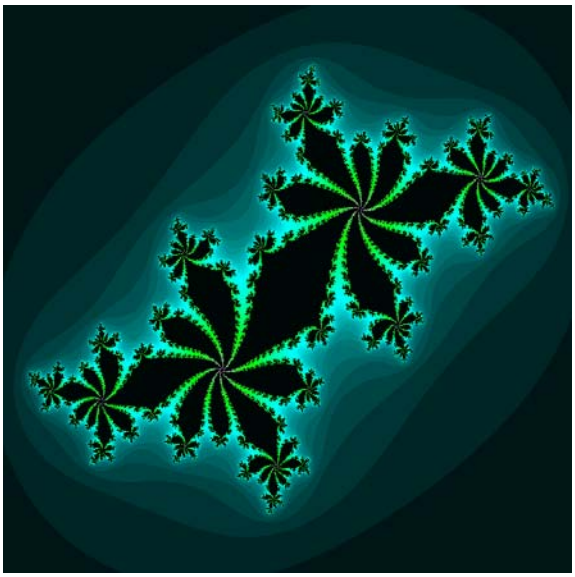
a) Souvislá; c= -0.097 -0.64872i



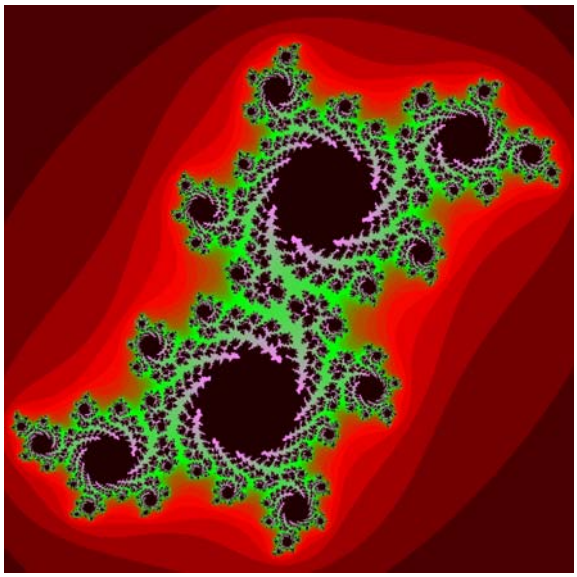
b) Souvislá; c= -0.504 +0.501968504i



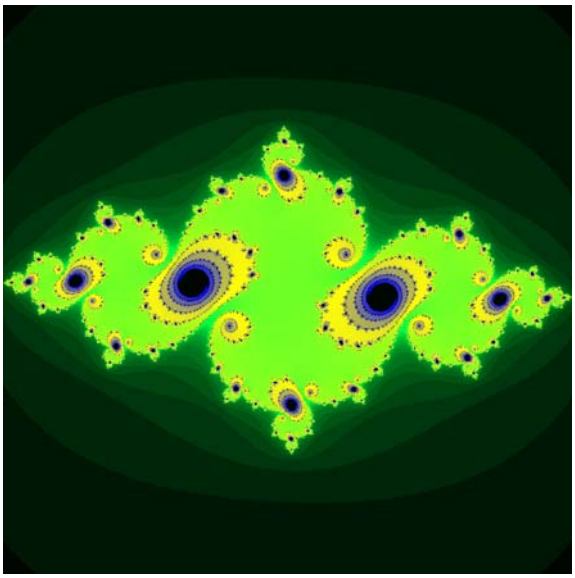
c) Souvislá; c= 0.37688 +0.216457933i



d) Souvislá; c= -0.3555 -0.62007874i



e) Nesouvislá; c= 0.060124-0.624228873i



f) Nesouvislá; c= -0.7515 -0.072i

Obrázek 14 – Příklady Juliových množin

5 Mandelbrotova množina

5.1 Definice

Mandelbrotova množina je narozdíl od Juliových množin jen jedna. Je opět vytvářena pomocí iterace funkce komplexní paraboly:

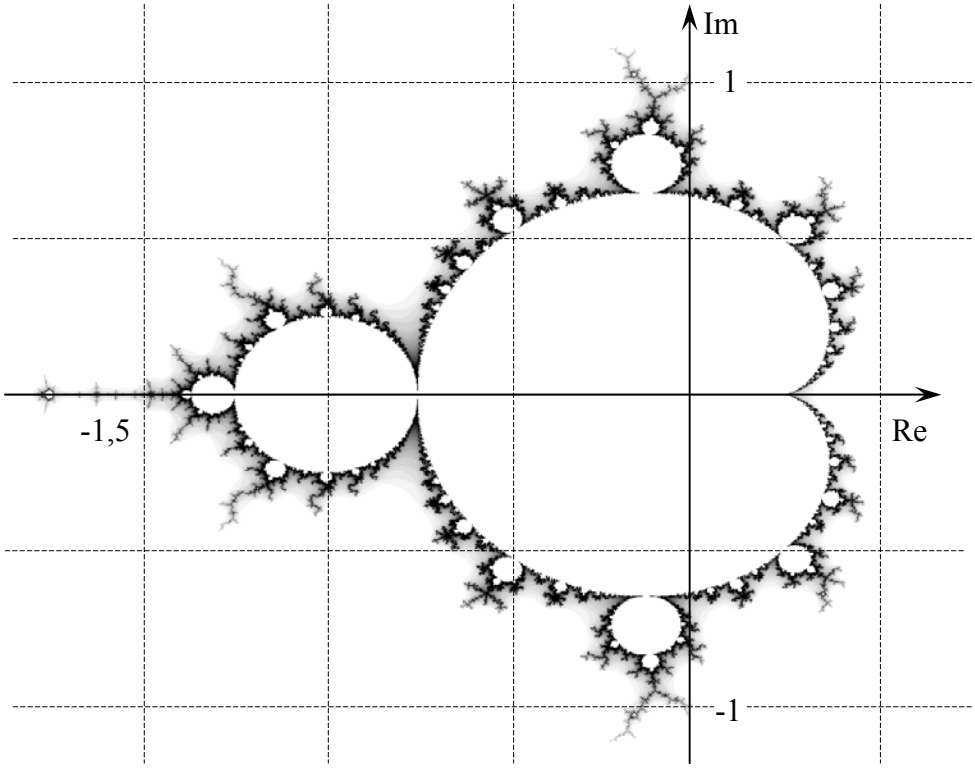
$$z_{n+1} = z_n^2 + c$$

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

$$M = \left\{ c \in C \mid c \to c^2 + c \to \cdots \text{ je omezená} \right\},$$

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

$$M = \left\{ c \in C \mid J_c \text{ je souvislá} \right\}$$

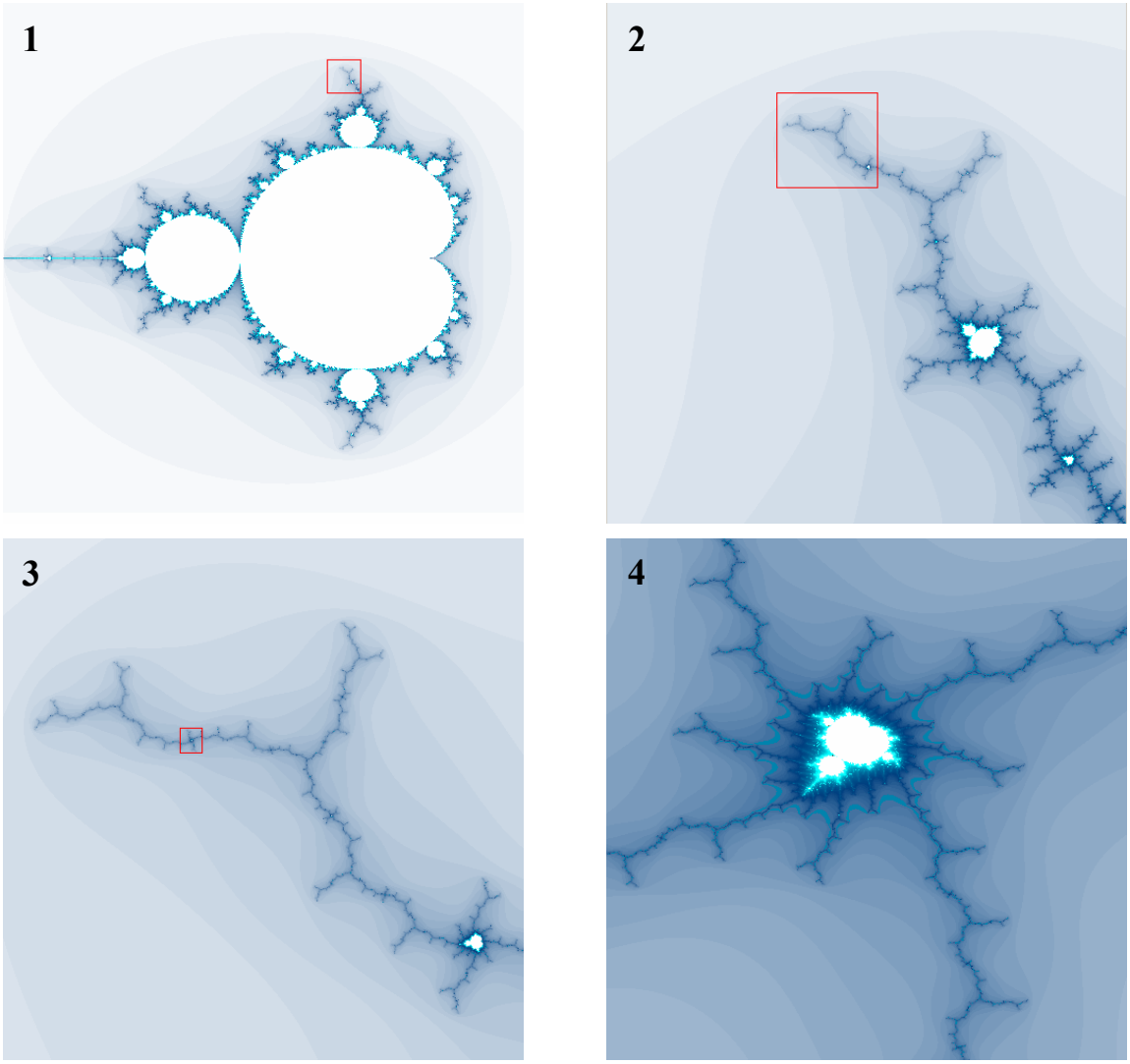


Obrázek 15 - Mandelbrotova množina

5.2 Vlastnosti Mandelbrotovy množiny

Mandelbrotova množina je souvislá. Toto tvrzení dokázali roku 1982 A. Douady a J. H. Hubbard.

Při zkoumání Mandelbrotovy množiny zjistíme, že po obvodu jsou množiny stejného tvaru. Tyto množiny nejsou pouze po obvodu, ale i „osamoceny“ v okolí. Ovšem vždy jsou spojeny s hlavní částí, M je souvislá.



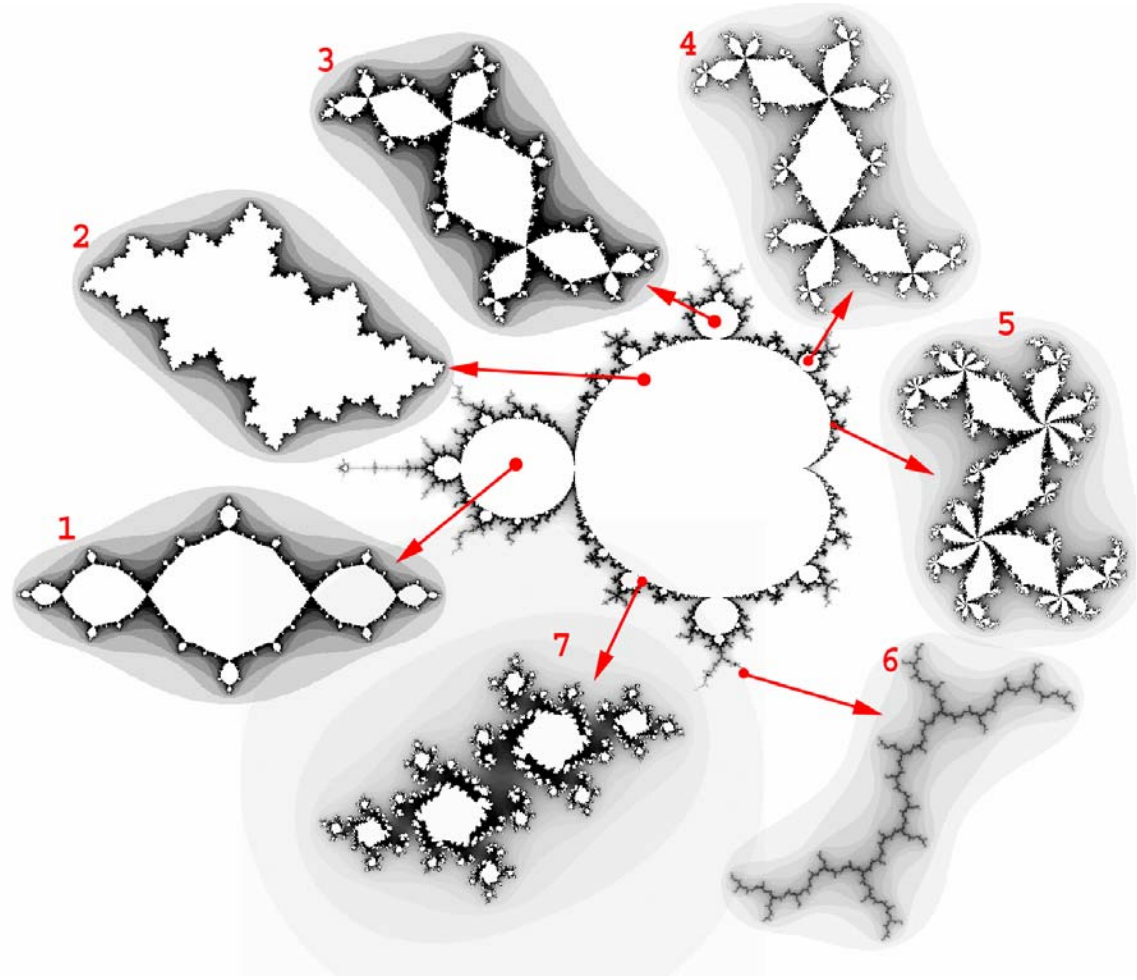
Obrázek 16 – Zvětšení množiny *M*. Množina obsahuje kopie sama sebe. Úhlopříčka posledního obrázku má velikost 0,00112.

Mandelbrotova množina má úzký vztah k množinám Juliovým. V definici bylo zmíněno, že pro bod z množiny M je odpovídající Juliova množina souvislá. Čím blíže bude bod c hranici M, tím nepravidelnější bude množina J_c. Zajímavé je také vybrat body z menších podmnožin po obvodu (viz obr. 17).

5.3 Prahový poloměr divergence

Stejně jako v případě Juliových množin, musíme i teď z konečného počtu členů posloupnosti rozhodnout, zda konverguje či diverguje. Pro Mandelbrotovu množinu platí, že posloupnost jde k nekonečnu právě tehdy, když velikost nějakého členu posloupnosti překročí hodnotu 2.

Zaměřme se nyní na členy posloupnosti z_n. Pro body mimo množinu M tedy posloupnosti divergují. Pro body v množině však posloupnost může konvergovat k nějakému číslu nebo oscilovat. V následujících grafech jsou příklady posloupností pro různé body Mandelbrotovy množiny.

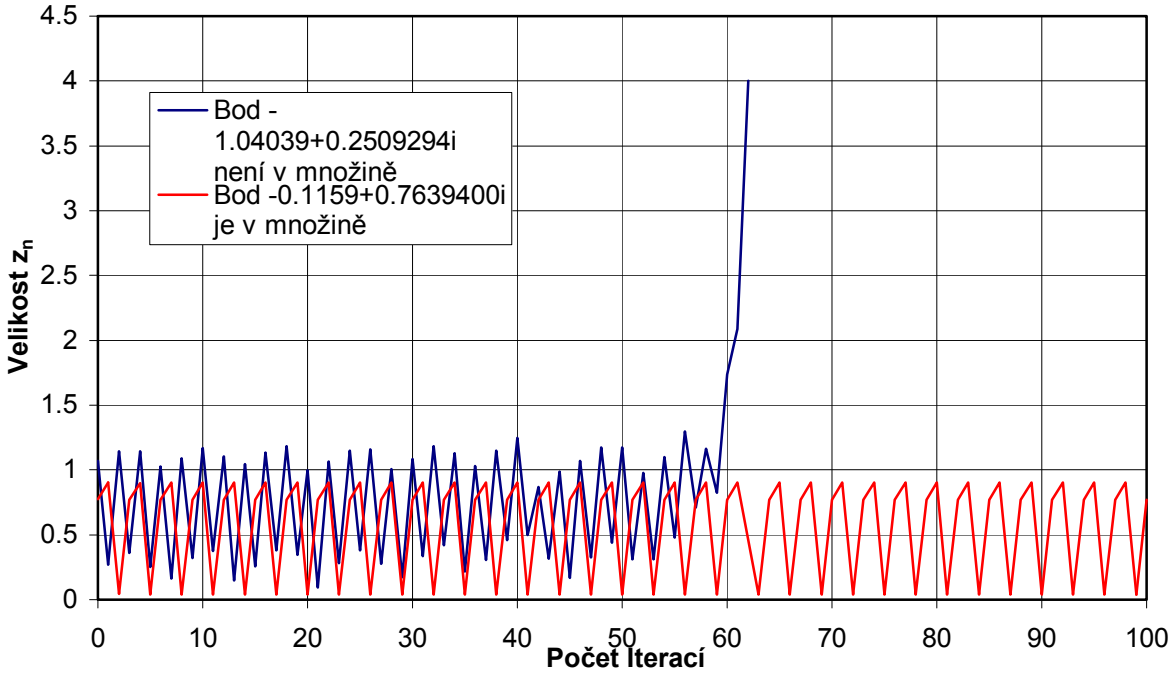


Obrázek 17 – Ukázky Juliových množin v závislosti na pozici v Mandelbrotově množině.
1) c= -1.0155+0.00590551181i; 2) c= -0.46+0.496062992i; 3) c= -0.119+0.761811024i;
4) c= 0.2935+0.537401575i; 5) c= 0.3925+0.212598425i; 6) c= 0+1i;
7) c= -0.451805+0.572113894i; (je mimo množinu)

V prvním grafu (obr. 18) jsou vyneseny absolutní hodnoty členů posloupnosti. Bod $-1.04039 + 0.2509294i$ (modrá barva) není v M, ale nachází se velmi blízko hranice, proto je třeba vypočítat relativně mnoho členů posloupnosti. Zajímavé je také to, že z prvních členů posloupnosti nelze nic usoudit o konvergenci, jelikož se absolutní hodnota chová podobně jako u bodu v množině. Avšak od jistého členu posloupnosti absolutní hodnota prudce roste.

Pro body v množině M se velmi často stává, že posloupnost osciluje. Posloupnosti vlastně konvergují k nějakému komplexnímu číslu pouze pro body, které jsou v hlavní části množiny (v „bublině“ obsažené v obdélníku -0,75 +0,66i a 0,4 -0,66i).

V následujících dvou grafech jsou body propojeny čarami jen pro názornost, z algoritmu samozřejmě získáme diskrétní hodnoty, nikoliv spojitou funkci.



Obrázek 18 – Velikost z_n pro dvě počáteční hodnoty z_0. První posloupnost diverguje, protože velikost z_n roste nade všechny meze, zatímco druhá posloupnost je omezená.

Další zajímavé informace lze získat vynesením posloupnosti z_n do grafu v komplexní rovině. V bublinách po obvodu množiny posloupnosti vždy oscilují. Čím je větší je bublina, tím méně různých bodů v posloupnosti nalezneme. Například v bublině obsahující bod $-0.12+0.75i$ tvoří posloupnost bodů přibližně trojúhelník (viz obr. 19) . Pokusme se vyřešit, pro který počáteční bod posloupnosti je to přesný trojúhelník. Musíme vyřešit rovnici, kdy se počáteční bod rovná bodu po třetí iteraci, tj. $z_0 = z_3$. Bod z_3 si přepíšeme jako $z_3 = [(z_0^2 + z_0)^2 + z_0]^2 + z_0$ a řešíme rovnici:

$$\begin{aligned} z_0 &= [(z_0^2 + z_0)^2 + z_0]^2 + z_0 \\ 0 &= [(z_0^2 + z_0)^2 + z_0]^2 \\ 0 &= z_0^2 + 2z_0^3 + 5z_0^4 + 6z_0^5 + 6z_0^6 + 4z_0^7 + z_0^8 \\ 0 &= z_0^2(1 + z_0 + 2z_0^2 + z_0^3)^2 \end{aligned}$$

Řešení jsou tedy: $z_0 = 0$,

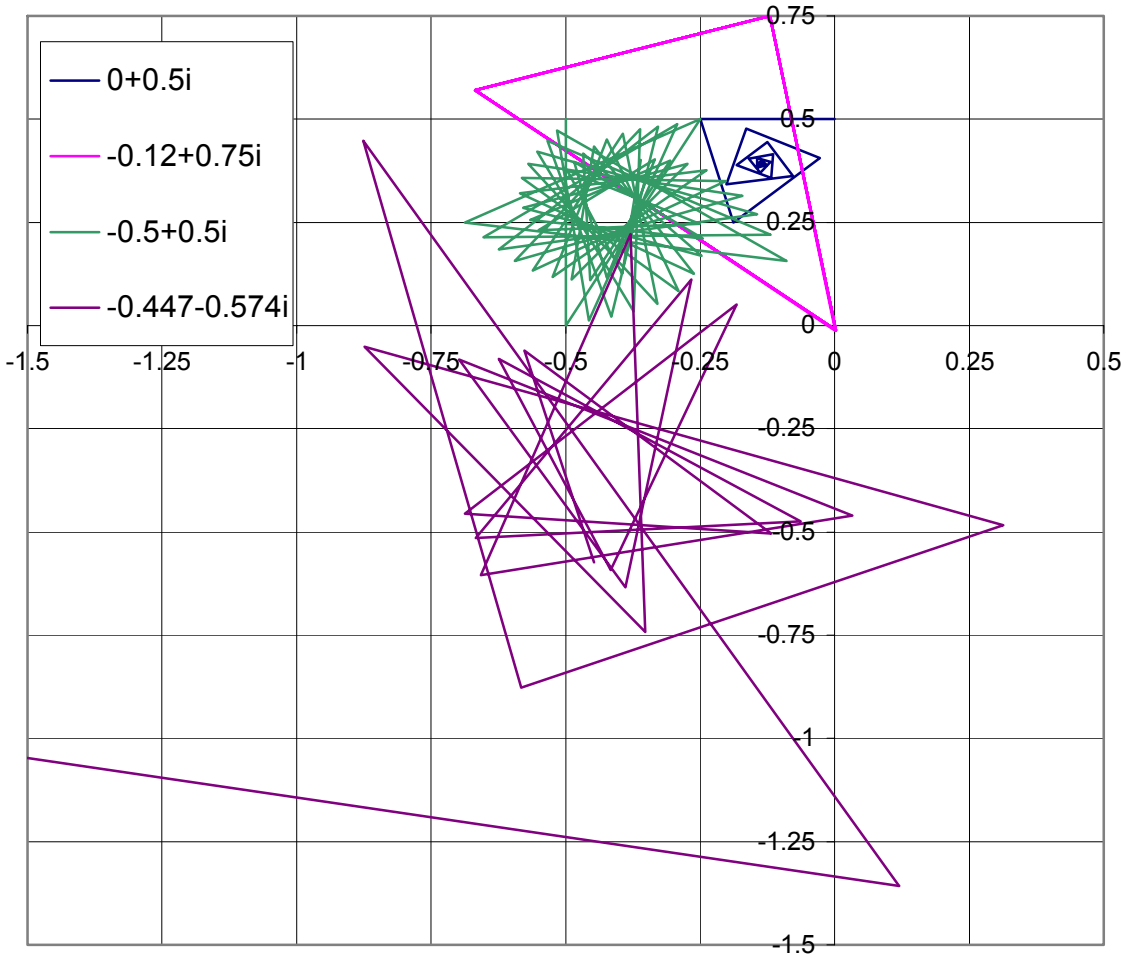
$$z_0 = \frac{1}{3}\left(-2 - \left(\frac{2}{25-3\sqrt{69}}\right)^{1/3} - \left(\frac{1}{2}\left(25-3\sqrt{69}\right)\right)^{1/3}\right) \doteq \text{-1.75487766624669},$$

$$z_0 = -\frac{2}{3} + \frac{1}{6\sqrt[3]{3}}\left(1-\sqrt{3}i\right)\left(25-3\sqrt{69}\right)^{1/3} + \frac{1+\sqrt{3}i}{3\sqrt[3]{4}\left(25-3\sqrt{69}\right)^{1/3}} \doteq \text{-0.12256116} + 0.74486176i,$$

$$z_0 = -\frac{2}{3} + \frac{1}{6\sqrt[3]{3}}\left(1+\sqrt{3}i\right)\left(25-3\sqrt{69}\right)^{1/3} + \frac{1+\sqrt{3}i}{3\sqrt[3]{4}\left(25-3\sqrt{69}\right)^{1/3}} \doteq \text{-0.12256116} - 0.74486176i.$$

Všechny kořeny jsou dvojnásobné. Dostáváme tedy osm řešení, ale pro nás jsou zajímavá hlavně poslední dvě. Ty se totiž nacházejí v bublině nahoře a symetricky dole. Pokud tedy

zvolíme pro iteraci body z bubliny obsahující některý z těchto dvou bodů, pak bude výsledný graf připomínat trojúhelník. Čím blíže bude vybraný bod námi spočítané hodnotě, tím přesnější trojúhelník dostaneme.



Obrázek 19 - Posloupnosti bodů v komplexní rovině
1) Konverguje 2) Osciluje 3) Velmi pomalu konverguje 4) Diverguje

5.4 Vykreslování Mandelbrotovy množiny

Postup při výpočtu bodů ležících v Mandelbrotově množině lze zapsat pomocí jednoduchého algoritmu. Vstupem algoritmu je komplexní číslo c a konstanta určující maximální počet iterací MaxIter.

nastav iter:=0
nastav z:=0
pokud iter<MaxIter prováděj smyčku:
  nastav $z:=z^2+c$
  jestliže $|z|>2$ *bod neleží v Mandelbrotově množině; konec*
  nastav iter:=iter+1
konec smyčky
bod leží uvnitř Mandelbrotovy množiny; konec

Opět příklad v jazyce C:

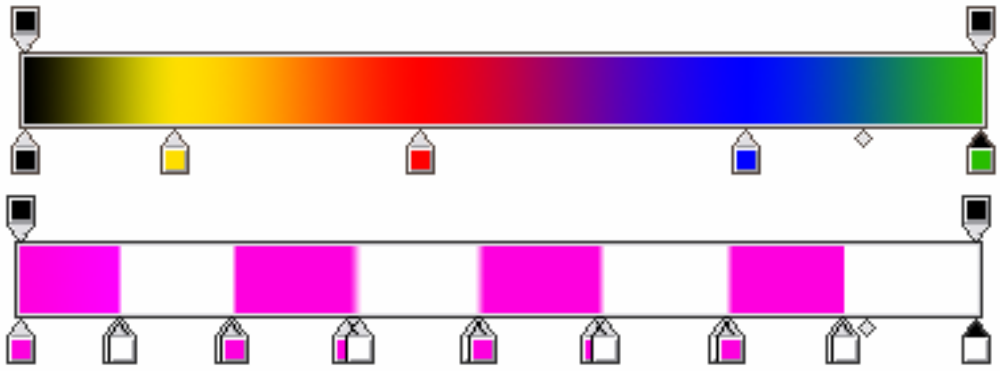
```
int MandelbrotTest(double cx, double cy)
{
    double      zx,zy,zx2,zy2;                  // komplexni promenna "z"
    int         iter;                           // pocet iteraci

    zx=0;                                       // vynulovat komplexni promennou "c"
    zy=0;

    iter=0;                                     // vynulovat pocitadlo iteraci

    do {                                        // iteracni smycka
        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++;                                 // zvysit pocet iteraci
    } while (iter<maxiter && (zx2+zy2)<4.0);    // test na poc. iteraci a bailout

    if (iter==maxiter)                          // bod je uvnitr Mandelbrotovy mnoziny
        return LEZI_UVNITR;
    else                                        // bod je vne Mandelbotovy mnoziny
        return LEZI_VNE;
}
```

6 Obarvovací algoritmy (Coloring Algorithms)

Každý fraktální algoritmus produkuje posloupnost hodnot z_0, z_1, z_2, ..., z_n. Při zobrazení fraktálu musíme pro každý pixel získat tuto posloupnost a z ní potom určit barvu pixelu.

Obarvovací algoritmy většinou vygenerují jednu hodnotu barvy pro každý pixel. V počítačové grafice se ovšem nejčastěji používá tří hodnot pro zadaní jedné barvy, a to hodnota červené (R), zelené (G) a modré (B), kdy smícháním vznikne barva výsledná. Z tohoto důvodu je třeba nějakým způsobem rozšířit jednu hodnotu do tří. Obvykle se vytvoří paleta barev, neboli posloupnost trojic (R_i,G_i,B_i). Hodnota, kterou získáme z algoritmu pro fraktál, potom určuje pozici v této posloupnosti. Pokud algoritmus produkuje neceločíselné hodnoty, je třeba posloupnost barev interpolovat. Volba palety barev je velmi důležitá pro výsledný vizuální dojem z obrázku. Dva stejné fraktály s různou volbou palet mohou vypadat naprosto odlišně.

Některé obarvovací algoritmy produkují diskrétní hodnoty, jiné zase spojité spektrum. Při použití algoritmu s diskrétními hodnotami jsou na výsledném obrázku vidět barevné skoky. V minulosti to nevadilo, neboť počítače byly schopny zobrazovat pouze 8-bitové barvy (tj. 256 barev) a barevné skoky by byly vidět stejně. Dnes jsou grafické karty schopné zobrazovat 24-bitové barvy (16777216 barev), a proto algoritmy produkující spojité spektrum začínají nabývat důležitosti.



Obrázek 20 - Ukázka palety barev s plynulými přechody (nahoře) a s ostrými barevnými skoky.

6.1 Únikový algoritmus (Escape-Time Algorithm)

Je jedním z nejstarších obarvovacích algoritmů. Jeho výhodou je jednoduchost implementace, proto je vhodný pro začínající vývojáře programů na fraktály. Z estetického hlediska je již překonaný, jelikož produkuje diskrétní hodnoty.

Algoritmus je založen na počtu iterací potřebných pro zjištění, zda posloupnost bodů konverguje k nekonečnu či nikoli. Generujeme tedy posloupnost z_0, z_1, z_2, ..., z_n a jakmile zjistíme, že pro nějaké z_n byla překročena hranice R, rozhodneme, že posloupnost diverguje. Nejmenší velikost a tvar oblasti R se samozřejmě liší pro různé fraktály. Délka posloupnosti, tedy číslo n, je výsledná hodnota barvy. Tato hodnota se v implementaci získá velmi lehce, neboť počet iterací se musí stejně počítat, aby program nezůstal ve smyčce při výpočtu jednoho bodu.

Obvykle se oblast R definuje jako kruh o poloměru 2, jelikož se dá dokázat, že pro Mandelbrotovu množinu každá posloupnost obsahující prvek $|z| > 2$ diverguje (viz. kapitola o Mandelbrotově množině). Zajímavých výsledků se dá dosáhnout změnou R například na

elipsu, trojúhelník, čtverec, hvězdu nebo jiný tvar. Z matematického hlediska musí oblast vždy obsahovat kruh s poloměrem 2, aby se dalo hovořit o konvergenci či divergenci.

Únikový algoritmus může být považován za neeuklidovskou vzdálenost od libovolného bodu z_0 k hranici množiny. Diskrétní hodnoty způsobí, že výsledný obrázek obsahuje barevné plochy podobné těm, které jsou na topografických mapách. Toho lze využít pro vygenerování zajímavých obrázků vhodnou volbou palety barev. Například takzvané tygří pruhy, kdy v posloupnosti barev jsou výrazné skoky.

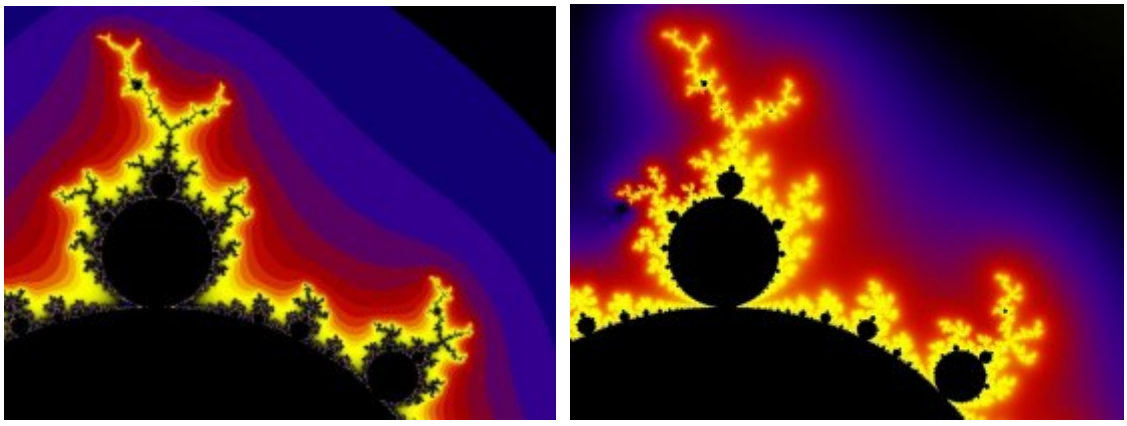
6.2 Odhad vzdálenosti (Distance Estimator)

Nyní se pokusíme rozšířit únikový algoritmus tak, aby vzdálenost od z_0 k hranici R byla spojitá funkce. Následující algoritmy nedávají přesnou euklidovskou vzdálenost, ale přijatelné spojité hodnoty.

Historicky nejstarší algoritmus se jmenuje *algoritmus odhadování vzdálenosti* (*distance estimation algorithm*) a používá funkci

$$\frac{2\,|z_n|\log|z_n|}{|\overline{z_n}|}.$$

Hodnota této funkce je blízko euklidovské vzdálenosti, ale výsledné barvy se trochu odlišují od únikového algoritmu.



Obrázek 21 - Diskrétní (escape-time) a spojité (odhadování vzdálenosti) obarvovací algoritmy.

Dalším z algoritmů je *algoritmus spojitého potenciálu* (*continuous potential algorithm*). Pokud bychom fraktální obrazec nekonečně rozšířili nad a pod komplexní rovinu jako fraktálem vytvarovaný „válec“, a považovali bychom ho za vodič, který generuje elektrické pole, potom lze elektrický potenciál aproximovat funkcí

$$\frac{\log|z_n|}{2^n},$$

za předpokladu, že počet iterací n je dost velký a tím pádem i $|z_n|$ je velké.

Dalším algoritmem produkujícím spojité hodnoty je *Algoritmus normalizovaného počtu iterací* (*normalized iteration count algorithm*). Tento algoritmus zachovává rozložení barev stejné jako escape-time algoritmus, ale „zespojiťuje“ barevné skoky. Vzorec pro výpočet barvy je

$$n+1-\frac{\log(\log|z_n|)}{\log 2}.$$

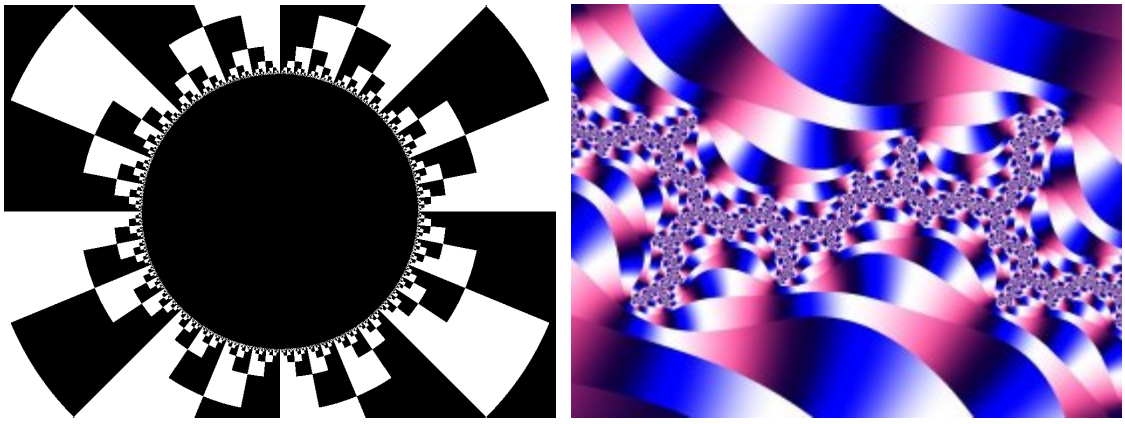
Jako poslední odhad vzdálenosti uveďme vyhlazování pomocí $e^{-|z|}$. Jednoduše se spočítá součet $e^{-|z|}$ přes všechny iterace. Se zvyšující se |z| klesá hodnota $e^{-|z|}$ a další členy mění sumu velice málo.

6.3 Únikový úhel (Escape angle)

Dosud popsané algoritmy používaly pro výpočet barvy buď počet iterací nebo velikosti čísel z_1, z_2, ..., z_n. Komplexní čísla se ovšem skládají z velikosti a úhlu. V následujících algoritmech bude právě úhel hlavní informací pro zvolení barvy.

Prvním popisovaným algoritmem je *binární dekompozice* (*binary decomposition*). Spočítáme iterace dokud nezjistíme, že posloupnost diverguje. Podle posledního členu z_n rozhodneme o barvě. Pokud je úhel z_n nad osou x (tzn. v intervalu $<0; \pi>$) přiřadíme jednu barvu a pokud je pod osou x (tzn. v intervalu $(\pi; 2\pi)$) přiřadíme barvu druhou. Binární dekompozice umožňuje přibližně zobrazit siločáry obklopující fraktál.

Algoritmus lze vylepšit například tak, že barvy přiřadíme podle kvadrantu. Pokud chceme algoritmus rozšířit na spojitý případ, můžeme úhel odečítat přímo a podle této hodnoty přiřadit barvu. Algoritmus se potom nazývá *spojitá dekompozice* (*continuous decomposition*).



Obrázek 22 - Binární a spojitá dekompozice dvou různých Juliových množin.

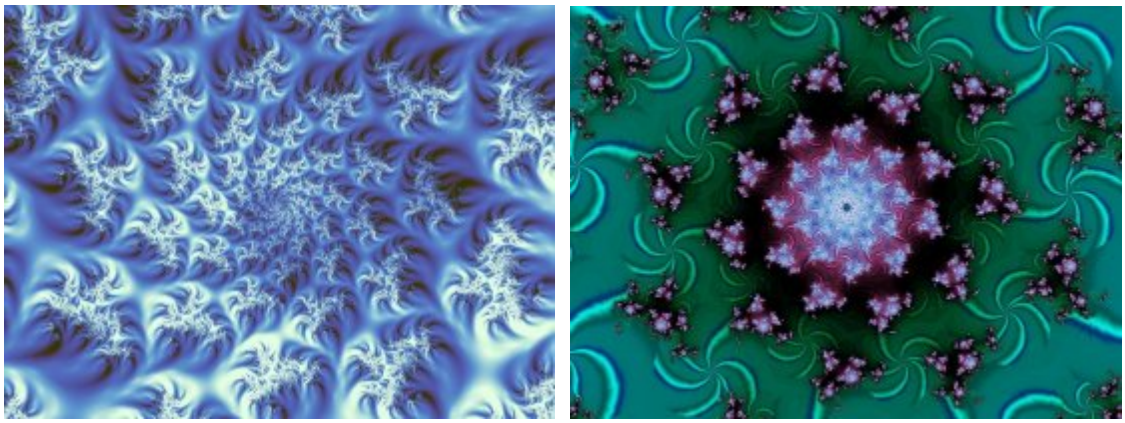
6.4 Odhad zakřivení (Curvature estimation)

Další vlastností, kterou lze měřit v posloupnosti z_1, z_2, ..., z_n, je zakřivení po sobě jdoucích iterací. Rychlý odhad lze spočítat pouze z posledních 2 iterací. Lepší výsledky lze dosáhnout tak, že provedeme průměr přes všechny iterace z výrazu

$$\left| \tan^{-1}\left[\frac{z_n - z_{n-1}}{z_{n-1} - z_{n-2}} \right] \right|.$$

6.5 Statistiky

Iterování produkuje posloupnost z_1, z_2, ..., z_n hodnot, které lze zpracovat statisticky. Můžeme použít průměr, rozptyl, směrodatnou odchylku nebo další statistické veličiny k přímému získání barvy.

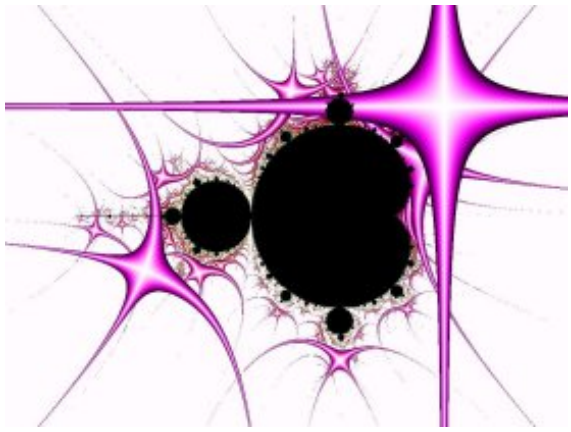


Obrázek 23 - Vpravo i vlevo je zobrazen stejný fraktál ve stejné části komplexní roviny, ovšem pokaždé s jiným obarvovacím algoritmem. Obrázek vlevo používá odhad zakřivení zatímco obrázek vpravo je výsledkem aplikace několika statistických metod.

6.6 Orbitální pasti (Orbit traps)

Toto je jedna z největších skupin obarvovacích algoritmů. Celé softwarové balíčky byly napsány pro výzkum těchto algoritmů. Idea spočívá v tom, že si zvolíme nějakou část komplexní roviny (označme ji T) a sledujeme vztah mezi hodnotami z_n a T. T je obvykle definováno jako ústřední tvar a prahová vzdálenost. Říkáme, že všechny body, které jsou blíž ústřednímu tvaru než je prahová vzdálenost, jsou uvnitř „pasti“. Tyto algoritmy lze rozdělit do několika tříd.

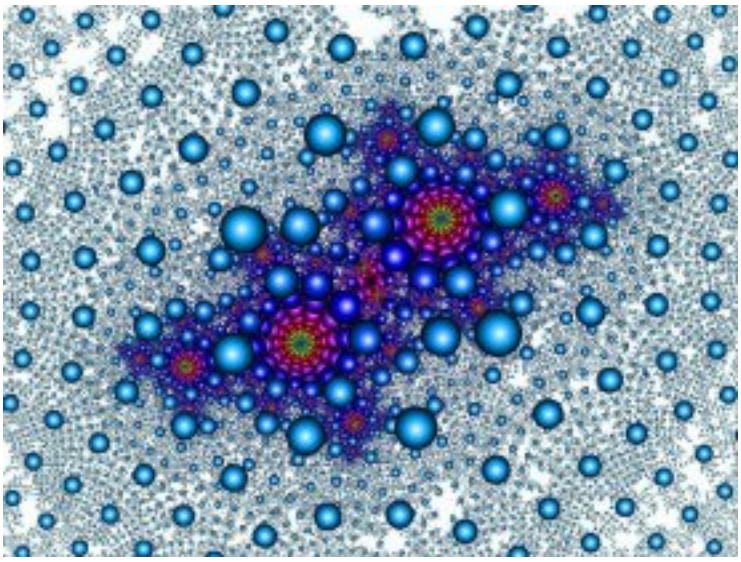
První třída se zabývá hlavně tvarem oblasti T. Můžeme generovat různé tvary od přímek, kruhů až po různé hyperboly, asteroidy atd. Jakmile se nějaké z_n ocitne uvnitř pasti, iterace skončí a bodu se přiřadí barva. Druhá třída zkoumá vztah mezi vzdáleností každého z_n a oblasti T. Podle toho potom přiřadí barvu. Existuje mnoho dalších vlastností, podle kterých rozhodneme o barvě. Můžeme použít úhel, se kterým vstoupil bod do pasti, vzdálenost posledního bodu, který se ještě v pasti nachází nebo mnoho dalších.



Obrázek 24 - Orbitální past ve tvaru kříže složeného z hyperbol.

6.7 Gaussovská celá čísla (Gaussian integer algorithm)

Gaussovské celé číslo je takové komplexní číslo, jehož reálná i imaginární část je celé číslo. Algoritmus spočívá v tom, že pro každé z_n spočítáme vzdálenost od nejbližšího gaussovkého celého čísla a podle ní přiřadíme barvu. Je to vlastně použití orbitální pasti s tvarem oblasti T rovné bodu, která je umístěna na pravidelnou mřížku v komplexní rovině. Tuto techniku můžeme rozšířit na libovolný tvar orbitální pasti a můžeme také měnit mřížku. Lze použít například trojúhelníkovou nebo radiální mřížku.

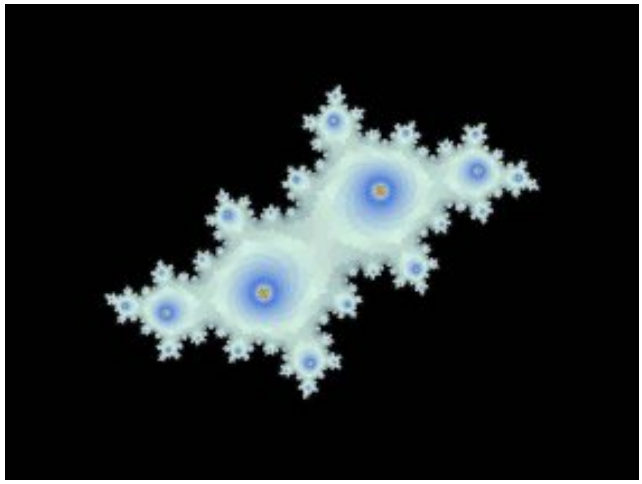


Obrázek 25 - Použití gaussovských celých čísel na Juliovu množinu.

6.8 Konečné atraktory (Finite attractors)

Ne všechny posloupnosti z_1, z_2, z_3, ..., z_n se blíží nekonečnu. U některých fraktálů je množina posloupností jdoucích k nekonečnu velmi malá. Posloupnosti, které se neblíží k nekonečnu, většinou konvergují k jednomu bodu nebo oscilují. Mnoho algoritmů lze přímo aplikovat i na konvergentní posloupnosti, některé však potřebují úpravy.

Nejjednodušší metodou je zjistit, zda změna z klesá. Pokud z_n konverguje k nějakému pevnému bodu, pak $|z_n - z_{n-1}|$ konverguje k 0. Pokud rozdíl klesne pod prahovou hodnotu, obarvíme podle toho bod vhodnou barvou. Můžeme opět použít počet iterací potřebných ke zjištění konvergence nebo jakoukoli jinou zde jmenovanou metodu.



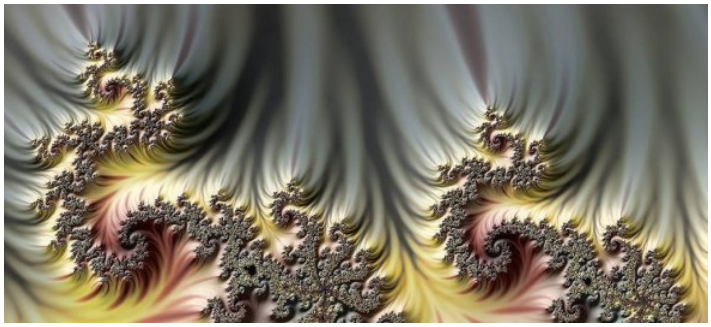
Obrázek 26 - Zde je vidět použití algoritmu pro zobrazení limitních bodů posloupností ve spojené Juliově množině.

6.9 Trojrozměrné efekty

Ačkoliv jsou fraktální obrázky většinou dvourozměrné, lze je pomocí různých technik zobrazit tak, aby měly trojrozměrný vzhled. Několik bodů blízko sebe (blíže než jeden pixel) se počítá zároveň a po skončení iterace se z nich vypočítá „výška“ bodu. Třemi body

proložíme rovinu (nyní v 3D) a spočítáme normálový vektor. Tím získáme směr roviny v okolí bodů. Podle úhlu mezi tímto vektorem a vektorem směru světa spočítáme množství světla dopadajícího na povrch.

Výškové hodnoty lze nejlépe získat použitím algoritmů pro odhad vzdálenosti.



Obrázek 27 - Mandelbrotova množina s 3D vzhledem

7 Fraktální komprese obrázků

V roce 1988 Barnsley a Sloan v [7] poprvé představili kompresi obrázků pomocí systému iterovaných funkcí , jejichž atraktor aproximuje původní obraz. A.E. Jacquin vylepšil jejich algoritmus na lokální systémy iterovaných funkcí a poprvé představil blokové fraktální kódování (Fractal Block Coding – FBC), což byla již v praxi použitelná kódovací technika. Princip spočívá v tom, že se obrázek rozdělí na segmenty o rozměrech BxB (tzv. range blocks). Potom pro každý blok hledáme kontrahující zobrazení a doménový blok (domain block). Komprese pomocí FBC je ztrátová. Její princip si nyní popíšeme.

7.1 Komprese

1) Segmentace

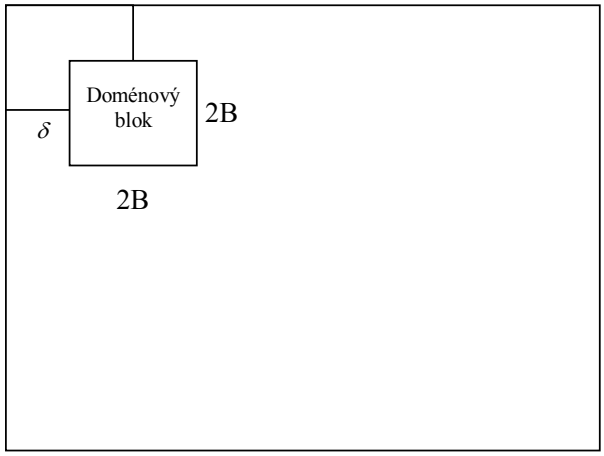
Obrázek se rovnoměrně rozdělí na segmenty o rozměrech BxB do dvourozměrného pole. Obvykle jsou rozměry obrázku $2^l \times 2^l$, např. pokud jsou rozměry obrázku 512×512 nebo 256×256, potom range bloky volíme 4×4, 8×8, 16×16, 32×32. Tyto bloky jsou zpravidla zpracovávány postupně řádek po řádku.

R_{11}	R_{12}	R_{13}	R_{14}	…	…	…	…	R_{1n}
R_{21}	R_{22}	R_{23}	R_{24}	…	…	…	…	R_{2n}
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
R_{m1}	R_{m2}	R_{m3}	R_{m4}	…	…	…	…	R_{mn}

Obrázek 28 – Rozdělení obrázku na range bloky.

2) Doménové bloky (domain blocks)

Pro každý range blok hledáme doménový blok a vhodnou transformaci tak, aby oba bloky byly po transformaci co nejvíce podobné. Vhodné doménové bloky hledáme v „zásobníku“ doménových bloků.



Obrázek 29 – Doménové bloky

Zásobník doménových bloků se skládá z bloků o rozměrech 2Bx2B z původního obrázku. Získáme ho posouváním 2Bx2B okénka po původním obrázku s krokem δ. Pokud má obrázek rozměry M x M, pak existuje právě

$$\left(\frac{M-2B}{\delta}+1\right)*\left(\frac{M-2B}{\delta}+1\right)$$

takových bloků. Například pokud je rozměr původního obrázku 256×256, range bloky jsou 4×4 a velikost kroku $\delta = 4$, potom existuje 63×63 doménových bloků.

Při kompresi porovnáváme každý range blok se všemi doménovými bloky a hledáme nejlepší dvojici. Postup hledání je popsán v následující kapitole.

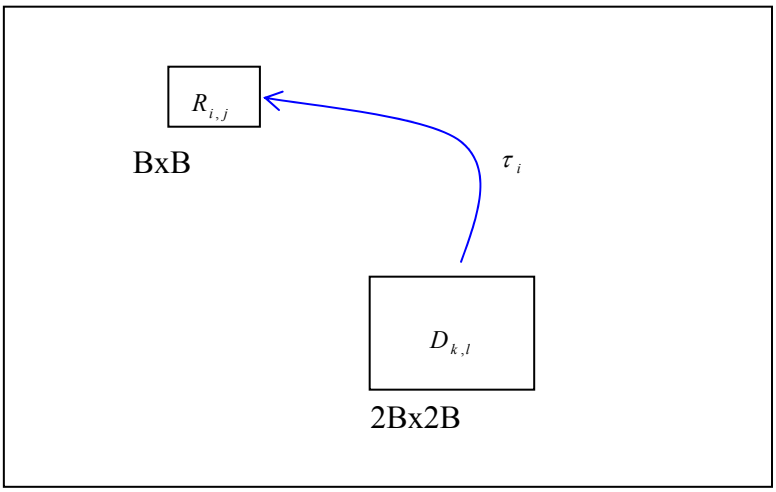
3) Afinní transformace

Afinní transformace je mapování doménového bloku na range blok. Je to složené zobrazení $\tau_j = T_j \circ S_j$ (viz. Obrázek 30),

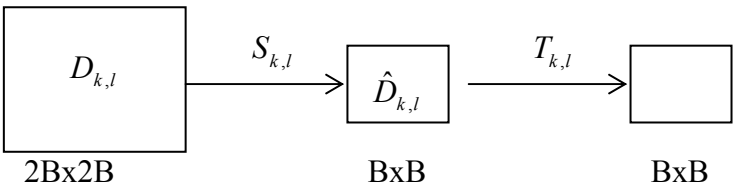
kde $S_j(\mu_{k,l}) = \mu_{2k,2l} + \mu_{2k+1,2l} + \mu_{2k,2l+1} + \mu_{2k+1,2l+1}$ je operátor průměru a $T_j(\mu_{k,l}) = s\mu_{k,l} + g$ je lineární mapa. Ve výrazu pro T je s známý faktor měřítka a je v rozmezí $0 \le s < 1$, g je posun. Doménový blok, s a g hledáme tak, aby se minimalizoval rozptyl, tedy podle vzorce

$$\min_{s,g,\hat{D}_{k,l}} \left| R_{i,j} - s\hat{D}_{k,l} - g \right|.$$

V tomto vzorci je $R_{i,j}$ zpracovávaný range blok a $\hat{D}_{k,l}$ je zmenšený doménový blok pomocí operátoru průměru S (minimum se hledá přes všechny doménové bloky). Ve FBC se používá pouze čtyř hodnot s, a to 1/4, 1/2, 3/4 a 1. Touto minimalizací nalezneme nejlepší pár a vhodné s a g.



Obrázek 30 – Mapování doménového bloku na range blok



Obrázek 31 – Afinní transformace $\tau_{k,l} = T_{k,l} \circ S_{k,l}$

a)  b) 

Obrázek 32 – a) 510x380 “Budova FJFI, Trojanova ulice”; b) 320x240 “Budova FJFI, Břehová ulice

Jako příklady vezměme obrázky budovy FJFI, viz obrázek 32. Pokud má range blok rozměry 4x4, pak lze obrázek 32b rozdělit na 80x60 range bloků. Pro krok $\delta = 4$ je zásobník doménových bloků tvořen 79×59 doménovými bloky. Pro každý range blok se nalezne nejlepší doménový blok a transformace.

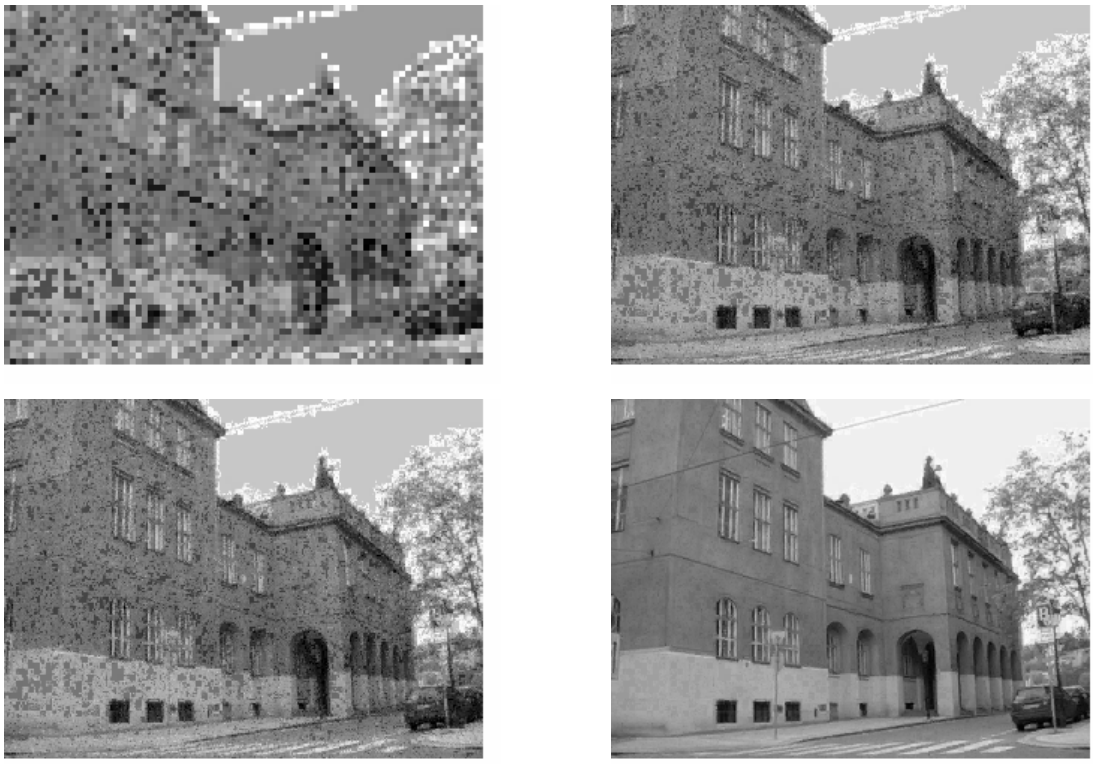
7.2 Dekomprese

Kompresní proces FBC je velice časově náročný, proto se většina výzkumníků snaží o jeho zrychlení. Naopak dekomprese je proces jednoduchý a velice rychlý. Stačí pouze iterovat získané transformace z libovolného počátečního obrázku. Pro dobrou kvalitu výsledného obrázku většinou postačí 8 iterací.

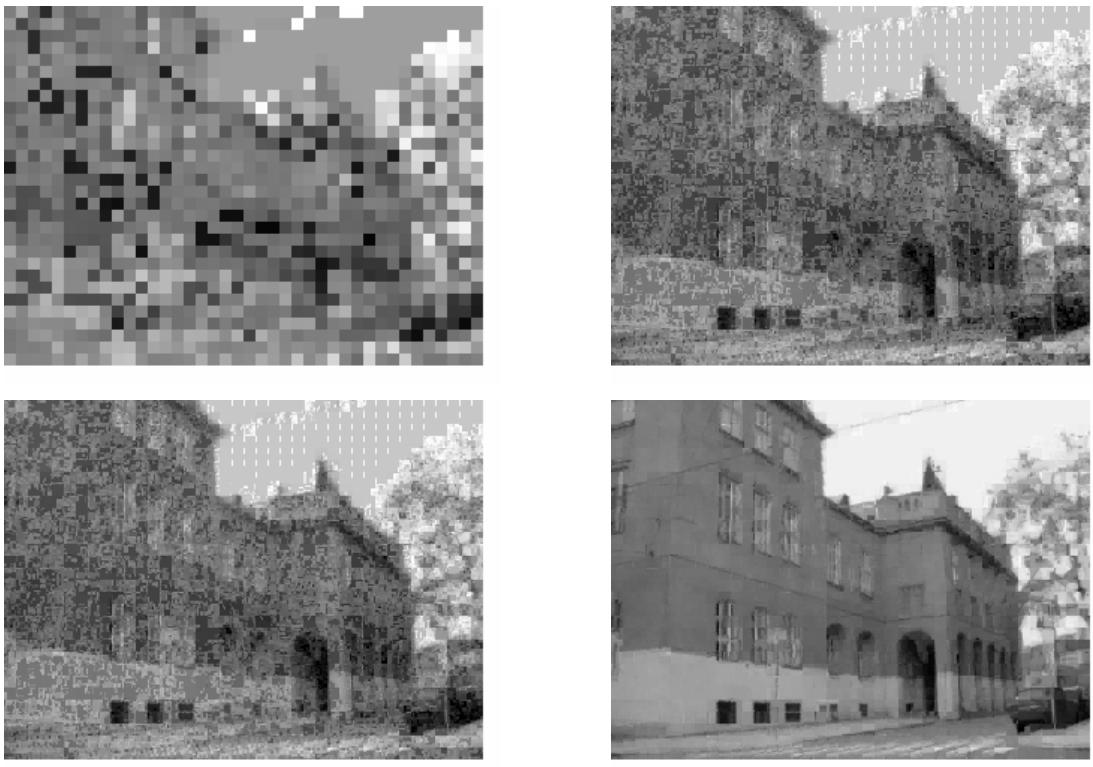
Obrázky 32a a 32b jsme zkomprimovali fraktální kompresí. Pro obrázek 32a bylo použito range bloků o velikosti 5x5 a na obrázek 32b range blok o velikosti 4x4 a 8x8. Následující obrázky ukazují výsledky dekomprese.

a) Dekomprese obrázku 32a

b) Dekomprese obrázku 32b, range blok 4x4



c) Dekomprese obrázku 32b, range blok 8x8



Obrázek 33 – První, druhá, třetí a osmá iterace při dekompresi obrázků 32a a 32b

8 Závěr

Fraktální geometrie a vše s ní související je velmi obsáhlá část matematiky, a proto bych rád pokračoval v jejím studiu i dále. Chtěl bych napsat vlastní program komprimující obrázky fraktální kompresí, program pro systémy iterovaných funkcí a vylepšit můj stávající program generující Mandelbrotovu a Juliovy množiny. Chtěl bych také fraktály popsat více matematicky než jak je tomu v této práci.

Prozatímní výsledky mé práce jsou vidět na obrázcích 14, 15, 16 a 17. Tyto obrázky jsou vytvořeny v mém vlastním programu. Tento program je napsán v Borland C++ builderu verze 5.0. Získané výsledky jsem ověřil pomocí programu WinFract 18.21. Program slouží kromě zobrazování Mandelbrotovy a Juliových množin také pro zkoumání fraktálů z matematického pohledu. Zabudoval jsem například funkce pro zobrazení komplexní posloupnosti v komplexní rovině nebo graf absolutních hodnot posloupnosti. Grafy i hodnoty posloupností lze hned ukládat do souboru a zpracovat v dalších programech. Program jsem se snažil vytvořit co nejvíce uživatelsky přátelský, ale stále je co vylepšovat.

9 Literatura

[1] Peitgen H.-O., Jurgens H., Saupe D.: "Chaos and Fractals: New Frontiers of Science", Springer-Verlag, New York, 1992

[2] Petyovský P., Tišnovský P.:
http://www.elektrorevue.cz/clanky/03019/index.htm,
http://www.elektrorevue.cz/clanky/01040/index.htm.

[3] Weisstein E. W.: "Self-Similarity",
http://mathworld.wolfram.com/Self-Similarity.html

[4] Barrallo J.: "Fractal Geometry", Anaya Multimedia, Madrid 1992.

[5] Peitgen H.-O., Richter P. H.: "The Beauty of Fractals", Springer-Verlag, Berlin Heidelberg, 1986.

[6] Hong M.: "Fractal Image Compression",
http://www.cs.ualberta.ca/~minghong/

[7] Barnsley M. F.: "Fractals Everywhere", Academic Press Inc., San Diego, 1988.