TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

FRAKTÁLNÍ GEOMETRIE V E-LEARNINGOVÉM KURZU NA ALS

Bakalářská práce

Studijní program: B2612 – Elektrotechnika a informatika
Studijní obor: 1802R022 – Informatika a logistika

Autor práce: **Jakub Hummer**
Vedoucí práce: RNDr. Klára Císařová, Ph.D.

Liberec 2015

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

THE FRACTAL GEOMETRY IN ELEARNING COURSE AT ALS

Bachelor thesis

Study programme: B2612 – Electrical Engineering and Informatics
Study branch: 1802R022 – Informatics and Logistics

Author: **Jakub Hummer**
Supervisor: RNDr. Klára Císařová, Ph.D.

Liberec 2015

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2014/2015**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jakub Hummer**
Osobní číslo:	**M12000027**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Informatika a logistika**
Název tématu:	**Fraktální geometrie v e-learningovém kurzu na ALS**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Seznamte se s pojmem fraktál, fraktální geometrie.
2. Prostudujte základní typy fraktálních struktur a jejich souvislost s procedurálním programováním.
3. Naprogramujte aplikaci, která vysvětlí vznik fraktální struktury s možností její parametrizace.
4. Vytvořte na e-learningovém portále ALS kurz z prostudovaných témat a naprogramované algoritmy budou v kurzu sloužit jako počítačový experiment.

Rozsah grafických prací:	**dle potřeby dokumentace**
Rozsah pracovní zprávy:	**30–40 stran**
Forma zpracování bakalářské práce:	**tištěná/elektronická**

Seznam odborné literatury:

[1] **Zelinka Ivan a kol.: Fraktální geometrie - principy a aplikace, Nakladatelství BEN - technická literatura, 2006.**

[2] **Zelinka Ivan: Aplikovaná informatika aneb úvod do fraktální geometrie, buněčných automatů: Vysoké učení technické v Brně, Fakulta technologická ve Zlíně, 1999.**

[3] **Gleick, J.: Chaos, vznik nové vědy. Ando publishing, 1996.**

[4] **Ian Stewart: Kabinet matematických kuriozit profesora Stewarta, Vydavatelství Dokořán, 2013.**

[5] **Radek Pelánek: Programátorská cvičebnice, Computer Press, 2012.**

[6] **Jiří Žára a kol.: Moderní počítačová grafika, Computer Press, 2005.**

[7] **http://fractals.hauner.cz/index**

[8] **http://www.ksr.tul.cz/fraktaly/obsah.html**

[9] **http://ww1.fractalartgallery.com/**

Vedoucí bakalářské práce:	**RNDr. Klára Císařová, Ph.D.** Ústav mechatroniky a technické informatiky
Datum zadání bakalářské práce:	**10. října 2014**
Termín odevzdání bakalářské práce:	**15. května 2015**

V. Kopecký

prof. Ing. Václav Kopecký, CSc.
děkan

Technická univerzita v Liberci - 14 -

L.S.

Kolář

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2014

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 14.5.2015

Podpis: 

Abstrakt

Bakalářská práce se zabývá tématikou fraktální geometrie. Nejprve se práce věnuje objevu a definování fraktálů. Poté rozebírá Hausdorffově dimenzi a teorii nutnou k vytvoření demonstračních aplikací pro e-learning. Praktická část je především věnována implementaci demonstračních aplikací. Popisuje programy na vykreslení Juliových množin s možností smooth coloringu, dále programy na vykreslení fraktální interpolace a demonstraci fraktální komprese obrázku.

Klíčová slova: Fraktál, Hausdorffova dimenze, IFS, Juliovy množiny, fraktální interpolace, fraktální komprese obrázků.

Abstract

This bachelor thesis looks into fractal geometry theme. Firstly it deals with discovery of fractals and their definition. Then it analyzes Hausdorff dimension and theory needed for creating demonstration applications for e-learning. Practical part is mostly devoted to implementation of demonstration applications. Thesis describes application for rendering Julia set with a possibility of smooth colouring, applications for rendering of fractal interpolation and demonstration of fractal image compression.

Key words: Fractal, Hausdorff dimension, IFS, Julia set, fractal interpolation, fractal compression of images.

Obsah

Úvod....10
1 Fraktály....11
2 Definice fraktálů....12
2.1 Deterministické fraktály....13
2.2 IFS....14
2.3 Nedeterministické fraktály....14
3 Fraktální množiny praktické části....15
3.1 Juliovy množiny....15
3.2 Fraktální interpolace....15
3.3 Fraktálni komprese obrazu....17
3.3.1 Tvorba range a domain bloků....17
3.3.2 Zakódování....19
3.3.3 Dekódování....20
3.3.4 Využití fraktálů....20
4 Praktická část....23
4.1 E-learningový kurz....23
4.2 Fraktální interpolace....24
4.3 Juliovy množiny....28
4.4 Fraktální komprese obrázku....30
4.4.1 Velikost komprese....32
4.4.2 Parametr s....32
4.4.3 Parametr o....34
4.4.4 Afinní transformace....35
4.4.5 Kvantifikace chyby....37
4.4.6 Range bloky....38
4.4.7 Domain bloky....38
4.4.8 Další měření....39
4.4.9 Využití fraktální komprese obrázků....41
Závěr....42
Seznam použité literatury....43
Obsah přiloženého CD....45
Příloha....46

Seznam ilustrací

Obrázek 1: Ukázka vlivu délky metru....13
Obrázek 2: Ukázka tvorby B bloků....18
Obrázek 3: Lorenzovo vodní kolo [6]....21
Obrázek 4: Vykreslený Lorenzův atraktor z pokusu s vodním kolem [6]....21
Obrázek 5: Fraktální interpolace s nulovými vertikálními zesíleními....26
Obrázek 6: Znázornění dvojic [x,y] a w4(x,y)....26
Obrázek 7: Ukázka soběpodobnosti u fraktální interpolace....27
Obrázek 8: Vykresnelá fraktální interpolace při nevhodně rozmístěných bodech....28
Obrázek 9: Barevné vykreslení Mandelbrotovi množiny bez použití barevných přechodů....29
Obrázek 10: Histogram parametru s....32
Obrázek 11: Přiblížené okolí nuly histogramu parametru s....33
Obrázek 12: Boxplot analýza parametru s....33
Obrázek 13: histogram parametrů o....34
Obrázek 14: Boxplot analýza parametru o....35
Obrázek 15: Histogram použitých transformací pro obrázek opice....36
Obrázek 16: histogram použitých transformací pro obrázek Leny....36
Obrázek 17: Histogram lokálních chyb....37
Obrázek 18: Původní černobílý obrázek Lena....46
Obrázek 19: Atraktor zakódování při range blocích o rozměrech 4x4 a 1000 domain blocích o rozměrech 8x8....46
Obrázek 20: Původní obrázek Lena....46
Obrázek 21: Atraktor zakódování při range blocích o rozměrech 4x4 a 1000 domain blocích o rozměrech 8x8....46
Obrázek 22: Atraktor zakódování při range blocích o rozměrech 8x8 a 500 domain blocích o rozměrech 16x16....47
Obrázek 23: Původní obrázek Lena....47
Obrázek 24: Původní obrázek Lena....47
Obrázek 25: Atraktor zakódování při range blocích o rozměrech 16x16 a 500 domain blocích o rozměrech 32x32....47
Obrázek 26: Původní fotka dívky se psem....48
Obrázek 27: Atraktor zakódování při range blocích o rozměrech 4x4 a 1000 domain blocích o rozměrech 8x8....48
Obrázek 28: Atraktor zakódování při range blocích o rozměrech 32x32 a 10 domain blocích o rozměrech 64x64....48
Obrázek 29: Původní obrázek gradientu....48
Obrázek 30: 0. iterace....49
Obrázek 31: 1.iterace....49
Obrázek 32: 3. iterace....49
Obrázek 33: 2. iterace....49
Obrázek 34: 5. iterace....49
Obrázek 35: 4. iterace....49

Seznam tabulek

Tabulka 1: Fraktální dimenze známých fraktálů.....13
Tabulka 2: Fraktální dimenze přírodních útvarů.....14
Tabulka 3: průměrná lokální chyba v závislosti na počtu domain bloků.....38
Tabulka 4: Průměrná lokální chyba v závislosti na rozměrech domain bloků.....39

Úvod

V současné době se student mechatroniky během studia nesetká s pojmem fraktální geometrie, což byl důvod k vytvoření e-learningového kurzu, který má za úkol seznámit studenty s tímto pojmem. Součástí e-learningového kurzu budou jak texty, tak demonstrační aplikace, které byly vytvořeny jak během ročníkového projektu, tak v rámci této bakalářské práce.

Vykreslování fraktálů je velmi zajímavým problémem počítačové grafiky a tento e-learningový kurz studentům umožní vyzkoušet vykreslení fraktálů pomocí různých algoritmů s možností úpravy parametrů. Pojem fraktál je ovšem čím dál významnější, nejenom v problematice počítačové grafiky, a proto je důležité, aby student měl alespoň elementární znalosti tohoto pojmu.

Bakalářská práce nejprve popisuje objev fraktálů a jejich definici a s tím spjatý pojem Hausdorffovy dimenze. Poté popisuje algoritmy nutné k vytvoření demonstračních aplikací a následně i jejich implementaci. Bakalářská práce se také hlouběji zabývá pojmem fraktálního kódování obrazu a to především možnostem jeho komprese.

1 Fraktály

Nejčastější definice pojmu fraktál je: Geometrický objekt, který je soběpodobný v různých měřítkách. Jinými slovy jde o objekt, který je tvořen svými vlastními menšími kopiemi. První známější objekt s těmito vlastnostmi je takzvané Cantovo diskontinium což je množina s nenulovou dimenzí menší než 1. Toto diskontinium vzniká tak začneme s úsečkou [0;1] na číselné ose. Z této úsečky vynecháme prostřední třetinu [1/3;2/3], čímž vzniknou dvě úsečky [0,1/3] a [2/3;1]. Iteračně poté pro všechny úsečky vzniklé v předchozí iteraci odeberu prostřední třetinu.

Další objev udělal švédský matematik Helge von Koch, který v roce 1906 popsal spojitou křivku, která nemá derivaci v žádném svém bodě. Tato křivka vzniká tak, že začneme opět s úsečkou [0;1] na číselné ose. Poté se rozdělí úsečka na třetiny, nad prostřední třetinou se vytvoří rovnostranný trojúhelník a původní prostřední třetina se odstraní. Tento postup iteračně opakujeme pro všechny úsečky, které vznikly v předchozí iteraci. Pokud místo úsečky začneme s rovnostranným trojúhelníkem neboli třemi úsečkami, vznikne útvar, který se nazývá Kochova vločka. Pokud se bude počet iterací blížit k nekonečnu bude mít tento útvar tu vlastnost, že jeho obvod bude nekonečně dlouhý, ale obsah bude konečně velký, konkrétněji 8/5 obsahu původního trojúhelníku.

Další důležitý objev byl proveden francouzskými matematiky Gastonem Juliou a Pierem Fatou. Ti popsali vlastnosti iteračních rovnic v komplexní rovině. O jejich objevu více v kapitole 3.1. Na jejich práci poté navázal Benoit Mandelbrot, který počítačově vykreslil množinu, která je dnes známa pod názvem Mandelbrotova množina. Tato množina, kterou popsali také již Julia s Fatouem, je tvořena pomocí iteračního výpočtu $z_{n+1} = z_n^2 + c$. Pro každý bod *c* v komplexní rovině počítáme z_n kde n=<1;∞) a pokud všechna z_n splňují podmínku $|z_n| \leq 2$, pak bod *c* patří do Mandelbrotovy množiny.

Mandelbrot, který v té době pracoval jako zaměstnanec IBM zkoumal cenu bavlny na trhu. Při tom si všiml, že kolísání ceny má jisté vlastnosti, které se opakují v různých měřítkách. Zkoumal také náhodné poruchy na telekomunikačních linkách, kde se poruchy opět objevovaly se stejnými charaktery v různých měřítkách. Toto ho vedlo k hlubšímu zkoumání tohoto jevu a narazil na podobný jev například v tisíciletých záznamech o stavu vody v Nilu. Z uvedeného lze vidět, že fraktální strukturu či fraktální chování mohou mít i „negeometrické“ objekty.

Mandelbrot také jako první zavedl pojem fraktál, který definoval pomocí Hausdorffovy dimenze.

Fraktály byly ovšem známé již dříve díky jejich výskytům v přírodě. Nejčastější příklad fraktálu

v přírodě je zelenina čeledi brukvovité jménem Romanesco, kterou můžete vidět na obrázku. V přírodě ovšem existuje celá řada objektů s fraktálním charakterem, které vykazují do jisté míry soběpodobnost. Jedná se například o stromy, kapradí, blesky, sněhové vločky, oblaka, pobřeží, řeky. Sám Mandelbrot řekl: „Oblaka nejsou koule, hory nejsou kužely, pobřeží nejsou kužele, kůra není hladká, blesk necestuje po přímce.“ Fraktální charakter mají často také útvary vzniklé působením elektrického proudu. Uvádí se, že i lidské plíce mají do jisté míry fraktální charakter a pro výpočet jejich povrchu se využívá znalostí získaných ze studia fraktálů.

Díky výskytů fraktálů v přírodě se fraktály vyskytují také v umění, a to především výtvarném. Japonský malíř Hakusai využíval fraktálů pro ztvárnění vln v 30. a 40. letech 20. století.

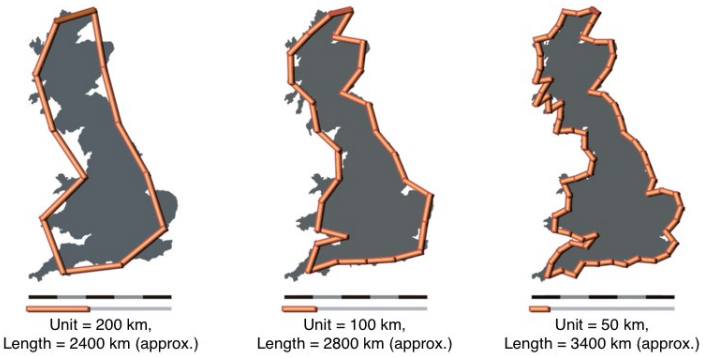
Fraktály lze nalézt také v architektuře. Konkrétněji například u Eiffelovy věže, některých evropských katedrál. Například basilika San Clemente a katedrála v Ravell v Itálii, kde lze najít vzor Sierpinského trojúhelníku, nebo u afrických sídlišť v severní Zambii.

Dnes se fraktály často používají na počítačově generovaná pohoří, mraky, stromy nebo pobřeží v zábavním průmyslu (počítačové hry, speciální efekty ve filmech).

2 Definice fraktálů

Pro definici fraktálů je nejprve nutná znalost pojmů topologická a Hausdorffova dimenze. První formální definice topologické dimenze byla vytvořena českým matematikem Eduardem Čechem. Ve skutečnosti se ovšem jedná o velice prostou vlastnost. Bod má topologickou dimenzi 0, přímka má topologickou dimenzi 1, rovina má topologickou dimenzi 2 a tak dále. Hausdorffova dimenze je definována pomocí vztahu $K = \lim_{\epsilon \to 0} N(\epsilon)\epsilon^D$. V podstatě se jedná o problém, při kterém se snažíme křivku pokrýt stejně dlouhými úsečkami, kde K je délka křivky, ϵ je délka jedné úsečky, $N(\epsilon)$ je počet úseček nutných k pokrytí křivky a D je poté Hausdorffova dimenze. Častá demonstrace měření délky členitých objektů je měření délky pobřeží, které jak již bylo zmíněno, má fraktální charakter. To také vedlo k různé naměřené délce portugalsko-španělské hranice, kterou oba státy změřili s 20% rozdílem.

Pokud by toto pobřeží bylo dostatečně členité a my bychom použili dostatečně malé úsečky, blížila by se spočítaná délka pobřeží k nekonečnu.



Obrázek 1: Ukázka vlivu délky metru

Fraktál je poté definován jako objekt, který má Hausdorffovu dimenzi ostře větší než topologickou. Pokud Hausdorffova dimenze vychází neceločíselná, je nazývána fraktální dimenzi, což je základní vlastnost, kterou popisujeme jednotlivé fraktály.

2.1 Deterministické fraktály

Deterministické fraktály jsou takové, které jsou přesně soběpodobné. Pokud vznikají pomocí *n* různých transformací φ_i tak platí vztah: $A = \bigcup_{i=1}^{n} \varphi(A)$

Fraktální dimenzi D deterministických fraktálů lze spočítat pomocí vztahu $D = \frac{\log n}{\log \frac{1}{S}}$, pokud fraktál vznikl n násobným opakováním transformace φ s koeficientem měřítka S. Pokud bylo použito n různých transformací φ_i s koeficientem měřítka S_i fraktální dimenzi D spočítáme vyřešením rovnice $\sum_{i=1}^{n} S_i^D = 1$.

Tabulka 1: Fraktální dimenze známých fraktálů

Fraktál	Fraktální dimenze
Cantorova množina	Log2/log3≈0,63
Okraj Kochovy vločky	Log4/log3≈1,26
Sierpinského trojúhelník	Log3/log2≈1,58
Mengerova houba	Log20/log27≈0,91
Dračí křivka	Log(√2)/log2=2
Mandelbrotova množina	2
Juliovy množiny	2

2.2 IFS

Velké množství deterministických fraktálů se tvoří pomocí IFS (iteration functions systém, neboli systém iteračních funkcí) Systém IFS je množina uspořádaných dvojic $\{\varphi_i,p_i\}$, kde φ_i je transformace a p_i pravděpodobnost použití i-té transformace. Transformaci φ_i můžeme pro dvourozměrný prostor definovat takto:

$\varphi(x,y)=\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}+\begin{pmatrix} e \\ f \end{pmatrix}$, kde hodnoty *a,b,c,d* odpovídají za zvětšení, zkosení, otočení a koeficienty *e,f* odpovídají za posunutí. Uspořádanou dvojici $\{\varphi_i,p_i\}$ lze nazývat IFS, pokud pro Euklidovu metriku $\rho(X)=\sqrt{x_1^2+x_2^2}$, kde $X=[x_1,x_2]$, každé transformaci φ_i přiřadíme nejmenší reálné číslo *s*, které splňuje předpis $\rho(\varphi(X)-\varphi(Y))<s\,\rho(X-Y)$. Pro pravděpodobnosti p_i poté musí platit $\sum_{i=1}^{n} P_i=1$ a $s_1^{p1}*s_2^{p2}....s_n^{pn}<1$.

2.3 Nedeterministické fraktály

Tyto fraktály nejsou přesně soběpodobné, ale statisticky soběpodobné. To znamená, že v objektu nejsou stejné útvary v různých měřítkách, ale naopak jsou v objektu útvary se stejnými statistickými vlastnostmi. Fraktální dimenze těchto fraktálů lze pouze odhadnout pomocí tzv. metody počítání čtverců pro kruhové okolí bodů. Křivku nejprve pokryjeme N1 nepřekrývajícími se kružnicemi o poloměru r1. Poté křivku pokryjeme N2 nepřekrývajícími se kružnicemi o poloměru r2 pro který platí r2<r1. Odhad fraktální dimenze je poté $D \approx \frac{\log\frac{N_2}{N_1}}{\log\frac{1}{r}}$, kde $r=\frac{r_1}{r_2}$.

Tabulka 2: Fraktální dimenze přírodních útvarů

Útvar	Odhadnutá fraktální dimenze
Pobřeží Jižní afriky	1,05 [10]
Pobřeží Austrálie	1,13 [10]
Pobřeží Velké Británie	1,25 [10]
Pobřeží Norska	1,52 [10]
Povrch mozku člověka	2,79 [12]
Povrch plic člověka	2,97 [12]
Soustava žil a cév člověka	1,63 [12]
Zhoubný melanom	1,05-1,30 [12]

3 Fraktální množiny praktické části

V následujících kapitolách se práce zabývá teorií nutnou ke zpracování demonstračních aplikací.

3.1 Juliovy množiny

Juliovy množiny jsou množiny bodů na komplexní rovině, pro které platí, že posloupnost $z_{n+1}=z_n^2+c$, kde *c* je libovolné komplexní číslo, nediverguje. Hranicí těchto bodů je fraktální křivka s Hausdorffovou dimenzí, která je rovna dvěma. Tyto množiny byly poprvé popsány francouzskými matematiky Gastonem Juliou a Pierem Fatou, kteří zkoumali obecné rekurzivní analytické funkce. Spolu s Mandelbrotovou množinou se jedná o nejznámější fraktální útvar, jejichž grafické znázornění hrálo důležitou roli při popularizaci fraktální geometrie.

Juliovy množiny lze konstruovat několika způsoby. Nejjednodušší způsob je time escape algorithm, neboli algoritmus časového úniku. Tento algoritmus vykresluje Juliovu množinu tak, že iteračně počítá hodnotu z_n pomocí vztahu $z_{n+1}=z_n^2+c$, kde komplexní číslo *c* je konstanta, dokud nebude z_n mimo ohraničenou oblast nebo nebude dosaženo předem daného počtu iterací *N*.

Konkrétněji to znamená, že program pro každý bod začne počítat s hodnotou $z_0=k$, kde *k* jsou souřadnice vyšetřovaného bodu v komplexní rovině. Poté bude program iteračně počítat hodnoty z_n, kde $n=1,2,...,N$. Pro každé z_n bude program kontrolovat, zdali je bod uvnitř kružnice se středem v počátku a poloměrem *R*, neboli zda platí $|z_n|\leq R$. Pokud všechny z_n splňují tuto podmínku, vybarvíme tento bod černě. Pokud jakékoliv z_n „opustí“ ohraničenou oblast, vybarvíme bod bíle, popřípadě barvou určenou vybarvovacím algoritmem. Hodnota poloměru *R* se může pro rovnice druhého stupně typu $z_{n+1}=z_n^2+c$ zvolit $R=2$, pokud platí $|c|<2$. Pro rovnice třetího stupně typu $z_{n+1}=z_n^2+az+b$ můžeme zvolit poloměr $R=max\{|(b)|,\sqrt{a+2}\}$. Pro obecné rovnice typu $z_{n+1}=z_n^m+c$, kde $m\geq 4$ můžeme zvolit poloměr $R=max\{|(c)|,2^{\frac{1}{m-1}}\}+\varepsilon$, kde $\varepsilon>0$ je libovolné kladné číslo. Nejedná se ovšem o podmínky optimální, avšak dostačující.[13]

3.2 Fraktální interpolace

Fraktální interpolace je proces, který vytváří křivku fraktálního charakteru, procházející předem zvolenými body. Jednoduchá fraktální interpolace bez skrytých proměnných probíhá pomocí IFS,

neboli pomocí systému iteračních funkcí. Mějme interpolační body $[x_n,y_n]$, kde $n=0,1..N$, kterými chceme vést fraktální křivku. Tuto křivku budeme sestrojovat pomoci IFS, neboli pomocí systému iteračních funkcí. Tento systém se skládá z funkcí w_n $(n=1,2..N)$. Obecně mají tyto iterační funkce podobu:

$$w_n \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_n & b_n \\ c_n & d_n \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e_n \\ f_n \end{pmatrix}$$

Pro fraktální interpolaci se ovšem používají funkce w_n, které mají hodnoty $b_n=0$. To umožňuje, že výsledná křivka bude procházet určenými body. Všechny funkce w_n musí dále splňovat podmínky:

$$w_n \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} x_{n-1} \\ y_{n-1} \end{pmatrix} \qquad w_n \begin{pmatrix} x_N \\ y_N \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \end{pmatrix}$$

Tyto podmínky způsobí, že funkce w_n bude zobrazovat interval $\langle x_0,x_N \rangle$ na interval $\langle x_{n-1},x_n \rangle$, jelikož se jedná o spojitou transformaci.

Na základě těchto podmínek dostaneme, při rozepsání složek rovnice iteračních funkcí, následující soustavu rovnic:

$$\begin{aligned} x_{n-1} &= a_n x_0 + e_n \\ x_n &= a_n x_N + e_n \\ y_{n-1} &= c_n x_0 + d_n y_0 + f_n \\ y_n &= c_n x_N + d_n y_N + f_n \end{aligned}$$

Řešením této soustavy je:

$$\begin{aligned} a_n &= \frac{x_n - x_{n-1}}{x_N - x_0} \\ e_n &= \frac{x_N x_{n-1} - x_0 x_n}{x_N - x_0} \\ c_n &= \frac{y_n - y_{n-1}}{x_N - x_0} - dn\frac{y_N - y_0}{x_N - x_0} \\ f_= &\frac{x_N y_{n-1} - x_0 y_n}{x_N - x_0} - dn\frac{x_N y_0 - x_0 y_N}{x_N - x_0} \end{aligned}$$

Hodnotu d_n nazýváme vertikální škálovací faktor, který je zvolen na základě vlastností fraktálu,

který má procházet interpolačními body. Platí věta:

Nechť N je přirozené číslo větší než 1 a $\{w_n : n=1,2,...,N\}$ je IFS asociovaný s daty $\{(x_n,y_n) : n=0,1,...,N\}$.

Nechť pro všechny vertikální faktory kontrakce je $0<d_n<1$. Pak existuje metrika d na R^2 ekvivalentní Euklidově metrice taková, že IFS je hyperbolický a jednoznačně určuje atraktor.

$$G = \bigcup_{n=1}^{N} w_n(G)$$

Pak atraktor G je graf spojité funkce, který interpoluje daná data. Takovou funkci nazýváme fraktální interpolací.[17],[18],[13]

3.3 Fraktálni komprese obrazu

Fraktální komprese obrazu je proces, pomocí kterého lze zakódovat obrazovou informaci pomocí soběpodobnosti bloků v obrazu.

Problém fraktální komprese lze rozdělit na tyto problémy:

- tvorba range a domain bloků
- zakódování obrazu
- rozkódování obrazu

3.3.1 Tvorba range a domain bloků

Range bloky musí být tvořeny tak, aby se nepřekrývaly a zároveň pokrývaly celý obraz. Range bloky lze tvořit závisle nebo nezávisle na obraze.

Při tvorbě range bloků nezávisle na obraze, je nejvhodnější volit range bloky jako stejně velké mnohoúhelníky. Nejčastěji se používají čtvercové range bloky, které jsou nejjednodušší, jak na implementaci, tak na výpočet. U čtvercových range bloků ovšem často dochází ke zkreslení na hranách dělení, které je pro uživatele často viditelnější, než kdybychom použili například čtvercové bloky zrotované o 45º.

Další z možných tvarů range bloků je například trojúhelník, šestiúhelník a podobně. Jedná se ovšem

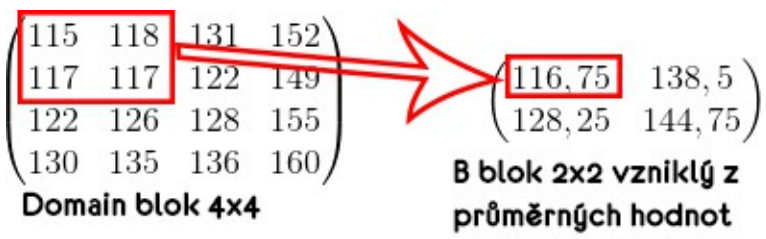
o výpočetně náročnější metody. Při použití jiných než čtvercových nebo obdélníkových bloků, je třeba si dát pozor na pokrytí celého obrazu. Pokrytí můžeme docílit například tím, že původní obdélníkový obraz rozšíříme na takový mnohoúhelník, který lze úplně pokrýt stejně velkými mnohoúhelníky. Všechny pixely, které tímto rozšířením přidáme, budou nulové a při dekódování se nebudou zobrazovat.

Obrazově závislá volba range bloků je například dělení obrazu pomocí kvadraturního stromu. Začneme s nezávisle zvolenými čtvercovými range bloky. U každého range bloku zkontrolujeme jeho homogenitu a možnosti jeho zakódování. Pokud nebude homogenita na dostatečné úrovni, nebo nebude možné range blok zakódovat s dostatečně malou lokální chybou, musíme range blok upravit. Range bloky vždy upravujeme tak, že je rozdělíme na čtyři nové range bloky s poloviční rozměry, u kterých budeme opět kontrolovat podmínky. Tento proces budeme opakovat pro všechny range bloky, dokud nebudou mít minimální rozměry, nebo nebudou splňovat podmínky.

Další možností je například tvorba range bloků pomocí horizontálně vertikálního dělení. To obraz rozdělí na různé mnohoúhelníky podobné obdélníkům. Dalším způsobem tvorby range bloků je použití trojúhelníkového dělení, které obraz rozdělí na trojúhelníky různých rozměrů.

U těchto metod je ovšem nutné zvolit metodu kontrakce domain bloků, která bude schopná, s co nejmenší chybou, kontrahovat domain blok na rozměry daného range bloku. Je také nutné zvolit metodu, která bude schopná zpětně rekonstruovat range bloky s nejmenším možným počtem uložených dat.

Jak již bylo řečeno, je nutné domain bloky kontrahovat na rozměry range bloků a vytvořit tím takzvané B bloky. Této kontrakce lze dosáhnout opět mnoha způsoby. Obecně lze tvrdit, že lze zvolit jednoduchou a přesnější metodu, pokud budou range a domain bloky splňovat dané požadavky. Pokud budou range a domain bloky čtvercové a rozměry domain bloků budou celočíselným *i*-násobkem rozměru range bloků, můžeme domain blok rozdělit na subbloky o rozměrech *i* krát *i*, kterých bude stejný počet jako pixelů v range blocích. Prvky B bloku poté budou vždy průmerné, maximální, nebo minimální hodnoty z odpovídajících subbloků domain bloku.



Obrázek 2: Ukázka tvorby B bloků

3.3.2 Zakódování

Východiskem při zakódování je především fakt, že při dekódování usilujeme o to, aby každý domain blok byl rozkódován pomocí jednoho z B bloků následujícím předpisem:

$$\overline{R_i} = s(i) * \overline{B_j} + o(i) * \overline{1}$$

kde

- R_i jsou prvky range bloku
- $s(i)$ je parametr pro transformaci kontrastu
- B_j jsou prvky B bloku
- $o(i)$ je parametr pro transformaci jasu
- 1 je jednotková matice o stejné velikosti jako B a range bloky

Chybu kontrakce můžeme podle [2] vyjádřit následovně:

$$E(i) = \sqrt{\sum_{k=1}^{n}(r_k - (s(i) * b_k + o(i)))^2}$$

kde r_k jsou prvky range bloku R_i, b_k jsou prvky B bloku B_j a n je počet prvků v domain a B blocích. Pokud chceme tuto chybu minimalizovat, musíme vyřešit následující soustavu rovnic:

$$\frac{\delta E}{\delta s} = 0 \qquad \frac{\delta E}{\delta o} = 0$$

Řešením této soustavy dostáváme rovnice pro výpočet parametrů s minimální chybou:

$$s = \frac{n\sum_{k=1}^{n} r_k b_k - \sum_{k=1}^{n} r_k \sum_{k=1}^{n} b_k}{n\sum_{k=1}^{n} r_k^2 - (\sum_{k=1}^{n} r_k)^2} \qquad o = \frac{1}{n}[\sum_{k=1}^{n} b_k - s\sum_{k=1}^{n} r_k]$$

Tyto parametry jsou dané tak, že pro dvojici domain a B bloku proběhne dekódování s minimální možnou chybou pro spárování těchto dvou bloků. Pro každý range blok musíme ovšem ověřit, který z B bloků má nejmenší možnou chybu *E*. Proto je potřeba spočítat hodnoty *s*, *o* a *E* pro každou dvojici range a B bloků.

Poté, co pro každý range blok *i* najdeme B blok *j* s nejmenší chybou *E*, uložíme si následující informace do matice řešení: $x_i^* = (x,y,s,o,i)$, kde *x* a *y* jsou souřadnice *j*-tého domain bloku, *s* a *o* jsou parametry transformace pro danou dvojici. Hodnota *i* je pořadové číslo rotace a otočení.

Pro přesnější zakódování obrazu lze testovat nejenom dvojice range a B bloků, ale i kombinace range bloků s otočenými nebo zrcadlově přetočenými B bloky. Těchto rotací a otočení se obecně používá šest. První je neotočená a nepřevrácená, další je otočená o 90º, o 180º a o 270º, a poslední je horizontálně a vertikálně převrácená.

3.3.3 Dekódování

Samotné dekódování probíhá pomocí následujícího předpisu:

$$\overline{R_i} = s(i) * \overline{B_j} + o(i) * \overline{1}$$

Tento předpis opakovaně aplikujeme na všechny range bloky. Jelikož je výsledný obraz atraktorem, tak můžeme pro nultou iteraci použít jakýkoliv obraz. Nejčastěji se ovšem používá obraz celý černý, nebo celý bílý.

Při každé iteraci musíme nejdříve přepočítat hodnoty prvků B bloků. Souřadnice použitých domain bloků můžeme získat z matice řešení. Musíme však také znát velikost domain a range bloků. Tato informace není nikde uložená, a proto musí být součástí vstupních informací.

Poté můžeme spočítat hodnoty prvků každého range bloku. Pro vykreslení range bloky do výsledného obrazu musíme znát rozměry původního obrazu. Jelikož známe rozměry range bloků a jejich počet (počet řádků matice řešení je roven počtu range bloků), stačí nám pouze jeden z rozměrů, nebo jejich poměr. Kolik iterací bude nutných provést, než dospějeme k atraktoru, záleží na nulté iteraci a na zakódování. Obecně ovšem stačí kolem patnácti iterací. Ukázku rozkódování můžete vidět v příloze na obrázcích 30-35. [2]

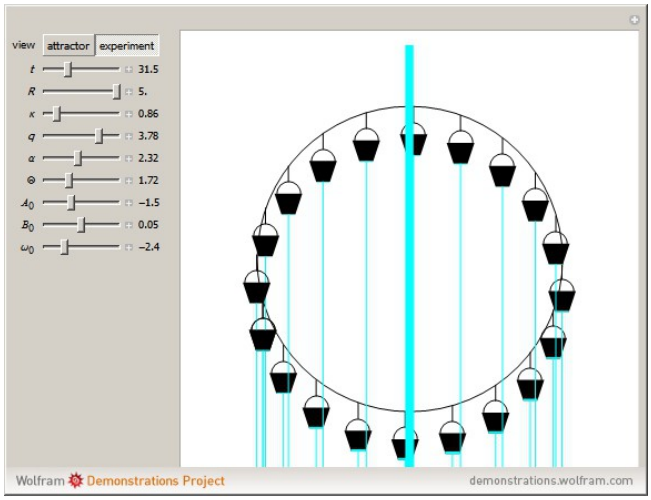
3.3.4 Využití fraktálů

Možná využití fraktální geometrie lze rozdělit do tří hlavních kategorií:

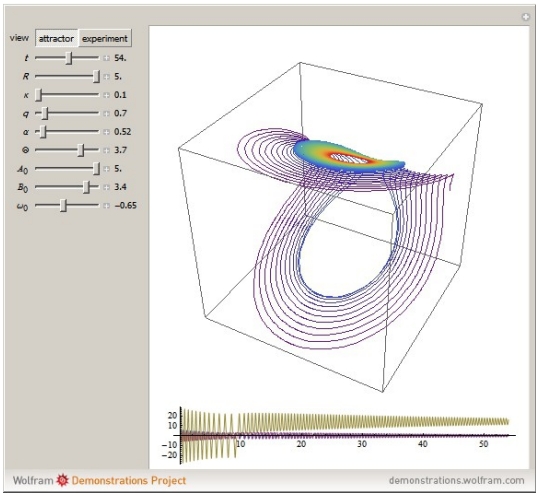
- zkoumání objektů s fraktálním charakterem
- vykreslování fraktálů
- využívání vlastností fraktálů

Jak bylo již zmíněno, fraktály můžeme najít v mnoha různých vědních odvětvích. Fraktální geometrii lze využít například v medicíně, neboť mnoho objektů v lidském těle má fraktální charakter. Pokud chceme určit plochu nebo obvod těchto objektů, je nutné využít objevů fraktální geometrie.

Fraktály také zasahují do meteorologie, protože spousta turbulentních proudění má fraktální charakter. Tento fraktální charakter popsal Edward Norton Lorenz pomocí Lorenzova atraktoru. Tento atraktor má pro určité parametry a vstupní hodnoty chaotické vlastnosti, které nelze příliš předpovídat. Tento atraktor popisuje také další chaotické jevy, jako například chování laseru, dynama a podobně. Existuje také pokus, který si každý může vyzkoušet doma často označovaný jako Lorenzovo vodní kolo. Vezmeme otočné kolo a umístíme na něj několik nádob s otvorem na dně. Do těchto nádob poté budeme pouštět vodu konstantní rychlostí. Při specifických počátečních podmínkách a při specifické rychlosti přitékající vody se rychlost otáčení bude chovat chaoticky. [13]



Obrázek 3: Lorenzovo vodní kolo [6]



Obrázek 4: Vykreslený Lorenzův atraktor z pokusu s vodním kolem [6]

Další významný výskyt fraktálů je v ekonomii. Jak již bylo zmíněno, tento úkaz vedl Mandelbrota ke studiu problematiky fraktálů. Fraktálů využívá například teorie Elliottových vln, pomocí které lze předpovídat budoucí vývoj cen na burze na základě opakující se soběpodobnosti v různých

měřítkách. Tato soběpodobnost vyplývá především z opakujících se reakcí burzovních obchodníků na opakující se podněty.[13]

Fraktály lze také využít na základě jejich vzhledu. Fraktální útvary se vyskytují v přírodě v podobě mraků, pobřeží, nebo pohoří. Existují metody, které jsou schopné náhodně vygenerovat plochu s fraktálním charakterem po zadání požadovaného zvrásnění. Pokud se zaměříme na počítačové hry, jsou v nich objekty nejčastěji vyjádřeny jako množina polygonů. Toto velmi zjednodušuje práci s fraktálními strukturami, jelikož fraktální soběpodobnost v tomto případě neznamená nic jiného, než že výsledný objekt je tvořen opakujícími se prvky v různých měřítkách, což v tomto případě budou různé polygony. Samotná tvorba fraktálního terénu poté probíhá iteračně pomocí takzvaného fraktálního Brownova pohybu (fBM). [8]

Časté použití fraktálů v technice je při tvorbě antén. Fraktální antény mají obecně tu vlastnost, že jsou schopné přijímat velké spektrum frekvencí. Další jejich výhodou je malá velikost, a tudíž jsou vhodné pro použití v mobilní nebo armádní technice.[9]

Soběpodobnost v různých měřítkách byla také zdokumentována u rozložení větších měst. Pokud v jednom měřítku bereme budovy a cesty, které je propojují a v dalším měřítku vezmeme bloky a hlavní silnice propojující jednotlivé bloky, lze pozorovat soběpodobnost v těchto měřítkách. Struktury jednotlivých bloků se také můžou často opakovat. Tohoto lze využít při plánování nové městské části, což může zefektivnit dopravní kapacitu.

Při fraktálním větvení proudu kapaliny lze předejít turbulentním jevům a docílit tak čistě lineárního proudění. To je důležité například při Ionizační výměně nebo při míchání dvou kapalin, kde turbulence může ovlivnit výsledný produkt.[7]

4 Praktická část

4.1 E-learningový kurz

Součástí této práce je vytvoření e-learningového kurzu, který má za úkol představit studentům pojem fraktální geometrie. Kurz bude dostupný všem studentům mechatroniky na Technické univerzitě v Liberci. Tento kurz bude obsahovat dokumenty a demonstračních aplikace, které budou mít za cíl představit studentům základní pojmy z oboru fraktální geometrie a prakticky je demonstrovat pomocí aplikací. Implementace aplikací a algoritmy použité aplikacemi budou vždy popsány v dokumentech odpovídající kapitoly. Tyto aplikace budou vykreslovat fraktální útvary pomocí různých algoritmů a budou uživateli umožňovat změnou různých parametrů zjišťovat jejich vliv na výslednou fraktální křivku. Parametry, které může uživatel měnit, jsou specifické pro použitý algoritmus. Jedná se například o počet vykreslených bodů, počet provedených iterací a podobně.

Jednotlivé kapitoly e-learningového kurzu budou:

- Historie a úvod do problematiky fraktální geometrie
- Hausdorffova dimenze
- IFS a Bernsleyho kapradí
- Teorie chaosu a Sierpinského trojúhelník
- Dračí křivka
- Time escape algorithm a Mandelbrotova množina
- Smooth coloring a Juliova množina
- Fraktální interpolace
- Fraktální komprese obrázků

Pro každou kapitolu je vytvořen pdf dokument, který bude mít za cíl především, vysvětlit dané pojmy srozumitelnou formou. Oproti informacím, které si student může vyhledat sám, bude mít tento kurz výhodu především v ucelenosti a v možnosti každý algoritmus vyzkoušet pomocí

demonstračních aplikací. Tyto dokumenty jsou proto psány s ohledem na těchto aplikace a budou se popisovat především algoritmy, které byly přímo použity v demonstračních aplikacích, aby měl student možnost daný algoritmus vyzkoušet s různými parametry.

Aplikace, které budou součástí kurzu a které byly zpracovány v rámci ročníkové projektu, jsou:

- Aplikace na vykreslování Sierpinského trojúhelníků pomocí tvorby trojúhelníku o poloviční rozměrech, který slouží na demonstraci nedostatků klasické počítačové grafiky
- Aplikace na vykreslení Sierpinského trojúhelníku pomocí hry chaosu/ náhodné procházky
- Aplikace na vykreslení kapradí pomocí pravděpodobnostního systému IFS
- Aplikace na vykreslení dračí křivky pomocí IFS
- Aplikace na vykreslení Mandelbrotovy množiny s možností přiblížení

Aplikace, které budou součástí kurzu a které jsou součástí bakalářské práce, jsou:

- Aplikace na vykreslení Juliových množin s možnosti smooth coloringu
- Aplikace na tvorbu fraktální interpolace s možností zadání interpolačních bodů
- Aplikace na demonstraci Fraktální komprese obrázků s možností nastavení parametrů kódování a výsledným dekódováním

Tyto aplikace byly tvořeny ve vývojovém prostředí Delphi pomocí jazyku object pascal. Aplikace budou na e-learningovém kurzu dostupné pouze jako exe soubor, jelikož nejpodstatnější části implementace budou popsány v dokumentaci. Tyto aplikace nejsou optimalizovány, co se týče paměťové nebo časové složitosti a často jsou omezené také možnosti jejich využití. Důvody těchto nedostatků spočívají v zaměření na co možná nejlepší demonstraci, a proto byla často opomíjena další rozšíření aplikace, která by ovšem nezvyšovala úroveň demonstrace. V dokumentaci budou tedy popsána možná vylepšení a rozšíření demonstračních aplikací stejně tak jako výhody a nevýhody použitých algoritmů. Součástí každé aplikace je také návod na její použití, aby student byl schopen co nejdříve pochopit, jak může nastavit různé parametry a jak může spustit jednotlivé funkce aplikace.

4.2 Fraktální interpolace

Demonstrační aplikace pro fraktální interpolaci má za úkol zpracovat uživatelem zadané body

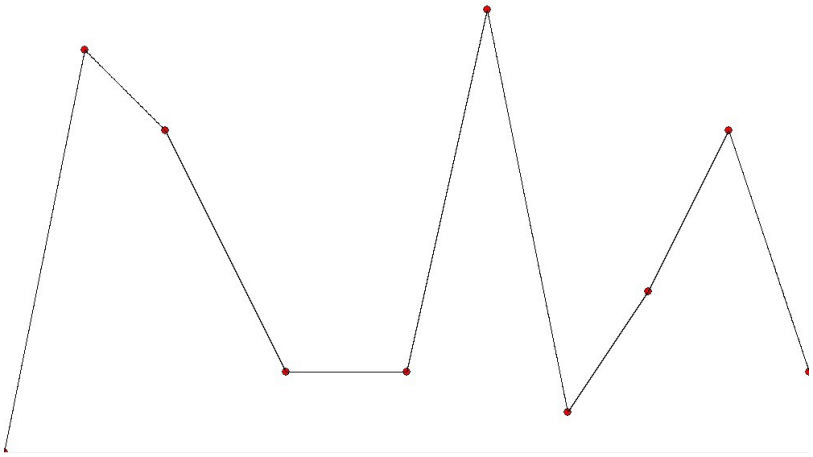
(většinou naměřené hodnoty) a hodnoty vertikálních škálovacích faktorů. Na základě těchto vstupů aplikace vykreslí fraktální křivku, která bude procházet zadanými body a která bude mít odpovídající vlastnosti vzhledem k vertikálním faktorům. Tuto křivku se program nebude snažit nijak vyjádřit, bude se ale snažit najít co nejvíce bodů, které na této křivce leží. Souřadnice těchto bodů se budou ukládat do polí *x* a *y*.

Nejprve se do těchto polí uloží hodnoty zadaných bodů, jelikož z principu interpolace vyplývá, že tyto body budou součástí výsledné křivky. Jelikož atraktor splňuje následující rovnost:

$$G = \bigcup_{n=1}^{N} w_n(G)$$

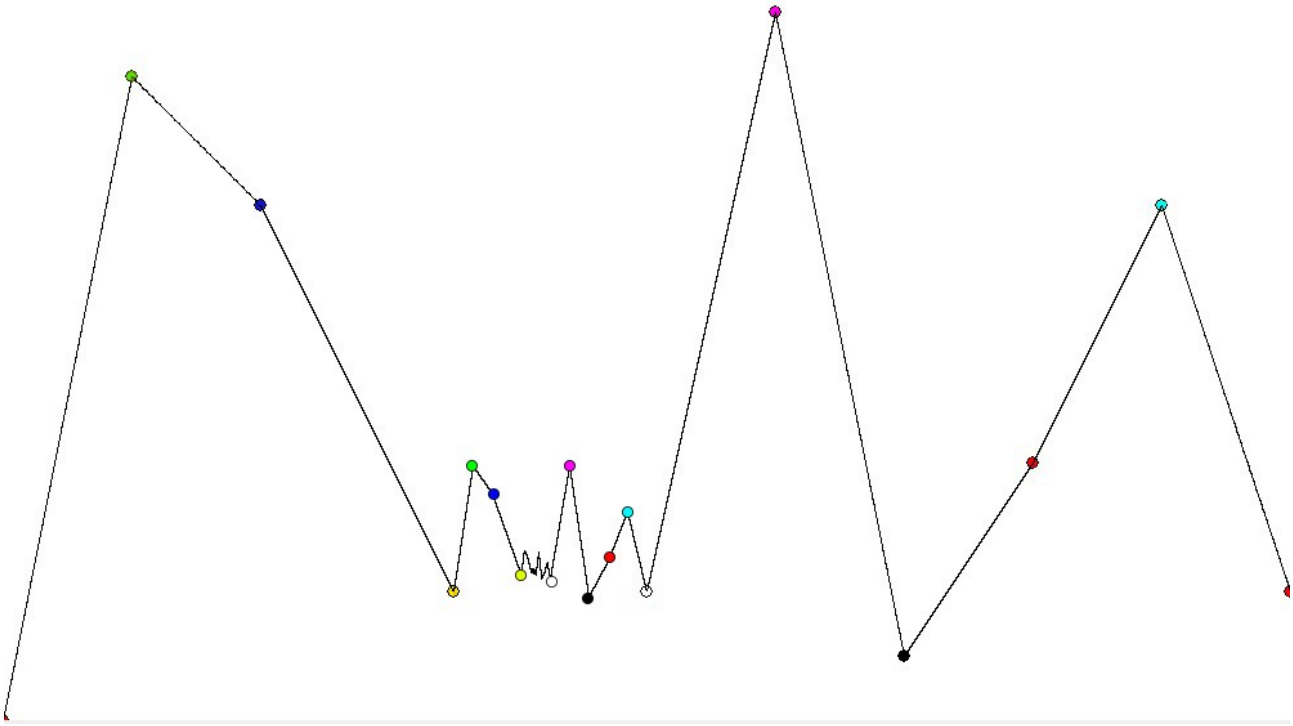
Platí tedy, že pokud zobrazíme body, o kterých víme, že jsou v atraktoru, dostaneme body, které musí také být součástí atraktoru. V první iteraci zobrazíme všechny iterační body pomocí všech zobrazení w_n. Dostaneme body, které si v paměti budeme ukládat jako dvojici reálných čísel, aby při několikanásobném zobrazení nedocházelo k příliš velkým zaokrouhlovacím chybám. Tyto body z R^2 budeme při zobrazování převádět na dvojici přirozených čísel, které budou odpovídat souřadnicím pixelů v obrázku. Toto převádění záleží především na velikosti obrázku a na požadovaném rozsahu zobrazovaných hodnot. Rozsah zobrazovaných hodnot je v případě demonstrační aplikace <0;100> pro *x*-ové hodnoty a <0;60> pro *y*-ové hodnoty. V následujících iteracích ze stejných důvodů vždy zobrazíme všechny nově vzniklé body z předchozí iterace pomocí všech funkcí w_n. Tento proces budeme opakovat, dokud nedosáhneme požadovaného počtu bodů.

Samotný fraktální charakter výsledné křivky nemusí být na první pohled zřejmý. Pokud bude $d_n=0$ pro $n=1,2,...,N$, bude interpolační křivka lomená čára, kterou lze vidět na následujícím obrázku (červeně jsou vyznačeny interpolační body).



Obrázek 5: Fraktální interpolace s nulovými vertikálními zesíleními

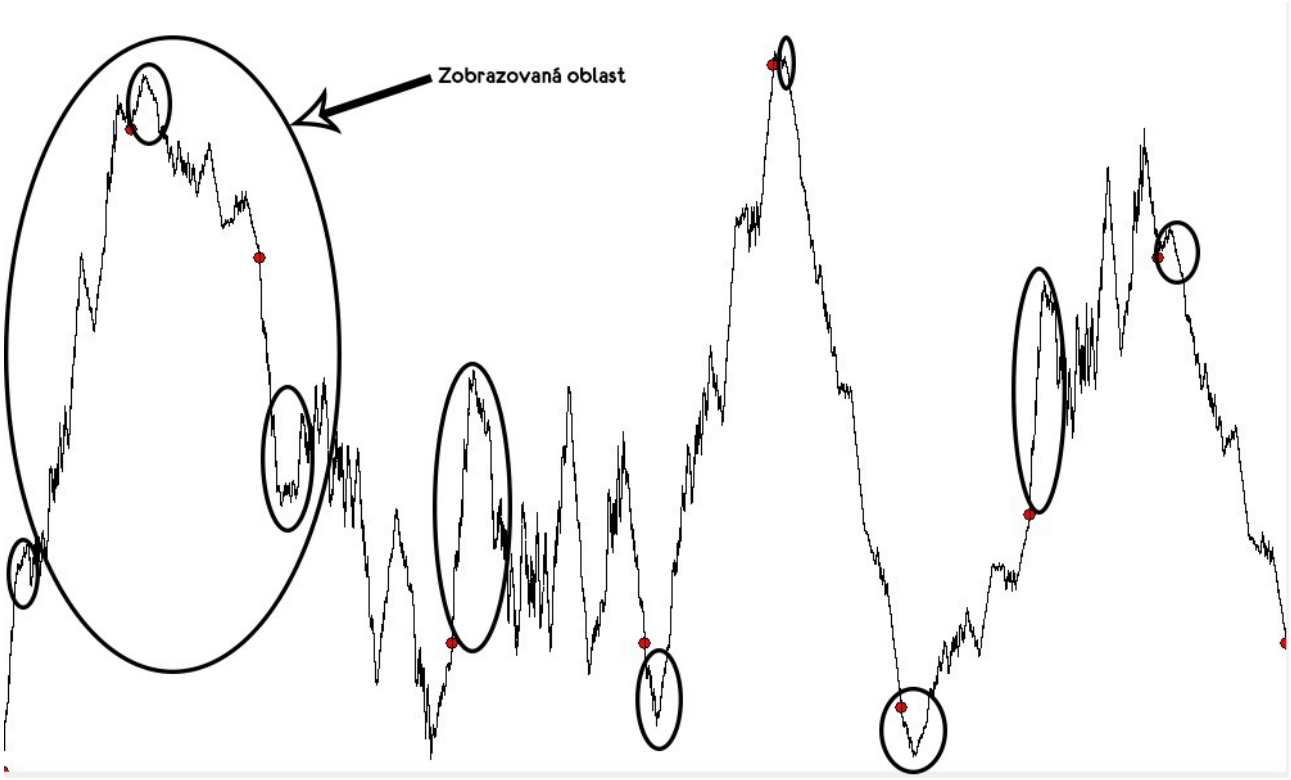
Pokud bude například $d_4=0,2$ je možné již na interpolační křivce pozorovat soběpodobnost. Tato soběpodobnost je patrná na dalším obrázku, kde interpolační bod x,y a bod $w_4(x,y)$ jsou znázorněny stejnou barvou.



Obrázek 6: Znázornění dvojic [x,y] a $w_4(x,y)$

Je možné si povšimnout, že i pro oblast mezi body $w_4(x_3,y_3)$ a $w_4(x_4,y_4)$ by šlo opět zvýraznit body, které jsou ve skutečnosti podruhé zobrazené iterační body. Mezi $w_4(w_4(x_3,y_3))$ a $w_4(w_4(x_4,y_4))$

bychom mohli najít iterační body, které byly zobrazeny třikrát a podobně. Pokud bude d_4=*-0,2*, bude celý úsek mezi body x_3 a x_4 osově převrácen podle osy, která prochází těmito body. Na následujícím obrázku je vidět opakující se motiv zobrazované oblasti na křivce fraktální interpolace.

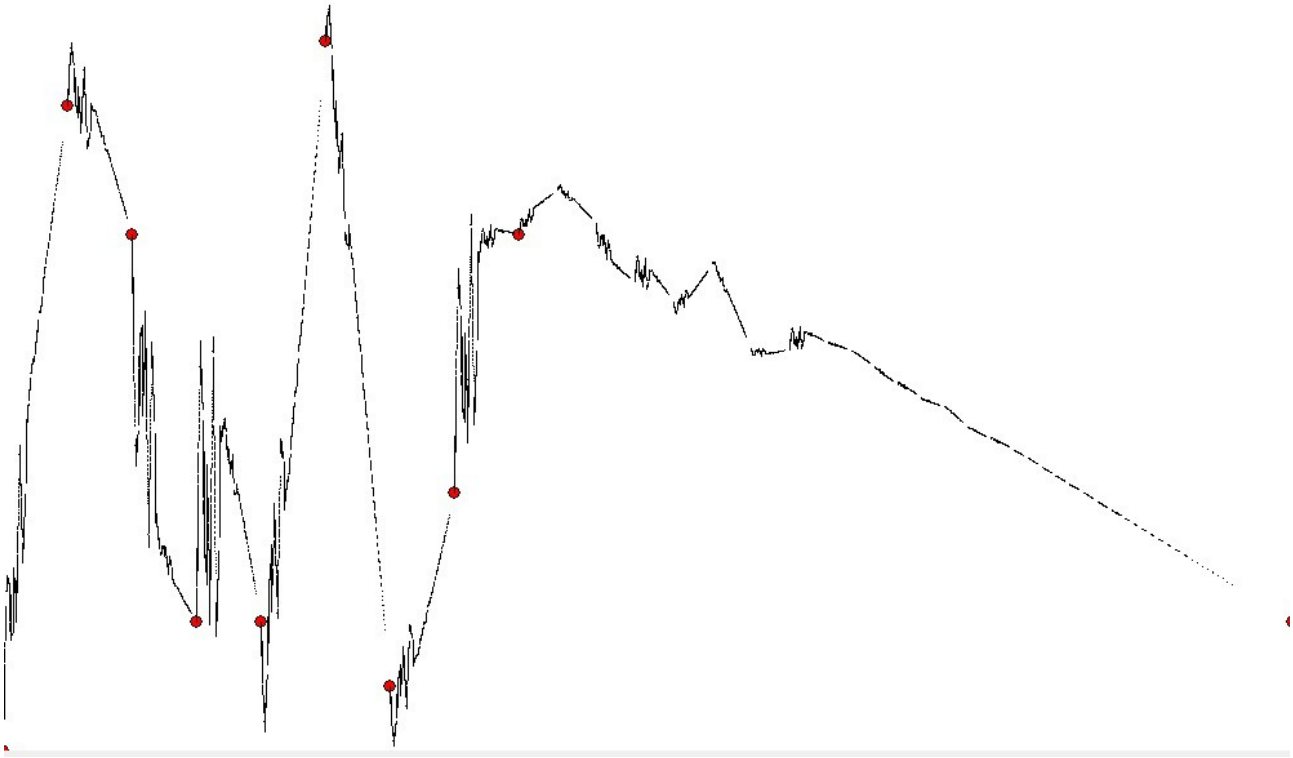


Obrázek 7: Ukázka soběpodobnosti u fraktální interpolace

Špatná volba počtu vykreslených bodů může způsobit, že křivka bude přerušovaná, nebo bude velké množství bodů splývat vlivem zaokrouhlovací chyby. Platí, že zobrazení w_n bude jakékoliv (nejen iterační) body z množiny $<x_0,x_N>$ zobrazovat na množinu $<x_{n-1},x_n>$, pomocí tohoto předpisu:

$$\frac{w_n(x) - w_n(x_{n-1})}{w_n(x_n) - w_n(x_{n-1})} = \frac{x - x_{n-1}}{x_n - x_{n-1}} \qquad kde\, x \in < x_{n-1}; x_n >$$

Tato vlastnost má výhodu, že pokud budou hodnoty x_n-x_{n-1} stejné pro $n=1,2,...,N$, bude stačit malý počet bodů pro vykreslení spojité křivky. Pokud ovšem bude většina bodů u sebe a naopak některé budou velmi vzdálené, bude potřeba většího počtu bodů, aby nedošlo k vykreslení přerušované křivky, jako na následujícím obrázku:



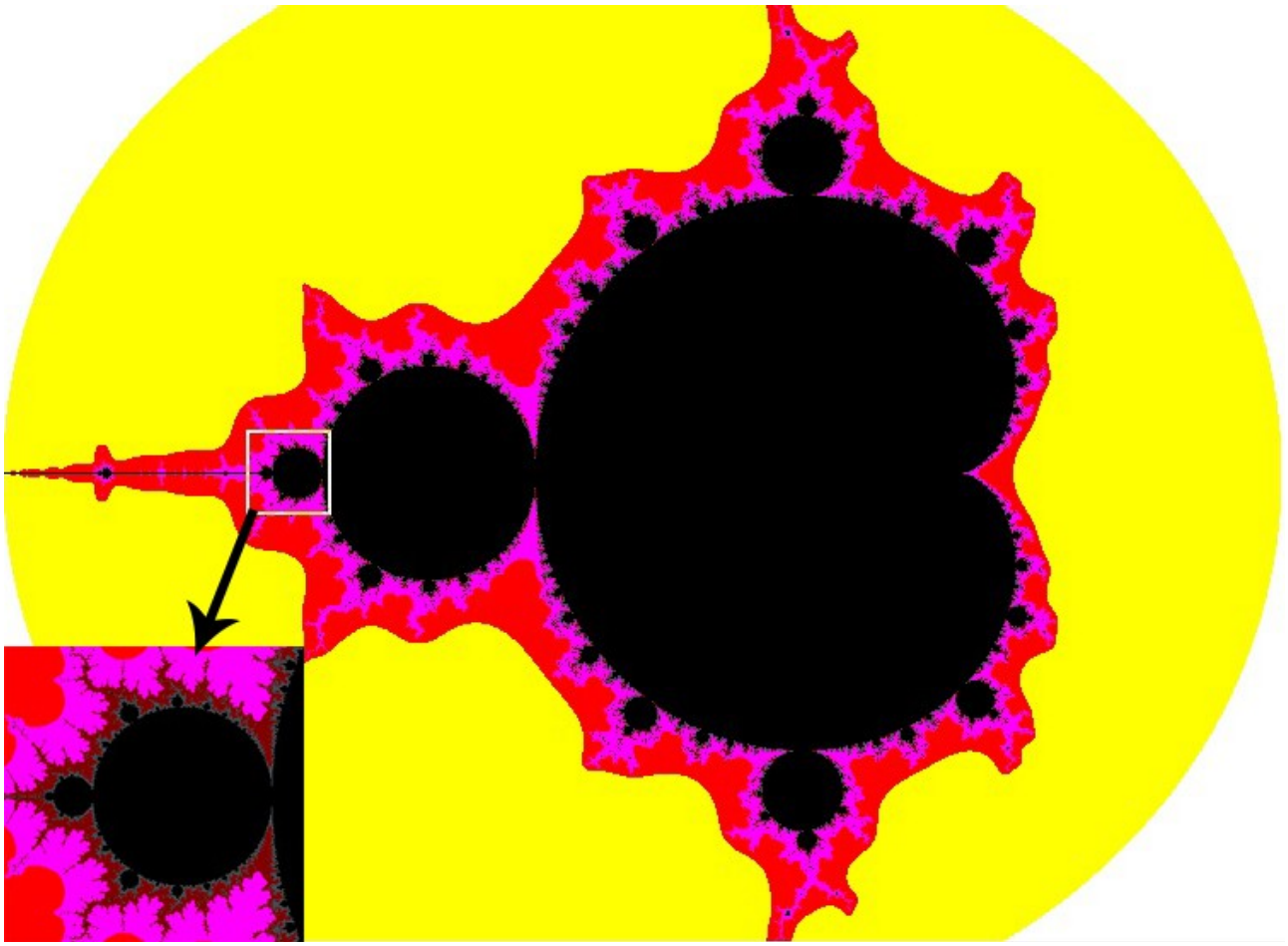
Obrázek 8: Vykresnelá fraktální interpolace při nevhodně rozmístěných bodech

4.3 Juliovy množiny

Aplikace na vykreslení Juliových množin umožňuje uživateli vykreslení této množiny druhého stupně s možností zadání hodnoty *c* a maximálního počtu iterací *N*. Samotné vykreslení probíhá tak, že program bere jednotlivé pixely, pro které přepočítá souřadnice odpovídajícího bodu v komplexní rovině. Pro tento bod program bude počítat hodnoty z_n podle rovnice $z_{n+1}=z_n^2+c$ $(z_0=0)$, dokud nebude platit, že $|z_n|>2$ nebo $n=N$.

Uživatel také může rozhodnout, zdali bude Juliova množina vykreslována černobíle nebo barevně. Pokud zvolí černobíle, budou černě vykresleny ty body v komplexní rovině, které splní $|z_n|\leq 2$ pro $n=1,2,...,N$. Všechny ostatní body, neboli body pro které existuje $n<N$ takové, že $|z_n|>2$, budou vykresleny bíle. Pokud uživatel zvolí barevnou metodu vykreslení, tak program využije smooth coloringu, neboli plynulého přechodu barev. Barevné vykreslení Juliovy množiny znamená, že program barevně znázorní, v jaké iteraci bod unikl. Velké množství bodů unikne při průběhu několika prvních iterací a naopak u pozdějších iterací unikne malé množství bodů. Pokud určíme barvu pro body, které uniknou v určité iteraci nebo v určité množině iterací, budou viditelné

isovrstevnice. Na následujícím obrázku můžete vidět vykreslenou Mandelbrotovu množinu s použitím pěti barev.



Obrázek 9: Barevné vykreslení Mandelbrotovi množiny bez použití barevných přechodů

Toto vybarvení je sice názornější, ovšem možnost tohoto vybarvení je již implementovaná v aplikaci demonstrující tvorbu Mandelbrotovy množiny. Graficky efektivnější je využití barevných přechodů, kde se neomezíme na celočíselné určení iterace, při které daný bod unikne. Tento proces probíhá následovně:

Nejprve pro každý bod *z* v komplexní rovině určíme hodnotu *n*, což je pořadové číslo iterace, při které bod unikl. Dále spočítáme:

$$x = \frac{n - \frac{ln(ln|z|)}{ln(2)}}{N} * H$$

kde *N* je maximální počet iterací a *H* je počet barev v paletě. Zaokrouhlená hodnota *x* poté určuje barvu z palety, která bude použita pro vybarvení bodu *z*. Pokud zvolíme *H=256* není nutné tvořit paletu, ale můžeme použít například RGB barvy (0,0,round(x)).

4.4 Fraktální komprese obrázku

Program na demonstraci fraktální komprese obrázku je schopný zakódovat obrázek o rozměrech 512x512 pixelů zadaný uživatelem. Program kóduje obrázek ve stupních šedi i v RGB formátu. Uživatel může vybrat z nabídky velikostí range a domain bloků a dále může zvolit počet domain bloků. Program je poté schopný daný obrázek rozkódovat, ať už iteraci po iteraci, nebo v patnácti iteracích najednou. Program po zakódování informuje uživatele o možné velikosti binárního souboru s informacemi nutnými k rozkódování zakódovaného obrázku a o velikosti původního obrazového souboru. V poslední řadě uživatel může zobrazit pokrytí obrázku range a domain bloky.

Při zakódování obrázku do odstínu šedi program nejprve převede všechny pixely obrázku z RGB hodnot do odstínů šedi pomocí vztahu:

$$I = 0.2989\mathrm{R} + 0.5870\mathrm{G} + 0.1140\mathrm{B}$$

Hodnoty odstínů šedi si program uloží do dvourozměrného pole o rozměrech 512x512 a také tyto hodnoty dosadí jako hodnoty R, G i B. V dalším kroku si do range a domain polí uloží souřadnice levých horních rohů range a domain bloků. Program nezávisle na obraze zvolí čtvercové range bloky. Pole range bloků není sice pro kódování nezbytné, ale zrychlí samotný proces kódování a jeho dekódování. Program zvolí nezávisle na obraze také čtvercové domain bloky a to náhodně pomocí funkce random tak, aby domain bloky nepřesahovaly mimo rozměry obrázku.

Dále program spočítá hodnotu B bloků jako průměrný stupeň šedi jednotlivých subbloků domain bloků. Rozměry těchto subbloků jsou rovny poměru rozměrů domain bloků k rozměrům range bloků. Jelikož rozměry domain a range bloků uživatel volí z nabídky, je vždy zaručena celočíselnost tohoto poměru.

V tuto chvíli má program již všechny informace, aby byl schopen provést zakódování. Pro každý range blok hledá B (domain) blok, pomocí kterého lze range blok zakódovat s nejmenší lokální chybou. Program nejprve pro dvojici range a B bloku spočítá hodnotu parametrů *s* a *o,* poté spočítá hodnotu lokální chyby. Pro každý range blok si program průběžně pamatuje dvojici, která měla do té doby nejmenší lokální chybu. Tato dvojice bude po otestování všech možných dvojic nejlepší možnou dvojicí pro tento range blok. Informace o této dvojici a parametrech transformace si uloží do dvourozměrného pole řešení.

Konkrétně si program na pozici odpovídající pořadovému číslu range bloku uloží tyto informace:

- pořadové číslo domain bloku

- parametry s a o
- pořadové číslo otoční a převrácení.

Program provede tento postup pro všechny range bloky. Jelikož tato aplikace slouží především k demonstraci fraktální komprese pro studenty, program si ukládá do matice řešení spíše pořadová čísla domain bloků, jelikož je pole domain bloků k dispozici. Pokud by měly být ovšem zakódované informace přenositelné, musel by si program ukládat souřadnice levého horního rohu domain bloku. Při dekódování by poté program musel, buďto sestavit seznam použitých domain bloků, aby z nich mohl počítat B bloky, nebo by musel při dekódování každého range bloku přepočítat hodnoty B bloku, které odpovídají domain boku nutnému k rozkódování tohoto range bloku. Obě tyto možnosti jsou však časově náročnější a nepřináší lepší demonstrační schopnosti aplikace.

Dekódování probíhá tak, že program nejprve nastaví obrázek na plně černý a v dvojrozměrném poli s hodnotami odstínů šedi uloží hodnotu 0 na všechny indexy. Poté iteračně dekóduje obrázek pomocí algoritmu, který je popsán výše. V první iteraci jsou všechny B bloky černé, a proto všechny pixely v rámci jednoho range bloku budou mít po první iteraci stejný odstín šedi. V druhé iteraci, jelikož pixely B bloku obecně nebudou mít stejný odstín šedi, pixely range bloků nebudou již mít stejný odstín šedi a začnou se pomalu vytvářet hrany. Při pozdějších iteracích se začnou vytvářet i plynulejší přechody a kolem patnácté iterace dojde program k atraktoru daného zakódování. Rozdíl mezi atraktorem zakódování a původním obrázkem můžete vidět v příloze na obrázcích 18 a 19.

Program je také rozšířen na kódování barevných obrázků. Tyto obrázky program kóduje obdobně jako obrázky v odstínech šedi. S tím rozdílem, že program barevným obrázkům kóduje odděleně složky R, G a B. Pro ukázku možných afinních transformací domain bloků program porovnává range bloky s domain bloky otočenými o 90º, 180º a 270º a také s horizontálně i vertikálně zrcadlově přetočenými domain bloky. Pomocí těchto rotací je možné dosáhnout menší průměrné lokální chyby za použití menšího počtu domain bloků.

Samotná implementace probíhá analogicky jako kódování odstínu šedi. Program si opět nejprve vytvoří pole. Tentokrát ovšem trojrozměrné o velikostech 512x512x3, kam si uloží hodnoty R, G a B všech pixelů. Oproti kódování do stupňů šedi se toto pole může zdát redundantní, ovšem volání funkce `Timage.canvas.pixels[i,j:integer]` je časově podstatně náročnější než čtení hodnot z pole. Dále program vytvoří range a domain bloky stejnými metodami jako pro kódování odstínu šedi. To znamená, že pro kódování všech tří složek program používá stejné domain bloky.

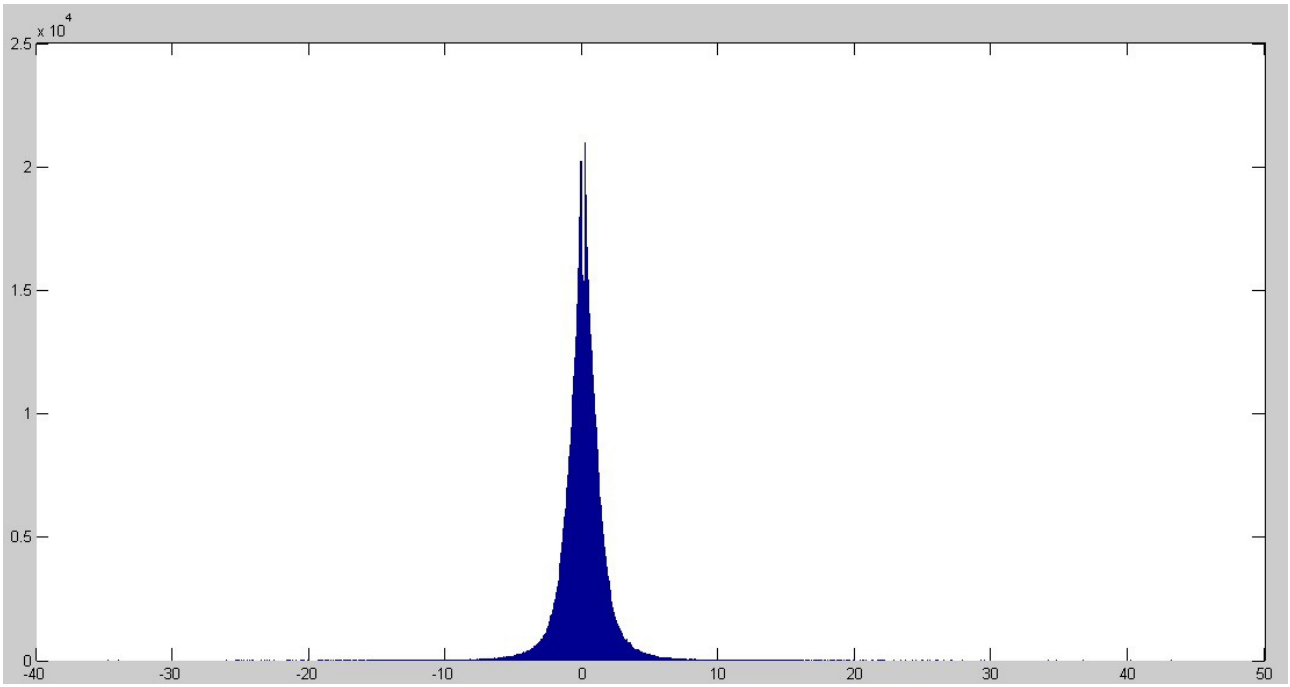
Program si poté vytvoří B bloky opět analogicky, tentokrát ovšem pro všechny tři složky. Samotné kódování je poté podstatně složitější, jelikož program kóduje range bloky pro všechny barevné složky a také pomocí všech transformací.

4.4.1 Velikost komprese

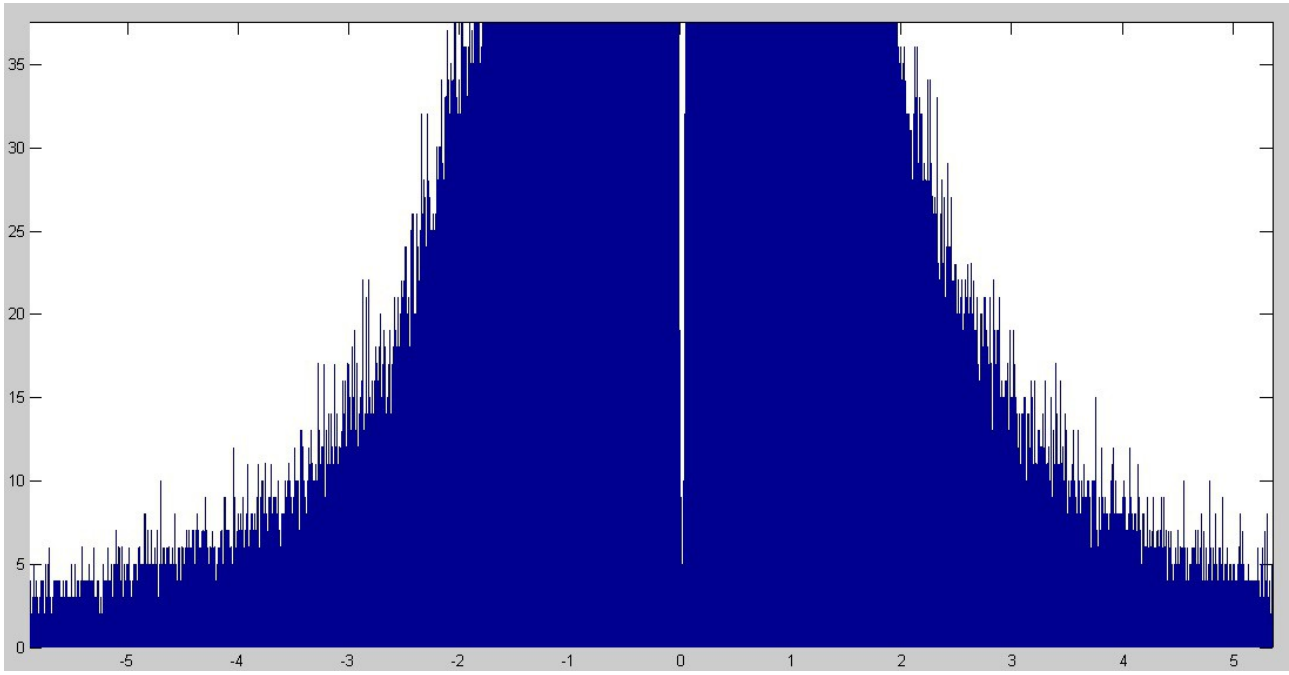
V literatuře se nejčastěji uvádí, že při rozměrech range bloků 4x4 má fraktální kódování kompresi 4:1. jelikož 16 pixelů, které jsou obecně uchovány v 16 bajtech, fraktální kódování ukládá do 4 bajtů. V Případě použití transformací se komprese změní na 3,2:1. V praxi je tato informace poměrně zavádějící, jelikož informaci o zakódování range bloků nemusí být možné uložit do 4 bajtů. Velikost informace o poloze domain bloků (souřadnice rohu nebo středu), záleží na velikosti zakódovaného obrazu. Demonstrační aplikace například kóduje obrázek o rozměrech 512x512, což znamená, že souřadnice domain bloku lze uložit do 2x9 bitů. V případě demonstrační aplikace lze volit souřadnice domain bloků pouze na sudých pixelech

4.4.2 Parametr *s*

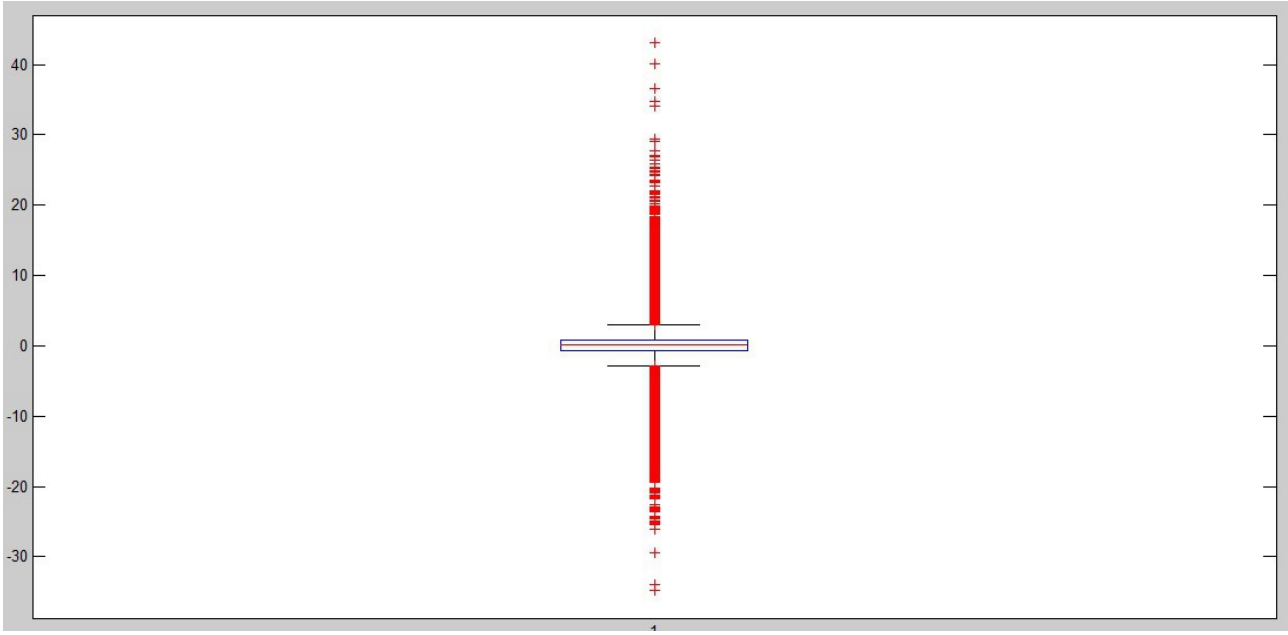
Parametr *s* je obecně reálné číslo, které nejčastěji nabývá hodnot kolem 0. V následujících obrázcích je zobrazen histogram a box-plot parametru *s* pro jedenáct různých zakódování obrázku Leny s různými domain bloky. Tato kódování byla provedena s range bloky o rozměrech 4x4, s 1000 domain bloky o rozměrech 8x8 a za použití afinních transformací. Jinými slovy, zkoumáme hodnoty parametru *s* ze zakódování 540 672 range bloků.



Obrázek 10: Histogram parametru s



Obrázek 11: Přiblížené okolí nuly histogramu parametru s



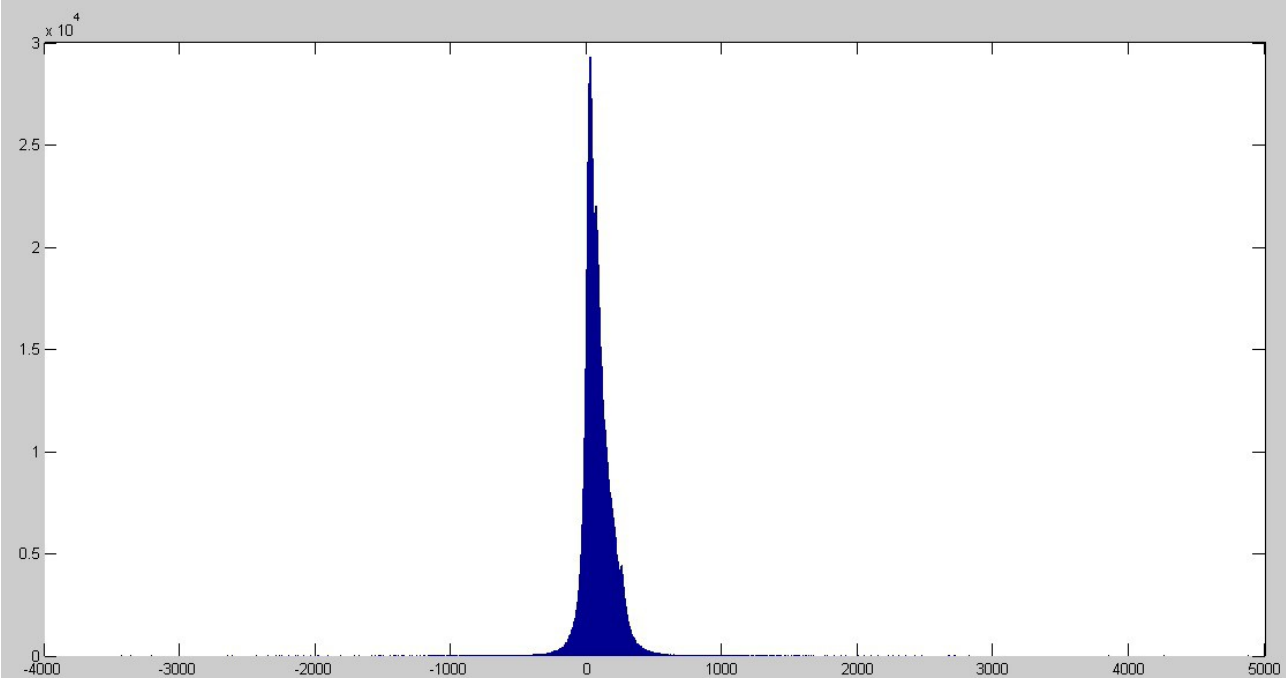
Obrázek 12: Boxplot analýza parametru s

Z histogramu je zřejmé, že hodnoty parametru *s* se nejčastěji pohybují kolem nuly, ovšem samotnou hodnotu nula nenabývají. Přesněji, z naměřených hodnot pouze jedna hodnota *s* splňovala vztah *abs(s)<0,006* a 46 hodnot splňovalo vztah *abs(s)<0,01*. Z boxplot grafu lze vyčíst, že hodnota

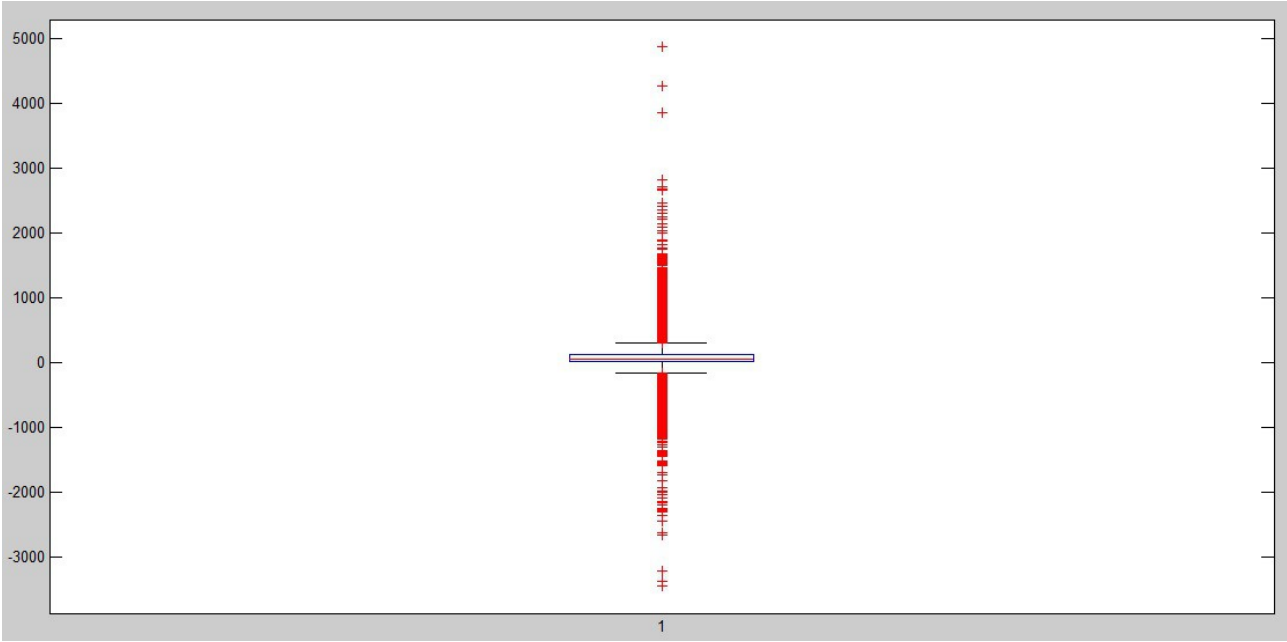
mediánu je 0,08, hodnota *25%* percentilu je -0,7 a hodnota *75%* percentilu je 0,76. Outliery jsou poté hodnoty, které jsou mimo interval <-2,91;2,96>. Ovšem samotný parametr *s* nabývá hodnot z intervalu <-34,8;43,14>. Z dalších měření vyplývá, že parametr *s* může nabývat i hodnoty mimo tento interval. Při ukládání parametru *s* na co nejmenší počet bitu nastávají dva problémy. Ukládání celé časti a ukládání desetinné části. Jelikož parametrem *s* při dekódování násobíme celočíselnými hodnotami z intervalu <0,255>, tak chyba menší než jeden pixel nastane při přesnosti parametru *s* na tři desetinná místa. Při přesnosti na dvě desetinná místa maximální chyba tohoto zaokrouhlení parametru *s* je 2,55 odstínu. Celou část lze redukovat pomocí znalostí o rozdělení parametru. Také lze hledat range bloku blok domain, který bude mít nejmenší lokální chybu a zároveň bude mít parametr *s* v intervalu <-1;1> (kvartily přibližně), popřípadě <-3;3> (outliery přibližně). Pokud budeme volit meze parametru *s* v intervalu <-1;1> a s přesností na dvě desetinná místa, lze tento parametr vynásobit hodnotou 100 a přičíst k výsledku hodnotu 128. Toto výsledné číslo lze uložit do jednoho bajtu. Ovšem průměrná lokální chyba se při implementaci těchto úprav zvýší z hodnoty 13,37 na hodnotu 14,63, a to pří stejných parametrech kódování jako při získávání dat.

4.4.3 Parametr *o*

Vlastnosti parametru *o* jsou zkoumány za stejných podmínek a stejnými metodami jako vlastnosti parametru *s*:



Obrázek 13: histogram parametrů o

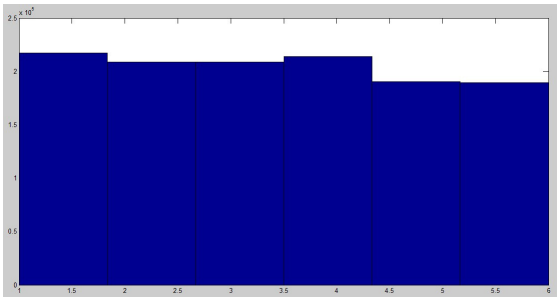


Obrázek 14: Boxplot analýza parametru o

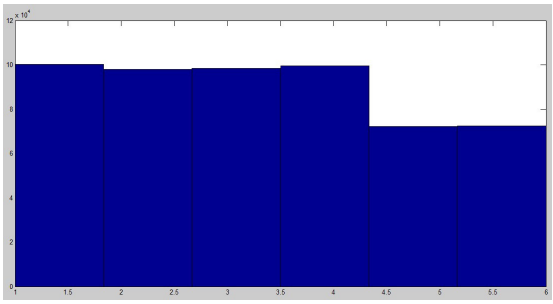
Z boxplot analýzy opět můžeme vyčíst, že hodnota mediánu parametru *o* je 57,88, hodnota 25% percentilu je 12,94 a hodnota 75% percentilu je 127,18. Outliery jsou hodnoty mimo interval <-158,43;298,551> a samotný parametr *o* nabývá hodnot v intervalu <-3442,25;4870,0437>. Oproti parametru *s*, u parametru *o* není tak významná desetinná část, jelikož zaokrouhlení zkreslí pixel maximálně o jednu hodnotu odstínu. Celou část opět můžeme omezit, například na interval <-180;330>, který je teoreticky možné uložit do 9bitů. Z důvodu tohoto omezení se průměrná lokální chyba zvýší z hodnoty 13,37 na hodnotu 13,6. Pokud implementujeme omezení na oba parametry, zvýší se průměrná lokální chyba na hodnotu 14,65.

4.4.4 Afinní transformace

Dále můžeme zkoumat vliv použití afinních transformací při kódování. Nejprve je dobré zjistit, jestli některá ze šesti možností není používána k zakódování více než ostatní možnosti transformace. Z různých měření vyplývá, že se nejčastěji nepoužije žádná transformace. Přesto je četnost případu, kdy není žádná transformace použitá, nižší, než by se dalo očekávat. Z měření dále vyplývá, že počet použití dalších transformací záleží na obrázku, který program zakóduje. Na následujících obrázcích lze vidět ukázky:



Obrázek 16: histogram použitých transformací pro obrázek Leny



Obrázek 15: Histogram použitých transformací pro obrázek opice

Informaci o použité transformaci lze uložit do tří bitů při použití šesti možných transformací, dále pak do dvou bitů při použití čtyř transformací nebo do jednoho bitu při použití dvou transformací a bez potřeby použití bitu, pokud budeme používat pouze identickou transformaci. Pokud použijeme pouze identickou transformaci, průměrná lokální chyba vzroste přibližně na hodnotu 15,54 (za stejných podmínek jako při předchozích měření), při použití dvou transformací se průměrná chyba změní na hodnotu 14,64, při použití čtyř transformací se změní na hodnotu 13,85 a při použití šesti transformací se opět změní na hodnotu 13,37.

Pokud program tedy zakóduje obrázek o rozměrech 512x512 pomocí range bloků o rozměrech 4x4 bez jakýchkoli omezení nebo cíleného zaokrouhlování, bude místo 16bajtů pro bmp obrázek ukládat následující bity:

- 9bitů pro *x* souřadnici domain bloku
- 9bitů pro *y* souřadnici domain bloku
- 4bajty pro *s* parametr
- 4bajty pro *o* parametr
- 2bity na pořadové číslo transformace

Celkem tedy 84bitů neboli 8,5bajtu, což znamená kompresi 1,9:1. Pokud zavedeme všechny výše zmíněné restrikce, kromě transformací, budou se ukládat následující informace:

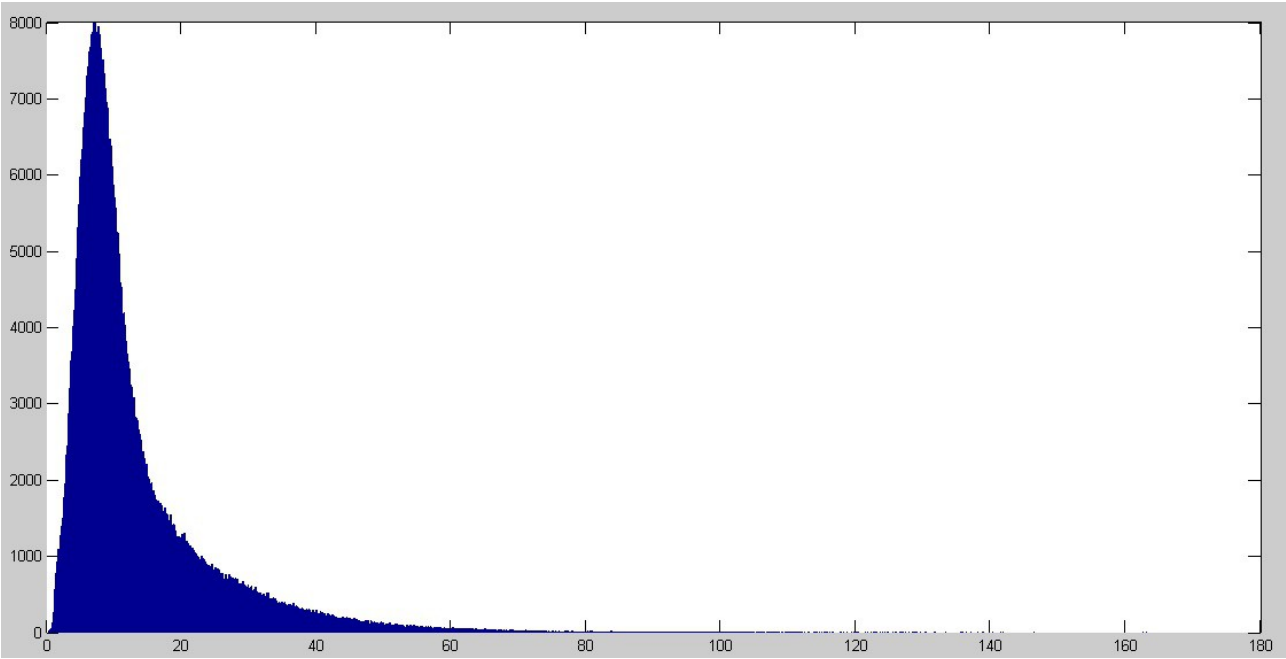
- 1bajt pro *x* souřadnici domain bloku
- 1bajt pro *y* souřadnici domain bloku
- 1bajt pro *s* parametr

- 9bitů pro *o* parametr
- 2bity na pořadové číslo transformace

Neboli 35 bajtů a komprese 3,6:1. Pro zakódování range bloků o velikosti 8x8 za použití těchto omezení dosáhneme komprese 14,6:1.

4.4.5 Kvantifikace chyby

Již několikrát byla v práci zmíněna průměrná lokální chyba. Lokální chyba je hlavním ukazatelem toho, jak dobře lze dvojici range a domain bloku zakódovat. Pokud se podíváme na vzorec pro výpočet lokální chyby, můžeme si povšimnout, že přes všechny pixely range bloku sčítáme čtverce rozdílu původního odstínu daného pixelu a odstínu, který bude pixel mít po jedné iteraci. Jinými slovy: zkoumáme, o kolik se obrázek změní po jedné iteraci daného zakódování. Opět lze tuto chybu statisticky zkoumat pomocí velkého množství naměřených dat. Histogram lokálních chyb zakódování půl milionu range bloků o rozměrech 4x4 z obrázku Leny vypadá takto:



Obrázek 17: Histogram lokálních chyb

Nejčastěji lokální chyba nabývá hodnoty přibližně 7. Medián nabývá hodnoty 9,52 a průměr má hodnotu 13,37. Průměr je značně ovlivněn extremními hodnotami, což ovšem není nežádoucí. Často se místo průměrné lokální chyby počítá se součtem lokální chyby, což má ovšem stejnou vypovídající hodnotu.

Další možností kvantifikace chyby zakódování je součet absolutních hodnot rozdílů odstínů pixelů

původního obrázku a odstínů pixelů atraktoru. Tuto hodnotu budeme nazývat absolutní chybou. Hlavní výhodou porovnávání odstínů pixelů původního obrazu a odstínů pixelů atraktoru je ten, že tento výpočet chyby není závislý na velikosti range bloků, tudíž je vhodnější pro porovnání dvou zakódování s různými rozměry range bloků. Lokální chyba je ovšem velmi úzce spjatá s rozměry range bloků, jelikož vždy sčítáme rozdíly odstínů všech pixelů range bloku. Tento součet bude tedy významně větší pro range bloky o větších rozměrech.

4.4.6 Range bloky

Jelikož range bloky musí pokrývat celý obrázek, tak počet range bloků je nepřímo úměrný rozměrům range bloků. Velikost range bloků také značně ovlivňuje poměr komprese a průměrnou lokální chybu. Například: pokud zvětšíme rozměry range bloků z velikosti 4x4 na velikost 8x8, zvýší se poměr komprese z 3,6:1 na 14,6:1. Průměrná lokální chyba se zvedne z hodnoty 13,37 na hodnostu 50. Hodnota absolutní chyby pro kódování s range bloky o rozměrech 4x4 je zhruba $3x10^6$, neboli každý pixel v atraktoru je průměrně o 4 odstíny odlišný od pixelu v původním obrázku. Při kódování pomocí range bloků o rozměrech 8x8 absolutní chyba vzroste přibližně na hodnotu $5x10^6$, neboli každý pixel v atraktoru je průměrně o 6 odstínů odlišný od pixelu v původním obrázku.

Rozměry a počet range bloků také ovlivní časovou náročnost kódování. Pokud se zvýší velikost range bloku, sníží se počet možných dvojic range a domain bloků, jelikož range bloků bude méně. Výpočet parametrů *i*, *o* a lokální chyby bude pro každou dvojici časově náročnější. Časová náročnost dekódování není ovlivněna rozměry range bloků.

4.4.7 Domain bloky

Vlastnosti zakódování lze ovlivnit jak rozměry, tak počtem domain bloků. Například pro range bloky o rozměrech 4x4 a domain bloky o rozměrech 8x8 s použitím všech afinních transformací, je vztah průměrné lokální chyby k počtu domain bloků následující:

Tabulka 3: průměrná lokální chyba v závislosti na počtu domain bloků

Počet domain bloků	Průměrná lokální chyba
10	20,48
50	17,37
100	16,34
250	14,97

500	14,14
1000	13,37
2500	12,49

Počet domain bloků poté významně ovlivňuje časovou složitost kódování, jelikož zvýšením množství domain bloků na dvojnásobek se zvedne počet možných dvojic na 36násobek. Každou dvojici totiž musíme otestovat pro 3 barevné odstíny a 6 možných transformací. Při dekódování záleží na počtu použitých domain bloků, pro které v každé iteraci budeme muset přepočítávat hodnoty B bloků. Počet použitých domain bloků ovšem záleží i na počtu range bloků. Demonstrační aplikace kóduje obrázky o rozměrech 512x512 pixelů, což znamená, že pokud range bloky budou mít rozměry 4x4, bude 16 384 range bloků pro jeden barevný kanál. To znamená 49 152 range bloků pro tři barevné kanály. Pravděpodobnost použití drtivé většiny domain bloků je tedy vysoká i při velkém počtu domain bloků. Například při tisíci domain blocích je průměrně použito 998,2 domain bloků.

Velikost domain bloků ovlivňuje opět jak velikost lokální chyby, tak časovou složitost kódování a dekódování. Pro range bloky o rozměrech 4x4 a za použití tisíce domain bloků je vztah mezi rozměry domain bloků a lokální chybou následující:

Tabulka 4: Průměrná lokální chyba v závislosti na rozměrech domain bloků

Rozměry domain bloků	Lokální chyba
4x4	13,02
8x8	13,37
16x16	13,25
32x32	13,23
64x64	13,32

Z měření pro obrázek Leny vyplývá, že nejvhodnější je volba domain bloků o rozměrech 4x4. Toto ovšem neplatí pro všechny obrázky obecně. S rostoucí velikostí domain bloků poté roste časová náročnost kódování a dekódování z důvodu vyšší časové náročnosti tvorby B bloků. V příloze můžete vidět různé atraktory kódování s různými rozměry a počtem domain bloků (obrázky 20-25).

4.4.8 Další měření

Všechna dosavadní měření byla prováděna na často používaném obrázku Leny, aby bylo možné porovnat vliv změn jednotlivých parametrů. Při testování demonstrační aplikace byla provedena i

další měření, ze kterých bylo zjištěno, že vliv parametrů na průměrnou lokální chybu je proporčně podobný jako u obrázku Leny.

Minimální lokální chyba byla naměřena u kódování obrázku gradientu, kde při kódování pomocí range bloků o rozměrech 4x4 a tisíce domain bloků o rozměrech 4x4 bylo dosaženo průměrné lokální chyby o hodnotě 0,8. Tento obrázek gradientu lze také zakódovat pomocí range bloků o rozměrech 32x32 a jednoho domain bloku o rozměrech 64x64 s absolutní chybou menší než 10^6. Atraktor tohoto zakódování můžete opět vidět v příloze (obrázek 28).I na kódováních odlišných obrázků si může uživatel povšimnout, že fraktální kódování je schopné vykreslit barevné přechody s velkou přesností.

Další zajímavé měření je možné zaznamenat kódování fotky, která zachycuje psa a dívku na louce. Tato fotka byla navíc upravena pomocí přechodového filtru HDR který zvýrazní hrany. U této fotky to znamená především zvýraznění srsti psa, vlasů dívky a stébel trávy. Tento obrázek fraktální kódování s range bloky o rozměrech 4x4 a tisíci domain bloky o rozměrech 8x8 provede s průměrnou lokální chybou 26,76. Při dekódování lze pozorovat, že nebe se dekóduje bez větších chyb oproti tomu tráva, srst a vlasy se dekódují s menší přesností. Tento jev lze pozorovat i u peří na obrázku Leny. Zde ovšem rozdíl oproti původnímu obrázku není tak velký, jelikož hrany nejsou tak ostré. Při 25 měřených kódování této fotky nabýval parametr *s* hodnot z intervalu <-421,6;592> a parametr *o* nabýval hodnot z intervalu <-28511,5;28702,85>. Hodnota absolutní chyby při kódování s range bloky o rozměrech 4x4 a tísíci domain bloky o rozměrech 8x8 nabývá hodnoty $5{,}3*10^6$ (obrázek 27 v příloze). Pokud změníme rozměry range bloků na velikost 8x8 a rozměry domain bloků na velikost 16x16, vzroste hodnota absolutní chyby na hodnotu $8{,}6*10^6$.

Kódování s velmi vysokou průměrnou lokální chybou bylo zaznamenáno u fotky útesů s mořem. Při kódování s range bloky o rozměrech 4x4 a tisíci domain bloky o rozměrech 8x8 byla průměrná lokální chyba 40,3 a absolutní chyba $7{,}8*10^6$ (pro range bloky o rozměrech 8x8 a domain bloky o rozměrech 16x16 nabývá absolutní chyba hodnoty$1{,}3*10^7$). Důvod je opět ve velkém množství ostrých hran na útesu, stejně jako ostré hrany odrazu ve zvlněné vodě. Naopak parametry *s* a *o* nabývají hodnot blíže k nule, než tomu bylo u obrázku Leny.

Fraktální kódování obrázků se tedy nejvíce vyplatí použít u obrázků s plynulými přechody barev a naopak se nevyplatí použít u obrázků s velkým počtem ostrých hran. Je také důležité dát si pozor na velikost ukládaných parametrů a případně, jak se změní vlastnosti kódování při implementaci různých omezení na parametry.

4.4.9 Využití fraktální komprese obrázků

V dnešní době se fraktální komprese používá pro komprimaci, jak obrázku, tak i videa. Společnost Iterated Systems Inc. Vyvinula v roce 2010 plugin jménem Genuine Fractals 6 pro Adobe Photoshop nebo Apple Apeture. Ten umožňuje ukládání a otevírání obrázků zakódovaných pomocí fraktální komprese. Tento plugin je schopný dosáhnout kompresního poměru 2:1 při bezztrátové kompresi a 5:1 při kompresi, která není vizuálně ztrátová. Další z funkcí Genuine Fractals 6 je možnost zvýšení rozlišení obrázku, což je operace, při které se vyplácí použít fraktální soběpodobnosti. Zachovává totiž ostré hrany a většinu detailů. Genuine Fractals 6 je schopný zvýšit rozlišení obrázku až 10krát při stejné kvalitě detailů. Nástupce Genuine Fractals 6, který se jmenuje Perfect resize 9.5 od stejné společnosti, je schopný zvýšit rozlišení obrázku až tisícinásobně při zachování kvality detailů. Tato funkce je populární pro snímky pořízené mobilním telefonem, které uživatel požaduje vytisknout ve vysokém rozlišení na vysoko formátový papír.

Fraktální komprese se také využívá pro ukládání satelitních snímků. Důvod je především v charakteru satelitních snímků, ve kterých se vyskytuje mnoho opakujících se prvků, stejně jako v nutnosti ukládání velkého množství satelitních snímků. Fraktální kódování optimalizované pro satelitní snímky je schopné kódovat například snímky pobřežních oblastí s kompresí až 172:1.

Dalším zajímavým použitím fraktální komprese je kódování snímků otisků prstů. Opět se jedná o situaci, kde je potřeba uložit velké množství snímků, které často vykazují soběpodobnost. Výhodou použití fraktální komprese pro tyto snímky je také možnost jejich porovnávání pomocí zakódované informace, tedy bez nutnosti dekódování.

Závěr

Cílem této práce bylo vytvoření uceleného e-learningového kurzu zabývajícím se fraktální geometrií. Fraktální geometrie je pojem, který v budoucnu určitě budeme slýchat čím dál častěji, a proto si myslím, že je nutné studenty seznámit se základními pojmy tohoto odvětví.

V teoretické části práce jsem se snažil popsat jak samotné fraktály, tak i jejich objev, který je podle mě velmi spjatý s výskytem fraktálů v mnoha vědních odvětvích. Oproti výskytu fraktálů, které může být zřejmé na první pohled, nemusí být jejich využití vždy tak očividné. To byl jeden z důvodů, proč jsem část svého práce věnoval možnostem jejich využití.

V praktické části se práce věnuje především implementaci tří demonstračních aplikací. První z aplikací vykresluje Juliovy množiny s možností smooth coloringu. Tato aplikace byla vytvořena především z toho důvodu, že takto vykreslené Juliovy množiny hrály velkou roli při zpopularizování fraktálů. Často jsou také první věcí, kterou si lidé vybaví ve spojitosti s fraktály.

Součástí e-learningu budou i aplikace, které jsem vytvořil v rámci ročníkového projektu, a které slouží k vykreslení různých fraktálů pomocí nejvýznamnějších algoritmů. Rozhodl jsem se proto, že vytvořím aplikace, které také předvedou možná využití fraktálů. První z těchto aplikací slouží k vykreslení fraktální křivky procházející zadanými body neboli fraktální interpolaci. Tato aplikace je zajímavá z toho důvodu, že používá velice chytře vlastnosti atraktoru. Zajímavé je také, že fraktální charakter výsledné křivky nemusí být na první pohled zřejmý, což může studenty motivovat k hledání fraktálů u křivek, se kterými se setkají v praxi.

Poslední demonstrační program, který jsem vytvořil, je program na demonstraci fraktálního kódování obrázků. Při studiu této metody jsem narazil na problém, že většina literatury byla buďto velmi obecná a popisovala pouze samotný algoritmus, nebo naopak popisovala implementaci velmi specifického druhu fraktálního kódování. Z tohoto důvodu jsem se rozhodl blíže zkoumat velikost komprese mé implementace, jelikož jsem nenašel literaturu, která by se tímto problémem více zabývala.

Seznam použité literatury

[1]: SOBOTA, Branislav a Ján MILIÁN. 1996. Grafické formáty. 1. vyd. České Budějovice: KOPP, 157 s. ISBN 80-858-2858-8.

[2]:ČANDÍK, Marek a Ján MILIÁN. 2005. Fraktálové kódovanie obrazov. 1. vyd. Ostrava: Oftis, 100 s. ISBN 80-900-7453-7.

[3]:ŽÁRA, Jiří, Bedřich BENEŠ a Petr FELKEL. 1998. Moderní počítačová grafika. Vyd. 1. Praha: Computer Press, xvi, 448 s. ISBN 80-722-6049-9.

[4]:Perfect Resize 9.5 - onOne Software. 2015. Award Winning Photography Software and Plug-ins - onOne Software [online]. [cit. 2015-05-12]. Dostupné z: http://www.on1.com/products/resize9/

[5]:Genuine Fractals 6 Review - PhotographyBLOG | PhotographyBLOG. 2009. PhotographyBLOG [online]. [cit. 2015-05-12]. Dostupné z: http://www.photographyblog.com/reviews_genuine_fractals_6.php

[6]:Wolfram Demonstrations Project. 2012. Wolfram Demonstrations Project [online]. [cit. 2015-05-12]. Dostupné z: http://demonstrations.wolfram.com/LorenzsWaterWheel/

[7]:Fractal Fluid Control - Technologies & Applications | Amalgamated Research LLC (ARi). 2014. Amalgamated Research LLC (ARi) [online]. [cit. 2015-05-12]. Dostupné z: http://www.arifractal.com/technologies-applications/fractal-fluid-control

[8]:Generating Random Fractal Terrain. 1997. No Normals [online]. [cit. 2015-05-12]. Dostupné z: http://www.gameprogrammer.com/fractal.html

[9]:Fractus Technology – fractal antenna technology for mobile and wireless telecoms. 2014. Fractus – world leader in fractal antenna IP and licensing MODIFIED [online]. [cit. 2015-05-12]. Dostupné z: http://www.fractus.com/index.php/fractus/technology/

[10]:EMS -- Nonlinear Geoscience (Fractals). Earth Monitoring Station (EMS) [online]. [cit. 2015-05-12]. Dostupné z: http://ems.gphys.unc.edu/nonlinear/fractals/moreexamples.html

[11]:WEE MENG WOON,, ANTHONY TUNG SHUEN HO, TAO YU, SIU CHUNG TAM, SIONG CHAI TAN a LIAN TECK YAP. 2000. Achieving high data compression of self-similar satellite images using fractal. IGARSS 2000. IEEE 2000 International Geoscience and Remote Sensing Symposium. Taking the Pulse of the Planet: The Role of Remote Sensing in Managing the

Environment. Proceedings (Cat. No.00CH37120) [online]. IEEE, : 609-611 [cit. 2015-05-12]. DOI: 10.1109/IGARSS.2000.861646. ISBN 0-7803-6359-0. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=861646

[12]:CROSS, Simon S. 1997. FRACTALS IN PATHOLOGY [online]. [cit. 2015-05-12]. DOI: 10.1002/(SICI)1096-9896(199705)182:1. Dostupné z: http://onlinelibrary.wiley.com/doi/10.1002/(SICI)1096-9896(199705)182:1%3C1::AID-PATH808%3E3.0.CO;2-B/epdf

[13]:ZELINKA, Ivan. 2006. Fraktální geometrie: principy a aplikace. 1. vyd. Praha, 159 s. ISBN 80-730-0191-8.

[14]:How Long is the Coastline of the Law: Additional Thoughts on the Fractal Nature of Legal Systems - Computational Legal Studies™Computational Legal Studies™. 2010. Computational Legal Studies™ -Computational Legal Studies™ [online]. [cit. 2015-05-12]. Dostupné z: http://computationallegalstudies.com/2010/10/how-long-is-the-coastline-of-the-law-additional-thoughts-on-the-fractal-nature-of-legal-systems/

[15]:COUFAL, Jaromír. 2010. Vizualizace fraktálů v komplexní rovině. Brno. Dostupné také z: http://is.muni.cz/th/251432/fi_b/COUFAL-Fraktaly_v_komplexni_rovine.pdf. Bakalářská práce. MASARYKOVA UNIVERZITA.

[16]:IFS and Fractal Coding. 2009. The waterloo fractal coding and analysis group [online]. [cit. 2015-05-12]. Dostupné z: http://links.uwaterloo.ca/ResearchIFSFractalCoding.html

[17]:RAMESH, GAYATRI. 2008. FRACTAL INTERPOLATION [online]. [cit. 2015-05-12]. Dostupné z: http://www.researchgate.net/profile/Gayatri_Ramesh/publication/47714303_FRACTAL_INTERPOLATION/links/0fcfd50ec6ade9cc0b000000.pdf

[18]:BOUBOULIS, Pantelis. 2012. Fractal Interpolation Theory and Applications in Image Compression. 1. Aufl. Saarbrücken: LAP LAMBERT Academic Publishing. ISBN 978-365-9251-665.

Obsah přiloženého CD

Přiložené CD obsahuje:

- Bakalářskou práci uloženou ve formátu PDF
- Demonstrační aplikace ve formátu spustitelných souborů .exe
- Obrázky ve formátu JPG které sloužili k demonstraci fraktální komprese
- Návod na ovládání aplikací formou textového souboru ve formátu .txt

Příloha

Obrázek 18: Původní černobílý obrázek Lena

Obrázek 19: Atraktor zakódování při range blocích o rozměrech 4x4 a 1000 domain blocích o rozměrech 8x8

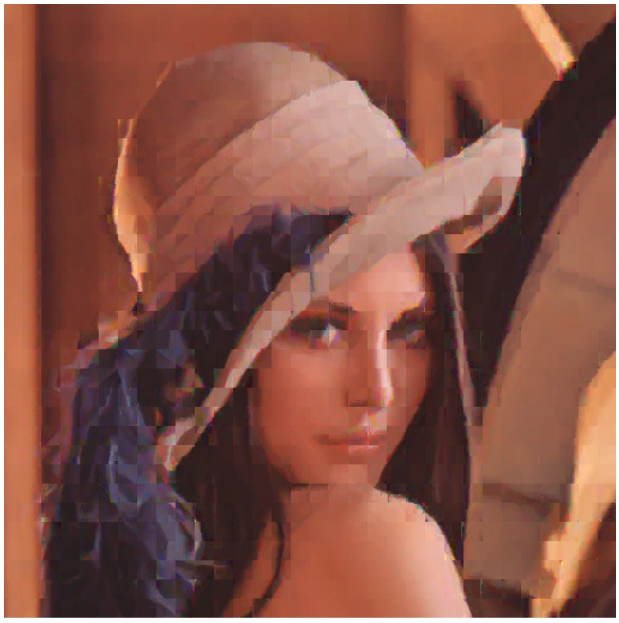
Obrázek 20: Původní obrázek Lena

Obrázek 21: Atraktor zakódování při range blocích o rozměrech 4x4 a 1000 domain blocích o rozměrech 8x8

Obrázek 23: Původní obrázek Lena

Obrázek 22: Atraktor zakódování při range blocích o rozměrech 8x8 a 500 domain blocích o rozměrech 16x16

Obrázek 24: Původní obrázek Lena

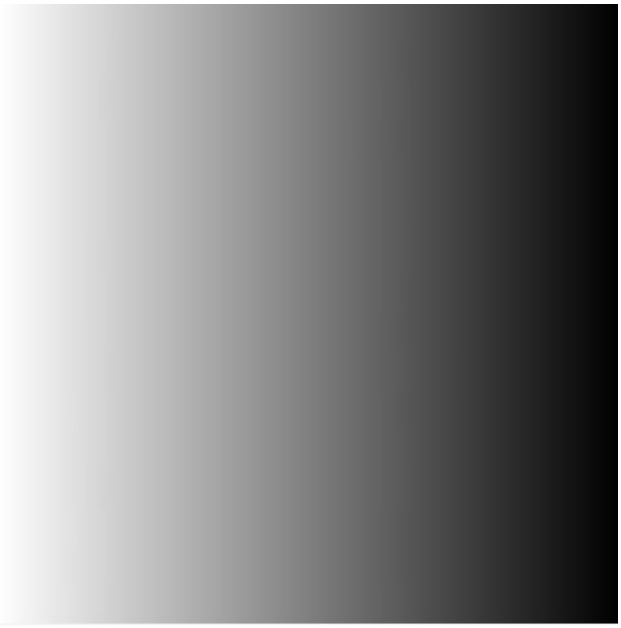


Obrázek 25: Atraktor zakódování při range blocích o rozměrech 16x16 a 500 domain blocích o rozměrech 32x32

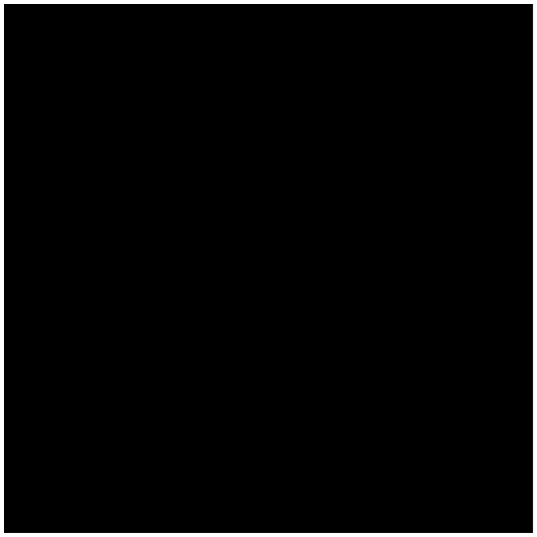
Obrázek 26: Původní fotka dívky se psem

Obrázek 27: Atraktor zakódování při range blocích o rozměrech 4x4 a 1000 domain blocích o rozměrech 8x8

Obrázek 29: Původní obrázek gradientu

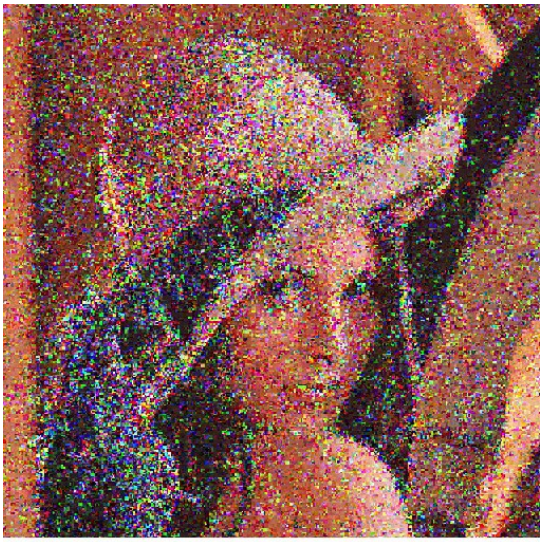


Obrázek 28: Atraktor zakódování při range blocích o rozměrech 32x32 a 10 domain blocích o rozměrech 64x64



Obrázek 30: 0. iterace

Obrázek 31: 1.iterace



Obrázek 33: 2. iterace

Obrázek 32: 3. iterace

Obrázek 35: 4. iterace

Obrázek 34: 5. iterace