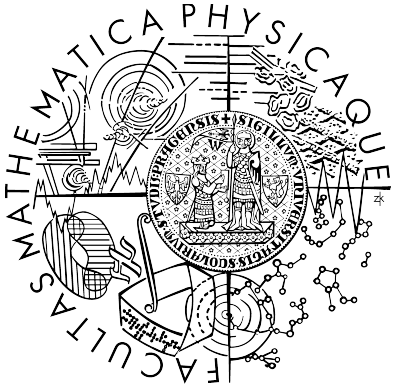
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Petr Štefan

Prohlížeč fraktálů

Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Mikuláš Patočka

Studijní program: Informatika

2006

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 09.08.2006 Petr Štefan

Obsah

Název Práce: Prohlížeč fraktálů
Autor: Petr Štefan
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: Mgr. Mikuláš Patočka
e-mail vedoucího: mikulas@artax.karlin.mff.cuni.cz

Abstrakt: Prohlížeč fraktálů je program, který umožňuje prohlížení několika typů fraktálů v čele s Madnelbrotovou množinou a umožňuje plynulé přibližování a oddalování zvolené oblasti. Program automaticky zvyšuje počet iterací pro výpočet tak, aby při zoomování nedocházelo ke ztrátě detailů. Prohlížeč fraktálů nabízí několik barevných schémat pro obarvení vypočteného fraktálu. Vybarvený obrázek lze modifikovat pomocí několika jednoduchých grafických filtrů. Výsledný obrázek lze uložit na pevný disk v některém z formátů PNG, BMP nebo JPG. Další funkcí programu je vyrenderování současného obrázku na pevný disk ve zvoleném rozlišení, kvalita výsledného obrázku je zvýšena pomocí antialiasigu.

Klíčová slova: fraktál, zoom, reálný čas

Title: Fractal explorer
Author: Petr Štefan
Department: Department of Software Engeneering
Supervisor: Mgr. Mikuláš Patočka
Supervisor's e-mail adress: mikulas@artax.karlin.mff.cuni.cz

Abstract: Fractal explorer is a program which enables user to explore Mandelbrot set and several other types of fractals and offers real time zooming in and out of the selected area. Program automaticaly increases the number of iterations it uses for computations to prevent detail loss with higher magnifications. Fractal explorer has several color schemes to color the calculated image. The colored image can be altered by several simple graphical filters. Resulting image can be saved to hard drive in PNG, BMP or JPG formats. Another function of the program is an ability to render current image to hard drive in specified resolution, the rendered image is antialiased to increase the visual quality.

Keywords: fractal, zoom, real time

Kapitola 1 – Úvod

1.1 Cíl

Cílem této bakalářské práce je vytvořit interaktivní prohlížeč fraktálů pro operační systém Windows, který by měl alespoň v některých ohledech překonat populární program XaoS. Vytvořený program by měl nabízet zejména následující funkce:

- zoomování v reálném čase – to znamená plynulé přibližování a oddalování zvolené oblasti fraktálu, uživatel volí oblast pomocí kurzoru myši
- posouvání fraktálu pomocí myši
- zvyšování přesnosti výpočtu – aby ani při velkém zvětšení nedocházelo ke ztrátě detailů, když to bude třeba program automaticky zvýší počet cyklů výpočtu
- vedle Mandelbrotovy množiny několik dalších typů fraktálů na výběr
- několik barevných schémat pro obarvení vypočteného fraktálu
- možnost upravit vybarvený obrázek pomocí jednoduchých grafických filtrů
- možnost uložit vygenerovaný obrázek v některém z běžných formátů jako PNG nebo BMP

Program by měl být dostatečně snadný na ovládání i pro méně zdatného uživatele, na druhé straně by měl zkušenějším uživatelům nabízet možnost měnit chování programu pomocí nastavení různých parametrů výpočtu.

1.2 Motivace

Hlavním důvodem pro vznik programu je dlouholetý zájem autora práce o problematiku fraktálů a snaha vytvořit netriviální nástroj pro jejich generování, který by mohl nabídnutými funkcemi soupeřit se zavedenými programy a nejlépe je alespoň v některých ohledech překonal.

1.3 Název

Místo původního trochu neobratného názvu „prohlížeč fraktálů“ byl pro aplikaci zvolen název anglický názvev „FractalDraw“. Dále bud používán jen anglický název.

Kapitola 2 – Algoritmy

2.1 Zoom

Hlavním cílem programu je nabídnout plynulé přibližování a oddalování zvolené oblasti fraktálu. Pro jeho splnění je třeba, aby byl program schopen generovat množství obrázků za sekundu. Kvůli značné výpočetní složitosti je proto nepřijatelné, aby aplikace počítala pokaždé všechny pixely výsledného obrázku. Takový přístup by mohl na běžném PC dát uspokojivé výsledky jen při nízkých rozlišeních jako 100 x 100, pro vyšší rozlišení je však nepoužitelný.

FractalDraw pro zrychlení výpočtu využívá pozorování, že není třeba počítat všechny pixely obrázku, neboť mnoho z nich již bylo vypočteno v průběhu generování předchozích obrázků. Většinou sice dříve vypočtené pixely nejsou přesně ty, jež jsou potřeba, ale pokud se jejich hodnota od hodnoty požadované liší méně, než je zvolená tolerance, lze tyto pixely použít v novém obrázku. Jedinou výjimkou je samozřejmě úplně první obrázek, který je třeba vypočíst celý.

Algoritmus použitý v aplikaci má několik fází – ukládání vypočítaných pixelů, aproximace nových pixelů starými, přesun starých pixelů na místo nových a výpočet pixelů, pro něž se nenašla vhodná aproximace.

Ukládáni vypočítaných pixelů – pro generování jednotlivých obrázků využívá program vždy pixelů předchozího obrázku. Pro další fázi je třeba znát přesné souřadnice každého pixelu, ale vzhledem ke způsobu, jakým aproximační algoritmus funguje nelze tyto hodnoty dopočítat a musejí být uloženy. Pokud byl totiž pixel v předchozím průchodu algoritmu nahrazen nějakým dříve vypočítamým, může se jeho skutečná souřadnice od očekávané hodnoty lišit. Protože všechny pixely v jedné řádce mají stejnou imaginární souřadnici a podobně všechny pixely v jednom sloupci mají stejnou reálnou souřadnici není třeba si pamatovat pro každý pixel dvojici reálná souřadnice, imaginární souřadnice, ale stačí si tedy pamatovat jen pro každý sloupec a řádku příslušnou složku komplexního čísla. Informace, které se ukládají jsou tedy souřadnice řádků a sloupců a pro každý pixel jeho hodnota, neboli počet iterací, který byl třeba k jeho výpočtu.

Aproximace nových pixelů starými – aby se snížil počet pixelů, které se nově počítají, aproximační algoritmus se pokusí nahradit pixely nového obrázku pixely již

dříve vypočítanými. Vzhledem k faktu, že všechny pixely v jedné řádce mají stejné imaginární souřadnice a všechny pixely v jednom sloupci mají stejné reálné souřadnice, není třeba aproximaci provádět pro každý jednotlivý pixel, ale lze jí provádět pro celý řádek nebo sloupec naráz. Algoritmus pracuje následujícím způsobem – pro každý sloupec nového obrázku zkusí najít takový sloupec ve starém obrázku, že se jeho reálná souřadnice liší od souřadnice, kterou by měl mít nový sloupec nejvýše o zvolenou toleranci. Právě proto, že pixely, kterými se nahrazuje nemusí mít přesně stejné souřadnice, jaké mají pixely, které se nahrazují, je třeba tyto hodnoty uložit, aby mohl algoritmus správně pracovat i v dalších krocích. Základní nastavení tolerance s kterou aproximační algoritmus řádky a sloupce nahrazuje je *0,5 * krok*, kde *krok* je vzdálenost mezi sousedními sloupci. Jestliže tedy má daný sloupec imaginární souřadnici *x* je možné ho nahradit už vypočítaným sloupcem z intervalu (*x - 0,5 * krok, x + 0,5 * krok*). Zcela analogicky se algoritmus pokusí najít náhradu za pro všechny řádky nového obrázku.

Po průběhu této fáze dojde k vytváření vlastního obrázku. Program začne v levém horním rohu postupně po řádcích vyplňuje jednotlivé pixely provedením některého ze dvou níže popsaných kroků.

Přesun starých pixelů na nové pozice – pokud pro daný pixel aproximační algoritmus nalezl náhradu, to znamená že pro sloupec i řádku, ve kterých se pixel nachází byla nalezena aproximace, je do nového obrázku zkopírován pixel ze starého obrázku.

Výpočet nových pixelů – pixely, pro něž nebyla nalezena vhodná aproximace je třeba vypočítat. Těchto pixelů může být stále relativně mnoho, proto aby program dále snížil jejich počet používá hádání hodnot některých pixelů. Tato metoda známá pod anglickým názvem *solid guessing* spočítá pixely na okraji zvoleného čtyřúhelníku, a pokud všechny pixely mají stejné hodnoty usoudí, že i pixely uvnitř čtyřúhelníku budou mít stejné hodnoty a tyto body tedy nejsou počítány. Program používá mírně modifikovanou metodu pro čtverec 3 x 3. Má-li program počítat pixel na souřadnicích *x* a *y* (zde se nejená o reálnou a imaginární část, ale o souřadnice pixelu v obrázku), pak v případě, že se nemají počítat sloupce *x - 1* a *x + 1* a řádka *y + 1*, zkusí program hodnotu pixelu uhodnout. Nastat mohou následující tři situace.

Má se počítat řádka *y*, nebo se má počítat sloupec *x*, nebo sloupec *x* i řádka *y* současně.

Pokud se má počítat řádka *y*, zkontroluje program pixely na pozicích [*x - 1*, *y - 1*], [*x*, *y - 1*], [*x + 1*, *y - 1*], [*x - 1*, *y*], [*x - 1*, *y + 1*], [*x*, *y + 1*] a [*x + 1*, *y + 1*]. Hodnoty těchto pixelů jsou známy, neboť už byly vypočteny, nebo pro ně aproximační algoritmus nalezl vhodnou náhradu. Toto je zajištěno požadavkem, aby sloupce *x - 1* a *x + 1* a řádka *y + 1* nebyly počítány. V opačném případě by totiž hodnoty pixelů [*x - 1*, *y + 1*], [*x*, *y + 1*] a [*x + 1*, *y + 1*] nebyly v tomto kroku algoritmu známy. Jestliže všechny tyto pixely mají stejnou hodnotu, pak i pixelu [*x*, *y*] je přiřazena tato hodnota a není počítán. (viz Obrázek 2.1)

	x - 1	x	x + 1	
y - 1	1	2	3	
y	4	**p**	?	
y + 1	5	6	7	

Obrázek 2.1: pokud pixely 1, 2, 3, 4, 5, 6 i 7 mají stejné hodnoty, pak i pixel p dostane přiřazenu tuto hodnotu.

Obdobně v případě, že se má počítat sloupec *x*, zkontroluje program pixely na pozicích [*x - 1*, *y - 1*], [*x*, *y - 1*], [*x + 1*, *y - 1*], [*x - 1*, *y*], [*x + 1*, *y*], [*x - 1*, *y + 1*], [*x + 1*, *y + 1*]. Jestliže se hodnoty všech pixelů rovnají, pak i pixelu [*x*, *y*] je přiřazena stejná hodnota a není počítán. (viz Obrázek 2.2)

	x - 1	x	x + 1	
y - 1	1	2	3	
y	4	**p**	5	
y + 1	6	?	7	

Obrázek 2.2: pokud pixely 1, 2, 3, 4, 5, 6 i 7 mají stejné hodnoty, pak i pixel p dostane přiřazenu tuto hodnotu.

Poslední možností je, že počítat se mají počítat jak sloupec *x*, tak řádek *y*. V tom případě program zkontroluje pixely na pozicích [*x - 1*, *y - 1*], [*x*, *y - 1*], [*x + 1*, *y - 1*], [*x - 1*, *y*], [*x - 1*, *y + 1*], [*x + 1*, *y + 1*]. K disopozici je tedy o jeden pixel méně, než v předchozích případech. Opět, pokud zkoumané pixely mají všechny stejnou hodnotu není pixel [*x*, *y*] počítán a je mu přiřazena tato hodnota. (viz Obrázek 2.3)

	x - 1	x	x + 1	
y - 1	1	2	3	
y	4	**p**	?	
y + 1	5	?	6	

Obrázek 2.3: pokud pixely 1, 2, 3, 4, 5 i 6 mají stejné hodnoty, pak i pixel p dostane přiřazenu tuto hodnotu.

Samozřejmě se často stane, že zkoumané pixely nemají všechny stejnou hodnotu a v tom případě nezbývá, než hodnotu pixelu opravdu vypočítat.

Výše popsaný algoritmus pracuje stejně pro přibližování i oddalování.

2.2 Posouvání

Problém posouvání lze řešit podobným přístupem, jako zoomování. Opět lze použít pixely z předchozího obrázku a tím ušetřit čas, který by byl jinak potřeba k jejich výpočtu. Je samozřejmě třeba zajistit, aby posouvání nijak nenarušilo chod algoritmu zoomování, tedy korektně uložit hodnoty pixelů a souřadnice sloupců a řádků. Proto je pro posouvání použit jen mírně modifikovaný algoritmus pro zoomování, takže algoritmus pro posouvání provádí stejné kroky jako algoritmus zoomování popsaný výše.

2.3 Efektivita algoritmů zoomování a posouvání

Neboli kolik výpočtů je ušetřeno v algoritmech zoomování a posouvání díky použití dříve spočítaných pixelů a díky hádání pixelů.

U zoomování je tento počet ovlivněn několika faktory. Za prvé je rozdíl, zda jde o přibližování nebo o oddalování. Dále záleží, jak moc jednolitá je zkoumaná část fraktálu. Algoritmus hádání ušetří tím více výpočtů, čím větší budou oblasti, ve kterých mají všechny pixely stejnou hodnotu. Jestliže se tedy ve výsledném obrázku nenachází žádná oblast velikosti alespoň 3 x 3, ve které mají všechny pixely stejnou hodnotu, nejde uhodnout barvu žádného pixelu a všechny se musejí počítat. To je samozřejmě ovlivněno i typem fraktálu, některé takových oblastí obsahují více (například fraktál spider), jiné méně (například fraktál barnsley1j).

Při rozlišení 500 x 400 je celkový počet pixelů obrázku 200 000. Počet pixelů použitých z předchozího obrázku se pohybuji při přibližování i oddalování kolem 170 000. Úspora tedy činí kolem 85 %. Další výpočty jsou ušetřeny hádáním hodnot pixelů. Zde se už hodnoty pro přibližování a oddalování liší.

Při přibližování totiž vzniká množství mezer o šířce jednoho pixelu mezi jednotlivými sloupci a řádky tam, kde aproximační algoritmus nenalezl náhradu, ale

kde našel náhradu pro sousední řádky a sloupce. V takových situacích může hádání pixelů výrazně pomoct. Konkrétní počty uhodnutých pixelů se nejčastěji pohybují mezi 5 000 a 25 000. Úspora se tedy pohybuje mezi 2,5 a 12,5 %. Pixelů které se počítají tedy může být občas jen kolem 5 000, tedy 2,5 %, obvykle je jich však více, někde okolo 12 000, což činí asi 6 %.

Při oddalování se většina řádek a sloupců, pro které se nenašla aproximace nacházejí u okrajů obrázku a navíc tvoří shluky, které neumožňují použití hádání pixelů. Proto je celkový počet počítaných pixelů trochu vyšší, než u přibližování. Počet uhodnutých pixelů se nejčastěji pohybuje mezi 5 000 a 10 000, což činí 2,5 až 5 %. Počítáno je tedy něco 20 000 a 25 000 pixelů, což činí 10 až 12,5 %. Oddalování tedy vyžaduje asi dvojnásobný počet výpočtů oproti přibližování.

U posouvání záleží v podstatě pouze na tom, o kolik pixelů se posouvá. Vypočíst totiž bude třeba celou část obrázku, která dříve na obrázku nebyla. Jestliže totiž dojde k posunu například o 1 pixel směrem dolů, bude třeba vypočítat celou horní řádku. Při posunu o 10 pixelů směrem dolů, bude třeba vypočítat 10 horních řádek. Zbytek obrázku naštěstí počítat třeba není, ten se vytvoří přesunutím pixelů předchozího obrázku. Při posouvání se však téměř nedostane ke slovu hádání pixelů. Specifické situace, které hádání vyžaduje k tomu, aby se mohlo použít (viz obrázky 2.1, 2.2 a 2.3) totiž při posouvání téměř nenastávají. Nově počítané řádky a sloupce totiž tvoří shluky a tak jediným případem, kdy jde hádání použít je situace, kdy dojde k posunu směrem dolů alespoň o 2 pixely. V takovém případě je možné hádání aplikovat na poslední nově přidanou řádku, takže tato metoda není tak užitečná jako u zoomování. Přesto nebylo hádání pixelů z algoritmu posouvání odstraněno. Počet pixelů, které se nově počítají se tak může pohybovat mezi počtem pixelů jedné řádky nebo jednoho sloupce až ke všem pixelům obrázku.

2.4 Zvyšování počtu iterací

Při dostatečném zvětšení dochází ke ztrátě detailů. To je způsobeno použitím konečného počtu iterací pro výpočet. Aby se tomuto předešlo, je třeba plynule zvyšovat počet iterací a tím i detailnost výsledného obrázku. Nejdůležitějším

problémem je poznat, kdy je třeba zvýšit počet iterací aby se toto zvýšení projevilo ve výsledku.

FractalDraw řeší tento problém tak, že si počítá, kolik pixelů na obrázku má hodnotu menší než je maximální počet iterací k výpočtu (dále jako maxiter). Dále si počítá kolik pixelů má hodnoty v intervalu <maxiter - 5, maxiter - 1>. Spočítá se poměr těchto dvou čísel, a pokud vyjde menší, než je zadaná kritická hodnota, zvýší se číslo maxiter o 1. Příliš velká kritická hodnota by vedla k tomu, že by ke zvyšování iterací došlo příliš brzy a na výsledném obrázku by se to neprojevilo vůbec nebo jen velmi málo a zbytečně by se tím zpomaloval výpočet. Příliš malá hodnota by naopak mohla vést k tomu, že ke zvyšování dojde příliš pozdě nebo dokonce vůbec nenastane. V programu je tato hodnota přednastavená na 25, neboli pokud počet pixelů s hodnotou menší, než maxiter děleno počet pixelů s hodnotou v intervalu <maxiter - 5, maxiter - 1> vyjde menší, než 25, pak dojde ke zvýšení čísla maxiter. FractalDraw umožňuje uživateli nastavit si vlastní kritickou hodnotu, může být větší i menší, než je přednastavená hodnota.

Kvůli zvyšování čísla maxiter je třeba mírně upravit algoritmus pro zoomování. V případě zvýšení počtu iterací by totiž došlo ke špatnému vybarvení pixelů, které mají hodnotu maxiter a které byly na základě práce aproximačního algoritmu přesunuty na nové pozice. Po zvýšení počtu iterací se jejich hodnota bude totiž rovnat maxiter - 1 a obarvovací algoritmus (viz 4.2) by těmto pixelům přiřadil chybnou barvu. Je proto třeba těmto pixelům zvýšit jejich hodnotu o 1, tak, aby se opět rovnala hodnotě maxiter.

Kapitola 3 – Fraktály

3.1 Úvod

V této kapitole jsou popsány fraktály, které jsou ve FractalDraw na výběr. U každého fraktálu je vypsána formule, pomocí níž je fraktál vytvořen. Dále je připojena podmínka omezující absolutní hodnotu čísel Z(n) tzv. bailout value. Výsledkem funkcí je nezáporné celé číslo, které znamená, po kolika iteracích hodnota Z(n) tuto podmínku porušila, nebo číslo maxiter, pokud i po maximálním počtu iterací číslo Z(n) tuto podmínku neporušilo.

V dalším textu `pixel` je komplexní číslo odpovídající reálné a imaginární souřadnici daného pixelu.

3.2 Mandelbrot

Mandelbrotova množina je nejznámnější fraktál, je vytvořena pomocí následující formule.

```
Z(0) = C = pixel
Z(n+1) = Z(n)^2 + C
|Z(n)| < 2
```

3.3 Mandelbrot4

Mandelbrotova množina čtvrté úrovně, číslo Z(n) je na rozdíl od klasické Mandelbrotovy množiny umocněno na čtvrtou.

```
Z(0) = C = pixel
Z(n+1) = Z(n)^4 + C
|Z(n)| < 2
```

3.4 Spider

Fraktál, který dostal jméno podle svého tvaru, připomínající pavouka

```
Z(0) = C(0) = pixel
Z(n+1) = Z(n)^2 + C(n)
C(n+1) = C(n) / 2 + Z(n+1)
|Z(n)| < 2
```

3.5 Sierpinsky

Sierpinského trojúhelník.

```
Z(0) = pixel
if(imag(Z(n)) > 0.5)
   Z(n+1) = (2 * real(Z(n)), 2 * imag(Z(n)) - 1)
else
   if(real(Z(n)) > 0.5)
        Z(n+1) = (2 * real(Z(n)) - 1, 2 * imag(Z(n)))
   else
        Z(n+1) = (2 * real(Z(n)), 2 * imag(Z(n)))
|Z(n)| < 127
```

3.6 Phoenix

```
Z(0) = pixel, Y(0) = 0
Z(n+1) = Z(n)^2 + p1 + p2 * Y(n)
Y(n+1) = Z(n)
p1 = 0.566666...
p2 = -0.5
|Z(n)| < 2
```

3.7 Barnsely1j

Zvláštní pojmenování vychází z konvence programu FractInt.

```
Z(0) = pixel;
if real(Z(n)) >= 0
   Z(n+1) = (Z(n) - 1) * C
else
   Z(n+1) = (Z(n) + 1) * C
real(C) = 0.6
imag(C) = 1.1
|Z(n)| <  2
```

3.8 Newton3

```
Z(0) = pixel
Y(n) = Z(n)
Z(n+1) = ((2 * Z(n)^3 + 1) / (3 * Z(n)^2))
|Z(n) - Y(n)| >= 0.00001
```

Kapitola 4 – Grafické filtry

4.1 Úvod

Výsledkem práce fraktálních algoritmů je přiřazení nezáporné celočíselné hodnoty každému pixelu obrázku. Jedná se o tzv. escape-time neboli počet iterací, kolik potřeboval program k tomu, aby usoudil, že pixel nepatří do počítaného fraktálu. Pro zobrazení výsledku na monitoru je třeba pixely nějakým způsobem vybarvit.

4.2 Vybarvení obrázku

Pro základní vybarvení obrázku používá FractalDraw jednoduchý algoritmus. Pixelům, jejichž hodnota se rovná maximálnímu počtu iterací je přiřazena černá barva. Ostatním pixelům je přiřazena barva z připravené barevné palety – seznamu barev pevné délky. Pokud je hodnota pixelu menší nebo rovna velikosti barevné palety, je mu přiřazena barva z tohoto seznamu na pozici odpovídající jeho hodnotě. Když je jeho hodnota větší, pak je mu přiřazena barva na pozici hodnota modulo velikost palety.

FractalDraw nabízí 10 barevných palet na výběr, každá z nich obsahuje 256 barev. Palety jsou vytvořeny tak, aby se po sobě následující barvy příliš nelišily a tím se předešlo ostrým barevným přechodům ve výsledném obrázku.

4.3 Rotace barev

Rotace barev je efekt, který při přibližování a oddalování fraktálu mění barvy jednotlivých pixelů tak, že pokud obarvovací algoritmus přiřadil pixelu barvu na pozici n, tak v dalším obrázku dostane pixel přiřazenu barvu $n + 1$. V dalším obrázku by tento pixel dostal barvu na pozici $n+2$ at tak pořád dále.

Rotace barev si může uživatel vypnout i zapnout, dále program nabízí možnost odstranit barevný posun, vzniklý použitím rotace barev, takže pokud po x krocích se

zapnutou rotací dostal pixel barvu $n + x$, odstraněním posunu dostane pixel barvu na pozici n.

4.4 Jednoduché filtry

Vybarvený obrázek lze dále upravit pomocí několika grafických filtrů. Nejjednodušší filtry vypočítají novou barvu pixelu jen na základě jeho stávající barvy.

FractalDraw nabízí dva takové filtry – invert, který invertuje barvy všech pixelů a grayscale, který obrázek převede do stupňů šedé barvy.

Filtr invert pixelu, jehož barva byla po složkách (červená, zelená a modrá barva) r, g, b přiřadí tomuto pixelu barvu 255 - r, $255 - g$, $255 - b$. Tím pixel získá opačnou barvu, než měl původně.

Filtr grayscale pixelu s barvou r, g, b přiřadí hodnotu $0.299 * r + 0.587 * g + 0.115 * b$ všem barevným složkám a tím převede obrázek do stupňů šedé barvy.

Vzhledem k tomu, že oba filtry načtou každý pixel obrázku jen jednou a proto pracují relativně rychle, FractalDraw umožňuje použití obou dvou filtrů naráz.

4.5 Složitější filtry

Složitější filtry vypočítají novou barvu pixelu nejen na základě jeho staré barvy, ale také na základě barvy okolních pixelů. Níže popsané filtry fungují na stejném principu, jsou to konvoluční filtry velikosti 3 x 3 (angl. convolution filter 3 x 3).

Konvoluční filtry fungují tak, že barva daného pixelu a jeho sousedů (v případě filtru 3 x 3 je sousedů 8) vynásobí určitými koeficienty, výsledky se sečtou vydělí jiným koeficientem a případně se k nim přičte daný ofset. Výjimku tvoří pixely na krajích obrázku, které nemají dostatek sousedů a proto je jim přiřazena nějaká barva, nejčastěji černá. Koeficienty, kterými se násobí barvy pixelů se pro názornost většinou zapisují pomocí matice. Na obrázku 4.1 je konvoluční filtr, který obrázek nijak nezmění, jde o identitu. Hodnota pixelu je vynásobena 1, hodnoty ostatních

pixelů jsou násobeny 0 takže vyjdou všechny 0, po sečtení hodnot je toto číslo vyděleno 1 a je k němu přičten ofset 0, takže výsledkem je původní hodnota pixelu.

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} / 1 + 0$$

Obrázek 4.1: 3 x 3 konvoluční filtr, který obrázek nezmění

FractalDraw nabízí čtyři přednastavené filtry, ale také možnost nastavení vlastních hodnot filtru. Přednastavené filtry se jmenují edge detection, emboss, sharpen a gaussian blur.

Edge detection – neboli detekce hran zvýrazní hranice oblastí s rozdílnými barvami. Čím rozdílnější tyto barvy jsou, tím výraznější jsou hrany na výsledném obrázku. Matice filtru je znázorněna na obrázku 4.2.

$$\begin{bmatrix} -1 & 0 & -1 \\ 0 & 4 & 0 \\ -1 & 0 & -1 \end{bmatrix} / 1 + 127$$

Obrázek 4.2: filtr edge detection

Emboss – tento filtr také zvýrazňuje hrany, vytváří pseudo 3 D efekt. Podobně jako u filtru edge detection jsou výsledky výraznější, čím odlišnější barvy mají sousední oblasti. Matice filtru je na obrázku 4.3.

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} / 1 + 127$$

Obrázek 4.3: filtr emboss

Gaussian blur – výsledkem práce tohoto filtru je rozmlžený nebo rozostřený obrázek. Hodnoty, se kterými filtr pracuje jsou na obrázku 4.4.

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} / 16 + 0$$

Obrázek 4.4: filtr gaussian blur

Sharpen – poslední přednastavený je opakem filtru gaussian blur, filtr sharpen obrázek naopak zostří. Jeho matice je na obrázku 4.5.

$$\begin{bmatrix} 0 & -2 & 0 \\ -2 & 11 & -2 \\ 0 & -2 & 0 \end{bmatrix} / 3 + 0$$

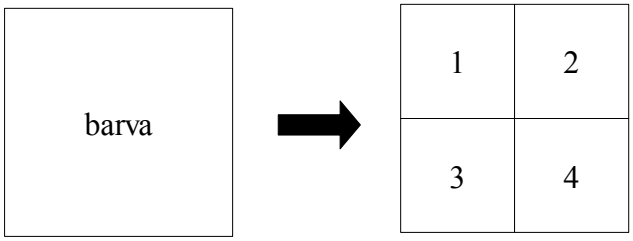
Obrázek 4.5: filtr sharpen

FractalDraw dále umožňuje uživateli nastavit hodnoty koeficientů a tím definovat vlastní filtr.

Vzhledem k větší výpočetní náročnosti je povoleno použití nejvýše jednoho z výše popsaných filtrů. Tyto složitější filtry lze libovolně kombinovat s jednoduchými filtry invert a grayscale popsanými v části 4.3.

4.6 Antialiasing

Jednou z funkcí, kterou FractalDraw nabízí je možnost vyrenderovat obrázek na disk v libovolném rozlišení. Pro zvýšení kvality výsledného obrázku provede program vyhlazení hran – antialiasing. Metoda, kterou FractalDraw používá je známá pod názvy jako oversample nebo supersample. Spočívá v tom, že pro určení výsledné barvy pixelu, je pixel rozdělen na několik menších pixelů, vypočteny jsou barvy těchto vzorků a výsledná barva je určena jako aritmetický průměr těchto barev. (viz Obrázek 4.6)

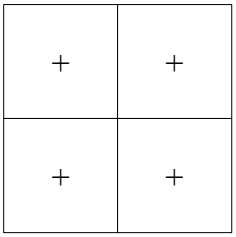


Obrázek 4.6: barva pixelu se vypočte jako součet barev pixelů 1, 2, 3 a 4 dělený jejich počtem, v tomto případě tedy 4

Kvalitu výsledného obrázku ovlivňuje počet vzorků, které se pro určení barvy pixelu použijí. Na obrázku 4.6 jsou to 4 vzorky, FractalDraw dále umožňuje zvolit počet vzorků jako 9 a 16, kdy se pixel rozdělí na 3 x 3 respektive 4 x 4 subpixely. Čím větší je počet vzorků, tím lepší je konečný výsledek, na druhé straně více vzorků znamená pomalejší výpočet.

Existuje několik druhů supersamplingu, které se liší podle toho, odkud z vnitřku pixelu se vezmou souřadnice vzorků. FractalDraw nabízí na výběr dva druhy, a to grid a jitter.

Grid neboli mřížka je nejjednodušší algoritmus, který pixel rozdělí na několik částí a souřadnice vzorků vezme ze středu každé části. (viz Obrázek 4.7)

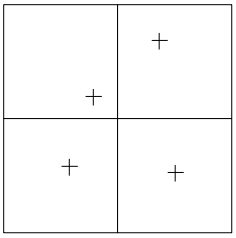


Obrázek 4.7: supersampling metoda grid

FractalDraw tuto metodu implementuje následovně: pokud má počítaný pixel souřadnice x a y a pokud je vzdálenost mezi sousedními pixely *krok*, pak za předpokladu použití 4 vzorků jsou spočítány barvy pixelů se souřadnicemi $[x, y]$, $[x, y + krok / 2]$, $[x + krok / 2, y]$ a $[x + krok / 2, y + krok / 2]$ a výsledná barva pixelu je určena jako aritmetický průměr těchto 4 barev. Jestliže je vzorků 9, pak se spočítají barvy pixelů na souřadnicích $[x, y]$, $[x + krok / 3, y]$, $[x + 2 * krok / 3, y]$, $[x, y + krok / 3]$, $[x + krok / 3, y + krok / 3]$, $[x + 2 * krok / 3, y + krok / 3]$,

[*x*, *y* + 2* *krok* / 3], [*x* + *krok* / 3,*y* + 2* *krok* / 3], [*x* + 2* *krok* / 3 , *y* + 2 * *krok* / 3] a výsledná barva je určena jako aritmetický průměr těchto 9 barev. Výpočet pro 16 vzorků je zcela analogický.

Metoda jitter podobně jako metoda grid rozdělí pixel na několik částí, ale souřadnice vzorků vezme z libovolného místa uvnitř každé části. (viz Obrázek 4.8)



Obrázek 4.8: supersampling metoda jitter

Implementace této metody je v podstatě stejná jako u metody grid, jediný rozdíl je, že ke každé souřadnici každého vzorku je připočten náhodně zvolený posun po reálné a imaginární ose z intervalu <*0*, *krok* / *2*> pro 4 vzorky, z intervalu <*0*, *krok* / *3*> pro 9 vzorků a z intervalu <*0*, *krok* / *4*> pro 16 vzorků.

Metoda jitter by údajně měla dávat lepší výsledky, než metoda grid, protože rozložení vzorků v pixelu není tak pravidelné, nicméně dle subjektivního dojmu autora vycházejí hezčí obrázky při použití metody grid.

Kapitola 5 – Implementace

5.1 MFC

Grafické rozhraní programu je napsáno pomocí tříd Microsoft Foundation Classes (MFC). Použití těchto tříd je jednodušší než psát přímo ve Win32 API a umožňuje lépe se soustředit na funkce programu. Výhodné je zejména při tvorbě dialogů, kde například metoda DoDataExchange kontroluje uživatelem zadané hodnoty a automaticky zobrazí varování, pokud se uživatel dopustil chyby.

5.2 CImage

Na třída reprezentující obrázek uvnitř programu bylo kladeno několik požadavků. Měla by umožňovat uchovávat barvy jednotlivých pixelů ve formátu RGB. Dále by měla umožňovat přímý přístup k jednotlivým pixelům a možnost přímo číst a zapisovat barvy libovolného pixelu. To proto, aby šla snadno použít v grafických filtrech. Mělo by být možné ji přímo použít pro kreslení v handleru hlavního okna OnPaint bez nutnosti nějaké konverze. A měla by nabízet možnost uložit uchovávaný obrázek na pevný disk ve formátech PNG, BMP a JPG.

Pro reprezentaci obrázku uvnitř programu byla zvolene třída CImage, protože těmto požadavkům nejlépe odpovídá. Pomocí jednoho z parametrů její metody Create lze nastavit počet bitů na pixel, FractalDraw používá 24 bitů. Pro přístup k pixelů je používána metoda GetBits(), která vrací ukazatel na pole bajtů, ve kterém jsou uloženy hodnoty barevných složek jednotlivých pixelů v pořadí modrá, zelená a červená barva. Jedná se o jednorozměrné pole, kde vzdálenost mezi sousedními řádky určuje číslo pitch, které se dá zjistit pomocí metody GetPitch(). Toto číslo se obecně nemusí rovnat šířce obrázku násobené počtem bajtů na pixel. Ukládání obrázků na pevný disk má na starost metoda Save, pomocí parametru lze nastavit formát ukládaného souboru jako PNG, BMP nebo JPG, drobnou nevýhodou je, že není možné nastavit kvalitu komprese pro formáta JPG. Schopnost ukládat obrázek na pevný disk byl hlavním důvodem proč před podobnou třídou CBitamap, dostala přednost právě třída CImage.

5.3 CHlavniOkno

Třída CHlavniOkno je potomkem MFC třídy CFrameWnd a reprezentuje hlavní okno aplikace. Je popsána v souborech HlavniOkno.h a HlavniOkno.cpp. Pomocí handlerů obsluhuje události hlavního okna jako výběr položky hlavního menu, změna velikosti okna, kliknutí myši atd. CHlavniOkno obsluhuje i zoomování a posouvání fraktálu, které mají na starosti handlery OnZoom a OnPosun.

5.4 CFractalApp

Třída CFractalApp je potomkem MFC třídy CWinApp a reprezentuje aplikaci. Hlavní okno aplikace je vytvořeno v metodě InitInstance. V metodě Run se nachází nekonečná smyčka, která se stará o zpracovávání zpráv. Metoda Run je mírně předefinovaná tak, že pokud je třeba provést zoomování nebo posouvání, přidá do fronty zpráv zprávu pro hlavní okno, že je třeba příslušnou akci provést. Zdrojové kódy třídy CFractalApp se nacházejí v souborech FractalApp.h a FractalApp.cpp.

5.5 CFractal

Zdrojové kódy třídy CFractal se nacházejí v souborech Fractal.h a Fractal.cpp. CFractal je stěžejní třída, která se stará o vše ohledně výpočtu fraktálu. Přesný výpočet má na starosti metoda Vypocti, pro zoomování a posouvání slouží metody Zoom a Posun. CFractal se také stará o vybarvování obrázku, metoda Vybarvi jako parametr dostane ukazatel na pole bajtů třídy CImage a barevnou paletu CPaleta a vybarví jednotlivé pixely obrázku. Metoda Render vypočítá obrázek přesně a v libobolném rozlišení a také obrázek vybarví, současně aplikuje na obrázek jednu ze zvolených metod antialiasingu popsaných ve 4.6. CFractal také obsahuje všechny funkce výpočtu fraktálů popsaných v kapitole 3. Další třídy definované ve Fractal.h jsou Czoom a CPosun, jedná se o struktury, které obsahují informace pro metody Zoom a Posun, jako o kolik pixelů se má posouvat nebo jaký je koeficient zoomování.

5.6 CPaleta

CPaleta je třída, která uchovává barevné palety pro vybarvování fraktálu. Palety jsou uloženy jako dvourozměrné pole bajtů velikosti 256 x 3, tedy paleta má velikost 256, a barvy jsou uloženy ve formátu RGB, každá složka je číslo velikosti 1 bajt, dohromady tedy 3 bajty. Pomocí metody GetColor vrací barvu z příslušné pozice v paletě. CPaleta se také stará o rotaci barev (viz 4.3), pomocí metody PosunBarvy. Zdrojové kódy třídy jsou v souborech Paleta.h a Paleta.cpp.

5.7 CImageFilter

Třída CImageFilter umožňuje modifikovat obrázek pomocí grafických filtrů popsaných v částech 4.4 a 4.5. Zdrojové kódy jsou v souborech ImageFilter.h a ImageFilter.cpp. V souboru ImageFilter.h je dále definována struktura ConvolveMatrix, která popisuje konvoluční filtr, jsou v ní uloženy matice 3 x 3 konvolučního filtru a další 2 koeficienty.

5.8 Dialogy

FractalDraw pracuje s množstvím dialogových oken, která zobrazují informace nebo umožňují uživateli nastavovat různé parametry. Konkrétní rozvržení oken dialogů bylo vytvořeno pomocí editoru dialogů MS Visual Studia.

CDialogFilter (DialogFilter.h a DialogFilter.cpp) umožňuje uživateli nastavit vlastní hodnoty konvolučního filtru.

CDialogIterace (DialogIterace.h a DialogIterace.cpp) nastavuje počet iterací pro výpočet.

CDialogParametry (DialogParametry.h DialogParametry.cpp) nastavuje několik parametrů ovlivňující chování aplikace, jako rychlost zoomování nebo kritická hodnota pro zvyšování iterací.

CDialogRender (DialogRender.h DialogRender.cpp) umožňuje uživateli vyrenderovat obrázek na disk. Uživatel může nastavit velikost, metodu antialiasingu, počet vzorků pro antialiasing, umístění na disku a formát obrázku.

CDialogVelikost (DialogVelikost.h DialogVelikost.cpp) slouží k nastavení rozměrů klientské části hlavního okna

CDlgAbout (DlgAbout.h a DlgAbout.cpp) zobrazí základní informace o programu.

Kapitola 6 – Závěr

6.1 Zkušenosti s aplikací

Vzhledem k výpočetní náročnosti fraktálů je pro plynulý chod programu doporučeno CPU alespoň 1 Ghz. V základním nastavení rozlišení 500 x 400, 150 iterací lze s takovým procesorem dosáhnout solidních výsledků. Zpomalení nastane teprve, když dojde k automatickému zvyšování počtu iterací (2.4). Při vyšším rozlišení nebo větším počtu iterací už program nemusí zvládat generovat dostatečný počet obrázků za sekundu aby byl zachován dojem plynulosti. Záleží samozřejmě také na tom, jaká oblast fraktálu je zobrazena. Pokud většina pixelů na obrázku potřebuje k výpočtu jen nízký počet iterací, tak ani při větších rozlišeních nedochází k trhání obrazu. Dalším faktorem, který ovlivňuje rychlost výpočtu je použití grafických filtrů. Filtry invert a grayscale pracují rychle a jejich použití není na výsledné rychlosti znát, ale konvoluční filtry už výpočet znatelně zpomalují, proto je povoleno použít nanejvýš jeden konvoluční filtr.

Pro velká rozlišení se tedy nedoporučuje používat konvoluční filtry ani velký počet iterací, i při nižších rozlišeních se doporučuje používat maximálně 500 iterací.

6.2 Srovnání s jinými programy

Jde hlavně o srovnání s programy, které nabízejí zoomování v reálném čase jako XaoS nebo FFFF.

FractalDraw nabízí dva typy fraktálů, které nejsou v XaoSu k dispozici. Jedná se o fraktály Sierpinsky a Spider. Dále umožňuje uživateli definovat si vlastní konvoluční grafický filtr. FractalDraw také umí automaticky zvyšovat počet iterací, v XaoSu musí uživatel počet iterací zvyšovat ručně. Největší výhodou je však možnost vyrenderovat obrázek na disk ve specifikovaném rozlišení a se zvoleným typem antialiasingu.

XaoS však nabízí množství jiných dalších funkcí jako různé algoritmy pro vybarvování obrázku, složitější grafické filtry a jiné. Jeho hlavní předností je především rychlost a to i při vysokých rozlišeních, používá totiž sofistikovanější algoritmy než FractalDraw včetně částí kódu psaných v assembleru. Zde se

FractalDraw XaoSu rovnat nemůže, ale i tak je schopen dosahovat solidních výsledků.

Druhý zmíněný program Fast Floating Fractal Fun (FFFF) na rozdíl od FractalDraw a XaoSu počítá hrubou silou všechny pixely každého obrázku a přesto je schopen dosáhnout velkých rychlostí díky specializovanému kódu, který využívá instrukcí SSE nebo 3DNow! apod. Nabídka jeho funkcí je však omezená, program se soustředí hlavně na rychlé výpočty, takže ho nelze s FractalDraw přímo srovnávat.

Jiné programy jako například UltraFractal nebo freewareový ChaosPro se soustředí spíše na grafickou práci s fraktály, nabízejí množství fraktálních formulí a algoritmů pro jejich vybarvování a mnoho dalších funkcí na úpravu obrázků, hlavně práci s více vrstvami. FractalDraw se nemůže jejich funkcím svými jednoduchými filtry rovnat, ani to nikdy nebylo cílem, přesto lze s pomocí FractalDraw vytvořit vizuálně působivé obrázky.

6.3 Možnosti dalšího vývoje

Naskýtá se mnoho možností a způsobů, jak FracalDraw rozšířit nebo vylepšit. Některé jednodušší návrhy:

- přidat nové typy fraktálů
- přidat další grafické filtry
- umožnit uživateli definovat vlastní barevné palety například pomocí načtení ze souboru
- jiné algoritmy pro vybarvování, než způsob popsaný ve 4.2
- algoritmy vybarvující také vnitřek fraktálu

Složitější by byly změny algoritmu zoomování, tak aby se program zrychlil a byl lépe použitelný pro vyšší rozlišení. Vhodné by také například bylo přidat vlákno, ve kterém by se prováděly výpočty fraktálů, obzvláště renderování obrázku na disk je časově náročnější funkce, která má za následek nereagující grafické rozhraní během průběhu renderování. Vzhledem k tomu, že renderování může být časově velmi náročná operace (pro velká rozlišení a velké počty iterací klidně i několik minut) bylo by vhodné, aby se uživateli zobrazoval průběh výpočtu s eventuální možností výpočet přerušit nebo úplně zastavit.

Jinou možností je vytvoření verze pro operační systém Linux. Práce na této verzi byla dokonce zahájena, ale dospěla zatím do stádia alfa verze. Linuxová verze nabízí jen základní funkce a je vytvořena pomocí trochu jiného přístupu, například má zvláštní vlákno, ve kterém probíhají veškeré výpočty fraktálů. Tato verze je zatím značně nestabilní.