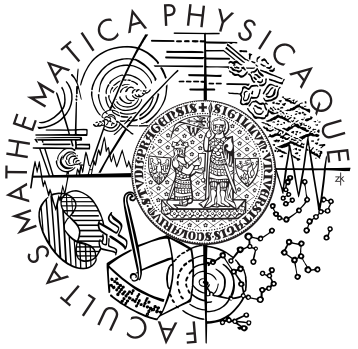
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Vladimír Čunát

Moderní metody fraktální komprese obrazu

Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Tomáš Dvořák, CSc.
Studijní program: informatika

2009

Děkuji svému vedoucímu RNDr. Tomáši Dvořákovi za jeho podporu a rady v průběhu vzniku bakalářské práce. Dále bych chtěl poděkovat RNDr. Janě Kalové za to, že mě přivedla k tomuto krásnému tématu.

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

V Praze dne 4. srpna 2009 Vladimír Čunát

Obsah

1 Úvod **6**

2 Vývoj a principy fraktální komprese obrazu **7**
2.1 IFS 7
2.2 PIFS 8
2.2.1 Konvergence PIFS 9
2.2.2 Optimální zobrazení 10
2.2.3 Rychlá konvergence 10
2.2.4 Úprava lineárních koeficientů 11
2.3 Nejčastější oblasti výzkumu 12
2.3.1 Zrychlení komprese 12
2.3.2 Zlepšení vhledu 14
2.3.3 Jiná vylepšení 19
2.3.4 Využití technik fraktální komprese pro jiné účely . . 20

3 Chyba zobrazení mezi bloky **21**
3.1 Chybové metriky 21
3.2 Klasický model 22
3.3 Chyba upravených zobrazení 23
3.4 Penalizace vysokého lineárního koeficientu 25

4 Implementace **27**
4.1 Modularizace 28
4.2 Kořenový modul 29
4.3 Konverze kvality 30
4.4 Zpracování barev 31
4.5 Počítání s maticemi pixelů 31
4.6 Změny tvarů 33
4.7 Dělení na cílové bloky 34
4.8 Výběr množiny zdrojových bloků 36
4.9 Hledání optimálních zobrazení 38

4.10 Predikce vhodných zdrojových bloků 40
4.11 Kódování celočíselných řad 45

5 Testování a vyhodnocení **46**
5.1 Účinnost penalizace 49
5.2 Účinnost predikce 49
5.3 Účinnost diferenčního kódování 52
5.4 Celková účinnost 53

6 Závěr **58**

Název práce: Moderní metody fraktální komprese obrazu
Autor: Vladimír Čunát
Katedra (ústav): Kabinet software a výuky informatiky
Vedoucí bakalářské práce: RNDr. Tomáš Dvořák, CSc.
E-mail vedoucího: Tomas.Dvorak@mff.cuni.cz

Abstrakt: V předložené práci studujeme metody fraktální komprese obrazu. Jsou zde rozebrány základní používané techniky, publikovaná rozšíření a jsou navržena a implementována drobná vylepšení stávajících metod. Dále je představen modulární systém umožňující vyměňování různých variant jednotlivých částí kompresního procesu a zjednodušující tak porovnání účinnosti různých algoritmů.

Nově navržená vylepšení jsou vyhodnocena na testovacích obrázcích. Představená penalizační metoda dosahuje vyšších kvalit dekomprimovaných obrázků. Diferenční kódování v kombinaci s přeuspořádáním cílových bloků snižuje nároky na prostor okolo 5%. Predikce implementovaná pomocí KD-stromů se naopak ukázala jako nepříliš účinná v porovnání s článkem používajícím jinou datovou strukturu.

Klíčová slova: fraktál, komprese obrazu, systém iterovaných funkcí

Title: Contemporary Methods in Fractal Image Compression
Author: Vladimír Čunát
Department: Department of Software and Computer Science Education
Supervisor: RNDr. Tomáš Dvořák, CSc.
Supervisor's e-mail address: Tomas.Dvorak@mff.cuni.cz

Abstract: In the present work we study fractal image compression. We discuss basic techniques, published improvements, and a few proposed enhancements of the current methods including their implementation. A framework for fractal compression is introduced. It is designed in order to be able to replace individual parts of the encoding process by different algorithms, simplifying comparison of their combinations.

Presented enhancements are evaluated on a set of testing images. With proposed penalisation method decompressed images reach higher qualities. The differential coding combined with reordering of range-blocks decreases the amount of needed space approximately by 5%. On the other hand the prediction implemented by KD-trees isn't as accurate in comparison to a paper using another data structure.

Keywords: fractal, image compression, iterated function system

Kapitola 1

Úvod

Množství a rozměry zpracovávaných digitálních obrázků se neustále zvětšují, a tak možnost jejich efektivní reprezentace narůstá na důležitosti. Kapacity pamětí a rychlosti síťových přenosů sice také rostou, ale na druhou stranu se rozšiřují mnohem menší, úspornější a méně výkonná zařízení pracující s obrázky.

Fraktální komprese obrazu je relativně nová a velmi netradiční ztrátová metoda, kde je obrázek charakterizován pomocí vzájemných vztahů mezi jeho částmi. Vykazuje mnoho neobvyklých vlastností, které lze využít i pro jiné účely než pro kompresi. Díky nim získala mnoho popularity, ale také se stala terčem mýtů.

Samotná komprese nemá daný pevný postup. Naopak, každou její část lze provádět mnoha různými způsoby. V této práci bylo cílem vytvořit modulární systém pro fraktální kompresi, kde by bylo možné vyměňovat jednotlivé části komprese, a umožnit tak snadné porovnávání jednotlivých algoritmů a vyhodnocení jejich interakcí.

V následující kapitole jsou popsány základní principy fraktální komprese a různé zkoumané způsoby jejího vylepšení. Kapitola 3 pak podrobněji rozebírá zobrazení mezi zdrojovými a cílovými bloky, která jsou základem komprese. V kapitole 4 jsou popsány věci související s implementací — použité vývojové nástroje, způsob modularizace a implementace jednotlivých technik. Kapitola 5 vyhodnocuje vliv použití různých technik ve fraktální kompresi na testovacích obrázcích a také porovnává účinnost komprese se standardními ztrátovými formáty JPEG a JPEG 2000.

Kapitola 2

Vývoj a principy fraktální komprese obrazu

V této kapitole bude používáno následující značení:

potenční množina $2^X = \{A : A \subseteq X\}$

iterace zobrazení mějme zobrazení $f : X \to X$, pak $f^k : X \to X$ je definováno tak, že f^0 je identita a

$$\forall k \geq 1 \quad \forall x \in X \quad f^k(x) = f\left(f^{k-1}(x)\right)$$

kontraktivita zobrazení zobrazení $f : X \to X$ na metrickém prostoru X s metrikou d je kontraktivní, pokud

$$\exists s < 1 \quad \forall x, y \in X \quad d(x, y) \leq s\, d(f(x), f(y))$$

takové s budeme nazývat faktorem kontraktivity f

2.1 IFS

Základy fraktální komprese byly položeny v druhé polovině 80. let, kdy Barnsley studoval systémy iterovaných funkcí (IFS). IFS je množina bodů A v úplném metrickém prostoru X definována pomocí souboru kontraktivních zobrazení X do sebe. Mějme soubor zobrazení $f_i : X \to X$ pro $i \in J$ a definujme jejich sjednocení $f : 2^X \to 2^X$ tak, že pro $B \subseteq X$ $f(B) = \{f_i(b) \mid i \in J,\ b \in B\}$. Na 2^X lze definovat metriku tak, abychom dostali zase úplný metrický prostor a zobrazení f bylo kontraktivní, jak je ukázáno například v [3] pomocí Hausdorffovy metriky. To podle Banachovy věty o pevném

bodě dává, že takto vzniklé zobrazení f má právě jeden pevný bod, kterým je právě hledaná množina A, často nazývaná atraktor nebo fraktál[1].

IFS mají několik zajímavých vlastností. Pro získání atraktoru stačí znát zobrazení f a iterovat ho na libovolné neprázdné kompaktní podmnožině prostoru X. Navíc vzniklé fraktály mohou mít detailní kresbu při libovolném přiblížení, přestože jejich matematický popis je velmi malý, takže pro tyto speciální obrázky lze dosáhnout libovolně velkého kompresního poměru. Mezi nejznámější IFS patří Barnsleyho kapradina, generovaná v $\mathbb{R}^2$ pomocí čtyř afinních zobrazení. Barnsley zkoumal, zda by nebylo možné proces obrátit — k danému obrázku najít soubor zobrazení, jehož pevný bod by obrázku byl velmi blízký. Pak by místo souboru pixelů stačil pro popis obrázku jen popis těchto zobrazení.

Zde se používá takzvaná kolážová věta, uvedená například v [5], která dává horní odhad na odchylku atraktoru od požadovaného obrázku:

Věta 2.1. *Nechť f je zobrazení úplného metrického prostoru do sebe, které je kontraktivní s faktorem $s < 1$ v metrice d, pak*

$$d\left(A, \lim_{n\to\infty} f^n(A)\right) \leq \frac{1}{1-s}\, d(A, f(A)).$$

Díky tomu stačí hledat zobrazení, která obrázek transformují a přitom ho změní co nejméně. Pro jistotu konvergence dokonce stačí, že f^k je kontraktivní s faktorem $s < 1$ pro nějaké k. Ukázalo se, že největší problém je v tom, že na rozdíl od klasických IFS málokterý obrázek lze charakterizovat jako sjednocení několika transformovaných zmenšenin celého obrázku. Bylo sice možné obrázek rozdělit na menší kusy charakterizované IFS, ale tuto metodu se nepodařilo zautomatizovat.

2.2 PIFS

Pro kompresi obrázků také bylo nutné najít lepší model, než množinu bodů v $\mathbb{R}^2$. Jedna z možností reprezentace obrázků ve stupních šedi je funkce tvaru $g : I^2 \to I$, kde I je značení pro interval $[0, 1] \subset \mathbb{R}$. Zde budeme analogicky hledat soubor funkcí na těchto obrázkových funkcích (budeme je nazývat operátory) takový, aby jejich sjednocením vznikl operátor F s pevným bodem co nejblíže danému obrázku (tedy jeho funkci). Aby bylo zaručeno, že lze sjednocení provést, v PIFS (partitioned IFS) je prostor obrázku rozdělen na disjunktní části

$$\forall i \in J \quad R_i \subset I^2, \qquad \bigcup_{i\in J} R_i = I^2, \qquad \forall i, j \in J \quad i \neq j \;\rightarrow\; R_i \cap R_j = \emptyset$$

[1] fraktály lze konstruovat i jinými způsoby než pomocí IFS

a operátory jsou tvaru $F_i : (I^2 \to I) \to (R_i \to I)$.

PIFS, které zavedl Jacquin, dnes tvoří základ naprosté většiny technik pro fraktální kompresi obrazu. Pro zjednodušení se zde uvažují pouze afinní operátory, kde se navíc vzájemně neovlivňuje transformace polohy s transformací barvy. Formální zápis aplikace operátoru na obrázek $g : I^2 \to I$

$$F_i g = h_i, \qquad h_i : R_i \to I, \qquad h_i\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = (c_i \circ g \circ w_i)\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)$$

$$\text{pro nějaká afinní zobrazení} \quad w_i : R_i \to I^2 \quad \text{a} \quad c_i : I \to \mathbb{R}$$

Za R_i se volí souvislé části obrazu, typicky čtverce o straně délky 2^k, a nazývají se cílové bloky. Díky spojitosti w_i jsou $D_i = w_i(R_i)$ také souvislé části obrazu a nazývají se zdrojové bloky.[2] Transformace w_i určuje, která část obrazu bude zobrazena do cílového bloku a také jak bude otočena a případně symetricky převrácena. Za w_i se volí kontraktivní zobrazení, přestože to pro níže uvedené tvrzení o konvergenci není potřeba. Tato kontraktivita mimo jiné způsobuje vytváření nových detailů při přiblížení.

2.2.1 Konvergence PIFS

Pro konvergenci a horní odhad chyby PIFS modelu lze zase použít kolážovou větu. Následující důkaz je modifikací důkazu uvedeného v [5]. Zde se hodí metrika na obrázcích (funkcích $g_i : I^2 \to I$)

$$d_{\sup}(g_1, g_2) = \sup_{\vec{z} \in I^2} \{|g_1(\vec{z}) - g_2(\vec{z})|\}$$

Postačí, aby existovalo $s < 1$ takové, že každé ze zobrazení c_i má lineární člen a_i : $|a_i| \leq s$. Pak pro libovolné obrázky $g_1, g_2 : I^2 \to I$ a každý bod $\vec{z} \in I^2$ existuje právě jedno R_i takové, že $\vec{z} \in R_i$ a platí

$$|(F_i g_1)(\vec{z}) - (F_i g_2)(\vec{z})| \;\leq\; s\,|(g_1 \circ w_i)(\vec{z}) - (g_2 \circ w_i)(\vec{z})| \;\leq\; s\, d_{\sup}(g_1, g_2)$$

Z toho už plyne kontraktivita celého operátoru F

$$d_{\sup}(F g_1, F g_2) \;\leq\; s\, d_{\sup}(g_1, g_2)$$

a díky tomu lze použít kolážovou větu.

Ukazuje se ale, že pro použití v implementaci je tento odhad příliš volný, stejně jako použitá metrika nedává dobré vizuální výsledky.

[2] značení D_i a R_i pochází z anglických termínů *domain block* a *range block*

2.2.2 Optimální zobrazení

Pro daný cílový blok R_i se volí afinní zobrazení w_i tak, aby mapovalo blok na jeden ze souboru zdrojových bloků. Tento soubor bývá pevně daný kvůli zjednodušení ukládání a vyhledávání. Volba zdrojového bloku je hlavním zdrojem výpočetní složitosti celé komprese a jejímu urychlení byla věnována značná část výzkumu v oblasti. Obecně je snaha zvolit takový blok D_i, že spolu s optimálním barevným zobrazením c_i bude pro složený operátor $F_i g = (c_i \circ g \circ w_i)$ vzdálenost $d(\ c_i \circ g \circ w_i,\ g\ )$ měřená na R_i co nejmenší.[3]

Volba optimálního zobrazení c_i a tedy i následně volba w_i je silně závislá na použité metrice. Ve fraktální kompresi se téměř výhradně používá RMSE metrika. Její hlavní výhody i nevýhody jsou dány tím, že se vždy berou v úvahu pouze sobě odpovídající body. To umožnuje velmi jednoduchou a rychlou práci s metrikou a také optimalizaci v ní, na druhou stranu díky tomu pro větší bloky nedává metrika výsledky odpovídající vizuálnímu rozdílu. Možnosti použití alternativních metrik jsou zmíněny v 2.3.2 na straně 18.

Jak je ukázáno v kapitole 3 pro diskrétní případ, při použití RMSE existuje právě jedno optimální c_i a lze ho (pro dané g a w_i) snadno spočítat analyticky. To platí za předpokladu, že $g \circ w_i$ není konstantní na R_i (tj. g není konstantní na D_i) — jinak by se jednalo o aproximaci konstantním blokem, volba lineárního koeficientu by byla libovolná a výsledek vždy horší, než při výběru jakéhokoliv nekonstantního zdrojového bloku.

2.2.3 Rychlá konvergence

Fraktální komprese v této podobě stále trpěla jedním skrytým problémem. Při dodržení kontraktivity všech zobrazení c_i kolážová věta sice zaručovala, že obrázek zkonverguje (v prostoru všech obrázků s libovolnou metrikou) a nebude se příliš lišit, ale nijak neomezovala počet nutných iteračních kroků. Ukázalo se, že tento problém není jen teoretický — i při přísnějších omezeních na kontraktivitu bylo pro některé obrázky nutné provádět mnoho desítek iterací než změny přestaly být okem viditelné.

Řešení tohoto problému bylo publikováno v [9] a podrobněji rozebráno v 6. kapitole knihy [5]. Jedná se jen o drobnou modifikaci barevného zobrazení. Pokud bylo původní optimální barevné zobrazení ve tvaru $c_i(z) = p_i z + q_i$, pak místo ukládání p_i i q_i je uloženo p_i a průměrná barva bloku R_i. Nové zobrazení $\tilde{c_i}$ pak znormalizuje barvu zdrojového bloku D_i odečtením jeho aktuálního průměru, výsledek vynásobí lineárním koeficienem p_i a přičte předpočítanou průměrnou barvu bloku R_i ve vstupním obrázku.

[3] funkce jejichž vzdálenost měříme jsou definovány pouze na R_i místo původního I^2

Přestože přičítaná průměrná barva R_i je uložena v charakteristice zobrazení a je stále stejná, skutečný konstantní koeficient (výše značený q_i) se může měnit s každou iterací, takže z matematického hlediska se $\tilde{c}_i$ chová pokaždé jako jiné zobrazení.

V rámci jedné iterace je ale $\tilde{c}_i$ stále afinní zobrazení. Navíc lze snadno ukázat, že při použití RMSE metriky optimální afinní zobrazení c_i také vždy zobrazuje průměnou barvu D_i na průměrnou barvu R_i, takže když stav obrázku při iterování konverguje k g (zdrojovému obrázku), pak i $\tilde{c}_i$ konverguje k c_i.

V tomto přístupu je tedy iterační operátor složitější na vyhodnocení, ale má lepší vlastnosti, například efektivnější ukládání koeficientů zobrazení díky jejich menší korelaci. Koeficienty q_i zobrazení c_i byly většinou v absolutní hodnotě malá čísla, ale mohly se pohybovat ve velkém rozsahu. Průměrná barva bloku se může pohybovat jen v rozmezí $[0, 1]$ a má jasnější význam — to pomáhá při volbě způsobu kvantizace při uložení do souboru a umožnuje například využít toho, že sousední bloky budou mít pravděpodobně blízkou barvu.

Další výhodou je mnohem rychlejší a jistější konvergence. Je vidět, že po každé iteraci mají všechny bloky R_i správnou průměrnou barvu. Tím se vzhled zdrojových bloků D_i už po první iteraci dostane blízko vzorovému obrázku, což se při další iteraci promítne do cílových bloků R_i.

Ve výše odkazovaných publikacích je ukázáno, že po takovéto úpravě barevných zobrazení iterační proces konverguje ke stejnému výsledku a také, že za trochu silnějších předpokladů než zde uvedené bude potřebný počet iterací malý a bude záviset pouze na velikostech zdrojových a cílových bloků. Používá se zde diskrétní model obrázku — posloupnost reálných čísel, kde cílové bloky jsou tvořeny 2^{b_i} po sobě jdoucími hodnotami a zdrojové bloky o velikosti 2^{d_i} jsou tvořeny několika po sobě jdoucími cílovými bloky. Zmenšování bloků vždy probíhá zprůměrováním $2^{d_i - b_i}$ po sobě jdoucích hodnot.

Na rozdíl od konvergence založené na kolážové větě je důkaz proveden bez jakéhokoliv omezení na kontraktivitu barevných zobrazení $\bar{c}_i$, což umožnuje mnohem přesnější aproximaci bloků s ostrými přechody bez ztráty jistoty konvergence. Zde se naopak využívá kontraktivity transformací w_i.

2.2.4 Úprava lineárních koeficientů

Podobným způsobem, jako byly konstantní koeficienty barevných zobrazení nahrazeny průměry, lze také lineární koeficienty nahradit směrodatnými odchylkami — barvy zdrojových bloků jsou vyděleny svou směrodatnou odchylkou a vynásobeny cílovou směrodatnou odchylkou. Jak je rozebráno v části 3.3, mírně se tím zvýší chyba optimálního zobrazení.

Výhody a nevýhody tohoto přístupu jsou podobné jako při nahrazování konstantních koeficientů zobrazení — především jednodušší reprezentace zobrazení a pravděpodobně také rychlejší konvergence.

Například v článku [8] a také v této práci je používán právě tento model barevných zobrazení, ale nikde se mi nepodařilo nalézt nějaké zdůvodnění nebo odkaz na text zabývající se jeho vlastnostmi. Nejsou mi tedy známy žádné výsledky týkající se konvergence ani pro případné omezené varianty.

Lineární koeficienty barevných zobrazení nejsou stejně jako v předchozím modelu nijak omezeny, ale experimenty prováděné v rámci této práce ukazují, že může být výhodné vyhýbat se zobrazením, kde je absolutní hodnota lineárního koeficientu vysoká. Tato úprava je diskutována v 3.4, včetně teoretického zdůvodnění.

2.3 Nejčastější oblasti výzkumu

Především v první polovině 90. let bylo věnováno fraktální kompresi velké úsilí. Bylo zkoumáno (většinou odděleně) mnoho odlišných způsobů vylepšení základních technik popsaných výše. V této části jsou shrnuty nejvýznamnější výsledky a zajímavé návrhy pro další výzkum. Pro přehlednost jsou rozčleněny podle toho, na které aspekty komprese se zaměřují.

2.3.1 Zrychlení komprese

Za největší problém byla považována obrovská výpočetní náročnost hledání vhodných zdrojových bloků. Od dob nejintenzivnějšího výzkumu v oblasti se sice výkon běžných počítačů mnohonásobně zvýšil, ale stále je nutné se tímto problémem zabývat. Jedním z důvodů je, že dnes je potřeba zpracovávat mnohem větší množství dat. Velikosti a množství fotografií roste a také je zkoumána možnost využití pro kompresi videozáznamů.

Klasifikační metody

Nejčastější technikou pro urychlení výběru vhodného zdrojového bloku byly klasifikační metody. Všechny cílové a potencionální zdrojové bloky jsou nějakým způsobem ohodnoceny a poté je vzájemná zobrazitelnost přesně vyhodnocována pouze pro „kompatibilní“ dvojice. To typicky vede k vynechání některých optimálních zobrazení, ale pokles kvality bývá velmi malý a urychlení mnohonásobné (zkomprimovaná velikost se nezmění).

Většinou jde o jednoduché rozdělení bloků do několika tříd a uvažování pouze zobrazení v rámci jedné třídy, například klasifikace navržené Jacqui-

nem nebo Fisherem [5]. Zde je nevýhodné, že pro bloky „na okrajích“ tříd často mohou mít podobné bloky v jiných „sousedních“ třídách. To vede ke složitějším metodám, které hledají i v příbuzných třídách, nebo používají hierarchické způsoby klasifikace, například [6].

Převod na hledání nejbližšího bodu

Problémem klasifikačních metod je, že přes výrazné zrychlení nejsou schopny zlepšit asymptotickou složitost výpočtu. Zde se nabízí zobecnění klasifikačních metod, kde by pro blok byl vygenerován vektor z nějakého prostoru uspořádaného tak, že vzdálenost v něm odpovídá míře vzájemné zobrazitelnosti. Pak by pro každý bod prostoru odpovídající nějakému cílovému bloku stačilo hledat nejbližší z množiny bodů prostoru odpovídající zdrojovým blokům. Hledání nejbližších sousedů je známá úloha, kterou už lze řešit výrazně rychleji než úměrně se součinem velikostí obou množin.

Tato možnost byla publikována mezi projekty v knize [5] a podrobněji rozebrána v samostatném článku [11]. Je zde především dokázáno, že při uvažování klasických afinních zobrazení pro transformaci barvy mezi bloky je pro pevný cílový blok optimální čtvercová chyba (SE) rovna určité rostoucí transformaci eukleidovské vzdálenosti normalizovaných vektorů obou bloků (cílového a zdrojového). Tedy pro nalezení zdrojového bloku „nejpodobnějšího“ k cílovému stačí jen najít nejbližší bod v eukleidovském prostoru.

Jsou zde různé komplikace — kvůli kvantizaci koeficientů nemusí nejbližší bod být ten nejlepší, při uvažování i zobrazení se zápornými lineárními koeficienty je nutné prohledávát strukturu dvakrát (nebo vložit každý bod dvakrát). Je zde také otázka spotřeby paměti, která sice není tak velkým problémem, jako v době publikace metody, ale stále potřeba se jí věnovat. Dále se problematikou zabýváme v části 4.10.

Lokální prohledávání

Za další možnost urychlení se považoval jiný způsob redukce nutných porovnání — přeskočení zdrojových bloků, které jsou v obrázku příliš vzdálené od cílového bloku (pro který hledáme zobrazení). Na rozdíl od klasifikačních metod je množina uvažovaných zdrojových bloků zřejmá už z polohy cílového bloku, takže lze ušetřit nějaký prostor nutný pro uložení jejich identifikačních čísel.

Základem této techniky je domněnka, že ideální zdrojový blok má větší pravděpodobnost výskytu blíže k danému cílovému bloku. V kapitole 3 knihy [5] je ukázáno, že posun rozdělení vzdáleností optimálních dvojic oproti vzdálenostem náhodně vybraných dvojic je zanedbatelný. Zrádné je zde především

to, že už jen rozdělení vzdálenosti dvou náhodných bodů je velmi nerovnoměrné.

Jsou zde i další nevýhody spojené s tím, že je množina uvažovaných zdrojových bloků pro každý cílový blok jiná, například obtížná kombinace s jinými metodami urychlení komprese.

Na druhou stranu například v nedávném článku [8] je zkoumána závislost podobnosti dvou cílových bloků na jejich vzdálenosti s opačnými výsledky na stejném obrázku (Lenna). To je pravděpodobně způsobeno tím, že při hledání podobnosti mezi zdrojovým a cílovým blokem vždy dochází ke zmenšování, zatímco blízké cílové bloky často mívají stejnou velikost.

Této podobnosti je zde využito nejen pro urychlení hledání zdrojového bloku tím, že jsou nejprve vyzkoušeny zdrojové bloky blízkých cílových bloků, ale hlavně ke zvýšení kompresního poměru. Místo ukládání identifikačního čísla zdrojového bloku stačí uložit relativní pozici nejbližšího cílového bloku se stejným zdrojovým blokem. Vzhledem k tomu, že cílových bloků bývá mnohem méně než zdrojových a podobné cílové bloky se často vyskytují blízko, lze takto ušetřit mnoho bitů ve výstupu za cenu minimálního poklesu kvality.

Přerušení prohledávání

Další metoda snížení počtu nutných porovnání je nehledat nejlepší zdrojový blok a spokojit se s dostatečně dobrým už nalezeným blokem. Dosažené zrychlení ale nejspíš nebude nijak veliké. Nejsou mi známy žádné práce, které by se podrobněji zabývaly touto technikou.

2.3.2 Zlepšení vhledu

Vetšinou nezávisle na rychlosti komprese byly zkoumány také metody zlepšující kvalitu dekomprimovaných obrázků. Zde jsou zařazeny i techniky zvyšující kompresní poměr, protože jde o stejný cíl — typicky lze pouhým nastavováním parametrů komprese zlepšovat jeden z těchto ukazatelů na úkor druhého.

Dělení na cílové bloky

Jedním ze základních faktorů ovlivňujících vzhled je způsob dělení obrázku na cílové bloky. Nejjednodušší je určit dělení napevno. To je ale v praxi nepoužitelné kvůli obtížné regulaci kvality a kompresního poměru. Navíc nalézt dobré zobrazení pro skoro konstatní bloky je velmi jednoduché, zatímco pro

bloky obsahující hodně detailů nejspíš žádný vhodný zdrojový blok k dispozici nebude. Proto bylo nutné hledat adaptivní metody, které by v „plochých“ místech nechávaly velké bloky a v místech se složitější kresbou mohly dělit obrázek na menší a zvyšovat u nich tak pravděpodobnost nalezení dostatečně dobrého zobrazení.

Jednoznačně nejpoužívanější metodou je dělení pomocí čtyřstromu (Quadtree), popsaného například v [5], kapitola 3. Pro zjednodušení se uvažují jako velikosti obrázku a bloků jen čtverce o hranách délky mocniny 2. Začne se s jedním velkým cílovým blokem pokrývajícím celý obrázek a postupuje se tak, že se vždy vezme nějaký cílový blok a najde se pro něj nejlepší zobrazení. Pokud je chyba dostatečně malá, je zobrazení použito, jinak je blok rozdělen na čtyři o poloviční velikosti, pro které se použije stejný postup.

Tento způsob dělení je celkem efektivní a také jednoduchý na implementaci, včetně velmi úsporného uložení tvaru čtyřstromu. Pro použití v praxi je potřeba čtyřstrom zobecnit i pro jiné velikosti obrázku a jsou vhodná i další vylepšení. Podrobněji je dělení čtyřstromem rozebráno v části 4.7.

Jednou z nevýhod čtyřstromu je, že skoky mezi jednotlivými velikostmi bloků jsou relativně velké, takže v některých případech dochází k dělení na zbytečně malé bloky. Tento problém je řešen v [10] pomocí přidání jednoho mezikroku, což sice vede k delšímu výpočtu, ale také mírně zlepšuje kvalitu komprimovaných obrázků.

Zobecněním dělení čtyřstromem je HV (horizontálně-vertikální) dělení, popsané v 6. kapitole knihy [5]. Základní myšlenka je stejná jako u čtyřstromů, akorát umožňuje dělit obecné obdelníkové bloky na dva menší podle libovolné svislé nebo vodorovné přímky.

To také vede k menším skokům mezi velikostmi bloků, ale hlavní výhoda je ve vyšší adaptivitě. V [5] je navržena heuristická metoda, která vybírá místo dělení nejčastěji těsně u silných barevných zlomů a zároveň se vyhýbá blokům s příliš velkým poměrem délek stran. To zvyšuje pravděpodobnost nalezení vhodného zdrojového bloku a vyvažuje tak větší složitost uložení struktury rozdělení obrázku.

Podle autorů dává metoda o něco lepší výsledky než klasický čtyřstrom, ale komplikuje ostatní fáze komprese. Protože cílové bloky mohou mít libovolné obdelníkové tvary, je mnohem těžší použití klasifikačních a jiných urychlujících metod.

Další typy dělení zmiňované v [5], například trojúhelníkové, polygonální, nebo šestiúhelníkové[4] jsou velmi komplikované na implementaci a pravděpo-

[4]toto dělení na rozdíl od šestiúhelníkové architektury zmíněné níže pracuje s klasickými čtvercovými pixely, jen zdrojové bloky mají přibližně tvar šestiúhelníků

dobně by nepřinesly významné zlepšení kvality. Mezi jejich předpokládané výhody ale patří méně „blokovitý“ vhled způsobený dělením podle šikmých nebo různě orientovaných hran, případně mírným překryvem pixelů na krajích bloků.

Dělení na nezávisle zobrazované bloky vytváří rušivou strukturu, jejíž redukcí je možné výrazně zlepšit subjektivní vzhled a někdy i PSNR komprimovaného obrázku. Problémem je vysoká citlivost lidského na hrany, obzvláště horizontálně a vertikálně orientované. Pro potlačení této struktury lze dělat zásahy na dvou různých místech celého kompresního procesu.

Nejjednodušší možnost je věnovat se tomuto problému až při dekódování. Hranice bloků jsou vždy známé, takže je možné vyhlazovat výsledný obrázek jen na jejich okrajích. Vizuálního zlepšení lze dosáhnout už použitím jednoduchých nebo vážených průměrů [2, 5]. U obrázků s vyšším kompresním poměrem a také u jednotlivých větších cílových bloků je větší potřeba vyhlazování a obvykle dojde i ke zvýšení PSNR, ale naopak při nastavení vysoké kvality po vyhlazování PSNR klesne (přestože obrázek často „vypadá lépe“).

Další možností je snažit se redukovat tyto přechody už při kompresi. Zde mi nejsou známy žádné publikované výsledky kromě návrhů způsobů řešení. Lze vážit chybu na okrajích cílových bloků více než chybu uvnitř, případně použít dělení na překrývající se bloky a při dekódování hodnoty na překryvu kombinovat.

Zdrojové bloky

Dalším faktorem ovlivňujícím vzhled je volba souboru zdrojových bloků. Důležitý je především počet bloků v souboru. Zvýšení počtu sice vede k potřebě většího prostoru na uložení identifikačních čísel zdrojových bloků, ale také zvyšuje pravděpodobnost nalezení vhodného bloku, takže lze při stejné kvalitě použít rozdělení obrázku na menší počet větších cílových bloků.

Výsledky testování (například v [11, 5]) ukazují, že zvyšování množství zdrojových bloků typicky vede ke zlepšení kvality. Nevýhodou je vyšší výpočetní náročnost. Zde se ukazuje spojitost se zrychlením komprese — při vyšší rychlosti by bylo únosné prohledat větší soubor bloků a zlepšit tak kvalitu (případně kompresní poměr).

Kromě množství zdrojových bloků je potřeba rozhodnout, jak zdrojové bloky vybírat. Základní podmínkou je, že musí být větší než příslušné cílové bloky, aby byla prostorová část zobrazení kontraktivní. Ekvivalentně lze uvažovat výběr zdrojových bloků stejně velkých jako příslušné cílové z nějaké zmenšeniny původního obrázku. To je díky své obecnosti někdy výhodné i

pro implementaci (využito také v této práci).

Samotné zmenšování lze provádět mnoha různými způsoby. Kvůli jednoduché implementaci se nejčastěji používá především zmenšování v obou rozměrech na polovinu (průměrováním hodnot po čtveřicích) a navíc se uvažuje všech osm transformací čtverce složených ze symetrií a otáčení o násobky 90°. V [2] je navíc zkoumáno použití zmenšování s kontraktivitou rozdílnou pro obě osy nebo s přidáním otočení o 45° s nejasnými výsledky. Tyto myšlenky byly také využity při implementaci (část 4.8).

Podle [12] přidávání osmi transformací čtverce nijak nezlepšuje kvalitu, ale může sloužit spíše jako způsob zvětšování počtu zdrojových bloků. V takto zvětšeném souboru bloků je často pak možné dosáhnout úspory paměti a zrychlení oproti jiným způsobům zvětšování souboru.

Šestiúhelníková architektura

V poslední době je zkoumáno také pojetí obrázku jako bodů uspořádaných do šestiúhelníkové mříže místo čtvercové. Tato architektura mnohem lépe odpovídá rozložení tyčinek a čípků na sítnici oka a také vede k přirozené redukci horizontálních a vertikálních artefaktů. Další výhodou je větší množství přirozených rotací na mříži.

Protože běžné formáty obrázků podporují pouze obdélníkový tvar se čtvercovými body, je potřeba nejprve stanovit způsob převodu mezi mřížemi. Objevují se dva různé přístupy. První možnost je představit si čtvercovou mříž o 60° zkosenou, tím se středy pixelů dostanou právě do pravidelné šestiúhelníkové mříže a je možné je převádět 1 : 1 (například v [14]). Druhou možností je body přepočítat pomocí váženého průměrování, například metodou ukázanou v [7].

Výhoda prvního přístupu oproti druhému je, že zobrazuje právě jeden čtvercový pixel na jeden šestiúhelníkový, takže nedochází k žádnému rozmazávání. Na druhou stranu se v podstatě pracuje se zkoseným obrázkem, takže interpretace vzdáleností bude v různých směrech různě změněna, což může vést k nerovnoměrným projevům zkreslení způsobeného kompresí.

Změna architektury také přináší mnoho otázek a implementačních problémů. Po libovolném převodu obdélníkového obrázku budou v šestiúhelníkové mříži „díry“ neobvyklých tvarů nebo nepravidelné okraje a už samotný způsob reprezentace mříže v paměti je netriviální. Ve zmíněných článcích jsou tyto potíže řešeny kompresí pouze části obrázku o vhodném tvaru, ale ani tak zde dosažené výsledky nejsou přesvědčivé.

Zpracování barev

Ve většině případů je fraktální komprese zkoumána pouze na obrázcích ve stupních šedi s argumentem, že barevné obrázky lze snadno komprimovat jako tři nezávislé jednobarevné. Je ale zřejmé, že RGB není vhodný model pro kompresi obrazu, protože jsou jeho složky vzájemně silně závislé.

Nejjednodušší je použít některý z barevných prostorů navržených pro tyto účely, například YCbCr nebo L*a*b* [21]. Toto je standardní řešení v mnoha jiných metodách komprese obrazu. V závislosti na použitém modelu může být vhodné navíc upravit nastavení kvality komprese jednotlivých složek, protože citlivost oka na různé barvy se liší. Ve formátech založených na standardu JPEG se dokonce u barevných složek snižuje rozlišení, ale ve fraktální kompresi je mnohem výhodnější rozlišení ponechat a snížit kvalitu, protože zde výhody oproti JPEGu rostou s kompresním poměrem (experimenty, například v příloze D knihy [5], jasně ukazují, že fraktální komprese nejsnáze překonává JPEG při extrémních kompresních poměrech).

Alternativně lze spočítat vlastní barevný prostor pro konkrétní obrázek. Budeme pro jednoduchost uvažovat jen změnu báze vektorového prostoru, charakterizovanou maticí velikosti 3×3 (barva pixelu v RGB prostoru je prvkem $\mathbb{R}^3$). Zde lze použít metodu zvanou PCA (principal component analysis), známou také jako KLT (Karhunen–Loève transform). Metoda nalezne nejlepší ortonormální bázi z hlediska soustředění maxima informace (rozptylu) do minima souřadnic. Transformační matice je dána vlastními vektory kovariační matice a navíc příslušná vlastní čísla charakterizují důležitost jednotlivých souřadnic.[5]

Je možné také volit jiné přístupy ke zpracování barev. V [13] je předvedena metoda, která pomocí čtyřstromu dělí barevný obrázek na bloky (podobně jako při dělení cílových bloků) dokud nejsou barvy v blocích dostatečně korelované. Parametry korelace barev pro každý z bloků jsou uloženy a do dalšího kódování postupuje již jen jeden obrázek ve stupních šedi. Zde je výhoda v adaptivní kompresi informací o barvě a také v urychlení díky tomu, že se složité metody fraktální komprese použijí jen na jeden jednobarevný obrázek, přestože většina vnímané informace bude obsažena právě v něm.

Další metody

Zde ještě ještě stručně zmíněno několik dalších metod používaných pro zlepšení vzhledu komprimovaných obrázků.

[5] PCA samotnou lze použít ke kompresi obázků tak, že se obrázek rozdělí na bloky a s každým z nich se pracuje jako s vektorem. Po náročném výpočtu pak vznikne transformace, v praxi často velmi podobná diskrétní kosinové transformaci.

Obzvláště při vysokých kompresních poměrech je výhodné počítat chybu zobrazení ze zdrojových bloků do cílových s použitím kvantizovaných parametrů zobrazení místo teoretických optim (tedy z hodnot dostupných při dekompresi místo hodnot dostupných při kompresi). Aby šlo metodu použít, je potřeba se omezit jen na některé způsoby kvantizace a také výpočet této chyby bývá trochu složitější.

Jak již bylo zmíněno, RMSE metrika pro větší bloky nepopisuje dobře vnímaný rozdíl. Použití přesnějších metrik bylo zatím věnováno jen velmi málo pozornosti, především díky jednoduchosti práce s RMSE umožnující relativně rychlou implementaci jejího vyhodnocení. Vhodností jiných metrik pro fraktální kompresi se zabývá například [1].

PIFS uvažuje afinní zobrazení z jednoho zdrojového bloku do jednoho cílového bloku. To lze zobecnit na hledání zobrazení z dvou nebo více zdrojových bloků. Víceprůchodové komprese, kde se jednoduše kóduje chyba předchozího kroku, nedávají lepší výsledky. Přímé hledání zobrazení z více zdrojových bloků ke zlepšení kvality výsledku vede, ale množství kombinací zde roste ještě mnohem rychleji než v klasické PIFS kompresi. Například v [4] je navrženo snižovat složitost pomocí redukce podobných zdrojových bloků a použití hladového přístupu.

2.3.3 Jiná vylepšení

Rychlost dekomprese

Fraktální dekomprese bývá považována za velmi rychlou, což je způsobeno srovnáváním s kompresní fází. Ve skutečnosti se stále vyplatí pracovat na jejím zrychlení. Při prvních iteracích zobrazení je obrázek složen z jednolitých bloků a je tedy zbytečné pracovat v plném rozlišení.

Jak je uvedeno v [5], stačí začít dekompresi na zmenšeném obrázku, kde nejmenší cílové bloky zabírají jediný pixel, a po každé iteraci zobrazení pak zvětšit obrázek na dvojnásobek (efektivní především při dělení čtyřstromem). V plné velikosti pak stačí udělat jen okolo dvou iterací.

Progresivní dekódování

Při praktickém využití je důležitá možnost progresivního dekódování obrázků — generování náhledů s použitím pouze malé počáteční části souboru. To je užitečné především na internetových stránkách, nebo při prohlížení adresářů s velkým množstvím fotografií. Progresivním dekódováním pro klasický PIFS se zabývá [15].

Při použití modifikací nahrazující konstantní koeficienty barevných zobrazení průměrnými barvami cílových bloků lze ale postupovat mnohem efektivněji — uložit nejprve způsob rozdělení na bloky (zabere velmi málo místa), potom průměrné barvy všech cílových bloků a pak až ostatní parametry. Tím lze generovat z velmi malé části souboru náhled jen z průměrných barev bloků, který může vypadat relativně dobře, obzvláště pokud se pro zobrazení použijí pokročilé interpolační metody.

2.3.4 Využití technik fraktální komprese pro jiné účely

Fraktální komprese má velmi zajímavé vlastnosti, které lze využít i pro další, často velmi odlišné, účely. V této části budou nějdůležitějších z nich zmíněny, ale jen velmi stručně a bez odkazů na zdroje.

Díky možnosti dekódování v libovolném rozlišení lze použít fraktální kompresi pro inteligentní zvětšování obrázků. Při správném nastavení dává velmi dobré výsledky díky tomu, že zvětšenina nepůsobí rozmazaným dojmem a ostré hrany jsou výborně zachovány.

Z podobných důvodů lze fraktální kompresi použít i pro čištění obrázků od šumu. Při kontraktivních zobrazeních mezi bloky se totiž šum velmi potlačí a proto nelze typicky najít zdrojový blok, který by šum v cílovém bloku napodobil. Větší útvary a ostré hrany se naopak většinou podaří mezi zdrojovými bloky najít, takže zůstanou zachovány.

Je také zkoumáno přidávání „vodoznaků“ do obrázků pomocí fraktální komprese. Ty mají velmi zajímavé vlastnosti — nejsou vidět (ale lze je z obrázku extrahovat) a jsou velmi odolné proti ořezávání obrázku, JPEG kompresi a jiným pokusům o odstranění.

Dále je diskutována možnost využití technik fraktální komprese pro videonahrávky. Největší redundance je zde v podobnosti mezi po sobě následujícími snímky. V klasických metodách se používají bloky některých snímků jako predikce stejně umístěných (případně trochu posunutých) bloků jiných snímků a kóduje se jen rozdíl. Stejnou metodu lze využít i při fraktální kompresi, ale nabízí se mnohem přirozenější rozšíření. Stačí k dvoudimenzionálnímu prostoru pixelů v obrázcích přidat třetí dimenzi tvořenou časem. Potom videozáznam dělený na „kvádry“ místo obdélníků lze komprimovat podobným způsobem jako obrázky pomocí zobrazení mezi trojdimenzionálními bloky. Zde je zajímavá například možnost dekódování ve vyšším rozlišení — lze tak generovat rozumné mezisnímky, které v původní nahrávce nebyly.

Kapitola 3

Chyba zobrazení mezi bloky

V této kapitole je rozebrána nejdůležitější část komprese — výpočet chyby optimálního zobrazení mezi dvojicí bloků.

V následujících výpočtech budeme pro zjednodušení pracovat se zdrojovým blokem, který je už zmenšen na velikost cílového a správně otočen. Navíc použijeme diskrétní model obrázku, přestože analogická odvození lze udělat i se spojitým modelem. Protože zde pracujeme s RMSE metrikou, nehraje tvar bloků žádnou roli. Bloky budou uvažovány jako jednorozměrné vektory hodnot, indexované tak, aby sobě odpovídající hodnoty zdrojových a cílových bloků měly stejné indexy (ovlivněno rotacemi). Hodnoty i-tého pixelu zdrojového a cílového bloku budeme značit d_i a r_i, kde index i je z rozsahu $\{1, 2, \ldots, n\}$.

Je potřeba si uvědomit, že zde jsou počítané odchylky mezi původním obrázkem a jeho obrazem, přestože ve skutečnosti potřebujeme omezit odchylku obrázku od pevného bodu zobrazení. V části 2.2 byla diskutována možnost použití kolážové věty, která skutečnou chybu omezuje, ale v praxi je nepoužitelná. Hlavním důvodem je to, že dává příliš volný odhad, který neodpovídá realitě a také je nepříjemné, že plnění předpokladu kontraktivity barevných zobrazení výslednou odchylku výrazně zvětšuje.

3.1 Chybové metriky

Ve fraktální kompresi se používá několik ukazatelů míry odchylky obrázku od originálu, které jsou vzájemně velmi provázané. Výpočty vycházejí z e_i — rozdílů hodnot jednotlivých pixelů (uvažujeme jednobarevný obrázek).

Pro výpočty je velmi vhodná SE (*square error*) metrika měřící čtvercovou

chybu.

$$\mathrm{SE} = \sum_{i=1}^{n} e_i^2$$

Někdy se používá její normalizovaná verze, která umožňuje korektní porovnávání odchylek naměřených na různě velkých obrázcích. Rozsah RMSE (*root mean square error*) je vždy stejný jako rozsah hodnot pixelů.

$$\mathrm{RMSE} = \sqrt{\frac{\mathrm{SE}}{n}}$$

Pro porovnávání různých metod je vhodnějším měřítkem PSNR kvalita (*peak signal-to-noise ratio*), protože je nezávislá na rozsahu hodnot pixelů a je v logaritmickém měřítku. Hodnoty se měří v decibelech a jsou v rozsahu nula až nekonečno, kde nekonečno odpovídá nulové odchylce.

$$\mathrm{PSNR} = -20 \log_{10} \frac{\mathrm{RMSE}}{M}, \quad \text{kde hodnoty pixelů jsou z rozsahu } 0\text{–}M.$$

3.2 Klasický model

Při klasické fraktální kompresi je chyba zobrazení dána

$$\mathrm{SE} = \sum_{i=1}^{n} \left(u d_i + v - r_i\right)^2,$$

přičemž uvažujeme minimum pro libovolné konstanty u a v. Úloha odpovídá lineární regresi používané ve statistice, kde RMSE metrika dává volbu koeficientů metodou nejmenších čtverců. Použitím derivací a řešením jednoduché soustavy rovnic[1]

$$\begin{aligned} 2u \sum d_i^2 + 2v \sum d_i - 2 \sum r_i d_i &= 0 \\ 2u \sum d_i + 2nv - 2 \sum r_i &= 0 \end{aligned}$$

lze spočítat obecné optimální hodnoty koeficientů

$$\hat{u} = \frac{n \sum r_i d_i - \left(\sum d_i\right)\left(\sum r_i\right)}{n \sum d_i^2 - \left(\sum d_i\right)^2}, \qquad \hat{v} = \frac{\left(\sum d_i^2\right)\left(\sum r_i\right) - \left(\sum d_i\right)\left(\sum r_i d_i\right)}{n \sum d_i^2 - \left(\sum d_i\right)^2}.$$

[1] rozsahy u sum jsou dále vynechány, protože jsou vždy stejné a zbytečně by výpočty znepřehledňovaly

Jmenovatele jsou nulové právě když má zdrojový blok konstantní barvu, což je případ, který můžeme vyloučit například proto, že při něm chyba zobrazení nemůže být lepší než s libovoným jiným zdrojovým blokem. Díky tomu, že matice druhých parciálních derivací

$$\begin{pmatrix} \frac{\partial^2 \mathrm{SE}}{\partial u^2} & \frac{\partial^2 \mathrm{SE}}{\partial u\,\partial v} \\ \frac{\partial^2 \mathrm{SE}}{\partial u\,\partial v} & \frac{\partial^2 \mathrm{SE}}{\partial v^2} \end{pmatrix} = \begin{pmatrix} 2\sum d_i^2 & 2\sum d_i \\ 2\sum d_i & 2n \end{pmatrix}$$

je pozitivně definitní pro libovolný nekonstantní zdrojový blok, jedná se o jediné minimum. Zpětným dosazením optimálních parametrů do vzorce pro chybu a úpravou získáme optimální chybu

$$\widehat{SE} = \frac{\left(\sum r_i d_i\right)\left[2\left(\sum d_i\right)\left(\sum r_i\right) - n\sum r_i d_i\right] - \left(\sum d_i^2\right)\left(\sum d_i\right)^2}{n\sum d_i^2 - \left(\sum d_i\right)^2} + \sum r_i^2.$$

Pomocí struktur uvedených v 4.5 lze všechny sumy kromě jedné snadno spočítat. Součet $\sum r_i d_i$ je nutné vyčíslit pro každou uvažovanou dvojici bloků zvlášť.

3.3 Chyba upravených zobrazení

V implementaci je použita úprava klasického modelu popsaná v části 2.2.4, kde místo koeficientů u a v je uložena průměrná hodnota a směrodatná odchylka hodnot cílového bloku. Koeficienty zobrazení jsou pak počítány v každé iteraci z těchto uložených hodnot a ze stavu zdrojového bloku v předchozí iteraci. Nová definice koeficientů je

$$\bar{u} = \pm\frac{\sqrt{n\sum r_i^2 - \left(\sum r_i\right)^2}}{\sqrt{n\sum \bar{d}_i^2 - \left(\sum \bar{d}_i\right)^2}}, \qquad \bar{v} = \frac{1}{n}\sum r_i - \frac{\bar{u}}{n}\sum \bar{d}_i,$$

kde $\bar{d}_i$ je značení pro hodnotu pixelu zdrojového bloku z přechozí iterace.

Tím je ovlivněna i optimální chyba zobrazení, protože ani za předpokladu zobrazování původního obrázku se nemusí $\bar{u}$ rovnat u. Pro následující odvození bude použito statistické značení, které je zde názornější:

výběrová směrodatná odchylka $s_r = \sqrt{\frac{1}{n-1}\left[\sum_{i=1}^{n} r_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} r_i\right)^2\right]}$

výběrová kovariance $q_{d,r} = \frac{1}{n-1}\left(\sum_{i=1}^{n} d_i r_i - \frac{1}{n}\sum_{i=1}^{n} d_i \sum_{i=1}^{n} r_i\right)$

výběrová korelace $\rho_{d,r} = \dfrac{q_{d,r}}{s_d s_r}$

konvergence frázemi typu „*pokud* $a \to b$, *pak* $c \to d$“ je myšlen zápis $\lim\limits_{a \to b} c = d$

Optimální lineární koeficient $u = \dfrac{q_{d,r}}{s_d^2}$ bude nahrazen koeficientem $\bar{u} = \pm\dfrac{s_r}{s_{\bar{d}}}$. Pokud povolíme zobrazení se záporným lineárním koeficientem, je nutné také uložit znaménko (zde značeno symbolem $\pm$). Při implementaci je navíc potřeba ošetřit situaci, kdy $s_{\bar{d}} = 0$, což nastává hlavně při první iteraci pro typický případ inicializace celého obrázku šedivou barvou.

Pokud $s_{\bar{d}} \to s_d$ (platí speciálně když $\bar{d} \to d$), pak

$$\frac{u}{\bar{u}} \to \pm\frac{q_{d,r}}{s_d^2}\frac{s_d}{s_r} = \pm\frac{q_{d,r}}{s_d s_r} = \pm\rho_{d,r}.$$

Pokud byla chyba barevného zobrazení s koeficientem u malá (konverguje k nule), pak korelace $\rho_{d,r} \to \pm 1$. Navíc znaménko u je stejné jako znaménko $\rho_{d,r}$, takže $\bar{u} \to u$. Z toho je zřejmé, že cílový obrázek bude pevným bodem i takto pozměněného operátoru.

Novou čtvercovou chybu lze vyjádřit jako

$$\overline{\mathrm{SE}} \;=\; \sum_{i=1}^{n} \left(\bar{u}\bar{d}_i + \bar{v}\right)^2. \tag{3.1}$$

Po postupném dosazení $\bar{v} = \frac{1}{n}\sum r_i - \frac{\bar{u}}{n}\sum \bar{d}_i$ a $\bar{u} = \pm\dfrac{s_r}{s_{\bar{d}}}$ dostaneme

$$\overline{\mathrm{SE}} \;=\; \sum_{i=1}^{n}\left[\bar{u}\left(\bar{d}_i - \frac{1}{n}\sum \bar{d}_j\right) - \left(r_i - \frac{1}{n}\sum r_j\right)\right]^2 \;=$$

$$=\; \bar{u}^2\sum_{i=1}^{n}\left(\bar{d}_i - \tfrac{\sum \bar{d}_j}{n}\right)^2 - 2\bar{u}\sum_{i=1}^{n}\left(\bar{d}_i - \tfrac{\sum \bar{d}_j}{n}\right)\left(r_i - \tfrac{\sum r_j}{n}\right) + \sum_{i=1}^{n}\left(r_i - \tfrac{\sum r_j}{n}\right)^2 \;=$$

$$=\; n\bar{u}^2 s_{\bar{d}}^2 - 2n\bar{u}q_{\bar{d},r} + ns_r^2 \;=\; n\frac{s_r^2}{s_{\bar{d}}^2}s_{\bar{d}}^2 \mp 2n\frac{s_r}{s_{\bar{d}}}q_{\bar{d},r} + ns_r^2 \;=\; 2n\left(s_r^2 \mp \frac{s_r}{s_{\bar{d}}}q_{\bar{d},r}\right).$$

Protože $q_{\bar{d},r}$ má vždy stejné znaménko jako $\bar{u}$, lze výraz upravit na

$$\overline{\mathrm{SE}} \;=\; 2ns_r^2\left(1 - \frac{\left|q_{\bar{d},r}\right|}{s_{\bar{d}}s_r}\right) \;=\; 2ns_r^2\left(1 - \left|\rho_{\bar{d},r}\right|\right).$$

Původní optimální čtvercovou chybu lze přepsat do tvaru:

$$\widehat{\mathrm{SE}} \;=\; n\frac{s_d^2 s_r^2 - q_{d,r}^2}{s_d^2} \;=\; n s_r^2 \left(1 - \rho_{d,r}^2\right) \;=\; n s_r^2 \left(1 - |\rho_{d,r}|\right)\left(1 + |\rho_{d,r}|\right).$$

Za předpokladu, že $s_{\bar{d}} \to s_d$, dostaneme absolutní a relativní nárůst chyby

$$\overline{\mathrm{SE}} - \widehat{\mathrm{SE}} \;=\; n s_r^2 \left(1 - |\rho_{d,r}|\right)\left[2 - \left(1 + |\rho_{d,r}|\right)\right] \;=\; n s_r^2 \left(1 - |\rho_{d,r}|\right)^2$$

$$\frac{\overline{\mathrm{SE}} - \widehat{\mathrm{SE}}}{\widehat{\mathrm{SE}}} = \frac{\left(1 - |\rho_{d,r}|\right)^2}{\left(1 - |\rho_{d,r}|\right)\left(1 + |\rho_{d,r}|\right)} = \frac{1 - |\rho_{d,r}|}{1 + |\rho_{d,r}|}$$

Pro vzájemně dobře zobrazitelné bloky $|\rho_{d,r}| \to 1$, takže nárůst chyby bude velmi malý.

3.4 Penalizace vysokého lineárního koeficientu

Výše zmíněnou úpravou barevných zobrazení bylo sníženo riziko divergence zobrazení bez nutnosti omezení lineárního koeficientu. Díky tomu, že všechny hodnoty jsou omezeny v intervalu $\langle 0, 1\rangle$ a cílové bloky mají vždy správnou průměrnou hodnotu, nemohou se obrazy zobrazení od původního obrázku příliš lišit (v praxi funguje dobře, ale bez jakéhokoliv důkazu omezení odchylky).

Přesto jsou vysoké lineární koeficienty nebezpečné, protože zdrojové bloky jsou při dekódování vždy pouze aproximací jejich stavu v původním obrázku. Čím vyšší je lineární koeficient, tím více se případná odchylka zdrojového bloku distribuuje do cílového a poté případně v dalších iteracích i do jiných zdrojových bloků.

Proto je zde navržena následující penalizace, která zvyšuje odhadovanou chybu zobrazení v závislosti na jeho lineárním koeficientu. Pokud vyjdeme ze vzorce pro chybu (3.1 na předchozí straně) a přičteme k hodnotám zdrojového bloku odchylku ϵ_i, dostaneme

$$\widetilde{\mathrm{SE}} \;=\; \sum_{i=1}^{n} \left[\bar{u}\left(\bar{d}_i + \epsilon_i\right) + \bar{v}\right]^2 \;=\; \sum_{i=1}^{n} \left[\left(\bar{u}\bar{d}_i + \bar{v}\right)^2 + \left(\bar{u}\bar{d}_i + \bar{v}\right)\bar{u}\epsilon_i + \bar{u}^2\epsilon_i^2\right].$$

Když označíme původní odchylku $e_i = \bar{u}\bar{d}_i + \bar{v}$, pak

$$\widetilde{\mathrm{SE}} \;=\; \sum e_i^2 + \bar{u}\sum e_i\epsilon_i + \bar{u}^2\sum\epsilon_i^2.$$

První suma je zřejmě rovna původní chybě $\overline{\mathrm{SE}}$ (vzorec 3.1). Hodnota druhé sumy je závislá na korelaci odchylek způsobených nepřesností zobrazení a odchylek ve zdrojovém bloku. Lze snadno ukázat, že $\sum e_i = 0$ (platí díky tomu, že optimální zobrazení v RMSE metrice zobrazují průměrnou hodnotu zdrojového bloku na průměrnou hodnotu cílového bloku). Za předpokladu, že chyby ϵ_i nejsou korelované, vyjde nulová očekávaná hodnota[2] druhé sumy.

Třetí suma odpovídá odchylce zmenšeného zdrojového bloku. Pokud požadovanou odchylku pro cílový blok označíme SE' a poměr velikostí zdrojového bloku po zmenšení a před zmenšením označíme $\alpha \in (0, 1)$, pak lze očekávat, že se bude odchylka zdrojového bloku před zmenšením pohybovat okolo $\frac{\mathrm{SE}'}{\alpha}$. Když při zmenšování jeden pixel vzniká jako průměr $\frac{1}{\alpha}$ pixelů zdrojového bloku s nekorelovanými odchylkami, pak očekávaná čtvercová chyba zmenšeného zdrojového bloku je rovna $\alpha \cdot \mathrm{SE}'$, tedy nová chyba zobrazení zahrnující očekávanou odchylku se zjednoduší na

$$\widetilde{\mathrm{SE}} \;\approx\; \overline{\mathrm{SE}} \;+\; \bar{u}^2 \alpha\, \mathrm{SE}' \;=\; \overline{\mathrm{SE}} \;+\; \alpha \frac{s_r^2}{s_{\bar{d}}^2}\, \mathrm{SE}'$$

Celé odvození penalizace lze analogicky provést i pro klasický model zobrazování (viz 3.2), ale v obou případech jsou používany nepodložené předpoklady, především předpoklad nekorelovanosti chyb jednotlivých pixelů, který by bylo vhodné alespoň statisticky ověřit.

Implementace příslušného modulu (popsaná v 4.9) dává možnost započítat penalizaci do chyb zobrazení. Při jejím použití je dosahována výrazně vyšší kvalita dekódovaného obrázku než při klasickém počítání s omezenou nebo neomezenou velikostí lineárních koeficientů za cenu pouze nepatrného poklesu kompresního poměru.

[2] spojení *očekávaná hodnota* (nebo také *střední hodnota*) je zde používáno ve smyslu teorie pravděpodobnosti a statistiky

Kapitola 4

Implementace

Cíle a zásady

Cílem implementace bylo vytvořit framework pro studium fraktální komprese, kde by byly jednotlivé části algoritmů oddělené do modulů, aby bylo možné kompatibilní algoritmy snadno vzájemně vyměnovat a aby implementace jednotlivých technik byla tvořena převážně algoritmickým kódem, odděleně od správy závislostí mezi moduly, interakce s uživatelem a nezávisle na způsobu implementace ostatních modulů.

Aby bylo možné jednotlivé techniky objektivně porovnávat, je kladen důraz na efektivitu implementace. Proto je v některých algoritmech, kde by dělení do více modulů významným způsobem snížilo efektivitu, upřednostněna konfigurace modulu pomocí jiných parametrů.

Zvolené vývojové nástroje

Jako programovací jazyk pro implementaci bylo zvoleno C++, především kvůli kombinaci vysoké efektivity a mnoha pokročilých vlastností (například generické programování). Pro překlad byl používán překladač z rodiny GNU překladačů (GCC [16]).

Grafické uživatelské rozhraní (GUI) je realizováno pomocí knihoven Qt [19], které také zajišťují čtení a zápis obrázků ve standardních formátech (BMP, PNG, JFIF).

Díky použitým prostředkům by měl být celý program snadno přenositelný (na úrovni zdrojového kódu). Vývoj a ladění probíhaly na GNU/Linux, s testováním také na MS Windows XP.

4.1 Modularizace

Oddělení modulů do jednotlivých dynamických knihoven by zde nepřineslo významný užitek, takže byla zvolena kompilace celého programu do jednoho spustitelného souboru (kromě knihoven Qt). Přesto jsou zde minimalizovány vzájemné závislosti mezi moduly.

Deklarace související s modularizací jsou v souboru modules.h, většina implementace je pro přehlednost v modules.cpp.

Základní vlastnosti modulů

Všechny moduly jsou potomky třídy `Module`, která definuje základní společné vlastnosti, především způsob reprezentace nastavení. Je zde virtuální metoda `info`, která vrací referenci na strukturu obsahující informace o typu modulu — jeho identifikační číslo, jméno, text popisující funkci, počet parametrů nastavení a ukazatel na pole obsahující vlastnosti jednotlivých parametrů nastavení (typ, rozsah hodnot, název a popis). Dále třída `Module` obsahuje ukazatel na pole skutečných hodnot nastavení pro daný modul a také spoustu dalších pomocných typů a metod.

Tento způsob práce s nastaveními umožňuje jejich automatické měnění, ukládání a zobrazování libovolným uživatelským rozhraním. Vše pomocí společného kódu odděleného od všech modulů.

Obsah zmíněné informační struktury je potřeba pro každý typ modulu nadefinovat, což by znamenalo mnoho opakujícího se technického kódu, a proto jsou pro tento účel vytvořeny funkce a makra, kterým stačí předat pouze jednotlivé vlastnosti (název, typy parametrů, ...), takže implementace jednotlivých modulů je jednodušší a přehlednější.

Správa modulů

Aby framework mohl s modulem pracovat, je potřeba upravit soubor modules.cpp vložením hlavičkového souboru (pomocí `#include`) a přidáním typu modulu do seznamu `Modules`. Společný kód v souboru se postará o možnost volby modulu, závislosti a další technické záležitosti. Zde je nejsložitější generování některých metod z tohoto seznamu typů, pro které je využita část z knihovny, kterou napsal Andei Alexandrescu jako přílohu ke knize *Modern C++ Design* [18].

Pro správu modulů a vytváření nových slouží třída `ModuleFactory`. Ta má jedinou instanci, která při startu vytvoří prototyp pro každý typ modulu. Prototypy uchovávají výchozí nastavení jednotlivých typů modulů a je možné

je použít pro získání obecných informací o typech modulů (metoda `info`) nebo pro vytváření dalších instancí.

Nové moduly jsou vytvářeny zásadně klonováním, mělkým nebo hlubokým. Oba způsoby vytvoří novou instanci pomocí kopírovacího konstruktoru a zkopírují nastavení modulu. Mělké klonování navíc vynuluje odkazy na podřízené moduly, zatímco hluboké klonování je naklonuje stejným způsobem. Hluboké klonování se hodí například pro vytvoření nového výchozího stromu modulů (například před kompresí) a mělké se používá při načítání ze souborů, kde není ve chvíli vytváření modulů zřejmé, jakých typů budou podřízené moduly.

Závislosti mezi moduly

Jsou definována rozhraní, každé popisující jednu požadovanou funkcionalitu. Všechna použitá rozhraní se nacházejí v souborech interfaces.h a interfaces.cpp. Každé z nich je potomkem šablonované třídy `Interface`, která zpřístupňuje seznam identifikátorů všech typů modulů implementujících toto rozhraní (kód je vygenerován ze seznamu `Modules`).

Každý modul implementuje právě jedno rozhraní a může využívat libovolné množství dalších rozhraní — jedním z typů parametrů modulů je podřízený modul implementující zvolené rozhraní. Uživatel pak na tato místa může zapojit libovolný typ modulu (jsou zobrazeny pouze kompatibilní typy). Konfigurace komprese je tedy dána tímto stromem typů modulů spolu s jejich dalším nastavením.

4.2 Kořenový modul

IRoot

Rozhraní pro kořenový modul je navrženo jako jediný přístupový bod pro GUI ke kompresi a dekompresi.

Kořen včetně celého stromu může být ve třech různých stavech. V počátečním prázdném stavu strom neobsahuje žádný obrázek, ale může v něm docházet ke změnám konfigurace, včetně vyměňování podřízených modulů.

Z počátečního stavu může přejít do komprimovaného stavu úspěšným voláním metody `encode`, které je předán bitmapový obrázek ke kompresi (třída knihoven Qt) a reference na strukturu umožňující komunikaci s jiným vláknem během tohoto často zdlouhavého procesu. Pokud nebylo možné kompresi dokončit (například kvůli přerušení uživatelem), vrátí metoda `false` a strom zůstává v počátečním stavu.

Druhou možností je přejít z počátečního stavu do dekomprimovaného stavu úspěšným voláním metody `fromStream`, které je předán proud bytů obsahující komprimovanou podobu obrázku a míra přiblížení pro dekompresi. Rozměry dekomprimovaného obrázku jsou původní rozměry vynásobené 2^i pro nezáporné i. Pokud nebylo možné dekódovat obrázek, například kvůli špatnému formátu souboru, vrátí metoda `false` a stav stromu se nezmění.

Dále rozhraní obsahuje metody pro ukládání obrázku v komprimovaném stavu do proudu bytů, pro provádění dekódovacích akcí (vyčištění obrázku nebo iterování zobrazení), pro uložení momentálního stavu dekódování do bitmapového obrázku (třída knihoven Qt) a pro ukládání a nahrávání nastavení celého stromu.

MRoot (soubory modules/root.*)

Implementace kořenového modulu umožňuje jednotné nastavení nejzákladnějších parametrů komprese a rozděluje fáze procesu mezi několik nezávislých modulů.

Kvůli uživatelské přívětivosti bylo zvoleno nastavování kvality komprese v rozsahu 0–100. Hodnota je pomocí zvoleného modulu s rozhraním `IQuality2SE` konvertována na nejvyšší možnou čtvercovou chybu pro danou velikost cílového bloku (modul je ve skutečnosti použit až v podřízených modulech).

Dále lze zvolit modul s rozhraním `IColorTransformer` starající se o barvy v obrázku, modul s rozhraním `IShapeTransformer` zajišťující další fáze komprese a také maximální množství zdrojových bloků, protože je to velmi důležitý parametr pro všechny metody fraktální komprese.

4.3 Konverze kvality

IQuality2SE

Rozhraní pro konverzi kvality — z velikosti cílového bloku a nastavení kvality spočítá odpovídající maximální přípustnou čtvercovou odchylku.

MQuality2SE (soubor modules/quality2SE.h)

Třída `MQuality2SE_std` je triviální implementací rozhraní pro konverzi kvality, která používá čtvercovou odchylku nezávislou na velikostech bloků. Přípustná odchylka je pro nejvyšší kvalitu nulová a pak roste přibližně exponenciálně (exponenciální funkce je mírně posunutá tak, aby procházela nulou při kvalitě 100%).

Bylo by sice přirozenější mít pro danou kvalitu místo čtvercové chyby konstantní její podíl s počtem pixelů bloku. To by vedlo k větší variabilitě velikostí bloků a ve většině případů i k nepatrně lepší objektivní kvalitě (při stejném kompresním poměru), ale protože v tomto projektu nebyly implementovány žádné vyhlazovací techniky, objevily by se u velkých bloků velmi výrazné hrany, takže by se subjektivní kvalita zhoršila. Pro porovnání je implementována i tato možnost v podobě modulu `MQuality2SE_alt`.

4.4 Zpracování barev

IColorTransformer

Rozhraní převádí práci s jedním barevným obrázkem na práci s několika jednobarevnými. Barevné obrázky jsou reprezentované třídou `QImage` z knihoven Qt, jednobarevné obrázky jsou zde definovány jako matice pixelů spolu s ukazatelem na strukturu obsahující některé parametry (rozměry, přiblížení, kvalita komprese a modul pro její konverzi, maximální počet zdrojových bloků).

Metody rozhraní umožňují vytvoření jednobarevných obrázků pro dané parametry a barevný obrázek, zpětné složení do barevného obrázku, uložení nebo načtení případných dat modulu.

MColorModel (soubory modules/colorModel.*)

Třída `MColorModel` je triviální implementací rozhraní pro zpracování barev, která nabízí práci v modelech RGB a YCbCr. Protože je citlivost oka na jednotlivé barevné složky různá, je navíc umožněno zvolit pro každou ze složek číslo z intervalu $[0; 1]$, kterým se vynásobí původní kvalita komprese.

4.5 Počítání s maticemi pixelů

V souboru matrixutil.h je definováno mnoho generických struktur a několik funkcí pro jednoduchou a efektivní práci s maticemi. Tyto nástroje jsou pak použity v mnoha modulech a jejich rozhraních.

Reprezentace pixelů

Při ukládání digitálních obrázků se pro reprezentaci pixelu nejčastěji používá celé číslo v rozsahu 0–255. Tento způsob je paměťově velmi úsporný, ale pro tento projekt se ukázal jako příliš omezující. Například už při převodu do jiných barevných modelů by mohlo docházet k výraznějším ztrátám

informace. Proto byla zvolena reprezentace pomocí čísla s plovoucí řádovou čárkou z intervalu $[0; 1]$. Pro reprezentaci běžných pixelů postačují čísla s *jednoduchou přesností*, ale pro většinu složitějších výpočtů a mezivýsledků jsou použita čísla s *dvojitou přesností*, aby byly minimalizovány zaokrouhlovací chyby.

Reprezentace matic

Matice jsou reprezentovány tak, že jsou jejich prvky po sloupcích uloženy za sebe do pole. Vlastní práci s maticemi zajišťuje šablona `MatrixSlice` (parametrizovaná typem prvků matice), která uchovává ukazatel na levý horní prvek a počet kroků nutných k přechodu na další sloupec. Navenek poskytuje přirozené indexování, konverzi na instanci pouze pro čtení a další operace.

Oproti reprezentacím založených na ukládání jednotlivých řádkových nebo sloupcových vektorů má toto řešení mnoho výhod. Má menší nároky na spotřebu paměťi a na její alokátor, průchody jsou překladačem lépe optimalizovatelné a je možné vytvářet odkazy na podmatice.

Rychlé sčítání podmatic

Některé moduly potřebují znát součty hodnot pixelů a jejich druhých mocnin na blocích se kterými pracují. Pro tyto účely je používána jednoduchá struktura (implementovaná v šabloně `MatrixSummer`), která si předpočítá součty pro všechny obdélníky začínající v levém horním rohu a pak může spočítat součet libovolného obdélníka pouze ze čtyř rohových hodnot.

Šablona `MatrixSummer` není používána samostaně, ale jako součást šablony `SummedMatrix`, která kromě matice vlastních hodnot obsahuje její rozměry a šablonu `MatrixSummer` pro rychlý výpočet součtů hodnot a jejich druhých mocnin. Konkrétně je definována struktura `SummedPixels`, která je jen instanciací šablony `SummedMatrix` pro již zmíněné typy hodnot pixelů používané v projektu.

Iterace na matici

Na mnoha místech v projektu je potřeba současně procházet dvě stejně velké obdélníkové podmatice a provádět nějakou činnost s příslušnými dvojicemi jejich prvků, navíc je často nutné počítat s libovolnou z 8 vzájemných afinních transformací podmatic. Opakovaně ručně psané cykly by velmi znepřehledňovaly kód a byly by potencionálním zdrojem chyb. Proto je zde použit jiný přístup využívající generické programovaní a optimalizace kompilátoru.

Základem celé techniky je funkce `walkOperate`, která má tři parametry — řídící iterátor, druhý (podřízený) iterátor a operátor. Funkce pak pouze najednou posouvá oba iterátory a na odpovídajících dvojicích prvků volá operátor. Díky tomu, že se jednotlivé části cyklu takto oddělí, je možné konkrétní alternativy implementovat a pak libovolně kombinovat.

Iterátory musí podporovat metody pro získání aktuálního prvku (nejlépe formou reference), posun na další prvek, posun na další seznam (řádek/sloupec), zahájení práce se seznamem a v případě řídícího iterátoru také test konce seznamu a test úplného konce iterace. Operátor pak musí být schopný volání s parametry prvků prvního a druhého iterátoru. Díky tomu, že funkce `walkOperate` vrací koncový stav operátoru, lze jako operátor použít i strukturu, která do sebe sbírá nějaká data.

V souboru jsou přímo implementovány iterátory pro osm afinních zobrazení a také přetížení funkce `walkOperate`, které má v parametrech místo druhého iterátoru číslo afinní transformace, podle kterého vybere příslušný iterátor. Dále je zde iterátor pro třídu QImage používaný pro konverzi na matici a také několik jednoduchých operátorů pro výpočet součtu vzájemných součinů a pro afinní transformaci hodnot.

4.6 Změny tvarů

IShapeTransformer

Rozhraní pro „transformaci tvarů“ zajišťuje kompresi a dekompresi jednobarevných obrázků (definovaných v rozhraní `IColorTransformer`). Je zamýšleno jako rozcestí pro několik druhů voleb, například pro možnost přechodu k šestiúhelníkovému tvaru pixelů.

Ze souboru vstupujících obrázků je vytvořeno několik nezávislých *úkolů*, pro které lze provádět výpočty vzájemně paralelně. Rozhraní poskytuje metody pro provádění komprese a dekódovacích akcí pro jednotlivé *úkoly* a také ukládání do proudu bytů a zpětné nahrávání.

Data všech *úkolů* jsou ukládána po fázích, což umožňuje přidat v budoucnu implementaci progresivního dekódování, přestože pro něj momentálně není v kořenovém modulu podpora. Důležité pro účinnost progresivního dekódování je, že uspořádání fází má vyšší prioritu než dělení na jednobarevné části způsobené zpracováním barev nebo přílišnou velikostí obrázku. Proto bylo nutné dostat tento koncept už do tak obecného rozhraní.

MSquarePixels (soubory modules/squarePixels.*)

Třída `MSquarePixels` je implementací rozhraní pro „transformaci tvarů“ zůstávající u čtvercového tvaru pixelů, umožňující navíc dělení velkých obrázků na menší, nezávisle (případně i paralelně) zpracovávané. Pro vlastní práci s *úkoly* jsou použity zvolené moduly pro dělení na cílové bloky, pro výběr množiny zdrojových bloků a pro hledání optimálních zobrazení.

Dělení jednobarevných obrázků na *úkoly* vychází z předaných bloků a probíhá rekurzivně. Každý blok, který je větší než nastavená mez, je rozdělen podle delší souřadnice tak, že poměr velikostí je nejvýše dvojnásobný a navíc nově vzniklá strana levé/horní části má délku mocniny 2.

„Čtvercové“ moduly

Na třídu `MSquarePixels` navazuje několik rozhraní a modulů obsahující v názvu slovo *square*. Všechny pracují pouze s klasickou (čtvercovou) topologií pixelů a upřednostňují čtvercové bloky o hraně délky mocniny 2. Často se pro libovolný blok definuje jeho *úroveň*, což je nejmenší i takové, že oba rozměry jsou nejvýše 2^i. Nejmenší povolená *úroveň* je 2, což nejčastěji odpovídá blokům velikosti 4×4.

Moduly s rozhraními `ISquareRanges`, `ISquareDomains` a `ISquareEncoder` (definovanými níže) spolupracující na jenom *úkolu* často potřebují vzájemně volat své metody a také sdílet některá další data. Proto je definována struktura `PlaneBlock` s instancí pro každý *úkol*.

`PlaneBlock` obsahuje vlastní jednobarevnou část obrázku zpracovávnou *úkolem*, schovanou v podstruktuře `SummedPixels` (zmíněné v 4.5). Dále obsahuje ukazatele na moduly spolupracující na *úkolu* a na strukturu s některými nastaveními (definovanou v rozhraní `IColorTransformer`).

4.7 Dělení na cílové bloky

ISquareRanges

Rozhraní pro dělení na obdélníkové cílové bloky definuje základní strukturu `RangeBlock` pro reprezentaci cílového bloku, která obsahuje souřadnice jeho okrajů, přepočítanou *úroveň* bloku a přídavný ukazatel, do kterého si spolupracující modul s rozhraním `ISquareEncoder` může ukládat pomocná data. Moduly implementující rozhraní mohou používat libovolný typ odvozený z `RangeBlock`, ale ostatním je viditelná pouze část definovaná v rozhraní.

Kromě klasických metod pro ukládání a nahrávání nastavení a dalších dat je zde metoda pro získání seznamu ukazatelů na bloky, na které je *úkol*

rozdělen, a metoda pro zpracování celého *úkolu*. Při jejím provádění je celý blok rozdělen na cílové bloky a pomocí volání metod modulů spolupracujících na úkolu je pro každý z nich nalezeno optimální zobrazení. Tyto činnosti jsou spojeny proto, že schopnost nalézt dostatečně dobrá zobrazení typicky ovlivňuje jemnost rozdělení na cílové bloky.

MQuadTree (soubory modules/quadTree.*)

Třída `MQuadTree` implementuje klasický čtyřstromový algoritmus s několika drobnými rozšířeními. Je možné zvolit zda se má používat heuristické dělení a také nejmenší a největší povolenou velikost bloku.

Způsob dělení Původní algoritmus počítá pouze s obrázky tvaru čtverce o hraně mocniny 2, což by zde nebylo dostačující. Naštěstí stačí pracovat s bloky stejným způsobem, jen v některých případech jsou jejich skutečné rozměry menší, protože jsou „ořízlé“ okrajem obrázku. Je pouze potřeba navíc ošetřit případy, kdy by při dělení na čtvrtiny vznikly prázdné bloky.

Dělení čtyřstromem se řídí jednoduchým principem. Na začátku je celý vstupní obrázek jeden cílový blok. Pak se rekurzivně pro každý blok zkouší nalézt dostatečně dobré zobrazení a v případě neúspěchu je blok rozdělen na čtyři o poloviční velikosti. Samozřejmě, bloky větší než nastavené maximum jsou rozděleny bez hledání zobrazení a u bloků o velikosti nastaveného minima se modul vždy spokojí s nejlepším nalezeným zobrazením.

Heuristika Byla zde jedna velká nevýhoda — původní algoritmus slepě zkoušel hledat zobrazení i pro bloky, pro které bylo zřejmé, že nemůže uspět. Proto zde může být dělení prováděno podle jednoduché heuristiky, která rychle odhadne (z rozptylu hodnot v bloku) chybu optimálního zobrazení. Pokud pro nějaký blok bylo hledáno dostatečně dobré zobrazení, ale neúspěšně, je blok dále dělen původním algoritmem. Pokud byl nějaký blok naopak spekulativně rozdělen a všem jeho následníkům se podařilo najít optimální zobrazení, pak se zkusí hledat zobrazení i pro něj a v případě úspěchu je zpětně sloučen (to může způsobit kaskádu slučování).

Heuristické dělení dává stejný výsledek za předpokladu (který platí téměř vždy), že pokud je možné nalézt dostatečně dobré zobrazení pro blok, pak je to možné i pro jeho čtvrtiny. V praxi představuje tato technika významné urychlení, přestože není heuristika moc přesná, například chyby bloků s „náhodným“ šumem jsou podhodnoceny a naopak chyby bloků s jedním pozvolným nebo ostrým přechodem výrazně nadhodnoceny. Urychlení nastává zejména proto, že přeskočené hledání zobrazení pro velké bloky bývá časově velmi náročné, často kvůli nutnosti stavby složitých datových struktur.

Bylo by možné heuristiku ještě výrazně zpřesnit pomocí rozšíření rozhraní `ISquareEncoder` o příslušnou odhadující metodu. Například implementace založené na mnohadimenzionálních prostorových datových strukturách by pak mohly odhadovat optimální chybu mnohem přesněji a také velmi rychle.

Způsob uložení Efektivita uložení do proudu bytů je při kompresi obrazu velmi důležitá. Proto je nejprve uložena největší a nejmenší použitá *úroveň* (tedy ne nastavené limity) a potom při rekurzivním průchodu čtyřstromem je pro blok uložen jeden bit nastavený podle toho, zda blok byl nebo nebyl rozdělen. Bit se ukládá pouze pro bloky, kdy není možné informaci získat už z uložené minimální a maximální *úrovně*.

Cílové bloky jsou vraceny z rozhraní v pořadí jejich výskytu na Hilbertově křivce. Tím jsou sníženy vzájemné vzdálenosti po sobě jdoucích bloků a tedy i zefektivněny diferenční metody používané v implementaci rozhraní `ISquareEncoder` (část 4.9).

4.8 Výběr množiny zdrojových bloků

ISquareDomains

Vytváření rozhraní vybírajícího množinu zdrojových bloků bylo obtížné. Zdrojové bloky lze volit mnoha různými způsoby. Pokud by ale bylo rozhraní příliš obecné, mohla by se velmi výrazně snížit rychlost komprese a dekomprese.

Rozhraní poskytuje několik matic pro cílové bloky, jejichž obsah je už zmenšen tak, že zbytek zobrazení mezi zdrojovými a cílovými bloky nemění rozměry. Poskytované zdrojové bloky jsou vždy čtverce o hranách délky mocniny 2, které jsou pravidelně rozmístěny po matici. Omezení velikostí zdrojových bloků ve skutečnosti neomezuje velikosti cílových bloků, protože modul pro zobrazování (s rozhraním `ISquareEncoder`) nemusí využít celý blok. Pro každou z těchto matic a pro každou *úroveň* bloku je rozhraním poskytnut rozestup — počet pixelů, o který jsou vzdáleny odpovídající rohy zdrojových bloků v obou souřadnicích.

Metody rozhraní umožňují inicializaci modulu pro daný *úkol*, aktualizaci hodnot pixelů v maticích pro zdrojové bloky (ze stavu *úkolu*), vrácení seznamu těchto matic, ukládání a nahrávaní nastavení a dalších dat. Také je zde metoda pro zjištění hustoty pokrytí zdrojovými bloky pro všechny matice, danou úroveň a daný maximální počet zdrojových bloků.

MStdDomains (soubory modules/stdDomains.*)

Třída `MStdDomains` implementuje rozhraní `ISquareDomains` a nabízí mnoho možností nastavení.

Předně lze regulovat maximální množství zdrojových bloků v závislosti na jejich *úrovni*. Původní maximum platí pro *úroveň* 2, pro vyšší lze nastavit míru poklesu (koeficient geometrické řady).

Druhy zdrojových bloků Třída umožňuje vytvářet několik druhů matic pro zdrojové bloky. Základní matice jsou čtyř druhů: *klasické*, *vodorovné*, *svislé* a *kosočtvercové*. Vzájemné poměry množství zdrojových bloků generovaných z matic jednotlivých druhů může uživatel nastavit (přibližně).

Klasické matice vznikají zmenšením celého vstupu v obou směrech na polovinu pomocí průměrování hodnot pixelů po čtveřicích. To je ve fraktální kompresi nejčastější metoda, protože je velmi jednoduchá, rychlá a dává v praxi dobré výsledky. Další způsoby zmenšování jsou inspirovány především článkem [2], ale podobné návrhy lze najít i na mnoha jiných místech.

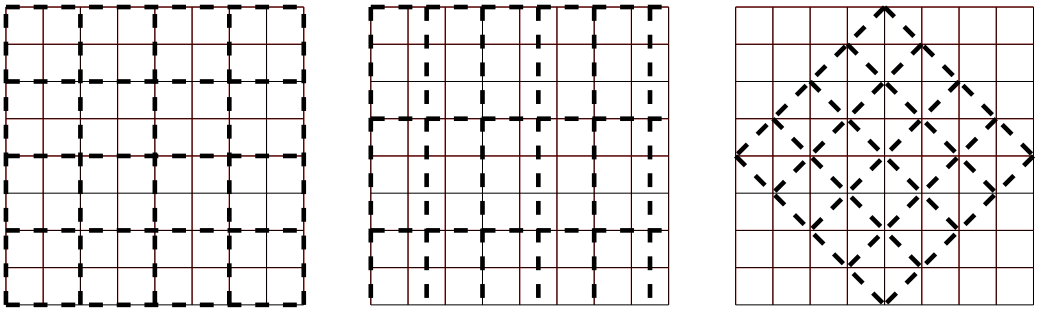
Vodorovné a *svislé* matice vznikají nerovnoměrným zmenšením celého vstupu v obou směrech — v jednom směru probíhá zmenšení na 1/3 a v druhém směru na 2/3.[1] Zmenšování probíhá výpočtem váženého průměru, kde jeden pixel vzniká ze tří celých a tří polovin pixelů, viz obrázek 4.1.

Kosočtvercové matice vznikají otáčením částí vstupu o 45° proti směru hodinových ručiček. Přes svůj název (odvozený z anglického *diamond* z [2]) vznikají z části o tvaru pootočeného čtverce. Zde je použit jednoduchý způsob zmenšování — středy nových pixelů leží na některých rozích vstupních pixelů a jejich hodnota je spočtena jako průměr hodnot čtveřice vstupních pixelů sdílející daný roh. Při čtvercovém vstupu se takto využije jen polovina jeho plochy a pokud by poměr stran byl větší, bylo by to ještě méně. Proto jsou takové vstupy pokryty více *kosočtvercovými* maticemi rozmístěnými tak, aby nevznikaly identické zdrojové bloky, viz obrázek 4.1.

Vícenásobné zmenšování Ze základních matic lze ještě vytvářet další, ještě více zmenšené (a z nich rekurzivně další). Použit je stejný algoritmus jako pro vytváření *klasických* matic. Lze navíc zvolit, jak velké množství zdrojových bloků bude z těchto matic vytvořeno — buď stejné ze všech zmenšujících kroků (přestože ty více zmenšené mají menší plochu) nebo se

[1] Ve starší implementaci projektu probíhalo zmenšování pouze v jednom směru, což těsně porušovalo podmínku kontraktivity z 2.2.3. Při okrajovém nastavení pak bylo snadno možné dosáhnout obrázku, kde některé bloky konvergovaly velmi pomalu a k chybnému vodorovně nebo svisle „pruhovanému“ výsledku.

Obrázek 4.1: Schéma zmenšování pro *klasické*, *svislé* a *kosočtvercové* zdrojové bloky. Tenké čáry značí původní pixely, čárkované značí jejich pokrytí pixely po zmenšení.



bude s každým krokem množství vytvořených bloků zmenšovat na polovinu nebo se vícenásobné zmenšování nepoužije.

Kvůli obecnosti návrhu rozhraní se i při dekompresi nezobrazují bloky rovnou, ale přes tyto matice, takže počet matic pro zdrojové bloky významně ovlivňuje rychlost dekomprese (na rozdíl od komprese, kde nehraje znatelnou roli).

4.9 Hledání optimálních zobrazení

Hledání optimálních zobrazení je nejdůležitější část celého kompresního procesu. Zde se nejvíce ovlivňuje kvalita výsledku a také stráví nejvíce výpočetního času.

ISquareEncoder

`ISquareEncoder` je rozhraní pro správu zobrazení mezi zdrojovými bloky generovanými modulem s rozhraním `ISquareDomains` a cílovými bloky vytvářenými modulem s rozhraním `ISquareRanges`.

Rozhraní obsahuje metody pro inicializaci před (de)kompresí *úkolu*, pro hledání optimálního zobrazení do daného cílového bloku, pro dokončení komprese a pro provádění dekódovacích akcí (vyčištění obrázku nebo iterování zobrazení). Dále jsou zde metody pro uložení a nahrání nastavení, pro zjištění počtu ukládacích fází a pro uložení nebo nahrání dat jednotlivých fází (význam fází je vysvětlen v 4.6).

MStdEncoder (soubory modules/stdEncoder.*)

Třída `MStdEncoder` implementuje rozhraní podle modelu upravených zobrazení popsaného v části 2.2.4 na straně 11 a využívá vzorců odvozených v kapitole 3.

Při hledání optimálního zdrojového bloku pro daný cílový blok (při kompresi) jsou nejprve předpočítány různé údaje z hodnot pixelů v bloku. Pak je postupně pro všechny zdrojové bloky předané predikčním modulem s rozhraním `IStdEncPredictor` (zvolen uživatelem, rozhraní popsáno v 4.10) počítána chyba zobrazení mezi dvojicí a je zapamatována nejlepší kombinace. Uživatel také může zvolit, zda bude zkoušeno osm způsobů otočení nebo jen identita, zda budou povoleny záporné lineární koeficienty, jak moc budou penalizovány vysoké lineární koeficienty (diskutováno v 3.4), zda budou započítány chyby vzniklé kvantizací, jak bude omezena absolutní hodnota lineárních koeficientů, jak nízká dosažená chyba bude stačit pro okamžité ukončení hledání, jak jemná bude kvantizace průměrů a směrodatných odchylek a jaké moduly se budou starat o ukládání kvantizovaných hodnot.

Moduly pro ukládání kvantizovaných průměrů a směrodatných odchylek cílových bloků splňují rozhraní `IIntCodec` (popsané v 4.11) a jsou zapojeny proto, že po sobě jdoucí hodnoty mají vysokou pravděpodobnost být sobě blízké. To zajišťuje například modul `MQuadTree` (popsán v 4.7), který řadí bloky podle jejich pořadí na Hilbertově křivce, což výrazně zvyšuje lokalitu posloupnosti. Tyto vlastnosti ukládaných posloupností lze využít různými způsoby ke zvýšení efektivity jejich uložení, a proto je tato práce oddělena do jiných modulů.

Celá chyba lze pro všechny konfigurace snadno spočítat ze součtu hodnot ve zdrojovém bloku, součtu jejich druhých mocnin, z předpočítaných údajů o cílovém bloku a ze součtu součinů hodnot odpovídajících pixelů zdrojového a cílového bloku. Tento součet součinů je jediný potřebný údaj, který nelze získat v konstantním čase. Ostatní hodnoty jsou spočteny snadno, nejčastěji pomocí struktur zmíněných v části 4.5.

Uživatel má mnoho možností pro změnu způsobu výpočtu chyby mezi dvojicí bloků. Zjišťovat konfiguraci a vyhodnocovat mnoho podmínek při každém výpočtu chyby by způsobilo výrazné zpomalení. Proto jsou možnosti realizovány jako šablonové parametry funkce počítající chybu, která je instanciována pro každou kombinaci parametrů. Na začátku hledání optimálního zdrojového bloku pro cílový blok se podle konfigurace zjistí adresa příslušné funkce, která je pak vždy volána. Každá z funkcí je tedy kompilátorem optimalizována zvlášť a již nevyhodnocuje žádné podmínky způsobené konfigurací. Velké množství instanciovaných funkcí zvyšuje kromě rychlosti také velikost binárního souboru, ale díky tomu, že funkce obsahují jen nej-

nutnější výpočty, je tento přínos pro celkovou velikost souboru nevýznamný.

Modul implementuje podporu pro progresivní dekódování založenou na ukládácích fázích zavedených v části 4.6. V první fázi jsou uloženy průměrné hodnoty všech bloků. Průměry zaberou málo místa, ale už z nich samotných lze zrekonstruovat použitelný náhled výsledku. V druhé fázi jsou uloženy směrodatné odchylky bloků. V třetí, poslední, fázi je uložen zbytek potřebných informací pro cílové bloky s nenulovou směrodatnou odchylkou — vždy bit pro znaménko lineárního koeficientu a tři bity kódující otočení (jen pokud jsou potřeba) a index zdrojového bloku.

4.10 Predikce vhodných zdrojových bloků

V části 4.9 je řešen přesný výpočet vhodnosti zobrazení mezi dvojicí bloků, ale provádět ho pro každou možnost by bylo příliš zdlouhavé. Je tedy nutné umět rychle odhadnout, které zdrojové bloky (a v jakých rotacích) je vhodné pro daný cílový blok zkoušet (možné přístupy byly rozebrány v 2.3.1).

IStdEncPredictor

Každý modul s rozhraním `IStdEncPredictor` slouží jako generátor prediktorů vhodných zdrojových bloků pro modul `MStdEncoder`. Definuje strukturu, která obsahuje všechny potřebné informace o daném cílovém bloku a nastavení komprese včetně mnoha předpočítaných údajů.

Po předání správně vyplněné instance této struktury do rozhraní je vytvořena instance třídy s rozhraním `IStdEncPredictor::IOneRangePredictor`, která postupně vrací vhodné zdrojové bloky (pro tento účel jsou různé rotace bloku považovány za různé zdrojové bloky). Bloky nejsou vraceny po jednom, ale v seznamech, aby se tak snížila režie spojená s voláním virtuální metody. Je také předávána maximální akceptovaná čtvercová chyba, což ulehčuje prediktoru rozhodování, které bloky je bezpečné vynechat.

Dále je definována metoda volaná po skončení všech predikčních požadavků. Je určena k uvolnění potenciální paměti alokované pro urychlení predikcí.

MNoPredictor (soubor modules/noPredictor.h)

Modul `MNoPredictor` je zcela triviální implementací rozhraní, která nedělá žádnou predikci a vrací všechny možnosti. Slouží hlavně pro vyhodnocení účinnosti ostatních implementací.

MSaupePredictor (soubory modules/saupePredictor.*)

Modul `MSaupePredictor` je založen na větě publikované v [11]. Zde je uvedena v mírně upravené podobě, která je bližší značení z částí 2.2 a 3.

Věta 4.1. *Nechť* $n \geq 2$ *a* $X = \mathbb{R}^n \setminus \{r \cdot (1, 1, \ldots, 1) \,|r \in \mathbb{R}\}$. *Definujme normalizační funkci* $\Phi : X \to X$ *a funkci* $D : X \times X \to \langle 0, \sqrt{2} \rangle$ *následovně:*

$$(\Phi(x))_i = \frac{x_i - \frac{1}{n}\sum_{j=1}^{n} x_j}{\sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(x_j - \frac{1}{n}\sum_{k=1}^{n} x_k\right)^2}} \quad a$$

$$D(r, d) = \min\{\ \|\Phi(r) - \Phi(d)\|,\ \|-\Phi(r) - \Phi(d)\|\ \},$$

kde $\|.\|$ *značí eukleidovskou normu.*

Pak pro nejmenší čtvercovou chybu definovanou

$$E(r, d) = \min_{p,q \in \mathbb{R}} \sum_{i=1}^{n} (p\, d_i + q - r_i)^2 \quad \textit{platí}$$

$$E(r, d) = \left(\sum_{i=1}^{n} (\Phi(r))_i\, r_i\right) g(D(r, d)), \quad \textit{kde } g(x) = x^2\left(1 - \frac{x^2}{4}\right).$$

Protože je součet $\sum_{i=1}^{n} (\Phi(r))_i\, r_i$ pro daný cílový blok konstantní a funkce $g(x)$ je rostoucí na oboru hodnot funkce $D(r, d)$, lze převést hledání optimálního zdrojového bloku na dvojici problémů hledání nejbližšího bodu v pevně dané množině.

Pro každou velikost cílového bloku je tedy postavena datová struktura obsahující všechny možné zdrojové bloky normalizované funkcí Φ, konkrétně je použita varianta statického KD-stromu popsaná na straně 43. K danému bodu jsou vraceny prvky struktury v pořadí podle dolních odhadů na jejich vzdálenost od tohoto bodu. Struktury jsou stavěny vždy až když je potřeba je prohledávat. Poté jsou uloženy v modulu a sdíleny všemi prediktory vracenými modulem (struktury jsou statické).

Omezení prohledávání Pokud zvolíme limit m pro čtvercovou chybu, pak maximální přípustná čtvercová chyba v normalizovaném prostoru je rovna

$$2 - 2\sqrt{1 - \frac{m}{\sum_{i=1}^{n} (\Phi(r))_i r_i}} \quad \text{(pokud existuje)}.$$

Tento limit sám nemusí dostatečně zkrátit prohledávání prostoru, protože téměř konstantní cílové bloky lze dobře aproximovat už konstantním blokem a aproximace nekonstantním blokem není nikdy horší. V takovém případě vyjde pod odmocninou záporné číslo. Proto má uživatel možnost v modulu nastavit, jaká největší část zdrojových bloků může být predikována.

Vícenásobné prohledávání Z věty 4.1 je zřejmé, že je potřeba vždy pro dvojici bloků r^i a d^j uvažovat dvě možnosti $\|\Phi(r^i) - \Phi(d^j)\|$ a $\|-\Phi(r^i) - \Phi(d^j)\|$ (pokud jsou povolena zobrazení se zápornými lineárními koeficienty). Jedna možnost řešení je, že struktura obsahuje vždy $\Phi(d^j)$ i $-\Phi(d^j)$ a je hledán nejbližší bod k $\Phi(r^i)$. V druhé možnosti obsahuje struktura pouze $\Phi(d^j)$, ale je v ní hledáno okolí $\Phi(r^i)$ a $-\Phi(r^i)$. Analogicky lze dvěma způsoby řešit uvažování osmi vzájemných rotací (pokud jsou povoleny uživatelem) — buď je struktura postavena z osminásobného množství otočených bloků, nebo je nutné ji osminásobně prohledávat. Kvůli úspoře paměti byla zde v obou případech zvolena varianta s menší strukturou a násobným prohledáváním.

Vícenásobné prohledávání je realizováno tak, že jsou stavy všech prohledávání udržovány v haldě (použita je generická STL implementace binární haldy) uspořádané podle příštího odhadu vzdálenosti ve struktuře a prohledává se tedy vždy ta varianta, která je nejblíže k výchozímu bodu hledání (podle odhadu struktury).

Zmenšování bloků Pro účely predikce jsou bloky zmenšeny na velikost 4×4, stejně jako v [11]. Už ze zmenšenin lze vzájemnou zobrazitelnost dobře odhadnout — RMSE optimálního zobrazení mezi zmenšeninami je dolním odhadem na RMSE optimálního zobrazení v původním rozlišení a rovnost nastává, pokud v rámci zprůměrovaných skupin pixelů byly „stejné vzájemné odchylky“ ve zdrojovém i cílovém bloku. Díky spojitému charakteru běžných obrázků bývá tento odhad dostatečně blízko, takže už podle něj lze většinu kandidátů bezpečně vyřadit (protože odhad překročí maximální povolenou chybu).

Navíc práce s velkými zdrojovými bloky by přinášela řadu problémů. Jedním z důvodů je, že efektivita KD-stromů klesá s růstem počtu souřadnic. Dalším důvodem je paměťová náročnost. Pokud bychom pro účel stavby stromu uložili obrazy funkce Φ, potřebujeme už při velikosti 16×16 pro každý blok 1 KB paměti. Počty zdrojových bloků se běžně pohybují v řádu stovek tisíc, což dává neúnosnou spotřebu stovek megabytů paměti. Alternativně by bylo možné uchovávat v paměti pouze koeficienty funkce Φ (pro pevný blok je to afinní funkce) a počítat hodnotu na každé souřadnici normalizovaného bloku z původního obrázku jednotlivě při každém přístupu. To by ale vedlo k mnohonásobnému zpomalení.

Pro predikci by bylo možné použít i jiné způsoby snižování počtu dimenzí, například výběr několika nejdůležitějších koeficientů dvourozměrné diskrétní kosinové transformace bloku — z použití transformace ve standardu JPEG je zřejmé, že je to velmi dobrý způsob aproximace obrázku malým množstvím souřadnic.

KD-stromy (soubor kdTree.h)

KD-strom je binární strom, který rekurzivně dělí spravovaný prostor v každém svém vrcholu podle hodnoty jedné ze souřadnic. Tato struktura byla zvolena především pro svou jednoduchost a možnost paměťově úsporné implementace.

Reprezentace stromu Protože je strom statický a binární, lze jeho tvar volit stejně jako pro binární haldy a ukládat jeho vrcholy v poli. Je zvolena varianta, kde jsou všechny obsažené body až v listech. Pro každý vnitřní vrchol pak stačí uložit jen souřadnici, podle které je dělen podprostor vrcholu, a hodnotu, na které je hranice dělení. Pro listy je pak uložen jen index bodu, kterému odpovídá.

V klasických KD-stromech by byly uchovány také souřadnice všech obsažených bodů. Ty by ale ovlivnily prohledávání až po dělení vrcholů nad listy, kdy už je dolní odhad vzdálenosti blízko skutečnosti. Zde navíc není nutné znát vzdálenosti přesně — většinou se pracuje se zmenšeninami a přesné porovnání je stejně poté provedeno jiným modulem. Souřadnice všech bodů také zabírají mnoho místa, takže bylo zvoleno, že budou využity pouze pro postavení stromu a pak zahozeny. Tato volba pravděpodobně zhorší výkon struktury, ale alternativa zde není implementována, takže míra jejího vlivu není známa (je to možnost rozšíření v budoucnu).

Části prostoru patřící jednotlivým vrcholům jsou definovány rekurzivně. Kořen reprezentuje hyperkvádr uložený ve stromě, který těsně obaluje všechny body ve struktuře (minima a maxima všech bodů v každé ze souřadnic ve struktuře). Synové každého vrcholu si rozdělí jeho hyperkvádr podle nadroviny kolmé na jednu ze souřadnic (souřadnice a mezní hodnota je ve vrcholu uložena). V této implementaci patří body ležící na mezní nadrovině do obou podstromů, aby bylo možné vždy volit libovolné velikosti podstromů a dosáhnout tak tvaru odpovídajícímu binární haldě.

Prohledávání stromu Hledání nejbližších bodů staví postupu typickém pro všechny podobné struktury. Je udržována halda z vrcholů stromu uspořádaná podle vzdáleností podprostorů vrcholů stromu od výchozího bodu (toho, k němuž hledáme nejbližší body). Na začátku je v haldě jen kořen a pak se vždy odebere z haldy minimum a vloží se do ní synové obsaženého vrcholu stromu. Pokud je ale v minimu haldy list stromu, je pouze poslán na výstup jako další z nejbližších bodů.

Aby byla práce s haldou efektivní, jsou v každém prvku kromě čísla vrcholu stromu uloženy souřadnice bodu z části prostoru patřící vrcholu, který

je nejblíže výchozímu bodu, a také druhá mocnina jeho eukleidovské vzdálenosti od výchozího bodu (podle ní je halda uspořádaná). Umocnění vzdálenosti nepokazí uspořádání, protože je to pro nezáporná čísla rostoucí funkce, a významně zjednoduší výpočty. Tyto údaje lze při dělení na syny velmi snadno přepočítat, protože nejbližší bod se může změnit jen v jedné souřadnici a druhá mocnina jeho vzdálenosti se tedy změní jen o rozdíl druhých mocnin rozdílů hodnoty výchozího bodu v té souřadnici od nové a původní hodnoty nejbližšího bodu.

Při prohledávání lze navíc nastavit, že souřadnice výchozího bodu, které obsahují speciální hodnotu NaN (*not a number*), budou ignorovány. To je modulem `MSaupePredictor` využito pro lepší predikci pro ořízlé cílové bloky, které jsou menší, protože leží na okraji obrázku.

Stavba stromu Strom je stavěn rekurzivně — při stavbě vrcholu v je dán interval b v poli p obsahujícím indexy bodů, které mají být reprezentovány podstromem vrcholu v. Nejprve je určeno, jaké počty bodů budou patřit synům vrcholu v. „Haldový tvar“ je určen jednoznačně počtem listů, který zde odpovídá počtu bodů ve struktuře, a navíc se tento tvar dědí — pokud má podstrom nějakého vrcholu tvar haldy, pak ho mají i podstromy jeho synů a tedy i všechny ostatní podstromy.

Pak je určena souřadnice i, podle které bude část prostoru patřící v dělena (metody rozebrány níže). Podle ní se obsah pole p v intervalu b rozdělí na dvě části tak, aby jejich velikosti odpovídaly počtům bodů, které budou patřit synům, a aby žádný bod levého syna neměl v i-té souřadnici hodnotu větší než nějaký bod pravého syna. To je provedeno pomocí generického algoritmu `nth_element` obsaženého v STL, který najde k-tý nejmenší prvek a přeuspořádá ostatní prvky korektně vzhledem k jeho pozici. Pak se uloží hodnota souřadnice i na vzniklé hranici do vrcholu v a rekurzivně se zkonstruují jeho synové (pokud na ně zbývá více než jeden bod).

Při stavbě je možno zvolit jeden z několika implementovaných způsobů volby souřadnice i. Jsou zde dva triviální algoritmy — jeden cyklicky střídá souřadnice podle houbky a druhý je volí náhodně. Modul `MSaupePredictor` používá lepší algoritmus, který volí tu souřadnici, ve které má část prostoru patřící vrcholu v největší rozpětí.

Korespondence se zdrojovým kódem Struktura stromu je reprezentována generickou třídou `KDTree`. Prohledávání provádí její podtřída `PointHeap`, která při konstrukci požaduje zadání výchozího bodu, instance prohledávaného stromu a volbu zda se mají ignorovat některé souřadnice. Pak na požadavek vrací indexy jednotlivých bodů nebo momentální odhad vzdále-

nosti od výchozího bodu.

Vytváření stromu provádí statická (ve smyslu objektové terminologie) metoda `makeTree` oddělené třídy `KDBuilder`. Ta umožňuje také zvolit jednu z metod třídy `KDBuilder`, která bude použita pro volbu souřadnice i.

4.11 Kódování celočíselných řad

IIntCodec

`IIntCodec` je rozhraní pro kódování posloupností celých čísel z rozsahu $\{0, 1, \ldots, n-1\}$ do proudu bytů. V této práci je využito modulem `MStdEncoder` pro ukládání průměrných barev a směrodatných odchylek cílových bloků.

Jsou v něm poskytnuty metody pro nastavení počtu možností n, pro uložení seznamu čísel do proudu bytů a pro zpětné zrekonstruování seznamu dané délky z proudu. Dále jsou zde metody pro uložení a nahrání nastavení modulu.

MDifferentialVLICodec (soubory modules/vliCodec.*)

Modul `MDifferentialVLICodec` je kodek určený pro efektivní ukládání posloupností čísel, kde mají po sobě jdoucí prvky s vysokou pravděpodobností blízké hodnoty. Místo samotných prvků posloupnosti se uvažuje vždy rozdíl hodnoty od předchozího prvku z rozsahu $\{-\lceil \frac{n}{2} \rceil, \ldots, \lfloor \frac{n}{2} \rfloor\}$ počítaný modulo n. Tyto rozdíly jsou pak jednotlivě kódovány pomocnou třídou `VLI`.

Třída `VLI` provádí kódování tak, že pro čísla blízká nule použije menší množství bitů. Nejprve převede všechna čísla do rozsahu $\{0, 1, \ldots, n-1\}$ podle vzorce

$$f(x) = \begin{cases} 2x & (x \geq 0) \\ -2x - 1 & (x < 0) \end{cases}.$$

Pak je celý interval rozdělen na několik částí kódovaných různými počty bitů. Délka první části je nastavena uživatelem (je to mocnina čísla 2) a každá další část má dvojnásobnou délku, dokud se nevyčerpají všechna čísla z intervalu. Každé číslo je pak reprezentováno několika bity udávajícími do jaké části patří a pak dalším blokem bitů (jejich počet závisí na velikosti dané části), které udávají pozici v části.

Kapitola 5

Testování a vyhodnocení

V této kapitole budou ukázány výsledky testování komprese na vybraných černobílých obrázcích.

Testy jsou omezeny pouze na odstíny šedi kvůli jednoduššímu vyhodnocování. Publikované výsledky se většinou také zabývají pouze černobílými obrázky a navíc by bylo obtížné sladit míru komprese všech barevných složek na podobnou úroveň, aby bylo možné výsledky rozumně porovnat.

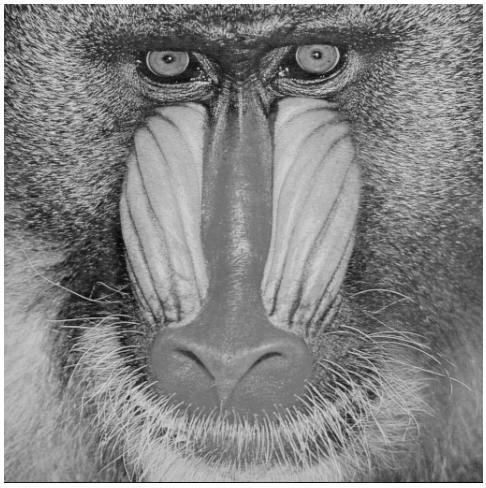
Použito je celkem šest testovacích obrázků získaných z různorodé kolekce jedné výzkumné skupiny univerzity ve Waterloo [20]. Obrázky *Lena*, *Mandrill*, *Peppers* a *Goldhill* jsou ve fraktální kompresi často používané a mají klasické rozměry 512×512 pixelů. Obrázky *Serrano* a *Clegg* jsou velmi odlišné — nejsou to fotografie (spíše se dají popsat spojením „digitální umění“) a mají i neobvyklé rozměry.

Ke kompresi pomocí standardů JPEG a JPEG 2000 byly použity utility projektu *ImageMagick* [17]. Všechny uvedené doby fraktální komprese, hodnoty PSNR a kompresní poměry byly naměřeny samotným programem.[1] Doby fraktální i standardní komprese byly získány jednovláknovým spouštěním v dávkovém režimu (bez GUI) na počítači s procesorem o frekvenci 2 GHz, systémem Gentoo Linux a kompilátorem GCC verze 4.2.1.

Následují tři skupiny testů zaměřené na vyhodnocení konkrétních částí komprese a jedna zkoumající celkový výsledek. Parametry modulů, které nejsou uvedeny, jsou vždy nechávány na přednastavených hodnotách.

[1] Kompresním poměrem se zde rozumí poměr nekomprimované velikosti ke komprimované, přičemž velikost „nekomprimovaného obrázku“ v bytech je dána jako počet jeho pixelů (obrázky jsou černobílé).

Obrázek 5.1: Testovací obrázky *Lena* a *Mandrill* (velikost 512×512 pixelů)



Obrázek 5.2: Testovací obrázky *Peppers* a *Goldhill* (velikost 512×512 pixelů)

Obrázek 5.3: Testovací obrázek *Serrano* (velikost 629 × 794 pixelů)

Obrázek 5.4: Testovací obrázek *Clegg* (velikost 814 × 880 pixelů)

5.1 Účinnost penalizace

V části 3.4 byla navržena alternativa k omezení absolutní hodnoty lineárních koficientů barevných zobrazení mezi bloky a bylo zdůvodněno, proč by to mělo být výhodnější. Zde bude porovnán vliv penalizace a omezení koeficientů na kvalitu a kompresní poměr. Aby bylo hodnocení vlivu objektivnější, je pro predikci zvolen modul `MNoPredictor` a je zrušen limit pro předčasné ukončení hledání, tedy jsou vždy zkoušeny všechny zdrojové bloky.

Výsledky testů jsou v tabulce 5.1 na následující straně. Pro šest obrázků a několik nastavení míry penalizace nebo omezení koeficientů je změřen kompresní poměr a hodnota PSNR. Absolutní hodnoty lineárních koeficientů barevných zobrazení jsou omezovány (v modulu `MStdEncoder`) tak, že kombinace bloků, kde by byla hodnota příliš vysoká, nejsou uvažovány.

Výrazně nejhorší výsledky vyšly pro nastavení, kde není použita ani jedna z metod. Omezení koeficientů pod 1.0 nutné pro zaručení konvergence dekódování sice dává dobré výsledky pro některé obrázky (*Lena* a *Goldhill*), ale v ostatních případech je kvalita hluboko pod nejlepším výsledkem. Proto je často omezení uvolněno a jsou dovoleny i hodnoty vyšší (zkoumáno například v kapitole 3 Fisherovy knihy [5]). To vede k výraznému zlepšení a kvalita většiny obrázků se přiblíží k nejlepším hodnotám dosaženým v testu. Přesto u obrázku *Clegg* je PSNR stále více než 0.9 dB pod optimem.

Při použití penalizace místo omezení lineárních koeficientů vycházejí nepatrně nižší kopresní poměry, ale kvalita se pohybuje blízko nejvyšší hodnoty u všech testovaných obrázků. V implicitním nastavení proto nejsou omezeny lineární koeficenty zobrazení mezi bloky a je použita penalizace s koeficientem 0.25 (kompromis mezi kvalitou a kompresním poměrem).

5.2 Účinnost predikce

Zkoušení všech zdrojových bloků pro každý cílový je neúnosně zdlouhavý proces. Proto je podstatná efektivní predikce. V této části je zkoumána účinnost predikce modulem popsaným v 4.10 na straně 40.

Tabulka 5.2 na straně 51 ukazuje pro různá množství zdrojových bloků dosaženou kvalitu, kompresní poměr a spotřebovaný čas. Hodnoty jsou naměřeny vždy pro různá nastavení limitu na predikovanou část ze všech bloků včetně varianty s úplným prohledáváním místo predikce.

Z výsledků je především vidět, že při malém množství zdrojových bloků je implementovaná predikce velmi nevýhodná — vede k výrazně horší kvalitě i horšímu kompresnímu poměru při relativně malém množství ušetřeného času. To je nejspíše způsobeno vysokým počtem dimenzí prostoru indexo-

Tabulka 5.1: Testování účinností penalizace a omezení koeficientů

Obrázek	Míra penal.	Omez. koef.	Komp. poměr	PSNR (dB)	Obrázek	Míra penal.	Omez. koef.	Komp. poměr	PSNR (dB)
Peppers	2.00	-	10.23	37.22	Lena	2.00	-	9.61	36.50
	1.00	-	10.72	**37.25**		1.00	-	9.91	36.60
	0.50	-	11.12	**37.27**		0.50	-	10.08	**36.66**
	0.25	-	11.31	**37.25**		0.25	-	10.19	**36.68**
	0.15	-	11.43	37.20		0.15	-	10.23	**36.68**
	0.05	-	11.55	36.74		0.05	-	10.30	36.55
	-	-	11.60	35.34		-	-	10.35	35.44
	-	1.0	11.11	30.61		-	1.0	10.16	36.41
	-	1.2	11.22	35.61		-	1.2	10.21	36.49
	-	2.0	11.56	36.85		-	2.0	10.33	36.37
Goldhill	2.00	-	6.32	34.93	Serrano	2.00	-	8.41	35.50
	1.00	-	6.44	35.12		1.00	-	8.59	35.88
	0.50	-	6.52	35.26		0.50	-	8.71	36.00
	0.25	-	6.59	**35.32**		0.25	-	8.79	**36.11**
	0.15	-	6.63	**35.32**		0.15	-	8.84	35.90
	0.05	-	6.65	35.20		0.05	-	8.71	35.80
	-	-	6.67	34.12		-	-	8.93	22.32
	-	1.0	6.61	34.92		-	1.0	8.64	28.42
	-	1.2	6.65	35.06		-	1.2	8.72	34.59
	-	2.0	6.67	35.00		-	2.0	8.85	35.63
Mandrill	2.00	-	4.42	28.94	Clegg	2.00	-	4.93	32.38
	1.00	-	4.43	29.18		1.00	-	4.97	32.78
	0.50	-	4.45	29.33		0.50	-	5.00	32.93
	0.25	-	4.46	**29.42**		0.25	-	5.06	**32.98**
	0.15	-	4.46	**29.46**		0.15	-	5.07	32.92
	0.05	-	4.47	**29.45**		0.05	-	5.08	32.85
	-	-	4.47	28.89		-	-	5.09	32.32
	-	1.0	4.48	26.63		-	1.0	5.02	23.88
	-	1.2	4.48	28.07		-	1.2	5.03	30.34
	-	2.0	4.47	29.32		-	2.0	5.08	32.07

Tabulka 5.2: Testování účinnosti predikce

Obrázek	Zdrojové bloky	Predikovaná část	PSNR (dB)	Kompresní poměr	Čas (s)
Lena	2^{11}	2^{-10}	26.78	8.85	1.2
		2^{-8}	26.78	8.85	1.2
		2^{-6}	30.10	9.02	1.2
		vše	34.26	11.15	7.7
	2^{15}	2^{-10}	32.04	7.87	1.3
		2^{-8}	34.17	8.34	1.5
		2^{-6}	35.03	8.82	2.1
		vše	35.87	10.66	123
	2^{19}	2^{-10}	35.78	8.15	2.7
		2^{-8}	36.15	8.41	5.2
		2^{-6}	36.36	8.66	14.2
		vše	36.66	10.19	1808
Mandrill	2^{11}	2^{-10}	20.49	5.53	1.8
		2^{-8}	20.49	5.53	1.8
		2^{-6}	22.30	5.53	1.9
		vše	25.60	5.67	13.6
	2^{15}	2^{-10}	24.15	4.74	2.0
		2^{-8}	25.77	4.74	2.4
		2^{-6}	26.86	4.76	3.9
		vše	27.77	4.98	235
	2^{19}	2^{-10}	28.33	4.20	4.8
		2^{-8}	28.98	4.23	10.7
		2^{-6}	29.31	4.24	32.5
		vše	29.42	4.46	3593
Serrano	2^{11}	2^{-10}	23.16	8.09	2.2
		2^{-8}	23.16	8.09	2.2
		2^{-6}	25.87	7.98	2.3
		vše	31.62	9.27	10.8
	2^{15}	2^{-10}	29.90	7.17	2.3
		2^{-8}	32.17	7.37	2.6
		2^{-6}	33.59	7.64	3.3
		vše	34.39	8.97	149
	2^{19}	2^{-10}	33.64	7.07	3.3
		2^{-8}	35.20	7.33	5.1
		2^{-6}	35.74	7.57	11.5
		vše	36.05	8.79	1233

vaného KD-stromem. Množství zdrojových bloků je uvažováno i s rotacemi, takže limit 2^{11} odpovídá 2^8 bodům v KD-stromu. Protože body ve stromě jsou 16-prvkové vektory, jsou podprostory patřící listům omezeny nejvýše v polovině ze všech souřadnic, což je velmi málo. Zde by nejspíše pomohla volba varianty stromu, kde listy obsahují přesné souřadnice bodu. To by vedlo k přesnějším odhadům výměnou za větší časovou a paměťovou náročnost.

Pro účinnou kompresi jsou ale vhodnější větší počty zdrojových bloků a zde je již predikce účinnější. Při 2^{19} zdrojových blocích a predikování jejich části o relativní velikosti 2^{-8} nebo 2^{-6} jsou už poklesy PSNR okolo 0.5 dB, ale u některých obrázků znatelně klesá i kompresní poměr.

Zajímavé je porovnání s výsledky z článku publikujícím základní použitou větu. [11] Zde jsou také bloky pro účely predikce zmenšeny na velikost 4×4, ale všechna čísla jsou kvantizována na 8 bitů a místo KD-stromů je použita mnohem složitější strutura navržená pro přibližné hledání nejbližších sousedů. Další drobnou odlišností je, že v článku jsou obě metody prohledávání prováděny pouze v rámci klasifikační třídy (použit je kód od Fishera [5]), ale to by na porovnání nemělo mít větší vliv. Výsledky na identických obrázcích *Lena* a *Mandrill* pro různá množství zdrojových bloků zde ukazují pokles PSNR pod 0.5 dB při zanedbatelném poklesu kompresního poměru, přestože je predikováno vždy pouze 5×16 bloků (je uvažováno 8 možných otočení a obě možnosti znaménka lineárního koeficientu barevného zobrazení).[2] Nejpravděpodobnějším důvodem vyšší účinnosti predikce z článku [11] je použití jiné datové struktury a tedy KD-stromy pro tento problém zřejmě nebudou vhodné (alespoň v uvedené implementaci bez informací o souřadnicích bodů v listech).

Přesto se celkové výsledky v implicitním nastavení pohybují okolo nejlepších hodnot dosahovaných Fisherovým programem (v klasické variantě používající čtyřstrom a klasifikaci) a jeho zrychlenou verzí (data z [5, 11]), jak ukazuje tabulka 5.3 na následující straně. Pro obrázek *Mandrill* je dokonce kvalita výrazně vyšší.

5.3 Účinnost diferenčního kódování

Modul `MStdEncoder` (4.9) sloužící pro správu zobrazení mezi bloky ukládá kvantizované průměrné hodnoty a směrodatné odchylky bloků pomocí kodeků, které spotřebovávají méně místa pro posloupnosti se vzájemně blízkými po sobě jdoucími prvky (4.11). Protože kódování ovlivňuje pouze způsob uložení a nemá vliv na přesnost hodnot, stačí testovat pouze změnu

[2] obrázek zde nazývaný *Mandrill* je v článku [11] nazýván *Baboon*

Tabulka 5.3: Porovnání s publikovanými výsledky [5, 11]

Obrázek	Program	Komp. poměr	PSNR (dB)	Čas[a] (s)
Lena	Fisher	8.43	36.21	460
	Saupe	8.31	35.82	57
	Zde předvedený	8.41	36.15	*5.2*
Mandrill	Fisher	4.17	27.13	857
	Saupe	4.16	26.69	90
	Zde předvedený	4.23	28.98	*10.7*

[a]publikované doby komprese nejsou porovnatelné, protože byly získány na o mnoho let starším hardwaru (procesor MIPS R4000)

kompresního poměru (vliv na spotřebovaný čas je zanedbatelný). V této části bude zjištěno, kolik prostoru technika skutečně ušetří.

Prostor zabraný klasickou metodou byl změřen tak, že byly diferenčním modulům nastaveny počty symbolů v první části na jejich celkový počet, což vede k tomu, že se na každou hodnotu spotřebuje pevný počet bitů, právě dvojkový logaritmus z počtu symbolů (to je pro `MStdEncoder` vždy celé číslo).

Výsledky pro tři obrázky a různá nastavení kvality ukazuje tabulka 5.4 na následující straně, kde diferenční kódování vždy přináší úsporu prostoru, obvykle okolo 5%. Jsou zde také vidět dva trendy — větší účinek při vyšší kvalitě a také při menším počtu zdrojových bloků. Zvýšení nastavené kvality vede k rozdělení na menší cílové bloky, což zvyšuje korelaci jejich vlastností a tím i efektivitu metody. Snížení počtu zdrojových bloků také vede k rozdělení na více cílových bloků, protože je tím snížena pravděpodobnost nalezení vhodného zobrazení, a navíc ještě zvyšuje podíl výsledného prostoru ovlivněného metodou, protože identifikátory zdrojových bloků zaberou méně místa.

5.4 Celková účinnost

Během předchozích testů bylo doladěno implicitní nastavení některých parametrů komprese. V této části je vyhodnocována celková efektivita komprese a porovnána se standardními metodami JPEG a JPEG 2000.

Fraktální komprese je testována pro několik vybraných nastavení počtu zdrojových bloků a jejich maximální predikované části (značeny z a p), které představují různé poměry účinnosti a spotřebovaného času. Implicitní nastavení je $z = 2^{19}$ a $p = 2^{-8}$.

Tabulka 5.4: Testování účinnosti diferenčního kódování

Obrázek	Nastav. kvality	Komp. poměr pro 2^{19} zdroj. bl. dif. kód.	klas. kód.	podíl	Komp. poměr pro 2^{15} zdroj. bl. dif. kód.	klas. kód.	podíl
Lena	40	37.87	36.55	1.04	30.72	29.49	1.04
	70	16.52	15.72	1.05	14.27	13.43	1.06
	90	8.41	7.79	1.08	8.34	7.61	1.10
	95	6.46	5.91	1.09	6.79	6.10	1.11
Mandrill	40	9.09	8.69	1.05	9.08	8.65	1.05
	70	5.40	5.12	1.05	5.80	5.46	1.06
	90	4.23	3.96	1.07	4.74	4.40	1.08
	95	4.02	3.75	1.07	4.60	4.25	1.08
Serrano	40	16.37	16.14	1.01	13.92	13.63	1.02
	70	9.94	9.67	1.03	9.32	8.93	1.04
	90	7.33	6.97	1.05	7.37	6.90	1.07
	95	6.59	6.18	1.07	6.86	6.36	1.08

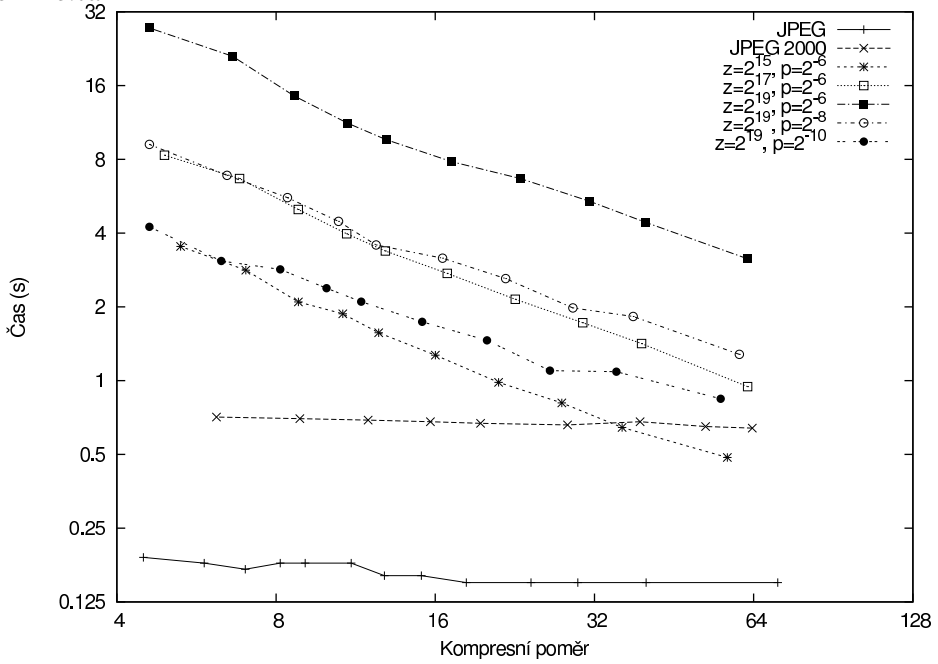
Obrázek 5.5 na následující straně ukazuje závislost spotřebovaného času na kompresním poměru pro obrázek *Lena*. Graf potvrzuje fakt, že fraktální komprese spotřebuje při použitelné kvalitě řádově více času než standardní metody. U těch se spotřebovaný čas s kompresním poměrem výrazněji nemění, ale fraktální komprese spotřebuje při nízkém nastavení kvality času méně. Grafy závislosti času na kompresním poměru pro ostatních pět testovaných obrázků vypadají stejně, a proto zde nejsou uvedeny.

Zajímavější je závislost kvality na kompresním poměru. Typickým grafem je 5.6 pro obrázek *Lenna*, kde mají všechny křivky podobný průběh a fraktální komprese nedosahuje účinnosti standardních metod. Stejný vzhled mají i neuvedené grafy pro obrázky *Goldhill* a *Peppers*. Podobný je také graf pro obrázek *Mandrill* (5.7), akorát mají křivky jiné tvary a při použitelných hodnotách kvality se fraktální komprese více přibližuje JPEG kompresi.

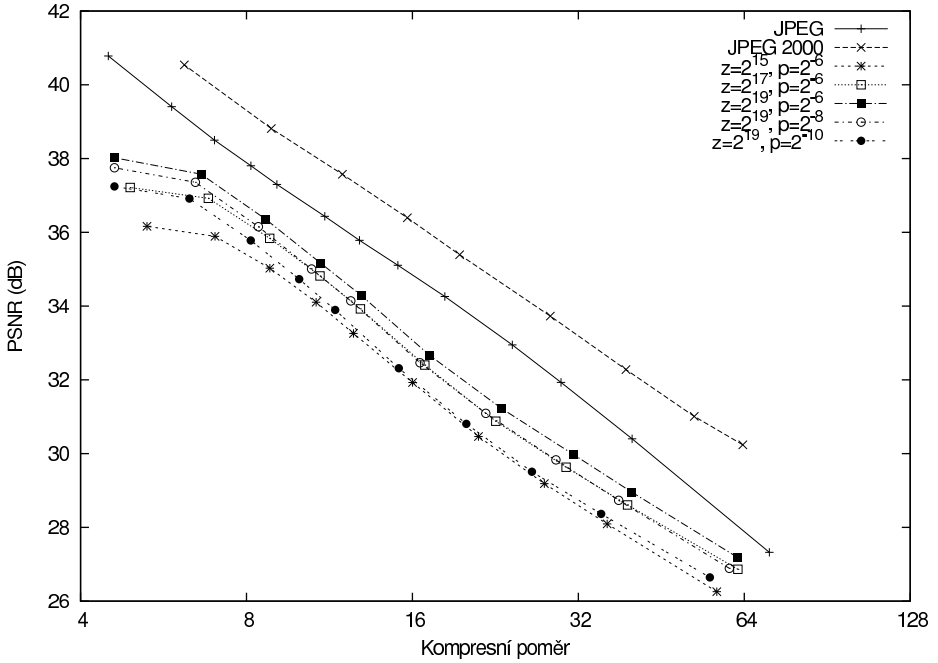
Jiná situace je u obrázku *Serrano* (5.8). Zde dává fraktální komprese výrazně lepší výsledky než JPEG a v některých případech dokonce i srovnatelné s JPEG 2000. Podobně vypadá i graf pro obrázek *Clegg* (5.9), ale zde již nedává fraktální komprese tak dobré výsledky.

Testy ukazují, že spíše než na běžné spojité fotografie je zde implementovaná fraktální komprese vhodnější pro obrázky obsahující ostré hrany, jako zkoušené umělecké obrázky.

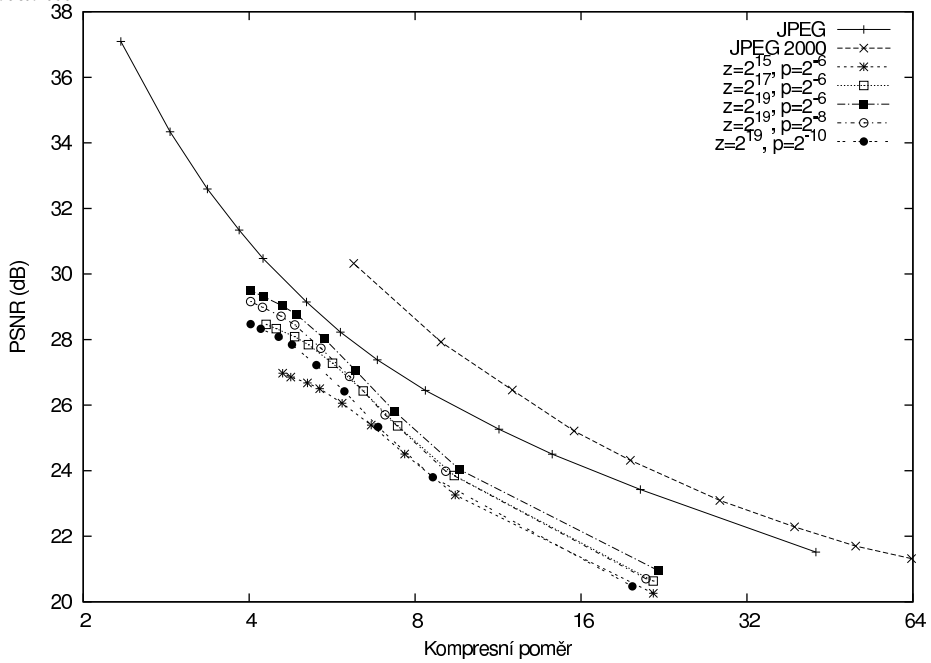
Obrázek 5.5: Graf závislosti doby komprese na kompresním poměru pro obrázek *Lena*



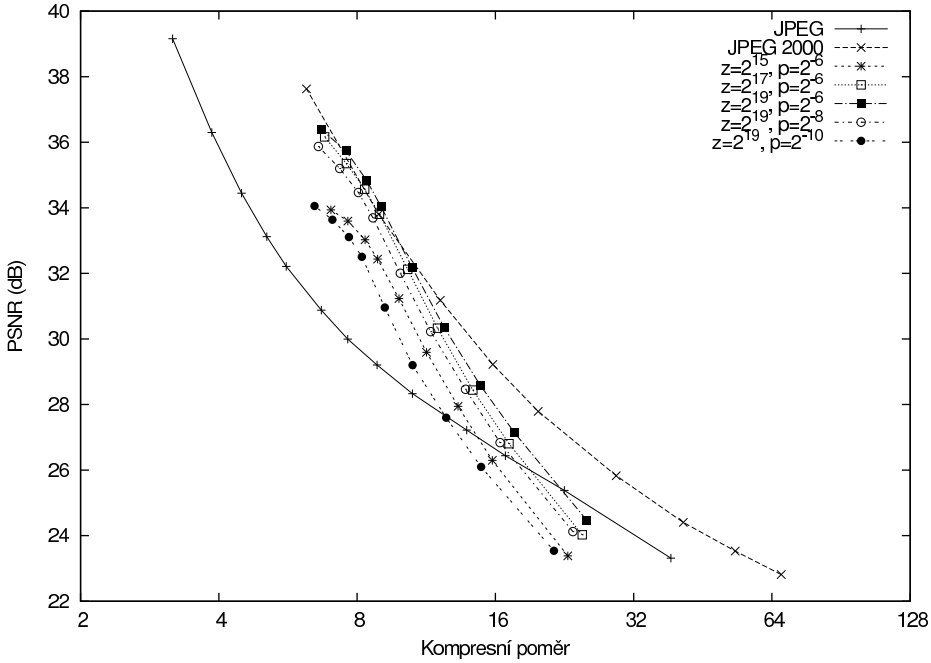
Obrázek 5.6: Graf závislosti kvality na kompresním poměru pro obrázek *Lena*



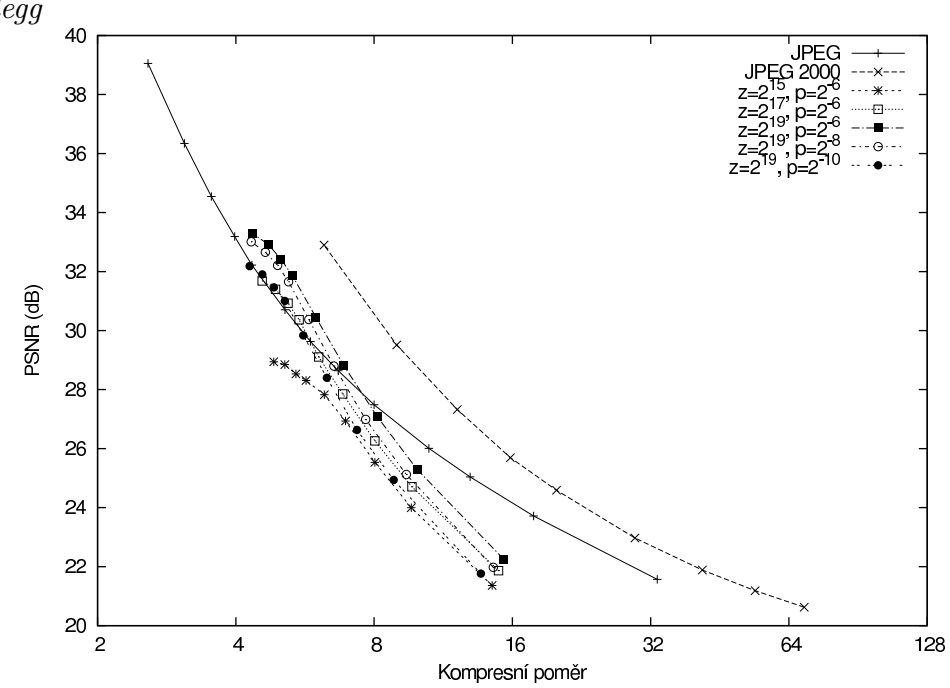
Obrázek 5.7: Graf závislosti kvality na kompresním poměru pro obrázek *Mandrill*



Obrázek 5.8: Graf závislosti kvality na kompresním poměru pro obrázek *Serrano*



Obrázek 5.9: Graf závislosti kvality na kompresním poměru pro obrázek *Clegg*



Kapitola 6

Závěr

V této práci byl představen modulární systém pro fraktální kompresi obrazu umožňující efektivní implementaci vyměnitelných algoritmů s minimální zátěží technického kódu. Pro tento systém byla implementována sada modulů provádějící většinou klasická řešení jednotlivých části fraktální komprese. Spolu s grafickým a dávkovým uživatelským rozhraním vznikl funkční celek vhodný pro výzkum vlastností komprese. Kromě použití známých metod byla navržena drobná vylepšení. Jejich vliv a také celkový výsledek byly experimentálně vyhodnoceny a porovnány se standardními metodami komprese.

Implementace byla prováděna s ohledem na možnost snadného doplnění dalších algoritmů a jejich kombinování. Mnoho možností dalšího rozšíření bylo zmíněno v části 2.3. Mezi nejzajímavější patří horizontálně-vertikální dělení cílových bloků a možnost aproximace cílových bloků pomocí většího počtu zdrojových bloků. Dalším vhodným rozšířením implementace by v budoucnosti mohlo být vylepšení nebo nahrazení stávajícího predikčního modulu, jehož účinnost podle provedených testů není ideální.

Reference

[1] Andrea F. Abate, Michele Nappi, and Daniel Riccio. Embedding quality measures in pifs fractal coding. In Mohamed S. Kamel and Aurélio C. Campilho, editors, *ICIAR*, volume 4633 of *Lecture Notes in Computer Science*, pages 784–793. Springer, 2007.

[2] B. Aoued. Accelerating fractal image compression by domain pool reduction adaptive partitioning and structural block classification. *Computers and Communications, IEEE Symposium on*, 2:585–589, 2004.

[3] Michael Barnsley. *Fractals everywhere*. Academic Press Professional, Inc., San Diego, CA, USA, 1988.

[4] Kin-Wah Ching Eugene and Ghim-Hwee Ong. A two-pass improved encoding scheme for fractal image compression. *Computer Graphics, Imaging and Visualization, International Conference on*, 0:214–219, 2006.

[5] Yuval Fisher, editor. *Fractal image compression: theory and application*. Springer-Verlag, London, UK, 1995.

[6] Jinshu Han. Speeding up fractal image compression based on local extreme points. *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, ACIS International Conference on*, 3:732–737, 2007.

[7] Xiangjian He, Huaqing Wang, Qiang Wu, Tom Hintz, and Namho Hur. Fractal image compression on spiral architecture. *Computer Graphics, Imaging and Visualization, International Conference on*, 0:76–83, 2006.

[8] Xiaotong Hu, Shuping Qiu, and Hideo Kuroda. Reduction of transmitted information using similarities between range blocks in fractal image coding. *Image and Graphics, International Conference on*, 0:189–194, 2007.

[9] G.E. Oien, Z. Baharav, S. Lepsoy, and E. Karnin. A new improved collage theorem with applications to multiresolution fractal image coding. *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, 2:565–568, 1994.

[10] Ghim-Hwee Ong, Chorng-Meng Chew, and Yi Cao. A simple partitioning approach to fractal image compression. In *SAC '01: Proceedings of the 2001 ACM symposium on Applied computing*, pages 301–305, New York, NY, USA, 2001. ACM.

[11] D. Saupe. Accelerating fractal image compression by multi-dimensional nearest neighbor search. In *DCC '95: Proceedings of the Conference on Data Compression*, page 222, Washington, DC, USA, 1995. IEEE Computer Society.

[12] Dietmar Saupe. The futility of square isometries in fractal image compression. In *in: Proc. ICIP-96 IEEE International Conference on Image Processing*, pages 161–164, 1996.

[13] Nileshsingh V. Thakur and O. G. Kakde. Color image compression on a pseudo spiral architecture. *Cybernetics and Intelligent Systems, 2nd IEEE International Conference on*, pages 228–233, 2006.

[14] Nileshsingh V. Thakur and O. G. Kakde. Color image compression on spiral architecture using optimized domain blocks in fractal coding. *Information Technology: New Generations, Third International Conference on*, 0:234–242, 2007.

[15] Huaqing Wang, Qiang Wu, Xiangjian He, and Tom Hintz. A novel interactive progressive decoding method for fractal image compression. *Innovative Computing, Information and Control, International Conference on*, 3:613–617, 2006.

Internetové zdroje

[16] GCC, the GNU compiler collection. http://gcc.gnu.org.

[17] Imagemagick: Convert, edit, and compose images. http://www.imagemagick.org.

[18] Loki library. http://sourceforge.net/projects/loki-lib.

[19] Qt — a cross-platform application and UI framework. http://www.qtsoftware.com.

[20] The waterloo fractal coding and analysis group. http://links.uwaterloo.ca.

[21] Wikipedia. Lab color space — wikipedia, the free encyclopedia, 2009.