VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

OBRAZOVÉ MOZAIKY

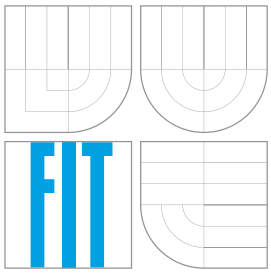
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VOJTĚCH BÍLÝ

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

OBRAZOVÉ MOZAIKY

IMAGE MOSAICS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VOJTĚCH BÍLÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MIROSLAV ŠVUB

BRNO 2009

Abstrakt

Tato bakalářská práce se zabývá vytvářením obrazových mozaiek – obrázků, které jsou složeny z menších obrázků tak, aby se z dostatečné vzdálenosti jevily jako jeden celek. Dále se tato práce zabývá tvorbou databáze obrázků potřebných k vytváření mozaiek.

Abstract

This bachelor's thesis deals with creating picture mosaics – pictures, which are compound of a smaller pictures, so that they appears as integral units from the sufficient distance. This thesis also deals with generation of picture database necessary to compositing mosaics.

Klíčová slova

obrazová mozaika, počítačová grafika, rastrová grafika, porovnávání obrázků, ASCII art

Keywords

picture mosaic, computer graphics, raster graphics, picture comparing, ASCII art

Citace

Vojtěch Bílý: Obrazové mozaiky, bakalářská práce, Brno, FIT VUT v Brně, 2009

Obrazové mozaiky

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Miroslava Švuba. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vojtěch Bílý
18. května 2009

Obsah

Kapitola 1

Úvod

Mozaika je obraz, vytvořený vkládáním malých kousků barevného skla, kamene nebo jiných materiálů do cementu, tmelu nebo jiného pojiva. Tento obraz se z dostatečné vzdálenosti jeví jako jeden celek a to díky integrační schopnosti lidského oka. (zdroje: [1] [7]) Obrazová moziaka se tvoří z malých kousků materiálu (dlaždic), na kterých je již namalován jiný obraz. Tato práce se zabývá vytvářením obrazových mozaiek pomocí počítače. Mozaika je tedy složena z rastrových obrázků uložených v počítači.

První využití obrazových mozaiek v počítačích byl ASCII art. Jako dlaždice (dílčí obrázky) se používaly znaky obsažené v ASCII tabulce. Tímto způsobem bylo možné zobrazovat grafiku na výstupních znakových zařízeních. Moderní obrazové mozaiky se využívají například v reklamní sféře, v níž se především jedná o nevšední prezentaci obrazů se schopností upoutat pozornost.

Následující kapitola se zabývá teorií. Nejprve historií mozaiek, dále základními informacemi o prezentaci obrázků v počítači, barevnými prostory, porovnáváním obrázků na základě barevné podobnosti, zmenšováním a zvětšováním obrázků, ASCII artem a nakonec formátem XML.

Třetí kapitola popisuje cíle, kterých by mělo být dosaženo implementací, a návrh samotného systému tvorby mozaiek.

Čtvrtá kapitola je stěžejní částí této bakalářské práce. Zde se podrobně popisuje implementace navrženého systému.

Pátá kapitola na příkladech názorně ukazuje způsob ovládání jednotlivých aplikací systému.

V šesté kapitole jsou zhodnoceny dosažené výsledky. Druhá část této kapitoly nastiňuje další možná vylepšení.

V poslední, sedmé, kapitole jsou shrnuty všechny poznatky této bakalářské práce.

Kapitola 2

Teorie

Tato kapitola se zabývá teorií, kterou je potřeba znát k pochopení problematiky řešené v následujících kapitolách.

2.1 Historie mozaiek

2.1.1 Nejstarší nálezy

Nejstarší dochované mozaiky pochází ze 4. stol. př.K. z Makedonie. V antice a starověkém orientu byly mozaiky nejvíce používány jako výzdoby podlah. Často zobrazovaly lovecké scény nebo důležité historické či mytologické momenty. (zdroj: [3])

2.1.2 Křesťanství

Křesťané převzali od starověkých římanů tradici vyzdobování významných míst mozaikami a od 4. stol. n. l. se mozaiky objevují v basilikách. (zdroj: [3])

Speciální druh mozaiky – vitraj (různobarevné kusy skla spojené kovovými plíšky), se začal využívat při stavbách středověkých katedrál. Katedrály měly okna tak veliká, že je nebylo možné zasklít jedním kusem skla. Proto se používala skla spojená. Postupem času se spojování barevných skel stalo uměleckou záležitostí. Právě v podobě vitrají se používání mozaiek přeneslo až do dnešních dnů. (zdroj: [9])

2.2 Počítačová grafika a ukládání obrázků

Počítačová grafika je obor zabývající se tvorbou a analýzou grafické obrazové informace. Tento obor se může dělit do několika oblastí: editace obrázků, animace, zpracování videa, 3D modelování či 3D rendering. Existují dva hlavní způsoby přístupu k počítačové 2D grafice: vektorová a rastrová grafika.

Rastrový obrázek je tvořen čtvercovou maticí (2-rozměrným polem) primitiv. Tato primitiva se nazývají pixely. Protože rastrová grafika pracuje převážně s RGB barevným modelem (viz další kapitola), pro každý pixel se uchovává hodnota červené, zelené, modré barvy a volitelně i informace o průhlednosti. Rastrový obrázek se získává manuálně (editory rastrové grafiky, např. Malování ze systému MS Windows), převodem z vektorového obrázku (rasterizace) nebo snímáním reálných objektů (fotoaparát, skener). Také lidské oko "vidí"rastrově, protože v oku je matice (sítnice) receptorů (tyčinky a čípky) reagujících na barevné záření.

Ve vektorovém obrázku jsou všechny části uloženy jako popis vektorových entit (úsečka, Beziérova křivka, elipsa, obdélník atd). Každá entita má svou polohu, rozměry, barvy a informace specifické pro danou entitu. Tento systém je často uložen ve formě XML dokumentu. Vektorový obrázek se většinou vytvoří manuálně, převod z rastrového obrázku je složitý a nedokonalý (jedním z mála použití převodu je rozpoznávání textu). Mezi největší výhody vektorové grafiky patří změna velikosti obrázku bez ztráty kvality, možnost práce s jednotlivými entitami odděleně či menší datová velikost. Protože v současnosti výstupní zařízení (monitor, projektor, tiskárna) zobrazují pouze rastrovou grafiku, provádí se rasterizace, tedy převod vektorové grafiky na rastrovou.

2.3 Barevné modely

Barevný model popisuje základní barvy a způsob jejich mísení. Barva je v přírodě dána světlem různých vlnových délek a různé barevné modely se snaží napodobit barvu co nejvěrněji. V praxi se používají modely, u kterých je zvolen vhodný kompromis mezi přesností podání barevného dojmu a složitostí konkrétního modelu. Barevné modely se rozdělují podle typu míchání barev.

- **Aditivní míchání** – barevný model pracující se světelnými zdroji barev (např. monitor, kamera nebo projektor). Toto míchání využívá RGB model.
- **Subtraktivní míchání** – barevný model pracující s odrazem bílého světla, tedy s mícháním pigmentů (např. tiskárna, plotr nebo ofset). Toto míchání využívá CMYK model.

(zdroj: [5])

2.3.1 RGB

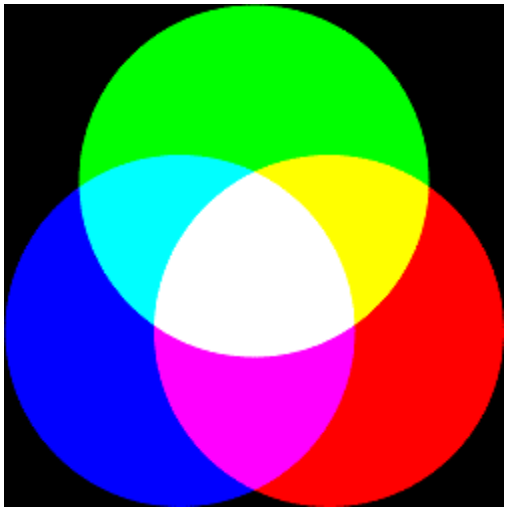
RGB barevný model se zakládá na teorii Younga – Helmholtze [1] , trojbarevného vidění, a na Maxwellově barevném trojúhelníku [2]. Použití RGB barevného modelu má své kořeny v letech 1953 u RCA barevné TV normy a v použití Edwin Loandova RGB standardu v Land/Polaroidu.

RGB (Red-Green-Blue, červená-zelená-modrá) je aditivní způsob míchání barev (viz obr. 2.1) používaný ve všech monitorech a projektorech. Používá míchání vyzařovaného světla, tudíž nepotřebuje žádné vnější světlo (monitor zobrazuje i v naprosté tmě).

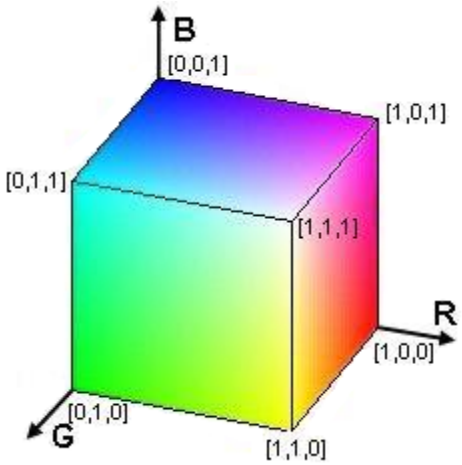
RGB model sám o sobě nedefinuje co je míněno červenou, modrou či zelenou a tak výsledek smíchání složek není přesný, ale relativní. Když bude přesně definována chromatičnost barevných složek, potom se barevný model stává absolutním barevným prostorem. Jakákoliv barva je udána mohutností tří základních barev. Základní barvy mají vlnové délky 630, 530 a 450 nm. Mohutnost se udává buď v poměru (0 – 1) nebo podle použité barevné hloubky jako určitý počet bitů vyhrazených pro barevnou komponentu (pro 8 bitů na komponentu je rozsah hodnot 0 – 255, pro 16 bitů na komponentu je rozsah hodnot 0 – 65535), přičemž čím větší je mohutnost, tím s vyšší intenzitou se barva komponenty zobrazuje.

Model RGB je možné zobrazit jako jednotkovou krychli (viz obr. 2.2), ve které každá z kolmých hran udává škálu mohutností barevných složek. Potom libovolný bod se souřadnicemi (r,g,b) v této krychli udává hodnotu výsledné barvy. (zdroj: [8])

[1] `http://en.wikipedia.org/wiki/Young%E2%80%93Helmholtz_theory`
[2] `http://en.wikipedia.org/wiki/Maxwell%27s_discs#Maxwell.27s_disc`



Obrázek 2.1: Aditivní míchání barev



Obrázek 2.2: Reprezentace RGB modelu jako jednotková krychle

Tento model je ideální pro zobrazovací techniku, ale není vhodný pro manuální práci s barvami. Např. pokud by jsme chtěli dostat ze světle modré barvy tmavě modrou, musí se změnit všechny tři základní složky. Taktéž při práci v odstínech šedé je potřeba vypočítávat a nastavovat všechny tři složky.

Tuto nepříjemnou vlastnost řeší HSV barevný prostor.

2.3.2 HSV

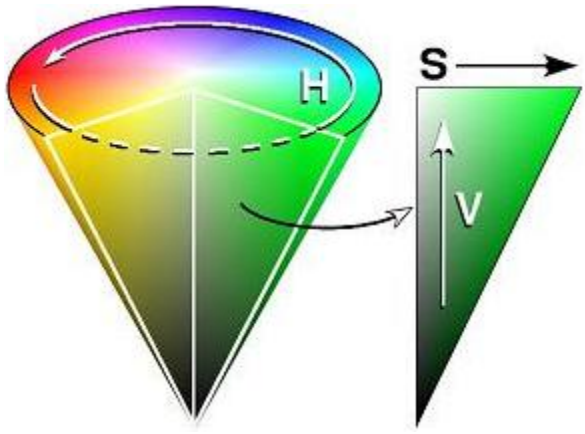
HSV barevný model vytvořil v roce 1978 Alvy Ray Smith. Tento barevný model nejvíce odpovídá lidskému vnímání barev. Skládá se ze tří složek:

- **Hue** – barevný tón neboli odstín – barva odražená nebo procházející objektem. Měří se jako poloha na standardním barevném kole (0° červená, 120° zelená, 240° modrá a 360° opět červená). Obecně se odstín označuje názvem barvy.

- **Saturation** – sytost barvy, příměs jiné barvy. Někdy též chroma, síla nebo čistota barvy, představuje množství šedi v poměru k odstínu, měří se v procentech od 0% (šedá) do 100% (plně sytá barva). Na barevném kole vzrůstá sytost barvy od středu k okrajům. Např. červená s 50% sytostí bude růžová.

- **Value** – hodnota jasu, množství bílého světla. Způsobuje relativní světlost nebo tmavost barvy. Jas vyjadřuje kolik světla barva odráží, dalo by se také říct přidávání černé do základní barvy. Měří se v procentech od 0% (černá) do 100% (bílá).

HSV model je reprezentace bodů v barevném prostoru RGB. Tato reprezentace se nejčastěji zobrazuje jako převrácený barevný kužel s černým bodem dole a plně sytými barvami kolem kruhu nahoře (viz obr. 2.3). (zdroj:[6])



Obrázek 2.3: Reprezentace HSV prostoru jako barevný kužel

Protože vybírání barvy v 3-rozměrném modelu není příliš uživatelsky přívětivé, používá se 2-rozměrné zobrazení HSV prostoru. Na obrázku 2.4 jsou ukázaný příklady těchto zobrazení.

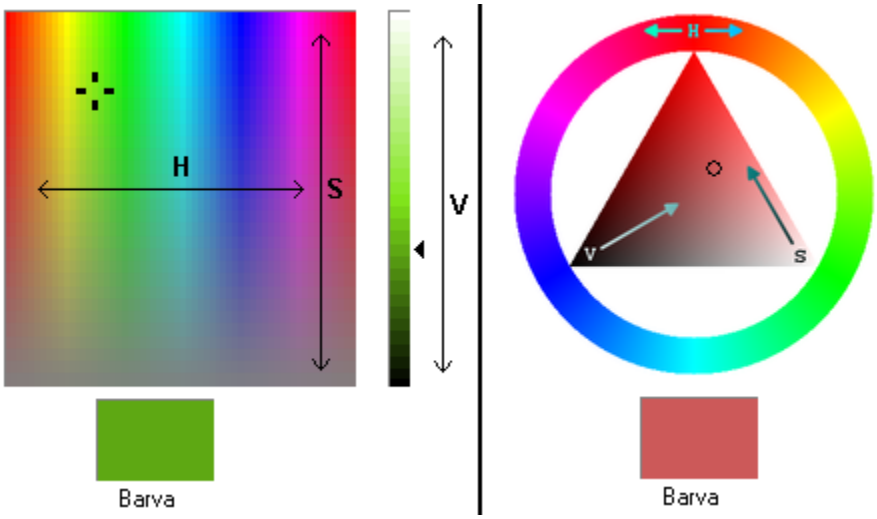
V části o modelu RGB byly zmiňovány problémy se změnou barvy. Pokud chceme v HSV modelu dostat ze světle modré barvy tmavě modrou, stačí měnit pouze složku jasu. Stejně tak práce v barvách šedi je usnadněna tím, že stačí nastavit sytost na 0 a změnou jasu měnit požadovaný odstín. Složka H nemá na výsledek žádný vliv.

Toto jsou velmi zajímavé možnosti úpravy barev, ale tato práce se zabývá HSV kvůli jiné vlastnosti. Jedná se o takzvanou HSV rotaci barev obrázku. Princip je velmi jednoduchý. Složky jas a sytost zůstávají v celém obrázku stejné a složka odstín se mění o konstantní hodnotu. Tímto způsobem můžeme jednoduše docílit změny barvy obrázku, při zachování původního motivu, jak je vidět na obrázku 2.5.

Bohužel rotace barev v HSV má dvě nepříjemné vlastnosti. První z nich se týká různobarevnosti obrázku. Rotace má význam pouze u obrázků s jednou dominantní barvou. Rotací barev různobarevných obrázků vznikjí nepřirozené kombinace které působí velmi rušivě. Pokud se tedy v této práci bude zmiňovat HSV rotace, tak se bude jednat o rotaci jednobarevných obrázků.

Druhá vlastnost se týká průměrné barvy. Jak bylo zmiňováno výše, složka *Hue* je reprezentována ve formě kruhu, má rozsah 0–359. Kvůli 8-bitové prezentaci dat v počítači, kdy je maximální hodnota rovna 255, se hodnota odstínu dělí dvěmi, tedy nabývá hodnot

0–179. Problém nastává u červených obrázků, protože jako odstíny červené se dají považovat hodnoty 0 – 10 a 170 – 180. Průměrná hodnota červeného obrázku je tedy okolo hodnoty 90, což odpovídá azurově modré. Jako jediné řešení tohoto problému se ukázalo zjistit maximální a minimální barvu obrázku. Pokud se minimální barva pohybuje mírně nad nulou a maximální barva mírně pod 180, tak se musí obrázek upravit. Buď stačí barvy posunout o 90, spočítat průměr a tento průměr posunout zpět o 90 nebo všechny hodnoty přesahující 90 změnit na *-(180-hodnota)* a spočítat průměr.



Obrázek 2.4: Možnosti výběru barvy v HSV modelu

Obrázek 2.5: HSV rotace barev. Odstín pravého obrázku je posunut o 120° oproti levému obrázku.

2.4 Porovnávání obrázků

Tato část práce se zabývá pouze porovnáváním rastrové grafiky. Existuje více možností jak porovnávat obrázky:

- **Podle nejvýznamnější barvy** – z histogramu se určí nejvýznamnější barva a hledají se obrázky s nejpodobnější barvou.

- **Podle barevnosti** – vypočítá se barevný průměr několika částí nebo celého obrázku a podle těchto průměrů se porovnává.
- **Podle nalezených objektů** – provede se detekce hran a porovnává se jejich tvar a vzdálenost.

Podrobněji si rozebereme druhou možnost, tedy porovnávání podle barevnosti. Uvažujme obrázek **A** a množinu **M** obrázků (o stejné velikosti jako **A**), z kterých budeme vybírat obrázek nejpodobnější obrázku **A**. Nejpodobnější obrázek je takový, který má nejmenší rozdíl barev. Rozdíl barev se vypočítá podle vzorce 2.1.

$$\sqrt{\sum_{i=1}^{A_h * A_w} \left((R_{Ai} - R_{Mi})^2 + (G_{Ai} - G_{Mi})^2 + (B_{Ai} - B_{Mi})^2 \right)} \tag{2.1}$$

Co však znamenají tyto symboly?

- A_h – A_{height}, výška obrázku **A** v pixelech.
- A_w – A_{width}, šířka obrázku **A** v pixelech.
- R_{Ai} – Mohutnost červené barvy i-tého pixelu obrázku **A**.
- G_{Ai} – Mohutnost zelené barvy i-tého pixelu obrázku **A**.
- B_{Ai} – Mohutnost modré barvy i-tého pixelu obrázku **A**.
- R_{Mi} – Mohutnost červené barvy i-tého pixelu obrázku z množiny**M**.
- G_{Mi} – Mohutnost zelené barvy i-tého pixelu obrázku z množiny**M**.
- B_{Mi} – Mohutnost modré barvy i-tého pixelu obrázku z množiny**M**.

Postup při vyhodnocování rozdílu barev nejlépe vystihne pseudokód:

```
int rozdilBarev = 0;
for(int i=0; i<A_height; i++){    // A_height = výška v~pixelech
  for(int j=0; j<A_width; j++){   // A_width = šířka v~pixelech
    rozdilBarev += (A[i,j].red - M[i,j].red)^2;
    rozdilBarev += (A[i,j].green - M[i,j].green)^2;
    rozdilBarev += (A[i,j].blue - M[i,j].blue)^2;
  }
}
rozdilBarev = sqrt(rozdilBarev);
```

2.5 Změna rozměrů obrázků

2.5.1 Resampling (Převzorkování)

Při změně velikosti (zvětšování, zmenšování) rastrového obrázku (bitmapy) se mění počet pixelů. Je tedy potřeba vypočítat barvy nových pixelů, takzvaně převzorkovat obrázek. Hodnoty v nových bodech se vypočítají ze známých hodnot ve starých bodech. V bitmapě

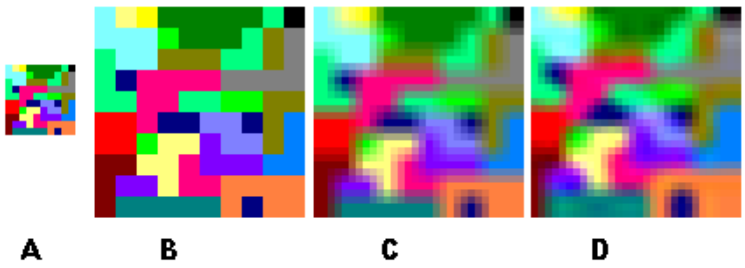
je pro každý pixel jeden vzorek (barva). Převzorkování takovéto bitmapy začíná vytvořením vzorkovací mřížky, která se umístí přes bitmapu. Vzhledem k vzdálenosti každého bodu mřížky a původních středů pixelů a vzhledem k používanému vzorkovacímu algoritmu se každému bodu mřížky nastaví nová barva. Vzorkovací algoritmy využívají dvou-dimenzionální interpolaci pro každou barevnou složku.

Nejjednodušší vzorkovací metoda je *Interpolace nejbližším sousedem* (*Nearest-neighbor interpolation*). Jde o výběr barvy podle jednoho nejbližšího pixelu, bez zohledňování ostatních okolních pixelů. Tato metoda je nejrychlejší a nejjednodušší, ale dává nejhorší výsledky.

Bilineární iterpolace má nepoměrně lepší výsledky než předchozí metoda. Pro vzorkovaný bod se vezmou středy nejbližších 4 pixelů a v závislosti na jejich vzdálenosti od vzorkovaného bodu se lineárně aproximují mohutosti jejich barev. Mohutnost barvy se vypočítavá podle vzorece 2.2. Tento vzorec popisuje situaci znázorněnou na obrázku 2.7.

U *bikubické interpolace* se aproximace provádí polynomem třetího stupně z 16-pixelového okolí. Tato metoda velmi dobře vyhlazuje hrany v obrázcích, ale využívá náročnější výpočty.

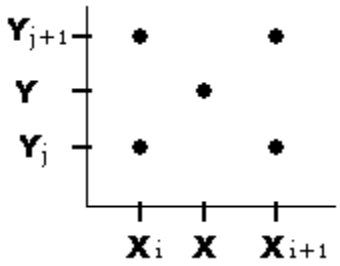
Interpolace nejbližším sousedem a *bilineární interpolace* se využívají v aplikacích, kdy záleží na rychlosti provádění, *bikubická interpolace* při požadavku dobrého vyhlazování.



Obrázek 2.6: Interpolace barev. **A** je originální obrázek v rozlišení 10x10 pixelů, ostatní byly zvětšeny na 30x30 pixelů. **B** byl získán interpolací nejbližšího souseda, **C** bilineární interpolací a **D** bikubickou interpolací.

$$z(x,y) = (1-t)(1-u)z_{i,j} + t(1-u)z_{i+1,j} + (1-t)uz_{i,j+1} + tuz_{i+1,j+1} \tag{2.2}$$

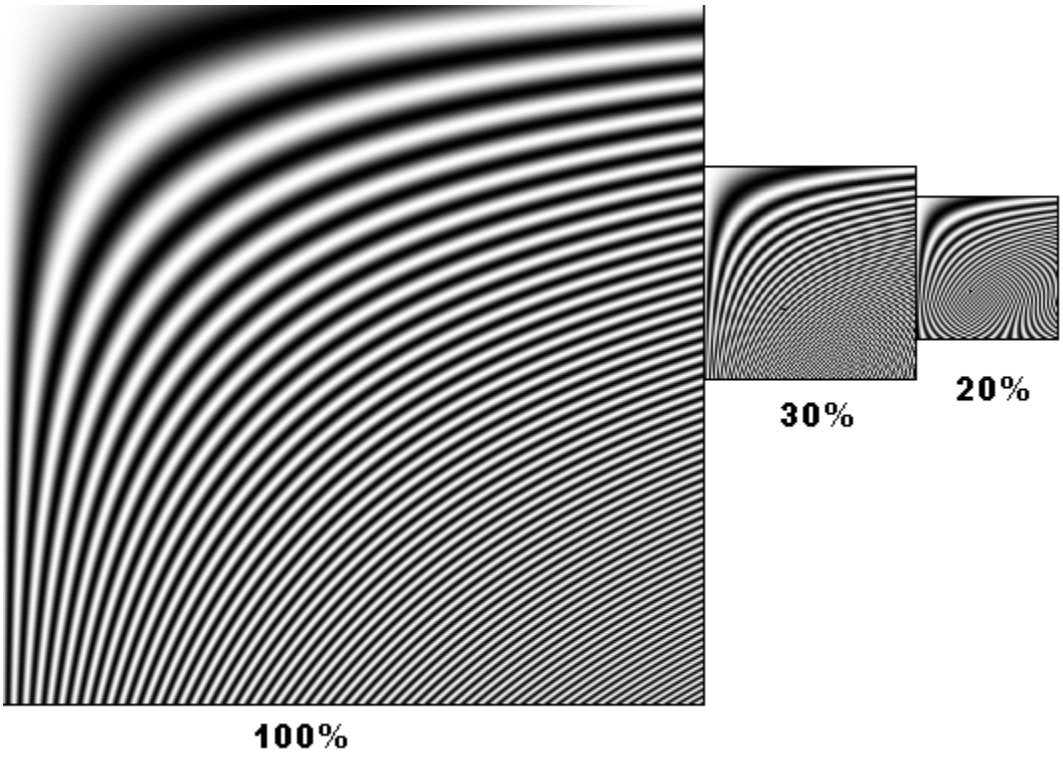
$$t = (x-x_i)/(x_{i+1}-x_i) \qquad u = (y-y_j)/(y_{j+1}-y_j) \tag{2.3}$$



Obrázek 2.7: Rozložení bodů při výpočtu barvy bilineární transformací, popsaným vzorcem 2.2. **Z** je mohutnost barvy, **t** a **u** udávají poměr vzdáleností mezi body.

2.5.2 Moiré

Moiré jsou různé pravidelné vzory, které do obrázku nepatří. Vznikají při zmenšování obrázků s drobnou periodickou texturou. Děje se tak kvůli aliasingu. Aliasing vzniká, když funkci s vysokou frekvencí navzorkujeme na příliš málo místech. Výsledek se bude jevit jako jiná funkce s nižší frekvencí. Dochází tak ke vzniku falešných frekvencí, které v obrázku vytváří moiré. Příklad moiré je na obrázku 2.8. (zdroj: [2])



Obrázek 2.8: Moire

2.6 ASCII art

ASCII art je výtvarné umění, které pracuje s textem jako s výtvarným médiem. Obrázky, které tvoří, se skládají ze znaků kódu ASCII. Znaky musí být z neproporcionálního fontu (každé písmeno má stejnou šířku, podobně jako na psacím stroji či terminálu).

První ASCII artové obrázky vznikaly již na konci 19. století, kdy se začaly rozšiřovat psací stroje.

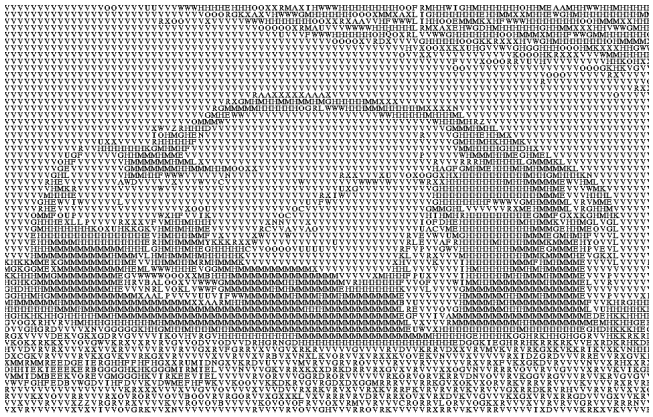
ASCII art se používá kdekoli se text přenáší nebo zobrazuje snadněji než grafika anebo v případech, kdy není přenos obrázků vůbec možný, tedy na psacích strojích, dálnopisech, negrafických počítačových terminálech,v starších počítačových sítích (např. BBS), emailech či v Usenet news. ASCII art se také používá ve zdrojových kódech programů pro zobrazení loga firmy nebo produktu a kreslení diagramů. (zdroj:[4]) V zásadě rozdělujeme ASCII art

na 2 druhy:

- Podle tvaru znaku (”Oldskool”nebo ”Amiga”styl).
- Podle zabarvenosti políčka (”Newskool”styl).

Rozdíl je nejlépe vidět na příkladech. Obrázek 2.9 představuje ASCII art tvořený na základě tvaru znaku, naproti tomu obrázek 2.10 představuje ASCII art tvořený na základě zabarvenosti políčka.

Obrázek 2.9: ASCII art – Kráva



Obrázek 2.10: ASCII art – Auto

Obrazovým mozaikám odpovídá druhý způsob tvorby ASCII artu, kdy se porovnává intenzita výseku obrázku s intenzitou jednotlivých znaků ASCII a výsledný obrázek je sestaven z nejpodobnějších znaků. Například pro nejsvětlejší místo se používá znak *mezera* a pro nejtmavší *mříže*. (zdroj: [1])

2.7 XML

2.7.1 XML

XML (eXtensible Markup Language) je univerzální značkovací jazyk vycházející ze značkovacího jazyka SGML (Standard Generalized Markup Language). XML vyvíjí organizace World Wide Web Consortium (W3C). Označení extensible (rozšiřitelný) znamená, že si uživatelé mohou definovat vlastní značky (názvy elementů). Jazyk je určen pro ukládaní strukturovaných dat v podobě čistého textu, sdílení a vyměňování těchto dat mezi aplikacemi i systémy. XML dokument se skládá z vnořených XML elementů. Tyto elementy

mohou mít atributy a obsah. Syntaxe XML elementu může být například:

<jméno_elementu *jméno_atributu* = "*hodnota_atributu*">
Obsah elementu
</jméno_elementu>

Celý XML dokument může vypadat následovně:

```
<?xml version="1.0"?>
<Zamestnanci>
  <Zamestnanec ID="1">
    <Jmeno>Jan</Jmeno>
    <Prijmeni>Novák</Prijmeni>
    <Plat>24000</Plat>
  </Zamestnanec>
  <Zamestnanec ID="2">
    <Jmeno>Václav</Jmeno>
    <Prijmeni>Řezník</Prijmeni>
    <Plat>25000</Plat>
  </Zamestnanec>
</Zamestnanci>
```

2.7.2 XML schéma

Jak je vidět z ukázky, samotné XML nedefinuje žádná pravidla, v jakém formátu musí být uloženy informace. Z tohoto důvodu se používá XML schéma. Jedná se o popis elementů v XML dokumentu. XML schéma je zapsané pomocí XML. Pro výše uvedený příklad platí toto schéma:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Zamestnanci">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Zamestnanec" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Jmeno" type="xs:string"/>
              <xs:element name="Prijmeni" type="xs:string"/>
              <xs:element name="Plat" type="xs:decimal"/>
            </xs:sequence>
            <xs:attribute name="ID" type="xs:integer"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Na příkladu můžeme vidět, že celé schéma je dokument XML, který používá speciální elementy. Všechny tyto elementy musí patřit do jmenného prostoru (xmlns – XML Name Space) `http://www.w3.org/2001/XMLSchema`. Obvykle se pro tento jmenný prostor používá prefix `xs` nebo `xsd`.

Celé schéma musí být vždy uzavřeno v elementu *schema* obsahujícím definice elementů, které lze použít jako kořenové elementy. Definice elementu se zapisuje pomocí elementu *element*. Pro každý element musí schéma určit jeho typ. Rozlišovány jsou dva druhy typů – jednoduché a komplexní. Jednoduché typy se používají pro skalární hodnoty jako řetězec, číslo, datum apod. Obsahuje-li však element další elementy nebo atributy, musíme použít komplexní typ *complexType*. Pomocí elementu *sequence* říkáme, že se element skládá z po sobě následujících elementů. Každý z těchto elementů je povinný, má určené své jméno pomocí atributu *name* a datový typ pomocí atributu *type*. Datové typy se uvádějí jako kvalifikované názvy patřící rovněž do jmenného prostoru XML schémat. Atributy se deklarují až za vnořenými elementy pomocí elementu *attribute*. U atributů je rovněž určen jeho název a datový typ. Atribut *maxOccurs* určuje maximální počet výskytů elementů. Hodnota *unbounded* se používá k označení neomezeného výskytu elementu. (zdroj: [10])

Kapitola 3

Návrh

Tato kapitola se zabývá návrhem systému a cíli, jichž by měl dosáhnout.

Podle zadání bakalářské práce má výsledný systém umět sestavovat obrazové mozaiky z databáze obrázků. Program by měl být přeložitelný na systémech MS Windows a Linux.

Systém je určen pro běžné domácí použití, proto by mělo být jeho ovládání zcela intuitivní a hardwarové nároky by měly odpovídat současným osobním počítačům.

Celý systém se tedy bude skládat ze 3 částí:

- Databáze obrázků a soubor s jejich popisem.
- Aplikace pro zmenšování obrázků a přidávání informací do databáze (zmensi.exe).
- Aplikace na vytváření mozaiek (sestav.exe).

3.1 Databáze

Obrázky v jedné databázi musí být umístěny ve stejném adresáři jako soubor s jejich popisem a musí všechny mít stejné rozměry. Ideální velikost by měla být zvolena jako kompromis, protože na velmi malých obrázcích většinou není rozeznatelná obrazová informace, a z velkých obrázků by vznikla příliš nekomaktní mozaika. Autor experimentálně zjistil, že takovým kompromisem jsou obrázky o rozměrech 20x20 pixelů.

Soubor s popisem obrázků bude XML soubor. Tento XML soubor musí odpovídat všem pravidlům, tzn. soubor začíná hlavičkou, obsahuje právě jeden kořenový element a všechny jeho elementy musí být uzavřeny.

Dále by měl vyhovovat následujícímu XML schématu:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="database">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="picture" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string"/>
              <xs:element name="type" type="xs:string"/>
              <xs:element name="pixels" type="xs:string"/>
```

```
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Tedy jeden kořenový element *database* obsahující neomezený počet elementů *picture*. Každý element *picture* obsahuje elementy *name*, *type* a *pixels*.

Element *name* by měl obsahovat text s názvem obrázku. Tento název musí být pro celou databázi jedinečný. Element *type* by měl obsahovat text popisující obrázek, například: "tvář", "auto", "krajina". Těchto typů by mělo být v databázi pouze několik. Element *pixels* by měl obsahovat text popisující barvy obrázku. Aby bylo možné dosáhnout plynulých přechodů a ostrých hran, nebude se ukládat průměrná barva celého obrázku, ale průměrné barvy v 9 částech. Kvůli větší přehlednosti se budou mohutnosti barev ukládat v hexadecimálním formátu.

3.2 Zmenšovač

Aplikace by měla sloužit k přidávání obrázků do stávající databáze, popřípadě musí umět tuto databázi vytvořit. Protože se předpokládá spouštění v rámci skriptu, bude se jednat o konzolovou aplikaci. Ke správnému fungování musí být přes parametry předán název obrázku a rozměry, na které má být obrázek zmenšen.

Výstupem programu bude obrázek zmenšený na zadané rozměry a přidání informací do XML souboru.

Aplikace bude naprogramována v programovacím jazyku *C++*, pro práci s obrázky (zmenšování, přístup k barvě pixelu) bude využívat funkce knihovny *OpenCV*.

Kvůli jednoduchosti a lineárnosti (téměř žádné větvení programu) nebude aplikace využívat prvky objektově orientovaného programování.

3.3 Sestavovač

S touto aplikací by již měli uživatelé interaktivně pracovat, proto bude obsahovat grafické uživatelské rozhraní.

Aplikace musí umět načíst databázi, načíst obrázek, z obrázků v databázi sestavit mozaiku tak, aby odpovídala načtenému obrázku a mozaiku uložit.

Dále je vhodné, aby uživatel viděl jak načtený obrázek tak i výslednou mozaiku. Proto bude potřeba, aby aplikace měla část, kde se budou tyto obrázky zobrazovat.

Princip samotné tvorby mozaiky je znázorněn na obrázku 3.2. Níže uvedené rozšíření je na obrázku jako 5. rámeček od shora.

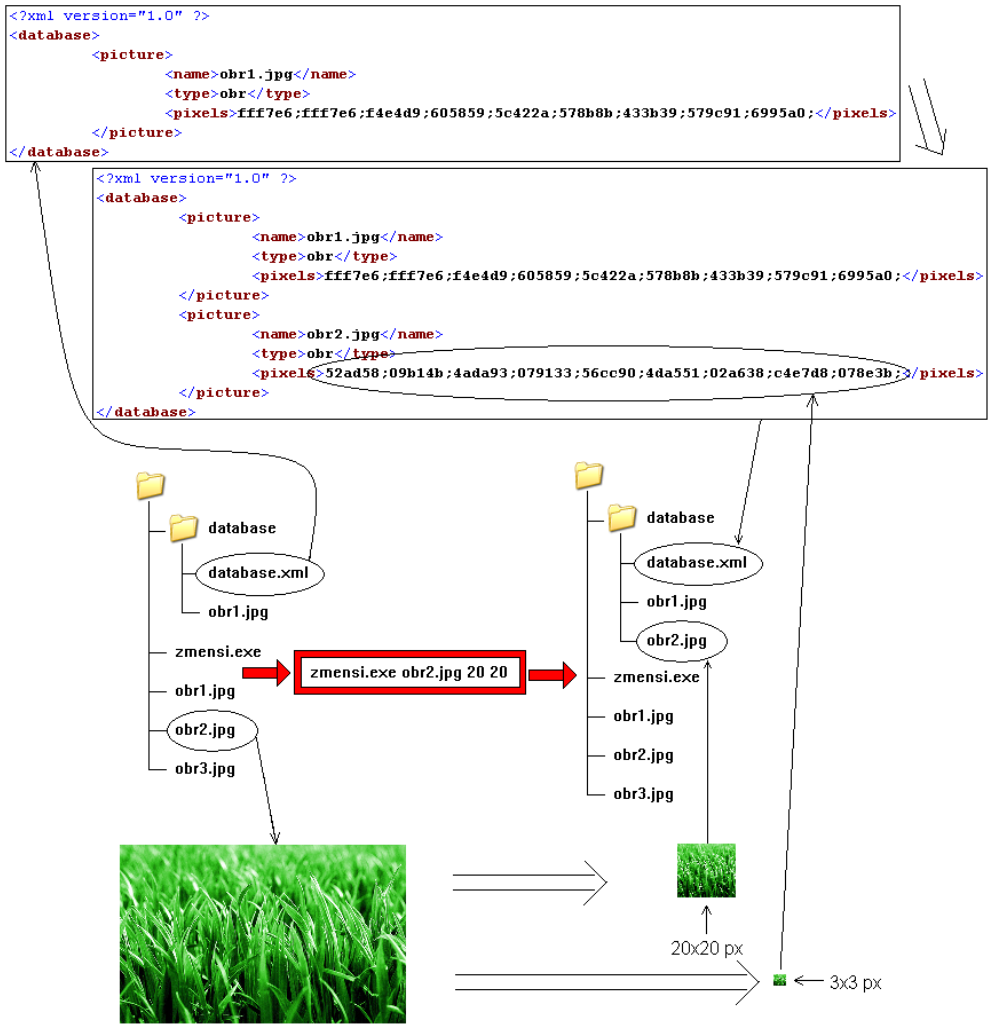
Protože existuje velké množství vzájemně si podobných programů na vytváření mozaiek, po konzultaci s vedoucím mé bakalářské prace jsem se rozhodl přidat rozšíření v podobě HSV rotace barev obrázků z databáze. Princip rozšíření je následující:

1. Pokud se v databázi nenalezne obrázek barevně odpovídající porovnávané části (*obr1*), z databáze se vybere intenzitně nejpodobnější obrázek (*obr2*). Mez, kdy si již obrázky nejsou podobné, je určena konstantou na začátku programu.

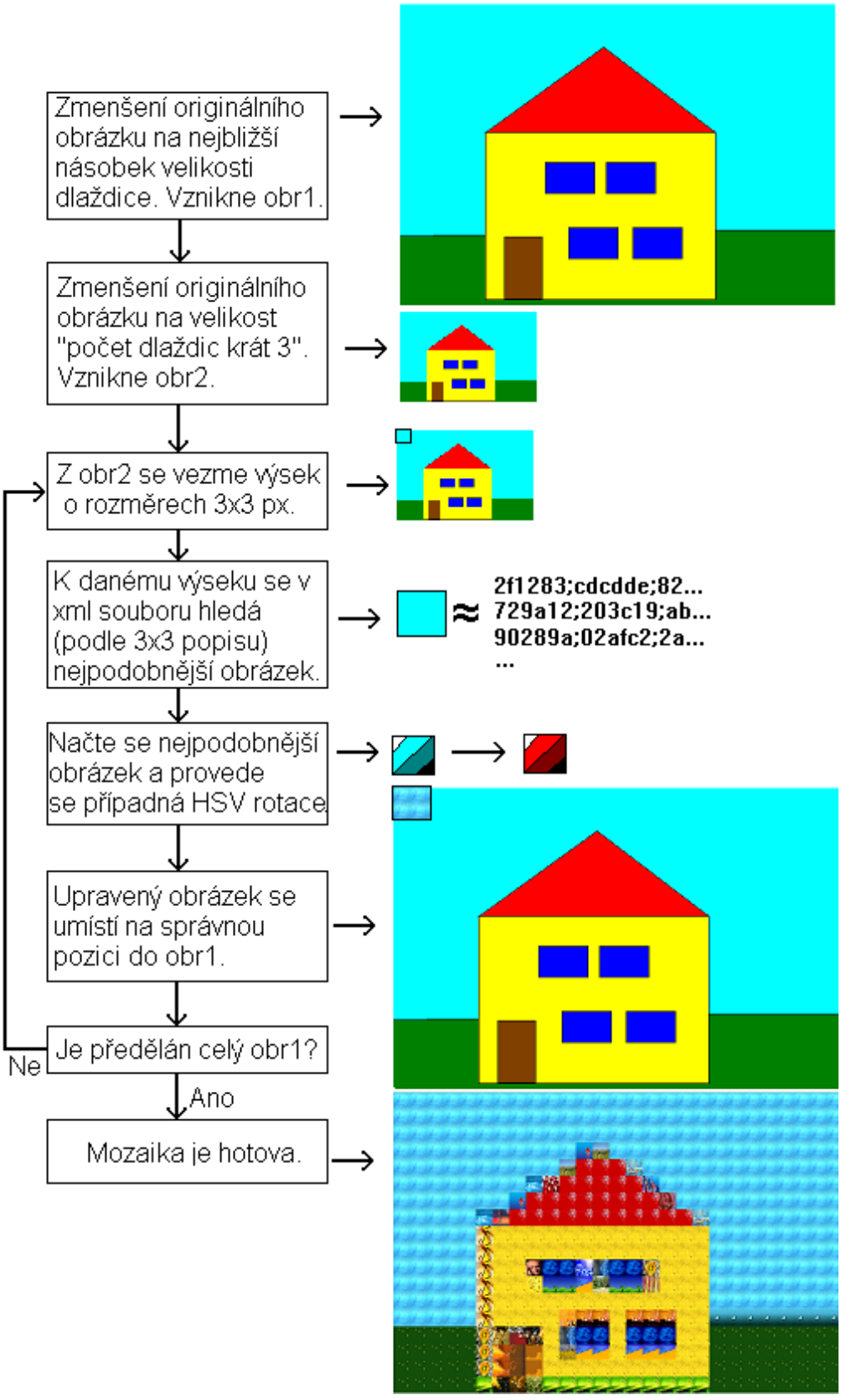
2. Obrázky *obr1* a *obr2* se převedou na barevný model HSV.
3. Spočítají se aritmetické průměry barev obrázků *obr1* a *obr2*.
4. Průměry se od sebe odečtou.
5. K barvě každého pixelu obrázku *obr2* se přičte rozdíl získaný v předchozím kroku.
6. Obrázek *obr2* se převede zpět na barevný model RGB

Díky tomuto principu postačuje menší databáze, protože mnoho barevných kombinací může být dopočítáno.

Aplikace bude naprogramována v *programovacím jazyku C++*, pro práci s obrázky bude využívat funkce knihovny *OpenCV*, pro GUI (grafické uživatelské rozhraní) bude využívat funkce knihovny *wxWidgets* a pro práci s XML knihovnu *TinyXML*.



Obrázek 3.1: Princip tvorby databáze.Vlevo uprostřed je stav databáze před spuštením aplikace (naznačeno uprostřed). Vpravo je stav po skončení aplikace.



Obrázek 3.2: Princip sestavování mozaiky.

Kapitola 4

Implementace

Tato kapitola popisuje implementaci systému. Systém obsahuje zdrojové soubory *zmensi.cpp* pro vytvoření zmenšovače, *sestav.cpp* pro vytvoření sestavovače, *makefile* pro kompilaci zdrojových souborů, *skripty* pro vytvoření databáze a knihovny wxWidgets, OpenCV a TinyXML.

4.1 zmensi.cpp

Celá implementační logika se nachází v hlavní funkci `int main(int argc, char *argv[])`. Protože aplikace po spuštění nekomunikuje s uživatelem, lze poměrně přesně popsat jednotlivé kroky:

1. Nejprve se provádí kontrola argumentů. Pokud není zadán žádný argument nebo 1. argument odpovídá některému z řetězců `"--help"`, `"-help"` nebo `"/?"`, vypíše se nápověda a program skončí.
   Pokud je zadán jeden parametr a není to žádost o nápovědu, program předpokládá, že se jedná o název obrázku určeného ke zpracování. Jako rozměry výstupního obrázku se nastaví defaultní hodnota 20x20px.
   Pokud jsou zadány dva parametry, první se chápe jako název obrázku a druhý jako šířka i výška výstupního obrázku.
   Pokud jsou zadány tři parametry, první se chápe jako název obrázku, druhý jako šířka obrázku a třetí jako výška výstupního obrázku.
   Obrázek a sestavovač musí být ve stejné složce.

2. Pomocí funkce `IplImage* cvLoadImage (const char* filename, int flags)` se načte obrázek zadaný parametrem aplikace. Tento obrázek budeme nazývat *originální*.

3. Funkcí `IplImage* cvCreateImage (CvSize size, int depth, int channels )` se vytvoří prázdný obrázek o rozměrech zadaných parametry. Tento obrázek budeme nazývat *střední*.

4. Funkcí `void cvResize (const CvArr* src, CvArr* dst, int interpolation)` se uloží do *středního* obrázku zmenšený *originální* obrázek.

5. Funkcí `int cvSaveImage (const char* filename, const CvArr* image )` se uloží *střední* obrázek na disk. *Střední* obrázky se ukládají do podadresáře `database`.

6. Pomocí `cvCreateImage ()` se vytvoří *malý* obrázek o rozměrech 3x3px, funkcí `cvResize ()` se do něj zmenší *originální* obrázek.

7. Otevře se soubor *database.xml* z podadresáře `database` v režimu přidávání (append). Pokud soubor neexistuje, vytvoří se a přidá se do něj hlavička a kořenový element.

8. Protože aplikace pracuje s XML databází jako s textovým dokumentem, nejprve odmaže zavírací tag kořenového elementu.

9. Do XML databáze se přidá element `picture` s vnořenými elementy:

   - `name` – hodnota: *jméno originálního obrázku*
   - `type` – hodnota: *typ*. Typ obrázku se získává ze jména obrázku tak, že se vezmou všechna písmena od začátku názvu až po první nepísmeno. Předpokládá se, že názvy obrázků budou ve formátu *TypČíslo.přípona*, např. *Car35.bmp* nebo *Face15.jpg*. Typ těchto obrázků je tedy *Car* a *Face*.
   - `pixels` – Pro každý pixel *malého* obrázku se postupně vezmou mohutnosti jednotlivých barev a uloží se v hexadecimální podobě. Pro větší přehlednost jsou informace o jednotlivých pixelech odděleny středníkem.
     Příklad:`<pixels>57582e;5c5c39;35371a;68673f;686a3f;1e210c;59582d;43431e;454726;</pixels>`

10. Do databáze se přidá ukončovací tag kořenového elementu – i když je pravděpodobné, že se během několika vteřin aplikace spustí znovu s jiným obrázkem a tedy tento tag se bude opět mazat, databáze musí po skončení aplikace splňovat výše uvedené XML schéma a být konzistentní.

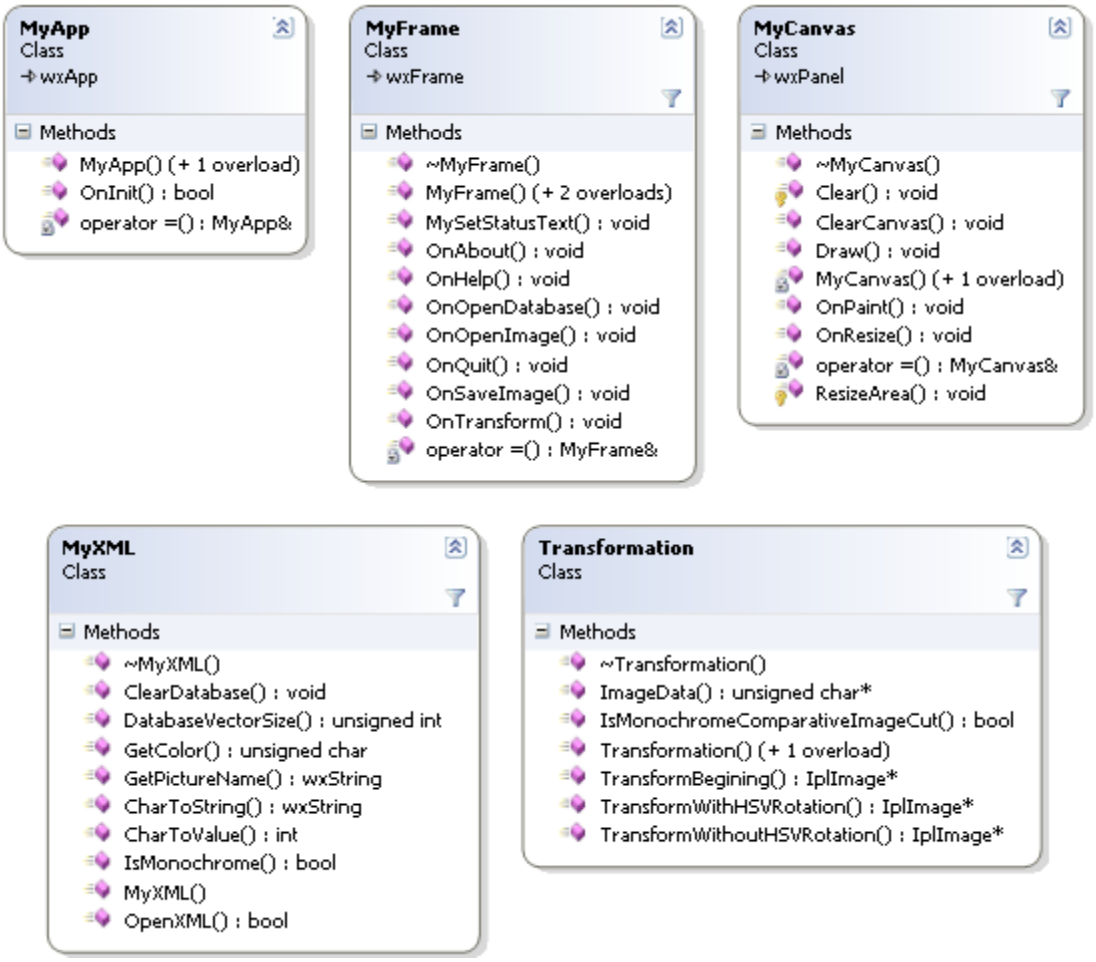
11. Databáze se uloží a zavře. *Originální* obrázek se zavře.

12. Aplikace končí

4.2 sestav.cpp

Aplikace začíná voláním metody `OnInit()` třídy `MyApp`. V této metodě se vytvoří objekt třídy `MyFrame`. Tento objekt představuje viditelnou část aplikace. Nyní již program čeká na reakce uživatele (stisknutí tlačítka v hlavním menu). Podle stisknutého tlačítka se volá příslušná metoda třídy `MyFrame`. Samotné čekání se realizuje zachytáváním událostí.

Na obrázku 4.1 jsou znázorněny všechny třídy a jejich metody.



Obrázek 4.1: Třídy a metody v sestav.cpp

.

$$\sum_{i=1}^{A_h * A_w} \left(0.3 * (R_{Ai} - R_{Mi})^2 + 0.59 * (G_{Ai} - G_{Mi})^2 + 0.11 * (B_{Ai} - B_{Mi})^2\right) \tag{4.1}$$

Popis tříd a jejich metod.

- **MyApp** – třída obsahující celou aplikaci.
  - **MyApp()** – Konstruktor.

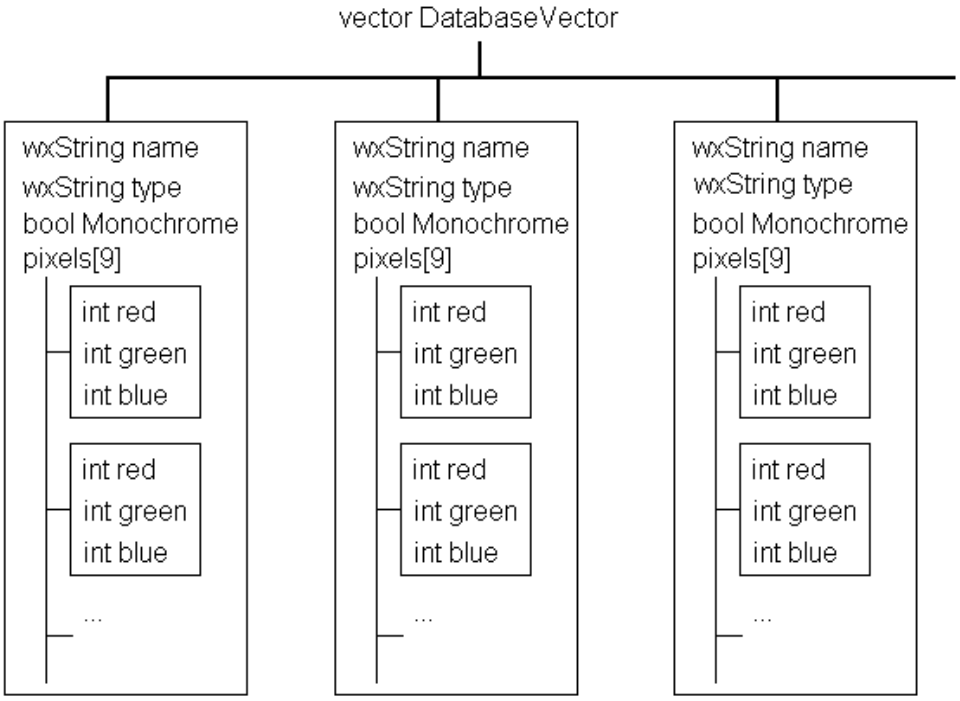
  - **bool OnInit()** – Vytváří instanci třídy MyFrame, nastavuje viditelnost rámu a rám umístí do popředí.

- **MyFrame** – třída spravující rám aplikace, tzn. hlavní menu a status bar .
  - **MyFrame()** – Konstruktor. Vytvoří hlavní menu a status bar. Inicializuje proměnné a vytvoří objekty `m_canvas` z třídy `MyCanvas` a `m_xml` z třídy `MyXML`.
  - **void MySetStatusText(wxString, int i)** – Zobrazí v i-tém políčku status baru text zadaný parametrem.
  - **void OnAbout(wxCommandEvent&)** – Metoda se vyvolává stisknutím tlačítka *About* v hlavním menu. Zobrazí se okno s informacemi o autorovi, názvu, funkci a verzi aplikace.
  - **void OnHelp(wxCommandEvent&)** – Metoda se vyvolává stisknutím tlačítka *Help* v hlavním menu. Zobrazí se okno s postupem na vytvoření mozaiky.
  - **void OnOpenDatabaze(wxCommandEvent&)** – Metoda se vyvolává stisknutím tlačítka *Open Database* v hlavním menu. Zobrazí se standardní dialogové okno pro otevírání souboru. Uživatel vybírá ze souborů s příponou *xml*. Uloží se absolutní cesta adresáře, ve kterém se nachází databáze. Pokud je již načtena jiná databáze, stará databáze se smaže voláním metody `m_xml-> ClearDatabase`. Poté se vytvoří nová databáze voláním metody `m_xml->OpenXML`.
  - **void OnOpenImage(wxCommandEvent&)** – Metoda se vyvolává stisknutím tlačítka *Open Image* v hlavním menu. Zobrazí se standardní dialogové okno pro otevírání souboru. Podporovány jsou formáty *jpg, bmp, png, dib, jpeg, jpe, pbm, pgm, ppm, sr, ras, tiff* a *tif*. Pokud byl již předtím otevřen jiný obrázek, tak se starý obrázek smaže (uvolní se pamět). Načte se nový obrázek a zobrazí se na kreslící plochu pomocí metody `m_canvas->Draw`.
  - **void OnQuit(wxCommandEvent&)** – Metoda se vyvolává stisknutím tlačítka *Exit* v hlavním menu. Uvolňuje se pamět po obrázcích a databázi.
  - **void OnSaveImage(wxCommandEvent&)**– Metoda se vyvolává stisknutím tlačítka *Save Mozaic* v hlavním menu. Pokud je vytvořena mozaika, zobrazí se standardní dialogové okno pro ukládání souborů. Podporovány jsou formáty *jpg*, *bmp* a *png*. Poté se mozaika uloží do zvoleného adresáře.
  - **void OnTransform(wxCommandEvent&)** – Metoda se vyvolává stisknutím tlačítka *Transform* v hlavním menu. Vytvoří se objekt `m_transform` třídy `Transformation`. Konstruktoru se přes parametry předávájí ukazatele na objekt m_frame, načtený obrázek, objekt m_xml a absolutní cesta k adresáři s databází. Dále se volá metoda `m_transform->TransformBegining`, čímž začne vytváření mozaiky.

- **MyCanvas** – třída pro zobrazování obrázků.
  - **MyCanvas(MyFrame*)** – Konstruktor. Vytvoří a inicializuje kreslící plochu – canvas.
  - **void Clear(wxDc&)** – Zobrazí bílý canvas.
  - **void ClearCanvas()** – Vyčistí pamět po obrázcích.

- **void Draw(wxString name)** – Načte z disku obrázek. Pokud se obrázek nevejde do canvasu, zmenší ho při zachování poměru stran. Obrázek vykreslí do canvasu a uloží do paměti.
- **void OnPaint(wxPaintEvent&)** – Metoda se vyvolává při vykreslovací události. Metodou `this->Clear` smaže canvas a znovu ho vykreslí podle obrázku v paměti.
- **void OnResize(wxResizeEvent&)** – Metoda se vyvolává při změně velikosti canvasu. Volá metodu `this->ResizeArea` s novými rozměry canvasu.
- **void ResizeArea(int width, int height)** – Pokud je obrázek na canvasu větší než nová velikost canvasu, velikost obrázku se změní. Obrázek se uloží do paměti.

- **MyXML** – třída pro práci s databází. Obsahuje vector `DatabaseVector` s informacemi o jednotlivých obrázcích v databázi. Schéma vectoru je znázorněno na obrázku 4.2.



Obrázek 4.2: Schéma vectoru s informacemi o obrázcích

- **MyXML(MyFrame*)** – Konstruktor.
- **void ClearDatabase()** – Smaže všechny záznamy v `DatabaseVector`.
- **unsigned char GetColor(int picture, int pixel, int channel)** – V `DatabaseVector` vyhledá obrázek na pozici `picture`, jeho pixel na pozici `pixel` a barvu `channel`
- **wxString GetPictureName(int i)** – V `DatabaseVector` vyhledá jméno i-tého obrázku.

- **wxString CharsToString(char*)** – Metoda slouží k převodu řetězce `char*` na řetězec `wxString`. Převod se provádí postupnou konverzí přes datový typ `wxChar`.
- **int CharToValue(wxChar)** – Metoda slouží k převodu hexadecimálního znaku v datovém typu `wxChar` na jeho hodnotu v desítkové soustavě.
- **bool IsMonochrome(int i)** – V `DatabaseVector` vyhledá, zda je i-tý obrázek jednobarevný.
- **bool OpenXML(wxString path)** – Metoda slouží k naplnění `DatabaseVector` z xml souboru popisující databázi obrázků. Načte xml soubor určený parametrem a přejde na první element `picture`. Do struktury popisující obrázek (viz obr. 4.2) se načte jméno, typ a pomocí metody `this->CharToValue` se vypočítají mohutnosti barev jednotlivých pixelů. Zároveň se vypočítává, zda se jedná o jednobarevný obrázek. Za jednobarevný se považuje takový obrázek, u něhož rozdíl mohutností stejné barvy nepřesahuje definovanou mez, tedy splňuje výraz 4.2. Tato struktura se přidá na konec vectoru `DatabaseVector` a pokračuje se načítáním dalšího elementu z xml souboru.

$$max((R_{max} - R_{min}), (G_{max} - G_{min}), (B_{max} - B_{min})) < Treshold \tag{4.2}$$

- **Transformation** – třída pro sestavení mozaiky.
  - **Transformation()** – Konstuktor. Provádí inicializaci proměných.
  - **unsigned char* ImageData(IplImage* image, int row, int column, int channel)** – Metoda vrací ukazetel na barvu `channel` pixelu na pozici [`column,row`] v obrázku `image`. Ukazatel se vrací z důvodu umožnění zápisu i čtení dat.
  - **bool IsMonochromeImageCut(int row, int column)** – Metoda zjišťuje zda výsek obrázku začínající na souřadnicích [*column,row*] je jednobarevný. Za jednobarevný se považuje obrázek splňující výraz 4.2.
  - **IplImage* TransformBegining()** – Pokud není načtena databáze a otevřen obrázek, metoda končí. Zjišťují se rozměry prvního obrázku v databázi. Pokud mají obrázky v databázi rozdílné rozměry, muže dojít k nepovolenému přístupu do paměti a aplikace skončí chybou. Dále se provádí změna rozměrů obrázku na výslednou mozaiku(vzorce 4.3 a 4.4) a na porovnávání(vzorce 4.5 a 4.6). Do těchto obrázku se zmenší originální obrázek. Poté je uživatel dotázán, zda chce provést vytvoření mozaiky za pomoci HSV rotací barev obrázků z databáze. Podle odpovědi se volá metoda `this->TransformWithHSVRotation` nebo `this->TransformWithoutHSVRotation`. Tyto metody vrací ukazatel na hotovou mozaiku. Tento ukazatel vrací i tato metoda.

$$Width = (OriginalImgWidth/SmallImgW) * SmallImgW \tag{4.3}$$
$$Height = (OriginalImgHeight/SmallImgH) * SmallImgH \tag{4.4}$$

4.3, 4.4 – Protože všechny hodnoty jsou celočíselné, dochází k zaokrouhlování. Rozměry výsledné mozaiky jsou tedy rozměry originálního obrázku zaokrouhleny na nejbližší menší násobek rozměrů malého obrázku z databáze.

$$Width = (OriginalImgWidth/SmallImgWidth) * 3 \tag{4.5}$$
$$Height = (OriginalImgHeight/SmallImgHeight) * 3 \tag{4.6}$$

4.5, 4.6 – Porovnávací obrázek slouží k porovnávání barev mezi originálním obrázkem a 3x3 popisem v `DatabaseVector` (viz obr. 4.2). Proto rozměry musí být *počet dlaždic* * 3.

– **IplImage* TransformWithoutHSVRotation()** – Metoda na samotné sestavení mozaiky. Postupně se prochází obrázek pro porovnávání po výsecích 3x3px a pro každý výsek se prochází celý `DatabaseVector`. Za pomoci metod `m_xml->GetColor` a `this->ImageData` se podle rozdílu barev(viz vzorec 2.1) vybírá nejpodobnější obrázek. Díky metodě `m_xml->GetPictureName` známe jméno nejpodobnějšího obrázku, tento obrázek otevřeme a zkopírujeme na příslušné pozice obrázku s výslednou mozaikou. Celý postup se opakuje pro všechny výseky obrázku pro porovnávání. Po průchodu celého porovnávacího obrázku se obrázek s mozaikou uloží na disk, vykreslí se pomocí metody `m_canvas->Draw` a ukazatel na obrázek s mozaikou se vrátí jako návratová hodnota.

– **IplImage* TransformWithHSVRotation()** – Postupně se prochází obrázek pro porovnávání po výsecích 3x3px, metodou `this->IsMonochromeImageCut` se zjišťuje, zda je výsek jednobarevný a pro každý výsek se prochází celý `DatabaseVector`. Za pomoci metod `m_xml->GetColor` a `this->ImageData` se podle rozdílu barev(viz vzorec 2.1) vybírá nejpodobnější obrázek a podle rozdílu intenzit (viz vzorec 4.1) se vybírá nejpodobnější obrázek v barvách šedi. Pokud není výsek porovnávacího obrázku jednobarevný nebo je rozdíl barev menší než předem definovaná mez, postupuje se stejně jako u transformace bez HSV rotace barev. V opačném případě se používá o něco složitejší postup a proto si zaslouží přehlednější přístup:

1. Do paměti se z databáze načte obrázek s nejmenším rozdílem intenzit.
2. Vytvoří se nový obrázek a zkopíruje se do něho část originálního obrázku odpovídající pravě zpracovávanému výseku obrázku pro porovnávání. Tento obrázek má stejné rozměry jako obrázek načtený v předchozím kroku.
3. Oba obrázky se převedou na barevný model HSV.
4. Je potřeba předejít problému s rotacemi barev popsaném v části Teorie-Barevné modely-HSV.
   U obou obrázků se zjistí nejmenší a největší hodnota složky *Hue*.
5. Pokud je minimum v rozsahu 0–*práh jednobarevnosti* a maximum v rozsahu (180-*práh jednobarevnosti*) – 179, převedou se u obou obrázků hodnoty barvy přesahující 90 na hodnotu -(180-*původní hodnota*).
6. Spočítá se aritmetický průměr barev obou obrázků.
7. Tyto průměry se od sebe odečtou, vznikne rozdíl průměrů.
8. V obrázku načteném v 1. kroku se u všech pixelů přičte ke složce *hue* rozdíl průměrů.
9. Oba obrázky se převedou na barevný model RGB.
10. První obrázek se zkopíruje na příslušnou pozici obrázku s výslednou mozaikou.

Celý postup se opakuje pro všechny výseky obrázku pro porovnávání. Po průchodu celého porovnávacího obrázku se obrázek s mozaikou uloží na disk, vykreslí se pomocí metody `m_canvas->Draw` a vrací se ukazatel na obrázek s mozaikou.

Kapitola 5

Ovládání

5.1 zmensi.exe

Nejprve je potřeba vytvořit databázi obrázků. Obrázky (libovolné velikosti), program *zmensi.exe* a *skript* (viz dále) musí být ve stejném adresáři. Aplikace *zmensi* do podadresáře *database* uloží obrázek zadaný 1. parametrem zmenšený na velikost zadanou 2. (šířka) a 3. (výška) parametrem a přidá informace do xml souboru. Například:
Windows: `zmensi.exe obrazek.jpg 20 20`
Linux: `./zmensi obrazek.jpg 20 20`

Pro vytvoření databáze, ze které by vznikaly přijatelné mozaiky, jsou potřeba tisíce obrázků. Je zřejmé, že manuální přidávání po jednom obrázku by bylo zdlouhavé. Proto jsou připraveny skripty, pomocí kterých se vytvoří databáze jedním kliknutím.:
Windows (*skript.bat*):

```
@ECHO OFF
mkdir database
FOR %%obrazek IN (*.jpg *.jpeg *.bmp)  DO zmensi.exe %%obrazek 20 20
@ECHO ON
```

Linux (*skript.sh*):

```
#!/bin/bash
mkdir database
for obrazek in *.jpg *.bmp *.jpeg; do
  ./zmensi $obrazek 20 20
done
```

Přenositelný skript - Python (*skript.py*):

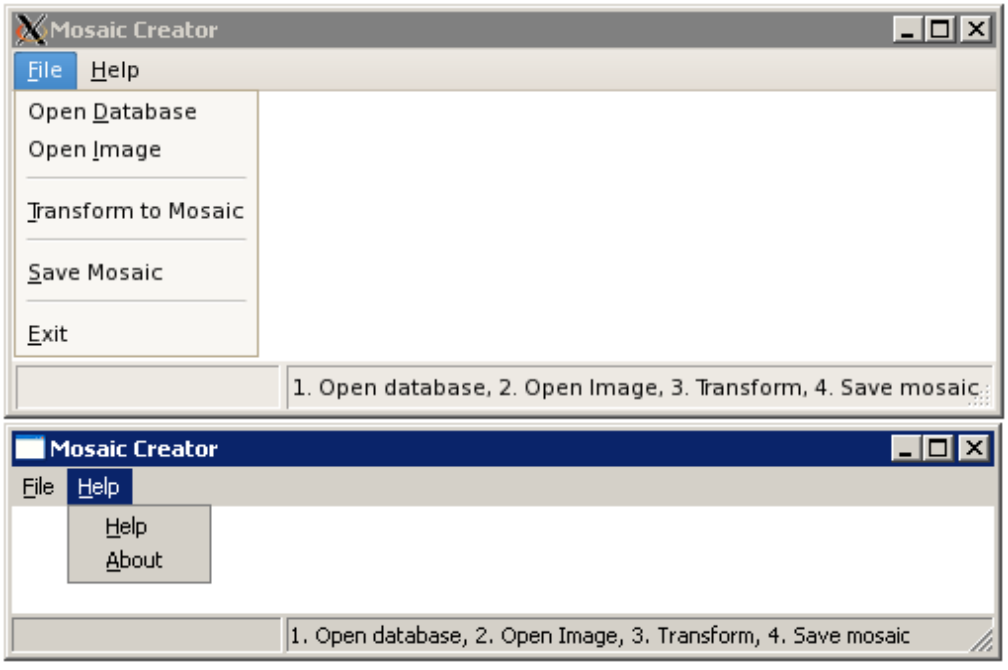
```
import glob , os , sys
dir = "database"
if not os.path.isdir(dir):
  os.mkdir(dir)
filelist = glob.glob('*.jpg') + glob.glob('*.bmp')
for obrazek in filelist:
  if os.name == "posix":
    os.system(".\zmensi "+ obrazek + " 20 20")
  else:
    os.system("zmensi.exe " + obrazek + " 20 20")
```

Všechny tyto skripty nejprve vytvoří složku *database* a následně spouští aplikaci *zmensi.exe* pro všechny soubory s příponami jpg, jpeg a bmp z aktuálního adresáře. Protože se jedná o textové soubory, není problém do skriptu přidat další grafické formáty (png, tiff) či změnit rozměry výsledných obrázků (defaultně 20x20px).

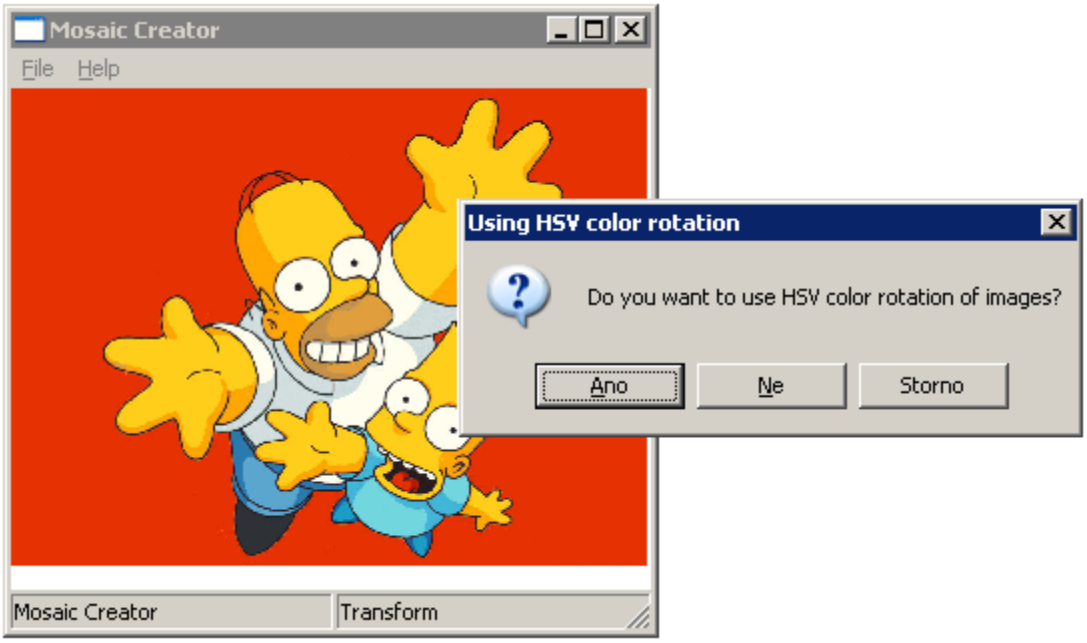
5.2 sestav.exe

Po spuštění aplikace se zobrazí úvodní obrazovka podobná jako na obrázku 5.1. Grafická podoba závisí na aktuálním nastavení systému. Aplikace se ovládá přes tlačítka v hlavním menu. Stisknutím tlačítka *File–>Open Database* se zobrazí standardní okno pro otevření souboru. Uživatel vybírá XML soubor z adresáře s databází. Po stisknutí tlačítka *File–>Open Picture* se zobrazí standardní okno pro otevření souboru, uživatel tentokrát vybírá obrázek, který bude předobrazem mozaiky. Tlačítko *File–>Transform* zobrazí okno s dotazem, zda chce uživatel při sestavování mozaiky využívat HSV rotaci barvy u obrázků z databáze (viz obr. 5.2). Samotné sestavování mozaiky může trvat i několik minut. Tato doba závisí na výkonu počítače, počtu obrázků v databázi, velikosti obrázku a zvoleném způsobu sestavování. Výsledná mozaika se zobrazí v aplikaci. Od té chvíle lze po stisknutí tlačítka *File–>Save Mozaic* a výběru souboru ve standardním dialogovém okně mozaiku uložit. Podporované formáty pro ukládání jsou *bmp, jpg a png*. Uživatel může vybrat jinou databázi či obrázek a vytvářet další mozaiky nebo program ukončit tlačítkem *File–>Exit*. Položka *Help* v hlavním menu nabízí dvě tlačítka, *About* a *Help*. Tlačítko *About* zobrazí okno s informacemi o autorovi, názvu a verzi programu. *Help* zobrazí okno se stručnou nápovědou.

Celý program lze také ovládat pomocí klávesových zkratek. Jednotlivé zkratky odpovídají podtrženým písmenům v menu.



Obrázek 5.1: Úvodní obrazovka s otevřeným menu. Nahoře systém Linux, dole Windows.



Obrázek 5.2: Okno z dotazem, zda chce uživatel využít HSV rotace barev.

Obrázek 5.3: Výsledná mozaika.

Kapitola 6

Výsledky

Tato kapitola hodnotí dosažené výsledky a zabývá se dalšími možnými vylepšeními.

6.1 Dosažené výsledky

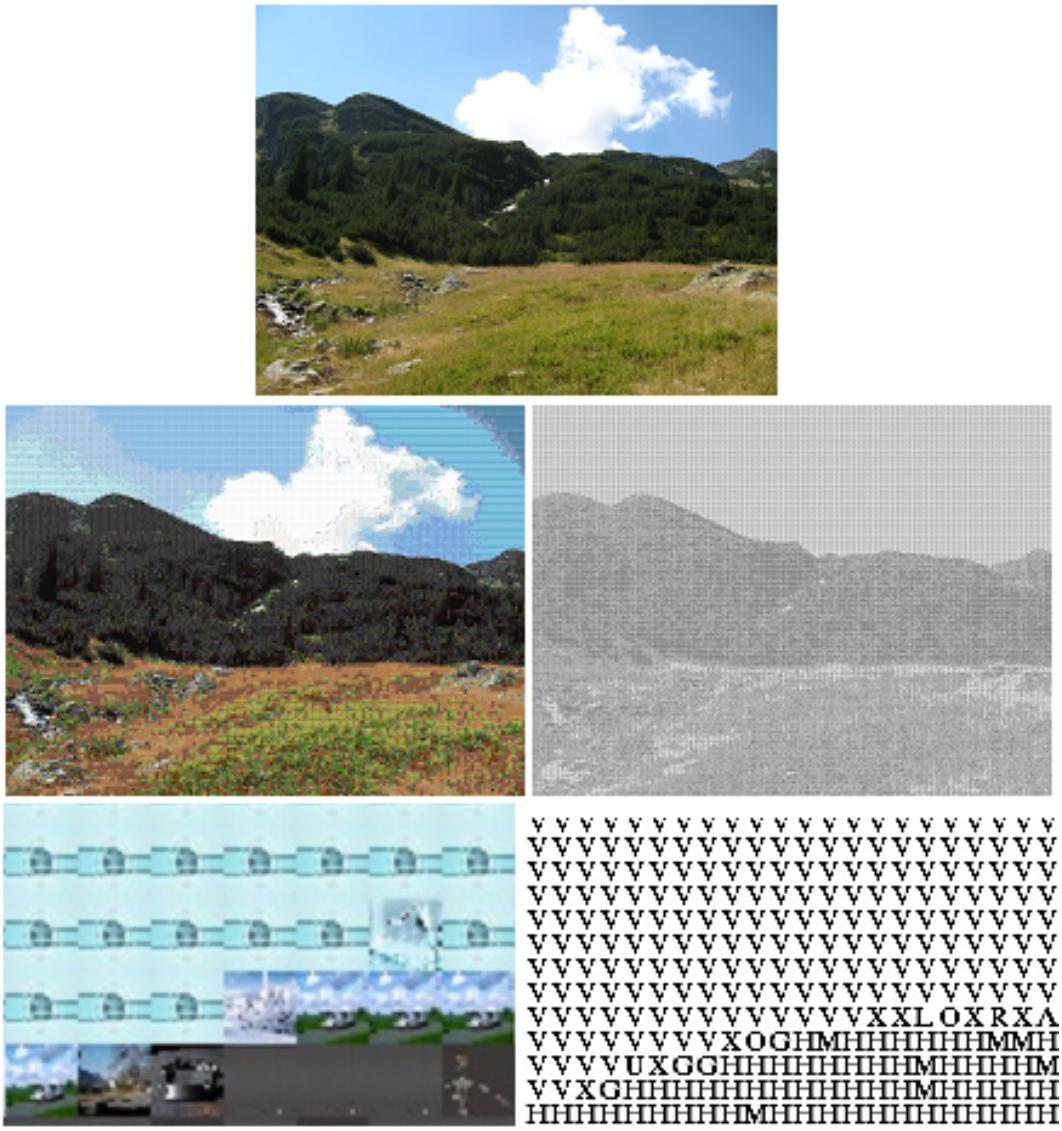
Systém navržený v kapitole *Návrh* se podařilo implementovat. Výsledné mozaiky odpovídají předloze jak svou barevností tak i zachováváním hran. Vytváření mozaiky muže trvat i několik minut. Při několikanásobném zvětšení databáze by zde popsaný koncept databáze byl nevyhovující. Možnosti vylepšení tohoto konceptu budou popsány v části *Možná vylepšení*.

Zde prezentované mozaiky (obrázky 6.1, 6.2 a 6.3) byly vytvořeny z databáze obsahující přes 4000 obrázků. Bohužel tyto obrázky mají velmi malou rozmanitost. ASCII artové mozaiky byly vytvořeny z 28 obrázků velkých písmen anglické abecedy. Kvůli absenci znaků jako mezera, čárka či podtržítko se špatně zobrazují světlé části.

Očekávání, že HSV rotace barev bude vhodná pouze pro jednobarevné dlaždice, bylo chybné. Potvrdilo se sise, že HSV rotace různobarevných obrázků podle předpokladu může vytvářet podivné barevné kombinace, avšak tyto nehodící se dlaždice jsou ve velké menšině oproti vhodně změněným dlaždicím. Celkový dojem z mozaiky působí tedy lepé při HSV rotaci všech obrázků.

Bohužel systému chybí beta-testování. Programy byly zkoušeny pouze na školním serveru "Merlin"(CentOS 5.3 64bit Linux, wxWidgets 2.8.9) a na autorovu notebooku (Win XP SP3, Visual Studio 2008 Profesional Edition, wxWidgets 2.8.9 a OpenCV 1.1pre1). Na serveru Merlin program funguje bez jakýchkoliv problémů, na notebooku nesprávně fungují standardní dialogová okna. Při startu aplikace se zobrazí chybové okno, dialogová okna nedokáží zobrazit "Tento počítač"ani obnovovat výpis souborů při změně formátu a při ukončení aplikace zůstane v systému neukončený proces. Tato chyba je nejspíše způsobena špatnou verzí knihovny *wxWidgets*.

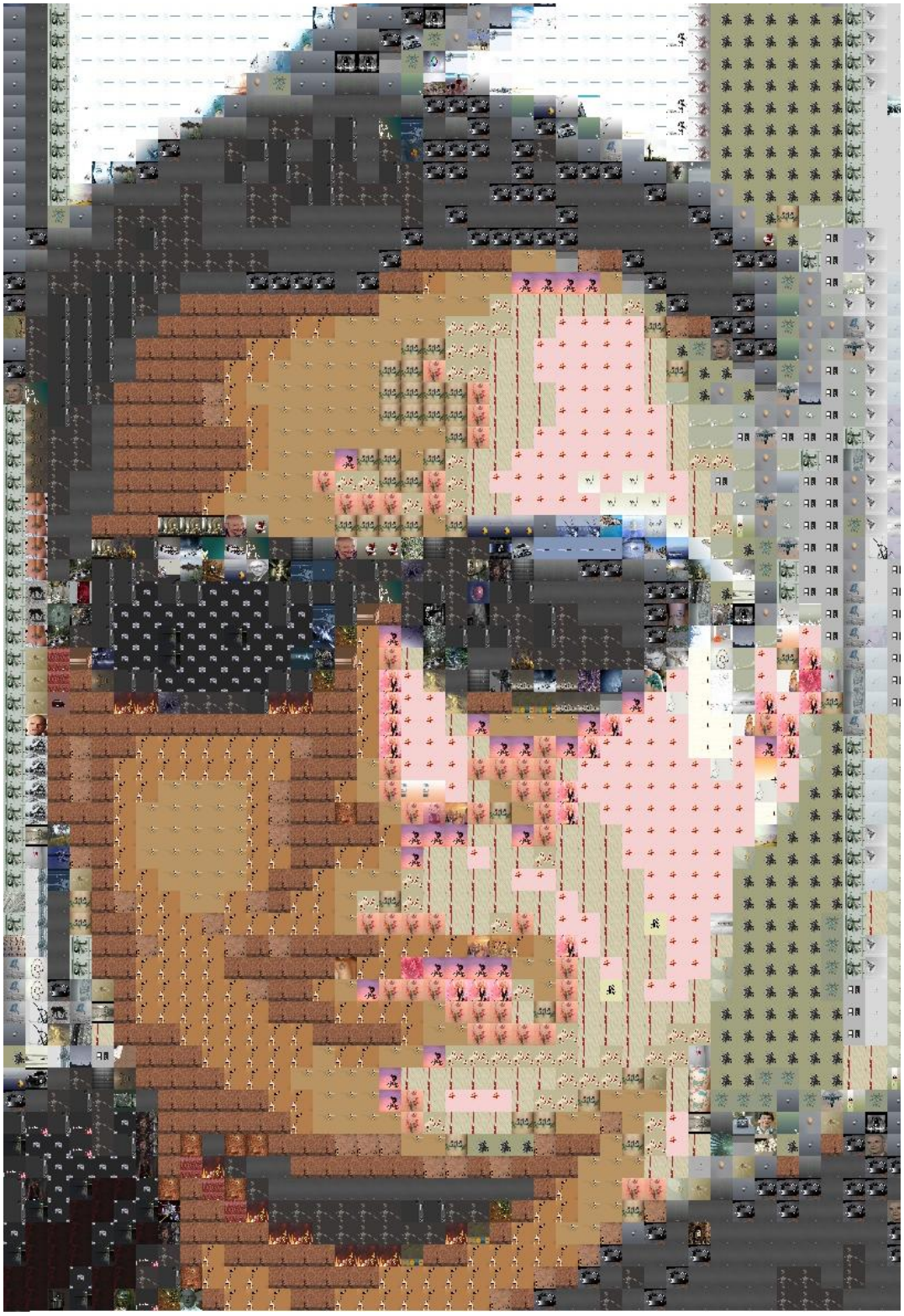
Další nedostatek se tyká databáze obrázků, která ale není předmětem této práce. Obrázky v databázi musí být rozmanité. Například sto obrázků zobrazující zelenou louku a nad ní modrou oblohu pouze zpomaluje vytváření mozaiky. Ačkoliv by mozaika měla zachovávat hrany pomocí různobarevných obrázků, největší část mozaiky je tvořena dlaždicemi s jednou dominantní barvou. Tyto jednobarevné dlaždice jsou zároveň potřebné pro HSV rotaci barvy. Proto by měly být v databázi v dostatečném počtu.



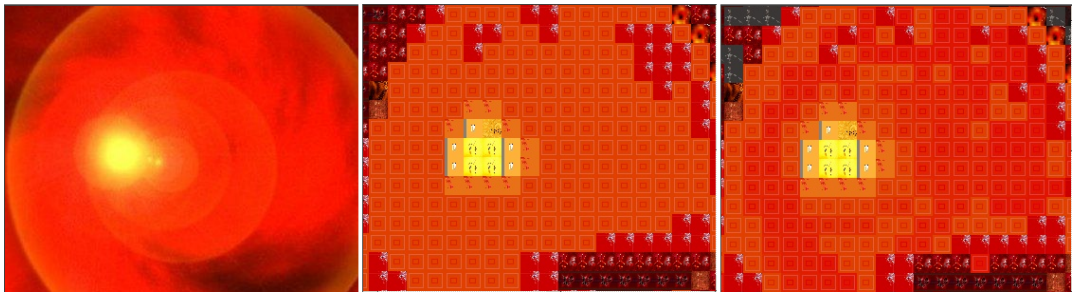
Obrázek 6.1: Ukázka výsledků. Po směru hodinových ručiček: Originální obrázek, ASCII art, detail ASCII artu, detail mozaiky a mozaika.

6.2 Možná vylepšení

Aplikace *zmensi.exe* přistupuje k XML souboru s popisem databáze. Při spouštění aplikace ze skriptu se musí čekat na otevření a zavření souboru s databází. Tyto operace provádění skriptu velmi zdržují a proto by bylo vhodné aplikaci upravit tak, aby mohla na svůj vstup dostat seznam obrázků. Tím by se zajistilo, že soubor by se otevřel pouze pro první obrázek a zavřel by se až po posledním obrázku.
Aplikace *sestav.exe* ukládá informace o databázi obrázků do lineárního seznamu. Vhodnější uspořádání by byl hierarchický strom (viz obr. 6.4). Na nejvyšší úrovni jsou základní barvy, na prostřední úrovni odstíny základních barev a na nejnižší úrovni se již nacházejí informace o jednotlivých obrázcích.



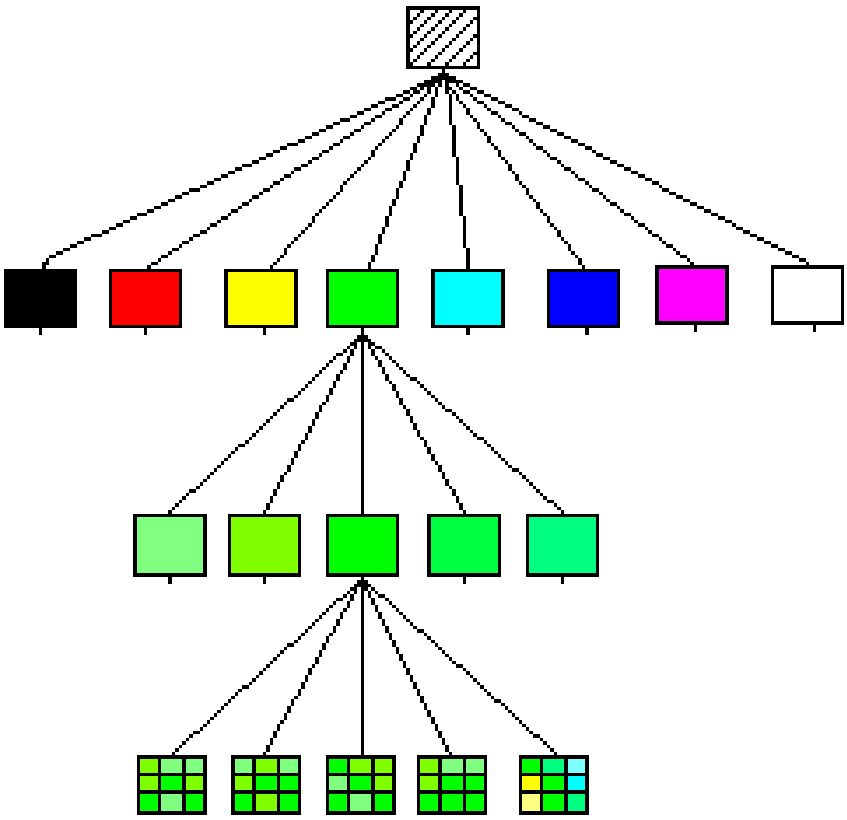
Obrázek 6.2: Ukázka výsledné mozaiky.



Obrázek 6.3: Ukázka HSV rotace barev. Vlevo originální obrázek, uprostřed mozaika bez HSV rotace, vpravo mozaika s HSV rotací.

Další část, která by mohla být řešena lépe je zobrazování výsledných mozaiek. Rozměrné mozaiky mohou mít i desítky MB a práce s nimi není nejrychlejší. Zvláště změna jejich rozměrů trvá dlouho. Možným řešením by bylo uložení mozaiky ve více rozměrech a pracovat s mozaikou velikostně nejbližší aktuálně otevřenému oknu aplikace.

Při zobrazování ASCII artu je viditelné výrazné moire. To by se dalo odstranit buď pomocí antialiasingu nebo rozmazáním mozaiky. Protože se předpokládá, že uživatel mozaiku uloží, ošetřování moire by zbytečně zpomalovalo aplikaci.



Obrázek 6.4: Princip uložení informací o obrazcích v databázi jako hierarchický strom.

Kapitola 7

Závěr

Cílem této práce bylo navržení a implementování programu na vytváření obrazových mozaiek. Tento cíl se mi podařilo splnit.

První bod zadání je spojil s objasněním teorií nutných k pochopení následujících kapitol. Informace jsem získal na základě dostupných materiálů. Zmiňuji se zde o principu uložení obrázků v počítačích, barevných modelech, změnách rozměrů obrázků a možnostech jejich porovnávání. V závěru této kapitoly se věnuji technice ASCII art a technologii XML.

Druhému bodu zadání, tedy návrhu systému tvorby mozaiek, jsem věnoval kapitola číslo 3. Tato kapitola je rozdělena na tři části. První část se zabývá podrobným návrhem databáze. Prostřední část popisuje program pro tvorbu databáze. Poslední část navrhuje program na sestavování mozaiek.

Následující bod zadání požaduje implementaci systému. Zpracování tohoto bodu je rozděleno do dvou samostatných kapitol. V kapitole Implementace jsem podrobně popsal programy *zmensi.cpp* a *sestav.cpp*. Díky funkčnosti těchto programů byl splněn hlavní cíl bakalářské práce. V kapitole Ovládání je názorně popsáno jejich ovládání.

V první části předposlední kapitoly jsou shrnuty dosažené výsledky s obrazovou ukázkou některých mazaiek. V druhé části jsou navrženy postupy pro další vylepšení. Tyto návrhy se týkají především zrychlení funkčnosti aplikací. Domnívám se, že ukládání informací o databázi v struktuře "hierarchický strom"by mohlo vést až k stonásobnému urychlení procesu tvorby mozaiky. Tato vylepšení nebyla implementována buď v důsledku nedostatečného návrhu či překročení rozsahu zadání bakalářské práce.

Díky zpracování tématu mé bakalářské práce jsem se zdokonalil v návrhu a realizaci objektově orientovaného programu. Vylepšil jsem si své programátorské dovednosti při samostatném tvoření takto rozsáhlé aplikace. Zároveň jsem si prohloubil základní znalosti v oblasti počítačové grafiky.

Věřím, že program vytvořen v rámci mé bakalářské práce najde své uplatnění.

Literatura

[1] Kršek, P.; M.Španěl: Základy počítačové grafiky - Redukce barevného prostoru. , 2007.

[2] WWW stránky: Co se děje, když se obrázky zmenšují a zvětšují. http://www.paladix.cz/clanky/co-se-deje-kdyz-se-obrazky-zmensuji-a-zvetsuji.html.

[3] WWW stránky: Wikipedia - Mozaic. http://en.wikipedia.org/wiki/Mozaic.

[4] WWW stránky: Wikipedie - ASCII art. http://cs.wikipedia.org/wiki/ASCII_art.

[5] WWW stránky: Wikipedie - Barevný model. http://cs.wikipedia.org/wiki/Barevn%C3%BD_model.

[6] WWW stránky: Wikipedie - HSV. http://cs.wikipedia.org/wiki/HSV.

[7] WWW stránky: Wikipedie - Mozaika. http://cs.wikipedia.org/wiki/Mozaika.

[8] WWW stránky: Wikipedie - RGB. http://cs.wikipedia.org/wiki/RGB.

[9] WWW stránky: Wikipedie - Vitraj. http://cs.wikipedia.org/wiki/Vitraj.

[10] WWW stránky: XML Schéma. http://www.kosek.cz/xml/schema/wxs.html.