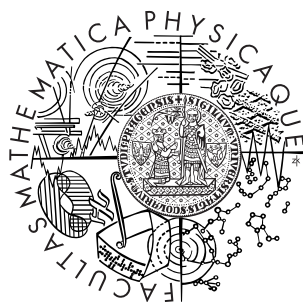


Univerzita Karlova v Praze  
Matematicko-fyzikální fakulta

## BAKALÁŘSKÁ PRÁCE



Marek Novotný

### Grafické filtry pro projekt AGE

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Pavel Ježek

Studijní program: Informatika

Studijní obor: Programování

2010

Na tomto místě bych chtěl poděkovat vedoucímu své bakalářské práce panu Mgr. Pavlu Ježkovi za cenné odborné rady a připomínky a za čas, který strávil během konzultací.

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

V Praze dne 3. srpna 2010

Marek Novotný

# Obsah

Úvod	6
<b>1 Analýza</b>	<b>8</b>
1.1 Nástroje pro implementaci	8
1.2 Repräsentace a práce s barvou	8
1.2.1 Požadavky na implementaci	9
1.2.2 Možnosti	9
1.2.3 Závěrečné rozhodnutí	10
1.3 Repräsentace a práce s rastrem	12
1.3.1 Možnosti	12
1.3.2 Závěrečné rozhodnutí	13
1.4 Panorámata	13
1.4.1 Možnosti	14
1.4.2 Závěrečné rozhodnutí	15
1.5 Filtry	16
1.5.1 Transformační filtry	16
1.5.2 Konvoluční filtry	20
1.5.3 Barevné filtry	23
1.6 GUI	25
<b>2 Implementace</b>	<b>26</b>
2.1 Základní návrh	26
2.2 Barvy	28
2.3 Bitmapy	29
2.3.1 Změna velikosti	31
2.3.2 Paralelismus na bitmapě	32
2.4 Filtry a jejich nástroje	33
2.4.1 Nejobecnější třídy filtrů	34
2.4.2 Barevné filtry	35
2.4.3 Transformační filtry	36
2.4.4 Konvoluční filtry	37

2.4.5	Nástroje . . . . .	39
2.5	Panoramata . . . . .	40
2.6	GUI . . . . .	41
2.6.1	Prostředí . . . . .	42
2.6.2	Editace filtrů . . . . .	43
2.6.3	Editace panoramat . . . . .	44
<b>3</b>	<b>Srovnání s konkurencí</b>	<b>46</b>
3.1	Gimp . . . . .	46
3.2	Adobe Photoshop . . . . .	47
3.3	Shrnutí . . . . .	48
	<b>Závěr</b>	<b>49</b>
	<b>Literatura</b>	<b>51</b>

Název práce: Grafické filtry pro projekt AGE  
Autor: Marek Novotný  
Katedra (ústav): Katedra distribuovaných a spolehlivých systémů  
Vedoucí bakalářské práce: Mgr. Pavel Ježek  
E-mail vedoucího: pavel.jezek@d3s.mff.cuni.cz

Abstrakt: Předložená práce se zabývá implementací knihoven pro projekt AGE. Knihovny obsahují implementaci transformačních a kovolučních filtrů a filtrů pro práci s barvou. Naleznou se zde konvoluční filtry ve dvou variantách a to ve formě nízkofrekvenční a vysokofrekvenční. V knihovnách je dále zahrnuta tvorba jednorozměrných panoramat z částečně se překrývajícími rastrovými obrázky se zasazením do struktury objektů projektu AGE. K prezentaci jednotlivých funkcionalit knihoven byl vytvořen grafický editor, v němž se dají vytvářet zmíněná panoramata a aplikovat grafické filtry na vektorový či rastrový obrázek.

Klíčová slova: grafický, filtr, panorama, editor, AGE

Title: Graphical Filters for the AGE Project  
Author: Marek Novotný  
Department: Department of Distributed and Dependable Systems  
Supervisor: Mgr. Pavel Ježek  
Supervisor's e-mail address: pavel.jezek@d3s.mff.cuni.cz

Abstract: The goal of this thesis was to develop a set of libraries for the AGE project. The libraries implement common transform, convolution (both high and low frequency) and color filters. The libraries also include an implementation of one-dimensional panoramas of overlapping images. All of the functionality is designed to seamlessly fit into the AGE project class structure. In order to present the main results of the thesis a simple graphical editor was created as part of the thesis. The editor supports generation of panoramas as well as application of all the implemented filters to bitmap and vector images.

Keywords: graphical, filter, panorama, editor, AGE

# Úvod

Cílem bakalářské práce je vytvořit skupinu knihoven pro projekt AGE (Advanced Graphics Editor), které by implementovaly grafické filtry a další funkční prvky využívající tyto filtry.

Grafický filtr je názvem pro funkci na grafických prvcích jako je vektorový či rastrový obrázek. Je dán vztah  $Y = f(X)$ . Zobrazení  $f$  je zmiňovaný grafický filtr a  $X$  vstupní obrázek.  $Y$  představuje transformovaný obrázek, jež vznikl aplikací filtru na zdrojový obrázek. Transformací se rozumí například změna barvy, geometrická transformace obrázku atd., jež závisí na implementaci a aktuálním nastavením filtru. V průběhu transformace může nastat situace, kdy bude potřeba převést vektor na bitmapu a filtr aplikovat na výsledné bitmapě. To později zapříčiní rozdělení filtrů na kategorie vektorové a rastrové. Filtry se budou dále dělit na nízkofrekvenční, vysokofrekvenční, transformační a filtry pro práci s barvou. Tato problematika však bude podrobněji rozebrána v dalších kapitolách.

Dalším prvkem, který by měly knihovny zahrnovat, je tvorba panoramat. Panoramatem se rozumí pohled na nějaký objekt či krajinu v širokém zorném poli, které může dosahovat hodnoty až 360 stupňů. Většina fotoaparátů nemá možnost fotit ve velmi širokém zorném poli, a proto se nabízí jediná možnost nafotit více snímků, které se pak určitým způsobem složí.

Dále by knihovny měly obsahovat tvorbu opakovatelných grafických textur. Pod tímto pojmem je myšlena transformace bitmapového obrázku tak, aby se transformovaný obrázek při jeho horizontálním a vertikálním opakování jevil jako textura, která neobsahuje skokový přechod.

V neposlední řadě bude potřeba rozmyslet a naimplementovat reprezentaci rastrového (bitmapového) obrázku, na který se budou aplikovat předchozí body funkcionality.

Zmiňovaný projekt AGE je integrovaným prostředím pro grafickou tvorbu zejména v oblastech vektorové, rastrové a prostorové grafiky, jehož hlavní myšlenkou je poskytovat uživateli komplexní grafické prostředí, jehož možnosti uspokojí jeho veškeré potřeby, a nebude tak nucen použít jiný grafický nástroj s úzkým spektrem možností. Jelikož AGE zatím existuje

pouze ve vývojové fázi, bylo nutné vytvořit aplikaci pro prezentaci knihoven. Tato aplikace využívá i další knihovny, které jsou součástí Bakalářské práce Jana Šebetovského [1] navázané na Bakalářskou práci Tomáše Hercega [2], která není přímo využívána, ale některé její části jsou nutné pro chod knihoven z práce [1]. V textu bude zmíněno využití jednotlivých částí těchto knihoven.

# Kapitola 1

## Analýza

V této kapitole budou popsány myšlenky a rozhodování autora při návrhu této práce. Budou zde zobrazeny klady a zápory jednotlivých variant řešení problémů a po celkovém zhodnocení dojde k výběru jedné z variant a opodstatnění jejího výběru.

### 1.1 Nástroje pro implementaci

Aplikace a knihovny této práce budou napsány pomocí vývojové platformy .NET verze 3.5 a programovacího jazyka *C#* verze 3.0. K tomuto rozhodnutí vedl fakt, že autor této práce má se zmíněnými implementačními nástroji v porovnání s jinými možnostmi největší zkušenosti. Dále je bráno v potaz, že výsledná aplikace AGE bude napsána taktéž pomocí těchto nástrojů.

### 1.2 Reprezentace a práce s barvou

Před úvahami nad tím, jak by měla být implementována barva, je nutné si položit otázku, co to vlastně barva je. Ve své podstatě se jedná o vlnovou délku elektromagnetického záření, které lidé nazývají viditelným světlem. Rozsah vlnové délky viditelného světla se pro informaci pohybuje mezi 400-750 nm, přičemž modré barvě přísluší nejkratší vlnová délka, červené zase ta nejdelší a mezi tím je rozprostřeno celé spektrum barev.

Jak je dobře známé, lidské oko není dokonalé a ne všechny části pro člověka viditelného spektra vnímá se stejnou intenzitou. Některé barvy či odstíny barev jsou oproti jiným potlačeny. V literatuře [3] je možno se dostat k podrobnějším informacím, kde je viditelné spektrum lidského oka označováno jako *Lab Color*. Jelikož při používání dnešních technických zařízení nelze pracovat přímo s vlnovou délkou barvy, je možno se ve zmíněné literatuře dočíst, jaké

existují možnosti reprezentace barvy v dnešní technice, a to pomocí reprezentace RGB a CMYK. V této práci budou tyto reprezentace označeny jako barevné prostory. I tyto barevné prostory nebudou dokonalým vyjádřením barvy, jelikož nepokryjí celé viditelné spektrum lidského oka a navíc se nebudou moci mezi sebou převádět beze ztrát.

### 1.2.1 Požadavky na implementaci

Jaké požadavky mohou být kladeny na implementaci barev? Jelikož s barvou bude pracovat velké množství entit jako je filtr nebo panorama, bude důležité, aby byla barva vyjádřena pouze jedním barevným prostorem. V případě, že by reprezentací bylo víc, potom by se stalo nutností vytvářet navázané prvky víckrát. Na tento jediný prostor barev bude kladen důraz, aby se do něj převáděly bezeztrátově ostatní prostory. Dále bude nutné, aby daný prostor nekladl veliké paměťové a implementační nároky.

### 1.2.2 Možnosti

Nyní budou nastíněny možnosti pro výběr barevného prostoru.

#### Barevný prostor RGB

Tento barevný prostor je definován pomocí tří barev; červené (Red), zelené (Green) a modré (Blue). Pomocí skládání těchto tří složek se dostávají ostatní barvy spektra prostoru. V praxi bývá tento prostor implementován tak, že pro jeden barevný kanál se vyhradí jedno osmibitové celé číslo. Čím je číslo větší, tím se dostane barva s větším podílem jasu do daného kanálu barvy.

Mezi největší klady barevného prostoru RGB patří, že je asi nejrozšířenější barevnou reprezentací ve výpočetní sféře. Těžko se najde aplikace pro práci s grafikou, která by tento prostor nevyužívala. Při použití tohoto barevného prostoru částečně či úplně odpadla implementace, jelikož je tento prostor již hotov v prostředí .NET, které bude využíváno. Dále je nutno zmínit, že drtivá většina témat literatur a různých článků provádí své úvahy nad tímto prostorem. Jedinou, avšak důležitou nevýhodou by bylo, že tento barevný prostor nepokrývá spektrum lidského oka a konverze z jiných barevných prostorů by byla ztrátová.

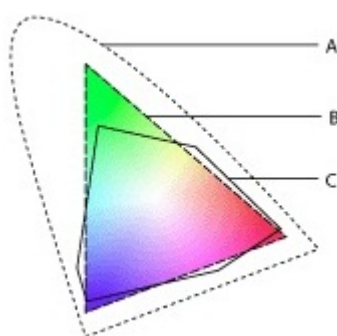
#### Barevný prostor CMYK

Barevný prostor založený na subtraktivním míchání barev je složen ze čtyř barevných složek, a to z azurové (Cyan), purpurové (Magenta), žluté (Yellow)

a černé (black). Subtraktivní míchání barev je míchání barev, ve kterém se barvy nesčítají, ale naopak odčítají. Tato barevná reprezentace se využívá především v oblasti tisku. Podrobněji je tento barevný prostor popsán v [4].

Na první pohled pro potřebu této aplikace není moc vhodným reprezentantem, jelikož nepokrývá barevné spektrum viditelnosti lidského oka a konverze do tohoto prostoru by byla ztrátová. Odborná literatura navíc tento barevný prostor neuvažuje. Dále by při potenciálním vybrání nastala povinnost tento prostor naimplementovat.

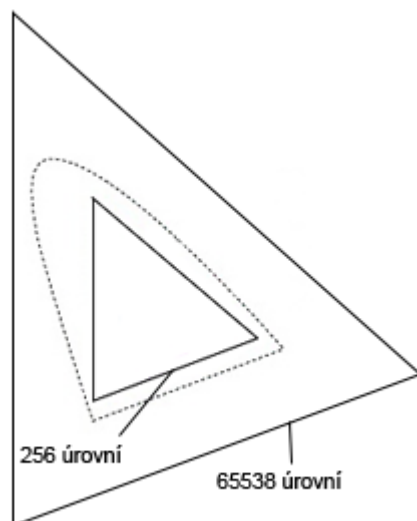
### 1.2.3 Závěrečné rozhodnutí



Obrázek 1.1: Obrázek překrytí barevných prostorů převzatý z [3]. A - viditelné spektrum lidského oka, B - RGB model, C - CMYK model

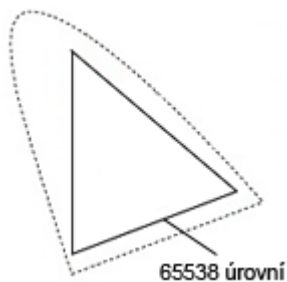
Zatím všechny barevné prostory nezahrnovaly celé viditelné spektrum lidského oka, viz obrázek 1.1. Tudíž nezbyvá než takovýto barevný prostor navrhnout. RGB prostor, až na tuto podmínku, celkem vyhovoval, z čehož vyplývá poznatek, že by bylo vhodné z tohoto modelu vycházet. RGB model obsahuje celkem 256 úrovní jasu na kanál a tvoří jakýsi trojúhelník uprostřed barevného spektra viditelnosti barev. Proč tedy tento trojúhelník ne zvětšit tak, že by barevný prostor lidského oka pojmul, viz obrázek 1.2. To by znamenalo, že by se musely přidat další úrovně jasu barevného kanálu, ale se stejnou granularitou. Dle zmiňovaného obrázku bude 16bitů čili 65538 úrovní jasu na kanál stačit.

Rozhodnutí použít tento barevný nadprostor spektra lidského oka umožňuje bezztrátový převod do toho prostoru a díky analogii s RGB modelem lze bez problémů a úskalí využít literatury poposující oblasti, které budou později využity. Jedinou nevýhodou je, že se tento prostor bude muset implementovat od základu, což by nemělo působit takové problémy.



Obrázek 1.2: Obrázek znázorňující budoucí stav nadprostoru lidského oka.

Pro malé ulehčení se zatím tento prostor bude převodově jevit jako prostor RGB se šesnásobnou granularitou viz obrázek 1.3, jelikož zatím nebudou jiné barevné modely potřeba. Bude zde pouze aplikován převod RGB modelu kvůli zobrazování operací aplikace v GUI. Později, až se v aplikaci AGE projeví potřeba pracovat s dalšími prostory, převod bude upraven dle popisované idey znázorněné na obrázku 1.2.



Obrázek 1.3: Obrázek znázorňující prozatimní stav nadprostoru lidského oka.

Dále se nesmí zapomenout na alfa kanál, který bude udávat průhlednost. U všech zmiňovaných prostorů by stačila reprezentace pomocí 8 bitů celého čísla.

## 1.3 Reprezentace a práce s rastrem

Celá ideologie aplikace AGE se soustřeďuje kolem scény, která je předmětem práce [1], jež má představovat pracovní plochu pro uživatele. Scéna bude obsahovat strom objektů, kde každý objekt bude vracet svoje vyrenderování. Toto vyrenderování má představovat grafickou reprezentaci daného objektu ve vektorové či rastrové formě. Vyrenderování bude reprezentované třídou *Rendered* a její vektorová varianta pomocí *RenderedCurves*. Tyto dvě třídy jsou součástí práce [1]. A pro cíl této práce zbývá doimplementovat třídu, která bude implementovat rastrovou variantu vyrenderování.

Nabízí se varianta rozdělit implementaci bitové mapy nebo-li rastru do dvou tříd, kde jedna třída bude druhou obalovat a přeposílat jí požadavky. První třída by plnila funkci konkrétní reprezentace bitové mapy a druhá třída by zapadala od hierarchie dědičnosti vyrenderování. Toto rozdělení by poskytovalo výhodu v tom, že v případě špatné volby reprezentace by následná oprava obnášela pouze vyměnění vnitřní třídy a venkovní třída by zůstala zachována.

V dnešní době mnoho výpočetních zařízení nabízí několik jader, která dokáží pracovat paralelně. Z tohoto vyvstává myšlenka, že by měl být vytvořen komplexní mechanismus pro paralelní práci s bitovou mapou, jež by se z navazujících částí snadno používal.

### 1.3.1 Možnosti

Zde budou položeny a následně vybrány implementované či neimplementované možnosti konkrétní reprezentace bitové mapy.

#### Použití bitmapy z .NET

Jednou z možností, jak se na celou situaci dívat, je využití implementované bitmapy, která se nachází ve využívaném prostředí .NET jmenného prostoru *System.Drawing*. Tato třída by se poté obalila třídou zapadající do hierarchie vyrenderování. Přístup k jednotlivým pixelům bitmapy by musel být proveden přes "unmanaged" kód, jelikož nejsou na bitmapě definovány příslušné metody.

To představuje jistou nevýhodu v tom, že by implementace reprezentace bitové mapy byla obtížnější z hlediska udržení stability kódu. Avšak hlavní nevýhodou této bitmapy je, že pouze užívá barevný prostor RGB a tím by popřela ideu uvažovanou v předchozí podkapitole.

## Implementace vlastní bitové mapy

Další možností se jeví implementace nové bitmapy. Tato bitmapa by byla napsána dle potřeb následných funkcionalit aplikace AGE a kdykoliv by se dala dle nutnosti upravit. Podporovala by barevné spektrum lidského oka, jelikož by pixely byly tvořeny reprezentací barevného prostoru ze závěru předchozí podkapitoly.

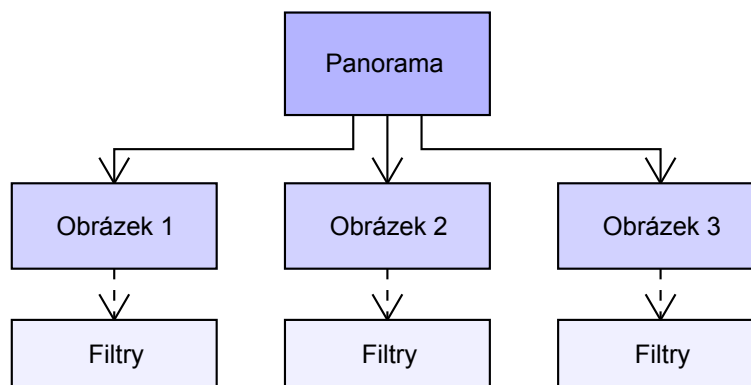
Jelikož bude tato bitmapa tvořena od základu, nastavá zde také nutnost naimplementovat převod do .NET bitmapy ze jmenného prostoru *System.Drawing*, jelikož bude využita při zobrazování v grafickém rozhraní.

### 1.3.2 Závěrečné rozhodnutí

Po zvážení obou variant by bezesporu jednoznačně měla být naimplementována varianta druhá, jelikož možnost měnit kód reprezentace bitové bitmapy bude velice vhodná v horizontu vývoje aplikace AGE.

## 1.4 Panoramata

Panorama, jinak řečeno skupina obrázků, která je určitým způsobem pozměněna tak, že se tváří jako jeden, by měla rozumným způsobem zapadat do hierarchie AGE. Mělo by se využít toho, že scénu AGE tvoří strom grafických objektů, na něž se aplikují grafické filtry.



Obrázek 1.4: Diagram zobrazující strukturu panoramata

Jak by reprezentace panoramat mohla vypadat? Jelikož je panorama určitá skupina, mohlo by být panorama objektem scény, která bude obsahovat další

podobjetky reprezentace rastrového obrazu. Na tyto podobjetky by se aplikovaly filtry, které by provedly konkrétní změnu obrázku tak, aby zapadaly do výsledného panorama viz obrázek 1.4.

### 1.4.1 Možnosti

V dnešní době se nachází více způsobů, jak modifikovat zdrojové obrázky tak, aby ve výsledku tvořily panorama. Nyní budou popsány a následně zhodnoceny způsoby tvorby panoramat.

#### Dvourozměrná panoramata

Dvourozměrným panoramatem se rozumí spojování obrázků jak v horizontální ose, tak i v ose vertikální. Z toho vyplývá, že je možné nafotit velké množství fotografií, které se nějakým způsobem překrývají a mechanismus je sám spojí. Focené snímky nemusí být zarovnané v žádné ose, stačí pouze překrytí. Tento mechanismus je podrobně popsán v literatuře [5]. Funguje zhruba tak, že se nejprve ze snímků získají *SIFT* viz [6] příznaky z obrázků, jež tvoří maximální, či minimální rozdíly Gaussovy funkce. Dále se najde  $k$  sousedních příznaků pro každý příznak, které se uloží do K-D stromu viz [7]. Po této operaci se zjistí, které obrázky na sebe navazují. Pomocí daných příznaků se obrázky transformují a může dojít k jejich překrytí a prolnutí.

Tato panoramata mají velikou výhodu, neboť nepotřebují žádné vstupní informace od uživatele a dovedou pracovat v plně automatizovaném režimu. Jedná se o nejpokročilejší metody jak tvořit panoramata a měly by pokrýt všechny požadavky uživatele. Naproti tomu veliká nevýhoda se nachází v tom, že je tento mechanismus značně náročný na implementaci a jeho zasazení do práce by zabralo příliš mnoho času, jelikož tvorba těchto panoramat obsahuje mnoho pokročilých algoritmů se složitou implementací, jak vyplývá z literatury [5].

#### Jednorozměrná panoramata

Jednorozměrná panoramata spojují obrázky pouze v horizontální ose. Jde o typ panoramat, se kterými se lze běžně setkat. Idea je taková, že uživatel nafotí snímky s určitým překrytím na stativu či bez něj a bude se otáčet s fotoaparátem kolem osy kolmé k zemské ploše. Body těchto snímků se poté promítnou na abstraktní válec a rozvinou se do roviny. Potom se pomocí *Lucas-Kanede* algoritmu určí vzájemná pozice snímků. Následně se snímky prolnou a výsledné panorama se ořízne. Prodrobnější popis lze nalézt v literatuře [8].

Jen stručně, jak *Lucas-Kanede* algoritmus funguje. Algoritmus dostane dva obrázky s počátečním odhadem překrytí. Z každého obrázku vytvoří pyramidu obrázků takovou, že se obrázek zmenšuje vždy na poloviční rozměry, dokud nebude tak malý, že by jeho zmenšení nemělo žádný smysl. V případě, že budeme mít obrázek o velikosti 400x300, bude pyramida složena z obrázků o velikosti 400x300, 200x150, 100x75, 50x37, atd. Algoritmus postupně bere obrázky z obou pyramid od nejmenšího po největší a počítá, o kolik a kam se mají obrázky posunout tak, aby navazovaly tzv. světelné toky ve zmenšených obrázcích.

Tento druh panoramat neklade takové nároky na implementaci a jejich tvorba je dobře popsána. Nevýhodou však pro uživatele je, že bude muset zadávat odhad překrytí jednotlivých obrázků tak, aby navázání světelných toků se neubíralo špatným směrem.

### **Posouvání obrázků s prolnutím**

Tato varianta je zjednodušením jednorozměrných panoramat a inspirovala se v pluginu *Pandora* viz [9] grafické aplikace *Gimp*. Zde uživatel nejprve vloží nafocené snímky, seřadí je dle návaznosti, určí jednotlivá překrytí a po potvrzení vstupních informací dostane výsledné panorama. Jednotlivým snímkům jsou přiřazeny vrstvy a kraje navazujících snímků jsou umazány. Jediným způsobem jak panoramata doladit je posouvat vrstvy snímků vůči sobě.

Tato implementace je velice triviální a nezabrala by skoro žádný čas. Tato výhoda je však vykoupena tím, že ve výsledku bude panorama obsahovat velké množství nepřesností, které by byly na první pohled patrné. Navíc po vytvoření panoramata není možné pohodlným způsobem měnit prolnutí obrázků, jelikož umazání okrajů je trvalé. Tato situace by se dala vyřešit pomocí budoucí hierarchie panoramat. Na každém obrázku, jež bude podobně objektem panoramatu, se budou moci přidávat, odebírat a měnit filtry a zmíněné odmazávání by bylo řešeno filtrem prolnutí do průhledna. Dále tento mechanismus postrádá veškerou automatizaci tvorby panoramat a uživatel je nucen si vytvářet vše sám.

### **1.4.2 Závěrečné rozhodnutí**

Pro konečnou tvorbu panoramat se rozhodl autor této práce využít druhé varianty. Tato varianta by měla tvořit jistý kompromis mezi kvalitou tvořených panoramat a mírou obtížnosti implementace. Dále bude moci uživatel výsledek procesu tvorby panoramat modifikovat, jelikož budou jednotlivé operace reprezentovány pozicí objektů a aplikací filtrů. Z hlediska

tvorby tohoto mechanismu bude potřeba naimplementovat zmenšovací filtr, jež se využije pro tvorbu grafických pyramid, soudkovací filtr obstarávající funkcionalitu projekce na válec, gradientní filtr umožňující prolnutí a nakonec ořezový filtr, jehož název popisuje funkcionalitu. Všechny tyto filtry budou popsány v následující podkapitole.

## 1.5 Filtry

V této sekci práce nastává otázka, jaké druhy filtrů by měly být vytvořeny. Sortiment filtrů by měl pokrýt alespoň základní požadavky uživatele. Běžně uživatel potřebuje upravovat své rastrové obrázky pro tvorbu grafiky umístěné na webu a jiných místech. Bezsporně bude potřebovat měnit barvu a tvar zdrojových obrázků. K tomuto účelu budou sloužit transformační filtry a filtry pro práci s barvou. Faktem je, že zdrojové obrázky většinou nebývají vytvořeny s ideální kvalitou. Často se v nich nachází šum a bývají rozostřené. K tomuto účelu bude sloužit kategorie konvolučních filtrů.

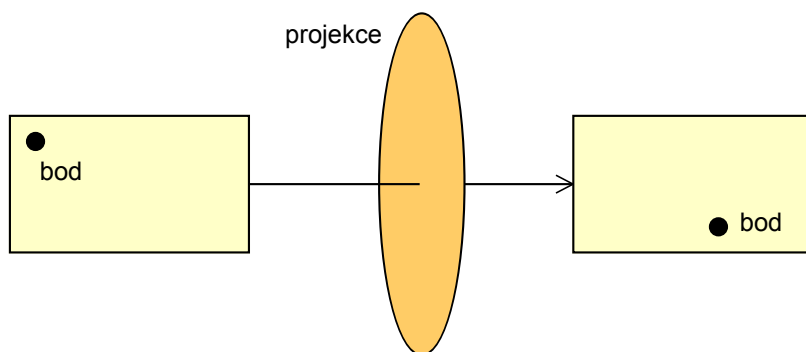
Dále je nutné uvážit, že by filtry měly umět pracovat nad vektorovým obrazem, jelikož by AGE měl být vektorověrastrový editor. U některých filtrů však tohoto požadavku nelze docílit, jelikož jejich definice uvažuje pouze obrázky s rastrovou strukturou. Příkladem jsou konvoluční filtry. Tento problém se dá vyřešit tím, že se pro tyto filtry bude vektorový obrázek převádět na rastrový. A ten v této podobě již zůstane. Tím vznikne další kategorie filtrů. Vektorové filtry budou pracovat přímo s vektorovým obrázkem, kdežto rastrové filtry využijí převodu.

### 1.5.1 Transformační filtry

Všechny transformační filtry ve své podstatě fungují obdobně, pouze jiným způsobem a s jiným výsledkem. Vezmou zdrojový bod obrázku s určitou pozicí a promítnou jej do jiné pozice, viz obrázek 1.5. Při této operaci však může dojít ke zvětšení či specifické deformaci obrázku, která se neobejde bez návaznosti na jiné body. Touto návazností je například myšleno, že barva některých bodů se bude muset průměrovat dle specifických pravidel.

#### Maticové filtry

Maticový filtr je speciálním případem transformačního filtru a promítnutí bodu je vyřešeno pomocí maticového násobení. K dané operaci jako je posunutí, otočení či zkosení odpovídá příslušná matice viz literatura [10]. Souřadnice zdrojového bodu tvoří vektor  $[x, y]$  ten je přenásoben zleva



Obrázek 1.5: Znázornění operace transformačního filtru.

konkrétní maticí  $A$  a z daného maticového součinu se získá cílová pozice bodu  $[x', y']$ .

V implementaci se bude postupovat opačný způsobem. Implementace se bude ptát, jaký příslušný bod bude náležet do předem určené cílové pozice. Proto bude nutné násobit cílovou souřadnici  $[x', y']$  inverzní maticí k  $A$ , aby se získala souřadnice zdroje  $[x, y]$ . Aby matice měla  $A$  inverzní protějšek, nesmí být singulární, což zapříčiní jistá omezení v implementaci. Důvod použití zpětné projekce je takový, že při paralelní aplikaci filtru při obyčejné projekci by docházelo ke kolizím mezi jednotlivými vlákny. Jak je možno se dále v této práci dočíst, každému vláknu bude přisouzena část rastrového obrazu. Transformace obrazu však způsobuje, že bod se může promítnout i do části jiného vlákna, než způsobilo danou projekci. Tím může dojít k případné kolizi při zapisování výsledku transformace.

**Rotační filtr** Tento filtr by měl umožňovat otáčet objekty ve scéně pomocí definice úhlu otočení a spolupráce s myší. Pro takovouto interakci s uživatelem by měl být vytvořen nástroj, který bude ovládání myši předávat filtru. Otočení objektu by mělo být provedeno okolo specifického bodu, jež může být definován jako střed daného objektu.

$$A = \begin{pmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{pmatrix}$$

Výše se nachází přesná definice matice  $A$  pro rotační filtr viz [10]. Znak  $\phi$  značí úhel rotace.

**Zkosovací filtr** Na tento filtr by měly být kladeny podobné požadavky jako na předchozí, s tím rozdílem, že u tohoto filtru budou figurovat úhly dva a

nebude se nikam otáčet, nýbrž se pomocí něj bude zkosovat obrázek vůči horizontální nebo vertikální ose o určený úhel se středem definovaném jako v předchozím filtru.

$$A = \begin{pmatrix} 1 & \tan(\alpha) \\ \tan(\beta) & 1 \end{pmatrix}$$

Na uvedené matici  $A$ , která nebyla převzata z žádné literatury, znamená  $\alpha$  úhel posunutí ve směru vertikální osy a  $\beta$  úhel posunutí ve směru osy horizontální. U této matice se může stát, že bude singulární, jelikož funkce tangenty může nabývat hodnot 1 či  $-1$  a velikost úhlů je navzájem nezávislá. Proto bude jejich absolutní součet omezen na 70 stupňů s jistou rezervou, aby se zbytečně nevytvářel veliký výsledný obrázek a nezatěžoval tak aplikaci, jelikož u součtu blížícího se k 90 stupňům vznikne z obrázku úsečka končící v nekonečnu.

### Ostatní filtry

Tyto filtry se nedají dále kategorizovat, jelikož jejich charakteristika je natolik specifická, že by nemohla tvořit žádnou ucelenou skupinu s podobnou funkcionalitou.

**Převracení filtr** Měl by umožňovat zrcadlového převracení objektu dle jeho vertikální či horizontální osy.

**Ořezový filtr** Pomocí něho by se měly dát provádět výřezy na předložených objektech. Pro tuto funkci by se bezesporu hodil nástroj ovládací tento filtr pomocí myši. Výřez by měl být specifikován vzdálenostmi ořezových hran od středového bodu, pomocí něhož by se výřez dal posouvat.

**Soudkovací filtr** Stejně jako předchozí filtr bude tento filtr vytvořen z důvodu tvorby panoramat v této aplikaci. Soudkování představuje promítnutí souřadnic obrázku na virtuální válec a jeho pomyslného rozstříhnutí a rozbalení do roviny viz [8]. Daná projekce by se měla dát ovlivnit vzdáleností obrázku od středu válce a jeho poloměrem.

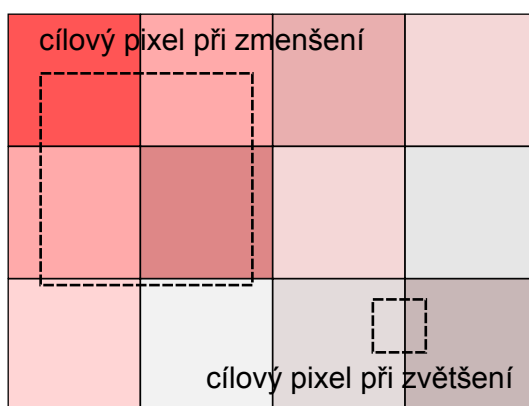
**Dlaždičkový filtr** Zde bude implementována funkcionalita pro tvorbu textur, jelikož je zbytečné členit toto odvětví do speciální části. K účelu naprosto postačí jeden filtr.

Ze zdrojového obrázku nejprve odřízne kus z levé či pravé strany a vloží ho na opačnou stranu, kde se tento kus prolne zbytkem obrázku. Prolnutím

je myšleno vážené míchání barev přemístěného kusu a zbytku obrázku dle vzdálenosti od kraje. Čím bude vzdálenost od kraje větší, tím bude váha barev přemístěného kusu menší. Tím se získal opakovatelný obrázek v horizontálním směru. A tato operace bude ještě jednou analogicky provedena pro vertikální směr.

**Zoom filtr** Funkcionalitou tohoto filtru bude zvětšování či zmenšování zdrojových obrázků. V praxi existuje několik metod, jak provádět změnu velikosti rastrového obrázku viz [11]. Pro implementaci tohoto filtru budou vybrány jen některé, které je možno vidět níže.

**Nejblíže bod** Tento algoritmus prochází cílovou bitmapu a ptá se zdrojové bitmapy, jakou má mít bod barvu. Ve své podstatě vezme algoritmus souřadnici cílového bodu, promítne ji do zdrojové bitmapy viz obrázek 1.6 a určí, který bod zdrojové bitmapy je nejblíže a jeho barvou obarví cílový bod.



Obrázek 1.6: Znázornění promítnutí cílového bodu do původního obrázku.

**Bilineárně** Algoritmus pracuje obdobně jako ten předchozí s tím rozdílem, že neurčí pouze jeden nejblíže bod, ale hned čtyři. Z těchto čtyřech bodů se pak pomocí váženého míchání barev získává výsledná barva. Zmíněná váha pro toto míchání je určena vzdáleností pozice od konkrétního bodu. Čím více vzdálenější od bodu, tím menší váha barvy.

**Supersampling** U tohoto algoritmu je přístup odlišný. Z cílové bitmapy se promítá do zdrojové celý čtverec určující velikost rozměry pixelu. Podle toho, kam se čtverec pixelu promítne a jaké body zdrojové bitmapy bude překrývat, se provede namíchání barvy cílového pixelu.

**Pixelizační filtr** Tento filtr bude zmenšovat rozlišení zdrojového obrázku při zachování rozměrů. Výsledek by měl vypadat jako rozkostičkování zdrojového obrazu, které by pro tento účel mělo využívat algoritmů z předešlého filtru.

## 1.5.2 Konvoluční filtry

Tyto filtry získaly svůj název podle konvoluční masky, jež pro svou práci využívají. Konvoluční maska je označení pro čtvercovou matici definující váhy jednotlivých bodů. Algoritmus těchto filtrů probíhá tak, že konvoluční maska rozměru  $n*n$  jede po zdrojovém obrázku a jednotlivým bodům přisuzuje váhy dle hodnot v matici. Z těchto vážených barev se pomocí míchání dostává barva cílového bodu. Souřadnice cílového bodu bývá zpravidla stejná jako souřadnice zdrojového bodu, nad nímž se nachází střed konvoluční masky. Pro vytvoření ideí funkcionality těchto filtrů bylo čerpáno z [12], v níž popis jednotlivých částí je dosti vágní.

### Nízkofrekvenční filtry

Hlavním úkolem třídy těchto filtrů viz [12] je odstraňovat z rastrového obrazu šum. Ten bývá charakterizován skokovou změnou obrazu, což je ve své podstatě vysoká frekvence změny jasu. Filtry tedy budou propouštět pouze nízké frekvenční spektrum jasu. U těchto filtrů se dá měnit velikost konvoluční masky a zapojit tím do operace více pixelů. Nejmenší maska však přichází v úvahu o velikosti  $3 * 3$ , jelikož souřadnice cílového bodu musí ležet uprostřed masky, a případné zvětšení se provede ve skocích rozměrů o dva. Nutností je, aby byl součet hodnot konvolučních masek 1. V případě, že by tomu tak nebylo, docházelo by k zesvětlování či ztmavování obrazu.

**Medián** Tento filtr viz [12] nepoužívá konvoluční masku jako takovou a dalo by se polemizovat, jestli zde má být tento filtr zařazen. Na druhou stranu se jedná o nízkofrekvenční filtr, který posouvá čtvercové okénko určité velikosti po zdrojovém obrázku. Výřez okénka určí body, ze kterých se bude vypočítávat medián barev. Tato barva bude výslednou pro cílový bod.

**Průměrové filtry** Tyto filtry viz [12] budou dělat aritmetický průměr barev bodů, nad nimiž se nachází určitý grafický obrazec jako je čtverec, kosočtverec či kruh. Níže jsou uvedeny příklady příslušných konvolučních masek o velikosti  $5 * 5$ .

$$\begin{pmatrix} 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1/13 & 0 & 0 \\ 0 & 1/13 & 1/13 & 1/13 & 0 \\ 1/13 & 1/13 & 1/13 & 1/13 & 1/13 \\ 0 & 1/13 & 1/13 & 1/13 & 0 \\ 0 & 0 & 1/13 & 0 & 0 \end{pmatrix}$$

**Gaussův filtr** Tento filtr viz [12] funguje obdobným způsobem jako předchozí kategorie průměrových filtrů s tím rozdílem, že Gaussův filtr dává větší váhu těm bodům, které se blíží více středu. Toto rozdělení je dáno Gaussovou křivkou, jež maska bude aproximovat. Další způsob, jakým si lze vyložit masku tohoto filtru je, že masku tvoří kvadrát řádku Pascalova trojúhelníku s příslušným počtem čísel. Níže je uvedena maska o velikosti  $5 * 5$ .

$$\frac{1}{256} \begin{pmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix}$$

### Vysokofrekvenční filtry

Tyto filtry viz [12] slouží k detekci hran v obraze a oproti nízkofrekvenčním filtrům vysokou frekvencí jasů vyhledávají. Hledání hran je řešeno pomocí parciálních derivací jasu. Jelikož se zde bude pracovat na obrázcích a nikoliv na spojitých funkcích, bude muset být parciální derivace aproximována. K tomu budou sloužit příslušné aproximační masky. Součet hodnot těchto masek bude dávat hodnotu 0 a jejich velikost bude vždy konstantní.

Jak bylo zmíněno v [12], filtry se dají využít i pro ostření obrazu. Pro ostrý obraz je specifické, že obsahuje velmi ostré hrany. Neostrý obraz by se dal doostřit tím, že by se původní obraz sečetl s detekovanou hranou. Dle autora této práce by se ostření obrazu mělo docílit tím, že do prostřední hodnoty masky se přičte jednička, čímž se přičte i bod zdrojové bitmapy. Dále by u těchto filtrů byla vhodná možnost měnit efektivitu, respektive ji snižovat. Toho autor této práce docílil tím, že se příslušná konvoluční maska vynásobila koeficientem nabývajícím hodnoty mezi 0 a 1. To zapříčinilo, že charakteristika hrany nebude tak patrná. Při záměru ostřit obrázky se však musí násobení koeficientu aplikovat dříve, než se do konvoluční masky bude přičítat jednička.

**Robertsův filtr** Tento filtr používá dvě konvoluční matice 2\*2 a to zároveň. Cílový bod se nachází v levém horním rohu každé konvoluční matice. Výpočet probíhá tak, že se vypočítají absolutní jasy po aplikaci obou masek a sečtou se. Níže jsou uvedeny zmíněné masky.

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

**Filtry první parciální derivace** Zde se využívá prvních parciálních derivací, při nichž se testuje kolmice na směr hrany (skokového gradientu). Jak již bylo řečeno, derivace se musí aproximovat. Zde se používá osmice konvolučních masek o velikosti 3 \* 3, přičemž každá symbolizuje jiný směr derivace. Všechny se postupně aplikují a vybere se výsledek té, jejíž aplikace vrátila největší jas. Aproximace derivace jasu probíhá tak, že jasy bodů na jedné straně konvoluční masky se přičtou a na druhé odečtou. V případě, že dané body mají podobnou hladinu jasu, dostává se zanedbatelná změna, jež je reprezentována černou barvou. Naopak v případě, že jasy bodů, které se přičtou, budou příliš rozdílné oproti jasům bodů, které se odečtou, pak bude výsledek reprezentován barvou blížící se bílé a tím se znázorní hrana. Možností, jak aproximovat derivace, je celá řada, proto existuje více filtrů řešících ten samý problém. Níže jsou uvedeny masky pro směr zezdola nahoru: Sobelova, Prewittova, Robinsonova a Kirschova filtru viz [12]. Ostatní masky příslušného filtru se získají postupným otáčením matice o 45 stupňů zvoleným směrem.

$$\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ -1 & -1 & -1 \end{pmatrix} \begin{pmatrix} 3 & 3 & 3 \\ 3 & 0 & 3 \\ -5 & -5 & -5 \end{pmatrix}$$

**Laplacův filtr** Filtr je jediný zástupce filtrů pracujících s druhou derivací. Oproti předchozí kategorii používá najednou pouze jednu matici 3 \* 3 aproximující druhou derivaci, neboť u druhé derivace není potřeba určovat nějaký směr, jelikož pomocí druhé derivace se hledá minimum či maximum v obraze. Aproximace probíhá tak, že se testuje jasová změna bodu, který se odečítá, oproti bodům okolním, jež jejich jasové složky se přičítají. Tento filtr je definován ve čtyřech variantách viz [12], jejichž konvoluční masky jsou vidět níže.

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 2 & -1 & 2 \\ -1 & -4 & -1 \\ 2 & -1 & 2 \end{pmatrix} \begin{pmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & -1 \end{pmatrix}$$

### 1.5.3 Barevné filtry

Při úvahách v této sekci nebylo využito žádných cizích zdrojů a literatury. Tyto filtry mění barvy jednotlivých bodů ať už vektorových či rastrových obrázků dle specifického pravidla konkrétního filtru. U těchto filtrů by se měla operace na barvě delegovat do operací na jednotlivých barevných kanálech, aby bylo možné operaci na konkrétním kanálu vyloučit. Delegací operace do operací na barevných kanálech je myšleno, že daná operace se bude provádět zvlášť po jednotlivých barevných složkách.

**Jasový filtr** Tento filtr bude umožňovat zmenšování či zvětšování jasu na daném obrázku, pomocí poměrného míchání barvy bodu s černou či bílou barvou. Čím více váhy se bude přikládat původní barvě, tím bude jas obrázku více zachován.

**Kontrastový filtr** Funkcionalitou filtru bude změna kontrastu zdrojového obrázku. Kontrast barvy se dá chápat jako odstup bílé barvy jakožto maxima a barvy černé jakožto minima. Tudíž nulovou barvou se rozumí barva šedá. Nabízí se pro zmenšení kontrastu barvy poměrové míchání se šedivou barvou. Naopak zvětšení kontrastu bude provedeno tak, že se vezme odstup barvy od šedivé a s určitým koeficientem bude navýšen. Bude-li barevná složka mít menší jas než příslušná složka šedivé, bude její jas ještě víc snížen. Opačný případ bude řešen analogicky.

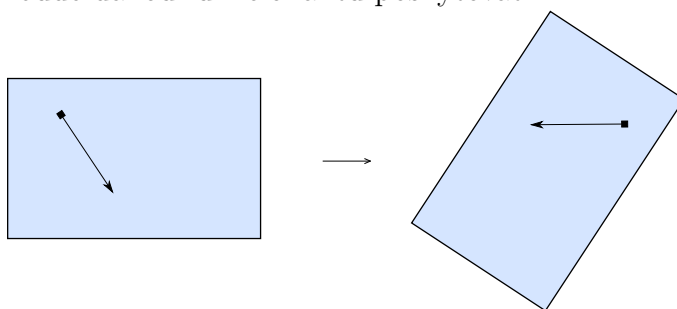
**Filtr změny barvy** U tohoto filtru budou mít hlavní roli v operaci dvě barvy. První, vůči které se bude porovnávat barva bodu obrázku. Porovnáním se myslí testování podobnosti barev dle hodnoty jasu jednotlivých kanálů s určitou tolerancí. V případě, že barvy budou označeny jako shodné, barva bodu se nahradí barvou druhou.

**Invertovací filtr** Jak je z názvu patrné, funkcionalitou toho filtru bude invertace jasu jednotlivých kanálů, přičemž inverzí jasu se rozumí převrácení rozdílu jasu od intenzity barevné složky šedivé zdola nahoru či obráceně.

**Odbarvovací filtr** Zde bude probíhat transformace barvy do barvy šedivé škály, jejíž kanály mají stejnou hodnotu jasu. Při této operaci však bude muset být zachován celkový jas barvy.

**Obarvovací filtr** Obarvováním bitmapy se rozumí poměrové míchání konkrétní obarvovací barvy s barvou bodu bitmapy. Čím větší váhu bude mít obarvovací barva, tím bude barva obrázku více tónována do této barvy.

**Gradientní filtr** Tento filtr bude vytvořen jako jedna z funkčních částí tvorby panoramat. Tato část má za úkol zprůhlednit okraje obrázků tak, aby na sebe navazovaly. Jelikož se jedná o prolnutí obrázku s průhlednou barvou, bude filtr napsán obecně. U tohoto prolnutí se bude moci nastavit barva, se kterou se obrázek prolne. Rovněž bude možné určovat směr tohoto prolnutí na obrázku pomocí počátečního a koncového bodu. Bylo by vhodné, aby byla poskytnuta možnost ovládat body myši za pomoci vytvoření příslušného nástroje, jež bude danou funkcionalitu poskytovat.



Obrázek 1.7: Obrázek znázorňující otočení souřadnic zdrojového obrázku při aplikaci gradientního filtru.

Počáteční a koncový bod určují vektor prolnutí nebo-li gradientu. Jak tento mechanismus bude fungovat? Jelikož vektor gradientu může mít libovolný směr, nelze snadno určit, s jakou intenzitou může být aplikována barva gradientu. To znamená, že se nedá jednoznačně určit, jestli bude gradientní barva aplikována s plnou intenzitou od koncového bodu vpravo, vlevo, nahoru či dolů. Ani jedna z možností není správná, jelikož gradient nesměruje vždy po vertikální či horizontální ose. Proto se obrázek nejprve pootočí, viz obrázek 1.7 tak, že bude směr gradientu směřovat horizontálně zprava doleva. Ve skutečnosti se pootočí (přepočítají) souřadnice jednotlivých bodů obrázku. Tím se počáteční a koncový bod ocitnou na horizontální ose. Pro tuto akci bude využito vnitřních částí implementovaných v rotačním filtru. Body, které se vyskytnou vlevo od koncového bodu, budou obarveny barvou gradientu. Naopak barvy napravo od počátečního bodu budou zachovány. Mezi body bude provedeno poměrové míchání barvy bodu s barvou gradientu.

## 1.6 GUI

GUI (grafické uživatelské rozhraní) by mělo zpřístupnit funkce aplikace pro práci s filtry a panoramaty uživateli. Uživatel by měl přehledně vidět výsledek své práce a jednoduše najít příslušná nastavení. Grafické rozhraní by mělo vypadat zhruba tak, že bude obsahovat pracovní plochu, kde uvidí výsledek své práce, menu s běžnými prvky jako bývá načítání, ukládání, apod. a dále různé informativní části, které by usnadnily se orientovat v práci uživatele.

Nyní je nutné se rozhodnout, jakou technologii pro tvorbu grafického rozhraní zvolit. Budou zde zmíněny pouze dvě varianty, které obsahuje prostředí .NET. Ostatní byly rovnou zavržené, jelikož by bylo jejich použití komplikovanější. *WinForms* je technologie využívající staříčké *Win32 API*. Tuto technologii autor této práce dobře zná, a nebyl by problém ji použít. Dále se nabízí možnost použít technologii *Windows Presentation Foundation* zkráceně *WPF*. Její velikou výhodou je, že své zobrazení prezentuje pomocí *DirectX*. S touto technologií nemá autor žádné předchozí zkušenosti.

Nakonec se však autor rozhodl využívat technologii *WPF*, jelikož se ji chce naučit. Tato technologie by měla mít před sebou perspektivní budoucnost. Navíc GUI výsledné aplikace AGE by mělo být pomocí *WPF* vytvořeno. Části aplikace pro nastavení filtrů a tvorby panoramat by později mohly být použity v GUI aplikace AGE.

# Kapitola 2

## Implementace

Tato kapitola popisuje podrobný náhled na implementaci knihoven a výsledné aplikace pro práci s panoramaty a grafickými filtry. Čtenář zde nalezne rozdělení jednotlivých bloků funkcionalit do knihoven a jmenných prostorů. Je zde přesně zobrazeno, jak kód této práce navazuje na kód práce [1]. A v neposlední řadě se zde nalezne popis jednotlivých implementovaných tříd s jejich vzájemnými návaznostmi.

### 2.1 Základní návrh

Kód této práce je úzce spjat s kódem práce [1]. Na obrázku 2.1 je znázorněno, jak jsou jednotlivé knihovny obou prací propojeny. Šedivě jsou vyznačeny knihovny, v nichž se nachází kód této práce. Na obrázku se však nevyskytují všechny části práce [1], jelikož nejsou provázány s knihovnami této práce, z čehož vyplývá, že nejsou potřebné pro chod aplikace této práce. Zároveň se zde dají najít části obsažené v práci [2], jež tato práce přímo nevyužívá.

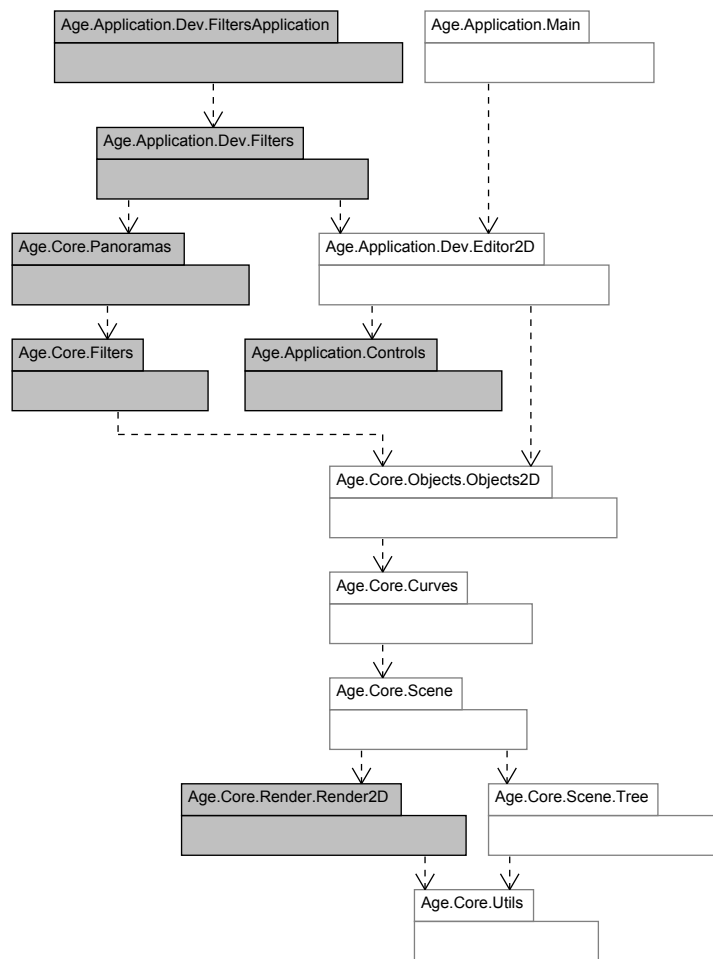
V následujícím seznamu jsou stručně popsány obsažené funkční části jednotlivých knihoven této práce. Později se však dostane podrobnějšího popisu.

#### **Age.Core.Render.Render2D**

Obsahuje implementaci barev, bitmap a mimo jiné vektorových křivek z práce [1]. Tyto grafické prvky se budou moci využívat různými objekty, které je budou modifikovat, například grafické filtry.

#### **Age.Core.Filters**

Obsahuje veškerý kód, který nějakým způsobem souvisí s filtry. Existují zde konkrétní filtry, ale i jejich abstraktní třídy a nástroje pro některé z nich, jež jsou důležité pro snadnější interakci mezi aplikací a uživatelem.



Obrázek 2.1: Digram popisující závislosti jednotlivých knihoven této práce. Nejsou zde znázorněny tranzitivní závislosti.

### **Age.Core.Panoramas**

Zde lze nalézt implementaci panoramat, jež pro svojí tvorbu využívají některé grafické filtry.

### **Age.Dev.Filters**

Obsahuje grafické komponenty, z nichž se skládá výsledné GUI.

### **Age.Application.Controls**

Rovněž obsahuje grafické komponenty pro výsledné GUI. Dále tato aplikace využívá komponenty práce [1] v *Age.Application.Dev.Editor2D*. Rozdělení komponent do více knihoven je důsledkem toho, že bylo nutné odseparovat práci jednotlivých autorů. Navíc práce [1] potřebovala

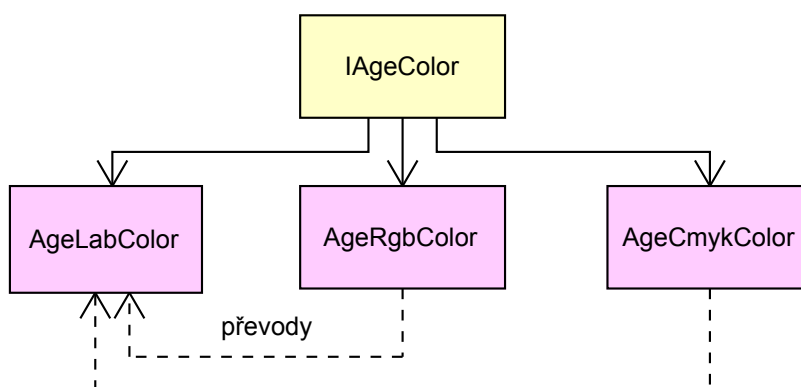
používat komponenty obsažené v *Age.Application.Controls*, a zároveň nebylo možné, aby využití probíhalo přímo z knihovny *Age.Dev.Filters*, jelikož některé z komponent této knihovny využívají komponenty práce [1] v *Age.Application.Dev.Editor2D*. Toto celkové rozdělení knihoven je prozatím a pro výslednou aplikaci AGE bude sjednoceno.

### Age.Dev.FiltersApplication

Tento projekt je vytvořen pouze ke spuštění aplikace a využívá předešlé knihovny komponent.

## 2.2 Barvy

Ve jmenném prostoru *Age.Core.Render.Render2D.Colors* se nachází veškerý kód reprezentace barev. Nalezne se zde několik struktur a rozhraní, které reprezentují barvu, jež se dají mezi sebou převádět jak je to vyznačené na obrázku 2.2. Dále jsou zde obsaženy i dva výčtové typy.



Obrázek 2.2: Diagram popisující hierarchii tříd barev.

### AgeColorStatus

Výčtový typ udávající příznak, jestli je barva prázdná, průhledná či normální. Normální barva je jakákoliv barva, u které lze rozpoznat určitý barevný odstín. Naopak průhledná a prázdná barva žádný barevný odstín nemají. Rozdíl mezi průhlednou a prázdnou barvou je takový, že průhledná barva má při míchání dvou barev určitou váhu, z čehož plyne, že průhledná barva ovlivní výsledek, kdežto prázdná barva se při míšení úplně zanedbává.

### **GrayscaleConversionType**

Výčtový typ udává, jakou metodou se bude barva převádět na stupeň šedi.

### **IAgeColor**

Každá třída reprezentující barvu musí implementovat toto rozhraní. Jsou v něm definovány převody z a do struktury *AgeLabColor*.

### **AgeRgbColor**

Tato struktura reprezentuje barvu v RGB modelu a je jí možno převést do struktury *AgeLabColor*.

### **AgeCmykColor**

Tato struktura reprezentuje barvu v CMYK modelu a je jí možno převádět do struktury *AgeLabColor*.

### **AgeLabColor**

Představuje implementaci nadprostoru viditelnosti lidského oka zmíněného v kapitole Analýza. Jak již bylo uvedeno, daný prostor je zatím implementován provizorně, a to tak, že se z a do RGB prostoru převádí jako jeho 16ti bitové zjemnění pro barevný kanál. Standardně se pro RGB kanál využívá 8 bitů. K tomuto provizornímu řešení vedlo usnadnění práce a situace, že se pro zobrazování zatím nebude používat jiný než RGB model. Později by měl převod vypadat tak, že se rozestupy mezi jednotlivými úrovněmi kanálu budou převádět z *AgeLabColor* do *AgeRgbColor* jedna ku jedné a *AgeLabColor* bude skutečným nadprostorem lidského oka.

## **2.3 Bitmapy**

Implementace bitových map, respektive reprezentace rastrového obrazu, je obsažena celkem ve třech jmených prostorech.

### **Age.Core.Render.Render2D.BitmapThreadPool**

Zde se vyskytuje mechanismus umožňující paralelní operace na bitových mapách, jež bude popsán později.

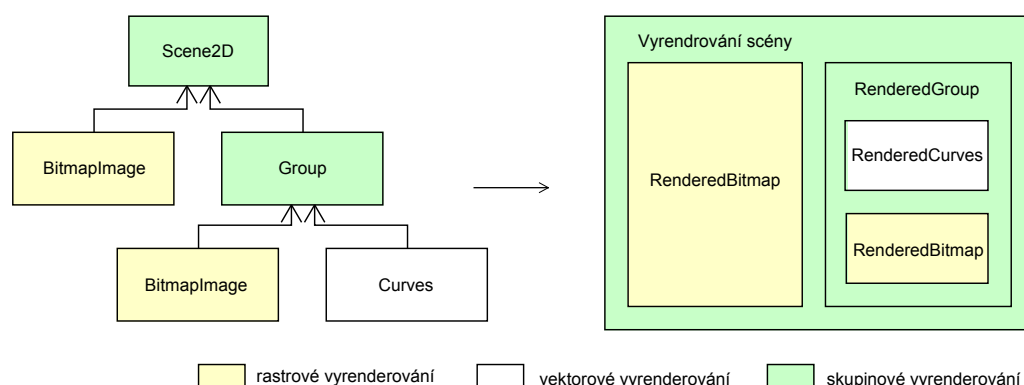
### **Age.Core.Render.Render2D.Bitmaps**

Obsahuje implementaci reprezentace bitové mapy *AgeBitmap* a mechanismy pro změnu její velikosti.

### **AgeBitmap**

Tato třída představuje bitovou mapu. Pod touto třídou si ve zjednušené formě lze představit pole struktur *AgeLabColor* uložené

po sloupcích, které reprezentují barvu pixelu. Jsou zde definovány přístupy k jednotlivým pixelům pomocí dvourozměrných souřadnic. Lze zde najít i další funkce, například konverze do jiných reprezentací bitových map, konkrétněji do bitmapy používané v prostředí .NET. Tuto bitovou mapu lze nalézt ve jmenném prostoru *System.Drawing*.



Obrázek 2.3: Diagram popisující hierarchii vyrenderování v závislosti na zdrojových objektech.

### Age.Core.Render.Render2D.Rendered

Z hlediska bitmap obsahuje pouze jednu třídu autora této práce se jménem *RenderedBitmap*, která umísťuje reprezentaci rastru do hierarchie vyrenderování AGE. Mimo jiné jsou zde obsaženy jiné implementační prvky, jež jsou součástí [1].

Pro pochopení situace bude popsána hierarchie scény AGE, která je předmětem práce [1]. Pod scénou je si třeba představit kreslicí plátno s množinou objektů reprezentující grafické útvary, jejichž třídy jsou uloženy ve jmenném prostoru práce [1] *Age.Age.Core.Objects.Objects2D*. Tato množina objektů tvoří stromovou strukturu.

Obrázek 2.3 popisuje následující situaci. Aby byl objekt vykreslen (vyrenderován), musí vrátit své vykreslení reprezentované třídou *Rendered*. Tato vyrenderování se vykreslují do sebe dle úrovně stromové struktury. Vyrenderování potomka je vždy vykresleno do vyrenderování nadřazeného objektu a to pokračuje až do objektu scény, jež představuje kořen stromu. Vyrenderování mohou být rozdílná, například vektorová, skupinová či rastrová. V případě, že na jedné úrovni stromové hierarchie se vyskytuje rastrové a vektorové či skupinové vyrenderování, je vektorové či skupinové vyrenderování převedeno na rastrové. Toto

přetypování se dále šíří do horních pater stromové hierarchie vyrenderování.

### **Rendered**

Abstraktní třída reprezentující obecné vyrenderování a je součástí práce [1].

### **RenderedBitmap**

Tato třída byla vytvořena autorem této práce pro zapojení bitových map do hierarchie vykreslování. Obsahuje funkcionalitu, která vykresluje vyrenderování do vyrenderování a pro reprezentaci bitové mapy ve své podstatě *RenderedBitmap* zaobaluje třídu *AgeBitmap*. Drží na ni referenci. Veškeré veřejné metody a vlastnosti jsou znovu napsány v *RenderedBitmap* a přeposílány na referencovanou třídu.

### **RenderedCurves**

Třída reprezentující vektorové vyrenderování, jež má obdobnou úlohu jako *RenderedBitmap* s vektorovou formou. Třída je součástí práce [1].

### **RenderedGroup**

Rovněž třída obsažená v práci [1] reprezentuje vyrenderování skupiny objektů.

## **2.3.1 Změna velikosti**

Jak už bylo zmíněno, každý objekt ve scéně AGE, aby mohl být vykreslen, musí vrátit svá vyrenderování a ta se dají vykreslovat do sebe. U každého vyrenderování je definován obdélníkový výřez udávaný v abstraktních AGE jednotkách, které jsou ekvivalentní metrické stupnici v milimetrech. Výřez určuje velikost a pozici vyrenderování vůči vyrenderování rodičovského objektu a rozlišení určené vztahem mezi AGE jednotkou a počtem pixelů. Jelikož vyrenderování mohou mít rozdílná rozlišení, je nutné je sjednotit při vykreslování jednoho vyrenderování do druhého. Změna rozlišení na vektorovém obrázku se provede jednoduše, prostě se změní. Naopak u rastrového obrázku probíhá změna složitěji. Velikost rozlišení u rastrového obrázku je určena poměrem velikosti bitmapy, která je pevná, a velikostí výřezu. Tudíž bude zde nutné měnit velikost bitmapy.

Ke změně velikosti bitmapy slouží implementační prvky obsažené ve jmenném prostoru *Age.Core.Render.Render2D.Bitmaps*.

### **AgeBitmapRescaler**

Statická třída obsahuje naimplementované metody pro změnu velikosti,

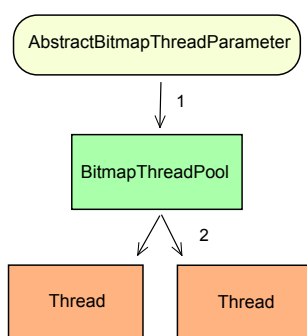
o nichž bylo pojednáno v předchozí kapitole.

### AgeBitmapRescalerType

Výčtový typ, který slouží jako identifikátor metody pro změnu velikosti bitmapy v případě, že je potřeba změnit velikost bitové mapy.

## 2.3.2 Paralelismus na bitmapě

V dnešní době mnoho počítačů obsahuje několik procesorových jader a je efektivní, když se mohou výpočty provádět paralelně. Z tohoto důvodu byl vytvořen mechanismus, jež umožňuje paralelizovat operace na bitové mapě.



Obrázek 2.4: Diagram zobrazující vztah objektů v mechanismu paralelní práce na rastru.

### BitmapThreadPool

Tato třída stojí za reprezentací zmíněného mechanismu. Jak tedy celý mechanismus funguje? Obrázek 2.4 znázorňuje situaci v následujících bodech.

**Bod 1.** Pomocí parametrů jsou *BitmapThreadPool* předány informace o prováděné operaci. Ty obsahují cílovou bitmapu s delegátem na prováděnou operaci a další pomocné informace, které lze vyčíst z programátorské dokumentace. Tyto informace jsou reprezentovány třídou *AbstractBitmapThreadParameter*.

**Bod 2.** Díky tomu, že *AbstractBitmapThreadParameter* obsahuje informaci o počtu sloupců v cílové bitmapě, *BitmapThreadPool* rozhodne, kolik paralelních běhů se provede. Tento počet nikdy nepřesáhne počet jader výpočetního stroje. Každému běhu je přisouzen určitý počet sloupců cílové bitmapy, na nichž bude pracovat. Jednotlivé běhy

jsou reprezentovány pomocí třídy *Thread* ze jmenného prostoru *System.Threading* prostředí .NET. Každému běhu před jeho spuštěním se opět předá parametr *AbstractBitmapThreadParameter*, jak již bylo u *BitmapThreadPool*.

### **AbstractBitmapThreadParameter**

Abstraktní třída reprezentující parametr informací předávaný *BitmapThreadPool*.

### **BitmapThreadParameter**

Třída poděděná od *AbstractBitmapThreadParameter* reprezentuje konkrétní parametr, u kterého se předpokládá, že pro prováděnou operaci bude stačit pouze jedna bitová mapa. Ta bude sloužit jak pro získávání zdrojových barevných bodů, tak i pro uložení výsledku.

### **BitmapThreadCopyParameter**

Potomek *BitmapThreadParameter*, který počítá s tím, že budou pro operaci potřebné bitmapy dvě, a to bitmapa zdroje dat a bitmapa, do které se uloží výsledek operace.

Všechny třídy parametrů jsou generické, což poskytuje možnost pracovat jak s *AgeBitmap*, tak i s *RenderedBitmap*.

## **2.4 Filtry a jejich nástroje**

V předchozí kapitole bylo řečeno, že existují různé kategorie filtrů. To se promítlo do rozdělení implementace mezi jmenné prostory.

### **Age.Core.Filters**

V tomto hlavním kořenovém jmenném prostoru se nachází kód nej-abstraktnějších tříd a specifikace většiny rozhraní, které budou popsány později.

### **Age.Core.Filters.ColorFilters**

Obsahuje třídy filtrů pro práci s barvou.

### **Age.Core.Filters.TransformationFilters**

Zde se nacházejí třídy a rozhraní implementující transformační filtry.

### **Age.Core.Filters.ConvolutionMaskFilters**

Zde jsou obsaženy abstraktní třídy se společnou funkcionalitou pro všechny konvoluční filtry.

### **Age.Core.Filters.ConvolutionMaskFilters.LowFrequencyFilters**

Tento prostor je určen pro nízkofrekvenční (vyhlazovací) filtry.

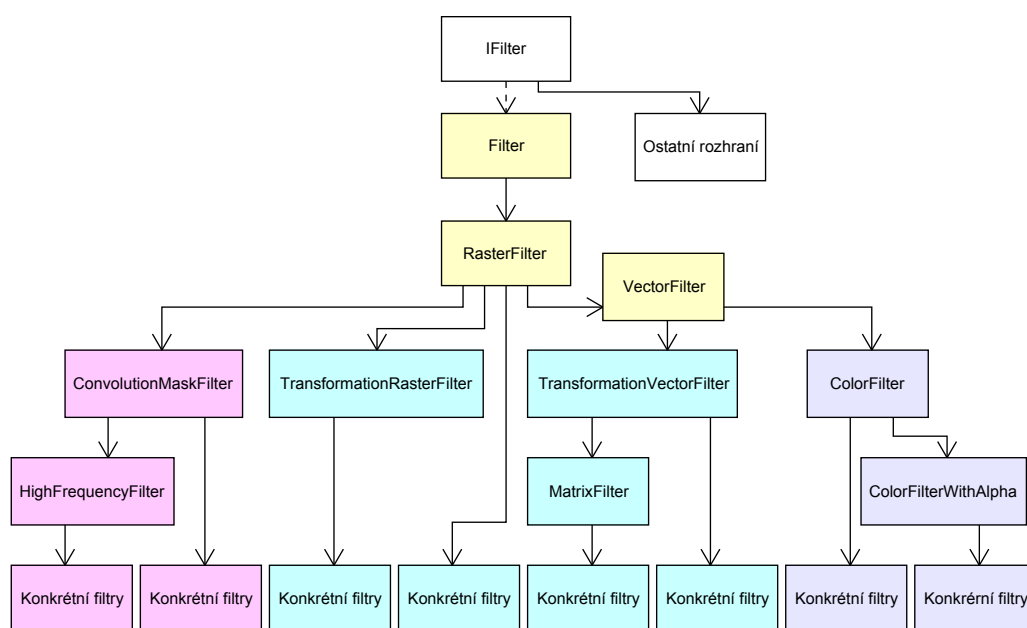
## Age.Core.Filters.ConvolutionMaskFilters.HighFrequencyFilters

Tento prostor je vyhrazen pro třídy vysokofrekvenčních (ostrících) filtrů.

## Age.Core.Filters.Tools

Obsahuje nástroje pro vybrané filtry, které by měly usnadnit práci uživateli.

### 2.4.1 Nejobecnější třídy filtrů



Obrázek 2.5: Diagram hierarchie dědičnosti abstraktních tříd filtrů.

Jak už bylo řečeno, *Age.Core.Filters* obsahuje třídy, které tvoří nejvrchnější patra stromového uspořádání dědičnosti. Na obrázku 2.5 je znázorněna dědičnost abstraktních tříd filtrů.

#### IFilter

Toto rozhraní slouží jako prvek kolekční třídy *FilterManager*. *FilterManager* plní funkci mechanismu pro postupnou aplikaci filtru na objekt ve scéně a je společně se zmíněným rozhraním součástí práce [1] a je možné jej najít v prostoru *Age.Core.Scene.Scene2D.Filters*.

#### Filter

Třída tvoří samotný kořen hierarchie dědičnosti a implementuje především metody rozhraní *IFilter*.

### **RasterFilter**

Třída dědí od třídy *Filter*. Jsou zde definovány metody pro paralelní aplikaci filtru na bitovou mapu. Od všech potomků této třídy, jež nebudou ve větvi dědicí od *VectorFilter*, se předpokládá, že když obdrží libovolné vyrenderování, převedou si ho na rastrovou podobu.

### **VectorFilter**

Třída dědí od *RasterFilter*. U tříd dědicích od *VectorFilter* bude příslušná operace filtru na vektorové vyrenderování aplikována vektorově.

V popisovaném prostoru se dále vyskytují rozhraní, jež mají být společným reprezentantem filtrů s podobnou specifickou charakteristikou, která je zejména uplatněna při implementaci grafického rozhraní. Tato rozhraní taktéž jako třída *Filter* implementují rozhraní *IFilter* z důvodu přístupu k metodám či vlastnostem implementovaných ve *Filter* při přetypování na dané rozhraní. Pro podrobnější popis rozhraní je dobré se podívat do programátorské dokumentace.

### **IEffectivityFilter**

Třídy implementující rozhraní mohou měnit intenzitu působnosti filtru. Tím je myšleno, že lze u filtru nastavit míru změny, kterou filtr provede.

### **IDifferentColorFilter**

Toto rozhraní zaručuje, že operace příslušného filtru na určitém pixelu bude provedena po barevných složkách a dle nastavení filtru lze aplikaci operace na jednotlivých složkách vynechat.

### **IDifferentColorFilterWithAlfa**

Toto rozhraní dědí od *IDifferentColorFilter* a umožňuje oproti rodiči modifikovat kanál alfa.

### **IResizableFilter**

Rozhraní definující konvoluční filtry, u kterých se dá měnit velikost konvoluční masky. Viz sekce s Nízkofrekvenčními filtry.

## **2.4.2 Barevné filtry**

Barevné filtry, nacházející se ve jmenném prostoru *Age.Core.Filters.ColorFilters*, lze považovat za filtry vektorové a je nutno zmínit, že barevné filtry využívají speciálního mechanismu pro aplikaci operace filtru na vektorové vyrenderování. Tento mechanismus, jež je implementován v každé třídě vyrenderování ze jmenného prostoru *Age.Core.Render.Render2D*, vznikl z toho důvodu, aby bylo možné

jednoduchým způsobem měnit barvu na jednotlivých vyrenderováních v práci [1]. Ve své podstatě jsou na filtru definovány dvě operace. Jedna pro modifikaci barvy a druhá pro paralelní modifikaci bitové mapy. Tyto operace jsou předány zmíněnému mechanismu vyrenderování, ten dle uvážení zvolí příslušnou operaci. Pro přeměnu vektorového vyrenderování se využije operace barvy a pro rastrové vyrenderování naopak operace bitmapová.

### **ColorFilter**

Nejabstraktnější třída barevných filtrů dědí od třídy *VectorFilter*, viz obrázek 2.1. Třída implementuje rozhraní *IEffectivityFilter* pro změnu intenzity filtru a dále rozhraní *IDifferentColorFilter* pro možnost aplikovat filtr zvlášť na barevné složce. Od této třídy dědí třídy konkrétních filtrů jako je *BrightnessFilter*, *ContrastFilter* a *GrayScaleFilter*.

### **ColorFilterWithAlpha**

Tato třída je potomkem *ColorFilter* viz obrázek 2.1. Díky tomu, že implementuje rozhraní *IDifferentColorFilterWithAlfa*, její potomci mohou modifikovat alfa kanál. Mezi tyto potomky se řadí *OverlayFilter*, *ColorInvertFilter* a *ColorChangeFilter*.

## **2.4.3 Transformační filtry**

U transformačních filtrů, nacházejících se ve jmenném prostoru *Age.Core.Filters.TransformationFilters*, není situace zařazení filtrů jednoznačná. Existují zde dvě větve tříd. Jedna větev reprezentuje transformační rastrové filtry a druhá ty vektorové.

### **TransformationRasterFilter**

Tato abstraktní třída reprezentuje rastrovou větev transformačních filtrů a tudíž dědí od *RasterFilter* viz obrázek 2.1. Zmíněná větev zahrnuje filtry, které si pro svojí aplikaci převedou vektorové vyrenderování na rastrové. Mezi tyto filtry patří *CropFilter*, *TileFilter* a *CylinderFilter*, které dědí přímo od popisované abstraktní třídy.

### **TransformationVectorFilter**

Tato abstraktní třída pro změnu reprezentuje větev vektorových transformačních filtrů. Filtry v této větvi pracují zvlášť s vektorovým a rastrovým obrazem. Tato větev je dále členěna na filtry, jejichž třídy přímo dědí od zmíněné abstraktní třídy, sem patří třídy *OverturnFilter*, *PixelizationFilter* a *ZoomFilter*, a filtry s dědičností proloženou abstraktní třídou *MatrixFilter*

### **MatrixFilter**

Tato abstraktní třída reprezentuje filtry měnící pozice bodů zdrojového

zobrazení pomocí maticové projekce rozebrané v předchozí kapitole. *MatixFilter* plní funkci implementace mechanismu projekce, kdežto konkrétní třídy pouze definují konkrétní matice pro danou projekci. Mezi tyto třídy patří *RotationFilter* a *SlopeFilter*.

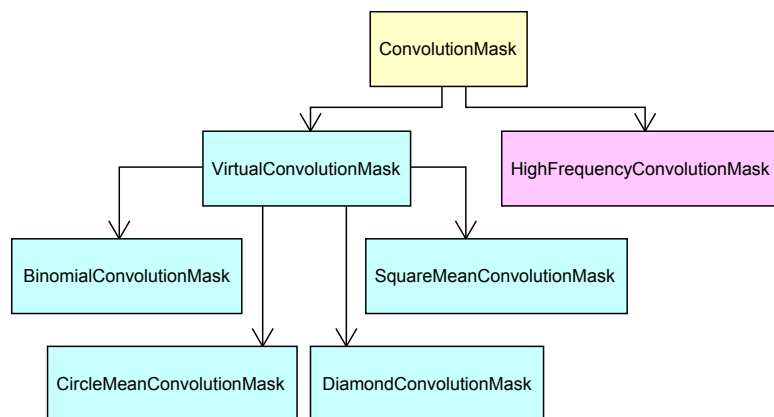
### IMiddlePointFilter

Rozhraní reprezentující filtry, u nichž je potřeba definovat středový bod. Operace příslušného filtru pro svou aplikaci využívá tohoto středového bodu a je u něj možno měnit pozici vůči počátku čili středu editovaného obrázku.

V této sekci filtrů se nalézá podobná situace, jako je tomu u barevných filtrů. U transformačních vektorových filtrů existuje podobný mechanismus pro aplikaci změny pozice bodu. Jsou zde opět nadefinovány dvě operace. Jedna operace pro změnu pozice bodu a druhá pro celkovou paralelní transformaci bitové mapy. Tyto operace jsou znovu předány aplikovanému vyrenderování a to aplikuje operace dle svého uvážení.

## 2.4.4 Konvoluční filtry

Konvoluční filtry se považují za filtry rastrové, tudíž se zdrojový obrázek nejdřív převede na rastrovou podobu a pak se na něj aplikuje operace filtru.



Obrázek 2.6: Diagram zobrazující dědičnost konvolučních masek.

### ConvolutionFilter

Tato třída slouží jako kořen hierarchie konvolučních filtrů, která dědí od třídy *RasterFilter* a implementuje rozhraní *IDifferentColorFilterWithAlpha*. Oba dva implementační prvky se nachází ve jmenném prostoru *Age.Core.Filters*.

## ConvolutionMask

Třída tvoří abstrakci mezi konvolučními maskami. Od této třídy dědí abstraktní třídy jak nízkofrekvenčních masek, tak i masek vysokofrekvenčních. Na obrázku 2.6 je vyznačena zmíněná dědičnost.

## Nízkofrekvenční filtry

Nízkofrekvenční filtry tvoří dvě třídy a nacházejí se ve jmenném prostoru *Age.Core.Filters.ConvolutionMaskFilters.LowFrequencyFilters*. Konvoluční masky těchto filtrů se nacházejí ve jmenném podprostoru s názvem *ConvolutionMasks*. Mezi těmito maskami se také objevuje dědičnost viz obrázek 2.6.

## MedianFilter

Dědí od třídy *RasterFilter* a je přímou implementací mediánového filtru. Tento filtr tvoří výjimku, protože se nejedná o konvoluční filtr, ale je zde zařazen, protože se naopak považuje za filtr nízkofrekvenční. Tato třída implementuje rozhraní *IResizableFilter*, které filtr využívá pro definici velikosti čtverce bodů, ze kterých je vybírán medián.

## LinearConvolutionMask

Generická třída *LinearConvolutionMask*, podděněná od *ConvolutionFilter*, slouží pro účel reprezentace průměrových filtrů popsanych v kapitole Analýza. Samostatné filtry jsou potom odlišeny pomocí různých konvolučních masek, jejichž typ se předá jako generický parametr třídy *LinearConvolutionFilter*. Třída implementuje rozhraní *IResizableFilter*. To zapříčiní možnost měnit velikost konvoluční masky.

## VirtualConvolutionMask

Tato abstraktní třída reprezentuje masky nízkofrekvenčních konvolučních filtrů a všechny třídy masek konkrétních průměrových filtrů od této třídy dědí a je zároveň podděněna od třídy *ConvolutionMask*.

## Vysokofrekvenční filtry

Vysokofrekvenční filtry se nacházejí ve jmenném prostoru *Age.Core.Filters.ConvolutionMaskFilters.HighFrequencyFilters* a obsahují pouze jednu abstraktní třídu, viz obrázek 2.5. Dále se ve jmenném podprostoru *ConvolutionMasks* nachází implementační prvky vysokofrekvenčních konvolučních masek.

## HighFrequencyMaskFilter

Abstraktní třída podděněná od třídy *ConvolutionFilter* definuje paralelní

aplikaci vysokofrekvenčních konvolučních masek na rastrový obrázek. Konkrétní třídy poděděné od abstraktní třídy, které se dají nalézt v programátorské dokumentaci, definují pouze konkrétní instanci masek a téměř žádné významně důležité metody.

### **HighFrequencyConvolutionMask**

Třída reprezentující vysokofrekvenční konvoluční masku dědí od třídy *ConvolutionMask*, viz obrázek 2.6. Na rozdíl od nízkofrekvenčních obsahuje tato třída více masek pevných rozměrů.

### **HighFrequencyConvolutionMaskSelection**

Třída, pomocí níž se nastavuje jedna konvoluční maska vysokofrekvenčního filtru jako aktuální.

## **2.4.5 Nástroje**

Implementace nástrojů hierarchicky navazuje na nástroje z práce [1], které je možno nalézt ve jmenném prostoru *Age.Core.Objects.Objects2D.Tool*. Nástroje spadající do předmětu této práce se nacházejí v *Age.Core.Filters.Tools*.

### **AbstractTool**

Abstraktní třída, jež umožňuje zapojení veškerých nástrojů do scény, je součástí práce [1].

### **SelectionTool**

Třída reprezentuje nástroj pro výběr a posouvání objektů, je poděděna od třídy *AbstractTool*, je rovněž součástí práce [1].

### **ZoomSelectionTool**

Třída opět reprezentuje nástroj pro výběr a posouvání objektů. Implementačním vzorem této třídy se stala třída *SelectionTool*. Vytvoření této třídy bylo způsobeno z důvodu, že *SelectionTool* obsahuje některé funkční prvky, jež nejsou potřebné pro výsledek této práce.

### **AbstractFilterTool**

Filtrové nástroje, které jsou předmětem této práce, jsou reprezentovány touto třídou, která obsahuje funkcionality společné pro konkrétní nástroje, a to především přepočty pozic bodů po aplikaci filtrů ve scéně. Od této třídy už dědí pouze konkrétní nástroje, jež jsou uvedeny v programátorské dokumentaci a především slouží k ovládání jednotlivých filtrů myší.

## 2.5 Panoramata

Implementace panoramat byla shrnuta pouze do jednoho jmenného prostoru *Age.Core.Panoramas*. V kapitole analýzy se navrhla myšlenka, jak bude třída panoramat zapadat do hierarchie scény AGE. Jedná se tedy o skupinu rastrových obrázků, na něž jsou aplikovány filtry. Rastrový obrázek je objektem scény a příslušné třídy jeho reprezentace lze nalézt ve jmenném prostoru *Age.Core.Objects.Objects2D.Objects* práce [1]. Nyní budou až na výjimky popsány jen třídy týkající se tvorby panoramat.

### BitmapImage

Tato třída je obecnou reprezentací rastrového obrázku a spadá do práce [1].

### BitmapFromFile

Třída dědí od *BitmapImage*. Její přídatnou funkcionalitou oproti rodiči je načítat se nebo se ukládat do soboru standardního typu, jako je .png, .jpg, .bmp, atd. Třída je součástí práce [1].

### PanoramaObject

Tato třída reprezentuje panoramata jako taková, dědí od třídy *Group* ze zmíněného jmenného prostoru objektů *Age.Core.Objects.Objects2D.Objects* z práce [1]. V *PanoramaObject* je implementován proces pro tvorbu panoramat, zmíněný v předchozí kapitole. Jen pro připomenutí, proces je tvořen z částí jako je zesoudkování jednotlivých obrázků, výpočet pozic obrázků pomocí bitmapových pyramid, prolnutí obrázků a oříznutí výsledku.

### ImagePyramid

Tato třída slouží pro reprezentaci bitmapových pyramid. Tu pro své výpočty používá třída *DistanceCalculator*.

### DistanceCalculator

Tato třída provádí výpočet výsledného rozestupu mezi obrázky a pro svoji funkcionalitu ji využívá *PanoramaObject*.

### ImagePath

Třída reprezentuje cestu k obrázku uloženého na disku. Je vytvořena se záměrem uchovávat cestu v průběhu existence obrázku v panoramatu.

## 2.6 GUI

Grafické prostředí aplikace je soustředěno ve třech knihovnách.

### **Age.Application.Dev.FiltersApplication**

Tato knihovna obsahuje pouze jedno okno *MainWindow*, viz obrázek 2.7 část 1, které se objeví po spuštění aplikace. Toto okno obsahuje pouze jednu komponentu, jež zastřešuje veškeré funkční prvky. Tato komponenta se nachází v *Age.Application.Dev.Filters*.

### **Age.Application.Dev.Filters**

Tato knihovna obsahuje komponenty a dialogová okna, z nichž je tvořeno grafické rozhraní této práce. Na rozdíl od ostatních grafických knihoven obsahuje několik jmenných prostorů.

#### **Age.Application.Dev.Filters.Components**

Zde se nachází zmiňovaná zastřešující komponenta grafického rozhraní *FiltersApplicationControl*.

#### **Age.Application.Dev.Filters.Components.Application**

Obsahuje grafické komponenty, z nichž je složena komponenta aplikace *FiltersApplicationControl*.

#### **Age.Application.Dev.Filters.Components.Filter**

Obsahuje grafické komponenty pro tvorbu dialogového okna s nastavením jednotlivých filtrů.

#### **Age.Application.Dev.Filters.Window.Filters**

Jmenný prostor zahrnuje mechanismy pro tvorbu zmíněného okna editace filtrů.

#### **Age.Application.Dev.Window.Panoramas**

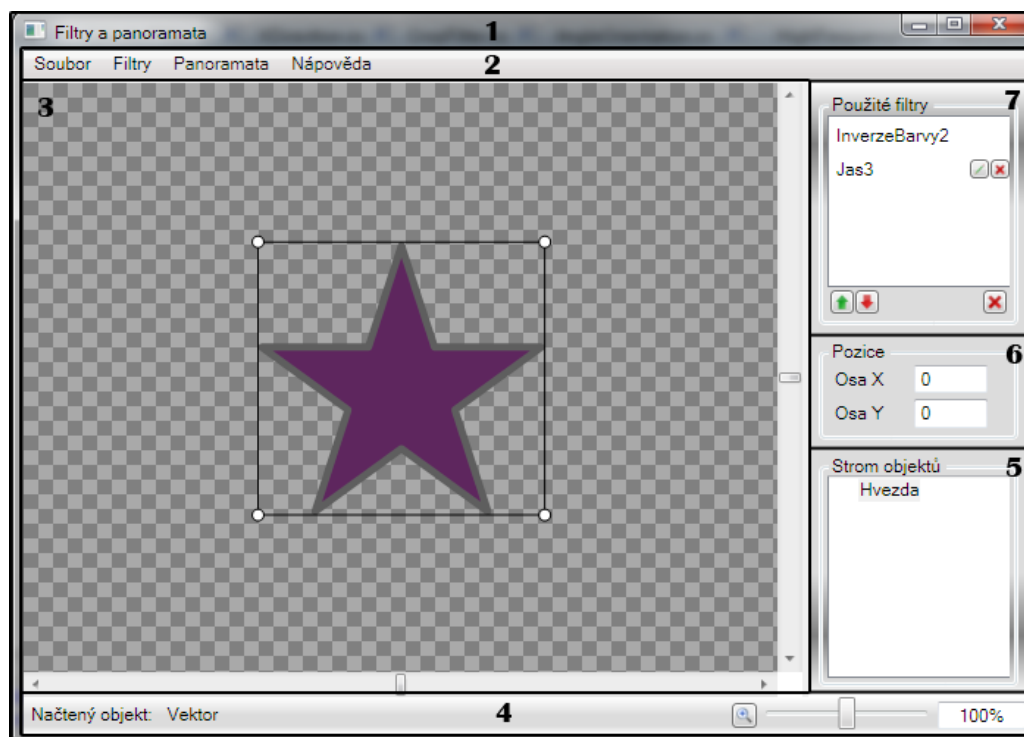
Jmenný prostor obsahuje mechanismus pro tvorbu dialogového okna pro tvorbu panoramat.

### **Age.Application.Controls**

Obsahuje rovněž grafické komponenty, z nichž je tvořeno GUI této práce. Rozdělení komponent do dvou knihoven je důsledkem toho, že práce [1] potřebovala používat komponenty obsažené v *Age.Application.Controls*, a nebylo možné je využívat přímo z *Age.Application.Dev.Filters*, protože některé z komponent této knihovny využívají komponenty z práce [1]. Rozdělením se zabránilo kruhovým referencím. Toto celkové rozdělení projektu AGE je prozatím a pro výslednou aplikaci bude sjednoceno.

## 2.6.1 Prostředí

Pracovní prostředí této práce je znázorněno na obrázku 2.7, který bude v této sekci využíván k popisu.



Obrázek 2.7: Obrázek zobrazuje hlavní okno grafického rozhraní.

### FiltersApplicationControl

Jak již bylo zmíněno, hlavní prostředí aplikace tvoří především tato komponenta, na kterou se odkazují všechny ostatní komponenty, z nichž je složena, které potřebují informace o scéně či editovaném objektu jako jsou reference na scénu či zpracovávaný nebo vybraný objekt.

### ScrollableViewport

Komponenta viz obrázek 2.7 část 3 patří do jmenného prostoru práce [1] *Age.Application.Dev.Editor2D*, umožňuje zobrazovat scénu s obsaženými objekty.

### FiltersMenu

Tato komponenta reprezentuje hlavní menu, viz obrázek 2.7 část 2., a drží referenci na komponentu hlavního okna *FiltersApplicationControl*. Pomocí toho může provést načtení scény ze souboru a jejího zařazení

do kontextu práce uživatele. Stejně tak, se dají načítat a ukládat rasterové objekty ze scény. Konkrétní implementace načítání či ukládání u obou případů je řešena třídami *BinarySaver* a *BitmapSaver* ze jmenného prostoru *Age.Core.Scene.Scene2D.Saving* práce [1]. Přidání objektů do scény lze také docílit tvorbou či editací panoramat pomocí spuštění příslušného dialogu, který poskytne daný objekt a menu ho přes referenci zařadí do běhu aplikace. Dále se v menu vytvářejí konkrétní filtry, které jsou poté přiřazeny vybranému objektu.

### **ZoomInformationPanel**

Komponenta viz obrázek 2.7 část 4. Již bylo zmíněno, že aplikace načítá scénu nebo objekty do scény. Ve skutečnosti neobsahuje scéna pouze prvky, které se načítaly. Vedle načtených prvků se do kořene stromu objektů scény přidává objekt *Group*, na který je aplikován filtr zoomování. Tento objekt ve stromě scény je uživateli skryt, ale ve své podstatě slouží k přibližování či oddalování uživatelovy práce ve scéně. Pro nastavení tohoto přiblížení slouží tato komponenta z níž je dále možno spustit dialog pro nastavení jmenovaného zoomovacího filtru. Hlavní komponenta aplikace dále do této komponenty ukládá informace o typu načteného objektu, který je pak následně zobrazen.

### **TreeComponent**

Komponenta viz obrázek 2.7 část 5. slouží k vybírání aktuálního objektu a zároveň zobrazuje strom objektů scénou bez zmíněného objektu pro přiblížení. Vybraný objekt nastaví komponenta při výběru do hlavní komponenty aplikace a ta se postará o to, aby byl objekt aktualizován v komponentě *PositionComponent* a *AppliedFilters*.

### **PositionComponent**

Tato komponenta viz obrázek 2.7 část 6. slouží ke změně pozice vybraného objektu.

## **2.6.2 Editace filtrů**

### **AppliedFilters**

Tato komponenta viz obrázek 2.7 část 7. slouží k editaci filtrů na vybraném objektu. Tato komponenta je grafickou reprezentací *FilterManager* daného objektu, lze zde měnit pořadí aplikace filtrů a nechtěné filtry smazat z *FilterManager*. Změny provedené na této kolekci se automaticky aktualizují pomocí rozhraní *INotifyCollectionChanged* a *INotifyPropertyChanged*, jež jsou součástí .NET.

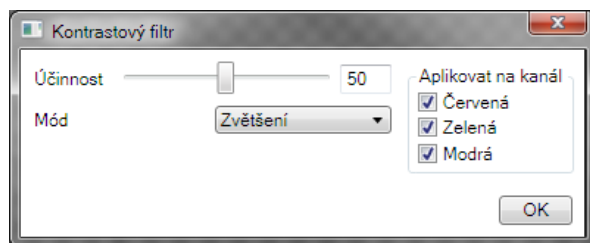
### **AppliedFilter**

Komponenta je grafickou reprezentací filtru. Při vytvoření této grafické

reprezentace se vytvoří skryté dialogové okno pro editaci konkrétního filtru a po stisknutí tlačítka se zviditelní. U *AppliedFilter* lze také měnit název filtru, jež bylo docíleno pomocí komponenty *EditableLabel* z knihovny *Age.Application.Controls*.

### FilterWindow

Dialogové okno viz. obrázek 2.8 je určeno pro editaci nastavení konkrétního filtru. Obsah tohoto okna se vytváří dynamicky dle toho, jaký typ filtru byl oknu předán a dle toho, jaké rozhraní třída tohoto filtru implementuje. Vytvářením obsahu se rozumí výběr komponent příslušné charakteristiky filtru ze jmenného prostoru *Age.Application.Dev.Filters.Components.Filters*. Mezi tyto komponenty patří různé slidery, comboxy s titulky, atd., které lze najít v programátorské dokumentaci.

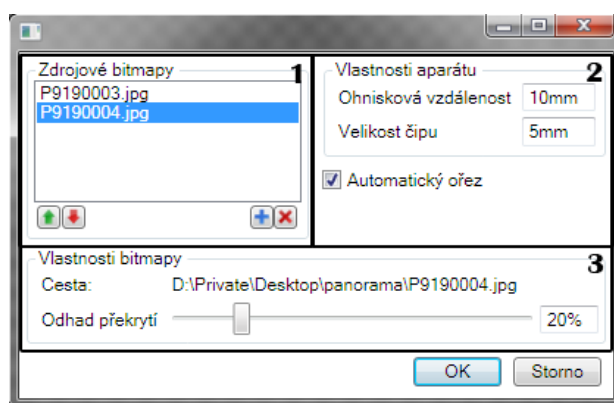


Obrázek 2.8: Obrázek příklad dialogového okna pro editaci kontrastového filtru.

## 2.6.3 Editace panoramat

### PanoramaWindow

Toto okno viz obrázek 2.9 obstarává funkcionalitu editace panoramat. Při vytváření tohoto okna předá komponenta menu *FiltersMenu* referenci na *PanoramaObject*, který se bude editovat, a okno si z něj načte potřebné informace. U předávaného panoramatu se může jednat o úplně nové prázdné panorama nebo panorama již zařazené do scény. Okno se skládá ze tří částí viz obrázek 2.9. První část načítá cesty a zobrazuje názvy obrázků, které budou pro tvorbu panoramata využity. Druhá část zajišťuje zadání informací o fotoaparátu a poslední část umožňuje zobrazovat a nastavovat vlastnosti obrázků, které budou důležité pro tvorbu panorama, jako je počáteční překrytí vůči předchozímu obrázku. Po potvrzení dialogu jsou všechny informace předány panoramatu a na něm je zavolán přepočít, který potrvá jistou dobu.



Obrázek 2.9: Obrázek zobrazuje dialogové okno pro tvorbu panoramat.

### ProgressBarWindow

Toto okno slouží k zobrazení stavu výpočtu tvorby panorama.

# Kapitola 3

## Srovnání s konkurencí

Tato kapitola obsahuje pojednání o tom, jak si produkt této práce stojí mezi již zaběhlou konkurencí, která má jistě velký náskok v tom, jak dlouho na ní probíhá vývoj. U aplikace se dá hovořit o délce doby vývoje v řádech měsíců, zatímco u konkurence se jedná o řády let. Dále je třeba uvážit, že se práci věnoval pouze jeden autor s využitím knihoven jednoho jiného autora, kdežto konkurenci tvoří větší vývojové týmy. I přesto budou aplikace srovnány dle následujících kritérií.

U jednotlivých produktů konkurence bude vždy srovnána rychlost operací oproti rychlosti operací v aplikaci této práce. Bude zde hodnoceno množství a zpracování grafických filtrů s ohledem na vektorovou a rastrovou grafiku. A mimo jiné bude u každé konkurenční aplikace popsáno, zda řeší tvorbu panoramat a opakovatelných grafických textur. V případě že ano, bude blíže specifikováno jakým způsobem.

### 3.1 Gimp

Tento grafický editor slouží k editaci rastrového obrázku. To znamená, že umožňuje úpravu nafocených fotografií, tvorbu rastrových obrázků, které lze získat pomocí vektorových nástrojů jako jsou Bazierovy křivky a podobně. Je volně šířen a napsán pod licencí *GPL*, jež umožňuje vývojářům z celého světa do tohoto projektu přispívat. Pomocí dialektu jazyka *Scheme* jsou definovány základní operace v aplikaci. To má za význam, že se dají do této aplikace psát "scripty", jež se poté dají do aplikace zařazovat jako dodatečné zásuvné moduly neboli "pluginy". Srovnání bude provedeno na verzi 2.6.7 viz [13].

V aplikaci lze nalézt všechny kategorie filtrů, jež jsou implementovány v aplikaci této práce. Při hledání se však musí dávat pozor, jelikož transformační filtry a filtry pro práci s barvou jsou reprezentovány pomocí nástrojů. Celkově

*Gimp* obsahuje mnohem více filtrů než aplikace této práce. Najdou se ale i detaily, které se v *Gimpu* nenachází jako například filtr pro nahrazení barvy barvou, filtr pro tvorbu opakovatelných grafických textur a jiné. Dále se u některých filtrů postrádá určitá funkcionalita jako je u odbarvovacího filtru možnost měnit intenzitu barvy a u vysokofrekvenčních filtrů možnost nastavit režim ostření. Zde se vysokofrekvenční filtry používají pouze pro detekci hran v obraze. Další nevýhodou *Gimpu* je bezesporu, že se po definitivní aplikaci filtru nedá filtr dále nastavovat. Nastavení se dá pouze měnit odstraněním filtru z historie změn a jeho opětovným přidáním. Tento postup může být docela nepříjemný, jelikož po aplikaci filtru mohou následovat i další operace. Může dojít k promíchání tohoto pořadí z důvodu, že se aplikace filtru jakožto změna na obrázku přidá vždy nakonec.

Jak už bylo řečeno v kapitole Analýza, *Gimp* umožňuje tvorbu panoramat pomocí "pluginu" *Pandora*, který vytváří panorama pomocí posouvání obrázků s prolnutím, jež neposkytuje příliš kvalitní výsledky.

Při porovnání rychlosti *Gimpu* a aplikace této práce vychází *Gimp* několikanásobně lépe. To je zapříčiněno pomalostí jednotlivých operací v aplikaci této práce.

## 3.2 Adobe Photoshop

Tento produkt by měl být absolutní špičkou mezi grafickými editory pro práci s rastrovou grafikou. Jeho vývoj má ve své kompetenci gigant v oblasti počítačové grafiky *Adobe Systems*. Vývoj na tomto produktu probíhá přes 20 let a byly do něj investovány nemalé finanční prostředky, jež měly motivovat pracovníky k intenzivnímu vývoji. Nyní aplikace existuje ve svém nejnovějším vydání *12.0 CS5*, s nímž se aplikace této práce bude srovnávat v její trialové verzi viz [14].

Funkcionalita transformačních filtrů je zde rozprostřena do ovládacích nástrojů a nastavení příslušných kreslicích vrstev, z nichž se skládá scéna aplikace. Podobná situace se vyskytuje u filtrů pro práci s barvou. S tím rozdílem, že funkční prvky těchto filtrů lze najít pouze u nastavení prolnutí příslušné vrstvy. Filtry jako takové zde vytvářejí složité grafické efekty, z nichž pouhým pohledem nelze vyčíst algoritmus. Konvoluční filtry jsou zde prezentovány pouze formou existence jednoho filtru určeného k detekci hran. Oproti aplikaci *Gimp* se zde vyskytuje o trochu méně filtrů. Pravděpodobně to bylo způsobeno tím, že se k testu použilo pouze trialové verze.

Nevýhoda této aplikace tkví v tom, že uživateli neposkytuje žádný mechanismus pro usnadnění práce při vytváření panoramat. Uživateli nezbyvá nic jiného než rozřadit obrázky do vrstev a manuálně je posouvat a prolínat.

Rychlost této aplikace a jejich jednotlivých operací lze považovat za nejlepší mezi rychlostmi uvažovaných aplikací.

### 3.3 Shrnutí

Pouhým pohledem je patrné, že na dvou konkurenčních aplikacích bylo provedeno více práce. To se promítlo na počet naimplementovaných filtrů. Přesto se našla i odvětví, ve kterých aplikace této práce předčila ostatní. Kamenem úrazu se však stala doba prováděných operací, která by měla být pro budoucí projekt *AGE* zkrácena. V porovnání s ostatními má editor této práce výhodu v tom, že se filtry dají aplikovat na rastr i vektor zároveň, jelikož ostatní editory dovedou upravovat jenom rastr. Níže se nachází tabulka charakterizující výsledek testu. Jednotlivé body jsou hodnoceny známkami jedna až čtyři. Čím nižší číslo, tím lepší hodnocení.

Program	Množství filtrů	Panoramata	Rychlost
AGE – Filtry a panoramata	2	2	3
Gimp	1	3	1
Adobe Photoshop	1	4	1

Tabulka 3.1: Souhrnná tabulka srovnání s konkurencí

# Závěr

V této práci se povedlo vytvořit grafický editor, jež umožňuje aplikovat běžné grafické filtry na vektorové či rastrové obrázky. Výsledná aplikace trpí jistými nedostatky.

První z nich je pomalost operací na rastrovém obrazu, jež byla dokázána v předchozí kapitole. Ta existuje i přesto, že na bitové mapě byl vytvořen mechanismus pro paralelní práci a všechny filtry jej naplno využívají. Jedním z možných faktorů je pomalost knihoven práce [1], s níž je aplikace této práce úzce spjata. K této myšlence vede fakt, že rychlost aplikace filtrů na vektor a následné překreslení vektorové scény neprobíhá příliš rychle. To však neopodstatňuje pomalost práce aplikace s bitmapou. Tato problematika si bude v budoucnu při vývoji grafického prostředí AGE žádat většího rozboru a návrhu řešení.

Dále i tvorba panoramat není zcela ideální. V případě, že uživatel zadá nevyhovující odhad rozestupu mezi jednotlivými obrázky, stane se, že se algoritmus vydá nesprávným směrem a obrázky se prolnou nevhodným způsobem. To potom nutí uživatele měnit pozice obrázků a nastavení filtrů ručně, nebo absolvovat zadání odhadu rozestupu a následného výpočtu znovu. Proto by nebylo od věci ještě jednou zvážit implementaci dvourozměrných panoramat, jež jsou plně automatizovaná.

Bezespору se dá aplikaci této práce vytknout, že při několikanásobném přiblížení scény nesedí graficky nástroje filtrů přesně tak, jak by měly. To je způsobné tím, že pozice nástroje se vypočítává s vektorovou přesností, kdežto při vykreslování bitmapy dojde zaokrouhlení pozice bitmapy na velikost jednotlivých pixelů. Tato odchylka se při přiblížení násobí určitým koeficientem větším než jedna, tudíž je na pohled patrná. Nástroje grafických filtrů byly vytvořeny nad rámec zadání pro zpříjemnění použití této aplikace a nejsou nepostradatelnou součástí této práce.

Jak již bylo popsáno v základním návrhu kapitoly Implementace a dále v celém obsahu této kapitoly, je aplikace této práce úzce spjata s některými částmi aplikace práce [1]. Tohoto faktu bude při dalším vývoji aplikace AGE využito a dojde k vytvoření projektu *Age.Application.Main*, který bude spouštět výslednou aplikaci viz obrázek 2.1.

# Literatura

- [1] ŠEBETOVSKÝ, Jan: *Vektorový grafický editor pro projekt AGE*, Praha: Univerzita Karlova, Matematicko-fyzikální fakulta, Katedra distribuovaných a spolehlivých systémů, 2010. Vedoucí bakalářské práce Mgr. Pavel Ježek
- [2] HERCEG, Tomáš: *Grafický editor 3D scén pro projekt AGE*, Praha: Univerzita Karlova. Matematicko-fyzikální fakulta, Katedra distribuovaných a spolehlivých systémů, 2010. Vedoucí bakalářské práce Mgr. Pavel Ježek
- [3] cmpg.org: *An Introduction to Color Space and Color Calibration*, [online], 2005 [cit. 2010-07-22]. <<http://www.cmpg.org/site/2005/07/26/an-introduction-to-color-space-and-color-calibration/>>
- [4] Wikipedia: *CMYK color model*, [online], 2010 [cit. 2010-07-22], <[http://en.wikipedia.org/wiki/CMYK\\_color\\_model](http://en.wikipedia.org/wiki/CMYK_color_model)>
- [5] BROWN, Matthew; LOWE, David G.: *Automatic Panoramic Image Stitching using Invariant Features*, [PDF soubor, online], Vancouver (Canada): Department of Computer Science, University of British Columbia, 2007 [cit. 2010-07-22]. <<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.68.7321&rep=rep1&type=pdf>>
- [6] Wikipedia: *Scale-invariant feature transform*, [online], 2010 [cit. 2010-07-22], <[http://en.wikipedia.org/wiki/Scale-invariant\\_feature\\_transform](http://en.wikipedia.org/wiki/Scale-invariant_feature_transform)>
- [7] Wikipedia: *kd-tree*, [online], 2010 [cit. 2010-07-22], <<http://en.wikipedia.org/wiki/Kd-tree>>
- [8] CHANG Yung-Yu: *Image stitching*, [PDF soubor, online], 2007-04-03 [cit. 2010-07-22]. <[http://www.csie.ntu.edu.tw/~cyy/courses/vfx/07spring/lectures/handouts/lec06\\_stitching\\_4up.pdf](http://www.csie.ntu.edu.tw/~cyy/courses/vfx/07spring/lectures/handouts/lec06_stitching_4up.pdf)>

- [9] shallowsky.com: *Pandora: a GIMP Plugin for Making Panoramas*, [online], 2007 [cit. 2010-07-22]. <<http://www.shallowsky.com/software/pandora/>>
- [10] SZELISKI, Richard: *Image Alignment and Stitching: A Tutorial1*, [PDF soubor, online], 2006-12-10 [cit. 2010-07-22], s. 2-4. <<http://research.microsoft.com/pubs/70092/tr-2004-92.pdf>>
- [11] Wikipedia: *Image scaling*, [online], 2010-06-03 [cit. 2010-07-22]. <[http://en.wikipedia.org/wiki/CMYK\\_color\\_model](http://en.wikipedia.org/wiki/CMYK_color_model)>
- [12] HLAVÁČ, Václav; SEDLÁČEK, Miloš: *Zpracování signálů a obrazu*. Praha: Vydavatelství ČVUT Praha, 2007 [cit. 2010-07-22], s. 191-210.
- [13] The GIMP Team: *GNU Image Manipulation Program*, [online], 2010 [cit. 2010-07-22]. <<http://www.gimp.org/>>
- [14] Adobe Systems: *Adobe Photoshop CS5*, [online], 2010 [cit. 2010-07-22]. <<http://www.adobe.com/cz/products/photoshop/compare/>>