

# 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

## NEREALISTICKÉ ZOBRAZENÍ VIDEOA

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. DANIELA JOHANNESOVÁ

BRNO 2011



**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

## **NEREALISTICKÉ ZOBRAZENÍ VIDEOA**

NON-REALISTIC VIDEO RENDERING

**DIPLOMOVÁ PRÁCE**

MASTER'S THESIS

**AUTOR PRÁCE**

AUTHOR

**Bc. DANIELA JOHANNESOVÁ**

**VEDOUCÍ PRÁCE**

SUPERVISOR

**Doc. Dr. Ing. PAVEL ZEMČÍK**

BRNO 2011

## **Abstrakt**

Tématem této práce je nerealistické zobrazení videa. Je zde uveden souhrn existujících technik nerealistického zobrazení a dále se práce soustředí na vybrané metody zpracování videa schopné pracovat v reálném čase. Pro efektivnější zpracování videa bylo využito akcelerace pomocí grafické karty za použití knihovny OpenGL a jazyka GLSL.

## **Abstract**

The aim of this thesis is non-realistic video rendering. It starts with a summary of existing techniques and then this thesis concentrates on selected methods that are able to work in real-time. To process video more effectively, we use acceleration on graphical processing unit with usage of OpenGL and GLSL

## **Klíčová slova**

Nerealistické zobrazení videa, real-time zpracování, shadery, abstrakce videa , filtry přizpůsobující se struktuře, zpracování videa,OpenGL

## **Keywords**

Non-photorealistic rendering of video, real-time processing, shaders, video abstraction, structure adaptive filtering, video processing, OpenGL

## **Citace**

Daniela Johannesová: Nerealistické zobrazení videa, diplomová práce, Brno, FIT VUT v Brně, 2011

# Nerealistické zobrazení videa

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Doc. Pavla Zemčika. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala

.....  
Daniela Johannesová  
25. května 2011

## Poděkování

Ráda bych poděkovala vedoucímu mé diplomové práce Doc. Pavlu Zemčikovi za odborné vedení.

© Daniela Johannesová, 2011.

*Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.*

# Obsah

<b>1</b>	<b>Úvod</b>	<b>3</b>
<b>2</b>	<b>Nerealistické zobrazování</b>	<b>4</b>
2.1	Definice . . . . .	4
2.2	Motivace . . . . .	4
<b>3</b>	<b>Přehled existujících metod nerealistického zobrazení videa</b>	<b>6</b>
3.1	Metody založené na zpracování obrazu . . . . .	7
3.2	Simulace malby . . . . .	8
3.3	Ostatní metody . . . . .	11
<b>4</b>	<b>Průběh zpracování videa</b>	<b>13</b>
4.1	Popis průběhu zpracování videa . . . . .	13
4.2	Filtrace obrazu závislá na jeho struktuře . . . . .	14
4.3	Barevná Paleta . . . . .	15
<b>5</b>	<b>Výpočet lokální struktury obrázku</b>	<b>17</b>
5.1	Edge tangent flow . . . . .	17
5.2	Strukturní tenzor . . . . .	18
5.3	Zobrazení struktury . . . . .	19
5.4	Zlepšení kvality odhadnuté struktury . . . . .	20
<b>6</b>	<b>Abstrakce videa</b>	<b>21</b>
6.1	Bilaterální filtr . . . . .	21
6.2	Kuwaharův filtr . . . . .	24
6.3	Shock-based filtrace . . . . .	26
6.4	Morfologické operace . . . . .	29
<b>7</b>	<b>Detekce hran</b>	<b>30</b>
7.1	LoG . . . . .	30
7.2	Rozdíl Gaussovských rozostření . . . . .	30
7.3	Orientovaný rozdíl Gaussovských rozostření . . . . .	31
<b>8</b>	<b>Popis řešení</b>	<b>33</b>
8.1	Vzorkování vstupního obrázku . . . . .	33
8.2	Průběh zpracování videa . . . . .	36
8.3	Vývojové prostředí . . . . .	37
8.4	Způsob práce s OpenGL . . . . .	38
8.5	Popis výsledné aplikace . . . . .	39

<b>9</b>	<b>Výsledky</b>	<b>40</b>
9.1	Časová náročnost algoritmů . . . . .	40
9.2	Grafické výstupy programů . . . . .	41
9.3	Reakce diváků . . . . .	44
<b>10</b>	<b>Závěr</b>	<b>45</b>
<b>A</b>	<b>Obsah přiloženého CD</b>	<b>48</b>

# Kapitola 1

## Úvod

Společně s rozvojem a neustálým zdokonalováním realistického zobrazování grafiky, kterého jsme v posledních letech svědky, dochází také k růstu zájmu o nerealistické zobrazování. Nerealistické zobrazování si klade za cíl vytvoření uměleckého dojmu, ať již se jedná o zobrazení 3D modelů, obrazových dat, či videa.

Nerealistické zobrazování může zahrnovat širokou paletu variant uměleckého zpracování. Může se jednat o simulaci existujících malířských technik jako je například akvarel, perokresba, kresba tuží apod., nebo o aplikaci jiných technik, které generují výstup podobný těmto uměleckým technikám.

Použité techniky se často velmi liší v závislosti na tom jaká data má daná technika zpracovávat. Techniky se dělí dle toho zda dokáží zpracovávat 3D modely, video či jen obrázky. Existuje jen velmi malá skupina metod, které dokáží zpracovávat všechny uvedené druhy vstupních dat. V této práci se budu zabývat skupinou technik nerealistického zobrazení sloužící ke zpracování právě videa.

Tato práce si klade za cíl zmapování zajímavých existujících technik nerealistického zobrazení videa a následně implementaci několika vybraných metod. Pro implementaci byly vybrány metody schopné zpracovat video v reálném čase, nebo se takové rychlosti alespoň přiblížit. Pro akceleraci zpracování bylo použito knihovny OpenGL a fragment shaderů.

V druhé kapitole této práce je podrobněji popsáno co to je nerealistické zobrazování a také jaké jsou důvody pro jeho použití. Poté ve třetí kapitole následuje stručný přehled existujících metod pro nerealistické zobrazení videa. V dalších kapitolách jsou popsány techniky zobrazení videa použité v této práci. Čtvrtá kapitola se zabývá základním pohledem na tuto problematiku. Poté jsou tři kapitoly zabývající se podrobně konkrétními etapami potřebnými pro nerealistické zobrazení videa: kapitola pátá popisuje možnosti zjištění lokální struktury obrázku, šestá kapitola způsob abstrakce videa a sedmá použité metody detekce hran. V osmé kapitole je popsáno navržené řešení a způsob jeho implementace a v deváté kapitole jsou diskutovány dosažené výsledky.

## Kapitola 2

# Nerealistické zobrazování

V této kapitole bude vysvětleno, co to vůbec je nerealistické zobrazování a dále také proč se vůbec nerealistickým zobrazováním zabývat. Informace uvedené v této kapitole je možné považovat za platné pro všechny skupiny nerealistického zobrazování, tady zobrazení 3D modelu, obrázků i videa.

### 2.1 Definice

Nerealistické zobrazování bývá také někdy nazýváno Nefotorealistic rendering, nebo spíše známějším anglickým názvem Non-photorealistic rendering a z toho odvozenou zkratkou NPR.

Nerealistické zobrazování není snadné definovat, protože dle názvu by sem měly spadat všechny techniky zobrazování neboli renderingu, jejich výsledkem je nerealistický vzhled výsledného zobrazení. Toto ovšem není pravda protože do nerealistického zobrazování spadají také některé modelační techniky, nebo techniky provádějící post-processing obrázků.

Také výraz nerealistické je problematický protože může mít jiný význam pro umělce a jiný význam pro výzkumníka zabývajícího se počítačovou grafikou. Pro umělce to může reprezentovat určitý umělecký směr, nebo malířskou techniku, pro výzkumníka v oblasti počítačové grafiky to může být cokoli co vypadá jinak než od reality.

Pokud tedy budeme chtít nerealistické zobrazení alespoň nějak definovat, tak se jedná o oblast počítačové grafiky, která je zaměřena na širokou paletu uměleckých stylů. Nerealistické zobrazování se inspirovalo olejomalbou, perokresbou, technickými ilustracemi, animovanými filmy apod.

### 2.2 Motivace

Nejčastějším důvodem vytváření metod nerealistického zobrazení je automatizované vytváření uměleckých efektů. Ať již se jedná o plně automatické metody pro nerealistické zobrazení nebo daná metoda slouží pouze jako pomoc pro umělce při časově či technicky náročných úkolech.

Další, autory často uváděnou, motivací pro použití nerealistického zobrazení, je usnadnění vizuální komunikace. Tohoto je využíváno například při tvorbě manuálů, kdy je celkový tvar lépe zachycen několika čarami, než původní fotografií. Tohoto bývá využíváno například také při zobrazování architektury, kdy je možné zobrazit danou budovu, tak aby byly

zdůrazněny ty správné prvky, které mohou při správném použití vést k nalákání potenciálního zákazníka, v tomto případě se tedy jedná o nerealistické zobrazení příslušného 3D modelu dané budovy.

Další často uváděný cíl nerealistického zobrazení, je zefektivnění vizuální komunikaci úpravou již existujícího obrazu či videa. Princip těchto metod spočívá v tom, že výrazné prvky ve videu či obraze jsou ještě více zvýrazněny a méně významné prvky jsou dosti zjednodušeny či úplně odstraněny.

Tímto by mělo být mimojiné možné dosáhnou toho, že divák se bude více soustředit na výrazné prvky v obrázku a nenechá se rozptýlit nevýznamnými detaily. Toto si klade za cíl například [24], kde byl součástí zveřejněné práce i průzkum rychlosti rozpoznání obrázků a to jak snadno si je lidé zapamatují. Výzkum dokázal, že použitá stylizace usnadňuje zapamatování a rozeznání obrázku. Jednou z možných aplikací takovéto metody je při zobrazování videa použité pro videokonference s využitím mobilních zařízení s malou obrazovkou.

Dalším často uváděným důvodem pro použití nerealistického zobrazení, je že realistické zobrazení je až příliš dokonalé a diváka to začíná nudit. Pro tento účel je nerealistické zobrazení ideální protože dokáže pozorovatele zaujmout. Nerealistické zobrazení dokáže zaujmout také z toho důvodu, že zobrazený objekt může vždy vypadat poněkud odlišně v závislosti na zvoleném způsobu zobrazení.

Hertzmann v [6] příkládá nerealistickému zobrazení obrázků a videa velký význam jelikož by mohlo přispět k poznání fungování lidského vidění. Nerealistické zobrazování může umožnit zjistit proč funguje umění a jak je zpracováváno lidským mozkiem. Jak sám autor uvádí: Jak může několik čar a fleků nakreslených na papíře vyjadřovat pohyb, tvar, náladu nebo emoce?. Pokud by se na tuto otázku podařilo odpovědět, tak by bylo možné mnohem efektivněji navrhovat algoritmy nerealistického zobrazování, ale také by to jistě znamenalo velký pokrok ve výzkumu lidského vidění.

## Kapitola 3

# Přehled existujících metod nerealistického zobrazení videa

Cílem této kapitoly je stručně shrnout existující metody sloužící k nerealistickému zobrazení videa. Do současné doby již bylo navrženo obrovské množství metod jak dosáhnout nerealistického zobrazení a proto jejich výčet v této práci nemá za cíl být vyčerpávajícím seznamem metod, ani to vzhledem k rozsahu práce není možné.

Výběr popsaných metod byl částečně subjektivní volbou autora práce a částečně byl výběr metod řízen jejich 'významností', tj. jak často byly dané metody citovány a rozšiřovány jinými autory.

Ještě výčtem samotných metod zobrazení videa je nutné vysvětlit čím se liší od metod pro nerealistické zobrazení jednotlivých obrázků. Nejjednodušším způsobem jak by bylo možné vytvořit nerealistické zobrazení videa je aplikovat metody pro nerealistické zobrazení obrázku na jednotlivé snímky bez ohledu na konkrétní metodu pro zobrazení a bez ohledu na to o jaké se jedná video. To může vést k velkým změnám ve výstupním videu, tam kde ve vstupním videu jsou jen malé změny, což je divákem vnímáno jako nežádoucí blikání rušící průběh videa. Jinak je možné toto blikání popsat také jako oblasti, které se ve vstupním videu jeví jako (téměř) statické a ve výstupním videu jsou v každém snímku vykresleny jinak. Takové blikání je považováno za nežádoucí, z toho důvodu, že odvádí pozornost diváka od toho co se děje v samotném videu a také samozřejmě dochází k degradaci výstupního videa z estetického hlediska.

Metody nerealistického zobrazení videa je možné rozdělit do skupin na základě různých kritérií, jsou to například dělení dle těchto kritérií:

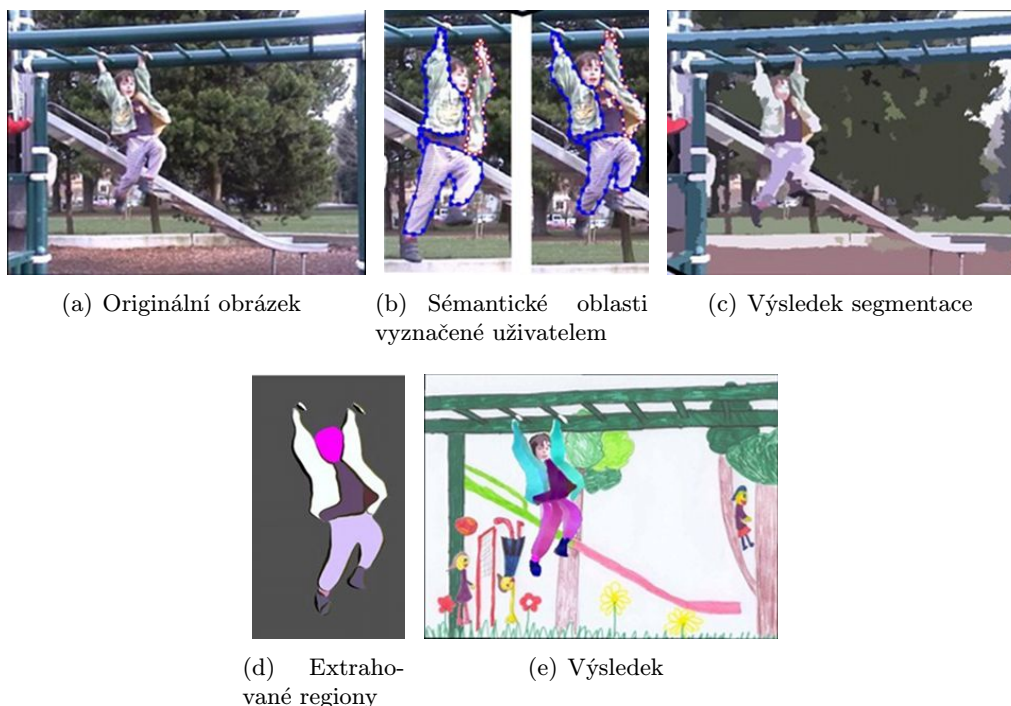
1. Metody založené na simulaci malby vs. metody založené na zpracování obrazu
2. Zda je nutné při vykreslování aktuálního snímku brát v úvahu i předchozí snímky, nebo zda je možné snímky zpracovávat jednotlivě a výsledné video neobsahuje artefakty již z principu fungování té dané metody.
3. Časová náročnost - existují některé metody, které zvládají zpracovávat video v reálném čase, jiné potřebují i jednotky minut na zpracování jednoho snímku
4. Dělení na metody, které vyžadují interakci uživatele a metody, které pracují automaticky

### 3.1 Metody založené na zpracování obrazu

Tyto metody používají pro vytvoření nerealistického zobrazení pouze metody zpracování obrazu. V této kapitole budou popsány pouze některé metody, metody konkrétně použité v této práci budou podrobně popsány v kapitole 4.

Existuje zde významná skupina metod, které vstupní video zpracovávají jako 3D volumetrická data. Tyto metody často dosahují výborných výsledků, ale pochopitelně jsou také velmi časově náročné. Jednou z nich je například metoda popsaná v [21]. Princip této metody je následující:

- Uživatel označí významné regiony ve vybraných klíčových snímcích 3.1(b),
- Vyznačené regiony jsou interpolovány i v ostatních snímcích pomocí techniky založené na segmentaci metodou mean-shift 3.1(c),
- Následně jsou extrahovány sémanticky významné regiony 3.1(d),
- Extrahovaným regionům může být definována barevná výplň a uživatel může zvolit jiné pozadí 3.1(e).



Obrázek 3.1: Etapy algoritmu VideoTooning, zdroj [21]

Jiná metoda zpracovávající 3D volumetrická data byla uvedena v [25]. Tato metoda také pracuje s celým 3D volumetrickým objemem obrazových dat, ovšem zároveň je velmi efektivní. Při použití akcelerace této metody na GPU je schopná zpracovávat video v reálném čase.<sup>1</sup>

<sup>1</sup> Zpracování v reálném čase v případě nerealistického renderingu videa obvykle nepředstavuje FPS rovnu 30, tak jako třeba při natáčení videa, ale méně. Obvykle se jako ideální uvádí pro nerealistické zobrazení použít rychlost snímkování v rozmezí 10-15 FPS [7]

Princip této metody je poměrně jednoduchý:

1. Nejprve je zjištěna lokální orientace ve 3D objemových datech, zjištění orientace vychází z detekce hran a výsledkem je vektorové pole určující každému pixelu směr nejmenší změny v jeho nejbližším okolí. Je zde použita metoda popsána v 5.1, ale je zobecněna do 3D.
2. Následuje iterativní simulace pohybu částic v prostoru, to přes jaké pixely (jaké barevné hodnoty) částice putovala určí její výslednou barvu. Počet iterací, určující kolikrát je provedena simulace částic, zde určuje jak moc se výsledek změní, tj. jak moc bude abstraktní.



(a) Originální ob- (b) Výsledek po (c) Výsledek po  
rázek malém počtu větším počtu  
iterací iterací

Obrázek 3.2: Ukázky výstupu pomocí metody [25]

## 3.2 Simulace malby

Tato skupina metod je založena, jak již název napovídá, na různých způsobech jak je možné simulovat malbu, obvykle se jedná o simulaci pomocí různých metod umístování jednotlivých tahů štětcem do obrázku, které po vykreslení dohromady vytvoří výslednou malbu. Tah štětcem si můžeme představit jako datovou strukturu uchovávající potřebné informace pro jeho vykreslení. Potřebné informace se liší dle konkrétní metody, ve všech metodách je to alespoň pozice v obrázku, dále bývá často ukládána orientace, barva, délka a šířka, textura apod.

Rozdíl mezi těmito metodami a metodami uvedenými v 3.1, není ten, že by zde metody zpracování obrazu nebyly vůbec používány, ale jsou používány pouze jako podpůrné, stěžejní je právě výše zmíněné umístování a vykreslování tahů štětcem.

Pravděpodobně první metoda simulace malby použitelná pro video, která se dostala do obecného povědomí byla uvedena v [16]. Simulace malby je zde prováděna následujícím způsobem:

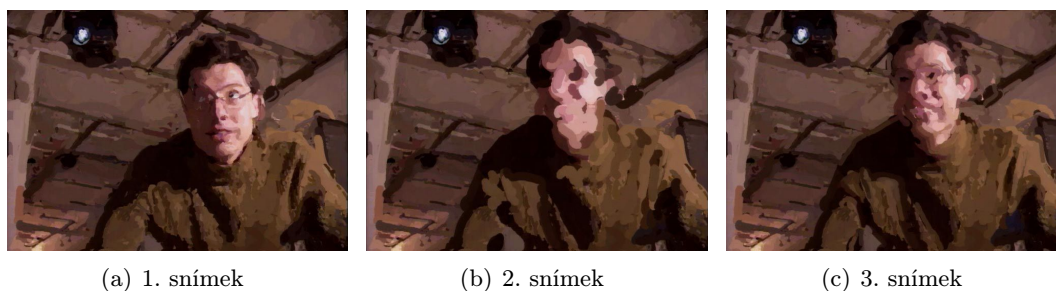
- Pozice, kam budou umístěny tahy štětcem jsou v prvním obrázku určeny pravidelnou mřížkou, tj. mezi tahy jsou konstantní mezery,

- orientace tahů štětce je určena gradientem v obrázku, zjištěném pomocí Sobelova operátoru,
- Barva tahu štětce je určena barvou původního obrázku, v pozici kam je tah štětce umístěn
- V obrázku jsou detekovány hrany pomocí Cannyho operátoru a délky tahů štětce jsou upraveny, tak aby končily v místě, kde je detekována hrana (tam kde je to možné)
- Jednotlivé tahy štětce jsou následně vykresleny jako vyhlazené čáry příslušné barvy.
- V případě zpracování videa je třeba zajišťovat koherenci mezi snímky, tj. aby přechody mezi snímky byly pokud možno co nejvíce plynulé. Zde je plynulost přechodu zajištěna tím způsobem, že všechny tahy jsou vygenerovány pouze v případě prvního snímku a u dalších snímků jsou tahy štětce posunuty pomocí zjištěného optického toku. Při takovém posunutí vznikají v obrázku oblasti, které nejsou dostatečně pokryty tahy štětce a naopak oblasti, kde je tahů štětce příliš velké množství a zbytečně je zvyšována výpočetní náročnost. V nedostatečně pokrytých oblastech jsou tedy vygenerovány nové tahy štětce pomocí Delaunayho triangulace a ve zbytečně hustě pokrytých oblastech jsou některé tahy odstraněny na základě vzdálenosti k sousedním tahům štětce.

Na podobném principu pracuje metoda popsaná v [7]. Rozdíly mezi těmito metodami jsou následující:

- Umístění tahů štětce není určováno pomocí pravidelné mřížky a optického toku, ale jsou umísťovány tam kde je největší rozdíl mezi aktuálním "plátnem" a vykreslovaným obrázkem. Plátno je zde obvykle reprezentováno obrázkem v paměti, kam již byly umístěny všechny dosud vygenerované tahy,
- Tahy štětce jsou umísťovány v několika vrstvách. Nejprve jsou umístěny tahy štětce s největším průměrem, následně tam kde je stále velký rozdíl mezi plátnem a původním vykreslovaným obrázkem jsou umísťovány tahy štětce další vrstvy s menším průměrem
- Metoda se liší ve způsobu zachování koherence mezi navazujícími snímky. Opět podobně jako v předchozí metodě je plně vykreslen pouze první snímek, ten je uložen v paměti a při zpracování dalšího snímku jsou umístěny nové tahy štětce pouze tam kde je rozdíl mezi vstupním obrázkem a předchozím vykresleným obrázkem velký. Aby bylo možné zachytit také pomalé změny ve videu (tj. rozdíl mezi dvěma po sobě následujícími snímky není příliš velký, ale změna za více snímků je již výrazná), tak je chyba akumulována a při vykreslení nového tahu štětce se v daném místě vynuluje.
- Tahy nejsou vykreslovány pouze jako přímky, ale místo toho mají podobu Bézierovy křivky.

Tyto dvě metody poskytují poměrně dobré výsledky, pokud mají na vstupu video ve kterém nedochází k prudkým změnám. Pokud dojde k výrazné změně ve vstupním videu, tak tyto metody to již z principu svého fungování nemohou příliš dobře zvládnout. Ukázka jak vypadá výstup takové metody je na obrázku 3.3, kde jsou zobrazeny tři po sobě následující snímky videa.



Obrázek 3.3: Ukázka nerealistického zobrazení videa pomocí simulace malby, zdroj [7]

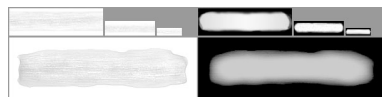
Další známou metodou simulace malby je [5].

Její princip je následující:

- Tahy štětcem nejsou vykresleny pouze jako vyhlazené čáry příslušné barvy, ale jsou reprezentovány texturou a maskou učující neprůhlednost. Zjednodušeně řečeno, při vykreslování tahu štětcem je vždy textura vynásobena touto barvou a následně je aplikována maska, která vytvoří pozvolný přechod mezi daným tahem štětce a zbytkem obrázku. Textura reprezentující tah štětce může vypadat například jako na obrázku 3.4.
- Podobně jako v předchozí metodě jsou tahy štětcem umísťovány v několika vrstvách (v článku [5] jsou použity 4 vrstvy). Rozdílem je to kam jsou tahy v jednotlivých vrstvách umísťovány. První vrstva s nejširšími tahy štětce je umístěna v rozsahu celého obrázku. Další vrstvy jsou umísťovány do okolí detekovaných hran.
- Rozdílný je také způsob určení orientace tahu štětcem. Podobně jako v předchozích metodách je využito směru gradientu. Ovšem gradient obrázku uvažujeme pouze tam kde je jeho velikost vysoká. Orientace tahů v místech vysokého gradientu je následně interpolována pomocí RBF a tím se určí orientace i v místech ostatních tahů štětce.
- Pro zachování koherence mezi jednotlivými snímky je opět využit optický tok. Pomocí něj jsou jednotlivé tahy štětce posunuty a je upravena jejich orientace a délka. Velikost změny délky či úhlu, která je možná mezi jednotlivými snímky, je omezená volitelným parametrem. Obvykle je možná změna délky maximálně o jeden pixel a změna úhlu maximálně o jeden úhlový stupeň. Tam kde vzniknou mezery jsou opět umístěny nové tahy štětce. Nové tahy štětcem se objevují postupně, tzn. nejprve jsou do snímku umístěny tahy štětce s velkou průhledností a následně se jejich průhlednost snižuje, obdobně při odstraňování nepotřebných tahů se postupně stávají průhlednými a až jsou úplně průhledné tak jsou odstraněny.
- Pořadí tahů štětce je mezi jednotlivými snímky zachováno, nové tahy štětce jsou umístěny na konec fronty.

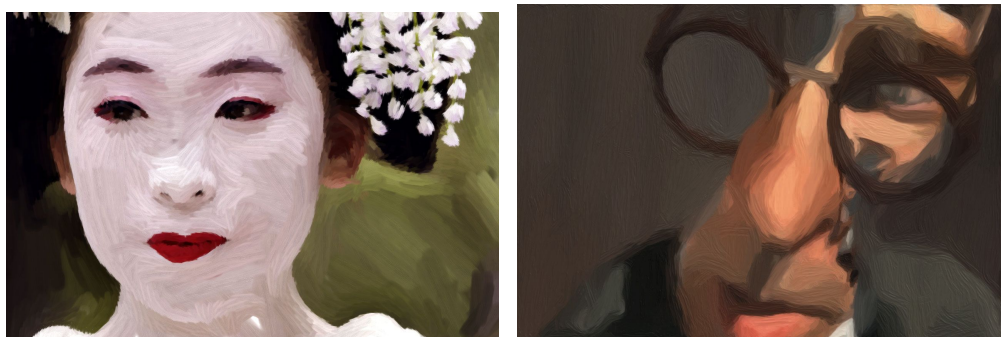
Tyto metody jsou dále zdokonalované a rozvíjené jako například metoda popsaná v: [18].

Stručný princip této metody je:



Obrázek 3.4: Textury a maska reprezentující tah stětce

- Tahy stětce jsou opět reprezentovány Bézierovou křivkou, ovšem v tomto případě je na křivku namapována textura.
- Metoda vyžaduje interakci s uživatelem
- Tvary tahů stětce mezi jednotlivými snímky jsou umísťovány pomocí minimalizace energie



(a) Snímek vykreslený pomocí metody [5]      (b) Snímek vykreslený pomocí metody [18]

Obrázek 3.5: Ukázka nerealistického zobrazení videa pomocí malby, zdroj [5] a [18]

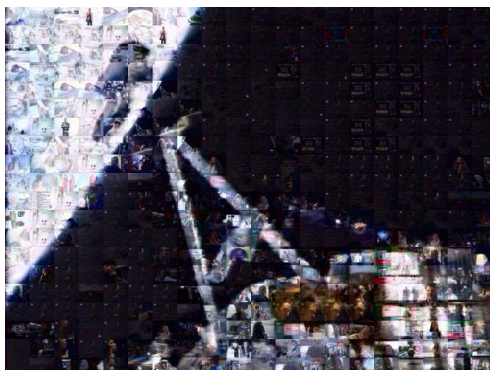
### 3.3 Ostatní metody

Zajímavou metodou pro nerealistické zobrazení, ktero lze těžko zařadit mezi ty předchozí je [11]. Principem této metody je to, že se z množiny zdrojových videí snaží vytvořit video co nejvíce podobné zpracovávanému videu. Zpracovávané video je tedy ve výsledku tvořeno z množiny malých videosekvencí vytvořených ze zdrojových videí. Tyto musí být samozřejmě poněkud upraveny, aby co nejvíce odpovídaly zpracovávanému videu, zároveň je zde snaha o zachování charakteristiky původního zdrojového videa.

Fungování metody je tedy následující:

- Máme množinu zdrojových videí a zpracovávané video, všechny videa jsou rozděleny na dlaždice
- Každá dlaždice je zpracována vlnkovou dekompozicí, jejíž výsledek slouží později pro porovnání míry podobnosti mezi jednotlivými dlaždicemi.
- Každé dlaždici zpracovávaného videa je přiřazena jedna dlaždice ze zdrojového videa.

- Na závěr proběhne barevná korekce zdrojové dlaždice, tak aby se co nejvíce podobala dlaždici ve zpracovávaném videu a zároveň, aby si zachovala svou původní tvarovou charakteristiku



Obrázek 3.6: Video mozaika [11]

Jiná zajímavá metoda je metoda napodobující akvarel v [2]. Tato metoda používá morfologických operátorů pro získání abstraktnější podoby videa (bude popsáno podrobněji dále). Dále je zde využita textura reprezentující papír, která je postupně deformována na základě vypočítaného optického toku.

## Kapitola 4

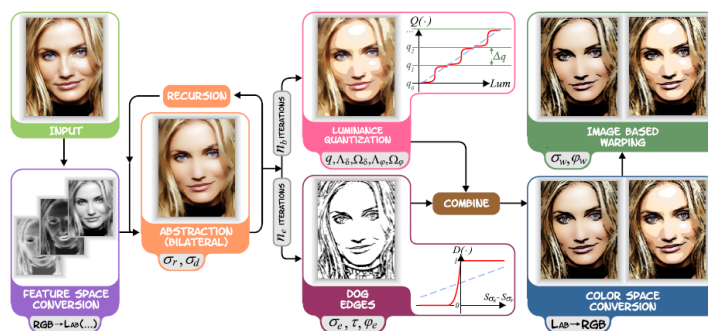
# Průběh zpracování videa

Tato kapitola popisuje průběh zpracování videa, který sloužil jako základ pro zobecněný postup použitý v této práci. Metody nerealistického zobrazení videa byly vybrány na základě výpočetní efektivity a zároveň samozřejmě také estetické kvality. V metodách vybraných pro implementaci není nutné uvažovat žádná interakci mezi snímky, každý snímek videa může být zpracován, jako by se jednalo o samostatný obrázek. Pro získání kvalitního výstupního videa bez blikání je dostačující vhodné nastavení parametrů.

Samozřejmě i u těchto metod bude možné nalézt videa, pro která metody selžou, například při velmi prudkých změnách, ovšem taková videa nejsou příliš obvyklá a navíc si s nimi neporadí téměř žádná metoda nerealistického zobrazení.

V následující podkapitole 4.1 bude popsán průběh zpracování videa. To se skládá obecněji z několika etap, bude zde posán způsob jakým na sebe tyto etapy navazují. .

### 4.1 Popis průběhu zpracování videa



Obrázek 4.1: Průběh zpracování videa [24]

Na obrázku 4.1 je naznačen průběh zpracování jednoho snímku videa. Z tohoto způsobu zpracování bude vycházet i tato práce.

V této kapitole budou stručně popsány jednotlivé etapy. Podrobnosti jednotlivých metod realizujících tyto etapy budou popsány v následujících kapitolách. V následujících kapitolách budou také popsány některé metody, které v původním průběhu zpracování nejsou a které mají za cíl průběh zpracování videa rozšířit.

Snímky původního videa jsou vesměs zpracovávány jednotlivě, tj. předchozí snímky neovlivňují průběh zpracování aktuálního snímku.

Jako první je proveden převod videa z barevného prostoru *RGB* do prostoru *Lab*. Toto je v případě aplikace bilaterálního filtru velmi důležité protože při aplikaci bilaterálního filtru v *RGB* může docházet ke vzniku barevných artefaktů. Barevný prostor *Lab* je také užitečný při provádění detekce hran, jelikož detekce může probíhat na prvním barevné složce obrázku *L* a není nutné obrázek převádět na šedotónový.

### **Abstrace obrázku**

Tato etapa je pravděpodobně nejvýznamnější, jelikož jsou na ní závislé všechny další metody.

Cílem abstrakce obrázku je jeho zjednodušení a zároveň zachování, nebo i zdůraznění, vizuálně významných prvků. Asi nejčastěji používanou metodou pro abstrakci obrázku je iterativní aplikace bilaterálního filtru.

Podrobněji budou popsány použité metody pro provedení abstrakce jednotlivých obrázků v kapitole 6.

### **Detekce hran**

Detekce hran dostává jako svůj vstup obrázek, který prošel jen malým počtem filtrací bilaterálním filtrem, aplikace detektoru hran je obvykle prováděna hned po první iteraci bilaterálního filtru.

Přidání hran do výstupního obrázku umožní lokálně zvýšit kontrast a tím podtrhnout dojem animovaného videa či obrázku.

Detekce hran může být provedena pouze na první barevné složce obrázku, jelikož obrázek je stále v barevném modelu *Lab*. 7.

### **Kvantizace**

Slouží pro zdůraznění kresleného vzhledu, pokud je prováděna. Tato část zpracování také profituje z toho, že daný obrázek je v barevném modelu *Lab*, jelikož stačí provést kvantizaci jen na jeho první složce *L*.

### **Zkombinování výstupu**

Sloučení abstrahovaného obrázku a hran, po provedení sloučení je nutné převést obrázek z barevného modelu *Lab* do *RGB*.

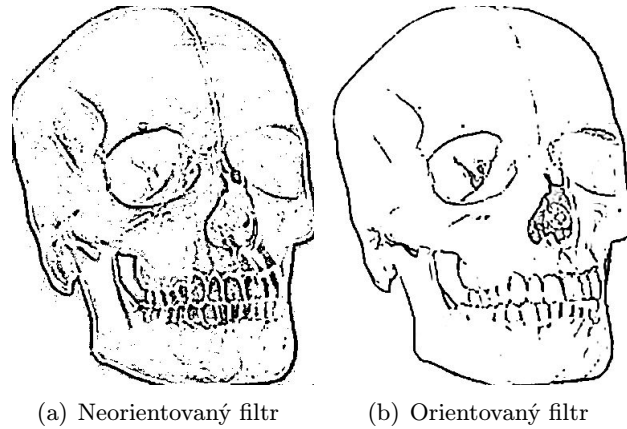
## **4.2 Filtrace obrazu závislá na jeho struktuře**

Cílem této podkapitoly je vysvětlit, proč je užitečné zjišťování struktury obrázku, které bylo popsáno v kapitole 5. Podrobněji bude způsob extrakce popsán v samostatné kapitole 5.

Zjištěná struktura umožňuje lépe provádět filtraci obrazu, tak že aplikovaný filtr je vždy orientován dle této lokální struktury. Lokální orientování prováděné filtrace obrazu vede k výrazně lepším výsledkům, než aplikace neorientovaného filtru.

Příkladem jak kvalitní je výstup při použití orientovaného filtru je obrázek 4.2, kde pro detekci hran na obrázku vlevo byl použit neorientovaný detektor hran (použitý detektor

hran je rozdíl Gaussovských rozostření - zkráceně označovaný jako DoG), při zpracování obrázku vpravo byl použit stejný filtr, pouze orientovaný dle lokální struktury obrázku.



Obrázek 4.2: Rozdíl mezi výsledky detekce hran

To jakým způsobem jednotlivé metody využívají zjištěnou lokální orientaci je konkrétněji rozebráno u daných metod.

### 4.3 Barevná Paleta

Jako volitelný krok nerealistického zobrazení videa je možné převést jednotlivé snímky s využitím barevné palety, tak aby byl počet barev v obrázku redukován. Tento proces bývá také nazýván jako kvantování barev, či kvantizace. Provedení kvantování barev vede k tomu, že výsledné obrázky se více podobají kreslené podobě.

Pro provedení kvantizace není možné použít pevně definovanou paletu, kterou by bylo možné použít u samostatného obrázku, jelikož taková metoda by vedla určitě k blikání ve videu, pokud by se odpovídající pixely ve videu lišily jen málo, ale byly by jim přiřazeny po aplikaci barevné palety rozdílné barvy.

Nejvhodnější barevný prostor pro provedení kvantizace obrázku je *Lab* a následně je samotná kvantizace prováděna pouze na první barevné složce, tj. na složce reprezentující jas obrázku.

Asi nejčastěji používaná metoda pro kvantizaci videa je uvedena v článku [24].

Principem této metody je, že přechody mezi jednotlivými úrovněmi jasu nejsou úplně ostré, ale jsou mezi nimi částečně vyhlazené přechody. Tyto přechody jsou definovány funkcí *tanh* a uživatelsky definovaným parametrem. Ostré přechody mezi úrovněmi jasu by byly vhodné pro zpracování samostatných obrázků, ale jsou nevhodné pro zpracování videa. Při ostrých přechodech mezi úrovněmi jasu dochází k nepříjemnému blikání pokud určitá oblast v jednom snímku náleží do jiné úrovně než v dalším snímku.

$$Q(\hat{x}, q, \varphi_q) = q_{nearest} + \frac{\Delta q}{2} \tanh(\varphi_q \cdot (f(\hat{x}) - q_{nearest})) \quad (4.1)$$

Výsledkem je tedy pseudo-kvantovaný obrázek  $Q(\cdot)$ ,  $\Delta q$  je šířka jasu spadající do jedné úrovně,  $q_{nearest}$  je úroveň nejbližší k  $f(\hat{x})$  a  $\varphi_q$  je parametr kontrolující ostrost přechodů mezi úrovněmi jasu. Rovnice 4.1 je tedy formálně nespojitá, ale pro vysoké hodnoty  $\varphi_q$  jsou tyto nespojitosti neznatelné.

Pro konstantní  $\varphi_q$  je ostrost mezi přechody nezávislá na samotném obrázku a může docházet k tomu, že vznikají výrazné přechody v rozsháhlých regionech bez výrazných změn barvy. Pro minimalizaci těchto nežádoucích efektů definujeme  $\varphi_q$  jako funkci velikosti gradientu v jasové složce obrázku. Tam kde je velikost gradientu vysoká bude vysoká i hodnota  $\varphi_q$  a budou zde ostré přechody mezi hranami, naopak tam kde je malá velikost gradientu nízká hodnota  $\varphi_q$  a přechody mezi oblastmi budou více vyhlazené.

Další možností jak provést kvanizaci je využít bilaterální mřížku popsanou v podkapitole 6.1. Základní princip této metody zůstává stejný pouze v při získávání výsledného obrázku bude výsledná hodnota pixelu určena nejbližší hodnotou intenzity a poté bilineární interpolací v prostorových souřadnicích. V rozměru určujícím intenzitu není na rozdíl od původní verze provedena interpolace.

Pro kvanizaci barev je také možné využít segmentaci obrazu, např. pomocí metody mean-shift. Ovšem pro zachování časové koherence videa je nutné provádět segmentaci ve 3D což je velmi časově náročné.

Segmentace videa schopná pracovat v reálném čase byla ukázána v [26], kde je pro zachování časové koherence třeba pracovat pouze s aktuálním a předchozím snímkem.



(a) Po filtraci bilaterálním filtrem

(b) Po aplikaci palety

Obrázek 4.3: Aplikace palety na filtrovaný obrázek

## Kapitola 5

# Výpočet lokální struktury obrázku

V této části práce budou popsány některé metody pro zjištění lokální struktury obrázku. Lokální struktura obrázku je reprezentována vektorovým polem, kde každý z pixelů má přiřazený vektor reprezentující směr nejmenší změny v obraze, tj. můžeme říct, že se jedná o vektory tečné k hranám v obraze. Z tohoto vektorového pole je možné snadno zjistit naopak směr maximální změny v obrázku, což jsou vektory kolmé k vektorům v tomto vektorovém poli.

Pro zjištění lokální struktury existují dvě metody, první z nich popsaná v podkapitole 5.1 je o něco starší a výpočetně náročnější a proto byla v této práci použita pouze metoda uvvedená v 5.2.

Pokud je zjištěna lokální struktura obrázku, tak je následně umožněno lépe přizpůsobit prováděnou filtraci na základě právě lokální struktury v místě daného pixelu.

Provádění samotné filtrace pomocí lokální orientace bude podrobněji popsáno dále, ale zde je možno ve stručnosti říci, že takto zjištěné vektorové pole určuje orientaci 1D filtru. Např. při použití bilaterálního nebo Gaussova filtru jsou tyto filtry orientovány ve stejném směru jako je směr určený tímto vektorovým polem. Naopak při detekci hran je filtr orientován ve směru kolmém k vektorům ve zjištěném vektorovém poli. Následná filtrace je prováděna s pixely ležícími pod takto orientovaným 1D filtrem stejným způsobem jako u běžného způsobu filtrace.

### 5.1 Edge tangent flow

Název této metody nemá přesný český překlad, pravděpodobně by bylo možné název přeložit jako 'tečný hranový tok', ale v této práci se budu raději držet anglického názvu.

Tato metoda byla představena v článku [10]. Jak už bylo řečeno výše, účelem této metody je konstrukce vektorového pole, kde každý pixel v obrázku má přiřazený svůj vektor, značící směr nejmenší změny v obrázku. Tento vektor je přibližně kolmý na gradient obrázku, ale kromě toho musí splňovat tyto tři podmínky:

- Vektorové pole musí reprezentovat orientaci významných hran v okolí každého pixelu,
- Sousední vektory musí mít podobnou orientaci s výjimkou ostrých rohů,
- V místě významných hran v obrázku musí být zachován původní směr.

Vektorové pole je nejprve zkonstruováno z vstupního obrázku  $I(x)$ , kde  $\mathbf{x} = (x, y)$  označuje pixel v obrázku. Definujeme zde tangentu k hraně  $t(x)$  jako vektor kolmý k obrazovému gradientu.

Edge tangent flow je tedy následně zkonstruován iterativně dle těchto rovnic:

$$t^{new}(\mathbf{x}) = \frac{1}{k} \sum_{\mathbf{y} \in \Omega(\mathbf{x})} \Phi(\mathbf{x}, \mathbf{y}) t_{cur}(\mathbf{y}) w_s(\mathbf{x}, \mathbf{y}) w_m(\mathbf{x}, \mathbf{y}) w_d(\mathbf{x}, \mathbf{y}) \quad (5.1)$$

kde  $\Omega(x)$  označuje okolí pixelu  $\mathbf{x}$ , výpočet tedy představuje váhovaný průměr mezi vektory v okolí daného pixelu

$$w_s(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{pokud } \|x - y\| < r \\ 0 & \text{jinak} \end{cases} \quad (5.2)$$

Následující dvě váhovací funkce  $w_m$  a  $w_d$  hrají klíčovou roli v tom, aby zůstaly zachovány významné významné tvary v obrázku.

Funcke  $w_m$  dává větší váhu bodům, kde je velikost gradientu větší než velikost v daném bodě  $\mathbf{x}$ . Toto zajistí zachování významných hran v obraze.

$$w_m(x, y) = \frac{1}{2} (1 + \tanh[\eta(\hat{g}(y) - \hat{g}(x))]) \quad (5.3)$$

$$w_m(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{pokud } \|x - y\| < r \\ 0 & \text{jinak} \end{cases} \quad (5.4)$$

Funcke  $w_d$  slouží k zachování původní orentace, dává větší váhu těm vektorům, které jsou více podobné vektoru v bodě  $\mathbf{x}$ . Vektory vstupující do této funkce musí být normalizované.

$$w_d(x, y) = |t^{cur}(x) \cdot t^{cur}(y)| \quad (5.5)$$

$$\Phi(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{pokud } t^{cur}(x) \cdot t^{cur}(y) > 0 \\ -1 & \text{jinak} \end{cases} \quad (5.6)$$

Funcke  $\Phi$  slouží k tomu, že pokud je úhel mezi vektory v bodech  $\mathbf{x}$  a  $\mathbf{y}$  větší než  $90^\circ$  tak je vektor v bodě  $\mathbf{y}$  otočen, tak aby se vektory navzájem nevyrušily.

## 5.2 Strukturní tenzor

Použití strukturního tenzoru (angl. 'structure tensor') pro výpočet lokální orientace v obrázku byla použita mimo jiné v [13], odkud bylo čerpáno i v této práci.

Lokální orientace v obrázku určená touto metodou je velmi podobná výstupu metody popsané v 5.1, ale její výpočet je výrazně efektivnější, jelikož není nutné provádět několik iterací přes celý obrázek. Odhad lokální orientace je založen na vlastních číslech strukturního tenzoru v místě daného pixelu.

Abychom využili kompletně informaci ve všech třech barevných složkách RGB obrázku, tak nejprve zjistíme derivaci obrázku v x-ovém a y-ovém směru pomocí Sobelova filtru:

$$\frac{\partial f}{\partial x} = \begin{pmatrix} \frac{\partial R}{\partial x} & \frac{\partial G}{\partial x} & \frac{\partial B}{\partial x} \end{pmatrix}^t \quad (5.7)$$

$$\frac{\partial f}{\partial y} = \begin{pmatrix} \frac{\partial R}{\partial y} & \frac{\partial G}{\partial y} & \frac{\partial B}{\partial y} \end{pmatrix}^t \quad (5.8)$$

Budiž  $J = (\partial f/\partial x, \partial f/\partial y)$  Jacobiho determinant, poté strukturní tenzor definujeme takto:

$$(g_{ij}) = J^T J = \begin{pmatrix} \frac{\partial f}{\partial x} \cdot \frac{\partial f}{\partial x} & \frac{\partial f}{\partial x} \cdot \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial x} \cdot \frac{\partial f}{\partial y} & \frac{\partial f}{\partial y} \cdot \frac{\partial f}{\partial y} \end{pmatrix} =: \begin{pmatrix} E & F \\ F & G \end{pmatrix} \quad (5.9)$$

Strukturní tenzor určuje pro daný bod kvadratický výraz určující velikost změny ve směru vektoru  $n = (n_x, n_y)$

$$S(n) = En_x^2 + 2Fn_xn_y + Gn_y^2 \quad (5.10)$$

Krajní hodnoty výrazu  $S(n)$  na jednotkové kružnici odpovídají vlastním číslům  $(g_{ij})$ . Jelikož je  $(g_{ij})$  symetrická matice, tak mají vlastní čísla reálné hodnoty a vlastní vektory jsou an sebe kolmé. Obě vlastní čísla matice mohou být nalezena vyřešením rovnice  $\det((g_{ij} - \lambda_i I) = 0$ . Což vede k hodnotám vlastních čísel:

$$\lambda_{1,2} = \frac{E + G \pm \sqrt{(E - G)^2 + 4F^2}}{2} \quad (5.11)$$

a odpovídající vlastní vektory mají hodnoty:

$$\nu_1 = \begin{pmatrix} F \\ \lambda_1 - E \end{pmatrix} \quad \nu_2 = \begin{pmatrix} \lambda_2 - G \\ F \end{pmatrix} \quad (5.12)$$

Směr maximální změny v obrázku je určen vektorem  $\nu_1$  a směr minimální změny vektorem  $\nu_2$ . Velikost změny je určena hodnotami  $\sqrt{\lambda_1}$  resp.  $\sqrt{\lambda_2}$ .

Také je důležité vzít v úvahu, že  $\nu_1$  a  $\nu_2$  reprezentují pouze orientaci a ne přímo směr protože matice  $J$  i  $-J$  odpovídají stejnému strukturnímu tenzoru  $(g_{ij})$ .

Pokud přiřadíme vlastní vektor  $\nu_2$ , odpovídající menšímu z vlastních čísel, každému bodu v obrázku, tak získáme vektorové pole reprezentující lokální orientaci hran. Bohužel toto vektorové pole má nespojitosti a není vyhlazené a proto jeho použití nedává dobré výsledky. Pro vyhlazení tohoto vektorového pole je na něj aplikována filtrace Gaussovým filtrem o velikosti  $5 \times 5$ , ale je možné použít i filtr o velikosti  $7 \times 7$  či  $9 \times 9$ .

Možným rozšířením této metody by bylo získávání orientace ve 3D objemových datech a pomocí této orientace provádět filtraci ve 3D. V článku [25] již bylo navrženo zobecnění metody Edge tangent flow 5.1 do pro získání 3D orientace, tak by jistě bylo zajímavé zobecnit i tuto metodu.

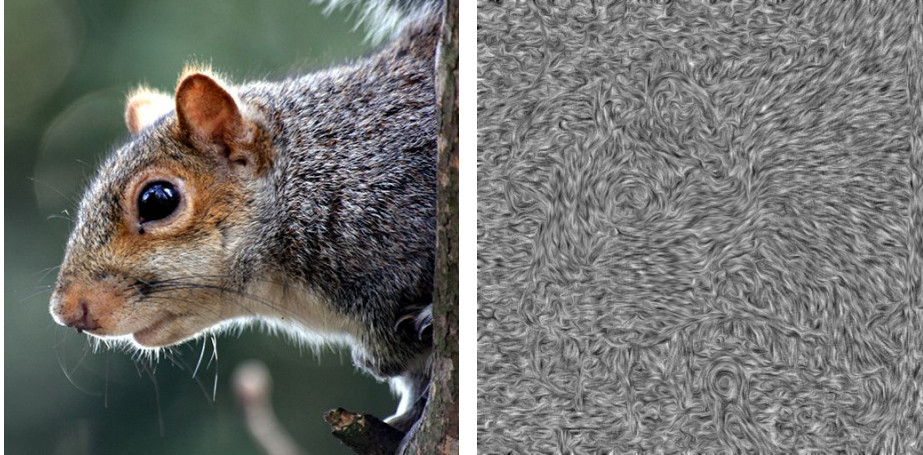
### 5.3 Zobrazení struktury

Pokud již máme nalezenou strukturu obrázku, tak je občas také užitečné ji moci zobrazit. Nejjednodušší možností jak zobrazit 2D vektorové pole je namapovat dvě ze tří barev v obrázku na hodnoty, které mají vektory a tento obrázek zobrazit, ovšem tento přístup nedává příliš dobré výsledky.

Lepší metoda byla navržena v [3].

Princip této metody je následující:

- Na vektorové pole uchováající nalezené orientace je namapována textura obsahující šum
- V každém bodě vektorového pole je následně sečteno  $L$  pixelů na obě strany ve směru určeném vektorovým polem. Můžeme také říci, že obrázek je v každém bodě filtrován maskou o šířce 1 a délce  $2L$ , jejíž tvar je určen vektorovým polem.
- Hodnota získaná sečtením pixelů pod maskou je normalizována délkou masky a poté je umístěna do výstupního obrázku.



(a) Originální obrázek

(b) Výsledek

Obrázek 5.1: Ukázka zobrazení struktury obrázku

## 5.4 Zlepšení kvality odhadnuté struktury

V oblastech obrazu s nízkým kontrastem není získaný gradient spolehlivý a proto i vypočtená orientace velmi nepřesná. Jedna z metod jak přesnost zvýšit je uvedena v [14].

Princip metody spočívá v tom, že nalezneme pixely s nízkým gradientem a jejich strukturální tenzory nahradíme průměrem z okolních pixelů. Je třeba zdůraznit, že tato operace je aplikována na samotné strukturální tenzory a ne na vektorové pole určující lokální orientaci, které je následně získáno.

Definujme tedy:

$$\partial\Omega = \{x \in \Omega \mid \sqrt{E(x)^2 + G(x)^2 + 2F(x)^2} > \tau_r\} \quad (5.13)$$

jako množinu bodů je velikost gradientu menší než volitelný práh  $\tau_r$

Pokud definujeme  $s_{i,j}^0$  jako strukturální tenzor v bodě  $(i, j)$  v nulté iteraci, pak jeden krok této metody spočívá v:

$$s_{i,j}^{k+1} = \begin{cases} s_{i,j}^k & \text{pokud } (i,j) \in \partial\Omega \\ \frac{s_{i+1,j}^k + s_{i-1,j}^k + s_{i,j+1}^k + s_{i,j-1}^k}{4} & \text{jinak} \end{cases}$$

## Kapitola 6

# Abstrakce videa

Jednou z hlavních etap při vytváření nerealistického videa je jeho tzv. abstrakce. Abstrakce videa slouží ke zjednodušení jeho obsahu, tak aby byly odstraněny nevýznamné detaily a zároveň významné prvky v daném videu musí zůstat zachovány, případně mohou být i zdůrazněny.

Opět zde můžeme narazit na problém časové koherence mezi jednotlivými snímky. Z toho důvodu nesmíme abstrahovat video příliš, ale jen tak aby nebyly změny mezi jednotlivými snímky příliš zřetelné.

### 6.1 Bilaterální filtr

Bilaterální filtr je pravděpodobně jeden z nejvíce používaných při provádění abstrakce snímku či videa. V původní verzi je poměrně dost časově náročný, protože použité konvoluční jádro se mění pro každý pixel. Také není možné ho aplikovat odděleně, tj. nejprve v x-ovém směru obrázku a následně v y-ovém směru.

Pro vytváření nerealistického zobrazení videa byl tento filtr použit v [24]. Nerealistický efekt je vytvářen opakovanou aplikací bilineárního filtru na vstupní obrázek. Tím dojde k výraznému zjednodušení obrázku, ale zároveň jsou zachovány hrany a výrazné prvky obrázku, což je přesně druh filtrace obrazu, který potřebujeme právě pro nerealistické zobrazení videa či obrazu.

Před provedením samotné aplikace bilaterálního filtru je vstupní obrázek transformován z barevného prostoru RGB do prostoru CIE-Lab, který je vizuálně mnohem více uniformní. Je to z toho důvodu, že na základě rozdílů mezi hodnotami sousedních pixelů je přizpůsobováno konvoluční jádro a proto je třeba, aby vzdálenosti mezi barvami byly co nejvíce smysluplné.

Výstup bilaterálního filtru je definován následovně:

$$H(\hat{x}, \sigma_d, \sigma_c) = \frac{\int w_d(x, \hat{x}) w_c(x, \hat{x}) f(x) dx}{\int w_d(x, \hat{x}) w_c(x, \hat{x})} \quad (6.1)$$

$$w_d(x, \hat{x}, \sigma_d) = e^{-\frac{1}{2} \left( \frac{\|\hat{x} - x\|}{\sigma_d} \right)^2} \quad (6.2)$$

$$w_c(x, \hat{x}, \sigma_c) = e^{-\frac{1}{2} \left( \frac{\|f(\hat{x}) - f(x)\|}{\sigma_c} \right)^2} \quad (6.3)$$

Je tedy integrováno okolí pixelu, s tím že okolní pixely mají přidělenou váhu v závislosti na vzdálenosti od polohy pixelu a na vzdálenosti v barevném prostoru.

V minulosti již bylo mnoho pokusů o vylepšení bilaterálního filtru, ať už se jednalo o snížení časové náročnosti jeho výpočtu, nebo zlepšení kvality výstupu.

Jednou z možností jak urychlit bilaterální filtr je použít pouze jednorozměrný filtr a jeho aplikace nejprve podél x-ové osy a následně podél y-ové osy. Takováto aplikace není ekvivalentní použití celého okolí pro filtraci, ale při jeho použití vzniká jen velmi málo vizuálních artefaktů.

Další možnost vylepšení bilaterálního filtru byla ukázána v [13]. Zde je také použit pouze jednorozměrný bilaterální filtr, ale jeho směr je orientován vždy dle lokální struktury obrazu, jejíž výpočet byl popsán v 5.2. Je možné použít také výpočet lokální struktury popsáný v kapitole 5.1, ale ten je zbytečně výpočetně náročný.

Vzorkování obrazu v požadovaném směru je prováděno s konstantním krokem, který má vždy jednotkovou délku buď podél x-ové osy nebo podél y-ové osy.

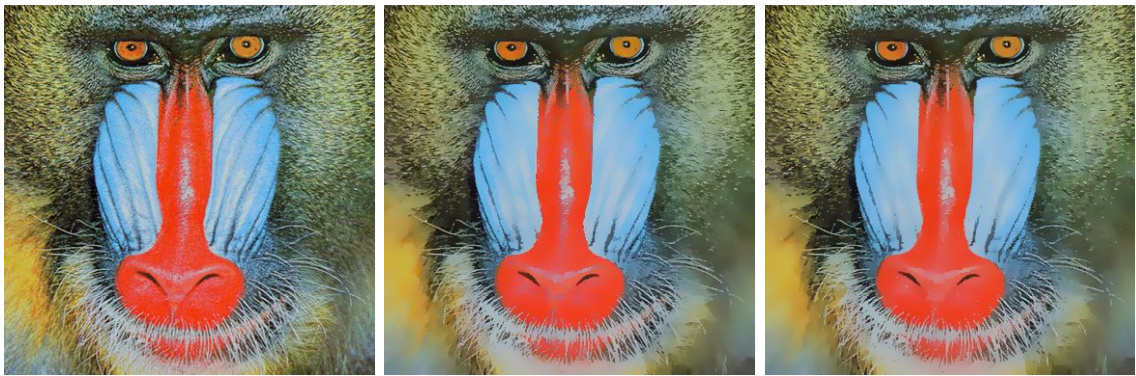
$$\delta(d) = \begin{cases} \left(1, \frac{d_y}{d_x}\right) & \text{pokud } |d_x| \geq |d_y| \\ \left(\frac{d_x}{d_y}, 1\right) & \text{pokud } |d_x| < |d_y| \end{cases} \quad (6.4)$$

Použitím jednotkové délky dosáhneme toho, že je dostačující interpolovat pouze mezi dvěma sousedními pixely a není nutné interpolovat mezi čtyřmi pixely.

Budiž  $c_0$  aktuální bod a  $t_0$  je aktuální směr podél kterého chceme obrázek vzorkovat. Tj. jedná se buď o tangentu k hraně nebo o vektor k ní kolmý. Poté obrázek vzorkujeme v místech určených rovnicí  $(c_i)^\pm = c_0 \pm i \cdot t_0$  a odezva bilaterálního filtru je poté dána výrazem:

$$\frac{1}{k} [f(c_0) + \sum_{i=1}^N G_{\sigma_d}(i \|t_0\|) [G_{\sigma_r}(\|f(c_i^+) - f_0\|) f(c_i^+) + G_{\sigma_r}(\|f(c_i^-) - f_0\|) f(c_i^-)]] \quad (6.5)$$

V angličtině bývá tento filtr nazýván jako oriented, já ho v průběhu této práce budu nazývat jako orientovaný bilaterální filtr.



(a) Originální obrázek

(b) Bilaterální filtr

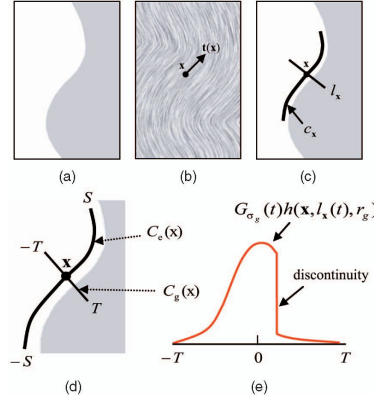
(c) Orientovaný bilaterální filtr

Obrázek 6.1: Srovnání výstupu různých variant bilaterálního filtru

Další možnost jak dostat pomocí bilaterálního filtru ještě o něco lepší výstup je uvedena v [9]. Základní princip tohoto filtru je ten, že při filtraci podél tangenty k hraně se orientace

kroku postupně mění a není konstantní tak jako u předchozí metody. Fungování tohoto filtru je naznaceno na obrázku 6.2.

Filtrace je nejprve provedena ve směru kolmém k lokální orientaci a následně v druhém kroku je výstup toho předchozího filtrován ve směru odpovídajícím lokální orientaci.



Obrázek 6.2: Průběh aplikace bilaterálního filtru , zdroj: [10]

### Bilaterální mřížka

Vzhledem k výpočetní náročnosti bilaterálního filtru existují různé metody pro zrychlení jeho výpočtu. Jednou z nich je metoda prezentovaná v [4], pro nerealistické zobrazení videa byla tato metoda použita v [27].

Základem této metody je použití datové struktury nazývané bilaterální mřížka (angl. bilateral grid). Tato datová struktura může být použita i pro různé operace zpracování obrazu, jak bylo popsáno v [4], zde bude popsáno pouze použití pro implementaci bilaterálního filtru.

Bilaterální mřížka je 3D pole, kde první dva rozměry  $(x, y)$  odpovídají pozicím pixelů v obrázcích a třetí rozměr  $z$  odpovídá intenzitě.

Bilaterální filtr pomocí bilaterální mřížky se skládá ze tří kroků. Nejprve je zkonstruována samotná mřížka  $\Gamma$  akumulací hodnoty každého pixelu do příslušného prvku mřížky následujícím způsobem:

Inicializace:

$$\Gamma(i, j, k) = (0, 0) \quad (6.6)$$

Naplnění:

$$\Gamma([x/s_s], [y/s_s], [I(x)/s_r])_+ = (I(x), 1) \quad (6.7)$$

kde  $s_s$  a  $s_r$  jsou vzorkovací poměry na prostorových osách a na ose značící intenzitu. Operátor  $[\cdot]$  značí nejbližší celé číslo.

Jako druhá etapa je na takto vytvořenou bilaterální mřížku aplikován Gaussův filtr v každém ze tří směrů.

Výsledná hodnota pixelu je získána pomocí trilineární interpolace z takto filtrované mřížky v bodě:

$$(x/s_s, y/s_s, I(x)/s_r) \quad (6.8)$$

V kapitole 4.3 je popsána možnost využití bilaterální mřížky pro kvantování hodnot v obrázku.

## 6.2 Kuwaharův filtr

První verze tohoto filtru byla uvedena v článku [12]. Principem tohoto filtru je rozdělit okolí každého pixelu do čtyř čtvercových regionů a výslednou hodnotou je průměrná hodnota subregionu s nejmenším rozptylem hodnot.

Uvažujme šedotónový obrázek  $I(x, y)$  a čtverec o délce  $2a$  se středem v bodě  $(x, y)$ , který je rozdělen do čtyř oblastí  $Q_1, Q_2, Q_3, Q_4$  následujícím způsobem:

$$\begin{cases} Q_1(x, y) = [x, x + a] \times [y, y + a] \\ Q_2(x, y) = [x - a, x] \times [y, y + a] \\ Q_3(x, y) = [x - a, x] \times [y - a, y] \\ Q_4(x, y) = [x, x + a] \times [y - a, y] \end{cases} \quad (6.9)$$

, kde  $x$  označuje skalární součin.

Budiž  $m_i(x, y)$  a  $s_i(x, y)$  průměr s standardní odchylka v každém ze čtyř regionů  $Q_i(x, y), i = 1..4$  definovaných dříve. Poté výstup Kuwaharova filtru  $\Phi$  v bodě  $(x, y)$  je definován jako průměr  $m_i(x, y)$  regionu s minimální standardní odchylkou hodnot  $s_i(x, y)$ .

Při práci s barevnými obrázky je každá barevná složka zpracována zvlášť.

Toto může být zapsáno následovně:

$$\Phi(x, y) = \sum m_i(x, y) f_i(x, y) \quad (6.10)$$

kde:

$$f_i(x, y) = \begin{cases} 1 & \text{pokud } s_i(x, y) = \min_k \{s_k(x, y)\} \\ 0 & \text{jinak} \end{cases} \quad (6.11)$$

Tato verze Kuwaharova filtru má svá omezení, jedním z nich je výstup na kterém je jasně zřetelný čtvercový tvar filtru, jak je vidět na obrázku 6.3(a).

Dalším nedostatkem je to, že tento operátor není dobře matematicky definován. Pokud dosáhne standardní odchylka hodnot ve dvou nebo více regionech minimálních hodnot, tak není jasně definované jaká hodnota má být výsledkem, jelikož tyto dva regiony mohou mít různé průměrné hodnoty.

Existuje velké množství rozšíření Kuwaharova filtru, které byly prezentovány od vzniku této původní verze. Jedno z těchto rozšíření, vhodné pro nefotorealitické zobrazení, je uvedeno v [19]. Tento článek zobecňuje Kuwaharův filtr, tím způsobem, že definuje obecně  $N$  oblastí v obrázku a váhované průměry hodnot pixelů  $m_i$  a standardní odchylky  $s_i$  pomocí váhovací funkce  $w_i$

$$m_i = I * w_i \quad (6.12)$$

$$s_i^2 = I^2 * w_i - m_i^2 \quad (6.13)$$

$$\int \int w_i(x, y) dx dy = 1 \quad (6.14)$$

Konkrétní definice filtru je následující, kruhové okolí každého pixelu je rozděleno do  $N$  sektorů  $S_i, i = 1 \dots N$  a v každém tomto sektoru je spočítán váhovaný průměr a standardní odchylka. Budiž  $g_\sigma(x, y)$  Gaussovský filtr:

$$g_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (6.15)$$

Sektory  $S_i$  jsou definovány následujícími funkcemi  $V_i(r, v)$  vyjádřenými v polárních souřadnicích

$$V_i = U_i * g_{\text{sigma}/4} \quad (6.16)$$

$$U_i = \begin{cases} N & \text{pokud } i - \frac{1}{2} < \frac{N}{2\pi}v < i + \frac{1}{2} \\ 0 & \text{jinak} \end{cases} \quad (6.17)$$

Váhovací funkce jsou definovány vynásobením Gaussovské masky  $g_\sigma$  a funkcí  $V_i$

$$w_i = g_\sigma V_i \quad (6.18)$$

$$\frac{1}{N} \sum_{i=1}^N V_i(x, y) = 1 \frac{1}{N} \sum_{i=1}^N w_i(x, y) = g_\sigma(x, y) \quad (6.19)$$

Funcke  $f_i(x, y)$  6.11 je nahrazena funkcí:

$$f_i^{(q)}(x, y) = \frac{s_i^{-q}(x, y)}{\sum_{i=1}^N s_i^{-q}(x, y)} \quad (6.20)$$

, kde  $q$  je volitelný parametr. Poté je výstup filtru  $\Phi_q(x, y)$  dán rovnicí:

$$\Phi_q(x, y) = \frac{\sum_i m_i s_i^{-q}}{\sum_i s_i^{-q}} \quad (6.21)$$

Výstup filtru  $\Phi_q(x, y)$  je tedy určen váhovaným průměrem středních hodnot  $m_i$  s váhami rovnými  $s_i^{-q}(x, y)m_i$ . Předpokládáme, že  $q$  je kladné a větší váha je tedy dána hodnotám  $m_i$  které odpovídají menším hodnotám  $s_i$ .

Pokud je  $q = 0$ , tak výstup funkce  $\Phi_q(x, y)$  se zjednodušuje na aritmetický průměr středních hodnot  $m_i (i = 1 \dots N)$  a navržený filtr je ekvivalentní Gaussovu filtru. Naopak pokud  $q \rightarrow \infty$  tak je navržený filtr redukován na původní verzi Kuwaharova filtru.

Rozšíření filtru pro barevné obrázky: Budiž  $I^{(1)}(x, y), I^{(2)}(x, y), I^{(3)}(x, y)$  složky v daném barevném prostoru. Z nich jsou spočítány střední hodnoty  $m_i^{(c)}(x, y)$  a rozptyly  $s_i^{(c)}(x, y)$  pro každou barevnou složku  $c$  zvlášť. Následně je výsledná hodnota spočítána následovně:

$$\Phi_q = \frac{\sum_i m_i s_i^{-q}}{\sum_i s_i^{-q}} \quad (6.22)$$

$$m_i = [m_i^{(1)}, m_i^{(2)}, m_i^{(3)}]^T \quad s_i = \sqrt{\sum_{c=1}^3 [s_i^{(c)}]^2} \quad (6.23)$$

Další vylepšení této metody bylo uvedeno v [15].

Pro tuto metodu je třeba vypočítat míru anizotropie, tj. jak moc dochází v okolí pixelu ke změnám. Pokud v okolí pixelu nedochází ke změnám, tak toto okolí nazýváme izotropickým, pokud je v okolí pixelu změna, tak se jedná o anizotropický region. Anizotropie  $A$  má tedy hodnoty od 0 do 1, kde 0 odpovídá izotropickému a 1 naprosto anizotropickému regionu. Tzn. pokud je anizotropie rovna hodnotě 1, tak v sousedství pixelu dochází k významné změně v hodnotách pixelů. Anizotropie je definována jako:

$$A = \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2} \quad (6.24)$$

Budiž  $f : \mathbb{R}^2 \rightarrow \mathbb{R}^3$  vstupní obrázek a  $x_0 \in \mathbb{R}^2$  bod obrázku,  $\varphi$  je lokální orientace a  $A$  je anizotropie v bodě  $x_0$ . Tvar filtru v konkrétním bodě budeme určovat pomocí matice

$$S = \begin{pmatrix} \frac{\alpha}{\alpha + A} & 0 \\ 0 & \frac{\alpha + A}{\alpha} \end{pmatrix} \quad (6.25)$$

, kde  $\alpha > 0$  je uživatelsky definovaný parametr. Pro  $\alpha \rightarrow \infty$  se z matice  $A$  stává jednotková matice. V této práci je použito  $\alpha = 1$ .

Budiž  $R_\varphi = 1$  matice definující rotaci o úhel  $\varphi$  a budiž  $h = \lceil 2\sigma_r \rceil$ , poté množina bodů:

$$\Phi = \{x \in \mathbb{R}^2 : \|S R_{-\varphi} x\| \leq h\} \quad (6.26)$$

Definuje elipsu jejíž hlavní osa zarovnána na lokální orientaci v obraze. V oblastech kde je vysoká anizotropie je elipsa hodně zploštěná neboli excentrická a naopak v isotropických oblastech se blíží kruhu.



(a) Původní verze Kuwaharova filtru (b) Zobecněná verze Kuwaharova filtru (c) Anizotropická verze Kuwaharova filtru

Obrázek 6.3: Ukázky výstupu různých variant Kuwaharova filtru

### 6.3 Shock-based filtrace

Tento druh filtrace obrazu nemá přesný překlad do češtiny a proto se budu držet anglického názvu shock filtr, nebo případně shock-based filtrace.

Tato skupina filtrů je založena na myšlence lokálně aplikovat buď dilataci nebo erozi dle toho, do které oblasti patří konkrétní pixel. Pixel může patřit buď od zóny maxima

nebo minima. Pokud náleží do zóny maxima, tak je prováděna eroze, tj. je mu přiřazena maximální hodnota v určitém jeho okolí. V opačném případě patří do zóny minima a je prováděna dilatace, tj. je přiřazena minimální hodnota z určitého okolí.

Takto je vytvářen ostrý přechod mezi dvěma oblastmi a je vyprodukována po částech konstantní segmentace obrázku. Tento filtr také využívá zjištěné lokální struktury obrázku, podobně jako ostatní metody. Jedna z možných variant shock filtru byla navržena v článku [22], ale tato není příliš vhodná pro zpracování videa, jelikož použitá filtrace je dosti agresivní a proto se po sobě následující obrázky ve videu dost liší.

Pro zpracování videa byla navržena vylepšená metoda, uvedená v [14], která kombinuje shock filter s orientovanou filtrací Gaussovským filtrem.

Tato metoda kombinuje opakovanou aplikaci adaptivního vyhlazování následovanou shock based filtrací. Adaptivní vyhlazování i shock based filtrace využívají, podobně jako předchozí metody, zjištěné lokální orientace v obraze.

Popis adaptivního vyhlazování:

Nejprve je třeba zjistit jak velká míra vyhlazování by měla být aplikována na daný pixel. Toto je zjištěné pomocí anizotropie jejíž výpočet již byl popsán v 6.2.

$$A = \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2} \quad (6.27)$$

Anizotropie má hodnoty v rozsahu mezi 0 a 1, kde 1 představuje maximální anizotropii, a v tomto místě chceme aplikovat maximální míru vyhlazení. Standardní odchylka Gaussova filtru je následně definována následujícím vztahem:

$$\tilde{\sigma}_s = \frac{1}{4}\sigma_s(1 + A)^2 \quad (6.28)$$

Parametr  $\sigma_s$  je definovaný uživatelem a umožňuje měnit míru vyhlazení obrázku globálně. Obvykle je použito  $\sigma_s = 6$ . Upravená standardní odchylka  $\tilde{\sigma}_s$  poté leží v rozsahu  $\sigma_s/4 \leq \tilde{\sigma}_s \leq \sigma_s$ .

Pro vzorkování obrázku je zde využito vztahu:

$$x_{k+1}^\pm = x_k^\pm + t_k^\pm(x_k^\pm + \frac{1}{2}t_k^\pm(x_k)) \quad (6.29)$$

kde  $t_k^\pm(x_k)$  je lokální orientace v bodě  $x_k$ .

Princip shock-based filtrace

Shock filter je aproximován maximem nebo minimem v závislosti na odezvě vybraného hranového detektoru.

Pro tuto metodu filtrace byla navržena jednorozměrná aproximace filtru LoG (Laplacián Gaussova rozostření)

$$\sigma_g^2 G''_{sigma_g}(t) = \sigma_g^2 \frac{d^2 G_{\sigma_g}(t)}{dt^2} = \frac{x^2 - \sigma_g^2}{\sqrt{2\pi}\sigma_g^3} e^{-\frac{t^2}{2\sigma_g^2}} \quad (6.30)$$

Je zde zaveden ještě další parametr  $\sigma_i$ , který provádí další volitelné vyhlazení:  $\nu = G * \sigma_i * L$  kde  $L$  je vstupní obrázek konvertovaný do ostnů šedi.

Filtrace tímto LoG filtrem je provedena následujícím způsobem:

$$z(x_0) = \sigma_g^2 \int G''_{\sigma_g}(t) \nu(x_0 + t \cdot \eta(x_0)) dt \quad (6.31)$$

kde  $\eta(x_0)$  je vektor kolmý k výpočítané lokální orientaci.

Pokud definujeme  $x_0$  jako vstupní bod, tak odezva shock filtru je definována následujícím způsobem

$$\begin{cases} \min_{x \in \Lambda_r(x_0)} f(x) & \text{pokud } z(x_0) > +\tau_s \\ \max_{x \in \Lambda_r(x_0)} f(x) & \text{pokud } z(x_0) < -\tau_s \\ f(x_0) & \text{jinak} \end{cases} \quad (6.32)$$

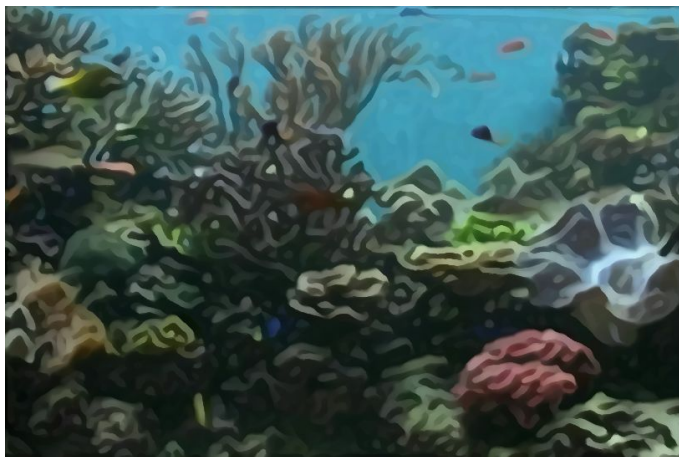
kde  $f$  označuje vstupní obrázek. Minimum a maximum je určeno na základě příslušného šedotónového obrázku. Okolí filtru o poloměru  $r$  je definováno následovně:

$$\Lambda_r(x_0) = \{x \in \Omega \mid \exists \lambda \in [-r, r] : x = x_0 + \lambda \eta(x_0)\} \quad (6.33)$$

Na obrázku 6.4 je ukázka výstupu tohoto filtru, jak je vidět, tak tento filtr je poměrně dost agresivní, proto je vhodné pro zpracování videa používat malý počet iterací (u většiny videí 2–6 iterací)



(a) Originální obrázek



(b) Výstup po pěti iteracích

Obrázek 6.4: Zpracování obrázku pomocí shock filtru

## 6.4 Morfologické operace

Morfologické operace jsou pravděpodobně častěji definované pro binární obrázky, ale v článku [2] bylo uvedeno jejich zobecnění i pro šedotónové obrázky. V tomto článku bylo také popsáno jejich použití pro zpracování videa.

Eroze je definována jako:

$$\delta_B(I)(x) = \max_{b \in B} \{I(x + b)\} \quad (6.34)$$

Dilatace je definována jako:

$$\varepsilon_B(I)(x) = \min_{b \in B} \{I(x + b)\} \quad (6.35)$$

Operace otevření je poté definována jako eroze obrázku následovaná dilatací a obráceně morfologické uzavření je definováno jak dilatace následovaná erozí.

Pro abstrakci obrázku je poté použita operace uzavření následovaná otevřením.

Při aplikaci tohoto morfologického operátoru na každý obrázek zvlášť bez ohledu na další obrázky vede k výraznému blikání ve videu.

Jednou z možností jak řešit tento problém je použít 3D morfologický operátor. v [2] byl použit morfologický operátor kuželovitého tvaru a o poměrně velkých rozměrech (7x7x9). Tvar operátoru se přizpůsobuje optickému toku, pokud je v časovém rozměru příliš velký rozdíl mezi hodnotami, tak pro daný pixel ukončíme zpracování.

Při zpracování této práce byl otestován morfologický operátor aplikovaný na jednotlivé snímky, bez uvažování časové koherence, ale tento způsob nedával příliš dobré výsledky.



(a) Originální obrázek

(b) Výsledek aplikace morfologického operátoru

Obrázek 6.5: Aplikace morfologického operátoru na obrázek

## Kapitola 7

# Detekce hran

### 7.1 LoG

Při použití této metody detekce hran je na obrázek nejprve aplikován Gaussův filtr a následně Laplacův operátor, což je aproximace druhé derivace obrázku. Tyto dva filtry je možné zkombinovat do jednoho a následně aplikovat na obrázek v jednom kroku. Tento filtr detekce hran byl prezentován v článku [17]. Aplikace tohoto filtru je ovšem dosti výpočetně náročná jelikož není možné použít výpočet odděleně pro x-ový směr a y-ový směr podobně jako třeba u Gaussova filtru. Další důvod výpočetní náročnosti je, že používané konvoluční operátory jsou dost velké ve srovnání s jinými filtry. V [20] je uváděna velikost filtru  $N = 3 \sim 43$  pro  $\sigma = 3 \sim 43$ .

V článku [20] je navržena verze tohoto filtru, která se snaží o výpočet pro video v reálném čase pomocí předpočítaných odezev daného filtru. Tato verze detekce hran byla použita pro nerealistické zobrazení videa v článku [8].

### 7.2 Rozdíl Gaussovských rozostření

Tato metoda detekce hran se snaží aproximovat detektor hran popsany v podkapitole 7.1 a definovaný v článku [17].

Tento způsob detekce hran ve videu byl použit již mnohokrát, asi nejznámější v oblasti nerealistického zobrazení videa je článek [24] odkud bylo čerpáno i pro tuto práci.

Tato metoda detekce používá rozdíl dvou Gaussových, kde každý z nich má jiný parametr  $\sigma_r$  a  $\sigma_e$ , obvykle je používáno  $\sigma_r = \sqrt{1,6}\sigma_e$  a  $\sigma_e = 1$  Proto je také tento filtr v angličtině nazýván difference-of-Gaussians, nebo zkráceně DoG.

Pro získání výsledných hran je výstup tohoto filtru prahován. Při zpracování videa je vhodné, aby byly výsledné hrany vyhlazené, tj. výstup filtru nebude pouze rozhodnutí mezi bílým a černým pixelem, ale budou zde i odstíny šedi. Detekce hran bez odstínů šedi je používána spíše pro zpracování obrázků, u videa je tímto způsobováno nepříjemné blikání, pokud v určitém snímku hrana je a v dalším není. Při použití odstínů šedi je nevýrazná hrana pouze šedá a pokud v dalším snímku zmizí, tak to není příliš vizuálně výrazné.

Parametr  $\tau$  v rovnici 7.1 kontroluje jak velký by měl být rozdíl mezi výstupy daných Gaussových filtrů, aby bylo možné považovat daný pixel za hranu. Jinak řečeno tento parametr určuje citlivost detekce hran. Pro menší hodnoty  $\tau$  je detekováno méně šumu, ale zároveň jsou méně zřetelné i skutečné hrany. Obvykle je používáno  $\tau = 0,98$ . Pro  $\tau \rightarrow 1$  přestává být filtr stabilní.

Dále je v rovnici parametr  $\varphi_e$ , který ovlivňuje ostrost detekované hrany, obvykle se pohybuje v intervalu  $\varphi_e \in [0, 75; 5, 0]$ .

Pro následující rovnice definujeme  $S_{\sigma_e} \equiv S(\hat{x}, \sigma_e)$  a  $S_{\sigma_r} \equiv S(\hat{x}, \sigma_e \cdot \sqrt{(1, 6)})$

$$D(\hat{x}, \sigma_e, \tau, \varphi_e) = \begin{cases} 1 & \text{pokud } (S_{\sigma_e} - \tau S_{\sigma_r}) > 0 \\ 1 + \tanh(\varphi_e \cdot (S_{\sigma_e} - \tau S_{\sigma_r})) & \text{jinak} \end{cases} \quad (7.1)$$

$$S(\hat{x}, \sigma_e) = \frac{1}{2\pi\sigma_e} \int f(x) e^{-\frac{1}{2} \left( \frac{\|\hat{x} - x\|}{\sigma_e} \right)^2} dx \quad (7.2)$$

, kde  $S(\hat{x}, \sigma_e)$  je Gaussův filtr s velikostí standardní odchylky rovné  $\sigma_e$ . Funkce  $\tanh$  použitá v rovnici 7.1 umožňuje vznik vyhlazených hran, tzn. hrany nejsou definovány binárně hodnotami 0 a 1, ale obsahují hodnoty mezi 0 a 1. Rovnicí 7.1 je tedy definován výstup daného detektoru hran.

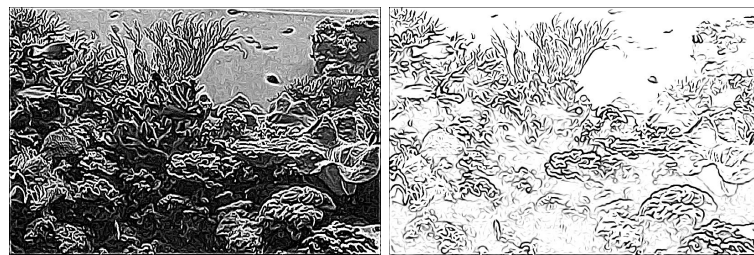
### 7.3 Orientovaný rozdíl Gaussovských rozostření

Tento detektor hran funguje na podobném principu jako bilaterální filtru ukázaný na obrázku 6.2.

Pro detekci hran je opět použit rozdíl Gaussovských rozostření s různou velikostí standardní odchylky. Odchylky jsou určeny jako  $\sigma_1 = \sqrt{\sigma_2}$ .

Filtrace probíhá podobně jako u orientovaného bilaterálního filtru ve dvou etapách. V první etapě je aplikován jednodimenzionální detektor hran ve směru kolmém na lokální orientaci v obraze. Výstup této etapy je následně zpracován v další etapě kdy je na obrázek aplikován Gaussův filtr ve směru sledujícím lokální orientaci obrazu.

Způsob provádění filtrace, tak aby byla sledována lokální orientace je podrobně popsán v kapitole 8.1. Zjednodušeně řečeno, tvar Gaussova filtru kopíruje právě lokální strukturu v obraze. Ve místě aktuálně filtrovaného pixelu je zjištěna lokální orientace a filtr sleduje jedním směrem tuto orientaci a ve druhém směru orientaci opačnou. V každém novém pixelu je znova zjištěna lokální orientace a směr filtrace se dle toho příslušným způsobem změní.



(a) Po filtraci detektorem hran v kolmém směru (b) Po vyhlazení Gaussovým filtrem a prahování

Obrázek 7.1: Detekce hran

Po vyhlazení Gaussovým filtrem je provedeno prahování, které extrahuje výsledné hrany. Prahoání může být dvojího druhu, buď je obrázek prahován binárně, tzn. u každého pixelu je nastavena buď hodnota 0 značící čenou barvu nebo 1, označující bílou barvu. Druhou možností je prahovat hodnoty, tak aby výsledkem byl šedotónový obrázek, tj. obrázek obsahující

i hodnoty mezi 0 a 1. První varianta dává o něco lepší výsledky v případě samostatných obrázků, ovšem druhá varianta je vhodnější pro zobrazení videa.

Filtraci hran je možné provádět i iterativně, to znamená že vstupní obrázek je vynásoben obrázkem s detekovanými hranami a tento je opět filtrován stejným způsobem. To vede k zesílení odezvy detektoru hran v místech, kde byla předtím odezva slabá. Tento způsob iterativního filtrování opět není příliš vhodný pro zpracování videa, jelikož tímto způsobem vznikají nežádoucí výrazné hrany, způsobující blikání mezi jednotlivými snímky videa.

# Kapitola 8

## Popis řešení

### 8.1 Vzorkování vstupního obrázku

Způsob jakým je prováděno vzorkování vstupního obrázku může mít velký vliv na kvalitu výsledku. Tato problematika již byla, částečně zmíněna u jednotlivých metod filtrace, ale zde bych ji chtěla shrnout, jelikož na správné implementaci metod vzorkování částečně závisí také úspěšnost samotné filtrace obrazu.

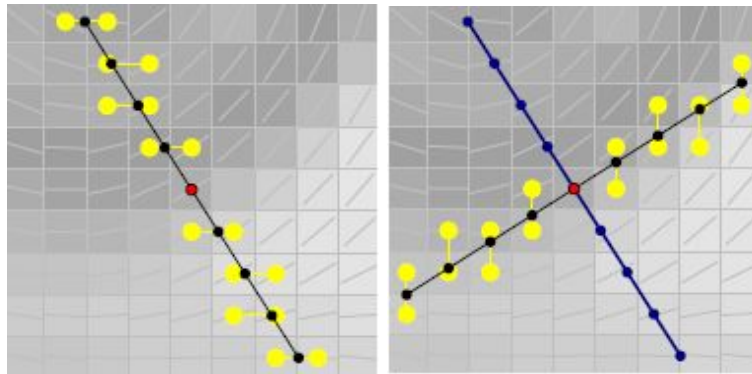
#### Uv-orientované vzorkování

Princip této metody je naznačen na obrázku 8.1. Při použití tohoto druhu vzorkování, nejprve provedeme filtraci ve směru kolmém ke zjištěné orientaci (žlutě označený směr v levém obrázku, či modře označený směr v pravém obrázku 8.1), výstup této filtrace je pouze mezivýsledek je uložen do pomocné paměti. Následně je provedena filtrace ve směru odpovídajícím zjištěné lokální orientaci v obraze, tedy ve směru odpovídajícím tečně k hraně v obrázku. Vstupem tohoto druhého kroku filtrace obrázku je mezivýsledek z předchozího kroku uložený v pomocné paměti.

Na obrázku 8.1 je naznačen ještě jeden zásadní rozdíl mezi dvěma způsoby vzorkování obrazu. Zatímco žlutě označený způsob vzorkování se posouvá v každém kroku o konstantní krok, tak aby x-ová nebo y-ová komponenta toho kroku byla rovna 1. Oproti tomu v případě modře označeného vzorkování obrazu je krok vždy o délce jedna, bez ohledu na konkrétní orientaci.

Nedá se přesně určit, který z těchto způsobů je lepší, modře označený způsob vzorkování poskytuje obvykle přesnější výsledky, ale je méně efektivní. Opačná fakta platí o žlutě označeném vzorkování obrazu. Žlutě označený způsob vzorkování je eektivnější z toho důvodu, že velikost kroku vždy přizpůsobuje konkrétní orientaci v daném bodě a díky tomu je dostačující použít pouze lineární interpolaci dvou sousedních pixelů, kdežto u modře označeného vzorkování je nutné bilineárně interpolovat ze čtyř sousedních hodnot.

Modře označené vzorkování obrazu by mělo být obecně přednější a dávat lepší výsledky, ovšem při aplikaci některých filtrů jsou výsledky těchto dvou velmi málo vizálně odlišné. Oproti tomu například způsob filtrace popsany v 6.3 je velmi citlivý na kvalitu použitého vzorkování.

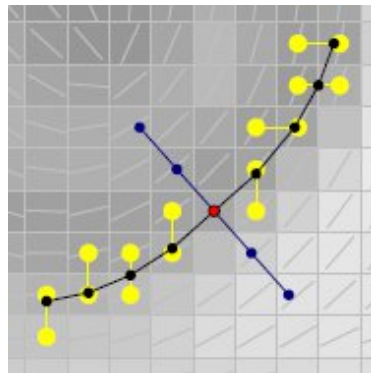


Obrázek 8.1: Znázornění průběhu vzorkování obrázku

### Filtrace sledující tok orientace v obraze

Dalším vylepšením vzorkování při provádění filtrace je sledování toku orientace v daném vstupním obraze, toto je naznačeno na obrázku 8.2.

Zde je opět uvedena efektivnější varianta provedení filtrace, která v každém kroku filtrace přizpůsobuje svůj krok, tak aby souřadnice alespoň v jednom ze směrů zůstaly celočíselné. Díky tomu je možné opět použít pouze lineární interpolaci místo bilineární interpolace, která by byla nutná pokud ani jedna souřadnice není celočíselná.



Obrázek 8.2: Filtrace ve směru vypočítané orientace

### Vzorkování pomocí Eulerovy integrace

In order to perform line integral convolution, the stream line through the point to smooth must be found. A straightforward approach is to use the Euler integration method (Figure 5). Special attention needs to be paid to the sign of the minor eigenvector. The structure tensor defines only orientation, but no particular direction. This is due to the quadratic nature of the structure tensor. Therefore, the sign of the minor eigenvector is chosen, depending upon the direction of the previous step:

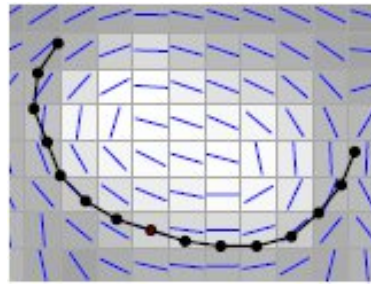
Při použití této metody jsou procházeny pixely na jednu stranu ve směru lokální orientace a na druhou stranu ve směru opačném.

Je tedy nezbytné při použití této metody i při integraci pomocí metody Runge-Kutta věnovat pozornost směru nalezené lokální orientace, je to z toho důvodu, že nalezené vektory definují pouze orientaci, ale již ne konkrétní směr, není tudíž zřejmé zda při filtraci postupovat ve směru daném tímto vektorem nebo ve směru opačném. Proto je znaménko lokální orientace vybráno na základě orientace použité v předchozím kroce:

$$t_k^\pm(\mathbf{x}) = \begin{cases} \pm \eta x & \text{pokud } k = 0 \\ \text{sign}(t_{k-1}^\pm \cdot \eta(x)) \cdot \eta(x) & \text{jinak} \end{cases} \quad (8.1)$$

kde  $\eta x$  je lokální orientace v obraze, tak jak byla určena pomocí strukturního tenzoru. V prvním kroce je nastaveno  $x_0^\pm = x_0$ , kde  $x_0$  je aktuálně zpracovávaný bod. Jeden krok Eulerovy metody je definován následovně:

$$x_{k+1}^\pm = x_k^\pm + t_k^\pm(x_k) \quad (8.2)$$



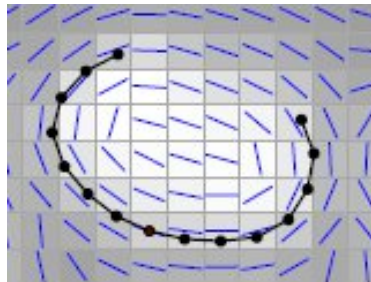
Obrázek 8.3: Vzorkování pomocí Eulerovy integrace

### Vzorkování pomocí metody integrace Runge-Kutta

Při filtrování je použita metoda Runge-Kutta druhého řádu, bylo by možné použít i metody vyšších řádů, ale to by bylo zbytečně výpočetně náročné.

Jeden krok této metody je definován následovně:

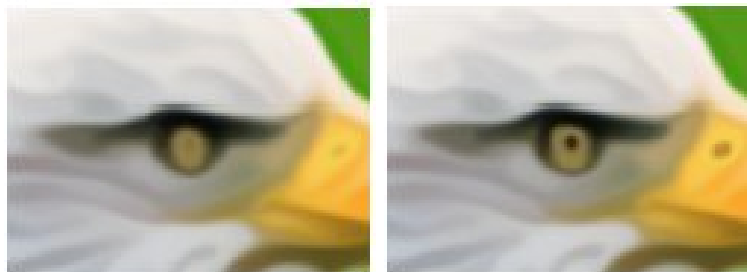
$$x_{k+1}^\pm = x_k^\pm + t_k^\pm(x_k^\pm + \frac{1}{2}t_k^\pm(x_k)) \quad (8.3)$$



Obrázek 8.4: Vzorkování pomocí Runge-Kutta

Jak je vidět na obrázcích 8.3 a 8.4, tak metoda Runge-Kutta mnohem lépe sleduje lokální orientaci v obrázku.

Mohlo by se zdát, že způsob vzorkování nemá příliš velký vliv na výsledný obrázek, ale je to právě naopak jak je vidět na obrázku 8.5. Při zpracování obrázku vlevo byla použita Eulerova integrace pro vzorkování a pro zpracování obrázku byla použita metoda Runge-Kutta a jak je vidět, tak rozdíl mezi výsledky je velmi výrazný.



Obrázek 8.5: Srovnání zorkování pomocí Eulerovy metody a metody Runge-Kutta

## 8.2 Průběh zpracování videa

Základní průběh zpracování videa byl již vysvětlen v kapitole 4.1.

V této práci byl průběh rozšířen o některé další prvky, a některé prvky byly upraveny

### Výpočet lokální struktury v obrázku

Některé metody filtrace používají získanou lokální strukturu obrázku, aby mohly lépe provádět samotnou filtraci obrazu, proto v průběhu zpracování přibyla tato volitelná část. Volitelná je proto, že získaná lokální struktura není používána všemi metodami.

### Abstrace obrázku

Tato etapa je uvedena již v původním průběhu zpracování videa, ale ja jsem jí ve své práci zobecnila, tak že samotná abstrakce může být prováděna i jinými filtry ne jen bilaterálním.

Je možné použít zobecněno verzi bilaterálního filtru, respektující lokální strukturu obrazu, dále je možné použít anisotropický Kuwaharův filtr a shock filtr, popsané v předchozích kapitolách. Další možností bylo použití morfologických operátorů, ovšem to při zpracování snímků samostatně nedávalo dobré výsledky.

Při zobecnění abstrakce obrázku je nutné dbát na to, aby byl vstupní obrázek ve správném formátu. Tzn. pro aplikaci bilaterálního filtru je nutné mít obrázek ve formátu *Lab* a pro použití ostatních metod je nutné mít vstupní obrázek ve formátu *RGB*.

### Detekce hran

Provedení detekce hran má v navržené aplikaci také několik možností provedení. Je možné zvolit mezi detekcí hran pomocí rozdílu Gaussovských rozostření aniž by filtr bral v úvahu lokální orientaci, nebo je možné použít orientovaný detektor hran.

Tato etapa byla také zobecněna v tom smyslu, že je možné provést detekci hran až po provedení kvantizace barev. Po kvantizaci barev se přechody mezi těmito barvami stanou

ostřejší a detektor má často odezvu právě v těchto místech. Bohužel toto zobecnění nepřineslo příliš dobré výsledky, jak bude ukázáno v kapitole 9.2, protože detekované hrany jsou často příliš ostré a tím způsobují blikání a ani pro samostatné obrázky to nepřineslo kvalitní výstupy.

V této etapě je opět nutné hlídat formát vstupního obrázku, v závislosti na použitých předchozích metodách. Pokud je obrázek v *RGB* tak je třeba ho převést na šedotónový.

### 8.3 Vývojové prostředí

Pro vývoj aplikace bylo použito programovacího jazyka C++ a program byl vyvíjen pod MS Windows za použití Visual Studia 10. Přesto by měla být aplikace snadno přenositelná i na jiné platformy.

#### OpenCV

Implementovaná aplikace využívá pro zpracování obrazu knihovnu OpenCV (Open Computer Vision Library). Jedná se o volně dostupnou knihovnu původně vyvíjenou společností Intel. Tato knihovna poskytuje širokou paletu funkcí zpracování obrazu, počítačového vidění a také umožňuje práci s různými vstupními a výstupními formáty obrazu.

V této aplikaci byla použita aktuální verze knihovny, tj. verze 2.2, ovšem aplikace je kompatibilní s všemi verzemi OpenCV 2.0 a vyšší. Verze knihovny nižší než 2.0 nebudou s aplikací kompatibilní jelikož ve verzi 2.0 bylo uvedeno objektově orientované rozhraní, které aplikace využívá a které a v nižších verzích neexistovalo. Ve starších verzích existovaly pouze funkce v jazyce C.

Bohužel v aplikaci nešlo možné plně využít optimalizovaných funkcí pro filtraci obrazu, které jsou již v této knihovně implementované. Je to z toho důvodu, že při provádění filtrace jsou často využívány informace o lokální struktuře obrazu a dle toho je filtrace v aktuálním bodě přizpůsobována a žádná funkce pro provedení takového typu filtrace v OpenCV aktuálně neexistuje.

I přesto byla knihovna velmi užitečná pro načítání a ukládání obrázků a pro další práci s nimi.

#### Načítání a ukládání videa

Jako nejlepší volba pro mou aplikaci se jevila knihovna FFMpeg a to především z toho důvodu, že je s její pomocí možné načítat a ukládat video víceméně v jakémkoli existujícím formátu. Knihovna FFmpeg poskytuje velké množství funkcí pro práci s videem i audiem. Pomocí této knihovny je možné video nahrávat, konvertovat mezi různými formáty, načítat ze souboru jednotlivé snímky a opačně snímky do souboru ukládat.

Součástí této knihovny jsou již existující programy, které umožňují zpracování videa pomocí příkazové řádky. Pro samotné zpracování videa je to program ffmpeg, dále jsou zde programy ffplay a ffprobe. Uživatel knihovny si může zvolit zda použije již existující spustitelný program, nebo zda integruje knihovnu přímo do své aplikace.

Bohužel integrace knihovny do vlastní aplikace není dle mého názoru příliš uživatelsky přívětivá, z důvodu absence jednoduchého návodu jak knihovnu použít. Takovéto tutoriály ke knihovně FFmpeg je možné na internetu nalézt, ovšem část z nich již je zastaralá jelikož aktuální API knihovny není kompatibilní s minulými verzemi API. Pravděpodobně nejvíce aktuální verze tutorálu je [1].

Po určitých problémech bylo rozhodnuto nepodporovat načítání přímo z videa, ale místo toho jsou načítány sekvence snímků uložených jako obrázky pomocí OpenCV ze zadaného adresáře.

Tento přístup je z programátorského hlediska o dosti jednodušší, pouze bylo nutné v aplikaci použít knihovnu Boost, pro procházení obsahu adresáře, jelikož standardní knihovny C++ toto neumožňují.

## OpenGL

Při tvorbě výsledného programu bylo použito také knihovny OpenGL. Knihovna OpenGL nebyla použita pro samotné vykreslování obrázků, což je její obvyklé použití, ale pro akceleraci některých operací zpracování obrazu.

V aplikaci jsou použity shadery vytvořené pomocí jazyka GLSL, který je součástí standardu od verze 2.0. Současná verze knihovny OpenGL je 4.1. z června 2010, proto je možné předpokládat, že vytvořené shadery budou schopné fungovat na většině současných grafických karet. V případě, že by přesto shadery nebyly funkční, tak je GLSL implementace nahrazena vlastní implementací v OpenCV.

Jazyk GLSL umožňuje zasahovat do fixní pipeline vykreslovacího řetězce OpenGL. Tato verze OpenGL také odstraňuje omezení rozměrů textur na mocninu dvou. Od verze 3.0 je zavedena plná podpora vykreslování do objektu framebufferu, čehož je v této aplikaci také využíváno.

Při vytváření aplikací s využitím OpenGL se často využívají také některé rozšiřující knihovny jako je GLU <sup>1</sup> nebo GLUT <sup>2</sup>

Další často používanou knihovnou je GLEW <sup>3</sup> Knihovna GLEW je také používaná ve vytvořené aplikaci.

Pro zde popisovanou aplikaci bylo třeba použít především knihovnu GLEW, která umožňuje použití rozšíření pro práci s GLSL shadery. Všechny použité shadery jsou fragment shadery, jelikož pro zpracování obrazu nemají vertex shadery žádný smysl.

## wxWidgets

Pro tvorbu grafického uživatelského rozhraní mé aplikace jsem použila knihovnu wxWidgets. Výhodou této knihovny je především její multiplatformnost s všeobecné povědomí o této knihovně.

Knihovna poskytuje obrovské množství tříd pro tvorbu GUI, ale bohužel trochu ztrácí krok s některými konkurenty jako je například Qt. To ovšem v případě této aplikace není výrazný problém, jelikož pro aplikaci jsou dostačující naprosto základní prvky GUI.

## 8.4 Způsob práce s OpenGL

Aplikace pro zpracování vstupních obrázků používá knihovny OpenGL a shaderů v jazyce GLSL. Aby bylo možné zpracovávat obrázky pomocí shaderů, aniž by bylo nutné vykreslovat výsledek přímo na obrazovku, takzvané Framebuffer objects, tedy objekty, které

<sup>1</sup>GLU (OpenGL Utility Library) - rozšiřující knihovna pro práci s kvadriky, mipmapami, křivkami, teselací polygonů...

<sup>2</sup>GLUT (OpenGL Utility Toolkit) - knihovna, která obsahuje nástroje pro správu okna a zachytávání udalostí a vykreslování základních objektů jako je kostka nebo koule (nebo čajová konvice)

<sup>3</sup>GLEW (OpenGL Extension Wrangler Library) - knihovna pro práci s rozšířeními OpenGL.

umožňují vykreslovat do paměti počítače a není nutné díky tomu vykreslovat na obrazovku, tak jako u klasických OpenGL aplikací.

Při startu aplikace je tedy vygenerováno potřebné množství textur, které jsou následně svázány s příslušným Framebuffer objektem. Pokud by nás zajímal jen finální výstup provedené abstrakce obrázku, v aplikaci je generováno více textur než je nezbytně nutné, jelikož některé z nich jsou používány pro uložení mezivýsledků, které si uživatel může chtít zobrazit a které by neměly být přepsány v dalších krocích. Uživateli je například umožněno zobrazit samostatně detekované hrany bez zbytku obrázku, proto se vygeneruje speciální textura kam jsou hrany vykresleny a v následných krocích je osud čteno. Případně jsou načtena data v této textuře a zobrazena uživateli. Pokud je prováděna operace, jejíž zobrazení není pro uživatele zajímavé, tak je výsledek vykreslen pouze do dočasné textury, která může být v některém dalším kroku přepsána. Jedná se například o převod z barevného prostoru *RGB* do *Lab*.

## 8.5 Popis výsledné aplikace

Jak již bylo řečeno výsledná aplikace je aplikace s GUI vytvořená pomocí knihovny wxWidgets. Aplikace má poměrně jednoduché uživatelské rozhraní, kde na levé straně je zobrazen načtený snímek a na pravé straně jsou různé prvky GUI umožňující přizpůsobovat nastavení dle přání uživatele.

Sekvenci vstupních snímků je možné načíst pomocí menu, případně lze také načíst pouze jeden snímek, například pro otestování parametrů na daném snímku. Při spouštění aplikace je důležité, aby v aktuálním adresáři byla složka obsahující soubory se shadery, protože aplikace tyto shadery načítá při svém startu.

# Kapitola 9

## Výsledky

### 9.1 Časová náročnost algoritmů

Časová náročnost implementovaných algoritmů je v tomto případě poměrně významná, jelikož dle zdrojů ve kterých jsou algoritmy popsány, by tyto algoritmy měly pracovat v reálném čase. Dle naměřených výsledků se podařilo ověřit, že algoritmy jsou opravdu schopny vesměs pracovat v reálném čase.

Vyjímkou je Kuwaharův filtr, který je schopen pracovat v reálném čase pouze na výkonné grafické kartě a to se ještě pocítá s tím, že požadavky na zpracování v reálném čase jsou nižší než u běžného videa, tj. méně než 30 snímků za vteřinu.

Pro testování byly použity dva stolní počítače, jeden z nich byl vybaven procesorem INTEL Core i7, 8GBB RAM a GeForce 9600.

Druhý stolní počítač byl vybaven procesorem Core2 Duo 7700, 4GB RAM, 8800 GTX.

Třetím testovaným strojem byl notebook s procesorem Core2 Duo 7700, 4GB RAM a grafickou kartou ATI HD 4330.

Tabulka 9.1: Orientovaný bilaterální filtr + detekce hran

	GeForce 9600	8800 GTX	ATI HD 4330
640 x 480	5,1 ms	8,2 ms	10,2 ms
1280 x 720	14,2 ms	23,8 ms	28,6 ms
1920 x 1080	29,2 ms	48,4 ms	56,2 ms

Tabulka 9.2: Anizotropický Kuwaharův filtr + detekce hran

	GeForce 9600	8800 GTX	ATI HD 4330
640 x 480	131,2 ms	192,4 ms	223,9 ms
1280 x 720	412,6 ms	532,6 ms	618,5 ms
1920 x 1080	722,5 ms	821,3 ms	951,4 ms

Při navrhování metod nerealistického zobrazení obvykle není kladen příliš velký důraz na jejich efektivitu, jelikož je upřednostňována kvalita výstupu před rychlostí. I proto je možné říci, že určitě existují metody poskytující kvalitnější výsledky než jsou ty prezentované v této práci, ovšem málokterá z nich je takto výpočetně nenáročná.

Tabulka 9.3: Shock based filter + detekce hran

	GeForce 9600	8800 GTX	ATI HD 4330
640 x 480	6,2 ms	11,8 ms	14,6 ms
1280 x 720	16,4 ms	31,4 ms	36,4 ms
1920 x 1080	30,2 ms	53,2 ms	68,2 ms

Tabulka 9.4: Pouze detekce hran

	GeForce 9600	8800 GTX	ATI HD 4330
640 x 480	3,2 ms	5,4 ms	7,2 ms
1280 x 720	8,9 ms	14,8 ms	22,5 ms
1920 x 1080	19,1 ms	29,2 ms	46,3 ms

## 9.2 Grafické výstupy programů

V této kapitole bych ráda ukázala některé výsledky implementované aplikace. Jelikož nemá smysl zahrnovat čtenáře obrovským množstvím snímků, tak budou ukázány pouze ty, které vznikly díky zobecněním provedeným ve způsobu provádění zpracování videa, navrženým v této práci. Některé ukázky videosekvencí vygenerovaných implementovaným systémem jsou ukázány na přiloženém CD.

Jelikož je těžké zobrazit zpracované video, tak jsem zvolila pro ukázkou dva po sobě následující snímky ve, kterých je snad dost pohybu, aby byla ukázána efektivita zvolených metod. Video ze kterého pochází snímky bylo vytvořeno skupinou "The Procrastinators" na MIT.



Obrázek 9.1: Vstupní snímky

Na obrázcích 9.3 je vidět aplikace hranového detektoru na výstup Kuwaharova filtru. Výsledky jsou asi o něco horší, než při použití původní verze s bilineárním filtrem, jelikož detaily obrázku jsou více rozmazané.

Naopak využití Kuwaharova filtru má tu výhodu, že nevyžaduje kvantizaci barev a proto ve videu nejsou ostré přechody mezi jednotlivými regiony.

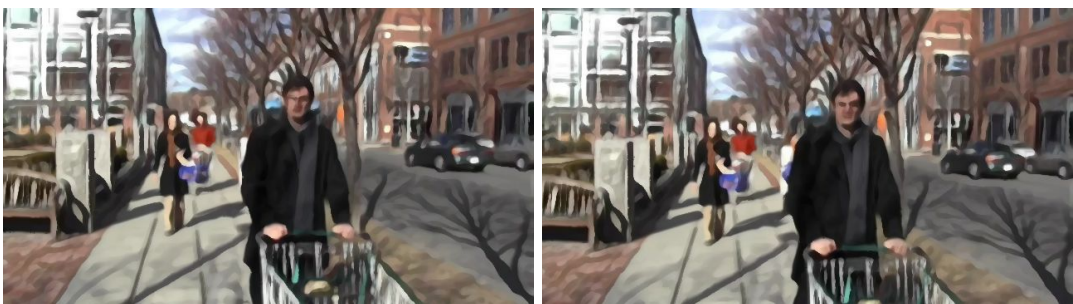
Na obrázku 9.4 je výsledek aplikace shock filtru na daný obrázek. Tato metoda odstraní z jednotlivých snímků poměrně velké množství detailů, ovšem to je přímo funkcí tohoto filtru, takže tento výsledek bylo možné očekávat. Shock filtr slouží k tvorbě vysoce abstrahovaných obrázků, což zde bylo splněno.



Obrázek 9.2: Bilaterární filtr + detekce hran



Obrázek 9.3: Kuwaharův filtr + detekce hran



Obrázek 9.4: Aplikace shock filtru

Obrázek 9.5 následně zobrazuje výsledek aplikace detektoru hran na výstup shock filtru. V těchto obrázcích jsou hrany velmi výrazné a zároveň je odstraněno příliš velké množství detailů v obraze.



Obrázek 9.5: Shock filtr + detekce hran

Poslední obrázek 9.7 ukazuje výsledek aplikace detekce hran, až po kompletním zpracování obrázku bilaterálním filtrem a po jeho kvantizaci. Tento způsob dává poměrně dobré výsledky pro samostatné obrázky, ovšem pro video není příliš vhodný, jelikož hrany se mezi jednotlivými snímky výrazně mění.



Obrázek 9.6: Detekce hran na již zkvantizovaném obrázku



Obrázek 9.7: Pouze detekované hrany

### 9.3 Reakce diváků

Někteří autoři metod pro abstrakci videa často uvádějí jako motivaci pro tvorbu takových metod nejen estetické požadavky na výstupní video, ale také zlepšení kvality vizální komunikace s uživatelem, tím že jsou zdůrazněny pouze významné prvky videa a ty méně významné jsou odstraněny nebo zjednodušeny. Pozorovatel takového obrázku by tedy měl být schopen ho rychleji identifikovat, či si ho snadněji zapamatovat.

Ovšem v rozporu s těmito tvrzeními, téměř nikdo se nezabývá tím zda je reakce uživatelů opravdu taková, jakou autoři očekávají, tj. zda jsou opravdu lidé schopni si daný obrázek lépe zapamatovat, či ho rozpoznat. Jako jeden z mála se takovými studii zabýval H. Winnemllöer ve svém článku [24]. A následně toto téma také rozvíjel ve své dizertační práci [23]. V těchto studiích se ukazuje, že doba nutná pro rozpoznání obrázku se opravdu zkrátila a lidé jsou schopni si opravdu lépe zapamatovat dané obrázky.

Průzkumem reakce diváků na podobně zpracované video se pravděpodobně dosud nikdo nezabýval. Průzkum reakce diváků na nerealisticky zobrazená videa nebyl hlavním cílem této práce, i přesto byla proveden menší výzkum. Prováděné průzkumy nebyly empiricky měřitelné, tak jako průzkumy prováděné v [23], ale byly založeny pouze na subjektivních dojmech diváků. I přesto tento průzkum určitě jistou váhu má.

Průzkum byl prováděn následovně: Nejprve bylo vygenerováno 10 nerealisticky zobrazených videosekvencí, které byla ukázány postupně osmnácti lidem. Po shlédnutí byl zjišťován jejich názor, na to zda si obsah videa lépe zapamatovali. Z těchto osmnácti osob osm tvrdilo, že se obsah videa vnímá a pamatuje hůř jelikož příliš vnímali samotné nerealistické zobrazení. Pro šest osob byla schopnost pamatovat si a vnímat obsah videí přibližně stejná a pouze čtyři lidé tvrdili, že se jim obsah videa vnímá lépe.

Z tohoto průzkumu samozřejmě nelze vyvozovat nějaké významné závěry, slouží pouze jako orientační. I přesto z něj je možné vyvodit, že nerealistické zobrazení videa pravděpodobně nebude moc vhodné například pro zobrazení celovečerního filmu, kdy chceme vnímat především děj filmu, ale má své místo především u krátkých videosekvencí, kdy je cílem zaujmout diváka. Dále může mít význam u videokonferencí s využitím malých mobilních zařízení. Zde se není nutné zabývat tím, zda divák je schopen vnímat děj a je především důležité, aby byl člověk schopen toho druhého na obrazovce poznat.

# Kapitola 10

## Závěr

Cílem této práce bylo nastudování problematiky nerealistického zobrazení videa a technik používaných v této oblasti pro vytvoření nerealistického výstupu a poté implementace vybraných technik ve vlastní aplikaci.

Techniky nerealistického zobrazení videa byly nejprve popsány stručně a poté byla jedna z technik rozebrána podrobněji, což sloužilo také jako základ pro tuto práci. Tato technika byla následně zkombinována s několika dalšími, díky čemuž byl původní postup zobecněn a umožňoval uživateli větší svobodu při volbě průběhu zpracování videa.

Podle takto navrženého postupu byl následně vytvořen program umožňující aplikovat tyto metody nerealistické zobrazení na vstupní videosekvenci dle volby uživatele. Při volbě implementovaných metod nerealistického zobrazení byl kladen důraz na efektivitu, tak aby bylo možné zpracovávat video v reálném čase. To se téměř povedlo s výjimkou použitého Kuwaharova filtru, který se práci v reálném čase přibližuje jen pokud aplikace běží na dosti výkonné grafické kartě.

Přínosem tohoto projektu pro mě bylo získání zkušeností se zpracováním obrazu s pomocí grafické karty. Před tvorbou této práce jsem měla pouze povrchní znalosti v této oblasti. Dále jsem si nastudovala některé zajímavé techniky zpracování obrazu, které byly při zpracování videa použity a které jsem dosud neznala.

V případě dalšího rozšiřování projektu je možné pokračovat implementací dalších metod zpracování videa a jejich zapojením do již existujícího průběhu zpracování. Další možností pokračování vývoje projektu je pokusit se nějakým novým způsobem vylepšit již existující filtry, které jsou zde použity. Z hlediska uživatelského pohodlí by bylo vhodné integrovat zpět načítání snímků přímo z videa. Načítání snímků z videa bylo v průběhu vývoje vypuštěno a je nahrazeno načtením sekvence samostatných obrázků ze zadaného adresáře.

# Literatura

- [1] Bohme, M.: *An ffmpeg and SDL Tutorial*. [Online], Verze z (2003), [cit. 2011-03-30]. URL <http://dranger.com/ffmpeg/>
- [2] Bousseau, A.; Neyret, F.; Thollot, J.; aj.: Video Watercolorization using Bidirectional Texture Advection. *ACM Transaction on Graphics (Proceedings of SIGGRAPH 2007)*, 2007.
- [3] Cabral, B.; Leedom, L. C.: Imaging Vector Fields Using Line Integral Convolution. *SIGGRAPH '93 Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, 1993.
- [4] Chen, J.; Paris, S.; Durand, F.: Real-time Edge-aware Image Processing with the Bilateral Grid. *ACM Transactions on Graphics*, 2007.
- [5] Hays, J.; Essa, I.: Image and Video Based Painterly Animation. *NPAR 2004*, 2004.
- [6] Hertzmann, A.: Non-Photorealistic Rendering and the Science of Art. *In Proc. NPAR*, 2010.
- [7] Hertzmann, A.; Perlin, K.: Painterly Rendering for Video and Interaction. *First International Symposium on Non-Photorealistic Animation and Rendering*, 2000.
- [8] Hong, C.; Yang, Z.; Bu, J.; aj.: Cartoon-like stylization of video for real-time applications. *Multimedia and Expo*, 2008.
- [9] Kang, H.; Lee, S.; Chui, C. K.: Flow-Based Image Abstraction. *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*, 2009.
- [10] Kang, H.; Leey, S.; Chuiz, C. K.: Coherent Line Drawing. *Proc. ACM Symposium on Non-photorealistic Animation and Rendering*, 2007.
- [11] Klein, A.; Grant, T.; Finkelstein, A.; aj.: Video Mosaics. *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, 2002.
- [12] Kuwahara, M.; Hachimura, K.; Ehiu, S.; aj.: Processing of ri-angiocardigraphic images. *Digital Processing of Biomedical Images*, 1976.
- [13] Kyprianidis, J. E.; Döllner, J.: Image Abstraction by Structure Adaptive Filtering. *In Proc. EG UK Theory and Practice of Computer Graphics*, 2008.
- [14] Kyprianidis, J. E.; Kang, H.: Image and Video Abstraction by Coherence-Enhancing Filtering. *Proceedings Eurographics 2011*, 2011.

- [15] Kyprianidis, J. E.; Kang, H.; Döllner, J.: Image and Video Abstraction by Anisotropic Kuwahara Filtering. *Computer Graphics Forum*, 2009.
- [16] Litwinowicz, P.: Processing Images and Video for An Impressionist Effect. 1997.
- [17] Marr, D.; Hildreth, E.: Theory of edge detection. *Proc. Royal Soc. London*, 1980.
- [18] O'Donovan, P.; Hertzmann, A.: AniPaint: Interactive Painterly Animation From Video. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 2011.
- [19] Papari, G.; Petkov, N.; ; aj.: Artistic Edge and Corner Enhancing Smoothing. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 2007.
- [20] Shen, L. Q.; Shen, D. G.; Qi, F. H.: Edge Detection on Real Time Using LOG Filter. *ISSIPNN*, 1994.
- [21] Wang, J.; Xu, Y.; Shum, H.-Y.; aj.: Video Tooning. 2004.
- [22] Weickert, J.: Coherence-Enhancing Shock Filters. *Lecture Notes in Computer Science*, 2003.
- [23] Winnemölle, H.: Perceptually-motivated Non-Photorealistic Graphics.
- [24] Winnemöller, H.; Olsen, S. C.; Gooch, B.: Real-Time Video Abstraction. *ACM Siggraph*, 2006.
- [25] Yoon, J.-C.; Lee, I.-K.; Kang, H.: Video Painting Based on a Stabilized Time-Varying Flow Field. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 2011.
- [26] Zhang, S.-H.; Li, X.-Y.; Hu, S.-M.; aj.: Online Video Stream Stylization.
- [27] Zhao, H.; Jin, X.; Shen, J.; aj.: Real-time feature-aware video abstraction. *The Visual Computer: International Journal of Computer Graphics*, 2008.

# Příloha A

## Obsah příloženého CD

- /src/ - zdrojové kódy vytvořené aplikace
- /build/ - soubory pro Visual Studio 10 a Visual Studio 9 umožňující kompilaci aplikace
- /examples/ - ukázky vstupních a vygenerovaných výstupních obrázkových sad
- /doc/ - programová dokumentace a manuál k programu
- /report/ - technická zpráva ve formátu PDF a zdrojové kódy v LaTeXu
- README - informace o aplikaci
- INSTALL - instalační návod