České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra kybernetiky

Přenos stylu na 3D model v reálném čase pomocí neuronové sítě

Real-Time Style Transfer to 3D Models Using Deep Convolutional Networks

BAKALÁŘSKÁ PRÁCE

Studijní program: Otevřená informatika
Specializace: Základy umělé inteligence a počítačových věd

Vypracoval: Petr Šádek
Vedoucí práce: prof. Ing. Daniel Sýkora, Ph.D.
Rok: 2022

ČVUT ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Šádek** Jméno: **Petr** Osobní číslo: **474479**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Otevřená informatika**

Specializace: **Základy umělé inteligence a počítačových věd**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Přenos stylu na 3D model v reálném čase pomocí neuronové sítě

Název bakalářské práce anglicky:

Real-Time Style Transfer to 3D Models Using Deep Convolutional Networks

Pokyny pro vypracování:

Prostudujte techniky pro přenos výtvarného stylu z kreslené předlohy na 3D model v reálném čase, jenž využívají syntézu založenou na použití záplat [1, 2]. Porovnejte jejich funkcionalitu s metodami využívajícími neuronové sítě [3, 4]. Vytvořte aplikaci, která umožní vykreslit jednoduchou 3D scénu a interaktivně měnit polohu kamery. Výsledný obraz následně použijte jako vstup do neuronové sítě, která provede vlastní stylizaci. Pokuste se co nejvíce optimalizovat průběh výpočtu tak, aby bylo možné dosáhnout interaktivní odezvy na GPU. Prozkoumejte možnosti zlepšení výstupu pomocí přidání další informace na vstupu tak, jako to dělají přístupy založené na použití záplat [1, 2] (např. informace o hloubce, normálové mapě či pozici světla). Výslednou implementaci ověřte na sadě vstupních dat, které dodá vedoucí práce.

Seznam doporučené literatury:

[1] Sýkora et al.: StyleBlit: Fast Example-based Stylization with Local Guidance, Computer Graphics Forum 38(2):83–91, 2019.
[2] Hauptfleisch et al.: StyleProp: Real-time Example-based Stylization of 3D Models, Computer Graphics Forum 39(7), 2020.
[3] Futschik et al.: Real-Time Patch-based Stylization of Portraits Using Generative Adversarial Network, Proceedings of the 8th ACM/EG Expressive Symposium, pp. 33-42, 2019.
[4] Texler et al.: Interactive Video Stylization Using Few-shot Patch-based Training, ACM Transactions on Graphics 39(4):73, 2020.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

prof. Ing. Daniel Sýkora, Ph.D., Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **07.01.2021** Termín odevzdání bakalářské práce: **04.01.2022**

Platnost zadání bakalářské práce: **30.09.2022**

prof. Ing. Daniel Sýkora, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

© VUT v Praze, Design: VUT v Praze, VIC

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studenta

© VUT v Praze, Design: VUT v Praze, VIC

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 4. ledna 2022

....

Petr Šádek

Poděkování

Děkuji prof. Ing. Danielovi Sýkorovi, Ph.D. za vedení mé bakalářské práce a za jeho rady. Také děkuji své rodině a přátelům za jejich podporu.

Petr Šádek

Název práce:
Přenos stylu na 3D model v reálném čase pomocí neuronové sítě

Autor: Petr Šádek

Studijní program: Otevřená informatika
Obor: Základy umělé inteligence a počítačových věd
Druh práce: Bakalářská práce

Vedoucí práce: prof. Ing. Daniel Sýkora, Ph.D.

Abstrakt: S dnešním vývojem neuronových sítí se stále rozšiřují jejich aplikace. Jednou z nich je přenos výtvarného stylu z jednoho obrázku na druhý. Tedy výsledný obrázek si zachová jasné rysy objektů obsažených v něm, ale změní se jeho barvy a lokální textury. Existuje několik metod s použitím neuronových sítí i bez nich, které tento problém řeší a každá má své výhody i nevýhody. Tato práce je zaměřená na přenos stylu na počítačem vykreslený obrázek 3D modelu. Model se nachází v interaktivní scéně, a proto je potřeba metoda s co nejmenším časem vyhodnocení pro vysokou odezvu. Cílem je vybrat správnou metodu a přizpůsobit výpočet této aplikaci.

Klíčová slova: přenos stylu, neuronové sítě, 3D počítačová grafika

Title:
Real-Time Style Transfer to 3D Models Using Deep Convolutional Networks
Author: Petr Šádek

Abstract: With today's development of neural networks, their applications are still expanding. One of them is the transfer of artistic style from one image to another. Thus, the resulting image retains the clear features of the objects contained in it, but changes its colors and local textures. There are several methods with and without use of neural networks that solve this problem, and each has its advantages and disadvantages. This work is focused on style transfer on a computer-rendered image of a 3D model. The model is in an interactive scene, so a method with the shortest possible evaluation time for a high response is needed. The goal is to choose the right method and adapt the calculation to this application.

Key words: style transfer, neural networks, 3D computer graphics

Obsah

Seznam použitých zkratek xi

Seznam obrázků xii

Úvod 1

1 Neuronové sítě 3
1.1 Perceptron . . . 3
1.2 Optimalizace . . . 3
1.3 Konvoluční neuronová síť . . . 4
1.4 Generativní kontradiktorní síť (GAN) . . . 4

2 Přenos stylu 5
2.1 Metody založené na použití záplat . . . 6
2.1.1 Analogie obrázků . . . 6
2.1.2 StyLit . . . 8
2.1.3 FaceStyle . . . 9
2.1.4 EbSynth . . . 11
2.1.5 StyleBlit . . . 12
2.1.6 StyleProp . . . 13
2.2 Metody s využitím neuronových sítí . . . 15
2.2.1 Translační síť se ztrátovou funkcí pro každý pixel . . . 15
2.2.2 Ztrátová funkce podle aktivací vrstev na klasifikační síti . . . 15
2.2.3 Kombinace ztrátových funkcí pro každý pixel a podle aktivace vrstev na klasifikační síti . . . 16
2.2.4 Stylizace tváří . . . 17
2.2.5 Stylizace založená na použití záplat . . . 18

3 Implementace 21
3.1 Struktura projektu . . . 21
3.2 Uživatelské rozhraní . . . 22
3.3 Vykreslování . . . 23
3.4 Trénování neuronové sítě . . . 24
3.5 Aplikace neuronové sítě . . . 24
3.5.1 Pomocí PyTorch C++ API . . . 24
3.5.2 Vlastní implementace s OpenGL a GLSL . . . 25
3.6 Přidání pomocných kanálů . . . 26
3.7 Celý proces vyhodnocení a zobrazení . . . 27
3.8 Knihovny . . . 27

4 Výsledky 29
4.1 Trénování . . . 29
4.1.1 Přenos stylu na kouli . . . 30

4.1.2 Trénování na kouli použito na složitější model 32
4.1.3 Přenos stylu na složitější model 33
4.2 Vliv transformace 3D modelu 35
4.3 Interaktivita 37

Závěr **39**

Bibliografie **41**

Přílohy **43**
A Manuál k aplikaci 43
B Zdrojový kód projektu 44
C Vlastní grafický engine 44
D Interactive Video Stylization Using Few-Shot Patch-Based Training 44
E Použité knihovny 44

Seznam použitých zkratek

GAN Generativní kontradiktorní síť (Generative adversarial network)

GUI Grafické uživatelské rozhraní (Graphical user interface)

GPU Graphics processing unit

API Application programming interface

RAII Získávání prostředků je inicializace (Resource acquisition is initialization)

NNF Pole nejbližších sousedů (Nearest neighbor field)

FPS Počet snímků za sekundu (Frames per second)

GLSL OpenGL Shading Language

GLFW Graphics Library Framework

GLM OpenGL Mathematics

SIMD Single instruction, multiple data

ReLU Rectified Linear Unit

Seznam obrázků

2.1 Příklad přenosu stylu z [3]. Vlevo (a) vzorový styl, (b) vstupní obrázek, vpravo výsledná stylizace. 5
2.2 Příklad analogií z [5]. 6
2.3 Příklad textur podle čísel z [5]. 7
2.4 Příklad analogie z [8]. 8
2.5 Přídavné kanály v [8]. (a) celý render, (b) difuzní složka, (c) specular složka, (d) složka difuzních odrazů, (e) složka difuzní interreflexe, (f) manuálně vytvořený styl. 8
2.6 Přenos stylu na záznam videa v [9]. 9
2.7 Vodící kanály z [9]. $G_{\texttt{seg}}$ segmentace obličeje, $G_{\texttt{pos}}$ informace o pozici, $G_{\texttt{app}}$ vzhled, $G_{\texttt{temp}}$ informace pro zajištění dočasné koherence videa. . 10
2.8 Vodící kanály z [12]. $G_{\texttt{col}}$ původní obrázek, $G_{\texttt{mask}}$ maska, $G_{\texttt{pos}}$ informace o pozici, $G_{\texttt{edge}}$ hrany, $G_{\texttt{temp}}$ informace pro zajištění dočasné koherence videa. 11
2.9 Příklad přenosu stylu na 3D model z [13]. 12
2.10 Znázornění překopírování sektoru s chybou nepřasahující požadovanou hranici z [13]. 13
2.11 Přenos stylu na 3D model z [15]. (a) je vzor stylu, (c, d) jsou výstupní stylizace pro konkrétní pohledy kamery. 13
2.12 Znázornění celého procesu syntézy z [15]. Nejprve manuální příprava stylizace, poté preprocesing NNF a nakonec samotné vykreslování. . 14
2.13 Demonstrace přenosu stylu z [19]. 16
2.14 Ilustrace modelu metody z [20]. Jako klasifikační síť použili konkrétně model VGG-16 natrénovaný na ImageNet datasetu. 17
2.15 Demonstrace přenosu stylu na lidské tváře z [11]. 17
2.16 Změna architektury sítě mezi [20] a [11]. 18
2.17 Znázornění trénovací strategie použité v [21]. 18
2.18 Odstranění nejednoznačnosti (b) a (e) v [21]. 19

3.1 Vzhled aplikace vykonávající přenos stylu. 22
3.2 Okénko s podrobným nastavením aplikace. 23
3.3 Vstupní obrázek. 24
3.4 Vzorový obrázek s původní stylizací. 24
3.5 Výstup z naučené sítě. 24
3.6 Normálové vektory. 26
3.7 Směr ke světlu. 26
3.8 Vlastní textura. 26

4.1 Vzorový obrázek se stylizací. 29
4.2 Vstupní stíněná koule. 30

4.3 Přenos stylu natrénovaný pouze se vstupem stínovaného modelu. . . . 30
4.4 Vstupní normály. 31
4.5 Natrénováno pomocí normál. 31
4.6 Rovnoměrně rozpoložené gauss disky. 31
4.7 Výsledek sítě natrénované s pomocí gauss disků. 31
4.8 Rovnoměrně rozpoložené barvy gauss disků na povrchu koule uložené ve sférických souřadnicích do 2D textury. 32
4.9 Koule vykreslená pomocí textury gauss disků. 32
4.10 Výsledek sítě natrénované s pomocí textury gauss disků. 32
4.11 Výstup neuronové sítě (c) naučené pomocí normálových vektorů koule použité na vstup stínovaného modelu králíka (a) a jeho normálových vektorů (b). 33
4.12 Vzorový styl golema vytvořený pomocí EbSynth. 33
4.13 Vzorový styl králíka vytvořený pomocí EbSynth. 33
4.14 Textura gauss disků pro model golema. 34
4.15 Aplikovaná textura gauss disků vytvořená pro model golema. 34
4.16 Aplikovaná textura gauss disků vytvořená pro model králíka. 34
4.17 Výsledek stylizace vedené texturou gauss disků pro model golema. . . 35
4.18 Výsledek stylizace vedené texturou gauss disků pro model králíka. . . 35
4.19 Transformace koule korektně zachovávající požadovaný styl. 35
4.20 Transformace koule, které nezachovávají požadovaný styl. 36
4.21 Výsledky přenosu stylu pro redukované modely neuronových sítí. . . . 38

Úvod

Přenos výtvarného stylu na obrázek může být užitečný nástroj v grafické tvorbě. Obecně lze přenos stylu popsat jako: přenos barev a malých kousků textur z jednoho obrazu na druhý při zachování významu obrazu (například tvary a umístění objektů a podobně). Konkrétní příklad by mohl být obrázek člověka kterému dáme vzhled kresby pomocí přenosu stylu. Tedy na obrázku se zachová člověk, ale změní se jeho vzhled tak, že vypadá jako by byl nakreslený. Další aplikací může být aplikace stylu na celou sekvenci videa. Pro pár vybraných snímků umělec vytvoří verzi se stylem a algoritmus poté pomocí těchto příkladů aplikuje tento styl na ostatní snímky videa.

Problém kterým se zabývá tato práce je přenos stylu v reálném čase na 3D model vykreslený počítačem. Pojmem "v reálném čase" se myslí dostatečně rychlé vyhodnocení výsledného algoritmu na to, aby se pro člověka jevilo jako instantní a v ideálním případě dokázalo vygenerovat 60 snímků za sekundu. Pokud toto je možné, můžeme měnit pozici a natočení modelu ve scéně a okamžitě pozorovat jeho verzi s aplikovaným stylem. Využití této aplikace by mohlo být například v CGI filmech nebo počítačových hrách.

Cílem je vytvořit aplikaci, která bude zobrazovat jednoduchou scénu nebo model. Uživatel bude moci pohybovat s kamerou nebo s objektem. Na obrázek se bude přenášet předem definovaný styl. Je důležité vybrat správnou metodu přenosu stylu, aby měla vhodné vlastnosti pro tento případ. Nejsložitější část bude optimalizovat implementaci, tak aby se styl přenášel v reálném čase.

Nejprve si uvedeme základní pojmy týkající se neuronových sítí, protože na jejich využití se tato práce zaměřuje. Poté se podíváme na různé metody přenosu stylu. Představíme si metody jak bez využití neuronových sítí, tak i s využitím neuronových sítí. Popíšeme jejich vlastnosti a porovnáme metody mezi sebou. V další části nahlédneme na implementaci aplikace vykonávající přenos stylu na 3D model, která byla v rámci tohoto projektu vytvořena. Nakonec zhodnotíme výstupy vytvořené aplikace.

Kapitola 1

Neuronové sítě

Umělé neuronové sítě mají dnes široké spektrum využití. Díky své vysoké schopnosti aproximovat model reprezentující daný problém, mohou řešit úlohy jako: klasifikace objektů na obrázcích, rozpoznávání umístění objektů na obrázku, rozpoznávání řeči a mnoho dalších. Jedním z těchto využití je také přenos stylu, kterým se tato práce zabývá. Proto si zde uvedeme několik základních pojmů o neuronových sítích.

1.1 Perceptron

Model neuronu v umělých neuronových sítích je inspirován neuronem z biologie. Má několik vstupů připojených z ostatních neuronů a jeden výstup. Pokud na vstupech nastane určitá kombinace intenzit signálů, na kterou je neuron nastavený, tak aktivuje signál na svém výstupu.

Matematický model který replikuje toto chování se jmenuje perceptron. Perceptron se skládá ze dvou částí: afinní funkce a aktivační funkce. Afinní funkce rozděluje vstupní signály pomocí nadroviny na ty pro které se výstup aktivuje a ty pro které ne. Aktivační funkce poté vnese do výsledku nelinearitu, která mění chování výstupu podle toho na jaké straně oddělující nadroviny se nachází vstupní signály. Jako aktivační funkce se používají například sigmoid, tanh a ReLU (nejvíce používaná, jedná se o $f(x) = max(0, x)$).

Perceptrony jsou uspořádány do vrstev. Síť se skládá z několika vrstev. Vrstvy po sobě následující mají mezi sebou propojené vstupy a výstupy neuronů. Zhruba platí že čím více má neuronová síť vrstev a neuronů, tím obecnější problémy dokáže řešit.

1.2 Optimalizace

Samotný model neuronové sítě však na řešení problémů nestačí. Je potřeba správně nastavit parametry jednotlivých neuronů, aby síť řešila danou úlohu. K tomu si definujeme takzvanou ztrátovou funkci, která přebírá výsledek neuronové sítě a zhodnotí jak moc se liší od správného výsledku. Typ ztrátové funkce závisí na řešeném problému. Ve chvíli kdy tuto funkci máme, můžeme pomocí optimalizačních metod z matematické analýzy najít hodnoty parametrů neuronové sítě, pro které je hodnota ztrátové funkce minimální (ve skutečnosti jen lokálně minimální). Tedy neuronová síť dává výsledky co se nejvíc blíží těm správným.

Konkrétně se používá metoda gradientního sestupu a její varianty. Protože jsou všechny funkce ze kterých se síť skládá diferencovatelné, je možné spočítat gradient pro konkrétní vstup postupným aplikováním řetízkového pravidla na neurony od konce sítě (anglicky se toto označuje termínem backpropagation). Metoda gradientního sestupu funguje tak, že se hodnoty parametrů iterativně posouvají proti směru gradientu (gradient je zde směr největšího růstu ztrátové funkce), a tím se postupně parametry posouvají do lokálního minima ztrátové funkce (pokud nastává konvergence). Pokud máme dostatečné množství příkladů vstupů a správných výstupů, můžeme pomocí nich vykonávat gradientní sestup. Tomuto postupu optimalizace neuronové sítě se také často říká učení nebo trénování neuronové sítě, protože se podle předložených příkladů postupně zlepšuje neboli učí/trénuje.

1.3 Konvoluční neuronová síť

U přenosu stylu se stejně jako u jiných úloh které zpracovávají obrázky, používají konkrétně konvoluční neuronové sítě. Jejich výhoda je zachování informace o pozici dat. Ta je u obrázků zásadní. Funguje to tak, že oproti klasickým neuronovým sítím nejsou mezi vrstvami propojené neurony každý s každým, ale jsou reprezentovány konvolučním filtrem, který se postupně aplikuje na jednotlivé části výstupu z předchozí vrstvy. Výstupy jsou potom umístěné ve vrstvách naproti sobě a je pro síť snadnější lépe reprezentovat model problému.

1.4 Generativní kontradiktorní síť (GAN)

GAN [1] je způsob trénování neuronové sítě, který se skládá ze dvou neuronových sítí nazvaných generátor a diskriminátor. Generátor na základě daného vstupu generuje data příslušné domény. Diskriminátor na vstupu přijímá tyto generované data a klasifikuje, zda do domény patří či ne. Obě neuronové sítě spolu hrají hru s nulovým součtem, ve které se obě postupným učením zlepšují. Generátor se učí generovat více věrohodná data. Diskriminátor se učí lépe rozpoznávat falešná data od správných. Zlepšení jednoho z nich je penalitou pro druhého. V ideálním případě to skončí tím, že generátor umí vytvářet data tak dobře, že diskriminátor je nedovede rozopoznat a má 50% chybovost.

Hlavní výhodou je lepší generalizace výstupů sítě. Tento způsob je použit u některých metod přenosu stylu pomocí neuronových sítí, které jsou uvedené v následující kapitole. Jednou z dalších aplikací je například projekt StyleGAN [2], který využívá GAN pro věrohodné náhodné generování lidských tváří.

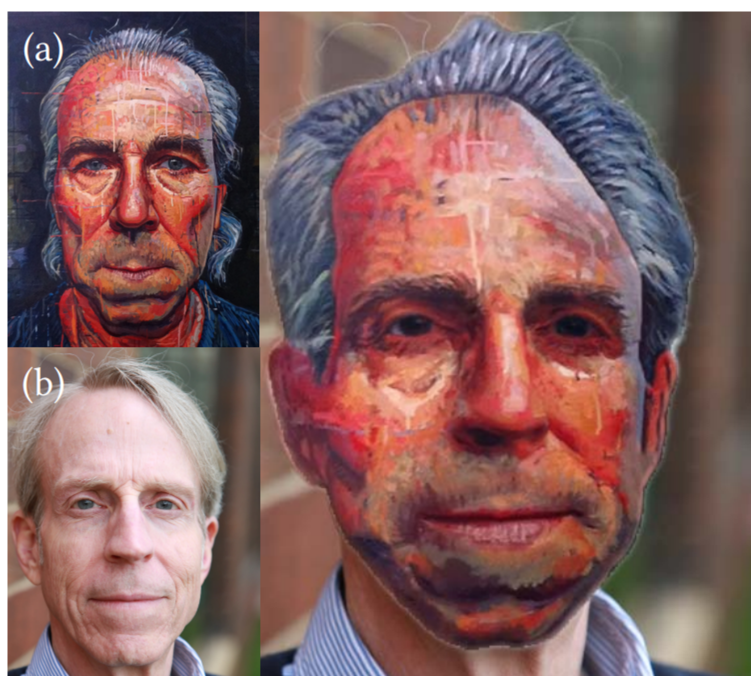
Kapitola 2

Přenos stylu

Definujme přenos stylu jako přenos barev a malých kousků textur z jednoho obrázku na druhý při zachování významu obrázku. Konkrétně je významem myšleno například tvary objektů, jejich umístění a jejich hrany.

Konkrétní příklad by byl, že vezmeme například nějakou fotografii a upravíme ji tak, že vypadá jako kdyby to byla například nějaká malba štětcem, nebo kresba tužkou. To způsobí jinou texturu, ale samotné barvy také nemusí úplně sedět. Můžeme třeba studené barvy zaměnit za teplejší.

Kategorie metod přenosu stylu, kterou se tato práce zabývá, je přenos stylu založený na předloze. Ten požaduje na vstup vzorový obrázek (nebo i vícero obrázku) stylu a obrázek, na ktrerý se má styl přenést. Na výstupu algoritmu je poté obrázek, který je významově stejný, ale jeho vzhled je upraven podle vzorového stylu (příklad na obrázku 2.1).



Obrázek 2.1: Příklad přenosu stylu z [3]. Vlevo (a) vzorový styl, (b) vstupní obrázek, vpravo výsledná stylizace.

Existují také typy přenosu stylu, které nejsou závislé na předloze. Například umělé tahy štětcem, které jsou postupně generované, tak aby barvami vyplnily spe-

cifikovaný obrázek. Příkladem tohoto typu je program FotoSketcher.

Zde si uvedeme dvě hlavní skupiny metod přenosu stylu: metody založené na použití záplat a metody využívající neuronových sítí. Tato práce je sice implementací zaměřená na metody využívající neuronových sítí, ale ty se v této oblasti začaly používat relativně nedávno (zhruba od roku 2015 [4]). Proto je vhodné tyto metody porovnat s metodami, které se používají podstatně delší dobu, konkrétně se zaměříme na metody založené na použití záplat. Nelze striktně říci, že by byly metody z jedné oblasti ve všech ohledech lepší než v druhé. Metody z obou oblastí mají různé výhody a nevýhody. Ty jsou popsány v následujících kapitolách.

2.1 Metody založené na použití záplat

Tyto metody skládají výsledný obrázek pomocí malých částí vybraných z obrázku se vzorovým stylem, takzvaných záplat (anglicky patches). Typicky jsou to čtvercové výseky z daného obrázku na libovolné pozici s velikostí v rámci desítek pixelů. Nejprve si uvedeme analogie obrázků, které jsou způsob jakým metody založené na použití záplat definují úlohu přenosu stylu a k tomu nástin obecného algoritmu, který tyto úlohy řeší. Dále následují konkrétní metody přenosu stylu, každá zaměřená na svojí specifickou problematiku.

2.1.1 Analogie obrázků

Analogie $A : A' :: B : B'$ znamená: chceme najít "analogický" obrázek B', jehož "vztah" k obrázku B je stejný jako "vztah" obrázku A' k obrázku A.

Tímto způsobem se definuje většina algoritmů založených na použití záplat. To konkrétně znamená, že předloha se skládá ze dvou obrázků, které představují vzor vstupního a výstupního obrázku, označme je A, A'. Výstupní obrázek A' je potom styl, který byl manuálně připraven podle vstupního obrázku A. Algoritmus poté na další vstupní obrázek B, ke kterému už nemáme manuálně vytvořenou stylizovanou variantu, vytvoří stylizovanou variantu B' zhruba podle toho, jak se mění vzhled mezi obrázkem A a obrázkem A'.

Obrázek 2.2: Příklad analogií z [5].

Prvotní definice tohoto přístupu pochází z [5] (příklad jedné jejich analogie na obrázku 2.2). V práci [5] byl vytvořen algoritmus, který řeší problém definovaný pomocí analogie. Algoritmus se tak chová jako obecný filtr, který lze naučit podle příkladu jeho aplikace.

Základní princip algoritmu je velice jednoduchý. Funguje tak, že pro každý pixel obrázku A' najde pixel v obrázku A, který je jeho takzvaným nejbližším sousedem. V tomto případě to znamená, že hodnoty barev daných pixelů a jejich okolí se v rámci

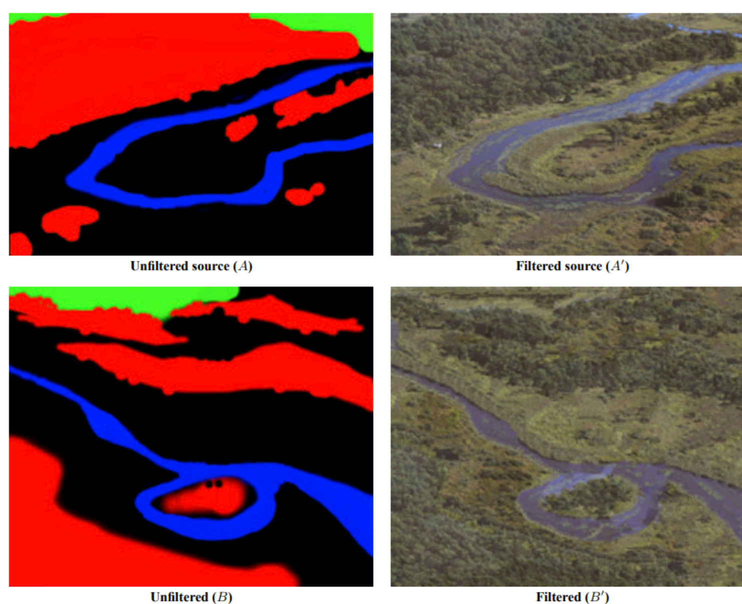
všech pixelů obrázku nejvíce shodují (euklidovská vzdálenost vektorů jejich barev je co nejmenší). Nalezené souřadnice nejbližších sousedů se uloží do pole (nearest neighbor field - NNF). Výsledná hodnota pixelu v obrázku B' je hodnota z obrázku A', která je od pixelu v B' posunutá stejně jako je v A nejbližší soused posunut vůči B. Tedy do B' se přiřadí hodnota z A' na indexu obsaženém v NNF. Celý proces je znázorněn v pseudokódu 1.

Algoritmus tak ale nečiní jen na samotném obrázku, ale má vytvořenou takzvanou pyramidu, která obsahuje i všechna nižši rozlišení (vždy o polovinu menší). Hledání nejbližších sousedů probíhá nejprve u nejnižších rozlišení. Při hledání nejbližšího souseda u vyššího rozlišení se do shody započítává i shoda ve stejném okolí v nižším rozlišení. Je zde také zajištěná koherence s již přiřazenými hodnotami sousedních pixelů. Pokud přiřazení v sousedním pixelu vede na lepší výsledky, pak je přiřazena jeho sousední záplata.

Algorithm 1 Algoritmus analogie obrázků [5]

function CREATEIMAGEANALOGY(A, A', B)
    for `each pyramid level` $l \in \{0, \cdots, L\}$ **do**
        for `each pixel` $q \in B'_l$ **do**
            $p \leftarrow BestMatch(A, A', B, B', l, q, NNF)$
            $B'_l(q) \leftarrow A'_l(p)$
            $NNF_l(q) \leftarrow p$
    return B'_L

Vektor reprezentující pixel obrázku nemusí obsahovat pouze barvu RGB, ale i různé další informace, jako jsou informace o svícení, informace o pozici a informace o hranách. Algoritmus tak lze zobecnit na mnoho činností. Také je tak možné více upřesnit chování algoritmu tím, že máme více kanálů informace.



Obrázek 2.3: Příklad textur podle čísel z [5].

Potenciální použití by bylo i pro přenos stylu, ale v [5] použití tohoto algoritmu nemělo příliš dobré výsledky. Algoritmus je možné použít na velké množství dalších úloh, jako jsou syntéza textury (například odstranění části obrázku a nahrazení

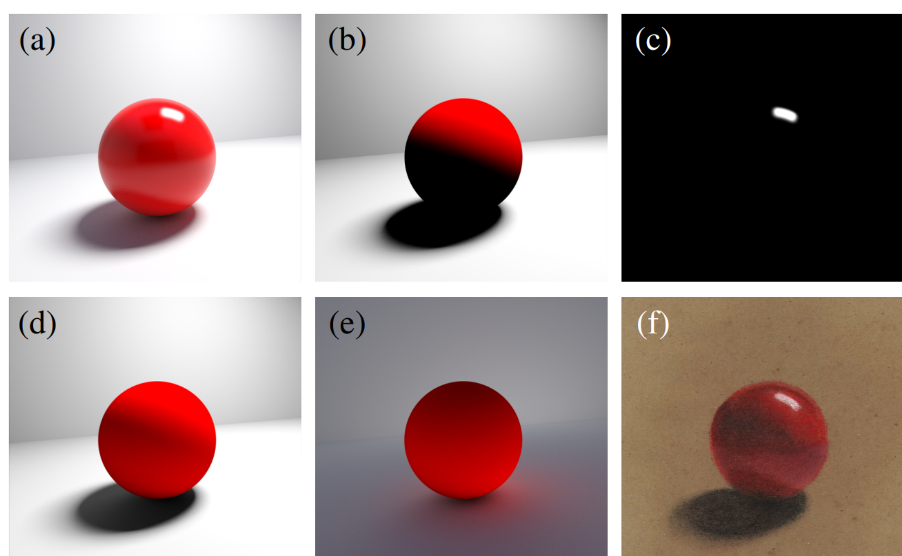
jiným kusem), zvyšování rozlišení, přenos textury (podobné jako přenos stylu, ale textura nesedí na vstupní obrázek, co se má stylizovat, jednoduše vytvoří opakující se vzor podle dané textury), textury podle čísel (anglicky texture-by-numbers, generuje stejné textury s jiným rozpoložením specifikovaném podle malého množství charakteristických barev, obrázek 2.3).

Vylepšenou verzí tohoto algoritmu je algoritmus PatchMatch [6], [7]. Ten je realizován iterativním způsobem, který postupně konverguje s větším počtem iterací. Jedna iterace je v podstatě průběh algoritmu z [5], jen má NNF ze začátku náhodně přiřazené hodnoty, které se s každou iterací postupně vylepšují. Tento algoritmus je hojně používán v následujících metodách.

2.1.2 StyLit

Obrázek 2.4: Příklad analogie z [8].

Prvním příkladem přenosu stylu definovaném pomocí analogií a využívající algoritmus PatchMatch je StyLit [8]. Tato metoda konkrétně řeší úlohu přenosu stylu na render 3D modelu. Úloha je definovaná pomocí analogie, kde vzorová dvojce je render koule A a jeho stylizovaná varianta A'. Aplikace stylu se vykonává na obrázek B, který je render nějakého složitějšího 3D modelu (obrázek 2.4).



Obrázek 2.5: Přídavné kanály v [8]. (a) celý render, (b) difuzní složka, (c) specular složka, (d) složka difuzních odrazů, (e) složka difuzní interreflexe, (f) manuálně vytvořený styl.

Hlavní úprava vůči základnímu algoritmu PatchMatch je přidání dalších kanálů informace pro každý pixel zdrojových obrázků (obrázky A a B). Konkrétně jsou přidány hodnoty jednotlivých složek svícení, ze kterých je celkové osvícení scény spočítáno. Tyto složky se dělí na difuzní složku, specular složku, složku difuzních odrazů a složku difuzní interreflexe (obrázek 2.5).

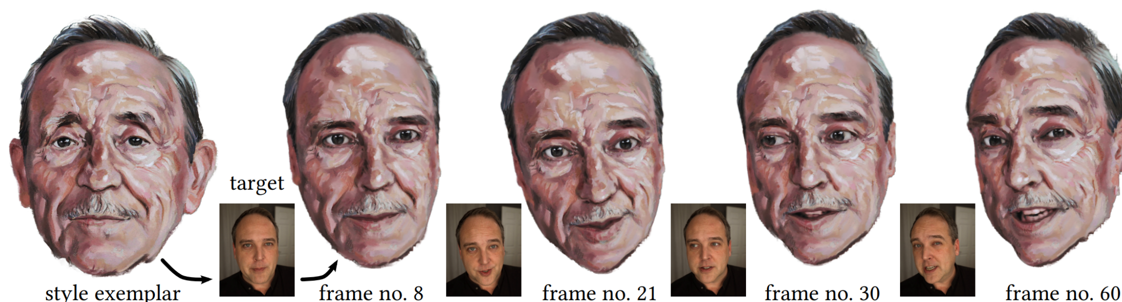
Další úpravou je také větší redukce chybných přiřazení záplat. Je zde specifikována hranice limitující množství takovýchto přiřazení. Pokud je tato hranice překročena, pak se místo přiřazování záplat podle NNF přiřazuje podle sousedních již přiřazených záplat.

Tato metoda řeší část problému, na který je tato práce zaměřená, a to konkrétně přenos stylu na 3D model. Z hlediska výsledků produkovaných touto metodou nelze nic vytknout. Styl je perfektně zachován a korektně přenesen i pro složitější 3D modely. Vzor stylu je vždy vytvořen pomocí stylizované koule (ale hypoteticky by mohl být použit jako vzor složitější model). To přidává na jednoduchosti přípravy předlohy.

Avšak požadavek na odezvu v reálném čase uvádí tuto metodu nepoužitelnou pro náš případ. Čas jednoho vyhodnocení této stylizace je v nejlepším maximálně optimalizovaném příkladě v rámci několika sekund. K tomu je potřeba udělat velice náročnou realistickou syntézu obrazu pomocí globálního osvětlení, abychom měli k dispozici všechny kanály přídavné informace. Ale pro vykreslování 3D modelu v reálném čase se používají mnohem jednodušší, méně realistické metody přímého svícení. To znemožňuje interaktivní vykreslování modelu, které bychom ideálně chtěli v rámci 60 snímků za sekundu.

2.1.3 FaceStyle

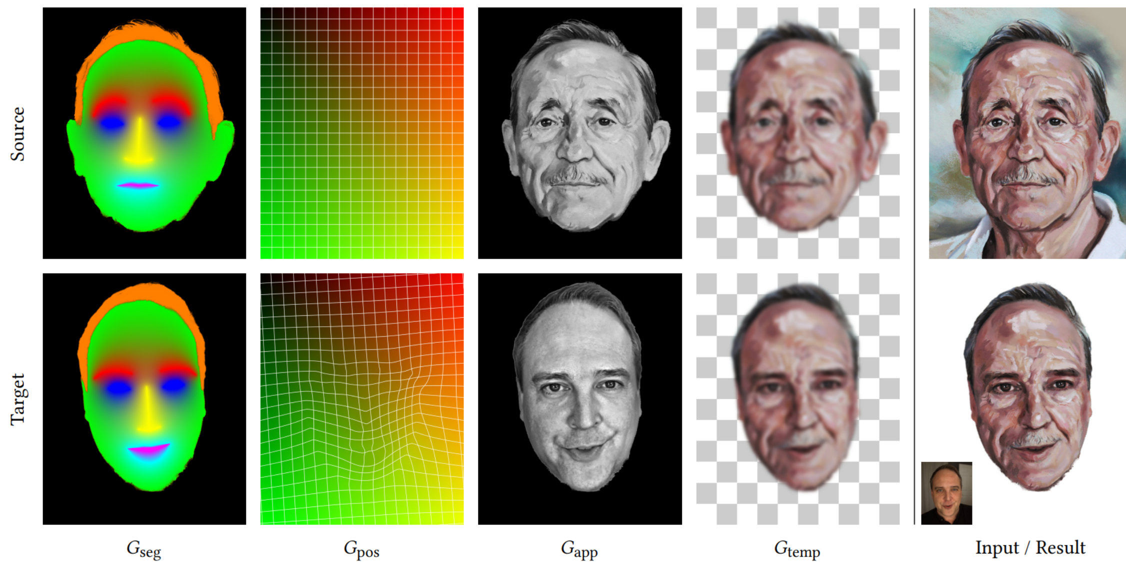
V základním principu fungování je předchozí metodě StyLit [8] velice podobná metoda FaceStyle [9]. Ta se pro změnu zaměřuje na stylizaci lidských obličejů, konkrétně stylizaci celého videa záznamu obličeje podle jednoho obrázku se vzorovým stylem (obrázek 2.6). Metoda je opět realizována totožným algoritmem a rozdíl je v použitých kanálech s doplňující informací a v jejich přípravě. Také pro změnu není potřeba vzorový pár obrázků bez stylu a se stylem. Stačí jen samotný obrázek se stylem. Podmínka je jen aby to byl portrét lidské tváře.



Obrázek 2.6: Přenos stylu na záznam videa v [9].

Obrázky s vodící informací jsou čtyři. Každý z nich obsahuje charakteristickou informaci o obsahu obrázku. Jsou jimi: informace o segmentaci obličeje, informace o pozici, informace o vzhledu a informace pro zajištění dočasné koherence videa. Obrázek se segmentací obličeje obsahuje pevně dané barvy pro různé oblasti obličeje,

tedy například nos je žlutá barva, oči modrá, vlasy oranžová. Tato segmentace je generována poměrně komplikovaným způsobem. Je generována jak pro vzorový styl, tak pro obrázky ze záznamu videa. Obrázek s informací o pozici je pro vzor stylu pouze převedením souřadnic pixelů do barev v kanálech červené a zelené pomocí normalizace na rozmezí od 0 do 1. Pro vstupní obrázek je obrázek pozic generován pomocí deformace jeho varianty pro vzor stylu. Tato deformace je vedená podle již vytvořené segmentace obličeje. Obrázek vzhledu je jednoduše vytvořen jako černobílá varianta původních obrázků s úpravou jejich intenzity. Obrázek s informací pro zajištění dočasné koherence videa obsahuje rozostřenou variantu stylizovaného předchozího snímku, který je posunut podle optického toku. Tyto vodící informace jsou znázorněny na obrázku 2.7.



Obrázek 2.7: Vodící kanály z [9]. $G_{\texttt{seg}}$ segmentace obličeje, $G_{\texttt{pos}}$ informace o pozici, $G_{\texttt{app}}$ vzhled, $G_{\texttt{temp}}$ informace pro zajištění dočasné koherence videa.

Výsledky této metody velice dobře zachovávají aplikovaný styl. Díky vodící informaci o segmentaci obličeje se na stejných částech obličeje projevují stejné části vzorového stylu. To byl v době publikace FaceStyle velký problém metod přenosu stylu pomocí neuronových sítí (například [10]), které moc nerespektovaly konkrétní rozpoložení v rámci obrázku a spíše se chovaly jako aplikace nějaké textury rovnoměrně po daném obrázku. Později ovšem byl tento způsob založený na použití záplat spojen s metodami využívajících neuronových sítí a vznikla metoda benefitující z výhod obou přístupů [11]. To je více popsáno v kapitole o stylizaci tváří pomocí neuronových sítí.

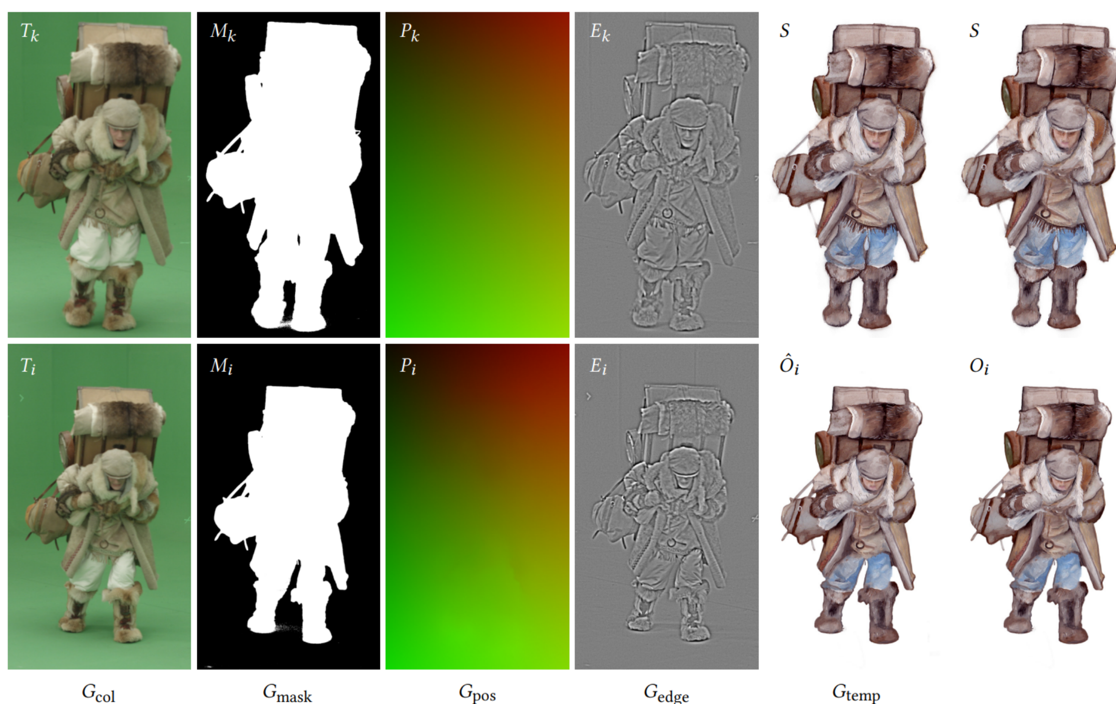
Pomocí vodícího kanálu s informací pro zajištění dočasné koherence videa, je přechod mezi jednotlivými po sobě jdoucími snímky plynulejší. Bez vzájemné závislosti snímků, by snímky měly více náhodný charakter. To způsobuje dojem rušivého přeskakování barev mezi snímky. Tento efekt je velice častý u přenosu stylu na sekvenci videa. V některých situacích však může být naopak žádaný. Tato metoda se tohoto efektu sice nezbaví úplně, ale určitým způsobem ho redukuje.

Značnou nevýhodou tohoto přístupu je opět velice časově náročné vyhodnocení. Také je zde limitace na možnosti pohybu obličeje. Pokud je obličej příliš drasticky pozměněn (například značnou rotací), výsledky jsou nesprávné.

2.1.4 EbSynth

Další metodou přenosu stylu založenou na použití záplat je EbSynth [12]. Tato metoda je podobně jako předchozí FaceStyle [9] určená na přenos stylu na sekvenci videa. Ale na rozdíl od ní se nezaměřuje na stylizaci tváří, ale umožňuje přenos stylu obecně na jakékoli video. Opět je zde využit stejný algoritmus PatchMatch [6] a rozdíl je hlavně v použitých vodících kanálech s pomocnou informací.

V této metodě je nutné, aby vytvořená předloha stylu seděla na snímek ze sekvence videa. To je podobné jako tomu bylo u metody StyLit [8] a rozdílné vůči FaceStyle [9], kde tento požadavek nebyl. Z celé sekvence videa stačí tyto stylizované varianty vytvořit jen pro pár klíčových snímků. Pokud syntéza neprodukuje dobré výsledky, lze je zlepšit přidáním dalších manuálně stylizovaných snímků.



Obrázek 2.8: Vodící kanály z [12]. $G_{\texttt{col}}$ původní obrázek, $G_{\texttt{mask}}$ maska, $G_{\texttt{pos}}$ informace o pozici, $G_{\texttt{edge}}$ hrany, $G_{\texttt{temp}}$ informace pro zajištění dočasné koherence videa.

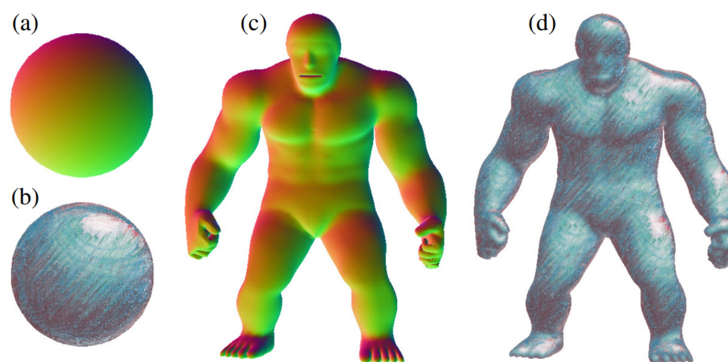
Je zde celkem použito 5 vodících kanálů (obrázek 2.8). Do nich se započítává i snímek původní sekvence videa. Dalšími kanály jsou maska, informace o pozici, informace o hranách a informace pro zajištění dočasné koherence videa. Maska bílou barvou vyznačuje objekt, který chceme stylizovat, a černou zbytek obrázku. To pomáhá s rozlišením hranic objektu, řeší problémy s překrýváním a umožňuje vrstvenou stylizaci. Informace o pozici je u prvního snímku vytvořená podobně jako u FaceStyle, kde pixely obsahují barvami zakódovanou pozici daného pixelu. Pro další snímky je tento obrázek vytvořen pomocí optického toku z minulého snímku. Tento kanál snižuje nejednoznačnost mezi původním snímkem a stylem (některé stejně barevné záplaty v původním snímku jsou stylizované jiným způsobem). Informace o hranách zachycuje hrany objektů ve snímku. Díky tomu se styl lépe uchytí na struktury obsažené ve snímku. Tento obrázek lze vytvořit například pomocí odečtení barev rozostřené varianty obrázku od původního obrázku. Obrázek s informací pro zajištění dočasné koherence videa je vždy vytvořen z předchozího stylizovaného

snímku posunutím pomocí optického toku. To podobně jako u FaceStyle zajišťuje lepší plynulost videa.

Metoda má velice kvalitní výsledky. Díky možnosti volby snímků se stylizovaným vzorem je velice flexibilní. Celkově výrazně usnadní práci oproti tomu, kdy by člověk měl manuálně vytvořit všechny snímky. Nevýhodou je náročný výpočet. Čas zpracování jednoho snímku je v rámci několika sekund. To znemožňuje interaktivitu v reálném čase. Tento problém řeší metody využívající neuronových sítí, které si později uvedeme v jejich příslušné kapitole.

2.1.5 StyleBlit

V důsledku pozorování chování metody StyLit [8] v malých lokální sektorech vznikla metoda StyleBlit [13]. Tento algoritmus používá jako vstup pro aplikaci stylu vykreslený model s vizuální reprezentací jeho normálových vektorů. Jako vzor je použita opět koule. Tedy dvojce obrázků, jeden je koule s normálovými vektory, druhý stylizovaná varianta koule. Vstup je obrázek s normálami nějakého 3D modelu a výstup je jeho stylizovaná varianta vytvořená podle stylu koule. Znázorněno na obrázku 2.9.

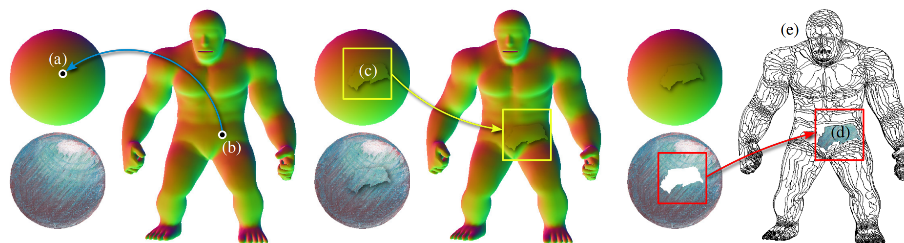


Obrázek 2.9: Příklad přenosu stylu na 3D model z [13].

Zachování koherence pomocí přepoužívání sousedních záplat v metodě StyLit vede na vznik sektorů, ve kterých jsou použity záplaty ze vzorového stylu sousedící stejným způsobem. V rámci takovýchto sektorů je koherentní chyba nulová a chybu v rozdílu vodící informace lze ohraničit horním odhadem. Díky tomu lze toto chování aproximovat pomocí překopírování celých sektorů ze vzoru stylu, v rámci kterých chyba nepřesáhne určitou pevně danou hranici. Proto je algoritmus realizován v principu jednoduše (reálná implementace je o něco složitější) jako překopírování těchto sektorů ze vzorového stylu do výsledného obrázku (znázorněno v 2.10).

Díky tomu, že vzorový obrázek je promítnutá koule, tak lze pro konkrétní bod na modelu jednoduchým způsobem najít bod v obrázku vzorového stylu se stejným normálovým vektorem. Tím se zbavíme náročné operace hledání nejbližšího souseda a redukujeme ji na operaci s konstantní asymptotickou složitostí.

Tyto optimalizace, a také paralelní implementace v rámci [13], výrazně snižují výpočetní nároky oproti metodě StyLit, a to do té míry, že lze StyleBlit vyhodnocovat interaktivně v reálném čase. Pro StyleBlit proto existuje například aplikace, kde je algoritmus vyhodnocován v rámci shaderu běžícím na grafické kartě každý snímek a uživatel může se stylizovaným 3D modelem pohybovat. Z tohoto hlediska



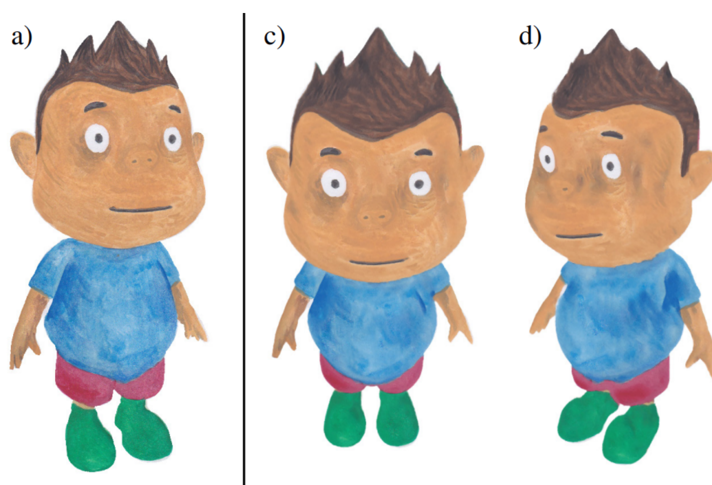
Obrázek 2.10: Znázornění překopírování sektoru s chybou nepřasahující požadovanou hranici z [13].

StyleBlit řeší úlohu, kterou se tato práce zabývá, jen k tomu nevyužívá neuronových sítí.

StyleBlit netrpí deformací texury jako je třeba u The Lit Sphere [14]. Ten obsahuje podobný nápad využití normálových vektorů pro přenos stylu, ale konkrétně pomocí enviromentálního mapování z koule. V případě algoritmu StyleBlit jsou části vzoru mapovány proporcionálně 1 : 1 na výsledek, proto ke zkreslení nedochází.

Nabízí se otázka, zda nebudou přechody mezi sektory na výsledném obrázku znatelné, tím že výsledek složíme z několika oddělených sektorů, vytažených ze vzorového stylu. Podle [13] to ve značném množství případů znatelné není, buď díky jemnosti navádění pomocí normálových vektorů, nebo stochastické přirozenosti kreslených vzorů (to je zrovna situace na obrázku 2.9). Ovšem začne to být vidět při přechodu mezi vícero snímky, kde se daný 3D model pohybuje. Tento efekt lze alespoň zakrýt pomocí náhodného chvění zakomponovaného do algoritmu StyleBlit.

2.1.6 StyleProp

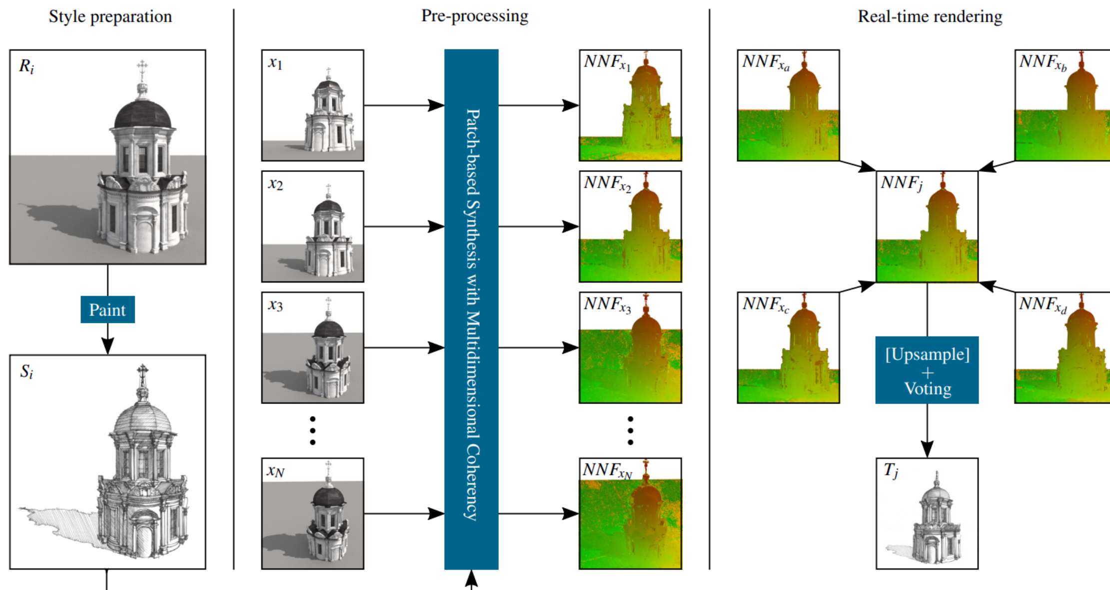


Obrázek 2.11: Přenos stylu na 3D model z [15]. (a) je vzor stylu, (c, d) jsou výstupní stylizace pro konkrétní pohledy kamery.

Další metodou přenosu stylu na 3D model v reálném čase je StyleProp [15] (obrázek 2.11). Opět je založená na algoritmu StyLit [8], ale na rozdíl od metody StyleBlit [13], která vyhodnocení zrychluje její aproximací, řeší problém předpočítáním části algoritmu StyLit před jeho použitím a při samotném použití se výsledek z těchto předpočítaných dat složí výpočetně nenáročným způsobem. Tím je zajištěná

interaktivita a opět lze vytvořit aplikaci, ve které si uživatel může se stylizovaným modelem volně hýbat.

Tato metoda je specifická tím, že je potřeba dopředu znát množinu všech transformací, které bude možné v rámci aplikace s 3D modelem vykonávat. Z této množiny se vybere N transformací, které jsou v rámci množiny rovnoměrně rozpoložené a dobře tím reprezentují daný prostor transformací. Z těchto N transformací se vybere jedna, pro kterou je manuálně vytvořena stylizace vykresleného obrázku s touto transformací. Vůči této konkrétní transformaci se pomocí StyLit spočítá NNF pro každou další z N transformací. Tím končí připravovací fáze. Při běhu v aplikaci se pro aktuální transformaci, kterou je potřeba vykreslit, najdou z N transformací, co byly zpracovány v rámci preporocesingu, ty nejbližší. Pro tyto vybrané transformace se poté vezmou předpřipravené NNF a zkombinují se dohromady podle toho, jak moc se každá z nich odlišuje od aktuální požadované transformace. Indexací výsledného NNF jsou pak překopírovány barvy z obrázku se vzorovým stylem na výstupní obrázek. Celý proces je znázorněn na obrázku 2.12.



Obrázek 2.12: Znázornění celého procesu syntézy z [15]. Nejprve manuální příprava stylizace, poté preprocesing NNF a nakonec samotné vykreslování.

Samotné fungování algoritmu StyLit je rozšířeno o zajištění dočasné koherence mezi snímky. To je důležité, aby byla změna mezi různými transformacemi modelu plynulá. Také je zde trochu jiné použití vodících kanálů. Jsou zde použity dvě varianty. První je prakticky totožná s originálním StyLit, kde byla přidávána informace o svícení ve scéně. Zde je navíc přidána informace o hranách 3D modelu. Druhá varianta je v práci použita pro modely 3D charakterů. Používá jen texturovanou barvu modelu a zobrazení identifikačních barev pro jednotlivé typy textur na modelu.

Nevýhodou je limitace v prostoru transformací, ve kterém lze s modelem pohybovat. Prostor transformací nemůže být moc velký, protože se potom do obrazu dostávají části modelu, které nejsou dostatečně zachycené v obrázku se vzorovým stylem.

2.2 Metody s využitím neuronových sítí

Je zde několik metod, jak k problému přenosu stylu pomocí neuronové sítě přistupovat. V historii se postupně vyvíjely a často jedna je vylepšením předchozí, nebo spojením vícero předchozích. Dále jsou tyto metody popsané postupně jak v čase vznikaly, až ke konci ta která je v rámci této práce použita.

2.2.1 Translační síť se ztrátovou funkcí pro každý pixel

Je asi nejjednodušším způsobem z hlediska implementace, jak použít neuronové sítě na přenos stylu. Je potřeba neuronová síť, která má na výstupu stejné dimenze, jako na vstupu (aproximuje transformaci). V případě obrázků, použitých jako vstup a výstup, stačí správně naučit parametry neuronové sítě tak, aby výstupní obrázek byl stylizovanou verzí vstupního.

Učení je realizováno pomocí sady obrázků a sady jejich variant s aplikovaným stylem. Potom stačí jako ztrátová funkce například euklidovská vzdálenost mezi vektory hodnot pixelů obrázků. Tento postup je sice funkční, ale má spoustu problémů. Je potřeba hodně příkladů dvojic obrázků pro správné naučení stylu. Vytvořit stylizovanou verzi může vyžadovat náročnou práci umělce. Proto je žádané počet vzorových obrázků naopak minimalizovat. Další problém je generalizace na jiné obrázky než ty v trénovací sadě. Tento způsob nemusí dávat moc dobré výsledky pro obrázky, které se liší od těch trénovacích.

Avšak tato metoda má stále dobrá praktická využití, jako jsou například zvětšování rozlišení obrázků [16], kolorizace [17] a segmentace [18].

2.2.2 Ztrátová funkce podle aktivací vrstev na klasifikační síti

Tento způsob, uveden v [19], využívá neuronové sítě určené ke klasifikaci obrázků. Například ke zjištění zda se v obrázku nachází, nebo nenachází člověk. Celkový výstup této sítě reprezentující klasifikaci je sice pro řešení úlohy přenosu stylu zbytečný, avšak podstatná informace se skrývá na výstupech mezi jednotlivými vrstvami této neuronové sítě. Pokud je síť již naučená na klasifikaci, pak tyto výstupy mezi vrstvami reprezentují nějakou obecnější informaci o datech se kterými pracuje.

Metoda [19] využívá těchto výstupů jednak na porovnávání významového obsahu obrázků a také na porovnání stylu obrázků. Obsah je reprezentován samotnými výstupy na vrstvách a styl korelační maticí mezi výstupy sousedních vrstev. Jak na obsah, tak na styl v [19] definovali ztrátovou funkci. Tyto dvě ztrátové funkce jsou spojeny do jedné lineární kombinací, kde násobení skaláry udává poměr mezi stylem a obsahem. Ve výsledku se pracuje s 3 obrázky. Jeden reprezentuje styl, druhý obrázek co se má stylizovat a třetí ze začátku inicializovaný náhodným šumem se iterativně optimalizuje, aby minimalizoval ztrátovou funkci. Tedy metoda nedává výsledek přímým vyhodnocením neuronové sítě, ale používá jí v rámci ztrátové funkce na optimalizaci hodnot obrázku.

Výhodou je nezávislost obrázku obsahující styl a obrázku, na který se má styl aplikovat, můžou být úplně odlišné. Nemusí se manuálně vytvářet stylizované verze původních obrázku pro trénování. Také lze snadno kontrolovat vliv stylu na výsledek, protože ztrátová funkce je lineární kombinací ztrátových funkcí obsahu a stylu.

Pomocí konstant, kterými jsou ztrátové funkce násobeny můžeme interpolovat mezi stylem a původním obrázkem.

Avšak jsou zde značné nevýhody. Jedna je čas vyhodnocení. Pro každou stylizaci obrázku se musí provádět náročný proces optimalizace. Jak již bylo zmíněno, může být výhodou nezávislost vstupního obrázku a obrázku se stylem, ale to zároveň dává menší kontrolu nad tím jak se ve výsledku styl aplikuje. Tím je myšlena například kontrola nad tím, na jakých místech, s jakou barvou a texturou se různé části stylu aplikují.

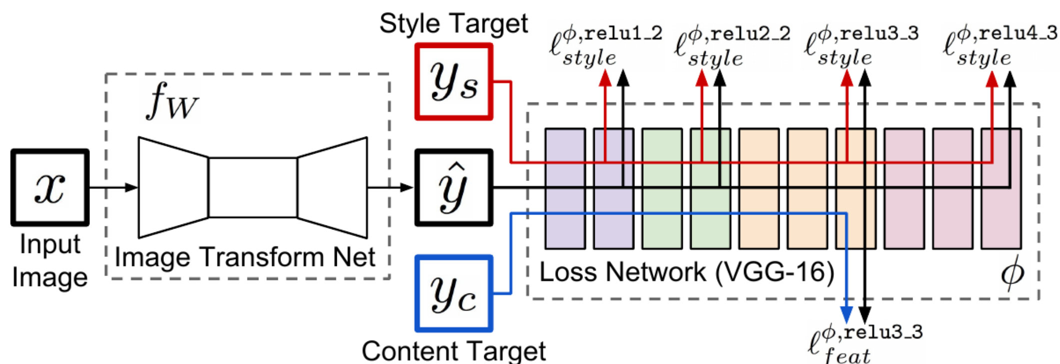
Obrázek 2.13: Demonstrace přenosu stylu z [19].

2.2.3 Kombinace ztrátových funkcí pro každý pixel a podle aktivace vrstev na klasifikační síti

Tato metoda je spojení dvou předchozích metod. Byla uvedena v [20]. Zjednodušeně se dá říci, že vezmeme vše z druhé metody (tedy klasifikační síť, ztrátové funkce a vstupní obrázky) a pouze obrázek, jehož hodnoty pixelů se postupně optimalizovaly, nahradíme výstupem z transformační neuronové sítě. Poté se místo optimalizace hodnot obrázku, optimalizují parametry této neuronové sítě. Ztrátová funkce je stále složená z lineární kombinace ztrátové funkce stylu a ztrátové funkce obsahu. Do této lineární kombinace se ještě přidá ztrátová funkce z první metody, která porovnává vstupní a výstupní obrázek po každém pixelu. To podle [20] zlepšuje výsledky.

Ve výsledku se spojí i výhody obou metod. Optimalizace se provádí pouze pro naučení stylu. K přenesení stylu stačí předat obrázek do naučené sítě. Zůstává ale

stále nevýhoda nízké kontroly nad tím, jakým způsobem se styl aplikuje.



Obrázek 2.14: Ilustrace modelu metody z [20]. Jako klasifikační síť použili konkrétně model VGG-16 natrénovaný na ImageNet datasetu.

2.2.4 Stylizace tváří

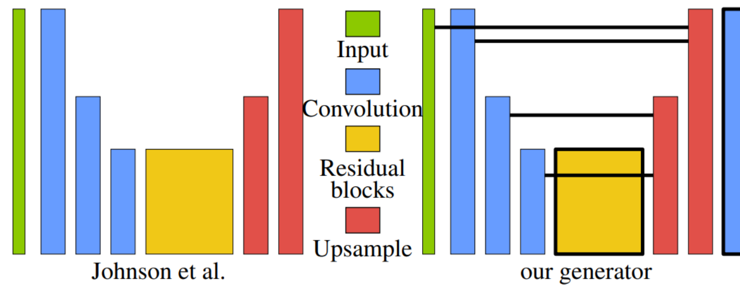
Významný průlom v přenosu stylu přinesla metoda [11], která je specializovaná na stylizace lidských obličejů (obrázek 2.15). Tato metoda staví na [20] (popsáno v předchozí podkapitole), ale trénování je zakomponováno do GAN frameworku [1] a data pro trénování jsou generována pomocí metody založené na použití záplat FaceStyle [9].

Obrázek 2.15: Demonstrace přenosu stylu na lidské tráře z [11].

Od [20] se v architektuře sítě liší především přidáním spojů mezi vrstvami na začátku a konci sítě (obrázek 2.16). To má za důsledek lepší propagaci gradientu při trénování. Ztrátová funkce je lineární kombinací ztrátové funkce pro každý pixel (color loss), ztrátové funkce podle aktivace vrstev na klasifikační síti VGG (perceptual loss) a ztrátové funkce GAN metody (adversarial loss).

Trénování pak probíhá tak, že předáme jeden portrét s příkladem stylu, a také velké množství vstupních nestylizovaných obrázků lidských obličejů. Metoda FaceStyle [9] poté pro každý vstupní obrázek vygeneruje stylizovanou variantu. Dvojce původních vstupních obrázků a obrázků vygenerovaných pomocí FaceStyle se nakonec předají do trénování neuronové sítě.

Jednou výhodou tohoto přístupu je dobrá generalizace díky použití GAN modelu. Metoda je tak odolnější proti některým chybným výsledkům metody FaceStyle, a proto má dokonce lepší výsledky než FaceStyle, z jehož generovaných příkladů se trénuje.



Obrázek 2.16: Změna architektury sítě mezi [20] a [11].

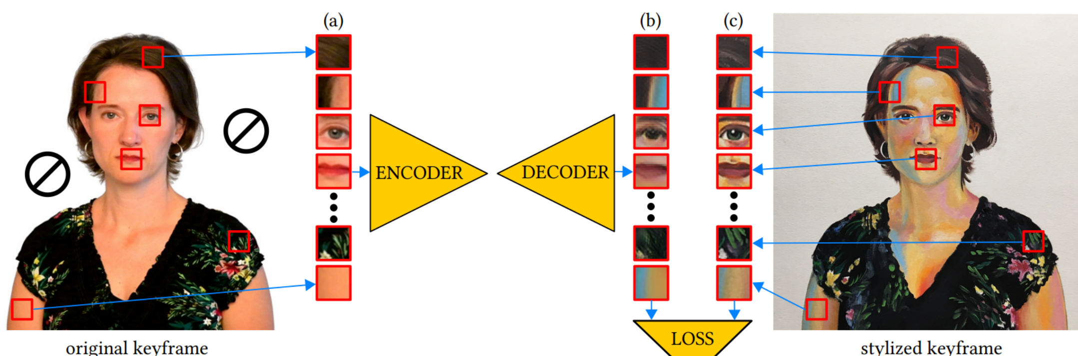
Oproti předchozím zmíněným metodám, které využívají pro přenos stylu neuronových sítí, umožňuje tato metoda mnohem větší kontrolu nad výsledkem přenesené stylizace. Metoda věrně zachovává část stylu pro sémanticky stejné části obrázku (vlasy, oči, nos, ústa).

Další výhodou je potřeba pouze jednoho vzoru stylu pro trénování. Naopak značná nevýhoda je v potřebě ohromného množství vstupních obrázků pro trénování. To znemožňuje (kromě závislosti na FaceStyle) přenositelnost této metody na jiné domény problémů, kde nemáme k dispozici takové množství vstupních vzorů, jako tomu je zde s obličeji.

Nevýhodou je dlouhý čas trénování, ale na druhou stranu je výhodou velice krátký čas jednoho vyhodnocení natrénované sítě oproti metodě FaceStyle. To umožňuje stylizaci obličeje v reálném čase z výstupu webkamery (za předpokladu že má uživatel dostatečně výkonnou grafickou kartu).

2.2.5 Stylizace založená na použití záplat

Problémy metody z předchozí podkapitoly [11] řeší metoda [21]. Využívá opět stejného modelu neuronové sítě. Ztrátová funkce pro trénování je složená stejným způsobem z lineární kombinace ztrátové funkce pro každý pixel (color loss), ztrátové funkce podle aktivace vrstev na klasifikační síti VGG (perceptual loss) a ztrátové funkce GAN metody (adversarial loss).



Obrázek 2.17: Znázornění trénovací strategie použité v [21].

Rozdíl je v datech, která jsou použita pro trénování. Zde je metoda inspirovaná metodami založenými na použití záplat. Při trénování se neuronová síť nevyhodnocuje na celém obrázku oproti [11], ale na jeho menší části - tzv. záplatě (anglicky patch, typicky se jedná o desítky pixelů). Ve ztrátové funkci se pak výsledek porovnává se záplatou z obrázku se stylizovanou verzí na stejné pozici jako vstupní

záplata ve vstupním obrázku. Celé trénování tak probíhá postupně vyhodnocováním na náhodně vybraných záplatách, vybíraných ze zdrojových obrázků (obrázek 2.17). Vyhodnocení natrénované sítě pak probíhá na celém obrázku. To je možné díky tomu, že síť je plně konvoluční a můžeme na vstup vložit obrázek s libovolným rozlišením. I přesto, že trénování probíhalo na menších částech, vyhodnocení celého obrázku dává správné výsledky.

Metoda už není závislá na generování vstupních dat pomocí FaceStyle. Proto také není omezena na stylizaci lidských obličejů, ale lze jí aplikovat prakticky i na jakýkoli jiný problém přenosu stylu. Avšak je zde nutno na rozdíl od FaceStyle mít pro každý vstupní obrázek jeho stylizovanou variantu, protože každá dvojce záplat sdílí stejnou pozici v obou obrázcích.

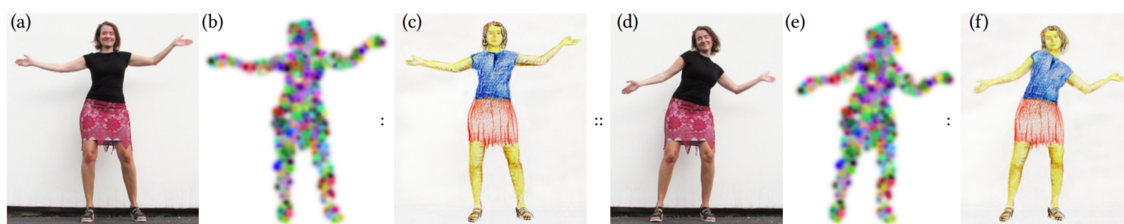
Tento přístup vyřeší hned několik problémů. Z jedné dvojce vzorových obrázků se tímto způsobem vygeneruje velké množství trénovacích dat. Díky tomu není nutné velké množství vzorových dvojic obrázků. Ve velké části případů stačí dokonce jedna dvojce obrázků pro trénování.

Jelikož architektura modelu neuronové sítě je zhruba stejná, tak vyhodnocení natrénované sítě zůstáva rychlé podobně jako u předchozí metody. Celkové trénování je značně rychlejší než [11]. To umožňuje dokonce interaktivní vytváření stylizace obrázku tím způsobem, že umělec v průběhu vytváření stylizované varianty vzorového obrázku vidí průběžné výsledky z neuronové sítě, která se souběžně s jeho prací trénuje na již odvedené práci.

Tato metoda dokonce ještě lépe generalizuje. Tím že trénování probíhá na menších lokálních částech, není tolik závislá na globální struktuře obrázku. Je díky tomu vhodná například na stylizaci celého videa s použitím pouze malého množství stylizovaných klíčových snímků pro trénování.

Na druhou stranu nezávislost na globální struktuře obrázku může vést na nejednoznačné přiřazení více stylizovaných záplat k jedné stejné vstupní záplatě. To znamená, že malé kousky, co na vstupním obrázku vypadají stejně, jsou ve stylizované verzi obrázku stylizované jinak. Důsledkem toho je špatné přiřazení těchto záplat u natrénované sítě.

Avšak tento problém lze vyřešit přidáním další informace o každém pixelu obrázku mimo jeho barvu, která tyto nejednoznačné záplaty se stejnou barvou rozliší. To je podobný přístup jako u metod založených na použití záplat jako jsou například StyLit [8], FaceStyle [9] a EbSynth [12]. Zde se pro příklad videa používá náhodné rozmístění tzv. gauss disků (obrázek 2.18). Jedná se o kruhy s fixním rádiusem a minimální vzdáleností od sebe. Každý má náhodně přiřazenou barvu, jejíž intenzita klesá od středu podle gaussovy křivky. Pro zbytek sekvence videa se pak tyto kruhy pohybují podle pohybu obsaženém ve videu (to lze pomocí předpočítaného optického toku ve videu).



Obrázek 2.18: Odstranění nejednoznačnosti (b) a (e) v [21].

Díky zmíněným vlastnostem, jako je rychlé vyhodnocení a dobrá generalizace, byla tato metoda zvolena pro řešení problematiky přenostu stylu na 3D model v reálném čase, kterou se tato práce zabývá.

Kapitola 3

Implementace

V této kapitole je obsažen popis aplikace vykonávající přenos stylu na 3D model v reálném čase, která je výstupem tohoto projektu. Hlavní funkce aplikace jsou: zobrazování 3D modelu, zobrazování varianty 3D modelu s aplikovaným stylem s dostatečnou časovou odezvou a změna parametrů scény jako jsou pozice kamery, nebo vlastnosti světla a vlastnosti materiálu modelu. Vedle toho jsou zde pomocné skripty, buď na zpracování dat, nebo trénování modelu neuronové sítě. Aplikace je psaná primárně v jazyce C++. Je zde pár částí v jazyce Cuda C++. Pomocné skripty jsou napsané v Pythonu. V následujících podkapitolách je rozebrána struktura aplikace a fungování jejích jednotlivých částí. Zdrojový kód projektu je v příloze B.

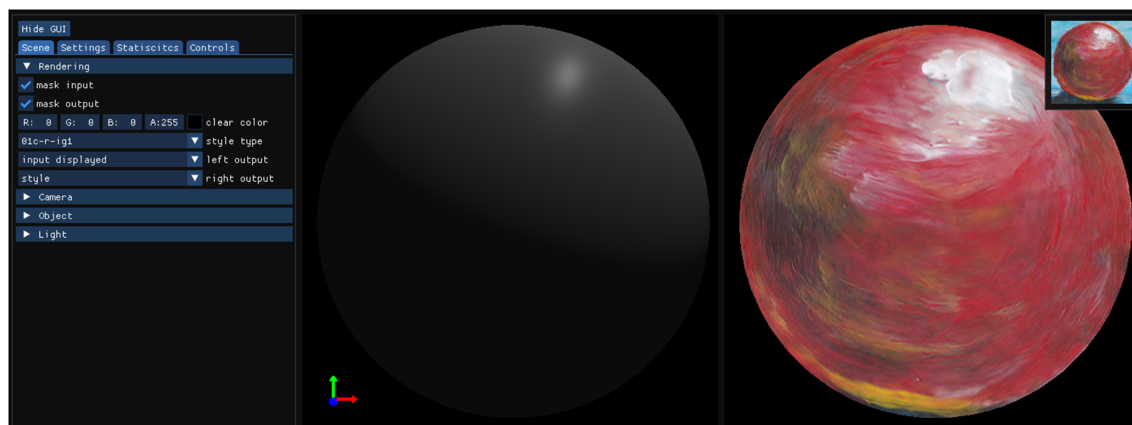
3.1 Struktura projektu

Projekt lze rozdělit na několik hlavních částí. První je uživatelské rozhraní řídící stav a chování aplikace podle vstupu uživatele. Dále kód, co vykresluje scénu a zobrazuje jí společně s výstupem z neuronové sítě v okně. A nakonec kód vyhodnocující neuronovou síť, pro který byly vytvořeny dvě různé implementace. Také je třeba zmínit řadu knihoven, použitých pro různé činnosti v různých částech projektu.

Sestavení projektu do binární podoby je organizováno pomocí systému CMake. Jeho výhodou je multiplatformovost, flexibilita v jeho nastavení a to, že je používán ve většině C++ projektů. Projekt je sice psán z většiny multiplatformním způsobem, ale jeho sestavení bylo testováno a je funkční pouze na operačním systému GNU/Linux. Funkční sestavení pro Windows tedy zatím nefunguje.

3.2 Uživatelské rozhraní

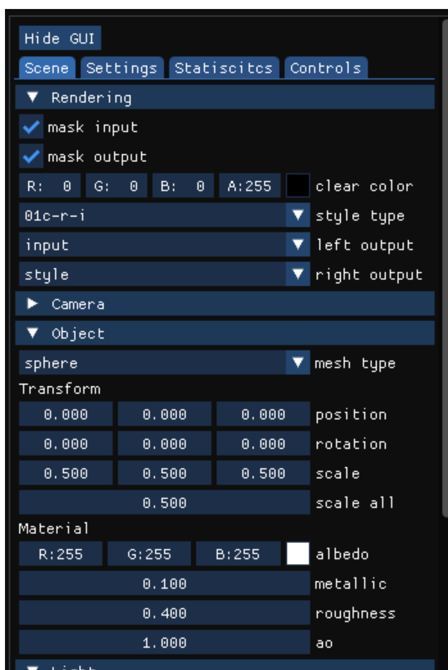
Obsah okna aplikace je rozdělen na 3 části. V levé části okna je GUI sloužící pro ovládání a nastavení aplikace. V uprostřed okna je vykreslován 3D model. V pravé části je zobrazena jeho stylizovaná varianta, vyhodnocená neuronovou sítí. V této části je v pravém horním rohu zobrazen vzorový styl, který je aplikován. Okno aplikace tedy může vypadat, tak jak je zobrazeno na obrázku 3.1. V tomto konkrétním případě je použit model koule a neuronová síť je naučená na daný vzhled malby.



Obrázek 3.1: Vzhled aplikace vykonávající přenos stylu.

Vstupem z klávesnice může uživatel měnit pozici a natočení kamery, pomocí které je model transformován do výsledného obrazu. V levém horním rohu je vždy tlačítko, kterým lze zobrazovat a schovávat ImGui okénko 3.2, ve kterém lze podrobně nastavovat vlastnosti scény a chování aplikace. ImGui je knihovna pro snadné vytváření uživatelského rozhraní, vhodná pro použití v aplikaci vykreslující s nízkoúrovňovým grafickým rozhraním. To je v tomto případě konkrétně OpenGL. Toto bude více rozebráno v podkapitole o vykreslování. Jsou zde také klávesy pro pořizování screenshotů textur používaných aplikací. Mezi hlavní patří vykreslený model s aplikováním stylu nebo bez aplikování stylu. Dále lze získat snímek skrytých obrázků pro speciální použití, jako jsou například znázornění normál modelu, nebo směru ke světlu od povrchu modelu. Konkrétní klávesy pro jednotlivé akce a očekávané chování je specifikováno v manuálu v příloze A.

V ImGui okně lze nastavovat parametry scény, jako jsou pozice a rotace kamery, vlastnosti světla, vlastnosti materiálu modelu a typ zobrazovaného modelu. Dále jsou zde možnosti pro ovlivnění četnosti vyhodnocení přenosu stylu zobrazovaného v pravé části okna. V případě dlouhého vyhodnocení lze přenos stylu dočasně úplně vypnout. Také je tu možnost takzvaného líného vyhodnocování, která je v základním nastavení zapnutá. Líné vyhodnocování nevykonává přenos stylu pro každý vykreslovaný snímek, ale pouze v případě kdy uživatel přestane měnit parametry scény (konkrétně například pohybovat s kamerou) a zároveň pokud se scéna nemění vůbec, pak také nic nevyhodnocuje. Tento způsob vyhodnocování umožňuje rozumnou ovladatelnost i pro zařízení s méně výkonnou grafickou kartou. Podrobnější chování celého ImGui okna je opět definováno v manuálu v příloze A.



Obrázek 3.2: Okénko s podrobným nastavením aplikace.

3.3 Vykreslování

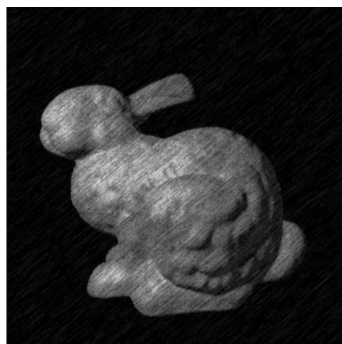
Aplikace využívá k vykreslování knihovnu OpenGL. Alternativou by bylo použít buď jiné nízkoúrovňové grafické rozhraní, nebo framework či engine s vyšší abstrakcí. Pro použití do této práce se více hodí programování grafiky na nízké úrovni, protože umožňuje plnou kontrolu nad chováním vykreslovacího řetězce. Nevýhodou je vyšší složitost implementace a s tím spojená náchylnost pro vznik chyb. Z těchto důvodů byla použita grafická knihovna Morph Engine kterou vytvořil autor této práce. Podrobněji o této knihovně je napsáno v podkapitole o knihovnách využívaných v rámci projektu.

Způsob vykreslování s aplikací stylu je následující. Nejprve se klasickým způsobem pomocí OpenGL vertex a fragment shaderů vykreslí scéna. Díky již zmíněné vysoké kontrole nad vykreslovacím řetězcem ji můžeme vykreslit do textury místo rovnou do okna aplikace. Tato textura se předá neuronové síti, ta jako výsledek vrátí opět texturu. Původní textura scény a textura po aplikaci neuronové sítě se nakonec vykreslí vedle sebe do okna aplikace tak jak bylo znázorněno na obrázku 3.1.

3.4 Trénování neuronové sítě

Pro natrénování neuronové sítě je použita PyTorch implementace metody [21], která je volně dostupná v github repositáři (odkaz v příloze D). Toto je asi nejlepší volba způsobu trénování, protože projekt je robustní a jde snadno konfigurovat. Dělat vlastní implementaci, by zabralo velké množství času. Na 3.3, 3.4 a 3.5 je zkouška trénování neuronové sítě na efekt dojmu kresby tužkou. Obrázek 3.5 vpravo je výstupem sítě, která je naučená po 1500 epochách se základním nastavením hyperparametrů.

Obrázek 3.3: Vstupní obrázek.



Obrázek 3.4: Vzorový obrázek s původní stylizací.

Obrázek 3.5: Výstup z naučené sítě.

3.5 Aplikace neuronové sítě

V rámci projetu byly vytvořeny dvě implementace pro vyhodnocení modelu neuronové sítě v aplikaci. První využívá PyTorch C++ API. Druhá je složená z kompletně vlastního kódu s využitím OpenGL a GLSL. Projekt aktuálně přednostně používá první variantu. Druhá varianta byla z důvodů uvedených v její příslušné kapitole opuštěna.

3.5.1 Pomocí PyTorch C++ API

Tato implementace využívá pro vyhodnocení neuronové sítě populárního frameworku PyTorch. Za normálních okolností je tento framework většinou používán přes jeho Python rozhraní. Pod tímto rozhraním se však skrývá vysoce optimalizovaná implementace napsaná v jazyce C++. Její efektivita je zajištěná buď skrze paralelizaci a využití SIMD pro vyhodnocení na CPU, nebo Cuda implementaci v knihovně cuDNN pro vyhodnocení na GPU.

Kromě Python rozhraní existuje také C++ rozhraní, které je přímo napojené na zmíněnou interní implementaci. Rozhraní má dvě varianty. Jednu s pevně daným vyhodnocením na CPU, druhou s možností vyhodnocení na GPU pomocí Cuda implementace. Pro tento projekt je použito konkrétně C++ rozhraní s Cuda implementací.

Instantní výhodou použití této varianty je stejný programovací jazyk jako ve zbytku aplikace. Další výhodou je efektivita propojení s OpenGL implementací. Tedy konkrétně předání OpenGL dat pro vyhodnocení neuronové sítě. To je možné pomocí Cuda-OpenGL interop rozhraní, které umožňuje označit OpenGL buffer pro

použití v Cuda kódu. Díky tomu nemusí docházet k náročným přesunům dat mezi operační pamětí a pamětí GPU, které by bylo nutné v případě, kdy bychom tuto možnost neměli. Celkově je výhodou vysoce optimalizovaná implementace.

Zároveň je framework na vyhodnocení totožný s tím, co je použit na trénování. Proto není potřeba konvertovat uložený soubor neuronové sítě do kompletně jiného formátu, případně implementovat jeho vlastní parsování. Avšak je potřeba po trénování uložit model neuronové sítě do speciální varianty formátu, která ukládá informaci o struktuře modelu, a ne jenom hodnoty jeho parametrů. Za normálních okolností při práci v Pythonu máme strukturu modelu k dispozici, protože je obsažená ve zdrojovém kódu, se kterým pracujeme. Tuto informaci o struktuře ale musíme doplnit, při přechodu do jazyka C++. Řešením by bylo přepsání zdrojového kódu modelu z Pythonu do C++. To je ale pracné a nepraktické, v případě změny původní implementace v Pythonu. Proto je to vyřešeno uvedeným způsobem s přídavnou informací o struktuře modelu v souboru.

Z tohoto přístupu by mohly benefitovat i některé další projekty využívající neuronových sítí. Většina těchto projektů je psána právě například v jazyce Python pomocí frameworku PyTorch. Distribuce výsledných aplikací může být pro uživatele, kteří nemají zkušenosti s programováním, velice nepřívětivá. A to konkrétně z důvodu nutnosti nainstalovat Python a knihovny, na kterých je daný projekt závislý, nebo netradiční způsob spouštění těchto programů, na které není běžný uživatel zvyklý. Oproti tomu v řešení implementace pomocí C++ s použitím PyTorch C++ API lze projekt zkompilovat do binární EXE podoby s přiloženými DLL knihovnami. Uživatel si pak program pouze stáhne a tradičním způsobem ho spustí. Tedy tento přístup značně zjednodušuje distribuci pro uživatele aplikace.

3.5.2 Vlastní implementace s OpenGL a GLSL

V této implementaci je celé vyhodnocení neuronové sítě vytvořeno vlastním kódem. Základní struktura je velice podobná tomu, jak jsou neuronové sítě implementovány ve frameworku PyTorch. To konkrétně v tom smyslu, že každý typ vrstvy neuronové sítě (např konvoluce, batch normalizace, ReLU) má v implementaci vlastní modul který jí vyhodnocuje. Celá neuronová síť lze potom jednoduše složit z několika libovolných vrstev.

Pro tento projekt je implementováno několik typů vrstev neuronové sítě které jsou nutné pro vyhodnocení používaných modelů. Mezi ně patří například 2D konvoluce, batch normlalizace, ReLU a upsampling. Vyhodnocení každé vrstvy je implementováno v GLSL compute shaderu. Ten pak má svého protějška v kódu aplikace formou C++ třídy, která pomocí OpenGL tento shader spustí a nastaví mu vstupní a výstupní buffery.

Konkrétní model neuronové sítě je při spuštění programu sestaven z jednotlivých vrstev podle specifikace v souboru modelu neuronové sítě. Soubor modelu neuronové sítě je uložen a načítán pomocí knihovny Protocol Buffer, která zajišťuje přenositelnost formátu mezi různými programovacími jazyky. Tímto způsobem to bylo uděláno z nutnosti přenést model mezi Pythonem a C++ a jinými implementacemi neuronové sítě v příslušných jazycích.

Při samotném vyhodnocování se poté spouštějí vrstvy tak, jak jdou po sobě a vždy přijímají na vstup některé konkrétní výstupy předchozích vrstev. Na vstup a výstup neuronové sítě jako celku jsou jednoduše předány textury, které jsou použity ve vykreslovací pipeline. Ve vstupní textuře je vykreslená scéna. Výstupní textura

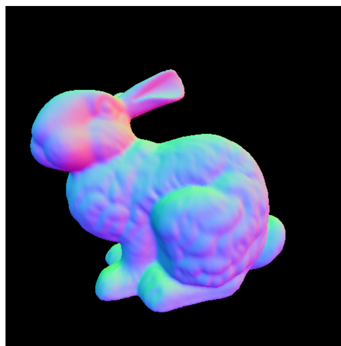
je pak použita pro zobrazení do okna.

Od této varianty vyhodnocení neuronové sítě bylo v průběhu práce upuštěno. Někde v implementaci se nachází numerická chyba, která vede na chybné výsledky neuronové sítě. Autor strávil hledáním chyby několik týdnů, a proto z časových důvodů tento přístup zanechal ve prospěch implementace pomocí PyTorch C++ API. Ale i přesto měla tato část implementace zajímavé některé výsledky a vlastnosti, které by bylo dobré zmínit.

Na rozdíl od PyTorch C++ API s Cuda implementací není toto řešení závislé na použití Nvidia grafických karet. Je možné použít třeba AMD nebo Intel. Další výhoda je snadné propojení s vykreslovacím kódem, protože je obojí psáno s OpenGL. Na druhou stranu je tato implementace v průměru zhruba 20× pomalejší než PyTorch implementace (ta využívá vysoce optimalizovanou knihovnu cuDNN). Je zde však několik způsobů jak dále optimalizovat. Jedním je zlepšit efektivitu přístupů do paměti. Toho lze dosáhnout lepším cachováním vstupů, nebo nalezením optimální velikosti groupy compute shaderu, nebo použitím optimálního rozložení vstupních dat v paměti.

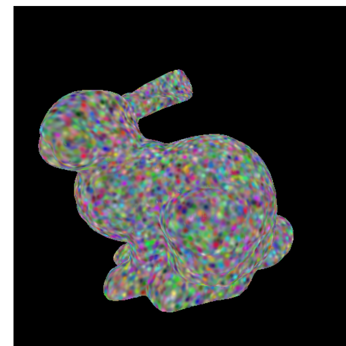
3.6 Přidání pomocných kanálů

Metoda přenosu stylu využitá v této aplikaci nabízí možnost rozšířit barvy vstupního obrázku o další pomocné kanály, které mohou pomoci s odstraněním nejednoznačnosti přiřazovaných záplat. Proto aplikace kromě vykreslování stíněného modelu, také vykresluje vizualizaci jeho normálových vektorů, vizualizaci směru ke světlu nebo manuálně zvolenou texturu (obrázky 3.6, 3.7, 3.8).



Obrázek 3.6: Normálové vektory.

Obrázek 3.7: Směr ke světlu.



Obrázek 3.8: Vlastní textura.

Implementace v aplikaci umožňuje předávat libovolné množství přídavných kanálů na vstup neuronové sítě. Avšak v konkrétní situaci, kdy je neuronová síť natrénovaná na konkrétní počet kanálů, aplikace podle nastavené konfigurace dovolí předat je tyto specifikované kanály.

3.7 Celý proces vyhodnocení a zobrazení

Nejprve jsou vykresleny všechny obrázky, potřebné pro vstup neuronové sítě. Tedy stínovaný model a k tomu volitelně navíc další přídavné kanály. Také je vykreslena maska modelu. To je textura, která do obrázku model vyplní bílou barvou a všechno ostatní černou. Výsledné obrázky se poté složí do jednoho tensoru, který se poté předá ke zpracování neuronové síti aplikující stylizaci. Výstupní tensor je převeden do OpenGL textury. Na tu se poté volitelně může aplikovat maska. Tím se zachová pouze část výsledného obrázku, kde je obsažen model. Zbytek se vyplní specifikovanou barvou pozadí. Výsledný obrázek se vykreslí do okna aplikace.

3.8 Knihovny

Důležitou součástí projektu je grafická knihovna Morph Engine (odkaz v příloze C), na které autor pracuje mimo rozsah této práce. Tato knihovna je v projektu použitá pro veškerý vykreslovací kód. Její účel je abstrahovat a zjednodušit práci s nízkoúrovňovými grafickými rozhraními při zachování jejich flexibility. Zároveň má poskytnout bezpečné rozhraní které pomocí dobře definovaných datový typů zabrání vzniku nekonzistentních stavů. Těchto požadavků je dosaženo především pomocí konceptů programovacího jazyka C++ jako je RAII a template programování, nebo konceptu sum type převzatého z funkcionálních programovacích jazyků. Dvě nejdůležitější rozhraní, které knihovna abstrahuje jsou: grafické rozhraní OpenGL a rozhraní GLFW pro práci se systémovými okny a získávání vstupu od uživatele (například klávesnice a myš).

Pro vyhodnocení neuronové sítě je použito PyTorch C++ API s Cuda implementací. Jedná se asi o nejjednodušší způsob jak vyhodnocovat model neuronové sítě trénovaný pomocí PyTorch v aplikaci napsané v C++. Díky možnosti Cuda rozhraní registrovat OpenGL data, je možné efektivně propojit vstup a výstup neuronové sítě s OpenGL vykreslovací pipeline.

Je zde použitá celá řada dalších knihoven. Většinou slouží pro zjednodušení nějaké specifické činnosti. Například spdlog pro logování, GoogleTest pro unit testy, ImGui pro snadné GUI, stb pro čtení a zápis obrázků z disku nebo json pro ukládání konfiguračních souborů. Celý list použitých knihoven je v příloze E.

Kapitola 4

Výsledky

Nyní se podíváme na možnosti a výstupy toho, co bylo vytvořeno v rámci implementace. Nejprve si projdeme způsoby trénování modelu neuronové sítě, kterou pak bude aplikace využívat. Poté si pomocí vyhodnocení neuronové sítě v rámci aplikace rozebereme, jak dobře reaguje daná neuronová síť na změny transformací našeho modelu. Nakonec zhodnotíme schopnost aplikace reagovat dostatečně interaktivně. Prozkoumáme, co je na výpočtu nejvíce náročné a navrhneme možná vylepšení.

4.1 Trénování

Zde se budeme zabývat způsobem trénování neuronové sítě pro přenos stylu na 3D model. Nejprve natrénujeme neuronou síť pro přenos stylu na kouli. Poté zobecníme tento postup pro libovolný 3D model. Podíváme se na 2 způsoby. Jeden s využitím již natrénované sítě podle předlohy stylizované koule. Druhý trénováním podle předlohy vytvořené přímo pro daný 3D model. Všechny následující experimenty při trénování vycházejí z vzoru stylu na obrázku 4.1. Model neuronové sítě pro trénování byl použit z D. Je použito základní nastavení, které má počty konvolučních filtrů na jednotlivých vrstvách: 32, 64, 128, 128, 128 a 64. Počet resnet bloků je nastaven na pouze jeden blok.

Obrázek 4.1: Vzorový obrázek se stylizací.

4.1.1 Přenos stylu na kouli

Cílem je natrénovat neuronovou síť, která je schopná ze vstupu na obrázku 4.2 vytvořit obrázek podobný vzoru stylu 4.1.

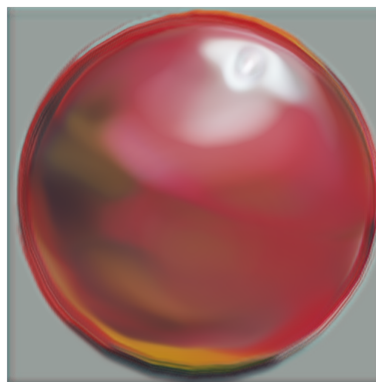
Obrázek 4.2: Vstupní stíněná koule.

Při prvním pokusu natrénovat neuronovou síť použijeme pouze obrázek stínované koule jako vstup. Výstup takto natrénované sítě na obrázku 4.3 je daleko od požadovaného výsledku. Prakticky se barva celé koule změnila na průměrnou barvu cíleného stylu. Důvodem je vysoká nejednodznačnost ve vstupním obrázku 4.2. Přechod barev je v něm velice hladký a v rámci obrázku můžeme najít mnoho malých lokálních kousků, které jsou prakticky totožné. To je charakteristika metody kterou požíváme, protože je založená na zpracování menších částí nezávisle na celku. Proto musíme přidat kanály s přídavnou informací abychom se této nejednoznačnosti zbavili.

Obrázek 4.3: Přenos stylu natrénovaný pouze se vstupem stínovaného modelu.

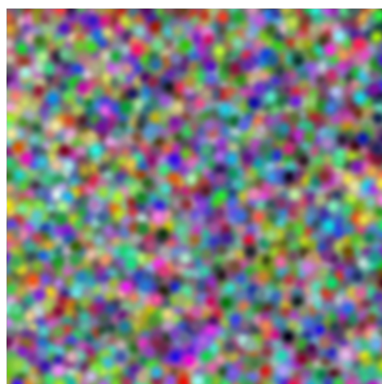
Jako první zkusíme přidat informaci o normálových vektorech 4.4. Výsledek na obrázku 4.5 už zachytil více informace ze vzorového stylu. Jsou zde vidět některé části z původního vzoru. Avšak úplné detaily se ve výsledku ztratily a stala se z toho prakticky rozostřená varianta vzorového stylu. Normálové vektory tedy samotné nestačí na dosažení požadovaného výsledku. Opět vytváří problém to, že přechod mezi barvami je příliš pozvolný. Lze tak rozlišit malé kousky, co jsou daleko od sebe, ale pokud jsou moc blízko, pak je nedovede neuronová síť odlišit a naučí se aplikovat průměr jejich požadovaných mapování na styl. Podobný efekt nastane při použití informace o světlu.

Obrázek 4.4: Vstupní normály.

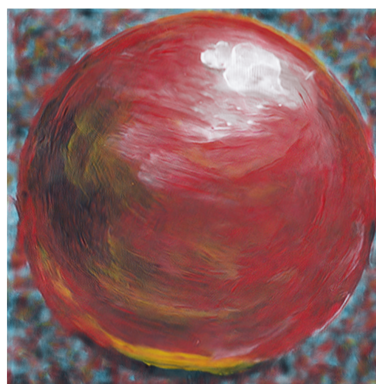


Obrázek 4.5: Natrénováno pomocí normál.

Pro další pokus vyzkoušíme způsob, který byl v práci [21] použit na odstranění nejednoznačnosti (obrázek 2.18). Použijeme tedy obrázek rovnoměrně rozpoložených gauss disků 4.6, jako přídavnou informaci pro trénování. Výsledný obrázek 4.7 už zachycuje všechny detaily cíleného stylu.



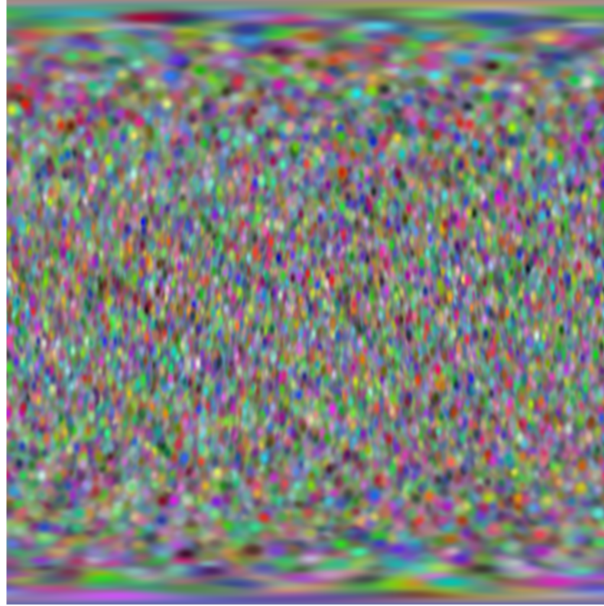
Obrázek 4.6: Rovnoměrně rozpoložené gauss disky.



Obrázek 4.7: Výsledek sítě natrénované s pomocí gauss disků.

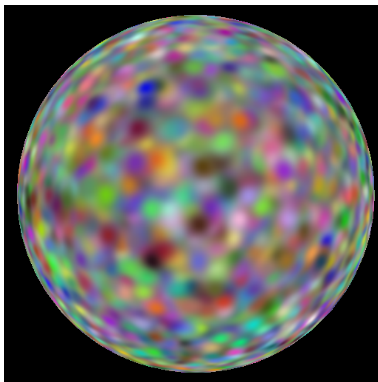
Avšak obrázek 4.6 nemá smysl používat v obecném případě, protože je použitelný jen na tuto konkrétní transformaci koule ve scéně. Cílem je abychom mohli neuronovou síť aplikovat na jinou transformaci modelu a aby výsledek vypadal jako vzorový styl transformovaný stejným způsobem.

Tento problém můžeme vyřešit tím, že místo rozpoložení gauss disků rovnoměrně v rovině obrázku je rozpoložíme rovnoměrně na povrchu koule. To lze docílit tím, že vytvoříme 3D variantu gauss disků na obrázku 4.6 a prořízneme jí povrchem koule. Získané hodnoty pak můžeme pomocí sférických souřadnic namapovat do 2D textury 4.8.



Obrázek 4.8: Rovnoměrně rozpoložené barvy gauss disků na povrchu koule uložené ve sférických souřadnicích do 2D textury.

S pomocí této vygenerované textury potom můžeme vykreslit gauss disky na povrchu koule do obrázku jako přídavnou informaci pro neuronovou síť (obrázek 4.9). Výsledek 4.10 je totožný s předchozím výsledkem 4.7 a opět zachovává všechny detaily požadovaného stylu. Ale na rozdíl od předchozího řešení můžeme přídavnou informaci gauss disků transformovat.



Obrázek 4.9: Koule vykreslená pomocí textury gauss disků.

Obrázek 4.10: Výsledek sítě natrénované s pomocí textury gauss disků.

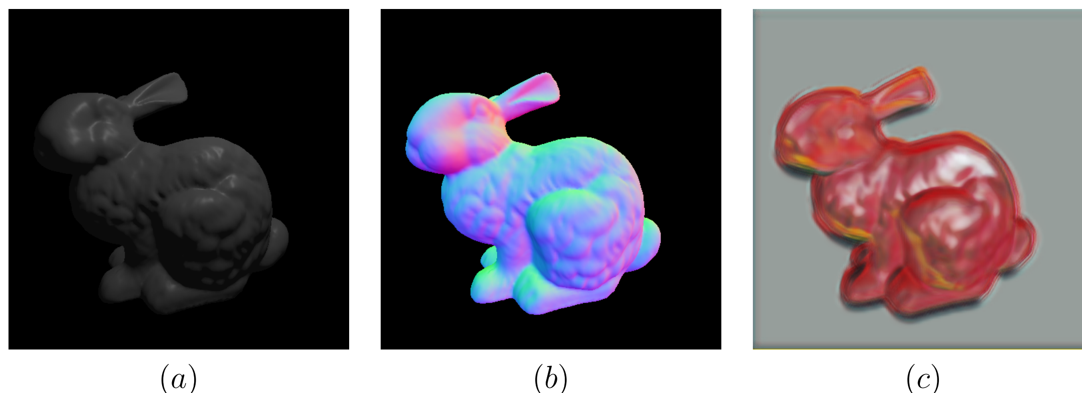
4.1.2 Trénování na kouli použito na složitější model

Zajímavým řešením, jak realizovat přenos stylu na složitější model, by bylo podobně jako v metodě StyLit [8] použít model neuronové sítě, který je natrénovaný na přenos stylu na kouli. Bohužel nejlépe fungující přístup z předchozí sekce, který využívá přídavné informace ve formě gauss disků, není přenositelný mezi různými 3D modely.

Bylo by možné provést například environmentální mapování z koule na daný 3D model, ale to vytváří značné deformace, které způsobí kompletně jiný charakter

rozpoležení výsledných barev na povrchu 3D modelu, tak že neuronová sít nedovede identifikovat malé lokální kousky stejně jako u koule.

Přenositelnou možností je například varianta využívající jako přídavnou informaci normálové vektory modelu. To si ukážeme na příkladu 4.11. Obrázky (*a*) a (*b*) vložíme na vstup neuronové sítě natrénované na kouli. Výsledek (*c*) rozumně zachovává styl podle směru normál. Avšak tím že samotná natrénovaná síť nedávám moc dobré výsledky už u koule, kde jsou ztraceny detaily, tak i zde dochází k stejnému efektu.



(*a*) (*b*) (*c*)

Obrázek 4.11: Výstup neuronové sítě (*c*) naučené pomocí normálových vektorů koule použité na vstup stínovaného modelu králíka (*a*) a jeho normálových vektorů (*b*).

4.1.3 Přenos stylu na složitější model

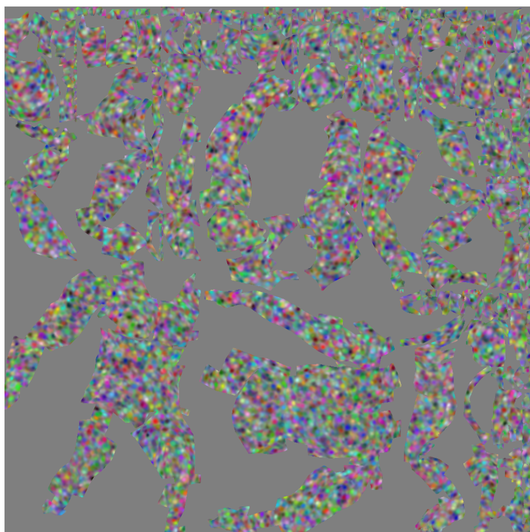
Pokud máme vzor stylu pro daný model, můžeme použít podobný přístup pomocí přídavné informace gauss disků jako u koule. Protože vzor pro model nemáme k dispozici, tak si ho pro tento příklad vygenerujeme pomocí veřejně dostupné implementace EbSynth. V této implementaci má uživatel možnost si navolit vlastní kanály s přídavnou informací. Lze tak například docílit stejného fungování jako u metod StyLit a FaceStyle. Pro tento případ použijeme informaci o normálových vektorech a směru ke světlu. Výsledek pro modely golema a králíka je na obrázcích 4.12 a 4.13.

Obrázek 4.12: Vzorový styl golema vytvořený pomocí EbSynth.

Obrázek 4.13: Vzorový styl králíka vytvořený pomocí EbSynth.

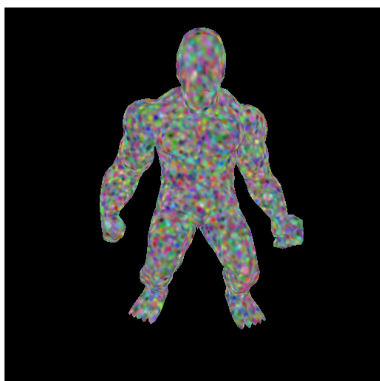
Textury gauss disků musíme vygenerovat znovu pro konkrétní modely. Jak již bylo zmíněno, byla by zde možnost použít enviromentálního mapování z koule, ale to

vytvoří nežádoucí deformace, proto to tímto způsobem dělat nebudeme. Generování je provedeno v principu stejným způsobem jako u koule. Vytvoříme prostor 3D gauss disků a prořízneme ho povrchem daného modelu a namapujeme získané hodnoty do 2D textury pomocí UV souřadnic uložených v modelu. Příklad produkované textury pro golema je na obrázku 4.14.

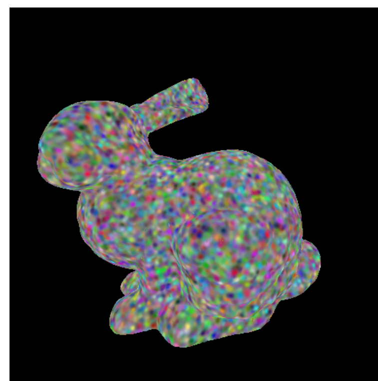


Obrázek 4.14: Textura gauss disků pro model golema.

Hlavní rozdíl oproti kouli tedy je, že místo sférických souřadnic použijeme předem dané UV souřadnice modelu. Toto je vlastně obecnější způsob a je možné ho použít i na kouli, za předpokladu že u ní máme k dispozici UV souřadnice. Tímto způsobem jsou gauss disky na povrchu modelu rovnoměrně rozprostřeny. Výsledné aplikace textur jsou na obrázcích 4.15 a 4.16.

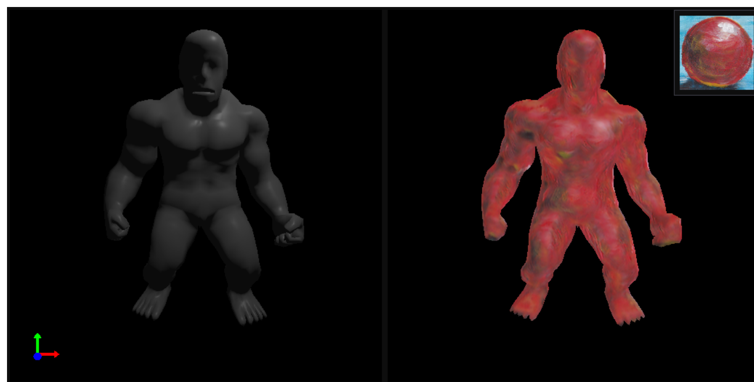


Obrázek 4.15: Aplikovaná textura gauss disků vytvořená pro model golema.

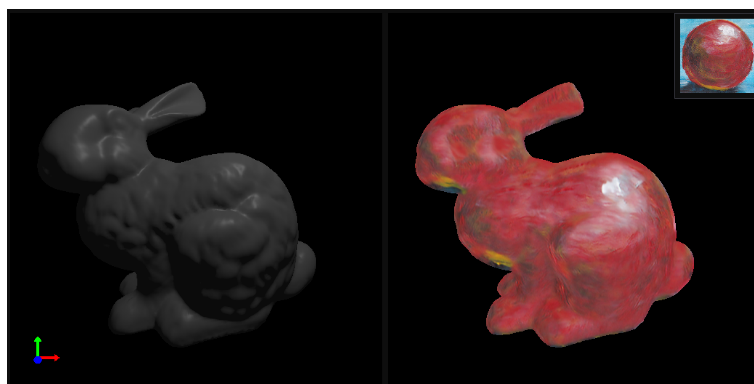


Obrázek 4.16: Aplikovaná textura gauss disků vytvořená pro model králíka.

Z těchto dat natrénovaná neuronová síť produkuje v aplikaci pro model golema a model králíky výstupy 4.17 a 4.18.



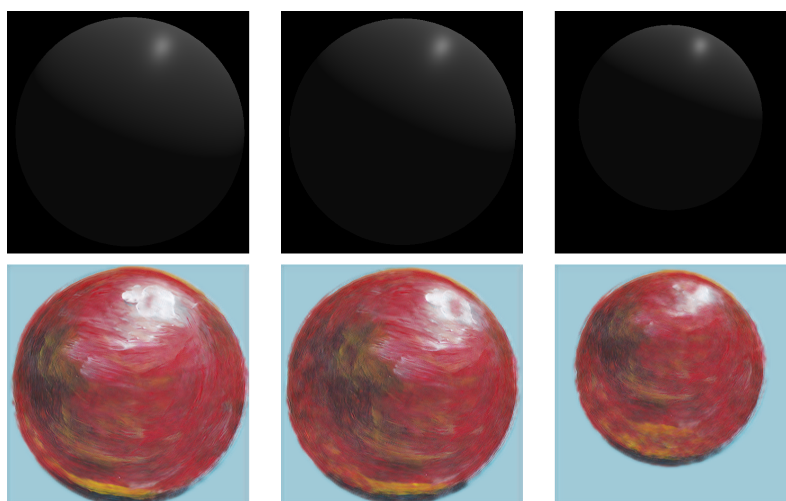
Obrázek 4.17: Výsledek stylizace vedené texturou gauss disků pro model golema.



Obrázek 4.18: Výsledek stylizace vedené texturou gauss disků pro model králíka.

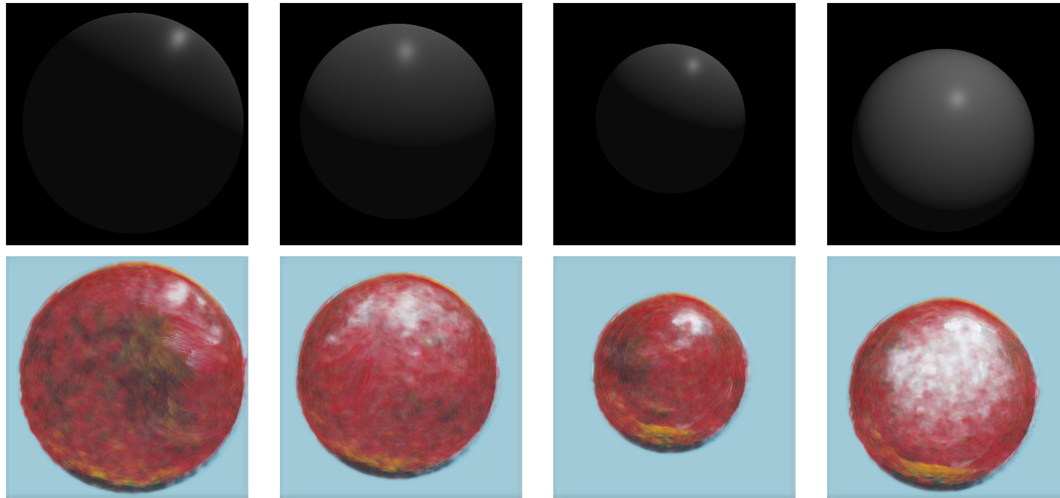
4.2 Vliv transformace 3D modelu

Díky způsobu, jakým jsme vyřešili kanály přídavné informace, můžeme nyní vyzkoušet aplikovat různé transformace našeho modelu a pozorovat výsledky přenosu stylu, který je naučený na jedné konkrétní transformaci. Pro malé odchylky od původní transformace se styl aplikuje korektním způsobem, tak jak bychom očekávali. Transformace stínovaného modelu způsobí totožnou transformaci stylu. Příklady těchto transformací jsou na obrázcích 4.19.



Obrázek 4.19: Transformace koule korektně zachovávající požadovaný styl.

Pro transformace příliš vzdálené od té původní, na které se trénovala neuronová síť, jsou výsledky podstatně horší (obrázky 4.20). Možnou příčinou by mohlo být jednak například to, že při rotaci se odkryje část modelu, která nebyla v exempláři použitém pro trénování. Druhou možnou příčinou je fakt, že pokud otáčíme s nějakou ploškou (konkrétně polygony, ze kterých se model skládá), textura na ní se smršťuje nebo roztahuje. To způsobuje, že textura na krajích koule je ve vykresleném obrázku více smrštěná oproti středu. Důsledkem toho jsou další nesrovnalosti oproti tomu na co je neuronová síť naučená.



Obrázek 4.20: Transformace koule, které nezachovávají požadovaný styl.

Tento problém by bylo možné vyřešit přidáním více pohledů, ze kterých se neuronová síť učí. Hypoteticky by se neuronová síť naučila mezi těmito pohledy přecházet, a tím by se nám zvětšil prostor transformací, ve kterém se můžeme pohybovat. Je to trochu podobné principu metody StyleProp, kde se výsledek skládá pomocí NNF předpočítaných z několika různých pohledů. V tomto případě je ale problém, že pro každou transformaci vybranou pro trénování potřebujeme stylizovanou variantu. Opět je možnost jich vícero vygenerovat, například pomocí jedné z metod založených na použití záplat. To ale také nemusí vést na dobré výsledky, pokud je transformace příliš odlišná od té původní.

4.3 Interaktivita

Nyní se zaměříme na efektivitu přenosu stylu v aplikaci z hlediska času výpočtu. Ta přímo určuje míru interaktivity, protože čím kratší je celý výpočet, tím více snímků můžeme vykreslit. Zásadní vliv na výkon v našem případě má grafická karta. To z toho důvodu, že vyhodnocení neuronové sítě probíhá pomocí Cuda implementace na GPU a veškeré vykreslování je realizováno pomocí OpenGL, také na GPU. Proto uvedeme, že všechna měření v této sekci probíhala na počítači s grafickou kartou NVidia GeForce GTX 950M.

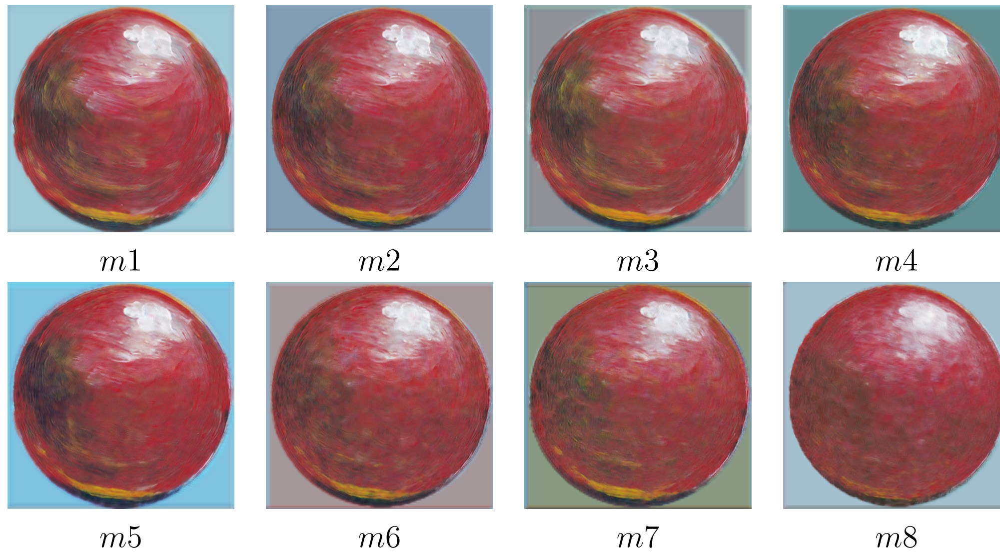
Pro základní model neuronové sítě, použitém v předchozí sekci o trénování, je čas získání jednoho snímku v průměru 519 ms, tedy zhruba 2 FPS. To je absolutně nedostačující pro interaktivní odezvu, u které bychom chtěli ideálně 60 FPS (16.7 ms), ale bylo by možné se spokojit i s 30 FPS (33.3 ms). Potřebujeme tedy identifikovat části výpočtu, které tvoří většinu výpočetního času, a zefektivnit je. Podrobným měřením jednotlivých částí zjistíme, že prakticky skoro celý čas výpočtu zabírá samotné vyhodnocení neuronové sítě. Ostatní výpočty, jako je například všechno vykreslování OpenGL, nebo konverze OpenGL dat do PyTorch tensorů pro neuronovou síť a obráceně, zabírají čas v rámci jednotek milisekund. To je oproti 500 milisekundám prakticky zanedbatelné.

Vyhodnocení neuronové sítě vykonává knihovna PyTorch pomocí Cuda knihovny cuDNN. Implementace této knihovny je vysoce optimalizovaná. Naprogramovat efektivnější řešení by proto bylo velice složité a nejspíš by nepomohlo značným způsobem. Avšak náročnost vyhodnocení neuronové sítě je daná tím, že je tato neuronová síť velká. Konkrétně má hodně vrstev, které provádí konvoluce s velkým počtem konvolučních filtrů. Konvoluce jsou výpočetně nejnáročnější operace, co síť vykonává. Proto se zaměříme na redukci počtu konvolučních filtrů, kterému je výpočetní náročnost konvoluce přímo úměrná. Na tabulce 4.1 je znázorněno několik redukovaných modelů neuronové sítě s jejich parametry a průměrnými časy vyhodnocení. Model $m1$ je základní model, který byl doposud používán.

model	$l1$	$l2$	$l3$	$l4$	$l5$	$l6$	počet parametrů	čas (ms)	FPS
$m1$	32	64	128	128	128	64	1 420 995	519	1.9
$m2$	32	64	128	128	128	64	978 115	480	2.1
$m3$	16	32	64	64	64	32	359 267	165	6.1
$m4$	16	32	64	64	64	32	248 419	156	6.4
$m5$	16	32	32	32	32	32	152 067	98	10.2
$m6$	16	32	32	32	32	32	124 291	96	10.4
$m7$	16	32	32	32	32	32	85 283	65	15.4
$m8$	16	16	16	16	16	16	24 387	40	25.0

Tabulka 4.1: Parametry a časy redukovaných modelů neuronové sítě $m1, \ldots, m8$. Ve sloupcích $l1, \ldots, l6$ jsou počty konvoluční filtrů na jednotlivých vrstvách neuronové sítě. Dále je zde počet parametrů, který naznačuje složitost daného modelu. Poté průměrný čas vyhodnocení (v milisekundách, zaokrouhleno na celá čísla) jednoho obrázku s rozlišením 512×512. Nakonec je zde průměrný počet snímků za sekundu - FPS (zaokrouhleno na desetiny), který je spočítaný z času vyhodnocení. K tomu navíc modely $m1$, $m3$ a $m5$ mají mezi vrstvami $l3$ a $l4$ jeden resnet blok, ostatní modely ho nemají. Modely $m7$ a $m8$ neobsahují spojení mezi vrstvami napříč sítí vyznačeno v 2.16.

Je vidět, že výpočet lze takto významně zrychlit. Ovšem redukce počtu konvolučních filtrů zhoršuje vlastnosti neuronové sítě a tím má negativní dopad na kvalitu výstupů. Na obrázcích 4.21 jsou výstupy pro jednotlivé redukované modely. Do redukce $m5$ se zdají být výsledky odpovídající správnému výstupu. Poté se kvalita začne rychle ztrácet. Proto také nebyla uvedena větší redukce než $m8$, která už není schopná správně zachytit žádné detaily, ale pouze základní barvy a jejich šumivý charakter.



Obrázek 4.21: Výsledky přenosu stylu pro redukované modely neuronových sítí.

Tedy nejlepší výkon při zachování rozumné kvality, kterého jsme byli schopni dosáhnout na grafické kartě GTX 950M, je zhruba 10 FPS. Grafická karta GTX 950M je oproti dnešním modelům grafických karet výrazně méně výkonná. Proto by hypoteticky bylo možné dosáhnout úplně interaktivní odezvy na novější grafické kartě. S jinou grafickou kartou toto měření v rámci práce nebylo provedeno.

V práci [21] byli autoři schopni s neredukovanou verzí modelu neuronové sítě použité v této práci na grafické kartě NVidia RTX 2080 vyhodnocovat obrázky rozlišení 512×512 při 17 FPS. Pokud by tedy platilo, že redukce na model $m5$ zrychlí výpočet zhruba pětkrát (tak jak tomu bylo v našem případě), pak by bylo možné na grafické kartě NVidia RTX 2080 dosáhnout 85 FPS. V dané práci je sice použita jiná implementace, která je napsaná v Pythonu, ale PyTorch interně volá stejné knihovny, které vykonávají většinu výpočtu, proto by časy neměly být příliš odlišné.

Avšak pro grafickou kartu GTX 950M a jí výkonem podobné, případně dokonce méně výkonné, není vše ztraceno. Pomocí takzvaného líného vyhodnocování, které je popsáno v kapitole o implementaci, lze s modelem interaktivně pohybovat a výsledky stylizace vidíme průběžně. Není to sice úplná interaktivita, při které se vyhodnotí každý snímek, ale alespoň toto funguje jako částečné řešení problému v případě že nemáme na výběr.

Závěr

V rámci této práce jsme si nejprve uvedli základní pojmy o umělých neuronových sítích, abychom dále lepe porozuměli jejich aplikaci v metodách přenosu stylu. Poté jsme prošli různé metody, kterými lze přistupovat k problematice přenosu stylu. Konkrétně se jednalo o dvě hlavní kategorie: metody založené na použití záplat a metody využívající neuronových sítí. Podle prozkoumaných vlastností metod přenosu stylu byla jedna zvolena pro použití do implementace řešící situaci přenosu stylu na 3D model. Konkrétně byla vybrána metoda kombinující obě uvedené kategorie přenosu stylu [21].

Dále byla vytvořena aplikace, která na 3D model pomocí předem natrénované neuronové sítě přenáší styl. V aplikaci může uživatel s modelem pohybovat nebo případně měnit další parametry scény a při tom může sledovat změny toho, jak se styl pomocí neuronové sítě aplikuje v různých situacích.

Nakonec jsme zhodnotili výsledky vytvořené implementace. Bylo dosaženo natrénovaných neuronových sítí, které věrně přenáší vzorový styl na 3D modely. Pomocí možnosti pohybovat s modelem v aplikaci a sledovat vyhodnocení stylizace jsme zjistili, že pro transformace, které jsou blízko transformaci ve vzorovém stylu, ze kterého se učí neuronová síť, se styl aplikuje korektním způsobem. Naopak pro transformace příliš daleko od té vzorové jsou výsledky chybné. Možným řešením tohoto problému do budoucna by mohlo být použití několika vzorových exemplářů s různými transformacemi pro trénování neuronové sítě.

Tato práce je také zaměřená na to, aby byl přenos stylu vyhodnocován v reálném čase. To si můžeme představit tak, že budeme v aplikaci pohybovat s 3D modelem a při každém snímku uvidíme jeho stylizovanou verzi. Hypoteticky bylo tohoto cíle dosaženo za předpokladu, že uživatel má dostatečně výkonnou grafickou kartu (například NVidia RTX 2080). Pro méně výkonnou grafickou kartu (NVidia GTX 950M), se kterou byl tento projekt vyvíjen, lze dosáhnout jen částečné interaktivity, protože čas vyhodnocení není dost krátký na to, aby se výsledek zobrazoval každý snímek. V této situaci je proto tento problém zjemněn pomocí takzvaného líného vyhodnocování.

Pro další případné pokračování v práci na této problematice je zde mnoho prostoru na zlepšení. Hlavně z hlediska kvality výsledné stylizace a flexibility jejího použití. Například by bylo vhodné prozkoumat další možnosti realizace kanálů s pomocnou informací pro trénování neuronové sítě.

Bibliografie

1. GOODFELLOW, Ian; POUGET-ABADIE, Jean; MIRZA, Mehdi; XU, Bing; WARDE-FARLEY, David; OZAIR, Sherjil; COURVILLE, Aaron; BENGIO, Yoshua. Generative Adversarial Nets. In: *Proceedings of the International Conference on Neural Information Processing Systems*. 2014, s. 2672–2680.
2. KARRAS, Tero; LAINE, Samuli; AILA, Timo. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2019.
3. TEXLER, Aneta; TEXLER, Ondřej; KUČERA, Michal; CHAI, Menglei; SÝKORA, Daniel. FaceBlit: Instant Real-time Example-based Style Transfer to Facial Videos. *Proceedings of the ACM in Computer Graphics and Interactive Techniques*. 2021, roč. 4, č. 1, s. 14.
4. GATYS, Leon A.; ECKER, Alexander S.; BETHGE, Matthias. *A Neural Algorithm of Artistic Style*. 2015.
5. HERTZMANN, A.; JACOBS, C. E.; OLIVER, N.; CURLESS, B.; SALESIN, D. H. *Image analogies*. 2001.
6. BARNES, C.; SHECHTMAN, E.; FINKELSTEIN, A.; GOLDMAN, D. B. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics*. 2009, roč. 28, č. 3.
7. BARNES, C.; SHECHTMAN, E.; GOLDMAN, D. B.; FINKELSTEIN, A. The generalized PatchMatch correspondence algorithm. *In Proceedings of European Conference on Computer Vision*. 2010, s. 29–43.
8. FIŠER, Jakub; JAMRIŠKA, Ondřej; LUKÁČ, Michal; SHECHTMAN, Eli; ASENTE, Paul; LU, Jingwan; SÝKORA, Daniel. StyLit: Illumination-Guided Example-Based Stylization of 3D Renderings. *ACM Transactions on Graphics*. 2016, roč. 35, č. 4.
9. FIŠER, Jakub; JAMRIŠKA, Ondřej; SIMONS, David; SHECHTMAN, Eli; LU, Jingwan; ASENTE, Paul; LUKÁČ, Michal; SÝKORA, Daniel. Example-Based Synthesis of Stylized Facial Animations. *ACM Transactions on Graphics*. 2017, roč. 36, č. 4.
10. SELIM, Ahmed; ELGHARIB, Mohamed; DOYLE, Linda. Painting Style Transfer for Head Portraits Using Convolutional Neural Networks. *ACM Transactions on Graphics*. 2016.
11. FUTSCHIK, David; CHAI, Menglei; CAO, Chen; MA, Chongyang; STOLIAR, Aleksei; KOROLEV, Sergey; TULYAKOV, Sergey; KUČERA, Michal; SÝKORA, Daniel. Real-Time Patch-Based Stylization of Portraits Using Generative Adversarial Network. In: *Proceedings of the 8th ACM/EG Expressive Symposium*. 2019, s. 33–42.

12. JAMRIŠKA, Ondřej; SOCHOROVÁ, Šárka; TEXLER, Ondřej; LUKÁČ, Michal; FIŠER, Jakub; LU, Jingwan; SHECHTMAN, Eli; SÝKORA, Daniel. Stylizing Video by Example. *ACM Transactions on Graphics*. 2019, roč. 38, č. 4.
13. SÝKORA, Daniel; JAMRIŠKA, Ondřej; TEXLER, Ondřej; FIŠER, Jakub; LUKÁČ, Michal; LU, Jingwan; SHECHTMAN, Eli. StyleBlit: Fast Example-Based Stylization with Local Guidance. *Computer Graphics Forum*. 2019, roč. 38, č. 2, s. 83–91.
14. SLOAN, P.-P. J.; MARTIN, W.; GOOCH, A.; GOOCH, B. The Lit Sphere: A model for capturing NPR shading from art. *In Proceedings of Graphics Interface*. 2001, s. 143–150.
15. HAUPTFLEISCH, Filip; TEXLER, Ondřej; TEXLER, Aneta; KŘIVÁNEK, Jaroslav; SÝKORA, Daniel. StyleProp: Real-time Example-based Stylization of 3D Models. *Computer Graphics Forum*. 2020, roč. 39, č. 7, s. 575–586.
16. DONG, Chao; LOY, Chen Change; HE, Kaiming; TANG, Xiaoou. *Image Super-Resolution Using Deep Convolutional Networks*. 2015.
17. CHENG, Zezhou; YANG, Q.; SHENG, B. Deep Colorization. *IEEE International Conference on Computer Vision (ICCV)*. 2015, s. 415–423.
18. LONG, Jonathan; SHELHAMER, Evan; DARRELL, Trevor. *Fully Convolutional Networks for Semantic Segmentation*. 2015.
19. GATYS, Leon A.; ECKER, Alexander S.; BETHGE, Matthias. Image Style Transfer Using Convolutional Neural Networks. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, s. 2414–2423.
20. JOHNSON, Justin; ALAHI, Alexandre; FEI-FEI, Li. *In Proceedings of European Conference on Computer Vision*. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. 2016.
21. TEXLER, Ondřej; FUTSCHIK, David; KUČERA, Michal; JAMRIŠKA, Ondřej; SOCHOROVÁ, Šárka; CHAI, Menglei; TULYAKOV, Sergey; SÝKORA, Daniel. Interactive Video Stylization Using Few-Shot Patch-Based Training. *ACM Transactions on Graphics*. 2020, roč. 39, č. 4, s. 73.

Přílohy

A Manuál k aplikaci

Klávesa	Funkce
W	pohyb dopředu
S	pohyb dozadu
A	pohyb vlevo
D	pohyb vpravo
Mezerník	pohyb nahoru
Levý shift	pohyb dolu
Q nebo šipka doleva	rotace vlevo
E nebo šipka doprava	rotace vpravo
Šipka nahoru	rotace nahoru
Šipka dolu	rotace dolu
G	zobrazit/schovat GUI
L	zapnutí líného vykreslování
U	vynucení vykreslení
Točení kolečka myši	zoom kamery
Držení levého tlačítka myši	posun kamery v rovině okna
Držení kolečka myši	rotace kamery

Tabulka 2: Tabulka kláves a jejich funkcí pro ovládání aplikace.

GUI aplikace se skládá ze 4 záložek. První záložka Scene obsahuje nastavení objektů ve scéně, a také způsob jakým se scéna vykresluje na obrazovku. Z hlediska nastavení objektů ve scéně lze nastavit parametry a transformaci kamery, transformaci vykreslovaného objektu, a také vlastnosti jeho materiálu a parametry světla. Ze samotného vykreslování lze nastavit aplikace masky na vykreslený výstup, barva pozadí a model neuronové sítě který má aplikovat styl

Druhá záložka Settings obsahuje pokročilejší nastavení chování aplikace. Lze zde přepínat mezi módem vykreslování dvou obrázků vedle sebe, nebo vykreslováním pouze jednoho obrázku. Je zde nastavení četnosti vyhodnocení neuronové sítě. To lze buď úplně vypnout, nebo nechat počítat každý snímek, nebo vyhodnocení takzvaným líným způsobem. Vyhodnocení líným způsobem znamená, že neuronová síť aplikuje styl pouze v situaci kdy uživatel přestane měnit parametry scény, a to buď v gui nebo pohybem pomocí kláves. Také jsou zde tlačítka pro ukládání na disk snímků vykreslovaných obrázků, nebo ukládání aktuální konfigurace aplikace, nebo scény.

V třetí záložce Statistics jsou informace o časech provedení některých částí procesu vykreslování. Je zde také celkový čas využití procesoru za jeden snímek a z toho spočítané FPS.

Čtvrtá záložka Controls obsahuje seznam kláves, kterými lze aplikaci ovládat, a jejich popis. Tyto klávesy jsou uvedeny v tabulce 2.

B Zdrojový kód projektu

Je přiložený k práci. Je také možné ho získat z git repositáře:
`https://gitlab.com/shetr/realtime_style_transfer_3d`
`https://gitlab.com/shetr/realtime_style_transfer_3d_python_scripts`

C Vlastní grafický engine

`https://gitlab.com/shetr/morphengine`

D Interactive Video Stylization Using Few-Shot Patch-Based Training

`https://github.com/OndrejTexler/Few-Shot-Patch-Based-Training`

E Použité knihovny

PyTorch C++ API	`https://pytorch.org/cppdocs/`
Glad	`https://glad.dav1d.de/`
GLFW	`https://github.com/glfw/glfw`
GLM	`https://github.com/g-truc/glm`
GoogleTest	`https://github.com/google/googletest`
fmt	`https://github.com/fmtlib/fmt`
spdlog	`https://github.com/gabime/spdlog`
stb	`https://github.com/nothings/stb`
ImGui	`https://github.com/ocornut/imgui`
nameof	`https://github.com/Neargye/nameof`
magic enum	`https://github.com/Neargye/magic_enum`
Protocol Buffers	`https://github.com/protocolbuffers/protobuf`
json	`https://github.com/nlohmann/json`
Poisson Disk Sampling	`https://github.com/thinks/poisson-disk-sampling`