FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Example-based Style Transfer to 3D Models using Mobile Device
Student:	Bc. Marek Alexa
Supervisor:	prof. Ing. Daniel Sýkora, Ph.D.
Study program:	Informatics
Branch / specialization:	Web and Software Engineering, specialization Software Engineering
Department:	Department of Software Engineering
Validity:	until the end of winter semester 2022/2023

Instructions

Explore current state-of-the-art in the field of automatic transfer of artistic style to 3D models [1, 2, 3, 4]. Implement the StyleBlit algorithm [3], which allows transferring artistic style from a physical template to a 3D model in real-time. To capture the style, use the mobile device's rear camera, which captures the template with the drawing, performs rectification, and transfers the style to the selected 3D model. The stylized model will appear on the display of the mobile device, where the user will rotate it. Verify the developed algorithm's functionality on various 3D models and artistic styles, which will be provided by the thesis supervisor.

–

[1] Bénard et al.: Stylizing Animation by Example, ACM Transactions on Graphics 32(4):119, 2013.

[2] Fišer et al.: StyLit: Illumination-guided Example-based Stylization of 3D Renderings, ACM Transactions on Graphics 35(4):92, 2016.

[3] Sýkora et al.: StyleBlit: Fast Example-Based Stylization with Local Guidance, Computer Graphics Forum 38(2):83–91, 2019.

[4] Hauptfleisch et al.: StyleProp: Real-time Example-based Stylization of 3D Models, Computer Graphics Forum 39(7), 2020.

Electronically approved by Ing. Michal Valenta, Ph.D. on 15 February 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Example-based Style Transfer to 3D Models using Mobile Device

Bc. Marek Alexa

Department of Software Engineering
Supervisor: prof. Ing. Daniel Sýkora, Ph.D.

January 6, 2022

Acknowledgements

First and foremost, I would like to wholeheartedly thank my supervisor for his general support and helpful remarks. Another thanks goes to my friends and family for moral support, in particular Lubomír, Tadeáš, and Štěpán.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on January 6, 2022 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Alexa, Marek. *Example-based Style Transfer to 3D Models using Mobile Device*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022. Also available from: ⟨`https://gitlab.fit.cvut.cz/alexama1/styletransfer`⟩.

Abstrakt

Tato práce představuje unikátní kombinaci zážitku z rozšířené reality, počítačově podporované stylizace s akvizicí vzorového stylu a interakce s 3D modelem. Práce nastiňuje přehled problému stylizace a různé přístupy k němu, přičemž některé z těchto metod jsou prozkoumány detailněji. Následně je vysvětlena teorie za rozšířenou realitou a skenování stylu. Nakonec jsou uvedeny detaily implementace. Výsledkem je aplikace s rozšířenou realitou pro OS Android, která je schopna stylizace a získávání stylu v reálném čase.

Klíčová slova Rozšířená Realita, Stylizace, 3D Modely, Unity, Android

Abstract

This thesis presents a unique combination of the augmented reality experience, computer-assisted stylization with style exemplar acquisition, and 3D model interaction. An overview of the stylization problem and various approaches is outlined, while a few of these methods are explored more in-depth. Subsequently, the theory behind augmented reality and style scanning is explained. Finally, the implementation details are provided with the result being an augmented reality application for the Android OS, capable of stylization and style acquisition in real-time.

Keywords Augmented Reality, Stylization, 3D Models, Unity, Android

Contents

List of Figures

List of Tables

Introduction

Art, specifically painting, is often regarded as one of the most fascinating manifestations of the human spirit. The play of colors, light, and darkness has enthralled humankind for millennia. A stylized view of the real world can help emphasize the author's feelings.

Until recently, art has been the exclusive domain of mankind. With the advent of computing technology, a question arose if there is a possibility that a machine could mimic the art produced by humans. There have been efforts to simulate specific painting techniques since the 1980s [30]. While these algorithms have impressive results, the images are easily recognizable as artificial.

The next step came with a method called Image Analogies [7], which suggested using the style as one of the inputs. Nothing brings the feeling of awe as seeing your creation replicated in a completely different scenario. As the years progressed, so did the techniques. Nowadays, we can achieve authentic-looking stylization of images, 3D models, or even videos with some techniques even being able to operate in real-time. However, there is always a compromise between the quality of the stylization and performance, with the goal being to achieve both.

Recently, there has been an effort to bring the world of stylization closer to ordinary people. May it be with face filters [23], or through an augmented reality experience [31]. This thesis joins this effort as it brings a previously unrealized amalgamation of style exemplar acquisition and interactive manipulation of the result, all within an augmented reality experience. The ingredients used in this thesis are depicted in figure 1.

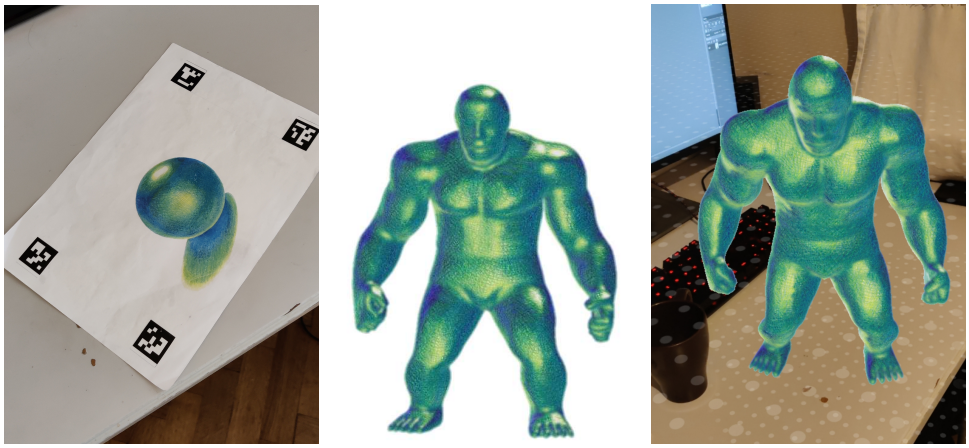


Figure 1: What this thesis tries to achieve: acquire a style input (left)), apply it to a 3D model (middle) [1], and show the result in augmented reality(right).

Structure

The field of computer-assisted stylization is outlined in the first chapter. The second chapter describes everything that is required to understand the inner workings of the project, with the project itself being covered in the third chapter. Finally, the last chapter summarizes the project and indicates possible improvements.

Chapter 1

State-of-the-art

1.1 Pattern-based methods

Paul Haeberli [2], was the first to explore computer-assisted stylization, which is about trying to convert an image into a stylized artistic painting. There are multiple approaches [32] to this problem. One of the more popular is using a set patterns like pen contours [33, 34], hatch [35] or brush strokes [3, 36, 37] to stylize either 2D or 3D inputs. These patterns are placed repeatedly on the output image in order to produce a stylized version of the input with their combinations. The placement of the patterns can be guided either locally [2] or globally [4]. The user can generally provide their set of patterns, allowing for customization. However, this approach can output only a limited number of style variations, as it is bounded by the predefined algorithm and the set of patterns unless the algorithm and patterns are hand-crafted for every target style. A few examples from this branch of stylization are displayed in figure 1.1.

Figure 1.1: Stylized examples from Haeberli [2], Litwinowicz [3], and Hertzmann [4].

1.2 Filter-based methods

Filter-based methods refer to a process that composes various filtering and processing of the input image. These methods are commonly found in photo editing software. A simple example (while not strictly speaking a stylization) of a filter is the Gaussian blur [38]. A number of methods in this category perform edge detection by the difference of Gaussians method [39], for example, XDoG [5] uses it to produce various stylizations. Another example of this approach is trying to mimic pencil drawing by combining the tone and stroke structures [6]. See figure 1.2 for a visualization.

Most of the filter-based methods can be adapted to run parallelly or even on the GPU, making them suitable for real-time usage.

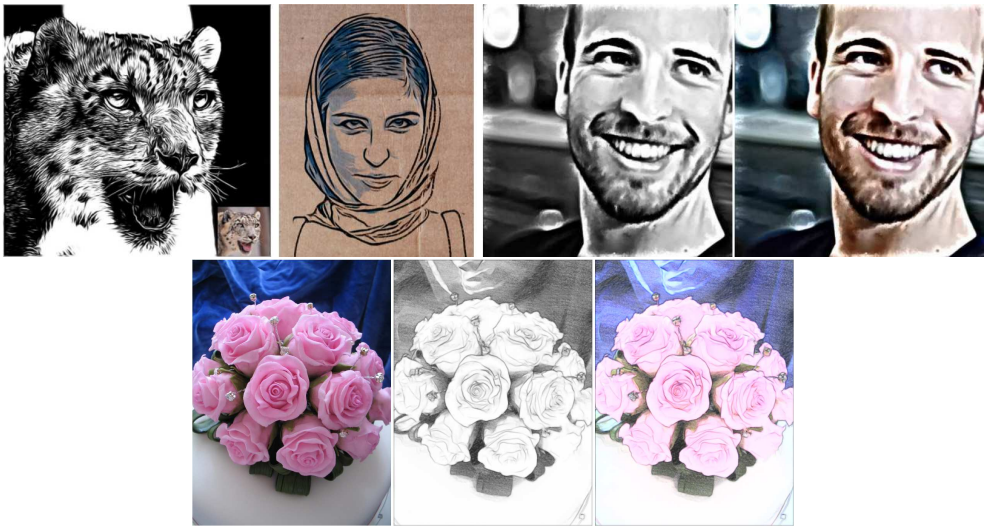


Figure 1.2: Stylized example outputs from XDoG [5] (first row) and an example with source image by Lu [6] (second row).

1.3 Patch-based methods

1.3.1 Image Analogies

Image Analogies [7] introduces a new approach to stylization, as it generalizes the stylization problem by making the desired style one of the inputs of the algorithm. The exemplar input consists of two images – unfiltered and filtered. The approach is patch-based with guidance based on color. It matches pixels from the target image to the unfiltered input and copies the look from the filtered input. Due to its patch-based nature and reliance on color, this method can produce visible seams and cannot yield correct illumination.

There are two stages: a design phase, where a pair of training images is provided, and an application phase, in which the learned relation (filter) is

applied to a new image in order to create an analogous result. The method is based on multi-scale autoregression. An output example from the original article [7] can be seen in figure 1.3.

Figure 1.3: Image Analogies output example [7].

To understand Image Analogies more in-depth, a description of the used algorithm follows.

```
1  function CreateImageAnalogy(A, A', B):
2    Compute Gaussian pyramids for A, A', and B
3    Compute features for A, A', and B
4    Initialize the search structures (e.g., for ANN)
5    foreach level l, from coarsest to finest, do:
6      foreach pixel q∈B'_l, in scan-line order, do:
7        p ← BestMatch(A, A', B, B', s, l, q)
8        B'_l(q) ← A'_l(p)
9        s_l(q) ← p
10   return B'_L
```

Where A and A' are the mentioned pair of training images (original and filtered respectively) and B is the target image (to be stylized).

The BestMatch function, shown in listing 1.3.1, compares the approximate nearest neighbor search (ANN) with the best coherence match. For its breakdown, please see the following pseudocode:

```
1  function BestMatch(A, A', B, B', s, l, q)
2    p_app ← BestApproximateMatch(A, A', B, B', l, q)
3    p_coh ← BestCoherenceMatch(A, A', B, B', s, l, q)
4    d_app ← ||F_l(p_app) − F_l(q)||^2
5    d_coh ← ||F_l(p_coh) − F_l(q)||^2
6    if d_coh ≤ d_app(1 + 2^(l−L)κ) then
7      return p_coh
8    else
9      return p_app
```

Where $F_l(p)$ is used to denote the concatenation of all feature vectors within the neighborhood (5x5 or 3x3) of both images A and A' and also both the current l and $l-1$. Similarly, $F_l(q)$ is the concatenation of feature vectors for images B and B'. κ is a simple coherence parameter, where the larger the value, the more is coherence favored.

The Image Analogies [7] framework was further extended to handle 3D renders [14] or animations [9], but these proved to be too computationally demanding for real-time use.

1.3.2 The Lit Sphere

The Lit Sphere is not a patch-based method, but the idea of using normals as a local guiding channel is extended upon in the following methods.

In parallel to Image Analogies [7] (described in subsection 1.3.1), a technique called The Lit Sphere, proposed by Sloan [8], was developed. The method tries to leverage vertex normals of a 3D model to create correctly illuminated outputs, also known as environment mapping [40]. The user provides a stylized (shaded) sphere as an exemplar. An example from the original article is shown in figure 1.4. The approach can handle only a simple shading scenario where the target and exemplar scenes have the same lighting environment and often leads to stretched-texture artifacts. Please see figure 1.7 to see the limitations of this approach.

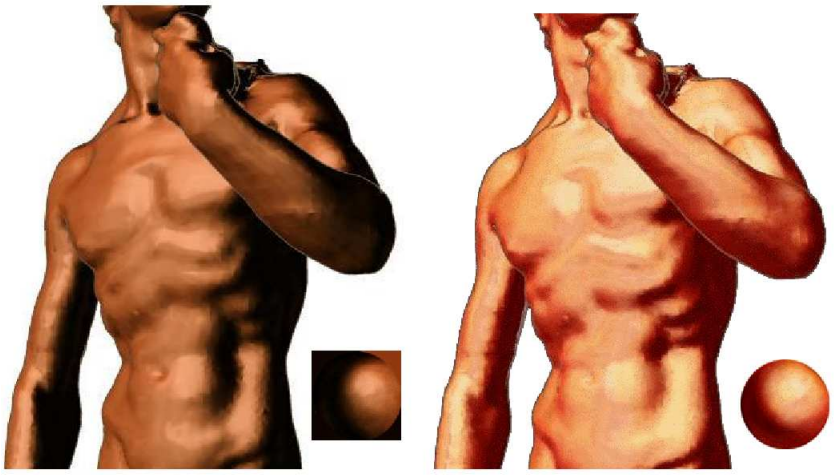


Figure 1.4: Stylized David model with corresponding hand-drawn spheres [8].

1.3.3 Stylizing Animation by Example

The Stylizing Animation by Example [9] method is mainly focused on animation, where an artist provides painted keyframes and the algorithm tries to calculate the in-betweens while maintaining temporal coherence. An output of this method can be found in figure 1.5.

Figure 1.5: Stylizing Animation by Example outputs. Top row: stylized animation; bottom row: input shaded images. The two lateral frames are keyframes painted by an artist. [9]

The algorithm shares its core with Image Analogies [7] (subsection 1.3.1). The goal function G_t serves as an evaluation for different offsets of individual pixels. The algorithm tries to minimize the goal function.

$$G_t(\boldsymbol{p}) = \sum_{\Delta \boldsymbol{p} \in \Omega} w(\Delta \boldsymbol{p}) g(\boldsymbol{p}, \Delta \boldsymbol{p})$$

Where $\boldsymbol{p}$ is a point in $\hat{S}$, Ω represents a set of vectors from $\boldsymbol{p}$ to its neighbors, and $w(\Delta \boldsymbol{p})$ are weights for given vectors. The weights have Gaussian fall-off in a 9 x 9 neighborhood. The overall goal function is a sum of four main goals:

$$\begin{aligned} g(\boldsymbol{p}, \Delta \boldsymbol{p})) = w_{out} g_{out}(\boldsymbol{p}, \Delta \boldsymbol{p}) + w_{in} g_{in}(\boldsymbol{p}, \Delta \boldsymbol{p}) \\ + w_{sc} g_{sc}(\boldsymbol{p}, \Delta \boldsymbol{p}) + w_{tc} g_{tc}(\boldsymbol{p}) \\ + w_h g_h(\boldsymbol{p}) + w_{dt} g_{dt}(\boldsymbol{p}) \end{aligned}$$

The weights w are meant to be manipulated by a user and are not important in this explanation. These six goal functions can be explained as follows:

- g_{out} – make each local neighborhood of output image $\hat{I}$ look like a neighborhood from exemplar style output image $\hat{S}$
- g_{in} – make each local neighborhood of input image I look like a neighborhood from exemplar style input image S
- g_{sc} – make mapping $\boldsymbol{p} \to M_t(\boldsymbol{p})$ spatially continuous
- g_{tc} – prevent sudden color changes caused by motion
- g_h – penalize repeated patterns with a histogram of offsets

- g_{dt} – maintain painterly stroke styles

The solution space has $\mathcal{O}(TN)$ dimensions for T frames and N pixels per frame, so an optimization with a heuristic is needed. To arrive at convergence earlier, the method uses coarse-to-fine synthesis with multiple forward-backward sweeps at each level with parallel PatchMatch [41] as the core of the optimization. In a sweep, a new frame t is generated by randomly merging the previous frame $t-1$ with frame t from the previous level $l-1$ of coarseness. Diagram 1.6 shows the described heuristic.

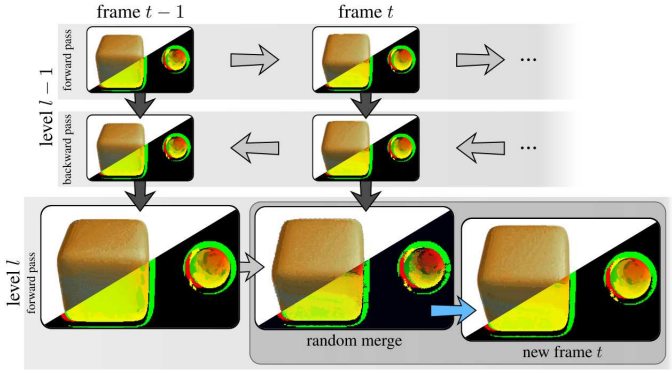


Figure 1.6: Stylizing Animation by Example heuristic diagram [9].

1.3.4 Recent improvements

The Lit Sphere [8] (subsection 1.3.2) approach was extended, where instead of using normals as guidance, UV coordinates are used [31]. This allowed the artist to draw a stylized 2D image of a 3D model and use that as an input exemplar. The style is then transferred using texture mapping. While this approach works in real-time, it distorts high-frequency details. The StyLit [14] approach (described in more detail in the Background chapter 2.1) resolves the issue by utilizing both local guidance and textural coherence. However, this improvement came with higher computational demand, making it unusable in real-time applications.

Recently, Styleblit [1] has come with a solution that is on-par with the mentioned methods when it comes to the quality of the outputs but has lowered the computational requirements by multiple orders of magnitude, making it suitable even for mobile devices. StyleBlit is described in more detail in the Background chapter 2.2. To visualize the mentioned methods, a comparison between them is shown in figure 1.7.

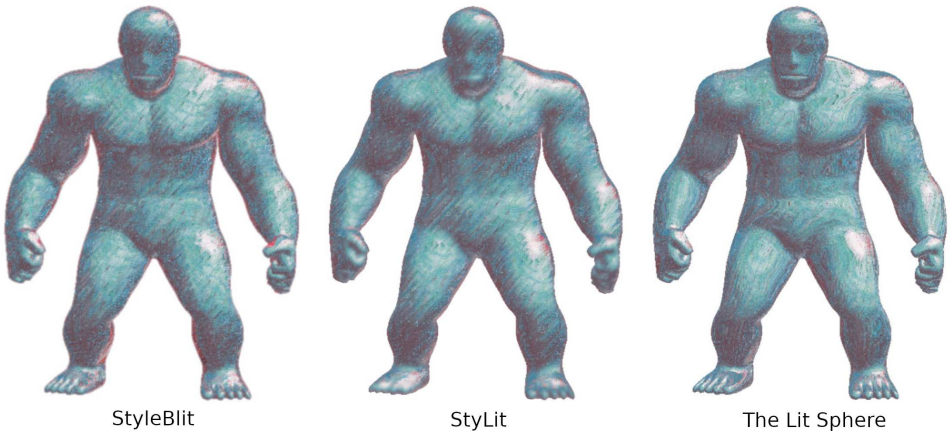


Figure 1.7: A comparison from StyleBlit's article [1] between the StyleBlit, StyLit, and The Lit Sphere methods. The StyleBlit approach produces similar result quality while being orders of magnitude faster when compared to StyLit. The Lit Sphere approach is comparable in speed to StyleBlit but does not retain high-level structure (directional brush strokes).

Further building on the StyLit [14] approach, StyleProp is a method of rendering 3D models from different angles with only a single hand-drawn exemplar. The method achieves real-time performance with a pre-calculated set of sparse samples. The algorithm uses the StyLit [14] method for patch-based synthesis. At first sight, the StyleProp method, visualized in figure 1.8, might seem similar to the Stylizing Animation by Example 1.3.3. However, the difference is that the Stylizing Animation by Example technique maintains coherence only in one dimension – in time. In this scenario, consistency across all possible dimensions is required.

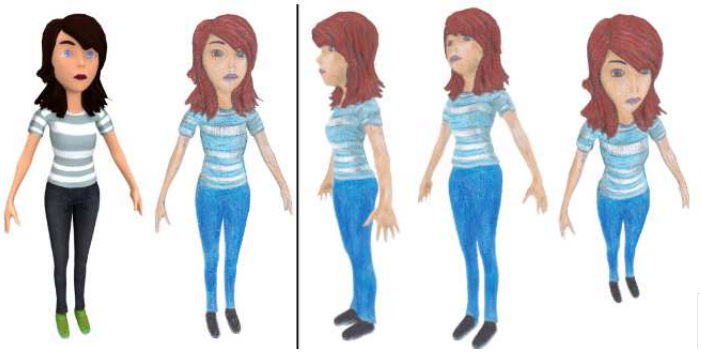


Figure 1.8: Results from StyleProp [10].

1.4 Neural-based methods

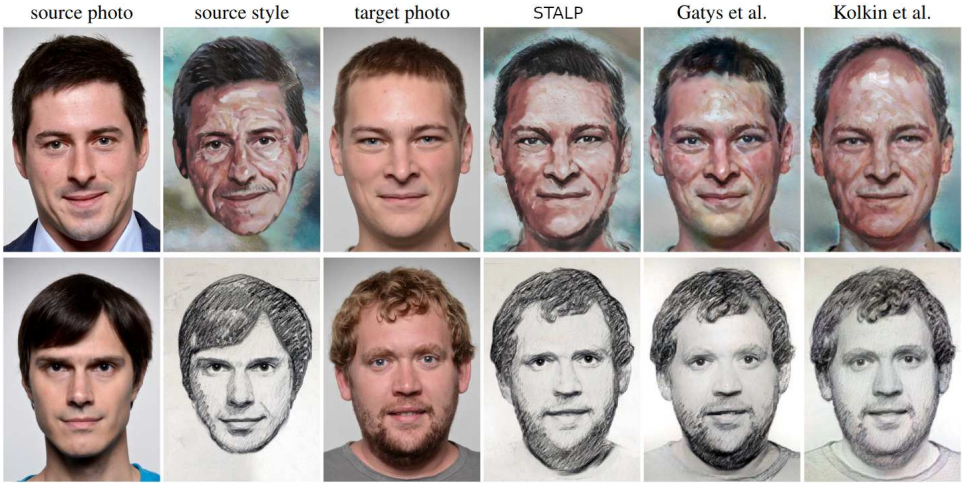


Figure 1.9: A comparison of stylized example outputs from the STALP article [11]. Methods are shown in order are [11, 12, 13].

Another branch of style transfer based on a deep convolutional neural network (DCNN) pre-trained for object recognition [42] was originally proposed by Gatys [12] and became popular thanks to publicly available implementations and seemingly impressive results. A downside to this approach is that it cannot reproduce fine details from the original style exemplar [14]. This downside can be mitigated with a combination of patch-based synthesis and the neural-style transfer [43]. However, these approaches are not suitable for real-time applications.

The problem of style transfer can also be solved by using generative adversarial neural networks (GANN) [44]. These can be used for both image [45] and video [46] stylization. Some techniques use an encoder-decoder scheme using a single network [47]. While these approaches can be employed in real-time applications, they require a large database of training data.

Subsequent work building upon [12] produced many methods, including a recent method named Style Transfer by Relaxed Optimal Transport and Self-Similarity [13]. This method focuses on stylizing videos by defining a sophisticated loss function and manages to produce results that are coherent in time while also maintaining the style.

A current state-of-the-art method in the neural-based branch is the STALP: Style Transfer with Auxiliary Limited Pairing [11] method. The training process is a combination of two objectives – minimize the loss on (stylized and original) keyframes and also minimize loss between the output images and artist-created style images in the same domain. An output from STALP and other mentioned neural-based methods can be found in figure 1.9.

1.5 Summary

Multiple branches of stylization methods were explored. A brief summary of each follows.

1.5.1 Pattern-based methods

The pattern-based methods were among the first to tackle computer-assisted stylization, but even with recent developments are not suitable for our use-case due to limited stylization variations and computational requirements.

1.5.2 Filter-based methods

A common example of stylization, the filter and image processing methods can have great performance but also suffer from a limited stylization variability.

1.5.3 Patch-based methods

The patch-based methods have greatly expanded on the Image Analogies and The Lit Sphere methods. Using various guiding channels in a patch-based synthesis led to producing state-of-the-art stylizations. These methods suffer from high computational demands. However, the method StyleBlit manages to perform stylization in real-time with guidance based on normals.

1.5.4 Neural-based methods

While the results are comparable in quality to patch-based methods, the neural-based methods require training and thus cannot be employed in real-time applications when the style exemplar is acquired at run-time.

Chapter 2

Background

The goal of this thesis is to merge the stylization of a 3D model and the acquisition of an exemplar into an augmented reality experience. To achieve the goal, this thesis explores two related stylization methods in detail, how the style can be acquired, and the augmented reality concept. Background information for Unity and OpenCV is also provided.

2.1 StyLit: Illumination-guided Example-based Stylization of 3D Renderings

The main difference between the StyLit [14] method and the previously mentioned methods is that the stylization is based on light propagation in the scene, not colors. As the name suggests, the process is example-based. This means that an artist has to provide a style exemplar (usually a sphere on a table) with global illumination effects. The exemplar is then aligned to a simple 3D scene, which allows the algorithm to synthesize renderings of complex new scenes with the visual style of the exemplar. Sample outputs are displayed in figure 2.1. The StyLit method can handle any guiding channels including normals, although they produce subpar results.

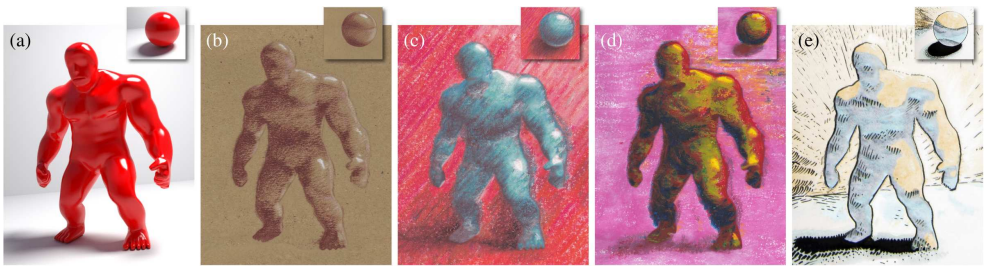


Figure 2.1: StyLit outputs example using various style exemplars [14].

2.1.1 Light Path Expressions

In addition to the classic schema from Image Analogies [7] of exemplar, stylized exemplar and, target scene as inputs, the method can also leverage multiple images obtained by a technique known as Light Path Expressions (LPE) [48]. These images (figure 2.2) contain illumination information from 3D scenes that helps guide the synthesis algorithm.

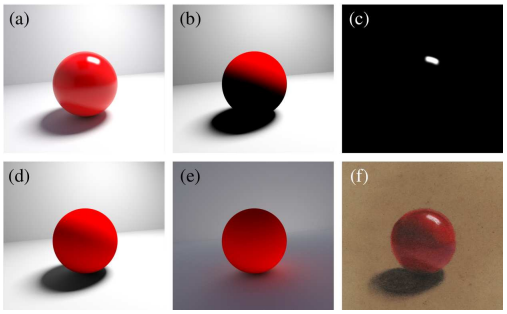


Figure 2.2: Style exemplar with Light Path Expressions images used in StyLit [14].

2.1.2 Algorithm

The method uses the following optimization scheme, which minimizes energy:

$$\sum_{q \in \boldsymbol{B}} \min_{p \in \boldsymbol{A}} E(\boldsymbol{A}, \boldsymbol{B}, p, q, \mu)$$

Here $\boldsymbol{A} = \{A, A'\}$, $\boldsymbol{B} = \{B, B'\}$, where A is exemplar scene with LPE channels, A' denotes stylized exemplar aligned to the exemplar scene, B is target scene with LPE channels, and B' is a new target image. p and q are pixels from A' and B' respectively. μ represents a weight that controls the influence of guidance.

The algorithm uses multiple iterations from coarse to fine resolution:

```
1  function StyLit(A, A', B, B'_k)
2    for each pixel q ∈ B_k do
3      NNF(q) = argmin_{p∈A} E(A, B, p, q, μ)
4
5    for each pixel q ∈ B_k do
6      B'_{k+1}(q) = Average(A, NNF, q)
7
8    return B'_{k+1}
```

Where B'_k contains current pixel values and B'_{k+1} the updated values. NNF is an abbreviation for Nearest Neighbor Field – mapping of a source patch to each target patch.

To mitigate the problematic wash-out effect of this approach, the solution encourages uniform source patch use, while at the same time leveraging the idea of reversed NNF retrieval [49]. This allows the algorithm to predict cases of erroneous assignments ahead of time, estimating an error budget T. The algorithm is then altered to maximize the number of used source patches $|\boldsymbol{A}^*|$, while keeping the sum of the energy function less than T. The reasoning behind this mitigation strategy is depicted in figure 2.3.

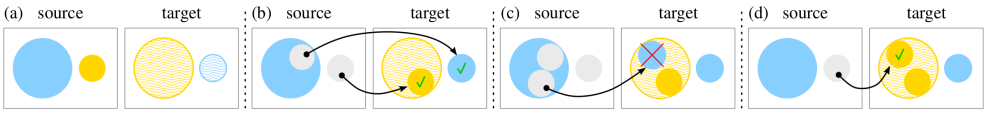


Figure 2.3: How StyLit mitigates erroneous assignments [14]

2.2 StyleBlit: Fast Example-Based Stylization with Local Guidance

The StyleBlit [1] method tries to achieve both high-quality stylized renderings and real-time performance. The approach is patch-based with local guidance. The StyleBlit method is inspired by StyLit but uses normals as local guidance to avoid computationally expensive optimization, which makes it vastly faster than other competing methods. It seeks large coherent chunks of style exemplar's regions directly using pixel-level operations. A demonstration of this method can be seen in figure 2.4, where the first row is the source exemplar and the second row is the stylized result.

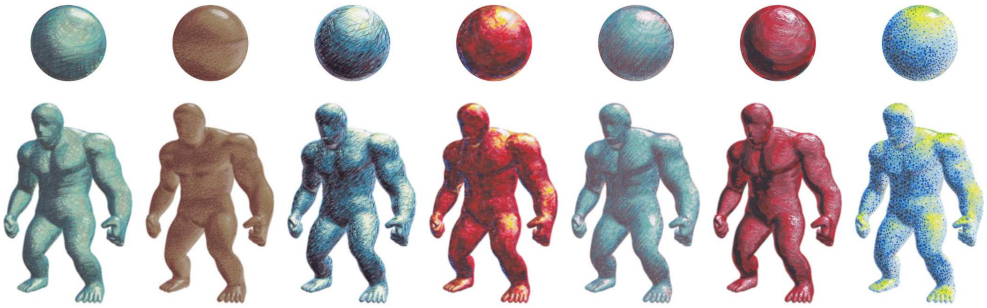


Figure 2.4: Results from StyleBlit [1].

2.2.1 Approach

The core idea behind StyleBlit is visualized in figure 2.5, where (b) is a randomly selected pixel from the target image and (a) is a location in the source exemplar that is found using $(b)'s$ guidance value. The patch (c) consists of all neighboring pixels of (b) with their guidance value difference

below a user-defined threshold. Then, all pixels from the chunk are copied to the target image (d). These steps are repeated until all patches are determined (e).

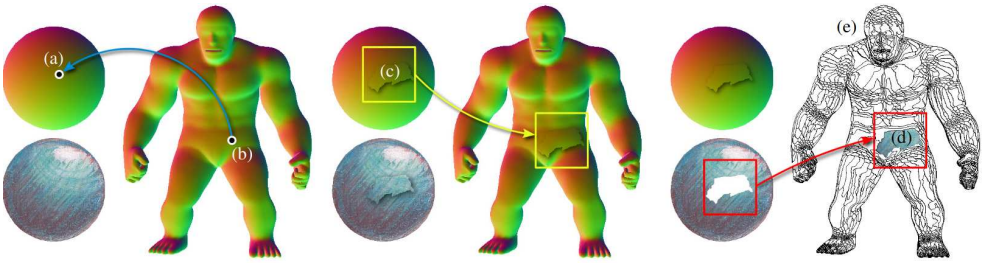


Figure 2.5: A visualization of StyleBlit's patch-based approach [1].

2.2.2 Brute force algorithm

The simplicity of StyleBlit's approach is demonstrated in the following pseudocode:

```
1  function StyleBlit(C_S, G_S, G_T, t)
2    for each pixel p ∈ C_T do
3      if C_T[p] is empty then
4        u* = argmin_u ||G_T[p] − G_S[u]||
5          for each pixel q ∈ C_S do
6            if C_T[p + (q − u*)] is empty then
7              e = ||G_T[p + (q − u*)] − G_S[q]||
8              if e < t then
9                C_T[p + (q − u*)] = C_S[q]
10   return C_T
```

Where C_S is style exemplar, G_S are source guides, G_T represent target guides and t is the error threshold.

While easy to implement, the algorithm suffers from its sequential nature and redundant querying of target pixels that have already been assigned in a patch.

2.2.3 Parallel algorithm

To avoid redundant visiting of target pixels and allow for parallelization, algorithm 2.2.3 uses a hierarchy of target seeds with different levels of granularity. The levels are traversed in order and the nearest seed is found on each. The algorithm checks if the guidance error e is below a specified threshold t. If the condition passes on any level, the value mapped from C_S is copied to C_T and the search stops. The nearest seed search is randomly offset using the $RandomJitterTable$ that contains values between 0 and 1.

Outside of the levels hierarchy, the parameters are the same as in the brute force version 2.2.2.

```
1  function NearestSeed(pixel p, seed spacing h)
2    b = ⌊p/h⌋
3    j = RandomJitterTable[b]
4    return ⌊h * (b + j)⌋
5
6  function NearestSeed(pixel p, seed spacing h)
7    d* = ∞
8    for x ∈ −1, 0, +1 do
9      for y ∈ −1, 0, +1 do
10       s = SeedPoint(p + h * (x, y), h)
11       d = ||s − p||
12       if d < d* then
13         s* = s
14         d* = d
15
16 function ParallelStyleBlit(pixel p, C_S, G_S, G_T, t)
17   for each level l ∈ (L, ..., 1) do
18     q_l = NearestSeed(p, 2^l)
19     u* = argmin_u ||G_T[q_l] − G_S[u]||
20     e = ||G_T[p] − G_S[u* + (p − q_l)]||
21     if e < t then
22       C_T[p] = C_S[u* + (p − q_l)]
23       break
```

2.2.4 Extensions

When needed, the algorithm can be modified to apply linear blending for the seams between patches. This is achieved by replacing the copying of colors with pixel coordinates from the source exemplar. The final color is determined as the average color of neighboring pixels from source patches that intersect the target pixel.

While this method is suitable for stochastic exemplars, it does not work well with exemplars that feature both smooth gradients and high-frequency features. This can be mitigated by separating the input exemplar into a smooth base and high-frequency detail layer using a Gaussian filter and its subtraction from the exemplar. The base layer is stylized with the Lit Sphere algorithm [8]. The detail layer is stylized by StyleBlit. Lastly, both layers are put together. This approach can be observed in figure 2.6, where (a) denotes the source exemplar, (b) is the result of the non-layered approach, (c) is the base layer with the corresponding result (e), (d) is the detail layer with the corresponding result (f), and finally, (g) is the sum of both layers.

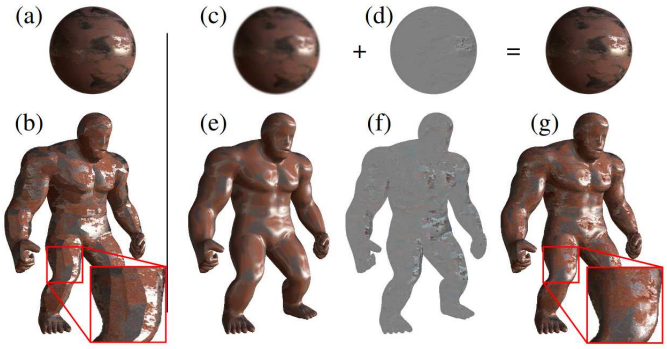


Figure 2.6: StyleBlit's multi-layered approach [1].

2.3 Style exemplar acquisition

The application needs to be capable of capturing a template drawing. To achieve this, the application has to detect the exemplar and perform a perspective transformation to align the scanned image with a pre-set scene. A blank style exemplar is shown in figure 2.7.

Figure 2.7: A blank style exemplar used in StyleBlit [1].

2.3.1 Projective transformation

Because the camera that will be scanning the template drawing will most likely be hand-held, the captured image containing the style exemplar needs additional processing to be correctly aligned. This means detecting the four corners of the exemplar in 3D space and transforming the cut-out into a 2D rectangle with predefined dimensions. A visualization of this transformation from both 2D and 3D perspectives can be seen in figure 2.8.

From a mathematical point of view, the transformation is a function that maps one vector space into another, which can be done by multiplying a matrix consisting of pixel coordinates and 1 with the following matrix:

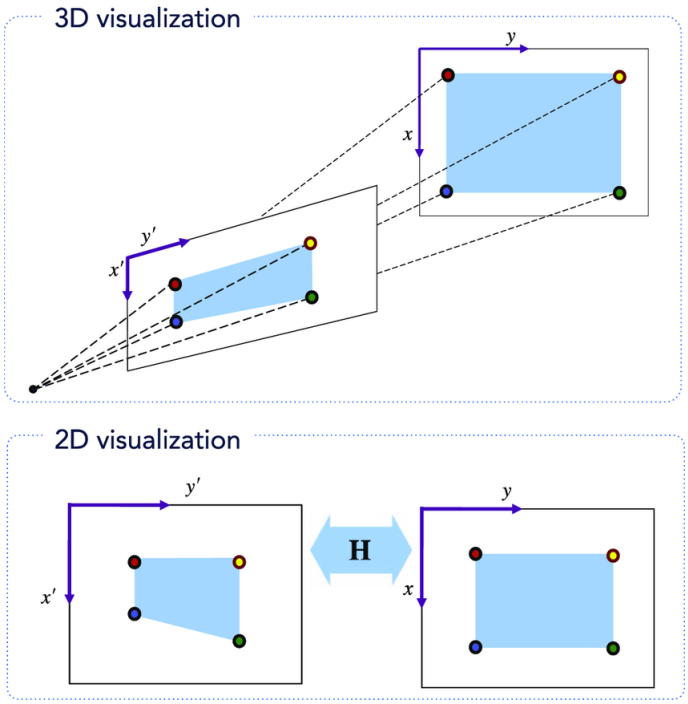


Figure 2.8: Projective transformation visualization from 3D and 2D perspectives [15].

$$\begin{pmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{pmatrix}$$

Where:

- $\begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix}$ is called a rotation matrix. This matrix defines the transformation to be performed (scaling, rotation).
- $\begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$ B is the translation vector. It moves pixels on the x and y axes.
- $\begin{pmatrix} c_1 & c_2 \end{pmatrix}$ is the projection vector.

So we get the following equation:

$$\begin{pmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} \tag{2.1}$$

Here, x and y are coordinates of a pixel that need to be transferred. x' and y' are the corresponding target coordinates. Equations for both can also be formulated:

$$x' = \frac{a_1 x + a_2 c + b_1}{c_1 x + c_2 y + 1}$$

$$y' = \frac{a_3 x + a_4 c + b_2}{c_1 x + c_2 y + 1}$$

2.3.2 Mapping

The equation 2.1 described in the previous subsection (2.3.1) is doing what is called a forward mapping – it goes through every pixel in the original image and transforms it to a target pixel. The inherent problem using this approach is that when the scaling factor is smaller than 1, then it maps multiple pixels from the source image to the same coordinations in the target image, and when the scaling factor is greater than 1, it can leave unmapped "holes" in the target image. An example of this problem can be observed in figure 2.9.

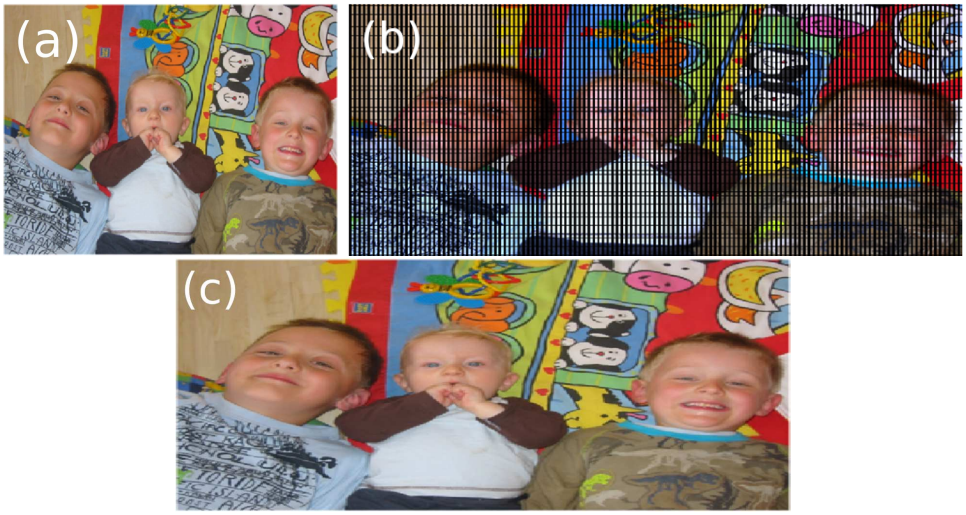


Figure 2.9: An example of a forward-mapping scaling problem. Taken from a web tutorial on what-when-how [16]. The input (a) is stretched to a larger image (b), leaving holes. The third image (c) depicts the correct transformation.

To solve these issues, a technique called backward mapping needs to be used. The technique goes through every target pixel and applies a reverse transformation to it to get the source coordinates.

Both forward and backward mapping have a problem in that the resulting coordinates are not integers, but floating-point numbers. The solution to this issue is a technique called resampling [17] and has two simple approaches. The

first is a nearest-neighbor approach – round the floating-point numbers to get an integer. The second, more sophisticated way to resolve the problem is to take pixel values from the four surrounding pixels and average their values based on their proximity to the calculated coordinates. This is equivalent to bilinear interpolation [50]. Both approaches are shown in figure 2.10.

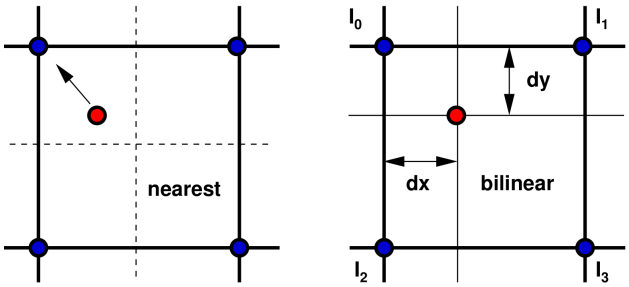


Figure 2.10: Visualization of the nearest neighbor and bilinear interpolation resampling techniques. Here I_* are points from the source image, and dx with dy are weights based on proximity [17].

2.3.3 Exemplar detection

The last and most important piece of the puzzle the application needs to be capable of is detecting the style exemplar with the device's camera. This discipline is called object recognition, more specifically image recognition and there are two branches of approaches.

The first, older branch is the non-neural approach, which consists of two steps: define features (keypoints) with one of the algorithms, then identify features in a captured image and compare these new features with the defined ones. Algorithms from this branch include Scale Invariant Feature Transform [51], Speeded Up Robust Features [52], KAZE features [53] and Binary Robust Invariant Scalable Keypoints [54].

The second branch is based on convolutional neural networks (CNN) [55] and also has two steps: training of the CNN on a (large) dataset, with the second step being the classification of the target image. This approach is more general because it is independent of prior knowledge in feature extraction and does not need human intervention. Notable approaches of building these networks include R-CNN [56], Single Shot MultiBox Detector [57] and You Only Look Once [58].

While these approaches are the state-of-the-art of image recognition, they are not suitable to the specific use-case of recognizing the style exemplar, because the exemplar can be stylized arbitrarily. This causes the style exemplar to have unknown feature points, rendering both techniques unusable. Instead, the application can use easily recognizable (from the application's perspective) fiducial markers [18]. These markers can serve as a point of reference, measure, and orientation (pose). The biggest advantage of this marker-based approach

is the speed of detection and the precision of alignment with the pre-set scene. On the other hand, using markers can somewhat degrade the user experience and cause confusion. How these fiducial markers can look is shown in figure 2.11.

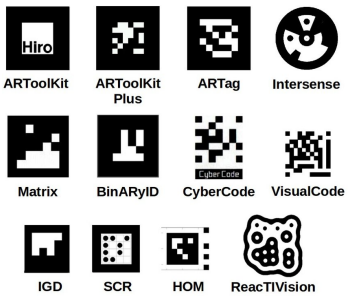


Figure 2.11: Examples of different fiducial markers [18].

2.4 Augmented reality introduction and terms

The augmented reality concept falls into a broader category of immersive technologies and often gets confused for others. There are a few terms that need to be defined in order to have a clear idea of what augmented reality refers to.

2.4.1 Extended reality

Extended reality (XR) is an emerging term that represents a superset of all immersive technologies with arbitrary combinations of both real and virtual environments in human-computer interactions. XR encompasses all terms defined in this section. The XR spectrum is visualized in figure 2.12.

2.4.2 Augmented reality

The concept of augmented reality (AR) is a process of adding virtual objects into a camera feed from a real device before displaying the feed to the user. This enables an attractive relay of information to a user interactively. As this thesis focuses mainly on AR, there is a whole section (2.5) dedicated to it.

2.4.3 Virtual reality

Situated on the other end of the XR spectrum, virtual reality (VR) presents a simulated fictional world. The user is wearing a headset that is fully blocking their view and captures their movement. VR is mainly known for games, but it is also used in more practical use-cases, for example a flying simulator in the military.

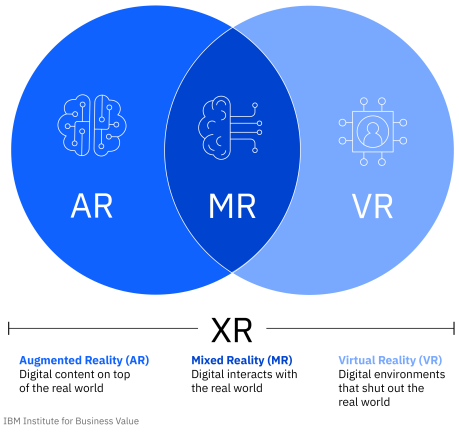


Figure 2.12: The extended reality spectrum [19].

2.4.4 Mixed reality

A step between augmented reality and virtual reality, mixed reality (MR) can be based in either the real world or the virtual world.

- **Based in the real world** – There are virtual objects with which the user can interact. The user usually wears a headset with built-in cameras and sensors that capture the environment. This form of mixed reality is often considered an advanced form of AR. See figure 2.13 for an example usage.
- **Based in the virtual world** – The virtual world is intertwined with the real one. This means that while the user is completely immersed in virtual reality, they can see representations of walls and objects that have a counterpart in the real world. To see how the worlds are mixed together, see figure 2.14.

Figure 2.13: An example of mixed reality based in the real world [19].

Figure 2.14: An example of mixed reality based in the virtual world [20]. The first image is from the real world, while the second shows how the mixed reality presented to the user looks.

2.5 Augmented reality

This section briefly describes the technical side behind AR, showcases some applications of AR, and explores different options for its implementation.

2.5.1 How it works

Augmented reality can be divided into two categories based on how they are tracking the real world. The first category is marker-based AR. As the name implies, it uses fiducial [18] markers for better environment tracking, allowing it to place virtual objects with precision. This is useful for multiplayer applications or when precise placement of virtual content is required. A visualization of how the marker-based AR can look from the user's point of view is shown in figure 2.15.

Figure 2.15: An example of marker-based application of augmented reality from the user's perspective [21].

The second category of augmented reality is markerless. It generally works by tracking the position of the device in the real world. This is achieved by a combination of readings from the device's sensors (gyroscopes

and accelerometer) and tracking feature points in time using the device's camera. The real-world representation is often supplemented with flat surface detection and light estimation. With the representation in hand, the application is then able to place objects or other information into the real world.

A set of distinct features [29] related to AR has emerged over time, they can be summarized in the following list:

- **Device tracking**: Track position of the device in the real world.
- **Plane detection**: Detect surfaces.
- **Point clouds**: Track surrounding environment with feature points.
- **Anchor**: A user-chosen position that is being tracked by the engine.
- **Light estimation**: Intensity and color correction based on guesssed light in environment.
- **Environment probe**: Able to simulate reflections of placed objects (cube mapping).
- **Face tracking**: Detect and track faces and their regions.
- **2D image tracking**: Detect and track 2D images in the real world.
- **3D object tracking**: Detect and track real 3D objects.
- **Meshing**: Generate a mesh that reflects the real world.
- **Body tracking**: Track a person in the physical environment.
- **Colaborative participants**: Share data about environment between multiple devices running the same application.
- **Raycast**: Determine if a ray, defined by an origin and direction, intersects a registered object.
- **Pass-through video**: Use camera feed as background for AR content.
- **Occlusion**: Objects are shaded with (ambient) occlusion.

2.5.2 Real world usage examples

IKEA was one of the first companies to realize AR's potential and has developed a mobile application[1] that can place pieces of furniture into the real world, thus allowing the customer to get an idea of how the product will fit (both physically

[1]Currently, the app is only available for iOS, though an Android version was briefly available.

and visually) into their environment. Figure 2.16 shows a promotional picture with a usage example. [22]

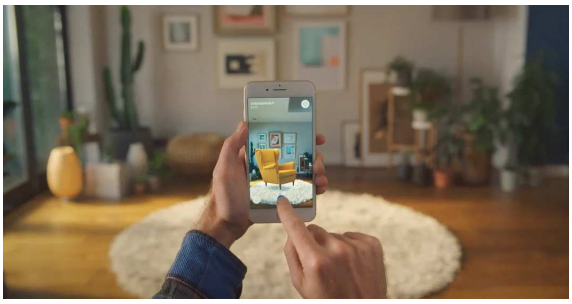


Figure 2.16: Promotional picture for IKEA place [22].

One of the most common uses of AR nowadays are so-called face filters. Popularized by Facebook and Instagram, these filters allow the user to add objects (e.g. a crown, bunny ears) to their face in a humorous manner. An example of these filters can be seen in figure 2.17.

Figure 2.17: Promotional picture for Instagram face filters [23].

AR also has applications in the military. For example, it helps pilots of fighter jets by projecting vital information on see-through displays in their helmets (figure 2.18).

Figure 2.18: Augmented reality helmet for F-35 Lightning II fighter jet [24].

2.5.3 ARCore

ARCore [59] is a set of SDKs for building augmented reality applications. The platform was released in 2018 as an answer to Apple's ARKit (described in 2.5.4), is developed by Google, and is open-source – licensed under the Apache license version 2 [60]. ARCore can be used with both Android[2] and iOS. The platform can also be used with popular game engines Unity and Unreal engine. This enables cross-platform cooperation through ARCore Cloud Anchor [61].

On Android, which is the focus of this thesis, the devices need to be certified by Google to have the ARCore APK available. The market share of these certified devices is estimated to be at 41 % [62]. A list of all supported device models is available at the ARCore documentation page [63].

2.5.4 ARKit

Released in 2017 with iOS 11, ARKit [64] is the first widely used SDK for AR development. Developed by Apple, ARKit was able to maintain its position at the forefront of AR SDKs. Apple has also built a framework RealityKit [65] that makes the development of AR applications significantly simplified.

Arguably, the biggest downside of ARKit is that it is only available for iPhones (starting with 6S) and iPads (starting with the iPad Pro). However, when compared to ARCore (subsection 2.5.3), ARKit has a few additional substantive features, for example, 3D object tracking and human segmentation.

2.5.5 AR Foundation

AR Foundation [66] is a package that is used to build multi-platform AR applications in Unity. It does not implement any AR features itself, instead, it presents a unified interface for developers to use. Under the hood, AR

[2]It is worth mentioning that Android is by far the most used platform of those supported by ARCore.

Foundation uses both ARCore (subsection 2.5.3) and ARKit (subsection 2.5.4) for mobile applications, and Magic Leap and HoloLens, which are used with their respective AR glasses. These four platforms all have a different set of features, which forces the developer to use only those features that the targeted platforms share in common. A list of the features can be seen in table 2.1.

Feature	ARCore	ARKit	Magic Leap	HoloLens
Device tracking	✓	✓	✓	✓
Plane tracking	✓	✓	✓	
Point clouds	✓	✓		
Anchors	✓	✓	✓	✓
Light estimation	✓	✓		
Environment probes	✓	✓		
Face tracking	✓	✓		
2D Image tracking	✓	✓	✓	
3D Object tracking		✓		
Meshing		✓	✓	✓
2D & 3D body tracking		✓		
Collaborative participants		✓		
Raycast	✓	✓	✓	
Pass-through video	✓	✓		
Occlusion	✓	✓		

Table 2.1: A list of AR-related features that the specific platforms used in AR Foundation support. Sourced from the AR Foundation manual [29].

AR Foundation unites the main SDKs used on mobile platforms and therefore is a clear choice of this analysis.

2.6 Unity

2.6.1 General information

Formerly known as Unity3D, Unity is a multi-platform game engine developed by Unity Technologies with its first release in 2005 [67]. Thanks to its large community and detailed documentation, Unity is the most popular game engine with some reports stating that 61 % of game developers are using Unity [68]. It comes with an IDE called Unity Editor [69] for managing scenes, assets, and builds. The Unity Editor is supported on all major platforms – Windows, macOS, and the Linux platform.

Unity has a yearly subscription service model, but has a free version for students and personal use, as long as the revenue is less than $100K in the last 12 months [70].

At the time of writing, Unity supports targeting 19 different platforms [71]:

- **Mobile** – Android, iOS, and tvOS
- **Console** – PlayStation 4 and 5, Xbox One and Series X, Nintendo Switch, and Google Stadia
- **Desktop** – Windows (7, 10, and 11), Universal Windows Platform, macOS, and Linux
- **Web** – WebGL
- **Extended reality** – Oculus, Windows Mixed Reality, Magic Leap, ARCore, and ARKit

2.6.2 Installation

The recommended way of installing the Unity Editor is through the Unity Hub [72]. The Unity Hub is a management tool for Unity projects, Editor installations, and Unity license. It lets the user select which target platforms' build support should be installed along with the Editor installation, while also supporting multiple Editor installations. This is shown in figure 2.19. The Unity Editor can also be downloaded directly through the Unity download archive page [73].

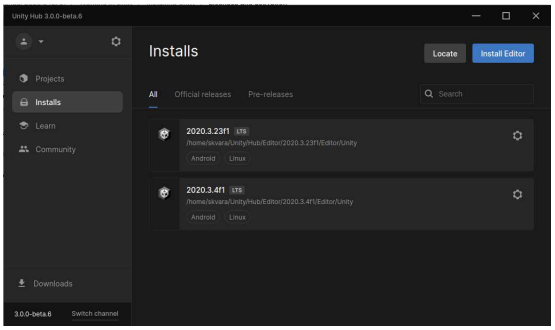


Figure 2.19: The Unity Hub with different versions of the Editor installed.

2.6.3 Scenes

A Unity project is divided into scenes. Scenes can be thought of as separate screens, each with its own objects and settings. Screens can represent the main menu, settings page, or individual levels. A scene is where the user can create and manipulate content. Every object (camera, 3D model, button) in a scene is a GameObject. GameObjects are organized in a tree hierarchy with the scene being the root.

2.6.4 GameObject

A GameObject can be seen as a container for components. The only component all GameObjects have is the Transform component, which defines the position, rotation, and scale. The attached components define how the GameObject looks, how it updates, and how it reacts to input. There are many components provided by the Unity Editor. However, the user can write their own components (subsection 2.6.5).

2.6.5 Scripting in Unity

The programming language of choice is C# and the recommended IDE for writing the code is Visual Studio or Visual Studio Code [74]. While it is possible to get substantial work done in Unity without needing to handle code, understanding the code opens up a lot more possibilities. This subsection describes the basics of writing code, which is called scripting in the Unity lingo [75].

All newly created scripts (see listing 2.1) with the Unity Editor inherit from a class called MonoBehaviour, which provides methods that will be called according to the Unity lifecycle. Inheriting from MonoBehaviour allows the script to be attached to a GameObject as a component. Without an attachment to an active GameObject, none of the code in the script will be executed. When the script is attached as a component, all of its public variables are visible and modifiable in the Unity Editor. Other components or whole GameObjects can be linked through those public variables, allowing for interaction between scripts.

```
using UnityEngine;
using System.Collections;

public class MainPlayer : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

Listing 2.1: The initial contents of a newly created script file.

2.6.6 Lifecycle

As mentioned in the Scripting in Unity subsection (2.6.5), scripts that inherit from the MonoBehaviour class gain access to a number of lifecycle methods. A full list is available in the Unity documentation [25]. A list of methods that are important for understanding the Unity project accompanied by a diagram 2.20 follows:

- **Awake** – Called when a script instance is being loaded. Mainly used in place of a constructor to set up references between scripts. Awake is called on all active GameObjects before any Start method calls.
- **OnEnable** – Called every time the object becomes enabled. Ideal place for subscriptions.
- **Start** – Called after the script is enabled (on the same frame) and before any Update method calls. The programmer can assume other components have been initialized.
- **OnMouseXXX** – This particular method does not exist and is used as a placeholder for all methods that can be called based on the user's interaction.
- **Update** – Called every frame. Is the most commonly used method for a game script. Typical usage is a calculation that needs to be done every frame, for example, a rotating object that needs to update its pose every frame.
- **OnDisable** – Called when the object becomes disabled. Can be used to cancel any subscriptions.
- **OnDestroy** – Called when the object is being destroyed. Clean-up should be done here.

2.6.7 Prefabs

As the name suggests, prefabs are essentially blueprints for creating GameObjects. The main usage of prefabs is creating complicated GameObjects at run time, for example, spawning a building as a reaction to a button click. They are also useful when there is a need for using multiple same GameObjects in a scene, for example, to ensure that all the buttons look the same. When used in a scene as a GameObject, editing it will also edit every other instance in the scene.

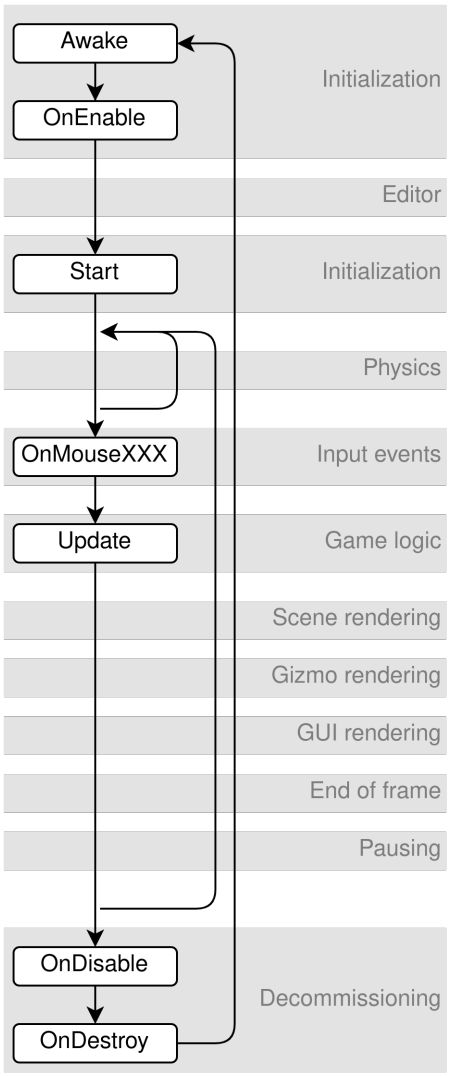


Figure 2.20: Vastly simplified script lifecycle overview. Full version available in the Unity documentation [25].

2.6.8 Assets

Every item that can be used in a Unity project is called an asset. Assets may include music, scripts, 3D models, textures, or color gradients and can be grouped into multiple separate bundles. This allows the project to load and unload them when needed. These bundles can also be distributed to the cloud and downloaded only when needed [76], cutting down on initial download time and saving valuable space.

2.6.9 Packages

A Unity package is a file that contains a collection of assets and scenes, which can be imported into an existing unity project. A package can be created by exporting from a unity project. The exported package will include all dependencies of selected assets or scenes so that no errors will be introduced by importing it.

Unity has its own asset store [77], which is a marketplace that is connected with the Unity Editor. The assets are bundled together in a Unity package. Both Unity Technologies and the community can create and publish assets to the store. Using the asset store can greatly accelerate project development. The store offers anything from textures to entire project samples. Assets can either be free or paid.

2.6.10 Shaders

Writing hand-crafted shaders is not considered basic Unity knowledge, but it is necessary to implement the StyleBlit's [1] algorithm. Shaders are stand-alone programs that can run on the GPU. In Unity, shaders are divided into three categories [78]:

1. Shaders that are part of the graphics pipeline[3]
2. Compute shaders
3. Ray tracing shaders

The shader that needs to be written will output individual pixels, so it falls into the first category. Other categories will not be described here.

A **Shader object** is a Unity-specific way to work with shaders [79]. It can contain multiple shader programs, settings for the GPU, and information for the Unity Editor. Anatomy of a Shader object is visualized with the following nested list, where each level describes the contents of the higher-level item:

[3]Graphics pipeline is a set of steps that takes a 3D scene as an input and outputs a 2D image.

- **Shader object**
  - Information about itself
  - Optional fallback shader
  - One or more **SubShaders**
    - Information about compatibility with hardware, render pipelines, and runtime settings
    - SubShader tags
    - One or more **Passes**
      - Pass tags
      - Instructions for changing the GPU settings
      - Shader programs

In the Shader object's context, tags are reffering to key-value pairs that provide information about the SubShader (e.g., "Queue" = "Geometry") or Pass (e.g., "LightMode" = "Always").

The recommended language for writing shaders is the High-level shader language (HLSL) developed by Microsoft for the Direct3D 9 [80]. Unity originally used the Cg shading language [81], which is very similar and was developed alongside HLSL.

2.6.11 Shader example

To better understand what a shader looks like and how it is written, a basic unlit shader (listing 2.2) will be described.

The Properties block defines shader variables that are visible in the Material Inspector [82]. In the example, there is a single texture property. This texture is then accessed in the Pass block with a sampler.

There are two shader programs in this example – the vertex shader and the fragment shader. The vertex shader runs on each vertex of the 3D model. In this example, it is projecting a 3D coordinate into a 2D window. The fragment shader runs on each visible pixel of the 3D model and takes the output of the fragment shader as input. In this case, the fragment shader looks up the pixel value from the main texture. The directive `#pragma` is used to define the name of the respective shader function.

The code also demonstrates how to use multiple variables contained in structs (`appdata` and `v2f`) as inputs and outputs of functions.

```
Shader "Unlit/SimpleUnlitTexturedShader"
{
    Properties
    {
        [NoScaleOffset] _MainTex ("Texture", 2D) = "white" {}
    }
    SubShader
    {
        Pass
        {
            CGPROGRAM
            // use "vert" function as the vertex shader
            #pragma vertex vert
            // use "frag" function as the pixel (fragment) shader
            #pragma fragment frag

            // vertex shader inputs
            struct appdata
            {
                float4 vertex : POSITION; // vertex position
                float2 uv : TEXCOORD0; // texture coordinate
            };

            // vertex shader outputs ("vertex to fragment")
            struct v2f
            {
                float2 uv : TEXCOORD0; // texture coordinate
                float4 vertex : SV_POSITION; // clip space position
            };

            // vertex shader
            v2f vert (appdata v)
            {
                v2f o;
                // transform position to clip space
                // (multiply with model*view*projection matrix)
                o.vertex = mul(UNITY_MATRIX_MVP, v.vertex);
                // just pass the texture coordinate
                o.uv = v.uv;
                return o;
            }

            // texture we will sample
            sampler2D _MainTex;

            // pixel shader; returns low precision ("fixed4" type)
            // color ("SV_Target" semantic)
            fixed4 frag (v2f i) : SV_Target
            {
                // sample texture and return it
                fixed4 col = tex2D(_MainTex, i.uv);
                return col;
            }
            ENDCG
        }
    }
}
```

Listing 2.2: A simple unlit shader example. Taken from the Unity documentation page [83].

2.7 OpenCV

OpenCV [84] is a cross-platform library for real-time computer vision. With the first alpha release in 2000, it was originally developed by Intel and is free to use under the open-source Apache 2 license [85]. The library includes more

than 2500 algorithms related to computer vision [86] and machine learning.

The main language used in OpenCV is C++ and it has additional interfaces for Python, Java, and MATLAB. The library is extensively used with thousands of community members and over 23 million downloads [87].

2.7.1 ArUco

ArUco [26] is a minimal library for came pose estimation using markers. It is a module in the OpenCV's so-called contrib repository [88]. The OpenCV contrib repository is not part of the official OpenCV distribution.

The creators of StyleBlit [1] that this application is meant to implement have devised a way to scan the exemplar by using ArUco markers. ArUco markers are fiducial markers that are designed to be used in computer vision applications. An example of an ArUco marker is shown in figure 2.21.

Figure 2.21: An example of a generated 6×6 ArUco marker [26].

ArUco markers can have arbitrary (square) dimensions starting from 4×4 (plus border). The dimensionality affects:

- How many markers can be generated.
- How easy it is to detect markers.
- How distinguishable are the markers from each other. This is measured in hamming distance [89].
- How fast is the recognition.

The markers are organized into sets that are called dictionaries [26] and they have a unique id based on order within those dictionaries. Though it is recommended to generate a new dictionary for every application, there are several predefined dictionaries in the library.

The detection process is also able to identify the marker's rotation and pose in the real-world relative to the camera, which is demonstrated in figure 2.22.

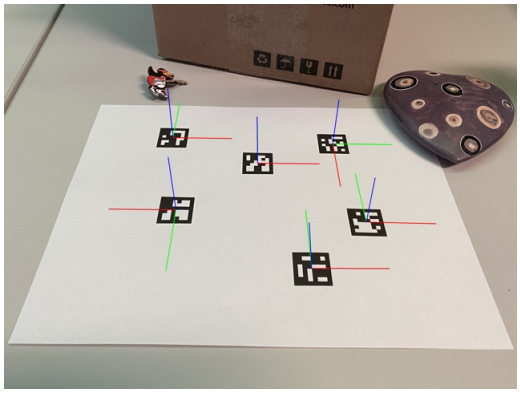


Figure 2.22: An example of detecting ArUco marker's pose and visualizing the axes [26].

Chapter 3

Realisation

The resulting application, named StyleTransfer, provides a unique blend of stylization and exemplar acquisition within an AR experience. StyleTransfer was developed in the Unity game engine and runs on Android devices that both support ARCore [63] and their system version is 7.0 (API level 24) or newer[4]. The application also requires the camera permission for apparent reasons.

3.1 Application design

The use-cases from the user's perspective are not complicated. The user needs to be able to:

- Scan a style exemplar.
- See a stylized 3D model (within AR) and rotate it.

The simplest solution would be to have two screens – one for exemplar acquisition and the other for interaction with the 3D model. However, they can be combined into a single screen with a toggle for state switching.

Given that obtaining an exemplar template, stylizing it, and finally scanning it requires a significant effort from the user, the application will contain multiple prearranged style exemplars to showcase the StyleBlit [1] method. The user will be able to switch between the exemplars as they will be organized into a toggle group. The exemplar acquired by scanning the style template will also need to be a part of this toggle group.

The application will also contain several 3D models to facilitate testing of the stylization algorithm's functionality. These models can also be organized into a separate toggle group. Having two separate toggle groups will allow for testing arbitrary combinations of the exemplar and the 3D model.

[4]Over 90 % of Android devices that have been active in the last month have system version 7.0 or newer worldwide [90].

To visually distinguish between the two toggle groups, they will be separated by the scanning button. The proposed design is presented as a mockup in figure 3.1.

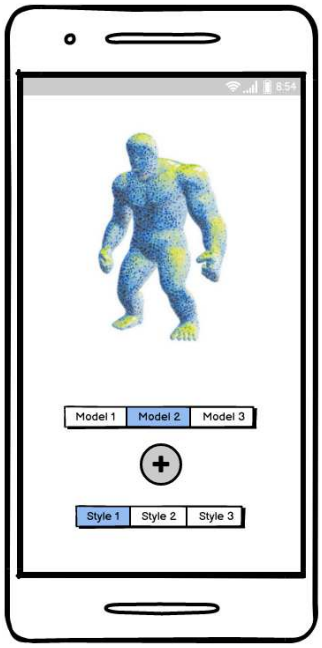


Figure 3.1: A mockup of the proposed design, showcasing two toggle groups and a switch for style scanning. The 3D golem is taken from StyleBlit's article [1]. Made with the Balsamiq wireframing tool [27].

3.2 Main flow

With the first application start, the application asks for the camera permission. After granting the permission, the user is presented with a camera feed overlayed with GUI. At this point, the underlying AR framework is trying to identify horizontal planes (see section 2.5 for an explanation of how this works), which can be helped by the user slowly moving their device. Once a plane is detected, it is marked by transparent white circles. The plane visualization is visible in the first image of figure 3.2.

With a plane detected, the user can place a (stylized) 3D model onto the plane by either tapping it or clicking the place button. The model can be scaled with a scale gesture and rotated with a twist gesture [91]. Both gestures require two touchpoints. The user can choose from the selection of 3D models and style exemplars with the changes happening in real-time. A visualization of the whole main flow is shown in figure 3.2.

Figure 3.2: The main flow, visualized in screenshots. The images show in order: plane detection, 3D model placement, selection of different models and styles, and manipulation of the model.

3.3 GUI

As proposed in the Application design section 3.1, the GUI (shown in figure 3.3) is separated into three distinct rows. The bottom row displays six stylized exemplars. The exemplars form a toggle button group. The active button is indicated with a green circle. Similarly, the first row consists of five 3D models organized into a toggle button group with the green circle marking the active button. The thumbnails preview how the model looks with applied stylization and are generated anew with each style exemplar change. To avoid mistaking the exemplars and 3D models for each other, they are visually separated with the middle row.

Figure 3.3: Screenshots from the StyleTransfer application showing the GUI.

The middle row has three buttons. The first one is just another style exemplar toggle button. It is meant to be replaced when a new style is successfully scanned. The middle button with a plus sign activates style

scanning when pressed. Style exemplar scanning is further explained in section 3.4. The last button, depicting a cube being placed down, serves for moving the 3D model to a different location. When pressed, the model is moved to the middle of the screen, if there is a detected plane at that location. This button was added because the user's gestures for interacting with the 3D model were often mistaken as an intention to relocate the model.

When using a development build, the GUI also displays an FPS counter in the top-right corner (visible in figure 3.2).

3.4 Style exemplar acquisition

The application must be able to perform scanning of the style exemplar in the real world. This is done by detecting the four corners of the exemplar and performing a perspective transformation to align the sphere with pre-calculated normals.

3.4.1 Corner detection

With the goal being an identification of the four corners of the exemplar template, the creators of StyleBlit [1] have simply surrounded the template with four ArUco [26] markers (ArUco markers are described in subsection 2.7.1), aligning them with the corners. This is shown in figure 3.4. With the knowledge of the marker's real-world size and the order they were generated, the identification of the template's corners is straightforward with one exception. There is a slight gap between the markers and the template's borders, but this can be resolved using the knowledge of the marker's real-world size and the size of the gap.

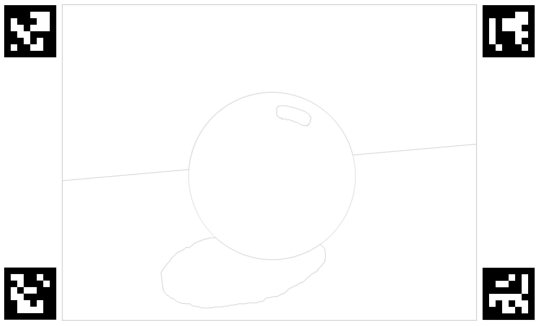


Figure 3.4: StyleTransfer's style exemplar template ArUco markers for easy scanning [1].

The markers used for the style template scanning are from a pre-generated dictionary containing 250 markers with 6×6 dimensions and they are organized in order starting from the top-left and continuing clockwise. While the choice

of the dictionary is not the best for this use case, the advantages of using a better-suited one are negligible.

To simplify things, the StyleTransfer application uses a style template that does not have a gap between the ArUco markers and the style exemplar. This allows it to substitute the appropriate corner of a marker for the corner of the style exemplar, eliminating an inherently inaccurate calculation. The modified template With this modification, the application does not need to know the marker's real-world size beforehand.

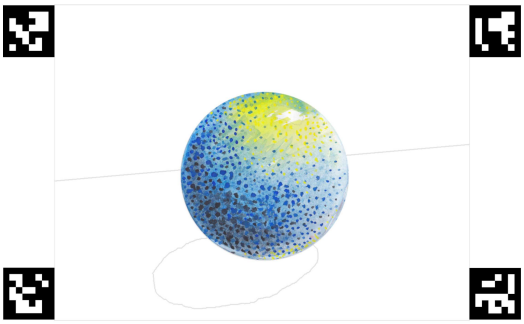


Figure 3.5: StyleTransfer's style exemplar template with pre-filled style and ArUco markers moved to the border of the template.

3.4.2 Scanning flow

The scanning is activated by pressing the plus sign button (see the GUI section 3.3). When active, the application grabs a picture from the camera feed every 10 frames. The picture is then searched for the ArUco markers (see subsection 2.7.1 for more details about ArUco) with the `detectMarkers` [92] function in OpenCV's ArUco module. This function performs marker detection on the input image and returns ids and corners of all detected markers from the specified dictionary. If exactly four markers are detected, they are identified by their ID (0-3) and sorted. Then, a correct corner of each marker is selected. The corners start from 0 in the top-left and increase counting clockwise.

Now that the application knows the aforementioned four points of the style exemplar's corners, it can carry out the transformation. First, the perspective transformation matrix is calculated with the OpenCV's `getPerspectiveTransform` [93] function from the image processing module. The function takes two sets of four points – the first set (source) consists of the detected corners and the second set (destination) can be constructed based on the desired dimensions of the exemplar (600×456 in this case). For more background information about the concept of a projective[5] transformation,

[5]Every perspective transformation is a projective transformation, but a projective transformation is not necessarily a perspective transformation.

Figure 3.6: The scanning flow is demonstrated in three steps from left to right: (1) place an object, (2) click the plus button, and (3) make sure all four markers are visible.

please see subsection 2.3.1.

With the perspective transformation matrix, the application calls the OpenCV's `warpPerspective` [94] function to finally transform the image into a style exemplar. The resulting style exemplar is then assigned to the first button in the middle row of the GUI and the button is selected, effectively turning the scanning off. Please see figure 3.6 to get an idea of the user experience regarding the whole scanning flow.

The scanning can function under unfavorable conditions, but the quality diminishes with distance and camera angle. The distance problem is intuitive – the camera is far away, and thus unable to capture details in the exemplar, causing a smooth appearance. With an increasing angle, the exemplar is scanned at, less information is available for the perspective transformation, resulting in a distorted image. The distance and angle problems are shown in figure 3.7. Lighting also affects the captured style, but this is not an issue of the scanning process.

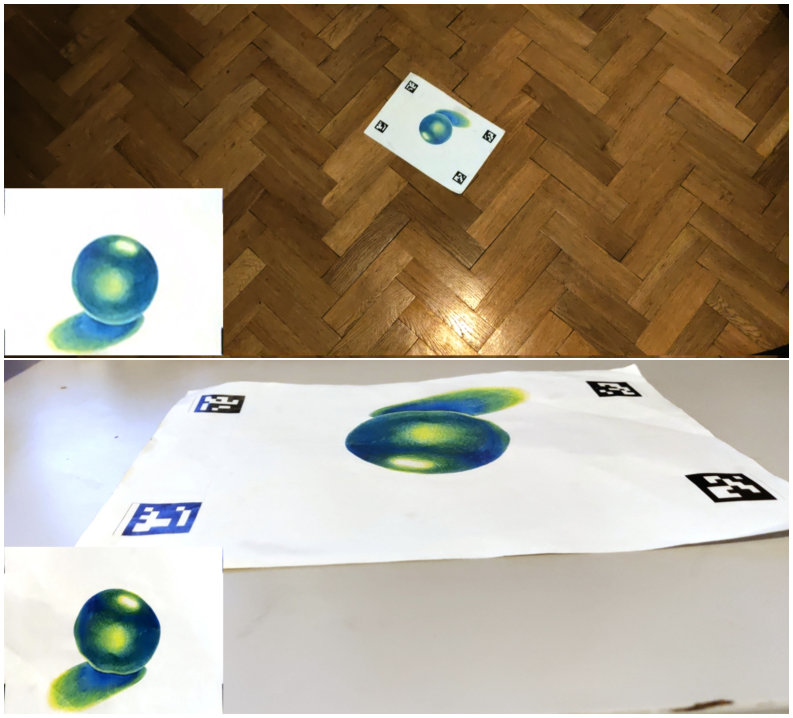


Figure 3.7: Screenshots from the ScannerTest scene (see subsection 3.5.2), showing extreme conditions for the scanner – the camera is far away (top), and the camera is at an angle (bottom).

3.5 Scenes

The project consists of three scenes – StyleTransfer, ScannerTest, and MaterialExample.

3.5.1 StyleTransfer

The StyleTransfer scene is the main scene of the application, providing the AR experience. It contains multiple GameObjects, each serving a different function.

- AR Camera – This object is providing the feed from a device's rear camera as background while also tracking its position in the environment. It also houses the ARMarkerDetector component for exemplar scanning.
- AR Session – The AR Session object controls the lifecycle of an AR experience. It handles enabling of the underlying framework (in this case, the ARCore framework) and reports the state.
- AR Session Origin – Transforms trackable features (planar surfaces and feature points) into the Unity scene. This allows it to handle the pose

and scale of virtual content. The AR Session Origin contains most of the custom scripts described in the Scripts section 3.6.

- Canvas – Contains all GameObjects relating to GUI.
- EventSystem – Allows sending user input events to objects in the application.
- LeanTouch – A GameObject from the Lean Touch package [95]. Handles gestures for scaling and rotating displayed 3D model. Relies on the EventSystem GameObject.

For more information on how this scene functions, please see the Main flow 3.2 and GUI 3.3 sections.

3.5.2 Testing scenes

Figure 3.8: A screenshot from the ScannerTest scene, demonstrating the scanner's functionality. The bottom-left corner shows the result of the perspective transformation.

The ScannerTest scene is a copy of the StyleTransfer scene with most of the functionality disabled and without any GUI. The purpose of this scene is to test the style exemplar acquisition functionality, described in detail in section 3.4. The of the process output is shown in the bottom-left when the device is in landscape orientation. A screenshot from the scene is shown in figure 3.8.

The second testing scene is called MaterialExample and is shown in figure 3.9. This scene applies StyleBlit's algorithm to 3D models in a regular Unity environment that can be run in the Unity player. The 3D models can have arbitrary sizes. This scene can also be used for testing the shader that implements StyleBlit's algorithm. The scene is not designed to be run on mobile devices.

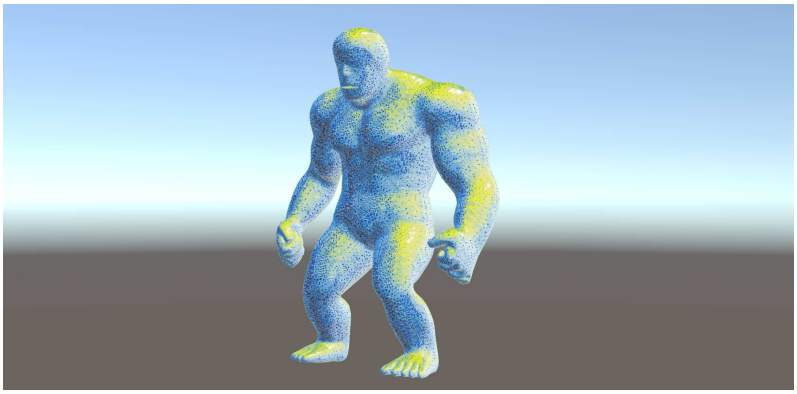


Figure 3.9: A screenshot from the MaterialExample scene, showcasing the StyleBlit's algorithm on a large 3D model.

3.6 Scripts

The application uses several scripts to manage the inner state of the application and allow for custom behavior. Here is their list:

- ARObjectLocationManager – Handles the first placement of a 3D model and their relocation using the place object button.
- ARStyleScanner – This script is responsible for the style exemplar acquisition. It implements the scanning flow described in subsection 3.4.2.
- ChangeSourceStyle – Replaces the input 2D texture for StyleBlit's shader. Called every time a style exemplar selection is changed.
- ChangeStyleToggleBackground – After a new style is acquired, this script is called to change the appearance of the toggle button to the new style.
- CustomStyleManager – Listens to the style scanning toggle button and manages the ARStyleScanner. Handles the result from ARStyleScanner by changing the custom style toggle's appearance and activating it.
- MarkerImageResult – Used only in the ScannerTest scene. Manages the continuous usage of the ARStyleScanner and adapts the result to a displayed image.
- SetupScreenResolution – Halves the screen's resolution at startup to achieve better performance.
- TargetObjectManager – This is where the style exemplar and the 3D model toggle groups meet. The script manages which style exemplar is used in the StyleBlit shader and which 3D model is shown. Also handles the generation of new stylized thumbnails of the 3D models with every

style exemplar change. The 3D models are instantiated at startup to optimize the changeover of 3D models.

- ToggleCircle – A simple script that shows or hides the green circle around a toggle button based on its state.

3.7 Used libraries

Unity's AR Foundation framework [66] was used as the basis for the AR experience. All 3D models and style exemplars found in the project were supplied by the creators of StyleBlit [1]. The application also uses several third-party packages. The packages and their usages are:

- StyleBlit Unity plugin [96] – An implementation of the StyleBlit's algorithm in Unity.
- OpenCV for Unity [97] – For using OpenCV within Unity.
- Lean Touch [95] – A library that handles touch controls of 3D objects (scale and rotation).
- Runtime Preview Generator [98] – For generating thumbnails of stylized 3D objects.

The 3rd party packages were separated from other assets by placing them in the Plugins folder.

3.8 Shader

The application utilizes a shader that implements the StyleBlit algorithm. The shader is part of the StyleBlit Unity plugin [96]. This section describes the shader's implementation and relies on the general shaders description in subsection 2.6.10.

The shader uses nine properties with the first three being adjustable in the editor:

- `_threshold` – A threshold value for the allowed error when assigning a pixel to a chunk. This affects the overall fragmentation of the 3D model.
- `_votespan` – A measure of smoothness. The shader blends the patches to hide seams and noise. The bigger the votespan, the more pixels contribute to the resulting pixel, making the output texture look smoother.
- `sourceStyle` – The input style exemplar. This property changes with every style exemplar selection change by the user.

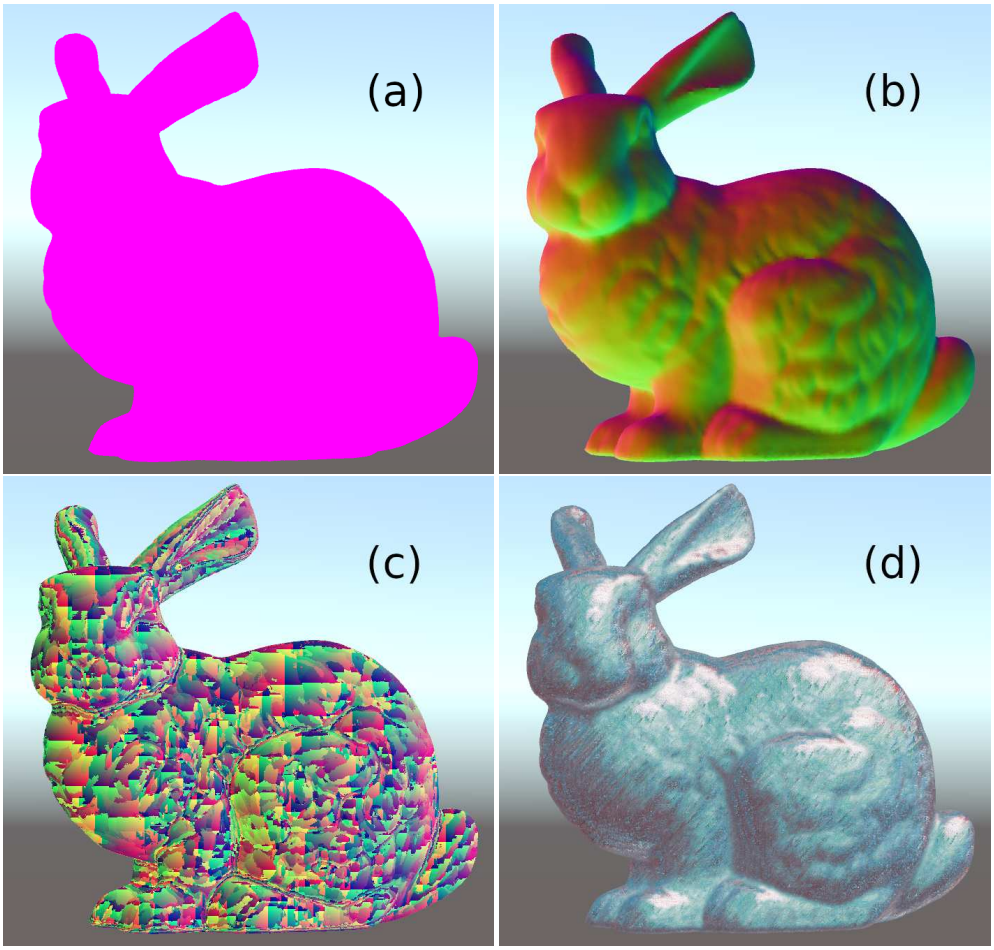


Figure 3.10: Screenshots from the MaterialExample scene, showcasing individual passes of the StyleBlit's shader on a large 3D model, where (a) is an input to the shader, (b) is the model with normal texture, (c) output from StyleBlit's algorithm as coordinates, and (d) is the output from a full render.

- `_splatsize` – The number of levels in the hierarchy of seeds.
- `_splitCo` – Used for transferring data from the second to third pass.
- `_jitterCo` – A coefficient for the seed distribution. The nearest seed search is randomly offset using a value from the noise texture multiplied by this coefficient.
- `normalToSourceLUT` – A lookup texture mapping normals to coordinates from the source style. Used to accelerate retrieving the source texture pixel.

- `sourceNormals` – A normal map of the style exemplar template scene. Only the sphere from the scene is used for mapping in this implementation.
- `noiseTexture` – A texture with white noise. Used for randomly distributing the seeds.

The shader has one subshader. The subshader consists of three passes with two grab passes in between. A grab pass is a special type of pass that takes the contents of a frame buffer and converts them to a texure [99].

The first pass is very similar to the one described in the Shader example subsection 2.6.11. It computes a normal for the vertex and passes it on as a color[6]. The grab pass then collects the colors, resulting in a normal texture.

The second pass is the primary pass, implementing StyleBlit's parallel algorithm. The parallel algorithm is detailed in subsection 2.2.3. The shader searches for the nearest seed through multiple levels until the guidance error is below the defined threshold. If such a seed is found, the current pixel can use the seed to copy the corresponding source pixel's coordinates into the target place. The output of this shader is a texture that encodes the coordinates into color.

The last pass, with coordinates from the style exemplar as input supplied by a grab pass, reconstructs the coordinates from color and performs smoothing by voting. It averages the pixel values in their respective neighborhoods. This blends the seams and suppresses noise pixels.

The output from individual passes is shown in figure 3.10 on the bunny 3D model. The first render (a) is what comes to the shader as input. The second image (b) is just a normal texture of the model from the first pass. The third picture (c) is showing the output from the first and second passes. This is the output from StyleBlit's algorithm with encoded target coordinates as colors, which makes the individual chunks identifiable. The last image (d) shows the output of all three passes.

3.9 Performance

The application is limited to the inherited framerate of the device's camera (generally 30) and achieves this figure when there is no virtual content being rendered. The performance of the application strongly correlates with the number of pixels the stylization shader has to process. This is why the application runs at half of the maximum resolution by design.

The application was developed and tested mainly on the OnePlus Nord and Google Pixel 3a XL mobile phones, which are not flagship-level devices by any means. Figure 3.11 demonstrates how the framerate behaves under different conditions.

[6]The process of converting a normal to color (in Unity) is this: the XYZ directions of the vector are mapped to RGB by first halving them and then adding 0.5 [100].

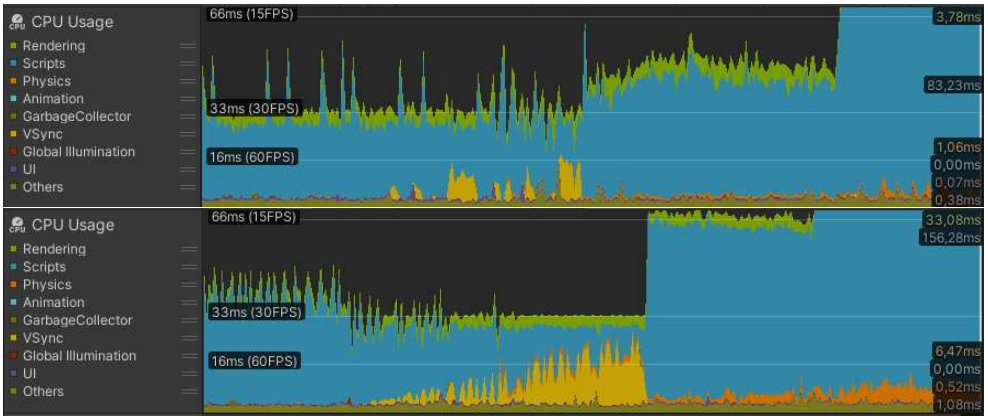


Figure 3.11: Performance graphs of the OnePlus Nord (top) and Google Pixel 3a XL (bottom) devices from the Unity Profiler [28]. The graphs show the application going through a simple scenario: detect surface, place 3D model, and enlarge the 3D model so that it takes up most of the screen. The individual steps are clearly noticeable in the graphs as they cause a significant drop in the frames per second (FPS) measure (the measure increases from top to bottom in the graph). The OnePlus device performs measurably better, but both devices dip into single-digit framerate in the worst-case scenario. Note: the graph is labeled as CPU Usage. This is because Unity reports GPU usage as CPU usage in mobile devices.

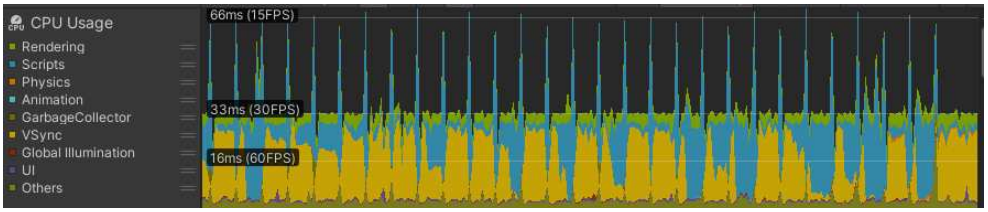


Figure 3.12: A performance graph illustrating the performance impact of active style exemplar scanning. No 3D model is being rendered in this scenario. After a successful scan, the subsequent transformation of the style exemplar and thumbnail generation of the 3D models causes no measurable effect in terms of performance. The scanning is successful at the last performance spike and it is indistinguishable from the previous spikes. Measured on the OnePlus Nord device. Note: the graph is labeled as CPU Usage. This is because Unity reports GPU usage as CPU usage in mobile devices.

Another action impeding performance is the style exemplar scanning. The script acquires a full resolution image from the camera every ten frames to try and recognize the style exemplar template, which causes visible spikes in performance. This behavior is shown in figure 3.12.

3.10 Limitations

Outside of the computational requirements, the main limitations are inherited either from the StyleBlit algorithm or from using AR Foundation.

When targeting Android while using AR Foundation, the ARCore supporting services need to be installed. However, to be able to install those services, Google needs to certify the specific device first. The market share of all certified devices currently in use is estimated to be at 41 % [62].

There are a few limitations of the StyleBlit method. The first one is that the method is unable to keep a regular texture, for example, a brick wall, which produces visible misalignments. Another issue arises when the target model contains large planar areas. This causes repetition of patches, producing "scales". Both of these issues are shown in figure 3.13. However, these issues are considered edge-cases and generally do not occur in the application.

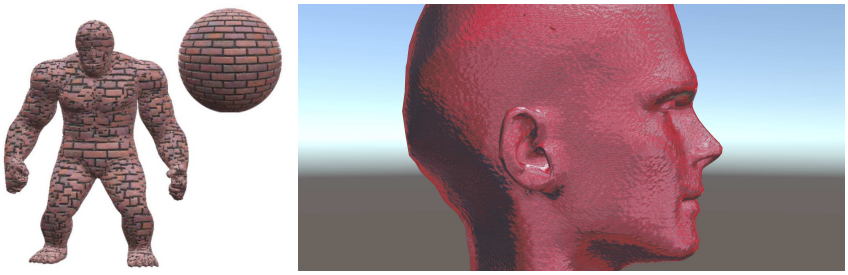
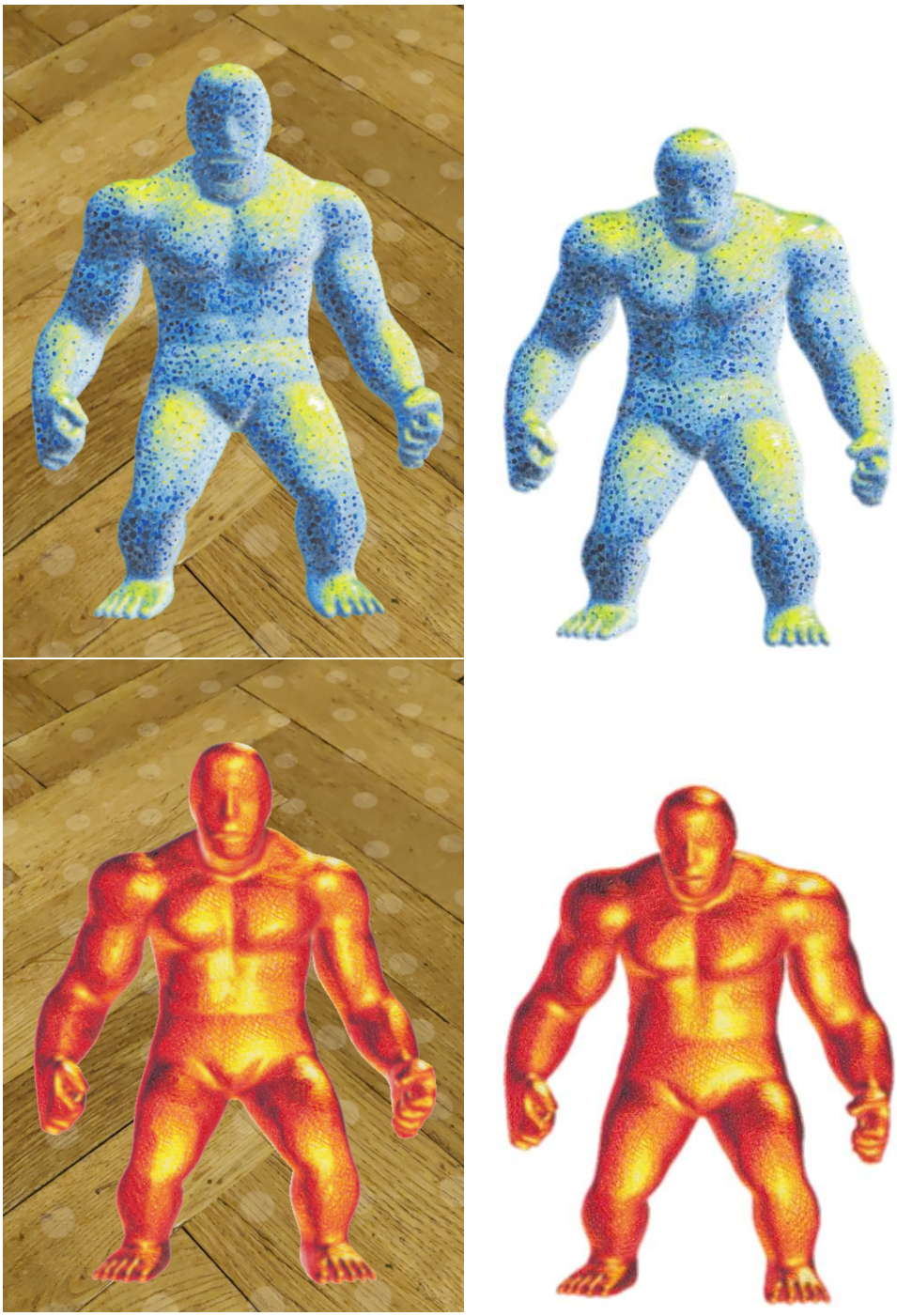


Figure 3.13: Known limitations of the StyleBlit [1] method. The first image (left) shows the inability to reproduce a texture that contains structure. The second image (right) shows the repetition of patches when there is little surface variation.

Limitations of the scanning process are described in section **??**,

3.11 Stylization comparison

As the application implements StyleBlit's algorithm, it is reasonable to compare the output with the original StyleBlit. The golem 3D model along with several different styles was chosen for this comparison. Please see figure 3.14 and its subfigures for the comparison. Even though their pose is not exact and the algorithm is stochastic by design, we can see that both algorithms behave similarly and their outputs are comparable. One noticeable difference is that the transition between the model and background is smooth in the original StyleBlit, but the same transition is abrupt in the application. This is caused by the shader being supplied only with the pixels for the target model.



(a) Application output (left) in comparison to the original StyleBlit [1] (right).

(b) Application output (left) in comparison to the original StyleBlit [1] (right).

(c) Application output (left) in comparison to the original StyleBlit [1] (right).

(d) Application output (left) in comparison to the original StyleBlit [1] (right).

Figure 3.14: This figure compares seven different styles rendered by the application with the original StyleBlit [1] algorithm. The poses are not exactly aligned but are sufficient for comparison.

Conclusion

This thesis has provided a broad overview of the computer-assisted stylization field, describing four different branches along with their development. The focus then shifted to a number of contextual topics, all of which were needed to fully comprehend and implement the project. Among these topics were two patch-based stylization methods, the math behind perspective transformation, detection of the style exemplar, augmented reality – both its place in the XR spectrum and implementations, the Unity game engine, and the OpenCV library.

With this background knowledge, the StyleTransfer application was developed in Unity game engine with the AR Foundation framework. Until now, there was no application combining stylization of an interactable 3D model with style exemplar acquisition within the augmented reality domain. The application is able to perform all these actions in real-time. The components of the application were described in detail, including the typical usage, GUI, and exemplar scanning. Vital parts of the Unity project were also recounted, including a shader that implements the StyleBlit algorithm. Finally, the thesis evaluates the performance and compares the output from the application to the original StyleBlit, while also mentioning its limitations. With the resulting application, the thesis has reached its goals.

Further work

While the thesis has reached its target, there is always room for improvement. A substantial improvement to the project would be an iOS application, leveraging the AR Foundation framework's multi-platform capability. While this was one of the motivations for using AR Foundation, it requires a hand-crafted shader, because it uses the Metal API (instead of OpenGL) for GPU-based computation. A web-based version could also be viable.

Another thing that could be improved is the 3D model manipulation within

the scene. Currently, the user has to press a button to relocate the 3D model. A more intuitive way would be to drag the model manually, but this was interfering with the rotate and scale gestures.

One last area that would benefit from improvements is performance. While the application' performance is adequate, implementing the concept of adaptive resolution would allow it to keep high framerates even when rendering large-scale objects.

Bibliography

[1] Sýkora, D.; Jamriška, O.; et al. StyleBlit: Fast Example-Based Stylization with Local Guidance. *Computer Graphics Forum*, volume 38, no. 2, 2019: pp. 83–91.

[2] Haeberli, P. Paint by Numbers: Abstract Image Representations. *SIGGRAPH Computer Graphics*, volume 24, no. 4, 1990: pp. 207–214.

[3] Litwinowicz, P. Processing Images and Video for an Impressionist Effect. In *SIGGRAPH*, 1997, pp. 407–414.

[4] Hertzmann, A. Paint By Relaxation. In *Proceedings of Computer Graphics International*, 2001, pp. 47–54.

[5] Winnemöller, H.; Kyprianidis, J. E.; et al. XDoG: An eXtended Difference-of-Gaussians Compendium Including Advanced Image Stylization. *Computers & Graphics*, volume 36, no. 6, 2012: pp. 740–753.

[6] Lu, C.; Xu, L.; et al. Combining sketch and tone for pencil drawing production. In *Proceedings of International Symposium on Non-Photorealistic Animation and Rendering*, 2012, pp. 65–73.

[7] Hertzmann, A.; Jacobs, C. E.; et al. Image Analogies. In *SIGGRAPH Conference Proceedings*, 2001, pp. 327–340.

[8] Sloan, P.-P. J.; Martin, W.; et al. The Lit Sphere: A Model for Capturing NPR Shading from Art. In *Proceedings of Graphics Interface*, 2001, pp. 143–150.

[9] Bénard, P.; Cole, F.; et al. Stylizing Animation By Example. *ACM Transactions on Graphics*, volume 32, no. 4, 2013: p. 119.

[10] Hauptfleisch, F.; Texler, O.; et al. StyleProp: Real-time Example-based Stylization of 3D Models. *Computer Graphics Forum*, volume 39, no. 7, 2020: pp. 575–586.

[11] Futschik, D.; Kučera, M.; et al. STALP: Style Transfer with Auxiliary Limited Pairing. *Computer Graphics Forum*, volume 40, no. 2, 2021: pp. 563–573.

[12] Gatys, L. A.; Ecker, A. S.; et al. A Neural Algorithm of Artistic Style. *CoRR*, volume abs/1508.06576, 2015.

[13] Kolkin, N. I.; Salavon, J.; et al. Style Transfer by Relaxed Optimal Transport and Self-Similarity. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10051–10060.

[14] Fišer, J.; Jamriška, O.; et al. StyLit: Illumination-Guided Example-Based Stylization of 3D Renderings. *ACM Transactions on Graphics*, volume 35, no. 4, 2016: p. 92.

[15] Cáceres, E.; Carrasco, M.; et al. Evaluation of an eye-pointer interaction device for human-computer interaction. *Heliyon*, volume 4, 2018.

[16] The-Crankshaft Publishing. Geometric Transformations (Introduction to Video and Image Processing) Part 1. [online], [cit. 26.12.2021]. Available from: `http://what-when-how.com/introduction-to-video-and-image-processing/geometric-transformations-introduction-to-video-and-image-processing-part-1/`

[17] Daniel Sýkora. Image Deformation. [online], [cit. 26.12.2021]. Available from: `https://dcgi.fel.cvut.cz/home/sykorad/dzo/slides/dzo-l07.pdf`

[18] Garrido-Jurado, S.; Muñoz-Salinas, R.; et al. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, volume 47, 2014: pp. 2280–2292.

[19] Elizabeth Robinson. Microsoft outlines three trends that will impact mixed reality in 2018. [online], [cit. 14.12.2021]. Available from: `https://www.technologyrecord.com/Article/microsoft-outlines-three-trends-that-will-impact-mixed-reality-in-2018-63570`

[20] Microsoft. What is mixed reality? [online], [cit. 15.12.2021]. Available from: `https://docs.microsoft.com/en-us/windows/mixed-reality/discover/mixed-reality`

[21] El Filali, Y.; Salah-ddine, K. Augmented Reality Types and Popular Use Cases. 2018, pp. 107–110.

[22] IKEA. IKEA Place. [online], [cit. 6.12.2021]. Available from: `https://www.ikea.com/au/en/customer-service/mobile-apps/say-hej-to-ikea-place-pub1f8af050`

[23] Instagram Business Team. Introducing Face Filters and More on Instagram. [online], [cit. 13.12.2021]. Available from: https://www.facebook.com/business/news/instagram/introducing-face-filters

[24] Pascal Bregeon. Here Comes a $400,000 Augmented Reality Helmet for Fighter Jet. [online], [cit. 14.12.2021]. Available from: http://www.augmentedrealitytrends.com/augmented-reality/helmet-for-fighter-jet

[25] Unity Technologies. Order of execution for event functions. [online], [cit. 28.12.2021]. Available from: https://docs.unity3d.com/Manual/ExecutionOrder.html

[26] OpenCV team. Detection of ArUco Markers. [online], [cit. 27.12.2021]. Available from: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html

[27] Balsamiq Studios, LLC. Quick and Easy Wireframing Tool. [online], [cit. 2.1.2022]. Available from: https://balsamiq.com/wireframes/

[28] Unity Technologies. Profiler overview. [online], [cit. 4.1.2022]. Available from: https://docs.unity3d.com/Manual/Profiler.html

[29] Unity Technologies. About AR Foundation. [online], [cit. 19.12.2021]. Available from: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.2/manual/index.html

[30] Lambert, N.; Latham, W.; et al. The Emergence and Growth of Evolutionary Art — 1980—1993. *Leonardo*, volume 46, 2013: pp. 367–375.

[31] Magnenat, S.; Ngo, D. T.; et al. Live Texturing of Augmented Reality Characters from Colored Drawings. *IEEE Transactions on Visualization and Computer Graphics*, volume 21, no. 11, 2015: pp. 1201–1210.

[32] Kyprianidis, J. E.; Collomosse, J.; et al. State of the "Art": A Taxonomy of Artistic Stylization Techniques for Images and Video. *IEEE Transactions on Visualization and Computer Graphics*, volume 19, no. 5, 2013: pp. 866–885.

[33] Salisbury, M. P.; Wong, M. T.; et al. Orientable Textures for Image-based Pen-and-ink Illustration. In *SIGGRAPH Conference Proceedings*, 1997, pp. 401–406.

[34] Snavely, N.; Zitnick, L.; et al. Stylizing 2.5-D video. In *SIGGRAPH Sketches*, edited by J. Buhler, 2005, p. 94.

[35] Breslav, S.; Szerszen, K.; et al. Dynamic 2D patterns for shading 3D scenes. *ACM Transactions on Graphics*, volume 26, no. 3, 2007: p. 20.

[36] Hays, J.; Essa, I. A. Image and Video Based Painterly Animation. In *Proceedings of International Symposium on Non-Photorealistic Animation and Rendering*, 2004, pp. 113–120.

[37] Zhao, M.; Zhu, S.-C. Portrait Painting Using Active Templates. In *Proceedings of International Symposium on Non-Photorealistic Animation and Rendering*, 2011, pp. 117–124.

[38] Gedraite, E. S.; Hadad, M. Investigation on the effect of a Gaussian Blur in image filtering and segmentation. In *Proceedings ELMAR-2011*, 2011, pp. 393–396.

[39] Assirati, L.; Rosa, N.; et al. Performing edge detection by difference of Gaussians using q-Gaussian kernels. *Journal of Physics Conference Series*, volume 490, 2013.

[40] Blinn, J. F.; Newell, M. E. Texture and Reflection in Computer Generated Images. *Communications of the ACM*, volume 19, no. 10, 1976: pp. 542–547.

[41] Barnes, C.; Shechtman, E.; et al. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics*, volume 28, no. 3, 2009: p. 24.

[42] Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, volume abs/1409.1556, 2014.

[43] Liao, J.; Yao, Y.; et al. Visual Attribute Transfer Through Deep Image Analogy. *ACM Transactions on Graphics*, volume 36, no. 4, 2017: p. 120.

[44] Goodfellow, I.; Pouget-Abadie, J.; et al. Generative Adversarial Networks. *Advances in Neural Information Processing Systems*, volume 3, 2014.

[45] Isola, P.; Zhu, J.-Y.; et al. Image-to-Image Translation with Conditional Adversarial Networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5967–5976.

[46] Tulyakov, S.; Liu, M.-Y.; et al. MoCoGAN: Decomposing Motion and Content for Video Generation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1526–1535.

[47] Huang, X.; Belongie, S. J. Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization. *Proceedings of IEEE International Conference on Computer Vision*, 2017: pp. 1510–1519.

[48] Heckbert, P. S. Adaptive Radiosity Textures for Bidirectional Ray Tracing. *SIGGRAPH Computer Graphics*, volume 24, no. 4, 1990: pp. 145–154.

[49] Rosenberger, A.; Cohen-Or, D.; et al. Layered Shape Synthesis: Automatic generation of control maps for non-stationary textures. *ACM Transactions on Graphics*, volume 28, no. 5, 2009: p. 107.

[50] Mastyło, M. Bilinear interpolation theorems and applications. *Journal of Functional Analysis*, volume 265, 2013: pp. 185–207.

[51] Lowe, D. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, 1999, pp. 1150–1157, doi:10.1109/"ProceedingsofIEEEInternationalConferenceonComputerVision".1999.790410.

[52] Bay, H.; Ess, A.; et al. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, volume 110, 2008: pp. 346–359.

[53] Alcantarilla, P. F.; Bartoli, A.; et al. KAZE Features. In *ECCV (6)*, volume 7577, edited by A. W. Fitzgibbon; S. Lazebnik; P. Perona; Y. Sato; C. Schmid, 2012, pp. 214–227.

[54] Leutenegger, S.; Chli, M.; et al. BRISK: Binary Robust invariant scalable keypoints. 2011, pp. 2548–2555.

[55] Radzi, F.; Khalil-Hani, M.; et al. Generalizing convolutional neural networks for pattern recognition tasks. *ARPN Journal of Engineering and Applied Sciences*, volume 10, 2015: pp. 5298–5308.

[56] Girshick, R.; Donahue, J.; et al. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587.

[57] Liu, W.; Anguelov, D.; et al. SSD: Single Shot MultiBox Detector. *Lecture Notes in Computer Science*, 2016: pp. 21–37.

[58] Redmon, Joseph and Divvala, Santosh and Girshick, Ross and Farhadi, Ali. You Only Look Once: Unified, Real-Time Object Detection. 2015. Available from: `http://arxiv.org/abs/1506.02640`

[59] Google LLC. Overview of ARCore and supported development environments. [online], [cit. 17.12.2021]. Available from: `https://developers.google.com/ar/develop`

[60] Google LLC. ARCore SDK for Android. [online], [cit. 16.12.2021]. Available from: `https://github.com/google-ar/arcore-android-sdk`

[61] Google LLC. Introduction to Cloud Anchors on Android. [online], [cit. 17.12.2021]. Available from: https://developers.google.com/ar/develop/java/cloud-anchors/introduction

[62] Richard Révész. How many AR supported phones are in the world? – 2020. [online], [cit. 17.12.2021]. Available from: https://www.aronplatform.com/mobile-ar-penetration-2020/

[63] Google LLC. ARCore supported devices. [online], [cit. 17.12.2021]. Available from: https://developers.google.com/ar/devices

[64] Apple Inc. More to Explore with ARKit 5. [online], [cit. 18.12.2021]. Available from: https://developer.apple.com/augmented-reality/arkit/

[65] Apple Inc. RealityKit. [online], [cit. 19.12.2021]. Available from: https://developer.apple.com/augmented-reality/realitykit/

[66] Unity Technologies. AR Foundation. [online], [cit. 19.12.2021]. Available from: https://unity.com/unity/features/arfoundation

[67] John K Haas. A History of the Unity Game Engine. 2014.

[68] Unity Technologies. 2021 Gaming Report. [online], [cit. 27.12.2021]. Available from: https://create.unity3d.com/2021-game-report

[69] Unity Technologies. Editor Features. [online], [cit. 27.12.2021]. Available from: https://docs.unity3d.com/Manual/EditorFeatures.html

[70] Unity Technologies. Plans and pricing. [online], [cit. 27.12.2021]. Available from: https://store.unity.com

[71] Unity Technologies. System requirements for Unity 2020 LTS. [online], [cit. 27.12.2021]. Available from: https://docs.unity3d.com/Manual/system-requirements.html

[72] Unity Technologies. Installing the Unity Hub. [online], [cit. 27.12.2021]. Available from: https://docs.unity3d.com/Manual/GettingStartedInstallingHub.html

[73] Unity Technologies. Unity download archive. [online], [cit. 27.12.2021]. Available from: https://unity3d.com/get-unity/download/archive

[74] Unity Technologies. Integrated development environment (IDE) support. [online], [cit. 27.12.2021]. Available from: https://docs.unity3d.com/Manual/ScriptingToolsIDEs.html

[75] Unity Technologies. Scripting. [online], [cit. 27.12.2021]. Available from: https://docs.unity3d.com/Manual/ScriptingSection.html

[76] Unity Technologies. Asset Workflow. [online], [cit. 28.12.2021]. Available from: https://docs.unity3d.com/Manual/AssetWorkflow.html

[77] Unity Technologies. Unity Asset Store. [online], [cit. 28.12.2021]. Available from: https://assetstore.unity.com/

[78] Unity Technologies. Shaders introduction. [online], [cit. 29.12.2021]. Available from: https://docs.unity3d.com/Manual/shader-introduction.html

[79] Unity Technologies. The Shader class. [online], [cit. 29.12.2021]. Available from: https://docs.unity3d.com/Manual/shader-objects.html

[80] Microsoft Corporation. High-level shader language (HLSL). [online], [cit. 29.12.2021]. Available from: https://docs.microsoft.com/en-us/windows/win32/direct3dhlsl/dx-graphics-hlsl

[81] NVIDIA Corp. Cg Language Specification. [online], [cit. 29.12.2021]. Available from: https://developer.download.nvidia.com/cg/Cg_language.html

[82] Unity Technologies. Material Inspector reference. [online], [cit. 30.12.2021]. Available from: https://docs.unity3d.com/Manual/class-Material.html

[83] Unity Technologies. Custom shader fundamentals. [online], [cit. 29.12.2021]. Available from: https://docs.unity3d.com/Manual/SL-VertexFragmentShaderExamples.html

[84] OpenCV team. OpenCV: Home. [online], [cit. 27.12.2021]. Available from: https://opencv.org/

[85] OpenCV team. OpenCV license file. [online], [cit. 30.12.2021]. Available from: https://github.com/opencv/opencv/blob/4.x/LICENSE

[86] OpenCV team. About. [online], [cit. 30.12.2021]. Available from: https://opencv.org/about/

[87] Slashdot Media. Source Forge Download Statistics. [online], [cit. 30.12.2021]. Available from: https://sourceforge.net/projects/opencvlibrary/files/stats/timeline?dates=2001-03-01%20to%202021-12-01&period=monthly

[88] OpenCV team. Repository for OpenCV's extra modules. [online], [cit. 30.12.2021]. Available from: https://github.com/opencv/opencv_contrib

[89] Bookstein, A.; Kulyukin, V. A.; et al. Generalized Hamming Distance. *Inf. Retr.*, volume 5, 2002: pp. 353–375.

[90] StatCounter. Mobile & Tablet Android Version Market Share Worldwide. [online], [cit. 31.12.2021]. Available from: https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide/#monthly-202111-202111-bar

[91] Google LLC. Gestures. [online], [cit. 2.1.2022]. Available from: https://material.io/design/interaction/gestures.html

[92] OpenCV team. ArUco Marker Detection documentation. [online], [cit. 2.1.2022]. Available from: https://docs.opencv.org/4.x/d9/d6a/group__aruco.html#gab9159aa69250d8d3642593e508cb6baa

[93] OpenCV team. Geometric Image Transformations documentation: getPerspectiveTransform. [online], [cit. 2.1.2022]. Available from: https://docs.opencv.org/4.x/da/d54/group__imgproc__transform.html#ga20f62aa3235d869c9956436c870893ae

[94] OpenCV team. Geometric Image Transformations documentation: warpPerspective. [online], [cit. 2.1.2022]. Available from: https://docs.opencv.org/4.x/da/d54/group__imgproc__transform.html#gaf73673a7e8e18ec6963e3774e6a94b87

[95] Carlos Wilkes. Lean Touch. [online], [cit. 31.12.2021]. Available from: https://assetstore.unity.com/packages/tools/input-management/lean-touch-30111

[96] Adam Pospíšil. StyleBlit Unity Plugin. [online], [cit. 31.12.2021]. Available from: https://github.com/AdamPospisil/StyleBlitUnity

[97] Enox Software. OpenCV for Unity. [online], [cit. 31.12.2021]. Available from: https://assetstore.unity.com/packages/tools/integration/opencv-for-unity-21088

[98] yasirkula. Runtime Preview Generator. [online], [cit. 31.12.2021]. Available from: https://assetstore.unity.com/packages/tools/camera/runtime-preview-generator-112860

[99] Unity Technologies. ShaderLab command: GrabPass. [online], [cit. 3.1.2022]. Available from: https://docs.unity3d.com/Manual/SL-GrabPass.html

[100] Unity Technologies. Normal map (Bump mapping). [online], [cit. 3.1.2022]. Available from: https://docs.unity3d.com/Manual/StandardShaderMaterialParameterNormalMap.html

Appendix A

Acronyms

LPE Light Path Expressions

NNF Nearest Neighbor Field

ANN Approximate Nearest Neighbor

CPU Central Processing Unit

GPU Graphics Processing Unit

SDK Software Development Kit

APK Android Application Package

CNN Convolutional Neural Network

DCNN Deep Convolutional Neural Network

GANN Generative Adversarial Neural Network

IDE Integrated Development Environment

HLSL High-Level Shader Language

API Application Programming Interface

GUI Graphical User Interface

FPS Frames Per Second

Appendix B

Contents of enclosed CD

- `apk` ..... the directory with executables
- `src` ..... the directory of source codes
  - `unity` ..... implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
  - `thesis.ps` ..... the thesis text in PS format

Appendix C

Installation manual

To install the attached APKs, please follow these steps:

1. Connect the Android device to your PC.
2. Enable file transfer mode. The mode is usually displayed as a persistent notification.
3. Copy the desired APK(s) to your Android device.
4. Open a file manager application on the Android device.
5. Find the desired APK and open it.
6. Confirm the warning dialogs. Wait until the installation finishes.
7. Find the application in the list of all applications.