MSc thesis in Geomatics

Neural Surface Reconstruction and Stylization

Fabian Visser
2023

MSc thesis in Geomatics

Neural Surface Reconstruction and Stylization

Fabian Visser

June 2023

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of Master of Science in Geomatics

Fabian Visser: *Neural Surface Reconstruction and Stylization* (2023)
ⓒⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.
To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/.

The work in this thesis was carried out in the:

3D geoinformation group
Delft University of Technology

Supervisors: Nail Ibrahimli
Dr. Liangliang Nan
Co-reader: Lukas Uzolas

Abstract

Style transfer is a recent field in the development of deep neural networks, which allows for the style from one image to be transferred onto another image. This has been well-researched for 2D images, but transferring style onto 3D reconstructed content can still be further developed. Being able to style a 3D reconstruction would allow users to recreate anything in the real world, such as a chair, with any style they see fit. Where other methods use texture-based approaches which often create low quality geometry and appearance, or use radiance fields which style a whole scene instead of just the 3D reconstructed object, we have developed a method which styles an implicit surface.

We achieve this by using Implicit Differentiable Renderer (IDR), which trains, using masked images as input, two neural networks that learn the geometry and appearance. Rendered views of the object are styled using 2D neural style transfer (NST) methods, and the style information is used to further train the appearance network to display the given style. With *Masked deferred back-propagation* we are able to optimize the appearance renderer, which is normally trained on only patches of the rendered image to save memory, while using style transfers designed for full-resolution images.

We showcase different results from our method using different 3D reconstruction datasets and style images, and showcase how to implement a user-created dataset. We carry out extensive tests on what effects different parameters have on the final result. Comparing our results to similar 3D style methods demonstrates that our method performs equally well in achieving faithful style transfer, while having the benefits of creating high quality geometry and only styling the reconstructed surface.

Acknowledgements

I want to thank my family for being supportive throughout the whole process, and pushing me to make the most of this thesis. Thank you to all the Geolab friends who kept me sane during the long workdays, and entertained me during the evenings at the Bouwpub. Also to the friends who were my escape in between the periods working.

A sincere thanks to Nail, who introduced me to this beautiful yet brutal thesis topic. You always believed in my success, even when I felt completely lost. Without our meetings I would've been nowhere. The free coffee was also an appreciated gesture. Thank you to Liangliang for being always critical and brainstorming new ideas for me to explore. Thank you to Lukas and Azarakhsh for being present and interested throughout the thesis process.

Contents

List of Figures

Acronyms

IDR	Implicit Differentiable Renderer	1
SDF	signed distance function	5
BRDF	bidirectional reflectance distribution function	6
CNN	convolutional neural network	2
NST	Neural Style Transfer	2
NeRF	Neural Radiance Fields	17
ARF	Artistic Radiance Fields	2
NNFM	Nearest Neighbor Feature Matching	3
MSE	mean squared error	12

1. Introduction

1.1. Motivation

Two interesting fields in machine learning right now are the development of differentiable 3D reconstruction and stylistic rendering. Using deep neural networks it is possible to reconstruct from a collection of images an object as a 3D model with the correct appearance. It is also possible to transfer style, as in the artistic elements applied by certain artists, from one image to another image, using deep learning. A user is then able to create a new art piece of any subject containing any style they want. The next step is to expand this to the third dimension, having 3D reconstructed models styled based on a given style image. We explore the possibilities of this by styling a neural network representing appearance which we create using Implicit Differentiable Renderer (IDR).

IDR recreates the 3D geometry, color and reflectance of an object, requiring only masked images as input. The 3D reconstructed object is represented by an implicit surface, instead of the commonly used pointcloud and triangle mesh representations. The benefit of implicit surfaces are the fixed memory footprint and no need for discretization [Niemeyer et al., 2020]. This surface is differentiable, which implies that it can be trained using a neural network to learn the correct shape and appearance [Loper and Black, 2014]. IDR further introduces an appearance renderer, which is able to store a faithful representation of the surface's color and reflectance. Instead of representing the original object's appearance, we want to instead have it represent a style based off of some input style image, using stylistic rendering.

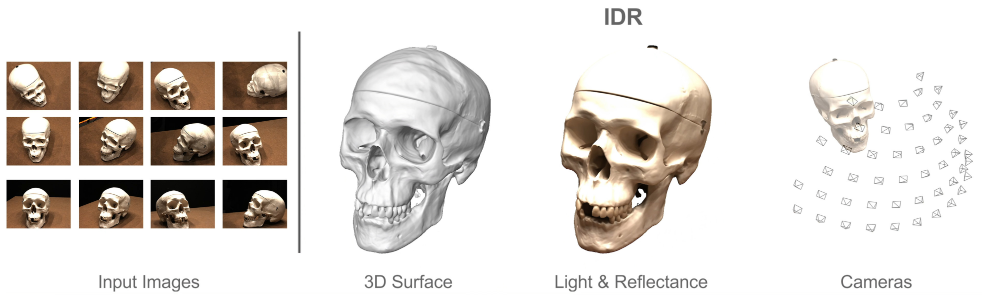


Figure 1.1.: From a set of input images, IDR is able to reconstruct the geometry, as well as it's appearance and camera information. Figure taken from [Yariv et al., 2020].

Stylistic rendering, first popularized by Neural Style Transfer (NST) Gatys et al. [2015a], allows for the ability to transfer style, which can be described as the artistic characteristics of an artwork, to an image. Style transfer is achieved by extracting features from different layers of a convolutional neural network (CNN), and uses those features to evoke the style from a given style image onto a different image, as shown in Figure 1.2.

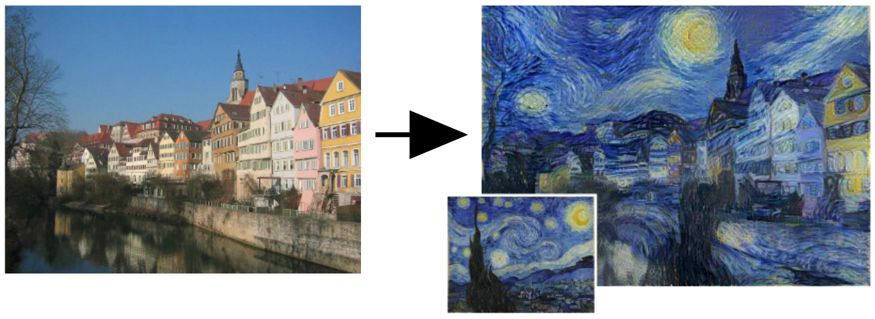


Figure 1.2.: The Neckarfront in Tübingen, Germany, stylized using van Gogh's The Starry Night. Figure taken from [Gatys et al., 2015a].

The power of style transfer is that we are able to apply a variety of styles onto different images with minimal effort. Users are able to easily create new works of art inspired by any artist. The next development is to extend this into the third dimension, which can be useful in developing 3D printed furniture based on 3D reconstructed and styled objects.

Joris Laarman is a Dutch designer focused on creating 3D-printed furniture. His experiments with making aesthetic pieces of furniture lead to the development of the Makerchairs, designs that are publicly available and easily replicable with consumer-grade 3D printers [High Museum of Art, 2018]. As 3D color printers become larger and more affordable [Peels, 2021], a future where consumers are able to not only create, but also *recreate* furniture from the real world, with a chosen style applied to it, becomes ever more realizable. What is required is a method to transfer style from a 2D image to a reconstructed 3D object.

While style transfer for 2D images is well-developed, extending this to 3D introduces new challenges. The styling effect should be applied in a logical fashion for the given geometry, and be consistent across different views. Previous methods have relied on styling a texture, which require storing the geometry as a mesh, limiting the final result to the quality of the geometry. Using neural networks and implicit surfaces avoids this problem. Zhang et al. [2022] achieved this with Artistic Radiance Fields (ARF), which styles a neural radiance field, a function able to create new views of a scene based on a set of input images. The whole scene is however styled, which is undesirable when only a specific object, for example a piece of furniture or a building, are the expected outputs from style transfer. By instead styling the implicit surface from IDR, we are able limit our style transfer to a faithful reconstruction of the desired object.

We explore the possibility of fusing IDR and NST, such that we can create a 3D reconstruction from a collection of 2D masked images, as well as further train the learnt appearance to have the style transferred from some style image. We want to ensure that no matter the view direction when rendering an image of the styled object, the style will remain unchanged.

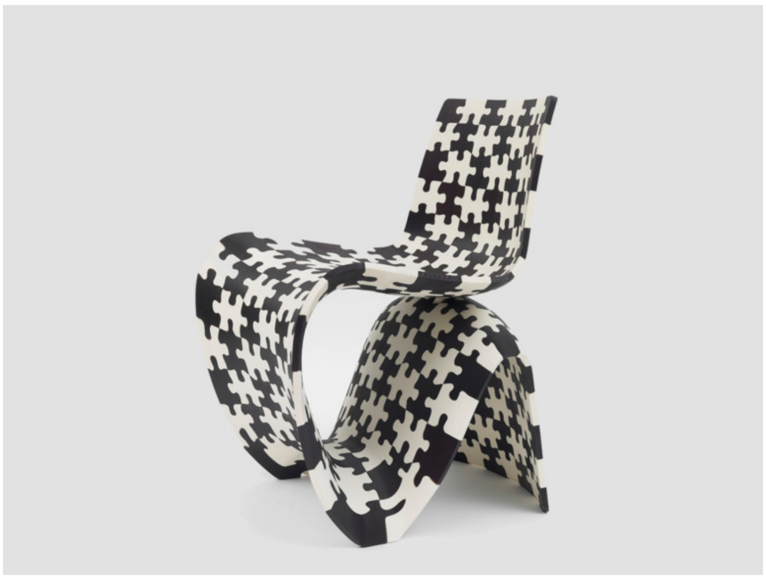


Figure 1.3.: Joris Leerman's Puzzle 3D Makerchair. Figure taken from [Friedman Benda, 2014]

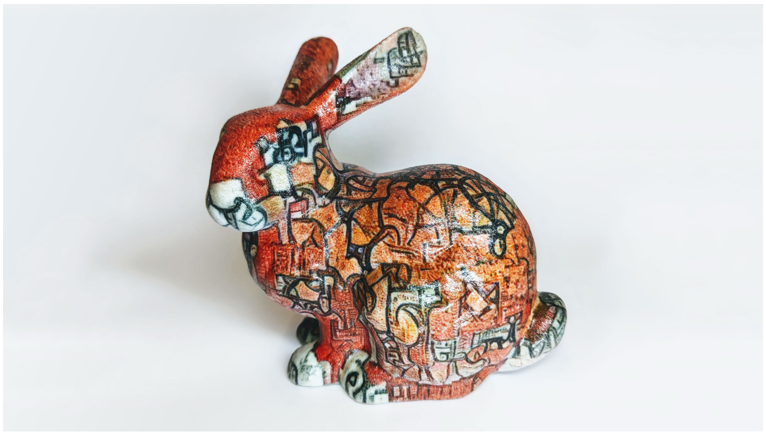


Figure 1.4.: An example of what is possible in 3D printing a styled reconstruction. Shown is a 3D color print of the Stanford bunny [Turk, 2000], styled using Leger's La grand parade sur fond rouge. Figure take from Mordvintsev et al. [2018].

We explore two methods of style transfer, Gram matrix styling which was introduced in NST, and Nearest Neighbor Feature Matching (NNFM) which was introduced in ARF. We compare the results of each method to see which creates the most faithful 3D style transfer. We will also compare our results to the previously mentioned texture based approach, as well as ARF.

1.2. Research Question

The research question is as follows,

To what extent can a styled 3D reconstruction from a set of content images and a style image be created such that style consistency is present across all views.

This will be achieved by training IDR's appearance network to optimize for style, which is created using NST. Further subquestions are:

- **What style transfer method creates the most faithful result for 3D style transfer?**
- **How does our method compare to other 3D style transfer methods?**
- **What effect do the parameters have on the final result?**

1.3. Scope

The focus of this thesis is to develop a model that is able to reconstruct 3D geometry based on masked images, as well as style the geometry using a style image. The 3D reconstruction method as well as the styling will be created using previous works, which will be combined into one pipeline seamlessly. The geometry trained will not be affected by styling, only the appearance will change. Once a method has been established, we will explore how to achieve the best results based on the different parameters available, as well as compare our method to previous 3D style transfer models.

1.4. Thesis Outline

In Chapter 2 we expand on how IDR achieves 3D reconstruction, and discuss two methods, using Gram matrices and NNFM, for styling the surface. We discuss further in Chapter 3 how ARF is able to create 3D style transfer, and outline an approach using textures by Mordvintsev et al. [2018]. We examine both their limitations. We describe our method for 3D style transfer in Chapter 4, showcasing the general architecture and going into further detail on how style transfer is achieved. We also discuss the implementation differences of the style methods. We mention how we will compare our method to previous approaches, describing the qualitative comparison and user study performed. In Chapter 5 we examine how to implement the model, which datasets have been used for experimentation, and how a user can implement their own dataset. Chapter 6 describes the results of the experiments, and includes a discussion on how to achieve the best results based on the different parameters available. We also conduct a comparison study between previous methods and our model, and showcase the results of the user study. We finish by discussing the main contributions as well as limitations of our model, and describe possible future work in Chapter 7.

2. Theoretical Background

2.1. Implicit Differentiable Renderer

Implicit Differentiable Renderer (IDR) developed by Yariv et al. [2020] allows us to create a 3D reconstruction of an object as well as infer the appearance of said object, using an input of masked 2D images. This is achieved by training two neural networks to represent the geometry and appearance. An image of the reconstructed object is then rendered based on learnable camera information, which is used to compare its result to the original images, as a way to optimize the networks.

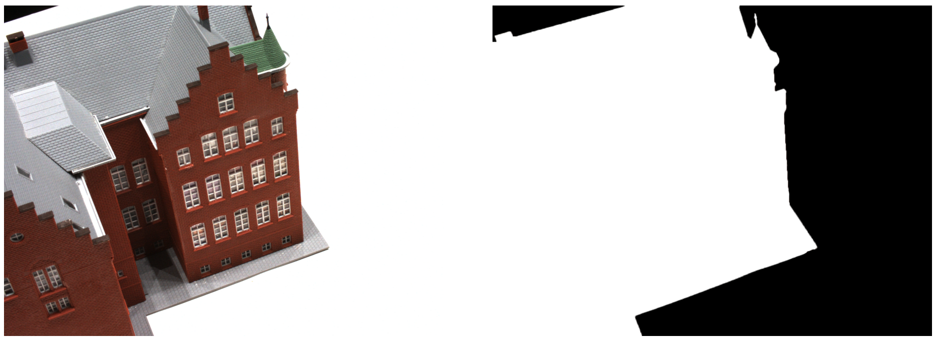


Figure 2.1.: Example of input image with its mask. The dataset is taken from Jensen et al. [2014a].

We define our parameters for learning: θ for geometry , γ for appearance, and τ for camera information.

IDR is initialized with a grey unit sphere (Figure 2.3). The surface will be defined by a signed distance function (SDF) **f**, an implicit function with the property that, for some 3D point **x**,

- $f(x;\theta) > 0$, if the point is outside the surface
- $f(x;\theta) < 0$, if the point is inside the surface
- $f(x;\theta) = 0$, if the point is on the surface

We can therefore define our surface as the zero level set of **f**,

$$\mathcal{S}_\theta = \{\mathbf{x} \in \mathbb{R}^3 \mid f(\mathbf{x};\theta) = 0\}$$

This function is represented by a neural network, optimizing θ, which will be morphed into the correct geometry.

From a given camera position $\mathbf{c}(\tau)$ and some image pixel $\mathbf{p}$ we can define the viewing direction $\mathbf{v}(\tau)$ and calculate the intersection $\mathbf{x}(\theta,\tau)$ of the viewing ray $R(\tau) = \{\mathbf{c} + t\mathbf{v} \mid t \geq 0\}$, shown in red in Figure 2.2, with the implicit surface. This is achieved by sphere tracing (Section 2.1.1). From this point on the surface we calculate the normal using the gradient $\mathbf{n}(\theta) = \nabla f(\mathbf{x})$.

To calculate appearance, we take as input the calculated surface point $\mathbf{x}$, normal $\mathbf{n}$, camera direction $\mathbf{v}$, and global light effects, a global feature vector which allows the renderer to better handle lighting effects created by the geometry itself, such as secondary lighting and self-shadows. This is accounted for by introducing a vector $\mathbf{z}(\mathbf{x};\theta)$

The amount of light reflected from the surface point $\mathbf{x}$ into the camera pointing in direction $\mathbf{v}$ is determined based on the light emitted in the scene, as well as the bidirectional reflectance distribution function (BRDF), which describes the amount of light leaving surface point $\mathbf{x}$ towards the camera in proportion to the light arriving at $\mathbf{x}$. We store these values as a function, which is approximated by a neural network $M(\mathbf{x},\mathbf{n},\mathbf{z},\mathbf{v};\gamma)$.

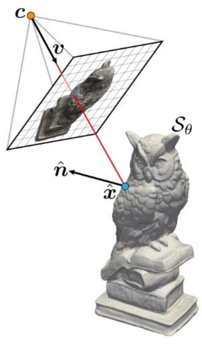


Figure 2.2.: Notation used by Yariv et al. [2020]

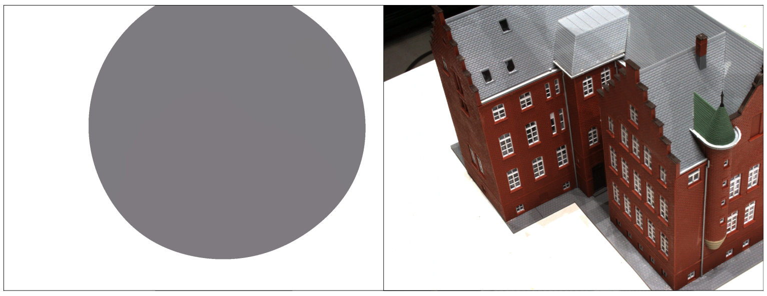


Figure 2.3.: Initialisation compared to ground truth

By combining these two models, we are then able to render predicted images $L(\theta,\gamma,\tau)$ of the trainable geometry and appearance at a camera positions similar to our input images. We compare pixel by pixel the predicted images to the ground truth from our input images to compute a loss used to optimize these parameters. Included are three losses; the color (RGB) and mask loss at pixel p, as well as the *Eikonal loss term*, ensuring the geometry is regular and smooth [Gropp et al., 2020].

The mask loss is calculated by creating a mask of the approximation L, based on if the ray and the surface intersect,

$$S(\theta,\tau) = \begin{cases} 1 & \text{if } R(\tau) \cap \mathcal{S}_\theta \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

As this is not differentiable, we approximate,

$$S_\alpha(\theta,\tau) = \operatorname{sigmoid}(-\alpha \min_{t\geq 0} f(\mathbf{c} + t\mathbf{v};\theta)) \tag{2.2}$$

Such that $\lim_{\alpha \to \infty} S_\alpha = S$. We can now compare S_α with the ground truth mask O for each pixel of the input and ground truth image, using the cross-entropy loss,

$$\mathcal{L}_{\text{mask}}(\theta, \tau) = \frac{1}{\alpha |P|} \sum_{p \in P^{\text{out}}} O_p \log(S_{p,\alpha}(\theta, \tau)) \tag{2.3}$$

with P the collection of pixels of the image, and $P^{\text{out}} \subset P$ containing only pixels outside of the mask of S_α or O.

The RGB loss is calculated by comparing the color value of each pixel of the input and ground truth image,

$$\mathcal{L}_{\text{RGB}}(\theta, \gamma, \tau) = \frac{1}{|P|} \sum_{p \in P^{\text{in}}} \mid I_p - L_p(\theta, \gamma, \tau) \mid \tag{2.4}$$

with $P^{\text{in}} \subset P$ containing only pixels inside of the mask of S_α and O. Finally, we have the Eikonal loss term, applied to the SDF,

$$\mathcal{L}_{\text{E}}(\theta) = \mathbb{E}_{\mathbf{x}}(\|\nabla_{\mathbf{x}} f(\mathbf{x}; \theta)\| - 1)^2 \tag{2.5}$$

with $\mathbb{E}$ the Eikonal term which encourages the normals to have a unit magnitude. This ensures the surface to be smooth. The three loss terms are combined,

$$\mathcal{L}(\theta, \gamma, \tau) = \mathcal{L}_{\text{RGB}}(\theta, \gamma, \tau) + \rho \mathcal{L}_{\text{mask}}(\theta, \tau) + \lambda \mathcal{L}_{\text{E}}(\theta) \tag{2.6}$$

with weights ρ, λ to adjust the importance of each loss term. To decrease memory usage, a random subsample of points are taken instead of rendering a complete view every iteration to compare to the ground truth. A useful ability that IDR has is that the two networks are seperate, which allows geometry and appearance to be trained seperately from eachother. After creating a faithful 3D reconstruction of the object, the apperance renderer can be trained further to achieve a different appearance. This is the basis for this research.

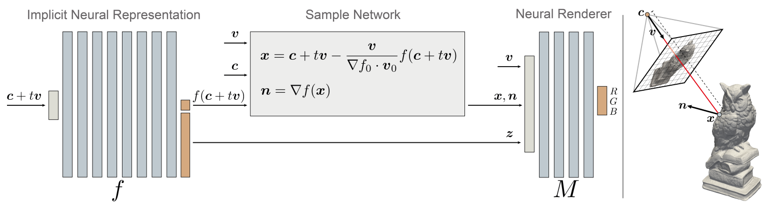


Figure 2.4.: IDR architecture, taken from Yariv et al. [2020].

2.1.1. Sphere tracing algorithm

To calculate the exact intersection $\mathbf{x}_0$ of the ray $\{\mathbf{c} + t\mathbf{v} \mid t \geq 0\}$ and the surface S_θ, we apply sphere tracing. We postulate that the object lies inside the unit sphere, which is achieved by preprocessing the dataset. We begin from the intersection point $\mathbf{c} + t_*\mathbf{v}$ of the ray and the unit sphere, with t_* the value at which the ray intersects the surface. We calculate the distance from the surface using the SDF, equal to $f(\mathbf{c} + t_*\mathbf{v})$. We then march by $f(\mathbf{c} + t_*\mathbf{v})$ along the ray, getting us closer to the surface. We repeat this until the SDF value falls below some threshold, in which case we can say it intersects with the surface, or diverges, i.e. the ray steps out of the unit sphere, which implies no intersection.

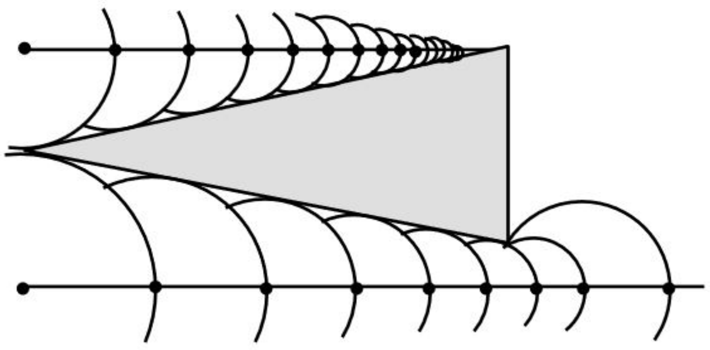


Figure 2.5.: Example of a hit and miss in sphere tracing. From Hart [1995].

2.2. Neural Style Transfer

Schapiro [1952] defines style as ”the constant form - and sometimes the constant elements, qualities, and expression - in the art of an individual or a group [...], exemplified in a motive or pattern”. This consistency has in recent years become replicable using deep neural networks, which synthesize the content from one image and the style from another.

2.2.1. VGG Network

To learn styling, Neural Style Transfer (NST) relies on the VGG network, developed by Simonyan and Zisserman [2015], designed for large-scale image recognition. It is a type of convolutional neural network (CNN), which filters an input image to extract certain features present. These extracted features can then be used to detect certain details in the image, such as a face being recognised from specific facial features, such as eyes or a mouth. For style transfer, feature extraction allows for the content as well as style of an image to be detected.

Convolutional Neural Network

Convolutional neural networks (CNNs) process images to extract lines, patterns, and specific details such as the aforementioned facial features. This is achieved by utilizing filters called kernels, which are matrices applied on a neighborhood of pixels in a process called convolution. This convolution is applied along the whole image, creating a *feature map*, a representative image of a specific feature found in the input image.

Kernels are designed around emphasizing certain features, such as edges or corners. In Figure 2.6 we see the outputs created by the following two kernels,

$$\mathbf{k}_1 = \begin{pmatrix} 0 & -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 \end{pmatrix}, \qquad \mathbf{k}_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & -1 & -1 & -1 \\ 2 & 2 & 2 & 2 & 2 \\ -1 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.7}$$

which respectively detect vertical and horizontal lines in the input image.

An activation function, for example the ReLU function $f(x) = \max(0, x)$, is applied to each output. Only positive values are then present, emphasizing the exaggerated features, while removing any noise from unimportant features.

A collection of kernels, which is called a *convolutional layer*, is applied to the input image, as well as ReLU, to create feature maps representing different specific features of the input image [Bealdung, 2023].

Further feature detection is realized by reapplying kernels to the created feature maps, as well as applying a max pooling layer, which lowers the resolution of the input image by only taking the maximum in a region of pixels [Wood, 2020].

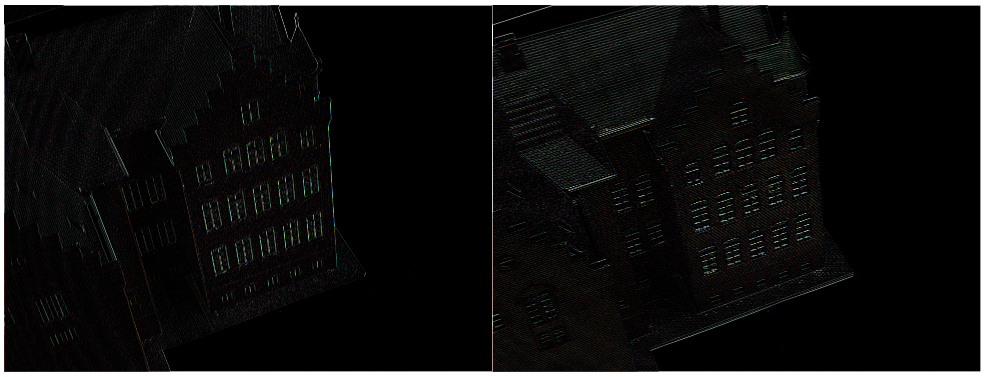


Figure 2.6.: Feature maps from applying kernels 2.7 to the input image shown in Figure 2.1. We see that the vertical and horizontal lines are only visible.

VGG-16 Architecture

VGG-16 is a publicly available CNN used mainly for image classification, and is the backbone for the style transfers discussed in this paper. VGG-16 consists of 16 convolutional layers split into 5 blocks, with a pooling layers in between each block. VGG-16 takes in images of size 244x244, which are normalised by the Imagenet mean $\mu = [0.485, 0.456, 0.406]$, and std $\sigma = [0.229, 0.224, 0.225])$ [Simonyan and Zisserman, 2015].

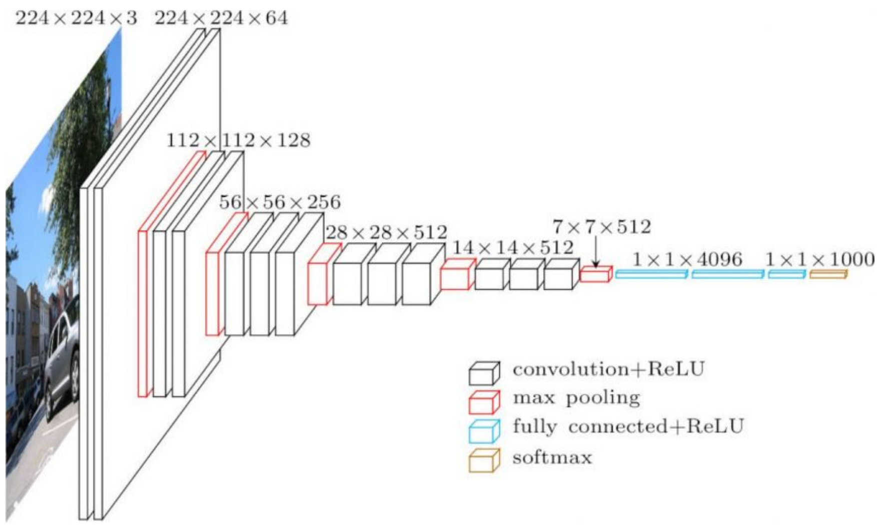

$224 \times 224 \times 3$
$224 \times 224 \times 64$
$112 \times 112 \times 128$
$56 \times 56 \times 256$
$28 \times 28 \times 512$
$14 \times 14 \times 512$
$7 \times 7 \times 512$
$1 \times 1 \times 4096$
$1 \times 1 \times 1000$
convolution+ReLU
max pooling
fully connected+ReLU
softmax

Figure 2.7.: The architecture of VGG-16

From these features we are able to detect objects in the input image, regardless of appearance. For style transfer this can be used to detect what we want to keep consistent from our content image, i.e. having the original shape of the building still visible, while changing its appearance based on a style.

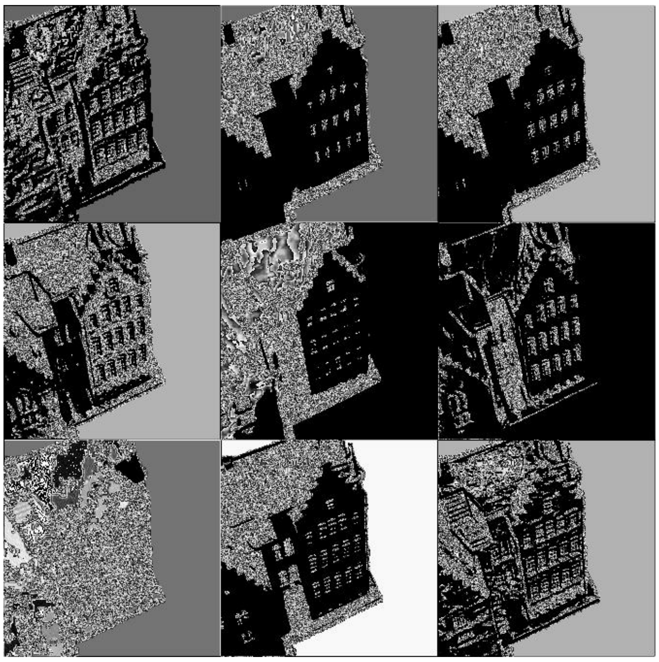


Figure 2.8.: Feature map of Figure 2.1 from the first convolution layer (1_1) of VGG-16, which contains 64 images of shape 224x224 (here 9 are shown). We see certain features emphasized in different layers, such as the complete façade, or just the windows.

2.2.2. Extracting Content and Style from VGG-16

Gatys et al. [2015a] developed a method using the VGG-16 feature extraction to learn and transfer content and style from two different images. NST takes as input two images, a content image $\vec{p}$ and a style image $\vec{a}$. An output image $\vec{x}$, which as an example case will be initialized as a random noise image, will be optimized by minimizing the difference between its content and style compared to the input images. For style minimization, we will discuss two methods; using Gram matrices as applied by Gatys et al. [2015a], and NNFM from Zhang et al. [2022].

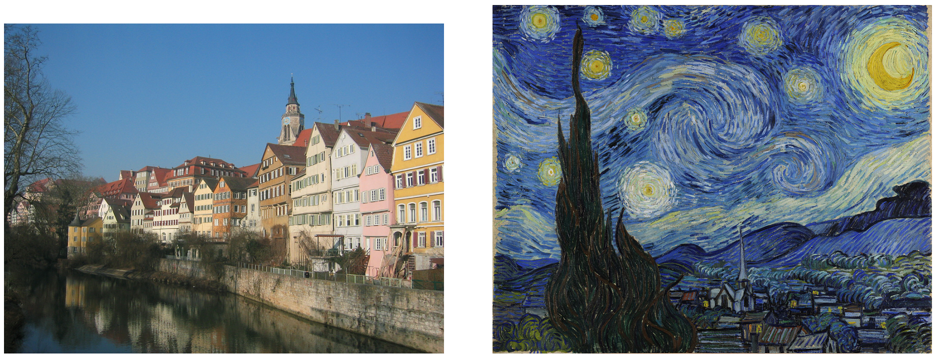


Figure 2.9.: Example content image $\vec{p}$ and style image $\vec{a}$, van Gogh's The Starry Night.

Content loss minimization

As mentioned in the Section 2.2.1, the content can be found by feature extraction. Passing the output image $\vec{x}$ and content image $\vec{p}$ through VGG-16 allows us to extract and compare the content of each image. This is achieved by calculating the mean squared error (MSE) of each feature map of convolutional layer l,

$$\mathcal{L}_{\text{content}}(\vec{x}, \vec{p}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2 \tag{2.8}$$

with F^l, P^l containing N_l feature maps each of size M_l, calculated by multiplying the map's height and width. i, j correspond to the i^{th} feature map at pixel coordinate j. We can see in Figure 2.10 the result of minimizing content loss between the output image $\vec{x}$ and content image $\vec{p}$.

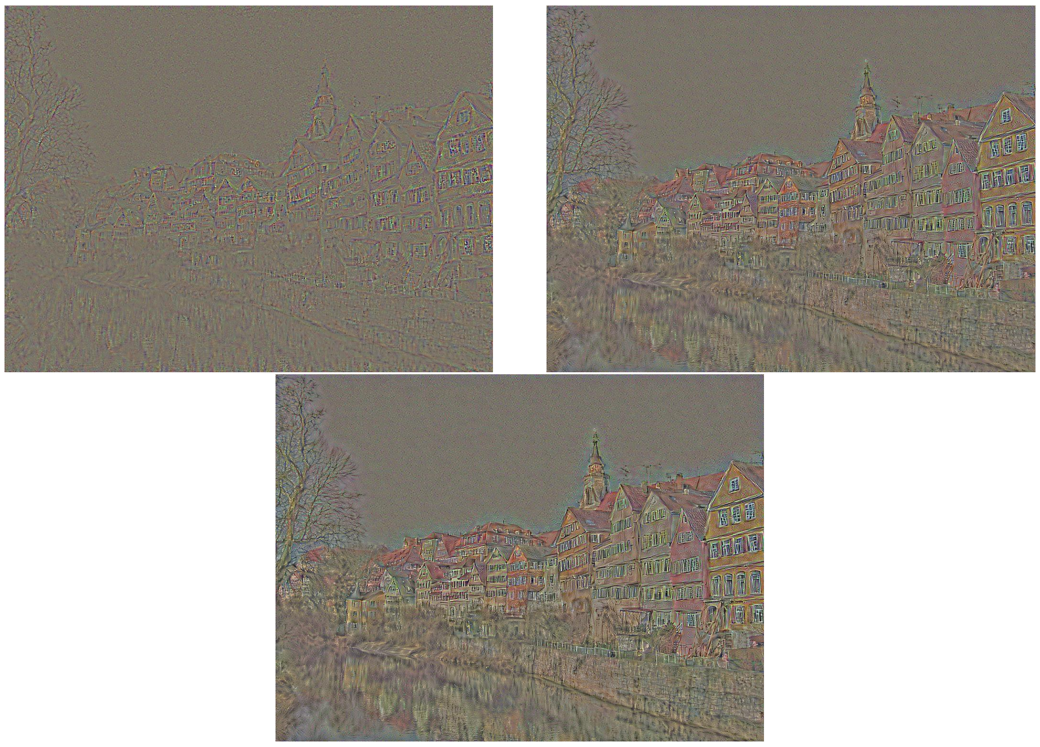


Figure 2.10.: Output image from content learning after 10, 30, 50 epochs. The buildings as well as their color become more clearly defined.

Gram matrix style transfer

For styling, Gatys et al. [2015b] developed a method able to generate image textures from a source image. Image textures quantify the regular pattern visible in an image, and are therefore invariant to spatial information. A description of the texture that discards said spatial information can be given by the correlation between different features in the feature maps. For some pixel location j, we compute this correlation from the product of the feature maps. If both features occur together, then the product will be large, while if there is no correlation the product will be small. Comparing all feature maps for each pixel of the image creates a usable description of texture, which is given by the Gram matrix G^l_{ik}, which for feature maps i and k at layer l, summing over all pixel coordinates j, is defined as,

$$G^l_{ik} = \sum_j F^l_{ij} F^l_{kj} \tag{2.9}$$

The definition of the Gram matrix aligns with the definition we gave for style at the start of Section 2.2. When applied to a piece of visual art, the Gram matrix is able to represent the consistent patterns that we correspond with style. We see this is the case in Figure 2.11.

Just as we optimized content by minimizing the difference between feature maps of our output image $\vec{x}$ and content image $\vec{p}$, we can optimize style by minimizing the difference between the entries of the Gram matrices G^l and A^l of the output image $\vec{x}$ and style image $\vec{a}$. The loss function at layer l is as follows,

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,k} (G^l_{ik} - A^l_{ik})^2 \tag{2.10}$$

with, as mentioned earlier, N_l the number of feature maps, each of size M_l. The total style loss is,

$$\mathcal{L}_{\text{style}}(\vec{x}, \vec{a}) = \sum_l^L w_l E_l \tag{2.11}$$

for some collection of layers L, with w_l a weighting factor defined for each layer. We can see in Figure 2.11 the result of minimizing style loss between the output image $\vec{x}$ and content image $\vec{a}$.

By simultaneously training the output image $\vec{x}$ for content and style, we can optimise for an image including both. We sum both loss functions 2.8 and 2.11,

$$\mathcal{L}_{\text{total}}(\vec{x}, \vec{p}, \vec{a}) = \alpha \mathcal{L}_{\text{content}}(\vec{x}, \vec{p}) + \beta \mathcal{L}_{\text{style}}(\vec{x}, \vec{a}) \tag{2.12}$$

with α and β weighting factors deciding how much of either is visible in the output image.

Since only the feature maps are compared, image size is not restricted, however to ensure that the feature maps have the same effect on both images, they should be of the same size. The style image is therefore resized before computation. To improve convergence time, the output image $\vec{x}$ can be initialized with the content image, instead of with a noise image.

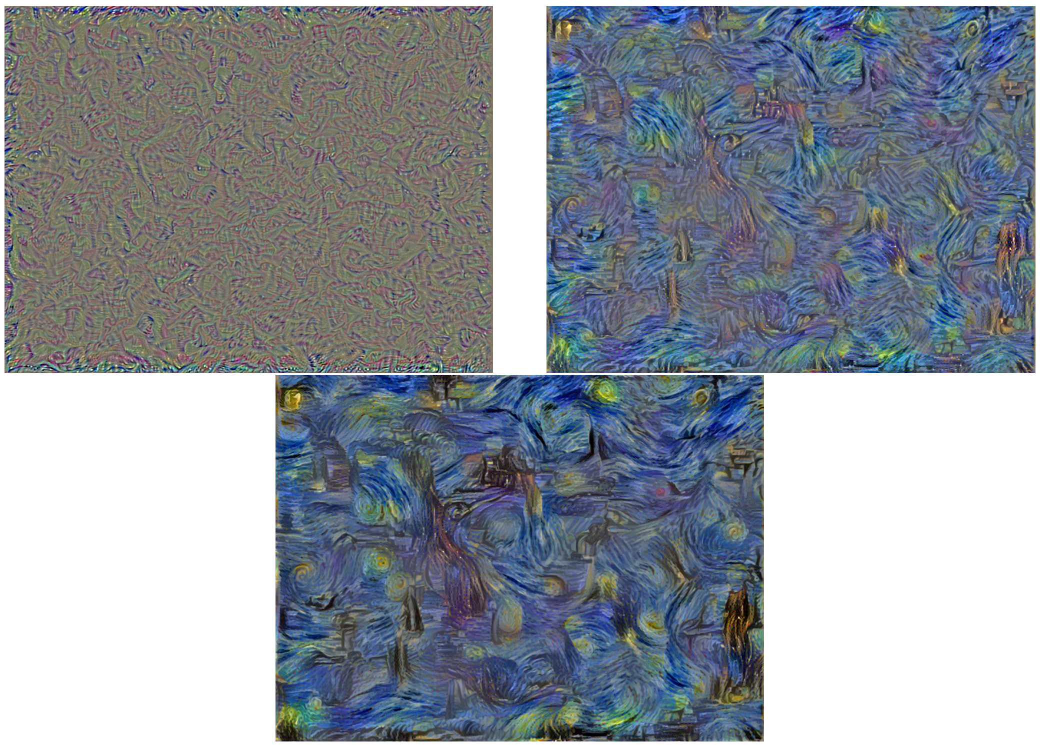


Figure 2.11.: Output image from style learning after 20, 100, 300 epochs. It takes more epochs for the expected style patterns to appear.

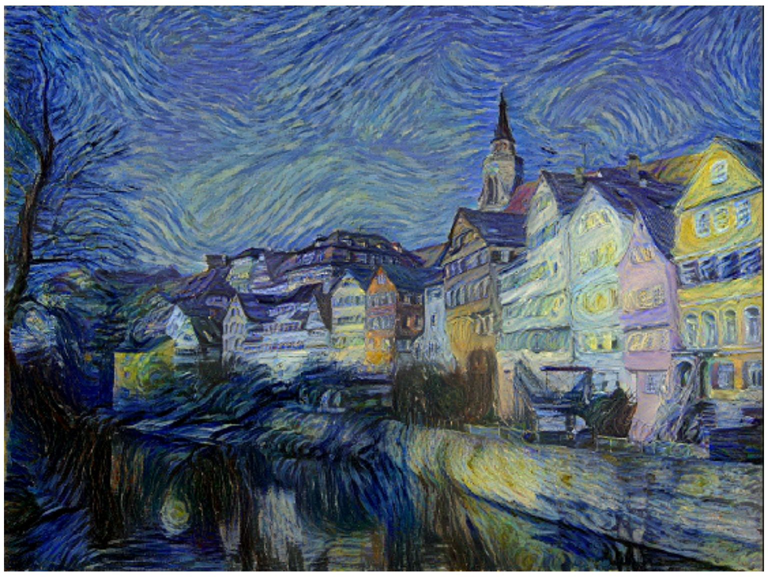
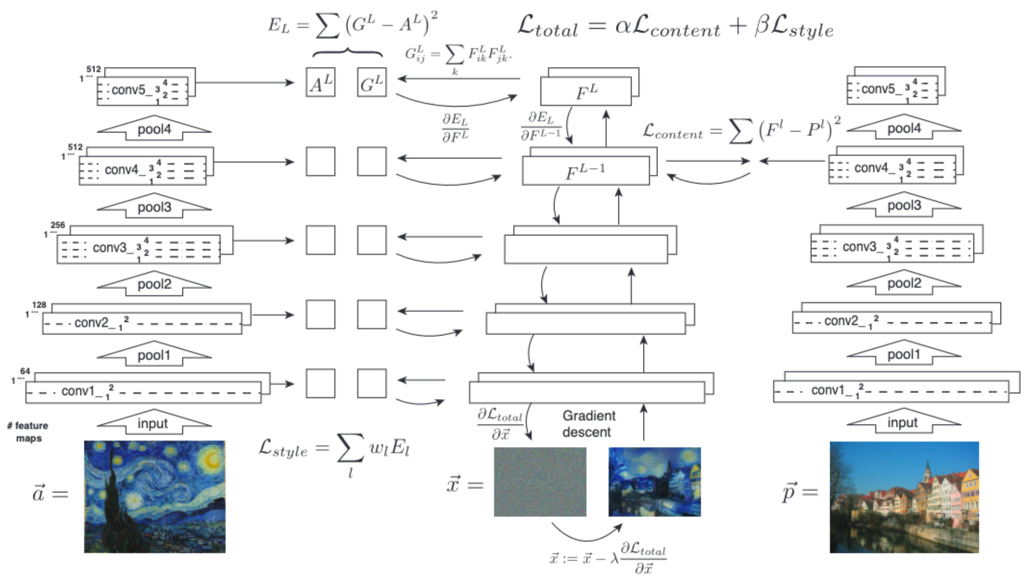


Figure 2.12.: Output of NST. We see the buildings are preserved while van Gogh's thick stroke style, as well as color, becomes visible.



$E_L = \sum (G^L - A^L)^2$
$\mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style}$
$G^L_{ij} = \sum_k F^L_{ik} F^L_{jk}$
$\frac{\partial E_L}{\partial F^L}$
$\frac{\partial E_L}{\partial F^{L-1}}$
$\mathcal{L}_{content} = \sum (F^l - P^l)^2$
$\mathcal{L}_{style} = \sum_l w_l E_l$
$\frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
Gradient descent
$\vec{x} := \vec{x} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
conv5_1 conv4_1 conv3_1 conv2_1 conv1_1
pool4 pool3 pool2 pool1
input
# feature maps
$\vec{a} =$
$\vec{x} =$
$\vec{p} =$

Figure 2.13.: Architecture of NST.

Nearest Neighbour Feature Matching

A different method for Style transfer was developed by Zhang et al. [2022], specifically for 3D style transfer. This method, NNFM, takes a similar input of a content and style image, and calculates content in a similar fashion as NST. NNFM differs however in the method to calculate style loss.

Zhang et al. argue that for 3D scenes, a global measurement of feature correlation fails to capture local details. Instead of calculating Gram matrices, they apply cosine similarity. The cosine distance of two vectors $\mathbf{v}_1, \mathbf{v}_2$ is

$$D(\mathbf{v}_1, \mathbf{v}_2) = 1 - \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\| \|\mathbf{v}_2\|} \tag{2.13}$$

The output of the cosine distance lies between $[0, 1]$, with 0 implying max similarity, the vectors are colinear, and 1 decorrelation, the vectors are perpendicular. For some layer l of the input image and style image, we take the value of each feature map along the pixel location j as our input vectors, giving us the feature vectors $F^l_{\cdot j}$, $P^l_{\cdot j}$. For each feature vector of the input image F^l we find the most correlated feature vector in the style image P^l, and minimize their cosine distance.

$$\mathcal{L}_{\text{nnfm}}(\vec{x}, \vec{a}, l) = \frac{1}{M_l} \sum_j \min_k D(F_{\cdot j}, P_{\cdot k}) \tag{2.14}$$

Note that by taking the minimum, global correlation is avoided, only the strongest feature correlation is taken. By matching the most similar features, we find that similar colors and edges are matched between images, and the features from the input image are converted to the matched features of the style image to create a styled image.

As before, we can calculate a complete loss of content and style by joining the two loss functions 2.8 and 2.14,

$$\mathcal{L}_{\text{total}}(\vec{x}, \vec{p}, \vec{a}) = \alpha \mathcal{L}_{\text{content}}(\vec{x}, \vec{p}) + \beta \mathcal{L}_{\text{nnfm}}(\vec{x}, \vec{a}) \tag{2.15}$$

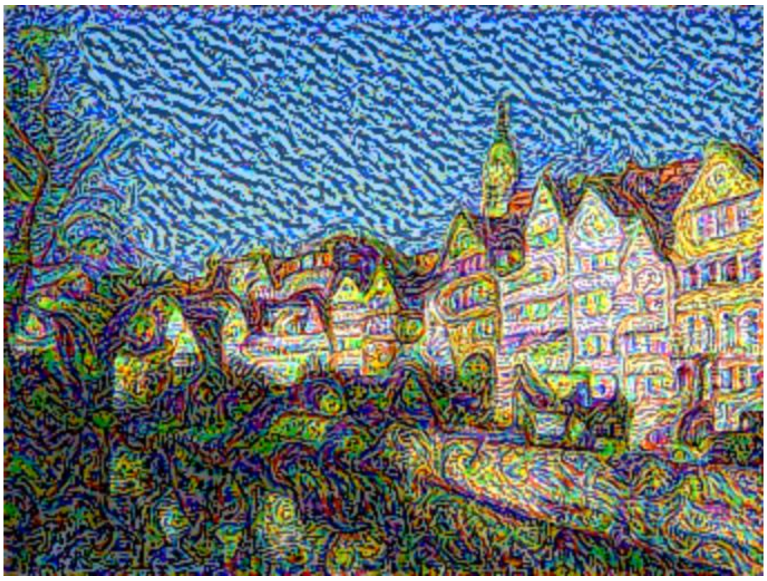


Figure 2.14.: Result of NNFM styling.

We find that the output styling from NNFM is less faithful compared to using Gram matrices. This is because each pixel has its own feature correlation, without any effect from neighboring pixels being included. A neighborhood of pixels can therefore have a lot of variation, which we see with the noisy result in Figure 2.14. However later it will be apparent that the more local approach to style transfer will benefit the final result of 3D style transfer.

3. Related work

3.1. Artistic Radiance Fields (ARF)

3.1.1. NeRF

Neural Radiance Fields (NeRF), introduced by Mildenhall et al. [2020] are currently a popular approach to create new photorealistic images of a scene from a collection of input images. Different to IDR where an implicit function represents the geometry, NeRF creates a volume density σ.

NeRF defines a radiance field, based on rays from known camera locations,

$$F(\mathbf{x}, \theta, \phi) \rightarrow (\mathbf{c}, \sigma) \tag{3.1}$$

where $\mathbf{x}$ is a 3D point along the ray, (θ, ϕ) viewing direction angles, $\mathbf{c}$ the RGB color and σ the volume density. The volume density represents the probability that the ray terminates at $\mathbf{x}$. To render out a view, called *Volume Rendering*, we integrate along the ray $\mathbf{r} = \mathbf{o} + t\mathbf{v}$,

$$C(\mathbf{r}) = \int T\sigma\mathbf{c}\, dt \tag{3.2}$$

With T the accumalated transmittence, the probability the ray does not terminate. The output image can then be compared to the ground truth, and the rendering loss can be back-propagated to create a correct 3D reconstruction. Since this radiance field can be calculated for any camera location, it is possible to synthesize new views.

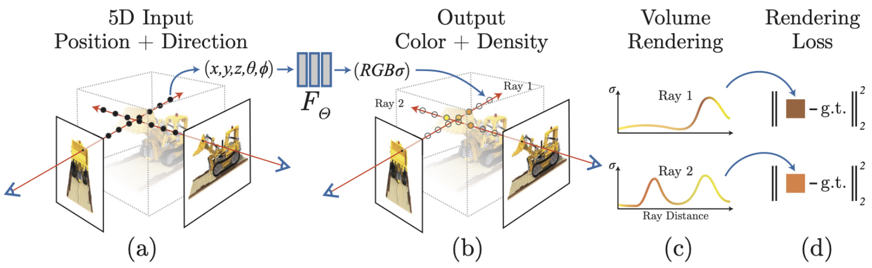


Figure 3.1.: NeRF's approach to 3D reconstruction.

3.1.2. ARF

From these novel view renderings, Zhang et al. [2022] developed ARF to create, similar to our goal, a view-consistent, stylized rendering. This is achieved by taking the rendered image from NeRF as input for style transfer.

As mentioned in Section 2.2.2, VGG-16 feature maps from the rendered and styled image are calculated, and NNFM is applied to calculate style loss. The loss is then back-propagated to update the color of the scene based on the style.

An important addition ARF included for combining 3D reconstruction with style transfer is deferred back-propagation. As every pixel of the output of ARF is rendered from a ray, creating a new full-resolution image at each optimization step would be very memory intensive. Sparse pixel-sampling, which as mentioned before is used by IDR for training, but styling requires the whole image for optimization.

Instead, full-resolution images are rendered with automatic differentiation turned off, so that the gradient for each ray is not stored in memory. We then compute the style loss and gradient of each pixel, giving us a cached gradient image. We can then rerender the image using random pixel-sampling, now with auto-differentiation enabled, using the cached gradient image as gradients to back-propagate, reducing memory usage.

There are limitations to ARF, specifically due to the usage of NeRF. NeRF recreates a complete scene, instead of just the surface with IDR. ARF then creates a complete styling of the scene instead of only the surface. For our purposes it is preferred to have only the surface styled, as this allows us to extract it from the rest of the scene. It also can create artifacts called floaters, where an incorrect ray intersection is established, leaving floating bits of geometry. While there are methods to remove this [Warburg et al., 2023], we find that IDR improves on this as it does not create such artifacts.

3.2. Texture-based 3D style transfer

Mordvintsev et al. [2018] apply a method for style transfer by styling and creating a texture for the object. The approach taken is similar to UV-mapping, where each point $\vec{x}$ of the geometry is associated with a coordinate (u, v) of the texture image. Instead of styling the UV-map and re-projecting the style colors to 3D, which causes issues discussed in appendix B, their approach is to optimize the style texture by, similar to ARF, creating a view which is used as input for style transfer. The style loss is then back-propagated through the renderer, optimizing the texture.

While this method has the benefit of creating a style of only the surface, since a texture is being optimized, the styled 3D object still has some discontinuities caused by trying to project a 3D object onto a 2D plane. The 3D object is also represented using a triangulated mesh, which can be of varying quality when 3D reconstructed and can be computationally costly [Niemeyer et al., 2020].

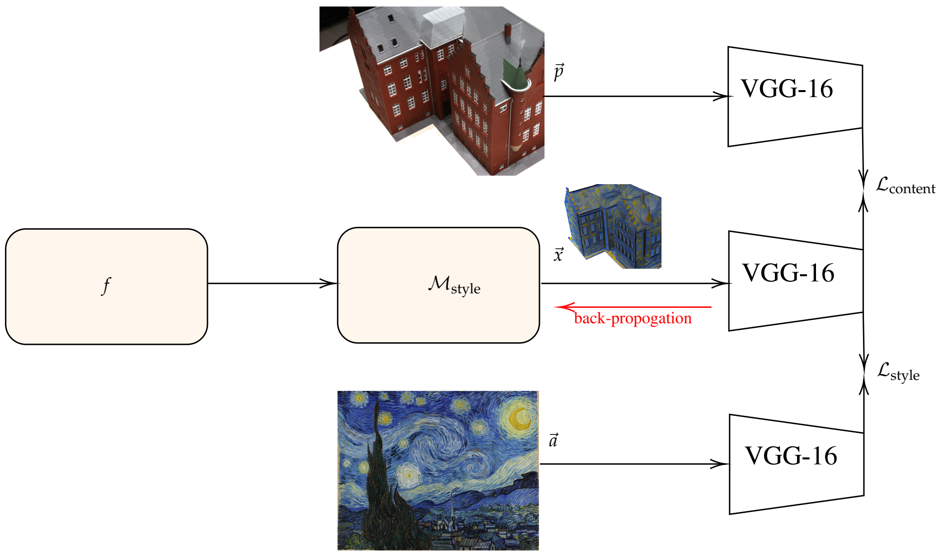
4. Methodology

4.1. Data preparation

For initialization, we require masked images for 3D reconstruction, and a style image. The masks are generated manually,using photoshop. IDR also requires a noisy camera setup, which can be created using an included preprocessing script. This creates an approximate camera projection matrix and normalization matrix, to ensure that the object is inside of the unit sphere, required for sphere tracing, as mentioned in Section 2.1.1.

4.2. Architecture

The network architecture is shown in Figure 4.1. We combine Implicit Differentiable Renderer (IDR) and Neural Style Transfer (NST), and do so in a seamless manner.



$\vec{p}$
VGG-16
$\mathcal{L}_{\text{content}}$
f
$\mathcal{M}_{\text{style}}$
$\vec{x}$
VGG-16
back-propogation
$\mathcal{L}_{\text{style}}$
$\vec{a}$
VGG-16

Figure 4.1.: Architecture of our method. We train the style renderer to create a styled appearance of the original object.

The approach is to first train both geometry f and appearance M using IDR. We then train a copy of the appearance renderer M_{style} to further style the appearance based on a given style image, which is outlined in Section 4.3.

We adapt from ARF deferred back-propagation for our masked input (Section 4.4) to reduce memory usage. We further discuss the differences of styling methods (Section 4.5) for styling the geometry.

4.3. Style transfer

Our goal is to train a style renderer M_{style} to create a stylized 3D scene given a 2D style image. This is achieved by creating a rendered view of the trained object, and using style transfer to calculate the content and style loss of said view. We then back-propagate the losses to optimize the renderer.

Once we have the rendered view, it is resized to reduce memory cost. The amount of rescaling affects the detail in the final render however, so image size should be maximised. As described in Section 2.2, we use the view as the training image $\vec{x}$, introduce a style image $\vec{a}$, and a content image $\vec{p}$, which can be the original image. We calculate the content and style loss, which we back-propagate to the renderer to optimize its parameters. We go into further detail on how this is done in the next section. This process is iterated for this single view to achieve the desired amount of style. We then repeat the process also for different views to create the optimal style for all views. Training using different views ensures that areas of the surface that were hidden at an are correctly styled. Figure 4.2 shows what happens when styling is only done for to one view.

Since geometry has been trained, we disable grad calculation for the geometry network. This ensures that geometry stays consistent while appearance is further trained.

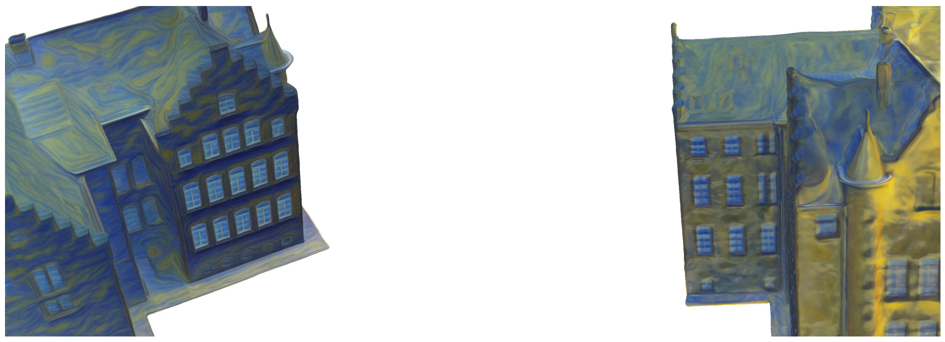


Figure 4.2.: Styling using a single view gives a correctly styled result from one angle, but not from the other.

4.4. Masked deferred back-propagation

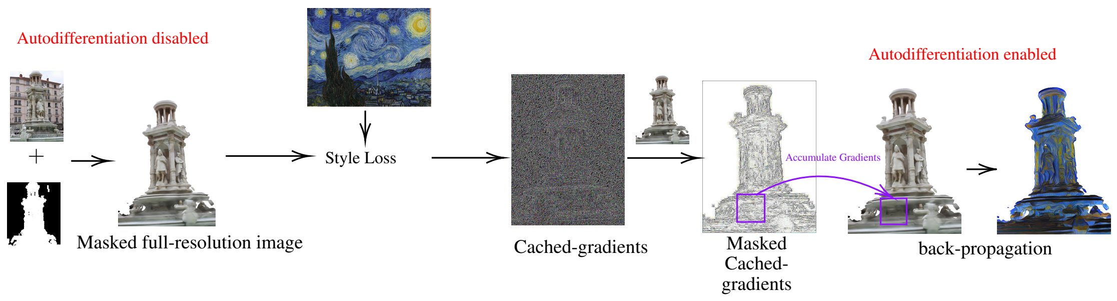


Figure 4.3.: The structure of masked deferred back-propogation. We disable autodifferentiation to save on memory, and render a full-resolution image. We then calculate style loss and store the loss gradient of each pixel. We enable auto-differentiation and return to patch based rendering to back-propagate the losses and train the style renderer.

It is important to note that what we are optimizing is not the whole scene visible in the training image, but the masked output of the Neural Renderer. This masked output is created by sphere tracing from the camera position to the geometry, based on pixels inside of the input mask. As sphere tracing is a memory-intensive process, and since only certain pixels are of importance during training, to improve performance we use masked deferred back-propagation.

Instead of rendering the full-resolution image, IDR uses sparse pixel-sampling, where random sets of pixels are sampled, to save on memory. Since style transfer methods, relying on CNNs, require the complete image to create the correct styling, it is necessary to combine these patches which is done using deferred back-propagation [Zhang et al., 2022].

For each patch, we render the appearance without auto-differentiation. These patches are then synthesized to form a full-resolution image, which is used to calculate style loss. This calculated loss gives for each pixel a gradient, generating a cached gradient image. The Neural Renderer is only trained on pixels inside of the mask, therefore background pixels have their gradients filtered out.

Re-rendering the appearance of each patch, now with auto-differentiation enabled, allows for the masked cached gradient to be used for back-propagation. This ensures that image rendering for styling as well as back-propagation is less memory-intensive, while having only masked points styled.

4.5. NNFM and Gram matrix styling

When styling, we can style with either NNFM or Gram matrix styling. It is also possible to combine both methods, using weights,

$$\mathcal{L}_{\text{style}} = \alpha \mathcal{L}_{\text{Gram matrix}} + \beta \mathcal{L}_{\text{NNFM}} \tag{4.1}$$

Since the styling methods have a different order of magnitude, we scale our values of α and β such that they have a similar effect on the final output.

4.5.1. Qualitative Comparison and User Study

We compare our results to ARF and texture-based 3D style transfer. We compare our results based off of a similar result from the texture-based approach, specifically the Louvre statue [Bardou, 2017], with styling from van Gogh's Starry Night. For ARF we created a result using the same dataset as for our model, but we find that the output is not of high enough quality for comparison, so we also include a render of the lego truck dataset from Yu et al. [2021] in our comparison study, taken straight from the ARF project page.

We also perform a user study to compare the related works to our method. We present a video of an object created by each method, styled using van Gogh's Starry Night. Users are then asked to rate out of 5 how faithful each method preserves the style when transferring it onto the 3D object.

The surveyed group consist of Geomatics students as well as members of the general public, to ensure a wide range of results.

5. Implementation and Experiments

5.1. Implementation

For IDR, the parameters are untouched, with the geometry network f having 8 layers with hidden layers of width 512. The renderer M has 4 layers with width 512 hidden layers. The loss function has an Eikonal weight of 0.1 and a mask weight of 100. The mask approximation value α is initialized at 50, and multiplied by 2 every every 250 epochs.

IDR's learning rate is set at 1e−4, which decreases by a factor of 2 at epochs 1000 and 1500, trained for a total of 2000 epochs. The Adam optimizer is used [Kingma and Ba, 2017].

Styling uses pre-trained VGG-16 model, rendering the styling using convolutional block 2 for both NNFM and NST. Styling is also trained using Adam, for consistency. It's learning rate is set at 1e−4, trained for 200 epochs on each view. The results are rendered using only style loss, except for the content ablation study, where the content weight is set to $5e^{-3}$ and style weight to 1, and the combined style ablation study, where the Gram weight is set to 0.05 and the NNFM weight to $1e^{11}$.

PyTorch [Paszke et al., 2019] is used to create and train the neural networks.

Both IDR and style transfer were trained on the DelftBlue supercomputer using a NVIDIA Tesla V100. IDR ran for 8 hours, while styling took 3 minutes per view. The images used for style loss were resized to 500x500.

For the comparison study, we use the ARF-svox2 release of ARF, which uses Plenoxels [Yu et al., 2021], a model similar to NeRF, since it is as of now the only available ARF model. This was trained on a NVIDIA GTX 1600s card.

5.2. Datasets

For experimentation, we use

- The DTU dataset from Jensen et al. [2014b], specifically scan24, with mask information and fixed camera from Yariv et al. [2020]
- Statue dataset from BlendedMVS [Yao et al., 2020]

We use a diverse collection of style images to test for quality with different style forms, specifically

- van Gogh's Starry Night
- Munch's the Scream
- Escher's Relativity
- Mondriaan's Compositie met groot rood vlak, geel, zwart, grijs en blauw
- Kandinsky's Squares with Concentric Circles

5.3. Creating your own dataset

To create a 3D reconstruction and style a collection of user created images, a preprocessing python script is included to create the required files for IDR to run. The script is specifically designed to read COLMAP outputs.

COLMAP [Schönberger and Frahm, 2016] is able to estimate camera information and position from a set of images. This is an easy method to create usable input for data IDR. The preprocessing script converts the output camera files from COLMAP to be readable by IDR, as well as normalize the camera matrices.

We create masks using photoshop, simply using the wand tool to remove the background. Figure 5.1 displays an overview of the complete method.

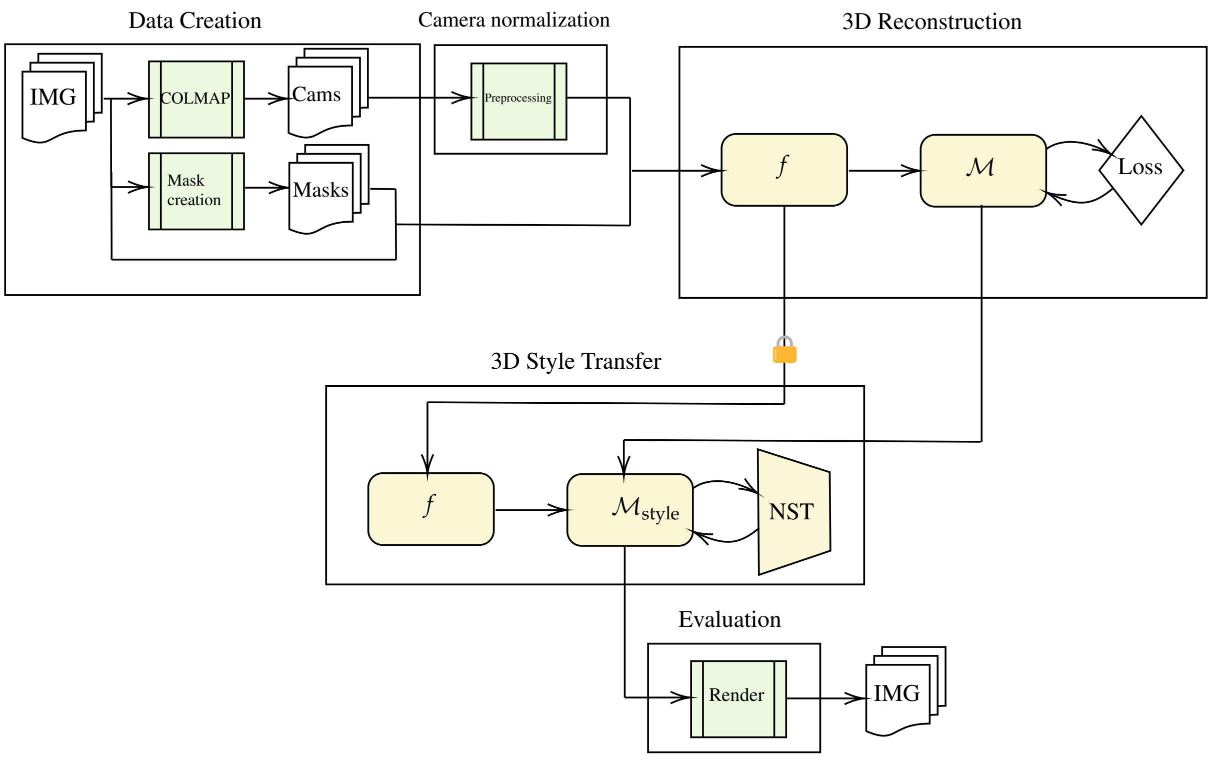


Figure 5.1.: Overview of how to apply our method with a user-generated dataset.

6. Results and Analysis

6.1. Multiview 3D reconstruction and styling

We find when styling with NNFM, shown in Figure 6.1, the faithful reconstruction as well as styling of the DTU dataset scan24 is achieved. The content of the building, such as the windows and roof are still visible, while being stylized in a fashion that corresponds to the style image. When styling using Gram matrix style transfer, shown in Figure 6.2, we find that color has been transferred, the styling is less painterly.

Once trained, we can set the appearance renderer to evaluate, allowing us to render different views, shown in Figure 6.4.

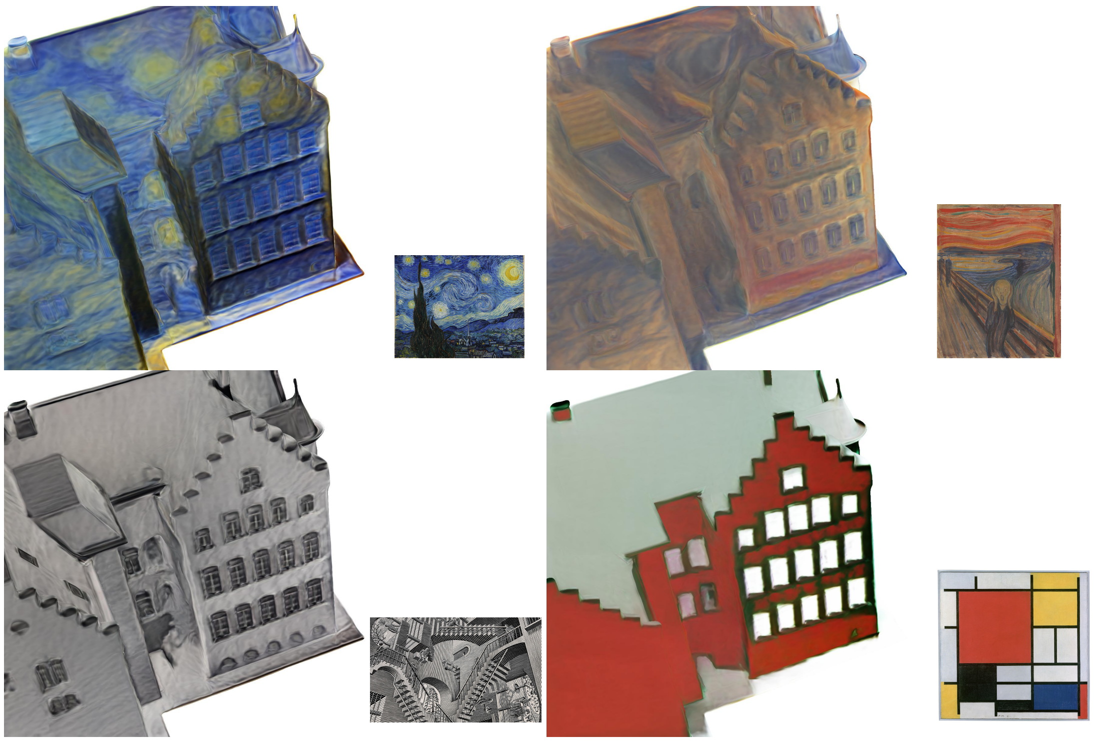


Figure 6.1.: Results of 3D style transfer using our method, with NNFM style loss.

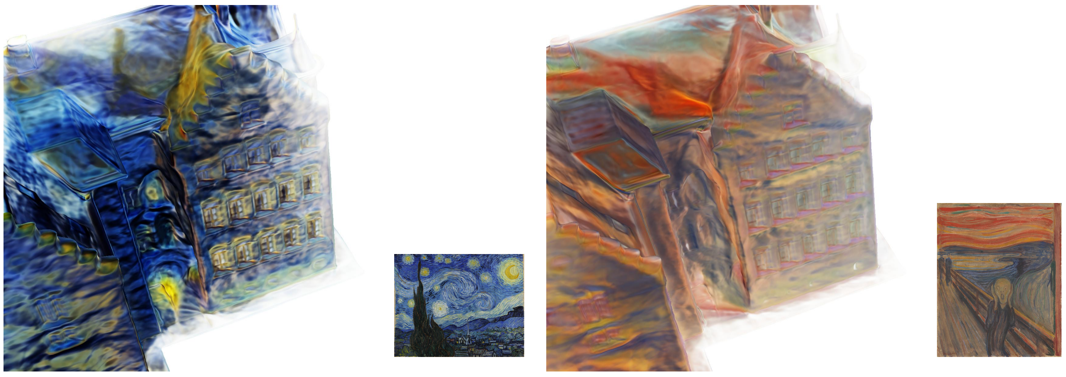


Figure 6.2.: Results of 3D style transfer using our method, with Gram matrix style loss.

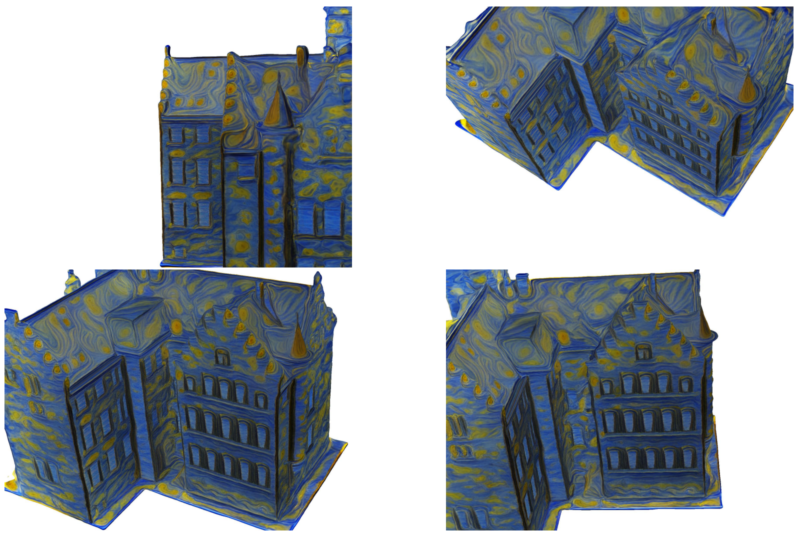


Figure 6.3.: Render of different views.

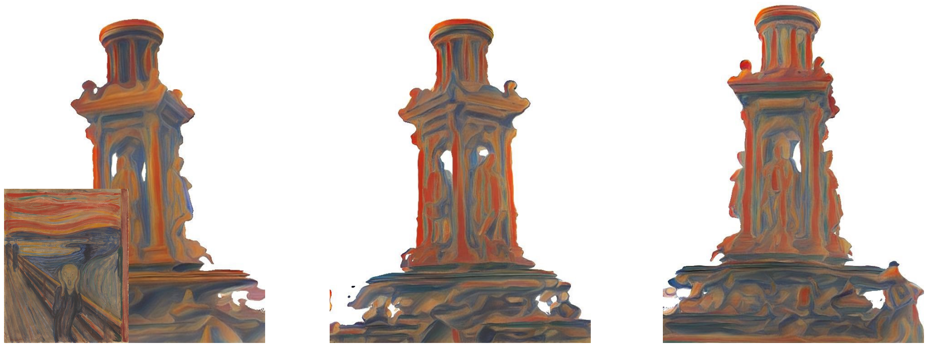


Figure 6.4.: Render of the statue dataset from BlendedMVS [Yao et al., 2020], styled with Munch's the Scream.

6.2. BK and table dataset

We create a styled 3D reconstruction of two user generated datasets, one of the TU Delft Faculty of Architecture and the Built Environment building, and another of a coffee table. We create estimated camera positions using COLMAP, and apply our method to create the results shown in Figure 6.5 and Figure 6.6.

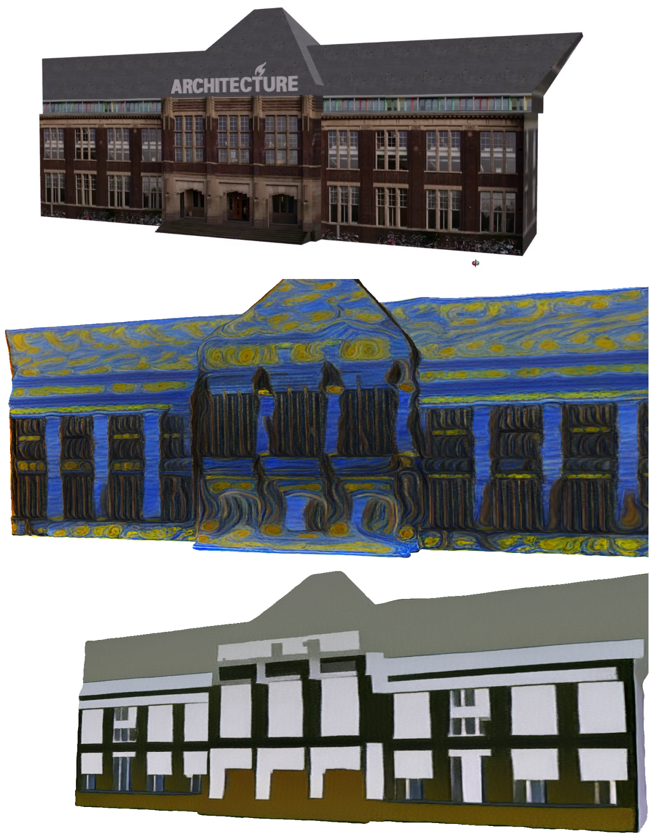


Figure 6.5.: The Architecture building styled. We use a 3D model of the building to improve IDR's result, as it allows for more varied camera poses. The model is taken from van Faassen [2014]

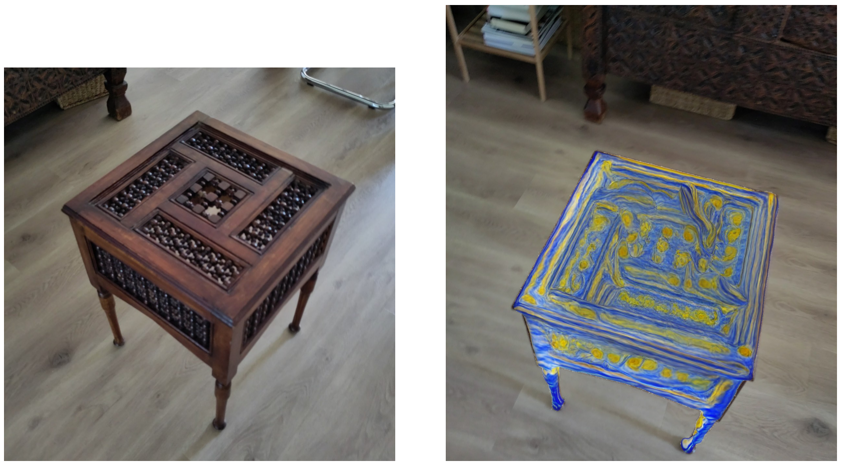


Figure 6.6.: A coffee table styled. The complicated geometry as well as lack of horizontal camera views creates artifacts on the top surface. We visualize what it is like having the object recreated in the real world.

6.3. Qualitative comparisons

When comparing results of each method, we find that our method is able to display a similar style representation to ARF, with the benefit that the background is not reconstructed and styled. Using IDR also creates a closed surface, avoiding the floating artifacts present in NeRF. ARF is however a lot faster in creating a 3D reconstruction. This is due to sphere-tracing being a time-consuming method. While the texture-based approach achieves a similar result, and transfers detail well, we find that the color is not faithful to the painting, and has more apparent patches, due to the texture having separate regions in 2D, shown in Figure 6.9.

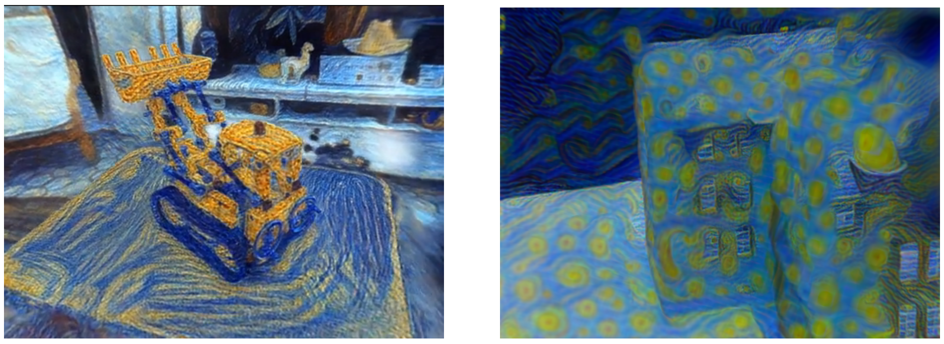


Figure 6.7.: ARF, with our result on the right and the left image taken from the project page [Zhang et al., 2022]. We find the result to be a faithful style transfer similar to our method, but the background is also styled.

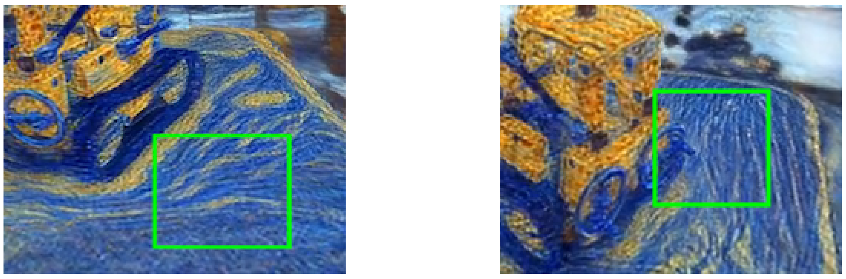


Figure 6.8.: We see that between two views that the surface changes color, with one view showing yellow streaks not present in the other. This is not due to inconsistent styling between the views, but due to floaters covering the yellow streaks.

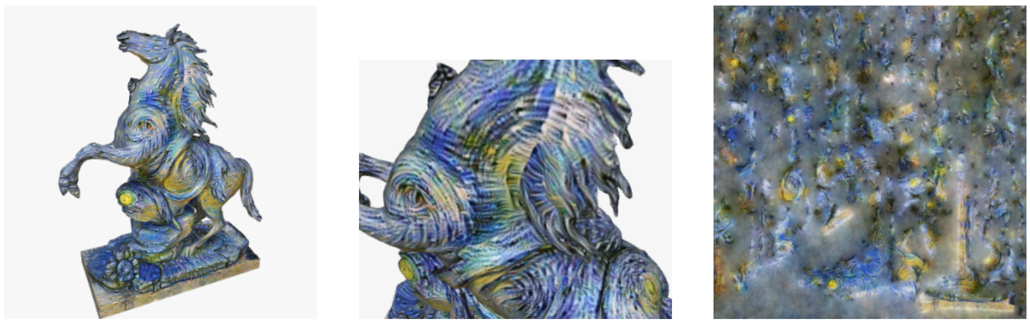


Figure 6.9.: Results from [Mordvintsev et al., 2018]. The strokes are there, but the colors do not completely match, and there are regions of inconsistencies, visible at the mane, created by the gaps in the texture shown on the right.

6.4. User study

From a collection of 55 respondents, we find that on average ARF received a score of 3.77, textured style transfer received a score of 3.49, and our method scored 3.75, all out of 5. The complete results are shown in Figure 6.10. We therefore argue that our method is just as faithful in transferring style as the other methods mentioned.

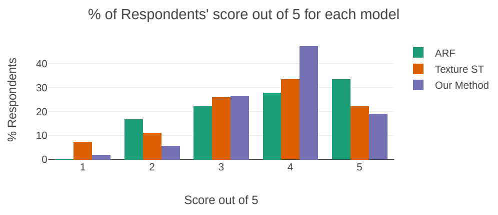


Figure 6.10.: User Study Results.

6.5. Ablation study

Style transfer method

We find that the results of NNFM are more faithful compared to Gram matrix styling. Gram matrix styling varies more when styling different views, leading to the inconsisten result. Furthermore, as shown in Figure 6.2, Gram matrix styling has the white background bleed into the geometry styling, causing a white vignette to appear at the borders. While Gram matrix styling is not as faithful as NNFM, it does emphasize the global characteristics of the style. This is visible in Figure 6.11, where NNFM only styles using two colours, while Gram matrix styling shows a wider range of palettes . By combining both styles, we can introduce a more faithful, as well as more colourful style transfer.

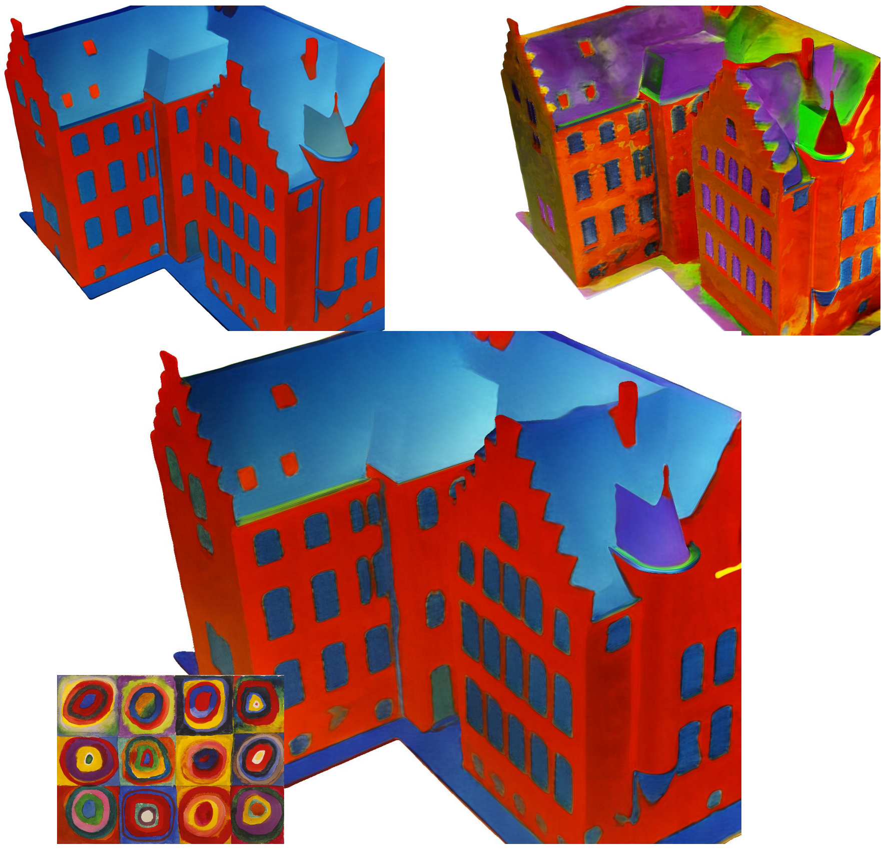


Figure 6.11.: Styling from Kandinsky only with NNFM (L), Gram matrix styling (R), and with both (B). We see that while NNFM is more pleasant looking, Gram matrix styling introduces more colours. Combining both methods allows for an even better result.

Content loss

Introducing content loss ensures that more detail of the reconstructed object is maintained when styling. In Figure 6.12 we see the window frames are largely untouched after styling. The effect is minimal, but can be further emphasized by increasing the content weight.

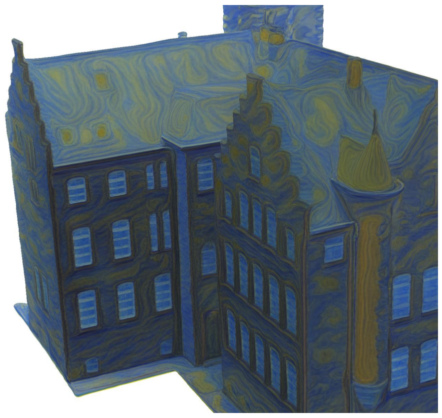


Figure 6.12.: Including content loss ensures that the window frames are clearly visible.

Image size

Rescaling the rendered image before applying style transfer ensures that the process does not run out of memory. It however also has an effect on the final styling, which is shown in Figure 6.13. Having a larger image size allows for more detail to be transferred, as there are more pixels carrying style information.

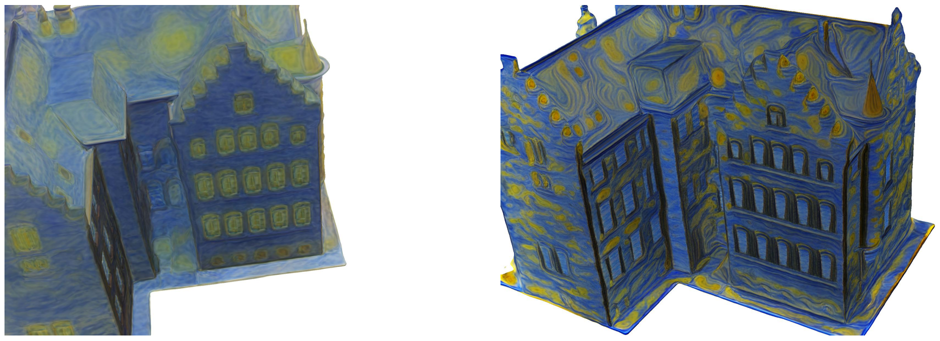


Figure 6.13.: Results where the input for style transfer is scaled to be (300,300) and (500,500).

Convolutional layer

In Section 2.2.1 we see that VGG-16 has 5 blocks of layers. We experiment styling with different blocks, shown in Figure 6.14.

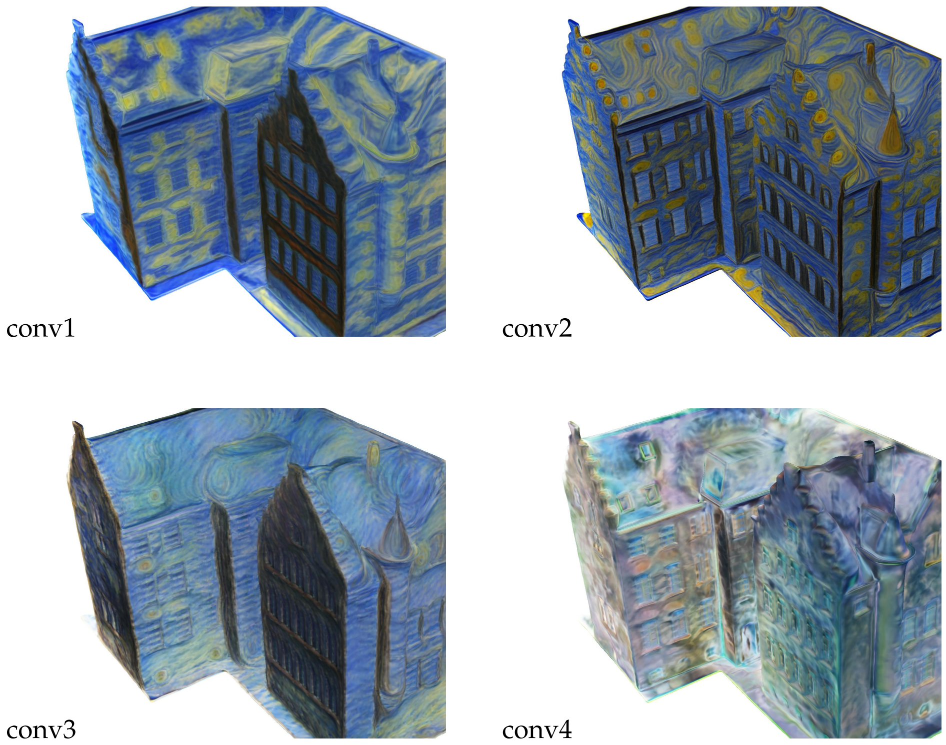


Figure 6.14.: Styling using different convolutional layers from VGG-16.

Note that as the we use deeper layers, the color becomes less striking, while the painterly technique is more developed. Blocks conv2 and conv3 are the most aesthetically faithful.

View and normal independent results

We examine the results of training the style renderer without view information and normal information. We set the input of the view and normal vectors to 0, so that they do not affect the render results.

The results, shown in Figure 6.15, show that without view information, the strokes become more apparent, leading to a better stylized image. For IDR to approximate the correct lighting effects, such as specularity, viewing direction is required [Yariv et al., 2020]. However NST, which is specifically designed for 2D images, is invariant to viewing direction. We however argue that, since we expect style consistency across views and lighting effects such as specularity are not important, it is preferable to have viewing direction ignored.

Ignoring normals however has a negative effect on the overall result. When normal information is ignored, the estimated lighting does not become unique, leading to inconsistent results.

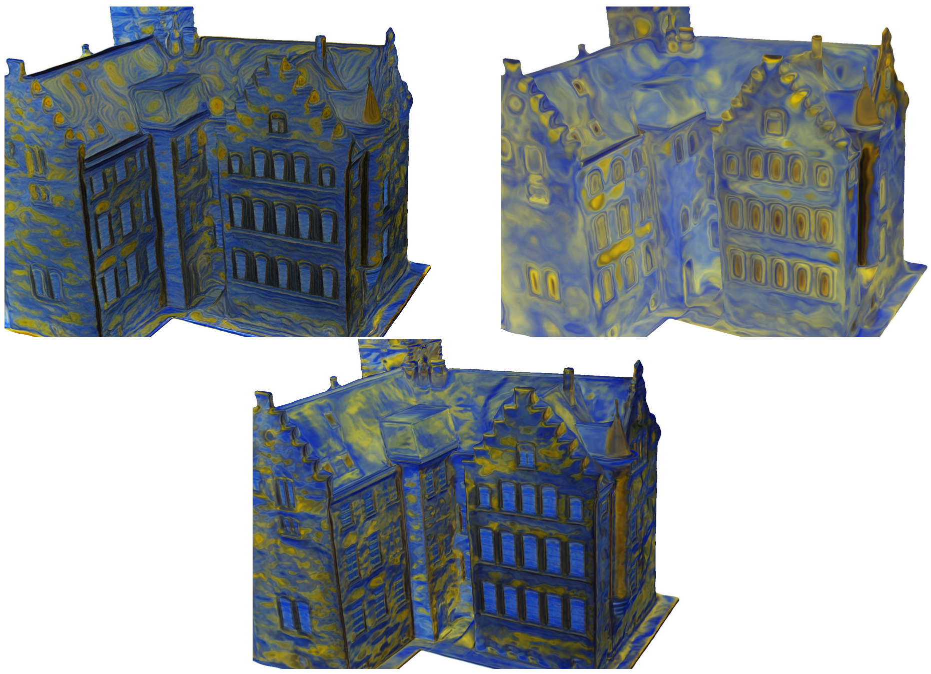


Figure 6.15.: Without viewing direction (L), without normals (R), and both included (B).

7. Conclusions

This thesis focused on developing a novel method able to 3D reconstruct and style an object from a collection of masked images and a style image. We approached this by using IDR's neural renderer, which has been trained to represent the appearance of the 3D reconstructed object, and training it further by applying NST models to the rendered views. Since IDR trains using only a subsample of pixels, and NST methods require a complete image, we introduced masked deferred back-propagation. We examined the effect different parameters have on the final styling results, and discuss how to achieve the most faithful styling. As 3D printing becomes more important for architecture in creating new designs, our model allows for a streamlined method to create a styled 3D reconstruction of any object, from a table to a building, for example in Figure 7.1.

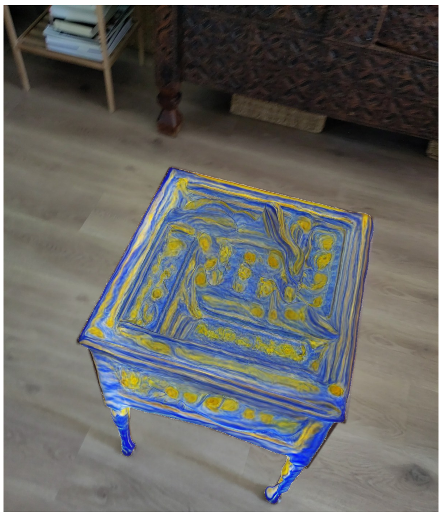


Figure 7.1.: An imagined recreation of a coffee table styled with van Gogh's Starry Night.

We restate the research question,

To what extent can a styled 3D reconstruction from a set of content images and a style image be created such that style consistency is present across all views.

By creating a 3D reconstruction using IDR, we further train the neural renderer by calculating the style loss of a rendered output, and back-propagating the calculated gradient. After training on sufficient views, we are left with a styled result. By having view direction not affect the renderer, we are able to have style consistency across all views. Performing a user study allows us to conclude that the styled result is consistent and faithful to the original style image.

What style transfer method creates the most faithful result for 3D style transfer?

We have explored two style transfer methods, using Gram matrices and using NNFM. We found that NNFM creates the most faithful results, which we correspond to the fact that it focuses on comparing local features instead of the effect of global features. This ensures that when training different views there is more style consistency, and that the white mask has no effect on the geometry's appearance. We do however see a fault in NNFM in Figure 6.11. NNFM does not utilize all the colors present in the painting, as a consequence of nearest neighbour matching. Gram matrix styling does include more colors, and by combining both methods we can overcome both these method's limitations.

How does our method compare to other 3D style transfer methods?

We find that our method is able to create comparable results to other 3D style transfer methods, specifically ARF and 3D texture-based style transfer. Based on user feedback the result is just as faithful in transferring style as the other methods. We argue that our method has the further benefit that only the surface is styled, and in a consistent manner without patches.

What effect do the parameters have on the final result?

We find that training using deeper convolutional layers amplify the strokes found in the painting, at the cost of the less vivid colors being displayed. We argue that block conv2 creates the most faithful result. To have the original appearance be visible, the content loss can be included. Including content loss when calculating style loss ensures that finer details, such as the window frames of the building, stay present. To have as much detail transferred as possible, the image size as input for the style transfer should be maximized.

We find having the viewing direction removed while including normal information creates the most pleasing results. With viewing direction the swirls disappear, while without normal information flat surfaces become too homogeneous.

7.1. Discussion

7.1.1. Contributions

This research furthers the field of 3D reconstruction and style transfer. We have accomplished,

- Creating a new form of 3D style transfer that transfers style onto an implicit surface, which has a fixed memory footprint and requiring no discretization. Only the surface is styled, creating a closed separate styled model.
- Reviewing and experimenting with different datasets, styles, and with different parameters to explore how to create the most faithful results.
- Comparing our method with established 3D style transfer methods, and discussing similarities and differences the results.

7.1.2. Limitations

Geometric artifacts from IDR, due to a lack of information caused by occlusion, has an effect on the final styling. We can see this in the surface of Figure 6.6. When training, no horizontal images were included, causing unnatural artifacts to be created which are still visible in the final style results.

IDR itself is time-consuming method. All the datasets shown took 8 hours to reconstruct. Since for styling we require the object to be correctly modelled, it is not feasible to begin styling before IDR has completed. To optimize for different views, views have to be rendered completely, which requires sphere-traching, which is also time-consuming.

For users to create their own datasets, it is required for the input dataset to include mask information, which has to be manually created. Some form of segmentation can be used to simplify this process.

While we have performed a user study to argue how successful our method is in faithfully transferring style, one can still wonder how, or if it is even possible, to objectively quantify style. A larger study can be performed to create a clear answer to simplify further research in this field.

7.1.3. Future work

For future work, one may explore a maskless adaption of our model, using for example VolSDF [Yariv et al., 2021] or NeuS [Wang et al., 2023]. While having masked data allows for the creation and styling of scenes containing only the intended geometry, having not to create masks can simplify the process while still creating a implicit surface to be styled.

A new developement by Yariv et al. [2023] is BakedSDF, which not only includes a separable appearance renderer similar to IDR, but also allows for the separation of diffuse and specular information. It is very interesting to explore the possibilities for styling only color, keeping specularity intact.

Having a geometry network affected by style could create interesting results, where not only the appearance is affected, but also the geometry, creating for example a chair with sharp angles based on Mondriaan's style.

Now that the method has been established, the next step is for someone to use it to style a piece of furniture, such as a chair, and recreate it using 3D printing.

A. Reproducibility self-assessment

A.1. Marks for each of the criteria

1. **Input data: 3**
   All input data used is available and free to use.
2. **Preprocessing: 3**
   COLMAP and preprocessing script are available, and minimal preprocessing is required.
3. **Methods: 3**
   Method will be available on Github, and is easy to use.
4. **Computational environment: 1**
   CUDA is required, and for high quality results a good graphics card is needed.
5. **Results: 3**
   Results as images and checkpoints will be available on Github.

A.2. Self-reflection

It is possible to reproduce this thesis, only with lower quality results. The input datasets, for 3D reconstruction and styling, are publicly available, and designed for such a method in mind. Therefore minimal preprocessing is required, only needing the necessary input masks and normalized camera information. The method itself is user-friendly, using open-source methods. To run the method however requires a good graphics card and CUDA, which for this thesis was achieved using the Deltblue supercomputer. A user could create results of lower quality themselves. The results created will also be publicly available, with results images as well as checkpoints of the model before and after styling.

B. Projection Style Transfer (Preliminary method)

A preliminary method examined during this thesis was to style the output surface from IDR. The approach was two unwrap the surface, represented as a triangle mesh, using a form of 2D projection, such that the created UV-map could be stylized using NST. The stylized UV-map could then be reprojected onto the surface, creating a styled surface.

An example of a 2D projection is equirectangular projection, where the surface points are projected on the unit sphere,

$$\theta = \arccos \frac{z}{\sqrt{x^2+y^2+z^2}}, \phi = \arctan \frac{y}{x} \tag{B.1}$$

which is then unwrapped using for example equirectangular projection.

$$u = \frac{M\phi}{2\pi}, v = \frac{N\theta}{\pi} \tag{B.2}$$

With N, M the size of output image. This image can be stylized using NST, to create a stylized image which can be reprojected onto the surface. This method is however rather expensive, as many points have to then be projected for a high resolution. Since a 3D to 2D projection is discontinuous, this will create seams at, in the case of equirectangular projection, the poles of the object. This method was therefore discontinued in favor of styling IDRs neural renderer based on rendered views.

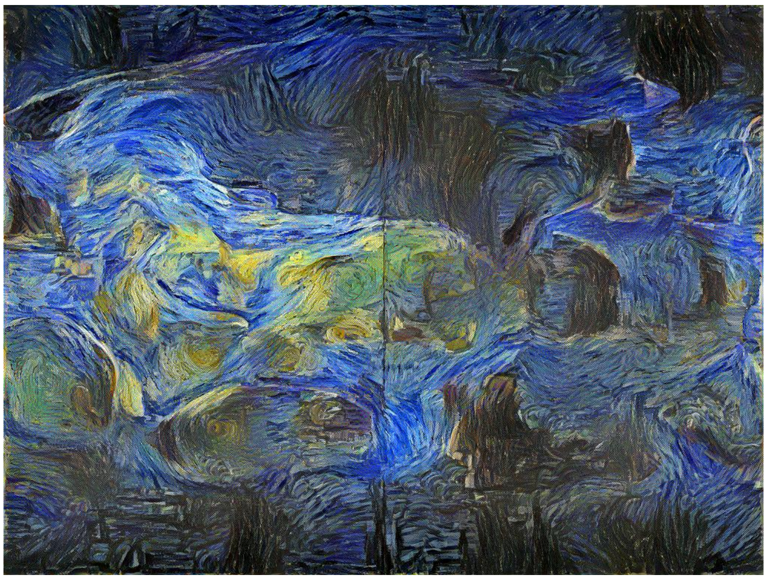


Figure B.1.: 2D projection stylized of DTU's scan65 (skull).

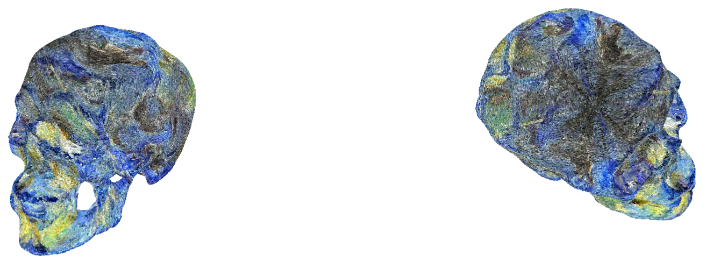


Figure B.2.: Output styling. While the styling looks faithful, the resolution is low. We also see discontinuities at the poles.

Bibliography

Bardou, B. (2017). Marly - louvre museum (low definition).

Bealdung (2023). What is the purpose of a feature map in a convolutional neural network. *Bealdung*.

Friedman Benda (2014). Joris laarman: Maker series.

Gatys, L. A., Ecker, A. S., and Bethge, M. (2015a). A neural algorithm of artistic style.

Gatys, L. A., Ecker, A. S., and Bethge, M. (2015b). Texture synthesis using convolutional neural networks.

Gropp, A., Yariv, L., Haim, N., Atzmon, M., and Lipman, Y. (2020). Implicit geometric regularization for learning shapes.

Hart, J. (1995). Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12.

High Museum of Art (2018). Joris laarman lab: Design in the digital age.

Jensen, R., Dahl, A., Vogiatzis, G., Tola, E., and Aanæs, H. (2014a). Large scale multi-view stereopsis evaluation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 406–413. IEEE.

Jensen, R., Dahl, A., Vogiatzis, G., Tola, E., and Aanæs, H. (2014b). Large scale multi-view stereopsis evaluation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 406–413. IEEE.

Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

Loper, M. M. and Black, M. J. (2014). Opendr: An approximate differentiable renderer. In Fleet, D., Pajdla, T., Schiele, B., and Tuytelaars, T., editors, *Computer Vision – ECCV 2014*, pages 154–169, Cham. Springer International Publishing.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. (2020). Nerf: Representing scenes as neural radiance fields for view synthesis.

Mordvintsev, A., Pezzotti, N., Schubert, L., and Olah, C. (2018). Differentiable image parameterizations. *Distill*. https://distill.pub/2018/differentiable-parameterizations.

Niemeyer, M., Mescheder, L., Oechsle, M., and Geiger, A. (2020). Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Peels, J. (2021). 3d printed furniture and design, the time is now. *3Dprint*.

Schapiro, M. (1952). Style. *Anthropology Today*, page 287.

Schönberger, J. L. and Frahm, J.-M. (2016). Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition.

Turk, G. (2000). The stanford bunny.

van Faassen, W. (2014). Faculteit bouwkunde, tu delft work in progres.

Wang, P., Liu, L., Liu, Y., Theobalt, C., Komura, T., and Wang, W. (2023). Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction.

Warburg, F., Weber, E., Tancik, M., Holynski, A., and Kanazawa, A. (2023). Nerfbusters: Removing ghostly artifacts from casually captured nerfs.

Wood, T. (2020). Convolutional Neural Network. *DeepAI*.

Yao, Y., Luo, Z., Li, S., Zhang, J., Ren, Y., Zhou, L., Fang, T., and Quan, L. (2020). Blendedmvs: A large-scale dataset for generalized multi-view stereo networks.

Yariv, L., Gu, J., Kasten, Y., and Lipman, Y. (2021). Volume rendering of neural implicit surfaces.

Yariv, L., Hedman, P., Reiser, C., Verbin, D., Srinivasan, P. P., Szeliski, R., Barron, J. T., and Mildenhall, B. (2023). Bakedsdf: Meshing neural sdfs for real-time view synthesis.

Yariv, L., Kasten, Y., Moran, D., Galun, M., Atzmon, M., Basri, R., and Lipman, Y. (2020). Multiview neural surface reconstruction by disentangling geometry and appearance.

Yu, A., Fridovich-Keil, S., Tancik, M., Chen, Q., Recht, B., and Kanazawa, A. (2021). Plenoxels: Radiance fields without neural networks.

Zhang, K., Kolkin, N., Bi, S., Luan, F., Xu, Z., Shechtman, E., and Snavely, N. (2022). Arf: Artistic radiance fields.

Colophon

This document was typeset using LaTeX, using the KOMA-Script class `scrbook`. The main font is Palatino.