TU WIEN Informatics

3D Style Transfer

Lifting 2D Methods to 3D and Enabling Interactive Guidance

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Mgr. Áron Samuel Kovács
Matrikelnummer 12130536

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dr.in Renata Georgia Raidou
Zweitbetreuung: Assistant Prof. Pedro Hermosilla

Diese Dissertation haben begutachtet:

Stefan Bruckner	Thomas Leimkühler

Wien, 7. Oktober 2025

Áron Samuel Kovács

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

3D Style Transfer

Lifting 2D Methods to 3D and Enabling Interactive Guidance

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Mgr. Áron Samuel Kovács
Registration Number 12130536

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dr.in Renata Georgia Raidou
Second advisor: Assistant Prof. Pedro Hermosilla

The dissertation has been reviewed by:

Stefan Bruckner	Thomas Leimkühler

Vienna, October 7, 2025

Áron Samuel Kovács

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Mgr. Áron Samuel Kovács

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 7. Oktober 2025

Áron Samuel Kovács

Acknowledgements

I would like to thank my family, friends, and colleagues for their unwavering support.

Kurzfassung

3D-Stilübertragung bezeichnet die Veränderung des visuellen Erscheinungsbildes von 3D-Objekten und Szenen, sodass sie einem vorgegebenen (künstlerischen) Stil entsprechen, der in der Regel als Bild vorgegeben wird. Mittels 3D-Stilübertragung kann die Erstellung von 3D-Modellen wie Umgebungsrequisiten in Spielen, VFX-Elementen oder umfangreichen virtuellen Szenen deutlich vereinfacht werden. Es gibt jedoch einige Herausforderungen, wie die Konsistenz über mehrere Ansichten zu gewährleisten, Rechen- und Speicherbeschränkungen einzuhalten, sowie künstlerische Kontrolle zu ermöglichen. In dieser Dissertation stellen wir drei Methoden vor mit dem Ziel 3D-Modelle zu stilisieren und gleichzeitig die genannten Herausforderungen anzugehen. Wir konzentrieren uns auf optimierungsbasierte Methoden, da diese im Vergleich zu Verfahren mit nur einem Durchlauf Ergebnisse in höherer Qualität liefern. Mit unseren Beiträgen tragen wir zur Weiterentwicklung des aktuellen Standes der Technik bei, indem wir (i) neuartige, oberflächenbewusste CNN-Operatoren für direkte Polygonnetz-Texturierung einführen, (ii) die erste Gaussian Splatting (GS) Methode präsentieren, die sowohl hochfrequente Details als auch großflächige Muster übertragen kann, und (iii) eine interaktive Methode entwickeln, die eine richtungsbezogene und regionsbasierte Steuerung des Stilübertragungsprozesses ermöglicht. Alle drei Methoden bieten eine höhere visuelle Qualität und Robustheit als der aktuelle Ausgangszustand.

In drei komplementären Projekten untersuchen wir unterschiedliche Facetten der 3D-Stilübertragung. Im ersten Projekt schlagen wir eine Methode vor, die Texturen direkt auf der Oberfläche eines Polygonnetzes erzeugt. Indem wir die standardmäßigen 2D-Faltungs- und Pooling-Schichten in einem vortrainierten zweidimensionalen CNN durch oberflächenbasierte Operationen austauschen, erreichen wir eine nahtlose, über mehrere Ansichten konsistente Textursynthese, ohne auf stellvertretende 2D-Bilder angewiesen zu sein. Im zweiten Projekt übertragen wir sowohl hochfrequente Details als auch großflächige Muster mittels GS, während wir representationsspezifische Artefakte wie übergroße oder gestreckte Gauss'sche Elemente adressieren. Darüber hinaus entwerfen wir eine Stil-Verlustfunktion, die Stilmuster in mehreren Größen übertragen kann, was zu visuell ansprechenden, stilisierten Szenen führt, in denen sowohl feine Details als auch großflächige Motive bewahrt werden. Im dritten Projekt entwickeln wir eine interaktive Methode, die es ermöglicht, die Stilübertragung zu steuern, indem Linien gezeichnet werden, die die Musterrichtung vorgeben, und Regionen sowohl auf der 3D-Oberfläche als auch im Stilbild ausgemalt werden, um festzulegen, wo und wie bestimmte Stilmuster angewendet werden sollen.

Wir zeigen durch unsere umfangreichen qualitativen und quantitativen Auswertungen, dass unsere Methoden existierende Verfahren übertreffen. Wir demonstrieren außerdem ihre Robustheit über unterschiedliche 3D-Objekte, Szenen und Stile hinweg und heben die Flexibilität der vorgestellten Ansätze hervor. Weiterführende Forschung könnte Erweiterungen untersuchen, wie geometrieverändernde Stilübertragungen, effizientere Synthese großflächiger Muster, zeitliche Kohärenz in dynamischen oder videobasierten Szenen, sowie verfeinerte interaktive Steuerungsmöglichkeiten, die direkt Rückmeldungen der Künstler integrieren, um ihre kreativen Absichten besser in den Stilisierungsprozess einfließen zu lassen.

Abstract

3D style transfer refers to altering the visual appearance of 3D objects and scenes to match a given (artistic) style, usually provided as an image. 3D style transfer presents significant potential in streamlining the creation of 3D assets such as game environment props, VFX elements, or large-scale virtual scenes. However, it faces challenges such as ensuring multi-view consistency, respecting computational and memory constraints, and enabling artist control. In this dissertation, we propose three methods that aim at stylizing 3D assets while addressing these challenges. We focus on optimization-based methods due to the higher quality of results compared to single-pass methods. Our contributions advance the state-of-the-art by introducing: (i) novel surface-aware CNN operators for direct mesh texturing, (ii) the first Gaussian Splatting (GS) method capable of transferring both high-frequency details and large-scale patterns, and (iii) an interactive method that allows directional and region-based control over the stylization process. Each of these methods outperforms existing baselines in visual fidelity and robustness.

Across three complementary projects, we explore different facets of 3D style transfer. In the first project, we propose a method that creates textures directly on the surface of a mesh. By replacing the standard 2D convolution and pooling layers in a pre-trained 2D CNN with surface-based operations, we achieve seamless, multi-view-consistent texture synthesis without relying on proxy 2D images. In the second project, we transfer both high-frequency and large-scale patterns using GS, while addressing representation-specific artifacts such as oversized or elongated Gaussians. Furthermore, we design a style loss capable of transferring style patterns at multiple scales, resulting in visually appealing stylized scenes that preserve both intricate details and large-scale motifs. In the third project, we propose an interactive method that allows users to guide stylization by drawing lines to control pattern direction, and painting regions on both the 3D surface and style image to specify where and how specific style patterns should be applied.

Through our extensive qualitative and quantitative evaluations, we show that our methods surpass state-of-the-art techniques. We also demonstrate their robustness across diverse 3D objects, scenes, and styles, highlighting the flexibility of the presented methods. Future work may explore extensions such as geometry modification for style-driven shape changes, more efficient large-scale pattern synthesis, temporal coherence in dynamic or video-based scenes, and refined interactive controls informed by direct artist feedback to better integrate creative intent into the stylization pipeline.

Contents

CHAPTER 1

Introduction

Computers have long been used in the process of creating art. This can take many forms, such as simulating painting tools, physical simulations, synthesizing sounds of musical instruments, producing repeating visual patterns, inbetweening, etc. However, creating art has been and largely still is specific to humans, i.e., it is still a challenging task for computers to replicate [Gar24]. Nevertheless, a few recent forays by automated systems—such as Stable Diffusion [RBL+22] that can generate pictures based on a text prompt—show that computers are capable of producing artifacts that human audiences could consider to be art.

Today, neural networks are successful at approaching and potentially even reaching the subjective quality that human-produced art holds. This was, for instance, demonstrated recently when an AI-generated image won an art contest [Kut22]. This can be largely attributed to the fact that an AI model's internal processes are not constrained by human-defined rules, but are based on a complex subsymbolic understanding of human-created art, which is used to train the neural networks [RBL+22, RDN+22]. By rehashing and combining what these neural networks are fed, computers have become capable of producing new works, a process reminiscent of a quote often attributed to Pablo Picasso: "*Good artists copy, great artists steal.*"

Advances in representations of 3D scenes that are suitable for machine learning have opened large areas of possible research, such as novel view synthesis [MST+20], text-guided object generation [SCL+22, JMB+22], and many others. Among those, we also encounter style transfer for 3D objects and scenes. Style transfer uses a style to change the content so that the appearance of the content changes to match the style, with the content being recognizable [EF01, HJO+01, GEB16]. In the context of 2D style transfer, *content* refers to the high-level concept of objects and their arrangement in a scene or image, while *style* refers to the visual patterns that build up the content [GEB16]. An example of style transfer is shown in Figure 1.1 where *The Scream* by Edvard Munch is used as the style to adapt *The Starry Night* by Vincent van Gogh.

Figure 1.1: Artistic 2D style transfer. Left: *The Scream* by Edvard Munch used as the style. Middle: *The Starry Night* by Vincent van Gogh used as the content. Right: The resulting stylized image produced with our own work presented in Chapter 5, adapted to 2D images.

This process is especially interesting for 3D objects and scenes, as it is often necessary to create a large number of simple objects that need to share a visual style. For instance, this is the case for game artists creating backgrounds or environmental objects. This process is both tedious and time-consuming, making it an ideal target for automation. Additionally, stylization could be used to stylize whole scenes so that artists can investigate whether a certain style suits their needs, and if the stylized scene is of sufficient quality, they can use it as a starting point for their work.

While relatively simple for 2D cases, using neural network-based approaches becomes much more complex for 3D objects and scenes. Suddenly, it is necessary to ensure multi-view consistency [MPSO18a, HJN22a] so that the patterns observed while moving in the scene stay coherent. It is also important to have a scene representation that is efficient and suitable for machine learning—not just in terms of memory and computational speed, but also in how well it supports gradient-based optimization. Therefore, care must be taken to ensure that the optimization process preserves the coherence and validity of the scene geometry, avoiding artifacts or invalid results that would degrade the quality of the output. This also ties in with computational demands, as 3D scenes tend to be orders of magnitude larger than 2D images. At times, this leads to necessary compromises when reasonable computational time has priority over the quality of stylization, thus making the process interactive at the expense of achieving the highest possible visual fidelity. Finally, while style transfer is usually a fully automatic process, in which the user needs only to choose the style and the content, adding optional guidance could be desirable. For example, users might want to pick and choose which styles should be used for which parts of the scene or be able to influence the direction of brushstrokes. This way, the result could better match the artists' vision of the final artwork.

1.1 Problem Statement

In this dissertation, we investigate style transfer for 3D objects and scenes with a focus on novel operators, representations, and interactivity. The work of Gatys et al. [GEB16] has long served as the basis for future research into artistic style transfer. While this paper

focuses on 2D style transfer, other papers appeared quite quickly that extended their approach to 3D. Visual patterns are inherently perceived as 2D by our visual system, which projects the 3D world around us onto the retina. As such, 3D style transfer that aims to transfer and synthesize these visual patterns and not just colors, needs to consider the 2D projection of the 3D scene to use the given patterns successfully. In literature, this is usually done by creating proxy 2D images, which are then used in concert with 2D style transfer methods to stylize the 3D scene. These proxy images are obtained by either rendering the scene [MPSO18a, HJN22a], a process similar to human vision, or by slicing the 3D volume [GRGH19a], thus *lifting the 2D methods to 3D*. However, as these are just proxies, they introduce either needless computational complexity or artifacts. By working directly with the curved surface, these difficulties could be resolved, potentially surpassing both kinds of approaches relying on 2D proxies. While seemingly trivial, the design space of neural surface operators is relatively large, and special care needs to be taken to be able to use them for the task of style transfer, in order to achieve a similar visual quality to that of the 2D precursors and still be fast and affordable to compute.

In recent years, novel view synthesis and scene reconstruction have seen large leaps in terms of quality, speed, and also accessibility, thanks to novel 3D representations and machine learning methods used to create them. Neural Radiance Fields (NeRFs) [MST$^+$20] utilize a neural network to learn the distribution of colors and density in 3D space, which can be reconstructed into images with traditional ray marching techniques. This work showed that optimizable volumetric representations are promising, and not long after, extensions of this method and other similar scene representations appeared. One notable example of them is the *Gaussian Splatting* representation [KKLD23], which uses colored Gaussians that are then splatted to achieve high reconstruction quality with a short optimization time and at the same time real-time rendering speeds. Suddenly, it became possible to quickly create high-quality scans of scenes or just single objects with relatively cheap equipment, opening many possibilities to both the regular audience and professionals.

Scanned objects may not directly suit the needs of users, therefore, they may want to further process them or interact with them. Using new data representations, in this case, 3D scene representations, always raises several important considerations. One key question is whether the new representations should be used throughout the entire pipeline or whether they should eventually be converted into a more traditional form. If a conversion is deemed necessary, it is also critical to determine the appropriate stage in the pipeline at which this transition should occur. Often, representations are not interchangeable, and transforming across them can lead to the loss of information and, therefore, to the loss of detail or to artifacts. To be able to answer the aforementioned questions, it is necessary to know the limits of the new representations and whether it is possible to use certain transformations with them, i.e., style transfer.

To apply style transfer methods, it is necessary to analyze the properties of the patterns and content and to use statistical methods to select which patterns to use where. However, this process is usually fully automated, and during stylization, the placement of patterns is more or less random and not predictable. While this may allow for quick prototyping

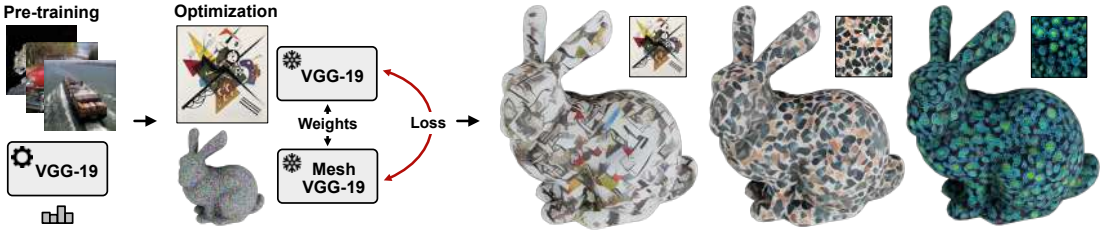


Figure 1.2: Our mesh texture synthesis algorithm [KHR24b] employs two neural networks with the same architecture: the first one is a conventional 2D CNN designed for images, while the second operates directly on the tangent space of a mesh. The shared architecture allows us to use the weights of the 2D CNN—pre-trained on thousands of natural images—on the CNN operating on the mesh. Consequently, with the parameters of the networks frozen, we optimize the content of a mesh texture to match the content of a specified image. This method is presented in Chapter 3.

or trying new styles in general, it may be necessary to have more control over the process to use the result. In the context of this dissertation, we refer to this control as *guidance*. Essentially, the user guides the stylization process, which at heart is still based on statistical methods and randomness, though the process can be constrained by the user. This guidance can take many forms, e.g., selecting which style to use where. However, existing methods are usually only 2D and can be difficult to extend to 3D, especially if they rely on transformations of flat 2D images, which may not be generalizable to 3D scenes.

1.2 Objective and Contributions

The primary objective of this dissertation is to explore and present the possibilities of 3D style transfer, focusing on the fundamentals and also on higher-level principles. The overarching research question that we aim to answer is the following:

How can we develop user-guided, 3D-aware style transfer methods that lift 2D techniques to modern 3D representations?

To address this question, we decompose it into the following focused research questions, each targeting a key component of the problem:

Q1: How do we lift a 2D texture synthesis method to 3D by working directly on the surface of a mesh? 3D style transfer methods tend to rely on either projecting or slicing the underlying 3D representation to get a proxy 2D image on which it is possible to apply 2D methods. However, this proxy image does not exactly capture the overall structure of the 3D object, i.e., it distorts areas, only parts of the whole object can be seen at once, points that are far apart can be close in the proxy image, etc. For surface-based representations, e.g., textured meshes, we could use the *curved surface* itself instead of the flat 2D proxy. We investigate neural surface operators and how they can be used to create textures for meshes. An example of this is shown in Figure 1.2 and further discussed in Chapter 3.

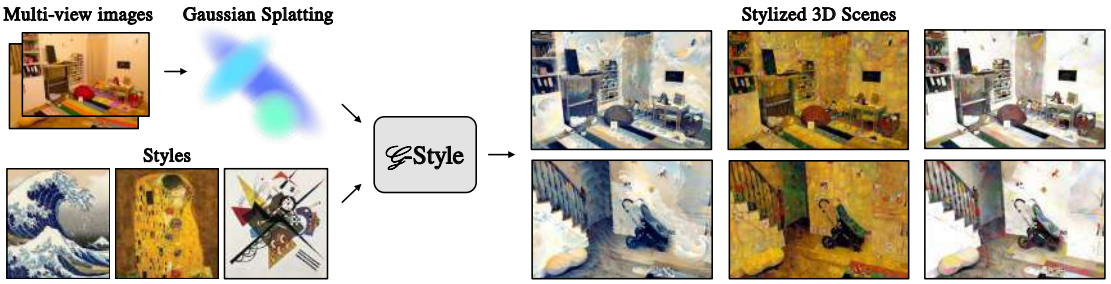


Figure 1.3: $\mathscr{G}$-Style [KHR24a]: Our method takes a 3D scene, represented using Gaussian Splatting, and a style image exemplar as input, and generates a stylized version of the scene that closely matches the visual style of the exemplar. By modifying the geometry of the scene and designing losses that capture style patterns at different scales, we achieve high-quality stylized scenes efficiently, with results generated in just a few minutes. This method is presented in Chapter 4.

Q2: How do we lift style transfer methods to modern volumetric representations? In recent years, volumetric representations have gained traction [MST+20, FKYT+22, KKLD23]. However, working with these representations is more complex than with meshes and requires special care. Furthermore, these volumetric objects and scenes are usually created by scanning real-world objects and scenes, which introduces noise and uncertainties. While it is possible to create a mesh from these representations, this is costly and it may cause artifacts or lead to the loss of detail. As such, style transfer methods need to be extended to these volumetric representations while accounting for the noisiness present in the data. We investigate this by proposing a method primarily meant for Gaussian Splatting, which, at this time, is one of the most promising volumetric representations, and account for the specifics of this representation and its limitations. An example of this is shown in Figure 1.3 and further discussed in Chapter 4.

Q3: How do we empower the user to guide the stylization process? Style transfer means that the style features of the style input are used to construct the content of the content input. This usually corresponds to matching higher-order statistics of patterns and features [GEB16], where different style transfer methods match different statistics of different patterns and features, and where the matching is done in different ways. However, the user might want to guide the process in order to match their vision and not just rely on an automated method. We investigate different ways of modifying style transfer methods to allow the user to guide the stylization process, emphasizing easy and intuitive interactivity. An example of this is shown in Figure 1.4 and further discussed in Chapter 5.

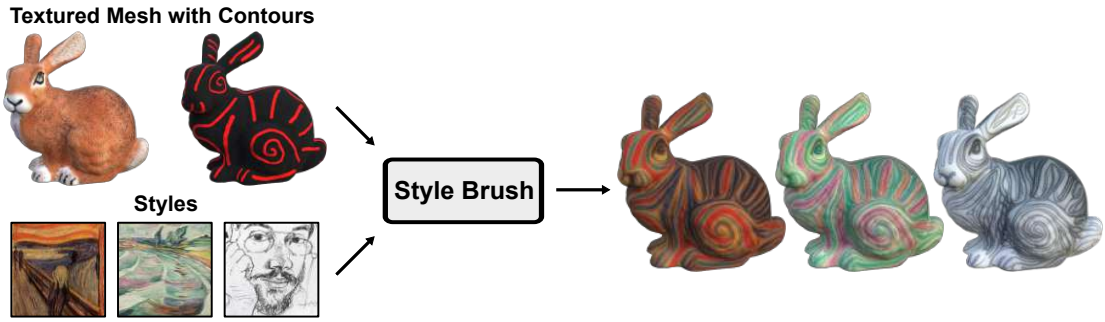


Figure 1.4: Style Brush takes a textured mesh, an additional texture with user-determined contours, and one (or multiple) style images as input. By optimizing the input texture and guiding the synthesized patterns with the contours, our method generates high-quality, stylized textures in just a few minutes that faithfully adhere to the directions specified by the user. Here, we demonstrate Style Brush using multiple style images, with each one producing a stylized texture; however, it also supports combinations of multiple styles. This method is presented in Chapter 5.

Respectively, the contributions of our work are as follows:

Surface-aware Mesh Texture Synthesis with Pre-trained 2D CNNs. In Chapter 3, we present a method to do texture synthesis and style transfer directly on the surface of a mesh (Figure 1.2). We accomplish this by taking a foundational 2D texture synthesis and style transfer method that utilizes a pre-trained neural network and replacing 2D convolutional and pooling layers with carefully designed surface-based ones, making adjustments and corrections as necessary to account for the domain change. We show that a surface-based approach for 3D texture synthesis and style transfer can perform similarly or better than the alternative 3D methods that utilize a differentiable renderer or slicing to obtain proxy 2D images, which are then optimized with 2D methods.

$\mathscr{G}$-Style. We propose a method to stylize 3D scenes represented with Gaussian Splatting, which we present in Chapter 4 (Figure 1.3). This method can transfer both high-frequency and large-scale patterns by utilizing a combination of different losses while accounting for and correcting the artifacts specific to Gaussian Splatting, namely, large Gaussians and elongated Gaussians. To our knowledge, this was the first published style transfer method for Gaussian Splatting that was capable of transferring large-scale patterns and not just adjusting colors or transferring small patterns.

Style Brush. Finally, in Chapter 5, we present a method for guided style transfer for meshes (Figure 1.4). *Style Brush* is an interactive approach for artistic 3D style transfer, allowing the user to specify the directions of directed features, e.g., brush strokes or pencil strokes, by drawing simple guiding lines. Furthermore, users can specify which parts of the style image should be applied to which parts of the mesh by painting regions on the mesh and the style image to indicate which parts should be used where. To our knowledge, this method is the first proposed 3D method that allows guiding the directions of patterns in the context of neural style transfer.

1.3 Dissertation Overview

The research projects presented in this dissertation were conducted during my doctoral studies at TU Wien. Two of them resulted in a scientific publication, and the last one was under review during the writing of this dissertation. All publications were co-authored with Renata G. Raidou, my primary supervisor, and Pedro Hermosilla, my co-supervisor, with me as the first author. Chapters 3, 4, and 5 are based on the following papers:

- Áron Samuel Kovács, Pedro Hermosilla, and Renata G. Raidou. "Surface-aware Mesh Texture Synthesis with Pre-trained 2D CNNs." Computer Graphics Forum. Vol. 43, No. 2, p. e15016, 2024. DOI: https://doi.org/10.1111/cgf.15016
- Áron Samuel Kovács, Pedro Hermosilla, and Renata G. Raidou. "$\mathscr{G}$-Style: Stylized Gaussian Splatting." Computer Graphics Forum. Vol. 43, No. 7, p. e15259, 2024. DOI: https://doi.org/10.1111/cgf.15259
- Áron Samuel Kovács, Pedro Hermosilla, and Renata G. Raidou. "Style Brush: Guided Style Transfer for 3D Objects." Under review, 2025. Preprint available at: https://arxiv.org/abs/2510.03433

CHAPTER 2

Related Work

This dissertation builds upon advances in texture synthesis and style transfer, mainly for 3D objects and scenes. In this chapter, we provide an overview of these two topics, focusing on methods for both 2D and 3D cases. In Sections 2.1 and 2.2, we discuss 2D and 3D approaches, respectively. Section 2.4 discusses how to measure similarity between styles of different images. Finally, Section 2.5 identifies the gap in style transfer literature we aim to address in this dissertation.

In literature, a "texture" can refer to two close but separate concepts. The first one refers to flat 2D images representing the surface appearance or a pattern of material, e.g., grass, bricks, fabric, etc. The second one refers to those images that are mapped to the surface of 3D objects to define visual characteristics such as color, normals, etc. For the sake of consistency with the existing literature, this chapter uses the same term to refer to both concepts.

2.1 2D Approaches

Textures can be modeled with Markov Random Fields (MRFs), meaning that a texture is treated as a realization of a random process that is both local and stationary [WLKT09]. The locality property indicates that each pixel's value depends only on its neighborhood and is independent of distant pixels. On the other hand, stationarity indicates that parts of the texture appear the same regardless of their position. From the MRF viewpoint, the task of texture synthesis is to synthesize an output texture, given an input texture, so that for each of its pixels, the pixel's spatial neighborhood matches or is similar to at least one neighborhood from the input texture [WLKT09].

This approach to modeling textures is particularly well-suited for generating new textures by resampling existing ones. Efros et al. [EL99] propose an approach where an initial seed region, a 3×3 square taken from the input, is grown pixel by pixel. To synthesize a new pixel, they create a candidate set of samples from the input that matches the partial

neighborhood of already synthesized pixels within a certain tolerance. A pixel is randomly chosen from this candidate set as the new pixel, and this process repeats until they synthesize the whole texture. Furthermore, they propose a variation on their approach for hole filling, where the hole is closed from the boundary inward. Wei et al. [WL00] note that the shape of neighborhoods of the method by Efros et al. [EL99] varies and propose to synthesize the texture in a scan line order, from top left to bottom right and using an L-shaped neighborhood, which considers pixels to the top and to the left. This allows them to optimize the search for the best-matching neighborhood, thus speeding up the process. Instead of synthesizing the texture one pixel at a time, another alternative is to take entire patches from the input textures. However, when patches are simply placed next to each other, they may have conflicting borders, thus, it is necessary to deal with these sharp edges. Liang et al. [LLX+01] partially overlap the selected patches and blend the overlapping regions, which can result in blurry transitions. Efros et al. [EF01] use dynamic programming to find a path through overlapping patches along which to cut the overlapping regions, which Kwatra et al. [KSE+03] further improve using graph cuts. Such techniques are capable of producing natural textures very efficiently—yet, they do not provide an actual texture model. A complete review of MRF-based techniques is provided by Wei et al. [WLKT09].

Instead of resampling existing textures, another possibility is to explicitly define a parametric texture model. Julesz [Jul62] models textures using n-th order joint probability distributions of pixel values. Heeger et al. [HB95] introduce a texture synthesis method that analyzes the input texture to compute first-order statistics of its color distribution and responses to a steerable pyramid, a multi-scale, multi-orientation filter bank. Starting from white noise, their method iteratively modifies the output by matching histograms alternatively in the image domain and in the steerable pyramid domain. Portilla et al. [PS00] propose a parametric model that encodes textures via joint statistics of the coefficients from an overcomplete complex wavelet transform. Although these approaches are effective for a wide range of textures, they are not sufficient in representing natural textures [GEB15a].

Image and texture generation approaches can be categorized into two main groups within the context of our research. The first category involves the generation of new textures with high-level features, derived from an exemplar. The second category further tries to preserve the content of an additional input image. Often, both categories share similar methodologies, requiring only minor modifications before being used interchangeably. In both cases, the goal is to reuse features from the exemplar, which can be constrained by features in the content image, if desired.

The foundational work in neural style transfer has been laid by Gatys et al. [GEB16], where the technique was developed to transfer artistic styles, which can be seen in Figure 2.1. Their approach is based on extracting features from a pre-trained VGG-19 network [SZ14] and using them to transfer the style and also to preserve the original content. They represent style as Gram matrices extracted from several hidden layers of the network. They compute these Gram matrices from the style image and the optimized

image, and then they minimize the mean squared distance between the two sets of Gram matrices. The Gram matrix $G \in \mathbb{R}^{\mathbb{C}\times\mathbb{C}}$ of a flattened feature map $\mathcal{F} \in \mathbb{R}^{C\times N}$, where C is the number of channels and N is the number of spatial locations, is given by:

$$G_{ij} = \sum_k \mathcal{F}_{ik}\mathcal{F}_{jk} \tag{2.1}$$

In order to keep the semantics of the original content, Gatys et al. minimize the mean squared distance between the feature maps of the original content image and the optimized image. This concept has been since repurposed for synthesizing a texture by using a noise image as the content input [GEB15a]—thus, not preserving any content. The approach of Gatys et al. served as an important starting point and inspired many follow-up papers. Note that 3D approaches can be used for 2D, which is possible by using a flat plane as the 3D object. While some of the presented methods have contributions specific to them using 3D objects, the main contributions of some can be used in the context of 2D synthesis. These methods can be roughly categorized into two main groups.

The first category of approaches is optimization-based. They optimize the initial image using a variety of different losses to better preserve certain aspects of the input images, possibly using different neural architectures. These include transferring features on multiple scales [GCLY18], utilizing Generative Adversarial Networks (GANs) [JBV17], or diffusion models. Zhang et al. [ZHT+23] use an attention-based textual inversion method that learns high-level textual descriptions from a single painting image. These descriptions are then used to guide a text-to-image generative diffusion model to produce images with specific artistic appearances. Wang et al. [WZX23] propose an approach for interpretable and controllable content-style disentanglement and style transfer that employs diffusion models to explicitly extract content and implicitly learn style. They introduce a CLIP-based [RKH+21] style disentanglement loss combined with a style reconstruction prior to enhance disentanglement and improve style transfer performance. CLIP stands for *Contrastive Language-Image Pre-training* and it refers to a technique of training a pair of neural networks. One network processes text, the other network processes images, and they encode their inputs into a shared embedding space, where matching text-image pairs are close together, and non-matching pairs are far apart. Chung et al. [CHH24] introduce a style transfer method that uses a pre-trained large-scale diffusion model without requiring optimization, by manipulating self-attention features through cross-attention key and value substitution between content and style images. Their approach enables content preservation and local texture-aware style transfer, enhanced by techniques such as query preservation, attention temperature scaling, and an AdaIN-based [HB17] initialization to deal with disharmonious colors.

While some directly base their losses on the original approach by Gatys et al., i.e., matching Gram matrices between images, other works search for nearest neighbors in the feature space to minimize the distance between them. Li et al. [LW16] use a combination of generative Markov random field models and discriminatively trained deep CNNs. However, artifacts might occur when style and content images consist

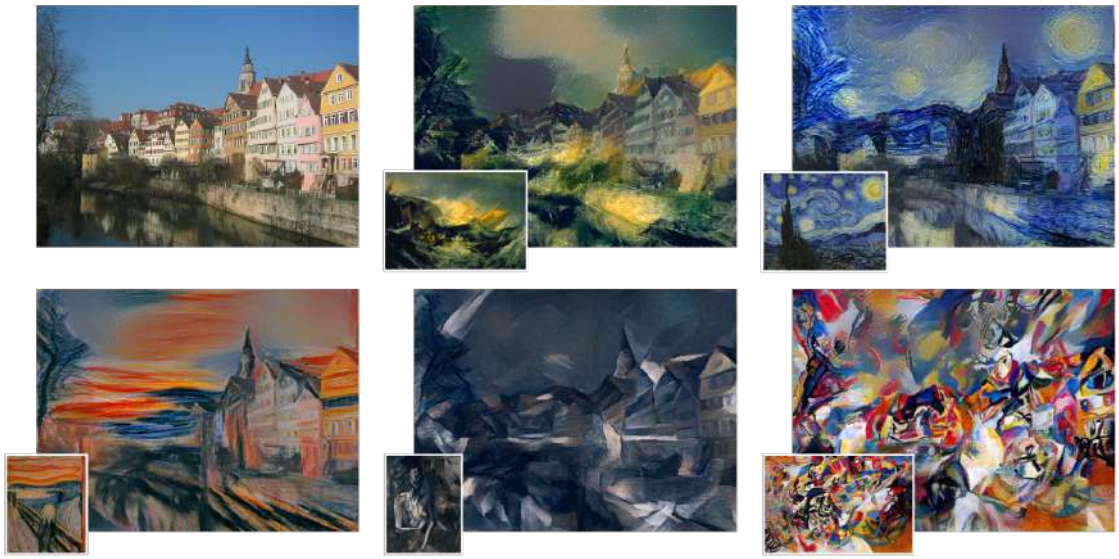


Figure 2.1: Stylized images from the foundational paper by Gatys et al. The content image (top left) is iteratively optimized, so that the Gram matrices of the content image and a given style image (bottom left of each synthesized image) match. Images taken from Gatys et al. [GEB16], licensed under CC BY-NC-ND 4.0.

of differently shaped elements, or if they are different due to the changes in size or perspective. Chen et al. [CS16] find closest patches in feature space using the normalized cross-correlation measure and then using an optimization-based process, create an image that approximately corresponds to the new features. This method also aims to perform style transfer for videos, which is feasible due to its speed, however, it can occasionally lead to local flickering artifacts due to certain simplifications, which, however, contribute to its speed. Kolkin et al. [KSS19] utilizes a style loss based on the earth mover's distance, however, this method is relatively slow due to the computational demands of their loss. Liao et al. [LYY+17] find semantically-meaningful dense correspondences between two input images using extracted neural features. This method relies on a pre-trained network to compute the correspondences and can lead to artifacts or mismatches if the pre-trained network does not recognize certain objects or may struggle to find accurate correspondences in textureless regions due to low neural activations. Zhang et al. [ZKB+22a] find nearest neighbors in the VGG-16 [SZ14] feature space, and the style loss corresponds to the difference between the optimized features and the found nearest neighbor features, this process is called nearest neighbour feature matching (NNFM). The NNFM loss requires computing the distances between all pairs of extracted features, which can lead to poor performance for high-resolution images due to its quadratic scaling. This approach is extended by Zhang et al. [ZFLS24], where the nearest neighbors are selected by combining the original VGG-16 features with features from a network trained for semantic segmentation, LSeg [LWB+22], but only the distance between the VGG-16 features is used for optimization. Similar to the work of Liao et al. [LYY+17], LSeg can fail to recognize similar classes of objects.

The other group of approaches explicitly minimizes the loss function in a single feed-forward pass. Chen et al. [CS16] describe a modification to their method, where instead of optimizing to find an image that corresponds to certain style activations, they train an inverse network to do so in a single step. Huang et al. [HB17] use an adaptive instance normalization layer that modifies the content features by matching their mean and variance to those of the style features, enabling arbitrary style transfer. An et al. [AHS+21] use reversible neural flows and an unbiased feature transfer module to prevent content leak during style transfer. They are generally faster than their optimization-based counterparts, however, some may require lengthy pre-training for each style, and their results tend to be of lower quality than the optimization-based approaches.

2.2 3D Approaches

3D style transfer refers to modifying the appearance of a 3D object or a scene so that when viewed from different angles, it matches the style of a given exemplar. Existing approaches usually rely on a single way of representing objects and whole scenes, which is also the main point of influence for their performance. The approach of Cao et al. [CWNN20] uses point clouds, yielding holes due to measurement errors during scanning and being unsuitable for representing surfaces and real-world scenes. Conversely, Mordvintsev et al. [MPSO18a] and Höllein et al. [HJN22a] use textured meshes, which can be reconstructed from the aforementioned point clouds.

To reuse a 2D style transfer method, one needs to extract and employ a 2D image of a given scene. This can be either obtained by utilizing a differentiable renderer or by slicing the 3D volume. When the style image aims to represent a material that can be represented with a volume, such as wood or marble, solid texture approaches can be used. Gutierrez et al. [GRGH19a] slices the 3D volume, and uses the slices as proxy 2D images to which they apply a perceptual style loss. Similarly, Portenier et al. [PABG20] use a similar strategy, but also use a GAN to increase the quality of generated results and fine-tune the parameters of their model to better generalize to unseen data. Zhao et al. [ZGW+22] make further improvements by utilizing multi-scale 2D texture discriminators. Solid texture methods directly assign values, such as color or density, to every point in 3D space that can be sampled onto a mesh. Hence, during the synthesis part of the pipeline, these methods are not aware of the mesh's surface. Not all materials or styles can be represented with a solid texture that exhibits a high degree of symmetry, as features are correlated based on Euclidean distance and do not consider the separation on the surface.

To address these shortcomings, render-based approaches adopt a process of differentiable rendering objects and subsequently matching the style of the projected parts. Mordvintsev et al. [MPSO18a] use a differentiable renderer in a straightforward way, showcasing that rendering a 3D object and then applying a simple style loss produces textures of similar quality to 2D images, but in 3D. Höllein et al. [HJN22a] extend this approach to scenes by utilizing a depth and angle-aware optimization, accounting for surfaces that may not be aligned to the camera's viewing plane, and that objects may have different screen

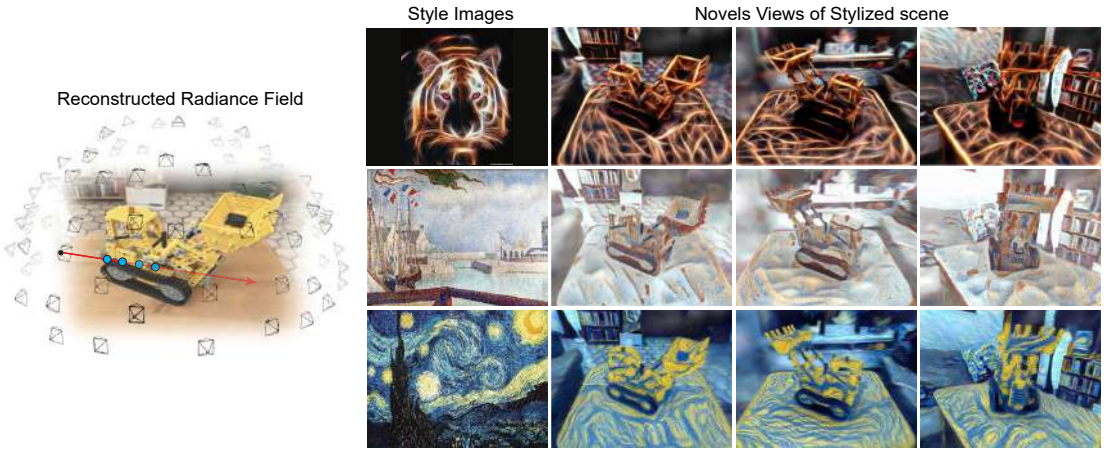


Figure 2.2: Stylized 3D scene. The scene is rendered from multiple viewpoints, and the rendered images are used to compute a style loss using neural features from given style images. The gradients from the loss are then backpropagated to the underlying scene representation. Figure taken from Zhang et al. [ZKB+22a], used with permission from Springer Nature Switzerland (© 2022 Springer Nature Switzerland).

space sizes depending on the distance from the camera. The depth-aware stylization is accomplished by rendering depth images, detecting patches that have similar depth, and then constructing a multi-scale pyramid, effectively pairing the patches to pyramid levels. For angle-aware optimization, they calculate the normal-to-view angle for each pixel, threshold it, and apply fine stylizations only to those below the threshold. Such methods require rendering the object from multiple viewpoints, thereby introducing challenges when dealing with objects of high complexity, with multiple holes, or overlapping wire-like features, where achieving visibility of all surfaces becomes difficult.

Zhang et al. [ZKB+22a] and Zhang et al. [ZFLS24] utilize more advanced style losses as described in Section 2.1, however, without Höllein et al.'s corrections. The results of the Artistic Radiance Fields by Zhang et al. [ZKB+22a] can be seen in Figure 2.2. When the camera is suitably positioned, the projection process can effectively capture the separation of surfaces by empty space. Consequently, these methods can break correlations between surfaces that are close in Euclidean distance but far apart along the surface, preventing style patterns from leaking across disconnected regions.

Alternatively, methods that do not wish to utilize a differentiable renderer or use representations that cannot be sliced may need to use differentiable operators specific to the used representation. Enabling neural networks to directly process meshes is still an active area of research. While vertex-based approaches that represent the mesh as a graph exist, they cannot easily capture data not solely associated with individual vertices or faces. Li et al. [YHSG17] propose a spectral CNN method that achieves weight sharing by expressing convolutional kernels in the spectral domain defined by graph Laplacian eigenbases. They further develop a spectral parameterization for dilated

convolution kernels and introduce a spectral transformer network, which together enable multi-scale analysis and allow learned coefficients to be shared across related yet structurally different shapes represented by different graphs. Verma et al. [VBV18] present a graph-convolutional operator designed to find correspondences between filter weights and neighborhoods in a graph regardless of the graph's connectivity. These correspondences are computed dynamically based on features learned during training. Fey et al. [FLFM18] utilize spline-based graph convolutional layers that employ B-splines. This way, their kernel functions are continuous but are parametrized with a fixed number of trainable parameters. Hanocka et al. [HHF+19] use mesh convolutional and pooling layers. Their convolutional layers process edges together with the four edges adjacent to it within its two incident triangles, and the pooling operator collapses edges while still retaining the surface topology.

Other approaches have defined the convolution operations directly on the Riemannian manifold, which can be categorized into two groups. The first one uses a diffusive approach related to heat diffusion on surfaces. Sharp et al. [SACO22] address deep learning on surfaces by integrating three key components: per-point MLPs that model scalar functions over feature channels, a learned diffusion operation for propagating information across the surface, and local spatial gradient-based features to enable filters that are not limited to radial symmetry. These elements together make the network robust to variations in surface resolution and sampling, and even allow generalizing it to point clouds. Monti et al. [MBM+17] propose *mixture model networks*, a unified and flexible framework for constructing convolutional neural networks on non-Euclidean domains such as graphs or manifolds. The primary contribution of their method is a parametric patch operator based on learnable pseudo-coordinates and Gaussian kernels. Masci et al. [MBBV15] propose geodesic convolutional neural networks, which operate directly on non-Euclidead manifolds by extracting local geodesic patches using polar coordinates. These patches are then passed through trainable layers to capture invariant shape features.

A different approach is to use equivariant convolutions on surfaces that are designed to address the rotation ambiguity problem of the tangent plane. Poulenard et al. [PO18] propose convolutional layers for processing signals on curved surfaces, introducing directional functions that can maintain the directional ambiguity, and then resolving it in the final layer by taking the maximum response. Yang et al. [YLP+20] introduce a convolutional neural network architecture that enables effective feature learning on surface meshes by using parallel frames to mimic standard Euclidean convolutions, which is accomplished by aligning tangent planes through locally flat connections. Mitchel et al. [MKK21] present field convolutions, a surface convolution operator for vector fields that combines intrinsic spatial convolutions with parallel transport, allowing each neighbor to describe the position of a point in its own local coordinate frame, resulting in noise-robust convolutions.

Lately, Neural Radiance Fields (NeRFs) [MST+20], TensoRF [CXG+22] Gaussian Splatting [KKLD23] have become very prominent in reconstructing objects and scenes. Unlike

point clouds and meshes, NeRFs, TensoRF, and Gaussians are soft volumetric representations. NeRFs use a neural network to estimate color and density at each point in 3D space, and then use traditional ray marching to render the scene. The original NeRFs are slow to train and to render, as such, considerable effort has been put into improving their performance, e.g., using a multi-scale representation [BMT+21] or using multi-resolution hash positional encoding [MESK22]. However, neural networks are black boxes, can be difficult to interpret, and may face problems with finding good local minima, therefore, methods that do not utilize neural networks can be attractive if they can avoid these pitfalls. TensoRF [CXG+22] represents scenes as tensors that can be factorized into multiple compact low-rank tensor components, achieving significantly lower memory footprint and fast optimization times. Gaussian Splatting [KKLD23] uses 3D Gaussians, which are then rendered by splatting, enabling real-time rendering, fast optimization time, with the resulting representation being easily interpretable and easy to interact with. Naturally, these representations can be utilized for scene reconstruction and novel view synthesis, which was the primary task for them, but also for many downstream tasks, including style transfer.

Like in the 2D case, these style transfer methods can be divided into two categories: zero-shot and iterative methods. Zero-shot methods focus on matching colors, relighting, or transferring details on a small scale to achieve multi-view consistency. Liu et al. [LZL+23] propose Universal Photorealistic Style Transfer, an approach that incorporates StyleNet, a network for per-instance style transfer without the need to pre-train on paired datasets, and an instance-adaptive optimization strategy that uses an adaptive coefficient to balance between content preservation and stylization strength. Liu et al. [LZC+23a] introduce a 3D style transfer method, StyleRF that addresses the trade-off between accurate geometry, high-quality stylization, and generalizability by operating in the feature space of a radiance field. StyleRF introduces two key innovations: a sampling-invariant content transformation to ensure multi-view consistency, and a deferred style transformation on 2D feature maps to reduce memory usage while maintaining stylization fidelity. Saroha et al. [SGC+24] propose a method for stylized 3D rendering by combining Gaussian Splatting with a multi-resolution hash grid and a lightweight MLP conditioned on style codes, enabling flexible style generalization at test time. Liu et al. [LZX+24] use Gaussian Splatting and embed VGG features into the Gaussians, which are later used to render a stylized image. These methods struggle to synthesize large patterns that span significant portions of a given scene due to their inability to quickly and consistently embed large features into scenes.

On the other hand, iterative methods leverage rendering scenes from multiple viewpoints. Hence, they can construct large-scale patterns that remain consistent across different views, using a relatively slow optimization process, where the patterns are progressively built or dissolved to reach consensus across many different viewpoints or slices [HJN22a].

In this dissertation, we primarily classify 3D representations into two groups, they are either *surface-based* or *volumetric*. Surface-based representations describe only the boundary of an object, e.g., meshes or parametric surfaces. On the other hand,

volumetric representations encode information throughout the interior, such as voxel grids [FKYT+22, CXG+22], 3D Gaussians [KKLD23], or NeRFs [MST+20]. As an alternative perspective, we also note a classification described by Guo et al. [GWHM24], which frames geometry representations as either *Eulerian* or *Lagrangian*. Eulerian methods describe geometry on a fixed spatial grid or via continuous fields, e.g., voxel-like representations [FKYT+22, CXG+22] or NeRFs [MST+20], associating each coordinate with properties like occupancy, density, and possibly additional data like color. They are popular, but face a compute-quality trade-off, capturing fine details demands either high-resolution grids or large neural networks, both costly and often insufficient for thin structures. In contrast, Lagrangian methods track a set of geometry primitives (e.g., triangles that are possibly connected into meshes or 3D Gaussians [KKLD23]) that move in world space. They typically use fewer resources for detailed shapes, however, because these primitives may move independently during optimization, Lagrangian representations can suffer from artifacts and can even result in invalid shapes (e.g., non-manifold meshes), or get stuck in local minima, which may be unsuitable for high-fidelity rendering or simulations.

For completeness, we also want to highlight several alternative 3D representations. Point clouds are unordered sets of 3D points, often with per-point attributes such as color or normals. They do not specify an explicit surface and are usually the output of photogrammetry or 3D scanning [Ull79, CCL+18]. Volumetric meshes partition a solid's interior into discrete elements, e.g., tetrahedra or hexahedra, encoding material properties of both the surface and the interior and enabling volumetric operations like computing internal stresses or forces throughout the object [PCS+22]. Spatial partitioning trees, e.g., binary space partitioning trees (BSP trees), octrees, bounding volume hierarchies (BVHs), k-d trees, hierarchically organize 3D space, storing node bounds, occupancy information, or lists of primitives at nodes. They are useful for accelerating spatial queries, e.g., ray intersections, or collision detection, and for sparse storage [CNS+11, WM19]. However, beyond this overview, we do not make use of these representations in this dissertation.

Several recent works on style transfer for volumetric representations have been published following the completion of the main body of research presented in this dissertation, they are included here for completeness and to contextualize recent developments in the field. Galerne et al. [GWRM25] introduce a method that aims to enable ultra-high resolution style transfer. As they deem the traditional style losses insufficient for this task, they introduced a simultaneously optimized scales loss, which should enable style transfer across all scales at once. In their framework, Liu et al. [LLY+25] employ a controllable matching phase that uses segmentation masks to precisely align scene content with style features. Furthermore, they also define a style transfer loss based on feature alignment to ensure the global style is accurately captured. Zhuang et al. [ZHZ+25] introduce a framework combining multi-modal style conditioning, multi-level semantic alignment, and enhancements for perceptual quality. They utilize Stable Diffusion's latent space [RBL+22] for semantic alignment, adopt a Contrastive Style Descriptor for

texture transfer that is both localized and content-aware, employ the simultaneously optimized scales loss. Furthermore, they use a differentiable aesthetic prior that was trained on human-rated data to reduce artifacts.

2.3 Guidance

Style transfer *guidance* means that the user can meaningfully influence the stylization process. This can take many forms. In the context of artistic style transfer, the simplest is selecting which style images to use and optionally applying simple transformations such as cropping or masking to select specific patterns. This is used in the work of Zhang et al. [ZFLS24] to stylize 3D scenes based on semantic-similarity maps. Another possibility is to use semantic label maps. By using the semantic layout as input and stacking convolutional, normalization, and nonlinear layers, it is possible to synthesize realistic-looking images, with various features, such as "rock", "tree", or "cloud", at user-defined locations [IZZE17, WLZ+18]. This can be further enhanced by including a spatially adaptive, learned transformation to modulate activations [PLWZ19], the results of which can be seen in Figure 2.3. Another form of guidance is to direct features so that the user can define the direction of brush strokes, pen strokes, pencil strokes, etc. This can be accomplished by training a neural network to detect the directions and building a loss function with it to align the flow of patterns with the user-defined flow [WSZL19]. Another possibility is to use reversible content transformations to adjust the orientation of directed patterns, which can also be used for other transformations such as scaling [RBS+22] as shown in Figure 2.4.

2.4 Style Similarity

In the context of this dissertation, *style similarity* refers to how *styles* of different images are similar, regardless of their *content*. Reasoning about style similarity is crucial when comparing different approaches and the quality of their results. As style loss functions are directly used by neural network–based approaches to transfer styles, they can also serve as a basis for comparing different methods, with lower losses indicating a greater style similarity. However, while these loss functions may mimic human perception, they do not perfectly match it [PNC+23].

Jing et al. [JYF+19] note in their 2019 survey that the evaluation of style transfer algorithms is an open problem. Their qualitative evaluation relies on *the aesthetic judgments of observers*, while their quantitative evaluation uses the following five metrics: stylization speed, training time, average loss, loss variation, and style scalability. *Stylization speed* measures the time required to create a stylized image. *Training time* captures how long it takes to train the model used for stylization. *Average loss* measures how well the loss function is minimized across different content images. *Loss variation* measures the changes in loss during training to indicate how quickly methods converge. Finally, *style scalability* measures how well the method works for different styles. Of the quantitative

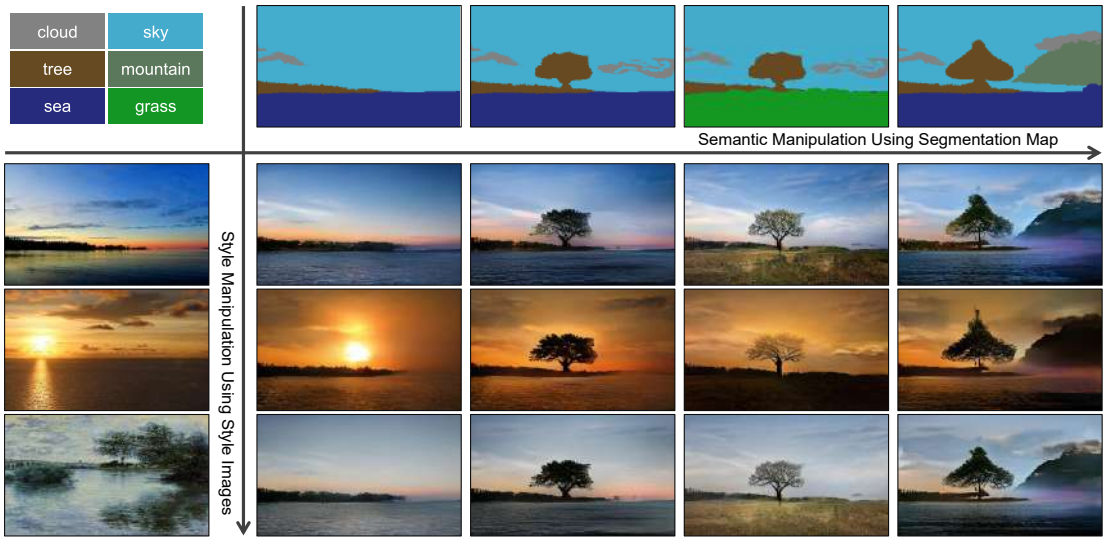
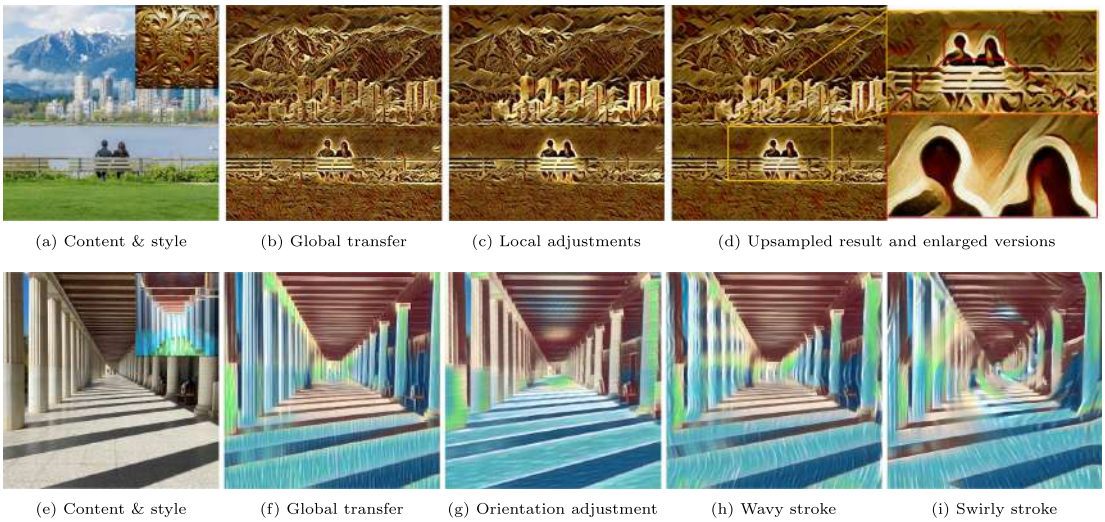


Figure 2.3: Semantic label guidance with style images. The semantic content (e.g., the sky, a tree, etc.) is controlled via a label map (top row), while the style is controlled via the style image (left column). Figure taken from Park et al. [PLWZ19], used with permission from IEEE (© 2019 IEEE).

metrics, only *average loss* and *style scalability* can capture the visual difference between the results of multiple methods, as the other metrics can only capture computational performance. However, as we already noted, the loss comparison metric is limited by the fact that loss functions do not perfectly correspond to human perception, and style scalability may not matter if the focus is only on a few styles or if the method can be fine-tuned to new styles. A recent preprint of a survey by Zhang et al. [ZT25] makes a similar observation, i.e., there is no universally accepted perceptual metric.

Nevertheless, some metrics have been proposed to enable an objective comparison between various style transfer methods. Yeh et al. [YTB+20] propose a metric that relies on two statistics: *effectiveness* to measure the extent to which a style has been transferred using Gram matrices extracted from a pre-trained VGG-16 [SZ14], and *coherence* to measure the extent to which the original content is preserved using extracted contours. Wang et al. [WZC+21] decompose the quality of style transfer into three quantifiable components: *content fidelity*, *global effects*, and *local patterns*. *Content fidelity* measures how well the content is preserved at multiple scales. The *global effects* metric between the synthesized image and the style image refers to the similarity between global colors, measured as the similarity between histograms, and holistic textures, expressed as the similarity between Gram matrices extracted from neural networks. Finally, the *local effects* metric measures how similar small patches are to the style image's patches and how varied they are. Wright et al. [WO22] combine the Fréchet inception distance [HRU+17] between the generated image and the style image with content preservation to obtain their metric. Although our literature survey suggests that none of these metrics are



(a) Content & style (b) Global transfer (c) Local adjustments (d) Upsampled result and enlarged versions

(e) Content & style (f) Global transfer (g) Orientation adjustment (h) Wavy stroke (i) Swirly stroke

Figure 2.4: Adjustable style transfer. Content and style images (**a** and **e**) are stylized so that the content is recreated with the features from the style images (**b** and **f**). This stylization can be locally adjusted by increasing stroke size and intensity (**c**), which is further improved by style-guided upsampling to get a higher resolution version of the stylized image (**d**). Reversible content transformation can be used to adjust the orientation of brush strokes (**g**), to produce wavy brush strokes (**h**), or to create swirly brush strokes (**i**). Figure taken from Reimann et al. [RBS+22], licensed under CC BY 4.0.

universally used to compare style transfer methods, we find that the metrics overall tend to focus on small-scale visual patterns and on content preservation, though their approach to measuring those differs.

In light of these findings, we have to conclude that a proper evaluation of style transfer methods is difficult, as there is no consensus about the metrics to be employed or even an agreement on how to judge the similarity. While performance characteristics, i.e., time and memory requirements, are easy to obtain and compare, the visual quality of results is not. Therefore, in this dissertation, we follow suit of the aforementioned style transfer papers, such as the works of Höllein et al. [HJN22a] or Zhang et al. [ZKB+22a], and rely on subjective qualitative comparisons and user studies to compare and validate the visual quality of our results.

2.5 Identified Literature Gap

While reviewing the literature that is related to our work, we have identified the following gaps that we aim to bridge with this dissertation, corresponding to the research questions posed in Chapter 1.

When using 2D methods for 3D style transfer, it is necessary to project from 3D to 2D. Existing works accomplish this by either slicing the volume [GRGH19a, PABG20, ZGW+22]

or by rendering it [MPSO18a, HJN22a, ZKB+22a, LZX+24, SGC+24, ZFLS24]. We note that for representations that directly describe the surface, e.g., meshes, there exists another method of projection: treating the non-flat surface as locally flat and directly applying the 2D methods and then, if necessary, making adjustments to accommodate for the differences between them. This is the gap that we aim to close with the work described in Chapter 3.

Furthermore, we observe that Gaussian Splatting is a Lagrangian representation, i.e., it uses individual Gaussians to represent whole 3D objects or scenes [KKLD23]. However, existing methods for style transfer for Gaussian Splatting [LZX+24, SGC+24] do not add more Gaussian nor modify the geometry of the existing ones, which results in low-resolution stylizations. Adding more Gaussians is a trade-off: each added Gaussian takes more memory but is then able to represent more details. For high-quality stylizations that are still efficient to render, it is necessary to add new Gaussians, but in a controlled manner. Moreover, existing approaches for Gaussian Splatting are not able to accurately transfer artistic patterns and resemble more color transfer than style transfer [LZX+24, SGC+24]. This is the gap that we aim to close with the work described in Chapter 4.

Lastly, we are not aware of any user-guided direction-aware style transfer method for 3D scenes. Direction-aware style transfer methods are those that take into account *directed features*, e.g., brush strokes. We also find existing approaches for 2D images to be lacking. The approach by Wu et al. [WSZL19] relies on a pre-trained direction estimation network to force the flow of features, however, the feature extractor is not directly informed about the directions, leading to its inability to synthesize directed patterns corresponding to the given style image. The method proposed by Reimann et al. [RBS+22] attempts to address this by applying reversible transformations to the content image. While they can correctly synthesize directed patterns, their results fall short of expectations, and the patterns may not correspond to the used style image and may be blurry in certain areas. Furthermore, there is no straightforward way of extending their approach to 3D. This is the gap that we aim to close with the work described in Chapter 5.

CHAPTER 3

Surface-aware Mesh Texture Synthesis with Pre-trained 2D CNNs

This chapter is based on the following publication:

> Áron Samuel Kovács, Pedro Hermosilla, and Renata G. Raidou. "Surface-aware Mesh Texture Synthesis with Pre-trained 2D CNNs." Computer Graphics Forum. Vol. 43, No. 2, p. e15016, 2024. DOI: https://doi.org/10.1111/cgf.15016

Mesh texture synthesis is a key component in the automatic generation of 3D content. Existing learning-based methods have drawbacks—either due to disregarding the shape manifold during texture generation or due to requiring a large number of different views to mitigate occlusion-related inconsistencies. In this chapter, we present a novel surface-aware approach for mesh texture synthesis that overcomes these drawbacks by leveraging the pre-trained weights of 2D CNNs with the same architecture, but with convolutions designed for 3D meshes. Our proposed network keeps track of the oriented patches surrounding each texel, enabling seamless texture synthesis and retaining local similarity to classical 2D convolutions with square kernels. Our approach allows us to synthesize textures that account for the geometric content of mesh surfaces, eliminating discontinuities and achieving comparable quality to 2D image synthesis algorithms. We compare our approach with state-of-the-art methods [GRGH19a, MPSO18a, HJN22a] where, through qualitative and quantitative evaluations, we demonstrate that our approach is more effective for a variety of meshes and styles, while also producing visually appealing and consistent textures on meshes.

3.1 Introduction

In *computer graphics*, textures refer to two-dimensional images applied to the surfaces of 3D meshes, by mapping the pixels of the former onto the vertices or polygons of the latter. During mapping, we determine which pixel in the texture maps to a mesh surface point in consideration—ultimately simulating a diverse range of surface properties, such as color, reflectivity, transparency, and more. This process is fundamentally oriented towards creating sophisticated and realistic visual effects, leveraging highly intricate texture images while preserving rendering efficiency. Conversely, in *computer vision*, textures refer to visual patterns or structures intrinsic to an image. These patterns or structures are identified through the spatial arrangement of pixel intensities, color distributions, and other visual features. Texture analysis, entailing the extraction and characterization of these patterns, focuses on revealing the underlying structure of an image and understanding its content.

Generating high-quality textures for 3D meshes is a manual and tedious process, due to the inherent disparity between the 2D nature of textures and the 3D shape to which they are eventually projected. Consequently, the automatic generation of textures for 3D meshes, known as *mesh texture synthesis* has emerged as a crucial technique for the rapid and controllable creation of 3D content. Situated at the intersection of computer graphics and computer vision, mesh texture synthesis aims to generate textures for 3D meshes that exhibit visual coherence, meaningfulness, and realism. To accomplish this, mesh texture synthesis takes into account both the underlying geometry and topology of a 3D mesh, as well as the underlying structure or pattern present within the texture image. Thus, mesh texture synthesis may enable the creation of visually compelling and contextually appropriate texture representations for 3D meshes.

Several works have addressed the problem of controllable texture synthesis from a *2D perspective*. These methods operate by taking a 2D image exemplar as input and generating random variations thereof. Traditional methods either define mechanical processes for generating such variations [EL99, EF01, WLKT09] or employ a parametric texture model [Jul62, PS00, SF95]. In recent years, Convolutional Neural Networks (CNNs) have also been trained on natural images to generate variations of a given exemplar [GEB15a, JAFF16]. Although these methods produce visually appealing results, they are not explicitly designed for mesh textures since they are unaware of the eventual geometric context.

Alternative approaches have tackled the same problem from a *3D perspective*. One such approach is solid texture synthesis, which aims to generate a texture in 3D space based on a 2D exemplar. In this context, colors are associated with specific positions within a bounded or unbounded 3D volume [HMR20, GRGH19a, KFCO$^+$07]. However, these methods are designed to generate high-frequency and abstract textures, such as marble, and do not consider the lower-dimensional manifold of 3D meshes during the generation process. Another line of research employs 2D CNNs in conjunction with a differentiable renderer to optimize the resulting mesh texture from a given 2D exemplar across multiple

viewpoints [HJN22a, MPSO18a]. These methods heavily rely on sampling a large number of views, which can be computationally demanding and may lead to occlusion-related inconsistencies.

In this chapter, we propose a novel surface-aware mesh texture synthesis method that mitigates the drawbacks observed in prior research efforts. Our methodology leverages the pre-trained weights of a 2D CNN to another CNN with an identical architecture—but with convolutions operating *directly on the tangent space* of a mesh. This design enables the utilization of pre-existing weights from the 2D network, trained on an extensive corpus of natural images. Moreover, it facilitates the comparison of Gram matrices between the two networks, as we define a loss function between a 2D texture and a mesh texture directly. In a comparative evaluation, we demonstrate that our approach generates visually appealing mesh textures from a large variety of exemplars while respecting the geometric context of the mesh. Our implementation is publicly available in our repository (https://github.com/AronKovacs/mesh-texture-synthesis).

3.2 Our Approach

Our method, inspired by previous works in the field of 2D texture synthesis [GEB15a], introduces a novel spatial invariant parametric texture model. This model is built upon a hierarchical CNN that has been pre-trained on the task of object recognition in natural images. In contrast to previous works, we extend this concept to 3D meshes by generalizing the building blocks of the 2D network architecture to operate on the tangent space of a 3D mesh (Figure 3.1). This generalization involves redefining two of the main building blocks of the network: the *convolution operation* and the *pooling operation*. Figure 3.1 illustrates schematically these operations.

Whilst the convolution operation applies filters on the image to detect patterns (Sec. 3.2.1), the pooling operation reduces the image size to increase the receptive field of the subsequent filters and reduce the computational burden (Sec. 3.2.2). By redefining these operations to operate on the surface of a mesh, we can directly leverage the weights of a pre-trained 2D network, while maintaining the rest of the architecture intact. To this end, our method optimizes 3D mesh convolution and pooling by precomputing geodesic neighborhoods and pooling groups with linear time complexity relative to texel counts and triangle numbers (Sec. 3.2.3). Consequently, we generate textures directly on the surface of a 3D mesh (Sec. 3.2.4). In the following subsections, we describe in detail each of these operations and the optimization process of our proposed algorithm.

3.2.1 Convolution

2D Definition. Given an input feature map $\mathcal{F}_i \in \mathbb{R}^{W \times H}$, the standard 2D discrete convolution computes an output feature map $\mathcal{F}_o \in \mathbb{R}^{W \times H}$. The new feature value for each pixel $p \in \mathbb{N}^2$ is the weighted sum of input features from pixels in its neighborhood

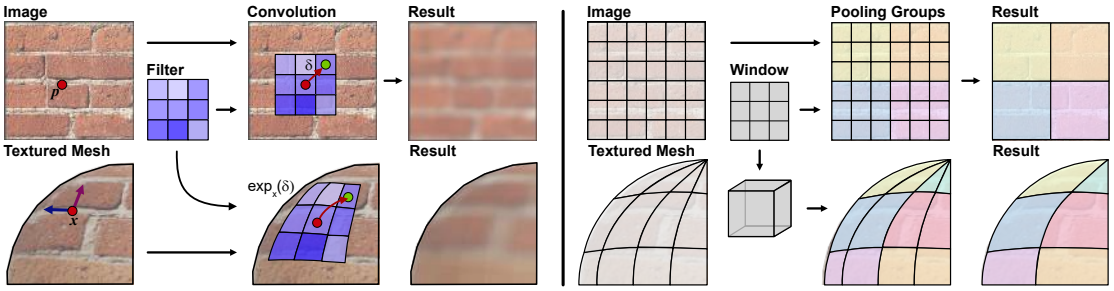


Figure 3.1: The convolution and pooling operation, as redefined within the context of our work. *Left:* The convolution is applied to the input data (image vs. textured mesh) to filter the available information and produce a feature map. However, for the textured mesh, we modify the neighborhood sampling to account for the mesh topology. *Right:* During the pooling, we define a sliding 3D window that selects texels to aggregate based on their geodesic path (indicated with the colors).

N. The weights of this sum are defined by a kernel function $g(\delta)$ that takes the relative position $\delta \in \mathbb{Z}^2$ of the neighboring pixel w.r.t. p:

$$\mathcal{F}_o(p) = \sum_{\delta \in N} \mathcal{F}_i(p + \delta) g(\delta) \tag{3.1}$$

The neighborhood N is defined as a set of vectors covering a square area of $k \times k$, which is usually implemented as a matrix of the same $k \times k$ shape. In this definition, we omit the third dimension of the feature maps describing multiple channels for simplicity.

3D Mesh Definition. To re-use the weights of a 2D convolution in our mesh convolutions, we use the same definition of convolution as in Equation 3.1. However, we modify the neighborhood sampling to account for the mesh topology. We assume that our 3D meshes are 2D Riemannian manifolds M without boundary embedded in 3D space. Therefore, we can make use of exponential maps to describe the sampling in Equation 3.1 for each point x in our manifold:

$$\mathcal{F}_o(x) = \sum_{\delta \in N} \mathcal{F}_i(\exp_x(\delta)) g(\delta) \tag{3.2}$$

where $\delta \in T_x M$, being $T_x M$ the tangent plane at point x. The exponential map $\exp_x(\delta)$ will follow the geodesic path $\gamma_\delta(1)$ and retrieve a neighboring point y in that direction. Therefore, the square area defined by the neighborhood N in Equation 3.1 is now covering a square area in the manifold locally around x. Note that this definition of convolution is directly inspired by Masci et al. [MBBV15], but we use square neighborhoods instead of circular ones.

Feature Maps. While in Equation 3.1 the feature maps $\mathcal{F}$ were images, now in Equation 3.2 $\mathcal{F}$ denotes scalar-valued functions on the manifold $\mathcal{F} : M \to \mathbb{R}$. Different representations can be used for $\mathcal{F}$. In this chapter, we choose to represent $\mathcal{F}$ as mesh textures. Given the set of points x on M and a 2D texture map $\mathcal{F}$, we define a bijective

map $t : M \to [0,1]^2$ that maps points on M to points in $\mathcal{F}$. Our convolution then becomes:

$$\mathcal{F}_o(p) = \sum_{\delta \in N} \mathcal{F}_i(p')g(\delta) \tag{3.3}$$
$$p' = t(\exp_{t^{-1}(p)}(\delta))$$

Since $\mathcal{F}_i$ is now a discretized 2D feature map, we need to adjust the length of the geodesic path $\gamma_\delta(1)$ based on the distance between texels in our feature map. Our vector δ, therefore, becomes $\delta' = \delta s$, where s is the size of a texel in world space.

To sample our texture map $\mathcal{F}$ at continuous positions, we need to define an interpolation function. Nearest neighbor and bilinear interpolations are fast to execute and implemented by hardware. In our experiments, we used bilinear interpolation in the first layer and nearest neighbor interpolation in the remaining layers of the network, having empirically confirmed that it yields better results. We use the *xatlas* [You22] library to generate the UV mappings, though this choice was primarily motivated by the ease of use and any method that tries to preserve areas should work comparably. Note that relying on a general-purpose UV mapping algorithm may not be optimal, and specialized algorithms could yield better results if they also minimize distortions or the number of UV islands.

Tangent Frame. Our new definition of convolutions requires a local reference frame defined at each point x. For simplicity, given the normal N at point x, we compute the tangent vectors T and B with the Gram-Schmidt process using N and a random vector w. However, this process is not defined when N and w are parallel. In such cases, we use a different vector w', which is perpendicular to w. Note that w and w' are the same for all reference frames. Our algorithm is not bound to this method for computing the tangent frames and other methods that generate smoother tangent frames could be used, e.g., the approach by Fisher et al. [FSDH07].

3.2.2 Pooling

2D Definition. The pooling operations used for images usually define a window of size $k \times k$ that is overlaid over the image and a pooling operation that aggregates the values inside the window. For simplicity, in this chapter, we will consider non-overlapping windows, i.e., each pixel is not used by more than one $k \times k$ window.

3D Mesh Definition. To define a similar pooling operation for meshes, we need to define a sliding window that selects texels to aggregate. We use a simple—yet effective—algorithm by defining a 3D voxelization of the space and aggregating texels within a voxel that are connected through a geodesic path within this voxel. The voxel grid of each pooling layer is defined as $k^n \cdot s$, where n is the index of the pooling layer within the network starting from 1, s is the size of the texel in world space, and k the size of the window. Note that this algorithm could generate several pooling groups inside the same voxel. Also note that this algorithm will generate pooling groups of different sizes.

Texel Graph. To accelerate the pooling operation, we build an undirected graph E of the texels of our texture at each level in the network. An edge e_{ij} on this graph connects texels i and j if they are adjacent, i.e., if there is a boundary connecting the two texels. This allows for fast queries of connected components during the determination of pooling groups. Our pooling operation generates pooling groups of varying sizes and, thus, texels of varying sizes in deeper layers of the network. This might cause our exponential maps to overshoot and select a distant texel that is not connected to the central texel at p, for which the convolution is being computed. We depict this in Figure 3.2. To avoid such cases, we use our graph E to determine if the neighboring point is adjacent to our central texel. If not, we follow $\gamma_{\delta'}(1)$ backward, until we find a point along $\gamma_{\delta'}(1)$ belonging to a texel adjacent to our central texel at p.

3.2.3 Precomputation

Unlike regular 2D convolution and pooling, where the position of each neighbor is trivial to compute, in 3D mesh convolution and pooling it is not sufficient to sample surrounding points in UV space. However, the local geodesic neighborhood of each texel can be computed in advance, assuming that a given mesh is not being deformed during the training process. For the convolution, our implementation precomputes the local neighborhood of each texel, storing the position and bilinear factor for each sample in GPU buffers. The time complexity of this process is at worst linear w.r.t. the number of triangles and the number of texels, as the computation of the geodesic paths consists primarily of crossing from one face to another. During training, our method only depends on the number of texels in each layer and should scale linearly.

Similarly, we precompute the grouping into pooling groups which we then again store on the GPU. This has a linear time complexity w.r.t. the number of texels as this mostly involves grouping them into voxels and then computing the connected components using the aforementioned texel graph. As a UV unwrapping may not fully utilize the whole UV space, we work only with the texels that are used and store only their features. This approach enables us to ignore unused texture parts—thus, reducing the needed memory.

3.2.4 Texture Synthesis

Our algorithm makes use of two CNNs with the same architecture. We used the VGG-19 architecture [SZ14], which is one of the most commonly used architectures for image synthesis and style transfer for 2D images [GEB16, GEB15a, LW16]. The choice between the architecture variants of VGG was based on the documented advantages of VGG-19 over VGG-16 with regard to its increased discriminative power, improved representational capacity of complex images, and better feature extraction due to the deeper architecture. Whilst one of our networks is designed to work on 2D images, the other one uses convolution and pooling operations for 3D meshes.

We start with a VGG-19 network, pre-trained for the task of image classification on ImageNet1000 [RDS+14]. Subsequently, we use the pre-trained weights to initialize the

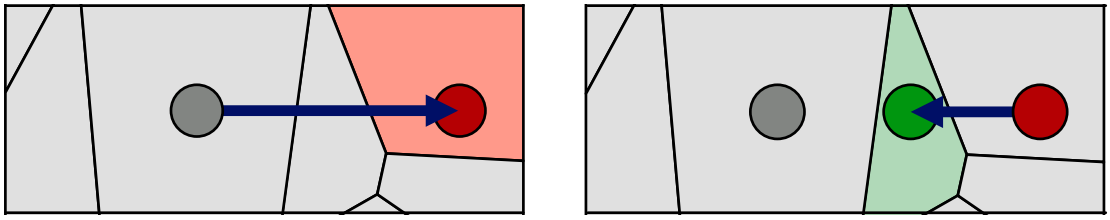


Figure 3.2: We employ a mechanism to avoid overshooting from a texel at position p (in grey) to a distant disconnected texel (in red). Therefore, we correctly determine the adjacent texel (in green).

second network that operates on 3D meshes. Once the weights of both networks are defined, they are frozen before the texture generation. We initialize the mesh texture with random noise values drawn from a uniform distribution on the range $[0, 0.2]$. The choice has been motivated by the findings of previous work [GEB15a]. Then, we execute the 2D network with the 2D image example as input, and the mesh network with the random texture as input. For each layer of the network, the Gram matrices are computed [GEB15a]. The final loss is defined as the mean square error between the Gram matrices of all layers in the network, where all of them are assigned the same weight. In this way, we measure the discrepancy between the Gram matrices, reflecting the difference in style representation between the generated outcome and the target. The gradients of this loss are back-propagated through the mesh network until the input mesh texture and its values are optimized. After several optimization steps, the final mesh texture is generated.

3.3 Results and Evaluation

In this section, we present an analysis of the results generated with our method (Sec. 3.3.1). We also provide a comparison to other indicative state-of-the-art approaches (Sec. 3.3.2). All synthesis results are generated using our tool written in a combination of Python, Rust, TensorFlow, and CUDA, and running on a desktop machine equipped with an AMD Ryzen 9 3900X with 128 GB RAM and an NVIDIA GeForce RTX 3080 Ti. We use the Adam optimizer with a rather large learning rate of 0.1, which is halved after 200 iterations and again after 400 iterations. The optimization process stops after 500 iterations. Depending on the complexity of the mesh and the fraction of the utilized UV space, the precomputation part takes 2–5 minutes and the optimization process takes approximately 50 minutes for textures of size 1024×1024. More detailed measurements are included in Tables 3.1 and 3.2.

3.3.1 Visual Quality of our Approach

To demonstrate the versatility of our approach, we prepared a selection of testing sequences with a variety of meshes and textures as input data. These include meshes

with different topologies and at different levels of detail (i.e., coarseness), and texture exemplars with different stimuli and resolutions.

Results with Different Meshes and Textures. Figure 3.3 depicts the outcomes of our approach applied to the bunny and the dragon from the Stanford 3D scanning repository, as well as the Mother and Child mesh, for five textures with a diverse set of stimuli—ranging from high-frequency textures [PS00] to isotropic materials [GRGH19a], and also artistic styles [GEB15a]. More results with additional meshes and textures are included in Figure 3.9. The results indicate that our approach can infer a reasonable mesh texture from the 2D exemplar while preserving the features learned along the mesh, for a large variety of textures. Our method successfully reproduces the underlying structure of the different exemplars, while capturing their colors and variations, and also following the mesh geometry. For instance, when employing textures with *Kandinsky's on White II* or *Hokusai's The Great Wave off Kanagawa*, we observe that colors and patterns (fine-grained and coarse) are preserved, while the geometrical context of the surface is also respected—even for meshes of different topologies or higher genus, such as the dragon or the Mother and Child mesh in the last two rows of Figure 3.3. For the plant texture, we notice a few bright spots in all meshes (e.g., at the back foot of the bunny in the fourth column of Figure 3.3). This might be due to rapid changes in the tangent field in the surrounding area. This may be causing the mesh network with VGG-19 weights to

Table 3.1: Training times (**t**) and peak VRAM usage for the 3D texture generation. For the render-based approaches of Mordvintsev et al. and Höllein et al., we used the bunny from the Stanford 3D scanning repository for the training. For the volume-based approach of Gutierrez et al., the measurements below refer to training with the entire volume. For our approach, we show results for different meshes. All measurements were done for a 1024×1024 marble texture (unless specified otherwise), and the outcomes are depicted in Figure 3.5. Note that the used UV space is not relevant for the measurements and is, thus, not indicated in the table.

	***t* (m:s)**	**VRAM (MB)**	**Used UV**
Gutierrez et al. [GRGH19a]	59:43	564	–
Mordvintsev et al. [MPSO18a]	43:32	6122	–
Höllein et al. [HJN22a]	40:48	711	–
Ours – Bunny	47:17	2422	63.33%
Ours – Bunny (512×512)	12:39	737	64.53%
Ours – Bunny (256×256)	3:51	291	66.84%
Ours – Armadillo	43:53	2314	64.21%
Ours – Buddha	49:46	2309	62.49%
Ours – Dragon	38:49	2397	68.27%
Ours – Mother	49:58	1933	52.61%
Ours – Snail	36:07	1848	48.23%
Ours – Teapot	53:24	2687	69.14%

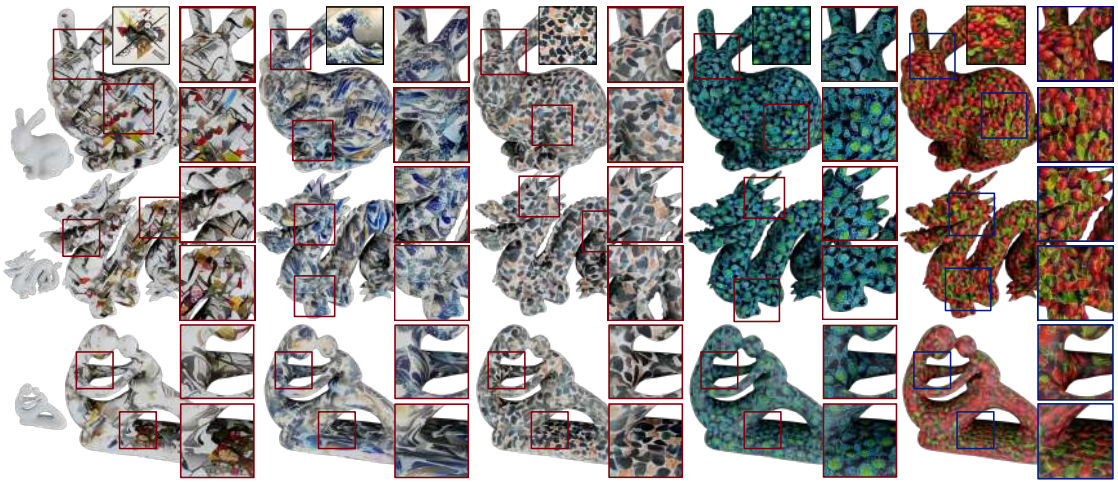


Figure 3.3: Our method applied to three meshes (the bunny and the dragon from the Stanford 3D scanning repository, and the Mother and Child by Brian Weston (CC BY-SA)) and five textures with a diverse set of stimuli.

lose local context to such an extent that the extracted local features cannot be properly matched with the provided style. For the plant and the radishes textures (fourth and fifth column of Figure 3.3), we also notice that the shapes are slightly more elongated as opposed to the rounder structures in the exemplar. This might be due to the different sizes and shapes of the pooling groups.

Results at Different Levels of Detail for the Meshes and Different Texture Resolutions. We also experimented with different resolutions for the textures and different levels of details, i.e., coarseness, for the meshes. In Figure 3.4 (a), we depict one example with the armadillo from the Stanford 3D scanning repository at two different levels of detail (2,124 and 212,574 polys) and the *Kandinsky* texture at two different resolutions (128×128 and 256×256). As anticipated, a higher texture resolution

Table 3.2: Precomputation times (**t**) of our approach using various meshes. A 1024×1024 texture (unless specified otherwise) is used for the 6 texel layers needed for VGG-19 with 5 pooling layers.

	***t* (m:s)**	**Used UV**	**Triangles**
Bunny	1:44	63.33%	5002
Bunny (512×512)	0:42	64.53%	5002
Bunny (256×256)	0:21	66.84%	5002
Armadillo	2:30	64.21%	212574
Buddha	3:23	62.49%	108770
Dragon	2:45	68.27%	217853
Mother	1:33	52.61%	56512
Snail	2:13	48.23%	574
Teapot	3:32	69.14%	6320

and a higher number of polygons both influence the outcome. Although the texture resolution qualitatively seems to be the most influential factor, it is noteworthy that using a finer mesh also contributes to fewer artifacts, e.g., on the chest of the armadillo in Figure 3.4 (a).

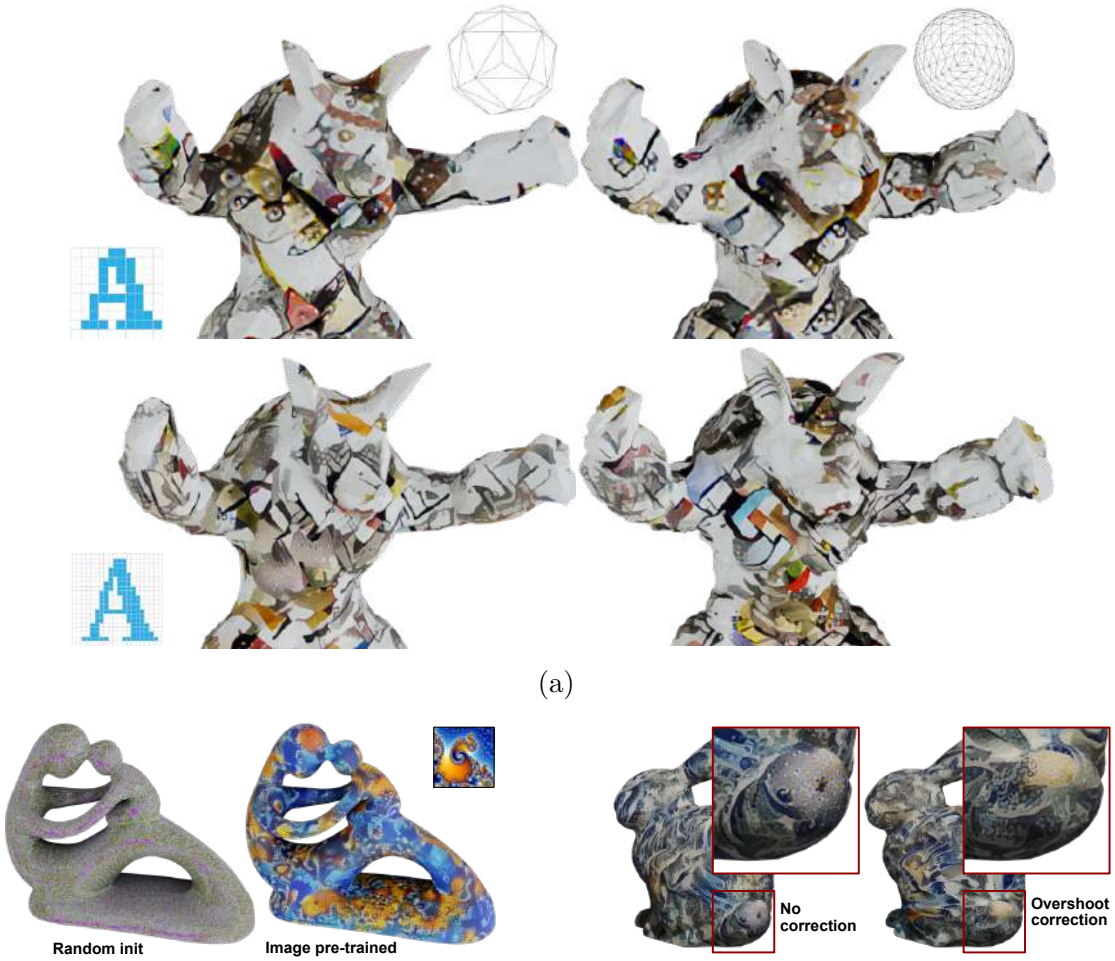
Impact of Weight Initialization. Moreover, we evaluate the effect of the pre-training process in 2D. Figure 3.4 (b) offers a comparison of the results of our optimization process when using the weights of the 2D pre-trained network against those of a neural network with randomly initialized weights. We see that, without the prior knowledge acquired during pre-training, the algorithm is not able to generate a plausible mesh texture (left). On the other hand, when the pre-trained weights are used, the generated mesh texture matches the patterns of the original image (right).

Impact of the Overshooting Correction. We finally evaluate the effect of our correction that shortens geodesic paths to avoid overshooting as shown in Figure 3.2. Figure 3.4 (c) shows that without this correction, artifacts can be seen in certain parts of the texture such as the bunny tail. The pre-trained network does not expect the overshooting to occur and, therefore, skipping over to more distant unconnected parts of the texture may cause it to extract incorrect features.

3.3.2 Comparison to the State of the Art

We compare our approach to selected approaches from the state of the art—namely, the approaches of Gatys et al. [GEB15a], Gutierrez et al. [GRGH19a], Mordvintsev et al. [MPSO18a], and Höllein et al. [HJN22a] (Figure 3.5). The work of Gatys et al. is a traditional 2D image texture synthesis neural approach. Gutierrez et al. propose a solid texture approach, where a volume is defined and sliced so that the style of the slices matches a given style image. Afterward, the volume is sampled at the object-space texel positions. Conversely, the approaches of Mordvintsev et al. and Höllein et al. are both render-based. The former renders a textured mesh and then matches the Gram matrices with a style image, while the latter introduces additional viewing angle and depth corrections. These works showcase different perspectives and applications, ranging from traditional 2D image style transfer to volumetric texture synthesis and style transfer in rendering 3D scenes. However, they all focus on generating or transforming images in a way that captures and transfers specific visual styles or textures. The approach of Gatys et al. is only used as a baseline for the 2D case, while the other three are used for the 3D scenario comparison.

Our selection has also been motivated by the availability of open-source implementations [GEB15b, MPSO18b, GRGH19b, HJN22b]. To maintain uniformity across methodologies, we deliberately chose to employ a VGG-19 network consistently in our study. As Gutierrez et al. and Höllein et al. already utilize VGG-19 in their approaches, we adapted our methodology to incorporate this network. It is worth noting that we had to transform the original implementation of Mordvintsev et al., which initially employed an Inception v1 network [SVI+16]. The quality of results is contingent on the chosen network, and



(a) (b) (c)

Figure 3.4: Ablation study: (a) Our method applied to the armadillo from the Stanford 3D scanning repository (at two different levels of detail: 2,124 and 212,574 polys) and the *Kandinsky* texture (at two different resolutions: 128×128 and 256×256). (b) Our method applied to the Mother and Child by Brian Weston (CC BY-SA) once with randomly initialized weights (left) and once with pre-trained VGG-19 weights. (c) Our method applied to the bunny from the Stanford 3D scanning repository without (left) and with (right) the overshooting correction.

we aimed to employ a consistent network across all approaches. Therefore, we had to integrate a VGG-19 network instead, ensuring a cohesive and comparable evaluation framework across all approaches. Still, the usage of VGG-19 is not fundamental for our approach and that is also the case for the neural networks chosen by the approaches we compare against. All approaches use these 2D networks just as feature extractors and the contribution of all these works is the method to bridge the gap between 3D and 2D (rendering or slicing) and/or corrections to account for this projection. We additionally note that the approach of Höllein et al. is primarily meant to stylize scenes. To adapt it

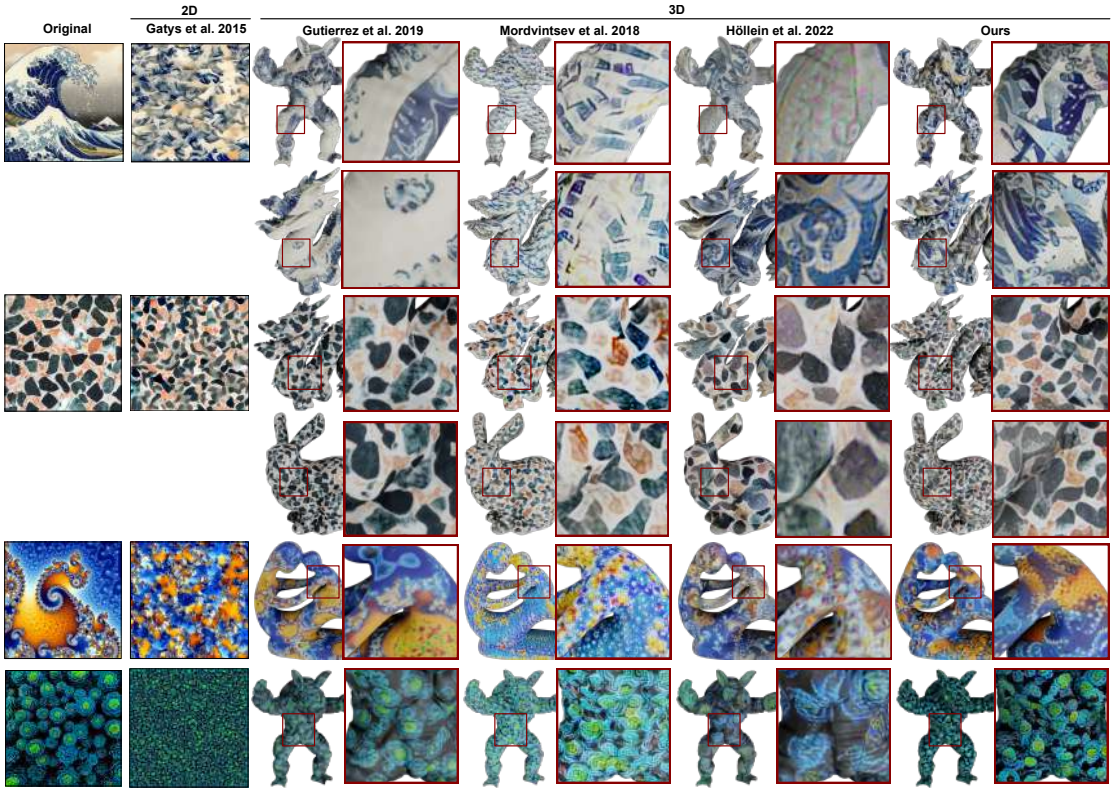


Figure 3.5: Results of our approach compared to those by Gatys et al. [GEB15a], Gutierrez et al. [GRGH19a], Mordvintsev et al. [MPSO18a], and Höllein et al. [HJN22a] for four meshes (armadillo, dragon, and bunny from the the Stanford 3D scanning repository, and the Mother and Child by Brian Weston (CC BY-SA)) and four textures.

to synthesize textures for individual meshes, we had to place cameras uniformly around each mesh and used the output of each hidden layer to compute the Gram matrices. Finally, we used the same UV unwrapping for all approaches.

Visual Comparison. For the comparison to the state of the art, we generated comparable cases for all approaches. The comparison outcomes are shown in Figure 3.5 for *four meshes* (the armadillo, the dragon, and the bunny from the the Stanford 3D scanning repository, as well as the Mother and Child mesh) and *four textures* (*The Great Wave*, a marble texture, a texture depicting the Mandelbrot set, and a texture with succulent plants from GitHub). As anticipated, the approach of Gatys et al. is capable of producing natural 2D textures both for *The Great Wave* and the isotropic marble material texture. We observe, however, a few artifacts—for instance, at the bottom right corner of *The Great Wave* texture and also at the left edge of the marble texture. These might be due to the proximity to the edges of the generated image, where zero-padding does not match the distribution of values in the rest of the image, and the network extracts unsuitable features. Finally, for the Mandelbrot texture and the plants texture, the method is also

able to capture the structure and patterns of the original texture, but at a different scale due to the different texture resolutions used in the original texture and the generated one.

Subsequently, we move on to the comparison with the approaches of Gutierrez et al. [GRGH19a], Mordvintsev et al. [MPSO18a], and Höllein et al. [HJN22a]. The approach of Gutierrez et al. works particularly well for isotropic materials, such as the marble texture. This is anticipated because isotropic materials exhibit a high degree of symmetry, and as such any two slices of the stylized volume should resemble each other. With regard to *The Great Wave*, we see that the surface of the mesh is not well-respected (e.g., at the thigh of the armadillo in Figure 3.5). Here, we have harsh artifacts, which can be observed as darker stripes on the white part of the texture. The same kind of artifacts can also be seen when employing the plants texture (e.g., at the belly of the armadillo in Figure 3.5). Moreover, the colors and the sharpness are compromised to a degree, while for the Mandelbrot texture bright artifacts appear (e.g., at the child's leg in Figure 3.5).

Conversely, the approach of Mordvintsev et al. does not work as expected. A potential reason for that is the use of VGG-19, as opposed to the original Inception v1 network. Although Inception networks are anticipated to yield superior results, comparing them to our employed VGG-19 would be unfair. Therefore, we leave this investigation as a point for future work—potentially, within a wider ablation study. The presence of blurry artifacts in render-based baselines could be attributed to the selection of camera viewpoints, as uniform placement might result in certain patches being inadequately viewed from optimal, i.e., orthogonal, perspectives. For *The Great Wave*, the outcomes of the approach of Mordvintsev et al. are vaguely resembling the exemplar, where only the colors are preserved. Additionally, some artifacts resembling speckle noise are visible, which are also evident in the images generated with the approach of Gatys et al. [GEB15a]. This can be mitigated by blurring or by including a term in the loss function that penalizes noise. The same artifacts are also present in the marble case, although the overall visual quality of the result is better with this texture. For the Mandelbrot and the plants texture, we notice severe artifacts that compromise both the represented structures and the saturation of the texture.

Finally, the performance of the approach of Höllein et al. seems to depend on the employed mesh. Notably, for *The Great Wave*, the performance with the armadillo mesh is suboptimal, while the dragon has only a few speckle-like artifacts. The marble texture works well for both employed mesh specimens, while the performance with the Mandelbrot and plants textures is insufficient. For the Mandelbrot case, the colors are desaturated and large artifacts are evident, while large streak artifacts appear in the plants texture also. An additional reason for the artifacts in the results obtained with Höllein et al.'s implementation could be due to *mipmapping*. This is a technique where a texture is progressively downsampled to increase rendering speed and reduce aliasing artifacts. In neural approaches, mipmapping can be used as a pooling operation [LLZ+19] or as a method for working with coarser features [HJN22a]. However, this approach may introduce errors. In some cases, the UV mapping may generate islands adjacent

to each other in the UV space—yet, far apart in a geodesic sense. These islands merge during mipmapping, causing the gradient to flow through incorrectly merged pixels, and potentially resulting in erroneous correlations between regions or introducing undesirable noise. This drawback is counteracted in our implementation by pooling based on the geodesic space (instead of the UV space). We, therefore, suggest that techniques utilizing mipmapping should also use adequate island margins during training, or should consider alternative methods for pooling or representing coarse features.

User Study. To evaluate our method, we conducted an informal, online user study with 30 participants, where we used several of the cases shown in Figure 3.3 and 3.9. We presented each participant with the produced outcomes of our approach and the approaches of Gutierrez et al., Mordvintsev et al., and Höllein et al. together with the respective texture exemplars. Without disclosing any information about any of the approaches, we interviewed the participants to gain some qualitative feedback about the outputs. Namely, we asked them to rank the four approaches concerning their similarity to the provided texture exemplar. For each of the generated results, we also asked the study participants to rate on a 1–5 Likert scale their visual appeal and coherence.

The analyzed outcomes of the user study are shown in Figure 3.6. The study participants ranked our approach as the closest to the texture exemplar ($\mu \pm \sigma = 64.3 \pm 26.9\%$ of the participants), followed by the approach of Gutierrez et al. ($\mu \pm \sigma = 17.6 \pm 16.5\%$) and Höllein et al. ($\mu \pm \sigma = 17.6 \pm 29.5\%$), and last by the approach of Mordvintsev et al. ($\mu \pm \sigma = 0.5 \pm 1.3\%$). The results are statistically significant ($F = 8.44196$; $p = .001028$), as shown with an ANOVA test followed by pairwise t-tests. The approach of Mordvintsev et al. was judged as the least similar to the texture exemplar ($\mu \pm \sigma = 57.6 \pm 33.3\%$ of the participants), followed by the approach of Höllein et al. ($\mu \pm \sigma = 35.7 \pm 35.1\%$), and Gutierrez et al. ($\mu \pm \sigma = 4.8 \pm 6.6\%$), and last by ours ($\mu \pm \sigma = 1.9 \pm 2.6\%$). The results are statistically significant ($F = 6.2159$; $p = .004376$), as shown with an ANOVA test followed by pairwise t-tests. In terms of visual appeal and coherence, the overall preferred approach is ours as indicated also visually in the plots of Figure 3.6. The distribution of the ratings of our approach differs statistically significantly from the other three approaches in coherence ($F = 15.08026$; $p = .000037$) and visual appeal ($F = 13.77907$, $p = .000065$), as shown with ANOVA tests followed by pairwise t-tests. To sum up, according to our study participants, our approach outperforms qualitatively the other three methods in all investigated aspects.

Speed and Memory Comparison. We measured the training times and peak VRAM usage during the training phase of all approaches. Recall that our approach only considers the texels that are being used by a given UV unwrapping, hence for our approach we give the measurements for each mesh together with the percentage of used texels. The measurements in Table 3.1 indicate that our training times are comparable to the other approaches, sometimes even surpassing them depending on the fraction of used UV space. Even though we currently require more memory than Gutierrez et al. and Höllein et al., this can be alleviated with a better implementation, however, we would still need to store the neighborhoods for convolution and pooling. On the positive side, though,

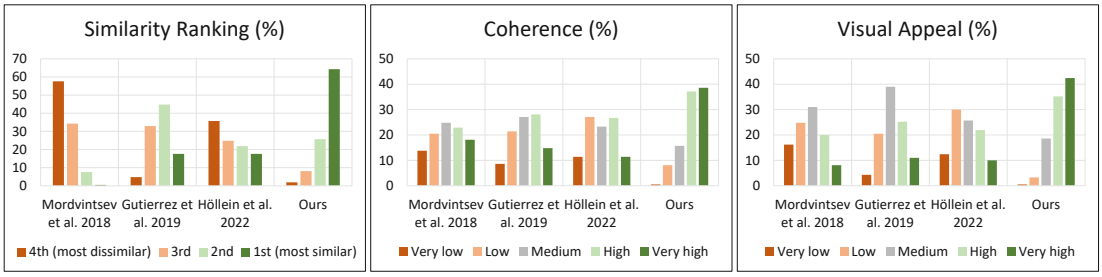


Figure 3.6: Perceived similarity to the 2D exemplar, coherence, and visual appeal of our approach vs. Gutierrez et al. [GRGH19a], Mordvintsev et al. [MPSO18a], and Höllein et al. [HJN22a] in a user study with 30 participants.

even low-end GPUs have enough memory to be able to utilize our method. Both training times and memory scale with the used UV space, but there are small deviations from a linear scaling, which could be caused by different sizes of pooling groups. Finally, in Table 3.2 we show the precomputation times for the neighborhoods during convolution and pooling, which are an order of magnitude smaller than the training times.

Extension to Other Tasks. Although we have heavily showcased our approach within the context of texture synthesis for single objects, there is a possibility of extension to a broader variety of tasks. This includes, for instance, style transfer and texture synthesis for whole scenes, but also segmentation or classification—similar to the previous work of Li et al. [LLZ+19]. We have not investigated the latter, but we showcase a few initial results of style transferring (Figure 3.7) and stylization of whole scenes (Figure 3.8). For the style transfer task, we have used an additional content texture that represents ambient occlusion and also an RGB texture [TL22]. For the former, we stylize with *The Great Wave* texture and the newspaper (used also by Mordvintsev et al. [MPSO18a]). We use the same approach as Gatys et al. [GEB16], i.e., we match the Gram matrices of the generated texture with those of the style image and match the values of feature layers of both the generated texture and the original content texture. We use the output of the following layers to compute the Gram matrices: $block1\_conv1$, $block2\_conv1$, $block3\_conv1$, $block4\_conv1$, $block5\_conv1$ which are weighted the same. We use the output of $block4\_conv2$ to compute the content difference, which we multiply by 1000.

In these preliminary examples, we observe that our approach also performs reasonably well for style transfer, despite not being explicitly designed for it. The style textures are applied in a manner that respects the geometry of the underlying mesh and the input texture. Observe the inner cavity of the bunny ear, and the creases under its neck or between its legs in Figure 3.7 (a: top row); and compare them with the respective renderings provided in Figure 3.9. The same effect can be noticed on the chest and the clothing of the Happy Buddha (Figure 3.7, a: bottom row vs. Figure 3.9). For the RGB texture, notice the contours around the eyes and nose of the bunny (Figure 3.7 (b)). The same meshes and textures have been employed, with the sole addition of the content texture for the style transfer task.

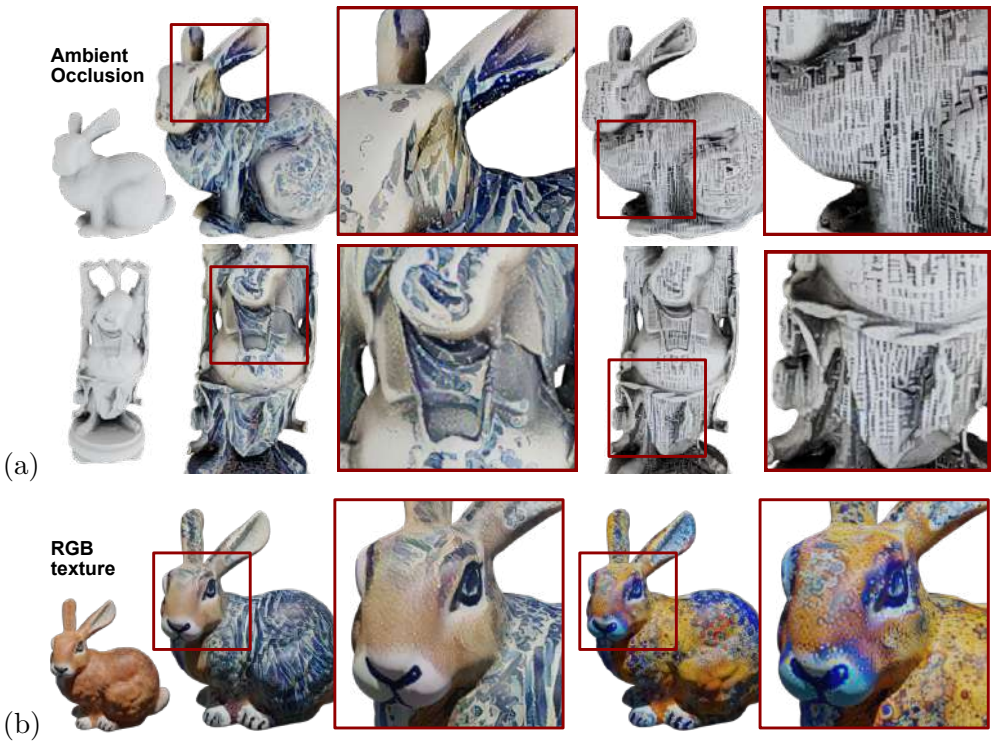


Figure 3.7: (a) Two examples of style transfer with our approach with a texture that represents ambient occlusion. Top row: The input is the Stanford bunny. It is stylized with *The Great Wave off Kanagawa* (left) and a newspaper texture (right). Bottom row: The input is the Happy Buddha, stylized with the same two textures as the previous case. (b) Style transfer with two different styles (left: *The Great Wave off Kanagawa*, right: *Mandelbrot*) on the Stanford bunny with an RGB content texture [TL22].

For the whole scene texture synthesis task, we provide a comparison of our results with the approach of Höllein et al. in Figure 3.8. The mesh was reconstructed from real-world data, therefore, it contains several holes and is noisy. This poses a challenge for our approach, as it favors surfaces that do not have a boundary. Furthermore, our method creates a rapidly changing tangent field in the noisy areas, which causes a loss of local context. Hence, the synthesis process is not able to create texture patches resembling the style exemplar. In contrast, the approach of Höllein et al. does not have this problem. Yet, it struggles in a few areas close to the windows, which are at the bottom left part of the scene in Figure 3.8. Those parts have not been sufficiently captured by the camera and, after rendering, the thin geometry is surrounded by a black background.

3.4 Limitations

Being inspired by the approach of Gatys et al., unavoidably our approach faces similar limitations. As discussed also in Sec. 3.3, we have high computational costs, especially

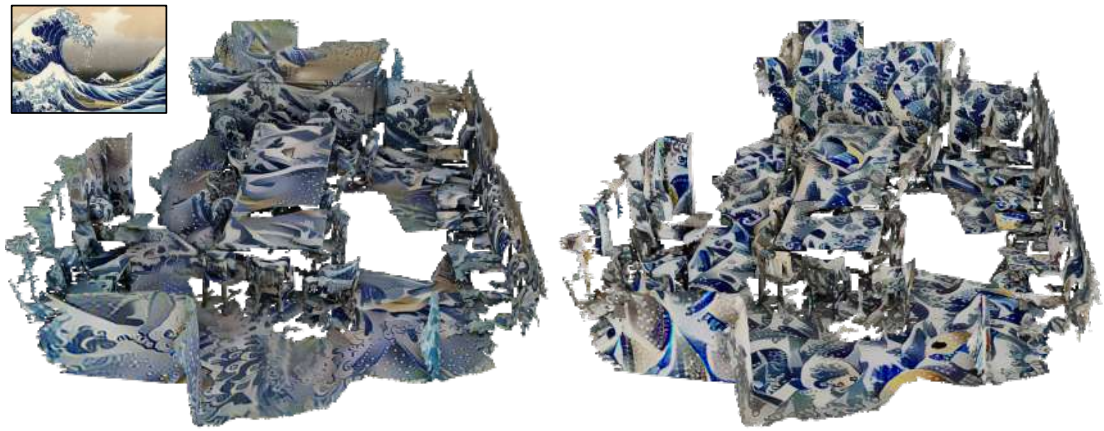


Figure 3.8: Stylization for scene 0291_00 from the ScanNet dataset [DCS+17] achieved with the approach of Höllein et al. [HJN22a] (left) and ours (right) using *The Great Wave off Kanagawa* as the style.

in the optimization step. Furthermore, although our approach has reasonable control over the geometry and topology of the underlying mesh and produces visually appealing results as demonstrated in Sec. 3.3.1, we do not always have fine-grained control over the specific features or elements we want to transfer or retain in the synthesized texture. High-frequency changes in the tangent field may cause certain artifacts, e.g., colored spots that do not match the style texture. This is, for instance, visible in the examples with the plants texture (Figure 3.3).

Our examples showcased that we are overall effective when applying a diverse set of textures or patterns—not only artistic styles but also natural textures and complex geometric patterns (Figure 3.9). Yet, with a more robust architecture, we might be able to preserve better the texture structures or patterns. As expected, the algorithm's performance and the quality of the synthesized texture can be sensitive to several hyperparameters, requiring manual tuning and experimentation to achieve satisfactory results (Figure 3.3 and 3.4). Moreover, uneven sampling and pooling may introduce information loss due to bias towards dominant features or regions in the data, as well as distortions or misalignments in the spatial relationships between features, impacting subsequent tasks that rely on accurate spatial information.

Lastly, our convolution definition in Equation 3.2 assumes that the displacements along the geodesic paths are localized to a small neighborhood around the points. However, for network architectures with a large number of layers and, more importantly, several pooling operations, the covered patch might span large portions of the manifold breaking the assumption of locality.

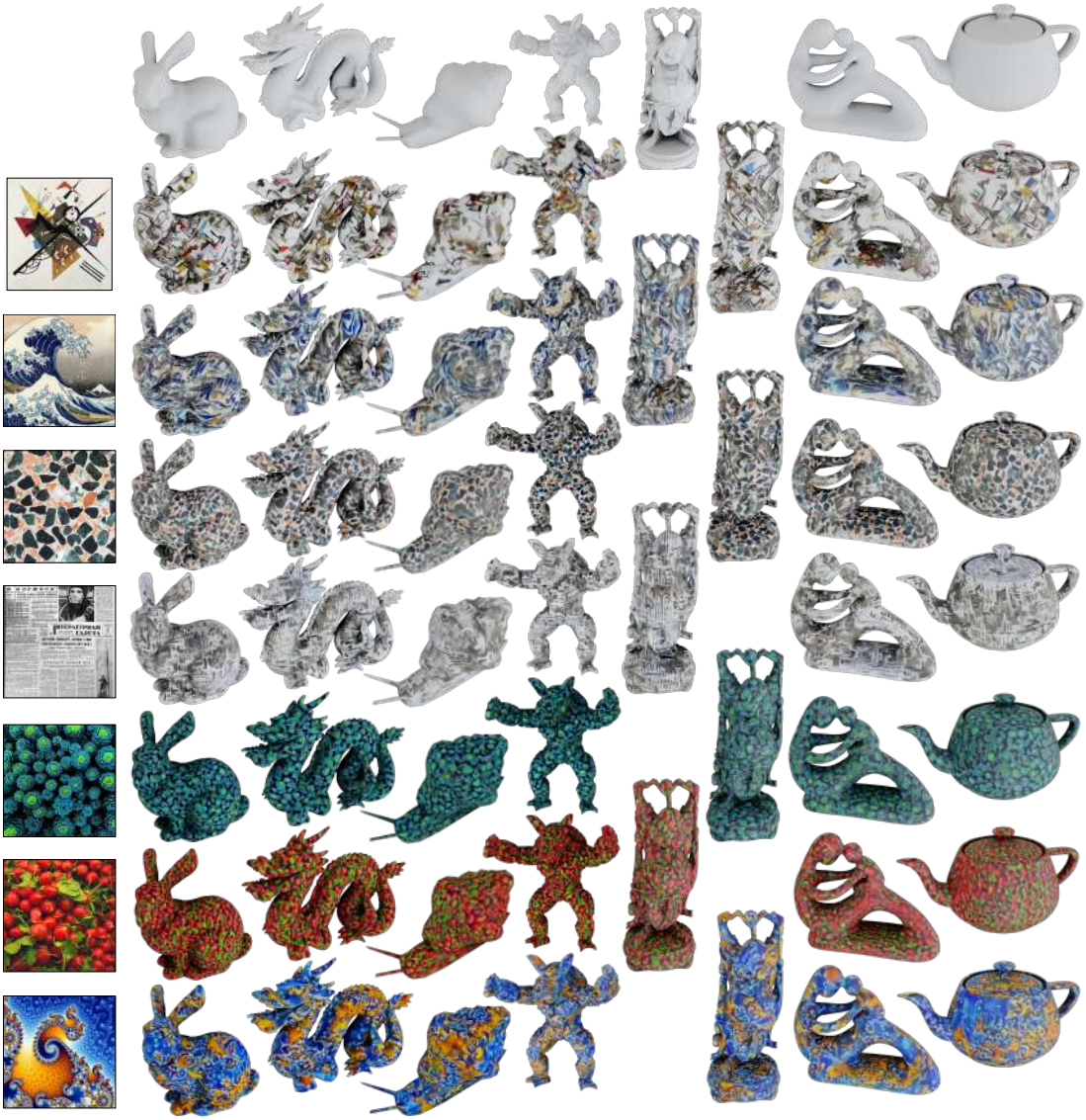


Figure 3.9: Our method applied to seven meshes and seven textures with a diverse set of stimuli. The meshes include the bunny, the dragon, the armadillo, and the Happy Buddha from the Stanford 3D scanning repository, a snail mesh created in Blender, the Mother and Child by Brian Weston (CC BY-SA), and the Utah teapot. The textures include (*from top to bottom*) two artistic styles—namely, *Kandinsky's on White II* and *The Great Wave off Kanagawa*, an isotropic marble texture similar to those used by Gutierrez et al. [GRGH19a], the newspaper texture used by Mordvintsev et al. [MPSO18a], a high-frequency abstract texture containing succulent plants obtained from GitHub, the radishes texture also used by Gatys et al. [GEB15a] and previously by Portilla and Simoncelli [PS00], and the cropped Mandelbrot texture created by Wolfgang Beyer (CC BY-SA).

3.5 Conclusion and Future Work

We have presented an example-based approach for texture synthesis for textured mesh objects. Our method uses a modification of the well-tested approach for style transfer of Gatys et al. [GEB16], where the underlying data representation—instead of being a flat 2D plane—is the curved 3D surface of a given mesh. In this way, our approach takes into consideration the topology and geometry of the mesh in a manner superior to the previously proposed approaches. We showed that our approach works well for a variety of meshes with different styles. Our approach minimizes artifacts of existing learning-based methods, by being seamless and taking into account the local topology. Our method also minimizes feature bleeding across the Euclidean space.

In future work, we propose to investigate 2D convolutional networks resistant to the domain change from flat images to curved surfaces. It would also be interesting to modify the underlying mesh geometry to capture both visual style and 3D shape, similar to Hertz et al. [HHGCO20]. Another future direction could investigate other architectures, beyond VGG, and with other loss functions. Additionally, our work could provide further insights into repurposing image-trained neural networks for general tasks with different local structures. Finally, in a future evaluation, it would be interesting to extend our approach to other tasks, e.g., segmentation or classification, and to compare our method to more recent NeRF-based stylization approaches [CYL+25, HHY+22, ZKB+22a, LZC+23a] and text-based stylization approaches [MBOL+22, KXBP22, RMA+23, MWZ+23].

CHAPTER 4

$\mathscr{G}$-Style: Stylized Gaussian Splatting

This chapter is based on the following publication:

> Áron Samuel Kovács, Pedro Hermosilla, and Renata G. Raidou. "G-Style: Stylized Gaussian Splatting." Computer Graphics Forum. Vol. 43, No. 7, p. e15259, 2024. DOI: https://doi.org/10.1111/cgf.15259

We introduce $\mathscr{G}$-Style, a novel algorithm designed to transfer the style of an image onto a 3D scene represented using Gaussian Splatting [KKLD23]. Gaussian Splatting is a powerful 3D representation for novel view synthesis, as—compared to other approaches based on Neural Radiance Fields—it provides fast scene renderings and user control over the scene. Recent preprints have demonstrated that the style of Gaussian Splatting scenes can be modified using an image exemplar. However, since the scene geometry remains fixed during the stylization process, current solutions fall short of producing satisfactory results. Our algorithm aims to address these limitations by following a three-step process: In a pre-processing step, we remove undesirable Gaussians with large projection areas or highly elongated shapes. Subsequently, we combine several losses carefully designed to preserve different scales of the style in the image, while maintaining as much as possible the integrity of the original scene content. During the stylization process and following the original design of Gaussian Splatting, we split Gaussians where additional detail is necessary within our scene by tracking the gradient of the stylized color. Our experiments demonstrate that $\mathscr{G}$-Style generates high-quality stylizations within just a few minutes, outperforming existing methods both qualitatively and quantitatively.

4.1 Introduction

While humans excel at creating paintings with specific contents and styles, this task has proven challenging for computers to replicate. With the advent of neural networks—and in particular, Convolutional Neural Networks—algorithms have been developed to transfer the style of one image onto another [GEB16]. In this process, a *content image* refers to the original image whose subject matter we aim to retain, while a style image is the one whose artistic style we want to apply to the content image. These algorithms enabled the modification of the style of an image while preserving its content by matching the statistical properties of the embeddings of both content and style images, as obtained from a deep neural network.

With the appearance of novel view synthesis methods based on neural networks (NeRFs) [MST+20], researchers turned their attention to applying style transfer techniques to entire 3D scenes. Style transfer for 3D scenes aims to generate novel views of a scene from a finite number of images of the same scene with a particular style specified by a style image examplar. To succeed in this task, the style transfer method should ensure multi-view consistency between views to provide a smooth navigation experience. Despite the high-quality results provided by 2D style transfer methods, the same algorithms were not able to provide *consistent styles* across 3D scene views [HTS+21, HHY+22, MWWL22, NPLX22]. Therefore, they have been deemed unfit for 3D style transfer. The limitations of these works have been addressed by several methods [NPLX22, ZKB+22a]—either by modifying the colors of the pre-trained NeRF representations or by techniques such as color transfer [ZKB+22a], provided that the original methods were able to support view-dependent effects. Still, NeRF-based approaches require *large training* and *rendering times*.

Recently, Gaussian Splatting has established itself as an active area of research [CW24, FXZ+24, WYZ+24] by demonstrating that it can excel in view synthesis quality while requiring significantly less time for optimization and rendering. Liu et al. [LZX+24] and Saroha et al. [SGC+24] suggested using Gaussian Splatting as the main scene representation and modifying the color of these to obtain stylized renderings of the scene. These methods do not alter the position and shape of the Gaussian representing the scene, which results in a *low resolution* in some areas of the scene and, hence, *low-resolution stylized colors*. In addition to the aforementioned problems, all 3D style transfer methods only focus on transferring *high-frequency patterns* from the style image, such as brush strokes or color statistics. However, the notion of the style of an image is more nuanced and can significantly differ from image to image. In some cases, the style is mostly defined by large patterns that existing methods might miss—thus, reducing their effectiveness in transferring the intended style.

To overcome all these limitations, we introduce $\mathscr{G}$-Style, a novel method to transfer the style of an image onto a 3D scene represented using Gaussian Splatting. Our approach requires only a few minutes to optimize and delivers a high-quality stylized Gaussian Splatting representation of the scene. Our method comprises several steps: First, we

pre-process the scene represented with a set of Gaussians to ensure a uniform coverage. Then, our method starts the stylization process by updating the color associated with each Gaussian. During the stylization process, we enhance the resolution by splitting Gaussians with high color gradients, adding finer details where necessary. Due to the sparse nature of the scene and by only representing diffuse surfaces, our method provides view consistency by construction. Moreover, our dual loss function enables the algorithm to capture both high-frequency and low-frequency patterns in the style image, therefore generating high-quality 3D scene renderings that reproduce a large variety of artistic styles. Our extensive evaluation demonstrates that our approach outperforms existing methods in style transfer quality and rendering time.

4.2 Background: NeRFs and Gaussian Splatting

In this section, we briefly describe the two most commonly used 3D representations for novel view synthesis, NeRFs [MST+20] and Gaussian Splatting [KKLD23].

NeRFs. Neural radiance fields have revolutionized the field of novel view synthesis by introducing a new scene representation, and an optimization algorithm to train this representation only from images. The outgoing radiance at any point x in the scene for any view direction v is modeled by a parametric model $\phi_\theta(x, v)$ with parameters θ. This model is usually a multi-layer perceptron (MLP), which is chosen due to the universality provided by this type of model. To train this model, the scene is rendered from multiple views using the volume rendering algorithm [Max95]. By comparing the generated image $\mathcal{I}_\nabla$ to a real picture of the scene $\mathcal{I}_{gt}$, gradients can be computed for the parameters θ through the rendering operation and the scene representation can be updated to match the ground truth images. Despite the high-quality images generated by NeRFs, the control provided to the user is limited, and they demand extensive rendering times due to the multiple evaluations required to compute the value of a pixel.

Gaussian Splatting. Gaussian Splatting [KKLD23] has emerged as a viable alternative to address the main limitations of NeRFs: limited scene control and low rendering speed. To reconstruct a real-life scene from ground truth images $\mathcal{I}_{gt}$, Kerbl et al. use the Structure from Motion algorithm [Ull79] to obtain camera poses for each image and a sparse set of initial 3D points. The 3D points are then transformed into Gaussian functions, each representing a point in the scene. In this way, Gaussian Splatting represents the function $\phi_\theta(x, v)$ as a set of 3D Gaussians.

Each Gaussian is defined by its mean μ, covariance matrix Σ, opacity δ, and color c. By representing the color with spherical harmonics, view-dependent effects can also be captured. Since the covariance matrix Σ has to be positive semi-definite, which is difficult to enforce during optimization, the covariance matrix is obtained by RSS^TR^T, where S is a scaling matrix and R a rotation matrix obtained from a quaternion. The covariance matrices of the Gaussians are initialized accounting for their neighbors to conservatively cover the surfaces and prevent holes in the reconstruction.

The optimization process used in Gaussian Splatting is similar to the one used in NeRFs, where volume rendering is used to generate images by querying density and color along the rays emanating from the camera. However, due to the sparse nature of the representation achieved with Gaussian Splatting, the rendering process can be efficiently computed by projecting the Gaussians into the image and combining them using alpha blending. Furthermore, since the initial set of Gaussians may not be sufficient to capture all the necessary geometric and color details, the Gaussians are periodically split or cloned based on their accumulated μ gradient. However, this splitting is designed to represent the original scene and might not be sufficient for its stylized version. As we discuss in the upcoming sections, we propose a fast, high-quality, and consistent approach for stylizing 3D scenes by modifying the style and geometry of scenes represented by Gaussian Splatting with the style of an additional image or texture.

4.3 Methodology of $\mathscr{G}$-Style: Gaussian Splatting with Style

In this section, we describe our proposed algorithm: $\mathscr{G}$-Style. We first provide an overview of our approach (also illustrated in Figure 4.1), followed by a detailed explanation of its substeps.

4.3.1 Overview

Our algorithm takes as input a scene represented with a set of Gaussians $\mathcal{G}$ and a set of ground truth images $\mathcal{I}_{gt}$ and their associated camera poses $\mathcal{T}_{gt}$. Subsequently, it modifies $\mathcal{G}$ based on the style provided from a style exemplar $\mathcal{I}_s$. First, we *pre-process* $\mathcal{G}$ to remove long narrow Gaussians and Gaussians covering large areas, making the initial set of Gaussians uniform. Once the initial representation has been pre-processed, we create an additional color c_s associated with each Gaussian which is initialized with the original color c_{gt}. These new colors c_s are modified during a *stylization* process that uses a composition of several losses to preserve different properties of the style of $\mathcal{I}_s$. Since the geometry provided by the initial pre-processing step can be limited to represent detailed style features, $\mathcal{G}$ undergoes a *geometric fine-tuning* step. In this step, the Gaussians are split based on the gradient of c_s and fine-tuned to match the original scene images $\mathcal{I}_{gt}$ by modifying μ, Σ, δ, and c_{gt}. The stylization and geometric fine-tuning steps are repeated until convergence.

4.3.2 Pre-processing Step

Although Gaussian Splatting offers high-quality scene representation, the original approach has limitations. It often generates large flat Gaussians to depict uniform, flat surfaces (e.g., walls), and narrow elongated Gaussians to capture high-frequency details. The latter might also come as a byproduct of the optimization process. When incorporating an additional style into the scene, large flat areas might need additional Gaussians to incorporate extra details, while already highly detailed areas may not need more Gaussians to stylize them properly. Therefore, in the initial step of our approach, the

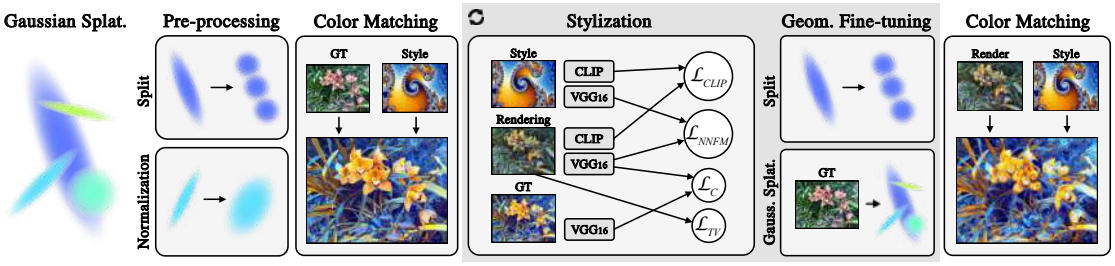


Figure 4.1: Overview of our method: We take a 3D scene represented using Gaussian Splatting and pre-process it to subdivide large Gaussians and normalize elongated ones. Initially, we perform a color matching between the ground truth images and the style image. Subsequently, we start the iterative stylization process. First, we optimize the colors of the Gaussians using multiple losses to capture style patterns at different scales while preserving the content of the scene. Then, we fine-tune the geometry of the scene and add details for Gaussians with a large gradient of the stylized color. We repeat the stylization and geometry fine-tuning steps until convergence. At the end, we perform an additional color matching step between the renderings of the resulting scene and the style image.

scene undergoes a Gaussian normalization process where the resulting representation $\mathcal{G}$ is composed of Gaussians of similar shape and size.

Flat Gaussian Split. To detect under-sampled areas, we compute the approximated maximum projected area of each Gaussian, A_i. We compute A_i by multiplying the two highest components of the Gaussian's scaling matrix S_i. Then, we mark for splitting all Gaussian above a threshold $t_f = \mu_A + \gamma\sigma_A$, where μ_A is the mean of all A in the scene, σ_A is the standard deviation of A, and γ is a user-defined parameter controlling the number of Gaussian mark for splitting. As splitting Gaussians modifies the geometry of the scene, we are obliged to optimize them. For this, we employ the same optimization algorithm as in the original work of Kerbl et al. [KKLD23].

Narrow Gaussian Normalization. To identify Gaussians with a narrow shape, we compute their elongation factor E_i by dividing the highest component of each Gaussian's scaling matrix S_i by its second highest component. If E_i is above a certain user-defined threshold t_e, we mark it for normalization, which sets the largest component to the average of the largest and second-largest components. We repeatedly perform this operation during the optimization process after the flat Gaussian split. This optimization process retrains the scene to match the appearance of the ground truth images $\mathcal{I}_{gt}$. As such, it corrects the deformations caused by this narrowing step while keeping the Gaussians more rounded. In Figure 4.2, we illustrate the effect of normalizing Gaussians (right) as opposed to not normalizing them (left).

Diffuse Color Transform. In 3D style transfer, where there is no concrete ground truth image associated with a view, enforcing multi-view consistency is key for seamless

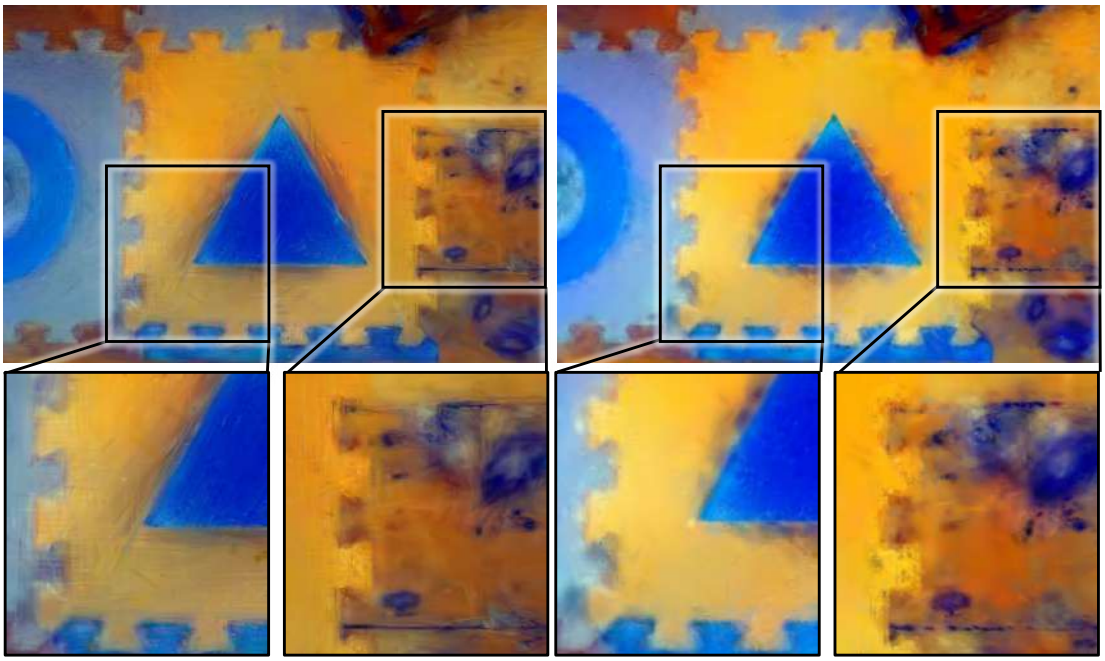


Figure 4.2: Gaussian Normalization: We normalize (right) the size of narrow Gaussians to avoid multiple overlying Gaussians (left).

navigation through the scene. In the original Gaussian Splatting algorithm, spherical harmonics enable modeling view-dependent effects such as reflections. However, they also generate undesirable artifacts on the stylized version of the scene, where each view direction could result in a completely different stylization. To avoid this, we only use the zeroth term of the spherical harmonics representation, limiting the model to represent diffuse objects only, therefore, enforcing view consistency.

Parameterizations. We perform five rounds of the splitting–normalization–optimization process, which in our experiments offers a good balance between the pre-computation time and the even distribution of Gaussian sizes. Initially, we set γ to 1.1, but in each subsequent round, we multiply it by 1.125 to deal with very large Gaussians that may still be remaining. The initial value for γ is quite low and, thus, we split at most 5% of the largest Gaussian in each round. Additionally, we set the threshold for elongation t_e to 1.5, which allows a degree of elongation but normalizes the shape of those Gaussians that heavily affect the final appearance. After pre-processing, the resulting set of Gaussians still matches the visual characteristics of a scene, but the distribution of their perceived sizes is uniform, as can be seen in Figure 4.3. Note that incorporating more Gaussians into the scene results in additional details—and additional memory demands. However, since we limit our representation to diffuse objects, the storage required to store the color of each Gaussian is reduced from 192 bytes to 12 bytes for the three components of the diffuse color. Our implementation, thus, allows us to increase the number of Gaussians to represent the scene while at the same time reducing memory consumption.

Figure 4.3: Pre-processing: The effect of our pre-training step. Left: before pre-training, right: after pre-training. The scale of Gaussians is set to 0.25, otherwise, both images would look identical.

4.3.3 Stylization Step

Once we have pre-processed the scene, we start the stylization process. During stylization, we iteratively render the scene from all $\mathcal{T}_{gt}$ using c_s and compute several losses over the images carefully designed to preserve the style of $\mathcal{I}_s$ at different scales. We hereby describe in detail the different losses used in our algorithm.

Low-frequency Style. The style of an image is determined by color and high-frequency details and also by large-scale patterns and features. To preserve these low-frequency features of the style image, we employ a CLIP-based [RKH+21] loss. We encode our rendered images $\mathcal{I}_r^k$ and the style image $\mathcal{I}_s$ into a feature vector using the image encoder of the CLIP model, $\mathcal{C}$. Then, our loss is defined as the similarity between these feature vectors where similarity is measured as the l_2:

$$\mathcal{L}_{CLIP} = \frac{1}{K}\sum_{k=1}^{K} \|\mathcal{C}(\mathcal{I}_r^k) - \mathcal{C}(\mathcal{I}_s)\|_2^2 \tag{4.1}$$

where K is the number of subimages extracted from $\mathcal{I}_r$.

High-frequency Style. Our CLIP-based loss can capture large-scale patterns in the style image. Yet, fine details, such as brush strokes in a painting, might not be well represented with this loss. Therefore, following Zhang et al. [ZKB+22a], we use a nearest neighbor feature matching (NNFM) loss utilizing a pre-trained VGG-16 network [SZ14] to capture the high-frequency style patterns. The architecture of a feature extractor can play a significant role in the quality of the generated results. We selected VGG-16 for its proven effectiveness in previous style transfer solutions, facilitating easier comparisons with those methods. The NNFM loss is a replacement for the widely used Gram matrix-based loss of Gatys et al. [GEB16]. Instead of matching statistics of feature maps $\mathcal{F}_r$ and $\mathcal{F}_s$, this loss searches for nearest neighbors in the feature space. Let $\mathcal{F}_r$ and $\mathcal{F}_s$ be

the VGG-16 features maps of $\mathcal{I}_r$ and $\mathcal{I}_s$ respectively, and let $\mathcal{F}(i,j)$ be the feature vector at pixel location (i,j). The NNFM loss is then defined as:

$$\mathcal{L}_{NNFM}(\mathcal{F}_r,\mathcal{F}_s) = \frac{1}{N}\sum_{i,j}\min_{i',j'} D(\mathcal{F}_r(i,j),\mathcal{F}_s(i',j')) \tag{4.2}$$

where N is the number of pixels in $\mathcal{F}_r$, and D is the cosine distance between two vectors. As in Zhang et al. [ZKB+22a], we use feature maps from the third block of VGG-16.

Regularization. To avoid the deterioration of features that are necessary for scene understanding, like in the work of Zhang et al. [ZKB+22a], we utilize a content loss $\mathcal{L}_C$, which is the l_2 loss between the features of rendered images $\mathcal{F}_r$ and ground truth images $\mathcal{F}_{gt}$. This loss also uses the features from the third block of VGG-16. Additionally, we incorporate a total variation term in our loss $\mathcal{L}_{TV}$ to prevent noise in the resulting renderings.

Complete Style Loss. Our final loss is a weighted sum of all the losses, given by the equation:

$$\mathcal{L} = \lambda_{CLIP}\mathcal{L}_{CLIP} + \lambda_{NNFM}\mathcal{L}_{NNFM} + \lambda_C\mathcal{L}_C + \lambda_{TV}\mathcal{L}_{TV} \tag{4.3}$$

We perform the stylization process for 15 epochs. Note that forward-facing scenes do not have as strict multi-view consistency requirements as 360° scenes due to the limited viewing angles of the ground truth images. As such, they converge more easily compared to 360° scenes. To account for the differences between these two types of scenes, we use different parameterizations. For forward-facing scenes, we use an exponentially decaying learning rate from 1e-1 to 1e-2, and for 360° scenes the learning rate decays from 1e-2 to 5e-3. Also, we set λ_{CLIP} to 10, λ_{NNFM} to 100, λ_C to 0.05, and λ_{TV} to 1e-4. For 360° scenes, we set λ_{NNFM} to 10.

4.3.4 Geometric Fine-tuning Step

In the pre-processing step, we place more Gaussians in the undersampled areas, making the Gaussians more uniform in size. This step is conservative enough to not overly increase the number of Gaussians and, thus, not oversample a given scene. With the new Gaussians, it is possible to synthesize features that otherwise could not be represented. However, the size of the Gaussians still limits the synthesis of very fine features in our stylized scene.

To overcome this, during the stylization process, we periodically split Gaussians based on their c_s gradient. Similarly to the work of Kerbl et al. [KKLD23], we keep an accumulation buffer $\mathcal{B}$ to store the norm of the gradient c_s. After each iteration and for each Gaussian, we add the norm of the c_s gradient to $\mathcal{B}$ and periodically split a user-defined percentage of Gaussians with the highest value. Since splitting Gaussians modifies the geometry of the scene, after each splitting, we optimize their μ, Σ, δ, and c_{gt} in the same optimization process as Kerbl et al. [KKLD23].

4.3.5 Style Color Matching

To ensure a similar color distribution between the resulting stylized 3D scene and the original style image, at the beginning of our algorithm, we match the mean and covariance matrix of colors from our ground truth images $\mathcal{I}_{gt}$ and the style image $\mathcal{I}_s$. Let C be a matrix containing pixel colors of the ground truth images of the scene $\mathcal{I}_{gt}$ and S be a matrix of pixels from $\mathcal{I}_s$, where each row is one pixel and columns are used for RGB components. We analytically solve for a linear transformation A, such that $E(AC) = E(S)$ and $Cov(AC) = Cov(S)$, and finally modify our initial Gaussian Splatting scene representation so that the rendered color images match the color-corrected ground truth images. Even though this color transfer step assumes unimodal distributions of colors, in our experiments, we empirically identified that it is sufficient for all tested style images. Since optimizing with a loss that considers the activations of hidden layers does not guarantee that the resulting colors are accurate to the style, we apply the same correction at the end of our algorithm to $\mathcal{I}_r$.

4.4 Results

In this section, we provide an analysis of the results produced by our method. Additionally, we offer a comparison to other leading state-of-the-art approaches. All of our results are generated using a modified 3D Gaussian Splatting codebase [Ker23], which is publicly available in our repository (https://github.com/AronKovacs/g-style).

4.4.1 Datasets

To evaluate our approach, we prepared a series of forward-facing scenes: *Flower*, *Horns*, *Orchid*, *T-Rex*, and *Fern* (employed in the work of Mildenhall et al. [MSOC+19]) and 360° scenes: *Playroom* [HPP+18], and *Truck* and *Train* [KPZK17] together with a variety of styles ranging from classical paintings to abstract images: *The Starry Night* by *Vincent van Gogh*, *The Scream* by *Edvard Munch*, *The Great Wave off Kanagawa* by *Hokusai*, *On White II* by *Wassily Kandinsky*, *The Kiss* by *Gustav Klimt*, and a colored image of the *Mandelbrot Set* created by Wolfgang Beyer (CC BY-SA). The scenes contain vastly different complexities in the represented stimuli—including highly detailed areas, such as the bookcases in the *Playroom*, but also flat white walls. Our generated results for the forward-facing scenes can be seen in Figure 4.4 and for the 360° scenes in Figure 4.5.

4.4.2 Comparison to the State of the Art

We compare our approach to three state-of-the-art approaches: a recent NeRF-based approach, Artistic Radiance Fields (ARF) [ZKB+22a], a recent zero-shot style transfer method for NeRF scenes, StyleRF [LZC+23a], and a recent preprint that performs style transfer for Gaussian Splatting scenes, StyleGaussian [LZX+24]. For this comparison, we used their official implementations and pre-trained checkpoints [ZKB+22b, LZC+23b, LZX+24]. We do not compare against Gaussian Splatting in Style [SGC+24] and

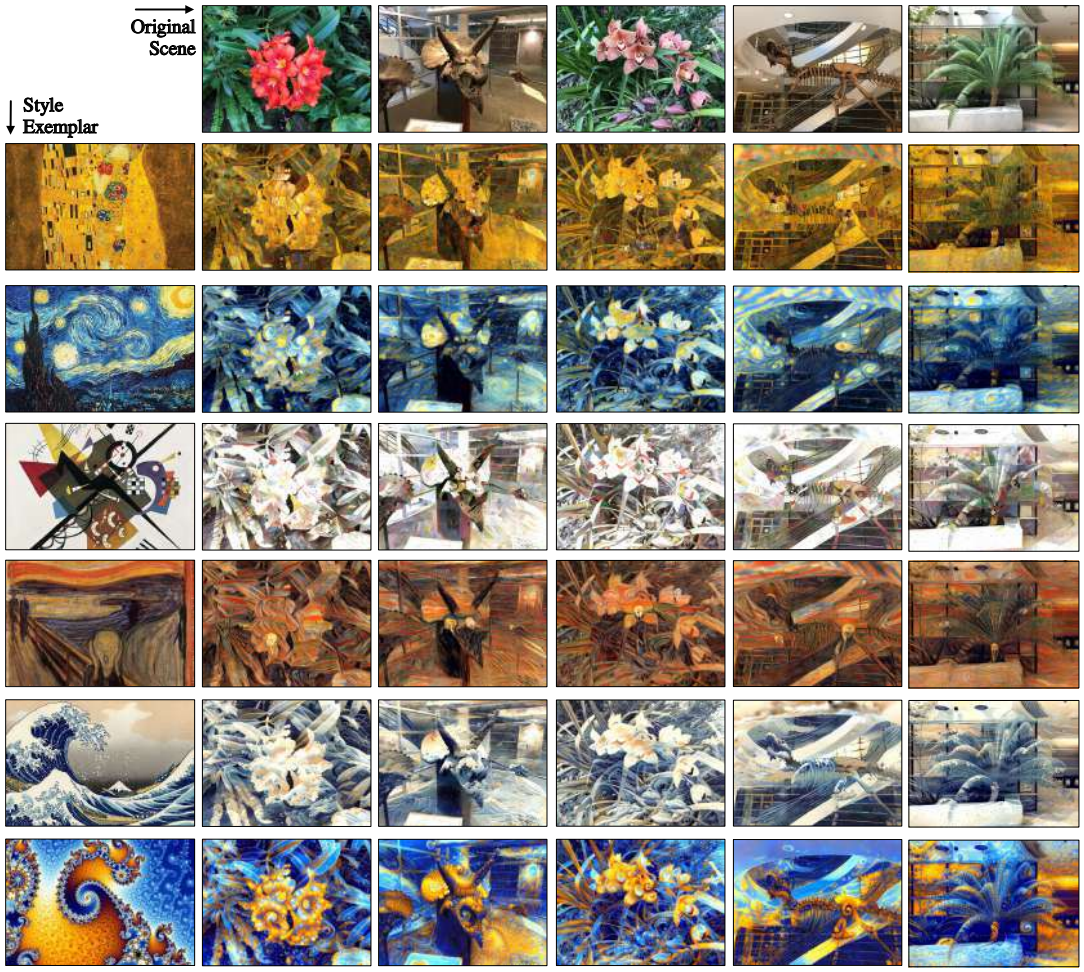


Figure 4.4: Results generated with our approach, $\mathscr{G}$-Style, for five forward-facing scenes (columns) given six style exemplars (rows).

StylizedGS [ZCY+24], as their implementations were not publicly available when the paper on which this chapter is based was written.

ARF uses the NNFM loss to transfer artistic style and also utilizes explicit color matching to achieve more accurate colors. This approach can use different volumetric scene representations as its backbone. However, the only publicly available implementation [LZC+23b] uses TensoRF [CXG+22], which in essence is a 3D voxel-based representation, decomposed into several lower-rank tensors. Other representations showcased in the original paper are neural radiance fields and Plenoxels [FKYT+22], but these could not be tested as they were not publicly available. StyleRF also uses TensoRF as its backbone. This approach is based on embedding high-dimensional features into the structure of the scene and, during rendering, uses them to transfer style. StyleGaussian is similar to StyleRF but uses Gaussian Splats as its backbone.

Figure 4.5: Results generated with our approach, $\mathscr{G}$-Style, for five 360° scenes (columns) given six style exemplars (rows).

Qualitative Comparison. A comparison for the forward-facing scenes can be seen in Figure 4.6 and for the 360° scenes in Figure 4.7. In these figures, we present the comparison of our method to the available checkpoints of other approaches for different scenes. Our approach exhibits better or, at least, comparable results to the other methods. Both StyleRF and StyleGaussian do not faithfully capture the visual style, because they are not able to synthesize small patterns (as is the case for the *Mandelbrot* style in Figure 4.6, second and seventh row) nor brushstrokes (for *The Starry Night* and *The Scream* styles) in Figures 4.6 and 4.7. These two approaches also fail to recreate any bigger patterns, such as in the two *Truck* scenes of Figure 4.7, especially in the stylization with *On White II*. Furthermore, StyleGaussian maintains the original Gaussians as resulting from the reconstruction phase. It is, thus, not able to create any meaningful patterns in undersampled areas, which can be easily seen on the walls in the examples of Figure 4.6, and on the ground or in the sky in all the examples depicted in Figure 4.7. Moreover, as shown in the *Train* scene in Figure 4.7, StyleGaussian produces large colorful Gaussians in the sky which do not match any of the used style images. Thus, for the scenes and styles we chose, StyleRF and StyleGaussian cannot reliably generate style-specific details and patterns; but rather focus on recoloring the scenes. Yet, the visual style of an image goes beyond just the colors.

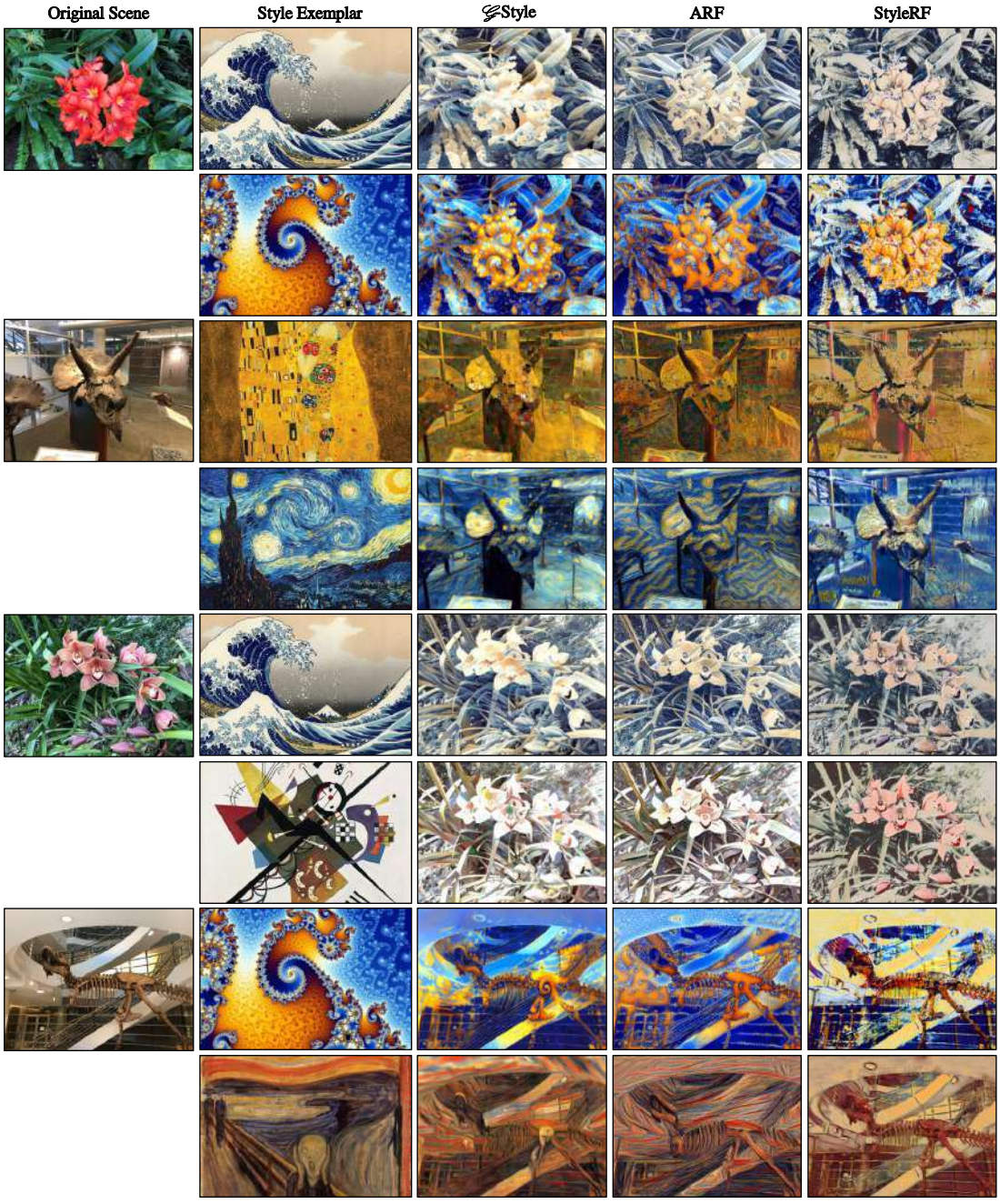


Figure 4.6: Results generated with our approach (𝒢-Style), ARF [ZKB+22a], and StyleRF [LZC+23a] (columns) for four forward-facing scenes and two style exemplars for each scene (rows).

Figure 4.7: Results generated with our approach ($\mathscr{G}$-Style), ARF [ZKB+22a], StyleRF [LZC+23a], and StyleGaussian [LZX+24] (columns) for 360° scenes and two style exemplars for each scene (rows).

Similarly to ARF, we produce high-frequency details, but by utilizing the CLIP loss, we can also create bigger patterns. This is something that ARF seems to struggle with. In the case of *The Starry Night* (in both Figure 4.6 and 4.7), our stylized scenes contain brush-like patterns, but also moon-like and star-like shapes not present in the results of ARF. When using *The Scream* our method occasionally also creates head-like shapes (see the femur of the *T-Rex* in Figure 4.6 and the surface of the *Train* in Figure 4.7). If this is not desired, it could be removed by modifying the style image and/or cropping it. Furthermore, *On White II* consists of simple patterns using only a single color. While our method can capture those patterns and create colorful shapes, ARF only produces desaturated areas, which are not truthful to the painting. This is evident in the sixth row of Figure 4.6 and the *Truck* in Figure 4.7. Finally, when using *The Kiss*, the flower pattern in Figure 4.6 is less blurry and more recognizable with our method, as opposed to ARF. We conclude that, while ARF can produce highly stylized images that match certain style images, it focuses only on fine details and ignores bigger patterns which may be important to capture a given style successfully.

Speed Comparison. We measured the optimization times of all the compared methods, and we performed all of our measurements on an NVIDIA L4 in Google Colab. The chosen approaches are fundamentally different: our approach and ARF [ZKB+22b] optimize directly the colors of a scene, while StyleRF [LZC+23a] and StyleGaussian [LZX+24] embed VGG features in the scene and later use them during rendering to stylize them. For the tested forward-facing scenes, our pre-processing step takes approximately 5 minutes, and stylization 3–8 minutes depending on the complexity of the scene, the number of ground truth images, and the size of the style image. For the same scenes, ARF needs 2–7 minutes. For the tested 360° scenes, the preprocessing step of our method takes 8 minutes, and stylization 20–28 minutes. For the same scenes, ARF requires 23–33 minutes. Note that depending on the particular scene and the quality of its reconstruction, we may not need to perform the pre-processing step, as its purpose is to ensure an approximately uniform distribution of the shapes of Gaussians. For StyleGaussian, the process when the embedded features are infused with the style information takes approximately 18 hours per scene, and the rendering of stylized images can be done in real-time. StyleRF needs 30–36 seconds per frame, which includes both style transfer and rendering.

4.4.3 User Study

To evaluate our method, we conducted an informal, online user study with 24 participants, where we used several of the cases shown in Figures 4.4–4.7. We presented each participant with the results of our approach, ARF, StyleRF, and StyleGaussian together with the respective style exemplars and original scenes. Without disclosing any information about any of the approaches, we interviewed the participants to gain feedback about the outputs. Specifically, we asked them to rank the approaches w.r.t. their similarity to the provided style exemplar. For each of the generated results, we also asked the study participants to rate their visual appeal and ability to recognize the original scene content on a 1–5 Likert scale.

The analyzed outcomes of the user study are shown in Figure 4.8. The study participants ranked our approach as most similar to the style exemplar (47.9% of the participants for the forward-facing scenes vs. 67.4% for the 360° scenes), followed by ARF (44.4% for the forward-facing scenes vs. 32% for the 360° scenes). StyleRF (7.6% for the forward-facing scenes vs. 0% for the 360° scenes) and StyleGaussian (0.7% for the 360° scenes) ranked last. The differences between our approach and ARF are statistically significant only for the 360° scenes, as shown with an ANOVA test followed by pairwise t-tests. The most visually appealing approaches are deemed to be ours and ARF, as indicated visually in the plots of Figure 4.8. However, the distributions of the ratings of our approach and ARF do not statistically significantly differ. In terms of content recognition, our approach and ARF are comparable to each other. Both are better than StyleRF for forward-facing scenes, but this changes for the 360° scenes, where the ratings of StyleRF (followed by StyleGaussian) are higher. This is to be expected as the effect of the latter two on the stylization of the scenes is less drastic than the former. To sum up, according to our study participants, our approach is comparable to ARF in all investigated aspects—yet, for 360° scenes, ours yields results more faithful to the style.

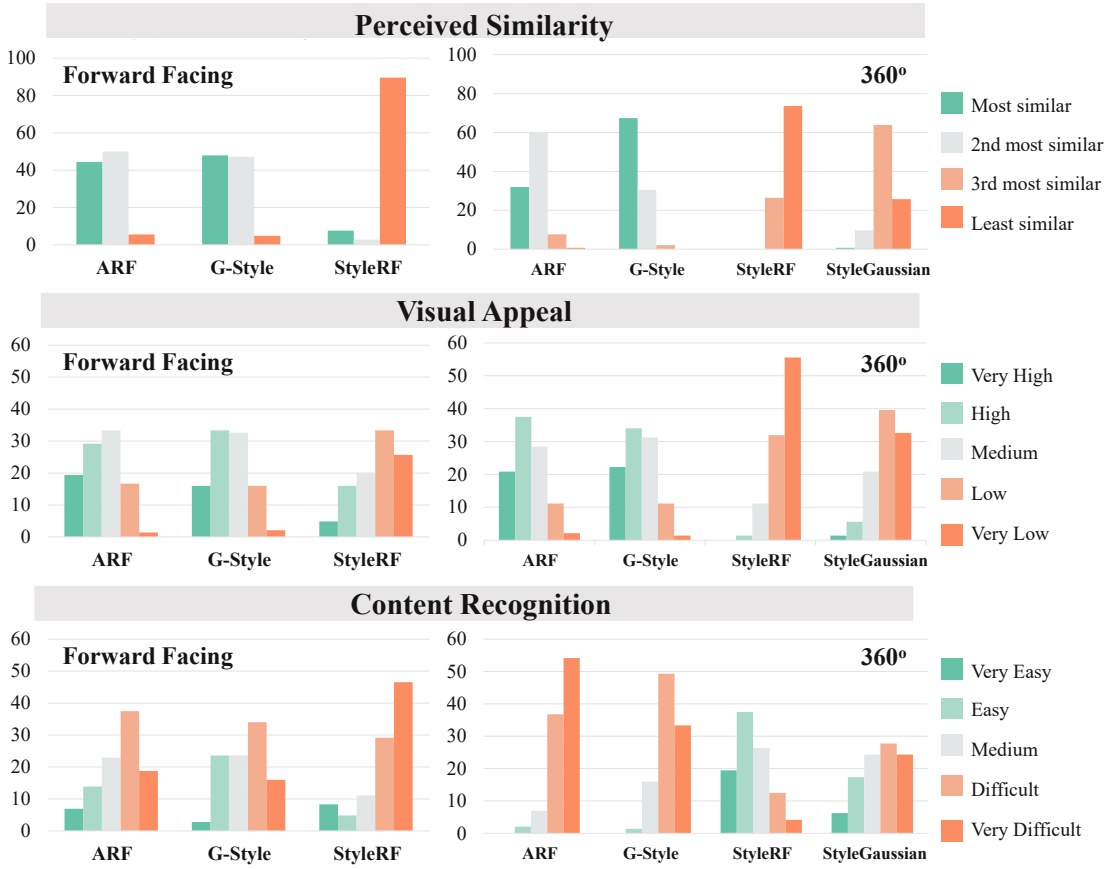


Figure 4.8: User study results: our approach ($\mathscr{G}$-Style) vs. ARF [ZKB+22a], StyleRF [LZC+23a], and StyleGaussian [LZX+24].

4.4.4 Ablations

Our loss does not enforce color transfer and relies on pre-trained networks to synthesize new features. Often, these new features are patterns that do not necessarily have the same color as in the style image. When we do not perform any explicit **color matching**—both during and after training—synthesized images are discolored (see Figure 4.9). Conversely, with color matching, the colors are truthful to the style image, as also shown in the work of Zhang et al. [ZKB+22a]. Introducing another style loss, $\mathcal{L}_{CLIP}$, still does not ensure that the optimization process matches the color statistic in synthesized images. **Without splitting**, the number and shape of Gaussians are the same as after the initial optimization with the ground truth images. This results in blurred areas, where not many Gaussians were originally needed, such as the walls marked in Figure 4.9. By splitting Gaussians, we are able to introduce details even in those areas. The **content loss** conditions the stylization. This loss helps to keep certain features intact or recognizable, e.g., the letter on the floor in the *Playroom* scene (see Figure 4.9). Furthermore, it helps to suppress noise, which is not present in the ground truth images. Moreover, the

Figure 4.9: Ablation studies performed for our approach.

CLIP loss enables our method to focus on bigger features. Without it, only very fine features are transferred. For example, without the CLIP loss, the "melting" patterns visible in *The Scream* are not synthesized as depicted in Figure 4.9. On the other hand, the **NNFM loss** focuses on high-frequency patterns. For instance, the brush strokes are not as visible when switching off the NNFM loss, as depicted in Figure 4.9. The pre-trained networks used for the style losses were not originally trained for this task but they were trained for classification. As such, using them for this purpose may cause them to produce noise-like artifacts (see Figure 4.9, no total variation loss). By using the **total variation loss**, we can remove these artifacts. If an overly smooth appearance is desired, this can be achieved by using an even higher weight for this loss, which needs to be tuned for this specific purpose (see Figure 4.9, high total variation loss).

4.5 Limitations

By utilizing Gaussian Splatting as the underlying data representation, we also inherit certain visual artifacts that are either caused by their construction process or are fundamental to this representation. Namely, when reconstructing a real-world scene, some "stray" Gaussians may appear as floaters, potentially inhibiting the stylization process. This could be remedied by using a modified version of the original Gaussian Splatting approach [KKLD23], which is currently an active area of research [CW24, FXZ+24, WYZ+24]. Furthermore, splatting can lead to compositing artifacts. Changing the viewpoint slightly may cause a Gaussian to pop in front of another one, given that during rendering the Gaussians are sorted based on the distance from their mean to the camera. Moreover, the stylization process is unguided, which means that an artist lacks full control over the placement of style features. Also, there is no straightforward way to ensure that patterns such as brush strokes point in the desired direction. To the best

of our knowledge, there are no reliable ways of selecting style patterns to reconstruct the content of a given image. This seems to be an inherent limitation of style transfer methods based on transferring the distribution of high-level features across images. This could be remedied in future work by considering more advanced style transfer models that can be conditioned to ensure this degree of control.

Furthermore, we rely on pre-trained neural networks to extract features based on which we transfer styles. However, there is no guarantee that the networks can truthfully capture the style features within their latent variables, which could lead to unconvincing results. According to our results shown in Figures 4.4 and 4.5, we are able to capture and synthesize the style of varied style images. Yet, with a more robust architecture, we might be able to do so more closely to the original style. Lastly, our approach relies on the stylization of 2D projected areas obtained with a differentiable renderer from a finite set of viewpoints. Depending on the distribution of those viewpoints, certain areas may take longer to converge or might not be fully optimized. Conducting an analysis to identify infrequently viewed areas and strategically placing new cameras in those locations could lead to faster convergence and improved results.

4.6 Conclusions

We introduced a novel algorithm for stylizing a 3D scene represented by a set of Gaussians to match the style of a given image, $\mathscr{G}$-Style. By optimizing the geometry of the scene based on the needs of the stylization process and by using a dual loss that captures high and low-frequency style patterns, we generate stylized scenes with higher quality than existing methods in a matter of minutes per style image. Future work could address the aforementioned limitations stemming from the Gaussian Splatting representation and the employed neural network architectures to further enhance the quality of stylized scenes. Another direction for future work is to explore editing, such as blending multiple styles into a single content image or conducting localized or semantic style transfer on distinct regions of the content image.

CHAPTER 5

Style Brush: Guided Style Transfer for 3D Objects

This chapter is based on a paper that, at the time of writing this dissertation, was under review. A preprint is available on arXiv:

> Áron Samuel Kovács, Pedro Hermosilla, and Renata G. Raidou. "Style Brush: Guided Style Transfer for 3D Objects." Under review, 2025. Preprint available at: https://arxiv.org/abs/2510.03433

In this chapter, we introduce Style Brush, a novel style transfer method for textured meshes designed to empower artists with fine-grained control over the stylization process. Our approach extends traditional 3D style transfer methods by introducing a novel loss function that captures style directionality, supports multiple style images or portions thereof, and enables smooth transitions between styles in the synthesized texture. The use of easily generated guiding textures streamlines user interaction, making our approach accessible to a broad audience. Extensive evaluations with various meshes, style images, and contour shapes, demonstrate the flexibility of our method and showcase the visual appeal of the generated textures.

5.1 Introduction

Style transfer refers to the process of applying a visual style of one image (e.g., a painting) to the content of another image, object, or scene—often by means of deep learning models [GEB16]. The image that provides artistic features, such as textures, colors, and/or brushstrokes, is referred to as *style*. The style is applied to the *content*, which is

the counterpart that contributes to the structure. In this case, individual salient parts, whole objects, or their arrangement, are kept, but modified so that patterns from the style are used to construct them.

This process is especially interesting in a 3D context, as it is often necessary to create a large number of simple objects that share a visual style. This is the case when game artists create backgrounds or environmental objects in a scene. This process can be both tedious and time-consuming, making it an ideal target for automation. Alternatively, style transfer could be used to investigate whether a visual style matches the artist's vision and suits their needs, before dedicating considerable resources to manually applying the style, possibly using the generated artifact as a starting point.

While relatively simple for images, using neural network-based approaches becomes much more complex for 3D objects and scenes. In contrast to simple 2D images, it is necessary to ensure multi-view consistency so that the patterns observed while moving in the scene stay coherent. With the currently available technologies, building patterns that span across significant parts of a 3D object or scene requires using a relatively slow optimization-based process so that many viewpoints are in agreement as to which patterns should be used where [MPSO18a, GRGH19a, HJN22a, ZKB+22a, ZFLS24, KHR24a].

However, when utilizing such powerful tools, many of the creative decisions are left up to the more or less fully automatic random processes. When artists use such a tool, they relinquish a lot of control, which may not be suitable for their projects. For this reason, some methods allow the user to keep a certain degree of guidance to influence the creation process, e.g., by determining which patterns should be used where, the size of transferred patterns, or the directionality of transferred patterns [WSZL19, RBS+22]. Unfortunately, to allow more complex guidance, e.g., determining directions of stroke-like patterns, existing methods either sacrifice quality or heavily constrain the type of possible interaction. We are also not aware of any method that would allow these types of complex interactions specifically in 3D.

In this chapter, we propose Style Brush, a novel artistic 3D style transfer method that allows the user to guide the stylization process by determining which style patterns should be used on which parts of the 3D objects, and the directionality of the synthesized patterns. Our approach is based on using a differentiable renderer to achieve coherent stylization across different viewpoints, delivering a high-quality stylized mesh in just a few minutes. For ease of use, we focus on using meshes, as creating *guiding textures* for them is a straightforward task that potential users can handle. Even then, our approach requires only basic painting of guiding lines and regions. In our evaluation, we show that our approach generates visually appealing mesh textures that respect the user-defined guidance, using a large variety of different textured meshes and styles. We evaluate our method with different kinds of contour shapes (e.g., straight lines, spirals, circles) and style regions, showcasing our method's flexibility.

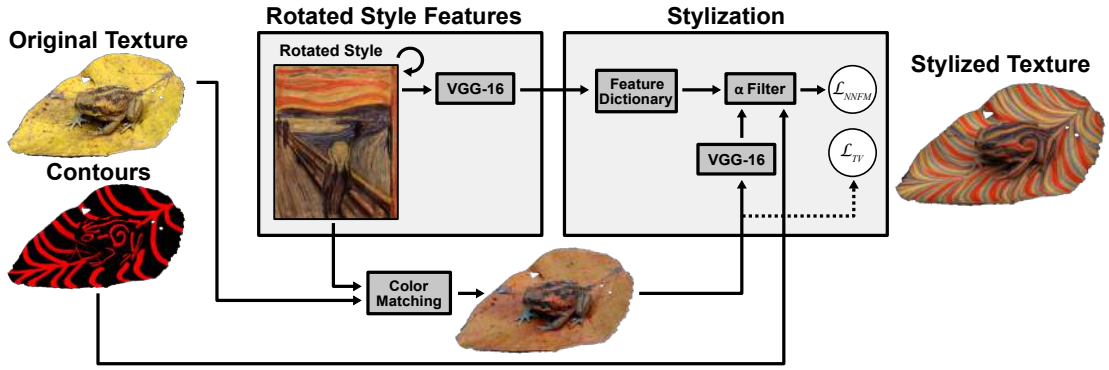


Figure 5.1: Overview of our method: We take a textured mesh, an additional texture with guiding lines, and a style image. Initially, we extract rotated style features by rotating the style image, passing it through a feature extractor, and assigning to each feature its directionality, thus building an angle-based feature dictionary. Next, we perform a color matching step between the style image and the original texture. Finally, we optimize the texture by rendering the mesh from multiple viewpoints, extracting features, and calculating the style loss by matching the rotated style features to the directions defined by the rendered contours. We minimize the total variation loss to suppress noise and repeat the entire process until convergence.

5.2 Methodology

In this section, we describe our proposed algorithm for the stylization of meshes with the use of sketch-like guidance. Firstly, we provide an overview of our approach, which is followed by a detailed explanation of each step.

5.2.1 Overview

A schematic overview of Style Brush is provided in Figure 5.1. Our algorithm takes as input a *mesh* with the original content $\mathcal{T}_\mathcal{C}$, a set of n *style images* $\mathcal{I}_\mathcal{S}^n$, a *directional guidance texture* $\mathcal{T}_\mathcal{D}$, and possibly a *style mask texture* $\mathcal{T}_\mathcal{S}$. The *input mesh* may already have an initial texture (e.g., colors, patterns, etc.), but our algorithm can also work without it. The *set of style images* are n 2D images that define one or more artistic styles to be applied to different regions of the mesh, such as painting styles, patterns, or textures that influence the final look of the textured mesh. The *directional guidance texture* provides directional information for applying the style (Figure 5.1, "Contours"). It can be used to control the flow of brush strokes or align patterns along a specific direction across the surface of the mesh. The *style mask texture* acts as a mask or segmentation map, indicating where the different style images should be applied to specific parts of the 3D mesh. Note that Figure 5.1 shows an example with only one style image and without style masks.

Our method renders multiple views of a 3D mesh and then iteratively modifies its texture with a differentiable renderer. To enforce the desired guidance, we ensure that the synthesized texture follows the direction indicated by the directional guidance texture.

To do this, we first compute the edge tangent flow [KLC07] of each style image, extract features with a neural extractor, and assign to each feature its direction based on the flow (Figure 5.1, "Rotated Style Features"). Before optimization, we match the color statistics of the texture $\mathcal{T}_\mathcal{C}$ to the style image, which speeds up convergence and ensures accurate color reproduction (Figure 5.1, "Color Matching"). Finally, we optimize the texture by applying a style loss that selects and minimizes the distance between the extracted features of the rendered texture and the style image, taking into account the guidance textures $\mathcal{T}_\mathcal{D}$ and $\mathcal{T}_\mathcal{S}$. Here, we also minimize the total variation of the rendered images to suppress noise (Figure 5.1, "Stylization"). The entire optimization process runs until convergence, ensuring that the final stylized texture faithfully reflects the desired artistic characteristics. Detailed descriptions of each of these steps are provided in the following subsections.

5.2.2 Dictionary of Rotated Style Features

Artistic style images tend to contain directional patterns, e.g., brush strokes, pencil strokes, pen strokes, etc. When applying these styles to a 3D mesh, the patterns must align with the directional guidance provided by the user. In our case, we determine the directionality of patterns by computing the edge tangent flow, an approach also utilized by Wu et al. [WSZL19]. Edge tangent flow (ETF) is a smooth direction field that corresponds to the flow of patterns in the input image. In our implementation, we utilize the ETF algorithm by Kang et al. [KLC07]. This algorithm is based on computing the gradient of an image and subsequently iteratively smoothing it, accounting for the gradient magnitude. Note that, in general, patterns flowing in a certain direction are indistinguishable from patterns going in the exact opposite direction. Thus, we treat them equally, meaning that we only need to consider orientations in the range of $[0, \pi)$ radians rather than $[0, 2\pi)$. This is also the case for the ETF algorithm, which outputs an image $\mathcal{I}_{\mathcal{ETF}}$ where each pixel is in $[0, \pi)$ radians, indicating the flow of features. The ETF algorithm is parametrized by the kernel radius used during smoothing and the number of iterations. The choice of these parameters impacts the extent to which small patterns can be detected, particularly when they flow in a different direction from their surrounding region [KLC07]. We set both of these parameters to 10.

Having computed $\mathcal{I}_{\mathcal{ETF}}$ for a particular style image, we discretize the directions into angle sets. Each angle set $\mathcal{S}_\alpha$ contains pixels having a similar direction α within a parametrized tolerance τ. We represent these angle sets as an integer image $\mathcal{I}_\alpha$. This discretization step helps us to organize the directionality of the patterns in a way that facilitates accurate matching, pairing, and application of patterns in the style transfer process. We set τ to 5°, meaning that the first set contains pixels with ETF of $[177.5°, 180°) \cup [0°, 2.5°)$, the second $[2.5°, 7.5°)$, and so on.

Once we have computed the angle set image $\mathcal{I}_\alpha$, we generate rotated versions of the style image and its I_α in increments of τ and process each with a pre-trained VGG-16 network. Having done so, we extract features from the VGG-16's [SZ14] hidden layers. Inspired by Kolkin et al. [KKP+22], we extract features from the first seven layers and resize the feature maps to $\frac{W}{4}\frac{H}{4}$, where W and H are the width and height of the input image. This choice is further motivated by the fact that we need to store features for every rotated

version of the style image, which is very demanding on the available GPU memory. This number can be further adjusted, thus having fewer or more style features, which may impact the stylization quality. Using the rotated I_α, we extract the features associated with each direction α and store them in feature sets F_α^S.

5.2.3 Stylization

Once we have prepared the style features and masks, we can start the stylization process. Our approach is render-based, which means we need to render the mesh from many different viewpoints, applying our style loss, and back-propagating the gradient to the mesh's texture. The object's texture can be either a pre-made one, $\mathcal{T}_\mathcal{C}$, in case it is desired that the resulting texture keeps its features, or a randomly initialized texture, in case there is no available texture or the original content is not important. For simplicity, we will refer to both as $\mathcal{T}_\mathcal{C}$. As our primary goal is to stylize single objects, we assume that a uniform distribution of viewpoints on a sphere around the object sufficiently covers all parts of the mesh. If that is not the case, an extension of our method may consider a different distribution of viewpoints to ensure better coverage of the object. We generate a uniform set of viewpoints using the Fibonacci sphere, rendering the mesh from the points on the sphere pointing towards the center of the mesh.

For each viewpoint, we compute which directional features to use for which part of the rendered image. Again, we utilize Kang et al.'s ETF algorithm [KLC07]. However, instead of computing the ETF of the rendered image, we allow the user to guide the ETF computation by defining a texture $\mathcal{T}_\mathcal{D}$, which contains guiding lines, as depicted in Figure 5.1 (see "Contours"). We compute the ETF in screen space. First, we render the mesh with the contour texture. In order to have a non-zero ETF everywhere, we detect the edges of the rendered contours and compute the distance to the nearest edge pixel, and we use this distance image to compute the ETF. We set the kernel radius and the number of iterations to 5, and then we discretize it into $\mathcal{R}_\alpha$ using the same τ from the style feature extraction process. Because $\mathcal{R}_\alpha$ and the directional feature sets F_α^S share the same discretization granularity, we can directly match and select which style features to apply to specific regions of the stylized image.

Our style loss is based on the work of Zhang et al. [ZKB+22a] and Kolkin et al. [KKP+22]. Both use style losses based on nearest neighbor feature matching (NNFM), utilizing a pre-trained VGG-16 network as the feature extractor to capture high-frequency style patterns. Unlike the widely used loss of Gatys et al. [GEB16], which uses Gram matrices of features to define the distance between styles, the NNFM loss finds the nearest neighbors in feature space and minimizes the distance between them. Let $\mathcal{F}^\mathcal{R}$ and $\mathcal{F}^\mathcal{S}$ be the extracted feature maps of a rendered image $\mathcal{R}_\mathcal{C}$ of the mesh using the $\mathcal{T}_\mathcal{C}$ texture and a style image respectively, and let $\mathcal{F}(i)$ be the i-th feature from the map. Then, the NNFM loss is defined by Zhang et al. as:

$$\mathcal{L}_{NNFM}(\mathcal{F}^R, \mathcal{F}^S) = \frac{1}{N} \sum_i \min_j \; D(\mathcal{F}^R(i), \mathcal{F}^S(j)) \tag{5.1}$$

where N is the number of features in $\mathcal{F}^\mathcal{R}$ and D is the cosine distance between vectors.

Kolkin et al. make further adjustments to the loss by finding the nearest neighbors for each *layer* separately, then combining them into a mixed feature vector, and aligning the means of the features, such that they are centered at 0. This allows the loss to synthesize more varied features and more effectively transfer patterns, depending on how well the style of the reference image aligns with the style of the optimized content image. Let $\mathcal{F}(L,i)$ be the i-th feature in the L-th layer of the feature maps. Then, the adjusted NNFM loss is defined as:

$$\mathcal{L}_{NNFM}(\mathcal{F}^R,\mathcal{F}^S) = \frac{1}{N}\sum_i \min_j \; D(\bigcup_L (\mathcal{F}^R(L,i) - \mu_L^R, \mathcal{F}^S(L,j) - \mu_L^S)) \tag{5.2}$$

where μ_L^R and μ_L^S are the means of the L-th layers of $\mathcal{F}^R$ and $\mathcal{F}^S$.

We make further adjustments to this loss by placing features into sets based on their directionality and only finding the nearest neighbors in the sets that have the same direction. Thus, our new style loss becomes:

$$\mathcal{L}_{NNFM}(\mathcal{F}^R,\mathcal{F}^S) = \frac{1}{N}\sum_\alpha \sum_i \min_j \; D(\bigcup_L (\mathcal{F}_\alpha^R(L,i) - \mu_L^R, \mathcal{F}_\alpha^S(L,j) - \mu_L^S)) \tag{5.3}$$

where $\mathcal{F}_\alpha^R$ was generated from $\mathcal{F}^R$ using $\mathcal{R}_\alpha$ with the same process that we used for style images without additional rotations.

Furthermore, we use the total variation loss $\mathcal{L}_{TV}$ to reduce noise. Note that we only compute total variation with non-background pixels to avoid bleeding of the background into the mesh. Thus, our final loss is defined as:

$$\mathcal{L} = \mathcal{L}_{NNFM} + \lambda \mathcal{L}_{TV} \tag{5.4}$$

where λ is the weighting coefficient for the total variation loss.

5.2.4 Style Color Matching

The NNFM loss exhibits the ability to synthesize high-level patterns, however, it struggles with correctly transferring color between the style image and the optimized texture. This can be explained by the fact that features of hidden layers do not necessarily carry all color information. The color of the converged texture $\mathcal{T}_\mathcal{C}$ heavily depends on its initial stage. For this reason, we match the distribution of colors between the texture and the style image, before we start the optimization discussed in Section 5.2.3.

Let C be the matrix $R^{N\times 3}$ of the used texels of $\mathcal{T}_\mathcal{C}$, i.e., the parts of the texture that may be potentially sampled during rendering, and let S be the matrix $R^{M\times 3}$ of the style image's pixels. As in the work of Zhang et al. [ZKB+22a], we solve analytically for a linear transformation M so that $E(MC) = E(S)$ and $Cov(MC) = Cov(S)$. We show the effectiveness of this step in our ablation studies (see Section 5.3.4).

5.2.5 Multiscale Feature Transfer

Images often contain patterns at multiple scales, e.g., ranging from fine brush strokes to larger structures composed of those strokes. To effectively transfer patterns at different scales, we employ a simple yet effective approach inspired by Kolkin et al. [KKP+22]. First, we downscale the texture $\mathcal{T}_\mathcal{C}$, the style images $\mathcal{I}_\mathcal{S}^n$, and render the meshes at a reduced resolution, all using the same downsampling ratio. Next, we perform style transfer, ensuring that only features recognizable by the feature extractor at the given scale are transferred. Finally, we upsample to match the next downsampling ratio, which is defined relative to the original sizes. We repeat this process iteratively until we match the original resolution. In our results, we downsample once by 2, although the number of intermediate steps is parametrizable by the user.

After each intermediate step, we blend the original texture at that scale with the newly generated texture using a blending coefficient β. This allows us to control the influence of larger patterns in the final result while preserving the original content's details, thus, it can also be thought of as a content weight. Note that this blending step is done with the color-matched version of the content texture to ensure a suitable distribution of colors. With this, we aim to encourage the formation of larger patterns.

5.2.6 Partial and Multiple Styles

Furthermore, we also enable the use of partial style images (i.e., only specific regions of the style image are applied) and multiple style images (i.e., applying different style sources to distinct parts of the object) through the use of style masks. We outline the adjustments made to our method to support these two cases.

Partial Styles. In case it is desired to only apply a part of the style image, the most straightforward way is to crop the style image. However, this confines the selection to a simple rectangular region. To allow for more control and flexibility, the user can create a binary style mask, which we apply during the creation of the rotated feature dictionary. By rotating the mask, we retain only the features within the masked area. The color matching step is only performed with the pixels from the active region.

Multiple Styles. To support multiple styles, we utilize the style mask texture $\mathcal{T}_\mathcal{S}$. We render this texture during stylization to determine which style image features should be considered in the nearest neighbor search when computing the style loss. Furthermore, $\mathcal{T}_\mathcal{S}$ is used to determine which parts of the texture are used for which style image during the color matching step.

5.2.7 Implementation Details

We set the render resolution to 512^2, unless the object is elongated, in which case we set it to 1024^2. We do this to ensure that the number of non-background pixels is approximately the same, so we achieve a similar level of detail for objects that are more round and for objects that are elongated (see Figure 5.2 and compare the Stanford bunny and the macaw). Furthermore, all of the textures we use are 2048^2.

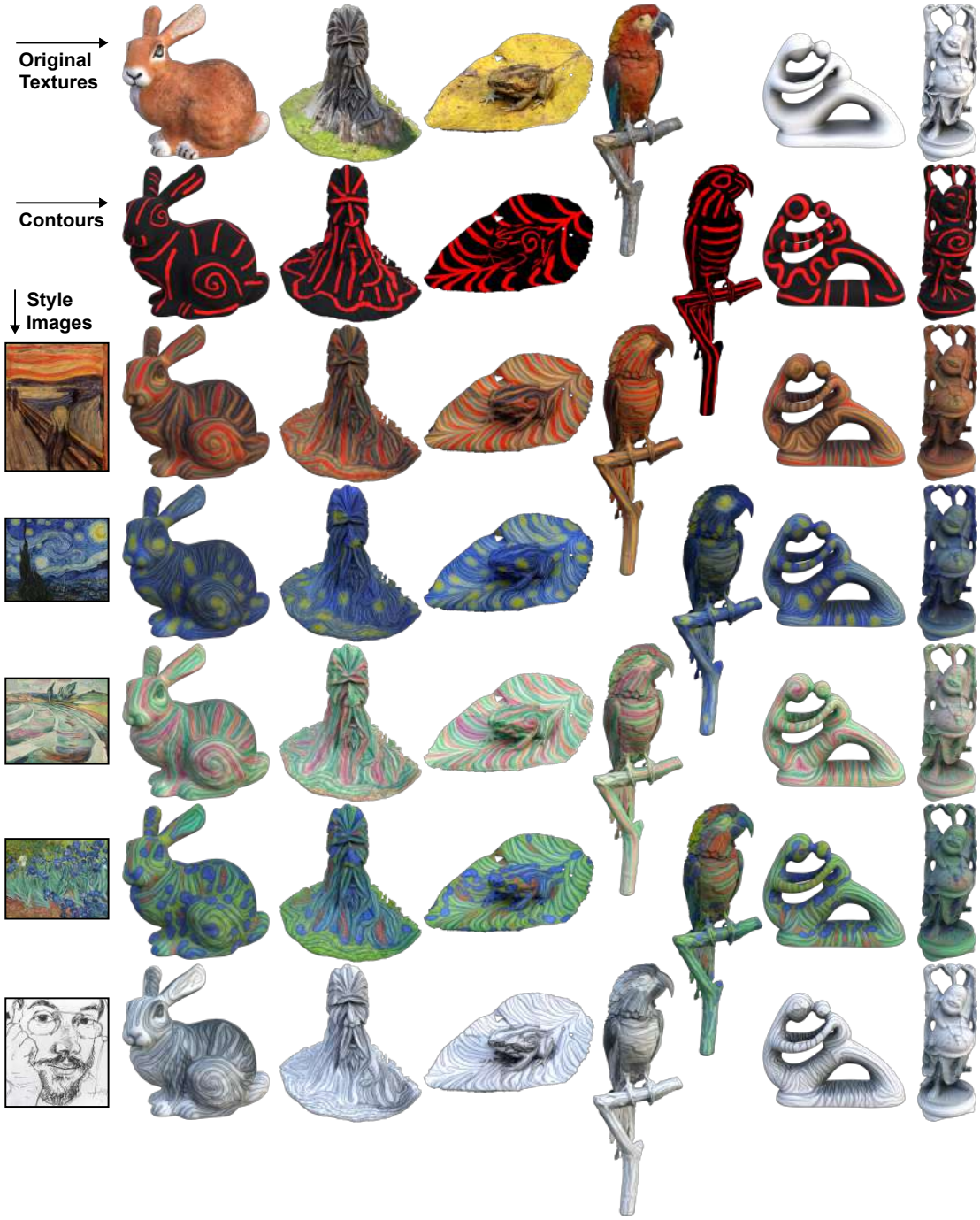


Figure 5.2: Results generated with Style Brush for six textured meshes (columns), using as input the initial textured mesh (first row), directional guidance in the form of contours (second row), and five style exemplars (last five rows).

Note that rendering the geometry and the computation of ETF are computationally expensive. For this reason, we precompute fragments, i.e., we associate pixels with texture coordinates, sample the guidance textures with the fragments, and compute the ETF. We use the precomputed fragments during the stylization. As our implementation of the ETF algorithm is CPU-based, computing many directional fields from multiple viewpoints is trivially parallelizable. However, this comes at the cost of the memory needed to store this data. To account for this, in our experiments, we generate 250 viewpoints around an object using the Fibonacci sphere. For the optimization, we set λ to 2e−5 and we use the Adam optimizer with the learning rate set to 0.01 and optimize for 1000 iterations for each multiscale step.

5.3 Results

In this section, we present an analysis of the results generated using our method. We implemented our method in Python and Rust, utilizing PyTorch and PyTorch3D. Our code will be made publicly available here https://github.com/AronKovacs/style-brush once our paper is accepted.

5.3.1 Dataset

We evaluated our method on a variety of publicly available meshes. The meshes used for the evaluation are the following: the Stanford bunny and Happy Buddha from the Stanford 3D scanning repository, the Green Man mesh by the user "gerg" licensed under CC BY 4.0, a marine toad on a leaf mesh by DigitalLife3D licensed under CC BY-NC 4.0, a cuban macaw mesh provided by the Natural History Museum Vienna licensed under CC BY-NC 4.0, and the Mother and Child mesh by the user "edcorusa" licensed under CC BY-SA 3.0. The models that were generated by scanning real-world objects were cleaned as appropriate to regularize their geometry. The Stanford bunny, Happy Buddha, and the Mother and Child meshes are not textured. We used a texture for the Stanford bunny taken from http://alice.loria.fr/index.php/software/7-data/37-unwrapped-meshes.html (archived at https://web.archive.org/web/20220714184311/http://alice.loria.fr/index.php/software/7-data/37-unwrapped-meshes.html, the original website is no longer available), and the textures for the Happy Buddha and Mother and Child meshes were derived from their ambient occlusion textures. The other meshes already have a texture.

Furthermore, we used five artistic style images that consist of directed patterns, i.e., brush strokes and pencil strokes: *The Scream* and *The Waves* by *Edvard Munch*, *The Starry Night* and *Irises* by *Vincent van Gogh*, and a self-portrait by *Alexandr Benois* with a mask shown in Figure 5.4. These have been chosen to reflect a diversity in patterns, colors, strokes, and other visual features.

5.3.2 Main Results

Using the aforementioned meshes and styles we obtain the results presented in Figure 5.2, showcasing the flexibility of our approach across diverse styles. For example, the swirling brushstrokes of *The Starry Night* (Figure 5.2, second row) are effectively used to stylize

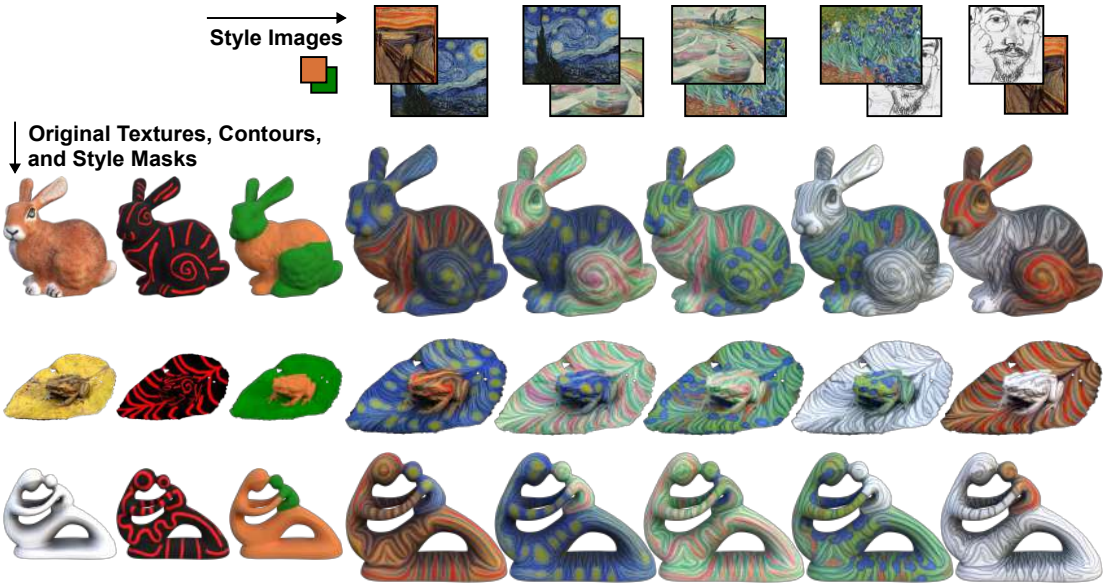


Figure 5.3: The results of our method for three meshes and five different styles, where two styles are combined. Here, two different style images are used on different parts of the mesh, defined by the style mask texture (indicated with orange and green). The style mask texture is given as an input to Style Brush, in addition to the original textured mesh and the guiding contours.

even meshes with intricate surface details such as the Green Man (Figure 5.2, second column) or the Happy Buddha (Figure 5.2, last column). Also, the more structured pencil strokes of *The Scream* (Figure 5.2, first row) or the *Irises* (Figure 5.2, fourth row) stylize our meshes following the given guidance, proving the versatility of our method in handling different artistic elements.

In Figure 5.3, we show how our method can be fine-tuned to apply only a portion of the style, allowing for partial adaptation and offering fine control over the final appearance of the stylized mesh. This flexibility is particularly useful when specific stylistic elements need to be emphasized. Moreover, our approach can seamlessly handle multiple styles applied to different regions of the mesh (see Figure 5.3 and Figure 5.4). This demonstrates how well our method adapts to complex and varied artistic inputs. By altering the directional guidance textures, we enable further customization, allowing for a high degree of control over how each style is represented, as seen in Figure 5.5.

In all those examples, we see that our method works effectively across different mesh topologies, handling both simple and complex shapes, making it a versatile tool for diverse applications. Overall, the combination of flexible style handling, control over directional guidance, and adaptability to different mesh structures makes our method highly effective in producing high-quality, stylized textures.

(a) (b)

Figure 5.4: (a) Two examples of our method: the one on the left uses the full style image, while the one on the right applies only the features corresponding to a mask (bottom right). (b) A stylized texture generated with our method, combining three style regions, each with a different style image.

5.3.3 Comparison to the State of the Art

To the best of our knowledge, there is no published research on guided artistic style transfer for 3D objects. As such, we are only able to compare our method with the work of Wu et al. [WSZL19], which is an optimization-based method for guided artistic style transfer in 2D.

Wu et al.'s approach is based on utilizing a differentiable ETF estimator and then using its estimate to compute an additional loss term, the mean squared difference between the desired directional field and the directional field of the currently optimized image. Other than this, it uses Gatys et al.'s [GEB16] style and content losses. As their code implementation is not public, we extended the implementation of Gatys et al. with Wu et al.'s directional loss utilizing a direct computation of ETF, which is differentiable, and employing the same differentiable renderer setup as we do for our method. In this way, we aim to reproduce their results and extend them to textured meshes.

Qualitative Comparison. A comparison between our method and the method of Wu et al. [WSZL19] can be seen in Figure 5.6. We compare these two methods using two meshes (the Stanford bunny and the marine toad on a leaf) and five style images.

We observe that even though the method of Wu et al. is capable of producing patterns partially resembling the style images, their generated textures are not as truthful to the style images as ours. For instance, notice the difference in the generated stars when using *The Starry Night* (Figure 5.6, second row) or the colorful brush strokes when using *The Wave* (Figure 5.6, third row). All of the generated textures using Wu et al.'s approach are densely covered with lines and curves aiming to resemble brush or pencil strokes, though

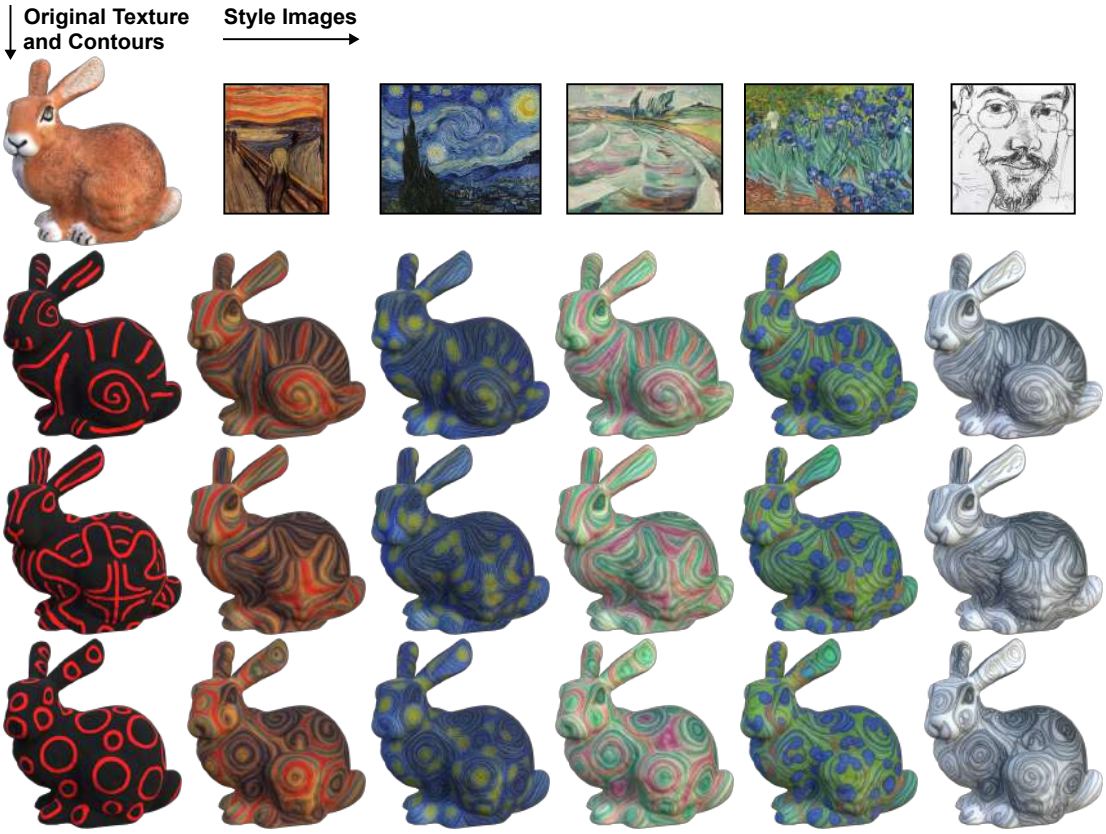


Figure 5.5: Results generated with our approach using three different directional guidance textures.

the patterns themselves fail to capture the visual quality of the brush or pencil strokes. For instance, notice the case of *The Scream* style (Figure 5.6, first row) or the *Irises* style (Figure 5.6, fourth row), where Style Brush excels in comparison to Wu et al.'s approach.

While we observe that the directions of the patterns generated with Wu et al.'s approach correspond to the guiding lines to an extent, they are more chaotic and ultimately do not closely match. This can be best seen with the last style image, the self-portrait by *Alexandr Benois* (Figure 5.6, last row). The results of both methods respect the desired flow of patterns, however, our approach exhibits a more faithful transfer of style features. This is evident with all selected styles in Figure 5.6 and is a direct consequence of how the two methods work, as the approach of Wu et al. is not able to force the synthesis of appropriately rotated patterns.

Furthermore, based on our experimentation with their method and the results in their paper, their choice of parameters seems to be suited for images with large uniform regions and with the user guiding the directional field in those regions. Note that even though the total loss is a combination of losses for style transfer (style loss and content preservation loss) and directional guidance, these losses are not informed about each other. Thus, the

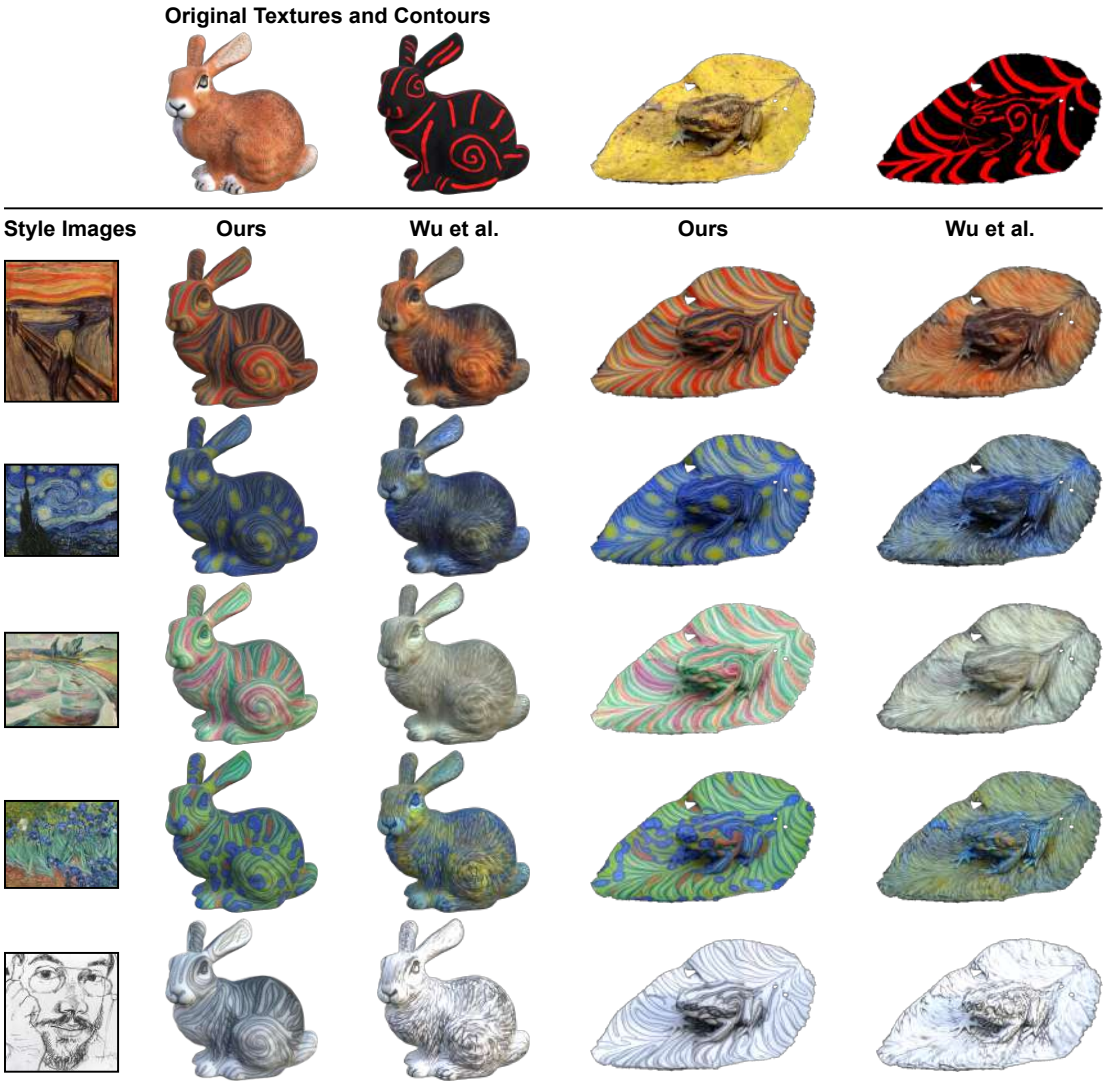


Figure 5.6: Results generated with our approach compared to those generated by the adapted (from 2D to 3D) method of Wu et al. [WSZL19] for two textured meshes (the Stanford Bunny and the marine toad on a leaf mesh) and five style images.

method relies on the directional loss creating directional patterns that may or may not be utilized by the style transfer losses to synthesize appropriately rotated style patterns.

5.3.4 Ablation Study

We performed several experiments to evaluate our method and the impact of different parameters on the results, which can be seen in Figure 5.7. We selected the Stanford bunny as the mesh and *The Starry Night* and *The Scream* as the style images, because the impact of certain parameters may not be as strongly visible using only one of the styles.

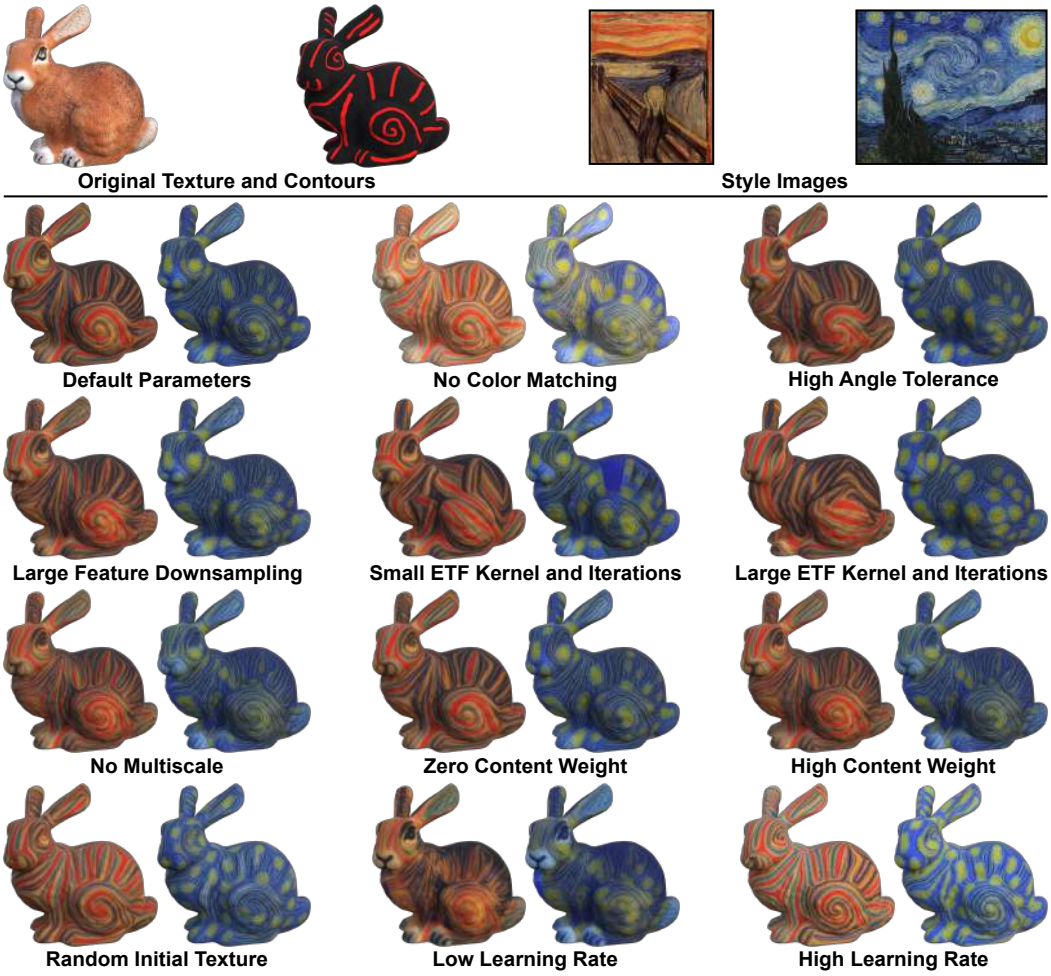


Figure 5.7: Ablation studies for our approach.

Color Matching. When we do not perform the explicit color transfer step at the beginning, the resulting textures are discolored. The effect of color matching is shown in Figure 5.7 (compare "No Color Matching" and "Default Parameters"). Our loss does not explicitly enforce color transfer between the style images and the generated texture—only VGG-16 features of the first few layers. While these layers contain information about colors, just matching the features does not accurately match color statistics.

Angle Set Granularity. During the extraction of style features, we assign them to individual angle sets with a tolerance τ. Our default choice of τ is 5°, in our ablation study, we show the result for 45°. The effect of τ is shown in Figure 5.7 (see Figure 5.7, "High Angle Tolerance" vs. "Default Parameters"). Even though τ is much higher, we still observe that the synthesized patterns approximately correspond to the user-defined directional field, however, the spiral at the hind leg has fewer turns, and the lines at its back are crooked. The choice of τ impacts the ability of our method to correctly match

the user-defined directional field, with higher granularity, the discretized directional field better corresponds to the continuous directional field, albeit at the cost of more memory and higher optimization time.

Feature Downsampling. During the extraction of style features and then during the optimization process, we downsample the extracted features so that all of the feature maps are $\frac{W}{4}\frac{H}{4}$, where W and H are the width and height of the image. For comparison, we show the result of downsampling by 8—therefore 64 times fewer features—and we observe that, when compared to being downsampled by 4, the brush strokes are less defined and there are noise-like artifacts present (see Figure 5.7, "Large Feature Downsampling" vs. "Default Parameters"). The optimization process relies on finding nearest neighbors in feature space—thus, by having fewer of them, fewer patterns and details may be transferred.

Edge Tangent Flow Parameters. We use the edge tangent flow to estimate the directions of the patterns in style images and also to compute the directional field of the contours. Kang et al.'s [KLC07] algorithm is parametrized by the kernel radius and the number of iterations, both influencing how smooth the resulting field is and how big the patterns must be to be recognized as flowing in a different direction than their surrounding. By default, we set these parameters to 10 for style images and to 5 for contours. We present two experiments, in the first one we set the kernel radius and the number of iterations to 1, and in the second one we set them to 20 (see Figure 5.7, "Small" vs. "Large ETF Kernel and Iterations"). In both cases, we observe that the patterns do not flow in the desired direction, unlike with the default parameters we use. While the choice of these parameters is to a certain extent subjective, they need to be large enough to be resistant to noise but small enough to assign meaningful directions to patterns so that they may be properly recognized as flowing in a certain direction. Furthermore, we rely on extracted features computed by a convolutional neural network, thus, the features correspond to *overlapping areas* in the images, not just *single pixels*. For this reason, using a small kernel radius and a few iterations may result in a directional field that is of a higher frequency than the feature maps, making the directional field inaccurate for the extracted features.

Multiscale Style Transfer. We optimize the texture twice, once at half of the original resolution and once at the original resolution, synthesizing patterns at different scales. When upsampling to start the second stage, we combine the original color-matched unoptimized texture with the generated texture. We perform three experiments (see Figure 5.7, "No Multiscale" vs. "Zero" vs. "High Content Weight"). In the first one, we only perform the optimization process at the original resolution ("No Multiscale"), in the second one, we do not blend the optimized texture with the original texture ("Zero Content Weight"), and in the third one, we blend with a factor of 0.95, giving more weight to the original texture ("High Content Weight"). In the first experiment, we observe that the generated patterns are smaller, namely the stars when using *The Starry Night* as style. In the second one, we see that the stylization is stronger and less content is preserved compared to our default choice of the blending weight, which is 0.25. And in the third

one, we observe that much of the original content is preserved and the stylization is much weaker. Ultimately, the choice of this parameter is a matter of personal choice and depends on the user's intended outcome.

Randomly Initialized Content Texture. Instead of using a texture that contains some kind of meaningful content, e.g., a texture created by an artist, we can also use a texture that was initialized with noise (see Figure 5.7, "Random Initial Texture"). In that case, the synthesized patterns are random, their randomness corresponding to the hints of patterns in the random texture that will be amplified during the optimization process while being constrained by the directional guidance and, of course, the choice of the style image. For this experiment, we used a random uniform noise $[0, 1]$, and we observed that with the original content texture, the mouth and paw areas still clearly resemble a mouth and paws, which is not the case for the result generated with the random texture. Similarly, in the original texture, the bunny's back contains some darker flecks, which result in darker stylization in those areas, however, when using the random texture, the areas have approximately the same distribution of patterns as the rest of the texture.

Learning Rate. The loss we use is based on approaching nearest neighbors in a very high-dimensional feature space. By modifying the learning rate (see Figure 5.7, "Low" vs. "High Learning Rate"), we influence how quickly the optimized features approach their nearest neighbors, and if the learning rate is large enough, they can overshoot and the new nearest neighbors may be different than the original ones as noted by Kolkin et al. [KKP+22]. Thus, by changing the learning rate, we can modulate how much the original content is preserved, although with a higher learning rate, noise-like artifacts start to appear which may be partially compensated for by increasing the weight of the total variation loss. With a smaller learning rate, the original content is more preserved.

5.3.5 Performance

We measured the optimization time of our method on a PC with an NVIDIA GeForce RTX 4080 SUPER GPU with 16 GB of VRAM. Computing the rotated style features of the style images we use takes approximately 15 seconds. With the default parameters and when using only a single style image, Style Brush takes approximately 8 minutes to create a stylized texture, of which the precomputation part takes 1 minute and the optimization takes 7 minutes. When using multiple styles, the optimization takes n times longer for n style images.

5.4 Limitations

By utilizing a loss based on finding nearest neighbors in feature space, we also inherit its limitations. Namely, the nearest neighbor search has a time complexity of $O(n^2)$, which is relatively slow. An approximate nearest neighbor search [JDS10, ZTL20] might offer a speed up at a potential loss of stylization quality, though a proper evaluation should be performed first to evaluate its impact. Furthermore, these nearest neighbor

losses generally utilize a simple CNN, e.g., VGG-16, to extract features. However, more advanced neural approaches for style transfer might utilize different architectures, e.g., transformers, or a completely different way to approach this problem, e.g., by utilizing a diffusion model. In such cases, assigning directionality to the extracted feature vectors like we do, may not be feasible, and a fundamentally different way to offer directional guidance might be necessary. Moreover, GPUs of today have a relatively small VRAM. We rely on extracting and storing style features of a large number of rotated style images. When using large style images or using many of them, we may have to rely on the slower RAM and then copy the style features to the VRAM on demand, or drop quality by downsampling style features, or increasing the angle tolerance. Lastly, we utilize the Fibonacci sphere to place cameras around the object, which may cause certain areas to take longer to converge than necessary if they are not frequently seen from the viewpoints. A detailed analysis of the object or scenes to place cameras could lead to a faster convergence.

5.5 Conclusions and Future Work

We introduced a novel algorithm, Style Brush, for stylizing textured meshes to match the style of given style images while respecting additional directional guidance. By optimizing the texture with a nearest neighbor-based loss that filters style features on their directionality, we allow the users of our method to guide the stylization process to create high-quality stylized textures in a few minutes. To our knowledge, we propose the first method that allows this kind of guidance for 3D objects. In future work, we propose to explore alternative networks for guided style transfer and potentially extend our method to networks that do not easily associate extracted feature vectors with their directional properties. Finally, one could consider modifying the geometry of the provided mesh to better suit the provided style.

CHAPTER 6

Conclusions

In this dissertation, we explore different ways of performing artistic style transfer for 3D objects and scenes with a focus on lifting 2D methods to 3D space and enabling guidance. This chapter serves as a reflection of the presented results and discusses directions for future work.

6.1 Reflection

This dissertation aims to answer three research questions posed in Chapter 1. Chapters 3, 4, and 5 have answered these questions, which we summarize below.

Q1: How do we lift a 2D texture synthesis method to 3D by working directly on the surface of a mesh? In Chapter 3, we propose a conceptually simple way of generalizing 2D convolutional and pooling layers so that they can operate on curved surfaces embedded in 3D space. This way, we can directly reuse convolutional 2D networks for 3D objects, achieving results that are either better or the same as other competing methods, which we showed in our evaluation. Our approach has the possibility to generalize beyond style transfer and future research may try to extend this approach to other tasks, such as supersampling. Nevertheless, our method provides proof that there is another useful way of projecting 3D scenes to 2D, not just using a differentiable renderer [MPSO18a, HJN22a] or taking slices [GRGH19a], but also by directly considering the curved surface. The choice depends on the specific task, as all of them have advantages and disadvantages, with our approach struggling with noisy meshes that may have many disconnected parts and holes, thus further straying from the assumption that meshes are locally flat.

Q2: How do we lift style transfer methods to modern volumetric representations? In Chapter 4, we present a novel method for stylization of Gaussian Splatting [KKLD23] called $\mathscr{G}$-Style. Unlike in the work presented in Chapter 3, Gaus-

sians do not have a properly defined surface, which is also the case for other volumetric representations. For this reason, it is not possible to apply a surface-based style transfer method. Furthermore, in the scenes typically used for Gaussian Splatting, only the surfaces of objects are of interest, specifically, the Gaussians near those surfaces. As a result, taking slices is not meaningful, since we are not interested in the interior regions of objects. As such, using a differentiable renderer to create proxy 2D images was the only suitable option. Moreover, the geometry of the scenes is usually constructed with an optimization-based process based on real-world images and can contain noisy parts and optimization artifacts. However, when rendered from the predefined viewpoints used during optimization, these still closely match the ground truth. These artifacts can quickly become apparent when we modify the colors, which is the case for style transfer. For this reason, we need to perform adjustments and corrections that are specific to the chosen volumetric representation so that when we change its colors and render them, they are free of artifacts. In the case of Gaussian Splatting, this corresponds to detecting undersampled areas, increasing the number of Gaussians there, and regularizing the shape of Gaussians. Other representations may have their own artifacts that need to be accounted for, and thus require representation-specific solutions. However, we show that when we account for the artifacts and use a differentiable renderer, we can create high-quality stylized scenes.

Q3: How do we empower the user to guide the stylization process? In Chapter 5, we introduce a new approach to guide the stylization process of 3D objects. Our approach is based on a simple but powerful idea of having a large set of patterns and selectively applying only those that correspond to the user's guidance, thus applying a novel guidance-enabled 2D style transfer method on proxy images created with a differentiable renderer. We allow the user to guide the directions of patterns and which parts of the style images should be used on which parts of the 3D objects. We accomplish this by enabling the user to create guiding lines, to specify the directions, and to paint style regions, obtaining control over the placement of individual style parts. This way, we allow the user to interactively steer the randomness inherent in the stylization process, giving them control to match their vision. Our method shows that performing this guidance in 2D is sufficient, which is not necessarily guaranteed, as the projected guidance may not match the unprojected guidance. However, our method is sufficiently robust to handle the domain change.

With these focused research questions answered, we address the overarching research question as follows:

How can we develop user-guided, 3D-aware style transfer methods that lift 2D techniques to modern 3D representations?

We show that working with modern 3D representations requires special care to effectively use them and produce high-quality results. This is a direct consequence of their nature, as they are usually specialized for different tasks than style transfer, and they are often produced with optimization-based processes that may create artifacts on a local level [MST+20, FKYT+22, CXG+22, KKLD23]. Addressing these issues thus depends

on the representation used. In our work, we focused on Gaussian Splatting and on meshes, as they are still the industry standard. For Gaussian Splatting, we introduce an approach for high-quality stylizations focusing on different levels of detail of style patterns. For meshes, we propose two methods. The first one introduces an innovative way of generalizing convolutional and pooling layers to work directly on the surface and the second one introduces a method to allow the user to guide the stylization process.

With these three approaches in place, we answer the overarching research question within the context of modern research by showing that high-quality, user-guided, 3D stylization emerges from a process combining: (i) representation-aware lifting of 2D operators, whether via surface-aware CNN operators, differentiable rendering, or slicing so that existing 2D style transfer methods can be reused for 3D assets, (ii) representation-specific adaptation, in which each 3D representation's unique sampling or optimization artifacts are detected and corrected (e.g., by resampling and regularizing Gaussians) to ensure clean stylized outputs, and (iii) interactive guidance, which empowers artists through intuitive tools that provide fine-grained control over the style transfer process without sacrificing stylization quality. Together, these three components make it possible to develop powerful, user-guided 3D style transfer methods that preserve the strengths of 2D techniques while ensuring consistency, quality, and control in the 3D domain.

6.2 Future Work

In this section, we discuss possible future directions and applications of 3D style transfer, continuing the work presented in this dissertation.

Geometry Modification. In this dissertation, we focus on stylizing 3D objects by changing their colors, and not by changing the appearance of geometry. Note that in the second project, $\mathscr{G}$-Style, we change the geometry by changing the shape of Gaussians and adding new ones, but we ensure that their appearance matches that of the initial distribution. However, changing the geometry may be necessary to better suit certain styles, e.g., by creating spikes, grooves, or by adding new objects. While there has already been some work [HHGCO20], it mostly focuses on modifying the surface of objects, adding new objects to scenes would probably involve combining several approaches, e.g., consulting an LLM to get a list of objects that would fit the style, synthesizing them, and then placing them at the right locations in the scene. Style transfer could thus become another tool for large-scale, varied procedural generation, which could be utilized by video game developers making open-world games. In this setting, a style-aware generator would not only recolor existing assets but could also propose and instantiate new geometry, such as decorative props, vegetation variants, and carved motifs that reinforce the global art direction across various parts of the generated worlds, e.g., biomes and settlements.

Large Patterns and Space-Time Coherency. Currently, in order to synthesize large patterns that span significant portions of the scene, even across multiple objects, it is necessary to use relatively slow optimization-based methods to build visual patterns, and possibly dissolve them if it is not possible to reach consensus across different viewpoints

or slices. This results in the methods being unnecessarily slow and solving this possibly means more than just using light-weight neural networks, having faster implementations, or faster hardware. This entails rethinking how to approach this problem and use a fundamentally different way of synthesizing patterns that need to be consistent and coherent with themselves and each other. Furthermore, an extension of style transfer for scenes is style transfer for videos. This is challenging as it needs to deal with the same problems and additionally needs to consider the evolution of scenes through time and deal with object permanence, i.e., objects that can be seen, then they are out of view, and then can be seen again, need to be stylized the same way across their appearances on the screen. AR/VR raises a distinct, demanding set of requirements for scene-level style transfer. Stylization must be coherent across space and time, and it must be applied in real-time. Furthermore, many AR devices, whether standalone glasses or phones connected to them, have limited processing power, memory capacity, battery life, and tight thermal constraints. If server-side computing is not a desirable option (e.g., because of privacy concerns), highly efficient methods are necessary for such applications.

Interactivity. In Chapter 5, we present a method for guided style transfer with a main focus on the synthesis of directed patterns and the ability to pick and choose styles from different images or just their parts. The interactions are based on user-provided textures with guiding lines and regions determining the properties of the to-be-synthesized patterns. Future work in this direction entails interviewing artists to determine in detail how to help them fulfill their creative vision, with an emphasis on the way they interact with their media and which parts of their process can be either enhanced or replaced with their automated versions, possibly incorporating further guidance. This may involve long-term studies and developing specifically designed tools for concrete artists or teams of artists, and then generalizing them to a wider audience. Furthermore, style transfer could become integrated directly into asset-authoring pipelines. This would enable artists to produce multiple style variants automatically, populate scenes with style-consistent props, or generate different versions for A/B testing. This would reduce manual retouching, iteration time, and allow artists to fine-tune the style of their assets. With advances in processing power and with new, faster methods, live-preview plugins and in-editor tools would allow artists to steer the stylization of scenes, lock individual regions, or have more control to propagate patterns across scenes. Real-time or near-real-time feedback would encourage exploratory workflows where an artist can try dozens of alternatives quickly, while minimizing repetitive tasks.

In summary, this dissertation demonstrates how 2D style transfer methods can be faithfully and flexibly extended to 3D, through careful lifting of existing operators, tailored adaptations to representation-specific artifacts, and artist-centric guidance mechanisms, thus pushing forward the boundaries of style transfer to yet unexplored areas.

Overview of Generative AI Tools Used

We used Grammarly to check grammar and spelling.

List of Figures

List of Tables

Bibliography

[AHS+21] Jie An, Siyu Huang, Yibing Song, Dejing Dou, Wei Liu, and Jiebo Luo. Artflow: Unbiased image style transfer via reversible neural flows. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 862–871, 2021.

[BMT+21] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. *ICCV*, pages 5835–5844, 2021.

[CCL+18] Liang Cheng, Song Chen, Xiaoqiang Liu, Hao Xu, Yang Wu, Manchun Li, and Yanming Chen. Registration of laser scanning point clouds: A review. *Sensors*, 18(5):1641, 2018.

[CHH24] Jiwoo Chung, Sangeek Hyun, and Jae-Pil Heo. Style injection in diffusion: A training-free approach for adapting large-scale diffusion models for style transfer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8795–8805, 2024.

[CNS+11] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. Interactive indirect illumination using voxel cone tracing. *Computer Graphics Forum*, 30(7):1921–1930, 2011.

[CS16] Tian Qi Chen and Mark Schmidt. Fast patch-based style transfer of arbitrary style. In *arXiv preprint arXiv:1612.04337*, 2016.

[CW24] Guikun Chen and Wenguan Wang. A survey on 3D gaussian splatting. *arXiv preprint arXiv:2401.03890*, 2024.

[CWNN20] Xu Cao, Weimin Wang, Katashi Nagao, and Ryosuke Nakamura. PSNet: A Style Transfer Network for Point Cloud Stylization on Geometry and Color. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 3326–3334, 2020.

[CXG+22] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision (ECCV)*, page 333–350, 2022.

[CYL+25] Yaosen Chen, Qi Yuan, Zhiqiang Li, Yuegen Liu, Wei Wang, Chaoping Xie, Xuming Wen, and Qien Yu. Upst-nerf: Universal photorealistic style transfer of neural radiance fields for 3d scene. *IEEE Transactions on Visualization and Computer Graphics*, 31:2045–2057, 2025.

[DCS+17] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[EF01] Alexei A Efros and William T Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 341–346, 2001.

[EL99] Alexei A Efros and Thomas K Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the 7th IEEE International Conference on Computer Vision*, volume 2, pages 1033–1038. IEEE, 1999.

[FKYT+22] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5491–5500, 2022.

[FLFM18] Matthias Fey, Jan Eric Lenssen, Weichert Frank, and Heinrich Muller. SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 869–877, 2018.

[FSDH07] Matthew Fisher, Peter Schröder, Mathieu Desbrun, and Hugues Hoppe. Design of tangent vector fields. *ACM Transactions on Graphics*, 26(3):56–es, 2007.

[FXZ+24] Ben Fei, Jingyi Xu, Rui Zhang, Qingyuan Zhou, Weidong Yang, and Ying He. 3D gaussian splatting as new era: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 2024.

[Gar24] Manuel Garcia. The paradox of artificial creativity: Challenges and opportunities of generative ai artistry. *Creativity Research Journal*, pages 1–14, 05 2024.

[GCLY18] Shuyang Gu, Congliang Chen, Jing Liao, and Lu Yuan. Arbitrary style transfer with deep feature reshuffle. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8222–8231, 2018.

[GEB15a] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *Proceedings of the 29th International Conference on Neural Information Processing Systems*, volume 1, page 262–270, 2015.

[GEB15b] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Texture Synthesis Using Convolutional Neural Networks— Open Source Implementation on GitHub. `https://github.com/meet-minimalist/Texture-Synthesis-Using-Convolutional-Neural-Networks`, 2015.

[GEB16] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. *Journal of Vision*, 16(12), 2016.

[GRGH19a] Jorge Gutierrez, Julien Rabin, Bruno Galerne, and Thomas Hurtut. On Demand Solid Texture Synthesis Using Deep 3D Networks. *Computer Graphics Forum*, 39, 2019.

[GRGH19b] Jorge Gutierrez, Julien Rabin, Bruno Galerne, and Thomas Hurtut. On Demand Solid Texture Synthesis Using Deep 3D Networks—Open Source Implementation on GitHub. `https://github.com/JorgeGtz/SolidTextureNets`, 2019.

[GWHM24] Minghao Guo, Bohan Wang, Kaiming He, and Wojciech Matusik. Tetsphere splatting: Representing high-quality geometry with lagrangian volumetric meshes. *arXiv preprint arXiv:2405.20283*, 2024.

[GWRM25] Bruno Galerne, Jianling Wang, Lara Raad, and Jean-Michel Morel. Sgsst: Scaling gaussian splatting style transfer. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 26535–26544, 2025.

[HB95] David J Heeger and James R Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 229–238, 1995.

[HB17] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE international conference on computer vision*, pages 1510–1519, 2017.

[HHF+19] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. MeshCNN: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

[HHGCO20] Amir Hertz, Rana Hanocka, Raja Giryes, and Daniel Cohen-Or. Deep geometric texture synthesis. *ACM Transactions on Graphics (TOG)*, 39(4):108–1, 2020.

[HHY+22] Yi-Hua Huang, Yue He, Yu-Jie Yuan, Yu-Kun Lai, and Lin Gao. Stylizednerf: Consistent 3d scene stylization as stylized nerf via 2d-3d mutual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18321–18331, 2022.

[HJN22a] Lukas Höllein, Justin Johnson, and Matthias Nießner. Stylemesh: Style Transfer for Indoor 3D Scene Reconstructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6188–6198, 2022.

[HJN22b] Lukas Höllein, Justin Johnson, and Matthias Nießner. Stylemesh: Style Transfer for Indoor 3D Scene Reconstructions—Open Source Implementation on GitHub. `https://github.com/lukasHoel/stylemesh`, 2022.

[HJO+01] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, page 327–340. Association for Computing Machinery, 2001.

[HMR20] Philipp Henzler, Niloy J Mitra, and Tobias Ritschel. Learning a Neural 3D Texture Space from 2D Exemplars. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[HPP+18] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (ToG)*, 37:1–15, 2018.

[HRU+17] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[HTS+21] Hsin-Ping Huang, Hung-Yu Tseng, Saurabh Saini, Maneesh Singh, and Ming-Hsuan Yang. Learning to stylize novel views. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13849–13858, 2021.

[IZZE17] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[JAFF16] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *In Proceedings of the 14th European Conference on Computer Vision (ECCV), Part II*, pages 694–711. Springer, 2016.

[JBV17] Nikolay Jetchev, Urs Bergmann, and Roland Vollgraf. Texture synthesis with spatial generative adversarial networks. *arXiv preprint arXiv:1611.08207*, 2017.

[JDS10] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

[JMB+22] Ajay Jain, Ben Mildenhall, Jonathan T Barron, Pieter Abbeel, and Ben Poole. Zero-shot text-guided object generation with dream fields. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 867–876, 2022.

[Jul62] Bela Julesz. Visual pattern discrimination. *IRE Transactions on Information Theory*, 8(2):84–92, 1962.

[JYF+19] Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, Yizhou Yu, and Mingli Song. Neural style transfer: A review. *IEEE transactions on visualization and computer graphics*, 26(11):3365–3385, 2019.

[Ker23] Kerbl, Bernhard and Kopanas, Georgios and Leimkühler, Thomas and Drettakis, George. 3D Gaussian Splatting for Real-Time Radiance Field Rendering — Implementation. `https://github.com/graphdeco-inria/gaussian-splatting`, 2023.

[KFCO+07] Johannes Kopf, Chi-Wing Fu, Daniel Cohen-Or, Oliver Deussen, Dani Lischinski, and Tien-Tsin Wong. Solid Texture Synthesis from 2D Exemplars. In *ACM SIGGRAPH 2007*. ACM, 2007.

[KHR24a] Áron Samuel Kovács, Pedro Hermosilla, and Renata G. Raidou. $\mathscr{G}$-Style: Stylized Gaussian Splatting. *Computer Graphics Forum*, 43(7):e15259, 2024.

[KHR24b] Áron Samuel Kovács, Pedro Hermosilla, and Renata G. Raidou. Surface-aware Mesh Texture Synthesis with Pre-trained 2D CNNs. *Computer Graphics Forum*, 43(2):e15016, 2024.

[KKLD23] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics*, 42:1–14, 2023.

[KKP+22] Nicholas Kolkin, Michal Kucera, Sylvain Paris, Daniel Sykora, Eli Shechtman, and Greg Shakhnarovich. Neural neighbor style transfer. *arXiv preprint arXiv:2203.13215*, 2022.

[KLC07] Henry Kang, Seungyong Lee, and Charles K Chui. Coherent line drawing. In *Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, pages 43–50, 2007.

[KPZK17] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics*, 36:1–13, 2017.

[KSE+03] Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics (TOG)*, 22(3):277–286, 2003.

[KSS19] Nicholas Kolkin, Jason Salavon, and Gregory Shakhnarovich. Style transfer by relaxed optimal transport and self-similarity. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10043–10052, 2019.

[Kut22] Sarah Kuta. Art made with artificial intelligence wins at state fair. https://www.smithsonianmag.com/smart-news/artificial-intelligence-art-wins-colorado-state-fair-180980703/, 2022. Accessed on 24.2.2025.

[KXBP22] Nasir Mohammad Khalid, Tianhao Xie, Eugene Belilovsky, and Tiberiu Popa. CLIP-mesh: Generating textured meshes from text using pretrained image-text models. In *SIGGRAPH Asia 2022 Conference Papers*. ACM, nov 2022.

[LLX+01] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics (ToG)*, 20(3):127–150, 2001.

[LLY+25] Wenjie Liu, Zhongliang Liu, Xiaoyan Yang, Man Sha, and Yang Li. Abc-gs: Alignment-based controllable style transfer for 3d gaussian splatting. *arXiv preprint arXiv:2503.22218*, 2025.

[LLZ+19] Shiwei Li, Zixin Luo, Mingmin Zhen, Yao Yao, Tianwei Shen, Tian Fang, and Long Quan. Cross-atlas convolution for parameterization invariant learning on textured mesh surface. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6136–6145, 2019.

[LW16] Chuan Li and Michael Wand. Combining markov random fields and convolutional neural networks for image synthesis. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2479–2486, 2016.

[LWB+22] Boyi Li, Kilian Q Weinberger, Serge Belongie, Vladlen Koltun, and René Ranftl. Language-driven semantic segmentation. *arXiv preprint arXiv:2201.03546*, 2022.

[LYY+17] Jing Liao, Yuan Yao, Lu Yuan, Gang Hua, and Sing Bing Kang. Visual attribute transfer through deep image analogy. *ACM Transactions on Graphics*, 36, 2017.

[LZC+23a] Kunhao Liu, Fangneng Zhan, Yiwen Chen, Jiahui Zhang, Yingchen Yu, Abdulmotaleb El Saddik, Shijian Lu, and Eric Xing. Stylerf: Zero-shot 3D style transfer of neural radiance fields. *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 8338–8348, 2023.

[LZC+23b] Kunhao Liu, Fangneng Zhan, Yiwen Chen, Jiahui Zhang, Yingchen Yu, Abdulmotaleb El Saddik, Shijian Lu, and Eric Xing. Stylerf: Zero-shot 3D style transfer of neural radiance fields — Implementation. `https://github.com/Kunhao-Liu/StyleRF`, 2023.

[LZL+23] Rong Liu, Enyu Zhao, Zhiyuan Liu, Andrew Feng, and Scott John Easley. Instant photorealistic style transfer: A lightweight and adaptive approach. *arXiv preprint arXiv:2309.10011*, 2023.

[LZX+24] Kunhao Liu, Fangneng Zhan, Muyu Xu, Christian Theobalt, Ling Shao, and Shijian Lu. StyleGaussian: Instant 3D Style Transfer with Gaussian Splatting. *arXiv preprint arXiv:2403.07807*, 2024.

[Max95] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1:99 – 108, 1995.

[MBBV15] Jonathan Masci, Davide Boscaini, Michael M. Bronstein, and Pierre Vandergheynst. Geodesic Convolutional Neural Networks on Riemannian Manifolds. In *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, pages 832–840, 2015.

[MBM+17] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M. Bronstein. Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5425–5434, 2017.

[MBOL+22] Oscar Michel, Roi Bar-On, Richard Liu, Sagie Benaim, and Rana Hanocka. Text2mesh: Text-driven neural stylization for meshes, 2022.

[MESK22] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022.

[MKK21] Thomas W. Mitchel, Vladimir G. Kim, and Michael Kazhdan. Field Convolutions for Surface CNNs, 2021.

[MPSO18a] Alexander Mordvintsev, Nicola Pezzotti, Ludwig Schubert, and Chris Olah. Differentiable image parameterizations. *Distill*, 3(7):e12, 2018.

[MPSO18b] Alexander Mordvintsev, Nicola Pezzotti, Ludwig Schubert, and Chris Olah. Differentiable Image Parameterizations—Open Source Implementation on Distill. `https://distill.pub/2018/differentiable-parameterizations/`, 2018.

[MSOC+19] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 38:1–14, 2019.

[MST+20] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *European Conference on Computer Vision (ECCV)*, 2020.

[MWWL22] Fangzhou Mu, Jian Wang, Yicheng Wu, and Yin Li. 3D photo stylization: Learning to generate stylized novel views from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16252–16261, 2022.

[MWZ+23] Yiwei Ma, Haowei Wang, Xiaoqing Zhang, Guannan Jiang, Xiaoshuai Sun, Weilin Zhuang, Jiayi Ji, and Rongrong Ji. X-mesh: Towards fast and accurate text-driven 3D stylization via dynamic textual guidance, 2023.

[NPLX22] Thu Nguyen-Phuoc, Feng Liu, and Lei Xiao. Snerf: stylized neural implicit representations for 3D scenes. *SIGGRAPH*, 41:1–11, 2022.

[PABG20] Tiziano Portenier, Siavash Arjomand Bigdeli, and Orcun Goksel. Gramgan: Deep 3D texture synthesis from 2D exemplars, 2020.

[PCS+22] Nico Pietroni, Marcel Campen, Alla Sheffer, Gianmarco Cherchi, David Bommes, Xifeng Gao, Riccardo Scateni, Franck Ledoux, Jean Remacle, and Marco Livesu. Hex-mesh generation and processing: a survey. *ACM transactions on graphics*, 42(2):1–44, 2022.

[PLWZ19] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2337–2346, 2019.

[PNC+23] Gustav Grund Pihlgren, Konstantina Nikolaidou, Prakash Chandra Chhipa, Nosheen Abid, Rajkumar Saini, Fredrik Sandin, and Marcus Liwicki. A systematic performance analysis of deep perceptual loss networks: Breaking transfer learning conventions. *arXiv preprint arXiv:2302.04032*, 2023.

[PO18] Adrien Poulenard and Maks Ovsjanikov. Multi-directional geodesic neural networks via equivariant convolution. *ACM Transactions on Graphics*, 37:1–14, 2018.

[PS00] Javier Portilla and Eero P Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, 40:49–70, 2000.

[RBL+22] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[RBS+22] Max Reimann, Benito Buchheim, Amir Semmo, Jürgen Döllner, and Matthias Trapp. Controlling strokes in fast neural style transfer using content transforms. *The Visual Computer*, 38(12):4019–4033, 2022.

[RDN+22] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3, 2022.

[RDS+14] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115, 2014.

[RKH+21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, 2021.

[RMA+23] Elad Richardson, Gal Metzer, Yuval Alaluf, Raja Giryes, and Daniel Cohen-Or. Texture: Text-guided texturing of 3d shapes, 2023.

[SACO22] Nicholas Sharp, Souhaib Attaiki, Keenan Crane, and Maks Ovsjanikov. Diffusionnet: Discretization agnostic learning on surfaces. *ACM Transactions on Graphics (TOG)*, 41:1–16, 2022.

[SCL+22] Aditya Sanghi, Hang Chu, Joseph G. Lambourne, Ye Wang, Chin-Yi Cheng, Marco Fumero, and Kamal Rahimi Malekshan. Clip-forge: Towards zero-shot text-to-shape generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18603–18613, June 2022.

[SF95] Eero P Simoncelli and William T Freeman. The steerable pyramid: A flexible architecture for multi-scale derivative computation. In *In Proceedings of the International Conference on Image Processing*, volume 3, pages 444–447. IEEE, 1995.

[SGC+24] Abhishek Saroha, Mariia Gladkova, Cecilia Curreli, Tarun Yenamandra, and Daniel Cremers. Gaussian splatting in style. *arXiv preprint arXiv:2403.08498*, 2024.

[SVI+16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations (ICLR)*, 2014.

[TL22] Greg Turk and Marc Levoy. Stanford Bunny Texture. http://alice.loria.fr/index.php/software/7-data/37-unwrapped-meshes.html, 2022. Archived at https://web.archive.org/web/20220714184311/http://alice.loria.fr/index.php/software/7-data/37-unwrapped-meshes.html, accessed on 2025-03-21.

[Ull79] Shimon Ullman. The interpretation of structure from motion. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 203:405–26, 1979.

[VBV18] Nitika Verma, Edmond Boyer, and Jakob Verbeek. FeaStNet: Feature-Steered Graph Convolutions for 3D Shape Analysis. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2598–2606, 2018.

[WL00] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 479–488, 2000.

[WLKT09] Li-Yi Wei, Sylvain Lefebvre, Vivek Kwatra, and Greg Turk. State of the art in example-based texture synthesis. *In Eurographics 2009 State of the Art Reports (EG-STAR)*, pages 93–117, 2009.

[WLZ+18] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8798–8807, 2018.

[WM19] Chris Wyman and Adam Marrs. Introduction to directx raytracing. In *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs*, pages 21–47. Springer, 2019.

[WO22] Matthias Wright and Björn Ommer. Artfid: Quantitative evaluation of neural style transfer. In *DAGM German Conference on Pattern Recognition*, pages 560–576. Springer, 2022.

[WSZL19] Hao Wu, Zhengxing Sun, Yan Zhang, and Qian Li. Direction-aware neural style transfer with texture enhancement. *Neurocomputing*, 370:39–55, 2019.

[WYZ+24] Tong Wu, Yu-Jie Yuan, Ling-Xiao Zhang, Jie Yang, Yan-Pei Cao, Ling-Qi Yan, and Lin Gao. Recent advances in 3D gaussian splatting. *Computational Visual Media*, pages 1–30, 2024.

[WZC+21] Zhizhong Wang, Lei Zhao, Haibo Chen, Zhiwen Zuo, Ailin Li, Wei Xing, and Dongming Lu. Evaluate and improve the quality of neural style transfer. *Computer Vision and Image Understanding*, 207:103203, 2021.

[WZX23] Zhizhong Wang, Lei Zhao, and Wei Xing. Stylediffusion: Controllable disentangled style transfer via diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7643–7655, 2023.

[YHSG17] Li Yi, Xingwen Guo Hao Su, and Leonidas Guibas. SyncSpecCNN: Synchronized Spectral CNN for 3D Shape Segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6584–6592, 2017.

[YLP+20] Yuqi Yang, Shilin Liu, Hao Pan, Yang Liu, and Xin Tong. PFCNN: Convolutional Neural Networks on 3D Surfaces Using Parallel Frames. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13575–13584, 2020.

[You22] Jonathan Young. xatlas—Open Source Implementation on GitHub. `https://github.com/jpcy/xatlas`, 2022.

[YTB+20] Mao-Chuang Yeh, Shuai Tang, Anand Bhattad, Chuhang Zou, and David Forsyth. Improving style transfer with calibrated metrics. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3160–3168, 2020.

[ZCY+24] Dingxi Zhang, Zhuoxun Chen, Yu-Jie Yuan, Fang-Lue Zhang, Zhenliang He, Shiguang Shan, and Lin Gao. StylizedGS: Controllable Stylization for 3D Gaussian Splatting. *arXiv preprint arXiv:2404.05220*, 2024.

[ZFLS24] Deheng Zhang, Clara Fernandez-Labrador, and Christopher Schroers. CoARF: Controllable 3D Artistic Style Transfer for Radiance Fields. *Internation Conference on 3D Vision (3DV)*, pages 612–622, 2024.

[ZGW+22] Xin Zhao, Jifeng Guo, Lin Wang, Fanqi Li, Junteng Zheng, and Bo Yang. STS-GAN: Can We Synthesize Solid Texture with High Fidelity from Arbitrary Exemplars? *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 1768–1776, 2022.

[ZHT+23] Yuxin Zhang, Nisha Huang, Fan Tang, Haibin Huang, Chongyang Ma, Weiming Dong, and Changsheng Xu. Inversion-based style transfer with diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 215–224, 2023.

[ZHZ+25] Cailin Zhuang, Yaoqi Hu, Xuanyang Zhang, Wei Cheng, Jiacheng Bao, Shengqi Liu, Yiying Yang, Xianfang Zeng, Gang Yu, and Ming Li. Styleme3d: Stylization with disentangled priors by multiple encoders on 3d gaussians. *arXiv preprint arXiv:2504.15281*, 2025.

[ZKB+22a] Kai Zhang, Nick Kolkin, Sai Bi, Fujun Luan, Zexiang Xu, Eli Shechtman, and Noah Snavely. Arf: Artistic radiance fields. In *Computer Vision – ECCV 2022*, pages 717–733, Cham, 2022. Springer Nature Switzerland.

[ZKB+22b] Kai Zhang, Nick Kolkin, Sai Bi, Fujun Luan, Zexiang Xu, Eli Shechtman, and Noah Snavely. ARF: Artistic radiance fields — Implementation. `https://github.com/Kai-46/ARF-svox2`, 2022.

[ZT25] Tianshan Zhang and Hao Tang. Style transfer: A decade survey. *arXiv preprint arXiv:2506.19278*, 2025.

[ZTL20] Weijie Zhao, Shulong Tan, and Ping Li. Song: Approximate nearest neighbor search on gpu. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1033–1044. IEEE, 2020.