Self-Supervised Shape and Appearance Modeling via Neural Differentiable Graphics

Philipp Henzler

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Computer Science

University College London

Feb 16, 2023

I, Philipp Henzler, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Philipp Henzler

Abstract

Inferring 3D shape and appearance from natural images is a fundamental challenge in computer vision. Despite recent progress using deep learning methods, a key limitation is the availability of annotated training data, as acquisition is often very challenging and expensive, especially at a large scale. This thesis proposes to incorporate physical priors into neural networks that allow for self-supervised learning. As a result, easy-to-access unlabeled data can be used for model training. In particular, novel algorithms in the context of 3D reconstruction and texture/material synthesis are introduced, where only image data is available as supervisory signal.

First, a method that learns to reason about 3D shape and appearance solely from unstructured 2D images, achieved via differentiable rendering in an adversarial fashion, is proposed. As shown next, learning from videos significantly improves 3D reconstruction quality. To this end, a novel ray-conditioned warp embedding is proposed that aggregates pixel-wise features from multiple source images.

Addressing the challenging task of disentangling shape and appearance, first a method that enables 3D texture synthesis independent of shape or resolution is presented. For this purpose, 3D noise fields of different scales are transformed into stationary textures. The method is able to produce 3D textures, despite only requiring 2D textures for training. Lastly, the surface characteristics of textures under different illumination conditions are modeled in the form of material parameters. Therefore, a self-supervised approach is proposed that has no access to material parameters but only flash images. Similar to the previous method, random noise fields are reshaped to material parameters, which are conditioned to replicate the visual appearance of the input under matching light.

Acknowledgements

The work presented in this thesis would not have been possible without the support of the many wonderful people who helped me along the way. First and foremost, I am extremely grateful to my supervisor, Tobias Ritschel. He provided me with the best possible guidance I could have hoped for. His vision, creativity and passion for research have helped me to become the researcher that I am today.

I would also like to extend a special thank you to Niloy J. Mitra for constantly sharing invaluable research insights and demonstrating endless enthusiasm and dedication. I am grateful for all the time he dedicated to salvaging seemingly lost projects and the plethora of great ideas he provided for this thesis.

A big thank you goes out to Timo Ropinski. His advice and guidance have been pivotal in my early research career. I would not have written these words without Timo, as he was the one who introduced me to and sparked my passion for computer vision and graphics.

David Novotny also deserves my deepest gratitude for being a role model with a relentless work ethic, mentality to never give up and for fixing many of my bugs during the deadline crunch. Chapter 4 would have never made it into this thesis without him and the incredible help from Andrea Vedaldi and Roman Shapovalov.

A special thanks to Valentin Deschaintre. His exceptional material expertise and dedication were instrumental in saving Chapter 6 and bringing it to fruition.

I express my sincere gratitude to everyone at UCL. I would like to extend a special thanks to the people across the vision and graphics groups at UCL for the many lunches, reading group meetings, and fruitful discussions we have had over the years. I am especially grateful to Mohamed, Preddy, David and Stephan, who have stood

by me through all the ups and downs.

I cannot express how much I owe to my amazing flatmates and friends in London. Lucas, Elise, Vincenzo, Oli, Felix, and Matthieu, thank you for being there for me through thick and thin, cheering me up during stressful times and keeping me grounded. Our dinners, sports activities, and fun conversations were the ideal ways to unwind and take a break from the PhD stress.

Finally, I am incredibly thankful to my parents, siblings, and of course, my dear Ramessa for your constant support, love, and encouragement throughout my entire academic journey. Your sacrifices, patience, and understanding mean the world to me. Thank you for being my rock, cheerleaders, and best friends.

Impact Statement

In this thesis, four novel approaches for self-supervised shape and appearance modeling via neural differentiable graphics are proposed. All contributions are published at CVPR, ICCV or SIGGRAPH Asia, the premier venues for scholarly work in computer vision and graphics. To encourage further academic work, code, datasets and additional results for all methods have been released.

Philipp Henzler, Niloy Mitra, and Tobias Ritschel. Escaping plato's cave using adversarial training: 3d shape from unstructured 2d image collections. In *Proc. ICCV*, 2019, Chapter 3

Philipp Henzler, Jeremy Reizenstein, Patrick Labatut, Roman Shapovalov, Tobias Ritschel, Andrea Vedaldi, and David Novotny. Unsupervised learning of 3d object categories from videos in the wild. In *Proc. CVPR*, 2021, Chapter 4

Philipp Henzler, Niloy J Mitra, and Tobias Ritschel. Learning a neural 3d texture space from 2d exemplars. In *Proc. CVPR*, 2020, Chapter 5

Philipp Henzler, Valentin Deschaintre, Niloy J Mitra, and Tobias Ritschel. Generative modelling of brdf textures from flash images. *ACM Trans Graph (Proc. SIGGRAPH Asia)*, 40(6):195–206, 2021, Chapter 6

Contents

Chapter 1

Introduction

We, humans, have the ability to perceive our 3D world from 2D observations, captured by our eyes and processed by the human visual system at a remarkable ease. In our daily lives, we constantly interact with 3D objects and reason about their shape and appearance intuitively. Teaching machines to reason about our 3D world from visual observations such as images, similar to humans, is a common goal in scientific fields ranging from computer vision to computer graphics to robotics. Extracting 3D information from 2D images is a highly underconstrained and ill-posed problem. An image is a projection of our 3D world onto a 2D plane from a given camera pose which can be explained by an infinite amount of 3D variations. The interplay of geometry, material and illumination influences the incoming amount of radiance for each 2D location. Geometry determines the 2D location in the image plane where any 3D point along a ray is mapped onto. The material characteristics describe the surface of the geometry and its interaction with illumination. Thus, extracting meaningful 3D information from a given image requires knowledge of camera pose, geometry, material and illumination, which are referred to as *scene parameters* in this thesis, see Fig. 1.1.

Machines perform tasks more consistently than humans by their very nature, however, have failed to surpass human-level performance up until the deep learning era.

Early works on image classification [5], semantic segmentation [6, 7] or object detection [8, 9] have demonstrated impressive results while relying on large amounts of annotated 2D data [10] for supervised learning.

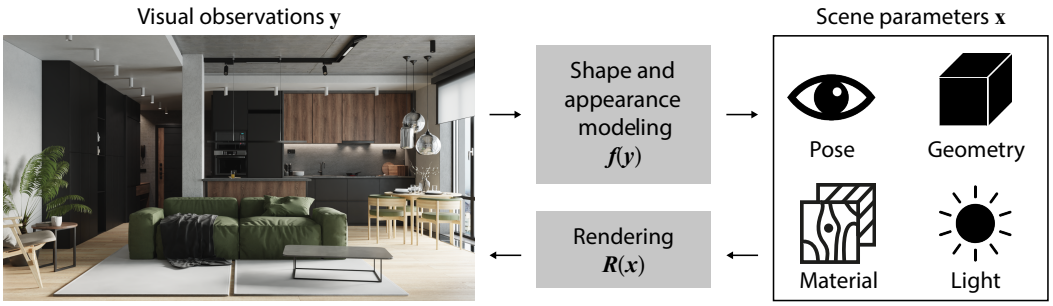


Figure 1.1: Shape and appearance modeling is the task of inferring meaningful 3D information from visual observations. The inverse process is physically modeled by rendering, i. e., mapping higher-dimensional data in the form of scene parameters $\mathbf{x}$ to lower-dimensional data such as natural images $\mathbf{y}$.

Efforts have been made to acquire datasets providing 3D information, e. g., synthetic datasets [11, 12] or 3D scans [13, 14], which allowed to train algorithms that reason about 3D in a supervised manner. However, their performance is bound by the limitations of current annotated 3D datasets, which lack size, realism and diversity. Furthermore, only limited supervision is provided. This leads to restricted generalization and expressiveness of supervised algorithms. Currently, no real-world dataset exists that provides full information about scene parameters at a large scale. As a result, overcoming the need for *supervised learning* is crucial, as large annotated 3D datasets may never exist. Images or videos, on the other hand, are easily accessible as vast amounts already are available on the internet and can also be easily captured at a low cost.

> *"Will he not fancy that the shadows which he formerly saw are truer than the objects which are now shown to him?"* — PLATO;

Inspired by Plato, it seems desirable to learn from visual observations only. Doing so would consequently reduce the required amount of supervision significantly.

1.1 Goal

This thesis addresses the challenging task of shape and appearance modeling where only images as supervisory signals are available. Given a visual observation $\mathbf{y}$, the goal is to predict the underlying 3D scene parameters $\mathbf{x}$ without having access to 3D information, i. e., no supervised training is possible. This is achieved by incorporating

physical priors in the method pipeline that map the 3D scene parameters back to the original input domain. Even though this physical process, which we know as rendering [15] is well understood and capable of producing photorealistic results as demonstrated in Hollywood films [16, 17], it is not directly applicable to predict scene parameters. The reason is that this task is highly ambiguous, i. e., many different combinations of pose, geometry, material and lighting can map to the same 2D image.

Currently, simplifying assumptions are made to constrain those ambiguities in order to make the task feasible, e. g., camera poses are known, access to template shapes is given, or key points are required. Unfortunately, these assumptions do not only simplify the task but also prevent the underlying algorithm from fully solving the problem, as part of it is already solved, i. e., provided in terms of supervision. The goal of this thesis is to limit the amount of supervision as much as possible and therefore make the task at hand harder in order to increase the ability of the method to learn a full 3D understanding.

Furthermore, neural networks follow the principle of Occam's razor, i. e., they converge to the most simple explanation that satisfies the objective function. This means that ambiguous solutions cannot be resolved by carefully designed objective functions. In this thesis, we will see that explicitly constraining neural networks to adhere to known mathematical concepts helps to reduce aforementioned ambiguities and in some cases can even turn these to an advantage. Several contributions that share the same underlying principle are discussed next.

1.2 Contributions

In this thesis, neural network architectures that incorporate physical priors as intermediate differentiable building blocks are introduced. Such architectures can be trained end-to-end in a self-supervised fashion which allows tapping into easy-to-access data such as image collections or videos. Four contributions addressing different tasks in the space of shape and appearance modeling are proposed and divided into two parts. The first part discusses single-image 3D reconstruction challenges in Chapter

3 and Chapter 4, where only images or videos are available as supervision. In the second part, approaches that focus on texture and material synthesis are explored. A 3D texture synthesis method that lifts 2D textures to solid 3D textures without explicit supervision is presented in Chapter 5. In Chapter 6, the acquired knowledge from the previous chapter is used to additionally explain the surface characteristics of textures in order to synthesize material maps from flash images.

1.2.1 3D Reconstruction

In Chapter 3 PLATONICGAN is introduced, which aims to discover the 3D structure and appearance of a specific object class from an unstructured collection of 2D images, i. e., where no relation between photos is known. The key idea is to train a deep neural network to generate 3D shapes which, when rendered to images, are indistinguishable from ground truth images (for a discriminator) under various camera poses. To establish constraints between 2D image observation and their 3D interpretation, a family of rendering layers that are effectively differentiable is suggested, which enables a self-supervised design (see Fig. 1.2). Discriminating 2D images instead of 3D shapes allows tapping into unstructured 2D photo collections instead of relying on curated (e.g., aligned, annotated, etc.) 3D datasets or assuming the availability of 2D primitives such as key points. At test time, this method is capable of reconstructing shape and appearance from a single image.

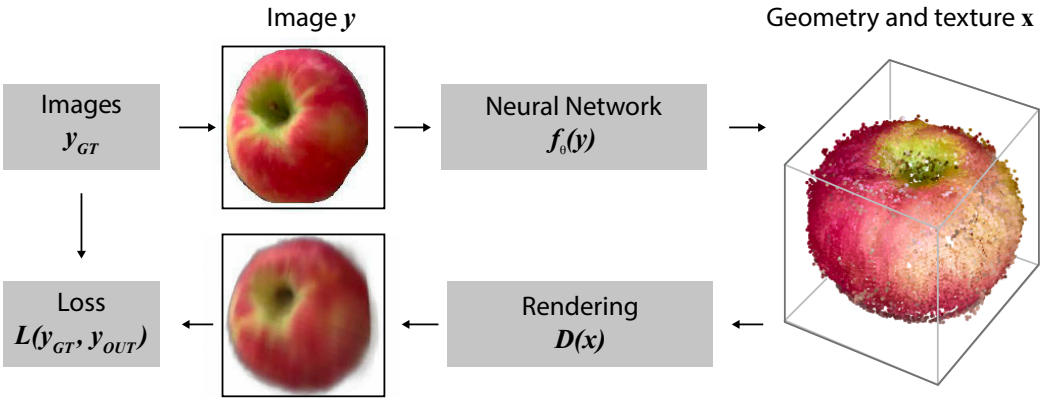


Figure 1.2: Given an image **y** of an object, a neural network **f** is trained to extract the 3D geometry and texture **x** without having access to 3D data. This is achieved by incorporating a differentiable rendering module that maps higher-dimensional 3D data back to the original 2D domain, enabling self-supervised training.

The goal in Chapter 4 is to train a deep neural network that, given a small number of images of an object of a given category, will reconstruct its 3D shape and appearance. Similar to the previous chapter, challenging real data with no manual annotations is used for training. Instead of relying on unstructured image collections, a new large dataset of object-centric videos suitable for multi-view training is introduced. Exploiting multi-view data, a novel neural network design, called warp-conditioned ray embedding (WCE), is proposed. It allows us to aggregate information from an arbitrary number of views to aid 3D reconstruction and again makes use of differentiable rendering which enables self-supervised training, as relative camera poses can be extracted from multi-view data using Structure-from-Motion. In comparison with the single-view method proposed in Chapter 3, this method significantly improves reconstruction quality.

1.2.2 Texture and material synthesis

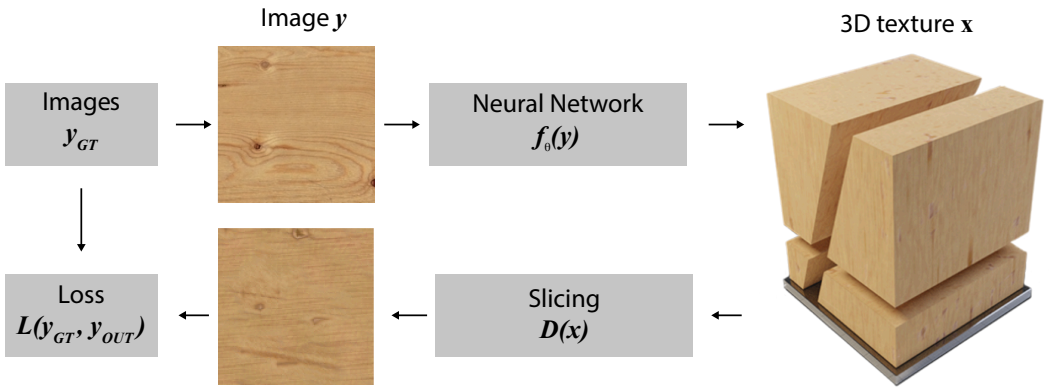


Figure 1.3: Given an image **y** of an object, a neural network **f** is trained to extract the 3D geometry and texture **x** without having access to 3D data. This is achieved by incorporating a differentiable rendering module that maps higher-dimensional 3D data back to the original 2D domain, enabling self-supervised training.

A generative model of 3D natural textures with diversity, visual fidelity and high computational efficiency is proposed in Chapter 5. For a given texture exemplar in 2D, the goal is to produce a 3D texture with the same visual characteristics. The main challenge is that no access to real-world 3D textures is possible whereas 2D textures can be casually captured using mobile phones. Solving such a task seems impossible, however, under the assumption of stationarity, i. e., image statistics are the same for

every part of the image, it becomes feasible. The idea is to use a hard-coded slicing operation that extracts 2D slices from 3D textures, see Fig. 1.3. This way, a statistical comparison can be performed in 2D rather than 3D which allows us to produce 3D textures without 3D supervision in a self-supervised fashion. In order to generate high-fidelity textures, random 3D noise fields of different frequencies are mapped to 3D textures using a neural network.

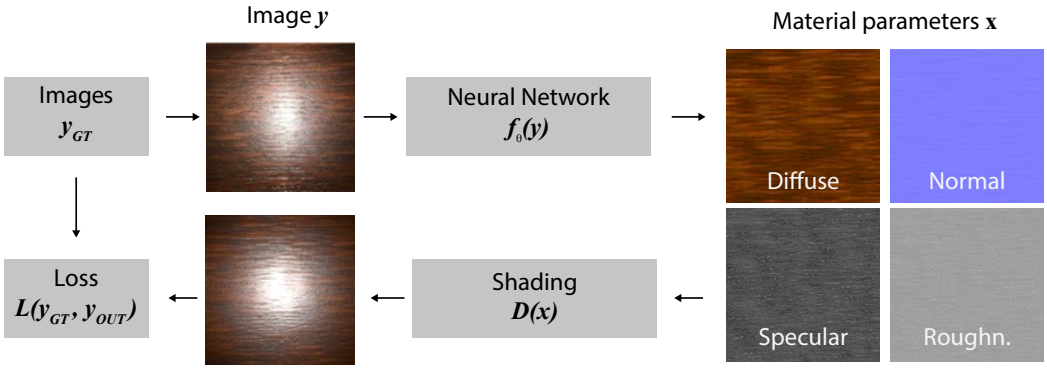


Figure 1.4: Given a natural texture **y**, e. g., wood, a neural network **f** is trained to produce a 3D texture with the same statistics **x** where no access to 3D textures is available. This is achieved by incorporating a slicing module that maps higher-dimensional 3D textures back to the original 2D domain, enabling self-supervised training.

In Chapter 6, which builds upon the previous approach, additionally models the surface characteristics of textures under different illuminations by synthesizing materials (e. g., leather, wood, etc.) from flash images. When users provide a photo of a natural material captured under flash light illumination, the presented method produces an infinite and diverse spatial field of material parameters. This is achieved without having access to material parameters at all. As presented in Fig. 1.4, the produced material parameters are mapped back to the flash image domain through a shading operation that is differentiable. This enables the use of a loss function in a lower-dimensional domain.

1.2.3 Self-supervised learning via differentiable physical priors

Overall, a family of methods that follow the same underlying self-supervised design, illustrated in Fig. 1.5, are proposed. Each method is based on an easy-to-acquire dataset which can be captured using a smartphone and does not require any data annotation. Either unstructured image or video collections of category-specific objects,

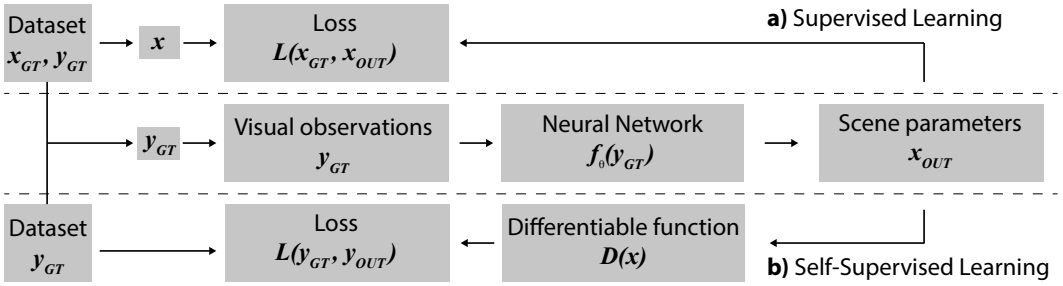


Figure 1.5: Shape and appearance modeling is the task of inferring meaningful 3D information from visual observations. The inverse task is rendering, i. e., mapping higher-dimensional data, in the form of scene parameters, to lower-dimensional data such as images.

texture collections or flash images of materials are used. During training, individual visual observations $\mathbf{y}_{GT}$ from the dataset are sampled. Note that a supervised approach would additionally provide the corresponding scene parameter information $\mathbf{x}_{GT}$. This two-dimensional input data is then lifted to higher dimensions $\mathbf{x}_{OUT}$, such as 3D geometry, 3D textures or material parameters, through a non-linear function in the form of a neural network $\mathbf{x_{OUT}} = \mathbf{f}(\mathbf{y}_{GT})$. This is achieved by using Convolutional Neural Networks (CNNs) or Multilayer Perceptrons (MLPs) depending on the desired representation. In a second step, differentiable functions $\mathbf{D}$, which map the higher-dimensional data back to the original input domain $\mathbf{x}_{OUT} = \mathbf{D}(\mathbf{f}(\mathbf{y}_{GT}))$, are carefully incorporated as discussed above. This enforces the representation to be explicit, i. e., in a human-readable format rather than a neural representation that cannot be interpreted. Finally, the output of the differentiable function is compared to the original input image through an objective function. This stands in contrast to supervised methods that perform comparison in the higher-dimensional space, which requires hard-to-acquire annotated data.

Chapter 2

Background and Previous Work

This thesis addresses tasks at the intersection of computer vision, graphics and deep learning. More specifically, the focus lies on modeling shape and appearance from images only, using inverse differentiable graphics. This requires knowledge in several distinct areas which will be discussed in this section. Firstly, core deep learning techniques are introduced before diving into how synthetic scenes are represented and visualized (rendered) in computer graphics. Further, an overview of related work in the context of 3D reconstruction algorithm is provided, specifically focusing on the importance of deep learning for inverse rendering. Secondly, relevant concepts for texture and material synthesis are addressed. Therefore, classical methods as well as modern deep learning methods are presented before focusing on material capture from flash images.

2.1 Deep Learning

Deep learning (DL) is a type of machine learning (ML) and artificial intelligence (AI) that teaches neural networks to solve complex problems by learning from large amounts of data.

A common deep neural network consists of an input and output layer with multiple hidden layers in between. Each layer is represented by a mathematical function that extracts features from incoming data, passes them through non-linearities and uses the output as the input to the next layer. Overall the goal is to approximate any desired function which is parameterized by learnable network weights. To achieve

this, the network weights are optimized such that the network best maps the input data to the desired output. This process is referred to as training a neural network and is outlined below.

First, the input data is processed by the neural network to make a prediction (forward pass). An objective function then determines the error (loss) between the predicted output and the target output (ground truth). Next, the network weights are updated following the backpropagation algorithm introduced by Rumelhart et al. [18]. Hence, the gradient of the loss with respect to the weights, using the chain rule, is calculated backwards starting from the last layer to the first (backward pass). Finally, each weight is updated according to the direction of the gradient. This process is repeated until convergence. In practice, many different optimization algorithms can be used, e. g., Stochastic Gradient Descent (SGD) [19] or Adam [20].

2.1.1 Multilayer perceptron

One of the earliest forms of AI, the perceptron, dates back to 1957 when Rosenblatt [21] introduced a binary classification model. A perceptron takes as input a vector $\mathbf{x} = (x_1, ..., x_n)$, multiplies it with a weight $\theta = (\theta_1, ..., \theta_n)$, adds a bias $\mathbf{b}$ and returns the weighted sum before passing the value through an activation function σ.

$$\sigma(\sum_{1}^{n} \theta_i \times x_i + \mathbf{b}) \tag{2.1}$$

A single perceptron can only learn linearly separable patterns. For instance, it cannot learn to approximate an XOR function as no single straight line exists that can separate the input vectors with respect to their output value. However, when stacking up multiple perceptrons as is the case for MLPs, any continuous non-linear function can be approximated [22]. In practice, the Rectified Linear Unit (ReLU) [23] activation function is widely used. However, any non-linear function can be used as an activation function.

There are two major drawbacks of MLPs with regard to image processing. Firstly, each perceptron is connected to every other perceptron, which exponentially increases the number of learnable network weights with image resolution. This leads to

memory problems in higher resolutions. Secondly, MLPs are not translation invariant, resulting in redundant operations for image processing.

2.1.2 Convolutional Neural Networks

A different type of architecture, that addresses these drawbacks, are Convolutional Neural Networks (CNNs). These were first introduced by LeCun et al. [24], before AlexNet [5] marked a major breakthrough by winning the ImageNet [10] competition in 2012. An important component in that success was the use of convolutional layers as those address the main drawbacks of MLPs. Convolutional layers slide local trainable filters, defined by kernel size and stride, over input images in order to extract local features. In comparison with MLPs, each filter has access to a spatial local neighbourhood instead of processing the entire image at once. This has several advantages. The same filter is applied across spatial regions of the image, which makes CNNs translation invariant. Moreover, higher resolution inputs can be processed, as the number of filter elements is independent of the image size.
Another important concept, which cannot be exploited by MLPs, are pooling layers, which reduce the dimensions of feature maps. To this end, a region, defined by width and height, is compressed to a single value through a *pooling* operation. The two most common operations are *max pooling* and *average pooling*.
Stacking many convolutional layers followed by non-linearities and pooling operations allows the extraction of increasingly complex features [25]. The first few layers of a CNN learn lower-level features such as edges and circles. Deeper layers operate on lower-level features and extract complex textures and patterns. Eventually, objects or parts of objects can be represented at even deeper levels.

2.1.3 Generative Adversarial Networks

Generative adversarial networks were introduced by Goodfellow et al. [27] and extend the aforementioned single-network approaches by introducing an additional critic network. Two networks, a generator and a discriminator compete with each in order to produce new synthetic instances of a given data distribution. The generator aims to produce images that fool the discriminator, which, in turn, tries to detect

Figure 2.1: Synthetic images generated by a state-of-the-art generative adversarial network. [26]

whether a provided image is real or fake. The goal is to play a min-max game where both networks try to outperform each other. If training is successful both networks learn from each other, become better and eventually the generator is able to produce new synthetic images that cannot be distinguished from the real data, even for humans as depicted in Fig. 2.1. In Chapter 3, PlatonicGAN exploits adversarial training in order to produce 3D shape and appearance that when rendered to synthetic 2D images are not distinguishable from real images.

2.1.4 Neural Style Transfer

Another important concept that this Chapter 5 and Chapter 6 are inspired by is Style transfer [28]. It is an optimization technique that, given a content and a style image, optimizes an output image that finds the minimum distance between *content* and *style*. In other words, the aim is to transfer the *style* from the style image to the content image as illustrated in Fig. 2.2. This is achieved by a two-fold loss term, one that minimizes the *content* and another that minimizes the *style*. Style transfer leverages the fact that CNNs provide low-level pixel information in higher layers

Figure 2.2: Given a *content* image (top-left) and a *style* image (thumbnail images), the method proposed by Gatys et al. [28] produces new images combining *content* and *style*.

while deeper layers represent more global features [29]. Features of intermediate layers l are defined as $F^l \in \mathbb{R}^{C_l \times N_l}$ where C are the number of filters and N is the spatial size. The content loss is then defined as the squared error between features of the content image F and the target image $\hat{F}$.

$$L_{content} = \sum_l \sum_i (F_i^l - \hat{F}_i^l)^2 \tag{2.2}$$

The *style* is represented as the correlation between learned feature maps, in particular, their gram matrices. The gram matrix is calculated as dot product between flattened features:

$$G = F^T \cdot F \tag{2.3}$$

The final style loss is measured by comparing the squared error between gram matrices:

$$L_{style} = \sum_{i,j} (G^l_{i,j} - \hat{G}^l_{i,j})^2 \tag{2.4}$$

The gram matrix serves as a statistical texture representation for stationary textures (see Sec. 2.5). Other statistical measures such as mean and variance could be used as well, however, empirically do not perform as well. Style transfer commonly uses a pre-trained VGG network [29], as also used in this thesis, for feature extraction.

2.2 Basics of Computer Graphics

Fundamental to computer graphics is the interplay of geometry, material and lighting as briefly discussed in Chapter 1. Each of these components influences the process of generating a synthetic image, formally known as rendering. In the following section, the rendering equation along with each component will be discussed in greater detail.

2.2.1 Geometry

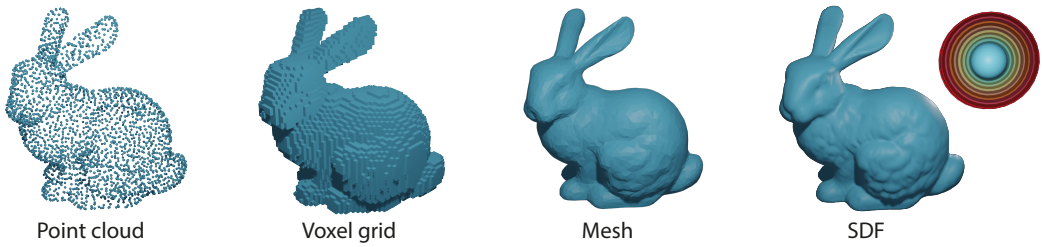


Figure 2.3: Visualisation of different 3D shape representations.

Geometry can be represented in many different ways, as illustrated in Fig. 2.3. Polygon meshes are the most common representation for rendering-type applications such as computer games or movies and have been an established industry standard for the past decades. A mesh is defined as a collection of vertices, edges and faces (polygons) that describe the surface of an object with a fixed topology. However, for reconstruction tasks meshes are not the preferred choice as they rely on initial template shapes that are deformed to match a desired output shape. Due to the fixed topology, it is not possible to deform any template shape to a specific target shape, e. g., chairs with a different number of legs require different template shapes.
A more flexible way of representing arbitrary shapes are voxel grids and point clouds.

A point cloud, similarly to a mesh, consists of a set of vertices where each point is specified by a 3D location and can further contain attributes such as colour, normal, etc. Point clouds are usually produced by 3D scanners or retrieved from multi-view imagery (see Sec. 2.3). Unlike meshes, neither any connectivity between points exists nor any surface information is provided, which causes several drawbacks. Firstly, no trivial local neighbourhood relations can be established which is crucial for feature extraction via neural networks as explained above. Secondly, capturing accurate surface texture information is nearly impossible, especially for sparse point clouds.

Voxel grids on the other hand are a discretized version of point clouds that subdivide a bounded 3D space into a regular grid of 3D cells (similar to 2D pixels), which are referred to as voxels. The geometry of a voxel grid is represented as density values where 0 equals empty space and 1 means fully occupied. Due to their regular form, accessing spatial neighbourhoods is straightforward and therefore convolutional layers can directly be applied, which makes them very attractive for Deep Learning. However, voxel grids are usually restricted to lower resolutions as small increases in resolution cubically increase memory consumption.

Growing in popularity, neural 3D shape representations are defined as continuous functions parameterized by neural networks. They are either represented as signed distance functions (SDFs) or occupancy functions. An SDF determines the distance to the surface at a given point. Concretely, the surface is represented as a zero-level set of a neural network: $\mathscr{S} = \{\mathbf{x} \in \mathbb{R}^3 \mid f(\mathbf{x};\theta) = 0\}$. The function returns negative values if the given point lies outside the surface and positive values if the point lies inside. On the other hand, an occupancy function returns a density value $\mathbf{o} \in \mathbb{R}^1$ for each location $\mathbf{x} \in \mathbb{R}^3$ similarly to voxel grids. In this thesis, voxel grids as well as neural representations in the form of occupancy functions are used as these provide the most flexibility.

2.2.2 Lighting

The second component which is inevitable for scene understanding is lighting [30]. In computer graphics, light is represented in the form of light rays that are comprised

of an origin and direction. During simulation, a light ray is cast into a scene where it interacts with geometry. Incoming light at a surface is absorbed and/or scattered. Physically correct models require to trace light rays bouncing off geometry multiple times. This process is computationally very expensive and will be described in more detail in Sec. 2.2.4. How light interacts with geometry requires knowledge of surface characteristics, as explained next.

2.2.3 Material

Material properties define how light is reflected or refracted on the surface of geometry. For this thesis, it is sufficient to only model the reflective property in the form of a bidirectional reflectance distribution function (BRDF) [31]. Given an incoming light direction ω_i and outgoing light direction ω_o at a surface point $\mathbf{x}$, the function calculates the amount of reflected energy. Different material models have been proposed over the years that aim to approximate this function. Traditionally, sophisticated hardware was used to capture the complex behaviour of materials by densely sampling over lighting and viewing directions [32]. Such hardware is very expensive and requires accurate calibration to achieve good results. An example of such hardware is shown in Fig. 2.4.

For each light/view combination the material properties have to be stored in a lookup table and when querying new light/view directions interpolation is required. Storing each light-view combination for each different material is not feasible in practice and therefore simplified mathematical models have been proposed that approximate material parameters in a more compact manner.

Originally, a non-physical model was introduced by Phong [33]. This model is very simple and parameterized by three parameters: diffuse albedo, specular albedo and glossiness. The diffuse and specular albedos describe the reflected colours of materials. The diffuse part reflects the colour evenly in all directions (lambertian reflection) whereas the specular part reflects the light in a single direction, assuming a perfectly smooth surface. The glossiness parameter controls how rough or smooth a surface behaves. A very high glossiness value produces mirror-like behaviours whereas lower values simulate scattering of light into different directions.

Figure 2.4: A hardware design for material acquisition. [32]

Given the incoming light direction ω_i and the surface normal **n** the diffuse part is calculated by taking the dot product between them. The specular part additionally requires the outgoing viewing direction ω_o and calculates the dot product between ω_o and the reflected direction **r** (negative values are clamped to 0) before raising it to the power of α, which represents glossiness. An illustration is provided in Fig. 2.5.

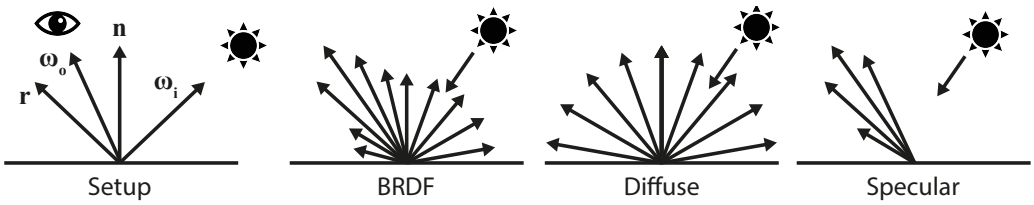


Figure 2.5: Illustration of the parameters that define a BRDF.

A few years later this model was slightly improved by Blinn [34], however, yet not physically accurate. In order to adhere to physical laws, the material model needs to be reciprocal and energy conserving which neither of the proposed methods is. Helmholtz-reciprocity [35] states that incoming and outgoing light directions must

be interchangeable without changing the output of the material model. The energy conservation requirement is met if the total amount of energy of outgoing reflected light is equal (or less) to the energy of incoming light.

Physically realistic models that comply with both conditions are commonly based on microfacet models [36]. The microfacet theory at a macro level assumes a flat surface which at the micro level is comprised of different tiny microfacets with varying surface normals. The aim is to simulate how incoming light at the geometric surface is reflected and scattered given the different orientations of microfacets. Simple material models such as Phong do not model those variations, but microfacets models approximate them in the form of a normal (microfacet) distribution function D. Further, self-shadowing and self-masking effects among microfacets are modeled via a geometric term G.

In this thesis, the Cook-Torrance microfacet BRDF model [37] is used in Chapter 6. It is parameterized by three parameters: diffuse albedo, specular albedo and roughness. While the computation of the diffuse part is mostly identical to Phong, the specular part is based on the microfacet theory and requires to compute D and G as well as a Fresnel term F. The Fresnel term returns the reflectance ratio on the surface for different incident angles. While for each of these parts numerous alternatives exist, the GGX [38] microfacet distribution, a revised version of Smith's geometric function [39] and Schlick's Fresnel term [40] are used in this thesis.

2.2.4 Rendering equation

Calculating the final reflectance value per pixel for a rendered image is achieved by solving the rendering equation [41].

$$L_o(\mathbf{x},\omega_o,\lambda,t) = \underbrace{L_e(\mathbf{x},\omega_o,\lambda,t)}_{\text{light source}} + \int_\Omega \underbrace{f_r(\mathbf{x},\omega_o,\omega_i,\lambda,t)}_{\text{BRDF}} \underbrace{L_i(\mathbf{x},\omega_i,\lambda,t)}_{\text{incoming light}} \underbrace{(\omega_i \cdot \mathbf{n})}_{\text{attenuation}}\, d\omega_i \tag{2.5}$$

The rendering equation describes the total amount of reflected radiance at a point in space $\mathbf{x}$, representing the geometry, along a particular viewing direction ω_o, wavelength λ and time t. For static scenes, which are the focus in this thesis, t can

be neglected and assumed to be constant. λ describes the perceived colour and is usually discretized to the RGB (red, green and blue) colour spectrum. L_e is the emitted radiance at a given point and direction and is only relevant if there exists a light source at this location. The remaining part of the equation is an integral over all incoming light directions Ω. It is defined by calculating the BRDF f_r, representing material properties as explained in Sec. 2.2.3. L_i is a function returning the incoming light at a given point $\mathbf{x}$ and incoming direction ω_i. The last part is the attenuation factor decreasing the radiance based on the incident angle between incoming direction ω_i and surface normal $\mathbf{n}$ defined by geometry. It becomes clear that manipulating either geometry, material or lighting will change the final rendered image. However, solving global illumination with such a recursive equation is very computationally expensive and cannot be solved in real-time yet. Usually, it is approximated using Monte Carlo methods [42] or other simplifying assumptions are made such as pre-computing environment maps [43]. In this thesis, simplifying assumptions are made with a focus on BRDF estimation in Chapter 6 and volume rendering [44] in Chapter 3 and Chapter 4, instead of solving the entire rendering equation. Volume rendering is the process of casting rays through a volumetric density field while simulating the absorption of light along the ray [44]. Different image formation models exist which will be explained in detail in Sec. 3.3.

2.3 View Synthesis

Reconstructing shape and appearance from multiple images has a long history in computer vision. Over the years, various methods such as Structure-from-Motion [45, 46], Multi-View-Stereo [47, 48, 49, 50], Image-Based-Rendering (IBR) [51, 52, 53] and the more recent Neural Radiance Fields (NeRF) [54] approach have emerged.

Under the assumption that input images are unstructured, most Multi-View-Stereo, Image-Based-Rendering and NeRF methods are built on Structure-from-Motion to extract relative camera poses.

2.3.1 Structure-from-Motion

Structure-from-Motion (SfM) consists of two main stages. The first stage finds feature correspondences between images. This requires to detect unique features such as corners and edges before finding point correspondences between pairs of images. A widely used approach for feature extraction is Scale-invariant feature transform (SIFT) [55] which is invariant to scale, rotation and changes in illumination.
The second stage in SfM uses these correspondences for joint 3D reconstruction and camera pose estimation. The relation between two images of the same scene from different camera views imposes geometric constraints (epipolar geometry) which along with known point correspondences allow to infer relative camera poses between two images. Once the relative poses are known, the corresponding 3D location of any correspondence pair can then be estimated using triangulation. However, not all computed point correspondences will be correct due to noisy data. To remove outliers a well-established method called Random Sampling Consensus or RANSAC in short is commonly used [56]. While estimated relative camera poses might be accurate for individual image pairs, it does not necessarily mean that the estimates are consistent for all combinations. Especially, since the same features might exist across multiple images. To this end, all camera parameters and 3D points are optimized simultaneously to minimize reprojection errors in all images. This step is referred to as bundle adjustment [57].

2.3.2 Multi-view-Stereo

Even though SfM is able to extract relative camera poses and a sparse point cloud, it is desired to obtain dense reconstructions, i. e., to obtain depth information for each pixel and therefore maximum utilization of the available input. Multi-View-Stereo (MVS) complements SfM towards this goal. It requires camera parameters as input which can be provided by SfM and unlike SfM returns a dense 3D reconstruction. This is achieved by finding dense correspondences between images. More specifically, the key is to determine the depth for each pixel such that when re-projecting it into another view the appearance matches. To do so, a metric is

required that measures the photo-consistency, i. e., the probability of a pixel being the potential match. Commonly, local pixel neighbourhoods rather than single pixels are compared as this allows for a more robust and invariant measure under different illuminations and noisy cameras. For instance, the Sum-of Squared-Differences (SSD) or Normalized Cross-Correlation (NCC) could be used as a measure. Finding photo-consistent correspondences has its limitations. For example, texture-less surfaces are ambiguous to photo-consistency and thus cannot be uniquely identified. Furthermore, glossy surfaces even when textured are difficult to be matched due to view-dependent changes.

2.3.3 Image-Based-Rendering

Image-Based-Rendering (IBR) aims to produce photo-realistic re-rendering of a scene given a collection of images from different views. While SfM and MVS explicitly model the geometry, IBR requires a full understanding of a scene that also includes material and lighting. Unlike classical rendering geometry, material and lighting do not necessarily have to be modeled explicitly for IBR. Over the years, different scene representations have been proposed that provide *shortcuts* to classical rendering. For instance, under the assumption that input images only differ in rotational changes, image stitching techniques [58] allow to blend overlapping regions without knowledge of explicit geometry. Another representation that does not model geometry explicitly are light fields [59]. Unlike image stitching, which does not allow for view-dependent changes, light fields model the radiance in every direction at every location in space, which is represented by a 5-dimensional plenoptic function [60]. Storing and sampling this function is not only memory expensive but also computationally infeasible. In practice, it is often reduced to 4 dimensions under the assumption that empty space such as air is transparent, i. e., radiance stays constant when travelling through empty space along a certain viewing direction [61] and therefore does not need to be modeled. Many more representations that go beyond the scope of this thesis exist and are discussed in more detail by [62]. Closely related to this thesis, the seminal work by Mildenhall et al. [54] (NeRF) proposes to approximate a radiance field using an MLP that given a spatial location

and direction returns the opacity and colour. It further approximates the behaviour of the plenoptic function by using differentiable volume rendering to produce the corresponding colour values for a given camera pose. Despite the stunning result quality, rendering an image is computationally very expensive and can take up to several minutes.

Follow-up methods such as NSVF [63] combine NeRF and voxel grids to improve the scalability and expressivity of the model whereas Yariv et al [64] use sphere tracing to render signed distance fields.

2.4 3D Reconstruction

While SfM, MVs and NeRF require many views of the same scene, this thesis focuses on 3D reconstruction from only a few or even single images. This task requires training neural networks that are capable of learning a prior over an entire dataset in order to make accurate predictions for unseen objects at test time.

In this section, related methods that provide different levels of supervision will be reviewed and sorted in decreasing order by the level of supervision they receive. When reconstructing category-specific objects silhouette supervision is assumed to be a mandatory requirement in this thesis. Currently, very few approaches exist that seek to perform object-specific single-view reconstruction without silhouette supervision [65]. Therefore, silhouettes are assumed to be available as supervision, unless stated otherwise.

2.4.1 3D supervision

Full supervision in the form of annotated 3D datasets such as Shapenet [11] allows for supervised learning in 3D. Several methods suggest learning 3D voxel representations conditioned on single images. The general design of such networks is based on an encoder that generates a latent code which is then fed into a generator to produce a 3D representation (i. e., a voxel grid). Wu et al. [66], Yang et al. [67], Varley et al. [68] learn binary 3D voxel grids from 2.5D depth maps. Given a 2D natural image, [69] predict 2.5D sketches (depth maps, normals and silhouettes) in a supervised manner, before feeding these into a 3D decoder to estimate the full 3D shape. In a similar

fashion, Wang et al. [70] first predict a coarse shape, followed by a 3D refinement decoder. Wu et al. [71] directly predict 3D shapes from single images which is accomplished by adversarial training using a 3D discriminator in combination with a supervised 3D loss. Girdhar et al. [72] propose a joint embedding of 3D voxels and 2D images. [73] recursive design takes multiple images as input and refines the 3D reconstructions in a recurrent manner. Kar et al. [74] also require multiple images and propose a simple "unprojection" network component to establish a relation between 2D pixels and 3D voxels without resolving occlusion. Addressing the problem of limited voxel resolutions, Mescheder et al. [75] propose to predict binary per-point occupancies in a continuous manner. Alternatively, [76, 77] reconstruct meshes from single views and Fan et al. [78] produce points instead of voxel grids from 2D images.

All aforementioned methods require full supervision in the form of images and corresponding 3D CAD models. In order for this to work, the models in the dataset are required to have the same scale and to be canonically aligned, e. g., all cars are up-right and face forward. The reason being, that predicting scale and pose of objects from images is ambiguous. Currently, annotated 3D datasets that provide full supervision suffer from a domain gap to real-world data and further lack in size. Thus, approaches that do not require supervision in the form of 3D, but rather learn to reason about 3D from 2D images were proposed. The key to success is to simultaneously predict the pose and shape from images which is highly ambiguous.

2.4.2 Pose supervision

To simplify the problem, several methods have been proposed that provide images with corresponding camera poses. In that case, one can follow the general design of fully 3D supervised methods. First, the input image is embedded into a global latent space before estimating the 3D geometry using a decoder network. Additionally, the 3D shape is then projected back to 2D given the camera pose and the objective function is defined in 2D rather than 3D. In order for it to work, the objects with respect to the camera poses have to be canonically aligned and of the same scale, similar to fully 3D supervised methods. If that was not the case, the shape and pose

ambiguity would not be resolved as the decoder has neither any knowledge about the alignment of the object nor the size. Another requirement is that the projection step has to be differentiable for backpropagation.

The pioneering work by Rezende et al. [79] proposes a black box renderer for meshes which relies on REINFORCE [80] to approximate gradients. A generalization from visual hull maps to full 3D voxel grids is proposed by Yan et al. [81]. OpenDR proposes [82] the first differentiable renderer for meshes that is limited to surface orientation and shading. The key difficulty here is to make the rasterization step differentiable as all other steps in the common graphics pipeline are differentiable. Several methods are proposed to do so [83, 84]. They all learn to deform an initial mesh template shape while also learning textures in the form of UV maps. Empirically, the preferred choice for the template is a sphere. This thesis focuses on voxels and neural representations which can express arbitrary topology, e. g., chairs with drastically different layouts, which are not a mere deformation of a base shape. Chen et al. [85] improve existing methods while additionally estimating lighting. Tulsiani et al. [86] enable differentiable volume rendering, allowing for reconstruction of voxel grids from images only.

Unfortunately, pose supervision is hard to acquire in real-world scenarios, due to the reasons explained above. Thus, the aforementioned methods rely on synthetic datasets which contain 3D shapes. These can be rendered from random poses to generate training data. To circumvent the need for direct pose supervision, cues such as key points, template shapes or multi-view data are exploited, which will be discussed in the next sections.

2.4.3 Keypoint supervision

Manual pose labeling is a task that humans are not capable of. One of the early works to tackle single-image 3D reconstruction on natural images that aims to require as little supervision and user input as possible was developed by Cashman and Fitzgibbon [87]. The idea is to fit a morphable template model to given silhouettes and key points [87]. First, a user has to position the template shape roughly over the desired object to reconstruct. Then the method optimizes shape and poses jointly

based on given silhouettes and key points. Another line of work [88, 89] first exploit SfM to estimate rough camera poses using keypoint and silhouette information. In a second step, shape surrogates, i. e., similar object shapes from a 3D dataset, are sampled and the best matching shape is chosen based on visual hull. In a similar fashion, CMR [90] first use SfM and 2D key points to initialize camera poses on the CUB [91] dataset. They then train a neural network to predict refined camera poses, template deformations and texture maps. With the use of differentiable mesh rendering [83] they optimize for silhouette, projected 2D key points and textured renderings. These methods still access 3D shape templates and 2D key points, despite not requiring any explicit 3D supervision.

2.4.4 Multi-view supervision

Multi-view information can help to alleviate the need for pose and key point information. Inter-view constraints can be used to estimate depth maps [92, 93] using reprojection constraints: If the depth label is correct, re-projecting one image into the other view has to produce the other image. Tulsiani et al. [94] propose a method that is supervised by multi-view pairs of the same object category with an associated verification image for the second image. They use the first image to predict the shape as a voxel grid while the second image is used to predict the pose. The predicted shape is then rendered from the estimated pose, which enables the resulting image to be compared to the verification image, ultimately enforcing multi-view consistency. As a consequence, the predicted shape is aligned canonically. In a similar fashion, Insafutdinov and Dosovitskiy [95] enforce multi-view consistency, however, instead of using a voxel grid as shape representation they predict a point cloud. Using object-specific video data, Novotny et al.[96, 97] canonically align point clouds. They predict poses of multiple images of the same object and then enforce the relative poses to be consistent. Training on multiple different objects enforces the pose predictor to predict absolute poses.

Recent works produce pixel-wise feature encodings per input view, which are then aggregated in different ways using knowledge of relative poses [98, 99, 100, 101, 102]. Yu et al. [101] averages features over multiple views. However, simply

averaging features from significantly different viewpoints hurts performance due to bleeding artefacts. Wang et al. [102] learn to interpolate between views in an IBR fashion, which prevents inpainting in unseen areas. Similar to those works, an aggregation function that prioritizes features based on their relative angles is proposed in Chapter 4.

2.4.5 Template supervision

Multi-view data curation is feasible, however, unstructured image data is easier to retrieve and hence more appealing. In the following, methods that solely receive template supervision are presented. This is desired as manual annotation of key points is very time-consuming and does not scale whereas in the case of template supervision, only a single template is required for a specific class of objects. Kulkarni et al. [103] predict a pixel-to-surface mappings that are consistent across a canonical 3D template. As a result, pose prediction does not require keypoint information, however, the shape itself is not learned explicitly as it is given by the template. Kulkarni et al. [104] extend this idea by also learning articulations.

Goel et al. [105] jointly predict pose and deform a 3D template. In a first pass, shape, texture as well as multiple camera poses are estimated for a single input image. During training, each proposed view is rendered and compared to a reconstruction loss. Once the network is converged, the best possible camera for a single image is trained by another network such that at test time the approach is able to predict a single pose. Due to the rough shape template only small intra-class variations are possible, i. e., different articulations of a bird such as closed and open wings are not possible.

2.4.6 Minimal supervision

In the previous sections, we have seen approaches that require more than just images as supervision. Since large amounts of data are key for machine learning techniques it is desirable to only rely on unstructured image collections as these exist in abundance. Exploiting the StyleGAN [106] latent space, Zhang et al. [107] extract camera poses that can be used for self-supervision when rendering. However, a few manual pose

annotations are required for bootstrapping. Wu et al. [108] propose a fully self-supervised approach that exploits shading to extract albedo, pose, and lighting. This has only been demonstrated for limited viewpoint variation and does not reconstruct a full 3D model but rather 2.5D in the form of a depth map. Li et al. [109], use self-supervised semantic features of [110] as a proxy for 2D key points as well as further constraints such as symmetry to help the reconstruction. Similar to the approach in Chapter 3, Gadelha et al. [111] tackle the problem at hand in an adversarial fashion. They train a 3D voxel generator whose 2D projections are not distinguishable from a 2D discriminator. The method receives three sources of supervision: view information gets explicitly encoded as a dimension in the latent vector; views come from a manually-chosen 1D subspace (circle); and there are only 8 discrete views. Those constraints are alleviated in Chapter 3, as PlatonicGAN works on completely unstructured image collections in a fully self-supervised manner. Extending this idea, GRAF [112] propose the use of neural representation in the form of an MLP instead of a voxel-grid, allowing for higher resolutions. Unlike the aforementioned methods that exploit explicit differentiable rendering, HoloGAN [113] learns a 3D voxel feature grid that is mapped to 2D via reshaping and the final images are rendered using 2D convolutions. While this method can 'hallucinate' high-quality images, the resulting images from different poses are not multi-view consistent as no explicit 3D model is enforced.

2.5 Texture Synthesis

Given an image, the goal of texture synthesis is to generate a new image that has the same statistics but is different pixel-wise. A classic definition of texture is defined by Julesz [114]: *a texture is an image full of features that in some representation have the same statistics.*

Traditionally, textures have been classified as either deterministic or stochastic. A deterministic texture is usually composed of small and easily identifiable components that form a regular pattern whereas a stochastic texture is comprised of less recognisable components as they appear to be more random. A few examples of

Figure 2.6: Examples of textures.

different textures are shown in Fig. 2.6 [115].

Another property that textures can be distinguished by is stationarity. An image fulfils the stationarity constraint if the statistics of two random crops of an image are identical, see Fig. 2.7. In this thesis, textures are assumed to be stochastic and stationary.

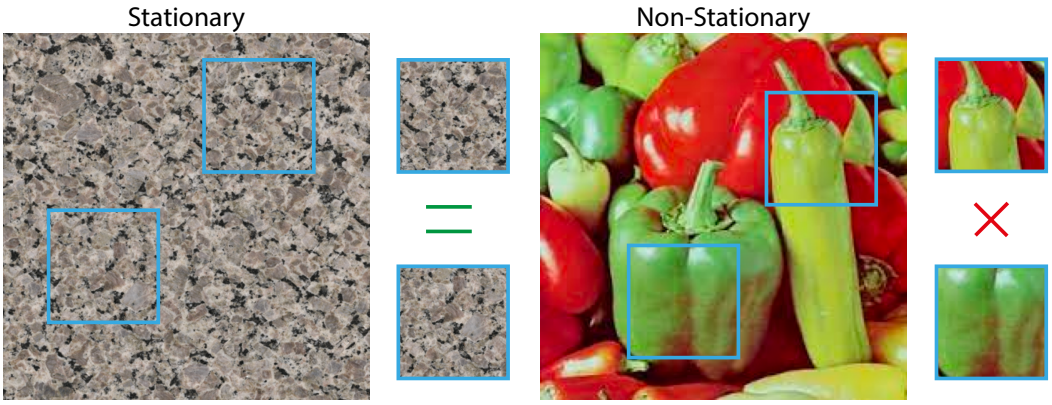


Figure 2.7: A visualisation of a stationary (left) and a non-stationary (right) texture.

2.5.1 Traditional texture synthesis

Capturing the variations of nature using stochastic textures has a long history [116]. Making noise useful for graphics and vision is due to Perlin's 1995 work [117]. Here, textures are generated by computing noise at different frequencies and mixing it with linear weights. A key benefit is that this noise can be evaluated in 2D as well as in 3D making it popular for many graphics applications. Portilla and Simoncelli [118] have provided a practical method to compute representations in which to do statistics on,

using linear filters on multiple scales. Wavelet noise [119] moved this idea further by band-limiting the noise that is combined. Such methods can be used for materials, e. g., gloss maps, bump maps, etc. It however does not provide a solution to acquire a texture from an exemplar, which is left to manual adjustment.

Computer vision typically had looked into generating textures from exemplars, such as by non-parametric sampling [120], vector quantization [121], optimization [122] or nearest-neighbour field synthesis (PatchMatch [123]) with applications in in-painting and also (3D) graphics. However, achieving spatial coherence and details as well as computational scalability remains a challenge and limits their adoption in production rendering or games.

The word "texture" can be ambiguous to mean stochastic variation, as well as images attached on surfaces to localize colour features. This thesis focuses on stochastic variation in the sense of Julesz [114] or Portilla and Simoncelli [118].

2.5.2 Texture synthesis meets deep learning

The next level of quality was achieved when representations became learned, such as the internal activations of the VGG network [29]. Neural style transfer [124] as explained in Sec. 2.1.4 played a key role. VGG was also used for optimization-based multi-scale texture synthesis [125]. Such methods require optimizations for each individual exemplar.

Ulyanov et al. [126] and Johnson et al. [127] have proposed networks that directly produce the texture without optimization. While now a network generated the texture, it was still limited to one exemplar, and no diversity was demonstrated. However, noise at different resolutions [117] is input to these methods, as well as inspiration to the approach presented in Chapter 5. Follow-up work [128] has addressed exactly this difficulty by introducing an explicit diversity term i. e., asking all results in a batch to be different. Unfortunately, this frequently introduces mid-frequency oscillations of brightness that appear admissible to VGG instead of producing true diversity. In this thesis, diversity is achieved, by restricting the network input to stochastic values only, i. e., diversity-by-construction.

In the human vision [114] and computer vision literature [120, 129], texture synthesis

exclusively refers to stochastic variation. In computer graphics, e. g., OpenGL, "texture" can model both stochastic and non-stochastic variations of colour. For example, Visual Object Networks [130] generate a voxel representation of shape and diffuse albedo and refer to the localized colour appearance, e. g., wheels of a car are dark, the rim is silver, etc., as "texture". Similarly, Oechsle et al. [131] and Saito et al. [132] use an implicit function to model variation of appearance beyond voxel resolutions.

The comparisons in Sec. 5.4 will show how methods tackling space of non-stochastic texture variation [131, 130], are not suitable to model stochastic appearance. This thesis makes progress towards learning spaces of stochastic and non-stochastic textures.

Some work has used adversarial training to capture the essence of textures [133, 134], including the non-stationary case [135] or even inside a single image [133]. In particular, StyleGAN [136] generates images with details by transforming noise in adversarial training. These challenges of adversarial training can be avoided by training a neural network to match VGG statistics as shown in Chapter 5.

2.5.3 Space of Textures

At any rate, none of the texture works in graphics or vision [117, 124, 126, 120, 123, 137, 138] generate a space of stochastic textures, as is suggested in Chapter 5. Current methods work on a single texture while the ones that work on a space of exemplars [130, 131] do not create stochastic textures. Chapter 5 closes this gap, by creating a space of stochastic textures.

Finally, all these methods require learning the texture in the same space it will be used, while the approach in Chapter 5 can operate in any dimension and across dimensions, including the important case of generating procedural 3D solid textures from 2D observations [139] or slices [140] only.

2.6 Material Modeling

In Chapter 6, the physical surface characteristics of textures are modeled in the form of BRDFs (Bidirectional Reflectance Distribution Function) [31]. Representing

appearance in simulation-based graphics has been an active research field for decades. The survey by Guarnera et al. [141] presents detailed discussion of the many different material models and BRDF acquisition approaches.

Many methods have been proposed to acquire materials using data-driven approaches. Matusik [142] proposed a data-driven BRDF linear model. More recently, Rematas et al. [143] extract reflectance maps from 2D images using a CNN trained in a supervised manner. Material and illumination acquisition was further explored by Georgoulis et al. [144]. Deschaintre et al. [145] proposed a rendering loss to capture svRBDFs from flash images.
Nam et al. [146] jointly reconstructed Spatially-varying Bi-directional Reflectance Distribution Function (svBRDF), normals, and 3D geometry in an iterative inverse-rendering setup towards a practical acquisition setup, while different methods relied on deep learning to estimate object shape and svBRDF from one or multiple images [147, 148, 149]. Li et al. [150] propose a weakly supervised learning-based method for generating novel category-specific 3D shapes and demonstrate that it can help in learning material-class specific svBRDFs from image distributions. Ye et al. [151] use a mixture of images and procedural material maps to train a network for modeling svBRDFs. Hu et al. [152] developed a reduced svBRDF model, using only diffuse and normal channels, towards solving inverse procedural textures matching from reference, while Guo et al. [153] used Bayesian inference for material synthesis. Recently, Shi et al. [154] developed a differentiable material graph nodes library to optimize material parameters to match an input material, given material graphs.
U-net [155] inspired many approaches for image-to-image translation to translate RGB pixels to material attributes [147, 156, 157, 145].

Most work now includes a differentiable shading step [158, 147, 145, 159, 160], which is a key component for self-supervised material synthesis as demonstrated in Chapter 6. Gao et al. [161] and Guo et al. [160] propose to use a post-optimization in an encoded latent space, improving an initial material estimation, and comparing renderings of their results directly to their input pictures. Deschaintre et al. [162] propose to fine-tune their material acquisition network on svBRDF parameter exam-

ples to transfer them to a larger scale. Zhou and Kalantari [163] propose a partially unsupervised approach, which requires real image pairs under different illumination conditions. The approach presented in Guo et al. [164] addresses the issue of strong highlights baked into svBRDF maps through highlight-aware convolutions and an attention-based feature selection module. Chapter 6 exploits the stationarity of textures, which inherently prevents any flash residual to be left in the results.

All these approaches focus on capturing a single instance of a svBRDF map, but with little or no editing options across materials (space) or generalization across the spatial domain (diversity). For rapid materials generation, Zsolnai-Fehér et al. [165] propose to use Gaussian process regression.

Most of these methods require synthetic svBRDF supervision for training, while directly learning from flash images without access to channel-level supervision is demonstrated in Chapter 6. In particular, this removes the risk of a domain gap between synthetic and real materials and enables fine-tuning as we will see.

Chapter 6 builds upon the work by Aittala et al. [166] who extended the approach of Gatys et al. [124] to generate svBRDF parameter maps from a single picture of a stationary material exemplar and propose an approach for improved diversity, generation and quality.

2.6.1 Spaces-of

Spaces of colour [167], materials [142, 160, 161], textures [168], faces [169], human bodies [170], and more have been useful in graphics for content creation and edition. Matusik et al. [168] has devised a space of textures. Here, users can interpolate combinations of visually similar textures. They warp all pairs of exemplars to each other and construct graph edges for interpolation when there is evidence that the warping is admissible. To blend between them, histogram adjustments are made. Consequently, interpolation between exemplars does not take a straight path in pixel space from one to the other but traverses only valid regions. Photoshape [171] learns the relation of given material textures over a database of 3D objects. Serrano et al. [172] allow users to semantically control captured BRDF data. They represent BRDFs using the derived principal component basis [142] and map the first five

PCA components to semantic attributes through learned radial basis functions. Guo et al. [160] and Gao et al. [161] produce spaces of materials that can be interpolated. Chapter 6 takes inspiration from this body of work and builds a latent space allowing svBRDFs generation and interpolation.

Chapter 3

Escaping Plato's Cave: 3D Shape From Adversarial Rendering

3.1 Overview

In this chapter, we suggest a method to learn 3D structure from 2D images only (Fig. 3.1). Reasoning about the 3D structure from 2D observations without assuming anything about their relation is challenging as illustrated by Plato's Allegory of the Cave [173]: *How can we hope to understand higher dimensions from only ever seeing projections?* If multiple views (maybe only two [92, 93]) of the same object are available, multi-view analysis without 3D supervision has been successful. Regrettably, most photo collections do not come in this form but are now and will remain *unstructured*: they show random instances under random poses, uncalibrated lighting in unknown relations, and multiple views of the same objects are not available.

Our first main contribution (Sec. 3.2) is to use adversarial training of a 3D generator with a discriminator that operates exclusively on widely available unstructured collections of 2D images, which we call *platonic discriminator*. Here, during training, the generator produces a 3D shape that is projected (rendered) to 2D and presented to the 2D Platonic discriminator. Making a connection between the 3D generator and the 2D discriminator, our second key contribution is enabled by a family of *rendering layers* that can account for occlusion and colour (Sec. 3.3).

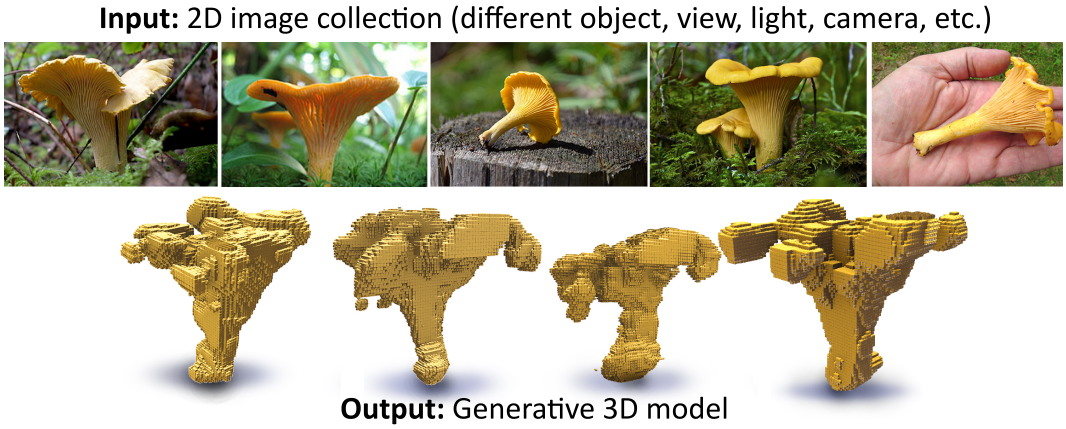


Figure 3.1: PLATONICGANs allow converting an unstructured collection of 2D images of a rare class (subset shown on top) into a generative 3D model (random samples below).

These layers do not need any learnable parameters and allow for backpropagation [18]. From these two key blocks we construct a system that learns the 3D shapes of common classes such as chairs and cars, but also exotic classes from unstructured 2D photo collections.

We demonstrate 3D reconstruction from a single 2D image as a key application (Sec. 3.4). While recent works focus on using as little explicit supervision [174, 83, 175, 86, 94, 111] as possible, they all rely on either annotations, 3D templates, known camera poses, specific views or multi-view images during training. Our approach takes it a step further by receiving no such supervision, see Tbl. 3.1.

Table 3.1: Taxonomy of different methods that learn 3D shapes with no explicit 3D supervision. We compare Kanazawa et al. [174], Kato et al. [83], Eslami et al. [175], Tulsiani et al. [86], Tulsiani et al. [94], PrGan [111] with our method in terms of degree of supervision.

Supervision at training time	[174]	[83]	[175]	[86]	[94]	[111]	Ours
Annotation-free	×	✓	✓	✓	✓	✓	✓
3D template-Free	×	×	✓	✓	✓	✓	✓
Unknown camera pose	✓	×	×	×	✓	✓	✓
No pre-defined camera poses	✓	✓	✓	✓	×	×	✓
Only single view required	✓	×	×	×	×	✓	✓
Color	✓	✓	✓	✓	×	×	✓

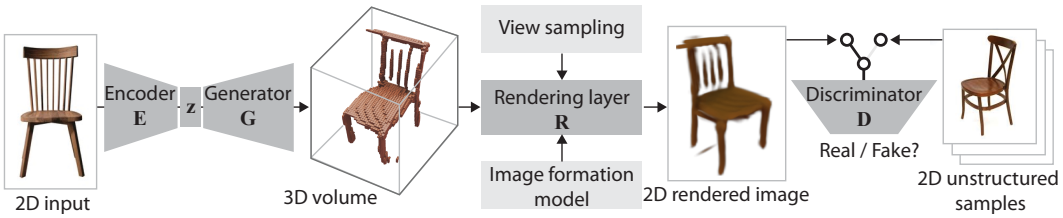


Figure 3.2: Overview: We encode a 2D input image using an encoder E into a latent code $\mathbf{z}$ and feed it to a generator G to produce a 3D volume. This 3D volume is inserted into a rendering layer R to produce a 2D rendered image which is presented to a discriminator D. The rendering layer is controlled by an image formation model: visual hull (VH), absorption-only (AO) or emission-absorption (EA) and view sampling. The discriminator D is trained to distinguish such rendered imagery from an unstructured 2D photo collection, i. e., images of the same class of objects, but not necessarily having repeated instances, view or lighting and with no assumptions about their relation (e.g., annotated feature points, view specifications).

3.2 3D Shape From 2D Photo Collections

We now introduce PLATONICGAN (Fig. 3.2). The rendering layers used here will be introduced in Sec. 3.3.

Common GAN Our method is a classic (generative) adversarial design [27] with two main differences: The discriminator D operates in 2D while the 3D generator G produces 3D output. The two are linked by a fixed-function projection operator, i. e., non-learnable (see Sec. 3.3).

Let us recall the classic adversarial learning of 3D shapes [71], which is a min-max game

$$\min_{\Theta} \max_{\Psi} \; c_{\text{Dis}}(\Psi) + c_{\text{Gen}'}(\Theta) \tag{3.1}$$

between the discriminator and the generator cost, respectively c_{Dis} and $c_{\text{Gen}'}$.

The discriminator cost is

$$c_{\text{Dis}}(\Psi) = \mathbb{E}_{p_{\text{Data}}(\mathbf{x})}[\log(D_\Psi(\mathbf{x}))] \tag{3.2}$$

where D_Ψ is the discriminator with learned parameters Ψ which is presented with samples $\mathbf{x}$ from the distribution of real 3D shapes $\mathbf{x} \sim p_{\text{Data}}$. Here $\mathbb{E}_p$ denotes the

expected value of the distribution p.

The generator cost is

$$c_{\text{Gen}'}(\Theta) = \mathbb{E}_{p_{\text{Gen}}(\mathbf{z})}[\log(1 - D_\Psi(G_\Theta(\mathbf{z}))] \tag{3.3}$$

where G_Θ is the generator with parameters Θ that maps the latent code $\mathbf{z} \sim p_{\text{Gen}}$ to the data domain.

PLATONICGAN The discriminator cost is calculated identically to the common GAN with the only difference that the input samples are rendered 2D images with generation cost

$$c_{\text{Gen}}(\Theta) = \mathbb{E}_{p_{\text{Gen}}(\mathbf{z})}\mathbb{E}_{p_{\text{View}}(\omega)}[\log(1 - D_\Psi(R(\omega, G_\Theta(\mathbf{z}))))], \tag{3.4}$$

where R projects the generator result $G_\Theta(\mathbf{z})$ from 3D to 2D along the sampled view direction ω. See Sec. 3.2.1 for details.

While many parameterizations for views are possible, we choose an orthographic camera with fixed upright orientation that points at the origin from a Euclidean position $\omega \in \mathbb{S}^2$ on the unit sphere. $\mathbb{E}_{p_{\text{View}}(\omega)}$ is the expected value across the distributions $\omega \sim p_{\text{View}}$ of views.

PLATONICGAN3D Reconstruction Two components in addition to our Platonic concept are required to allow for 3D reconstruction, resulting in

$$\min_{\Psi} \max_{\Theta,\Phi} \; c_{\text{Disc}}(\Psi) + c_{\text{Gen}}(\Theta, \Phi) + \lambda c_{\text{Rec}}(\Theta, \Phi), \tag{3.5}$$

where c_{Gen} includes an encoding step and c_{Rec} encourages the encoded generated-and-projected result to be similar to the encoder input where $\lambda = 100$. We detail both of these steps in the following paragraphs:

Generator The generator G_Θ does not directly work on a latent code $\mathbf{z}$, but allows for an encoder E_Φ with parameters Φ that encodes a 2D input image $\mathbf{I}$ to a latent

code $\mathbf{z} = E_\Phi(\mathbf{I})$. The cost becomes,

$$c_{\text{Gen}}(\Theta,\Phi) = \mathbb{E}_{p_{\text{Dat}}(\mathbf{I})}\mathbb{E}_{p_{\text{View}}(\omega)}[\log(1 - D_\Psi(R(\omega, G_\Theta(E_\Phi(\mathbf{I})))))]. \tag{3.6}$$

Reconstruction We encourage the encoder E_Φ and generator G_Θ to reproduce the input in the $\mathscr{L}_2$ sense: by convention the input view is $\omega_0 = (0,0)$,

$$c_{\text{Rec}}(\Theta,\Phi) = \|\mathbf{y} - R(\omega_0, G_\Theta(E_\Phi(\mathbf{I})))\|_2^2 \tag{3.7}$$

where $\mathbf{y}$ represents the ground truth image. While this step is not required for generation it is mandatory for reconstruction. Furthermore, it adds stability to the optimization as it is easy to find an initial solution that matches this 2D cost before refining the 3D structure.

3.2.1 Optimization

Two key properties are essential to successfully optimize our PLATONICGAN: First, maximizing the expected value across the distribution of views p_{View} and second, back-propagation through the projection operator R. We extend the classic GAN optimization procedure in Alg. 1.

Algorithm 1 PLATONICGANReconstruction Update Step

$I_{\text{Dat}} \leftarrow \text{SAMPLEIMAGE}(p_{\text{Dat}})$
$\omega \leftarrow \text{SAMPLEVIEW}(p_{\text{View}})$
$z \leftarrow \text{E}(I_{\text{Dat}})$
$v \leftarrow \text{G}(z)$
$I_{\text{View}} \leftarrow \text{R}(\omega, v)$
$I_{\text{Front}} \leftarrow \text{R}(\omega_0, v)$
$c_{\text{Dis}} \leftarrow \log D(I_{\text{Dat}}) + \log(1 - D(I_{\text{View}}))$
$c_{\text{Gen}} \leftarrow \log(1 - D(I_{\text{View}}))$
$c_{\text{Rec}} \leftarrow \text{L2}(I_{\text{Dat}} - I_{\text{Front}})$
$\Psi \leftarrow \text{MAXMIZE}(c_{\text{Dis}})$
$\Theta, \Phi \leftarrow \text{MINIMIZE}(c_{\text{Gen}} + \lambda c_{\text{Rec}})$

Projection We focus on the case of a 3D generator on a regular voxel grid $\mathbf{v}^{n_c \times n_p^3}$ and a 2D discriminator on a regular image $\mathbf{I}^{n_c \times n_p^2}$ where n_c denotes the number of channels and $n_p = 64$ corresponds to the resolution. In section Sec. 3.3, we discuss three different projection operators. We use $R(\omega, \mathbf{v})$ to map a 3D voxel grid $\mathbf{v}$ under a view direction $\omega \in \mathbb{S}^2$ to a 2D image $\mathbf{I}$.

We further define $R(\omega, \mathbf{v}) := \rho(\mathsf{T}(\omega)\mathbf{v})$ with rotation matrix $\mathsf{T}(\omega)$ according to the view direction ω and an image formation function $\rho(\mathbf{v})$ that is view-independent. The same transformation is shared by all implementations of the rendering layer, so we will only discuss the key differences of ρ in the following. Note that a rotation and a linear resampling are back-propagatable and typically provided in a deep learning framework, e. g., as `torch.nn.functional.grid_sample` in PyTorch [176]. While we work in orthographic space, ρ could also model a perspective transformation.

View sampling We assume uniform view sampling.

3.3 Rendering Layers

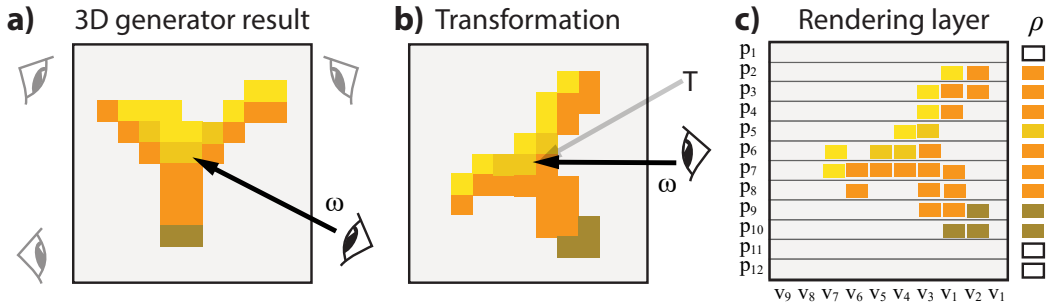


Figure 3.3: Rendering layers (Please see text).

Rendering layers (Fig. 3.3) map 3D information to 2D images so they can be presented to a discriminator. We first assume the 3D volume to be rotated (Fig. 3.3, a) into camera space from view direction ω (Fig. 3.3, b), such that the pixel value p is to be computed from all voxel values $\mathbf{v}_i$ and only those (Fig. 3.3, c). The rendering layer maps a sequence of n_z voxels to a pixel value $\rho(\mathbf{v}) \in \mathbb{R}^{n_c \times n_p^3} \to \mathbb{R}^{n_c \times n_p^2}$. Composing the full image $\mathbf{I}$ just amounts to executing ρ for every pixel p resp. all voxels $\mathbf{v} = v_1, \ldots, v_{n_z}$ at that pixel.

Note, that the rendering layer does not have any learnable parameters. We will now discuss several variants of ρ, implementing different forms of volume rendering [44]. Fig. 3.4 shows the image formation models we currently support.

Visual hull (VH) Visual hull [177] is the simplest variant (Fig. 3.4). It converts scalar density voxels into binary opacity images. A voxel value of 0 means empty space and a value of 1 means fully occupied, i. e., $v_i \in [0,1]$. Output is a binary value indicating if any voxel blocked the ray. It is approximated as

$$\rho_{\mathrm{VH}}(\mathbf{v}) = 1 - \exp(\sum_i -v_i). \tag{3.8}$$

Note that the sum operator can both be back-propagated and is efficiently computable on a GPU using a parallel scan. We can apply this to learn 3D structure from binary 2D data such as segmented 2D images.

Absorption-only (AO) The absorption-only model is the gradual variant of visual hull. This allows for "softer" attenuation of rays. It is designed as:

$$\rho_{\mathrm{AO}}(\mathbf{v}) = 1 - \prod_i (1 - v_i). \tag{3.9}$$

If v_i are fractional the result is similar to an x-ray, i. e., $v_i \in [0,1]$. This image formation allows learning from x-rays or other transparent 2D images. Typically, these are single-channel images, but a coloured variant (e. g., x-ray at different wavelength or RGB images of coloured transparent objects) could technically be done.

Emission-absorption (EA) Emission-absorption allows the voxels not only to absorb light coming towards the observer but also to emit new light at any position. This

Figure 3.4: Different image formation models visual hull (VH), absorption-only (AO) and emission-absorption (EA).

interplay of emission and absorption can model occlusion, which we will see is useful to make 3D sense of a 3D world. Fig. 3.3 uses emission-absorption with high absorption, effectively realizing an opaque surface with visibility.

A typical choice is to have the absorption v_a monochromatic and the emission v_e chromatic.

The complete emission-absorption equation is

$$\rho_\mathrm{EA}(\mathbf{v}) = \sum_{i=1}^{n_z} \underbrace{\left(\prod_{j=1}^{i}(1 - v_{\mathrm{a},j})\right)}_{\text{Transmission } t_i} v_{\mathrm{e},i} \tag{3.10}$$

While such equations are typically solved using ray-marching [44], they can be rewritten to become differentiable in practice: First, we note that the transmission t_i to voxel i is a product of one minus the density of all voxels before i. Similar to a sum such a cumulative product can be back-propagated and computed efficiently using parallel scans, e. g., using `torch.cumprod`. A numerical alternative, that performed similar in our experiments, is to work in the log domain and use `torch.cumsum`.

3.4 Evaluation

Our evaluation comprises of a quantitative (Sec. 3.4.4) and a qualitative analysis (Sec. 3.4.5) that compares different previous techniques and ablations to our work (Sec. 3.4.2).

3.4.1 Datasets

Synthetic We evaluate on two synthetic datasets: (a) ShapeNet [178] and (b) mammalian skulls [179]. For our quantitative analysis, we use ShapeNet models as 3D ground truth is required, but strictly only for evaluation, never in our training. 2D images of 3D shapes are rendered for the three image formation models VH, AO, EA. Each shape is rendered from a random view (50 per object), with random natural illumination. ShapeNet only provides 3D density volumes, which is not sufficient for EA analysis. To this end, we use volumetric projective texturing to propagate the

appearance information from thin 3D surface crust as defined by ShapeNet's textures into the 3D voxelization in order to retrieve RGBA volumes where A corresponds to density. We use shapes from the classes `airplane`, `car`, `chair`, `rifle` and `lamp`. The same train / validation / test split as proposed by [178] is adopted.
We also train on a synthetic x-ray dataset that consists of 466,200 mammalian skull x-rays [179]. We used the monkey skulls subset of that dataset ($\sim$30k x-rays).

Real We use two datasets of rare classes: (a) `chanterelle` (60 images) and (b) `tree` (37 images) (not strictly rare, but difficult to 3D-model). These images are RGBA, masked, on white background. Note, that results on these input data has to remain qualitative, as we lack the 3D information to compare to and do not even have a second view of the same object to even perform an image comparison.

3.4.2 Baselines and comparison

2D supervision First, we compare the publicly available implementation of PrGAN [111] with our Platonic method. PrGAN is trained on an explicitly created dataset adhering to their view restrictions (8 views along a single axis). Compared to our method, it is only trained on visual hull images, however for evaluation purposes absorption-only and emission-absorption (in the form of luminance) images are used as input images at test time. Note that PrGAN allows for object-space view reconstruction due to view information in the latent space whereas our method performs reconstruction in view-space. Due to the possible ambiguities in the input images (multiple images can belong to the same 3D volume), the optimal transformation into object space is found using a grid search across all rotations.

3D supervision The first baseline with 3D supervision is MULTI-VIEW, which has training-time access to multiple images of the same object [81] in a known spatial relation. Note, that this is a stronger requirement than for PLATONICGANthat does not require any structure in the adversarial examples: geometry, view, light – all change, while in this method only the view changes in a prescribed way.
The second competitor is a classic 3DGAN [71] trained with a Wasserstein loss [180] and gradient penalty [181].
To compare PLATONICGANagainst methods having access to 3D information, we

also propose a variant PLATONIC3D by adding the PLATONICGANadversarial loss term (for all images and shapes) to the 3DGAN framework.

3.4.3 Evaluation Metrics

2D evaluation measures Since lifting 2D information to 3D can be ambiguous, absolute 3D measures might not be the best suitable measures for evaluation on our task. For instance, a shift in depth of an object under an orthographic camera assumption will result in a higher error for metrics in 3D, but the shift would not have any effect on a rendered image. Thus, we render both the reconstructed and the reference volume from the same 10 random views and compare their images using SSIM / DSSIM [182] and VGG16 [29] features. For this re-rendering, we further employ four different rendering methods: the original (i. e., ρ) image formation (IF), volume rendering (VOL), iso-surface rendering with an iso-value of .1 (ISO) and a voxel rendering (VOX), all under random natural illumination.

3D evaluation measures We report root-mean-squared-error (RMSE), intersection-over-union (IoU) and chamfer distance (CD). For the chamfer distance, we compute a weighted directional distance:

$$d_{\mathrm{CD}}(T,O) = \frac{1}{N} \sum_{p_i \in T} \min_{p_j \in O} w_j \|\mathbf{p}_i - \mathbf{p}_j\|_2^2,$$

where T and O correspond to output and target volumes respectively, and w_j denotes the density value of the voxel at location $\mathbf{p}_j$. The weighting makes intuitive sense as our results have scalar values rather than binary values, i. e., higher densities get penalized more, and N is the total number of voxels in the volume. We give preference to such a weighting as opposed to finding a threshold value for binarization.

3.4.4 Quantitative evaluation

Tbl. 3.2 summarizes our main results for the `airplane` class. Concerning the image formation models, we see that the overall values are best for AO, which is expected: VH asks for scalar density but has only a binary image; AO provides internal structures but only needs to produce scalar density; EA is hardest, as it needs to resolve both density and colour. Nonetheless, the differences between us and

Table 3.2: Performance of different methods with varying degrees of supervision (superv.) **(rows)** on different metrics **(columns)** for the class `airplane`. Evaluation is performed on all three image formations (IF): visual hull (VH), absorption-only (AO) and emission-absorption (EA). Note, DSSIM and VGG values are multiplied by 10, RMSE by 10^2 and CD by 10^3. Lower is better except for IoU.

Method	IF	Superv.		2D Image Re-synthesis										3D Volume			FID
				VH		AO		EA		VOX		ISO					EA
		2D	3D	DSSIM	VGG	DSSIM	VGG	DSSIM	VGG	DSSIM	VGG	DSSIM	VGG	RMSE	IoU	CD	
PrGAN [111]	VH	✓	×	1.55	6.57	1.37	**4.85**	1.41	**4.63**	1.68	5.41	1.83	6.15	**7.46**	0.11	**3.59**	207
Ours	VH	✓	×	**1.14**	**5.37**	**1.16**	4.93	**1.12**	4.68	**1.33**	**5.22**	**1.28**	**5.96**	9.16	**0.20**	11.77	**55**
Mult.-View [81]	VH	✓	×	0.87	4.89	0.80	4.31	0.90	4.07	1.38	4.83	1.21	5.56	5.37	0.36	**9.31**	155
3DGAN [71]	VH	✓	×	0.83	5.01	**0.75**	4.02	0.86	**3.83**	1.30	4.73	1.17	5.82	**4.97**	**0.46**	14.60	111
Ours 3D	VH	✓	×	**0.81**	**4.82**	0.77	**3.98**	**0.83**	**3.83**	**1.18**	**4.59**	**1.09**	**5.50**	5.20	0.44	12.33	**98**
PrGAN [111]	AO	✓	×	1.41	6.40	1.27	4.80	1.27	4.52	1.53	5.32	1.63	6.00	7.11	0.09	**2.78**	190
Ours	AO	✓	×	**0.94**	**5.35**	**0.93**	**4.46**	**0.91**	**4.26**	**1.11**	**4.96**	**1.09**	**5.75**	**5.70**	**0.27**	6.98	**90**
Mult.-View [81]	AO	✓	×	0.95	4.99	0.78	4.23	0.91	4.01	1.51	4.92	1.29	5.39	**4.89**	0.34	**9.47**	165
3DGAN [71]	AO	✓	×	0.67	4.37	0.69	3.77	0.72	3.57	0.99	**4.25**	0.97	**4.92**	5.08	**0.43**	14.92	**58**
Ours 3D	AO	✓	×	**0.66**	**4.36**	**0.66**	**3.73**	**0.70**	**3.52**	**0.98**	4.28	**0.96**	4.94	5.17	0.37	15.43	64
PrGAN [111]	EA	✓	×	**1.31**	**6.22**	**1.15**	**4.77**	**1.16**	**5.37**	**1.36**	**6.71**	**1.47**	**7.07**	**6.80**	0.08	**2.36**	196
Ours	EA	✓	×	2.18	6.53	1.99	5.38	1.89	6.00	2.21	7.43	2.36	7.92	14.13	**0.13**	10.53	**181**
Mult.-View [81]	EA	✓	×	1.62	6.21	1.53	4.58	1.63	5.48	1.95	6.97	1.94	7.41	15.05	0.12	32.07	172
3DGAN [71]	EA	✓	×	0.89	5.28	**0.78**	**3.93**	0.98	4.79	1.29	6.76	1.30	7.09	**5.24**	**0.46**	**13.66**	110
Ours 3D	EA	✓	×	**0.82**	**4.71**	0.82	3.96	**0.97**	**4.77**	**1.12**	**6.12**	**1.16**	**6.47**	7.43	0.04	18.82	**73**

competitors are similar across the image formation models.

2D supervision We see that overall, our 2D supervised method outperforms PrGAN for VH and AO. Even though PrGAN was not trained on EA it wins for all metrics against our 2D supervised method. However, it even outperforms the 3D supervised methods 3DGAN and MULTI-VIEW which demonstrates the complexity of the task itself. However, PrGAN for EA only produces density volumes, unlike all other methods that produce RGBA volumes. Comparing our 2D supervised method against the 3D supervised methods we see that overall our method produces competitive results. Regarding MULTI-VIEW we sometimes even perform better.

3D supervision Comparing our PLATONIC3D variant to the 3D baselines we observe our method to mostly outperform them for 2D metrics. Not surprisingly our method performs worse for 3D metrics as our approach only operates in 2D.

In Tbl. 3.3 we investigate the performance across different classes. `rifle` performs best: the approach learns quickly that a gun has an outer 3D shape that is a revolute structure. `chair` performs worst, likely due to its high intra-class variation.

Table 3.3: Reconstruction performance of our method for different image formation models **(columns)** on different classes **(rows)**. The error metric is SSIM (higher is better).

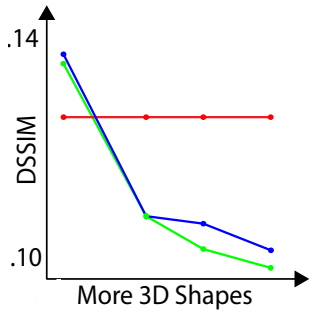
Class	VH			AO			EA		
	VOL	ISO	VOX	VOL	ISO	VOX	VOL	ISO	VOX
`plane`	0.93	0.92	0.93	0.94	0.93	0.93	0.85	0.76	0.77
`rifle`	0.95	0.94	0.95	0.95	0.94	0.95	0.90	0.78	0.80
`chair`	0.86	0.85	0.85	0.86	0.85	0.86	0.80	0.61	0.63
`car`	.841	.846	.851	.844	.846	.850	.800	.731	.743
`lamp`	.920	.915	.920	.926	.914	.920	.883	.790	.803

In Tbl. 3.4 we compare the mean VGG error of a vanilla 3D GAN trained only on 3D shapes, a Platonic approach accessing only 2D images, and PLATONIC3D that has access to both. We keep the number of 2D images fixed, and increase the number of 3D shapes available; the horizontal axis in Tbl. 3.4. Without making use of 3D supervision, the error of PLATONICGANremains constant, independent of the number of 3D models. Like this, we see that a PLATONICGAN(red line) can beat both other approaches in a condition where little 3D data is available (left). When more 3D data is available, PLATONICGAN(green line) wins over a pure 3D GAN (blue line). We conclude that adding 2D image information to a 3D corpus helps, and when the corpus is small enough even outperforms 3D-only supervised methods.

3.4.5 Qualitative

Synthetic Fig. 3.5 shows typical results for the reconstruction task. We see that our reconstruction can produce `airplane`, `chair` and `rifle` 3D models representative of the input 2D image. Most importantly, these 3D models look plausible for

Table 3.4: Effect of number of 3D shapes and 2D images on learning different methods in terms of mean DSSIM error. Lower is better.



2D images	70k	70k	70k	70k
3D shapes	5	50	250	1.5k
2D-3D ratio	14k	1.4k	280	47
• 3D	.135	**.108**	.106	.101
• Ours	**.125**	.125	.125	.125
• Ours 3D	.134	**.108**	**.102**	**.099**

Figure 3.5: Visual results for 3D reconstruction of three classes (`airplane`, `chair`, `rifle`) from multiple views.

multiple views, not only from the input one. The results on the chair category also show that the model captures the relevant variation, ranging from straight chairs over club chairs to armchairs. For `gun`, the results turn out almost perfect, in agreement with the numbers reported before. In summary, our quality is comparable to GANs with 3D supervision.

2D vs. 3D vs. 2D+3D Qualitative comparison of 2D-only, 3D-only and mixed 2D-3D training can be seen in Fig. 3.6.

Synthetic rare We explored reconstructing skulls from x-ray (i. e., the AO IF model) images [179] in Fig. 3.7. We find the method to recover both external and internal

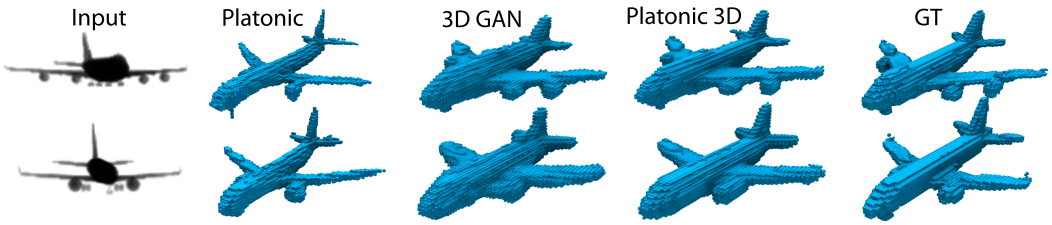


Figure 3.6: Comparison of 3D reconstruction results using the class `plane` between different forms of supervision **(columns)** for two different input views **(rows)**. PLATONICGAN, in the second column, can reconstruct a plausible plane, but with errors such as a wrong number of engines. The 3D GAN in the third column fixes this error but at the expense of slight mode collapse where instances look similar and slightly "fat". Combining a 3D GAN with adversarial rendering as in the fourth row is closest to the reference in the fifth row.

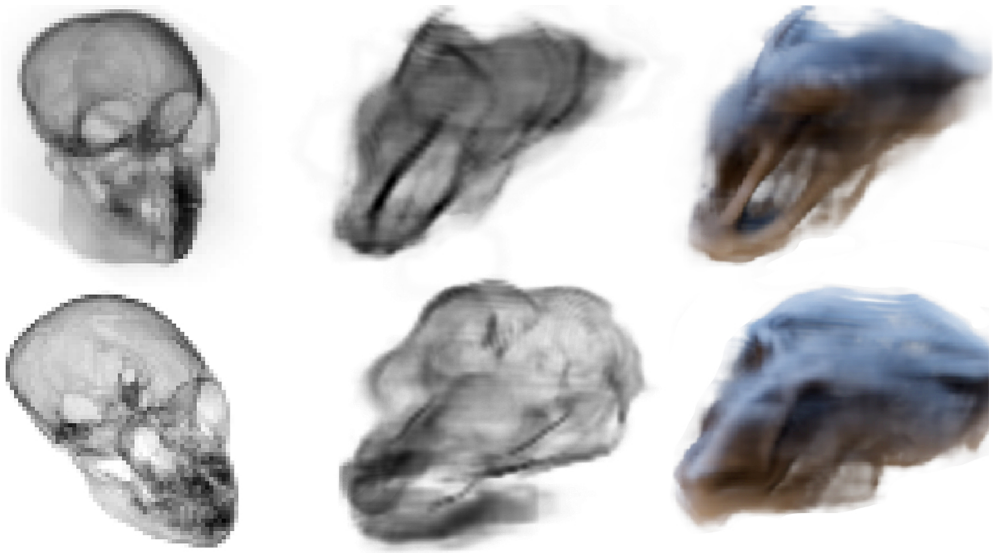


Figure 3.7: PlatonicGANs trained on 2D x-rays (i. e., AO IF) of mammalian skulls **(a)**. The resulting 3D volumes can be rendered from novel views using x-ray **(b)** and under novel views in different appearance, here, using image-based lighting **(c)**.

structures.

Real rare Results for rare classes are seen in Fig. 3.1 and Fig. Fig. 3.8. We see that our method produces plausible details from multiple views while respecting the input image, even in this difficult case. No metric can be applied to these data as no 3D volume is available to compare in 3D or re-project.

3.5 Discussion

Why not have a multi-view discriminator? It is tempting to suggest a discriminator that does not only look at a single image but at multiple views at the same time to judge if the generator result is plausible holistically. But while we can generate "fake" images from multiple views p_{Data}, the set of "real" natural images does not come in such a form. As a key advantage, our method only expects unstructured data: online repositories hold images with unknown camera, 3D geometry or illumination.

Failure cases are depicted in Fig. 3.9. Our method struggles to reconstruct the correct pose as lifting 2D images to 3D shapes is ambiguous for view-space reconstruction.

Supplemental More analysis, videos, training data and network definitions are available at `https://geometry.cs.ucl.ac.uk/projects/2019/`

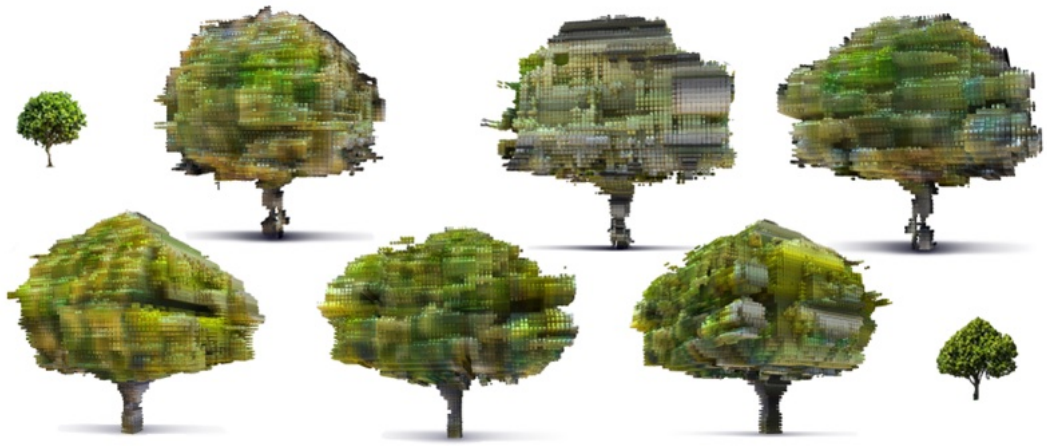


Figure 3.8: 3D Reconstruction of different trees using the emission-absorption image formation model, seen from different views **(columns)**. The small images were used as input. We see that PLATONICGANhas understood the 3D structure, including a distinctly coloured stem, fractal geometry and structured leave textures.

`platonicgan/`.

3.6 Conclusion

In this chapter, we have presented PLATONICGAN, a new approach to learning 3D shapes from unstructured collections of 2D images. The key to our "escape plan" is to train a 3D generator outside the cave that will fool a discriminator into seeing projections inside the cave.

We have shown a family of rendering operators that can be GPU-efficiently back-propagated and account for occlusion and colour. These support a range of input modalities, ranging from binary masks, over opacity maps to RGB images with transparency. Our 3D reconstruction application is built on top of this idea to capture varied and detailed 3D shapes, including colour, from 2D images. Training is exclusively performed on 2D images, enabling 2D photo collections to contribute to generating 3D shapes.

Future work could include shading that is related to gradients of density [44] into classic volume rendering. Furthermore, any sort of differentiable rendering operator ρ can be added. Devising such operators is a key future challenge. Other adversarial applications such as 2D supervised completion of 3D shapes seem worth exploring. Enabling object-space as opposed to view-space reconstruction would help to prevent

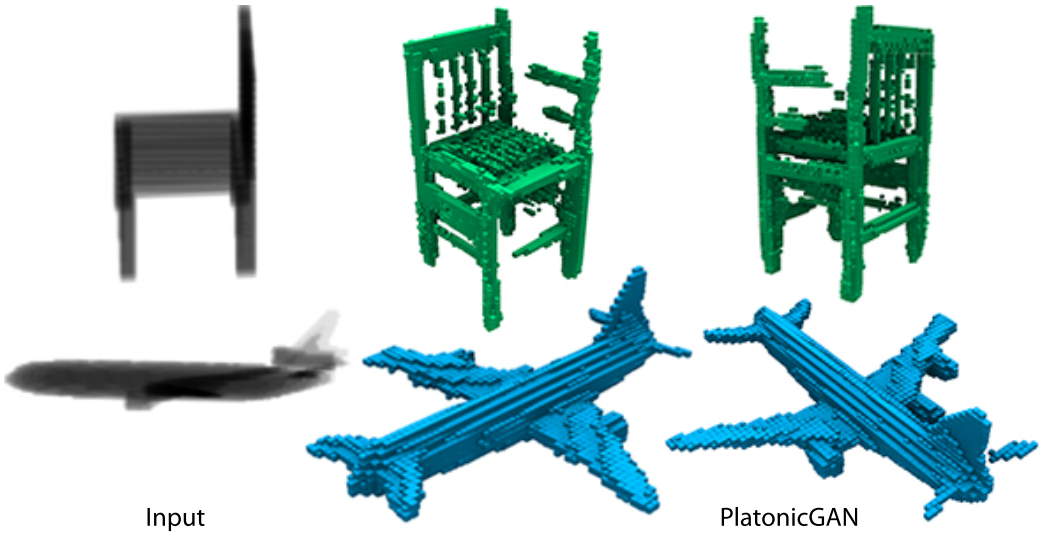


Figure 3.9: Failure cases of a chair **(top)** and an airplane **(bottom)**. The encoder is unable to estimate the correct camera pose due to view ambiguities in the input image and symmetries in the shapes. The generator then tries to satisfy multiple different camera poses.

failure cases as shown in Fig. 3.9.

While we combine 2D observations with 3D interpretations, similar relations might exist in higher dimensions, between 3D observations and 4D (3D shapes in motion) but also in lower dimensions, such as for 1D row scanners in robotics or 2D slices of 3D data such as in tomography.

Chapter 4

Unsupervised Learning of 3D Object Categories from Videos in the Wild

While the method presented in Chapter 3 is able to reconstruct shape and appearance of objects in a self-supervised manner without 3D data annotations, the result quality is limited by the voxel resolution and reconstruction was performed in camera space. Our goal in this chapter is to address these limitations and instead of single-image reconstruction, we seek to train a deep network that, given a small number of images of an object of a given category, recovers its shape and appearance in world space. Again, we are interested in working with challenging real data and with no manual annotations. We show that existing techniques leveraging meshes, voxels, or implicit surfaces, which work well for reconstructing isolated objects, fail on this challenging data. Finally, we propose a new neural network design, called warp-conditioned ray embedding (WCE), which significantly improves reconstruction while obtaining a detailed implicit representation of the object surface and texture.

4.1 Overview

Our first contribution is to introduce a new dataset of videos collected 'in the wild' by Amazon Mechanical Turk workers (Fig. 4.3). These videos capture a large number of object instances from the viewpoint of a moving camera, with an effect similar to a turntable. Viewpoint changes are estimated with high accuracy using off-the-shelf Structure from Motion (SfM) techniques. Hundreds of videos of several different

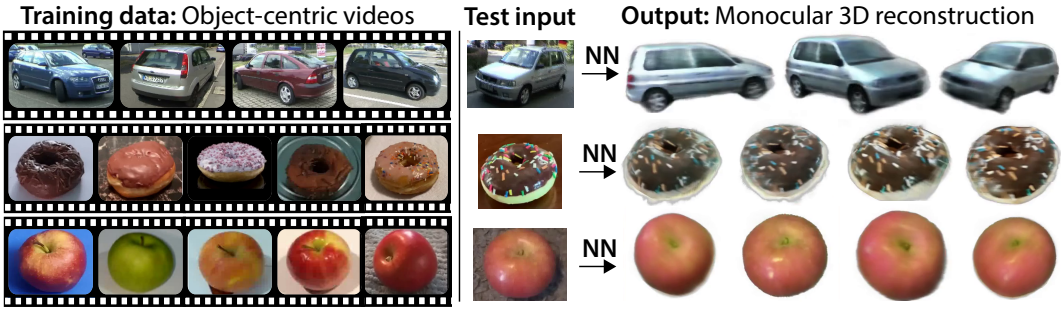


Figure 4.1: We present a novel deep architecture that contributes *Warp-conditioned Ray Embedding (WCE)* to reconstruct and render new views (right) of object categories from one or few input images (middle). Our model has learned automatically from videos of the objects (left) and works on difficult real data where competitor architectures fail to produce good results.

categories were collected and provided by collaborators.

Our second contribution is to assess current reconstruction technology on our new 'in the wild' data. For example, since each video provides several views of a single object with known camera parameters, it is suitable for an application of recent methods such as NeRF [54], and we find that learning *individual videos* works very well, as expected. However, we show that a direct application of such models to several videos of different but related objects is *much* harder. In fact, we experiment with related representations such as voxels and meshes and find that they also do not work well if applied naïvely to this task. This is true even though reconstructions are focused on a single object at a time — thus disregarding the background — suggesting that these architectures have a difficult time handling even relatively mild geometric variability.

Our final contribution is to propose a novel deep neural network architecture to better learn 3D object categories in such difficult conditions. We hypothesize that the main challenge in extending high-quality reconstruction techniques, that work well for single objects, to object categories is the difficulty of absorbing the geometric variability that comes with tackling many different objects together. An obvious but important source of variability is *viewpoint*: given only real images of different objects, it is not obvious how these should align in 3D space, and a lack of alignment adds to the variability that the model must cope with. We address this issue with a

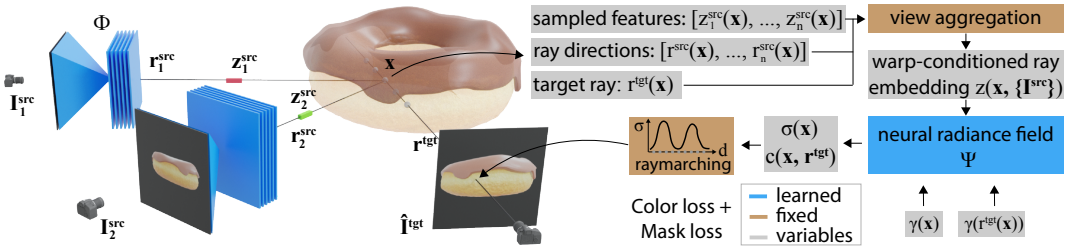


Figure 4.2: Our method takes as input an image and produces per pixel features using a U-Net Φ. We then shoot rays from a target view and retrieve per-pixel features from one or multiple source images. Once all spatial feature vectors are aggregated into a single feature vector (see Sec. 4.2.3 for more details), we combine them with their harmonic embeddings and pass them to an MLP yielding per location colours and opacities. Finally, we use differentiable raymarching to produce a rendered image.

novel idea of *Warp-Conditioned Ray Embeddings (WCE)*, a new neural rendering approach that is far less sensitive to inaccurate 3D alignment in the input data. Our method modifies previous differentiable ray marchers to pool information at variable locations in input views, conditioned on the 3D location of reconstructed points.

With this, we are able to train deep neural networks that, given as input a small number of images of new object instances in a given target category, can reconstruct them in 3D, including generating high-quality new views of the objects. Compared to existing state-of-the-art reconstruction techniques, our method achieves better reconstruction quality in challenging datasets of real-world objects.

4.2 Method

Overview. The goal of our method is to learn a model of a 3D object category from a dataset $\{\mathcal{V}^p\}_{p=1}^{N_{\text{video}}}$ of video sequences. Each video $\mathcal{V}^p = (I_t^p)_{0 \le t < T^p}$ consists of $T^p \in \mathbb{N}$ color frames $I_t^p \in \mathbb{R}^{3 \times H \times W}$. While we do not use any manual annotations for the videos, we do pre-process them using a Structure-from-Motion algorithm (COLMAP [183]). In this manner, for each video frame I_t^p, we obtain sequence-specific camera poses $g_t^p \in SE(3)$ and the camera instrinsics $K_t^p \in \mathbb{R}^{3 \times 3}$. We further obtain a segmentation mask $m_t^p \in R^{1 \times H \times W}$ of the given category using Mask-RCNN [184].

The model parametrizes the appearance and geometry of the object in each video

with an implicit surface map Ψ:

$$\Psi : \mathbb{R}^3 \times \mathbb{S}^2 \times \mathscr{Z} \to \mathbb{R}^3 \times \mathbb{R}_+ \quad \Psi(\mathbf{x}, \mathbf{r}, \mathbf{z}) = (\mathbf{c}, \sigma),$$

which labels each 3D scene point $\mathbf{x} \in \mathbb{R}^3$ and viewing direction $\mathbf{r} \in \mathbb{S}^2$ with an RGB triplet $\mathbf{c}(\mathbf{x}, \mathbf{r}, \mathbf{z}) \in \mathbb{R}^3$ and an occupancy value $\sigma(\mathbf{x}, \mathbf{z}) \in (0, 1]$ representing the opaqueness of the 3D space. Furthermore, the implicit function Ψ is conditioned on a latent code $\mathbf{z} \in \mathscr{Z}$ that captures the factors of variation of the object. By changing $\mathbf{z}$ we can adjust the occupancy field to represent shapes of different objects of a visual category. As described in Sec. 4.2.3, the design of the latent space $\mathscr{Z}$ is crucial for the success of the method.

While we use video sequences to train the model, at test time we would like to reconstruct any new object instance from a small number of images. To this end, we learn an encoder function

$$\Phi : \mathbb{R}^{3 \times H \times W \times N_{\text{src}}} \to \mathscr{Z},$$

that takes a number of input *source* images $\{I_1^{\text{src}}, \ldots, I_{N_{\text{src}}}^{\text{src}}\}$ of the new instance and produces the latent code $\mathbf{z} \in \mathscr{Z}$.

Given a known target view (different view than the source images) we render the implicit surface to form a colour image $\hat{I}^{\text{tgt}} \in \mathbb{R}^{3 \times H \times W}$ and minimize the discrepancy between the rendered $\hat{I}^{\text{tgt}}$ and the masked ground truth image I^{tgt}.

In the following, we describe the main building blocks of our method. The rendering step follows Emission-Absorption raymarching [185, 1, 54, 94] as detailed in Sec. 4.2.1. Sec. 4.2.2 describes the specifics of the surface function Ψ, and Sec. 4.2.3 introduces the main technical contribution — a novel Warp-Conditioned Ray Embedding that defines the image encoder Φ.

4.2.1 Implicit surface rendering

In order to render a target image $\hat{I}^{\text{tgt}}$, we emit a ray from the camera center through each pixel, assigning the colour of the ray's first 'intersection' with the surface to the respective pixel. Formally, let $\Omega = \{0, \ldots, W-1\} \times \{0, \ldots, H-1\}$ be an image

grid, $u \in \Omega$ the index of a pixel, and $Z \in \mathbb{R}_+$ a depth value. Following the ray from the camera center through u to depth $Z \geq 0$ results in the 3D point: $\bar{\mathbf{x}}(u,Z) = Z \cdot K^{-1}[u^\top \ 1]^\top$, where $K \in \mathbb{R}^{3\times 3}$ are the camera intrinsics. The camera's pose is given by an Euclidean transformation $g^{\text{tgt}} \in SE(3)$, where we use the convention that $\bar{\mathbf{x}} = g^{\text{tgt}}(\mathbf{x})$ maps points $\mathbf{x}$ expressed in the world reference frame to points $\bar{\mathbf{x}}$ in camera coordinates.

In order to determine the colour of a pixel $u \in \Omega$, we then 'shoot' a ray seeking the surface intersection. To do so, we sample points $\mathscr{X}_u = (\mathbf{x}(u,Z_i))_{i=0}^{N_Z+1}$ for depth values $Z_0 \leq \cdots \leq Z_{N_Z}$ obtaining their colors and occupancies:

$$(\mathbf{c}_i, \sigma_i) = \Psi(\mathbf{x}(u,Z_i), \mathbf{r}, \mathbf{z}), \quad i = 0,\ldots,N_Z. \tag{4.1}$$

The probability of the ray *not* intersecting the surface in the interval $(Z_{i+1}, Z_i]$ is set to $T_i = e^{-(Z_{i+1}-Z_i)\sigma_i(\mathbf{x}(u,Z_i),\mathbf{z})}$ (transmission probability). Summing over all possible intersections $Z_0,\ldots,Z_i$, the probability $p(Z = Z_i|u)$ of a ray terminating at depth Z_i is thus defined as:

$$p(Z = Z_i|u) = \left(\prod_{j=0}^{i-1} T_j\right)(1 - T_i), \ \ \hat{m}_u = 1 - \prod_{i=0}^{N_Z-1} T_i,$$

with the overall probability of intersection $\hat{m}_u$. Given the distributions of ray-termination probabilities $p(Z|u)$, the rendered color $\hat{\mathbf{c}}_u(\mathscr{X}_u, \mathbf{r}, \mathbf{z}) \in \mathbb{R}^3$ and opacity $\hat{\sigma}_u(\mathscr{X}_u, \mathbf{z}) \in \mathbb{R}$ are defined as an expectation over the outputs of the implicit function within the range $[0,\ldots,N_Z - 1]$:

$$\hat{\mathbf{c}}_u = \sum_{i=0}^{N_Z-1} p(Z = Z_i|u)\mathbf{c}_i, \ \ \hat{\sigma}_u = \sum_{i=0}^{N_Z-1} p(Z = Z_i|u)\sigma_i.$$

Since we are only interested in rendering the interior of the object, the colours $\mathbf{c}_u$ are softly masked with $\hat{m}_u$ leading to the final target image render $\hat{I}^{\text{tgt}} \in \mathbb{R}^{3\times H\times W}$:

$$\hat{I}^{\text{tgt}} = I(g^{\text{tgt}}, \mathbf{z}) = \hat{m} \odot \hat{\mathbf{c}}. \tag{4.2}$$

Note that the reconstruction depends on the target viewpoint g^{tgt} and the object code $\mathbf{z}$, which is viewpoint independent.

4.2.2 Neural implicit surface

Next, we detail the implicit surface function Ψ. Similar to previous methods [54, 186, 75], we exploit the representational power of deep neural networks and define Ψ as a deep multi-layer perceptron (MLP): $(\mathbf{c}, \sigma) = \Psi_{\text{nr}}(\mathbf{x}, \mathbf{r}, \mathbf{z})$. The network Ψ_{nr} follows a design similar to [54]. In particular, the world-coordinates $\mathbf{x}$ are preprocessed with the *harmonic encoding* $\gamma_{N_f^x}(\mathbf{x}) = [\sin(\mathbf{x}), \cos(\mathbf{x}), \ldots, \sin(2^{N_f^x}\mathbf{x}), \cos(2^{N_f^x}\mathbf{x})] \in \mathbb{R}^{2N_f^x}$ before being input to the first layer of the MLP. In order to enable modeling of viewpoint-dependent colour variations, we further use the harmonic encoding of the target ray direction $\gamma_{N_f^r}(\mathbf{r}^{\text{tgt}}(\mathbf{x})) \in \mathbb{R}^{2N_f^r}$ as input (see Fig. 4.2).

4.2.3 Warp-conditioned ray embedding

An important component of our method is the design of the latent code $\mathbf{z}$. A naïve solution is to first map a source image I^{src} to a D-dimensional vector $\mathbf{z}_{\text{CNN}} = \Phi_{\text{CNN}}(I^{\text{src}}) \in \mathbb{R}^D$ with a deep convolutional neural network Φ_{CNN}, followed by appending a copy of $\mathbf{z}_{\text{CNN}}$ to each positional embedding $\gamma(\mathbf{x})$ to form an input to the neural occupancy function Ψ_{nr}. This approach, successfully utilized in [94, 84] for synthetic datasets where the training shapes are approximately rigidly aligned, is however insufficient when facing more challenging in-the-wild scenarios.

To show why there is an issue here, recall that our inputs are *videos* $\mathcal{V}^p$ of different object instances, each consisting of a sequence $(I_t^p)_{0 \le t < T^p}$ of video frames, together with viewpoint transformations $g_t^p \in SE(3)$ recovered by SfM. Crucially, due to the global coordinate frame and scaling ambiguity of the SfM reconstructions [48], there is no relationship between the camera positions g^p and g^q reconstructed for two different videos $p \neq q$. Even two identical videos $\mathcal{V}^p = \mathcal{V}^q$, reconstructed using SfM from two different random initializations, will result in two different sets of cameras $(g_t^p)_{0 \le t < T^p}$, $(g_t^q = g^\star g_t^p)_{0 \le t < T^p}$, related by an unknown similarity transformation $g^\star \in S(3)$. Since the frames $I_t^p = I_t^q$ are identical, the reconstruction network Φ_{CNN} must assign to them identical codes: $\mathbf{z}_{\text{CNN},t} = \mathbf{z}_{\text{CNN,t}}^p = \Phi_{\text{CNN}}(I_t^p) =$

$\Phi_{\text{CNN}}(I_t^q) = \mathbf{z}_{\text{CNN,t}}^q$. Plugging this in Eq. 4.2, means that two identical frames are reconstructed from the same code $\mathbf{z}_{\text{CNN},t}$ but two different viewpoints $g_t^p \neq g_t^q$: $\hat{I}_t^p = I(g_t^p, \mathbf{z}_{\text{CNN},t}) = I(g_t^q, \mathbf{z}_{\text{CNN},t}) = \hat{I}_t^q$. While of course, we do not work with identical copies of the same videos, this extreme case demonstrates a fundamental issue with the naïve model, where different object instances must be reconstructed with respect to unrelated viewpoints.

We can partially tackle this issue by using a variant of [187] to approximately align the viewpoint of different video sequences before training (see supplemental).

Next, we introduce a more fundamental change to the model that also helps addressing this issue. The idea is to change the implicit surface (4.1)

$$\Psi_{\text{WCE}}(\mathbf{x}, \mathbf{z}(\mathbf{x})), \tag{4.3}$$

such that the code $\mathbf{z}$ is a *function of the queried ray point* $\mathbf{x}$ in world coordinates. Given a source image I_t^{src} with viewpoint g_t, the projection of this point in the image is: $u_t(\mathbf{x}) = \pi_t(\mathbf{x}) = \pi(Kg_t\mathbf{x})$ where π denotes the perspective projection operator $\mathbb{R}^3 \to \Omega$. In particular, if $\mathbf{x}$ is also a point on the surface of the object, then $u_t(\mathbf{x})$ is the image of the corresponding point in the source view I_t^{src}.

More specifically, we task a convolutional neural network Φ to map the image I_t^{src} to a feature field $\Phi(I_t^{\text{src}}) \in \mathbb{R}^{D\times H\times W}$ (see supplementary for details). In this way, for each pixel u_t in the source view, we obtain a corresponding embedding vector $\Phi(I_t)[u_t(\mathbf{x})]$ (using differentiable bilinear interpolation $[\cdot]$):

$$\mathbf{z}_t(\mathbf{x}) = \Phi(I_t)[\pi_t(\mathbf{x})] \in \mathbb{R}^D, \tag{4.4}$$

and call it **Warp-Conditioned Ray Embedding (WCE)**.

Intuitively, as shown in Fig. 4.2, by using Eq. 4.3Eq. 4.4 during ray marching, the implicit surface network Ψ_{WCE} can pool information from relevant 2D locations u_t in the source view I_t^{src}. Importantly, this occurs in a manner which is invariant to the global viewpoint ambiguity. In fact, if the geometry is now changed by the application of an arbitrary similarity transformation $g^\star$, then the

3D point changes as $\mathbf{x}' = g^\star \mathbf{x}$, but the viewpoint also changes as $g'_t = g_t (g^\star)^{-1}$, so that $g'_t \mathbf{x}' = g'_t (g^\star)^{-1} g^\star \mathbf{x} = g_t \mathbf{x}$ and the encoding of the points $\mathbf{x}$ and $\mathbf{x}'$ is the same: $\Phi(I_t)[\pi_t(\mathbf{x})] = \Phi(I_t)[\pi'_t(\mathbf{x}')]$ Finally, note that the network Eq. 4.3 combines two sources of information: (1) codes $\mathbf{z}(\mathbf{x})$ that capture the appearance of each point in a manner which is invariant from the global coordinate transforms; and (2) the absolute location of the 3D point $\mathbf{x}$ (internally encoded by using position-sensitive coding $\gamma(\mathbf{x})$). The combination of 1) and 2) above allows us to resolve misalignments by localizing the implicit surface equivariantly with changes of the global coordinates.

Multi-view aggregation. Having described WCE for a single source image we now extend to the more common case with multiple source images. For a set of source views $\{I_t^{\text{src}}\}_{t=1}^{N_{\text{src}}}$ with their warp-conditioned embeddings $\mathbf{z}_t^{\text{src}}(\mathbf{x})$, source rays $\mathbf{r}_t^{\text{src}}(\mathbf{x})$, and the target ray $\mathbf{r}^{\text{tgt}}(\mathbf{x})$ (see Fig. 4.2), we calculate the aggregate WCE $\mathbf{z}(\mathbf{x}, \{I_t^{\text{src}}\})$:

$$\mathbf{z}(\mathbf{x}, \{I_t^{\text{src}}\}) = \text{cat}\big(\mathbf{z}^\mu(\mathbf{x}, \{I_t^{\text{src}}\}), \mathbf{z}^\sigma(\mathbf{x}, \{I_t^{\text{src}}\}), \mathbf{z}_{\text{CNN}}(\{I_t^{\text{src}}\})\big),$$

as a concatenation (cat) of the angle-weighted mean and variance embedding $\mathbf{z}^\mu \in \mathbb{R}^D$ and $\mathbf{z}^\sigma \in \mathbb{R}_+$ respectively, and a plain average $\mathbf{z}_{\text{CNN}} = N_{\text{src}}^{-1} \sum_t \mathbf{z}_{\text{CNN},t}$ over global source embeddings $\mathbf{z}_{\text{CNN},t}$.

The mean $\mathbf{z}^\mu(\mathbf{x}, \{I_t^{\text{src}}\}) = \sum_{t=1}^{N_{\text{src}}} w_t(\mathbf{x}) \mathbf{z}_t^{src}(\mathbf{x})$ is a weighted average of the source embeddings $\mathbf{z}_t^{src}(\mathbf{x})$ with the weight $w_t(\mathbf{x})$ defined as

$$w_t(\mathbf{x}) = W(\mathbf{x})^{-1} (1 + \mathbf{r}_t^{\text{src}}(\mathbf{x}) \cdot \mathbf{r}^{\text{tgt}}(\mathbf{x})).$$

$W(\mathbf{x}) = \sum_{t=1}^{N_{\text{src}}} w_t(\mathbf{x})$ is a normalization constant ensuring the weights integrate to 1. This gives more weight to the source-view features that are imaged from a viewpoint which is closer to the target view. The variance embedding $\mathbf{z}^\sigma \in \mathbb{R}_+$ is defined analogously as an average over dimension-specific $w_t(\mathbf{x})$-weighted standard deviations of the source embedding set $\{\mathbf{z}_t^{\text{src}}(\mathbf{x})\}_{t=1}^{N_{\text{src}}}$.

4.2.4 Overall learning objective

For training, we optimize the loss $\mathscr{L} = \lambda \mathscr{L}_{\text{mask}} + \mathscr{L}_{\text{rgb}}$ where $\lambda = 0.05$. $\mathscr{L}_{\text{mask}}$ is defined as the binary cross-entropy between the rendered opacity and ground truth mask. For the appearance loss $\mathscr{L}_{\text{rgb}}$ we use the mean-squared error between the masked target view and our rendering.

4.3 Experiments

We discuss implementation details, data and evaluation protocols (Sec. 4.3.1) and assess our method and baselines on the tasks of novel-view synthesis and depth prediction.

Implementation details. As noted in Sec. 4.2.3, although WCE is in principle capable of dealing with the scene misalignments by itself, we found it beneficial to approximately "synchronize" the viewpoints of different videos in pre-processing, using a modified version of the method from [187]. First, we use the scene point clouds from SfM to register translation and scale by centering (subtracting the mean) and dividing by average per-dimension variance, resulting in adjusted viewpoints $\bar{g}_t$. We then proceed with training the rotation part of the viewpoint factorization branch of the VpDR network from [187], in order to align the rotational components of the viewpoints.

4.3.1 AMT Objects and other benchmarks

Figure 4.3: In order to study learning 3D object categories in the wild, we crowd-sourced a large collection of object-centric videos from Amazon Mechanical Turk. The top row shows frames from three example videos, the bottom two rows show SfM reconstructions of the videos together with tracked cameras.

One of our main contributions is to introduce the **AMT Objects** dataset, a large collection of object-centric videos that we collected (Fig. 4.3) using Amazon Mechanical Turk. The dataset contains 7 object categories from the MS COCO classes [188]: apple, sandwich, orange, donut, banana, carrot and hydrant. For each class, we ask Turkers to collect a video by looking 'around' a class instance, resulting in a turntable video. For reconstruction, we uniformly sampled 100 frames from each video, discarding any video where COLMAP pre-processing was unsuccessful. The dataset contains 169-457 videos per class. For each class, we randomly split videos into training and testing videos in an 8:1 ratio.

We also consider the **Freiburg Cars** [189], consisting of 45 training and 5 testing videos of various parked cars.

For every video, we define three disjoint sets of frames on which we either train or evaluate: (1) *train-train*, (2) *train-test* and (3) *test*. For each training video, we form the train-test set by randomly selecting 16 frames and a disjoint train-train set containing the complement of train-test. While the train-train frames are utilized for training, the train-test frames are never seen during training and only serve for evaluation. The evaluation on the test set is the most challenging since it is conducted with views of previously unseen object instances.

Evaluation protocol. Recall that, at test time, our network takes as input a certain number of source images I^{src} and reconstructs a target image $\hat{I}^{\text{tgt}}$ seen from a different viewpoint. We assess the view synthesis and depth reconstruction quality of this prediction. To this end, for each object category, we randomly extract a batch of 8 different images from the *train-test* and *test* respectively. To increase view variability we repeat this process 5 times for every object. For each batch one of the images is picked as a target image I^{tgt} and from the remaining images we individually select 1,3,5,7 images and perform the forward pass to generate $\hat{I}^{\text{tgt}}$ for each selection.

In order to assess the quality of view synthesis, we calculate the $\ell_1^{\mathbf{RGB}}$ error, between the target and predicted image. We also use the $\ell_1^{\mathbf{VGG}}$ perceptual metric, which computes the ℓ_1 distance between the two images encoded by means of the VGG-19 network [25] pretrained on ImageNet. For depth reconstruction, we compute the

$\ell_1^{\textbf{Depth}}$ distance between ground truth depth map (obtained from COLMAP SfM) and the predicted one in the target view. Finally, we report Intersection-over-Union (**IoU**) between the predicted object mask and the object mask obtained by Mask-RCNN in the target view.

4.3.2 Baselines

In this section, we detail the baselines we compare with. The first is **MLP**, corresponding to a naïve version of the latent global encoding $\mathbf{z}_{\text{CNN}}$ already discussed in Sec. 4.2.3. Here, the N^{src} source images $\{I_t^{src}\}_{t=1}^{N^{\text{src}}}$ are first independently mapped to embedding vectors $\{\mathbf{z}_t \in \mathbb{R}^{256}\}_{t=1}^{N^{\text{src}}}$ by a ResNet50 [190] encoder and subsequently averaged to form an encoding of the object $\mathbf{z}_{\text{CNN}} = \frac{1}{N^{\text{src}}} \sum_{t=1}^{N^{\text{src}}} \mathbf{z}_t$. A copy of $\mathbf{z}_{\text{CNN}}$ is then concatenated to each positional embedding $\gamma(\mathbf{x})$ of each target ray point $\mathbf{x}$. MLP renders with the EA ray marcher (Sec. 4.2.1).

The second baseline is **Voxel**, which closely resembles [86]. This uses the same encoding scheme as **MLP**, but differs by the fact that the object is represented by a voxel grid. Specifically, $\mathbf{z}_{\text{CNN}}$ is decoded with a series of 3D convolution-transpose layers to a 128^3 voxel grid containing RGB and opacity values. **Voxel** also renders with EA.

Next, **Voxel+MLP** is inspired by Neural Sparse Voxel fields [63] and marries NeRF [54] with voxel grids. As in **Voxel**, $\mathbf{z}_{\text{CNN}}$ is first 3D-deconvolved into a 128^3 volume of 32-dimensional features. Each target view ray point $\mathbf{x}$ is then described with a positional embedding $\gamma(\mathbf{x})$, and a latent feature $\mathbf{z}_{\text{g}}(\mathbf{x}) \in \mathbb{R}^{32}$ trilinearly sampled at the voxel grid location $\mathbf{x}$. The rest is the same as in **MLP**.

Finally, the **Mesh** baseline uses the soft-rasterization of [85] as implemented in PyTorch3D [191] with the top-k face accumulation. The scene encoding $\mathbf{z}_{\text{CNN}}$ is converted with a pair of linear layers to: (1) a set $\{v_i(\mathbf{z}) \in \mathbb{R}^3\}_{i=1}^{N_{\text{vertex}}}$ of 3D vertex locations of the object mesh, and (2) a 128×128 UV map of the texture mapped to the surface of the mesh, which is rendered in order to evaluate the reconstruction losses from Sec. 4.2.4. The mesh is initialized with an icosahedral sphere with 642 vertices.

Table 4.1: Novel-view synthesis on AMT Objects and Freiburg Cars. Each row evaluates either a baseline or **our** method. Results are reported for two perceptual metrics ℓ_1^{RGB}, ℓ_1^{VGG}, depth error ℓ_1^{Depth}, and intersection-over-union (IoU). For training we randomly selected between 1 and 7 source images. For testing we separately calculated the error metrics for 1, 3, 5 and 7 source images respectively and provide the average among those. For a more detailed evaluation we refer to the supplemental. Lower is better for $\ell_1^{\textbf{RGB}}$, $\ell_1^{\textbf{VGG}}$, and $\ell_1^{\textbf{Depth}}$, whereas higher is better for **IoU**. The best result is **bolded**.

	AMT								**Freiburg Cars**							
	Train-test				**Test**				**Train-test**				**Test**			
Method	$\ell_1^{\textbf{RGB}}$	$\ell_1^{\textbf{VGG}}$	**IoU**	$\ell_1^{\textbf{Depth}}$	$\ell_1^{\textbf{RGB}}$	$\ell_1^{\textbf{VGG}}$	**IoU**	$\ell_1^{\textbf{Depth}}$	$\ell_1^{\textbf{RGB}}$	$\ell_1^{\textbf{VGG}}$	**IoU**	$\ell_1^{\textbf{Depth}}$	$\ell_1^{\textbf{RGB}}$	$\ell_1^{\textbf{VGG}}$	**IoU**	$\ell_1^{\textbf{Depth}}$
Mesh	0.10	1.17	0.60	5.13	0.10	1.16	0.60	5.09	0.14	2.03	0.60	1.19	0.17	2.17	0.56	**1.06**
Voxel	0.06	1.05	0.78	2.14	0.09	1.13	0.66	3.07	0.05	1.58	0.89	0.59	0.16	2.05	0.51	2.18
Voxel+MLP	0.06	1.04	0.78	1.95	0.09	1.13	0.65	2.87	0.05	1.47	0.88	**0.48**	0.16	2.06	0.54	1.97
MLP	0.04	0.90	0.87	1.38	0.09	1.13	0.65	3.59	**0.04**	**1.39**	0.87	0.59	0.15	2.03	0.47	2.52
Ours	**0.03**	**0.86**	**0.88**	**1.31**	**0.05**	**0.93**	**0.83**	**1.90**	**0.04**	**1.39**	**0.90**	**0.48**	**0.12**	**1.89**	**0.62**	1.60

4.3.3 Quantitative Results

Tbl. 4.1 presents quantitative results on Freiburg Cars and the AMT Objects, respectively. In terms of all perceptual metrics ($\ell_1^{\textbf{RGB}}$, $\ell_1^{\textbf{VGG}}$) as well as depth and IoU, our method is on par with the MLP on the *train-test* split. On the *test* split, we outperform all other baselines in $\ell_1^{\textbf{RGB}}$, $\ell_1^{\textbf{VGG}}$ and IoU on all 7 classes of AMT Objects and Freiburg Cars. This indicates significantly better ability of our warp-conditioned embedding to generalize to previously unseen object instances.

We further find that our method is better at leveraging multiple source views $N_{\text{src}} > 1$, outperforming all baselines for the ℓ_1^{RGB} error, see Tbl. 4.2. When increasing the number of source images our method performance for all metrics improves whereas for all baselines it stays more or less constant. This further shows the effectiveness of the warp-conditioned embedding (WCE).

Table 4.2: We evaluate the impact of increasing the number of source views during test time for the ℓ_1^{RGB} metric. Target renders and the corresponding metrics are produced for 1, 3, 5 and 7 source images. The best result is **bolded** where lower is better.

		AMT								**Freiburg Cars**							
		Train-test				**Test**				**Train-test**				**Test**			
	Method	1	3	5	7	1	3	5	7	1	3	5	7	1	3	5	7
$\ell_1^{\textbf{RGB}}$	Mesh	.096	.096	.096	.096	.102	.102	.102	.102	.141	.141	.140	.140	.166	.166	.166	.166
	Voxel	.062	.061	.061	.061	.091	.091	.091	.091	.055	.055	.055	.054	.159	.159	.158	.158
	Voxel+MLP	.059	.059	.058	.059	.090	.090	.090	.090	.045	.045	.045	.045	.158	.157	.158	.157
	MLP	**.037**	.036	.036	.036	.088	.088	.088	.088	**.041**	**.041**	**.041**	.041	.152	.152	.152	.152
	Ours	.038	**.032**	**.031**	**.030**	**.058**	**.046**	**.043**	**.042**	.046	**.041**	**.041**	**.040**	**.130**	**.120**	**.115**	**.114**

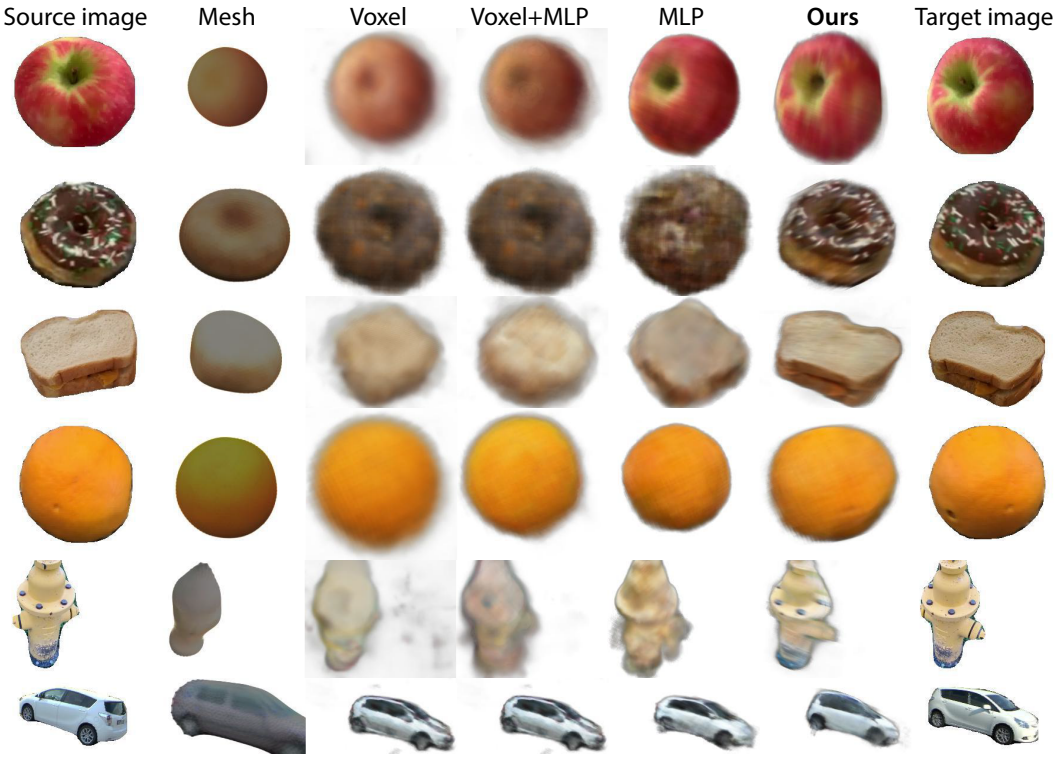


Figure 4.4: Monocular reconstruction on Freiburg Cars and AMT Objects. In each row, a single source image (1st column) is processed by one of the evaluated methods (Mesh, Voxel, MLP+Voxel, MLP, **Ours** - columns 2 to 6) to generate a prescribed target view (last column). We show results on the test split.

Regarding depth reconstruction (ℓ_1^{Depth}), our method outperforms all alternatives on all datasets except the test split of Freiburg Cars, where we are 2nd after Mesh. Here, we note that ℓ_1^{Depth} is only an approximate measure because: 1) the predicted depth is compared to the COLMAP-MVS estimate of depth [50], which tends to be noisy and; 2) the scale ambiguity in SfM reconstructions that supervise learning leads to a significantly unconstrained problem of estimating the scale of a testing scene given a small number of source views, which is challenging to resolve for any method.

4.3.4 Qualitative Results

Fig. 4.4 provides qualitative comparisons for monocular novel-view synthesis. It shows that our method produces significantly more detailed novel views, probably due to its ability to retrieve spatial encodings from the given source view. Fig. 4.5 further demonstrates the reconstruction improvement when multiple source views $N^{\text{src}} > 1$ are available.

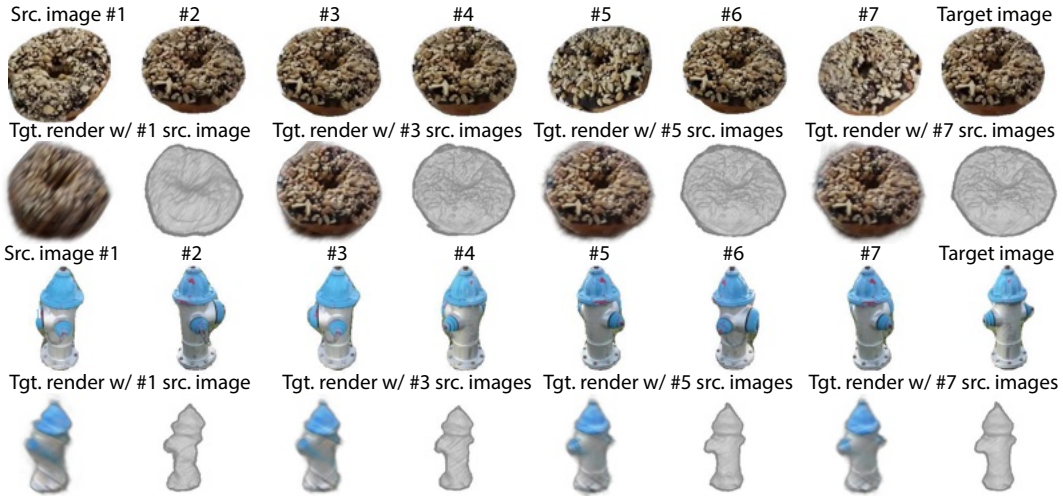


Figure 4.5: Reconstruction with multiple source views. For each object, the top row shows all available source images (columns 1-7) for a given target image (top right). The bottom row contains results conditioned on 1, 3, 5 or 7 source images. In addition to the rendered new RGB views we also provide shaded surface renderings.

4.4 Discussion and conclusions

Limitations. Even though our method outperforms baselines on the vast majority of metrics and datasets, there are still several limitations. First, the execution of the deep MLP at every 3D ray-location in a rendered frame is relatively slow (depending on the number of source views rendering takes between 3 and 8 sec for a 128×256 image on average), which makes a real-time deployment challenging. Secondly, due to our template-free approach, the object silhouettes can be blurry. Lastly, despite no manual labelling being necessary, our method still relies on segmentation masks that were automatically generated with Mask-RCNN.

Conclusions. In this chapter, we have presented a method that is able to reconstruct category-specific 3D shape and appearance from videos of object categories in the wild alone, without requiring manual annotations. We demonstrated that our main contribution, Warp-Conditioned Ray Embedding, can successfully deal with the inherent ambiguities present in the video SfM reconstructions that provide our supervisory signal, outperforming alternatives on a novel dataset of crowd-sourced object videos. Future work could include decomposition of shape, appearance and lighting allowing for more control over the rendered images.

Chapter 5

Learning a Neural 3D Texture Space from 2D Exemplars

In the previous chapters, we have seen two methods that given a single or a few images of an object perform 3D reconstruction of shape and appearance. In the next part of this thesis, a different aspect of appearance modeling is addressed: texture synthesis.

5.1 Overview

Textures are stochastic variations of attributes over 2D or 3D space with applications in both image understanding and synthesis. This chapter suggests a generative model of natural textures. Previous texture models either capture a single exemplar (e. g., wood) alone or address the non-stochastic (stationary) variation of appearance across space: Which location on a chair should have a wood colour? Which should be cloth? Which metal? Our work combines these two complementary views.

Requirements We design the family of methods with several requirements in mind: completeness, generativeness, compactness, interpolation, infinite domains, diversity, infinite zoom, and high speed.

A space of textures is *complete* if every natural texture has a compact code $\mathbf{z}$ in that embedding. To be *generative*, every texture code should map to a useful texture. This is important for an intuitive design where a user manipulates the texture code and expects the outcome to be a texture. *Compactness* is achieved if codes are low-

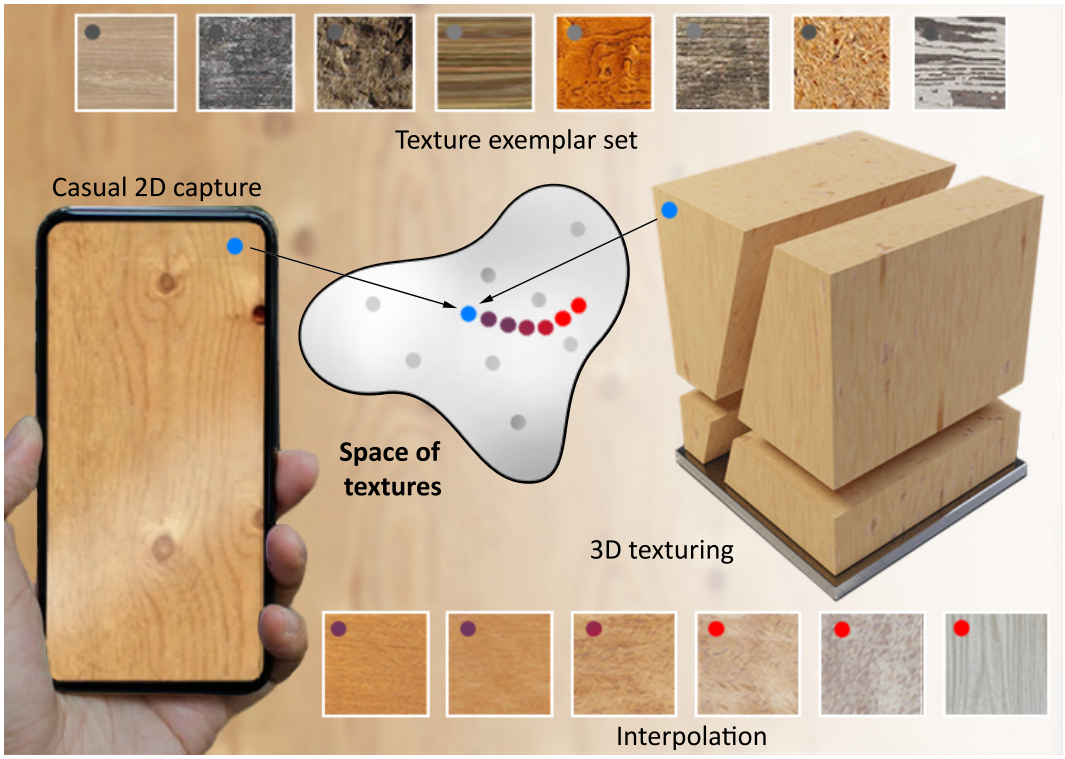


Figure 5.1: Our approach allows casually-captured 2D textures (blue) to be mapped to latent texture codes and support interpolation (blue-to-red), projection, or synthesis of volumetric textures.

dimensional. We also demand the method to provide *interpolation*: texture generated at coordinates between $\mathbf{z}_1$ and $\mathbf{z}_2$ should also be valid. This is important for design or when storing texture codes into a (low-resolution) 2D image, 3D volume or at mesh vertices with the desire to interpolate. The first four points are typical for generative modelling; achieving them jointly while meeting more texture-specific requirements (stochasticity, efficiency) is our key contribution.

First, we want to support *infinite domains*: Holding the texture code $\mathbf{e}$ fixed, we want to be able to query this texture so that a patch around any position $\mathbf{x}$ has the statistics of the exemplar. This is important for querying textures in graphics applications for extended virtual worlds, i. e., grass on a football field where it extends the size of the texture.

Second, for visual fidelity, the statistics under which textures are *similar* to the exemplar. The Gram matrix of VGG activations is one established metric for this similarity [124].

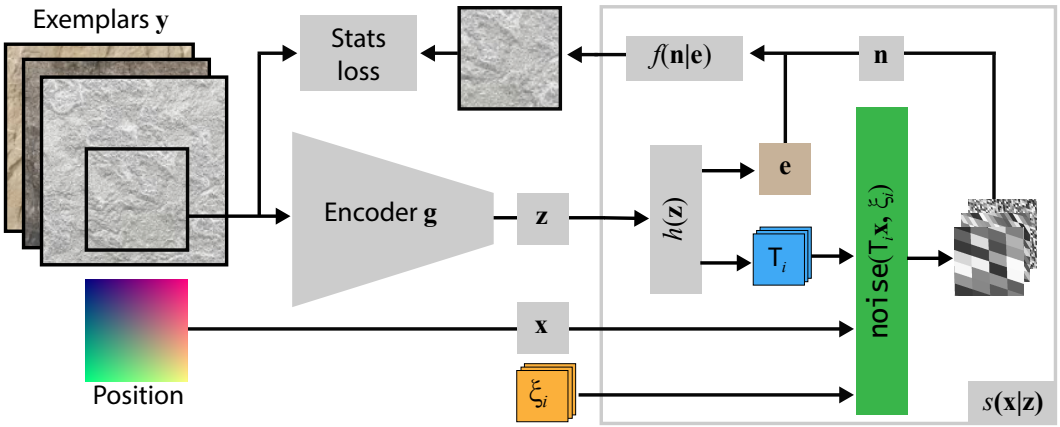


Figure 5.2: Overview of our approach as explained in Sec. 5.1.

Third, *infinite zoom* means each texture should have variations on a wide range of scales and not be limited to any fixed resolution that can be held in memory. This is required to zoom into details of geometry and appreciate the fine variation such as wood grains, etc. In practice, we are limited by the frequency content of the exemplars we train on, but the method should not impose any limitations across scales.

Fourth and finally, our aim is *computational efficiency*: the texture needs to be queryable without requiring prohibitive amounts of memory or time, in any dimension. Ideally, it would be constant in both and parallel. This rules out simple CNNs, that do not scale favourably in memory consumption to 3D.

5.2 Method

Our approach has two steps. The first embeds the exemplar into a latent space using an *encoder*. The second provides *sampling* at any position by reading noise fields at that position and combining them using a learned mapping to match the exemplar statistics. We now detail both steps.

Encoder The encoder g maps a 2D texture exemplar image $\mathbf{y}$ to a latent texture code $\mathbf{z} = g(\mathbf{y})$. We use a convolutional neural network to encode the high number of exemplar pixels into a compact latent texture code $\mathbf{z}$.

Sampler Sampling $s(\mathbf{x}|\mathbf{z})$ of a texture with code $\mathbf{z}$ at individual 2D or 3D positions $\mathbf{x}$ has two steps: a *translator* and a *decoder*, which are both described next.

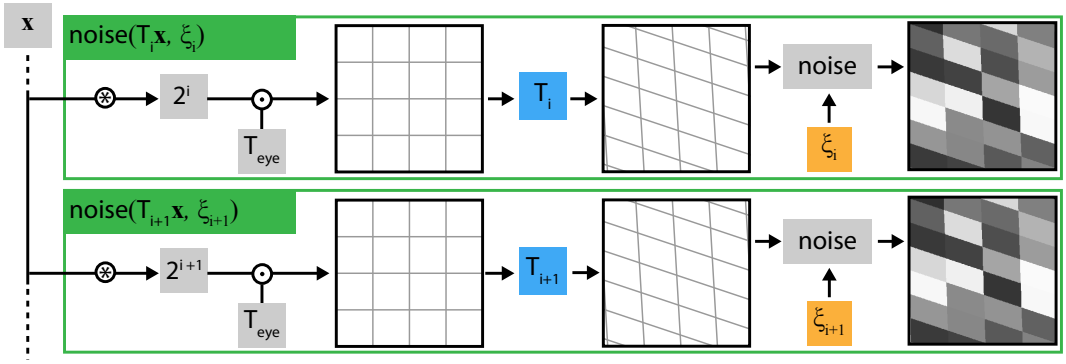


Figure 5.3: Noise field for different octaves and transformations T.

Decoder Our key idea is to prevent the decoder $f(\mathbf{n}|\mathbf{e})$ to access the position $\mathbf{x}$ and to use a vector of noise values $\mathbf{n}$ instead. Each $n_i = \texttt{noise}(\mathsf{T}_i 2^{i-1}\mathbf{x}|\xi_i)$ is read at different linear transformations $\mathsf{T}_i 2^{i-1}\mathbf{x}$ of that position $\mathbf{x}$ from random fields with different seeds ξ_i. The random field $\texttt{noise}(\mathbf{x}|\xi_i)$ is implemented as an infinite, single-channel 2D or 3D function that has the same random value for all continuous coordinates $\mathbf{x}$ in each integer lattice cell for one seed ξ_i. The factors of 2^{i-1} initialize the decoder to behave similarly to Perlins's octaves for identity T_i. Applying $\mathsf{T}_i 2^{i-1}$ to $\mathbf{x}$ is similar to Spatial Transformer Networks [192]. (Fig. 5.3).

These noise values are combined with the extended texture code $\mathbf{e}$ in a learned way. It is the task of the translator, explained next, to control, given the exemplar, how noise is transformed and to generate an extended texture code.

Translator The translator $h(\mathbf{z}) = \{\mathbf{e}, \mathsf{T}\}$ maps the texture code $\mathbf{z}$ to a tuple of parameters required by the decoder: the vector of transformation matrices T and an extended texture code vector $\mathbf{e}$. The matrices T are used to transform the coordinates before reading the noise as explained before. The extended texture parameter code $\mathbf{e}$ is less compact than the texture code $\mathbf{z}$, but allows the sampler to execute more effectively, i. e., do not repeat computations required for different $\mathbf{x}$ as they are redundant for the same z.

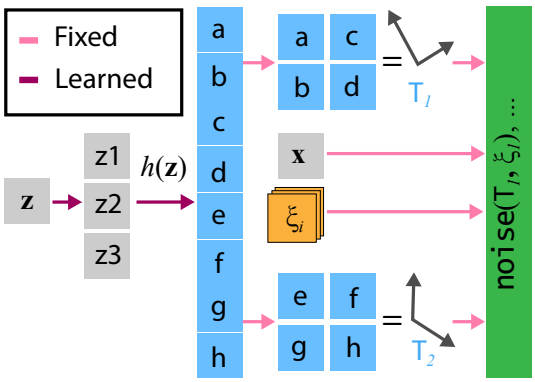


Figure 5.4: Translator.

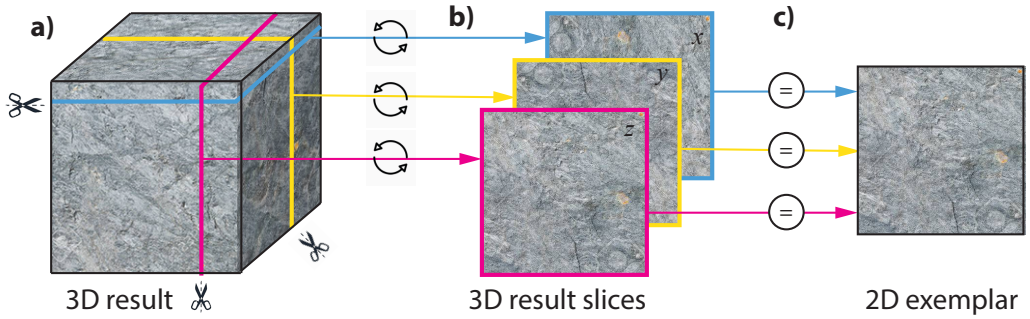


Figure 5.5: Sliced loss for learning 3D procedural textures from 2D exemplars: Our method, as it is non-convolutional, can sample the 3D texture (a) at arbitrary 3D positions. This enables to also sample arbitrary 2D slices (b). For learning, this allows us to simply slice 3D space along the three major axes (red, yellow, blue) and ask each slice to have the same VGG statistics as the exemplar (c).

See Fig. 5.4 where for example two 2×2 transformation matrices with 8 DOF are parameterized by three parameters.

Training For training, the encoder is fed with a random 128×128 patch P_e of a random exemplar $\mathbf{y}$, followed by the sampler evaluating a regular grid of 128×128 points $\mathbf{x}$ in random 2D slices of the target domain to produce a "slice" image P_s (Fig. 5.5). The seed ξ is held constant per train step, as one lattice cell will map to multiple pixels, and the decoder f relies on these being consistent. During inference changing the seed ξ and keeping the texture code $\mathbf{e}$ will yield diverse textures.

The loss is the $\mathscr{L}_2$ distance of Gram matrix of VGG feature activations [124, 127, 128, 126, 166] of the patches P_e and P_s.

If the source and target domain are the same (synthesizing 2D textures from 2D exemplars) the slicing operation is the identity. However, it also allows for the important condition in which the target domain has more dimensions than the source domain, such as learning 3D from 2D exemplars.

Spaces-of Our method can be used to either fit a *single* exemplar or an entire *space* of textures. In the single mode, we directly optimize for the trainable parameters $\theta = \{\theta_\mathrm{d}\}$ of the decoder. When learning the entire space of textures, the full cascade of encoder g, translator h and sampler s parameters are trained, i. e., $\theta = \{\theta_\mathrm{g}, \theta_\mathrm{h}, \theta_\mathrm{d}\}$ jointly.

5.3 Learning stochastic space coloring

Here we will introduce different implementations of samplers $s\colon \mathbb{R}^n \to \mathbb{R}^3$ which "colour" 2D or 3D space at position $\mathbf{x}$. We discuss the pros and cons with respect to the requirements from the introduction, ultimately leading to our approach.

Perlin noise is a simple and effective method to generate natural textures in 2D or 3D [117], defined as

$$s(\mathbf{x}|\mathbf{z}) = \sum_{i=1}^{m} \texttt{noise}(2^{i-1}\mathbf{x}, \xi_i) \otimes w_i, \tag{5.1}$$

where $h(\mathbf{z}) = \{w_1, w_2, \ldots\}$ are the RGB weights for m different noise functions $\texttt{noise}_i$ which return bilinearly-sampled RGB values from an integer grid. $\otimes$ is channel-wise multiplication. Here, $\mathbf{e}$ is a list of all linear per-layer RGB weights e. g., an 8×3 vector for the $m = 8$ octaves we use. This is a simple latent code, but we will see increasingly complex ones later. Also, our encoder g is designed such that it can cater to all decoders, even Perlin noise i. e., we can also create a space of textures with a Perlin noise back-end.

Coordinates $\mathbf{x}$ are multiplied by factors of two (octaves), so with increasing i, increasingly smooth noises are combined. This is motivated well in the spectra of natural signals [116, 117], but also limiting. Perlin's linear scaling allows the noise to have different colours, yet no linear operation can reshape a distribution to match a target. Our work seeks to overcome these two limitations but tries to retain the desirable properties of Perlin noise: simplicity and computational efficiency as well as generalization to 3D.

Transformed Perlin relaxes the scaling by powers of two

$$s(\mathbf{x}|\mathbf{z}) = \sum_{i=1}^{m} \texttt{noise}(\mathsf{T}_i 2^{i-1}\mathbf{x}, \xi_i) \otimes w_i \tag{5.2}$$

by allowing each noise i to be independently scaled by its own transformation matrix T_i since $h(\mathbf{z}) = \{w_1, \mathsf{T}_1, w_2, \mathsf{T}_2, \ldots\}$. Please note, that the choice of noise frequency is now achieved by scaling the coordinates reading the noise. This allows us to make use of anisotropic scaling for elongated structures, different orientations or multiple

random inputs at the same scale.

CNN utilizes the same encoder g as our approach to generate a texture code that is fed in combination with noise to a convolutional decoder similar to [128].

$$s(\mathbf{x}|\mathbf{z}) = \texttt{cnn}(\mathbf{x}|\mathbf{e}, \texttt{noise}(\xi)) \tag{5.3}$$

The CNN is conditioned on $\mathbf{e}$ without additional translation. Their visual quality is stunning, CNNs are powerful and the loss is able to capture perceptually important texture features, hence CNNs are a target to chase for us in 2D in terms of quality. However, there are two main limitations of this approach we seek to lift: efficiency and diversity.

CNNs do not scale well to 3D in high resolutions. To compute intermediate features at $\mathbf{x}$, they need to have access to neighbours. While this is effective and output-sensitive in 2D, it is not in 3D: we need results for 2D surfaces embedded in 3D, and do so in spatial high resolution (say 1024×1024), but this requires CNNs to compute a full 3D volume with the same order of pixels. While in 2D partial outputs can be achieved with sliding windows, it is less clear how to slide a window in 3D, such that it covers all points required to cover all 3D points that are part of the visible surface.

The second issue is diversity: CNNs are great for producing a re-synthesis of the input exemplar, but it has not been demonstrated that changing the seed ξ will lead to variation in the output in most classic works [126, 127] and in classic style transfer [124] diversity is eventually introduced due to the randomness in SGD. Recent work by Ulyanov and colleagues [128] explicitly incentivizes diversity in the loss. The main idea is to increase the pixel variance inside all exemplars produced in one batch. Regrettably, this often is achieved by merely shifting the same one exemplar slightly spatially or introducing random brightness fluctuations.

MLP maps a 3D coordinate to appearance:

$$s(\mathbf{x}|\mathbf{z}) = \texttt{mlp}(\mathbf{x}|\mathbf{e}) \tag{5.4}$$

where $h(\mathbf{z}) = \mathbf{e}$. Texture-fields [131] have used this approach to produce what they call "texture", detailed and high-quality appearance decoration of 3D surfaces, but what was probably not intended is to produce diversity or any stochastic results. At least, there is no parameter that introduces any randomness, so all results are identical. We took inspiration from their work, as it makes use of 3D point operations, that do not require accessing any neighbours and no intermediate storage for features in any dimensions, including 3D. It hence reduces bandwidth compared to CNN and is perfectly data-parallel and scalable. The only aspect missing to make it our colourization operator, required to create a space and evolve from 2D exemplars to 3D textures, is stochasticity.

Ours combines the noise from transformed Perlin for stochasticity, the losses used in style and texture synthesis CNNs for quality as well as the point operations in MLPs for efficiency as follows:

$$s(\mathbf{x}|\mathbf{z}) = f(\texttt{noise}(\mathsf{T}_1 2^0 \mathbf{x}, \xi_1), \ldots, \texttt{noise}(\mathsf{T}_m 2^{m-1} \mathbf{x}, \xi_m)|\mathbf{e}) \tag{5.5}$$

Different from MLPs that take the coordinate $\mathbf{x}$ as input, the position itself is hidden. Instead of position, we take multiple copies of spatially smooth noise $\texttt{noise}(\mathbf{x})$ as input, with explicit control of how the noise is aligned in space expressed by the transformations T. Hence, the MLP requires to map the entire distribution of noise values such that it suits the loss, resulting in build-in diversity. We chose number of octaves m to be 8, i. e., the transformation matrices $\mathsf{T}_1, \ldots, \mathsf{T}_m$ require $8 \times 4 = 32$ values in 2D. The texture code size $\mathbf{e}$ is 64 and the compact code $\mathbf{z}$ is 8. The decoder f consists of four stacked linear layers, with 128 units each followed by ReLUs. The last layer is 3-valued RGB.

Non-stochastic ablation seeks to investigate what happens if we do not limit our approach to random variables, but also provide access to deterministic information $\mathbf{x}$:

$$s(\mathbf{x}|\mathbf{z}) = f(\mathbf{x}, \texttt{noise}(2^0 \mathbf{x}, \xi_1), \ldots, \texttt{noise}(2^{m-1} \mathbf{x}, \xi_m)|\mathbf{e}) \tag{5.6}$$

is the same as MLP, but with access to noise. We will see that this effectively removes diversity.

Non-transformed ablation evaluates if our method were to read only from multi-scale noise without control over how it is transformed. Its definition

$$s(\mathbf{x}|\mathbf{z}) = f(\texttt{noise}(2^0\mathbf{x}, \xi_1), \ldots, \texttt{noise}(2^{m-1}\mathbf{x}, \xi_m)|\mathbf{e}) \tag{5.7}$$

5.4 Evaluation

Our evaluation covers qualitative (Sec. 5.4.3) and quantitative (Sec. 3.4.5) aspects as well as a user study (Sec. 5.4.4). We further discuss and compare the capabilities of our method with respect to the requirements introduced in Sec. 5.1.

5.4.1 Protocol

We suggest a data set that for which we explore the relation of different methods, according to different metrics to quantify texture similarity and diversity.

Data set Our data set contains four classes (WOOD, MARBLE, GRASS and RUST) of 2D textures, acquired from internet image sources. Each class contains 100 images.

Methods We compare eight different methods that are competitors, ablations and ours. As five *competitors* we study variants of Perlin noise, CNNs and MLPs. `perlin` implements Perlin noise (Eq. 5.1, [117]) and `perlinT` our variant extending it by a linear transformation (Eq. 5.2). Next, `cnn` is a classic TextureNet [126] and `cnnD` the extension to incentivise diversity ([128], Eq. 5.3). `mlp` uses an MLP following Eq. 5.4.

We study three *ablations*. First, we compare to `oursP` which is our method, but with the absolute position as input and no transform. Second, `oursNoT` omits the absolute position as input and transformation but still uses Perlin's octaves (Eq. 5.7). The final method is `ours` method (Eq. 5.5).

Metrics We evaluate methods with respect to three metrics: similarity and diversity and a joint measure, success.

Similarity is high, if the result produced has the same statistics as the exemplar in

terms of L2 differences of VGG Gram matrices. This is identical to the loss used. The similarity is measured on a single exemplar.

Diversity is not part of the loss but can be measured on a set of exemplars produced by a method. We measure diversity by looking at the VGG differences between all pairs of results in a set produced for a different random seed. Note, that this does not utilize any reference. Diversity is maximized by generating random VGG responses, yet without similarity.

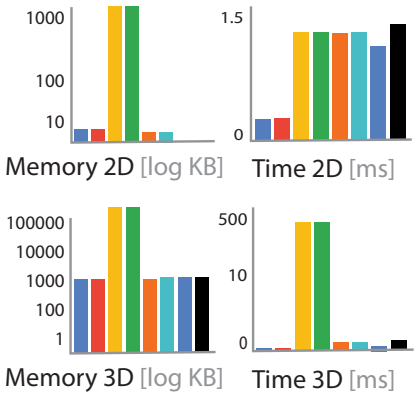
Success of the entire method is measured as the product of diversity and the maximum style error minus the style error. We apply this metric, as it combines similarity and diversity which are conflicting goals we jointly want to maximize.

Memory and speed are measured at a resolution of 128 pixels/voxels on an Nvidia Titan Xp.

5.4.2 Quantitative results

Table 5.1: Efficiency in terms of compute time and memory usage in 2D and 3D **(columns)** for different methods **(rows)**.

Method	Time		Memory	
	2D	3D	2D	3D
`perlin` •	0.18 ms	0.18 ms	65 k	16 M
`perlinT` •	0.25 ms	0.25 ms	65 k	16 M
`cnn` •	1.45 ms	551.59 ms	8,000 k	646 M
`cnnD` •	1.45 ms	551.59 ms	8,000 k	646 M
`mlp` •	1.43 ms	1.43 ms	65 k	16 M
`oursP` •	1.44 ms	1.44 ms	65 k	16 M
`oursNoT` •	1.24 ms	1.24 ms	65 k	16 M
`ours` •	1.55 ms	1.50 ms	65 k	16 M



Efficiency We first look at computational efficiency in Tbl. 5.1. We see that our method shares the speed and memory efficiency with Perlin noise and MLPs / Texture Fields [131]. Using a CNN [126, 128] to generate 3D textures as volumes is not practical in terms of memory, even at a modest resolution. Ours scales linearly with pixel resolution as an MLP is a point-estimate in any dimension that does not require any memory other than its output. A CNN has to store the internal activations of all layers in memory for information exchange between neighbours.

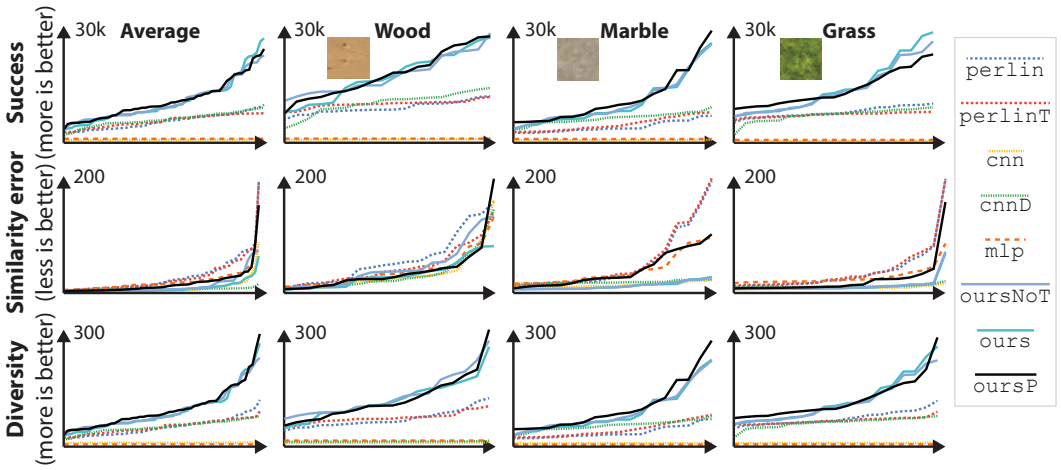


Figure 5.6: Quantitative evaluation. Each plot shows the histogram of a quantity (from top to bottom: success, style error and diversity) for different data sets (from left to right: all space together, WOOD, MARBLE, GRASS). For a discussion, see the last paragraph in Sec. 5.4.2.

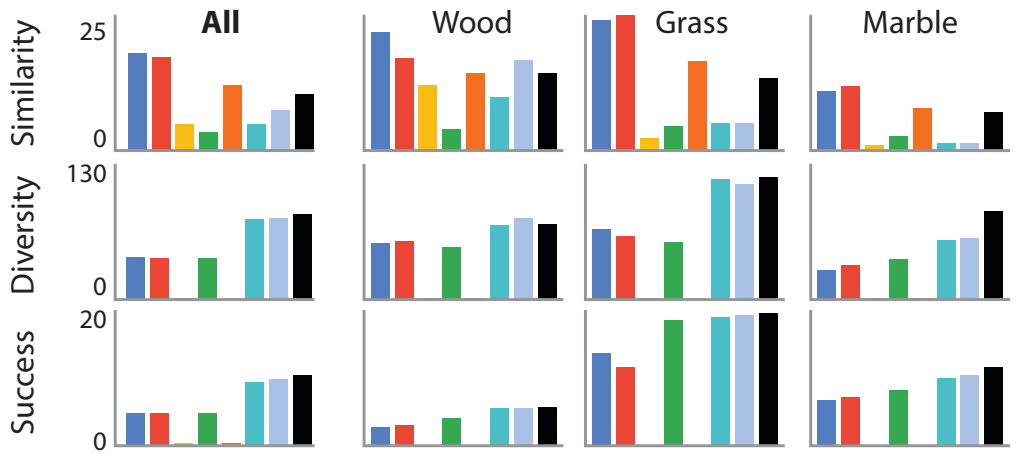
Fidelity Fig. 5.6 and Tbl. 5.2 summarize similarity, diversity and success of all methods in numbers. `ours` method (black) comes best in diversity and success on average across all sets (first column in Tbl. 5.2 and top first plot in Fig. 5.6). `cnn` (yellow) and `cnnd` (green) have better similarity than any of our methods. However, no other method combines similarity with diversity as well as ours. This is visible from the overall leading performance in the final measure, success. This is a substantial achievement, as maximizing for only one goal is trivial: an `identity` method has zero similarity error while a `random` method has infinite diversity.

When looking at the similarity, we see that both a `cnn` and its diverse variant `cnnD` can perform similarly. Perlin noise produces the largest error. In particular, `perlinT` has a large error, indicating it is not sufficient to merely add a transform. Similarly, `mlp` alone cannot solve the task, as it has no access to noise and needs to fit exactly, which is doable for single exemplars, but impossible for a space. `oursNoT` has error similar to `ours`, but less diversity.

When looking at diversity, it is clear that both `cnn` and `mlp` have no diversity as they either do not have the right loss to incentivize it or have no input to generate it. `perlin` and `perlinT` both create some level of diversity, which is not surprising as they are simple remappings of random numbers. However, they do not manage to

Table 5.2: Similarity and diversity for methods on different textures.

Method	ALL			WOOD			GRASS			MARBLE		
	Sim	Div	Suc	Sim	Div	Suc	Sim	Div	Suc	Sim	Div	Suc
`perlin` •	20.6	48.0	7.0	23.8	37.9	4.9	24.6	72.8	18.1	13.3	31.8	7.84
`perlinT` •	19.6	48.2	7.2	18.4	39.6	5.02	25.9	65.6	13.8	14.2	38.4	8.03
`cnn` •	5.4	0.5	7.5	13.4	0.5	0.07	**1.9**	0.5	0.14	**1.1**	0.3	0.08
`cnnD` •	**3.9**	48.2	7.75	**3.9**	35.2	5.19	4.8	59.2	20.9	3.6	48.8	8.5
`mlp` •	14.1	0.0	7.98	15.7	0.0	0.0	16.7	0.0	0.0	9.6	0.0	0.0
`oursP` •	5.4	93.4	8.23	9.7	67.4	5.33	4.8	126	21.5	1.8	84.5	9.0
`oursNoT` •	8.4	94.5	8.54	18.3	**74.7**	5.40	5.1	120	21.7	1.9	87.0	9.3
`ours` •	12.1	**99.7**	**8.82**	13.3	72.5	**5.48**	13.6	**127**	**22.1**	9.4	**98.2**	**9.6**



span the full VGG space, which only `ours` and its ablations can do.

Generating 3D textures from the exemplars in Fig. 5.7, we find that our diversity and similarity are 44.5 and 1.48, which compares favorable to Perlin 3D Noise at 14.9 and 7.11.

5.4.3 Qualitative results

Visual examples from the quantitative evaluation on a single exemplar for different methods can be seen in Fig. 5.7. We see that some methods have diversity when the seed is changed (rows one vs. two and three vs. four) and some do not. Diversity is clear for Perlin and its variant, CNNs with a diversity term and our approach. No diversity is found for MLPs and CNNs. We also note, that CNNs with diversity produce typically shifted copies of the same exemplar, so their diversity is over-estimated by the metric.

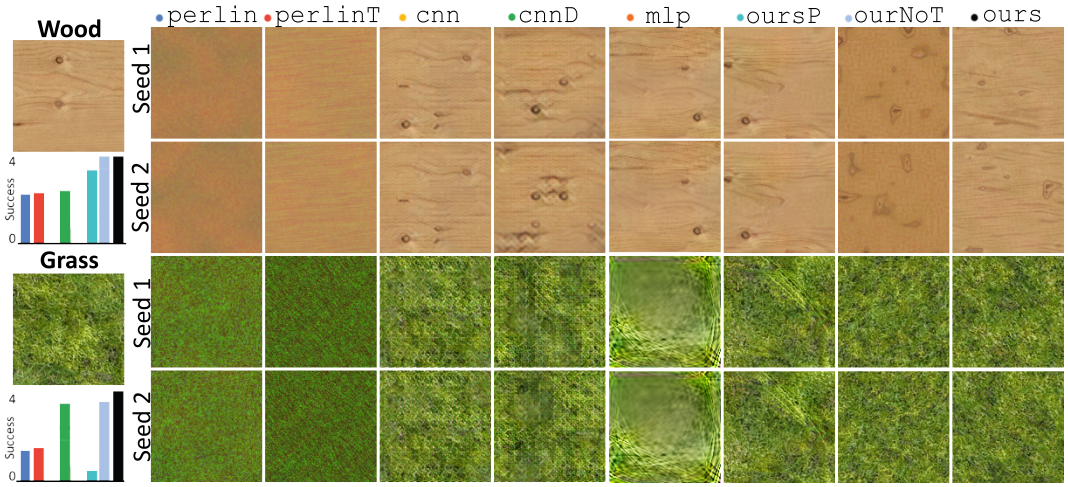


Figure 5.7: Different methods and the exemplar **(columns)**, as defined in Sec. 5.4.2, applied to different exemplars **(rows)**. Each row shows, arranged vertically, two re-synthesises with different seeds. Please see the text for discussion.

Fig. 5.8 shows a stripe re-synthesized from a single exemplar. We note that the pattern captures the statistics, but does not repeat.

Figure 5.8: Stripes of re-synthesized textures from exemplars on the left. See the supplemental for more examples.

Our system can construct textures and spaces of textures in 3D from 2D exemplars alone. This is shown in Fig. 5.9. We first notice that the textures have been transferred to 3D faithfully, inheriting all the benefits of procedural textures in image synthesis. We can now take any shape without a texture parametrization and, by simply running the NN at each pixel's 3D coordinate, produce a colour. We compare to a 2D

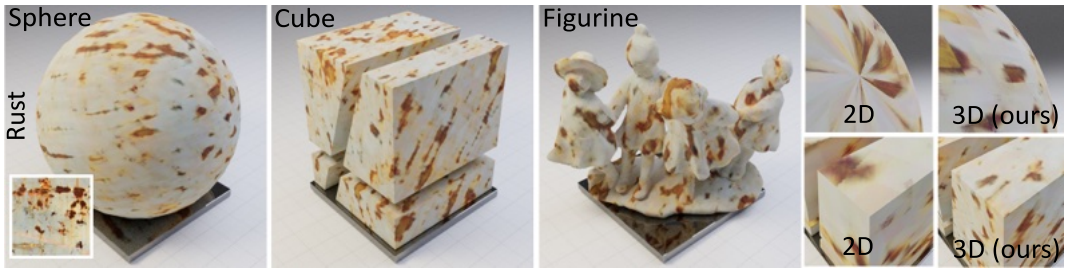


Figure 5.9: 3D texturing of different 3D shapes. Insets (right) compare ours to 2D texturing. See supplemental for 3D spin.

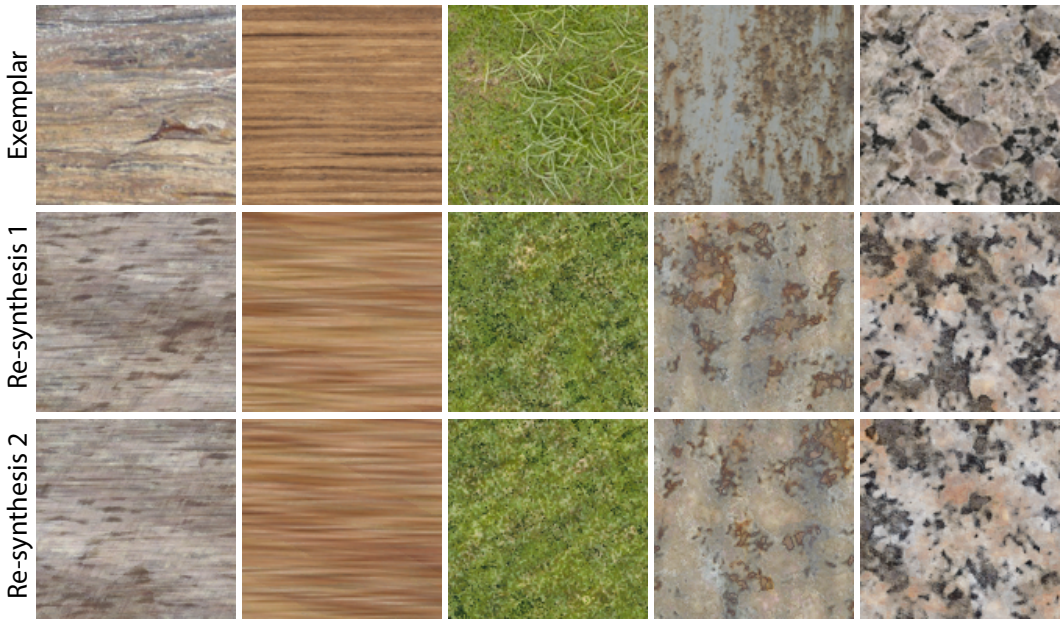


Figure 5.10: Our reconstruction of WOOD, GRASS, RUST, and MARBLE. The first row shows different input exemplars. The second and third rows show our reconstruction with different seeds.

approach by loading the objects in Blender and applying its state-of-the-art UV mapping approach [193]. Inevitably, a sphere will have discontinuities and poles that can not be resolved in 2D, which are no issue to our 3D approach while both take the same 2D as input.

Fig. 5.10 documents the ability to reproduce the entire space. We mapped exemplars unobserved at training time to texture codes, from which we reconstruct them, in 2D. We find that our approach reproduces the exemplars faithfully, albeit totally different on the pixel level.

Our method does not work on an explicit pixel grid but rather a continuous function, which allows zooming into arbitrary fine details as depicted in Fig. 5.11, which compares favorable to cubic upsampling. This is particularly useful in the 3D domain, where storing a complete volume to span multiple levels of detail requires prohibitive amounts of memory, while ours is output-sensitive.

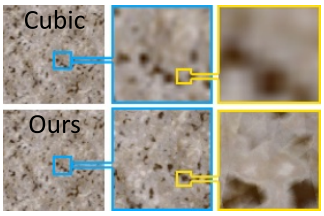


Figure 5.11: Zoom.

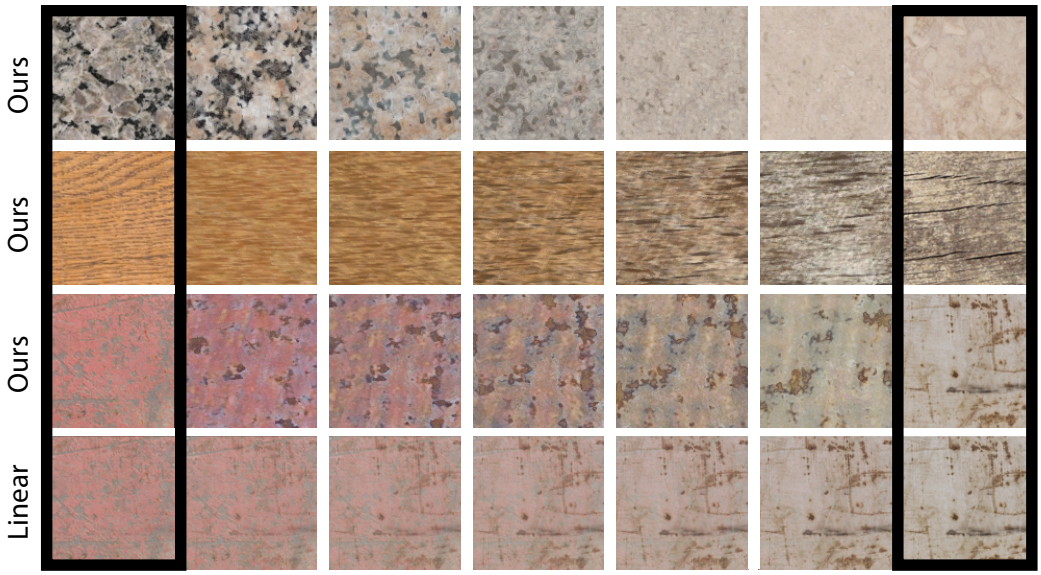


Figure 5.12: Interpolation of one exemplar **(left)** into another one **(right)** in latent space (first three rows) and linear (last row).

A meaningful latent texture code space should also allow for interpolation as seen in Fig. 5.12, where we took pairs of texture codes (left and right-most exemplar) and interpolated rows in-between. We see, that different paths produce plausible blends, with details appearing and disappearing, which is not the case for a linear blend.

5.4.4 User study

Presenting $M = 144$ pairs of images produced by either `perlinT`, `cnnD`, `mlp`, `oursP`, `oursNoT` and `ours` for one exemplar texture to $N = 28$ subjects and asking which result "they prefer" in a two-alternative forced choice, we find that 16.7% prefer ground truth, 4.9% `perlin`, 7.7% `perlinT`, 14.3% `cnn`, 8.8% `cnnD`, 9.4% `mlp`, 10.8% `oursNoT`, 12.9% `oursP` and 14.5% `ours` (statistical significance; $p < .1$, binomial test). Given ground truth and `cnn` are not diverse, our results are preferred over all others methods that synthesize infinite textures.

5.4.5 Method properties

We compare different properties of our method and competitors. An overview is depicted in Tbl. 5.3. Rows list different methods while columns address different aspects of each method. A method is "Diverse" if more than a single exemplar can be produced. MLP [131] is not diverse as the absolute position allows overfitting. We

Table 5.3: Comparison of texture synthesis methods. Please see text for refined definition of the rows and columns.

Method		Diverse	Details	Speed	3D	Quality	Space	2D-to-3D
• Perlin	`perlin`	✓	✓	✓	✓	×	×	×
• Perlin + transform	`perlinT`	✓	✓	✓	✓	×	×	×
• CNN	`cnn`	×	×	×	×	✓	×	×
• CNN + diversity	`cnnD`	✓	×	×	×	×	×	×
• MLP	`mlp`	×	×	✓	✓	×	×	✓
• Ours + position	`oursP`	×	✓	✓	✓	×	✓	✓
• Ours - transform	`oursNoT`	×	×	✓	✓	✓	✓	✓
• Ours	`ours`	✓	✓	✓	✓	✓	✓	✓

denote a method to have "Detail" if it can produce features on all scales. CNN does not have details, as, in particular in 3D, it needs to represent the entire domain in memory, while MLPs and ours are point operations. "Speed" refers to computational efficiency. Due to high bandwidth and lacking data parallelism, a CNN, in particular in 3D, is less efficient than ours. This prevents application to "3D". "Quality" refers to visual fidelity, a subjective property. CNN, MLP and ours achieve this, but Perlin is too simple a model. CNN with diversity [128] have decent quality, but is a step back from [126]. Our approach creates a "Space" of a class of textures, while all others only work with single exemplars. Finally, our approach allows us to learn from a single 2D observation i. e., 2D-to-3D. MLP [131] also learn from 2D images, but have multiple images of one exemplar, and pixels are labelled with depth.

5.5 Conclusion

We have proposed a generative model of natural 3D textures. It is trained on 2D exemplars only and provides interpolation, synthesis and reconstruction in 3D. The key inspiration is Perlin Noise – now more than 30 years old – revisited with NNs to match complex colour relations in 3D according to the statistics of VGG activations in 2D. The approach has the best combination of similarity and diversity compared to a range of published alternatives, that are less computationally efficient.

Reshaping noise to match VGG activations using MLPs can be a scalable solution to other problems in even higher dimensions, such as time, that are difficult for CNNs.

Chapter 6

Generative Modelling of BRDF Textures from Flash Images

While the previous chapter presented a method that synthesizes 3D textures, it does not capture the surface characteristics of textures. In this chapter, we learn a latent space for easy capture, consistent interpolation, and efficient reproduction of visual material appearance. When users provide a photo of a stationary natural material captured under flashlight illumination, first it is converted into a latent material code. Then, in the second step, conditioned on the material code, our method produces an infinite and diverse spatial field of BRDF model parameters (diffuse albedo, normals, roughness, specular albedo) that subsequently allows rendering in complex scenes and illuminations, matching the appearance of the input photograph. supervision.

6.1 Overview

Rendering realistic images for feature films or computer games requires adequate simulation of light transport. Besides geometry and illumination, an important factor is material appearance.

Material appearance has three aspects of variation: First, when view or light direction changes, reflected light changes. The physics of this process is well-understood and can be simulated provided the input parameters are available. Second, behaviour changes across materials. For example, leather reacts differently to light or view changes than paper would, yet, different forms of leather clearly share visual prop-

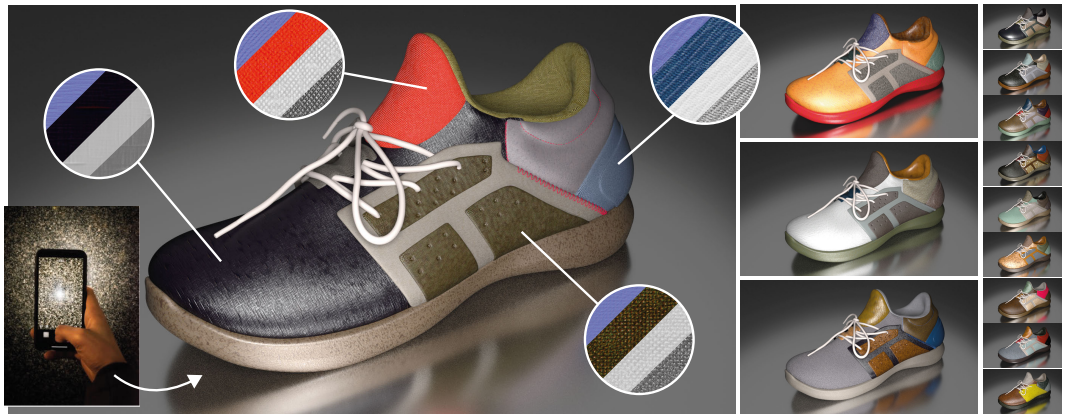


Figure 6.1: Results from our generative model of BRDF maps, assigned to a 3D object of a shoe. Circular insets show diffuse, normal, roughness and specular maps. Our model outputs a space of BRDF materials that can be sampled from, projected to, and interpolated across. The BRDF generative model is trained exclusively from a set of (306) RGB flash images (example shown in the lower left inset) without any BRDF supervision, and shortly fine-tuned in the case of material capture to best match the input picture.

erties, i. e., form a (material) space. Third, appearance details depend on spatial position. Different locations in the same leather exemplar behave differently but share the same visual statistics [118], i. e., they form a *texture*.

Classic computer graphics captures appearance by *reflection models*, which predict for a given i) light-view configuration, ii) material, and iii) spatial position, how much light is reflected. Typically, the first variation (light and view direction) is covered by *BRDF models*, analytic expressions, such as Phong [33] which map the light and view direction vector to scalar reflectance. The second variation (material) is covered by choosing *BRDF model parameters*, such as specularity or roughness. In practice, it can be difficult, given a desired appearance, to choose those parameters e. g., how to make a leather look more like the one on a nice jacket. One can measure BRDF model parameters, but it traditionally requires complex capture hardware for accurate results. The third variation (spatial) is addressed by storing multiple BRDF model parameters in images of finite size –often referred to as svBRDF maps– or writing functional expressions to reproduce their behaviour. It is even more challenging to choose these parameters to produce something coherent like leather, in particular over a large spatial extent. Additionally, storing all these values requires substantial

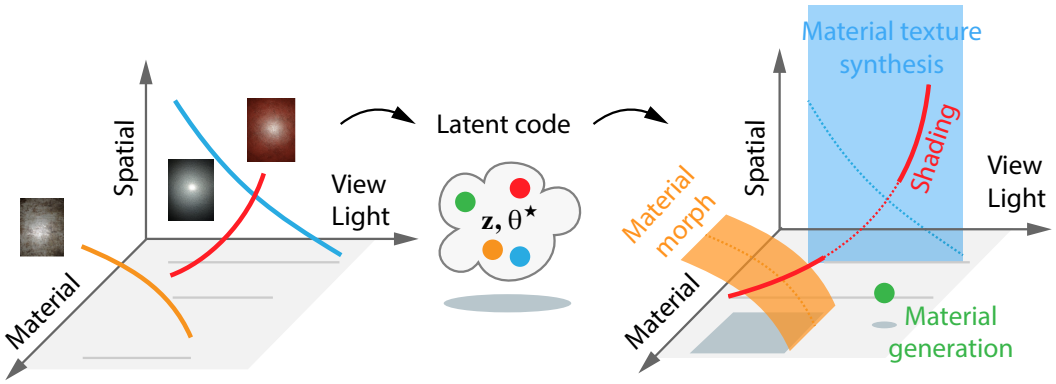


Figure 6.2: BRDF space. From a flash image, which contains sparse observations across material, space and view-light **(left)** we map to a latent code $\mathbf{z}$ / $\theta^\star$ **(middle)** so that changes in these codes can be decoded to enable **(right)** material synthesis (holding material fixed and moving spatially), material morphing (holding space and view/light fixed and changing material), or classical shading and material generation (points in the latent space).

memory and programming functional expressions to mimic their statistics requires expert skills and time. Capturing the spatial variation of BRDF model parameters over space using sensors requires even more complex hardware [32].

Addressing those issues, we provide a reflectance model to jointly generalize across all of these three axes. Instead of using analytic parameters, we parameterize appearance by latent codes from a learned space and our decoder weights, allowing for acquisition, interpolation and generation. Without involved capture equipment, these codes are produced by presenting the system with a simple 2D flash image, which is then embedded into the latent space. Avoiding to store any finite image texture, we learn a second mapping to produce svBRDF maps from the infinite random field (noise) on-the-fly, conditioned on the latent material code and decoder weights. Instead of using any advanced capture device for learning, flash images will be the only supervision we use. This unsupervised approach allows us to consider our decoder weights as part of the latent representation, which we fine-tune at test time in a few minutes.

A use case of our approach is shown in Fig. 6.1. First, a user provides a "flash image", a photo of a flat material sample under flash illumination. This sample is embedded as a code into a latent space using a CNN and used to fine-tune our decoders' weights. This code and weights can then be manipulated, e. g., interpolated with a different

material. Conditioned on this code, our fine-tuned decoder can generate an infinite field of BRDF to be directly used in rendering.

For training, we solely rely on real flash images. The key insight, inspired by Aittala et al. [166], is that these flash images reveal the same material at different image locations –they are stationary– but under different view and light angles. Using this constraint, Aittala et al. [166] were able to decompose a small patch of a single input image to capture the parameters of a material model that could then be rendered under novel view or light directions. However, this covers only part of the generalization we are targeting: it generalizes across view and light, but not across location or material. Further, they perform an optimization for every exemplar, requiring time in the order of an hour, while ours takes minutes only.

In summary, our main contributions are

- a generative model of a BRDF material texture space;
- generation of maps that are diverse over the infinite plane;
- a flash image dataset of materials enabling our training with no BRDF parameter supervision or synthetic data

Our implementation as well as an interactive webpage are publicly available: `https://henzler.github.io/publication/neuralmaterial/`.

6.2 Background

Aittala et al. [166] leveraged the fact that a single flash image of a stationary material reveals multiple realizations of the same reflectance statistics under different light and view angles. We will now recall a simplified definition of their approach.

A flash image is an RGB image of a material, taken in conditions where a mobile phone's flashlight is the dominant light source. We write $L(\mathbf{x})$ to denote the RGB radiance value at every image location $\mathbf{x}$. The illumination is expected to be an isotropic point light collocated with the camera. Further, the geometry is assumed to be flat and captured in a fronto-parallel setting, so that the direction from light to

every image location in 3D is known. Self-occlusion and parallax are assumed to be negligible.

Reflectance is parameterized by a *material*, represented as a function $f(\mathbf{x})$ mapping image location $\mathbf{x}$ to shading model parameters, including the shading normal. Under these conditions, the reflected radiance is $L = \mathbf{R}f$, where $\mathbf{R}$ is the *differentiable* rendering operator, mapping shading model parameters to radiance.

A material f explains a flash image L if it is *visually similar* to L when rendered. Unfortunately, without further constraints, there are many materials to explain the flash image. This ambiguity can be resolved when assuming that the material f is *stationary*. We say a material is stationary if local statistics of the shading model parameters f do not change across the image.

Putting both –visual similarity and stationarity– together, the best material from a family f_θ of material mapping functions parameterized by a vector θ, can be found by minimizing a loss:

$$\mathscr{L}'(\theta) := \mathscr{T}(L, \mathbf{R}f_\theta) + \lambda \mathscr{S}(f_\theta), \tag{6.1}$$

where $\mathscr{T}(L, \mathbf{R}f_\theta)$ is a metric of visual similarity between a flash image L and a differentiable rendering $\mathbf{R}f_\theta$, and $\mathscr{S}(f)$ is a measure of stationarity of a material map f.

Comparison, $\mathscr{T}$, of two textures is not trivial. Pixel-by-pixel comparison is typically not suitable to evaluate visual statistical similarity. Instead, images are mapped to a feature space in which images that are perceived as similar textures, map to similar points [118]. Different mappings are possible here. Classic texture synthesis [195] uses moments of linear multi-scale filter responses. Gatys et al. [124] proposed to use Gram matrices of non-linear multi-scale filter responses such as those of the VGG [29] detection network. Such a characterization of textures was also used by Aittala et al. [166] and, without loss of generality, will be used and extended in this work as well.

While f is stationary, L is not –due to the lighting– and has features at different

Table 6.1: Comparison of features between different previous methods. We distinguish methods producing RGB from those generating **BRDF** or **svBRDF**, whether those can be **Non-Stationary** and **Infinitely** sampled. We also distinguish if their results for one input can be **Diverse** if they form a **Space** which can be queried and how **Fast** direct sampling is.

Method	Supervision	BRDF	svBRDF	Non-Stat.	Infinite	Diverse	Space	Fast Gen.
Classic texture synth	RGB	×	×	✓	✓	✓	×	✓
Matusik et al. [168]	RGB	×	×	✓	✓	✓	✓	×
Matusik [142]	BRDF	✓	×	×	×	✓	✓	✓
Georgoulis et al. [144]	BRDF	✓	×	×	×	×	✓	✓
Deschaintre et al. [145]	svBRDF	✓	✓	×	×	×	×	✓
Zhao et al. [194]	Flash image	✓	✓	✓	×	×	×	×
Aittala et al. [166]	Flash image	✓	✓	✓	×	×	×	×
Gao et al. [161]	svBRDF	✓	✓	✓	×	×	×	×
Guo et al. [160]	svBRDF	✓	✓	✓	×	×	✓	×
Ours	Flash image	✓	✓	×	✓	✓	✓	✓

random positions $\mathbf{x}$ which are compared as

$$\mathscr{T}'(L_1,L_2) := \mathbb{E}_{\mathbf{x}\sim(0,1)^2,s\sim(0,1)}[|\mathscr{P}(L_1,\mathbf{x},s) - \mathscr{P}(L_2,\mathbf{x},s)|_1], \tag{6.2}$$

where $\mathscr{P}'(L,\mathbf{x})$ crops a patch of randomly chosen scale s at the location $\mathbf{x}$ and resamples it to the input resolution of VGG [29], computes the filter responses and their Gram matrices:

$$\mathscr{P}(L,\mathbf{x},s) := \texttt{gram}(\texttt{vgg}(\texttt{resample}(\texttt{crop}(L,\mathbf{x},s)))). \tag{6.3}$$

Minimizing θ with respect to Eq. 6.1 for a given L results in a material. f_θ can represent different approaches. Aittala et al. [166] directly use the pixel basis and optimize discrete material maps for θ using a single input flash image L. With their approach, optimizing for both visual similarity and stationarity is challenging. In particular, the reflectance stationarity term $\mathscr{S}$, requires a "spectral preconditioning" step as explained in their paper. Instead, we propose an approach in the form of a neural model f that is (i) defined on the infinite domain and (ii) stationary by

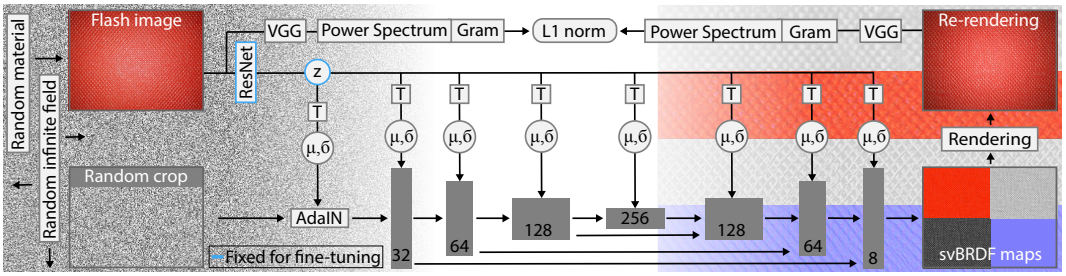


Figure 6.3: Our architecture. Starting from an exemplar **(top-left)** our trained encoder encodes the image to a compact latent space variable z. Additionally, a random infinite field is cropped with the same spatial dimensions as the flash input image. The noise crop is then reshaped based on a convolutional U-Net architecture. Each convolution in the network is followed by an Adaptive Instance Normalization (AdaIN) layer [196] reshaping the statistics (mean μ and standard deviation σ) of features. A learned affine transformation T-s per layer maps z to the desired μ-s and σ-s. The output of the network are the diffuse, specular, roughness, normal parameters of an svBRDF that, when rendered using a camera colocated flash light, look the same as the input. Our unsupervised setting allows us to fine-tune our trained network on materials to acquire.

construction. Thus, our loss does not need to include a stationarity term. To further demonstrate the capabilities of our method, we summarize the design space of current methods in Tbl. 6.1.

6.3 Method

An overview of our approach is shown in Fig. 6.3. We train a neural network which acts as a decoder $f_\theta(\mathbf{x}|\mathbf{z})$ that generalizes across spatial positions $\mathbf{x}$ as well as across materials, expressed as latent material codes $\mathbf{z}$. The material codes $\mathbf{z}$ are produced by an encoder g with $\mathbf{z} = g(L)$. Both encoder and decoder are trained jointly over a set of flash images using the loss:

$$\mathscr{L}(\theta) := \mathbb{E}_L[\mathscr{T}(L, \mathbf{R} f_\theta(\cdot|g_\theta(L)))]. \tag{6.4}$$

This equation is an adapted version of Eq. 6.1 to fit our objectives. In particular, we propose a neural network-based f_θ, leveraging the expectation $\mathbb{E}_L$ over all flash images in our training set and removing the stationarity term as it is enforced by construction in our network architecture. We describe the flash image encoder g (Sec. 6.3.1), the material texture decoder f (Sec. 6.3.2), the texture comparison

model $\mathscr{T}$ (Sec. 6.3.3) and our fine tuning approach (Sec. 6.3.5), next.

6.3.1 Encoder

The encoder g maps a flash image L to a latent code $\mathbf{z}$. The flash images used by our method are similar to those of recent svBRDF acquisition papers [166, 145]: we use a phone with a flash collocated with the camera and capture surface in a fronto-parallel way. Our encoder is implemented using ResNet-50 [190]. The ResNet starts at a resolution of $512\times$ 384 and maps to a compact latent code. Empirically, we find a $n_z = 64$-dimensional latent space to work best for our data and present all results using this number.

6.3.2 Decoder

The decoder f maps location $\mathbf{x}$, conditioned on a material code $\mathbf{z}$ to a set of material parameter maps. The key idea is to provide the architecture with access to noise, as previously done for style transfer [196], generative modelling [136] or 3D texturing proposed in Chapter 5. In particular, we sample rectangular patches with edge length of $n \times m$ pixels from an infinite random field and convert them to material maps using a U-net architecture [155]. The U-net starts at the desired output resolution $n \times m$ and reduces resolution four times using max-pooling before upsampling back to $n \times m$ through a series of bi-linear upsampling and convolutions. Let F be the array of input features. For $i = 0$, the first level, in full resolution, these features are sampled from the random field at $\mathbf{x}$. Then, output features are

$$F' := \texttt{adaIN}(\texttt{conv}_\theta(F), \mathsf{T}_\theta \mathbf{z}), \tag{6.5}$$

where `adaIN` is Adative Instance Normalization (AdaIN) [196], `conv` a convolution (including up- or down-sampling and ReLU non-linearity), z is a latent material code and T is an affine transformation. Components with learned parameters are denoted with subscript θ.

We use AdaIN as defined by Huang and Belongie [196] as

$$\texttt{adaIN}(\boldsymbol{\xi}, \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}) = \frac{\boldsymbol{\sigma}}{\boldsymbol{\sigma}_F}(\boldsymbol{\xi} - \boldsymbol{\mu}_F) + \boldsymbol{\mu} \tag{6.6}$$

and remaps the input features with mean $\boldsymbol{\mu}_F$ and variance $\boldsymbol{\sigma}_F^2$ to a distribution with mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}^2$.

The affine mapping T is implemented as $(n_z + 1) \times (2 \times c_i)$ matrices multiplied with the latent code z. Here $2 \times c_i$ represent a different mean and variance for each channel dimension c_i of a layer. It provides the link between the material code and the noise statistics. Each material code $\mathbf{z}$ is mapped to a mean and variance to control how the statistics of features are shaped at every channel on every layer of the decoder.

Our control of noise statistics from latent codes is similar to StyleGAN [136], with the key difference that we do not sample noise at different scales, but learn how to produce noise with different, complex, characteristics at different scales by repeatedly filtering it from high resolutions.

6.3.3 Images Comparison

As mentioned in Sec. 6.2, we want to evaluate visual similarity and stationarity. To this end, we propose to compare images based on a loss that accounts both for the statistics of activations [124] and their spectrum [197] on multiple scales across the infinite spatial field,

$$\mathscr{T}(L_1, L_2) := \mathbb{E}_{\mathbf{x}\sim\mathbb{R}^2, s\sim(s_{\min}, s_{\max})}[|\mathscr{P}(L_1, \mathbf{x}, s) - \mathscr{P}(L_2, \mathbf{x}, s)|_1]. \tag{6.7}$$

$$\mathscr{P}(L, \mathbf{x}, s) := \texttt{gram}(V(L, \mathbf{x}, s)) + \lambda \cdot \texttt{powerSpectrum}(V(L, \mathbf{x}, s)) \tag{6.8}$$

$$V(L, \mathbf{x}, s) := \texttt{vgg}(\texttt{resample}(\texttt{crop}(L, \mathbf{x}, s))) \tag{6.9}$$

Spectrum VGG Gram matrices capture the frequency of a feature appearance, unless it forms a regular pattern Liu et al. [2016]. Liu et al. [197] proposed to include the L1 norm of the power spectra of RGB images into the texture metric for texture synthesis. We combine both ideas and use VGG, but do not limit ourselves to its Gram matrix statistics, and also leverage its spectrum. We set $\lambda = 1e-3$.

Scale As VGG works at a specific scale of features it was trained for, it behaves differently at different scales. As the material should be visually plausible regardless of its scale we include multiple scales s, ranging from $s_{\min} = 0.1$ to $s_{\max} = 8$ in the loss computation.

Infinity Expectation over the infinite plane is implemented by simply training with different random seeds for the noise field. This results in the generation of statistically similar, but locally different variations of materials. As, given a seed, every generated patch is a coherent material, combinations of multiple patches remains coherent as well. This allows us to query an endless, seamless and diverse stream of patches without repetition. It also prevents over-fitting and is crucial to guarantee stationarity by design.

6.3.4 Training

To enforce a generalizable material prior, we first train the system as a Variational Auto-encoder (VAE) [198]. Instead of mapping to a single 64-D latent material code, the encoder g maps to a 64-D mean and variance vector, from which we sample in training. At test time we use the mean for each 64-D. We have omitted the additional VAE terms enforcing $\mathbf{z}$ to be normally distributed from Eq. 6.4 and Fig. 6.3 for clarity. We trained our model for 4 days and a batch size of 4 on an NVIDIA Tesla V100 using ADAM optimizer with a learning rate of 1e-4 and weight decay 1e-5.

6.3.5 Fine-tuning

Using the trained encoder-decoder pair we can instantaneously compress a 2D RGB flash image to a latent code and decompress it into an infinite svBRDF field. The quality of the decoding can further be improved by adapting the decoder weights to a specific exemplar $L^\star$ with a short one-shot training. To this end, all weights θ are held fixed, except for the decoder weights $\theta^\star \subset \theta$, which are further trained to reproduce a single flash image $L^\star$ at material code $\mathbf{z}^\star = g(L^\star)$. This is made possible by our completely unsupervised approach, allowing us to fine-tune any flash image, without requiring ground truth maps. Note that unlike [160] we use a style loss rather than a pixel-wise loss for fine-tuning, preserving the *diversity* properties of

our results. In practice, we fine-tune for 1000 steps with an increased learning rate by a factor of 10, for about 5 minutes.

Fine-tuning of two materials will result in two different decoders f_1 and f_2 as well as two latent codes $\mathbf{z}_1$ and $\mathbf{z}_2$ produced by the same encoder. We show that despite being a more complex space, interpolating both the latent code and decoder parameters, as in $\texttt{lerp}(f_1, f_2)(\texttt{lerp}(\mathbf{z}_1, \mathbf{z}_1))$ works well in practice, Unless otherwise specified, we show fine-tuned results in the remainder of this paper and ablate several variants in Sec. 6.4.4.

6.3.6 Material model

We use the Cook-Torrance Cook and Torrance [1982] micro-facet BRDF Model, with Smith's geometric term [39], Schlick's Schlick [1994] Fresnel and GGX [38]. Hence, the parameters are diffuse RGB albedo, monochromatic specular albedo, roughness and height, i. e., six dimensions. Instead of learning a normal map, a height field is generated from which normals are computed using finite differences. During our differentiable rendering step, we assume a FOV of 45° to simulate smartphone cameras.

6.3.7 Alignment

Many flash images entail a slight rotation as it can be difficult to take a completely fronto-parallel image. This was handled by Aittala et al. [166] by locating the brightest pixel and cropping, but we found our, more abstract, training to struggle with such a solution.

Instead, we add a horizontal and a vertical rotation angle to the parameter vector generated from the latent code (not shown in Fig. 6.3 for clarity). During training, these are used to rotate the plane, including the normals. During testing, these angles are not applied meaning that the output is in the local space of the exemplar.

We use a branch of the encoder to perform the alignment task, allowing to jointly align images based on their visual features.

A byproduct is that the encoder returns angular distance to fronto-parallelity, which could be used to guide users during capture.

6.4 Results

6.4.1 Dataset

We created an extended dataset of flash images for testing and training of our approach. It comprises 356 images of various types of materials we captured using four different smartphones. We reserve 50 images for testing, augmented by all images from Aittala et al. [166]. Hence, no image from Aittala et al. [166] was used for training.

6.4.2 Quantitative Evaluation

For quantitative analysis, we compare our approach to a range of alternative methods with respect to different metrics.

Methods We compare to five methods by (i) Aittala et al. [166], (ii) Deschaintre et al. [145], (iii) Gao et al. [161], (iv) Guo et al. [160], and (v) Zhao et al. [194]. All renderings of these methods are done with the material model described in their respective paper. While Gao et al. [161] and Guo et al. [160] were designed to be compatible with multiple image acquisition with known light positions, in our comparisons we provide the same input as to our method: a single input image and an approximate light position.

Metrics We quantify *style*, *diversity*, and *computational speed*. Style is captured by L1 difference of the VGG Gram matrices of rendered images. A good agreement in style has a low number i. e., less is better. We also evaluate XYZ histogram L1 difference and find that all methods have below 1% of difference with Ground Truth renderings, indicating good colour matching for all. Histogram difference does not however capture the complex visual difference when comparing materials (as can be seen in Fig. 6.5). Diversity is captured as the mean pairwise VGG L1 across all realizations. Here, more is better. The idea behind this diversity metric is, that for a diverse method, two realizations should have a high difference. A direct pixel metric would be sensitive to noise which generates small perturbations resulting in false-positive differences. Hence, the choice of VGG features detects whether realizations are indeed perceivably different. Note that we do not evaluate

Table 6.2: We compare to recent material acquisition approaches on the L1 difference between VGG Gram matrices (VGG Style, lower is better) on both real and synthetic results as described in Sec. 6.4.2. Additionally we evaluate each method's capacity to generate diverse realizations of a material with the mean pairwise VGG L1 across all realizations (Div, higher is better). We see that ours outperforms others on perceived similarity with the VGG style metric. Additionally, ours is the only one generating diverse material variations from a single image.

Method	Style err. ↓		Div. ↑
	Flash	Relit	
Aittala et al. [166]	0.922	0.512	0.00
Deschaintre et al. [145]	0.943	0.653	0.00
Gao et al. [161]	0.738	0.556	0.00
Zhao et al. [194]	**0.545**	0.618	0.00
Guo et al. [160]	0.843	0.582	0.00
Ours	0.597	**0.439**	**2.08**

pixel-wise metrics such as L1 or SSIM as these enforce local coherence, which is, by construction, not targeted by our method.

Comparisons We use the described metrics to compare against multiple state-of-the-art methods in material acquisition and report the results on real (Flash) and synthetic materials (Relit) in Tbl. 6.2. For real results, we only have access to the frontal flash-illuminated material and therefore compare the picture to a rendering of each method's result also under frontal illumination.

This, however, does not evaluate well the appearance under novel illumination, which is a key property of svBRDFs. To validate the generalization across light directions, we acquire 30 random stationary synthetic svBRDFs from CC0 Texture and render them to simulate a frontal-flash capture setup using Mitsuba2 [199]. All methods are then run with this simulated flash image as input. We report the average of the re-lighting error, against ground truth renderings, for all methods under 10 random point light illuminations.

As shown in Tbl. 6.2, our approach is the only one to target diverse results, i. e., we produce infinitely many realizations of a texture while all other approaches produce only one. Thus, diversity (Div.) is zero for compared methods, while our approach can generate varied realizations for each material.

In terms of computational speed, Aittala et al. [166] and Zhao et al. [194] both

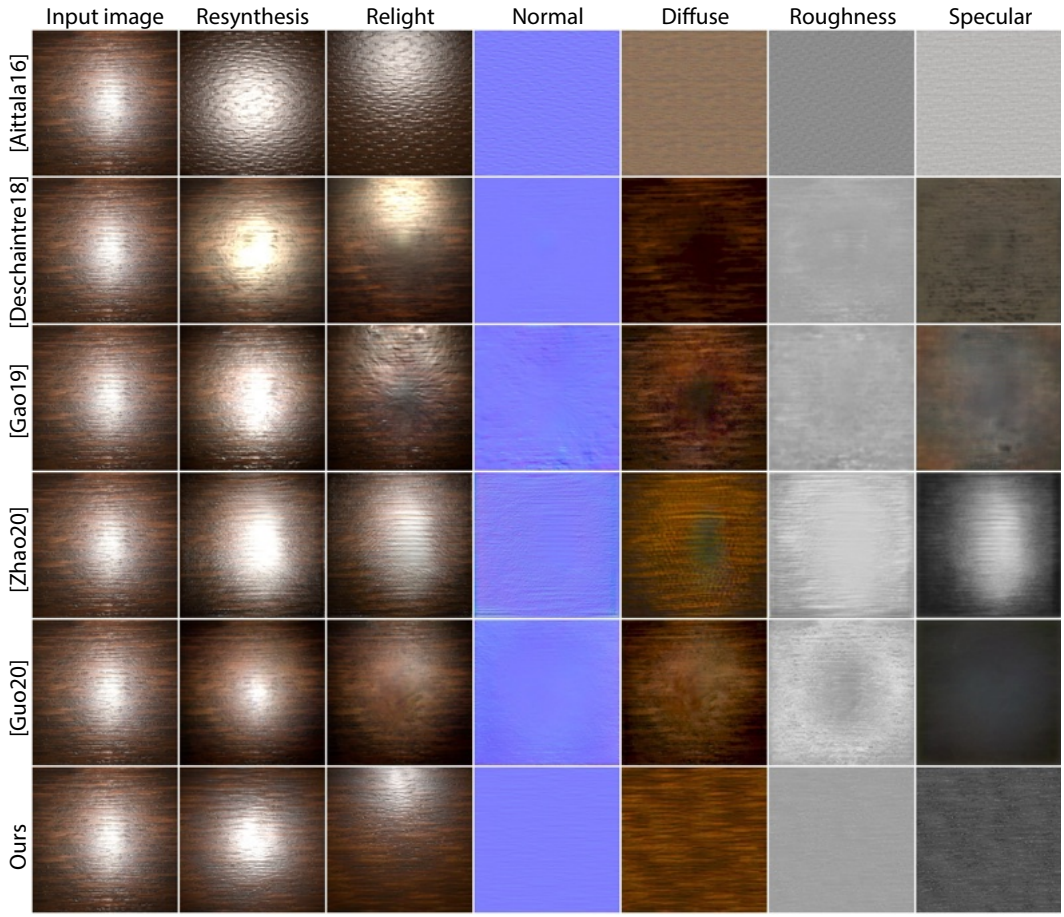


Figure 6.4: Comparison with other methods. Each method **(rows)** decomposes an image into svBRDF parameters **(columns)**. The first column shows the flash image input and the second column the rendering of the results under a similar fronto-parrallel lighting. The third column is the material relit from the top, showing the generalization capacity across light. Our method's quality is particularly visible under a novel illumination (see also Fig. 6.5). This is because other methods leave a trace of the flash in the svBRDF maps, as can be seen in the decomposed channels (four right-most columns). These results are obtained with our single image setting, compared methods Gao et al. [161] and Guo et al. [160] could benefit from additional aligned images or accurate light calibration when available. Please see the supplemental material for similar results on many more materials.

require long –between 1 and 3 hours– per-exemplar optimization to produce a stationary texture. Our approach requires around 500 ms to generate a material and a few minutes to fine-tune it to a given input. This is in the same order of speed as Deschaintre et al. [145] for generation and Gao et al. [161] and Guo et al. [160] for the fine-tuning. Once fine-tuned, our method can generate new realizations and high resolutions versions of the targeted material in around 500 ms.

6.4.3 Qualitative Evaluation

Decomposition A qualitative example of our svBRDF decomposition (Normal, Diffuse Roughness and Specular maps) and re-renderings under different lights are depicted in Fig. 6.4. Please see our supplemental material for all results decomposition and comparison. We see that our method captures best the material behaviour and does not suffer from artefact in the over-exposed area of the input image, which can be seen in previous work. As our method uses materials statistics rather than direct pixel-aligned image to material transformation, it is immune to such artefacts.

Relighting In Fig. 6.5, we show qualitative rendering comparisons on real materials with illumination coming from the top. In this more challenging setting, it is clear that existing works struggle to remove the highlight from the center of the flash image, which does not affect our method. As Aittala et al. [166] reconstruct a small (representative) patch of the large input picture, their method is also immune to flash artefacts, but results in a very zoomed representation of the material. To compensate for this "zoom factor", we tile the results in each direction. We empirically found that 3 times works best for most materials.

Seeds In Fig. 6.6, we show the variation of our results when changing the seed. The overall appearance of the material remains the same, but the details (such as the rust or the leather normals and colour variation) vary.

Overall, we see in Fig. 6.4, Fig. 6.5 and Fig. 6.6 that our approach can capture a large range of different stationary materials, reproducing their style, yet being diverse. This enables different properties described next.

Infinite We show in Fig. 6.7 the "infinite" resolution capacity of our approach against the common approach of tiling. Our result (top image) shows no sign of repetitiveness even for a very large resolution (4096×256).

Interpolation We show results of interpolation between materials, as described in Sec. 6.3.5, in Fig. 5.12 and Supplemental Material. We compare against the linear interpolation baseline and Guo et al. [160], which also allows interpolation. We find our method to provide smoother interpolation than the Linear approach and to better preserve intermediate material color than Guo et al. [160]. We additionally evaluate

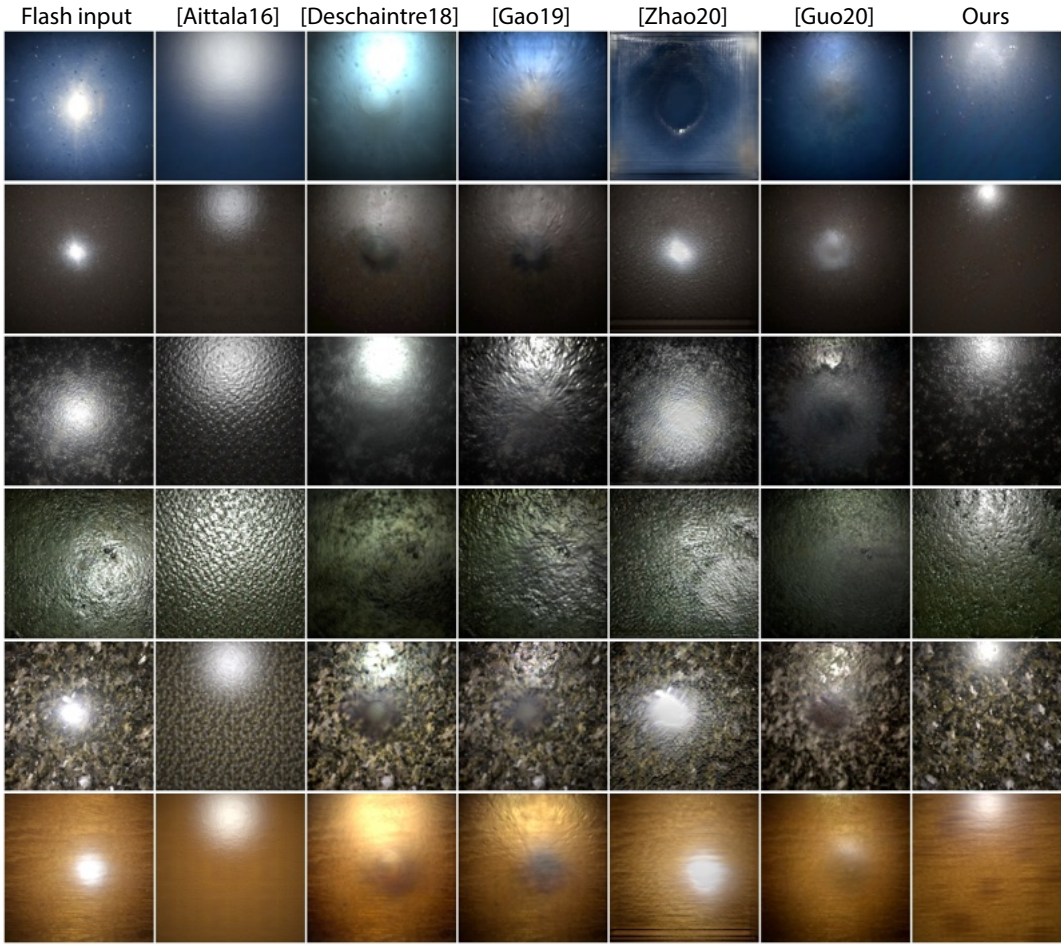


Figure 6.5: Relighting of different materials (rows) using material maps extracted by different methods (columns). The first column shows the input flash image where light is fronto-parallel. The light in all other images comes from the top. While no reference is available for this task, it is apparent that all the methods except ours struggle to generalize to novel light conditions. Note that Deschaintre et al. [145], Gao et al. [161] and Guo et al. [160] leave a dark residual of the flash in the material maps. Zhao et al. [194] and Aittala et al. [166] fare slightly better and avoid the residual, but the structures do not match. These results are obtained with our single image setting, compared methods Gao et al. [161] and Guo et al. [160] could benefit from additional aligned images or accurate light calibration when available.

interpolation if we directly train on material individually (without the training step described in Sec. 6.3.4). This confirms that this pre-training forms a coherent latent space in which we can navigate.

Texturing Fig. 3.1 shows examples of applying maps produced by our approach to a complex 3D shape. Thanks to our generative model, we can easily texture many sneakers, without spatial or material repetition. At any point, a user can randomize

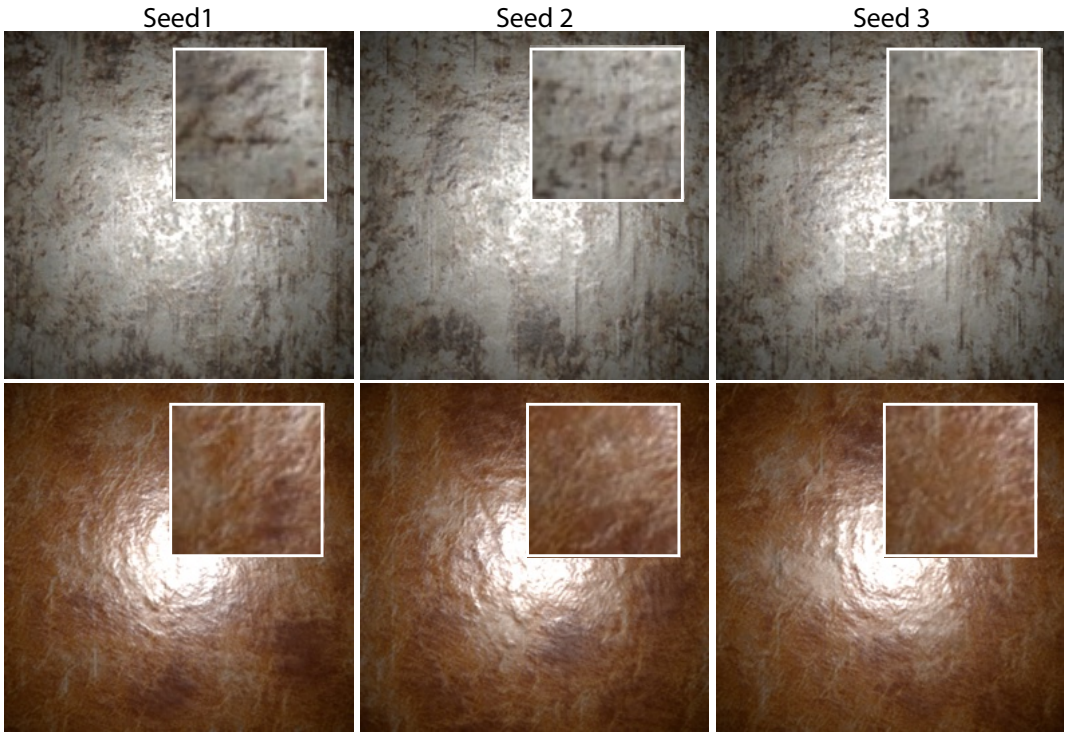


Figure 6.6: Seeds variation. We vary the seed for the generation of different realizations for acquired materials while preserving their overall appearance. The zoomed-in insets all show the same region of the material, allowing us to better appreciate the variations.

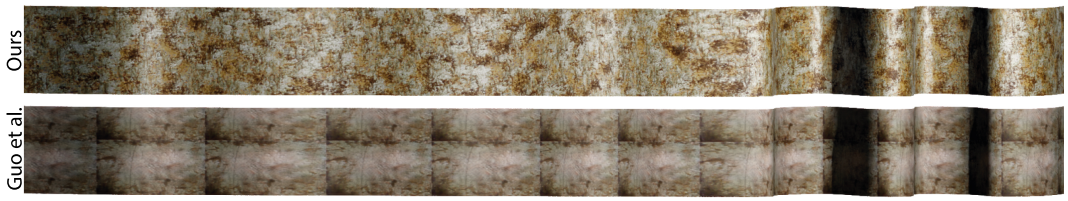


Figure 6.7: Infinite spatial extent. The top result is sampled at high resolution (256×4096) from our BRDF space, while the bottom result is a result from Guo et al. [160] at 256×256 resolution and horizontally tiled 16 times to achieve high resolution. The absence of repetitiveness in the top result demonstrates that our learned BRDF space can be sampled at any query (x,y) location without producing a visible repetition artefact. By construction, our network architecture does not require any special boundary alignment to avoid tiling artefacts.

the generated material, generate new materials from pictures or interpolate between new materials and old ones.

Generation Our **z** space can be sampled to generate new materials as shown in Fig. 6.9 with a variety of examples.

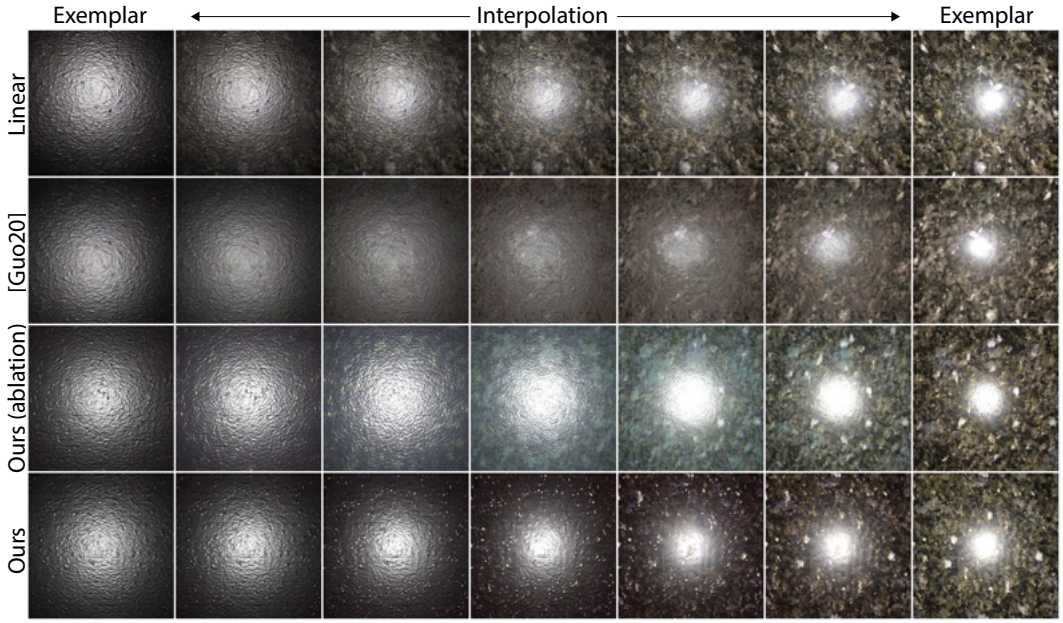


Figure 6.8: Interpolation of latent BRDF texture codes. In each row, a left and a right latent code and generator weights $\mathbf{z}_1$, $\mathbf{z}_2$, $\mathbf{g}_1$, $\mathbf{g}_2$ are obtained by encoding two flash images, respectively. The intermediate, continuous field of BRDF parameters is computed by interpolating, in the learned BRDF space, from $\mathbf{z}_1$ & $\mathbf{g}_1$ to $\mathbf{z}_2$ & $\mathbf{g}_2$ and conditioning the decoder Convolutional Neural Network (CNN) with the intermediate codes. The result is lit with a fronto-parallel light source to demonstrate the changes in appearance. For comparison, the first row shows image space linear interpolation, the second compares to Guo et al. [160]. The third row shows an ablation of our approach trained on a single material (without previous full dataset training). This lack of training prevents it from creating a cohesive space in which to interpolate. Overall our approach allows for interpolation, progressively changing both structure and reflectance.

Interactive demo The visual quality is best inspected from our interactive WebGL demo in the supplemental material. It allows exploring the space by relighting, changing the random seed and visualizing individual BRDF model channels and their combinations. The same package contains all channels of all materials as images as well as compared methods. See the accompanying video for a demonstration of our interactive interface.

Fine-tuning We show the results improve in quality when using the proposed fine-tuning approach in Fig. 6.10. We can see that the structure and details better match the input picture.

Figure 6.9: Generation. Random samples from our space. We generate new materials by sampling the **z** space and render them with a frontal flash. See supplemental materials for more generated materials.

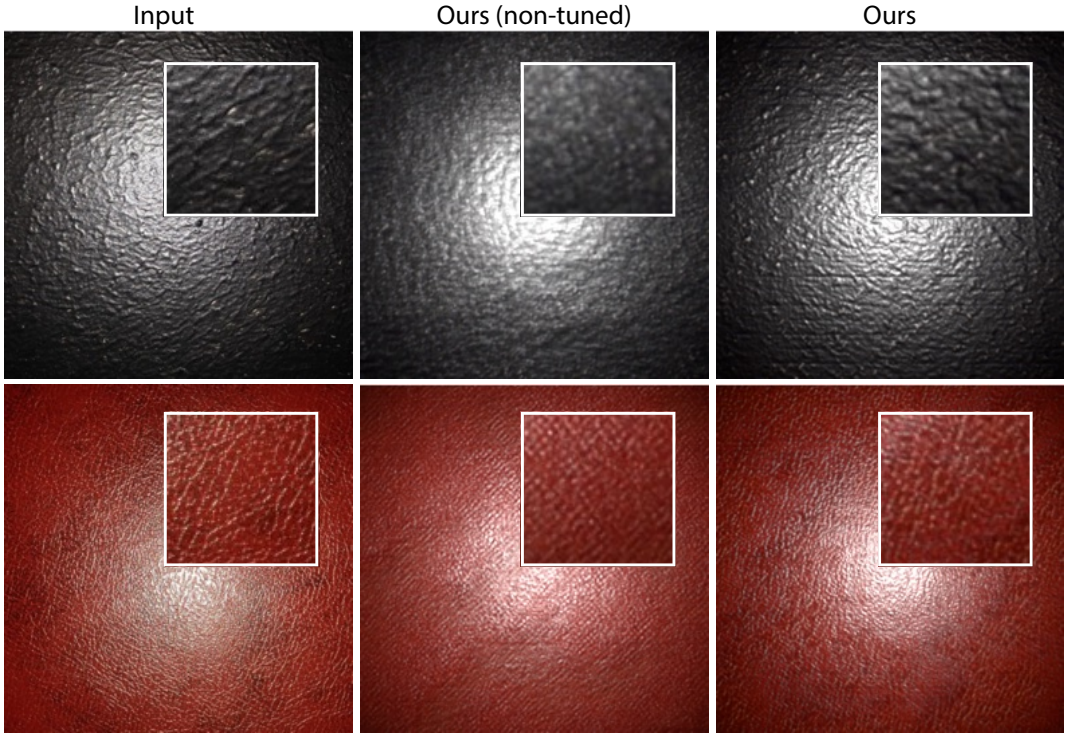


Figure 6.10: Fine-tuning. We show results on two results of real materials reproduced using our pre-trained network (ours non-tuned) and the same material using our fine-tuning approach. We can see that our fine-tuned results match the input material appearance significantly better. Note that fine-tuning is only with image supervision and does not have access to any underlying BRDF supervision.

6.4.4 Ablation Experiments

We study several variants of our approach to evaluate the relevance of individual contributions to our FULL method.

We report the results of this evaluation in Tbl. 6.3 with VGG Style error in Sec. 6.4.2.

We did not find the diversity of our method to be affected by these ablations.

SINGLE describes our method trained on a single example without the previous

training step. The results are slightly better than our FULL method but requires twice as much time per material training and does not generalize to a space, preventing interpolation and generation of materials.

NONTUNED is our method without the fine-tuning step from Sec. 6.3.5, confirming that it significantly improves the match to the acquired material. DECODERONLY describes the change of our generator to a decoder-only architecture. We show that removing the encoder part of the generator slightly degrades the results. FOURIER and LIGHT respectively result from the removal of the Fourier component (power spectrum) of our loss (Sec. 6.3.3) and the removal of the light alignment branch of our encoder (described in Sec. 6.3.7), which both lead to slightly worse results.

Table 6.3: VGG style error for ablations relative to our FULL. For reference, our full method has an absolute score of 0.44.

Ablation	Error ↓
SINGLE	-0.5%
NONTUNED	+24.0%
DECODERONLY	+2.0%
FOURIER	+0.9%
LIGHT	+1.7%

6.5 User experiment

We perform a user study to better understand the capabilities of different methods. Our main aim is to provide material maps that robustly generalize to all light conditions so they can be deployed in production rendering. Hence we study a *relighting task*: given an input image in one light condition, we ask humans to pick the method that looks most plausible "in a different light".

Methods Subjects anonymously completed an online form without a time limit. At the start of the user study, participants were shown two photos of a marble material taken under two different lighting conditions to exemplify what a valid relighting could look like. They performed 10 trials, each corresponding to one material. In each trial, they were presented a reference image rendered in one light condition ("flash") and six relit images in another light condition ("top"). Relit images were

Table 6.4: User preferences per method.

Method	Freq.
Aittala et al. [166]	21
Deschaintre et al. [145]	10
Gao et al. [161]	4
Guo et al. [160]	11
Zhao et al. [194]	30
Ours	314

displayed in a randomized spatial 2D layout. We consider six different methods: Aittala et al. [166], Deschaintre et al. [145], Gao et al. [161], Guo et al. [160], Zhao et al. [194] and ours. Samples of those stimuli are seen in Fig. 6.5. Participants were asked to pick the image (images were not named) that, according to them, was the best faithful relighting of the source (flash) image. Note that no relit reference was shown.

Analysis A total of $N = 39$ participants completed the experiment as summarized in Tbl. 6.4. A χ^2 test rejects ($p < 0.0001$) the hypothesis that choices were random. Pairwise binomial post-hoc tests further show that our method is different from any other method at the same significance level. Most importantly, subjects choose our method in 314 out of 390 total answers 80.5 %). We did not analyze the relation of other methods relative to each other.

6.6 Limitations

Our method relies on fronto-parallel flash acquisition. While we propose a mitigation solution in Sec. 6.3.7, we show in Fig. 6.11 that we are not completely invariant to large light and plane rotations. Our approach is also limited to stationary isotropic materials and relies on the planarity of the captured surface.

6.7 Conclusion

We have presented an approach to generate a space of BRDF textures using a small set of flash images in an unsupervised way. Comparing this approach to the literature shows competitive metrics for re-renderings with the unique advantage of being able

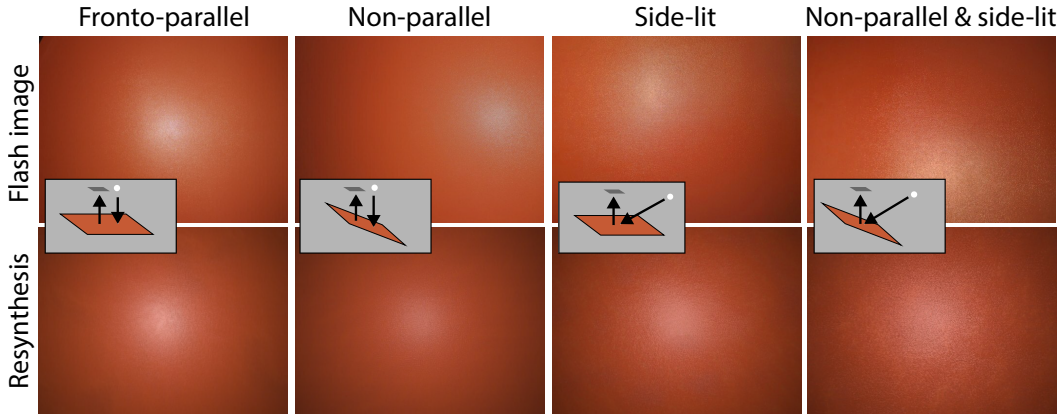


Figure 6.11: Flash acquisition assumption. We show examples of how our results degrade when fronto-parrallel, collocated flash assumptions are broken. The recovered material appearance varies (roughness, high frequency normal) but maintains the overall appearance of the input image.

to generate an infinite and diverse field of BRDF parameters.

In the future, it would be interesting to increase the complexity of supported material whether in terms of shading or non-stationarity. Also, not relying on fine-tuning to increase the network expressiveness would allow to create an even more cohesive space. Further, more refined differentiable rendering material models could be used to derive stochastic textures, including shadows, displacement, or scattering as well as volumetric or time-varying textures. We believe that our framework will represent a stepping stone for more complex, infinite and diverse BRDF acquisition.

Chapter 7

Conclusion

Self-supervised learning is an essential part of artificial intelligence. As we have seen in this thesis, the combination of AI and differentiable inverse graphics allows us to explicitly model higher-dimensional data without direct supervision during training time.

3D reconstruction Acquiring large amounts of annotated 3D data is expensive and tedious. In Chapter 3, hard-coded differentiable image formation models enabled self-supervised 3D reconstruction from unstructured image collections through adversarial rendering. However, the resulting quality of the proposed approach is limited by low voxel resolutions and camera poses were not explicitly modelled, i. e., 3D reconstructions are not canonically aligned as they are forced to be in camera space. These limitations were addressed in Chapter 4. A large-scale dataset consisting of object-centric videos is introduced providing multi-view data. Camera poses are extracted automatically by an off-the-shelf SfM algorithm, providing additional supervision without human effort. Exploiting multi-view data, a novel warp-conditioned ray embedding (WCE) facilitates aggregation of multiple input images and enables single- or multi-view reconstruction. The limiting voxel grid representation was replaced with a neural field leading to results with higher fidelity. Unlike the method in Chapter 3, reconstruction can be performed in world space instead of camera space, due to known camera poses, which further helps to improve the results.

Texture and material synthesis Disentangling shape, material and lighting is an important aspect towards controllable 3D content generation. The second part of the thesis focused on the disentanglement of material and lighting, for which modeling the material parameters as stationary textures was crucial. In Chapter 5, a self-supervised 3D texture synthesis method only relying on stationary 2D texture images is introduced. It features a generative model producing diverse 2D and 3D natural textures. A major advantage of the proposed method is the use of an encoder that can be used to map any given exemplar into a compact latent space while previous texture models need to be re-trained for each exemplar. Furthermore, it allows for 3D texturing independent of shape or resolution without the need for UV mapping. However, the method only captures RGB textures rather than material properties which are required for physically-based rendering.

Building upon the previous idea, the proposed method in Chapter 6 alleviates this by synthesizing material parameters from flash images. For a given stationary flash image, the method is trained to produce material parameters, that when re-rendered, resemble the input flash image under a statistical loss. Such a design does not require access to the underlying svBRDF decompositions and can be trained in a self-supervised fashion. Unlike prior work, an easy-to-acquire real-world dataset only consisting of flash images is used for training rather than relying on synthetic data containing pairs of flash images and svBRDF decompositions. Towards generative texture modeling, the method generates a space of svBRDF textures, which can be sampled from and produces diverse and infinite svBRDF textures.

7.1 Limitations

Even though methods advancing the progress towards shape and appearance modeling from single or few observations have been presented, they all lack individual control over camera pose, geometry, material and lighting. The two methods presented in Chapter 3 and Chapter 4 combine geometry, material and lighting into a single representation where only camera pose can be controlled. In Chapter 5, 2D textures are lifted to solid 3D textures, which decouples texturing from geometry,

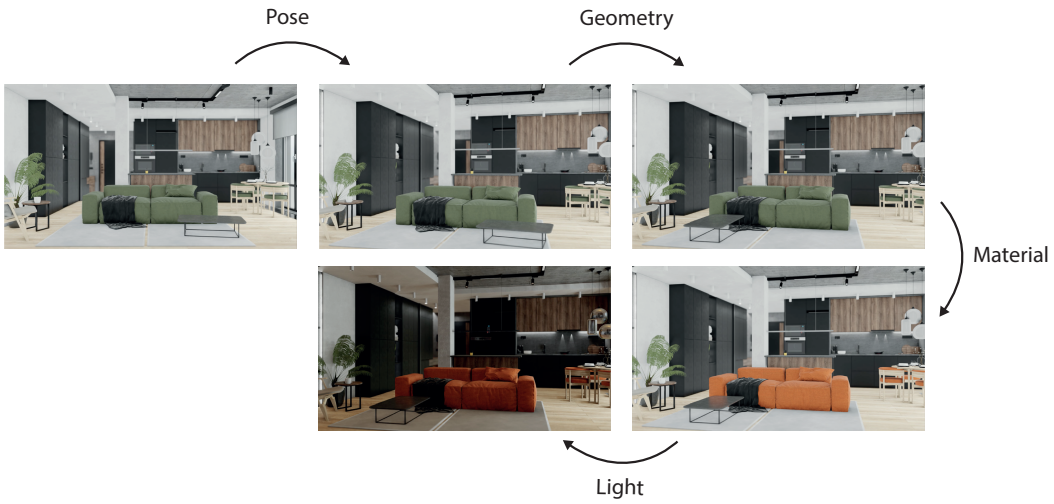


Figure 7.1: Examples of changing individual scene parameters are depicted. Beginning on the top left, pose, geometry, material and light are changed consecutively. First, the pose is tilted to the right, before moving the table from right to left. Next, the material of the couch and the chairs is changed from green to orange fabric. Lastly, the light is dimmed.

lighting and camera pose. However, this is accomplished in a well-defined setting where no 3D geometry, lighting or camera pose is modeled. Material and lighting are disentangled in Chapter 6, but again no 3D geometry or lighting are inferred as both were known at training time. Even though not all scene parameters can be controlled, each method provides a different subset of control. Ultimately, the question arises whether it is possible to disentangle all scene parameters from just a single or a few observations of a scene. An example of a potential pipeline to achieve this is presented in Fig. 7.1.

To gain a better understanding of the limitations prevailing in current methods, let us consider an abstract model. The aim of the model is to demonstrate the capabilities of current methods. These are defined by the type of data that is required for training as well as inference and the achieved amount of disentanglement. We start off with a model of an ideal world as illustrated in Fig. 7.2.

In an ideal world, no limitations, in terms of annotated data, exist as knowledge of an infinite amount of disentangled camera poses, geometry, materials and light is assumed. In deep learning terms, this is equivalent to fully annotated training data. Taking a snapshot of this world in the form of an image consequently produces an

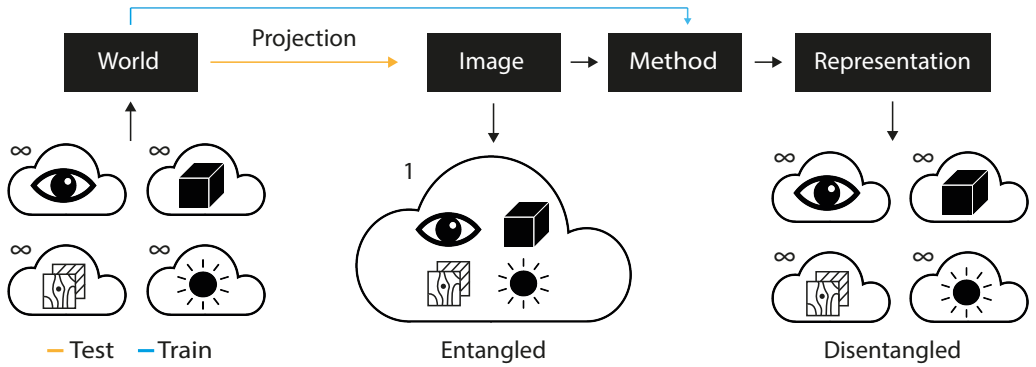


Figure 7.2: An abstract model that illustrates training a supervised algorithm that is capable of disentangling scene parameters from a single image. See text for more details.

entangled version, which we seek to disentangle into its original parameters. In this instance, supervised learning could be used to address the task, as full supervision would be available. However, as discussed in Chapter 1, realistic and large-scale annotated 3D datasets do currently not exist and potentially never will.

In the following, we will see how current methods relate to our abstract model, as illustrated in Fig. 7.3. Recently, NeRF [54] has exploited multi-view data of a single scene for novel-view synthesis and has shown results of impressive quality. Multi-view data provides several poses of the same geometry, material and light and thus provides partially disentangled supervision. NeRF provides full control of the camera pose, however, only for a single scene and static light, which prevents generalization. Following up on this idea, several methods have been proposed to further disentangle the parameter space by accounting for lighting changes. Some of these require different poses under different lighting conditions of the same geometry and material [200, 201]. Others either rely on a material database as supervision [202] or cannot disentangle shadows from diffuse albedo [203]. Although they are able to disentangle camera poses, light and some even materials, this is only possible for a single scene. Therefore, control is limited to single scenes where many hundreds of images are required for good quality which means there is no potential for scalability or extrapolation.

In comparison, PlatonicGAN has only access to unstructured and completely entangled images. Despite this challenge, the method disentangles camera poses in a

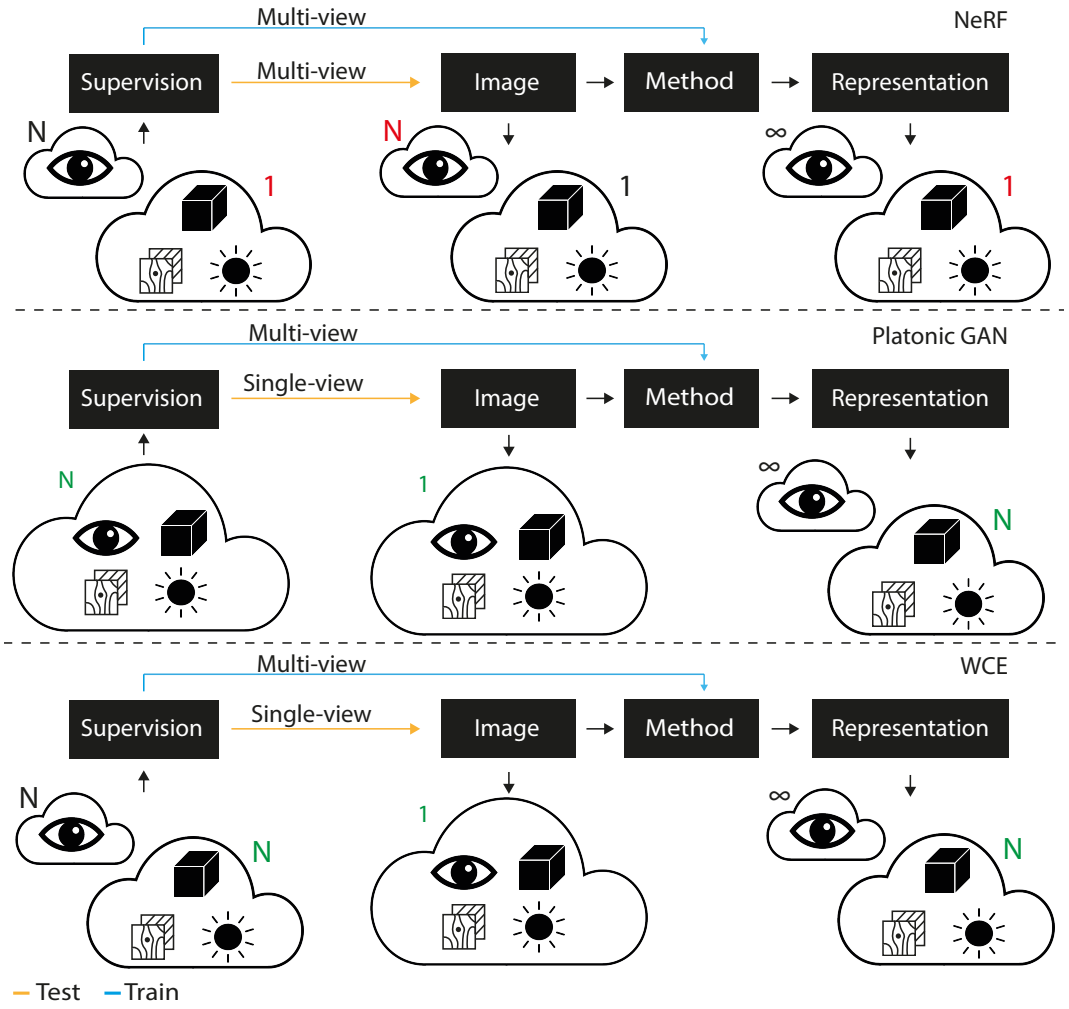


Figure 7.3: Current methods (NerF, PlatonicGAN, WCE) provide limited capabilities for scene parameter disentanglement. Following an abstract model required training and test supervision, the level of disentanglement and scalability potential are illustrated.

platonic way from only a single image. This is possible as the method is trained on an unstructured image collection containing images of different scenes. As a result, it is able to extrapolate to unseen scenes which is a desired property, especially with respect to disentanglement. Unfortunately, the resulting quality is very poor.

In Chapter 4, WCE is proposed, a method that significantly increases the resulting quality at the cost of multi-view data as supervision. Nevertheless, it still allows for generalization while being scalable. It would be interesting to see if this method setup can be extended for more disentanglement. Inspired by those insights, prospective future research directions are discussed in the next section.

7.2 Future Work

7.2.1 Platonic Way

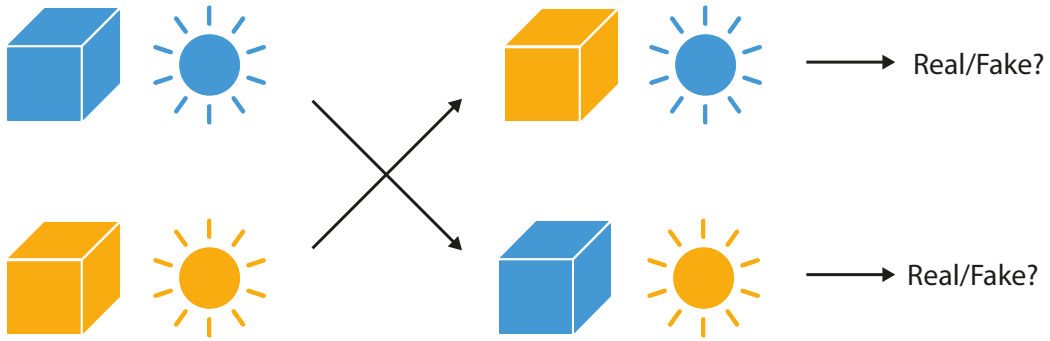


Figure 7.4: Disentangling scene parameters from images could be achieved by ensuring that arbitrary combinations of individual scene parameters of different scenes are not distinguishable from realistic images when rendered.

One way to achieve more disentanglement could be to revisit PlatonicGAN from Chapter 3. The key idea is to generate a 3D model that when rendered from a random pose looks natural, i. e., cannot be distinguished from real images by a discriminator network. Previously, geometry, material and illumination were predicted jointly which prevents disentanglement by construction. For a better disentanglement of the scene parameters, a novel method could model them individually. This would allow swapping of different parts between different scenes. For instance, geometry and lighting could be switched between two scenes as illustrated in Fig. 7.4. This way, arbitrary combinations of scene parameters can be rendered. In PlatonicGAN fashion, the resulting images could be enforced to look natural by a discriminator. Note that previously only one scene parameter, the camera pose, was changed. This setup does not explicitly resolve ambiguities that are caused by rendering, but incentivizes the individual elements to be plausible and therefore possibly allows for disentanglement. To further enforce disentanglement, each individual component could be embedded into a global latent space by an encoder network. After reconstruction, a single parameter could be changed multiple times and re-rendered. By feeding the re-rendered images into the encoder network, a self-supervised loss on the unchanged parameters could be applied, which should further reduce ambiguities.

7.2.2 Representation invariance

Figure 7.5: Instead of directly predicting RGB values of pixels it desired to represent each pixel as a meaningful latent vector that describes higher-level surface characteristics.

Current methods mainly use pixel-perfect objective functions such as Euclidean distances (L1 and L2) or perceptual losses [204] such as VGG [127]. Euclidean losses compute distances between pixels independently and average these. This often leads to blurry results and missing high-frequency details. Modern perceptual losses are based on Euclidean distances between higher-level feature activations from pre-trained CNNs and visually improve results. Both types of losses struggle to produce sharp and realistic images. To address these shortcomings, GANs minimize the distance between the output distribution and original data distribution such that a discriminator network is not able to distinguish between real and fake images. Although they generate realistic images [106], they cannot be applied to image reconstruction tasks without perceptual losses for regularization. Further, they are notoriously hard to train and suffer from mode collapse [205].

Instead, what if we do not ask for a pixel-perfect loss but rather pose the problem as local texture synthesis? We have already seen that the gram matrix [28] of VGG features is capable of representing global image statistics. The goal then would be to define local regions that share similar statistics and describe them independently from each other, similar to Park et al. [206]. These local regions can be represented by smooth segmentation masks in the form of a meaningful latent space that describes local surface characteristics, e. g., *brown wood*, *white paint*,

grey stone as illustrated in Fig. 7.5. Converting these smooth regions to match local statistics could be achieved by using stationary textures or materials that are invariant to local changes. Such an invariant representation could also be applied to geometry or other parameters in the scene. For instance, instead of applying a loss pointwise on densities, a statistical measure could take into account the local neighbourhood. Geometric properties such as round corners or sharp edges could be enforced in such a manner. Both of these ideas are promising directions for future work.

Appendix A

Escaping Plato's Cave: 3D Shape From Adversarial Rendering

A.1 Network architectures

We used PyTorch [176] in version `1.0.0`. Our network architectures are shown in Tbl. A.1, Tbl. A.2, Tbl. A.3, and Tbl. A.4. We trained the networks using a batch size of 4. We used two Adam optimizers for Encoder + Generator and Discriminator respectively with $\beta_1 = 0.5$ and $\beta_2 = 0.999$. We used learning rates of 0.0025 and 10^{-5} for encoder + generator and discriminator respectively and a latent space size z of 200.

Platonic For training stability reasons we only trained the discriminator when $accuracy > 0.8$ similar to [71].

3DGAN and Platonic3D We used $\lambda = 10^4$ for the 3D groundtruth reconstruction term.

A.2 Evaluation Details

We evaluated our method on more classes: `chair`, `car`, `lamp`, `rifle`. See Tbl. A.5.

Table A.1: Network architecture for the generator model. C corresponds to the number of channels for the different image formations. (VH, AO:C = 1, EA: C = 4). We use CONVTRANSPOSE (ConvT) layers for upsampling.

Layer	Kernel	Activation	Shape	# params
Input	—	—	200	—
ConvT	4x4x4	BN+ReLU	1024 x 4 x 4 x 4	∼13M
ConvT	4x4x4	BN+ReLU	512 x 8 x 8 x 8	∼34M
ConvT	4x4x4	BN+ReLU	256 x 16 x 16 x 16	∼8M
ConvT	4x4x4	BN+ReLU	128 x 32 x 32 x 32	∼2M
ConvT	1x1x1	—	c x 64 x 64 x 64	∼c*8k
Sigmoid	—	—	c x 64 x 64 x 64	—
# params	—	—		∼75.2M

Table A.2: Network architecture for the encoder model. C corresponds to the number of channels for the different image formations. (VH, AO:C = 1, EA: C = 4).We use LeakyReLU with `negative slope` = 0.2.

Layer	Kernel	Activation	Shape	# params
Input	—	—	c x 64 x 64	—
Conv	4x4	BN+LReLU	128 x 32 x 32	∼ c * 2k
Conv	4x4	BN+LReLU	256 x 16 x 16	∼ 525k
Conv	4x4	BN+LReLU	512 x 8 x 8	∼ 2M
Conv	4x4	BN+LReLU	1024 x 4 x 4	∼ 8M
Conv	4x4	BN+LReLU	2048 x 1 x 1	∼ 34M
Linear	—	—	200	∼ 400k
# params	—	—		45M

Table A.3: Network architecture for the discriminator model (2D). C corresponds to the number of channels for the different image formations. (VH, AO:C = 1, EA: C = 4).We use LeakyReLU with `negative slope` = 0.2.

Layer	Kernel	Activation	Shape	# params
Input	—	BN+LReLU	c x 64 x 64	—
Conv	4x4	BN+LReLU	128 x 32 x 32	∼ c * 2k
Conv	4x4	BN+LReLU	256 x 16 x 16	∼ 525k
Conv	4x4	BN+LReLU	512 x 8 x 8	∼ 2M
Conv	4x4	BN+LReLU	1024 x 4 x 4	∼ 8M
Linear	—	—	1	∼ 16k
# params	—	—		11M

Table A.4: Network architecture for the discriminator model (3D). C corresponds to the number of channels for the different image formations. (VH, AO:C = 1, EA: C = 4).We use LeakyReLU with `negative slope` = 0.2.

Layer	**Kernel**	**Activation**	**Shape**	**# params**
Input	—	BN+LReLU	c x 64 x 64	—
Conv	4x4x4	BN+LReLU	128 x 32 x 32	$\sim$ c * 4k
Conv	4x4x4	BN+LReLU	256 x 16 x 16	$\sim$ 2M
Conv	4x4x4	BN+LReLU	512 x 8 x 8	$\sim$ 8M
Conv	4x4x4	BN+LReLU	1024 x 4 x 4	$\sim$ 33M
Linear	—	—	1	$\sim$ 65k
# params	—	—		44M

Table A.5: Performance of different methods with varying degrees of supervision (superv.) **(rows)** on different metrics **(columns)** for the different classes: `chair`, `car`, `lamp`, `rifle` (200 shapes from the test set are used). Evaluation is performed on absorption-only (AO). Note, DSSIM and VGG values are multiplied by 10, RMSE by 10^2 and CD by 10^3. Lower is better except for IoU where higher is better.

Method	Class	Superv.	2D Image Re-synthesis										3D Volume		
			VH		AO		EA		VOX		ISO				
			DSSIM	VGG	DSSIM	VGG	DSSIM	VGG	DSSIM	VGG	DSSIM	VGG	RMSE	IoU	CD
Ours	chair	2D	1.58	6.48	1.66	5.37	1.46	5.64	**1.71**	6.31	1.59	6.79	**9.24**	0.25	**10.31**
3DGAN [71]	chair	3D	0.99	5.80	1.36	4.88	1.35	5.23	1.79	5.85	1.62	6.30	9.56	0.33	36.67
Ours 3D	chair	2D+3D	**0.95**	**5.58**	**1.31**	**4.84**	**1.29**	**5.17**	1.72	**5.81**	**1.52**	**6.15**	9.51	**0.39**	35.60
Ours	car	2D	1.01	4.56	1.45	3.88	1.20	4.38	1.56	4.96	1.45	5.52	**12.17**	0.39	**17.32**
3DGAN [71]	car	3D	**0.41**	**2.95**	**0.59**	**2.93**	**0.73**	**3.36**	**1.40**	**4.00**	**1.20**	**4.39**	13.76	**0.55**	93.64
Ours 3D	car	2D+3D	0.44	3.08	0.66	3.01	0.78	3.45	1.43	4.09	1.23	4.48	13.58	0.26	88.01
Ours	lamp	2D	1.21	5.92	1.14	4.78	**0.97**	**5.21**	**1.17**	5.74	**1.11**	**6.16**	**7.14**	0.26	**5.07**
3DGAN [71]	lamp	3D	1.00	6.17	1.14	4.84	1.15	5.29	1.42	5.85	1.39	6.45	8.13	**0.28**	24.95
Ours 3D	lamp	2D+3D	**0.92**	**5.87**	**1.07**	**4.74**	1.11	5.23	1.37	**5.69**	1.31	6.21	8.45	0.24	22.74
Ours	rifle	2D	0.65	4.41	0.65	3.73	0.64	3.75	0.73	4.13	0.73	4.74	3.95	0.22	**2.97**
3DGAN [71]	rifle	3D	**0.50**	**3.78**	**0.50**	**3.26**	**0.54**	**3.26**	**0.64**	**3.67**	**0.64**	**4.10**	**3.74**	**0.39**	5.35
Ours 3D	rifle	2D+3D	0.57	4.04	0.57	3.37	0.60	3.40	0.71	3.80	0.70	4.30	4.01	0.16	5.94

Appendix B

Unsupervised Learning of 3D Object Categories from Videos in the Wild

Table B.1: We complement the evaluation of the impact of the number of source views during test time for the metrics: ℓ_1^{VGG}, ℓ_1^{Depth}, **IoU**. We report results for 1, 3, 5 and 7 source images. The best result is **bolded** where lower is better for ℓ_1^{VGG}, ℓ_1^{Depth} and higher is better for **IoU**.

		AMT								**Freiburg Cars**							
		Train-test				**Test**				**Train-test**				**Test**			
	Method	1	3	5	7	1	3	5	7	1	3	5	7	1	3	5	7
ℓ_1^{VGG}	Mesh	1.163	1.167	1.168	1.169	1.160	1.161	1.163	1.163	2.030	2.029	2.028	2.023	2.170	2.168	2.166	2.167
	Voxel	1.052	1.051	1.051	1.051	1.127	1.127	1.127	1.127	1.581	1.581	1.580	1.580	2.050	2.050	2.046	2.046
	Voxel+MLP	1.041	1.040	1.040	1.040	1.131	1.130	1.130	1.130	1.469	1.468	1.468	1.468	2.067	2.063	2.063	2.064
	MLP	**0.900**	0.899	0.899	0.899	1.130	1.130	1.130	1.131	**1.391**	1.389	1.389	1.389	2.027	2.025	2.024	2.025
	Ours	0.905	**0.846**	**0.837**	**0.832**	**1.007**	**0.921**	**0.896**	**0.883**	1.450	**1.381**	**1.372**	**1.359**	**1.945**	**1.897**	**1.874**	**1.863**
IoU	Mesh	0.599	0.599	0.599	0.598	0.598	0.598	0.598	0.598	0.601	0.604	0.605	0.606	0.556	0.556	0.556	0.556
	Voxel	0.776	0.777	0.777	0.777	0.660	0.660	0.660	0.661	0.891	0.892	0.892	0.893	0.517	0.511	0.509	0.510
	Voxel+MLP	0.775	0.776	0.777	0.776	0.652	0.654	0.654	0.654	0.878	0.878	0.878	0.878	0.540	0.541	0.542	0.541
	MLP	**0.871**	0.871	0.872	0.872	0.654	0.653	0.653	0.653	0.872	0.872	0.872	0.872	0.472	0.470	0.472	0.471
	Ours	0.866	**0.884**	**0.886**	**0.889**	**0.774**	**0.788**	**0.787**	**0.787**	**0.889**	**0.897**	**0.898**	**0.897**	**0.600**	**0.624**	**0.629**	**0.632**
ℓ_1^{Depth}	Mesh	5.138	5.119	5.128	5.130	5.100	5.101	5.090	5.086	1.202	1.185	1.178	1.177	**1.062**	**1.061**	**1.063**	**1.063**
	Voxel	2.150	2.141	2.140	2.141	3.069	3.064	3.067	3.065	0.591	0.590	0.585	0.583	2.133	2.181	2.207	2.200
	Voxel+MLP	1.958	1.942	1.942	1.941	2.881	2.868	2.861	2.864	**0.478**	0.479	0.479	0.479	1.972	1.979	1.968	1.968
	MLP	**1.389**	1.378	1.377	1.377	3.583	3.587	3.590	3.593	0.595	0.593	0.594	0.593	2.521	2.530	2.519	2.520
	Ours	1.593	**1.291**	**1.201**	**1.172**	**2.186**	**1.847**	**1.802**	**1.776**	0.535	**0.467**	**0.457**	**0.453**	1.606	1.595	1.589	1.603

B.1 Additional implementation details

In this section, we provide more detailed information about the dense image descriptors Φ as well as the neural radiance field Ψ. Furthermore, we give more insights into the training process.

B.1.1 Dense image descriptors

This section describes in more detail the dense pixel-wise embeddings $\Phi(I_t)$ introduced in Section 3.3 in the main paper.

For a given source image I_t, the embedding field $\Phi(I_t)$ is composed of 3 different types of features: 1) learned $5 \cdot 32$-dimensional dense pixel-wise features output by a deep convolutional encoder network $\Phi_{\text{U-Net}}$, 2) raw image rgb colors $I_t \in \mathbb{R}^{3\times H\times W}$, and 3) the segmentation mask $m_t \in \mathbb{R}^{1\times H\times W}$.

Dense feature extractor $\Phi_{\text{U-Net}}$. The architecture of the U-Net inside $\Phi_{\text{U-Net}}$ is defined as follows (a detailed visualisation is present in Fig. B.2). A source image $I^{\text{src}} \in \mathbb{R}^{3\times H\times W}$, masked by m^{src} (retrieved from Mask-RCNN), is fed into a ResNet-50, which returns spatial features from intermediate convolutional layers (*layer1, layer2, layer3, layer4, layer5*), and the final linear ResNet layer, which outputs global features $\mathbf{z}_{\text{CNN}}$, i.e. non-spatial. Each feature layer, including the global one, is then passed through a 1x1 convolution to equalize the size of all feature channels to 32. The spatial features are further bilinearly upsampled to the spatial size of the source image and concatenated along the channel dimension to create a dense embedding field $\Phi_{\text{U-Net}}(I_t) \in \mathbb{R}^{5\cdot 32\times H\times W}$.

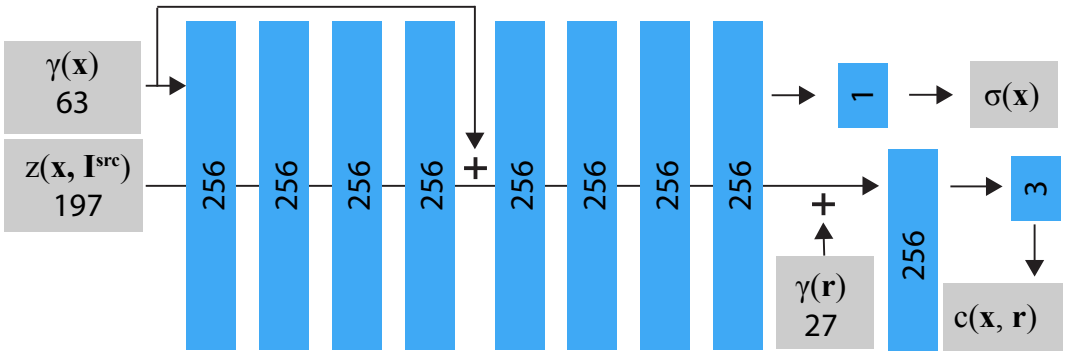


Figure B.1: The neural radiance field Ψ is represented by an MLP. It takes as input the warp-conditioned embedding $\mathbf{z}(\mathbf{x})$, the harmonic positional embedding $\gamma(\mathbf{x})$ and to account for viewpoint variations, the harmonic directional embedding $\gamma(\mathbf{r})$. It returns the rgb and opacity values.

Neural radiance field Ψ. Our scene is represented by a neural radiance field Ψ similar to [54] with the only difference that we additionally condition the field with our warp-conditioned ray embedding, see Fig. B.1.

B.1.2 Training details

We trained both the U-Net encoder $\Phi_{\text{U-Net}}$ and the neural radiance field Ψ with Adam optimizer. We set the batch size to 8 and the learning rate to 1e-4. Our method as well as all baselines were trained on an NVIDIA Tesla V100 for 7 days. For all raymarching baselines and our method, we shoot 1024 rays per iteration through random image pixels in Monte-Carlo fashion. For each ray, we first uniformly sample 128 times along the ray in order to retrieve a coarse rendering (voxel or mlp based depending on the method used). In the second pass, we sample each ray 128 times based on probabilistic importance sampling following [54].

For the mesh baseline, we shoot rays for each pixel per iteration and use soft rasterization to predict the surface intersection. In addition to the losses used for the other baselines as well as our method, we additionally use a negative IoU loss L_{iou}, a Laplacian loss L_{lap} and smoothness loss L_{sm} according to [191] and weighted them with 1.0, 19.0, 1.0 respectively.

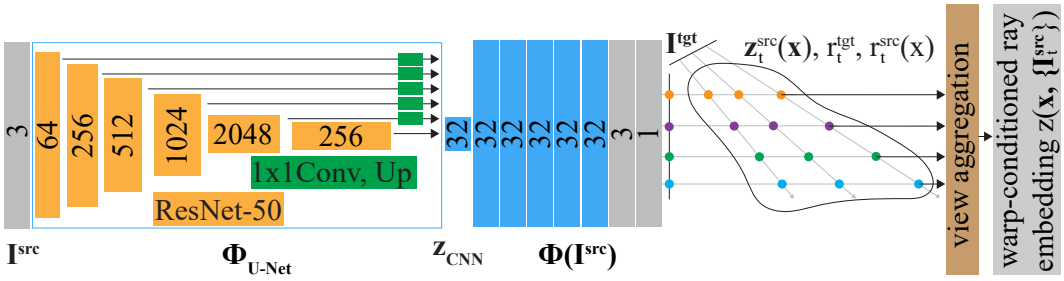


Figure B.2: The input to the dense feature extractor Φ is a source image from a given view. It first makes use of a ResNet-50 ($\Phi_{\text{U-Net}}$) to retrieve the layer-wise features. Then, each layer is independently fed to a 1x1 convolution followed by bilinear upsampling to the original input resolution. The resulting feature blocks are concatenated with the input image I^{src} and its corresponding object mask m^{src}. In case there are multiple source images available, this process is repeated for each of them. Once all per-view features are obtained, the warp-conditioned ray embedding is retrieved after applying the view-aggregation.

B.2 Additional qualitative results

Additional qualitative results are available and presented in Fig. B.4 and Fig. B.3. Also, we provide more qualitative results on our project webpage: `https://henzler.github.io/publication/unsupervised_videos/`. The

page contains comparison of our method to baselines by showing the scenes from the train-test or test subsets rendered from a viewpoint that rotates around the object of interest.

B.3 Test-time view ablation

Furthermore, we also provide a view ablation of our method at test time. Recall that we randomly sample between 1 and 7 source images during training. During test time we evaluated our method separately on 1, 3, 5 and 7 views as input. In Chapter 4, we provide an average of those numbers. In Tbl. B.1 we give insight into how changing the number of source views affects performance. Not surprisingly, increasing the number of source views consistently improves all metrics.

Figure B.3: In each row, a single source image (1st column) is processed by one of the evaluated methods (Mesh, Voxel, MLP+Voxel, MLP, **Ours** - columns 2 to 6) to generate a prescribed target view (last column). We show results on the test split.

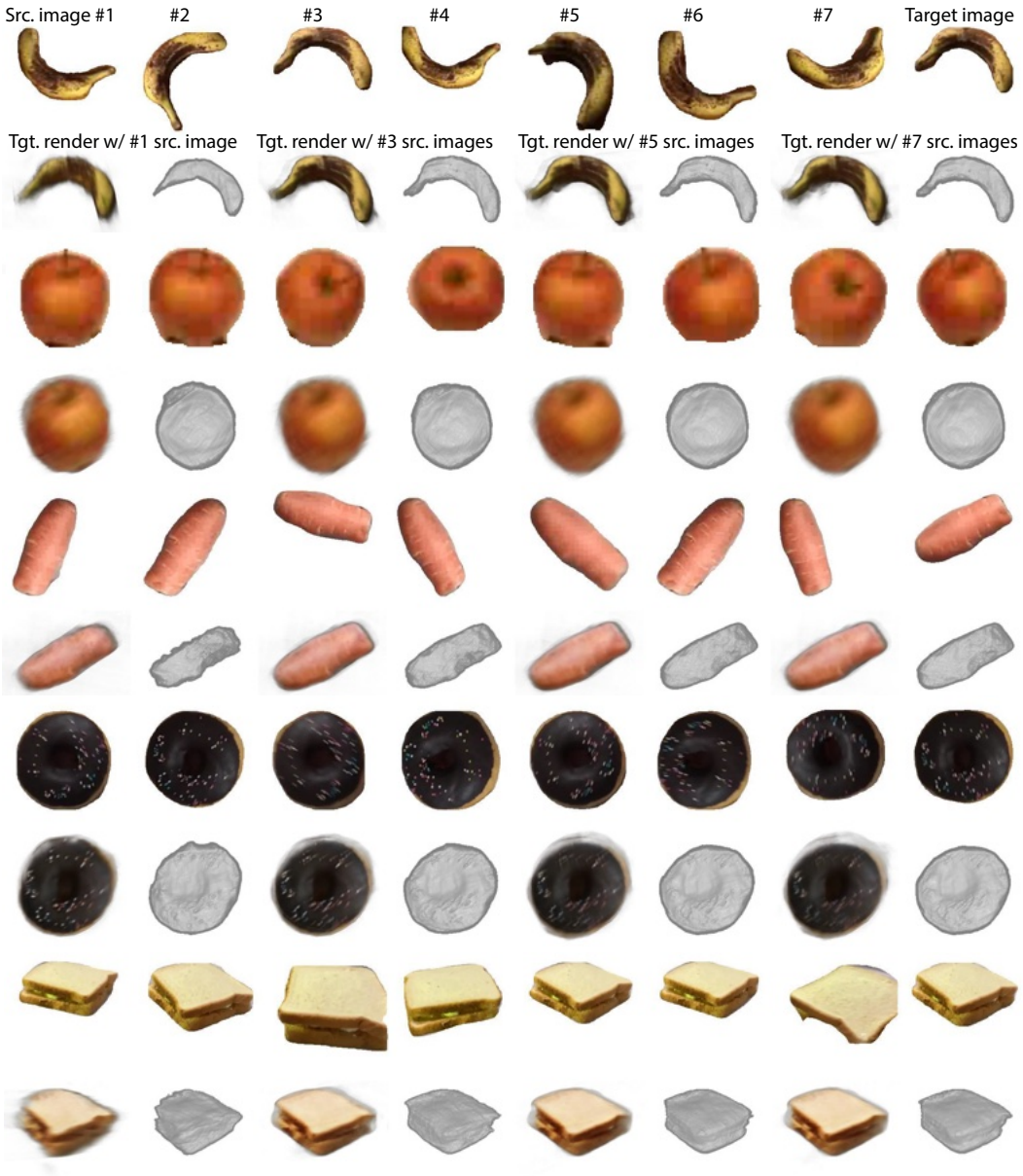


Figure B.4: Reconstruction with multiple source views. The top row for each object shows all available source images (columns 1-7) for a given target image (top right). The bottom row contains results conditioned on 1, 3, 5 or 7 source images. In addition to the rendered new RGB views we also provide shaded surface renderings.

Appendix C

Learning a Neural 3D Texture Space from 2D Exemplars

C.1 Network Architecture

C.1.1 Encoder

The architecture for the encoder network remains consistent for both ours and competitor methods. Depending on training for *space*, *single*, *w/o transform* the parameter N changes accordingly.

Table C.1: Network architecture for encoder.

Layer	Kernel	Activation	Shape	# params
Input	—	—	3 x 128 x 128	—
Conv	3x3	IN+LReLU	32 x 128 x 128	∼1k
Conv	4x4	IN+LReLU	64 x 64 x 64	∼32k
Conv	4x4	IN+LReLU	128 x 32 x 32	∼130k
Conv	4x4	IN+LReLU	256 x 16 x 16	∼524k
Conv	4x4	IN+LReLU	256 x 8 x 8	∼1M
Conv	4x4	IN+LReLU	256 x 4 x 4	∼1M
Linear	—	—	8	∼32k
Linear	—	—	N	∼0.5k
# params	—	—		∼2.8M

C.1.2 Sampler

The sampler architecture used for both our and the *mlp* [131] method consists of following convolutional architecture with 1x1 kernels emulating Linear layers:

Table C.2: Network architecture for sampler.

Layer	Kernel	Activation	Shape	# params
Input	—	—	N x 128 x 128	—
Conv	1x1	ReLU	128 x 128 x 128	∼10k
Conv	1x1	ReLU	128 x 128 x 128	∼16.5k
Conv	1x1	ReLU	128 x 128 x 128	∼16.5k
Conv	1x1	ReLU	128 x 128 x 128	∼16.5k
Conv	1x1	ReLU	128 x 128 x 128	∼16.5k
Conv	1x1	ReLU	3 x 128 x 128	∼400
# params	—	—		∼77k

C.1.3 CNN

For *cnn* and *cnnD* competitors we use a similar architecture to the proposed method of [128]:

Table C.3: Network architecture for convolutional methods.

Layer	Kernel	Activation	Shape	# params
Input	—	—	(32) + 256	—
Linear	—	—	(32) + 256	∼80k
Linear	—	—	256	∼70k
Reshape	—	—	16 x 4 x 4	—
ConvT	4x4	ReLU	128 x 8 x 8	∼32k
ConvT	4x4	ReLU	128 x 16 x 16	∼260k
ConvT	4x4	ReLU	128 x 32 x 32	∼260k
Upsample	—	—	128 x 64 x 64	—
Conv	3x3	ReLU	64 x 64 x 64	∼70k
Upsample	—	—	64 x 128 x 128	—
Conv	3x3	ReLU	3 x 128 x 128	∼2k
# params	—	—		∼790k

C.2 Results

Additional results of stripe images and interpolations are displayed below.

A webpage containing more results for all four classes (WOOD, MARBLE, GRASS and RUST) including competitors can be accessed online: `https://geometry.cs.ucl.ac.uk/projects/2020/neuraltexture`. Additionally, videos of rotating shapes textured by our method are provided. Our code is available at: `https://github.com/henzler/neuraltexture`

Figure C.1: Results derived from the encoded WOOD space.

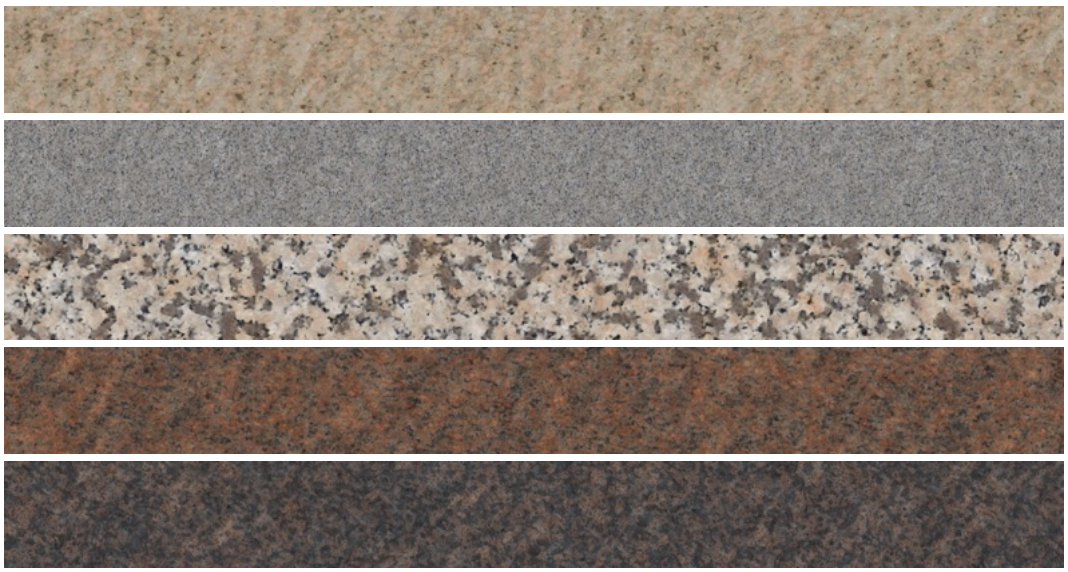


Figure C.2: Results derived from the encoded MARBLE space.

Figure C.3: Results derived from the encoded GRASS space.

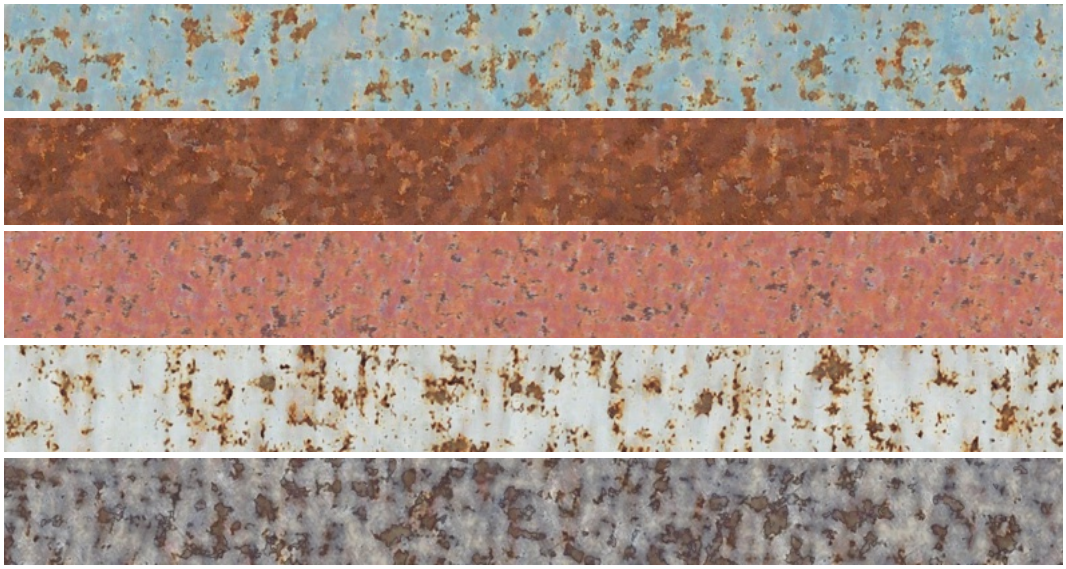


Figure C.4: Results derived from the encoded RUST space.

Figure C.5: Latent space interpolation from one ground truth wood exemplar (left) into secondary ground truth exemplar (right). Each row corresponds to independent interpolations.

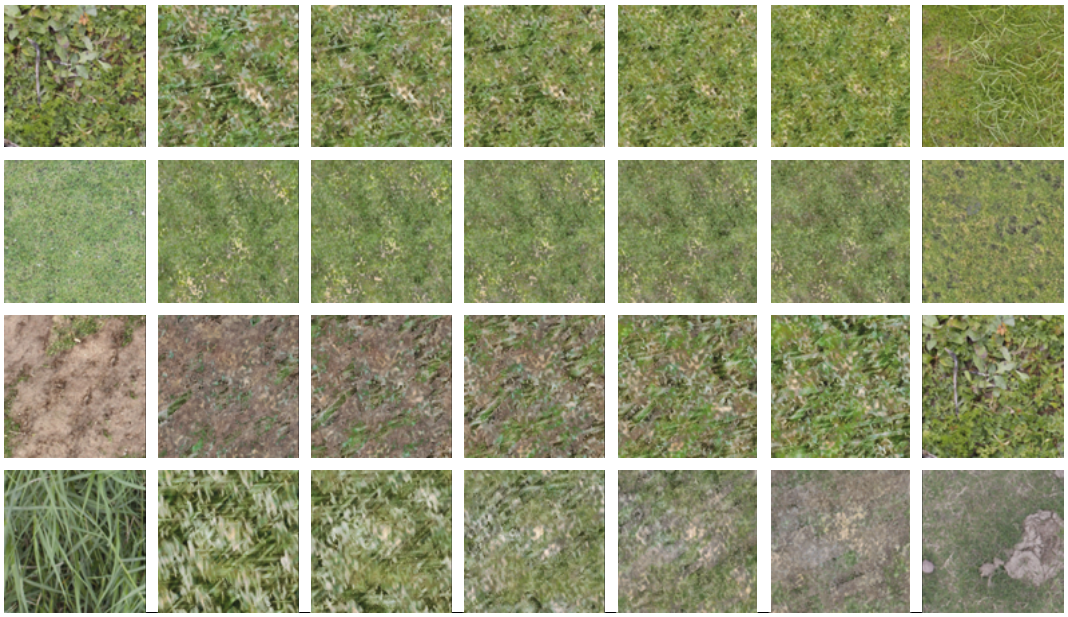


Figure C.6: Latent space interpolation from one ground truth grass exemplar (left) into secondary ground truth exemplar (right). Each row corresponds to independent interpolations.

Figure C.7: Latent space interpolation from one ground truth marble exemplar (left) into secondary ground truth exemplar (right). Each row corresponds to independent interpolations.

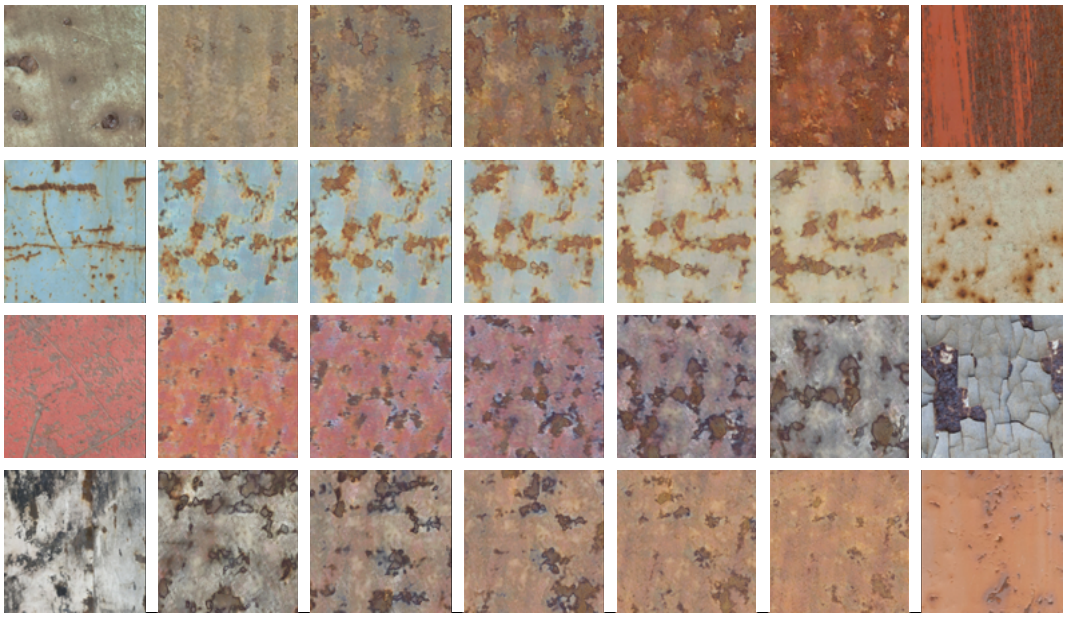


Figure C.8: Latent space interpolation from one ground truth rust exemplar (left) into secondary ground truth exemplar (right). Each row corresponds to independent interpolations.

Bibliography

[1] **Philipp Henzler**, Niloy Mitra, and Tobias Ritschel. Escaping plato's cave using adversarial training: 3d shape from unstructured 2d image collections. In *Proc. ICCV*, 2019.

[2] **Philipp Henzler**, Jeremy Reizenstein, Patrick Labatut, Roman Shapovalov, Tobias Ritschel, Andrea Vedaldi, and David Novotny. Unsupervised learning of 3d object categories from videos in the wild. In *Proc. CVPR*, 2021.

[3] **Philipp Henzler**, Niloy J Mitra, and Tobias Ritschel. Learning a neural 3d texture space from 2d exemplars. In *Proc. CVPR*, 2020.

[4] **Philipp Henzler**, Valentin Deschaintre, Niloy J Mitra, and Tobias Ritschel. Generative modelling of brdf textures from flash images. *ACM Trans Graph (Proc. SIGGRAPH Asia)*, 40(6):195–206, 2021.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[6] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proc. CVPR*, 2015.

[7] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. Region-based semantic segmentation with end-to-end training. In *Proc. ECCV*, 2016.

[8] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. CVPR*, 2014.

[9] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Region-based convolutional networks for accurate object detection and segmentation. *PAMI*, 2015.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. CVPR*, 2009.

[11] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv:1512.03012*, 2015.

[12] Mike Roberts, Jason Ramapuram, Anurag Ranjan, Atulit Kumar, Miguel Angel Bautista, Nathan Paczan, Russ Webb, and Joshua M. Susskind. Hypersim: A photorealistic synthetic dataset for holistic indoor scene understanding. In *Proc. ICCV*, 2021.

[13] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *International Conference on 3D Vision (3DV)*, 2017.

[14] Sungjoon Choi, Qian-Yi Zhou, Stephen Miller, and Vladlen Koltun. A large dataset of object scans. *arXiv:1602.02481*, 2016.

[15] Robert L Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *ACM SIGGRAPH computer graphics*, 1984.

[16] Per H Christensen, Wojciech Jarosz, et al. The path to path-traced movies. *Foundations and Trends® in Computer Graphics and Vision*, 10(2):103–175, 2016.

[17] Per Christensen, Julian Fong, Jonathan Shade, Wayne Wooten, Brenden Schubert, Andrew Kensler, Stephen Friedman, Charlie Kilpatrick, Cliff Ramshaw, Marc Bannister, et al. Renderman: An advanced path-tracing architecture for movie rendering. *ACM Transactions on Graphics (TOG)*, 37(3):30, 2018.

[18] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[19] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[20] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[21] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[22] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[23] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[24] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.

[25] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. ICLR*, 2015.

[26] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[27] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, pages 2672–80, 2014.

[28] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[29] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[30] Ruo Zhang, Ping-Sing Tsai, James Edwin Cryer, and Mubarak Shah. Shape-from-shading: a survey. *IEEE transactions on pattern analysis and machine intelligence*, 21(8):690–706, 1999.

[31] Fred E Nicodemus. Directional reflectance and emissivity of an opaque surface. *Applied optics*, 4(7):767–775, 1965.

[32] Christopher Schwartz, Ralf Sarlette, Michael Weinmann, and Reinhard Klein. Dome ii: A parallelized btf acquisition system. In *MAM*, pages 25–31, 2013.

[33] Bui Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975.

[34] James F Blinn. Models of light reflection for computer synthesized pictures. In *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, pages 192–198, 1977.

[35] Hermann Von Helmholtz. *Handbuch der physiologischen Optik: mit 213 in den Text eingedruckten Holzschnitten und 11 Tafeln*, volume 9. Voss, 1867.

[36] Kenneth E Torrance and Ephraim M Sparrow. Theory for off-specular reflection from roughened surfaces. *Josa*, 57(9):1105–1114, 1967.

[37] Robert L Cook and Kenneth E Torrance. A reflectance model for computer graphics. *ACM Trans Graph*, 1(1):7–24, 1982.

[38] Bruce Walter, Stephen R Marschner, Hongsong Li, and Kenneth E Torrance. Microfacet models for refraction through rough surfaces. In *Proc. EGSR*, pages 195–206, 2007.

[39] Eric Heitz. Understanding the masking-shadowing function in microfacet-based brdfs. *Journal of Computer Graphics Techniques*, 2014.

[40] Christophe Schlick. An inexpensive brdf model for physically-based rendering. In *Comp Graph Forum*, 1994.

[41] James T Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, 1986.

[42] Eric Lafortune. Mathematical models and monte carlo algorithms for physically based rendering. *Department of Computer Science, Faculty of Engineering, Katholieke Universiteit Leuven*, 20:74–79, 1996.

[43] James F Blinn and Martin E Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542–547, 1976.

[44] Robert A Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. In *Siggraph Computer Graphics*, 1988.

[45] Shimon Ullman. The interpretation of structure from motion. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 203(1153): 405–426, 1979.

[46] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4104–4113, 2016.

[47] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *Proc. IJCAI*, 1981.

[48] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[49] Andreas Geiger, Julius Ziegler, and Christoph Stiller. Stereoscan: Dense 3d reconstruction in real-time. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 963–968. Ieee, 2011.

[50] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise View Selection for Unstructured Multi-View Stereo. In *European Conference on Computer Vision (ECCV)*, 2016.

[51] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 279–288, 1993.

[52] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 39–46, 1995.

[53] Andrew Fitzgibbon, Yonatan Wexler, and Andrew Zisserman. Image-based rendering using image-based priors. *International Journal of Computer Vision*, 63(2):141–151, 2005.

[54] 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. *Proc. ECCV*, 2020.

[55] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[56] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[57] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999.

[58] Richard Szeliski et al. Image alignment and stitching: A tutorial. *Foundations and Trends® in Computer Graphics and Vision*, 2(1):1–104, 2007.

[59] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42, 1996.

[60] James R Bergen and Edward H Adelson. The plenoptic function and the elements of early vision. *Computational models of visual processing*, 1:8, 1991.

[61] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54, 1996.

[62] Sing Bing Kang, Yin Li, Xin Tong, Heung-Yeung Shum, et al. Image-based rendering. *Foundations and Trends® in Computer Graphics and Vision*, 2(3): 173–258, 2007.

[63] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. In *Proc. NeurIPS*, 2020.

[64] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. *Proc. NIPS*, 2020.

[65] Christopher Xie, Keunhong Park, Ricardo Martin-Brualla, and Matthew Brown. Fig-nerf: Figure-ground neural radiance fields for 3d object category modelling. In *2021 International Conference on 3D Vision (3DV)*, pages 962–971. IEEE, 2021.

[66] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D Shapenets: A deep representation for volumetric shapes. In *CVPR*, pages 1912–20, 2015.

[67] Bo Yang, Hongkai Wen, Sen Wang, Ronald Clark, Andrew Markham, and Niki Trigoni. 3d object reconstruction from a single depth view with adversarial learning. *arXiv preprint arXiv:1708.07969*, 2017.

[68] Jacob Varley, Chad DeChant, Adam Richardson, Joaquín Ruales, and Peter Allen. Shape completion enabled robotic grasping. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 2442–2447. IEEE, 2017.

[69] Jiajun Wu, Yifan Wang, Tianfan Xue, Xingyuan Sun, Bill Freeman, and Josh Tenenbaum. MarrNet: 3D shape reconstruction via 2.5D sketches. In *NIPS*, pages 540–550, 2017.

[70] Hanqing Wang, Jiaolong Yang, Wei Liang, and Xin Tong. Deep single-view 3D object reconstruction with visual hull embedding. *arXiv:1809.03451*, 2018.

[71] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In *NIPS*, pages 82–90, 2016.

[72] Rohit Girdhar, David F Fouhey, Mikel Rodriguez, and Abhinav Gupta. Learning a predictable and generative vector representation for objects. In *ECCV*, pages 484–99, 2016.

[73] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3D-R2N2: A unified approach for single and multi-view 3D object reconstruction. In *ECCV*, pages 628–44, 2016.

[74] Abhishek Kar, Christian Häne, and Jitendra Malik. Learning a multi-view stereo machine. In *NIPS*, pages 365–376, 2017.

[75] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proc. CVPR*, pages 4460–4470, 2019.

[76] Georgia Gkioxari, Justin Johnson, and Jitendra Malik. Mesh R-CNN. In *Proc. ICCV*, 2019.

[77] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proc. ECCV*, 2018.

[78] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *CVPR*, 2017.

[79] Danilo Jimenez Rezende, SM Ali Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg, and Nicolas Heess. Unsupervised learning of 3D structure from images. In *NIPS*, pages 4996–5004, 2016.

[80] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[81] Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective transformer nets: Learning single-view 3D object reconstruction without 3D supervision. In *NIPS*, pages 1696–1704, 2016.

[82] Matthew M. Loper and Michael J. Black. OpenDR: An approximate differentiable renderer. In *ECCV*, volume 8695, pages 154–69, 2014.

[83] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3D mesh renderer. In *CVPR*, pages 3907–16, 2018.

[84] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Proc. ICCV*, 2019.

[85] Wenzheng Chen, Huan Ling, Jun Gao, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. In *Proc. NeurIPS*, pages 9609–9619, 2019.

[86] Shubham Tulsiani, Tinghui Zhou, Alexei A Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *CVPR*, 2017.

[87] Thomas J Cashman and Andrew W Fitzgibbon. What shape are dolphins? building 3D morphable models from 2D images. *PAMI*, 35(1):232–44, 2013.

[88] Sara Vicente, Joao Carreira, Lourdes Agapito, and Jorge Batista. Reconstructing PASCAL VOC. In *Proc. CVPR*, 2014.

[89] Joao Carreira, Sara Vicente, Lourdes Agapito, and Jorge Batista. Lifting object detection datasets into 3d. *IEEE PAMI*, 38(7):1342–55, 2016.

[90] Angjoo Kanazawa, Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. Learning category-specific mesh reconstruction from image collections. In *Proc. ECCV*, 2018.

[91] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[92] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *CVPR*, 2017.

[93] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *CVPR*, pages 6602–6611, 2017.

[94] Shubham Tulsiani, Alexei A Efros, and Jitendra Malik. Multi-view consistency as supervisory signal for learning shape and pose prediction. In *CVPR*, pages 2897–2905, 2018.

[95] Eldar Insafutdinov and Alexey Dosovitskiy. Unsupervised learning of shape and pose with differentiable point clouds. In *Proc. NeurIPS*, pages 2802–2812, 2018.

[96] David Novotny, Diane Larlus, and Andrea Vedaldi. Learning 3d object categories by looking around them. In *Proc. ICCV*, 2017.

[97] David Novotný, Diane Larlus, and Andrea Vedaldi. Capturing the geometry of object categories from video supervision. *PAMI*, 2018.

[98] Zeng Huang, Tianye Li, Weikai Chen, Yajie Zhao, Jun Xing, Chloe LeGendre, Linjie Luo, Chongyang Ma, and Hao Li. Deep volumetric video from very sparse multi-view performance capture. In *Proc. ECCV*, pages 336–354, 2018.

[99] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *Proc. ICCV*, October 2019.

[100] Shunsuke Saito, Tomas Simon, Jason Saragih, and Hanbyul Joo. Pifuhd: Multi-level pixel-aligned implicit function for high-resolution 3d human digitization. In *Proc. CVPR*, June 2020.

[101] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. *arXiv*, 2020.

[102] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul Srinivasan, Howard Zhou, Jonathan T Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. Ibrnet: Learning multi-view image-based rendering. *arXiv*, 2021.

[103] Nilesh Kulkarni, Abhinav Gupta, and Shubham Tulsiani. Canonical surface mapping via geometric cycle consistency. In *Proc. ICCV*, 2019.

[104] Nilesh Kulkarni, Abhinav Gupta, David F. Fouhey, and Shubham Tulsiani. Articulation-aware canonical surface mapping. In *Proc. CVPR*, 2020.

[105] Shubham Goel, Angjoo Kanazawa, and Jitendra Malik. Shape and viewpoint without keypoints. *Proc. ECCV*, 2020.

[106] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proc. CVPR*, 2019.

[107] Yuxuan Zhang, Wenzheng Chen, Huan Ling, Jun Gao, Yinan Zhang, Antonio Torralba, and Sanja Fidler. Image gans meet differentiable rendering for inverse graphics and interpretable 3d neural rendering. In *Proc. ICLR*, 2021.

[108] Shangzhe Wu, Christian Rupprecht, and Andrea Vedaldi. Unsupervised learning of probably symmetric deformable 3d objects from images in the wild. In *Proc. CVPR*, 2020.

[109] Xueting Li, Sifei Liu, Kihwan Kim, Shalini De Mello, Varun Jampani, Ming-Hsuan Yang, and Jan Kautz. Self-supervised single-view 3d reconstruction via semantic consistency. In *Proc. ECCV*, 2020.

[110] Wei-Chih Hung, Varun Jampani, Sifei Liu, Pavlo Molchanov, Ming-Hsuan Yang, and Jan Kautz. Scops: Self-supervised co-part segmentation. In *Proc. CVPR*, 2019.

[111] Matheus Gadelha, Subhransu Maji, and Rui Wang. 3d shape induction from 2d views of multiple objects. In *3DV*, 2016.

[112] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. In *Proc. NeurIPS*, 2020.

[113] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. HoloGAN: Unsupervised learning of 3D representations from natural images. In *Proc. ICCV*, 2019.

[114] Bela Julesz. Texture and visual perception. *Scientific American*, 212(2), 1965.

[115] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. In *Proc. CVPR*, 2014.

[116] Benoit B Mandelbrot. *The fractal geometry of nature*, volume 173. WH Freeman New York, 1983.

[117] Ken Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3), 1985.

[118] Javier Portilla and Eero P Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *Int J Comp Vis*, 40(1):49–70, 2000.

[119] Robert L Cook and Tony DeRose. Wavelet noise. *ACM Trans Graph*, 24(3): 803–11, 2005.

[120] Alexei A Efros and Thomas K Leung. Texture synthesis by non-parametric sampling. In *ICCV*, 1999.

[121] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proc. SIGGRAPH*, 2000.

[122] Vivek Kwatra, Irfan Essa, Aaron Bobick, and Nipun Kwatra. Texture optimization for example-based synthesis. In *ACM Trans. Graph.*, 2005.

[123] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 28(3):24, 2009.

[124] Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *NIPS*, 2015.

[125] Omry Sendik and Daniel Cohen-Or. Deep correlations for texture synthesis. *ACM Trans. Graph.*, 36(5):161, 2017.

[126] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, 2016.

[127] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016.

[128] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *CVPR*, 2017.

[129] Aaron Hertzmann, Charles E Jacobs, Nuria Oliver, Brian Curless, and David H Salesin. Image analogies. In *Proc. SIGGRAPH*, 2001.

[130] Jun-Yan Zhu, Zhoutong Zhang, Chengkai Zhang, Jiajun Wu, Antonio Torralba, Josh Tenenbaum, and Bill Freeman. Visual object networks: Image generation with disentangled 3D representations. In *NIPS*, 2018.

[131] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. Texture fields: Learning texture representations in function space. In *ICCV*, 2019.

[132] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. PiFu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *CVPR*, pages 2304–2314, 2019.

[133] Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. SinGAN: Learning a generative model from a single natural image. In *ICCV*, 2019.

[134] Urs Bergmann, Nikolay Jetchev, and Roland Vollgraf. Learning texture manifolds with the periodic spatial gan. In *J MLR*, pages 469–477, 2017.

[135] Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Non-stationary texture synthesis by adversarial expansion. *arXiv:1805.04487*, 2018.

[136] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, pages 4401–4410, 2019.

[137] Wenqi Xian, Patsorn Sangkloy, Varun Agrawal, Amit Raj, Jingwan Lu, Chen Fang, Fisher Yu, and James Hays. TextureGAN: Controlling deep image synthesis with texture patches. In *CVPR*, 2018.

[138] Ning Yu, Connelly Barnes, Eli Shechtman, Sohrab Amirghodsi, and Michal Lukac. Texture Mixer: A network for controllable synthesis and interpolation of texture. In *CVPR*, 2019.

[139] Johannes Kopf, Chi-Wing Fu, Daniel Cohen-Or, Oliver Deussen, Dani Lischinski, and Tien-Tsin Wong. Solid texture synthesis from 2D exemplars. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 26(3):2, 2007.

[140] Nico Pietroni, Miguel A Otaduy, Bernd Bickel, Fabio Ganovelli, and Markus Gross. Texturing internal surfaces from a few cross sections. In *Comp. Graph. Forum*, 2007.

[141] Darya Guarnera, Giuseppe Claudio Guarnera, Abhijeet Ghosh, Cornelia Denk, and Mashhuda Glencross. Brdf representation and acquisition. In *Comp Graph Forum*, 2016.

[142] W Matusik. A data-driven reflectance model. *ACM Trans Graph*, 22(3): 759–769, 2003.

[143] K. Rematas, T. Ritschel, M. Fritz, E. Gavves, and T. Tuytelaars. Deep reflectance maps. In *CVPR*, 2016.

[144] Stamatios Georgoulis, Konstantinos Rematas, Tobias Ritschel, Efstratios Gavves, Mario Fritz, Luc Van Gool, and Tinne Tuytelaars. Reflectance and natural illumination from single-material specular objects using deep learning. *PAMI*, 40(8):1932–1947, 2017.

[145] Valentin Deschaintre, Miika Aittala, Fredo Durand, George Drettakis, and Adrien Bousseau. Single-image svbrdf capture with a rendering-aware deep network. *ACM Trans Graph (Proc. SIGGRAPH)*, 2018.

[146] Giljoo Nam, Joo Ho Lee, Diego Gutierrez, and Min H. Kim. Practical svbrdf acquisition of 3d objects with unstructured flash photography. *ACM Trans. Graph.*, 37(6), 2018.

[147] Zhengqin Li, Zexiang Xu, Ravi Ramamoorthi, Kalyan Sunkavalli, and Manmohan Chandraker. Learning to reconstruct shape and spatially-varying reflectance from a single image. In *SIGGRAPH Asia 2018*, page 269. ACM, 2018.

[148] Mark Boss, Varun Jampani, Kihwan Kim, Hendrik P.A. Lensch, and Jan Kautz. Two-shot spatially-varying brdf and shape estimation. In *Proc. CVPR*, 2020.

[149] Valentin Deschaintre, Yiming Lin, and Abhijeet Ghosh. Deep polarization imaging for 3d shape and svbrdf acquisition. In *Proc. CVPR*, 2021.

[150] Xiao Li, Yue Dong, Pieter Peers, and Xin Tong. Synthesizing 3d shapes from silhouette image collections using multi-projection generative adversarial networks. In *Proc. CVPR*, 2019.

[151] Wenjie Ye, Xiao Li, Yue Dong, Pieter Peers, and Xin Tong. Single image surface appearance modeling with self-augmented cnns and inexact supervision. *Comp Graph Forum*, 37(7):201–11, 2018.

[152] Yiwei Hu, Julie Dorsey, and Holly Rushmeier. A novel framework for inverse procedural texture modeling. *ACM Trans. Graph.*, 38(6), 2019. ISSN 0730-0301.

[153] Yu Guo, Miloš Hašan, Lingqi Yan, and Shuang Zhao. A bayesian inference framework for procedural material parameter estimation. *Comp Graph Forum*, 39(7), 2020.

[154] Liang Shi, Beichen Li, Miloš Hašan, Kalyan Sunkavalli, Tamy Boubekeur, Radomir Mech, and Wojciech Matusik. Match: Differentiable material graphs for procedural material capture. *ACM Trans. Graph.*, 39(6), 2020.

[155] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, pages 234–41, 2015.

[156] Xiao Li, Yue Dong, Pieter Peers, and Xin Tong. Modeling surface appearance from a single photograph using self-augmented convolutional neural networks. *ACM Trans Graph*, 36(4):45, 2017.

[157] Zhengqin Li, Kalyan Sunkavalli, and Manmohan Chandraker. Materials for masses: Svbrdf acquisition with a single mobile phone image. In *ECCV*, pages 72–87, 2018.

[158] Guilin Liu, Duygu Ceylan, Ersin Yumer, Jimei Yang, and Jyh-Ming Lien. Material editing using a physically based rendering network. In *ICCV*, pages 2261–2269, 2017.

[159] Valentin Deschaintre, Miika Aittala, Frédo Durand, George Drettakis, and Adrien Bousseau. Flexible svbrdf capture with a multi-image deep network. *Comp Graph Forum*, 38(4):1–13, 2019.

[160] Yu Guo, Cameron Smith, Miloš Hašan, Kalyan Sunkavalli, and Shuang Zhao. Materialgan: Reflectance capture using a generative svbrdf model. *ACM Trans Graph*, 39(6), 2020.

[161] Duan Gao, Xiao Li, Yue Dong, Pieter Peers, Kun Xu, and Xin Tong. Deep inverse rendering for high-resolution svbrdf estimation from an arbitrary number of images. *ACM Trans Graph (Proc. SIGGRAPH Asia)*, 38(4):134, 2019.

[162] Valentin Deschaintre, George Drettakis, and Adrien Bousseau. Guided fine-tuning for large-scale material transfer. *Comp Graph Forum (Proc. EGSR)*, 39(4), 2020.

[163] Xilong Zhou and Nima Khademi Kalantari. Adversarial Single-Image SVBRDF Estimation with Hybrid Training. *Computer Graphics Forum (Proc. Eurographics)*, 2021.

[164] Jie Guo, Shuichang Lai, Chengzhi Tao, Yuelong Cai, Lei Wang, Yanwen Guo, and Ling-Qi Yan. Highlight-aware two-stream network for single-image svbrdf acquisition. *ACM Trans Graph (Proc. SIGGRAPH)*, 40(4), 2021.

[165] Károly Zsolnai-Fehér, Peter Wonka, and Michael Wimmer. Gaussian material synthesis. *ACM Trans. Graph.*, 37(4), 2018.

[166] Miika Aittala, Timo Aila, and Jaakko Lehtinen. Reflectance modeling by neural texture synthesis. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 35(4):65, 2016.

[167] Chuong H Nguyen, Tobias Ritschel, and Hans-Peter Seidel. Data-driven color manifolds. *ACM Trans Graph*, 34(2):20, 2015.

[168] Wojciech Matusik, Matthias Zwicker, and Frédo Durand. Texture design using a simplicial complex of morphable textures. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 24(3), 2005.

[169] Volker Blanz, Thomas Vetter, et al. A morphable model for the synthesis of 3d faces. In *Siggraph*, 1999.

[170] Brett Allen, Brian Curless, and Zoran Popović. The space of human body shapes: reconstruction and parameterization from range scans. *ACM Trans Graph (Proc. SIGGRAPH)*, 22(3):587–94, 2003.

[171] Keunhong Park, Konstantinos Rematas, Ali Farhadi, and Steven M Seitz. Photoshape: photorealistic materials for large-scale shape collections. *ACM Transactions on Graphics (TOG)*, 37(6):192, 2019.

[172] Ana Serrano, Diego Gutierrez, Karol Myszkowski, Hans-Peter Seidel, and Belen Masia. An intuitive control space for material appearance. *ACM Trans Graph (Proc. SIGGRAPH Asia)*, 35(6), 2016.

[173] Eric H Warmington, Philip G Rouse, and WHD Rouse. *Great dialogues of Plato*. New American Library, 1956.

[174] Angjoo Kanazawa, Shubham Tulsiani, Alexei A Efros, and Jitendra Malik. Learning category-specific mesh reconstruction from image collections. In *Proc. ECCV*, pages 371–386, 2018.

[175] SM Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S Morcos, Marta Garnelo, Avraham Ruderman, Andrei A Rusu, Ivo Danihelka,

Karol Gregor, et al. Neural scene representation and rendering. *Science*, 360 (6394):1204–10, 2018.

[176] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

[177] Aldo Laurentini. The visual hull concept for silhouette-based image understanding. *AMI*, 16(2):150–62, 1994.

[178] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An information-rich 3D model repository. *arXiv:1512.03012*, 2015.

[179] **Philipp Henzler**, Volker Rasche, Timo Ropinski, and Tobias Ritschel. Single-Image Tomography: 3D Volumes from 2D Cranial X-Rays. *Computer Graphics Forum (Proceedings of Eurographics 2018)*, 2018.

[180] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[181] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *NIPS*, pages 5767–5777, 2017.

[182] Zhou Wang, Alan C Bovik, Hamid R Sheikh, Eero P Simoncelli, et al. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[183] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proc. CVPR*, 2016.

[184] K. He, G. Gkioxari, and P. Dollár and. R. Girshick. Mask R-CNN. In *Proc. ICCV*, 2017.

[185] Nelson L. Max. Optical models for direct volume rendering. *IEEE Trans. Vis. Comput. Graph.*, 1995.

[186] Michael Niemeyer, Lars M. Mescheder, Michael Oechsle, and Andreas Geiger. Occupancy flow: 4d reconstruction by learning particle dynamics. In *Proc. ICCV*, 2019.

[187] David Novotný, Diane Larlus, and Andrea Vedaldi. Learning the semantic structure of objects from web supervision. In *Proceedings of the ECCV workshop on Geometry Meets Deep Learning*, 2016.

[188] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In *Proc. ECCV*, 2014.

[189] Nima Sedaghat and Tomas Brox. Unsupevised generation of a viewpoint annotated car dataset from videos. In *Proc. ICCV*, 2015.

[190] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. CVPR*, 2016.

[191] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501*, 2020.

[192] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *NIPS*, pages 2017–2025, 2015.

[193] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least squares conformal maps for automatic texture atlas generation. In *ACM Trans Graph.*, 2002.

[194] Yezi Zhao, Beibei Wang, Yanning Xu, Zheng Zeng, Lu Wang, and Nicolas Holzschuch. Joint SVBRDF recovery and synthesis from a single image using an unsupervised generative adversarial network. In *EGSR*, 2020.

[195] David J Heeger and James R Bergen. Pyramid-based texture analysis/synthesis. In *Proc. SIGGRAPH*, pages 229–38, 1995.

[196] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*, pages 1501–10, 2017.

[197] Gang Liu, Yann Gousseau, and Gui-Song Xia. Texture synthesis through convolutional neural networks and spectrum constraints. In *ICPR*, pages 3234–9, 2016.

[198] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv:1312.6114*, 2013.

[199] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Trans Graph (Proc. SIGGRAPH Asia)*, 38(6), 2019.

[200] Pratul P Srinivasan, Boyang Deng, Xiuming Zhang, Matthew Tancik, Ben Mildenhall, and Jonathan T Barron. Nerv: Neural reflectance and visibility fields for relighting and view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7495–7504, 2021.

[201] Sai Bi, Zexiang Xu, Pratul Srinivasan, Ben Mildenhall, Kalyan Sunkavalli, Miloš Hašan, Yannick Hold-Geoffroy, David Kriegman, and Ravi Ramamoorthi. Neural reflectance fields for appearance acquisition. *arXiv preprint arXiv:2008.03824*, 2020.

[202] Xiuming Zhang, Pratul P Srinivasan, Boyang Deng, Paul Debevec, William T Freeman, and Jonathan T Barron. Nerfactor: Neural factorization of shape and

reflectance under an unknown illumination. *ACM Transactions on Graphics (TOG)*, 40(6):1–18, 2021.

[203] Mark Boss, Raphael Braun, Varun Jampani, Jonathan T Barron, Ce Liu, and Hendrik Lensch. Nerd: Neural reflectance decomposition from image collections. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12684–12694, 2021.

[204] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

[205] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. *arXiv:1710.10196*, 2017.

[206] 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.