Learning non-stationary SVBRDFs using GANs and Differentiable Rendering

by

Budmonde Duinkharjav

B.S., Massachusetts Institute of Technology (2018)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2019

Author . . . .
Department of Electrical Engineering and Computer Science
May 24, 2019

Certified by . . . .
Frédo Durand
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Katrina LaCurts
Chairman, Master of Engineering Thesis Committee

Learning non-stationary SVBRDFs using GANs and Differentiable Rendering

by

Budmonde Duinkharjav

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2019, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

In this thesis we propose a learning approach for generating realistic SVBRDFs using generative adversarial models and differentiable rendering. Our model learns a mapping from the *geometry buffer* of a surface to a corresponding albedo texture-map by training on images of the same surface rendered using a target texture-map. A key feature of this learning process is the ability to differentiate the render function within our model; this enables the optimization of the texture-map generator parameters using a loss function computed from the rendered 2D images. Our results show that differentiable rendering is applicable in complex neural network models such as GANs, opening up opportunities for more applications of deep learning methods in the computer graphics pipeline.

Thesis Supervisor: Frédo Durand
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

Professor Frédo Durand and Dr. Jun-Yan Zhu for their great guidance and mentorship throughout my thesis project.

Dr. Tzu-Mao Li for providing general direction and feedback for my thesis project, tutoring me in the research process, and heping me debug my code in numerous occasions.

Spandan Madan, Helen Ho, and Nishchal Bhandari, for their work on CoSy which allowed me to bootstrap my project codebase and necessary data.

Everyone in the Computer Graphics Group, for tutoring me in my research experience and providing useful feedback.

Toyota for providing funding for this research opportunity.

Tuul Damba-Ochir and Sophia Kwon, for collecting data for my project, proofreading my thesis write-up, and providing moral support.

Contents

List of Figures

Chapter 1

Introduction

Fidelity of images produced by rendering pipelines is largely affected by the surface reflectance model used in the rendering process. Physically based reflectance models used in modern ray tracing engines help achieve a high level of realism. However, materials encountered on real surfaces are rarely uniform and feature spatially-varying reflectances due to natural causes such as wear and tear, weathering, and other surface imperfections. Unfortunately, using procedural techniques to replicate the complexity of real spatially-varying materials largely relies on handcrafted heuristics which produce results that are subjective in terms of realism. To record the complexity of real materials, previous methods [1, 2, 3, 8] use real photographs of simple flat surfaces to extract spatially-varying bidirectional reflectance distribution functions (SVBRDFs). However, extracting the SVBRDF for more complex surfaces with variable curvatures, such as dirt textures on car hulls, is a significantly harder task not attempted by these methods not only due to the complexity of the surface geometry, but also because of the non-stationary nature of the SVBRDFs; different areas of the surface would experience different amounts of weathering, causing the resultant SVBRDF to not exhibit shift invariance.

Exemplar based synthesis of complex textures is also studied in computer vision applications. The advent of the GAN [14] paved the way for high fidelity results from neural network based generative models for texture synthesis and style transfer [35, 17, 34]. Unfortunately, the deep learning approach requires that a model's building blocks consist of differentiable functions. Since differentiating more complex functions is often impractical, the majority of neural network models use simple functions with explicit analytical derivatives. However, if we have methods for differentiating complex functions, we enable opportunities to utilize deep learning methods in more complex tasks.

Recent work by Li et al. [21] features a differentiable ray tracer which makes it possible to take derivatives of images produced by the ray tracer with respect to the scene description parameters such as surface vertices, material properties, camera parameters, etc. The ability to take derivatives with respect to scene variables allows us to incorporate the ray tracing function into deep learning methods to train complex neural network models for generating surface geometry and textures.

In this thesis we demonstrate an application of a deep learning model for computer graphics by training a neural network which incorporates the differentiable ray tracer [21]. Specifically, we propose and implement a GAN that synthesizes SVBRDFs of complex surfaces (like a car hull) using a dataset of 2D images of the surfaces. Our model takes advantage of the differentiable ray tracer to compute a loss on a rendered image and back-propagate it to train a generator network which synthesizes a realistic albedo texture-map for a given 3D shape. We leave the synthesis of more complex SVBRDFs featuring specular reflectance, glossiness, and normal maps as future work.

We present our results when trained on a synthetic dataset. Our dataset consists of rendered images of car models with a target texture-map generated by a procedural surface weathering texture model used commonly in computer graphics applications [5]. Unfortunately, for the scope of this thesis, we are not able to extend our model to successfully learn texture-maps from real images.

Chapter 2

Related Work

2.1 Texture synthesis and style transfer

Our task shares similarities with texture synthesis and style transfer. In both tasks, the objective is to separate the information describing *what* the scene is from *how* it was depicted – what style the scene was created in. E.g. a dataset of paintings of architecture depicts some architectural scenery stylized as paintings. Texture synthesis focuses on extracting the style of an image while style transfer focuses on morphing the style of a scene. However, these tasks are ill-posed as the distinction between a scene and its style is not well defined. Some work in the literature acknowledge this limitation and propose methods that incorporate the distinction as part of the problem [17, 35] while others use perceptual judgment to optimize for the best results [11, 12, 32, 19]. Other solutions assume the input image contains only style content and attempt to produce similar images of the given texture [33, 30, 34].

Fortunately, in our case, the distinction between the scene and style is concretely defined; the scene is specified by the surface geometry, while the style is defined by the shading materials used in the rendering pipeline.

The state-of-the-art approach for the task of texture-synthesis and style-transfer is to train a Generative Adversarial Network [14] that generates the desired output [19, 17, 35, 34]. We base our network architecture from the CycleGAN model by Zhu et al. [35].

2.2 Physically-based appearance modeling

Work in appearance modeling use the geometry of the models to simulate physical phenomena, such as erosion of terrains [25], chemical diffusion [31], metallic patinas [10], wet materials [18], weathering (e.g. [9, 7, 6]). These methods generate convincing materials from geometry, but need to use dedicated simulation methods for different scenarios. In contrast our method is fully data-driven and does not rely on knowing the physics of the materials. Combining two approaches remain an interesting research avenue.

Some other methods assume flat geometry, but use a more elaborate material model (e.g. [4, 3, 8]). These methods deliver high-quality textures but do not take geometry into consideration.

2.3 Deep learning models for computer graphics

Applications of deep learning models in computer graphics literature is hindered due to the complex operations used in the graphics pipeline. Specifically, using 2D rendered images to infer an underlying 3D scene geometry relies on differentiating the rendering function. A generative model with a differentiable renderer makes it possible to optimize the model parameters using a loss function computed from a rendered image of a 3D scene which contains the generator output.

A variety of methods propose models for indirectly differentiating the rendering function by optimizing a neural network to emulate it [20, 27, 26]. Others propose differentiable rasterization algorithms tailored for their use case [13, 22]. To our best knowledge, there is no prior work in generative deep learning models featuring a general purpose ray tracer.

Our method incorporates the differentiable ray tracer by Li et al. [21] into our model. This ray tracer is able to compute derivatives of the Monte-Carlo path tracing algorithm for arbitrary scenes with surfaces represented as triangle-meshes with respect to any scene parameters such as triangle-mesh vertices, shading materials, and camera poses. The advantage of using this ray tracer over others is that we are able to extract any desired scene variable explicitly which is useful in most practical applications.

Chapter 3

Method

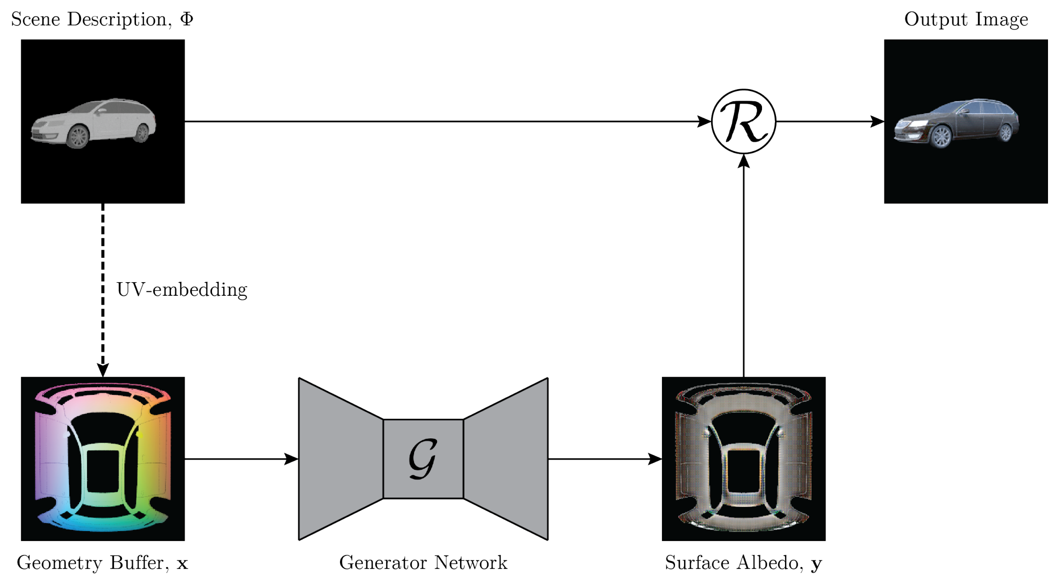
We take on the following task: given a dataset of images of weathered cars, learn a model that generates materials to approximate the appearance of weathering on a car surface. We focus on Lambertian materials in this thesis and represent the material with a semi-transparent albedo texture-map in the UV-embedding of the surface.

3.1 Differentiable Ray Tracing in Generative Models

The key contribution of this thesis is to frame this task as a learning problem and incorporate the differentiable ray tracing function [21] into the learning process. We present a schematic of our models in Figure 3-1 and Figure 3-2. We construct a convolutional neural network $\mathcal{G} : \mathbf{x} \rightarrow \mathbf{y}$, which, given a car's surface geometry $\mathbf{x}$, produces an albedo texture-map $\mathbf{y}$. We composit $\mathbf{y}$ onto the base material of the car and render it using the differentiable ray tracer $\mathcal{R}_\Phi : \mathbf{y} \rightarrow \mathbf{z}$ to obtain a rendered image $\mathbf{z}$ (Φ is the parameter set which describes the scene). The combined model $\mathcal{R}_\Phi \circ \mathcal{G}$ is fully differentiable, which allows us to train the network weights of $\mathcal{G}$ using a loss computed from the rendered image $\mathbf{z} = \mathcal{R}_\Phi \circ \mathcal{G}(\mathbf{x})$.

Although the render function is applicable for arbitrary network architectures, for the scope of this thesis we train a generative adversarial network (GAN). Specifically, along with the generator $\mathcal{G}$, our model also trains a discriminator $\mathcal{D}$ which learns to label a rendered image of a car based on whether the albedo texture-map of the

Figure 3-1: Our model pipeline takes as input a scene description Φ which contains: shape geometry, shading materials, camera parameters, and lighting parameters. The UV-embedding of the car triangle-mesh, allows us to extract the *position* and *normal* maps of the car in its UV-coordinates, which we refer to as the *geometry buffer*. The *geometry buffer* is used to generate the albedo of the dirt using our generator network $\mathcal{G}$. To produce the final render, we use a differentiable ray tracer $\mathcal{R}$ after compositing the dirt albedo onto the original shading material.


Scene Description, Φ
Output Image
$\mathcal{R}$
UV-embedding
$\mathcal{G}$
Geometry Buffer, $\mathbf{x}$
Generator Network
Surface Albedo, $\mathbf{y}$

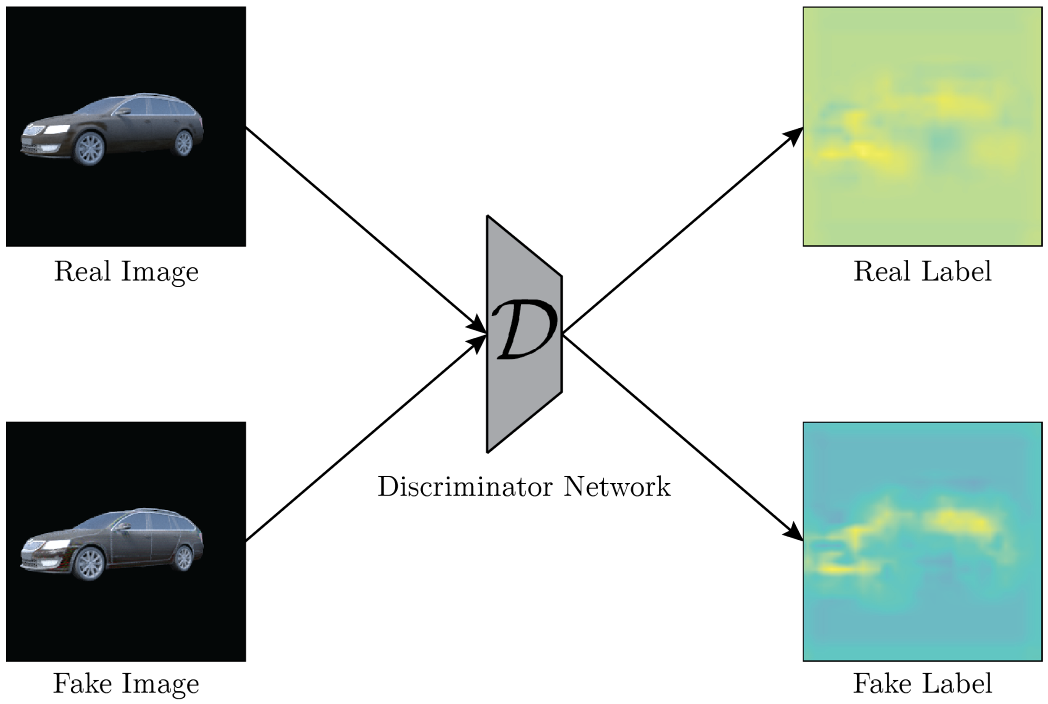
car was generated by $\mathcal{G}$ or by some ground-truth generator $\mathcal{G}_0$. $\mathcal{G}_0$ is a standard out-of-the-box procedural surface weathering texture-map generator [5].

3.2 Model Input and Surface Parameterization

So far we've discussed how $\mathcal{G}$ takes as input the shape geometry of a car $\mathbf{x}$ to produce a texture-map $\mathbf{y}$. For arbitrary generator models, the surface geometry can be represented using any parameterization. However, in order to utilize well tested CNN models such as [19], we require $\mathbf{x}$ to be in a representation that is compatible with 2D convolutions.

To re-parameterize $\mathbf{x}$ into a 2D grid structure, we bake $\mathbf{x}$ onto the surface's UV-embedding. I.e., given some UV-embededing for the input surface, for each UV-coordinate (u, v) we compute the surface's position vector $\mathbf{p}(u, v) \in \mathbb{R}^3$, and normal

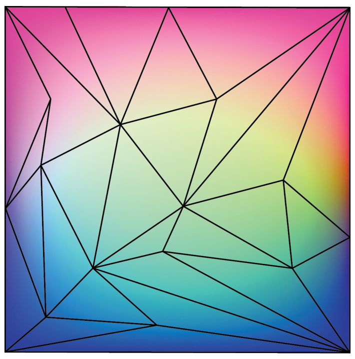
Figure 3-2: We use a GAN training procedure for our model. In addition to our generator model $\mathcal{G}$, we train a discriminator model $\mathcal{D}$ that learns to discriminate whether an image was produced by the generator (fake) or was sampled from the dataset (real). The discriminator produces a down-sampled grid of the input image, with each pixel representing the likelihood of the input being real or fake. In the visualization below, the correspondence is: yellow – real, cyan – fake.



vector $\mathbf{n}(u, v) \in \mathbb{R}^3$. Using $\mathbf{p}$ and $\mathbf{n}$, we form the surface *geometry buffer* $\mathbf{x} \in \mathbb{R}^6$ by concatenating the two vectors at each UV-coordinate.

The convolutions which $\mathcal{G}$ applies to the *geometry buffer* to generate the surface albedo are similar to using 3D shape convolutions as proposed by Masci et al. [24] with the caveat of the convolution filter size changing depending on the distortion of the UV-embedding. We provide more details about our choice of UV-embedding as well as a discussion about desired specifications of the UV-embedding in Section 4.1.

Figure 3-3: The *geometry buffer* is a mapping between UV-coordinates of the car triangle-mesh and corresponding values of surface coordinates **p** and shading normals **n**. We store these maps as two RGB encoded images, with the three channels representing each dimension of **p** and **n** respectively.



Chapter 4

Implementation

4.1 Surface UV-embedding

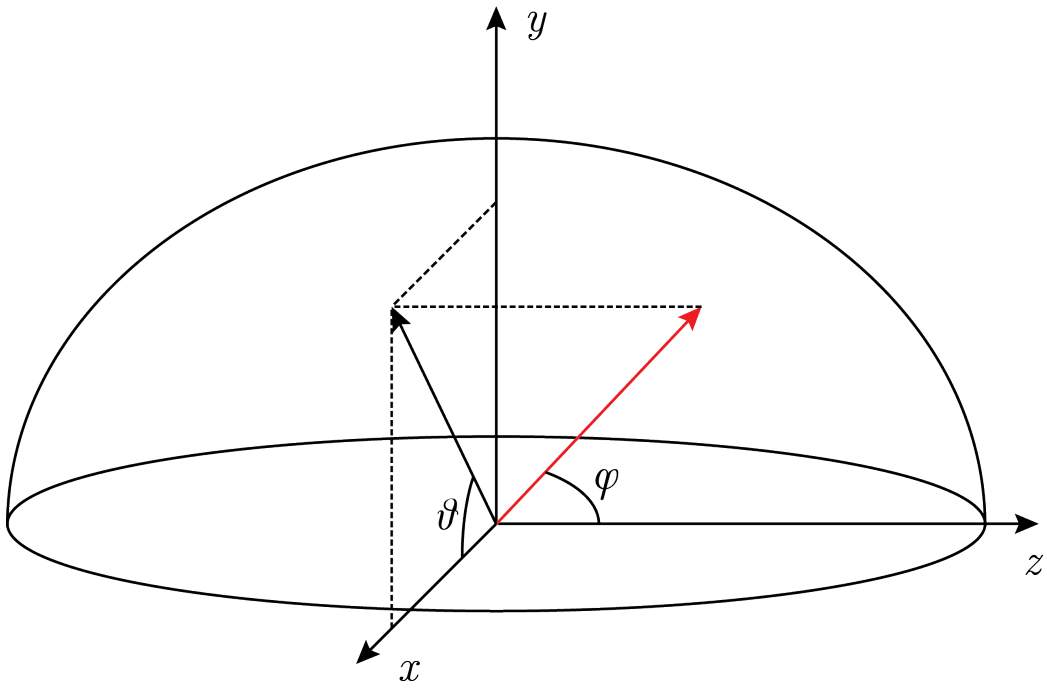
In Chapter 3, we briefly discussed that we construct the *geometry buffer* of a surface by using its UV-embedding. However, it is not obvious whether an arbitrary UV-embedding can be applied for our model. Ideally, the model accepts any valid UV-embedding and produces a corresponding albedo texture-map. However, there are several considerations that make certain embeddings more useful than others. A good embedding should have the following qualities:

– Similarity of embeddings for similar surfaces is an important feature required for generalization capabilities of our model. I.e., if we feed in two similar car surfaces, we should expect similar UV-embeddings. This similarity allows the generator to learn to generate similar texture-maps for similar surfaces.

– General compactness of the embedding is also very important since we're restricted to using convolutions; any unassigned coordinates in the UV-embedding will take up space in the convolution input while not being used to produce the final texture-map.

– Conservation of surface connectivity in the UV-embedding ensures that operations that operate in local neighborhoods, such as convolutions, respect

Figure 4-1: Our implementation features a *spherical UV-embedding* of the car surface. In this embedding we take the pair (θ, ϕ) as shown in the figure, and scaled to unity to specify a planar coordinate which corresponds to the surface. We use this embedding to specify a generalizable UV-embedding across all our car models.



the surface connectivity. Otherwise, discontinuities caused by seams in the UV-embedding would create discontinuities along the seam when a generated texture-map is composited onto a surface.

– Low distortion is important due to the static size of the convolution filter used in the model. Applying a filter on a stretched out area of the embedding will shrink the effective size of the filter on the surface manifold and vice versa.

For our implementation, we normalize the UV-embeddings across all car surfaces using a hemispherical parameterization. Given the (x, y, z) coordinates of a point in the standard Euclidean basis, we can compute its corresponding spherical coordinates (r, θ, ϕ), where the point is located a distance of r away from the origin and oriented by the azimuth angle θ and polar angle ϕ. Using this change of coordinates, we construct the hemispherical parameterization $(u, v) \in [0, 1]^2$ by discarding the radial distance r

and scaling θ and ϕ to a unit length. Since we're parameterizing a hemisphere, (as opposed to a whole sphere) we scale θ by π instead of 2π.

$$(u, v) = (\theta/\pi, \phi/\pi) \tag{4.1}$$

In general, the parameterization we described is not suitable for arbitrary shapes, especially if the shape has overlapping points along a radial direction as these points will be assigned to the same UV-coordinates. Fortunately, the roughly ellipsoid shape of car hulls allow us to use this parameterization with minimal artifacts.

This parameterization is (1) similar between similar surfaces as long as the center of each car model is standardized across models, (2) compact and roughly square since the surface is roughly shaped like an ellipsoid, and (3) reflects the connectivity of the surface well as our parameterization does not introduce any seams. Unfortunately, this parameterization has high distortion near the poles of the parameterization. For the scope of this thesis we do not address this issue, but we provide a discussion of a possible fix to this problem in Section 6.1.

4.2 Dataset Generation

We generate a synthetic dataset for model training using the ray tracer $\mathcal{R}_\Phi$ we described in Section 3.1, as well as a procedural surface weathering texture-map generator $\mathcal{G}_0$ [5]. In order to create a diverse training dataset, we sample a random scene description Φ from a pool of car meshes, base materials, environment maps, and camera parameters. We describe the distribution of assets and their sampling strategies in more detail in Appendix A. Additionally, we mark a subset of the materials as *learnable* and composit the surface weathering texture-map generated from $\mathcal{G}_0$ onto the base material of the surface using the alpha-channel of the texture-map.

During training, each exemplar image from this dataset is loaded along with its corresponding scene description Φ to aide in the training process. We evaluate our model $\mathcal{G}$ by comparing the fidelity of its results to $\mathcal{G}_0$. In Chapter 5, we discuss how the use of the scene description Φ of the training dataset affects the performance of the model.

4.3 Model Hyper-parameters

4.3.1 Network Architecture

We adopt the network architecture choices by Zhou et al.'s model for non-stationary texture synthesis [34]. Our generator network is the ResNet architecture by Johnson et al. [19], featuring several convolutions, followed by six residual blocks [16], and several fractionally strided convolutions. Our discriminator network is the PatchGAN architecture [17].

4.3.2 Loss Function

Empirical results show that training with the Wasserstein GAN loss with gradient penalty as proposed by Gulrajani et al. [15] produce the best results with the fastest convergence rates, compared to the vanilla GAN loss [14] and the LSGAN loss [23]. We present a comparison between these models in Chapter 5.

4.3.3 Training Process

The differentiable ray tracer [21] is a computation and memory intensive operation, even when run on a GPU. Our experiments show that rendering an image with resolution of 256×256, scene complexity of $\sim 1,000,000$ vertices, and Monte-Carlo sub-sampling rate of 200 per pixel, takes ~ 0.3 seconds in the forward direction and ~ 9.0 seconds in the backward direction (i.e. computing the derivative); running multiple render operations in parallel on the GPU further increases this runtime. We found that using a smaller Monte-Carlo sub-sampling rate in the backward direction gives us a more reasonable runtime of ~ 0.9 seconds at the cost of accuracy of the computed derivatives.

Still, coupled with the large memory footprint the operation requires to store the scene description and relevant metadata needed for derivative computations, training the model with a large batch size is unfeasible unless we have a dedicated cluster of GPUs with synchronized training. Due to these limitations, we conducted all experiments using batch size of 1 on a single GPU. [35] shows that training GAN architectures with a batch size of 1 produces good results.

Chapter 5

Experiments

In this section, we present the experiments we conducted for our model along with the corresponding results for each experiment.

5.1 Training Results

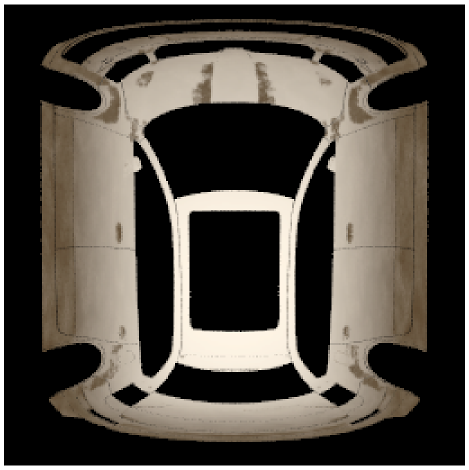
We found that the model fails to learn the target texture accurately when the distribution of the set of scene descriptions used for generating the dataset does not match the set of scene descriptions used at runtime during training. We attribute this failure mode to two primary causes.

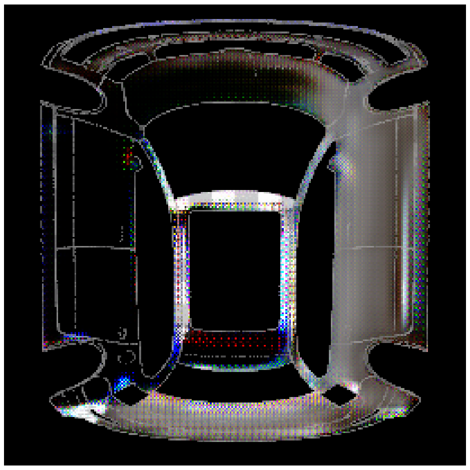
– The discriminator learns to distinguish between the real and fake images using the discrepancy in the statistics of the scene description that is used to render it instead of using the generator output. We refer to this failure mode as discriminator *mode collapse.*

– A batch size of 1 prevents the model from passing useful gradients to the network weights due to lack of feature overlap between real and fake images at each iteration.

In order to address both of these issues, we elect to train the network by specifying the scene description for each image using the scene description of the training

Figure 5-1: We present our results side-by-side with the ground truth albedo as generated by a procedural dirt texture generator $\mathcal{G}_0$. The dataset this model was trained on contains images of cars rendered only from one side with a constant elevation angle. This causes the model to not learn patches that weren't visible in the whole dataset.


(a) Target albedo


(b) Model output

images: at each training iteration of the discriminator, an output from the generator is computed and applied to the same scene as in the training image. The discriminator computes the GAN-loss on this pair of real and fake images. Throughout this training procedure, the only difference the discriminator observes between the pair of fake and real input images is the texture applied to the cars.

This approach allows us to determine a baseline for whether our model is capable of learning the target texture provided no ambiguity on what the discriminator can use to distinguish real images from fake images.

5.2 Ablation Study

As mentioned in the previous section, our model does not train well unless the input image scene description is shared between the real and fake images. We recognize that this constraint prevents us from training our model on anything beyond the synthetic dataset we generated for training purposes. If we hope to generalize our model for real image datasets in the future, determining the model's robustness to

Figure 5-2: We render the albedo-maps from Figure 5-1 onto its corresponding car mesh and compare to the ground truth images.

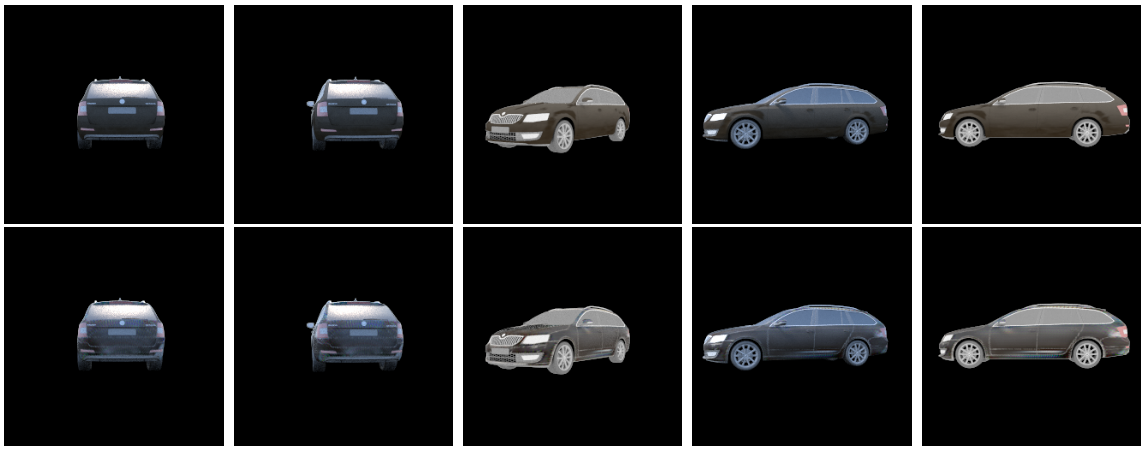
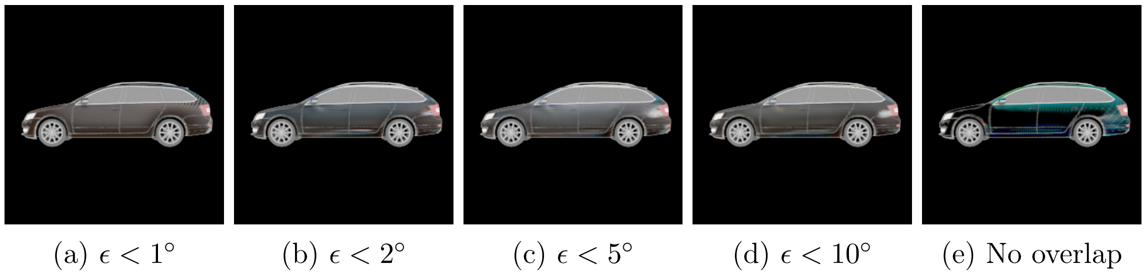


Figure 5-3: Increasing the noise ϵ in the pose estimation during training preserves the quality of the textures learned by the model. However, if the model and dataset pose distributions are non-overlapping, the discriminator experiences *mode collapse*.



non-matching scene-descriptions is crucial. In this section, we present results from training our model after relaxing the "matching scene description" constraint in a variety of controlled situations as a measure of stress-testing our model's robustness.

5.2.1 Camera Pose Distribution

We discussed the technical limitations that require us to train with a batch size of 1 in Section 4.3.3. As a result we are not able to benefit from gradient averaging used in deep learning models which decrease the volatility of the gradient at each iteration. If at each iteration, the discriminator observes vastly different fake and real images, the discriminator's task becomes more complex as it will have to learn to ignore the pose variation of the cars before learning to label images using the texture observed on the car surfaces.

However, having perfect estimation of the camera pose from a real image is not realistic. At best, we can get a noisy estimate of the camera pose. To investigate how much noise in camera pose estimation the model can tolerate and what the effect is on model convergence speed and the quality of the learned textures, we conduct several experiments where we perturb the camera pose loaded from the training image scene by some random uniform noise. The results for this experiments are shown in Figure 5-4. The figure shows that errors of up to 10 degrees in the pose estimation still allow the model to learn a reasonable texture-map. However, the training time for the model increases as the noise increases.

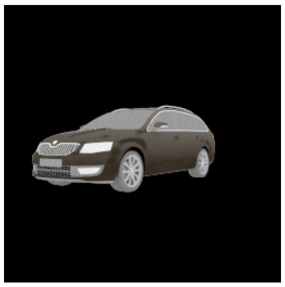
In order to confirm our assumptions stated earlier, we experiment with a setup where the distribution of the camera poses between the real and fake images is completely independent and non-overlapping. If our assumption about the discriminator's *mode collapse* as explained in Section 5.1 is correct, this model should fail to learn a reasonable texture-map. Unsurprisingly, this model does not learn a reasonable texture-map as shown in Figure 5-4, likely because the discriminator learns the difference in camera pose distributions between the real and fake image sets.

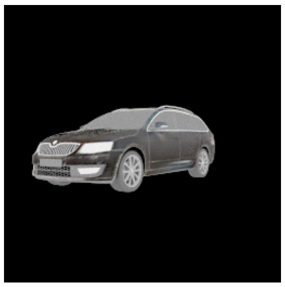
5.2.2 Environment Map Distribution

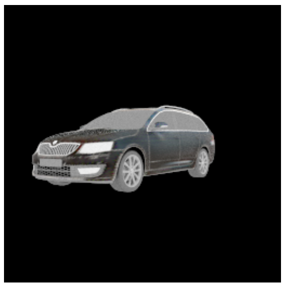
The overall brightness, contrast, and white balance of a rendered image is heavily affected by the lighting parameters of the scene. We observe that the choice of environment map distributions has the highest amount of effect on the network's ability to learn the target texture.

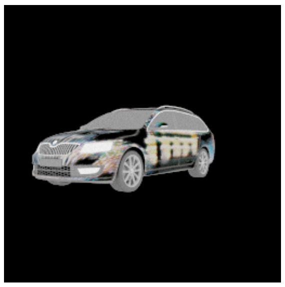
For the environment map ablation study, we experiment with two general setups for which we report our findings.

Figure 5-4: Varying the environment map distribution affects the quality of the outputs of our model. If we train a network with non-matching environment maps sampled from the training distribution, we learn a reasonable texture as shown in Figure 5-4c. Unfortunately, using environment maps sampled from a different distribution leads to *mode collapse*.


(a) Target Image


(b) Same dist, match


(c) Same dist, no match


(d) Different dist, no match

Non-matching samples drawn from the same distribution

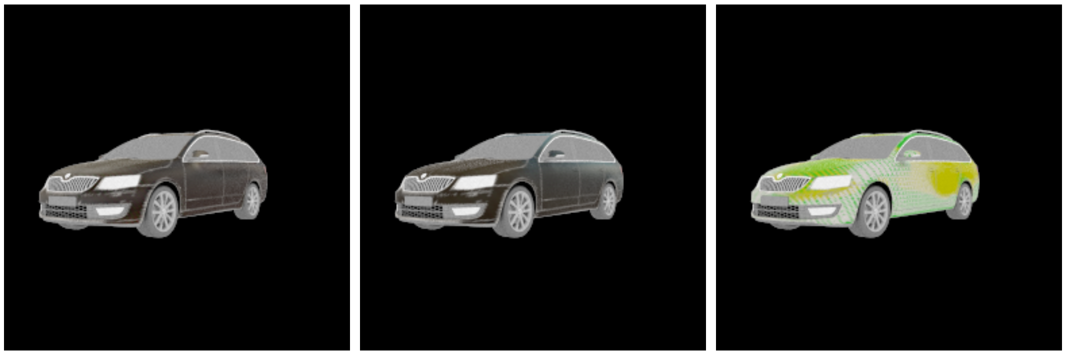
At each iteration, we render the fake image by randomly sampling an environment map configuration that does not match the configuration loaded from the scene description of the real image. However, we sample the configuration from the same distribution as the real image dataset. Doing so allows us to check whether there are any effects other than discriminator *mode collapse*. The results of this experiment show that the model successfully learns the target texture, albeit with a slower convergence rate, which is expected.

Samples drawn from a different distribution

In this setup, we draw the samples of environment map configurations for the fake and real images from a mutually exclusive distribution; each set is rendered using completely different environment maps. Unfortunately, in this experiment the model fails to converge on a realistic surface texture-map generator. This result is likely due to *mode collapse*.

The inability to learn a texture-map due to inaccuracy of the environment map makes it difficult for us to easily extend our model to train on real images. Unless we have a good estimation of the set of environment maps used in the dataset, the

Figure 5-5: Different GAN loss functions produce different results when training our model. From left to right, we show our results for vanilla GAN, WGAN w/ gradient penalty, and LSGAN. The LSGAN model falls into *mode collapse*.



(a) Vanilla GAN (b) WGAN w/ gp (c) LSGAN

model will fail to converge on real images.

5.3 GAN Loss function

Different GAN loss functions have vastly different results when trained for our model. Figure 5-5 shows the comparative results between the convergence speed and resultant textures between vanilla GAN loss, WGAN loss with gradient penalty, and LSGAN loss functions respectively. The WGAN with gradient penalty shows the best results visually, while the LSGAN model fails to learn a reasonable texture-map.

Chapter 6

Discussion and Further Work

This thesis presents promising results for a potential application of differentiable ray tracing [21] in deep learning models. Our results in Chapter 5 demonstrate the extent of our model's generative ability while also outlining the limitations of our technique. In this section, we discuss the source of these limitations and suggest potential improvements to be explored as further work.

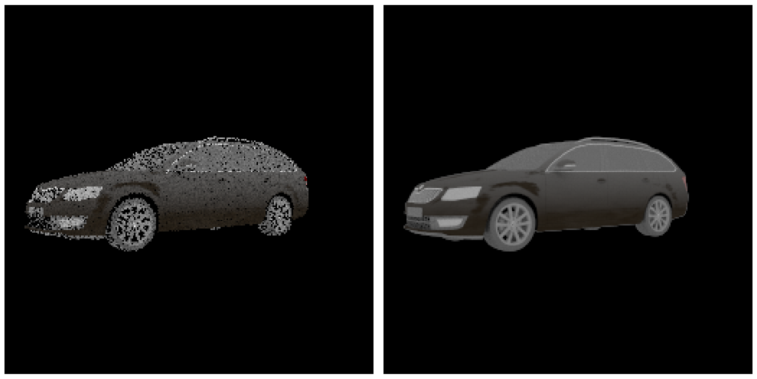
6.1 Distortion in mesh parameterization

In Section 4.1, we justified a simple UV-embedding which utilized the spherical parameterization of the points on the surface while noting the poor distortion qualities of the embedding. For the scope of this thesis, we focused on improving the learning task, and were not able to address the distortion artifacts appropriately. As future work, it is more desirable to use a better UV-embedding parameterization that minimizes surface distortion, such as Sander et al.'s solution of running an optimization to minimize the stretch metric of the embedding [29],

6.2 Generalizing the model for real images

Although the ultimate goal of our model is to learn texture-maps of surface weathering based on exemplar images of real cars, we found directly training GANs on a dataset

Figure 6-1: Monte-Carlo sub-sampling per pixel has a huge effect on the resultant render. Training the model using a low Monte-Carlo sub-sampling rate causes the discriminator to *mode collapse* due to the discrepancy in render quality.


(a) 1 sub-sample (b) 200 sub-samples

of real images to be difficult due to volatility of the model to *mode collapse*. There are many details of real scenes have which are very difficult to emulate in a 3D scene and vice versa.

An example of a fundamental difference between ray traced images and real scene images is Monte-Carlo noise artifacts produced in ray traced images. Monte-Carlo noise is caused by the stochastic nature of Monte-Carlo sampling and leads to noisy images as shown in Figure 6-1. Although this noise decreases with higher MC sampling rates, the SNR improves roughly with $O(n^{1/2})$ where n is the sampling rate. Therefore, computationally, it becomes infeasible to keep increasing the samples we use per pixel. However, unless the dataset has noticeably less Monte-Carlo noise compared to the input dataset, the discriminator learns to distinguish the fake and real images using the noise profile of its input image in a similar fashion as we showed in Chapter 5.

This artifact is only one of many difficulties associated with the problem of extending our model to train on real images. However, our current hypothesis is that the discriminator uses the most easily learnable feature to distinguish between real and fake images. If we keep eliminating the most easily learnable feature that is not the dirt texture on the surface of the car, we hope that the discriminator will eventually be forced to learn to distinguish clean cars from dirty ones.

6.3 Differentiable ray tracing in deep learning applications

This thesis primarily focused on generating simple diffuse textures. But our general method applies to learning arbitrary scene description parameters using deep learning models. Improving the surface shader complexity of our model, such as adding learnable specular components and normal maps, would add more expressiveness to the output textures. Furthermore, we can also attempt to train a model which learns shapes of surfaces in 3D using images of these shapes. Regardless of the specific application in mind, we hope that our work can be a good baseline for deep learning models featuring differentiable ray tracing in future research.

Appendix A

System Overview

We implemented the model using the auto-differentiation library PyTorch [28]. Most of our code was written in Python, and several auxiliary mesh processing scripts use the Blender Python API. Our pipeline features:

– a GAN model architecture featuring the differentiable ray tracer [21]

– a dataset generation pipeline which uses the same ray tracer as our model

– auxilliary input pre-processing scripts.

Our codebase structure is based off of the PyTorch implementation by Zhu et al. [35], and Isola et al. [17]. For features of the codebase such as network implementations, GAN model setup, and training scripts, we refer to their github repository at `https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix`. In this chapter, we describe the modules of the codebase that relate to the differentiable rendering and dataset generation process.

A.1 Render Utilities

The differentiable ray tracer [21] presents an generic interface that is not necessarily purposed for the PyTorch neural network module. We add a wrapper to the ray tracer so that we can add the render operation into a PyTorch neural network as a

`RenderLayer`. As described in Chapter 3, the `RenderLayer` takes as input the scene description Φ and a texture-map to render an image. The scene description Φ is specified by the ray tracer primitives, and the texture-map is a (Height $\times$ Width $\times$ Channel) PyTorch Tensor object. The last dimension of the Tensor encodes the `RGBA` values for each pixel.

Native PyTorch neural network modules are laid out in (Batch $\times$ Channel $\times$ Height $\times$ Width) order. In order to convert between these formats, we implement custom network layers that handle the conversion.

Loading large files containing car triangle-meshes and swapping them out at each iteration slows down the rendering process significantly. Additionally, since we're dealing with over 100 triangle-meshes during training, we cannot load all the car models into GPU memory simultaneously. To mitigate these issues we pre-process all elements in the scene description Φ by loading them into memory and storing them to disk in its pickled format.

A.2 Dataset Generation

When we generate a dataset for training, our goal is to have a high degree of control for the output datasets. In Chapter 5 we presented results from training our model on datasets with different distributions of the scene description, Φ. We implemented a `ConfigSampler` that allows us to specify what kind of distributions to sample Φ from. When generating each image of the new dataset, the `ConfigSampler` samples Φ from the specified distribution and renders an image specified by Φ and stores Φ to disk. During training, we use the ground truth Φ, to generate a similar scene description $\Phi_{training}$. This setup allows us to test the resilience of our model with respect to errors introduced to Φ.

A.3 Auxiliary Scripts

We provide several auxiliary scripts for

- adding a spherical parameterization for meshes
- generating the *geometry buffer* for a spherically parametrized mesh
- generating dirt texture-maps for a car mesh using the Blender API.

Bibliography

[1] Miika Aittala, Timo Aila, and Jaakko Lehtinen. Reflectance modeling by neural texture synthesis. *ACM Transaction on Graphics (Proceeings of SIGGRAPH)*, 35(4):65, 2016.

[2] Miika Aittala, Tim Weyrich, and Jaakko Lehtinen. Practical svbrdf capture in the frequency domain. *ACM Transaction on Graphics (Proceeings of SIGGRAPH)*, 32(4):110:1–110:12, 2013.

[3] Miika Aittala, Tim Weyrich, Jaakko Lehtinen, et al. Two-shot svbrdf capture for stationary materials. *ACM Transaction on Graphics (Proceeings of SIGGRAPH)*, 34(4):110:1–110:13, 2015.

[4] Jonathan T Barron and Jitendra Malik. Shape, illumination, and reflectance from shading. *Transactions on Pattern Analysis and Machine Intelligence*, 37(8):1670–1687, 2015.

[5] Nishchal Bhandari. *Procedural synthetic data for self-driving cars using 3D graphics*. PhD thesis, Massachusetts Institute of Technology, 2018.

[6] Carles Bosch, Pierre-Yves Laffont, Holly Rushmeier, Julie Dorsey, and George Drettakis. Image-guided weathering: A new approach applied to flow phenomena. *ACM Transaction on Graphics*, 30(3):20:1–20:13, 2011.

[7] Yanyun Chen, Lin Xia, Tien-Tsin Wong, Xin Tong, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Visual simulation of weathering by Γ-ton tracing. *ACM Transaction on Graphics (Proceedings of SIGGRAPH*, 24(3):1127–1133, 2005.

[8] Valentin Deschaintre, Miika Aittala, Fredo Durand, George Drettakis, and Adrien Bousseau. Single-image svbrdf capture with a rendering-aware deep network. *ACM Transactions on Graphics (TOG)*, 37(4):128, 2018.

[9] Julie Dorsey, Alan Edelman, Henrik Wann Jensen, Justin Legakis, and Hans Køhling Pedersen. Modeling and rendering of weathered stone. pages 225–234. ACM, 1999.

[10] Julie Dorsey and Pat Hanrahan. Modeling and rendering of metallic patinas. In *SIGGRAPH*, pages 387–396. ACM, 1996.

[11] Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *Advances in neural information processing systems*, pages 262–270, 2015.

[12] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

[13] Kyle Genova, Forrester Cole, Aaron Maschinot, Aaron Sarna, Daniel Vlasic, and William T Freeman. Unsupervised training for 3d morphable model regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8377–8386, 2018.

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[15] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777, 2017.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

[18] Henrik Wann Jensen, Justin Legakis, and Julie Dorsey. Rendering of wet materials. pages 273–282. Eurographics Association, 1999.

[19] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[20] Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *Advances in neural information processing systems*, pages 2539–2547, 2015.

[21] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. In *SIGGRAPH Asia 2018 Technical Papers*, page 222. ACM, 2018.

[22] Shichen Liu, Weikai Chen, Tianye Li, and Hao Li. Soft rasterizer: Differentiable rendering for unsupervised single-view mesh reconstruction. *arXiv preprint arXiv:1901.05567*, 2019.

[23] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2017.

[24] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Shapenet: Convolutional neural networks on non-euclidean manifolds. Technical report, 2015.

[25] F. K. Musgrave, C. E. Kolb, and R. S. Mace. The synthesis and rendering of eroded fractal terrains. *Computer Graphics (Proceedings of SIGGRAPH)*, 23(3):41–50, 1989.

[26] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. Hologan: Unsupervised learning of 3d representations from natural images. *arXiv preprint arXiv:1904.01326*, 2019.

[27] Thu H Nguyen-Phuoc, Chuan Li, Stephen Balaban, and Yongliang Yang. Rendernet: A deep convolutional network for differentiable rendering from 3d shapes. In *Advances in Neural Information Processing Systems*, pages 7891–7901, 2018.

[28] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[29] Pedro V Sander, John Snyder, Steven J Gortler, and Hugues Hoppe. Texture mapping progressive meshes. In *SIGGRAPH*, pages 409–416. ACM, 2001.

[30] Omry Sendik and Daniel Cohen-Or. Deep correlations for texture synthesis. *ACM Transactions on Graphics (TOG)*, 36(5):161, 2017.

[31] Greg Turk. Generating textures on arbitrary surfaces using reaction-diffusion. *Computer Graphics (Proceedings of SIGGRAPH)*, 25(4):289–298, 1991.

[32] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, volume 1, page 4, 2016.

[33] Li-Yi Wei, Sylvain Lefebvre, Vivek Kwatra, and Greg Turk. State of the art in example-based texture synthesis. In *Eurographics State of the Art Report*. Eurographics Association, 2009.

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

[35] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.