UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Mapping Materials to 3D Texture Field using GANs

Author:
Markiian NOVOSAD

Supervisor:
Vladyslav ZAVADSKYI

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2022

Declaration of Authorship

I, Markiian NOVOSAD, declare that this thesis titled, "Mapping Materials to 3D Texture Field using GANs" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"This is the real secret of life – to be completely engaged with what you are doing in the here and now. And instead of calling it work, realize it's a play"

Allan Watts

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Mapping Materials to 3D Texture Field using GANs

by Markiian NOVOSAD

Abstract

3D Texture Synthesis is a broad research field which lets content creators get visually satisfying look of the 3D object with minimal effort. However it is not so easy to achieve satisfying result coupled with computational efficiency. Many of the proposed methods are either computationally expensive and inefficient, or not capable of generating realistic visual appearance. To address this issue, we propose a new method – Mapping 2D Materials to 3D Texture Field using GANs. This method is based on Neural Implicit Representation network, thus able to internally represent a Texture Field without a need for storing additional information. This ability lets our method to be computationally inexpensive, theoretically being able to render the texture in real time. On the other hand, due to Generative Adversarial training strategy, our method is able to achieve highly realistic, visually satisfying looks. In this study we will describe our approach in detail.

Acknowledgements

Firs of all, I would like to express my sincere respect and gratitude to the Armed Forces of Ukraine, who in this trying times stay strong, keeping our homes safe. I want to express my gratitude to my supervisor, Vladyslav Zavadskyi for guiding me through my research and helping me to understand complex ideas better. I want to thank whole ZibraAI team for inspiring me to do this research. Finally, I want to thank my family.

Contents

Declaration of Authorship i

Abstract iii

Acknowledgements iv

1 Introduction 1

2 3D Texture Synthesis and Mapping Problem 3
2.1 Solid Texture Synthesis . . . 3
2.1.1 Procedural Methods . . . 3
2.1.2 Statistical Feature-Matching Methods . . . 3
2.1.3 Patch-based Method . . . 3
2.1.4 Markov Random Field-based Methods . . . 4
2.1.5 Neural Network-based Methods . . . 4
2.2 Implicit Texture Fields . . . 5
2.2.1 Reconstructing Geometry . . . 5
2.2.2 Scene Reconstruction . . . 5
2.2.3 Neural Texture Fields . . . 5

3 Related works 7
3.1 Solid Texture Synthesis . . . 7
3.1.1 GramGAN . . . 7
3.1.2 Solid Texture Synthesis using Generative Adversarial Networks 8
3.2 Periodic Implicit Generative Adversarial Networks . . . 8
3.2.1 Sinusoidal Representation Networks . . . 8
3.2.2 Feature-wise Linear Modulation . . . 9
3.3 Generative Adversarial Networks . . . 9
3.3.1 Wasserstein GAN Training . . . 10
3.4 Unsupervised Representation Learning . . . 10
3.4.1 Contrastive Learning . . . 11

4 Problem Formulation 12

5 Materials and Methods 14
5.1 Dataset . . . 14
5.2 SIREN-Based Texture Field . . . 14
5.3 ResNet-based Encoder network . . . 15
5.3.1 Encoder Architecture . . . 15
5.3.2 Contrastive Training . . . 16
5.4 Discriminator . . . 17
5.5 Objective . . . 17
5.5.1 Generator loss . . . 18
Cosine similarity . . . 18

Histogram similarity 18
5.5.2 Discriminator loss 19

6 Experiments **20**
6.1 Implementation details 20
6.2 Training details 20
6.3 Experiments with data 20
6.4 Experiments with generator 21
6.5 Experiments with discriminator 21

7 Results **23**

8 Conclusions and Future work **25**
8.1 Conclusion 25
8.2 Future work 25

9 Conclusions and future work **26**

Bibliography **27**

List of Figures

3.1 Coordinate Multilayer Perceptron-style architecture introduced in GramGAN [31] 7
3.2 Hierarchical architecture proposed in STS-GAN[37] 8

4.1 Render of the Stanford Bunny colored with texture, generated with our method 12

5.1 Building block of Residual Network[15] 16

6.1 Patches, sized 400 by 400 pixels, of same data sample: anti-aliasing filter applied on the right, no filter on the left 20
6.2 Top row: input data samples, bottom row: synthesized texture. On the left we have the results of experiment with application of anti-aliasing, to the right are the results of experiment without any filtering. One can clearly see, that with unwanted crispy grain removed, generator learns to capture more high-level shapes, when, on the other hand, without anti-aliasing generator eventually starts to synthesize more and more crisp texture 21
6.3 The results of our best model, trained in progressive setting. 22

List of Tables

To the bravest Nation in the World

Chapter 1

Introduction

3D content creation and editing is a cornerstone task in various huge industries, including filmmaking, game development, architecture, and the now-rising Metaverse, to name a few. 3d modeling is a process of developing both geometric and visual digital representations of an object, it can be done both from scratch or by using special software to create the foundation for the model from a real-world object with help of cameras and depth sensors. 3d object is a digital representation of the physical body's geometric and visual properties. (More in-depth explanation will be presented in chapter 2.)

Today, with film and game budgets rising, and mass popularizations of digital metaverses, creators, take on more and more ambitious projects involving creating enormous, sophisticated, immersive, and highly detailed digital worlds that can include millions or maybe billions of unique 3d objects. Each object requires special attention from the artist to be natural, realistic, and attractive, which makes the creation of any project involving computer graphics costly and time-consuming.

3D modeling consists of three main stages: geometry modeling, drawing texture materials, and animating an object. Every stage is complex and time-consuming as they all demand attention to detail. This work will focus on the second stage of 3d modeling – creating the texture. In computer graphics, the texture represents surface characteristics and the visual appearance of an object. Usually, textures are represented in 2d space by UV map images and then mapped onto 3d objects using a unique technique called Texture Mapping. The UV map is the format that represents the 3d object's surface by unwrapping it onto a 2d plane. After the object's surface is mapped to a UV map, the artist starts to draw a texture, using special software tools.

Because of the below-stated complexity, the popularity of automatic content creation instruments rises. In order to ease the process of texturing, a number of software products and research were proposed. The most successful ready-to-use product is ArtEngine which provides various AI-powered tools which tremendously ease the process of working with computer graphics, especially generating textures. Such tools as Unity and Blender offer instruments for procedurally generating textures.

Over the course of recent years, the use of AI to generate, represent, and render 3d content has seen a rise in popularity. Approaches like NeRF[26] can represent the whole scene as one black box using Neural Networks; Neural SDFs[27] and NGLOD[33] offer efficiency in terms of memory by implicitly representing 3d objects using Signed Distance Functions;

In this work we propose a method for generating 3d texture fields from a 2d exemplar patch, using implicit periodic activation functions. We aim to generate realistic textures for any object by creating a texture field, represented by an implicit function.

In chapter 2, we provide the overview of the methodes which were used during our study and neural rendering approaches, related to our research. In Chapter 3,

we define the problem of neural 3d-texture synthesis. In Chapter 4 we describe our approach to solving this problem. in Chapter 5, we describe the experiments conducted during our study, presenting visual examples. Finally, in Chapter 7, we make the conclusion about the quality performance of our approach and discuss future work.

Chapter 2

3D Texture Synthesis and Mapping Problem

The problem of Texture Synthesis is usually posed as producing large texture from given small exemplar. In 3D this problem is usually solved by creating large *Texture Solid Cube* or, given a 3D object, by synthesising or modifying existing UV Maps. In this chapter we will dive more deeply in covering different methods, which we will divide into two groups: Explicit Texture Synthesis methods which synthesize ready-to-use object, and Implicit Texture Fields, the methods to continiously represent a texture.

2.1 Solid Texture Synthesis

2.1.1 Procedural Methods

Procedural methods synthesize textures as a function of pixel coordinates and a set of manual tuning parameters. In Computer Graphics the most successful and widely used procedural method is Perlin Noise [30]. Perlin's *Image Synthesizer*[30] builds up naturalistic visual complexity by composing smooth gradient noise function from several non-linear functions, which is translation and rotation invariant, and band-limited by frequency. By combining enough non-linearities the *Image Synthesizer* can produce visually coherent pseudo-random patterns. The algorithm can be successfully used to synthesize clouds, fire and water waves to name a few, while being fast and memory efficient.

2.1.2 Statistical Feature-Matching Methods

The main idea of this method is to extract statistical appearance features from given exemplar and impose them on synthesized texture.

Heeger at al. [17] makes use of image representation called *Image Pyramid*, which represents and image on different levels of resolutions, starting with high-level small resolution at a top level, and progressively diving to more detail. Heeger's algorithm utilizes the pyramid by matching histogram features of target image with every level of the pyramid, thus preserving texture's appearance both on high and low level of perception. This method is able to produce satisfying results on stochastic cases, but struggles to keep up when the structure is added.

2.1.3 Patch-based Method

The Patch-based Texture Synthesis relies on idea of dividing the exemplar image into small patches and infinitely rearranging them to create a new texture. The

main problem-to-be-solved in this methods are finding the way to statistically match patches that best fit with each other, and finding a method to seamlessly connect them.

Efros and Freeman et al. [9] introduce the image *quilting* method, which utilizes the overlap region between adjacent patches to make shure that they feat each other. The algorithm minimizes the *overlap error* when step-by-step choosing new patches to be inserted into the texture, then *quilting* them together by finding the minimum cost path in overlap region.

Liang and others[25] utilize not only the overlap region information to match patches, but making use of the whole patches to sample them according to a non-parametric estimation of the local conditional *Markov Random Field* density function, which prevents from mismatching patches.

Kwatra et al.[24] improves the methods by introducing new algorithm to optimize the overlap error between patches using a graph-cut algorithm.

2.1.4 Markov Random Field-based Methods

These methods represent a texture as Markov Random Field (later **MRF**), where each pixel, or voxel in 3D applications, in resulted texture depends on it's neighbours. Wei et al.[35] first used this method in texture synthesis by applying nearest neighbourhood matching coupled with image pyramid optimization. Liang et al. [25] also uses this technique in patch-based method.

Kopf et al.[23] used MRF as similarity metric combined with histogram matching algorithm to synthesize a solid, where both high- and low-level statistics match with target's.

MRF-based algorithms show great performance in reproducing stochastic textures, but still lack ability of capturing high-level structure.

2.1.5 Neural Network-based Methods

Recently, number of neural network based methods where introduced to synthesize solid textures. Gutierrez et al. [13] introduced a method to synthesize volumetric textures using *Convolutional Neural Networks* (later **CNN**). The algorithm learns to hierarchically generate solid on different levels to achieve both high- and low- level detail coherency. The proposed method learns to synthesize a texture solid from a set of multi-channel volumetric white noise. The white noise is passed on different scale levels in order to achieve better visual coherency. Authors achieve fairly good results in their experiments, yet still convolutional networks prove to be inefficient, when applied in 3D domain.

Zhao et al. [37] takes further hierarchical framework and trains generator on given texture in multi-scale adversarial-manner. By utilising adversarial training, solid generator achieves great visual reconstruction which resembles visual propertiers of input texture on different scales and look realistic in general. Due to multi-scale discrimination framework the algorithm can successfully reproduce both global structure and fine details. The limitation of this framework is need for retraining for every given exemplar.

In general, CNN-based approaches are limited in terms of memory efficiency and speed performance in 3D applications. Also these approaches need to operate on same space they were trained on, which limits their fidelity.

2.2 Implicit Texture Fields

Recently, the popularity of Neural Implicit representations has emerged. These methods promise continuous and memory efficient representations of content.

In this section we will briefly describe several key works in this field to better show advantages of Neural Representations applied in reconstructing *Geometry*, *Scenes* and *Textures*, and how they they are applied to our problem.

2.2.1 Reconstructing Geometry

Park et al. [27] introduce method called **Deep Signed Distance Functions**. Signed Distance Function (later **SDF**) is a method to represent a 3D object by mapping 3-dimensional coordinate to signed distance to closest surface point of represented object. The advantage of SDF representation is memory efficiency due to continuous domain of the representation. Authors propose using Neural Network as a SDF, by overfitting network's weights on given object.

This approach was taken further by several works. Sitzmann et al.[32] proposed using sinusoidal activation in network and showed that this method performs better in terms of level of detail.

Takikawa et al.[33] solves two problems of existing DeepSDF approach – ability to capture fine detail and computational efficiency by introducing *Neural Geometric Level of Detail* [33]. The main idea of this method is encoding the shape of given object in octree-based feature volume which can adaptively fits shapes with multiple discrete levels of detail, thus using much smaller neural network. In result this method can achieve much finer detail and capable of rendering objects in real time.

2.2.2 Scene Reconstruction

Inspired by success of Deep SDF, Mildenhall and others[26] introduced a novel method for synthesizing novel views of complex static scenes – *Neural Radiance Fields* (later **NeRF**). The method's main concept is representing a scene by optimizing and underlying continuous volumetric scene function using a sparse set of input views. **NeRF** is built on Fully-Connected Multi Layer Perceptron, which maps 5-D input (spatial location (*x*,*y*,*z*) and viewing direction (θ, ϕ)).

This method achieves great level of detail and color when representing even very complex scenes. The main disadvantage of this method is need for optimizing network for each scene individually which takes a long time. There exist numerous methods which derive from original **NeRF** and introduce improvements in different areas.

2.2.3 Neural Texture Fields

Neural Texture Field is a method to represent texture solid on continuous domain using neural network, which given spatial input (x, y, z) and/or 2-D image exemplar produce *rgb* value. The advantage of this type of methods over classic Solid Texture Synthesis algorithms is computational and memory efficiency, and improved level of detail due to absence of resolution limitations of information to be presented.

Henzler et al. [18] introduced generative model of natural textures which takes advantage of noise fields, by learning *Translator model* , which learns to map *latent code* $\mathbf{z}$ to a tuple of noise fields and transformation matrices, which are then fed to

Multi Layer Perceptron which produces the resulting color. This method's advantage is computational efficiency and infinite continuous domain. Due to simple reconstruction training method, this algorithm struggles to capture hierarchical details at different scales and fails at reconstructing high-level structure.

GramGAN framework[31] takes this method further and by extending it with methods from style transfer and generative adversarial training. This method manages to improve previous algorithm, it still cannot solve the problem of hierarchical structure, also both these methods take long time to reproduce even single exemplar.

Chapter 3

Related works

In this chapter we will review the works which are in significant relevance to us, particularly the Periodic Implicit Generative Adversarial Networks (π-GAN) [6], as our work is mostly based on ideas used in this method.

3.1 Solid Texture Synthesis

3.1.1 GramGAN

Portenier et al. [31] presented a novel method for synthesizing infinite, high-quality 3D textures from 2D exemplars by combining ideas both from *Generative Adversarial Networks* and *Style Transfer* frameworks. The method achieves great performance in terms of realism of synthesized texture by utilizing Wasserstein-like Generative-Adversarial training, combined with Gram matrix loss.

The authors upgrade the method, first introduced by Henlzer et al. [18], by injecting noise directly into hidden layers, instead of feeding it through the whole network, which results in each layer specializing on subset of noise frequencies as shown in Figure 3.1

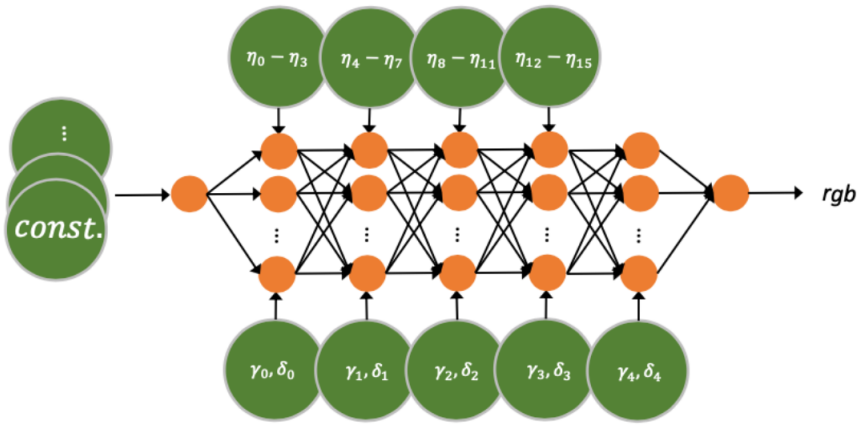

FIGURE 3.1: Coordinate Multilayer Perceptron-style architecture introduced in GramGAN [31]

Although GramGAN method achieves great performance on single exemplars, it lacks ability of capturing high-level structure when trained on set of arbitrary textures. Also, due to use of ReLU activations in hidden layers, architecture struggles to capture fine detail in reasonable training time.

3.1.2 Solid Texture Synthesis using Generative Adversarial Networks

Zhao et al. [37] revisits the idea of explicitly synthesizing Texture Solid and introduces the STS-GAN (Solid Texture Synthesis using Generative Adversarial Networks). Authors propose using Hierarchical architecture (shown in Figure 3.2) to train fully-convolutional generator and discriminator networks on multiple scales of data that significantly improves framework's performance in terms of capturing hierarchical detail on multiple scales.

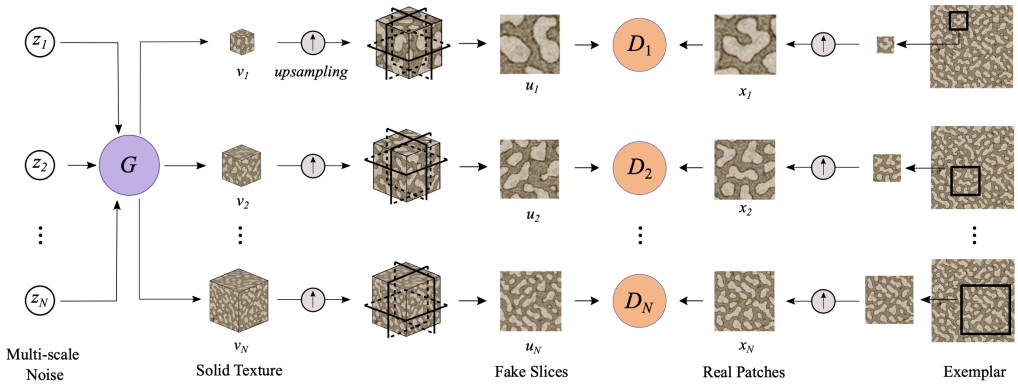

FIGURE 3.2: Hierarchical architecture proposed in STS-GAN[37]

Still, method relies on convolutional network in it's core which show poor efficiency in 3D applications and struggle to reproduce the texture beyond spatial domain they were trained on, which lays limitations in terms of resolution.

3.2 Periodic Implicit Generative Adversarial Networks

Chan and Monteiro et al. [6] take on 3D-aware image synthesis problem by utilising Neural Implicit Representations. PI-GAN achieves state-of-the-art performance in view synthesis and shows superiority of sinusoidal representation networks (SIREN) [32] in 3D view synthesis task.

3.2.1 Sinusoidal Representation Networks

Sitzmann et al. [32] proposes using periodic activations in Coordinate-MLP networks. Authors show in their work that such networks can achieve significantly better performance in terms of detail quality and robustness in fitting complicated natural signals.

Most of previous work in field of neural implicit representation where built on RELU (Rectified Linear Unit) based multilayer perceptrons, this study shows that such type of architectures lack capacity to represent fine details in underlying signals. This can be explained by the fact that RELU is partially a linear function, it's first derivative is zero or one and it's second derivative is always zero. In this study authors propose using sine as an activation function for implicit representation network instead of RELU to improve network's ability to represent complex natural signal. Due to the fact that sine is not a linear function it is able to model higher order derivatives better as the derivative of a sine function is cosine, or phase-shifted sine so the network is able to preserve non-linearity in it's derivatives, which does not hold up to other type of activation functions. This property can be very beneficial in various problems.

Authors show that With specific weight initialization the method can achieve significantly faster convergence rates compared to other method's, while being able to capture much more fine details. For example, in simple task of fitting a 2D image, the network is able to converge in couple hundreds of steps, taking few seconds on modern GPU.

The SIREN method is able not only to learn one specific signal but is capable of parametrizing the whole space of implicit function which will be more described in next section.This property is in great interest to us, as we try to learn not only one particular texture field, but to represent the whole space of different textures.

3.2.2 Feature-wise Linear Modulation

Feature-wise Linear Modulation, or **FiLM** is a general-purpose conditioning method for neural network, introduced by Perez et al. [29]. FiLM layers influence neural network computation via simple yet powerful feature-wise affine transformation based on conditioning information.

In general, FiLM learns to adaptively influence a neural network by applying feature-wise linear modulation to the network's intermediate features, based on some conditioning input. FiLM learns functions f and h which output $\gamma_{i,c}$ and $\beta_{i,c}$ as a function of input $\mathbf{x}_i$:

$$\gamma_{i,c} = f_c(\mathbf{x}_i) \qquad \beta_{i,c} = h_c(\mathbf{x}_i)$$

where $\gamma_{i,c}$ and $\beta_{i,c}$ modulate a neural network's activations $\mathbf{F}_{i,c}$, where i indicates the i^{th} layer and c input's c^{th} feature:

$$FiLM(\mathbf{F}_{i,c}|\gamma_{i,c}, \beta_{i,c}) = \gamma_{i,c}\mathbf{F}_{i,c} + \beta_{i,c}$$

FiLM's method of conditioning requires only two parameters per feature, which makes it computationaly efficient and scalable. The f and h can be treated as one function, as it can be beneficial to more efficient learning.

By various test, authors show great performance of method in such task as visual reasoning by conditioning Convolutional Network with FiLM. In pi-GAN framework [6] authors use this method to condition SIREN network which yields outstanding results in terms of visual appearance. Due to periodic properties of sinusoidal implicit representation networks explained above, linear modulation furthermore empowers the network in representing the whole space of different signals.

3.3 Generative Adversarial Networks

Goodfellow et al.[11] proposed a new fundamental framework called *Generative Adversarial Networks* (later GANs) for training generative models. The idea of the framework is simple yet very powerful: authors propose to train model via adversarial process, in which two models are simultaneosly trained – a generative model G, or *Generator*, that captures data distribution, and a discriminative or critic model D which has a goal of estimating that a sample came from real data distribution rather than from *Generator*. In this process, the goal of the *Generator* is to maximize a probability of *Discriminator* making a mistake.

By training a generative model in such a manner, one is able to get a model that can reproduce a real data distribution in unique manner, with no need of labeled

dataset. This framework becomes really handful in tasks when real ground truth for a specific point in data distribution can not be obtained.

After introduction of this method, in coming years it found unique applications and upgrades which led to tremendous results in such domains as image-to-image generation, style-transfer, text-to-image generation, image editing to name a few.

Yet, this method has a number of problems, one of which is training instabilities in a number of cases, which lead to failure of a training process or poor training results. Another problem of an original method, when dealing with complex data distributions, generative network fails to capture desired distribution, instead being misled by a discriminator network.

In following subsection we will briefly describe methods which intend to solving this problem.

3.3.1 Wasserstein GAN Training

Arjovsky et al. [2] tackles a problem of instabilities of an original GAN method [11]. The authors question the fundamentals of the framework and aim to solve the main problem – instability during training.

In this study authors show the fundamental property of GAN framework: in ideal case scenario both *generator* and *discriminator* network keep up with one another and *discriminator* produces gradients that are able to guide *generator* to gradually improve it's results. But in practice we often get to the case, whem *discriminator* outperforms it's rival thus not providing any usefull information to the *generator*, which leads to diminishing of the gradients. To fix this problem, authors propose constraining the *discriminator* network with *1-Lipshitz* constraint. The aim of *K-Lipshitz* constraint is to control how fast some function is growing by bounding function's gradients. Authors argue that by maintaining *1-Lipshitz* constraint drastically improves balance between *generator* and *discriminator* networks which results in more stable training and better results. Authors mathematically prove that such *K-Lipshitz continuity* can be achieved by optimizing Wassertein distance between real and generated data distributions. Also, to enforce the *1-Lipshitz* constraint, authors propose clipping weights of the *discriminator* to lie within a compact space $[-c : c]$.

Yet still, the original method proposed in [2] does not solve the problem completely, in fact still causing training instabilities in some cases. Gulrajani et al. [12] takes this method further, by proposing *gradient penalty* and demonstrate performance improvements, compared to the original method.

Authors at [12] propose directly constraining the gradient norm of the *discriminator's* output with respect to it's input. Authors prove, that the optimal *discriminator* contains straight lines with gradient norm 1 connecting coupled points from *real* and *fake* distributions [12], thus they propose sampling datapoints on those straight lines and constraining the gradient norm to 1 for the distribution of these datapoints.

3.4 Unsupervised Representation Learning

Fundamentaly *Machine Learning* task can be divided into two categories: *Supervised* and *Unsupervised* learning. In *Supervised* learning trained model is "supervised" by the training algorithm with labeled ground truth data. In *Unsupervised learning* tasks, the model is objected to build it's own internal patterns and learn to complete the task without being "supervised", thus not using labeled data.

During the recent years we've seen volumes of datasets used in for machine learning tasks grow to huge numbers, thus making it more and more complicated to produce labeles to such big datasets. Due to this fact unsupervised methods received much more research interest in past years as they can achieve great results without any labeled data needed.

Such methods are in particular interest to us as we use the unlabeled dataset for our research.

Self-supervised representation learning method can be defined as machine learning method that aims to provide deep feature representations of given data distribution. The main idea of such methods is to maximize representation agreement between different views of the same image. These methods can be divided into two approaches: *contrastive* methods and *informatin maximization* methods. In this work we rely on *contrastive* method as it gives us the result of satisfying quality.

3.4.1 Contrastive Learning

The main idea of contrastive self-supervised learning methods is to bring the representations of different views of the same image as close as possible while pushing views of different images as far apart as possible. The method is often used with Siamese network architectures [4], in which two identical networks with shared weights and are trained to produce representations [38] [5] [7] [19] [16]. Such methods when applied to large enough datasets yield great results, with the only downfall being enormous computational and memory costs, as these methods require large batch sizes in order to learn accurate representations.

Chapter 4

Problem Formulation

FIGURE 4.1: Render of the Stanford Bunny colored with texture, generated with our method

As stated in the introduction before, the task of creating a texture for an object can be very time-consuming. One of the both most relevant and challenging objectives is creating a realistic texture. Even when synthesizing texture with a pattern, like marble, procedural approaches struggle to achieve a realistic look, the texture can look natural on a 2d plane, but can lose its realism when wrapped around a 3d object.

Recently, deep learning is widely used to accomplish a similar task – 2d image generation. Generative adversarial networks [11] show high performance in terms of realism and robustness of a model. In pair with deep learning approaches, classic 2d texture-synthesis algorithms [30] [9] [17] achieve good performance on 2d planes but look odd, when wrapped around 3d objects.

On the other hand, Neural Rendering approaches offer highly detailed scene representations, such as NeRF [26], which can represent both geometry and color information of an object via implicit radiance function, though they need to be overfitted on desired scene or object which takes a long time, which means long creation time for every single scene.

In this work, we aim to create a framework, which given a single 2d exemplar, can generate an infinite amount of highly realistic texture for an object, sampled

directly in 3d space, with no need of using UV maps. In this study, we will use Pi-GAN (which relies on FiLM-SIREN as sampling network) as a basis for our research. We will try to modify and improve this work, to accomplish the task of implicitly representing texture field defined in 3d space. For extracting the representation from exemplars, we will experiment with existing unsupervised learning approaches, as we do not have annotated dataset nor time to create our own. We will experiment with using different losses to improve the visual quality of synthesized textures and different Generative Adversarial aprroaches to training our network, in order to achieve best performance

Summarazing, the main contribution of this work is, as far as we know, the first Implicit Representation-based method for efficient synthesis of realistic 3D texture fields in one-shot manner (Figure **??**).

Chapter 5

Materials and Methods

In this chapter we will present our proposed method and describe data, used for training.

Our method is a generative approach to learning texture field representations from unlabeled 2D images. The goal is to synthesize high-quality visually satisfying 3D textures given 2D image as an input. Our approach is based on the following components: convolutional residual Encoder network, SIREN-based coordinate multilayer perceptron decoder network and progressive-growing Discriminator network. We try different generative adversarial strategies, experiment with various loss functions and generator network architecture variations in order to improve visual performance of our framework.

In this chapter we will go through every essential component of our framework, describe training and pre-training methods, present details of neural network architecture and reason about loss functions used in this method. Also in this chapter we will present the material dataset used for traininng of our framework.

5.1 Dataset

For this work we used a manually scrapped dataset consisting of over a thousand of 8192×8192 large scale material renders (Figure **??**). During this study, the experiments were conducted on a subset of data consisting from marble materials due to more simplistic nature of marble, compared to other materials.

Firstly, training data was preprocessed as follows: data samples were resized to the dimension of 1200×1200 using anti-aliasing resize, during the experiments we observed that not using an anti-aliasing filter misled our model to producing noisy and grainy views.

5.2 SIREN-Based Texture Field

We proposed representing 3D Texture Solid as an infinite Implicit Texture Field. As a representation backbone of our framework we proposed using FiLM conditioned SIREN, due to great capabilities of this method shown by authors in pi-GAN [6]. Instead of using classic Radiance Field [26], we use simpler network, which maps spatial coordinate to it's color.

Our 3D Texture Solids are represented implicitly with neural texture field, which is parameterized as multilayer perceptron (MLP) that maps a spatial coordinate $\mathbf{x} = (x, y, z)$ to view-independent color value $(r, g, b) = \mathbf{c}(\mathbf{x}) : \mathbb{R}^3 \rightarrow \mathbb{R}^3$. We use StyleGAN-based mapping network, to condition our implicit representation network on latent vector $\mathbf{z}$ through **FiLM** conditioning [29] as proposed by authors in [6].

Our backbone representation network can be formalized as follows:

$$\Phi(\mathbf{x}) = \phi_{n-1} \circ \phi_{n-2} \circ \cdots \circ \phi_0(\mathbf{x}), \tag{5.2.1}$$

Where ϕ is defined as:

$$\phi_i(\mathbf{x}_i) = sin(\gamma_i \cdot (\mathbf{W_i x_i} + \mathbf{b_i}) + \beta_i), \tag{5.2.2}$$

where $\phi_i : \mathbb{R}^{M_i} \mapsto \mathbb{R}^{N_i}$ is the i^{th} layer of an Multilayer Perceptron. It consists of affine transform defined by weight matrix $\mathbf{W}_i \in \mathbb{R}^{N_i \times M_i}$ and biases $\mathbf{b}_i \in \mathbb{R}^{N_i}$ applied to input $\mathbf{x_i} \in R^{M_i}$, then transformed by FiLM's frequency $\gamma_i \in \mathbb{R}^{N_i}$ and phase shift $\beta_i \in \mathbb{R}^{N_i}$ followed by sine nonlinearity activation.

We use simple four-layer fully-connected ReLU MLP as mapping network, which maps n-dimensional latent vector $\mathbf{z}$ to conditioning frequencies γ_i and phace shifts β_i: $(\boldsymbol{\gamma}, \boldsymbol{\beta}) = \mathcal{M}(\mathbf{z}) : \mathbb{R}^L \to \mathbb{R}^{2 \times N \times D}$, where L is latent vector dimension, N is number of layers in conditioned network and D is number of features in each layers, following pi-GAN[6].

5.3 ResNet-based Encoder network

As stated above, our generative neural network's output is conditioned on latent vector $\mathbf{z}$. We added the encoder network which can map given input image $\mathbf{I}$ to latent representation representation space which carries insightful information mainly to optimize network training and to receive visually correct results. In this section, we will describe our Encoder network's architecture and training methods in order to achieve our goals.

5.3.1 Encoder Architecture

Our Encoder network can be defined as function $\mathbf{E}(I) = \mathbf{z}_I$, with $\mathbf{E} : \mathbb{R}^{H \times W \times 3} \to \mathbb{R}^L$, where H and W are height and width dimensions of input image respectively and L is dimension of latent space. As an underlying architecture of our encoder we decided to use Deep Residual Network, proposed by Kaiming et al. [15]. Authors show that such network architecture is easier to optimize and gives better results compared to other architectures while having lower complexity.

While the network can be formalized as a mapping function $y = \mathcal{H}(x)$, authors propose learn instead a residual mapping of a form $\mathcal{F}(x) := \mathcal{H}(x) - x$ (Figure 5.1). Authors argue that this type of mapping is easier to optimize [15].

We propose to modify underlying architecture with substituting downsampling layers with anti-aliasing *BlurPool* layers introduced by Zhang et al.[36] to further push accuracy of the model's representations. Zhang[36] and Karras et al. [20] argue, that applying anti-aliasing filters when down- or upscaling is crucial for accuracy of internal representation of natural signal processed by network. Authors show that standard hierarchical convolutional network tend to depend on absolute pixel locations in an unhealthy manner, thus propose a method to avoid that by not letting unwanted information to leak into internal representations of network.

In studies [36] [20] propose using *BlurPool* layers instead of standard Strided Convolution, Max Pooling or Average Pooling layers. The $\mathbf{BlurPool}_{m,s}$ operation consists of anti-aliasing filter with kernel $m \times m$, denoted as $Blur_m$ and subsampling

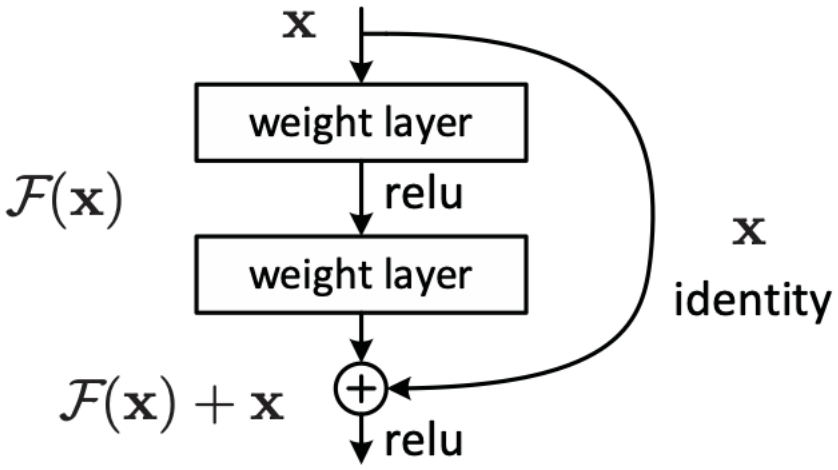

FIGURE 5.1: Building block of Residual Network[15]

operation with stride s denoted as $Subsample_s$:

$$\textbf{BlurPool}_{m,s} = Subsample_s \circ Blur_m \tag{5.3.1}$$

Therefore standard strided convolution operation with $k \times k$ kernel and s stride denoted as $StridedConv_{k,s}$ now can be modified as follows:

$$\textbf{ReLU} \circ \textbf{StridedConv}_{k,s} \longrightarrow \textbf{BlurPool}_{m,s} \circ \textbf{ReLU} \circ \textbf{Conv}_{k,1} \tag{5.3.2}$$

We propose using 50-layer Deep Residual Convolutional Neural Network [15], with blurred downsampling layers with additional linear fully-connected layer, to reduce dimension produced by the network to *512*. We find this architecture to optimal in terms of size and performance.

5.3.2 Contrastive Training

Learning Encoder to produce coherent representation when trained in generative adversarial network can be a tricky task. Due this fact, we decided to pretrain our Encoder network prior to training our whole framework and found it benefiting models in faster convergence and network's resulting visual coherency.

Due to the fact that our data is unlabeled, pretraining of the encoder network becomes a classic *Unsupervised Representation Learning* task. Among methods that solve this task we chose to use simple yet effective **SimCLR** framework, introduced by Chen et al. [7].

The main idea of **SimCLR** framework is to learn representation by maximizing representation agreement between two random augmentations of the same data example via contrastive loss in the latent space. Authors propose four essential components of the **SimCLR** framework[7]:

- Random data augmentation module, which given a single data sample as an input, randomly transforms it into two correlated views of the input, thus producing a positive pair $\tilde{\mathbf{x}}_i$ and $\tilde{\mathbf{x}}_j$.
- Base Encoder network which produces latent vectors, particularly the network we aim to train.

- Small Projection Head network. The main purpose of this small network is to project vector representations onto another space instead of applying contrastive loss directly on representation vectors. Authors argue [7], that applying the loss function in projected space can drastically improve both training and resulting representations.
- Contrastive Loss function which is applied on vector projections. Given a set of pairs of samples, the function aims to identify positive ones among negative.

Authors [7] define the contrastive loss function for a positive pair of data samples (i, j) as follows:

$$\mathrm{l}_{i,j} = -log\frac{exp(sim(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k\neq i]} exp(sim(\mathbf{z}_i, \mathbf{z}_k)/\tau)} \tag{5.3.3}$$

where N is a number of data samples in a batch, $\mathbb{1}_{[k\neq i]} \in 0, 1$ is an indicator function evaluating to 1 if $k \neq i$ and τ denotes temperature parameter.

5.4 Discriminator

In this method, we treat given exemplar 2D image of texture as a slice of real generated texture solid, thus when training *Discriminator* it's objective is to distinguish 2D slices from texture solid space, represented by our network and real slices – randomly cropped parts from our input texture.

We decided to adopt discriminator architecture from the method introduced by Zhao et al.[37], due to similarity of our problem to classic solid texture synthesis. This architecture can drastically improve generator's resulting performance in terms of capturing both high- and low-level detail.

Having a *generator* which synthesizes textures at arbitrary absolute scale and resolution we need to make sure that at different levels of scale it maintains visual appearance of given input exemplar texture. Having this goal in mind, authors argue that training *generator* with one *discriminator* on one specific scale can lead *generator* to failing to capture detail on other levels of scale[37]. Thus authors propose training N separate *discriminators* each on it's own specific scale, therefore guiding *generator* to learn both global and more local appearance of textures. As our *generator* is, in general, a coordinate function, we do not need to do any workarounds or modifications for such training method. For each absolute scale S_n we change the coordinates, which we use to sample information from the *generator*

Therefore, to formalize our adversarial training strategy: having N manually specified scales S_n, for each such scale we initialize discriminator network D_{S_n}, each to be trained on specified scale of view.

As an underlying architecture, we choose discriminator D_{S_n} to be a simple convolutional feature-extractor followed by linear fully-connected classifying layer.

5.5 Objective

In this section, we will describe the parts of our objective, which we use to improve performance of our framework.

5.5.1 Generator loss

As described in previous section, we decide to use multiple discriminators, so our *discriminator* loss function part will look different from standard: having N discriminator networks, we take the average score from every network.

The discriminator part of generator's objective can be denoted as

$$\mathcal{L}_G = -\frac{1}{N}\sum_{n=1}^{N} \mathrm{D}_{S_n}(\mathrm{G}(\mathbf{x}_{S_n})) \tag{5.5.1}$$

where D_{S_n} is a *discriminator* network operating on n^{th} scale S_n and N is a number of such scales.

Using only the discriminator part proved to be not enough in order to achieve satisfying visual result. To fix this and guide our generator network in right direction we add to additional parts: **cosine similarity** and **histogram similarity** losses.

Cosine similarity

Having already pretrained *encoder* network, which gives accurate latent representations of images, lets us use it to estimate how similar the generated view is to the data sample given as an input. We estimate the similarity of generated and real data samples by applying *cosine similarity* function to latent vector projections of real and fake data using our pretrained *encoder* and *projection head* networks. As argued by authors of **SimCLR** framework [7], applying similarity in projected space yields better results.

Thus, our *cosine similarity* loss part can be denoted as:

$$\mathcal{L}_{cos} = \frac{x_{\text{real}} \cdot x_{\text{fake}}}{\max(\|x_{\text{real}}\|_2 \cdot \|x_{\text{fake}}\|_2, \epsilon)} \tag{5.5.2}$$

where x_{real} and x_{fake} denote real and fake representation projections respectievly and ϵ is small constant for numerical stability.

Adding this loss in our case result in huge benefit in form of generator reaching satisfying visual faster and overall training stability.

Histogram similarity

In order to push our generator's performance even further, we decided to employ loss, proposed by Afifi et al. [1] – *histogram similarity loss*. As proposed in the study, for each color channel of an image we compute a histogram projected into log-chroma space as it is equally insightfull but more compact then standard 3D histogram defined in RGB space.

Log-chroma space is defined by intensity of one channel, normalized by the other two. Instead of selecting one particular color, we use all three variations as a $h \times h \times 3$ tensor, following the original study [1].

To obtain a histogram feature first we need to convert image $\mathbf{I}$ into log-chroma space. This can be done as follows:

$$\mathbf{I}_{uR}(\mathrm{x}) = \log(\frac{\mathbf{I}_R(\mathrm{x}) + \epsilon}{\mathbf{I}_G(\mathrm{x}) + \epsilon}), \quad \mathbf{I}_{vR}(\mathrm{x}) = \log(\frac{\mathbf{I}_R(\mathrm{x}) + \epsilon}{\mathbf{I}_B(\mathrm{x}) + \epsilon}) \tag{5.5.3}$$

where R, G, B subscripts refer to color channels, x is a pixel index, ϵ is a small numerical stability constant and (uR, vR) are uv coordinates in resulting image.

The resulting histogram $\mathbf{H}_c$ can be computed as follows:

$$\mathbf{H}(u, v, c) \propto \sum_x k(\mathbf{I}_{uc}(\mathrm{x}), \mathbf{I}_{uc}(\mathrm{x}), u, b)\mathbf{I}_y(\mathrm{x}), \tag{5.5.4}$$

where $c \in R, G, B$, and k is the inverse-quadratic kernel and $\mathbf{I}_y(\mathrm{x})$ is contribution of each pixel by intensity, denoted as:

$$\mathbf{I}_y(\mathrm{x}) = \sqrt{\mathbf{I}_R^2(\mathrm{x}), \mathbf{I}_G^2(\mathrm{x}), \mathbf{I}_B^2(\mathrm{x})} \tag{5.5.5}$$

Further the histogram feature is normalized to sum to one.

Finally, having histogram features of *real* and *fake* data samples $\mathbf{H}_{\text{real}}$ and $\mathbf{H}_{\text{fake}}$ respectively, we can estimate the similarity by computing Hellinger distance as proposed by authors [1], defined as:

$$\mathcal{L}_{hist} = C(\mathbf{H}_{\text{fake}}, \mathbf{H}_{\text{real}}) = \frac{1}{\sqrt{2}} \|\mathbf{H}_{\text{fake}}^{1/2} - \mathbf{H}_{\text{real}}^{1/2}\|_2, \tag{5.5.6}$$

where $\mathbf{H}^{1/2}$ is an element-vise square root of the histogram. This part of a final loss helps to guide generator to preserve the color scheme of original input sample.

Having explained all the essential parts we can denote our generator loss function as follows:

$$\mathcal{L}_G = -\frac{1}{N} \sum_{n=1}^{N} \mathrm{D}_{S_n}(\mathrm{G}(\mathbf{x}_{S_n})) + \lambda_{hist} \cdot \mathcal{L}_{hist} - \lambda_{cos} \cdot \mathcal{L}_{cos}, \tag{5.5.7}$$

where λ_{cos} and λ_{hist} are hyperparameters.

5.5.2 Discriminator loss

As stated above, we use multiple discriminator's D_{S_n} operating each on it's own specific scale. Our main component of discriminator's loss function is defined as:

$$\mathcal{L}_D = \frac{1}{N} \sum_{n=1}^{N} \mathrm{D}_{S_n}(\mathrm{G}(\mathbf{x}_{S_n})) - \frac{1}{N} \sum_{n=1}^{N} \mathrm{D}_{S_n}(\mathrm{I}_{\text{real}}) \tag{5.5.8}$$

We propose use Wasserstein-GAN gradient penalty, proposed by Gulrajani et al. [12], to improve training stability. The gradient penalty can be described as:

$$\mathcal{L}_{GP} = \frac{1}{N} \sum_{n=1}^{N} (\|\nabla_u \mathrm{D}_{S_n}(u)\|_2 - 1)^2 \tag{5.5.9}$$

Where u denotes an interpolation between real and fake images.

In this chapter we described details of our data and method. In next chapter we will show the results achieved with *Mapping Materials to 3D Texture Field using proposed GANs*.

Chapter 6

Experiments

In this chapter we will give an overview to training details and experiments, that were performed during this study.

6.1 Implementation details

The work, presented in this thesis is implemented using Python 3.9 [34] programming languages, which one of the best suits for deep learning tasks. Our implementation is based on the pi-GAN[6] and SimCLR[7] implementations, both written in PyTorch [28] framework. The whole training pipeline was written using PyTorch-Lightning [10] framework. During training, visualization of the process was done using Tensorboard. For image processing and additional visualization we use NumPy[14] and OpenCV[3].

6.2 Training details

All the experiments were conducted on two NVIDIA RTX 3080TI GPU's. We use Adam optimizer [22] with learning rate set to $1e^{-5}$ and (β_1, β_2) parameters set to $(0, 0.9)$. The encoder was pretrained on the whole material dataset with batch size 128, using *SGD* optimization technique.

6.3 Experiments with data

Initially, experiments where performed on data, resized without an anti-aliasing filter. We discovered that it was causing unwanted crisp grain. The difference can be seen on side-to-side comparison on Figure 6.1

FIGURE 6.1: Patches, sized 400 by 400 pixels, of same data sample: anti-aliasing filter applied on the right, no filter on the left

6.4 Experiments with generator

During the study many experiments where conducted on the architecture of encoder network. We tested 18-, 34-, 50- and 101-layer settings of ResNet architecture [15]. The 50-layer setting was chosen due to producing better representations compared to smaller models and 101-layer proved to be to large for this kind of task. Additionally we conducted experiments with "blurring" encoder's downsampling layers, as described in Section 5.3. During ablation study of "blurring" encoder, we saw that this technique helps the encoder recognize more high-level structure, therefore give better representation.

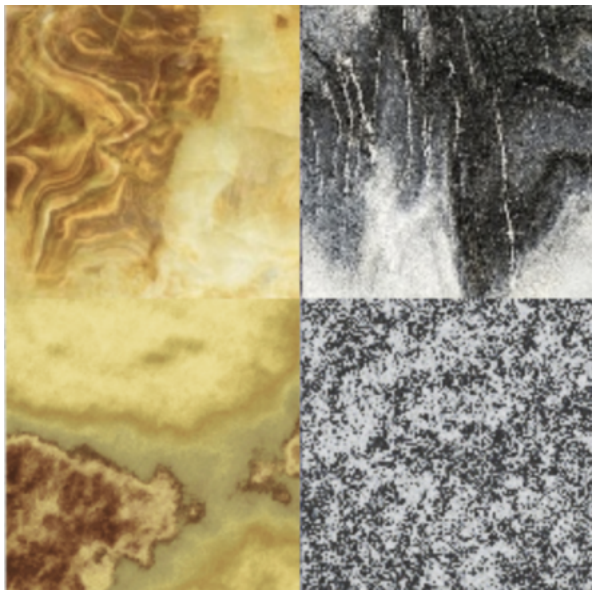

FIGURE 6.2: Top row: input data samples, bottom row: synthesized texture. On the left we have the results of experiment with application of anti-aliasing, to the right are the results of experiment without any filtering. One can clearly see, that with unwanted crispy grain removed, generator learns to capture more high-level shapes, when, on the other hand, without anti-aliasing generator eventually starts to synthesize more and more crisp texture

Following the recent study by Chng et al. [8], we conducted several experiments in which we implemented our version of Gaussian Activated MLP. We did a simple experiment of fitting the network to represent single image, and as a result SIREN proved to perform better in terms of convergence speed and image quality.

6.5 Experiments with discriminator

When conducting first experiments, we discovered that in every setting, generator network couldn't keep up with discriminator network, there discriminator quickly learned to outperform the generator and didn't give any useful information to generator. To tackle this problem, we experimented with number of different GAN training strategies: first, we added the WGAN-GP [12] gradient penalty in order to improve stability, secondly we tried out ProgressiveGAN [21], thirdly we used the hierarchical strategy described in 5.4. The ProgressiveGAN's main idea is to gradually increase resolution of generated images and gradually training specific blocks

of discriminator network. We conducted numerous experiments, trying out different settings and hyperparameters for this framework but were not able to achieve satisfying results. We hypothesise that through first stages of progressive training generator is not able to learn sufficient enough high-level structure representations, therefore struggling later and being outperformed as a result (Figure 6.3).

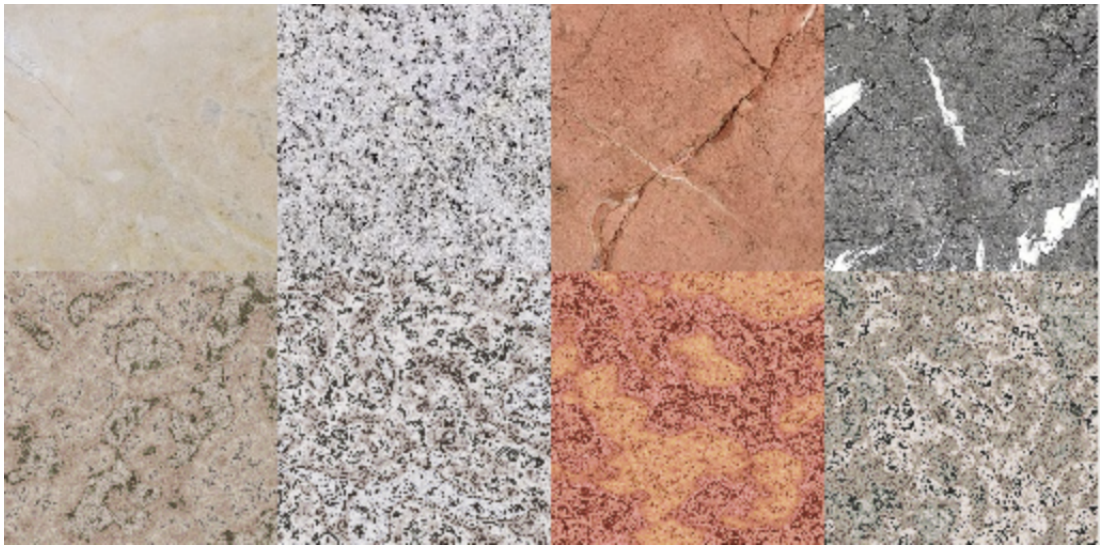

FIGURE 6.3: The results of our best model, trained in progressive setting.

We still continue conducting experiments with hierarchical setting as we didn't achieve best possible results, yet. The results of current best network will be shown in next chapter.

Chapter 7

Results

As this research is still in ongoing active state, thus results are not final. During this research, so far, we managed to achieve visually tolerable results on their own but still our framework is not yet able to accurately reconstruct given 2D input.

Here we will show some of the renders of geometric objects, textured with our texture field.

Chapter 8

Conclusions and Future work

8.1 Conclusion

In this work, we introduce a new method to implicitly represent and generate 3D Texture Fields in zero-shot manner using sinusoidal representation networks.

So far, we focused on finding best adversarial strategy to train our generator network and there still is work to be done.

We showed that sinusoidal representation networks [32] are capable of learning prior for huge space of different images and able to synthesize satisfying visual representations when conditioned on particular example.

8.2 Future work

We plan to continue this research to push the reconstruction abilities of the framework even further. We want to explore, how the framework will behave on other sets of date, not as simple as marble, for instance. We see two main directions for future improvements: upgrade of adversarial training strategy and conditioning of sampling network on 3D geometry data.

Chapter 9

Conclusions and future work

Bibliography

[1] Mahmoud Afifi, Marcus A. Brubaker, and Michael S. Brown. *HistoGAN: Controlling Colors of GAN-Generated and Real Images via Color Histograms*. 2020. eprint: arXiv:2011.11731.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. DOI: 10.48550/ARXIV.1701.07875. URL: https://arxiv.org/abs/1701.07875.

[3] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[4] Jane Bromley et al. "Signature Verification using a "Siamese" Time Delay Neural Network". In: *Advances in Neural Information Processing Systems*. Ed. by J. Cowan, G. Tesauro, and J. Alspector. Vol. 6. Morgan-Kaufmann, 1993. URL: https://proceedings.neurips.cc/paper/1993/file/288cc0ff022877bd3df94bc9360b9c5d-Paper.pdf.

[5] Jane Bromley et al. "Signature Verification using a "Siamese" Time Delay Neural Network". In: *Advances in Neural Information Processing Systems*. Ed. by J. Cowan, G. Tesauro, and J. Alspector. Vol. 6. Morgan-Kaufmann, 1993. URL: https://proceedings.neurips.cc/paper/1993/file/288cc0ff022877bd3df94bc9360b9c5d-Paper.pdf.

[6] Eric R. Chan et al. *pi-GAN: Periodic Implicit Generative Adversarial Networks for 3D-Aware Image Synthesis*. 2020. arXiv: 2012.00926.

[7] Ting Chen et al. *A Simple Framework for Contrastive Learning of Visual Representations*. 2020. eprint: arXiv:2002.05709.

[8] Shin-Fang Chng et al. *GARF: Gaussian Activated Radiance Fields for High Fidelity Reconstruction and Pose Estimation*. 2022. arXiv: 2204.05735 [cs.CV].

[9] Alexei A. Efros and William T. Freeman. "Image Quilting for Texture Synthesis and Transfer". In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. New York, NY, USA: Association for Computing Machinery, 2001, 341–346. ISBN: 158113374X. DOI: 10.1145/383259.383296. URL: https://doi.org/10.1145/383259.383296.

[10] William Falcon et al. "PyTorch Lightning". In: *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning* 3 (2019).

[11] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. eprint: arXiv:1406.2661.

[12] Ishaan Gulrajani et al. *Improved Training of Wasserstein GANs*. 2017. eprint: arXiv:1704.00028.

[13] J. Gutierrez et al. "On Demand Solid Texture Synthesis Using Deep 3D Networks". In: *Computer Graphics Forum* (2020). ISSN: 1467-8659. DOI: 10.1111/cgf.13889.

[14] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[15] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. eprint: arXiv:1512.03385.

[16] Kaiming He et al. *Momentum Contrast for Unsupervised Visual Representation Learning*. 2019. DOI: 10.48550/ARXIV.1911.05722. URL: https://arxiv.org/abs/1911.05722.

[17] David J. Heeger and James R. Bergen. "Pyramid-Based Texture Analysis/Synthesis". In: *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '95. New York, NY, USA: Association for Computing Machinery, 1995, 229–238. ISBN: 0897917014. DOI: 10.1145/218380.218446. URL: https://doi.org/10.1145/218380.218446.

[18] Philipp Henzler, Niloy J. Mitra, and Tobias Ritschel. *Learning a Neural 3D Texture Space from 2D Exemplars*. 2019. eprint: arXiv:1912.04158.

[19] R Devon Hjelm et al. *Learning deep representations by mutual information estimation and maximization*. 2018. DOI: 10.48550/ARXIV.1808.06670. URL: https://arxiv.org/abs/1808.06670.

[20] Tero Karras et al. *Alias-Free Generative Adversarial Networks*. 2021. eprint: arXiv:2106.12423.

[21] Tero Karras et al. *Progressive Growing of GANs for Improved Quality, Stability, and Variation*. 2017. DOI: 10.48550/ARXIV.1710.10196. URL: https://arxiv.org/abs/1710.10196.

[22] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

[23] Johannes Kopf et al. "Solid Texture Synthesis from 2D Exemplars". In: *ACM SIGGRAPH 2007 Papers*. SIGGRAPH '07. San Diego, California: Association for Computing Machinery, 2007, 2–es. ISBN: 9781450378369. DOI: 10.1145/1275808.1276380. URL: https://doi.org/10.1145/1275808.1276380.

[24] Vivek Kwatra et al. "Graphcut Textures: Image and Video Synthesis Using Graph Cuts". In: *ACM Trans. Graph.* 22.3 (2003), 277–286. ISSN: 0730-0301. DOI: 10.1145/882262.882264. URL: https://doi.org/10.1145/882262.882264.

[25] Lin Liang et al. "Real-Time Texture Synthesis by Patch-Based Sampling". In: *ACM Trans. Graph.* 20.3 (2001), 127–150. ISSN: 0730-0301. DOI: 10.1145/501786.501787. URL: https://doi.org/10.1145/501786.501787.

[26] Ben Mildenhall et al. *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*. 2020. DOI: 10.48550/ARXIV.2003.08934. URL: https://arxiv.org/abs/2003.08934.

[27] Jeong Joon Park et al. *DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation*. 2019. DOI: 10.48550/ARXIV.1901.05103. URL: https://arxiv.org/abs/1901.05103.

[28] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[29] Ethan Perez et al. *FiLM: Visual Reasoning with a General Conditioning Layer*. 2017. arXiv: 1709.07871.

[30] Ken Perlin. "An image synthesizer". In: *SIGGRAPH '85*. 1985.

[31] Tiziano Portenier, Siavash Bigdeli, and Orcun Goksel. *GramGAN: Deep 3D Texture Synthesis From 2D Exemplars*. 2020. eprint: arXiv:2006.16112.

[32] Vincent Sitzmann et al. *Implicit Neural Representations with Periodic Activation Functions*. 2020. arXiv: 2006.09661.

[33] Towaki Takikawa et al. *Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes*. 2021. DOI: 10.48550/ARXIV.2101.10994. URL: https://arxiv.org/abs/2101.10994.

[34] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[35] Li-Yi Wei. "Texture Synthesis by Fixed Neighborhood Searching". AAI3038169. PhD thesis. Stanford, CA, USA, 2002. ISBN: 0493520031.

[36] Richard Zhang. "Making Convolutional Networks Shift-Invariant Again". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 7324–7334. URL: https://proceedings.mlr.press/v97/zhang19a.html.

[37] Xin Zhao et al. *Solid Texture Synthesis using Generative Adversarial Networks*. 2021. eprint: arXiv:2102.03973.

[38] Bolei Zhou et al. "Learning Deep Features for Scene Recognition using Places Database". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper/2014/file/3fe94a002317b5f9259f82690aeea4cd-Paper.pdf.