Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Mathias Rønning

Deep Procedural Generation of 3D Objects Using Real-World Data

Masteroppgave i Datateknologi
Veileder: Bart Iver van Blokland
Juni 2023

NTNU
Kunnskap for en bedre verden

Mathias Rønning

Deep Procedural Generation of 3D Objects Using Real-World Data

Masteroppgave i Datateknologi
Veileder: Bart Iver van Blokland
Juni 2023

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU

Kunnskap for en bedre verden

Abstract

3D object recognition algorithms are used in various fields including virtual reality, autonomous vehicles, and robotics to determine useful information about a 3D scene such as object positions, poses, and classes. The evaluation of such algorithms relies on a large number of 3D objects from a wide range of categories. Currently, no genuinely large and diverse dataset of 3D objects exists, making the task of evaluating 3D recognition algorithms difficult. The main reason behind the absence of large and diverse 3D datasets is the time and resource intensive process of creating 3D objects. As such, this thesis explores an alternative approach to 3D object dataset creation by procedurally generating the objects using deep learning, which can greatly reduce the time required to generate a large 3D object dataset.

The thesis focuses specifically on deep procedural generation of 3D objects using real-world image data for training, with the rationale that real-world data is easier to acquire and can produce more realistic results. Accordingly, a state-of-the-art 3D generative deep model is trained using two different real-world image datasets, and the resulting generated objects are evaluated through different evaluation metrics. While the results obtained suggest that the proposed method is not capable of generating a novel 3D object dataset of adequate quality, a foundation is laid for future works to improve on.

Sammendrag

Algoritmer for 3D-objektgjenkjenning brukes i diverse felter som virtuell virkelighet, selvkjørende biler og robotikk for å innhente nyttig informasjon i tredimensjonelle systemer som posisjoner, positurer og klasser til objekter. Å evaluere slike algoritmer krever en stor mengde 3D-objekter fra et bredt spekter av kategorier. For øyeblikket eksisterer det ikke noe virkelig stort og mangfoldig datasett av 3D-objekter, som gjør å evaluere algoritmer for 3D-objektgjenkjenning til en vanskelig oppgave. Hovedgrunnen til manglen på et slikt datasett er at det krever mye tid og ressurser å lage 3D-objekter. Derfor utforsker denne oppgaven en alternativ fremgangsmåte for å lage et 3D-objektdatasett ved å prosedyrisk generere objektene ved bruk av dyp læring. Hensikten er å kunne drastisk redusere tiden som kreves for å generere et stort 3D-objektdatasett.

Oppgaven fokuserer spesifikt på dyp prosedyrisk generering av 3D-objekter ved å trene med reell bilde-data, med begrunnelse at reell data er enklere å anskaffe og kan produsere mer realistiske resultater. Følgelig blir en aktuell tredimensjonell dyp generativ model trent med to forskjellige reelle bilde-datasett, og resultatene blir evaluert med forskjellige evalueringsmetoder. Selv om de oppnådde resultatene tilsier at den foreslåtte metoden ikke er tilstrekkelig for å generere et nytt datasett av 3D-objekter, blir et grunnlag lagt som fremtidige metoder kan forbedre.

Preface

This thesis is part of the Master of Science degree in Computer Science a the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway, and was written during the spring of 2023. The thesis is supervised by Bart Iver van Blokland within the Visual Computing Group at the Department of Computer Science.

I would like to thank my supervisor Bart van Blokland for the support and feedback during the duration of the thesis. Additionally, I would like to thank everyone who participated in the user survey for helping make the thesis possible.

Mathias Rønning
Trondheim, 10th June 2023

Contents

List of Figures

List of Tables

Acronyms

CNN Convolutional Neural Network.

CO3D Common Objects in 3D.

FID Fréchet Inception Distance.

GAN Generative Adversarial Network.

NeRF Neural Radiance Field.

SDF Signed Distance Field.

1. Introduction

Procedural generation of 3D objects is a technique often used in video games and animated movies to generate 3D shapes algorithmically rather than relying on manual creation. By developing algorithms that can generate 3D objects with little human intervention, the time and resources spent on 3D modeling can be greatly reduced. Traditional methods for procedural generation are typically rule-based, with L-systems (Lindenmayer, 1968) and fractals (Carpenter, 1980) being examples of such methods. These methods have proven successful in generating organic shapes such as terrains or trees, but generation of human-made objects is generally more difficult (Freiknecht and Effelsberg, 2017). In recent years deep learning has emerged as a significant field of research, and deep generative methods have proven to be successful in generating various forms of content, including images (Rombach et al., 2022; Karras et al., 2019). However, using deep learning for 3D object generation is still in its early stages of development. As such, this thesis explores the state-of-the art of deep 3D procedural generation, and how it may be used to generate 3D objects.

This chapter explains the motivation behind the thesis, and introduces the goal and research questions proposed to achieve the thesis goal.

1.1. Background and Motivation

The field of 3D object recognition has since its inception evolved to become an active area of research. 3D object recognition algorithms solve the problem of inferring useful information about a 3D scene such as object poses and categories, and are used in fields such as augmented reality, virtual reality, autonomous vehicles, medical imaging, and robotics (Qi et al., 2021). Traditional algorithms achieve this through the use of various techniques, including 3D descriptors and sliding windows. In more recent years however, deep learning has been introduced to the field of 3D object

recognition (Su et al., 2015; Qi et al., 2016), and shown to outperform traditional methods.

In order to evaluate 3D object recognition algorithms, as well as training deep learning based methods, a large amount of 3D objects are required. Current algorithms are typically evaluated using the ShapeNet (Chang et al., 2015) and ModelNet (Wu et al., 2014) datasets that consist of several thousand objects from various categories. While these datasets are currently used to evaluate algorithms, they are lacking in terms of both diversity and scale. There is thus a need for a large-scale, diverse dataset of 3D objects. 3D objects are however difficult to acquire, requiring either manual creation by a skilled 3D artist or acquisition from the real-world using 3D scanners or photogrammetry, all of which are time and resource intensive processes.

As an alternative to manual acquisition of 3D objects, procedural generation techniques provide a less time-consuming and resource-intensive method of generating them. By creating an algorithm that can automatically generate 3D objects on the fly, the time and cost of creating a large dataset may be greatly reduced. A diverse, large-scale dataset of 3D objects may thus more easily be obtained through procedural generation than manual acquisition.

Recent work in procedural generation has utilized deep learning to generate 3D objects and shown promising results (Nash et al., 2020; Gao et al., 2022; Poole et al., 2022). In addition to being more efficient in generating novel objects, a deep neural network model is also significantly smaller in size than a large-scale dataset, and can be used to generate objects on the fly. However, existing methods usually require training on pre-existing 3D datasets. As such, the generated objects do not significantly diverge from those present in the datasets. With current datasets being limited in diversity and scale, employing a deep model trained on these is not a viable approach to generate a novel dataset.

Little research have been done on generating 3D objects from real-world image data, excluding reconstruction algorithms that do not aim to generate novel objects (Mildenhall et al., 2020; Müller et al., 2022), instead reconstructing existing ones. The advantages of using real-world image data is its higher degree of accuracy in representing reality, in addition to being easier to acquire in large quantities. By using real-world data to

train a deep 3D generative model, realistic and diverse 3D objects may be obtained. Accordingly, this thesis explores the possibility of generating a novel dataset of 3D objects using a deep procedural 3D model trained on real-world image data.

Though the use of real-world image data for 3D object generation offers the aforementioned advantages, it also presents significant challenges in comparison to synthetic data. Real-world image datasets typically lack or have algorithmically generated annotations such as camera information and segmentation masks that are commonly required by deep procedural 3D models. When using synthetic data in the form of rendered images of 3D objects these can be generated perfectly, while for real-world data the results may be imperfect and subject to noise.

1.2. Goals and Research Questions

Based on the thesis motivation presented in section 1.1, the goal of the thesis can be formulated as the following:

Goal *Explore the possibility of generating a large-scale dataset of 3D objects using a deep learning model trained on real-world data.*

In order to achieve this goal, a number of research questions are proposed to aid in the research process. Firstly, to be able to use a deep model for 3D procedural generation with real-world image data, the current state-of-the-art in deep 3D procedural generation must be evaluated. The first research question is thus as follows:

Research question 1 *To which degree can current deep learning methods generate 3D objects of high quality?*

To evaluate the current state-of-the-art deep 3D procedural generation models, it is first necessary to find a method to evaluate the similarity of the generated objects to their real-world counterparts. The second research question is thus as follows:

Research question 2 *How can the similarity of 3D objects to their real-world counterparts be objectively evaluated?*

Lastly, the state-of-the-art must be evaluated with regards to their ability to use real-world data for procedural generation. The final research question then becomes:

Research question 3 *To which degree are current deep 3D procedural models able to use real-world data for training?*

1.3. Thesis Structure

- **Chapter 1** introduces the thesis topic and the motivation behind it, in addition to bringing forward the goal and research questions.
- **Chapter 2** presents the background theory discussed throughout the thesis.
- **Chapter 3** discusses the current state-of-the-art 3D object datasets and deep 3D procedural models.
- **Chapter 4** presents the method used to evaluate the thesis research goal and research questions.
- **Chapter 5** presents the experiments conducted and the corresponding results.
- **Chapter 6** discusses the results in relation with the research questions and thesis goal.
- **Chapter 7** concludes the thesis.

1.4. Disclaimer

Parts of this thesis have been adapted from the specialization report conducted in the fall semester. These sections have been rewritten and altered to accommodate this thesis. The affected sections are 2.1.2, 2.1.3, 2.1.4, and 2.3.1.

2. Background Theory

This chapter covers the relevant background theory that the thesis builds upon. First the relevant background is given on how 3D data can be represented, which affects the performance and architecture of three-dimensional neural networks. Then a background is given on how computers render images using virtual 3D cameras, which is relevant to deep 3D procedural generation using image data. Lastly, the background theory regarding generative deep learning techniques is given as a basis for later architectural choices.

2.1. 3D Data Representations

There exist multiple methods for representing 3D information, each exhibiting its respective advantages and disadvantages. With regards to deep procedural generation, there are three main factors that influence their effectiveness. These are the memory footprint of the representation, compatibility with convolutional neural networks (CNNs), and their capability to be represented as a multi-layer perceptron.

The memory footprint affects the training time of the neural network, with high memory footprint representations requiring longer training times. This thus also affects the feasible resolution of the representation when used in a neural networks, which again influences the quality of the generated objects.

A representation being compatible with CNNs enables the transfer of the demonstrated advantages of two-dimensional CNNs (O'Shea and Nash, 2015) into three dimensions.

Representations that can be represented by a multi-layer perceptron (MLP) can be directly incorporated in a neural network, enabling the training of the representation in conjunction with the rest of the network.

This section gives a theoretic background on the more prevalent 3D

representations used with deep neural networks in order to clarify their advantages and disadvantages, and provides a basis for later architectural choices. The representations covered are voxels, point clouds, meshes, and signed distance fields (SDFs).

2.1.1. Voxels

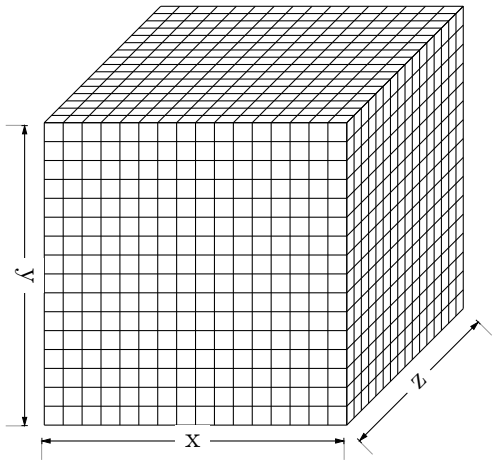


Figure 2.1.: The voxel representation as a three-dimensional cell grid

The voxel representation represents 3D information as a binary, three-dimensional occupancy grid of size $x \times y \times z$ as illustrated in Figure 2.1, where each cell is either occupied or non-occupied. When used with neural networks, the voxel dimensions are commonly set uniform at $n \times n \times n$ where $n = [64, 256]$ (Wu et al., 2016; Wang et al., 2018a; Ibing et al., 2021). Due to the uniform grid structure of voxels, which can be considered a three-dimensional extension of a binary image, traditional CNNs have been adapted for use with the voxel representation (Wu et al., 2016; Xie et al., 2019). This allows for neural networks using the voxel representation to benefit from the demonstrated advantages of CNNs. However, due to the redundant information stored in non-occupied cells and cells inside the object surface, in addition to the representation growing cubically with size, the memory footprint of voxels is high. This results in low resolution 3D objects when used for deep 3D procedural generation on modern hardware.

2.1.2. Point clouds

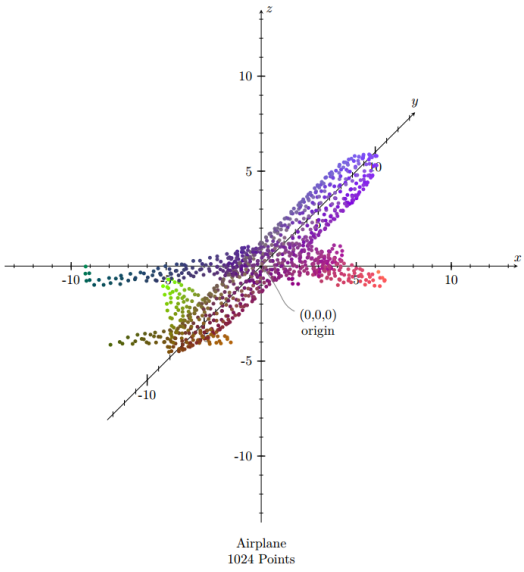


Figure 2.2.: A point cloud showing an airplane model of 1024 points. Figure generated using[1].

Point clouds are the standard representation used by 3D data acquisition devices such as the Kinect, LiDARs and phone depth cameras. Because of this, they are also frequently used by 3D recognition algorithms (Qi et al., 2016; Liu et al., 2019). A point cloud can be defined as an unordered set of 3D points $X = \{x_1, x_2, ..., x_n\}, x_i \in \mathbb{R}^3$ as illustrated in Figure 2.2. A *colored point cloud* additionally contains color information for each point. Since only points on the objects' surfaces are stored, the memory footprint of point clouds is small. However, due to their non-uniformity, 2D CNNs cannot be easily extended for use with point clouds. Specific convolution operations for point clouds have nonetheless been developed (Thomas et al., 2019; Wu et al., 2020) that allow for the creation of point cloud CNNs.

[1] `https://github.com/salehjg/pointcloud_to_tikz_converter`

2.1.3. Meshes

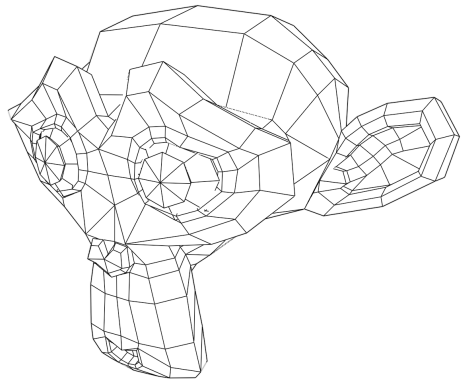


Figure 2.3.: The monkey mesh from Blender

Meshes are the predominant 3D representation used in computer graphics. A mesh is defined as a collection of vertices $\mathcal{V}$ and polygon faces $\mathcal{F}$ and an example of a mesh is illustrated in Figure 2.3. Each vertex $v \in \mathcal{V}$ is a 3D point and each face $f \in \mathcal{F}$ is a collection of co-planar vertices. Each vertex in the mesh can contain additional information such as normals and texture coordinates. Faces are typically limited to consist of 3 vertices, giving a triangle mesh, however n-gon meshes containing arbitrarily sized faces are also possible. Meshes generally have a slightly higher memory footprint than point clouds due to the topological information stored in the faces, though this depends on the point cloud resolution and the number of points needed to accurately represent the surface. Similarly to point clouds, meshes are non-trifvial to use with CNNs due to their non-uniform structure. However, specific mesh CNNs have been developed, such as MeshCNN (Hanocka et al., 2019).

2.1.4. Signed Distance Fields

Signed Distance Fields (SDFs) represent 3D data as a continuous function that for a given point $\mathbf{x}$ outputs the signed distance s to that point from its closest surface point, where the sign signifies whether the point is inside

or outside the surface:

$$SDF(\mathbf{x}) = s : \mathbf{x} \in \mathbb{R}^3, s \in \mathbb{R}$$

As such, the surface of the object is represented by $SDF(\cdot) = 0$. A discrete version of the function can be obtained by sub-dividing the 3D space into a grid and sampling the function at each grid point. The main advantage of the SDF representation is that the function can be expressed by an MLP and is therefore suited for use in an artificial neural network. Additionally, a 3D mesh can be obtained from the representation using the *marching cubes algorithm* (Lorensen and Cline, 1987). However, due to the use of a 3D grid, the memory footprint is similar to that of the voxel representation.

2.2. 3D Rendering

In order for a deep neural network to generate 3D objects by training the network on image data, a function that describes the relationship between the 3D data and the image is required. This function is commonly the *rendering function*, that maps 3D data to an image using a virtual camera. Virtual camera representations are inspired by real-world cameras, and act in a similar manner by projecting 3D points onto an image surface. This section covers how virtual cameras can render images from 3D data which lays a foundation for deep 3D procedural generation from image data.

2.2.1. Pinhole Camera Model

The most simple representation of a perspective camera is known as a pinhole camera, where the camera is modeled as a box with an infinitesimally small hole on one side, as illustrated in Figure 2.4. For ease of representation, the image plane is situated in front of the camera center instead of behind, which also prevents image inversion. The "pinhole" in located at $\mathcal{F}_c$, with the optical axis piercing through the image plane at the principal point. The focal length f is given by the distance from $\mathcal{F}_c$ to the principal point. For a point $P = (x, y, z) \in \mathbb{R}^3$ in the camera reference frame, the coordinates of the point projected onto the image plane $P' = (x', y')$ are then given by:

$$x' = f\frac{x}{z},\ y' = f\frac{y}{z}$$

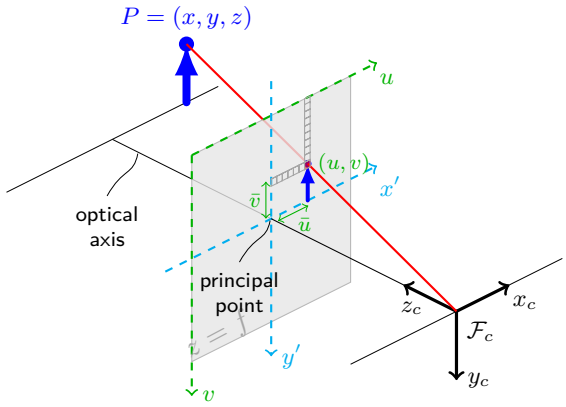


Figure 2.4.: The pinhole camera model

By dividing by the z coordinate, points at a further distance from the camera are scaled down futher, giving a perspective projection. The rationale behind the equations can be seen by considering the triangle displayed in Figure 2.5 showing the y' coordinate where $\frac{y}{z} = \frac{y'}{f}$ due to the similar triangles rule. A similar rationale can be made for the x' coordinate. The camera reference frame points and image points can be expressed using homogeneous coordinates, that is $P = (x, y, z, 1)$ and $P' = (x', y', 1)$. In this manner the perspective transformation can then be expressed in matrix form:

$$M = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

In order to obtain the pixel coordinates of the projected point $P'' = (u, v, 1)$, P' needs to be translated by u_0 and v_0 respectively, where (u_0, v_0) corresponds to the point in the upper left corner of the image. This is due to the image reference frame being located at the image center, while the pixel reference frame is located at the top left corner. The final matrix then becomes:

$$M = \begin{bmatrix} f & 0 & u_0 & 0 \\ 0 & f & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} I & 0 \end{bmatrix} = K \begin{bmatrix} I & 0 \end{bmatrix}$$

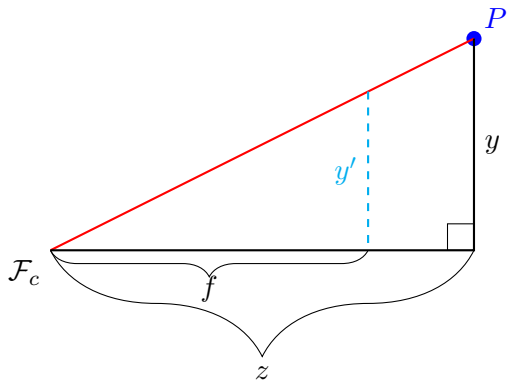


Figure 2.5.: The y' coordinate can be obtained by applying the similar triangles rule to these triangles

Where M is referred to as the *camera matrix* and K is referred to as the *intrinsic matrix*. As such, the projected pixel point P'' can be obtained by multiplying P with the camera matrix:

$$P'' = \begin{bmatrix} f & 0 & u_0 & 0 \\ 0 & f & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} P$$

This simplistic model does not take into account effects such as lens distortion and skew, but acts as a simplistic model for a camera's intrinsic parameters.

2.2.2. Extrinsic Parameters

The *camera matrix* acts upon on points in the camera reference frame. Thus, in order to apply the matrix on world reference frame points, they first need to be transformed from the world reference frame to the camera reference frame, as illustrated in Figure 2.6. This transformation can be represented by a rotation matrix R along with a translation vector $\mathbf{t}$, and are known as the *extrinsic parameters* of the camera. These are commonly combined into a *view matrix* $V = \begin{bmatrix} R & \mathbf{t} \\ 0 & 1 \end{bmatrix}$ that can be applied to homogeneous world

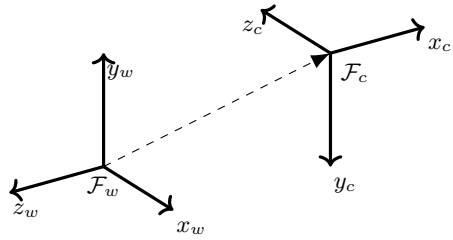


Figure 2.6.: Transformation from the world reference frame to the camera reference frame

coordinates. Thus, for a given world point $P_w = (x, y, z, 1)$ in homogeneous coordinates, the final image pixel coordinates $P'' = (u, v, 1)$ are obtained by multiplying the point with the intrinsic and extrinsic matrices respectively:

$$P_w = MV \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

2.2.3. Rasterization and Shading

While the camera model is able to project 3D points to 2D pixel coordinates, it does not describe how the points should be displayed. This is commonly done through a *rasterization* process followed by a *shading* process. The rasterization process typically takes 3D meshes as input and uses the camera matrix to project the mesh vertices to pixel coordinates, before determining which pixels are contained within the mesh faces. Each pixel is then output with corresponding vertex data which is interpolated between the face vertices. The pixels are then shaded according to a shading model, which uses parameters such as lights and textures in combination with the vertex data to determine the pixel color.

2.3. Generative Deep Learning

In order to generate 3D objects using deep learning, a generative deep learning model is required. Generative deep learning models aim to learn

a data distribution using unsupervised learning in order to generate new samples that are similar to samples from the same distribution. Different from *discriminative models*, that discriminate between decision boundaries in the data in order to predict labels of new data points by capturing the conditional probability $p(Y|X)$, generative models capture the joint probability $p(X, Y)$, where X is the data points and Y the labels. There exist many approaches to deep generative modeling, and to lay a foundation for the architectural choices taken in this thesis with regards to deep 3D procedural generation, this section gives a theoretic background on deep generative models. The techniques covered are the Generative Adversarial Network (GAN) architecture for generative modeling, the 3D-aware image synthesis technique for image generation consistent with 3D data, the Neural Radiance Field (NeRF) architecture for novel view synthesis, and the diffusion bases technique for image generation.

2.3.1. Generative Adversarial Networks

The GAN (Goodfellow et al., 2014) architecture is a generative deep learning architecture that trains two different neural networks simultaneously, a generator network G and a discriminator network D. The generator G aims to generate new plausible samples from the data distribution using a noise vector $\mathbf{z}$ as input, though with some variation to those present in the distribution. Simultaneously, the discriminator D estimates the probability that a sample comes from the training data or from G. In this manner, the two networks are playing a two-player minimax game where the gain of one network is the loss of the other. This game can be represented using the value function $V(G, D)$, where G aims to minimize the function and D aims to maximize it:

$$\min_G \max_D V(G, D) = \mathbb{E}_{\mathbf{x} \sim p_{data(\mathbf{x})}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{data(G(\mathbf{z}))}}[\log(1 - D(G(\mathbf{z})))]$$

Where the first addend denotes the log probability of D predicting that a real sample is genuine, and the second addend denotes the probability that D predicts a generated sample $G(\mathbf{z})$ to be non-genuine. Intuitively the game can be thought of as the generator learning to generate samples that "fool" the discriminator while the discriminator learns to determine whether a sample is "real" or "fake".

As a result of the simultaneous training of the two networks, GANs present significant challenges in the training process and have several common failure modes. In the case that the discriminator learns faster than the generator, the generator training can fail due to vanishing gradients. This is due to the loss for the generator equaling zero when the discriminator is optimal. Another failure case known as mode collapse can happen when the generator keeps producing a small number of outputs that are most plausible to the discriminator, producing little variety in the generated results. This is due to the discriminator getting stuck in a local minimum and determining the generated samples as real. GANs may also simply fail to converge in many cases. There are many approaches to minimize the GAN failure modes, such as using alternative loss functions like *Wasserstein loss* or adding noise to discriminator inputs.

A technique often applied to GANs to minimize failure modes is known as R1 regularization, with the aim of penalizing large updates to the discriminator weights when trained on real data. The regulariztion term is defined as follows:

$$R_1(\psi) = \frac{\gamma}{2} E_{p_D(x)} \left[||\nabla D_\psi(x)||^2 \right]$$

Where ψ is the discriminator weights, the *gamma* parameter γ is a hyperparameter of the GAN model and $E_{p_D(x)}$ denotes that the data is only sampled from the real distribution. As such, by changing the *gamma* parameter of the GAN network, one can avoid the discriminator out-learning the generator and find a common equilibrium.

2.3.2. 3D-aware Image Synthesis

The goal of 3D-aware image synthesis models is to generate images consistent with a 3D model or scene. Contrary to traditional image synthesis methods, 3D-aware image synthesis methods can generate images from specific camera angels, and in many cases a 3D object can be extracted. This is commonly done through the use of an internal 3D representation that is rendered to an image. In this manner, the deep model can be trained in image space while producing a 3D shape through the internal representation. Since most of deep learning methods use first order optimization techniques such as gradient descent, this requires the complete

architecture to be differentiable. As such, at the core of 3D-aware image synthesis models is a *differentiable renderer*.

A 3D rendering function $\mathcal{R}$ takes as parameters 3D scene information including geometric shapes, camera parameters, lighting parameters, and material information and outputs and image I. Thus, by parameterizing the input geometry, camera, materials, and lights as θ_G, θ_C, θ_M and θ_L respectively, the rendering function can be expressed as:

$$I = \mathcal{R}(\theta_G, \theta_C, \theta_M, \theta_L)$$

In order to make this function differentiable the gradients of the output image must be found with respect to the input parameters, and are given by $\frac{\partial I}{\partial \theta}$. The steps taken to find each gradient varies between the parameters and rendering process. Generally, the camera and geometry gradients can be found by inverting the perspective mapping which is inherently differentiable. The material and light gradients computation depends on the shading and lighting model used.

2.3.3. Neural Radiance Fields

Neural Radiance Fields (NeRFs) (Mildenhall et al., 2020) provide a method for synthesizing novel views of a 3D scene given a set of input images of that same scene. The scene is represented by a *multilayer perceptron* (MLP), a fully-connected deep neural network without any convolutional layers. The network takes as input a 5D coordinate (x, y, z, θ, ϕ), where $\mathbf{x} = (x, y, z)$ is a 3D coordinate and $\mathbf{d} = (\theta, \phi)$ is a viewing direction which in practice is expressed as a 3D Cartesian unit vector. Given the input, the network then outputs an emitted color $\mathbf{c} = (r, g, b)$ and a volume density σ. By employing a volumetric renderer that is naturally differentiable, images can be rendered by shooting a ray for each image pixel and sampling the output values $\mathbf{c}$ and σ along the ray, before integrating over the values. More formally, given an image position $\mathbf{x}$ and viewing direction $\mathbf{d}$, a ray origin $\mathbf{o}$ can be found at each image pixel, assuming the camera extrinsic parameters are known. By denoting the ray timesteps as t, the ray can be expressed as a function of time given by $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$. The expected outputted color $C(\mathbf{r})$ is then given by the following integral:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt$$

Where t_n and t_f are the near and far bounds of the ray and $T(t)$ is the accumulated transmittance given by:

$$T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right)$$

The network can be trained by shooting a ray for each image pixel of the dataset images and recording the outputted color $\mathbf{c}$ and volume density σ, which can then be used to find the L2 loss $||\mathbf{c} - \mathbf{c}_{gt}||^2$, where $\mathbf{c}_{gt}$ denotes the ground truth pixel value. A 3D mesh can be extracted from the NeRF by sampling it and using the *marching cubes algorithm*.

Since NeRFs represent a 3D scene within an MLP, a separate neural network must be trained for each scene. The original NeRF paper states that the training time for a single scene takes about 1-2 days on a single NVIDIA V100 GPU (Mildenhall et al., 2020). As such, NeRFs require a significant amount of processing power in order to generate a large amount of objects. Newer implementations aim to improve on this (Müller et al., 2022) by reducing the neural network size and applying an input encoding.

2.3.4. Diffusion

Diffusion based generative models have gained popularity in text-to-image synthesis, and shown to outperform GAN based methods in many cases (Rombach et al., 2022). A generative diffusion model adds Gaussian noise to the data points step-wise, and then learns to reverse the diffusion process in order to generate new samples. More formally, given a data point sampled from a data distribution $\mathbf{x}_0 \sim q(\mathbf{x})$, the *forward diffusion process* can be defined as:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

Where $t \in [0, T]$ is the time step and $\{\beta_t \in [0, 1]\}_{t=1}^{T}$ is the *variance schedule* that describes the amount of noise to add at each time step. It is possible to find $\mathbf{x}_t$ of a given timestep using:

$$\mathbf{x}_t = \sqrt{\bar{\mathbf{a}}_t}x_0 + \sqrt{1-\bar{\mathbf{a}}_t}\epsilon \tag{2.1}$$

Where $\bar{\mathbf{a}}_t = \prod_{s=0}^{t} 1 - \beta_t$ and $\epsilon \sim \mathcal{N}(0, \mathbf{I})$. As $T \to \infty$, $\mathbf{x}_T$ approaches an isotropic Gaussian distribution. The *reverse diffusion process* is then to

learn the joint distribution $p_\theta(\mathbf{x}_0 : T)$ given by:

$$p_\theta(\mathbf{x}_0 : T) := p(\mathbf{x}_T)\Pi_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := p(\mathbf{x}_T)\Pi_{t=1}^{T}\mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$$

This can then be used to generate new samples from Gaussian noise, by gradually de-noising the input.

3. Related Work

This chapter presents the related work with regards to procedural generation of 3D objects using deep learning. First an overview of current 3D object datasets is given, and then the current state-of-the-art in deep procedural 3D generation is given.

3.1. 3D Object Datasets

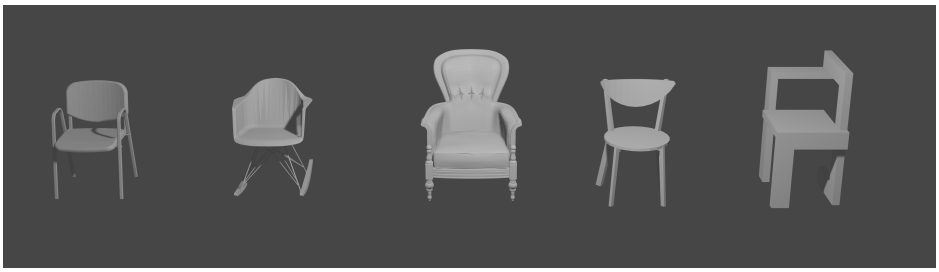


Figure 3.1.: Examples of ModelNet40 3D objects from the chair category. The objects have been scale adjusted.

While the availability of large-scale 3D object datasets is limited, there do exist several datasets of a smaller size. ModelNet (Wu et al., 2014) is a dataset consisting of 127,915 3D CAD models from 662 categories, and is commonly used for 3D object classification and retrieval benchmarking (Qi et al., 2016; Wu et al., 2016). While the full dataset consists of 127,915 CAD objects, they are not cleaned or pose aligned and are thus of varying quality. For this reason, two separate splits of the dataset have been created, namely ModelNet10 and ModelNet40. These splits contain fewer objects from less categories, but have been manually cleaned. The ModelNet10 split contains in total 4,899 objects from 10 categories that have been

manually cleaned and pose aligned, while the ModelNet40 split contains 12,311 objects that have been manually cleaned but not pose aligned. This dataset has not been used extensively for deep 3D object generation due to its size generally being a limitation for producing high-quality objects. However, the ModelNet40 and ModelNet10 datasets are widely used for 3D classification and retrieval tasks, with benchmark results available on the ModelNet website[1]. Some examples of ModelNet40 objects from the chair category are displayed in Figure 3.1.

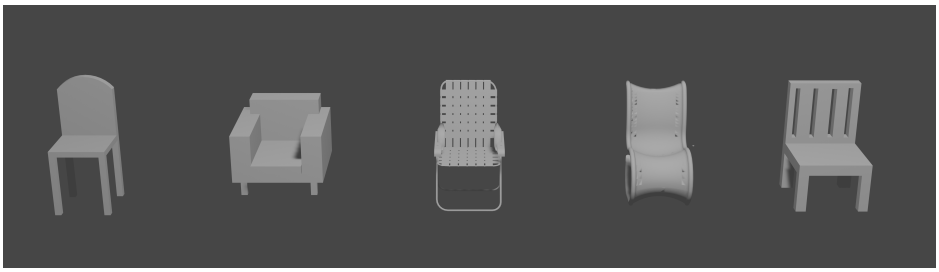


Figure 3.2.: Examples of ShapeNet 3D objects from the chair category with textures removed.

ShapeNet (Chang et al., 2015) is another 3D object dataset used extensively in computer graphics, computer vision, and robotics research. The objects are collected from a multitude of online 3D repositories before being annotated. While the full original dataset consists of roughly 3,000,000 objects, it is not publicly available. Instead, the publicly available dataset consists of two different subsets; ShapeNetCore with about 51,300 cleaned objects from 55 different categories and ShapeNetSem with 12,000 objects from 270 categories that are annotated with information such as real-world dimensions, material properties, and volume and weight estimates. ShapeNet has been used to evaluate a variety of methods for different tasks, including 3D object recognition (Qi et al., 2016) and shape reconstruction (Xie et al., 2019), in addition to commonly being used to evaluate deep 3D procedural generation models (Gao et al., 2022; Wu et al., 2016). While deep 3D procedural neural networks trained on the ShapeNet dataset are able to generate novel objects, the dataset size and categories is still a

[1] https://modelnet.cs.princeton.edu/

limitation for producing high-quality, varied 3D objects. Additionally, since ShapeNet objects are collected from online 3D repositories, the realism of the objects can vary significantly. Some examples of ShapeNet objects from the chair category are displayed in Figure 3.2.

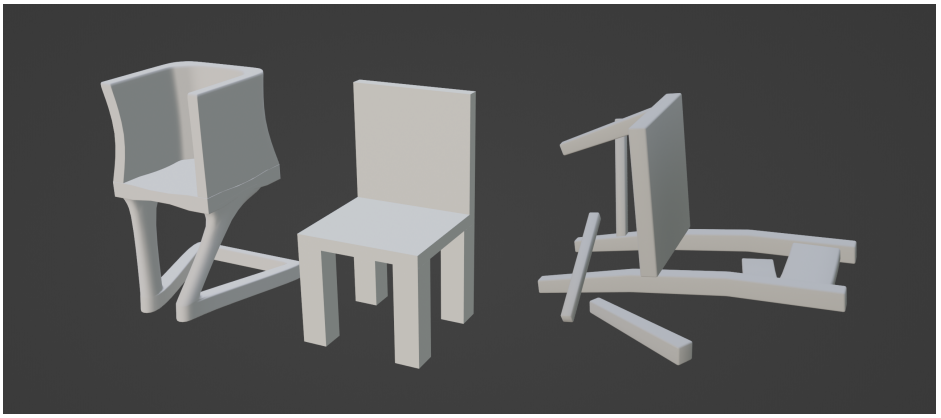


Figure 3.3.: Examples of poor quality and unrealistic models (without their textures) from the Objaverse chair category.

Objaverse (Deitke et al., 2022) is a recently published dataset of 3D objects with over 800,000 annotated 3D objects. With its extensive collection of objects, it can be argued to be the only dataset currently available that is genuinely large-scale. The objects present in the dataset are sourced from SketchFab[2], a web-based platform for 3D model sharing, where the models collected are created by over 100,000 different artists. As such, the models are of varying quality and realism, with a few examples of poor quality and unrealistic objects from the chair category displayed in Figure 3.3. Additionally, the annotation categories are broad in nature, including categories such as "architecture" and "science-technology". This makes retrieval of objects of a specific category difficult. A subset of the dataset is provided with annotated LVIS (Gupta et al., 2019) categories, though this consists of only 47,000 objects which is less than ShapeNet. As such, the Objaverse dataset is still limited for deep procedural 3D object generation of specific categories.

[2] https://sketchfab.com

3.2. Deep 3D Generative Models

Procedural generation of 3D objects using deep learning has become a prevalent field of research in recent years. With many different industries requiring a large amount of 3D objects, for purposes such as creating large 3D virtual worlds, deep 3D generative models have been proposed to automate the process of object generation. Multiple approaches have been suggested, and this section provides an overview of several methods using different approaches.

3.2.1. GAN Based Methods

Several methods have been proposed for using the GAN architecture for procedural 3D object generation. 3D-GAN (Wu et al., 2016) is an early approach that utilizes the voxel representation with a 3D convolutional neural network in order to generate novel 3D objects. The generator G maps a randomly sampled latent vector $\mathbf{z} \in \mathbb{R}^{200}$ to a $64 \times 64 \times 64$ voxel representation through the convolutional network, and the discriminator D gives a confidence value $D(x)$ of whether the 3D object x is real or fake. By training the model on single ShapeNet categories, it is able to generate novel objects, though at a low quality and low $64 \times 64 \times 64$ resolution. Other more recent approaches have expanded on the architecture (Wang et al., 2018a), but are generally limited by the feasible resolution of the voxel representation.

Shu et al. (2019) propose a GAN for generating 3D point clouds called tree-GAN. By representing the point clouds using a tree structure and using a tree-structured graph convolution network called TreeCGN, new samples can be generated from a noise vector $\mathbf{z}$. Different from other methods, tree-GAN can generate point clouds from different categories without training on each category separately. However, the model still requires 3D data for training, with ShapeNet being used to evaluate the model. Additionally, the generated point clouds consist of only 2048 points, resulting in lower resolution objects that lack finer details.

GET3D (Gao et al., 2022) is a deep generative model developed by NVIDIA to generate textured 3D objects. It utilizes the SDF based hybrid 3D representation DMTet (Shen et al., 2021) in combination with a differential renderer (Laine et al., 2020) and StyleGAN (Karras et al.,

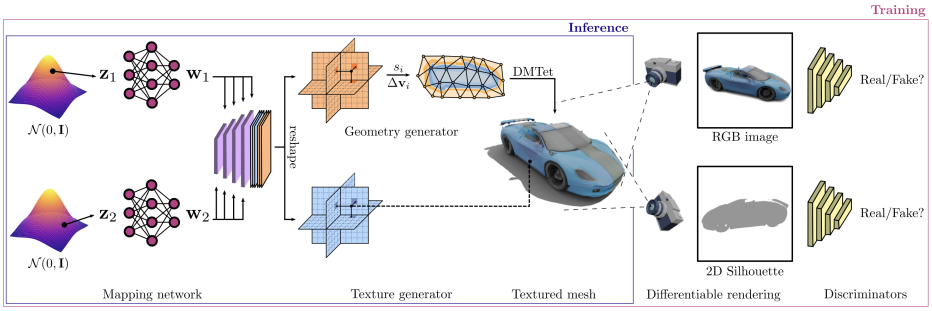


Figure 3.4.: The GET3D model architecture. Figure adapted from Gao et al. (2022)

2019). The generative network architecture is split in two, with a geometry generator and a texture generator. Given a Gaussian distributed variable $\mathbf{z} \in \mathcal{N}(0, \mathbf{I})$, the model aims to generate a mesh M with a corresponding texture T. The model architecture is illustrated in Figure 3.4. This figure includes the improved generator architecture from the GET3D paper apendix, which combines the texture and geometry generator into the same backbone. For simplicity, the architecture covered in this section is the architecture proposed in the main paper, which separates the texture and geometry generators.

The GET3D geometry generator takes as input a Gaussian variable $\mathbf{z}_1$ that is mapped to a latent vector $\mathbf{w}_1 \in \mathbb{R}^{512}$ through a mapping network. $\mathbf{w}_1$ is further mapped to SDF values through a series of 3D convolutional layers. The convolutional layers output an SDF value $s_i \in \mathbb{R}$ and a vertex deformation $\Delta\mathbf{v}_i$ for each vertex $\mathbf{v}_i \in V_T$ on a tetrahedral grid T. These are then passed into DMTet which uses a *differential marching tetrahedra* algorithm to extract a 3D mesh. By using a deformable tetrahedral grid, the mesh can be extracted more efficiently than through *marching cubes* using a cubical grid.

The texture generator works in a similar manner, taking a Gaussian variable $\mathbf{z}_2$ as input that is mapped to a latent vector $\mathbf{w}_2$. The concatenated latent vector $\mathbf{w}_1 \oplus \mathbf{w}_2$ is then used to generate a 3D texture field through the use of a conditional 2D convolutional network. 3D texture fields are a texture representation introduced by Oechsle et al. (2019) that maps a 3D point $\mathbf{p} \in \mathbb{R}^3$ to an RGB color $\mathbf{c} \in \mathbb{R}^3$.

Using the generated mesh and texture field, the object is rendered to a 2D image and segmentation mask using a differentiable renderer given a camera rotation and elevation angle. These are then passed to two separate StyleGAN discriminators that determine whether they are real or fake. As such, the GET3D model can be trained using 2D images of single objects in combination with their corresponding segmentation masks and camera angles. In order to generate novel objects using the trained model, the variables $\mathbf{z}_1$ and $\mathbf{z}_2$ can be randomly sampled and provided to the network.

The paper presents results after training the model on synthetic images. These images are generated by rendering 3D objects of a specific category using Blender, using camera angles evenly distributed along a hemisphere. The results presented shows that the model achieves state-of-the art performance, outperforming existing 3D-aware image synthesis methods such as GRAF(Schwarz et al., 2021) and EG3D(Chan et al., 2022).

3.2.2. Transformer Based Methods

Transformers are a neural network architecture first proposed in 2017 (Vaswani et al., 2017) by Google researchers. Originally proposed for sequential data, the transformer architecture aims to solve issues related to recurrent neural networks, including computational complexity and the inability to retain long-term information. The transformer architecture has later been used with inherently non-sequential data such as images and 3D objects. PolyGen (Nash et al., 2020) uses the transformer architecture with a 3D mesh representation by representing the meshes sequentially. This is done by first ordering the vertices $\mathcal{V}$ in order of their z, y and x coordinates, and then ordering the faces $\mathcal{F}$ in order of their respective vertices. The model is evaluated on the ShapeNet dataset, where the model is trained on the whole dataset using different conditioning variables, including class labels, images, and voxels. Log-likelihood is used as the evaluation metric, that is stated to correlate well with the sample quality. The results show that the model is able to produce high-quality objects, particularly on high-frequency categories of the ShapeNet dataset, though the model is dependent on 3D data for training. As such, due to the limited prevalence of objects of specific categories, the generated results on such categories are inferior.

Octree Transformer (Ibing et al., 2021) is another proposed transformer

based method for 3D object synthesis. The 3D data is represented sequentially using an octree of voxels, which reduces the memory footprint in comparison to rudimentary voxels. The model is evaluated on a voxelized version of the ShapeNet dataset, and is demostrated to outperform other voxel based models such as 3DGAN. It is still however limited by being trained on 3D data in addition to having a high memory footprint despite using octrees.

3.2.3. Diffusion Based Methods

DreamFusion (Poole et al., 2022) proposes a method for text-to-3D synthesis by employing the diffusion text-to-image synthesis model Imagen (Saharia et al., 2022) in combination with NeRFs. Given a text prompt, a NeRF is randomly initialized and then rendered to an image using a random lighting direction and normals computed using the NeRF volume density gradients. The rendered image is then diffused using the *forward diffusion process* and reconstructed using the Imagen model. The reconstructed image is then used to update the NeRF weights. Since a separate NeRF must be trained for each text prompt, and a single NeRF contains only a single object, the time required to generate a multitude of objects is long, with the paper stating that training a single NeRF takes 1.5 hours on a TPUv4 machine. As such, using DreamFusion to generate a novel dataset of 3D objects would be substantially time and resource intensive.

Magic3D (Lin et al., 2023) aims to reduce the DreamFusion training time and increase the object resolution. The training time is reduced by employing a hash encoding from Instant NGP (Müller et al., 2022) to represent the NeRF at a lower computational cost. Additionally, Magic3D employs a coarse-to-fine optimization approach, first optimizing using a low resolution diffusion prior before fine-tuning at a higher resolution. The paper states that Magic3D can generate objects in 40 minutes, and that user studies showed that 61.7% preferred the Magic3D-generated objects over DreamFusion ones. However, for the purpose of generating a large dataset of 3D objects, Magic3D is still significantly time and resource intensive.

Point-E (Nichol et al., 2022b) proposes a diffusion based method for generating 3D point clouds from a single text prompt. A text-to-image model is paired with an image-to-3D model such that a 3D object can be

produced by first sampling an image and then a 3D object. The models are trained using a non-public dataset of 3D objects, where the objects are rendered into RGBAD images from 20 random camera angles and point clouds produced from the images. A GLIDE (Nichol et al., 2022a) model is fine-tuned using a mixture of its original dataset and the generated dataset of 3D renderings to produce the text-to-image model. For the image-to-3D model, each point cloud is represented as a $K \times 6$ tensor, with K being the number of points and (x, y, z, R, G, B) being the point vectors containing position and color. A transformer based model is used to predict t, ϵ, the variance Σ of $p_\theta(x_{t-1}|x_t)$, and $\mathbf{x}_t$ from 2.1 in order to denoise random point cloud noise of shape $K \times 6$ and generate a point cloud. The model performs worse than other state-of-the-art text-to-3D models such as DreamFusion, however is able to generate objects in a much shorter time period at a number of seconds.

4. Method

This chapter aims to arrive at a method for generating 3D objects using real-world image data, to evaluate the thesis goal of generating a novel 3D dataset. Real-world data is used with the intent that the generated objects will resemble realistic objects to a greater degree. Additionally, acquisition of real-world image data is considerably less time-consuming than acquiring 3D data directly. As such, by certifying that the method is reliable, future works can acquire real-world image data to a greater degree to increase the quality and diversity of generated objects. In order to arrive at the method, several state-of-the-art 3D generative models are tested and evaluated to find a suitable architecture. After arriving at an architecture, the datasets used for training can be determined.

4.1. Architecture

This section presents the chosen architecture for deep 3D procedural generation with real-world image data and covers the steps taken to arrive at this architecture. While it would be possible to develop a specific architecture for this task, it would require a significant effort and expertise, thus surpassing the scope of this thesis. As such, this thesis instead focuses on using an existing architecture with real-world data.

4.1.1. Unused Architectures

The first research question of this thesis is as follows:

Research question 1 *To which degree can current deep learning methods generate 3D objects of high quality?*

To aid in answering this question and arrive at candidates for suitable architectures, several deep 3D generative models were considered and

tested. Some of the architectures were abandoned due to poor generation results.

Figure 4.1.: Objects generated from single RGB images using Pixel2Mesh.

Initially, the image-to-3D model Pixel2Mesh (Wang et al., 2018b) was tested. This model can generate 3D meshes from a single RGB image. While the model is trained on rendered images of ShapeNet objects from 13 categories, it can be conditioned on arbitrary images from the same categories. As such, the model was tested using conditioning on real-world images. The results are displayed in Figure 4.1. As can be seen from the results, the model was not able to generate 3D objects of a satisfactory quality in any of the cases. As such, it was disregarded as a candidate for a suitable architecture.

The image-to-3D model Pix2Vox (Xie et al., 2019) was also considered and tested. Similarly to Pixel2Mesh, this model can generate 3D objects from a single RGB image, though it uses a voxel representation as opposed to meshes. It is similarly trained on rendered images of ShapeNet objects from 13 categories. To facilitate the testing on custom images, the official implementation was forked and modified, and is available on Github[1]. The results after conditioning the model on real-world images from the

[1] https://github.com/Maro1/Pix2Vox

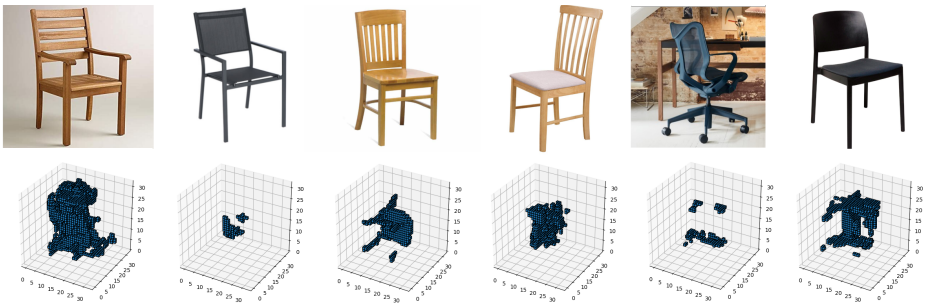


Figure 4.2.: Objects generated from single RGB images of chairs using Pix2Vox.

chair category are displayed in Figure 4.2. Since the chair category is the ShapeNet category with the highest number of samples, and the generated results are not of a high quality, Pix2Vox was disregarded as a candidate for a suitable architecture.

4.1.2. Evaluating the State-of-the-Art

Additional architectures were considered as potential candidates for 3D procedural generation using real-world data. These architectures produced better initial results, and were thus further evaluated. Firstly, PolyGen (Nash et al., 2020) was chosen due to its efficient generation on multiple categories and high log-likelihood score. It can additionally be conditioned on image data. Secondly, DreamFusion (Poole et al., 2022) was chosen due to being conditioned on text and thus being able to generate from an abundance of categories, in addition to outperforming several similar models. While Magic3D is stated to outperform DreamFusion, it does not have a public implementation and can thus not be evaluated. Lastly, GET3D was chosen because of its ability to produce high quality objects from image data in addition to outperforming related methods. The three models all use three different deep generative modeling approaches, namely Transformers, Diffusion, and GANs, and thus provide a diverse representation of the current state-of-the-art. As such, they lay a foundation for answering research question 1.

Model Evaluation

The second research question of this thesis considers the evaluation of the resemblance of 3D objects to their real-world counterparts:

Research question 2 *How can the similarity of 3D objects to their real-world counterparts be objectively evaluated?*

Evaluating the quality of 3D objects is a difficult problem, and there does not exist an established evaluation metric. Existing metrics typically rely on comparing the generated results to a ground-truth. *Chamfer distance* is an example of such a metric that calculates the distance between two point clouds by taking each point into account. This however requires a ground-truth point cloud to compare against, and since two of the models use image data for training it would be impractical to use. Similarly, an evaluation metric acting on images would be impractical to use since PolyGen is trained on 3D data.

Since the approach of comparing the generated results to a ground-truth is not applicable between the three chosen models, a different approach is taken by using a 3D classification network to classify the generated objects of each model. In this manner, the resemblance of the generated objects to the category they were conditioned on can be assessed. While this method does not evaluate the particular object's quality, and only its resemblance to a category, it serves as an objective evaluation metric regardless of the ground-truth and is thus deemed adequate for evaluation purposes.

To arrive at a suitable 3D classification network, the ModelNet40 classification benchmark (Wu et al., 2014) was reviewed. Since several of the models achieve a similar classification percentage at about 90%, the availability and simplicity of public implementations were also taken into account. This concluded with the PointNet model being chosen due to achieving a competitive benchmark score in addition to having several public implementations.

PointNet is a network architecture for classification and segmentation of point clouds. The baseline model achieves an accuracy of 89.2% on the ModelNet40 benchmark, which is a moderate score in comparison to the other listed algorithms. However, a PyTorch implementation available on Github[2] is used due to its simplicity in addition to containing code

[2] https://github.com/yanx27/Pointnet_Pointnet2_pytorch

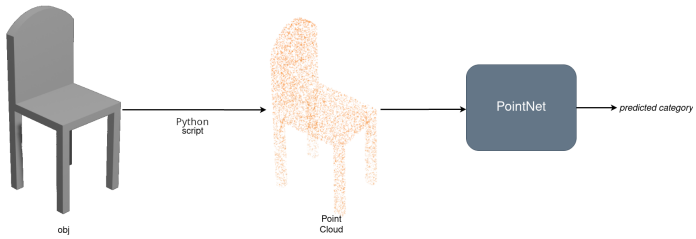


Figure 4.3.: The object evaluation process using PointNet.

for training on the ModelNet (Wu et al., 2014) and ShapeNet (Chang et al., 2015) datasets. This implementation includes the improved PointNet network model PointNet++ (Qi et al., 2017) that is stated to reach an accuracy of 92.8% for the *PointNet2_MSG (Pytorch with normal)* model. This is the model used for the evaluation and it is trained on the ModelNet40 dataset following the README.

Since the PointNet model takes point clouds as input while the models used output obj mesh files, a Python script is created to convert the obj files into point clouds. The script samples points evenly on the surface and outputs a point cloud containing 8192 points. This number of points was deemed sufficient to accurately model the mesh surface while still not impacting performance. PointNet is then run with each point cloud as input and the predicted class is recorded. The process is illustrated in Figure 4.3.

To evaluate the models, 100 objects are generated using the PolyGen and GET3D models on two different categories; chairs and tables. This is due to these two categories being the only two ShapeNet categories where GET3D provides pre-trained weights, and are thus common among all the models. Since the DreamFusion inference time is substantial due to having to train the model for each new object, only 10 objects of each category were generated using it. Additionally, ShapeNet is used as a baseline by sampling 100 objects from the two categories. The objects are then classified using the PointNet model.

Evaluation Setup

All the evaluations are run on NTNU's HPC cluster Idun[3], using NVIDIA V100 GPUs.

For the PolyGen model, the official implementation available on Github[4] is used, with its corresponding pre-trained weights from training on the ShapeNet dataset. The model is run on a single NDIVIA V100 GPU. The generation code is taken from the Colab example[5], using the default parameters.

Since the official DreamFusion implementation is not publically available, an alternative implementation using Stable Diffusion (Rombach et al., 2022) as the text-to-image model is used, which is available on Github[6]. This implementation is stated to produce slightly worse results than the official DreamFusion implementation, but since Imagen is not publicly available it is the only accessible implementation. The generation is performed by training the model on the prompts "a chair" and "a table" respectively, and running each instance on a separate NVIDIA V100 GPU.

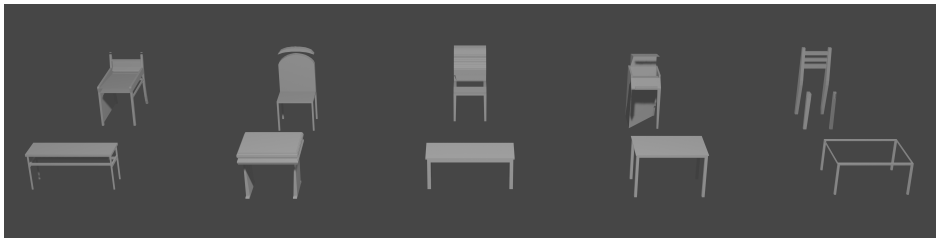
The official implementation of the GET3D model[7] is used for object generation. The pre-trained model weights for the ShapeNet chair and table categories are used and run on a single NVIDIA V100 GPU.

Evaluation Results

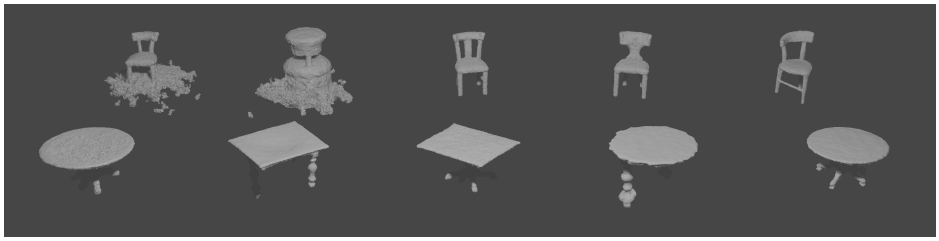
Figure 4.4 displays 5 of the generated objects from each category using the three models. For continuity, and since textures are not relevant for the task of 3D object recognition, the textures have been removed from the DreamFusion and GET3D generated objects.

As can be seen from Figure 4.4a, the PolyGen model is able to generate chair and table objects successfully in many cases. However, some of the objects lack essential parts, such as chairs missing the seat and tables missing the tabletop. Additionally, some of the objects have extra parts and incontinuities. Nonetheless, due to the mesh representation used by

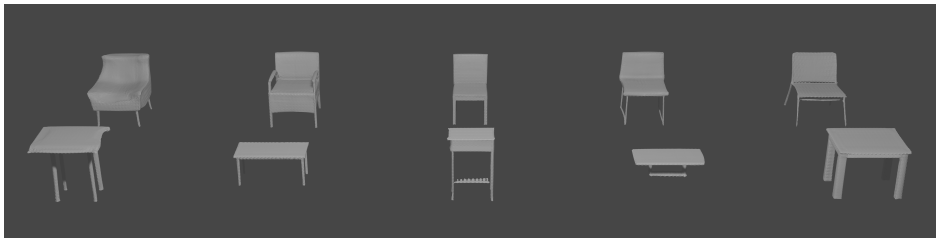
[3] https://www.hpc.ntnu.no/idun/
[4] https://github.com/deepmind/deepmind-research/tree/master/polygen
[5] https://colab.research.google.com/github/deepmind/deepmind-research/blob/master/polygen/sample-pretrained.ipynb
[6] https://github.com/ashawkey/stable-dreamfusion
[7] https://github.com/nv-tlabs/GET3D



(a) PolyGen generated chairs and tables



(b) DreamFusion generated chairs and tables



(c) GET3D generated chairs and tables

Figure 4.4.: Generation results from the three models.

PolyGen the objects have a low polygon count and model hard surfaces well.

The DreamFusion generated results shown in Figure 4.4b show that the model is able to generate chair and table objects successfully in most instances. However, in some cases there is noise contained within the mesh in addition to the object. Additionally, there are cases where the model is not able to generate chair objects successfully, and cases where the orientation of the various parts are mis-matched or legs are missing.

Figure 4.4c demonstrates that the GET3D model is able to successfully generate chair and table objects. The generated objects are coherent and

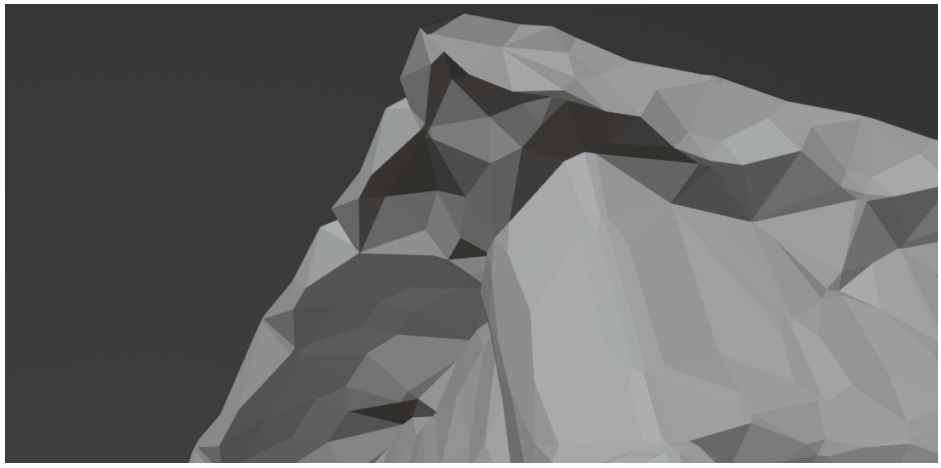


Figure 4.5.: Jagged edges on the GET3D generated objects.

do not have any evident artifacts. It is also able to model thin structures accurately, as can be seen from the chair legs in particular. However, by closely examining the meshes it is apparent that some of the edges and flat surfaces are jagged and and a triangular pattern is visible, as displayed in Figure 4.5. This is likely due to GET3D's use of a tetrahedral grid and *marching tetrahedra* for mesh extraction, and does not particularly lower the mesh quality.

Model	Classification Percentage	Average Percentage
GET3D Chair	53%	72%
GET3D Table	91%	
PolyGen Chair	95%	55%
PolyGen Table	15%	
DreamFusion Chair	30%	65%
DreamFusion Table	100%	
ShapeNet Chair	46%	62%
ShapeNet Table	78%	

Table 4.1.: PointNet classification results of the three models in comparison to the ShapeNet baseline.

The results of the PointNet classification are displayed in Table 4.1. For the chair category, both the "chair" and "stool" categories were accepted as correct identification, as they represent nearly identical objects. The "stool" category is also only present in the ModelNet40 dataset and not ShapeNet. Note that the DreamFusion percentage is only based on 10 objects for each category, while the GET3D and PolyGen percentage is based on 100 objects. As such, the classification percentage achieved by DreamFusion may not be regarded as equally accurate. Since PolyGen and ShapeNet output n-gon meshes, they are first triangulated using a blender script in order to convert them to point clouds. As can be seen from the results, the classification percentage varies greatly between the models and categories, with only the GET3D model achieving above 50% for both categories. GET3D achieved the highest average classification percentage on the two categories, even outperforming the ShapeNet baseline used to train the model.

Based on the evaluation results, GET3D performs the best out of all the three models. While PolyGen achieves a high percentage on the chair category and can be conditioned on image data, it is trained on 3D data in the form of ShapeNet. As such, though using real-world data to condition the model is possible, the sampled objects will still be from the ShapeNet trained distribution and the benefits of conditioning using real-world data is thus limited. While the GET3D models evaluated are trained on the ShapeNet dataset, it uses rendered images and can thus be trained using real-world data. DreamFusion achieves a high percentage on the table category, however the inference time is substantial and thus impractical for the task of generating a large amount of objects. Accordingly, due to its evaluation performance in addition to being trained on image data, GET3D is deemed the leading option for generating a large amount of 3D objects from real-world image data.

Additionally, while the GET3D paper does not present results from training on real-world data, the appendix demonstrates an experiment conducted using GANverse3D (Zhang et al., 2021) generated multi-view images of cars that resemble real images. GANverse3D is a GAN-based model that can generate realistic multi-view images of specific categories. Since GANverse3D does not output segmentation masks and camera poses, these are generated algorithmically and are are thus imperfect. Still, the

generated results when trained on this dataset are promising, and show the potential for use of the model with real-world data.

4.2. Datasets

In order to generate 3D objects from image data, a large, labeled dataset of images is required. To infer 3D information from the images, having multiple images of the same object from various viewing angles is highly advantageous. The GET3D model also requires the following input:

- Multi-view RGB images of object from several angles
- Camera extrinsic information for each image
- Segmentation mask for each image

As such, two multi-view image datasets are chosen as the basis for deep 3D procedural generation with real-world data. The datasets presented are the largest multi-view image datasets currently available, based on thorough search. They are still smaller with regards to number of unique objects when compared to the 3D datasets ShapeNet and ModelNet. However, they are deemed adequate for evaluating the potential of using real-world data with a deep 3D generative model. This is due to the number of objects in the more populated categories of the two datasets being similar to several of the ShapeNet and ModelNet categories.

4.2.1. Common Objects in 3D

The Common Objects in 3D (CO3D) (Reizenstein et al., 2021) dataset is a large-scale, multi-view image dataset of objects from 50 different categories. It contains in total 1.5 million multi-view images of nearly 19,000 different objects. The images are acquired by crowd-sourcing the task to Amazon Mechanical Turk and having the participants place an object on a solid surface and capture a video of it while circling around it. Each image is annotated with a segmentation mask generated by the PointRend (Kirillov et al., 2020) image segmentation network in addition to camera extrinsic information generated by the Structure-from-Motion framework COLMAP (Schönberger and Frahm, 2016). Additionally, the

dataset provides COLMAP generated 3D point clouds for some of the objects, where poor reconstructions are filtered out. Each image set is stated to cover a full 360 degree range of the objects.

The distribution of the objects' categories are shown in Figure 4.6. As can be seen from the category distribution, several of the categories contain at least 600 objects with accurate camera annotations. However, some of the categories contain only a small number of samples, making them unsuitable for training a deep model on single categories.

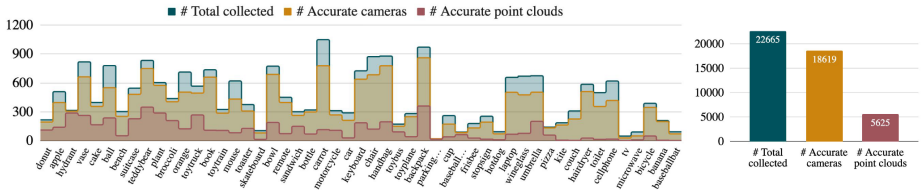


Figure 4.6.: The CO3D category distribution. Figure adapted from Reizenstein et al. (2021).

When compared to ShapeNet, it is evident that CO3D is a smaller dataset with fewer category samples on average. However, the CO3D dataset is deemed as well suited for evaluating the potential of using real-world data for deep 3D procedural generation by using the categories with the highest amount of samples, including the chair and book categories. GET3D is demonstrated to be able to generate high-quality objects when trained on a minimal set of 337 unique objects, indicating that the size of CO3D is still adequate for use with the model. Additionally, real-world image data is generally more uncomplicated to acquire than 3D data, meaning future methods could acquire real-world image data to a greater degree.

4.2.2. Objectron

Objectron (Ahmadyan et al., 2020) is a large-scale dataset containing multi-view images of objects from 9 unique categories. The paper states that the dataset contains in total 4 million images of 14,819 different objects, however the official release on Github[8] only contains 14,588 objects, of

[8] https://github.com/google-research-datasets/Objectron

which the category distribution is displayed in Table 4.2. As can be seen, the majority of categories contain around 2000 instances which is significantly higher than that of CO3D, though with less categories in total. The Objectron images are annotated with camera poses, manually annotated object poses through a 3D bounding box, in addition to a reconstructed point cloud. However, unlike CO3D, segmentation masks are not provided for the images. Additionally, the image sets do not cover a full 360 degree range of the objects in all cases.

category	objects
bike	476
book	2024
bottle	1928
camera	815
cereal box	1609
chair	1943
cup	2204
laptop	1473
shoe	2116

Table 4.2.: Objectron object category distribution

Though Objectron provides less categories when compared to CO3D, the larger number of objects per category makes it superior for evaluating deep 3D procedural generation on single categories.

4.3. Data preparation

In order for the image data from the datasets to be used to train the GET3D model, it first needs to be pre-processed into a suitable format. GET3D requires square RGBA images at a power of 2 resolution where the alpha channel is the segmentation mask containing the object. Additionally, for each image the extrinsic camera information must be provided in the form of an *elevation* angle and a *rotation* angle as illustrated in figure 4.7. Since the data format of the two datasets is different, they require individual pre-processing steps to make them suitable for use with GET3D.

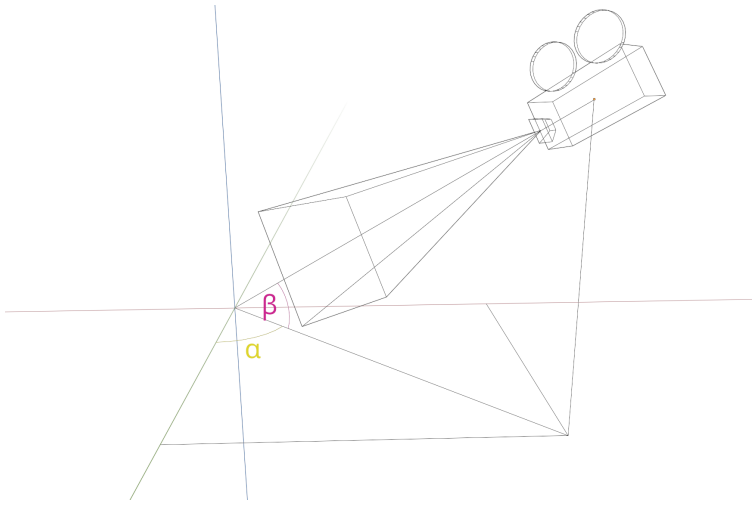


Figure 4.7.: The camera *rotation* and *elevation* angles. Here α corresponds to the *rotation* and β to the *elevation*.

This section covers the steps taken in order to pre-process the datasets to be used to train the GET3D model.

4.3.1. CO3D

The CO3D dataset contains segmentation masks generated by PointRend (Kirillov et al., 2020) that can be used as the alpha channel mask of the RGBA image to be used with GET3D. However, the segmentation masks provided are given in 8-bit grayscale where each pixel value corresponds to the probability that the pixel contains the object (where 0 corresponds to 0% and 255 corresponds to 100%). As such, the segmentation masks must be binarized in order to be used as the alpha channel. This is done using the Otsu's global binary threshold function available in OpenCV. The Otsu's binary threshold algorithm determines a global threshold value in the interval $[0, 255]$ by separating the image histogram into two clusters. This threshold is then applied to the original image, such that pixels with values above the threshold are included and pixels below excludes, producing a binary image. A global threshold is deemed sufficient since the masks contain no lighting information and an adaptive method would

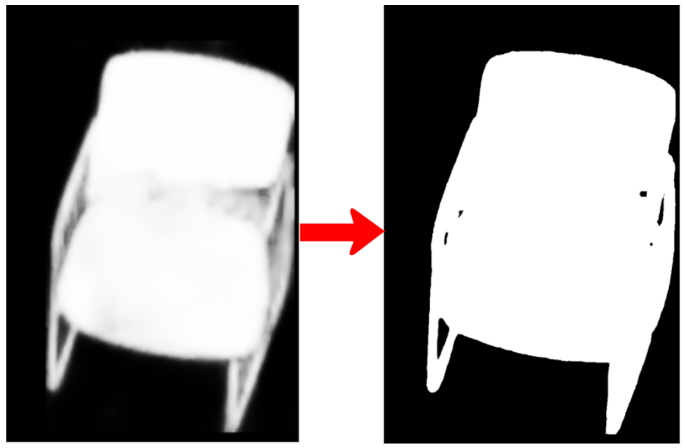


Figure 4.8.: The binarization process using Otsu's global binary threshold.

be excessive. Figure 4.8 illustrates the binarization process.

While CO3D provides camera annotations for each image, the cameras are not aligned with gravity. As such, the absolute camera elevation and rotation angles cannot be found relative to gravity, only relative to the other images. To overcome this issue, COLMAP is re-run on all the objects within a category with Manhattan world alignment enabled, which estimates the gravity direction and main horizontal axis using vanishing point detection[9]. The gravity alignment is illustrated in Figure 4.9. While this ensures the elevation angles are correct (assuming the objects are always oriented in the same direction vertically), the rotation angles are not relative to the object's rotation. As such, even from the same camera world rotation angle the object may be viewed from different angles as its world rotation is unknown. This is illustrated in figure 4.10. It is not possible to find the camera orientation relative to the object orientation without knowing the object orientation. As such, the resulting camera rotations after pre-processing of the CO3D dataset are globally incorrect, which may influence the results after training the GET3D model using it.

[9] https://colmap.github.io/faq.html#manhattan-world-alignment

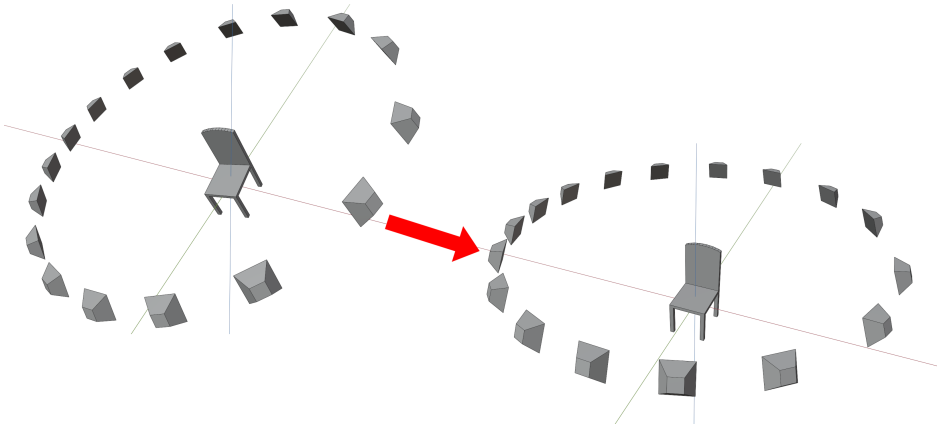


Figure 4.9.: Object and cameras before and after gravity alignment.

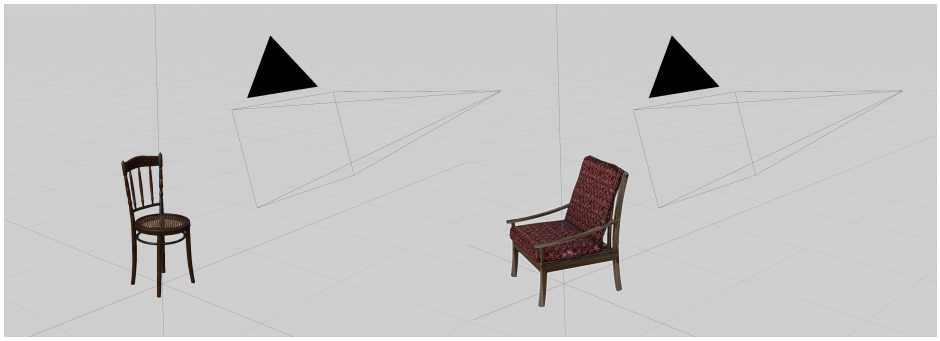


Figure 4.10.: Here it is illustrated that even for the same camera angles, the rotation may still be incorrect due to the orientation of the object.

4.3.2. Objectron

In addition to the camera extrinsic information, Objectron also provides a manually annotated 3D bounding box for each object which gives the translation and orientation of the object. This means that, contrary to CO3D, the camera rotation can be found relative to the object's rotation such that the same rotation angle always will display the object from the same view. Since Objectron provides the object orientation in view space, it can be used directly to find the camera rotation and elevation. The assumption used here is that the camera is always directed at the center of the object, which is approximately the truth for most object images.

The Objectron dataset does not provide segmentation masks for its objects. As such, these have to be generated for each image. This is done using PointRend (Kirillov et al., 2020), the same image segmentation neural network as used by CO3D. However, in some cases PointRend is not able to find the correct segmentation mask for the object. In certain instances no segmentation mask or an incorrect one is identified, and in other cases the wrong category is identified. This is especially evident for the book category since the books commonly contain cover images containing other objects.

Since a total of 100 images are used for each object, the instances where the wrong category is identified are still included. The reasoning behind this is that among a total of 100 images, small discrepancies in the segmentation masks are tolerated. Additionally, in some cases the correct segmentation mask is found but classified as the wrong category. Excluding wrongly classified masks could thus mean excluding correct ones, as it would not be possible to detect correct masks that are classified incorrectly without manual inspection. However, for the instances where no segmentation mask is identified all the images of that category are excluded. Some examples of poor and incorrect segmentation masks from the book category are displayed in Figure 4.11.

Additionally, all the Objectron images are given in a resolution of 480x640 while GET3D expects a square power of 2 resolution. Therefore the images are cropped to 480x480 before being up-sampled to 512x512. Since most objects are contained within the 480x480 cropped image, this is deemed adequate. The up-sampling is performed using the `resize()` function available in OpenCV.

Figure 4.11.: Examples of poor segmentation masks and wrongly identified objects.

4.4. Training GET3D

To train the GET3D model, 100 images are sampled for each object from each dataset category before pre-processing. This is the same number of images per object as used in the GET3D paper on the motorbike category, which contained few object instances. The sampling is done uniformly in order to obtain images with camera angles covering the whole object. For each of the dataset categories, the pre-processed images are split into 3 different splits; 80% training, 10% evaluation and 10% testing.

5. Experiments

The third and final research question of this thesis is as follows:

Research question 3 *To which degree are current deep 3D procedural models able to use real-world data for training?*

This chapter explains the experiments that were carried out and the procedure followed to produce the results, with the aim of answering the research question. All the code is available on Github[1] such that the results can be reproduced.

5.1. Experimental Plan

The objective of the experiments is to evaluate the thesis goal by evaluating the efficiency of current deep learning techniques in generating 3D objects using real-world data. As such, the state-of-the-art neural network model GET3D is used and trained on the two real-word, multi-view, image datasets CO3D and Objectron. The generated results are evaluated using 3D classification with the PointNet model, FID scores, in addition to a user study.

For the experiments, two distinct object categories from the CO3D and Objectron datasets are used to train the GET3D model, namely the book and chair categories. This is due to both datasets containing a large number of instances from these two categories. Additionally, the book category can easily be evaluated due to the simple cuboid geometry of books. The chair category from the GET3D ShapeNet trained model is already evaluated, and can act as a baseline for comparison with chair objects generated using a GET3D model trained on Objectron and CO3D respectively. Finally, the loss function of the GET3D model can be assessed after training to

[1] https://github.com/Maro1/master_thesis

obtain an indication of the model performance on the datasets, and act as a suggestion for changing the network hyperparameters.

5.2. Experiment evaluation

In order to evaluate the quality of the generated 3D objects, several evaluation methods are used. The 3D classification network PointNet (Qi et al., 2016) is used to classify the objects in addition to FID scores and a user study. The three evaluations provide different evaluation angles, where FID scores are mainly an indication of model performance, the 3D classification acts as an quantitative evaluation of the generated results, and the user study provides a human perspective on the generated results.

5.2.1. Fréchet Inception Distance

Fréchet inception distance (FID) is an evaluation metric used to assess the quality of generated images by comparing the generated image distribution with the ground-truth test set image distribution. It is commonly used for GAN-based models, and is the main evaluation metric used in the GET3D paper, where 50,000 images are rendered of generated objects and used for evaluation. A lower score indicates better performance. While the metric is useful and gives an indication of the quality of the objects, it also takes texture into account since it uses rendered images of textured objects. The metric is still included in the evaluation as it gives an indication of the performance of the model.

5.2.2. PointNet 3D classification

The evaluation using PointNet is performed by generating 100 obj files using the model weights of the trained model for the chair category of each dataset. PointNet is trained on the ModelNet40 dataset, and since this dataset does not contain the book category, it was omitted from this evaluation. This evaluation does not take texture into account, and is as explained in section 4.1.2 deemed an appropriate metric for evaluating the quality of the generated objects.

5.2.3. User study

In addition to the aforementioned evaluations, a user study is conducted in order to obtain an additional evaluation of the generated objects. By conducting a user study, a human evaluation can be obtained that acts as a counterpart to the quantitative evaluations. The user study is performed by capturing rendered images of 50 objects from each model and asking the participants to select the images that resemble their respective categories. Since neither ModelNet40 nor ShapeNet have a book category, the book objects are only generated from the Objectron and CO3D datasets as shown in Table 5.1.

Dataset	Categories
Objectron	Book, Chair
CO3D	Book, Chair
ShapeNet (Baseline)	Chair

Table 5.1.: The datasets and categories used for the user study

The survey is created as 10 different image-picker questions with 5 questions for each category, where each question contains 10 images from each model. The participants are asked to identify which images resemble their respective categories. Each question contains an equal number of images from each dataset. In total, 16 people participated in the user study.

5.3. Experimental Setup

All the experiments are run on NTNU's HPC cluster Idun[2]. The hardware and software used differs between the various experiment stages.

5.3.1. CO3D Pre-Processing

Re-running COLMAP on the CO3D images is done by first building the application from source using the Python build script, before running it as

[2] https://www.hpc.ntnu.no/idun/

a slurm array job using a single V100 GPU for each job. The following commands are used in order to run COLMAP with world alignment:

```
$ colmap automatic_reconstructor --quality low ...
$ colmap model_orientation_aligner ...
$ colmap model_converter --output_type TXT ...
```

Using a low quality setting for the `automatic_reconstructor` for performance reasons.

After running COLMAP, the remaining pre-processing of the CO3D dataset is performed using the Python script running as an array job, with each instance on a single CPU core.

5.3.2. Objectron Pre-Processing

The Objectron pre-processing stage is performed using a Python script running on a single CPU core with a single NVIDIA A100 80GB GPU. The Python script runs a pre-trained PointRend model for each image to obtain the segmentation masks. The pre-trained model is the COCO R50-FPN 3x available from the PointRend Github[3].

5.3.3. GET3D

Each GET3D model is trained on 4-8 NVIDIA V100 GPUs for 2048 iterations, taking a duration of 24-48 hours. For each dataset object, a total of 100 images are used. The official GET3D code was modified to enable the training on datasets not present in the GET3D paper. The following command is used to train each each model, using an image resolution of 512x512 and a gamma value of 40:

```
$ python train_3d.py --outdir=<log dir> --data=<
    image path> --camera_path <camera path> --gpus=8
     --batch=32 --gamma=40 --manifest_dir <manifest
    dir> --dmtet_scale 1.0 --one_3d_generator 1 --
    fp32 0 --img_res 512
```

[3] https://github.com/facebookresearch/detectron2/tree/main/projects/PointRend#pretrained-models

Additional experiments are conducted using different gamma values and are presented in section 5.4.5. Before each training the data is split into 3 splits: 10% validation, 10% test and 80% training.

5.4. Experimental Results

This section presents the results obtained for the real-world image data experiments. Additional results are available in Appendix B. After running the pre-processing steps, several of the dataset objects were excluded due to poor segmentation masks or poor COLMAP reconstructions. This is especially evident for the Objectron objects, as PointRend was not able to identify segmentation masks in many cases, and they were thus excluded. As such, even though the datasets originally contained a similar amount of objects for each of the categories, the resulting objects after pre-processing are significantly lower for Objectron. The number of objects per dataset category used for model training compared to the total present in the dataset is displayed in Table 5.2.

Dataset Category	# Objects in Dataset	# Objects Used
Objectron Book	2024	462
Objectron Chair	1943	870
CO3D Book	2299	2258
CO3D Chair	1475	1412

Table 5.2.: The number of dataset objects per category used for training the GET3D model after pre-processing.

5.4.1. Book generation

The first experiments generate objects in the book category, as books have a simple cuboid geometry that can be easily evaluated. Figure 5.1 displays the results after generating 5 objects using the CO3D-trained model. As can be seen, the generated objects somewhat diverge from a cuboid shape. For a cuboid book shape, the book cover should be parallel with the back of the book. While some of the generated objects have approximately parallel surfaces, most of the objects diverge from this. Additionally, books

typically have an indentation between the book cover and pages, a feature that is not present in the generated objects.

Figure 5.1.: Generated book objects using CO3D trained model

Figure 5.2.: Results after training on Objectron book category

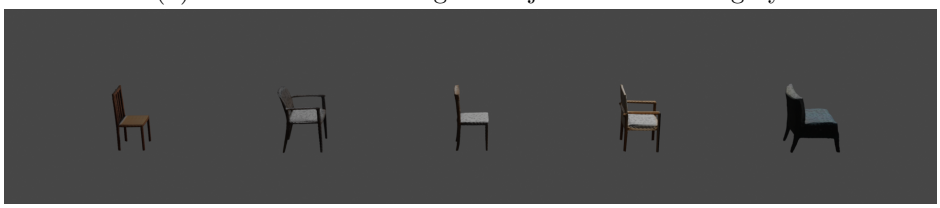
The results after generating 5 objects using the Objectron-trained model are displayed in Figure 5.2. As can be seen, some of the objects have a coherent shape, while others are amorphous and scattered, containing mostly empty space. This is likely due to the segmentation masks for the Objectron dataset often being incomplete, something further reinforced by the CO3D generated objects not having the same issue. While some of the Objectron objects are coherent, they still somewhat diverge from the cuboid shape associated with books. The long-side object surfaces are not flat, resulting in the generated objects resembling cylinders to a greater extent.

5.4.2. Chair generation

Figure 5.3 displays the results after generating 5 objects of the chair category using the CO3D and Objectron models. The pre-trained ShapeNet baseline is included for ease of comparison. As is evident from the results, the CO3D and Objectron generated objects are inferior to the ShapeNet

(a) Results after training on CO3D chair category

(b) Results after training on Objectron chair category



(c) Results using pre-trained model on the ShapeNet chair category

Figure 5.3.: Generation results using the different models on the chair category

baseline in terms of quality and diversity. While the Objectron and CO3D generated objects for the most part contain features that can be regarded as resembling chair legs, they are not refined and in most cases several legs are missing. The chair seats, backrests, and potential armrests are not evidently visible. Similarly to the book category, the Objectron generated objects contain samples consisting mostly of empty space with no clear coherence.

For this category, PointNet was used to predict the classes of the generated results. As a baseline, the ShapeNet chair pre-trained model from section 4.1.2 is used. The results of the PointNet classification are shown in Table 5.3. As can be seen from the classification results, the Objectron and CO3D models perform substantially worse than the ShapeNet baseline

with 14% and 0% prediction percentage respectively on the chair and stool categories, compared to the pre-trained ShapeNet baseline which achieved 56%. However, the Objectron percentage of 14% indicates that the generated objects have some resemblance to chairs. Looking at the most predicted class for each model, only the ShapeNet baseline has the chair and stool category. The most predicted class for the CO3D and Objectron models is the plant category.

PointNet Classification			
Model	Class	Percentage	Chair/Stool Percentage
CO3D	cup	2%	0%
	curtain	1%	
	door	1%	
	monitor	1%	
	person	31%	
	piano	2%	
	plant	56%	
	radio	1%	
	vase	2%	
	xbox	3%	
Objectron	bench	6%	14%
	person	14%	
	plant	35%	
	stool	14%	
	table	24%	
	vase	7%	
ShapeNet			56%

Table 5.3.: PointNet classification results

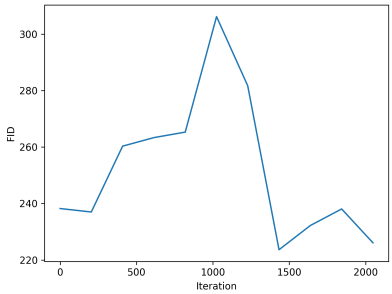
5.4.3. FID scores

GET3D outputs a `fid50k` score every 204-205 iterations, of which the final score at iteration 2048 is recorded and displayed in Table 5.4. The ShapeNet chair trained model from the GET3D paper is used as a baseline. As can be seen from the results, the real-world data models perform substantially

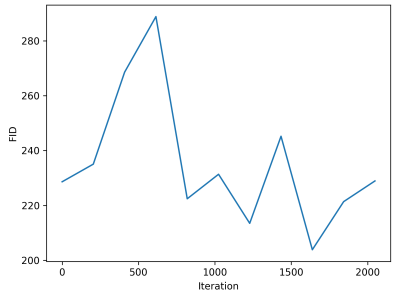
worse than the ShapeNet baseline. The real-world data trained model that achieves the best score is the Objectron chair model.

Model	FID Score
CO3D Book	226.09
CO3D Chair	228.94
Objectron Book	221.37
Objectron Chair	182.90
ShapeNet Chair (From paper)	**22.41**

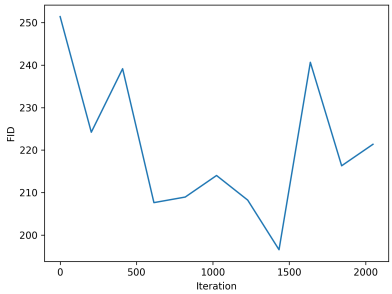
Table 5.4.: FID scores



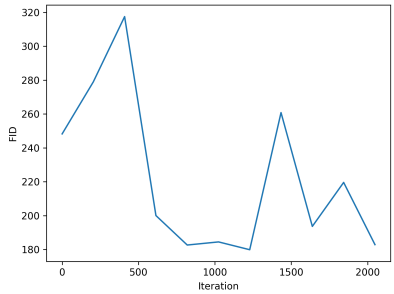
(a) CO3D book



(b) CO3D Chair



(c) Objectron Book



(d) Objectron Chair

Figure 5.4.: FID scores recorded every 204-205 iterations

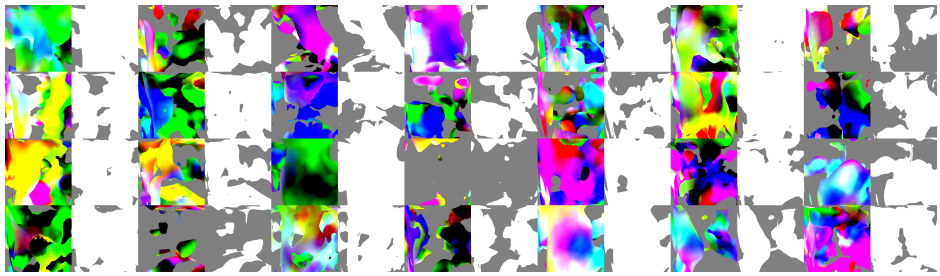


Figure 5.5.: Generated results after iteration 0 with randomly initialized weights on the CO3D book category. Both the rendered object and the segmentation mask is included.

Additionally, the FID scores recorded over the complete training period are displayed in Figure 5.4. It is observable that there is a great variance in the achieved FID scores, and that the scores do not seem to converge. In some cases, particularly for the CO3D chair model, the final FID score does not diverge heavily from the initial FID score. Additionally, for each model except the Objectron book model, a higher score than the initial is achieved over the duration of the training period. This can indicate that the models do not improve significantly over time. However, by observing the generated CO3D book objects of iteration 0 displayed in Figure 5.5, where the network weights are randomly initialized, it is evident that the quality is inferior to those of the final generated objects. As such, it can be concluded that the FID metric does not predict the sample quality particularly well in the case of the models used.

5.4.4. User study

The results of the user study are displayed in Table 5.5. Here the identification percentage corresponds to the average percentage of objects correctly identified for each dataset and category among all participants. The results clearly indicate that the ShapeNet chair generated objects are identified in substantially more cases than those generated from real-world data. The ShapeNet generated objects are identified as chairs in nearly all the cases at 95% while the real-world data generated objects are identified correctly in fewer cases, with the Objectron generated books having the highest

Category	Identification Percentage
CO3D Book	0.24%
Objectron Book	9.5%
CO3D Chair	0%
Objectron Chair	0.12%
ShapeNet Chair	**99.5%**

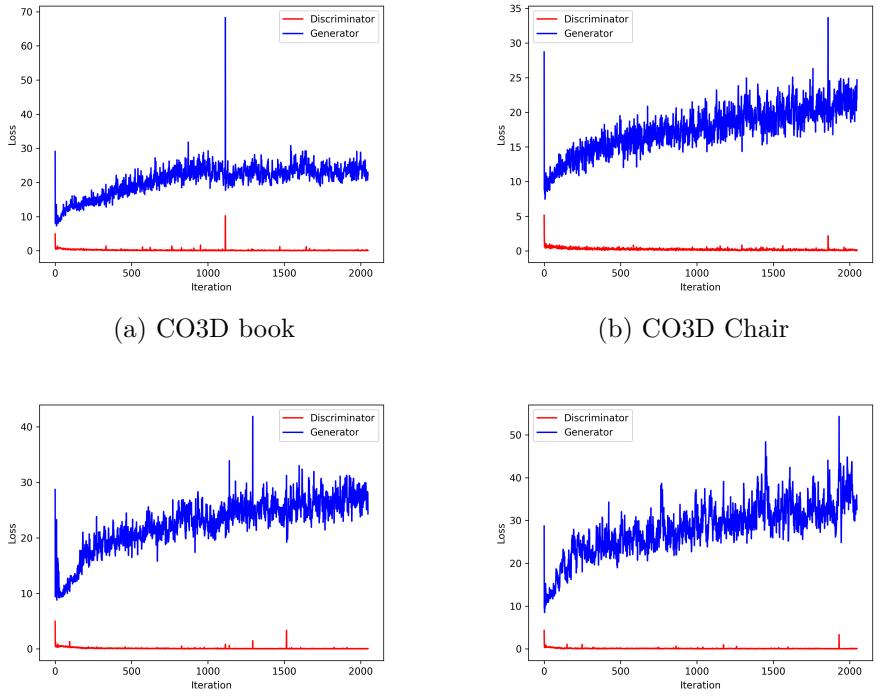
Table 5.5.: The results of the user study on each dataset and catagory

percentage at 9.5%. The chair identifications somewhat correlate with the classification presented in section 5.4.2, in that the ShapeNet model clearly outperforms the real-world data ones. However, the Objectron chair classification achieved of 14% is not reflected in the user study results.

5.4.5. Training Loss and Gamma Parameter

The generator and discriminator losses for each model over the duration of the training are displayed in Figure 5.6. These are the `Loss/D/loss` and `Loss/G/loss` mean values output by the GET3D model each iteration. Note that since the GET3D model contains two discriminators; one for the RGB image and one for the segmentation mask, the resulting losses are calculated from a combination of these. As can be seen, there is a large discrepancy between the generator and discriminator losses for each model. While the discriminator loss slowly decreases, the generator loss is increasing for each iteration. Since the discriminator is approaching optimality, the generator loses the ability to "fool" the discriminator. Despite the observed increase in the generator loss, this does not immediately entail that the generated objects are decreasing in quality. Rather, the loss increase can signify the discriminators increasing ability to separate real and fake images. This is however not optimal, and is a common GAN failure mode.

To overcome the issue of the discriminator outperforming the generator, additional experiments were conducted by changing the gamma parameter value. By increasing this value, further regularization is added to the discriminator with the aim of decreasing the rate of which it is able to correctly discriminate. These experiments were conducted on the Objectron chair category, using gamma values of 400, 1600, and 3200. The generator



(a) CO3D book

(b) CO3D Chair

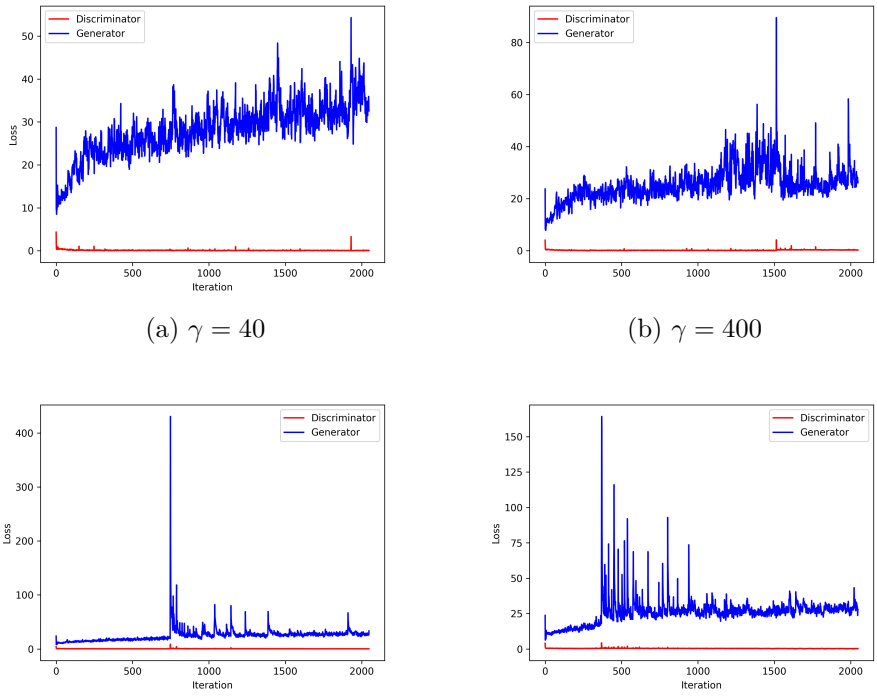
(c) Objectron Book

(d) Objectron Chair

Figure 5.6.: Discriminator and Generator loss each iteration

and discriminator losses are recorded for each experiment and illustrated in Figure 5.7.

As can be observed from the graphs, increasing the gamma value changes the generator loss over time. For the gamma values of 1600 and 3200, the generator loss steadily increases for a period before starting to peak periodically. The generator loss peaks seem to correlate with the discriminator, as the discriminator loss also peaks at the same time. However, increasing the gamma value does not seem to improve the generator's ability to "fool" the discriminator. When observing the generated results at iteration 2048 for the increased gamma value models displayed in Figure 5.8, it is evident that they are inferior to those generated by the original model, with no visible resemblance to chairs. Additionally, while the generated textures

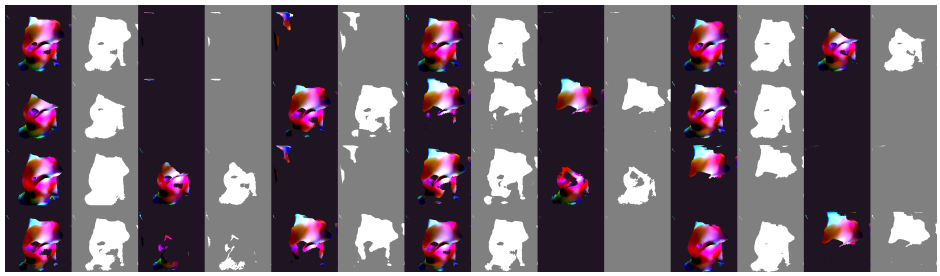


(a) $\gamma = 40$ (b) $\gamma = 400$

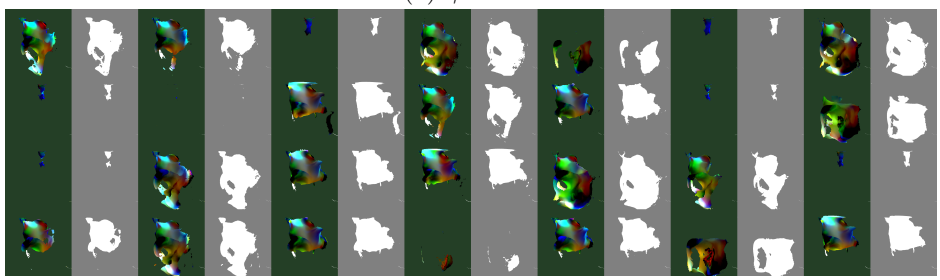
(c) $\gamma = 1600$ (d) $\gamma = 3200$

Figure 5.7.: Discriminator and Generator loss for different gamma values

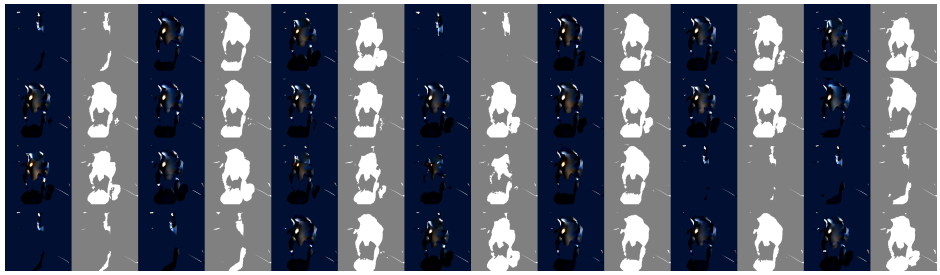
are not of importance to 3D object recognition algortihms, and accordingly not of particular interest, they give an indication of the performance of the model. As can be seen from the increased gamma parameter results, the object textures are colorful and incoherent, and seemingly randomly sampled like the randomly initialized model results from Figure 5.5. This is in stark contrast to the original model where the textures were more coherent and less colorful, which is in accordance with the expected texture of a chair. As such, it is evident that increasing the gamma parameter does not increase the model performance.



(a) $\gamma = 400$



(b) $\gamma = 1600$



(c) $\gamma = 3200$

Figure 5.8.: Generated results after 2048 iteration using different gamma parameters.

6. Discussion

The task of the GET3D generator can generally be considered more difficult than that of the two discriminators. While the discriminators have to determine the genuineness of real and fake samples, the generator needs to synthetically generate new samples that resemble real ones. Since the GET3D generator network is complex and includes several different components, including the convolutional layers, hybrid DMTet representation, and differential renderer, the required training for the generator can be expected to exceed that of the discriminator. However, even when penalizing the discriminator by increasing the gamma parameter, the generator is still unable to generalize well and produce acceptable samples.

There are several likely reasons as to why the generator is unable to synthesize 3D objects of a high quality, all of which are related to the datsets used to train the model. The resulting poor quality of the generated results are presumably due to a combination of these factors.

Intrinsic Parameters

Since the Objectron and CO3D dataset images are collected from many different contributors using phone cameras, the intrinsic parameters of the cameras vary between the object images. The GET3D model does not take intrinsic parameters as input for each sample, meaning the same intrinsic parameters are used by the differential renderer to render each image. When using synthetic data this is not an issue since the dataset images can all be rendered using the same intrinsic parameters, though for real-world data this is difficult to control without using the same camera to capture all images. A resolution to this issue could be to modify the GET3D model to take the intrinsic camera parameters as input for each sample, however this would add additional complexity to the network, in addition to adding further requirements to the dataset used. Though the intrinsic parameters may affect the generated results to some degree, it is

likely not a large contributor since the variance between images taken with different cameras is minor.

Extrinsic Parameters

Another potential contributor to the low-quality results, and an issue with using real-world data, is that the extrinsic parameters of the cameras used to capture the images are not coherent between different objects and within the same object. Since the images from the Objectron and CO3D datasets are captured by humans circling around the object, the camera will not always perfectly be directed at the center of the object. Additionally, the distance from the camera to the object will vary between different captures and between images of the same object. This is likely to affect the generator's ability to generate objects, since the differential renderer camera is always distanced a fixed amount from the object and directed at the center. Since the dataset images are captured from different distances, the resulting scale of the objects is incoherent. Additionally, there is a discrepancy between the center of the object in the dataset and network rendered images in the cases where the dataset images are not captured with the camera directed directly at the center of the object. This may cause object displacement, and is an issue that is difficult to overcome without capturing the images using a specific setup to maintain object distance and camera direction, which would add additional complexity to data acquisition. While the GET3D paper presents promising results when training on noisy cameras, the noise is only added to the camera elevation and rotation angles, while the images and segmentation masks remain identical. As such, the camera distance remains the same and always directed at the center.

Lighting

The lighting of the scene also affects the resulting images. When using synthetic data, the light parameters can be controlled explicitly and be kept equivalent across different objects, and the same parameters can be used by the differential renderer of the model. To achieve this with real-world data, a specific lighting setup must be created to allow the light to stay identical across different object captures, which is not the case for

the Objectron and CO3D datasets. However, the GET3D paper presents promising results when training on images with different lighting conditions in the case of the GANverse3D generated data, which indicates that this is not a large factor. Additionally, this should only affect the texture of the objects, since the segmentation masks remain identical regardless of the lighting conditions. As such, this is likely not a large contributor to the low quality of the generated results.

Segmentation Masks

A likely large contributor to the low quality of the generated results is the segmentation masks used. The segmentation masks for both datasets were generated algorithmically using the PointRend segmentation network. As such, the resulting segmentation masks are not perfect, and in many cases diverge considerably from the ground-truth. However, the GET3D paper also presents results when using segmentation masks generated by PointRend, using the rendered objects with an added background image, and show only a slight drop in quality. As such, the main factor in the case of the Objectron experiments is likely the failure of PointRend to classify the correct category for the segmentation masks. The reason behind the PointRend failure is unknown, and might be related to the Objectron dataset itself or the setup used to run the model. When observing some of the book segmentation masks generated by PointRend displayed in Figure 6.1, it is also clear that even when the correct category is identified, some of the masks are discontinuous and low quality. This however is only an issue related to the Objectron dataset, and should not affect the CO3D trained models to a large degree.

Dataset Differences

Though the number of CO3D objects used for model training is higher than that of Objectron due to the exclusion of Objectron objects with poor segmentation masks, the generated objects are not of a quantifiable higher quality. The Objectron generated objects perform better on all evaluation metrics except the user study for the chair category. However, since all the evaluation metrics show the models trained on both datasets perform poorly, the small discrepancy between them might not be a clear indication

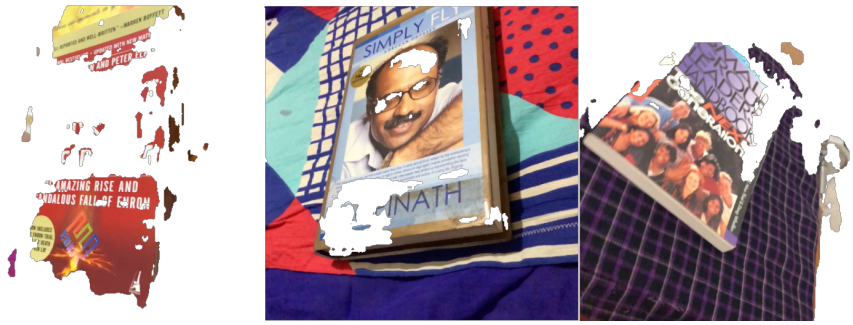


Figure 6.1.: Discontinous segmentation masks from Objectron book category

of variance in quality. Nonetheless, there might be different contributors to the poor quality results for each of the datasets. Common among them is the aforementioned issues related to the camera extrinsics, in addition to the likely smaller factors of camera intrinsics and lighting. The CO3D dataset also presents an additional likely contributing factor in the form of the rotation angles not being absolute as explained in 4.3.1. On the other hand Objectron presents issues related to the PointRend generated segmentation masks.

Dataset Size

The combination of the aforementioned factors in combination with the small size of the datasets is likely the main contributing factor to the low quality of the generated results. The GET3D paper presents promising results when training on few data samples, with the motorbike category used to train the model only containing 337 objects. However, in that case the model was trained on synthetic data containing accurate camera parameters and segmentation masks. When using inaccurate annotations with real-world data, the small size of the datasets likely contribute to the generator not being able to generalize well over the training data.

7. Conclusion and Future Work

This chapter concludes the thesis and provides suggestions for future work.

7.1. Conclusion

This thesis has explored the possibility of generating a novel dataset of 3D objects through deep 3D procedural generation using real-world data. Existing state-of-the-art models were tested and evaluated before arriving at a method for deep generation using two distinct real-world image datasets.

The first thesis research question was as follows:

Research question 1 *To which degree can current deep learning methods generate 3D objects of high quality?*

The state-of-the-art evaluation performed suggested that current methods can generate 3D objects of high quality when trained on 3D data. Most methods still have drawbacks such as artifacts, uneven surfaces and substantial generation time.

The second thesis research question considered the evaluation of 3D objects:

Research question 2 *How can the similarity of 3D objects to their real-world counterparts be objectively evaluated?*

This was achieved through the use of a 3D classification network that could predict the generated object classes. Since this evaluation acts on the generated objects alone, it can be used regardless of the data format used to train the models.

The final research question was aimed at current deep procedural 3D models' ability to use real-world data for training:

Research question 3 *To which degree are current deep 3D procedural models able to use real-world data for training?*

This was evaluated using two real-world image datasets with the GET3D model. The experiments conducted suggest that using real-world image data is not currently a viable approach and produces significantly worse results than synthetic data.

The end goal of the thesis was the following:

Goal *Explore the possibility of generating a large-scale dataset of 3D objects using a deep learning model trained on real-world data.*

The results obtained show that the method presented is not a viable approach for generating a novel 3D object dataset. While the model is able to generate coherent objects, they are of a low quality and do not resemble their respective categories to a satisfying degree. Issues related to camera extrinsics, segmentation masks for the Objectron dataset, and camera angles for the CO3D datasets are likely contributors to the low quality results. Alleviating the issues with regards to the camera extrinsic parameters would require a specialized acquisition setup, and might remove the advantage of real-world data being easier to acquire than manually creating 3D objects due to the additional complexity. Additionally, obtaining the objects' orientation as is done for the Objectron dataset requires manual annotation, which adds additional complexity.

7.2. Future Work

Future works may try and overcome the issues presented in this thesis. Current multi-view image datasets are not larger in scale than existing 3D datasets, though their acquisition is less time-consuming. By acquiring multi-view images at a truly large scale, the model might be able to generalize better regardless of the camera and segmentation mask annotations being imperfect. Alternatively, a specialized setup for acquiring the images may be attempted, using a camera rig and monochrome background to obtain correctly annotated images without requiring the use of algorithmic methods for acquiring the annotations.

Additionally, since the 3D object recognition task is independent of texture, experiments using depth images as the training data and a differential renderer producing depth images could be conducted. By narrowing the model to only generate geometry, improved results can be expected.

Bibliography

Adel Ahmadyan, Liangkai Zhang, Jianing Wei, Artsiom Ablavatski, and Matthias Grundmann. Objectron: A large scale dataset of object-centric videos in the wild with pose annotations, 2020.

Loren C. Carpenter. Computer rendering of fractal curves and surfaces. *SIGGRAPH Comput. Graph.*, 14(3):109, jul 1980. ISSN 0097-8930.

Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3d generative adversarial networks, 2022.

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects, 2022.

Jonas Freiknecht and Wolfgang Effelsberg. A survey on the procedural generation of virtual worlds. *Multimodal Technologies and Interaction*, 1 (4), 2017. ISSN 2414-4088.

Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. Get3d: A generative model of high quality 3d textured shapes learned from images, 2022.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

Agrim Gupta, Piotr Dollár, and Ross Girshick. Lvis: A dataset for large vocabulary instance segmentation, 2019.

Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. MeshCNN. *ACM Transactions on Graphics*, 38 (4):1–12, jul 2019.

Moritz Ibing, Gregor Kobsik, and Leif Kobbelt. Octree transformer: Autoregressive 3d shape generation on hierarchically structured sequences, 2021.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.

Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. Pointrend: Image segmentation as rendering, 2020.

Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering, 2020.

Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3d: High-resolution text-to-3d content creation, 2023.

Aristid Lindenmayer. Mathematical models for cellular interactions in development i. filaments with one-sided inputs. *Journal of Theoretical Biology*, 18(3):280–299, 1968. ISSN 0022-5193.

Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis, 2019.

William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, page 163–169, New York, NY, USA, 1987. Association for Computing Machinery. ISBN 0897912276.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics*, 41(4):1–15, jul 2022.

Charlie Nash, Yaroslav Ganin, S. M. Ali Eslami, and Peter W. Battaglia. Polygen: An autoregressive generative model of 3d meshes, 2020.

Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models, 2022a.

Alex Nichol, Heewoo Jun, Prafulla Dhariwal, Pamela Mishkin, and Mark Chen. Point-e: A system for generating 3d point clouds from complex prompts, 2022b.

Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. Texture fields: Learning texture representations in function space, 2019.

Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion, 2022.

Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *CoRR*, abs/1706.02413, 2017.

Shaohua Qi, Xin Ning, Guowei Yang, Liping Zhang, Peng Long, Weiwei Cai, and Weijun Li. Review of multi-view 3d object recognition methods based on deep learning. *Displays*, 69:102053, 2021. ISSN 0141-9382.

Jeremy Reizenstein, Roman Shapovalov, Philipp Henzler, Luca Sbordone, Patrick Labatut, and David Novotny. Common objects in 3d: Large-scale learning and evaluation of real-life 3d category reconstruction, 2021.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding, 2022.

Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-Motion Revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis, 2021.

Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis, 2021.

Dong Wook Shu, Sung Woo Park, and Junseok Kwon. 3d point cloud generative adversarial network based on tree structured graph convolutions, 2019.

Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition, 2015.

Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. Kpconv: Flexible and deformable convolution for point clouds, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

Hao Wang, Nadav Schor, Ruizhen Hu, Haibin Huang, Daniel Cohen-Or, and Hui Huang. Global-to-local generative model for 3d shapes. *ACM Trans. Graph.*, 37(6), dec 2018a. ISSN 0730-0301.

Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images, 2018b.

Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T Freeman, and Joshua B Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems*, pages 82–90, 2016.

Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds, 2020.

Zhirong Wu, Shuran Song, Aditya Khosla, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets for 2.5d object recognition and next-best-view prediction. *CoRR*, abs/1406.5670, 2014.

Haozhe Xie, Hongxun Yao, Xiaoshuai Sun, Shangchen Zhou, and Shengping Zhang. Pix2vox: Context-aware 3d reconstruction from single and multi-view images. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, oct 2019.

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

Appendices

A. Additional State-of-the-Art Evaluation Results

This appendix presents additional results for the PolyGen and GET3D models of the state-of-the-art evaluation. DreamFusion is excluded due to the long training time to produce results.

A.1. PolyGen

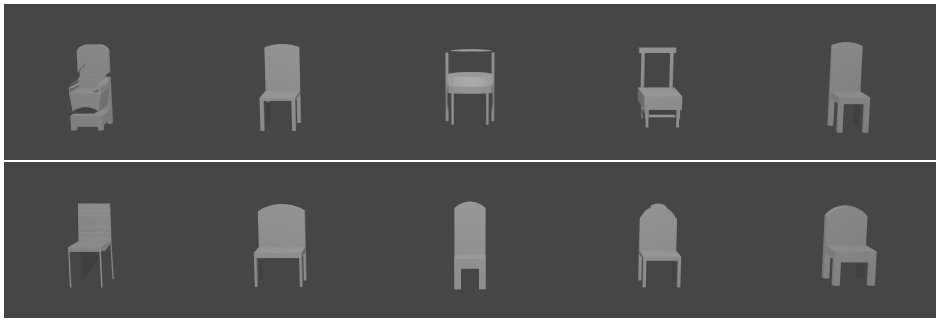


Figure A.1.: Chair Category

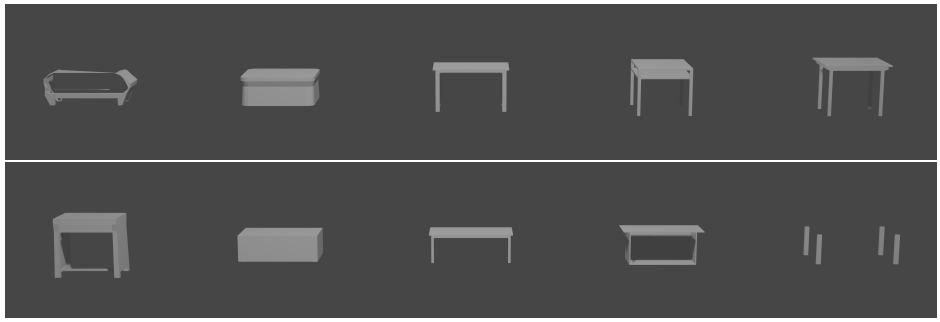


Figure A.2.: Table Category

A.2. GET3D

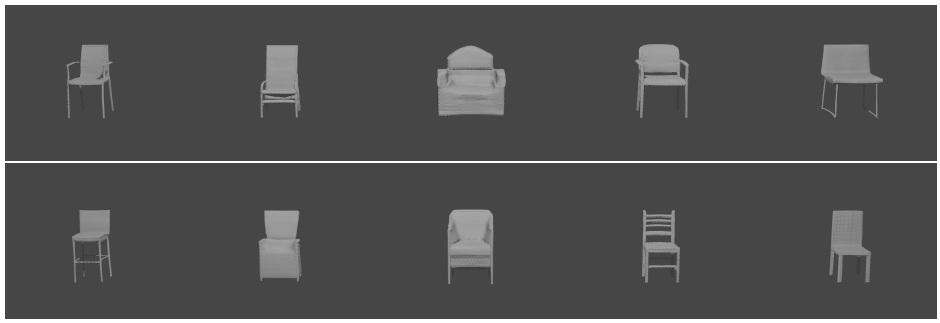


Figure A.3.: Chair Category

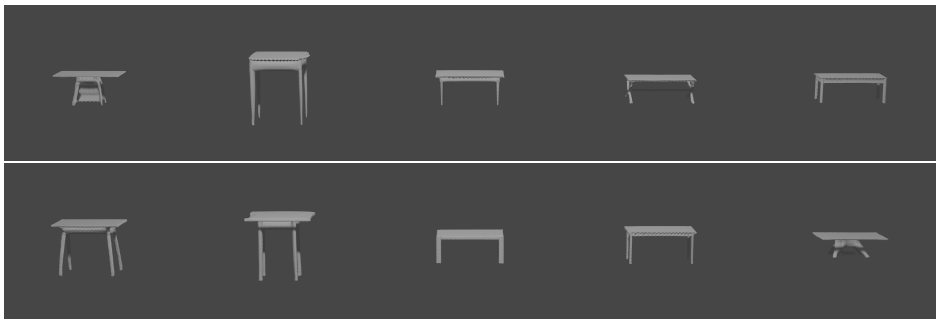
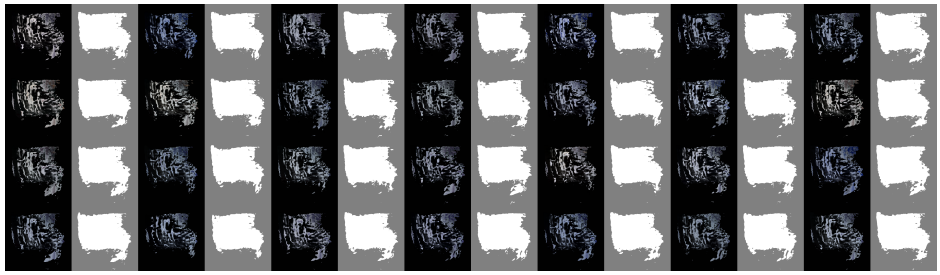


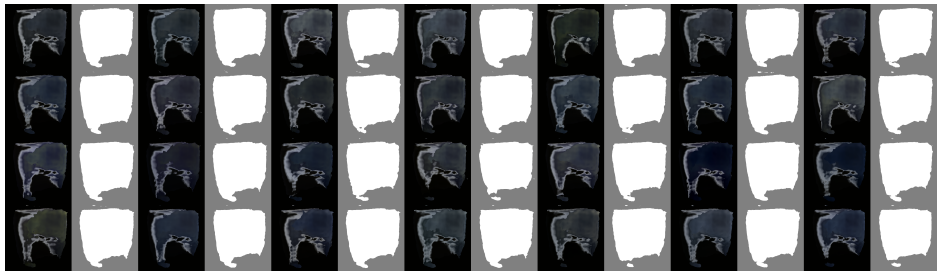
Figure A.4.: Table Category

B. Additional Real-World Data Results

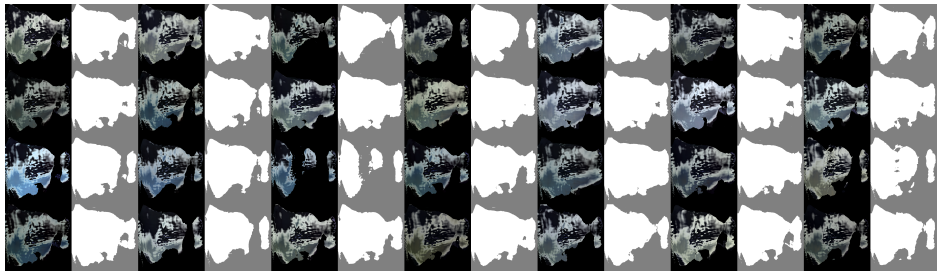
This appendix presents additional results for the real-world data experiments. For each model, the output generated by GET3D for each 512 iterations is provided, including the generated segmentation masks.



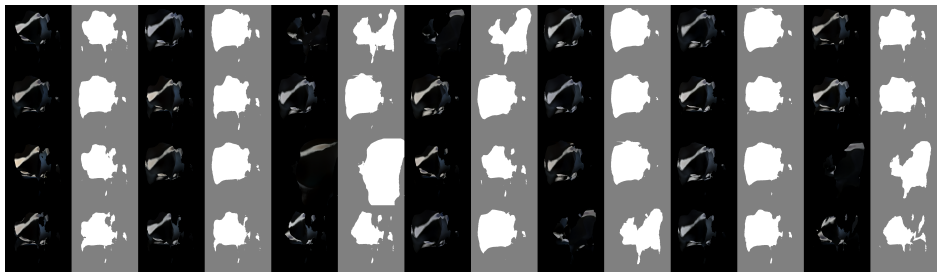
(a) Iteration 512



(b) Iteration 1024

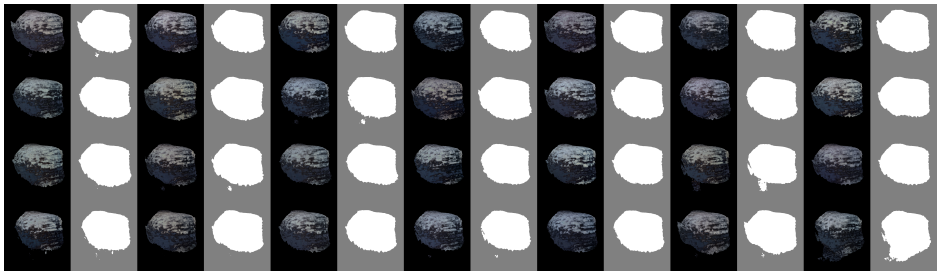


(c) Iteration 1536

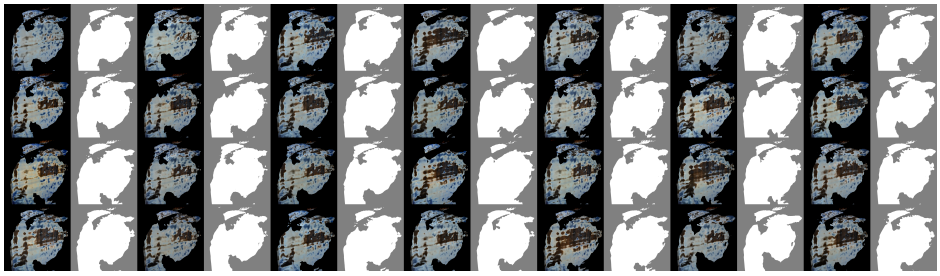


(d) Iteration 2048

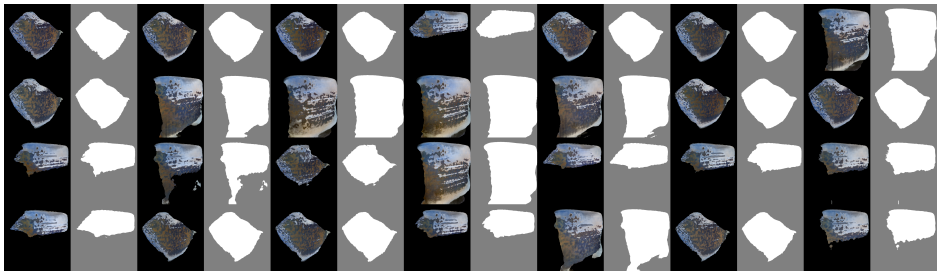
Figure B.1.: CO3D Chair



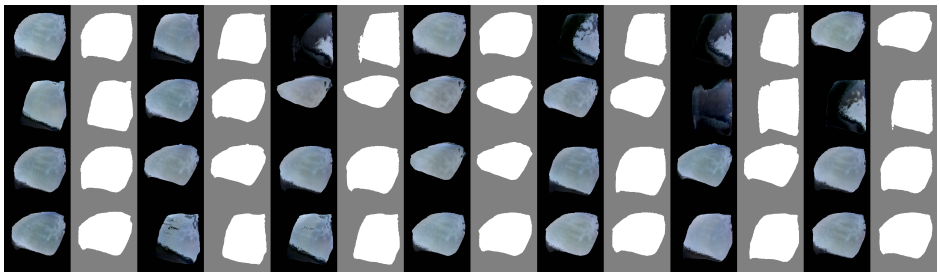
(a) Iteration 512



(b) Iteration 1024

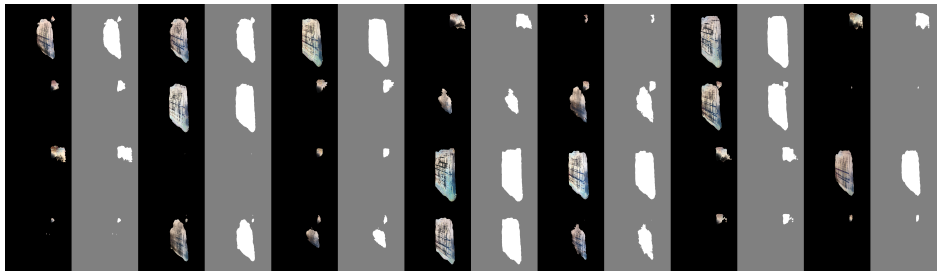


(c) Iteration 1536

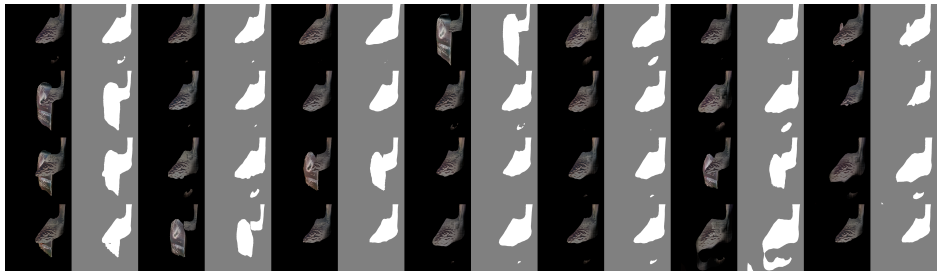


(d) Iteration 2048

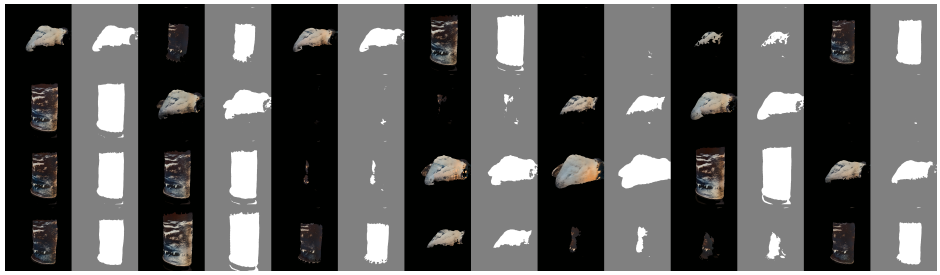
Figure B.2.: CO3D Book



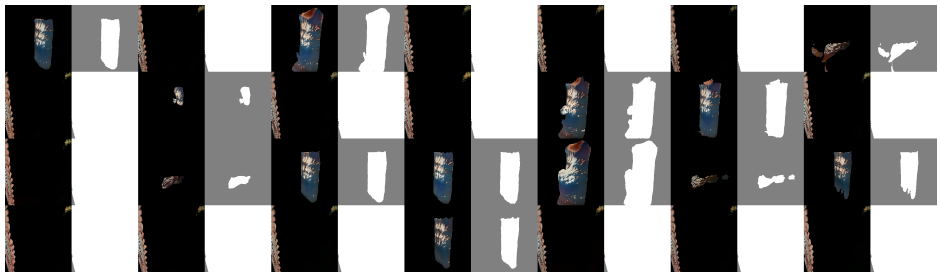
(a) Iteration 512



(b) Iteration 1024

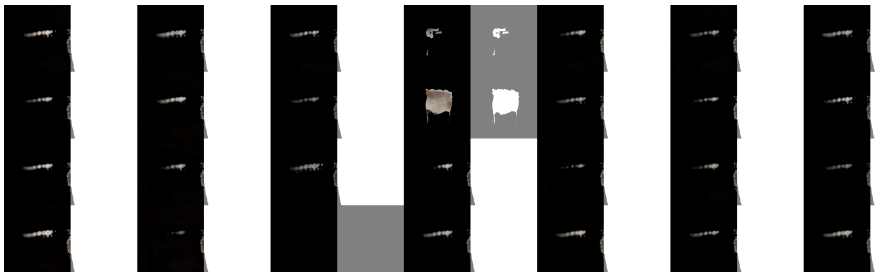


(c) Iteration 1536



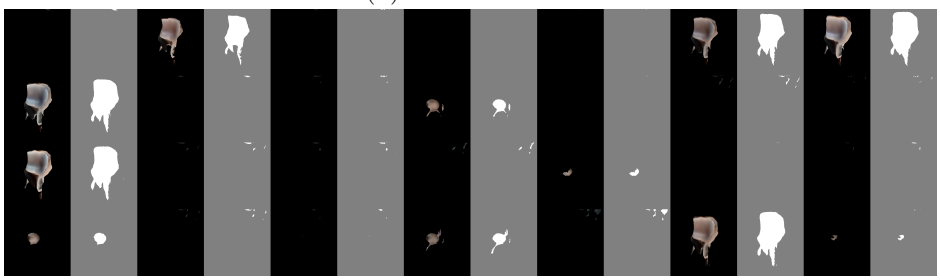
(d) Iteration 2048

Figure B.3.: Objectron Book

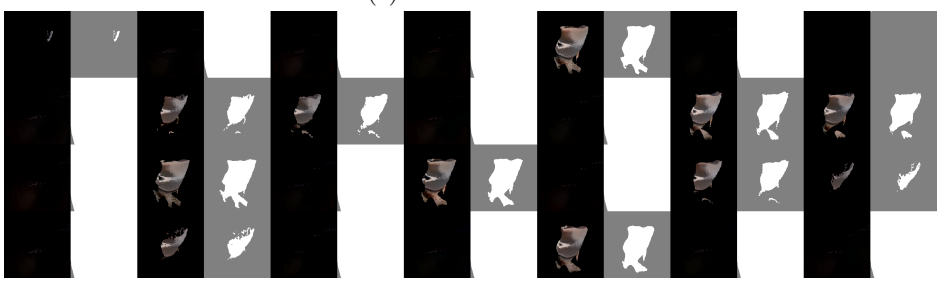


(a) Iteration 512

(b) Iteration 1024

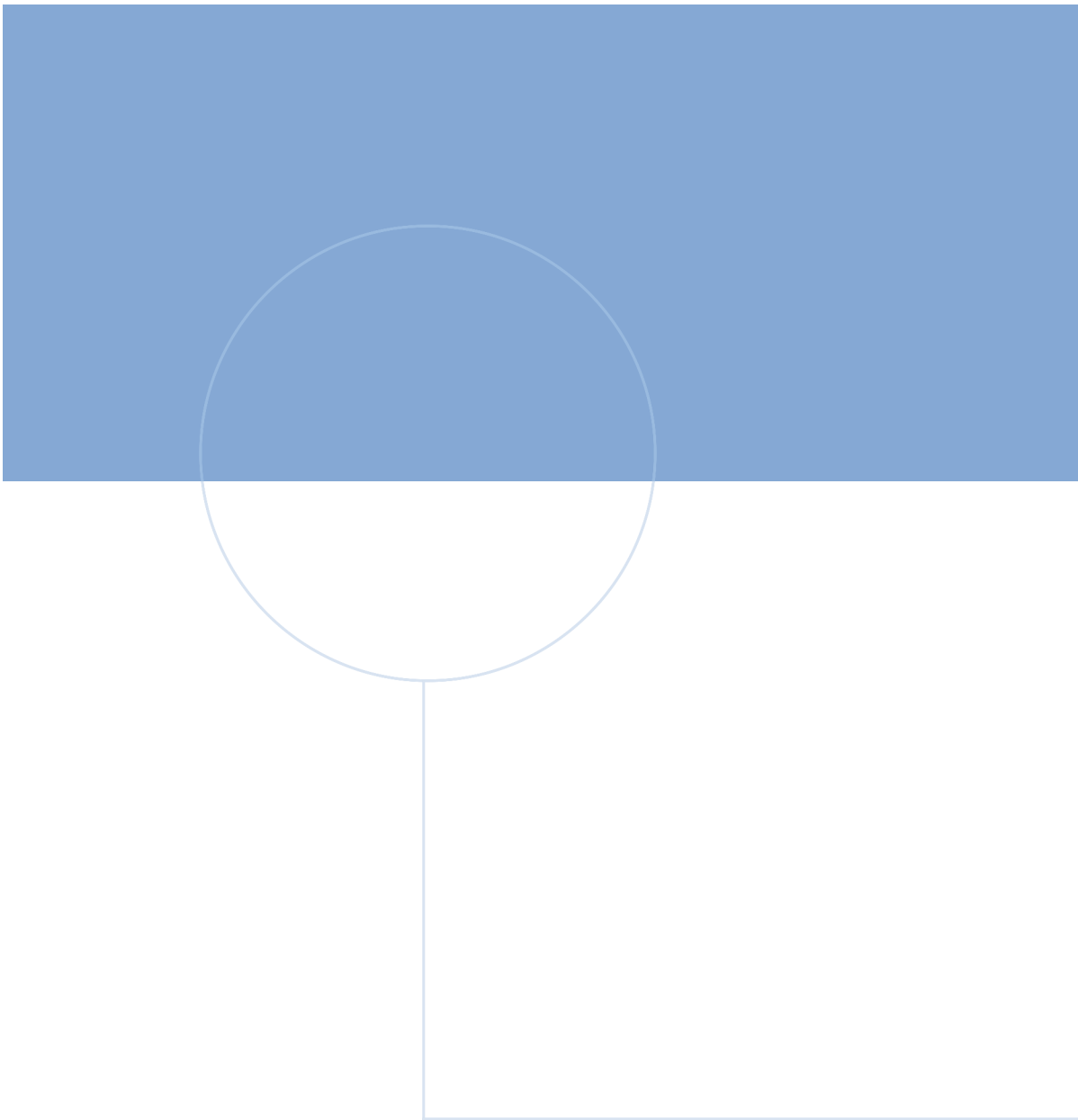


(c) Iteration 1536



(d) Iteration 2048

Figure B.4.: Objectron Chair



NTNU

Kunnskap for en bedre verden