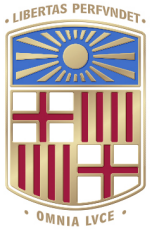

UNIVERSITAT DE BARCELONA

Facultat de Matemàtiques i Informàtica

DreamText: Harnessing Text Descriptions as an Intermediate Step for 3D Reconstruction

Bachelor's Thesis

Nazar Puriy Puriy Matriculation Number 20026414

1. Supervisor: Prof. Petia Radeva
2. Supervisor: Prof. Ricardo Marques

Submission date: June 13, 2023

Declaration

Unless otherwise indicated in the text or references, this thesis is entirely the product of my own scholarly work.

Barcelona, June 13, 2023

.....
Nazar Puriy Puriy

Abstract

The field of 3D generation, a rapidly emerging domain within generative AI, holds immense potential for various applications in fields such as architecture, product design, marketing, entertainment, and even in the novel realm of virtual reality. Enhancing 3D technologies bears significant utility in fostering society development and serves as a captivating and intellectually stimulating field of study, offering intriguing challenges and opportunities for innovative advancements.

In this dissertation, we introduce DreamText, an innovative Image to 3D generative model that harnesses text descriptors as an intermediate step for 3D reconstruction. Our proposed method effectively learns to describe objects within images, capturing crucial object details while disregarding extraneous contextual information such as lighting, point of view, or specific arrangements. This learned information serves as the foundation for generating compelling and novel views of the object, subsequently facilitating the creation of a comprehensive and accurate 3D reconstruction. Remarkably, our approach achieves high quality results, surpassing current state-of-the-art methodologies like RealFusion (CVPR2023)[1] in several test cases. Concrete evidence of our results can be observed in the following *link*.

Furthermore, we present FitFusion, which leverages the knowledge of a pretrained image generative model, Stable Diffusion, to train a Neural Radiance Field capable of generating 3D models when provided with image data during training. This concept stems from a comprehensive analysis and understanding of a previous model called Stable DreamFusion[2], combined with meticulous parameter tuning that culminates in improved outcomes.

This project entails extensive mathematical and experimental analysis of cutting-edge models, encompassing a comprehensive understanding of their intricate details.

Resumen

El campo de la generación 3D, un dominio emergente dentro de la IA generativa, encierra un inmenso potencial para diversas aplicaciones en campos como la arquitectura, el diseño de productos, marketing, entretenimiento e incluso en el novedoso ámbito de la realidad virtual. Mejorar las tecnologías 3D tiene por ende una utilidad significativa en el desarrollo de la sociedad y se presenta como un campo de estudio cautivador y estimulante, ofreciendo desafíos interesantes y oportunidades para avances innovadoras.

En esta tesis presentamos DreamText, un innovador modelo generativo que recibe imágenes como entrada y produce modelos 3D de estas. Para lograr su objetivo, el modelo utiliza descriptores textuales como un paso intermedio. Nuestro método aprende eficazmente a describir objetos presentes en imágenes, capturando detalles cruciales de estos y descartando información contextual superflua como la iluminación, el punto de vista o las disposiciones específicas. Esta información aprendida sirve como base para generar nuevas vistas del objeto, que serán utilizadas posteriormente para hacer una reconstrucción 3D del mismo. Notablemente, nuestro enfoque logra resultados de alta calidad, superando a metodologías de vanguardia como RealFusion (CVPR2023) [1] en numerosas pruebas realizadas. Se puede observar evidencia concreta de nuestros resultados en el siguiente *enlace*.

Además del modelo anterior, presentamos FitFusion, que aprovecha el previo conocimiento de un modelo generativo de imágenes ya preentrenado, Stable Diffusion, para entrenar un modelo llamado Neural Radiance Field capaz de generar objetos 3D cuando se le proporcionan datos de imagen durante el entrenamiento. Esta propuesta se deriva de un análisis exhaustivo y detallado de un modelo previo llamado Stable DreamFusion [2], cuyos parámetros han sido ajustados meticulosamente.

Este proyecto implica un extenso análisis matemático y la puesta en prueba de modelos de última generación, abarcando un completo entendimiento de sus detalles.

Resum

El camp de la generació 3D, un domini emergent dins de la IA generativa, conté un immens potencial per a diverses aplicacions en camps com l'arquitectura, el disseny de productes, el màrqueting, l'entreteniment i fins i tot en l'àmbit innovador de la realitat virtual. Millorar les tecnologies 3D té, per tant, una utilitat significativa en el desenvolupament de la societat i es presenta com un camp d'estudi captivador i estimulant, oferint reptes interessants i oportunitats per a avanços innovadors.

En aquesta tesi presentem DreamText, un innovador model generatiu que rep imatges com a entrada i produeix models 3D d'aquestes. Per aconseguir el seu objectiu, el model utilitza descriptors textuals com a pas intermig. El nostre mètode aprèn eficaçment a descriure objectes presents en imatges, capturant detalls crucials d'aquests i descartant informació contextual superflua com la il·luminació, el punt de vista o les disposicions específiques. Aquesta informació apresa serveix com a base per generar noves vistes de l'objecte, que seran utilitzades posteriorment per fer una reconstrucció 3D del mateix. Notablement, el nostre enfocament aconsegueix resultats d'alta qualitat, superant metodologies de última generació com RealFusion (CVPR2023) [1] en nombroses proves realitzades. Es pot observar evidència concreta dels nostres resultats al següent *enllaç*.

A més del model anterior, presentem FitFusion, que aprofita el coneixement d'un model generatiu d'imatges ja preentrenat, Stable Diffusion, per entrenar un model anomenat Neural Radiance Field capaç de generar models 3D quan se li proporcionen dades (imatges) durant l'entrenament. Aquesta proposta es deriva d'un anàlisi exhaustiu i detallat d'un model previ anomenat Stable DreamFusion [2]], els paràmetres del qual han estat ajustats meticulosament.

Aquest projecte implica una extensa anàlisi matemàtica i la posada a prova de models de última generació, abastant una comprensió completa dels seus detalls.

Contents

Acknowledgements

Agradecer el apoyo incondicional de mis profesores Petia y Ricardo durante esta tesis. A Petia por poner énfasis en llegar a unos resultados que veía imposibles. A Ricardo por las ayudas y el apoyo emocional durante todo el trayecto.

A Paula por ser la única persona que me ha acompañado toda y cada una de las tardes.

Chapter 1

Introduction

The aim of this chapter is to present a concise overview of the project's context, the author's motivations, and the structure of the thesis. Through this chapter, readers will gain a better understanding of the relevance and significance of the project, as well as the author's approach and contributions to the field. Please be aware that not all concepts or information can be presented within the scope of this dissertation. However, we have made a conscientious selection of the essential information necessary for comprehending the underlying work and results pertaining to the subject matter at hand.

1.1 Context

Artificial Intelligence (AI) is a multidisciplinary field that blends computer science and data analysis to develop problem-solving capabilities. The field encompasses sub-fields such as machine learning and deep learning, which are often associated with AI. At its core, AI is about enabling machines to learn from data and solve complex problems. This involves creating algorithms and models that can analyze vast amounts of data, identify patterns, and make predictions or decisions.

Definition 1.1. Machine Learning (ML) is a subset of AI in which the machine through the algorithms "learns" how to solve a problem learning from data.

The origins of artificial intelligence can be traced back to the mid-20th century, when Alan Turing posed the question, *"Can machines think?"* and developed the Turing test as a means of evaluating a machine's intelligence. Notable advancements were made during what is often referred to as the "golden age" of AI. In 1956, the *Logic Theorist* program was created to prove mathematical

theorems using logical rules. In 1958, Frank Rosenblatt unveiled the *Perceptron*, a neural network that is often regarded as the fundamental building block of other neural networks. *ELIZA*, one of the first chatbots, was developed in 1966, followed by the *Nearest Neighbor* algorithm in 1967, which marked the beginning of basic pattern recognition. Perhaps the most significant breakthrough of this era was the development of the *backpropagation* algorithm, which enabled the training of neural networks with multiple layers. This technique was first introduced in the early 1970s, and its impact on the field of artificial intelligence was profound. Unfortunately, progress in the field was hampered in the 1980s, when funding for research agencies like *DARPA* and *NRC* was suspended due to limited computing power and a lack of available data.

Over the years, the growth of computer power has followed *Moore's Law*, and processing structures like CPUs, GPUs and TPUs have become widely available in the market. Additionally, internet access has become ubiquitous in first-world regions, resulting in massive amounts of data produced through every user interaction with the digital world. This data has become a valuable resource for most internet-based companies. The combination of these two factors has enabled the creation of powerful algorithms and models that have surpassed human capabilities in various tasks like playing chess, driving cars, detecting objects or even in text and image generation.

The focus of this project will be on the emerging field of generative AI, which has rapidly gained attention due to its ability to generate realistic and diverse data.

Definition 1.2. A **Generative Artificial Neural Network** is a type of AI model that has the ability to create data such as text, images, video, or audio, that is, or aims to be, indistinguishable from real-world examples.

The field of generative artificial intelligence has experienced remarkable growth in the past decade. Image generative models have been around since the late 2000s when autoencoders [3] were utilized to compress and expand information, enabling the inference of new data. In 2013, Variation Autoencoders [4] were introduced, incorporating stochastic mechanisms into these models. A significant milestone occurred in 2014 with the introduction of Generative Adversarial Networks (GANs) [5], pioneering the generation of realistic images, style transfer, and accommodating diverse inputs for this purpose[6–8]. More recently, starting from 2019, there has been an influx of research papers on diffusion models which begin with a noisy image and progressively remove noise until reaching a real looking sample. In the field of text generation, Recurrent

Neural Networks (RNNs) [9] gained prominence around 2010, followed by the adoption of VAEs and GANs adapted to text generation, and finally the utilization of transformers structures [10], that leaded to the widely-known GPT models.

Nowadays, we are witnessing the development of exceptional models that demonstrate the ability to generate highly realistic images and text. However, 3D object generation is a field that has yet to be fully explored, partially due to the scarcity of 3D datasets.

1.2 Motivation

Stable Diffusion [11] stands out as a publicly available model, famous for its data handling capabilities, including text, images, depth maps, etc., high training speed, and high-quality generated images. Recently, the question was posed regarding the depth of knowledge of this generative model and whether it could be used to generate 3D data. Numerous papers have been published, one of which is called Dream Fusion [12], which achieved impressive results generating 3D models by leveraging the knowledge learned by diffusion models and applying it to train a Neural Radiance Fields [13] model (see Section 3.2), which generates 3D models from images.

The cheap and fast generation of 3D models is crucial for many industries worldwide: entertainment and media, including film, television, animation, and gaming, heavily relies on 3D models for creating realistic characters, environments, visual effects, and animations. Architecture and Construction use 3D models to visualize and plan structures, buildings, and infrastructure projects, allowing stakeholders to better understand the design, assess spatial relationships, and identify potential issues before construction begins. In Manufacturing and Product Design, for industries such as automotive, aerospace, and consumer goods, 3D models are used for product design, prototyping, and manufacturing, enabling designers to visualize and iterate on product concepts, test for fit and functionality, and optimize manufacturing processes. Marketing and Advertising employ 3D models to create visually appealing product presentations, virtual tours, and realistic visualizations for promotional materials. Virtual Reality (VR) and Augmented Reality (AR) rely heavily on 3D models to create immersive and interactive virtual environments.

Hence, improving these models is crucial for the progress of these industries and can represent a huge contribution to the society.

1.3 Objectives

The objectives of this Bachelor thesis project are multi-faceted, with both academic and personal motivations.
From an academic standpoint, this project is a dissertation that fulfills the requirements for a double degree, focusing on a field that encompasses them both. The computer science aspect of this project includes topics such as computer graphics and computer vision, as well as skills like memory management, class oriented programming, data processing, and proficiency in Python, NumPy, and Pytorch. The mathematical aspect of this project includes subjects such as statistics and linear algebra, and requires skills related to the theoretical foundations of the models such as problem detection, critical analysis and the capacity of writing mathematically all the demonstrations.

From a scientific perspective, the following objectives have been identified:

- To comprehend the relationship between Diffusion Models and Neural Radiance Fields models in the context of 3D generation.
- To investigate why the current performance level achieved by these models is not consistent with their individual performance levels.
- To suggest and implement one or more improvements to enhance the model's performance.
- To expand the model's functionality by incorporating new capabilities such as image conditioned 3D object generation.

From a personal perspective, there are two main goals: to apply the knowledge and skills acquired during my bachelor's degrees, and to explore and learn about the latest technologies in the field using the learned abilities.

1.4 Organization

The memory is structured in a linear way, beginning with general and basic concepts and going to the detailed architectures and notions used in the experimental part to achieve the goals. The memory can be divided basically into two parts: the theoretical framework, consisting of Chapters 2 and 3, and the experimental section, consisting of Chapters 4 and 5. In particular, each chapter includes:

2. **Background**: tackles the theoretical background of the project giving the foundations of the most basic components in machine learning.

3. **Related Work**: explores state-of-the-art models that are essential to comprehend our proposal.

4. **Analysis**: comprehensive analysis of the assumptions in the first part, giving mathematical demonstrations to them. Analysis of the model behaviour in order to the detect weaknesses and developing opportunities.

5. **Methodology**: our proposals consisting in a Model for quality enhancement(FitFusion) and a model for Image to 3D generation(DreamText). Besides, a method to improve generated samples diversity.

6. **Experiments**: conducted experiments to reach the proposed solution and comparisons. Examination of failure cases.

7. **Conclusions and Future Work**: analysis of the completeness of the objectives and the contributions made.

8. **Appendix**: additional information. Model specifications, tables, images and results.

9. **Bibliography**: a list of papers, articles and literature consulted during the project.

1.5 Notation

During this project common mathematical notation is widely used. However, some notations have to be mentioned.

- The "," symbol used inside a probability function $P(x, y)$ refers to the intersection or conjunction of both events x and y.

- The ":" symbol inside a probability function $P(x_1 : x_n)$ is used to compress the expression and refers to $P(x_1, x_2, \cdots, x_n)$.

- "A:=B" is used to refer an equality that is defining the expression A from a known expression B.

Chapter 2

Background on Artificial Neural Networks

This chapter serves as a comprehensive overview of the fundamental concepts in the field of machine learning. Our objective is to provide a solid understanding of the basic principles and building blocks that underpin the models presented later in this work.

An Artificial Neural Network, referred as ANN, is a computational model inspired by the human brain, designed with the purpose of mimicking the information processing and learning capabilities of a human. Typically, the objective is to establish a relationship between an input x and an output y. To achieve this, a substantial amount of pre-existing paired samples is provided, which serves as the basis for extracting the necessary information to make accurate predictions in the future.

2.1 Perceptron

Perceptrons were introduced for the first time in the 1950s. They were created to simulate the basic building blocks of biological neurons and perform simple binary classifications. Perceptrons were primarily constructed using a linear function, characterized by easily calculable derivatives combined with a non-linear function called activation function. This crucial aspect enables the neural network to effectively model intricate relationships and prevent the collapse of multiple linear functions into a single one.

Definition 2.1. In ML, an **Activation Function** is a non-linear function $h : \mathbb{R} \to \mathbb{R}$

The most usual ones are the sigmoid [14], the softmax [15] (see Equation (2.1)), the ReLU[16] (see Equation (2.2)) and the Leaky-ReLU [17] (see Equation (2.3), where $\alpha \in \mathbb{R}^+$).

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

$$\text{ReLU}(x) = \max(0, x) \tag{2.2}$$

$$\text{LeakyReLU}(x) = \begin{cases} \alpha x, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases} \tag{2.3}$$

Definition 2.2. In ML, a **Neuron**, conceived from the biological model of neurons, is a function f made of the composition of a linear function [18] $g : \mathbb{R}^n \to \mathbb{R}$ and an activation function h (see Equation (2.4)).

$$f(x) = h(g(x)) = h\left(\sum_{i=1}^{n} w_i\, x_i + b\right) \tag{2.4}$$

Definition 2.3. In Equation (2.4) $w_i \in \mathbb{R}$, $i \in \{1, ...n\}$, are called **weights** of the neuron while $b \in \mathbb{R}$ is called **bias**.

Definition 2.4. The **Perceptron** is a function $f : \mathbb{R}^n \to \{0, 1\}$, (see Equation (2.5)):

$$f(x) = \begin{cases} 0, & \text{if } g(x) < T \\ 1, & \text{if } g(x) \geq T \end{cases} \tag{2.5}$$

in which the activation function, $h(x)$, is a threshold. The input x represents the received data while the output y - binary class category annotated by $\{0, 1\}$.

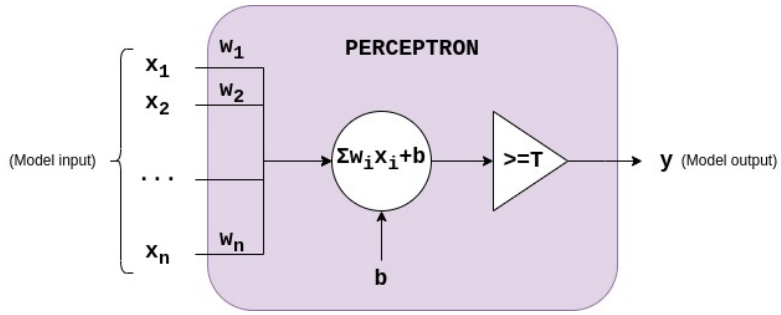

Figure 2.1: A visual representation of a Perceptron.

2.2 Multi-Layer Perceptron

Definition 2.5. In ML, a **Layer** is a linear function $f : x \in \mathbb{R}^n \rightarrow y \in \mathbb{R}^m$ in which $y = (y_1, \cdots, y_m)$ and $y_i = f_i(x)$ where $f_i(x)$ are neurons.

A layer is basically a group of neurons stacked together.

Definition 2.6. A **Multi-Layer Perceptron** is a function $mlp : x \in \mathbb{R}^n \rightarrow y \in \mathbb{R}^m$ made of the composition of multiple layers $f^1, \cdots, f^n$:

$$f(x) = f^n \circ f^{n-1} \circ \cdots \circ f^1(x) \tag{2.6}$$

In ML, an (**ANN**) (usually referred just as a model) is a mapping function that maps inputs x to an output y. Mathematically, it is composed by neurons distributed in multiple ways. If the function is carefully chosen, we can expect that it can extract insights from the data. For example, from car information such as the model, the manufactured year and the traveled distances (the input), we can extract the expected selling price (the output). Notice that all the information has to be previously expressed as numbers or vectors of numbers. One of the most basic or common ones is the multi-layer perceptron introduced in Definition 2.6.

2.3 Loss

In order to choose which concrete ANN model to use in a problem, we need a way to decide. One way to do it is defining a mathematical function that will tell us how well the network is performing: the loss function. Notice that this function, besides measuring the error of the network, will allow us latterly to improve the network capabilities by minimizing it.

Definition 2.7. A **Loss Function**, also referred as a cost function, is a function $loss : \mathbb{R}^m \text{x} \mathbb{R}^m \rightarrow \mathbb{R}^+$, $m \geq 1$, that is used to measure the difference between the output of a model and the expected one.

We can see that in order to evaluate the performance of an ANN, we need input data. This input data x is passed through the function of the ANN obtaining an output y_{pred} which is then passed together with the expected values y_{true} to the loss function. The loss function will return a value $\in \mathbb{R}^+$ representing how good the network predicts values, with lower values being better. Depending on the problem, there are multiple possible loss functions, the most common ones are the Mean Squared Error shown in Equation (2.7),

Binary-Cross Entropy Loss shown in Equation (2.8), the n-dimensional version of it called Categorical-Cross Entropy Loss[19] shown in Equation (2.9) or Kullback-Leibler (KL) Divergence [20] introduced later in Definition 3.2. In the equations below $\hat{y}$ is the real value or expected value, while y is the predicted one. N is the number of items in the dataset, which output value is predicted and C is the number of different categories.

$$\text{MeanSquaredError}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.7}$$

$$\text{BinaryCrossEntropyLoss}(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i)\log y_i + (1-\hat{y}_i)\log(1-y_i) \tag{2.8}$$

$$\text{CategoricalCrossEntropyLoss}(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C}\hat{y_{ij}}\log(y_{ij}) \tag{2.9}$$

The loss function plays a crucial role to decide which ANN to use. It is important to note that there are two distinct ways in which two ANNs can differ: structure and value. Structural differences arise from variations in activation functions, the number of layers, the number of neurons per layer, or the connectivity patterns. On the other hand, value differences occur when two neural networks have the same structure, but differ in the specific values of their weights and biases.

2.4 Gradient Descent

While selecting between neural networks in the first category cannot be directly automated, the process of choosing within the second category is an automatable procedure known as **training**. During training, the network learns to adjust its weights and biases to minimize the loss function, resulting in improved performance. To train the network, we can envision the loss function as a mountain, and our goal is to descend from it. To determine the direction of descent, we employ a mathematical tool known as the gradient.

Definition 2.8. If $f : \mathbb{R}^n \to \mathbb{R}$ is a differentiable function then the gradient of f is a function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ that for $x \in \mathbb{R}^n$ is defined as:

$$\nabla f(x) = \left(\frac{\partial f(x)}{\partial x_1}, \cdots, \frac{\partial f(x)}{\partial x_n}\right)$$

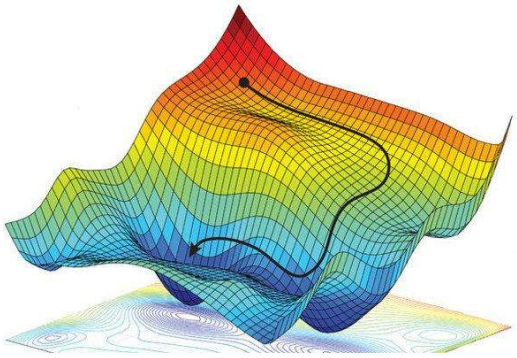

Figure 2.2: A Visual Representation of a Gradient Descent Process. Image extracted from [21]

.

Notice that the gradient points towards the direction in which the function f has higher variance. Then given $x \in \mathbb{R}^n$ and for a small enough $\delta \in \mathbb{R}^+$ we can assure, as the function is differentiable and therefore continuous, that $f(x) \geq f(x - \delta \nabla f(x))$.
We can then iterate $x_t = x_{t-1} - \delta_{t-1} \nabla f(x_{t-1})$ until arriving to a local minimum of f. Figure 2.2 illustrates how this process works.

Notice that the method mentioned before allows us to achieve the local minima of a given function. In our case, we fix the network structure, i.e. the activation functions, the number of layers, the number of neurons per layer, or how they connect. Then, we apply the gradient descent algorithm to the loss function, fixing as a constants the input x and y_{true}, and being the weights and the biases the unique variables. Therefore, reaching the local minima would mean reaching one of the the best functions inside our family of possible ones. Equation (2.10) shows the final function in which x and y_{true} are fixed values while the w_j^i and b^i are the variables that will be modified using gradient descent:

$$loss(w_j^i, b^i; f(x), y_{true}) \tag{2.10}$$

The function obtained at the local minimum is usually referred as **trained model** and the process of obtaining it, previously mentioned, is called **training**.

The problem that still is being not resolved, is which structure to use in each problem. Simple structures may not be sufficient for the function f to express the complexity of the data, a problem usually known as high bias or underfitting, while complex structures can lead to models that do not generalize well, a

problem referred as overfitting or high variance. Currently, one of the primary focuses in the field of ML revolves around studying different model structures and determining which models are most suitable for specific problems.

2.5 Autoencoders

An important structure that has to be conceptually introduced before proceeding the reading is the autoencoder [3].

Definition 2.9. An **autoencoder** is an ANN structure, made of the composition of two functions $g : \mathbb{R}^n \to \mathbb{R}^m$ and $h : \mathbb{R}^m \to \mathbb{R}^n$ called **encoder** and **decoder** respectively, and being $m < n \in \mathbb{N}$. The objective of the autoencoder is to minimize the difference between the input x and the output $h \circ g(x)$.

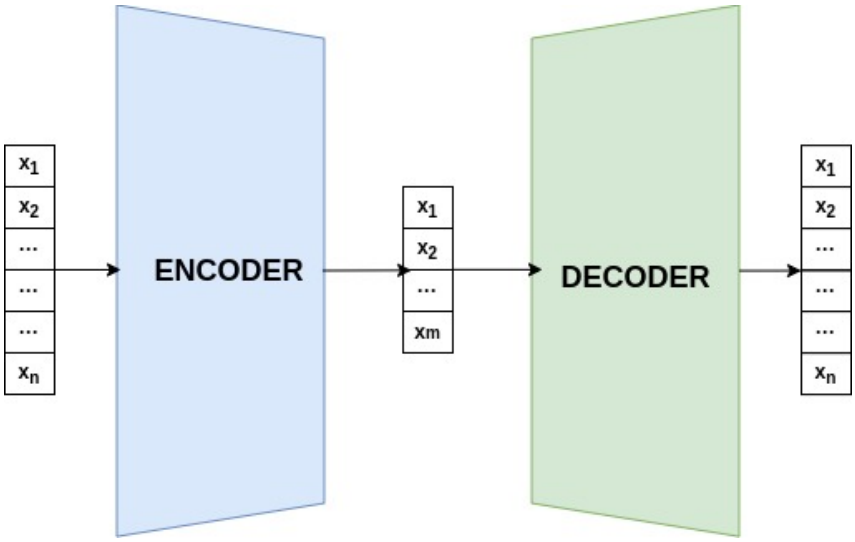

Figure 2.3: Autoencoder diagram. In blue the encoder that maps the input $x \in \mathbb{R}^n$ towards a lower dimensionality $z \in \mathbb{R}^m$. In green the decoder that maps this lower dimensionality to the previous higher one $y \in \mathbb{R}^n$. Notice that the internal structures of these models are complex and typically rely on the data provided or the specific problem being addressed.

Although the difference between input and output may appear straightforward, just use id function, the crucial aspect of the model lies in the dimensional reduction within the function, as $m < n$, see Figure 2.3. This reduction implies that the model must determine the essential information that needs to be conveyed. In other words, the model is tasked with selecting and transmitting the most crucial features or relevant aspects from the input to the output.

This process of information selection is a fundamental aspect of the model's functionality. After training, the encoder g can compress the data into a lower-dimensional representation. This compressed data can then be utilized to reconstruct the original content using the decoder h.

2.6 Further readings

The previously mentioned structures provide the most basic foundations for understanding the methodology and algorithms used in this project. For a more thorough understanding of these concepts, we recommend reading [22] or attending to the online ML Specialization by Andrew Ng available on *Coursera*.
In addition to the aforementioned basic structure, other relevant structures to understand this project include Convolutional Neural Networks [23], which are used for image processing, Recurrent Neural Networks [9], which are used for sequential data analysis, Attention layers [10], which are used to focus on the important data parts, and Dropout layers [24] used to avoid overfitting.

Chapter 3

Related Work on 3D Generation

In this chapter, we will introduce cutting-edge technologies in the machine learning field, which are essential to understand our final proposals as they serve as the foundation for our work. In Section 3.1, we will delve into image generative models known as diffusion models. Embarking on our exploration, we will delve into the foundational models, gradually advancing towards the cutting-edge technology known as Stable Diffusion [11]. Subsequently, in Section 3.2, we will explore Neural Radiance Fields [13], a 3D generative models which utilize images with known camera poses to reconstruct underlying objects. Finally, in Section 3.3, we will present DreamFusion [12], our foundational work that combines the aforementioned technologies to enable text-to-3D generation.

3.1 Diffusion Models

We are interested in 3D generative models able to learn to effectively reconstruct 3D representations of objects, relying on knowledge and insights derived from an image generative model. Consequently avoiding the need for expensive 3D datasets. Therefore, it is imperative to thoroughly comprehend the inner workings of the image generative process that we employ.

Diffusion models [25; 26] are latent-variable generative models (Definition 1.2) that learn to gradually transform a sample from a tractable noise distribution towards a data distribution. The main idea behind them is a deconstruction and a learned reconstruction of an image. In this case, differing from variational autoencoders [4], an autoencoder that includes uncertainity in its process, the deconstruction is a fixed process while the reconstruction is the learned one.

Intuitively, our objective is to take a random noisy image, Figure 3.1, and progressively remove the noise, ultimately generating a final realistic image. In order to train the model for this task, we will utilize real images and introduce noise artificially. The model will then be trained to predict the added noise. Consequently, the model is comprised of two main components: the forward process, also known as the noising process, and the backward process, also referred to as the denoising process.

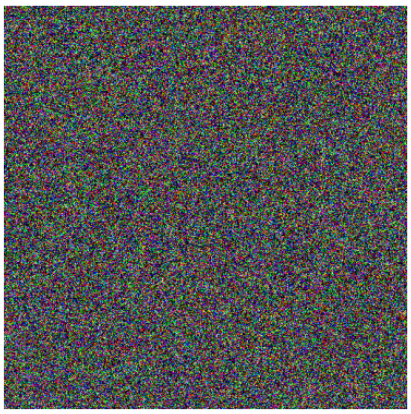

Figure 3.1: Noise image. Colour values are independently sampled from normal standard distribution.

3.1.1 Diffusion models architecture

Forward process

The objective of the forward or deconstruction process is to convert an image to a noise image, Figure 3.1. Mathematically, it aims to convert a data distribution into a Gaussian noise distribution. Although it is possible to convert directly an image to pure noise a Markov Chain, (see Definition 3.1), is used to do so, as experimentally it has shown better performance [26]. The number of steps used in the chain is $T \in \mathbb{N}$, being 1000 the most typical value.

Definition 3.1. A **Markov chain** is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state of the previous one. Mathematically, if $x_0, x_1, \cdots, x_n$ are a sequence of already occured events then $P(x_{n+1}|x_0 : x_n) = P(x_{n+1}|x_n)$.

Let us see mathematically how this process works. Be x_0 an image from the training dataset. Then every subsequent image can be obtained as $q(x_t|x_{t-1}) :=$

$\mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$. Here $\beta_1, \cdots, \beta_T$ are called the variance schedule and take linearly values from 10^{-4} to 0.02. The variance schedule refers to a mechanism that determines the rate at which the input will degrade into noise within a generative model. By adjusting the hyperparameter β, the model can control the characteristics of the generated samples. A higher value of β, closer to 1, corresponds to samples being generated with a distribution having a mean closer to 0 and a higher variance. From this definition and being ϵ a random variable following a standard normal distribution, we can write x_t depending on x_{t-1} as shown in Equation (3.1):

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon. \tag{3.1}$$

One key factor to notice is that the previous expression can be rewritten so that the x_t term depends directly on x_0 instead of x_{t-1} (see Equation (3.2)), where $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$:

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon \tag{3.2}$$

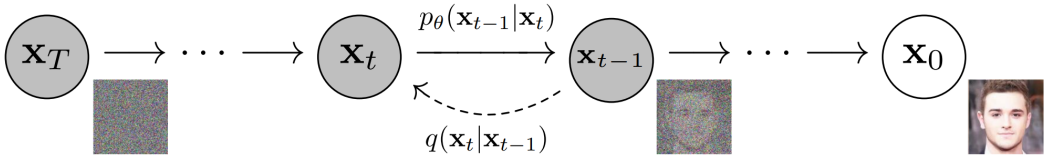

Figure 3.2: Diffusion backward and forward process as a Markov chain (figure extracted from [26]).

Notice that this allows to evaluate x_t in one step instead of doing t iterations. Hence, boosting considerably the execution speed for the forward process, while allowing to maintain the step by step reconstruction for the backward process, which leads to better generated quality. The derivations are shown in Section 4.1.1.

Figure 3.2 illustrates how the process works.

Backward process

The backward process, also known as the denoising process, is responsible for transforming noisy images into less noisy versions of themselves. At the end of its Markov chain, it aims to recover the original clean image or reduce the level of noise present in the input image. Every previous version of the noised image can be obtained as $p_\theta(x_{t-1}|x_t) := \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$. Notice

that μ_θ and Σ_θ depend on θ that are the parameters of the neural network that will be trained. Their definition is done in Section 4.1.1 to ensure they are specifically designed to simplify the final loss function. It is worth noting that both do not receive x_0 as an explicit input. This implies that they must learn to reconstruct the image solely based on the noised image and the time t.

3.1.2 Loss Function in Diffusion Models

Having defined both the forward (noising) process and the backward (denoising) process, the subsequent step is to devise a suitable loss function for training the neural network with the objective of learning to reconstruct real images. A natural approach is to maximize the probability of the backward process in producing realistic samples, as this aligns with our goal of creating a model that can generate high-quality images. This is mathematically the same as to minimize the negative log likelihood, as shown in Equation (3.3):

$$\mathbb{E}\left[-\log p_\theta(x_0)\right] \tag{3.3}$$

The loss function can be derived as seen in Section 4.1.1 to a lower bound, and a more tractable form, as seen in Equation (3.4):

$$\mathbb{E}_q\left[D_{\mathrm{KL}}(q(x_T|x_0)\,\|\,p(x_T)) + \sum_{t>1} D_{\mathrm{KL}}(q(x_{t-1}|x_t,x_0)\,\|\,p_\theta(x_{t-1}|x_t)) - \log p_\theta(x_0|x_1)\right] \tag{3.4}$$

Notice that we can ignore the $L_T = D_{\mathrm{KL}}(q(x_T|x_0)\,\|\,p(x_T))$ term as it has no trainable parameters and it would tend to 0 as $q(x_T|x_0) \approx \mathcal{N}(x_T; 0, I)$ if enough iterations are made, which is exactly the distribution of $p(x_T)$, taken by definition. On the other hand, the terms $L_{t-1} = D_{\mathrm{KL}}(q(x_{t-1}|x_t,x_0)||p_\theta(x_{t-1}|x_t))$ are one of the key components as they calculate the difference, and not the distance as it is not a symmetric function, between the known distribution and the one that the network predicts (see Definition 3.2). Finally, the last term, $\log p_\theta(x_0|x_1)$ is just the probability of generating the real image given the one-step noised image.

Definition 3.2. In mathematical statistics, the **Kullback-Leibler divergence** or D_{KL} is defined for two distributions P and Q of continuous random variables as $D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x)\, log\left(\frac{p(x)}{q(x)}\right) dx$, where p and q are the probability density functions of the variables.

Taking a specific structure for μ_θ and $\Sigma_\theta = \sigma^2 I$, we can arrive to the last equation for the L_{t-1} terms (see Equation (3.5)). Notice that, as previously mentioned, the specific election of these both therms is done in Section 4.1.1 with the objective of simplifying the final loss formula:

$$\mathbb{E}_{x_0,\epsilon}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}||\epsilon-\epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0+\sqrt{1-\bar{\alpha}_t}\epsilon,t)||^2\right] \tag{3.5}$$

Notice that now $\epsilon_\theta(x_t,t)=\epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0+\sqrt{1-\bar{\alpha}_t}\epsilon,t)$ is the trainable part aiming to predict the real noised from the noised image and the time-step.
Finally, the L_0 term, assuming that the image consists of integers in $\{0,1,\cdots,255\}$ scaled linearly to $[-1,1]$, can be written as Equation (3.6).

$$p_\theta(x_0|x_1)=\prod_{i=1}^{D}\int_{\delta_-(x_0^i)}^{\delta_+(x_0^i)}\mathcal{N}(x,\mu_\theta^i(x_1,1),\sigma_1^2)dx$$

$$\delta_+(x)=\begin{cases}\infty, & \text{if } x=1\\ x+\frac{1}{255}, & \text{if } x<1\end{cases}\quad \delta_-(x)=\begin{cases}-\infty, & \text{if } x=-1\\ x-\frac{1}{255}, & \text{if } x>-1\end{cases} \tag{3.6}$$

Notice that we are taking each pixel individually, supposing them independent, and calculating the probability that the pixel falls in the correct color, mathematically the integral in the range of the colour bin. D is the number of pixels of an image.
Finally, for the purpose of predicting noisy images, an U-Net [27], Figure 3.3 architecture will be employed as a basic network structure. Specifically, we will utilize the model architecture presented in [11], making use of its publicly pretrained version available on Hugging Face at the following *link*.
While this initial implementation of diffusion models had yielded commendable results, see Figure 3.4, there were still room for further improvements in several key areas. These areas include addressing the computational complexity, enhancing generation speed, improving scalability for larger image sizes, incorporating guided generation techniques, and enhancing the overall sample quality.

3.1.3 Improved proposals for diffusion models

Although this default implementation is capable of producing great results, further improvements were made in the model architecture, training schedule, model input and training procedure in order to enhance quality, speed or features of the preexisting model. This implementations has served as the foundation for the final model we employ; thus, we will now briefly illustrate

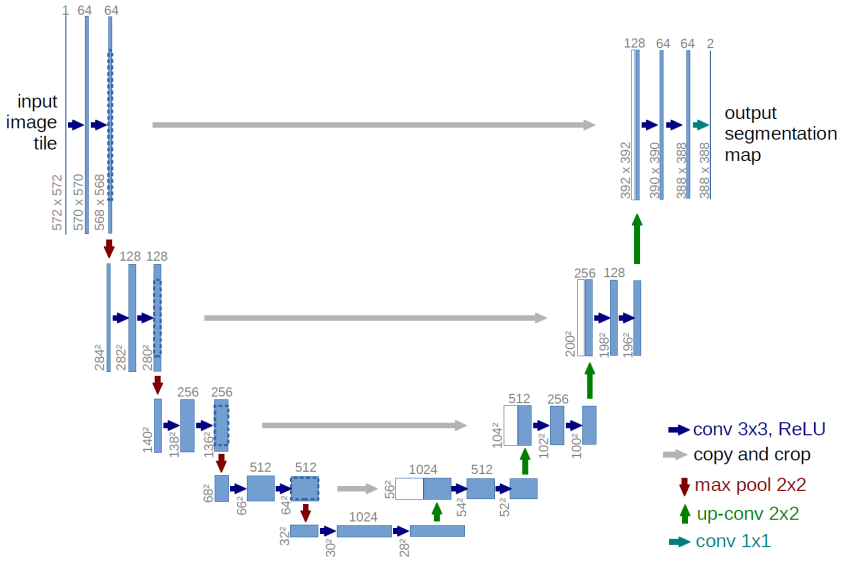

Figure 3.3: U-Net: A powerful deep learning architecture known for its exceptional feature extraction and contextual understanding. (figure extracted from [27]).

them in the following paragraphs.

Denoising Diffusion Implicits Models [28] show that the obtained local minima with the loss given in Equation (3.4) are also a local minimum for other Markov Chains. Two special cases are disgussed in this paper: deterministic backwards process, and faster sampling process. First is done taking special variance values and the second is done by making bigger denoising sampling steps (i.e. generating x_{t-2} or lower given x_t). They both together show faster and greater generative process than the previous implementation [26]. Moreover, the deterministic generation process enables interpolation of the generated images by conducting the interpolation in the noise space and subsequently generating the samples.

Improved Denoising Diffusion Probabilistic Models [29] tackles the possibility of using a learnable variance instead of the aforementioned fixed one. They also introduce a new variance schedule for smaller images (Equation (3.7), with s=0.008 and T being the number of steps in the diffusion process):

$$\bar{\alpha}_t = \left(\frac{\cos\left(\frac{t/T+s}{1+s}\frac{\pi}{2}\right)}{\cos\left(\frac{s}{1+s}\frac{\pi}{2}\right)} \right)^2 \tag{3.7}$$

The paper "Diffusion Models Beat GANs on Image Synthesis" [30] uses the

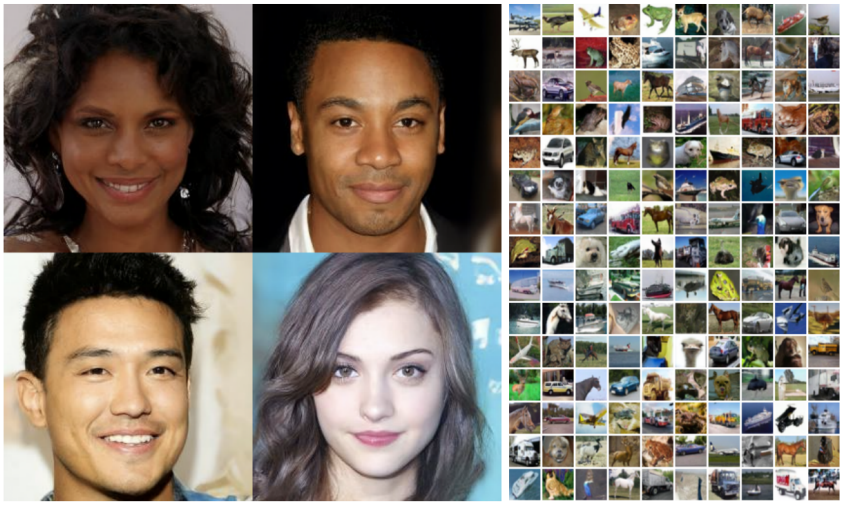

Figure 3.4: Diffusion models DDPM [26] implementation, figure extracted from original paper. Generated samples on CelebA-HQ 256 × 256 (left) and unconditional CIFAR10 (right)

previous improvements and adds a class condition y guiding to the backward process in order to make specific generations. To do so, they multiply the probability of obtaining a particular x_t by the probability of that generated x_t being part of the specified class y. This can be done with a pre-trained and fine-tuned classifier.

Finally, in the work of Classifier-Free Diffusion Guidance [31], the inclusion of a label y in the network's input is sometimes employed, while at other times, the symbol $\emptyset$ is added to indicate the absence of a label. This label is used as a conditioning and adds information to the model about the expected final image. Subsequently, the backward or generative process can be guided according to Equation (3.8). In this equation, the parameter s denotes the guidance scale, typically ranging from 0.1 to 5, which determines the level of importance assigned to the text in the generation process. This method serves as a foundation for further advancements discussed in subsequent works [32]:

$$\hat{\epsilon}_\theta(x_t|y) = \epsilon_\theta(x_t|y) + s(\epsilon_\theta(x_t|y) - \epsilon_\theta(x_t|\emptyset)) \tag{3.8}$$

3.1.4 Stable Diffusion

Stable Diffusion is referred to a group of publicly available models based on the paper "High-Resolution Image Synthesis with Latent Diffusion Models" [11].

The underlying concept driving these approaches is perceptual image compression, which involves reducing the size of images while preserving their essential information. This is achieved by training an auto-encoder with a perceptual loss [33] and a patch-based adversarial objective [34–37]. In essence, these methods learn how to effectively reduce the dimensionality of images while retaining all the necessary information to facilitate accurate reconstruction and preserve fine details.

As part of the training process, all images are encoded into a lower-dimensional space known as the latent space. This latent space serves as primary domain for model training. During the generative phase, samples are generated in the latent space and subsequently decoded to obtain high-quality images. This approach allows for efficient representation and manipulation of images while maintaining fidelity to the original content.

Furthermore, these approaches offer the flexibility to incorporate multi-modal inputs through an intermediate mapping of the conditioning. Once mapped to the intermediate space, the conditions are integrated into the U-Net backbone using a cross-attention mechanism [10], see Section 3.1.4. Figure 3.6 provides a visual depiction of how the autoencoder and conditioning elements are integrated into the diffusion pipeline. Detailed specifications of the model can be found in the original paper [11] or on the Hugging Face platform.

Cross-Attention mechanism

The objective of the cross-attention mechanism is to influence one embedding with another embedding in order to enrich the information within a given context, such as an image, text, or sound. In a concrete example, let us consider a blurry image that depicts an indistinguishable, sticky iron object. The model's initial embedding for the image would contain this limited information. However, by using the text embedding for the sentence "a sword," we can leverage the cross-attention mechanism to modify the image embedding and provide it with additional information.

Figure 3.5 illustrates this process. Formally, we have two embeddings, denoted as V and W. These embeddings are multiplied by three matrices, namely W^Q, W^K, and W^V, resulting in the query (Q), key (K), and value (V) vectors. The key represents the contextual information, and its associated details are stored in the value. The query represents the target information that needs to be contextualized. f the embedding W needs to be contextualized, it will be

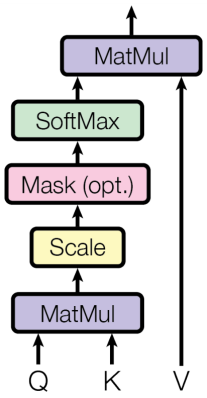

Figure 3.5: Attention mechanism. Q is influenced with V using K. Mask can be ignored for the purposes of this explanation (figure extracted from [10]).

multiplied by the matrix W^Q, while the other vector (e.g., the contextual information) will be multiplied by the matrices W^K and W^V. To achieve this, we compute the dot product between the query and key vectors, yielding a vector that represents the influence of different contextual elements on the query. The softmax function is applied to normalize these outputs. Finally, the resulting attention weights are multiplied by the values of the key to contextualize it. See Equation (3.9). It is worth noting that these matrices are trainable, enabling the model to learn how to contextualize effectively. For a more comprehensive understanding, we recommend referring to the paper [10], which provides a thorough explanation of this process.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.9}$$

In our particular case, we employ a CLIP model to obtain embeddings that serve as conditioning inputs. These embeddings are then seamlessly integrated into the model using the corresponding cross-attention mechanism.

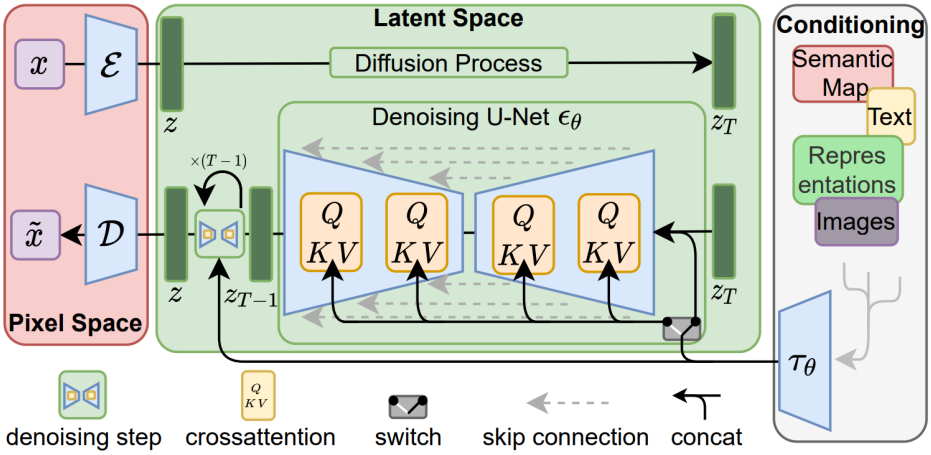

Figure 3.6: The figure shows how the autoencoder (red) and the conditioning (white) fit in the diffusion model's (green) pipeline. Figure extracted from the original Stable Diffusion paper [11].

3.1.5 CLIP

"Learning Transferable Visual Models From Natural Language Supervision" [38] presents CLIP, which is a model trained to pair images with text. To do so, there are two independent encoders: one for the images and one for the text, that are trained to produce a vector in the same dimensional space. For every batch of images with their captions, a list of image vectors $I_1, \cdots, I_n$ and a list of text vectors $T_1, \ldots, T_n$ are produced. Then, a scalar product is made among all the image and text vectors in a batch, obtaining a matrix of size nxn. This matrix contains on the diagonal the product corresponding to correct match of image and text vectors and outside - the wrong ones. Consequently, the objective of the model, a contrastive loss, is to maximize the values in the diagonal and minimize the ones outside.

CLIP offers two key features: text embedding space, which provides a strong representation of textual information that can be utilized to condition Stable Diffusion generation, and text pairing capability, enabling text-image matching as a powerful classifier, which we leverage for our DreamText proposal.

3.2 Neural Radiance Fields

Once a robust image generative model has been established, the next step is to define a 3D generative model that will be combined with it. Additionally to the disseration work, we present a comprehensive and **publicly available**

repository (*link*), made by ourselves, that covers this technology extensively. The repository includes complete implementations built from the ground up, allowing users to gain a deep understanding of the inner workings in an easy and interactive way.

NeRF (Neural Radiance Fields) models, as described in the paper [13], are designed to generate realistic 3D object reconstructions using a collection of images with known camera poses, without relying on pre-existing 3D datasets like ShapeNet [39]. Previous approaches attempted to predict voxel values within the 3D space in order to align their renderings with the given training images [40–42]. While these methods achieved impressive results, they lacked scalability to higher resolution imagery due to their computational intensity and memory requirements. NeRF addresses these limitations by learning a function that predicts voxel values, rather than directly predicting the values themselves. The behavior of NeRF can be understood in two main steps. First, it learns a function that outputs the color and density of a particular point in 3D space. The second step involves rendering images by "stacking" these values in a process called volume rendering. Figure 3.7 illustrates how the model works.

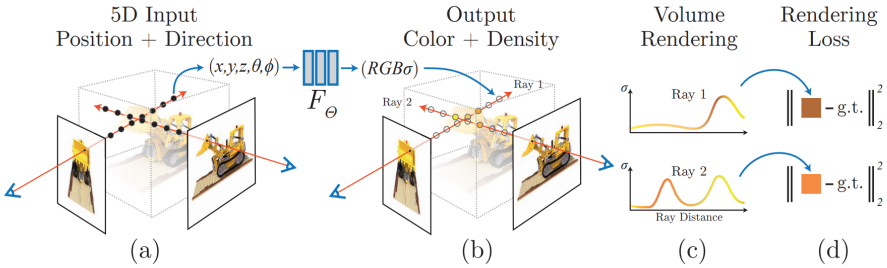

Figure 3.7: An overview of the NeRF procedure (extracted from [13]). (a) A set of samples is colected along each ray associated with each pixel. (b) Colour and density values are predicted for every sample, which is essentially a point in the 3D space. (c) Colour and density values are used to render the predicted images using volume rendering techniques. (d) The difference between the predicted image and the real image is used to train the colour and density predicting function F_Θ.

3.2.1 Function to predict colour and density

The objective is to train a function that will predict colour and densities for given 3D points, which then will be rendered into an image as explained in Section 3.2.2. The function is defined as $F_\Theta : \mathbb{R}^5 \rightarrow \mathbb{R}^4$, which takes (x, y, z, θ, ϕ) and outputs (R, G, B, σ), where (x, y, z) are the point coordinates, (θ, ϕ) is the

view direction and (R, G, B) is the output colour with σ being the associated point density. Intuitively, we want to predict the colour and density of a particular point in 3D space by training a network that receives the position of the point and in which direction, we are looking at it. The view direction is used to simulate the specular behaviour of the object, i.e. the reflections, as it is not the same to look towards a specific object from a different point of view. To encourage the 3D representation to be multiview consistent, the density σ only depends on the 3D coordinates (x, y, z), and not on the view direction (θ, ϕ).

3.2.2 Volume Rendering

Once there is a function able to predict the colour and density values for each point in 3D space, we need to perform a rendering to transform these values into an image. To do so, classical principles of volume rendering are used [43]. The density is interpreted as a probability of the ray ending in the infinitesimal position (x, y, z). Being now o - the origin of the ray, d - the direction of the ray, and t_n and t_f - the near and far bounds, we can obtain the point at time t as $\mathrm{r}(t) = \mathrm{o} + t\mathrm{d}$ and use it as input to the colour and density functions that will be rendered as follows:

$$C(\mathrm{r}) = \int_{t_n}^{t_f} T(t)\, \sigma(\mathrm{r}(t))\, c(\mathrm{r}(t), \mathrm{d})\, dt. \tag{3.10}$$

Notice that $F_\Theta(x, y, z, \theta, \phi) := (c(x, y, z, \theta, \phi), \sigma(x, y, z))$. Meanwhile, $T(t)$ is the probability of the ray not being hit until time t. This is known as accumulated transmittance, (see Equation (3.11)):

$$T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathrm{r}(s))\, ds\right) \tag{3.11}$$

Section 4.1.2 shows how this expression is obtained.

Numerically, this expression is estimated using quadrature. Stratified sampling is used in order to ensure that every position of the ray has probability of being sampled. This approach is necessary, because if the same spatial location is repeatedly sampled, it can lead to the emergence of grid-like patterns in the reconstructed 3D representation. To do so, the space $[t_n, t_f]$ is divided into N evenly spaced bins. Then, within each of the bins, a time t is sampled as seen in Equation (3.12) in which $i \in \{1, \cdots, N\}$, where N is the number of samples taken within each ray:

$$t_i \sim \mathcal{U}\left[t_n + \frac{i-1}{N}(t_f - t_n),\ t_n + \frac{i}{N}(t_f - t_n)\right] \tag{3.12}$$

Using this sampling, and supposing colour and density constant between two times t, we can simplify Equation (3.10) into Equation (3.13) as seen in Section 4.1.2:

$$\hat{C}(\mathrm{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))\mathrm{c}_i, \text{ where } T_i = \exp\left(-\sum_{j=1}^{i-1}\sigma_j\delta_j\right), \tag{3.13}$$

where $\delta_i = t_{i+1} - t_i$ the distance between adjacent samples and $\sigma_i = \sigma(\mathrm{r}(t_i)))$ and $\mathrm{c}_i = \mathrm{c}(\mathrm{r}(t_i))$.

3.2.3 Training a Neural Radiance Field

Before the training of the previously defined F_θ function, the one that predicts colour and density, some improvements were added:

- Positional encoding to ensure that the network is able to express sharp contours as follows in Equation (3.14). Notice that p is a vector, in our case we use it for the vector of positioning and for the vector of view direction:

$$\gamma(p) = (\sin(2^0\pi p), \cos(2^0\pi p), \cdots, \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p) \tag{3.14}$$

Experimentally good values of L are 10 for the encoding of the position and 4 for the encoding of the view direction.

- Hierarchical volume sampling: two stages of sampling to improve ray tracing efficiency and accuracy. Initially, N_c positions are sampled throughout the volume to estimate hit probabilities. In the second stage, an additional N_f positions are sampled, with an emphasis on oversampling regions where the hit probabilities are higher. First N_c samples are used to train a coarse network which output is referred as $\hat{C}_c(\mathrm{r})$, while the $N_c + N_f$ are used to train the final fine network which output is referred as $\hat{C}_f(\mathrm{r})$.

- COLMAP from structure-from-motion package [44] is used to estimate camera values such as position and view direction for real world samples taken from videos.

With these assumptions, the loss is simply the difference between the rendered and true pixel values, see Equation (3.15)), where $C(\mathrm{r})$ are the real or expected pixel values.

$$\mathcal{L} = \sum_{\mathrm{r}\in R}\left[\|\hat{C}_c(\mathrm{r}) - C(\mathrm{r})\|_2^2 + \|\hat{C}_f(\mathrm{r}) - C(\mathrm{r})\|_2^2\right]. \tag{3.15}$$

It is worth noting that the pixels can be trained independently. This means that there are no restrictions on selecting which subset of pixels to train, regardless of their proximity or whether they belong to the same image.

3.2.4 Instant Neural Graphics Primitives

One of the drawbacks of first NeRF model [13] and many of its variants is the long execution time required for training the model. Fortunately, [45] achieves a reasonable time without the loss of quality. This enhancements come from first programming directly the network on CUDA, and secondly from a trainable efficient encoding of the input (x, y, z) position.

The objective of the encoding is to associate each position in 3D space with a vector of values that captures meaningful information for the network. To do so, each point in 3D-space is associated with a set of trainable encodings via hash function. In order to make the storage efficient, the feature space, our 3D-space where the object lies, is partitioned into L (usually 16) 3D-grids of different size. The size goes from N_{min} to N_{max} as shown in Equation (3.16):

$$N_l := [N_{min} \cdot b^l], \qquad b := \exp\left(\frac{\ln N_{max} - \ln N_{min}}{L-1}\right) \tag{3.16}$$

Usual values are 16 for N_{min} and from 512 to 524288 for N_{max}. Notice that the number N^3 indicates the number of cubes in which the 3D-space will be partitioned. $b \in [1.26, 2]$ and $l \in 0, 1, \cdots, L$. Observe that the different grids are independent (see Figure 3.8 step 1).
Then, given a concrete point in the 3D space, and for each grid, upper and lower vertex are obtained with floor and ceil functions, as shown in Equation (3.17):

$$x_u = \lceil x \cdot N_l \rceil \qquad x_l = \lfloor x \cdot N_l \rfloor \tag{3.17}$$

Afterwards, all the possible combinations are made by extracting for each dimension a component from one of the two vectors, obtaining the $2^3 = 8$ cube vertexes. Then, the vertex corresponding encoding values are obtained from a hash-table. The hash tables contain T entries of size F. Typical values for T range from 2^{14} to 2^{24} and 2 for F. The hash table positions of each cube vertex are obtained using a spatial hash function (Equation (3.18)) [46], in which d is the number of dimensions ($d = 3$ for 3D-space and π are the prime numbers 1, 2 654 435 761 and 805 459 861:

$$h(x) = \left(\oplus_{i=1}^{d} x_i \pi_i\right) \tag{3.18}$$

$\oplus$ represents the bit-wise XOR operation (see Figure 3.8 step 2).

Finally, the obtained values corresponding to the vertex of the cube that contains the concrete point are linearly interpolated. This gives the ·D point value for a concrete grid size. Then, for each grid level the values are taken and concatenated together, adding also the encoding for the viewing direction, obtained as previously in [13]. This encoding will be used as input to the NeRF MLP, which now is reduced from the original size to only 2 layers, that will predict the colour and density (see Figure 3.8 steps 3,4,5.) This approach enables quicker sampling while upholding the sample quality.

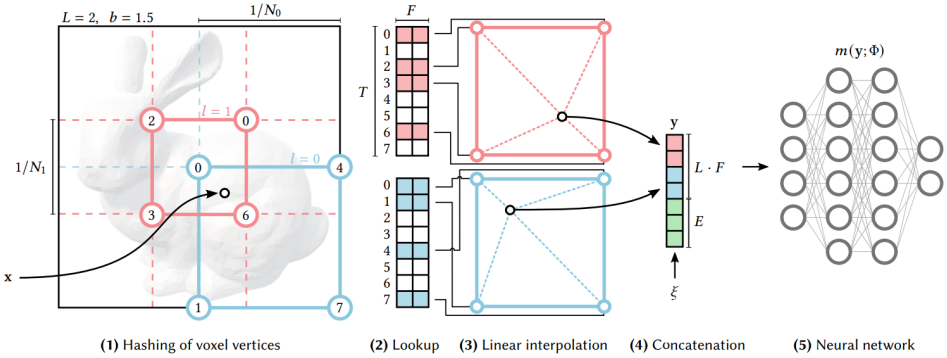

Figure 3.8: An overview of the multiresolution hash-encoding (extracted from [45]). This 2D example showcases two grid levels: red and blue. In this illustration, the pixel x is queried (1). Subsequently, the vertices of the corresponding square are obtained, and their values are retrieved from the hash table (2). These obtained values, four in this case (or eight for our 3D samples), are then interpolated to derive the final value (3). The values obtained at each grid level are concatenated, along with the encoding of the view direction (4). This resulting vector is subsequently input into the network (5).

3.3 DreamFusion

Now, with the introduction of both a text-to-image generative model and an image-to-3D model, it is time to explore their combination to achieve text-to-3D model generation.

The objective is to use the prior gained knowledge of a diffusion model to guide the NeRF model in its reconstruction process. [12] develops this idea, producing high-quality 3D models without relying on explicit 3D training data.

This is a significant advancement as acquiring large-scale 3D datasets, such as ShapeNet [39], can be challenging, time-consuming, and expensive.

3.3.1 DreamFusion architecture

The model architecture consists on the join of a diffusion model and a NeRF model. As explained in Section 3.1 the objective of the diffusion is to generate realisting looking data by removing noise. On the other hand, NeRF model learns a 3D representation of an object from different views of it.

The whole process can be visualized in Figure 3.9. First, random camera poses are sampled: camera position, camera view direction, field of view, etc. These values are passed as input to the NeRF model (Figure 3.9, step 1) that will render an image from that point of view (Figure 3.9, step 2). This is done in the exact same way as the NeRF does it with a standard training dataset. Then, noise is sampled from Gaussian standard distribution and a random time t, inside the range of the diffusion model (ours is 0 to 1000), is used to add this noise to the image (Figure 3.9, step 3). This noise is added using the forward diffusion step previously mentioned. Then, the noised image (Figure 3.9, step 4) is passed to the diffusion model that will use the text prompt (Figure 3.9, step 5) to predict the noise (Figure 3.9, step 6). The difference between the predicted and real noise will work as the error for the NeRF model (Figure 3.9, step 7).

Intuitively, the NeRF model serves to render images, which are intentionally corrupted with noise. These noisy images are then fed into the diffusion model, which attempts to predict the present noise. Assuming the diffusion model performs flawlessly, the discrepancies between the actual noise and the predicted noise would correspond to the areas in the image where the NeRF model is not working properly. Therefore, this is used as loss to train the network. For a more comprehensive visualization and understanding, one can refer to the complete diagram extracted from the original paper (Appendix Figure 8.2).

3.3.2 Loss for DreamFussion

The loss (see Equation 3.19) comes from the diffusion loss (see Equation 3.4) in which the term $\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}$ has been changed by $w(t)$ as it has been shown in [28] that the local minimum is achieved with the same parameters for any $w(t)$.

$$\mathcal{L}_{diff} = \mathbb{E}_{t\sim\mathcal{U}(0,1),\epsilon\sim\mathcal{N}(0,\mathrm{I}),\gamma}\left[w(t)\|\epsilon_\phi(z_t;t,y)-\epsilon\|_2^2\right] \tag{3.19}$$

The x_t is now called z_t as it is now obtained as $\alpha_t\, g(\theta, \gamma) + \sigma_t\, \epsilon$ instead of $\alpha_t\, x_0 + \sigma_t\, \epsilon$. Now, the start images x are rendered with $g(\theta, \gamma))$. γ represent the non-trainable camera poses while θ the trainable parameters of the NeRF network. Notice also that a y parameter has been added to the ϵ_θ network as now we have text prompts as seen in [31].
Our objective is to minimize this loss by changing the θ parameters of the NeRF network while maintaining the ϕ parameters of the diffusion networks constant. The following Equation (3.20) shows the gradient of the function respect the θ parameters.

$$\nabla_\theta \mathcal{L}_{diff}(\phi, \theta) = \mathbb{E}_{t,\epsilon,\gamma}\left[2\, w(t)(\epsilon_\phi(z_t; t, y) - \epsilon)\, \frac{\partial \epsilon_\phi(z_t; t, y)}{\partial z_t}\, \frac{\partial z_t}{\partial x}\, \frac{\partial x}{\partial \theta}\right] \tag{3.20}$$

Experimentally, it has been shown [12] that removing the second term $\frac{\partial \epsilon_\phi(z_t; t, y)}{z_t}$ produced much faster sampling without quality loss. This means that the gradient has to be manually specified in pytorch as in Equation 3.21:

$$\nabla_\theta \mathcal{L}_{diff}(\phi, \theta) = \mathbb{E}_{t,\epsilon,\gamma}\left[w(t)(\epsilon_\phi(z_t; t, y) - \epsilon)\, \frac{\partial x}{\partial \theta}\right] \tag{3.21}$$

Notice that $\frac{\partial z_t}{\partial x} = \alpha_t$ and the 2 has been absorbed by the $w(t)$ term.

3.3.3 Stable DreamFusion

The previous implementation [12] utilized Mip-NeRF [47] as the NeRF model and [48] as the diffusion model. However, both of these models faced limitations as they were neither publicly accessible, nor optimized for speed. To overcome these challenges, a more efficient and publicly available alternative was employed, as demonstrated in [2]. For the diffusion model, we adopted the *Stable Diffusion 2-1-base* [11], which offers improved performance. As for the NeRF model, we replaced it with a publicly implemented version of Instant-NGP [45], known as *torch-ngp*.

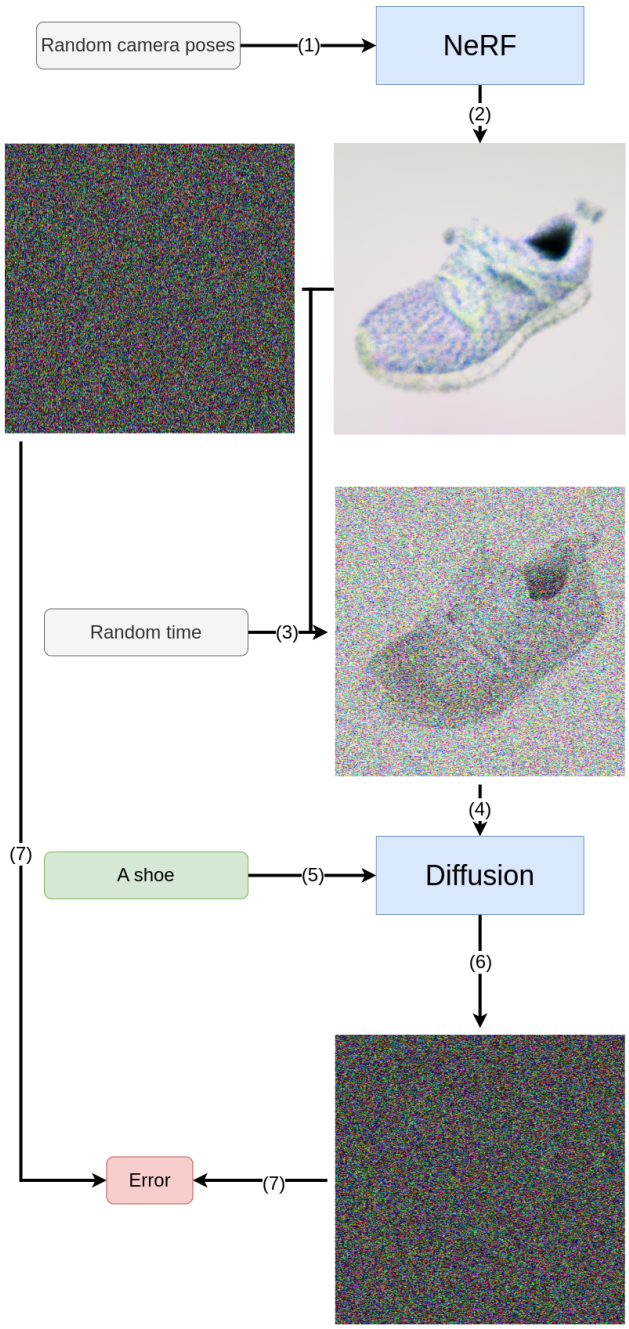

Figure 3.9: DreamFusion model pipeline. Gray rectangles are inputs that are automatically, randomly sampled. Blue rectangles are the used models. (1) Random camera poses are sampled and passed towards NeRF model. (2) NeRF model generates an image. (3) Noise is sampled from a Gaussian distribution and a random time t is sampled, both are used to noise the NeRF's image. (4) The noised image is passed towards the diffusion model that using the text prompt (5) will predict the noise. Finally, both the predicted and real noises (7) are compared to measure the error. This error will be back-propagated to update the NeRF's network parameters.

Chapter 4

Analysis: Mathematical Foundations and Challenges in Text-to-3D Generation

In this chapter, we will first present all the mathematical assumptions outlined in the initial section to establish a profound understanding of the model's behavior. Following that, we will perform an experimental analysis to examine the interrelationships among the various components of the model and identify the strengths and weaknesses associated with each. This analysis will lay the groundwork for our proposed methodologies in the next section.

4.1 Theoretical analysis

In this section, we will tackle the derivations of the previous formulations in order to explore in depth the performance of the Stable diffusion. These studies are usually not present in the papers and are mandatory to understand completely the algorithms and to justify the mathematical part of the Bachelor thesis. Some of the calculus is present in [25; 26]. However, none of those papers provide extensive formula derivations as done here.

4.1.1 Diffusion models derivations

One step forward process

Defining $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$, and being ϵ_t, ϵ_{t-1}, ϵ random variables with a Gaussian standard distribution, we can rewrite x_t in terms of x_0. The incorporation of this technique holds immense value as it allows for a substantial

reduction in the number of steps required to generate a noised sample. Specifically, the process that previously demanded 1000 steps can now be achieved with just a single step. This remarkable advancement directly translates into a significant decrease in the execution time of the algorithm.

$$\begin{aligned} x_t &= \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon_t = \sqrt{\alpha_t}\cdot(\sqrt{\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_{t-1}}\epsilon_{t-1}) + \sqrt{1-\alpha_t}\epsilon_t \\ &= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\epsilon_{t-1} + \sqrt{1-\alpha_t}\epsilon_t \qquad (4.1) \\ &= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1}) + (1-\alpha_t)}\epsilon \\ &= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\epsilon = \cdots \qquad (4.2) \\ &= \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon \end{aligned}$$

In this derivation, equality (4.1) comes from the *sum of two Gaussian* [1] while equality (4.2) is made by iteration.

Tractable lower bound

Our primary objective is to transform our loss function into a more manageable and computationally feasible form. To achieve this, we will undertake a systematic step-by-step process. Throughout this process, we will introduce several equalities with the purpose of utilizing them in subsequent equations, ultimately leading us to our desired outcome of an appealing loss function.

First let us rewrite the probability of generating x_0 as in Equation (4.3):

$$p_\theta(x_0) = \int p_\theta(x_{0:T})\, dx_1, \ldots, dx_T = \int p_\theta(x_{0:T}) \cdot \frac{q(x_{1:T}|x_0)}{q(x_{1:T}|x_0)}\, dx_1, \ldots, dx_T. \quad (4.3)$$

Now, we can use this to obtain our lower bound as done in equation (4.6). Notice that the expectation comes from the part that we want to minimize the log likelihood for all the images of the dataset. An image from the dataset has probability $\frac{1}{k}$ of being picked, $q(x_0) = \frac{1}{k}$, while an image outside the dataset has probability 0, being $k \in \mathbb{N}$ the number of elements in the dataset.

[1] https://en.wikipedia.org/wiki/Sum_of_normally_distributed_random_variables

$$\mathbb{E}[-\log p_\theta(x_0)] = \int -q(x_0) \cdot \log p_\theta(x_0) dx_0 \tag{4.4}$$

$$= \int -q(x_0) \cdot \log \left[\int p_\theta(x_{0:T}) \cdot \frac{q(x_{1:T}|x_0)}{q(x_{1:T}|x_0)} \, dx_1, \ldots, dx_T \right] dx_0 \tag{4.5}$$

$$\leq - \int q(x_{0:T}) \cdot \log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)} \, dx_0, \ldots, dx_T$$

$$= \mathbb{E}_q \left[-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)} \right] \tag{4.6}$$

Notice that equality (4.4) uses the previous expression obtained in Equation (4.3) while equality (4.5) comes from the *Jensen's inequality*[2]. Now, we can develop this expression even further, but first let us expand the numerator and denominator of the fraction:

$$p_\theta(x_{0:T}) = p_\theta(x_0|x_{1:T}) \cdot p_\theta(x_{1:T}) = p_\theta(x_0|x_1) \cdot p_\theta(x_1|x_{2:T}) \cdot p_\theta(x_{2:T}) = \cdots \tag{4.7}$$

$$= \left(\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \right) \cdot p_\theta(x_T) \tag{4.8}$$

$$q(x_{1:T}|x_0) = \frac{q(x_{0:T})}{q(x_0)} = \frac{q(x_T|x_{T-1:0}) \cdot q(x_{T-1:0})}{q(x_0)} = \frac{q(x_T|x_{T-1}) \cdot q(x_{T-1:0})}{q(x_0)} \tag{4.9}$$

$$= \cdots = \frac{\left(\prod_{t=1}^{T} q(x_t|x_{t-1}) \right) \cdot q(x_0)}{q(x_0)} = \prod_{t=1}^{T} q(x_t|x_{t-1}) \tag{4.10}$$

Notice that the equality at lines (4.7) and (4.9) come from the fact that q and p are Markov chains, so the next state only depends on the previous and consequently the steps before can be removed in the conditioning.

Now, we can plug the obtained results of this Equations (4.8) and (4.10) into the expression (4.6) as follows in the following derivation (4.12):

[2]https://en.wikipedia.org/wiki/Jensen%27s_inequality

$$\mathbb{E}_q\left[-\log\frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}\right] = \mathbb{E}_q\left[-\log\left(\prod_{t=1}^{T}\frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})}\cdot p_\theta(x_T)\right)\right] \tag{4.11}$$

$$= \mathbb{E}_q\left[-\sum_{t=1}^{T}\log\frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})} - \log p(x_T)\right] \tag{4.12}$$

Notice that equaltiy (4.11) comes from the fact that $p_\theta(x_T)$ follows a Gaussian standard distribution and does not depend on θ, so we can remove this term to showcase this observation. Let us develop two expressions more that will be used latterly: Equations (4.13) and (4.14):

$$\begin{aligned} q(x_t|x_{t-1}) = q(x_t|x_{t-1},x_0) &= \frac{q(x_t,x_{t-1},x_0)}{q(x_{t-1},x_0)} = \frac{q(x_{t-1}|x_t,x_0)\cdot q(x_t,x_0)}{q(x_{t-1},x_0)} \\ &= \frac{q(x_{t-1}|x_t,x_0)\cdot q(x_t|x_0)\cdot q(x_0)}{q(x_{t-1}|x_0)\cdot q(x_0)} = \frac{q(x_{t-1}|x_t,x_0)\cdot q(x_t|x_0)}{q(x_{t-1}|x_0)} \end{aligned} \tag{4.13}$$

$$\frac{q(x_1|x_0)}{q(x_2|x_0)}\cdot\frac{q(x_2|x_0)}{q(x_3|x_0)}\cdots\frac{q(x_{T-1}|x_0)}{q(x_T|x_0)} = \frac{q(x_1|x_0)}{q(x_T|x_0)} \tag{4.14}$$

Now, using these two expressions, we can develop the expression further as in Equation (4.17).

$$\mathbb{E}_q\left[-\sum_{t=1}^{T}\log\frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})} - \log p(x_T)\right]$$

$$= \mathbb{E}_q\left[-\sum_{t=2}^{T}\log\frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})} - \log p(x_T) - \frac{p_\theta(x_0|x_1)}{q(x_1|x_0)}\right] \tag{4.15}$$

$$= \mathbb{E}_q\left[-\log\prod_{t=2}^{T}\left(\frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t,x_0)}\cdot\frac{q(x_{t-1}|x_0)}{q(x_t|x_0)}\right) - \log p(x_T) - \log\frac{p_\theta(x_0|x_1)}{q(x_1|x_0)}\right] \tag{4.16}$$

$$= \mathbb{E}_q\left[-\log\prod_{t=2}^{T}\left(\frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t,x_0)}\right)\cdot\frac{q(x_1|x_0)}{q(x_T|x_0)} - \log p(x_T) - \log\frac{p_\theta(x_0|x_1)}{q(x_1|x_0)}\right]$$

$$= \mathbb{E}_q\left[-\sum_{t=2}^{T}\log\frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t,x_0)} - \log\frac{p(x_T)}{q(x_T|x_0)} - \log p_\theta(x_0|x_1)\right] \tag{4.17}$$

Notice that in equality (4.15), we use the expression obtained in Equation (4.13) and in equality (4.16) we use the expression obtained in Equation (4.14).

We see that our expression is very similar to the one desired except that we do not have the divergence yet. Let us take each term of the summation and convert it to D_{KL} divergence:

$$\begin{aligned}
&\mathbb{E}_q\left[\log\frac{q(x_{t-1}|x_t,x_0)}{p_\theta(x_{t-1}|x_t)}\right] = \int q(x_{0:T})\cdot\log\frac{q(x_{t-1}|x_t,x_0)}{p_\theta(x_{t-1}|x_t)}\,dx_0,\cdots,dx_T\\
&= \int\left(\int q(x_{0:T})\,dx_1,\cdots,d\hat{x}_{t-1},d\hat{x}_t,\cdots,dx_T\right)\cdot\log\frac{q(x_{t-1}|x_t,x_0)}{p_\theta(x_{t-1}|x_t)}\,dx_0,dx_{t-1},dx_t\\
&= \int q(x_0,x_{t-1},x_t)\cdot\log\frac{q(x_{t-1}|x_t,x_0)}{p_\theta(x_{t-1}|x_t)}\,dx_0,dx_{t-1},dx_t\\
&= \int q(x_{t-1}|x_t,x_0)\cdot q(x_t,x_0)\cdot\log\frac{q(x_{t-1}|x_t,x_0)}{p_\theta(x_{t-1}|x_t)}\,dx_0,dx_{t-1},dx_t\\
&= \int q(x_t,x_0)\cdot D_{KL}\left(q(x_{t-1}|x_t,x_0)\|p_\theta(x_{t-1}|x_t)\right)\,dx_0,dx_t\\
&= \int q(x_{0:T})\cdot D_{KL}\left(q(x_{t-1}|x_t,x_0)\,\|\,p_\theta(x_{t-1}|x_t)\right)dx_0,\cdots,dx_T\\
&= \mathbb{E}_q\left[D_{KL}\left(q(x_{t-1}|x_t,x_0)\,\|\,p_\theta(x_{t-1}|x_t)\right)\right] \qquad (4.18)
\end{aligned}$$

Now, with this last expression obtained in Equation (4.18), we can rewrite the Equation (4.17) to obtain our desired expression:

$$\mathbb{E}_q\left[D_{\mathrm{KL}}(q(x_T|x_0)\,\|\,p(x_T)) + \sum_{t>1}D_{\mathrm{KL}}(q(x_{t-1}|x_t,x_0)\,\|\,p_\theta(x_{t-1}|x_t)) - \log p_\theta(x_0|x_1)\right] \qquad (4.19)$$

Notice that the negative sign has vanished by changing the denominator with the numerator in the logarithm using $-\log\left(\frac{a}{b}\right) = \log\left(\frac{b}{a}\right)$.

Final L_t loss

In this final part, we want to expand our L_t terms until obtaining the final, manageable formulation. This final formulation will convert our loss into a weighted difference between the real and the expected noise. First of all let us analyse which is the value of $q(x_{t-1}|x_t,x_0)$:

$$
\begin{aligned}
q(x_{t-1}|x_t, x_0) &= \frac{q(x_{t-1}, x_t, x_0)}{q(x_t, x_0)} = \frac{q(x_t|x_{t-1}, x_0) \cdot \frac{q(x_{t-1}, x_0)}{q(x_0)}}{\frac{q(x_t, x_0)}{q(x_0)}} = \frac{q(x_t|x_{t-1}) \cdot q(x_{t-1}|x_0)}{q(x_t|x_0)} \\
&= \frac{\mathcal{N}(x_t; \sqrt{\alpha_t}x_{t-1}, 1-\alpha_t) \cdot \mathcal{N}(x_{t-1}; \sqrt{\bar{\alpha}_{t-1}}x_0, 1-\bar{\alpha}_{t-1})}{\mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, 1-\bar{\alpha}_t)} \\
&= \frac{\frac{e^{\frac{-1\cdot(x_t-\sqrt{\alpha_t}x_{t-1})^2}{2\cdot(1-\alpha_t)}}}{\sqrt{2\pi(1-\alpha_t)}} \cdot \frac{e^{\frac{-1\cdot(x_t-\sqrt{\bar{\alpha}_{t-1}}x_0)^2}{2\cdot(1-\bar{\alpha}_{t-1})}}}{\sqrt{2\pi(1-\bar{\alpha}_{t-1})}}}{\frac{e^{\frac{-1\cdot(x_t-\sqrt{\bar{\alpha}_t}x_0)^2}{2\cdot(1-\bar{\alpha}_t)}}}{\sqrt{2\pi(1-\bar{\alpha}_t)}}} = \frac{e^{\frac{-1\cdot\left(x_{t-1}-\left(\frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}x_0+\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t\right)\right)^2}{2\cdot\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t}}}{\sqrt{2\pi\left(\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t\right)}} \\
&= \mathcal{N}(x_{t-1}; \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t, \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t) \\
&= \mathcal{N}(x_{t-1}; \tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t) \qquad (4.20)
\end{aligned}
$$

The equality in the middle has been omitted due to its extension. It can be easily checked in any internet calculator that both expressions are equivalent. The step-by-step process begins with the isolation of the variance and the posterior reconstruction of the mean. We strongly do not recommend trying to do it. It is tedious and long work, and any small mistake can lead to a dead end. The important thing is that $q(x_{t-1}|x_t, x_0)$ follows a Gaussian distribution, and that $\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t$. Now, we can take Equation (4.19) L_t terms and, knowing the form of the D_{KL} *divergence between two Gaussians*[3], rewrite them as follows in Equation (4.22):

$$
\begin{aligned}
&\mathbb{E}_q\left[D_{\mathrm{KL}}(q(x_{t-1}|x_t, x_0) \,\|\, p_\theta(x_{t-1}|x_t))\right] \\
&= \mathbb{E}_q\left[\log\frac{\sqrt{\sigma_t^2}}{\sqrt{\tilde{\beta}_t}} + \frac{\tilde{\beta}_t + \|\tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t)\|^2}{2\sigma_t^2} - \frac{1}{2}\right] \\
&= \mathbb{E}_{x_0,\epsilon}\left[\frac{\|\tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t)\|^2}{2\sigma_t^2}\right] + C \qquad (4.21) \\
&= \mathbb{E}_{x_0,\epsilon}\left[\frac{1}{2\sigma_t^2}\left\|\frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right) - \mu_\theta(x_t, t)\right\|^2\right] + C. \qquad (4.22)
\end{aligned}
$$

Notice that we will be sampling the noised images x_t using the forward process previously developed: x_t as $\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$. This is the reason why the expectation in eq. (4.22) is done over x_0 and ϵ. We left x_t in the derivations as it is easier to work with and makes the formulas easier to visualise. The last

[3] https://scoste.fr/posts/dkl_gaussian/

expression (4.21) comes from rewriting of $\tilde{\mu}_t$ depending on x_t and ϵ instead of x_0 and x_t as follows in Equation (4.23), in which we use the previous obtained expressions at (4.20). Notice that we are writing x_0 in terms of x_t and ϵ by isolating it in the original forward Equation (3.1):

$$\begin{aligned}
&\tilde{\mu}(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\frac{x_t - \sqrt{1-\bar{\alpha}_t}\epsilon}{\sqrt{\bar{\alpha}_t}} + \\
&\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t = \frac{1}{\sqrt{\alpha_t}}\left(\frac{\beta_t x_t + \alpha_t(1-\bar{\alpha}_{t-1})x_t}{1-\bar{\alpha}_t} - \frac{\beta_t\epsilon}{\sqrt{1-\bar{\alpha}_t}}\right) \\
&= \frac{1}{\sqrt{\alpha_t}}\left(\frac{1-\alpha_t+\alpha_t-\bar{\alpha}_t}{1-\bar{\alpha}_t}x_t - \frac{\beta_t\epsilon}{\sqrt{1-\bar{\alpha}_t}}\right) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t\epsilon}{\sqrt{1-\bar{\alpha}_t}}\right)
\end{aligned} \tag{4.23}$$

So we see from Equation (4.23) a natural **choice** of μ_θ would be same as $\tilde{\mu}(x_t, x_0)$, but predicting x_0 by writing it in terms of x_t and ϵ_θ (see Equation (4.24)). The last equality is derived similarly as in Equation 4.23, but using instead these terms:

$$\mu_\theta(x_t, t) := \tilde{\mu}_t\left(x_t, \frac{x_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta(x_t, t)}{\sqrt{\bar{\alpha}_t}}\right) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) \tag{4.24}$$

We can now finally rewrite the L_t terms as in Equation (4.25):

$$\begin{aligned}
&\mathbb{E}_{x_0,\epsilon}\left[D_{\mathrm{KL}}(q(x_{t-1}|x_t, x_0) \,\|\, p_\theta(x_{t-1}|x_t))\right] = \\
&= \mathbb{E}_q\left[\frac{1}{2\sigma_t^2}\left\|\frac{1}{\sqrt{\sigma_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right) - \mu_\theta(x_t, t)\right\|^2\right] \\
&= \mathbb{E}_{x_0,\epsilon}\left[\frac{1}{2\sigma_t^2}\left\|\frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right) - \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right)\right\|^2\right] \\
&= \mathbb{E}_{x_0,\epsilon}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}\left\|(\epsilon - \epsilon_\theta(x_t, t))\right\|^2\right]
\end{aligned} \tag{4.25}$$

At the end, we see that our expression resumes to predicting the added noise given the noised image(x_t) and the time-step t.

4.1.2 NeRF model derivations

Transmittance

Let us analyse how the transmittance formula is obtained. We first have that $P[\text{no hits before t}] = T(t)$ and $P[\text{hit at t}] = \sigma(t)dt$ and they both can be related

as $T(t+dt) = T(t) \cdot (1 - \sigma(t)dt)$. Now, we can use the Taylor expansion at $dt = 0$ as in Equation (4.27):

$$\begin{aligned} & T(t) + T'(t)dt = T(t) - T(t)\sigma(t)dt \\ & \Rightarrow \frac{T'(t)}{T(t)}dt = -\sigma(t)dt \qquad (4.26) \\ & \Rightarrow \log T(t) = -\int_{t_0}^{t} \sigma(s)ds \\ & \Rightarrow T(t) = \exp\left(-\int_{t_0}^{t} \sigma(s)\, ds\right) \qquad (4.27) \end{aligned}$$

Notice that in expression (4.26), we have $log(T(t))$ and $-log(T(t_0))$ after applying the integral. By definition t_0 is the beginning time so $T(t_0) = 1$ and consequently $log(1) = 0$ and the term is ignored.

Finite raymarching

The objective is to break the integral into sum of tractable integrals. Being t_i, $i \in \{1, \cdots, n\}$ the sampled times and t_{n+1} the time when the ray intersects with the far plane, we have $\int T(t)\sigma(t)\mathrm{c}(\mathrm{t})\, dt \approx \sum_{i=1}^{n} \int_{t_i}^{t_{i+1}} T(t)\sigma_i \mathrm{c}_i$. Notice that in this expression between each sampled time, we suppose that colour and density are constant while, as we have seen by definition, $T(t)$ still has to depend on time t. Now, for $t \in [t_i, t_{i+1}]$, we have that:

$$\begin{aligned} T(t) &= \exp\left(-\int_{t_0}^{t} \sigma(s)\, ds\right) = \exp\left(-\int_{t_0}^{t_i} \sigma(s)\, ds\right) \cdot \exp\left(-\int_{t_i}^{t} \sigma(s)\, ds\right) \\ &= \exp\left[-\sum_{j=1}^{i-1} \sigma_j \cdot (t_{j+1} - t_j)\right] \cdot \exp\left(-\int_{t_i}^{t} \sigma_i\, ds\right) = T_i \exp(-\sigma_i(t - t_i)). \end{aligned} \qquad (4.28)$$

One can note how the definition of $T_i := \exp\left[-\sum_{j=1}^{i-1} \sigma_j \cdot \delta_j\right]$, where $\delta_j := (t_{j+1} - t_j)$, comes from the Equation (4.28) above. We can plug this obtained term at the previous equation obtaining the final (4.29) equation:

$$\int_{t_i}^{t_f} T(t)\sigma(t)\mathrm{c}(\mathrm{t})\,dt \approx \sum_{i=1}^{n} \int_{t_i}^{t_{i+1}} T(t)\sigma_i \mathrm{c}_i\,dt = \sum_{i=1}^{n} \int_{t_i}^{t_{i+1}} T_i \exp(-\sigma_i(t-t_i))\sigma_i \mathrm{c}_i\,dt$$
$$= \sum_{i=1}^{n} T_i\sigma_i \mathrm{c}_i \int_{t_i}^{t_{i+1}} \exp(-\sigma_i(t-t_i))\,dt = \sum_{i=1}^{n} T_i\sigma_i \mathrm{c}_i \cdot \frac{\exp(\sigma_i(t_{i+1}-t_i))-1}{-\sigma_i}$$
$$= \sum_{i=1}^{n} T_i \mathrm{c}_i(1-\exp(-\sigma_i\delta_i)) \tag{4.29}$$

This is the final formula used in NeRF to calculate pixel colours for a given ray.

4.2 Problem analysis

Due to the complexity of the used algorithm, we used public base code available on Github, called Stable Dreamfusion [2], see Section 3.3.3, instead of working directly from scratch, which would be probably impossible in the given amount of time. This decision was also impulsed by the availability and low-resource demand of the diffusion and NeRF models used in this implementation.

4.2.1 Low-Quality Output

Individually, both models, NeRF and Stable Diffusion have demonstrated exceptional performance and accuracy. However, when used in conjunction, their performance suffers. This is present on the original paper and even more on our used version of it. One of the objectives of this project is to investigate the behavior of the combined model, understand how each component relates to the others, and identify the reasons why its performance is suboptimal. By doing so, we aim to propose improvements to enhance the model's performance. Figure 4.1 includes examples of several generations of the model, which reveal that while the semantic generation is functioning correctly, there is a lack of quality and detail in the generated images, which needs to be addressed. For comparison purposes, see Appendix Figure 8.3 to see how the Stable Diffusion model is capable of generating quality and detailed samples.

4.2.2 Artifacts

The objective of the stable diffusion generated images is to look realistic, but it does not contain the behaviour of the objects that appear on the scenes. This makes the generated image look realistic at first glance, but then you can

A snowflake	A car	A piano	A tree

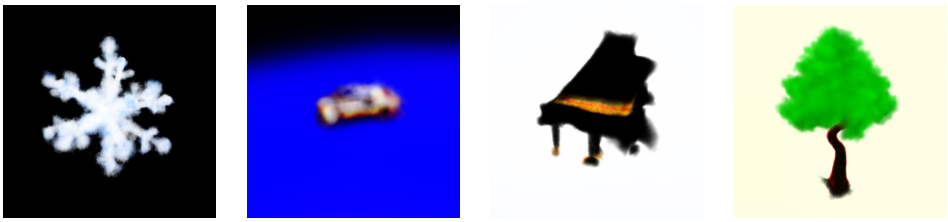

Figure 4.1: Examples of renderings done by Stable Dreamfusion model [2]. Please note the deficiency in quality and level of detail exhibited in the images.

notice strange parts on it. These artifacts usually are related with the number of components of an object: fingers on a hand, legs of an animal (see Figure 4.2 for an illustrative example) or wheels of a car, and also the disposition of the different components among the picture that might no match exactly with the expected ones. It is important to notice that these artifacts can be perpetuated to our 3d models although it is outside the scope of this study to minimize them, but it is important to have these limitations in mind.

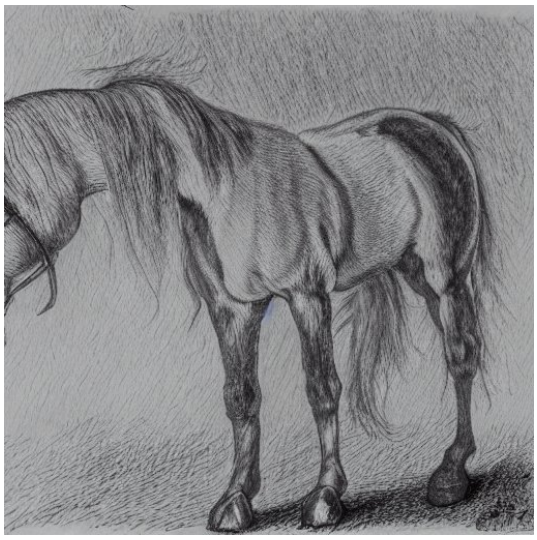

Figure 4.2: Stable Diffusion [11] generated image with the text prompt: "a horse". We can observe that the model struggles with the number of hind limbs.

4.2.3 Janus problem

This phenomenon is referred to as the Janus problem, named after the Roman god Janus, who is depicted with two faces. It occurs when a 3D model has

multiple faces, resulting in ambiguous or inconsistent renderings. An example of this is shown in Figure 4.3. This issue may arise from the bias of the diffusion model towards generating certain perspectives or faces, which is likely due to the dataset used to train the model [49], that was collected from the internet. The dataset, as in the case of a cat, might contain mostly front-facing images, or in the case of a house, images with the door visible, which results in the model producing similar perspectives. Figure 4.3 illustrates this behaviour. It is worth noting that the loss function used to train the NeRF model, shown previously in Equation (3.19), does not enforce consistency across different renderings. Consequently, it is possible to generate multiple renderings that are identical. To mitigate this issue, input text labels such as top, front, back, and side view are used to specify the desired rendering viewpoint. However, this problem persists and further research is needed to address it.

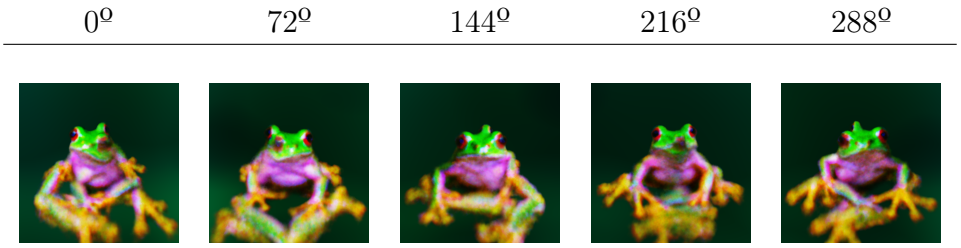

Figure 4.3: Janus multi-face problem with the input "A frog" rendered from different perspectives.

4.2.4 Mode collapse

One interesting observation when rendering multiple models from the same input is the striking similarity between them. In contrast to the Stable Diffusion model, which produces different images for the same input, our model generates almost identical 3D reconstruction, no matter the used random seed. The reason for this phenomenon is not yet fully understood, but it could be related to the mode-seeking nature of the loss function, as mentioned in [12]. Since the model needs to generate a plausible image for every perspective, it might ignore distinctive features of a particular sample and instead generate a class-looking reconstruction that is consistent across all perspectives. Therefore, the unique features that are preserved in the final generation must be specific to the input text. Figure 4.4 shows 4 rendering of 4 3D models which look surprisingly similar considering the stochastic nature of the underlying model.

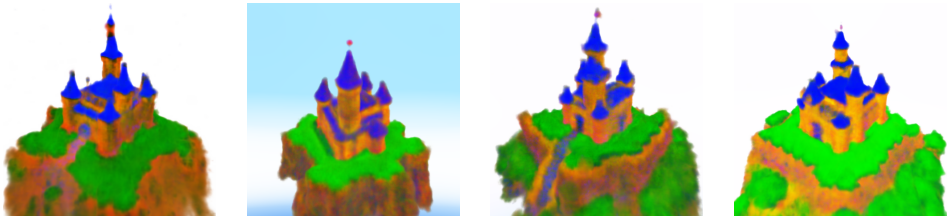

Figure 4.4: Rendered images from various reconstruction attempts using [2] with text prompt "A castle". We can observe that the samples exhibit strikingly similar characteristics. This phenomenon is commonly referred to as mode collapse.

4.3 Component analysis

In this section, we will conduct a comprehensive analysis of the model's most significant components to gain a deeper understanding of their interrelation and behavior. By examining these components, we aim to establish a solid foundation for comprehending the insights and behavior of the model.

4.3.1 Epochs

In our experiments, we investigated the impact of increasing the number of epochs on the quality and level of detail in the generated results. We realized that at the first stages the model tends to generate the objects contour while in the last it focuses on the colour and details. During the training process, we observed that around 100 epochs the model's ability to generate additional details on the object seemed to stop.

4.3.2 Image size

Image size is a critical factor in determining the performance of our model. While NeRF, which calculates loss per ray or per pixel, is not directly affected by image quality, the diffusion model, which serves as a loss for the NeRF model, might be influenced by it.

Notice that in contrast to the original paper [12], our model utilizes the Stable Diffusion technique, which operates with a larger image size of 512. It is important to note that if the NeRF model produces lower-resolution images, being 64 the base image size, the diffusion model may not fully capture all the intricate details present in the 3D reconstruction. Consequently, the model may not fully capture the subtle nuances and complexities of the reconstructed objects, resulting in suboptimal performance during training.

4.3.3 Guidance

Analyzing the role of guidance is crucial in our model, as it serves as a key factor in the diffusion process for generating desired samples. It has been shown in previous works [31] that using values of 5-10 instead of only text prompt leads to better generations of the diffusion models. However, it is worth noting that the guidance values used in our model are significantly higher compared to those typically used in the diffusion image generation process, with 100 being the default and most commonly employed value.

Our initial objective is to examine the behavior of the guidance values at a specific stage of the diffusion generative process. Isolating x_0 as in Equation (4.30), derived directly from Equation (3.2), allows us to visualize the impact of the guidance:

$$x_0 = \frac{(x_t - \sqrt{1 - \bar{\alpha}_t}\epsilon)}{\bar{\alpha}_t} \tag{4.30}$$

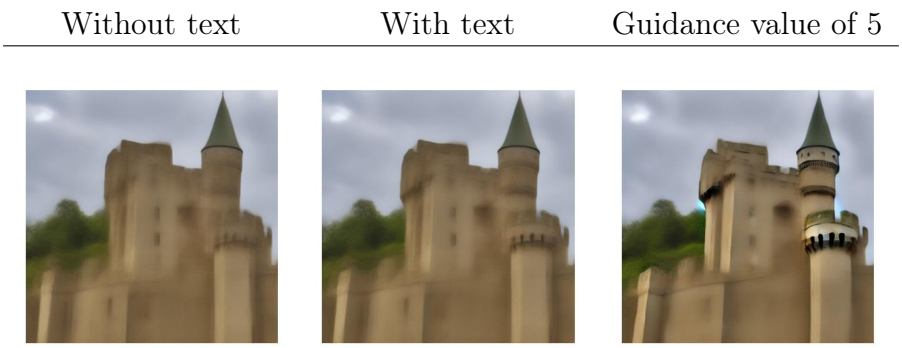

Figure 4.5: Text and guidance effect on stable diffusion generated samples with text input "A castle". First column shows the regenerated sample without any text prompt. Second column shows the regenerated sample predicting the noise with the text prompt. Third column shows the regenerated sample using guidance value of 5. We can observe how the last column enhances the level of detail in the image.

Figure 4.5 provides a visual example of how the guidance values capture important details and contours in the generated images.

When generating complete images using different guidance values, we observe two distinct patterns. Lower guidance values tend to include the object mentioned in the prompt, but it may not be the central focus of the image. On the other hand, higher guidance values tend to produce sketchy representations

of the object, lacking in specific details, but closely aligned with the intended concept of the prompt. This phenomenon is illustrated in Figure 4.6.

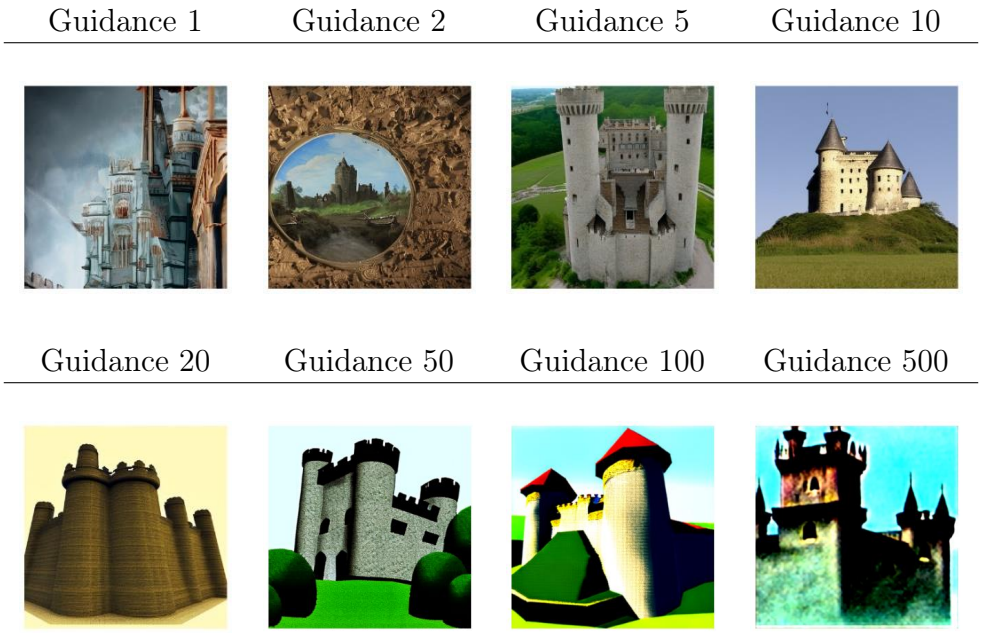

Figure 4.6: Generated images with Stable Diffusion using the text prompt "A castle". This example shows how with low guidance values the model uses the text prompt more like a context, while with higher guidance values in captures strictly the prompt idea without further details.

Upon careful examination of the diffusion generative process, we have observed that it exhibits remarkable noise removal capabilities. It exhibits an impressive capability to regenerate highly resembling images even from noised versions that would pose a significant challenge for human observers. We noticed that the model's reliance on the text prompt during the generative process varies depending on the noise level in the input image. With low-noised images, the model tends to rely less on the text prompt and more on the visual information in the image itself. However, with higher noise levels, the model relies more heavily on the text prompt for guidance.

This observation intrigued us, particularly regarding the initial structure of a gray-centered bubble in the NeRF model. This bubble is created by manually adding density, that is part of the model itself, to enhance the model's performance. When starting with the noise produced by the application of the forward diffusion process to the image, and applying the backward diffusion

process with our model and a text prompt, we noticed that the base image has a strong influence on the generated samples. This suggests that the model is highly biased towards the initial noised image. Consequently, when using Stable Diffusion for 3D reconstruction, higher guidance values may be necessary to rely more on the text prompt, and not in the default generated renderings. Figure 4.7 showcases the influence of this initial image on the posterior generations.

Figure 4.7: The initial image displays a randomly weighted NeRF model rendering. The formation of the bubble arises from the manual incorporation of density to enhance NeRF model performance. The succeeding columns showcase different generations produced by applying the backward diffusion process with the prompt "A castle" to a completely noised version of the bubble.

4.3.4 Other component analysis

In addition to our main component analysis, we investigated other components as well. One noteworthy was the generated model's opacity. Previous works [2; 12] used an opacity loss that came from the aggregation of densities over the rays. Interestingly, we found that using the default values for opacity loss led to empty image generations. However, when we increased the guidance values to 250 or higher, the model was able to generate the samples. Our initial expectation was that by augmenting the opacity loss, we would reduce unnecessary densities and consequently generate images with more defined borders and less blurriness. However, we observed that the model generated smaller versions of the objects instead. Appendix Figure 4.8 presents the results of this phenomenon.

Additionally, when examining other components such as time, we discovered that employing high noise values with time steps beyond 700 often led to suboptimal reconstructions, which were included in the training of the NeRF model. It is worth noting that these peculiar generations represented only a small fraction of the overall outputs produced by the Stable Diffusion model and can be regarded as noise within the dataset. Interestingly, the undesired images frequently depicted humans, cats, or text, very common and standard

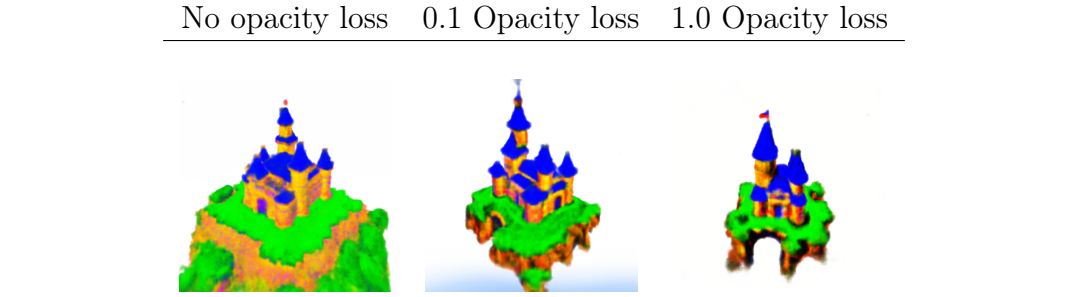

Figure 4.8: Generated-rendered images with different opacity values and guidance of 250. The text prompt used to generate these results was "A castle". Opacity loss penalized the sum of the overall density values over a ray. We see that higher opacity loss produce smaller objects.

objects. This behavior is visually shown in Figure 4.9.

Figure 4.9: This images correspond to generations of the Stable Diffusion model from a NeRF rendering which has been noised. Notice the presence of unusual images that reveal some of the challenges the model faces when reconstructing at higher time steps (t).

Finally, it is noteworthy to mention that we conducted an analysis of various other components including the angles used in the NeRF camera poses, the weight value of the diffusion loss $w(t)$, camera aperture and radius, and the text descriptions used to define the point of view. However, these investigations did not yield any significant insights or findings of note.

Chapter 5

Methodology for Image to 3D reconstruction and Enhanced text to 3D reconstruction

In this section, we present two novel techinques: FitFusion and DreamText. Our first proposal, FitFusion is a fine-tuning approach for Stable DreamFusion, aimed at enhancing the quality of generated 3D reconstructions reaching the expected outputs of NeRF and Stable Diffusion. Additionally, we introduce a second model, we call DreamText, a method specifically designed for image-to-3D generation. Both techniques contribute to advancing the field of 3D sample synthesis by leveraging the power of StableDiffusion and introducing innovative capabilities.

5.1 FitFusion

Stable DreamFusion [2] currently produces samples that fall short in comparison to its foundational models, Stable Diffusion [11] and NeRF [13]. To address this issue, we propose in this section fine-tuning certain network parameters that we have identified as essential in the previous chapter. Through this approach, we aim to improve the quality of the generated samples, bridging the gap between the base model and its superior foundational models.

5.1.1 Initial Approaches

We conducted a thorough exploration of time and guidance, two critical components of Stable Diffusion that were not fully explored in the context of 3D sample generation [2].

Time

Initially, we sought to constrain the time values used to add noise to the NeRF-rendered images. As mentioned in Section 3.3, the diffusion model operates as a loss prior, but it does not impose any mathematical requirements on the specific range of time values to be used. The network objective is to remove noise, with the difference between the generated and ground truth data serving as the NeRF model error. With the aim of enhancing the model accuracy, we conducted experiments involving different time values.

Our initial observations revealed that the network generated unfavorable predictions when high time values were employed, while low values led to no training progress as the loss was predicted almost perfectly. In response, we attempted to reduce the time range from the original 20-980 to narrower intervals, such as 100-800 and 200-700. Unfortunately, these adjustments did not yield significant improvements over the original renderings, but even to worse generations.

Subsequently, we conducted numerous experiments and formulated several hypotheses regarding the effects of time values on the model output, in order to have a deeper understanding of the model behaviour. Specifically, we observed that high values appeared to introduce new details, artifacts, and image components, while low values tended to smooth existing features. We speculate that this behavior stems from the model inclination to downplay the influence of text input at low time values, while relying on it more prominently at high values.

Finally, drawing inspiration from the work of [50], we attempted to implement a similar time schedule with the goal of generating new details using high time values, followed by smoothing with lower values, and repeating the process. Regrettably, this approach did not yield improvement in the output quality of our model. For illustrative purposes, Appendix Figure 8.4 showcases some of the failed results obtained when using the text prompt "An image of a castle."

Guidance

In addition to exploring time values, we also extensively examined the guidance parameter, used to drive the image generations to desired outputs. We conducted experiments with both higher and lower values to assess their impact on the quality of the generated 3D reconstructions. Higher values tended to introduce more details, but often resulted in poor overall reconstructions, with artifact like noise clouds present in most of it parts and with high contrast

colours. Meanwhile, lower values produced smoother results, but potentially missed important details. We also attempted to implement a cosine guidance schedule inspired by previous work, but unfortunately, it did not lead to noticeable improvements. Figure 8.5 in the Appendix serves as a visual representation of these failure cases, providing evidence for the aforementioned observations.

5.1.2 Final proposal

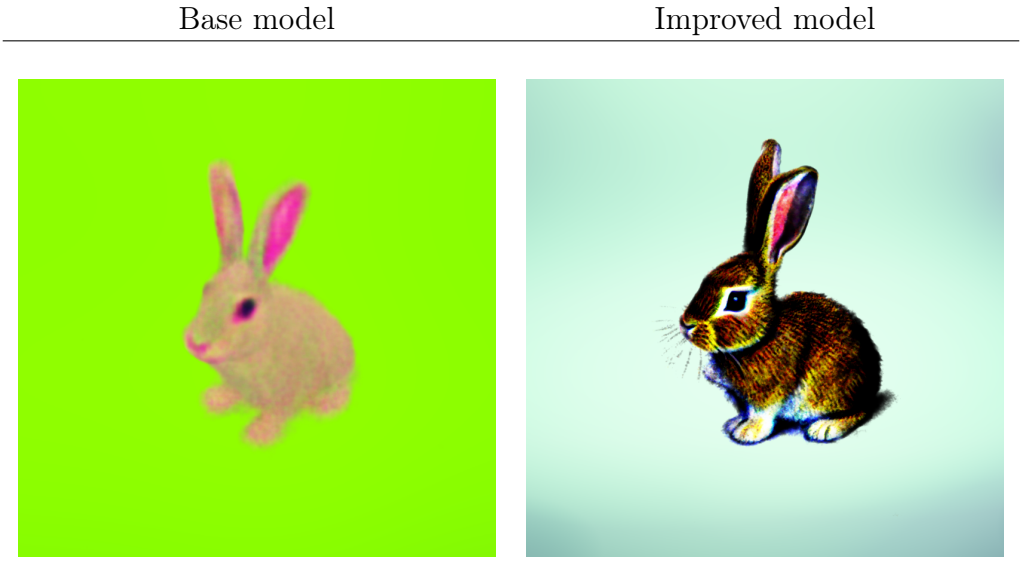

Figure 5.1: Rendered images from a reconstruction using the text prompt "An image of a rabbit". On the left, the rendering from Stable DreamFusion, while on the right, our FitFusion method, which clearly outperforms the previous one.

Our hypothesis is that combining larger images with higher guidance values significantly improves the output of the model, resulting in the highest quality generations, we had produced thus far. However, the model requires a greater number of training epochs to fully express these new details. It is worth noting that we used a 256x256 image size as it was the largest that could fit within our GPU, but we expect that using a size of 512x512 would yield even better results. The guidance proposed value is 1000, in contrast with the original value of 100. Of course, different objects may require different guidance values for optimal reconstruction. For complex objects with many parts or intricate details, lower values such as 800 lead better results, while objects that are relatively easier to reconstruct may benefit from higher guidance values of 1200. Notice that with bigger image size, higher guidance values should produce even

better results. Equation (5.1) shows the incorporation of the recommended guidance and image values in the base algorithm [2]:

$$\hat{\epsilon}_\theta(x_{t256}|y) = \epsilon_\theta(x_{t256}|y) + 1000 \cdot (\epsilon_\theta(x_{t256}|y) - \epsilon_\theta(x_{t256}|\emptyset)) \tag{5.1}$$

Figure 5.1 shows the difference between the base generation and our improved proposal. The guidance value of 1000 pushes the noise prediction to be more strongly conditioned on the text input. Simultaneously, the higher image quality is introduced through the x_{t256} samples, so the diffusion model is able to visualize more details of the renderings before predicting the noise.
Table 5.1 shows the value difference between both models.

Method	**Image Size**	**Guidance**	**Epochs**
Stable DreamFusion	64	100	100
FitFusion	256	1000	200

Table 5.1: Comparison of Stable DreamFusion and FitFusion model values.

5.1.3 Limitations

Despite the proposed enhancements, it is important to acknowledge that certain limitations from the original approach can still persist. Specifically, the Janus problem, characterized by the rendering of multiple faces, remains unresolved. It is important to note that the multi-face problem is observed within the initial 25 epochs of execution. One potential mitigation strategy for the multi-face problem is to halt and restart the model from scratch using different random seeds.

5.2 DreamText

We aim to address the challenge of reconstructing a 3D model from a single input image by developing a robust sentence descriptor. Our goal is to create a descriptor that can capture the unique characteristics of the object instance while maintaining sufficient generalization. Leveraging this powerful sentence representation, we will incorporate it as a conditioning factor in the diffusion model to guide the generation of high-quality 3D reconstructions. Our method can be divided in 3 parts: sentence selection, descriptor training and 3D-reconstruction.
The proposed method consists of the following steps:

- (1) The input image is encoded using a pre-trained CLIP image encoder.

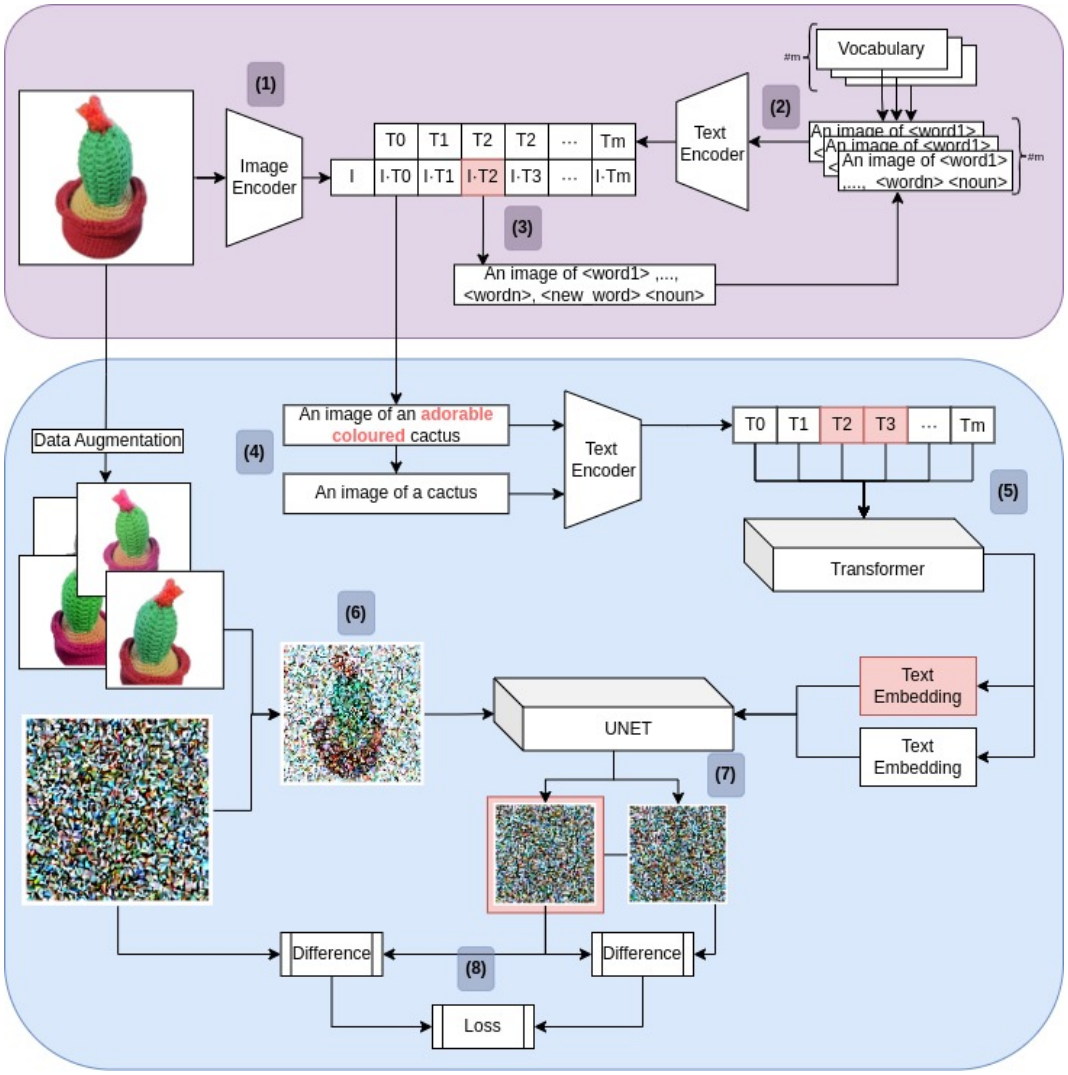

Figure 5.2: Summary of our approach: First, CLIP is used as zero-shot classifier to obtain a sentence embedding that has high similarity score with the image embedding. Later, part of this sentence is trained to describe more precisely the image embedding, using Stable Diffusion as a loss.

- (2) Vocabulary words are incrementally added to the starting sentence "An image of ", and each resulting sentence is encoded using CLIP's text encoder.
- (3) Similarity scores are calculated using dot product, and the best word is added to the sentence. This process is repeated from step 2 until the similarity score no longer improves.

- (4) The sentence is duplicated, with one version containing descriptive nouns and adjectives, and the other containing only the first found noun.
- (5) The embeddings of the descriptors are set as trainable.
- (6) Augmentations of the input image are noised.
- (7) The diffusion UNET predicts the noise given the complete sentence (with trainable parameters) and the basic sentence.
- (8) The difference between the predicted complete noise and the real and basic predicted noise is used as loss, which is backpropagated to train the descriptor embeddings.

5.2.1 Sentence selection

In the first step of our process, we focus on sentence building, aiming to automatically extract a suitable generic sentence that describes the input image. To achieve this, we leverage the image classification capabilities of CLIP. Initially, we encode the input image using CLIP's image encoder (see Figure 5.2 (1)). We then construct the evaluated sentences, starting with the base sentence "An image of" and appending words. We enforce the first appended word, at step 0, to be a noun. Meanwhile, at step n the sentence would be "An image of <$word_1$>, <$word_2$>, $\cdots$, <$word_n$>, <noun>" (see Figure 5.2 (2)). Next, the sentence is encoded using CLIP's text encoder, and the cosine similarity between the text and the image encodings is calculated. If the cosine similarity exceeds the previous step similarity, the corresponding word is added to the sentence (see Figure 5.2 (3)). If not, the existing sentence is passed to the next step. This process of sentence selection provides a solid foundation sentence that can be further trained to accurately describe the object depicted in the image.

5.2.2 Sentence training

In this step, our objective is to train the sentence to accurately describe the object without incorporating specific image characteristics such as disposition, pose, lighting, or view. To achieve this, we introduce a second sentence that solely contains the noun, "An image of a <noun>" (see Figure 5.2 (4)). Both the complete sentence and the noun sentence are encoded using CLIP's textual encoder. Notably, the embeddings of the tokens present in the complete sentence (all words except the noun obtained in the first step) are set as trainable (see Figure 5.2 (5)). These textual embeddings serve as conditioning inputs to

the Stable Diffusion UNET, which also receives noised images of augmented versions of the original image (see Figure 5.2 (6)). The augmented versions of the input image are essential to ensure that the model captures the characteristics of the object itself rather than specific details of the image. Then, the UNET predicts two different noises: one obtained conditioned with the embedding of the base sentence τ_b that remains unchanged, and the other with the embedding of the trainable sentence $\tau_\theta(y)$, (see Figure 5.2 (7)). The difference between the predicted noise using the trainable sentence and the actual noise ensures that the sentence learns to describe the specific object instance accurately. The difference with the predicted noise using the base sentence prevents loosing the general object class meaning and helps mitigate over-fitting to the image. This second loss can be viewed as a regularization term, and both losses together constitute our overall loss (see Figure 5.2 (8)). Equation (5.2) shows the first loss, Equation (5.3) shows the second loss. Equation (5.4) shows the combination of both losses, which is used to obtain the best trainable token embeddings, corresponding to the θ parameter in the equations:

$$\mathcal{L}_{diff_{real}} = \mathbb{E}_{t\sim\mathcal{U}(0,1),\epsilon\sim\mathcal{N}(0,\mathrm{I})}\left[\|\epsilon_\phi(\alpha_t \mathrm{x} + \sigma_t\epsilon, t, \tau_\theta(y)) - \epsilon\|_2^2\right] \tag{5.2}$$

$$\mathcal{L}_{diff_{base}} = \mathbb{E}_{t\sim\mathcal{U}(0,1),\epsilon\sim\mathcal{N}(0,\mathrm{I})}\left[\|\epsilon_\phi(\alpha_t \mathrm{x} + \sigma_t\epsilon, t, \tau_\theta(y)) - \epsilon_\phi(\alpha_t \mathrm{x} + \sigma_t\epsilon, t, \tau_b)\|_2^2\right] \tag{5.3}$$

$$\mathcal{L}_{diff_{base}} = \mathcal{L}_{diff_{real}} + \mathcal{L}_{diff_{base}} \tag{5.4}$$

5.2.3 3D reconstruction

The 3D reconstruction is made as in [2]. First of all, the trained embeddings are loaded in the text embeding model. Then the previously obtained sentences are passed to the Stable Diffusion model as the prompt. This will enable Stable Diffusion to incorporate influences from our trained textual descriptor inducing the NeRF model to produce a 3D reconstruction of the object present in our image.

5.3 Diversity augmentation

As discussed in the analysis section, prior works such as [12] and [2] encountered the challenge of mode collapse or low diversity in the generated images. To address this problem, we devised a direct solution. Our approach involves two steps:

First, we utilize Stable Diffusion, as demonstrated in [11], to generate an image based on the provided text prompt. Stable Diffusion is known for its ability to generate diverse samples, avoiding the issue of producing the same object, repeatedly.

Next, we leverage this generated image as input to our Image to 3D model, incorporating the mechanism described earlier. By doing so, we effectively make the prompt more specific to a concrete and detailed instance, thus providing the 3D reconstruction model with rich and specific information.

It is important to note that the lack of diversity in prior works stemmed from the model tendency to generate generic class objects that fit the given prompt. With our method, we address this issue by making the prompt more specific, resulting in a more diverse set of generated 3D reconstructions. Figure 5.3 showcases how we can leverage our proposal to make more diverse generations.

Figure 5.3: First image corresponds to the generated image with Stable Diffusion using the text prompt "An image of a castle". Notice how the model fits this new image and how finally the generation gains more diversity compared to the ones obtained directly with the text prompt as seen in Figure 4.4. See *GIF*.

Chapter 6

Experiments

In this section, we present an overview of the experiments conducted to validate our proposals. First we introduce the experimental setup in Section 6.1 in which we introduce the used dataset for the experiments, the pretrained models, the model architecture and some used values specifications. Then, we compare present a set of generations for each of the proposed models: FitFusion and DreamText. In the first case these are compared with the preexisting baseline, while in the second case they are compared with a state of the art model called RealFusion [1]. For the DreamText model we also include previous versions that lead to the final proposal.

6.1 Implementation Details

Most of the implementation details follow [12]. Other are slight variations introduced in our implementation or come from [2].
The graphic card used for the experiments is a 12GB GeForce RTX 3060. We had to use half precision floating point in our implementation to fit the models in our graphic card. Furthermore, we were not able to use the whole 512x512 image size as the GPU was not capable of fitting it. We anticipated a significant improvement when utilizing larger GPUs capable of accommodating full floating points and processing full-sized images.

6.1.1 Dataset

In the experimental section on DreamText (Section 6.3), we employed a dataset shown in Figure 6.1 that encompasses a variety of image samples. The dataset comprises five castle images in the first row, serving as a basis for comparing how the model describes their intricate details. In the second row, we included

five everyday objects, a potential model usage for the population. Additionally, the dataset incorporates two rows of examples sourced from [1], used for comparison purposes.

The vocabularies for nouns and adjectives used in the DreamText project, are included with the project code. These vocabularies serve as a predefined set of words specifically selected for generating text descriptions in the context of the project.

Figure 6.1: Images utilized as image conditioning for text regeneration in the subsequent section. The first two rows consist of new images used during this dissertation, meanwhile, the other two rows feature images from preexisting works.

6.1.2 Pre-trained models

Our work is based on pre-trained state of the art models. For the diffusion model, we use *Stable Diffusion 2-1-base* [11]. For the NeRF model, we use a

public implementation of Instant-NGP [45], called *torch-ngp*[1]. Notice that the base code for the model comes from a forked version at March 2023, from a *Stable-Dreamfusion*[2] [2].

6.1.3 NeRF Architecture

Out NeRF architecture consists on two MLP, one that predicts the object itself and the other used to predict the background. The first consists on 4 Residual blocks [51] with 96 neurons and a SiLU [52] activation function, while the second consists on 2 linear layers of 64 neurons with ReLU activation function. The multi-resolution hash encoding contains 6 resolution layers.

6.1.4 Shading

Three different types of shading were considered in our experiments: Lambertian, Albedo, and Textureless. For the first 10 epochs, Albedo shading is exclusively used. However, starting from epoch 11, the shading type is selected randomly with a probability of 0.2 for Albedo, 0.4 for Textureless, and 0.4 for Lambertian. Textureless shading, which is essentially Lambertian shading with all colors set to white, was introduced to promote the generation of 3D reconstructions rather than flat images. Light is sampled following Gaussian noise around ray origin, in order to avoid looking at dark faces.

6.1.5 Camera poses

The camera in our experiments is oriented towards the origin, where the object is centered. The radius of the camera position is selected uniformly from the interval [1.0, 1.5]. For 0.5 of the samples, the position vector components are uniformly selected from the interval [-1, 1], except the y component that is selected from [0,1]. Then the vector is normalized, and multiplied by the radius. For the remaining 0.5 of the samples, the horizontal and vertical angles are uniformly selected from the intervals [0, 120] and [0, 360], respectively. The horizontal angle sampling is limited to 90 degrees above the horizontal and 30 degrees below.

6.1.6 Density bias

A density blob is placed surrounding the origin. The value of the default density for a point $p = (x, y, z)$ can be obtained with the Equation (6.1):

[1] https://github.com/ashawkey/torch-ngp
[2] https://github.com/ashawkey/stable-dreamfusion

$$d = v \cdot \left(1 - \frac{\sqrt{x^2 + y^2 + z^2}}{b}\right) \tag{6.1}$$

where v is the value of the density, 10 in our case, and b is the blob radius, 0.5 in our case. The equation value is added to the predicted density of the network before applying the Sigmoid function, making a strong bias at the beginning towards a density blob in the center of coordinates.

6.1.7 View-Dependent Prompt

To prevent the Janus Problem and encourage the generation of diverse object views, view positions are incorporated into the text prompts. The views include overhead (elevation angle ranging from 0 to 30 degrees), bottom (elevation angle ranging from 150 to 180 degrees), front (azimuth angle ranging from 0 to 60 degrees), back (azimuth angle ranging from 180 to 240 degrees), and side views. Notice that overhead and bottom override the other views.

6.1.8 Execution time

FitFusion requires approximately 160 minutes of GPU time to complete the 3D reconstruction process. On the other hand, DreamText involves around 2 hours for training the text descriptor and an additional 80 minutes for reconstructing the object. It is worth noting that we did not explore lower training times for the text descriptor, which could potentially result in shorter execution times.

6.1.9 Data augmentation

Data augmentation has been used to prevent the DreamText model from overfitting to the input image. In this case, overfitting the image means learning the exact nuances of the image, such as lighting, object disposition, and camera pose, rather than focusing on the object's intrinsic details. By employing data augmentation techniques, variations are introduced into the training data, ensuring that the model generalizes effectively and learns the essential features of the objects, rather than memorizing specific image instances. The used augmentations are the following ones:

```
1 tf1 = transforms.RandomHorizontalFlip(p=0.5)
2 tf2 =
  ↪ transforms.RandomApply([transforms.RandomRotation(degrees=10,
  ↪ fill =1.0)], p=0.9)
```

```
tf3 = transforms.RandomApply([transforms.GaussianBlur(5, (0.1,
↪  2.0))], p=0.25)
tf4 =
↪  transforms.RandomApply([transforms.RandomCrop(img.shape[2]*0.8,
↪  fill=1.0)], p=0.5)
tf5 = transforms.RandomGrayscale(p=0.10)
tf6 = transforms.RandomApply([transforms.ColorJitter(0.1, 0.1,
↪  0.1, 0.1)])
tf7 = transforms.Resize(512)
transformations =
↪  transforms.Compose([tf1,tf2,tf3,tf4,tf5,tf6,tf7])
```

6.1.10 Text embedding training

We use Mean Squared Error loss previously introduced in Equation (2.7), which compares noise values. We use Adam [53] optimizer with pytorch default parameters and a learning rate of 3e-4.

6.2 FitFusion

All the experiments described in this section were conducted using a fork of the original code as of March 23 [3]. It is important to note that the code may have undergone changes and updates since then, which could potentially affect the comparison between the experimental results presented here and the current state of the code. However, we believe that the improvements and findings obtained from these experiments are noteworthy and can potentially be beneficial in other versions of the code as well. While the specific implementation details may vary, the insights gained from these experiments can provide valuable knowledge and guidance for future developments and enhancements in the codebase. Let us analyse first how different components behave separately.

6.2.1 Epochs

We found that improving the epochs solely did not produce any improvement in the reconstructed 3D sample quality. Figure 6.2 shows a comparison between rendered images after 100 epochs and after 200 epochs. We see that some samples gain colour quality and are more defined, however no one seems to contain more details.

[3] https://github.com/ashawkey/stable-dreamfusion

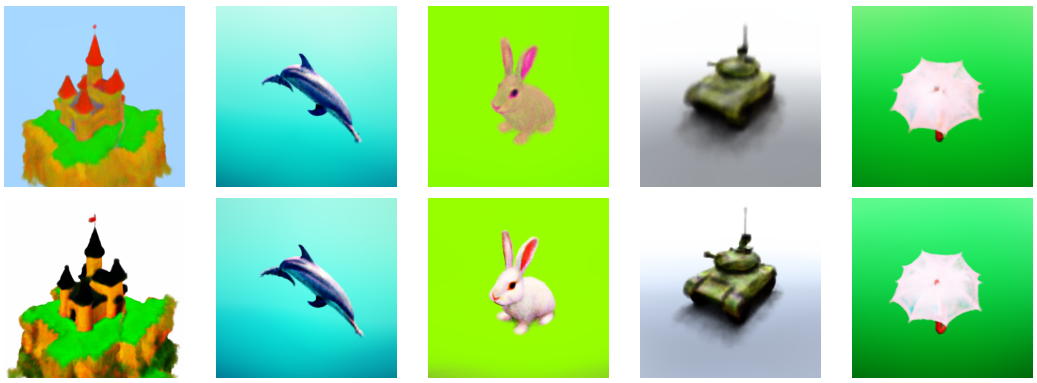

Figure 6.2: Epoch comparison on the result rendering from the NeRF model. First row shows rendering produced with 100 epochs of training, while second row shows renderings after 200 epochs of training. We observe slight improvements in color and overall image quality, although the level of detail remains relatively unchanged.

Please note that Appendix Section 8.5 contains additional rendered samples that demonstrate the observed details in greater depth.

6.2.2 Image size

We observed that increasing the image size resulted in longer execution times, ranging from approximately 60 to 80 minutes for augmentations from 64 to 256. See Figure 6.3. While some images displayed enhanced details, no notable observations were made regarding this aspect.

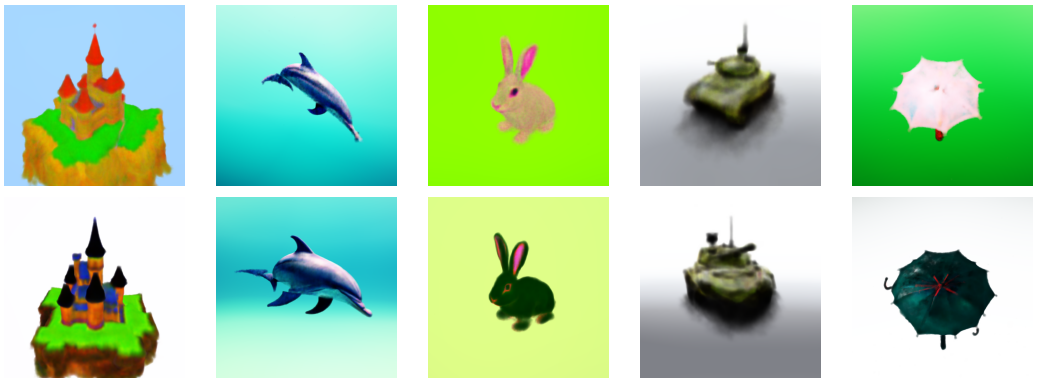

Figure 6.3: The first row of the figure displays the results obtained by rendering 64x64 images, while the second rows showcase the outcomes obtained by rendering 256x256 images. In some cases, the generated samples show improvement, while in others, there are noticeable color transitions that appear unusual.

6.2.3 Guidance

Figure 6.4: The first row of the figure displays the default results, while the second rows showcase the outcomes obtained with a guidance value of 1000. Here we can observe that the quantity of details is enhanced, but it appears that the model struggles to display them accurately.

We observed that increasing the guidance values resulted in higher contrast images with a greater presence of artifacts, see Figure 6.4. These artifacts represented elements of the 3D reconstruction intended to provide additional details, but were not accurately expressed. This finding led us to consider the combination of higher guidance values with larger image sizes, as we hypothesized that the image size acted as a bottleneck in capturing and representing the desired details.

6.2.4 Final proposal comparison

This new approach allows the model express the desired quality and details of the different generated reconstructions as seen in Figure 5.1. We have seen texture improvements in almost all generated samples and general quality augmentation in most of them. We were strongly convinced that the model was capable of generating high quality samples, so the cause must be more than 1 bottleneck. Using bigger image space in NeRF generations allowed the details to flow from the NeRF to the diffusion model. Using bigger guidance allowed the diffusion network to ignore the image prior and push forward these details. The quality improvement is huge while the execution time only increases from 60 to 80 minutes. Figure 6.5 shows more examples between the execution of the base provided code and the fine-tunned version.

Please note that Appendix Section 8.4 contains additional rendered samples that demonstrate the observed details in greater depth.

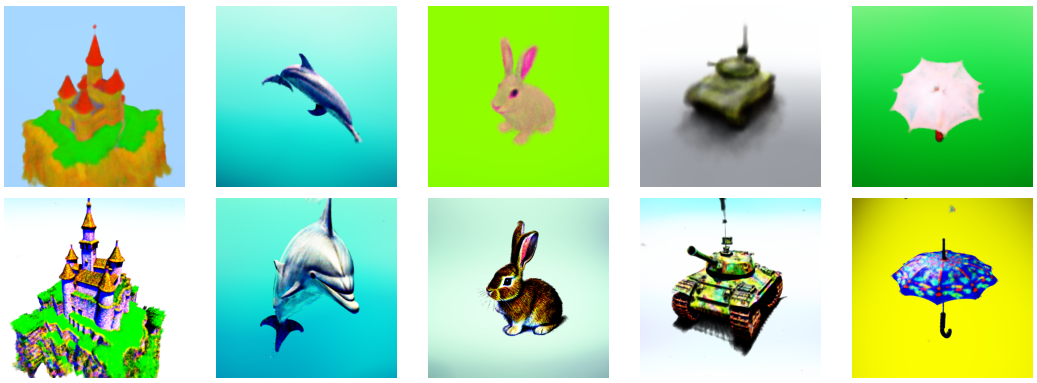

Figure 6.5: Comparison on the rendered results between the basic implementation(first row) and our fine tuned proposal (second row). Observe the significant improvement in sample quality.

6.3 DreamText

The experimental section of this study serves two main objectives. Firstly, it aims to showcase samples generated by our DreamText model, highlighting its capabilities in text generation. Secondly, it involves a comparative analysis between DreamText and other similar state of the art implementations. Lastly, we introduce a preliminary section that explores a previous approach [1], which ultimately led us to the development of our latest technology.

6.3.1 First approaches

Throughout our extensive research journey, we conducted numerous experiments, leading us to the development of our solution. Here, we highlight some of these experiments and the ideas extracted from them.

Embedding training

In our initial experimentation, we focused on training the text embedding used as input in the diffusion model, employing the loss function mentioned in equation (5.4). After training for 3000 epochs, we proceeded to assess the performance of the diffusion process by generating images using the embedding. The results of this evaluation can be observed in Figure 6.6, which showcases some of the Stable Diffusion model generations achieved using the trained embedding.
These outcomes unveiled a remarkable finding: the textual embedding can effectively preserve most of the image information, subsequently serving as valuable conditioning input for the Stable Diffusion model to regenerate it

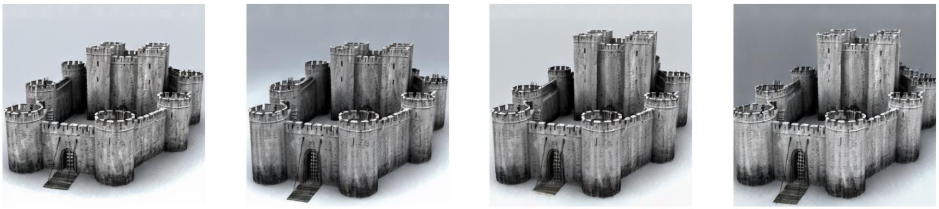

Figure 6.6: Regenerated images of the castle 1 from the dataset after pre-training the conditioning embedding. Notice how Stable Diffusion can reconstruct the image almost perfectly in all cases.

from pure noise. This revelation attests to the potential of leveraging the textual embedding as a means to store and utilize image-related data.

Initially, we held the optimistic belief that the text embedding could be leveraged for 3D generation purposes. However, after some experiments, it became evident that the embedding was excessively specific, resulting in the utilization of the same image for training every rendered perspective within the NeRF model. Consequently, this approach proved futile, as it failed to yield any meaningful reconstruction or contribute to effective training overall. Notice that training this embedding is similar as training the best sentence, in the embedding space, that describes the image.

Noun training

Our second approach involved selecting a noun that best described the image and fixing all parameters in the network except for its embedding, which we then fine-tuned. This process corresponds to the second part of the DreamText method, but training only the noun. This idea comes from [54] and is the one used at [1]. Our initial expectation was that the noun embedding would undergo a transition, progressing from a general to a more specific description of the object at hand. To explore this hypothesis, we initiated an experiment utilizing the text prompt "An image of a castle" and trained the model to refine the embedding specifically for the "castle" token. As anticipated, we observed a greater diversity in the regenerated images when employing Stable diffusion, since we were training the noun embedding rather than the final embedding. This result yielded a satisfactory range of generated images, showcasing the flexibility of stable diffusion.

However, despite the successful image generation, we encountered limitations when attempting to reconstruct 3D samples. The approach, while effective in producing image variety, did not sufficiently capture the necessary information

for accurate 3D reconstruction. This discrepancy prompted us to reassess the methodology and explore alternative avenues to overcome this particular challenge.

6.3.2 Descriptor training

In our subsequent approach, we drew inspiration from the work of [55] and aimed to train a descriptor for the class noun. Building upon a similar technique, we devised a straightforward yet effective strategy. Our objective was to train an additional word to describe the unique characteristics of the object depicted in the image, while preserving the general meaning within the noun. This approach aimed to utilize the noun for 3D reconstruction information while leveraging the descriptor for specific details. To enhance the descriptive power of the generated text, we introduced adjectives into the sentences. These adjectives were obtained using CLIP, similarly as in the initial phase of our final methodology.

This methodology has yielded impressive results, as demonstrated in Figures 6.7 and 6.8. However, one notable challenge is the need to manually determine the optimal reconstruction guidance and set the training time for the text tokens. Despite this inconvenience, the overall outcomes highlight the potential and effectiveness of this approach in generating high-quality reconstructions.

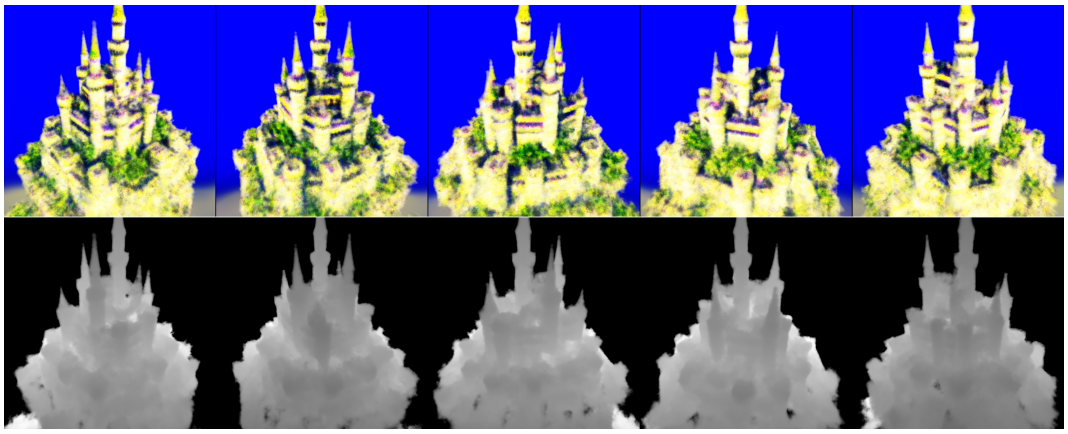

Figure 6.7: Example of a trained reconstruction utilizing descriptor training for Castle 3 from the training dataset. We see great results, although the resemblance to the original castle is limited.

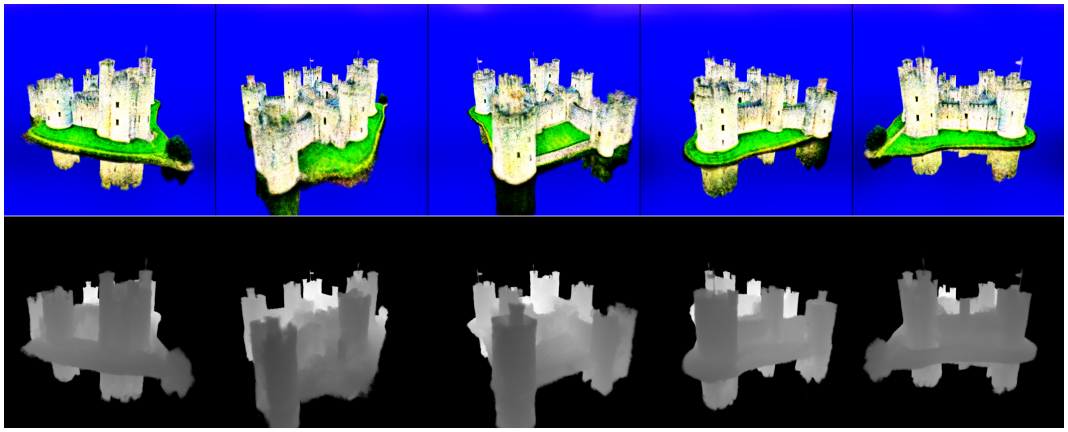

Figure 6.8: Example of a trained reconstruction utilizing descriptor training for Castle 3 from the training dataset. We observe excellent results, although there is not a strong resemblance to the original image.

6.3.3 Method comparison

Our approach final approach, DreamText, has delivered exceptional outcomes in numerous executions, consistently exceeding our initial expectations. It is worth noting that the execution time for our GPU-based model varies depending on the task at hand. Specifically, training the token requires approximately 2 hours and 10 minutes, encompassing around 10.000 iterations. On the other hand, the 3D reconstruction process takes approximately 80 minutes also for 10.000 iterations. Due to the thesis time constraints, we have not yet been able to thoroughly explore alternative iteration number for training the text embedding. However, it is worth considering that shorter training times may be viable for this phase.

Our model is compared with another similar approach for the problem of 3D reconstruction from images called RealFusion [1]. These comparisons can be completely visualized in a specific *webpage*[4] made for this project. This webpage is also included in the code of the project in a file named index.html

For successful 3D RealFusion approach incorporates the image as an additional loss to the NeRF model, requiring the NeRF model to render the image accurately from a specific viewpoint. A notable drawback of this approach is the manual input required to align the provided camera angles accurately with the image observed pose. Other poses are rendered with a trained noun token as described above and following [54]. As noted above the noun token alone usu-

[4]https://nazarpuriy.github.io/

ally cannot store the whole information of the object, but it leverages with the image information during the reconstruction process. Notice that their model descriptor time takes 3h in comparison with our that takes 2h of training. Notice that examples corresponding to the 3rd and 4th rows of the dataset 6.1.1 were reconstructed using already pre-trained token embedding that they provide with the code[5]. Notably, a decrease in performance was observed when reconstructing objects using freshly pre-trained tokens, prompting us to speculate that their embeddings may have undergone slight fine-tuning, potentially contributing to enhanced reconstruction quality.

Although we observed that this descriptor struggled to accurately describe the samples, it effectively utilized the information from the image input to generate 3D reconstructions. However, we discovered that significant disparities between the trained embedding and the image resulted in poor model performance. These discrepancies may not be readily apparent in static images, but they become evident when examining the accompanying videos or GIFs. While their model excels in generating images from a specific image perspective, our focus lies in achieving a comprehensive alignment across all rendered views, as our ultimate objective is to obtain a complete 3D reconstruction of the object instead of good sparse views. Figures 6.9, 6.10 and 6.11 showcase some examples of both model behaviours.

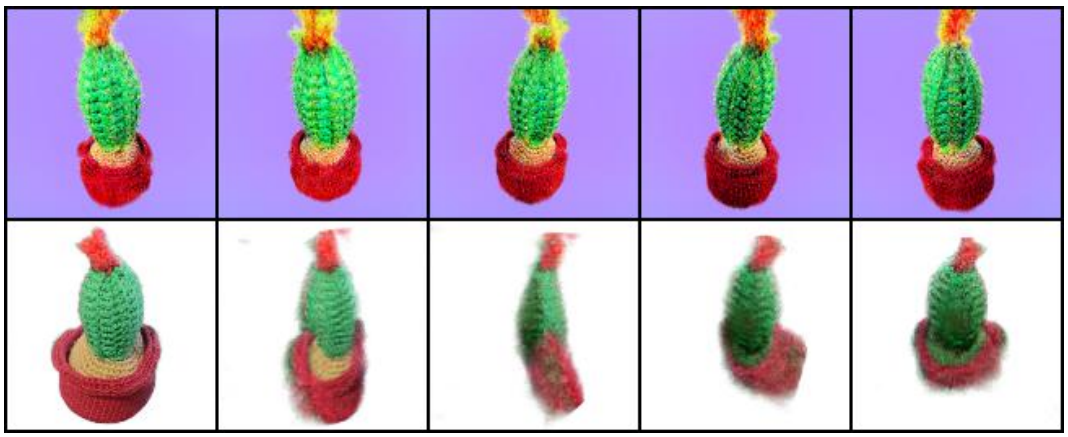

Figure 6.9: Cactus 1. Note the superior 3D spatial coherence of our proposed model (top), when compared to RealFusion (bottom).

[5]https://github.com/lukemelas/realfusion

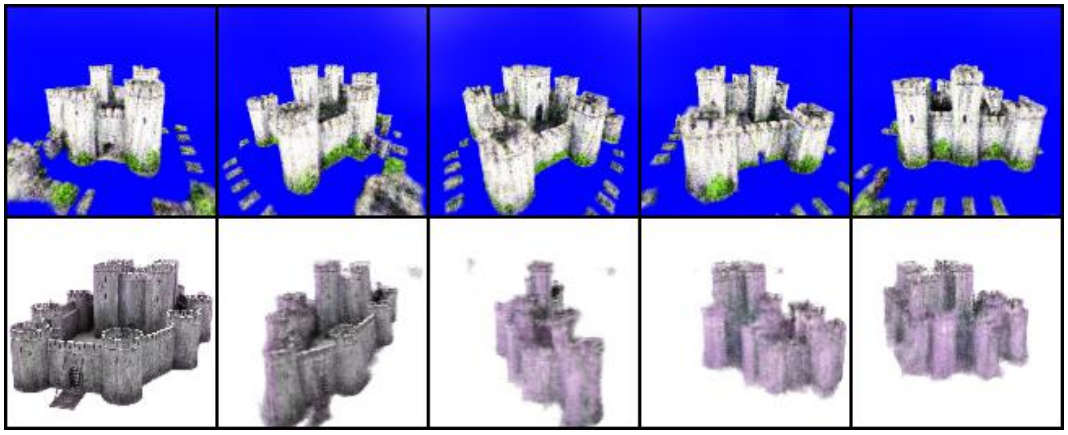

Figure 6.10: Castle 1. Our model (top) excels at capturing complex geometries with remarkable precision, effortlessly reproducing intricate details that RealFusion (bottom) struggles to acquire.

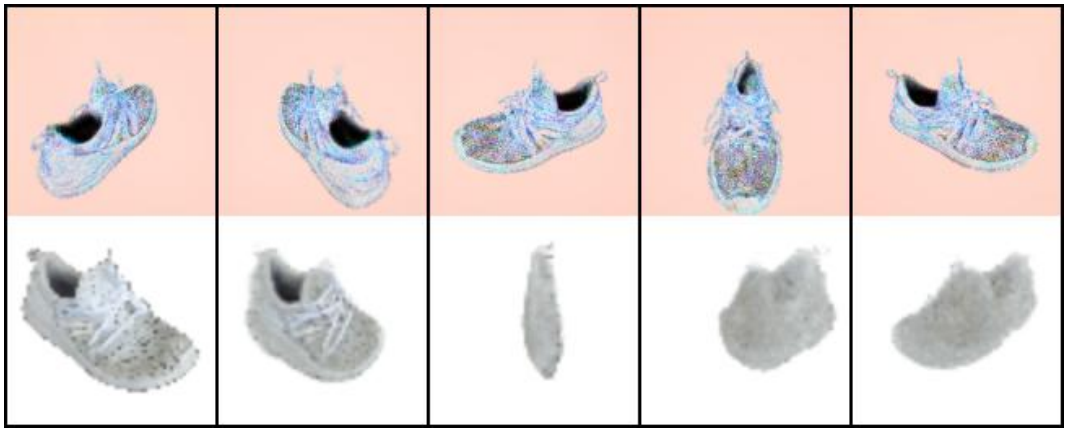

Figure 6.11: Sneaker. Our model demonstrates the ability to generate plausible views of the original sneaker, including a successful match with the original view (top row, last column). In contrast, their model predominantly renders the image view while mantaining a general flat reconstruction.

6.3.4 Limitations

Our model inherits certain limitations from both the Stable Diffusion and Stable DreamFusion models, upon which it is based. Notably, the presence of the Janus face problem stands out as a significant issue, hindering the generation of alternative views beyond the frontal face of an object. Additionally, we have observed instances where certain parts of the object are repeated in the generated samples, which can impact the overall quality. While text view conditioning helps mitigate this problem to some extent, further research and

improvements are required to address this issue comprehensively. Furthermore, when rendering complex objects, we encountered challenges on rendering all the details or learning all the information. We suggest that employing a larger image size could potentially overcome these obstacles.

Finally, it is worth highlighting that RealFusion demonstrates better performance in certain cases of the Janus problem. This can be attributed to the inclusion of a mask loss in their model architecture. The mask loss effectively removes the density of points that are projected onto pixels outside the boundaries of the object in the input image.

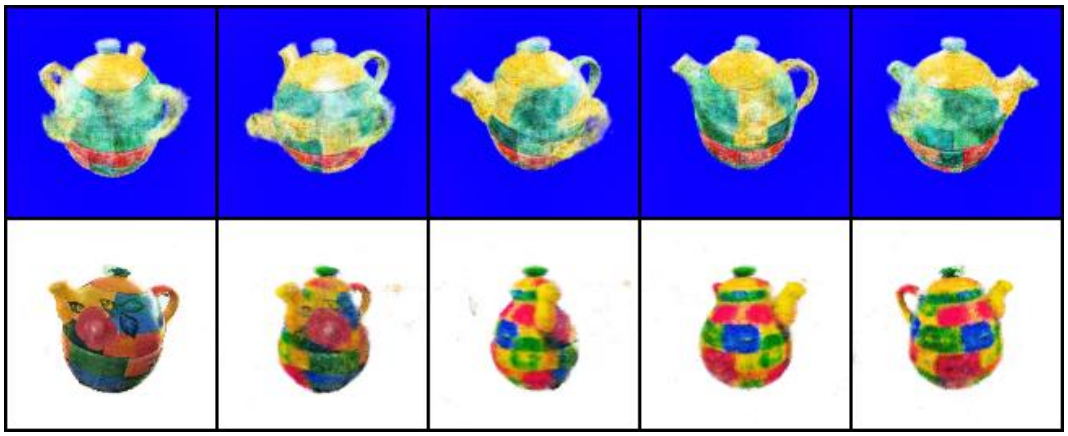

Figure 6.12: Teapot. An interesting observation in our (top) samples is the presence of the Janus problem, where the handle of the teapot is rendered multiple times. This artifact highlights a limitation in our model. In contrast, their model does not exhibit this behavior, suggesting that the mask error might help in the generation of some objects.

Chapter 7

Conclusions and Future Work

We have presented FitFusion, a high-quality text-to-3D generative model that has been developed through a comprehensive analysis of the most prominent state-of-the-art models and a meticulous extraction of relevant information from them. This enhancement has enabled our model to achieve the expected high-quality outputs comparable to those of Stable Diffusion and NeRF.

Furthermore, our research has significantly contributed to the advancement of Image-to-3D generation, surpassing the performance of a previously published model at CVPR 2023 [1]. By leveraging text descriptors, we have successfully extracted essential object information from an image, which was later used to condition a Stable Diffusion model during the training of a Neural Radiance Field. This approach has demonstrated that direct image conditioning is not necessary for 3D reconstruction, as the textual descriptors alone contain sufficient information to accurately reconstruct the corresponding 3D object.

From a personal perspective, I am extremely pleased with the outcome of this project. It has been a challenging yet exciting journey, filled with ups and downs. However, this experience has deepened my passion for these fields and has solidified my decision to pursue a Master's degree in Data Science. I am grateful for the opportunity that I have had to explore these technologies, conduct experiments, and contribute to the advancement of 3D generative models.

7.1 Future work

Despite the extensive interest in this research area and our detailed exploration and proposed improvements, there are still several areas for future work:

- Noise start: Investigate the use of a NeRF model that initially renders scenes resembling Gaussian noise to reduce bias in the 3D generation process.
- Further investigate the potential of training the entire model to make the text fit a specific sample, as shown in [55], to enhance Image-to-3D reconstructions.
- Conduct rigorous quantitative comparisons with other existing works to evaluate the performance of our model.
- Investigate the synergies and potential collaborations with concurrent work in the rapidly expanding field to further enhance 3D generation capabilities, since numerous related works [56–58] have been published during the writing of this dissertation. Notice that his works rely on 3D datasets and cannot be directly compared with our findings.

Chapter 8

Appendix

This additional chapter serves to supplement the project by providing supplementary information. Section 8.1 offers a comprehensive guide on how to replicate code executions, enabling further experimentation. Following that, Section 8.2 presents informative diagrams that aid in understanding NeRF and DreamFusion models. In Section 8.3, a collection of image samples is provided, which were instrumental in gaining insights into the behavior of Stable Diffusion.

Expanding on the discussion of the proposed models, Sections 8.4, 8.5.1, and 8.5.2 present additional samples and comparisons.

8.1 Code replication

Before working on Stable Diffusion, one should ensure that his/her computer has the CUDA libraries installed and a graphics card with at least 12 GB of memory in order to successfully run the code. Some libraries have dependencies on the others and the requirements.txt, obtained with pip freeze, might not work properly. We recommend to run it one time and then install the missing libraries manually. Also the exact version of CUDA should be checked, and torch libraries that you need for your GPU.

There are three scripts that can be executed to run our code: for the text embedding, for the 3D reconstruction and to generate image samples for the diversity enhancement. The following code contains three imperative variables: *WORKSPACE*, *TEXT* and *PATH*. First is used to indicate where all the generated data will be stored, second is used to indicate where the training image is located and the last is used as the guiding text for the text to 3D generations. Besides, for DreamText, we recommend to add noun and quan-

tifier parameters to specify this values to the network, although they can be automatically predicted. The following sniper of code generates the sentence that describes the image and trains it:

```
python ./descriptor.py --workspace WORKSPACE --noun watch --
   quantifier a --image_path PATH
```

Once the sentence is generated and trained the 3D reconstruction can be made. It is imperative to provide the same workspace directory for this. The following snipet of code reconstructs the 3D object after training the descriptive sentence:

```
python main.py --workspace "object3" -O --h 256 --w 256 --iters
   10100 --load_token 1
```

Here, a snippet of the code to execute the model as explained in FitFusion;

```
python main.py -O --g 1000 --h 256 --w 256 --text "TEXT" --
   workspace WORKSPACE
```

Finally, the code used to generate sample images for diversity augmentation can be found here:

```
python stable_diffusion.py --text "An␣image␣of␣a␣dog" --
   workspace example
```

Visualization

Samples can be visualized in a graphical user interface using:

```
python main.py -O --test --gui --workspace examples/sample1
```

We have uploaded 2 samples: examples/sample1 and examples/sample2. Change the workspace in the code above by one of these two directories to see them.

8.2 Diagrams

This section includes the architecture of the NeRF model (see Figure 8.1) and a visual diagram of the Stable DreamFussion model (see Figure 8.2). Both figures provide a visual representation of the respective models, illustrating the key components and their interactions.

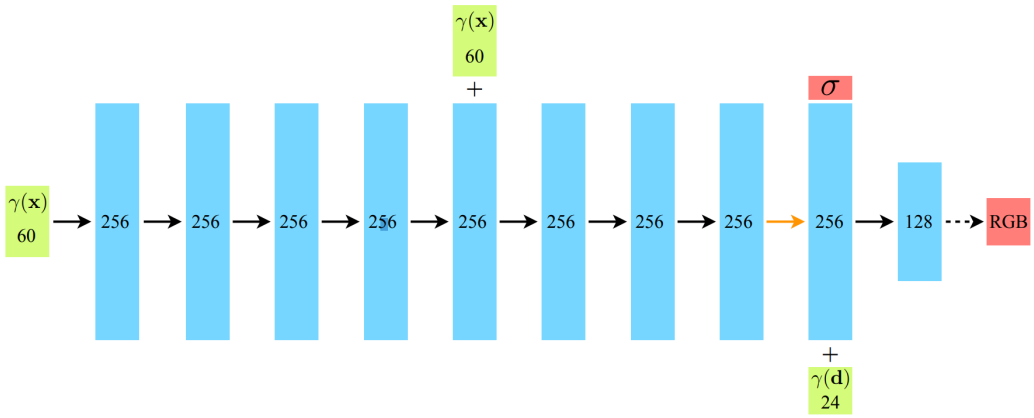

Figure 8.1: NeRF MLP architecture, extracted from [13]. Black arrow indicates ReLU, orange indicates no activation and dash arrow indicates sigmoid. + indicates concatenation. All blue blocks are fully connected layers. Green blocks are inputs and red blocks are outputs.

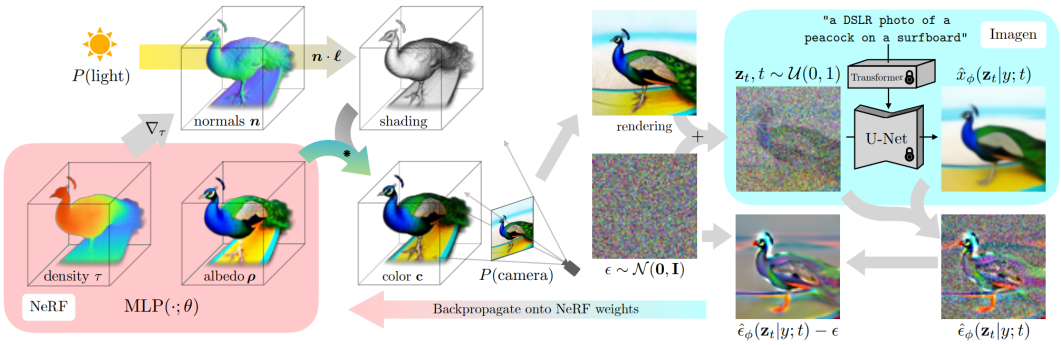

Figure 8.2: Pipeline of DreamFusion model, extracted from [12]. We see that the NeRF model generates a view of the peacock, which is shaded. The shaded version of the image, the rendering, is passed to the Stable Diffusion model that appends noise to it. Then the model itself predicts the noise using the text input. The difference between the real and predicted noise is used as an error to train the NeRF model.

8.3 Examples for model insight

This section provides a valuable opportunity to gain insights into the behavior of the Stable Diffusion model. The included generations and images offer a glimpse into the capabilities of this model.
Figure 8.3 showcases the impressive performance of Stable Diffusion [11], demonstrating its ability to generate high-quality samples. This serves as a point of

comparison, highlighting the contrasting quality of the initial results produced by the Stable DreamFusion model.

A snowflake	A car	A piano	A tree

Figure 8.3: Images generated with Stable Diffusion that showcase the details and the quality of the generated samples.

In addition, Figures 8.4 and 8.5 present failed experiments that explore the impact of different time and guidance values on the generated sample quality of Stable DreamFusion[2] model.

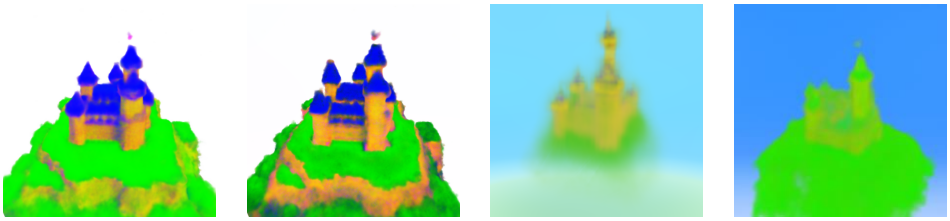

Figure 8.4: Failure cases during experiment with time parameter. First row corresponds to a random time between 500 and 980 included. Second corresponds to random time uniformly picked between 300 and 700. Third column corresponds to linearly decreasing time from 980 to 20. The last column corresponds to cosine appealing time.

Furthermore, Figure 8.6 illustrates the proficiency of Stable Diffusion in regenerating noised samples without the use of a text prompt. This showcases the model's capacity to leverage the information present in an image.

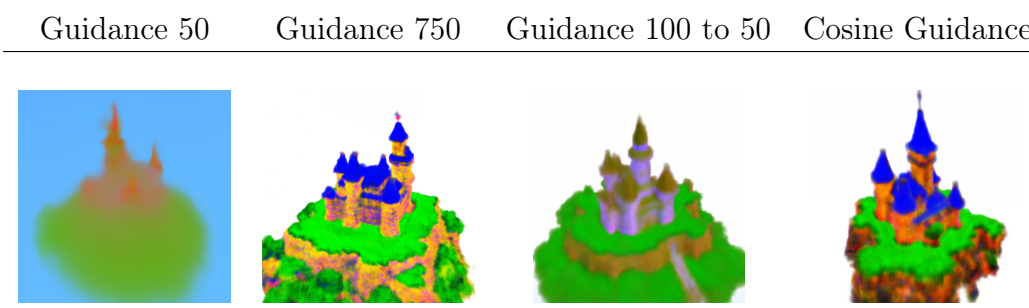

Figure 8.5: Failure cases during experiment with guidance parameter. First row corresponds to guidance value of 50 that does not capture any detail. The second image demonstrates that with a high value of 750, the model attempts to output details but is not capable of doing so successfully. Third row corresponds to a linearly decreasing guidance, without great results. Finally, the last example uses cosine appealing guidance, that has greater results that all the previous examples.

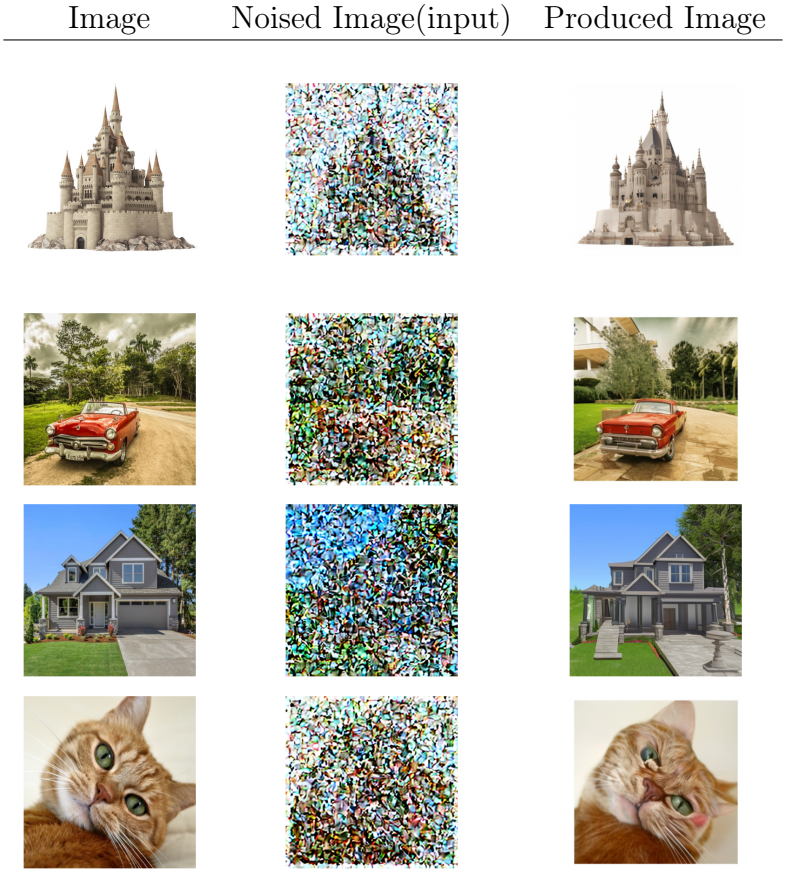

Figure 8.6: Stable Diffusion denoising process. This figure illustrates how the Stable Diffusion model is capable of removing the noise without having any text guidance of what he has to generate. The noise level used corresponds to the time step 550.

8.4 FitFusion Comparisons

This section expands on the examples provided in the experimental section, offering additional comparisons and insights. Specifically, we explore various modifications to the base model and compare them with the original results. In concrete compare the base model[2] with a version using more epochs (Figure 8.7), with a version with higher image size (Figure 8.8), with a version using higher guidance value (Figure 8.9) and finally with our proposal FitFusion (Figure 8.10).

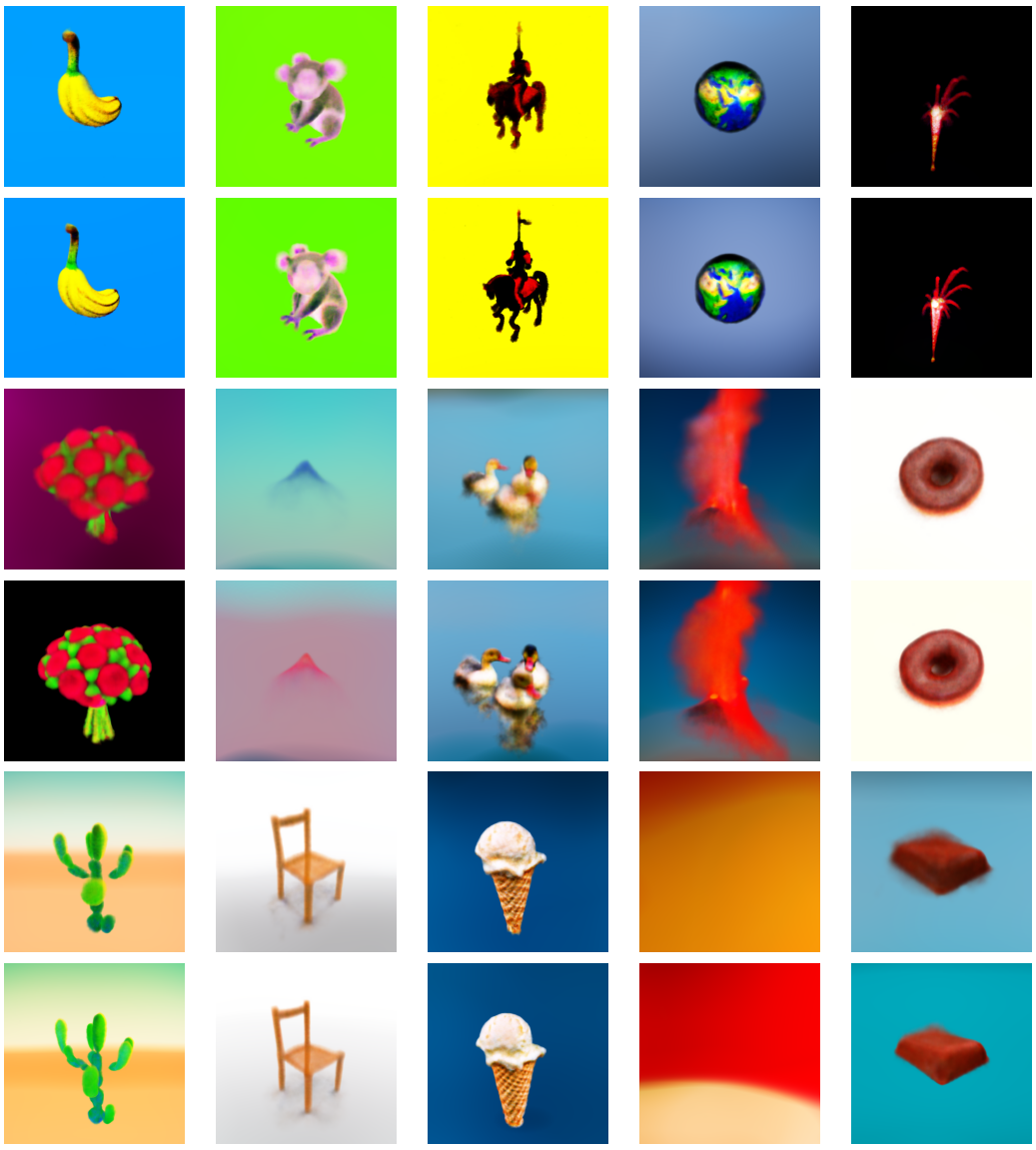

Figure 8.7: More comparisons of the epoch effects on the generated samples. Odd rows show the results obtained after training for 100 epochs, while the following even rows show the results obtained after training for 200 epochs. We observe slight improvements in color and overall image quality, although the level of detail remains relatively unchanged.

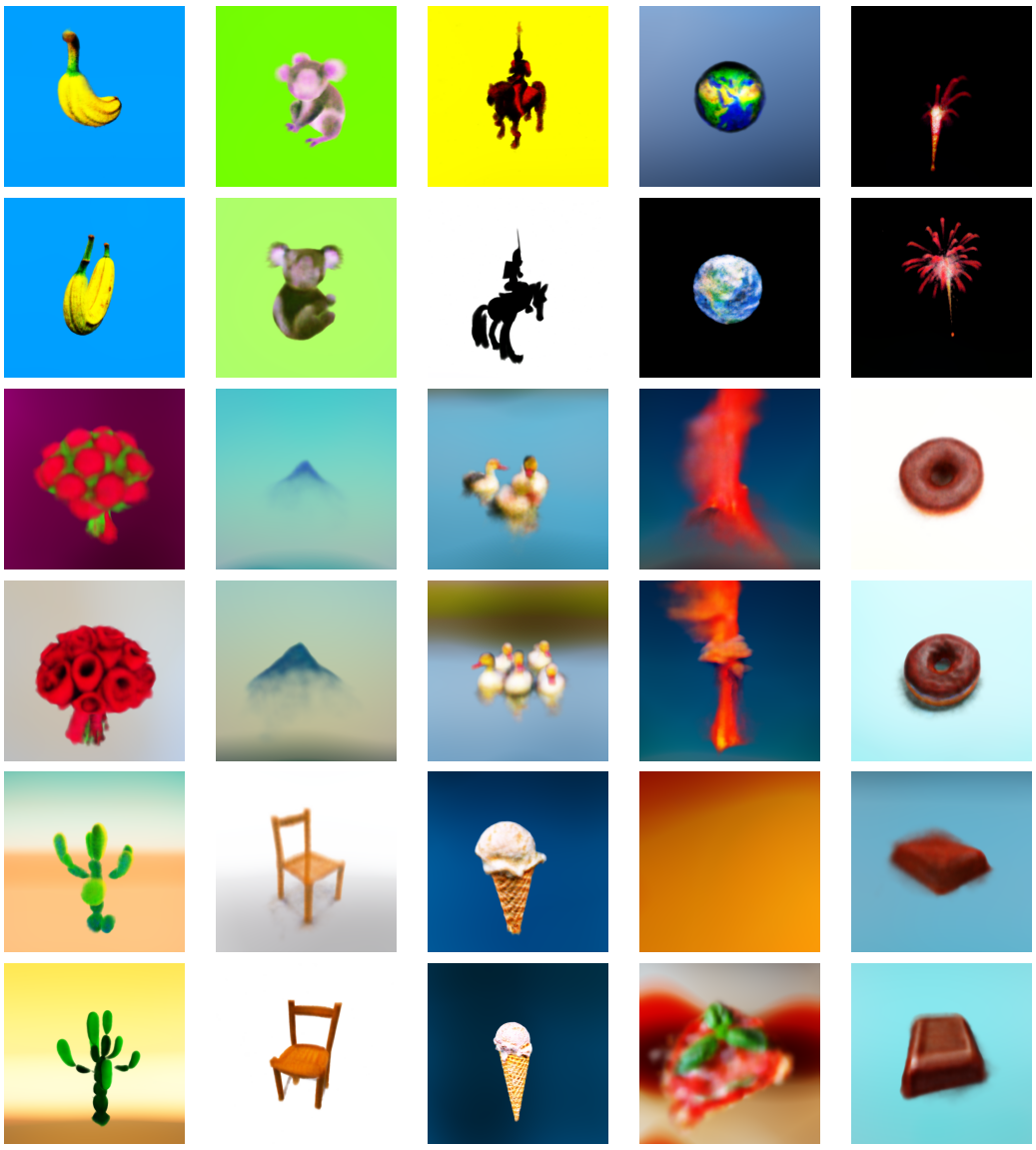

Figure 8.8: More comparisons of the image size effects on the generated samples. Odd rows show the results obtained with small image size of 64x64, while the following even rows show the results obtained with higher image size of 256x256. In some cases, the generated samples show improvement, while in others, there are noticeable color transitions that appear unusual.

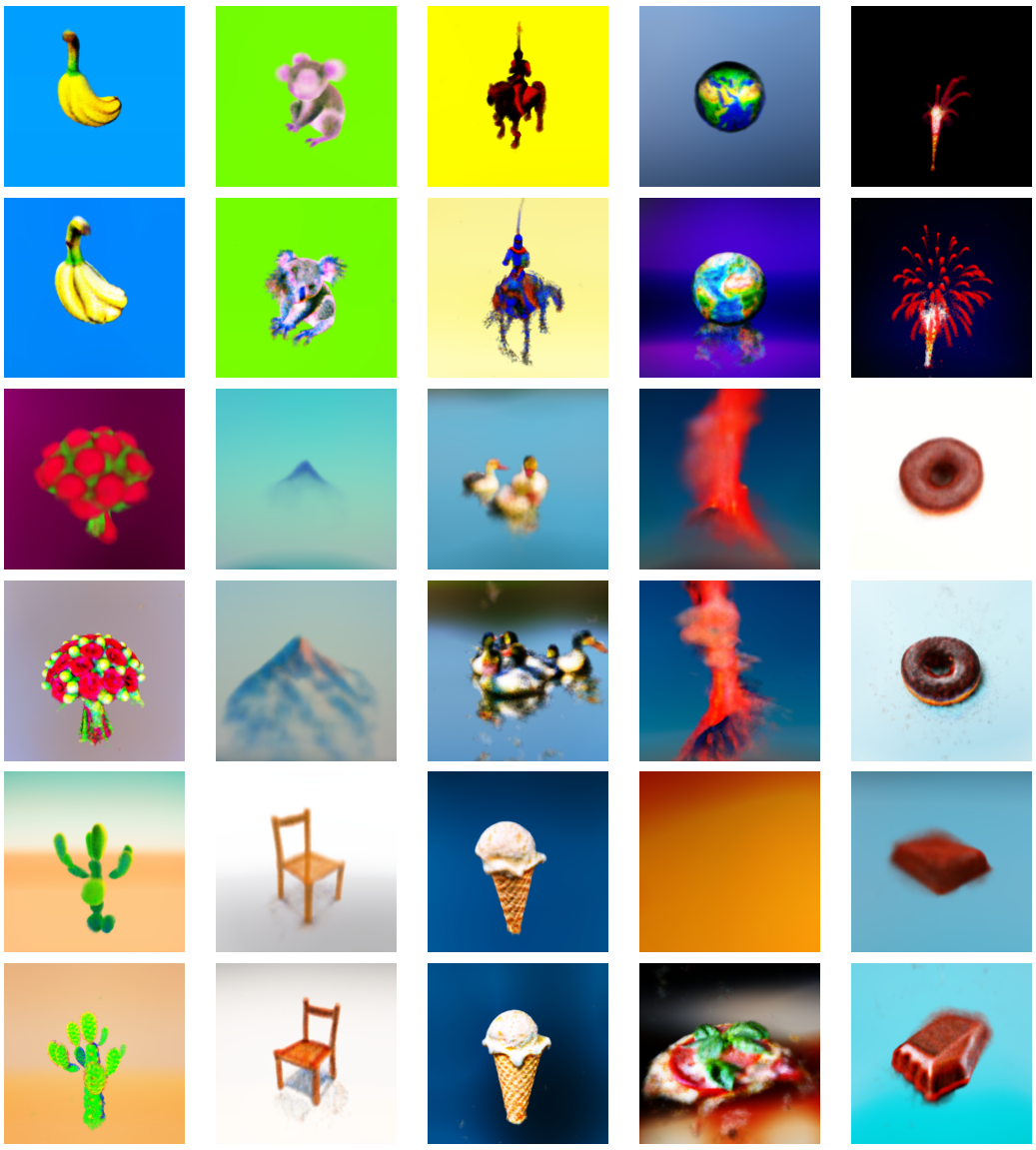

Figure 8.9: More comparisons of the guidance effects on the generated samples. Odd rows show the results obtained with guidance value of 100, while the following even rows show the results obtained with higher value of 1000. Here we can observe that the quantity of details is enhanced, but it appears that the model struggles to display them accurately

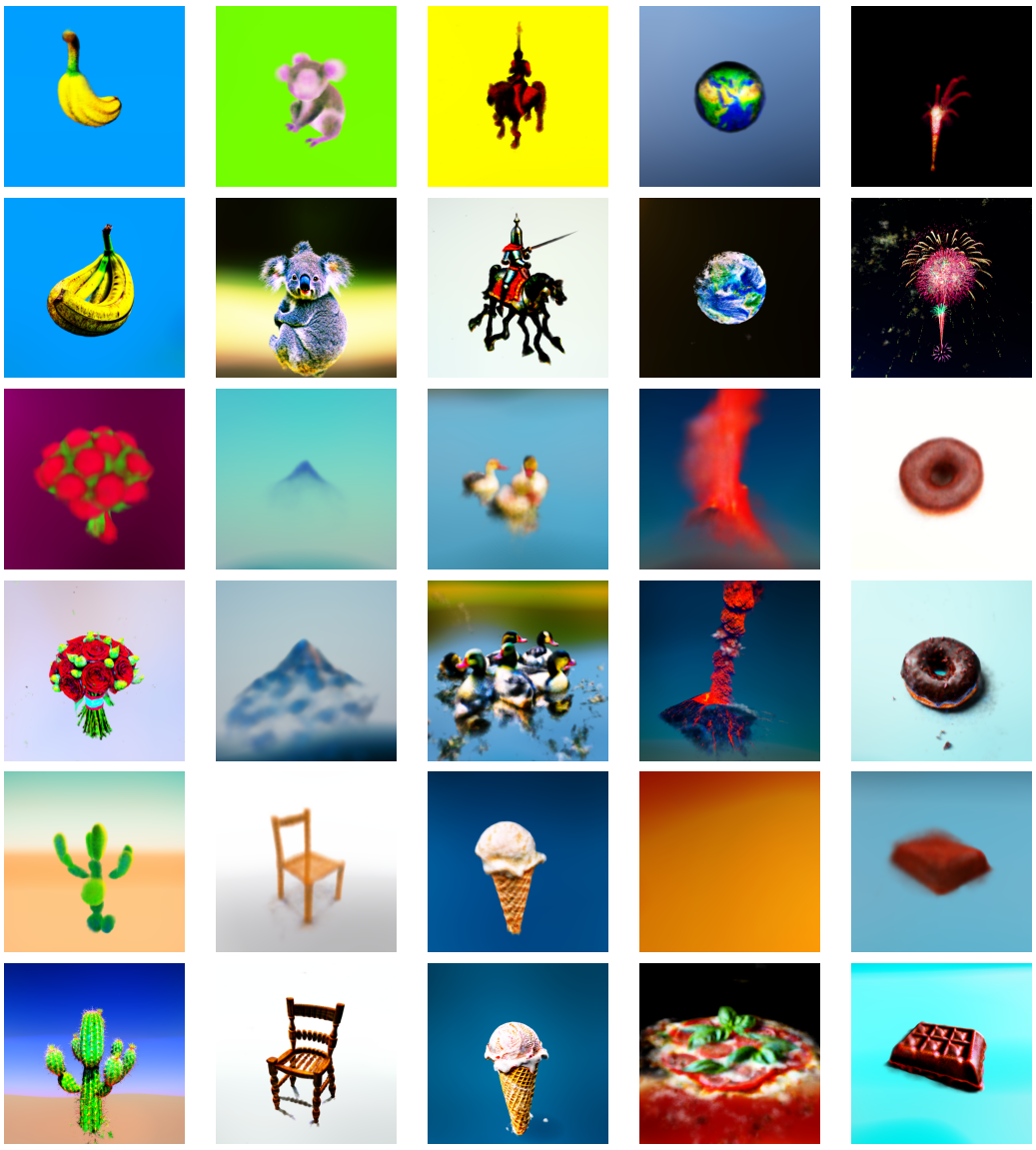

Figure 8.10: The comparison of renderings between the base provided code (odd rows) and our proposal (even rowds). Our proposal clearly shows higher quality and details in the reconstructions.

8.5 DreamText

8.5.1 Descriptor token examples

This section provides additional examples that showcase the previous approach to DreamText, referred as descriptor token. The Figures 8.11, 8.12, 8.13, 8.14,

8.15, 8.16, 8.17, and 8.18 illustrate the outcomes of this approach. While the results obtained are visually impressive, it is worth noting that the 3D reconstructions may not be sufficiently similar to the input images.

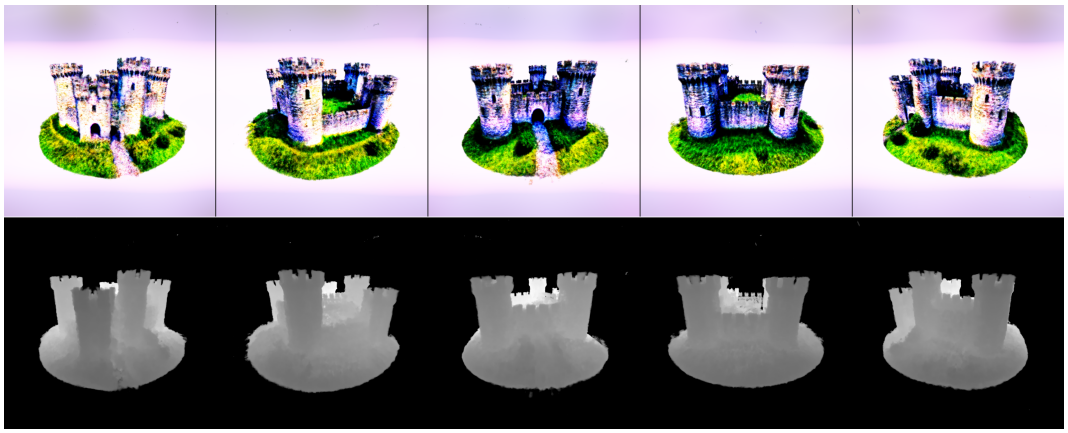

Figure 8.11: Example of a reconstructed castle 1 using descriptor training. We can see that, although the reconstruction is great, there isn't a high match with the provided image.

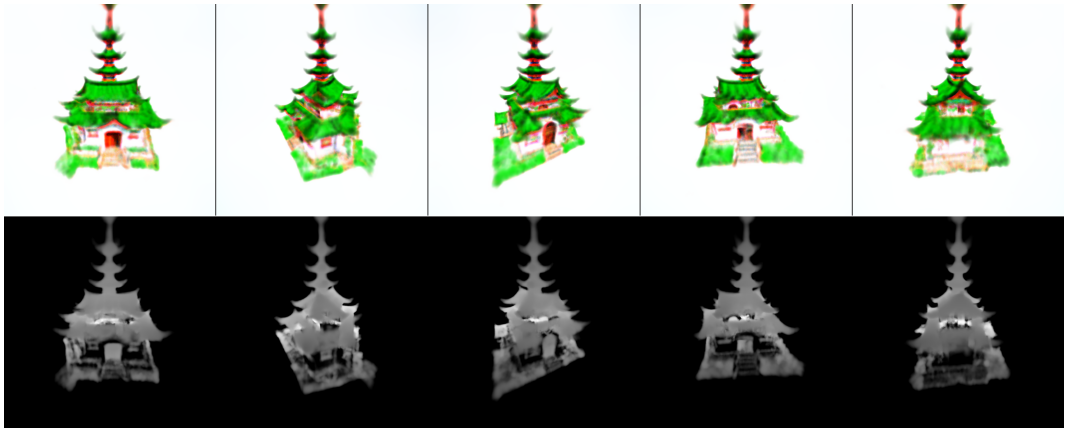

Figure 8.12: Example of a reconstructed castle 2 using descriptor training. We can see colour and density problems here.

8.5.2 DreamText examples

We present additional examples that further elucidate the strengths and difficulties encountered by our, DreamText model and RealFusion. These examples showcase the behavior of both approaches, shedding light on their respective capabilities and limitations. By examining a diverse range of objects and

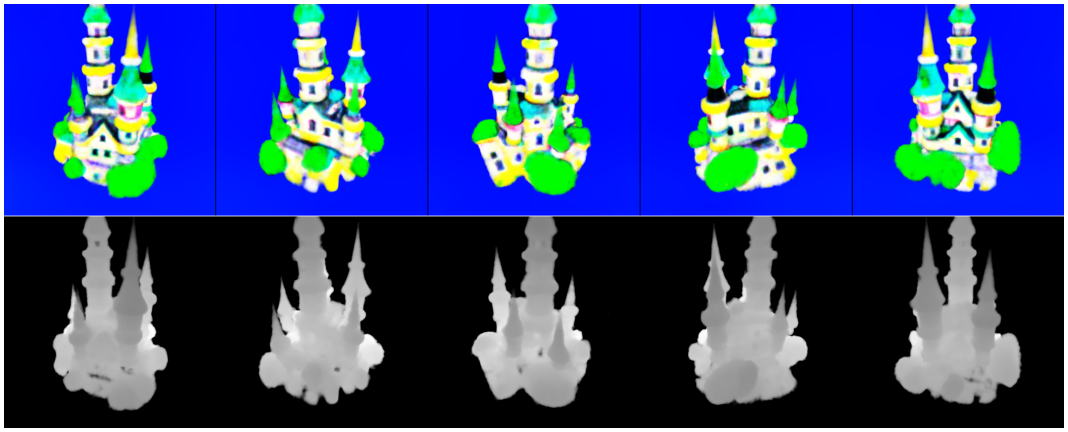

Figure 8.13: Example of a reconstructed castle 5 using descriptor training. As in the previous sample we see problem colours when comparing to the original image.

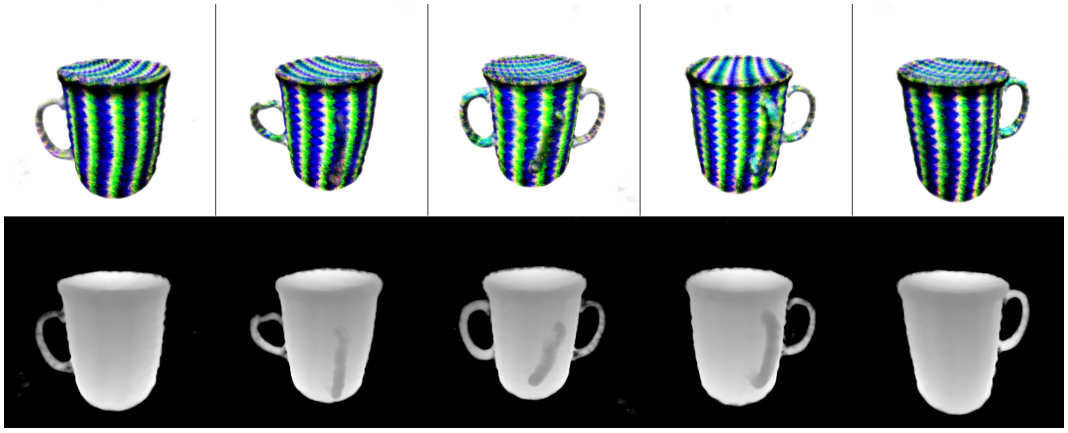

Figure 8.14: Example of a reconstructed mug using descriptor training. We can observe that while the object is preserved, the details of the image are completely lost.

scenarios, we aim to provide a comprehensive understanding of the outperformance and characteristics exhibited by our model in most of the samples. We would like to highlight that a more comprehensive and detailed comparison can be readily accessed on our webpage[1].

[1]https://nazarpuriy.github.io/

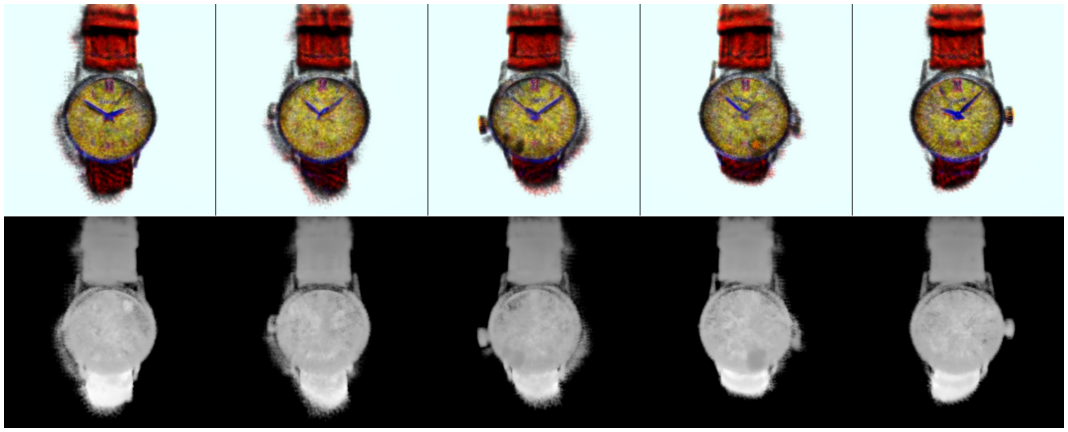

Figure 8.15: Here is an example of a reconstructed watch using descriptor training. It is important to note that the Janus problem can be observed in this case. This is a natural occurrence since there are limited photos available on the internet that capture the back sides of watches, thus Stable Diffusion must have a bias here.

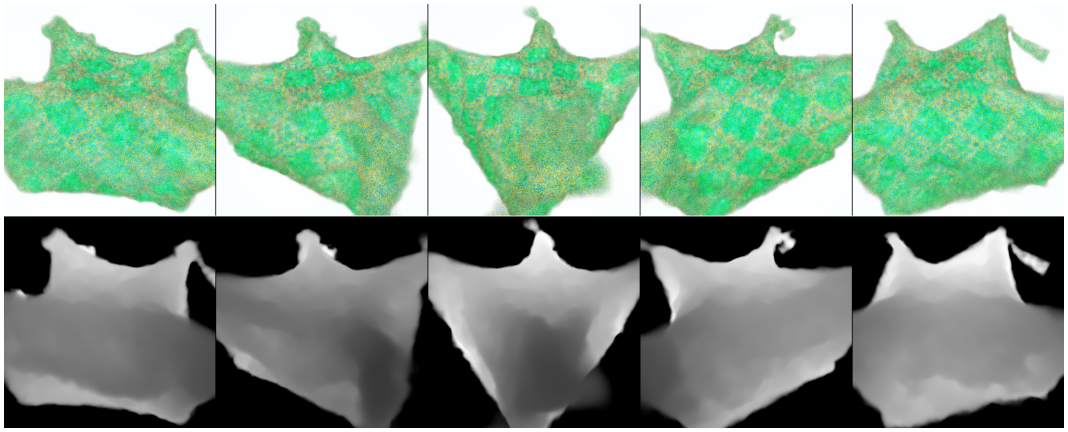

Figure 8.16: Example of a reconstructed purse using descriptor training. Here wee see quite good reconstruction of a it but without the correct geometry matching with the object in the image.

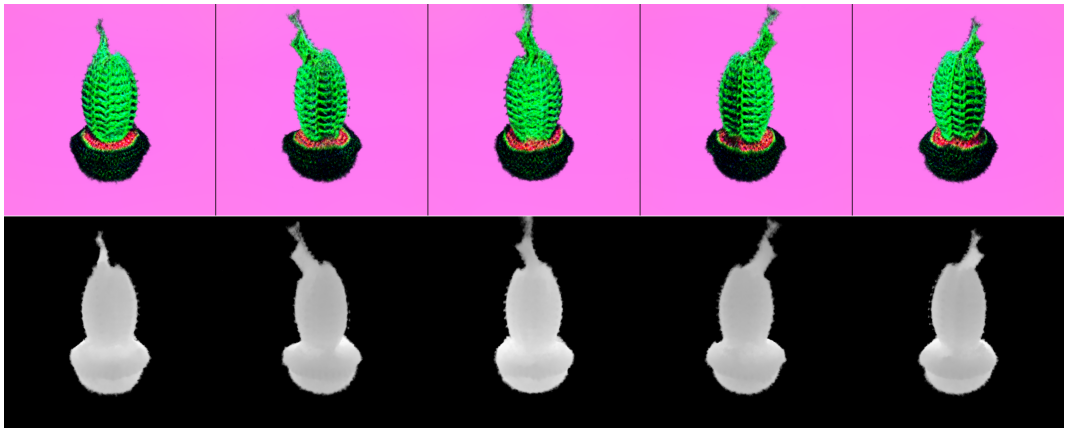

Figure 8.17: Example of a reconstructed cactus using descriptor training. The shape and texture are detailed but we see some colour problems on the basis.

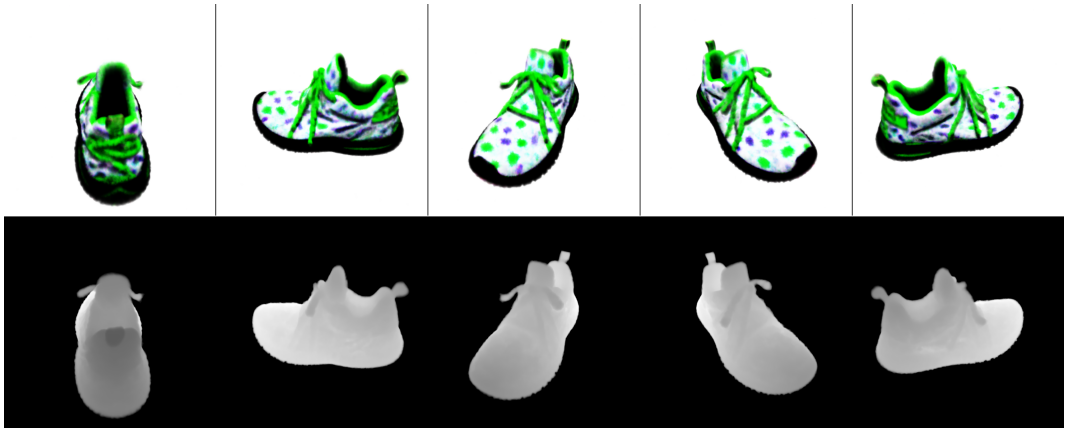

Figure 8.18: Example of a reconstructed sneaker using descriptor training. Similarly to the previous example we can observe colour problems, although the details and shape are correct.

Figure 8.19: DreamText (first row) and RealFusion (second row) renderings of the Microphone. Both models demonstrate proficient 3D reconstruction capabilities, producing commendable results. Nevertheless, it is notable that their model (bottom) lacks coherence between the input view and the corresponding views from different perspectives.

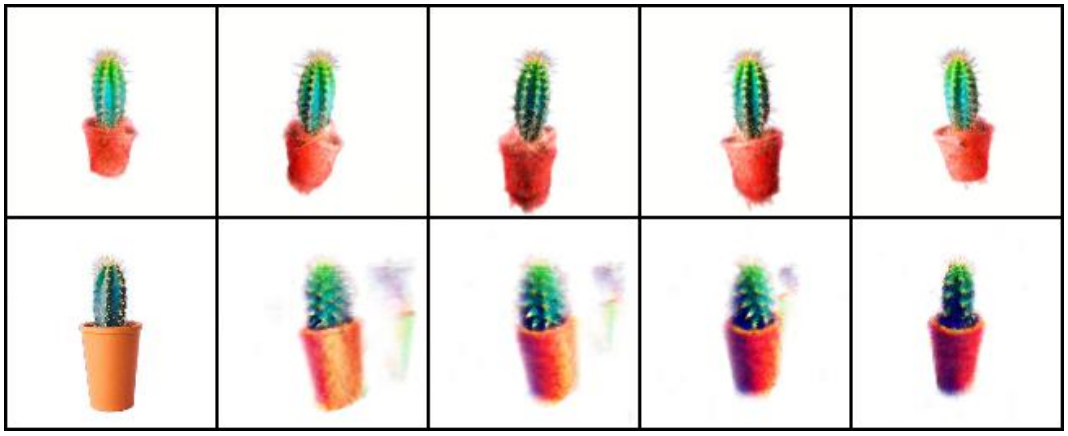

Figure 8.20: DreamText (first row) and RealFusion (second row) renderings of the Cactus 2. This example further emphasizes the proficiency of our model in generating accurate and compact 3D reconstructions of objects, while their model struggles to achieve alignment between the input view and the remaining perspectives. The second and third examples reveal the presence of fog noise, that comes from the influence of image loss.

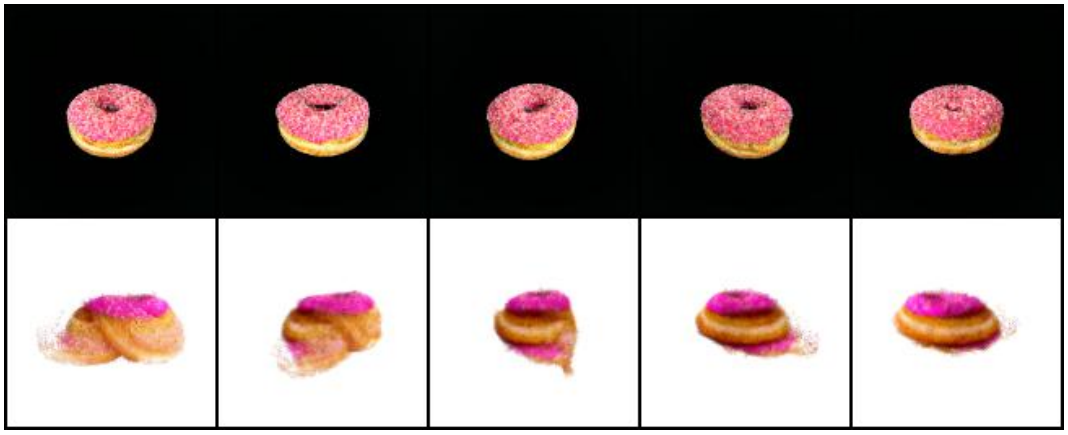

Figure 8.21: DreamText (first row) and RealFusion (second row) renderings of the Donut. In this example, our model demonstrates remarkable 3D reconstruction capabilities, producing visually pleasing results. However, it is important to note that our model falls short in achieving an exact match with the input image. On the other hand, their model encounters difficulties in rendering any view accurately.

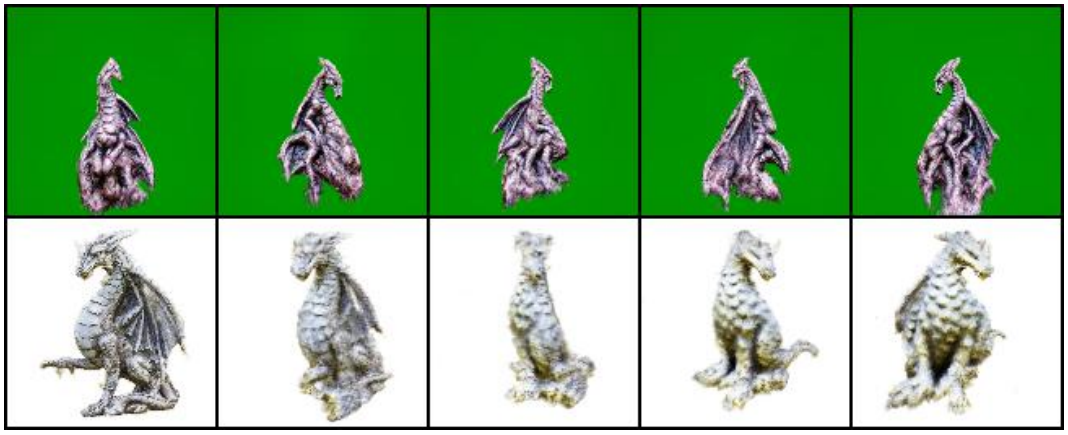

Figure 8.22: DreamText (first row) and RealFusion (second row) renderings of the Statue 1. Both models exhibit commendable results in generating 3D reconstructions. However, our model stands out by maintaining better overall coherence.

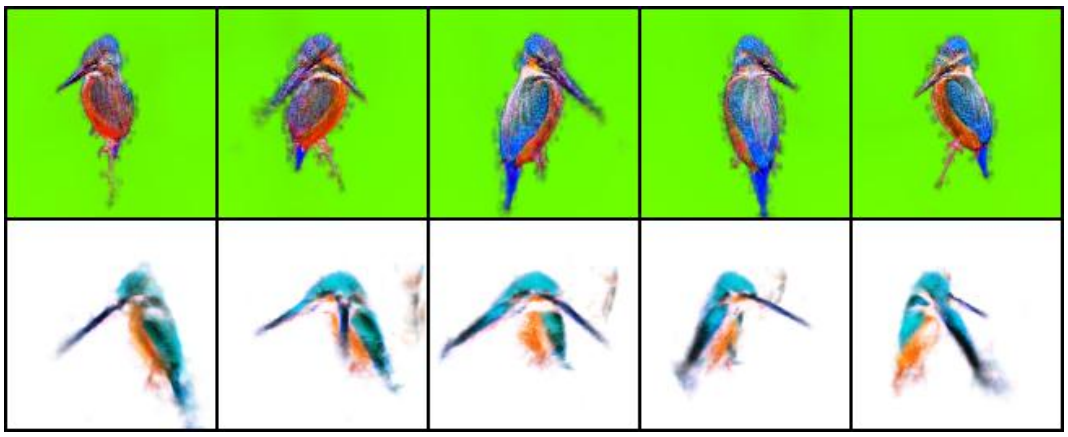

Figure 8.23: DreamText (first row) and RealFusion (second row) renderings of the bird. It highlights the challenges faced by both models, as the reconstruction faces the Janus problem. However, our approach successfully retains more of the object's characteristics.

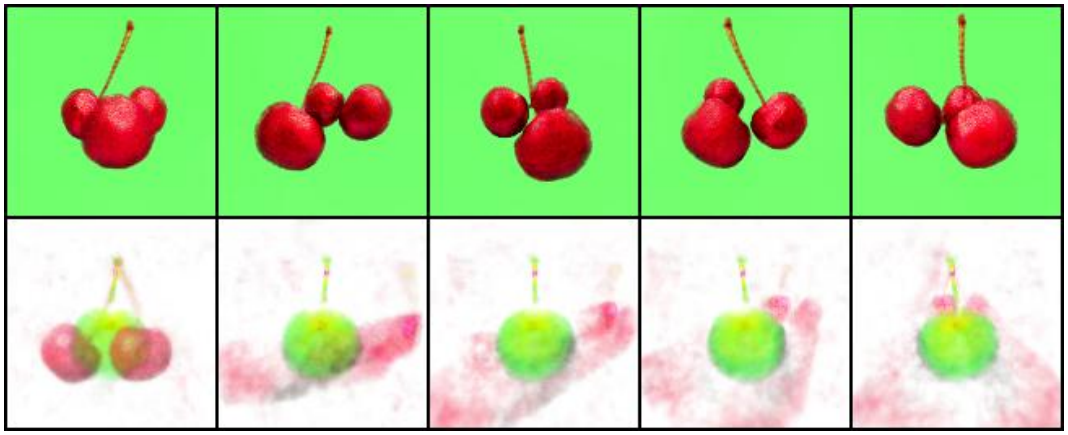

Figure 8.24: DreamText (first row) and RealFusion (second row) renderings of the Cherries. This particular case represents a failure for both models. Our model, unfortunately, falls short in accurately capturing the exact number of objects present in the image. This discrepancy can be attributed to inherent limitations in Stable Diffusion. Conversely, their model encounters challenges in effectively learning a robust token description, resulting in difficulties in generating faithful 3D reconstructions.

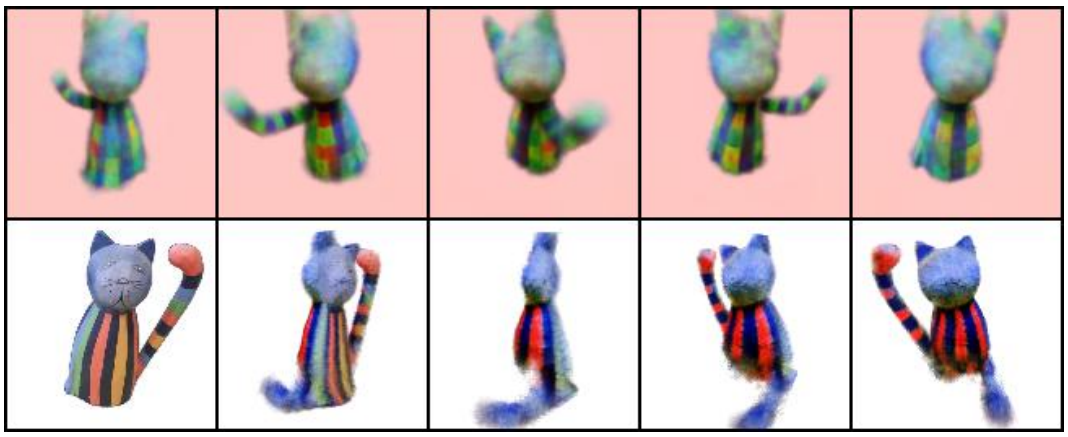

Figure 8.25: DreamText (first row) and RealFusion (second row) renderings of the Statue 2. We observe difficulties to render the object in both cases. However, it is noteworthy that their model surpasses our generations by effectively leveraging the information present in the image, resulting in a more accurate and refined 3D reconstruction.

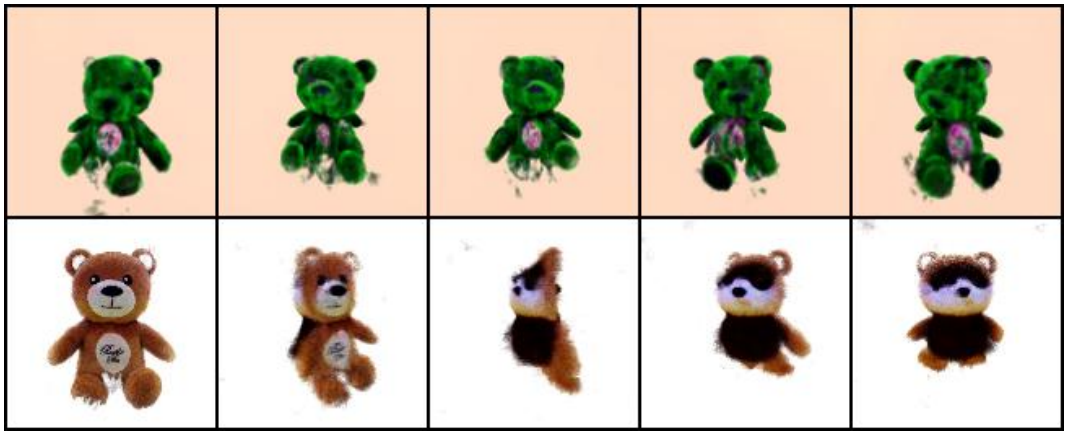

Figure 8.26: DreamText (first row) and RealFusion (second row) renderings of the Teddy. This example highlights a failure case for both models. Our model struggles to accurately render the color and shape of the object. Conversely, their model faces challenges in accurately rendering the shape and achieving a satisfactory alignment between the original image view and the other rendered perspectives.

Bibliography

[1] Luke Melas-Kyriazi, Christian Rupprecht, Iro Laina, and Andrea Vedaldi. Realfusion: 360° reconstruction of any object from a single image. In *Arxiv*, 2023.

[2] Jiaxiang Tang. Stable-dreamfusion: Text-to-3d with stable-diffusion, 2022. https://github.com/ashawkey/stable-dreamfusion.

[3] Geoffrey E. Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504 – 507, 2006.

[4] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

[5] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

[6] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *ArXiv*, abs/1411.1784, 2014.

[7] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976, 2016.

[8] Harshad Rai and Naman Shukla. Unpaired image-to-image translation using cycle-consistent adversarial networks. 2018.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[11] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695, June 2022.

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

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

[14] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[15] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86:2278–2324, 1998.

[16] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, 2010.

[17] Andrew L. Maas. Rectifier nonlinearities improve neural network acoustic models. 2013.

[18] Jürgen Esser. An introduction to linear algebra. 2006.

[19] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[20] Solomon Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86, 1951.

[21] Alexander Amini, Ava P. Soleimany, Sertac Karaman, and Daniela Rus. Spatial uncertainty sampling for end-to-end control. *ArXiv*, abs/1805.04829, 2018.

[22] Michael Douglass. Book review: Hands-on machine learning with scikit-learn, keras, and tensorflow, 2nd edition by aurélien géron. *Physical and Engineering Sciences in Medicine*, 43:1135 – 1136, 2020.

[23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86:2278–2324, 1998.

[24] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15:1929–1958, 2014.

[25] Jascha Narain Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *ArXiv*, abs/1503.03585, 2015.

[26] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arxiv:2006.11239*, 2020.

[27] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *ArXiv*, abs/1505.04597, 2015.

[28] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=St1giarCHLP`.

[29] Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. *ArXiv*, abs/2102.09672, 2021.

[30] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. *ArXiv*, abs/2105.05233, 2021.

[31] Jonathan Ho. Classifier-free diffusion guidance. *ArXiv*, abs/2207.12598, 2022.

[32] 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. In *International Conference on Machine Learning*, 2021.

[33] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 586–595, 2018.

[34] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976, 2016.

[35] Alexey Dosovitskiy and Thomas Brox. Generating images with perceptual similarity metrics based on deep networks. In *NIPS*, 2016.

[36] Jiahui Yu, Xin Li, Jing Yu Koh, Han Zhang, Ruoming Pang, James Qin, Alexander Ku, Yuanzhong Xu, Jason Baldridge, and Yonghui Wu. Vector-quantized image modeling with improved vqgan. *ArXiv*, abs/2110.04627, 2021.

[37] Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12868–12878, 2020.

[38] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*, 2021.

[39] Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, L. Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. *ArXiv*, abs/1512.03012, 2015.

[40] Steven M. Seitz and Charles R. Dyer. Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision*, 35:151–173, 1997.

[41] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer. Deepvoxels: Learning persistent 3d feature embeddings. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2432–2441, 2018.

[42] Richard Szeliski and Polina Golland. Stereo matching with transparency and matting. *International Journal of Computer Vision*, 32:45–61, 1998.

[43] James T. Kajiya and Brian Von Herzen. Ray tracing volume densities. *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, 1984.

[44] Johannes L. Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4104–4113, 2016.

[45] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. doi: 10.1145/3528223.3530127. URL `https://doi.org/10.1145/3528223.3530127`.

[46] Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomerantes, and Markus H. Gross. Optimized spatial hashing for collision detection of deformable objects. In *International Symposium on Vision, Modeling, and Visualization*, 2003.

[47] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *CVPR*, 2022.

[48] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L. Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, Seyedeh Sara Mahdavi, Raphael Gontijo Lopes, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. *ArXiv*, abs/2205.11487, 2022.

[49] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. Laion-5b: An open large-scale dataset for training next generation image-text models. *ArXiv*, abs/2210.08402, 2022.

[50] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv: Learning*, 2016.

[51] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

[52] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks : the official journal of the International Neural Network Society*, 107:3–11, 2017.

[53] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[54] Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H. Bermano, Gal Chechik, and Daniel Cohen-Or. An image is worth one word: Personalizing text-to-image generation using textual inversion, 2022. URL `https://arxiv.org/abs/2208.01618`.

[55] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. 2022.

[56] 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. *ArXiv*, abs/2211.10440, 2022.

[57] Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. Zero-1-to-3: Zero-shot one image to 3d object. *ArXiv*, abs/2303.11328, 2023.

[58] Rui Chen, Y. Chen, Ningxin Jiao, and Kui Jia. Fantasia3d: Disentangling geometry and appearance for high-quality text-to-3d content creation. *ArXiv*, abs/2303.13873, 2023.