ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MASTER'S DEGREE IN ARTIFICIAL INTELLIGENCE

A Comprehensive Benchmark of Efficient Text-Driven 3D Generative Models

MASTER THESIS IN
MACHINE LEARNING FOR
COMPUTER VISION

CANDIDATE
Matteo Conti

SUPERVISOR
Prof. Samuele Salti

CO-SUPERVISOR
Andrea Amaduzzi

SESSION III
ACADEMIC YEAR 2022-2023

dedicated(X) :- friend(X).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Text-to-3D Generation

In today's world, there's a significant demand for 3D digital content across various fields, such as gaming, film, virtual reality, architecture, and robotics. Developing these models requires extensive artistic skills and knowledge in 3D design, making the training of skilled 3D modelers a challenging task. With the growing importance of 3D model creation, leveraging generative AI technology to produce high-quality and scalable 3D models has become crucial. Furthermore, AI models capable of converting text descriptions into 3D models are proving invaluable in empowering both beginners and experts in the field of 3D content creation.

Generating 3D models from textual descriptions poses more challenges than creating 2D images. 3D models are complex, consisting of unstructured and diverse forms of data that defy the straightforward application of conventional 2D deep learning techniques. Unlike the uniform structure of 2D images, 3D models can be represented through voxels, point clouds, meshes, and implicit functions, each requiring consideration of their unique representation for effective deep learning. The success of generating 3D models significantly depends on the chosen representation's ability to accurately convey geometry and topology. Additionally, while there are extensive datasets for text-to-image generation, the scarcity and lower quality of annotated 3D data sets hamper the development of text-to-3D technologies.

However, recent technological advancements have ushered in new opportunities for text-to-3D modeling. Neural Radiance Fields (NeRF) [21], for example, have shown promise in 3D reconstruction and view synthesis tasks, leveraging real-world photographs to train models that can interpolate between multiple viewpoints with high precision and consistency. NeRF's neural-based approach allows for high-resolution sampling and simpler optimization compared to traditional methods, effectively addressing the issue of 3D data scarcity. Moreover, the progress in diffusion models [8], driven by vast text-image pair datasets, has significantly advanced AI-generated content. By integrating these models with pre-trained text-to-image generators, new methodologies for optimizing 3D modeling have emerged, showcasing the synergy between different AI technologies in enhancing 3D content creation.

Overall, this work conducts the first yet comprehensive benchmark of **efficient** text-driven 3D generation models. The rest of this work is organized as follows:

- *Background* (chapter 2): introduces the mian components behind text-to-3D generation
- *Models* (chapter 4): presents the core idea of each model implemented in this work
- *Benchmarks* (chapter 3): summarizes the benchmarks and the metrics used to evaluate generated 3D shapes
- *Experiments* (chapter 5): summarizes the assumptions and the configurations on top of which each experiment is conducted

- *Results* (chapter 6): presents the results of each experiment and the comparison between the results obtained by other researchers

Efficient Generation While recent work on text-conditional 3D object generation has shown promising results, the state-of-the-art methods typically require multiple GPU-hours to produce a single sample. This is in stark contrast to state-of-the-art generative image models, which produce samples in a number of seconds or minutes [11]. In the other hand, comparing images and 3D shape generation is not fair because of the inherent complexity of 3D data itself. Considering the main purpose of this research is crucial to take into account this discrepancy in order to ”fairly” define which are the key aspects that we need to consider when evaluating the ”performances” of an **efficient** text-driven 3D generation model.

1.2 Evaluation

Issues

Recent methods in text-to-3D leverage powerful pre-trained diffusion models to exploit NeRF and Gaussian Splatting optimization techniques. Notably, these methods are able to produce high-quality 3D scenes without training on 3D data. Due to the open-ended nature of the task, most studies evaluate their results ”manually” through subjective case studies.

Data Representation Matters

Generative networks that use voxels and implicit representations require significant computational resources and struggle with detail fidelity, while the Marching Cubes [16] method also falls short in capturing intricate details. Point cloud representations, which use numerous 3D points to form shapes, suffer from uncertainty and ambiguity due to a lack of local and overall topological connections, affecting shape accuracy. Meshes offer a balance between geometry and topology but pose difficulties in generative network design due to their irregularity, impacting the detail of generated shapes.

Popular Evaluation Method

Most of the research relies on a process simplified as ”human survey”: authors of a given model X present their results to anonymous **human** users and ask them to rate the *quality* of the resulting 3D shape (related to a given text prompt) compared to the 3D shapes obtained with other methods or models developed by different researches. This process is repeated for a large number of prompts and models, and the results are then averaged to obtain a final score for each model. This final score is used to capture the *preference* of the human users for the 3D shapes generated by a given model X compared to the 3D shapes generated by other models. This approach presentes several issues regarding the *reliability* of the results. First, the process is *subjective* and *biased* by the human users’ preferences. Second, the process is *time-consuming* and *expensive* in terms of human resources. Third, the dataset of text prompts used to evaluate the results may be *biased* and *limited* to a specific domain or topic. Fourth, the process is *not scalable* since it since it requires a large number of human users and *not reproducible* since the human evaluators are not always the same subjects. Fifth, the process is *not objective* since it does not provide a clear and quantitative measure of the quality of the 3D shapes generated by a given model X.

Main Components

The considerations explained in section 1.2 present a challenge in quantitatively addressing the question: ”How has current progress in Text-to-3D gone so far?” [5].

Authors of the research T3Bench [5] proposed to conduct the evaluation of text-to-3D models using a benchmark that consists of two main components: *3D Quality* and *Prompt Coherence*. These components are used to evaluate the performance of text-to-3D models in terms of the *quality* of the generated 3D shapes and the *coherence* between the text prompts and the generated 3D shapes.

This approach tries to address the issue of finding a balance respect the *quality vs prompt-coherence* trade-off. Indeed, we can develop models which are able to generate high-quality 3D shapes but are not able to capture the semantic relationships between the input prompt and the 3D shape, or we can develop models which are able to capture the semantic relationships between the input prompt and the 3D shape but are not able to generate high-quality 3D shapes.

3D Quality

If we forget, just for a moment, about the fact that in our research we assume that all 3D shapes are generated starting from a text prompt, we can reason about how can we effectively evaluate the quality of the 3D shapes generated by a given model X while focusing only on the generated 3D shape itself. In principle, we don't need to look at the input prompt to determine if a given 3D shape has a good quality (e.g. presents fine-grained details, has a good topology, ...) or not (e.g. presents artifacts, coarse representation of the main features, ...). One of the most basic approach[1] to evaluate the quality of a generated 3D objects is to exploit existing algorithms used to assess the quality of generated images (e.g. *Inception Score (IS)*[35]) and metrics for quantifying the realism and diversity of images (e.g. *Fréchet Inception Distance (FID)*[7]).

[1] in order to exploit existing metrics for evaluating the quality of generated images, previous works simply apply these algorithms taking as input multiple renderings of their generated 3D objects

Prompt Coherence

The most challenging part of the evaluation process is to determine the *coherence* between the text prompts and the generated 3D shapes. This is a very though task since it requires to evaluate the *semantic* relationships between the input prompt (text) components and 3D shape (visual) features.

Input prompts may contain a wide range of information, from simple descriptions of the 3D shape to complex and abstract concepts like colors, materials, orientations, sourroundings description and *weird combinations* of these ones. *Weird combinations* may be considered as a combination of different concepts that are not usually found together in the real world, like a ”*red metallic elephant flying* in the *sky*”.

In chapter 3 we will two main approaches: the first one is based on exploit existing evaluation metrics for text-to-image generation models, while the second one is based on the use of a new evaluation research called *T3Bench* [5] which try to focus on defining a more *”general”* method to evaluate text-driven 3D generation results.

1.3 Our Approach

In this chapter we cover the main ”components” of our contribution, which summarize the main steps we have followed to conduct the first yet comprehensive benchmark of **efficient** text-driven 3D generation models.

The final list of the models that we have analyzed and implemented in this work is presented in Table 1.1.

Acronym	Date	Research Title
LucidDreamer [15]	2023/11/19	LucidDreamer: Towards High-Fidelity Text-to-3D Generation via Interval Score Matching
Cap3D [19]	2023/06/12	Scalable 3D Captioning with Pretrained Models
HiFA [48]	2023/05/30	HiFA: High-fidelity Text-to-3D Generation with Advanced Diffusion Guidance
ProlificDreamer [43]	2023/05/25	ProlificDreamer: High-Fidelity and Diverse Text-to-3D Generation with Variational Score Distillation
Shap-E [11]	2023/05/03	Shap-E: Generating Conditional 3D Implicit Functions
TextMesh [40]	2023/04/24	TextMesh: Generation of Realistic 3D Meshes From Text Prompts
Fantasia3D [2]	2023/03/24	Fantasia3D: Disentangling Geometry and Appearance for High-quality Text-to-3D Content Creation
Point-E [25]	2022/12/16	Point-E: A system for generating 3D point clouds from complex prompts
SJC [41]	2022/12/01	Score Jacobian Chaining: Lifting Pretrained 2D Diffusion Models for 3D Generation
Magic3D [17]	2022/11/18	Magic3D: High-Resolution Text-to-3D Content Creation
LatentNeRF [20]	2022/11/14	Latent-NeRF for Shape-Guided Generation of 3D Shapes and Textures
DreamFusion [29]	2022/09/29	DreamFusion: Text-to-3D using 2D Diffusion

Table 1.1: List of all Text-Driven 3D Generation models implemented and analyzed in this research. Models are sorted by date of publication.

Personal Contribution

The main contributions of this work are focused on collecting and implementing most recent text-driven 3D generation models. In particular, we focused on the **Efficient** 3D Generation task, by constraining the available hardware and the maximum generation time of each selected model.

We started by collecting and analyzing more than 40 of the most recent researches on text-driven 3D generation. For each of this research we focused on collecting and analyzing the following information:

- date of publication
- source code availability
- required hardware (e.g. number of GPUs, memory, etc.)
- generation time statistics (e.g. how long it takes to generate a single 3D shape)
- possible training/fine-tuning requirements (e.g. some models may require a pre-traning phase of some components of the architecture, or may require a fine-tuning phase on a specific dataset to generate high-quality 3D shapes)

In order to conduct a proper evaluation and analysis, we have been forced to filter out all of the works which satisfy the following criteria:

- research too old (i.e. published before the year 2022)
- source code not available
- missing specifications regarding the required hardware
- models which do not satisfy the efficiency contraints defined in section 5.1

Once we have obtained the most prominent researches by following the above steps, we then proceed to filter out all of the models which satisfy the following criteria:

- priors too complex to be integrated ”at scale” e.g. some models requires, in addition to the text prompt, also an image representing the ”sketch” of the 3D shape to be generate

- model checkpoints not available (for models which require a pre-training phase)
- source code too complex to be adapted in the time frame established for this work

Efficient Generation

For limited scope of this work, we have only characterized the *efficiency* of each model in terms of computational resources such as **hardware requirements** and **generation time** complexity.

Hardware Requirements These are the minimum hardware requirements to run the model and generate a single 3D shape. The hardware requirements are expressed in terms of the number of GPUs, the amount of memory (VRAM), and the type of GPU required to run the model.

Generation Time This is the time required to generate a single 3D shape. The generation time is expressed in terms of the number of seconds/minutes/hours required to generate a single 3D shape.

In chapter 5 we provide detailed information about the constraints that we have imposed on the hardware requirements and the generation time of each model. A summary of the hardware requirements and the generation time of each model analyzed in this work is presented in Table 1.2.

Implementation Details

Regarding the implementation of the selected models, we have started by collecting the source code of each model mainly available on the Github platform. Then, we have proceeded to adapt the source code to our needs, by fixing any possible bug and by creating a standard interface for the generation of 3D shapes.

Model	Gener. Time	Hardware
LucidDreamer [15]	35 minutes	1x NVIDIA A100
Cap3D-ShapE [19]	2 minutes	1x NVIDIA RTX3090
Cap3D-PointE [19]	4 minutes	1x NVIDIA RTX3090
HiFA [48]	100 minutes	1x NVIDIA A100
ProlificDreamer [43]	4 hours	1x NVIDIA A100
Shap-E [11]	1 minute	1x NVIDIA RTX3090
Fantasia3D [2]	45 minutes	1x NVIDIA A100
Point-E [25]	3 minutes	1x NVIDIA RTX3090
SJC [41]	25 minutes	1x NVIDIA A100
Magic3D [17]	40 minutes	1x NVIDIA A100
LatentNeRF [20]	65 minutes	1x NVIDIA A100
DreamFusion [29]	30 minutes	1x NVIDIA A100

Table 1.2: Generative models generation time and hardware specifications.

This interface is then used to generate 3D shapes on a large scale, by using the same text prompt dataset for each model. The results of the generation are then exported as 3D meshes, and then they have been used as input for our custom evaluation pipeline.

Finally, we have develop a standardized evaluation pipeline, which include all the metrics described in chapter 3. This pipeline is used to comprehensively evaluate a specific 3D shape generated starting from:

- a text prompt
- a model (selected from the ones implemented in this work)
- an optional prior configuration (please refers the chapter 4 to see which priors are available for each model)

Chapter 2

Background

2.1 Data Representation

3D data can have different representations [1], divided into Euclidean and non-Euclidean. 3D Euclidean data has a potential grid structure, which allows global parameterization and a common coordinate system. These properties make extending existing 2D deep learning paradigms to 3D data a simple task, where convolution operations remain the same as 2D. On the other hand, 3D non-Euclidean data does not have a grid array structure and is not globally parameterized. Therefore, extending classical deep learning techniques to such representations is a challenging task [13].

Euclidean Data

The Euclidean data preserves the attribute of the grid structure, with global parameterization and a common coordinate system. The major 3D data representations in this category include voxel grids and multi-view images.

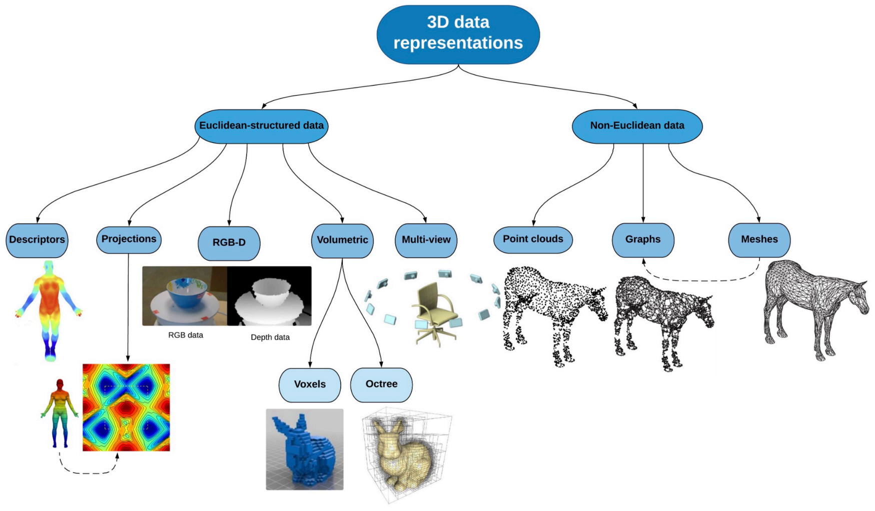


Figure 2.1: Hierarchy of several 3D data representations, grouped by Euclidean and non-Euclidean data.

Multi-View Images

The advancement of computer vision technology, combined with significant increases in computational power and the latest innovations in digital cameras, has made it simpler to capture extensive quantities of high-resolution images.

A multi-view image dataset compiles several images of an object or scene from various angles, such as the front, side, and top, to create a comprehensive collection. This is particularly valuable because acquiring 3D data from the real world can be a lengthy process, yet deep learning models require vast datasets for effective training. Thus, the primary benefit of multi-view image datasets lies in their ability to provide ample data. However, a limitation of multi-view images is that they don't exactly qualify as 3D model data. Nonetheless, they serve as a crucial intermediary between 2D and 3D visual representations. Recently, NeRF [21] has been introduced as an innovative technique for 3D reconstruction. It is particularly adept at meeting the substantial data needs of learning-based, generalizable NeRF methods that utilize extensive multi-view datasets [13].

Non-Euclidean Data

The second type of 3D data representation is non-Euclidean data. This type of data does not have global parametrization or common coordinate systems, which makes it difficult to extend 2D deep learning paradigms. Much effort has been made in learning this data representation and applying Deep Learning techniques.

Pointcloud

Point clouds consist of an unordered collection of discrete data points that represent the form of three-dimensional objects within a three-dimensional space. These data points are typically considered non-Euclidean due to their unstructured nature on a global scale. Nonetheless, they can be viewed as collections of small, globally parametrized Euclidean subsets [13]. The classification of a point cloud's structure hinges on the choice between focusing on its global or local attributes. In many cases, the emphasis is on capturing an object's overarching features for complex analyses, leading to the classification of point clouds as inherently non-Euclidean. Despite their straightforward acquisition, the uneven distribution of point clouds poses challenges for processing them using conventional 2D neural networks. Compared to voxel-based methods, point clouds offer a more precise depiction of three-dimensional shapes, as the 3D coordinates of the points directly convey the object's geometry. This efficiency and clarity have made point clouds a subject of interest among researchers.

Mesh

3D meshes [42] are a widely used form of representing 3D shapes. They consist of polygons, known as faces, which are defined by vertices that represent points in 3D space. These vertices are linked together based on a connectivity list, outlining their interconnections. Since meshes only represent the object's surface, they are relatively compact, facilitating the modeling of point relationships through the connectivity of surface points. However, when considering the broader perspective, 3D meshes represent non-Euclidean data. The local geometry of a mesh can be viewed as part of Euclidean space, but within this space, certain Euclidean properties, such as shift-invariance, operations in vector spaces, and universal parameterization systems, do not apply clearly [13]. This complexity makes deep learning applications for 3D meshes notably challenging [45].

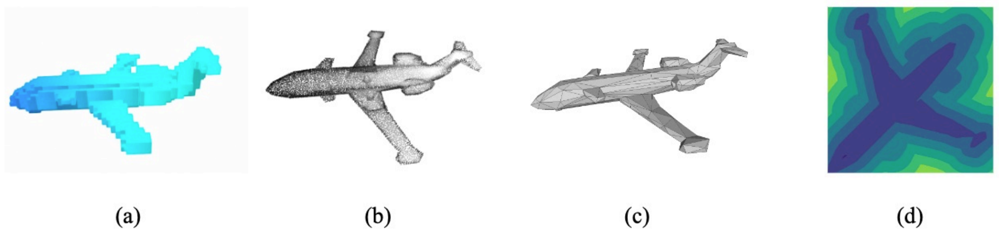


Figure 2.2: Different 3D shape representations, using: (a) voxels, (b) pointcloud, (c) mesh, (d) SDF. [45]

2.2 3D Learning

Recently, implicit neural representations (INRs) have become popular for encoding 3D assets. To represent a 3D asset, INRs typically map 3D coordinates to location-specific information such as density and color. In general, INRs can be thought of as resolution independent, since they can be queried at arbitrary input points rather than encoding information in a fixed grid or sequence.

In the following sections, we will review some of the most popular INRs, such as Neural Radiance Fields (NeRF), and Gaussian Splatting.

Neural Radiance Fields (NeRF)

NeRF. Neural Rediance Field (NeRF)[21] is a neural network-based implicit representation of 3D scenes, which can render projection images from a given viewpoint and a given position.

As presented in Figure 2.3, NeRF authors approach is based on the following steps:

(a) they synthesize images by sampling 5D coordinates (location and viewing direction) along camera rays

(b) they feed those locations into an MLP to produce a color and volume density

(c) they use volume rendering techniques to composite these values into an image

(d) **being this rendering function differentiable**, they can optimize their scene representation by minimizing the residual between synthesized and ground truth observed images

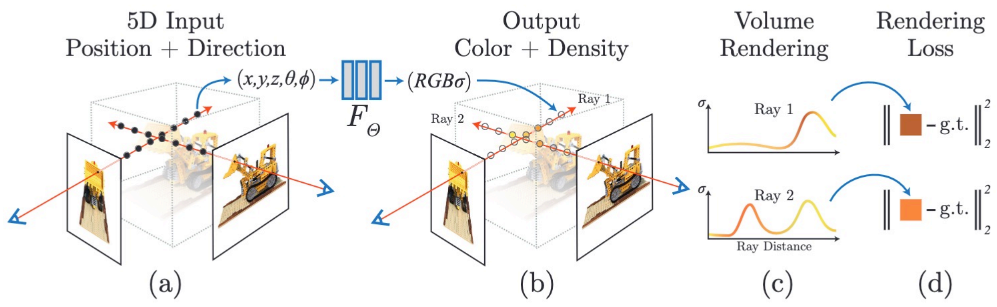


Figure 2.3: An overview of neural radiance field (NeRF) scene representation and differentiable rendering procedure.

Specifically, they represent a continuous scene as a 5D vector-valued function whose input is a 3D location $\mathbf{x} = (x, y, z) \in \mathbb{R}^3$ and 2D viewing direction $\mathbf{d} = (\theta, \phi) \in \mathbb{R}^2$ and whose output is an emitted color $\mathbf{c} = (r, g, b)$ and a volume density σ. Moreover, they express direction as a 3D Cartesian unit vector $\mathbf{d}$ and approximate this continuous 5D scene representation with an MLP network $F_\Theta : (\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma)$ and optimize its weights Θ to map from each input 5D coordinate to its corresponding volume density and directional emitted color.

Rendering of images from desired perspectives can be achieved by integrating color along a suitable ray $\mathbf{r} = \mathbf{o} + td$ (with near and far bounds t_n and t_f is) for each pixel in accordance with the volume rendering equation[21]:

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

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

The transmission coefficient $T(t)$ is defined as the probability that light is not absorbed from the near-field boundary t_n to t. In order to train NeRF network and optimize the predicted color $\hat{C}$ to fit with the ray $\mathcal{R}$ corresponding to the pixel in the training images, gradient descent is used to optimize the network and match the target pixel color by loss[21]:

$$\mathcal{L} = \sum_{r \in \mathcal{R}} ||C(r) - \hat{C}(r)||_2^2$$

In pratice, NeRF authors implemented a lot of additional steps[1] to make the training process feasible. The final result is a neural network that can render high-quality images of a 3D scene from any viewpoint, given a set of input images and their camera poses.

[1]such like hierarchical sampling, 3D positional encoding, approximating the continuos formulation of $\hat{C}$ with quadrature rule, implementing a loss strategy based on coarse+fine rendering,...

Gaussian Splatting

Gaussian Splatting[12] is a method for representing 3D scenes and rendering novel views. It can be thought of as an alternative to NeRF[21]-like models, and just like NeRF back in the day, Gaussian splatting led to lots of new research works that chose to use it as an underlying representation of a 3D object for various use cases.

Before diving into the details of Gaussian Splatting, it is important to remark that this method, in principle, do not involve any neural network at all. This technique can be simply thought of as a way to represent a 3D scene as a set of points in space.

Representation With this method, a 3D shape is represented with a set of 3D points, where each point is a 3D Gaussian with its own unique parameters that are fitted (per scene) such that renders of this object match closely to the known 3D scene priors (e.g. original formulation starts from a set of 2D images and their camera poses).
Each 3D Gaussian is parametrized by:

- Mean μ interpretable as location x, y, z;
- Covariance Σ;
- Opacity $\sigma(\alpha)$, a sigmoid function is applied to map the parameter to the $[0, 1]$ interval;
- Color parameters (e.g. (R, G, B) values for standard color representation, spherical harmonics (SH) coefficients, $\ldots$).

As for the covariance Σ, it is chosen to be anisotropic by design, that is, not isotropic. Practically, it means that a 3D point can be an ellipsoid rotated and stretched along any direction in space. It could have required 9 parameters, however, they cannot be optimized directly because a covariance matrix has a physical meaning only if it's a positive semi-definite matrix. Using gradient descent for optimization makes it hard to pose such constraints on a matrix directly, that is why it is factorized instead as follows:

$$\Sigma = RSS^TR^T$$

such factorization is known as eigendecomposition of a covariance matrix and can be understood as a configuration of an ellipsoid where:

- R is a 3x3 rotation matrix analytically expressed with 4 quaternions.
- S s a diagonal scaling matrix with 3 parameters for scale;

On one hand, each point effectively represents a limited area in space close to its mean μ, according to its covariance Σ. On the other hand, it has a theoretically infinite extent meaning that each Gaussian is defined on the whole 3D space and can be evaluated for any point.

Image Formation Given a set of 3D points, possibly, the most interesting part is to see how can it be used for rendering: it turns out that NeRFs[21] and Gaussian splatting share the same image formation model. For more details, please refer both to NeRF[21] and Gaussian Splatting[12] papers.

Rendering The *Image Formation* tells us how to get a color in a single pixel. Although, to render an entire image, it's still necessary to traverse through all the $H \times W$ rays, just like in NeRF, however, the process is much more lightweight because:

- for a given camera, each 3D point can be projected into 2D in advance, before iterating over pixels (hence, when a Gaussian is blended for a few nearby pixels, we won't need to re-project it over and over again);
- there is no MLP to be inferenced $H \times W \times P$ times for a single image, 2D Gaussians are blended onto an image directly;

- there is no ambiguity in which 3D point to evaluate along the ray, no need to choose a ray sampling strategy[2]: a set of 3D points overlapping the ray of each pixel is discrete and fixed after optimization;

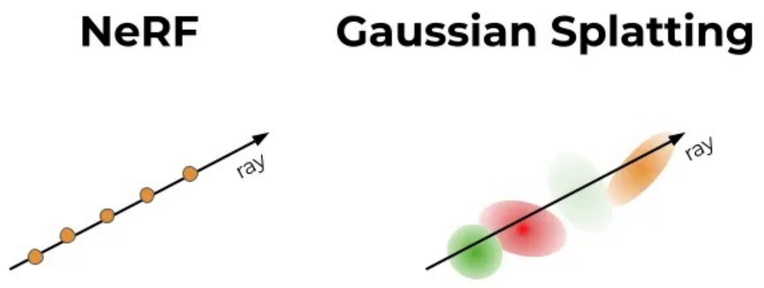


Figure 2.4: A conceptual difference between NeRF and Gaussian Splatting. (left) query a continuous MLP along the ray. (right) blend a discrete set of Gaussians relevant to the given ray .

Figure 2.4 shows a conceptual difference between NeRF and Gaussian Splatting: while NeRF queries a continuous MLP along the ray, Gaussian Splatting blends a discrete set of Gaussians relevant to the given ray.

Sorting Algorithm The last concept that we have to introduce is the creation of a *sorting algorithms* by Gaussian Splatting authors.

In practice, this algorithm does:

1. sort 3D points by *depth* (i.e. proximity to an image plane);
2. group them by tiles;

The first operation is needed to compute transmittance, and the latter allows to limit the weighted sum for each pixel to α-blending of the relevant 3D points only (or their 2D projections, to be more specific).

Thanks to sorting, the rendering of each pixel can be reduced to α-blending of pre-ordered points from the tile the pixel belongs to.

[2]as we have to do in NeRF-like models

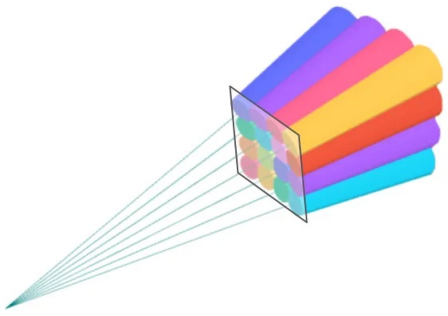


Figure 2.5: View frustums, each corresponding to a 16x16 image tile. Colors have no special meaning. The result of the sorting algorithm is a subset of 3D points within each tile sorted by depth.

2.3 Text-Driven 3D Generation

Recent breakthroughs in text-to-image synthesis have been driven by diffusion models trained on billions of image-text pairs. Adapting this approach to 3D synthesis would require large-scale datasets of labeled 3D data and efficient architectures for denoising 3D data, neither of which currently exist[29]. In the below sections we are going to how DreamFusion[29] authors circumvent these limitations by using a pretrained 2D text-to-image diffusion model to perform text-to-3D synthesis.

We we start by introducing the concept of diffusion models and how they can be used to generate 3D data. Then, we will introduce the concept of Score Distillation Sampling (SDS) and how it can be used to guide the generation of 3D data.

Diffusion Models

Diffusion models are inspired by non-equilibrium thermodynamics. They define a Markov chain of diffusion steps to slowly add random noise to data and then learn to reverse the diffusion process to construct desired data samples from the noise. When the diffusion consists of small amounts of Gaussian noise, it is sufficient to set the sampling chain transitions to conditional Gaussians too, allowing for a particularly simple neural network parameterization.

Forward Process In the forward process, diffusion model (DDPM) destroys the training data by gradually adding Gaussian noise. It starts from a data sample x_0 and iteratively generates noisier samples x_t with $q(x_t|x_{t-1})$ using a Gaussian diffusion kernel for integer timesteps $t \in [0, T]$:

$$q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1}) \qquad q(x_t|x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

Intuitively, this process gradually adds Gaussian noise to a signal, with the amount of noise added at each timestep determined by some noise (variance) schedule $\{\beta_t \in (0,1)\}_{t=1}^{T}$ (i.e. hyperparameter). Usually, the noise schedule is set such that, by the final timestep $t = T$, the sample x_T contains almost no information (i.e. it looks like Gaussian noise)[8].

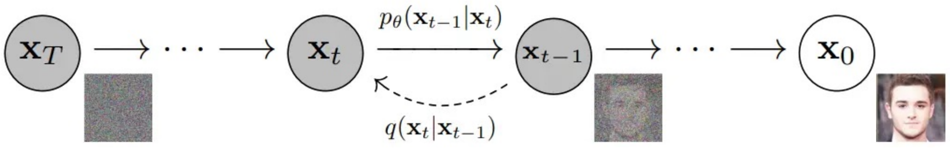

$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$\mathbf{x}_0$

Figure 2.6: The Markov chain of forward (reverse) diffusion process of generating a sample by slowly adding (removing) noise[8].

Reverse Process If we can reverse the above process and sample from $q(x_{t-1}|x_t)$, we will be able to recreate the true sample from a Gaussian noise input, $x_T \sim \mathcal{N}(0, I)$. To train a diffusion model, we approximate $q(x_{t-1}|x_t)$ as a neural network $p_\theta(x_{t-1}|x_t)$ We can then produce a sample by starting at random Gaussian noise x_T and gradually reversing the noising process until arriving at a noiseless sample x_0.

$$p_\theta(x_{0:T}) := p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \qquad p_\theta(x_{t-1}|x_t) := \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

More details regarding a possible conditioned generation and the so called *Conditional* and *Unconditional* Classifier Guidance can be found in the work of Dhariwal et al. [3] and Ho et al. [9].

Stable Diffusion Models

Latent diffusion model[3] (LDM)[34] runs the diffusion process in the latent space instead of pixel space, making training cost lower and inference speed faster. It has been motivated by the observation that most bits of an image contribute to perceptual details and the semantic and conceptual composition still remains after aggressive compression. LDM loosely decomposes the perceptual compression and semantic compression with generative modeling learning by first trimming off pixel-level redundancy with autoencoder and then manipulate/generate semantic concepts with diffusion process on learned latent.

First Stage The first stage train an encoder to produce latents $z = E(x)$ and a and a decoder to produce reconstructions $\tilde{x} = D(z)$. The encoder and decoder are trained in tandem to minimize a perceptual loss between $\tilde{x}$ and x, as well as a patchwise discriminator loss on $\tilde{x}$.

Second Stage After the first stage is completed, in the second stage a diffusion model is trained directly on encoded dataset samples. In particular, each dataset example x_i is encoded into a latent z_i , and then z_i is used as a training example for the diffusion model. To generate new samples, the diffusion model first generates a latent sample z, and then $D(z)$ (i.e. the decoder trained in the first stage) yields an image.

The diffusion and denoising processes happen on the latent vector z which has lower dimensionality than the original used image x (which is mapped in the pixel space). The denoising model is a time-conditioned U-Net, augmented with the cross-attention mechanism to handle flexible conditioning information for image

[3]usually called Stable Diffusion Model

generation (i.e. the textual input prompt in our scenario). A general architecture of the LDM is depicted in Figure 2.7.

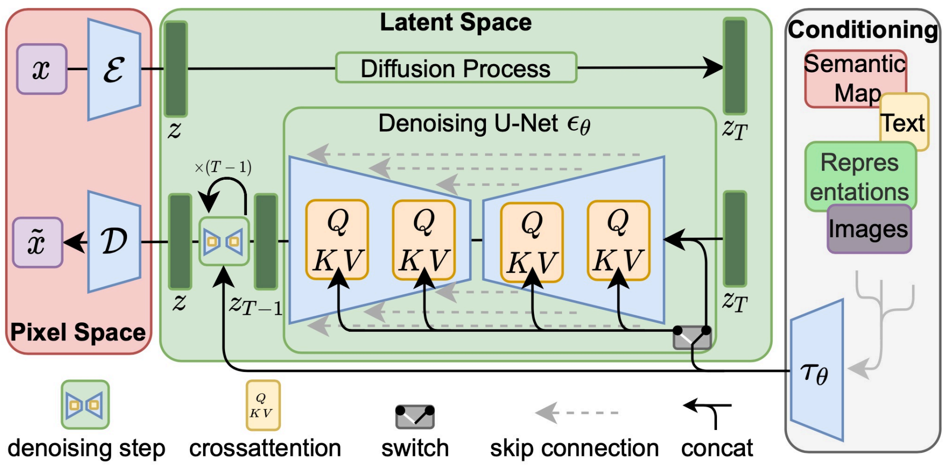


Figure 2.7: The architecture of latent diffusion model[34].

Score Distillation Sampling

Score Distillation is a method that enables using a diffusion model as a critic[4]. It has been introduced in DreamFusion[29] for guiding 3D generation. To perform score distillation, noise is first added to a given image (parametrized by the parameters of a differentiable generator). Then, the diffusion model is used to predict the added noise from the noised image. Finally, the difference between the predicted and added noises is used for calculating per-pixel gradients.

Diffusion Models Recap

Diffusion models[8] consist of a forward process q that slowly removes structure from data $\mathbf{x}$ by adding noise, and a reverse process or generative model p that slowly adds structure starting from noisisy data $\mathbf{z_t}$. This generative model p is trained to slowly add structure starting from random noise $p(\mathbf{z}_T) = \mathcal{N}(0, I)$ with transitions

[4]i.e. using it as a loss without explicitly back-propagating through the diffusion process

$p_\phi(\mathbf{z}_{t-1}|\mathbf{z}_t)$. Transitions are typically parameterized as $p_\phi(\mathbf{z}_{t-1}|\mathbf{z}_t) = q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x} = \hat{\mathbf{x}}_\phi(\mathbf{z}_t; t)$ where $q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x})$ is a posterior distribution derived from the forward process and $\hat{\mathbf{x}}_\phi(\mathbf{z}_t; t)$ is a learned approximation of the optimal denoiser. Training the generative model with a (weighted) evidence lower bound (ELBO) simplifies to a weighted denoising score matching objective for parameters ϕ[8]:

$$\mathcal{L}_{\text{Diff}}(\phi, \mathbf{x}) = \mathbb{E}_{t\sim\mathcal{U}(0,1),\epsilon\sim\mathcal{N}(0,I)} \left[w(t) \left\| \epsilon_\phi(\mathbf{z}_t; t) - \epsilon \right\|_2^2 \right] \tag{2.1}$$

where $w(t)$ is a weighting function that depends on the timestep t.

Authors have focused on building text-to-image diffusion models that learn $\epsilon_\phi(\mathbf{z}_t; t, y)$ conditioned on text embeddings y through the use of CFG[5][9] which jointly learns an unconditional model to enable higher quality generation via a guidance scale parameter:

$$\omega : \hat{\epsilon}_\phi(\mathbf{z}_t; y, t) = (1 + \omega)\epsilon_\phi(\mathbf{z}_t; y, t) - \omega\epsilon_\phi(\mathbf{z}; t)$$

SDS

DreamFusion[29] authors invesigated how existing approaches for sampling from diffusion models generate a sample that is the same type and dimensionality as the observed data the model was trained on [37]. Though conditional diffusion sampling enables quite a bit of flexibility, diffusion models trained on pixels have traditionally been used to sample only pixels.

They are not interested in sampling pixels[6]; they instead want to **create 3D models that look like good images when rendered from random angles**. Such models can be specified as a Differentiable Image Parameterization [22], where a differentiable generator g transforms parameters θ to create an image $\mathbf{x} = g(\theta)$.

For 3D, they let θ be parameters of a 3D volume and g a volumetric renderer. To

[5]Classifier-Free Guidance

[6]we are analyzing how we can exploit diffusion models to learn 3D data structures

learn these parameters, they required a loss function that can be applied to diffusion models.

Their approach leverages the structure of diffusion models to enable tractable sampling via optimization[7]. They optimized over parameters such that $\mathbf{x} = g(\theta)$ looks like a sample from the frozen diffusion model[8].

First attempt in reusing the diffusion training loss in Equation 2.1 to find modes of the learned conditional density $p(\mathbf{x}|y)$ results in minimizing the diffusion training loss with respect to a generated datapoint $\mathbf{x} = g(\theta)$ as:

$$\theta^{\star} = \text{argmin}_{\theta}\ \mathcal{L}_{\text{Diff}}(\phi, x = g(\theta))$$

In practice, they found that this loss function did not produce realistic samples. As a consequence, by furter investigating, they discovered that omitting the U-Net Jacobian term leads to an effective gradient for optimizing DIPs with diffusion models[9]:

$$\nabla_{\theta}\mathcal{L}_{\text{SDS}}(\phi, \mathbf{x} = g(\theta)) \triangleq \mathbb{E}_{t,\epsilon}\left[w(t)(\hat{\epsilon}_{\phi}(\mathbf{z}_t; y, t) - \epsilon)\frac{\partial \mathbf{x}}{\partial \theta}\right]$$

[7]i.e. a loss function that, when minimized, yields a sample

[8]to perform this optimization, we need a differentiable loss function where plausible images have low loss, and implausible images have high loss

[9]they found that the U-Net Jacobian term is expensive to compute since it requires backpropagating through the diffusion model U-Net

Chapter 3

Benchmarks

3.1 CLIP

Recent advances in multimodal learning have enabled the development of cross-modal matching models such as CLIP (Contrastive Language-Image Pre-training)[31] which learn shared representations from image-text pairs. This models is able to produce a scalar score that indicates whether an image and its associated caption match or not.

CLIP Similarity

The *CLIP Similarity* between an image and a caption is computed as the cosine similarity between the image and caption embeddings both respectively obtained from the visual and textual encoders of the CLIP[31] model. Hence, given an image with visual CLIP embedding v and a candidate caption with textual CLIP[31] embedding c, the *CLIP Similarity* is computed as:

$$\begin{aligned} \text{CLIP-Simil}(c, v) &= cos(c, v) \\ &= \frac{c \cdot v}{\max(||c||_2 \cdot ||v||_2,\ \epsilon)} \end{aligned}$$

where ϵ is a small constant to avoid division by zero.

Taking into account that CLIP model's weights are trained to maximize the scaled cosine similarity of truly corresponding image/caption pairs while simultaneously minimizing the similarity of mismatched image/caption pairs, the higher the similarity score, the more likely the caption and the image are semantically related.

CLIP Score

Authors of *CLIP-Score* [6] performed a numerical anaylis of the *CLIP-Similarity* metric and they discovered that, despite the cosine similarity, in theory, can range from $[-1, 1]$ they [6]:

1. never observed a negative cosine similarity
2. generally observed values ranging from roughly in the range of $[0, 0.4]$

Hence, also following the results obtained by another research [47], they defined the *CLIP-Score* as a rescaled version of the original CLIP-Similarity formulation. For an image with visual CLIP embedding v and a candidate caption with textual CLIP embedding c, they set a scaling factor w and formulated the *CLIP-Score* as:

$$\begin{aligned}\text{CLIP-Score}(c, v) &= w * \max(\text{CLIP-Simil}(c, v), 0)\\ &= w * \max(cos(c, v), 0)\end{aligned}$$

where $w = 2.5$.

This particular value of w attempts to stretch the range of the score distribution to $[0, 1]$. Authors justify this rescaling operation by arguing that:

> *while such a monotonic rescaling operation doesn't affect ranking results, for reporting purposes, it can be easier to compare raw values if*

they are on a scale more closely-aligned with other evaluation metrics (e.g., from roughly zero to roughly one). [6].

CLIP R-Precision

Park et al.[28] in 2021 introduced a new evaluation metric called *CLIP R-Precision*. This metric was designed to combine the goal of the *R-Precision* metric with the powerful image-text vector representations of the CLIP[31] model.

R-Precision Metric used to evaluate the correctness of the generated images with respect to the given caption. It calculates the top-R retrieval accuracy when retrieving the matching text t_p from T text candidates using the generated image i as a query. The retrieval function is based on a *similarity score* which should represent the semantic similarity between the image and the text respect to the context of the given task. Given a set of text non-matching textual candidates $t_1, t_2, \ldots, t_T \in T$, the original matching text t_p and the generated image i, the *R-Precision* is 1 if the matching text t_p is ranked in the top-R positions (based on the similarity score), otherwise it is 0.

CLIP R-Precision starts from the definition of the *R-Precision* metric and it extends it by replacing the original similarity function with the *CLIP-Similarity* in order to exploit the powerful image-text vector representations of the CLIP[31] model. In our scenario, text candidates are the prompts used to generate the 3D shapes and the images are the renderings extracted from the 3D objects themselves[1].

[1]considering that we render multiple images for each shape, the similarity score between a generated 3D object and the corresponding input prompt is computed as the maximum similarity score among all the possible combinations of (prompt, rendering$_i$)

3.2 T3Bench

As already introduced in section 1.2 evaluating the performance of text-to-3D models is a challenging task and, at the same time, the open-ended nature of the task makes it difficult to quantitatively assess the state of progress in Text-to-3D advancements.

For this reason, as we have already introduced in section 1.2, the authors of T3Bench [5] proposed a new benchmark for evaluating text-to-3D models in order to address the lack of a systematic approach in benchmarking current progress on text-to-3D methods [5]. In particular, they presented two main issues:

- a lack of a standard set of diverse, challenging test textual inputs [5].
- an absence of a set of comprehensive metrics to quantitatively measure the quality of the generated 3D scenes [5].

CLIP comparison T3Bench authors reported that several previous works assess 3D generation quality by rendering the generated 3D model into a single 2D image and measuring its alignment with the text prompt through CLIP metrics (e.g. similarity, score, r-precision, ...). Nevertheless, they highlight that these methods only consider one view of the 3D scene, failing to assess the overall 3D quality [5].

The authors introduced two automated evaluation metrics designed to account for multi-view data. One metric evaluates the subjective quality of created 3D scenes, while the other measures how well these scenes match the text descriptions provided. To compute these metrics, an initial step involves capturing a series of 2D images from various angles and focal points of the 3D scenes. The first metric (*Quality*) assigns scores to these images using text-image scoring models and aggregates these scores into a comprehensive quality measurement through regional convolution techniques. Conversely, the second metric (*Alignment*) employs multi-view captioning and analysis by a large language model (LLM) to determine how closely the 3D information aligns with the textual information in the input text prompt.

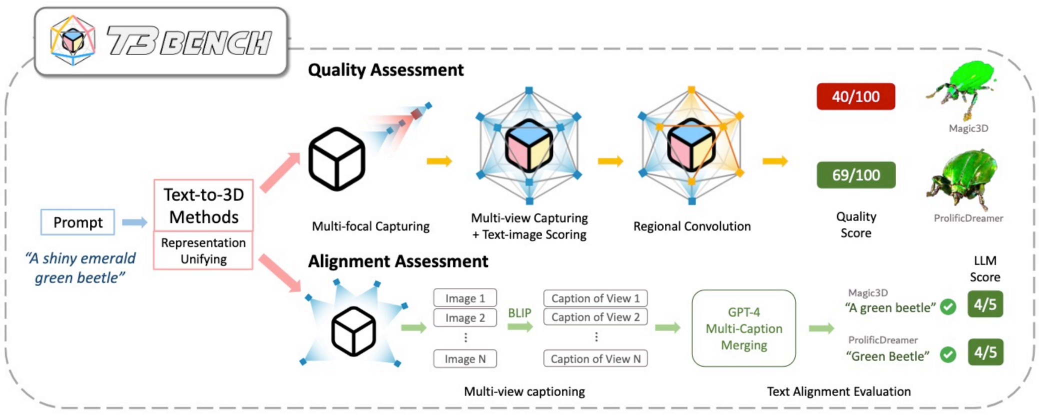
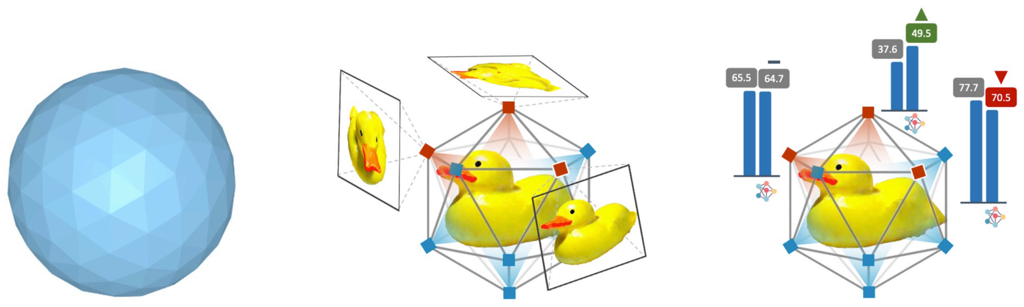


Figure 3.1: T3Bench: general overview of all the steps included in the evaluation pipeline of both *Alignment* and *Quality* metrics.

Quality Metric

T3Bench [5] authors highlight the importance of evaluating the quality of generated 3D scenes with a comprehensive approach due to the significance of spatial geometry information. They suggest that single viewpoint evaluation is insufficient and proposes a method that includes selecting appropriate viewpoints to better reflect the scene's quality and considering area coverage to examine global geometry and avoid biases towards locally optimal views. This method involves a detailed capturing and scoring procedure for a more reliable assessment of 3D quality. Figure 3.2 shows the main stages of this evaluation pipeline.
This method can be summarized in the following steps [5]:

1. **Multi-Focal Capturing**: authors employ five different focal lengths to capture renderings of the given mesh at each location in order deal with the issue of inappropriately choosing at priori too long or too short focal length values.
2. **Multi-View Capturing**: in order to capture the 3D scene as completely as



(a) 2-level Icosahedron (b) Multi-View Capturing (c) Regional Convolution

Figure 3.2: T3Bench Quality metric main components.

possible, the authors capture an image rendering from all the vertices of a level-2 icosahedron (which has 161 locations) built around the origin.

These first two steps produce as output a set of 5 images for each of the (161) locations of the icosahedron, for a total of 805 images foreach 3D object. Now we see how these images are scored and combined into a single overall quality measurement for the given 3D object.

The final steps are the following [5]:

3. **Multi-Focal Scoring**: at each location, they select the best rendering by scoring the 5 given images with a text-image scoring models (i.e. ImageReward [44]) and then they select the best one by taking the maximum score among these 5 (which represent the hypothetical best focal length for the given location).

4. **Scoring and Regional Convolution**: once they obtained the best rendering score for each location[2], they design a regional convolution mechanism to smooth out each score over its local region[3]. They treat the icosahedron as a graph composed of vertices and edges, and perform mean pooling on the

[2]161 scores, one foreach location of the icosahedron

[3]given a position of the icosahedron, its local region is defined as a set of N adjacent locations belonging to a pre-defined neighborhood function

graph with the following recursive formula[4]:

$$s_i^{(t+1)} = \frac{1}{|N(i) + 1|} \left(s_i^{(t)} + \sum_{j \in N(i)} s_j^{(t)} \right)$$

where $s_i^{(t)}$ is the score of point i on the icoshedron at t-th iteration, $N(i)$ is the set of neighboring points of i, and $|N(i)|$ is the number of neighbors of i. The superscript $(t + 1)$ denotes the score after the $(t + 1)$-th iteration[5].

After all these steps, T3Bench author select the highest score from all viewpoints as the final *quality score* for the 3D generation.

Quality Scores Normalization The authors normalize the final *quality score* from the range $[-2.5, 2.5]$ to the range $[0, 100]$ [5].

Alignment Metric

In addition to the evaluation from the quality aspect, the alignment between 3D semantic information and text is another crucial aspect that should be considered.

To measure the prompts alignment between different modalities, the authors utilize a 3D-to-text caption pipeline similar to Cap3D [19] to capture the 3D scene on the 12 locations of a level-0 icosahedron, each of which is captioned using a text-to-image model (i.e. BLIP [14]). Then, they employ GPT-4 [4] to merge these captions, resulting in the final 3D caption for the object [5].

[4]the goal of this formula is to smooth out the score over each local region since standard averaging of scores across all locations may not be appropriate, as most views, e.g., top or bottom, are not suitable for evaluation [5]

[5]authors choosed a total of $t = 3$ iterations as it ensures a balance between adequate smoothing and over-smoothing.[5]

Finally, they ask to GPT-4 [4] to rate the similarity between the computed caption described above and the textual prompt used for generating the 3D object. The rating score extracted from GPT-4 is used as the final *alignment score* for the 3D generation [5].

Alignment Score Normalization The authors normalize the final *alignment score* from the range $[1, 5]$ to the range $[0, 100]$ [5].

Chapter 4

Models

Recent methods for text-to-3D synthesis typically fall into one of two categories[25]:

1. **High Efficient Models**: methods which train generative models directly on paired (text, 3D) data or unlabeled 3D data. While these methods can leverage existing generative modeling approaches to produce samples efficiently, they are difficult to scale to diverse and complex text prompts due to the lack of large-scale 3D datasets.
2. **High Qualitative Models**: methods which leverage pre-trained text-image models to optimize differentiable 3D representations (section 2.3). These methods are often able to handle complex and diverse text prompts, but require expensive optimization processes to produce each sample (section 2.3). Furthermore, due to the lack of a strong 3D prior, these methods can fall into local minima which don't correspond to meaningful or coherent 3D objects.

4.1 High Efficient Models

Point-E

Starting from the two main categories presented in the initial paragraph of this chapter, authors of Point-E[25] propose to combine the benefits of both categories by pairing a text-to-image model with an image-to-3D model.

They propose to use a text-to-image model that leverages a large corpus of (text, image) pairs, allowing it to follow diverse and complex prompts, while, at the same time, to use an image-to-3D model trained on a smaller dataset of (image, 3D) pairs.

Then, To produce a 3D object from a text prompt, they first sample an image using the text-to-image model, and then sample a 3D object conditioned on the sampled image.

As we can see also in Table 1.2, **both of these steps can be performed in a number of seconds**, and do not require expensive optimization procedures.

Method

They break the generation process into three steps[25]:

1. they generate a synthetic view conditioned on a text caption by exploiting a 3-billion parameter GLIDE[24] model finetuned on rendered 3D models from their dataset
2. they produce a coarse pointcloud[1] (section 2.1) conditioned on the synthetic view by using a conditional, permutation invariant diffusion model (section 2.3)
3. they produce a fine pointcloud[2] (section 2.1) conditioned on the low-resolution pointcloud (step 2) and the synthetic view (step 1) by using a similar (but

[1] 1,024 points
[2] 4,096 points

smaller) diffusion model which is additionally conditioned on the low-resolution pointcloud

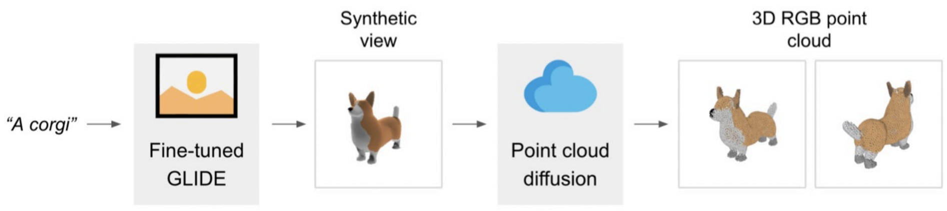


Figure 4.1: Point-E method overview.

Conversely to most of the models presented in this work, they **train their models on a dataset of several million 3D models and associated metadata**. They process the dataset into rendered views, text descriptions, and 3D point clouds with associated RGB colors for each point.
An overview of the Point-E method is illustrated in Figure 4.1.

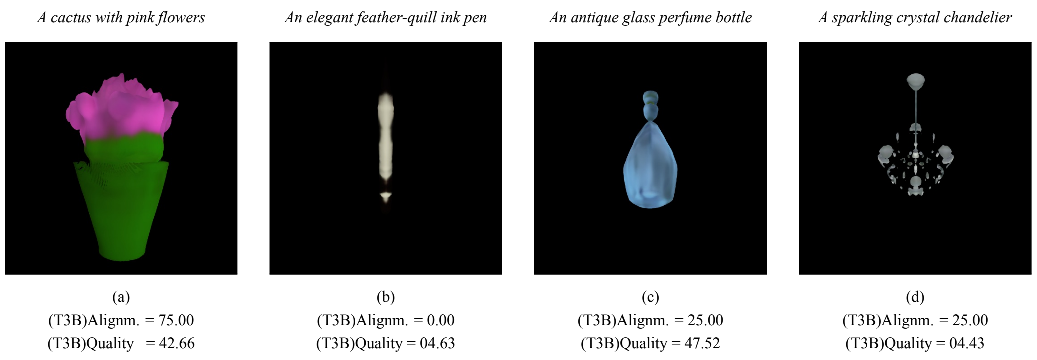


Table 4.1: Point-E: (a) prompt with the highest T3Bench Alignment score, (b) prompt with the lowest T3Bench Alignment score, (c) prompt with the highest T3Bench Quality score, (d) prompt with the lowest T3Bench Quality score.

Shap-E

Authors of *Shap-E* [11] proposed a novel approach to generate diverse and complex 3D implicit representations. They combined and scaled up several of the existing approaches to arrive at *Shap-E*, a conditional generative model for diverse and complex 3D implicit representations.

First, they trained a Transformer-based encoder to produce INR parameters for 3D assets. Next, they trained a diffusion model on outputs from the encoder. Unlike previous approaches, they produced INRs which represent both NeRF[21] (section 2.2) and meshes simultaneously, allowing them to be rendered in multiple ways or imported into downstream 3D applications[11].

Authors have also highlighted that, compared to *Point-E*[25] (section 4.1), their models converge faster and obtain comparable or superior results while sharing the same model architecture, datasets, and conditioning mechanisms [11].

Method

Before analyze in details the method, it is important to remark that **in this work they trained all of their models on a large dataset of 3D assets with corresponding renderings, point clouds, and text captions**. This approach goes in the opposite direction of most of the other models presented in this work, as we discuss in the following sections.

In their method, they first train an encoder to produce implicit representations, and then train diffusion models on the latent representations produced by the encoder. This method proceeds in two steps:

1. they train an encoder to produce the parameters of an implicit function given a dense explicit representation of a known 3D asset; in particular, the encoder produces a latent representation of a 3D asset which is then linearly projected to obtain weights of a multi-layer perceptron (MLP);
2. they train a diffusion prior on a dataset of latents obtained by applying the encoder to their dataset; this model is conditioned on either images or text

descriptions;

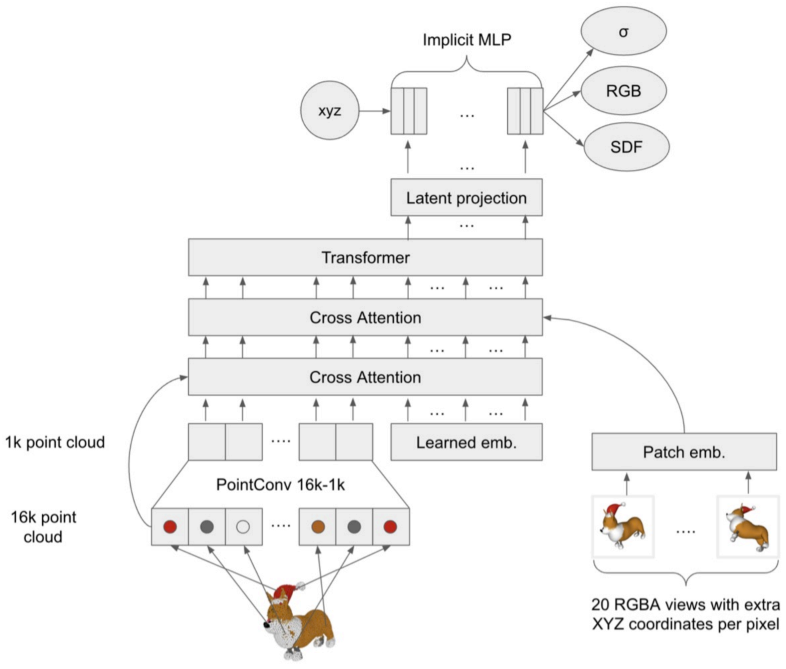


Figure 4.2: Shap-E encoder overview.

An overview of the Shap-E encoder architecture is illustrated in Figure 4.2.

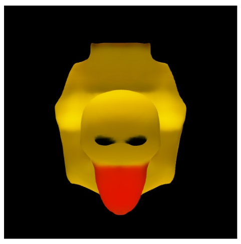
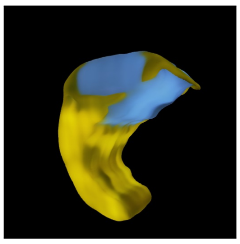
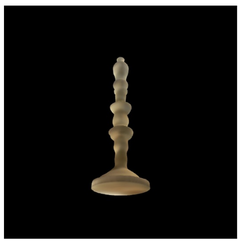
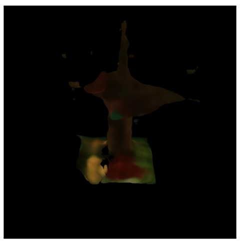
A bright, yellow rubber duck	*A sparkling diamond ring in a velvet box*	*An ivory candlestick holder*	*A chameleon perched on a tree branch*
			
(a) (T3B)Alignm. = 25.00 (T3B)Quality = 4.50	(b) (T3B)Alignm. = 100.0 (T3B)Quality = 53.60	(c) (T3B)Alignm. = 25.00 (T3B)Quality = 63.73	(d) (T3B)Alignm. = 25.00 (T3B)Quality = 4.50

Table 4.2: Shap-E: (a) prompt with the highest T3Bench Alignment score, (b) prompt with the lowest T3Bench Alignment score, (c) prompt with the highest T3Bench Quality score, (d) prompt with the lowest T3Bench Quality score.

Cap3D

Authors of *Cap3D* [19] proposed a method to automate 3D object annotation. Their key insight is to leverage the abundance of knowledge in pretrained image-text models to remedy the lack of existing 3D-text data.

The core of their data collection process is to apply an image captioning model (BLIP [14]) to a set of 3D asset renders, use an image-text alignment model (CLIP [31]) to filter captions, and apply a language model (GPT4 [4]) to fuse the filtered captions across views.

Since they have released their new created annotated 3D dataset, following their experiments, we have tested the improvements gathered by this new dataset by finetuning[3] two of our already implemented models: *Point-E*[25] and *Shap-E*[11].

We will refer to these finetuned versions as *Cap3D-Point-E* and *Cap3D-Shap-E*.

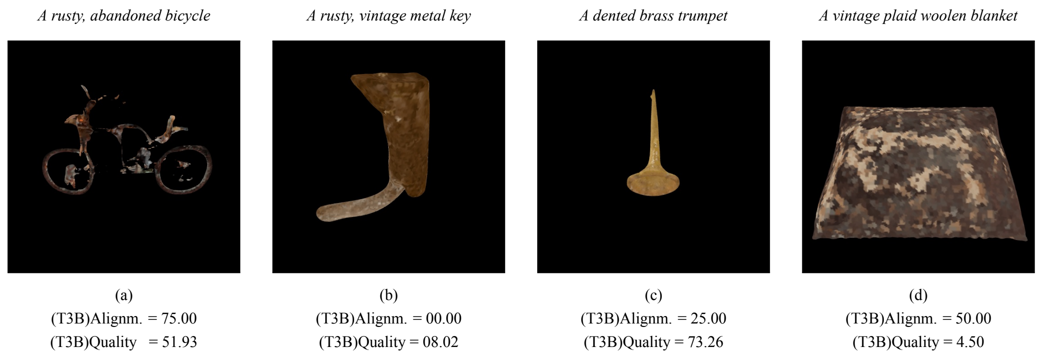


Table 4.3: Cap3D-Point-E: (a) prompt with the highest T3Bench Alignment score, (b) prompt with the lowest T3Bench Alignment score, (c) prompt with the highest T3Bench Quality score, (d) prompt with the lowest T3Bench Quality score.

[3]through the adaption of *ControlNet*[46] and *LORA*[10] for Stable Diffusion finetuning

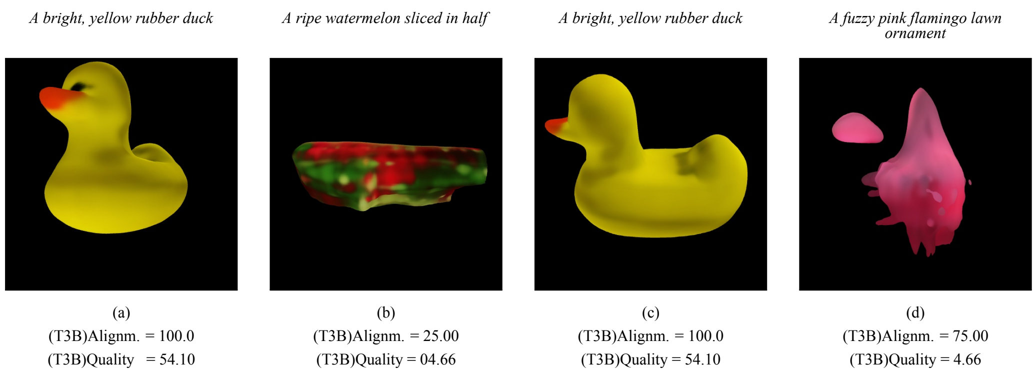


Table 4.4: Cap3D-Shap-E: (a) prompt with the highest T3Bench Alignment score, (b) prompt with the lowest T3Bench Alignment score, (c) prompt with the highest T3Bench Quality score, (d) prompt with the lowest T3Bench Quality score.

4.2 High Qualitative Models

DreamFusion

As introduced in section 2.3, DreamFusion[29] introduced Score Distillation Sampling (SDS) with the goal of creating a method in which a pre-trained diffusion model can be used as a loss within a generic continuous optimization problem to generate 3D assets from text. Since the diffusion model directly predicts the update direction, they did not need to backpropagate through the diffusion model; the model simply acts like an efficient, frozen critic that predicts image-space edits[29].

To synthesize a scene from text, they initialized a NeRF-like model[4] with random weights, then they repeatedly render views of that NeRF[21] from random camera positions and angles, using these renderings as the input to their score distillation loss function that wraps around the difussion model. As they demonstrated, simple gradient descent with this approach eventually results in a 3D model (parameterized as a NeRF[21]) that resembles the text[29].

[4]more details can be found in section 2.2

Method

For each text prompt, they train a randomly initialized NeRF from scratch. Each iteration of DreamFusion optimization performs the following:

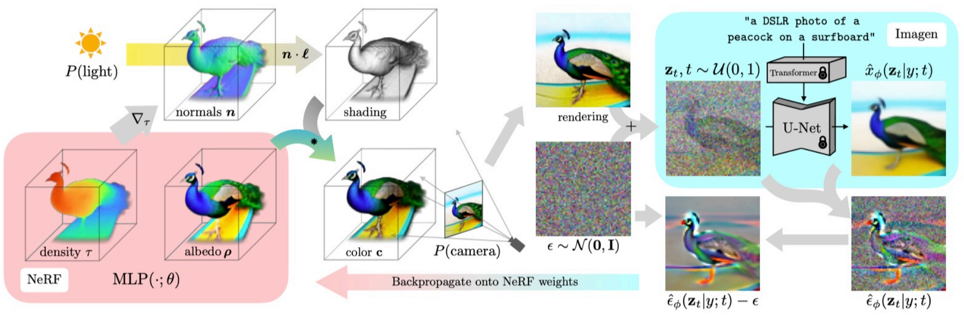

P(light)
$n \cdot \ell$
normals n
shading
∇_τ
density τ
albedo ρ
NeRF
MLP$(\cdot;\theta)$
color c
P(camera)
rendering
$\epsilon \sim \mathcal{N}(0, I)$
Backpropagate onto NeRF weights
"a DSLR photo of a peacock on a surfboard"
Imagen
$z_t, t \sim \mathcal{U}(0,1)$
Transformer
U-Net
$\hat{x}_\phi(z_t|y;t)$
$\hat{\epsilon}_\phi(z_t|y;t) - \epsilon$
$\hat{\epsilon}_\phi(z_t|y;t)$

Figure 4.3: DreamFusion method overview.

1. Randomly sample a camera and light
2. Render an image of the NeRF from that camera and shade with the light
3. Diffuse the rendering and reconstruct it with a (frozen) conditional diffusion model to predict the injected noise $\hat{\epsilon}_\phi(z_t|y;t)$[5]
4. compute gradients of the SDS loss with respect to the NeRF parameters
5. update the NeRF parameters using an optimizer

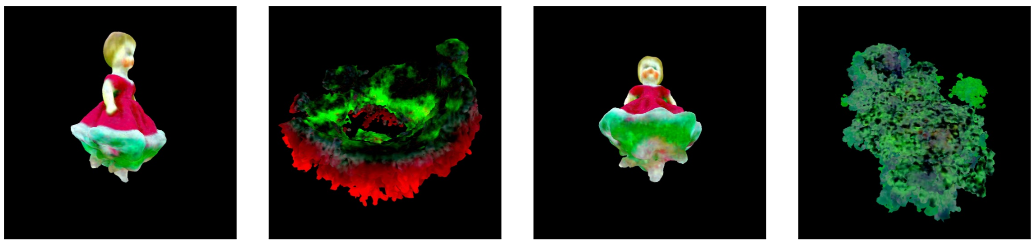
An overview of the DreamFusion method is illustrated in Figure 4.3.

Latent-NeRF

Authors of LatentNeRF[20] adapted the Score Distillation (section 2.3) introduced by DreamFusion[29] to the computationally efficient Latent Diffusion Models, which apply the entire diffusion process in a compact latent space of a pretrained autoencoder. As NeRFs operate in image space, a naive solution for guiding it with latent score distillation would require encoding to the latent space at each guidance

[5]this contains structure that should improve fidelity, but is high variance, hence subtracting the injected noise ϵ produces a low variance update direction $(\hat{\epsilon}_\phi - \epsilon)$ that is backpropagated through the rendering process to update the NeRF MLP parameters[29].

A vintage porcelain doll with a frilly dress	*A velvet cushion stitched with golden threads*	*A vintage porcelain doll with a frilly dress*	*A vintage plaid woolen blanket*



(a)	(b)	(c)	(d)
(T3B)Alignm. = 75.00	(T3B)Alignm. = 25.00	(T3B)Alignm. = 75.00	(T3B)Alignm. = 50.00
(T3B)Quality = 79.70	(T3B)Quality = 04.48	(T3B)Quality = 79.70	(T3B)Quality = 04.43

Table 4.5: DreamFusion(IF): (a) prompt with the highest T3Bench Alignment score, (b) prompt with the lowest T3Bench Alignment score, (c) prompt with the highest T3Bench Quality score, (d) prompt with the lowest T3Bench Quality score.

step[20]. Instead, they propose to bring the NeRF to the latent space, resulting in a model so called *Latent-NeRF*.

Method

First, instead of representing their NeRF[21] (section 2.2) in the standard RGB space, they propose a model which operates directly in the latent space of the LDM[6]. Secondly, they show that after training, one can easily transform a Latent-NeRF back into a regular NeRF in order to enable further refinement in RGB space[20].

An overview of the Latent-NeRF method is illustrated in Figure 4.4. The training process can be summarized as follows[20]:

1. they render the scene from a random view point to produce a feature map z
2. z is noised with ϵ according to a random diffusion step t.
3. The noised version of z (i.e., x_t) is denoised using Stable Diffusion[34], with the input text prompt

[6]Latent Diffusion Model

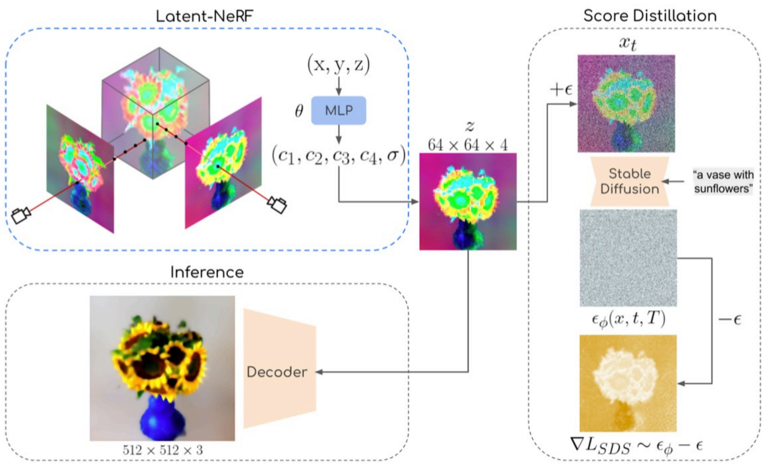


Figure 4.4: Latent-NeRF method overview.

4. the input noise is subtracted from the predicted noise[7] to approximate per-pixel gradients that are back propagated to the NeRF representation

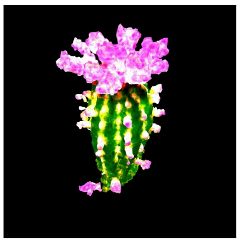
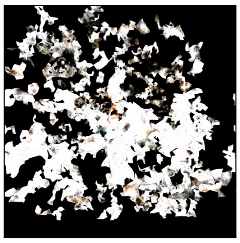
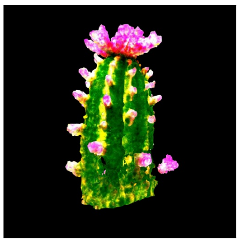
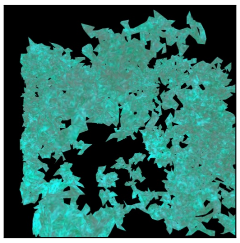
A cactus with pink flowers	*A sleek stainless steel teapot*	*A cactus with pink flowers*	*A gleaming silver saxophone*
			
(a)	(b)	(c)	(d)
(T3B)Alignm. = 75.00	(T3B)Alignm. = 00.00	(T3B)Alignm. = 75.00	(T3B)Alignm. = 25.00
(T3B)Quality = 84.99	(T3B)Quality = 04.43	(T3B)Quality = 84.99	(T3B)Quality = 04.44

Table 4.6: Latent-NeRF: (a) prompt with the highest T3Bench Alignment score, (b) prompt with the lowest T3Bench Alignment score, (c) prompt with the highest T3Bench Quality score, (d) prompt with the lowest T3Bench Quality score.

Magic3D

DreamFusion[29] has demonstrated the utility of a pre-trained text-to-image diffusion model to optimize Neural Radiance Fields (NeRF)[21], achieving remarkable

[7]extracted from the Stable Diffusion Model [34]

text-to-3D synthesis results.

However, authors of *Magic3D* highlighted that the method has two inherent limitations[17]:

1. extremely slow optimization of NeRF
2. low-resolution image space supervision on NeRF[8]

These choices result in two key limitations[17]:

1. high-resolution geometry or textures cannot be obtained[9]
2. the utility of a large global MLP for volume rendering is both computationally expensive as well as memory intensive[10]

Method

Following the limitations in the DreamFusion[29] method, *Magic3D* authors address these limitations by utilizing a two-stage optimization framework.

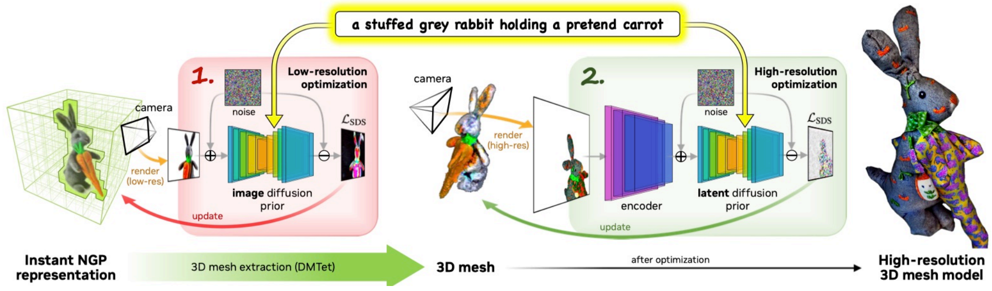

a stuffed grey rabbit holding a pretend carrot
1.
Low-resolution optimization
noise
camera
render (low-res)
image diffusion prior
$\mathcal{L}_{SDS}$
update
2.
High-resolution optimization
camera
render (high-res)
encoder
latent diffusion prior
update
Instant NGP representation
3D mesh extraction (DMTet)
3D mesh
after optimization
High-resolution 3D mesh model

Figure 4.5: Magic3D method overview.

First, they obtain a coarse model using a low-resolution diffusion prior and accelerate with a sparse 3D hash grid structure[11] (i.e. two single-layer neural networks,

[8]leading to low-quality 3D models with a long processing time.
[9]since the diffusion model only operates on 64×64 images.
[10]making this approach scale poorly with the increasing resolution of images.
[11]this hash grid encoding[23] allows them to represent high-frequency details at a much lower computational cost.

one predicting albedo and density and the other predicting normals).

Then, using the coarse representation as the initialization, they further optimize a textured 3D mesh model with an efficient differentiable renderer interacting with a high-resolution latent diffusion model. Moreover, while rendering the mesh, they increased the focal length to zoom in on object details, which is a critical step towards recovering high-frequency details[17].

An overview of the Magic3D method is illustrated in Figure 4.5.

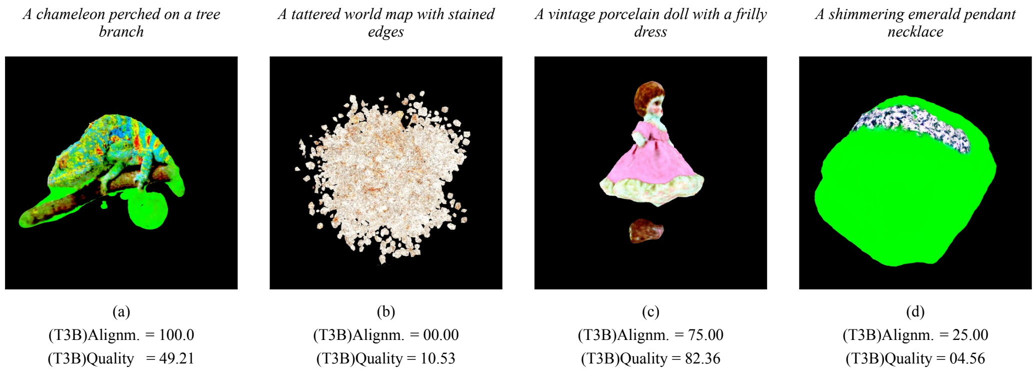


Table 4.7: Magic3D(IF): (a) prompt with the highest T3Bench Alignment score, (b) prompt with the lowest T3Bench Alignment score, (c) prompt with the highest T3Bench Quality score, (d) prompt with the lowest T3Bench Quality score.

SJC

Since diffusion models learns to predict a vector field of gradients, authors of *SJC*[41] to apply chain rule on the learned gradients and back-propagate the score of a diffusion model through the Jacobian of a differentiable renderer, which they instantiate to be a voxel radiance field. This setup aggregates 2D scores at multiple camera viewpoints into a 3D score, and repurposes a pretrained 2D model for 3D data generation.

Method

SJC[41] introduce a method that converts a pretrained 2D diffusion generative model on images into a 3D generative model of radiance fields, without requiring access to any 3D data. They highligh how the key insight is to interpret diffusion models as learned predictors of a gradient field[12], often referred to as the score function of the data log-likelihood.

Authors named their method *Score Jacobian Chaining (SJC)* as they apply the chain rule on the estimated score, which is the gradient of the log-density function with respect to the data.

Diffusion Model Diffusion models can be interpreted as modeling $\nabla_x \log p_\sigma(x)$, i.e. the denoising score at noise level σ. Generating a sample from a diffusion model involves repeated evaluations of the score function from large to small σ level, so that a sample x gradually moves closer to the data manifold. Despite other perspective, here they are primarily motivated from the viewpoint that diffusion models produce a gradient field.

They investigated the possibility of applying the chain rule to the learned gradients. Consider a diffusion model on images. An image x may be parameterized by some function f with parameters θ, i.e., $x = f(\theta)$. Applying the chain rule through the Jacobian $\frac{\partial x}{\partial \theta}$ converts a gradient on image x into a gradient on the parameter θ [41].

Their method uses differentiable rendering to aggregate 2D image gradients over multiple viewpoints into a 3D asset gradient, and lifts a generative model from 2D to 3D. They parameterize a 3D asset θ as a radiance field stored on voxels and choose f to be the volume rendering function.

[12]often referred to as the score function of the data log-likelihood

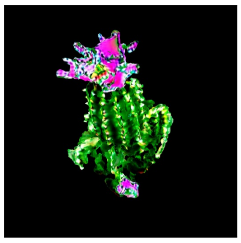
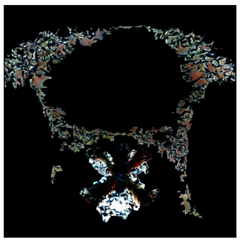
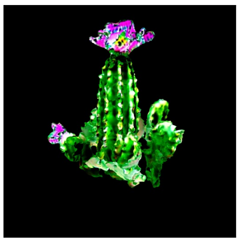
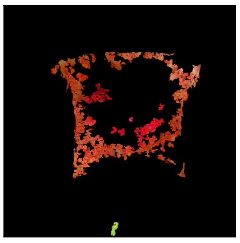
A cactus with pink flowers	*A pirate flag with skull and crossbones*	*A cactus with pink flowers*	*A vibrant, handmade patchwork quilt*
			
(a)	(b)	(c)	(d)
(T3B)Alignm. = 75.00	(T3B)Alignm. = 00.00	(T3B)Alignm. = 75.00	(T3B)Alignm. = 75.00
(T3B)Quality = 72.71	(T3B)Quality = 06.49	(T3B)Quality = 72.68	(T3B)Quality = 04.46

Table 4.8: SJC: (a) prompt with the highest T3Bench Alignment score, (b) prompt with the lowest T3Bench Alignment score, (c) prompt with the highest T3Bench Quality score, (d) prompt with the lowest T3Bench Quality score.

Fantasia3D

Author of *Fantasia3D* [2] propose a new method for high-quality text-to-3D content creation. Key to Fantasia3D is the disentangled modeling and learning of geometry and appearance. For geometry learning, they rely on a hybrid scene representation, and propose to encode surface normal extracted from the representation as the input of the image diffusion model. For appearance modeling, they introduce the spatially varying bidirectional reflectance distribution function (BRDF) into the text-to-3D task, and learn the surface material for photorealistic rendering of the generated surface.

This is in contrast to existing text-to-3D methods, which commonly use implicit scene representations, coupling the geometry and appearance via volume rendering resulting in being suboptimal in terms of recovering finer geometries and achieving photorealistic rendering[2].

Method

Authors, focused on disentangling the processes of modeling geometry and appearance. An overview of the Fantasia3D method is illustrated in Figure 4.6.

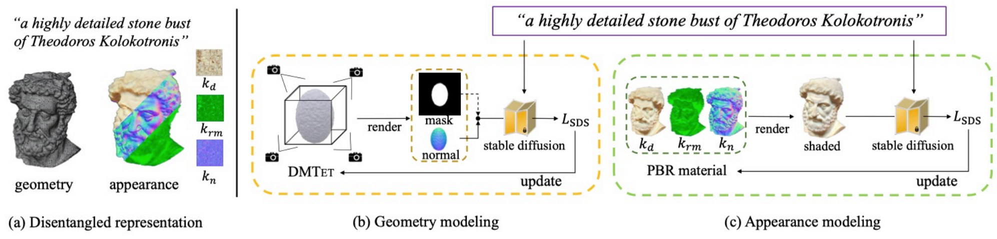


Figure 4.6: Fantasia3D method overview.

Geometry Modeling They utilized a hybrid surface representation called DMTET[36] in order to allow explicit shape control and geometry learning. Moreover, for the geometry learning process, they introduced the concept of encoding a rendered normal map as the input for shape encoding [13].

Appearance Modeling They introduced the full Bidirectional Reflectance Distribution Function (BRDF) into text-to-3D content creation, enabling the learning of surface materials for photorealistic rendering. This is a first in text-to-3D generation, facilitating the creation of high-quality 3D objects with realistic materials and textures.

Learning and Optimization Both geometry and appearance models are optimized using a loss derived from Score Distillation Sampling (section 2.3), back-propagated through a pre-trained image diffusion model, specifically stable diffusion. Moreover, the approach supports flexible initialization, either as a 3D ellipsoid or a customized 3D model provided by users, enabling control over the starting point of the generation process [2].

TextMesh

Authors of *TextMesh* [40] highlighted the main downsides of the DreamFusion [29] method:

[13]this is in contrasts with conventional approaches that typically encode color images

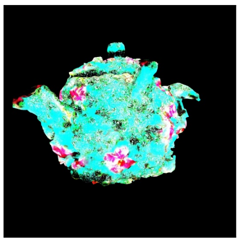
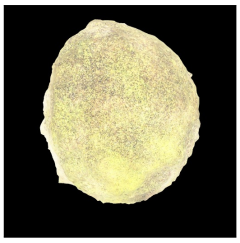
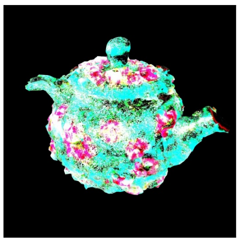
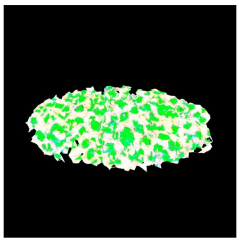
A ceramic teapot with floral patterns	*A silver mirror with ornate detailing*	*A ceramic teapot with floral patterns*	*A shimmering emerald pendant necklace*
			
(a)	(b)	(c)	(d)
(T3B)Alignm. = 75.00	(T3B)Alignm. = 00.00	(T3B)Alignm. = 75.00	(T3B)Alignm. = 50.00
(T3B)Quality = 62.90	(T3B)Quality = 04.69	(T3B)Quality = 62.90	(T3B)Quality = 04.40

Table 4.9: Fantasia3D: (a) prompt with the highest T3Bench Alignment score, (b) prompt with the lowest T3Bench Alignment score, (c) prompt with the highest T3Bench Quality score, (d) prompt with the lowest T3Bench Quality score.

1. it tends to produce objects with over-saturated colors due to the strong guidance required to make the model converge
2. it represents the 3D scene in the form of a Neural Radiance Field (NeRF)[21] (section 2.2), which renders the approach impractical to be used within standard computer graphics pipelines

Hence, they propose a novel method (i.e. *TextMesh*) for 3D shape generation from text prompts, targeted at generating photorealistic 3D content in the form of standard 3D meshes. They argue that their generated 3D meshes significantly improve upon the DreamFusion method for realism and can be directly utilized within standard computer graphics pipelines and applications in AR or VR.

In summary, the authors of *TextMesh* propose the following contributions [40]:

1. they modify DreamFusion to model radiance in the form of a signed distance function (SDF), to tailor the model towards mesh extraction
2. they propose a novel multi-view consistent and mesh conditioned re-texturing, enabling the generation of photorealistic 3D mesh models

3. they experimentally show that their obtained meshes are geometrically of high quality and showcase more natural textures than the current state-of-the-art, whilst being ready to be deployed into pre-existing graphics pipelines

Method

To accomplish the above mentioned contributions, *TextMesh* modifies the Dream-Fusion method to model radiance in the form of a signed distance function (SDF), allowing by design easy extraction of the surface as the 0-level set of the obtained volume. Furthermore, in an effort to enhance the mesh quality, they retexture the output by leveraging another diffusion model conditioned on color and depth from the mesh. To this end, they render the object from multiple viewpoints and use diffusion to guide texture optimization to enhance realism and details. Nevertheless, when processing individual views independently, the refined texture exhibits severe inconsistencies. Therefore, they propose to run several views simultaneously through the diffusion model instead. To obtain the final texture, they then train on the produced output views together with Score Distillation Sampling to ensure smooth transitions.

Method steps are illustrated in Figure 4.7:

1. (top left) Given the input text prompt, they train their initial distance field using Score Distillation Sampling (SDS) with view-dependent text prompting and an Diffusion prior
2. (top right) then, they extract the mesh with marching cubes but, since the obtained appearance lacks details and the colors tend to be oversaturated, they render the color and depth from four orthogonal views of their mesh
3. (bottom right) and run these views jointly through StableDiffusion to generate photorealistic and 3D consistent views of their mesh

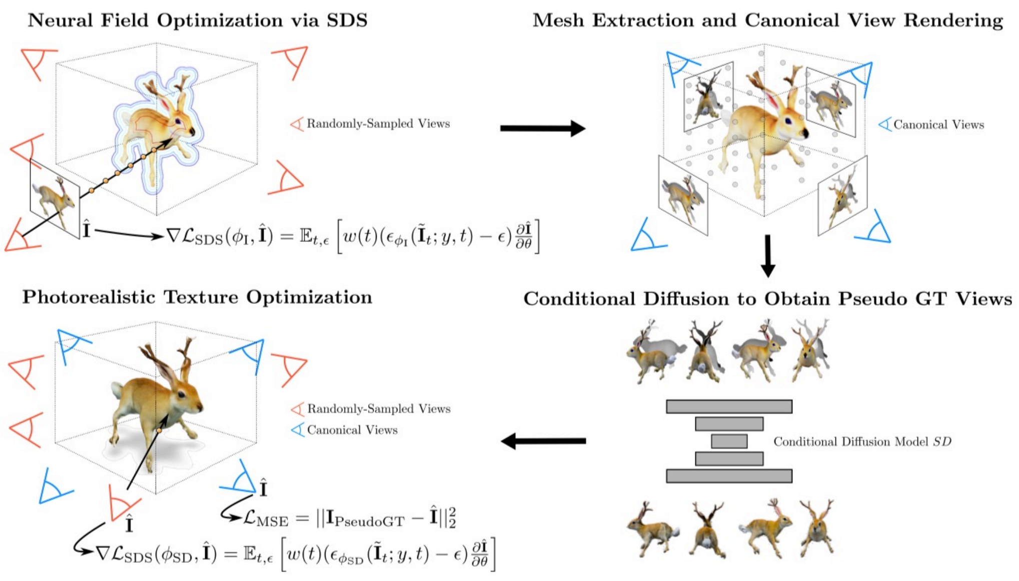

Neural Field Optimization via SDS
Randomly-Sampled Views
$\hat{\mathbf{I}}$
$\nabla\mathcal{L}_{\text{SDS}}(\phi_{\text{I}}, \hat{\mathbf{I}}) = \mathbb{E}_{t,\epsilon}\left[w(t)(\epsilon_{\phi_{\text{I}}}(\tilde{\mathbf{I}}_t; y, t) - \epsilon)\frac{\partial\hat{\mathbf{I}}}{\partial\theta}\right]$
Mesh Extraction and Canonical View Rendering
Canonical Views
Conditional Diffusion to Obtain Pseudo GT Views
Conditional Diffusion Model SD
Photorealistic Texture Optimization
Randomly-Sampled Views
Canonical Views
$\hat{\mathbf{I}}$
$\mathcal{L}_{\text{MSE}} = ||\mathbf{I}_{\text{PseudoGT}} - \hat{\mathbf{I}}||_2^2$
$\hat{\mathbf{I}}$
$\nabla\mathcal{L}_{\text{SDS}}(\phi_{\text{SD}}, \hat{\mathbf{I}}) = \mathbb{E}_{t,\epsilon}\left[w(t)(\epsilon_{\phi_{\text{SD}}}(\tilde{\mathbf{I}}_t; y, t) - \epsilon)\frac{\partial\hat{\mathbf{I}}}{\partial\theta}\right]$

Figure 4.7: TextMesh method overview.

4. (bottom left) eventually, they finetune the mesh texture on the obtained views together with a small SDS gradient to account for minor misalignments

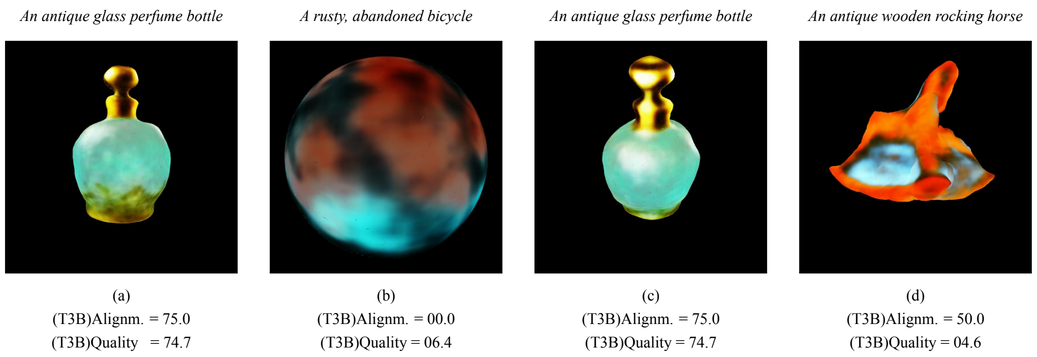


Table 4.10: TextMesh(IF): (a) prompt with the highest T3Bench Alignment score, (b) prompt with the lowest T3Bench Alignment score, (c) prompt with the highest T3Bench Quality score, (d) prompt with the lowest T3Bench Quality score.

Prolific-Dreamer

Authors of *ProlificDreamer* [43] highlighted how Score Distillation Sampling (SDS) (section 2.3) proposed by Poole, Jain, Barron, and Mildenhall [29] has shown great promise in text-to-3D generation by distilling pretrained large-scale text-to-image diffusion models, but suffers from over-saturation, over-smoothing, and low-diversity problems. In this work, they propose to model the 3D parameter as a random variable instead of a constant as in SDS and present variational score distillation (VSD), a principled particle-based variational framework to explain and address the aforementioned issues in text-to-3D.

They further present various improvements in the design space for text-to-3D such as distillation time schedule and density initialization, which are orthogonal to the distillation algorithm yet not well explored. In their work, they presented a systematic study of all these elements to obtain elaborate 3D representations.

Method

They first present Variational Score Distillation (VSD), which treats the corresponding 3D scene given a textual prompt as a random variable instead of a single point as in SDS [29].

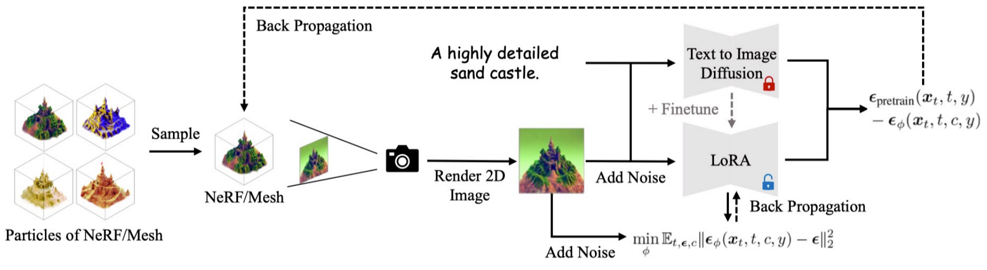

Back Propagation
A highly detailed sand castle.
Text to Image Diffusion
+ Finetune
LoRA
$\epsilon_{\text{pretrain}}(\boldsymbol{x}_t, t, y) - \epsilon_\phi(\boldsymbol{x}_t, t, c, y)$
Sample
Particles of NeRF/Mesh
NeRF/Mesh
Render 2D Image
Add Noise
Back Propagation
Add Noise
$\min_\phi \mathbb{E}_{t,\epsilon,c}\|\epsilon_\phi(\boldsymbol{x}_t, t, c, y) - \epsilon\|_2^2$

Figure 4.8: ProlificDreamer VSD method overview.

VSD optimizes a distribution of 3D scenes such that the distribution induced on images rendered from all views aligns as closely as possible, in terms of KL divergence, with the one defined by the pretrained 2D diffusion model. Under this variational formulation, VSD naturally characterizes the phenomenon that multiple 3D scenes can potentially align with one prompt. To solve it efficiently, VSD adopts particle-based variational inference, and maintains a set of 3D parameters as particles to represent the 3D distribution. They derive a novel gradient-based update rule for the particles via the Wasserstein gradient flow and guarantee that the particles will be samples from the desired distribution when the optimization converges[43].

Their update requires estimating the score function of the distribution on diffused rendered images, which can be efficiently and effectively implemented by a low-rank adaptation (LoRA) [10] of the pretrained diffusion model. The final algorithm alternatively updates the particles and score function.

An overview of the Variational Score Distillation (VSD) approach is illustrated in Figure 4.8.

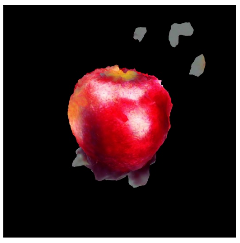
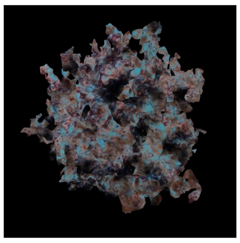
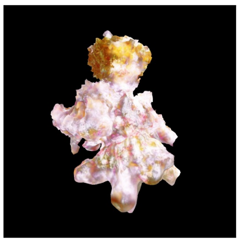
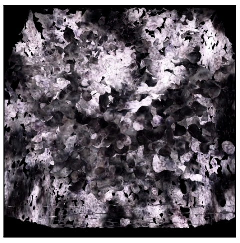
A shiny red apple	*An old bronze ship's wheel*	*A vintage porcelain doll with a frilly dress*	*A left-handed electric guitar painted black*
			
(a)	(b)	(c)	(d)
(T3B)Alignm. = 75.00	(T3B)Alignm. = 100.0	(T3B)Alignm. = 50.00	(T3B)Alignm. = 50.00
(T3B)Quality = 84.99	(T3B)Quality = 43.10	(T3B)Quality = 77.12	(T3B)Quality = 04.46

Table 4.11: ProlificDreamer: (a) prompt with the highest T3Bench Alignment score, (b) prompt with the lowest T3Bench Alignment score, (c) prompt with the highest T3Bench Quality score, (d) prompt with the lowest T3Bench Quality score.

HiFA

Authors of *HiFA*[48] highlighted how most existing methods use pre-trained text-to-image diffusion models to optimize 3D representations (e.g. NeRF[21]) often produce artifacts and inconsistencies across views due to their suboptimal optimization approaches and limited understanding of 3D geometry.

In their work, they propose holistic sampling and smoothing approaches to achieve high-quality text-to-3D generation, all in a single-stage optimization. They compute denoising scores in the text-to-image diffusion model's latent and image spaces. Instead of randomly sampling timesteps (also referred to as noise levels in denoising score matching), they introduce a novel timestep annealing approach that progressively reduces the sampled timestep throughout optimization. To generate high-quality renderings in a single-stage optimization, they propose regularization for the variance of z-coordinates along NeRF rays. To address texture flickering issues in NeRFs, they introduce a kernel smoothing technique that refines importance sampling weights coarse-to-fine, ensuring accurate and thorough sampling in high-density regions.

We summarize their technical contributions for two crucial components of text-to-3D generation: 3D representation and 3D optimization, which are outlined below[48]:

1. to achieve photo-realistic and highly-detailed text-to-3D generation, they propose score distillation in both the latent and image space of the pre-trained text-to-image diffusion models (moreover they introduce a timestep annealing approach for score distillation from text-to-image diffusion models);
2. to achieve sharp geometry quality through a single-stage training, they present a regularization method applied to the variance of z-coordinates along NeRF rays;

3. to address flickering issues in NeRFs, they propose a kernel smoothing technique that refines the PDF estimation in coarse-to-fine importance sampling;

Method

Authors of *HiFA* revisit the integration of the SDS approach with NeRFs, aiming to achieve photorealistic and high-quality text-to-3D generation through a single-stage optimization. In contrast to existing text-to-3D generation work, they distill the score in the text-to-image diffusion model's latent and image spaces for enhanced supervision. Moreover, they observe that the efficacy of the diffusion prior is limited in previous works when timesteps (also referred to as noise levels in denoising score matching) are randomly sampled during optimization.

Specifically, they observed that toward the end of the training process, the NeRF becomes almost determined in representing a particular 3D asset. Thus, they found that randomly sampling a large timestep drives the diffusion model to produce a denoised image that is distinct and unrelated to the original input rendering. This yields inconsistent distillation from the diffusion model and compromised optimization of NeRFs. To address this, they introduced a timestep annealing approach where the timestep in the forward diffusion process inversely correlates with the square root of the number of training iterations[48].

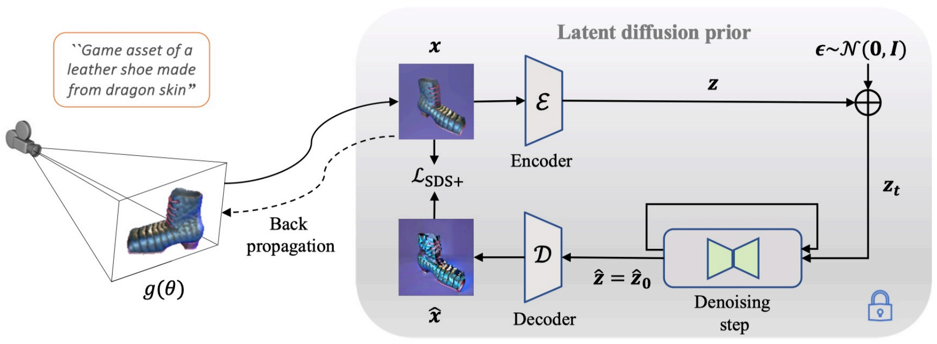


Figure 4.9: HiFA method overview.

One of the main goal of their work is to maintain the flexibility and photo-realism offered by the NeRF representation while at the same time achieving high-quality text-to-3D generation through a single-stage training.

For this reason, they proposed two techniques to advance NeRF optimization. Specifically, to address the cloudy geometry issue in NeRFs, they proposed a variance regularization that minimizes the variance of sampled z-coordinates distributed along NeRF rays. They observed that this technique enables NeRFs to more accurately represent crisp geometrical surfaces, thereby effectively mitigating the cloudiness issue[48].

Moreover, they proposed a kernel smoothing technique tailored for coarse-to-fine importance sampling along NeRF rays without an increase in the computational cost. This technique is inspired by the integrated positional encoding for spatial points within a cone, previously proposed to tackle aliasing issues in a single image view. In their case, the goal is to mitigate flickering issues across multiple views. Specifically, they use a kernel to refine the probability density function (PDF) estimated in the coarse sampling stage along a ray, which enables more comprehensive sampling near asset surface regions in the refined stage. This technique notably enhances the fidelity of importance sampling[48].

An overview of the HiFA method is illustrated in Figure 4.9.

Lucid-Dreamer

Authors of *Lucid-Dreamer*[15] highlighted how most existing methods use pre-trained text-to-image diffusion models to optimize 3D representations (e.g. NeRF[21]) often fall short in rendering detailed and high-quality 3D models.

This problem is especially prevalent as many methods base themselves on Score Distillation Sampling (SDS)[29] (section 2.3). Their work identifies a notable deficiency in SDS, that it brings inconsistent and low-quality updating direction for

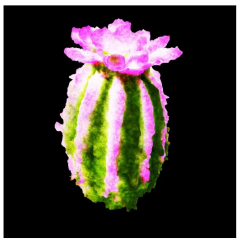
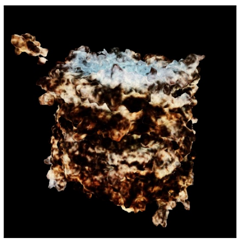
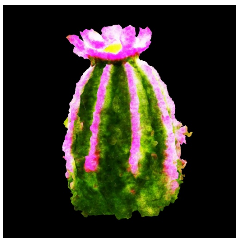
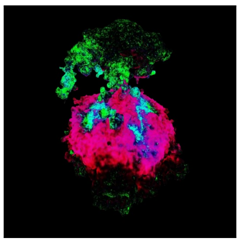
A cactus with pink flowers	*A dusty classic typewriter*	*A cactus with pink flowers*	*A shiny emerald green beetle*
			
(a)	(b)	(c)	(d)
(T3B)Alignm. = 75.00	(T3B)Alignm. = 25.00	(T3B)Alignm. = 75.00	(T3B)Alignm. = 75.00
(T3B)Quality = 81.92	(T3B)Quality = 04.45	(T3B)Quality = 81.92	(T3B)Quality = 04.44

Table 4.12: HiFA: (a) prompt with the highest T3Bench Alignment score, (b) prompt with the lowest T3Bench Alignment score, (c) prompt with the highest T3Bench Quality score, (d) prompt with the lowest T3Bench Quality score.

the 3D model, causing the over-smoothing effect. To address this, they propose a novel approach called *Interval Score Matching (ISM)*. ISM employs deterministic diffusing trajectories and utilizes interval-based score matching to counteract over-smoothing. Furthermore, they incorporate *3D Gaussian Splatting*[12] (section 2.2) into their text-to-3D generation pipeline.

Overall, the authors of *Lucid-Dreamer* have identified the following key contributions[15]:

1. an in-depth analysis of Score Distillation Sampling (SDS), the fundamental component in text-to-3D generation, and identification of its key limitations for providing inconsistent and low-quality pseudo-GTs;
2. in response to SDS's limitations, they propose the Interval Score Matching (with invertible diffusion trajectories and interval-based matching, ISM significantly outperforms SDS with highly realistic and detailed results);
3. by integrating with 3D Gaussian Splatting[12] (section 2.2), their model achieves state-of-the-art performance, surpassing existing methods with less training costs;

Method

In their work, they reveal the pseudo-Ground-Truth (pseudo-GTs) generated by the diffusion model are usually inconsistent and have low visual quality. Consequently, all update directions provided by these pseudo-GTs are subsequently applied to the same 3D model. Moreover, due to the average effect, the final results tend to be over-smooth and lack of details[15].

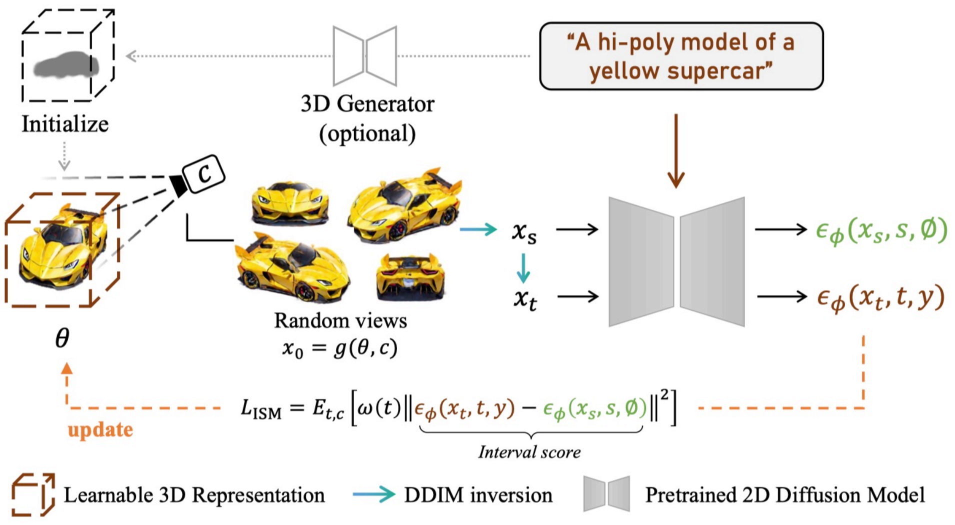

Initialize
3D Generator (optional)
"A hi-poly model of a yellow supercar"
c
θ
Random views
$x_0 = g(\theta, c)$
x_s
x_t
$\epsilon_\phi(x_s, s, \emptyset)$
$\epsilon_\phi(x_t, t, y)$
update
$L_{\text{ISM}} = E_{t,c}\left[\omega(t)\|\underbrace{\epsilon_\phi(x_t, t, y) - \epsilon_\phi(x_s, s, \emptyset)}_{\text{Interval score}}\|^2\right]$
Learnable 3D Representation
DDIM inversion
Pretrained 2D Diffusion Model

Figure 4.10: LucidDreamer method overview.

To address these issues, they propose a novel approach called Interval Score Matching (ISM). ISM improves SDS with two effective mechanisms. Firstly, by employing DDIM inversion, ISM produces an invertible diffusion trajectory and mitigates the averaging effect caused by pseudo-GT inconsistency. Secondly, rather than matching the pseudo-GTs with images rendered by the 3D model, ISM conducts matching between two interval steps in the diffusion trajectory, which avoids one-step reconstruction that yields high reconstruction error[15].

Finally, they show that their ISM is not only compatible with the original 3D model introduced in DreamFusion[29], but by utilizing a more advanced model – 3D Gaussian Splatting[12], their model achieves superior results compared to the state-of-the-art approaches[14] [15].

An overview of the Lucid-Dreamer method is illustrated in Figure 4.10.

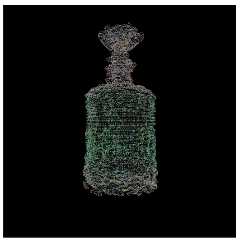
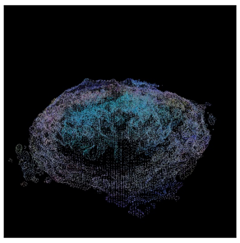
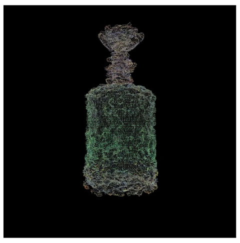
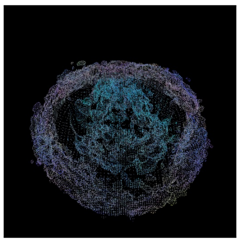
An antique glass perfume bottle	*A bright blue plastic swimming goggles*	*An antique glass perfume bottle*	*A bright blue plastic swimming goggles*
			
(a)	(b)	(c)	(d)
(T3B)Alignm. = 75.00	(T3B)Alignm. = 25.00	(T3B)Alignm. = 75.00	(T3B)Alignm. = 25.00
(T3B)Quality = 63.33	(T3B)Quality = 04.42	(T3B)Quality = 63.33	(T3B)Quality = 04.42

Table 4.13: LucidDreamer: (a) prompt with the highest T3Bench Alignment score, (b) prompt with the lowest T3Bench Alignment score, (c) prompt with the highest T3Bench Quality score, (d) prompt with the lowest T3Bench Quality score.

[14]other methods require multi-stage training, which is not needed in their model

Chapter 5

Experiments

5.1 Computational Resources Analysis

In section 1.1 we have introduced the concept of *efficient* 3D generation and we have defined the challenges regarding how we can define a ”fair” balance between the *output quality* and the *efficiency* associated to each model.

Moreover, in section 1.3 we have defined which types of *efficiency components* we have constrained or bounded (mainly related to the computational resources needed by each model) in this work in order to allow a fair comparison also in other common scenarios.

Once we have considered all the previous aspects, we can focus on which definition of an *Efficient 3D Model* we have used as baseline for all the experiments and the results presented in this work. As already mentioned in section 1.3, we have mainly characterized the *efficiency* of a 3D model in terms of *generation time* and *hardware requirements* needed. The list containing all the details regarding the computational resources originally used by the authors of all the models implemented in this work is presented in Table 1.2.

Efficient 3D Model is a model which is able to **generate** a 3D object in **less than 10 minutes**, using a **single GPU** and a **maximum of 24GB of VRAM** (we have used a single *NVIDIA RTX 3090* GPU in all of our experiments).

Considering that most of the selected models require from 30 minutes to 2 hours to generate a single 3D shape, we have decided **at priori** (i.e. not experimentally) to limit the maximum generation time of each model to 10 minutes in order to adhere to possible real-world scenarios where the generation time is a critical factor. We have analyzed the impact of this decision on the output quality of each model in chapter 6.

Limiting the Total Generation Time

Most of the models we have used in our experiments do not have a built-in mechanism to limit the total generation time. Generally, the concept of ”time” is not a parameter that can be directly expressed in the architectures of Deep Learning models. This concept is a consequence of several factors, such as the model's architecture, the hardware used, the size of the input, the hyperparameters specified, the training pipeline, the presence of multi-modal components, the stopping criterias defined, the complexity of the task, and so on.

The above statements is valid also for all the models we have used in our experiments. As we can see from Table 1.2, there are some models that already satisfy the constraints regarding the maximum generation time that we have imposed in section 5.1 while several other models do not.

In particular, for the models presented in section 4.1 which by construction prefers very low *generation time* over *quality*, we had not imposed any additional constraints. For all the other models presented in section 4.2 which conversely prefers *quality* over *generation time*, we had limited the maximum generation time to 10 minutes.

Hence, the final question is: *”how did we limit the total generation time of a model that does not have a built-in mechanism to do so?”*.

In order to effectively limit the total generation time of each model, we have perfomed a comprehensive analysis of the *generation statistics* of each model. This analysis has been conducted by monitoring the unbounded (i.e. without our efficiency constraints) generation of each model on a single prompt and by analyzing the average *number of iterations per second* needed by each model at each different stage of its generation pipeline[1].

Since most of our models exploit pre-trained text-to-image mechanism (e.g. through pre-trained Stable Diffusion Model [34]), we have focused on an alyzing the average *number of iterations per second* regarding the final architecture optimization steps commonly used by all the models (e.g. NeRF [21] MLP training, Gaussian Splatting [12] parameters optimization, ...).

Through this extraction, we have been able to define the *maximum number of iterations* that each model can perform in any custom given amount of time. Indeed, this value has been used as a *stopping criteria* for each model in order to limit the total generation time following respect to our constraints (i.e. 10 minutes). The results of this comprehensive analysis are presented in Table 5.1.

Efficiency vs Evaluation

In this section we want to point out some differences between our work and the work of T3bench [5] in order to properly contextualize the results presented in chapter 6

[1]As explained in chapter 4 some models rely on a multi-stage generation pipeline through which they generate intermediate representations (of the final 3D shape) in the earliest stages and then they refine them in the later stages.

Model	Stage	iters/sec	Tot. Iters	Tot. Time
LucidDreamer	~	1.40 it/s	850 iters	< 10 min
HiFA	~	4.40 it/s	2650 iters	< 10 min
ProlificDreamer	coarse	2.70 it/s	650 iters	< 4 min
	geometry	4.60 it/s	800 iters	< 3 min
	texture	2.75 it/s	500 iters	< 3 min
TextMesh(IF)	~	5.75 it/s	3500 iters	< 10 min
Fantasia3D	coarse	5.00 it/s	2400 iters	< 8 min
	texture	4.30 it/s	500 iters	< 2 min
SJC	~	16.45 it/s	10000 iters	< 10 min
Magic3D(IF)	coarse	5.85 it/s	1700 iters	< 5 min
	refine	4.30 it/s	1300 iters	< 5 min
LatenNeRF	coarse	6.45 it/s	1900 iters	< 5 min
	refine	4.58 it/s	1300 iters	< 5 min
DreamFusion(IF)	~	5.8 it/s	3500 iters	< 10 min

Table 5.1: Models: detailed report of all the statistics related to each model (splitted by stage) in terms of generative iterations and related generation time.

and compare the different approaches used in both works.

As already explained in chapter 1 and in section 3.2 the main goal of T3bench [5] is to provide a benchmark for evaluating the *comprehensive quality* of text-driven 3D generation shapes. Conversely, the main goal of our work is to exploit evaluation metrics such as the ones defined in the T3Bench research in order to properly evaluate the 3D generation results obtained by putting strong constraints regarding the *efficiency* of the models used.

This difference is remarkably evident if we consider, for instance, the constraint related to the *maximum total number of iterations* associated to each model. Since T3Bench had not imposed any constraints regarding the *efficiency* of the models used, **the authors of T3Bench have not limited the total number of iterations** associated to each model. Conversely, we have done the opposite in order to adhere to our constraints regarding the *efficiency* of the models used.

Differences between these two approaches have been reported in Table 5.2 where we have compared the *maximum total number of iterations* associated to each model used in our work with the *maximum total number of iterations* associated to each model used in the T3Bench work[2]. By looking at this results we can see that the average[3] *maximum total number of iterations* associated to the models used in our work is significantly lower than the average[3] *maximum total number of iterations* associated to the models used in the T3Bench work. This aspect have highly influenced our results as show in chapter 6.

Model	**T3B (iters)**	**Ours (iters)**	**Ours/T3B**
ProlificDreamer	70k it	2k it	3% it
Fantasia3D	15k it	3k it	20% it
SJC	10k it	10k it	100% it
Magic3D(IF)	15k it	2.5k it	17% it
LatentNeRF	20k it	3k it	15% it
DreamFusion(IF)	10k it	3.5k it	35% it

Table 5.2: T3Bench comparison: detailed report of all the statistics related to each model (splitted by stage) in terms of generative iterations and related generation time.

5.2 Prompts Dataset

In chapter 6 we have presented the results of our experiments by using a dataset of 100 prompts. This dataset has been extracted from the T3Bench prompts dataset [5] which originally contains three sets of 100 prompts each named *single*, *multi* and *surrounding*. As stated by T3Bench authors:

[2]We have only reported the comparison for the model implemented by TBench authors.
[3]computed as the mean of the total number of iterations performed on each prompt

The "single" object set represents the simplest scenario to establish a baseline level of performance, and the other two prompt sets (i.e. multi, surrounding) introduce increased levels of difficulty by incorporating additional information like surroundings or multiple objects. [5]

To generate these prompt sets, T3Bench authros used GPT-4 [4] to generate a large pool of candidate prompts, and then they have manually filtered out prompts that contain proper nouns or toponyms. Subsequently, they utilized ROUGE-L [18] to quantify prompt similarity and incrementally remove highly similar prompts until there remains a number of 100 distinct prompts with significant diversity in each prompt set [5].

Unfortunately, this prompt selection strategy has led to the creation of a dataset of prompts that is not well balanced in terms of features diversity and object's similarity. Indeed, some prompts are very similar to each other (e.g. different prompts which specify different colors or features for the same object) while others are very different (e.g. some prompts characterize objects not present in any other entry). In the chapter 6 we have shown how this aspect has influenced the results of our experiments.

Takining into account our constraints regarding the *efficiency* of the models used (which, as a consequence, limit the generation capability of our models), we have decided to use **only** the *single* group of 100 prompts in our experiments. This decision has been taken in order to adhere to the simplest scenario defined by T3Bench authors and to allow us to compare the effects of the efficiency constraints on the output quality of each model respect to the results obtained by them.

5.3 3D Priors Initialization

One of the main goal of our research is to analyze the impact of applying our custom-defined *efficiency* constraints (section 1.3) to existing methods for generating 3D shapes given natural language descriptions of the desired object.

In particular, we know that by limiting the time of generation of the 3D shapes, we are implicitly limiting the generative capabilities of these models. Hence, since we do not want to change neither the architecture of models nor our *efficiency* constraints, we have decided to explore the possibility of boosting the generative capabilities of the models by exploiting pre-defined priors.

Most of the models which exploits an implicit parametrization of the 3D shapes, start their training process by initializing a coarse 3D shape independently from the input prompt.

Many researches presented in chapter 4 pointed out several issues by highlighting that the most of the current text-to-3D models lack of what they are usually called *"3D priors"* or *"3D initialization bias"* which leads to an higher probability of generating 3D shapes that are not coherent with the input prompt.

For this reason, we have decided to explore the impact of using pre-defined 3D priors in order to boost the models' generative capabilities.

We can summarize the priors used in our experiments as follows[4]:

1. **SDF Bias**: we can specify the properties of an elementary 3D shape (e.g. a sphere, an ellipsoid, ...) that should be used as a prior for the 3D shape generation process;

[4]notice that our choice of priors is based on the availability of the models' implementation

2. **Initial Prompt**: we can specify a simpler[5] textual prompt and exploit an external model to generate the initial 3D shape representation[6];

Only three among all the models that we have analyzed in chapter 4 and implemented support the use of priors: *LucidDreamer* employs the *Initial Prompt*, while both *Fantasia3D* and *TextMesh(SD)* support the *SDF Bias* prior initialization.

5.4 Diffusion Model Backbones

As already done in the other sections of this chapter, we have reported in Table 5.3 the different statistics regarding the *number of iterations per second* used by each model divided by both the different stages of its generation pipeline and the stable difussion backbone used.

Model	**Vers.**	**Stage**	**iters/sec**	**Tot.Iters**	**Tot.Time**
TextMesh	IF	∼	5.75 it/s	3500 iters	< 10 min
	SD	∼	4.20 it/s	2500 iters	< 10 min
Magic3D	IF	coarse	5.85 it/s	1700 iters	< 5 min
		refine	4.30 it/s	1300 iters	< 5 min
	SD	coarse	4.30 it/s	1300 iters	< 5 min
		refine	4.40 it/s	1300 iters	< 5 min
DreamFusion	IF	∼	5.8 it/s	3500 iters	< 10 min
	SD	∼	4.45 it/s	2650 iters	< 10 min

Table 5.3: Diffusion Backbones: detailed report of all the statistics related to each model (splitted by stage) in terms of generative iterations and related generation time.

We have mainly analyzed the impact of two different diffusion backbones by taking into account the *efficiency* constraint that we have set on the hardware used to

[5]compared to the one used for generating the final 3D shape
[6]before starting the training process

generate the 3D shapes. The two diffusion backbones that we have considered are *StableDiffusion 1.5 (SD)* and *DeepFloyd (IF)*. Table 5.4 provide the references to the source code of these diffusion backbones.

5.5 Implementation Details

We have already introduced in section 1.3 that one of the main discriminant factors in the selection of the models to implement was the availability of open-source code. This requirements allow us to present our results under the assumption that experiments that we have conducted are reproducible and that the results are comparable with the results of the original authors. Moreover, we have also mentioned in section 1.3 that our approach, regarding the implementation phase, has been based on several steps which we are going to describe in the following paragraphs.

First of all, we have started by collecting the source code of each model mainly available on the Github platform. Then, we have proceeded to adapt the source code to our needs, by fixing any possible bug and by creating a standard interface for the generation of 3D shapes. This ”interface” allow anyone to easily generate 3D shapes on arbitrary dataset by simply defining as input the *model* and the *text prompts* themself.

Secondly, we have develop a new interface to export the 3D shapes generated by each model as 3D meshes. It is important to highlight that this is interface has been tailored to each specific 3D output format generated by each model. For example, some models generate 3D shapes in the form of pointclouds while some other don’t generate 3D shapes at all (e.g. they generate *latents* vectors related to an INR that need to be decoded in order to produce a valid 3D shape). Unfortunately, conversely to what we have done for the generation process, we were not able to standardize this interface considering the huge number of different requirements associated to each model output.

Thirdly, in order to maintain consistency respect to the results obtained by authors of T3Bench [5], we copied the rendering pipeline defined in this research and we used it to extract the renderings (i.e. multiple 2D images of a given 3D shape taken from different point of views) from the 3D meshes generated by each model.

Finally, we have integrated both CLIP and T3Bench evaluation pipelines starting from the rendering obtained in the previous step. The complete list containing the source-code reference of each component of our implementation process is presented in Table 5.4.

Component	**Type**	**Library**
Cap-ShapE Cap-PointE	Model	Cap3D[33]
ShapE PointE	Model	Shap-E[27] Point-E[26]
LucidDreamer	Model	LucidDreamer[32]
HiFA ProlificDreamer TextMesh Fantasia3D SJC Magic3D LatentNeRF DreamFusion	Model	Threestudio[30]
T3B Aligment T3B Quality	Metric	T3Bench[39]
StableDifussion 1.5 DeepFloyd IF	Diffusion Backbone	StabilityAI DeepFloyd[38]

Table 5.4: Developed components source code references.

Chapter 6

Results

In this chapter we present the results of our experiments. We start by comparing our results with the ones obtained by T3Bench in section 6.1. Then, in section 6.2, we present a comprehensive overview of the evaluation performed on all the models that we have implemented in this work. Finally, in section 6.3, we present additional results related to the investigation of the impact of the learning rate, the priors, and the diffusion backbones on the quality of the generated images.

6.1 T3Bench Comparison

The first aspect that we want to analyze is the comparison between the scores obtained by our *efficient* models[1] and the scores originally obtained by the T3Bench authors without imposing any boundary to the generation time or *capacity* of the models. This comparison is shown in Table 6.1.

The first thing that we noticed is the evidence of two different trends regarding the T3Bench *Alignment* and *Quality* scores.

[1]models for which we have defined a set of contraints regarding their generation *capacity* (i.e. time, hardware).

Model	T3Bench Results			Our Results		
	Qual.	Align.	Avg.	Qual.	Align.	Avg.
ProlificDreamer	**51.1**	**47.8**	49.4	26.7	47.3	37.0
Fantasia3D	**29.2**	23.5	26.4	16.1	**43.0**	29.6
SJC	**26.3**	23.0	24.7	23.6	**44.0**	33.8
Magic3D (SD)	**38.7**	35.3	37.0	27.4	**49.5**	38.5
LatentNeRF	**34.2**	32.0	33.1	18.1	**40.3**	29.2
DreamFusion (SD)	**24.9**	24.0	24.4	20.2	**47.5**	33.9

Table 6.1: Comparison between the scores obtained by our constrained models and the scores originally obtained by the T3Bench authors without imposing any boundary to the generation *capacity* (i.e. time, hardware) of the models.

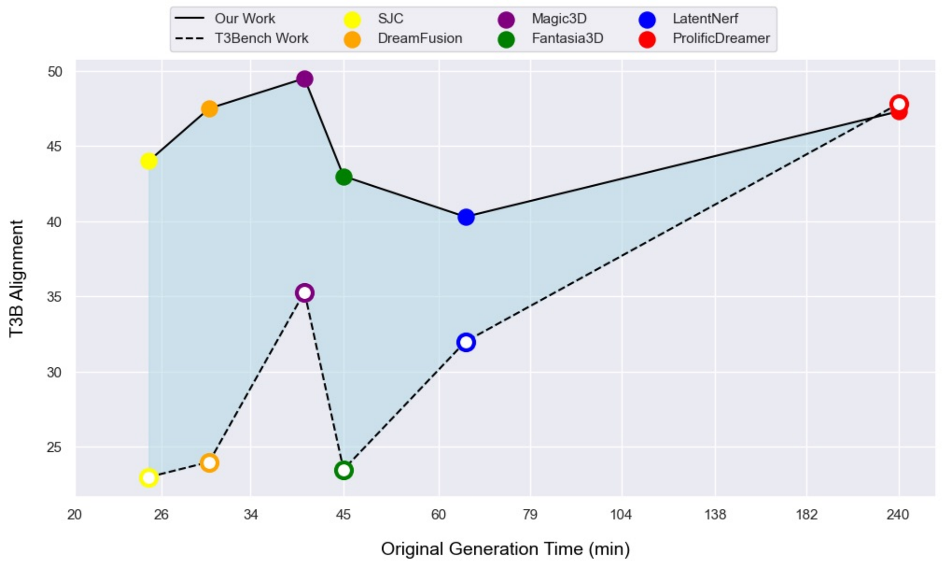
In particular, we can observe that, most of the time, the *Alignment* scores are generally higher for our *constrained* models respect to the *unconstrained* models used in the T3Bench research. Conversely, the *Quality* scores present an opposite trend, with the original *unconstrained* models generally outperforming our *constrained* models.

Figure 6.1 summarize these two different scores trends by showing the *Alignment* and *Quality* scores of our models as a function of the time required to generate them. The same scores for the T3Bench models are also shown for comparison.

These two different trends highlight the need of a deeper analysis for both the *Alignment* and *Quality* scores. In particular, we want to understand if these trends have been affected by our *efficiency constraints* or if they are just a consequence of different[2] choices that we have made during the development of our generation/exporting pipeline.

[2]respect to the T3Bench original research

(a) T3Bench Alignment



(b) T3Bench Quality

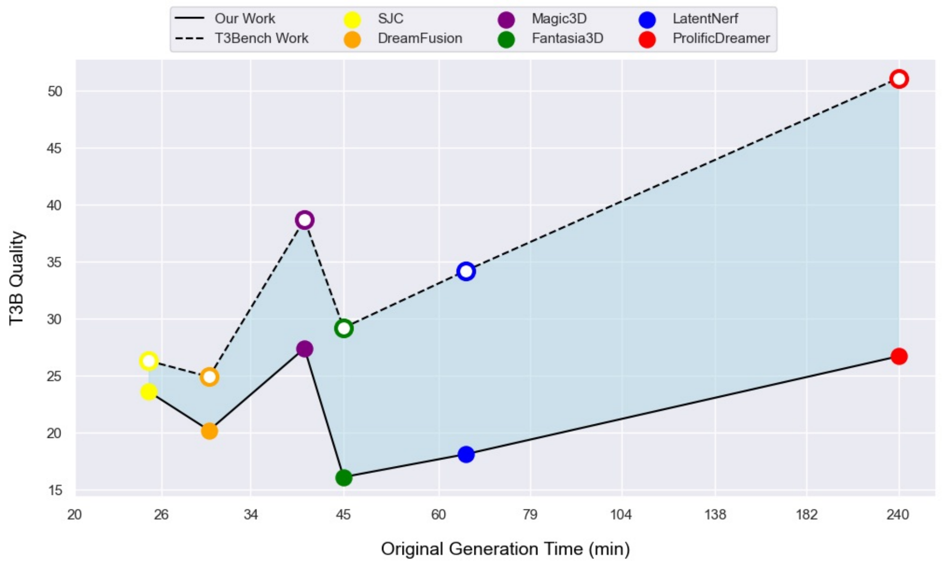


Figure 6.1: Comparison of the T3Bench scores trends obtained in the original T3Bench research and the ones obtained in our research. The scores are sorted by the original unbonded time required by each model to generate a 3D shape.

Alignment Scores At th beginning, we were not sure about the impact of our *efficiency constraints* on the *Alignment* scores. In fact, since the *Alignment* score is a measure of how much the generated shape and textual prompt are aligned, this is the most important open question that we wanted to address.

Table 6.1 and Figure 6.1 show that, in general, the *Alignment* scores of our models are higher than the ones obtained by the T3Bench authors. This is a very interesting result because it means that, despite the fact that we have imposed a set of constraints to the generation of our models, we have not affected the *Alignment* score in a significant way.

At the same time this result is in contrast for the results that we have obtained for the *Quality* score which measure the ”intrinsic” quality of the 3D shape itself. This leads to the following question: *how is it possible that the* Alignment *of our constrained models are higher than the ones obtained by the T3Bench authors, while knowing that the* Quality *scores are generally lower (meaning that we are generating more coarse and less detailed 3D shapes)?*.

In order to answer to this question we have to briefly analyze the *Alignment* score definition. As presented in the T3Bench research [5], the *Alignment* score is a measure of how much the generated shape and textual prompt are aligned. This means that, if we are able to generate 3D shapes which are more coherent to the input prompt, we will obtain higher *Alignment* scores. T3Bench authors have employed *BLIP* [14] in order to **describe** renderings of 3D shapes and then use a Large Language Model to both **merge** and **score** the generated descriptions. This process has three main critical points:

1. *image-to-text*: obtaining a description of coarse and less detailed 3D shapes may result in more generic and less detailed captions;
2. *merging*: asking a LLM to merge more generic and less detailed captions may result in a more generic and less detailed description of the 3D shape;
3. *scoring*: scoring through a LLM the coherence between the text prompt and

a more generic and less detailed description (of the 3D shape) may result in a higher *Alignment* score (since the likelihood of captioning details not related to the text prompt is lower by definition).

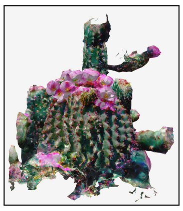
With this process, we can effectively evaluate only *unconstrained* models which are able to generate 3D shapes with fine-grained details related to the input prompt. Moving away from this assumption (as it is in our case), we may obtain a higher *Alignment* scores for 3D shapes which are more generic and less detailed since the merged captions will be more generic and, as a consequence, less prone to contain descriptions of details which are not present in the input prompt.

Moreover, another critical point is on the choice of the LLM itself. Despite we, as T3Bench authors, have employed the same ”type” of LLM (i.e. ChatGPT), due to technical limitations **we have used a different version of the model (ChatGPT 3.5-turbo)** respect to the one used by the T3Bench authors (ChatGPT 4[4]).

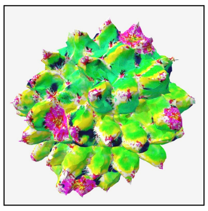
Fursther research is needed in order to understand the impact of chooosing a different version of the LLM used by T3Bench authors to merge the renderings captions (from BLIP) and score the merged captions in order to obtain the *Alignment* scores.

Renderings in Table 6.2 reflects this discrepancy by showing how much the *Alignment* scores of our *constrained* models are aligned with the ones obtained by the *unconstrained* models used in the T3Bench research (despite an huge gap in both the *Quality* and generation time boundaries).

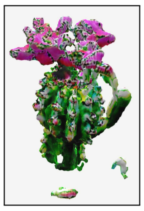
Quality Scores At the beginning, we expected that the *Quality* scores would have been affected by our *efficiency constraints* in very evident way. Since the *Quality* score is a measure of the ”intrinsic” quality of the 3D shape (which does not take into account the input prompt), we expected that by constraining the generation time of our models, we would have obtained, in general, 3D shapes which are more coarse


(T3B)ProlificDreamer

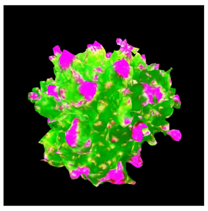
(T3B)Align. = 100.0
(T3B)Qual. = 83.8


(T3B) Fantasia3D

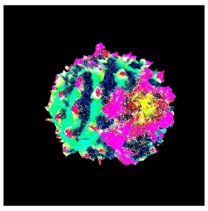
(T3B)Align. = 75.00
(T3B)Qual. = 63.8


(T3B) SJC

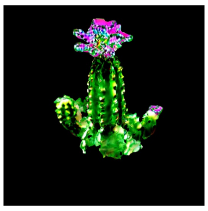
(T3B)Align. = 100.0
(T3B)Qual. = 62.0


(OUR) ProlificDreamer

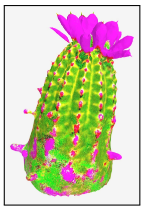
(OUR)Align. = 75.00
(OUR)Qual. = 27.29


(OUR) Fantasia3D

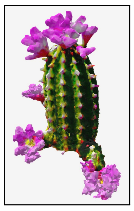
(OUR)Align. = 75.00
(OUR)Qual. = 10.21


(OUR) SJC

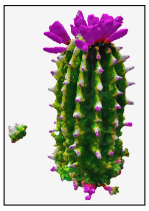
(OUR)Align. = 75.00
(OUR)Qual. = 72.69


(T3B) Magic3D

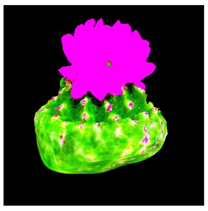
(T3B)Align. = 100.0
(T3B)Qual. = 79.2


(T3B) LatentNeRF

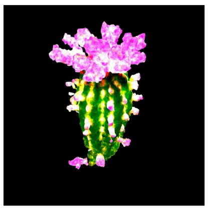
(T3B)Align. = 100.0
(T3B)Qual. = 86.3


(T3B) DreamFusion

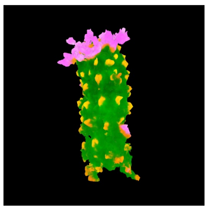
(T3B)Align. = 100.0
(T3B)Qual. = 82.4


(OUR) Magic3D

(OUR)Align. = 75.00
(OUR)Qual. = 61.85


(OUR) LatentNeRF

(OUR)Align. = 75.00
(OUR)Qual. = 84.99


(OUR) DreamFusion

(OUR)Align. = 75.00
(OUR)Qual. = 77.17

Table 6.2: Comparison between the original scores obtained by T3Bench and the ones obtained by our constrained generation respect to the prompt *"A cactus with pink flowers"*. The table shows the results for the 3D models ProlificDreamer, Fantasia3D, SJC, Magic3D, LatentNeRF, and DreamFusion (all the ones analyzed by T3Bench authors).

and less detailed[3] than the ones obtained by the T3Bench authors.

Table 6.1 confirm this expectation, and, at the same time Figure 6.1 shows how the *Quality* trend between our *constrained* models and the *unconstrained* models used in the T3Bench research only differs by an exponential scale factor. This means that, while the *Quality* scores of our models are generally lower than the ones obtained by the T3Bench authors, increasing the generation time of our models would result in increasing ”proportionally” the *Quality* scores.

In the other hand, looking at the comparison between renderings in the Table 6.2 obtained by our *constrained* models and the ones obtained by the *unconstrained* models used in the T3Bench research, we can see that *Quality* scores do not always properly reflect the huge quality diffrence that we can see in the renderings.

For instance, by looking at the renderings (Table 6.2) of *DreamFusion*[29] we can se that the difference between the *Quality* score obtained by our *constrained* model (i.e. 77.17) and the score obtained by *unconstrained* model of T3Bench (i.e. 82.4) only differs by a 10% factor[4] which does not properly represent the huge quality diffrence that we can see in the renderings.

As we can see in Table 6.1, while looking at the average T3Bench score, we can see that these two very different trends neglect each others, resulting in a more balanced *average score* for all models. At the same time, we can also highlight that, conversely to separately considering *Alignment* and *Quality* scores, our *average scores* are closer to *average scores* obtained by T3bench authors with unconstrained models[5].

[3]which may also still contain some artifacts ...
[4]over the full range of the *Quality* score ($[0, 100]$), this is a very small difference.
[5]ProlificDreamer [43] is the only exception

Average Scores Despite the fact that, as presented in the T3Bench [5] research, the *average score* should be a better indicator of the overall performance of the models, we have shown that the *average score* may be misleading in situations where the *Alignment* and *Quality* scores are affected by different trends and numerical magnitudes. This is the case for our models and this is the reason why we have decided to analyze the *Alignment* and *Quality* scores separately.

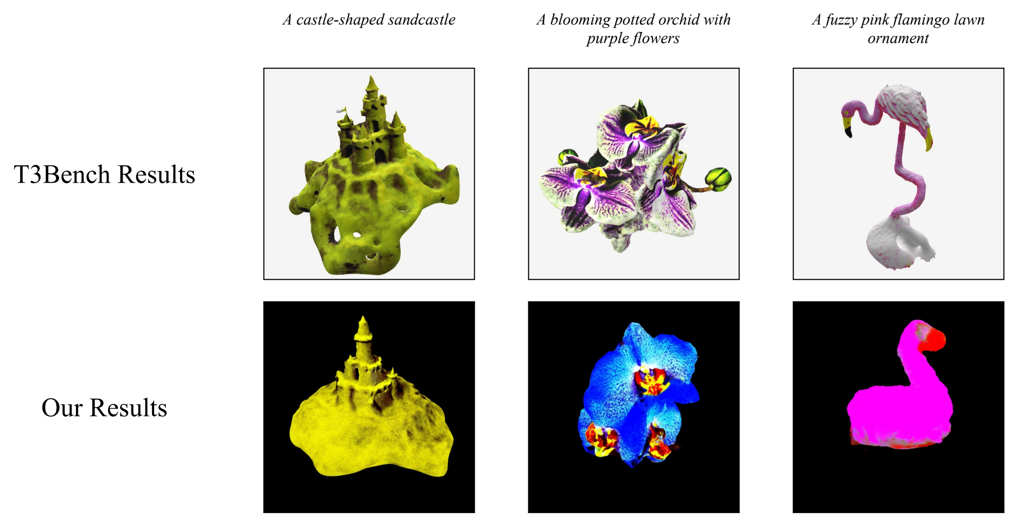
This intrinsic limitation of the *average score* (i.e. given by its formulation as the arithmetic mean of the *alignment* and *quality* scores) also may lead to unproper comparisons when we want to examine models taken from different researches. For instance, if we consider the goals of our research, we can see that the *average score* is not able to properly highlight the differences between our *constrained* models and the *unconstrained* models used in the T3Bench research.

T3Bench vs Human Preference Conversely to what expressed by T3ench [5] authors in their research, we have shown that these scores (especially the *Alignment* score) are not always a good indicator of generic human preference in terms of 3D shape evaluation.

For instance, if considering the results presented in Table 6.3, we can see the comparison between the renderings obtained by our *constrained Magic3D*[17] model and the renderings obtained by the *unconstrained Magic3D*[17] model used in the T3Bench research. In particular, this model (*Magic3D*[17]) has been chosen because it has:

1. the **best** *Alignment* score among our models;
2. the **closest** *average score* to its respective T3Bench *unconstrained* model;

and, as a consequence, **it should be the one that is more ”similar” to its respective *unconstrained* model used in the T3Bench research**.


A castle-shaped sandcastle
A blooming potted orchid with purple flowers
A fuzzy pink flamingo lawn ornament
T3Bench Results
Our Results

(a) good (b) medium (c) bad

Table 6.3: Magic3D: comparison of the results obtained by T3Bench and by our method for the three prompts sorted by decreasing scores of human preference.

By ”similar” we mean that, if the T3Bench scores are a good indicator of the human preference, we should expect that the renderings obtained by our *constrained* model are very close/similar to the ones obtained by the *unconstrained* model used in the T3Bench research. Again, as we can see in Table 6.3, this is not the case and we can also have an hint about how a generic human evaluator will score our renderings.

In the end, this confirms our suspicion about the fact that, at least for our scenario, the T3Bench scores are not always a good indicator of the human preference in terms of 3D shape evaluation.

Conclusion In conclusion, we need to remark that comparing the scores obtained by our *constrained* models with the scores obtained by the T3Bench authors for the same models is not completely ”fair”.

Indeed, the main goal of this section was to analyze the impact of our *efficiency constraints* on the scores of our models[6], extract the main trends related to how the

[6]respect to the scores obtained by the T3Bench authors which have not imposed any boundary to the generation time or *capacity* of the models.

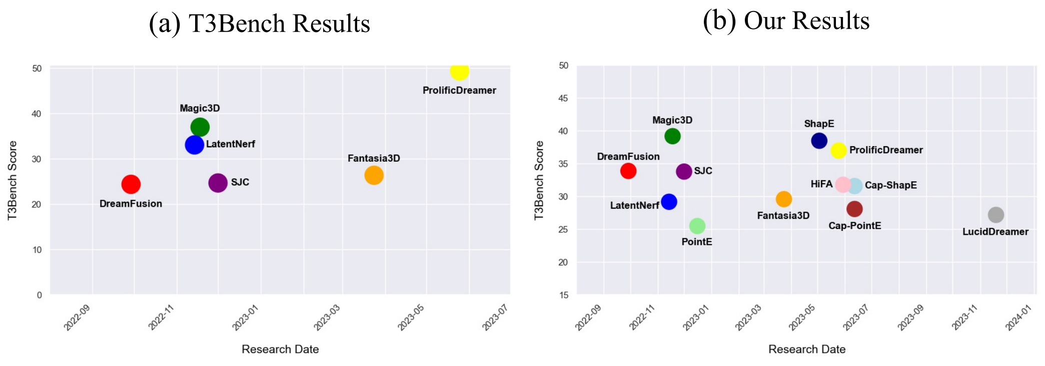


Figure 6.2: Overview of the T3Bench average scores obtained by each model, sorted by research date.

Alignment and *Quality* vary as long as we constraint models which requires a generation of time progressively increasing and to understand if the T3Bench scores are a good indicator of the human preference in terms of 3D shape evaluation in our scenario.

Figure 6.2 depicts the general overview of the original average scores obtained by the T3Bench authors and the average scores obtained by our *constrained* models.

6.2 Aggregated Results

In this section we would like to discuss the results of our experiments by comparing the different scores obtain by each one of the models that we have implemented. In particular, we want also to inspect the relationships between the *CLIP* and *T3Bench* scores in order to better highligh possibile patterns and trends between these ones.

We have summarized the results of our experiments in Table 6.4. In this table, we have reported the scores of the *CLIP* and *T3Bench* metrics for each one of the models as long as the *T3Bench average* metric.

Table 6.11, Table 6.12 and Table 6.13 also present some renderings examples foreach implemented model and three different prompts statements.

Our Results

Model	CLIP		T3Bench		
	Score	RPrec.	Qual.	Align.	Avg.
LucidDreamer	0.65	0.17	13.06	41.41	27.2
Cap-ShapE	0.71	0.40	18.31	44.95	31.6
Cap-PointE	0.69	0.30	16.03	40.25	28.1
HiFA	0.62	0.29	18.35	45.25	31.8
ProlificDreamer	0.69	0.46	26.68	**47.25**	37.0
ShapE	**0.72**	**0.49**	20.44	**47.47**	38.5
TextMesh (IF)	0.65	0.29	17.70	39.0	28.35
Fantasia3D	0.63	0.27	15.96	43.00	29.6
PointE	0.66	0.24	11.18	39.75	25.5
SJC	0.66	0.29	23.60	44.00	33.8
Magic3D (IF)	**0.73**	**0.49**	**33.69**	44.75	**39.2**
LatentNeRF	0.63	0.26	18.10	40.30	29.2
DreamFusion (IF)	0.65	0.30	19.55	42.50	31.0

Table 6.4: Scores: general overview of our constrained models' performances. The table shows the scores obtained by our models respect to all the integrated metrics.

Prompt Analysis The first thing that we can notice is that the quality of the results is highly dependent on the prompt statement. In particular, we can observe that:

1. some prompts are always pretty good, regardless of the model used.
2. some prompts are always very bad, regardless of the model used.
3. some prompts' quality varies remarkably across models.

These three results are very interesting and they are, in some sense, in contrast with the common belief that the "quality" of the results is mainly dependent on the model used.

At the beginning of our experiments, we were expecting that the quality of the results would have been mainly dependent on the model used. However, we discovered that some prompts are always very bad or pretty good, regardless of which model we exploit to generate their respective 3D shapes.

The above analysis remarks the importance regarding the definition of a proper dataset of prompts to be used for the evaluation of the models. Indeed, we have already highlight the issues related to our prompt dataset in section 5.2 but we were not expecting to discover such differences in the quality of the results among different prompts.

Model Analysis The 3D data parametrization performed by each model becomes a critical aspect while evaluating the results. Models which use more complex parametrization techniques, such as *LatentNeRF* and *Magic3D*, are expected to perform better on complex prompts (sometimes even in a constrained scenario, as the one as we are), while models which use simpler parametrization techniques, such as *Cap-PointE* and *PointE*, are expected to perform better on simpler prompts. This is a very interesting aspect that we would like to further investigate in the future[7].

However, have observed that some models are able to perform well on a given complex prompt and bad on other simple ones. This is the case of *Cap-PointE*[19] and *PointE*[25] which perform bad on the prompt *"A cactus with pink flowers"* and well on the prompt *"A bright red fire hydrant"*. Conversely, we have observed that some models are able to perform well on a given complex prompt and bad on other simpler ones. This is the case of *ProlificDreamer*[43] which performs well on the prompt *"A cactus with pink flowers"* and bad on the prompt *"A bright red fire hydrant"*.

[7]this also may pose an interesting challenge regarding the use of an hybrid approach based on the complexity of the prompt statement.

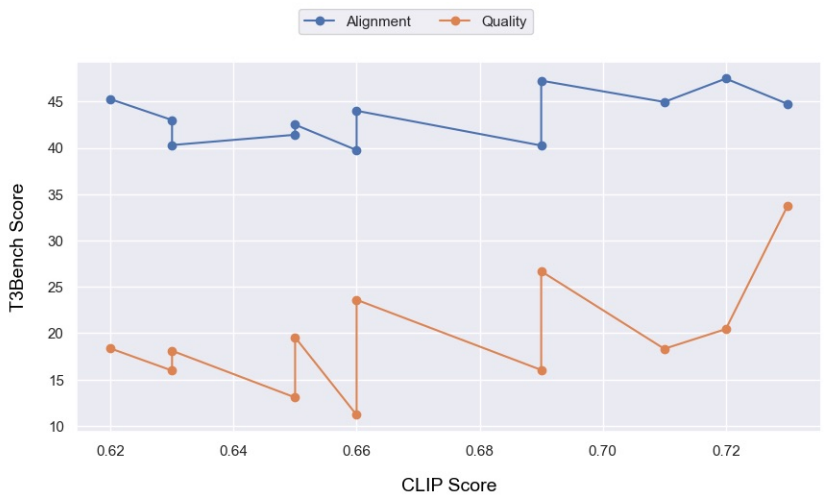


Figure 6.3: Comparison between the CLIP and T3Bench scores trends obtained foreach model.

CLIP vs T3Bench Figure 6.3 shows the relationship between the *CLIP-Score* and *T3Bench* scores for each one of the models. Since the obtained scores for both *CLIP-Score* and *T3Bench* metrics lay in a very small interval (our *CLIP-Score* values lay in the interval [0.62, 0.73] and our *T3Bench* scores lay in the interval [40.25, 47.47]), it is very difficult to find a clear relationship between these two metrics (despite in principle we would expect a positive correlation between them).

However, we can observe that the *CLIP-Score* and *T3Bench-Quality* scores are mostly positively correlated.

This is inline with our expectations considering that, being the *T3Bench-Quality* an indication of the intrinsic quality of a given 3D Shape, it is expected that, for 3D shapes with finer details and less artifacts, the renderings *CLIP*[31] embedding will be more likely to be ”aligned”[8] respect to the *CLIP*[31] text embeddings resulting in a higher *CLIP-Score* (section 3.1).

In the other hand, the lack of a clear relationship between the *CLIP-Score* and

[8]the cosine distance between the text and image embeddings of CLIP will be smaller

T3Bench-Alignment scores it is, unfortunately, very difficult to interpret. This may be mainly due to both the issues of evaluating the coherence between a textual prompts and a less detailed shapes[9] (as the *Quality* scores highlight) and differences in the implementation of *T3Bench-Alignment* itself respect to the original T3Bench formulation[39]. Both of these aspects have been already discussed in section 6.1.

6.3 Additional Results

Since one the most important aspects of our work the evaluation of current text-driven generative models at which we have applied several *efficiency* constraints (section 1.3) we have decided to explore the impact of different configurations which may help to improve the performance of our models.

Summarizing, we have explored the impact of different hyperparameters, priors and diffusion backbones on the performance of our models. The results of these experiments are presented in the following sections.

Hyperparameters

The first group of additional experiments is focused on the hyperparameters of the models.

Our Results

Model		CLIP		T3Bench		
name	lr	Score	RPrec.	Qual.	Align.	Avg.
Prolific Dreamer	x1	0.59	0.10	10.34	39.50	24.92
	x5	**0.69**	**0.46**	**26.68**	**47.25**	37.00

Table 6.5: Scores: overview of how different hyperparameters values affect the final scores obtained foreach metric.

[9]mainly obtained as effect of our *efficiency* constraints

As we have already introduced in section 5.3, we have decided mainly to focus only on the *ProlicficDreamer* model, as it is the one that requires the larger amount of time to generate a 3D shape given a prompt (Table 1.2).

Table 6.5 shows the results of the experiments we have conducted. We have tested two different learning rates scaler, *x1* and *x5*, and we have used the same priors as in the previous experiments.

As we can see from the results, the *ProlicficDreamer* model benefits from the use of priors and from the increase of the learning rate. The *CLIP-Score* increases from 0.59 to 0.69, the *CLIP-RPrecision* from 0.10 to 0.46, the *T3Bench-Quality* from 10.34 to 26.68, and the *T3Bench-Alignment* from 39.50 to 47.25.

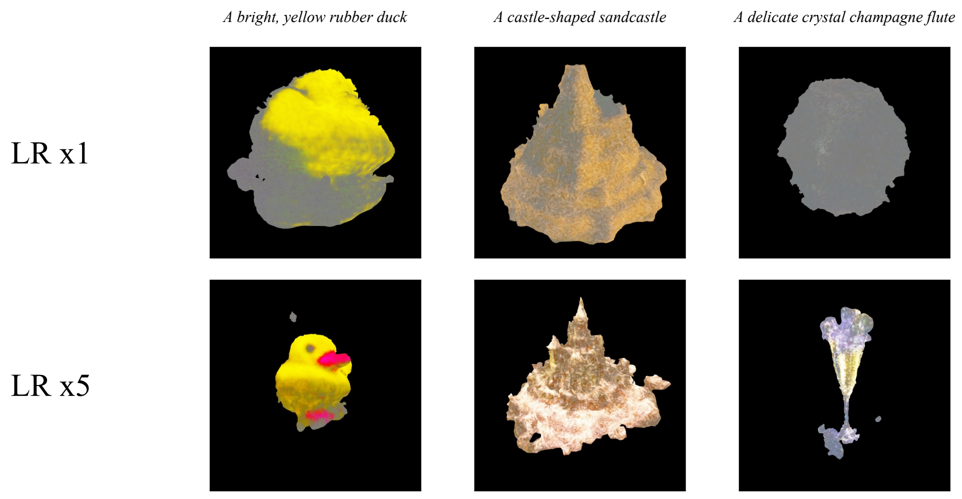

A bright, yellow rubber duck
A castle-shaped sandcastle
A delicate crystal champagne flute
LR x1
LR x5

Table 6.6: Comparison of the results obtained by the *ProlificDreamer* model using different learning rate settings.

The renderings examples presented in Table 6.6 confirm the scenario described above. Hence, we can conclude that the *ProlicficDreamer* model benefits from an increased learning settings.

Priors

The second group of additional experiments is focused on models which have the possibility to exploit priors to properly initialize their 3D shape parametric representation.

As we have already introduced in section 5.3, we have decided mainly to focus only 3 models which have the possibility to exploit priors, as they are also the ones that have the lower *CLIP-Score* (Table 6.4) and the lower *T3Bench-Quality* (Table 6.4) values.

Our Results

Model		CLIP		T3Bench		
name	prior	Score	RPrec.	Qual.	Align.	Avg.
LucidDreamer	yes	0.65	0.17	13.06	41.41	27.24
	no	0.56	0.07	7.96	38.38	23.17
Fantasia3D	yes	0.63	0.27	15.96	43.00	29.48
	no	0.64	0.28	16.52	42.75	29.63
TextMesh(SD)	yes	0.58	0.13	8.40	39.51	23,95
	no	0.53	0.05	9.06	32.00	20.53

Table 6.7: Scores: overview of how the usage of 3D intialization priors values affect the final scores obtained foreach metric.

Table 6.7 shows the results of the experiments we have conducted. We have tested *LucidDreamer*[15], *Fantasia3D*[2], and *TextMesh(SD)*[40] with and without priors initialization.

As we can see from the results, *LucidDreamer* is the only model for which we can see a significant improvement in the *CLIP-Score* and *T3Bench-Quality* when using priors. The *CLIP-Score* increases from 0.56 to 0.65, and the *T3Bench-Quality* from 7.96 to 13.06. In the other hand, *Fantasia3D* and *TextMesh(SD)* do not show any significant improvement when using priors.

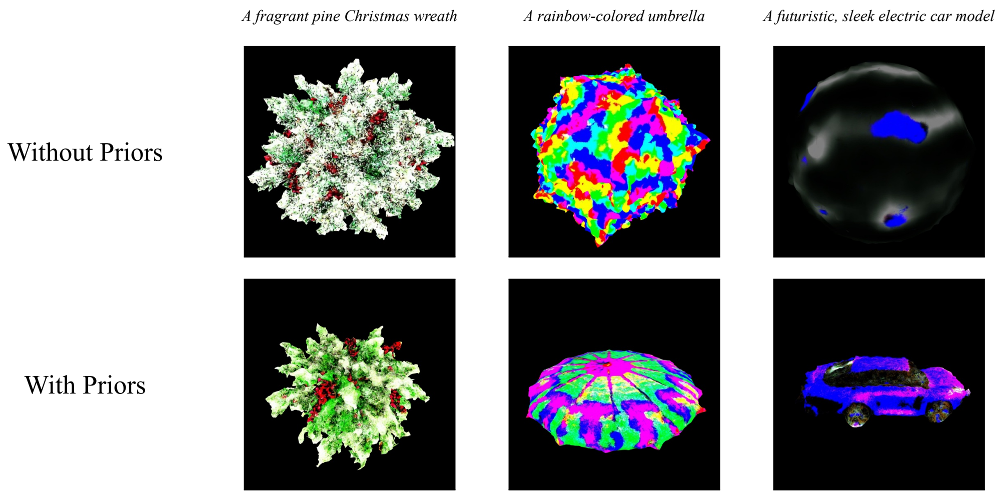

A fragrant pine Christmas wreath
A rainbow-colored umbrella
A futuristic, sleek electric car model
Without Priors
With Priors

Table 6.8: Comparison of the results obtained by Fantasia3D with and without using priors initialization for three different prompts.

The results obtained in Table 6.7 for *Fantasia3D* and *TextMesh(SD)* are not totally aligned with a possibile human evaluation directly performed on some renderings examples, as shown in Table 6.8. In fact, the renderings examples presented in Table 6.8 confirm that the *Fantasia3D* model highly benefits from the use of priors, showing again some difficulties by the T3bench metrics capture little improvements between these two different settings.

Diffusion Backbones

The third group of additional experiments is focused on models which have the possibility to exploit different diffusion backbones section 2.3 architectures to boost the image generation process. As we have already introduced in section 5.4, we have decided mainly to focus only 3 models which have the possibility to exploit different diffusion backbones.

Our Results

Model		CLIP		T3Bench		
name	type	Score	RPrec.	Qual.	Align.	Avg.
DreamFusion	SD	0.66	0.40	20.17	47.50	33.84
	IF	0.65	0.30	19.55	42.50	31.10
TextMesh	SD	0.53	0.05	09.06	32.00	20.53
	IF	0.65	0.30	17.70	39.00	28.35
Magic3D	SD	0.70	0.48	27.41	49.50	38.46
	IF	0.73	0.49	33.69	44.75	39.20

Table 6.9: Scores: overview of how different Diffusion Backbone affect the final scores obtained foreach metric.

Table 6.9 shows the results of the experiments we have conducted. We have tested *DreamFusion*, *TextMesh*, and *Magic3D* with the *StableDiffusion (SD)* and *DeepFloyd(IF)*[38] backbones.

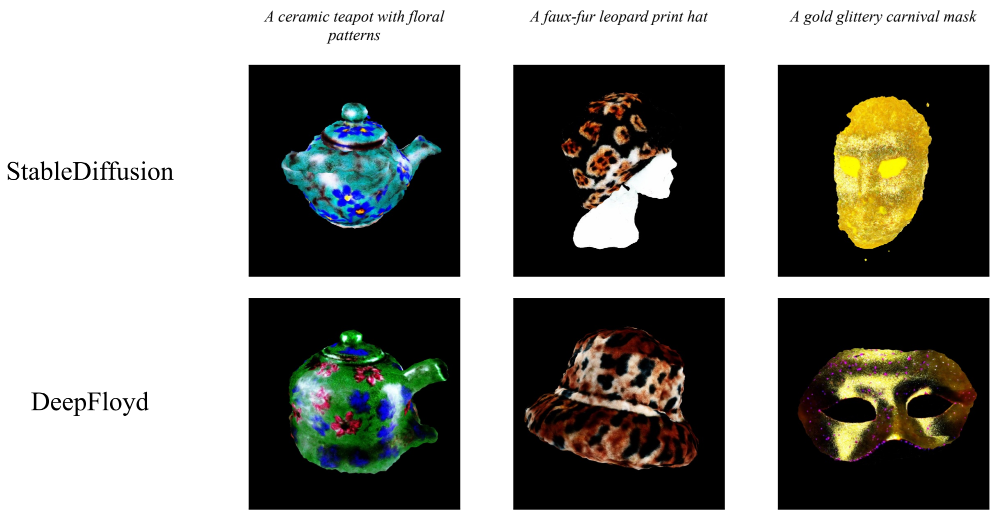
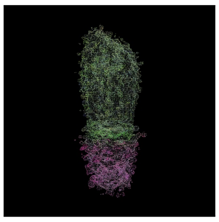
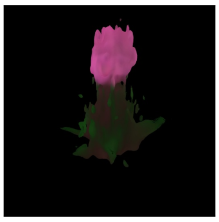

A ceramic teapot with floral patterns
A faux-fur leopard print hat
A gold glittery carnival mask
StableDiffusion
DeepFloyd

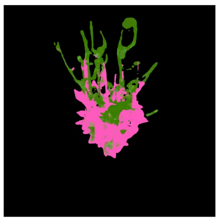
Table 6.10: Magic3D: comparison of the results obtained by the StableDiffusion (SD) and DeepFloy (IF) backbones for three different prompts.

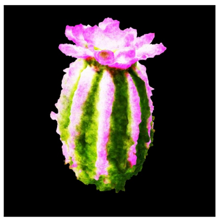
As we can see from the results, *TextMesh* is the only model for which we can see a significant improvement in the *CLIP-Score* and *T3Bench-Quality* when using the *DeepFloyd* backbone. The *CLIP-Score* increases from 0.53 to 0.65, and the *T3Bench-Quality* from 9.06 to 17.70. In the other hand, *DreamFusion* and *Magic3D* do not show any significant improvement when using the *DeepFloyd* backbone.

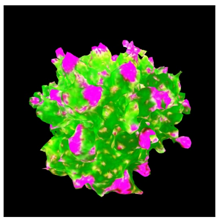
The results obtained in Table 6.9 for *Magic3D* are not totally aligned with a possible human evaluation directly performed on some renderings examples, as shown in Table 6.10. In fact, the renderings examples presented in Table 6.10 confirm that the *Magic3D* model highly benefits from the use of the *DeepFloyd(IF)* backbone, showing again some difficulties by the T3bench metrics capture little improvements between these two different settings.

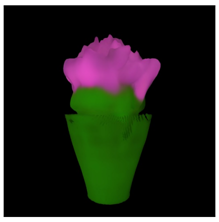

(1) LucidDreamer

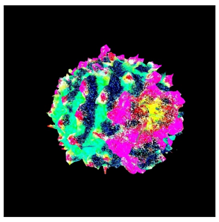

(2) Cap3D-ShapE

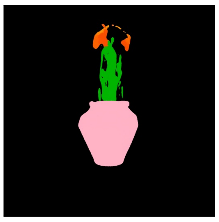

(3) Cap3D-PointE

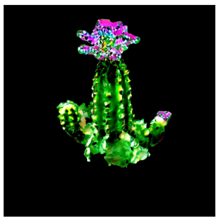

(4) HiFA

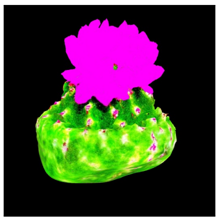

(5) ProlificDreamer

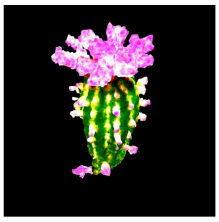

(6) ShapE


(7) Fantasia3D


(8) PointE


(9) SJC


(10) Magic3D(IF)


(11) LatentNeRF

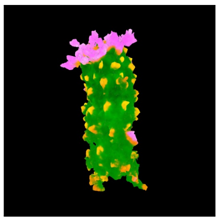
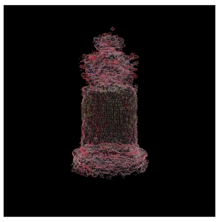
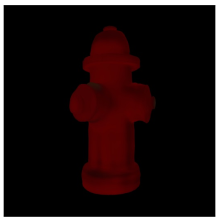
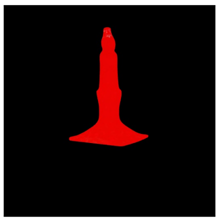
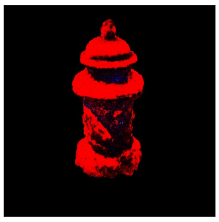

(12) DreamFusion

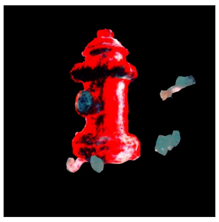
Table 6.11: Renderings extracted from all the implemented models for the prompt *"A cactus with pink flowers"*.

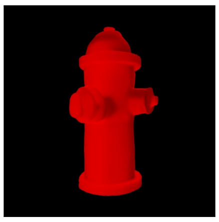

(1) LucidDreamer

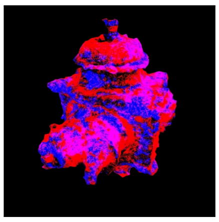

(2) Cap3D-ShapE

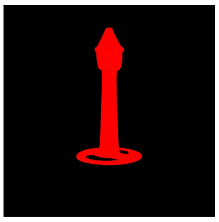

(3) Cap3D-PointE

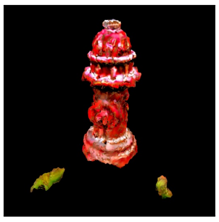

(4) HiFA

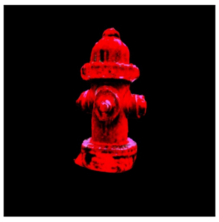

(5) ProlificDreamer

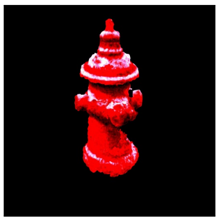

(6) ShapE


(7) Fantasia3D


(8) PointE


(9) SJC


(10) Magic3D(IF)


(11) LatentNeRF

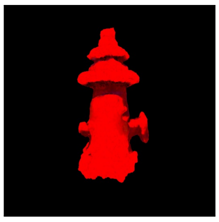
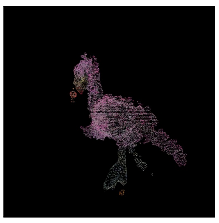
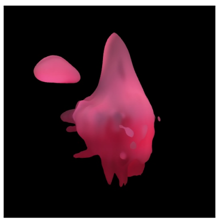
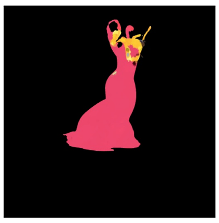
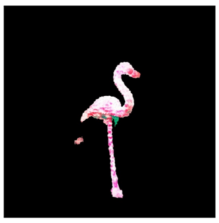

(12) DreamFusion

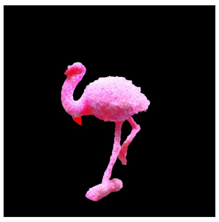
Table 6.12: Renderings extracted from all the implemented models for the prompt *"A bright red fire hydrant"*.

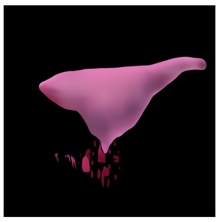

(1) LucidDreamer

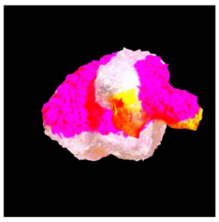

(2) Cap3D-ShapE

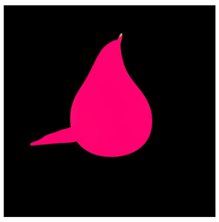

(3) Cap3D-PointE

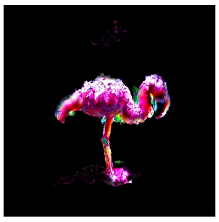

(4) HiFA

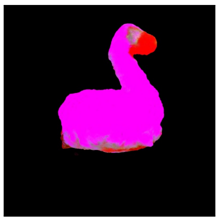

(5) ProlificDreamer

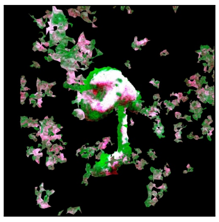

(6) ShapE


(7) Fantasia3D


(8) PointE


(9) SJC


(10) Magic3D(IF)


(11) LatentNeRF

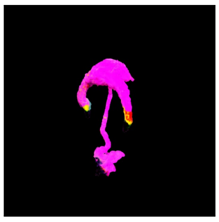

(12) DreamFusion

Table 6.13: Renderings extracted from all the implemented models for the prompt *"A fuzzy pink flamingo lawn ornament"*.

Chapter 7

Conclusion

In this work we have presented a comprehensive anyalis of the most recent text-driven 3D generation models. In particular, we have focused on evaluating the performances of some of the most recent models by imposing strong *efficiency* constraints related to the hardware requirements and the generation time that each model is allowed to use.

The goals of our research were to evaluate the performances of these *efficient* models respect to results obtained originally by each research without any constraints related to the generation configuration. Moreover, at the same time, we have integrated a prominent recent benchmark focused on defining a new evaluation standard for text-driven 3D generation models in order to assess the performances of these models respect to the *quality* and *coherence* (respect to the input prompt) of the generated 3D shapes.

After selecting the most promising models, we have conducted a series of experiments in order to asses the impact of our constraints on the general performances of these models and we have investigated the possibility of enhancing these performances by exploiting additional configurations (e.g. 3D priors initialization).

Most of the models that we have selected mainly rely on the use of powerful pre-trained diffusion models to optimize a 3D parametric representation of the object described in the input prompt. Notably, these models are able to produce high-quality 3D scenes without requiring any type of annotated 3D dataset. Due to the open-ended nature of the task, most studies evaluate their results ”manually” through subjective case studies.

In the other hand, we have explored the possibility of evaluating the performances of these *efficient* models by exploiting a recent benchmark named *T3Bench* in order to both deterministically and quantitatively assess the performances of these models and, at the same time, evaluate the alignment of this benchmark respect to human evaluation preferences.

Our results have shown that the overall fidelity of the generated 3D shapes highly depends on both the prompt statement and the model used. In particular, we have observed that the results obtained from some prompts are always very bad, while the results obtained from some other prompts are always pretty good, regardless of the model used.

In the other hand, we showed that the results obtained for another subset of prompts vary remarkably across models mainly due to the fact that different models exploit different 3D shape parametrization techniques.

As we were expecting, by constraining the generation time of each model, we have observed that most of the models generate 3D shapes which are more coarse respect to the ones obtained by the original researches. Despite this, some models are suprisingly able to generate 3D shapes which capture the semantic relationships of objects described in the input prompt and, at the same time, present a good level of details and a good topology.

In other hand, we have observed that, especially for models which originally requires a lot of computational time and resources, the generated 3D shapes are more coarse and have an high presence of artifacts.

We have investigated the correlation between the results described above and the scores obtained by the *T3Bench* benchmark. As we expected, since in general we generate more coarse 3D shapes, we have observed that most of our constrained models have obtained lower *Quality* scores respect to the original research.

Coversely, we have found that our *Alignment* scores are always higher respect to the ones obtained in the original research. This result is very interesting considering that the renderings obtained from our 3D shapes are less detailed and, as a consequence, present lower likelihood to be aligned respect to the input prompt.

In the Results section we have provided a detailed analysis of this aspect by exploring possibile causes of this behavior and by providing some examples of the renderings obtained from our 3D shapes.

In conclusion, our work shows that there is a huge potential for future research in the field of *efficient* text-driven 3D generation models. In particular, we believe that there is still a need for further analyses on how an *efficient* model may better learn more precise 3D shape details while capturing overall prompt semantic.

Moreover, such constrained scenario, pose a new challenge regarding the importance of the definition of a proper dataset of prompts that have to be used for evaluating these constrained models.

Finally, our results show how much the 3D quality highly depends on the prompt statement. Hence, we believe that the relationship between the 3D data parametrization defined by each model and a proper balance of between 3D *quality* and text-prompt *coherence* of the generated 3D shapes are the most interesting challenges for future researches.

Bibliography

[1] E. Ahmed, A. Saint, A. E. R. Shabayek, K. Cherenkova, R. Das, G. Gusev, D. Aouada, and B. Ottersten. A survey on deep learning advances on different 3d data representations, 2019. arXiv: 1808.01462 [cs.CV].

[2] R. Chen, Y. Chen, N. Jiao, and K. Jia. Fantasia3d: disentangling geometry and appearance for high-quality text-to-3d content creation, 2023. arXiv: 2303.13873 [cs.CV].

[3] P. Dhariwal and A. Nichol. Diffusion models beat gans on image synthesis, 2021. arXiv: 2105.05233 [cs.LG].

[4] J. A. et al. Gpt-4 technical report, 2023. arXiv: 2303.08774 [cs.CL].

[5] Y. He, Y. Bai, M. Lin, W. Zhao, Y. Hu, J. Sheng, R. Yi, J. Li, and Y.-J. Liu. T^{3}bench: benchmarking current progress in text-to-3d generation, 2023. arXiv: 2310.02977 [cs.CV].

[6] J. Hessel, A. Holtzman, M. Forbes, R. L. Bras, and Y. Choi. Clipscore: a reference-free evaluation metric for image captioning, 2022. arXiv: 2104.08718 [cs.CV].

[7] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018. arXiv: 1706.08500 [cs.LG].

[8] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models, 2020. arXiv: 2006.11239 [cs.LG].

[9] J. Ho and T. Salimans. Classifier-free diffusion guidance, 2022. arXiv: 2207.12598 [cs.LG].

[10] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: low-rank adaptation of large language models, 2021. arXiv: 2106.09685 [cs.CL].

[11] H. Jun and A. Nichol. Shap-e: generating conditional 3d implicit functions, 2023. arXiv: 2305.02463 [cs.CV].

[12] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis. 3d gaussian splatting for real-time radiance field rendering, 2023. arXiv: 2308.04079 [cs.GR].

[13] C. Li, C. Zhang, A. Waghwase, L.-H. Lee, F. Rameau, Y. Yang, S.-H. Bae, and C. S. Hong. Generative ai meets 3d: a survey on text-to-3d in aigc era, 2023. arXiv: 2305.06131 [cs.CV].

[14] J. Li, D. Li, C. Xiong, and S. Hoi. Blip: bootstrapping language-image pre-training for unified vision-language understanding and generation, 2022. arXiv: 2201.12086 [cs.CV].

[15] Y. Liang, X. Yang, J. Lin, H. Li, X. Xu, and Y. Chen. Luciddreamer: towards high-fidelity text-to-3d generation via interval score matching, 2023. arXiv: 2311.11284 [cs.CV].

[16] Y. Liao, S. Donné, and A. Geiger. Deep marching cubes: learning explicit surface representations. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2916–2925, 2018. DOI: 10.1109/CVPR.2018.00308.

[17] C.-H. Lin, J. Gao, L. Tang, T. Takikawa, X. Zeng, X. Huang, K. Kreis, S. Fidler, M.-Y. Liu, and T.-Y. Lin. Magic3d: high-resolution text-to-3d content creation, 2023. arXiv: 2211.10440 [cs.CV].

[18] C.-Y. Lin. Rouge: a package for automatic evaluation of summaries. In page 10, January 2004.

[19] T. Luo, C. Rockwell, H. Lee, and J. Johnson. Scalable 3d captioning with pretrained models, 2023. arXiv: 2306.07279 [cs.CV].

[20] G. Metzer, E. Richardson, O. Patashnik, R. Giryes, and D. Cohen-Or. Latent-nerf for shape-guided generation of 3d shapes and textures, 2022. arXiv: 2211.07600 [cs.CV].

[21] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: representing scenes as neural radiance fields for view synthesis, 2020. arXiv: 2003.08934 [cs.CV].

[22] A. Mordvintsev, N. Pezzotti, L. Schubert, and C. Olah. Differentiable image parameterizations. *Distill*, 2018. URL: https://api.semanticscholar.org/CorpusID:70083798.

[23] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics*, 41(4):1–15, July 2022. ISSN: 1557-7368. DOI: 10.1145/3528223.3530127. URL: http://dx.doi.org/10.1145/3528223.3530127.

[24] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, and M. Chen. Glide: towards photorealistic image generation and editing with text-guided diffusion models, 2022. arXiv: 2112.10741 [cs.CV].

[25] A. Nichol, H. Jun, P. Dhariwal, P. Mishkin, and M. Chen. Point-e: a system for generating 3d point clouds from complex prompts, 2022. arXiv: 2212.08751 [cs.CV].

[26] OpenAI. Point-e. https://github.com/openai/point-e, 2023.

[27] OpenAI. Shape-e. https://github.com/openai/shap-e, 2023.

[28] D. H. Park, S. Azadi, X. Liu, T. Darrell, and A. Rohrbach. Benchmark for compositional text-to-image synthesis. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. URL: `https://openreview.net/forum?id=bKBhQhPeKaF`.

[29] B. Poole, A. Jain, J. T. Barron, and B. Mildenhall. Dreamfusion: text-to-3d using 2d diffusion, 2022. arXiv: `2209.14988 [cs.CV]`.

[30] T. Project. Threestudio. `https://github.com/threestudio-project/threestudio`, 2023.

[31] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision, 2021. arXiv: `2103.00020 [cs.CV]`.

[32] E. Research. Lucid dreamer. `https://github.com/EnVision-Research/LucidDreamer`, 2023.

[33] C. Rockwell. Cap3d: scalable 3d captioning with pretrained models. `https://github.com/crockwell/Cap3D`, 2023.

[34] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models, 2022. arXiv: `2112.10752 [cs.CV]`.

[35] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans, 2016. arXiv: `1606.03498 [cs.LG]`.

[36] T. Shen, J. Gao, K. Yin, M.-Y. Liu, and S. Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis, 2021. arXiv: `2111.04276 [cs.CV]`.

[37] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations, 2021. arXiv: `2011.13456 [cs.LG]`.

[38] StabilityAI. Deepfloyd if. https://github.com/deep-floyd/IF, 2023.

[39] THU-LYJ-Lab. T3bench. https://github.com/THU-LYJ-Lab/T3Bench, 2023.

[40] C. Tsalicoglou, F. Manhardt, A. Tonioni, M. Niemeyer, and F. Tombari. Textmesh: generation of realistic 3d meshes from text prompts, 2023. arXiv: 2304.12439 [cs.CV].

[41] H. Wang, X. Du, J. Li, R. A. Yeh, and G. Shakhnarovich. Score jacobian chaining: lifting pretrained 2d diffusion models for 3d generation, 2022. arXiv: 2212.00774 [cs.CV].

[42] H. Wang and J. Zhang. A survey of deep learning-based mesh processing. *Communications in Mathematics and Statistics*, 10, February 2022. DOI: 10.1007/s40304-021-00246-7.

[43] Z. Wang, C. Lu, Y. Wang, F. Bao, C. Li, H. Su, and J. Zhu. Prolificdreamer: high-fidelity and diverse text-to-3d generation with variational score distillation, 2023. arXiv: 2305.16213 [cs.LG].

[44] J. Xu, X. Liu, Y. Wu, Y. Tong, Q. Li, M. Ding, J. Tang, and Y. Dong. Imagereward: learning and evaluating human preferences for text-to-image generation, 2023. arXiv: 2304.05977 [cs.CV].

[45] Q.-C. Xu, T.-J. Mu, and Y.-L. Yang. A survey of deep learning-based 3d shape generation. *Computational Visual Media*, 9(3):407–442, 2023. DOI: 10.1007/s41095-022-0321-5. URL: https://doi.org/10.1007/s41095-022-0321-5.

[46] L. Zhang, A. Rao, and M. Agrawala. Adding conditional control to text-to-image diffusion models, 2023. arXiv: 2302.05543 [cs.CV].

[47] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi. Bertscore: evaluating text generation with bert, 2020. arXiv: 1904.09675 [cs.CL].

[48] J. Zhu and P. Zhuang. Hifa: high-fidelity text-to-3d generation with advanced diffusion guidance, 2023. arXiv: `2305.18766 [cs.CV]`.

Acknowledgements

My heartfelt thanks to Professor Samuele Salti and Andrea Amaduzzi for their assistance at every stage of this research project.

I wish to extend my special thanks to my friends and family, whose support and encouragement have been invaluable to me throughout this journey.