Implementation and Analysis of DreamGaussian: Image-to-3D Mesh Conversion

Soumen Mondal (24SP06013), Himanshu Kumar(24SP06015)
IIT BHUBANESWAR
KHURDA, INDIA
Email:24sp06013@iitbbs.ac.in,24sp06015@iitbbs.ac.in

***Abstract*—This paper presents a comprehensive implementation and theoretical analysis of the DreamGaussian framework, a powerful approach for converting two-dimensional images into three-dimensional meshes. Utilizing advanced machine learning techniques such as diffusion models and Gaussian rasterization, this framework achieves high-precision reconstruction of 3D meshes from single-view images. The implementation was executed in a Python environment, specifically Google Colab with GPU acceleration.[2]**

I. Introduction

The generation of accurate 3D models from single-view 2D images has significant implications in computer vision, computer graphics, augmented reality, and virtual reality applications. Recent advancements, exemplified by the DreamGaussian method, have revolutionized 3D reconstruction techniques. This paper elaborates on the practical implementation of DreamGaussian, detailing the theory underpinning its methodologies and discussing observed results and implications.

II. Theory

A. Diffusion Models

Diffusion models, inspired by non-equilibrium thermodynamics, generate data by reversing a diffusion process. They progressively denoise an initially noisy input, guided by model parameters optimized through a Markov chain process described mathematically as:

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \tag{1}$$

where θ represents the learnable parameters, and x_t denotes the data at the diffusion step t.

B. Gaussian Rasterization

Gaussian rasterization encodes 3D points using oriented Gaussian distributions, capturing positional, scaling, and orientational attributes. This enables differentiable rendering from arbitrary camera viewpoints, formally expressed as:

$$I(u, v) = \sum_{i=1}^{N} G_i(u, v; \mu_i, \Sigma_i) \tag{2}$$

where $I(u, v)$ is the rendered image intensity at coordinates (u, v), μ_i is the mean, and Σ_i is the covariance matrix of the Gaussian.

III. Implementation Details

A. Environment Setup

The implementation utilized Google Colab with Python 3.11 and CUDA-enabled GPU (version 12.4). Essential libraries included PyTorch, diffusers, transformers, huggingface_hub, omegaconf, einops, trimesh, and xatlas.

B. Dependency Management

Conflicting dependencies necessitated isolated environments created with Python's built-in `venv`. Data communication between environments was achieved using JSON files as intermediate storage, ensuring smooth interoperability.

C. Diff_Gaussian_Rasterization

Due to compatibility issues with pre-built wheels, the `diff_gaussian_rasterization` module was manually compiled from source using the following commands:

```
git clone –recursive https://github.com/ashawkey/diff-gaussian-rasterization.git
pip install ./diff-gaussian-rasterization
```

D. Data Preprocessing and Mesh Generation

Raw input images underwent preprocessing to convert them to RGBA format using provided scripts (`process.py`). Mesh generation occurred over two stages, executed by scripts (`main.py` and `main2.py`), involving training, optimization, and refinement phases.

IV. Results and Discussion

The implementation yielded precise 3D meshes characterized by detailed textures and geometrical accuracy. GPU acceleration significantly enhanced computational efficiency, reducing training and rendering times markedly. However, practical challenges included managing complex library dependencies and ensuring compatibility across different modules.[1]

Key issues encountered included:

- Version conflicts between libraries such as `huggingface_hub`, `transformers`, and `tokenizers`, requiring careful dependency isolation.
- Difficulty installing the `diff_gaussian_rasterization` module due

Fig. 1. Original 2D Input Image showing the initial perspective and texture details.

Fig. 2. Preprocessed RGBA Image illustrating the transparent background and improved feature isolation.

to Python version compatibility, resolved by manually building the module from source.

- File handling and path issues, particularly with filenames containing spaces, solved by quoting paths and careful management of file names.
- Errors due to missing configuration files or incorrect mesh paths, addressed by explicitly specifying configurations and verifying file existence.

The generated results validated the robustness of diffusion models and Gaussian rasterization techniques in accurately reconstructing 3D geometry from single-view images. Potential improvements could involve enhanced automated dependency management tools or further algorithmic optimizations to boost performance.

A. Visualization

The figures below illustrate key stages of the DreamGaussian pipeline:

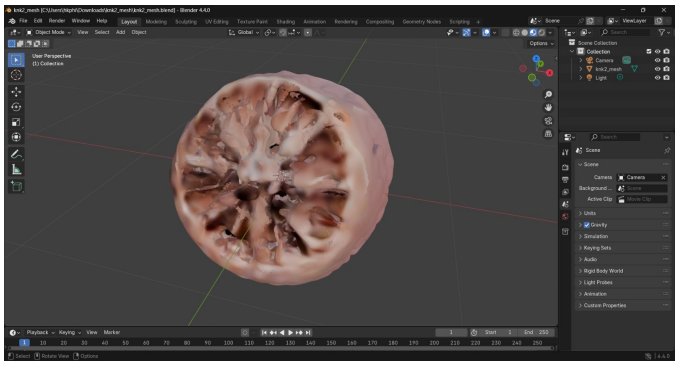

Fig. 3. Generated 3D Mesh after initial training (Stage 1), highlighting basic shape reconstruction.

V. Conclusion

The DreamGaussian pipeline provides a powerful and efficient solution for generating accurate 3D meshes from 2D images. The implementation, despite encountering dependency management complexities, demonstrated robust performance and accuracy. Future work should aim to refine dependency handling mechanisms and explore advanced computational optimizations to further enhance performance.

References

[1] DreamGaussian GitHub Repository. https://github.com/dreamgaussian/dreamgaussian

[2] J. Ho, A. Jain, and P. Abbeel, *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840-6851, 2020.

[3] B. Mildenhall *et al.*, *European Conference on Computer Vision*, 2020.