HW4 Report

Hsuan-Ting Lin

B11901132

1 3D Gaussian Splatting

1.1 What is 3D Gaussian Splatting?

3D Gaussian Splatting [1] is an innovative method for real-time radiance field rendering. A radiance field represents a continuous 5D function that models the distribution of light within a 3D space. It takes spatial coordinates and viewing angles as inputs and outputs color and opacity for each point in the scene.

Traditional methods like NeRF use a neural network to gain the implicit representation of the scene to be rendered, requiring high-cost training and long rendering time. In contrast, 3D Gaussian splattering uses 3D Gaussians for scene representation, which is an explicit representation with competitive training time and more importantly, enables real-time high-quality rendering.

1.1.1 Scene Representation

When representing 3D scenes, there are two approaches: implicit and explicit representations. Implicit representations such as NeRF use neural networks to represent scenes as continuous functions, while explicit methods directly store scene information in discrete structures like voxels or point clouds.

Gaussian splatting takes the explicit approach, it uses 3D Gaussians to build up a scene. A 3D Gaussian can be seen as a ellipsoid in 3D space, every Gaussian has its own position, covariance matrix that defines its shape and orientation, opacity and color coefficients represented using spherical harmonics for view-dependent appearance.

3D Gaussians are powerful as it provide smooth coverage of space, and could be easily projected to 2D for fast rendering. They could also be optimized using gradient descent, where the gradients are obtained from backpropagation.

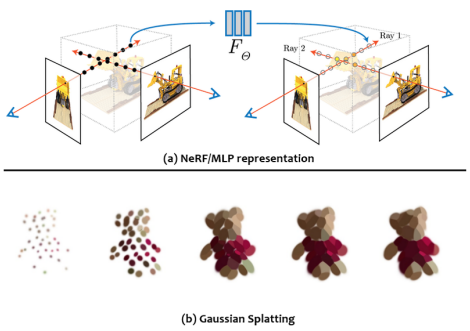

Figure 1: 3D representations using NeRF and 3D Gaussians [2]

1.1.2 Training Process

The Gaussians are initialized using SfM points obtained from feature matching between pairs of images of the same scene, forming a sparse point cloud. The Gaussians are then optimized through backpropagation, where the gradients are from the L1 and SSIM loss between ground truth images and rendered images.

One method worth noting is their adaptive density control, this involves densification and pruning. Densification is the process of adding new Gaussians in underrepresented or under/over-reconstructed regions to capture finer details, while pruning removes redundant Gaussians with low opacity or those contributing minimally to the scene representation.

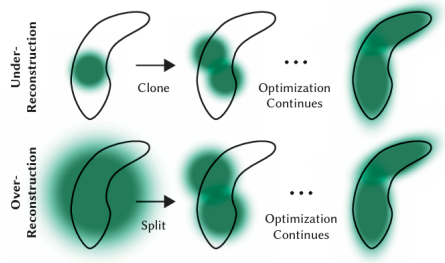

Figure 2: Adaptive Gaussian densification scheme

1.1.3 Rendering

To project Gaussains to form images, the covariance martricies are converted into 2D covariance matricies using viewing transformation matricies. To ensure correct visibility and blending, Gaussians are sorted along the depth axis from the camera's perspective, and the color of each pixel is computed by blending overlapping Gaussians.

A key method to enable real-time rendering is use tile-based rendering for parallelism. The image is split into 16*16 pixel tiles, each processes its contributing Gaussians independantly, reducing computational overhead.

1.2 3DGS vs NeRF

Listed below are the pros and cons of NeRF and 3D Gaussain Splatting:

Aspect	**Pros**	**Cons**
NeRF	• Compact representation (∼8.6MB) • Continuous scene representation • Good view interpolation	• Slow rendering (0.1 FPS) • Long training time (48h) • Uneditable • Computationally expensive ray marching
3DGS	• Real-time rendering (90-150+ FPS) • Fast training (6-51 min) • Editable • State-of-the-art quality	• Large memory footprint (270-734MB) • Popping artifacts possible • Memory grows with scene complexity

Table 1: General comparison of NeRF and 3DGS (3D Gaussian splatting)

1.3 Most Important Part of 3DGS

In my opinion, the most important aspect of 3D Gaussian splatting is its ability to render in real time. There are numerous applications that rely on 3D scene representations, such as video games, VR/AR, real-time visualization of various models, and more. Most of these applications require real-time rendering, which was previously not feasible for high-quality images before the advent of 3D Gaussian splatting.

Moreover, 3D Gaussian splatting achieves competitive training times and supports editing capabilities, making this method even more groundbreaking.

2 Implementation Details of 3D Gaussian Splatting

I used this Github repo [3] for 3D Gaussian splatting implementation, which was forked from the official repo. For this repo, the training pipeline can be described as follows:

a) **Parameter Initialization:**

- Position (xyz): initialized from SfM point cloud
- Colors: convert RGB to spherical harmonics coefficients
- Scale: computed from nearest neighbor distances
- Rotation: initialized as identity quaternions
- Opacity: set to initial value of 0.1

b) **Training Loop:**

- Randomly select camera view from training set
- Render current Gaussians using differentiable rasterizer
- Compute loss combining L1 and DSSIM metrics
- Update parameters using Adam optimizer

c) **Adaptive Density Control:**

- Track gradient magnitudes and screen-space size of Gaussians
- Clone Gaussians in regions with high gradients (under-reconstruction)
- Split large Gaussians with high gradients (over-reconstruction)
- Remove Gaussians with low opacity or excessive size
- Reset opacity periodically to avoid floaters

d) **Progressive Training:**

- Increase spherical harmonics degree every 1000 iterations
- Start with low resolution then progressively increase
- Apply exponential learning rate decay for positions
- Continue until maximum iterations reached

e) **Evaluation and Checkpointing:**

- Periodically compute PSNR and L1 metrics
- Save model checkpoints at specified iterations

3 Comparison of Performance Metrics Across Hyperparameter Settings

3.1 Performances Under Different Settings

The parameters not explicitly assigned are set to their default values from the official GitHub repository. Starting with the first checkpoint that achieves the baseline, I experimented with simultaneously lowering or raising all learning rates by a factor r, except for the positional learning rate, which I kept constant across all settings. This decision was based on its significant impact on the score, as I wanted to isolate the effects of other learning rates.

Here are the parameters I tuned across different settings:

```
# parameters in settings different from default, experiment with r
self.iterations = 120_000                # default = 30_000
self.position_lr_init = 0.000016         # default = 0.00016
self.position_lr_max_steps = 60_000      # default = 30_000
self.feature_lr = 0.025 * r
self.opacity_lr = 0.05 * r
self.scaling_lr = 0.001 * r
self.rotation_lr = 0.001 * r
self.lambda_dssim = 0.3                  # default = 0.2
```

Setting	PSNR	SSIM	LPIPS (vgg)	# 3D gaussians
$r = 0.66$	35.8614	0.97551	0.0843	641884
$r = 0.8$	35.7167	0.97552	0.0848	607848
$r = 1.0$	36.1908	0.97612	0.0810	595846
$r = 1.2$	36.2480	0.97595	0.0812	602146
$r = 1.5$	36.7449	0.97676	0.0780	616252

Table 2: Quantitative results with different learning rate ratio

I found out for this range of r, the performace of PSNR and SSIM generally increases with increasing learning rate while LPIPS descreases, which may indicate some kind of performace trade-off.

The number of Gaussains does not depend on r in a linear fashion, using the least when $r = 1$ and increasing with higher or lower r.

3.2 Metric Explanations

3.2.1 PSNR

PSNR (Peak Signal to Noise Ratio) is a term for the ratio between the maximum possible power of a signal (in this case, an image) and the power of corrupting noise, expressed in log scale. It is most commonly used to measure the quality of reconstruction of lossy compression codecs. When comparing compression codecs, PSNR is an approximation to human perception of reconstruction quality. [4]

For RGB images with 3 channels, we first define the mean squared error (MSE) between an original image and its approximation:

$$MSE = \frac{1}{3mn} \sum_{c=0}^{2} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j,c) - K(i,j,c)]^2 \tag{1}$$

where c represents the RGB channels, m and n are the dimensions of the image. PSNR is then defined as:

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) = 20 \cdot \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \tag{2}$$

where MAX_I is the maximum possible pixel value of the image. For 8-bit image representation (e.g., in image processing), MAX_I is 255.

3.2.2 SSIM

SSIM (Structural Similarity Index Measure) is a method used for measuring the similarity between two images. Unlike traditional metrics like Mean Squared Error (MSE) or Peak Signal-to-Noise Ratio (PSNR), which only focus on pixel-by-pixel differences, SSIM evaluates the perceived visual quality of an image by taking into account structural information, luminance, and contrast. [5]

SSIM is calculated on various windows of an image. The measure between two windows x and y of common size $N \times N$ is:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{3}$$

where:

- μ_x is the pixel sample mean of x
- μ_y is the pixel sample mean of y
- σ_x^2 is the variance of x
- σ_y^2 is the variance of y
- σ_{xy} is the covariance of x and y
- $c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$ are variables to stabilize division with weak denominator
- L is the dynamic range of pixel values (typically $2^{\text{bits per pixel}} - 1$)
- $k_1 = 0.01$ and $k_2 = 0.03$ by default

The overall SSIM score is then taken as the mean SSIM value across all possible sliding windows in the image.

3.2.3 LPIPS

LPIPS (Learned Perceptual Image Patch Similarity) is a perceptual metric that measures image similarity using deep network feature spaces. Unlike traditional metrics like MSE, PSNR, or SSIM that operate directly on pixel values, LPIPS leverages the learned representations from pre-trained neural networks (typically AlexNet, VGG, or SqueezeNet) to better match human perceptual judgments. [6]

The LPIPS distance between two images x and y is computed as:

$$d_{LPIPS}(x, y) = \sum_l \frac{1}{H_l W_l} \sum_{h,w} \|w_l \odot (F_l^x(h, w) - F_l^y(h, w))\|_2^2 \tag{4}$$

where:

- F_l^x, F_l^y are the feature maps extracted from layer l of a pre-trained network for images x and y respectively
- H_l, W_l are the height and width of the feature maps at layer l
- w_l represents learned channel weights for layer l
- $\odot$ denotes element-wise multiplication
- $\|\cdot\|_2^2$ is the squared L2 norm

The metric first extracts deep features from both images using a pre-trained network, computes normalized L2 distances between these features, and then takes a weighted sum across different layers. Lower LPIPS scores indicate greater perceptual similarity between images.

4 3D Gaussians With Random Initializing Points

4.1 Method

Since there are about 13000 SfM points for initialization in training dataset, I initailized 15000 Gaussian points at the start of training. The Gaussians are initialized randomly by

- **Position (xyz):** Uniformly sampled in a bounded volume $[-\text{scene\_extent}, \text{scene\_extent}]^3$, where scene_extent is determined by camera positions
- **Color Features:** Initialize spherical harmonics coefficients where
  - DC term (base color): Random RGB values in $[0, 1]$
  - Higher degree terms: All initialized to 0
  - Shape: $[\text{num\_points}, 3, (\text{max\_sh\_degree} + 1)^2]$
- **Scale:** All Gaussians start with small uniform scale (0.01) in each dimension, stored in log space
- **Rotation:** Initialize as identity quaternions $[1, 0, 0, 0]$, representing no rotation
- **Opacity:** All Gaussians start with opacity 0.1, transformed through inverse sigmoid for optimization

4.2 Performances

The settings are the same as those in Section 3.1, the only difference is that the Gaussians are initialized randomly.

Setting	PSNR	SSIM	LPIPS (vgg)	# 3D gaussians
$r = 0.66$	17.2784	0.61040	0.5590	364910
$r = 0.8$	17.0456	0.60167	0.5599	447030
$r = 1.0$	16.7622	0.59731	0.5632	534900
$r = 1.2$	16.8497	0.5930	0.5646	631210
$r = 1.5$	16.5101	0.58630	0.5709	695151

Table 3: Quantitative results with different learning rate ratio

When the Gaussians are initialized randomly, there is a clear trend between the metrics and r. As the learning rate increases, PSNR and SSIM decrease, while LPIPS(vgg) and the number of Gaussians increase. This trend could suggest that when the initial Gaussians are coarse, a higher learning rate captures the overall structure of the scene better (reflected in the higher LPIPS score), but finer details (as indicated by PSNR and SSIM) cannot be optimized simultaneously.

As expected, the performance of all metrics is lower compared to using SfM points for initialization, regardless of whether more or fewer Gaussian points are used.

To visually compare the differences between initializing from SfM points and initializing randomly, here is a comparison:

Figure 3: Rendered images using randomized Gaussians (left) and SfM points (right)

Acknowledgments

Some of the code in this project was generated with assistance from `ChatGPT` [7] and `Claude` [8]. Their contributions include code for model training, model evaluation, and general Python scripting.

References

[1] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," 2023. [Online]. Available: https://arxiv.org/abs/2308.04079

[2] A. Dalal, D. Hagen, K. G. Robbersmyr, and K. M. Knausgård, "Gaussian splatting: 3d reconstruction and novel view synthesis: A review," *IEEE Access*, vol. 12, p. 96797–96820, 2024. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2024.3408318

[3] camenduru, "3d gaussian splatting for real-time radiance field rendering," 2024, accessed: 2024-11-25. [Online]. Available: https://github.com/camenduru/gaussian-splatting

[4] Wikipedia contributors, "Peak signal-to-noise ratio — Wikipedia, the free encyclopedia," 2024, [Online; accessed 23-November-2024]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Peak_signal-to-noise_ratio&oldid=1247121104

[5] ——, "Structural similarity index measure — Wikipedia, the free encyclopedia," 2024, [Online; accessed 23-November-2024]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Structural_similarity_index_measure&oldid=1255137220

[6] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," 2018. [Online]. Available: https://arxiv.org/abs/1801.03924

[7] OpenAI, "Chatgpt: Gpt-4 model," `https://chat.openai.com`, 2024, `https://chat.openai.com`.

[8] Anthropic, "Claude: A large language model by anthropic," `https://www.anthropic.com`, 2024, `https://www.anthropic.com`.