DLCV HW4

NTU, FALL 2024

B10901151 林祐群
指導教授： 王鈺強
2024年12月2日

Problem: 3D Novel View Synthesis

1. (15%) Please explain:

A. Try to explain 3D Gaussian Splatting in your own words

B. Compare 3D Gaussian Splatting with NeRF (pros & cons)

C. Which part of 3D Gaussian Splatting is the most important you think? Why?

= =

Answer A:

3D Gaussian Splatting is a method for representing and rendering 3D scenes using Gaussian distributions. In this approach, a scene is represented as a set of 3D Gaussian points, each defined by parameters such as position, size, orientation, color, and opacity. These Gaussian points are projected onto the image plane during rendering, and their properties are blended to create high-quality 2D images.

The "smooth" nature of Gaussian distributions allows for natural handling of blurring and anti-aliasing effects, making it a robust way to visualize 3D data.

In a nutshell, 3D Gaussian Splatting converts a scene into a cloud of Gaussian points and optimizes their parameters to render realistic images.

Answer B:

NeRF (Neural Radiance Fields):

NeRF learns a scene's radiance field using a neural network and generates images through volumetric rendering.

- Pros:
  1. High-quality detail reproduction
  2. Flexibility (e.g. reflections and refractions)
  3. Versatility (for both static and dynamic scenes)
- Cons:
  1. High computational cost (both training and rendering)

2. Slow rendering
3. Large model size

3D Gaussian Splatting:

Represents a scene as a point cloud of 3D Gaussians and directly renders them without relying on a deep neural network.

- Pros:
  1. Fast rendering (need not of neural network)
  2. Lower resource requirements
  3. Intuitive representation
- Cons:
  1. Loss of detail
  2. Scene limitations (e.g. for complex lighting or dynamic scenes)
  3. Data dependence

Overall,

NeRF is ideal for high-precision rendering and complex lighting effects but is computationally expensive. 3D Gaussian Splatting provides a faster, resource-efficient alternative, suitable for quick visualization tasks.

Answer C:

The most critical part of 3D Gaussian Splatting, in my opinion, is the "optimization of Gaussian point parameters." The reasons are three-folded:

1. Core to visual quality: Parameters such as position, covariance (shape/size), color, and opacity directly determine the rendered image's fidelity.

2. Efficiency: Balancing the number of Gaussian points and their properties ensures both rendering speed and accuracy.

3. Adaptability: Proper parameterization allows 3D Gaussian Splatting to generalize across a variety of scenes, making it more versatile.

It is the key to achieving a balance between rendering quality and computational efficiency.

2. (15%) Describe the implementation details of ***your 3D Gaussian Splatting*** for the given dataset. You need to explain your ideas completely.

- You have to train 3D gaussians to represent a scene on the given dataset without loading pre-trained weights.
- We provide the following Github links for your reference.

3D Gaussian Splatting - LINK (recommended)

= =

The input dataset includes the following components:

1. Sparse 3D Points (`points3D.ply`): A point cloud representing the 3D scene geometry.
2. Camera Information: Calibration parameters such as intrinsics, extrinsics, and quaternion-based orientations.
3. Image Dataset: RGB images corresponding to different views.

Preprocessing Steps:

1. Normalize the Point Cloud:
   A. Translate the 3D points to center them at the origin by subtracting the mean.
   B. Scale the points to normalize their spatial range for stability.
2. Camera Transformations:
   1. Convert camera orientations from quaternions to rotation matrices.
   2. Extract the camera centers in world coordinates for rendering.

SFM-Based Initialization:

1. Extract points and colors directly from `points3D.ply`.
2. Normalize points and set colors from the dataset.
3. Provides strong alignment with the scene geometry.

Training:

Loss Functions:

1. Photometric Loss: Compare rendered and ground-truth images using:

- L1 loss for pixel-wise intensity differences.
- SSIM loss for structural similarity.

2. Depth Consistency Loss: Enforce alignment with ground-truth depth maps.
3. Opacity Regularization: Penalize overly transparent Gaussians to avoid excessive blending.

Optimization:

Use an optimizer like **Adam** or **Sparse Gaussian Adam** to update the Gaussian parameters (location, scaling, color, opacity, rotation).

Rendering Pipeline:

1. Gaussian Blending: Project Gaussians into the image space using camera intrinsics and extrinsics based on distance, opacity, and scaling.
2. Shading: Use spherical harmonics to simulate light interactions.

The rendering is performed using a differentiable rasterizer. Each Gaussian contributes to the rendered pixel intensities based on its distance and transparency.

3. (15%) Given novel view camera pose, your 3D gaussians should be able to render novel view images. Please evaluate your generated images and ground truth images with the following three metrics (mentioned in the 3DGS paper). Try to use at least **three** different hyper-parameter settings and discuss/ analyze the results.

A. Please report the PSNR/SSIM/LPIPS on the public testing set.
B. Also report the number of 3D gaussians.
C. You also need to explain the meaning of these metrics.
D. Different settings such as learning rate and densification interval, etc.

= =

I. Evaluating train:

(--position_lr_init 0.01 --position_lr_final 0.001 --feature_lr 0.01)

L1	0.010107
PSNR	35.162138
SSIM	0.966268
LPIPS	0.061759
Number of 3D gaussians:	1091172

II. Evaluating train:

(--position_lr_init 0.005 --position_lr_final 0.0005 --feature_lr 0.005)

L1	0.007982
PSNR	37.381929
SSIM	0.978721
LPIPS	0.039063
Number of 3D gaussians:	1414957

III. Evaluating train:

(--position_lr_init 0.001 --position_lr_final 0.0001 --feature_lr 0.001)

L1	0.007338

PSNR	38.587840
SSIM	0.983055
LPIPS	0.034990
Number of 3D gaussians:	827238

Metrics and Their Meaning

PSNR (Peak Signal-to-Noise Ratio):

1. Measures the image reconstruction quality.

2. Higher PSNR indicates closer similarity between the rendered and ground truth images.

3. Sensitive to pixel-level differences.

SSIM (Structural Similarity Index):

1. Evaluates perceived image quality based on structural information.

2. Higher SSIM indicates better preservation of spatial structures and textures in the image.

LPIPS (Learned Perceptual Image Patch Similarity):

1. Measures perceptual similarity using deep features.

2. Lower LPIPS indicates better alignment in terms of human-perceived quality.

Settings	L1 Loss	PSNR	SSIM	LPIPS	Number of Gaussians

`position_lr_init=0.01`
`position_lr_final=0.001`
`feature_lr=0.01`
	0.0101	35.1621	0.9662	0.0617	1091172

`position_lr_init=0.005`
`position_lr_final=0.0005`
`feature_lr=0.005`
	0.0079	37.3819	0.9787	0.0390	1414957

`position_lr_init=0.001`
`position_lr_final=0.0001`
`feature_lr=0.001`
	0.0073	38.5878	0.9831	0.0349	827238

4. (15%) Instead of initializing from SFM points [dataset/sparse/points3D.ply], please try to train your 3D gaussians with random initializing points.

A. Describe how you initialize 3D gaussians

B. Compare the performance with that in previous question.

= =

To train the 3D Gaussians using randomly initialized points, it's needed to replace the SFM-based initialization (`points3D.ply`) with a method that generates random points distributed within a specific range in 3D space.

1. Define the Spatial Bounds:

- Determine the bounding box (e.g., `[-1, 1]` in all axes) within which the random 3D points will be distributed. This range corresponds to the expected size and scale of the scene.

2. Generate Random Points:

- Use a uniform random distribution to generate 3D coordinates for each Gaussian point.

e.g. :

```
# Random XYZ positions within the range [-1, 1]
xyz = np.random.uniform(-1, 1, size=(n_points, 3))
# Random RGB colors between [0, 1]
colors = np.random.uniform(0, 1, size=(n_points, 3))
# Convert to PyTorch tensors
xyz_tensor = torch.tensor(xyz, dtype=torch.float32, device=device)
colors_tensor = torch.tensor(colors, dtype=torch.float32,
        device=device)
# Initialize Gaussian properties
gaussians = GaussianModel(sh_degree=sh_degree)
gaussians._xyz = xyz_tensor
```

3. Use Random Initialization in Training:

- Replace the SFM initialization in the training function with the random initialization.

Aspect	SFM-Based Initialization	Random Initialization
Convergence Speed	Faster due to better alignment with scene geometry.	Slower as random points lack initial alignment.
PSNR/SSIM	High, as points are aligned with the scene geometry. (38.59 / 0.983)	Lower, as random points need more time to adapt. (29.77 / 0.875)
Stability	Stable training due to structured initialization.	Instability in early stages.

References

1. 3DGS GitHub:

https://github.com/graphdeco-inria/gaussian-splatting

2. Survey on 3DGS:

https://arxiv.org/pdf/2401.03890

3. PSNR, SSIM:

https://adam-study-note.medium.com/電腦視覺-圖像衡量指標psnr-ssim介紹-a4fd76d9d84d

4. LPIPS:

https://github.com/richzhang/PerceptualSimilarity

5. Point Cloud Initialization:

https://github.com/graphdeco-inria/gaussian-splatting/issues/118

6. COLMAP:

https://colmap.github.io/format.html

7. NERF:

https://www.matthewtancik.com/nerf

8. NERF v.s. Gaussian Splatting:

https://www.hammermissions.com/post/comparing-nerf-3d-gaussian-splatting-and-drone-photogrammetry

9. ChatGPT

10. Claude