hw4_r13921068

DLCV HW4 Report

學號：**r13921068**

姓名：吳家萱

Problem : 3D Novel View Synthesis

1. Please explain:
    a. Try to explain 3D Gaussian Splatting in your own words

- 3D Gaussian Splatting 是一個 3D 重建的架構，能從一堆 2D 圖片中利用光柵化學到該場景的顯示表示法，結合了 NeRF 和點雲渲染的優點
- 3D Gaussian Splatting 透過訓練顯式表示法，將場景表示為空間中 3D 高斯函數的集合，每個高斯函數代表場景中的某個部分，並存儲其位置、形狀、顏色和不透明度等資訊，更快速的渲染算法，從而實現實時渲染
- 3D Gaussian Splatting render 的方式不像 NeRF 使用 ray tracing （光線追蹤）而是使用 rasterization（光柵化），把場景中的東西朝螢幕投影，看看有哪些像素被占據，把路線上獲得的資訊整理、累積起來（像把東西砸扁在螢幕上 splat），具體來說，它用了 tile-based 的方法，把畫面分成很多小格子，然後用 GPU 快速排序和混合這些高斯函數。這種優化方式讓它能達到即時渲染的速度（30 FPS 以上），而且畫質還超級棒
- 而且訓練相較 NeRF 速度快上兩個量級，因為沒有隱式表示法就沒有神經網路，不需要每個點都去問神經網路，且 GPU 十分適合 rasterization，並且顯示表示法可以直接被 import 到各種 3D 引擎中使用，可以更值觀的支援編輯

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

NeRF 是利用 ray tracing （光線追蹤）去訓練一個隱式表示法，而 3D Gaussian Splatting 是利用 rasterization（光柵化）去訓練一個顯示表示法

pros:

	NeRF	3D Gaussian Splatting
pros	Quality 上限高、 較為真實、 能很好處理反射折射陰影材質	速度快、 預覽視覺化容易 較簡單
cons	速度慢、 難以預覽、 較複雜	較不真實、 無法很好處理反射折射陰影材質

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

- 我覺得 3D Gaussian Splatting 最重要的部分是使用 rasterization（光柵化）去訓練一個顯示表示法，透過基於 tile 的渲染方式加上深度排序，不只保持了視覺品質，還讓即時渲染變得可能
- 以前的 NeRF 方法都要用較慢的 ray tracing （光線追蹤），每次都要採樣許多點，但用 3D Gaussian Splatting rasterization 直接把 3D Gaussian 投影到 2D，然後用 GPU 原生的架構去渲

染，效率提升很多

- 而且因為使用了 rasterization，訓練完的模型直接就能用傳統的渲染管線去即時渲染，打破了之前 NeRF 方法的速度限制，讓 3D Gaussian Splatting 速度提升很多

2. Describe the implementation details of your 3D Gaussian Spaltting for the given dataset. You need to explain your ideas completely.

- 我的 3D Gaussian Spaltting 實作方法參考了 3D Gaussian Splatting for Real-Time Radiance Field Rendering Github
- 我在 dataset_readers.py 的 readColmapSceneInfo 修改如下：
  - 我移除了先嘗試讀取二進制 (.bin) 文件的邏輯，直接使用 images.txt 和 cameras.txt 文件，使用 read_extrinsics_text 和 read_intrinsics_text 來讀取，移除沒有使用到的函式，例如所有深度相關處理的程式

```python
def readColmapSceneInfo(path, images, depths, eval, train_test_exp, llffhold=8):
    try:
        cameras_extrinsic_file = os.path.join(path, "sparse/0", "images.txt")
        cameras_intrinsic_file = os.path.join(path, "sparse/0", "cameras.txt")

        cam_extrinsics = read_extrinsics_text(cameras_extrinsic_file)
        cam_intrinsics = read_intrinsics_text(cameras_intrinsic_file)

        images_folder = os.path.join(path, "images")
        os.makedirs(images_folder, exist_ok=True)

        if eval:
            test_cam_names_list = []
            if "360" in path or llffhold:
                cam_names = [cam_extrinsics[cam_id].name for cam_id in cam_extrinsics]
                cam_names = sorted(cam_names)
                test_cam_names_list = [name for idx, name in enumerate(cam_names) if idx % llffhold == 0]
                print(f"Selected {len(test_cam_names_list)} cameras for testing")
        else:
            test_cam_names_list = []

        cam_infos = readColmapCameras(
            cam_extrinsics=cam_extrinsics,
            cam_intrinsics=cam_intrinsics,
            depths_params=None,
            images_folder=images_folder,
            depths_folder="",
            test_cam_names_list=test_cam_names_list
        )

        cam_infos = sorted(cam_infos, key=lambda x: x.image_name)

        # Decide which cameras go to train and test sets
        if eval:
            train_cam_infos = [c for c in cam_infos if c.image_name not in test_cam_names_list]
            test_cam_infos = [c for c in cam_infos if c.image_name in test_cam_names_list]
        else:
            train_cam_infos = cam_infos
            test_cam_infos = cam_infos

        print(f"Split into {len(train_cam_infos)} training and {len(test_cam_infos)} test cameras")

        nerf_normalization = getNerfppNorm(train_cam_infos)
```

  - 因為在 testing（eval=True） 時，我們只需要渲染新視角，不需要進行場景重建和參考原始的點雲數據，因此不需要使用 points3D.ply

```python
    # For validation/test set, we don't need point cloud data
    if eval:
        pcd = None
    else:
        ply_path = os.path.join(path, "sparse/0/points3D.ply")
        bin_path = os.path.join(path, "sparse/0/points3D.bin")
        txt_path = os.path.join(path, "sparse/0/points3D.txt")

        if not os.path.exists(ply_path):
            try:
                xyz, rgb, _ = read_points3D_binary(bin_path)
                storePly(ply_path, xyz, rgb)
            except FileNotFoundError:
                try:
                    xyz, rgb, _ = read_points3D_text(txt_path)
                    storePly(ply_path, xyz, rgb)
                except FileNotFoundError:
                    pcd = None

        if os.path.exists(ply_path):
            try:
                pcd = fetchPly(ply_path)
            except:
                pcd = None
        else:
            pcd = None

    scene_info = SceneInfo(point_cloud=pcd,
                           train_cameras=train_cam_infos,
                           test_cameras=test_cam_infos,
                           nerf_normalization=nerf_normalization,
                           ply_path=ply_path if not eval else "",
                           is_nerf_synthetic=False)
    return scene_info
```

- 我在 render.py 修改了以下設計：
  - 讓 output 的圖片根據從 image.txt 獲得的 image_name 來儲存圖片名稱，也就是以原本 input image {id}.png 的 id 來儲存圖片名稱

```python
def render_set(model_path, iteration, views, gaussians, pipeline, background, train_test_exp, separate_sh, output_path):
    render_path = output_path
    # render_path = os.path.join(output_path, "renders")
    makedirs(render_path, exist_ok=True)

    for idx, view in enumerate(tqdm(views, desc="Rendering progress")):
        try:
            # 修改這裡的 render 調用，移除 use_trained_exp 和 separate_sh 參數
            render_pkg = render(view, gaussians, pipeline, background)
            rendering = render_pkg["render"] if isinstance(render_pkg, dict) else render_pkg

            # Get original filename
            original_filename = view.image_name

            # Save rendered image to output path
            output_filepath = os.path.join(render_path, original_filename)
            torchvision.utils.save_image(rendering, output_filepath)

            # print(f"Successfully rendered and saved view {idx} to {output_filepath}")

        except Exception as e:
            print(f"Error rendering view {idx}: {str(e)}")
            import traceback
            traceback.print_exc()
```

- 我在 train.py 增加了以下設計：
  - 用來計算 3D gaussians 的數量

```
192     final_gaussian_count = scene.gaussians.get_xyz.shape[0]
193     print("\n" + "="*50)
194     print(f"Training complete.")
195     print(f"Final number of 3D Gaussians: {final_gaussian_count:,}")
196     print("="*50)
197
```

- 我在進行 training 時嘗試了不同的參數，參數值和修改原因如下：
  - --densify_from_iter 1000：給予高斯體初始優化的時間，讓基本的場景結構先形成，避免過早的密度化導致不穩定，在我的設定中會較晚才開始密度化，讓初始高斯體有足夠時間優化基本形狀
  - --densification_interval 900：減少密度化頻率，每 900 代才密度化，給予更多時間進行優化，為了避免過度密度化
  - --densify_grad_threshold 0.0001：設定密度化的梯度閾值，透過較小的閾值更容易觸發密度化，就能夠捕捉更細微的場景細節，以產生更精細的重建結果
  - --opacity_reset_interval 9999999：設定不透明度重置的間隔，設為極大值的原因為讓我的不透明度不會定期重置，好讓不透明度可以自然收斂，避免突然的變化影響訓練穩定性

arguments	value
--densify_from_iter	1000
--densification_interval	900
--densify_grad_threshold	0.0001
--opacity_reset_interval	9999999

- 在這個設計中，我在 public test dataset 成績可以在各個評估指標獲得如下成績：

PSNR	SSIM	LPIPS	the number of 3D gaussians
38.07125234603882	0.9802033295129479	0.07274182423949242	378,168

3. 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 hyperparameter settings and discuss/analyze the results.

- Please report the PSNR/SSIM/LPIPS on the public testing set.
- Also report the number of 3D gaussians.
- You also need to explain the meaning of these metrics.
- Different settings such as learning rate and densification interval, etc.
  - Setting 1 的設計如下：

        - --densify_from_iter 1000、--densify_grad_threshold 0.0001、--densification_interval 900、--opacity_reset_interval 9999999
    - Setting 2 的設計如下：
        - --densify_from_iter 1000、--densify_grad_threshold 0.0001、--densification_interval 900
        - 和 Setting 1 相比，沒有添加 opacity_reset_interval 參數
    - Setting 3 的設計如下：
        - --densify_from_iter 1000、--densify_grad_threshold 0.0001、--opacity_reset_interval 9999999
        - 和 Setting 1 相比，沒有添加 densification_interval 參數
    - Setting 4 的設計如下：
        - --densify_from_iter 200、--densify_grad_threshold 0.0001、--densification_interval 900、--opacity_reset_interval 9999999
        - 和 Setting 1 相比，調整了 densify_from_iter 參數
- 各 Setting 的 performance 如下：

Setting	PSNR	SSIM	LPIPS	the number of 3D gaussians
1	38.07125234603882	0.9802033295129479	0.07274182423949242	378,168
2	30.75358943939209	0.9413099271964559	0.17096166998147966	241,905
3	35.05728702545166	0.9698841752490506	0.10133297212421893	1,255,776
4	37.96283082962036	0.980254581120258	0.07291957966983319	396,925

- 分析與討論不同 setting 之間的 performance：
    - Setting 1 是 performance 表現最好的設計
        - 合理的 densification 起始時間 (1000) 允許模型先穩定學習
        - 適中的 Gaussians 數量顯示良好的空間利用效率
        - 較長的 opacity reset interval 避免了不必要的重置
    - Setting 2 是 performance 表現最差的設計
        - 可能是 opacity_reset_interval 維持預設的 3000，重置透明度可能會導致訓練不穩定，模型還沒有充分學習就被重置，打斷高斯點分布，無法形成穩定的場景表示，在重置時過多地刪除了有用的點
        - Gaussians 數量明顯較少，可能導致細節不足
    - Setting 3
        - 和 Setting 1 相比，densification_interval 參數維持預設的 100， 導致 Gaussians 的數量最高
        - 性能適中但不是最佳，表示 Gaussians 數量過多可能影響計算效率降低
    - Setting 4

    - 和 Setting 1 相比，修改了 densify_from_iter 為 200，最後結果的性能與 Setting 相近，表現較好
    - 代表模型對初始 densification 時間有一定的容忍度，200~1000 這個區間都是可以接受的
- 三種評估指標在 3D Gaussian Splatting 論文中的使用和意義：

1. PSNR (Peak Signal-to-Noise Ratio)

- 是一種基本和使用最廣泛的圖像質量評估指標，主要衡量重建圖像與原始圖像在像素層面的差異
- 計算方法：測量重建圖像與 ground truth 之間的均方誤差 (MSE)，然後轉換為對數
- 公式為：
    - $10 * log_{10}(\frac{MAX_I^2}{MSE})$ = $20 * log_{10}(\frac{MAX_I}{\sqrt{MSE}})$
- 公式中：
    - $MSE = (\frac{1}{(m*n)}) * \Sigma[I(i,j) - K(i,j)]^2$
    - MAX_I = 影像可能的最大像素值 (對於 8-bit 影像為 255)
    - m, n = 影像的寬度和高度
    - I = ground truth影像
    - K = 重建的影像

2. SSIM (Structural Similarity Index)

- 是一種接近人類視覺系統的評估方式，考慮圖像的結構信息而不只是逐像素比較
- 計算方法：比較圖像的亮度、對比度和結構相似性
- 公式為：
    - $SSIM(x, y)$ = $[l(x, y)]^\alpha * [c(x, y)]^\beta * [s(x, y)]^\gamma$
- 公式中：
    - $l(x, y)$ = $\frac{(2\mu x\mu y + c1)}{(\mu x^2 + \mu y^2 + c1)}$，亮度比較
    - $c(x, y)$ = $\frac{(2\sigma x\sigma y + c2)}{(\sigma x^2 + \sigma y^2 + c2)}$，對比度比較
    - $s(x, y)$ = $\frac{(\sigma xy + c3)}{(\sigma x\sigma y + c3)}$，結構比較
    - $\mu x, \mu y$ = x 與 y 的平均值
    - $\sigma x, \sigma y$ = x 與 y 的標準差
    - σxy = x 與 y 的協方差
    - $c1, c2, c3$ = 穩定常數
    - α, β, γ = 權重參數（通常設為1）

3. LPIPS (Learned Perceptual Image Patch Similarity)

- 是一種基於深度學習的現代評估指標，通過預訓練的神經網絡來模擬人類視覺系統
- 計算方法：使用預訓練的神經網絡來評估圖像 patch 之間的感知差異

- 公式為：
  - $d(x, x_0) = \sum_l (\frac{1}{H_l W_l}) * \sum_{h,w} ||w_l \odot (\hat{y}^l_{hw} - \hat{y}^l_{0hw})||_2^2$
- 公式中：
  - x, x_0 = 要比較的兩張圖片
  - l = CNN網絡的層索引
  - H_l, W_l = 第 l 層特徵圖的高度和寬度
  - w_l = 學習的每層權重
  - $\hat{y}^l_{hw}$ = 圖片 x 在第 l 層位置 (h, w) 的單位正規化特徵
  - $\hat{y}^l_{0hw}$ = 圖片 x_0 在第 l 層位置 (h, w) 的單位正規化特徵
  - $\odot$ = Hadamard 乘積 (element-wise multiplication)
  - $||_2^2$ = L2 範數

4. Instead of initializing from SFM points [dataset/sparse/points3D.ply], please try to train your 3D gaussians with random initializing points.
   - Describe how you initialize 3D gaussians
   - Compare the performance with that in previous question.

- 為了改為 train 3D gaussians with random initializing points，我修改了以下設計：
  - 創建 100,000 個隨機 3D 點，生成範圍在 [-1.3, 1.3] 間的隨機數，且為每個點隨機生成標準化到 [0, 1] 範圍間的顏色值，塑造一個合理的場景空間範圍，此時我形狀為 (100000, 3)，代表 100,000 個點，每個點有 x,y,z 三個坐標
  - 透過 pcd = BasicPointCloud，將隨機生成的位置、顏色和法向量組合成一個點雲對象，這個對象將作為 3D Gaussian Splatting 的初始狀態
  - 最後將隨機生成的點雲狀態保存為 .ply 檔案，且此時 * 255 將顏色值轉回 [0,255] 範圍
  - 透過 scene_info = SceneInfo 將點雲信息和相機參數打包成一個場景對象

```
    # 隨機初始化點雲並存儲為臨時 .ply 文件
    print("Initializing random 3D Gaussians...")
    num_pts = 100_000  # 設定點雲的數量，可根據需要調整
    xyz = np.random.random((num_pts, 3)) * 2.6 - 1.3  # 在 [-1.3, 1.3] 範圍內生成隨機點
    shs = np.random.random((num_pts, 3)) / 255.0  # 隨機生成 SH 值作為顏色
    pcd = BasicPointCloud(points=xyz, colors=SH2RGB(shs), normals=np.zeros((num_pts, 3)))

    # 將隨機點雲存為臨時 .ply 文件
    temp_ply_path = os.path.join(path, "sparse/0/temp_points3D.ply")
    storePly(temp_ply_path, xyz, SH2RGB(shs) * 255)

    scene_info = SceneInfo(
        point_cloud=pcd,
        train_cameras=train_cam_infos,
        test_cameras=test_cam_infos,
        nerf_normalization=nerf_normalization,
        ply_path=temp_ply_path,  # 指向臨時生成的 .ply 文件
        is_nerf_synthetic=False
    )
    return scene_info
```

- 我的 training setting 皆如下：
  - --densify_from_iter 1000、--densification_interval 900、--densify_grad_threshold 0.0001、--opacity_reset_interval 9999999
- 以下為我原本使用 SFM points 後的 performance：

PSNR	SSIM	LPIPS	the number of 3D gaussians
38.07125234603882	0.9802033295129479	0.07274182423949242	378,168

- 以下為我使用了 random initializing points 後的 performance：

PSNR	SSIM	LPIPS	the number of 3D gaussians
19.12304139137268	0.6497071554399442	0.5703224635124207	362,144

- 使用 SFM points 和 random initializing points 的 performance 比較：
  - 可以明顯地觀察到所有評估方式上， SFM points 的 performance 都比 random initializing points 要好，會有這樣的性能差異，我認為有以下幾種原因：

1. 初始點分布的質量
   - SFM points 是通過 Structure from Motion 產生的，這些點已經通過多視角重建和特徵匹配得到，代表了場景的實際幾何結構
   - Random points 完全隨機分布，沒有任何場景先驗信息，需要從零開始學習正確的位置和外觀
2. 優化難度
   - SFM 初始化的優化從一個"接近正確"的位置開始，主要是細化和改進
   - Random points 的優化需要從完全隨機的狀態開始，要找到正確的幾何結構，這是一個更困難的優化問題
3. 收斂特性
   - SFM 由於起點更好，更容易收斂到好的局部最優解
   - Random points 容易陷入次優的局部最優解，因為優化空間太大且起點較差
4. 空間覆蓋
   - SFM points 自然地集中在場景中有視覺特徵的區域
   - Random points 均勻分布在空間中，包括可能完全沒有實際物體的區域

Reference

3D Gaussian Splatting for Real-Time Radiance Field Rendering Github

3D Gaussian Splatting for Real-Time Radiance Field Rendering

3D 重建技術 (三): 3D Gaussian Splatting，讓 3D 重建從學術走向實際應用的革新技術！用彩色橢圓球球表示場景！

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis

A Survey on 3D Gaussian Splatting

gaussian-splatting Module install error #2 issue

gaussian-splatting Pip Downloads failed #257 issue

Claude