R12521504 吳玉文

1. Please explain

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

3D Gaussian Splatting 的組成是由 3D Gaussian 加上 Splatting，「3D Gaussian」指的是用 3D Gaussian Distribution 作為 3D 物體的顯示表示，「Splatting」則是一種光柵化（Rasterization）的渲染技術，就是把三維空間中的物體投影到相機鏡頭上，可以解釋為將物體由近而遠依序砸向鏡頭，就像砸雪球（Splatting）一樣，看看砸完之後 2D 平面上像素的覆蓋情形。

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

	3D Gaussian Splatting	NeRF
運算速度	較快	較慢
訓練時間	較短	較長
3D 視覺化過程	簡單	複雜
動態（4D）場景	順暢	卡頓
真實感（陰影、反射等）	較低	較高

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

我覺得最重要的部分是透過 Rasterization 進行 Render 的部分。因為相比於 Ray Tracing，Rasterization 更適合即時渲染應用，特別是在需要快速生成視覺結果的情境中。此外，它充分利用了 GPU 硬體的特性，降低了計算成本，同時保持了視覺品質。在性能與品質之間取得了良好的平衡，是整個技術中的關鍵要素。

Reference: 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！

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

我採用的是作業說明中提供的 3D Gaussian Splatting GitHub links 的作法，使用其中的 train.py 進行 model 的訓練。Input 為 images、images 的參數、camera 的參數、作為 initial 的 SFM points，將 Input 放入模型後，訓練的過程為 1. 隨機選擇攝影機視角 -> 2.進行場景渲染 -> 3. 計算 loss -> 4. 反向傳播更新參數。最後 Output 出訓練完成的 3D Gaussians (point_cloud.ply)。並透過修改 Scene 的__init__.py、camereas.py 跟 camera_utils.py 等等，讓程式碼在僅讀取 images 參數及 camera 參數的情況下能夠對需要的攝影機視角進行場景渲染 (使用修改過後的 render.py)。

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.*

- PSNR：度量兩張圖像在像素層面的差異，基於信噪比 (Signal-to-Noise Ratio)，通過均方誤差(MSE)計算兩張圖像的像素差異。優點是簡單直觀，計算高效，且對像素級差異敏感。缺點是無法考慮人類視覺感知，對結構性失真不敏感。

- SSIM：從結構、亮度和對比度三方面評估圖像相似性，模擬人類視覺對結構訊息的敏感性，透過局部計算圖像塊之間的結構相似性。優點是對結構訊息變化非常敏感、結果更符合人類視覺系統的結構偏好。缺點是有時對光照或對比度的變化過於敏感、計算上較 PSNR 複雜。

- LPIPS：基於深度學習模型，評估圖像之間的感知相似性，模擬人類視覺系統的主觀判斷。透過使用預訓練的深度神經網絡 (如 VGG) 提取特徵。比較輸入和參考圖像在特徵空間中的差異。優點是更接近人眼主觀感受，對模糊、顏色偏差等現象敏感。缺點是計算較慢，需要大量計算資源。

Setting	PSNR	SSIM	LPIPS	Num of 3D gaussians
--iteration 120000 --densification_interval 100 --scaling_lr 0.005 --lambda_dssim 0.4	35.50	0.970	0.104	589248
--iteration 120000 --densification_interval 100 --scaling_lr 0.005 --lambda_dssim 0.6	35.07	0.971	0.098	767466
--iteration 120000 --densification_interval 30 --scaling_lr 0.005 --lambda_dssim 0.6	27.68	0.843	0.257	2323465
--iteration 120000 --densification_interval 100 --scaling_lr 0.001 --lambda_dssim 0.5	33.84	0.972	0.085	798438
--iteration 240000 --densification_interval 100 --scaling_lr 0.001 --lambda_dssim 0.4	35.30	0.976	0.079	664204

從結果來看，densification_interval 設太小雖然在 train 上表現的很好，但在 public_test 上顯示出了 overfitting 的情形。而將 lambda_dssim 調高一點會讓 SSIM 分數較好，但略微犧牲了一點 PSNR 的分數。scaling_lr 也同樣是提高 SSIM 但犧牲了更多 PSNR。整體看下來 3D gaussians 的數量如果越多，在 training set 的 PSNR 都接近的情況下，越有可能出現 overfitting 的情形。

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

   參考 dataset 中 ply 的點的數量進行座標、法向量、顏色的 random，如下列程式碼所示：

```python
import numpy as np
import struct

num_points = 13414
points = np.random.uniform(-1, 1, (num_points, 3))  # 隨機生成 (x, y, z)
normals = np.random.uniform(-1, 1, (num_points, 3))  # 隨機生成法向量 (nx, ny, nz)
colors = np.random.randint(0, 256, (num_points, 3), dtype=np.uint8)  # 隨機生成顏色 (red, green, blue)

with open("random_points3D.ply", "wb") as f:
    f.write(b"ply\n")
    f.write(b"format binary_little_endian 1.0\n")
    f.write(f"element vertex {num_points}\n".encode())
    f.write(b"property float x\n")
    f.write(b"property float y\n")
    f.write(b"property float z\n")
    f.write(b"property float nx\n")
    f.write(b"property float ny\n")
    f.write(b"property float nz\n")
    f.write(b"property uchar red\n")
    f.write(b"property uchar green\n")
    f.write(b"property uchar blue\n")
    f.write(b"end_header\n")

    # 寫入二進制數據
    for i in range(num_points):
        # x, y, z, nx, ny, nz -> float32
        f.write(struct.pack("<6f", *points[i], *normals[i]))
        # r, g, b -> uint8
        f.write(struct.pack("<3B", *colors[i]))
```

設定是以 --iteration 120000 --densification_interval 100 --scaling_lr 0.005 --lambda_dssim 0.4 進行 training。訓練上耗費了久上許多的時間（原本只要一小時左右，random initial 要四個小時），而從結果來看，3D Gaussians 最終有 3662234 個，比起用 SFM points 作為 initial 的還要多上許多。雖然在 train 上的 PSNR 有 37.50，但在 public_test 上的 PSNR 分數為 16.35；SSIM 為 0.551；LPIPS 為 0.571，結果超差，顯示出明顯的 overfitting，因此還是要用 SFM points 作為 initial 會比較好。就算是 render 的比較好的情況下顏色也很失真，如下圖所呈現的樣子。