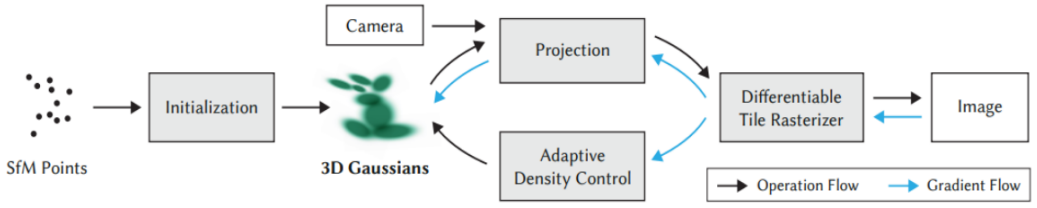
DLCV Hw-4

Name: 李維釗

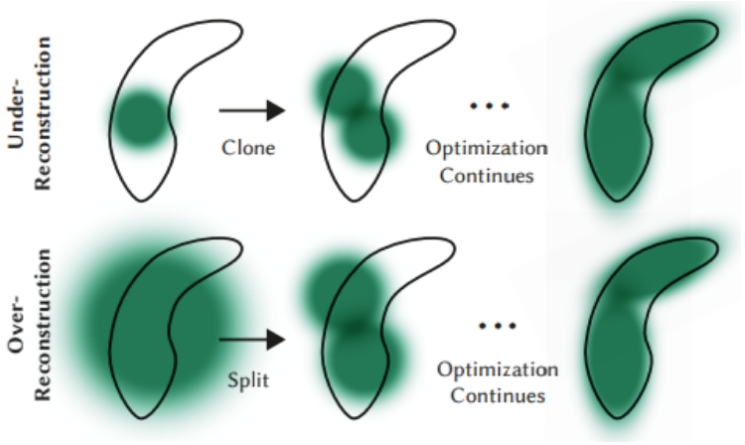
Student ID: R12942089

3D Novel View Synthesis

1. Please explain
    a. Try to explain 3D Gaussian Splatting in your own words
        - 在3DGS的論文中，主要重點在於3D Gaussian的表現以及Splatting到2D圖像的過程。



1. SfM Points Initialization
   透過SFM points的資料來對3D Gaussians進行初始化，算出各個高斯橢球的位置、形狀、顏色、透明度等參數。
2. Projection（3D → 2D）
   將每一顆3D Gaussians，根據它們與相機位置的距離（深度），由近到遠排序進行投影。透過將中心點座標去跟變換矩陣、協方差矩陣等數據進行運算，就能將三維空間中的高斯分布轉換到二維的平面上。
3. Differentiable Tile Rasterizer
   透過可微光柵化渲染，得到2D圖像。
4. Adaptive Density Control
   根據 Differentiable Tile Rasterizer 所得到的2D渲染圖像與GT之間的差異，計算出loss，並將loss沿藍色箭頭方向BP。藍色箭頭向上即更新3D Gaussians的參數，向下則送入Adaptive Density Control來更新3D Gaussians的密度（Clone, Split）以更好地擬合照片中的細節。



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

- 3D Gaussian Splatting
  - pros
    - 渲染速度極快
    - 高精度重建
    - 參數效率高
  - cons
    - 對輸入數據依賴性強
    - 訓練過程較複雜，需要精細的調參
- NeRF
  - pros
    - 通用性強
    - 插值能力好，對場景的幾何和外觀插值效果好
    - 理論基礎成熟
  - cons
    - 渲染速度慢
    - 計算資源要求高
    - 細節重建能力有限

	Gaussian Splatting	NeRF
Representation	Explicit (3D Gaussians)	Implicit (MLP)
Editability	O	X
Memory Consumption	High	Low
Training Time	Fast	Slow
Rendering Time	Fast	Slow

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

- 3D Gaussians的表示和渲染方法。
- 使用3D Gaussians來表示 3D 空間中的場景，而非point cloud或voxel。這是一種介於point cloud與voxel渲染之間的表示形式：
  - 相較於point cloud，高斯分布具有連續性，能有效表達局部空間的不確定性。
  - 相較於voxel，高斯的稀疏性降低了記憶體需求，同時保留了渲染的靈活性。
- 3D Gaussians提供了更多的參數化自由度（位置、大小、方向、顏色、密度），這使得它能更準確地描述場景細節並適應多視角優化。而3DGS 的渲染方法基於高斯分布的 2D 投影，可以高效地將 3D 信息轉換為 2D 圖像。不需要進行昂貴的voxel積分，顯著提升了渲染速度。

2. Describe the implementation details of your 3D Gaussian Spaltting for the given dataset. You need to explain your ideas completely.
   a. 按照助教提供的3DGS repo進行實做
   b. 只有inference在private dataset時，需要跳過SFM檔案不進行讀取（因為不提供）。
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.

	PSNR	**SSIM**	**LPIPS (vgg)**	**Number of 3D gaussians**
Setting 1	32.7722778	0.9619721	0.0843630	375858
Setting 2	35.6962318	0.9732233	0.0480763	430660
Setting 3	35.8623543	0.9746669	0.0477016	461194

- Setting 1 (all default)

```
CUDA_VISIBLE_DEVICES=1 python train.py -s ../dataset/train -m
--iterations 30000 \
--test_iterations 7000 30000 \
--save_iterations 7000 30000 \
--feature_lr 0.0025 \
--opacity_lr 0.05 \
--scaling_lr 0.005 \
--rotation_lr 0.001 \
--position_lr_init 0.00016 \
```

```
--position_lr_final 0.0000016 \
--densification_interval 100
```

- Setting 2

```
CUDA_VISIBLE_DEVICES=1 python train.py -s ../dataset/train -m
--iterations 100000 \
--feature_lr 0.0025 \
--opacity_lr 0.05 \
--scaling_lr 0.002 \
--rotation_lr 0.001 \
--position_lr_init 0.000015 \
--position_lr_final 0.0000016 \
--densification_interval 100
```

- Setting 3

```
CUDA_VISIBLE_DEVICES=1 python train.py -s ../dataset/train -m
--iterations 90000 \
--feature_lr 0.0025 \
--opacity_lr 0.05 \
--scaling_lr 0.002 \
--rotation_lr 0.001 \
--position_lr_init 0.000015 \
--position_lr_final 0.0000016 \
--densification_interval 80
```

- You also need to explain the meaning of these metrics.
  1. **PSNR**
     - 是衡量影像品質的傳統數字指標，用於評估兩張圖像之間的差異程度，主要基於像素值的差異。它計算的是峰值訊號與雜訊之間的比值，單位是分貝（dB）。
     - 值越大表示品質越好，因為誤差（雜訊）越小。
     - 不能很好地反映人類視覺的真實感受。
  2. **SSIM**
     - 專注於結構化相似度的指標，設計用來更好地模擬人眼感知的圖像品質。它不僅僅考慮像素值的差異，還分析了亮度、對比度和結構信息。
     - 能更準確地捕捉到影像結構的變化，比 PSNR 更符合人類視覺感知。
  3. **LPIPS (vgg)**
     - 基於深度學習的感知相似度指標，旨在模擬人類感知的圖像差異。它使用訓練好的卷積神經網路（如 VGG）提取圖像特徵，並比較兩張圖像的特徵差異。

        - 值越低表示圖像越相似，因為差異（距離）越小。
        - 基於深度特徵（VGG），LPIPS 能更好地捕捉人類感知層面的相似度，比 SSIM 和 PSNR 更準確。
    - Different settings such as learning rate and densification interval, etc.
        - 在 Setting1 中，使用了 3DGS repo 預設的參數，並且訓練的 iteration step 並不多，只有30000。
        - Setting2 則是根據 repo 內部說明，若是在多尺度細節場景（極端特寫，與遠距離鏡頭混合）會造成生成結果不佳，因此可以降低`--position_lr_init`、`--position_lr_final`和`--scaling_lr`。我在 Setting2 中只降低了`scaling_lr`與`position_lr_init`，而結果分數也有所優化。
        - Setting3 則是固定與 Setting2 一樣的 lr，降低`densification_interval`，而結果又更加的優化。這邊可以與 Setting2 比較得知：在需要高精度的 3D 建模中，應選擇較小的間隔；在快速預覽或大規模場景中，較大的間隔更實用。
        - 另外可以發現在三個設置中，隨著Setting 2, 3的參數優化，得到較好的分數的同時，3D gaussians的數量也有增長。可以理解為在optimize的過程中gaussians被分裂的更加細緻，以達到更好的生成影像。
4. 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
        - 在`scene class`中，利用 SFM point 初始化 gaussian 的部分是來自`self.gaussians.create_from_pcd`這個function。因此在訓練時需要從這個部分下手來處理初始化的資料。
            1. 先查看SFM points的格式

```
element vertex 13414
property float x
property float y
property float z
property float nx
property float ny
property float nz
property uchar red
property uchar green
property uchar blue
```

            2. 而在create_from_pcd這個function中，需要被隨機設定的為`fused_point_cloud`以及`fused_color`，兩個變數。

```
def create_from_pcd(self, pcd : BasicPointCloud, cam_infos : int, spatial_lr_scale : float):
    self.spatial_lr_scale = spatial_lr_scale
    fused_point_cloud = torch.tensor(np.asarray(pcd.points)).float().cuda()
    fused_color = RGB2SH(torch.tensor(np.asarray(pcd.colors)).float().cuda())
```

3. 而這兩個變數我分別使用以下方法來進行隨機初始化（固定與Setting 1~3一樣的數量=13414）。
    a. fused_point_cloud: torch.normal(mean=8.648315, std=16.716545, size=(13414, 3)
    b. fused_color: torch.rand(13414,3)

b. Compare the performance with that in previous question.

- Performance table

	PSNR	**SSIM**	**LPIPS (vgg)**	**Number of 3D gaussians**
Random init	31.4954128	0.9450120	0.1082649	501363

- 與有使用SFM points進行初始化的設定相比較，可以發現初始化的方式對於3DGS具有一定的影響。在將初始化改為隨機的設定後，各項**分數都表現的較差**，且**需要更多的3D gaussians**。