1. (15%) Please explain:

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

Ans: 3D Gaussian Splatting is a technique used to render 3D scenes or point clouds by replacing points with Gaussian functions. This approach provides smoother, more continuous visual representations.

In 3D GS, the scene is represented as a collection of smooth 3D Gaussian "splats," that are scattered throughout the space. Each splat represents a point in a point cloud, with its size and shape controlled by a Gaussian function. These splats are blended to create a smooth, continuous representation of the 3D scene, reducing sharp edges and visual artifacts.

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

Ans:

	3D GS	NeRF
Time Complexity	Lower	Higher
Space complexity	Higher	Lower
Editability	Higher	Lower
Photorealistic	High	High
Parallelizable workflow	Higher	Lower
Representation type	Explicit	Implicit

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

Ans: the most important idea of 3D GS is that it projects a learnable(optimizable) 3D gaussian into an image space because this idea not only allows parallel computation, but also preserves high resolution.

2. (15%) Describe the implementation details of ***your 3D Gaussian Spaltting*** 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 pretrained weights.**
- **We provide the following Github links for your reference.**
- **3D Gaussian Splatting** - **LINK** (recommended)

Implementation:

For this homework, I implemented 3D GS with the given github repository, and I directly modified the given repository in order to meet the requirements of this homework.

1. Training: according to README.md in the official repo, they provide several hyperparameter for us to train and render with the given dataset. I have tested the original setting (all hyperparameters are set default), but it did not reach the baseline.
    i. To enhance the training speed, I first try to adjust the Spherical harmonics feature learning rate from default 0.00025 to 0.0005
    ii. To enhance the overall performance, I set the number of iterations by 5 times more than default
    iii. Though some testing, I found that the SSIM is a little lower than expected, so I increased the lambda_dssim 2 times higher than the default.
    iv. Finally, to test on the effect of number of gaussian points, I adjust the densification interval to 80, which
2. Rendering:
    i. Since there are some inconsistencies between the homework requirements and the official evaluation method (which directly split the dataset into train-test set automatically), I did some modifications on the input and output paths.
    ii. Since the format of the private dataset do include any image (unlike official example and public test set), I simply hardcode the size of the output instead of using the input image as an example to avoid error.

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 hyperparameter settings and discuss/analyze the results.

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

Metrics explanation:

- PSNR (peak-signal-to-noise ratio): to measure the quality of reconstructed images
- SSIM (structural similarity index measure): to measure the similarity between two images.
- LPIPS (learned perceptual image patch similarity): to measure the similarity between features of two images extracted from a pretrained network

Experiment result:

Setting	PSNR	SSIM	LPIPS (vgg)	Number of 3D gaussians
Iteration: 90000 feature_lr: 0.005 densification_interval: 100 With white background Resolution:1 (original size)	35.212	0.9697	0.1096	350282
Iteration: 150000 **feature_lr: 0.005** **densification_interval: 100** **With white background** **lambda_dssim:0.4** **Resolution:1 (original size)**	**36.150**	**0.9731**	**0.0975**	527315
Iteration: 150000 feature_lr: 0.005 densification_interval: 80	35.961	0.9719	0.0987	710356

With white background lambda_dssim:0.4 Resolution:1 (original size)				

Discussion:

According to the table above, the larger number of iterations has better result on the three metrices, and since the SSIM score is almost cross the baseline, I tried to adjust the lambda_dssmi parameter higher to enhance the influence of SSIM on total loss, and it do increase a little bit. I also testing on the densification interval, it shows that a more frequent densification would lower the performance.

4. (15%) 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 the previous question

Initialization method:

I initialize the 3D gaussian by modifying ***fetchPly()*** function used for reading .ply file (gaussian-splatting/scene/dataset_readers.py). Here, I simply assigned a number for the length of the gaussian splats(number of points, equal to the given SFM file), and assign the values using np.random .

```
def fetchPly(path):
    try:
        plydata = PlyData.read(path)
        vertices = plydata['vertex']
        positions = np.vstack([vertices['x'], vertices['y'], vertices['z']]).T
        colors = np.vstack([vertices['red'], vertices['green'], vertices['blue']]).T / 255.0
        normals = np.vstack([vertices['nx'], vertices['ny'], vertices['nz']]).T
        print("len(vertices):", len(vertices))
        print("len(positions):", len(positions))
        return BasicPointCloud(points=positions, colors=colors, normals=normals)
    except:
        num=13414
        rng = np.random.default_rng()
        positions = np.vstack([rng.standard_normal(num), rng.standard_normal(num), rng.standard_normal(num)]).T
        colors = np.vstack([rng.integers(0,255,num), rng.integers(0,255,num), rng.integers(0,255,num)]).T / 255.0
        normals = np.vstack([rng.standard_normal(num),rng.standard_normal(num), rng.standard_normal(num)]).T
        return BasicPointCloud(points=positions, colors=colors, normals=normals)
```

Comparison:

Compare with using given SFM file(same number of iterations and other hyperparameters), the training time become longer (roughly 2~3 times longer), and the performance is significantly lower than using a SFM (Testing psnr 15.664870691299438 (avg), Testing ssim 0.5173662388428268 (avg), Testing lpips (vgg) 0.6025197720527649 (avg)).

The result number of gaussian point with random one is 2221970, compared to 350282 resulted from using given SFM.

The result implies that 3D gaussian splatting highly relying on a good initialization to generate good image and speed up training process.

The images shown below demonstrate the quality of the output respectively.

Figure 1 With random initialization

Figure 2 With given SFM initialization