影像處理HW3

1. 選擇論文介紹

a. Introduction

Neural volumetric representation such as neural radiance fields(NERF) have emerged as a compelling strategy for learning to represent 3D objects and scenes from image for the purpose of rendering photorealistic novel views.Although NeRF and its variants have demonstrated impressive results across a range of view synthesis tasks, NeRF's rendering model is flawed in a manner that can cause excessive blurring and aliasing. ***When the training images observe scene content at multiple resolutions, renderings from the recovered NeRF appear excessively blurred in close-up views and contain aliasing artifacts in distant views.***

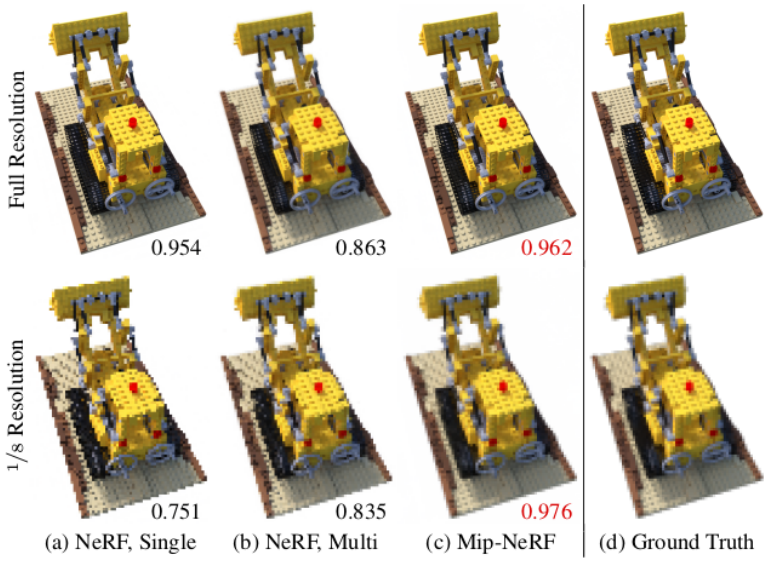

Figure1. NeRF vs MipNeRF

A straightforward solution is to adopt the strategy used in offline raytracing: supersampling each pixel by marching multiple rays through its footprint. But this is prohibitively expensive for neural volumetric representations such as NeRF, which require hundreds of MLP evaluations to render a single ray and several hours to reconstruct a single scene. So they present another effectively solution ***"Mipnerf".***

b. Method

To solve the problem that NeRF's point-sampling makes it vulnerable to issues related to sampling and aliasing. Instead of performing point-sampling along each ray, MipNeRF divides the cone being cast into a series of conical frustums . And instead of constructing positional encoding (PE) features from an infinitesimally small point in space, MipNeRF constructs an integrated positional encoding (IPE) representation of the volume covered by each conical frustum. ***These***

changes allow the MLP to reason about the size and shape of each conical frustum, instead of just its centroid. Below, we will focus on the following two points Cone Tracing and Positional Encoding.

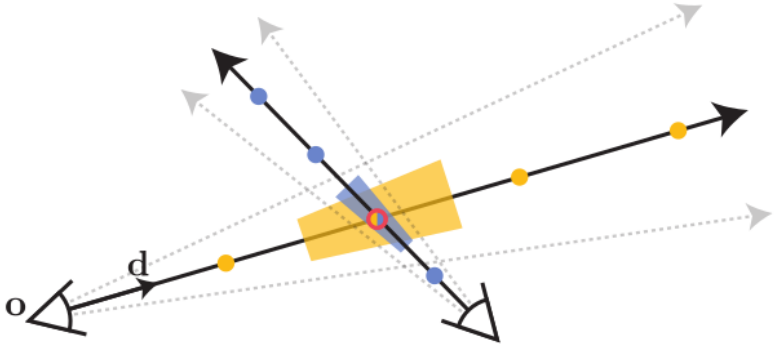

Figure 2. MipNeRF solution

Cone Tracing and Positional Encoding

For that pixel, we cast a cone from the camera's center of projection o along the direction d that passes through the pixel's center. The apex of that cone lies at o, and the radius of the cone at the image plane o + d is parameterized as ṙ. We set ṙ to the width of the pixel in world coordinates scaled by 2/ sqrt(12), which yields a cone whose section on the image plane has a variance in x and y that matches the variance of the pixel's footprint. The set of positions x that lie within a conical frustum between two t values [t0 , t1] is

$$\mathrm{F}(\mathbf{x},\mathbf{o},\mathbf{d},\dot{r},t_0,t_1)=\mathbb{1}\left\{\left(t_0<\frac{\mathbf{d}^{\mathrm{T}}(\mathbf{x}-\mathbf{o})}{\|\mathbf{d}\|_2^2}<t_1\right)\right.$$
$$\left.\wedge\left(\frac{\mathbf{d}^{\mathrm{T}}(\mathbf{x}-\mathbf{o})}{\|\mathbf{d}\|_2\|\mathbf{x}-\mathbf{o}\|_2}>\frac{1}{\sqrt{1+(\dot{r}/\|\mathbf{d}\|_2)^2}}\right)\right\}, \quad (5)$$

The featurized representation of the volume inside this conical frustum is

$$\gamma^*(\mathbf{o},\mathbf{d},\dot{r},t_0,t_1)=\frac{\int\gamma(\mathbf{x})\,\mathrm{F}(\mathbf{x},\mathbf{o},\mathbf{d},\dot{r},t_0,t_1)\,d\mathbf{x}}{\int\mathrm{F}(\mathbf{x},\mathbf{o},\mathbf{d},\dot{r},t_0,t_1)\,d\mathbf{x}}.$$

Thus, we approximate the conical frustum with a multivariate Gaussian which allows for an efficient approximation to the desired feature, which we will call an "integrated positional encoding" (IPE). To approximate a conical frustum with a multivariate Gaussian, we must compute the mean and covariance of F(x, ·). Because each conical frustum is assumed to be circular, and because conical frustums are symmetric around the axis of the cone, such a Gaussian

is fully characterized by three values (in addition to o and d): the mean distance along the ray μt , the variance along the ray σt2 , and the variance perpendicular to the ray σr2 :

$$\mu_t = t_\mu + \frac{2t_\mu t_\delta^2}{3t_\mu^2 + t_\delta^2}, \qquad \sigma_t^2 = \frac{t_\delta^2}{3} - \frac{4t_\delta^4(12t_\mu^2 - t_\delta^2)}{15(3t_\mu^2 + t_\delta^2)^2}$$
$$\sigma_r^2 = \dot{r}^2\left(\frac{t_\mu^2}{4} + \frac{5t_\delta^2}{12} - \frac{4t_\delta^4}{15(3t_\mu^2 + t_\delta^2)}\right).$$

These quantities are parameterized with respect to a mid-point tμ = (t0 + t1)/2 and a half-width tδ = (t1 − t0)/2, which is critical for numerical stability. Please refer to the supplement for a detailed derivation. We can transform this Gaussian from the coordinate frame of the conical frustum into world coordinates as follows:

$$\boldsymbol{\mu} = \mathbf{o} + \mu_t \mathbf{d}, \qquad \boldsymbol{\Sigma} = \sigma_t^2(\mathbf{d}\mathbf{d}^{\mathrm{T}}) + \sigma_r^2\left(\mathbf{I} - \frac{\mathbf{d}\mathbf{d}^{\mathrm{T}}}{\|\mathbf{d}\|_2^2}\right)$$

giving us our final multivariate Gaussian.
Next, we derive the IPE, which is the expectation of a positionally-encoded coordinate distributed according to the aforementioned Gaussian. To accomplish this, it is helpful to first rewrite the PE as a Fourier feature:

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 2 & 0 & 0 & & 2^{L-1} & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 0 & \cdots & 0 & 2^{L-1} & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & & 0 & 0 & 2^{L-1} \end{bmatrix}^{\mathrm{T}}, \quad \gamma(\mathbf{x}) = \begin{bmatrix} \sin(\mathbf{P}\mathbf{x}) \\ \cos(\mathbf{P}\mathbf{x}) \end{bmatrix}$$

This reparameterization allows us to derive a closed form for IPE. Using the fact that the covariance of a linear transformation of a variable is a linear transformation of the variable's covariance (Cov[Ax, By] = A Cov[x, y]BT) we can identify the mean and covariance of our conical frustum Gaussian after it has been lifted into the PE basis P:

$$\boldsymbol{\mu}_\gamma = \mathbf{P}\boldsymbol{\mu}, \qquad \boldsymbol{\Sigma}_\gamma = \mathbf{P}\boldsymbol{\Sigma}\mathbf{P}^{\mathrm{T}}.$$

The final step in producing an IPE feature is computing the expectation over this lifted multivariate Gaussian, modulated by the sine and the cosine of position. These expectations have simple closed-form expressions:

$$\mathrm{E}_{x\sim\mathcal{N}(\mu,\sigma^2)}[\sin(x)] = \sin(\mu)\exp\left(-(1/2)\sigma^2\right),$$
$$\mathrm{E}_{x\sim\mathcal{N}(\mu,\sigma^2)}[\cos(x)] = \cos(\mu)\exp\left(-(1/2)\sigma^2\right).$$

See that this expected sine or cosine is simply the sine or

cosine of the mean attenuated by a Gaussian function of the variance. With this we can compute our final IPE feature as the expected sines and cosines of the mean and the diagonal of the covariance matrix:

$$\begin{aligned}\gamma(\boldsymbol{\mu}, \boldsymbol{\Sigma}) &= \mathrm{E}_{\mathbf{x}\sim\mathcal{N}(\boldsymbol{\mu}_\gamma, \boldsymbol{\Sigma}_\gamma)}[\gamma(\mathbf{x})] \\ &= \begin{bmatrix}\sin(\boldsymbol{\mu}_\gamma) \circ \exp(-(1/2)\operatorname{diag}(\boldsymbol{\Sigma}_\gamma)) \\ \cos(\boldsymbol{\mu}_\gamma) \circ \exp(-(1/2)\operatorname{diag}(\boldsymbol{\Sigma}_\gamma))\end{bmatrix},\end{aligned}$$

where ◦ denotes element-wise multiplication. Because positional encoding encodes each dimension independently, this expected encoding relies on only the marginal distribution of γ(x), and only the diagonal of the covariance matrix (a vector of per-dimension variances) is needed. Because Σγ is prohibitively expensive to compute due its relatively large size, we directly compute the diagonal of Σγ :

$$\operatorname{diag}(\boldsymbol{\Sigma}_\gamma) = \left[\operatorname{diag}(\boldsymbol{\Sigma}), 4\operatorname{diag}(\boldsymbol{\Sigma}), \ldots, 4^{L-1}\operatorname{diag}(\boldsymbol{\Sigma})\right]^{\mathrm{T}}$$

This vector depends on just the diagonal of the 3D position's covariance Σ, which can be computed as:

$$\operatorname{diag}(\boldsymbol{\Sigma}) = \sigma_t^2(\mathbf{d}\circ\mathbf{d}) + \sigma_r^2\left(\mathbf{1} - \frac{\mathbf{d}\circ\mathbf{d}}{\|\mathbf{d}\|_2^2}\right).$$

If these diagonals are computed directly, IPE features are roughly as expensive as PE features to construct.

c. Experiment

MipNeRF evaluated on the Blender dataset presented in the original NeRF paper and also on a simple multiscale variant of that dataset designed to better probe accuracy on multi-resolution scenes and to highlight NeRF's critical vulnerability on such tasks. Use three kinds of error metrics:PSNR, SSIM, and LPIPS.

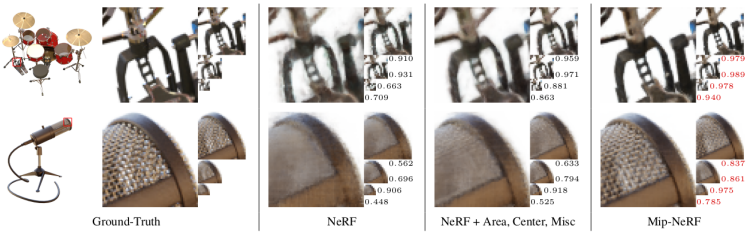

Figure 3. Visualization of Dataset

	PSNR ↑				SSIM ↑				LPIPS ↓						
	Full Res.	1/2 Res.	1/4 Res.	1/8 Res.	Full Res.	1/2 Res.	1/4 Res.	1/8 Res.	Full Res.	1/2 Res.	1/4 Res.	1/8 Res	Avg. ↓	Time (hours)	# Params
NeRF (Jax Impl.) [11,30]	31.196	30.647	26.252	22.533	0.9498	0.9560	0.9299	0.8709	0.0546	0.0342	0.0428	0.0750	0.0288	3.05 ± 0.04	1,191K
NeRF + Area Loss	27.224	29.578	29.445	25.039	0.9113	0.9394	0.9524	0.9176	0.1041	0.0677	0.0406	0.0469	0.0305	3.03 ± 0.03	1,191K
NeRF + Area, Centered Pixels	29.893	32.118	33.399	29.463	0.9376	0.9590	0.9728	0.9620	0.0747	0.0405	0.0245	0.0398	0.0191	3.02 ± 0.05	1,191K
NeRF + Area, Center, Misc.	29.900	32.127	33.404	29.470	0.9378	0.9592	0.9730	0.9622	0.0743	0.0402	0.0243	0.0394	0.0190	2.94 ± 0.02	1,191K
Mip-NeRF	32.629	34.336	35.471	35.602	0.9579	0.9703	0.9786	0.9833	0.0469	0.0260	0.0168	0.0120	0.0114	2.84 ± 0.01	612K
Mip-NeRF w/o Misc.	32.610	34.333	35.497	35.638	0.9577	0.9703	0.9787	0.9834	0.0470	0.0259	0.0167	0.0120	0.0114	2.82 ± 0.03	612K
Mip-NeRF w/o Single MLP	32.401	34.131	35.462	35.967	0.9566	0.9693	0.9780	0.9834	0.0479	0.0268	0.0169	0.0116	0.0115	3.40 ± 0.01	1,191K
Mip-NeRF w/o Area Loss	33.059	34.280	33.866	30.714	0.9605	0.9704	0.9747	0.9679	0.0427	0.0256	0.0213	0.0308	0.0139	2.82 ± 0.01	612K
Mip-NeRF w/o IPE	29.876	32.160	33.679	29.647	0.9384	0.9602	0.9742	0.9633	0.0742	0.0393	0.0226	0.0378	0.0186	2.79 ± 0.01	612K

Figure 4. A quantitative comparison of mip-NeRF and its ablations against NeRF and several NeRF variants on the test set of our multiscale Blender dataset.

d. Conculsion

This paper presents a model "MipNeRF", which can address the inherent aliasing of NeRF by ***casting cones, encoding the positions and sizes of conical frustums, and training a single neural network that models the scene at multiple scales***. Through reasoning explicitly about sampling and scale, mip-NeRF is able to reduce error rates relative to NeRF by 60% on our own multiscale dataset, and by 17% on NeRF's single-scale dataset, while also being 7% faster than NeRF. Mip-NeRF is also able to match the accuracy of a brute force supersampled NeRF variant, while being 22× faster.

2. 原架構缺點及可改善之處

a. Performing point-sampling along each ray

May therefore produce renderings that are excessively blurred or aliased when training or testing images observe scene content at different resolutions.

b. Constructing positional encoding (PE) features from an infinitesimally small point in space

two different cameras imaging the same position at different scales may produce the same ambiguous point-sampled feature, thereby significantly degrading NeRF's performance.

3. 改良實作或實機練習

a. Collecting the dataset

To compare with NeRF and MipNeRF fairly, We choose the same dataset as homework2. However, MipNeRF need multiscale image, so we construct it.

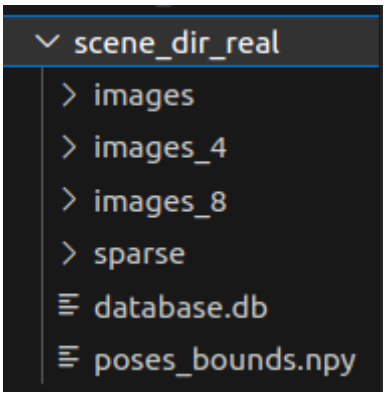

Figure 5. training set (images_4 is (H/2,W/2),and images_8 is (H/4,W/4))

b. Git clone the source code

```
git clone https://github.com/google/mipnerf.git
```

Figure 6. git clone command

c. Constructing the environment

Following the instruction from github to construct the environment for the model.

```
# Clone the repo
git clone https://github.com/google/mipnerf.git; cd mipnerf
# Create a conda environment, note you can use python 3.6-3.8 as
# one of the dependencies (TensorFlow) hasn't supported python 3.9 yet.
conda create --name mipnerf python=3.6.13; conda activate mipnerf
# Prepare pip
conda install pip; pip install --upgrade pip
# Install requirements
pip install -r requirements.txt
```

[Optional] Install GPU and TPU support for Jax

```
# Remember to change cuda101 to your CUDA version, e.g. cuda110 for CUDA 11.0.
pip install --upgrade jax jaxlib==0.1.65+cuda101 -f https://storage.googleapis.com/jax-releases/jax_
```

Figure 7. Instruction for constructing the environment

d. Training/Evaluating source code

Modify the contents of scripts provided by github to train/evaluate our dataset.

```
SCENE=trex
EXPERIMENT=debug
TRAIN_DIR=/data/scenario_retrieval/Interaction-benchmark/datasets/DIP_3/output_real
DATA_DIR=/data/scenario_retrieval/Interaction-benchmark/datasets/DIP_3/scene_dir_real
rm $TRAIN_DIR/*
CUDA_VISIBLE_DEVICES=2 python -m train \
  --data_dir=$DATA_DIR \
  --train_dir=$TRAIN_DIR \
  --gin_file=configs/llff.gin \
  --logtostderr
```

Figure 8.Training scripts

```
SCENE=trex
EXPERIMENT=debug
TRAIN_DIR=/data/scenario_retrieval/Interaction-benchmark/datasets/DIP_3/output_Real
DATA_DIR=/data/scenario_retrieval/Interaction-benchmark/datasets/DIP_3/scene_dir_real

CUDA_VISIBLE_DEVICES=2 python -m eval \
  --data_dir=$DATA_DIR \
  --train_dir=$TRAIN_DIR \
  --chunk=3076 \
  --gin_file=configs/llff.gin \
  --logtostderr
```

Figure 9.Evaluation scripts

4. 模型訓練與評估結果

Trainging result

Compared with NeRF, the training curve is more stable.

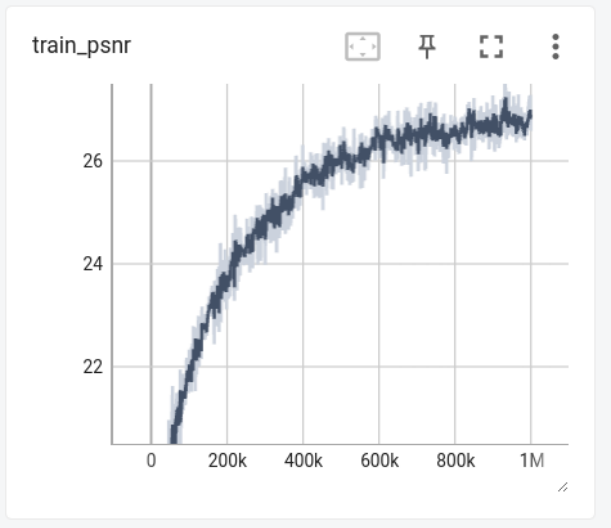

Figure 8. Training curve of psnr

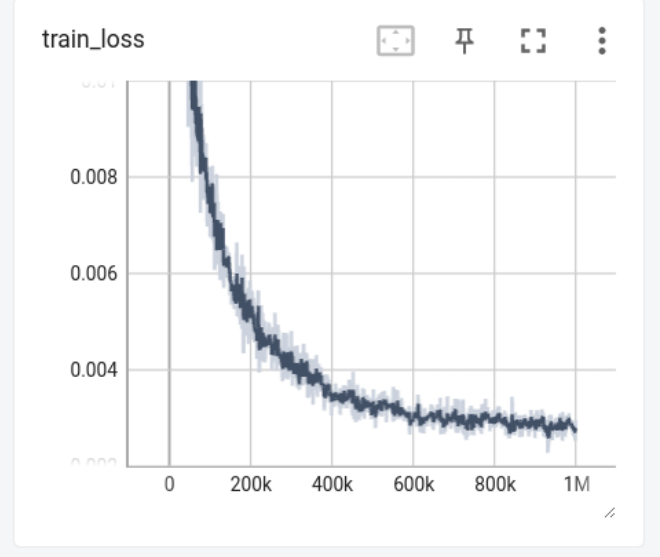

Figure 10.Training curve of loss

Evaluation result

The result of the evaluation is good. Its performance is better than NeRF.

Figure 11. Evaluation result

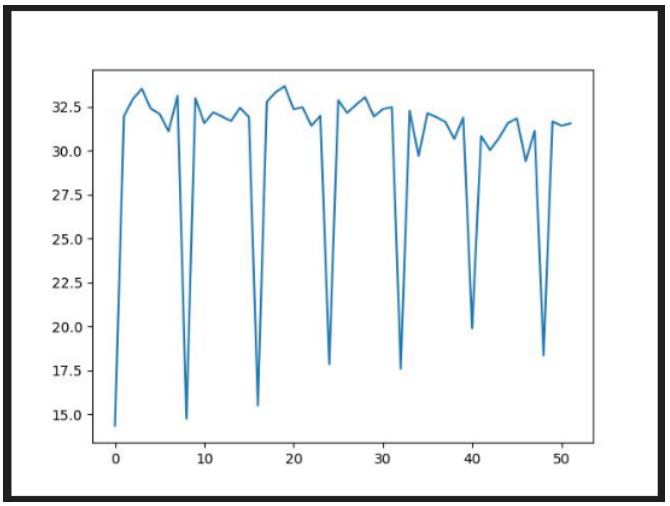

Figure 12. Test set image's psnr

5. 對作業三整體結果做出分析並討論

From these results, we can know that MiPNeRF contains the following attributes.

- Improved Rendering Efficiency: One key advantage of MiPNeRF over NeRF is its improved rendering efficiency. MiPNeRF achieves this by introducing a multi-resolution representation, allowing for faster rendering of complex scenes. This can significantly reduce the computational requirements and enable real-time or near-real-time rendering, making MiPNeRF more practical for interactive applications.
- High-Quality Reconstruction: MiPNeRF maintains high-quality reconstruction of scenes by leveraging a hierarchical representation. This enables accurate modeling of fine geometric details and complex lighting effects. By incorporating multi-scale features, MiPNeRF can capture both global scene structure and local variations, resulting in visually appealing and realistic renderings.

- Trade-offs in Accuracy: While MiPNeRF achieves faster rendering, there may be some trade-offs in accuracy compared to NeRF. The multi-resolution representation in MiPNeRF may result in slight compromises in capturing extremely fine details or handling occlusions. However, these trade-offs are often acceptable in practice considering the significant speed improvement.
- Scalability: MiPNeRF demonstrates better scalability compared to NeRF, particularly for large-scale scenes. By utilizing a hierarchical representation, MiPNeRF can efficiently handle scenes with a high level of complexity and a large number of scene elements. This scalability makes MiPNeRF suitable for various applications, including outdoor environments, virtual reality, and augmented reality.
- Practical Applications: The improved rendering efficiency of MiPNeRF opens up possibilities for real-time applications, such as interactive visualization, gaming, and virtual production. It allows for dynamic scene interactions, camera movements, and lighting changes with minimal latency. This makes MiPNeRF more accessible for real-world scenarios where responsiveness and interactivity are crucial.

Based on Figure 12, We can observe that most of the results are good. However, some of the results are not ideal. I think the reason comes from the data's quality. Because other model like NeRF have same shape of psnr curve.