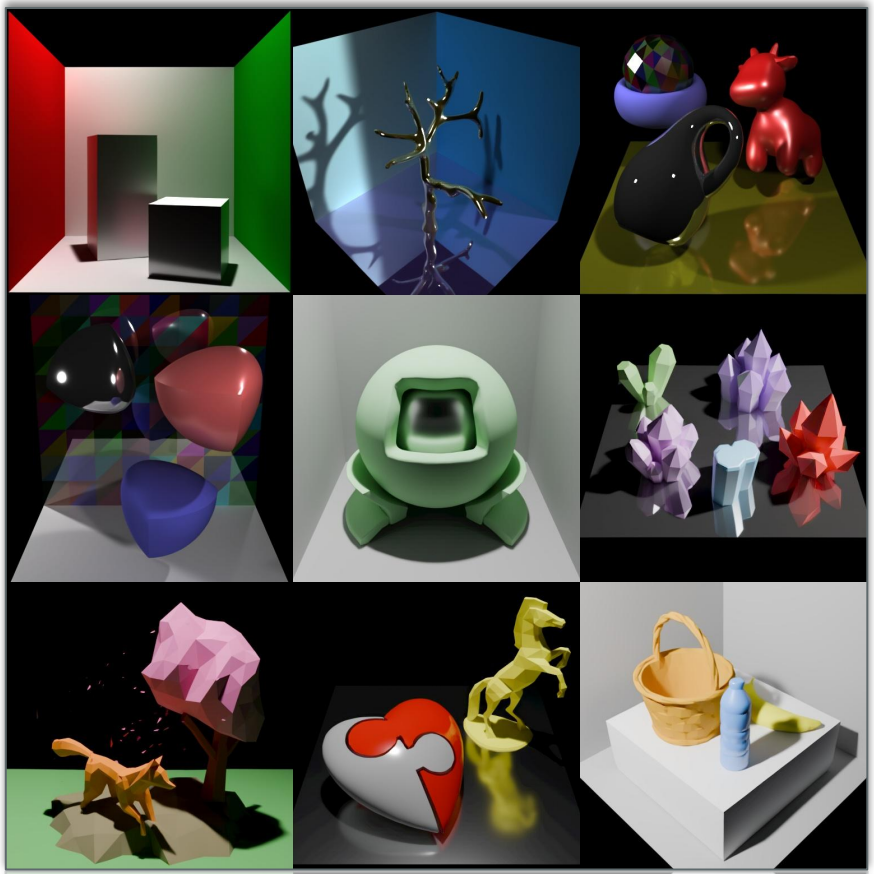
RenderFormer

Final Project Presentation

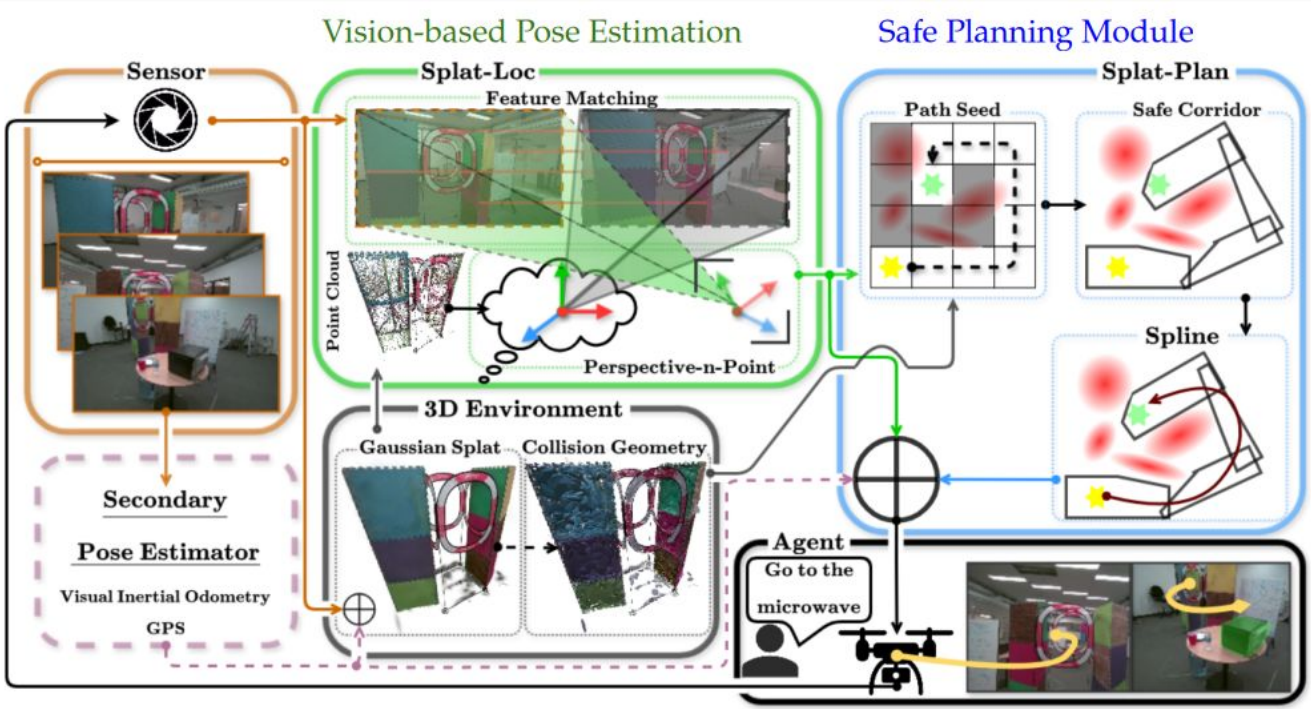
1st Dec, 2025

Team 4
Kyaw Ye Thu, Janghyun



https://microsoft.github.io/renderformer/

Review of Previous Presentation



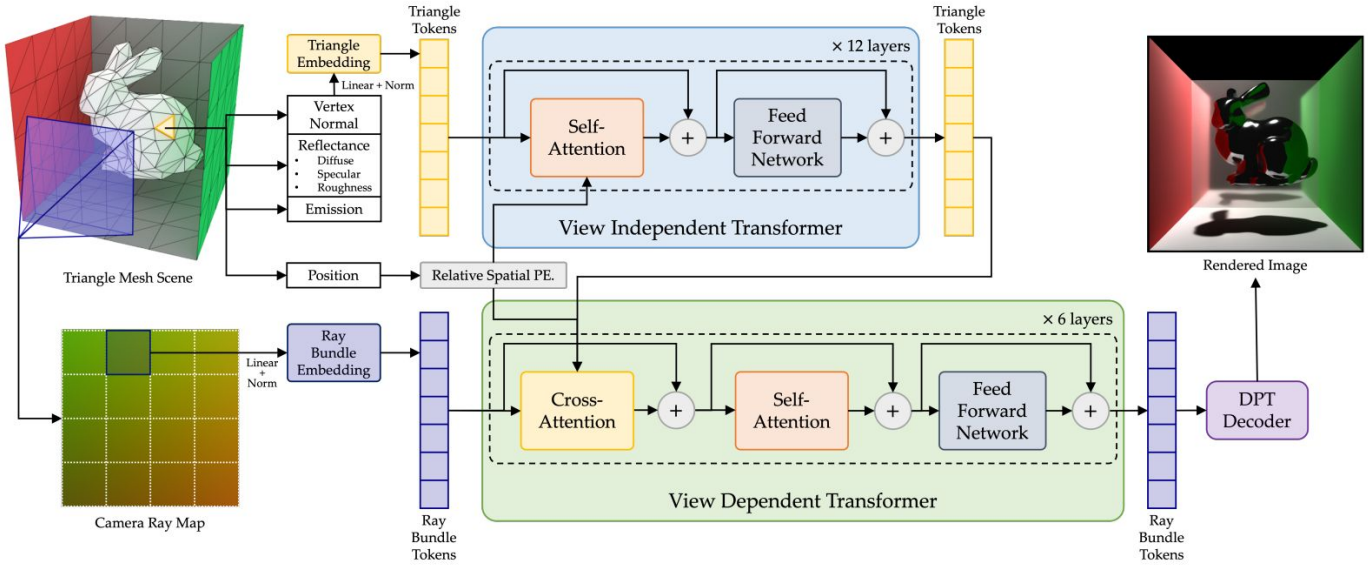
SplatNav : Safe Real-Time Robot Navigation in Gaussian Splatting Maps

Outline

- Our Work
- Brief Theory of Performer
- Dataset Generation & Training
- Results
  - Analysis of Rendered Images
  - Analysis of Time Complexity

Our Work

Recap of RenderFormer



- The view-independent stage scales roughly by O (#*tris*^2)
- view-dependent layers scales by O (#*bundles*^2 + #*bundles* × #*tris*).

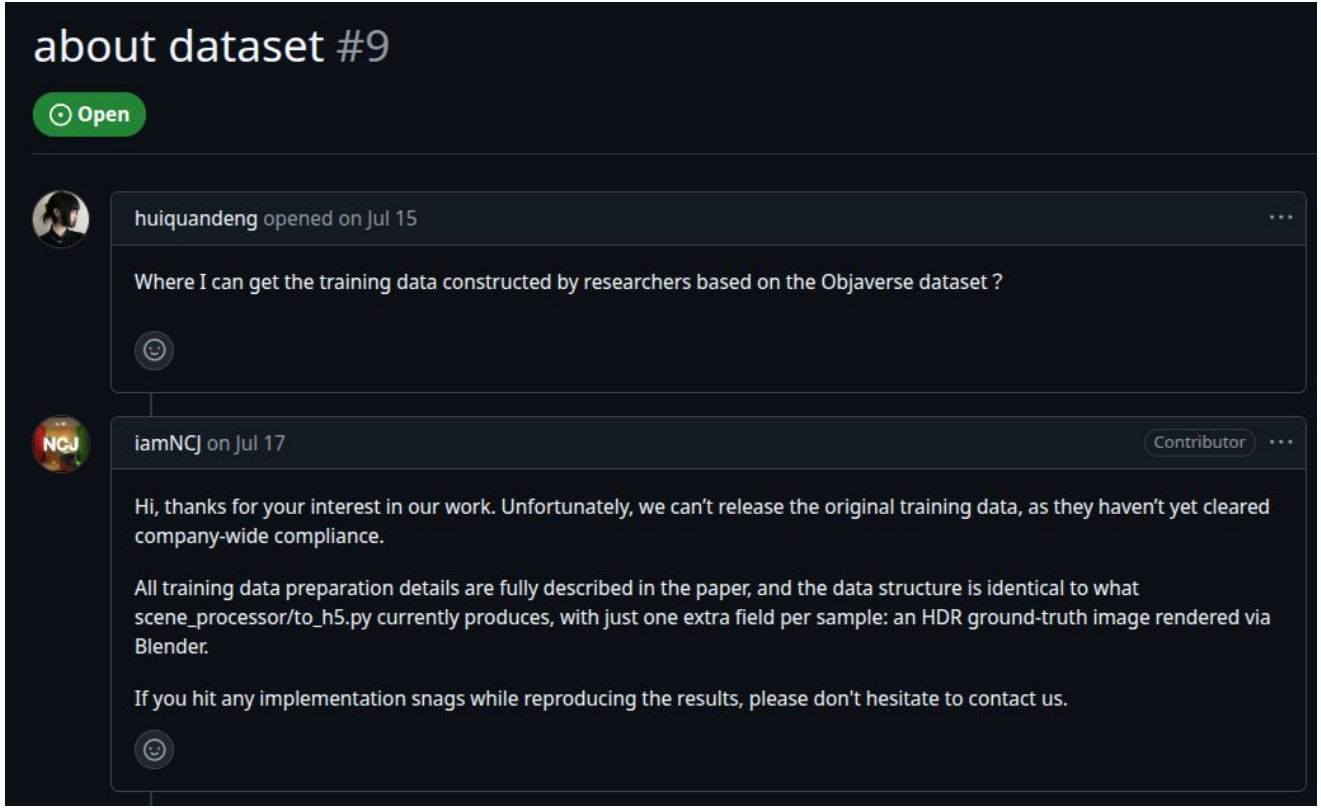
Our Plan (From Mid-term Presentation)

- Rewrite the **Inference** Part with Linear Attention or Performer
- Compare the Inference Speed and Rendering Result with the original
- Try **training** with Linear Attention or Performer
- Compare the Training Speed and Rendering Result with the original

What We Actually Did

- ✅ Rewrite the **Inference** Part with Linear Attention or Performer
- ✅ Compare the Inference Speed and Rendering Result with the original
- ✅ Try **training** with Linear Attention or Performer
  - Compare the *Training Speed* and ✅ Rendering Result with the original

No Dataset and Training Code Given😞



What We Actually Did

✅ Rewrite the **Inference** Part with Linear Attention or Performer

✅ Compare the Inference Speed and Rendering Result with the original

✅ Try **training** with Linear Attention or Performer

- Compare the *Training Speed* and ✅ Rendering Result with the original

✅ Create a Training Dataset

✅ Write Training Code

Our Roles

Dataset (Janghyun)

- Writing code for generating dataset
- Actually generating
- Making the dataset compatible with training

Training (Ye Thu)

- Writing code for training and performer
- Actually training
- Running inference speed experiments

https://github.com/kyaw-yethu/renderformer

Showing **41 changed files** with **8,693 additions** and **44 deletions.**

Brief Theory of Performer

Intuition of Linear Attention

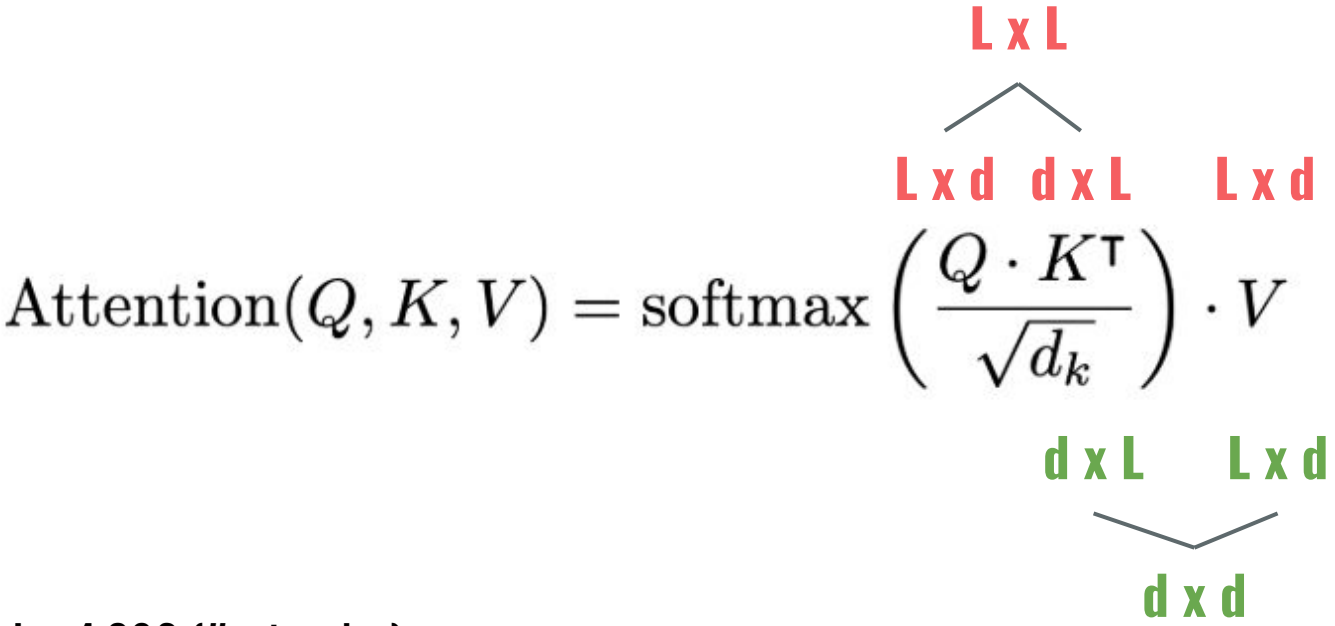
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q \cdot K^\intercal}{\sqrt{d_k}}\right) \cdot V$$

L x L

L x d d x L L x d

L = 4,096 (# triangles)
d = 768 (dimension of input sequence)

Intuition of Linear Attention



L = 4,096 (# triangles)
d = 768 (dimension of input sequence)

As long as d << L, significantly faster

Kernel trick used in Performer

Many kernels K(x,y) can be approximated as the inner products of x and y in some other dimensions

$$K(x, y) \approx \phi(x)^\top \phi(y)$$

where $\phi(\cdot)$ is a random nonlinear feature map.

Softmax-attention is a form of **kernel**:

$$\text{softmax}(q^\top k) = \exp(q^\top k) = \exp(q)^\top \exp(k)$$

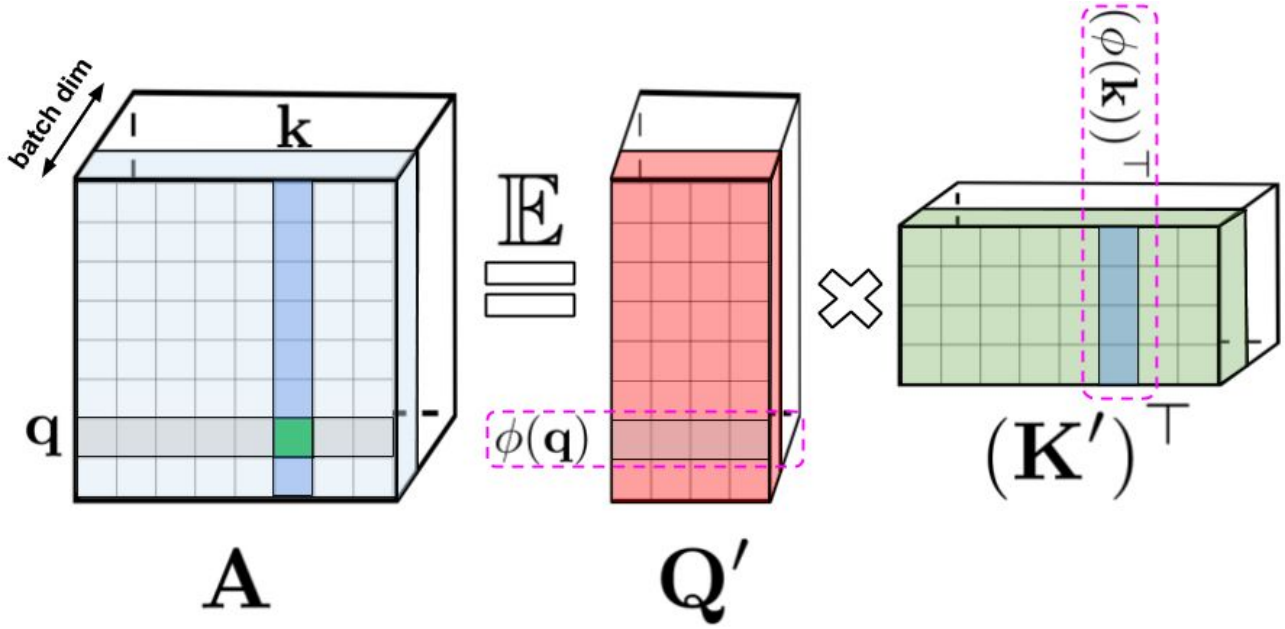
Performer / FAVOR+ approximates softmax with the following mapping.

$$\text{softmax}(qk^T) \approx \phi(q)\phi(k)^T$$

where $\phi(x) = \exp(\omega^T x - \frac{||x||^2}{2})$

FAVOR+: Fast Attention Via positive Orthogonal Random features

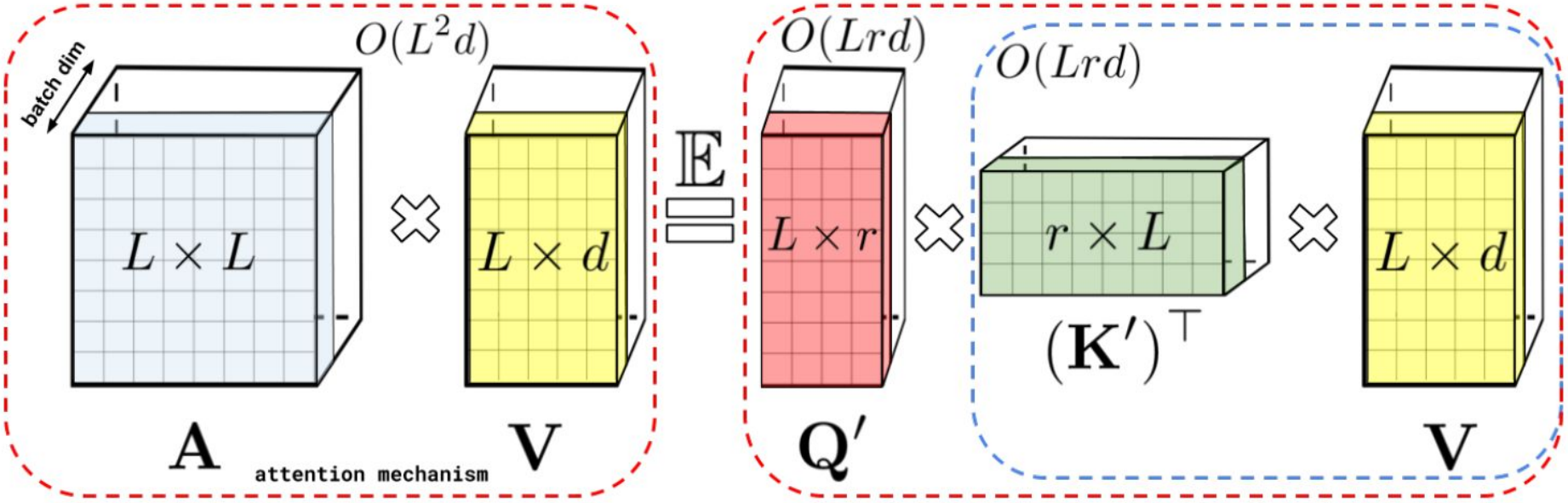
Approximating the Standard Attention with Lower-rank random matrices Q' and K'



$$A = \mathbb{E}[Q'K'^T] = \mathbb{E}[\phi(Q)\phi(K)T]$$

Overview of What Performer Does

Time complexity: $O(L^2 d)$ → $O(Lrd)$



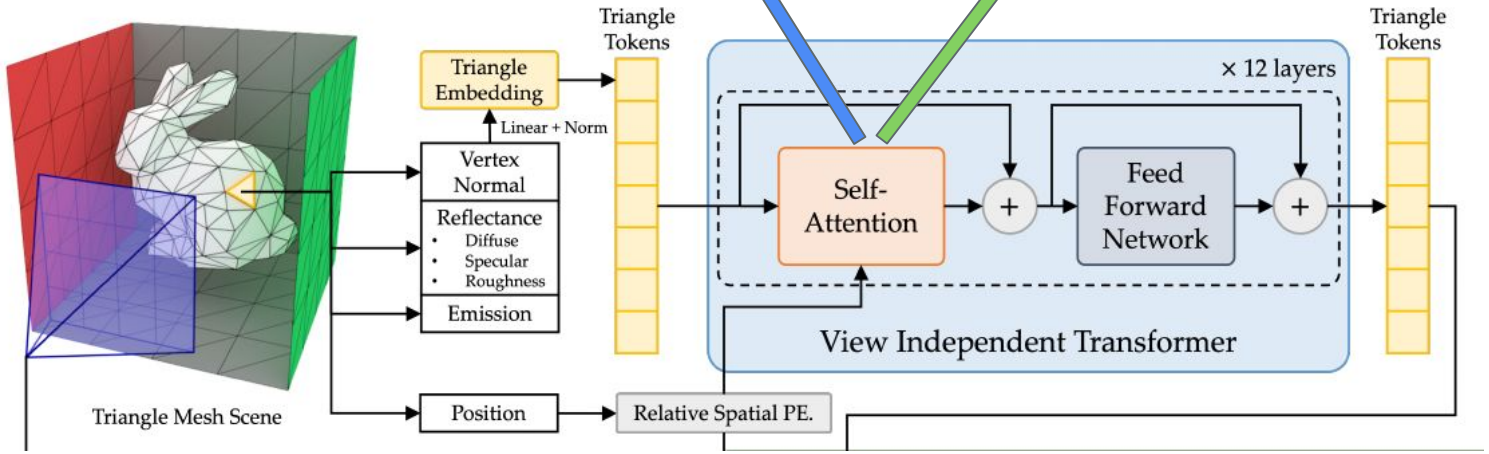
Space complexity: $O(L^2 + Ld)$ → $O(Lr + Ld + rd)$

Paper: Rethinking Attention with Performers

Recap of RenderFormer

Flash Attention (or) SDPA (Standard Dot-Product Attention)

Performer



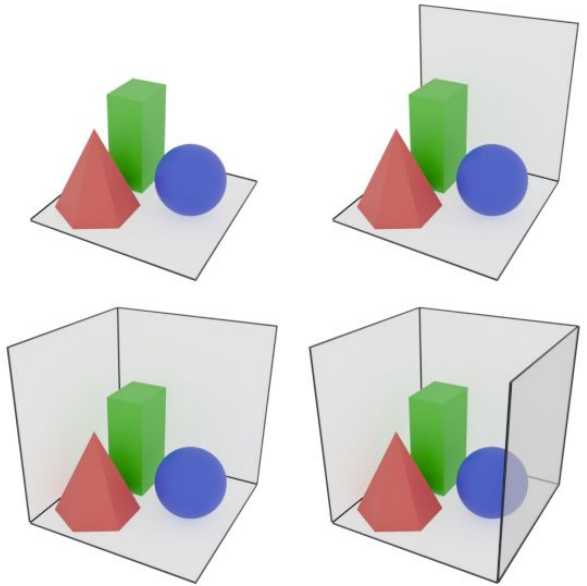
Base Model: microsoft/renderformer-v1-base

https://huggingface.co/microsoft/renderformer-v1-base

Dataset Generation & Training

Original dataset

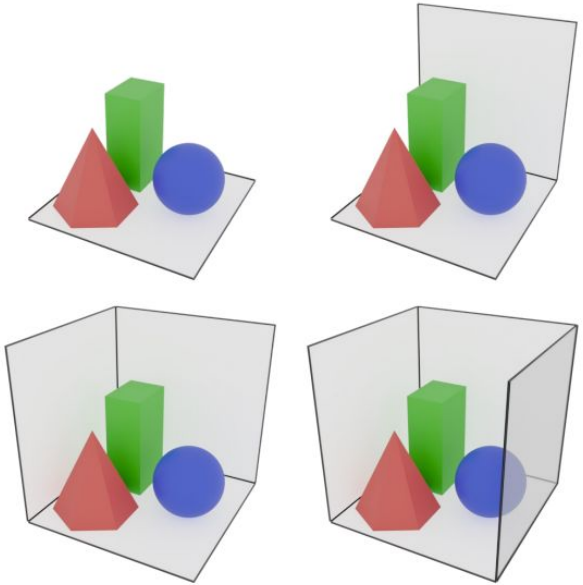
1~3 randomly selected objects
one of 4 random template scenes



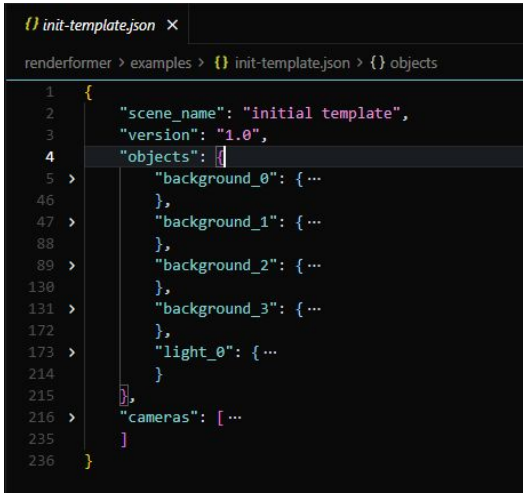
Slide Source: https://microsoft.github.io/renderformer/

Requirements for dataset

1~3 randomly selected objects
one of 4 random template scenes



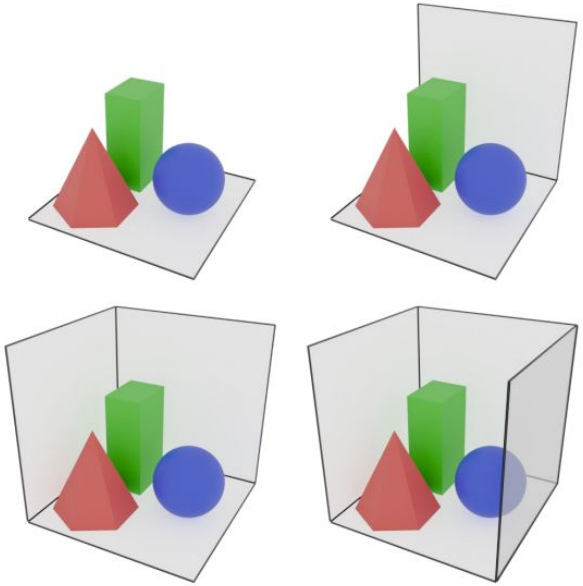
1. **Information for objects, light sources, and other elements forming each scene**
2. Ground truth of each scene



Slide Source: https://microsoft.github.io/renderformer/

Requirements for dataset

1~3 randomly selected objects
one of 4 random template scenes



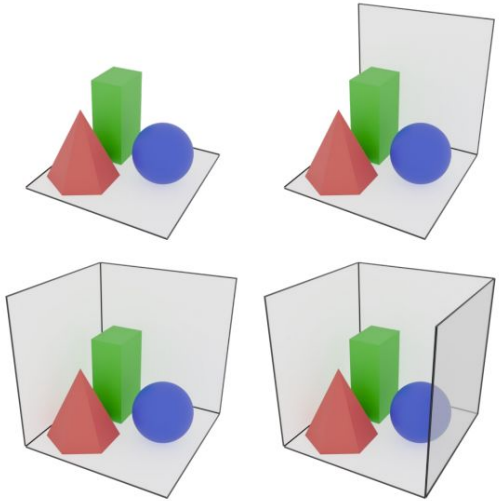
1. Information for objects, light sources, and other elements forming each scene
2. **Ground truth of each scene**

Slide Source: https://microsoft.github.io/renderformer/

Training Facts

Full Training (Their Work)

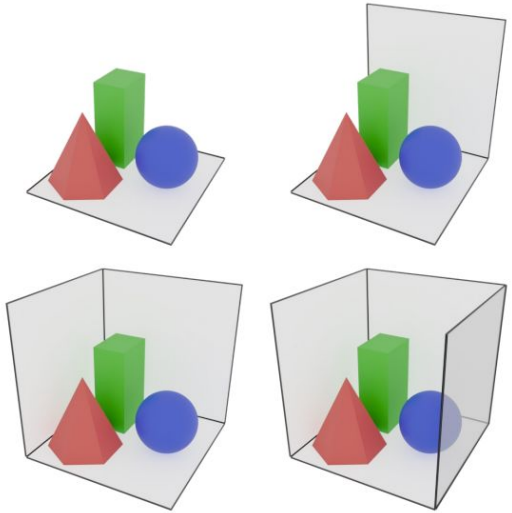
2M synthetic scenes



8M HDR training images
(4 different viewpoint per scene)

Fine-tuning (Our Work)

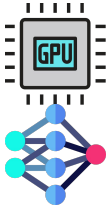
774 synthetic scenes



3096 HDR training images
(4 different viewpoint per scene)

Training Facts

Full Training (Their Work)



8x NVIDIA A100 GPUs with 40GB VRAM

Flash-Attention 2 and Liger Kernel

Batch Size = 128

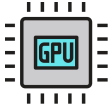
256x256 resolution
Max mesh size of 1,536 ▲

500K iterations
(~5 days)

512x512 resolution
Max mesh size of 4,096 ▲

+100K iterations
(~3 days)

Fine-tuning (Our Work)



1x A100 GPU 80 GB VRAM (Google Colab)

Performer for View Independent Stage
SDPA for View Dependent Stage

Batch Size = 4

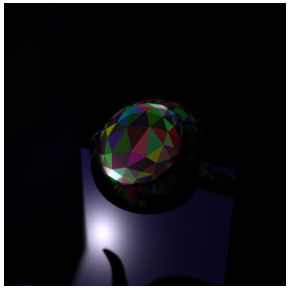
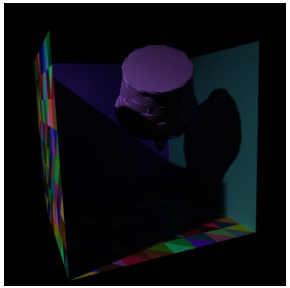
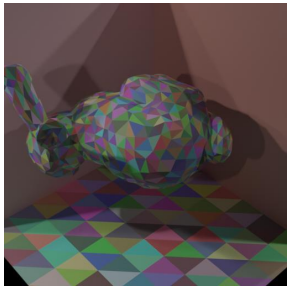
512x512 resolution
Max mesh size of 4,096 ▲

7,260 iterations
(~12 hours)

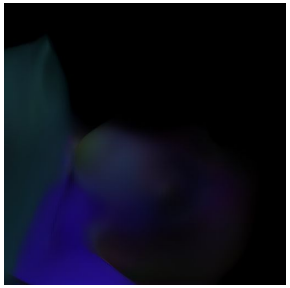
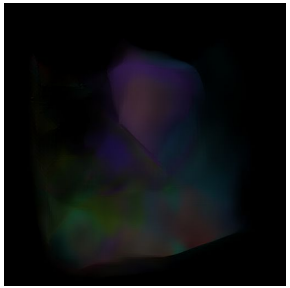
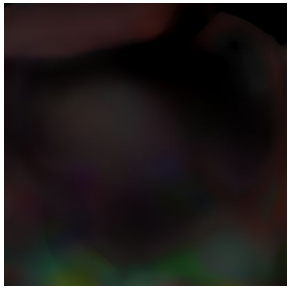
Analysis of Rendered Images

Rendered Images with Fine-tuned Model

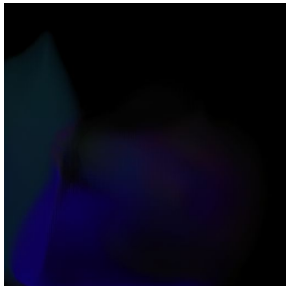
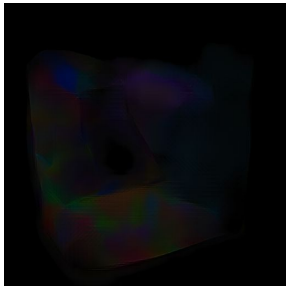
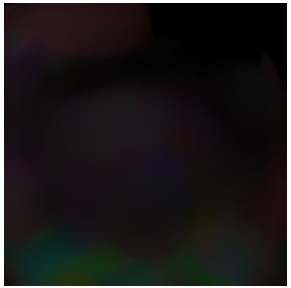
Ground-truth

Beginning of Fine-tuning

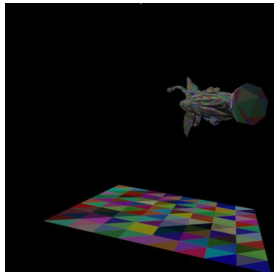
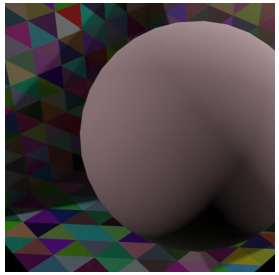
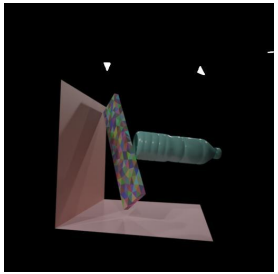




End of Fine-tuning

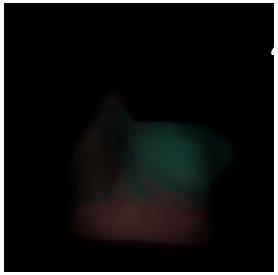
Rendered Images with Fine-tuned Model

Ground-truth

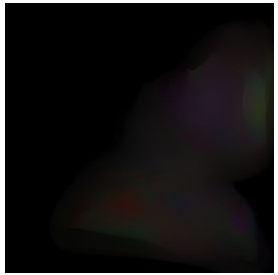
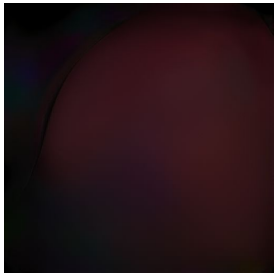
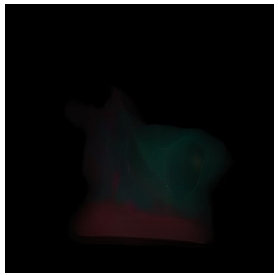




Beginning of Fine-tuning





End of Fine-tuning

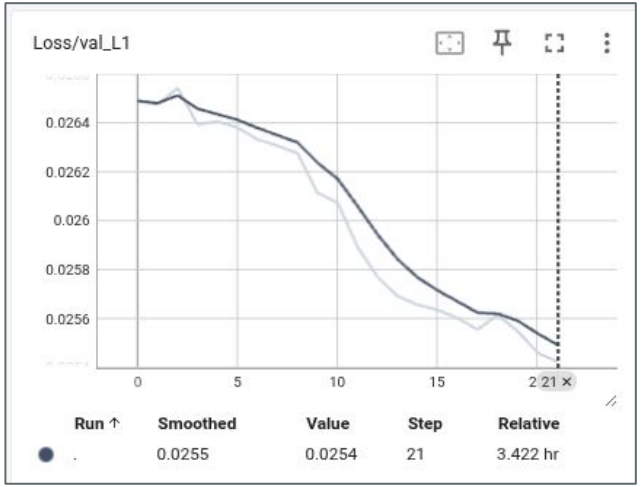




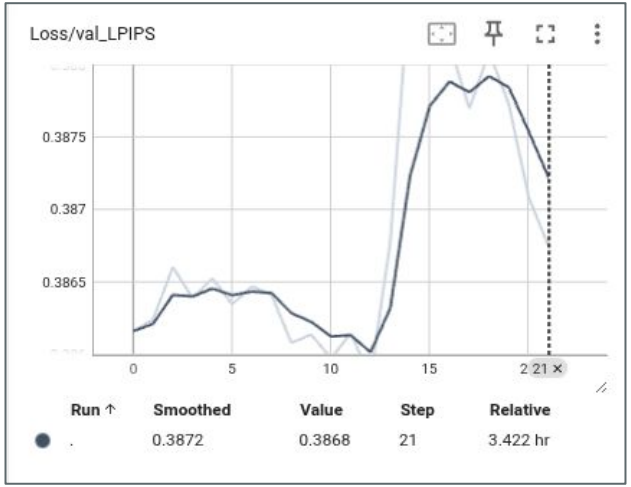
Validation Loss with Tensorboard

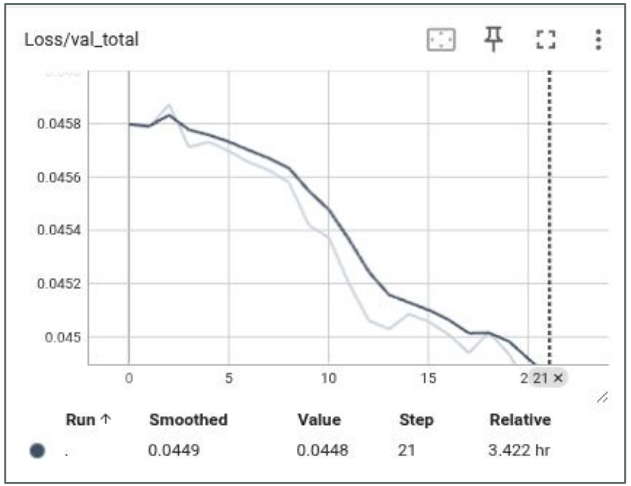
Apply log transform to the images first

$$loss_{L1} + 0.05 \cdot loss_{LPIPS}$$

(Learned Perceptual Image Patch Similarity)
Minimize perceptual differences







$loss_{L1}$

$loss_{LPIPS}$

Total Loss

A bug in the code …

LPIPS doesn't even backprop.

The gradient of `pred_flat` is cut from the graph -> `tone_pred` has no grad_fn

```
for i in range(batch_size):
    hdr_np = hdr_img[i].detach().cpu().numpy().astype(np.float32)
```

```
# LPIPS loss on tone-mapped images
tone_pred = self.tone_map(pred_flat)
tone_gt = self.tone_map(gt_flat)
print("tone_pred.requires_grad:", tone_pred.requires_grad)
# LPIPS expects [B, C, H, W] format
tone_pred = tone_pred.permute(0, 3, 1, 2)
tone_gt = tone_gt.permute(0, 3, 1, 2)

lpips_loss = self.lpips_fn(tone_pred, tone_gt).mean()

# Combined loss
total_loss = l1_loss + self.lpips_weight * lpips_loss

return total_loss, l1_loss, lpips_loss
```

```
Tone_pred.requires_grad: False
```

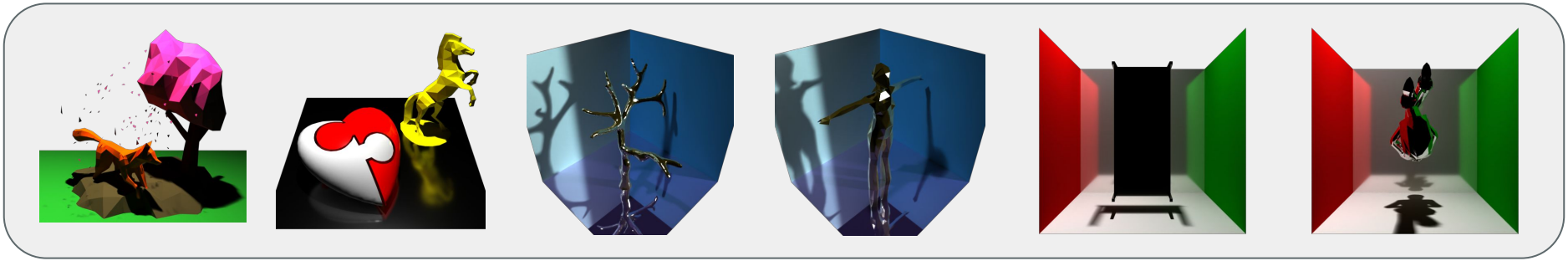
Analysis of Time Complexity

Inference Speed Comparison across Different Architectures

https://docs.google.com/spreadsheets/d/1XjYUAPYdK0RYXqfmNPjShg5rXeuOGQSVcsqDZzagxM0/edit?usp=sharing

*in seconds

Object Name	Triangles Count	Flash + SDPA			SDPA + SDPA			Performer + SDPA		
		No View	View	Total rendering	No View	View	Total rendering	No View	View	Total rendering
fox-in-the-wild	1161	0.0897	0.2711	1.5085	0.0432	0.2707	1.4407	0.1409	0.2758	1.5938
horse-and-heart	4894	0.0965	0.3074	1.5668	0.0632	0.3002	1.4913	0.1591	0.303	1.67
complex-tree	21082	0.394	0.4294	1.9749	0.5618	0.4277	2.1628	0.3369	0.4232	1.8667
tomdrinwoman	28362	0.6576	0.4899	2.2993	0.9269	0.526	2.685	0.4242	0.4932	2.0704
saloon_table	65535	2.5437	0.8091	4.5716	4.5024	0.8209	6.5227	0.8469	0.7963	2.813
sitfemale	65535	2.5466	0.7982	4.5415	4.5144	0.8318	6.5354	0.8533	0.7973	2.8184

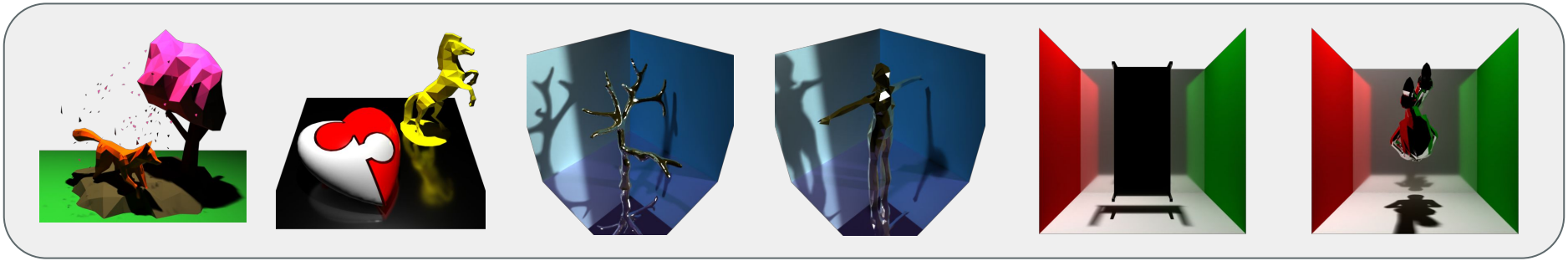


Inference Speed Comparison across Different Architectures

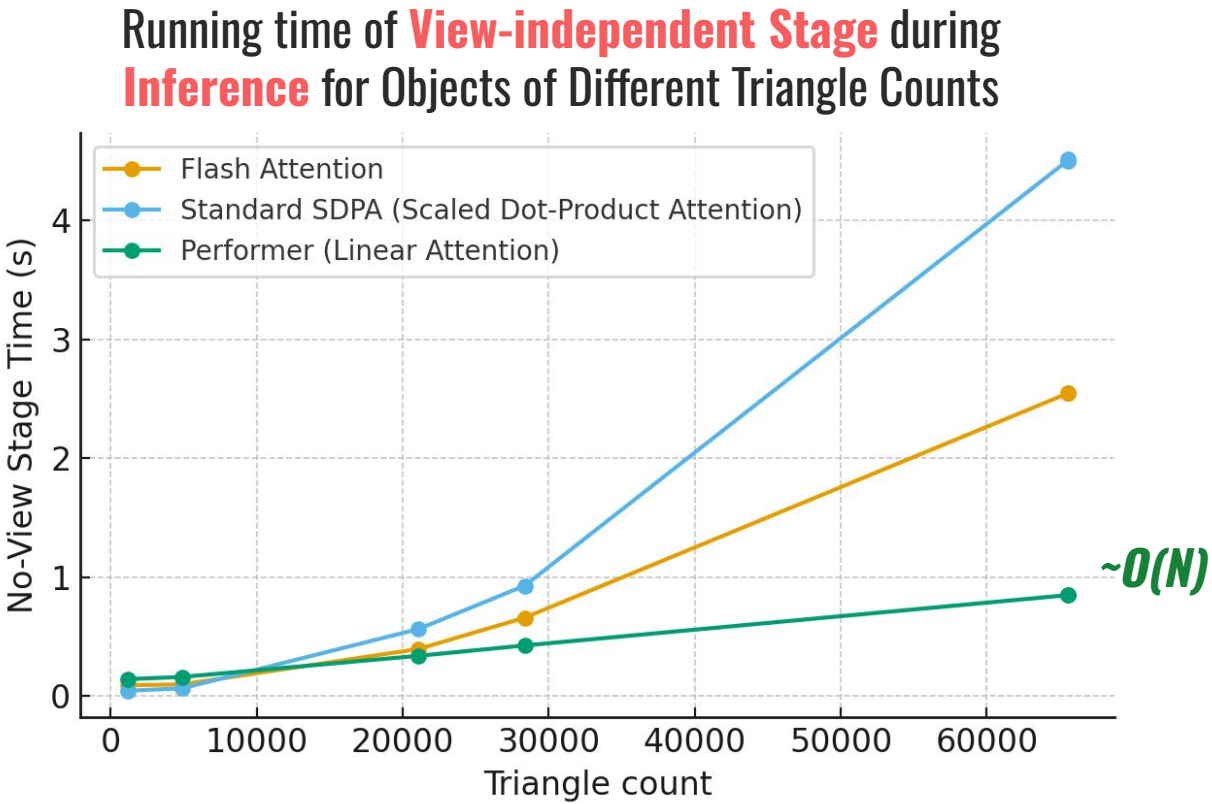
https://docs.google.com/spreadsheets/d/1XjYUAPYdK0RYXqfmNPjShg5rXeuOGQSVcsqDZzagxM0/edit?usp=sharing

*in seconds

Object Name	Triangles Count	Flash + SDPA			SDPA + SDPA			Performer + SDPA		
		No View	View	Total rendering	No View	View	Total rendering	No View	View	Total rendering
fox-in-the-wild	1161	0.0897	0.2711	1.5085	0.0432	0.2707	1.4407	0.1409	0.2758	1.5938
horse-and-heart	4894	0.0965	0.3074	1.5668	0.0632	0.3002	1.4913	0.1591	0.303	1.67
complex-tree	21082	0.394	0.4294	1.9749	0.5618	0.4277	2.1628	0.3369	0.4232	1.8667
tomdrinwoman	28362	0.6576	0.4899	2.2993	0.9269	0.526	2.685	0.4242	0.4932	2.0704
saloon_table	65535	2.5437	0.8091	4.5716	4.5024	0.8209	6.5227	0.8469	0.7963	2.813
sitfemale	65535	2.5466	0.7982	4.5415	4.5144	0.8318	6.5354	0.8533	0.7973	2.8184

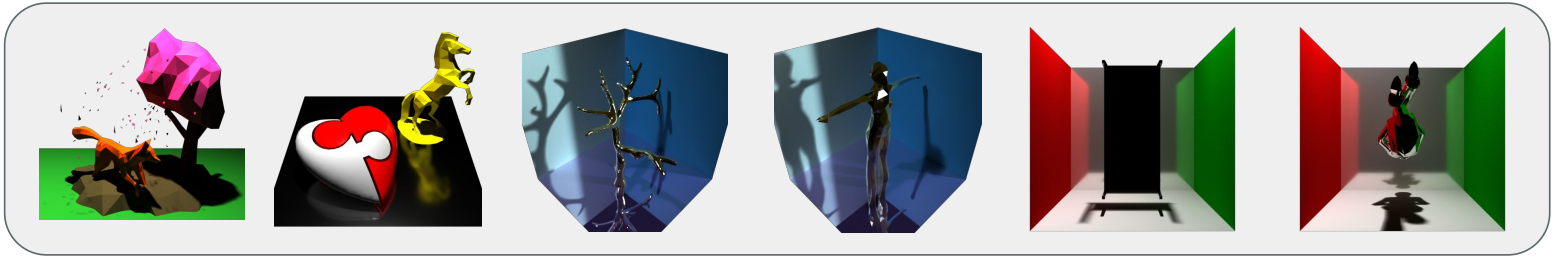


Inference Speed Comparison across Different Architectures

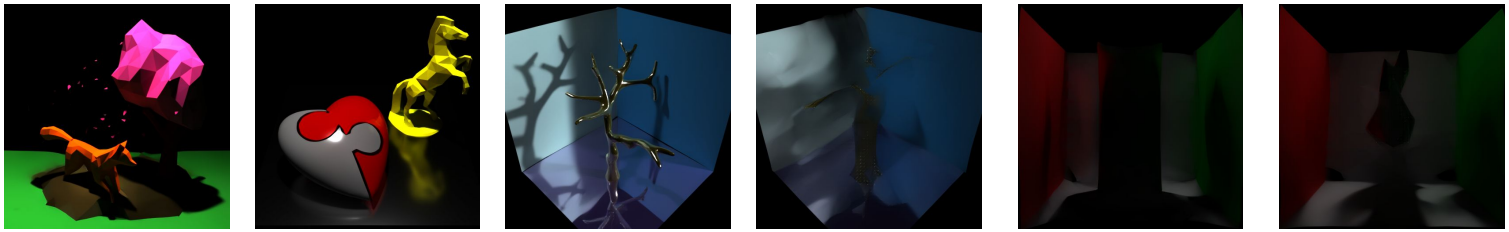


Rendered Images for Inference Speed Comparison

Ground Truth



Flash Attention



Performer

~5K

~20K

~60K

Inside Training Range

Outside Training Range

Final Thoughts

Final Thoughts

- If we can
  - train with a dataset as big as the original one,
  - train for more iterations
  - train without gradient flow bug

→ Match the performance of the original architecture

- Now with bigger triangle budget enabled by linear attention,

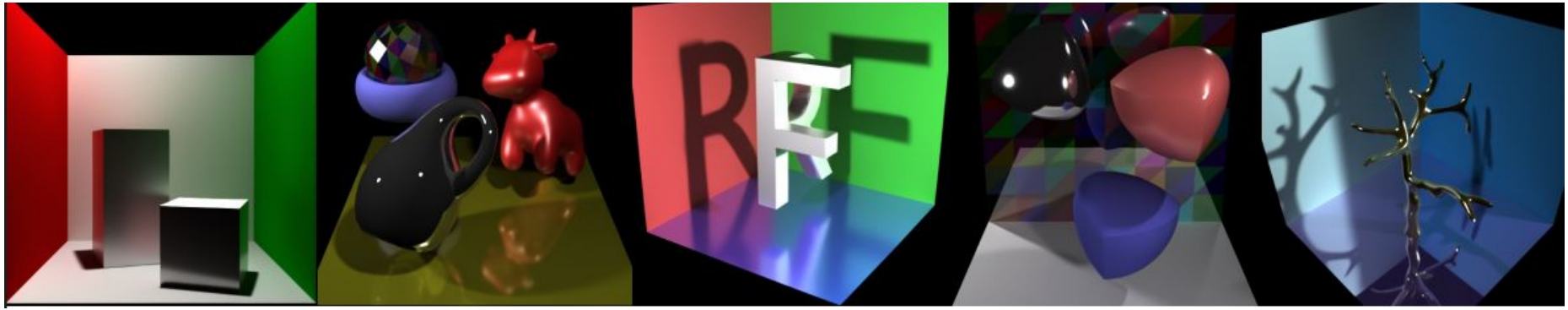
→ Feasible to train with scenes composed of more objects (triangles). (rendering of realistic scenes)

The architecture is fundamentally compatible with LLM-style scaling, but creating a dataset with every possible realistic scene may be unrealistic.

Conclusion

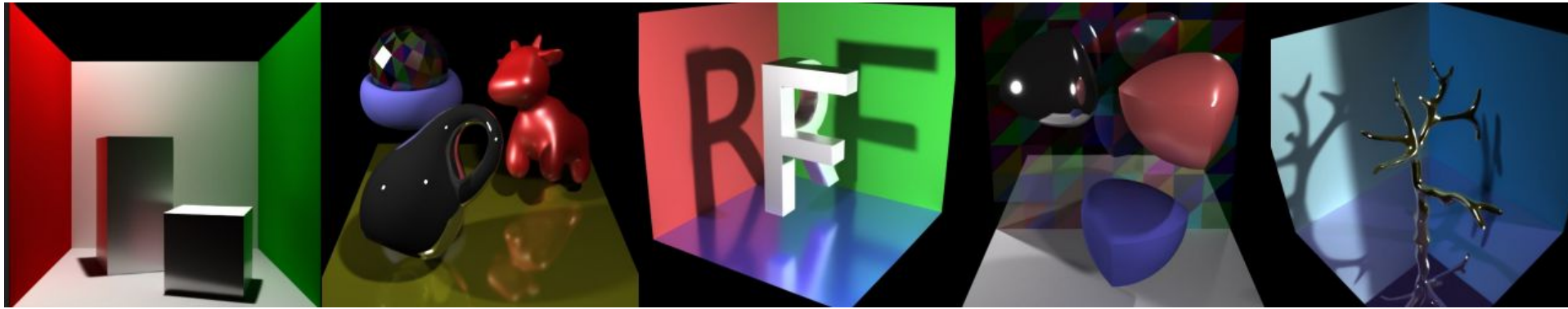
Attention Is ~~All~~ You Need for Rendering

Attention *can be Some* You Need for *Photorealistic* Rendering



Thank you for Your Attention!

RenderFormer: Transformer-based Neural Rendering of Triangle Meshes with Global Illumination



Project Deliverables

Github Repo: https://github.com/kyaw-yethu/renderformer

Google Drive (Colab file, Dataset, Training Logs, Inference Speed Experiments, Rendered Images: https://github.com/kyaw-yethu/renderformer