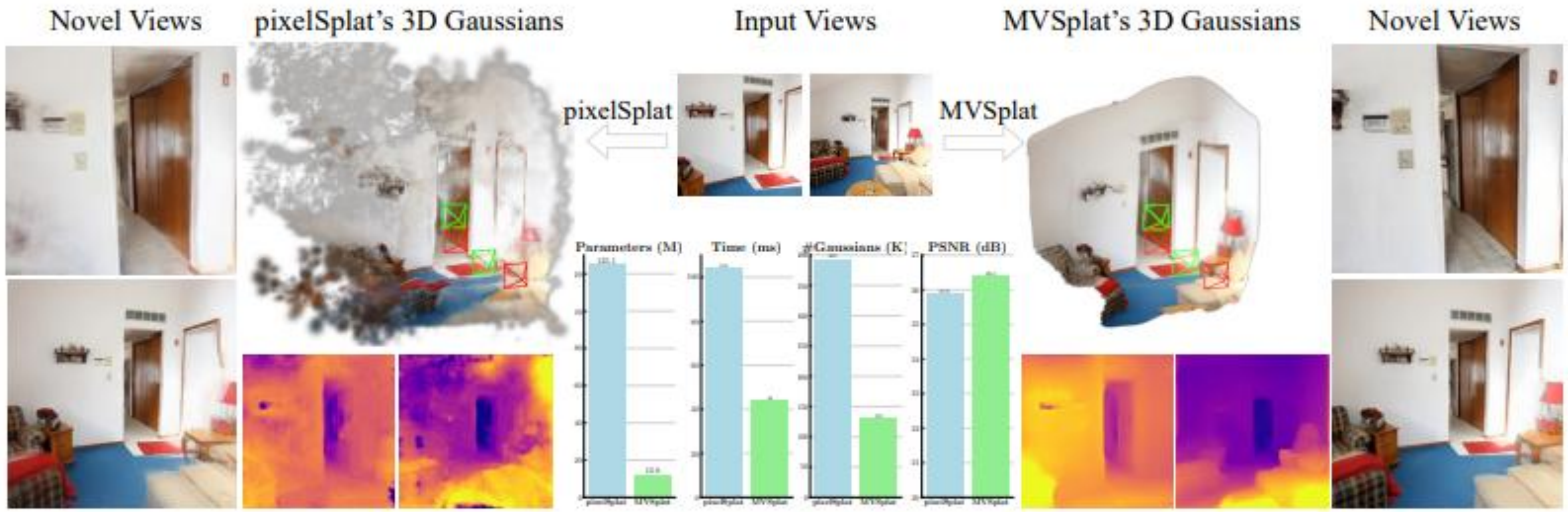
MVSplat: Efficient 3D Gaussian Splatting from Sparse Multi-View Images
(ECCV 2024 Oral)



한국과학기술연구원(KIST)
CVIPL 학생연구원 **김연욱**

한국과학기술연구원
Korea Institute of Science and Technology

Contents

OverView

MVSplat:
given **sparse multi-view images** as input, predicts clean **feed-forward 3D Gaussians**.

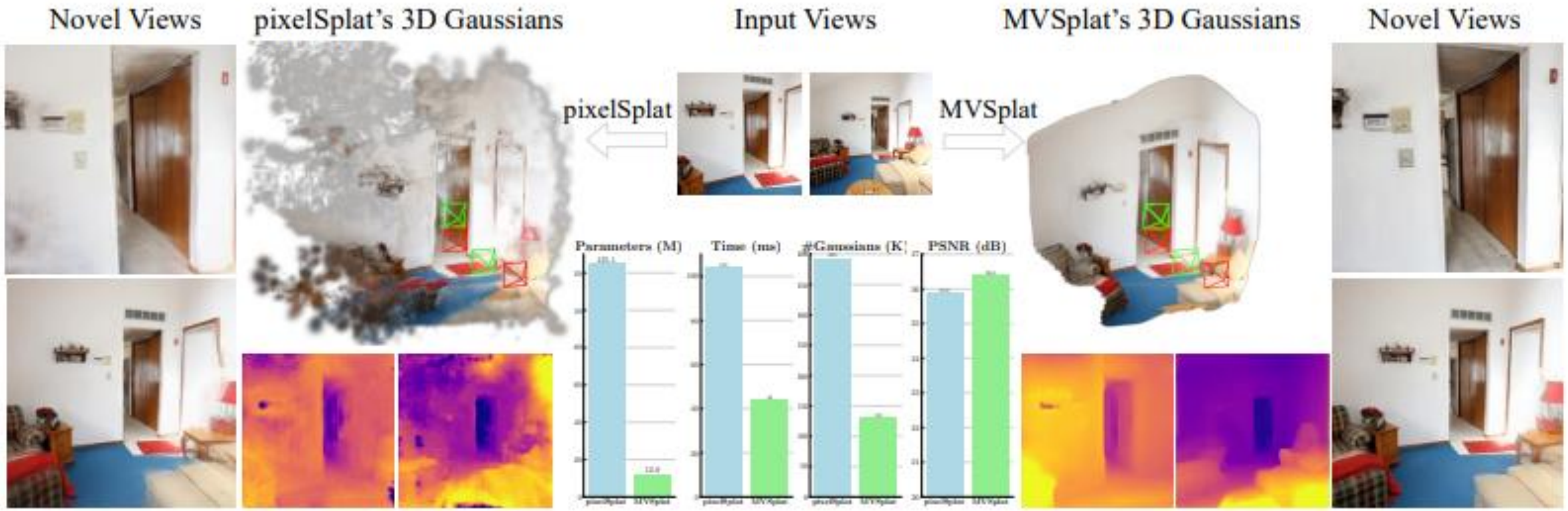
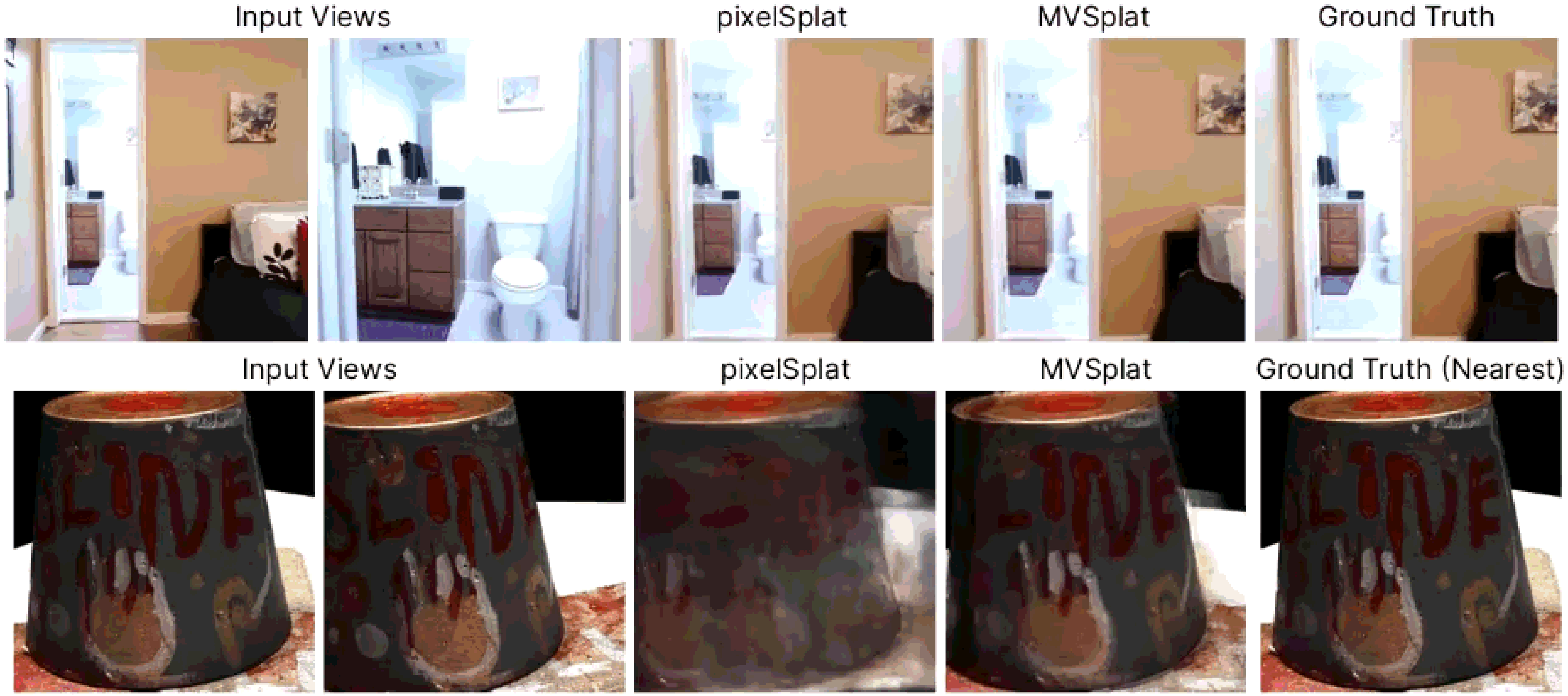


Fig. 1: Our MVSplat outperforms pixelSplat [1] in terms of both appearance and geometry quality with 10× fewer parameters and more than 2× faster inference speed.

OverView


Input Views
pixelSplat
MVSplat
Ground Truth
Input Views
pixelSplat
MVSplat
Ground Truth (Nearest)

Introduction

- Remarkable progress has been made in 3D Reconstruction.
- But, still **not satisfactory** for **practical applications**.

Due to …

- 기존 NeRF와 3DGS는 많은 수의 **views**를 필요로 한다.
- Scene마다 **개별적으로 최적화 과정**을 수행해야 한다

따라서 실제 응용에서는 어려울 수 있다.

→ Sparse input view로 Scene을 Reconstruction 및 Synthesis하는 연구가 증가

Introduction

Sparse input view method

- Per-scene approaches:
  - mainly focus on designing **effective regularization terms** for **optimization** process.
  - Expansive **per-scene gradient back-propagation process**

- Feed-forward approaches:
  - **learn** powerful **priors** from largescale **datasets**
  - so, 3D reconstruction and view synthesis can be achieved via **a single feed-forward inference**

Feed-forward NeRF Models : expensive volumetric sampling process 때문에 Inference 시간 ↑

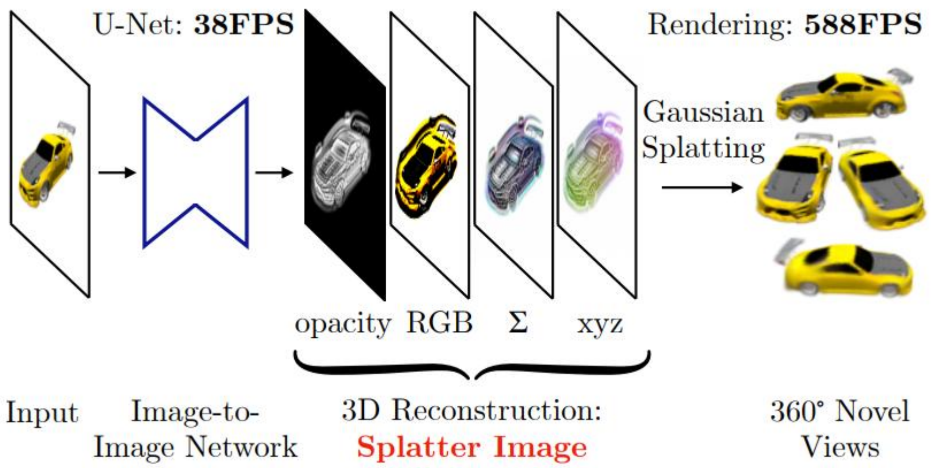
→ 3DGS 기반의 **Feed-forward Model**들이 활발히 연구되고 있다.

Preliminaries

Splatter Image

- **Single-view object-level** 3D reconstruction model
- Reconstructing a 3D scene from a **single image**
- But, inherently **ill-posed** and **ambiguous**

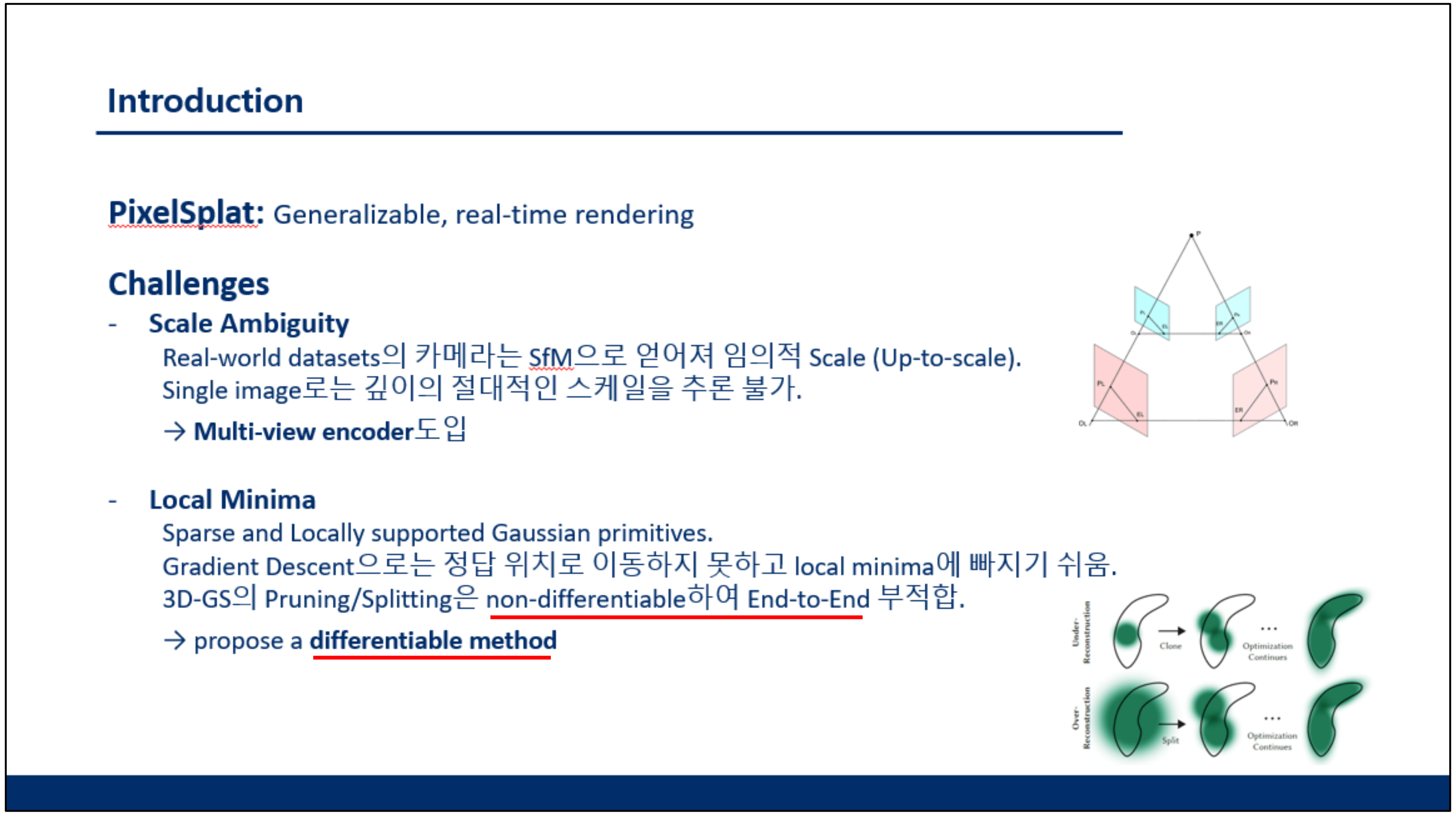
→ 일반적인 상황과 더 큰 Scene에선 큰 성능 저하



Preliminaries

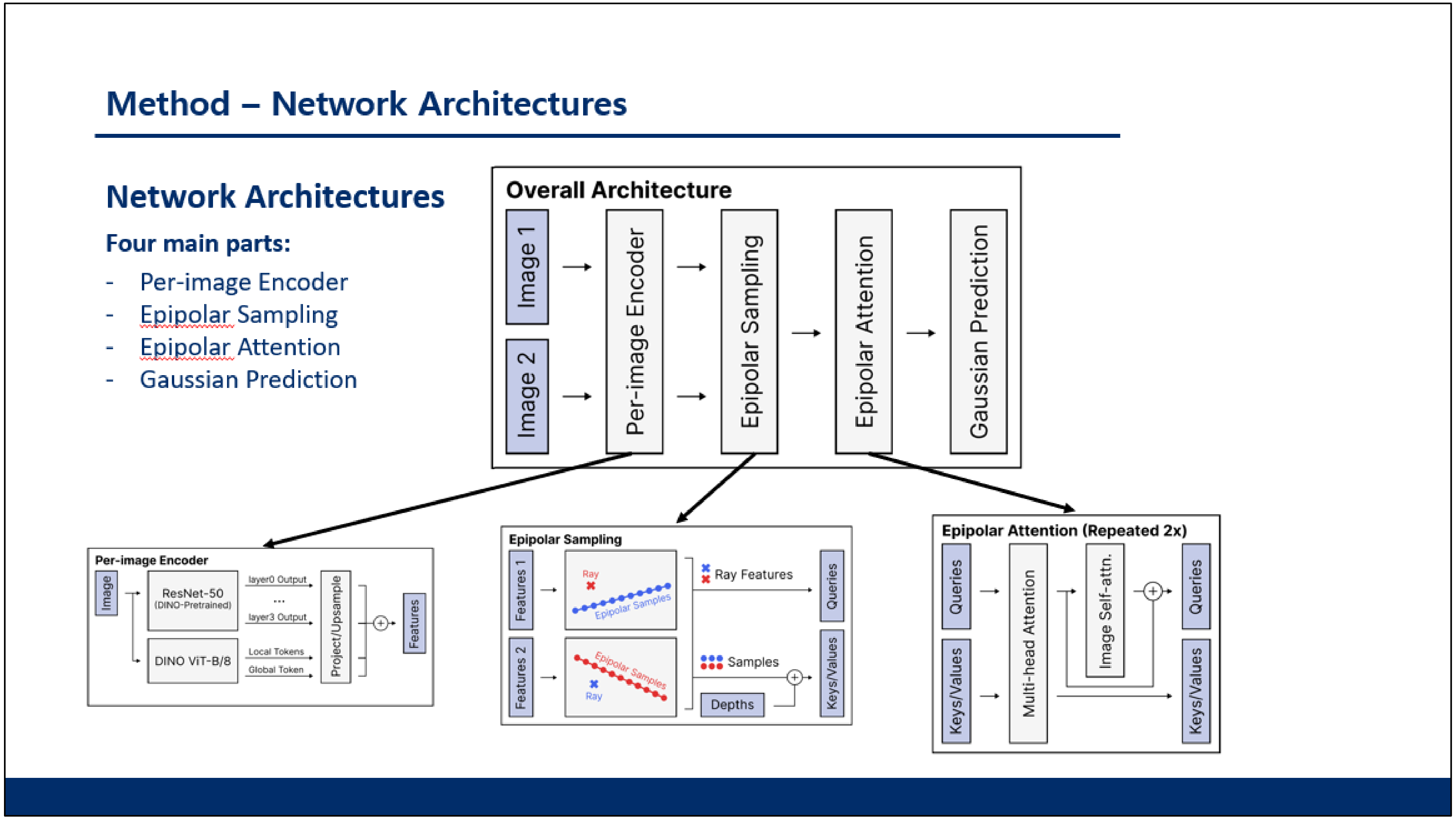
PixelSplat

- **Image 쌍**으로부터 **3D Gaussian primitives**로 표현된 **3D radiance field**를 재구성하는 **generalizable feed-forward** model



Preliminaries

PixelSplat



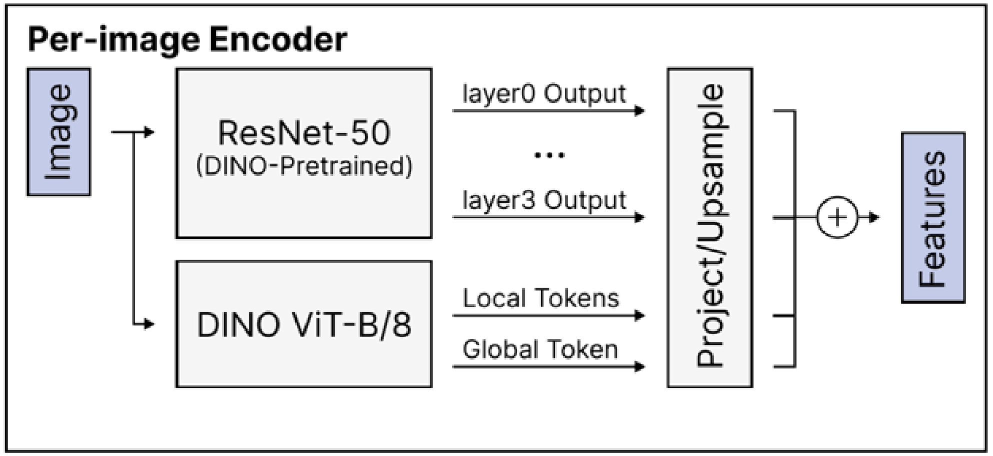
Preliminaries

PixelSplat

Method

Resolving Scale Ambiguity

1. Encode each view into feature volumes $\mathbf{F}$ and $\tilde{\mathbf{F}}$ via **per-image feature encode**



Preliminaries

PixelSplat

Method

Resolving Scale Ambiguity

2. Let **u** be **pixel coordinates** from image **I**, and **ℓ** be the **epipolar line** induced by its ray in **~I**.
3. Along **ℓ**, sample pixel coordinates.
4. compute its **distance** to I's camera origin $\tilde{d}_{\tilde{\mathbf{u}}_l}$ by **triangulation** of u and $\tilde{\mathbf{u}}_l$. (각 샘플까지의 거리 구하기)

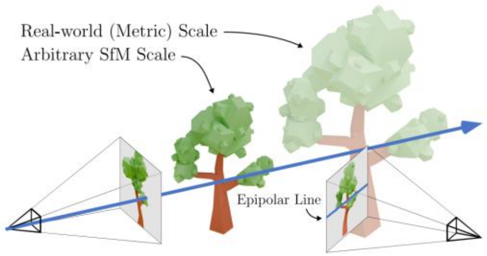
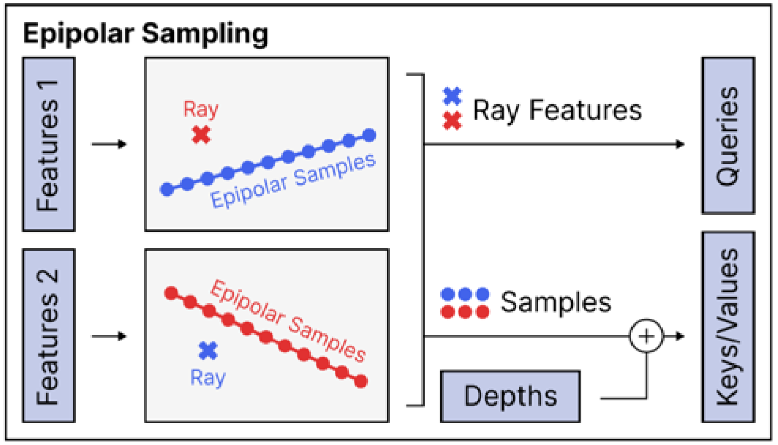


Figure 2. **Scale ambiguity.** SfM does not reconstruct camera poses in real-world, metric scale—poses are scaled by an arbitrary scale factor that is different for each scene. To render correct views, our model's 3D reconstruction needs to be consistent with this arbitrary scale. We illustrate how our epipolar encoder solves this problem. Features belonging to the ray's corresponding pixel on the left are compared with features sampled along the epipolar line on the right. Epipolar samples are augmented with their positionally-encoded depths along the ray, which allows our encoder to record correct depths. Recorded depths are later used for depth prediction.



Preliminaries

PixelSplat

Method

Resolving Scale Ambiguity

2. Let **u** be **pixel coordinates** from image **I**, and **ℓ** be the **epipolar line** induced by its ray in **~I**.
3. Along **ℓ**, sample pixel coordinates.
4. compute its **distance** to I's camera origin $\tilde{d}_{\tilde{\mathbf{u}}_l}$ by **triangulation** of u and $\tilde{\mathbf{u}}_l$.
5. Then, **cross-attention** and update per-pixel feature F[u] (어떤 sample이 진짜 정답인지)

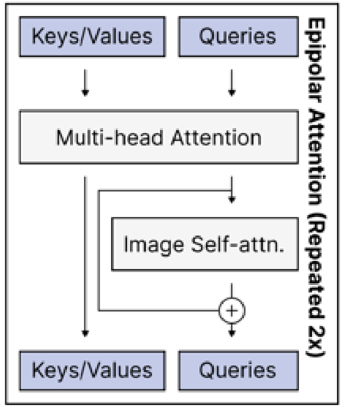
~I 이미지에 위치한 l번째 sample의 픽셀 좌표

l번째 sample의 카메라까지 거리

Source token

$$\mathbf{s} = \tilde{\mathbf{F}}[\tilde{\mathbf{u}}_l] \oplus \gamma(\tilde{d}_{\tilde{\mathbf{u}}_l}) \quad (1)$$

$$\mathbf{q} = \mathbf{Q} \cdot \mathbf{F}[\mathbf{u}], \quad \mathbf{k}_l = \mathbf{K} \cdot \mathbf{s}, \quad \mathbf{v}_l = \mathbf{V} \cdot \mathbf{s}, \quad (2)$$



Preliminaries

PixelSplat

Method

Resolving Scale Ambiguity

2. Let **u** be **pixel coordinates** from image **I**, and **ℓ** be the **epipolar line** induced by its ray in ~**I**.
3. Along **ℓ**, sample pixel coordinates.
4. compute its **distance** to I's camera origin $\tilde{d}_{\tilde{\mathbf{u}}_l}$ by **triangulation** of u and $\tilde{\mathbf{u}}_l$.
5. Then, **cross-attention** and update per-pixel feature F[u] (어떤 sample이 진짜 정답인지)

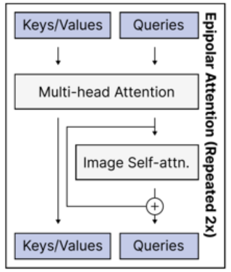
Source token

~I 이미지에 위치한 l번째 sample의 픽셀 좌표

l번째 sample의 카메라까지 거리

$$\mathbf{s} = \tilde{\mathbf{F}}[\tilde{\mathbf{u}}_l] \oplus \gamma(\tilde{d}_{\tilde{\mathbf{u}}_l}) \quad (1)$$

$$\mathbf{q} = \mathbf{Q} \cdot \mathbf{F}[\mathbf{u}], \quad \mathbf{k}_l = \mathbf{K} \cdot \mathbf{s}, \quad \mathbf{v}_l = \mathbf{V} \cdot \mathbf{s}, \quad (2)$$



Method

Resolving Scale Ambiguity

6. After this, each pixel feature F[u] encodes the **scaled depth** that is **consistent** with the arbitrary **scale factor** s_i of the camera poses.

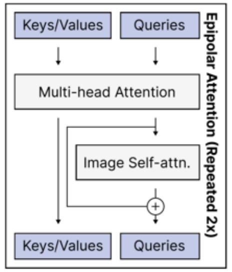
$$\mathbf{F}[\mathbf{u}] \mathrel{+}= \mathrm{Att}(\mathbf{q}, \{\mathbf{k}_l\}, \{\mathbf{v}_l\}), \quad (3)$$

Skip connection

7. This enables our encoder to **propagate scaled depth estimates to parts** of the image feature maps that may **not have any epipolar correspondences** in the opposite image.

$$\mathbf{F} \mathrel{+}= \mathrm{SelfAttention}(\mathbf{F}). \quad (4)$$

(Scale Ambiguity 해결한 Feature 생성)



- Mechanism can be **extended** to **more than two** input views

Preliminaries

PixelSplat

Method

Gaussian Parameter Prediction

- define a discrete probability distribution $p_\phi(z)$
- ϕ : vector of discrete probabilities (이산확률벡터)
- ϕ_z : probability that a surface exists in depth bucket b_z (표면이 b_z에 존재할 확률)
- δ : a per-bucket center offset (각 구간에서의 offset)
- $\mathbf{o}$: 카메라 렌즈의 위치(원점)
- $\mathbf{d}_u$: ray direction(vector)

$$\phi, \delta, \Sigma, \mathbf{S} = \underbrace{f}_{\text{FCNN}}(\mathbf{F}[\mathbf{u}]).$$

$$\boldsymbol{\mu} = \mathbf{o} + (\mathbf{b}_z + \boldsymbol{\delta}_z)\,\mathbf{d}_\mathbf{u},\ z \sim p_\phi(z),\ (\phi, \boldsymbol{\delta}) = f(\mathbf{F}[\mathbf{u}]) \quad (7)$$

네트워크가 예측한 확률 분포에 따라, 실제 가우시안을 심을 위치(인덱스 z)를 하나 '샘플링'

Method – Network Architectures

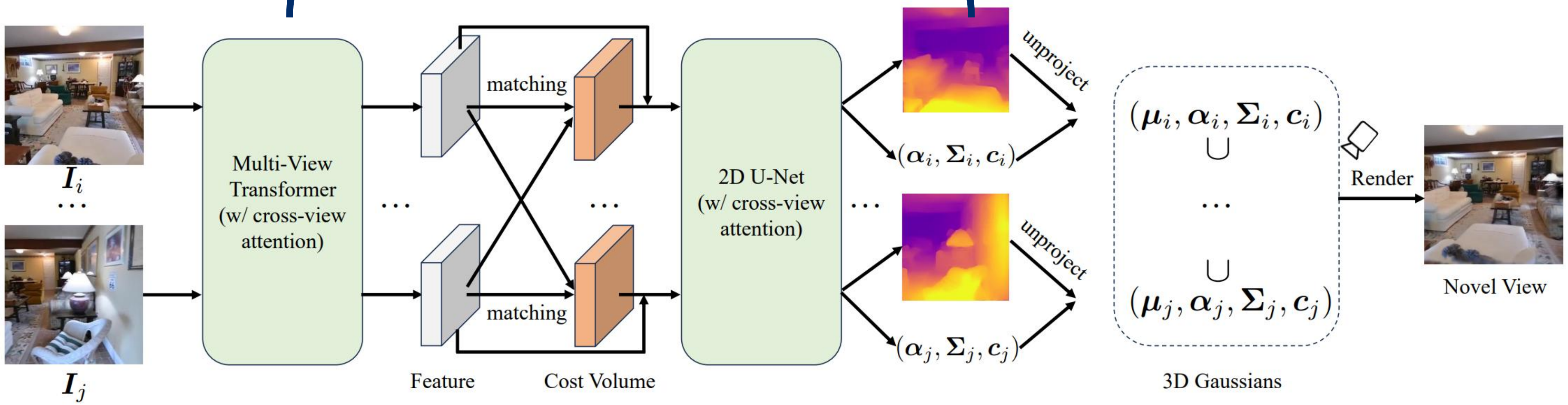
Network Architectures

Five main parts:

- Multi-View feature extraction
- Cost Volume Construction
- Cost Volume Refinement
- Depth Estimation (+ Refinement)
- Gaussian parameters Prediction

Key Point: Precise Depth Prediction

Depth Prediction



Method – Network Architectures

Goal: Learn a mapping $f_{\boldsymbol{\theta}}$ from images to 3D Gaussian parameters

Input: posed images

- K sparse-view images

  $\mathcal{I} = \{\boldsymbol{I}^i\}_{i=1}^{K}, \ (\boldsymbol{I}^i \in \mathbb{R}^{H \times W \times 3})$

- Camera projection matrices

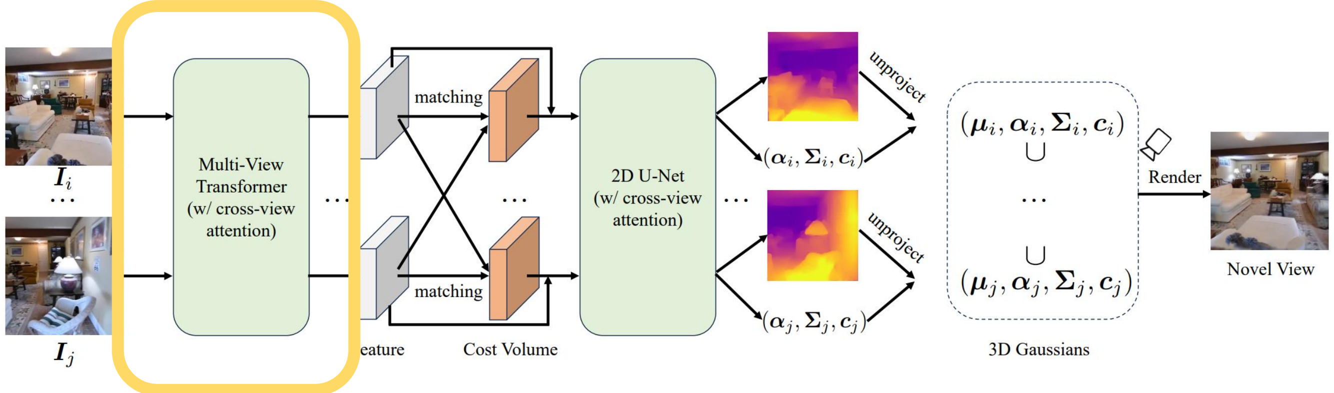
  $\mathcal{P} = \{\boldsymbol{P}^i\}_{i=1}^{K}, \ \boldsymbol{P}^i = \mathbf{K}^i[\mathbf{R}^i|\mathbf{t}^i]$ (Intrinsic K, Rotation R, Translation t)

$$f_{\boldsymbol{\theta}} : \{(\boldsymbol{I}^i, \boldsymbol{P}^i)\}_{i=1}^{K} \mapsto \{(\boldsymbol{\mu}_j, \alpha_j, \boldsymbol{\Sigma}_j, \boldsymbol{c}_j)\}_{j=1}^{\underline{H \times W \times K}}$$

이미지 크기 x 이미지 수
➔ Pixel-aligned

$f_{\boldsymbol{\theta}}$: feed-forward network
$\boldsymbol{\mu}_k$: mean
$\boldsymbol{\Sigma}_k$: covariance
$\boldsymbol{\alpha}_k$: opacity
$\mathbf{S}_k$: coefficient

Method



Multi-View feature extraction

- purely based on **2D convolutions** and **attentions**, **without** any **3D** convolutions
  → highly efficient

First, a shallow ResNet-like **CNN** is used to extract **4× downsampled** per-view **image features**.

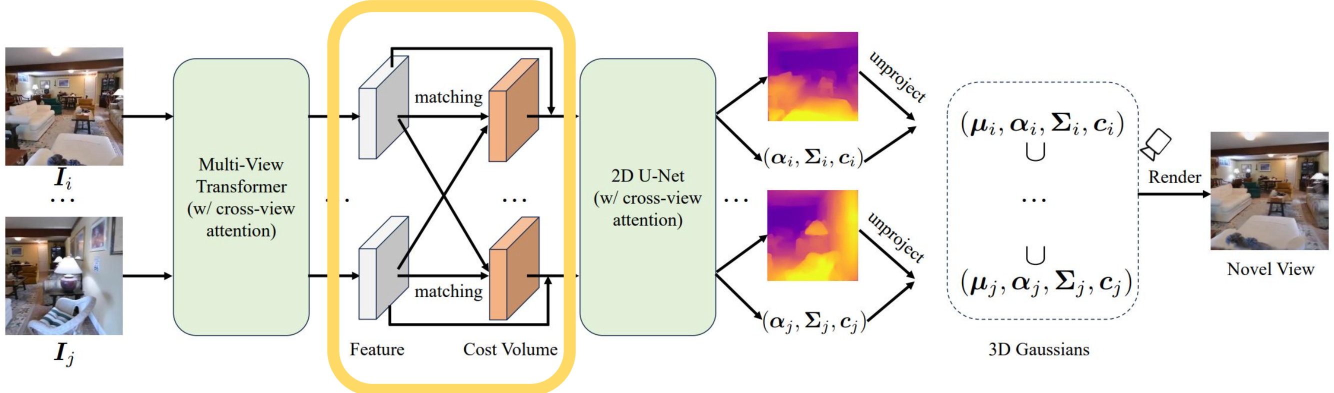
Then, use a multi-view **Swin Transformer**(for efficiency) with **self** and **cross-attention** layers

- **Self-attention** : to exchange information in one view.
- **Cross-attention** : to exchange information between different views.

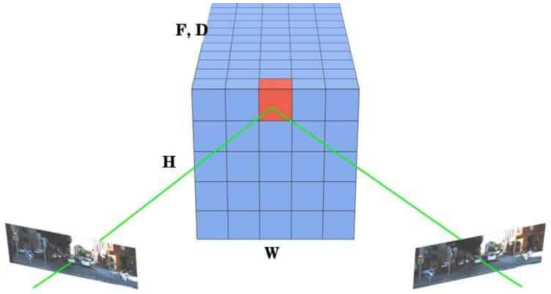
After this, can obtain <u>cross-view aware Transformer features</u>

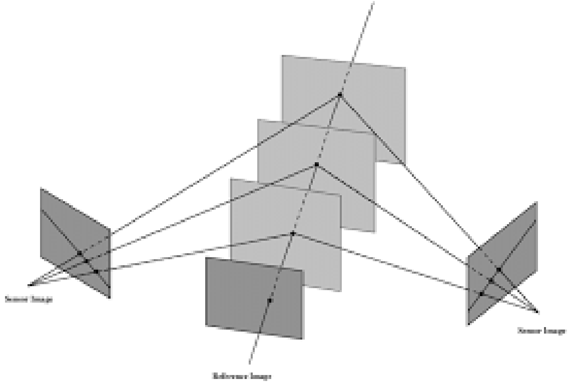
$$\{\boldsymbol{F}^i\}_{i=1}^{K} \ (\boldsymbol{F}^i \in \mathbb{R}^{\frac{H}{4} \times \frac{W}{4} \times C})$$

Method



Cost Volume Construction





Cost Volume

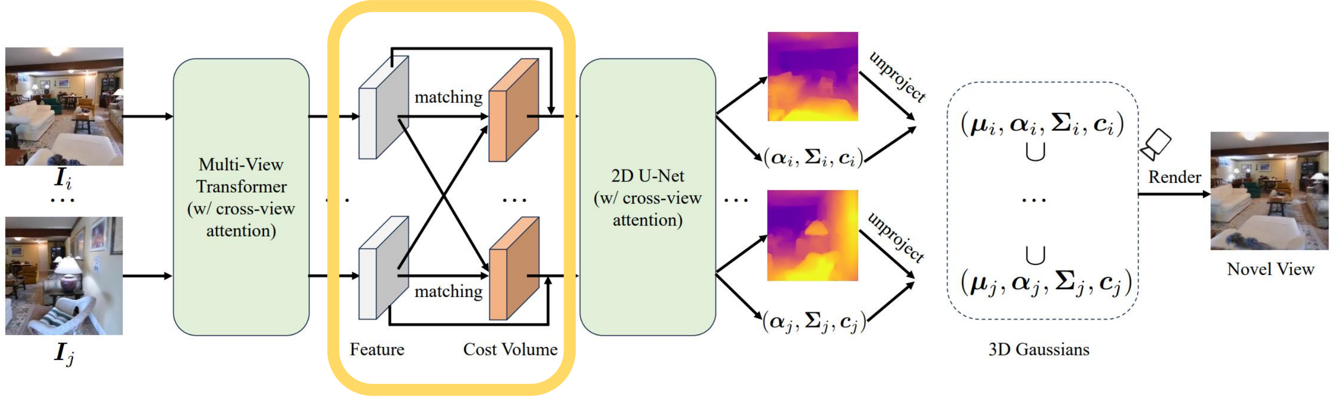
- 2D 이미지를 3D 공간에 매핑해서, 해당 위치에 표면이 있을 확률을 나타내는 데이터 구조

Plane Sweeping

- 연속적인 3D 공간의 Depth를 유한한 개수의 Plane들로 이산화하여, 각 평면에서의 매칭 비용을 계산하는 기법

이 논문에서는 Cost volume을 plane sweeping로 구현하였다.

Method



Cost Volume Construction

- The **key Component** of MVSplat.
- Models **cross-view feature matching information** with respect to different **depth candidates**

Ex) View *i*'s cost volume

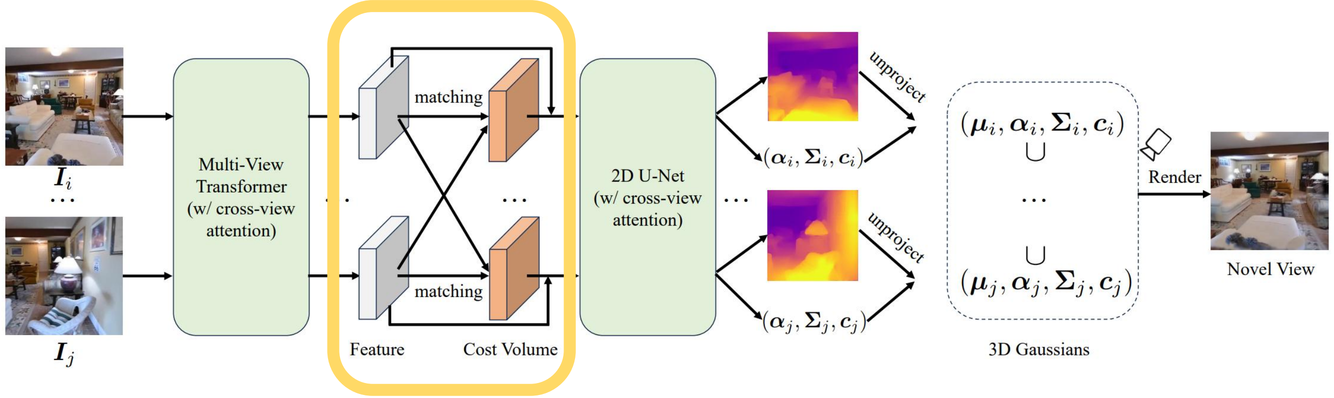
First, uniformly sample D depth candidates $\{d_m\}_{m=1}^{D}$ in the **inverse depth** domain.
(**D** = given near and far depth ranges)

Then, **warp** view ***j***'s feature $\boldsymbol{F}^j$ to view ***i*** with the **camera projection matrices** $\boldsymbol{P}^i$, $\boldsymbol{P}^j$ and each **depth candidate d_m** to obtain **D** warped features

$$\boldsymbol{F}_{d_m}^{j\to i} = \mathcal{W}(\boldsymbol{F}^j, \boldsymbol{P}^i, \boldsymbol{P}^j, d_m) \in \mathbb{R}^{\frac{H}{4}\times\frac{W}{4}\times C}, \quad m = 1, 2, \cdots, D,$$

***W** denotes the warping operation

Method



Cost Volume Construction

Then, compute the **dot product** between $\boldsymbol{F}^i$ and $\boldsymbol{F}_{d_m}^{j\to i}$ to obtain the correlation.

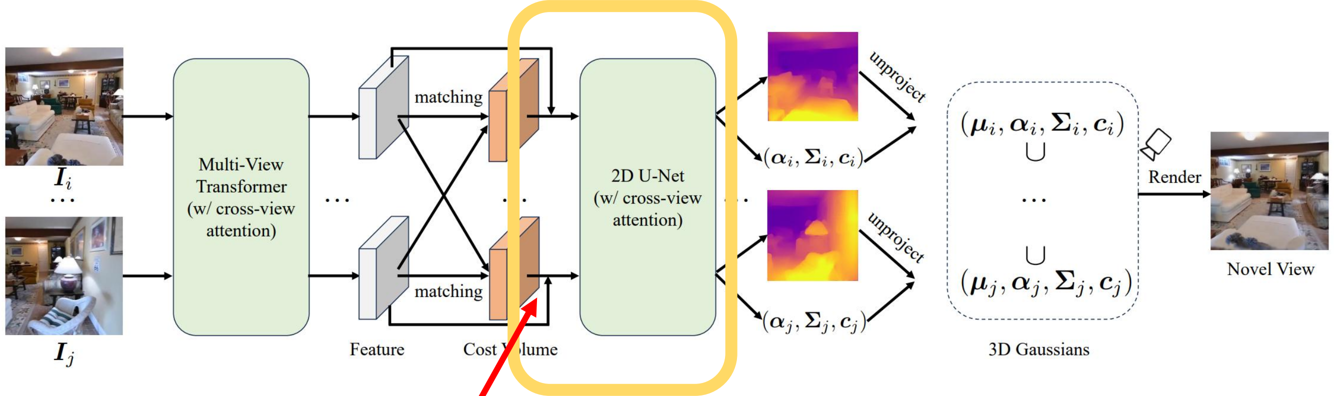
$$\boldsymbol{C}_{d_m}^i = \frac{\boldsymbol{F}^i \cdot \boldsymbol{F}_{d_m}^{j\to i}}{\sqrt{C}} \in \mathbb{R}^{\frac{H}{4}\times\frac{W}{4}}, \quad m = 1, 2, \cdots, D.$$

(When there are **more than two views** as inputs, we **similarly warp another view's feature to view i** and compute their correlations.)

Finally, all the correlations are **pixel-wise averaged.**(When there are **more than two views**)
Collecting **all the correlations** we obtain **view i's cost volume.**

$$\boldsymbol{C}^i = [\boldsymbol{C}_{d_1}^i, \boldsymbol{C}_{d_2}^i, \cdots, \boldsymbol{C}_{d_D}^i] \in \mathbb{R}^{\frac{H}{4}\times\frac{W}{4}\times D}.$$

Overall, <u>K</u> cost volumes $\{\boldsymbol{C}^i\}_{i=1}^K$ are obtained.



Method

Cost Volume Refinement

As the **cost volume** can be **ambiguous** for **texture-less regions**, **refine** it with a lightweight **2D U-Net**.

Input:

concatenation of Transformer features F^i and cost volume C^i

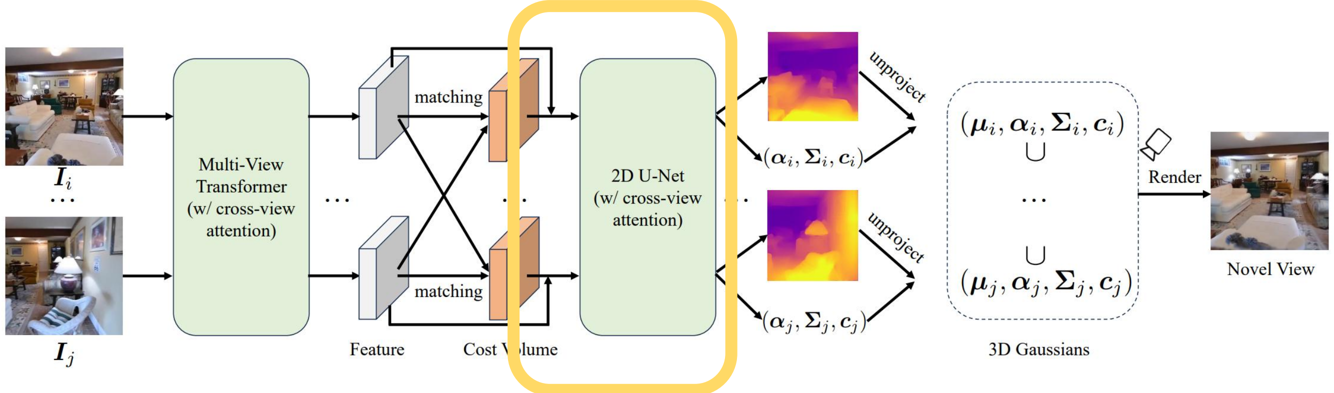
Output:

a residual $\Delta C^i \in \mathbb{R}^{\frac{H}{4} \times \frac{W}{4} \times D}$ that is **added** to the **initial cost volume** C^i

$$\tilde{C}^i = C^i + \Delta C^i \in \mathbb{R}^{\frac{H}{4} \times \frac{W}{4} \times D}.$$

To exchange information between cost volumes of different views, inject **cross-view attention layers.**

Method



Cost Volume Refinement

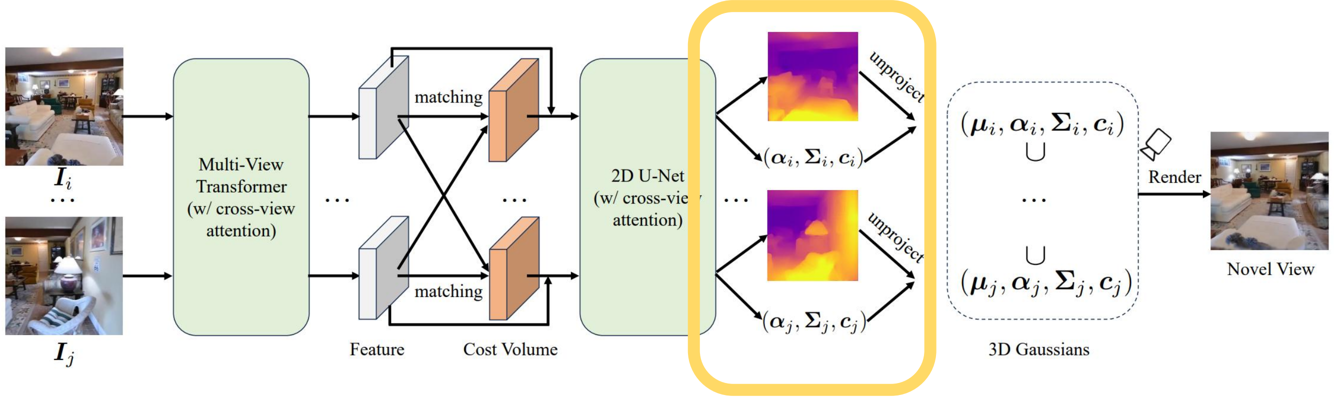
$$\tilde{C}^i = C^i + \Delta C^i \in \mathbb{R}^{\frac{H}{4} \times \frac{W}{4} \times D}.$$

To exchange information between cost volumes of different views, inject **cross-view attention layers.**

The **low-resolution cost volume** $\tilde{C}^i$ is finally **upsampled** to **full resolution** with a **CNN-based upsampler.**

Full resolution: $\hat{C}^i \in \mathbb{R}^{H \times W \times D}$

Method



Depth Estimation

Use the **softmax** operation to obtain **per-view depth predictions**.

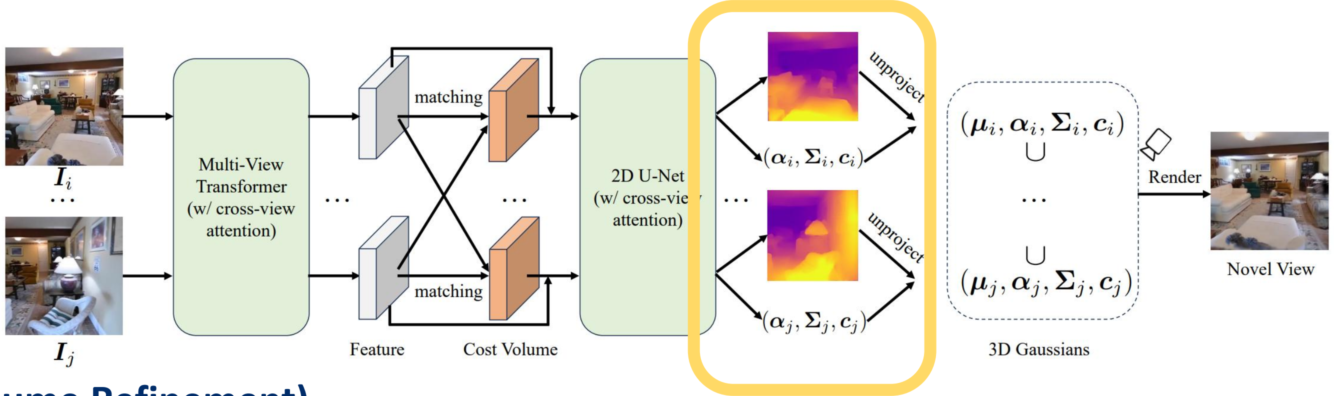
- First, **normalize** the refined **cost volume** $\hat{C}^i$ in the depth dimension .
- Then, perform a **weighted average** of all depth candidates $G = [d_1, d_2, \cdots, d_D] \in \mathbb{R}^D$

거리 후보 값들

$$V^i = \text{softmax}(\hat{C}^i)G \in \mathbb{R}^{H \times W}$$

확률을 반영한 거리

Method



Depth Refinement (almost same with Cost volume Refinement)

Refinement is performed with a very **lightweight 2D U-Net.**

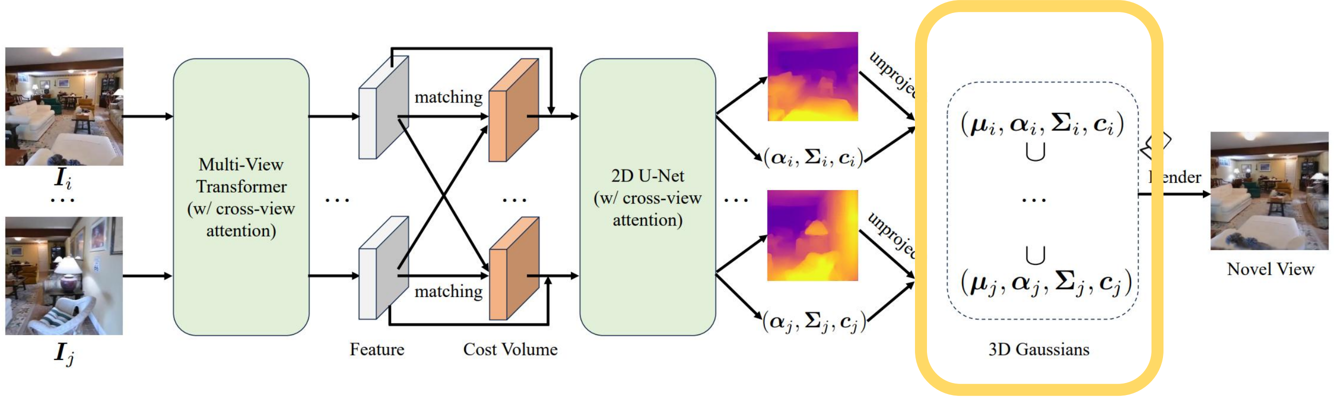
Input:
Multi-view images, features, and current depth predictions.

Output:
Per-view residual depths which are **added** to the **current depth predictions.**

$$D_{refined} = D + \Delta(D)$$

Also introduce **cross-view attention layers** in the **lowest resolution to exchange information across views**.

Method



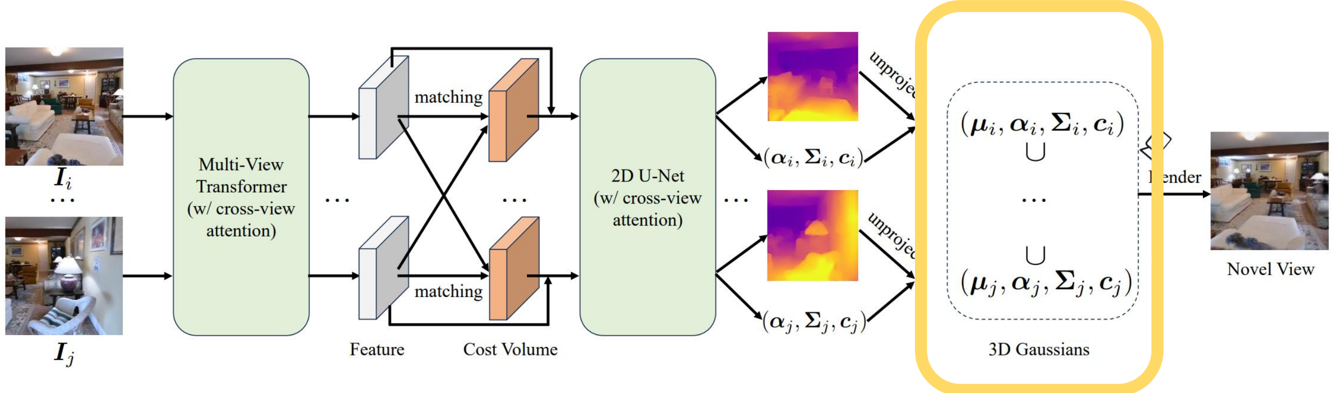
Gaussian parameters Prediction

predicts a set of 3D Gaussian parameters $\{(\boldsymbol{\mu}_j, \alpha_j, \boldsymbol{\Sigma}_j, \boldsymbol{c}_j)\}_{j=1}^{H \times W \times K}$

Gaussian centers μ:

- directly **unproject** the **multi-view depth predictions** to **3D point clouds** using the camera parameters
- Then, directly combine the **3D point clouds** as the **centers of the 3D Gaussians (μ)**

Method



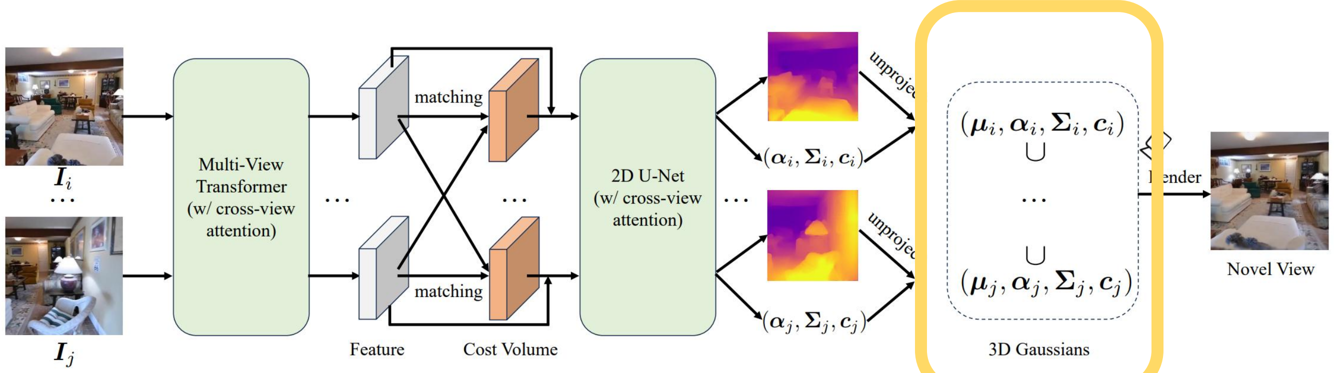
Gaussian parameters Prediction

predicts a set of 3D Gaussian parameters $\{(\boldsymbol{\mu}_j, \alpha_j, \boldsymbol{\Sigma}_j, \boldsymbol{c}_j)\}_{j=1}^{H\times W\times K}$

Opacity α:

- can obtain the **matching confidence** as the **maximum value** of the **softmax output**.
- **matching confidence** shares a similar **physical meaning** with the **opacity** (points with higher matching confidence are more likely to be on the surface)
- thus use **two convolution layers** to predict the **opacity** from the **matching confidence input**.

Method



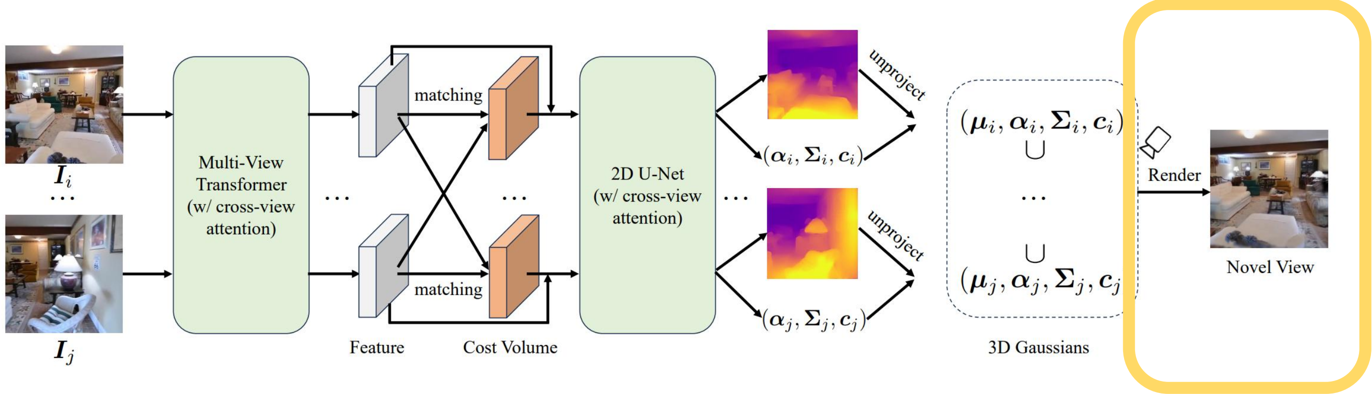
Gaussian parameters Prediction

predicts a set of 3D Gaussian parameters $\{(\boldsymbol{\mu}_j, \alpha_j, \boldsymbol{\Sigma}_j, \boldsymbol{c}_j)\}_{j=1}^{H \times W \times K}$

Covariance Σ and color c:

- predict these parameters using two convolution layers (!= opacity's conv layers)
- **Inputs:**
    - **concatenated image features, refined cost volume, original multi-view images.**

Method



Training Loss

- is trained with ground truth target **RGB** images as **supervision**.

- **training loss** is calculated as a **linear combination of ℓ2(1) and LPIPS(0.05)**

***LPIPS Loss : RGB 값으로 비교하는 것이 아니라, VGG나 AlexNet 같은 것을 거쳐서 나온 Feature Map를 가지고 비교하는 Loss**

Experiment

Evaluate the method on **novel view synthesis** from image pairs

Datasets

RealEstate10k : a dataset of **home walkthrough videos** downloaded from YouTube
ACID : a dataset of **aerial landscape videos** downloaded from YouTube

RealEstate10k

ACID

Experiment

Quantitative Comparison

Method	Time (s)	Param (M)	RealEstate10K [54]			ACID [21]		
			PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
pixelNeRF [49]	5.299	28.2	20.43	0.589	0.550	20.97	0.547	0.533
GPNR [35]	13.340	9.6	24.11	0.793	0.255	25.28	0.764	0.332
AttnRend [10]	1.325	125.1	24.78	0.820	0.213	26.88	0.799	0.218
MuRF [44]	0.186	**5.3**	26.10	0.858	0.143	28.09	0.841	0.155
pixelSplat [1]	0.104	125.4	25.89	0.858	0.142	28.14	0.839	0.150
MVSplat	**0.044**	12.0	**26.39**	**0.869**	**0.128**	**28.25**	**0.843**	**0.144**

Table 1: Comparisons with the state of the art. Running time includes both encoder and render, note that 3DGS-based methods (pixelSplat and MVSplat) render dramatically faster (∼ 500FPS for the render). Performances are averaged over thousands of test scenes in each dataset. For each scene, the model takes two views as input and renders three novel views for evaluation. MVSplat performs the best in terms of all visual metrics and runs the fastest with a lightweight model size.

Experiment

Qualitative Comparison

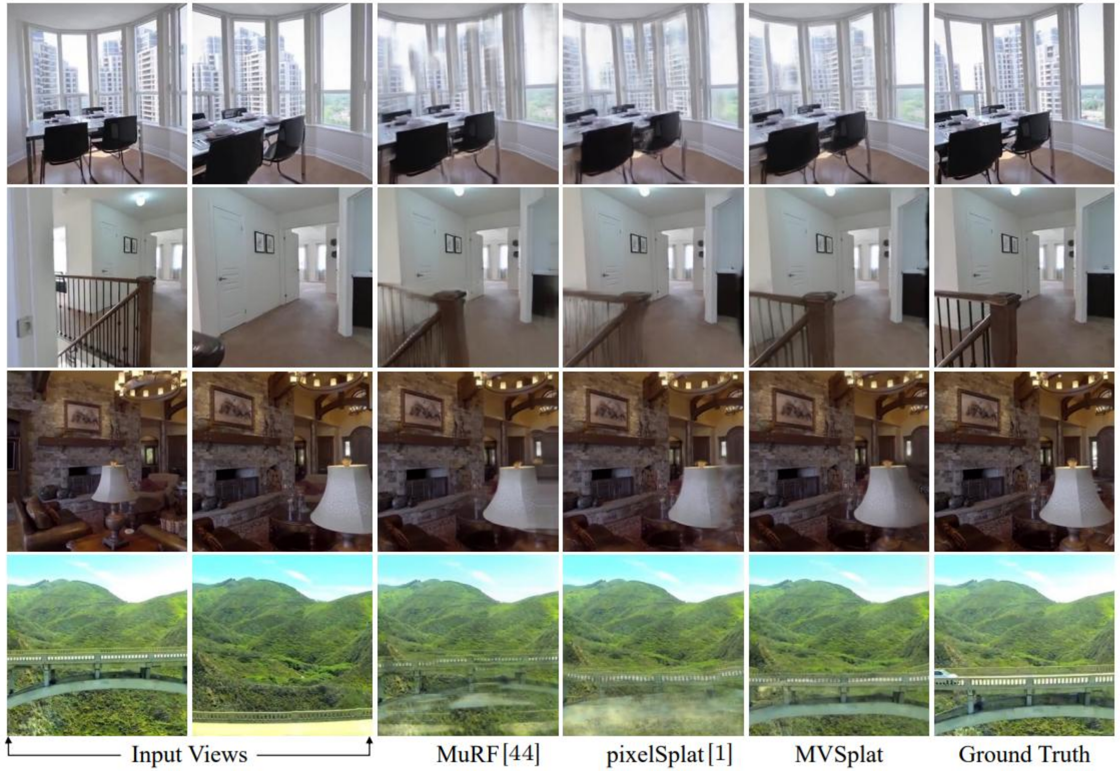


Fig. 3: Comparisons with the state of the art. The first three rows are from RealEstate10K (indoor scenes), while the last one is from ACID (outdoor scenes). Models are trained with a collection of training scenes from each indicated dataset, and tested on novel scenes from the same dataset. MVSplat surpasses all other competitive models in rendering challenging regions due to the effectiveness of our cost volume-based geometry representation.

Experiment

Geometry Quality Comparison

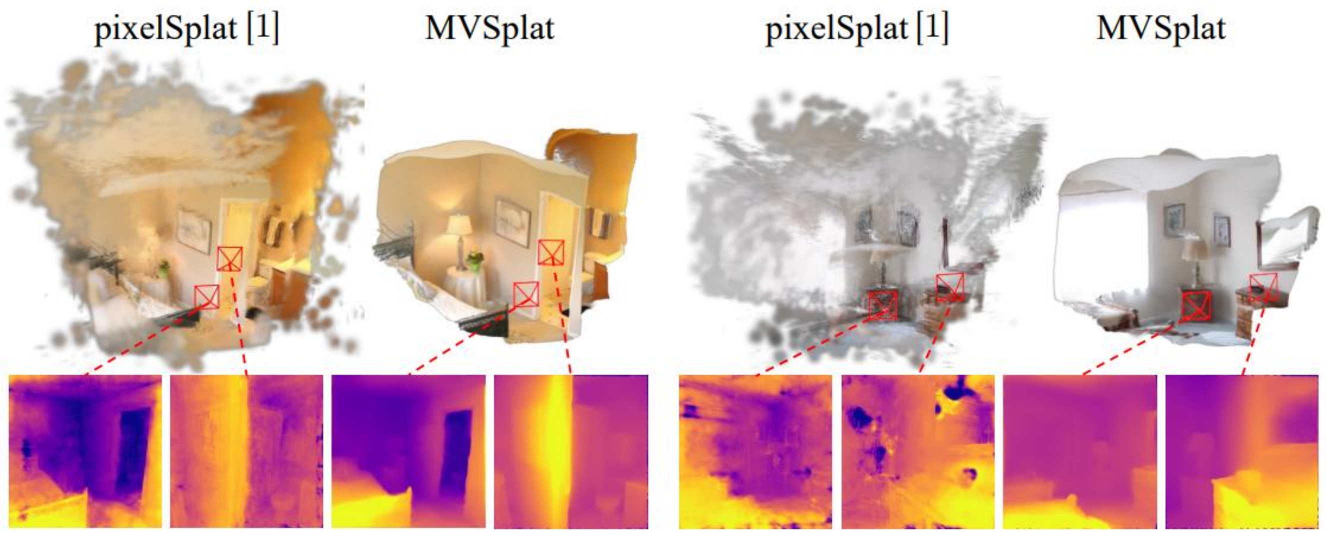


Fig. 4: Comparisons of 3D Gaussians (top) and depth maps (bottom). We compare the reconstructed geometry quality by visualizing zoom-out views of 3D Gaussians predicted by pixelSplat and our MVSplat, along with the predicted depth maps of two reference views. Extra fine-tuning is *not* performed on either model. Unlike pixelSplat that contains obvious floating artifacts, our MVSplat produces much higher quality 3D Gaussians and smoother depth, demonstrating the effectiveness of our cost volume-based 3D representation.

Experiment

Zero-Shot Test

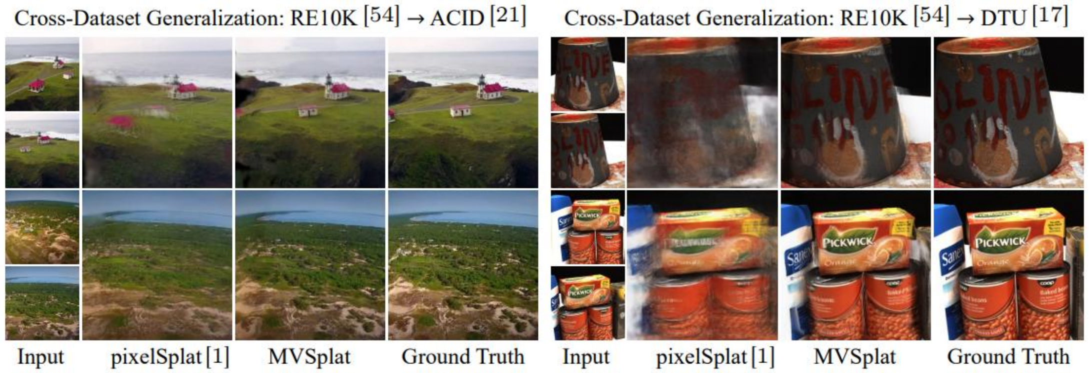


Fig. 5: Cross-dataset generalization. Models trained on the source dataset RealEstate10K (indoor scenes) are used to conduct zero-shot test on scenes from target datasets ACID (outdoor scenes) and DTU (object-centric scenes), without any fine-tuning. pixelSplat tends to render blurry images with obvious artifacts since feature distributions in the target datasets differ from the one in the source, while our MVSplat renders competitive outputs thanks to the feature-invariant cost volume based design.

Training data	Method	ACID [21]			DTU [17]		
		PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
RealEstate10K [54]	pixelSplat [1]	27.64	0.830	0.160	12.89	0.382	0.560
	MVSplat	**28.15**	**0.841**	**0.147**	**13.94**	**0.473**	**0.385**

Table 2: Cross-dataset generalization. Models trained on RE10K (indoor scenes) are directly used to test on scenes from ACID (outdoor scenes) and DTU (object-centric scenes), without any further fine-tuning. Our MVSplat generalizes better than pixelSplat, where the improvement is more significant when the gap between source and target datasets is larger (RE10K to DTU). It is also worth noting that our zero-shot generalization results on ACID even slightly surpass pixelSplat's ACID trained model (PSNR: 28.14, SSIM: 0.843, LPIPS: 0.144) reported in Tab. 1.

Quantitative Comparison

Qualitative Comparison

Experiment

Ablation Study

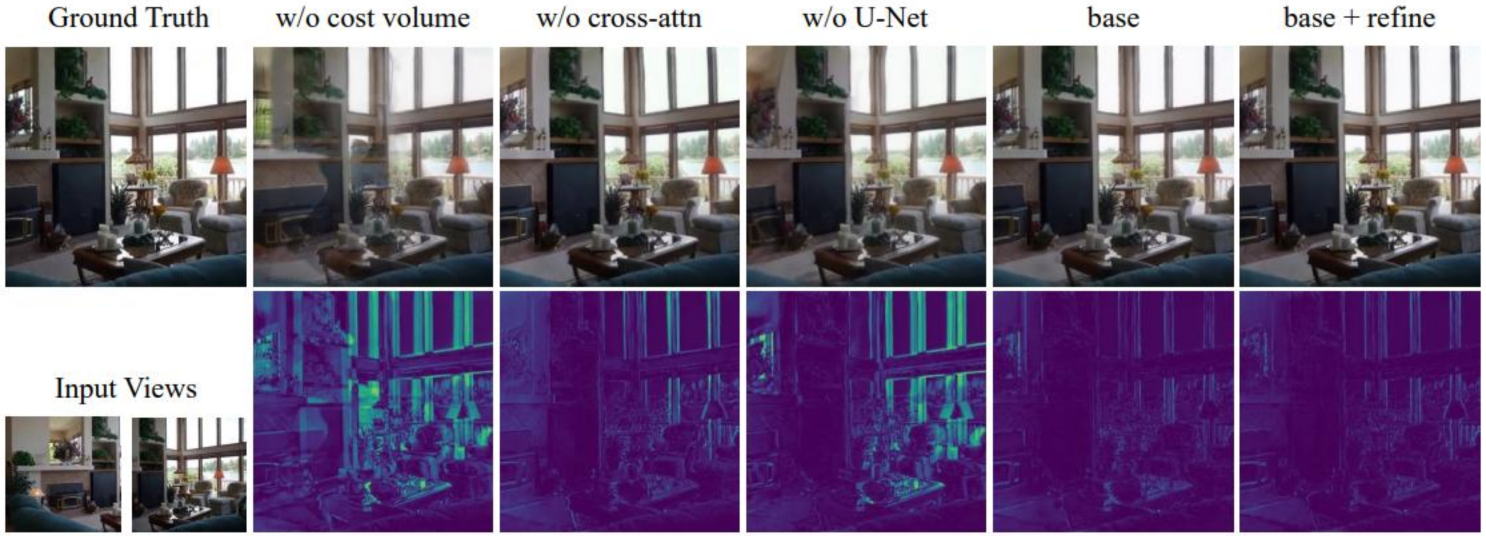
Setup	PSNR↑	SSIM↑	LPIPS↓
base + refine	**26.39**	**0.869**	**0.128**
base	26.12	0.864	0.133
w/o cost volume	22.83	0.753	0.197
w/o cross-view attention	25.19	0.852	0.152
w/o U-Net	25.45	0.847	0.150

Table 3: Ablations on RealEstate10K. The "base + refine" is our final model, where "refine" refers to the "depth refinement" detailed in Sec. 3.1. All other ablations are conducted on the "base" model w/o depth refinement. The cost volume module plays an indispensable role in MVSplat. Results of all models are obtained from the final converged step (300K in our experiments), except the one "w/o cross-view attention", which suffers from over-fitting (details are shown in the supplementary material Fig. A), hence we report its best performance.

*indispensable: 없어서는 안될

Experiment

Ablation Study



Colored error maps

Fig. 6: Ablations on RealEstate10K. Colored *error maps* obtained by calculating the differences between the rendered images and the ground truth are attached for better comparison. All models are based on the "base" model, w/o depth refinement module. "w/o cross-attn" is short for "w/o cross-view attention". Compared to "base", "base+refine" slightly reduces errors, "w/o cost volume" leads to the largest drop, "w/o U-Net" harms contents on the right that only exist in one input view, while "w/o cross-attn" increases the overall error intensity.

Experiment

Transformer: Epipolar vs Swin

Setup	Time (s)	PSNR↑	SSIM↑	LPIPS↓
MVSplat (w/ Epipolar Transformer [1])	0.055	26.09	0.865	0.133
MVSplat (w/ Swin Transformer)	**0.038**	26.12	0.864	0.133

Table B: Comparisons of the backbone Transformer. Our Swin Transformer [23]-based architecture is more efficient than the Epipolar Transformer counterpart in pixelSplat [1] since the expensive epipolar sampling process is avoided. Besides, there is no clear difference observed in their rendering qualities.

Conclusion

Contribution

- Proposes an efficient feed-forward 3D Gaussian Splatting model.
- Exploit multi-view correspondence information by using cost volume.
- State-of-the-art in two large-scale scene-level reconstruction.
- 10× fewer parameters and infers more than 2× faster.

Conclusion

Limitation

- Challenges with Non-Lambertian Surfaces.

  *Non-Lambertian Surfaces: 관찰하는 각도에 따라 그 밝기나 색상이 변하는 모든 표면
  ➔ 고차원 Spherical Harmonics 계수를 정확히 추론하는 데 어려움을 겪는다.

- Dependency on Accurate Camera Poses.

- Computational complexity increases with the number of input views.