CVPR 2021 (Oral)

NeRF in the Wild

Presenter: Seongjun Choi

sjchoi.dp@gmail.com

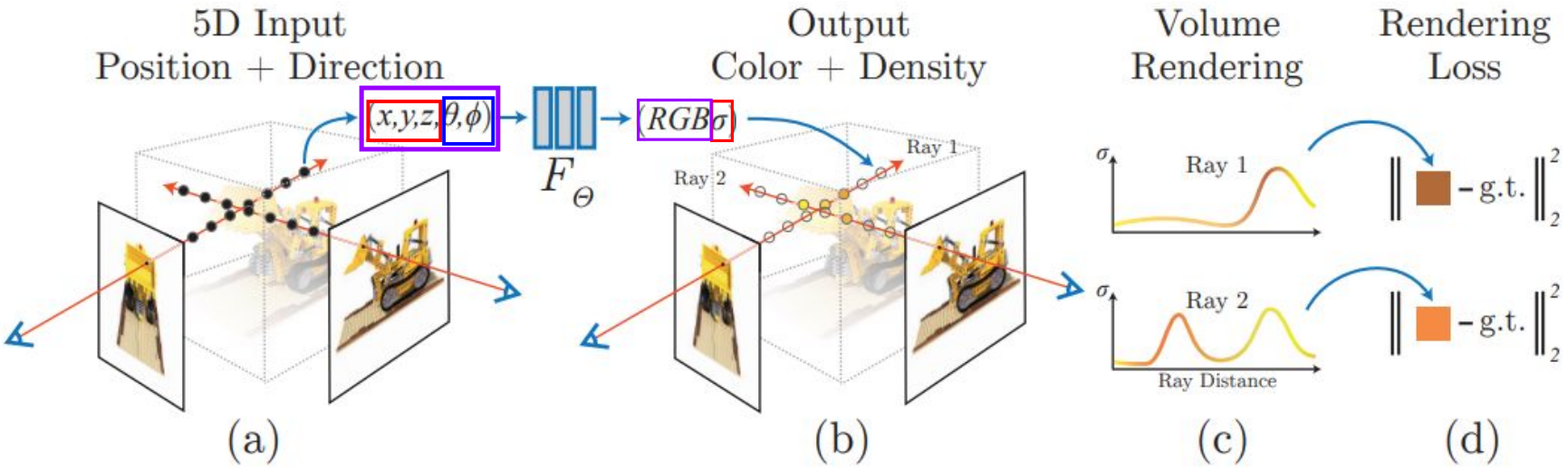
Contents

Introduction

Problem: 관광지에서 여러 사람이 촬영한 사진을 조합하여, Novel View Syntesis 가능할까?

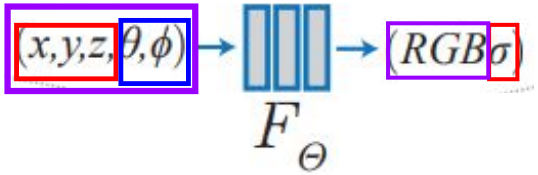
- Dataset: 다양한 사진들(명소)을 Flickr에서 얻어보자 (e.g. 독일 Brandenburg Gate)

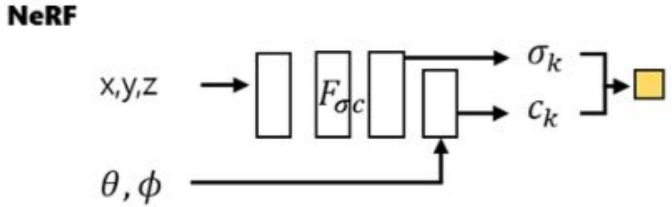
Background: NeRF



Mildenhall, Ben, et al. "Nerf: Representing scenes as neural radiance fields for view synthesis." ECCV 2020.

Background: NeRF





$$\mathbb{R}^5 \rightarrow \mathbb{R}^3$$

3D Position + **View Direction** → **Color**

$$\mathbb{R}^3 \rightarrow \mathbb{R}$$

3D Position → **Density**

Volume Rendering

$$\hat{C}(r) = \sum_{k=1}^{K} T(t_k)\,\alpha(\sigma(t_k)\delta_k)\,c(t_k)$$

$$\text{where} \quad T(t_k) = \exp\left(-\sum_{k'=1} \sigma(t_{k'})\delta_{k'}\right)$$

- 빨강: k번째 point가 surface일 정도
- 파랑: k번째 point의 radiance
- 보라: 1~k-1번째 point들의 투명한 정도

Mildenhall, Ben, et al. "Nerf: Representing scenes as neural radiance fields for view synthesis." ECCV 2020.

NeRF in the Wild (NeRF-W)

NeRF와 다른 점이 무엇일까? 사진 상의 동적 물체는 어떻게 없앨 수 있었지?

- Static한 네트워크와 Transient한 네트워크를 분리하고,
- radiance와 density에 대한 Uncertainty를 측정하자!

이를 위한 Contribution Point?

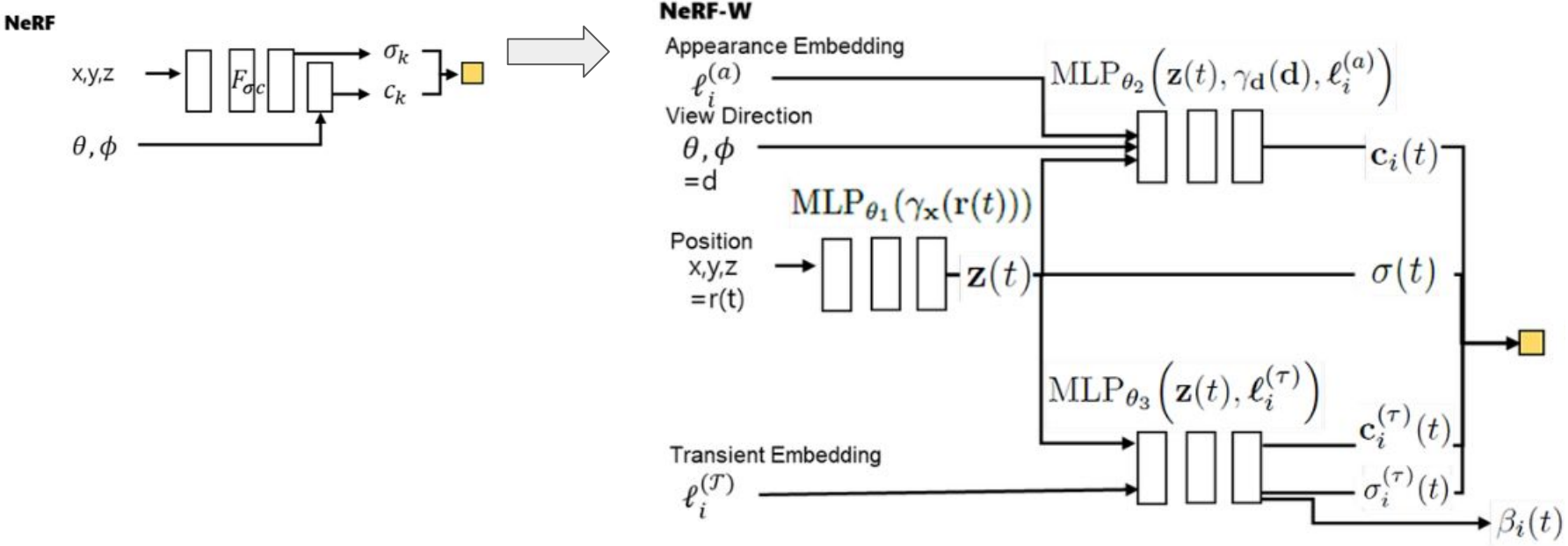
- Latent Appearance Modeling
- Transient Objects
- Optimization

결과는?

- Appearance 변화
- 일시적으로 찍힌 물체를 제거

NeRF in the Wild

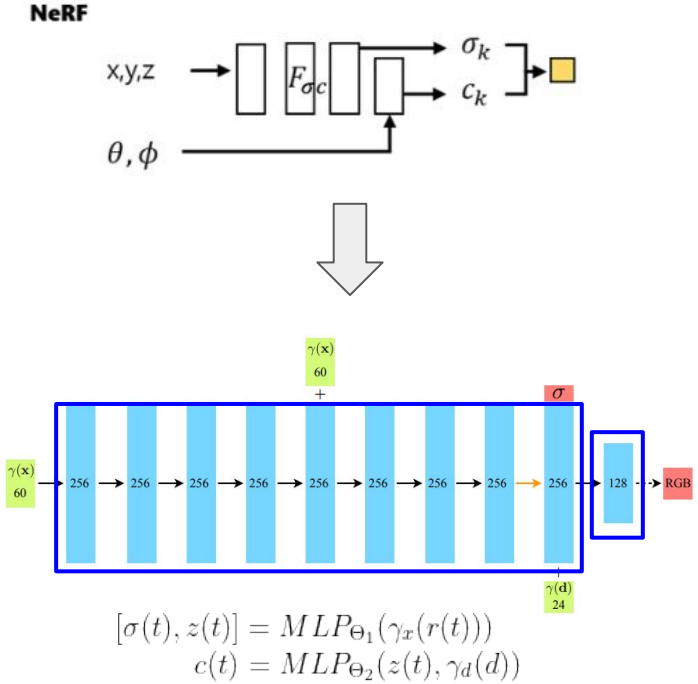
Architecture: NeRF vs. NeRF-W



[논문 리뷰] NeRF-W (CVPR 2021) : Uncertainty 접목 연구: https://xoft.tistory.com/17

NeRF in the Wild

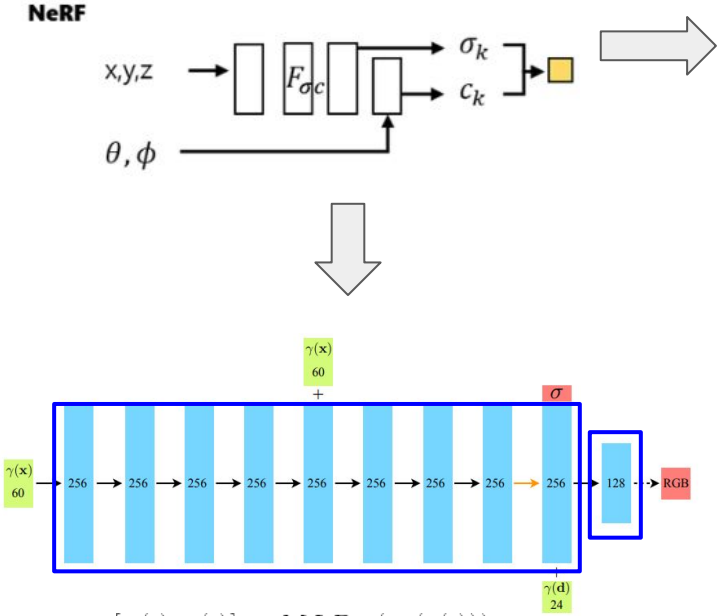
Architecture: NeRF vs. NeRF-W

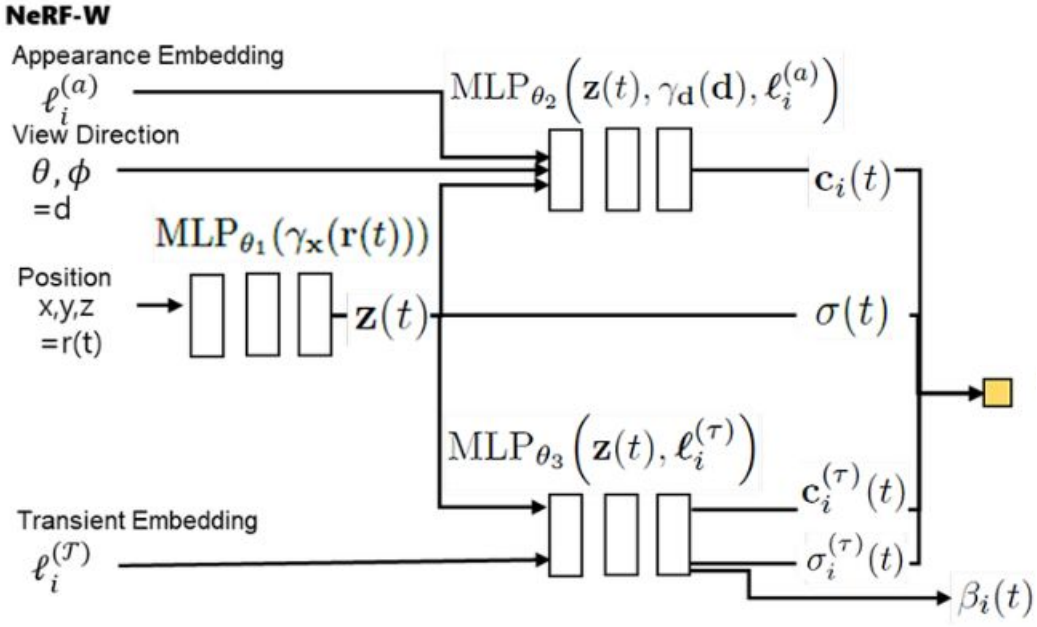


$$[\sigma(t), z(t)] = MLP_{\Theta_1}(\gamma_x(r(t)))$$
$$c(t) = MLP_{\Theta_2}(z(t), \gamma_d(d))$$

NeRF in the Wild

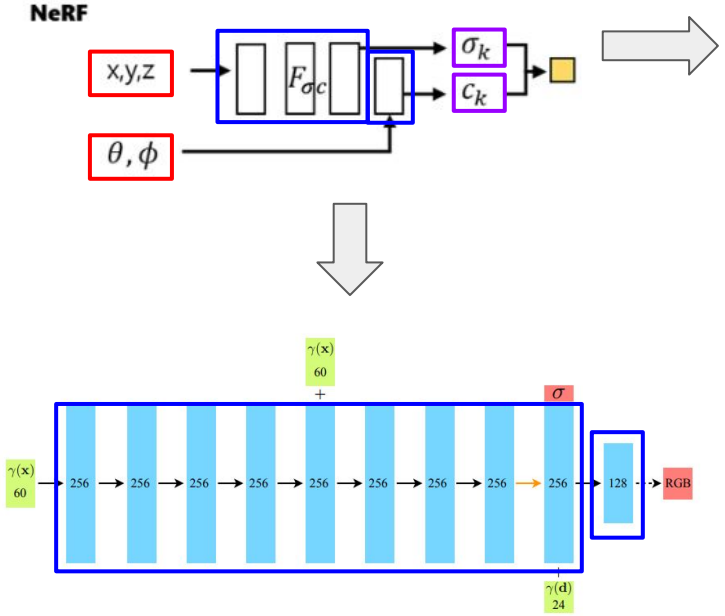
Architecture: NeRF vs. NeRF-W



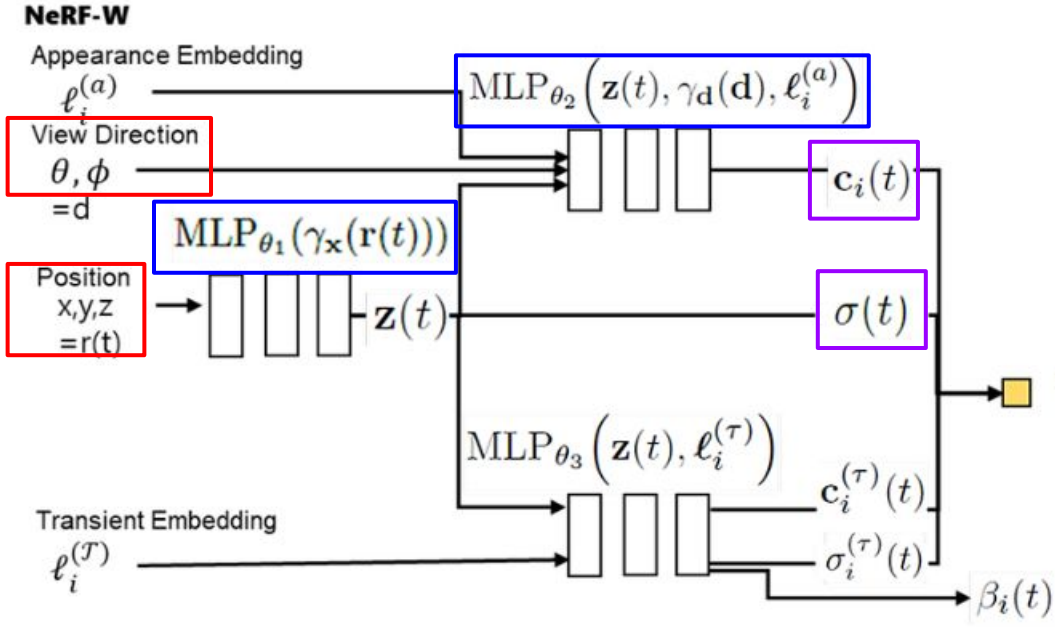


NeRF in the Wild

Architecture: NeRF vs. NeRF-W



$$[\sigma(t), z(t)] = MLP_{\Theta_1}(\gamma_x(r(t)))$$
$$c(t) = MLP_{\Theta_2}(z(t), \gamma_d(d))$$



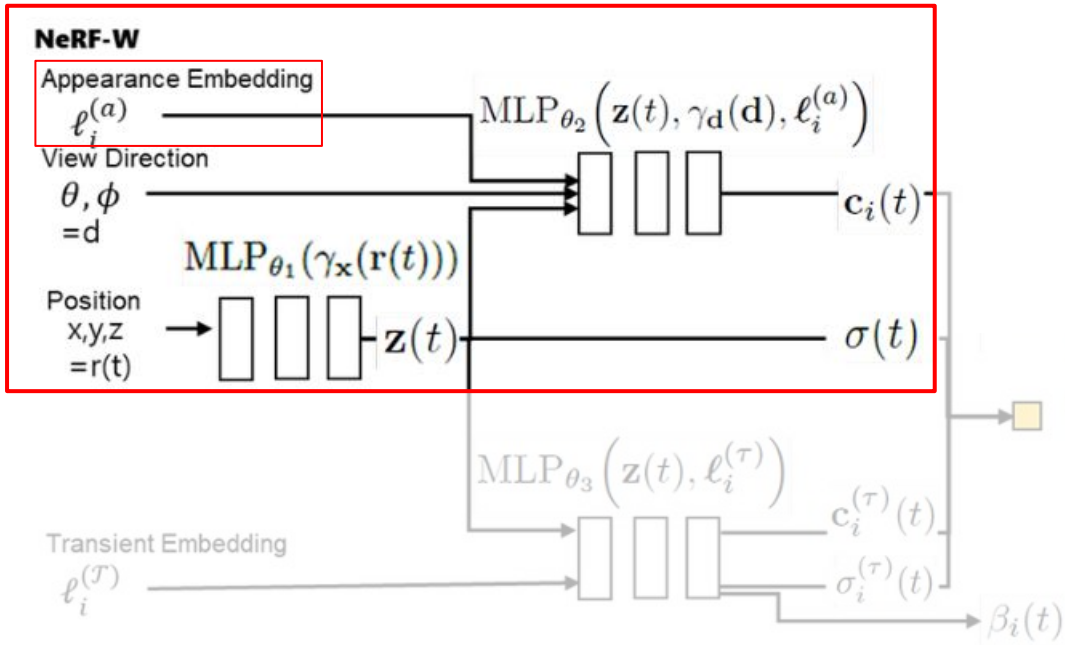
[논문 리뷰] NeRF-W (CVPR 2021) : Uncertainty 접목 연구: https://xoft.tistory.com/17

Latent Appearance Modeling

$$[\sigma(t), \mathbf{z}(t)] = \mathrm{MLP}_{\theta_1}(\gamma_{\mathbf{x}}(\mathbf{r}(t))), \quad (3)$$
$$\mathbf{c}(t) = \mathrm{MLP}_{\theta_2}(\mathbf{z}(t), \gamma_{\mathbf{d}}(\mathbf{d})), \quad (4)$$

Architecture: Static Network

γ_d : view direction positional encoding
γ_x : position positional encoding



Appearance Embedding?

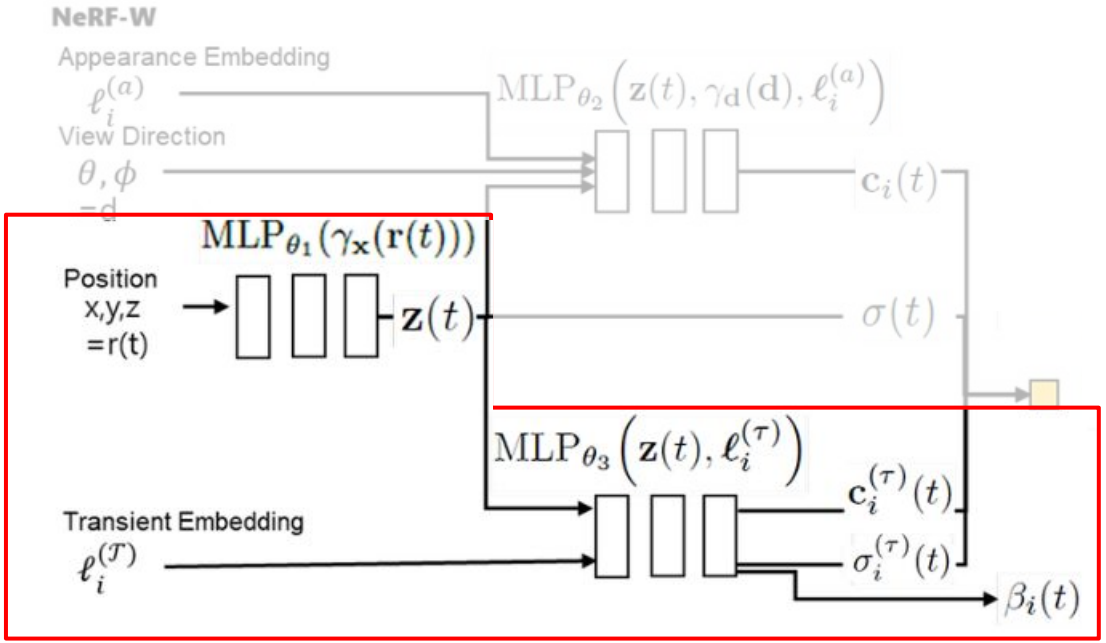
- Embedding Vector로, Random 초기화된 후 MLP를 통해 학습.
- 이미지의 Embedding Vector
- Embedding Vector를 수정하며 Appearance 조정 가능 (스타일 조정)
- 학습 데이터셋에 대해서 학습했기에, 테스트시 타겟 이미지에 적당한 것 선택


Transient Objects

$$\left[\sigma_i^{(\tau)}(t), \mathbf{c}_i^{(\tau)}(t), \tilde{\beta}_i(t)\right] = \mathrm{MLP}_{\theta_3}\left(\mathbf{z}(t), \boldsymbol{\ell}_i^{(\tau)}\right), \quad (11)$$

$$\beta_i(t) = \beta_{\min} + \log\left(1 + \exp\left(\tilde{\beta}_i(t)\right)\right), \quad (12)$$

Architecture: Transient Network



Transient

β_i : uncertainty

Transient Objects?

타겟 물체를 찍을때 가리고 있던 물체

- MLP1은 3D shape 정보를 담고 있음
- 여기에 Transient Embedding 를 넣어서 Transient한 density를 뽑아 제거 가능.

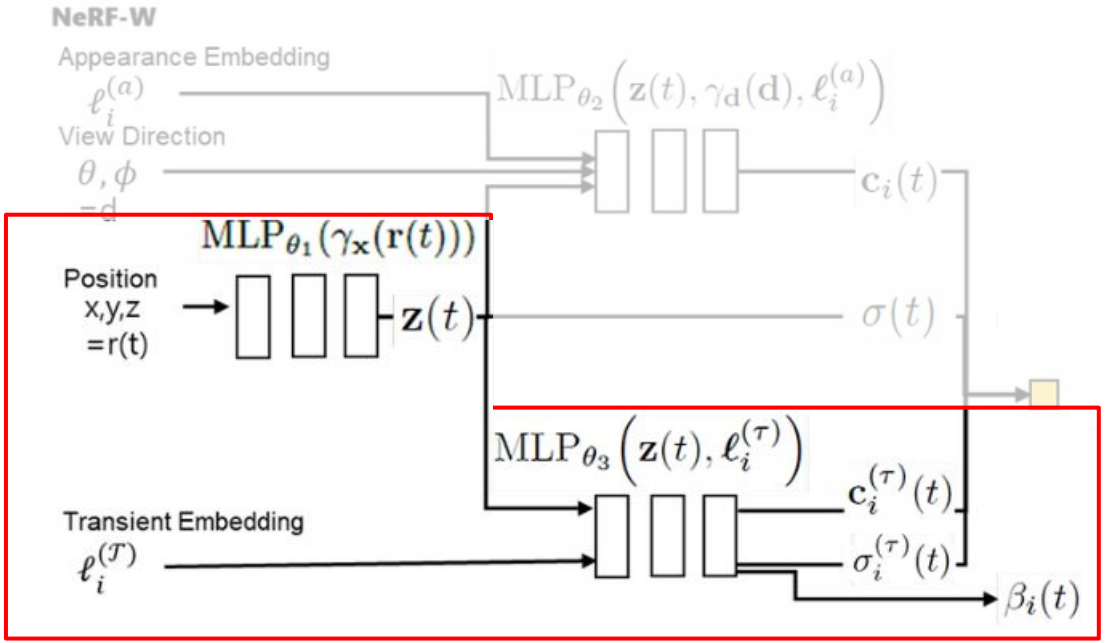
(a) Static (b) Transient (c) Composite

Transient Objects

$$\left[\sigma_i^{(\tau)}(t), \mathbf{c}_i^{(\tau)}(t), \tilde{\beta}_i(t)\right] = \mathrm{MLP}_{\theta_3}\left(\mathbf{z}(t), \boldsymbol{\ell}_i^{(\tau)}\right), \quad (11)$$

$$\beta_i(t) = \beta_{\min} + \log\left(1 + \exp\left(\tilde{\beta}_i(t)\right)\right), \quad (12)$$

Architecture: Transient Network



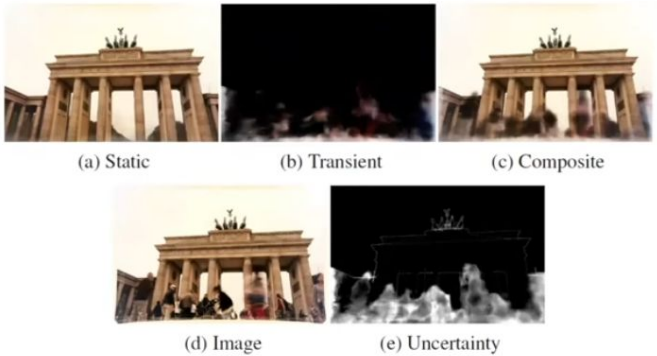
Transient

β_i : uncertainty

Uncertainty?

$$\hat{\mathbf{C}}_i(\mathbf{r}) = \mathcal{R}(\mathbf{r}, \mathbf{c}_i, \sigma), \Rightarrow \hat{\beta}_i(\mathbf{r}) = \mathcal{R}(\mathbf{r}, \beta_i, \sigma_i^{(\tau)}).$$

color 로 pixel 값을 렌더링 하듯,
uncertainty를 렌더링 하는 개념.



Volume Rendering

NeRF

$$\hat{\mathbf{C}}(\mathbf{r}) = \mathcal{R}(\mathbf{r}, \mathbf{c}, \sigma) = \sum_{k=1}^{K} T(t_k)\,\alpha(\sigma(t_k)\delta_k)\,\mathbf{c}(t_k), \quad (1)$$

$$\text{where} \quad T(t_k) = \exp\left(-\sum_{k'=1}^{k-1} \sigma(t_{k'})\delta_{k'}\right), \quad (2)$$

NeRF-W

$$\hat{\mathbf{C}}_i(\mathbf{r}) = \sum_{k=1}^{K} T_i(t_k)\left(\alpha(\sigma(t_k)\delta_k)\mathbf{c}_i(t_k) + \alpha\left(\sigma_i^{(\tau)}(t_k)\delta_k\right)\mathbf{c}_i^{(\tau)}(t_k)\right)$$

$$\text{where } T_i(t_k) = \exp\left(-\sum_{k'=1}^{k-1}\left(\sigma(t_{k'}) + \sigma_i^{(\tau)}(t_{k'})\right)\delta_{k'}\right)$$

- 빨강: k번째 point가 surface일 정도
- 파랑: k번째 point의 radiance
- 보라: 1~k-1번째 point들의 투명한 정도

Optimization

NeRF

$$\mathcal{L} = \sum_{\mathbf{r}\in\mathcal{R}} \left[\left\| \hat{C}_c(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 + \left\| \hat{C}_f(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 \right]$$

Coarse *Fine*

NeRF-W

$$\mathcal{L} = \sum_{ij} L_i(\mathbf{r}_{ij}) + \frac{1}{2} \left\| \mathbf{C}(\mathbf{r}_{ij}) - \hat{\mathbf{C}}_i^c(\mathbf{r}_{ij}) \right\|_2^2,$$

Fine *Coarse*

$$L_i(\mathbf{r}) = \frac{\left\| \mathbf{C}_i(\mathbf{r}) - \hat{\mathbf{C}}_i(\mathbf{r}) \right\|_2^2}{2\beta_i(\mathbf{r})^2} + \frac{\log \beta_i(\mathbf{r})^2}{2} + \frac{\lambda_u}{K} \sum_{k=1}^{K} \sigma_i^{(\tau)}(t_k).$$

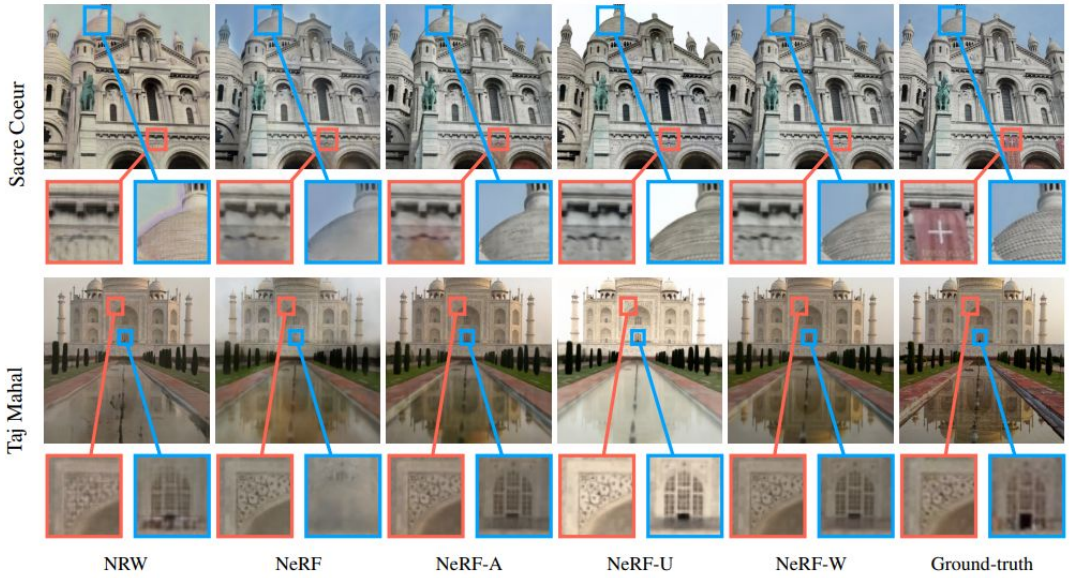
NLL Loss: uncertainty *L1 Regularizer: transient*

Optimization

- COLMAP을 이용하여 Camera Pose 추정
- 총 300,000회 반복, batch size 2048, 8개 Nvidia V100 GPU 이용

Experimental & Results

	Brandenburg Gate			Sacre Coeur			Trevi Fountain			Taj Mahal			Prague			Hagia Sophia		
	PSNR	MS-SSIM	LPIPS	PSNR	MS-SSIM	LPIPS	PSNR	MS-SSIM	LPIPS	PSNR	MS-SSIM	LPIPS	PSNR	MS-SSIM	LPIPS	PSNR	MS-SSIM	LPIPS
NRW [22]	23.85	0.914	0.141	19.39	0.797	0.229	20.56	0.811	0.242	21.24	0.844	**0.201**	19.89	0.803	**0.216**	20.75	0.796	**0.231**
NeRF	21.05	0.895	0.208	17.12	0.781	0.278	17.46	0.778	0.334	15.77	0.697	0.427	15.67	0.747	0.362	16.04	0.749	0.338
NeRF-A	27.96	0.941	0.145	24.43	0.923	0.174	26.24	0.924	0.211	25.99	0.893	0.225	22.52	0.870	0.244	21.83	0.820	0.276
NeRF-U	19.49	0.921	0.174	15.99	0.826	0.223	15.03	0.795	0.277	10.23	0.778	0.373	15.03	0.787	0.315	13.74	0.706	0.376
NeRF-W	**29.08**	**0.962**	**0.110**	**25.34**	**0.939**	**0.151**	**26.58**	**0.934**	**0.189**	**26.36**	**0.904**	0.207	**22.81**	**0.879**	0.227	**22.23**	**0.849**	0.250



(a) Reference (b) NeRF (c) NeRF-W

Figure 6: Depth maps from NeRF and NeRF-W, rendered by computing the expected termination depth of each ray. NeRF's geometry is corrupted by appearance variation and occluders, while NeRF-W is robust to such phenomena and produces accurate 3D reconstructions. Photos by Flickr users burkeandhare, photogreuhphies / CC BY.

Conclusion

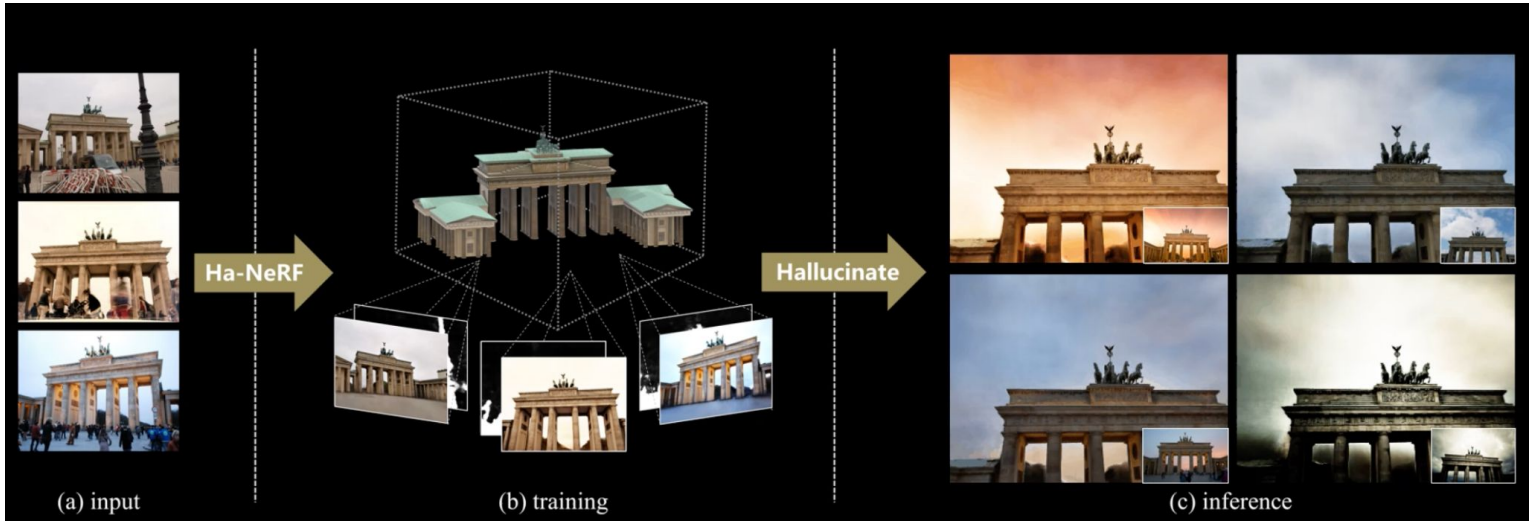
- 동적이고 조명이 바뀌는 상황에서 촬영된 이미지로 Neural Rendering 구현
- 이를 위해 각 이미지의 광도를 반영하는 Appearence Embedding 기법과, 타겟 물체를 가리고 있던 물체를 구별하는 transient network 제안

Weak

- 컴퓨터 리소스 (Scene 한 개당 8개의 Nvidia V100 GPU 이틀 학습)

Ha-NeRF

Hallucinated Neural Radiance Fields in the Wild



Introduction: Realistic NeRF

- Realistic NeRF를 얻기 위한 문제점
  - 이미지의 시간, 날짜에 따라 피사체가 변화할 수 있음
  - 다양한 물체들에 의해 렌더링할 피사체가 가려짐

- unseen appearance에 대한 이미지 렌더링 불가능 (NeRF, NeRF-W)

Introduction: NeRF-W

- **appearance embedding**을 최적화 -> varing appearance 문제 해결
- transient head 추가 -> input image의 static component와 transient component를 구분
- 기존 모델에 비해 더 realistic 이미지를 렌더링

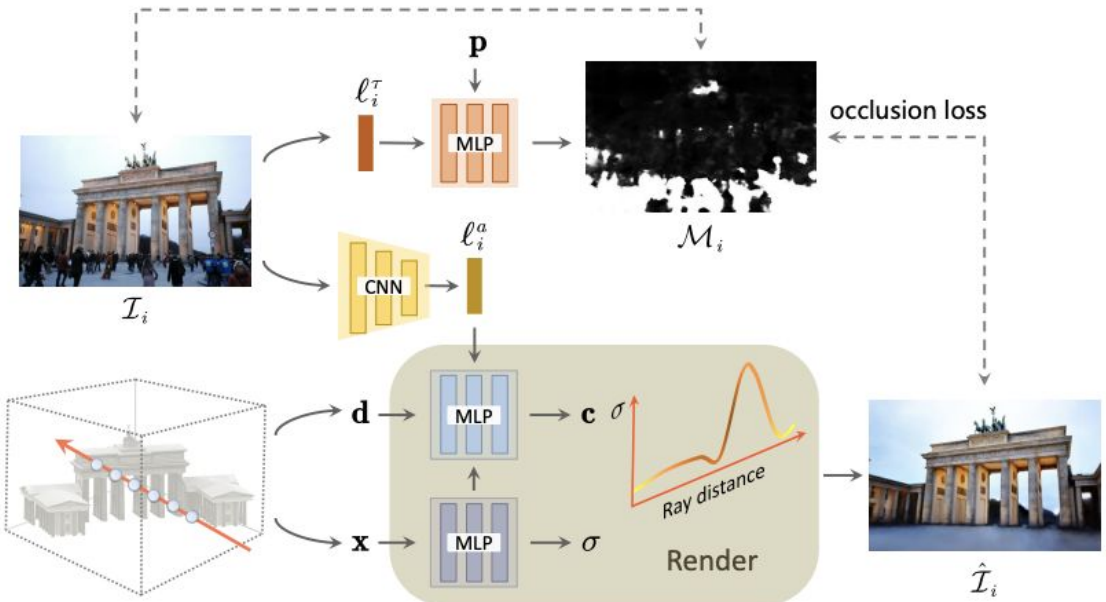
limitations

- 다른 데이터셋 or 새로운 이미지에 대한 hallucination 시에, appearance embedding 추가로 최적화 필요
- transient embedding이 occluder의 랜덤성에 취약함

Ha-NeRF: structure

1) image를 CNN으로 encoding
-> output: appearance latent vector

2) rays로부터 sampling한 ①location, ②viewing direction, ③appearance embedding을 MLP에 feed
-> output: color와 volume density
(임베딩 벡터 제외 NeRF와 동일)

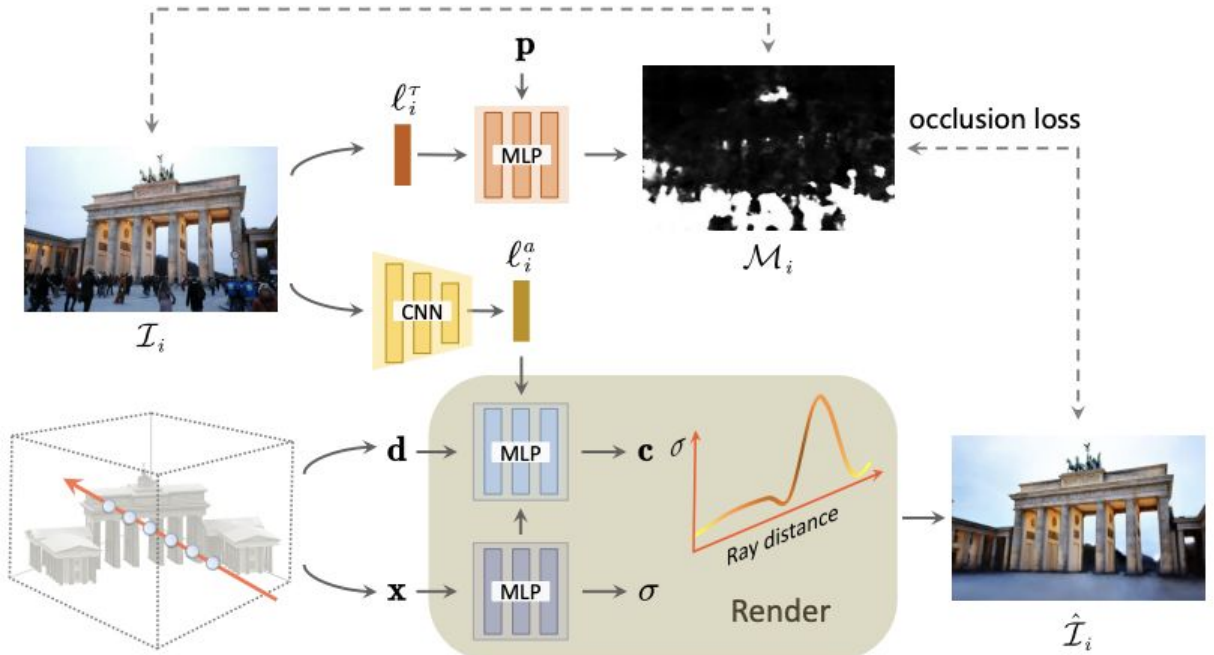


Ha-NeRF: structure

3) image reconstruction

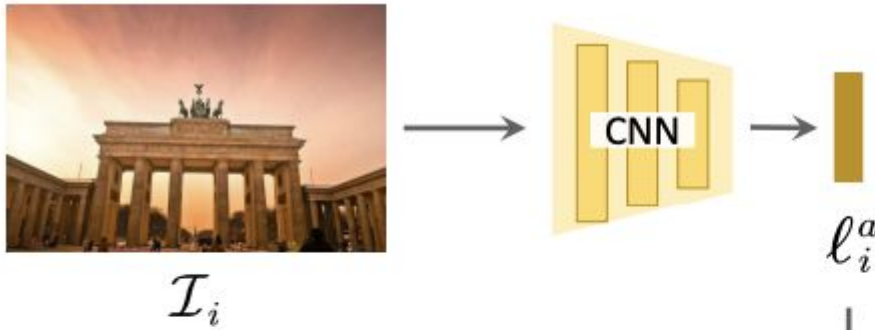
4) transient embedding과 image의 pixel location을 MLP에 feed

-> output: visible possibility mask



Ha-NeRF: View-consistent Hallucination

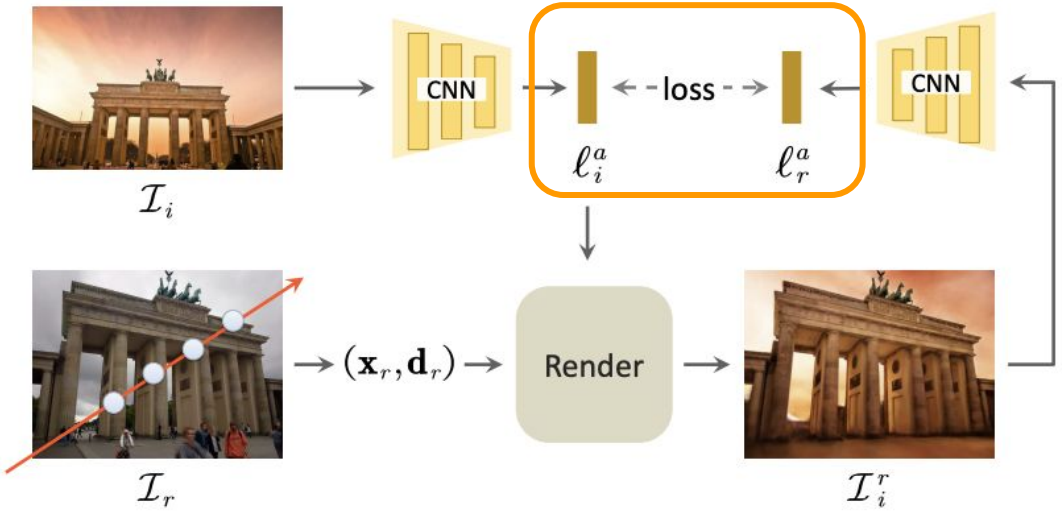
- CNN 인코더를 사용하여 appearance 학습
- varing illumination과 photometric postprocessing 부분 학습
- NeRF-W에서의 추가 최적화 문제 해결



Ha-NeRF: View-consistent Loss

- example image에 대해 encoding
- rays의 다른 view에서 샘플링하여 랜더링한 이미지를 인코딩
- 두 벡터를 L1 loss로 최적화

$$\mathcal{L}_{\mathrm{v}} = \|\mathrm{E}_{\phi}(\mathcal{I}_i^r) - \ell_i^a\|_1 ,$$

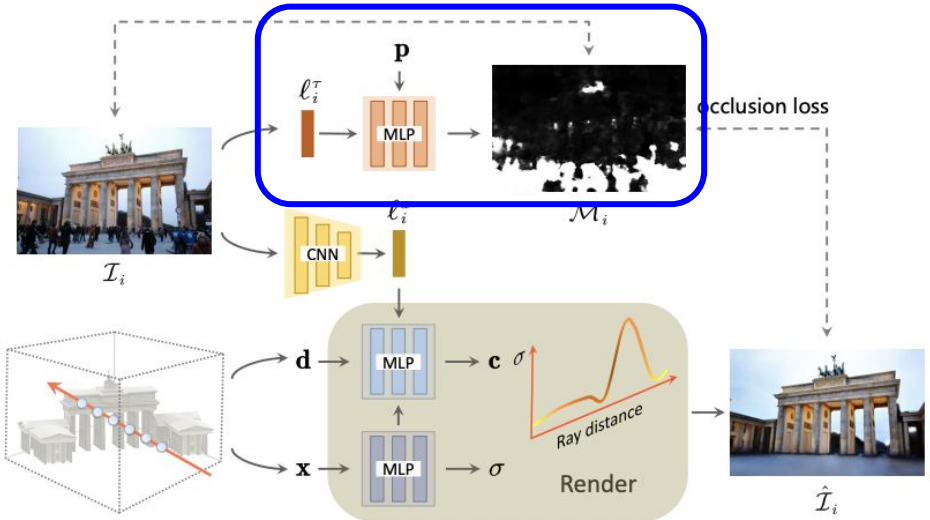


Ha-NeRF: View-consistent Hallucination

- view에 관계 없는 일정한 appearance rendering
- image geometry content를 추가로 인코딩할 필요 X
- 효율성을 위해, rays의 grid를 미리 샘플링한 후 사용

Ha-NeRF: Occlusion Handling

- NeRF-W의 3D transinet field를 사용하는 대신, 2D visibility map을 적용
- static scene과 transient objects의 분리를 더 정확하게 함
- 2D pixel location과 transient embedding을 mask로 매핑



Ha-NeRF: Occlusion Handling

```
w_samples, h_samples = torch.meshgrid([torch.linspace(0, 1-1/img_w, int(img_w)), \
                                       torch.linspace(0, 1-1/img_h, int(img_h))])

uv_sample = torch.cat((h_samples.permute(1, 0).contiguous().view(-1,1), \
                       w_samples.permute(1, 0).contiguous().view(-1,1)), -1)
```

```
tensor([[0.0000, 0.0000],
        [0.0000, 0.1000],
        [0.0000, 0.2000],
        [0.0000, 0.3000],
        [0.0000, 0.4000],
        [0.0000, 0.5000],
        [0.0000, 0.6000],
        [0.0000, 0.7000],
        [0.0000, 0.8000],
        [0.0000, 0.9000],
        [0.0500, 0.0000],
        [0.0500, 0.1000],
        [0.0500, 0.2000],
        [0.0500, 0.3000],
        [0.0500, 0.4000],
        [0.0500, 0.5000],
        [0.0500, 0.6000],
        [0.0500, 0.7000],
        [0.0500, 0.8000],
        [0.0500, 0.9000],
        [0.1000, 0.0000]
```

- 2D pixel location: image tensor가 갖는 픽셀의 위치 표현
- transient embedding: 임베딩 벡터로 설정하여 추가적으로 최적화

Ha-NeRF: Occlusion Handling

$$\mathcal{L}_{\mathrm{o}} = \mathcal{M}_{ij} \left\| \mathbf{C}\left(\mathbf{r}_{ij}\right) - \hat{\mathbf{C}}\left(\mathbf{r}_{ij}\right) \right\|_2^2 + \lambda_o (1 - \mathcal{M}_{ij})^2.$$

- first term: reconstruction error
  - rendered 이미지와 gt 이미지 간의 pixel visibility 계산
- second term: regularizer
  - static components를 무시하는 것 방지

- M의 값이 클수록 pixel의 중요도 올라감

Ha-NeRF: Optimization

$$\mathcal{L} = \lambda \sum_{i} \mathcal{L}_{\mathrm{v}} + \sum_{ij} \mathcal{L}_{\mathrm{o}}.$$

- trainable parameters:
  - θ: rendering model (MLP)
  - ϕ: CNN encoder
  - ψ: visibility model
  - transient embedding

Experiments: Qualitative

- NeRF:
  - ghosting effect + global color shifts
- NeRF-W:
  - fog effect (blur) in "Brandenburg Gate"
    - 3D transient field를 추정하다가 발생
  - different appearances:
    - sunshine, blue sky in "Sacre Coeur"
    - light reflection in "Trevi Fountain"

Experiments: Qualitative

- Ha-NeRF(A):
  - (+): 더 일관성 있는 appearnce [i.e. blue sky in "Sacre Coeur"]
  - (-): occlusion에 의해, high-frequency 디테일 X
- Ha-NeRF(T):
  - (+): occlusion에 robust [i.e. the square in "Brandenburg Gate"]
  - (-): varing photometric effects를 모델링 X
- Ha-NeRF: 두 ablation의 이점 모두 가짐..!

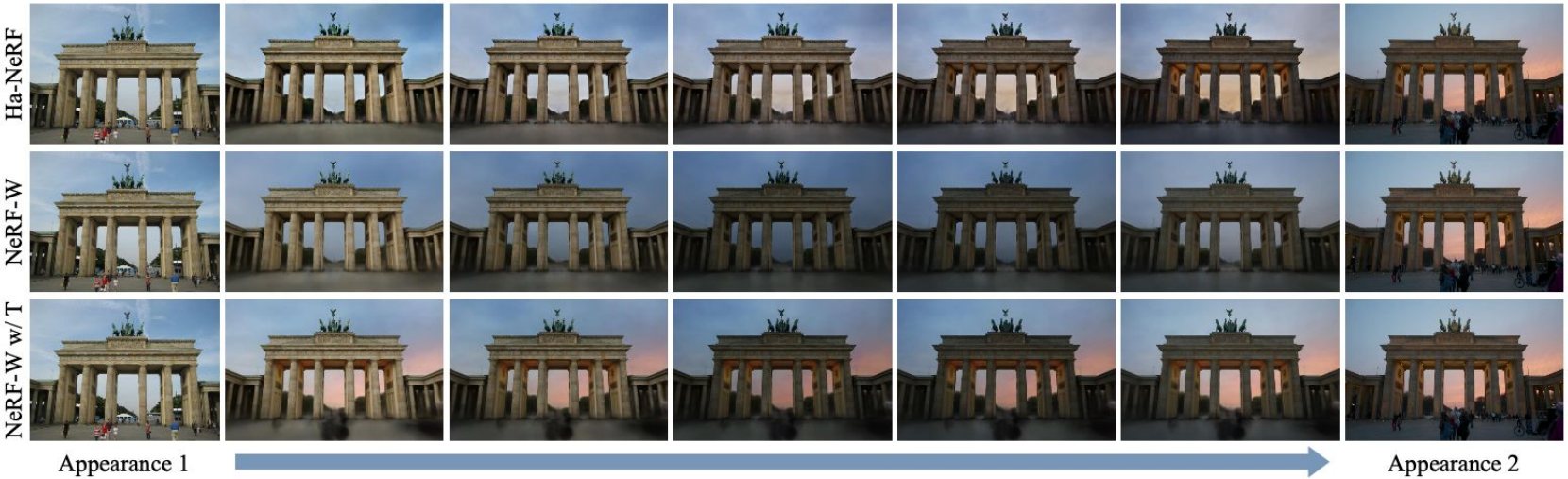
Experiments: Quantitative

	Brandenburg Gate			Sacre Coeur			Trevi Fountain		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF [33]	18.90	.8159	.2316	15.60	.7155	.2916	16.14	.6007	.3662
NeRF-W [28]*	24.17	.8905	.1670	19.20	.8076	.1915	18.97	.6984	.2652
Ha-NeRF(A)	22.93	.8517	.1727	19.57	.7864	.1839	19.89	.6798	.2377
Ha-NeRF(T)	19.84	.8368	.1835	16.66	.7657	.2267	15.92	.6186	.2830
Ha-NeRF	24.04	.8773	.1391	20.02	.8012	.1710	20.18	.6908	.2225

- PSNR과 SSIM은 NeRF-W와 비슷한 반면에, LPIPS에서는 outperform
- NeRF-W는 test images 절반을 이미 학습한 후 inference -> unfair

Experiments: Ha-NeRF vs NeRF-W

Experiments: Ha-NeRF vs NeRF-W

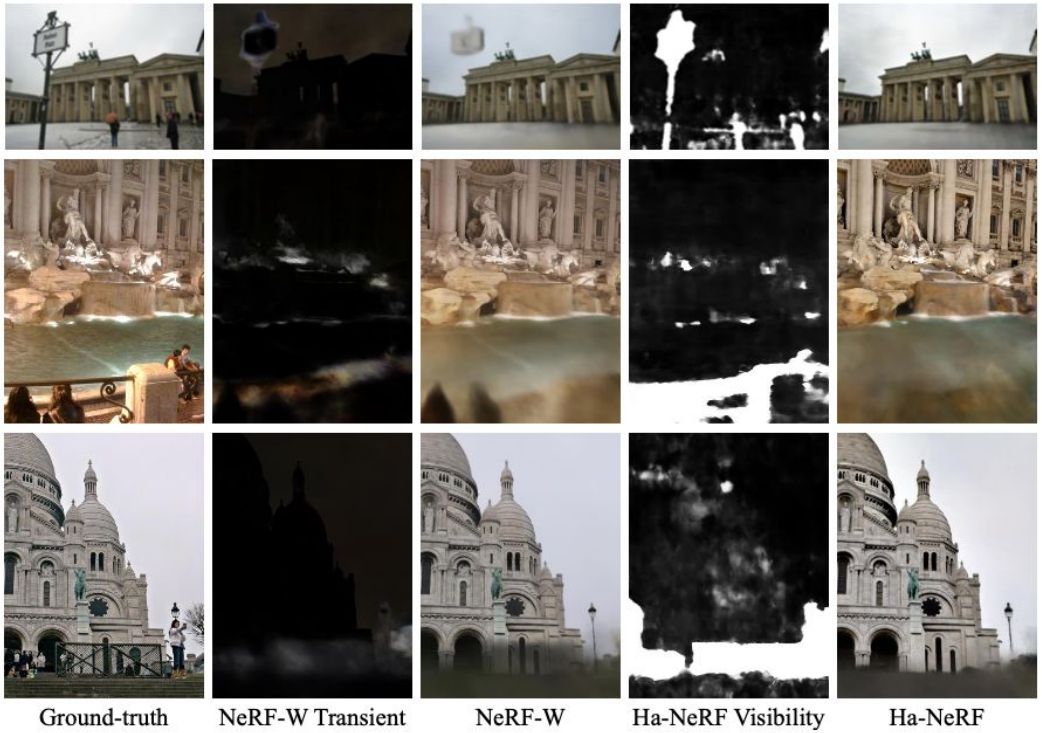


- appearance vector에 대한 interpolation 실험
- NeRF-W의 경우, interpolation 시 sunset glow 부분이 무시되었음
- NeRF-W가 transient 현상으로부터 변화하는 appearance를 구분하지 못함

Experiments: Ha-NeRF vs NeRF-W

- NeRF-W의 3D transient field와 Ha-NeRF의 visibility map 비교

- Ha-NeRF가 static scene과 transient objects를 더 잘 구분



Experiments: Limitations

- Ha-NeRF는 noisy camera extrinsic parameters로부터 안좋은 성능을 보임
- input images가 motion-blurred or defocused인 경우, 성능이 감소

Q&A

PseudoLab

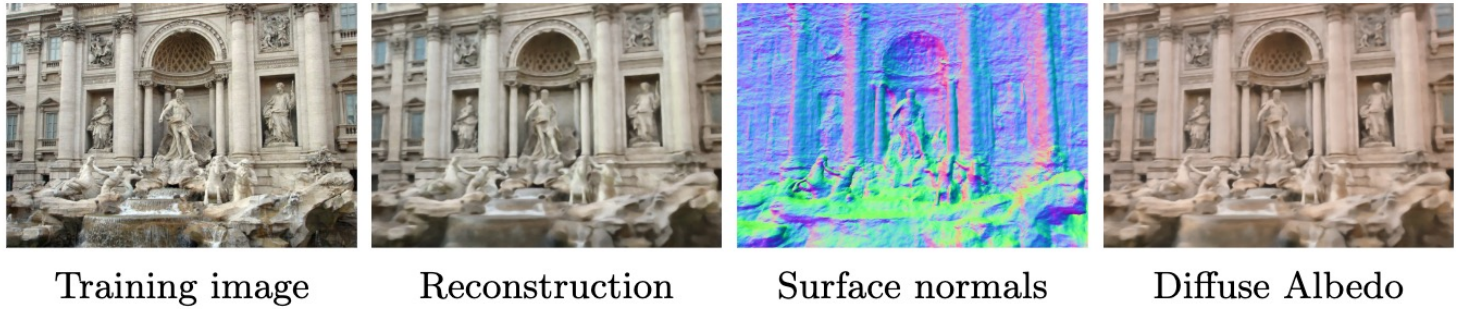
NeRF with Real world

- NeRF Outdoor Scene Relighting -

Wongi Park

https://github.com/r00tman/NeRF-OSR

Introduction



1. Decompose Intrinsic factors (Change Relight and Camera View)

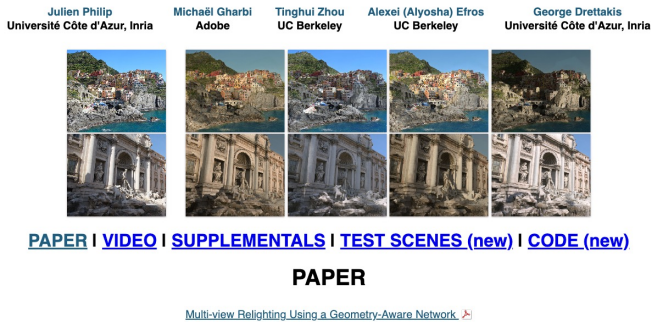
2. Proposed Dataset

Results

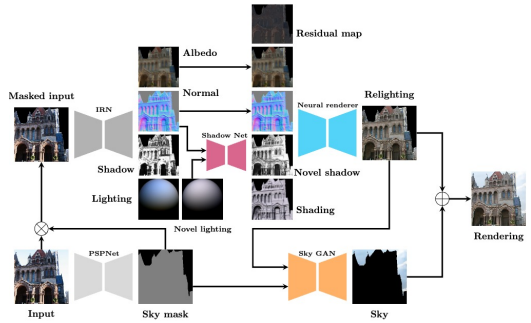
Related works

Scene Relighting

NeRF in the wild.

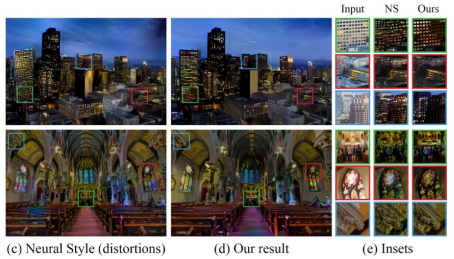


Multi-view Relighting Using a Geometry-Aware Network.



Self-supervised Outdoor Scene Relighting.

Style-based Editing



Deep Photo Style Transfer

Neural Rerendering in the Wild

1. Limitation by uncertainty factors

2. Not tackle intrinsic factors

3. Not considering Camera viewpoints **(Fixed illumination)**
   : light comes from fixed view points.

1. Lack explicit parametric control

2. Impossible decompose Intrinsic due to channel.

https://arxiv.org/pdf/2008.02268

Methods

1. Spherical Harmonics NeRF

Before (Existing methods)

$$\mathbf{C}(\mathbf{o},\mathbf{d}) = \mathbf{C}\left(\{\mathbf{x}_i\}_{i=1}^{N_{\text{depth}}}\right) = \sum_{i=1}^{N_{\text{depth}}} T(t_i)\alpha(\sigma(\mathbf{x}_i)\delta_i)\mathbf{c}(\mathbf{x}_i),$$

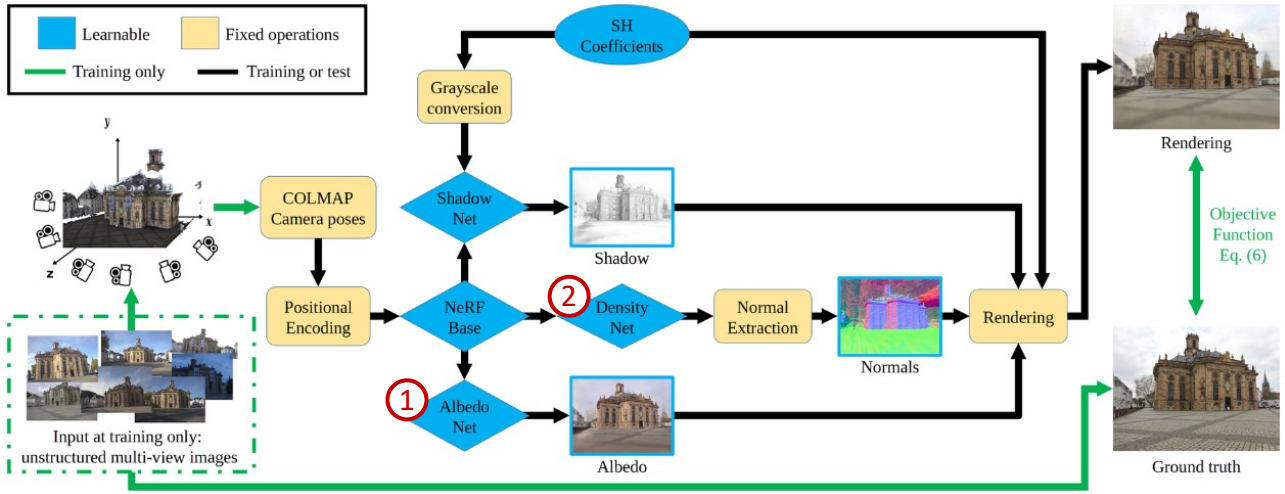
After (Ours)

Element wise multiplication

$$\mathbf{C}\left(\{\mathbf{x}_i\}_{i=1}^{N_{\text{depth}}}, \mathbf{L}\right) = \underbrace{\mathbf{A}}_{\text{Albedo}}\left(\{\mathbf{x}_i\}_{i=1}^{N_{\text{depth}}}\right) \odot \mathbf{Lb}\left(\mathbf{N}\left(\{\mathbf{x}_i\}_{i=1}^{N_{\text{depth}}}\right)\right)$$

$$\mathbf{N}\left(\{\mathbf{x}_i\}_{i=1}^{N_{\text{depth}}}\right) = \frac{\hat{\mathbf{N}}\left(\{\mathbf{x}_i\}_{i=1}^{N_{\text{depth}}}\right)}{\left\|\hat{\mathbf{N}}\left(\{\mathbf{x}_i\}_{i=1}^{N_{\text{depth}}}\right)\right\|^2},$$

$$\text{where } \hat{\mathbf{N}}\left(\{\mathbf{x}_i\}_{i=1}^{N_{\text{depth}}}\right) = \sum_{i=1}^{N_{\text{depth}}} \underbrace{\left(\frac{\partial}{\partial \mathbf{x}_i}\sigma(\mathbf{x}_i)\right)}_{\text{Role: Weight}} \odot T(t_i)\alpha(\sigma(\mathbf{x}_i)\delta_i).$$



NeRF-OSR Framework

Methods

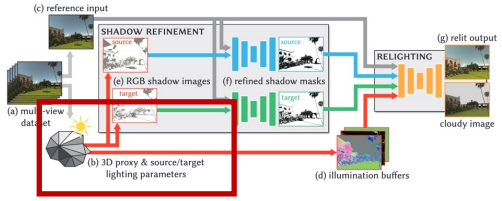
2. Shadow Generation Network

$$\mathbf{C}(\mathbf{o},\mathbf{d}) = \mathbf{C}\left(\{\mathbf{x}_i\}_{i=1}^{N_{\text{depth}}}\right) = \sum_{i=1}^{N_{\text{depth}}} T(t_i)\alpha(\sigma(\mathbf{x}_i)\delta_i)\mathbf{c}(\mathbf{x}_i),$$

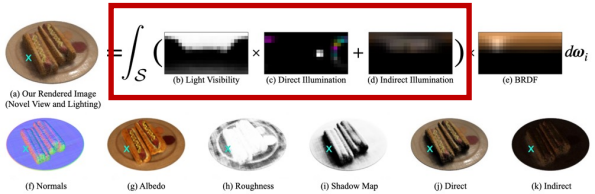
$$\mathbf{C}\left(\{\mathbf{x}_i\}_{i=1}^{N_{\text{depth}}}, \mathbf{L}\right) = \textcircled{3}\, S\left(\{\mathbf{x}_i\}_{i=1}^{N_{\text{depth}}}, \mathbf{L}\right) \mathbf{A}\left(\{\mathbf{x}_i\}_{i=1}^{N_{\text{depth}}}\right) \odot \mathbf{Lb}\left(\mathbf{N}\left(\{\mathbf{x}_i\}_{i=1}^{N_{\text{depth}}}\right)\right)$$

Shadow Generation

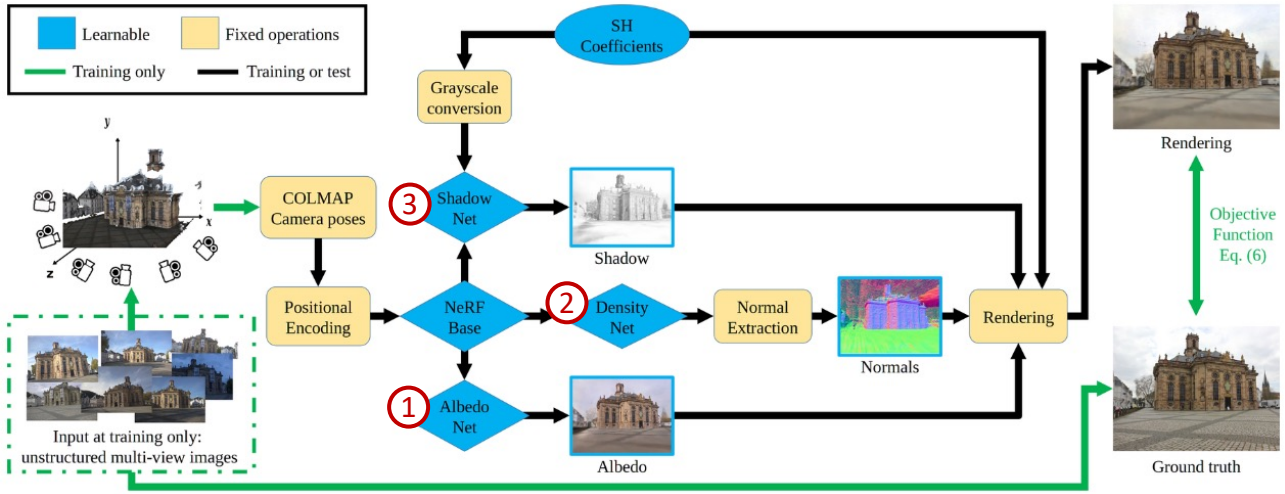
takes as input the SH coefficients in their grey-scale version, *i.e.*, $\mathbf{L} \in \mathbb{R}^{1\times 9}$ and not $\mathbb{R}^{3\times 9}$. This is motivated by the fact that shadows depend only on the



Multi-view Relighting using a Geometry-Aware Network



NeRV: Neural Reflectance and Visibility Fields for Relighting and View Synthesis



NeRF-OSR Framework

Methods

Total process

$$\mathbf{C}\left(\{\mathbf{x}_i\}_{i=1}^{N_{\text{depth}}}, \mathbf{L}\right) = S\left(\{\mathbf{x}_i\}_{i=1}^{N_{\text{depth}}}, \mathbf{L}\right) \mathbf{A}\left(\{\mathbf{x}_i\}_{i=1}^{N_{\text{depth}}}\right) \odot \mathbf{Lb}\left(\mathbf{N}\left(\{\mathbf{x}_i\}_{i=1}^{N_{\text{depth}}}\right)\right)$$

Self-supervised Approach

Shadow | Albedo | Normal

Loss

$$\mathcal{L}(\mathbf{C}, \mathbf{C}^{(\text{GT})}, S) = \text{MSE}(\mathbf{C}, \mathbf{C}^{(\text{GT})}) + \lambda \,\text{MSE}(S, 1),$$

Regularize Shadow Net

Frequency Annealing

$$\gamma_k(\mathbf{x})\text{: } \gamma'_k(\mathbf{x}) = \gamma_k(\mathbf{x})\beta_k(n)$$

$$\beta_k(n) = \tfrac{1}{2}(1 - \cos(\pi \text{clamp}(\alpha - i + N_{\text{fmin}}, 0, 1)))$$

$$\alpha(n) = (N_{\text{fmax}} - N_{\text{fmin}})\frac{n}{N_{\text{anneal}}}$$

12 | 8

* Prevent injecting noise into the rendering

the correct geometry. Hence, we alleviate this by using the annealing scheme slightly modified from Deformable NeRF [26], *i.e.*, we add an annealing coefficient $\beta_k(n)$ to each of the PE components $\gamma_k(\mathbf{x})$: $\gamma'_k(\mathbf{x}) = \gamma_k(\mathbf{x})\beta_k(n)$, where $\beta_k(n) = \frac{1}{2}(1 - \cos(\pi \text{clamp}(\alpha - i + N_{\text{fmin}}, 0, 1)))$, $\alpha(n) = (N_{\text{fmax}} - N_{\text{fmin}})\frac{n}{N_{\text{anneal}}}$, n is the current training iteration, N_{fmax} is the total number of used PE frequencies (the proposed model uses 12), N_{fmin} is the number of used PE frequences at the start (we use 8), N_{anneal} is tuned empirically to $3 \cdot 10^4$ for all sequences. This training strategy enables significantly improved geometry predictions.

Nerfies: Deformable Neural Radiance Fields

Ray Direction Jitter

$$x_i = \mathbf{o} + t_i(\mathbf{d} + \psi).$$

Jitter

* Improving robustness by adding some noise in the direction.

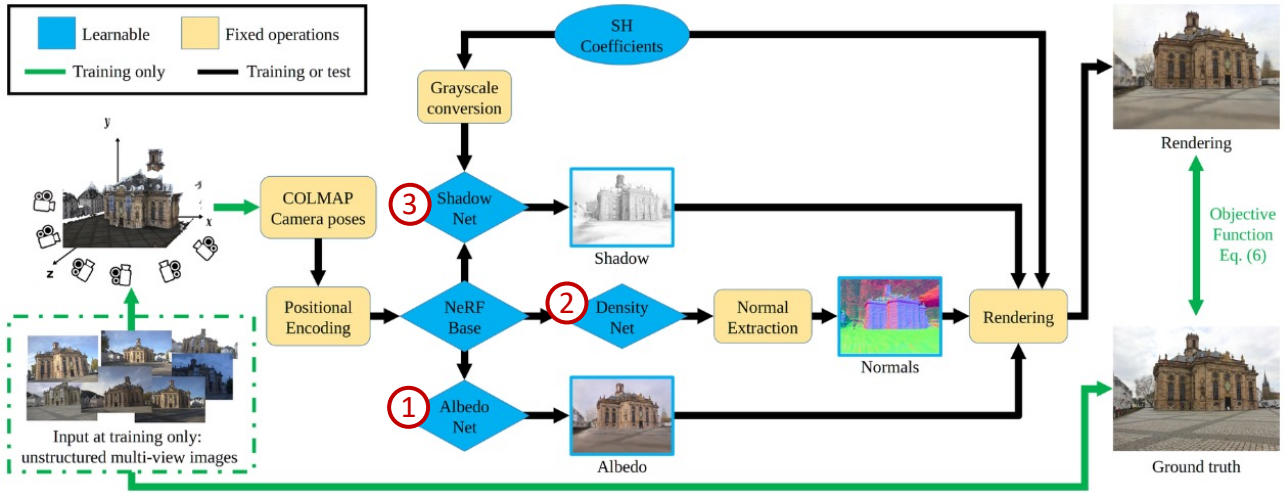
Ray Direction Jitter. To improve the generalisability of NeRF-OSR, we apply a sub-pixel jitter to the ray direction. Here, instead of shooting in the pixel centres, a jitter ψ is used as follows: $x_i = \mathbf{o} + t_i(\mathbf{d} + \psi)$. We sample ψ uniformly, such that the resulting ray still confines to the boundaries of its designated pixel.

Shadow Network Input Jitter

$$S'\left(\{\mathbf{x}_i\}_{i=1}^{N_{\text{depth}}}, \mathbf{L}\right) = S\left(\{\mathbf{x}_i\}_{i=1}^{N_{\text{depth}}}, \mathbf{L} + \varepsilon\right)$$

* Tackle overfitting issue.

where $\varepsilon \sim \mathcal{N}(0, 0.025I)$. (7) can be interpreted as a locality condition, *i.e.*, in similar lighting conditions, shadows should not be too different. This allows the model to learn smoother transitions between different lightings.

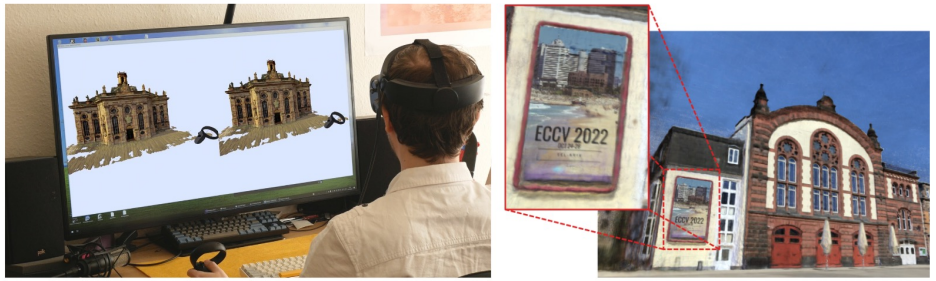


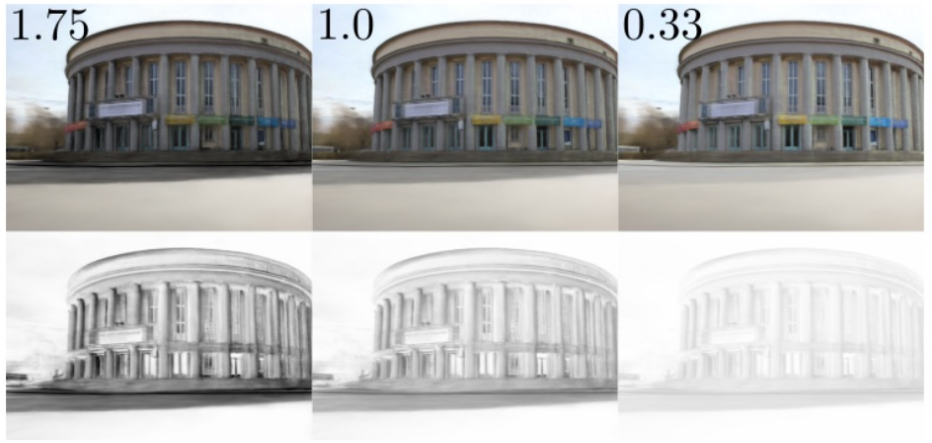
NeRF-OSR Framework

Implementation details

Implementation. We use NeRF++ [48] with the background network disabled as the code base and work within the unit sphere bounds of the foreground network. For training and evaluation, we use two Nvidia Quadro RTX 8000 GPUs. We train the model for $5 \cdot 10^5$ iterations using a batch size of 2^{10} rays, which takes ≈2 days.

We extract the geometry and albedo from the learned model of Site 1 as a mesh using Marching Cubes [16] at resolution 1000^3 voxels. Then we use them in our interactive VR renderer implemented with C++, OpenGL and SteamVR. The lighting model consists of the sun and a simple geometry-based shadow map [43]: $\mathbf{C}_{\text{interactive}} = \mathbf{C}_{\text{ambient}} + s \odot \mathbf{C}_{\text{sun}} \max\{0, \mathbf{D}_{\text{sun}}^T \mathbf{N}\}$, where $\mathbf{C}_{\text{ambient}}$ is the ambient colour, s is 0 when the rendered point is occluded and 1 if not, according to the shadow map, $\mathbf{C}_{\text{sun}}$ is the colour of the sun, $\mathbf{D}_{\text{sun}}$ is the direction of the sun and $\mathbf{N}$ is the normal of the mesh. The user can interactively move in the scene and control the sun direction with their controllers. The demo runs in real-time on a desktop computer with an Intel i7-4770 CPU, an Nvidia GeForce GTX 970 (4GB VRAM) GPU and an Oculus Rift S HMD. The system RAM usage of the application is below 3 GB. We provide an extensive demo in the supplementary video and show an extract in Fig. 7-(left).





Real time interactive VR

Expreiments

Training image | Diffuse Albedo | Surface normals | Shadows | Novel lighting and viewpoint

Ground truth | Philip et al.

Our NeRF-OSR | Yu et al.

Expreiments

Method	PSNR ↑	MSE ↓	MAE ↓	SSIM ↑
	Site 1			
Yu *et al.* [46]	18.71	0.014	0.088	0.4
Philip *et al.* [27] (d/s)	17.37	0.019	0.105	0.429
Ours (d/s)	**19.86**	**0.011**	**0.08**	**0.626**
Yu *et al.* [46] (u/s)	17.87	0.017	0.097	0.378
Philip *et al.* [27]	16.63	0.023	0.113	0.367
Ours	**18.72**	**0.014**	**0.09**	**0.468**
No shadows	17.82	0.017	0.101	0.418
No annealing	17.16	0.02	0.108	0.324
No ray jitter	18.43	0.015	0.093	0.433
No shadow jitter	18.28	0.016	0.095	0.413
No shadow regulariser	17.62	0.018	0.105	0.373

Method	PSNR ↑	MSE ↓	MAE ↓	SSIM ↑
	Site 2			
Yu *et al.* [46]	15.43	0.031	0.136	0.363
Philip *et al.* [27] (d/s)	11.85	0.07	0.21	0.184
Ours (d/s)	**15.83**	**0.026**	**0.128**	**0.556**
Yu *et al.* [46] (u/s)	15.28	0.032	0.138	0.385
Philip *et al.* [27]	12.34	0.065	0.2	0.272
Ours	**15.43**	**0.029**	**0.133**	**0.517**
	Site 3			
Yu *et al.* [46]	15.84	0.028	0.123	0.392
Philip *et al.* [27] (d/s)	12.85	0.054	0.169	0.164
Ours (d/s)	**17.38**	**0.021**	**0.106**	**0.576**
Yu *et al.* [46] (u/s)	15.17	0.033	0.133	0.376
Philip *et al.* [27]	12.28	0.062	0.179	0.319
Ours	**16.65**	**0.024**	**0.114**	**0.501**

Need to Comparison D/S

nique significantly outperforms related methods [27, 46]. "d/s" and "u/s" are shorthands for "downscaled" and "upscaled", respectively. Bottom left: ablation study of our various design choices. Our full model achieves the best result.

Conclusion

Limitations

- Need to capture high frequency details
- Some viewpoints occur blur and ghosting effects.

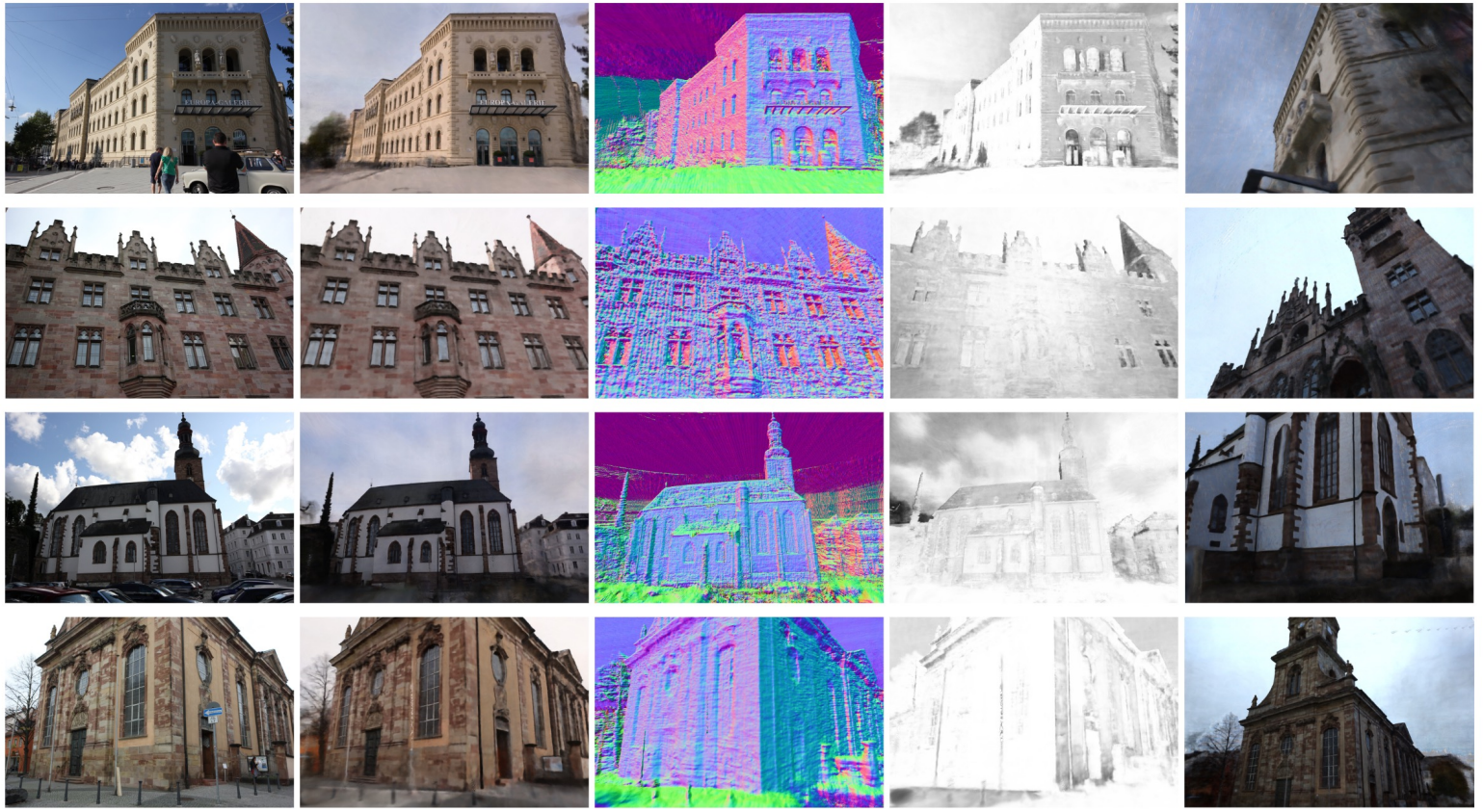
Conclusion

- Flexible camera viewing point and relighting In the Real-World
- Decompose Albedo, Shadows, Surface using self-supervised learning strategy.
- (Future Works) generalize on the Few training dataset

Supplementary Material

	Sessions	Views		Sessions	Views
Site 1	18	373	Site 5	13	493
Site 2	17	423	Site 6	12	379
Site 3	16	372	Site 7	11	468
Site 4	11	401	Site 8	12	331
			Total	110	3240

Dataset Info



Training image | Diffuse Albedo | Surface Normals | Shadows | N.l. and v.

Supplementary Material

Semantic-aware Occlusion Filtering Neural Radiance Fields in the Wild

Jaewon Lee[1], Injae Kim[1], Hwan Hwo[1], Hyunwoo J. Kim[1]

[1]Korea University

논문 리뷰 PseudoLab 9기 홍세준

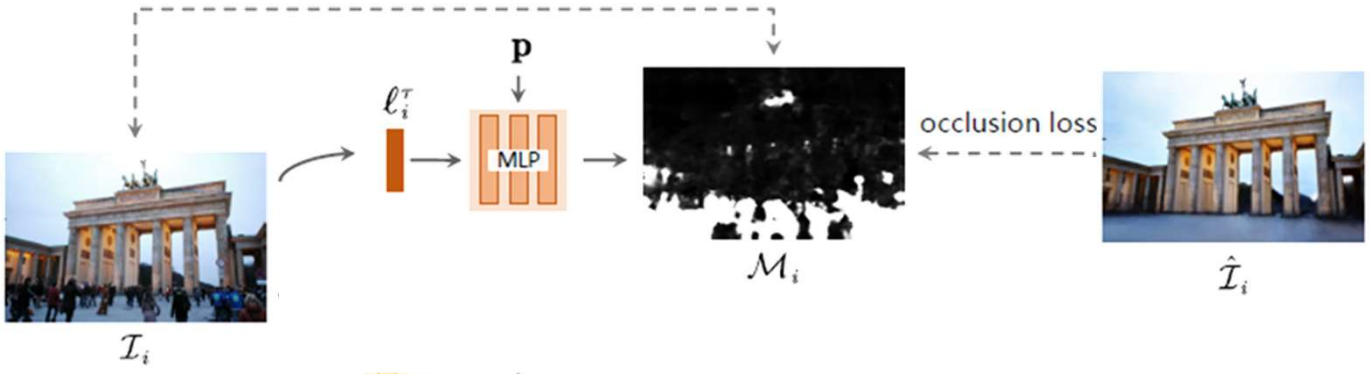
NeRFwithRealWorld

Introduction

- “Transient Occluders”

1) Utilize an additional transient module that separates transient components from the scene

ex) HA-NeRF



→ Require a large number of images to get rid of the occluders

Introduction

- “Transient Occluders”

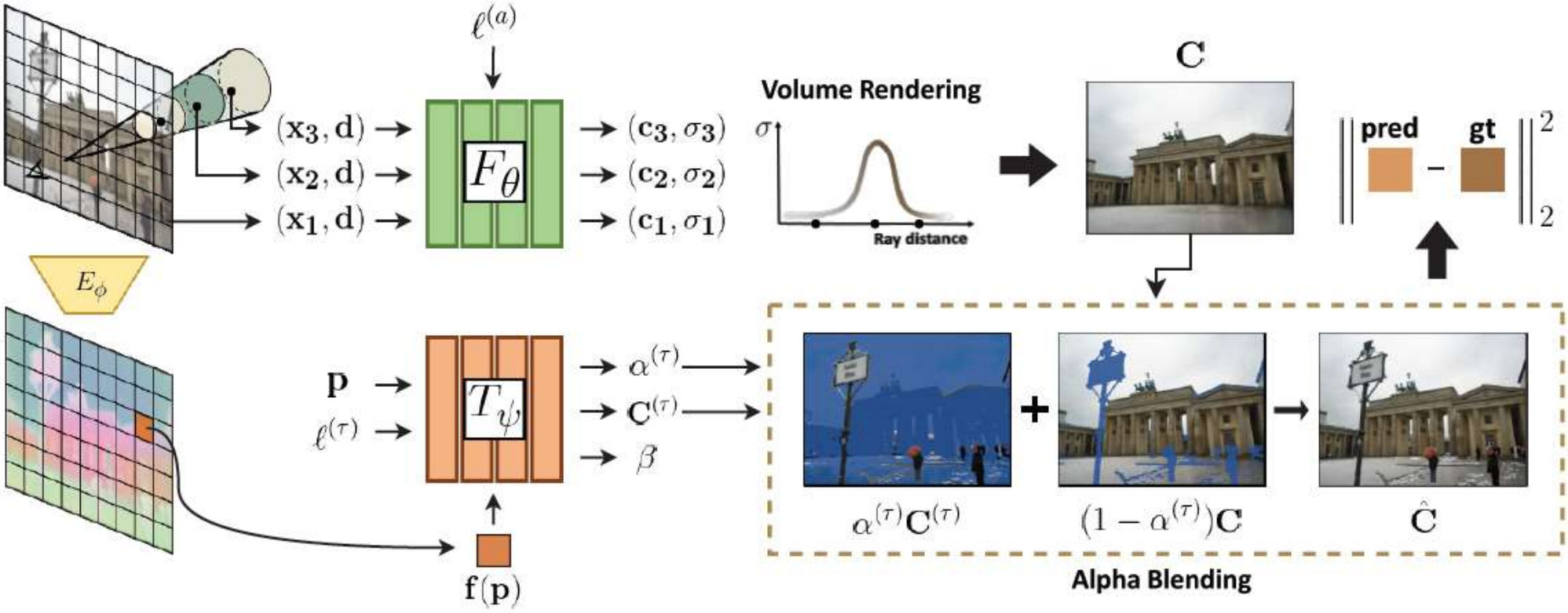
2) Use segmentation models to mask out moving objects

ex) Block-NeRF

→ Restrict pre-defined classes

Method

- **SF-NeRF Framework**



NeRFwithRealWorld

Method

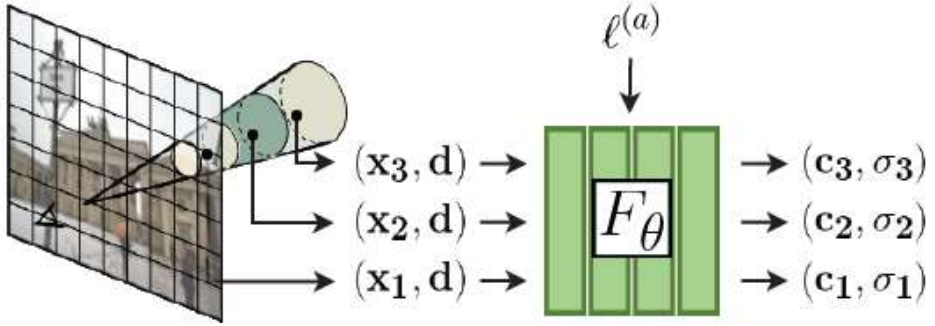
- **Latent Conditional NeRF & Mip-NeRF**

$$[\sigma,\ \mathbf{z}] = \mathrm{MLP}_{\theta_1}\left(\gamma_{\mathbf{x}}(\mathbf{x})\right),$$
$$\mathbf{c} = \mathrm{MLP}_{\theta_2}\left(\gamma_{\mathbf{d}}(\mathbf{d}),\ \mathbf{z}\right),$$

$$\mathbf{C}(\mathbf{r}) = \sum_{k=1}^{K} T(t_k)\alpha(t_k)\mathbf{c}(t_k),$$

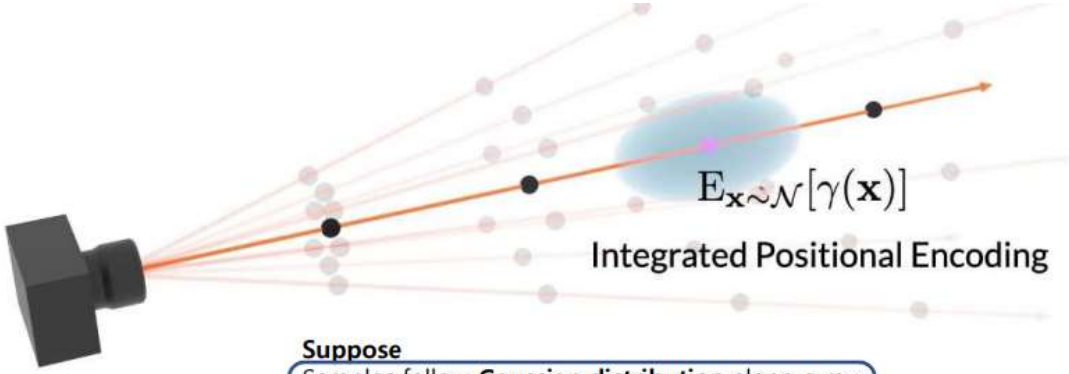
where $\alpha(t_k) = 1 - \exp(-\sigma(t_k)\delta_k),$

$$T(t_k) = \prod_{k'=1}^{k-1}\left(1 - \alpha(t_{k'})\right)$$
$$= \exp\left(-\sum_{k'=1}^{k-1}\sigma(t_{k'})\delta_{k'}\right)$$



$$\mathbf{c}_i = \mathrm{MLP}_{\theta_2}\left(\gamma_{\mathbf{d}}(\mathbf{d}), \mathbf{z}, \ell_i^{(a)}\right),$$

$$\mathbf{C}_i(\mathbf{r}) = \sum_{k=1}^{K} T(t_k)\alpha(t_k)\mathbf{c}_i(t_k).$$



Suppose
Samples follow **Gaussian distribution** along a ray direction and it vertical one (multivariate Gaussian distribution)

Method

1) Semantic-aware Scene Decomposition

transient embedding $\ell_i^{(\tau)} \in \mathbb{R}^{n^{(\tau)}}$

pixel location $\mathbf{p} \in \mathbb{R}^2$

encoded feature $\mathbf{f}_i(\mathbf{p}) \in \mathbb{R}^F$

$$\left[\alpha_i^{(\tau)}, \mathbf{C}_i^{(\tau)}, \beta_i\right] = T_\psi\left(\gamma_{\mathbf{p}}(\mathbf{p}), \ell_i^{(\tau)}, \mathbf{f}_i(\mathbf{p})\right)$$

transient color $\mathbf{C}_i^{(\tau)}$

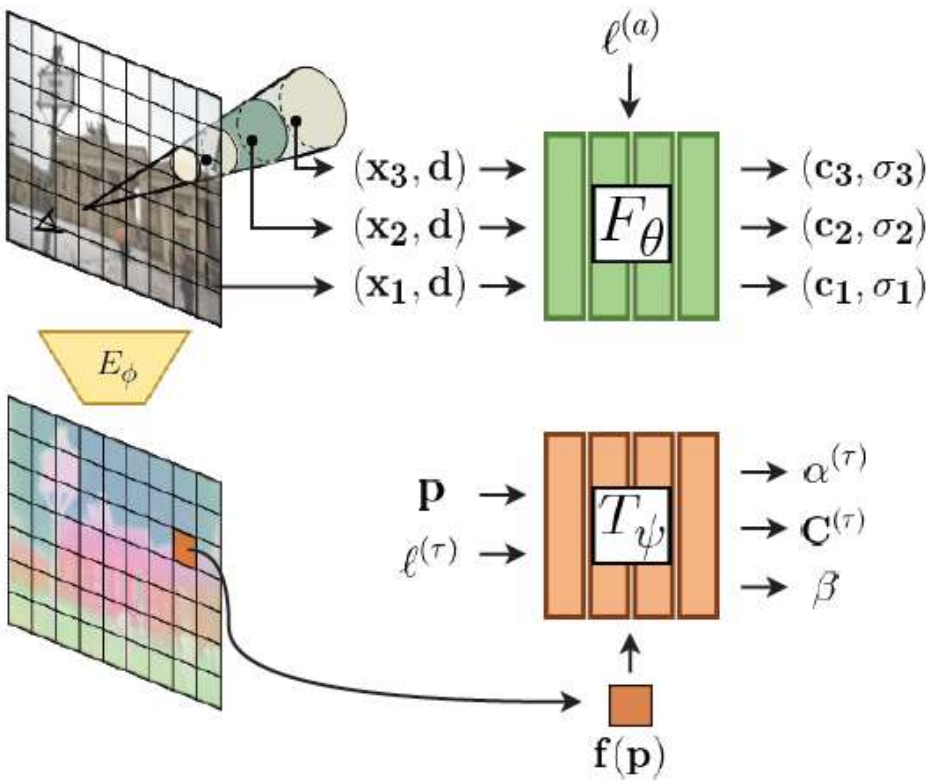
opacity $\alpha_i^{(\tau)} \in [0, 1]$

uncertainty $\beta_i \in \mathbb{R}^+$

$\gamma_{\mathbf{p}} : \mathbb{R}^2 \rightarrow \mathbb{R}^{4L}$ is the positional encoding

$$\hat{\mathbf{C}}_i(\mathbf{r}) = \alpha_i^{(\tau)}(\mathbf{p_r})\, \mathbf{C}_i^{(\tau)}(\mathbf{p_r}) + \left(1 - \alpha_i^{(\tau)}(\mathbf{p_r})\right) \mathbf{C}_i(\mathbf{r}).$$

$$\mathcal{L}_{\mathrm{t}}^{(i)}(\mathbf{r}) = \frac{\left\|\hat{\mathbf{C}}_i(\mathbf{r}) - \bar{\mathbf{C}}_i(\mathbf{r})\right\|_2^2}{2\beta_i(\mathbf{p_r})^2} + \frac{\log \beta_i(\mathbf{p_r})^2}{2} + \lambda_\alpha \alpha_i^{(\tau)}(\mathbf{p_r}).$$



Method

2) Transient Opacity Reparameterization

Binary Concrete random variable

$$\alpha_i^{(\tau)} = \text{sigmoid}\left(\frac{1}{t} \cdot (\log \tilde{\alpha}_i + \log U - \log (1 - U))\right)$$
$$U \sim \text{Uniform}(0, 1),$$

Method

3) Transient Opacity Smoothness Prior

$$\mathcal{L}_{\text{sm}}^{(i)}(\mathbf{p}) = \sum_{k=0}^{L-1} 2^k \left\| \frac{\partial \alpha_i^{(\tau)}(\mathbf{p})}{\partial \gamma_k(\mathbf{p})} \right\|_1 \qquad \gamma_k(\mathbf{p}) = \left[\cos(2^k \pi \mathbf{p}), \sin(2^k \pi \mathbf{p})\right]$$

$$\left\| \frac{\partial \alpha_i^{(\tau)}(\mathbf{p})}{\partial \mathbf{p}} \right\|_1 \leq \sum_{k=0}^{L-1} 2^k \pi \left\| \frac{\partial \alpha_i^{(\tau)}(\mathbf{p})}{\partial \gamma_k(\mathbf{p})} \right\|_1$$

4) Optimization

$$\sum_{i=1}^{N} \sum_{\mathbf{r} \in \mathcal{R}_i} \left[\mathcal{L}_{\text{t}}^{(i)}(\mathbf{r}) + \lambda_{\text{c}} \mathcal{L}_{\text{c}}^{(i)}(\mathbf{r}) + \lambda_{\text{sm}} \mathcal{L}_{\text{sm}}^{(i)}(\mathbf{p}_\mathbf{r}) + \lambda_{\text{sp}} \mathcal{L}_{\text{sp}}(\mathbf{r})\right] + \lambda_a \left\| \ell_i^{(a)} \right\|_2^2,$$

$$\mathcal{L}_{\text{c}}^{(i)}(\mathbf{r}) = \left(1 - \alpha_i^{(\tau)}(\mathbf{p}_\mathbf{r})\right) \left\| \mathbf{C}_i(\mathbf{r}; \mathbf{t}^c) - \bar{\mathbf{C}}_i(\mathbf{r}) \right\|_2^2,$$

$$\mathcal{L}_{\text{sp}}(\mathbf{r}) = \sum_k \log\left(1 + 2\sigma(t_k^c)^2\right),$$

Experiments

- **Datasets**: Phototourism (Few 15, Few 30)
- **Baselines**: NeRF, NeRF-W, HA-NeRF, NeRF-AM

    - Image Encoder: DINO
    - FilterNet: 5 fully connected layers of 128 hidden units

Experiments

- Quantitative Results

Few15	Brandenburg Gate			Sacre Coeur			Trevi Fountain			Taj Mahal		
	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
NeRF [1]	11.54	0.536	0.503	13.25	0.581	0.409	12.36	0.415	0.474	11.99	0.564	0.506
NeRF-AM	13.65	0.627	0.468	17.53	0.675	0.290	17.53	0.538	0.351	19.21	0.733	0.313
NeRF-W [28]	**20.80**	0.787	0.266	**17.87**	**0.715**	**0.235**	18.15	0.596	0.333	20.32	0.787	0.259
HA-NeRF [31]	13.83	0.693	0.516	15.41	0.675	0.423	14.60	0.540	0.416	13.84	0.703	0.458
SF-NeRF	19.72	**0.790**	**0.260**	17.69	0.707	0.240	**18.94**	**0.607**	**0.281**	**20.47**	**0.801**	**0.226**

Few30	Brandenburg Gate			Sacre Coeur			Trevi Fountain			Taj Mahal		
	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
NeRF [1]	11.42	0.514	0.488	10.04	0.421	0.551	11.31	0.333	0.496	11.28	0.490	0.588
NeRF-AM	13.20	0.628	0.434	12.94	0.506	0.531	15.44	0.441	0.445	18.41	0.686	0.338
NeRF-W [28]	22.74	**0.847**	0.188	19.31	0.750	0.205	19.54	0.646	0.298	**21.03**	0.792	0.259
HA-NeRF [31]	19.88	0.803	0.278	17.66	0.736	0.256	16.97	0.598	0.352	17.92	0.785	0.321
SF-NeRF	**23.23**	0.846	**0.178**	**19.64**	**0.757**	**0.186**	**20.24**	**0.657**	**0.243**	20.86	**0.820**	**0.208**

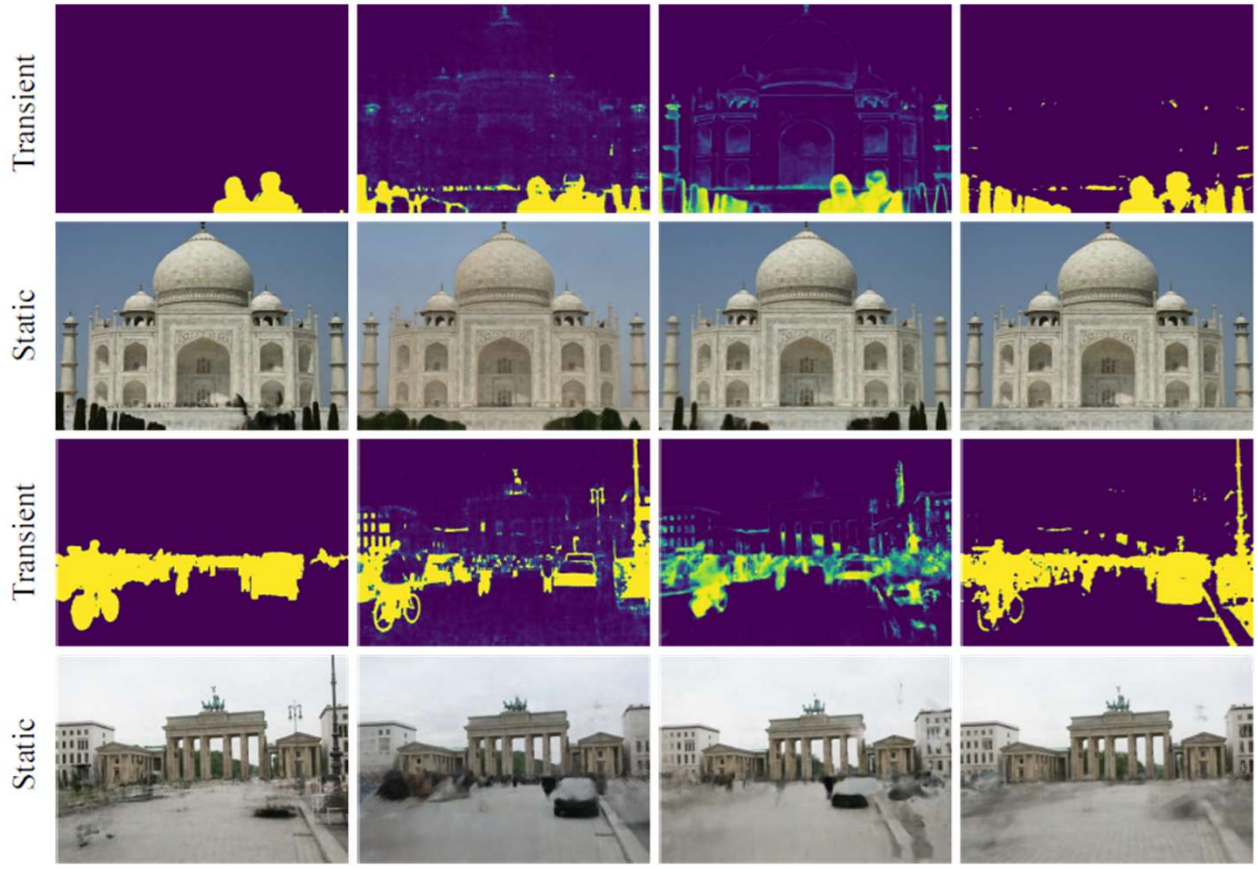
Experiments

- Qualitative Results

NeRFwithRealWorld

Experiments

- Qualitative Results

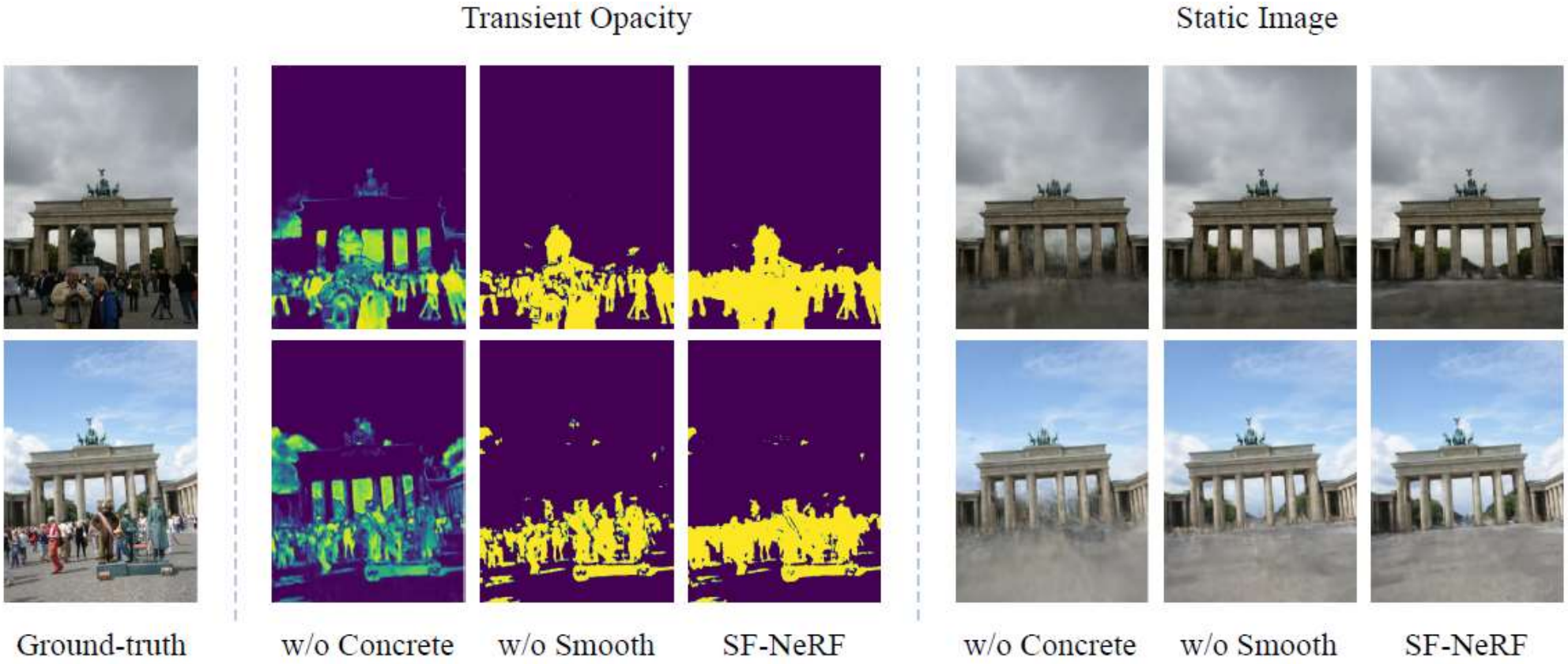


NeRFwithRealWorld

Experiments

- Ablation Results

	PSNR↑	SSIM↑	LPIPS↓
w/o Concrete	20.65	0.823	0.248
w/o Smooth	22.27	0.802	0.191
SF-NeRF	**23.23**	**0.846**	**0.178**



Q & A

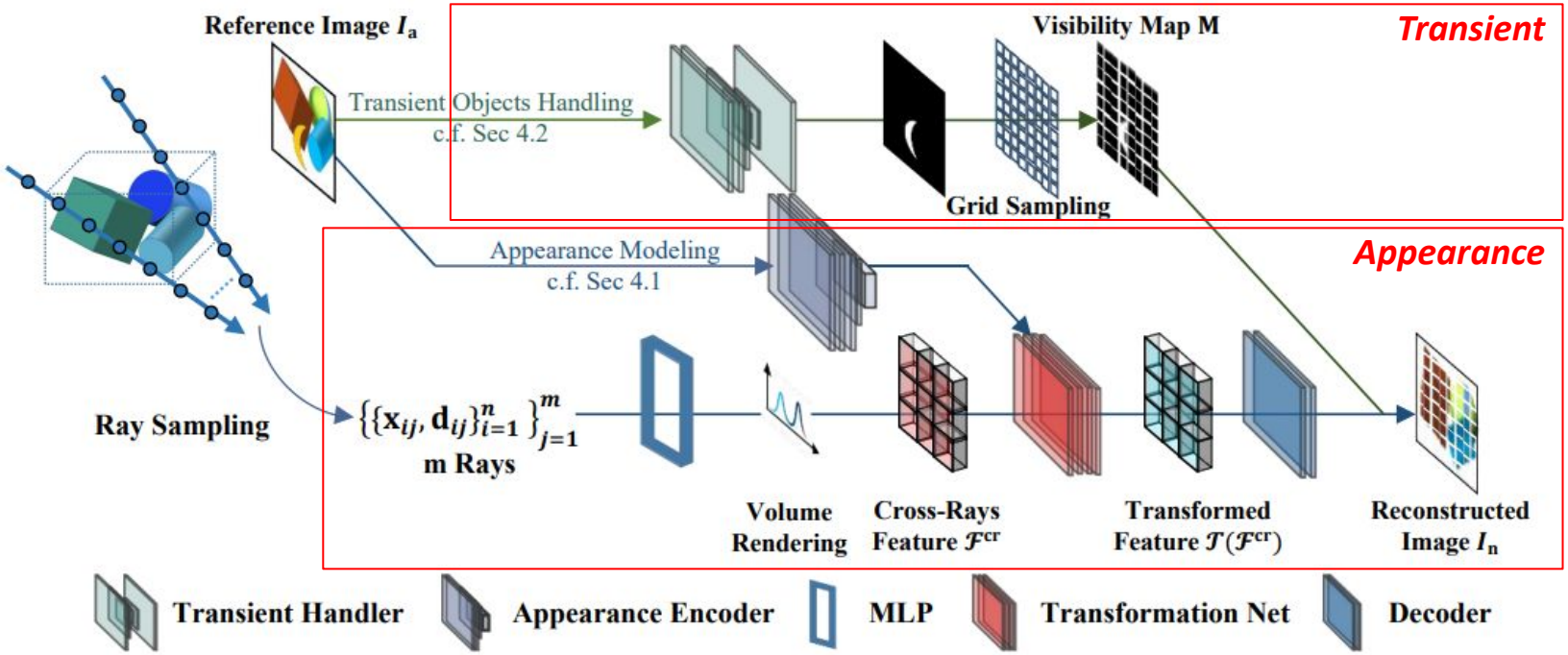
ICCV 2023

Cross-Ray Neural Radiance Fields for Novel-view Synthesis from Unconstrained Image Collections

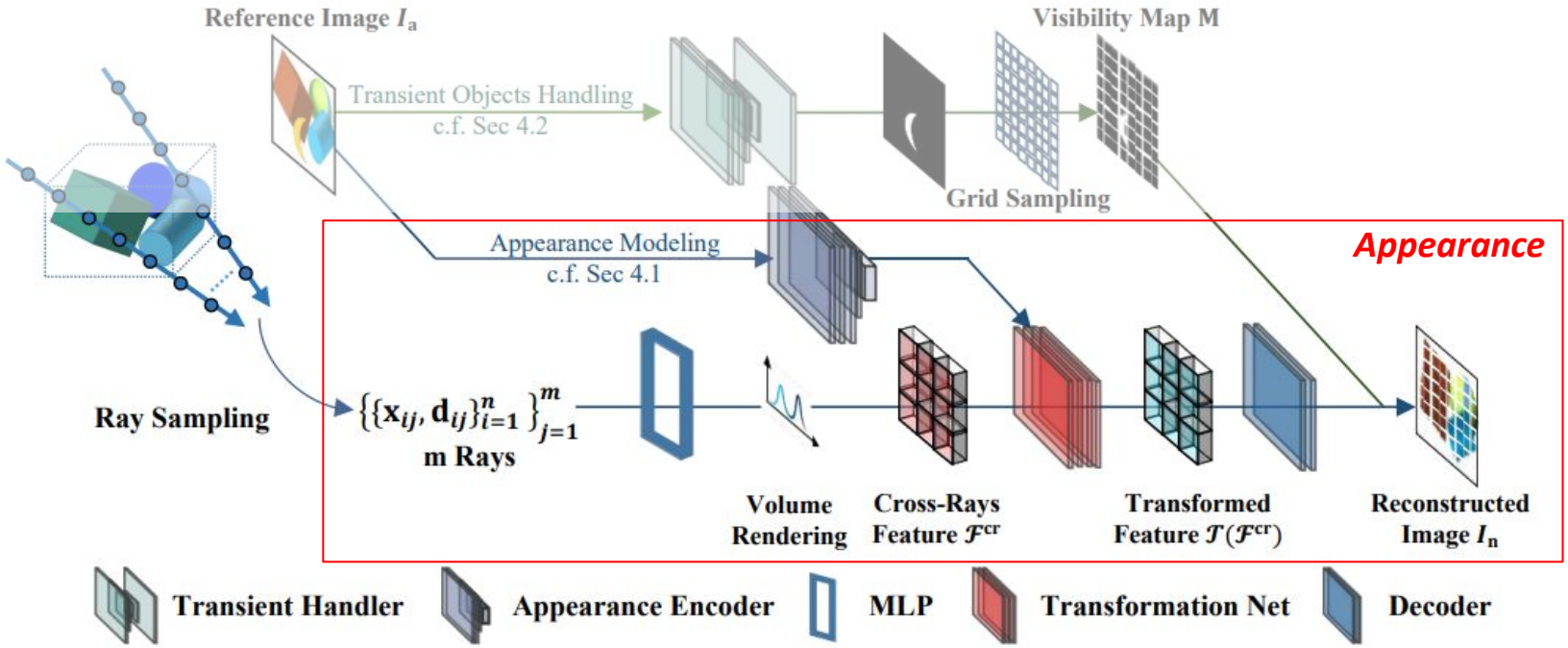
Contents

Cross-Ray Neural Radiance Fields

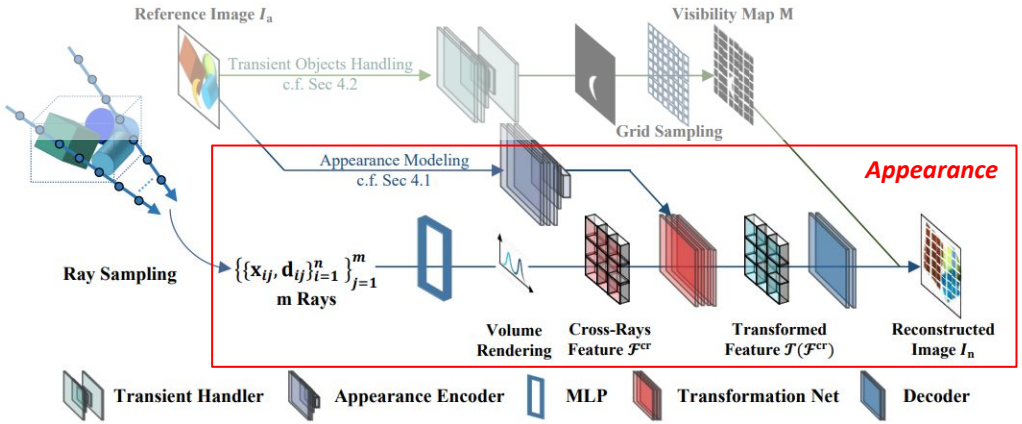
Architecture



Cross-Ray Appearance Modeling



Cross-Ray Appearance Modeling



1) multiple rays

$$\{\{\mathcal{F}_{ij}^{r}, \sigma_{ij}\}_{i=1}^{n}\}_{j=1}^{m} = \{\mathrm{MLP}_{\theta_1}(\{\mathbf{x}_{ij}, \mathbf{d}_{ij}\}_{i=1}^{n})\}_{j=1}^{m}, \quad (4)$$
$$\mathcal{F}^{\mathrm{cr}} = \{\mathrm{VR}(\{\mathcal{F}_{ij}^{r}, \sigma_{ij}, \delta_{ij}\}_{i=1}^{n})\}_{j=1}^{m},$$

2) Encoding Feture

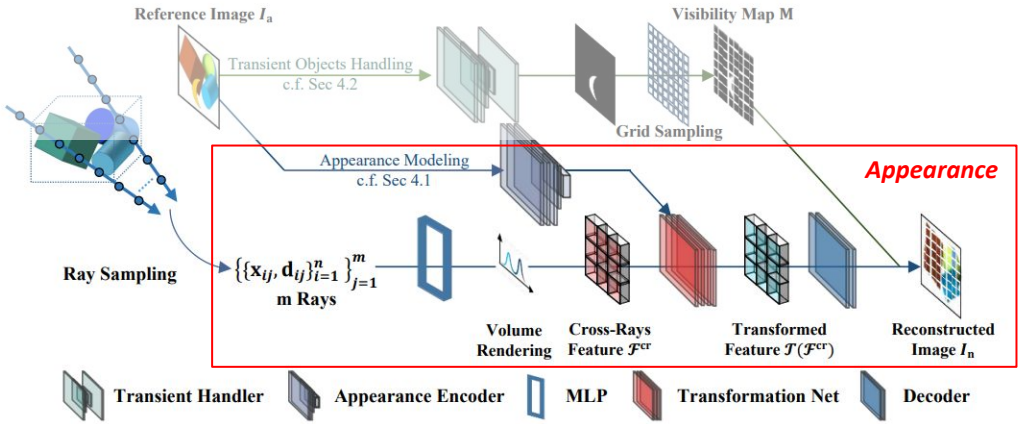
$$\mathcal{F}^{a} = E_{\theta_2}(\mathcal{I}_a).$$

3) Transformed Feature

we learn a transformation T to align the transferred cross-ray features

$$\mathcal{T}(\mathcal{F}^{\mathrm{cr}})$$

Cross-Ray Appearance Modeling



+ Loss Function for varying appearance modeling

$$\mathcal{L}_a = ||E_{\theta_2}[D_{\theta_4}(\mathcal{T}_{\theta_3}(\mathcal{F}^{\mathrm{cr}}))] - \mathcal{F}^{\mathrm{a}}||_2^2 + \beta||E_{\theta_5}[D_{\theta_4}(\mathcal{T}_{\theta_3}(\mathcal{F}^{\mathrm{cr}}))] - E_{\theta_5}[D_{\theta_4}(\mathcal{F}^{\mathrm{cr}})]||_2^2, \quad (8)$$

4) Injecting Appearance

$$\min_{\mathcal{T}} \mathbb{E}_{\mathcal{F}^{\mathrm{cr}}, \mathcal{F}^a} \|\mathcal{T}(\mathcal{F}^{\mathrm{cr}}) - \mathcal{F}^a\|_2^2 + \beta\|\mathbf{P}\mathcal{T}(\mathcal{F}^{\mathrm{cr}}) - \mathcal{F}^{\mathrm{cr}}\|_2^2, \quad (5)$$

where β is a trade-off parameter and $\mathbf{P}$ is a constant matrix for matching the transformed feature $\mathcal{T}(\mathcal{F}^{\mathrm{cr}})$ and $\mathcal{F}^{\mathrm{cr}}$.

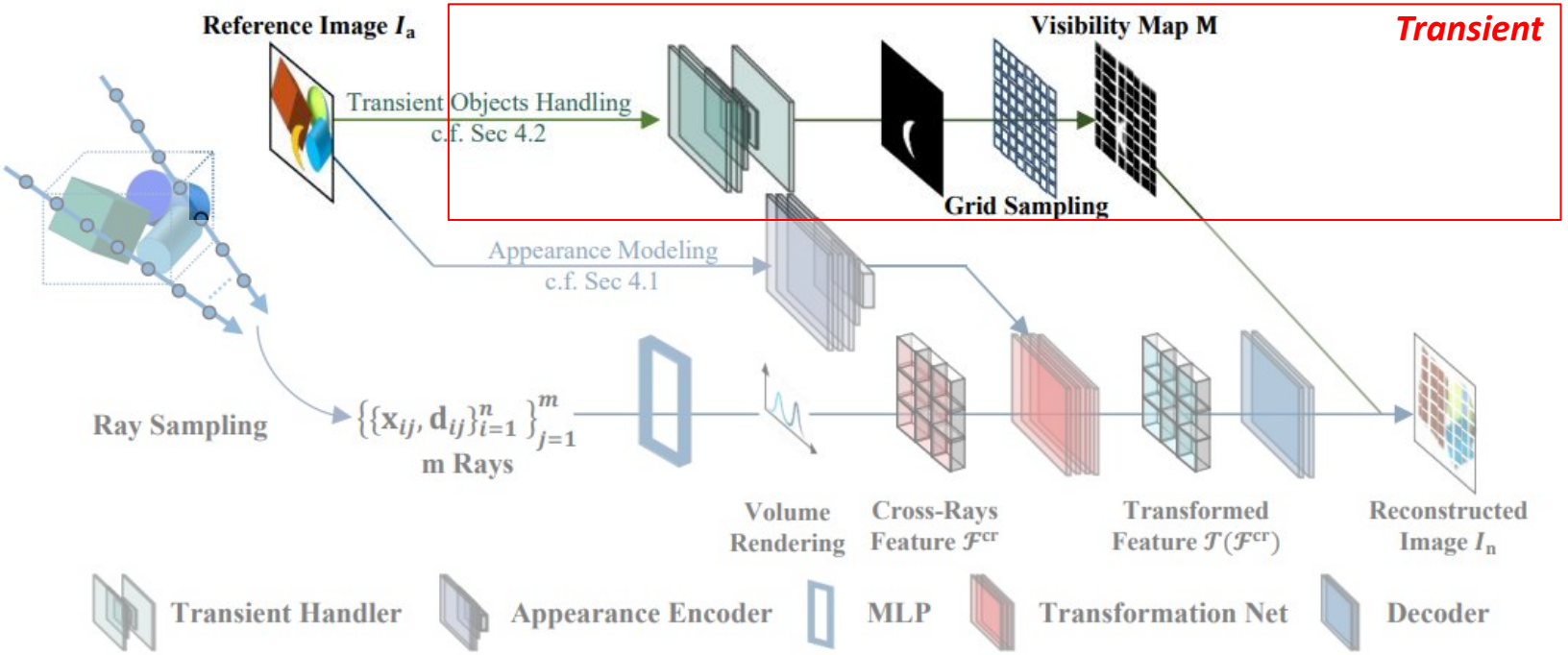
+ How to Transformed?

the transformation matrix T is determined by the covariance of Fcr and Fa

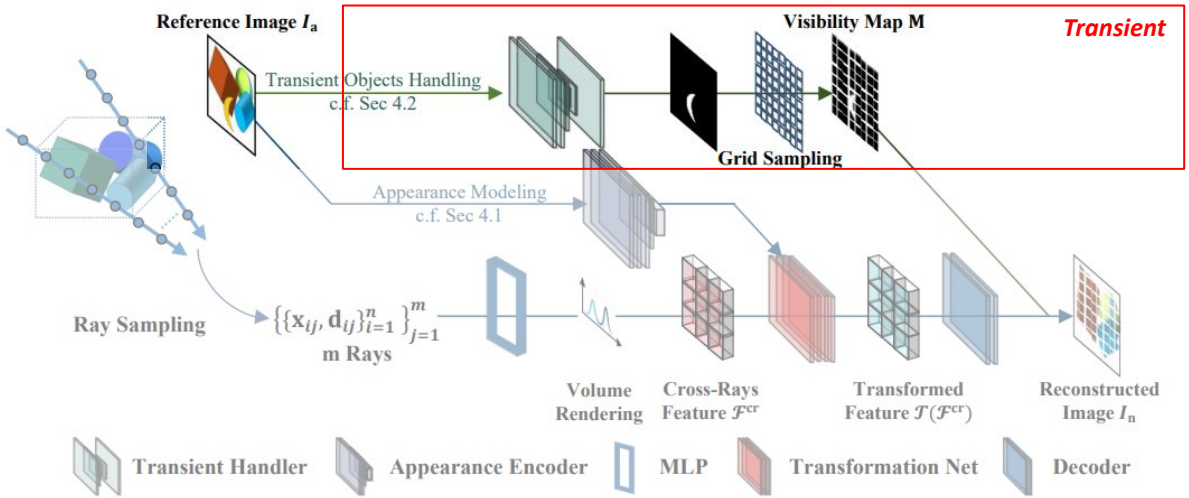
$$\mathcal{T}(\mathcal{F}^{\mathrm{cr}}) = \mathbf{T}\hat{\mathcal{F}}^{\mathrm{cr}},$$
$$\text{where } \mathbf{T} = Cov(\bar{\mathcal{F}}^{\mathrm{cr}})Cov(\bar{\mathcal{F}}^{a})), \quad (7)$$
$$\hat{\mathcal{F}}^{\mathrm{cr}} = \phi_1(\mathcal{F}^{\mathrm{cr}}), \bar{\mathcal{F}}^{\mathrm{cr}} = \phi_2(\mathcal{F}^{\mathrm{cr}}), \bar{\mathcal{F}}^{a} = \phi_3(\mathcal{F}^{a}).$$

Here, ϕ1, ϕ2, and ϕ3 are non-linear mappings parameterized by convolutional neural networks (CNN)

Transient Objects Handling



Transient Objects Handling



1) Segmentation Network

segmenting common objects such as tourists and cars, etc. (pretrained Cgnet [57])

$$\mathcal{S}_{\theta_\Delta}(\mathcal{I}_a)$$

* Since we cannot sample all rays that interact with Ia due to **limited GPU memory** in the training phase

2) Grid sampling strategy (GS)

samples $\mathcal{S}_{\theta_\Delta}(\mathcal{I}_a)$ to pair with m rays

$$\mathbf{M} = \mathrm{GS}(\mathcal{S}_{\theta_\Delta}(\mathcal{I}_a)), \qquad (9)$$

+ Loss function for eliminating transient objects

$$\mathcal{L}_t = ||(1-\mathbf{M}) \odot (\mathcal{I}_n - \mathcal{I}_a)^2||_1 + \lambda_0 ||\mathbf{M}||^2, \qquad (10)$$

$$\mathcal{L}_{overall} = \mathcal{L}_a + \lambda \mathcal{L}_t, \qquad (3)$$

[57] Tianyi Wu, Sheng Tang, Rui Zhang, Juan Cao, and Yongdong Zhang. Cgnet: A light-weight context guided network for semantic segmentation. IEEE TIP, 30:1169–1179, 2020

Experimental & Results

		Brandenburg Gate			Sacre Coeur			Trevi Fountain		
		PSNR (↑)	SSIM (↑)	LPIPS (↓)	PSNR (↑)	SSIM (↑)	LPIPS (↓)	PSNR (↑)	SSIM (↑)	LPIPS (↓)
R / 4	NeRF	19.62	0.8200	0.1455	16.21	0.7197	0.2181	16.40	0.6189	0.2422
	NeRF-W*	24.00	0.8758	0.1332	21.07	**0.8422**	0.1119	19.75	0.7207	0.2029
	Ha-NeRF	24.58	0.8829	0.0927	20.36	0.7947	0.1317	20.27	0.7270	0.1628
	CR-NeRF-R (Ours)	26.18	0.8937	0.0840	21.64	0.8206	0.1160	20.08	0.6538	0.2372
	CR-NeRF (Ours)	**26.86**	**0.9069**	**0.0733**	**22.03**	0.8369	**0.1060**	**22.02**	**0.7488**	**0.1354**
R / 2	NeRF	18.90	0.8159	0.2316	15.60	0.7155	0.2916	16.14	0.6007	0.3662
	NeRF-W*	24.17	0.8905	0.1670	19.20	0.8076	0.1915	18.97	0.6984	0.2652
	Ha-NeRF	24.04	0.8773	0.1391	20.02	0.8012	0.1710	20.18	0.6908	0.2225
	CR-NeRF-R (Ours)	25.94	0.8929	0.1378	21.66	0.8171	0.1646	21.37	0.7111	0.2212
	CR-NeRF (Ours)	**26.53**	**0.9003**	**0.1060**	**22.07**	**0.8233**	**0.1520**	**21.48**	**0.7117**	**0.2069**

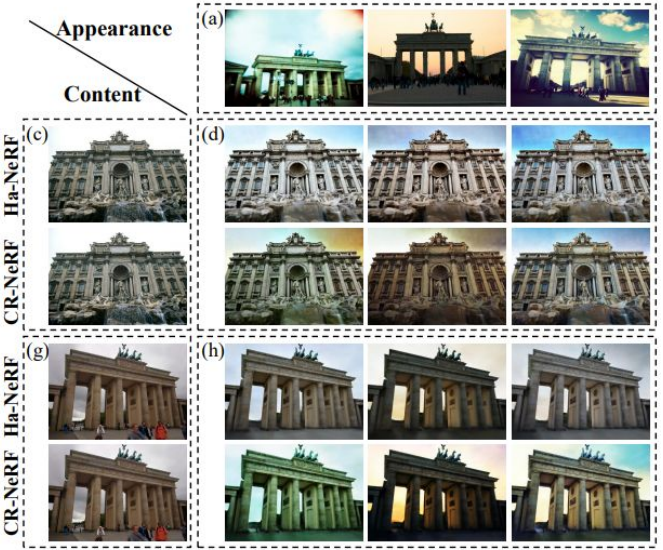
Table 1. Quantitative experimental results on three real-world datasets under two resolution settings, *i.e.*, downscaling original image resolution by 2 (R/2) and 4 (R/4). The **bold** and the underlined numbers indicate the best and second-best results, respectively.

	Brandenburg Gate			Sacre Coeur			Trevi Fountain		
	PSNR (↑)	SSIM (↑)	LPIPS (↓)	PSNR (↑)	SSIM (↑)	LPIPS (↓)	PSNR (↑)	SSIM (↑)	LPIPS (↓)
CR-NeRF-B	19.58	0.8216	0.1470	16.11	0.7145	0.2196	16.37	0.6206	0.2493
CR-NeRF-A	26.38	0.8929	0.0885	21.67	0.8182	0.1127	21.85	0.7473	0.1388
CR-NeRF-T	20.46	0.8361	0.1300	16.28	0.7650	0.1799	16.55	0.6446	0.2230
CR-NeRF	**26.86**	**0.9069**	**0.0733**	**22.03**	**0.8369**	**0.1060**	**22.02**	**0.7488**	**0.1354**

Table 2. Ablation studies of CR-NeRF on three datasets. The performance of our baseline (CR-NeRF-B) is progressively improved by adding the appearance modeling module (CR-NeRF-A) and the transient handler (CR-NeRF-T). The bold numbers indicate the best result.

Experimental & Results

Figure 3. Qualitative experimental results on three unconstrained datasets. CR-NeRF recovers realistic appearance (*e.g.,* green plant in Brandenburg, sunshine on statues in Trevi, and light blue sky in Sacre.). Moreover, CR-NeRF removes transient objects for a consistent geometry (*e.g.,* less ghost effects around pillars of Brandenburg and Sacre).



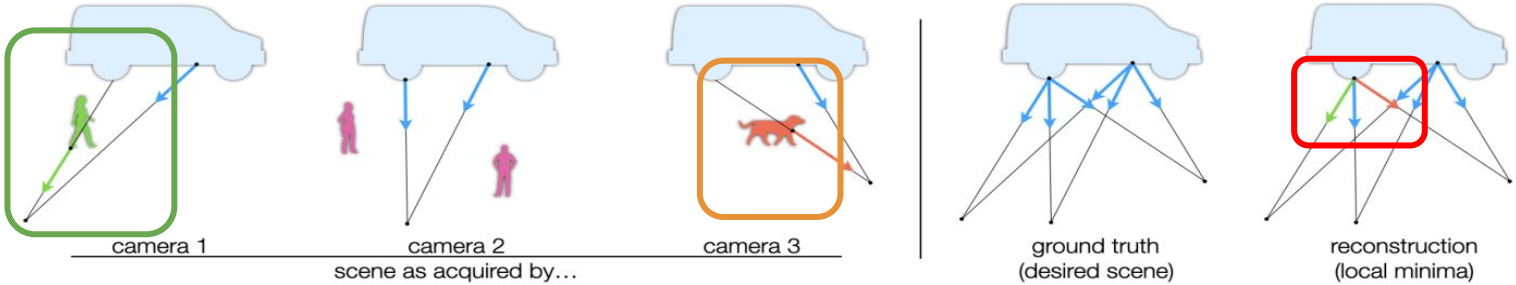
1) captures appearance information more accurately
2) successfully removes transient objects

RobustNeRF: Ignoring Distractors with Robust Losses

https://arxiv.org/pdf/2302.00833

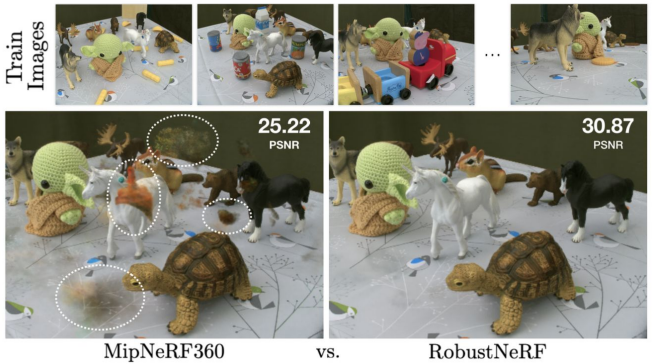
PseudoLab

Introduction



Main Problem

- distractor와 view-dependent 요소를 구분하기 어려움
- distractor가 렌더링 시, floater로 나타남



RobustNeRF

- Robust loss를 적용하여, 렌더링 시 distractor를 무시
- loss function에 IRLS + inductive bias 적용
- 구조가 단순하지만, 매우 효과적
- 다양한 데이터셋에 활용 가능

PseudoLab

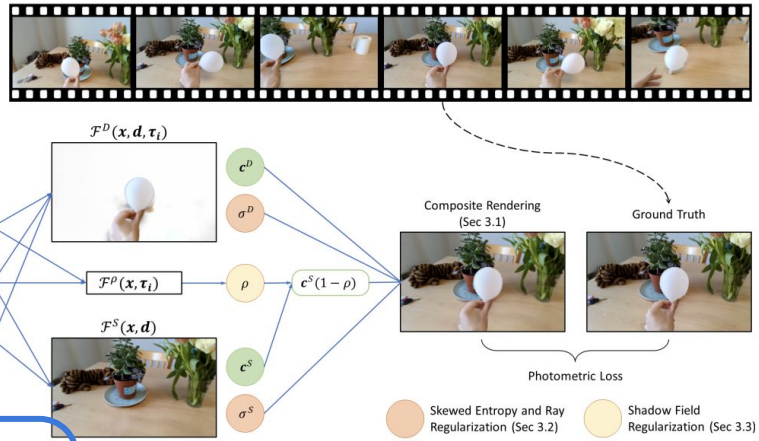
Related Work

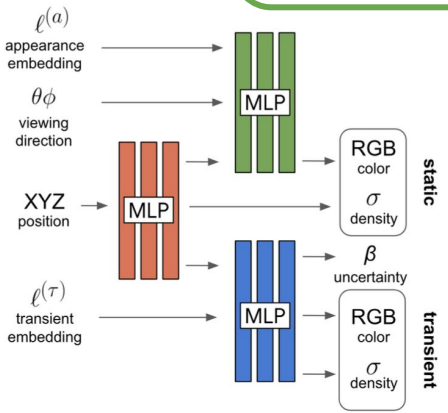
Urban Radiance Fields

- distractor가 특정 class(i.e. people)인 경우, segmentation으로 제거
- generalization X

D^2 NeRF

- <u>data in time</u> modeling
  - -> static/dynamic scene 분해
- video에만 적용





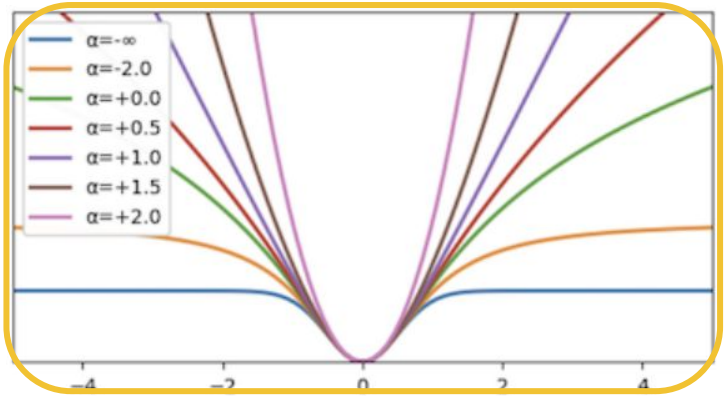
NeRF in the Wild

- transient/static modeling
- loss tuning 어려움

https://urban-radiance-fields.github.io/images/go_urf.pdf
https://arxiv.org/pdf/2008.02268
https://arxiv.org/pdf/2205.15838

PseudoLab

RobustNeRF: family of robust kernels



$$\mathcal{L}^{\mathbf{r},i}_{\text{robust}}(\boldsymbol{\theta}) = \kappa(||\mathbf{C}(\mathbf{r};\boldsymbol{\theta}) - \mathbf{C}_i(\mathbf{r})||_2)$$

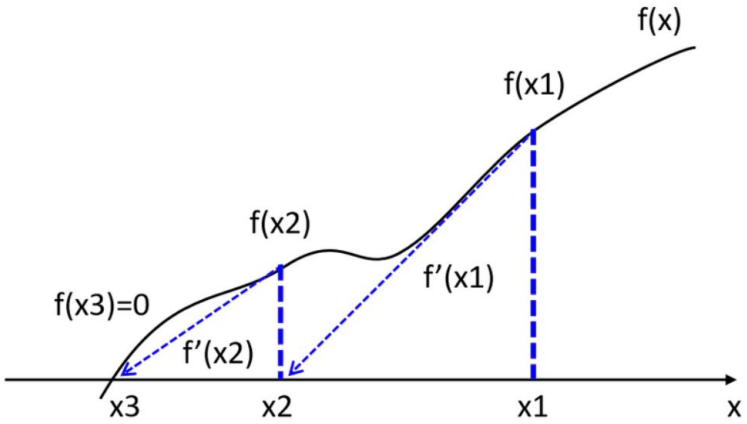
- κ(·): a robust kernel

Trade-off of Kernels

- A too aggressive Geman-McClure -> outlier와 high-frequency 정보 모두 제거
- A less aggressive Geman-McClure -> outlier 제거 X

PseudoLab

Iteratively Reweighted Least-sqaure



$$\mathbf{w}^{(\mathrm{new})} = \mathbf{w}^{(\mathrm{old})} - \mathbf{H}^{-1}\nabla E(\mathbf{w}).$$

IRLS(Newton-Raphson)

- E'(x)=0의 해를 찾기 위해, 반복적으로 아래 수식 적용하는 최적화
  - x1의 접선이 축과 만나는 점인 x2를 구함
  - 위의 과정 반복 -> 최적해 찾기
- error function(least-sqaure) 최소화

https://darkpgmr.tistory.com/142
https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf

PseudoLab

RobustNeRF: Robust loss

$$\left.\frac{\partial \kappa(\epsilon(\boldsymbol{\theta}))}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}^{(t)}} = \left.\frac{\partial \kappa(\epsilon)}{\partial \epsilon}\right|_{\epsilon(\boldsymbol{\theta}^{(t)})} \cdot \left.\frac{\partial \epsilon(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}^{(t)}}$$

- first term: kernel gradient
- second term: NeRF gradient

$$\omega(\epsilon) = \epsilon^{-1} \cdot \partial \kappa(\epsilon) / \partial \epsilon$$

- weight function
- 적용 시, local minima로 수렴

$$\mathcal{L}^{\mathbf{r},i}_{\text{robust}}(\boldsymbol{\theta}^{(t)}) = \omega(\boldsymbol{\epsilon}^{(t-1)}(\mathbf{r})) \cdot ||\mathbf{C}(\mathbf{r};\boldsymbol{\theta}^{(t)}) - \mathbf{C}_i(\mathbf{r})||_2^2$$

$$\boldsymbol{\epsilon}^{(t-1)}(\mathbf{r}) = ||\mathbf{C}(\mathbf{r};\boldsymbol{\theta}^{(t-1)}) - \mathbf{C}_i(\mathbf{r})||_2$$

- 설정한 weight를 IRLS로 최적화

PseudoLab

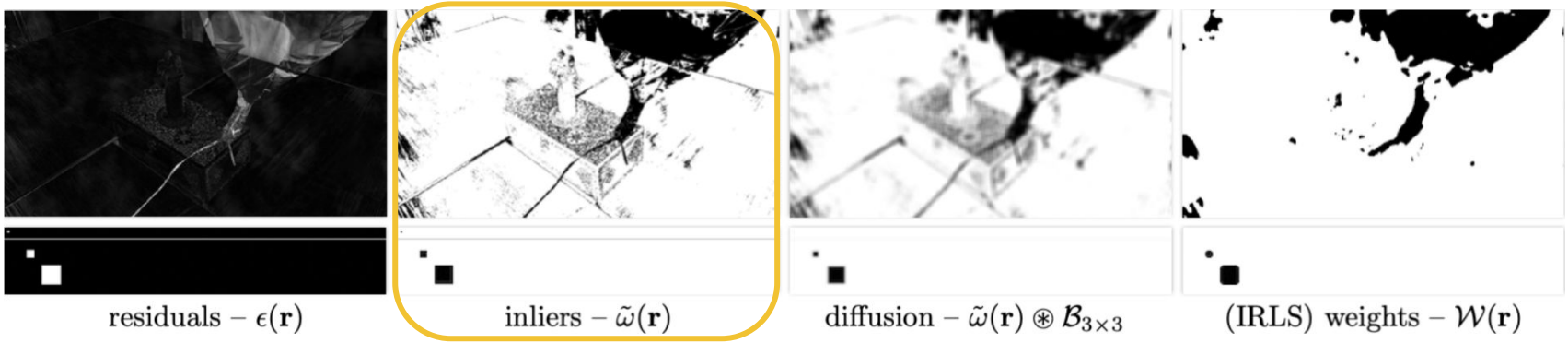
RobustNeRF: Trimmed/Diffused Least-square

$$\tilde{\omega}(\mathbf{r}) = \epsilon(\mathbf{r}) \leq \mathcal{T}_\epsilon\,, \quad \mathcal{T}_\epsilon = \text{Median}_\mathbf{r}\{\epsilon(\mathbf{r})\}$$

$$\mathcal{W}(\mathbf{r}) = \tilde{\omega}(\mathbf{r}) | (\tilde{\omega}(\mathbf{r}) \circledast \mathcal{B}_{3\times3}) \geq \mathcal{T}_\circledast\,, \quad \mathcal{T}_\circledast = 0.5$$

$$\omega(\mathcal{R}_8(\mathbf{r})) = \mathcal{W}(\mathbf{r}) | \mathbb{E}_{\mathbf{s}\sim\mathcal{R}_{16}(\mathbf{r})}\left[\mathcal{W}(\mathbf{s})\right] \geq \mathcal{T}_\mathcal{R}\,, \quad \mathcal{T}_\mathcal{R} = 0.6\,.$$

- residual sort -> 특정 percentile(median) 이내의 픽셀은 inlier로 추정



PseudoLab

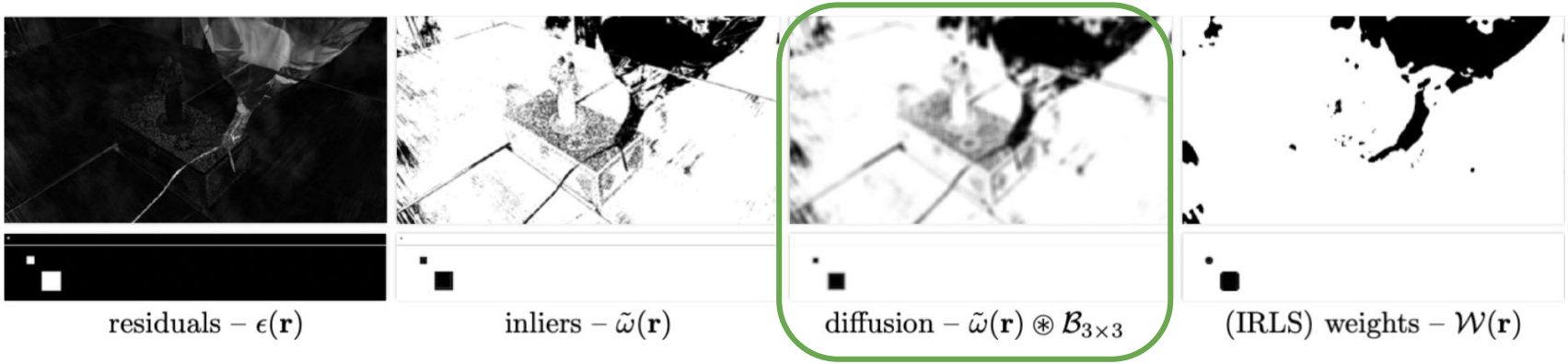
RobustNeRF: Trimmed/Diffused Least-square

$$\tilde{\omega}(\mathbf{r}) = \epsilon(\mathbf{r}) \leq \mathcal{T}_\epsilon\,, \quad \mathcal{T}_\epsilon = \text{Median}_\mathbf{r}\{\epsilon(\mathbf{r})\}$$

$$\mathcal{W}(\mathbf{r}) = \tilde{\omega}(\mathbf{r}) | (\tilde{\omega}(\mathbf{r}) \circledast \mathcal{B}_{3\times3}) \geq \mathcal{T}_\circledast\,, \quad \mathcal{T}_\circledast = 0.5$$

$$\omega(\mathcal{R}_8(\mathbf{r})) = \mathcal{W}(\mathbf{r}) | \mathbb{E}_{\mathbf{s}\sim\mathcal{R}_{16}(\mathbf{r})}[\mathcal{W}(\mathbf{s})] \geq \mathcal{T}_\mathcal{R}\,, \quad \mathcal{T}_\mathcal{R} = 0.6\,.$$

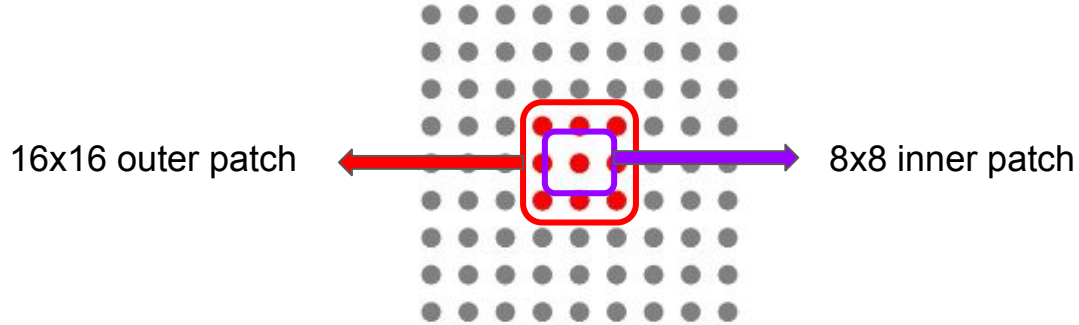
- 3x3 box kernel -> diffuse
- outlier가 spatial smoothness 특성을 가진다는 inductive bias 적용
- high-frequency detail을 outlier로 오분류 방지



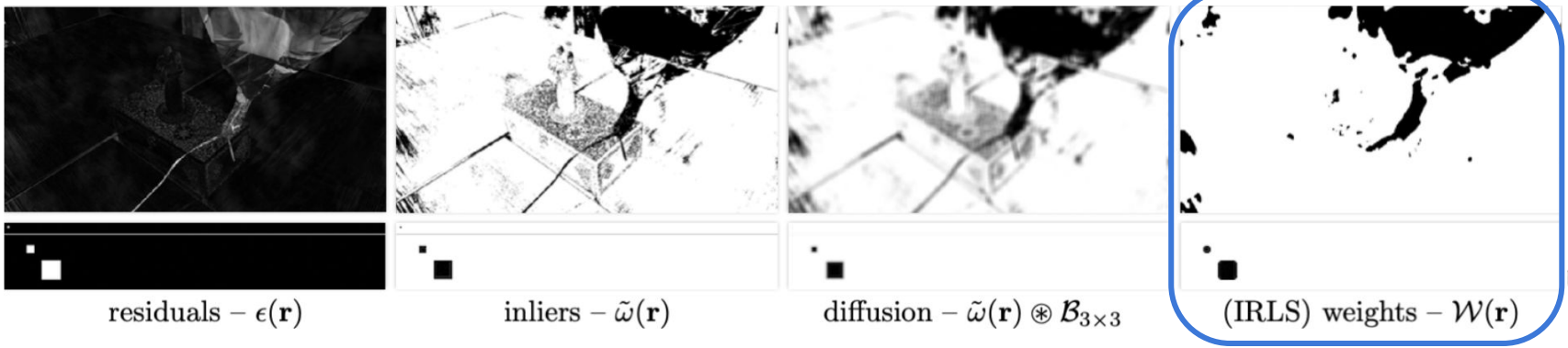
PseudoLab

RobustNeRF: Trimmed/Diffused Least-square

$$\omega(\mathcal{R}_8(\mathbf{r})) = \mathcal{W}(\mathbf{r}) | \mathbb{E}_{\mathbf{s}\sim\mathcal{R}_{16}(\mathbf{r})}\left[\mathcal{W}(\mathbf{s})\right] \geq \mathcal{T}_{\mathcal{R}}\,, \quad \mathcal{T}_{\mathcal{R}} = 0.6\,.$$

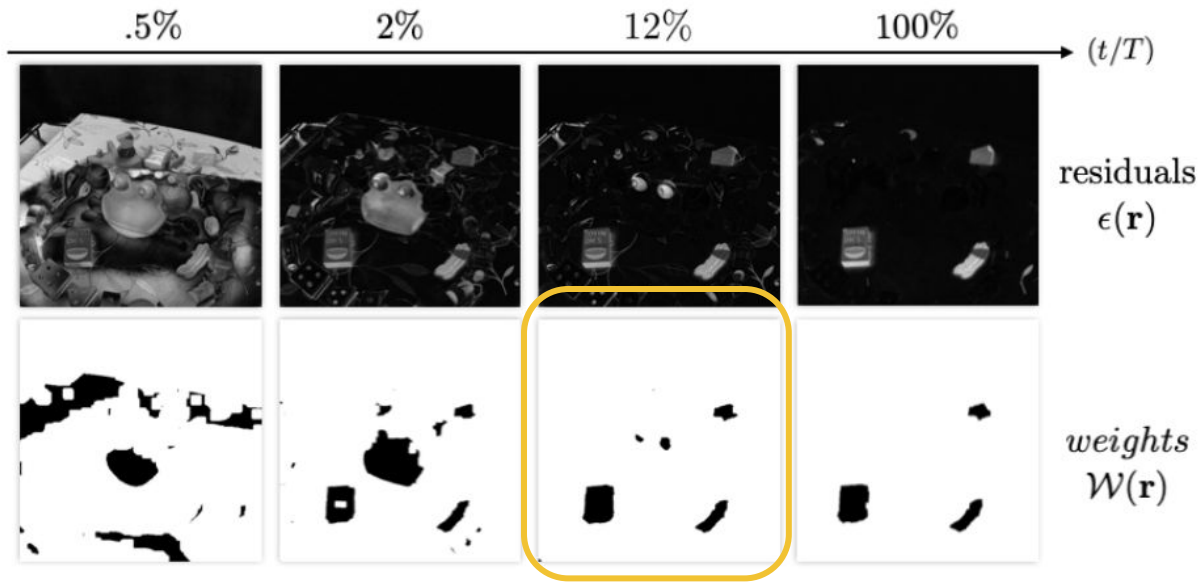


- 16x16 patch 내에 8x8 inner patch 생성
- inner patch의 주변 정보를 바탕으로 (~= patch avg) inner patch의 inlier/outlier 여부 추론
- 학습 초기(coarse)에 high-frequency 오분류 방지



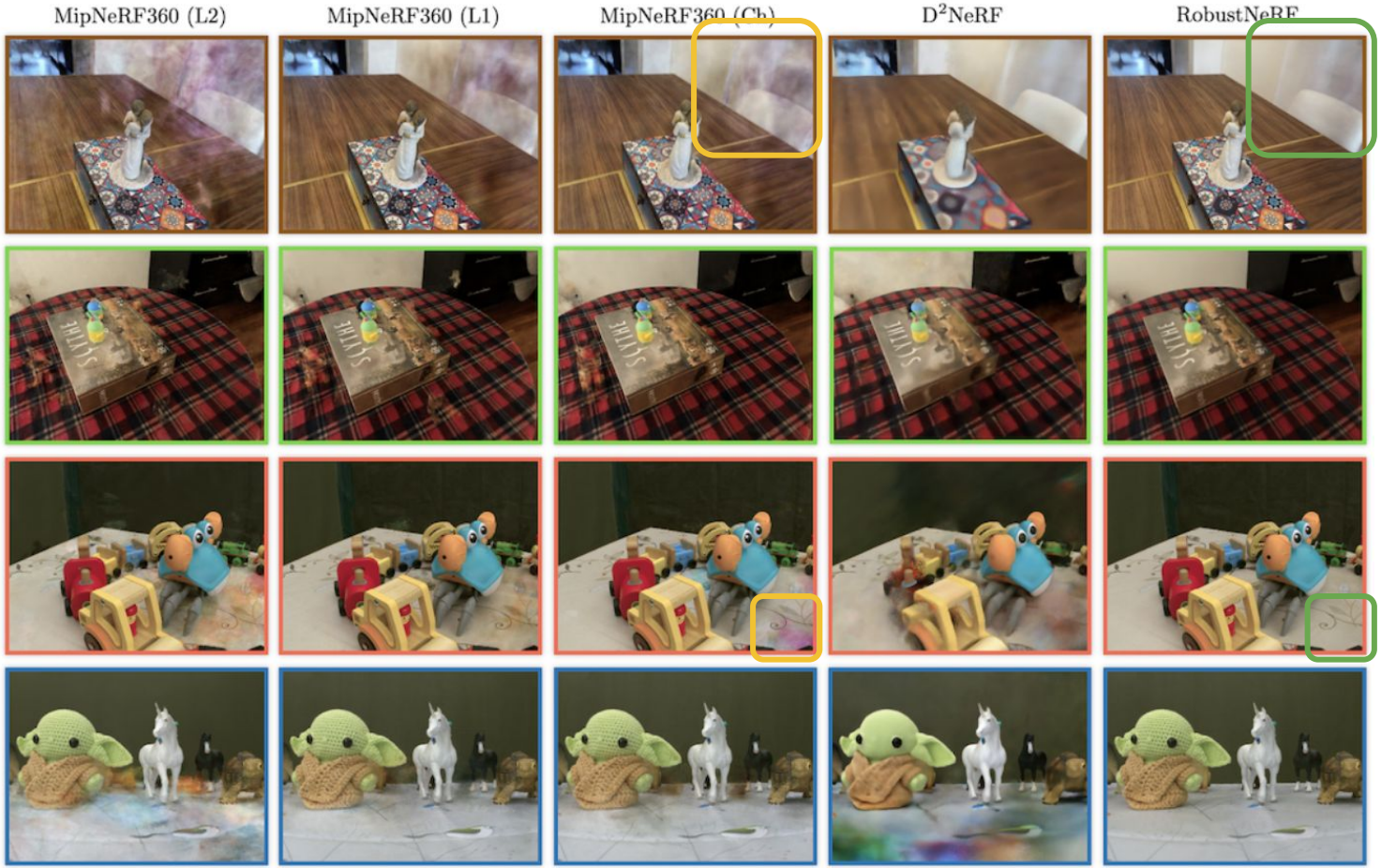
PseudoLab

RobustNeRF: Training



PseudoLab

Experiments: Natural Scenes



PseudoLab

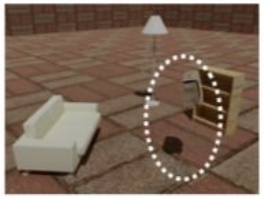
Experiments: Natural Scenes

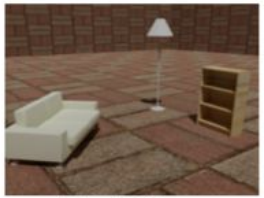
	Statue			Android			Crab			BabyYoda		
	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑
mip-NeRF 360 (L_2)	0.36	0.66	19.09	0.40	0.65	19.35	0.27	0.77	25.73	0.31	0.75	22.97
mip-NeRF 360 (L_1)	0.30	0.72	19.55	0.40	0.66	19.38	0.22	0.79	26.69	0.22	0.80	26.15
mip-NeRF 360 (Ch.)	0.30	0.73	19.64	0.40	0.66	19.53	0.21	0.80	27.72	0.23	0.80	25.22
D^2NeRF	0.48	0.49	19.09	0.43	0.57	20.61	0.42	0.68	21.18	0.44	0.65	17.32
RobustNeRF	**0.28**	**0.75**	**20.89**	**0.31**	**0.65**	**21.72**	**0.21**	**0.81**	**30.75**	**0.20**	**0.83**	**30.87**
mip-NeRF 360 (clean)	0.19	0.80	23.57	0.31	0.71	23.10	0.16	0.84	32.55	0.16	0.84	32.63

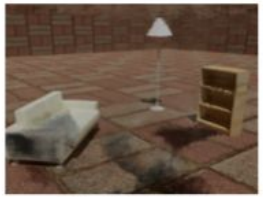
- **RobustNeRF는 outlier가 존재한다고 가정되어 있음**. -> limitation
- clean 버전일 때는, mip-NeRF 360가 더 좋은 성능을 보임

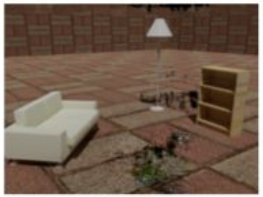
PseudoLab

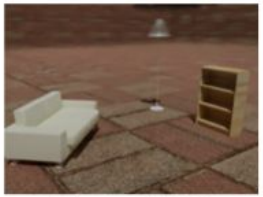
Experiments: Synthetic Scenes

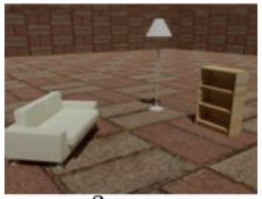

Train View

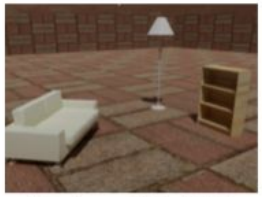

Test View


NeRF-W


NSFF


NeuralDiff


D^2NeRF


RobustNeRF

	Car			Cars			**Bag**			Chairs			Pillow		
	LPIPS↓	MS-SSIM↑	PSNR↑	LPIPS↓	MS-SSIM↑	PSNR↑	LPIPS↓	MS-SSIM↑	PSNR↑	LPIPS↓	MS-SSIM↑	PSNR↑	LPIPS↓	MS-SSIM↑	PSNR↑
NeRF-W [23]	.218	.814	24.23	.243	.873	24.51	.139	.791	20.65	.150	.681	23.77	.088	.935	28.24
NSFF [18]	.200	.806	24.90	.620	.376	10.29	.108	.892	25.62	.682	.284	12.82	.782	.343	4.55
NeuralDiff [46]	.065	.952	31.89	.098	.921	25.93	.117	.910	29.02	.112	.722	24.42	.565	.652	20.09
D^2NeRF [53]	.062	.975	34.27	.090	.953	26.27	.076	.979	34.14	.095	.707	24.63	.076	.979	36.58
RobustNeRF	**.013**	**.988**	**37.73**	**.063**	**.957**	**26.31**	**.006**	**.995**	**41.82**	**.007**	**.992**	**41.23**	**.018**	**.990**	**38.95**

PseudoLab

Experiments: Ablatations

	LPIPS↓	SSIM↑	PSNR↑	Updates to PSNR=30
mip-NeRF 360 (L_2)	0.31	0.75	22.97	–
+ robust (8)	0.39	0.60	18.21	–
+ smoothing (9)	0.22	0.81	30.01	250K
+ patching (10)	**0.21**	**0.81**	**30.75**	70K
oracle (clean)	0.16	0.84	32.55	25K

+ robust + smoothing + patching

- robust만 추가 시 성능 하락
- smoothing (diffuse) 추가했을 때 가장 높은 성능 향상폭 가짐

PseudoLab

Q&A

PseudoLab

IE-NeRF: Inpainting Enhanced Neural Radiance Fields in the Wild

NeRF with Real-World in 가짜연구소

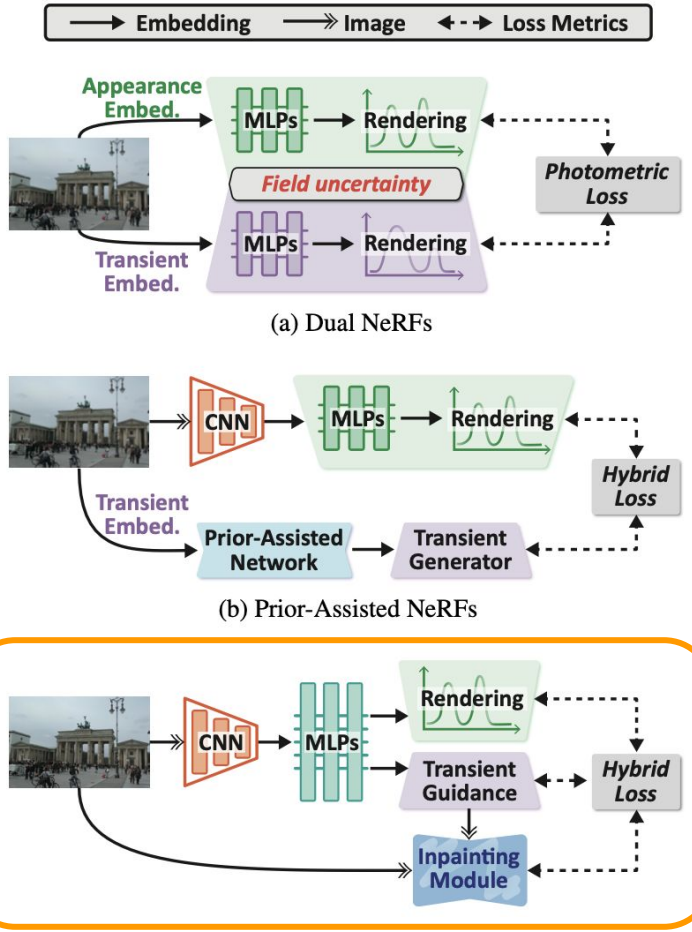
by minsol Kim

https://arxiv.org/pdf/2407.10695

PseudoLab

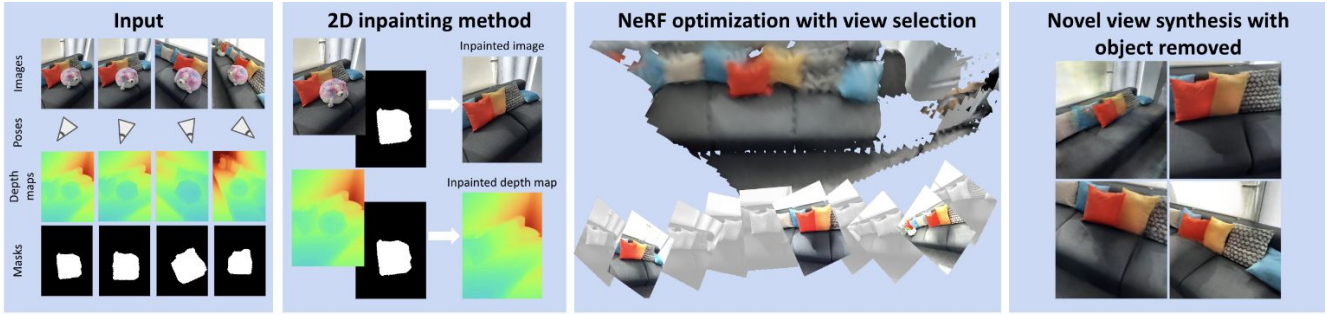
Introduction

- 하나의 NeRF MLP로 instrinsic(color, density), extrinsic mask 생성
- 추출한 mask에 inpainting
  - transient mask로 occlusion 제거
  - inpainting module: LaMa
- Photometric loss 적용



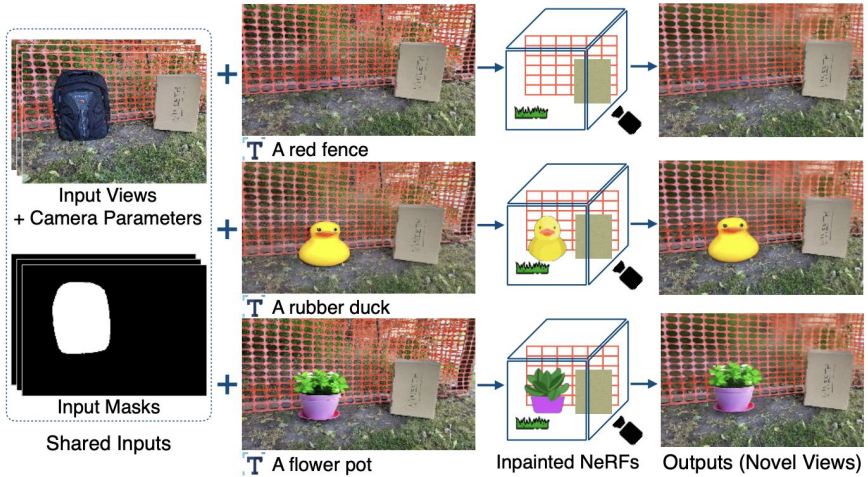
PseudoLab

Related Works



Removing Objects From Neural Radiance Fields

- RGB-D sequence data로부터 distractors 제거
- RGB 이미지와 Depth 이미지에 LaMa inpainting을 적용
- 이후, NeRF 모델 최적화

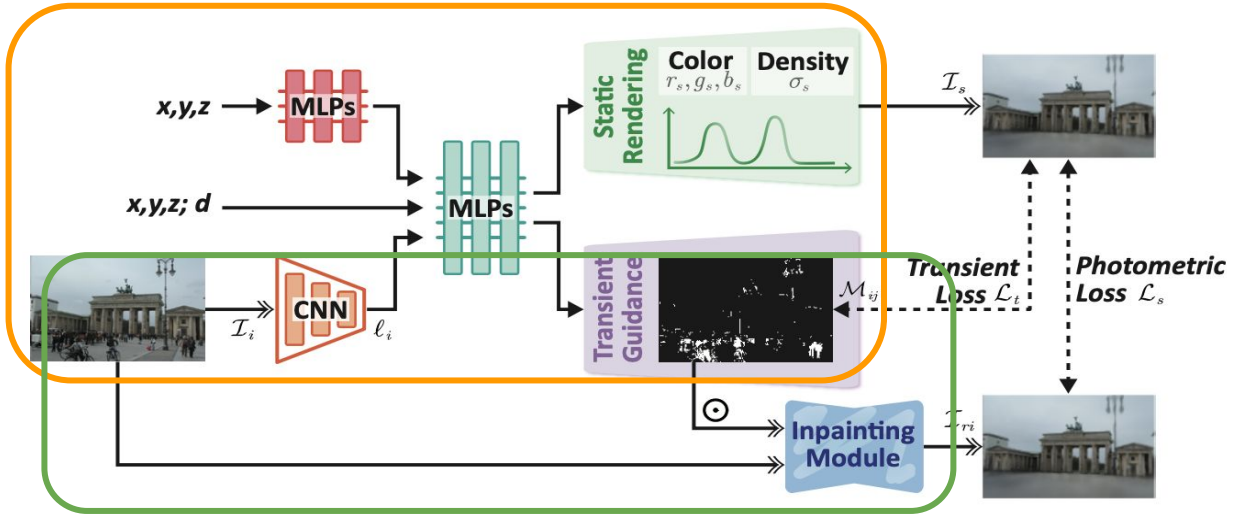


SPIn-NeRF

- image inpainting으로 geometry도 가이드
  - multi-view mask
- NeRF 3D inpainting

PseudoLab

IE-NeRF



Network

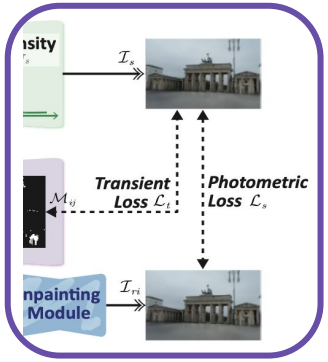
- Inputs:
  - pos, dir
  - appearance embedding from CNN
- Static Rendering, Mask Generation까지 Ha-NeRF와 동일
  - but, MLP를 하나만 사용

Inpainting

- mask 추출
  - inputs: appearance emb, pixel locations
- pretrained LaMa 모듈 사용
  - large mask에 robust + 가성비 (less params, time)
  - input: x * m + m (4 ch)

PseudoLab

IE-NeRF



Optimization

- **Scene loss**:
  - inpained image의 color와 coarse/fine rendered color 간의 MSE
- **Transient loss**:
  - first term:
    - original color <-> rendered color
    - distractor 존재하는지 판단
  - second term:
    - inpained color <-> rendered color
    - balancing static and transient elements

$$\mathcal{L}_s = \left\| C_s(r_{ij}) - \hat{C}_{ci}(r_{ij}) \right\|_2^2 + \left\| C_s(r_{ij}) - \hat{C}_{fi}(r_{ij}) \right\|_2^2$$

$$\mathcal{L}_t = (1 - \mathcal{M}_{ij}) \left\| C(r_{ij}) - \hat{C}_s(r_{ij}) \right\|_2^2 + \lambda \mathcal{M}_{ij} \left\| C_s(r_{ij}) - \hat{C}_s(r_{ij}) \right\|_2^2.$$

$$\mathcal{L} = \sum_{ij} \mathcal{L}_s + \beta \sum_{ij} \mathcal{L}_t.$$

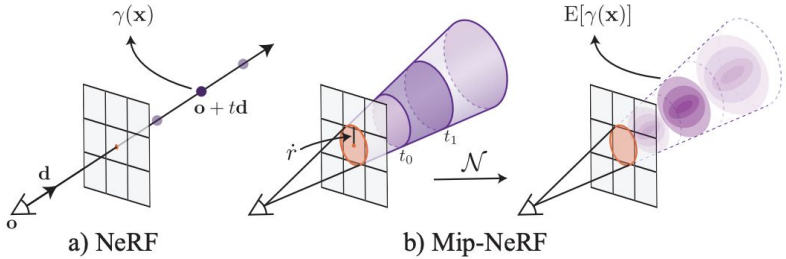
PseudoLab

IE-NeRF

Integrated Positional Encoding (IPE)

- PE integrating within conical frustum
  - multiscale representation 학습
  - high-frequency
  - as Mip-NeRF

- low-frequency에서 distractor 존재하는 경우 발생
  - PE를 점진적으로 적용 (=regularization)



https://arxiv.org/pdf/2103.13415

PseudoLab

Results

	Brandenburg Gate			Sacre Coeur			Trevi Fountain			Taj Mahal		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
NeRF	18.90	0.816	0.232	15.60	0.716	0.292	16.14	0.601	0.366	15.77	0.697	0.427
NeRF-W	24.17	0.891	0.167	19.20	0.807	0.192	18.97	0.698	0.265	**26.36**	**0.904**	0.207
Ha-NeRF	24.04	0.877	**0.139**	20.02	0.801	0.171	20.18	0.691	0.223	19.82	0.829	0.243
SF-NeRF (30-fews)	23.23	0.846	0.178	19.64	0.757	0.186	20.24	0.657	0.243	20.86	0.820	0.218
IE-NeRF (Ours)	**25.33**	**0.898**	0.158	**20.37**	**0.861**	**0.169**	**20.76**	**0.719**	**0.217**	25.86	0.889	**0.196**

Quantitative

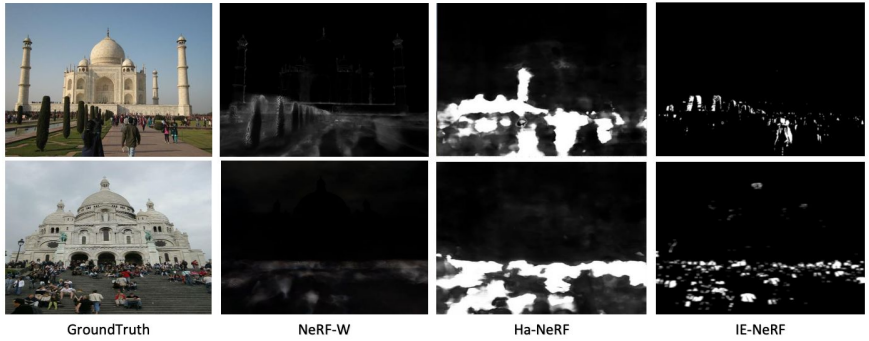
- Phototourism 데이터셋에 대한 평가
- 기존 NeRF들에 비해 더 높은 성능 기록
- WildGaussians 비교표로 확인하면, 중간 정도의 성능

	GPU hrs./ FPS	Brandenburg Gate			Sacre Coeur			Trevi Fountain		
		PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF [25]	-/<1	18.90	0.815	0.231	15.60	0.715	0.291	16.14	0.600	0.366
NeRF-W-re [24]	164/<1	24.17	0.890	0.167	19.20	0.807	0.191	18.97	0.698	0.265
Ha-NeRF [4]	452/<1	24.04	0.877	0.139	20.02	0.801	0.171	20.18	0.690	0.222
K-Planes [9]	0.6/<1	25.49	0.879	0.224	20.61	0.774	0.265	22.67	0.714	0.317
RefinedFields [13]	150/<1	26.64	0.886	-	22.26	0.817	-	23.42	0.737	-
3DGS [14]	2.2/57	19.37	0.880	0.141	17.44	0.835	0.204	17.58	0.709	0.266
GS-W† [52]	1.2/51	23.51	0.897	0.166	19.39	0.825	0.211	20.06	0.723	0.274
SWAG* [5]	0.8/15	26.33	0.929	0.139	21.16	0.860	0.185	23.10	0.815	0.208
Ours	7.2/117	27.77	0.927	0.133	22.56	0.859	0.177	23.63	0.766	0.228

https://arxiv.org/pdf/2407.08447

PseudoLab

Results



Qualitative

- NeRF-W, Ha-NeRF에 비해 깔끔한 마스킹
- 비교군에 비해 퀄리티 좋지만, 여전히 **floater** 발생

PseudoLab

Results

Method	PSNR↑	SSIM↑	LPIPS↓
IE-NeRF(IM)	22.27	0.802	0.191
IE-NeRF(SM)	24.65	0.879	0.187
IE-NeRF(Ours)	**25.33**	**0.898**	**0.158**

Training strategy	PSNR↑	SSIM↑	LPIPS↓
IPE	23.58	0.864	0.191
WT-IPE	19.86	0.723	0.289
RegFre-IPE	**25.33**	**0.898**	**0.158**

Ablations

- inpainting:
  - IM: an independent MLP
  - SM: pre-defined instance segmentation model (MaskDINO)
- IPE:
  - vanila IPE: Mip-NeRF 방식
  - WT-IPE: wavelet transform
  - RegFre-IPE: IE-NeRF 방식 (= IPE + regularization)

PseudoLab

Conclusion

Limitations

- transient mask가 충분한 정보를 담지 못함
  - in case:
    - small datasets
    - sparse inputs
- 현재 기술들과 비교 시, SOTA의 성능은 X

Contributions

- image inpainting으로 in-the-wild 렌더링 수행
- 좋은 퀄리티의 mask가 생성된다면, 성능이 올라갈 수 있음

PseudoLab

Q&A

PseudoLab

NeRF-HuGS: Improved Neural Radiance Fields in Non-static Scenes Using Heuristics-Guided Segmentation

Jiahao Chen[1], Yipeng Qin[2], Lingjie Liu[3], Jiangbo Lu[4], Guanbin Li[1]

[1]Sun Yat-sen University [2]Cardiff University [3]University of Pennsylvania [4]SmartMore Corporation

논문 리뷰 PseudoLab 9기 홍세준

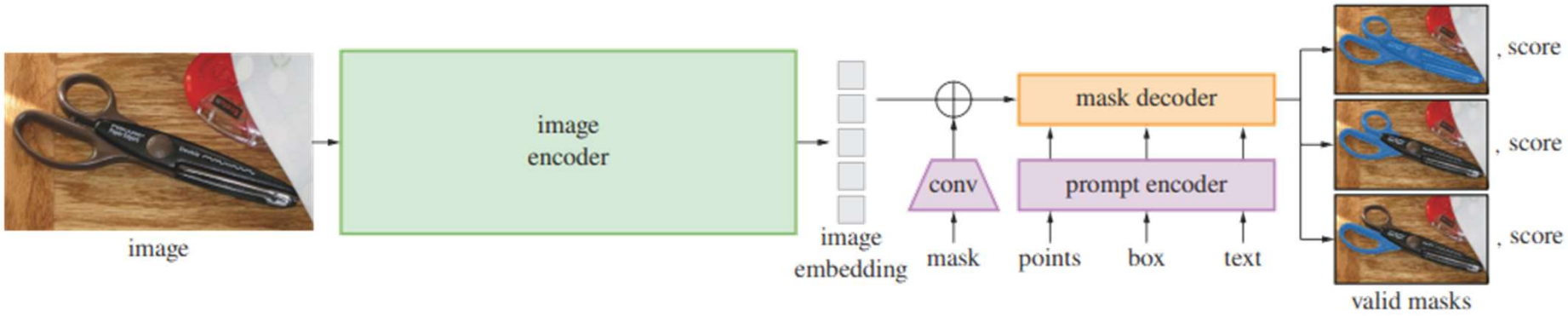
NeRFwithRealWorld

Introduction

- “transient distractors”

1) Leverage pre-trained segmentation models to detect transient distractorss

ex) Segment Anything (SAM)



→ Limited generality as it relies on additional prerequisites of prior knowledge

Introduction

- “transient distractors”

2) Separate transient distractors from static scenes through hand-crafted heuristics

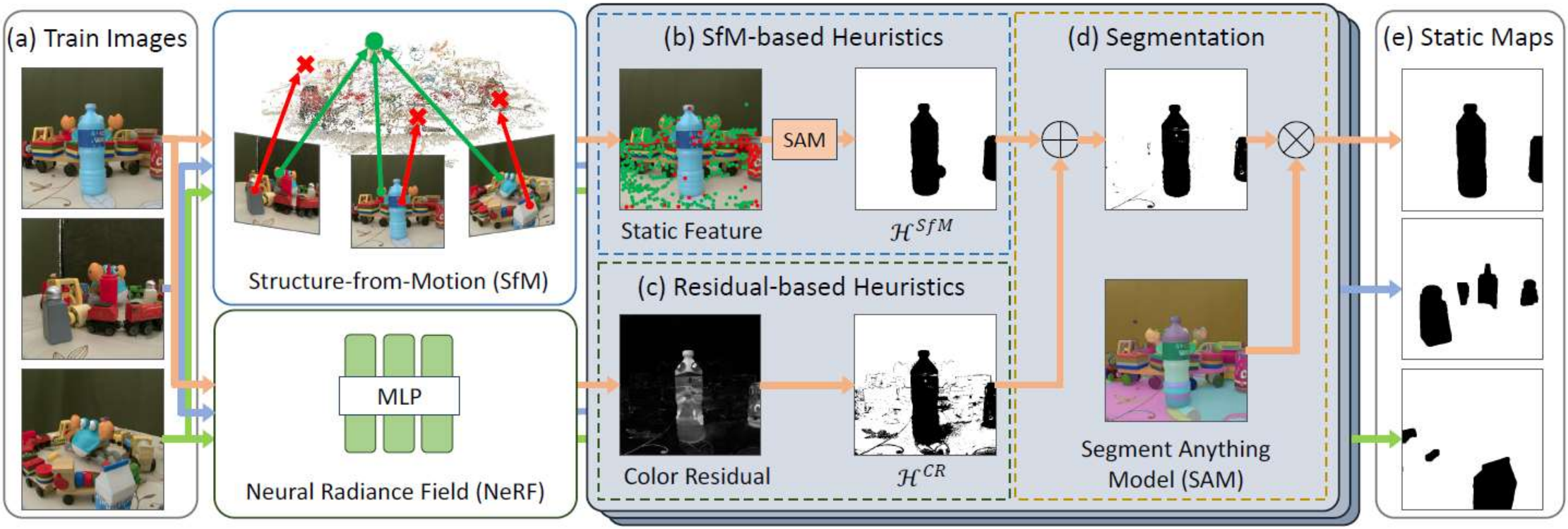
ex) NeRF in the Wild (NeRF-W)

(a) Static (b) Transient (c) Composite (d) Image (e) Uncertainty

→ Yield imprecise or erroness results

Method

- **NeRF-HuGS Framework**



Method

- **Structure-from-Motion (SfM)**

$$\mathcal{I} = \{I_i \mid i = 1, 2, \ldots, N_I\}$$

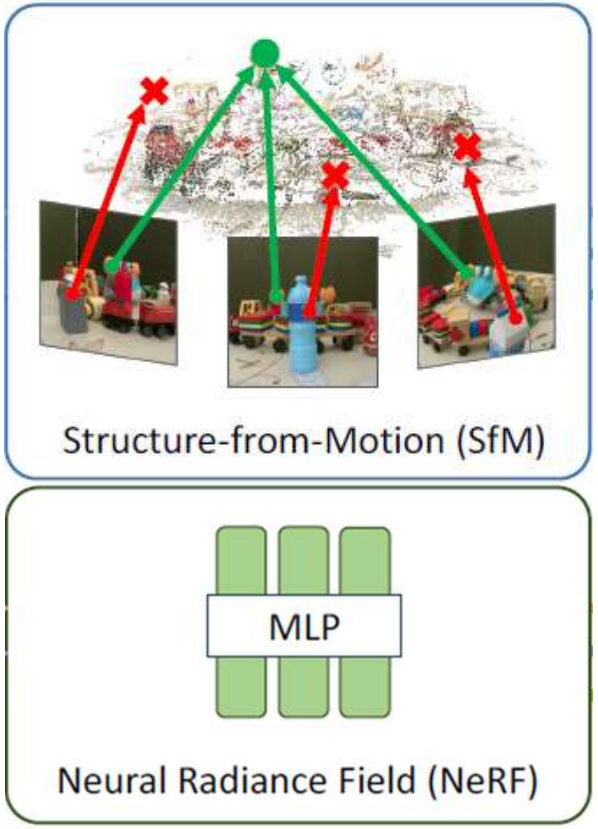
2D Local Feature point $\mathcal{F}_i = \left\{ \left(\mathbf{x}_i^j, \mathbf{f}_i^j \right) \mid j = 1, 2, \ldots, N_{F_i} \right\}$

- $\mathbf{f}_i^j$: appearance descriptor / $\mathbf{x}_i^j$: coordinate

- **NeRF in Static Scenes**

$$\mathcal{L}(\mathbf{r}) = M(\mathbf{r}) \mathcal{L}_{\text{recon}}(\hat{\mathbf{C}}(\mathbf{r}), \mathbf{C}(\mathbf{r}))$$

- M_i : static map



Method

1) SfM-based Heuristics

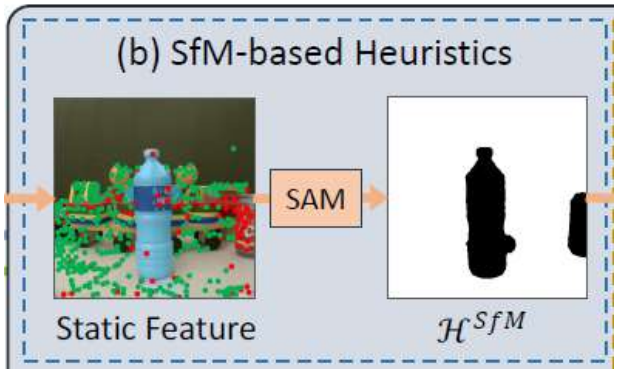
"*minority*" : frequency of occurrence (o)

spatial area coverage (x)

Observation 1. *The SfM features of static objects have much larger* n_i^j *than those of transient ones in image* I_i.

$$\mathcal{X}_i = \left\{ \mathbf{x}_i^j \;\middle|\; \mathbf{x}_i^j \in \mathcal{F}_i \text{ and } \frac{n_i^j}{N_I} \geq \mathcal{T}_{SfM} \right\}$$

$\mathcal{X}_i$ into SAM to obtain heuristics $\mathcal{H}_i^{SfM}$

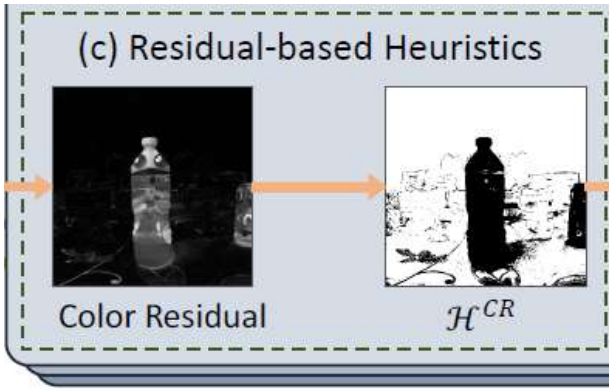


Method

2) Residual-based Heuristics

: color residual map $\mathcal{R}_i$

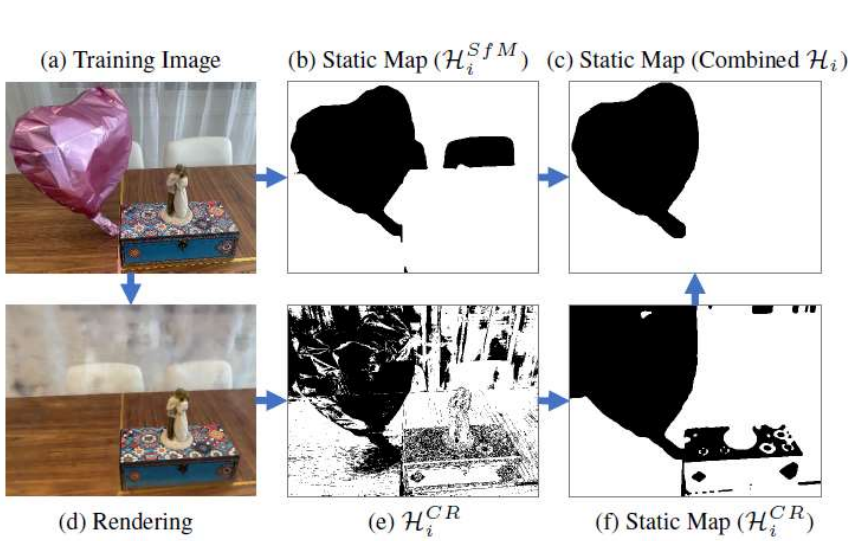
using the color residual for each ray **r** as $\epsilon(\mathbf{r}) = \|\hat{\mathbf{C}}(\mathbf{r}) - \mathbf{C}(\mathbf{r})\|_2$ (Nerfacto)

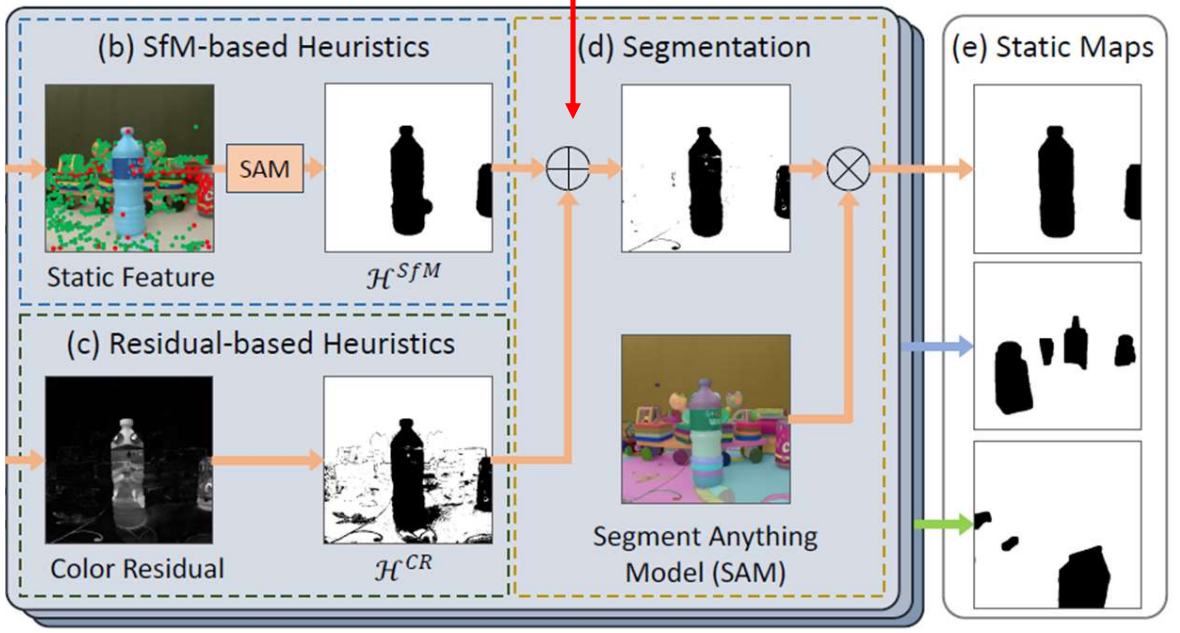


Method

3) Combined Heuristics

$$\hat{\mathcal{H}}_i = \mathcal{H}_i^{SfM} \cup \mathcal{H}_i^{CR} \quad \text{where } \mathcal{H}_i^{CR} = \mathcal{R}_i \leq \text{mean}(\mathcal{R}_i)$$





$$\mathcal{H}_i = \hat{\mathcal{H}}_i \cap \hat{\mathcal{H}}_i^{CR} \quad \text{where } \hat{\mathcal{H}}_i^{CR} = \mathcal{R}_i \leq \text{quantile}(\mathcal{R}_i, \mathcal{T}_{CR})$$

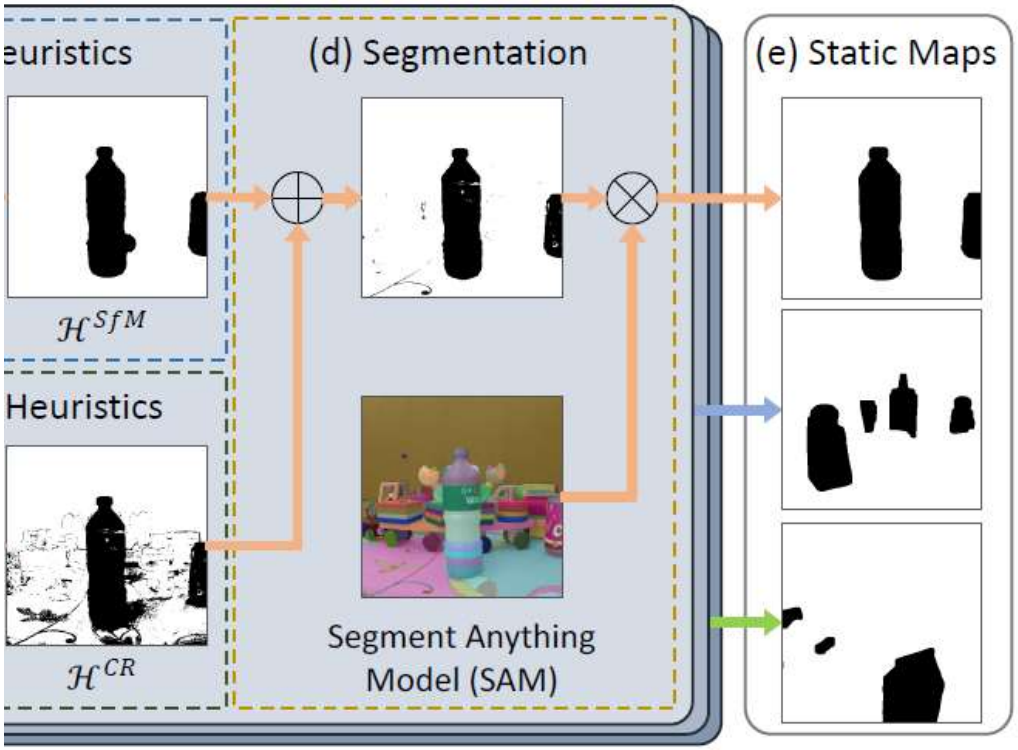
NeRFwithRealWorld

Method

4) Segmentation

$$S(I_i) = \{m_i^1, m_i^2, ..., m_i^{N_{M_i}}\}$$

$$M_i = \bigcup m_i^j, \ \forall \frac{m_i^j \cap \mathcal{H}_i}{m_i^j} \geq \mathcal{T}_m$$



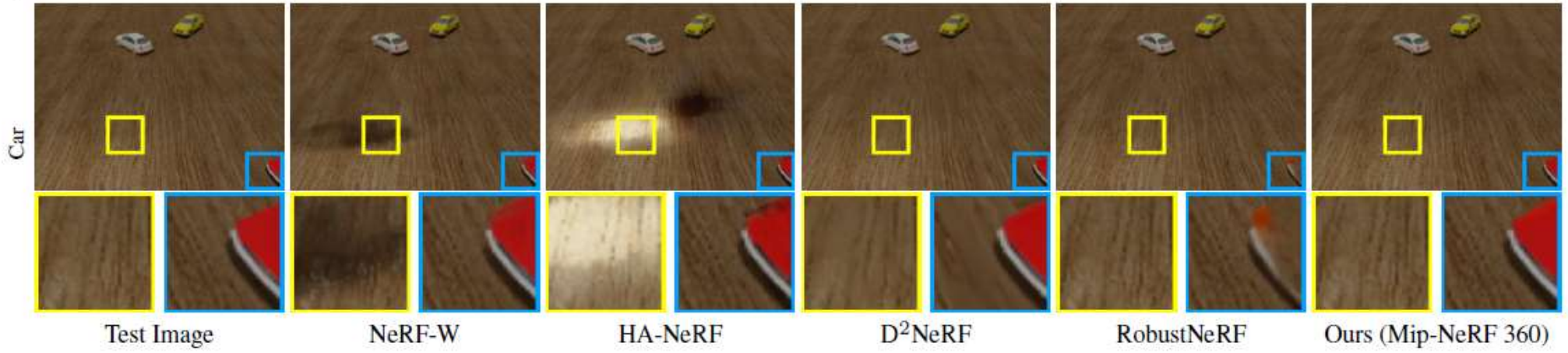
Experiments

- **Datasets**: Kubric, Distractor, Phototourism
- **Threshold**: $\mathcal{T}_m$ = 0.5 / $\mathcal{T}_{SfM}$ and $\mathcal{T}_{CR}$ (depending on datasets)
- **Baselines**: NeRF-W, HA-NeRF, RobustNeRF, D^{2}NeRF (View Synthesis)
- DeepLabv3+, Mask2Former, Grounded-SAM, DINO (Segmentation)

Experiments

- Results (Kubric dataset)

Method	Car			Cars			Bag			Chairs			Pillow			Avg.		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
Nerfacto [46]	30.71	.862	.227	29.50	.809	.316	31.32	.917	.103	27.41	.811	.258	29.86	.906	.148	29.76	.861	.210
Mip-NeRF 360 [2]	26.67	.846	.238	28.88	.822	.281	32.81	.948	.053	27.07	.839	.179	29.66	.919	.123	29.02	.875	.175
NeRF-W [28]	29.44	.901	.124	28.34	.867	.186	34.49	.946	.045	22.75	.826	.187	29.04	.915	.142	28.81	.891	.137
HA-NeRF [7]	28.69	.915	.124	31.95	.903	.143	38.48	.969	.021	33.48	.922	.071	31.66	.946	.083	32.85	.931	.089
D^{2}NeRF [53]	34.03	.874	.099	33.67	.844	.123	33.77	.889	.118	32.77	.875	.113	29.49	.907	.139	32.75	.878	.118
RobustNeRF [40]	37.31	.968	.040	40.52	.963	.047	40.50	.976	.026	38.56	.958	.037	41.31	.980	.028	39.64	.969	.036
Ours (Nerfacto)	39.49	.964	.042	39.95	.958	.045	41.39	.980	.017	38.48	.962	.036	42.70	.982	.025	40.40	.969	.033
Ours (Mip-NeRF 360)	39.75	.972	.036	40.74	.966	.046	42.32	.983	.019	39.32	.968	.033	43.90	.986	.023	41.21	.975	.032



NeRFwithRealWorld

Experiments

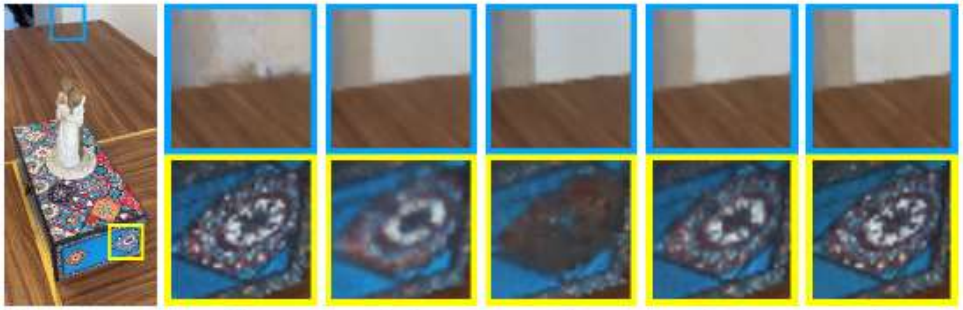
- Results (segmentation on Kubric dataset)

Method	Seg. Type	Require Prior	Car		Cars		Bag		Chairs		Pillow		Avg.	
			mIoU↑	F1↑	mIoU↑	F1↑	mIoU↑	F1↑	mIoU↑	F1↑	mIoU↑	F1↑	mIoU↑	F1↑
DeepLabv3+ [6]	*Semantic*	✓	.604	.378	.578	.293	.501	.048	.564	.239	.535	.149	.556	.221
Mask2Former [8]	*Semantic*	✓	.664	.520	.622	.376	.513	.071	.707	.561	.653	.377	.632	.381
Grounded-SAM [16, 25]	*Open-Set*	✓	.888	.877	.640	.406	.755	.671	.603	.373	**.851**	**.828**	.747	.631
DINO [4]	*Video*	✓	**.947**	**.947**	.720	.557	.591	.367	**.777**	**.703**	**.911**	**.904**	.789	.695
NeRF-W [28]	/	✗	.682	.575	.584	.328	.526	.099	.547	.298	.557	.296	.579	.319
HA-NeRF [7]	/	✗	.869	.852	**.823**	**.724**	**.813**	**.771**	.729	.595	.819	.766	**.811**	**.742**
D^2NeRF [53]	/	✗	**.912**	**.909**	**.895**	**.867**	**.794**	**.727**	.660	.507	.800	.760	**.812**	**.754**
RobustNeRF [40]	/	✗	.813	.784	.718	.547	.731	.633	**.731**	**.638**	.724	.633	.743	.647
HuGS (Ours)	/	✗	**.963**	**.964**	**.940**	**.907**	**.939**	**.935**	**.937**	**.927**	**.940**	**.937**	**.944**	**.934**

Experiments

- Ablation Results

	$\mathcal{H}^{SfM}$	$\mathcal{H}^{CR}$	SAM	Kubric (Avg.) PSNR↑	SSIM↑	LPIPS↓	Distractor (Avg.) PSNR↑	SSIM↑	LPIPS↓
(a)	-	-	-	29.76	.861	.210	22.06	.692	.241
(b)	✓	-	-	38.14	.962	.054	25.67	.788	.108
(c)	-	✓	-	39.86	.967	.036	24.95	.767	.146
(d)	-	✓	✓	40.12	.968	.033	24.58	.779	.126
(e)	✓	✓	-	40.11	.968	.035	25.40	.786	.116
(f)	✓	✓	✓	40.40	.969	.033	25.66	.791	.106



Test Image (b) (c) (d) (e) (f)

Q & A

Arxiv 2024

Wild-GS: Real-Time Novel View Synthesis from Unconstrained Photo Collections

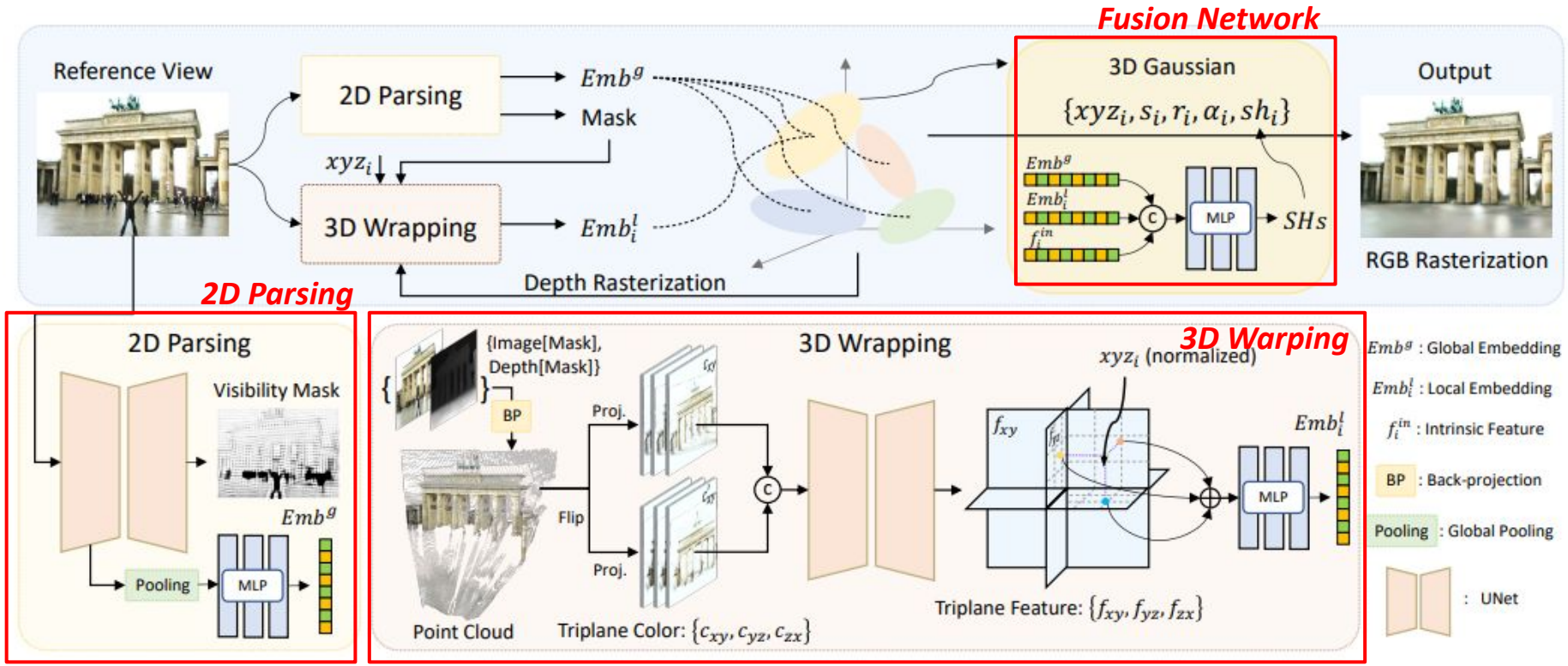
Presenter: Seongjun Choi

sjchoi.dp@gmail.com

NeRF with Real World, 가짜연구소

Contents

Architecture

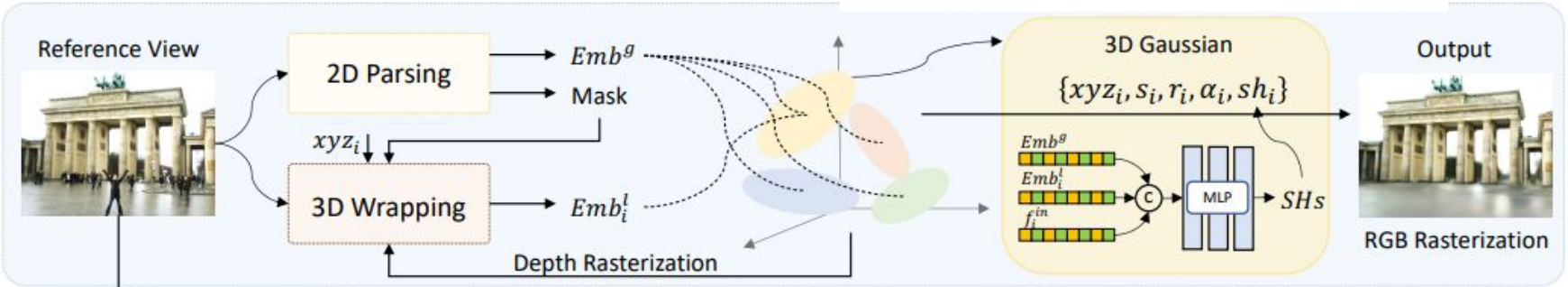


Hierarchical Appearance Modeling

- **(a) Global appearance embedding Emb^g** capturing the illumination level or tone mapping of the entire scene;
- **(b) Local appearance embedding Emb_i^l** describing the positional-aware local reflectance for i-th 3D Gaussian;
- **(c) Intrinsic feature f in i** storing the inherent attributes of the material in the dominant area for each Gaussian.

Before rasterization, a shared fusion network MF θ is leveraged to decode the view-dependent color sh from these three appearance components:

$$sh_i = M_\theta^F(Emb^g \oplus Emb_i^l \oplus f_i^{in}),$$



Hierarchical Appearance Modeling

(a) Global appearance embedding Emb^g

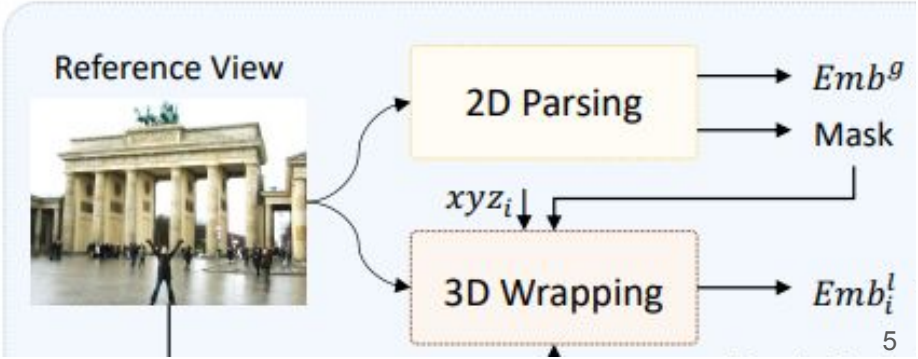
the low-frequency appearance changes among the entire scene.

$$Emb^g = M_\theta^G(AvgPooling(F_{I_R})).$$

- **F_I_R:** the feature maps obtained from the UNet encoder(ResNet-18 pre-trained by ImageNet) in the 2D Parsing module
- **AvgPooling:** global average pooling
- **M^G_θ:** a trainable MLP

(c) Intrinsic feature f in i

- a learnable intrinsic appearance feature
- which remain consistent despite environmental changes.
- Inspired by EAGLES (Girish et al., 2023), which compresses the attributes of the 3D Gaussian into a low-dimensional latent vector.



Hierarchical Appearance Modeling

(b) Local appearance embedding Emb_i^l

2D -> 3D: back-project the reference image *I_R* into the space using the rendered depth D^_I_R and camera parameters *ω_I_R*

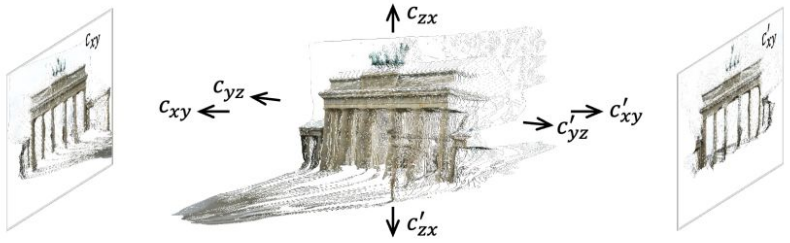
$$\{C_{I_R}, P_{I_R}\} = BP(I_R[M_{I_R} > Th], \hat{D}_{I_R}[M_{I_R} > Th], \omega_{I_R}),$$

- *M_I_R:* visibility mask
- *Th:* threshold distinguishing the transient objects
- {CIR , PIR} refers to the generated point cloud with positions PIR and colors CIR

3D -> 2D(triplane): the 3D points are normalized for re-projection onto the **three orthogonal planes defined by the triplane**

$$\{c_{xy}, c_{yz}, c_{zx}\} = Proj(C_{I_R}, \widetilde{P}_{I_R}),$$

where $\widetilde{P}_{I_R}$ represents the normalized point positions.



(a) Point cloud re-projection

$$F^T_{I_R} = U^{3D}_\theta(\{c_{xy}, c_{yz}, c_{zx}\} \oplus \{c'_{xy}, c'_{yz}, c'_{zx}\}).$$

- **{c'_xy, c'_yz, c'_zx}:** triplane color of each axis corresponding reverses
- **UNet U^3D_θ:** to extract the triplane feature maps F^T_I_R

$$Emb^l_i = M^L_\theta(f^i_{xy} + f^i_{yz} + f^i_{zx}).$$

The positional-awareness local appearance embedding utilized to interpolate the three plane features {fixy, fiyz, fizx} for each 3D Gaussian.

Depth Regularization

Depth information는 the sparse view reconstruction에 종종 사용함(Deng et al., 2022; Li et al., 2024; Zhu et al., 2023)

Wild-GS에서 렌더링된 깊이를 활용하여 참조 뷰를 back project 함

이때, 각 reference view에서 깊이 정보를 추정하기 위해 Depth Anything (Yang et al., 2024) 사용함

Pearson correlation loss(proposed by FSGS (Zhu et al., 2023) 를 약간 수정하여 사용함.

$$\hat{D}_{I_R} = \sum_{i=1}^{n} d_i \alpha'_i \prod_{j=1}^{i-1} (1 - \alpha'_j),$$

$$\mathcal{L}^D = \frac{Cov(\hat{D}_{I_R}[M_{I_R} > Th], D^{Est}_{I_R}[M_{I_R} > Th])}{\sqrt{Var(\hat{D}_{I_R}[M_{I_R} > Th]) \cdot Var(D^{Est}_{I_R}[M_{I_R} > Th])}}.$$

Handling Transient Objects

$$\mathcal{L}^I = \lambda^I |I_R \odot M_{I_R} - \hat{I}_R \odot M_{I_R}| + (1 - \lambda^I) \cdot SSIM(I_R \odot M_{I_R}, \hat{I}_R \odot M_{I_R}).$$

$$\mathcal{L}^M = (1 - M_{I_R})^2.$$

$$\mathcal{L}_{total} = \mathcal{L}^I + \lambda^M \mathcal{L}^M + \lambda^D \mathcal{L}^D.$$

λM is linearly reduced from 0.4 to 0.1 to stabilize the training process, while λD is kept constant 0.05 during the entire training process.

Experimental & Results

- we evaluate different methods on three in-the-wild datasets: "Brandenburg Gate", "Sacre Coeur", and "Trevi Fountain" extracted from the Phototourism dataset and downsample the images by 2 times (R/2).
- All the training times and inference speeds are tested on a single RTX3090 for fair comparison.

Experimental & Results

Method	Brandenburg Gate			Sacre Coeur			Trevi Fountain			Efficiency	
	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	Training	Inference
3DGS	19.63	0.8817	0.1378	17.95	0.8455	0.1633	17.23	0.6963	0.2815	**0.12**	220
NeRF-W	24.17	0.8905	0.1670	19.20	0.8076	0.1915	18.97	0.6984	0.2652	60.3	0.05
Ha-NeRF	24.04	0.8873	0.1391	20.02	0.8012	0.1710	20.18	0.6908	0.2225	71.6	0.05
CR-NeRF	26.53	0.9003	0.1060	22.07	0.8233	0.1520	21.48	0.7117	0.2069	101	0.02
Wild-GS†	27.81	0.9180	0.1085	23.85	0.8567	0.1575	22.77	0.7629	0.2229	0.23	225
Wild-GS	**29.65**	**0.9333**	**0.0951**	**24.99**	**0.8776**	**0.1270**	**24.45**	**0.8081**	**0.1622**	0.52	**227**
w/o Crop	29.15	0.9324	0.0968	24.93	0.8761	0.1323	23.61	0.7915	0.1856	0.85	223
w/o $\{c'\}$	28.52	0.9286	0.1019	24.37	0.8623	0.1379	24.14	0.8036	0.1657	0.47	227
w/o Global	29.00	0.9295	0.1006	24.82	0.8761	0.1309	24.18	0.8045	0.1702	0.53	224
w/o Mask	28.85	0.9292	0.0981	24.71	0.8833	0.1216	23.91	0.8063	0.1590	0.53	205
w/o Depth	28.36	0.9261	0.1064	23.36	0.8573	0.1672	23.20	0.7878	0.1827	0.48	218

Table 1: Quantitative experimental results of existing methods (NeRF-W (Martin-Brualla et al., 2021), Ha-NeRF (Chen et al., 2022b), and CR-NeRF (Yang et al., 2023)) and Wild-GS on Phototourism dataset. Wild-GS† indicates that the model is trained by 15k iterations. The efficiencies of different methods are quantified by their training times (hours) and inference speeds (frames per second). Crop, Global, Mask, and Depth are abbreviations for triplane cropping, global appearance encoding, transient mask prediction, and depth regularization, respectively. $\{c'\}$ refers to $\{c'_{xy}, c'_{yz}, c'_{zx}\}$.

Experimental & Results

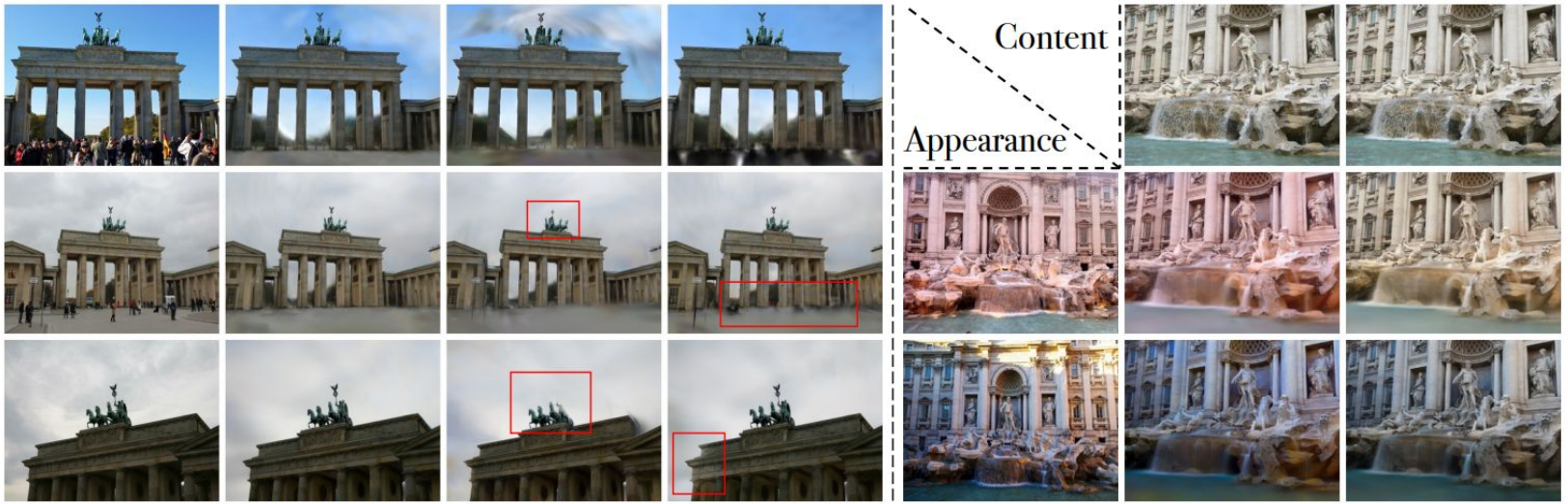


Figure 5: Rendering results of ablation study on Wild-GS when removing depth regularization, transient mask (left), and global appearance encoding (right). Red rectangles indicate the areas where geometry is missing or color inconsistency happens. Notations follow Table 1

Experimental & Results

Figure 6: Appearance and style transfer to novel views using reference images inside and outside the training dataset. Two arbitrary style images are borrowed from Ha-NeRF (Chen et al., 2022b).

PseudoLab

NeRF with Real world

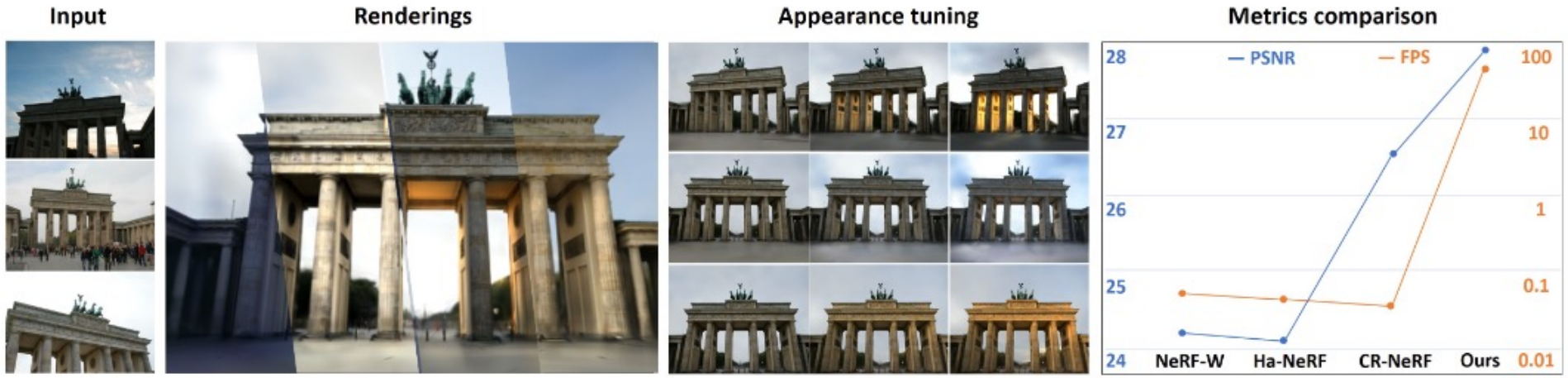
- Gaussian in the wild: 3D Gaussian Splatting for Unconstrained Image Collections -

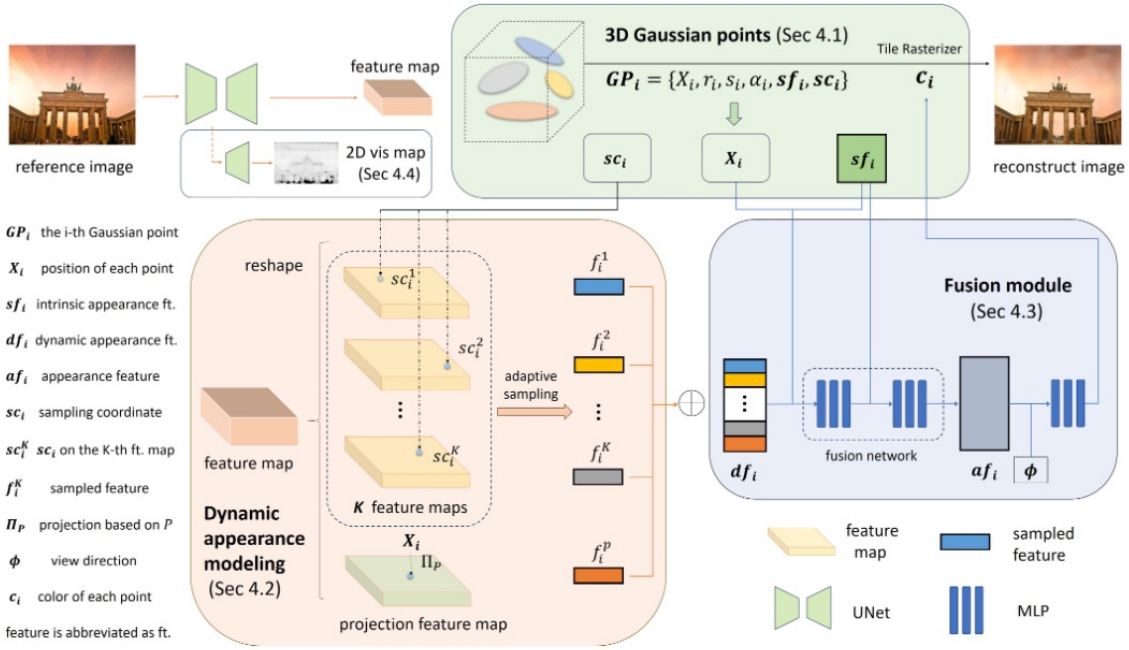
Wongi Park

https://eccv.ecva.net/virtual/2024/poster/2034

Introduction

reconstruct



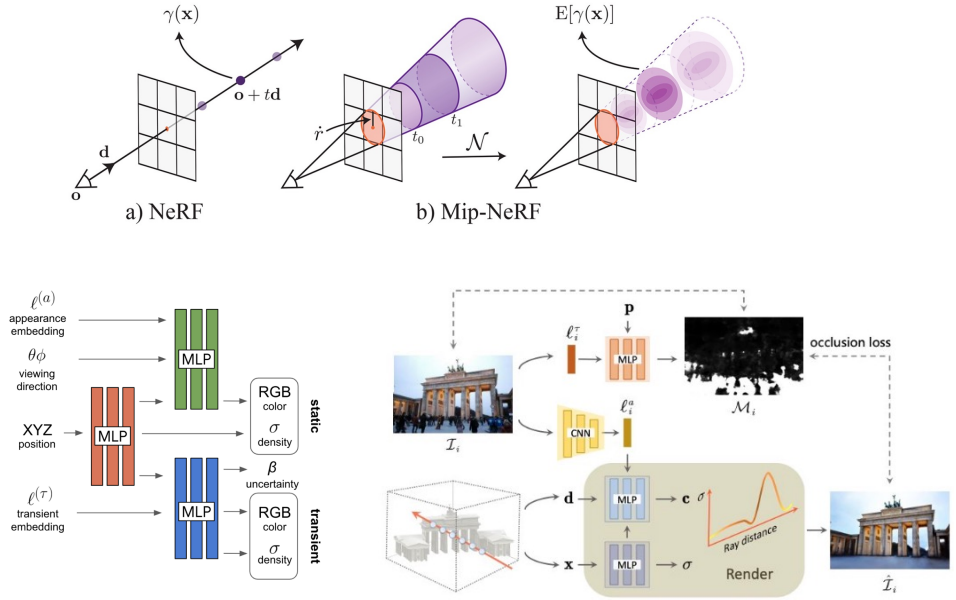


Key Contribution

1. Separate Intrinsic and dynamic factors
2. Propose Adaptive Sampling Strategy
3. SOTA, Fast FPS

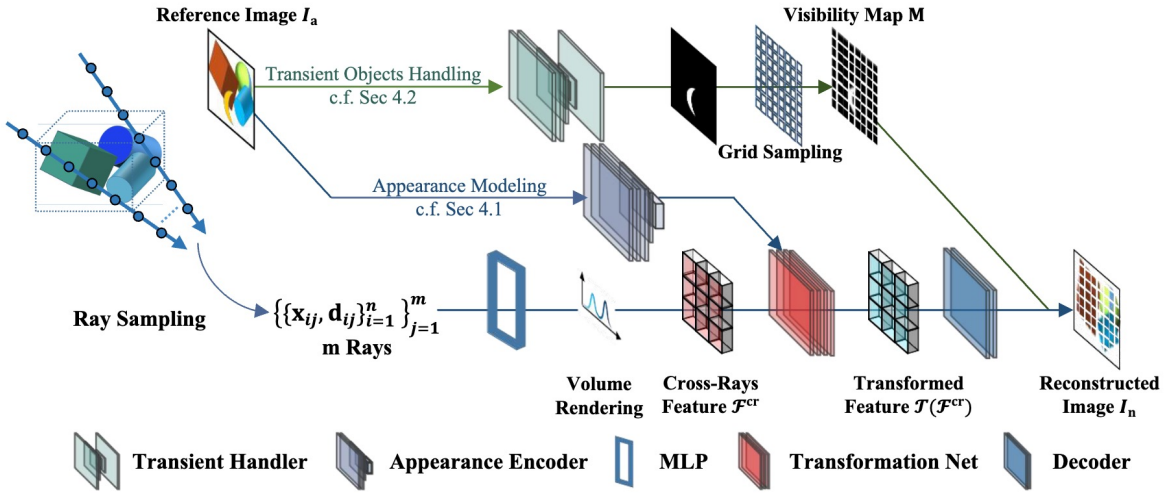
Related works

Novel view synthesis (based NeRF)



1. **3D conical frustum** for capturing high frequency details
2. **Use Appearance embedding** and Transient Embedding.
3. **Unclear** decompose transient factors.

CR-NeRF



1. **Lack** capturing high frequency details
2. **Blurring** intrinsic and dynamic appearance
3. **Highly** computation cost

Methods

Intrinsic and dynamic appearance

Ha-NeRF, NeRF-W, CR-NeRF

- Lacking dynamic environmental information in 3D. **(highlight and shadow)**
- Dynamic appearance features modeling

Solution: "Extract feature map"

1-1 $$f_i^P = BL(\Pi_P(X_i), F^P)$$

Projection Matrix

Feature Size

Bilinear interpolation

- Unreasonable under uneven lighting. **(identical along the same ray)**
- Only contains part of the scene **(Causing inconsistency)**

Solution: "Adaptive Sampling"

1-2 $$(f_i^1, f_i^2 ... f_i^K) = BL((sc_i^1, sc_i^2 ... sc_i^K), (F^1, F^2 ... F^K))$$

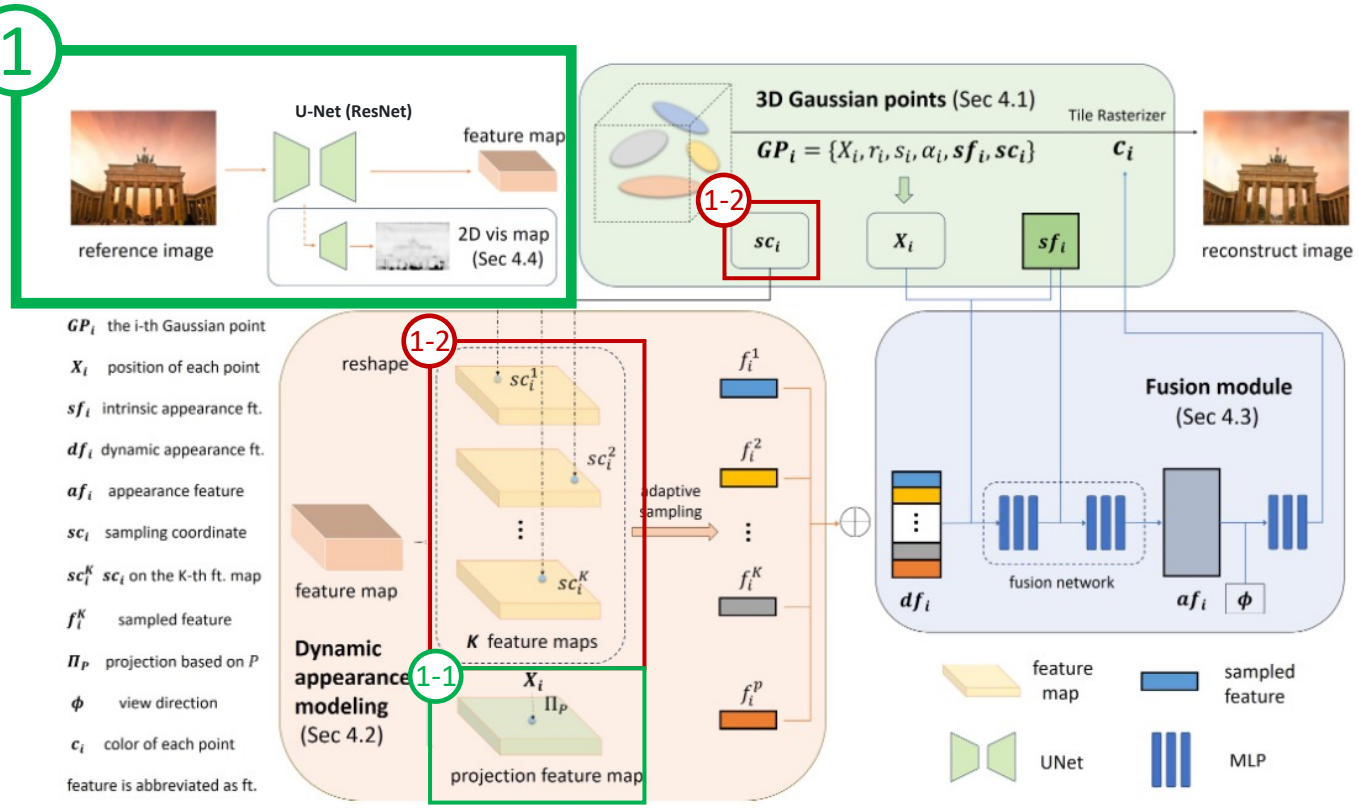
Coordination

Learnable features

+ Regularization Loss

$$L_{sc} = \frac{1}{N}\sum_i^N max\{0, |sc_i| - 1\}$$

- Deviating beyond the effective sampling range.



Methods

Dynamic appearance features modeling

Ha-NeRF, NeRF-W, CR-NeRF

- Small MLP couldn't capture high-frequency details

Solution 1: "Explicit separate intrinsic and dynamic"

Solution 2: "Fusion intrinsic and dynamic appearance features"

(2-1)
$$af_i = M_f(sf_i, df_i, X_i)$$
$$c_i = M_c(af_i, \phi)$$

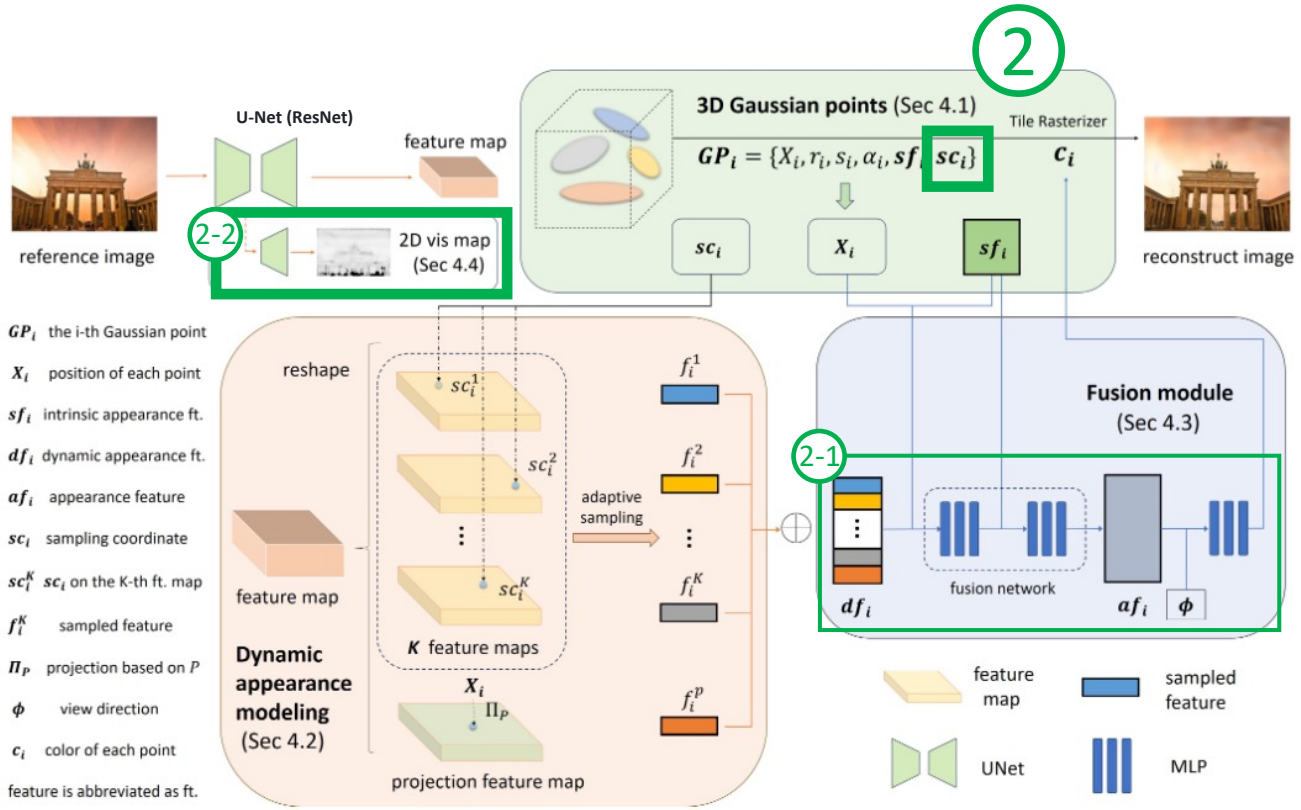
Transient Objects Handling

(2-2) $$L_{vm} = L_2(VM, 1)$$

To mitigate the impact of transient objects and prevent the occurrence of artifacts like floating points, we employ a 2D visibility map $VM \in \mathbb{R}^{1\times H\times W}$ obtained from a Unet model, facilitating accurate segmentation between transient and static objects. Leveraging the visibility map, we weight the loss calculation

Total Loss

$$L = L_c + \lambda_{sc}L_{sc} + \lambda_{vm}L_{vm}$$

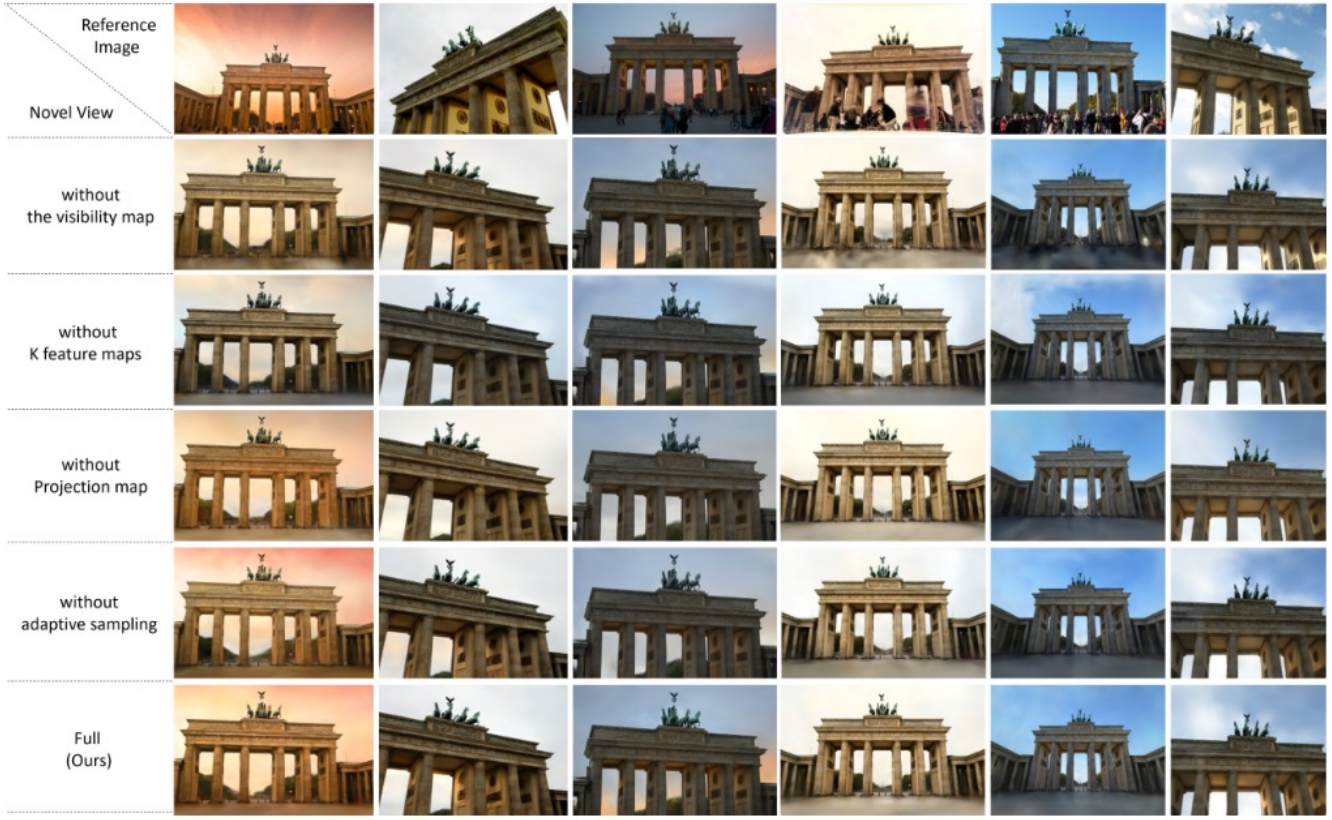


Implementation details and Experiments

We implement our method using Pytorch [30] and train our networks with Adam optimizer [17]. We set $K = 3$ in our experiment as the increasing K brings no performance improvement with meaningless computational cost. We train the full model on a single Nvidia RTX 3090 GPU for $70k$ steps and downsample all the images 2 times during training and evaluation, which takes approximately 2 hours. We also perform the adaptive control of the 3D Gaussians and follow other hyperparameter settings similar to 3DGS [16].

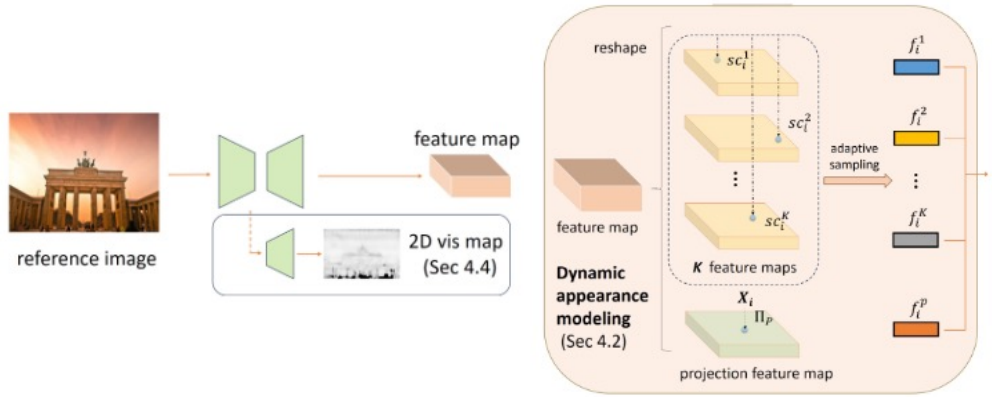
	Brandenburg Gate	Sacre Coeur	Trevi Fountain
3DGS	221	268	198
NeRF-W	0.0518	0.0514	0.0485
Ha-NeRF	0.0489	0.0497	0.0498
CR-NeRF	0.0445	0.0447	0.0446
Ours	55.8	58.3	38
Ours-cache	221	301	197

Implementation details and Experiments

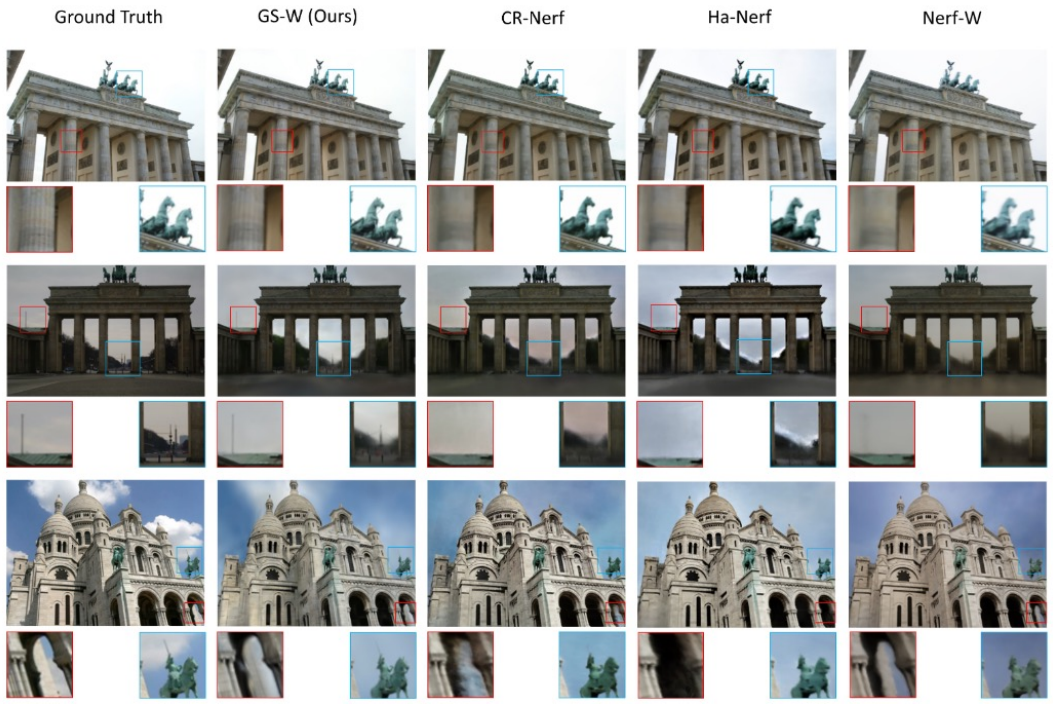


	Brandenburg Gate			Sacre Coeur			Trevi Fountain		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
w/o visibility map	**28.64**	**0.9324**	0.0867	**23.76**	**0.8723**	**0.1234**	23.03	0.8025	**0.1532**
w/o K feature maps	26.37	0.9210	0.1018	21.59	0.8521	0.1417	22.27	0.7859	0.1700
w/o Projection map	26.61	0.9285	0.0955	22.99	0.8635	0.1320	22.33	0.7980	0.1580
w/o adaptive sampling	26.44	0.9219	0.0961	21.45	0.8479	0.1439	22.18	0.7866	0.1674
w/o separation	28.22	0.9291	0.0942	22.94	0.8520	0.1469	**23.27**	0.7907	0.1714
full (Ours)	27.96	0.9319	**0.0862**	23.24	0.8632	0.1300	22.91	**0.8014**	0.1563

Implementation details and Experiments



Appearance Embedding guided



Appearance Embedding guided

PseudoLab

NeRF with Real world

- WildGaussians: 3D Gaussian Splatting in the Wild -

Wongi Park

https://arxiv.org/pdf/2407.08447

Introduction

Photo Tourism [35]

NeRF *On-the-go* [31]

Nerf Baselines

GitHub | Paper | Docs

NerfBaselines is a framework for **evaluating and comparing existing NeRF methods**. Currently, most official implementations use different dataset loaders, evaluation protocols, and metrics which renders the comparison of methods difficult. Therefore, this project aims to provide a **unified interface** for running and evaluating methods on different datasets in a consistent way using the same metrics. But instead of reimplementing the methods, **we use the official implementations** and wrap them so that they can be run easily using the same interface.

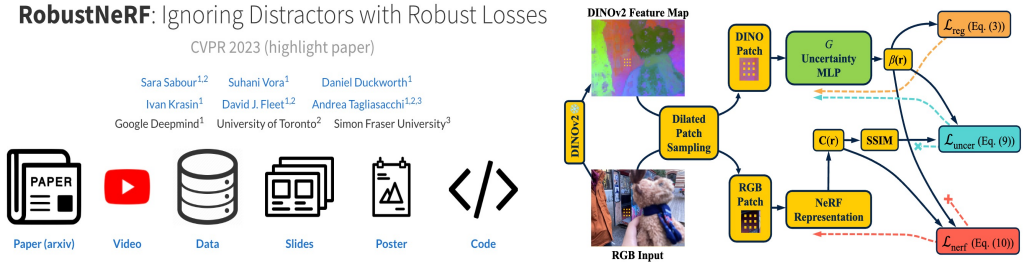
Contributions

- Appearance Modeling: tone-mapping MLP
- Uncertainty Optimization: leveraging DINO
- Append on NeRFbaselines

https://nerfbaselines.github.io/

Related works

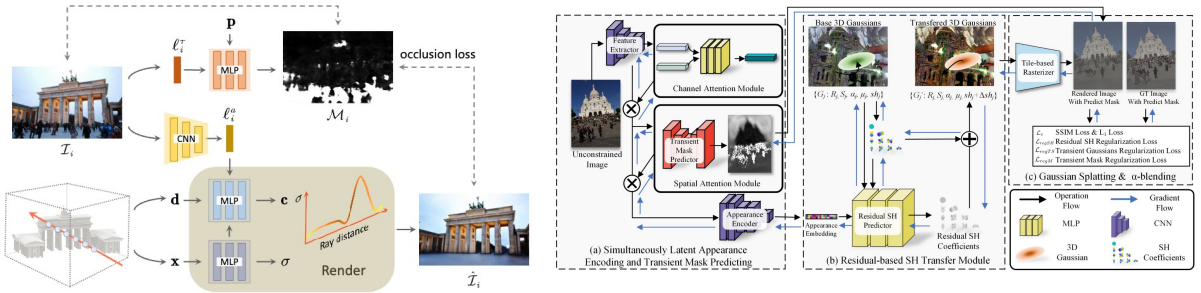
Novel View Synthesis in Dynamic Scenes.



Methods such as RobustNeRF[32] utilize Iteratively Reweighted Least Squares for outlier verification in small, controlled settings, while NeRF *On-the-go* [31] employs DINO v2 features [27] to predict uncertainties, allowing it to handle complex scenes with varying occlusion levels, albeit with long training times. Unlike these approaches, our method optimizes significantly faster. Moreover, we effectively handle dynamic scenarios even with changes in illumination.

- Depends on photometric loss
- Inspire by NeRF-on-the-go
- Slow rendering speed

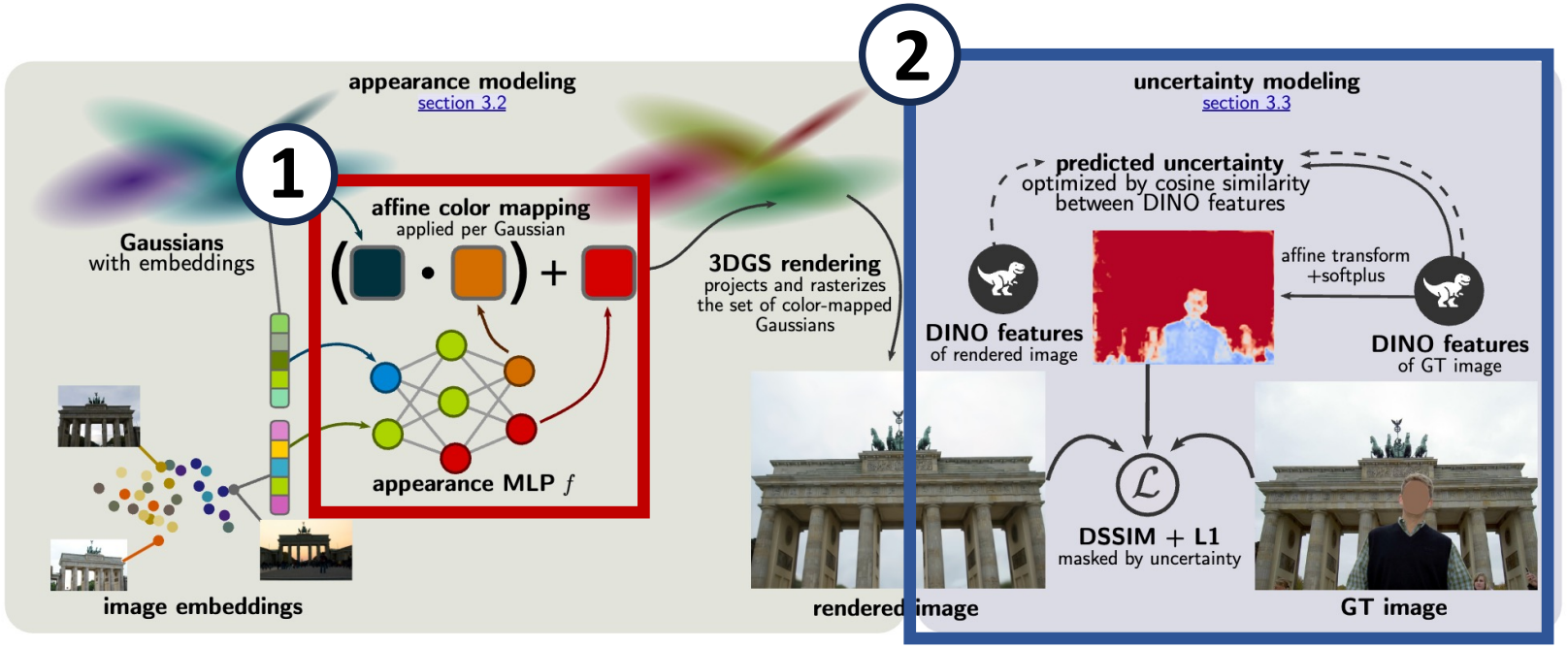
Novel View Synthesis for Unstructured Photo Collections.



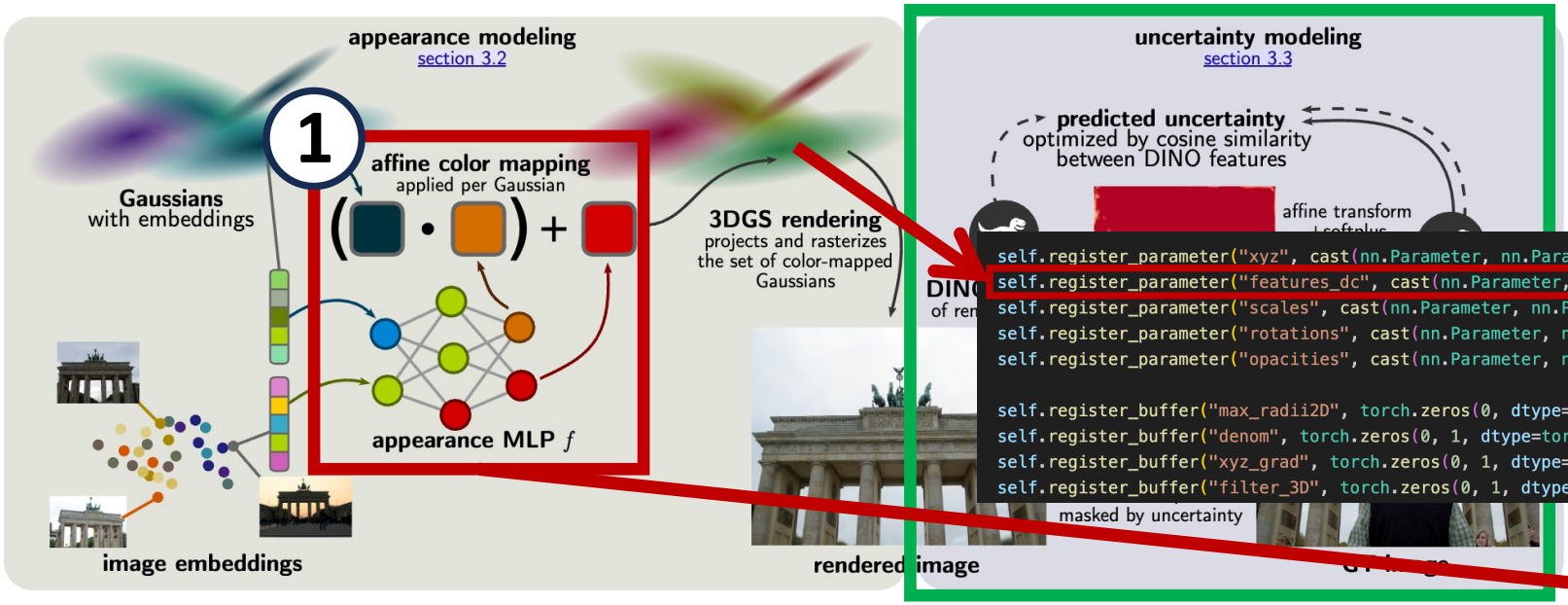
reference image. In contrast, our method employs a simpler and more scalable strategy by embedding appearance vectors directly within each Gaussian. This design not only simplifies the architecture but also enables us to 'bake' the trained representation back into 3DGS after appearances are fixed, enhancing both efficiency and adaptability. Finally, a concurrent work, Splatfacto-W [45], uses a similar appearance MLP to combine Gaussian and image embeddings to output spherical harmonics.

- Appearance embedding
- Couldn't consider heavy occlusion

Methods


appearance modeling
section 3.2
1
Gaussians
with embeddings
affine color mapping
applied per Gaussian
appearance MLP f
image embeddings
3DGS rendering
projects and rasterizes
the set of color-mapped
Gaussians
rendered image
2
uncertainty modeling
section 3.3
predicted uncertainty
optimized by cosine similarity
between DINO features
DINO features
of rendered image
affine transform
+softplus
DINO features
of GT image
$\mathcal{L}$
DSSIM + L1
masked by uncertainty
GT image

Methods




```
self.register_parameter("xyz", cast(nn.Parameter, nn.Parameter(torch.empty(0, 3, dtype=torch.float32, requires_grad=True))))
self.register_parameter("features_dc", cast(nn.Parameter, nn.Parameter(torch.empty(0, 3, dtype=torch.float32, requires_grad=True))))
self.register_parameter("scales", cast(nn.Parameter, nn.Parameter(torch.empty(0, 3, dtype=torch.float32, requires_grad=True))))
self.register_parameter("rotations", cast(nn.Parameter, nn.Parameter(torch.empty(0, 4, dtype=torch.float32, requires_grad=True))))
self.register_parameter("opacities", cast(nn.Parameter, nn.Parameter(torch.empty(0, 1, dtype=torch.float32, requires_grad=True))))

self.register_buffer("max_radii2D", torch.zeros(0, dtype=torch.float32))
self.register_buffer("denom", torch.zeros(0, 1, dtype=torch.float32))
self.register_buffer("xyz_grad", torch.zeros(0, 1, dtype=torch.float32))
self.register_buffer("filter_3D", torch.zeros(0, 1, dtype=torch.float32))
```

Following the literature on NeRFs [24, 30, 1], we use trainable per-image embeddings $\{\mathbf{e}_j\}_{j=1}^{N}$, where N is the number of training images, to handle images with varying appearances and illuminations, such as those shown in Fig. 1. Additionally, to enable varying colors of Gaussians under different appearances, we include a trainable embedding $\mathbf{g}_i$ for each Gaussian i. We input the per-image embedding $\mathbf{e}_j$, per-Gaussian embedding $\mathbf{g}_i$, and the base color $\bar{c}_i$ (0-th order SH) into an MLP f:

$$(\beta, \gamma) = f(\mathbf{e}_j, \mathbf{g}_i, \bar{c}_i) \ . \tag{4}$$

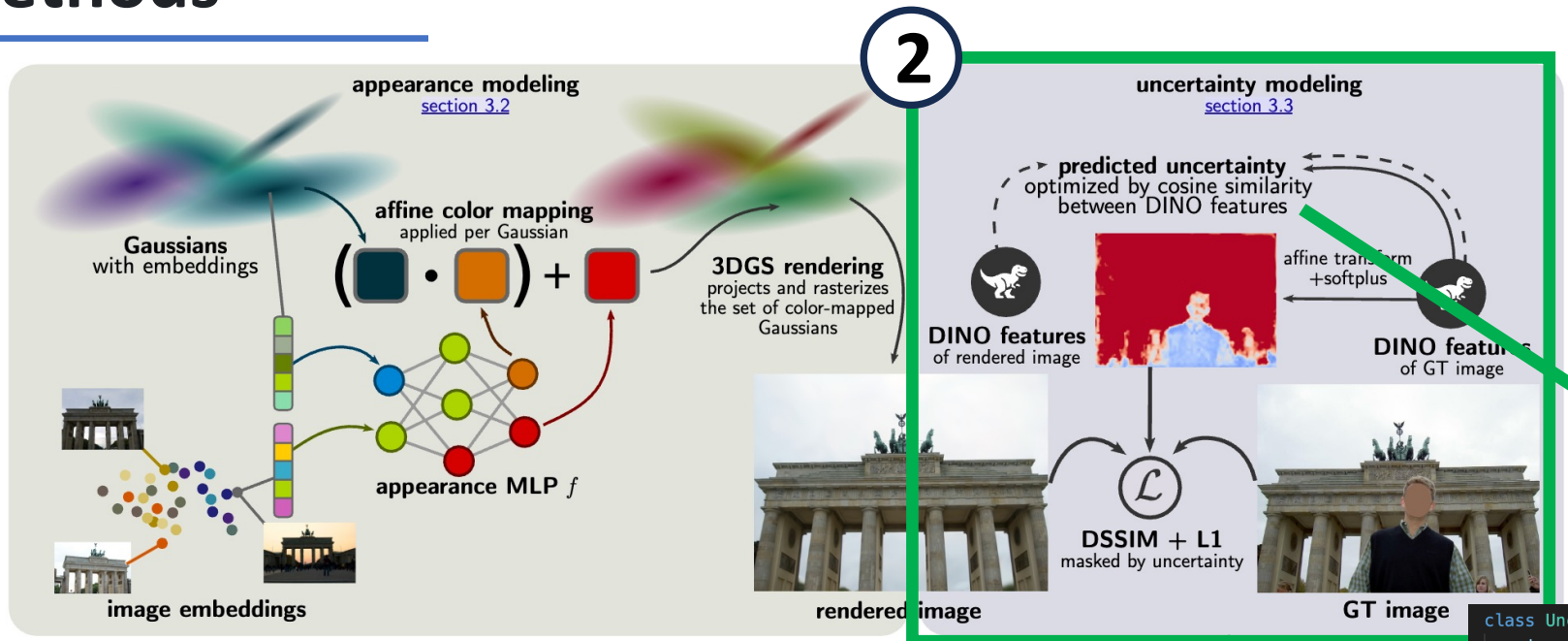
The output are the parameters of an affine transformation, where $(\beta, \gamma) = \{(\beta_k, \gamma_k)\}_{k=1}^{3}$ for each color channel k. Let $\hat{c}_i(\mathbf{r})$ be the i-th Gaussian's view-dependent color conditioned on the ray direction $\mathbf{r}$. The toned color of the Gaussian $\tilde{c}_i$ is given as:

$$\tilde{c}_i = \gamma \cdot \hat{c}_i(\mathbf{r}) + \beta \ . \tag{5}$$

```
class EmbeddingModel(nn.Module):
    def __init__(self, config: Config):
        super().__init__()
        self.config = config
        # sh_coeffs = 4**2
        feat_in = 3
        if config.appearance_model_sh:
            feat_in = ((config.sh_degree + 1) ** 2) * 3
        self.mlp = nn.Sequential(
            nn.Linear(config.appearance_embedding_dim + feat_in + 6 * self.config.appearance_n_fourier_freqs, 128),
            nn.ReLU(),
            nn.Linear(128, 128),
            nn.ReLU(),
            nn.Linear(128, feat_in*2),
        )

    def forward(self, gembedding, aembedding, color, viewdir=None):
        del viewdir  # Viewdirs interface is kept to be compatible with prev. version

        input_color = color
        if not self.config.appearance_model_sh:
            color = color[..., :3]
        inp = torch.cat((color, gembedding, aembedding), dim=-1)
        offset, mul = torch.split(self.mlp(inp) * 0.01, [color.shape[-1], color.shape[-1]], dim=-1)
        offset = torch.cat((offset / C0, torch.zeros_like(input_color[..., offset.shape[-1]:])), dim=-1)
        mul = mul.repeat(1, input_color.shape[-1] // mul.shape[-1])
        return input_color * mul + offset
```

Intrinsic color

Methods

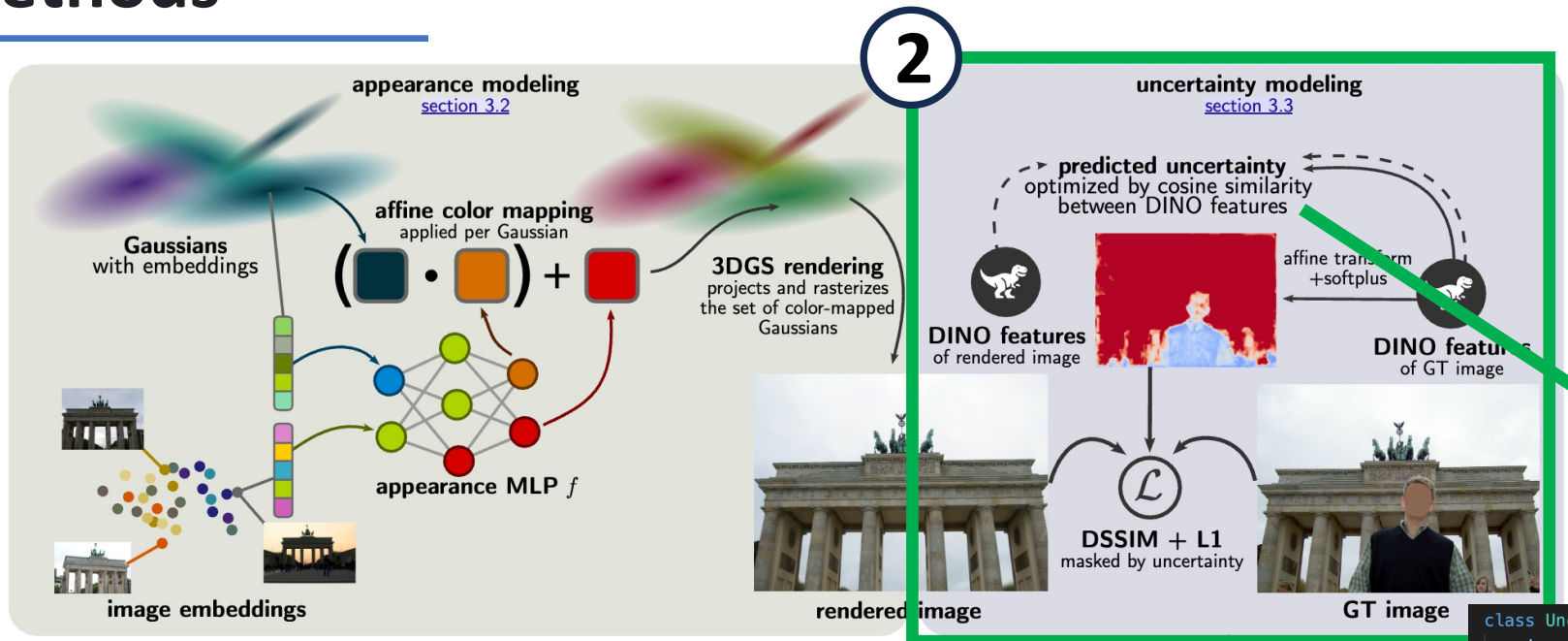



```python
def get_loss(self, gt_image, image, iteration, prefix='', _cache_entry=None):
    gt_torch = gt_image.unsqueeze(0)
    image = image.unsqueeze(0)
    loss, metrics, loss_mult = self._compute_losses(gt_torch, image, iteration, prefix, _cache_entry=_cache_entry)
    loss_mult = loss_mult.squeeze(0)
    metrics[f"{prefix}uncertainty_loss"] = metrics.pop(f"{prefix}loss")
    metrics.pop(f"{prefix}ssim")
    return loss, metrics, loss_mult
```

```python
def _compute_losses(self, gt, prediction, prefix='', _cache_entry=None):
    uncertainty = self(self._scale_input(gt, self.config.uncertainty_dino_max_size), _cache_entry=_cache_entry)
    log_uncertainty = torch.log(uncertainty)
    _dssim_go = dssim_go(gt, prediction, size_average=False).unsqueeze(1).clamp_max(self.config.uncertainty_dssim_clip_max)
    _dssim_go = 1 - ssim(gt, prediction).unsqueeze(1)
    _ssim = ssim_down(gt, prediction, max_size=400).unsqueeze(1)
    _msssim = msssim(gt, prediction, max_size=400, min_size=80).unsqueeze(1)
```

```python
class UncertaintyModel(nn.Module):
    img_norm_mean: Tensor
    img_norm_std: Tensor

    def __init__(self, config: Config):
        super().__init__()
        self.config = config

        self.backbone = getattr(dinov2, config.uncertainty_backbone)(pretrained=True
        self.patch_size = self.backbone.patch_size
        in_features = self.backbone.embed_dim
        self.conv_seg = nn.Conv2d(in_features, 1, kernel_size=1)
        self.bn = nn.SyncBatchNorm(in_features)
        nn.init.normal_(self.conv_seg.weight.data, 0, 0.01)
        nn.init.zeros_(assert_not_none(self.conv_seg.bias).data)

        img_norm_mean = torch.tensor([123.675, 116.28, 103.53], dtype=torch.float32)
        img_norm_std = torch.tensor([58.395, 57.12, 57.375], dtype=torch.float32)
        self.register_buffer("img_norm_mean", img_norm_mean / 255.)
        self.register_buffer("img_norm_std", img_norm_std / 255.)

        self._images_cache = {}

        # Freeze dinov2 backbone
        for p in self.backbone.parameters():
            p.requires_grad = False
```

Methods

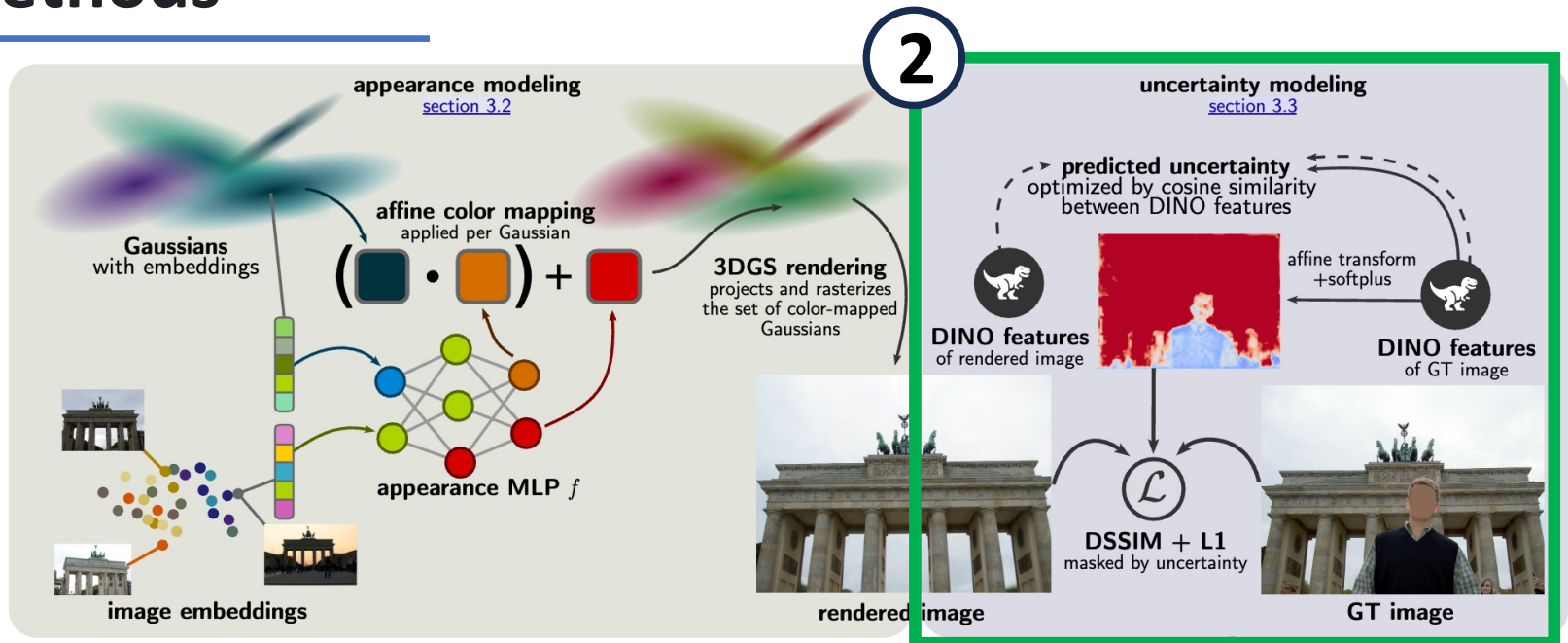



```python
def get_loss(self, gt_image, image, iteration, prefix='', _cache_entry=None):
    gt_torch = gt_image.unsqueeze(0)
    image = image.unsqueeze(0)
    loss, metrics, loss_mult = self._compute_losses(gt_torch, image, iteration, prefix, _cache_entry=_cache_entry)
    loss_mult = loss_mult.squeeze(0)
    metrics[f"{prefix}uncertainty_loss"] = metrics.pop(f"{prefix}loss")
    metrics.pop(f"{prefix}ssim")
    return loss, metrics, loss_mult
```

```python
def _compute_losses(self, gt, prediction, prefix='', _cache_entry=None):
    uncertainty = self(self._scale_input(gt, self.config.uncertainty_dino_max_size), _cache_entry=_cache_entry)
    log_uncertainty = torch.log(uncertainty)
    _dssim_go = dssim_go(gt, prediction, size_average=False).unsqueeze(1).clamp_max(self.config.uncertainty_dssim_clip_max)
    _dssim_go = 1 - ssim(gt, prediction).unsqueeze(1)
    _ssim = ssim_down(gt, prediction, max_size=400).unsqueeze(1)
    _msssim = msssim(gt, prediction, max_size=400, min_size=80).unsqueeze(1)
```

```python
class UncertaintyModel(nn.Module):
    img_norm_mean: Tensor
    img_norm_std: Tensor

    def __init__(self, config: Config):
        super().__init__()
        self.config = config

        self.backbone = getattr(dinov2, config.uncertainty_backbone)(pretrained=True
        self.patch_size = self.backbone.patch_size
        in_features = self.backbone.embed_dim
        self.conv_seg = nn.Conv2d(in_features, 1, kernel_size=1)
        self.bn = nn.SyncBatchNorm(in_features)
        nn.init.normal_(self.conv_seg.weight.data, 0, 0.01)
        nn.init.zeros_(assert_not_none(self.conv_seg.bias).data)

        img_norm_mean = torch.tensor([123.675, 116.28, 103.53], dtype=torch.float32)
        img_norm_std = torch.tensor([58.395, 57.12, 57.375], dtype=torch.float32)
        self.register_buffer("img_norm_mean", img_norm_mean / 255.)
        self.register_buffer("img_norm_std", img_norm_std / 255.)

        self._images_cache = {}

        # Freeze dinov2 backbone
        for p in self.backbone.parameters():
            p.requires_grad = False
```

Methods




```python
def _compute_cosine_similarity(self, x, y, _x_cache=None, _y_cache=None, max_size=None):
    # Normalize data
    h, w = x.shape[2:]
    if max_size is not None and (max_size < h or max_size < w):
        assert max_size % 14 == 0, "max_size must be divisible by 14"
        scale_factor = min(max_size/x.shape[-2], max_size/x.shape[-1])
        nh = int(h * scale_factor)
        nw = int(w * scale_factor)
        nh = ((nh + 13) // 14) * 14
        nw = ((nw + 13) // 14) * 14
        x = F.interpolate(x, size=(nh, nw), mode='bilinear')
        y = F.interpolate(y, size=(nh, nw), mode='bilinear')

    x = (x - self.img_norm_mean[None, :, None, None]) / self.img_norm_std[None, :, None, None]
    y = (y - self.img_norm_mean[None, :, None, None]) / self.img_norm_std[None, :, None, None]
    pads = list(itertools.chain.from_iterable(self._get_pad(m) for m in x.shape[:1:-1]))
    x = F.pad(x, pads)
    padded_shape = x.shape
    y = F.pad(y, pads)
```

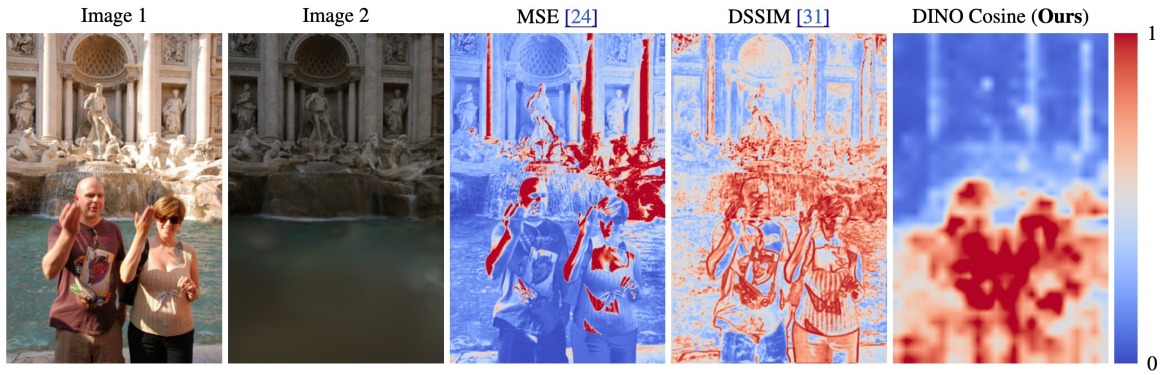
$$\mathcal{L}_u = -\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{\|\tilde{C}-C\|_2^2}{2\sigma^2}\right)\right) = \frac{\|\tilde{C}-C\|_2^2}{2\sigma^2} + \log\sigma + \frac{\log 2\pi}{2}$$

$$\mathcal{L}_{\text{dino}}(\tilde{D}, D) = \min\left(1, 2 - \frac{2\tilde{D}\cdot D}{\|\tilde{D}\|_2\|D\|_2}\right), \quad \mathcal{L}_{\text{uncertainty}} = \frac{\mathcal{L}_{\text{dino}}(\tilde{D}, D)}{2\sigma^2} + \lambda_{\text{prior}}\log\sigma,$$

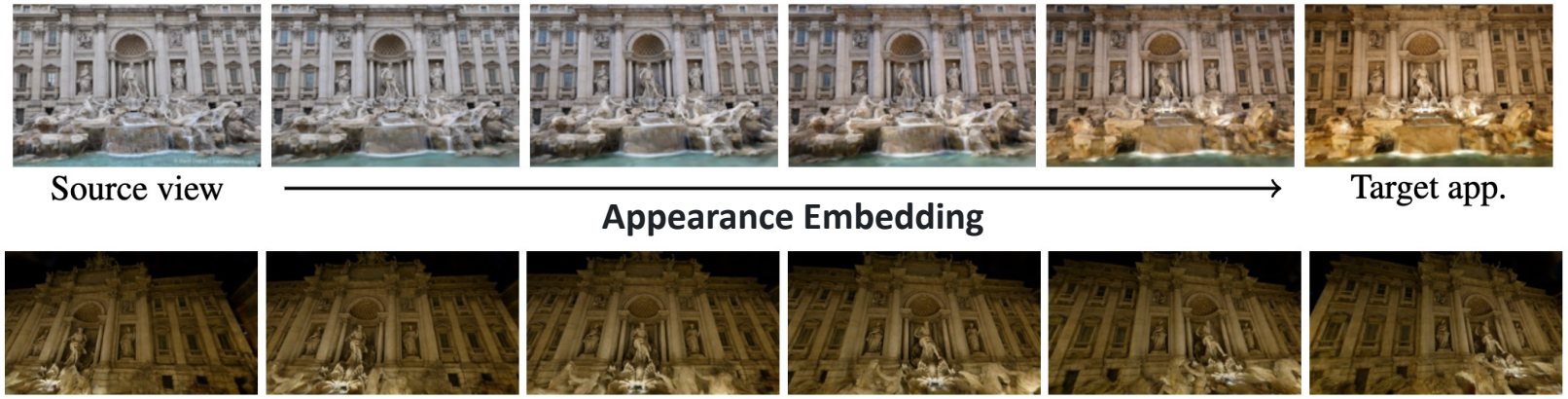
$$M = \mathbb{1}\left(\frac{1}{2\sigma^2} > 1\right) \quad \mathcal{L}_{\text{color-masked}} = \lambda_{\text{dssim}} M\,\text{DSSIM}(\hat{C}, C) + (1-\lambda_{\text{dssim}})M\|\tilde{C}-C\|_1$$

```python
loss = (
    (1.0 - self.config.lambda_dssim) * (Ll1 * loss_mult).mean() +
    self.config.lambda_dssim * ((1.0 - ssim_value) * loss_mult).mean() +
    uncertainty_loss
```

Ablation Study



- DINOv2 is better than Photometric loss and SSIM!



Fixed appearance and move camera viewpoints

- T-SNE for Appearance embedding

Ablation Study

	Brandenburg Gate			Sacre Coeur			Trevi Fountain		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
WildGaussians (Ours)	27.77	0.927	0.133	22.56	0.859	0.177	23.63	0.766	0.228
w/o Fourier feat.	27.33	0.924	0.141	22.32	0.853	0.188	23.78	0.766	0.229
w/o appearance modeling	20.81	0.874	0.187	17.19	0.811	0.237	17.42	0.697	0.299
w/o appearance&uncertainty	20.04	0.875	0.187	17.59	0.819	0.228	17.81	0.701	0.287
VastGaussian [22]	25.65	0.910	0.159	20.52	0.820	0.225	20.40	0.716	0.283
w/o Gaussian embeddings [30]	25.18	0.908	0.156	18.68	0.790	0.250	19.49	0.723	0.275
w/o uncertainty modeling	27.16	0.920	0.143	21.97	0.851	0.191	23.82	0.765	0.228
MSSIM uncert.	27.32	0.911	0.167	21.19	0.817	0.237	21.20	0.717	0.295
w. explicit masks	*26.87*	*0.923*	*0.138*	*22.24*	*0.855*	*0.186*	*23.84*	*0.765*	*0.231*

Table 4: **Detailed Ablation Study** conducted on the Photo Tourism dataset [35]. The first, second, and third values are highlighted.

	GPU hrs./ FPS	Brandenburg Gate			Sacre Coeur			Trevi Fountain		
		PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF [25]	-/<1	18.90	0.815	0.231	15.60	0.715	0.291	16.14	0.600	0.366
NeRF-W-re [24]	164/<1	24.17	0.890	0.167	19.20	0.807	0.191	18.97	0.698	0.265
Ha-NeRF [4]	452/<1	24.04	0.877	0.139	20.02	0.801	0.171	20.18	0.690	0.222
K-Planes [9]	0.6/<1	25.49	0.879	0.224	20.61	0.774	0.265	22.67	0.714	0.317
RefinedFields [13]	150/<1	26.64	0.886	-	22.26	0.817	-	23.42	0.737	-
3DGS [14]	2.2/57	19.37	0.880	0.141	17.44	0.835	0.204	17.58	0.709	0.266
GS-W† [52]	1.2/51	23.51	0.897	0.166	19.39	0.825	0.211	20.06	0.723	0.274
SWAG* [5]	0.8/15	26.33	0.929	0.139	21.16	0.860	0.185	23.10	0.815	0.208
Ours	7.2/117	27.77	0.927	0.133	22.56	0.859	0.177	23.63	0.766	0.228

† Evaluated using NeRF-W evaluation protocol [24, 18]; * Source code not available, numbers from the paper, unknown GPU used.

Method	GPU hrs./ FPS	Low Occlusion						Medium Occlusion				High Occlusion					
		Mountain			Fountain			Corner			Patio			Spot			Patio-High		
		PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
NeRF *On-the-go* [31]	43/<1	20.46	**0.661**	0.186	20.79	0.661	0.195	23.74	0.806	0.127	20.88	0.754	**0.133**	22.80	0.800	**0.132**	21.57	0.706	**0.205**
3DGS [14]	0.35/116	19.40	0.638	0.213	19.96	0.659	**0.185**	20.90	0.713	0.241	17.48	0.704	0.199	20.77	0.693	0.316	17.29	0.604	0.363
Mip-Splatting	*0.18/82**	19.86	0.649	0.200	20.19	0.672	0.189	21.15	0.728	0.230	18.31	0.639	0.328	20.18	0.689	0.338	18.31	0.639	0.328
Gaussian Opacity Fields	*0.41/43**	**20.70**	**0.661**	**0.169**	20.37	0.662	0.187	21.53	0.739	0.241	15.58	0.491	0.536	20.03	0.683	0.324	15.58	0.491	0.536
GS-W	*0.55/71**	19.43	0.596	0.299	20.06	**0.723**	0.274	22.17	0.793	0.155	19.90	0.681	0.260	17.13	0.608	0.409	19.90	0.681	0.260
Ours	0.50/108	20.43	0.653	0.255	**20.81**	0.662	0.215	**24.16**	**0.822**	**0.045**	**21.44**	**0.800**	0.138	**23.82**	**0.816**	0.138	**22.23**	**0.725**	0.206

* Methods were trained and evaluated on NVIDIA A100, while the rest used NVIDIA GTX 4090.

Table 5: **Extended Results on the NeRF *On-the-go* Dataset.**

Dataset (**ratio**)	MipNeRF360 [1] (**0%**)			Photo Tourism [35] (**3.5%**)			*On-the-go* [31] low. (**5%**)			*On-the-go* [31] high. (**26%**)		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
Ours	23.73	0.643	0.304	24.63	0.851	0.179	20.62	0.658	0.235	23.03	0.771	0.172
w/o uncert.	23.71	0.643	0.313	24.32	0.845	0.187	20.53	0.646	0.229	20.27	0.652	0.337
w/o app.	23.31	0.641	0.319	18.47	0.794	0.241	20.80	0.669	0.228	22.80	0.755	0.183

Table 3: We conduct **ablation studies** on the Photo Tourism [35], NeRF *On-the-go* [31], and MipNeRF360 (bicycle) [1] datasets with varying degree of occlusion. The first, second, and third values are highlighted.

Method	GPU hrs. /FPS	Low Occlusion			Medium Occlusion			High Occlusion			Average		
		PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
NeRF *On-the-go* [31]	43/<1	20.63	0.661	0.191	22.31	0.780	0.130	22.19	0.753	0.169	21.71	0.731	0.163
3DGS [14]	0.35/116	19.68	0.649	0.199	19.19	0.709	0.220	19.03	0.649	0.340	19.30	0.669	0.253
Gaussian Opacity Fields [51]	*0.41/43**	20.54	0.662	0.178	19.39	0.719	0.231	17.81	0.578	0.430	19.24	0.656	0.280
Mip-Splatting [50]	*0.18/82**	20.15	0.661	0.194	19.12	0.719	0.221	18.10	0.664	0.333	19.12	0.681	0.249
GS-W [52]	*0.55/71**	18.67	0.595	0.288	21.50	0.783	0.152	18.52	0.644	0.335	19.56	0.674	0.258
Ours	0.50/108	20.62	0.658	0.235	22.80	0.811	0.092	23.03	0.771	0.172	22.15	0.756	0.167

* Methods were trained and evaluated on NVIDIA A100, while the rest used NVIDIA GTX 4090.

Ablation Study