CMSC740
Advanced Computer Graphics

Matthias Zwicker
Fall 2025

Limitations of NeRF

- Doesn't attempt to represent surface directly (only volumetric density σ)
- Treats pixels as infinitesimal rays, doesn't take into account pixel areas
- Doesn't take into account imaging artifacts such as blur, over-/under-exposure
- Only works for static scenes
- Requires known camera parameters
- Training and rendering slow
- Requires many input images for high quality reconstruction
- Doesn't take into account potential appearance variation in input images (different illumination, time, time of year, etc.)
- Doesn't recover BRDF parameters and illumination
- Only uses 3D locations to predict scene (density, radiance); does not use correspondence information between 3D locations and images

Dealing with appearance changes

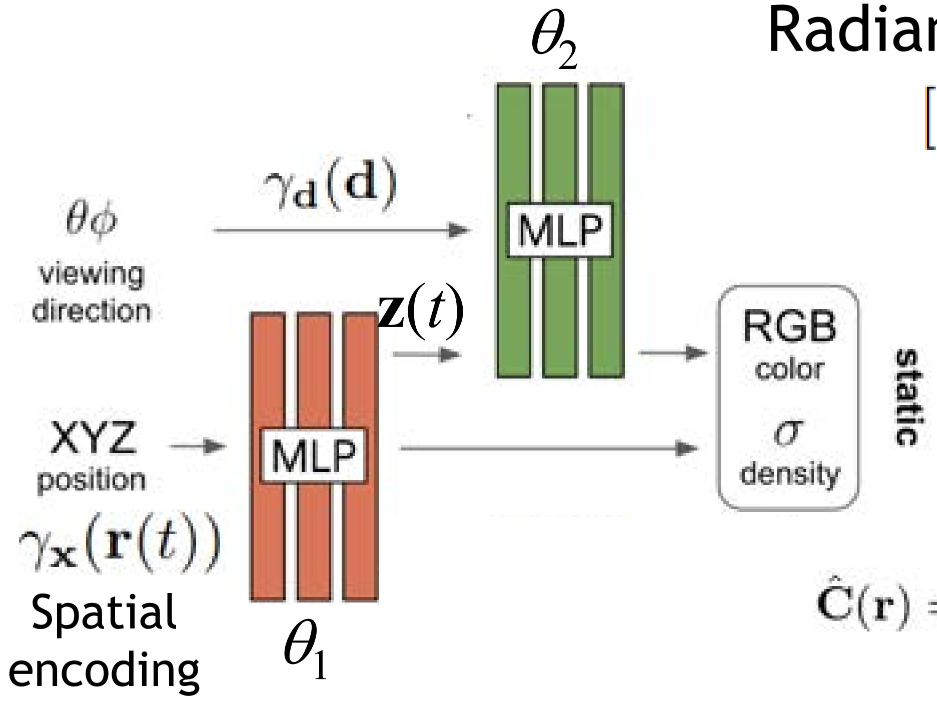
- Input images may be taken at different times
- Objects may move
- Illumination may change
- Static radiance field cannot model these effects

NeRF-W

- “NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections”, CVPR 2021, https://nerf-w.github.io/
- Goal: NeRF using images of outdoor scenes, acquired by different cameras, over extended period of time (e.g., tourist photos)
- NeRF-W addresses two challenges
  1. Different times of day, atmospheric conditions, imaging pipelines (exposure, white balance, tone-mapping)
  2. Transient objects

NeRF (recap of basic approach)

- Ray $\mathbf{r}$, ray parameter t, sample point $\mathbf{z}(t)$, spatial encoding $\gamma_{\mathbf{x}}$, density σ, radiance $\mathbf{c}$



Radiance, density at sample point

$$[\sigma(t), \mathbf{z}(t)] = \mathrm{MLP}_{\theta_1}(\gamma_{\mathbf{x}}(\mathbf{r}(t)))$$

$$\mathbf{c}(t) = \mathrm{MLP}_{\theta_2}(\mathbf{z}(t), \gamma_{\mathbf{d}}(\mathbf{d}))$$

Rendered pixel color

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

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

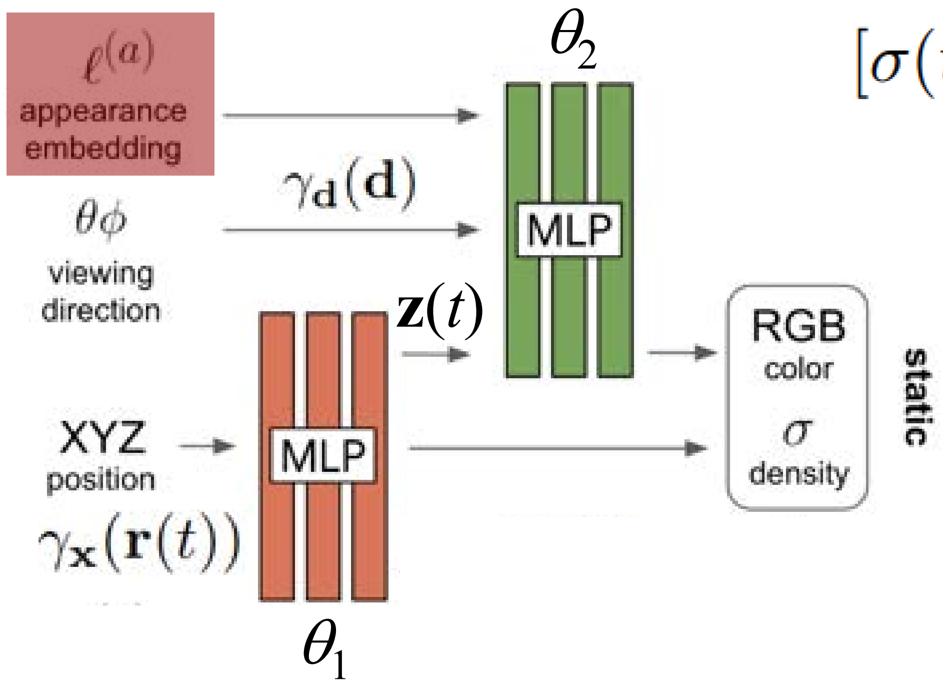
δ_k: distance between k-1 and k-th sample on ray $\alpha(x) = 1\text{-}\exp(-x)$

Latent appearance model

- Goal: allow image-dependent radiance along each ray based on per-image latent appearance vector
- Latent appearance vector
  - Represents "latent" influence of appearance (time of day, atmospheric conditions, parameters of imaging pipeline) of entire image on pixel values
  - Learned/optimized per image
  - Used as per-image input to radiance network

NeRF-W

- Latent appearance code $l_i^{(a)}$ per image i
- Image dependent radiance $\mathbf{c}_i$



$$[\sigma(t), \mathbf{z}(t)] = \mathrm{MLP}_{\theta_1}(\gamma_{\mathbf{x}}(\mathbf{r}(t)))$$

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

Transient objects

- Goals
  - Reconstruct images containing transient occluders (objects not present in all images)
  - Allow model to ignore unreliable (uncertain) pixels likely containing occluders
- Approach
  - Model transient objects separately using transient density and radiance
  - Estimate uncertainty to weigh loss

Transient radiance, density

- Optimize separate radiance fields $\mathbf{c}_i$, σ, and $\mathbf{c}_i^{(\tau)}$, $\sigma_i^{(\tau)}$ for static and transient (moving) scene parts

$$\hat{\mathbf{C}}_i(\mathbf{r}) = \sum_{k=1}^{K} T_i(t_k)\Big(\overbrace{\alpha(\sigma(t_k)\delta_k)\mathbf{c}_i(t_k)}^{\text{Static}} + \overbrace{\alpha\Big(\sigma_i^{(\tau)}(t_k)\delta_k\Big)\mathbf{c}_i^{(\tau)}(t_k)}^{\text{Transient density, radiance}}\Big)$$

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

Uncertainty

- Optimize uncertainty in addition to transient density, radiance using MLP

$$\Big[\overbrace{\sigma_i^{(\tau)}(t), \mathbf{c}_i^{(\tau)}(t)}^{\text{Transient density, radiance}}, \overbrace{\tilde{\beta}_i(t)}^{\text{Uncertainty}}\Big] = \mathrm{MLP}_{\theta_3}\Big(\mathbf{z}(t), \overbrace{\boldsymbol{\ell}_i^{(\tau)}}^{\text{Per image transient appearance}}\Big)$$

$$\beta_i(t) = \beta_{\min} + \log\Big(1 + \exp\Big(\tilde{\beta}_i(t)\Big)\Big),$$

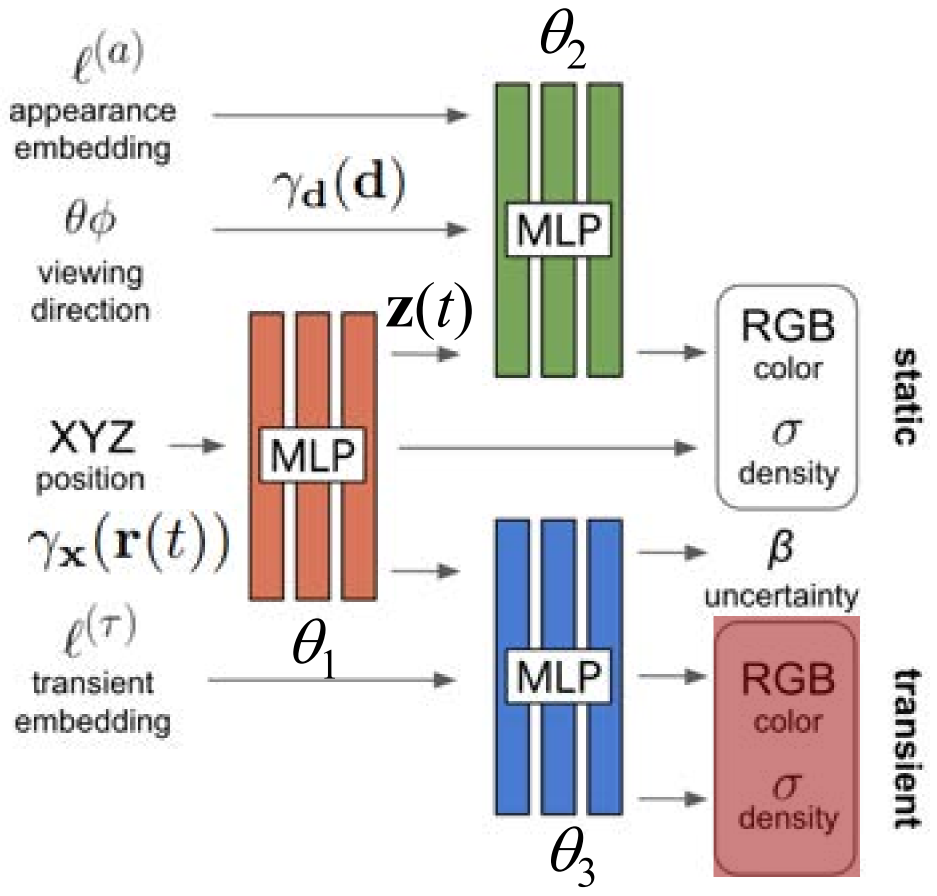
- Render uncertainty β_i to pixels similar as radiance, use to weight loss

NeRF-W architecture

- Full model

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

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



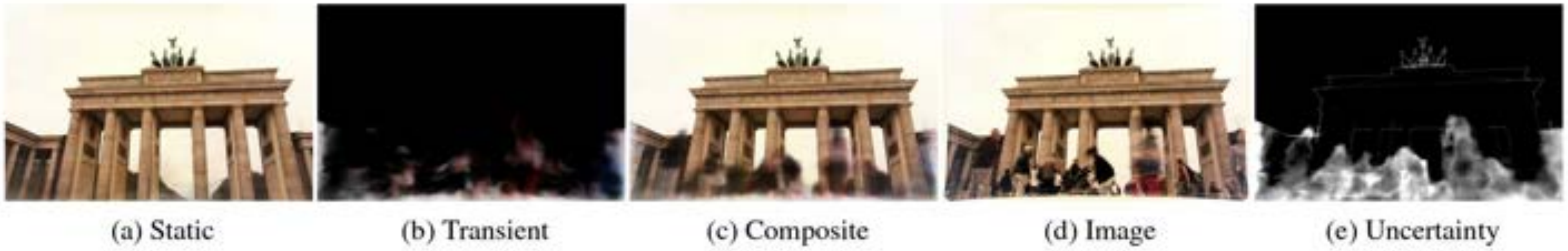
Per-pixel loss function $L_i(\mathbf{r})$

- First term: color difference weighted by uncertainty
- Second term: avoid trivial solution $\beta_i(\mathbf{r})$=infinity
- Third term: regularization, discourage large transient density

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

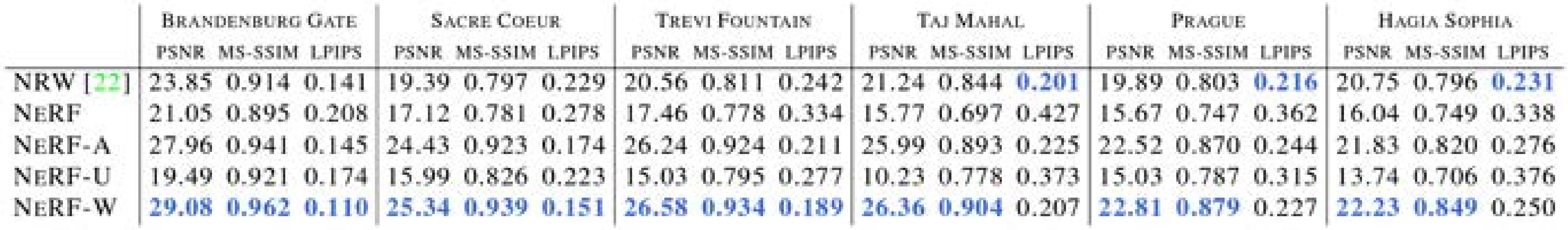
Visualization

- Can render static and transient components separately
- Loss minimizes difference between "composite" and ground truth image, weighted by rendered uncertainty


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

Results

- https://nerf-w.github.io/



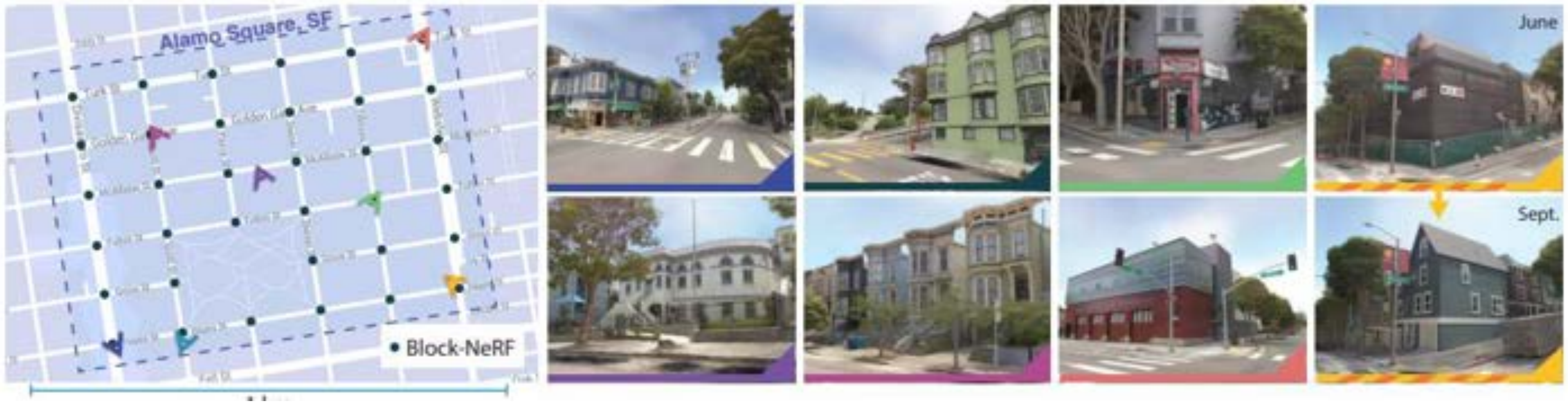
	Brandenburg Gate			Sacre Coeur			Trevi Fountain			Taj Mahal			Prague			Hagia Sophia		
	PSNR	MS-SSIM	LPIPS	PSNR	MS-SSIM	LPIPS	PSNR	MS-SSIM	LPIPS	PSNR	MS-SSIM	LPIPS	PSNR	MS-SSIM	LPIPS	PSNR	MS-SSIM	LPIPS
NRW [22]	23.85	0.914	0.141	19.39	0.797	0.229	20.56	0.811	0.242	21.24	0.844	**0.201**	19.89	0.803	**0.216**	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

Phototourism dataset

https://github.com/vcg-uvic/image-matching-benchmark

Extension to large scenes

- Challenge: training single NeRF to large scene is time/memory consuming
- Block-NeRF: train multiple NeRFs separately, with extensions to combine them for rendering
  https://waymo.com/research/block-nerf/
- Target application: city rendering



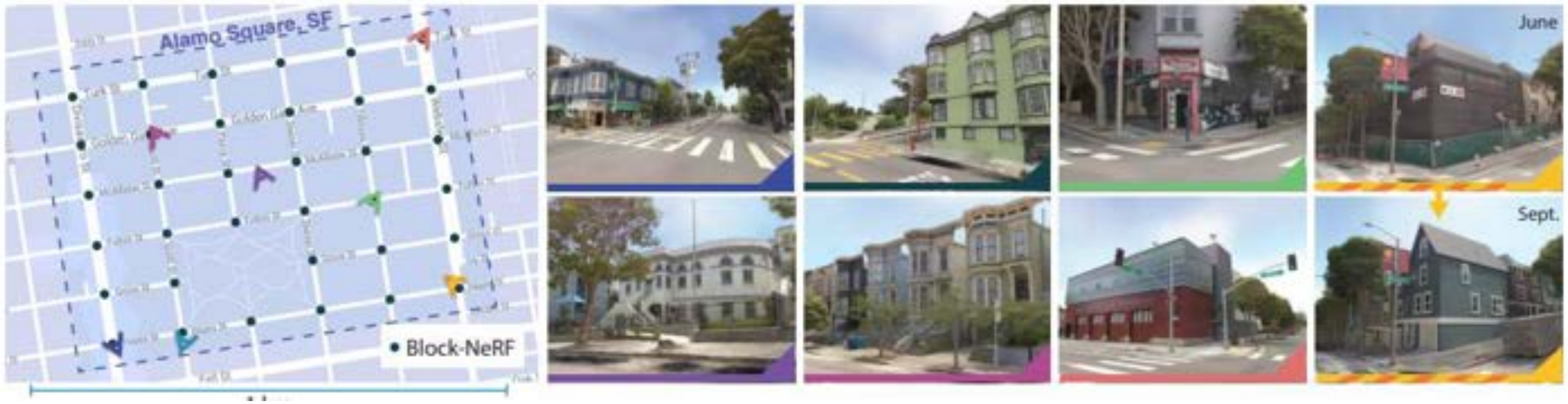
Large scale reconstruction with multiple NeRFs, input data collected "in the wild" over long periods of time

Block-NeRF

- Combines Mip-NeRF (extended positional encoding using beam geometry) and NeRF-W (latent appearance codes)
- Extensions
  - Placement of multiple blocks (separate NeRFs)
  - Visibility prediction
  - Rendering using multiple NeRFs

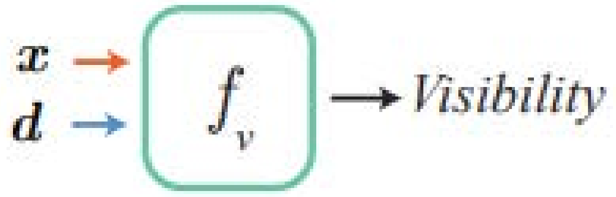
Block size, placement

- Target application: rendering cities
- Based on city blocks, place NeRF at city block intersections



Visibility network

- Visibility network trained to match transmittance given by main NeRF network
- Used to quickly predict visibility (faster than using NeRF densities to compute transmittances)



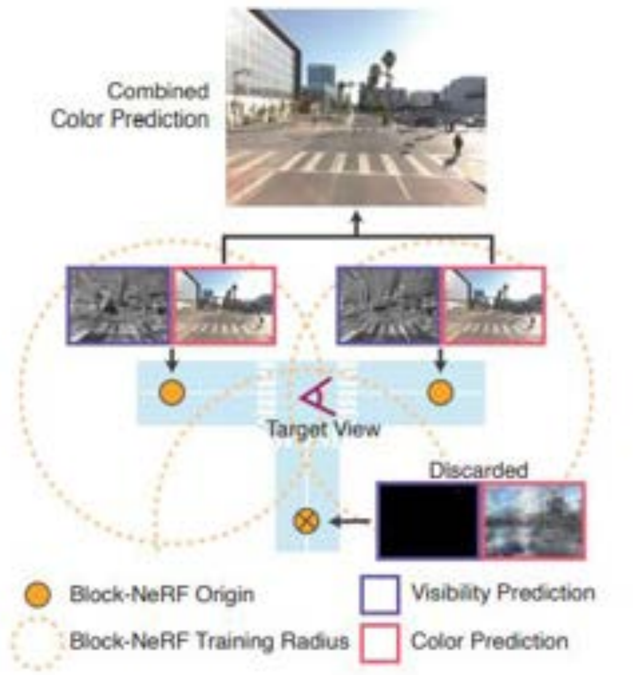
Additional details

- Refine camera poses during training
- Provide known image exposure times as input to NeRF
- Remove transient objects using semantic segmentation (people, cars, etc.)

Rendering target views

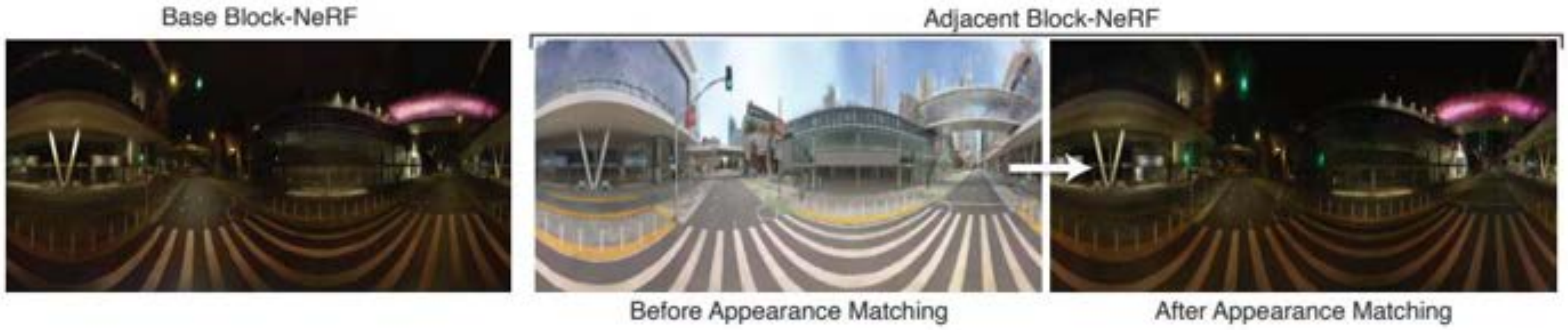
Steps

1. Appearance matching
2. Render target view from all blocks within given radius
3. Discard rendered views based on low visibility (average over pixels)
4. Blend views based on distance between virtual camera and block origin (large distance, low weight)

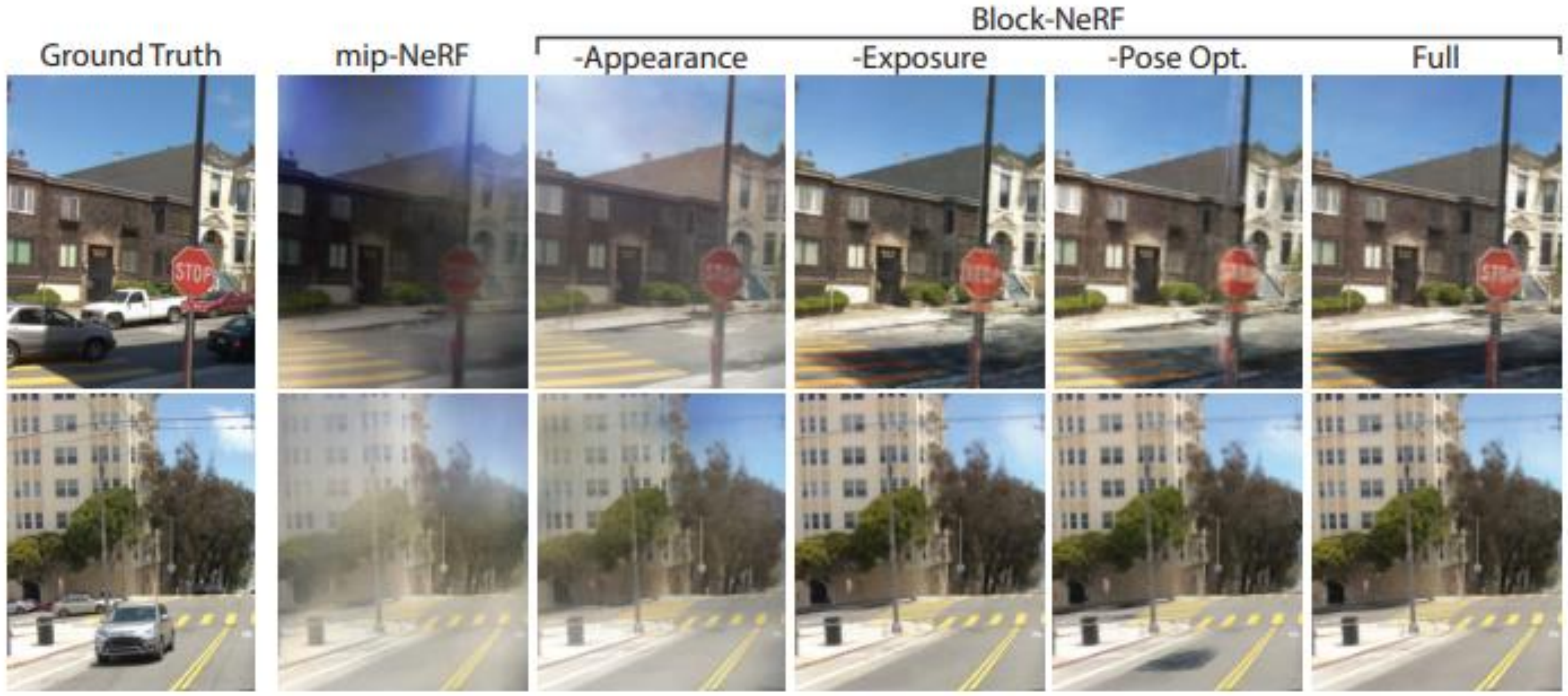


Appearance matching

- Goal: use latent appearance vectors to match appearance (time of day, white balance, weather, etc.) between NeRFs
- Naïve approach: use same appearance vector for all NeRFs
  - Doesn't work
- Greedy approach: fix appearance vector for one NeRF, then optimize appearances of "overlapping" NeRF
  - Select small number of high visibility 3D locations, optimize appearance vector to match colors at these points in both NeRFs



Ablation study

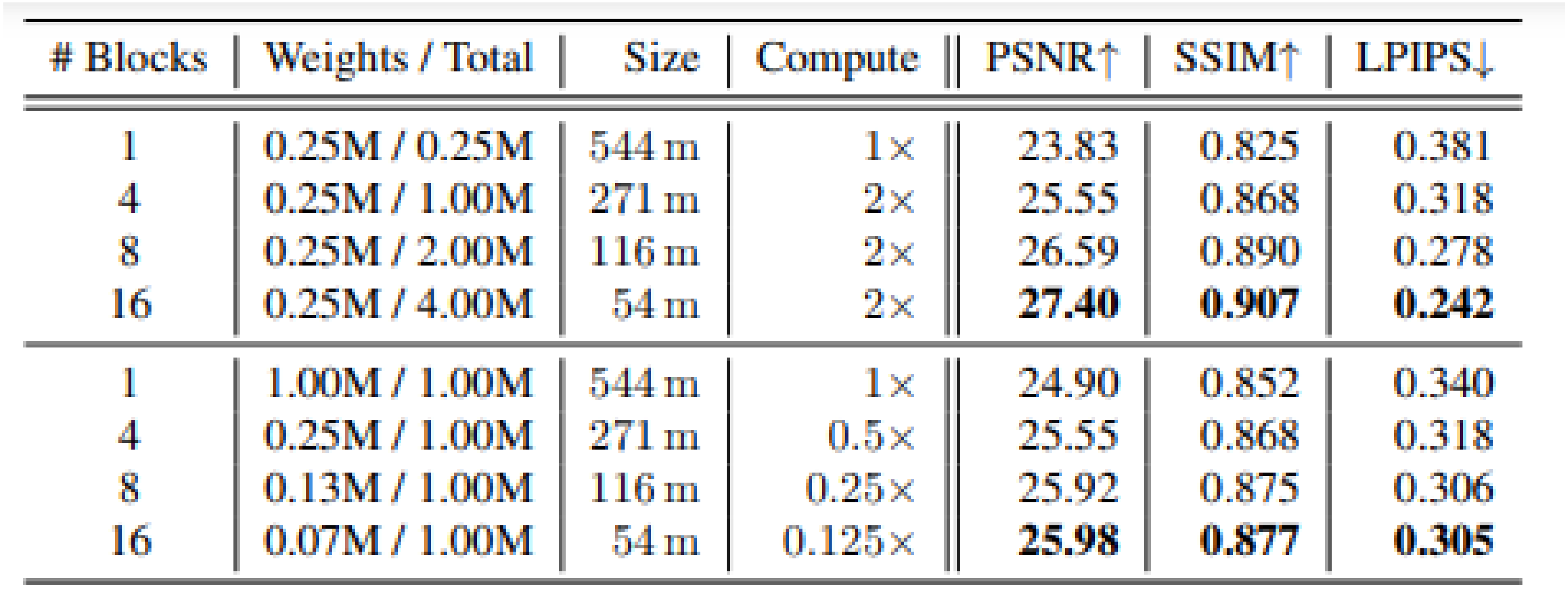


	NeRFs	PSNR↑	SSIM↑	LPIPS↓
	mip-NeRF	17.86	0.563	0.509
Ours	-Appearance	20.13	0.611	0.458
	-Exposure	23.55	**0.649**	0.418
	-Pose Opt.	23.05	0.625	0.442
	Full	**23.60**	**0.649**	**0.417**

Full model vs. without latent appearance vectors, exposure input, pose refiniement

One large vs. many (smaller) NerFs

- Size in meters
- Top: increasing number of NeRFs with same number of parameters each
- Bottom: increasing number of NeRFs with constant total number of parameters
- Compute time for rendering depends on NeRF size, how many NeRFs rendered per target image (assume average of 2 NeRFs per target view)



# Blocks	Weights / Total	Size	Compute	PSNR↑	SSIM↑	LPIPS↓
1	0.25M / 0.25M	544 m	1×	23.83	0.825	0.381
4	0.25M / 1.00M	271 m	2×	25.55	0.868	0.318
8	0.25M / 2.00M	116 m	2×	26.59	0.890	0.278
16	0.25M / 4.00M	54 m	2×	**27.40**	**0.907**	**0.242**
1	1.00M / 1.00M	544 m	1×	24.90	0.852	0.340
4	0.25M / 1.00M	271 m	0.5×	25.55	0.868	0.318
8	0.13M / 1.00M	116 m	0.25×	25.92	0.875	0.306
16	0.07M / 1.00M	54 m	0.125×	**25.98**	**0.877**	**0.305**

Results

- https://waymo.com/research/block-nerf/

Observation

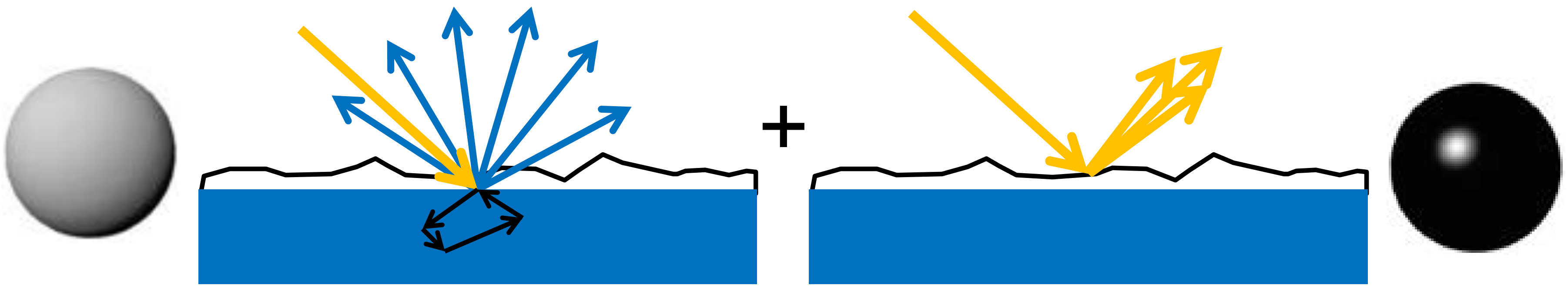
- NeRF predicts radiance along ray in brute force, ignorant of how radiance is produced
  - No model of reflection equation, or light transport
- Idea: build models of light transport into NeRF
  - Could potentially provide more accurate solutions, reconstruct additional scene properties (surface normal, BRDFs, illumination)

Ref-NeRF

- “Ref-NeRF: Structured View-Dependent Appearance for Neural Radiance Fields”, CVPR 2022
- Small step towards modeling light reflection in NeRF framework
- Intuition
  - Reflected light is (roughly) sum of diffuse and glossy
  - Diffuse is view independent
  - Glossy mostly determined by incident light around mirror reflection of viewing direction

Ref-NeRF

- Approach: represent radiance as sum of diffuse and glossy
- For glossy, use directional MLP using mirror reflection of view direction as input, instead of view direction itself



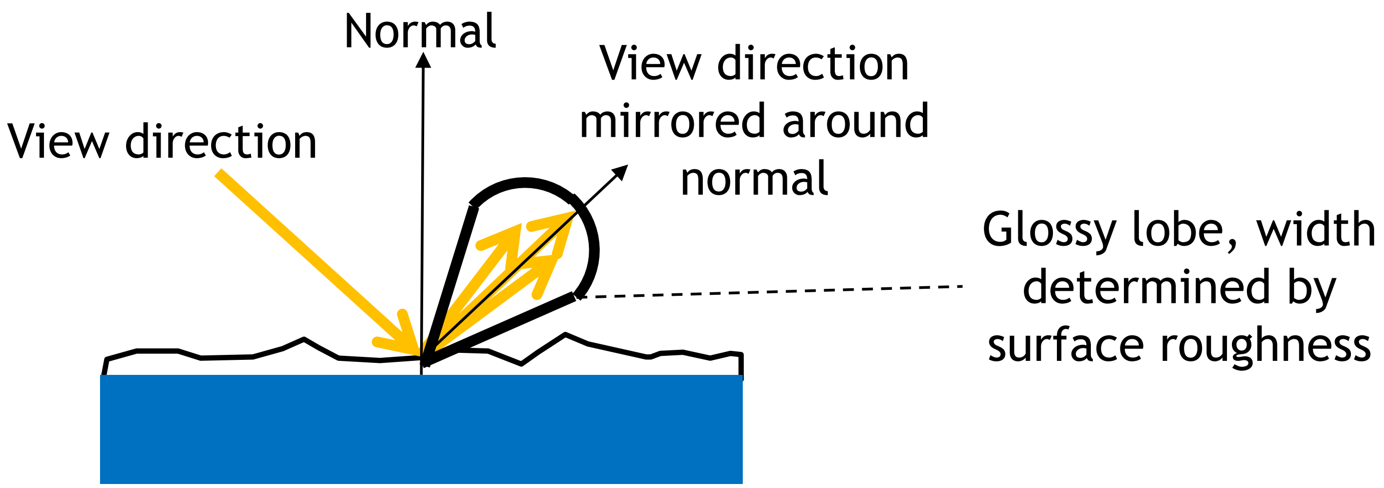
Diffuse

Glossy

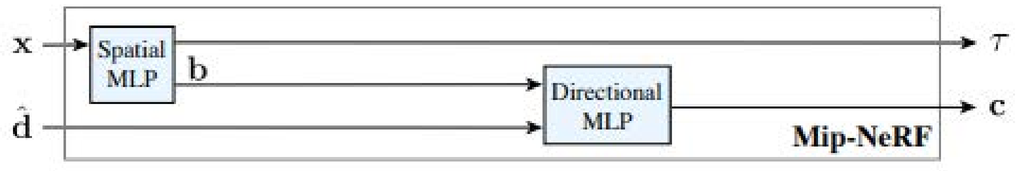
Observation: outgoing radiance determined by incident radiance around mirror reflection

Ref-NeRF

- Additional tricks
  - Clever encoding of mirror reflection direction, including estimate of surface roughness (integrated directional encoding, IDE)
  - Tone mapping function to map radiance to captured pixel colors (models overexposure, non-linear tone-mapping)

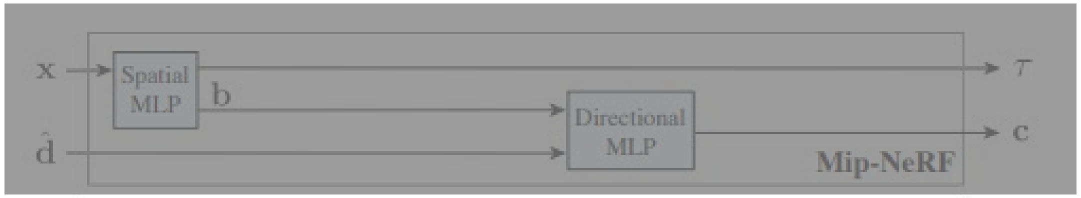

Normal
View direction
View direction mirrored around normal
Glossy lobe, width determined by surface roughness

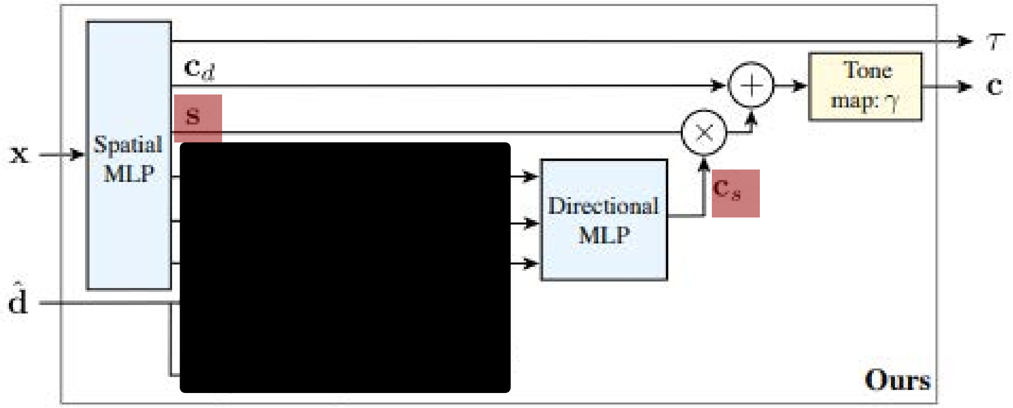
Ref-NeRF



Radiance $\mathbf{c}$, density τ at sample point $\mathbf{x}$, direction $\mathbf{d}$ in conventional NeRF

Ref-NeRF



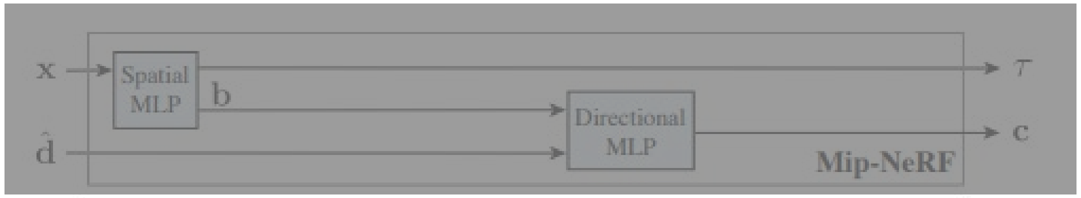


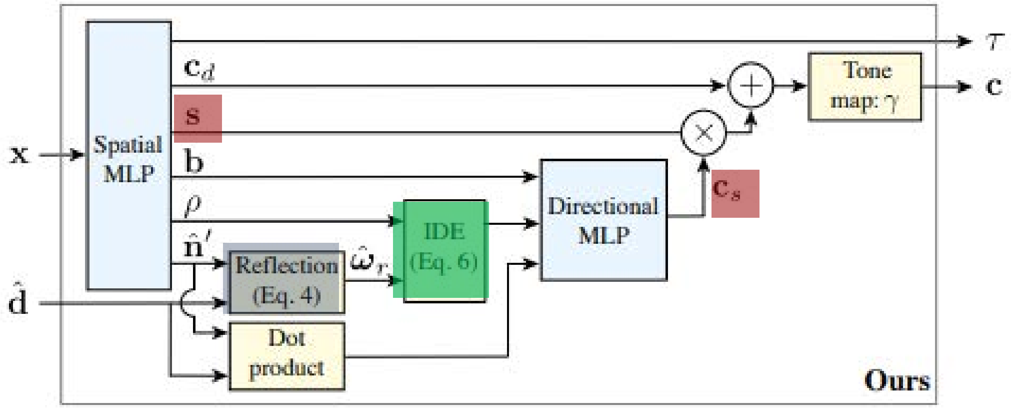
[redacted]

$\mathbf{c}_d$: diffuse radiance
$\mathbf{c}_s$: “incident light averaged over glossy lobe”
$\mathbf{s}$: specular tint, models glossy scattering
γ: tone mapping function
$\mathbf{c}$: radiance at sample point

$$\mathbf{c} = \gamma(\mathbf{c}_d + \mathbf{s} \odot \mathbf{c}_s)$$

Ref-NeRF





c_d: diffuse radiance
c_s: "incident light averaged over glossy lobe"
s: specular tint, models glossy scattering
γ: tone mapping function
c: radiance at sample point

$$\mathbf{c} = \gamma(\mathbf{c}_d + \mathbf{s} \odot \mathbf{c}_s)$$

ρ: surface roughness, determines width of glossy lobe
$\mathbf{n}'$: predicted surface normal

IDE: Integrated directional encoding using mirror reflection of ray direction around normal, represents direction and width of glossy lobe
(details of calculation see paper)

Predicting normal vectors

Regularization terms

- Consistency with gradient of density field

$$\mathcal{R}_\mathrm{p} = \sum_i w_i \|\underbrace{\hat{\mathbf{n}}_i}_{\text{Gradient of density field}} - \underbrace{\hat{\mathbf{n}}'_i}_{\text{Predicted normal by spatial MLP}}\|^2$$

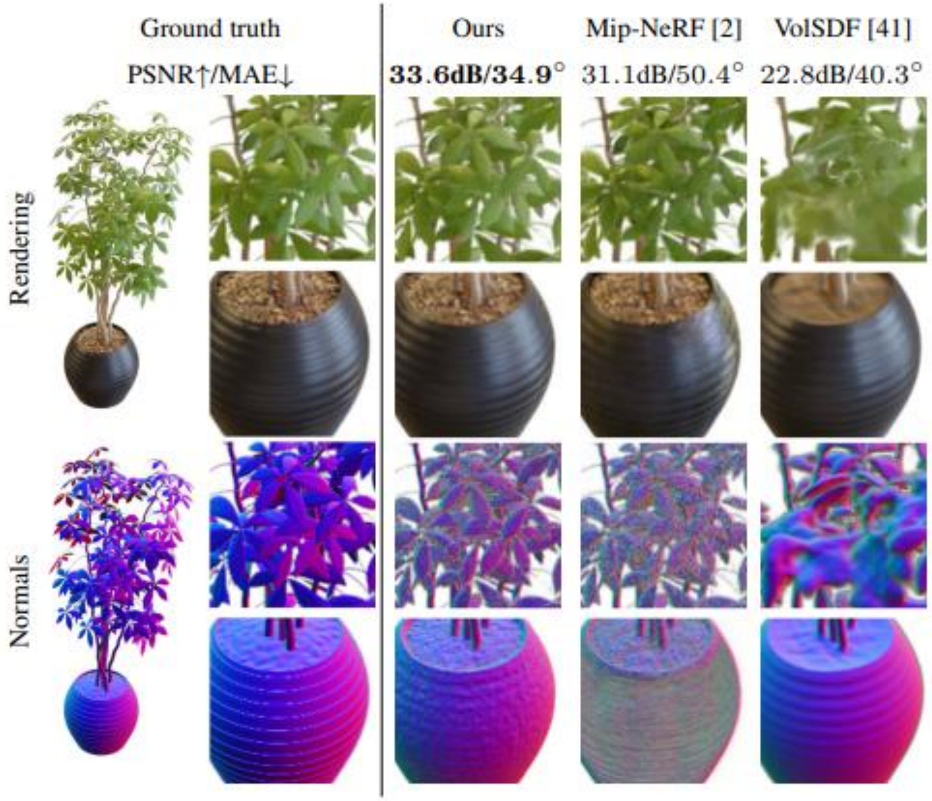
- Avoid backfacing normal (pointing away from camera; backfacing means positive dot product)

$$\mathcal{R}_\mathrm{o} = \sum_i w_i \max(0, \hat{\mathbf{n}}'_i \cdot \hat{\mathbf{d}})^2$$

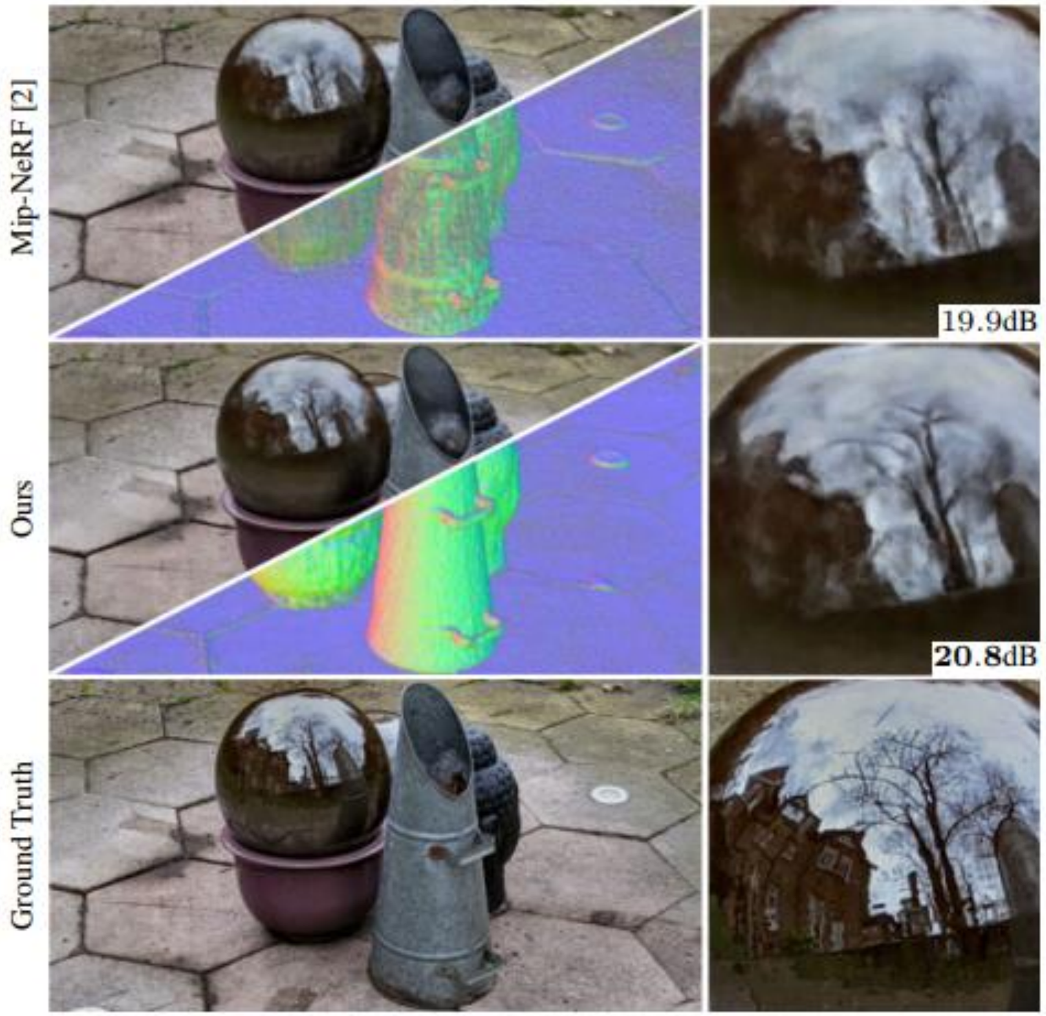
- Sample weight w_i (based on transmittance)

Results

	PSNR ↑	SSIM ↑	LPIPS ↓	MAE° ↓
PhySG [43] (requires object masks)	26.21	0.921	0.121	8.46
Mip-NeRF [2]	29.76	0.942	0.092	60.38
Mip-NeRF, 8 layers	31.59	0.956	0.072	58.07
Mip-NeRF, 8 layers, w/ normals	31.39	0.955	0.074	58.27
Mip-NeRF, 8 layers, w/ $\mathcal{R}_o$	31.48	0.955	0.073	57.37
Ours, no reflection	29.47	0.944	0.084	16.19
Ours, no $\mathcal{R}_o$	31.62	0.954	0.078	52.56
Ours, no pred. normals	30.91	0.936	0.105	30.67
Ours, concat. viewdir	35.42	0.966	0.061	21.25
Ours, fixed lobe	35.52	0.965	0.061	26.46
Ours, no diffuse color	33.32	0.962	0.067	26.13
Ours, no tint	35.45	0.965	0.060	22.70
Ours, no roughness	33.39	0.963	0.065	25.96
Ours, standard encoding	35.90	0.968	0.058	20.31
Ours	35.96	0.967	0.058	18.38

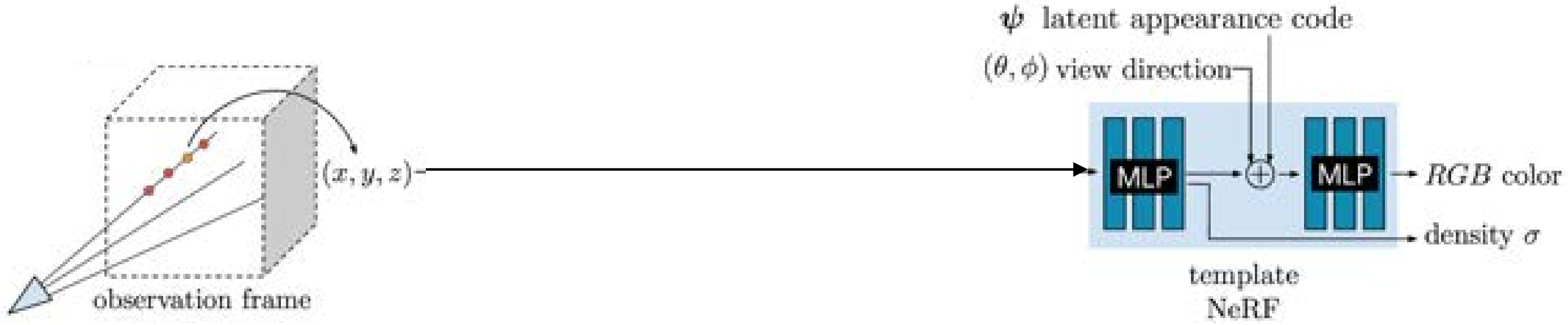


Results



NeRF for videos

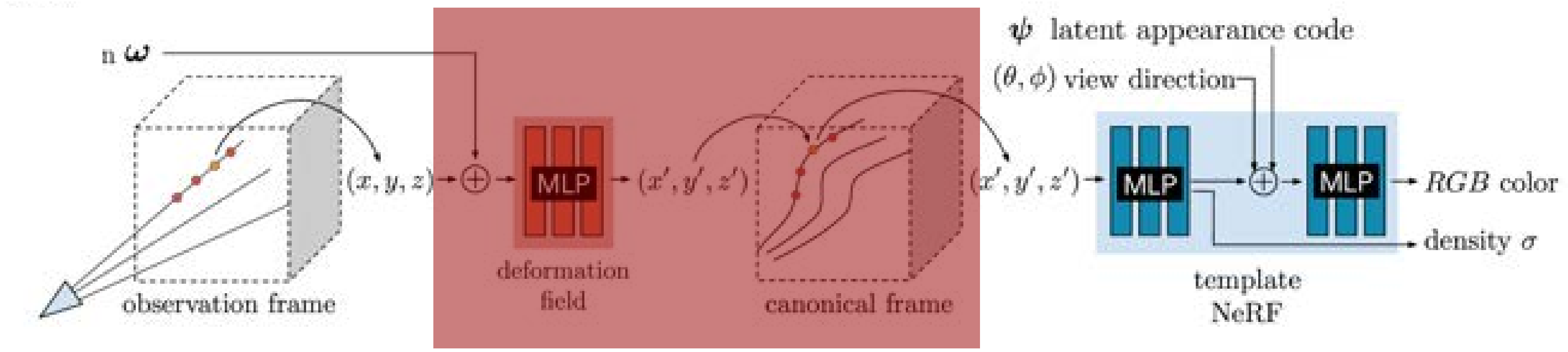
- “Nerfies: Deformable Neural Radiance Fields”, ICCV 2021, https://nerfies.github.io/
- Approach: model 3D deformation field to track moving surfaces



NeRF for static scenes

NeRF for videos

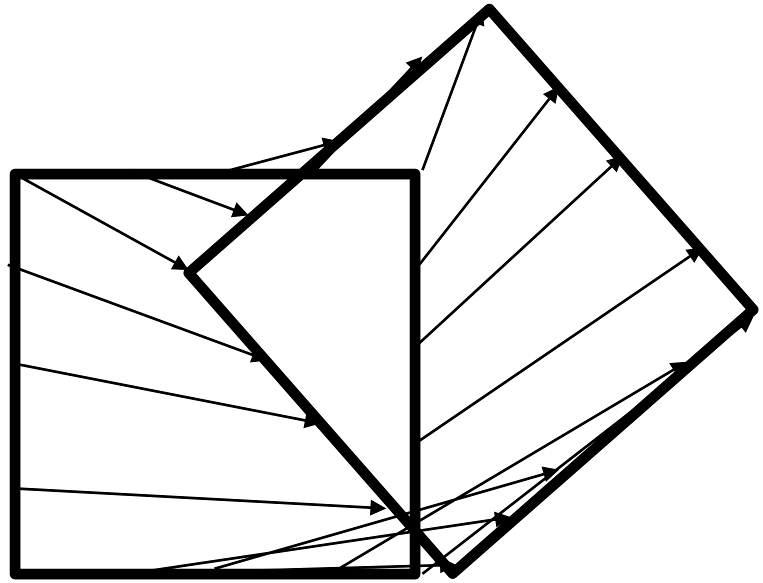
- “Nerfies: Deformable Neural Radiance Fields”, ICCV 2021, https://nerfies.github.io/
- Approach: model 3D deformation field to track moving surfaces



Space deformation to model deformable objects
Deformation modeled using latent vector ω

Mathematical model

- What is good representation for deformation field?
- Simple: displacement field (field of displacement vectors)
- Disadvantage: rigid transformation (rotation, translation) may lead to complicated displacement field


Displacement field for rigid transformation (rotation, translation)

SE(3) field

- Rigid transformations (rotation, translations) $\mathrm{SE}(3)$
- Deformation field $\mathbf{x}$ -> $\mathrm{SE}(3)$, 3D point $\mathbf{x}$
- Advantage: for motion of rigid object, mapping $\mathbf{x}$ -> $\mathrm{SE}(3)$ is constant over $\mathbf{x}$
- $\mathrm{SE}(3)$ parameterized using 6D vector, consisting of axis/angle for rotation $\mathbf{r}$, translation $\mathbf{v}$
  - Corresponding transformation matrix can be computed using Rodrigues' formula (see paper)

Regularization

Three techniques

- Elastic regularization
- Coarse-to-fine training
- Background regularization

Elastic Regularization

- Jacobian of deformation field at each point is 3x3 Jacobian matrix $\mathbf{J}_T$
- Jacobian is "best local linear approximation" of deformation
- Regularization: deformation should be as rigid as possible, that is, Jacobian should be as close as possible to rotation (no scale, shear)
- Using singular value decomposition (SVD), $\mathbf{U}$, $\mathbf{V}$: orthonormal, Σ: diagonal (singular values)

$$\mathbf{J}_T = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

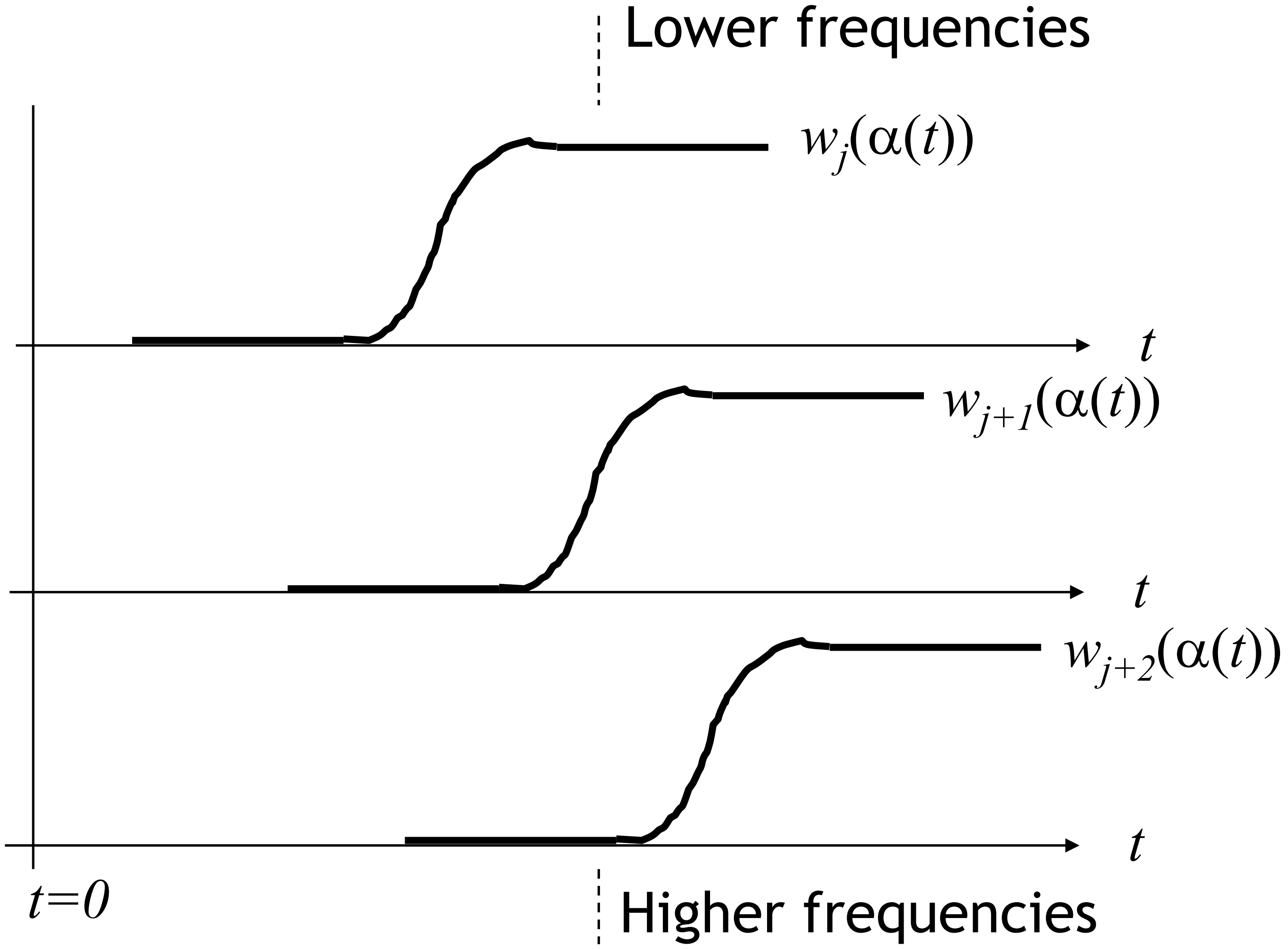
- Different possibilities to constrain $\mathbf{J}_T$, they use

$$L_{\text{elastic}}(\mathbf{x}) = \|\log \mathbf{\Sigma} - \log \mathbf{I}\|_F^2 = \|\log \mathbf{\Sigma}\|_F^2$$

Coarse-to-fine training

- Trade-off between modeling small vs. large deformations
  - Can get trapped in local minima, or produce overly smooth results
- Approach: positional encoding with weights $w_j(\alpha(t))$ that change during training based on value $\alpha(t)$, shifting from low to high frequencies j as training time t progresses

Coarse-to-fine training



Weight functions “activate” higher frequencies later in training

Coarse-to-fine training

- Weight functions

$$\gamma_\alpha(\mathbf{x}) = (\mathbf{x}, \ldots, w_k(\alpha)\sin(2^k\pi\mathbf{x}), w_k(\alpha)\cos(2^k\pi\mathbf{x}), \ldots)$$

Coarse-to-fine positional encoding

$$w_j(\alpha) = \frac{(1 - \cos(\pi\,\mathrm{clamp}(\alpha - j, 0, 1))}{2}$$

Weight function for frequency j

$$\alpha(t) = \frac{mt}{N}$$

t: training iteration
m: max. frequency
N: hyperparameter
"linear annealing"

Background regularization

- Use foreground segmentation to separate dynamic foreground from static background
- Make sure feature points on background don’t move (deformation field is identity transformation)

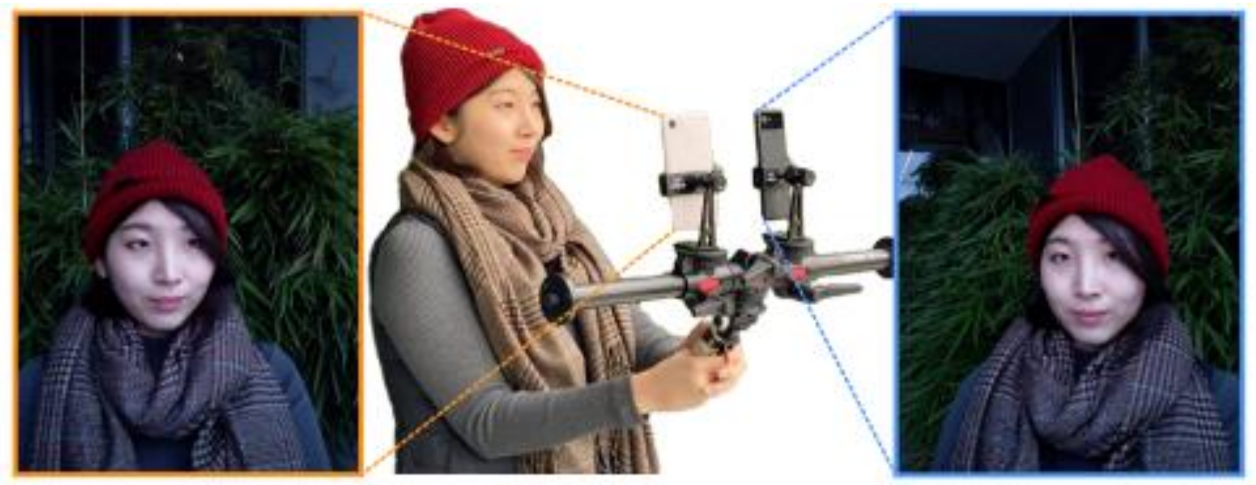
Loss

- Hyperparameter λ for regularization weight

$$L_{\text{total}} = L_{\text{rgb}} + \lambda(L_{\text{elastic}} + L_{\text{bg}})$$

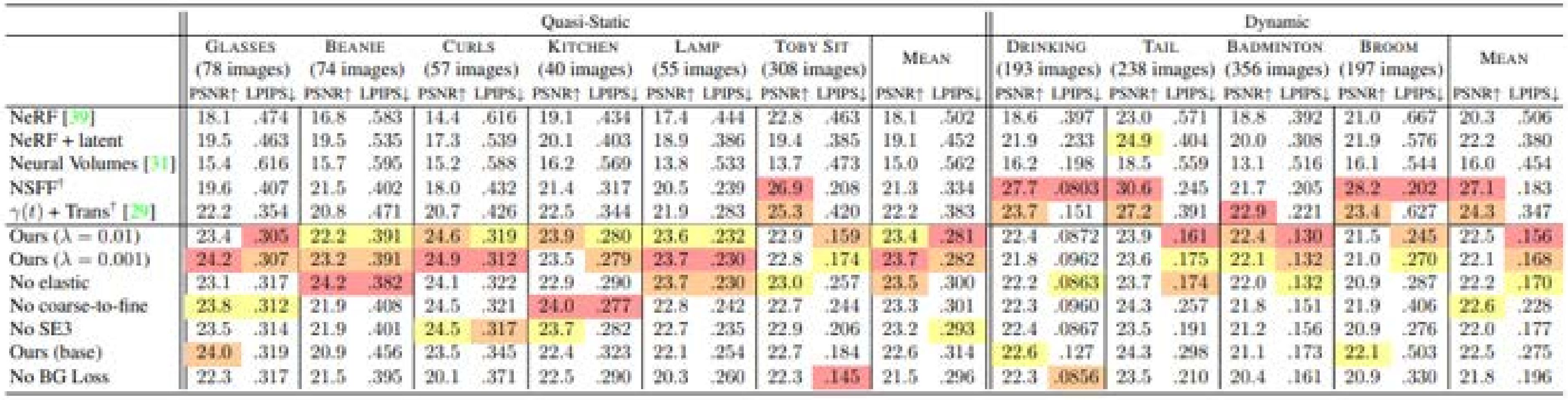
Evaluation

- Capture dynamic scenes with pair of cameras
- Reconstruct NeRF from first camera, render into viewpoint of second camera, compare to ground truth



Ablation study

- Empirical evaluation of effectiveness of regularization techniques, SE(3) deformation field



	Quasi-Static														Dynamic									
	GLASSES (78 images)		BEANIE (74 images)		CURLS (57 images)		KITCHEN (40 images)		LAMP (55 images)		TOBY SIT (308 images)		MEAN		DRINKING (193 images)		TAIL (238 images)		BADMINTON (356 images)		BROOM (197 images)		MEAN	
	PSNR↑	LPIPS↓	PSNR↑	LPIPS↓	PSNR↑	LPIPS↓	PSNR↑	LPIPS↓	PSNR↑	LPIPS↓	PSNR↑	LPIPS↓	PSNR↑	LPIPS↓	PSNR↑	LPIPS↓	PSNR↑	LPIPS↓	PSNR↑	LPIPS↓	PSNR↑	LPIPS↓	PSNR↑	LPIPS↓
NeRF [39]	18.1	.474	16.8	.583	14.4	.616	19.1	.434	17.4	.444	22.8	.463	18.1	.502	18.6	.397	23.0	.571	18.8	.392	21.0	.667	20.3	.506
NeRF + latent	19.5	.463	19.5	.535	17.3	.539	20.1	.403	18.9	.386	19.4	.385	19.1	.452	21.9	.233	24.9	.404	20.0	.308	21.9	.576	22.2	.380
Neural Volumes [31]	15.4	.616	15.7	.595	15.2	.588	16.2	.569	13.8	.533	13.7	.473	15.0	.562	16.2	.198	18.5	.559	13.1	.516	16.1	.544	16.0	.454
NSFF[†]	19.6	.407	21.5	.402	18.0	.432	21.4	.317	20.5	.239	26.9	.208	21.3	.334	27.7	.0803	30.6	.245	21.7	.205	28.2	.202	27.1	.183
$\gamma(t)$ + Trans[†] [29]	22.2	.354	20.8	.471	20.7	.426	22.5	.344	21.9	.283	25.3	.420	22.2	.383	23.7	.151	27.2	.391	22.9	.221	23.4	.627	24.3	.347
Ours ($\lambda = 0.01$)	23.4	.305	22.2	.391	24.6	.319	23.9	.280	23.6	.232	22.9	.159	23.4	.281	22.4	.0872	23.9	.161	22.4	.130	21.5	.245	22.5	.156
Ours ($\lambda = 0.001$)	24.2	.307	23.2	.391	24.9	.312	23.5	.279	23.7	.230	22.8	.174	23.7	.282	21.8	.0962	23.6	.175	22.1	.132	21.0	.270	22.1	.168
No elastic	23.1	.317	24.2	.382	24.1	.322	22.9	.290	23.7	.230	23.0	.257	23.5	.300	22.2	.0863	23.7	.174	22.0	.132	20.9	.287	22.2	.170
No coarse-to-fine	23.8	.312	21.9	.408	24.5	.321	24.0	.277	22.8	.242	22.7	.244	23.3	.301	22.3	.0960	24.3	.257	21.8	.151	21.9	.406	22.6	.228
No SE3	23.5	.314	21.9	.401	24.5	.317	23.7	.282	22.7	.235	22.9	.206	23.2	.293	22.4	.0867	23.5	.191	21.2	.156	20.9	.276	22.0	.177
Ours (base)	24.0	.319	20.9	.456	23.5	.345	22.4	.323	22.1	.254	22.7	.184	22.6	.314	22.6	.127	24.3	.298	21.1	.173	22.1	.503	22.5	.275
No BG Loss	22.3	.317	21.5	.395	20.1	.371	22.5	.290	20.3	.260	22.3	.145	21.5	.296	22.3	.0856	23.5	.210	20.4	.161	20.9	.330	21.8	.196

- Results, links: https://nerfies.github.io/