Lecture 24 - Apr 15

Sunday, March 10, 2024 8:43 PM

Neural Radiance Fields
Structure from Motion (SfM)

References

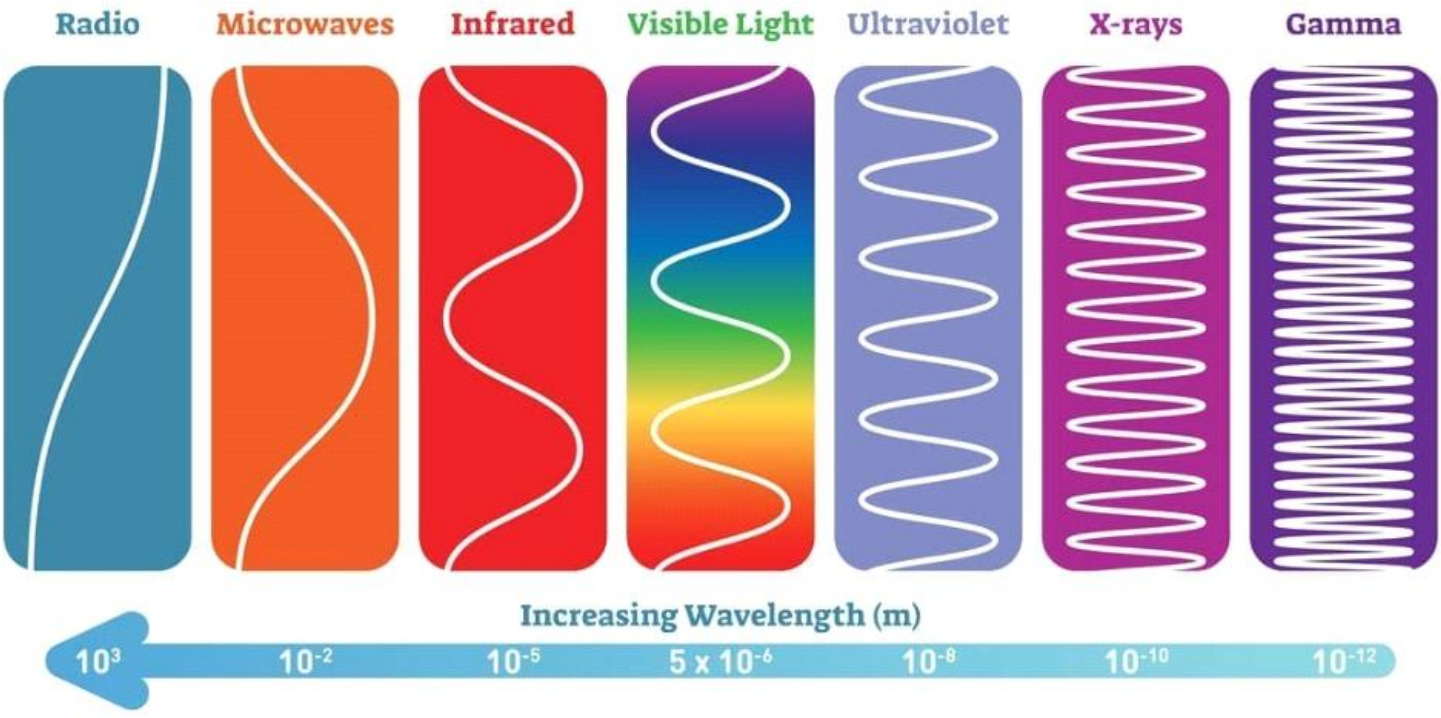
B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, R. Ng (2020). NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. *ECCV 2020*. https://www.matthewtancik.com/nerf

J. Schonberger Lutz and J-M. Frahm (2016). Structure-from-Motion Revisited. *CVPR 2015.*

Geometrical Optics

Saturday, September 10, 2022 12:46 AM

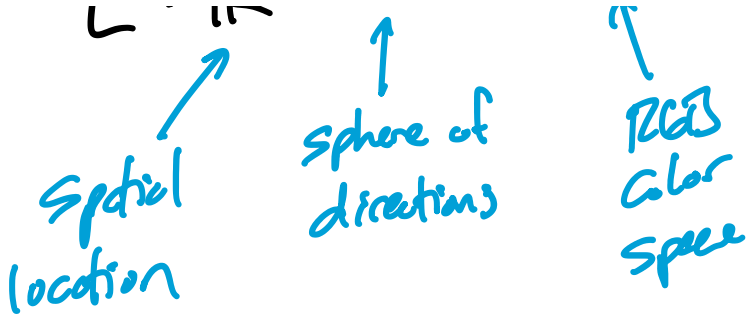
Wave propagation of light



Geometrical optics approximation assumes light propagates along straight lines (rays) with color

EM field simplifies to a radiance field

$$L : \mathbb{R}^3 \times S^2 \to \mathbb{R}^3$$



$$L(x,y,z,\theta,\phi) = (r,g,b)$$

$$L(x,d) = C$$

point in space — unit vector — RGB

Rendering equation shows how field interacts with surfaces

$$L(x,d) = L_e(x,d) + \int_{S^2} L(x,-d')f(d,d')|n\cdot d'|\,dd'$$

normal vector

$L_e(x,d)$: emitted light

$\int_{S^2} L(x,-d')f(d,d')|n\cdot d'|\,dd'$: filters incoming light through the bidirectional scattering distribution function

Rendering equation for volumetrics

radiance emitted per unit length along ray — absorbed light — in- & out-scattering

$$\frac{dL(x+td,d)}{dt} = L_e(x) - [\mu_a(x)+\mu_s(x)]\,L(x,d) + \mu_s(x)\int_{S^2} L(x,d')\,p(d,d')\,dd'$$

Abstract

SOTA synthesis of novel views of complex scenes by optimizing a volumetric scene function using a set of input views

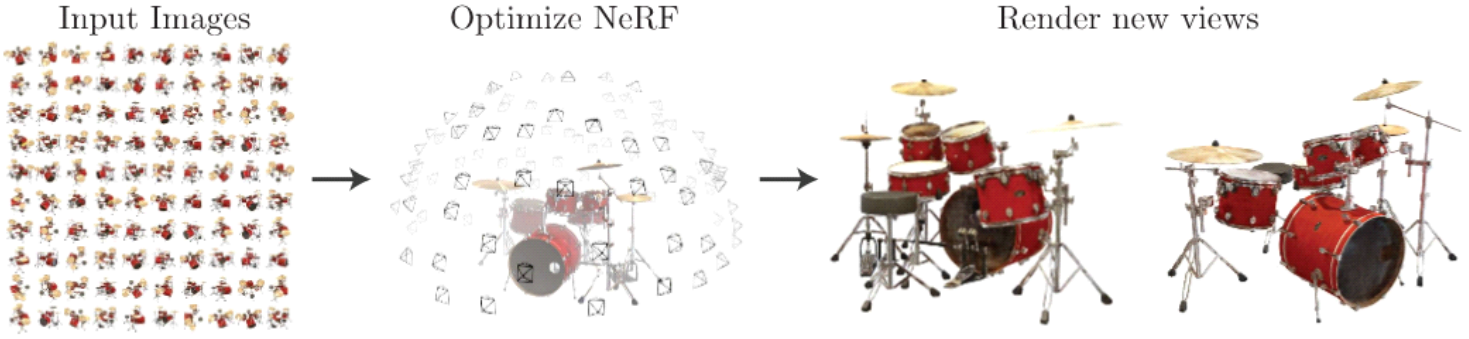


Fig. 1: We present a method that optimizes a continuous 5D neural radiance field representation (volume density and view-dependent color at any continuous location) of a scene from a set of input images. We use techniques from volume rendering to accumulate samples of this scene representation along rays to render the scene from any viewpoint. Here, we visualize the set of 100 input views of the synthetic *Drums* scene randomly captured on a surrounding hemisphere, and we show two novel views rendered from our optimized NeRF representation.

Algo: FC Net represents a scene as

input

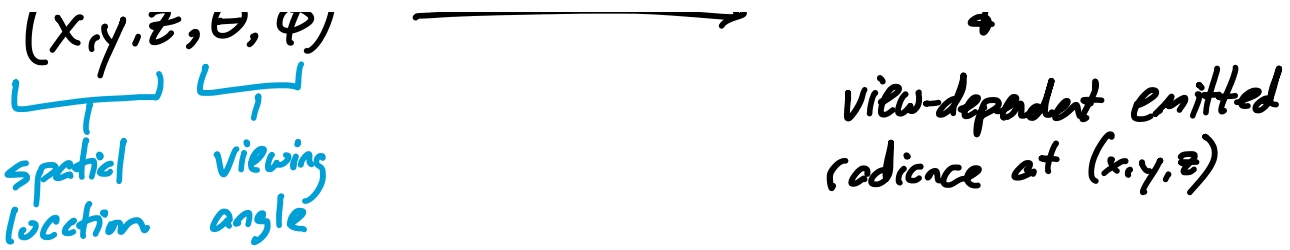
(x, y, z, θ, ϕ) → FC Net →

output

volume density at (x, y, z) (opacity)

+

view-dependent emitted


(x, y, z, θ, φ)
spatial location
viewing angle
view-dependent emitted radiance at (x, y, z)

Directly optimize 5D scene representation to minimize error in rendering a set of images

↳ scene outputs the radiance emitted in directions (θ, ϕ) at each point (x, y, z) and density

The density acts like opacity controlling radiance accumulation at (x, y, z).

$(x, y, z, \theta, \phi) \rightarrow$ MLP $\rightarrow (r, g, b, \sigma)$

(x, y, z): 3D location x; (θ, ϕ): 2D viewing direct.

(r, g, b): angle-dept. color c; σ: volume density

To render this neural radiance field (NeRF) from a given viewpoint:

(1) March camera rays through the scene to generate a sample of 3D points

(2) Feed sampled points + 2D viewing angles into MLP to get (r, g, b, σ)

(3) Classical volume rendering techniques to accumulate colors + densities into a 2D image

Since the process is differentiable, we can use gradient descent to minimize error between corresponding observed + rendered views

Optimizing over multiple scenes encourages the net to predict a coherent model

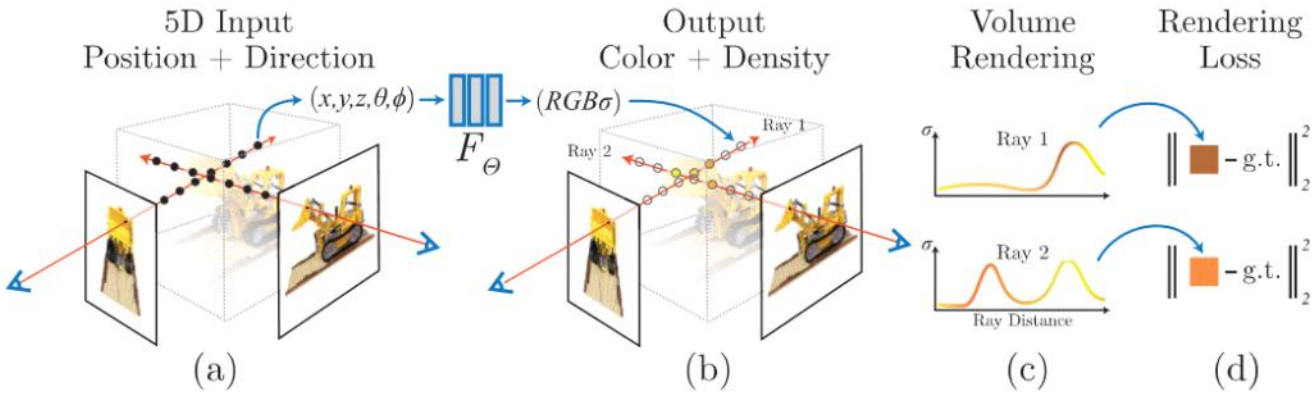


Fig. 2: An overview of our neural radiance field scene representation and differentiable rendering procedure. We synthesize images by sampling 5D coordinates (location and viewing direction) along camera rays (a), feeding those locations into an MLP to produce a color and volume density (b), and using volume rendering techniques to composite these values into an image (c). This rendering function is differentiable, so we can optimize our scene representation by minimizing the residual between synthesized and ground truth observed images (d).

Represent viewing angle (θ, ϕ) as 3D unit vector d

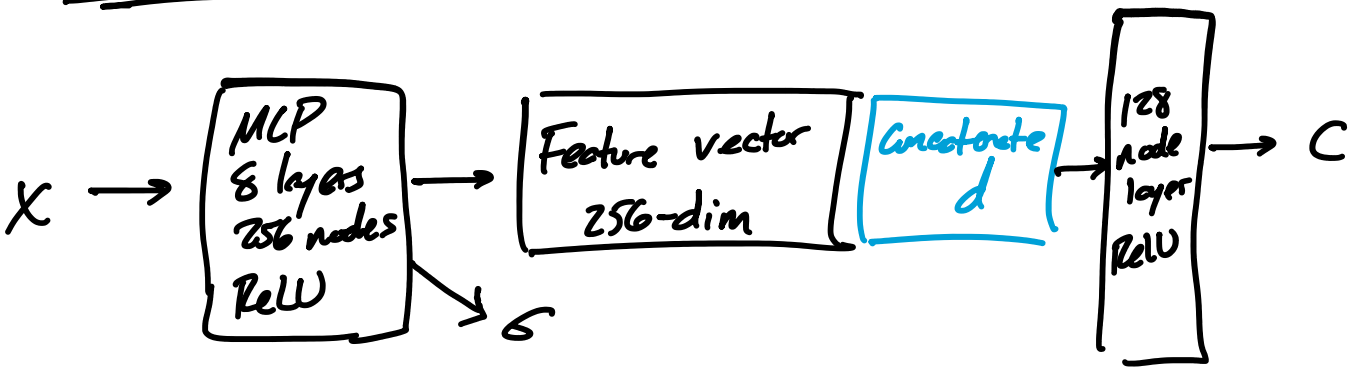
$x = (x, y, z)$, $c = (r, g, b)$.

Scene representation

$$F_\Theta : (x, d) \rightarrow (c, \sigma)$$

MLP

Restriction: σ depends on x but not d



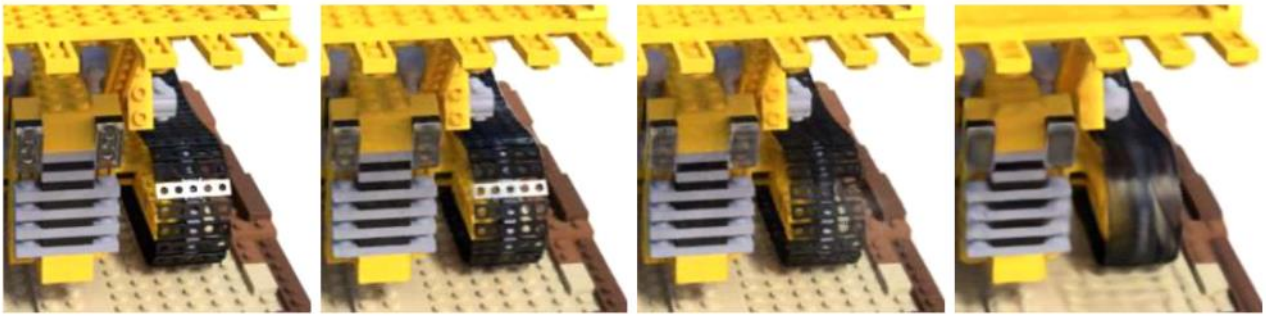


Fig. 4: Here we visualize how our full model benefits from representing view-dependent emitted radiance and from passing our input coordinates through a high-frequency positional encoding. Removing view dependence prevents the model from recreating the specular reflection on the bulldozer tread. Removing the positional encoding drastically decreases the model's ability to represent high frequency geometry and texture, resulting in an oversmoothed appearance.

d as input captures reflections

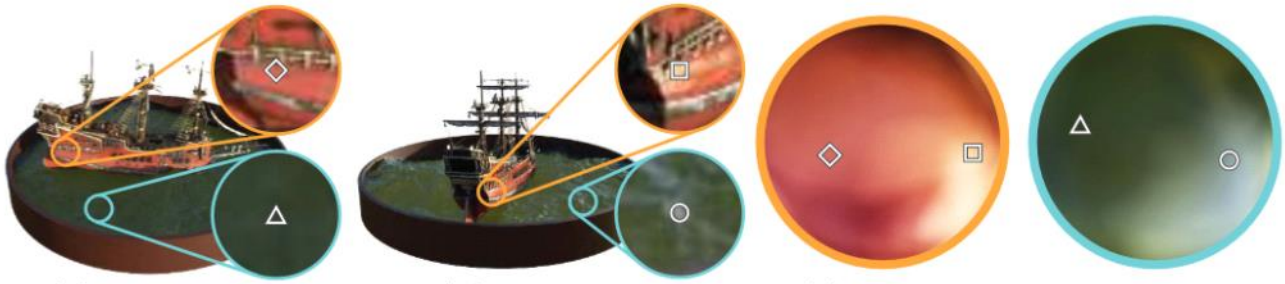


Fig. 3: A visualization of view-dependent emitted radiance. Our neural radiance field representation outputs RGB color as a 5D function of both spatial position $\mathbf{x}$ and viewing direction $\mathbf{d}$. Here, we visualize example directional color distributions for two spatial locations in our neural representation of the *Ship* scene. In (a) and (b), we show the appearance of two fixed 3D points from two different camera positions: one on the side of the ship (orange insets) and one on the surface of the water (blue insets). Our method predicts the changing specular appearance of these two 3D points, and in (c) we show how this behavior generalizes continuously across the whole hemisphere of viewing directions.

d helps model learn color depends on angle of view

NeRF represents scene as volume density σ and directional emitted radiance from any $x \in \mathbb{R}^3$

Render color of any ray passing through the scene using classical volume rendering

Volume density $\sigma(x)$ is the $dP = f\,dV$ (pdf, infinitesimal volume)

probability density of a ray terminating at a point x

The expected color $C(r)$ of camera ray $r(t) = o + td$ with $t_n < t < t_f$ is

(t_n: near bound; t_f: far bound)

$$C(r) = \int_{t_n}^{t_f} T(t)\,\sigma(r(t))\,c(r(t), d)\,dt \qquad (1)$$

(σ: density; c: color intensity)

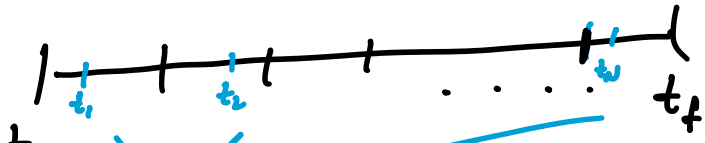
where $T(t) = \exp\left(-\int_{t_n}^{t} \sigma(r(s))\,ds\right)$

accumulated light

accumulated light
along the ray from t_n to t
i.e. probability ray goes from
t_n to t without hitting
another particle

Rendering a view requires estimating $C(r)$ for a camera ray traced through each pixel

We use numerical integration to estimate. Deterministic approaches only sample MLP at fixed locations, reducing resolution, so a stochastic version is used, sampling as:



(2)

sampled uniformly in subintervals (stratified sampling)

$\Rightarrow$ MLP is evaluated at continuous positions during training

$$\hat{C}(r) = \sum_{i=1}^{N} T_i \left(1 - e^{-\sigma_i (t_{in} - t_i)}\right) c_i \qquad (3)$$

$$T_i = \exp\left(-\sum_{j=1}^{i-1} (t_{j+1} - t_j)\sigma_j\right)$$

This is differentiable

Improvements: (1) Positional encoding to assist MLP to represent high-frequency functions

(2) Hierarchical sampling to efficiently sample this high-frequency function

5.1 Positional Encoding

F_Θ was poor at high-freq variation in color/geometry

↳ mapping inputs to a higher-dim space with high-freq functions improves this

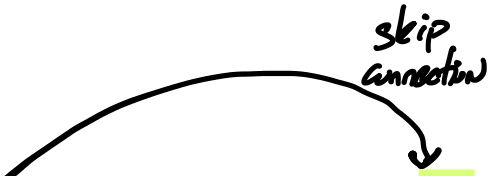
Reformulate $F_\Theta = G_\Theta \circ \gamma$

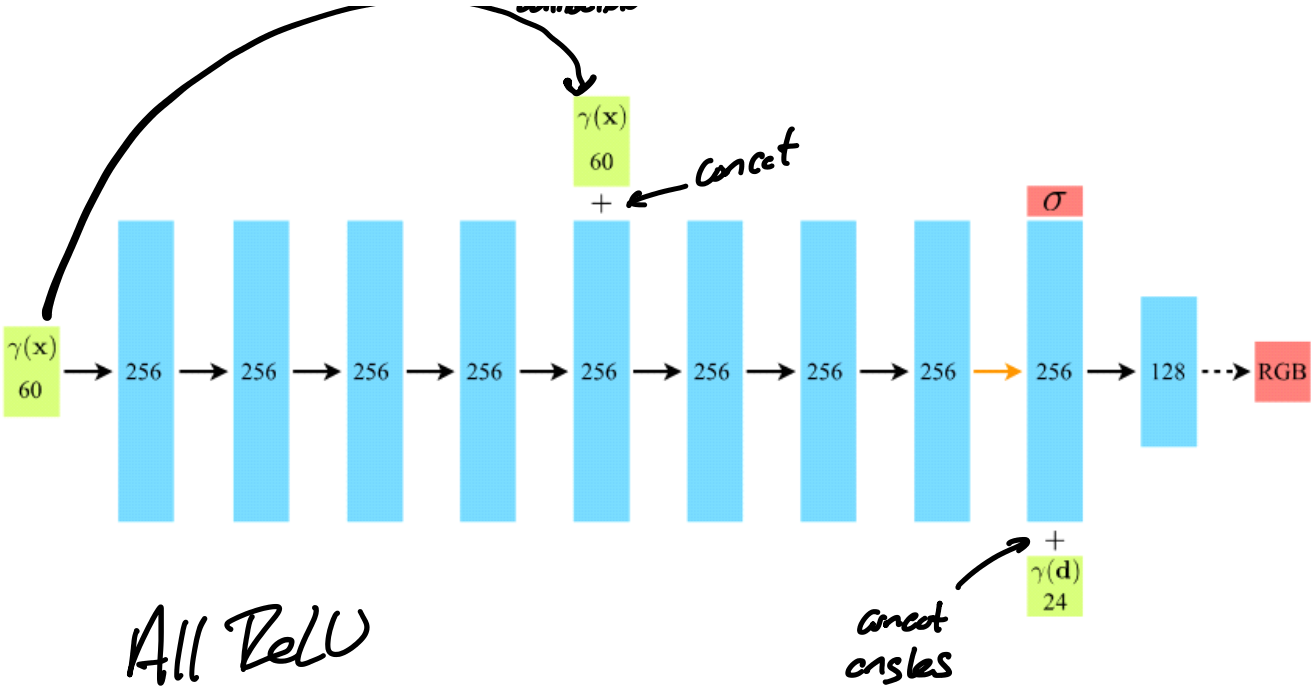
$\gamma : \mathbb{R} \to \mathbb{R}^{2L}$

$$\gamma(p) = \left(\sin(2^0 \pi p), \cos(2^0 \pi p), \ldots, \sin(2^{L-1} \pi p), \cos(2^{L-1} \pi p)\right) \quad (4)$$

where γ is applied to components of $x + d$ (all in $[-1,1]$) separately

↳ they use $\gamma(x)$ with $L = 10$ + $\gamma(d)$ with $L = 4$





5.2 Hierarchical Volume Sampling

Densely evaluating the NeRF net at N query points along each camera ray is inefficient

↳ occluded regions + empty space are sampled repeatedly but have no effect on the rendered image

↳ hierarchical sampling that allocates samples proportionally to expected effect on rendering

Optimize 2 networks to represent the scene: Coarse → fine

Optimize 2 networks

(a) Sample N_c locations by (2)-(3) for coarse net

(b) Sample points along each ray biased toward more relevant parts of the volume

↳ rewrite color from coarse net $\hat{C}_c(r)$ in (3) as weighted sum of sampled colors c_i along the ray:

$$\hat{C}_c(r) = \sum_{i=1}^{N_c} \underbrace{T_i\left(1-\exp(-\sigma_i\delta_i)\right)}_{w_i \text{ multiplier for color point}} c_i \qquad (5)$$

α_i opacity in a commonly-used form "alpha compositing"

Normalizing weights $\hat{w}_i = \frac{w_i}{\sum_{j=1}^{N_c} w_j}$ creates a pw-constant pdf along the ray

↳ Sample N_f locations from this distribution with inverse transform sampling (non-uniform discretization)

↳ evaluate fine net at $N_f + N_c$ points

↳ Compute $\hat{C}_f(r)$ rendered color of ray by (3)

This process allocates more samples to regions expected to have visible content

to have visible content

5.3 Implementation Details

Optimize a separate neural continuous volume representation network for each scene (with known camera poses, intrinsic parameters, and bounds or use COLMAP to estimate it)

$$\text{Loss: } \mathcal{L} = \sum_{r \in \mathcal{R}} \left[\| \hat{C}_c(r) - C(r) \|_2^2 + \| \hat{C}_f(r) - C(r) \|_2^2 \right] \quad (6)$$

- $\mathcal{R}$: set of rays in each batch
- $\| \hat{C}_c(r) - C(r) \|_2^2$: L^2 error for coarse render
- $\| \hat{C}_f(r) - C(r) \|_2^2$: L^2 error for fine render

Minimize coarse loss too so coarse weight distribution can be used to allocate samples in the fine net

Experiments: batch size 4096 rays
Coarse sample size $N_c = 64$
fine sample size $N_f = 128$
Adam with $\alpha = 5 \times 10^{-4} \longrightarrow 5 \times 10^{-5}$
$\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-7}$

$\beta_1 = 0.9, \beta_2 = 0.777, \varepsilon = 10^{-}$

100-300k iterations (1-2 days on V100)

NeRF outperforms prior methods + ablation studies

6.1 Datasets

Synthetic renderings: Diffuse Synthetic 360°
Realistic Synthetic 360°

4 Lambertian objects with simple geometry : DeepVoxels (512×512 viewpoints on upper hemisphere: 479 input 1000 test)

Pathtraced images of 8 non-Lambertian objects with complex geometry : original (800×800, 6 w/ upper hemisphere viewpoints, 2 w/ full sphere viewpoints, 100 input, 200 test)

Real images of complex scenes from phone cam : 3 original, 5 from LLFF (roughly forward-facing, 20-62 images, 1008 × 756 px)

6.2 Comparisons Metrics

Peak signal to noise ratio (PSNR)

Peak signal to noise ratio (PSNR)

$$PSNR(f,g) = 10\log_{10}\left(\frac{255^2}{\|f-g\|_2^2}\right)$$ (grayscale)

f near g $\Rightarrow$ high PSNR ✓

Structural similarity index measure (SSIM)

$$SSIM(f,g) = l(f,g)\,c(f,g)\,s(f,g)$$ ← each in $[0,1]$

- l: diff in luminance
- c: diff in contrast
- s: correlation coefficient

Learned perceptual image patch similarity (LPIPS)

↳ internal activations of trained on classification (e.g. ImageNet) correspond to human perception far better than SSIM/PSNR

↳ deep features trained on supervised, unsupervised, and self-supervised tasks all work

Method	Diffuse Synthetic 360° [41]			Realistic Synthetic 360°			Real Forward-Facing [28]		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
SRN [42]	33.20	0.963	0.073	22.26	0.846	0.170	22.84	0.668	0.378
NV [24]	29.62	0.929	0.099	26.05	0.893	0.160	-	-	-
LLFF [28]	34.38	0.985	0.048	24.88	0.911	0.114	24.13	0.798	**0.212**
Ours	**40.15**	**0.991**	**0.023**	**31.01**	**0.947**	**0.081**	**26.50**	**0.811**	0.250

Table 1: Our method quantitatively outperforms prior work on datasets of both synthetic and real images. We report PSNR/SSIM (higher is better) and LPIPS [50] (lower is better). The DeepVoxels [41] dataset consists of 4 diffuse objects with simple geometry. Our realistic synthetic dataset consists of pathtraced renderings of 8 geometrically complex objects with complex non-Lambertian materials. The real dataset consists of handheld forward-facing captures of 8 real-world scenes (NV cannot be evaluated on this data because it only reconstructs objects inside a bounded volume). Though LLFF achieves slightly better LPIPS, we urge readers to view our supplementary video where our method achieves better multiview consistency and produces fewer artifacts than all baselines.

Fig. 5: Comparisons on test-set views for scenes from our new synthetic dataset generated with a physically-based renderer. Our method is able to recover fine details in both geometry and appearance, such as *Ship*'s rigging, *Lego*'s gear and treads, *Microphone*'s shiny stand and mesh grille, and *Material*'s non-Lambertian reflectance. LLFF exhibits banding artifacts on the *Microphone* stand and *Material*'s object edges and ghosting artifacts in *Ship*'s mast and inside the *Lego* object. SRN produces blurry and distorted renderings in every case. Neural Volumes cannot capture the details on the *Microphone*'s grille or *Lego*'s gears, and it completely fails to recover the geometry of *Ship*'s rigging.

Fig. 6: Comparisons on test-set views of real world scenes. LLFF is specifically designed for this use case (forward-facing captures of real scenes). Our method is able to represent fine geometry more consistently across rendered views than LLFF, as shown in *Fern*'s leaves and the skeleton ribs and railing in *T-rex*. Our method also correctly reconstructs partially occluded regions that LLFF struggles to render cleanly, such as the yellow shelves behind the leaves in the bottom *Fern* crop and green leaves in the background of the bottom *Orchid* crop. Blending between multiples renderings can also cause repeated edges in LLFF, as seen in the top *Orchid* crop. SRN captures the low-frequency geometry and color variation in each scene but is unable to reproduce any fine detail.

Other methods: { Neural Volumes (NV) → expensive voxels cannot scale to fine details
Scene Representation Nets (SRN) → overly smoothed
train net for each scene
viewpoint

train net for each scene] Scene Representation Nets (SRN) - smoothed

Single 3D CNN on a large dataset → Local Light Field Fusion (LLFF) → viewpoint inconsistency

6.3 Discussion

All >12 hrs to train except LLFF (10 min)

NeRF 5MB LLFF 15GB

6.4 Ablation Studies

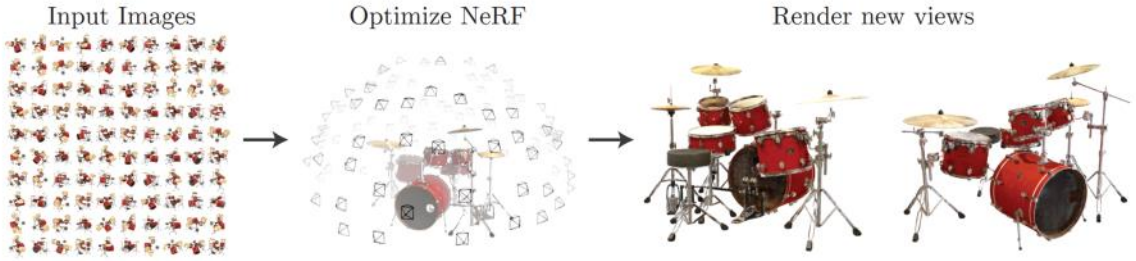
	Input	#Im.	L	(N_c, N_f)	PSNR↑	SSIM↑	LPIPS↓
1) No PE, VD, H	xyz	100	-	(256, -)	26.67	0.906	0.136
2) No Pos. Encoding	$xyz\theta\phi$	100	-	(64, 128)	28.77	0.924	0.108
3) No View Dependence	xyz	100	10	(64, 128)	27.66	0.925	0.117
4) No Hierarchical	$xyz\theta\phi$	100	10	(256, -)	30.06	0.938	0.109
5) Far Fewer Images	$xyz\theta\phi$	25	10	(64, 128)	27.78	0.925	0.107
6) Fewer Images	$xyz\theta\phi$	50	10	(64, 128)	29.79	0.940	0.096
7) Fewer Frequencies	$xyz\theta\phi$	100	5	(64, 128)	30.59	0.944	0.088
8) More Frequencies	$xyz\theta\phi$	100	15	(64, 128)	30.81	0.946	0.096
9) Complete Model	$xyz\theta\phi$	100	10	(64, 128)	**31.01**	**0.947**	**0.081**

Table 2: An ablation study of our model. Metrics are averaged over the 8 scenes from our realistic synthetic dataset. See Sec. 6.4 for detailed descriptions.

Slides

Tuesday, March 26, 2024 12:35 AM

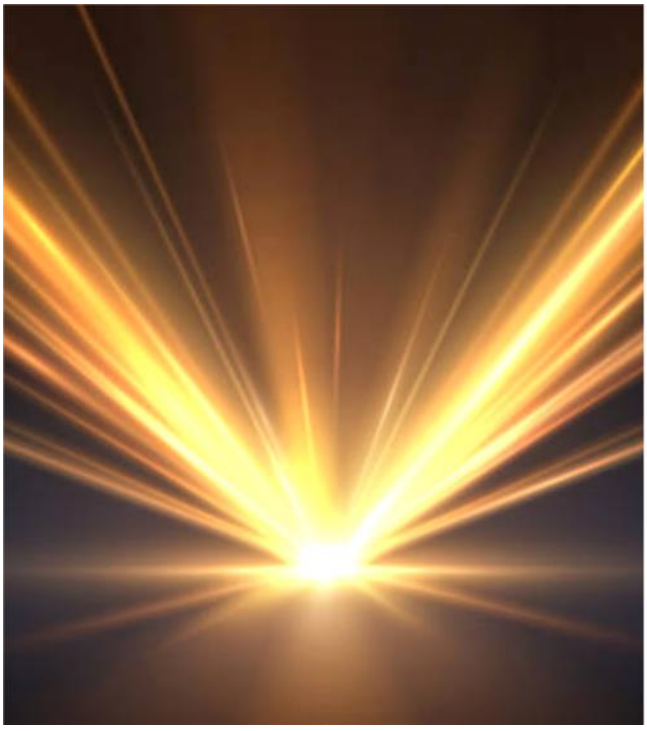
Paper B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. **NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis**. *European Conference on Computer Vision (ECCV) 2020.*



NeRF Examples (Synthetic Data)

NeRF Examples (Real Data)

NeRF Example for AR



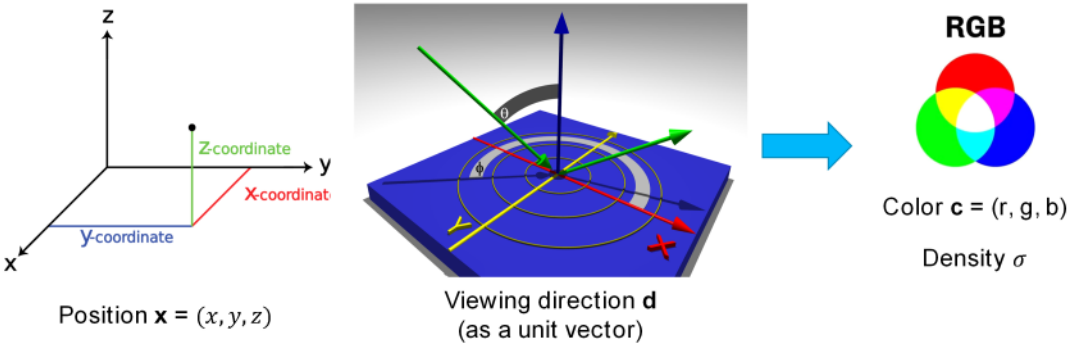
Ray Optics Simplification

- Assumes light travels along straight-line rays
- No wave propagation–ignore interference, scattering, etc.
- Electromagnetic field becomes a radiance field

Radiance Fields

A **radiance field** maps all 3D points **and** viewing angles of them to RGB color and density (opacity)

$$F: [-1,1]^3 \times S^2 \rightarrow [0,255]^3 \times [0,1]$$



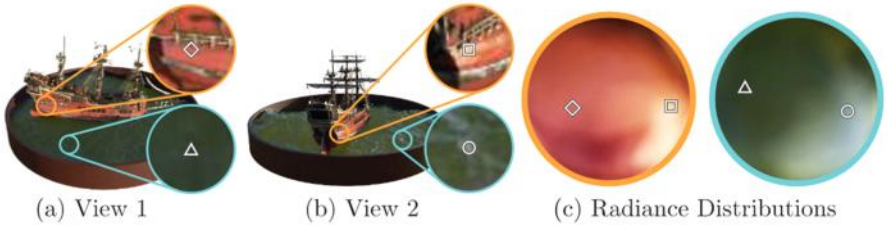
Position **x** = (x, y, z)

Viewing direction **d** (as a unit vector)

Color **c** = (r, g, b)

Density σ

View Dependence



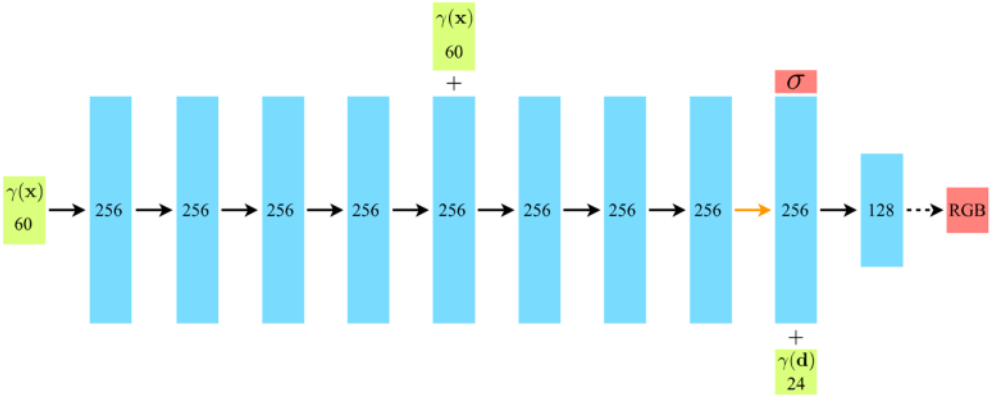
(a) View 1 (b) View 2 (c) Radiance Distributions

A point **x** should map to **different colors** but the **same densities** for different the viewing directions **d**

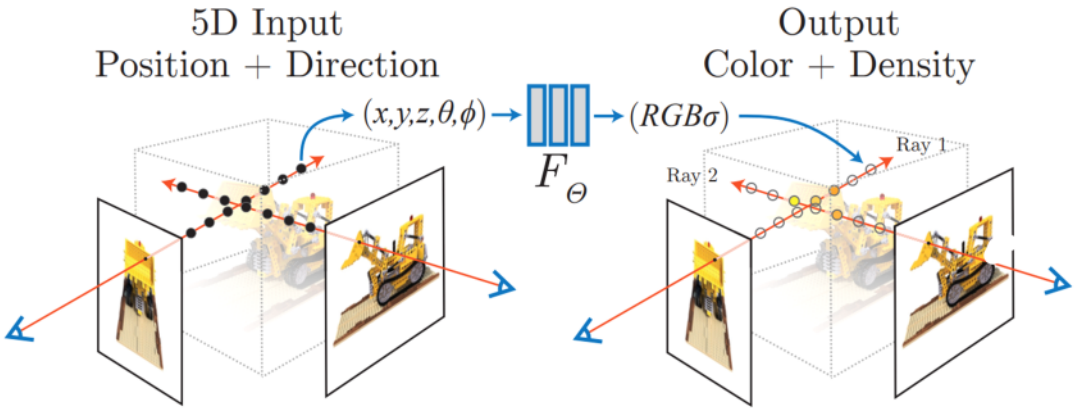
- Density $\sigma(\mathbf{x})$
- Color $\mathbf{c}(\mathbf{x}, \mathbf{d})$

Neural Radiance Field (NeRF)

NeRF represents a radiance field as a **fully-connected** neural network with learnable parameters Θ



Scene Synthesis (Step 1)

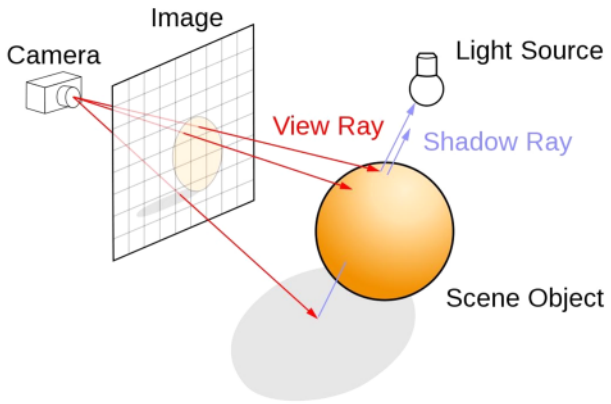


March rays through the scene to generate a sample of 3D points

Feed sampled 3D points through the NeRF to estimate (**c**, σ)

Scene Synthesis (Step 2)

Use **ray tracing** to accumulate colors and densities along rays through the scene to estimate each pixel color

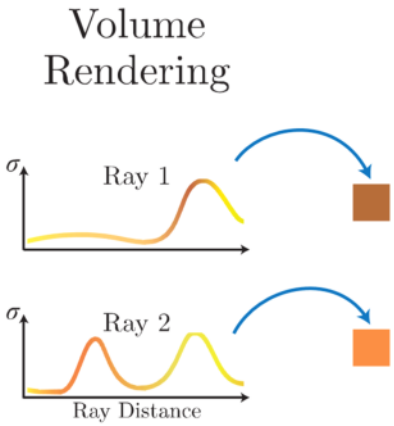


Volume Rendering via Ray Tracing

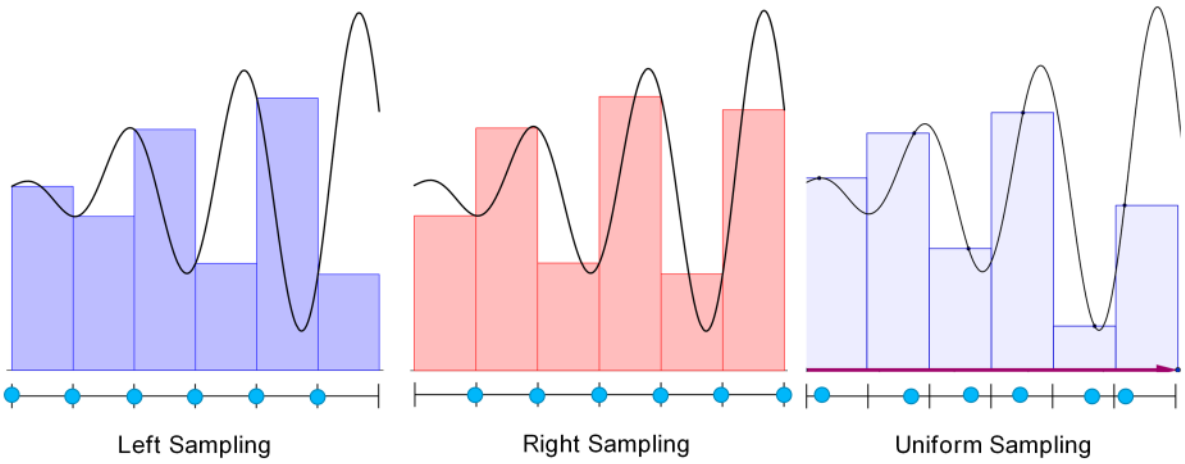
- Consider ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ for $t_n \le t \le t_f$
- The expected color of the ray is

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t) \underbrace{\sigma(\mathbf{r}(t))}_{\text{density}} \underbrace{\mathbf{c}(\mathbf{r}(t), \mathbf{d})}_{\text{color}} \, dt$$

- $T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s)) ds\right)$ is the probability the ray travels from t_n to t without hitting another particle
- C(**r**) for each of RGB gives the pixel color



Numerical Integration



Estimating the Integral

Each t_i is selected uniformly at random within mesh subintervals with $\Delta_i = t_{i+1} - t_i$

Expected color $C(\mathbf{r})$ is estimated as:

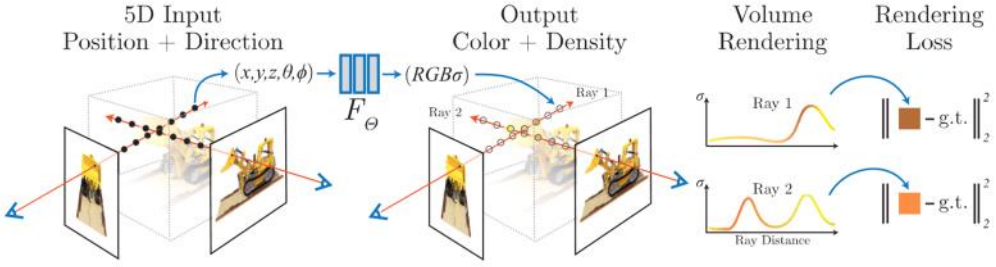
$$T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \Delta_j\right)$$

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i \left(1 - e^{-\sigma_i \Delta_i}\right) \mathbf{c}_i$$

where $\mathbf{c}_i$ and σ_i are the color and density at $\mathbf{r}(t_i)$ along the ray

Training a NeRF

Images of a target object from different viewing directions are used to learn the NeRF based on error reconstructing these same images.



Key feature: The loss is differentiable w.r.t. $\mathbf{c}_i$, σ_i, and the parameters Θ

Derivatives can be backpropagated through the **entire pipeline** to enable gradient-based learning end-to-end

Positional Encoding

- Training the NeRF F_Θ to operate on the input (**x**, **d**) results in renderings that poorly reproduce high-frequency changes in color and geometry
- Sin-cosine positional encoding is used

$$\gamma(p) = \begin{bmatrix} \sin(2^0 \pi p) \\ \cos(2^0 \pi p) \\ \vdots \\ \sin(2^{L-1} \pi p) \\ \cos(2^{L-1} \pi p) \end{bmatrix}$$

- They use $L = 10$ for $\gamma(\mathbf{x})$ and $L = 4$ for $\gamma(\mathbf{d})$

Ground Truth | Complete Model | No Positional Encoding

Speeding Up Ray Tracing

- Ray tracing densely evaluates the network at N points along each ray, which is **expensive**
- Free space and occluded regions are **irrelevant** to the rendered image, but they are sampled, which is **inefficient**
- **Hierarchical sampling** allocates samples proportionally to the expected effect of a region by introducing two networks:
  - A "coarse" network samples $N_c = 64$ points as above to get color estimates $C_c(\mathbf{r})$
  - A "fine" network samples $N_f = 128$ points via **importance sampling**
- Final color $C_f(\mathbf{r})$ is computed based on **all $N_c + N_f$ points**

Implementation

- Loss from both coarse and fine networks for a batch of rays $\mathcal{R}$

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

- Optimizing the coarse loss via $C_c(\mathbf{r})$ helps the net learn the importance sampling weights for the fine network

- Batch size 4096, Adam optimizer, 100-300k iterations on NVIDIA V100 GPU (1-2 days)

Structure-from-Motion (SfM) and COLMAP

Tuesday, March 26, 2024 12:39 AM