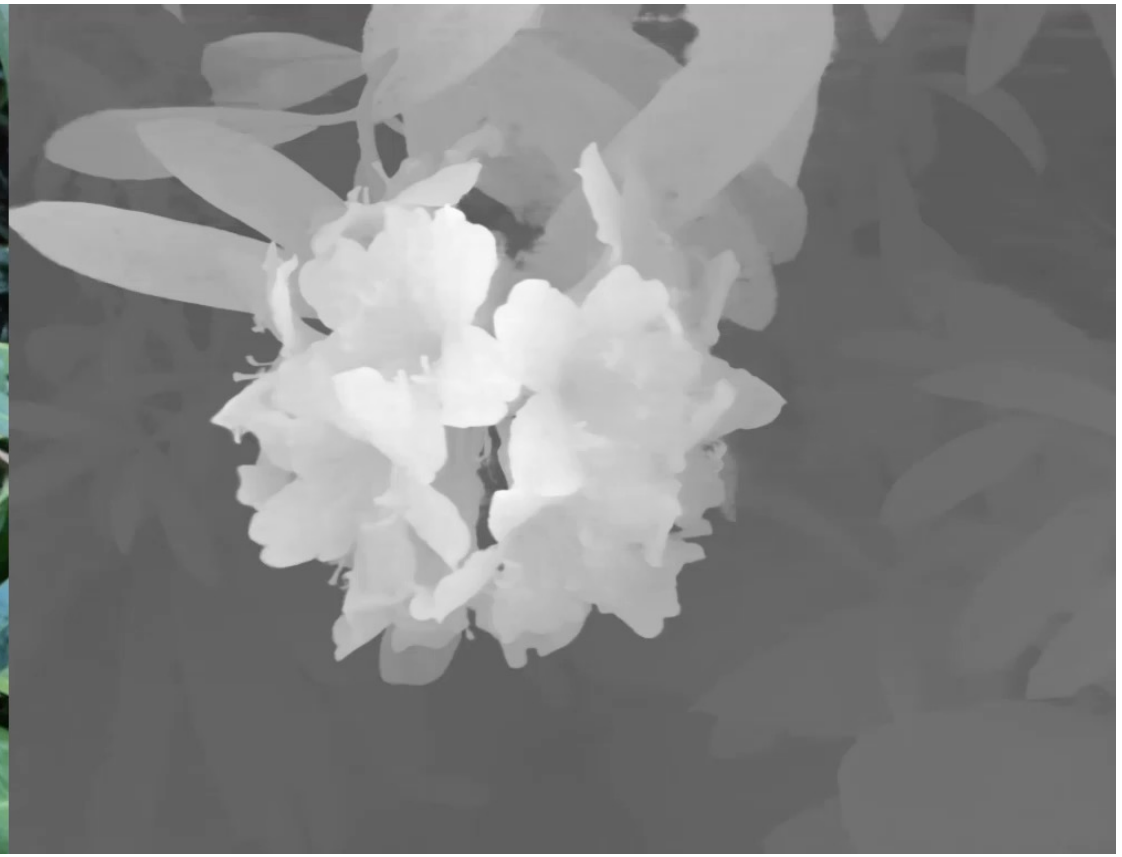
NeRF: Representing Scenes as Neural Radiance Fields

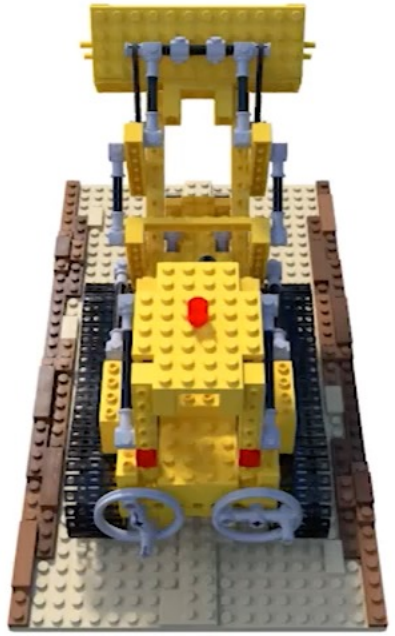
Володин Даниил 212

Content

1. Problem of view synthesis
2. Neural Radiance Field Scene Representation
3. Network
4. Volume Rendering and Expected Color
5. Intermediate results
6. Optimizations
    1. Positional Encoding
    2. Hierarchical volume sampling
7. Hyperparameters
8. Results and comparisons
9. Conclusion

1. Problem of view synthesis



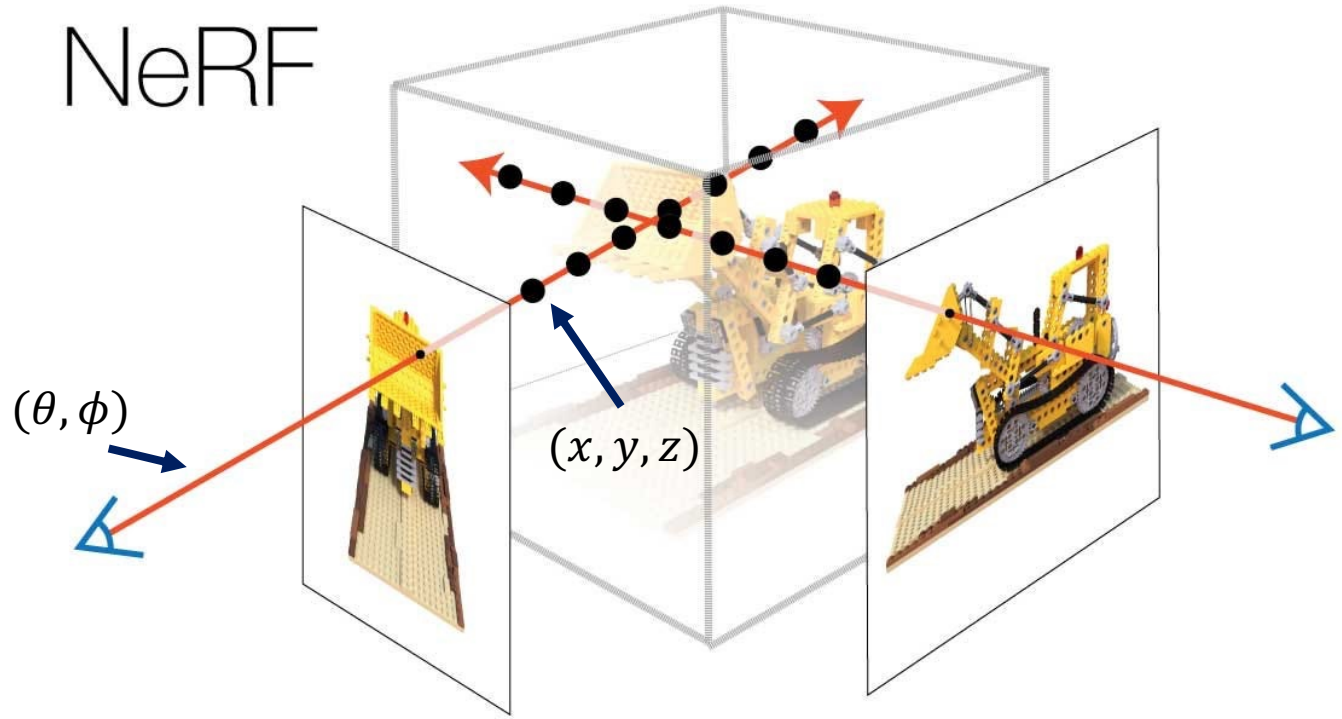


Input Images

Optimize NeRF

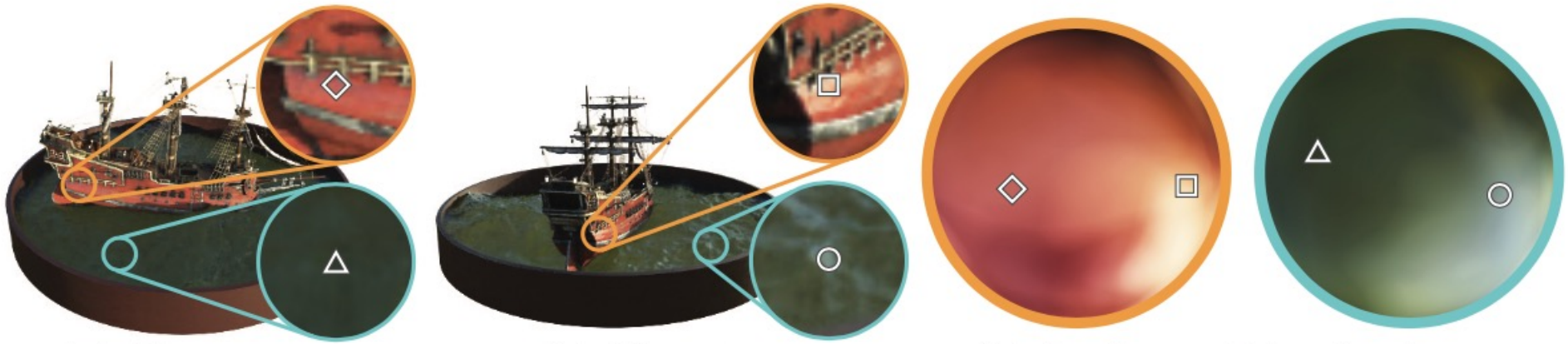
Render new views

2. Neural Radiance Field Scene Representation



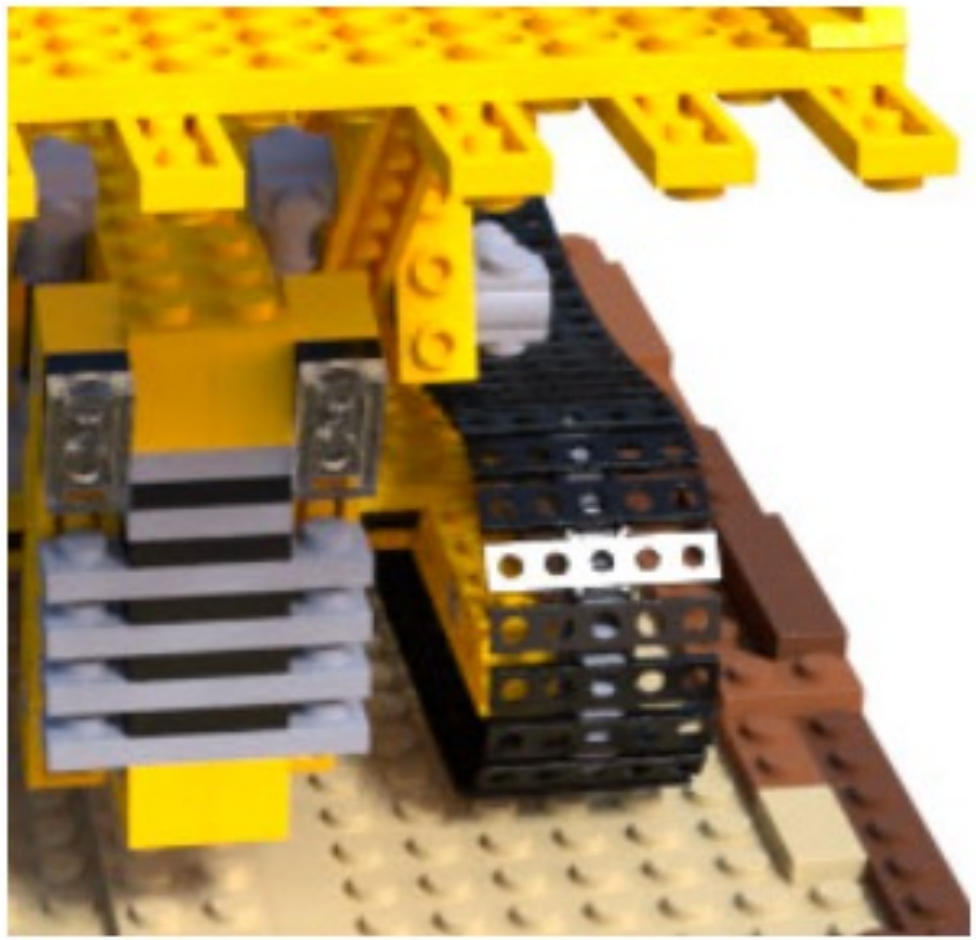
- A 3D location $\mathbf{x} = (x,\ y,\ z)$
- A 2D viewing direction (θ, ϕ)

The purpouse of view direction

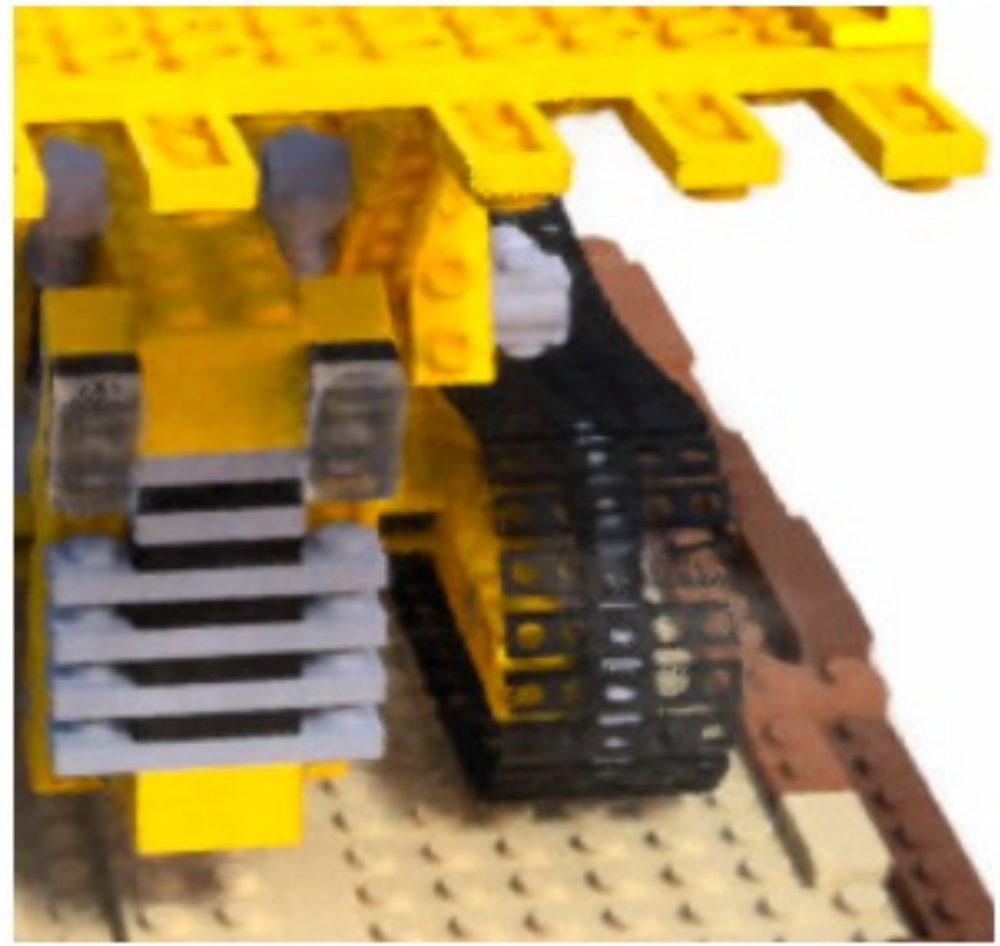


(a) View 1 (b) View 2 (c) Radiance Distributions

- (a) and (b) are the appearance of two fixed 3D points from two different camera positions
- The method predicts the changing specular appearance of the points (c)

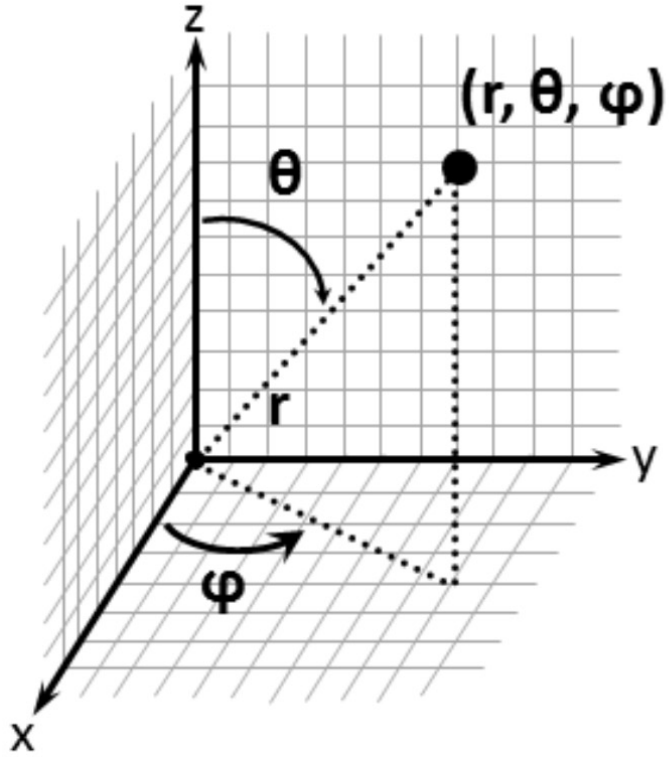


Ground Truth



No View Dependence

Viewing Direction

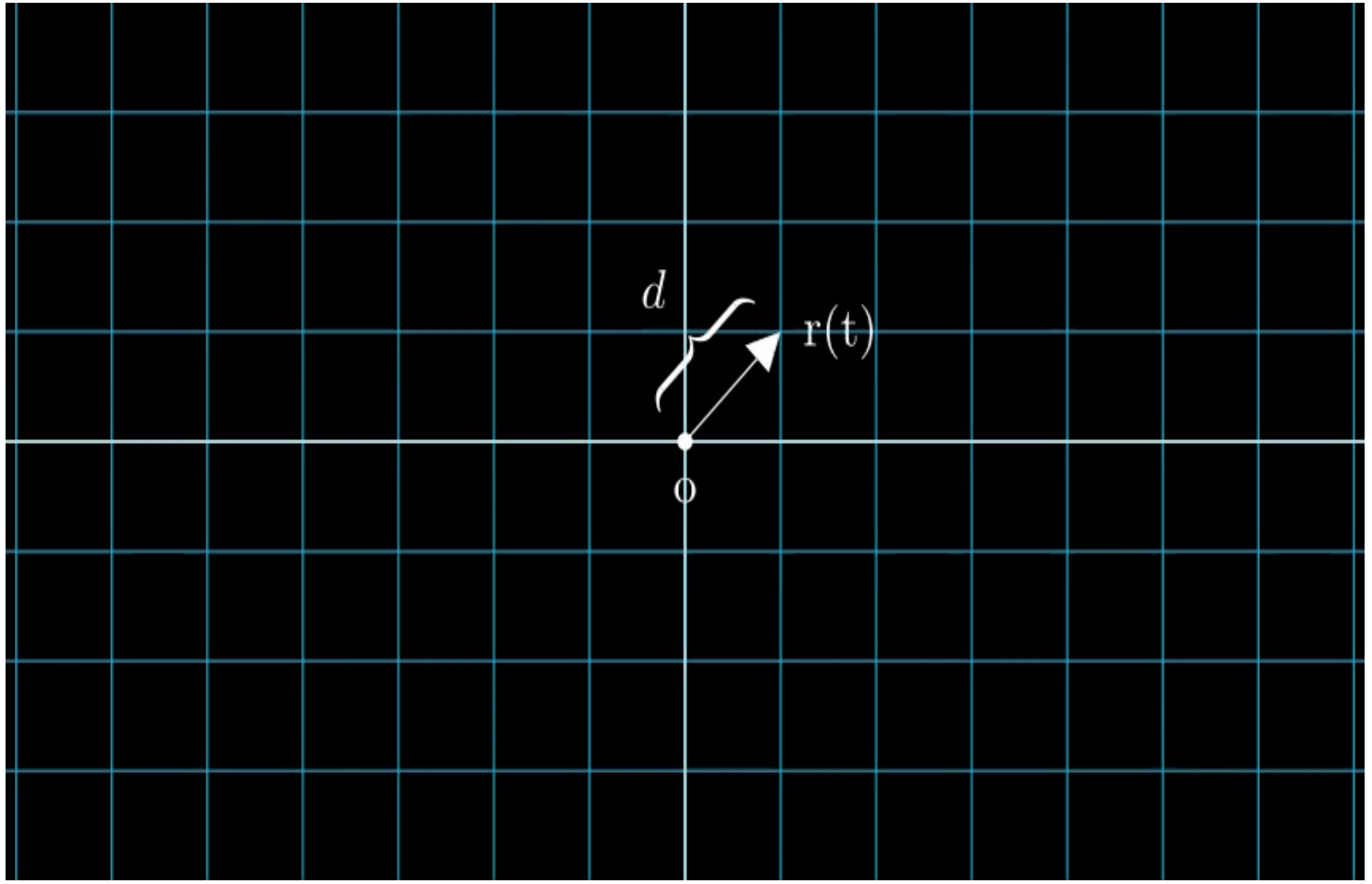


$$\mathbf{d} = (\hat{x}, \hat{y}, \hat{z})$$

$$\begin{cases} \hat{\boldsymbol{x}} = \cos(\phi) \cdot \sin(\theta) \\ \hat{\boldsymbol{y}} = \sin(\phi) \cdot \sin(\theta) \\ \hat{\boldsymbol{z}} = \cos(\theta) \end{cases}$$

In practice, viewing direction (θ, ϕ) is expressed as a 3D Cartesian unit vector $\mathbf{d}$

Viewing Ray

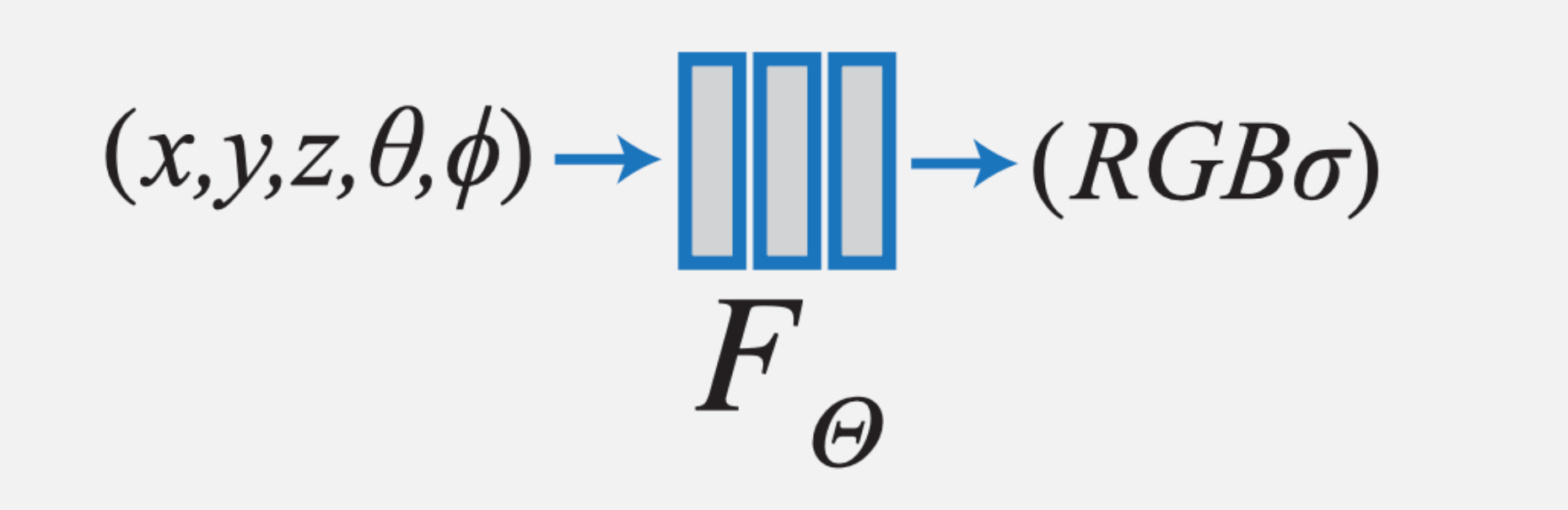


$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$

$\mathbf{o}$ – starting point

$\mathbf{d}$ – viewing direction

3. Network

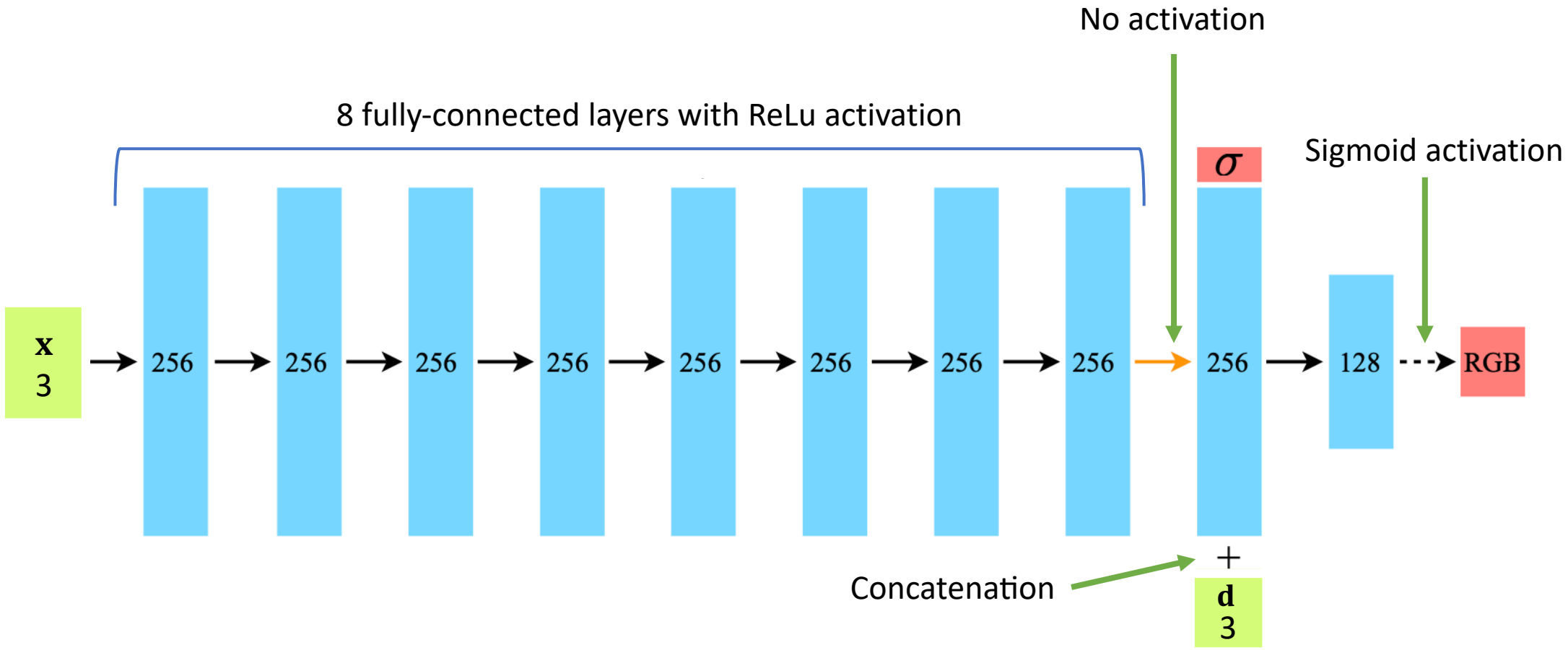


Network is fitted to predict points of a single scene!

F_θ - a multilayer perceptron or MLP

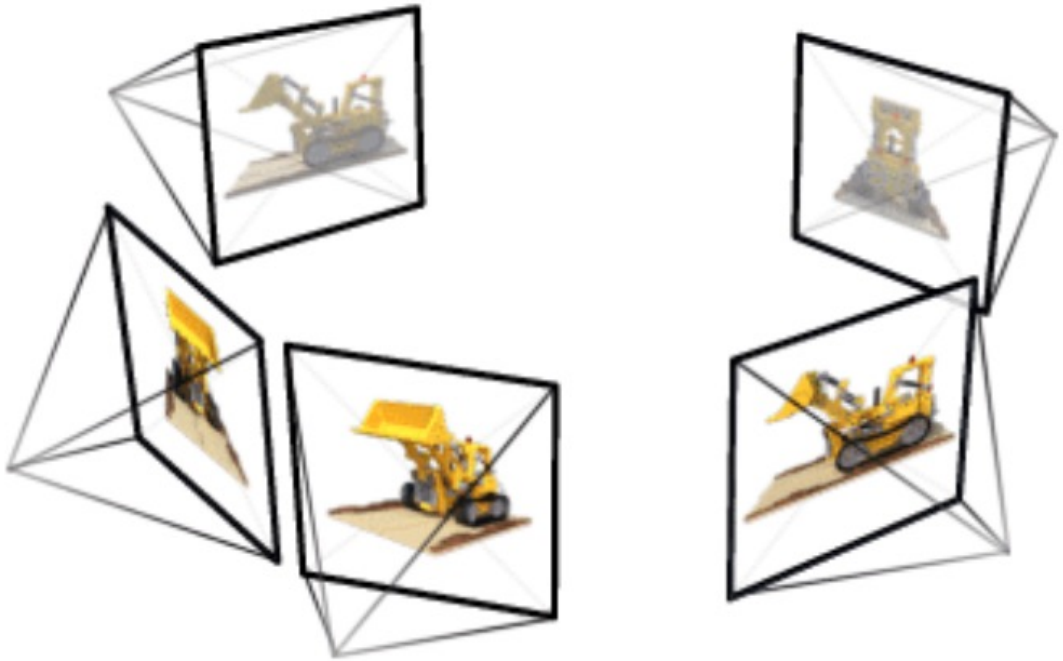
$\mathbf{c} = RGB$ - predicted color

σ – predicted volume density



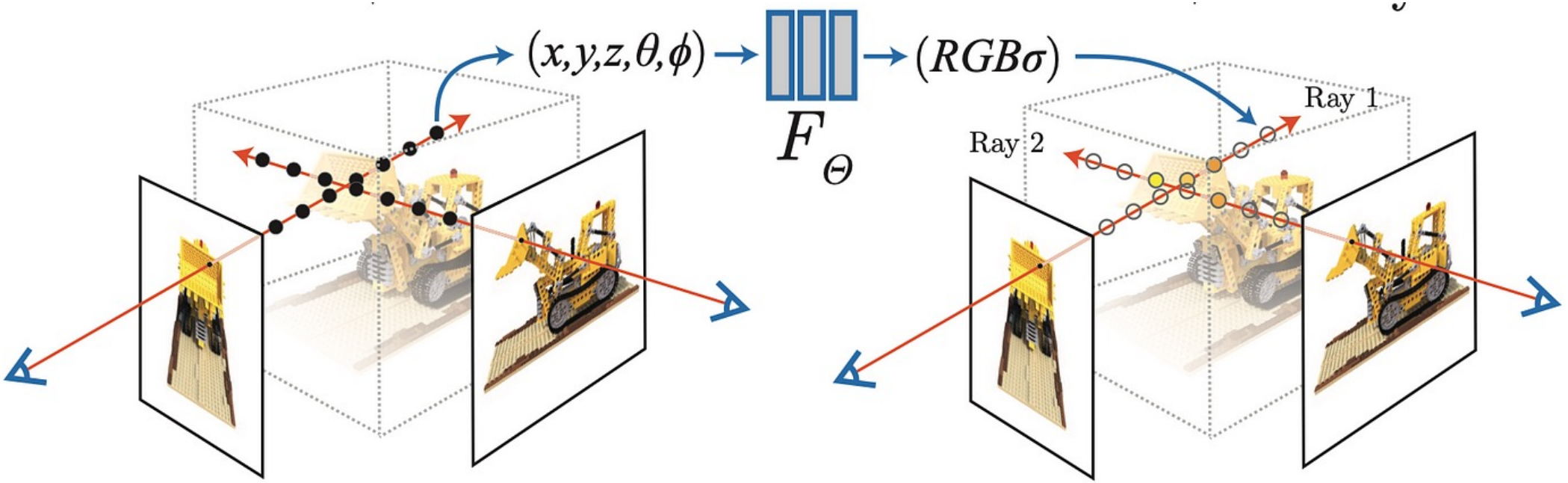
- σ is rectified using a ReLu to ensure that the output volume density is nonnegative
- σ is a function of only the location $\mathbf{x}$
- RGB is a function of both location $\mathbf{x}$ and viewing direction $\mathbf{d}$

Training Data

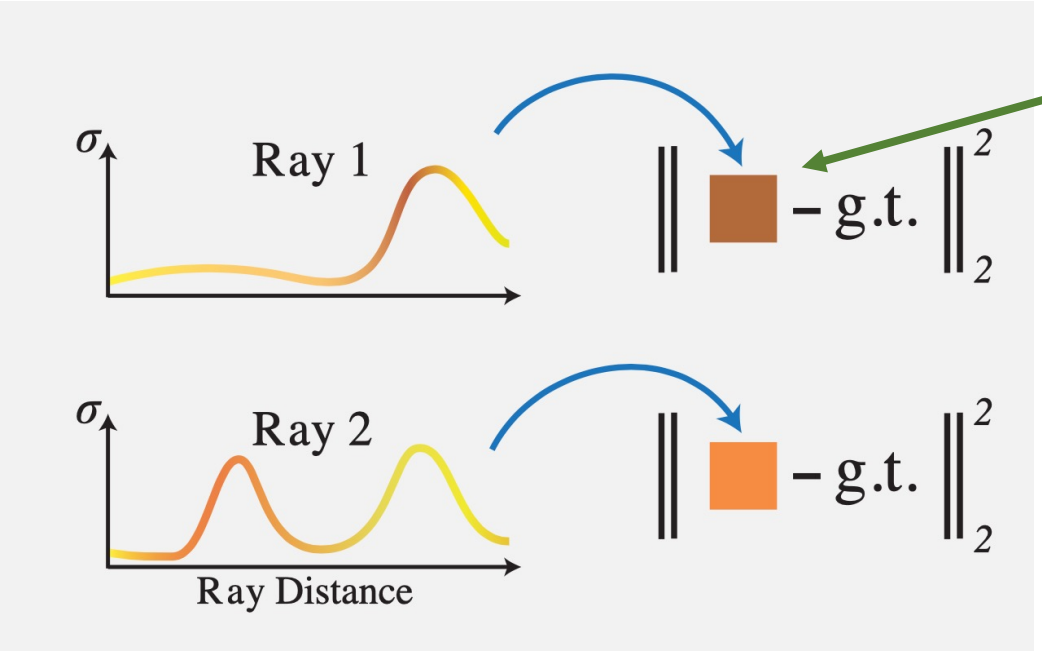


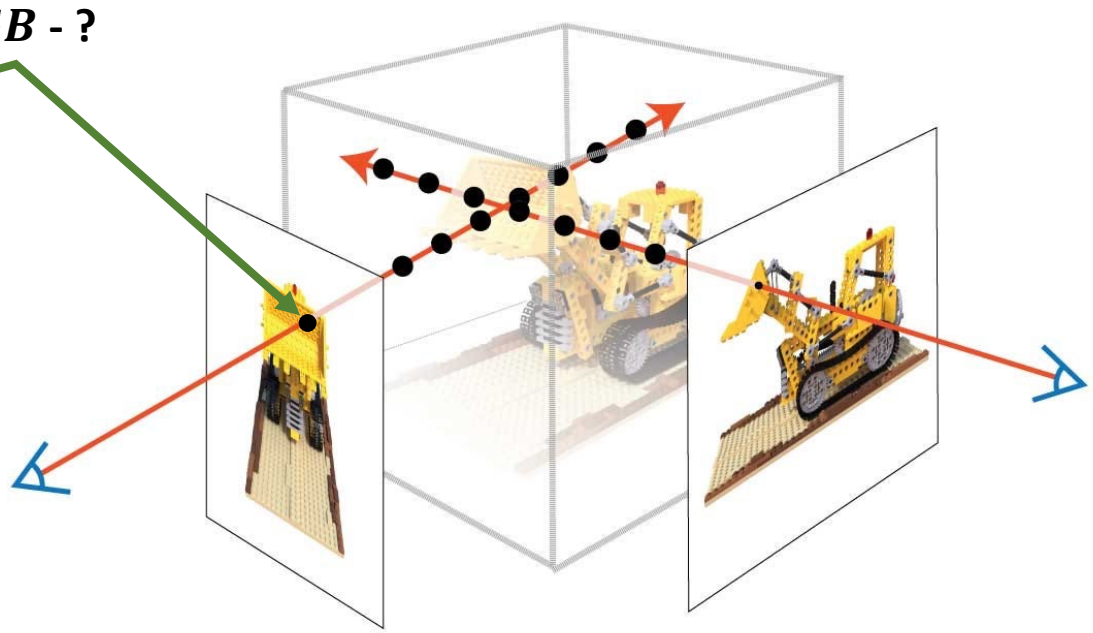
- Camera positions: (θ, ϕ)
- A ray can be shot through each pixel of a 2D image: $\mathbf{r}(t)$
- 3D location coordinates can be sampled on it: $(x,\ y,\ z)$
- Target - ?

4. Volume Rendering and Expected Color



Volume Density





How to get the exact color that we see from the given direction on a 2D image?

Volume Rendering[1]

2D color prediction

Density (color weight)

Probability that the ray travels from t_n to t without hitting any other particle

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt\,, \text{ where } T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right)$$

Color prediction in a 3D location

$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ – ray

$\mathbf{d}$ – viewing direction

t_n and t_f - near and far bounds

1. Kajiya, J.T., Herzen, B.P.V.: Ray tracing volume densities. Computer Graphics (SIGGRAPH) (1984)

Numericall estimation

1. Partition $[t_n, t_f]$ into N evenly-spaced bins
2. Draw one sample uniformly at random from each bin

$$t_i \sim \mathcal{U}\left[t_n + \frac{i-1}{N}(t_f - t_n),\ t_n + \frac{i}{N}(t_f - t_n)\right]$$

3. Estimate $C(\mathbf{r})$ with quadrature rule[2]

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))\mathbf{c}_i\,,\ \text{where } T_i = \exp\left(-\sum_{j=1}^{i-1}\sigma_j\delta_j\right)$$

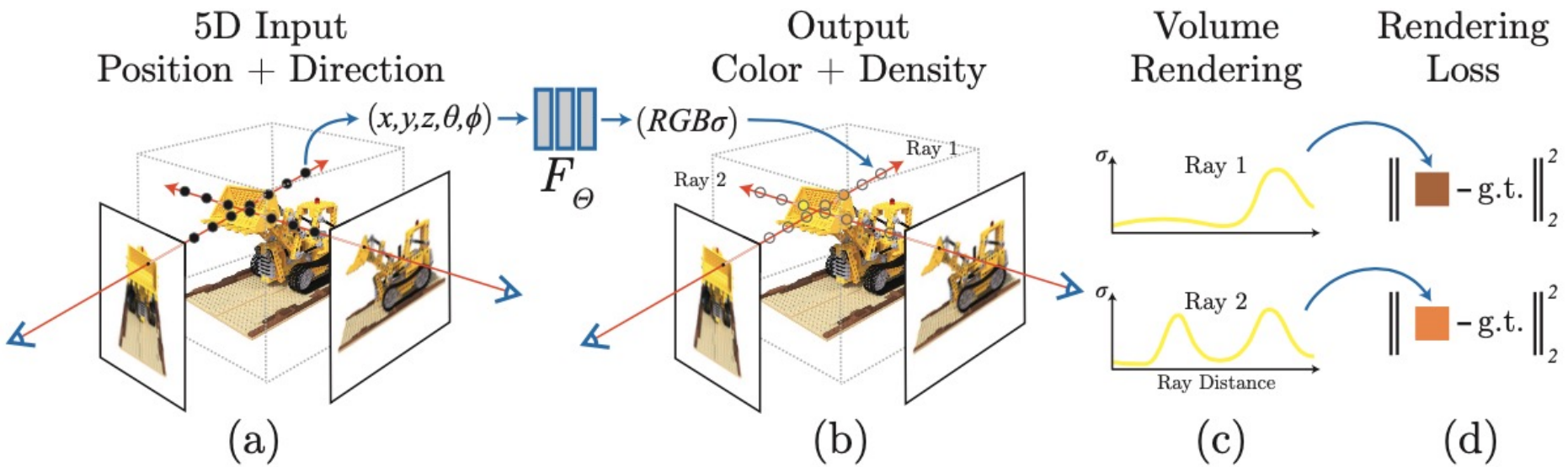
where $\delta_i = t_{i+1} - t_i$ - distance between adjecent samples.

The function is differentiable by (c_i, σ_i) and can be easily used with our loss.

2. Max, N.: Optical models for direct volume rendering. IEEE Transactions on Visualization and Computer Graphics (1995)

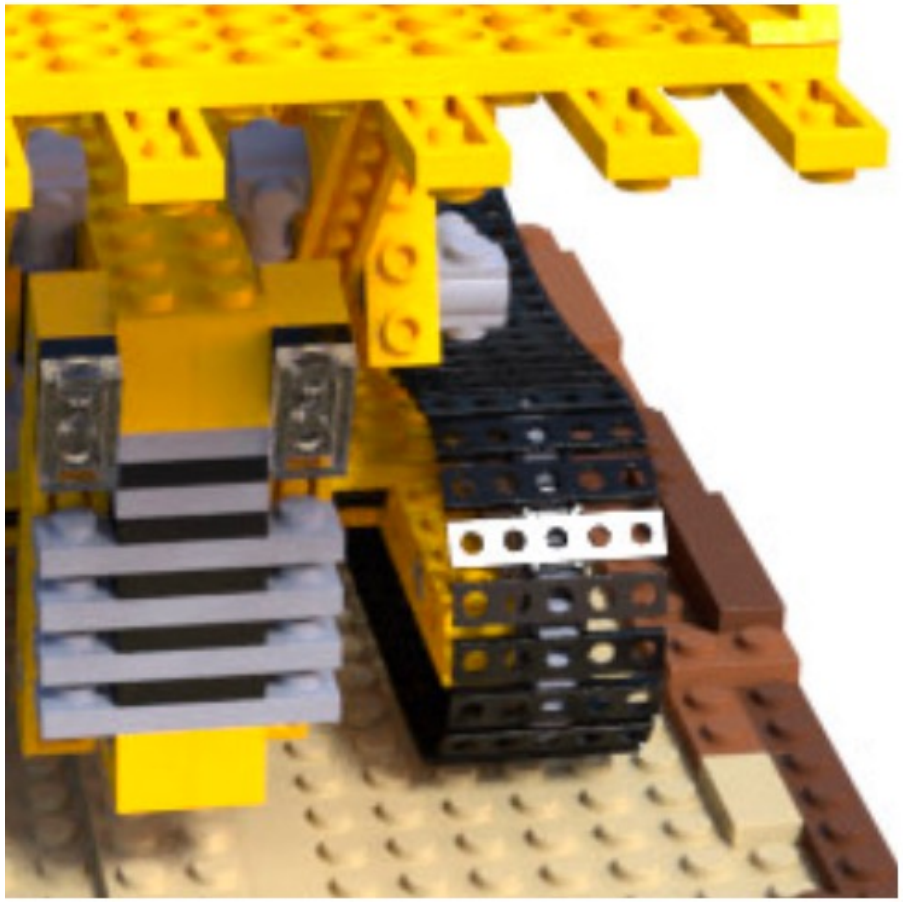
Loss function

$$\mathcal{L} = \sum_{\mathbf{r} \in \mathcal{R}} \left[\left\| \hat{C}(\mathbf{r}) - \text{g. t.} \right\|_2^2 \right]$$



1. Camera positions: (θ, ϕ)
2. Shot a ray through each pixel of a 2D image: $\mathbf{r}(t) \in \mathcal{R}$
3. Sample $(x,\ y,\ z)$ on a ray uniformly at random in each of N bins
4. Predict $(RGB\sigma)$ at each sampled 3D coordinate
5. Count expected color $\hat{C}(\mathbf{r})$ for each ray
6. Count quadratic loss $\mathcal{L}$ from the ground truth color
7. Optimize the network

5. Intermediate Results



Ground Truth

No Positional Encoding

6. Optimizations

1. Postional Encoding
2. Hierarchical volume sampling

6.1. Positional Encoding

Issue: current model is unable to represent high-frequency changes in the scene

Solution[3]: map the inputs to a higher dimesional space using high-frequency functions

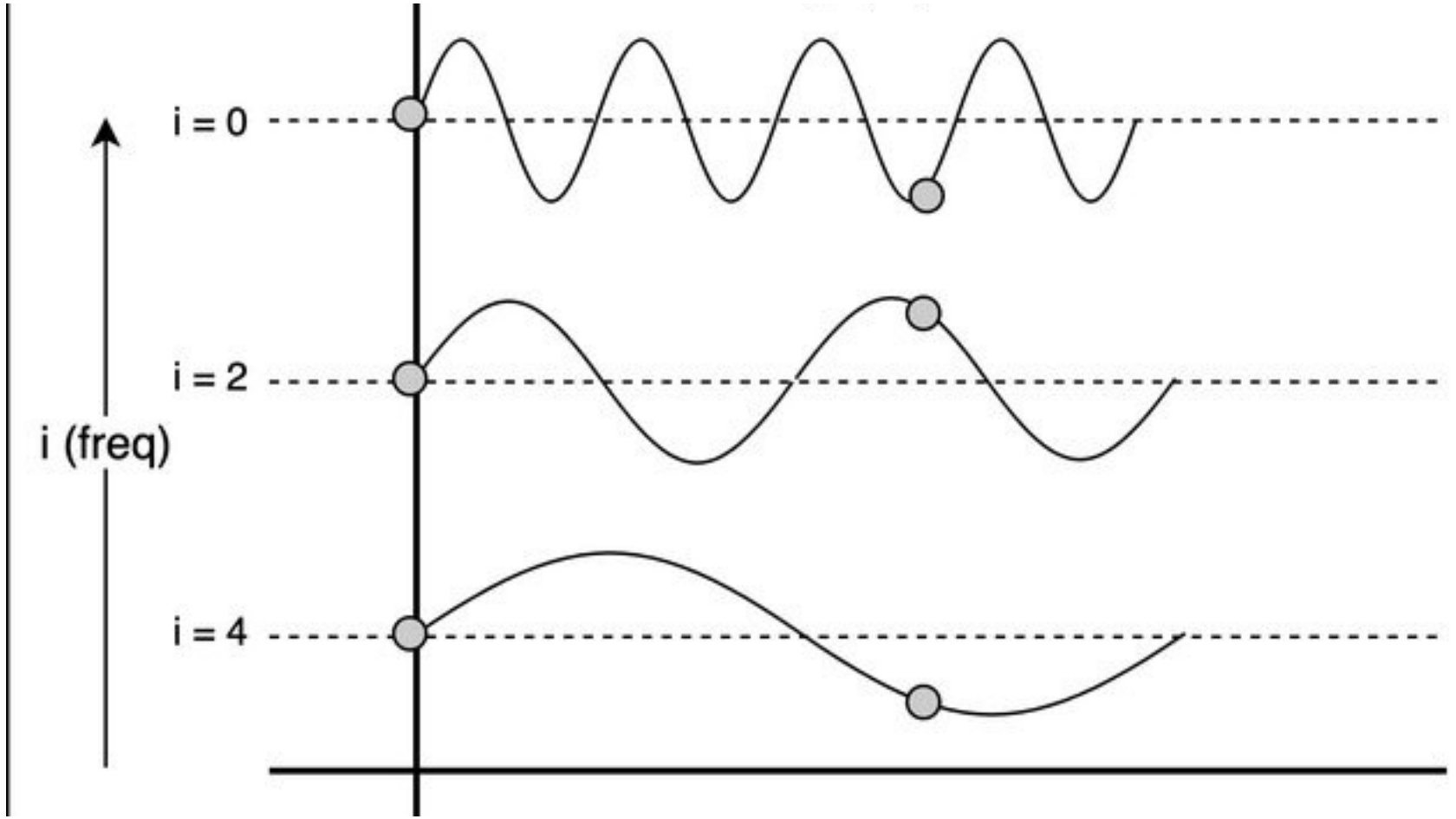
3. Rahaman, N., Baratin, A., Arpit, D., Dra¨xler, F., Lin, M., Hamprecht, F.A., Bengio, Y., Courville, A.C.: On the spectral bias of neural networks. In: ICML (2018)

Solution

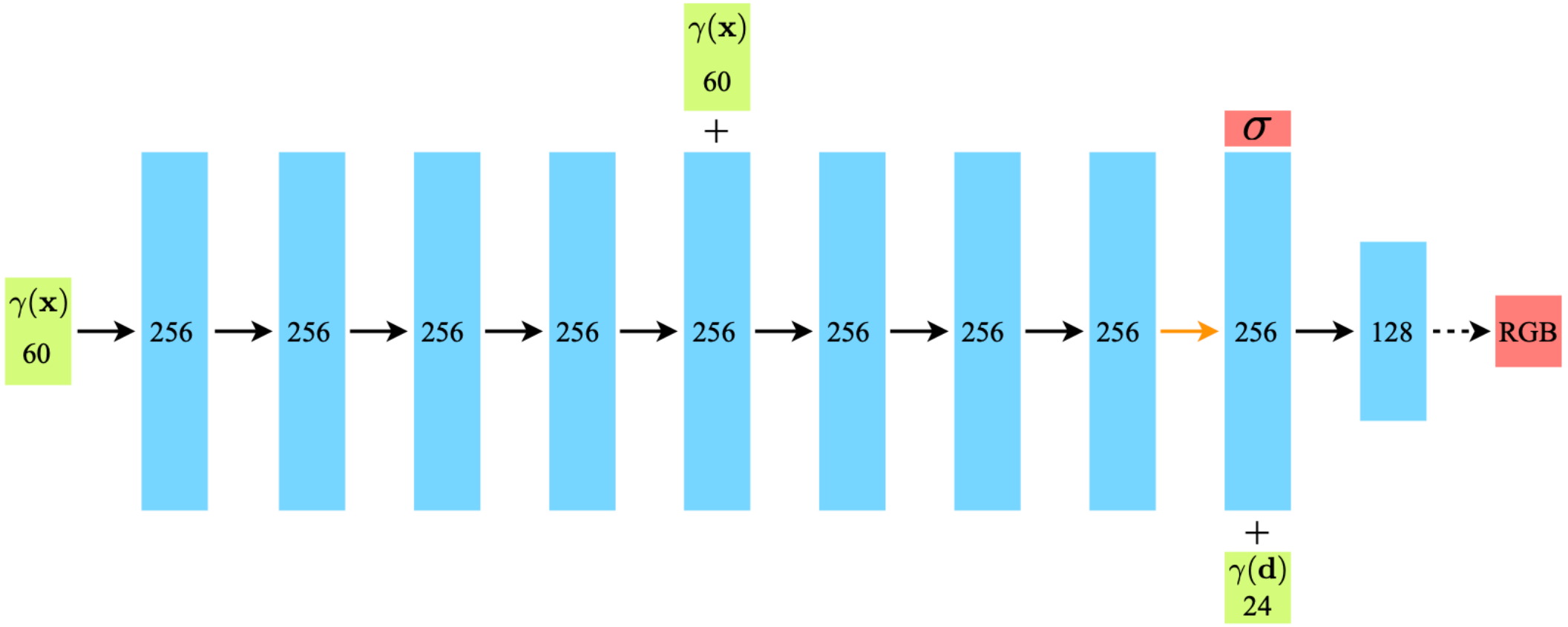
$$\gamma(p) = \left(\sin\left(2^0\pi p\right), \cos\left(2^0\pi p\right), \cdots, \sin\left(2^{L-1}\pi p\right), \cos\left(2^{L-1}\pi p\right)\right)$$

1. Normalize $(x,\ y,\ z)$ to lie in [-1, 1]
2. Apply $\gamma(\cdot)$ to each coordinate of $\mathbf{x}$ and $\mathbf{d}$ separately
3. In experiments $L = 10$ for $\mathbf{x}$ and $L = 4$ for $\mathbf{d}$ are used

Example



New Architecture



6.1. Hierarchical volume sampling

Issue: current sampling of points is inefficient: free space and occluded regions do not contribute to the image but are still sampled repeatedly

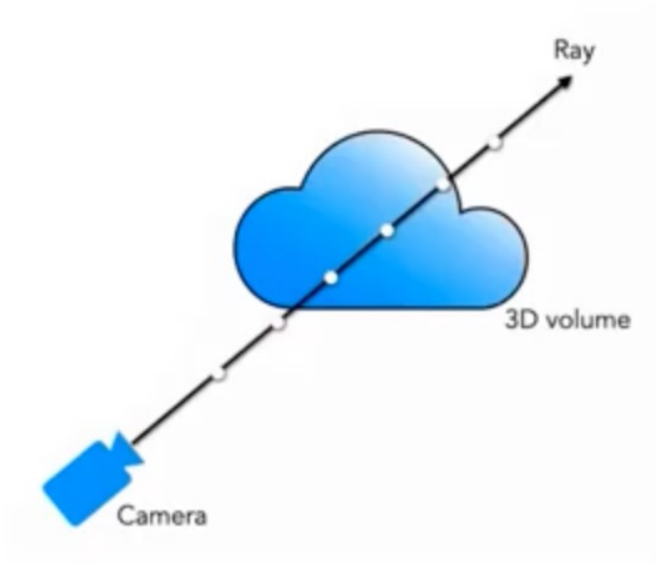
Solution: optimize two networks: “coarse” and “fine”, so that points for the second one are sampled using the output of the “coarse” model

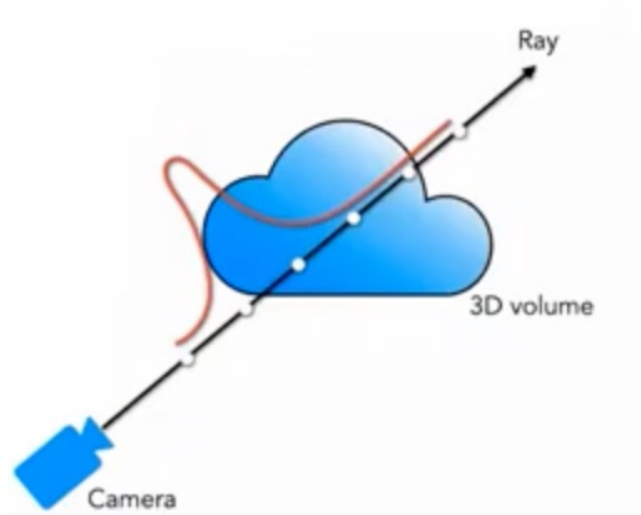
Solution

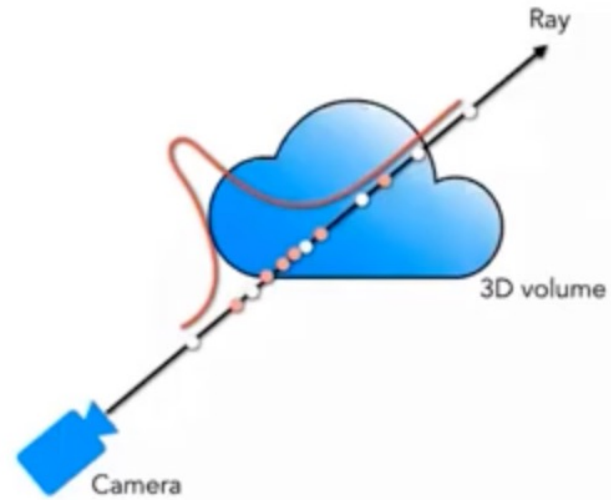
1. Sample a set of N_c locations using stratified sampling, and evaluate the "coarse" network as before:

$$\hat{C}_c(\mathbf{r}) = \sum_{i=1}^{N_c} w_i c_i \,, \qquad w_i = T_i(1 - \exp(-\sigma_i \delta_i))$$

2. Normalize the weights as $\widehat{w}_i = {}^{w_i}\!/_{\sum_{j=1}^{N_c} w_j}$ - a piecewise-constant PDF along the ray
3. Sample set of N_f locations from this distribution, and evaluate the "fine" network using the union of the two sets: $\hat{C}_f(\mathbf{r})$ is counted on $N_c + N_f$ points

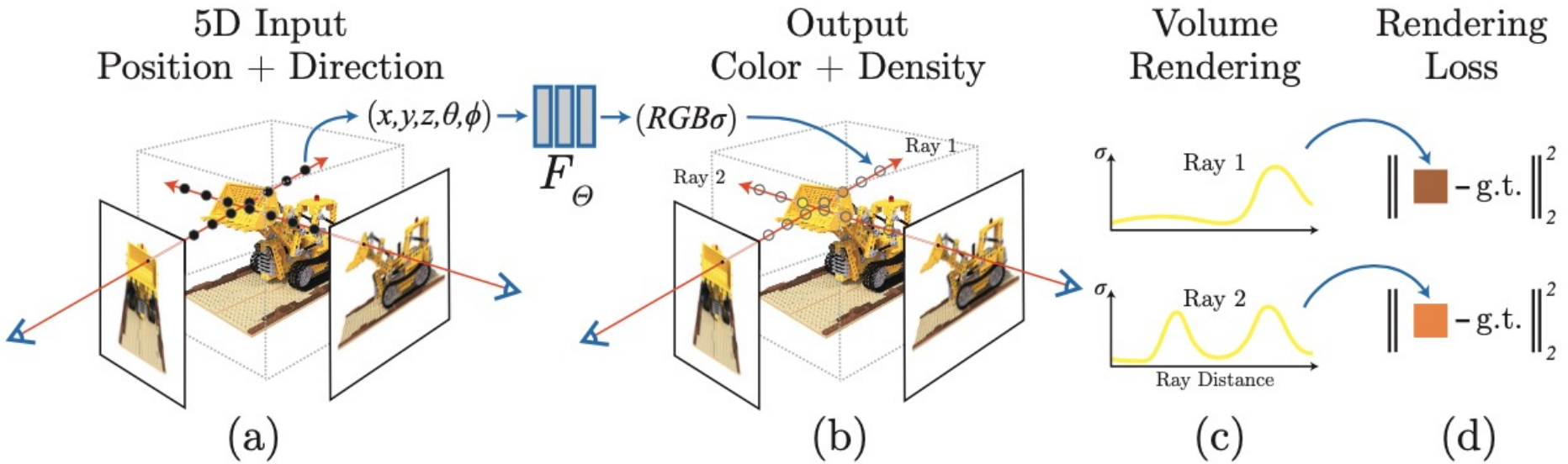

Ray
3D volume
Camera


Ray
3D volume
Camera


Ray
3D volume
Camera

New Loss Function

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



1. Camera positions: (θ, ϕ)
2. Shot a ray through each pixel of a 2D image: $\mathbf{r}(t) \in \mathcal{R}$
3. Sample N_c locations $(x,\ y,\ z)$ on a ray uniformly at random in each of N bins
4. Normalize and apply $\gamma(\cdot)$ on the parameters
5. Evaluate “coarse” model and count expected color $\hat{C}_c(\mathbf{r})$ for each ray
6. Normalize the weights as $\hat{w}_i = {}^{w_i}/_{\sum_{j=1}^{N_c} w_j}$ and sample N_f locations from this PDF
7. Evaluate “fine” model with two sets of points
8. Optimize both networks with the total loss

7. Hyperparameters

- Batch size – **4096** rays
- Samples: $N_c = 64,\ N_f = 128$
- Learning rate: exponential from $\mathbf{lr_0} = 5 \cdot 10^{-4}$ to $\mathbf{lr} = 5 \cdot 10^{-5}$
- Optimizer: **Adam** ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-7}$)
- GPU: NVIDIA V100

8. Results

2 types of datasets: **synthetic renderings of objects** and **real images of complex scenes**

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

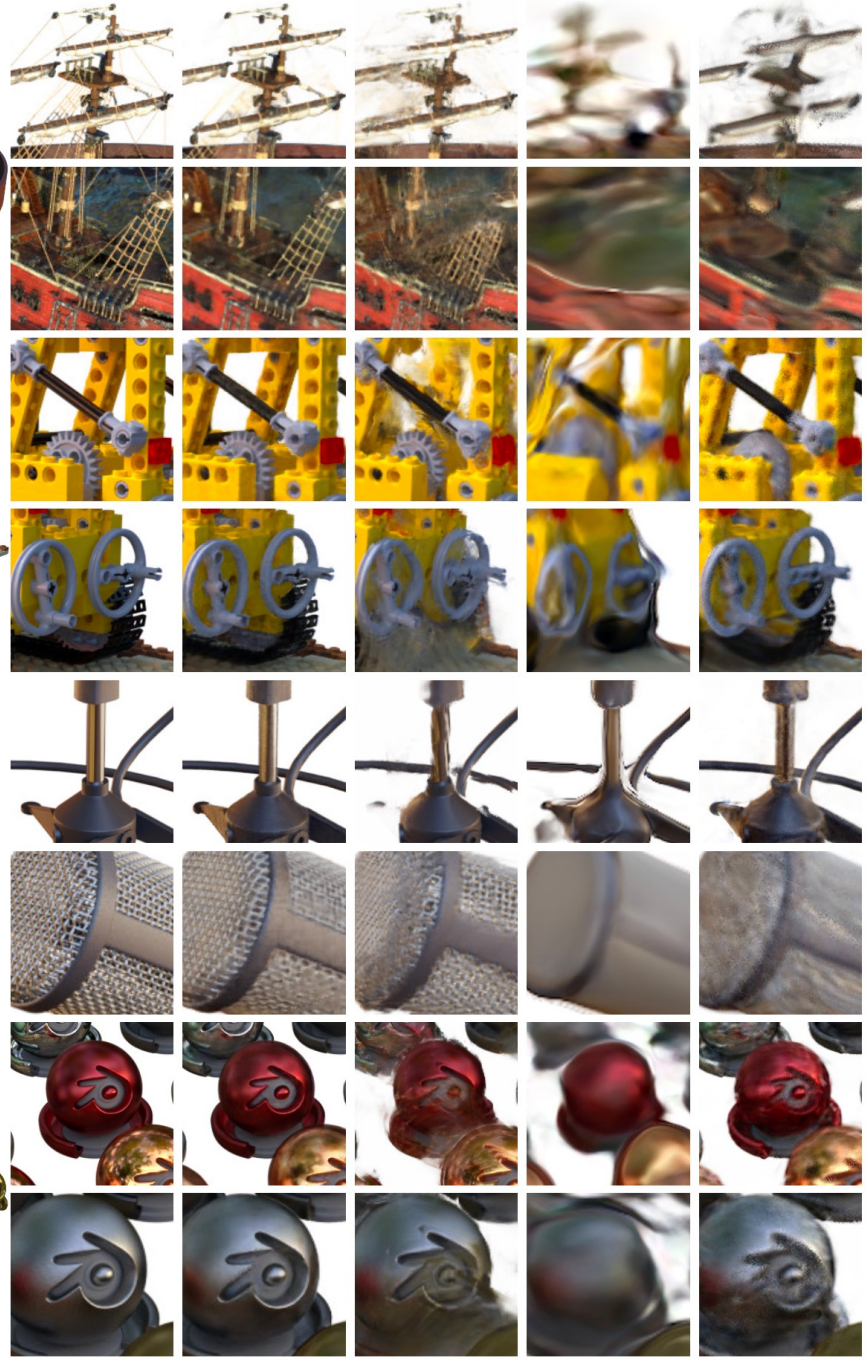
- ***Neural Volumes (NV)*** - deep 3D convolutional network
- ***Scene Representation Networks (SRN)*** – recurrent neural network
- ***Local Light Field Fusion (LLFF)*** - designed for producing photorealistic novel views for well-sampled forward facing scenes.

Ship

Lego

Microphone

Materials

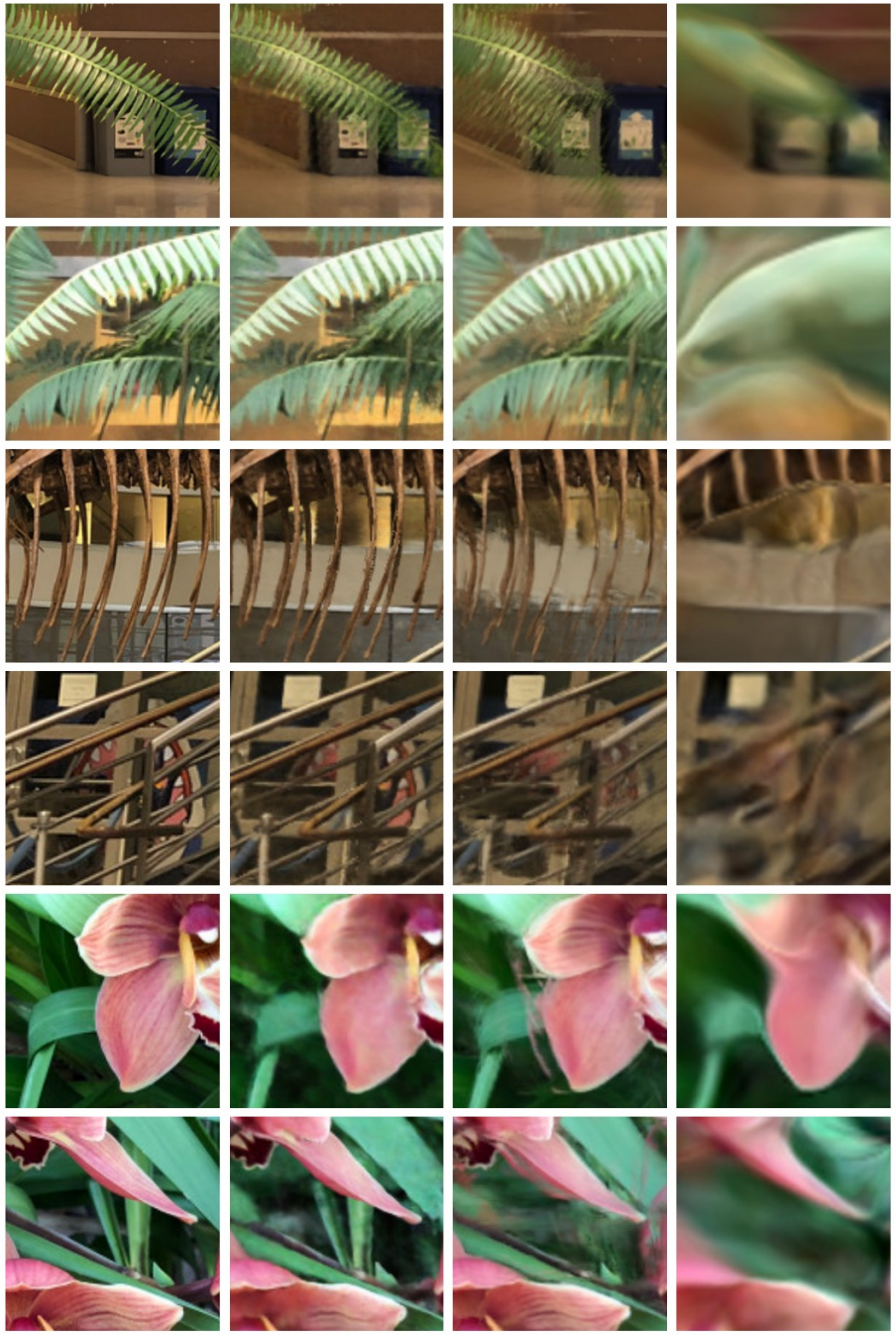


Ground Truth NeRF (ours) LLFF [28] SRN [42] NV [24]

Fern

T-Rex

Orchid



Ground Truth NeRF (ours) LLFF [28] SRN [42]

Comparision of optimizations

	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**

9. Conclusion

Benefits

- State-of-the-Art model
- Generates high-resolution images
- Learned on just 25 images still outperfoms other method learned on 100 images

Drawbacks

- Takes 100-300k iterations to converge on a single NVIDIA V100 GPU (1-2 days)
- Has to be trained for every single scene