UofT Computer Vision Club

Paper Breakdown Series

NeRFs: Neural Radiance Fields

youtube.com/@uoftcv

github.com/ut-cvdg/paper-breakdown

Follow us on social media

UofT CV is sponsored by GitHub

Sign up for the GitHub Student Developer Pack:
https://education.github.com/benefits

UNIVERSITY OF TORONTO | CVDG

About the Club

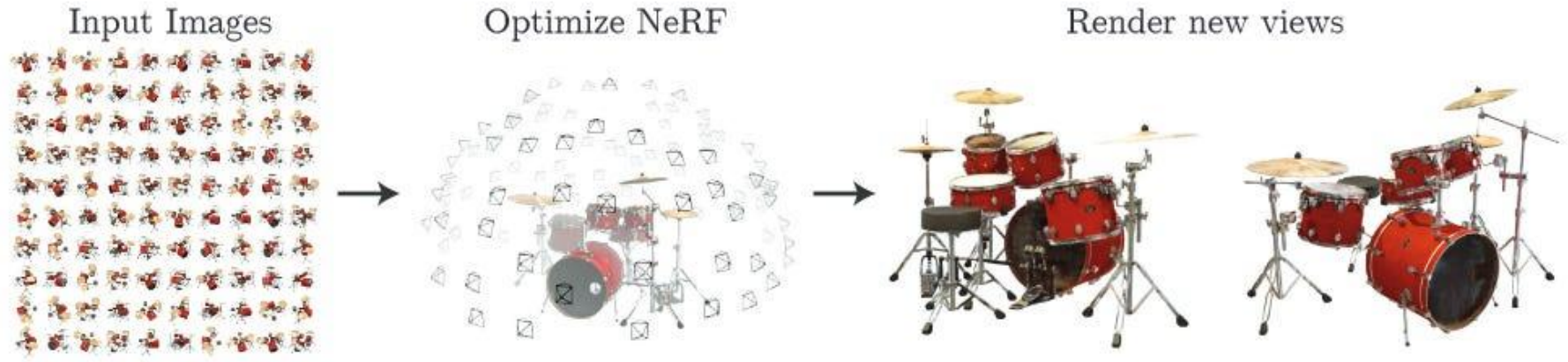
- We are a student club for Computer Vision and Machine Learning affiliated with the University of Toronto.
- We run:
  - paper “discussions” for paper in the past 1 week
  - conference-style research orals for pre-prints
  - Now, paper breakdown series

If you want to help run our club, talk to any of us or drop an email contact@cvdg.tech.

Vedant

1. Explain problem - show examples as well
2. intuitive recap of previous attempts to solve this
3. what is a neural net - borrow from trace 5d slides
4. what is the 5D input
5. given some images, how do they convert them to these 5d representations
6. volume rendering - eq 1, 2, 3

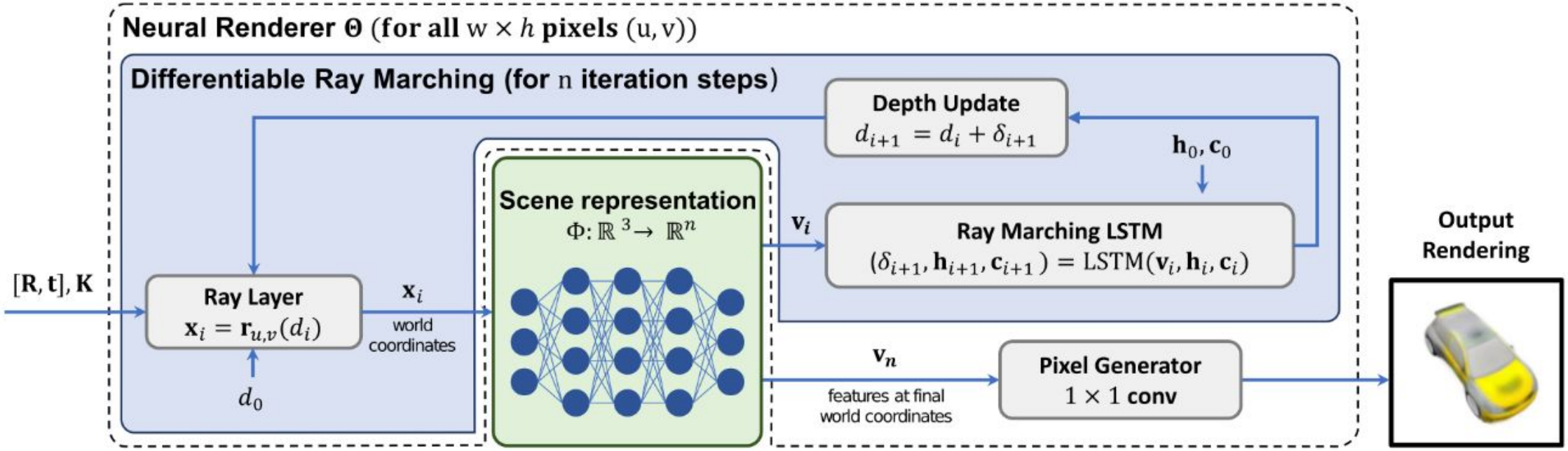
Problem Statement



UNIVERSITY OF TORONTO

Previous Solutions - SRN

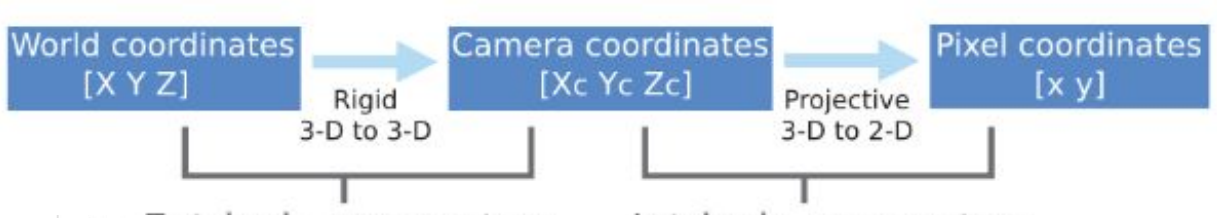
- Take the Implicit and Explicit Camera Matrices as Input.
- The output is a 2D RGB Image which denotes the Scene from a specified viewpoint.
- NERF vs SRNs

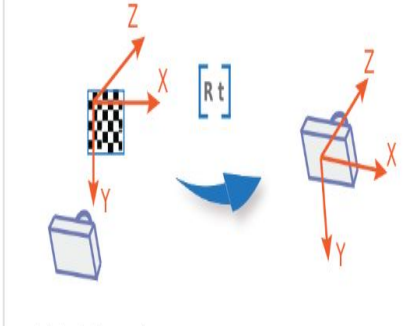


Previous Solutions - SRN

Camera Calibration Parameters

The calibration algorithm calculates the camera matrix using the extrinsic and coordinate system to the 3-D camera's coordinate system. The intrinsic param coordinates.





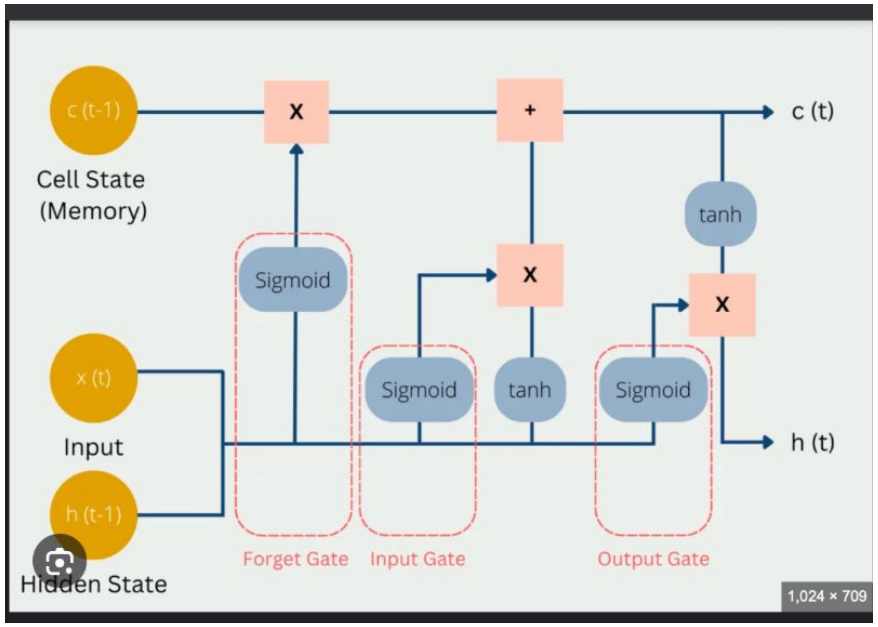
Intrinsic Parameters

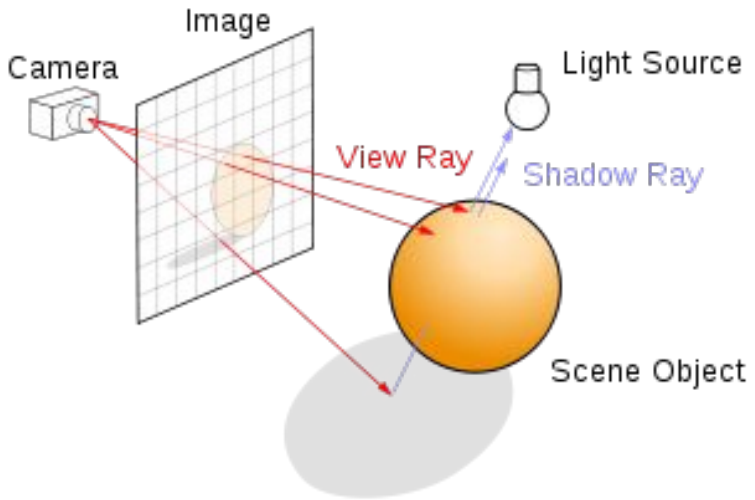
The intrinsic parameters include the focal length, the optical center, also known as the *principal point*, and the skew coeffic

$$\begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Previous Solutions - SRN

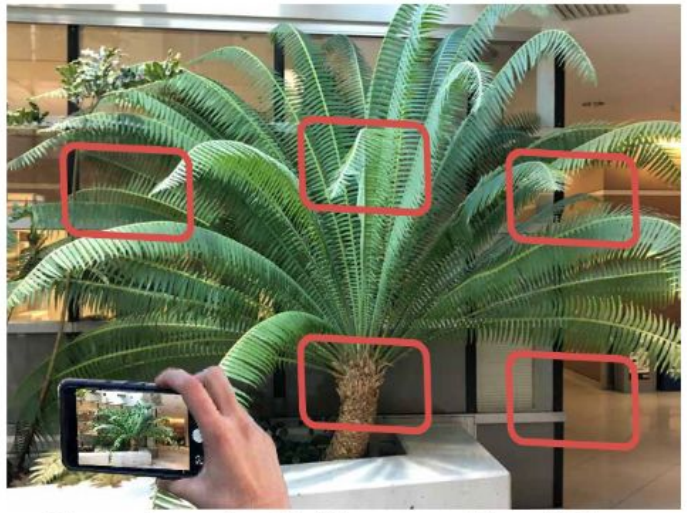
- SRNs employ use of Ray Marching Algorithms



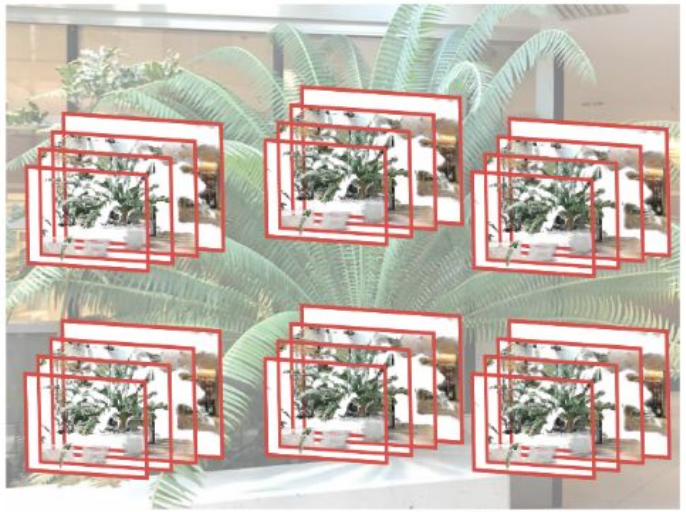


They use the LSTM model architecture in order for Ray Marching ALgorithms.

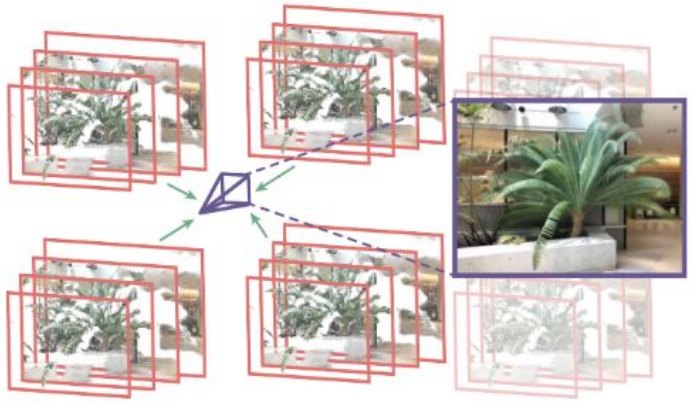
Previous Solutions - LLFF



Fast and easy handheld capture with guideline:
closest object moves at most D pixels between views

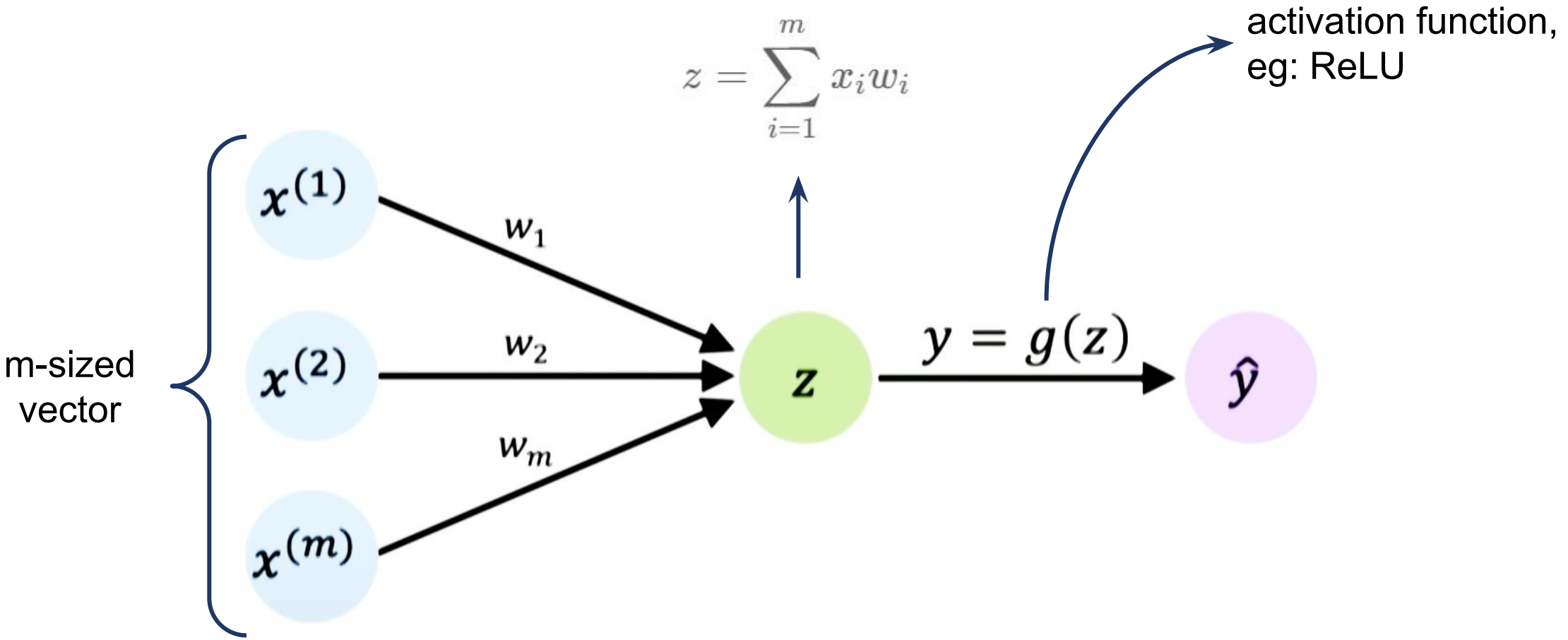


Promote sampled views to local light field
via layered scene representation



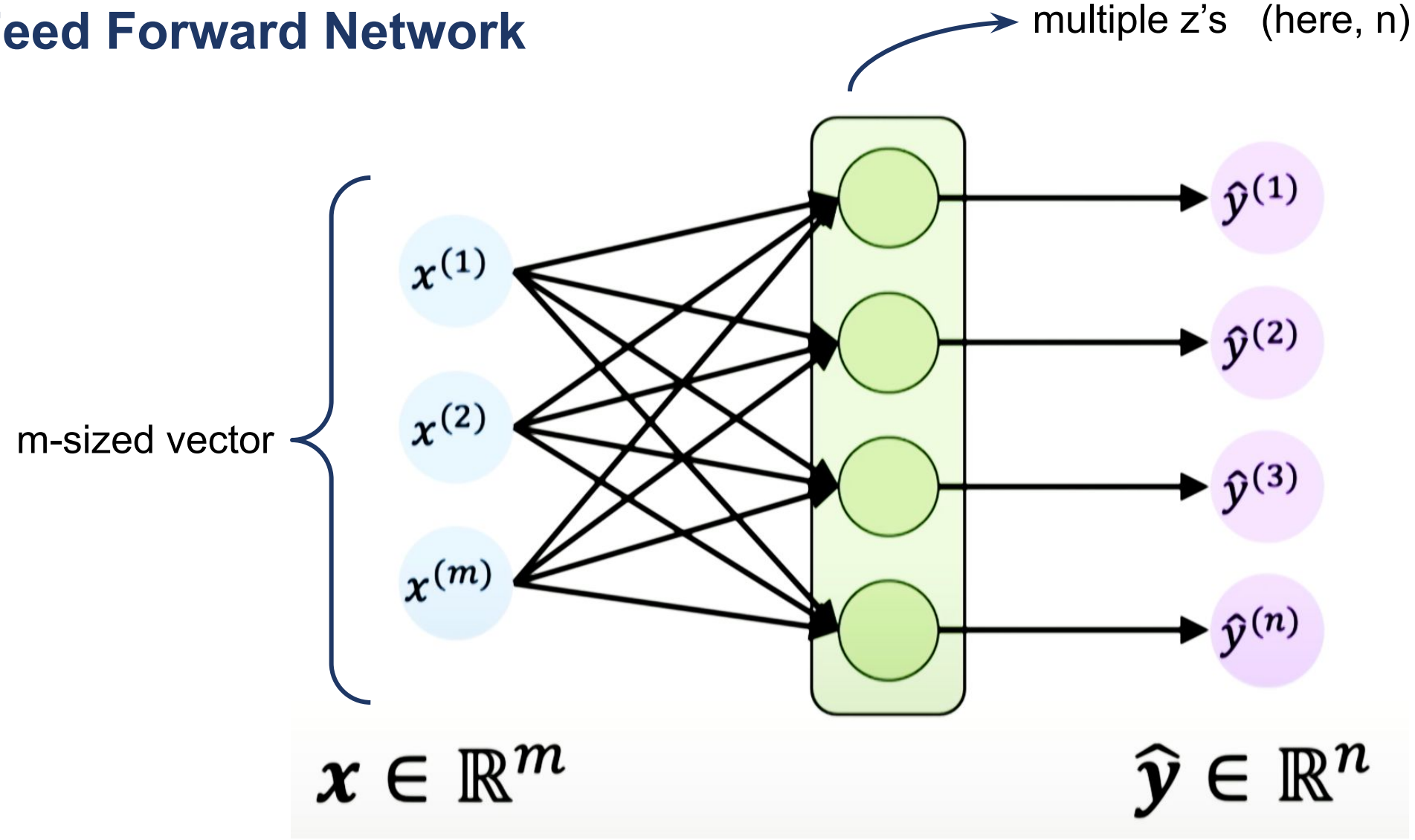
Blend neighboring local light fields
to render novel views

Perceptron



Source: MIT 6.S191

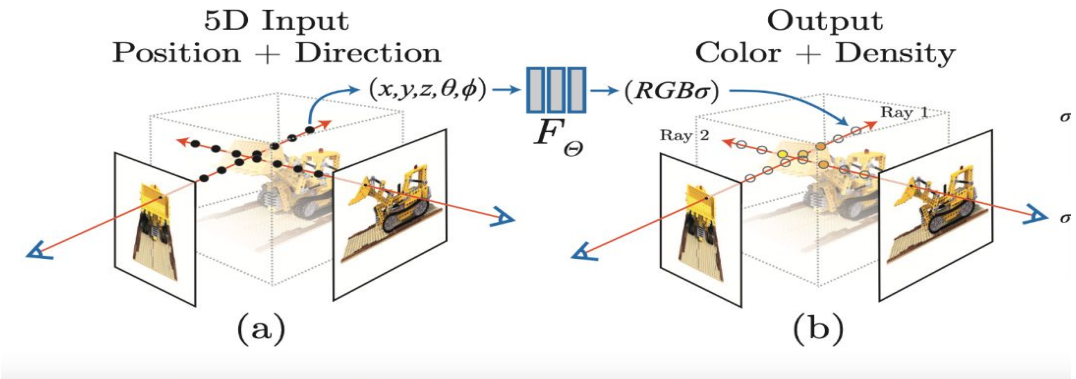
Feed Forward Network



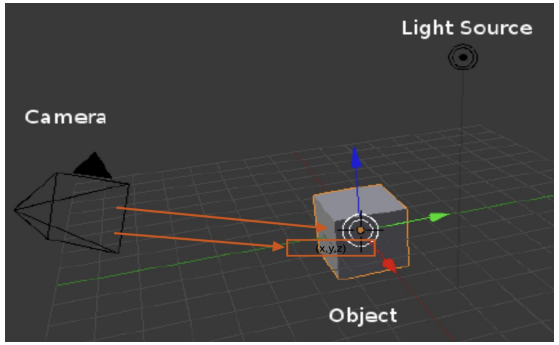
Source: MIT 6.S191

NERFs

>It is a function which takes 5 coordinates and returns the density and RGB value at the point

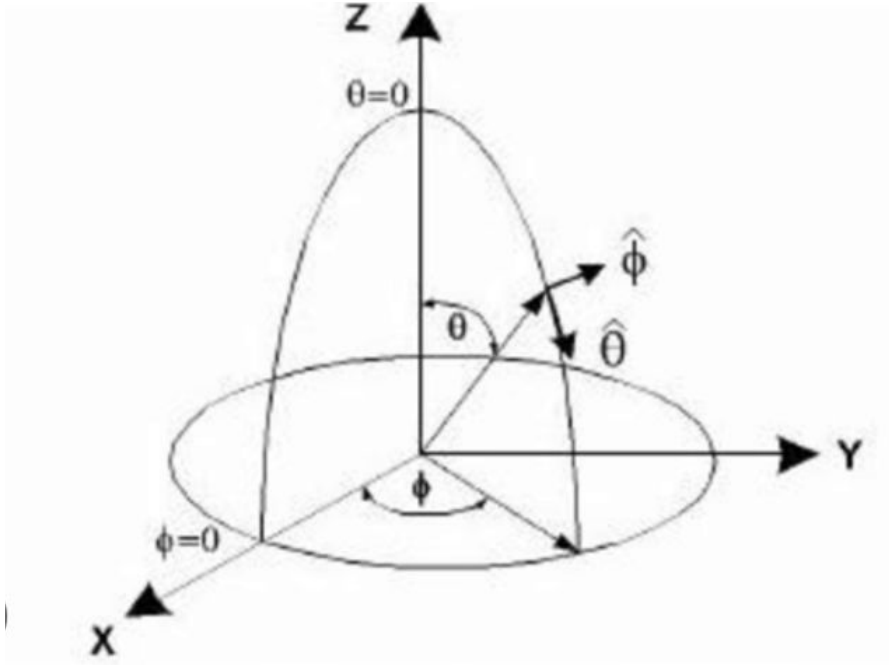


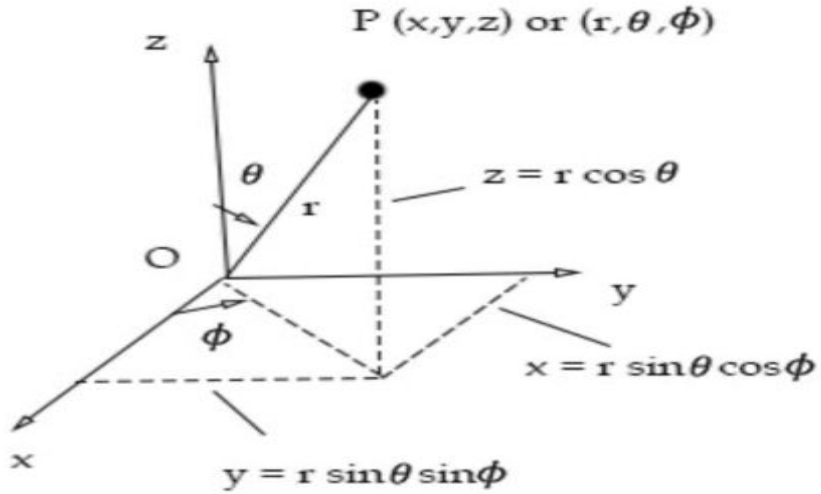
>The 3d coordinates of the point are one input.



UNIVERSITY OF TORONTO

>The other input is the view direction of the camera angles

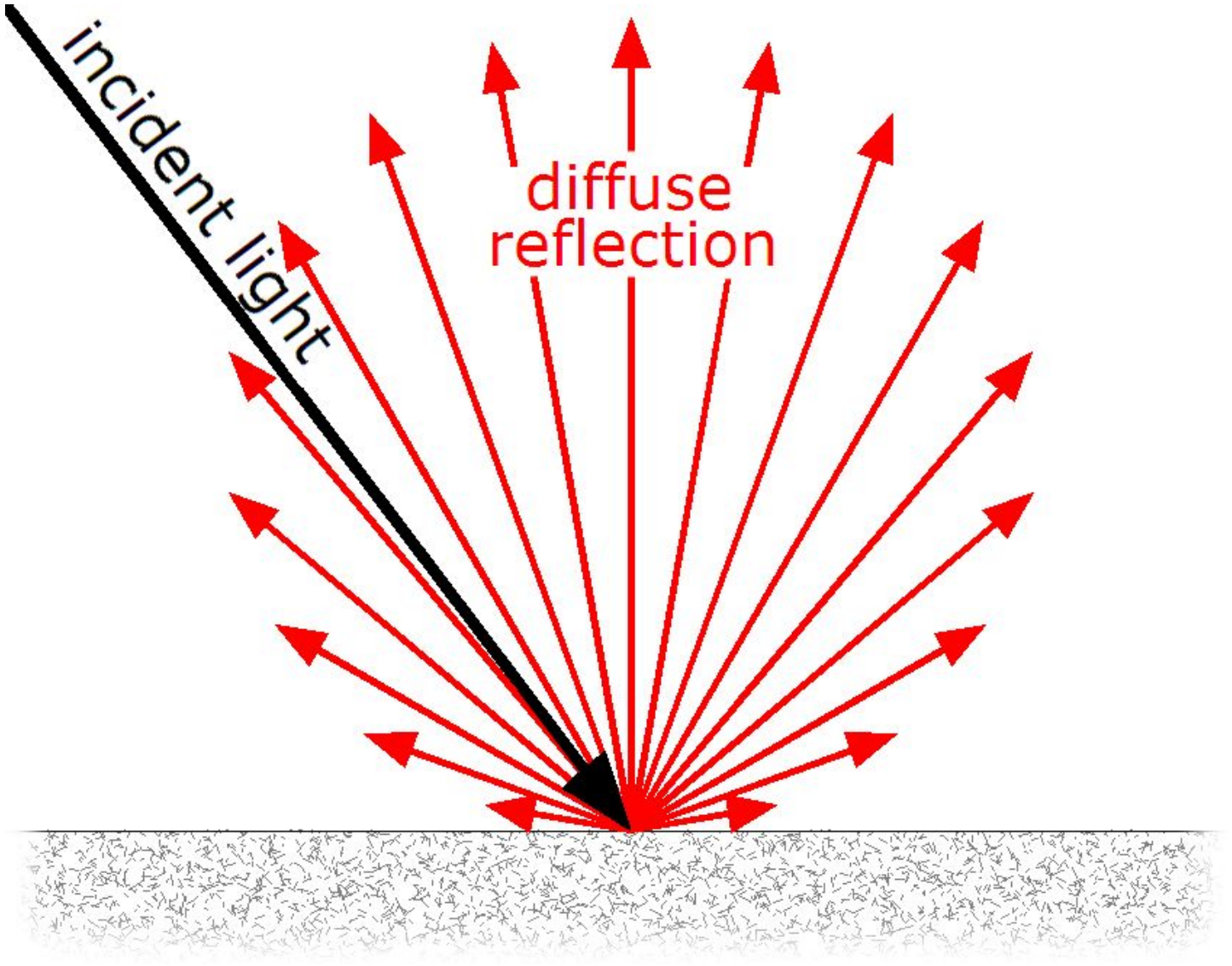




$$\text{cart} \leftrightarrow \text{sph} \quad \begin{cases} x = \rho \cos\theta \sin\phi, \\ y = \rho \sin\theta \sin\phi, \\ z = \rho \cos\phi, \end{cases}$$

$$\begin{cases} \rho = \sqrt{x^2 + y^2 + z^2}, \\ \theta = \arctan \dfrac{y}{x}, \\ \phi = \arctan \dfrac{\sqrt{x^2 + y^2}}{z} \\ \quad = \arccos \dfrac{z}{\sqrt{x^2 + y^2 + z^2}}. \end{cases}$$

- Path-traced objects with realistic non-Lambertian material



UNIVERSITY OF TORONTO

- Inserting virtual objects into real world scenes

Volume Rendering

Utilizing the Output of our model

Once we have the volume density and the directional emitted radiance at a point how are we going to render the image?

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

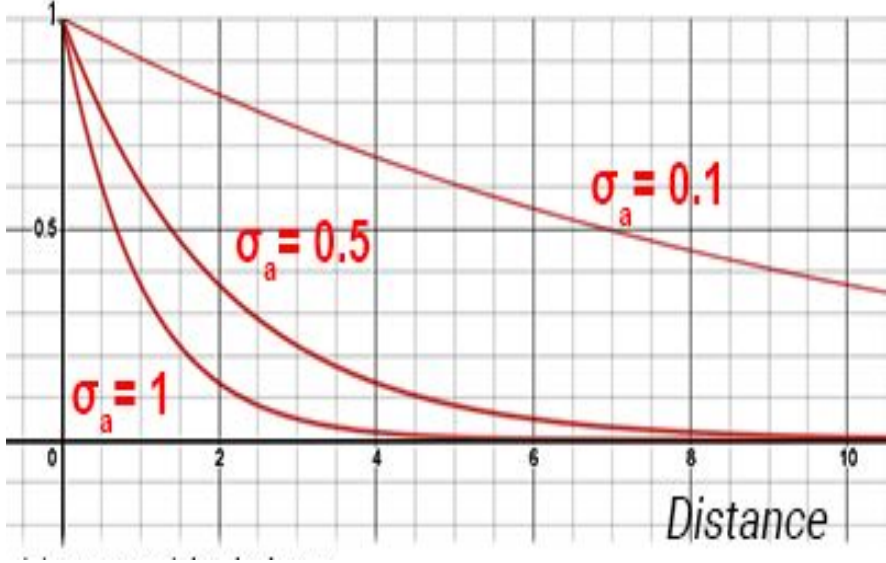
$\sigma(\mathbf{x})$: Colour Density at the position x

$C(\mathbf{r})$: Expected Colour of the ray r

$T(t)$: Accumulated Transmittance along the ray from tn to t

Calculation of Transmittance

Transmittance



$$\mathcal{T}(t+dt) = \mathcal{T}(t) \cdot (1 - dt \cdot \sigma(t))$$

$$\frac{\mathcal{T}(t+dt) - \mathcal{T}(t)}{dt} \equiv \mathcal{T}'(t) = -\mathcal{T}(t) \cdot \sigma(t)$$

This is a classical differential equation that can be solved as follows:

$$\mathcal{T}'(t) = -\mathcal{T}(t) \cdot \sigma(t) \tag{3}$$

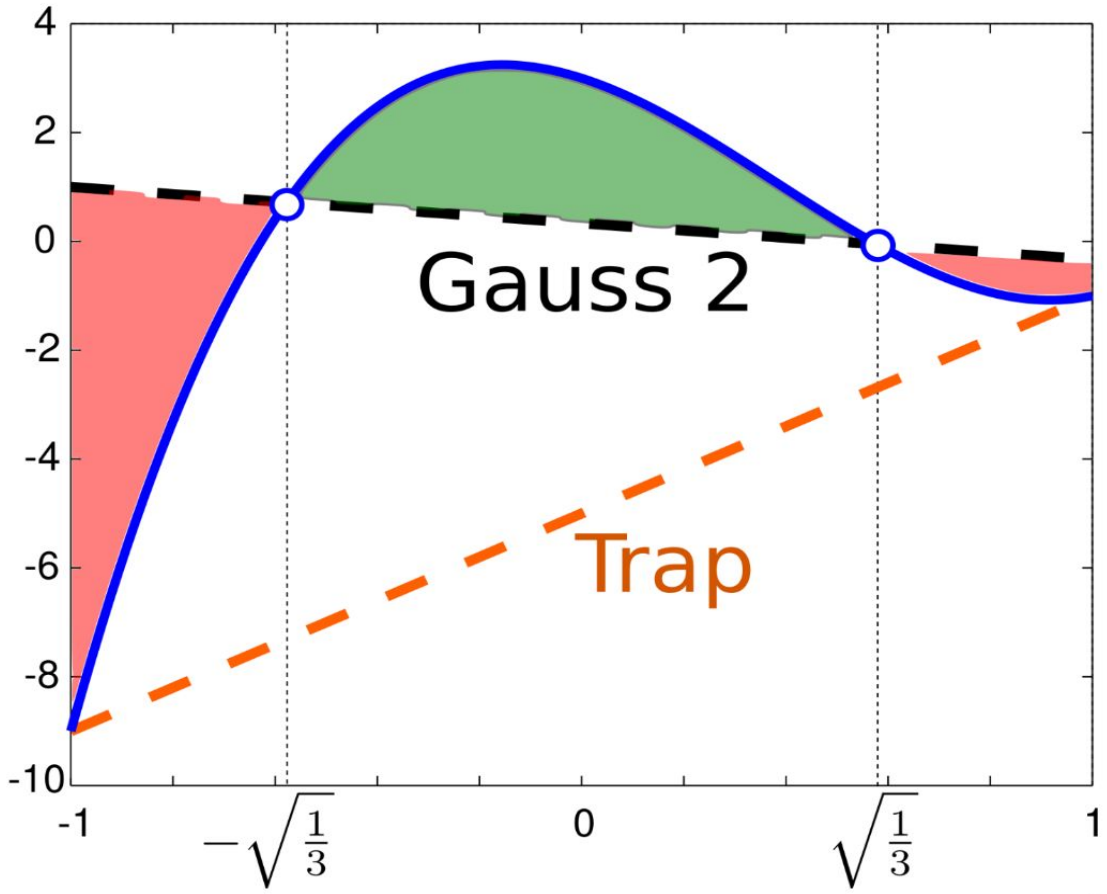
$$\frac{\mathcal{T}'(t)}{\mathcal{T}(t)} = -\sigma(t) \tag{4}$$

$$\int_a^b \frac{\mathcal{T}'(t)}{\mathcal{T}(t)}\, dt = -\int_a^b \sigma(t)\, dt \tag{5}$$

$$\log \mathcal{T}(t)|_a^b = -\int_a^b \sigma(t)\, dt \tag{6}$$

$$\mathcal{T}(a \to b) \equiv \frac{\mathcal{T}(b)}{\mathcal{T}(a)} = \exp\left(-\int_a^b \sigma(t)\, dt\right) \tag{7}$$

Deterministic Quadrature



$$\int_{-1}^{1} f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i),$$

$$w_i = \frac{2}{\left(1 - x_i^2\right)\left[P_n'(x_i)\right]^2}.$$

Stratified Sampling

We partition [tn,tf] into N even sized bins and sample from each of the bins.

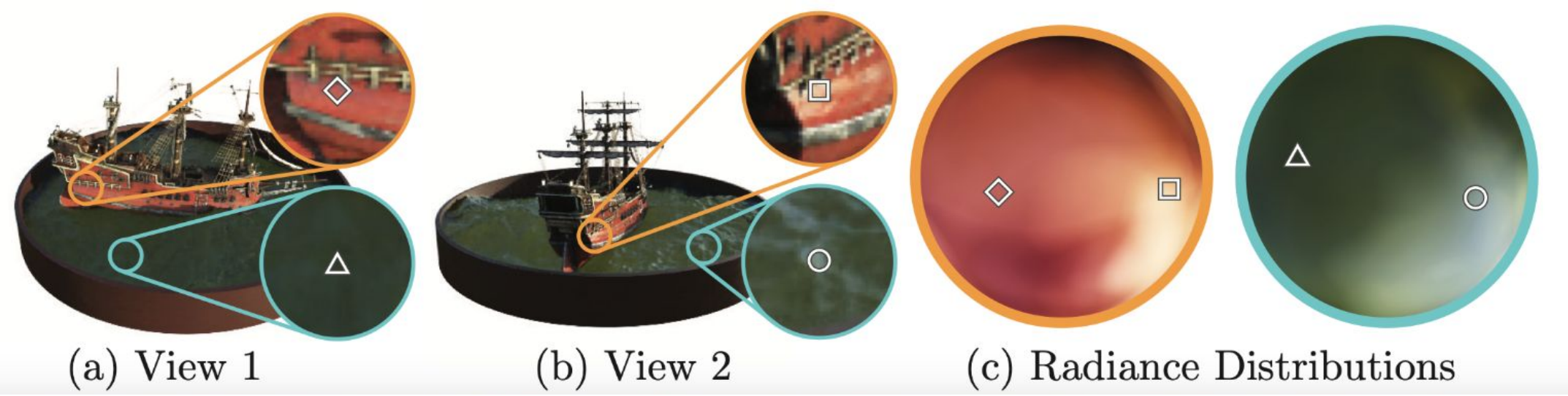
$$t_i \sim \mathcal{U}\left[t_n + \frac{i-1}{N}(t_f - t_n),\ t_n + \frac{i}{N}(t_f - t_n)\right]$$

We transform the integral into a Riemann Sum using the above sampling

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

An Example applying Stratified Sampling

View Dependent emitted radial field



Optimization

This is not enough for state-of-the-art yet

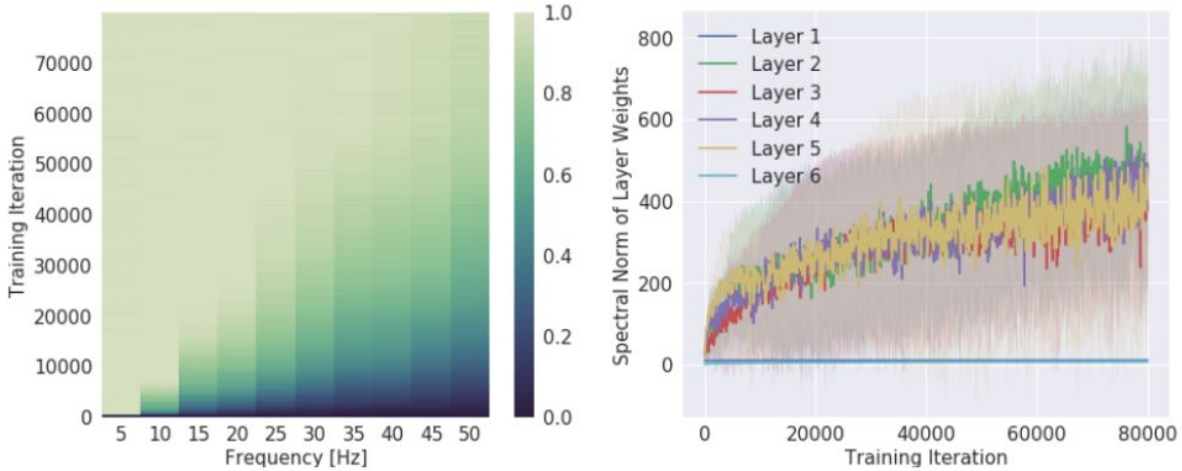
Positional Encoding

- NNs perform poorly on high frequency variations in colour and geometry
- Mapping the inputs to a higher dimensional space using high frequency functions before passing them helps

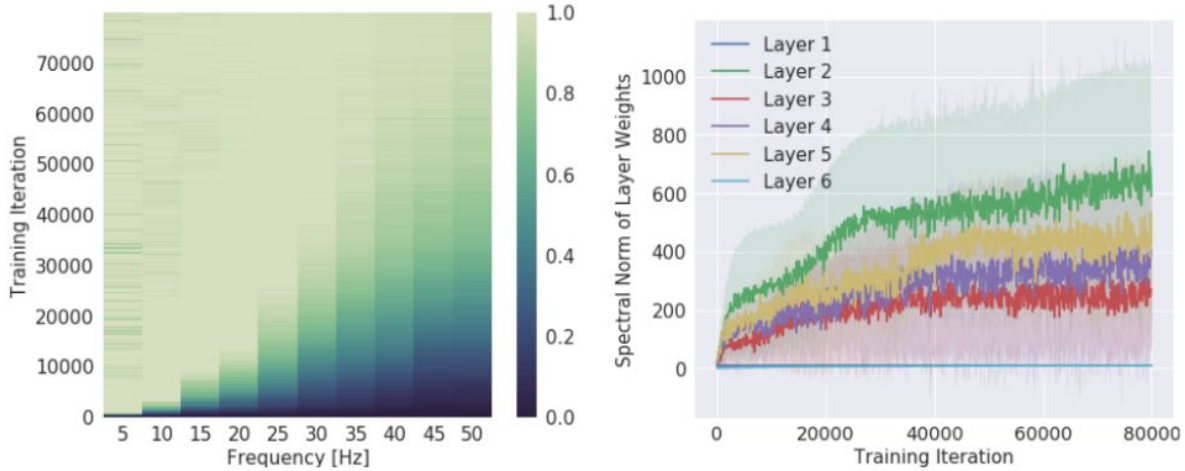
Positional Encoding

On the Spectral Bias of Neural Networks

Nasim Rahaman [*1 2] Aristide Baratin [*1] Devansh Arpit [1] Felix Draxler [2] Min Lin [1] Fred A. Hamprecht [2]
Yoshua Bengio [1] Aaron Courville [1]



(a) Equal Amplitudes



(b) Increasing Amplitudes

Positional Encoding

$$F_\Theta = F'_\Theta \circ \gamma$$

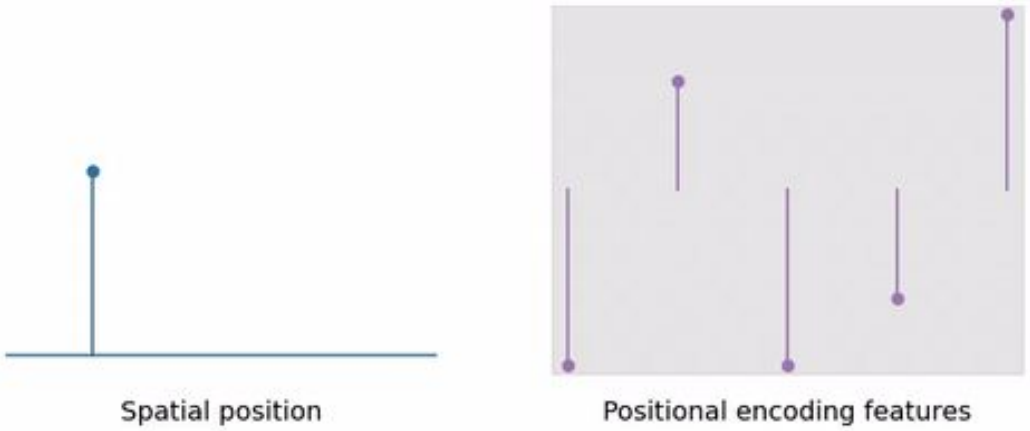
Here γ is a mapping from $\mathbb{R}$ into a higher dimensional space $\mathbb{R}^{2L}$, and F'_Θ is still simply a regular MLP. Formally, the encoding function we use is:

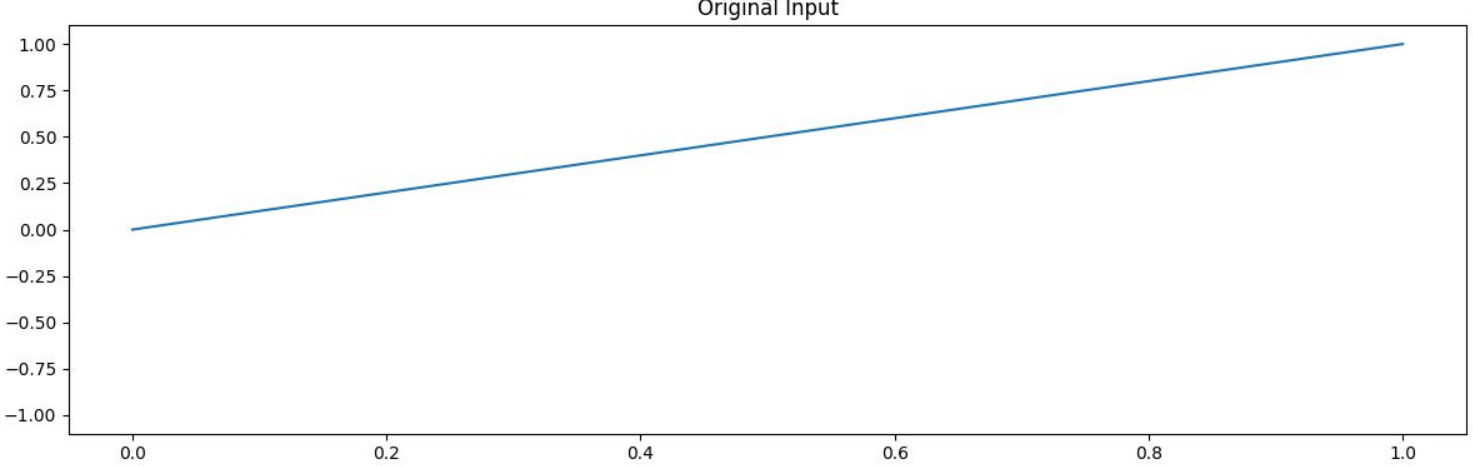
$$\gamma(p) = \left(\sin\left(2^0\pi p\right), \cos\left(2^0\pi p\right), \cdots, \sin\left(2^{L-1}\pi p\right), \cos\left(2^{L-1}\pi p\right)\right). \quad (4)$$

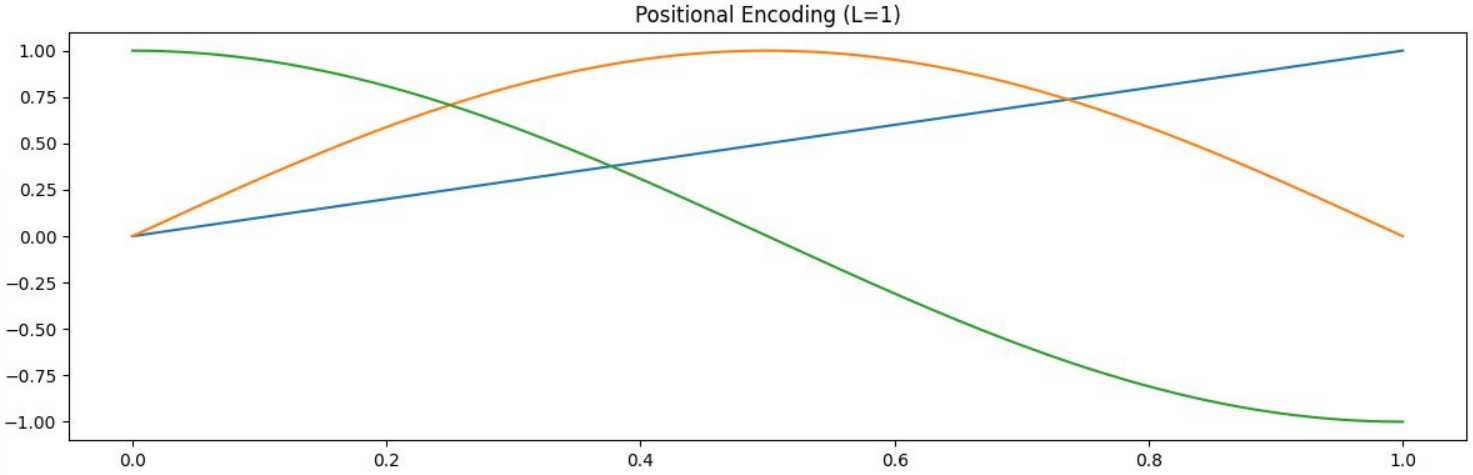
Positional Encoding

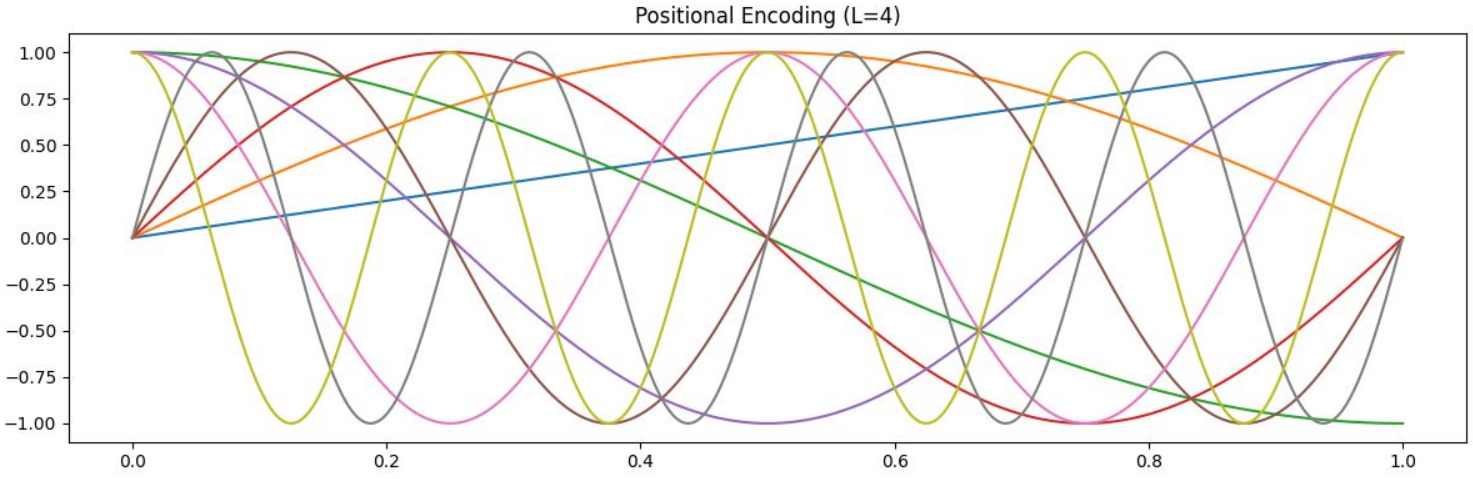
$$F_\Theta = F'_\Theta \circ \gamma$$

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









UNIVERSITY OF TORONTO | CVDG

Positional Encoding

$$F_\Theta = F'_\Theta \circ \gamma$$

Here γ is a mapping from $\mathbb{R}$ into a higher dimensional space $\mathbb{R}^{2L}$, and F'_Θ is still simply a regular MLP. Formally, the encoding function we use is:

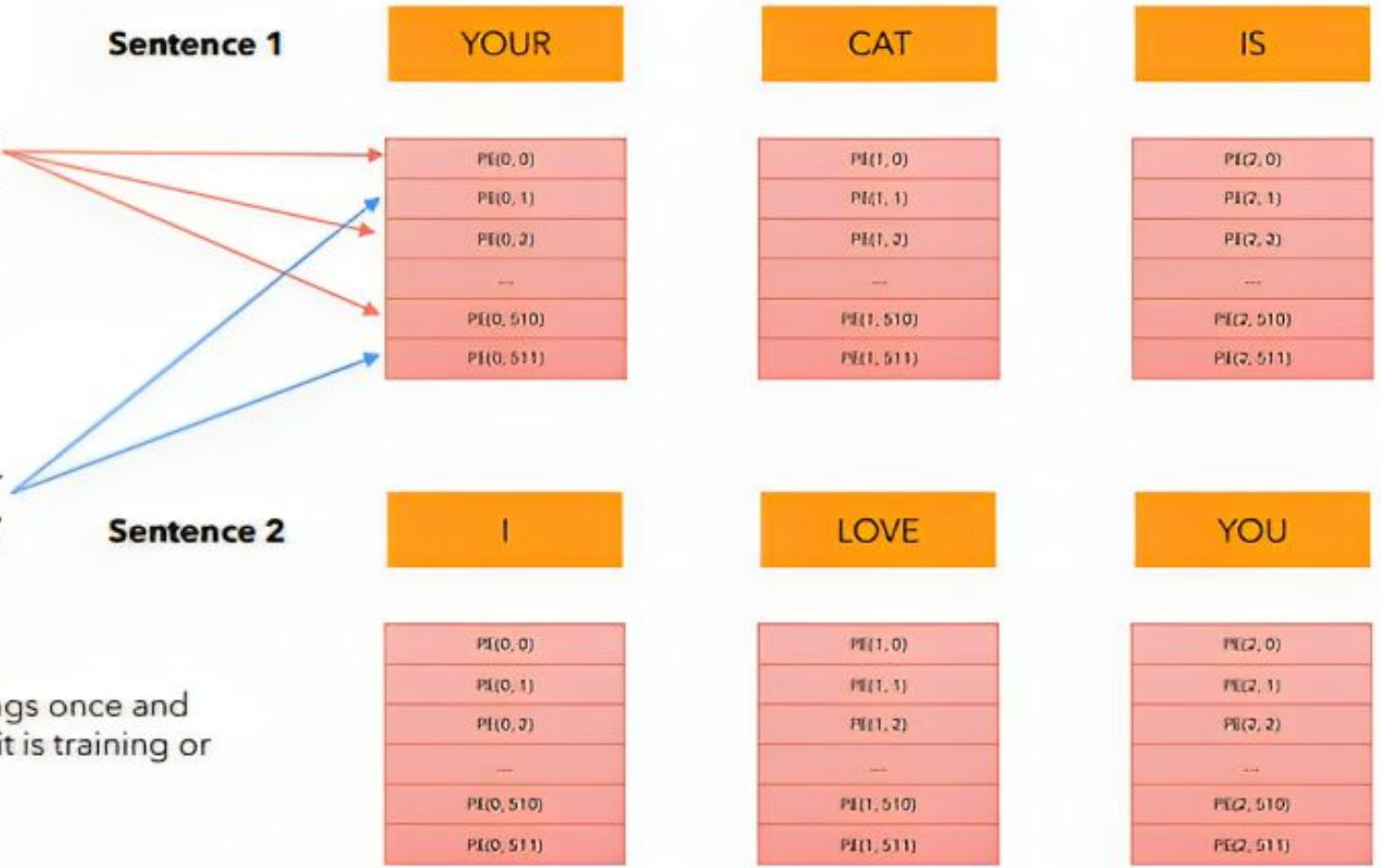
$$\gamma(p) = \left(\sin\left(2^0\pi p\right), \cos\left(2^0\pi p\right), \cdots, \sin\left(2^{L-1}\pi p\right), \cos\left(2^{L-1}\pi p\right)\right). \quad (4)$$

This function $\gamma(\cdot)$ is applied separately to each of the three coordinate values in $\mathbf{x}$ (which are normalized to lie in $[-1, 1]$) and to the three components of the Cartesian viewing direction unit vector $\mathbf{d}$ (which by construction lie in $[-1, 1]$)

Positional Encoding

$$PE(pos, 2i) = \sin \frac{pos}{10000^{\frac{2i}{d_{model}}}}$$

$$PE(pos, 2i+1) = \cos \frac{pos}{10000^{\frac{2i}{d_{model}}}}$$

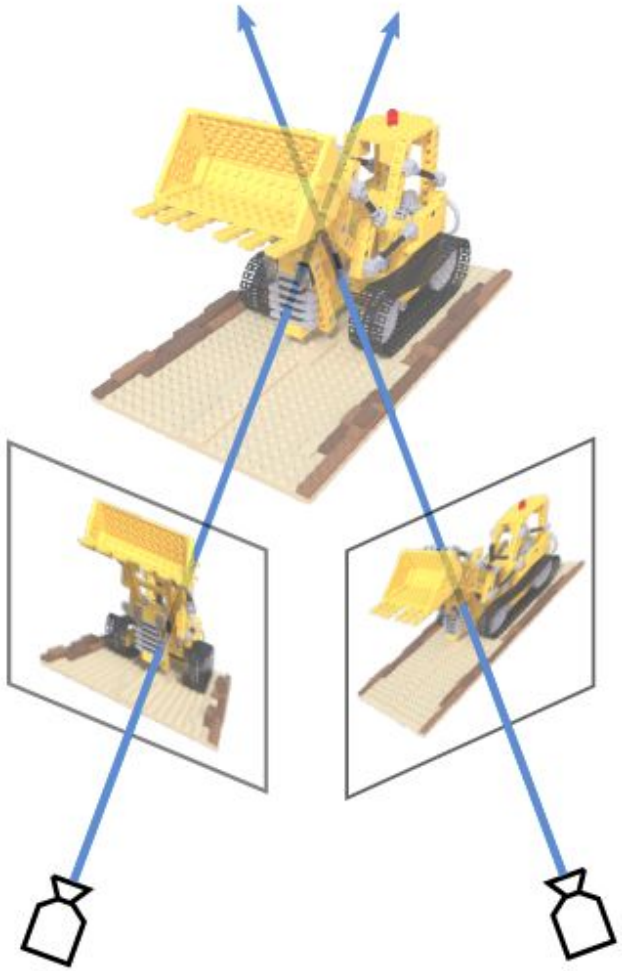


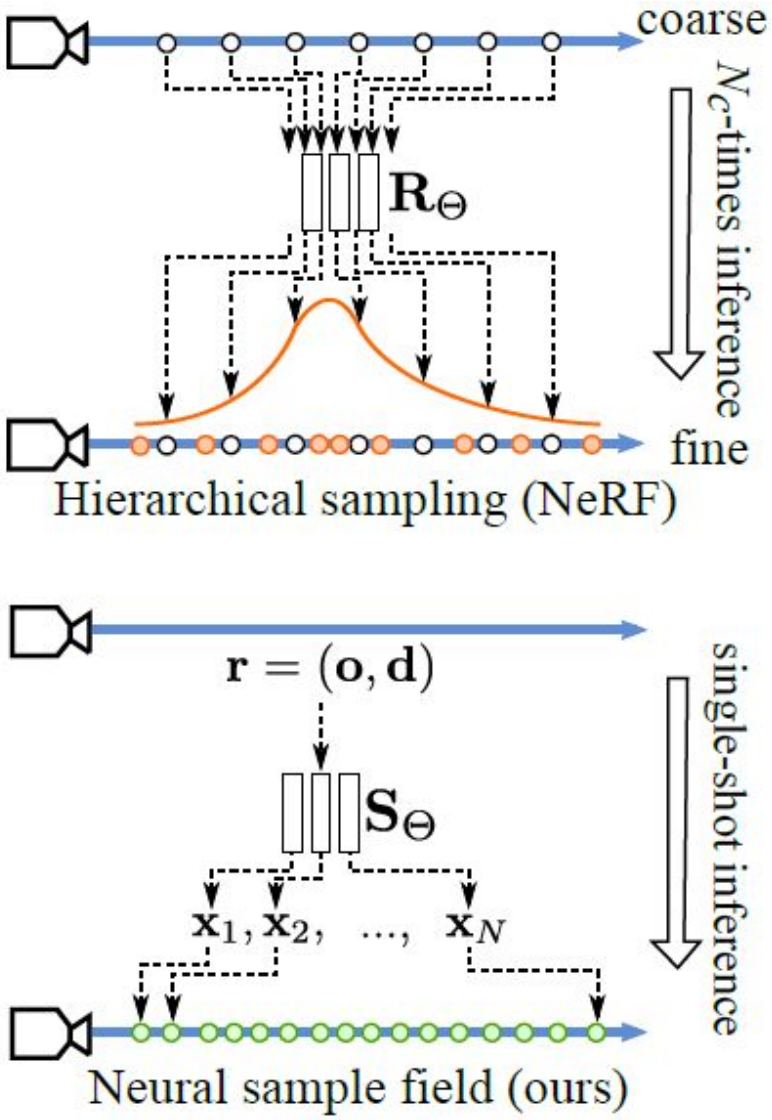
We only need to compute the positional encodings once and then reuse them for every sentence, no matter if it is training or inference.

[Source: Umar Jamil]

Hierarchical Volume Sampling

- Using N query points along each camera ray is inefficient: free space and occluded regions that do not contribute to the rendered image are still sampled repeatedly
- Now allocate samples proportionally to their expected effect on the final rendering

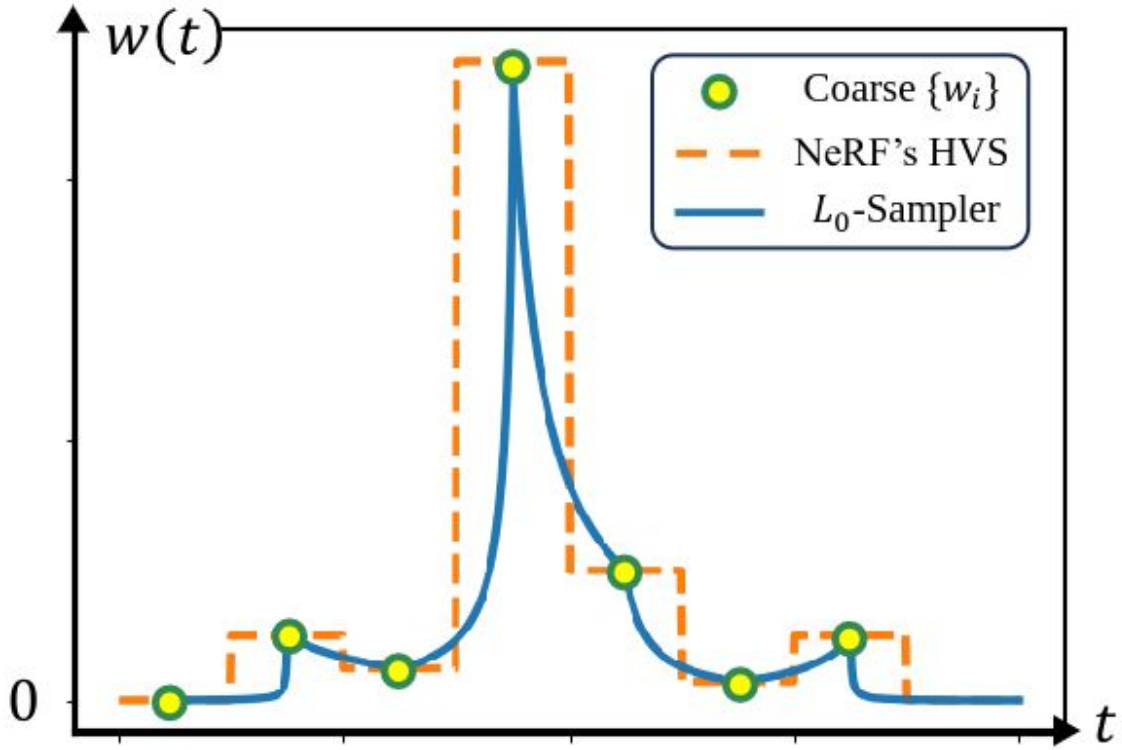




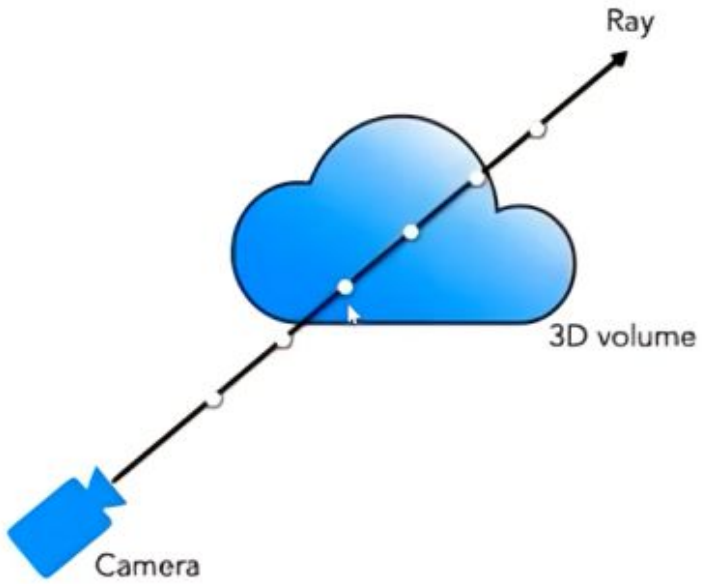
Hierarchical Volume Sampling

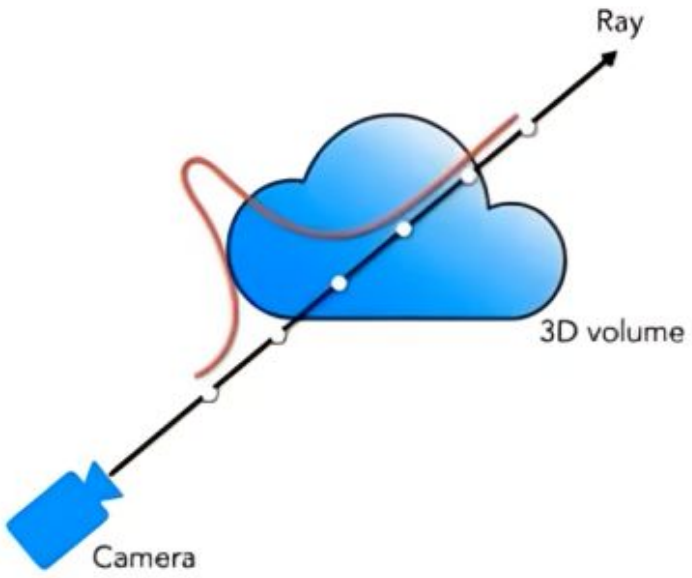
$$\hat{C}_c(\mathbf{r}) = \sum_{i=1}^{N_c} w_i c_i \,, \qquad w_i = T_i(1 - \exp(-\sigma_i \delta_i)) \,. \tag{5}$$

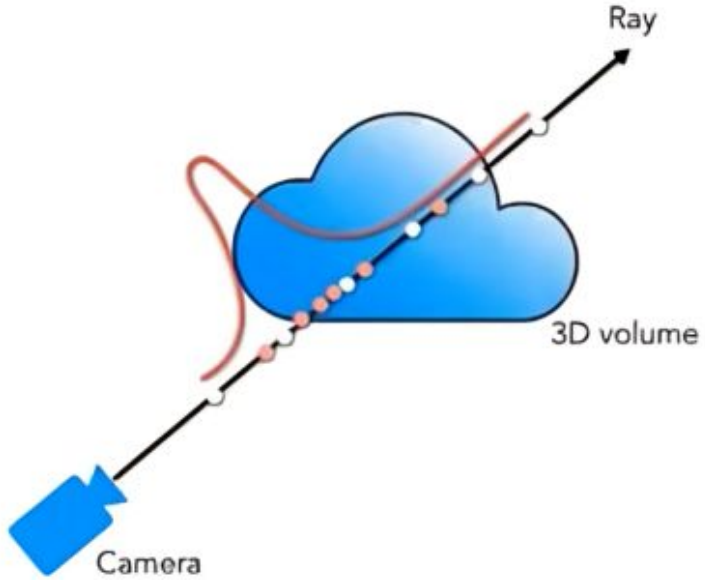
Normalizing these weights as $\hat{w}_i = w_i / \sum_{j=1}^{N_c} w_j$ produces a piecewise-constant PDF along the ray. We sample a second set of N_f locations from this distribution



Hierarchical Volume Sampling



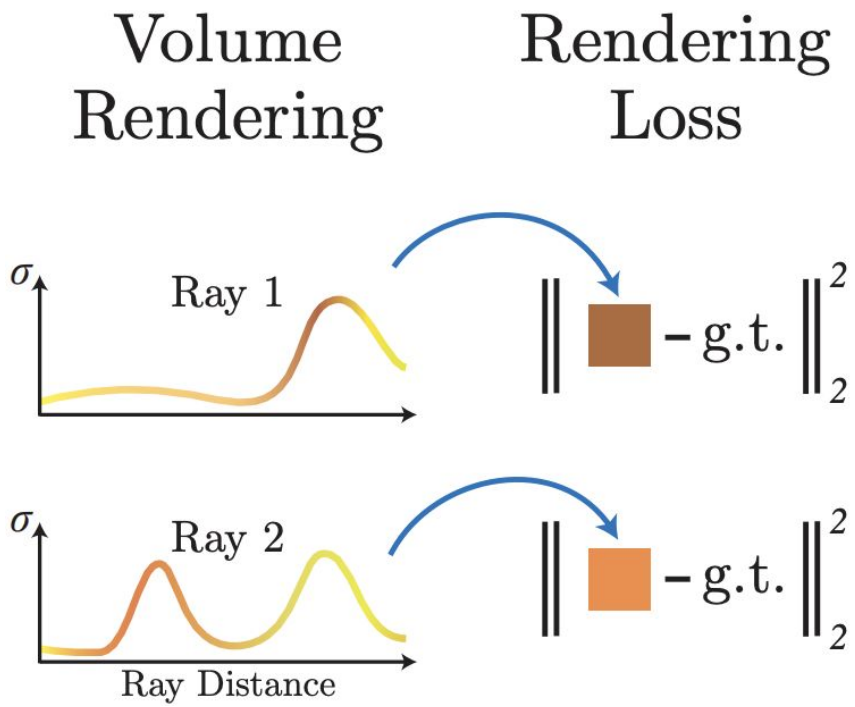




Implementation Details

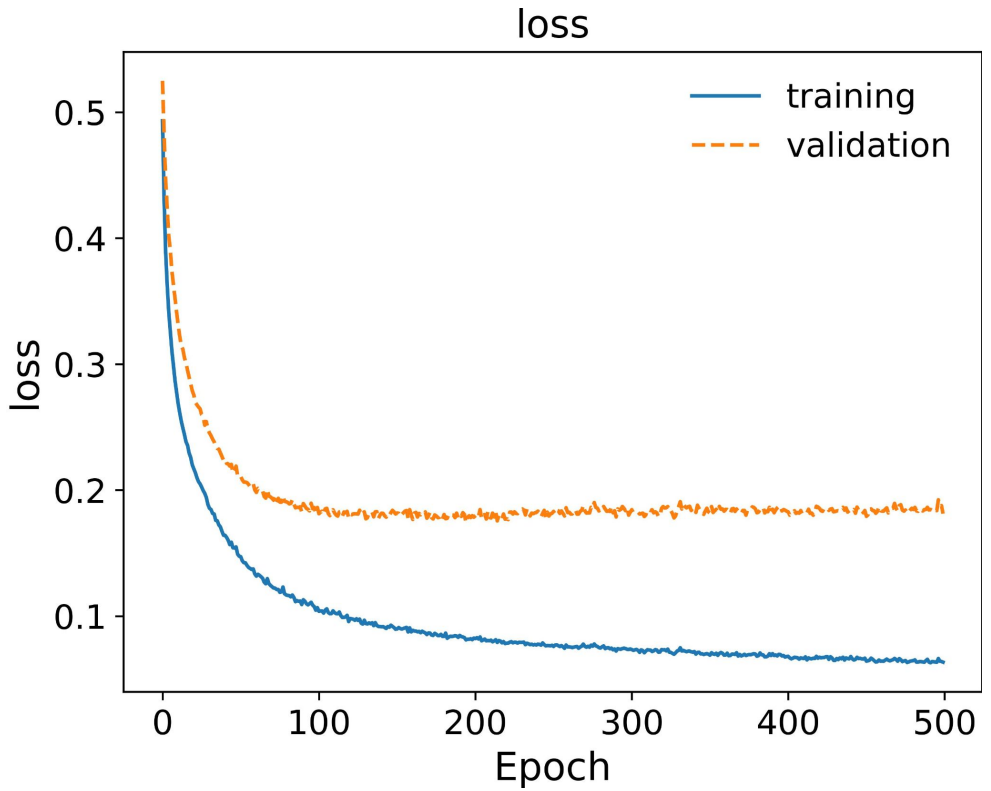
from both sets of samples. Our loss is simply the total squared error between the rendered and true pixel colors for both the coarse and fine renderings:

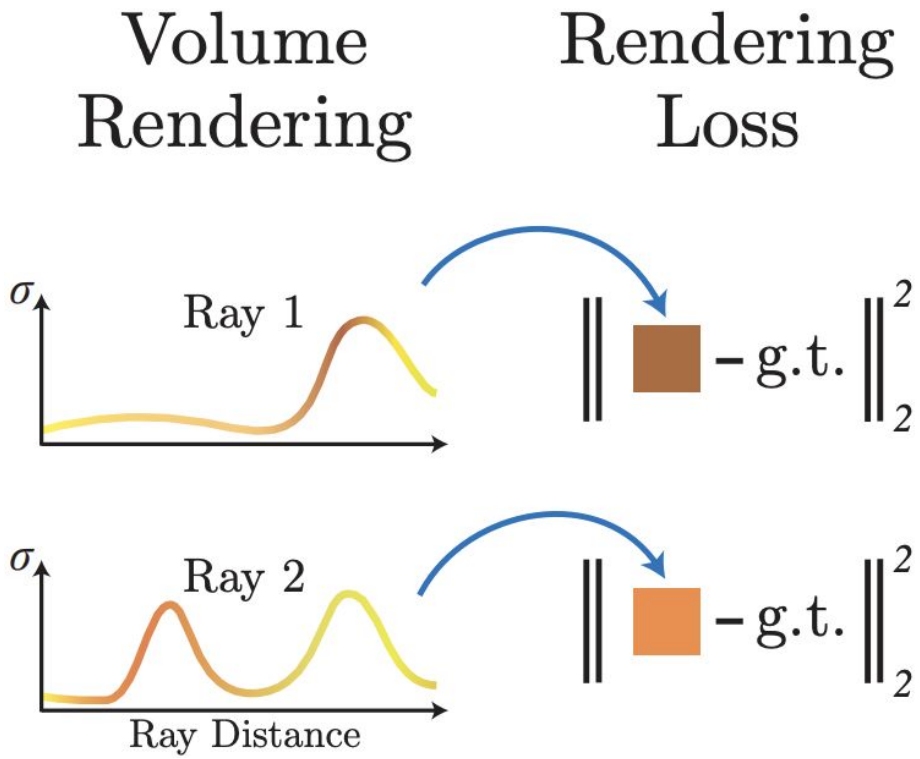
$$\mathcal{L} = \sum_{\mathbf{r}\in\mathcal{R}} \left[\left\| \hat{C}_c(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 + \left\| \hat{C}_f(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 \right] \tag{6}$$



Implementation Details

- Batch Size = 4096, Nc = 64, Nf = 128, Adam
- The optimization for a single scene takes 100-300k iterations to converge on a single NVIDIA V100 GPU (about 1–2 days)





Results

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

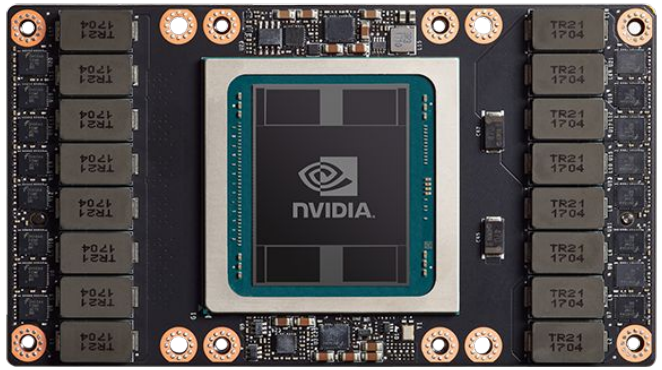
Table 1: Our method quantitatively outperforms prior work on datasets of both synthetic and real images. We report PSNR/SSIM (higher is better) and LPIPS [50] (lower is better). The DeepVoxels [41] dataset consists of 4 diffuse objects with simple geometry. Our realistic synthetic dataset consists of 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.

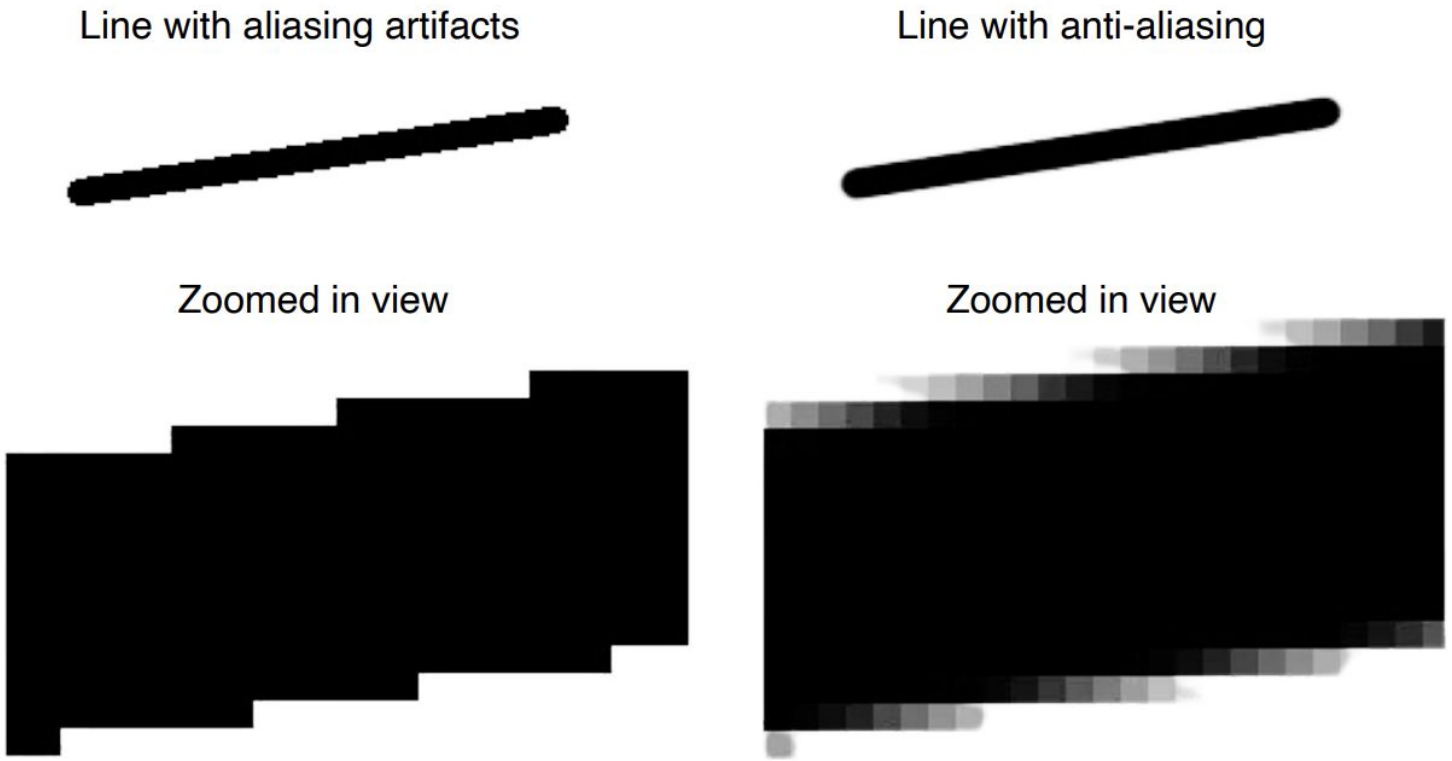
Limitations and Improvements

4 years is a long time…

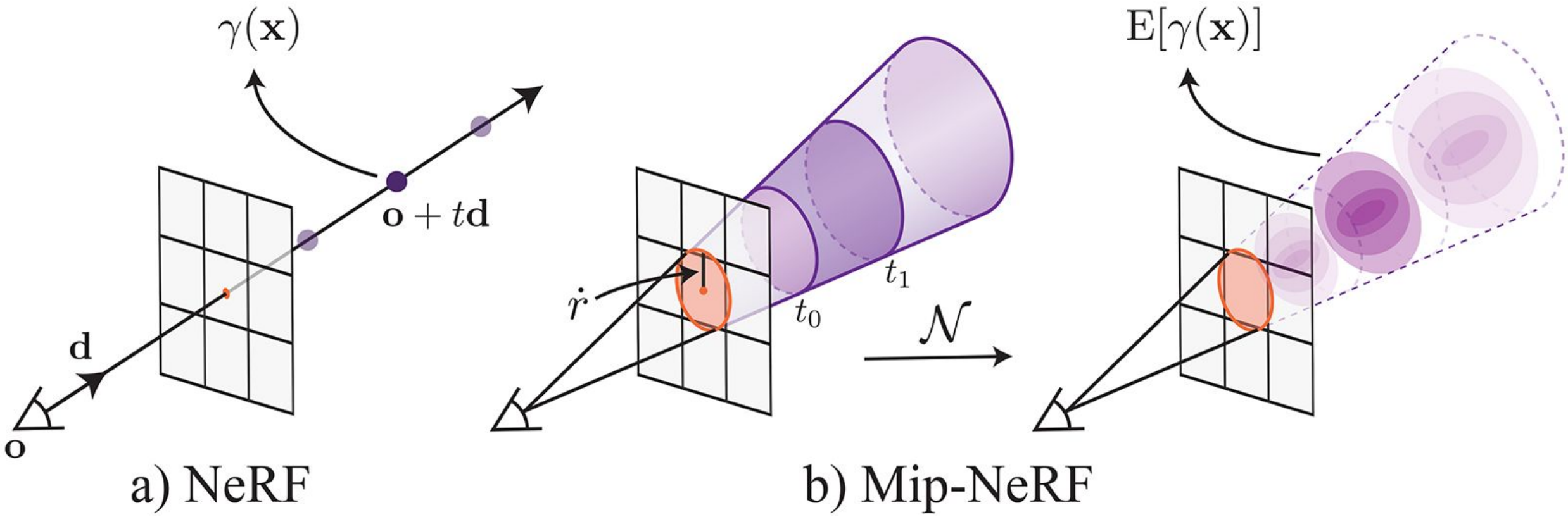
Limitations

- Aliasing problems
- Memory (12+ GB RAM) and compute (2 Days V100) requirement
- Large dataset required (50-100 per scene)



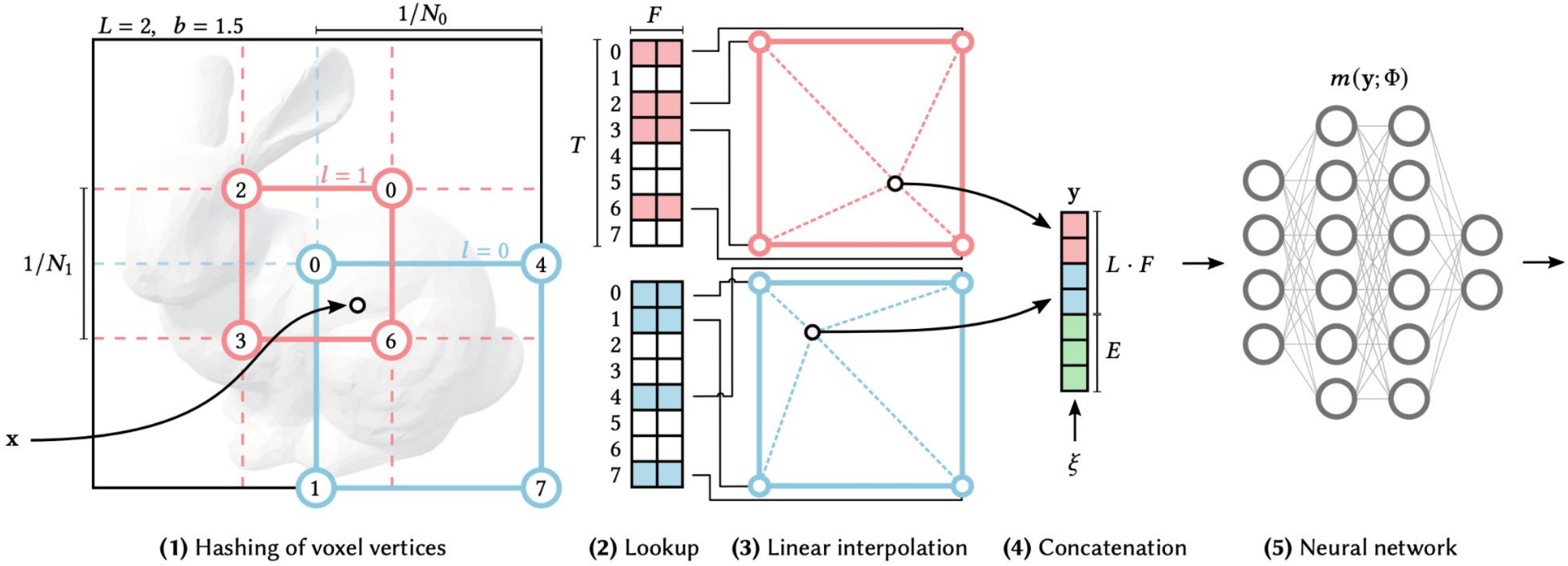


Mip-NeRF - Addressing Aliasing and Blurriness (2021)



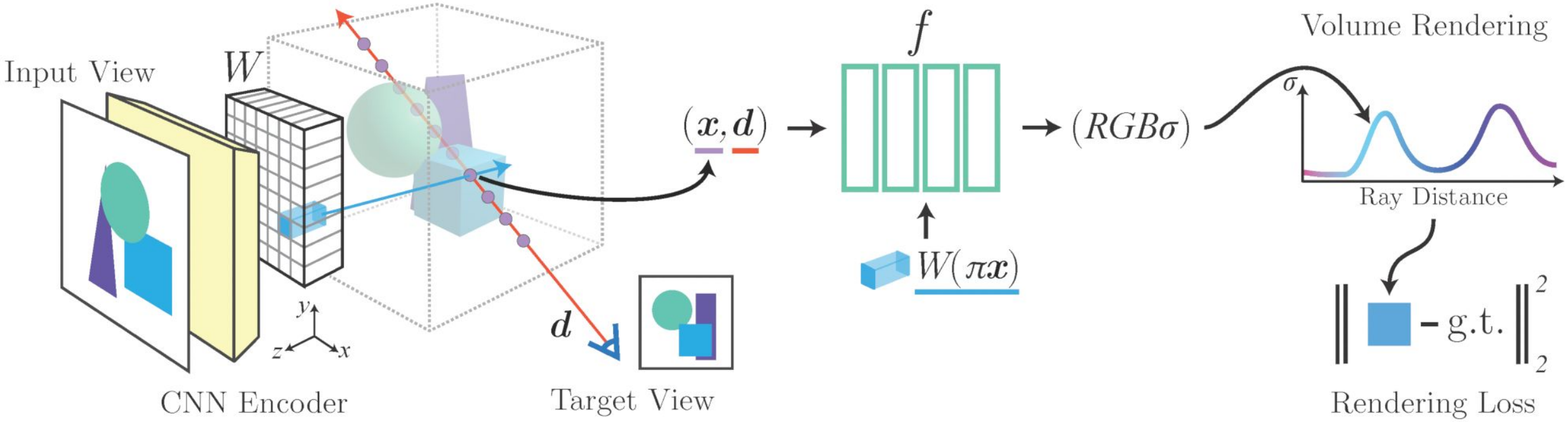
https://jonbarron.info/mipnerf/

Instant NGP - Addressing Time Constraints (2022)



https://nvlabs.github.io/instant-ngp/

pixelNeRF - Addressing Data Requirements (2021)



https://alexyu.net/pixelnerf/

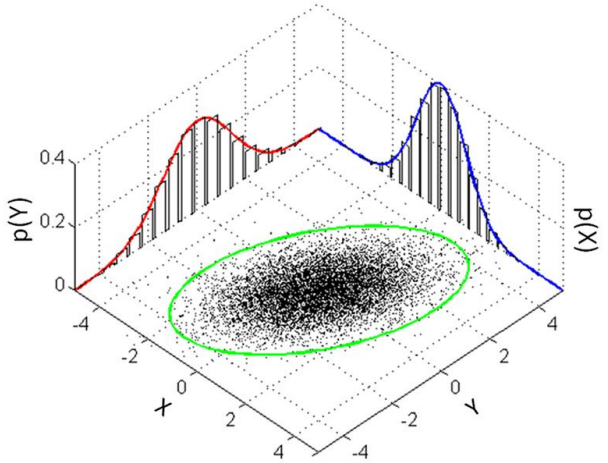
Other Works

- https://www.matthewtancik.com/nerf
- https://mobile-nerf.github.io/
- https://nerfies.github.io/
- https://www.nvidia.com/en-us/research/ai-art-gallery/instant-nerf/
- https://meganerf.cmusatyalab.org/

Slides on github.com/ut-cvdg/paper-breakdown

Thank You

Up Next: 3D Splatting



Follow us on social media

UNIVERSITY OF TORONTO | CVDG