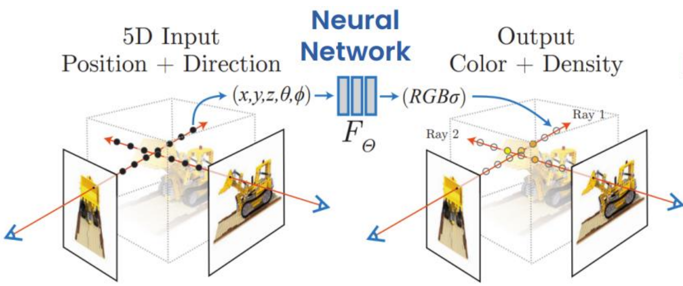
NeRF-like Approaches for Light Transport Algorithms

2022-05-02

Kyubeom Han

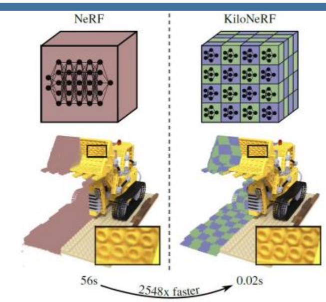
KAIST

NeRF and Extensions

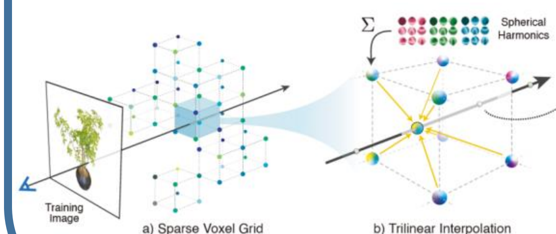


NeRF

Acceleration



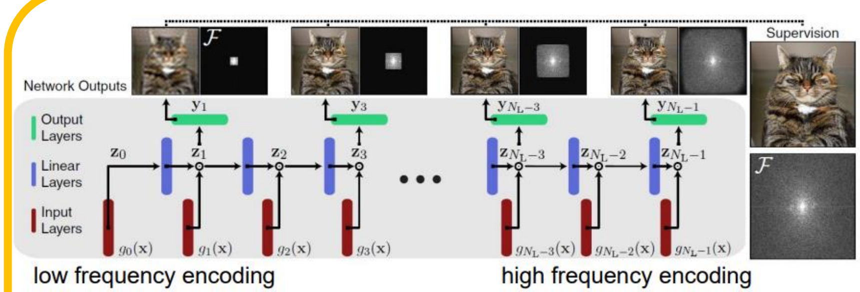
KiloNeRF



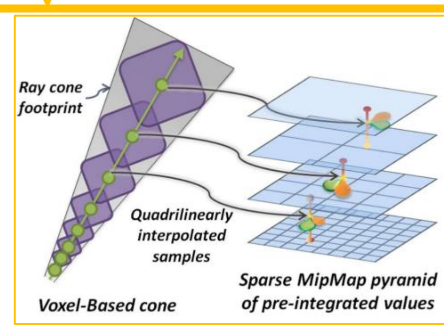
Plenoxels

Generalization

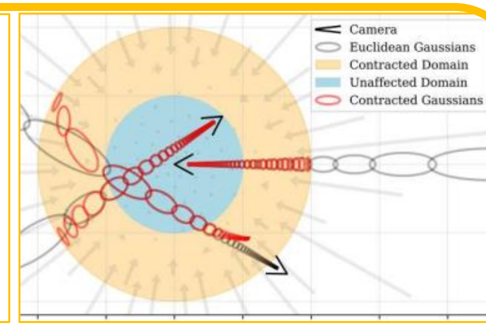
Scalability



Bacon

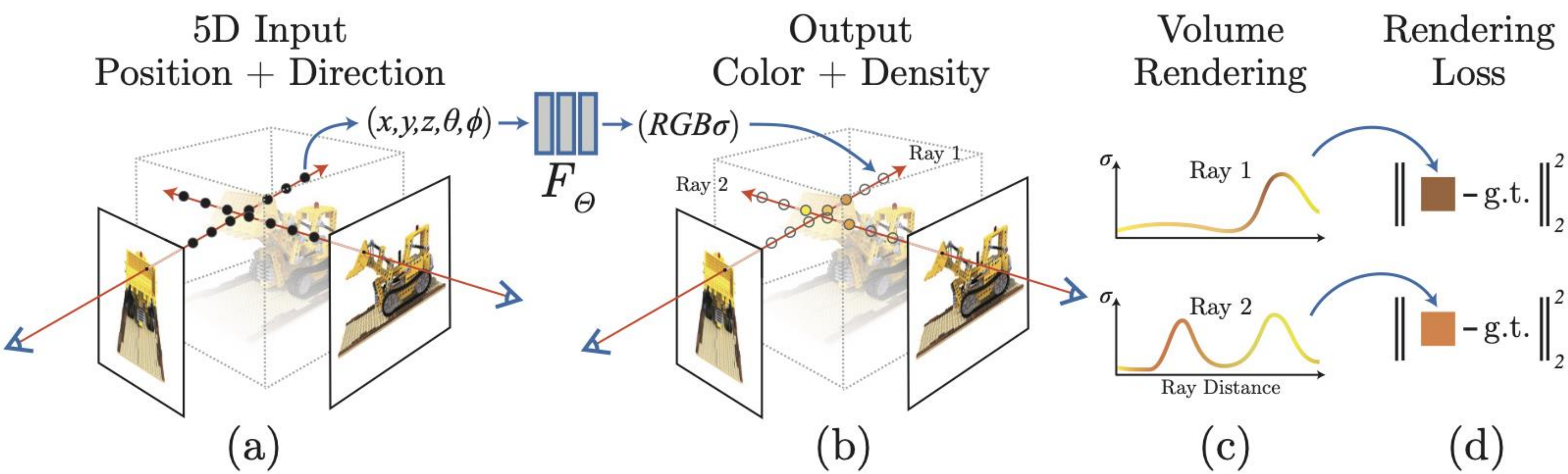


Mip-NeRF



Mip-NeRF 360

NeRF and Volume Rendering



- What about other light transport algorithms?
- **Especially, what we learned in CS580?**

NeRF and Light Transport Algos.

- Neural Radiosity (SIGGRAPH ASIA 2021)
  - Path Tracing + **Radiosity** + NeRF

- Real-time Neural Radiance Caching for Path Tracing (SIGGRAPH 2021)
  - Path Tracing + **Radiance Caching** + NeRF

Neural Radiosity

Hadadan et al., SIGGRAPH Asia 2021

Main Contribution

Solving the Rendering Equation by Radiance-predicting Neural Network via Radiosity-like Training

Main Contribution

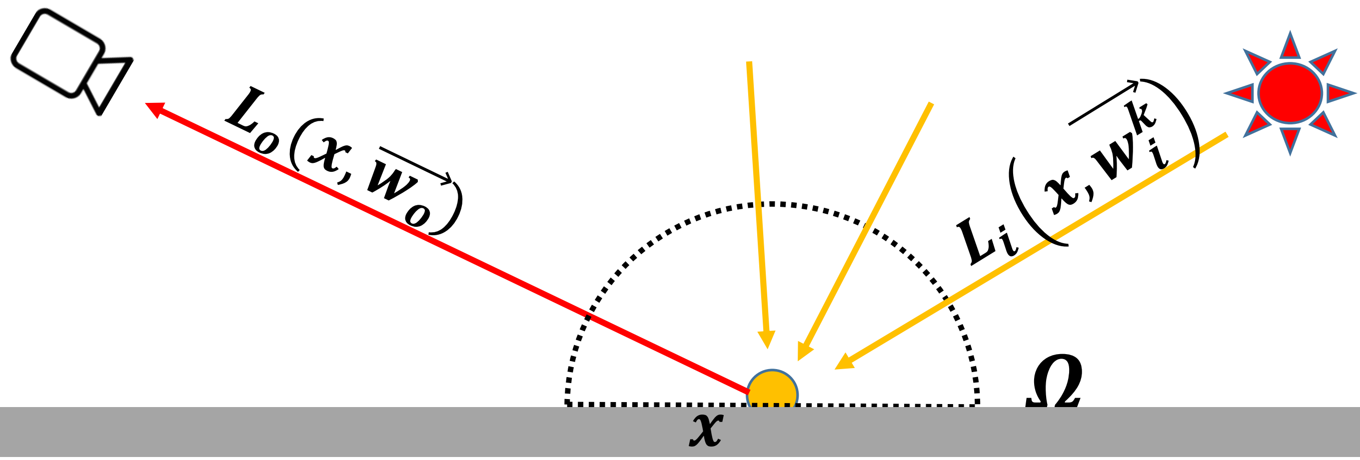
Solving the Rendering Equation by Radiance-predicting Neural Network via Radiosity-like Training

Radiance-predicting Neural Network

$$L_o(x,\overrightarrow{w_o}) = L_e(x,\overrightarrow{w_o}) + \int_\Omega f_r(x,\overrightarrow{w_i},\overrightarrow{w_o})L_i(x,\overrightarrow{w_i})(\overrightarrow{w_i}\cdot\vec{n})d\overrightarrow{w_i}$$

$$\sim L_e(x,\overrightarrow{w_o}) + \frac{1}{N}\sum_{k=1}^{N}\frac{f_r\left(x,\overrightarrow{w_i^k},\overrightarrow{w_o}\right)L_i\left(x,\overrightarrow{w_i^k}\right)\left(\overrightarrow{w_i^k}\cdot\vec{n}\right)}{p(\overrightarrow{w_i^k})}$$

Solving rendering equation via Monte Carlo Integration

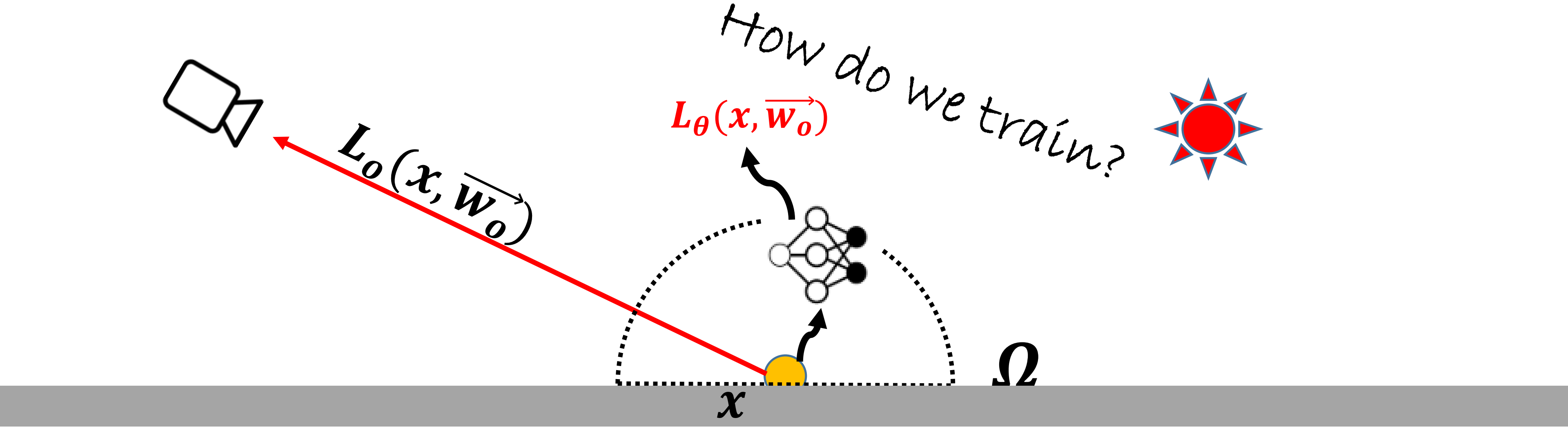


Radiance-predicting Neural Network

$$L_o(x, \overrightarrow{w_o}) = L_e(x, \overrightarrow{w_o}) + \int_{\Omega} f_r(x, \overrightarrow{w_i}, \overrightarrow{w_o}) L_i(x, \overrightarrow{w_i})(\overrightarrow{w_i} \cdot \vec{n}\,) d\overrightarrow{w_i}$$

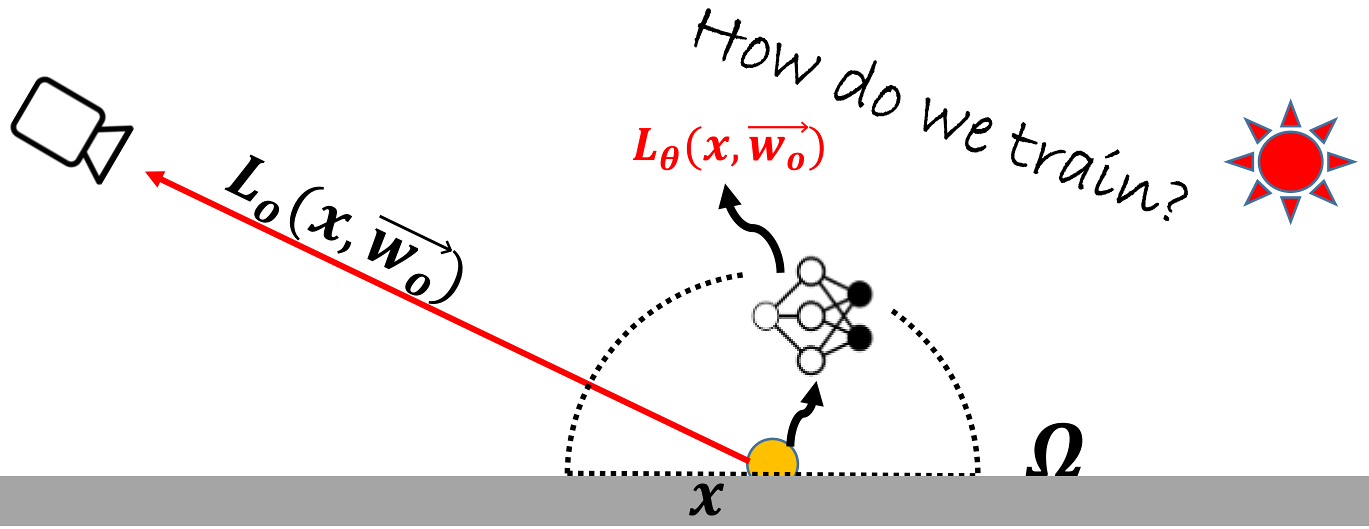
$$\sim L_e(x, \overrightarrow{w_o}) + L_\theta(x, \overrightarrow{w_o})$$

Solving rendering equation via Radiance-predicting Neural Network L_θ



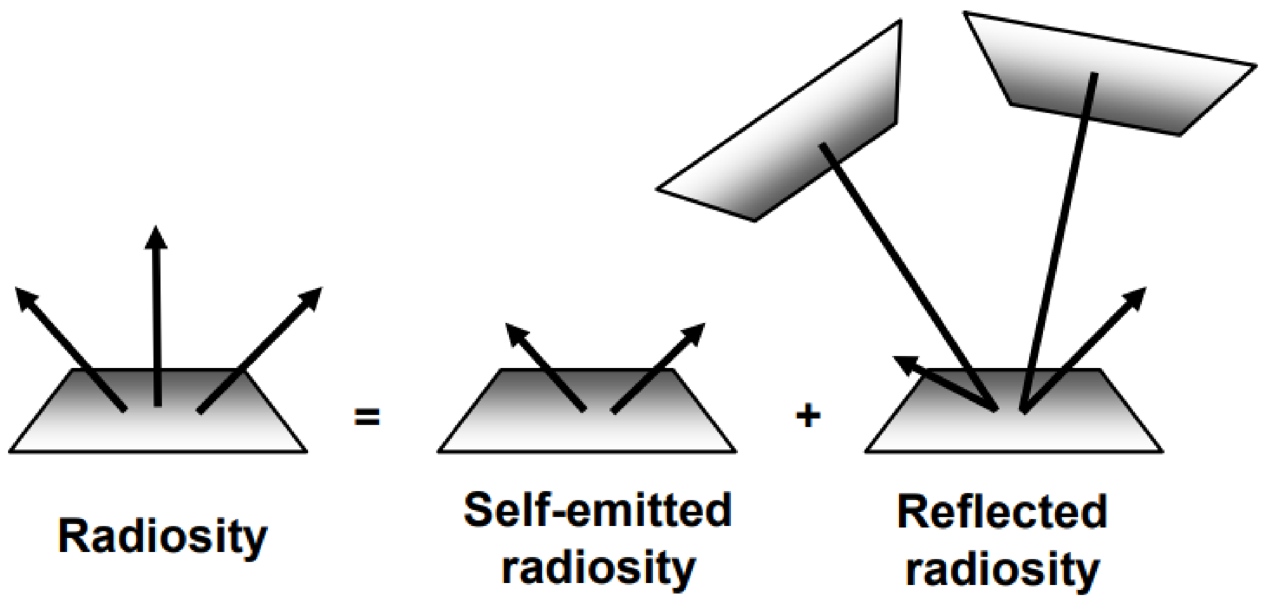
Radiance-predicting Neural Network

- Generating ground truth is to solve the rendering equation → Too much overhead!
- **How to train without the ground truth radiance?**



Radiosity: Recap

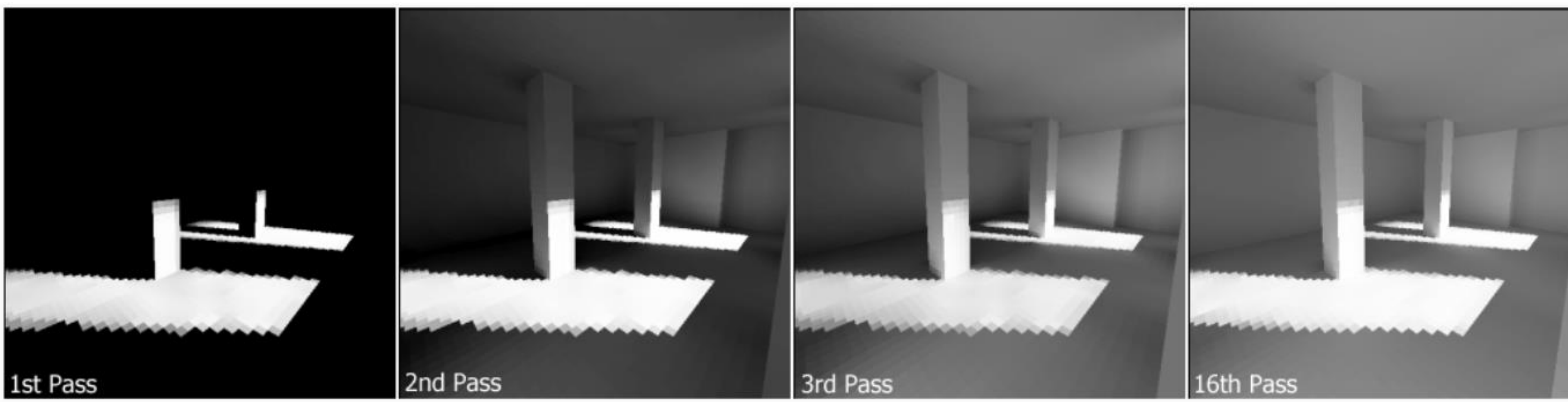
- Iteratively updating the radiosity of each polygon
  - Jacobi / Gauss-Seidel iteration



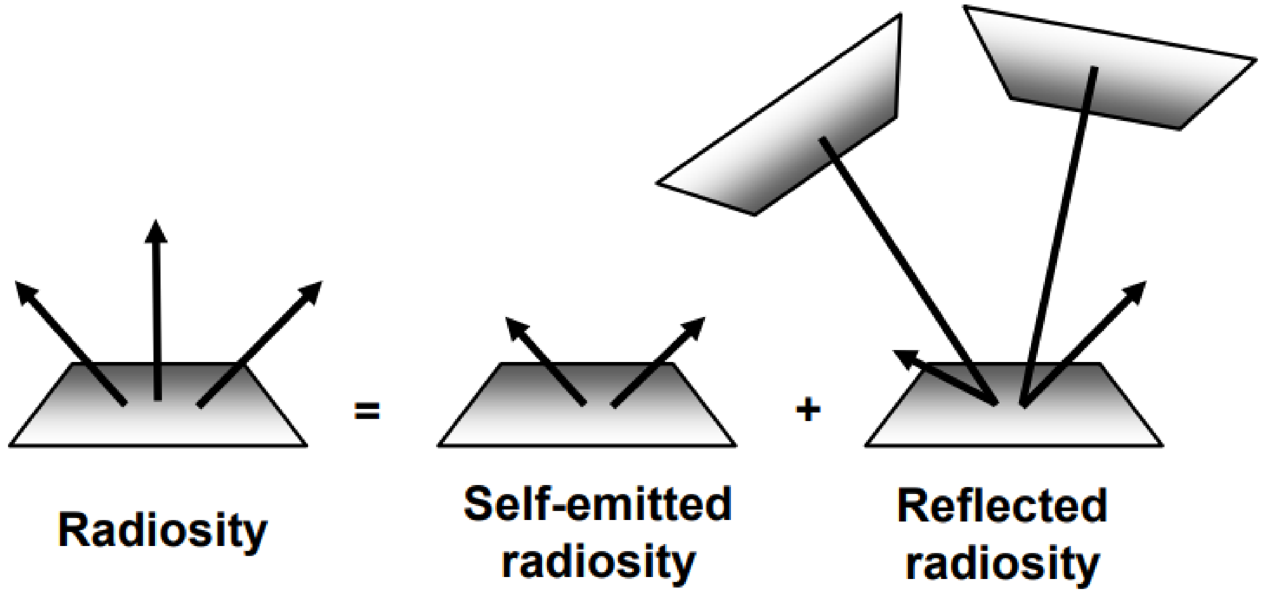
$$Radiosity_i = Radiosity_{self,i} + \sum_{j=1}^{N} a_{j\rightarrow i} Radiosity_j$$

Radiosity: Recap

- Iteratively updating the radiosity of each polygon
  - Jacobi / Gauss-Seidel iteration
- Updating allows to consider further light bounces



Neural Radiosity Method

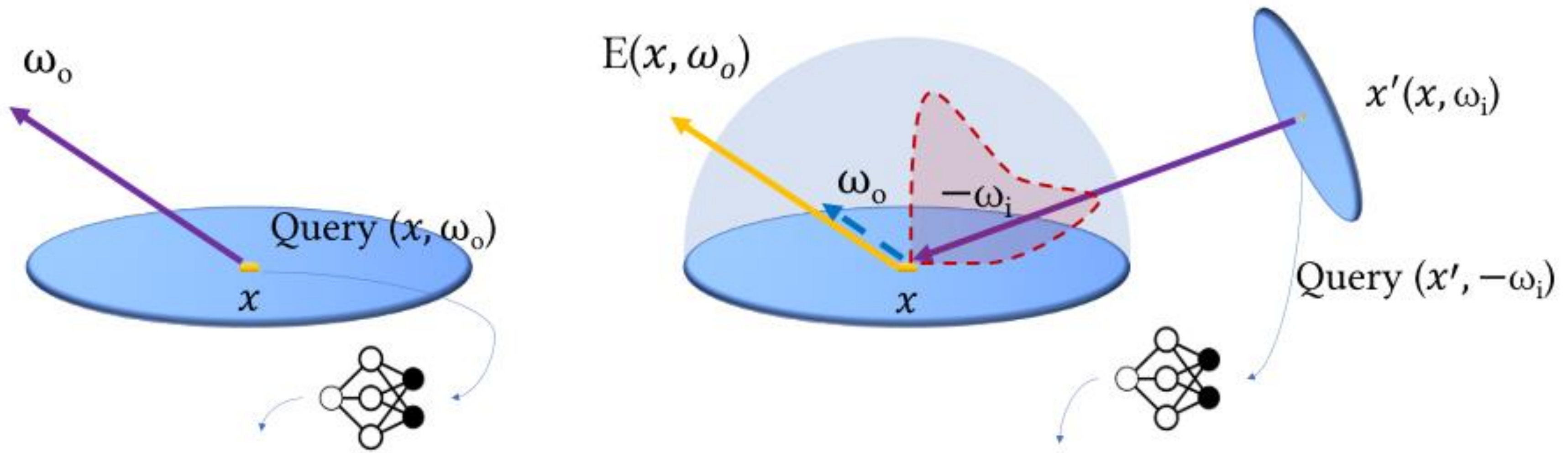


$$\underline{Radiosity_i} = \underline{Radiosity_{self,i}} + \underline{\sum_{j=1}^{N} a_{j \to i} Radiosity_j}$$

$$L_\theta(x, \omega_o) = L_e(x, \omega_o) + \int f(x, \omega_o, \omega_i) L_\theta(x'(x, \omega_i), -\omega_i) d\omega_i$$

Neural Radiosity Method

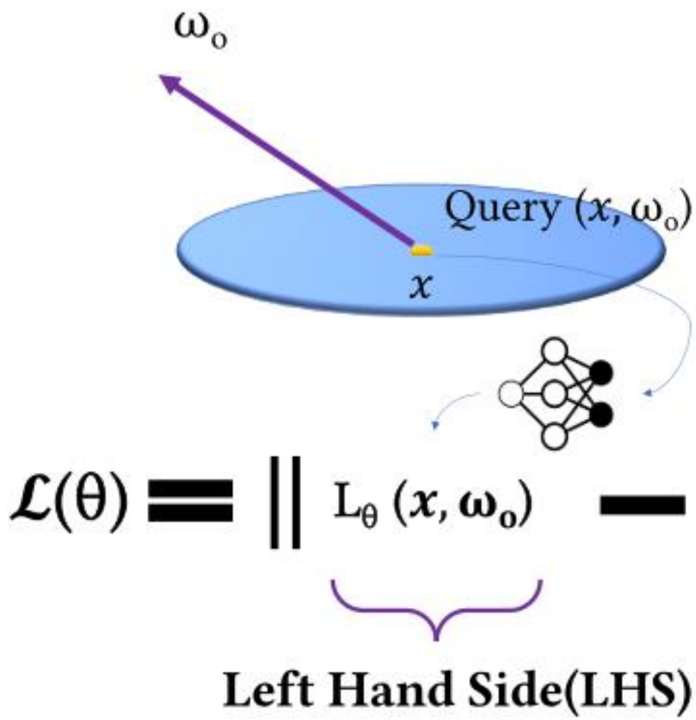
- Minimize the difference between the directly estimated outgoing radiance (LHS) and calculated outgoing radiance from estimated incoming radiances(RHS)



$$\mathcal{L}(\theta) = \Big\| \underbrace{L_\theta(\boldsymbol{x},\boldsymbol{\omega}_o)}_{\textbf{Left Hand Side(LHS)}} - \underbrace{\left(E(x,\omega_o) + \int f(x,\omega_o,\omega_i) . L_\theta\left(\boldsymbol{x}'(\boldsymbol{x},\boldsymbol{\omega}_i), -\boldsymbol{\omega}_i\right). d\omega_i^\perp\right)}_{\textbf{Right Hand Side(RHS)}} \Big\|_2^2$$

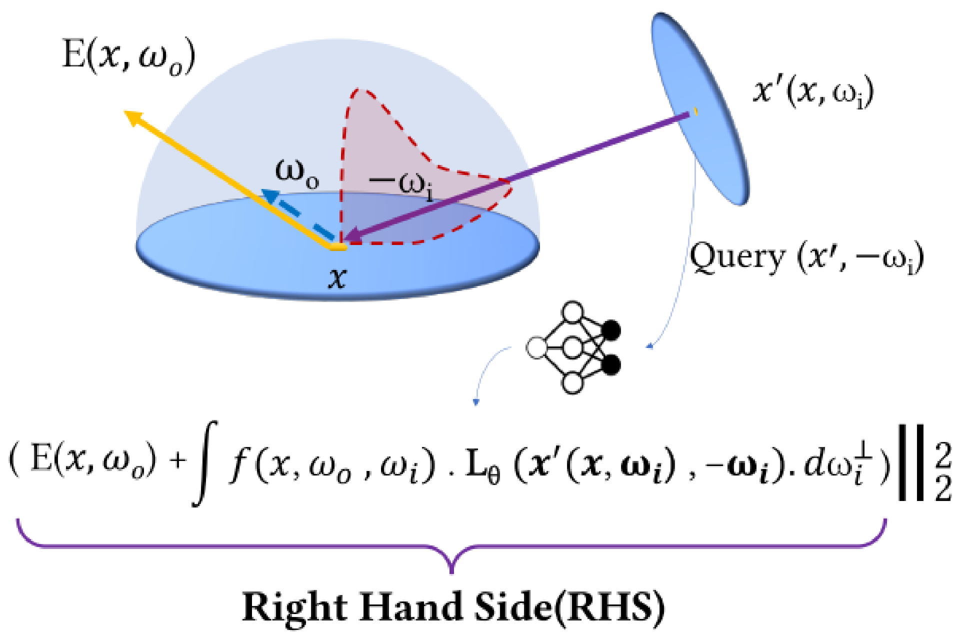
Neural Radiosity Method

- LHS: Outgoing radiance directly estimated by the network



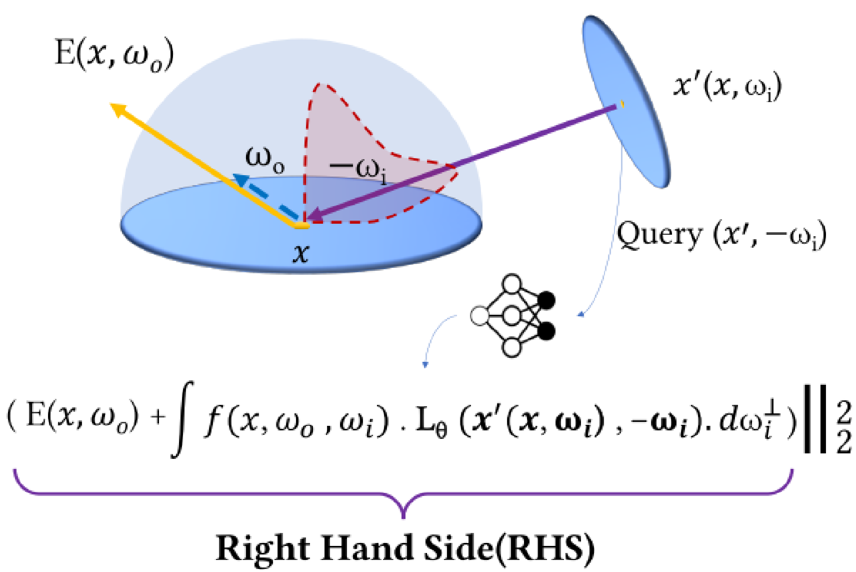
Neural Radiosity Method

- RHS: Outgoing radiance calculated from estimated incoming radiances
  - But we still have a rendering equation to solve…



Neural Radiosity Method

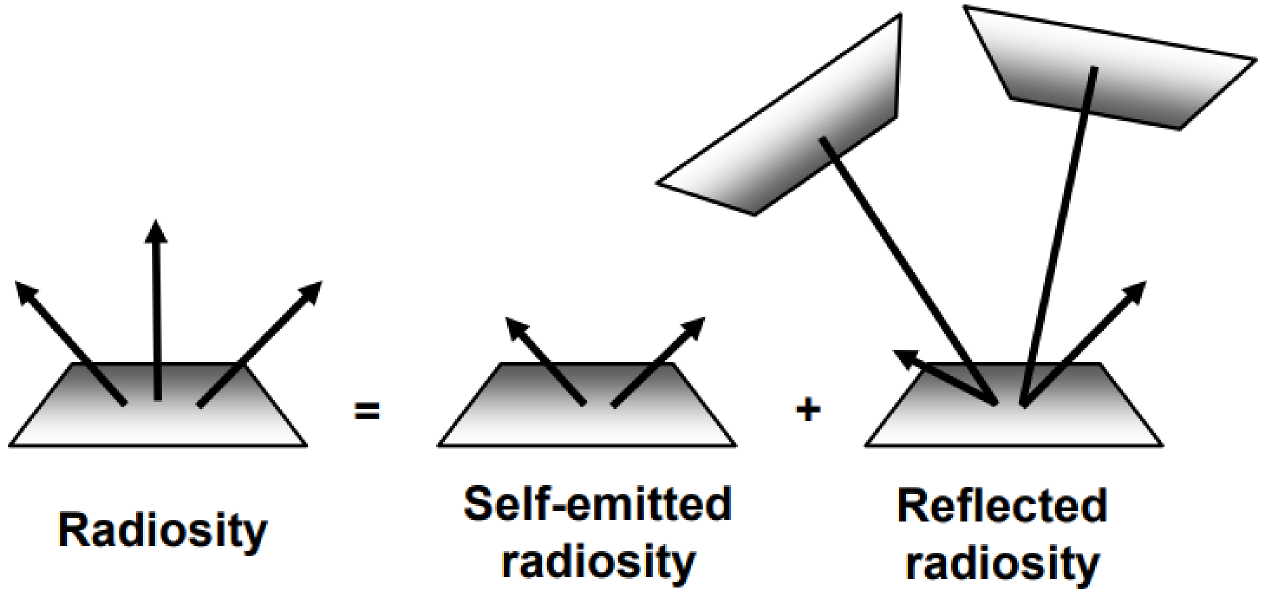
- RHS: Outgoing radiance calculated from estimated incoming radiances
  - Use Monte Carlo Integration!
  - Estimate the incoming radiance of the sampled $\omega_{i,k}, x'_k(x, \omega_{i,k})$



$$T\{L_\theta\}(x, \omega_o) = \frac{1}{M}\sum_{k=1}^{M}\frac{f(x, \omega_o, \omega_{i,k}) L_\theta(x'_k(x, \omega_{i,k}), -\omega_{i,k})}{p(\omega_{i,k})}$$

$$M \sim 16$$

Neural Radiosity Method



$$Radiosity_i = Radiosity_{self,i} + \sum_{j=1}^{N} a_{j\to i} Radiosity_j$$

$$\color{red}{r_\theta(x,\omega_o)} = L_\theta(x,\omega_o) - L_e(x,\omega_o) + \frac{1}{M}\sum_{k=1}^{M} \frac{f(x,\omega_o,\omega_{i,k}) L_\theta(x'_k(x,\omega_{i,k}), -\omega_{i,k})}{p(\omega_{i,k})}$$

Reducing the Residual Norm

- Residual norm $r_\theta(x, \omega_o)$

$$
\begin{aligned}
&r_\theta(x, \omega_o) \\
&= L_\theta(x, \omega_o) - L_e(x, \omega_o) - \frac{1}{M}\sum_{k=1}^{M} \frac{f(x, \omega_o, \omega_{i,k}) L_\theta\left(x'_k(x, \omega_{i,k}), -\omega_{i,k}\right)}{p(\omega_{i,k})} \\
&= L_\theta(x, \omega_o) - L_e(x, \omega_o) - T\{L_\theta\}(x, \omega_o)
\end{aligned}
$$

- $Loss(\theta) = \|r_\theta(x, \omega_o)\|^2$
- $Relative\ Loss(\theta) = \left\| \frac{r_\theta(x,\omega_o)}{sg(m_\theta(x,\omega_o))+\varepsilon} \right\|_2^2$
  - For a stable training with high dynamic range radiances
  - $m_\theta(x, \omega_o) = \frac{1}{2}\left(L_\theta(x, \omega_o) + L_e(x, \omega_o) + T\{L_\theta\}(x, \omega_o)\right)$
  - sg: stop gradient

Training with Neural Radiosity

- Now, we do not need to directly solve/approximate the rendering equation!

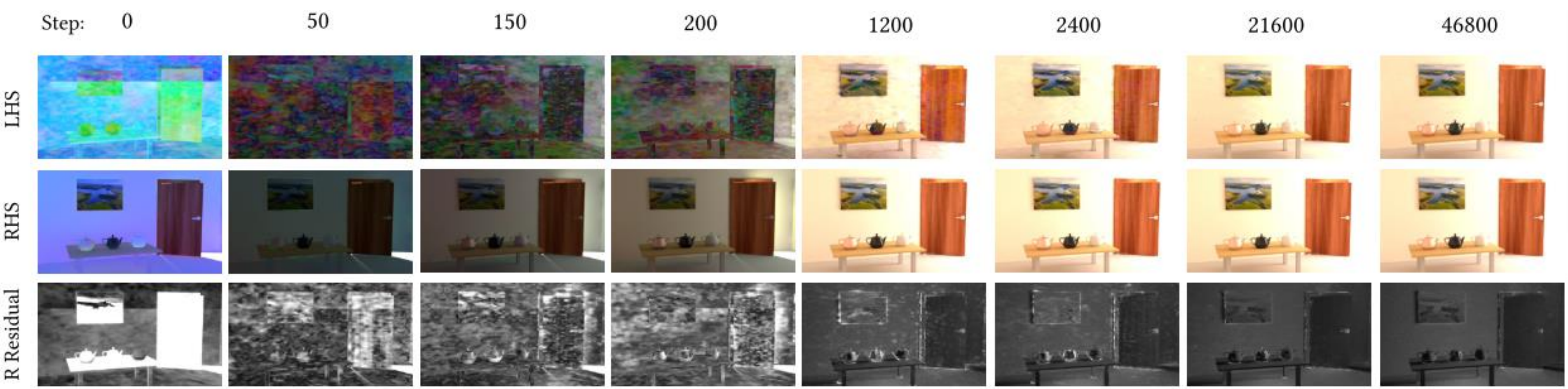
ALGORITHM 1: Minibatch stochastic gradient descent, learning rate η.

---

initialize network parameters θ;
while *not converged* **do**
    sample a set of surface points $\{x_j | j = 1 \dots N\}$ and outgoing directions $\{\omega_{o,j} | j = 1 \dots N\}$;
    for each $(x_j, \omega_{o,j})$, sample a set of incident directions $\{\omega_{i,j,k} | k = 1 \dots M\}$;
    use the samples to evaluate the Monte Carlo estimate of $\nabla_\theta \mathcal{L}(\theta)$ using Equations 6 and 8;
    $\theta = \theta - \eta \nabla_\theta \mathcal{L}(\theta)$;
end
return θ;

Training with Neural Radiosity

- **Training takes more time that Path Tracing**
  - 3~5 minutes per 1000 steps…
- But shows various applications once trained…



Main Contribution

Solving the Rendering Equation by

Radiance-predicting Neural Network

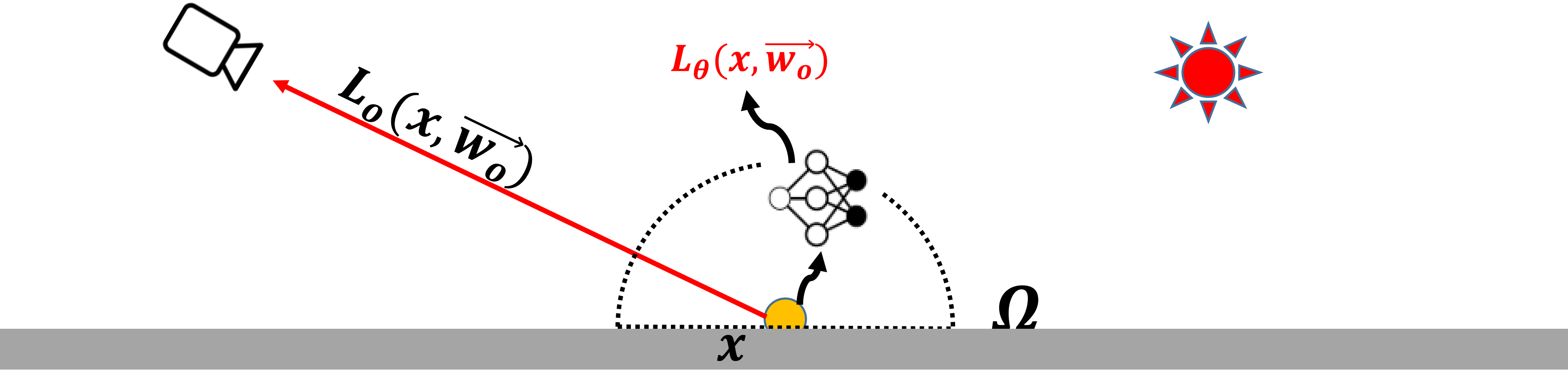
via Radiosity-like Training

Radiance-predicting Neural Network

$$L_o(x, \overrightarrow{w_o}) = L_e(x, \overrightarrow{w_o}) + \int_{\Omega} f_r(x, \overrightarrow{w_i}, \overrightarrow{w_o}) L_i(x, \overrightarrow{w_i})(\overrightarrow{w_i} \cdot \vec{n}\,) d\overrightarrow{w_i}$$

$$\sim L_e(x, \overrightarrow{w_o}) + L_\theta(x, \overrightarrow{w_o})$$

Solving rendering equation via Radiance-predicting Neural Network L_θ



Positional Encoding is not Enough

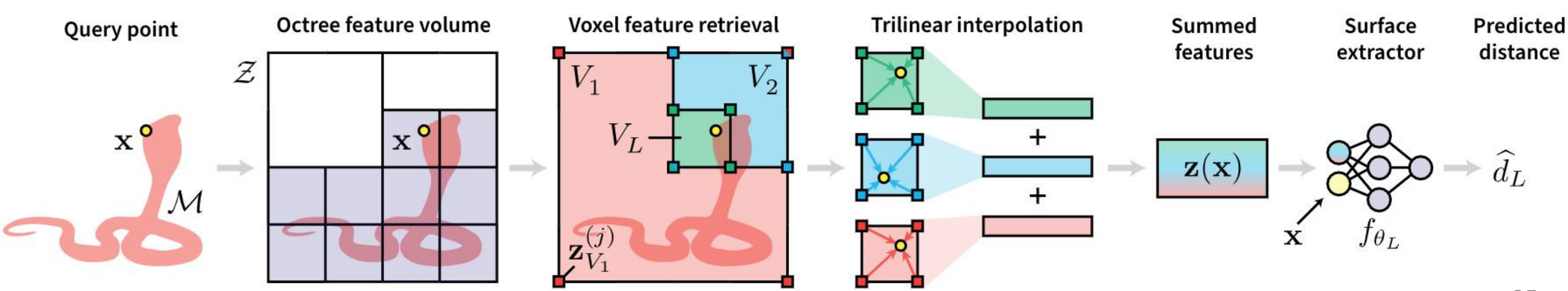
- Positional encoding like ***NeRF*** does not show better performance

$$L_\theta(x, \omega_o) = MLP\begin{pmatrix} x \\ \gamma(x) \\ \omega_o \end{pmatrix}, \quad \gamma(x) = \begin{pmatrix} \sin(2^0 \pi x) \\ \cos(2^0 \pi x) \\ \vdots \\ \sin(2^{k-1} \pi x) \\ \cos(2^{k-1} \pi x) \end{pmatrix}$$

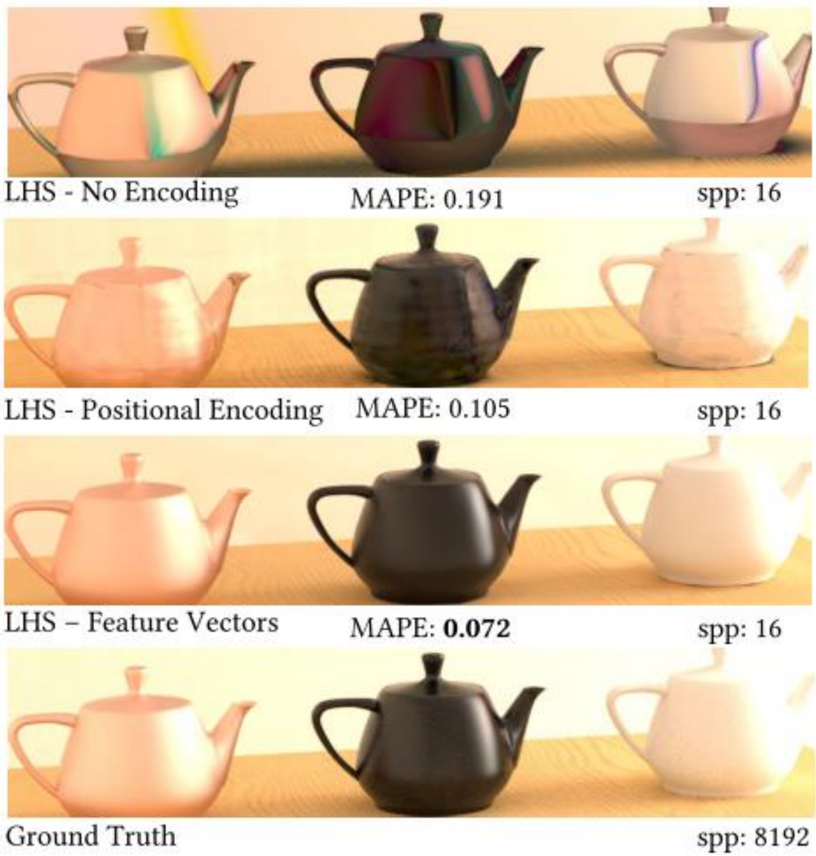
- Instead, use a multi-resolution feature grid with trainable features!
  - Similar approach with ***Plenoxels***, but with more scale

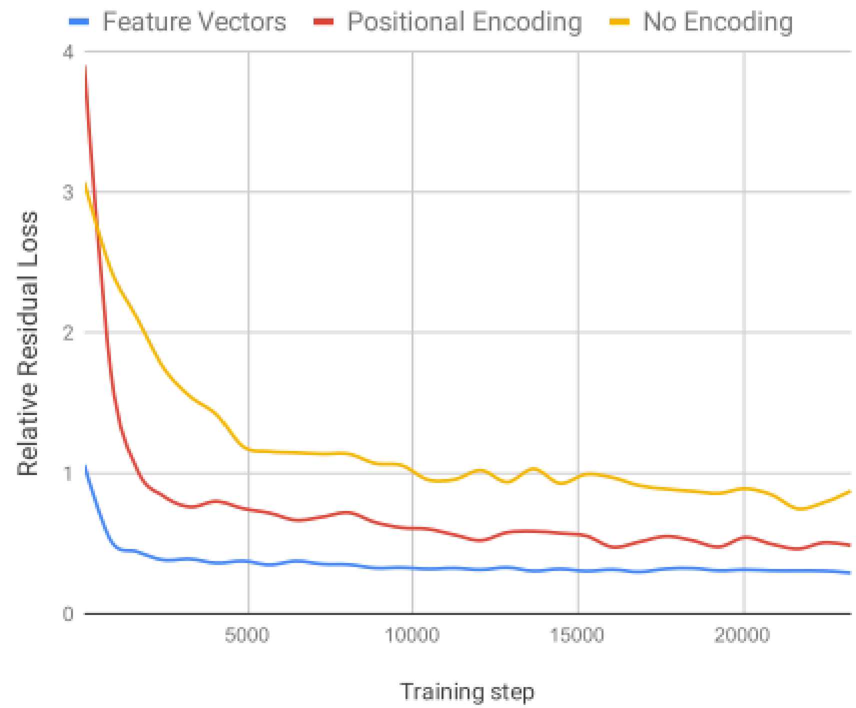
Multi-resolution Feature Grid

- Idea & Implementation borrowed from NGLOD
  - Neural Geometric Level of Detail, CVPR 2021
- Features of the query point as interpolated feature vectors of each level of voxel grids
- Allows better performance with using relatively shallow network

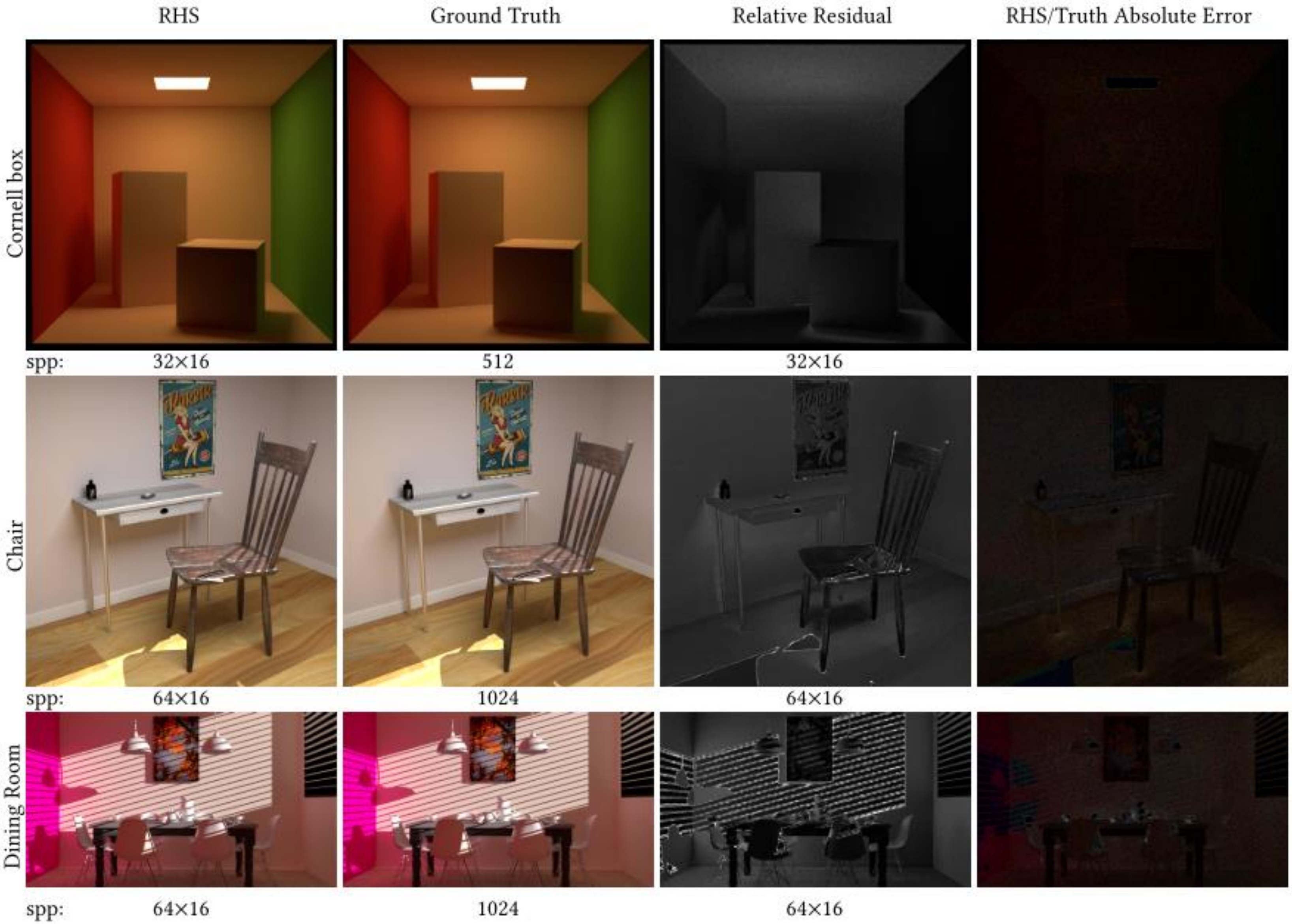


Multi-resolution Feature Grid



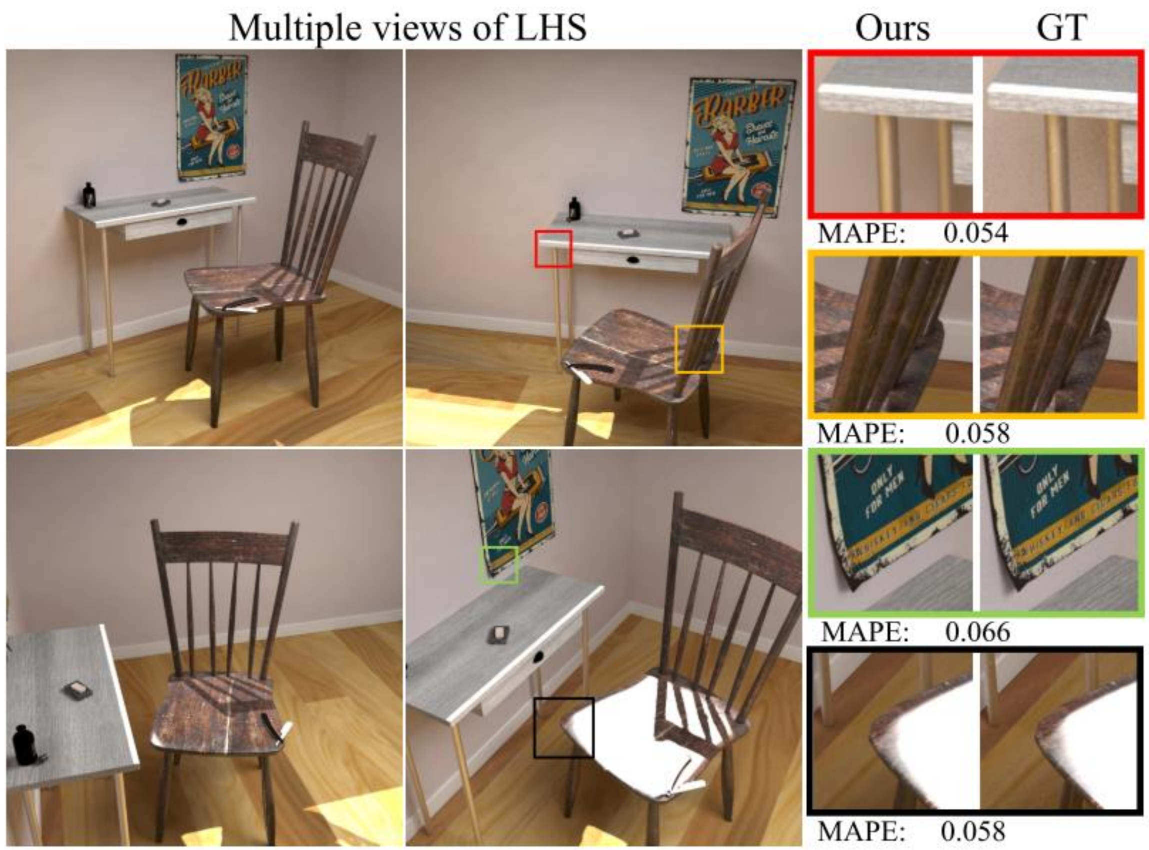


Results – Rendering



Results – View Synthesis

- Trained network represents the entire radiance distribution of the scene → Multi-view Synthesis!

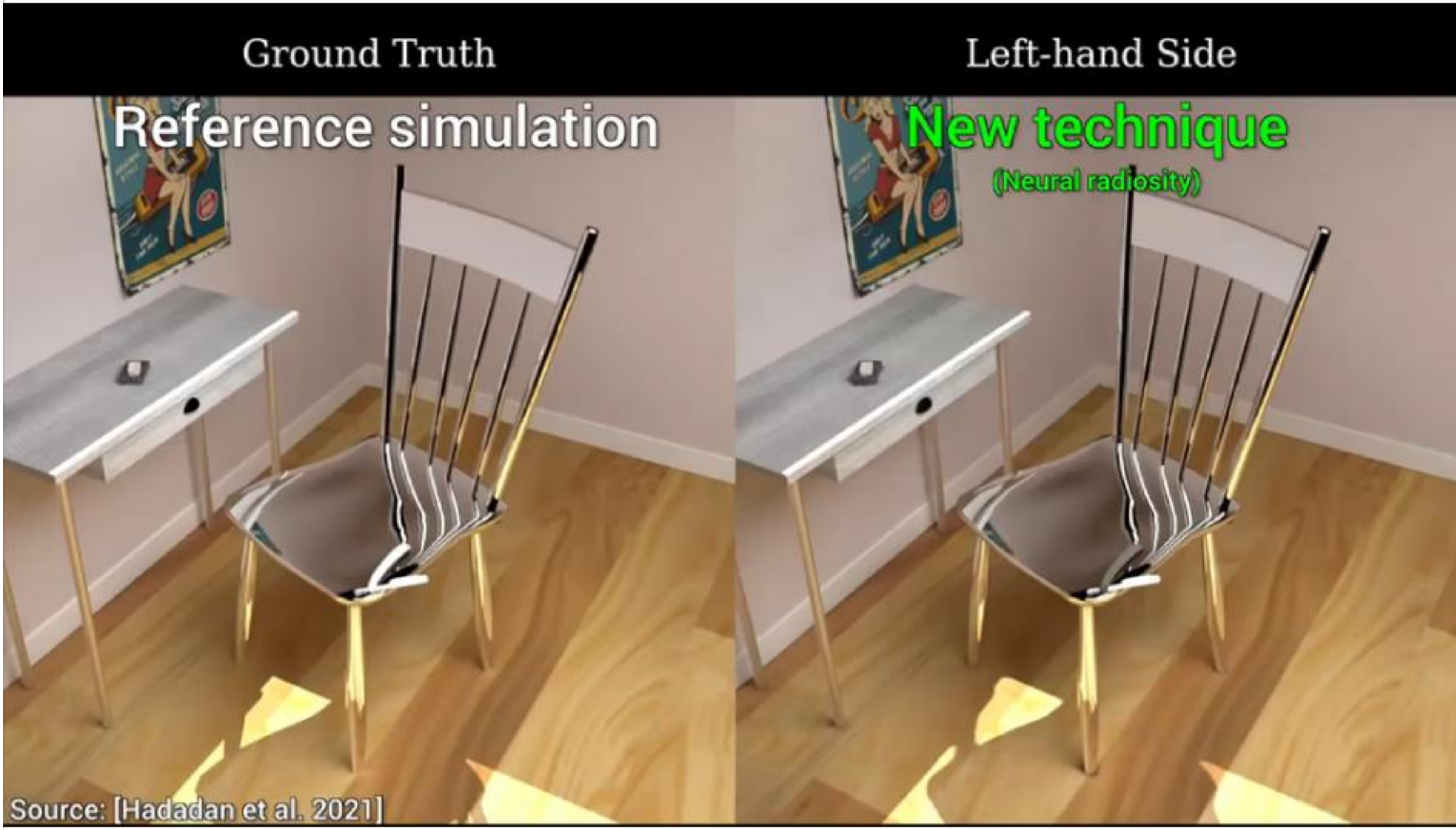


https://www.youtube.com/watch?v=cwS_Fw4u0rM

Results – Material Support

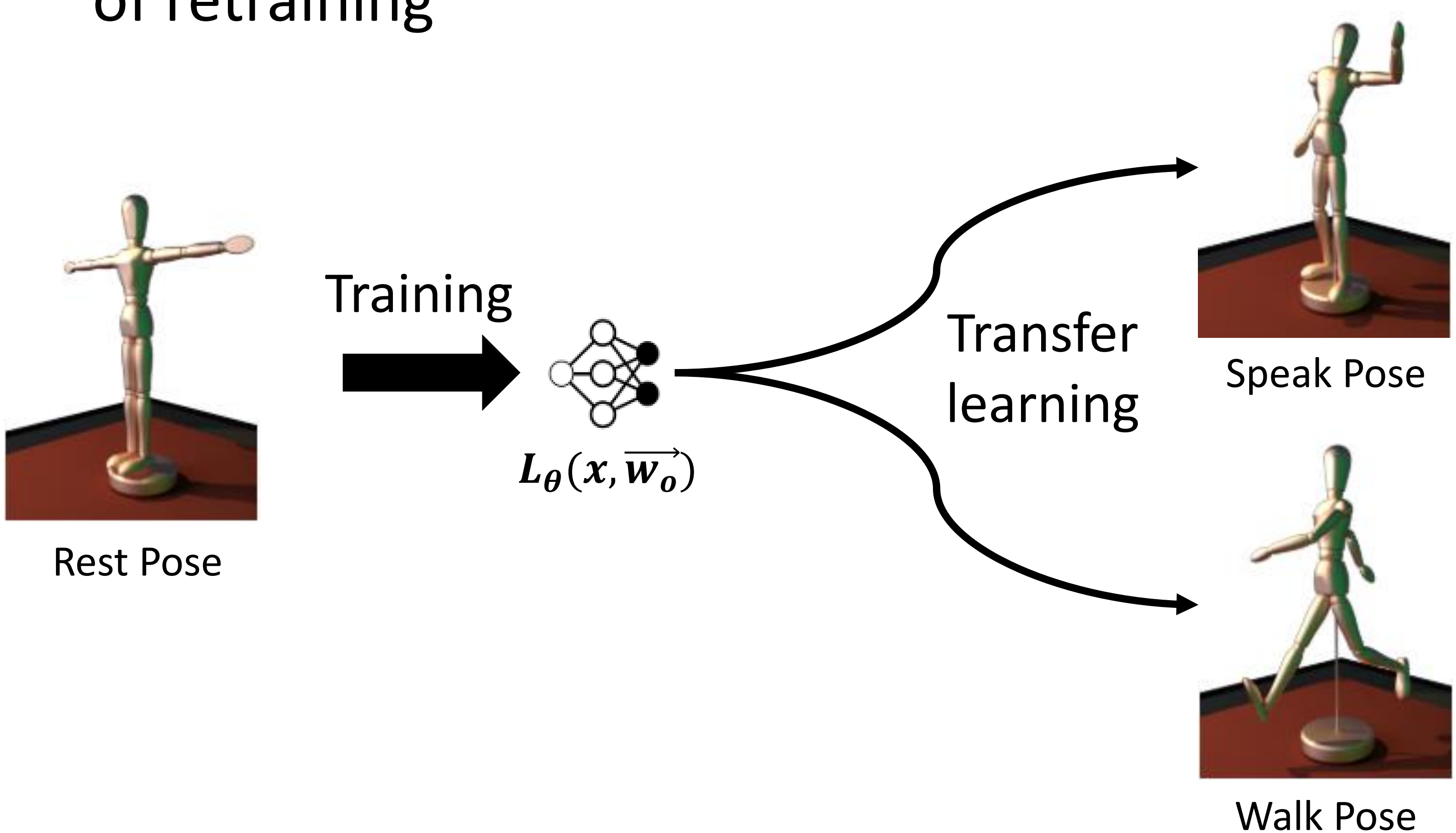
- Good quality for various materials
  - Note that original radiosity method only supported diffuse effects!

https://www.youtube.com/watch?v=cwS_Fw4u0rM



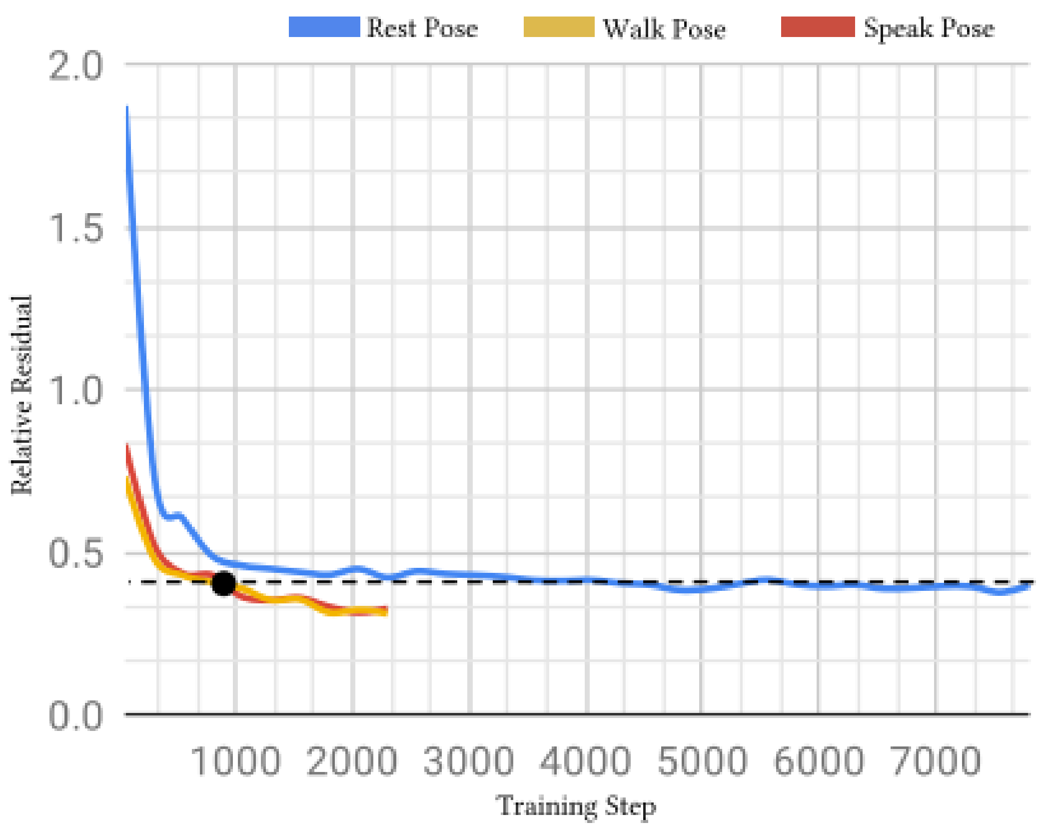
Results – Dynamic Scenes

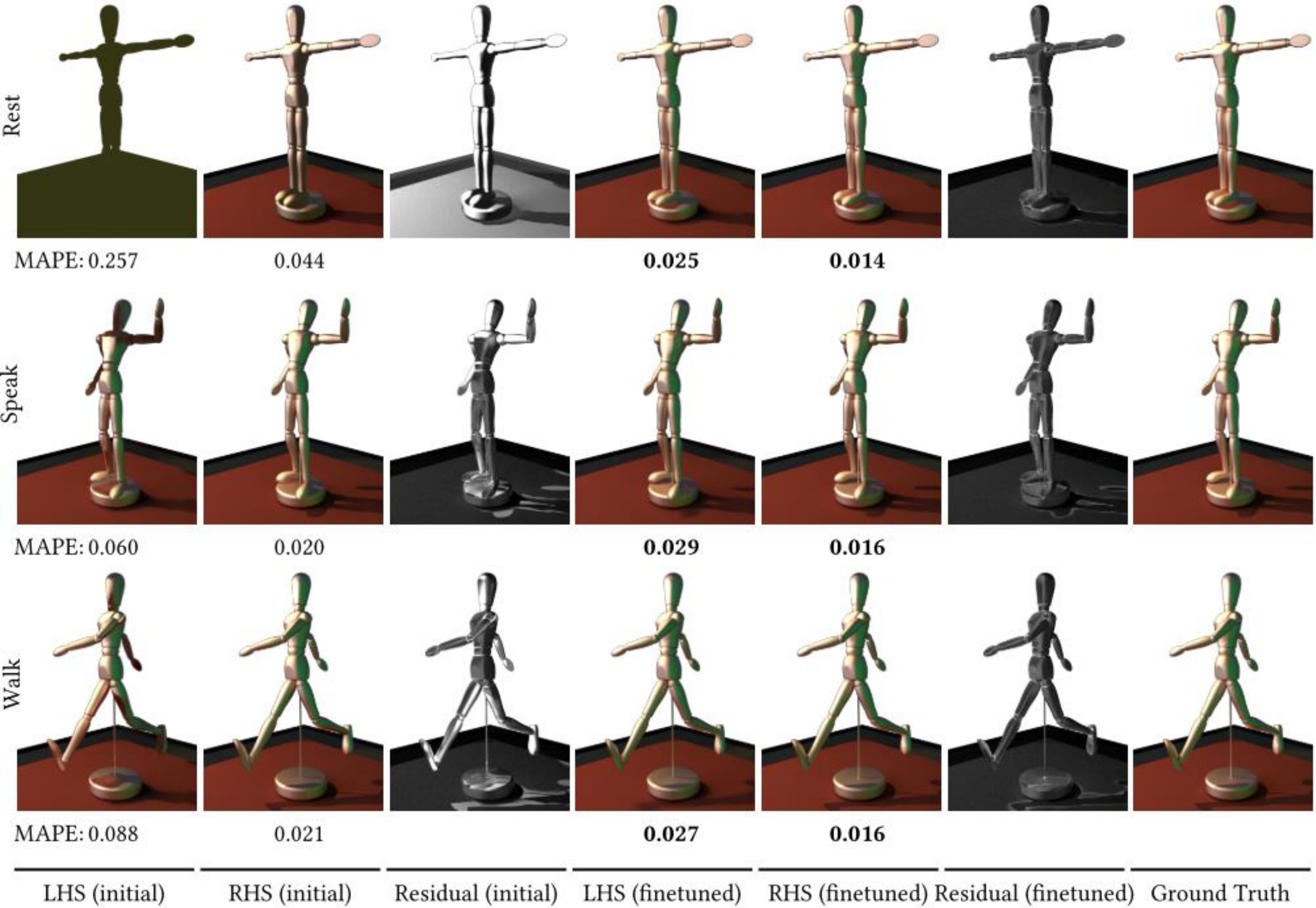
- Apply transfer learning for dynamic scenes instead of retraining



Results – Dynamic Scenes

- Apply transfer learning for dynamic scenes instead of retraining




Rest
Speak
Walk
MAPE: 0.257
0.044
0.025
0.014
MAPE: 0.060
0.020
0.029
0.016
MAPE: 0.088
0.021
0.027
0.016
LHS (initial)
RHS (initial)
Residual (initial)
LHS (finetuned)
RHS (finetuned)
Residual (finetuned)
Ground Truth

Neural Radiosity: Wrap-up

- A radiosity-like training to learn the entire radiance distribution of the scene

- Multi-resolution feature grid for new positional encoding

- Applied to multi-view synthesis, rendering dynamic scenes via transfer learning

Real-time
Neural Radiance Caching
for Path Tracing

Muller et al. SIGGRAPH 2021

Main Contributions

Radiance Caching with Neural Radiance Field

Self-training with Fast Adaptation

Other Techniques for Real-time Path Tracing

Main Contributions

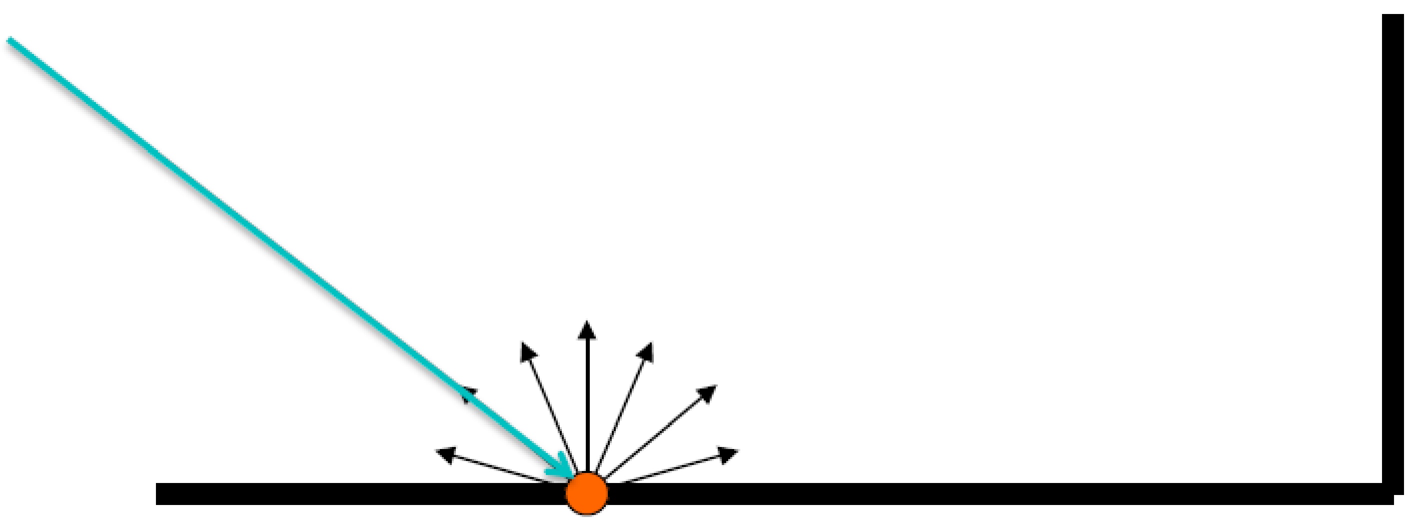
Radiance Caching with Neural Radiance Field

Self-training with Fast Adaptation

Other Techniques for Real-time Path Tracing

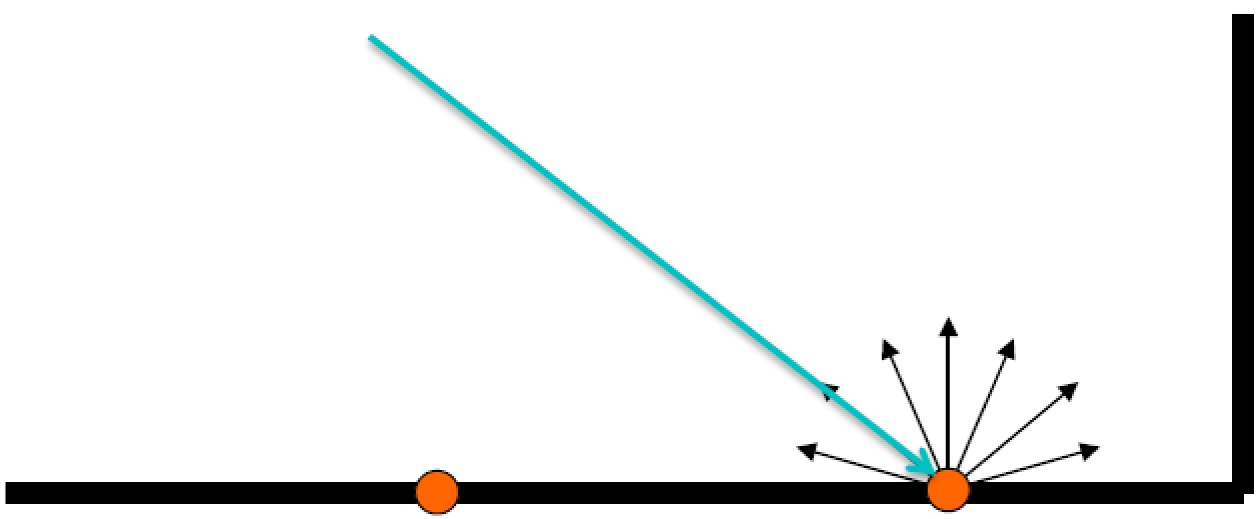
Irradiance Caching: Recap

- **Biased GI algorithm**
- Cache the irradiance of the point



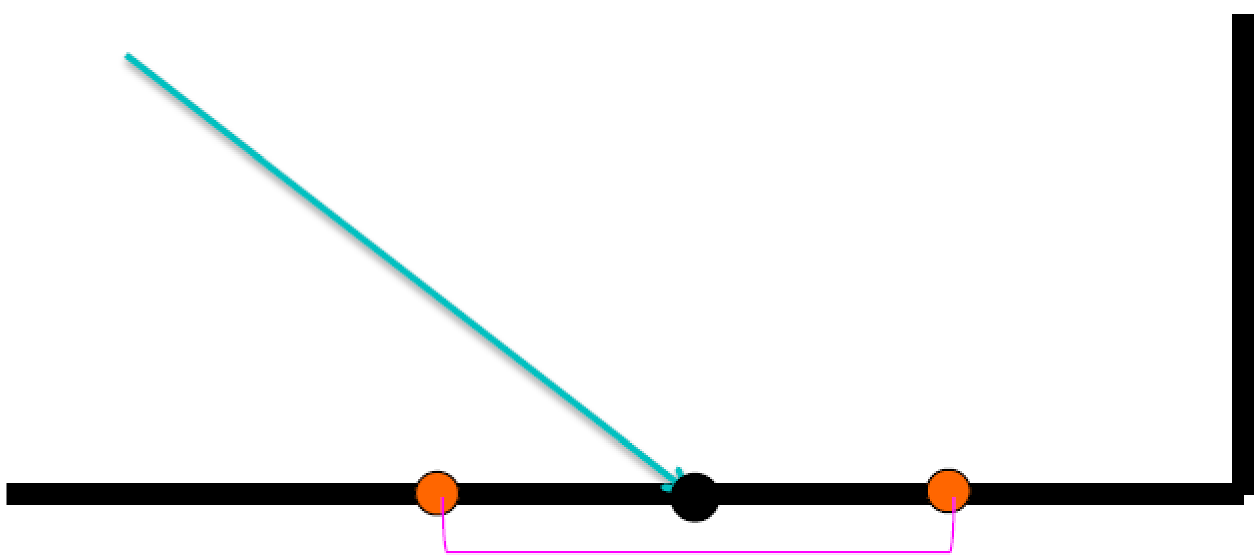
Irradiance Caching: Recap

- **Biased GI algorithm**
- Cache the irradiance of the point



Irradiance Caching: Recap

- **Biased GI algorithm**
- Cache the irradiance of the point
- Interpolate the irradiance of the query point

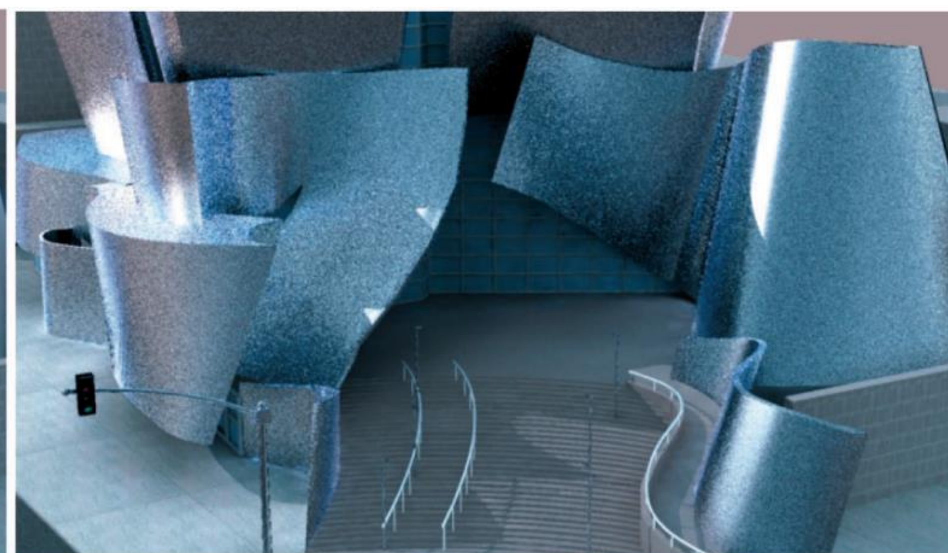


Radiance Caching for Efficient Global Illumination Computation, Krivanek et al., TVCG 2005
Plenoxels: Radiance Fields without Neural Networks, Yu et al., CVPR 2022

Radiance Caching

- Adding a directional information for caching
- Use Spherical Harmonics H_l^m like ***Plenoxels***
  - $L_i(\theta, \phi) \approx \sum_{l=0}^{n-1} \sum_{m=-l}^{l} \lambda_l^m H_l^m(\theta, \phi)$
- Interpolate the coefficients λ_l^m

Radiance caching



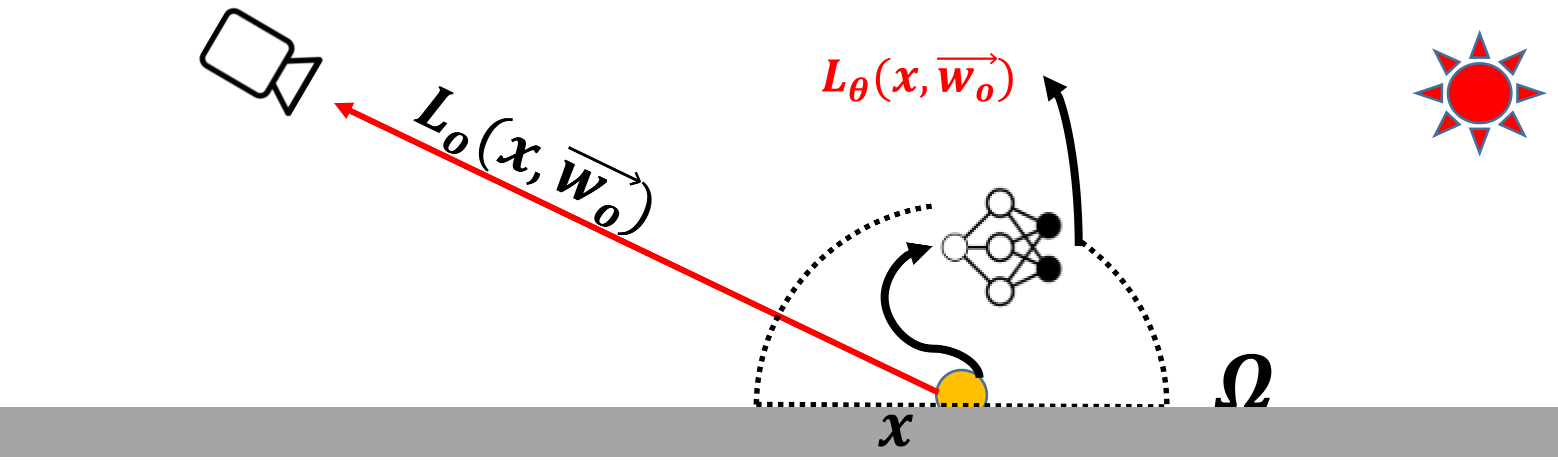
Monte Carlo sampling

Neural Radiance Cache

$$L_o(x, \overrightarrow{w_o}) = L_e(x, \overrightarrow{w_o}) + \int_\Omega f_r(x, \overrightarrow{w_i}, \overrightarrow{w_o}) L_i(x, \overrightarrow{w_i})(\overrightarrow{w_i} \cdot \vec{n}\,) d\overrightarrow{w_i}$$

$$\sim L_e(x, \overrightarrow{w_o}) + L_\theta(x, \overrightarrow{w_o})$$

Solving rendering equation via Radiance-predicting Neural Network L_θ

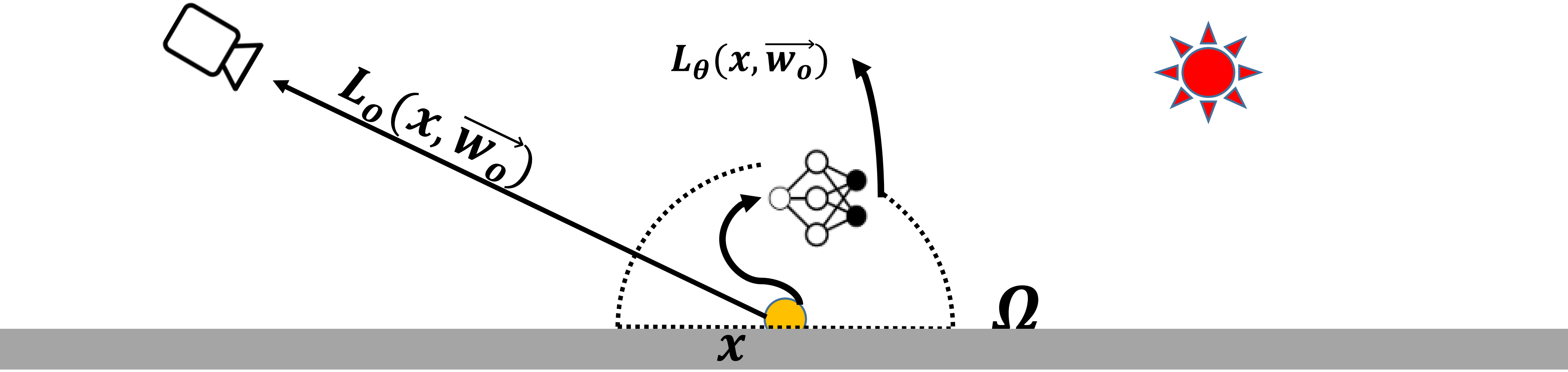


Neural Radiance Cache

$$L_o(x,\overrightarrow{w_o}) = L_e(x,\overrightarrow{w_o}) + \int_{\Omega} f_r(x,\overrightarrow{w_i},\overrightarrow{w_o}) L_i(x,\overrightarrow{w_i})(\overrightarrow{w_i}\cdot\vec{n}\,)d\overrightarrow{w_i}$$

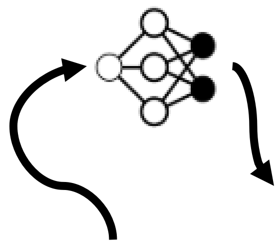
$$\sim L_e(x,\overrightarrow{w_o}) + L_\theta(x,\overrightarrow{w_o})$$

Train the neural network → Cache, Estimate the radiance → Interpolate



NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, Mildenhall et al., ECCV 2020
Neural Importance Sampling, Muller et al., SIGGRAPH 2019

Neural Radiance Cache



Radiance

Parameter	Symbol	with Encoding
Position	$\mathbf{x} \in \mathbb{R}^3$	$\mathrm{freq}(\mathbf{x}) \in \mathbb{R}^{3\times 12}$
Scattered dir.	$\omega \in S^2$	$\mathrm{ob}(\mathrm{sph}(\omega)) \in \mathbb{R}^{2\times 4}$
Surface normal	$\mathbf{n}(\mathbf{x}) \in S^2$	$\mathrm{ob}(\mathrm{sph}(\mathbf{n}(\mathbf{x}))) \in \mathbb{R}^{2\times 4}$
Surface roughness	$r(\mathbf{x}, \omega) \in \mathbb{R}$	$\mathrm{ob}\left(1 - e^{-r(\mathbf{x},\omega)}\right) \in \mathbb{R}^4$
Diffuse reflectance	$\alpha(\mathbf{x}, \omega) \in \mathbb{R}^3$	$\mathrm{id}(\alpha(\mathbf{x}, \omega)) \in \mathbb{R}^3$
Specular reflectance	$\beta(\mathbf{x}, \omega) \in \mathbb{R}^3$	$\mathrm{id}(\beta(\mathbf{x}, \omega)) \in \mathbb{R}^3$

Inputs for Neural Radiance Cache

$$freq(x) = \begin{pmatrix} \sin(2^0 \pi x) \\ \cos(2^0 \pi x) \\ \vdots \\ \sin(2^{k-1} \pi x) \\ \cos(2^{k-1} \pi x) \end{pmatrix}$$

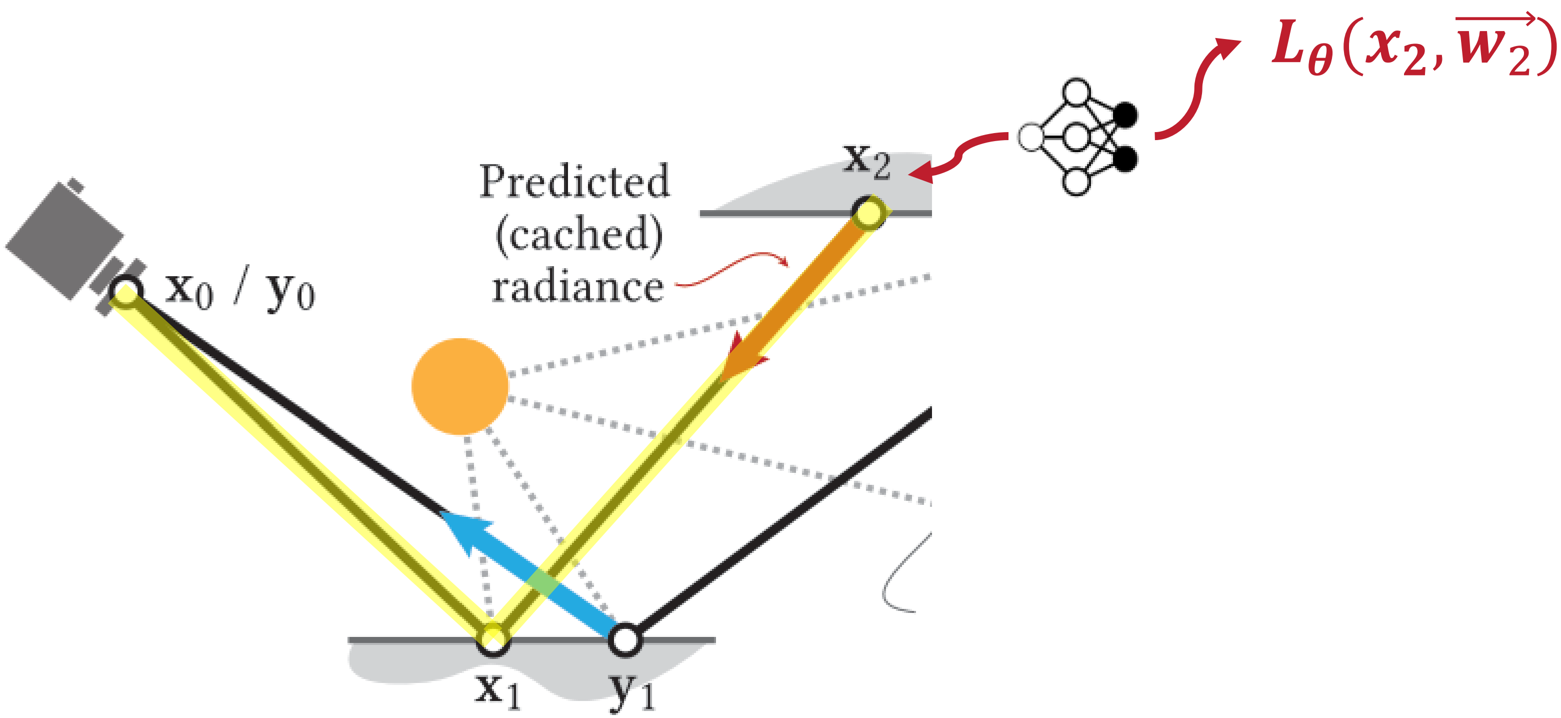
Positional Encoding from ***NeRF***

$$ob(x) = Gaussian(x, \frac{1}{k})$$

One-blob Encoding from
Neural Importance Sampling

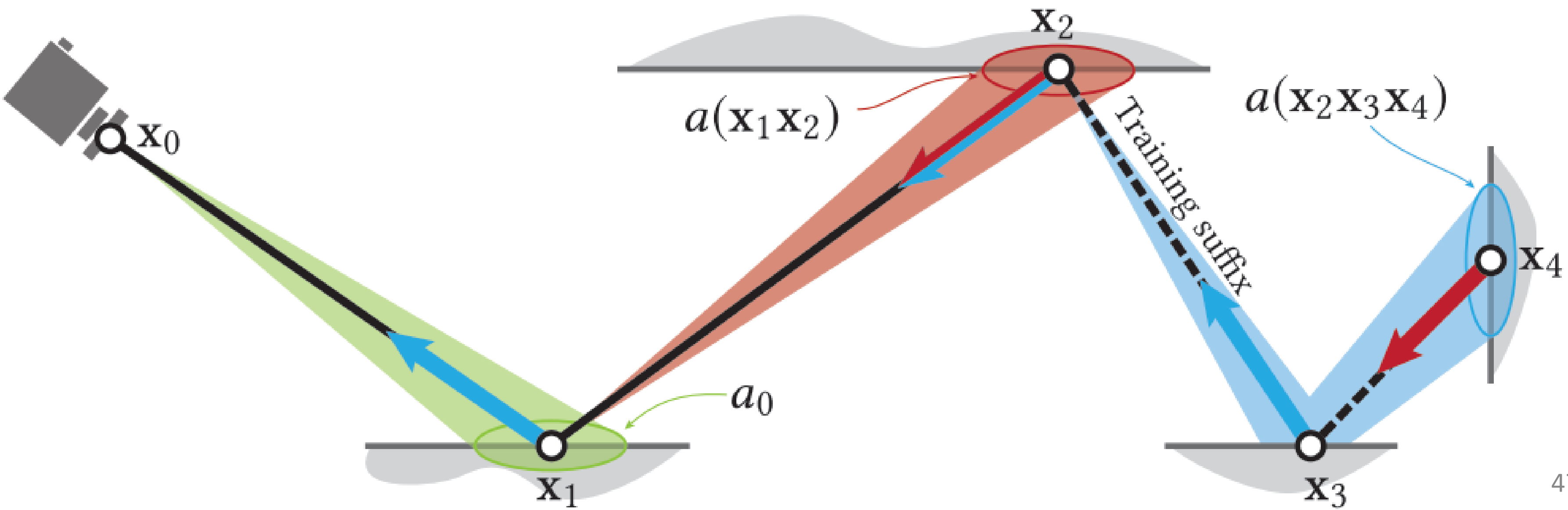
Rendering with Neural Radiance Caching

- Trace a short rendering path ($x_0 x_1 x_2$) where we used the cached(estimated) radiance in vertex x_2 $\boldsymbol{L_\theta(x_2, \overrightarrow{w_o})}$
  - When do we terminate?



Rendering with Neural Radiance Caching

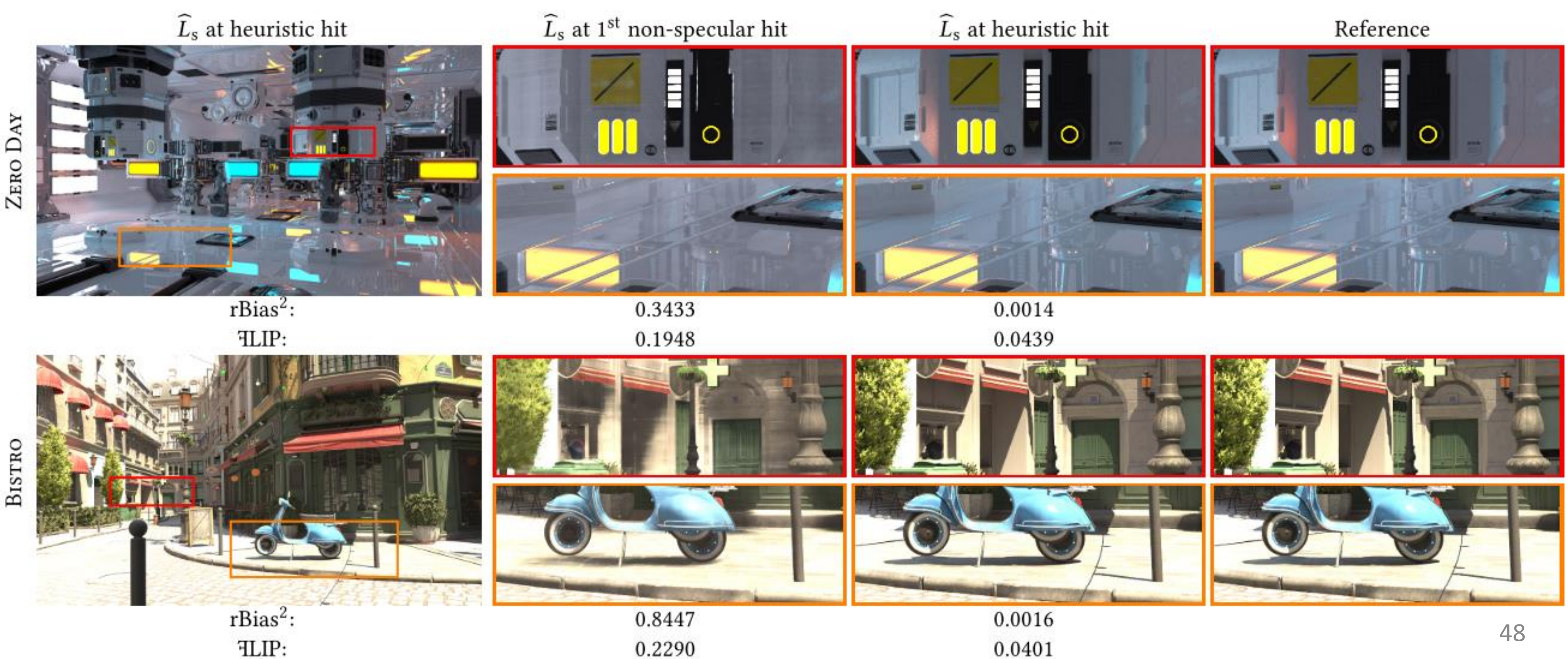
- Terminate when the **area spread** $\boldsymbol{a(x_1 \dots x_n)}$ becomes large enough to blur the inaccuracy in trained cache $\boldsymbol{a_0}$
  - $a(x_1 \dots x_n) > c \cdot a_0$
  - $a_0 = \frac{\|x_0 - x_1\|^2}{4\pi cos\theta_1}, a(x_1 \dots x_n) = \left(\sum_{i=2}^{n} \sqrt{\frac{\|x_0 - x_1\|^2}{p(\omega_i | x_{i-1}, \omega) |cos\theta_i|}}\right)^2$



A Custom Designed Density Estimation Method for Light Transport, Bekaert et al., Techinical Report 2003

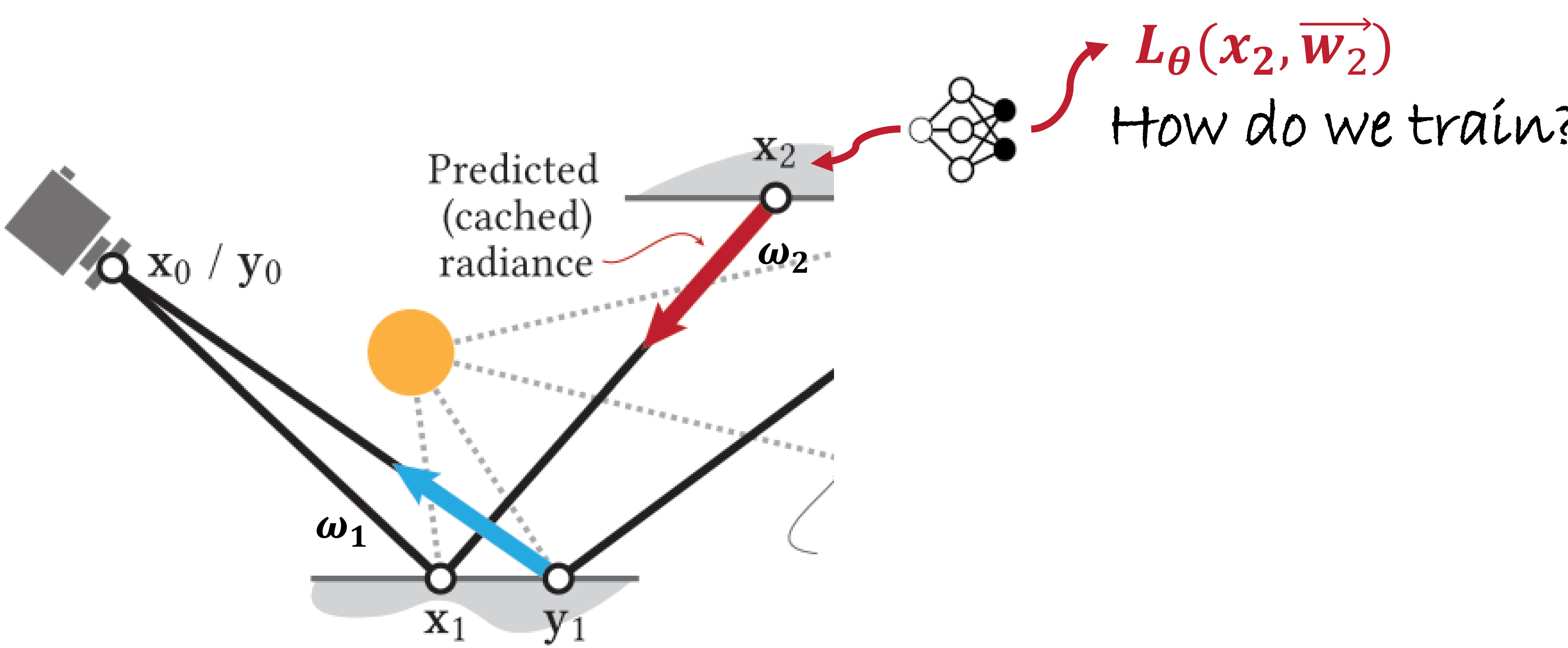
Rendering with Neural Radiance Caching

- Heuristic termination helps to avoid using poorly cached radiances in the primary hit vertex



Rendering with Neural Radiance Caching

- Caculate the radiance using the estimated radiance
  - $L(x_1, \omega_1) = L_e(x_2, \omega_2) + \frac{L_\theta(x_2,\omega_2) f(x_1,w_2,w_1) \cos(\omega_1 \cdot n_1)}{p(-\omega_2)}$



Main Contributions

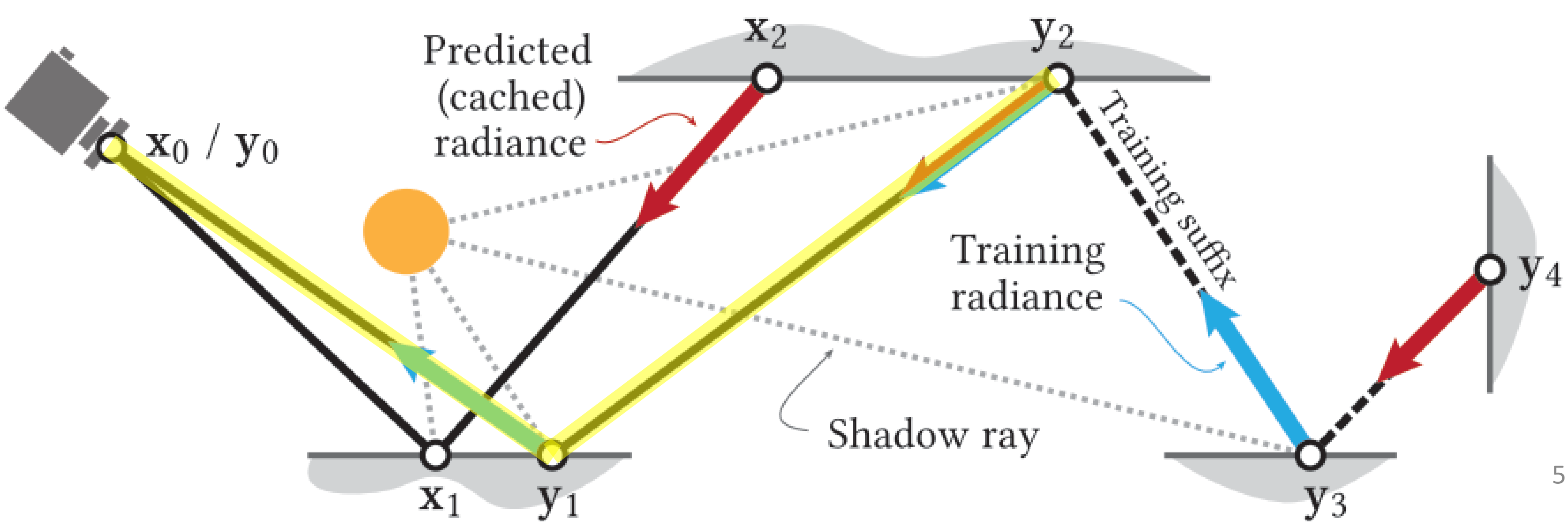
Radiance Caching with Neural Radiance Field

Self-training with Fast Adaptation

Other Techniques for Real-time Path Tracing

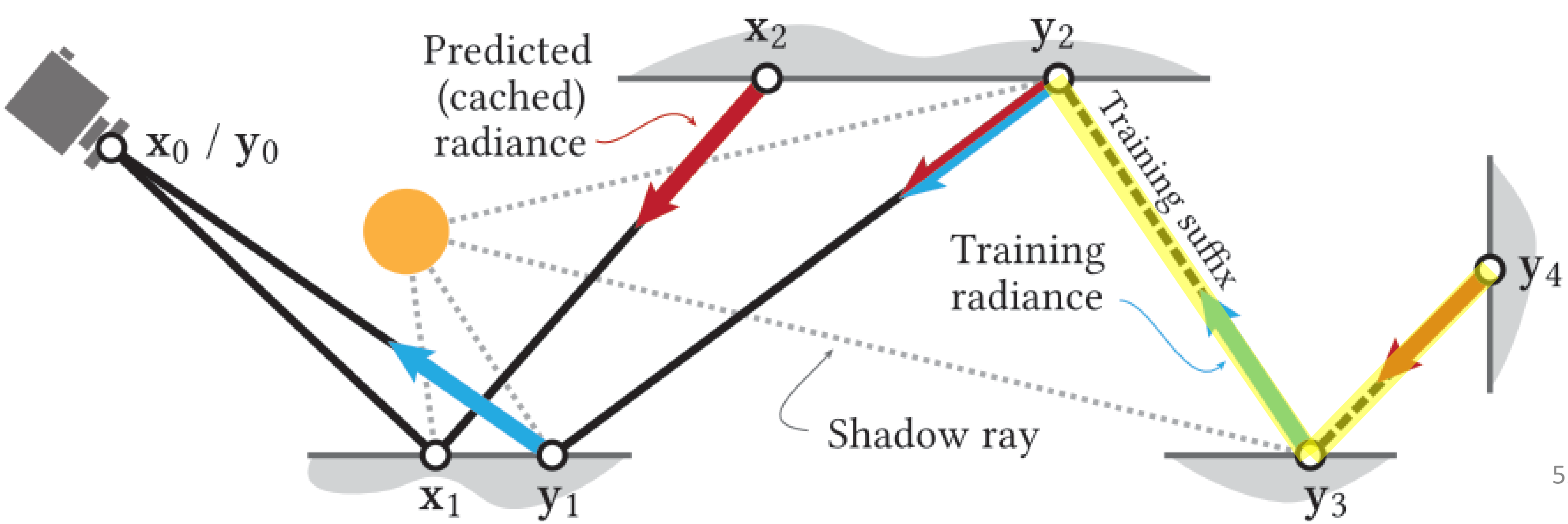
Self-training with Fast Adaptation

- Trace a short rendering path ($y_0 y_1 y_2$)
- Estimate the radiance of y_2 and calculate the radiance the sample



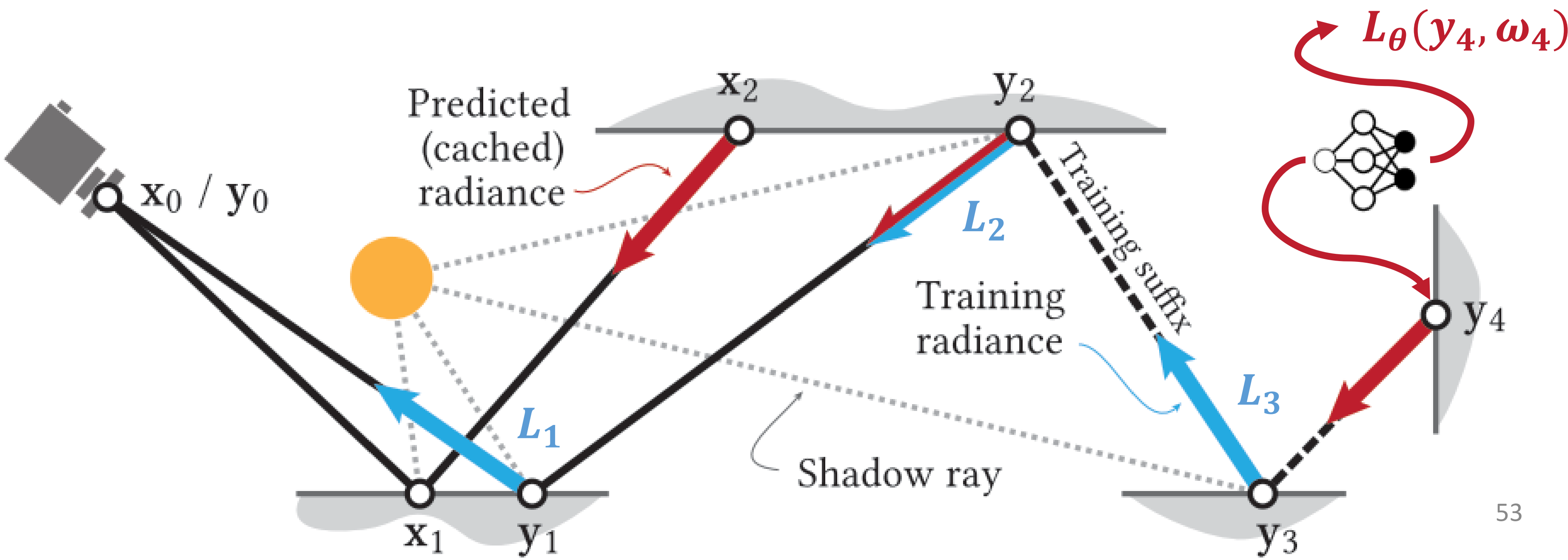
Self-training with Fast Adaptation

- Extend the rendering path with few vertices ($\cdots y_2 y_3 y_4$)
- **We use the same light sample for rendering & training!**



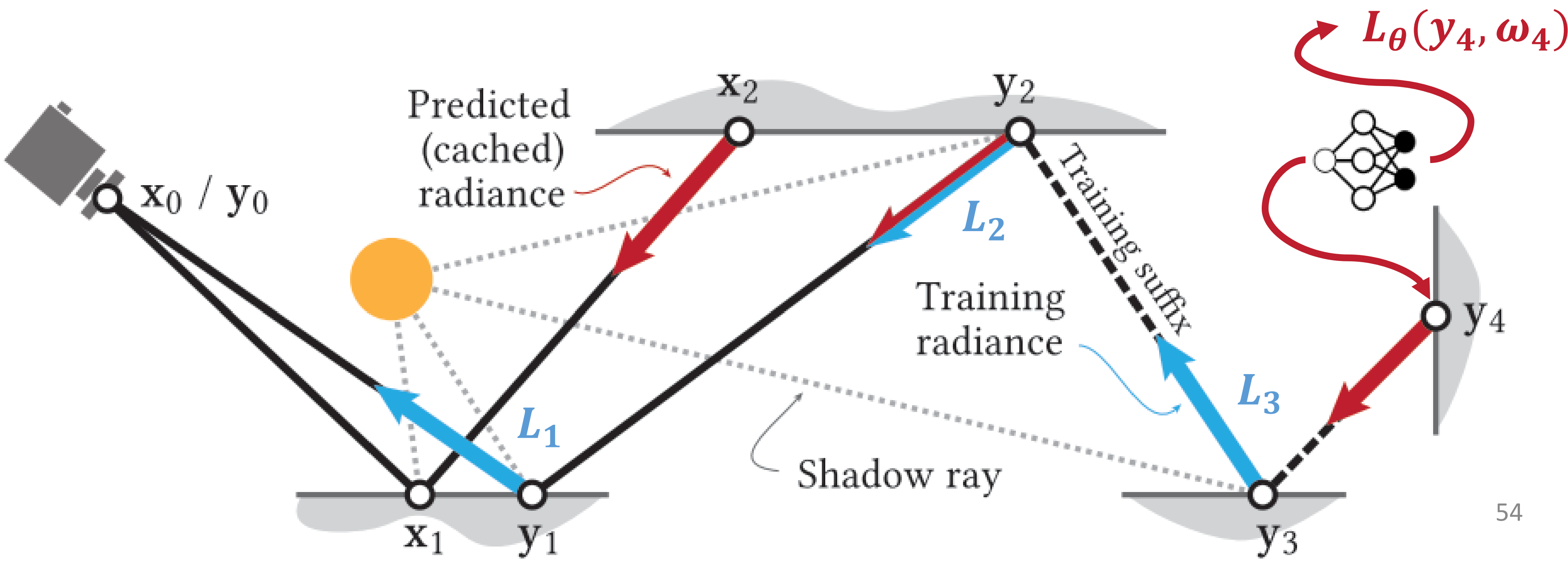
Self-training with Fast Adaptation

- Estimate the radiance in y_4: $\boldsymbol{L_\theta(y_4, \omega_4)}$
- Calculate the radiances on preceding vertices using the estimated radiance above
  - $\boldsymbol{L_1, L_2, L_3}$



Self-training with Fast Adaptation

- Minimize the loss between the calculated radiances and the estimated radiances of the preceding vertices
- $Loss = relL2\left(\boldsymbol{L_1}, \boldsymbol{L_\theta(y_1, \omega_1)}\right) + relL2\left(\boldsymbol{L_2}, \boldsymbol{L_\theta(y_2, \omega_2)}\right) + relL2\left(\boldsymbol{L_3}, \boldsymbol{L_\theta(y_3, \omega_3)}\right)$



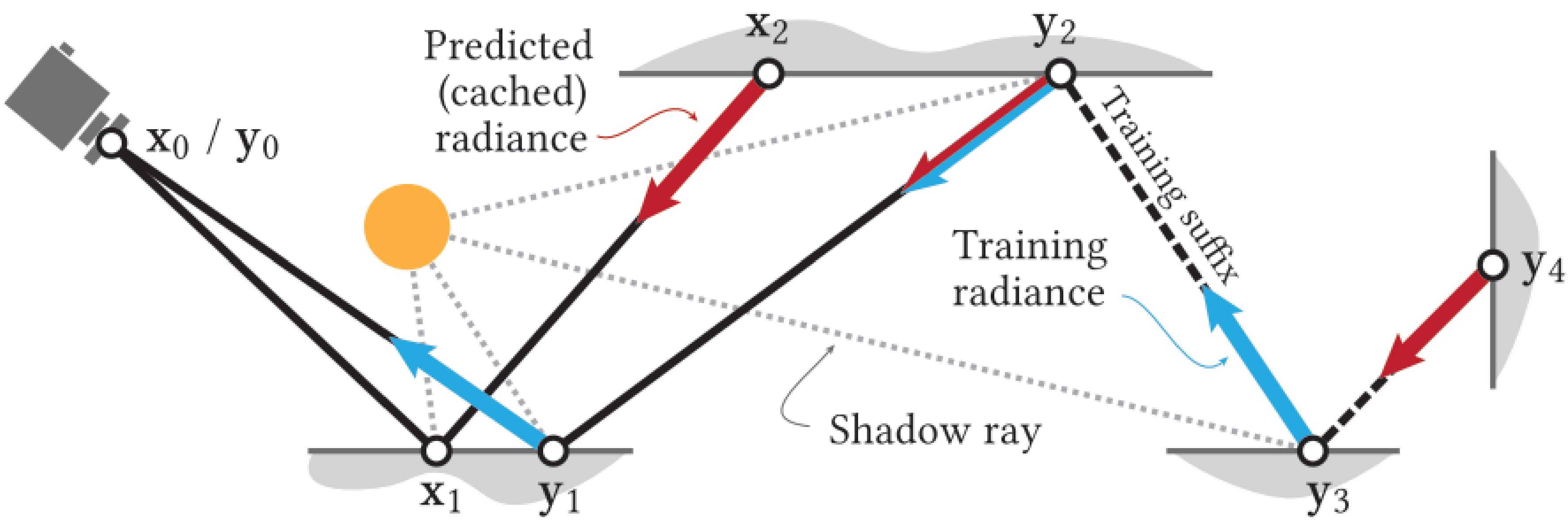
Self-training with Fast Adaptation

- No ground truth needed → **Self-training!**
  - Similar to Neural Radiosity

- High learning rate & Multiple gradient descent steps per frame with random subset of ray batches → **Fast Adaptation!**
  - One frame with 1spp, FHD → Batch size 2^{12}
  - Iteratively done for each frames

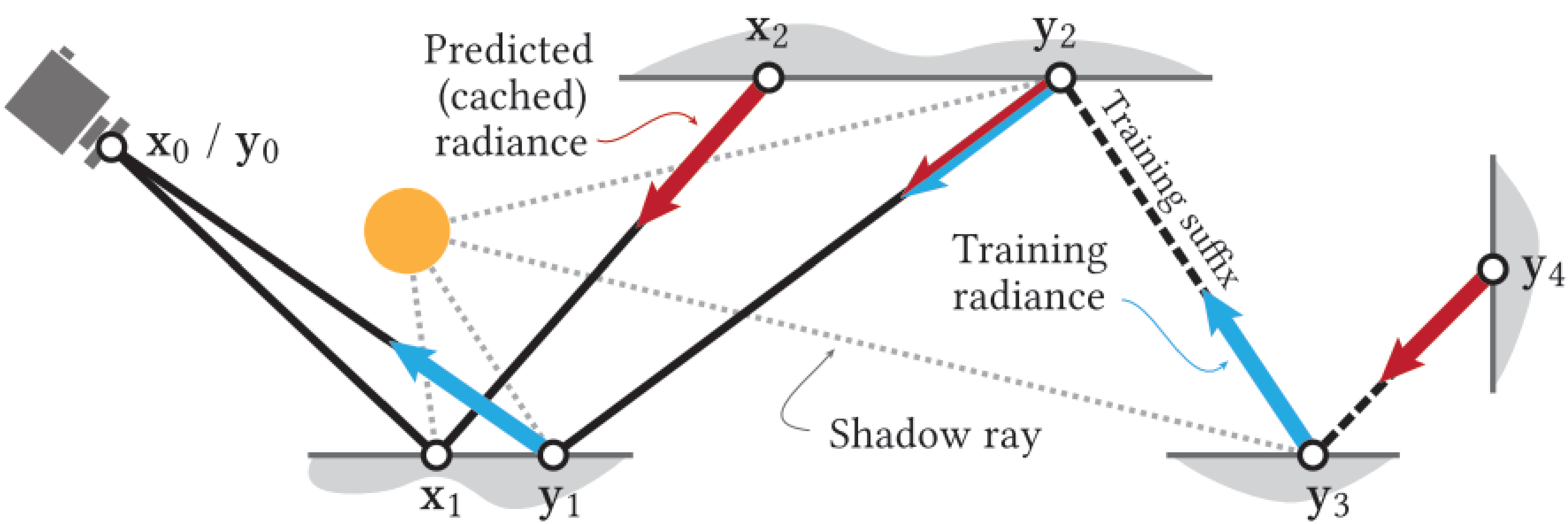
Limitations of Self-training

- What if the training path hits the surface never reached?
  - Results in unstable training…
- What if the extended vertices are close together?
  - Cannot complete cover the global illumination effect



Limitations of Self-training

- For balancing two issues, extend every N^{th} sample which is terminated by Russian roulette.
  - Can construct more unbiased training paths



Main Contributions

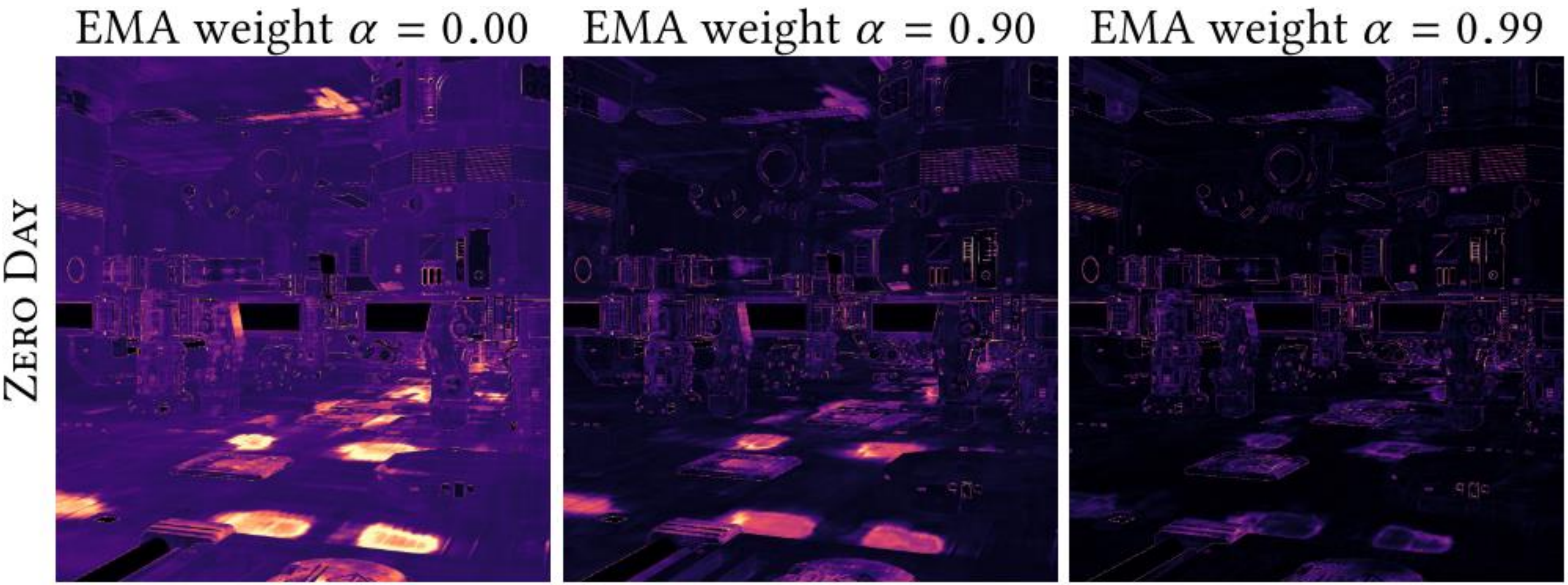
Radiance Caching with Neural Radiance Field

Self-training with Fast Adaptation

Other Techniques for Real-time Path Tracing

Temporal Stability via EMA (Exponential Moving Average)

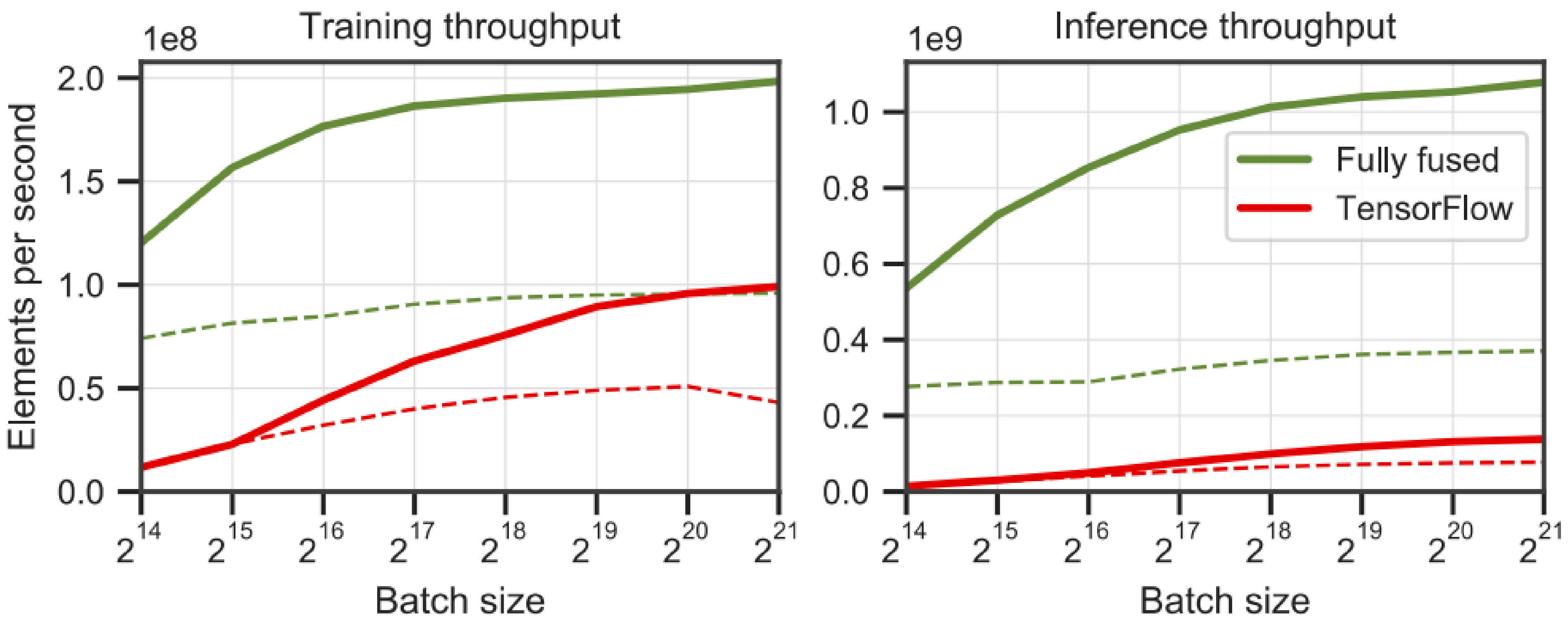
- Aggressive fast adaptation strategy might lead to overfitting, creating temporal artifacts like flickering
- $\bar{W}_t = \frac{1-\alpha}{\eta_t} \cdot W_t + \alpha \cdot \eta_{t-1} \cdot \bar{W}_{t-1}, \eta_t = 1 - \alpha^t$



https://github.com/NVlabs/tiny-cuda-nn

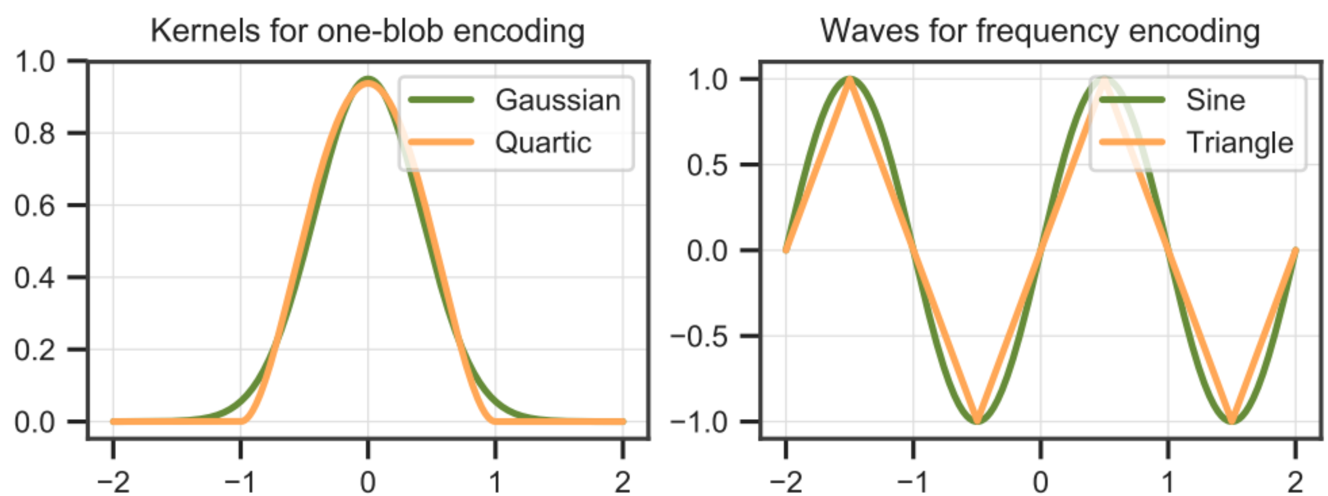
Fully-Fused Network

- Reducing memory bottleneck highly increases training & inference speed



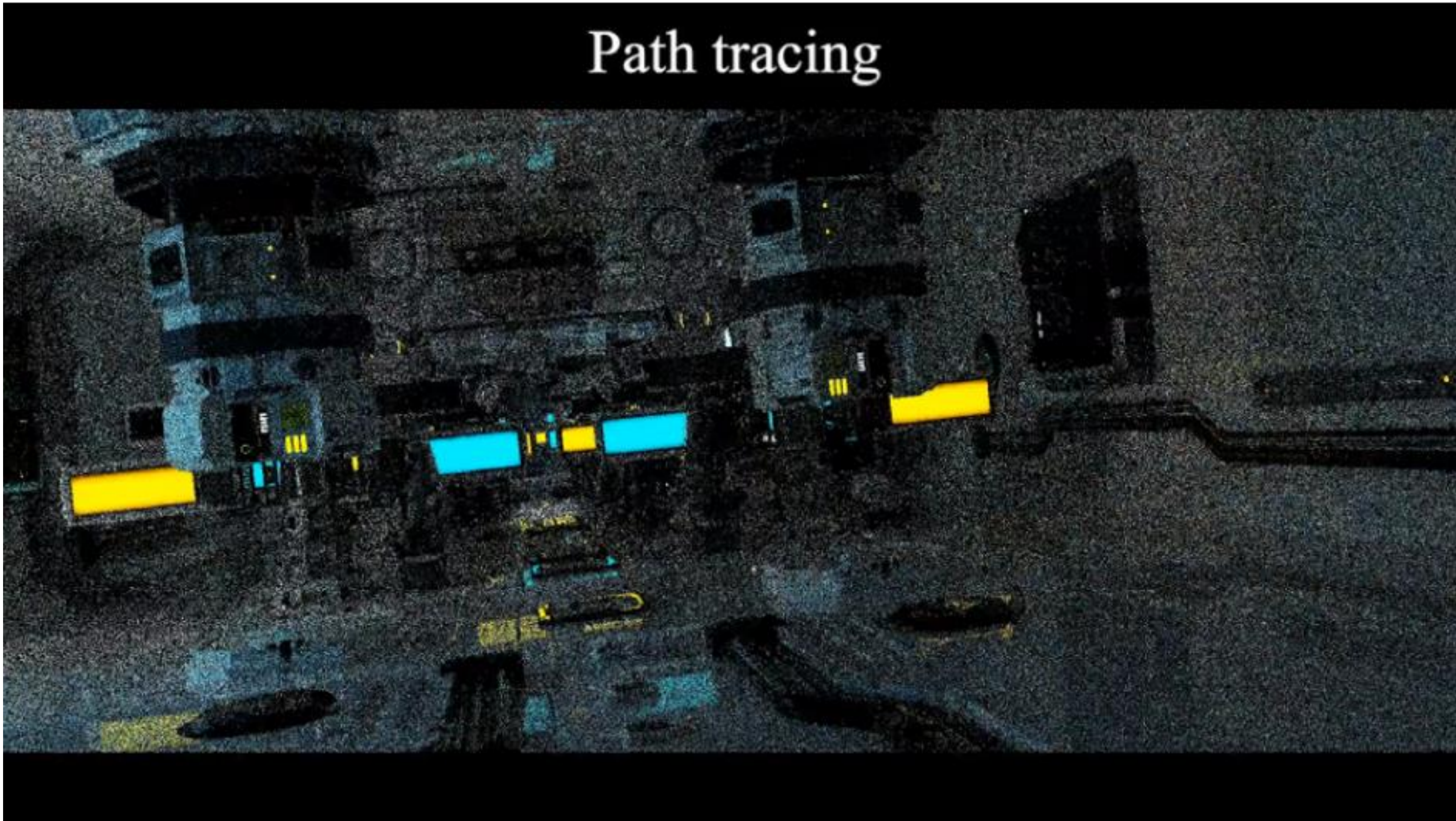
Efficient encoding for faster gradient computations

- Approximate the encoding functions into polynomial functions for faster gradient computations
- Buys 0.25ms per frame (1spp)



$\text{quartic}(x) := \frac{15}{16}(1 - x^2)^2$ $\quad$ $\text{tri}(x) := 2\,|x \bmod 2 - 1| - 1$

Results – 1spp Video



Results – 1spp Video

Spatiotemporal Reservoir Resampling for Real-time Ray Tracing with Dynamic Direct Lighting, Bitterli et al., SIGGRAPH 2020
Neural Temporal Adaptive Sampling and Denoising, Hasselgren et al., Computer Graphics Forum 2020

Results – With Image Denoiser

Results – Fast Adaptation

Neural Radiance Cache: Wrap-up

- Introducing a radiance caching techinique by training a radiance-caching neural network

- Self-training with Fast Adaptation to achieve real-time for rendering & training

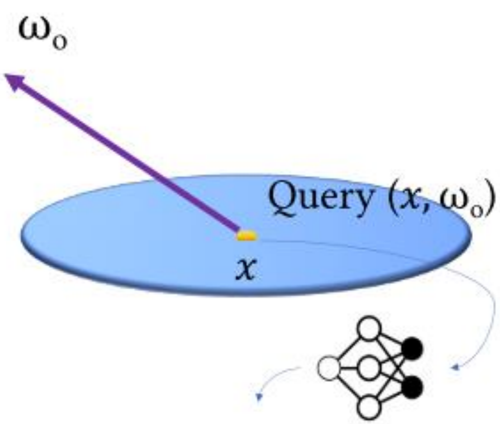
- Vast number of techniques to achieve real time

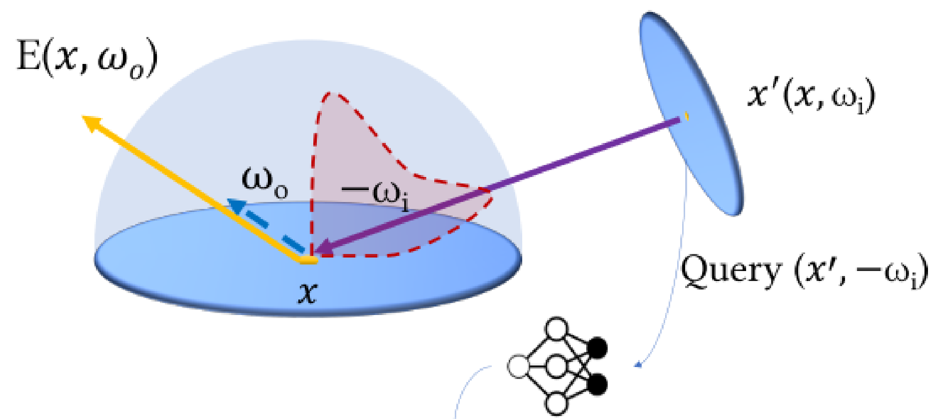
Appendices

Neural Radiosity

Rendering with Neural Radiosity

- Gather radiances estimated on the first bounce
- Gather radiances calculated with estimated incoming radiances into the first bounce





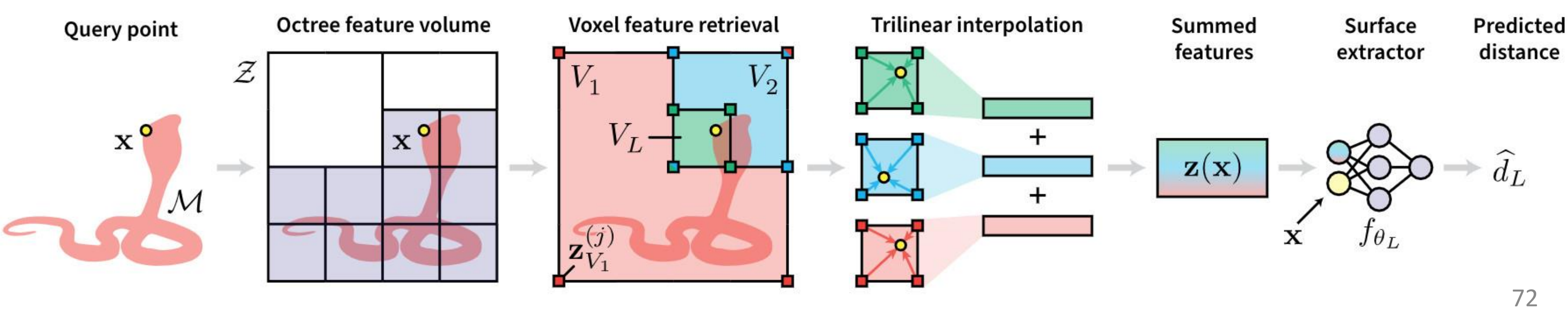
$$\mathcal{L}(\theta) = \Big\| \underbrace{L_\theta(\boldsymbol{x}, \boldsymbol{\omega}_o)}_{\text{Left Hand Side(LHS)}} - \underbrace{\left(E(x, \omega_o) + \int f(x, \omega_o, \omega_i) \cdot L_\theta(\boldsymbol{x}'(\boldsymbol{x}, \boldsymbol{\omega}_i), -\boldsymbol{\omega}_i). \, d\omega_i^\perp \right)}_{\text{Right Hand Side(RHS)}} \Big\|_2^2$$

Rendering with Neural Radiosity

- Rendering with RHS shows better quality, but has more overhead due to the calculation

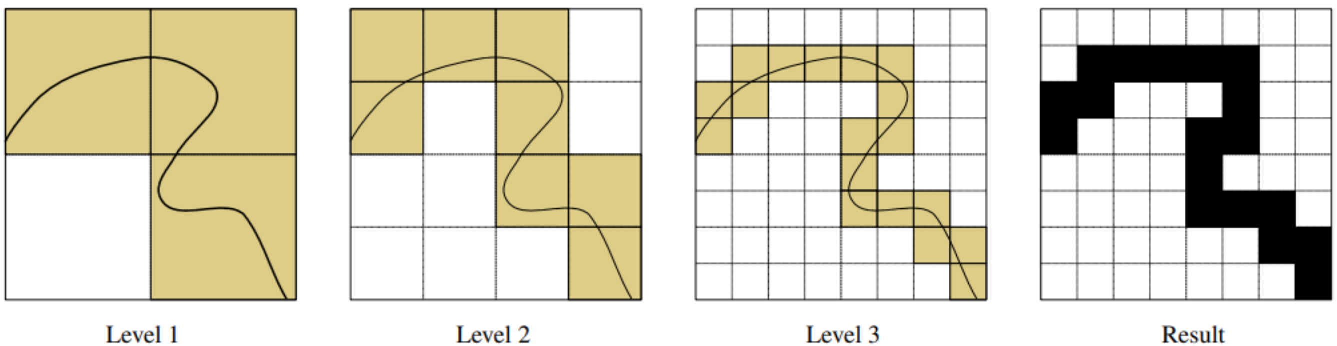
Multi-resolution Feature Grid

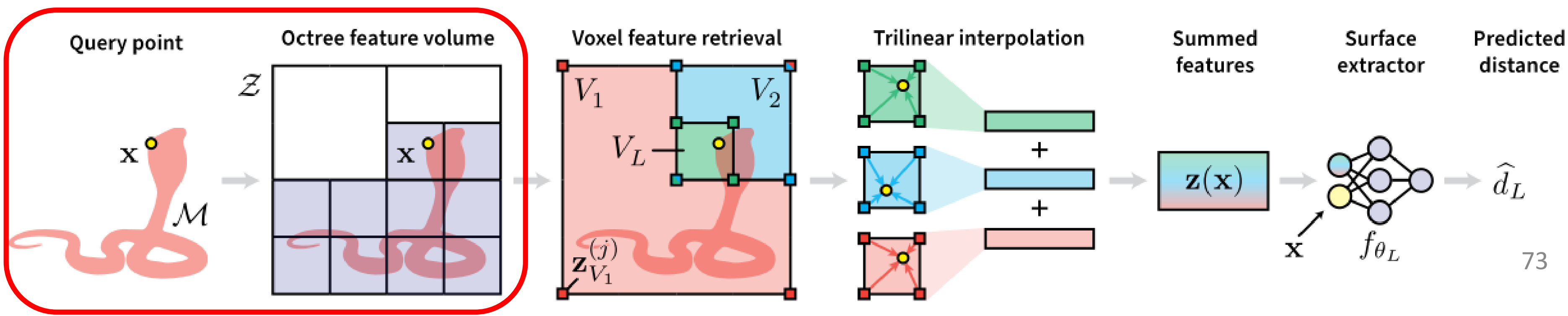
- Idea & Implementation borrowed by NGLOD
  - Neural Geometry Level of Details, CVPR 2021
- Originally for better representation of SDF



Multi-resolution Feature Grid

- Each level of voxel grids have trainable vectors
- Voxel Octree to implement multi-resolution voxel grids

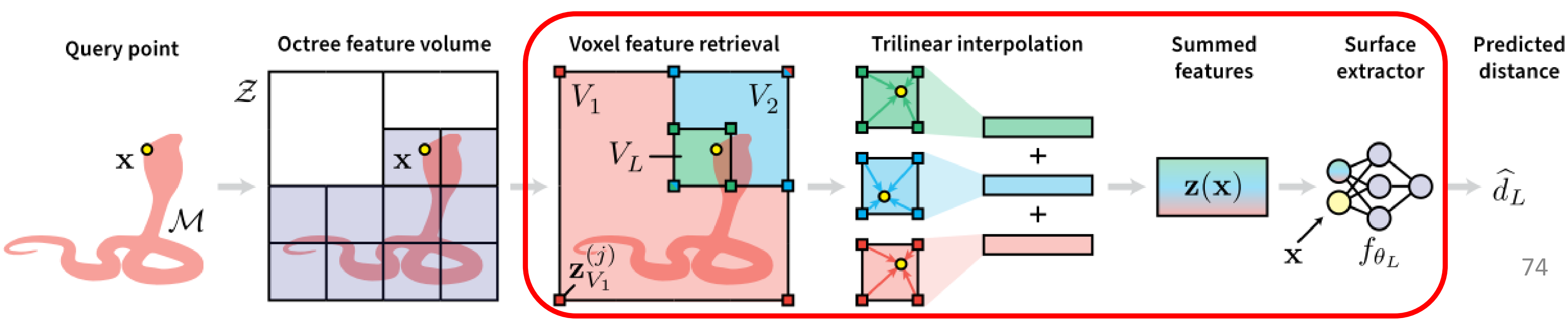




Multi-resolution Feature Grid

- Features of the query point as interpolated feature vectors of each level of voxel grids
- Allows better performance with using relatively shallow network

$$L_\theta(x, \omega_o) = MLP\begin{pmatrix} x \\ G(x) \\ \omega_o \end{pmatrix}, \quad \mathrm{G}(x) = \frac{1}{n}\sum_0^{n-1} trilinear(x, V_i[x])$$

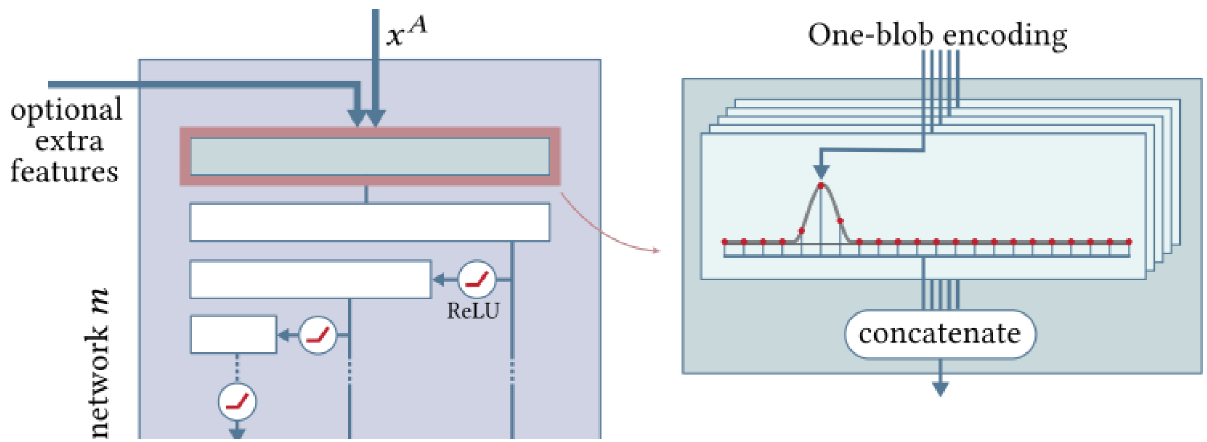


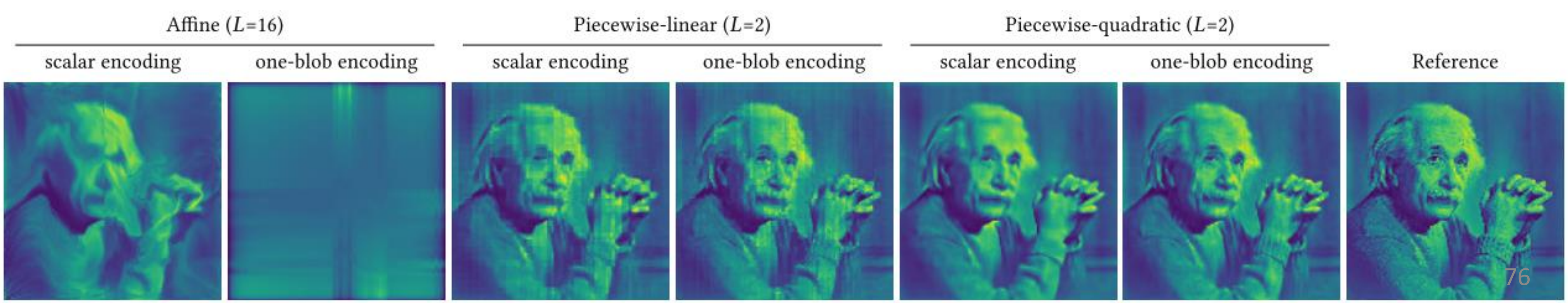
Appendices

Real-time Neural Radiance Caching for Path Tracing

One-blob Encoding

- Smoothes the one-hot vectors to reduce loss of information



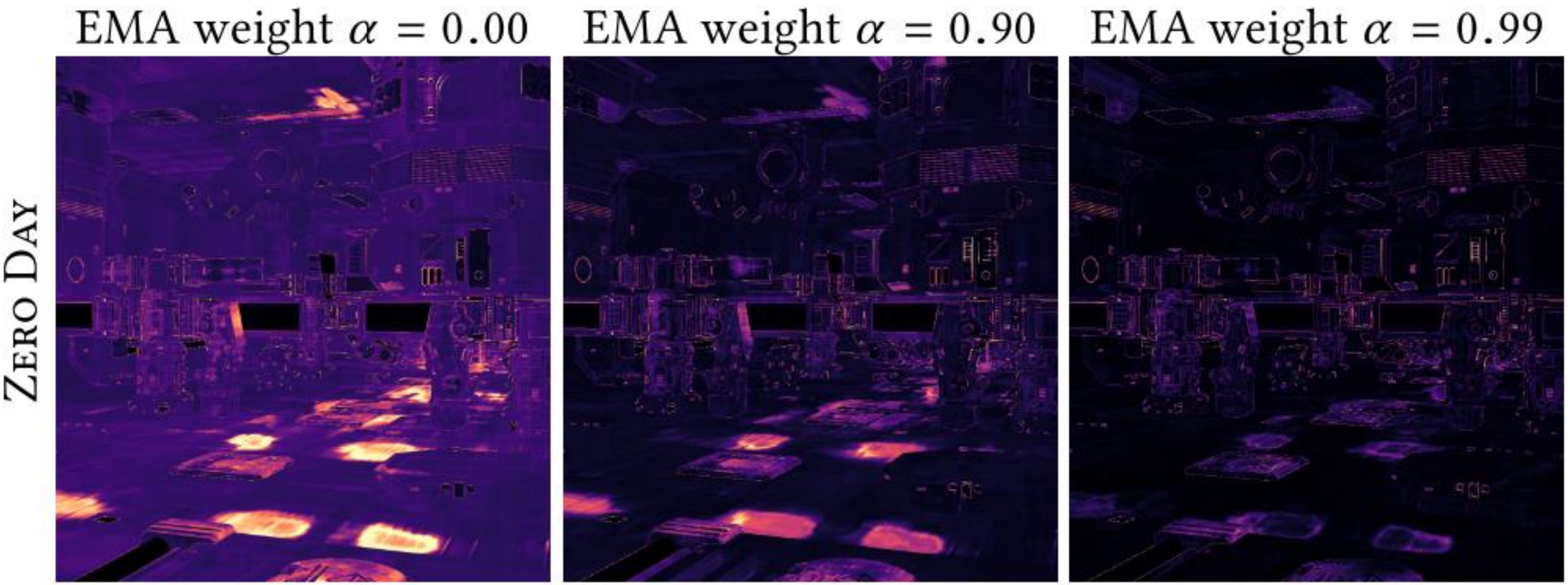


Temporal Stability via EMA

- Aggressive self-training strategy might lead to overfitting, creating temporal artifacts like flickering
- To reduce such phenomenon, we average the network weights via EMA
- $\bar{W}_t = \frac{1-\alpha}{\eta_t} \cdot W_t + \alpha \cdot \eta_{t-1} \cdot \bar{W}_{t-1}, \eta_t = 1 - \alpha^t$
  - $\alpha = 0.99$

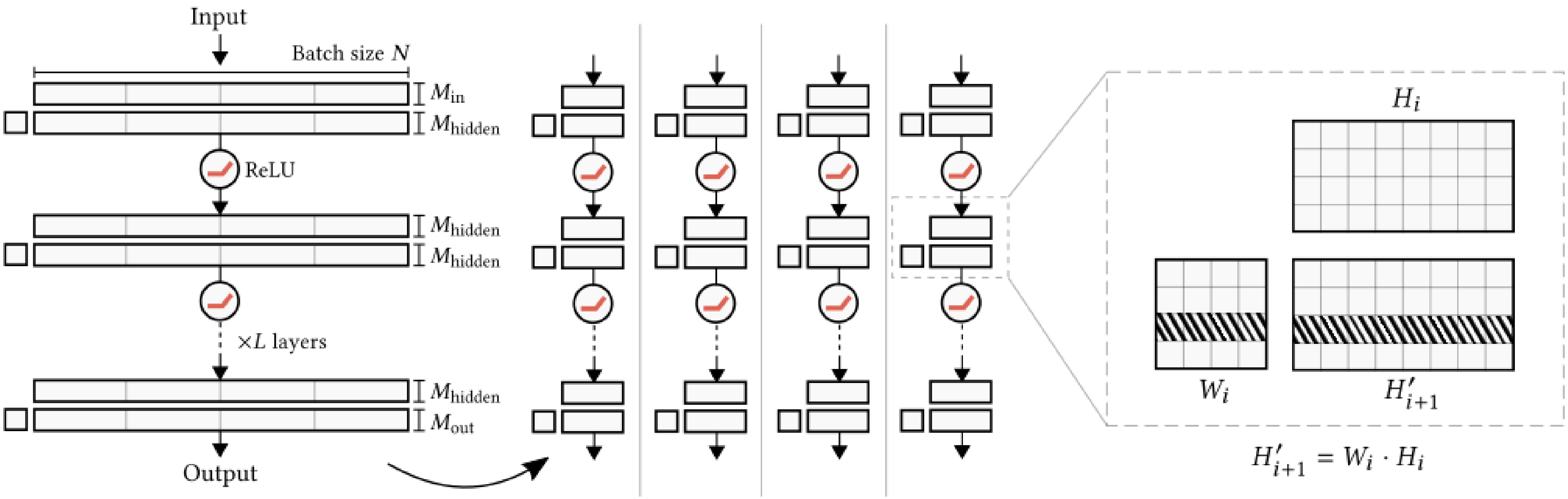
Temporal Stability via EMA

- Aggressive self-training strategy might lead to overfitting, creating temporal artifacts like flickering
- $\overline{W}_t = \frac{1-\alpha}{\eta_t} \cdot W_t + \alpha \cdot \eta_{t-1} \cdot \overline{W}_{t-1}, \eta_t = 1 - \alpha^t$



Fully-Fused Network

- A new GPU kernel that highly reduces the memory bottleneck between high-level memory(VRAM) and on-chip memory (low-level cahces, registers, etc…)



(a) Batched neural network evaluation **(b)** Distribution of a batch over thread blocks **(c)** Per-thread-block matrix multiplication

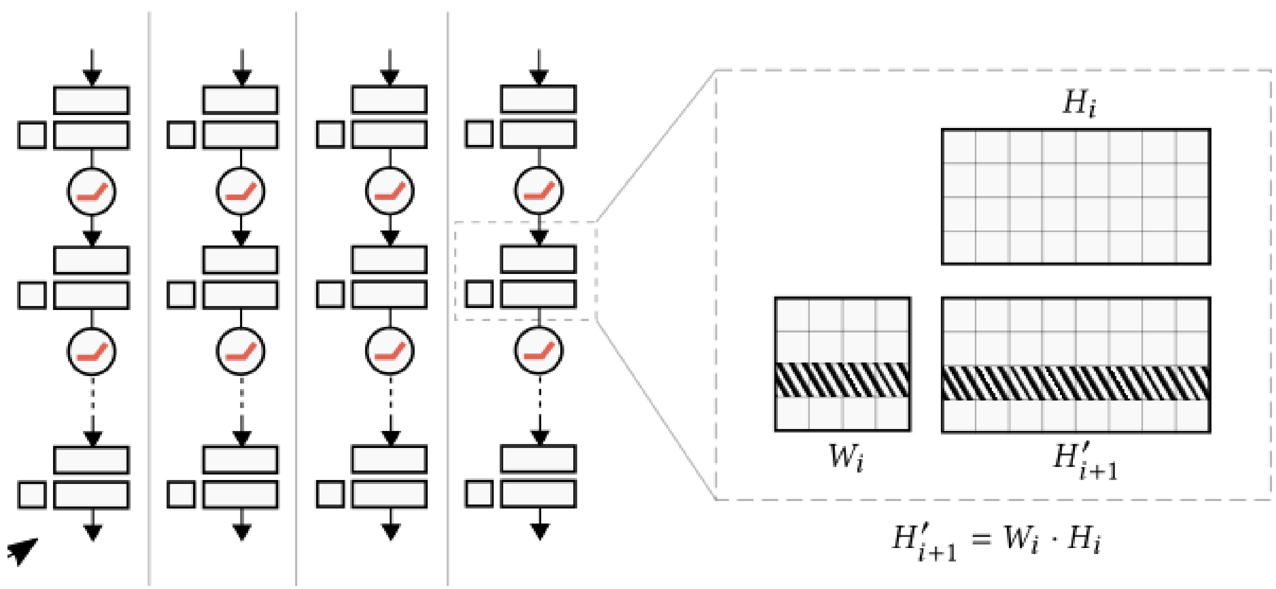
Fully-Fused Network

- Divide the large batch (2^{12} for FHD 1920x1080) into small minibatches (128)
  - Might differ by capacity of on-chip memory of GPU
- Each minibatch is used for training in each thread parallely

- Divide the large batch (2^{12} for FHD 19
  minibatches (128)
  - Might differ by capacity of on-chip memo
- Each minibatch is used for training in e

Fully-Fused Network

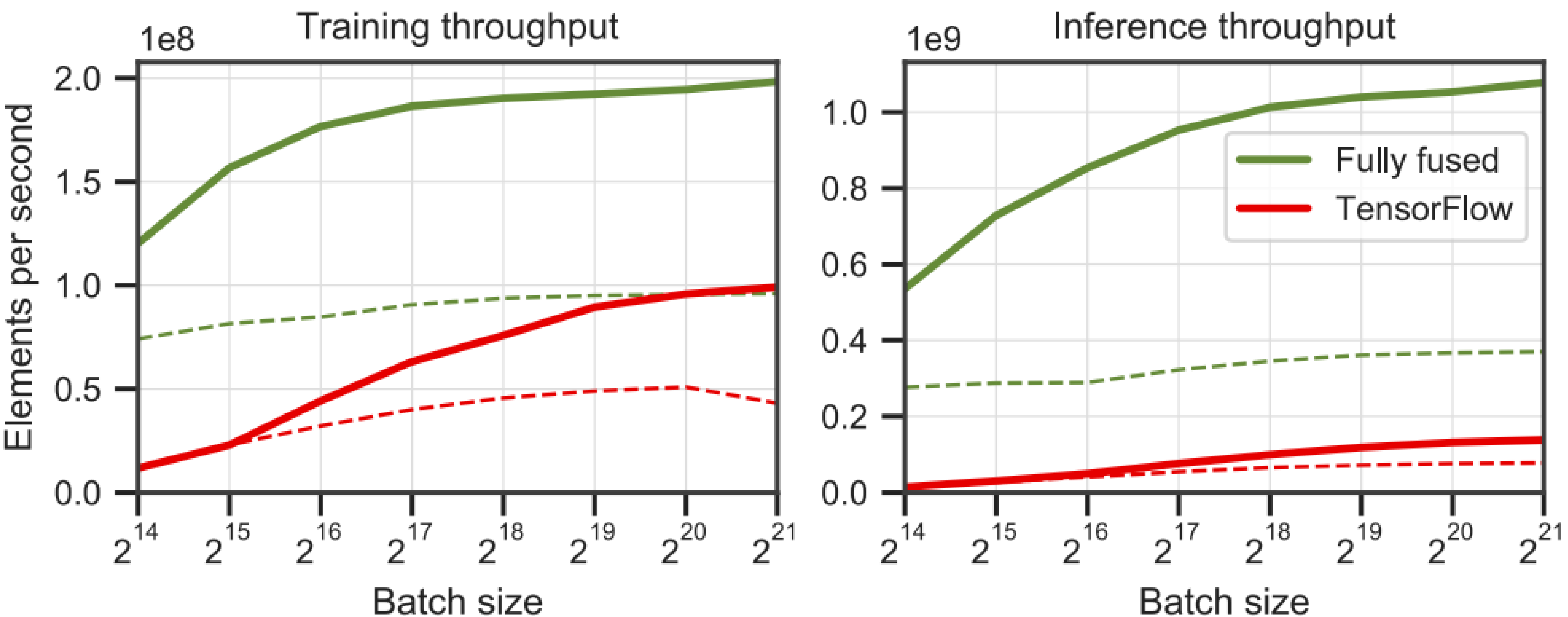
- The memory consumption of the matrix multiplication in each thread are set to perfectly fit the low-level memory
  - Specifically, multiplication of each row & column
  - For GTX 3090, minibatch of 128 and hidden layer of 64 fully utilizes its register



(b) Distribution of a batch over thread blocks **(c)** Per-thread-block matrix multiplication

Fully-Fused Network

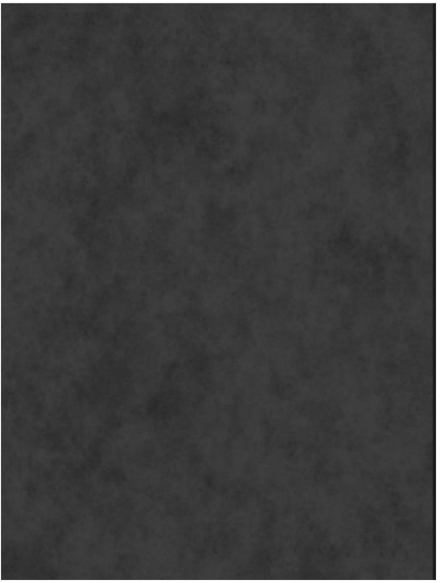
- Reducing memory bottleneck highly increases training & inference speed



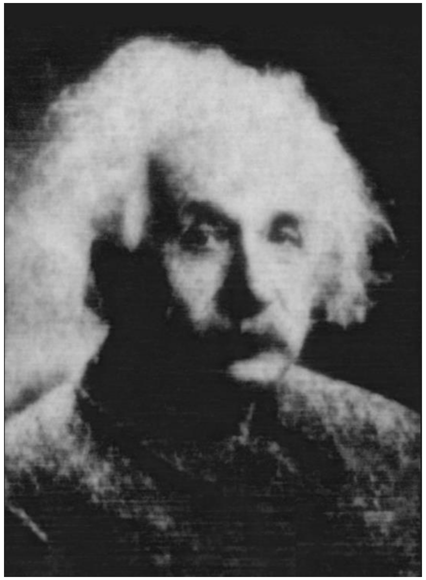
Comparison with XLA-enabled TensorFlow

Fully-Fused Network

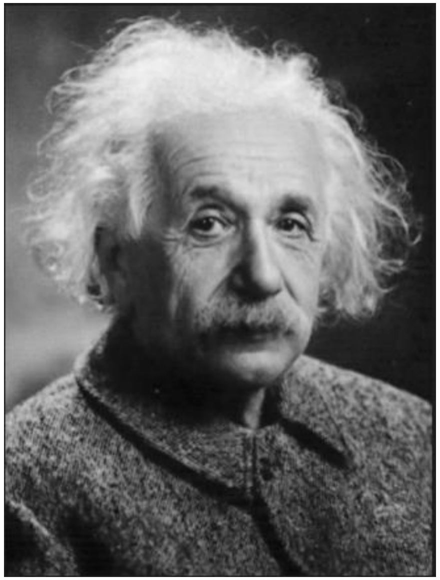
- Reducing memory bottleneck highly increases training & inference speed
- Fast image learning with high resolution (3250x4333)



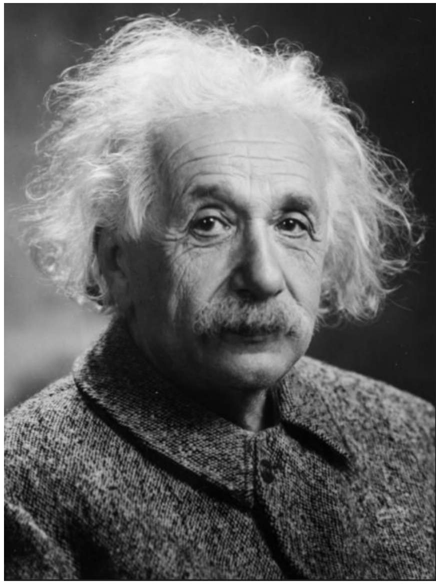
0 ms



4.2 ms



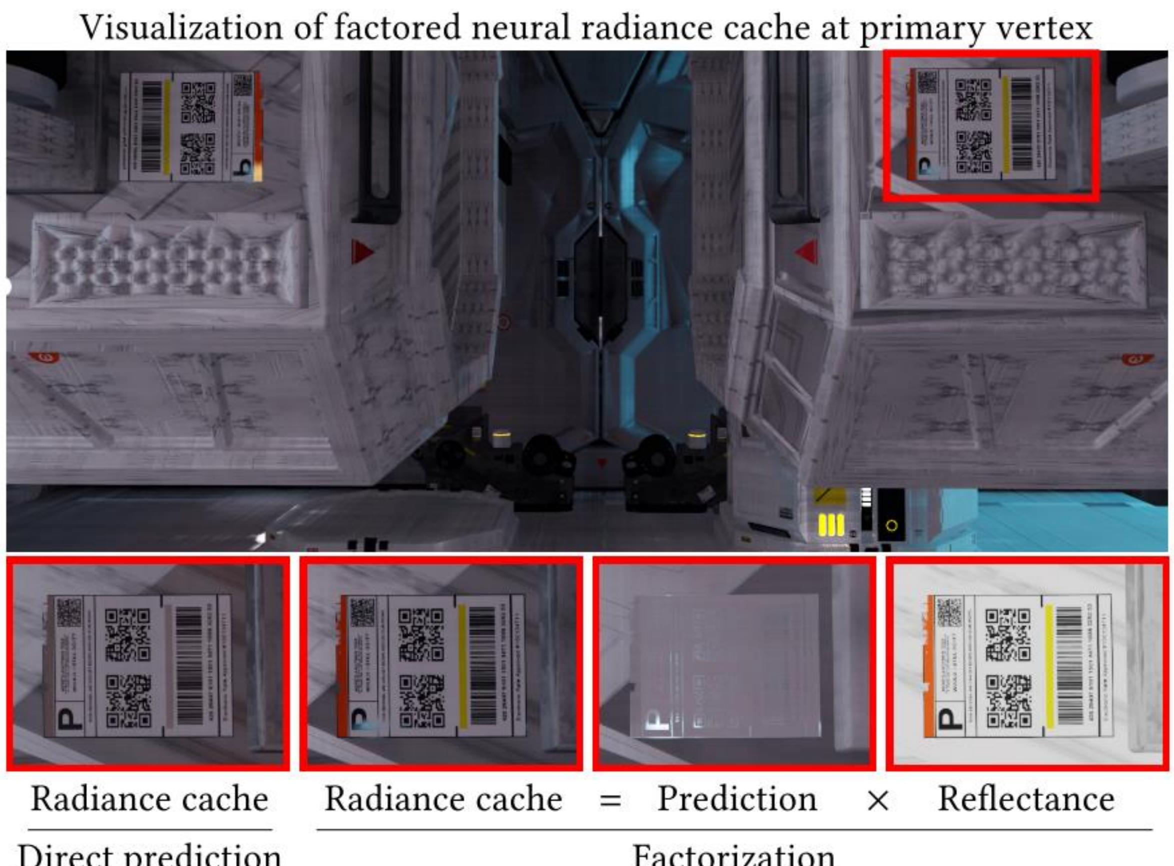
420 ms



GT

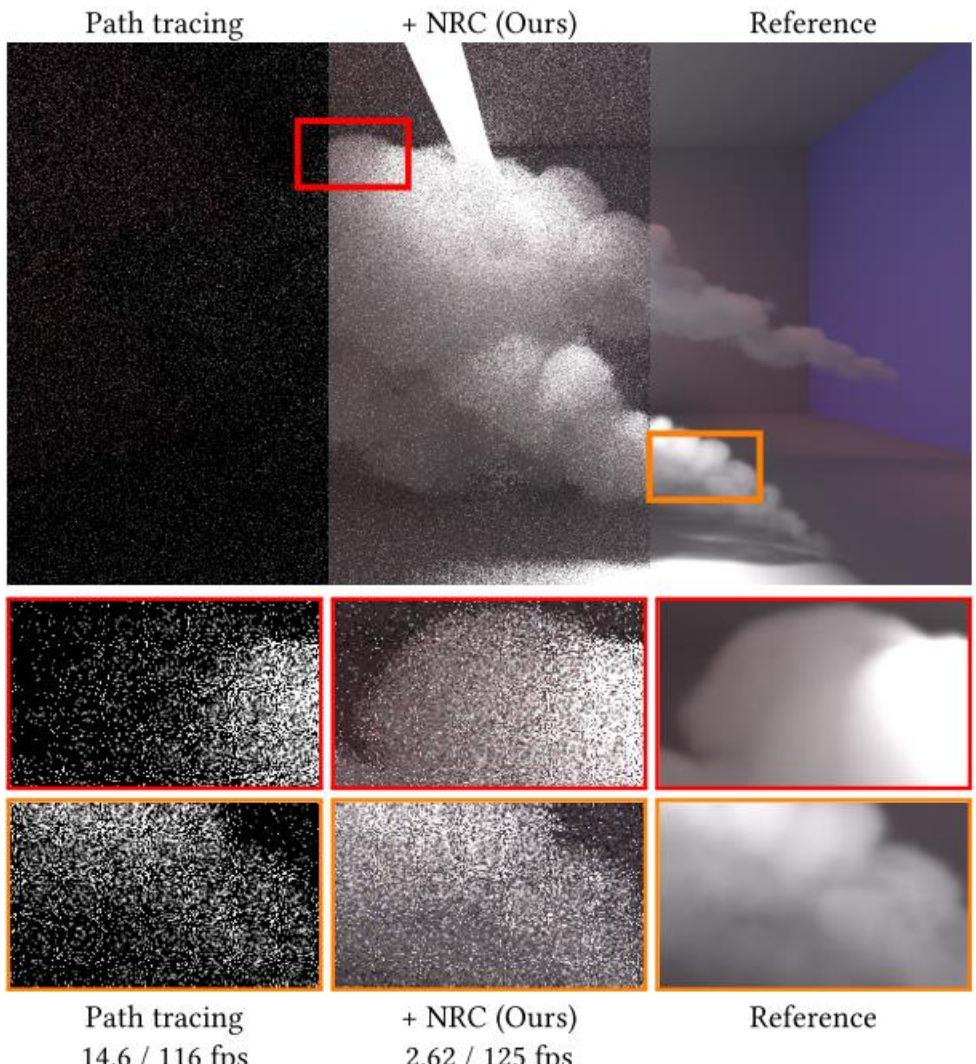
Reflectance Factorization

- Helps the network to focus on details by light transport rather than texture details



Results – Numerical Results

Results – Volume Rendering



Spatiotemporal Reservoir Resampling for Real-time Ray Tracing with Dynamic Direct Lighing, Bitterli et al., SIGGRAPH 2020
Dynamic Diffuse Global Illumination with Ray-traced Irradiance Fields, Majercik et al., JCGT 2019

Results – Rendering Cost

Table 3. Breakdown of rendering cost by component.

Scene	Method	Trace & shade	Query	Training	Total
Attic	PT+ReSTIR	12.96 ms	—	—	**12.96 ms**
	PT+ReSTIR+DDGI	11.56 ms	0.64 ms	1.78 ms	13.98 ms
	PT+ReSTIR+NRC	10.88 ms	1.66 ms	1.12 ms	13.66 ms
Bistro	PT+ReSTIR	13.75 ms	—	—	**13.75 ms**
	PT+ReSTIR+DDGI	12.71 ms	0.65 ms	1.68 ms	15.04 ms
	PT+ReSTIR+NRC	11.96 ms	1.38 ms	1.11 ms	14.45 ms
Classroom	PT+ReSTIR	18.06 ms	—	—	18.06 ms
	PT+ReSTIR+DDGI	12.93 ms	0.59 ms	1.65 ms	15.17 ms
	PT+ReSTIR+NRC	12.28 ms	1.70 ms	1.11 ms	**15.09 ms**
Living Room	PT+ReSTIR	8.32 ms	—	—	8.32 ms
	PT+ReSTIR+DDGI	5.68 ms	0.52 ms	0.99 ms	**7.19 ms**
	PT+ReSTIR+NRC	5.82 ms	1.85 ms	1.11 ms	8.78 ms
Pink Room	PT+ReSTIR	6.73 ms	—	—	**6.73 ms**
	PT+ReSTIR+DDGI	5.56 ms	0.52 ms	0.89 ms	6.97 ms
	PT+ReSTIR+NRC	5.36 ms	1.54 ms	1.12 ms	8.02 ms
Zero Day	PT+ReSTIR	13.89 ms	—	—	13.89 ms
	PT+ReSTIR+DDGI	8.34 ms	0.54 ms	1.21 ms	**10.09 ms**
	PT+ReSTIR+NRC	8.67 ms	1.41 ms	1.09 ms	11.17 ms
Average	PT+ReSTIR	12.29 ms	—	—	12.29 ms
	PT+ReSTIR+DDGI	9.46 ms	0.58 ms	1.37 ms	**11.41 ms**
	PT+ReSTIR+NRC	9.16 ms	1.59 ms	1.11 ms	11.86 ms