CMSC740
Advanced Computer Graphics

Fall 2025

Matthias Zwicker

Deep learning in graphics

- So far
  - Denoising (map noisy image to clean image)

Neural representations in graphics/vision

- General idea: represent continuous quantities/functions in computer graphics with neural networks
- Examples
  - Geometry (network implements implicit surface representation using signed distance fields, or binary occupancy/inside-outside function)
  - Radiance fields (given ray, network returns radiance)
  - Images, videos (given pixel coordinates, network returns color)
  - Etc.

General approach

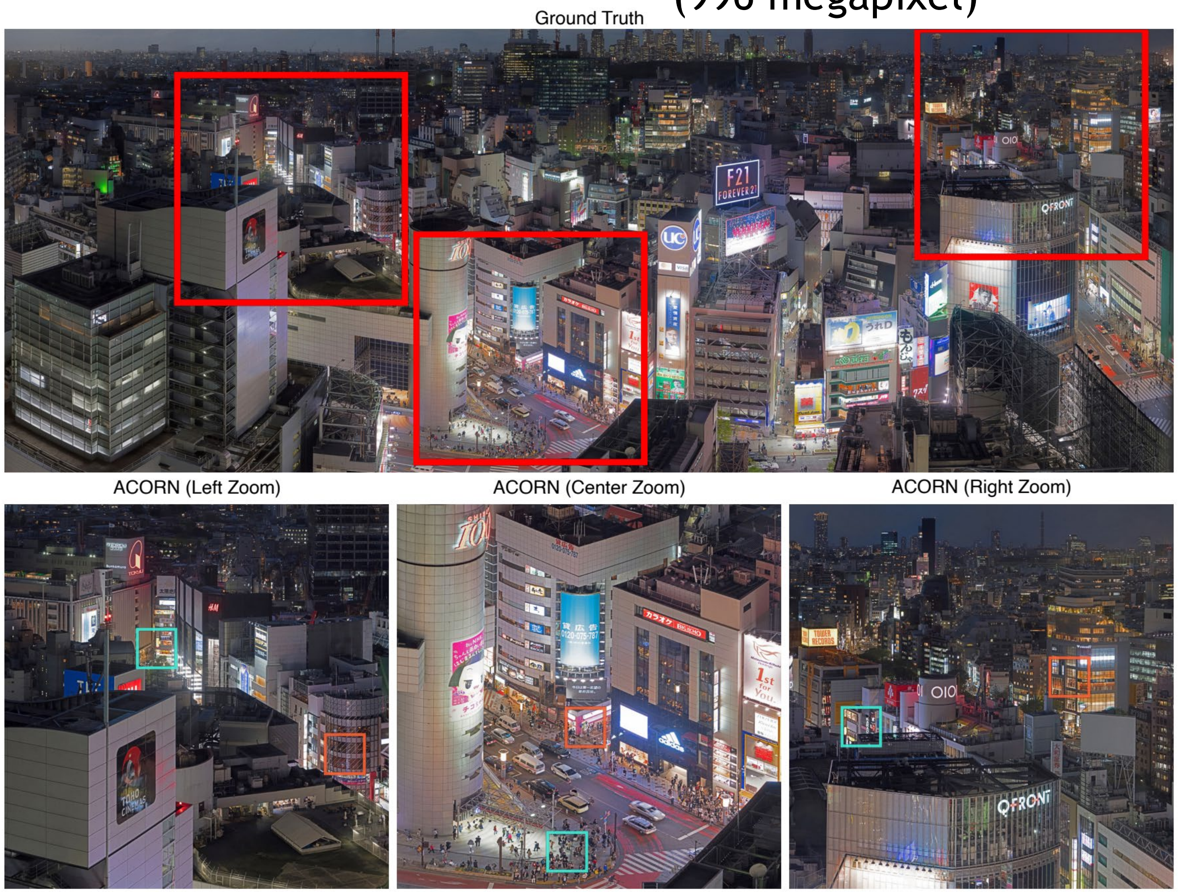
- Naïve: MLP as function representation for various objects

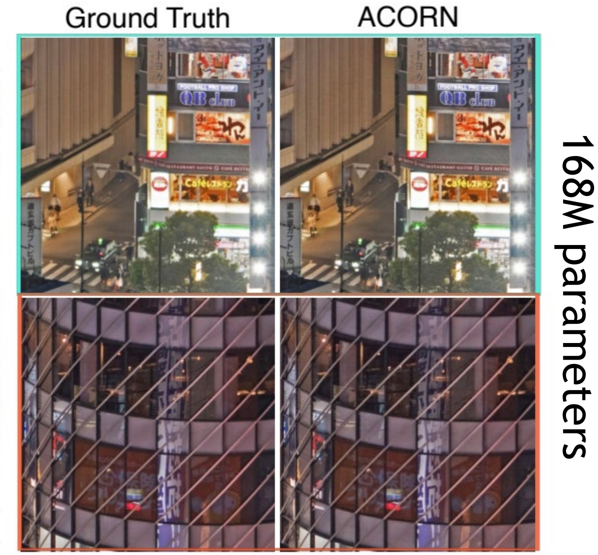
(x,y)	Neural network	RGB (+ time)
(x,y,z)	(simplest case:	3D occupancy or signed distance value (3D shape)
(x,y,z,θ,ϕ)	multilayer	Radiance
Etc.	perceptron, MLP)	Etc.

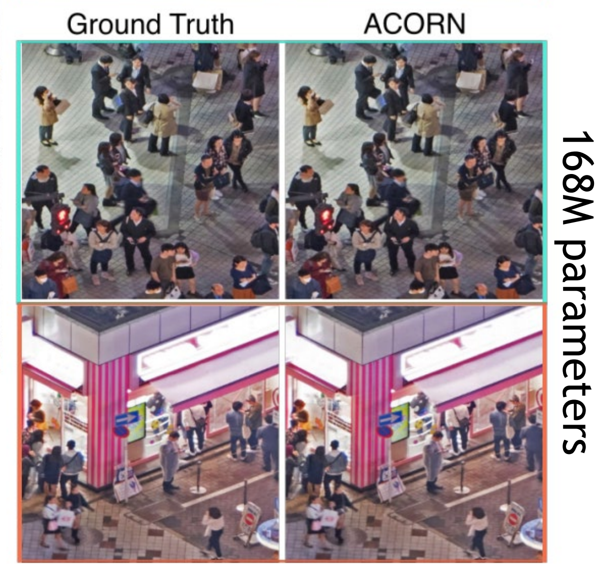
- Approach: train MLP for each “object” individually, using reconstruction loss (e.g., L2-loss distance between output and ground truth)
- “Objects”:
  - Radiance fields
  - 3D geometry
  - Images
  - Videos

Example: Image represented as NN

(996 megapixel)







ACORN: Adaptive Coordinate Networks for
Neural Scene Representation

https://arxiv.org/pdf/2105.02788.pdf

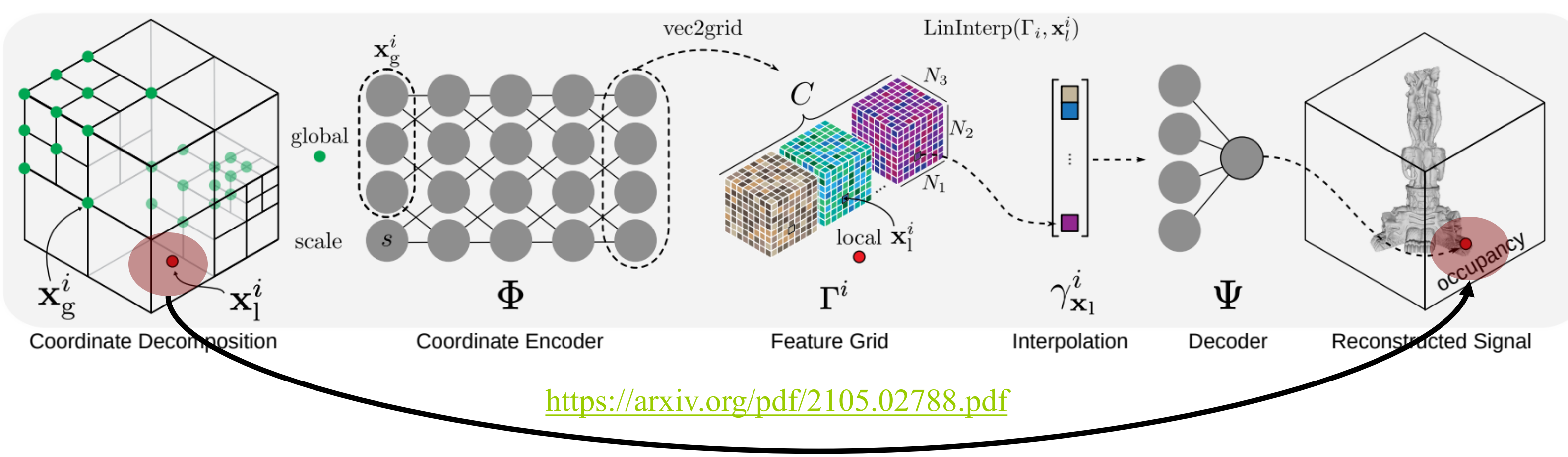
(x,y) -> Neural Network -> RGB, trained for each image separately

Approach

- Naïve

Input		Model		Output
(x,y) (x,y,z) (x,y,z,θ,ϕ) Etc.	→	Neural network (simplest case: **multilayer perceptron, MLP)**	→	RGB (+ time) 3D occupancy or signed distance value (3D shape) Radiance Etc.

- Better: replace MLP with more sophisticated architectures (e.g., ACORN: Adaptive Coordinate Networks)



https://arxiv.org/pdf/2105.02788.pdf

Advantages, disadvantages

- Conventional techniques: Splines, wavelets, etc.
- Advantages of neural networks
  - Represent complicated functions efficiently, with low storage cost (adaptive, nonlinear); most useful if desired function is given as solution of some equation (PDE, integral equation, etc.)
  - Leverage deep learning infrastructure (GPUs, automatic differentiation, numerical optimization techniques for neural networks, Python libraries)
- Disadvantages of neural networks
  - Requires nonlinear optimization (instead of linear as for some other techniques)
  - Slower to evaluate than conventional techniques
  - Relation of network architectures to accuracy, convergence properties not understood as well as for conventional techniques

Application in graphics: neural representation to solve rendering equation

- First: background on traditional radiosity technique to solve rendering equation
  https://en.wikipedia.org/wiki/Radiosity_(computer_graphics)
  - Radiosity solves simplified rendering equation only for **diffuse** surfaces
  - Instead of radiance, use radiosity that doesn't depend on direction at each surface point (due to diffuse-only reflection)
- Then: using neural networks
  - Works for general BRDFs

Radiosity (1984) (https://en.wikipedia.org/wiki/Radiosity_(computer_graphics))

- Rendering equation restricted to diffuse surfaces (radiosity B instead of radiance L, no directional dependence; diffuse BRDF ρ, scene surfaces M)

$$B(\mathbf{x}') = E(\mathbf{x}') + \rho(x') \int_{\mathcal{M}} B(\mathbf{x}) G(\mathbf{x} \leftrightarrow \mathbf{x}') d\mathbf{x},$$

3-point form, area integration

- Key idea: discretization of desired, continuous radiosity $B(x)$ using piecewise constant functions B_i over mesh elements i

$$B_i = E_i + \rho_i \sum_{j=1}^{n} F_{ij} B_j$$

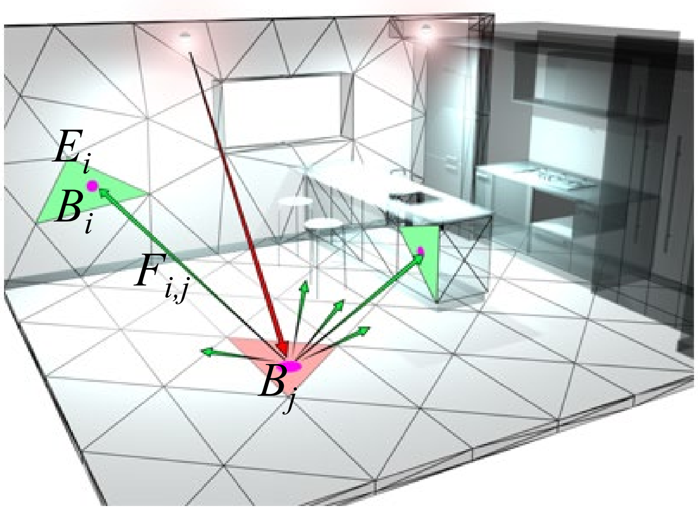
Sum over all mesh faces approximates integral over scene surfaces M

Constant radiosity over mesh face i

Constant emission over mesh face i

  - Form factors $F_{i,j}$ between mesh faces i, j

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \underbrace{G(\mathbf{x} \leftrightarrow \mathbf{x}')}_{\text{Geometry term}} \underbrace{dA_j dA_i}_{\text{Mesh faces}},$$



Radiosity solution

- Need to solve linear system for unknowns B_i

$$B_i = E_i + \rho_i \sum_{j=1}^{n} F_{ij} B_j$$

- Goal: minimize difference between left- and right-hand side to find "equilibrium"

$$\underset{\mathbf{B}}{\operatorname{argmin}} \Sigma_i \left\| B_i - \left(E_i + \rho_i \sum_{j=1}^{n} F_{ij} B_j \right) \right\|_2$$

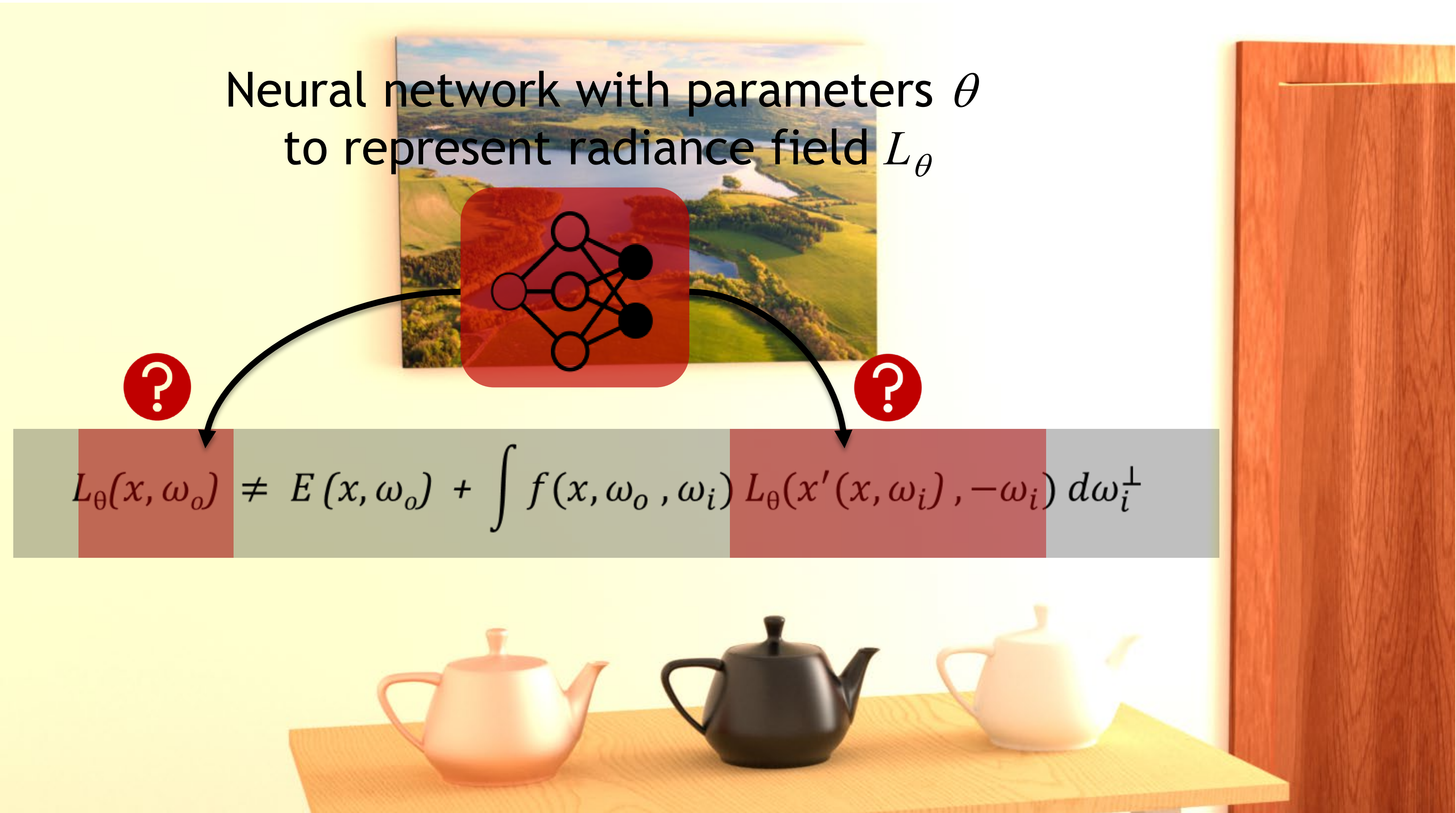
- Solution (n x n matrix $\mathbf{F}$, column vectors $\mathbf{B}, \mathbf{E}$)

$$\mathbf{B} = (\mathbf{I} - \rho \mathbf{F})^{-1} \mathbf{E}$$

Challenges

- High mesh resolution (i.e., small piecewise constant elements) required for high accuracy
  - Very large linear systems to solve
- Extending to arbitrary BRDFs, radiance fields requires 4D discretization
  - Instead of 2D mesh elements for diffuse surfaces
- Resurrect concept of radiosity using neural networks?
  - Use neural networks to represent radiance, instead of conventional approach such as piecewise constant functions over mesh

Neural radiosity

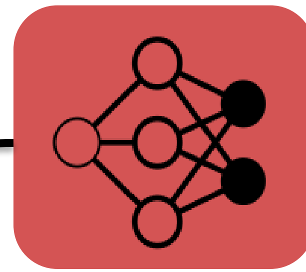


Neural network with parameters θ
to represent radiance field L_θ

$$L_\theta(x,\omega_o) \neq E(x,\omega_o) + \int f(x,\omega_o,\omega_i)\, L_\theta(x'(x,\omega_i),-\omega_i)\, d\omega_i^\perp$$

https://arxiv.org/abs/2105.12319

Training/optimization



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

Training loss L for network parameters θ
as norm of residual of rendering equation

Estimating norm of residual

- Residual: difference between LHS, RHS

$$r_\theta(x,\omega_o) = L_\theta(x,\omega_o) - \left(E(x,\omega_o) + \int f(x,\omega_o,\omega_i)\, L_\theta(x'(x,\omega_i), -\omega_i)\, d\omega_i^\perp\right)$$

- Monte Carlo sampling surface locations x_j, directions $\omega_{o,j}$

$$\boldsymbol{\mathcal{L}(\theta)} = \|\boldsymbol{r_\theta}(x,\omega_o)\|_2^2 = \frac{1}{N}\sum_{j=1}^{N}\frac{r_\theta(x_j,\omega_{o,j})}{p(x_j,\omega_{o,j})}$$

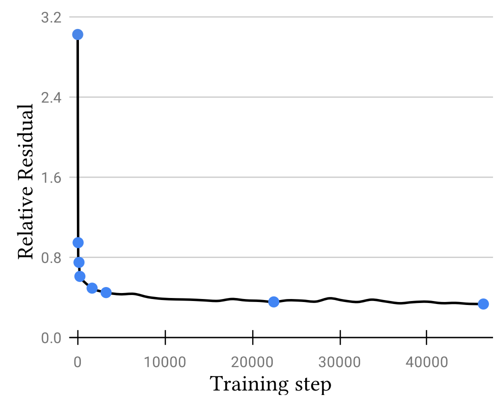
- Minimizing loss: stochastic gradient descent using batches of Monte Carlo samples

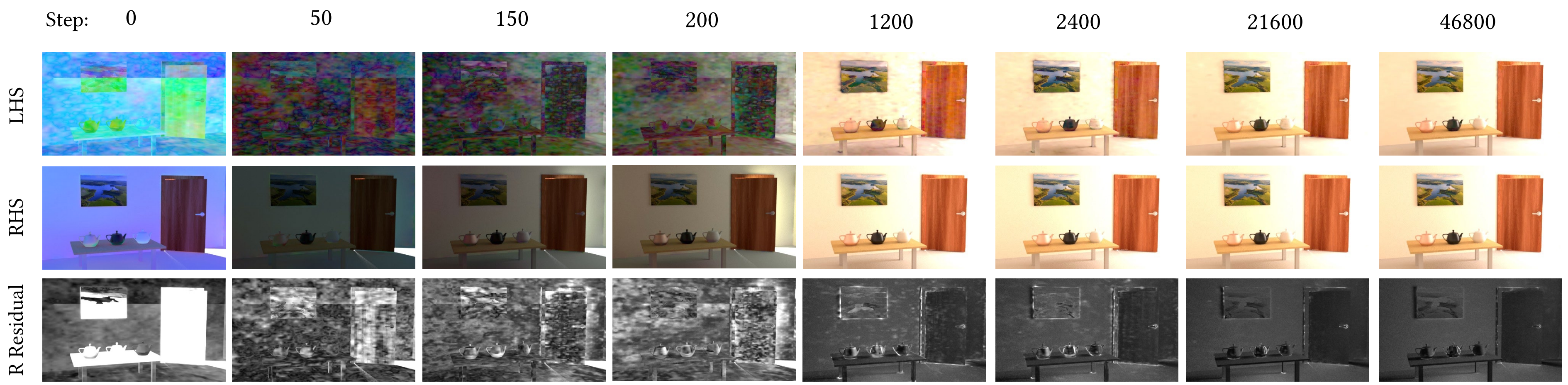
Network architecture

- Instead of naïve MLP, use sparse multi-resolution feature grid, details in paper
  https://arxiv.org/abs/2105.12319

Training/optimization

Network needs to be trained for each scene!





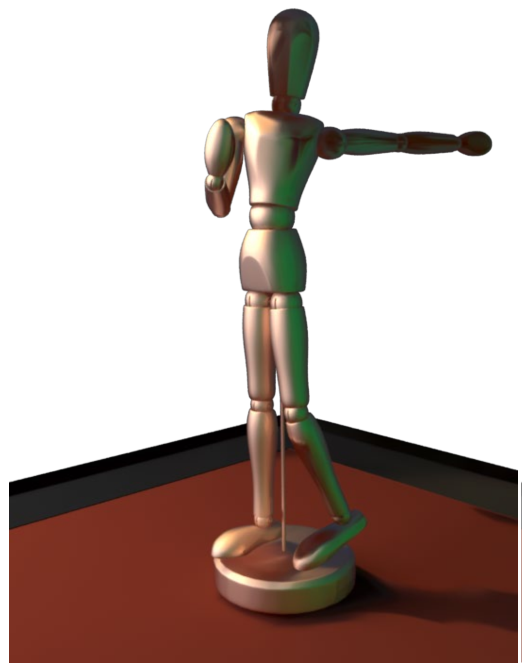
As training progresses, left- and right-hand side become more similar, residual vanishes, radiance field converges to solution of rendering equation

Dynamic scenes

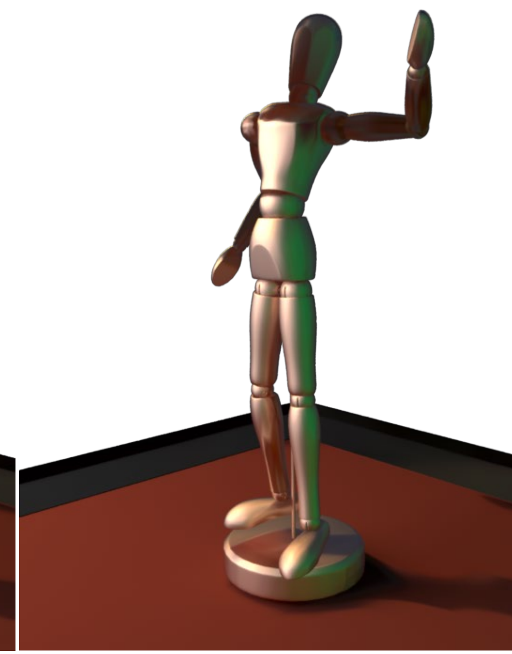
In dynamic scenes (animations), can “re-use” and adapt network trained for initial scene to save training time

LHS – Initial scene
Rest Pose

Same
network



LHS - Stretch Pose



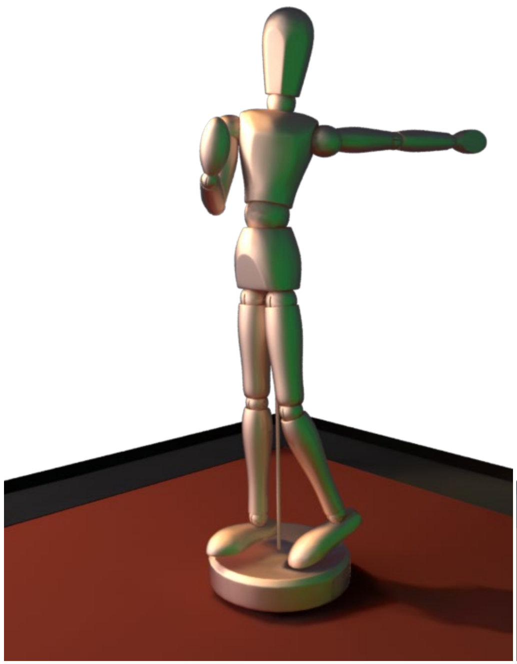
LHS - Speak Pose

Without fine-tuning (retraining), image looks roughly ok but network doesn’t represent correct solution after scene geometry changes

Dynamic scenes

LHS – Initial scene
Rest Pose

After
fine
tuning

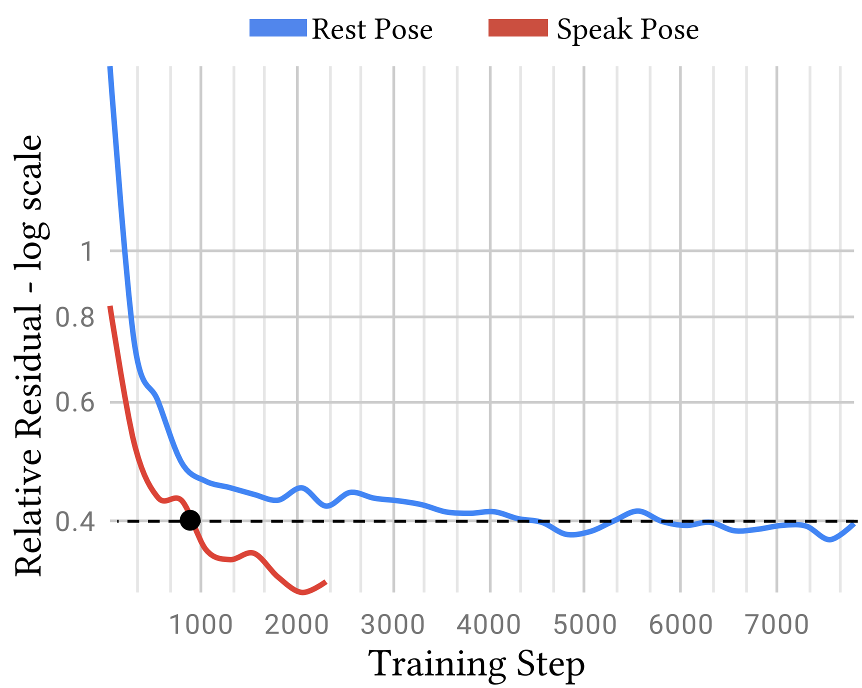


LHS - Stretch Pose

LHS - Speak Pose

Accurate radiance field after network was updated to represent new scene

Fine tuning



Fine-tuning (red line) converges faster than original training (blue line)

Related work: Real-time Neural Radiance Caching for Path Tracing

https://arxiv.org/abs/2106.12372

https://research.nvidia.com/publication/2021-06_real-time-neural-radiance-caching-path-tracing

Neural Radiosity Conclusions

- Neural networks can accurately represent complex radiance fields
  - Can use as representation for unknown functions of numerical optimization problems, such as rendering equation
- Neural radiosity requires per scene training, which can be slow
  - Training time can be amortized over multiple frames in dynamic scenes
- Real-time performance possible (real-time neural radiance caching)