Lecture 19:

Volumes and Points

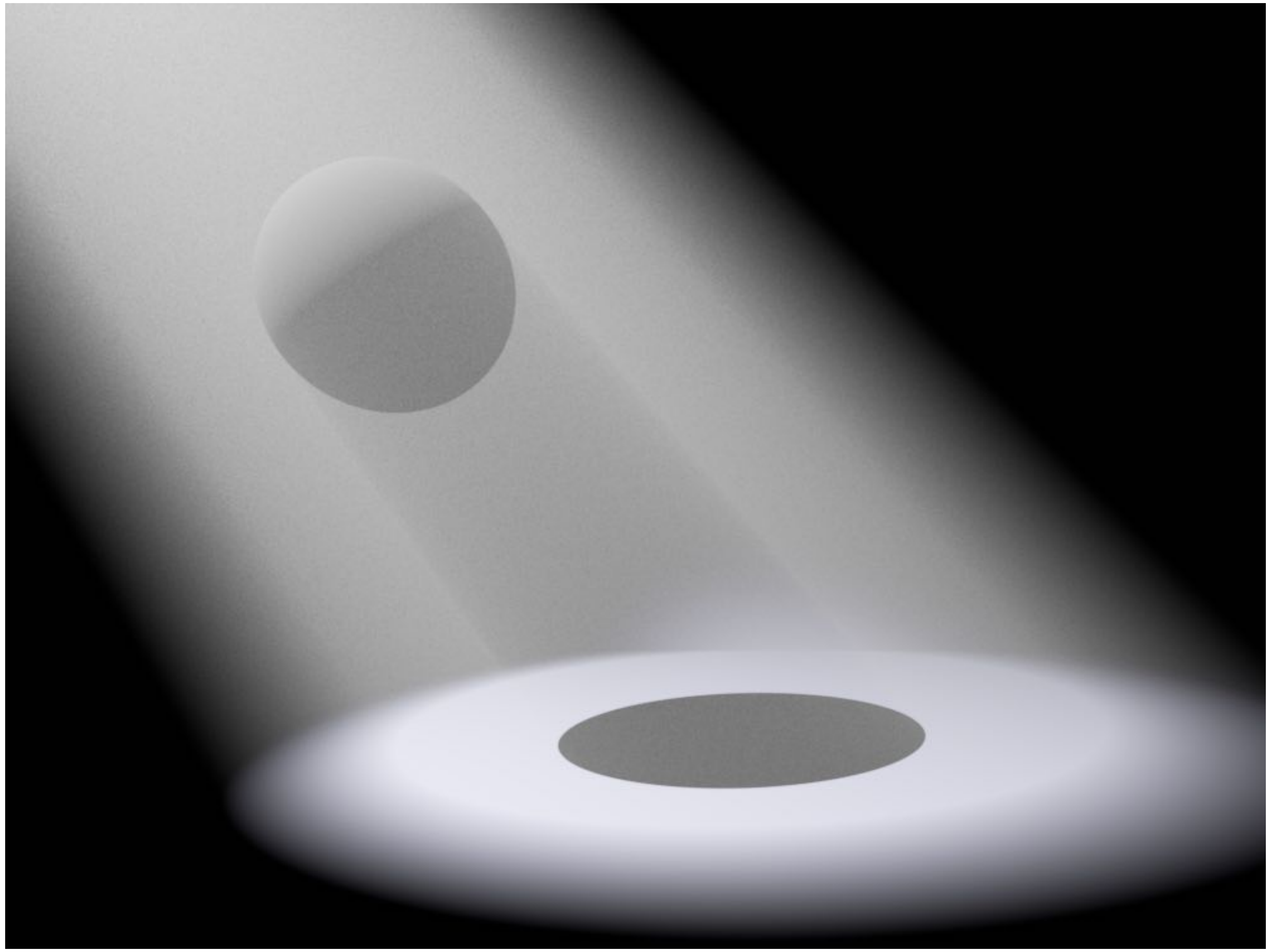
Computer Graphics: Rendering, Geometry, and Image Manipulation
Stanford CS248A, Winter 2024

Today's subject

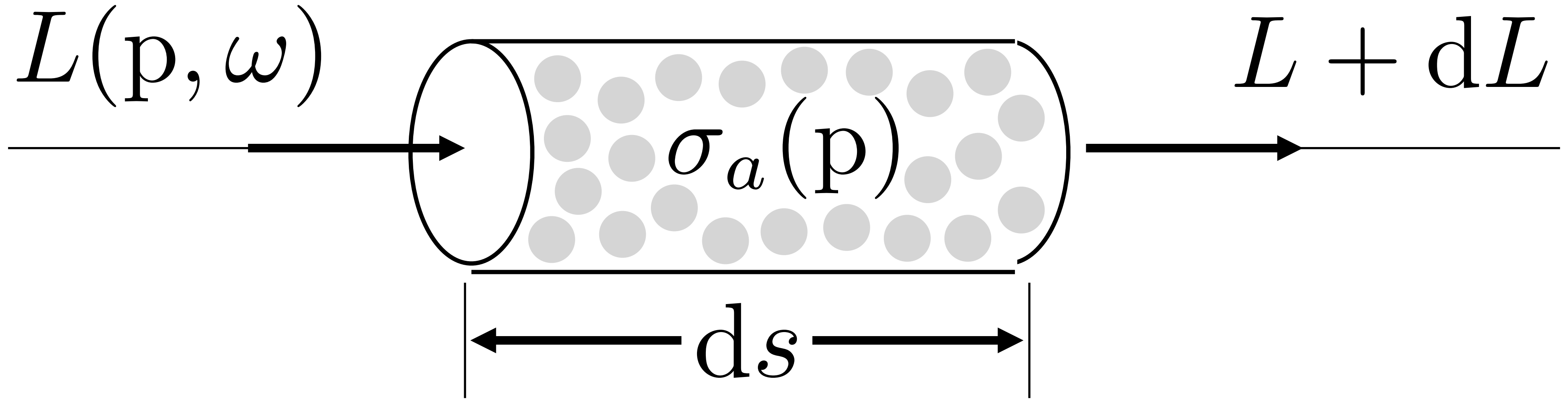
- Rendering from geometry representations that are not meshes
  - Volume rendering
  - Point rendering

- And their implications to modern progress in scene capture

Volumetric effects



Absorption in a volume

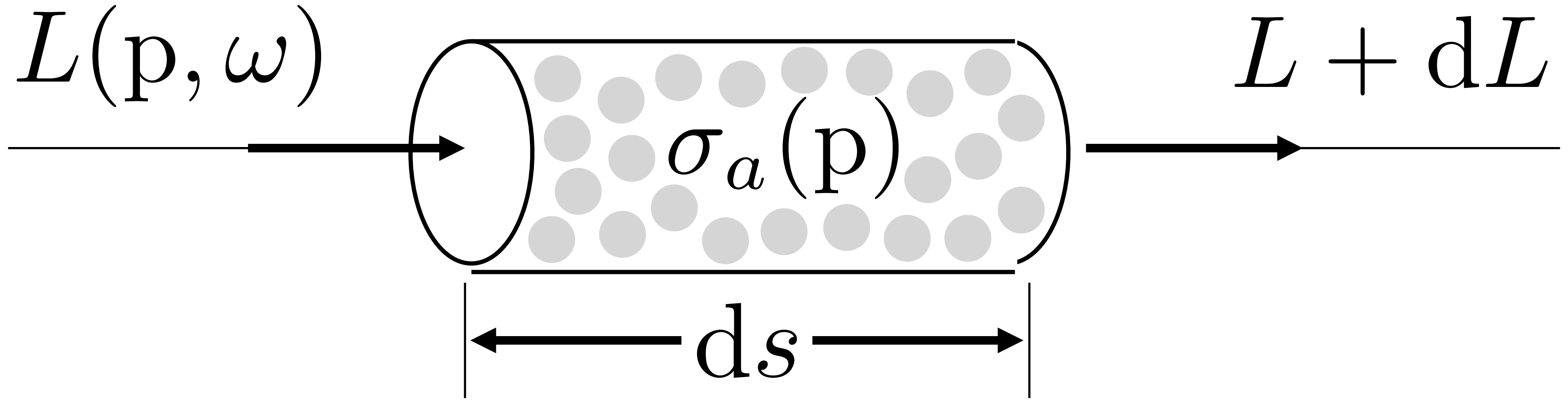


$$\mathrm{p} = (x, y, z)$$
$$\omega = (\phi, \theta)$$

$$\mathrm{d}L(\mathrm{p}, \omega) = -\sigma_a(\mathrm{p})\, L(\mathrm{p}, \omega)\, \mathrm{d}s$$
$$\frac{dL(\mathrm{p}, \omega)}{ds} = -\sigma_a(\mathrm{p}) L(\mathrm{p}, \omega)$$

- $L(\mathrm{p}, \omega)$ **radiance along a ray from *p* in direction ω**
- **Absorption cross section at point in space:** $\sigma_a(\mathrm{p})$
  - **Probability of being absorbed per unit length**
  - **Units: 1/distance**

Absorption in a volume



$$\mathrm{p} = (x, y, z)$$

$$\omega = (\phi, \theta)$$

$$\frac{dL(\mathrm{p},\omega)}{L(\mathrm{p},\omega)} = -\sigma_a(\mathrm{p})\mathrm{d}s$$

$$L(\mathrm{p}+s\omega,\omega) = \mathrm{e}^{-\int_0^s \sigma_\mathrm{a}(\mathrm{p}+s'\omega)\,\mathrm{d}s'} L(\mathrm{p},\omega) = T(s)\,L(\mathrm{p},\omega)$$

Transmittance: $T(s) = \mathrm{e}^{-\int_0^s \sigma_\mathrm{a}(\mathrm{p}+s'\omega,\omega)\,\mathrm{d}s'}$

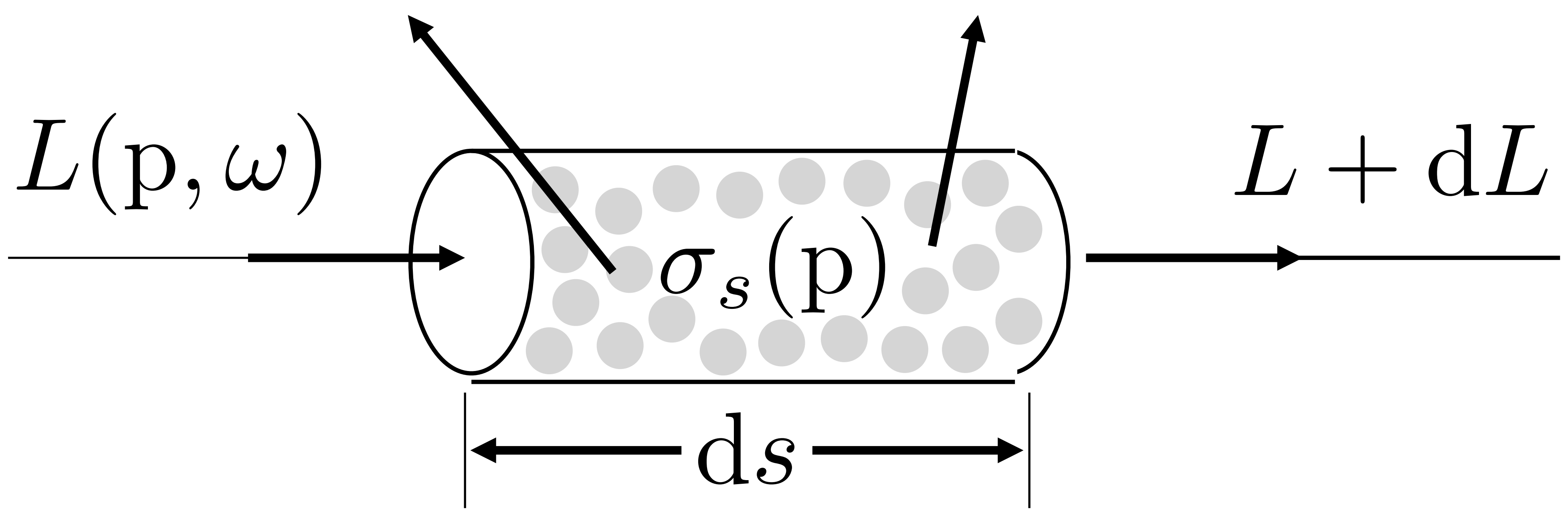
Absorption: lower density

Credit: Walt Disney Animation Studios

Absorption: higher density

Credit: Walt Disney Animation Studios

Out scattering



$$\mathrm{d}L(\mathrm{p}, \omega) = -\sigma_s(\mathrm{p})\, L(\mathrm{p}, \omega)\, \mathrm{d}s$$

- Scattering cross section at point in space: σ_s
  - Probability of being scattered per unit length
  - Units: 1/distance

Absorption and out scattering diminish radiance

Total cross section:

$$\sigma_t = \sigma_a + \sigma_s$$

$$\mathrm{d}L(\mathrm{p}, \omega) = -\sigma_t(\mathrm{p})\, L(\mathrm{p}, \omega)\, \mathrm{d}s$$

$$L(\mathrm{p} + s\omega, \omega) = T(s)\, L(\mathrm{p}, \omega)$$

Where total transmittance is:

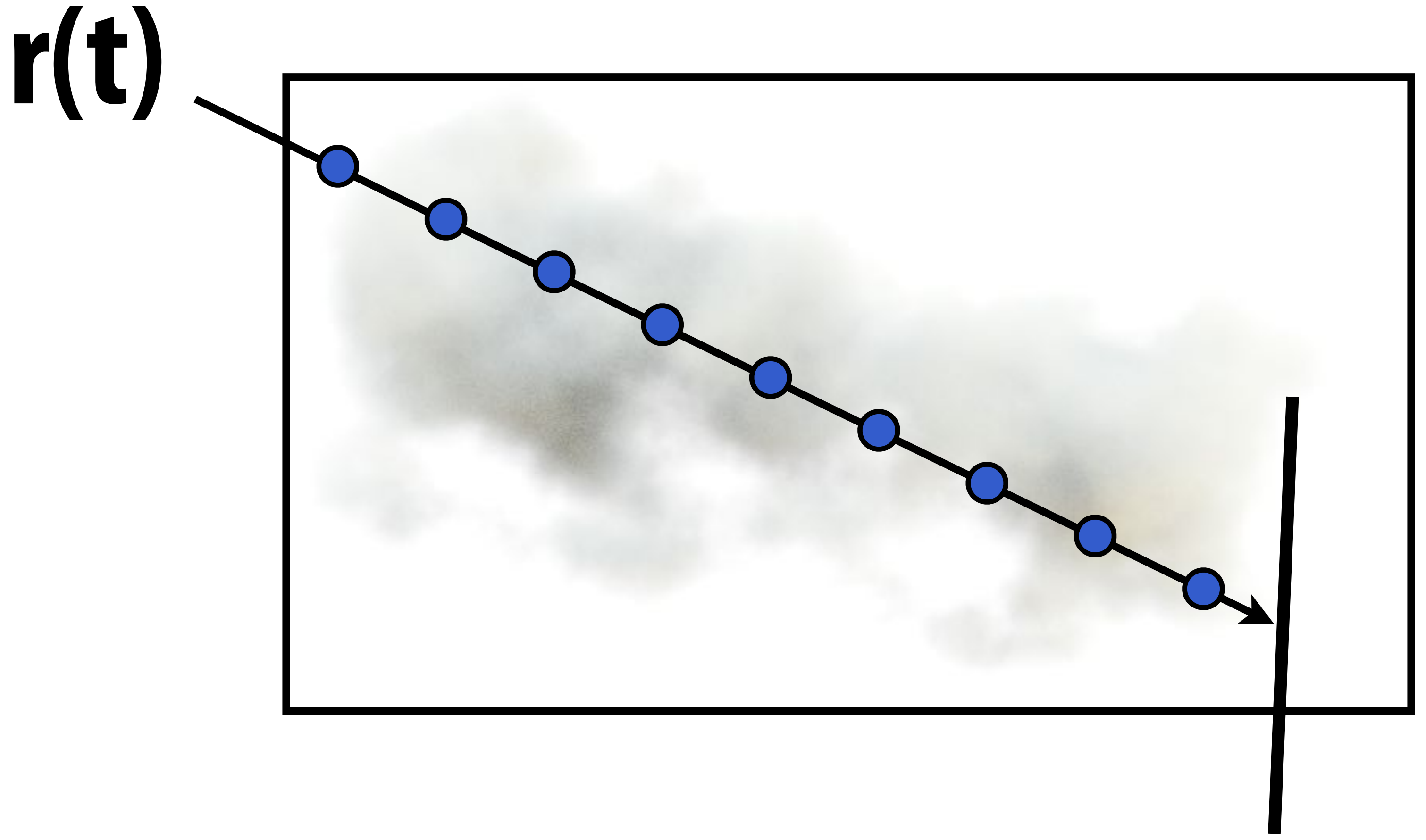
$$T(s) = \mathrm{e}^{-\int_0^s \sigma_{\mathrm{t}}(\mathrm{p}+s'\omega)\, \mathrm{d}s'} = e^{-\tau(s)}$$

$$\tau(s) = \int_0^s \sigma_t(\mathrm{p} + s'\omega)\, \mathrm{d}s'$$

"Optical distance" (from absorption and scattering)

Ray marching to compute transmittance

Step through volume in small steps



Given "camera ray" from point o in direction w….

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

And volume with density

$$\sigma(\mathrm{p})$$

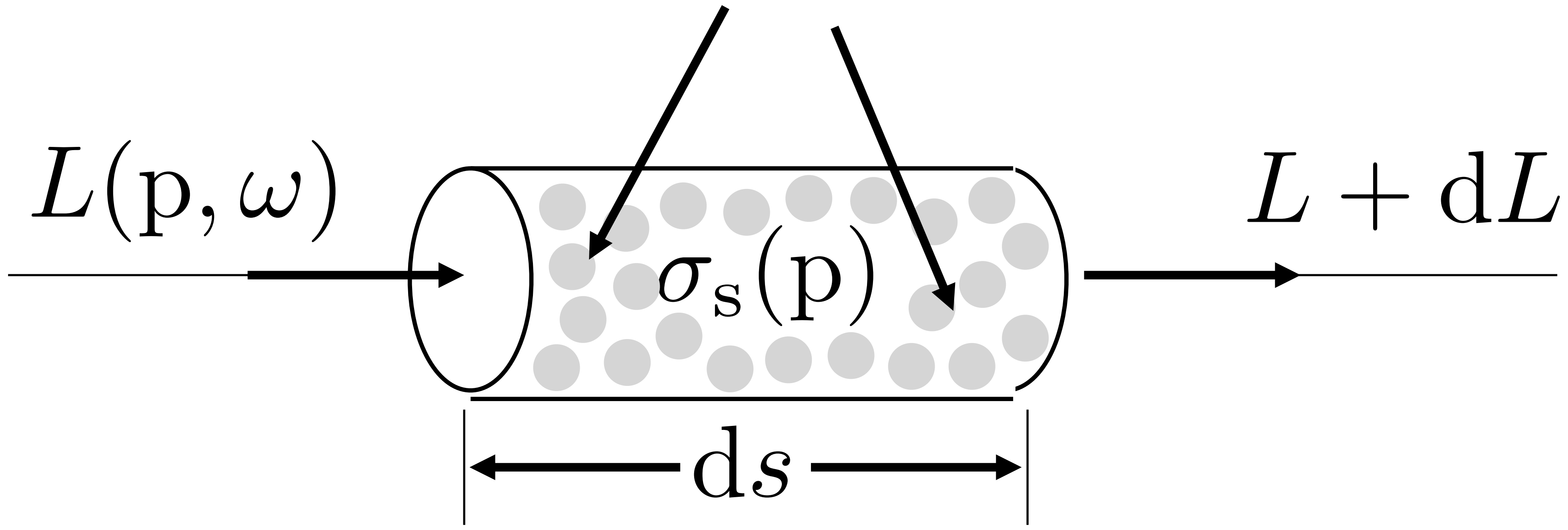
Estimate optical thickness as:

$$\tau(s) \approx \frac{s}{N} \sum_{i}^{N} \sigma_t(\mathrm{p}_i)$$

$$\mathrm{p}_i = \mathbf{o} + \frac{i + 0.5}{N} \omega$$

In scattering

- Light going in other directions also scatters into the direction ω
- In scattering increases radiance along ω



$$S(\mathrm{p}, \omega) = \sigma_s(\mathrm{p}) \int_{S^2} p(\omega' \to \omega)\, L(\mathrm{p}, \omega')\, \mathrm{d}\omega'$$

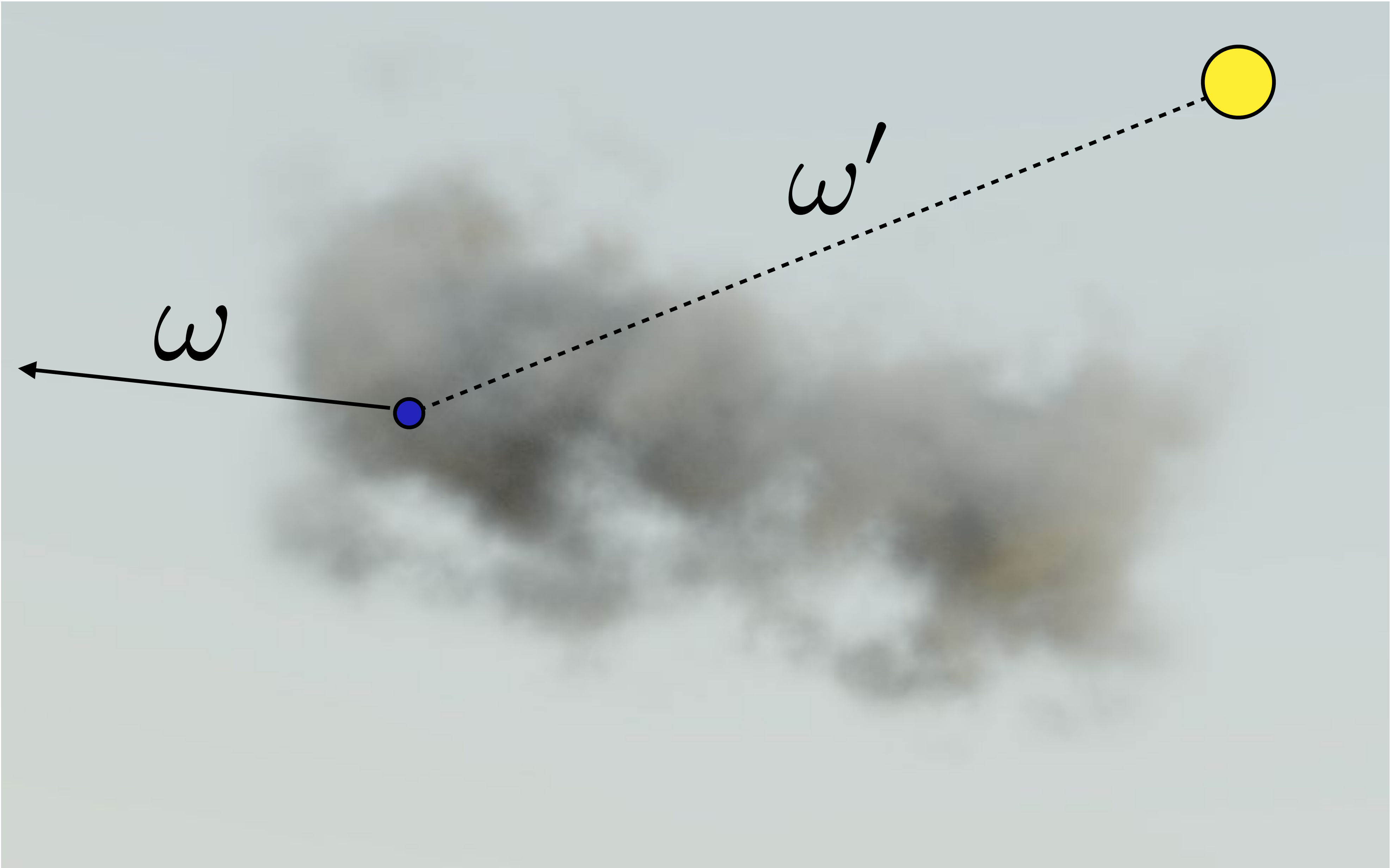
Phase function: $p(\omega' \to \omega)$

Energy conservation: $\int_{S^2} p(\omega' \to \omega)\, \mathrm{d}\omega' = 1$

Direct illumination in a volume

- **Can treat like direct illumination on a surface**
- **e.g., sample light sources (or from phase function's distribution)**

$$S_{\mathrm{d}}(\mathrm{p}', \omega) = \sigma_s(\mathrm{p}') \int_{S^2} p(\omega' \to \omega)\, L_{\mathrm{d}}(\mathrm{p}', \omega')\, \mathrm{d}\omega'$$

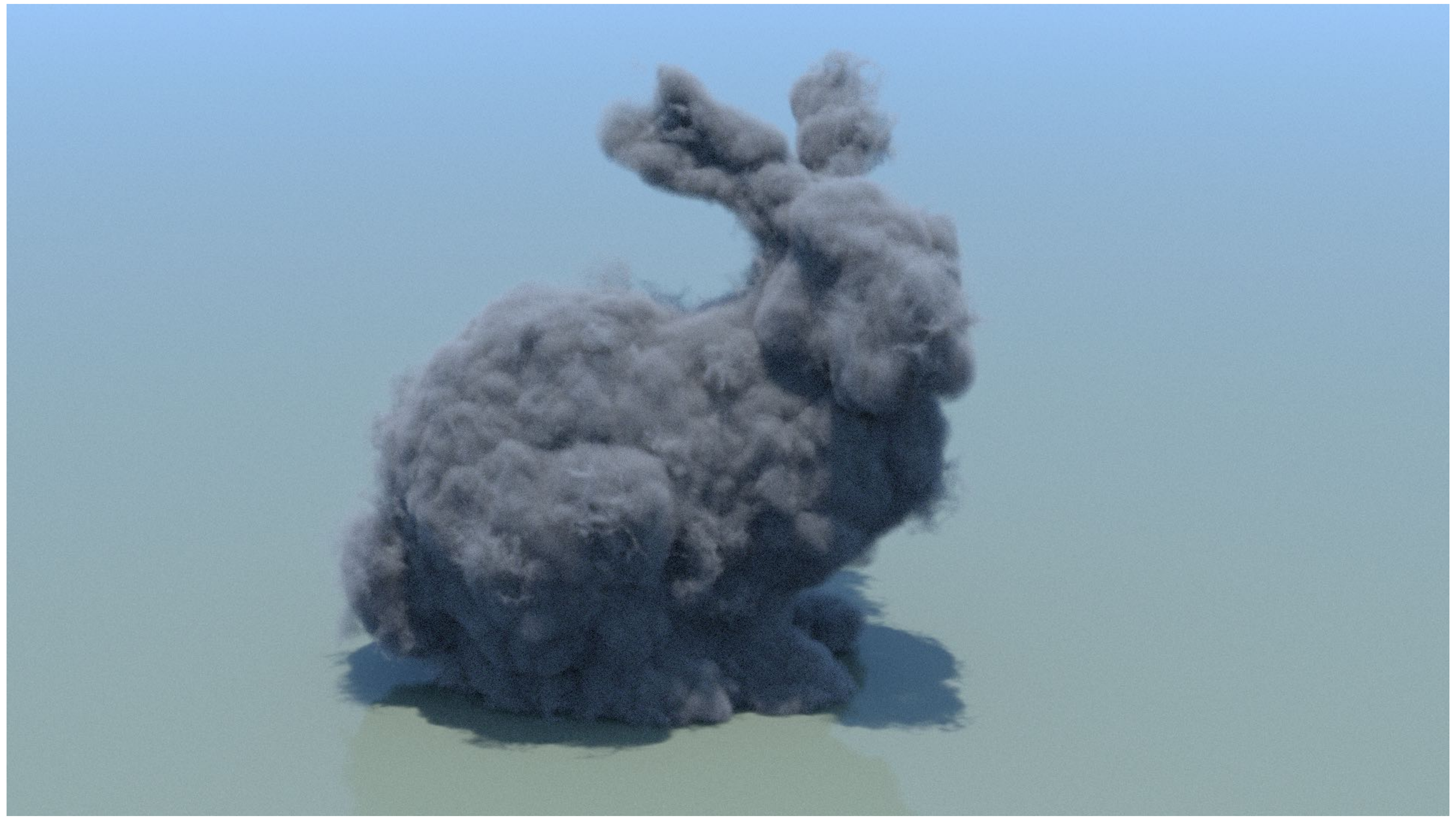


- **But computing direct lighting can be expensive**
- **Why?**
  - **Hint: requires more than a shadow ray**

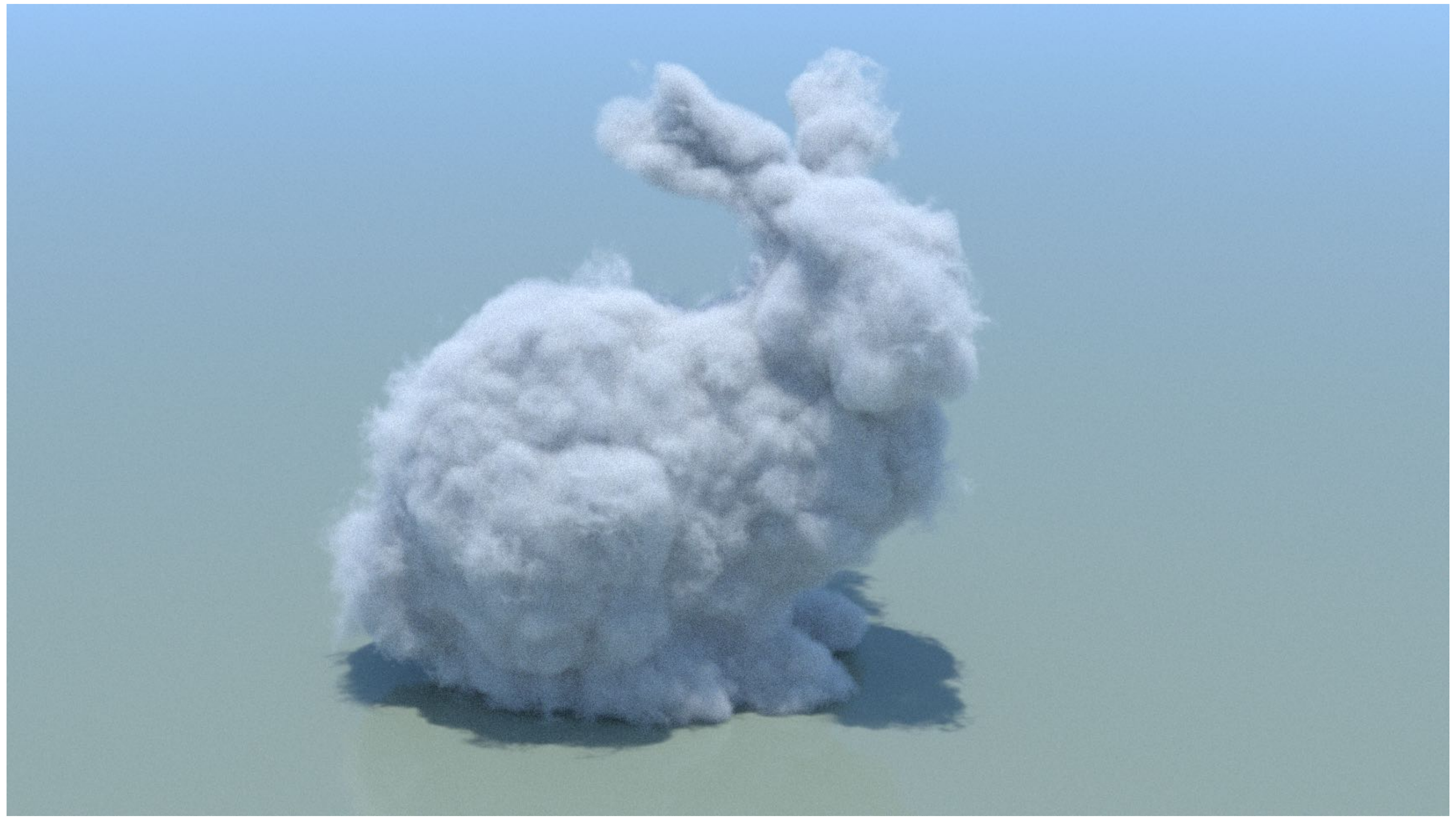
Single scattering

Multiple scattering (not discussed today)

- Appearance of volume is not just due to a single scattering event (light directly from light source scattering in direction of eye)
- Light scatters many times in the volume before existing in the direction of the eye
- Advanced rendering topic: Monte Carlo estimate of multiple scattering events ("volume rendering equation")



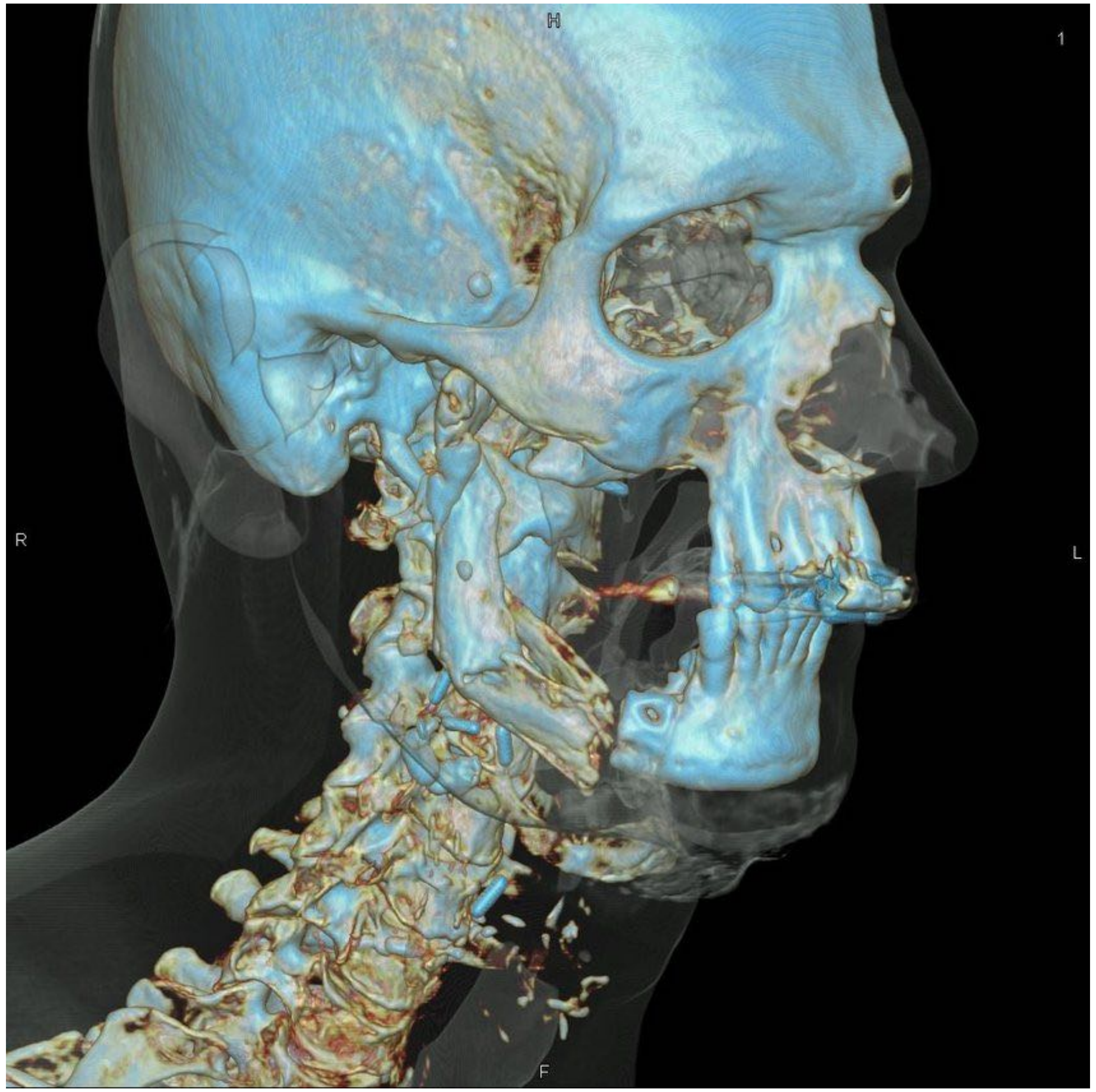
Single scattering



Multiple scattering

Let’s ignore lighting for a moment

Consider representing a scene as a volume



Volume rendered CT scan

Volume density and “color” at all points in space.

$$\sigma(\mathrm{p})$$

$$c(\mathrm{p}, \omega) = c(x, y, z, \phi, \theta)$$

The reflectance off surface at point p in direction ω

Volume rendered scene
(Mildenhall et al.)

Credit: Taubmann et al. , Siemens Healthineers

Rendering volumes

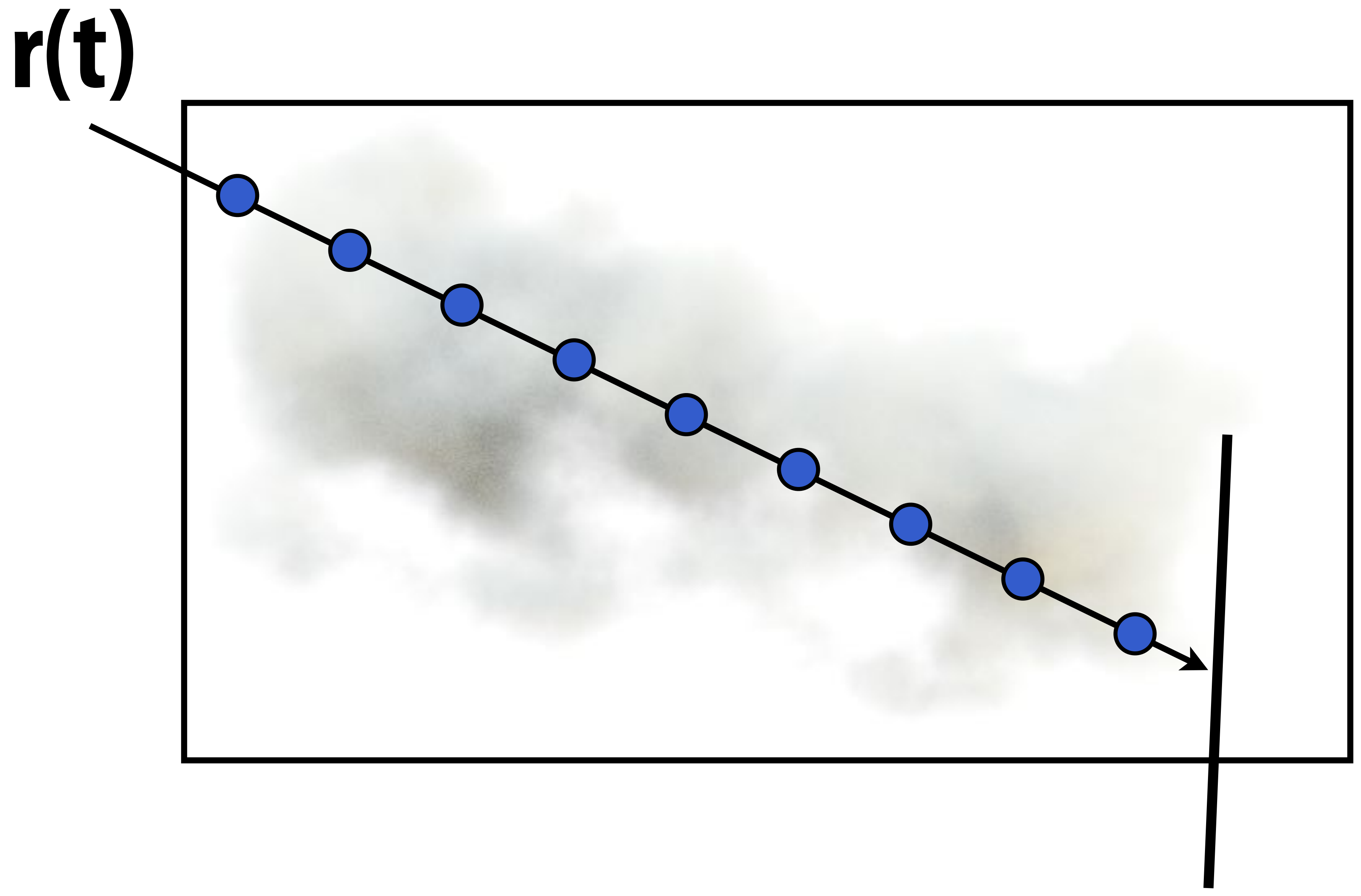
Given "camera ray" from point o in direction w….

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

And volume with density and directional radiance.

$$\sigma(\mathrm{p})$$
$$c(\mathrm{p}, \omega)$$

Volume density and color at all points in space.

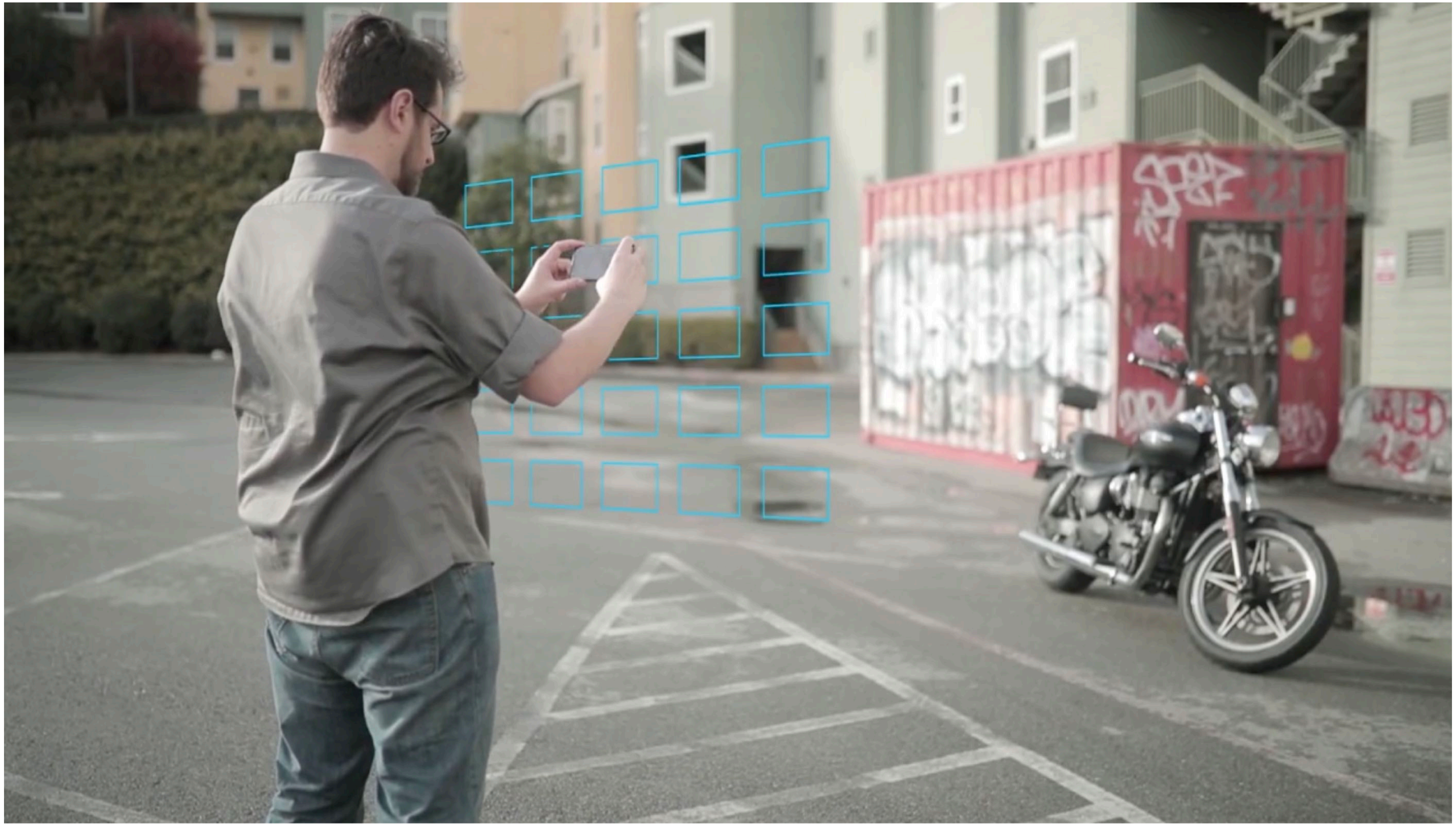


Step through the volume to compute radiance along the ray.

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

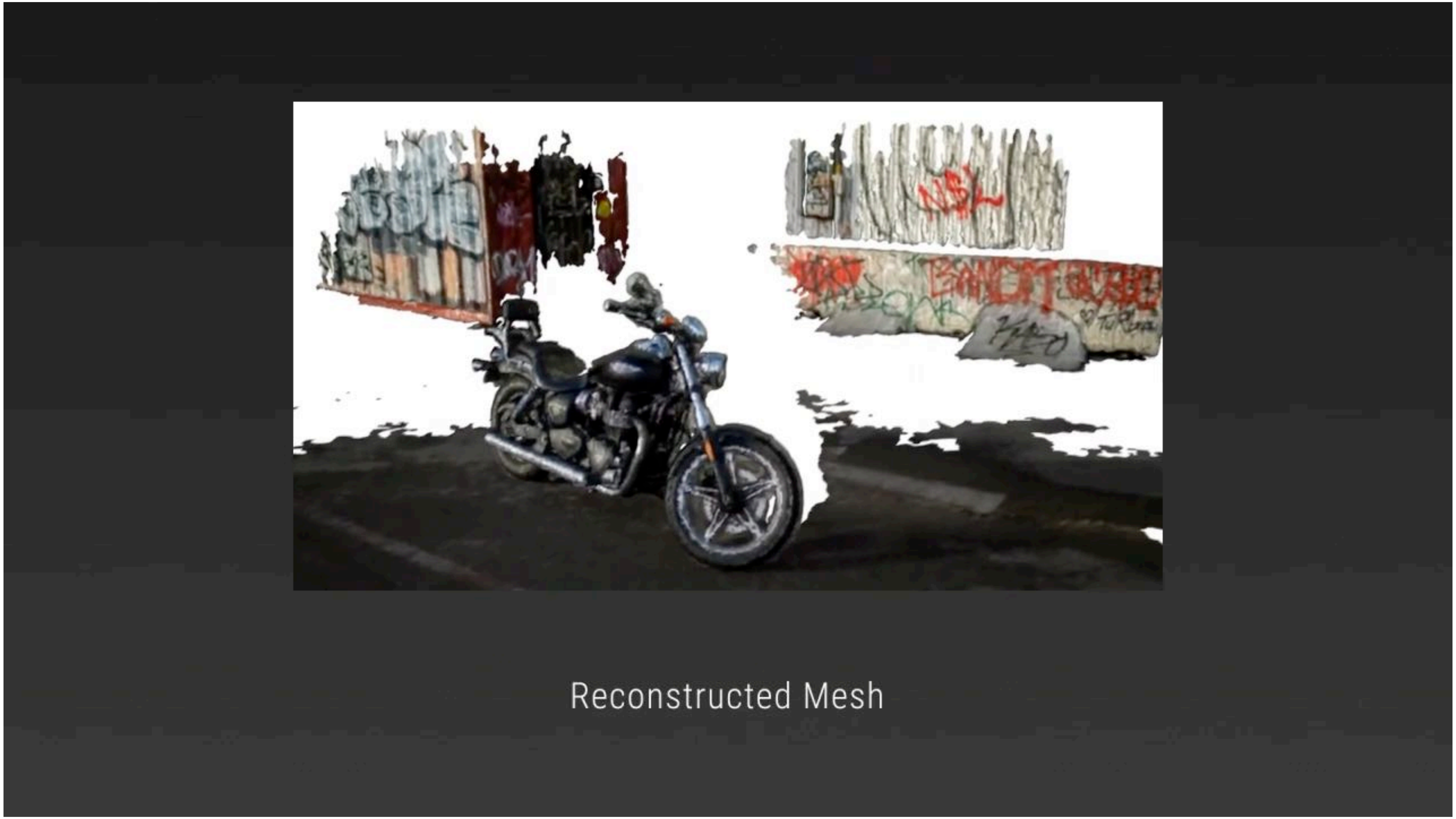
An interesting task

- Given a collection of photographs (from known camera viewpoints)
- Compute a 3D reconstruction of the scene (surface locations + color at each point on surface)



Credit: Mildenhall 2019

Estimating mesh geometry is tricky



Credit: Mildenhall 2019

Re-interest in using volume rendering (circa 2018)

- Let's just drop this triangle-based representation entirely, it's much simpler (and more versatile when it's unclear what the geometry is anyway) to emit a single volumetric representation

Credit: Lombardi 2019

Regular 3D grid representation of a volume

- Dense 3D grid
  - V[i,j,k] = rgba

- Note, this representation treats surface as diffuse, since: $c(\mathrm{p}, \omega) = c(\mathrm{p})$

- Would need σ[i,j,k] and c[i,j,k,phi,theta] to represent directional distribution of color

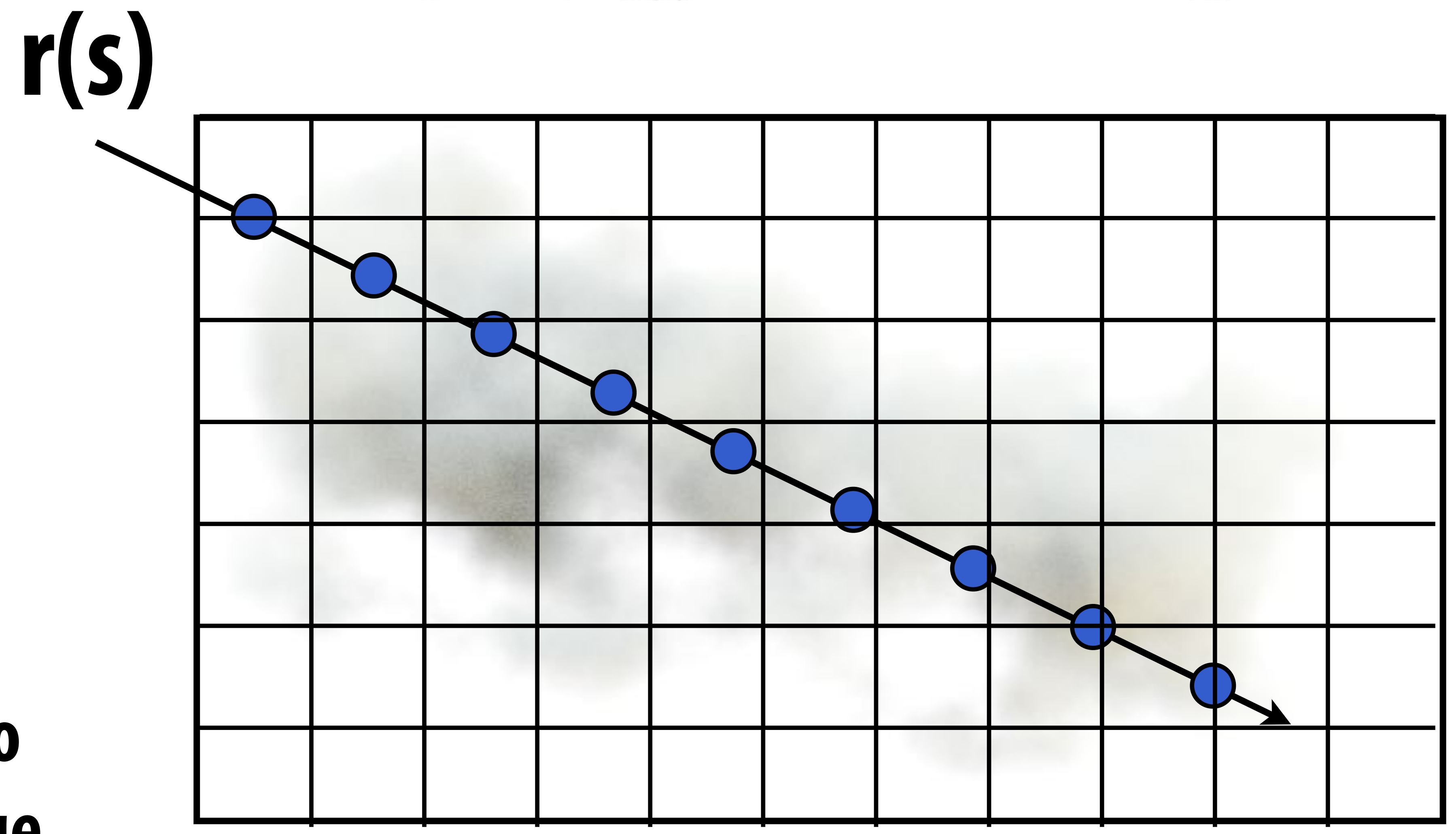
Credit: Voxel Ville NFT (voxelville.io)

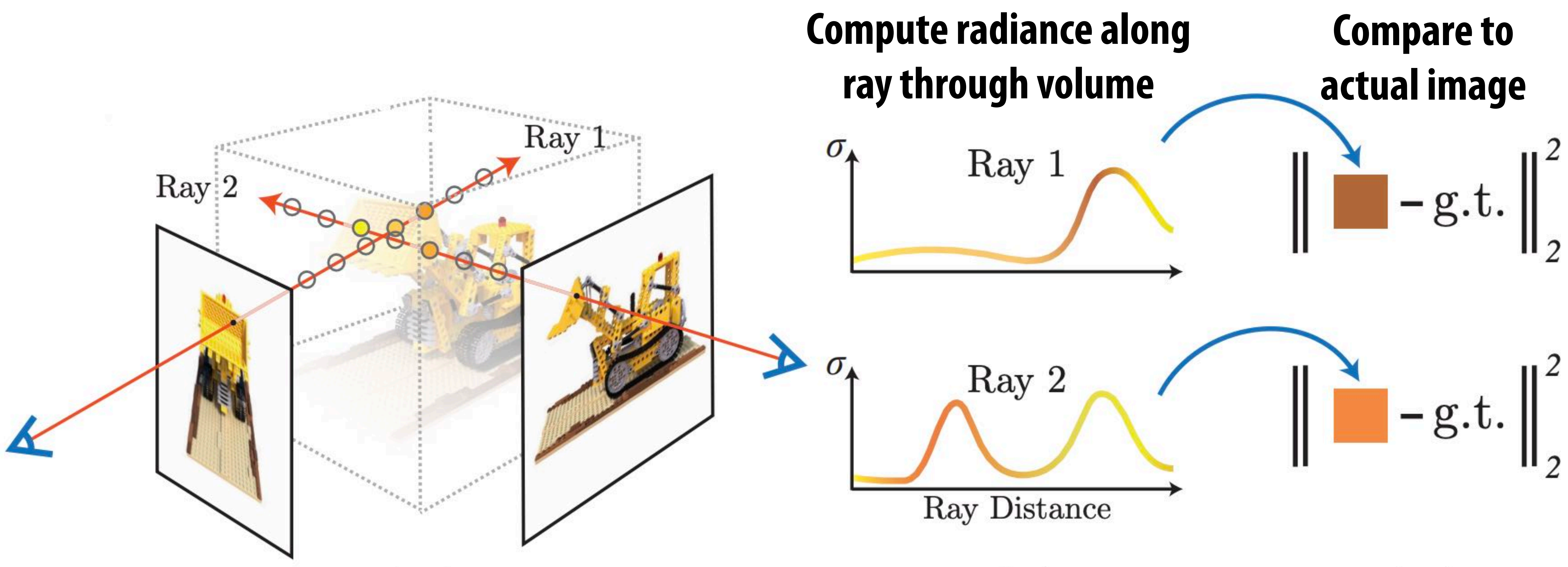
Optimizing volumes

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

Idea: optimize volume values (opacity and color) so that C(r) matches that of photos.

For many rays…. trace through volume… see if the result matches the photo… use error to update volume opacity/color values





Regular 3D grid representation of a volume

Consider storage requirements:
1024^3 cells

Ignore directional dependency: rgbσ 4 bytes/cell
~ 4GB

Now consider directional dependency of color
on (ϕ, θ) ... much worse storage cost

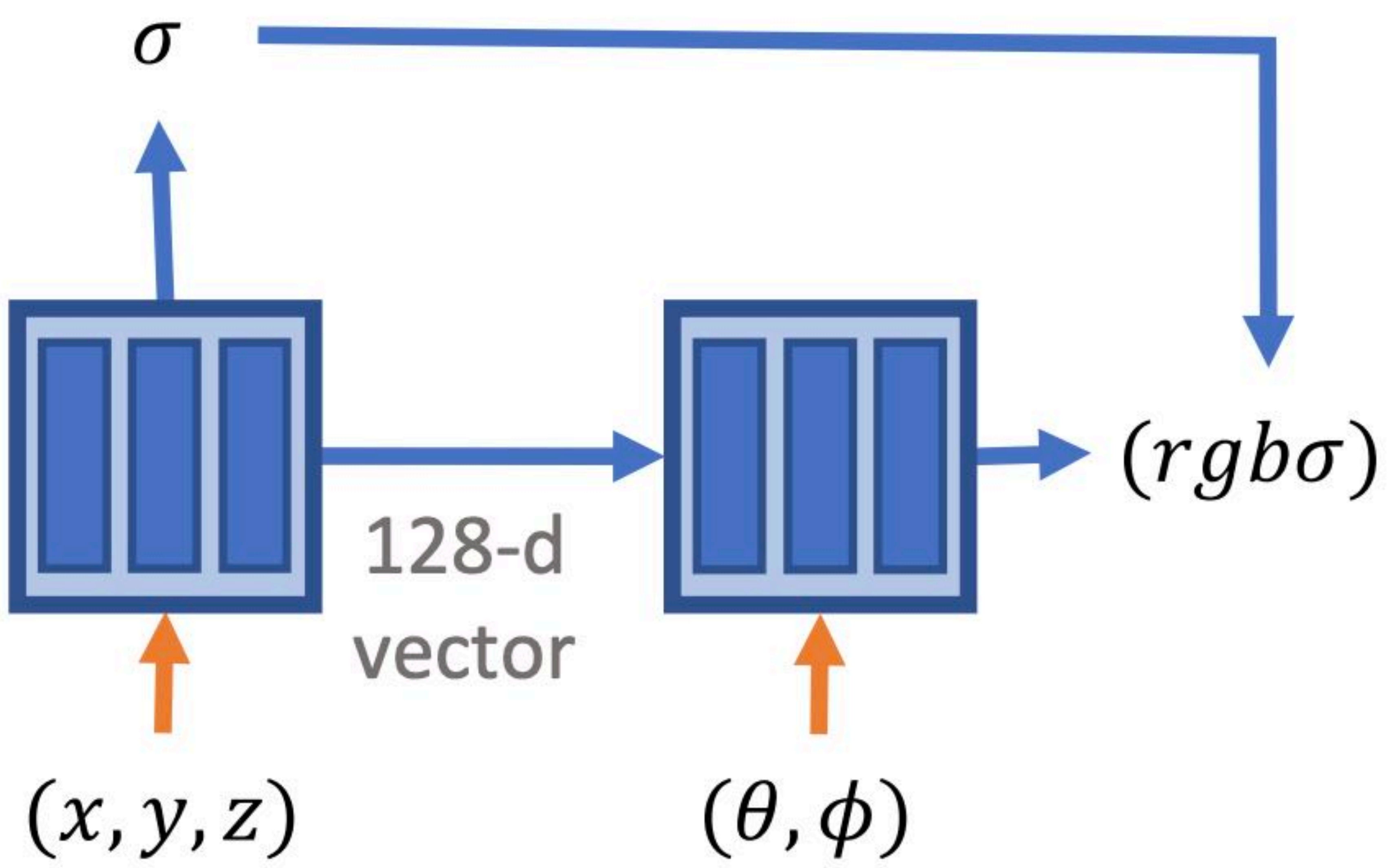
Typical challenge:
limited resolution

Credit: Voxel Ville NFT (voxelville.io)

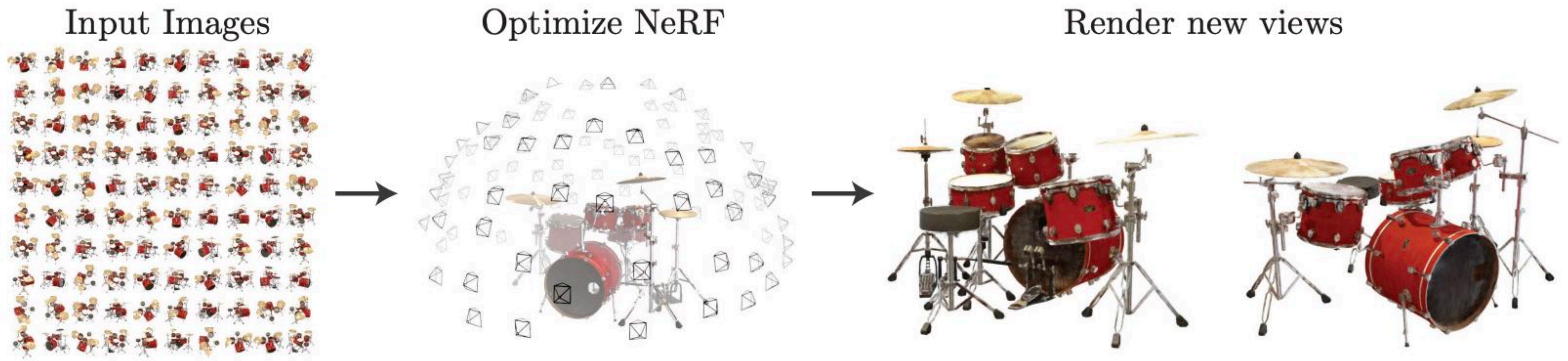
Learning (compressed) representations

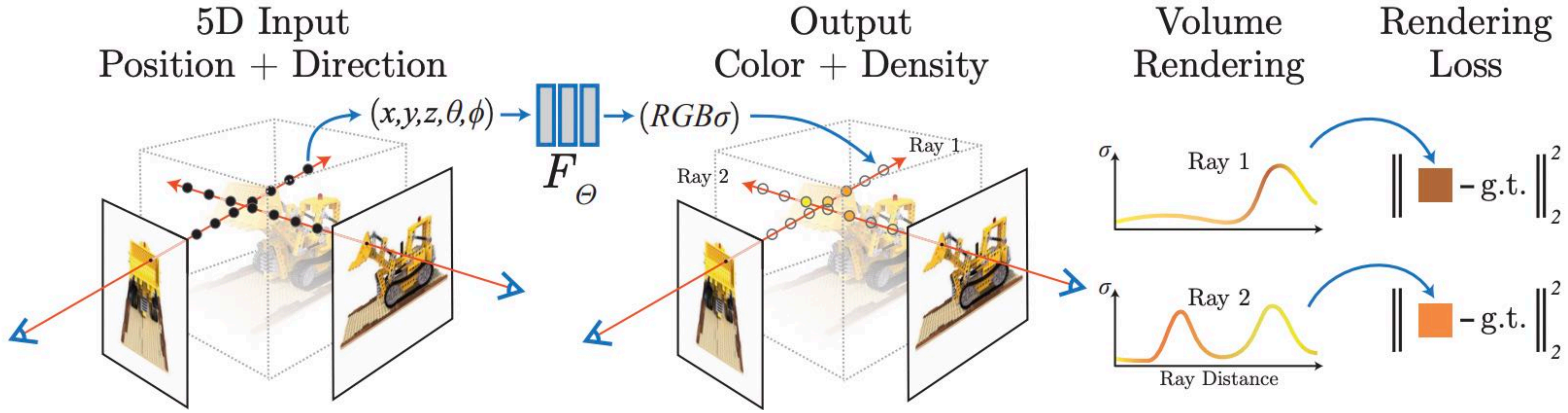
Why not just learn an approximation to the continuous function that matches observations from different viewpoints?

$$(\mathrm{p}, \omega) \rightarrow \boxed{F_\theta(\mathrm{p}, \omega)} \rightarrow \begin{matrix} \sigma(\mathrm{p}) \\ c(\mathrm{p}, \omega) \end{matrix}$$



Learning neural radiance fields (NeRF)





Key idea: differentiable volume renderer to compute dC/d(color)d(opacity)

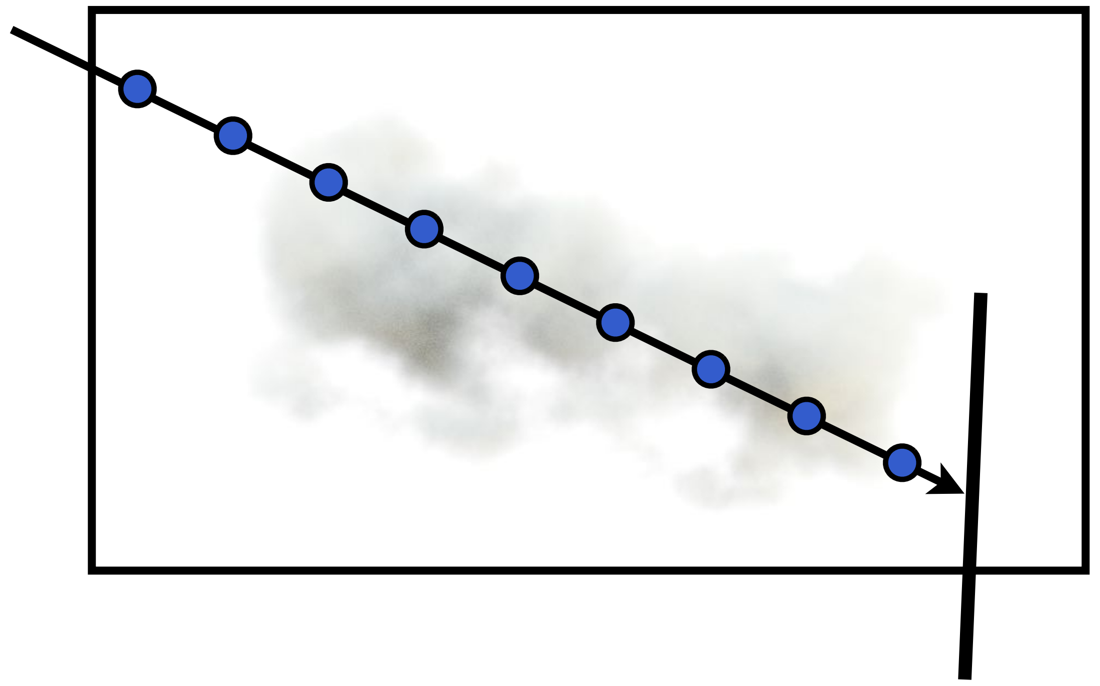
Great visual results!


Credit: Mildenhall 2023

What just happened?

- Continuous coordinate-based representation vs regular grid: DNN "learns" how to use its weights to produce high-resolution output where needed… given input data

- Compact representation: trades-off space for expensive rendering
  - Good: a few MBs = effectively very high-resolution dense grid
  - Bad: must evaluate DNN every step during ray marching
    - And the DNN is a "big" MLP (8-layer x 256) ← MLP must do real work to associate weights with 5D locations
  - Bad: must step densely (because we don't know where the surface is)



- Compact representation: optimization can learn to interpolate views despite complexity of volume density and radiance function
  - Only prior is the separation into positional σ and directional rgb
  - Training time: hours to a day to learn a good NeRF

Improving rendering performance

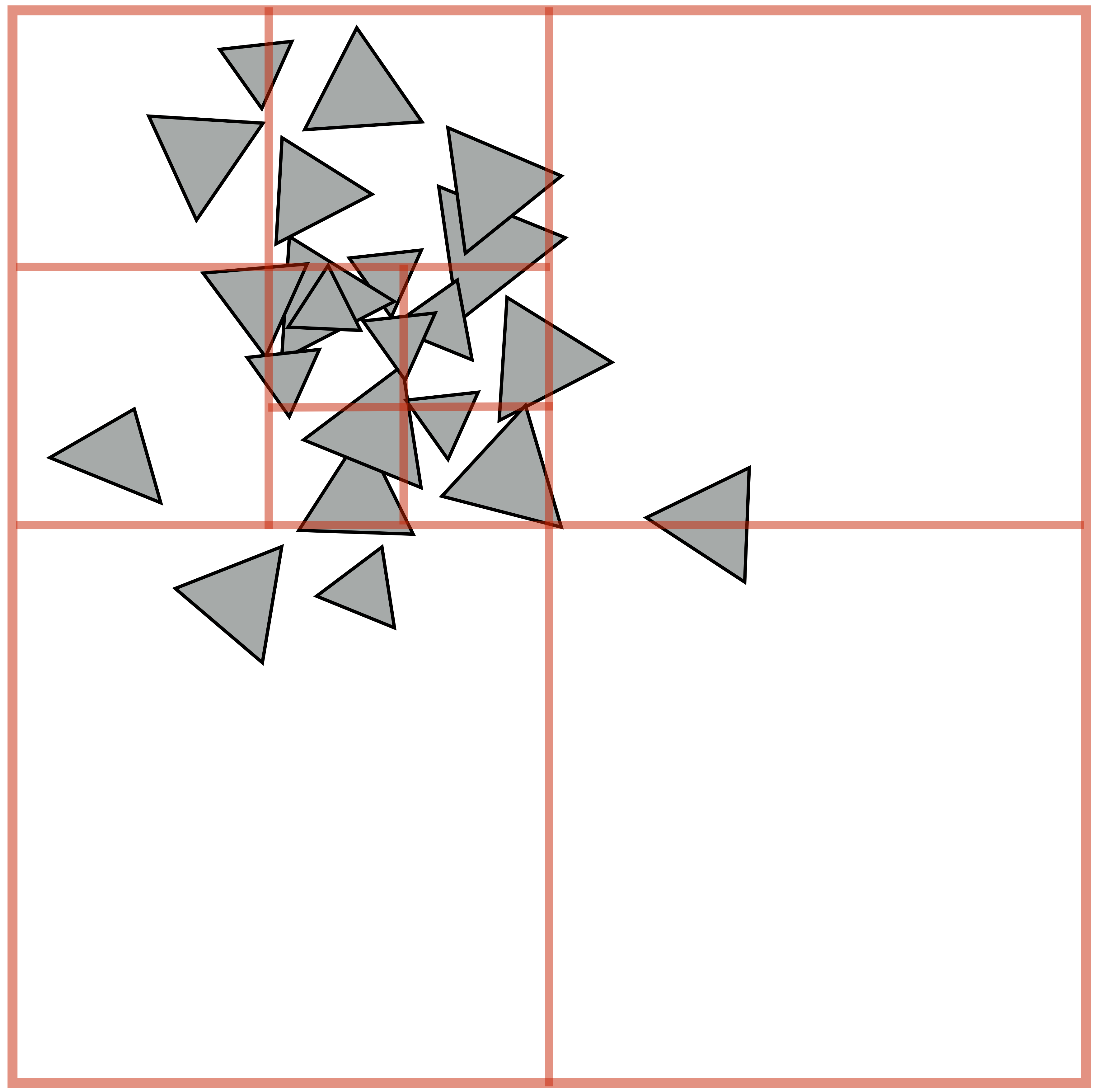
- Main idea: move to a different point in the compression-compute trade-off space

- Main ideas:
  - Avoid stepping densely through empty space (costly to evaluate the DNN to find density = 0)
  - Shrink the size of the DNN
  - Avoid evaluating the DNN when you can

Recall quad-tree / octree

Quad-tree: nodes have 4 children (partitions 2D space)

Octree: nodes have 8 children (partitions 3D space)



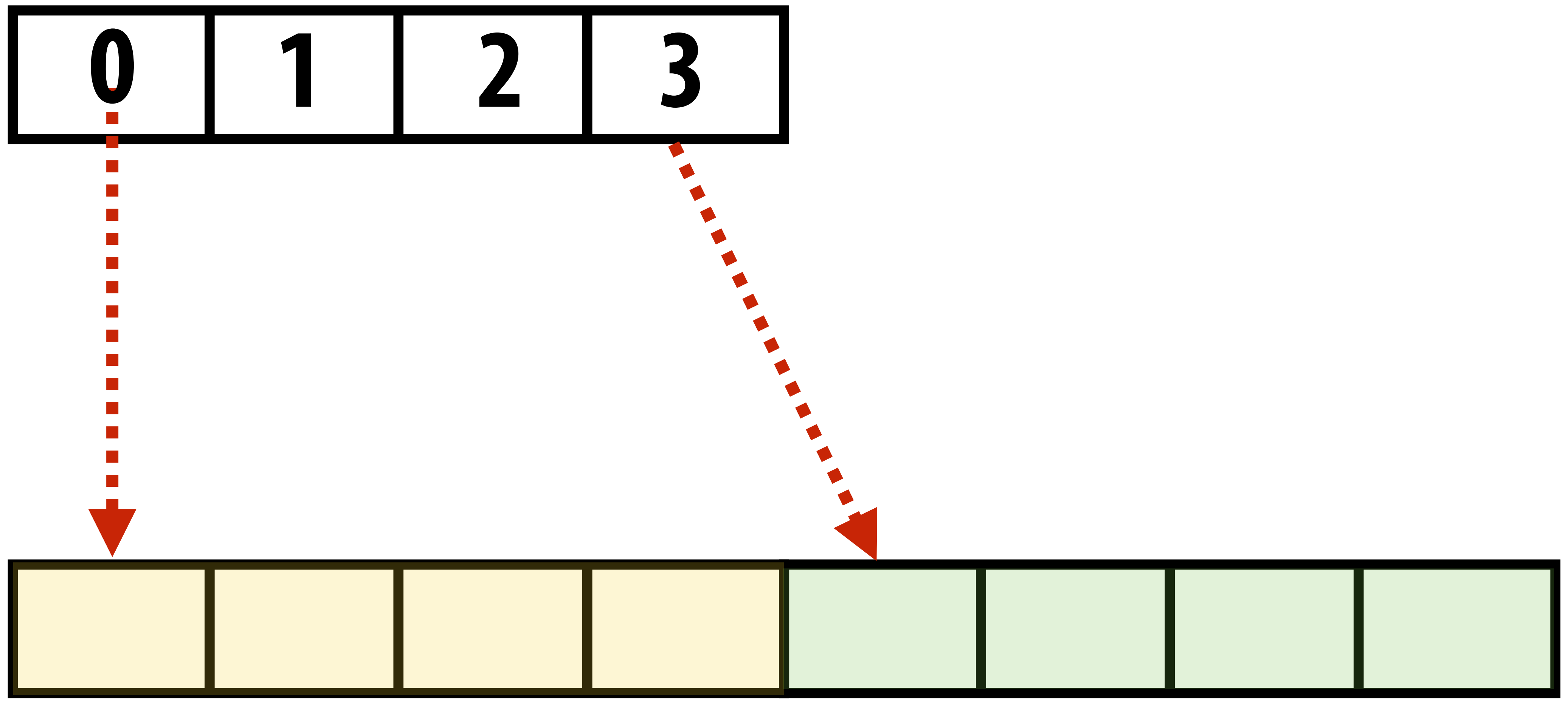
Like uniform grid: easy to build (don't have to choose partition planes)

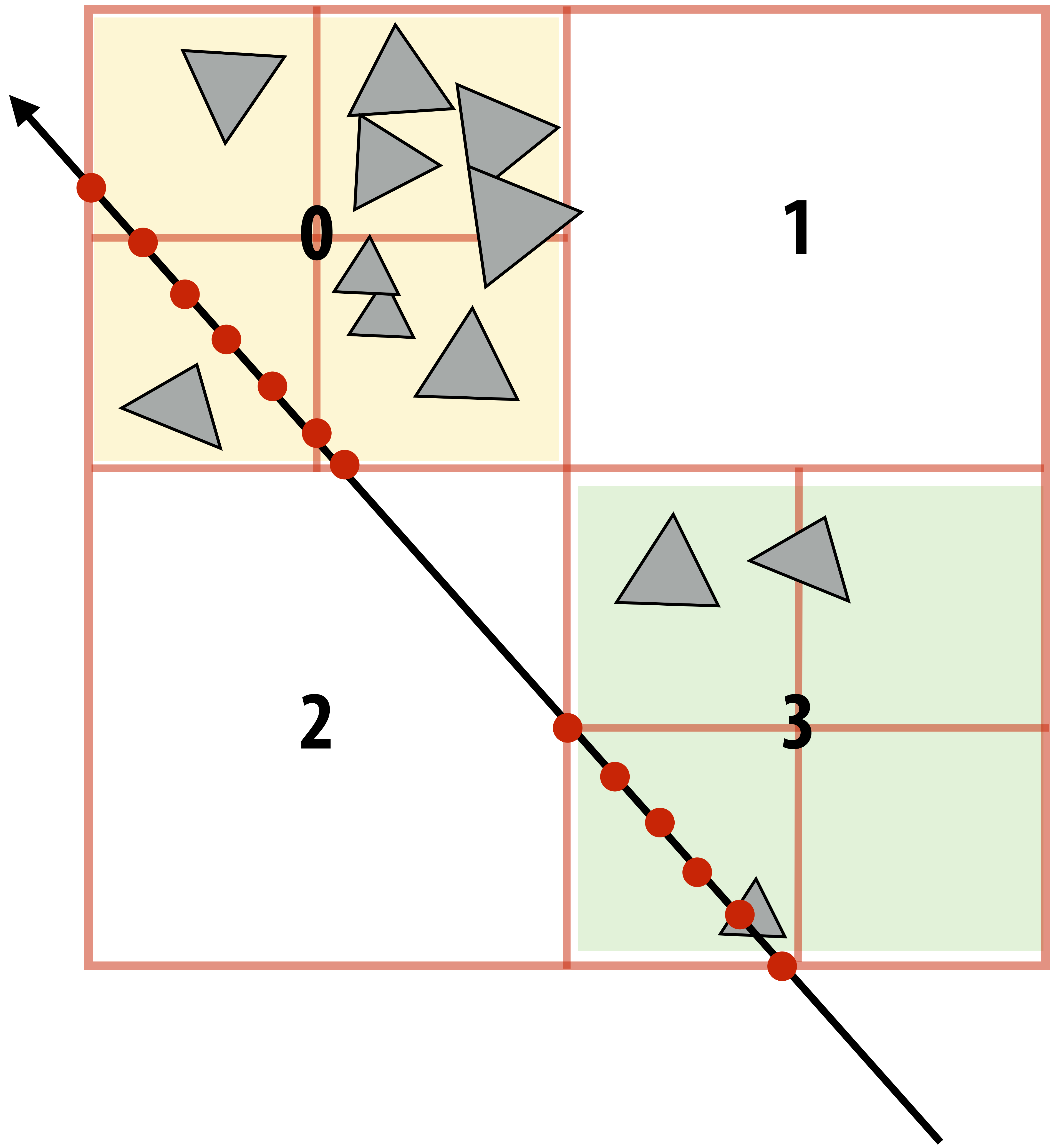
Has greater ability to adapt to location of scene geometry than uniform grid.

Simple two-level sparse quad tree

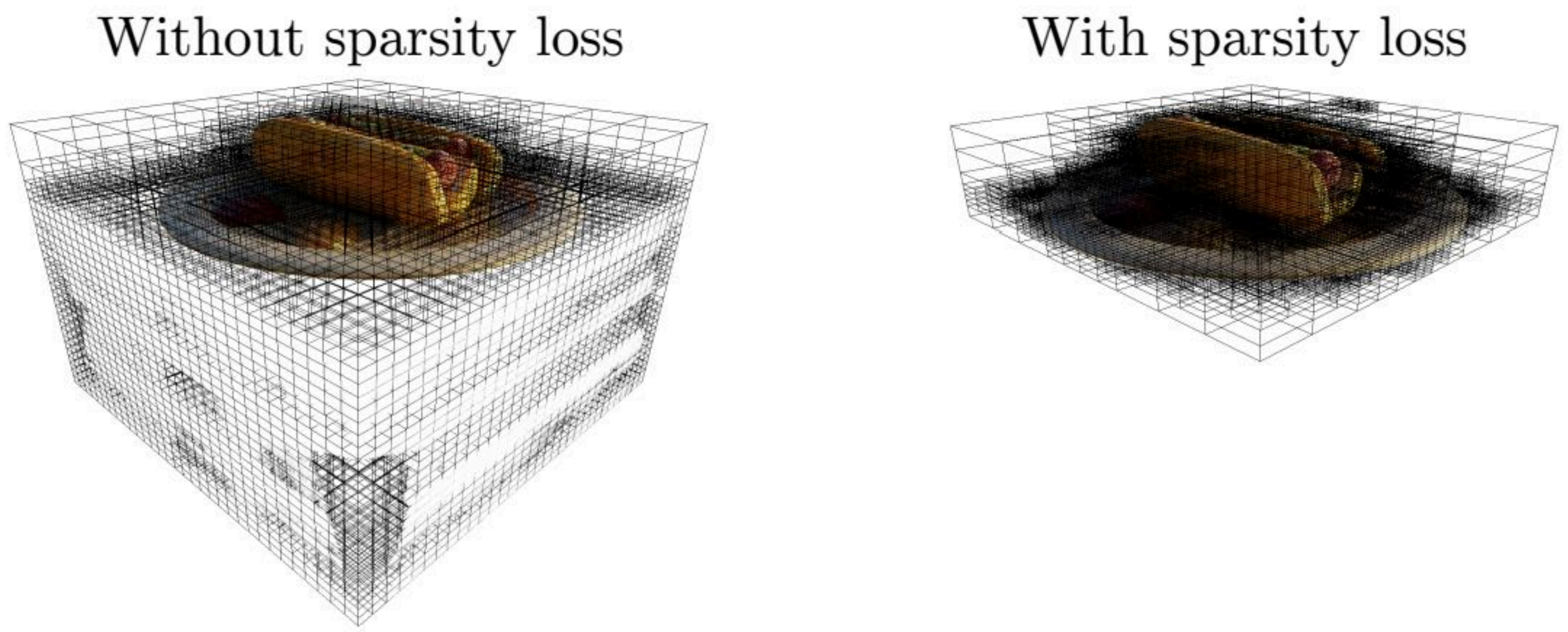
Quad-tree: nodes have 4 children (partitions 2D space)

Octree: nodes have 8 children (partitions 3D space)



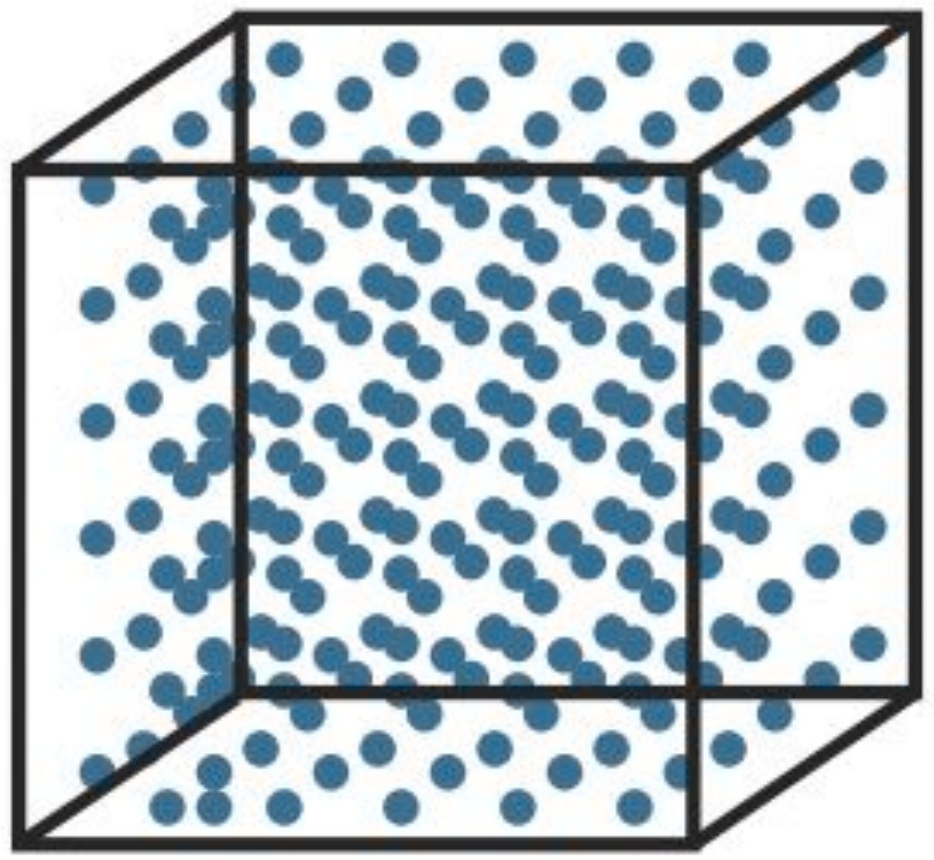


Let's just run optimization for a bit…

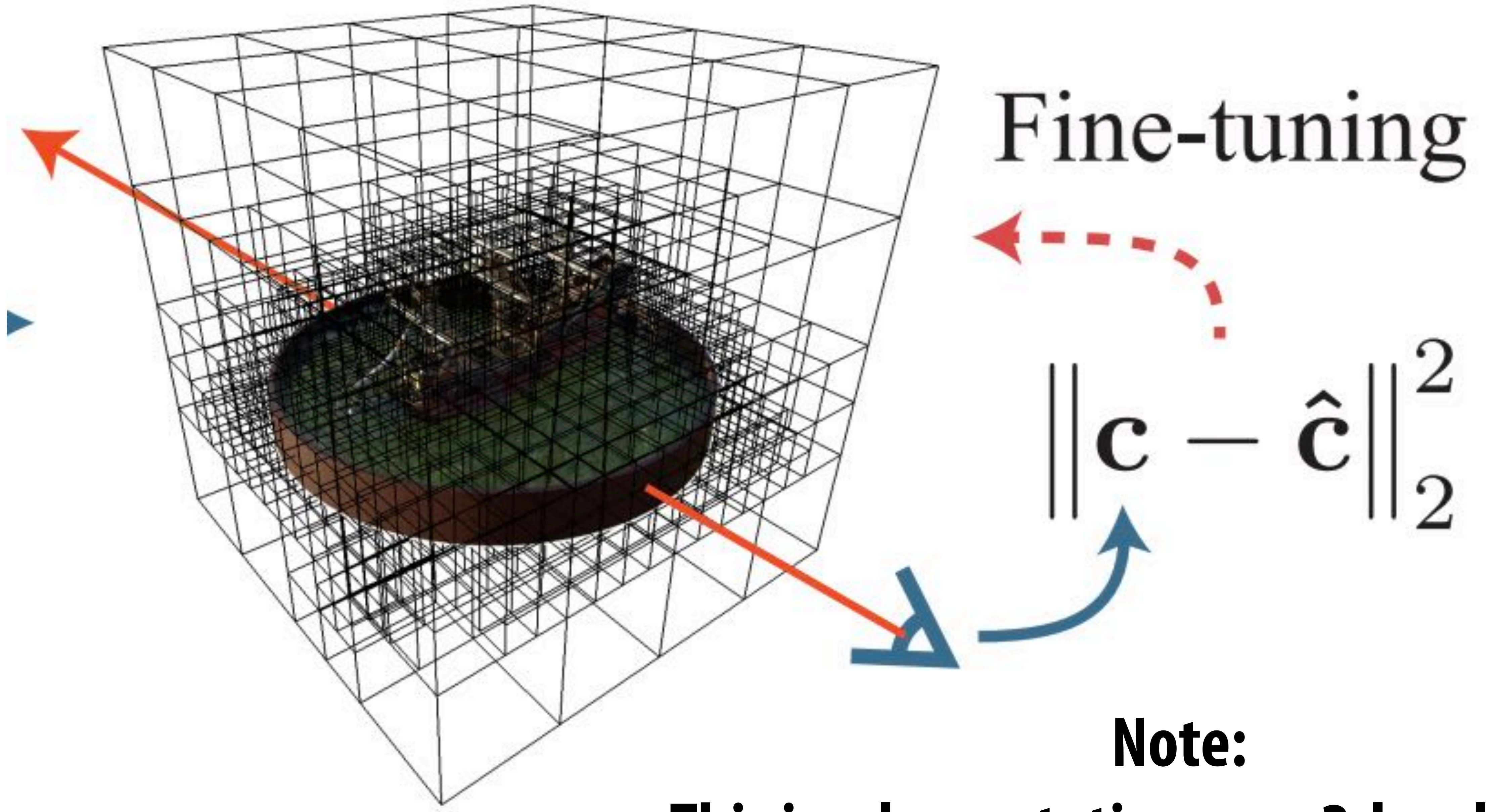


- Optimization will push some opacity values to 0
- DNN tells us where the empty space is!

- Then convert dense opacity grid to an octree representation that's more efficient to render from…
- With the octree structure *fixed*, we can continue to optimize color/density at leaves



Use the initial MLP to densely sample volume
(Find the empty space that's used to build the octree)

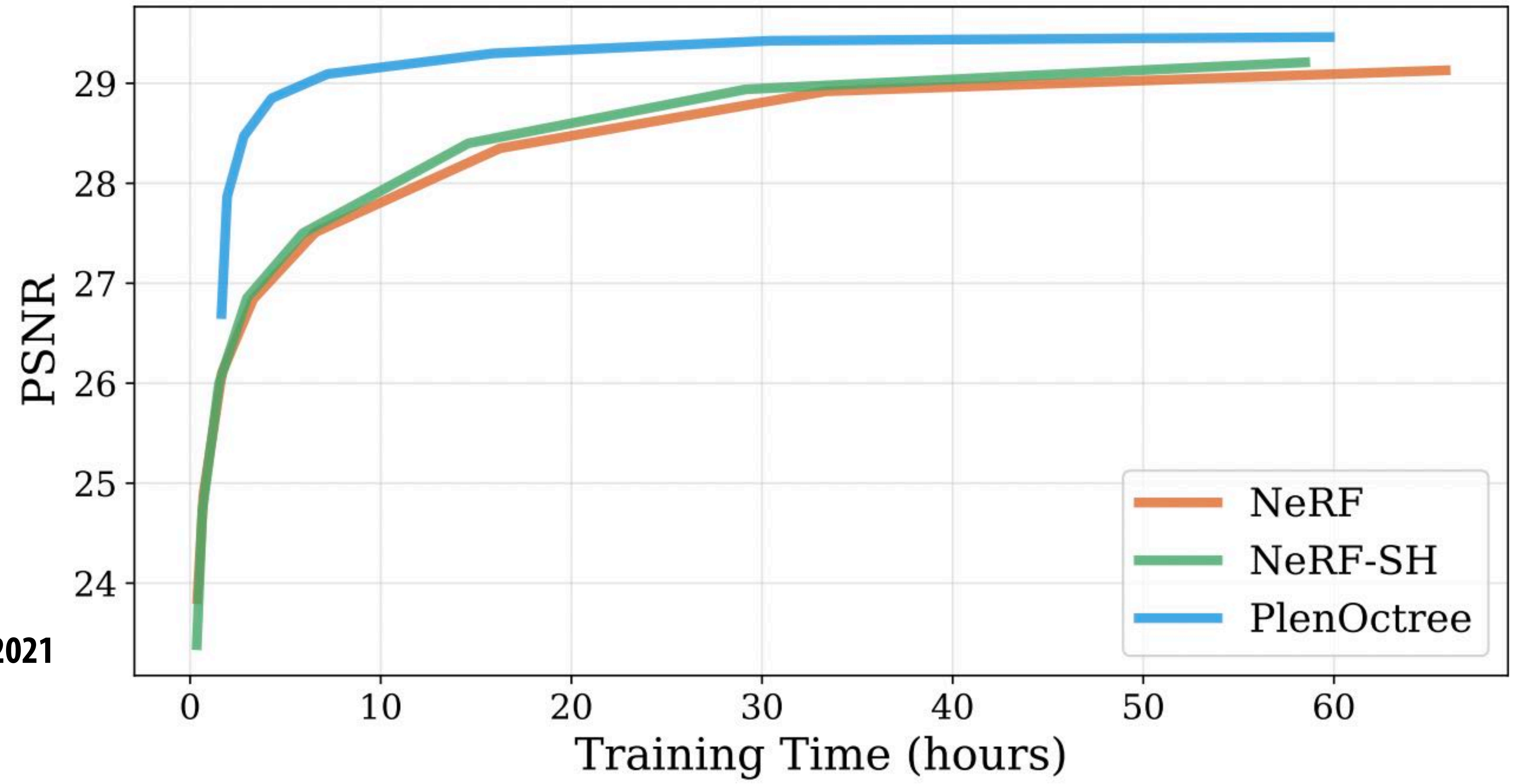


Note:
This implementation uses 2-level octreee

Credit: Yu 2021

What just happened?

- We performed initial training… a la original NeRF
- Once we get a sense of where the empty space is, we add a traditional acceleration structure to replace the “big” DNN. Can use used a little DNN at leaves.
- That structure speeds up rendering (a lot), and also speeds up “fine tuning” training, since the initial “big” DNN need not be trained to convergence

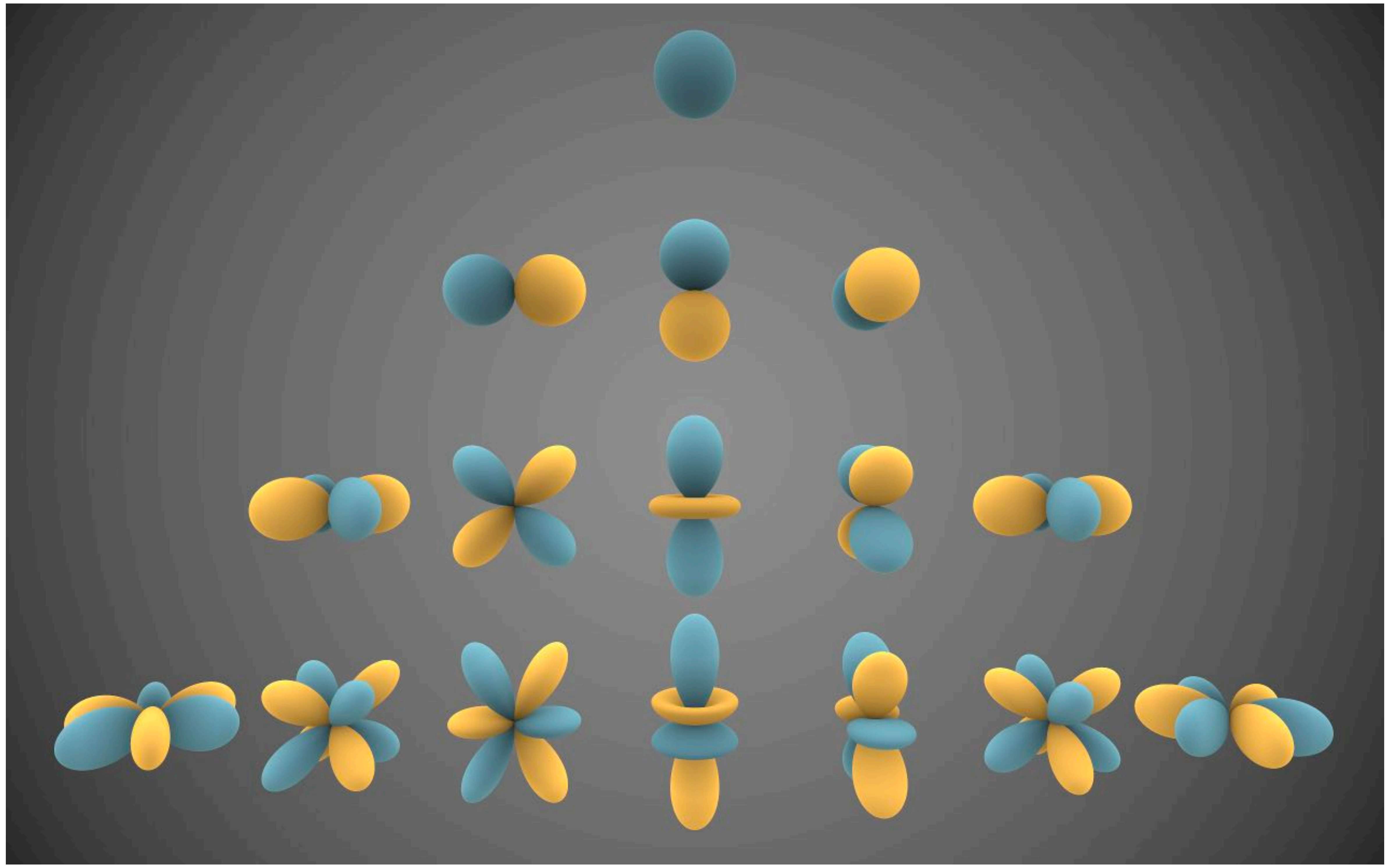


Credit: Yu 2021

- Cost? Octree structure now 100’s of MBs instead of a few MBs for MLP

Another idea: spherical harmonics

- Useful basis for representing directional information
- Analogy: cosine basis on the sphere



- Represent $c(\mathrm{p}, \omega)$ compactly by projecting into basis of SH.

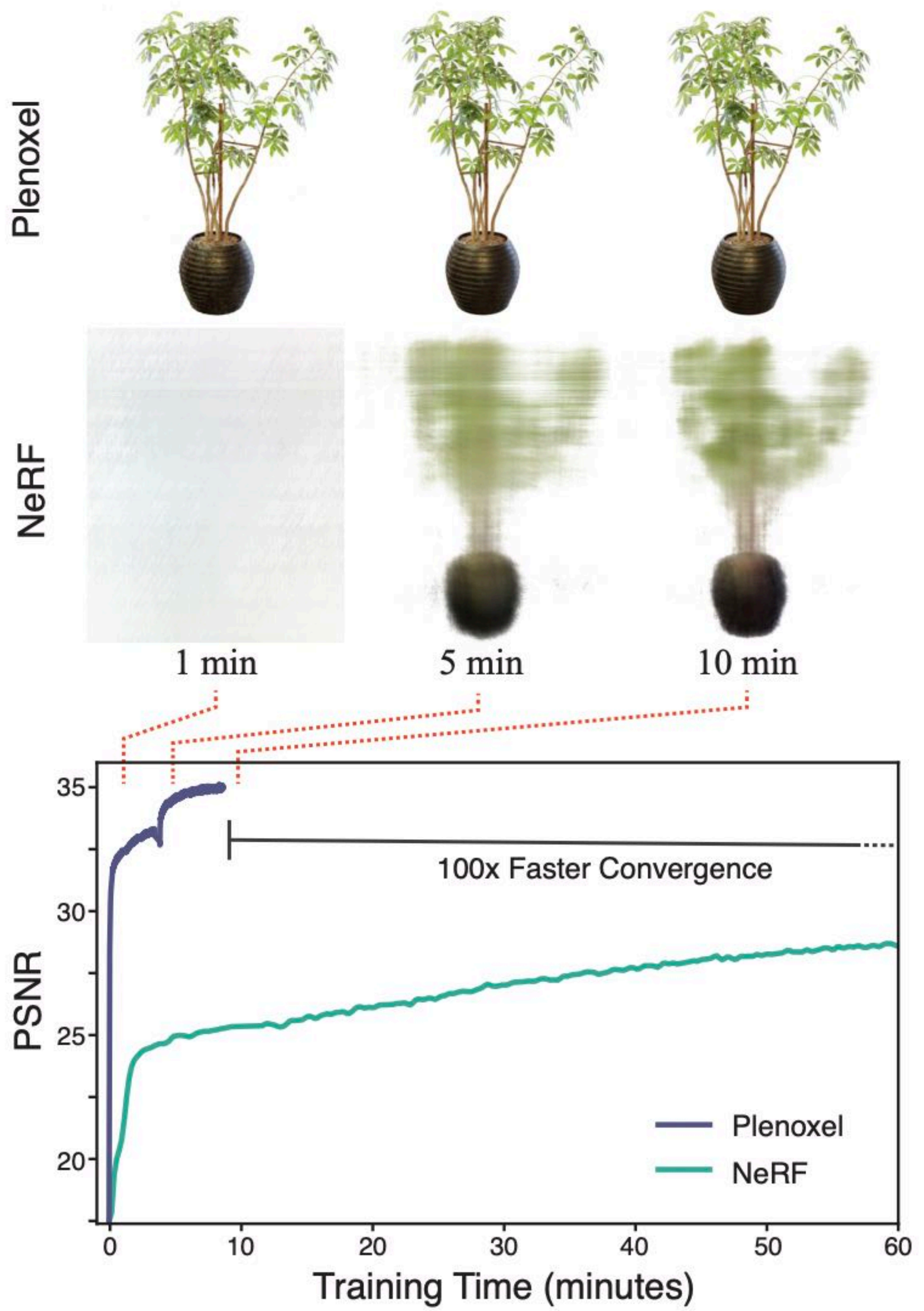
$$\mathrm{Y}_l^m(\omega)$$

Finally…back to where we began

Plenoxels [CVPR 22]

- Start with a dense 3D grid of SH coefficients, optimize those coefficients at low resolution
- Now move to a sparse higher resolution representation (octree)
- Directly optimize for opacities and SH coefficients using differentiable volume rendering
- No neural networks. Just optimizing the octree representation of baked SH lighting

- Takeaway: conventional computer graphics representations are efficient representations to learn/optimize on



Summarizing it all: the "template"

- **Train a DNN to gain understanding of 3D occupancy (where the surface is)**
  - **Little to no geometric priors (so need position encoding tricks, etc)**

$$(\mathrm{p}, \omega) \rightarrow \boxed{F_\theta(\mathrm{p}, \omega)} \rightarrow \begin{matrix} \sigma(\mathrm{p}) \\ c(\mathrm{p}, \omega) \end{matrix}$$

- **Then move to a traditional sparse encoding of occupancy (sparse volumetric structure)**
  - **Now the "topology" of the irregular data structure is fixed**
  - **Representation of surface/appearance/etc is stored at the nodes of this structure (spherical harmonics, neural code, etc.)**
  - **Most of the heavy lifting is now performed by the data structure**

$$\mathrm{p} \rightarrow \underbrace{\boxed{\mathrm{lookup}(\mathrm{p})}}_{\text{Traditional data structure}} \rightarrow \begin{matrix} \mathrm{z_p} \\ \sigma_p \\ \mathrm{c_p}(\omega) \\ \mathrm{SH}_p(\omega) \end{matrix}$$

- **Continue optimization on the fixed, sparse representation**
  - **Leverages differential volume rendering on sparse structure**
  - **What we're now learning is how to represent/compress the local details**

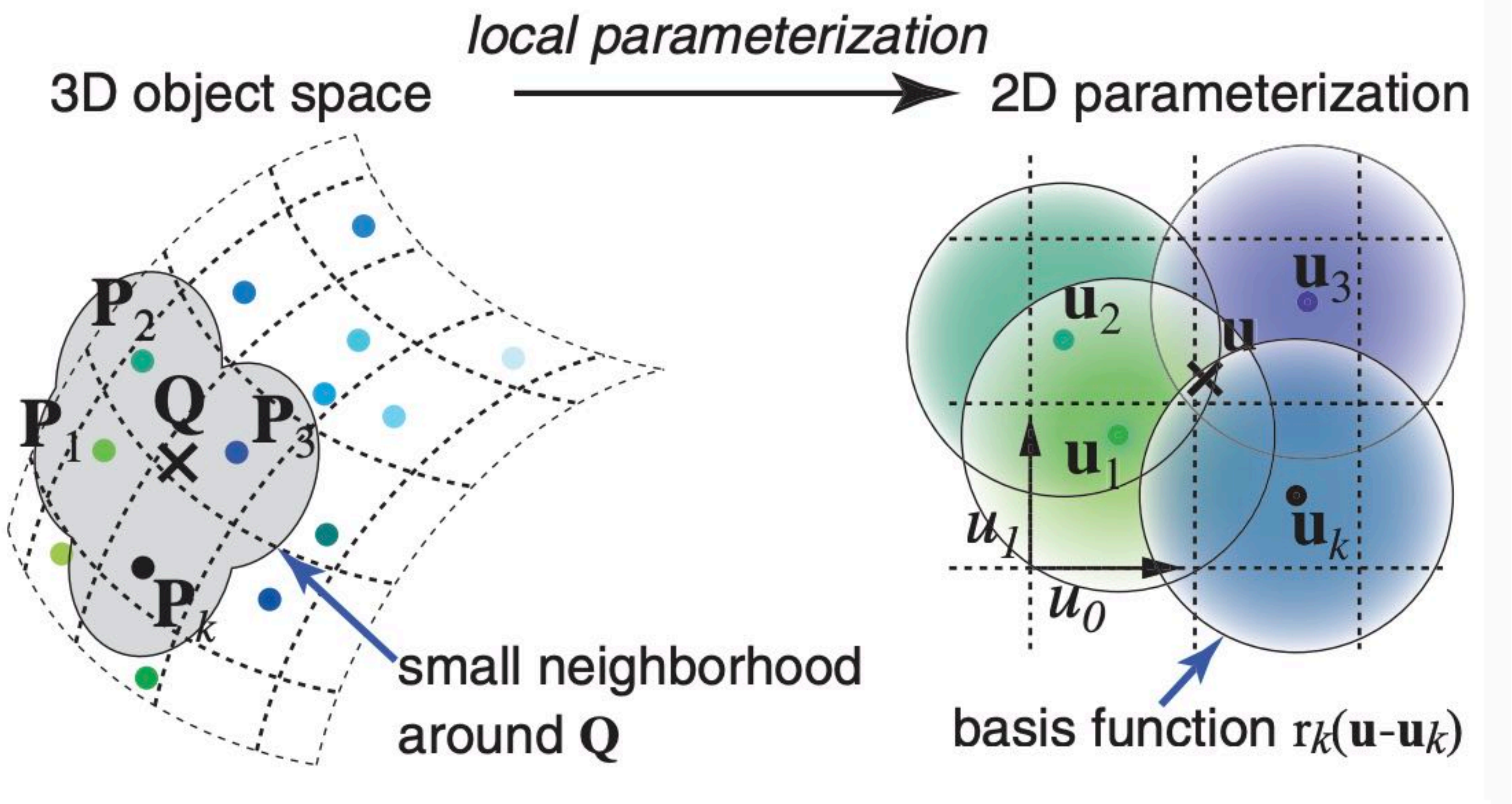
More recent innovations

- **Best practices since 2022 replace this two-step process with a single optimization passing using a data structure called a "hash grid"**
- **See "instant neural graphics primitives" (NGP)**

Rendering point clouds

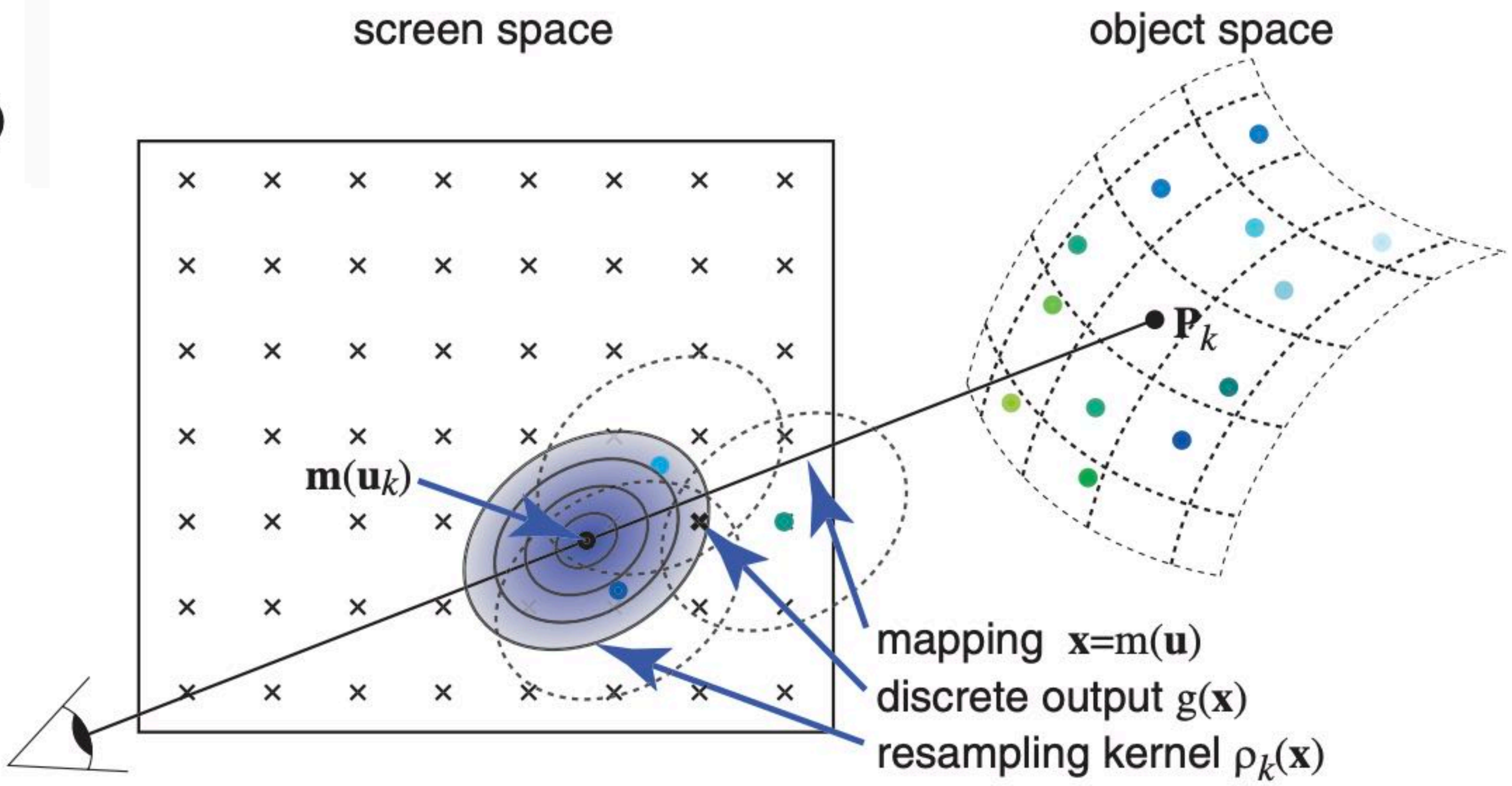
Anti-aliasing point clouds

- Treat surface as a collection of "Gaussian blobs" (convolve points with Gaussian filter)



- 3D Gaussians turn into oriented 2D gaussians when projected onto the 2D screen

- Can render the blobs back to front (requires alpha compositing)



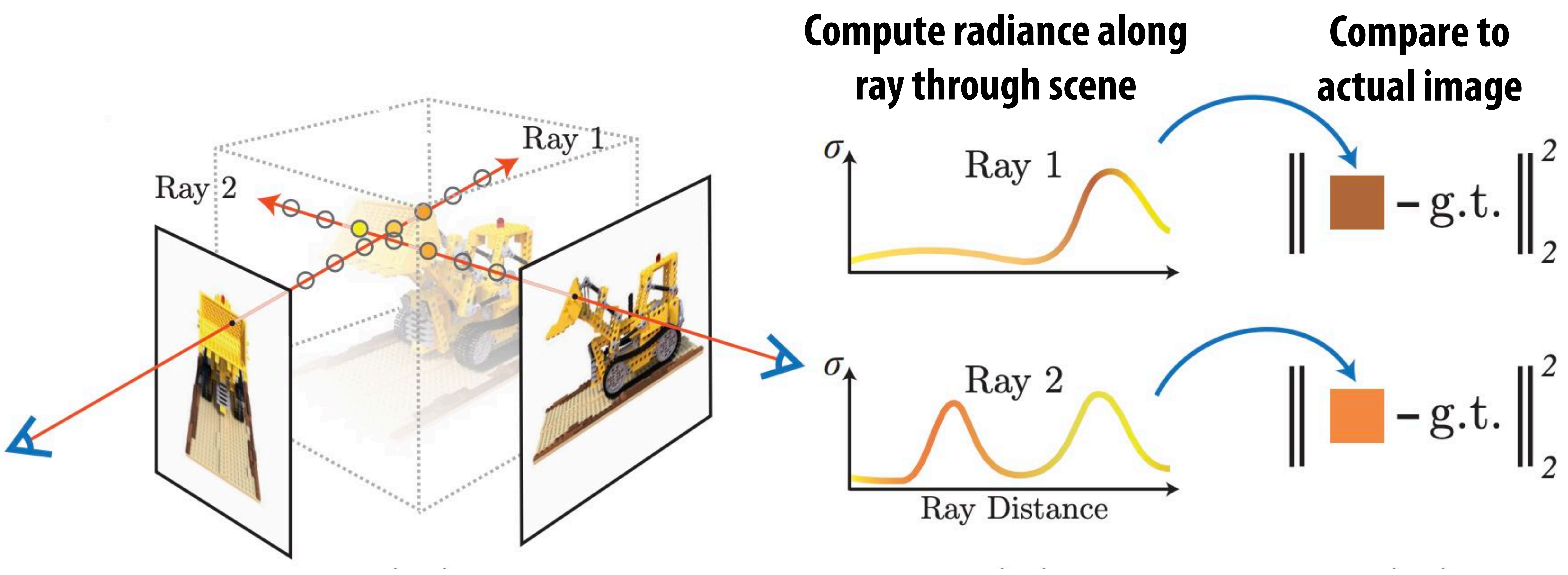
[Zwicker 2001]

"Point splatting"

[Kerbl 2023]

Optimization to produce Gaussians, not voxels

- Earlier in lecture: optimization produces color and opacity at each voxel
- Now: same idea, but optimization chooses color, position, and radius of the Gaussians
  - Now: also need to decide on the number of Gaussians (a bit tricker)



Key idea: differentiable Gaussian splatting rendering to compute dC/d(color)d(radius)d(location)

See "3D Gaussian Splatting for Real-Time Radiance Field Rendering" [Kerbl 2023]

Summary

- Volumes (voxels) and Gaussian points as two alternative representations of geometry and materials

- Significant interest in modern times due to ease of writing differentiable renderers for these representations

- Heavy use in reconstructing scenes from (potentially sparse) set of photos
  - Surprising effectiveness of large-scale optimization

- Some of these solutions employ interesting combinations of neural structures (learned DNN weights) and “traditional” graphics primitives
  - Takeaway for graphics students in 2024: need to be a master of both!

Summary

- Thanks to Matt Pharr, Pat Hanrahan for materials in these slides