Lecture 15:

Rendering Volumes, Points, and Gaussians

Computer Graphics: Rendering, Geometry, and Image Manipulation
Stanford CS248A, Winter 2025

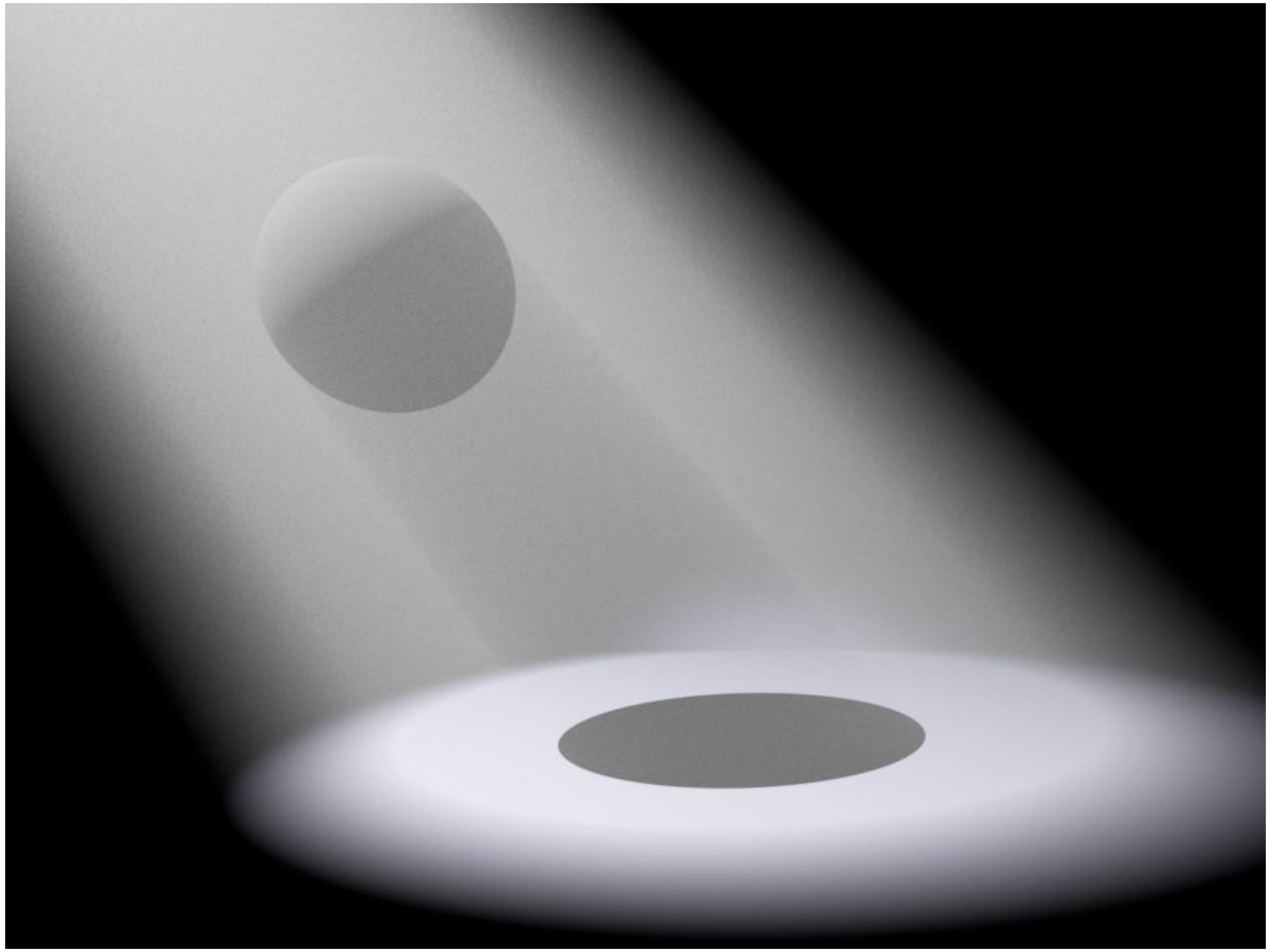
So far in this course

- We have discussed ray tracing algorithms for rendering surfaces in a vacuum
  - Rays traveled through empty space until the next surface
  - Radiance was conserved along a ray

- Let's change that today…

But what about scenes like this…

Volumetric effects

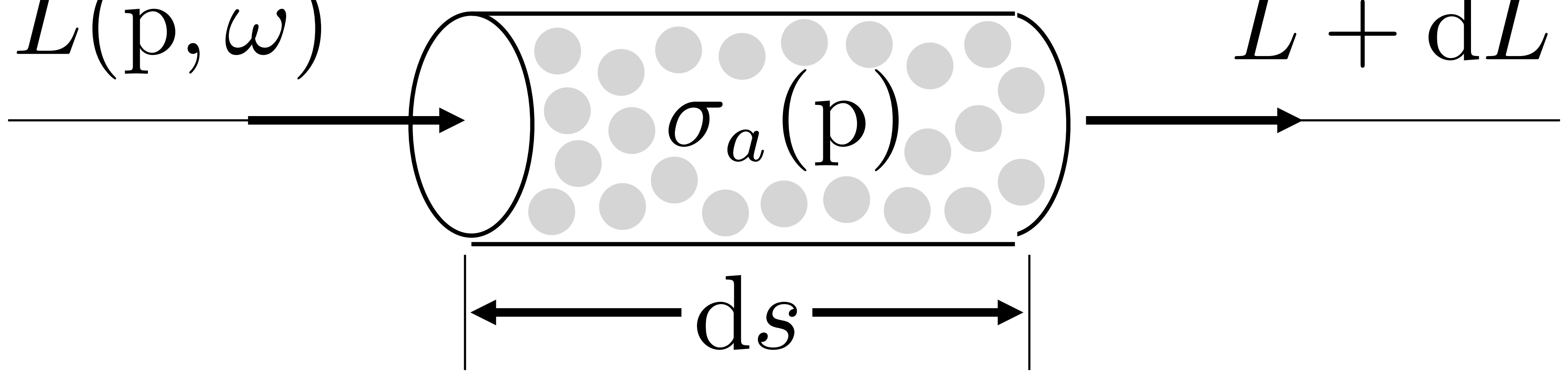


Today's subject

- Rendering from geometry representations that are not meshes
    - Rendering volumes
    - Rendering points/gaussians

- And their implications to modern progress in scene capture

Absorption in a volume



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

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

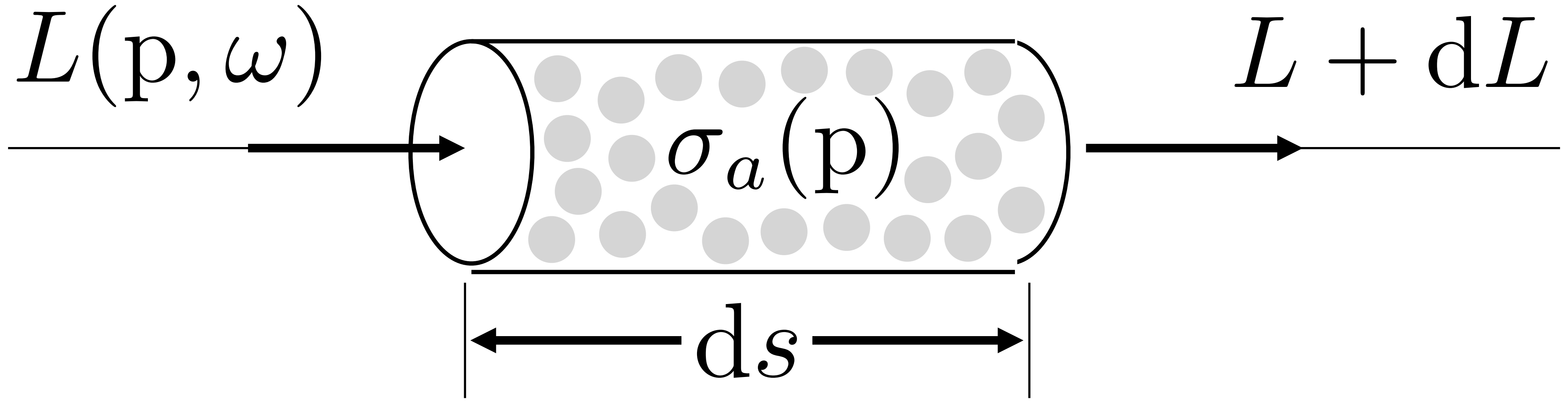
$$\mathrm{d}L(\mathrm{p}, \omega) = -\sigma_a(\mathrm{p})\, L(\mathrm{p}, \omega)\, \mathrm{d}s$$

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

- $L(\mathrm{p}, \omega)$ **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

Transmittance:



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

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

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

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

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

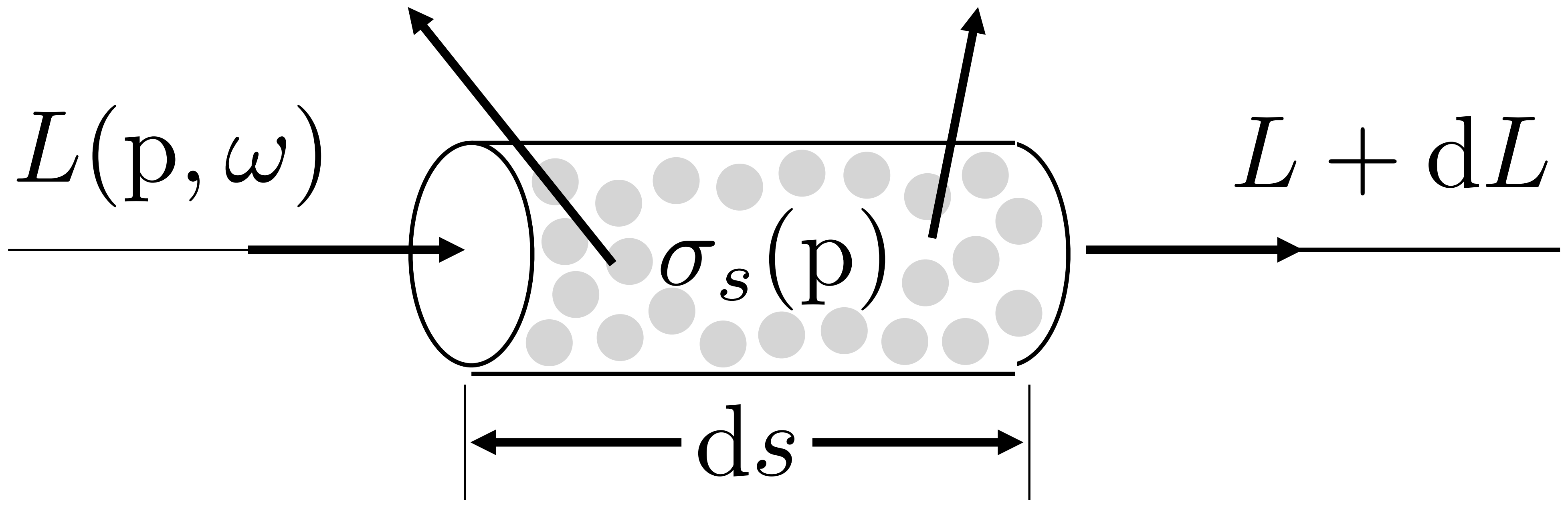
Absorption: lower density

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:

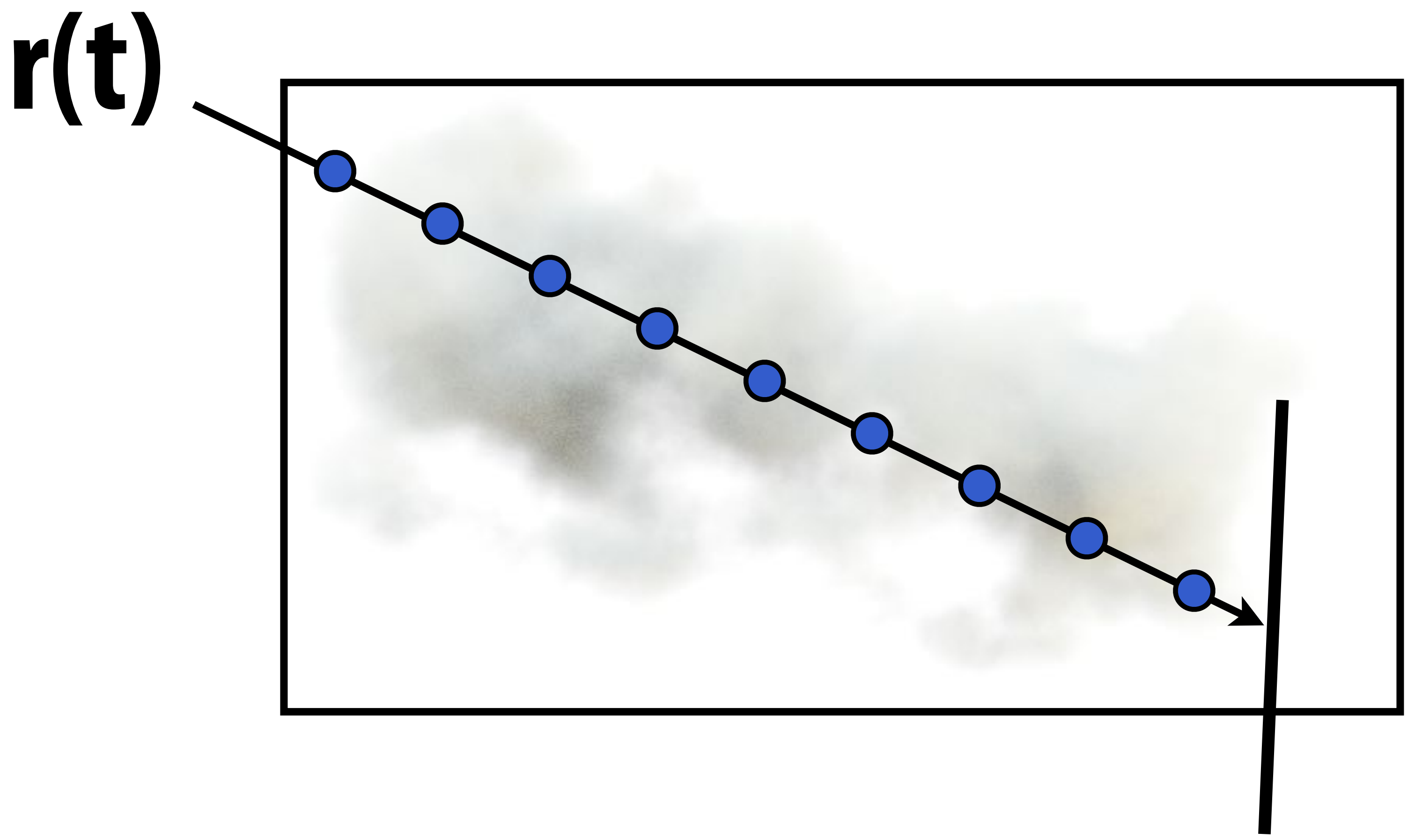
$$T(s) = \mathrm{e}^{-\int_0^s \sigma_\mathrm{t}(\mathrm{p}+s'\omega)\,\mathrm{d}s'} = e^{-\tau(s)}$$

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

"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:

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

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

Trying to approximate this integral

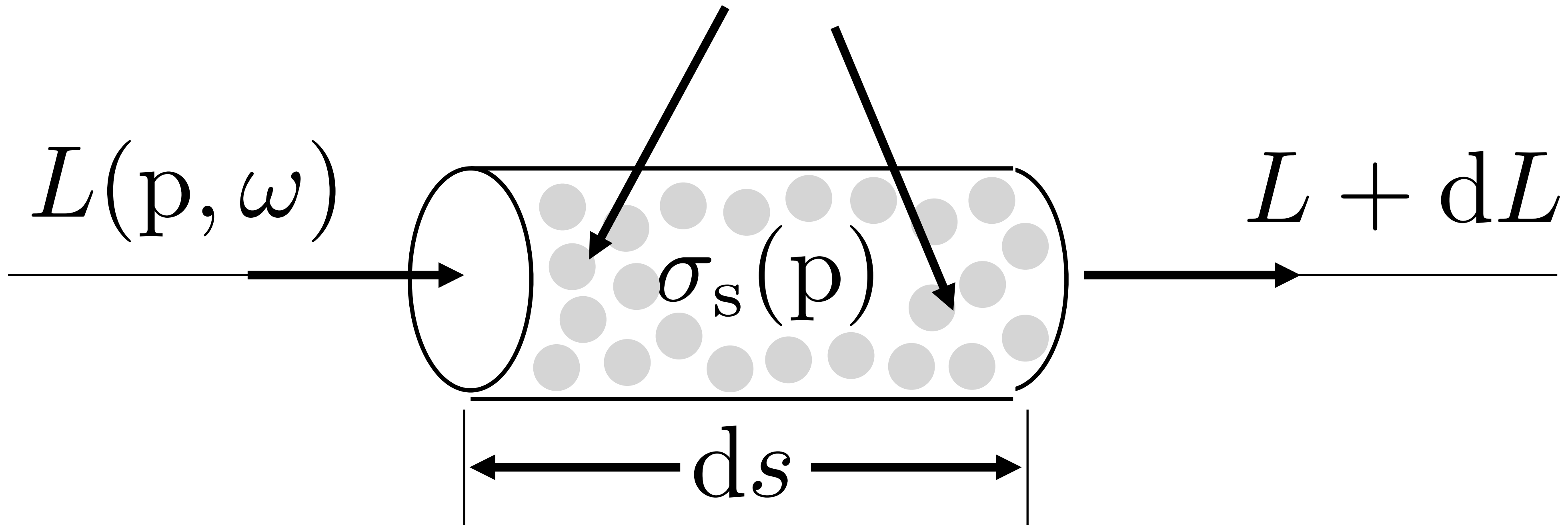
$$\tau(s) = \int_0^s \sigma_t(\mathrm{p} + s'\omega)\,\mathrm{d}s'$$

To compute:

$$T(s) = \mathrm{e}^{-\tau(s)}$$

In scattering

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



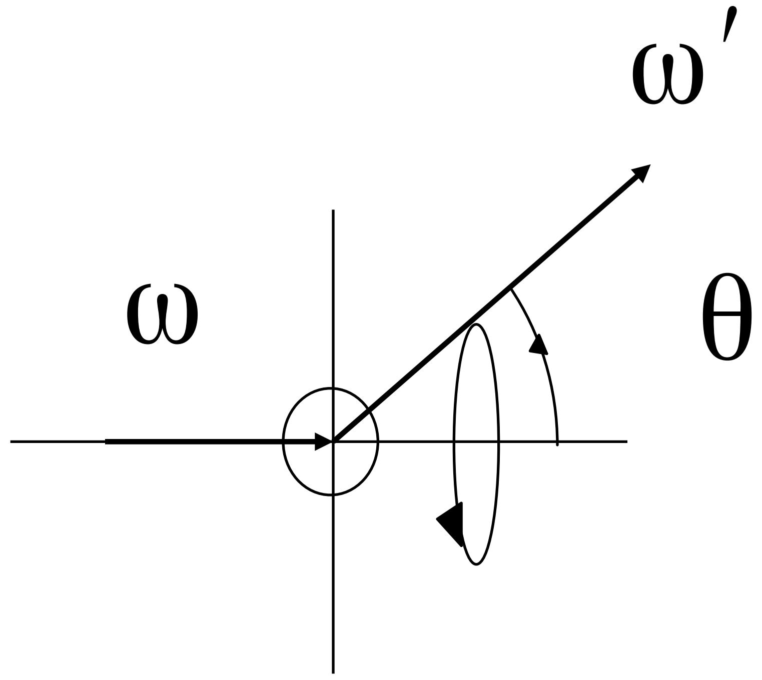
$$S(\mathrm{p}, \omega) = \sigma_s(\mathrm{p}) \int_{S^2} p(\omega' \to \omega)\, L(\mathrm{p}, \omega')\, \mathrm{d}\omega'$$

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

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

Phase Functions

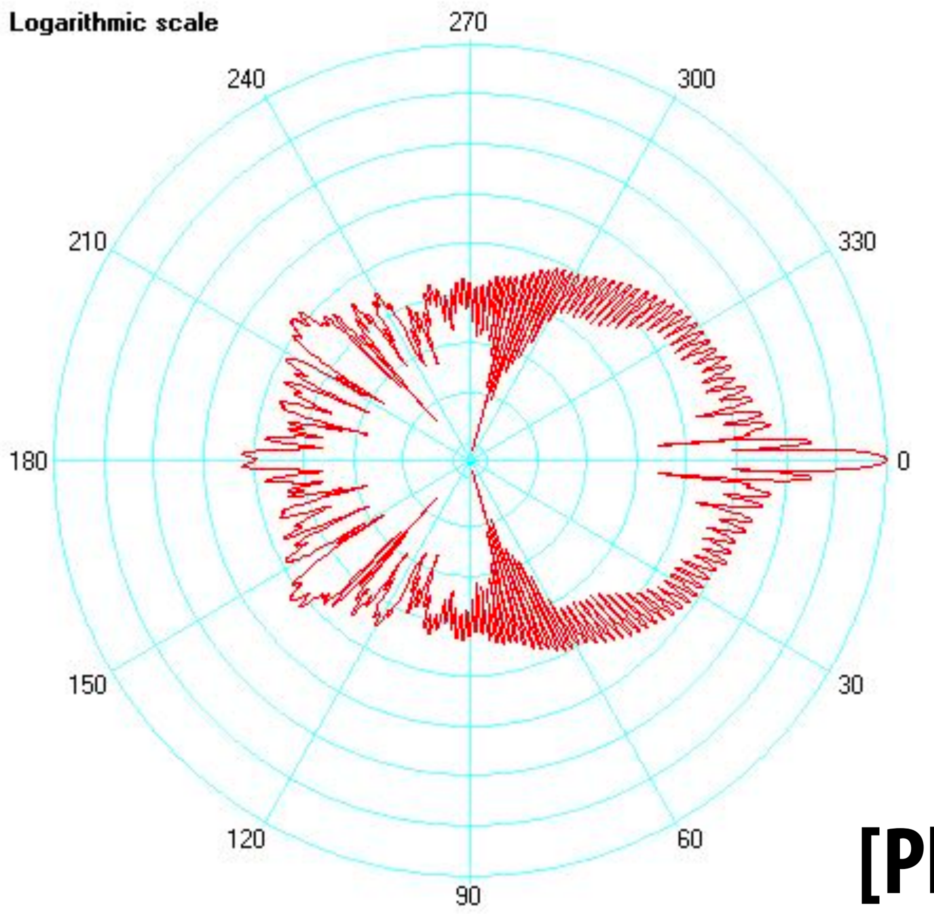
- **Phase angle** $\cos\theta = \omega \cdot \omega'$



- **Phase functions**
  - **Isotropic:** $p(\cos\theta) = \frac{1}{4\pi}$
  - **Rayleigh:** $p(\cos\theta) = \frac{3}{4}(1 + \cos^2\theta)$ **with** $\sigma_s \propto \frac{1}{\lambda^4}$

Note: probability of scattering is function of wavelength

  - **Mie:**



[Philip Laven]

Rayleigh Scattering: Blue Sky, Red Sunset

From Greenler: Rainbows, Halos, and Glories

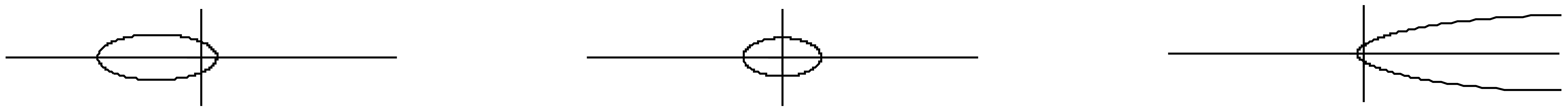
Henyey-Greenstein Phase Function

- **Empirical phase function**

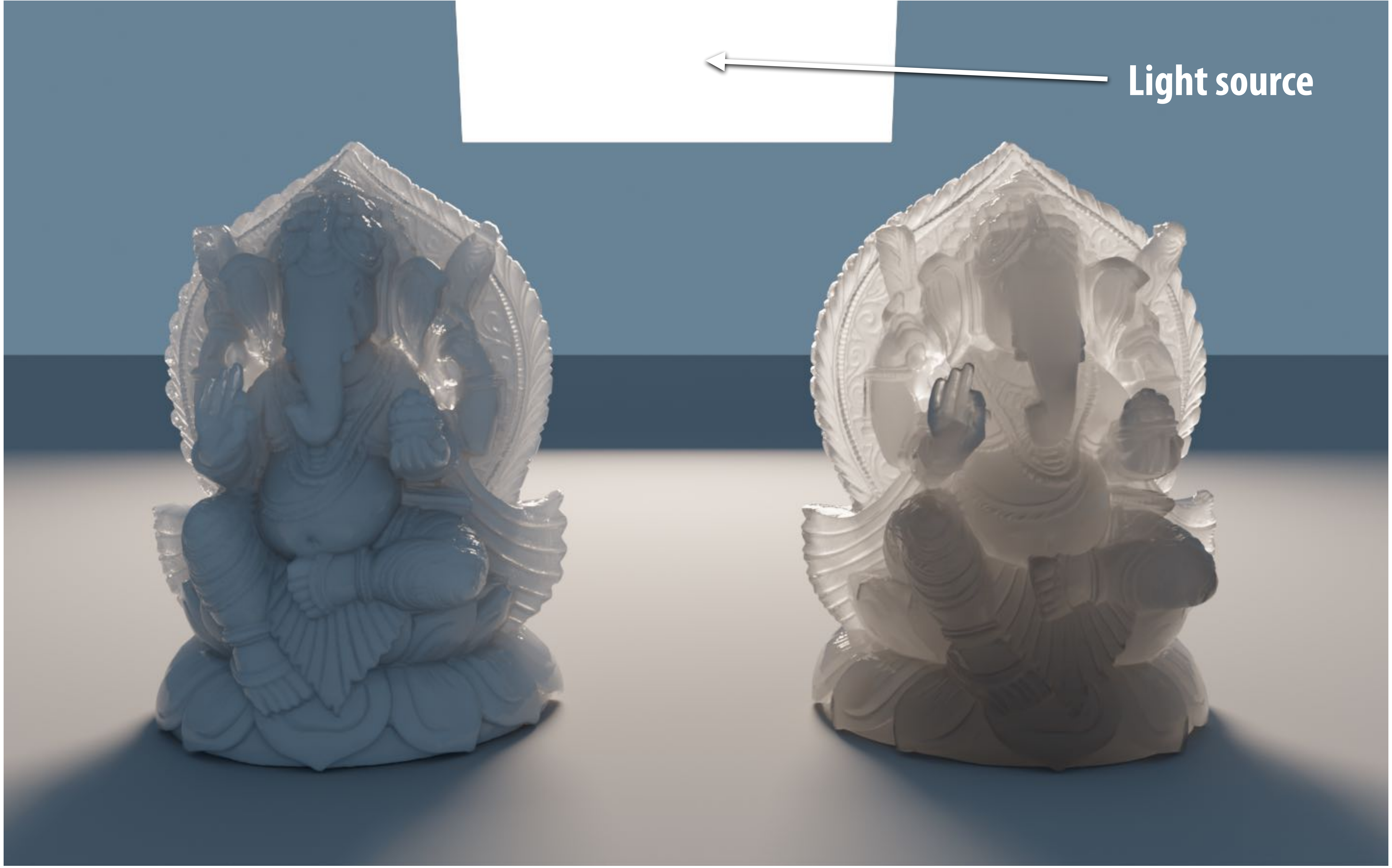
$$p(\cos\theta) = \frac{1}{4\pi}\frac{1-g^2}{(1+g^2-2g\cos\theta)^{3/2}}$$

- **Average phase angle g:**

$$g = 2\pi\int_0^{2\pi} p(\cos\theta)\ \cos\theta\sin\theta\,\mathrm{d}\theta$$



$g=-0.3$ $g=0$ $g=0.6$



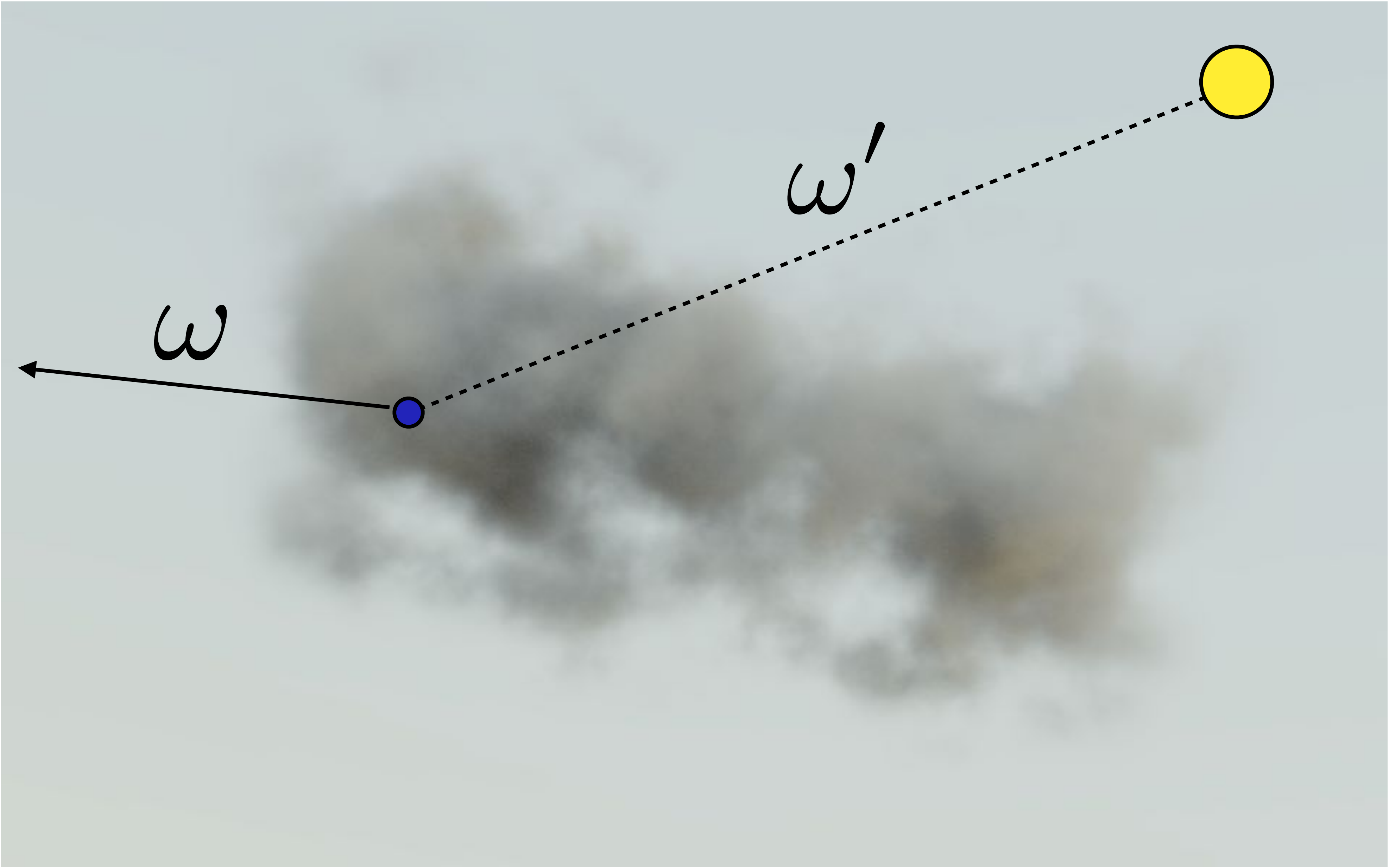
More Backward Scattering

More Forward Scattering

Direct illumination in a volume

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

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

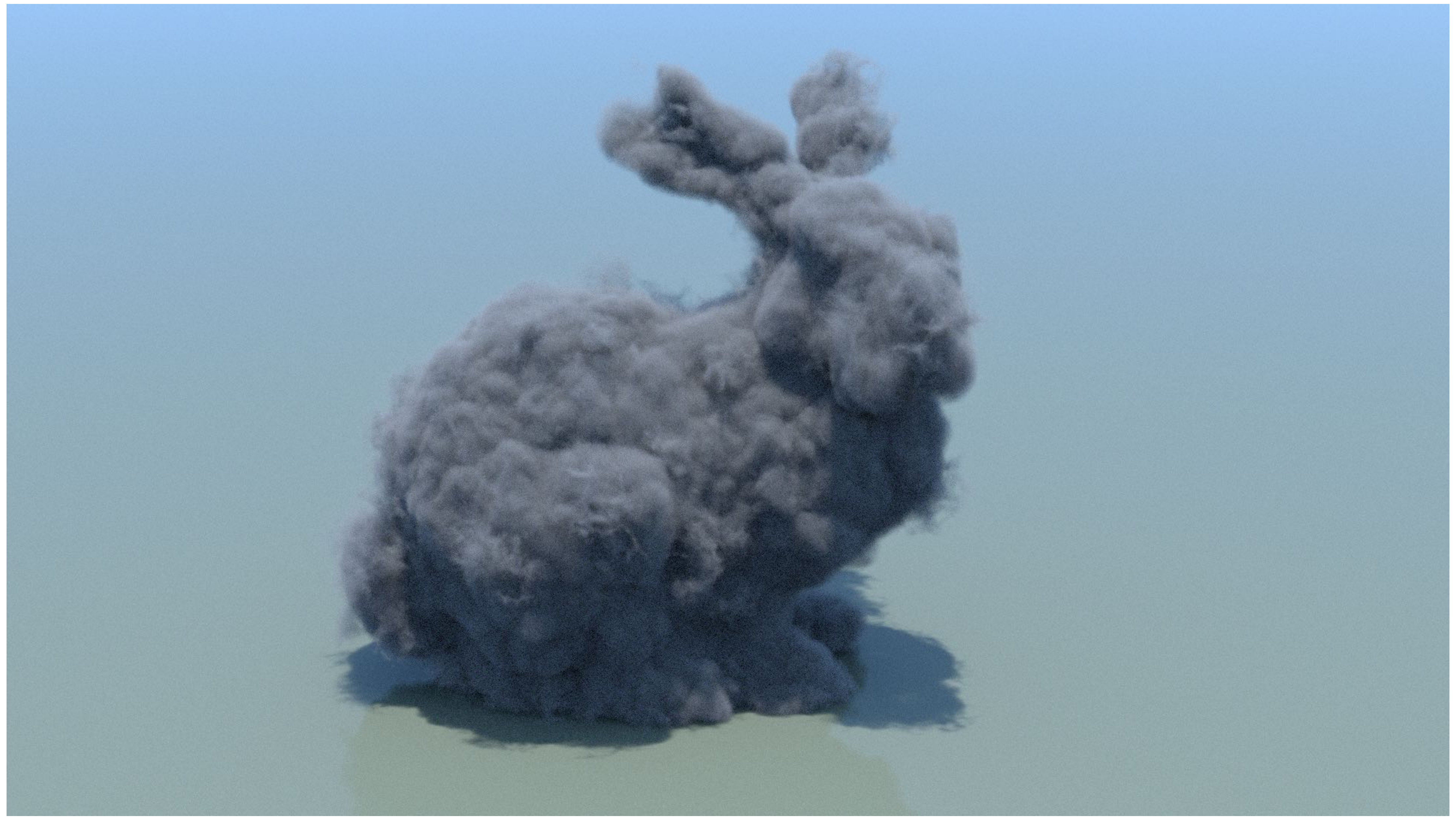


- **But computing direct lighting is now much more expensive**
- **Why? (Hint: requires more than a shadow ray)**

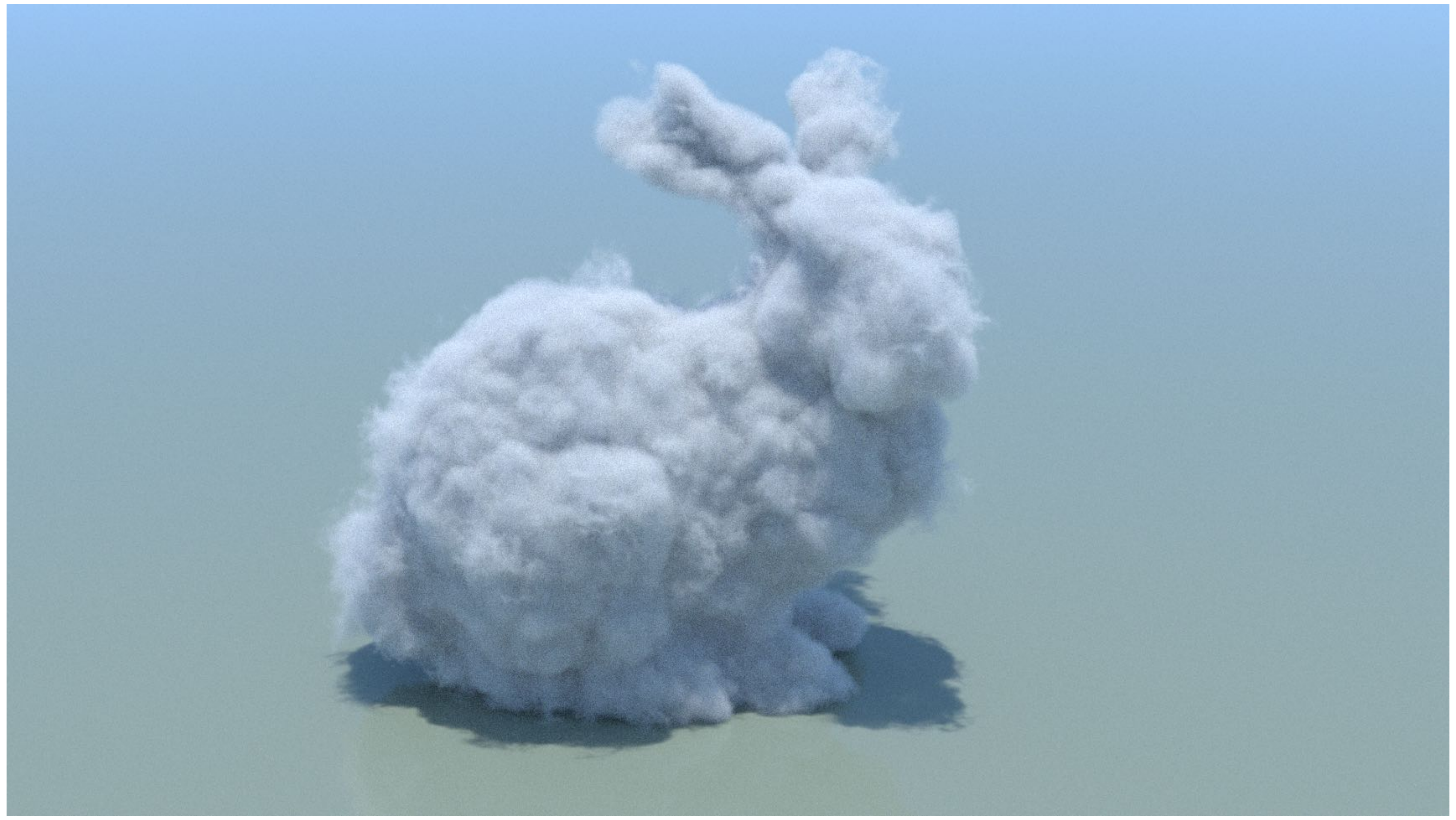
Single scattering

Multiple scattering (not discussed today)

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



Single scattering



Multiple scattering

Let's ignore lighting for a moment

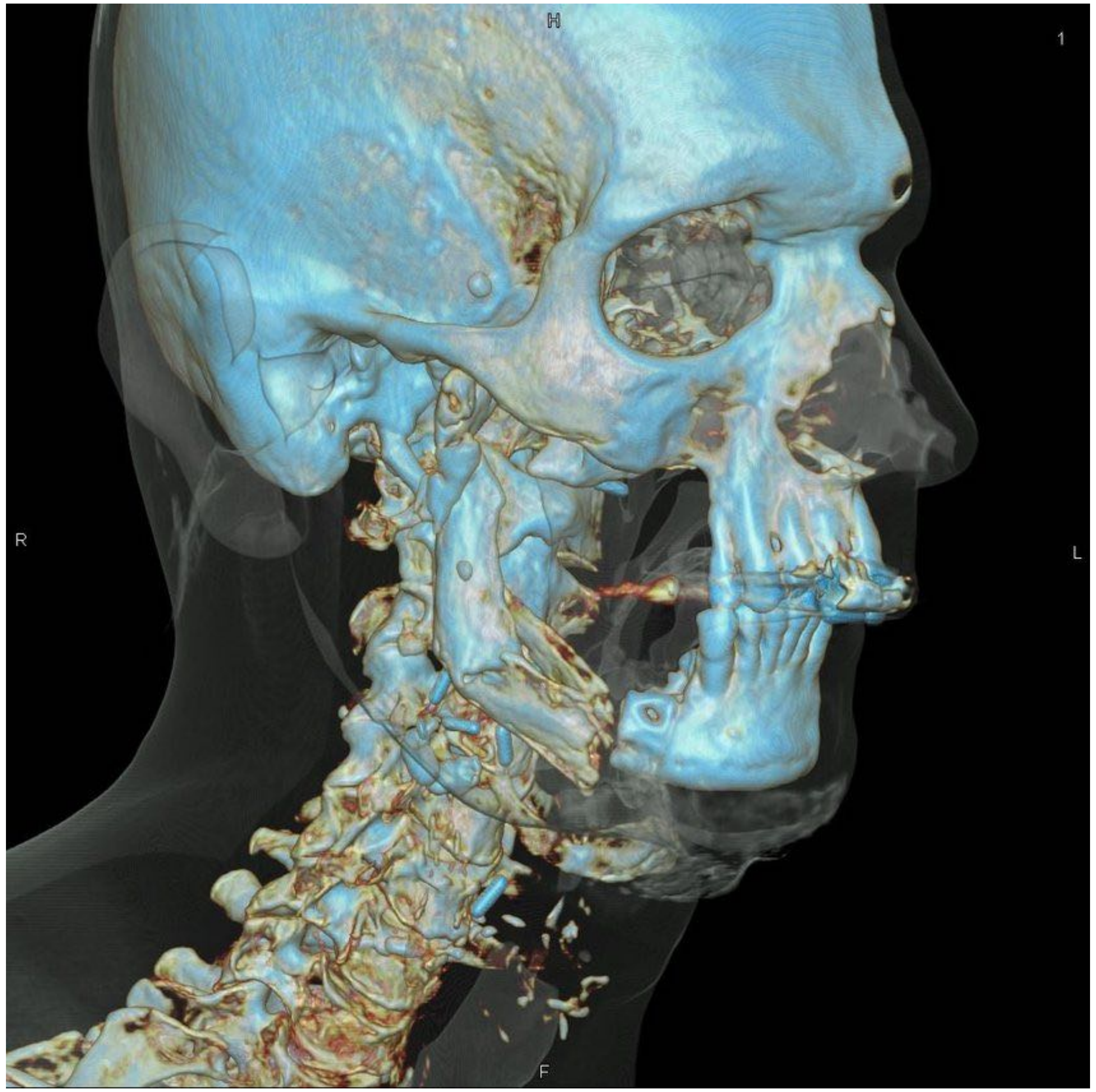
Consider representing a scene as a volume

Volume 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 CT scan

Volume rendered scene (Mildenhall et al.)

Credit: Taubmann et al. , Siemens Healthineers

Rendering volumes

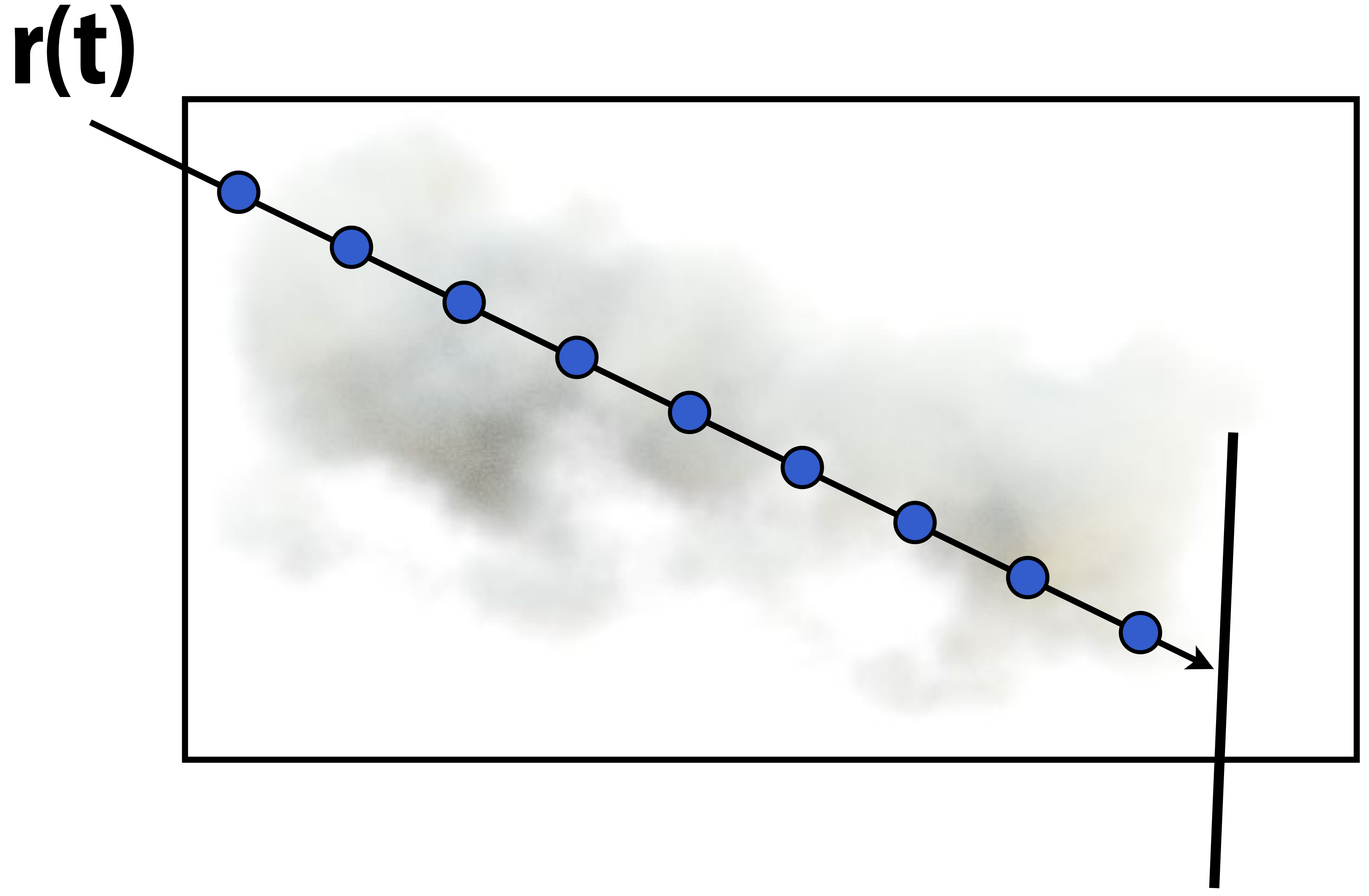
Given "camera ray" from point o in direction w….

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

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.

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

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)

Regular 3D grid representation of a volume

Consider storage requirements:
4096^3 cells

Ignore directional dependency: rgbσ 4 bytes/cell
~ 128 GB

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

Typical challenge:
limited resolution

Credit: Voxel Ville NFT (voxelville.io)

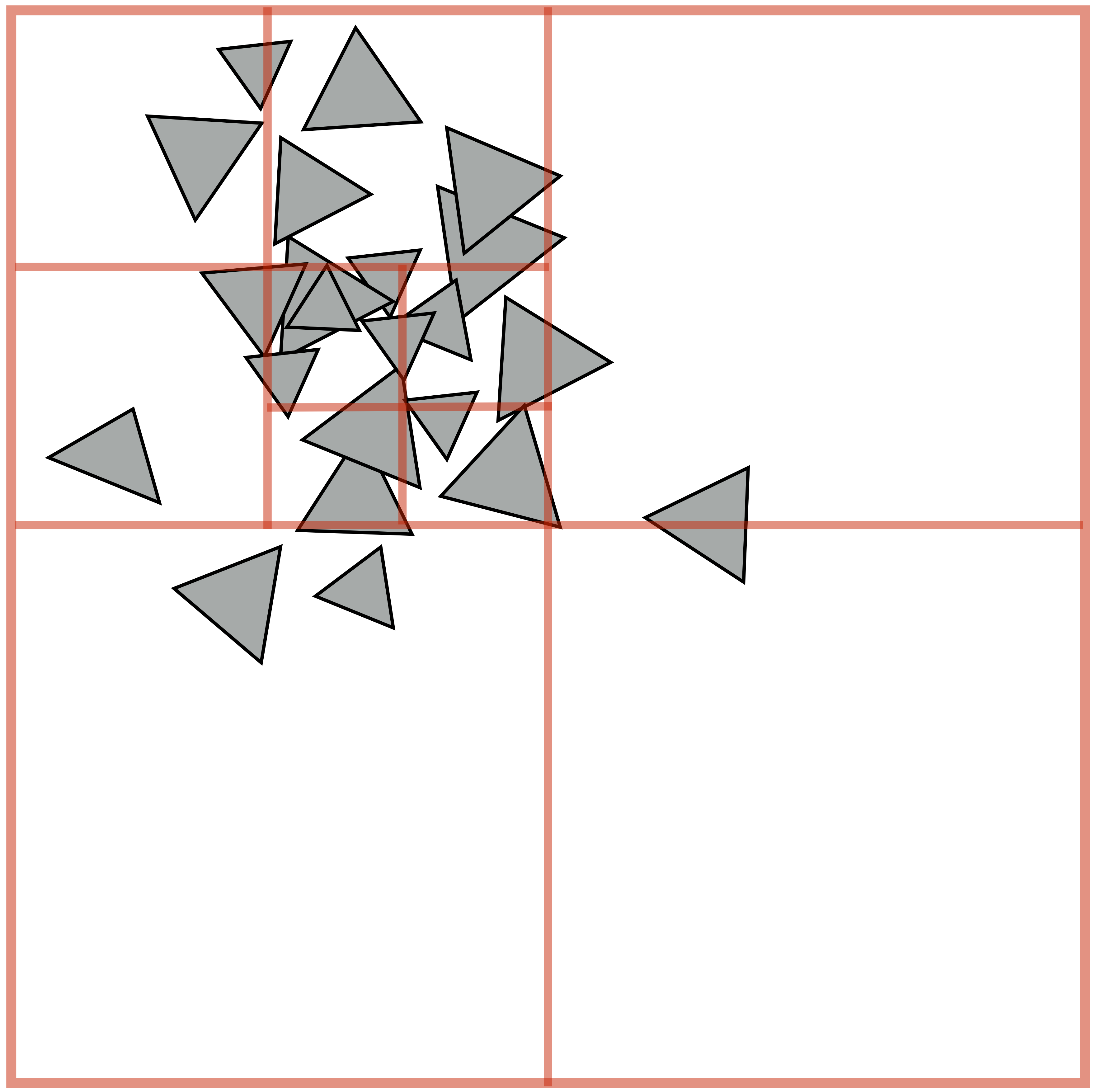
Recall quad-tree / octree

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

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

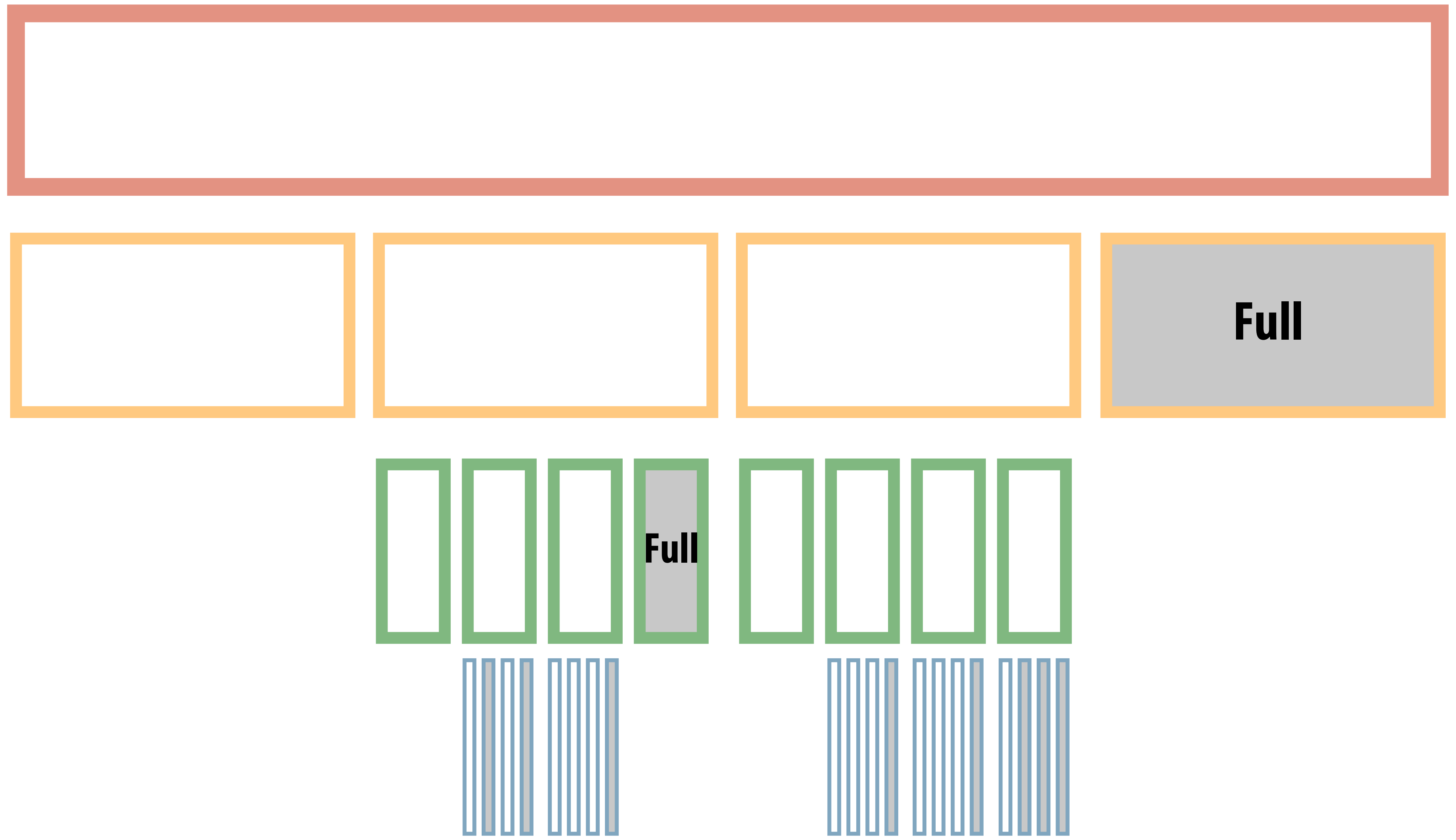
Like uniform grid: easy to build (don't have to choose partition planes)

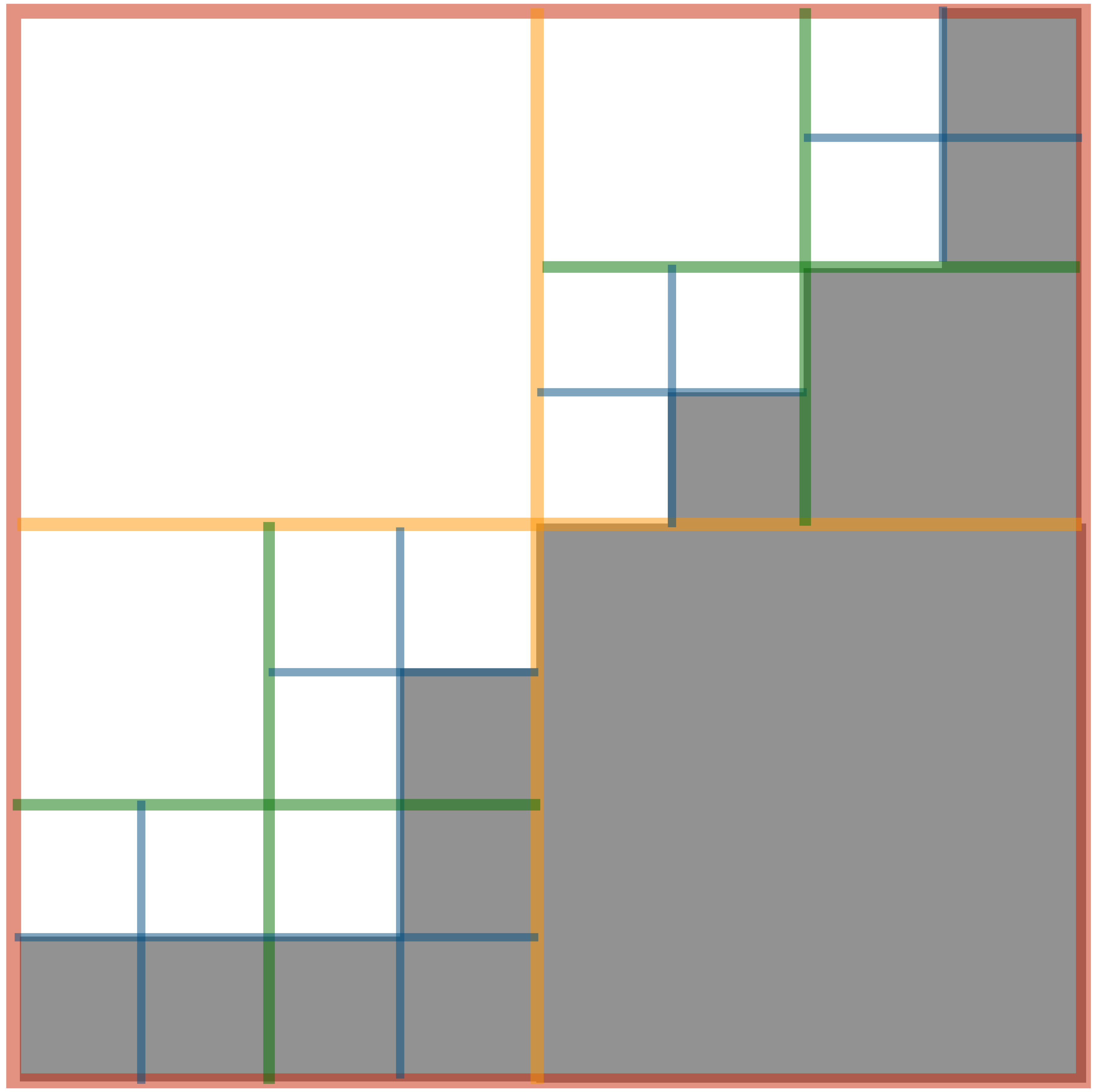
Has greater ability to adapt to location of scene geometry than uniform grid.



Recall quad-tree / octree

Now store samples of occupancy or density field in the tree structure, not triangles





Effective resolution in this example is 8x8: but structure only must store 20 leaf nodes

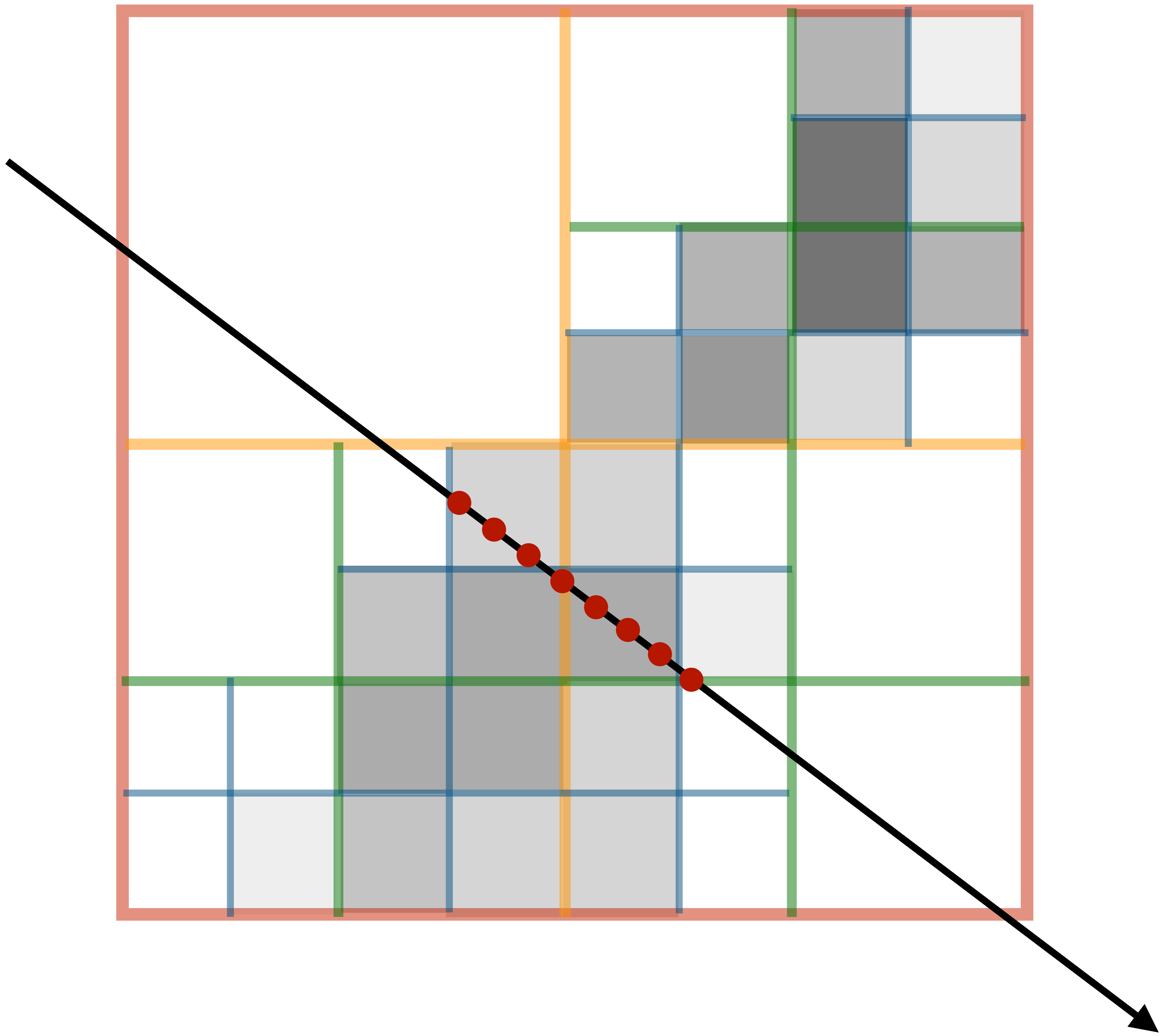
Interior nodes with no children → same "value" for all children in subtree

Value stored at nodes could be: binary occupancy, or value like: $\sigma_a(x, y, z)$ or $\sigma_s(x, y, z)$

Ray marching a sparse voxel grid

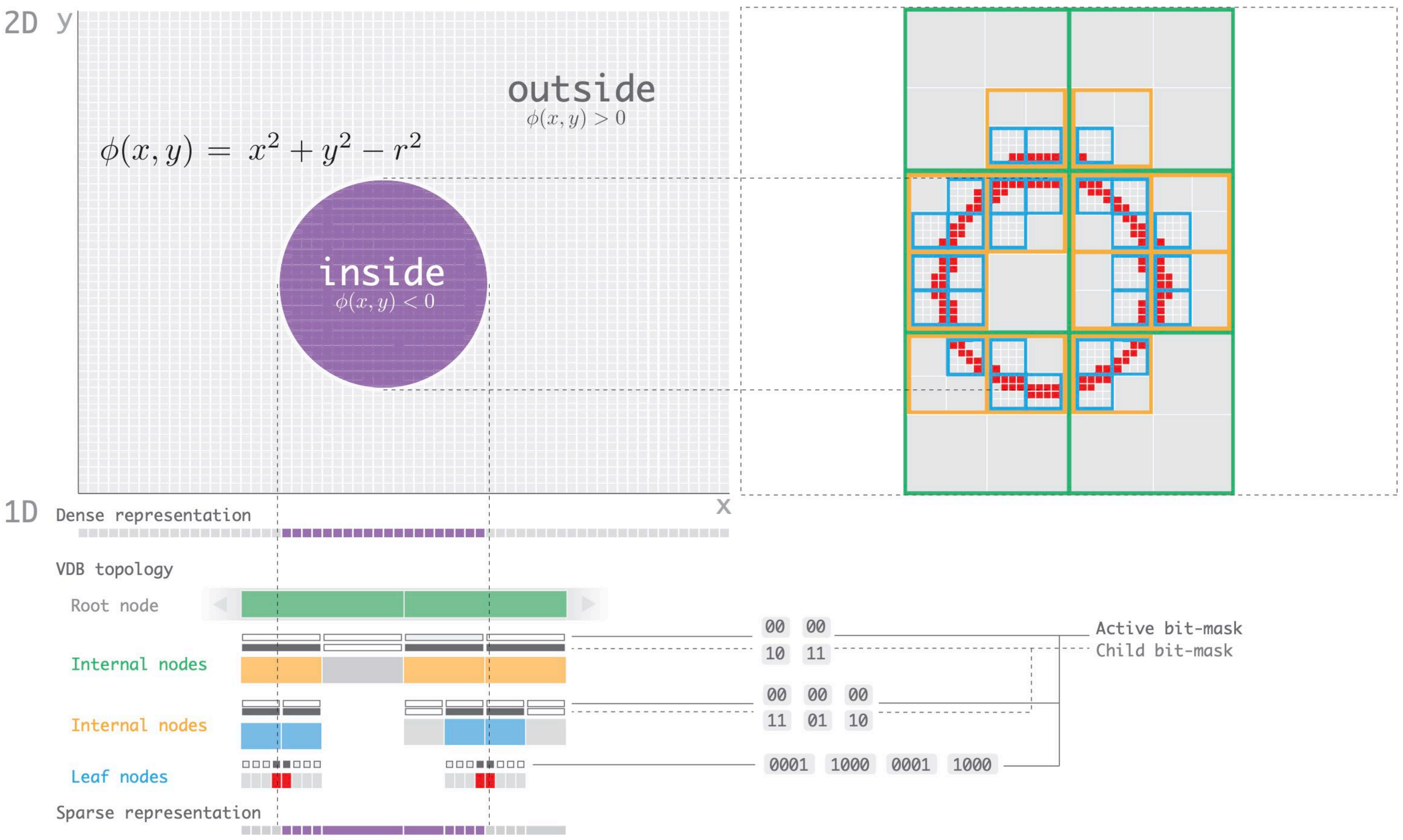
Ray can now "skip" through empty space

Ray marching is much more efficient when it's easy to determine where the "empty space" is



OpenVDB

- Popular tree-structure for representing sparse volumetric data
- Inspired by B+ trees used in databases



OpenVDB node visualization

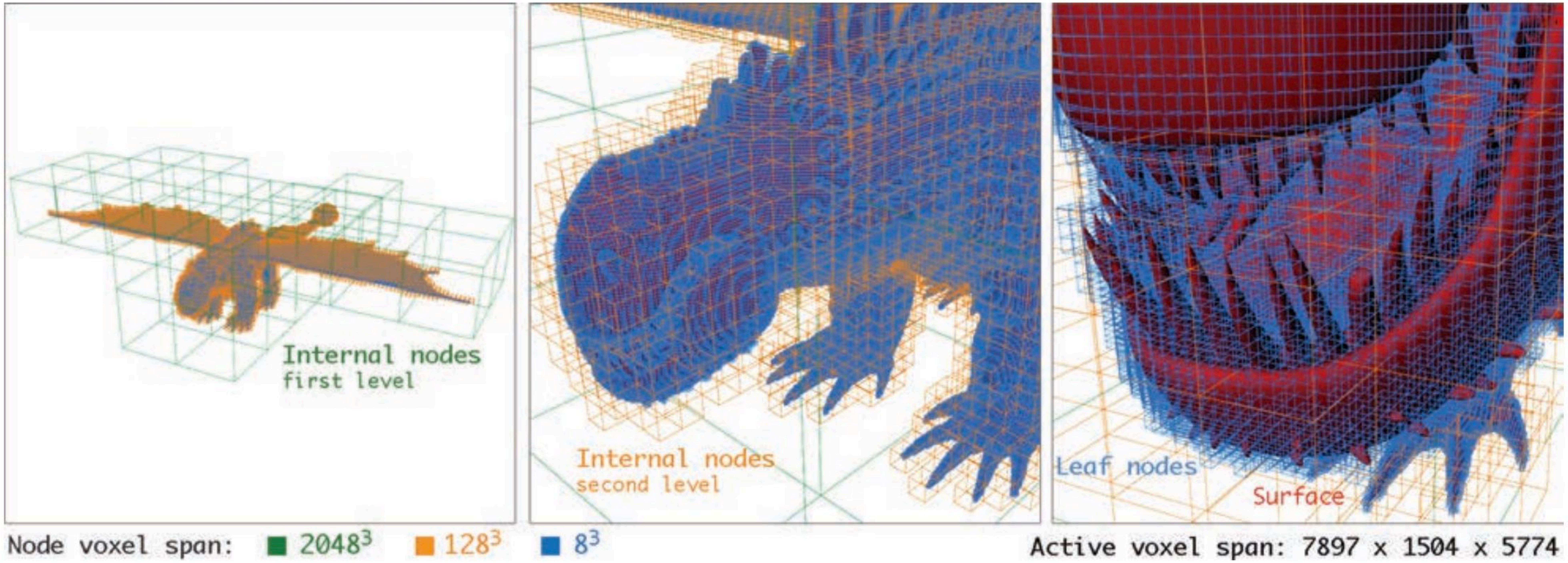


Fig. 4. High-resolution VDB created by converting polygonal model from *How To Train Your Dragon* to a narrow-band level set. The bounding resolution of the 228 million active voxels is $7897 \times 1504 \times 5774$ and the memory footprint of the VDB is 1GB, versus the $\frac{1}{4}$ TB for a corresponding dense volume. This VDB is configured with `LeafNodes` (blue) of size 8^3 and two levels of `InternalNodes` (green/orange) of size 16^3. The index extents of the various nodes are shown as colored wireframes, and a polygonal mesh representation of the zero level set is shaded red. Images are courtesy of *DreamWorks Animation*.

Example usage of volumetric data

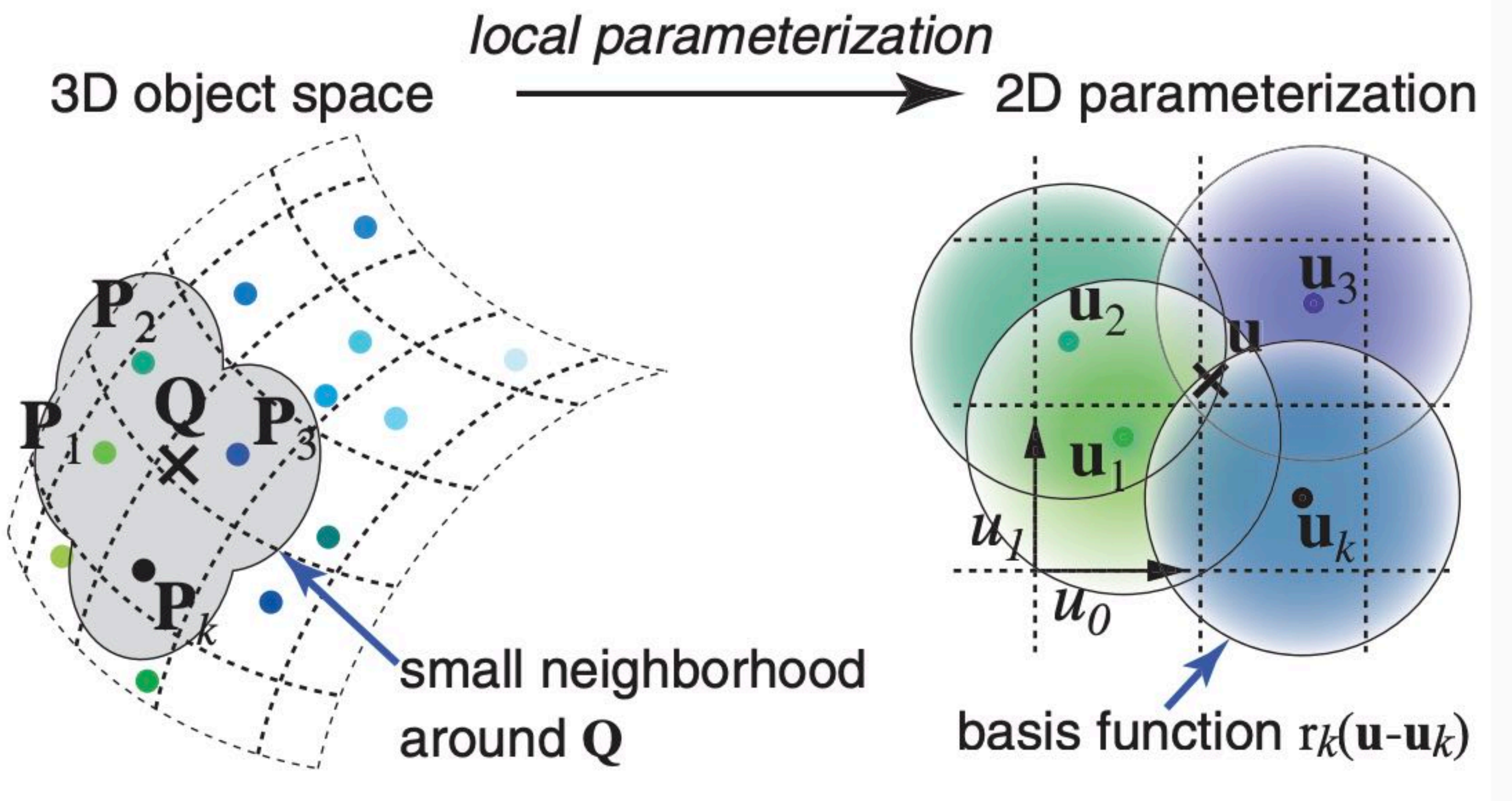
Fig. 1. Top: Shot from the animated feature *Puss in Boots*, showing high-resolution animated clouds generated using VDB [Miller et al. 2012]. Left: The clouds are initially modelled as polygonal surfaces, then scan-converted into narrow-band level sets, after which procedural noise is applied to create the puffy volumetric look. Right: The final animated sparse volumes typically have bounding voxel resolutions of $15,000 \times 900 \times 500$ and are rendered using a proprietary renderer that exploits VDB's hierarchical tree structure. Images are courtesy of *DreamWorks Animation*.

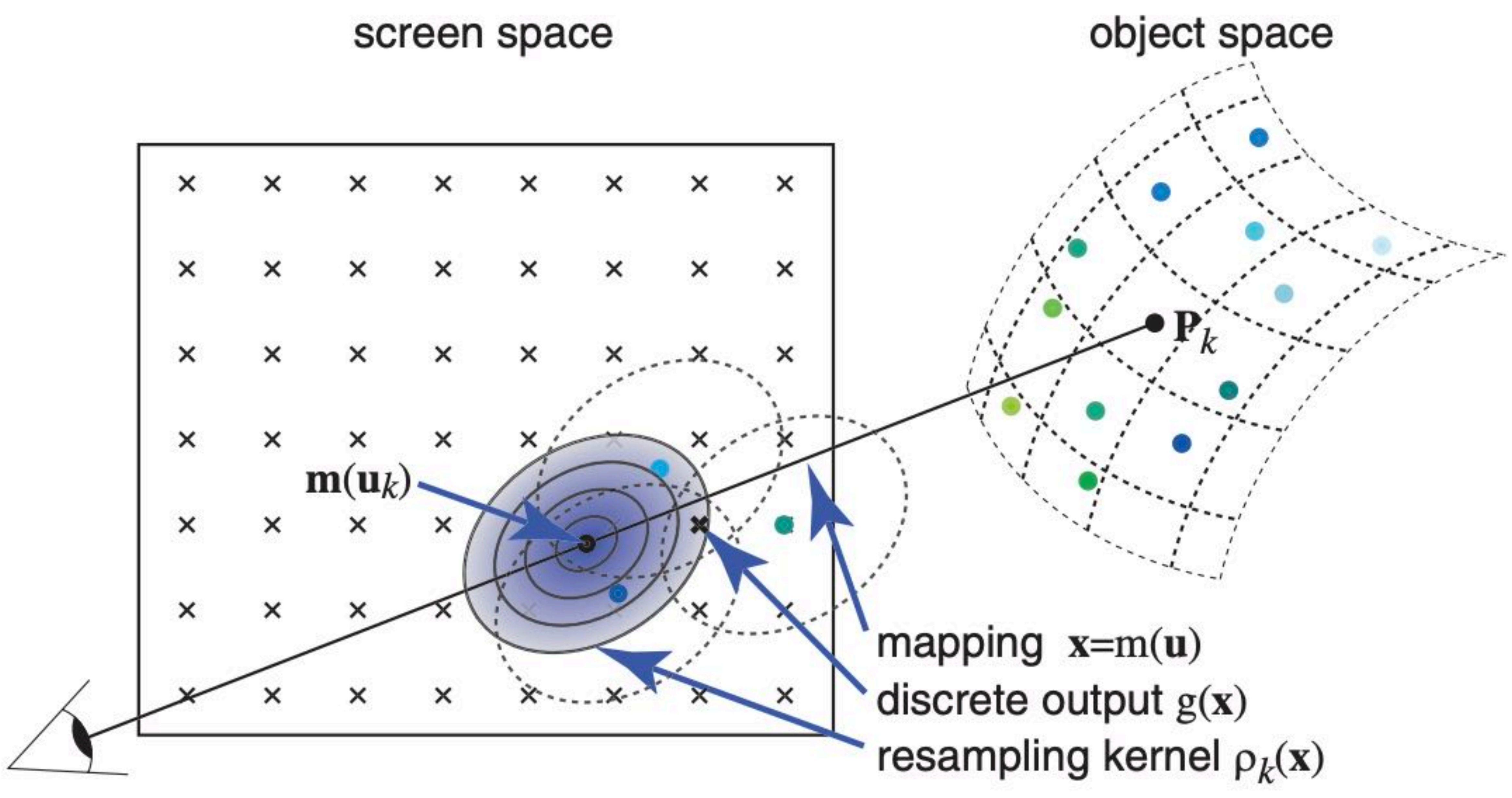
Rendering Points (and Gaussians)

Rendering point clouds

Anti-aliasing point clouds

- **Treat surface as a collection of "3D Gaussian blobs" (convolve points with Gaussian filter)**





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

- **Can render the blobs by rasterizing them back to front (this requires alpha compositing)**

[Zwicker 2001]

Visualization of 3D Gaussians

Rendered Result

Visualization of 3D Gaussians

[Credit: Kerbl 2023]

"Gaussing splatting"

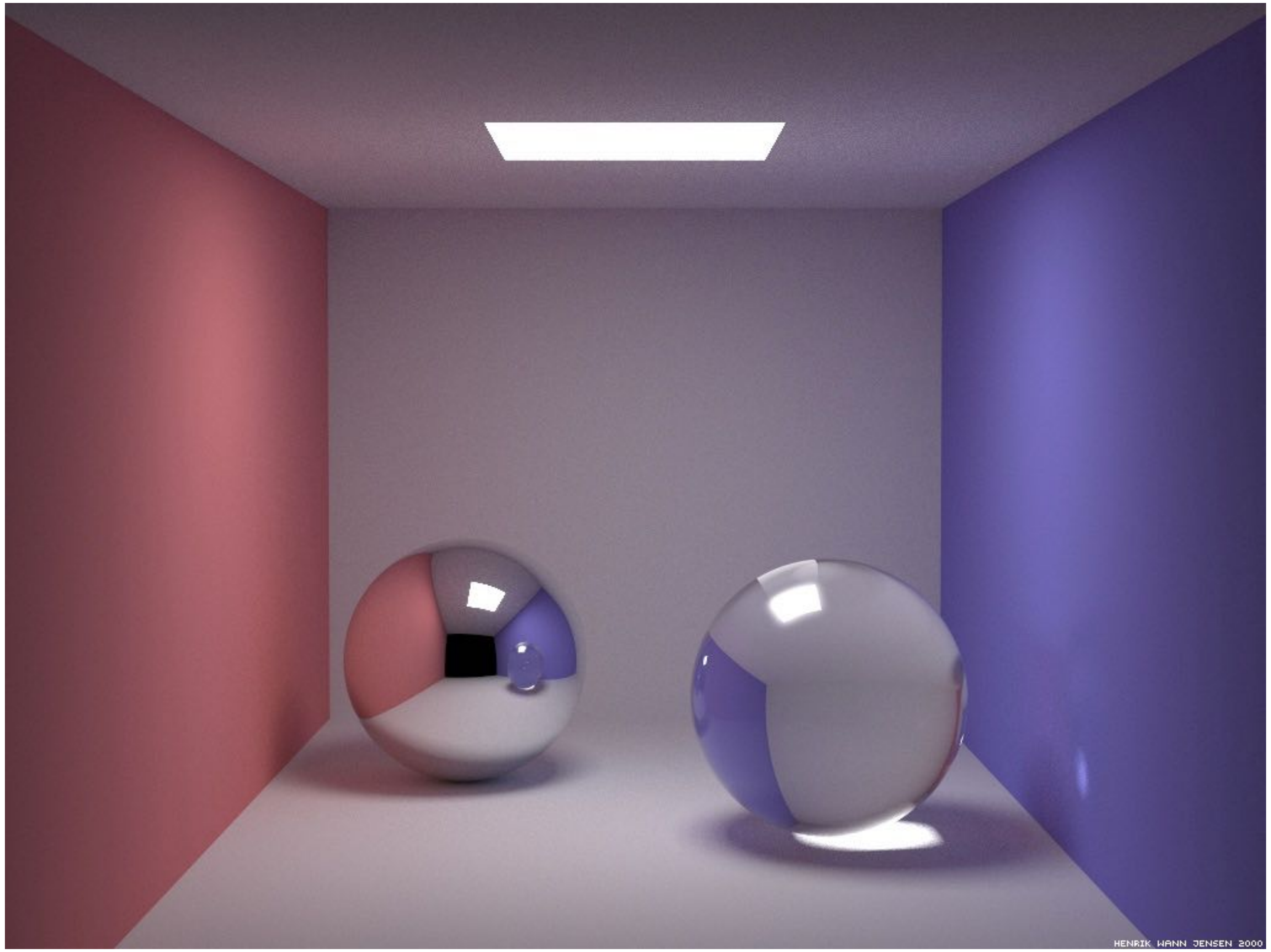
[Credit: Kerbl 2023]

Some questions for the class

- *Recall… when we introduced geometry, we stressed choosing the right representation for the job!*

- If you tell me a task, and then we can access the utility of different representations
  - Describe a scene that needs to be represented
  - Describe what operation you want to perform (e.g., in this lecture, the operation is rendering/visualization, but you can consider other operations: like editing or optimization to fit a measurement)

For example:
Consider representing this scene with ten's of thousands of gaussians vs. two spheres and a few triangles



Some questions for the class

- *Recall… when we introduced geometry, we stressed choosing the right representation for the job!*

- If you tell me a task, and then we can access the utility of different representations
  - Describe a scene that needs to be represented
  - Describe what operation you want to perform (e.g., in this lecture, the operation is rendering/visualization, but you can consider other operations: like editing or optimization to fit a measurement)

Does it make sense to represent this curved surface with a voxel grid?
How many voxels would you need?

Some questions for the class

- *Recall… when we introduced geometry, we stressed choosing the right representation for the job!*

- If you tell me a task, and then we can access the utility of different representations
  - Describe a scene that needs to be represented
  - Describe what operation you want to perform (e.g., in this lecture, the operation is rendering/visualization, but you can consider other operations: like editing or optimization to fit a measurement)

But what about accurately representing these scenes with triangles?

Summary

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

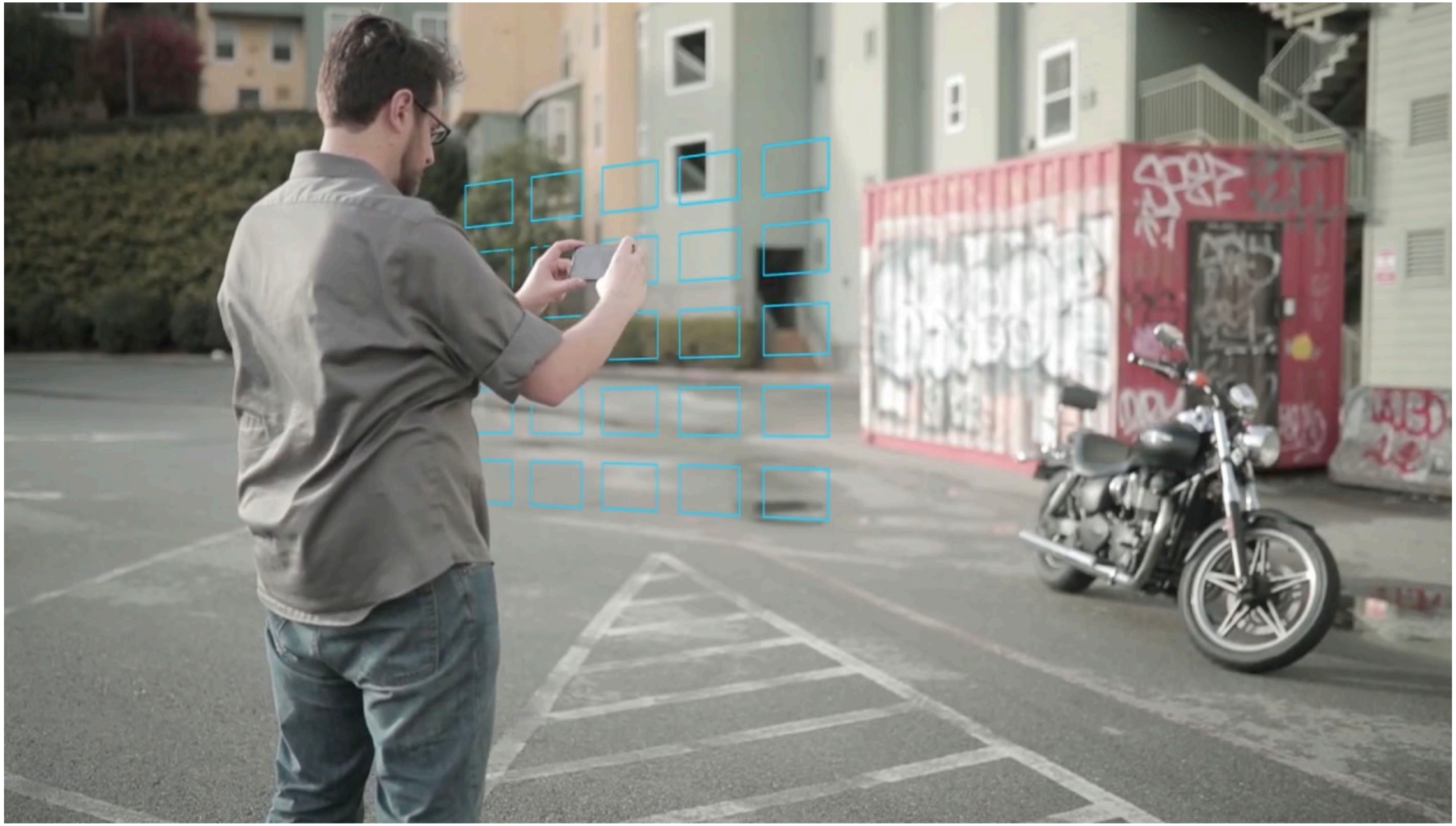
- Traditional uses in rendering:
  - When you need to represent volumetric scene elements
  - When you only have samples of a shape, not a connected surface representation

- These representations present rendering challenges not present when rendering triangles/surfaces
  - Where is the empty space? (ray marching efficiency)
  - What are properties of the surface needed for shading (like the normal!)

- Significant renewed interest in these representations in recent years… for two important reasons
  - Reconstructing geometry representations from samples (e.g., photographs, scans)
  - Generating shapes using generative AI. (e.g., "text prompt" to 3D model)
  - *In both cases, very helpful to have a differentiable renderer, and differentiable rendering is simpler on representations like voxels and gaussians where there's not an explicit notion of boundary.*

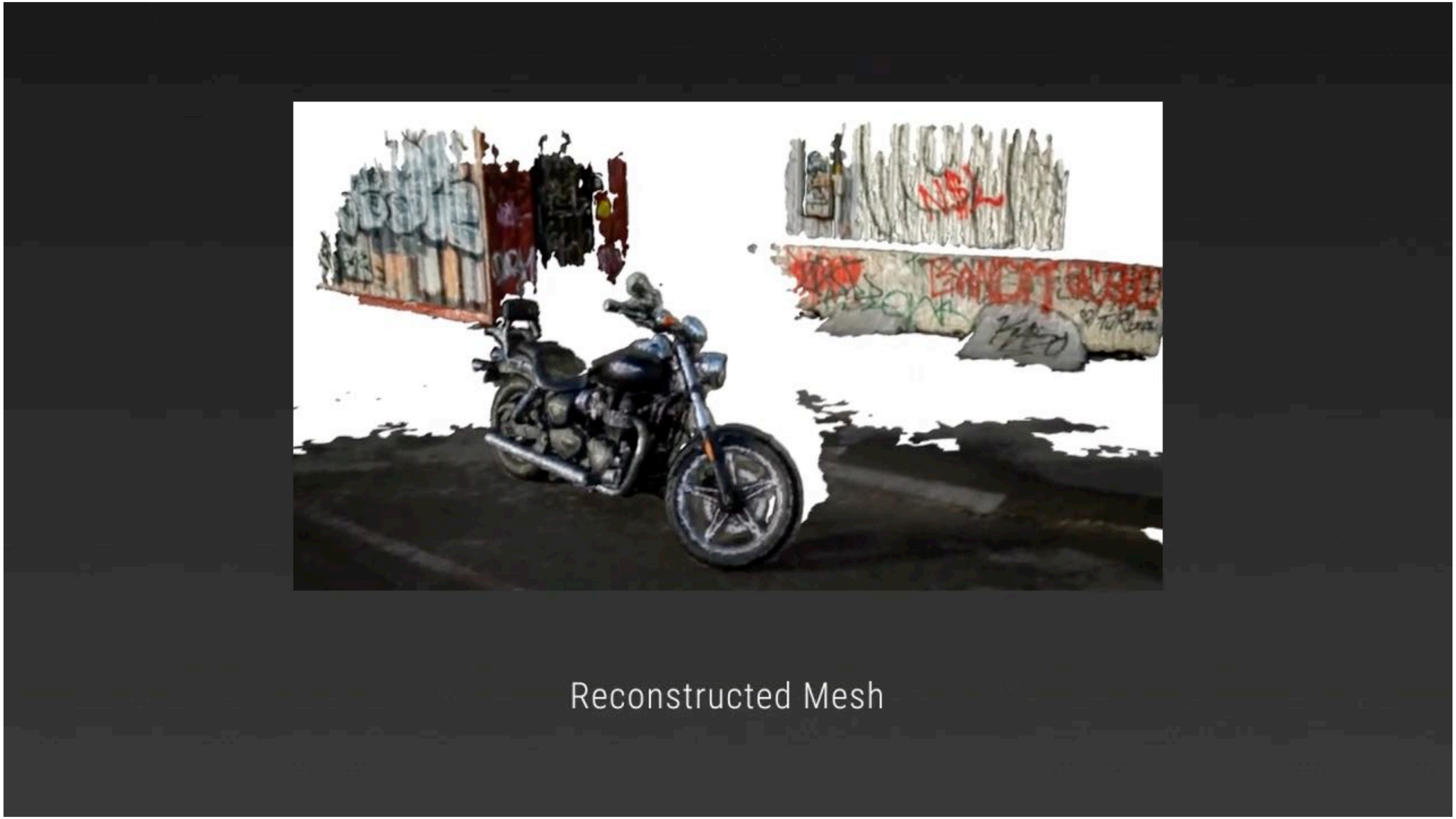
Teaser for next time: an interesting task

- **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 volumetric representation

A "reasonable" volume representing the scene is the one that, when volume rendered from the viewpoint of the photograph, produces a picture that looks like the photograph.

Credit: Lombardi 2019

Summary

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