Lecture 9:

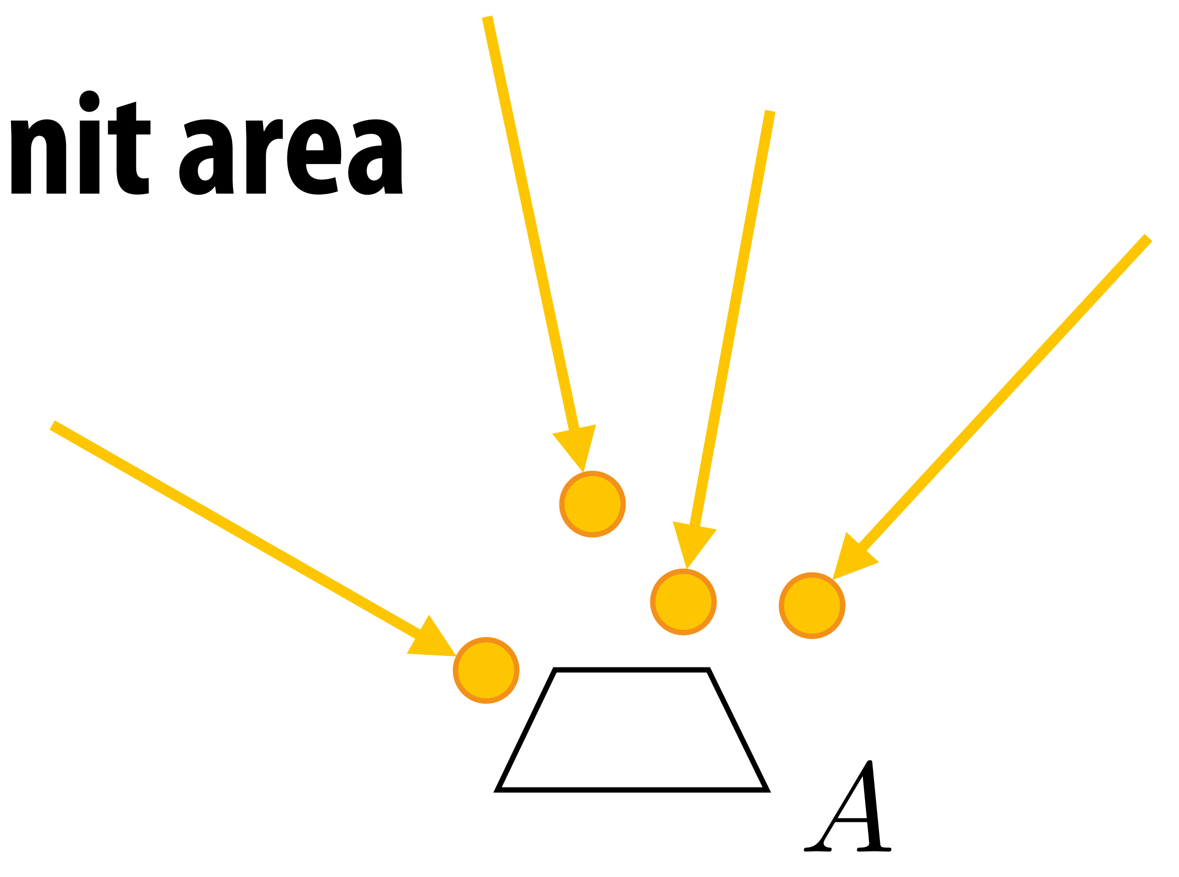
Recovering scene representations using gradient-based optimization

Computer Graphics: Rendering, Geometry, and Image Manipulation
Stanford CS248A, Winter 2026

Review of last class
(Light and reflectance)

Review: irradiance

- **Irradiance: flux (energy per unit time=power) per unit area**



Given a sensor of with area *A*, we can consider the average flux over the entire sensor area:

$$\frac{\Phi}{A}$$

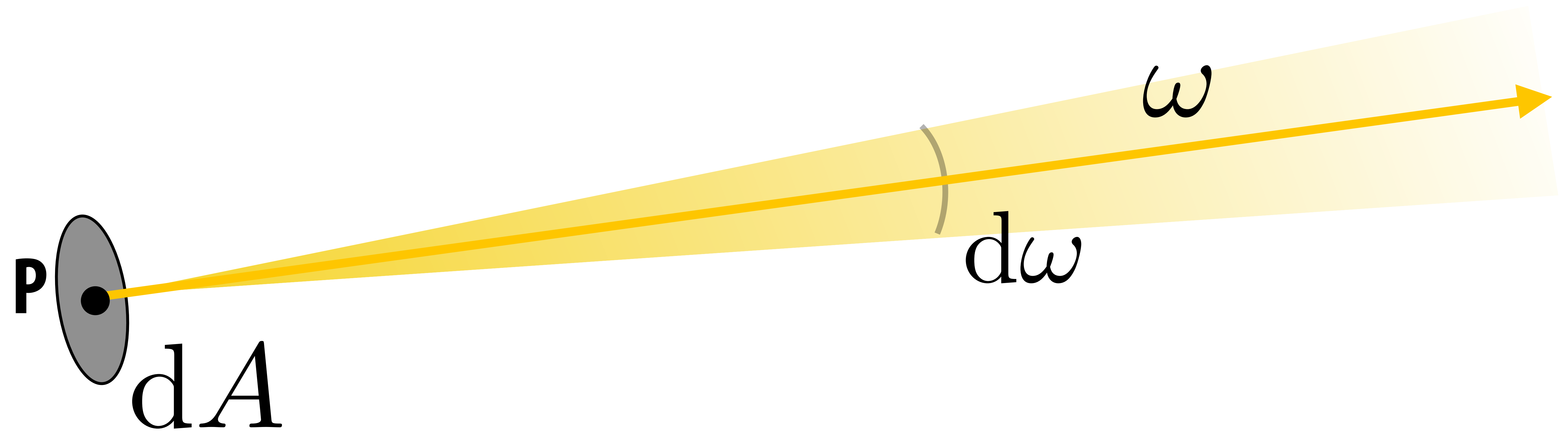
Irradiance (E) is given by taking the limit of area at a single point on the sensor:

$$E(\mathrm{p}) = \lim_{\Delta \to 0} \frac{\Delta\Phi(\mathrm{p})}{\Delta A} = \frac{\mathrm{d}\Phi(\mathrm{p})}{\mathrm{d}A} \quad \left[\frac{\mathrm{W}}{\mathrm{m}^2}\right]$$

Units = Watts per area

Review: radiance

- Radiance (L) is power along an infinitesimally small beam
- The solid angle density of irradiance (irradiance per unit direction) where the differential surface area is oriented to face in the direction ω

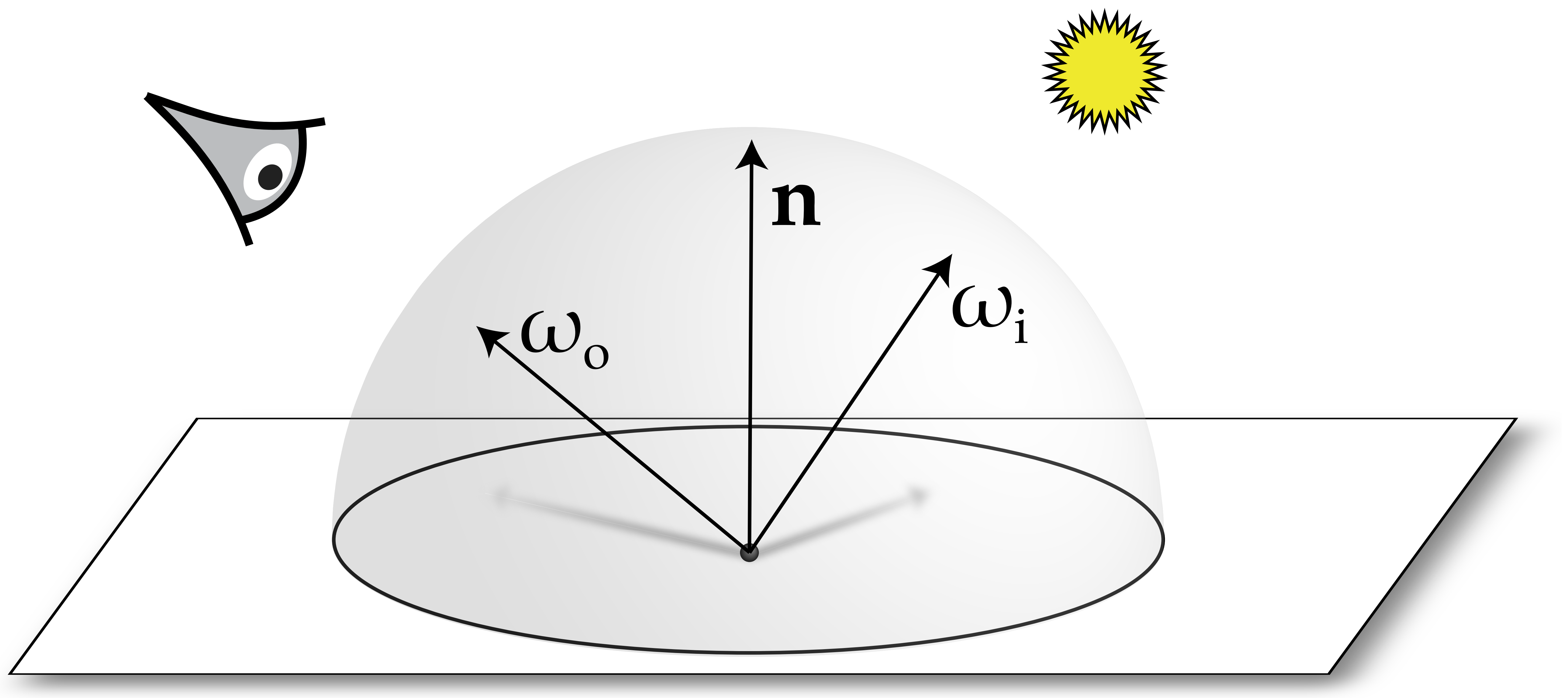


In other words, radiance is energy along a ray defined by origin point *p* and direction ω

$$L(\mathrm{p},\omega) = \lim_{\Delta\to 0} \frac{\Delta\Phi(\mathrm{p},\omega)}{\Delta A \Delta\omega} = \frac{\mathrm{d}^2\Phi(\mathrm{p},\omega)}{\mathrm{d}A\,\mathrm{d}\omega}$$

The reflection equation

Gives radiance reflected from point p in direction direction ω_0 due to light incident on the surface at p.

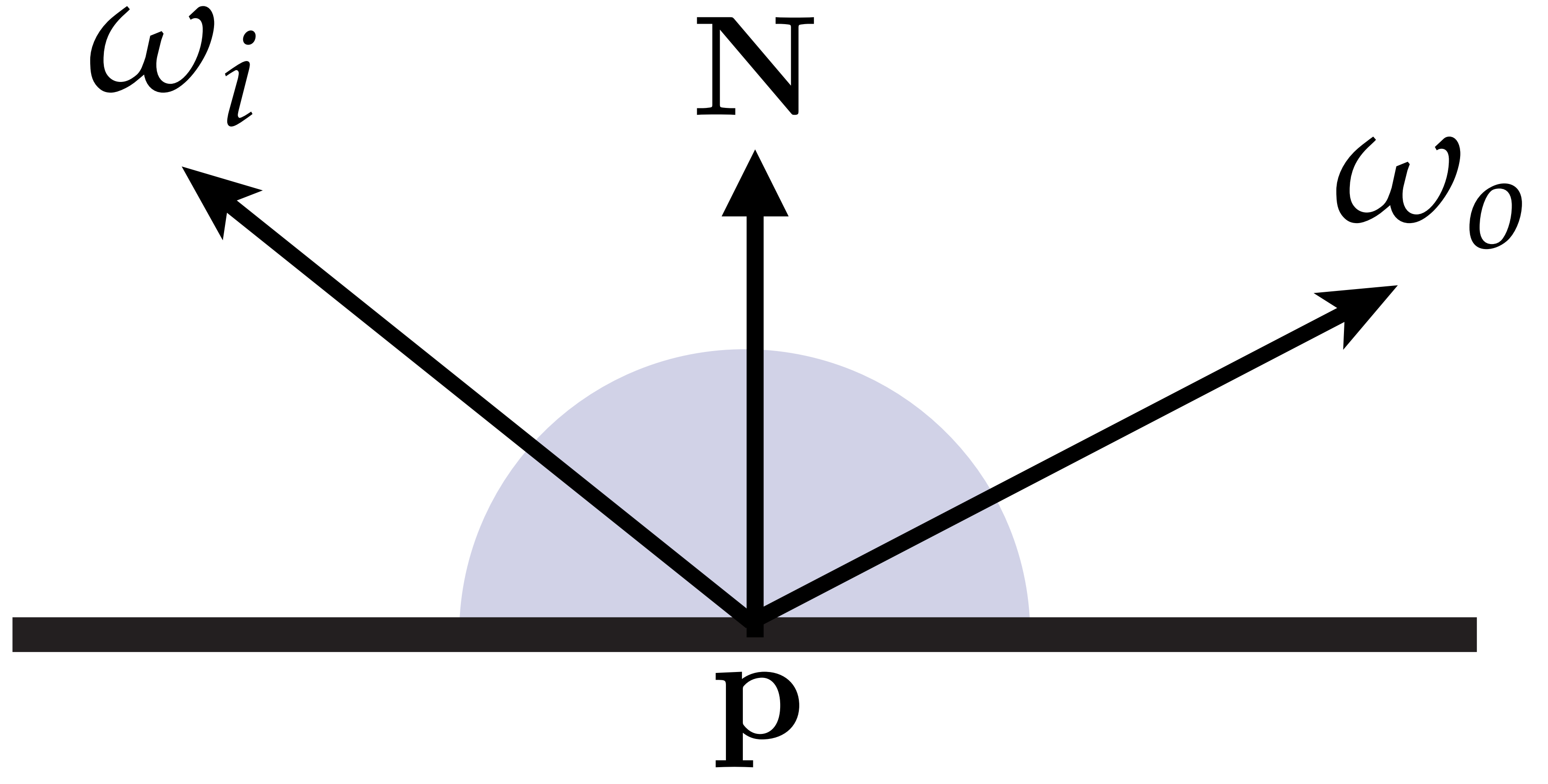


$$L_o(p, \omega_o) = \int_{\Omega^2} \underbrace{f_r(p, \omega_i \rightarrow \omega_o)}_{\textbf{BRDF}} \underbrace{L_i(p, \omega_i) \cos\theta_i}_{\textbf{Illumination}} \, d\omega_i$$

Bidirectional reflectance distribution function (BRDF)

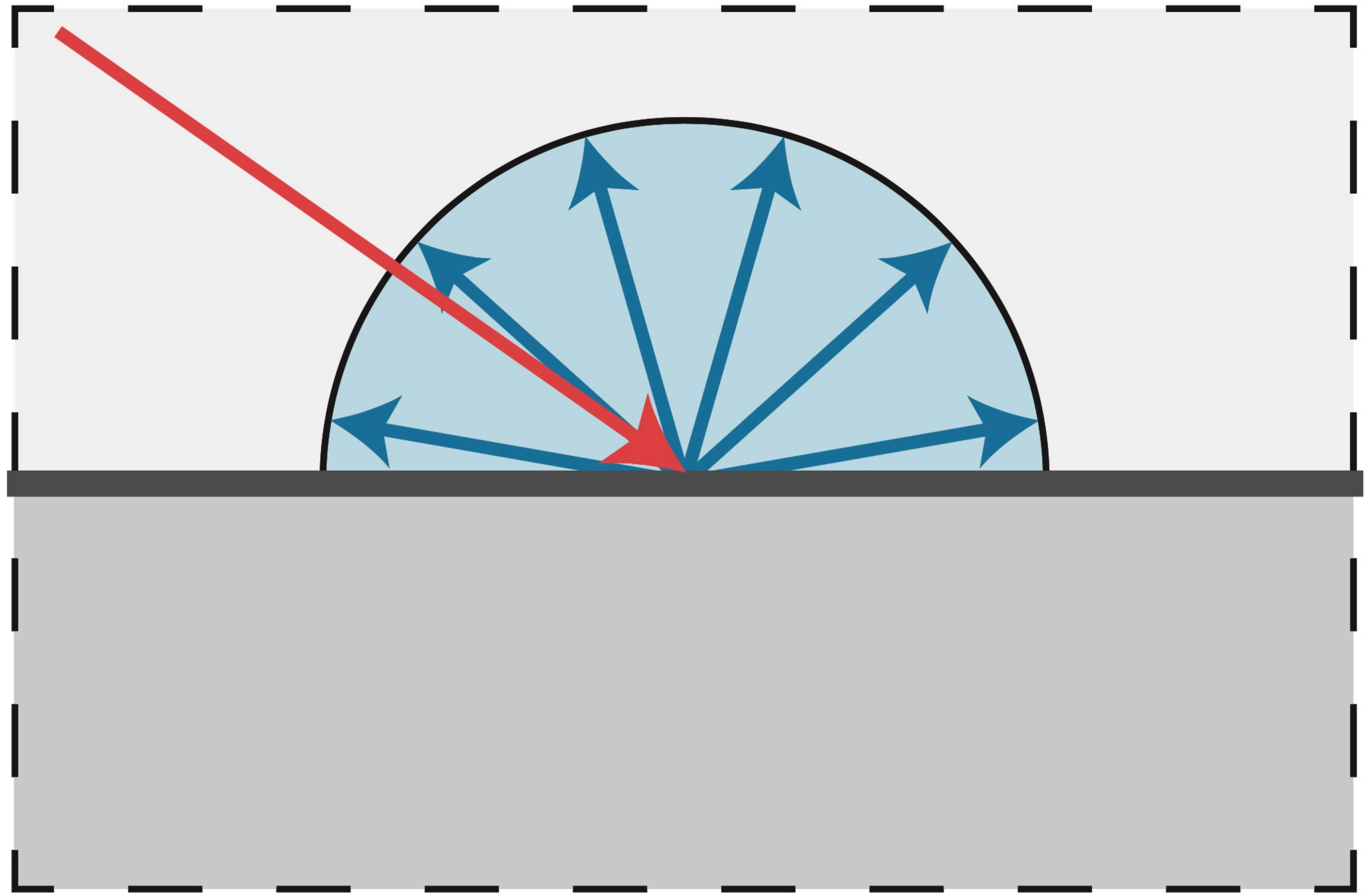
Gives fraction of light arriving at surface point p from incoming direction* ω_i is reflected in the direction ω_0 (outgoing direction)

$$f(\mathbf{p}, \omega_i, \omega_o)$$



*** (Convention: ω_i is oriented out from the surface "towards the incoming direction")**

What is this material?



Light is scattered equally in all directions

Today:

a small diversion before we dive deeper into reflection and materials next class

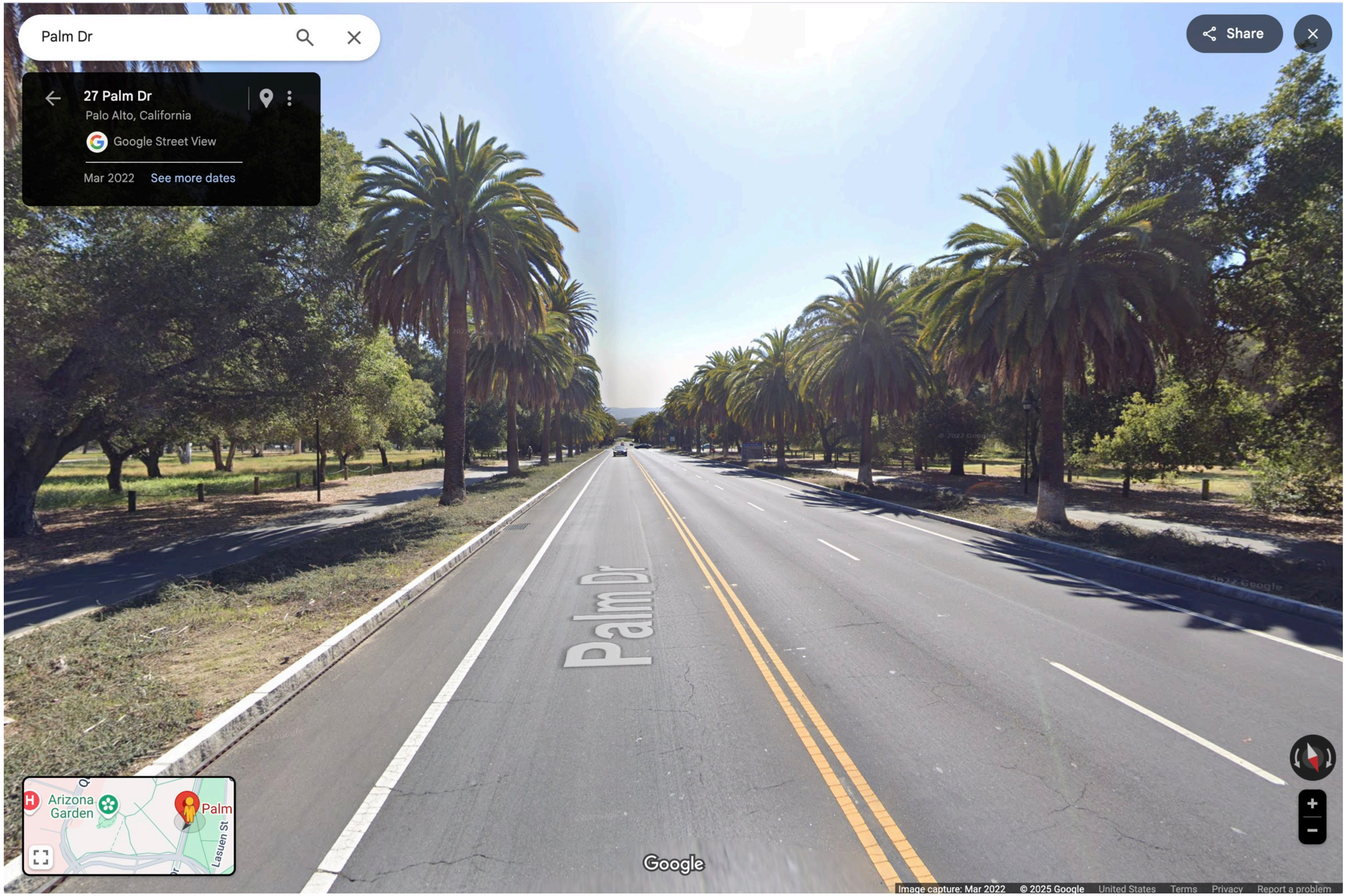
So far in class, our primary tasks of interest have been simulation

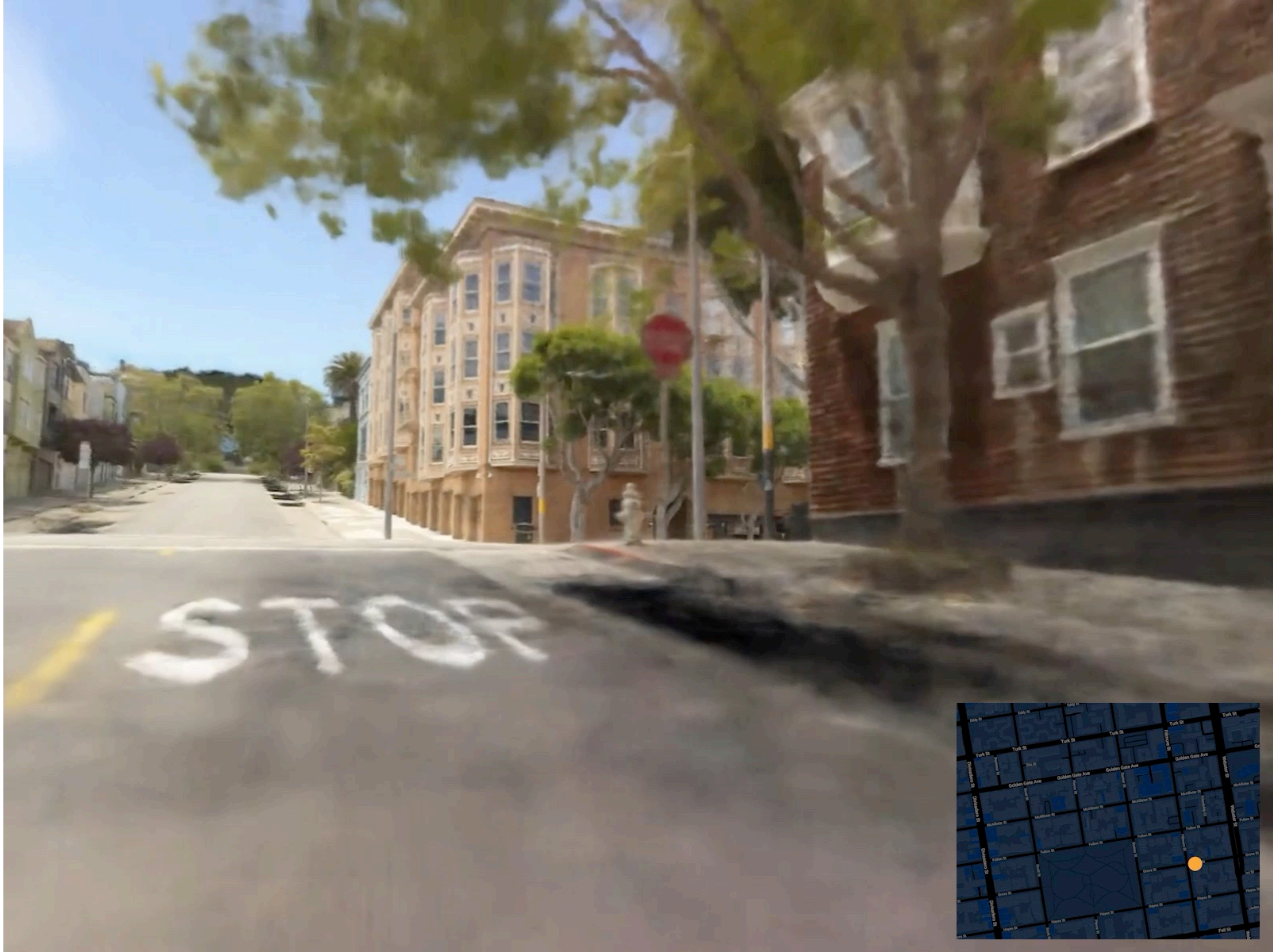
Simulating what a scene would look like (rendering)
Computing geometric relationships between objects
(e.g, inside/outside, distance to)

A longstanding challenge in computer graphics…

- Acquiring high-quality 3D content for rendering
- Consider making a high-quality 3D model and texture maps depicting Josephine the graphics cat…

Google Street View

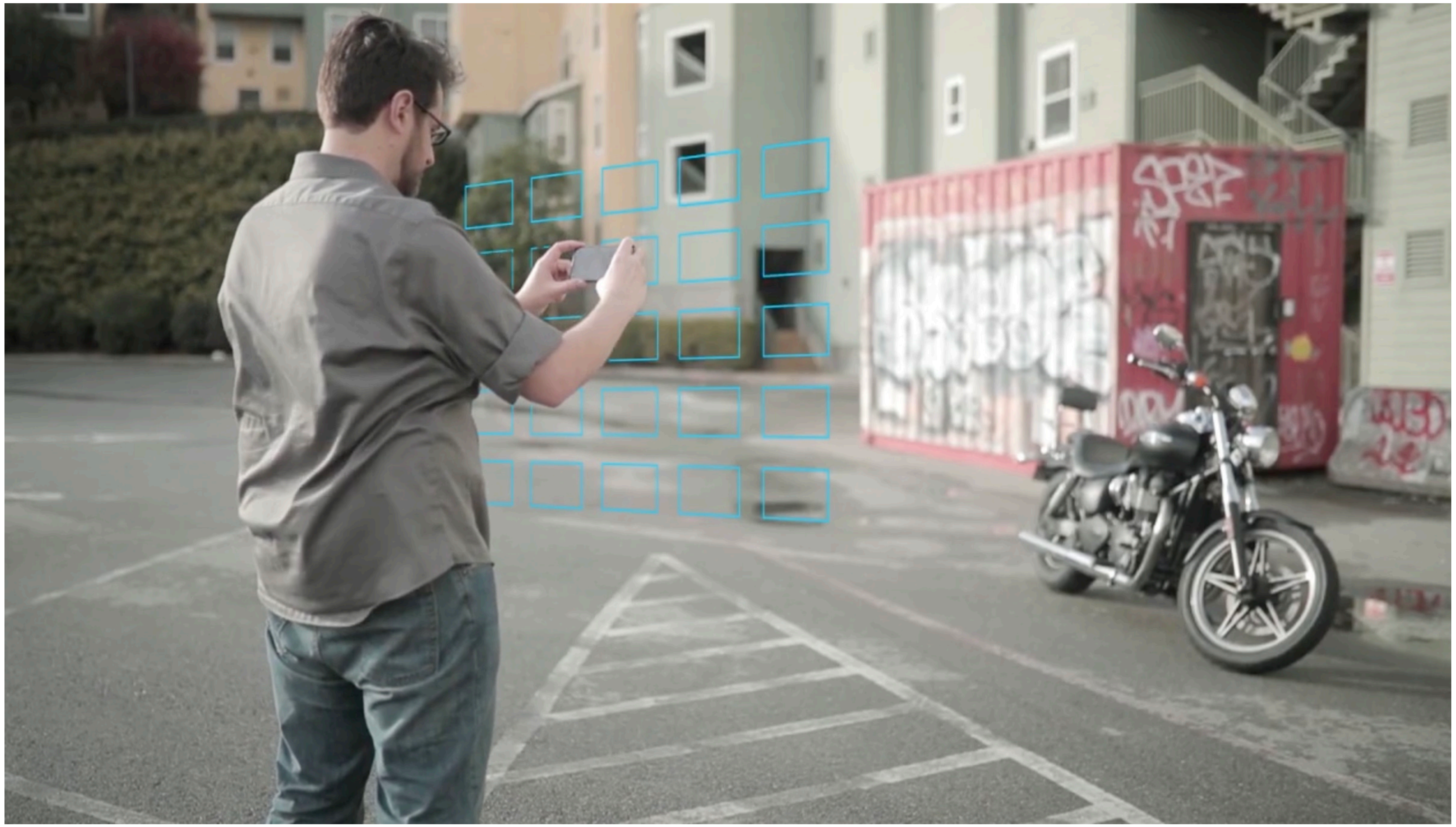




Credit: Tancik et al. 2022

An interesting task

- Given a collection of photographs (from known camera viewpoints)
- Recover a representation of the 3D scene (surface locations + color at each point on surface) that you could use for rendering the scene from novel viewpoints

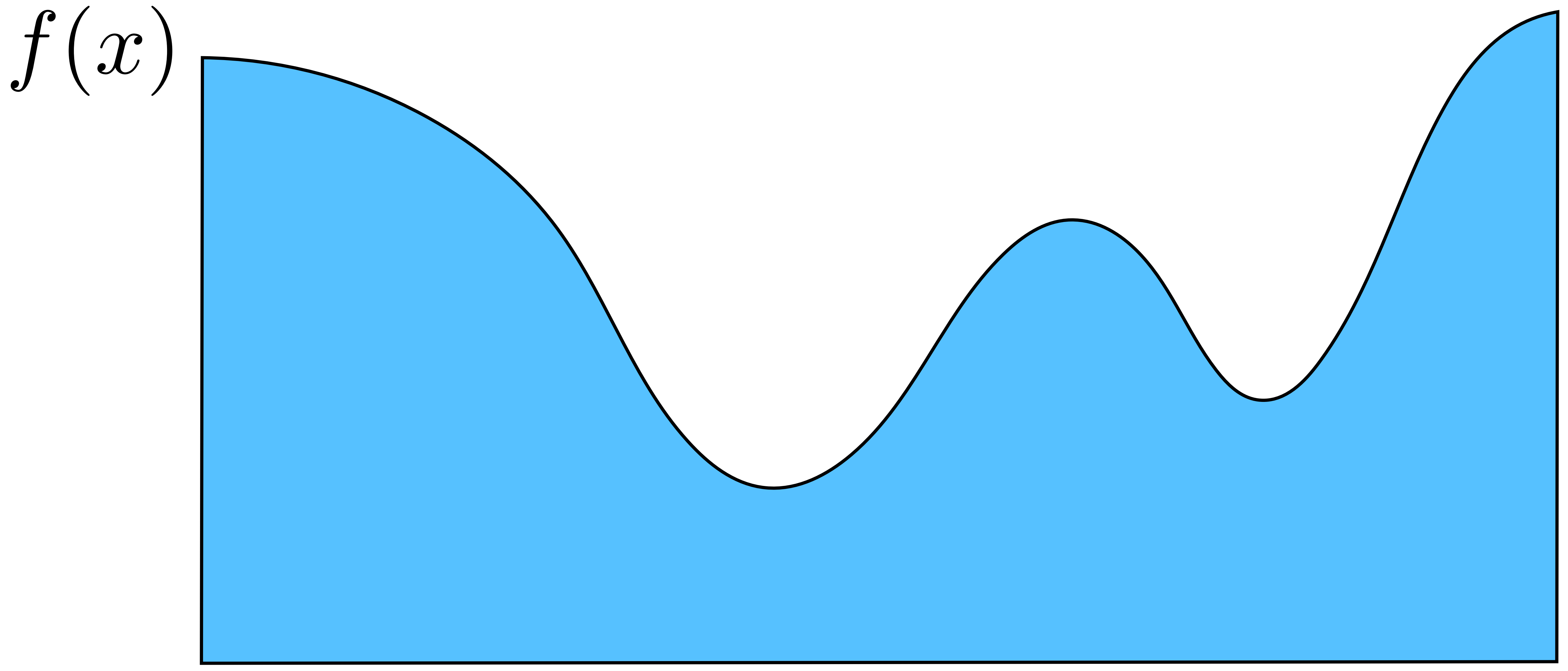


Credit: Mildenhall 2019

Mini intro to gradient-based optimization

Imagine we have a function $f(x)$

How can we find the minimum of the function?

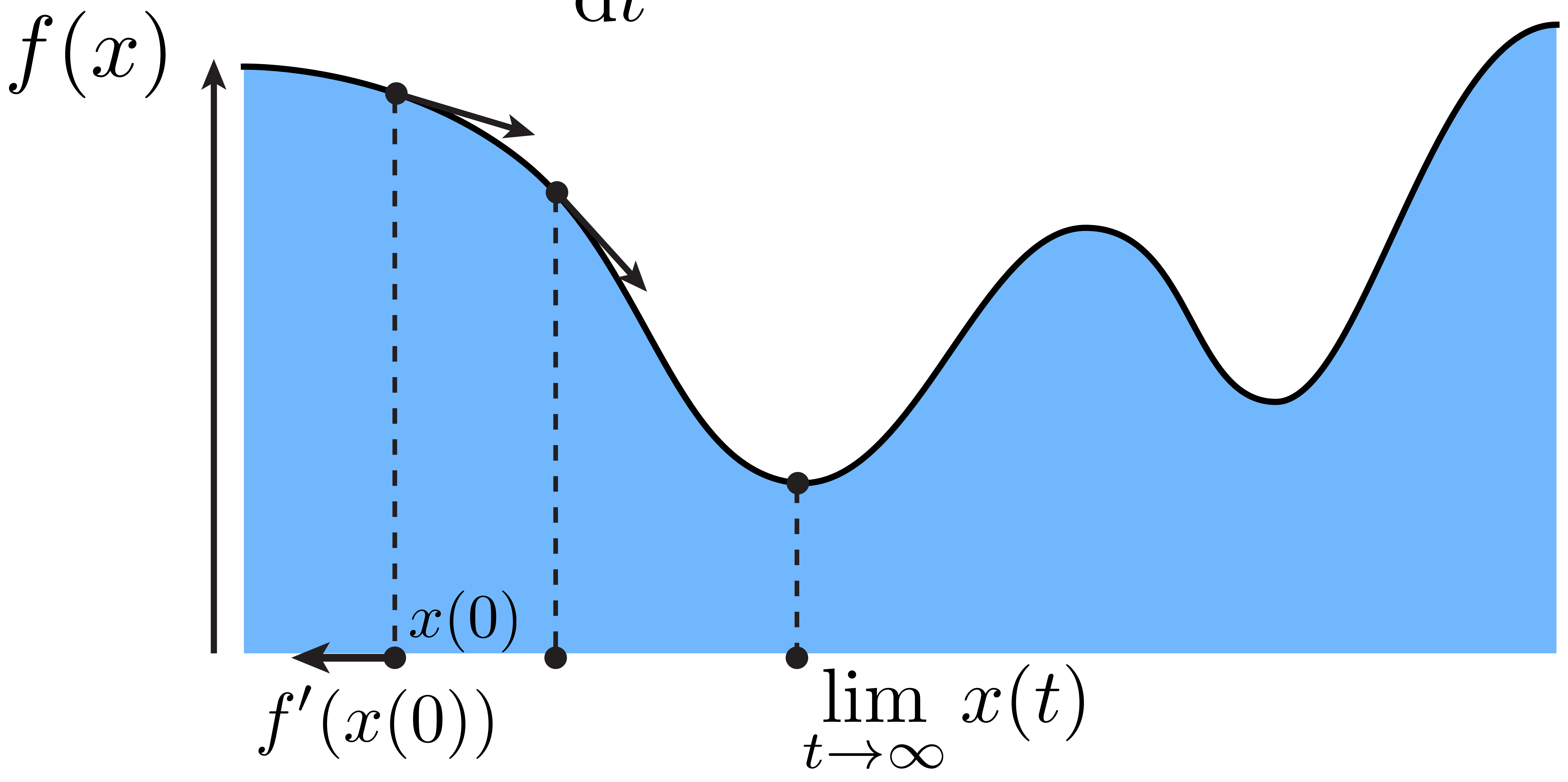


Descent methods

Gradient descent (1D)

- Basic idea: follow the gradient "downhill" until it's zero

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = -f'(x(t))$$



- Do we always end up at a (global) minimum?
- How do we compute gradient descent in practice?

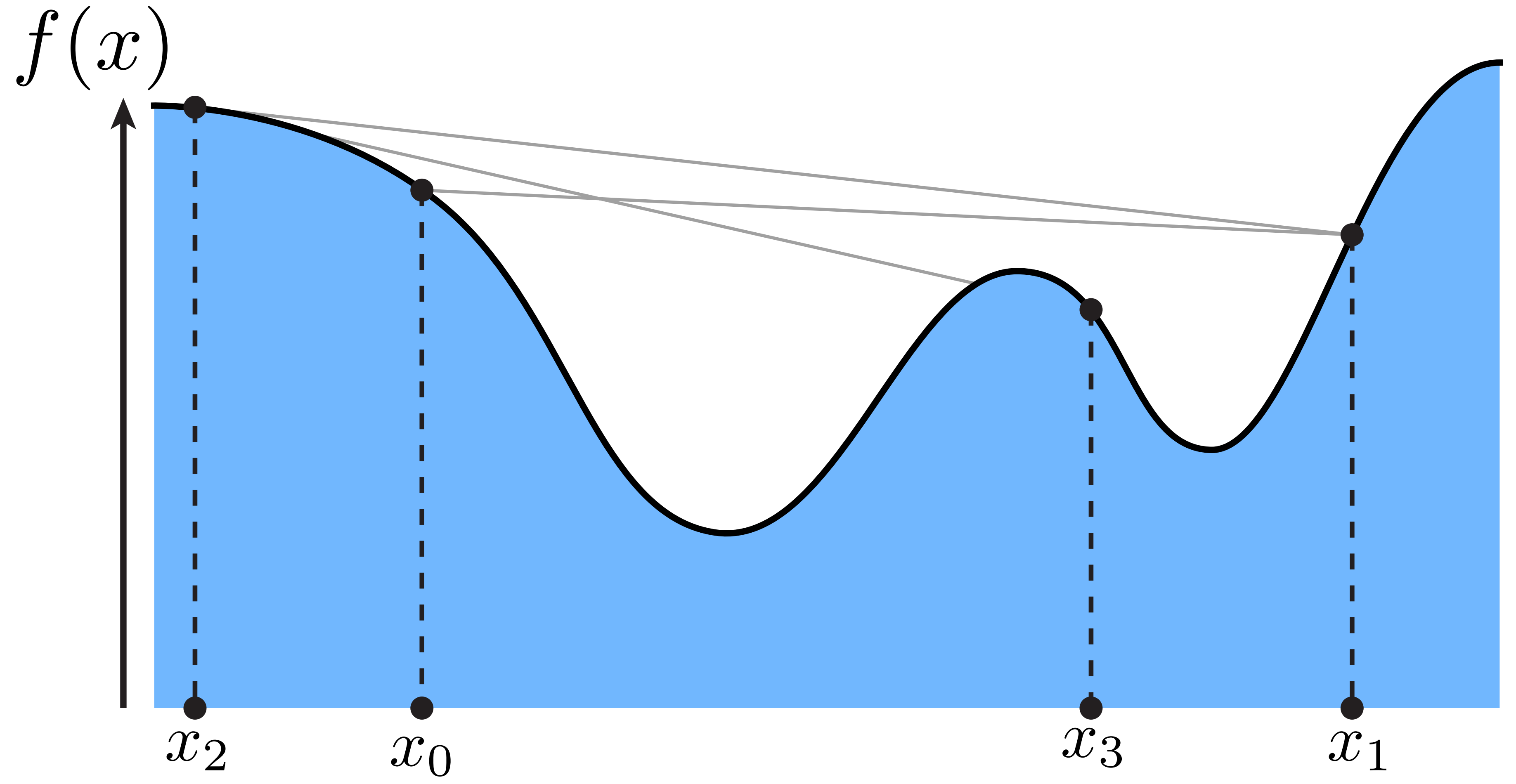
Gradient descent algorithm (1D)

- "Walk downhill"
- One simple way: forward Euler:

$$x_{k+1} = x_k - \tau f'(x_k)$$

new estimate

step size



- Q: How do we pick the step size?
- If we're not careful, we'll go zipping all over the place; won't make any progress.
- Basic idea: use *"step control"* to determine step size based on value of function and its derivatives
- For now we will do something simple: make τ *small!*

Gradient descent algorithm (n-D)

- Q: How do we write gradient descent equation in general?

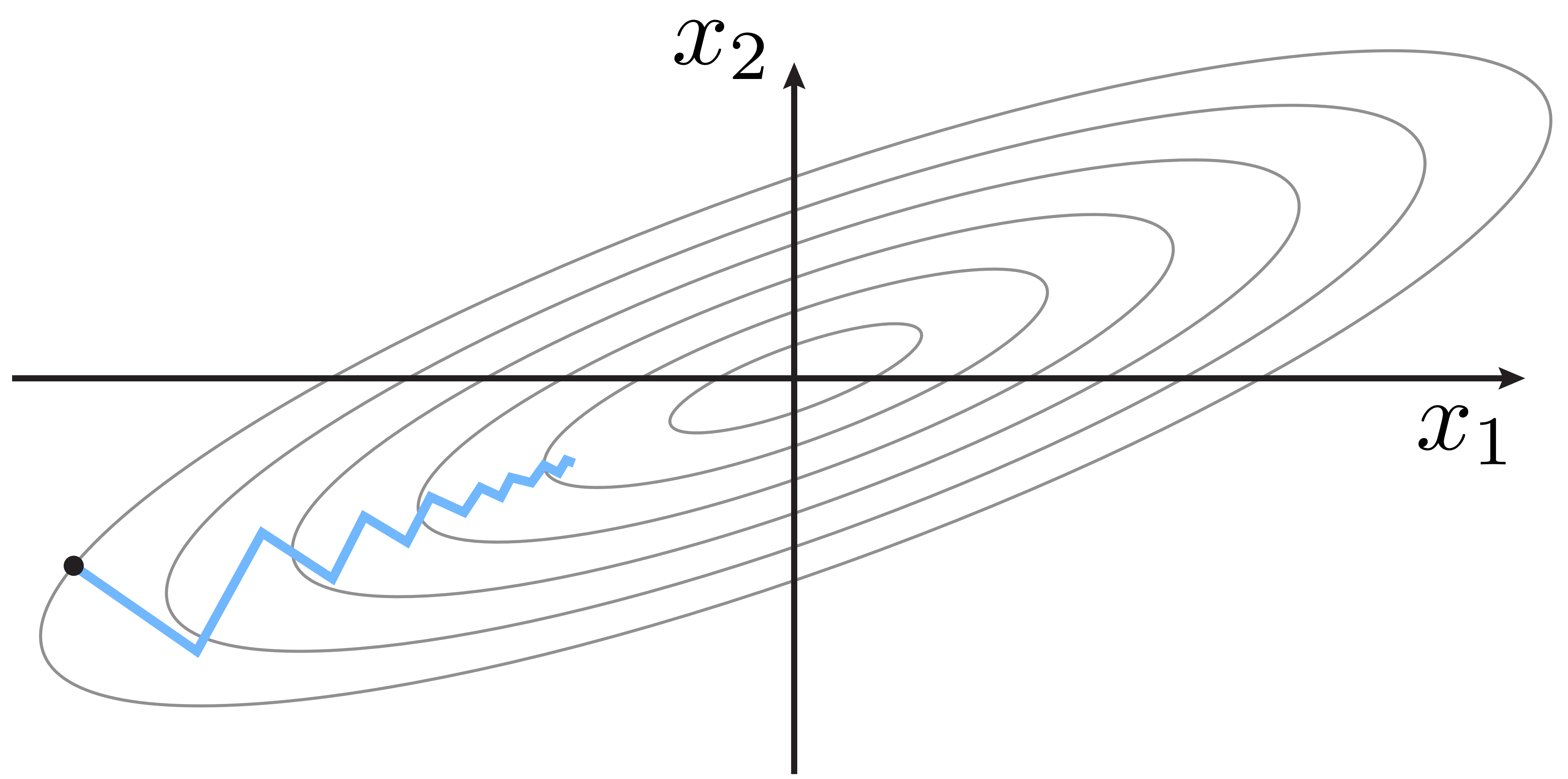
$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}(t) = -\nabla f(\mathbf{x}(t))$$

$$\begin{bmatrix} \frac{\mathrm{d}f}{\mathrm{d}\mathbf{x}_0} & \frac{\mathrm{d}f}{\mathrm{d}\mathbf{x}_1} & \cdots & \frac{\mathrm{d}f}{\mathrm{d}\mathbf{x}_{N-1}} \end{bmatrix}^T$$

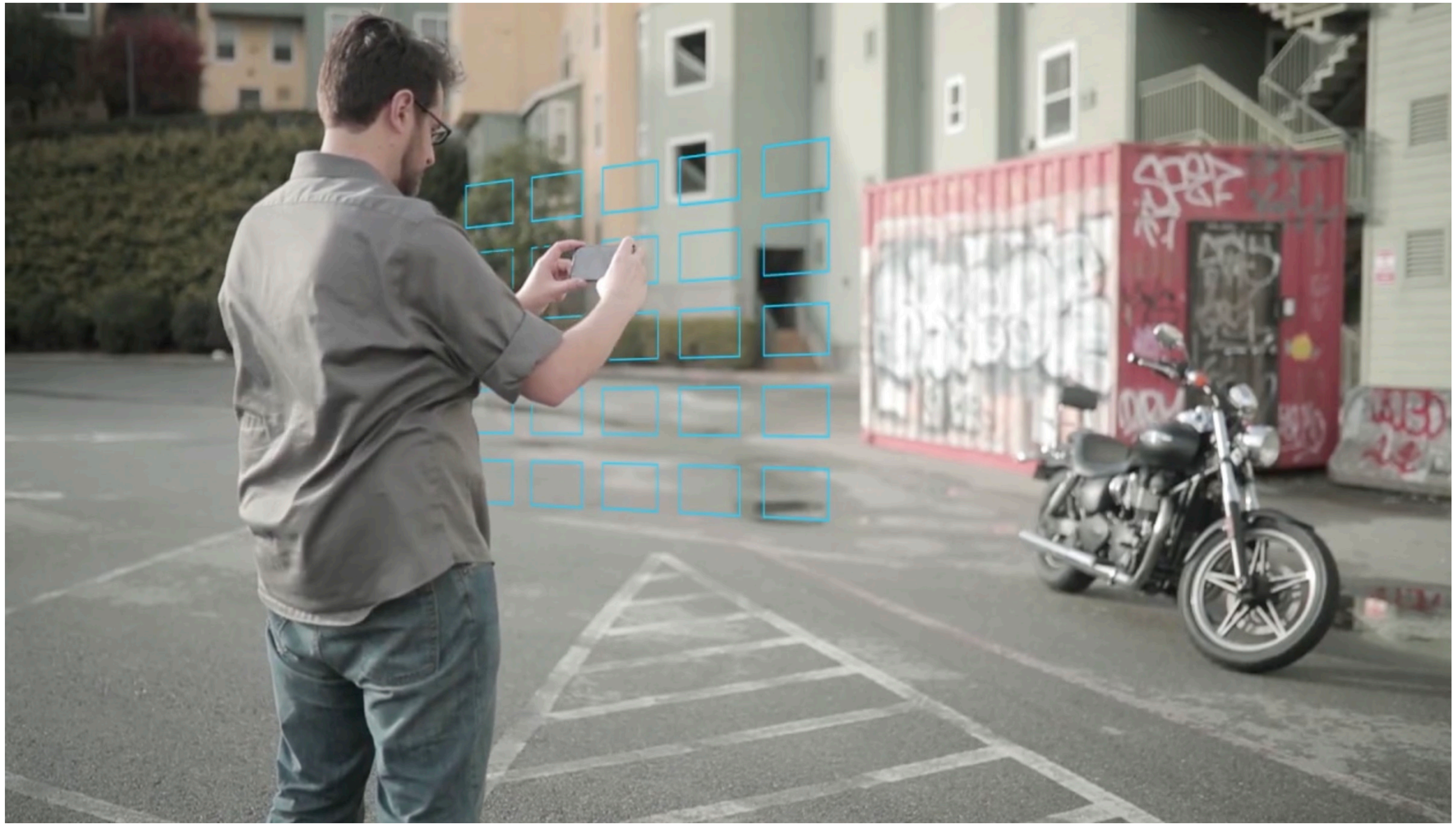
- Q: What's the corresponding discrete update?

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \tau\nabla f(\mathbf{x}_k)$$

- Basic challenge in nD:
  - solution can "oscillate"
  - takes many, many small steps
  - very slow to converge



Using gradient descent to recover scenes



But first let's consider a simpler toy problem

- **Let's try recovering the pixels of a texture map such that sampling the texture using bilinear interpolation approximates a 2D signal depicting a scene**

- **Given a 2D function we can measure (sample):** $\text{scene}(u, v)$
- **Let** $\text{texture}(\mathbf{x}, u, v)$ **be the result of sampling from NxN texture x using bilinear interpolation**

- **In the formulation of the previous slides:**
  - $\mathbf{x}$ **is an array of N² pixel values (unknown N x N texture map)**
  - $f(\mathbf{x}) = \boxed{\sum_i (\text{scene}(u_i, v_i) - \text{texture}(\mathbf{x}, u_i, v_i))^2}$

For a collection of samples (u_i, v_i)

We seek to minimize the sum of squared differences of textured result and function we are taking measurements of.

But how do we compute $\nabla f(\mathbf{x})$ **?**

Computing derivatives using the chain rule

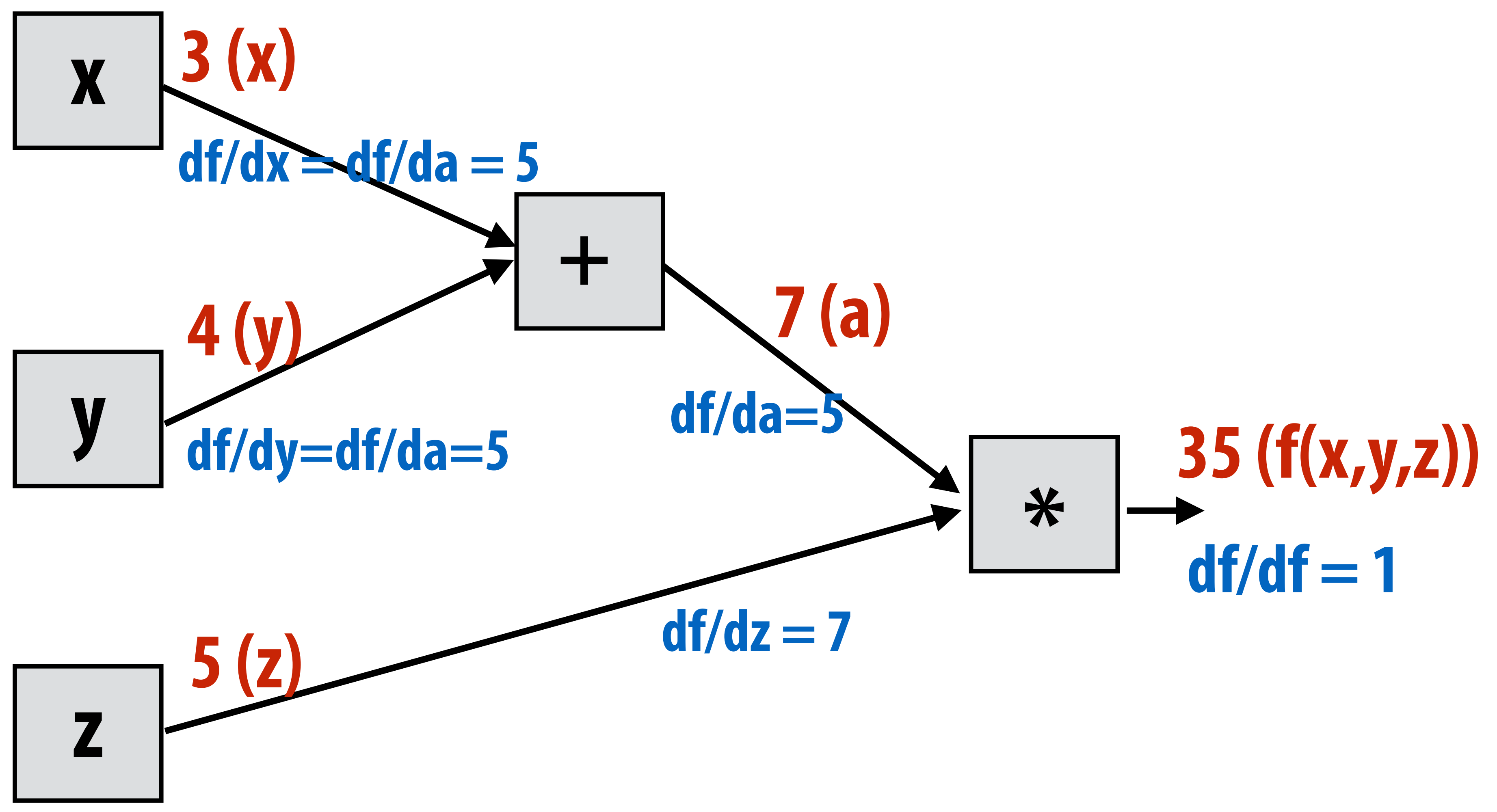
$$f(x,y,z) = (x+y)z = az$$

Where: $a = x + y$

$$\frac{df}{da} = z \quad \frac{da}{dx} = 1 \quad \frac{da}{dy} = 1$$

So, by the derivative chain rule:

$$\frac{df}{dx} = \frac{df}{da}\frac{da}{dx} = z$$



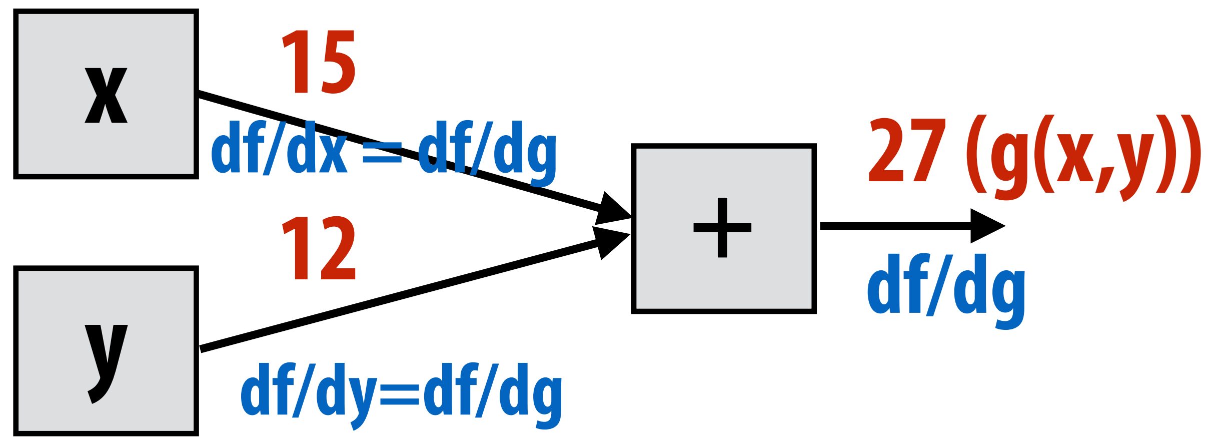
Red = output of node
Blue = df/dnode

Backpropagation (a form of reverse mode autodiff)

Red = output of node

Blue = df/dnode

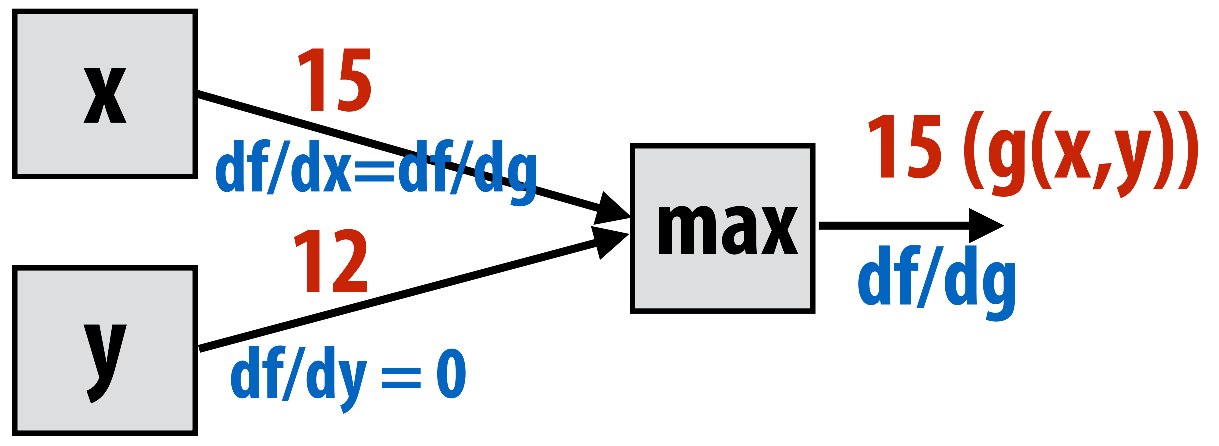
Recall: $\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx}$



$$g(x,y) = x + y$$

$$\frac{dg}{dx} = 1\,,\ \frac{dg}{dy} = 1$$

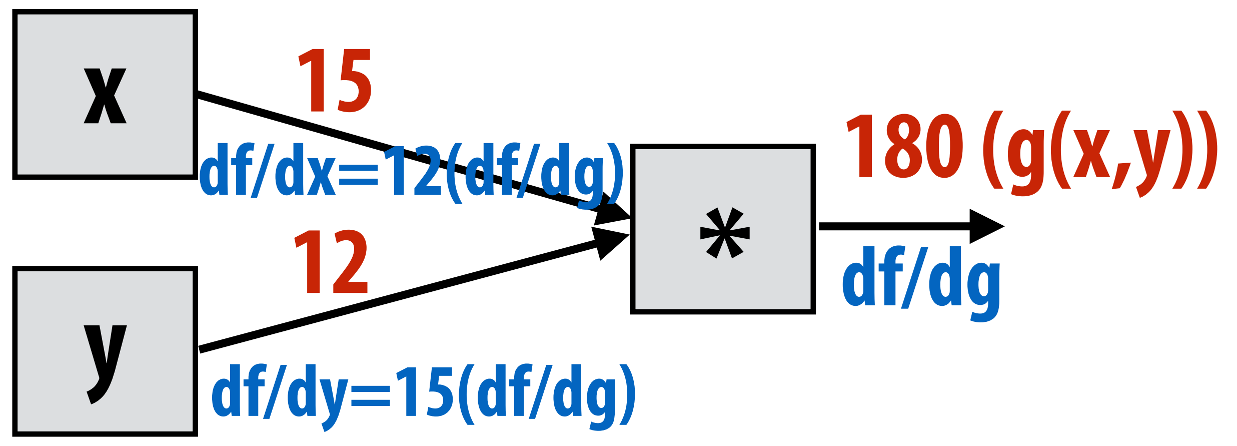
"Sum copies gradients"



$$g(x,y) = \max(x,y)$$

$\frac{dg}{dx} =$ **1, if x > y** / **0, otherwise**

"Max routes gradient"

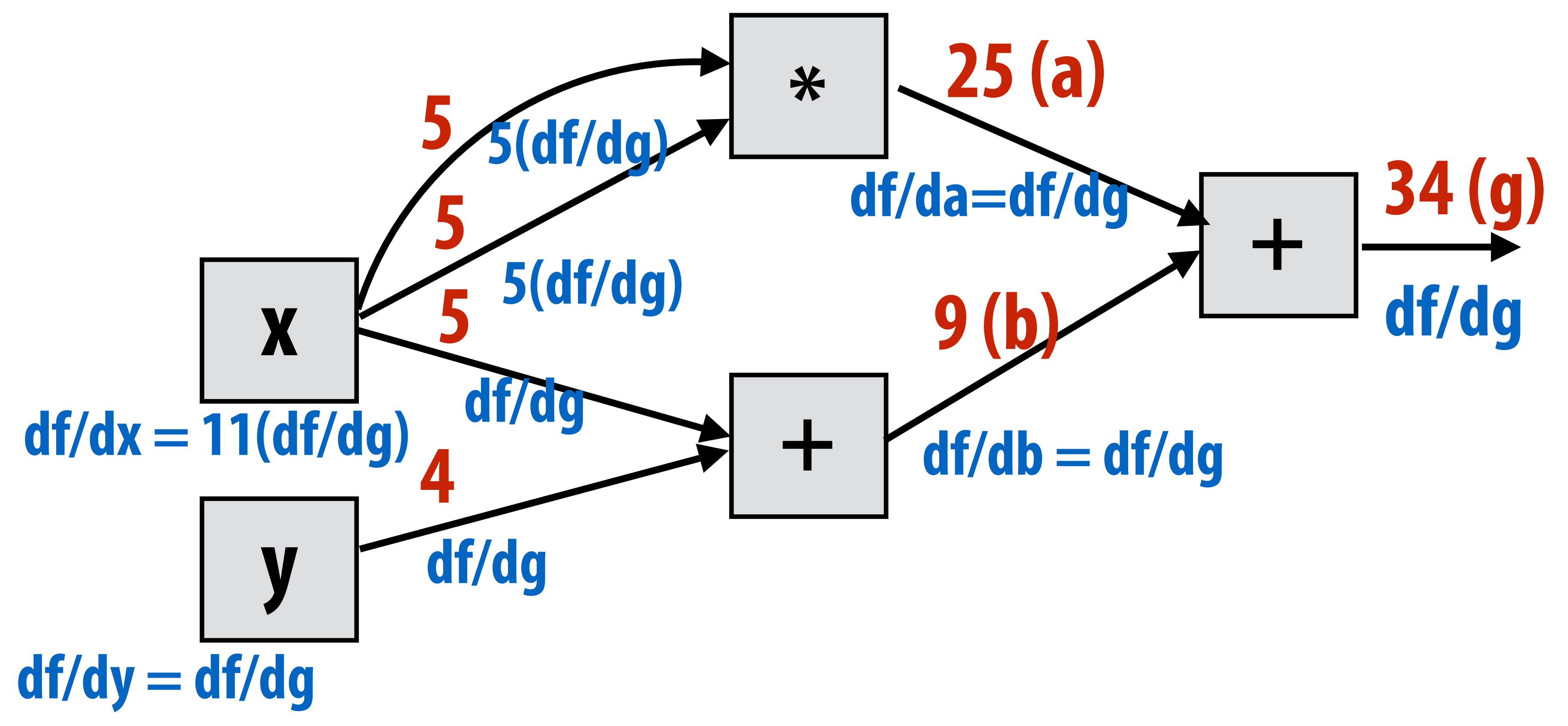


$$g(x,y) = xy$$

$$\frac{dg}{dx} = y\,,\ \frac{dg}{dy} = x$$

"Multiply scales gradient by opposite term"

Backpropagation with multiple uses of an input variable



$$g(x,y) = (x+y) + x*x = a + b$$

$$\frac{da}{dx} = 2x\,, \quad \frac{db}{dx} = 1$$

$$\frac{dg}{da} = 1\,, \frac{dg}{db} = 1$$

$$\frac{dg}{dx} = \frac{dg}{da}\frac{da}{dx} + \frac{dg}{db}\frac{db}{dx} = 2x + 1$$

Sum gradients from each use of variable:

Here:

$$\begin{aligned}\frac{df}{dx} &= \frac{df}{dg}\frac{dg}{dx} \\ &= \frac{df}{dg}\left(\frac{dg}{da}\frac{da}{dx} + \frac{dg}{db}\frac{db}{dx}\right) \\ &= \frac{df}{dg}(1 \times 2x + 1 \times 1) \\ &= \frac{df}{dg}(2x+1) \\ &= \frac{df}{dg}(11)\end{aligned}$$

Differentiating our loss function

$$f(\mathbf{x}) = \sum_i (\mathrm{scene}(u_i, v_i) - \mathrm{texture}(\mathbf{x}, u_i, v_i))^2$$

$$= \sum_i g(u_i, v_i)^2$$ **where** $$g(u, v) = \mathrm{scene}(u, v) - \mathrm{texture}(\mathbf{x}, u, v)$$

$$\frac{\mathrm{d}g}{\mathrm{d}\mathbf{x}} = \frac{\mathrm{d}g}{\mathrm{d}\,\mathrm{texture}} \frac{\mathrm{d}\,\mathrm{texture}}{\mathrm{d}\mathbf{x}} = -\frac{\mathrm{d}\,\mathrm{texture}}{\mathrm{d}\mathbf{x}}$$

So…

$$\nabla f(\mathbf{x}) = 2 \sum_i g(u_i, v_i) \frac{\mathrm{d}g}{\mathrm{d}\mathbf{x}} = -2 \sum_i g(u_i, v_i) \frac{\mathrm{d}\,\mathrm{texture}(u_i, v_i)}{\mathrm{d}\mathbf{x}}$$

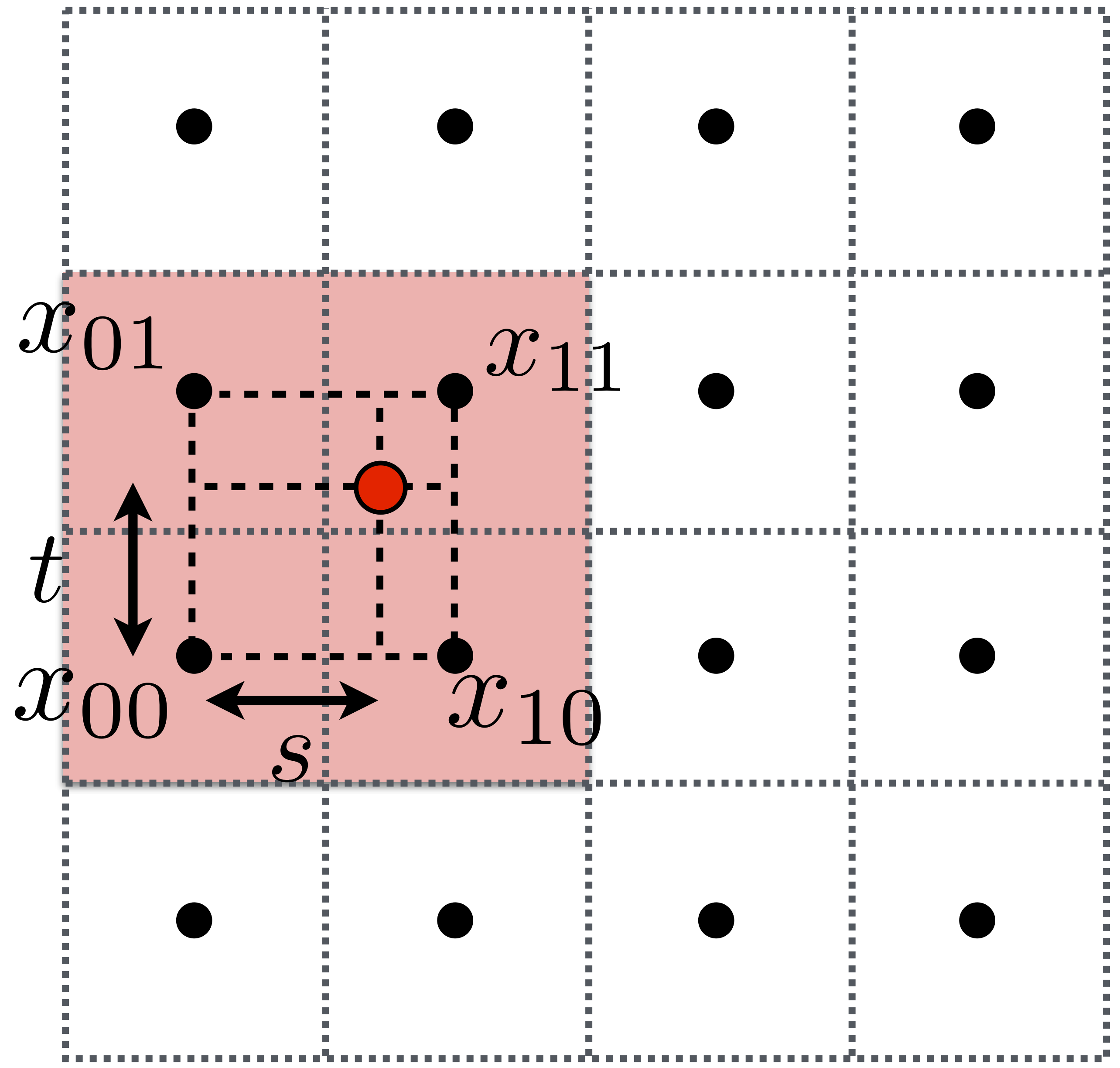
Recall: bilinear filtering

$\text{texture}(\mathbf{x}, u, v)$:

Let x00, x10, x01, x11 be the samples of texture $\mathbf{x}$ surrounding (u,v)

Let (s, t) be the x and y fractional offsets

```
w00 = (1.0 - s) * (1.0 - t);
w10 = s * (1.0 - t);
w01 = (1.0 - s) * t;
w11 = s * t;

result = w00 * x00 + w10 * x01 + w01 * x10 + w11 * x11
```



So what is $\dfrac{\mathrm{d}\,\text{texture}(\mathbf{x}, u, v)}{\mathrm{d}\mathbf{x}}$?

Simple gradient descent algorithm for recovering texture values

```
Let X = random texture values

while (loss too high):  # f(X) is too large

   Let UV = vector of (u_i, v_i) sample positions

   grad = f_grad(UV)

   X += -grad * step_size;
```

Gradient of the L2 difference between the value of the bilinearly filtered texture X, and the target signal we are trying to recover
(When measured at the given array of sample points)

$$f(\mathbf{x}) = \sum_i (\mathrm{scene}(u_i, v_i) - \mathrm{texture}(\mathbf{x}, u_i, v_i))^2$$

$$\nabla f(\mathbf{x}) = 2\sum_i g(u_i, v_i)\frac{\mathrm{d}g}{\mathrm{d}\mathbf{x}} = -2\sum_i g(u_i, v_i)\frac{\mathrm{d}\,\mathrm{texture}(u_i, v_i)}{\mathrm{d}\mathbf{x}}$$

Using Slang to automatically compute derivatives

The Slang compiler provides auto-differentiation services (backward autodial example below)

```
[Differentiable]
float foo(float a, float b)
{
    return a * b * b;
}
```

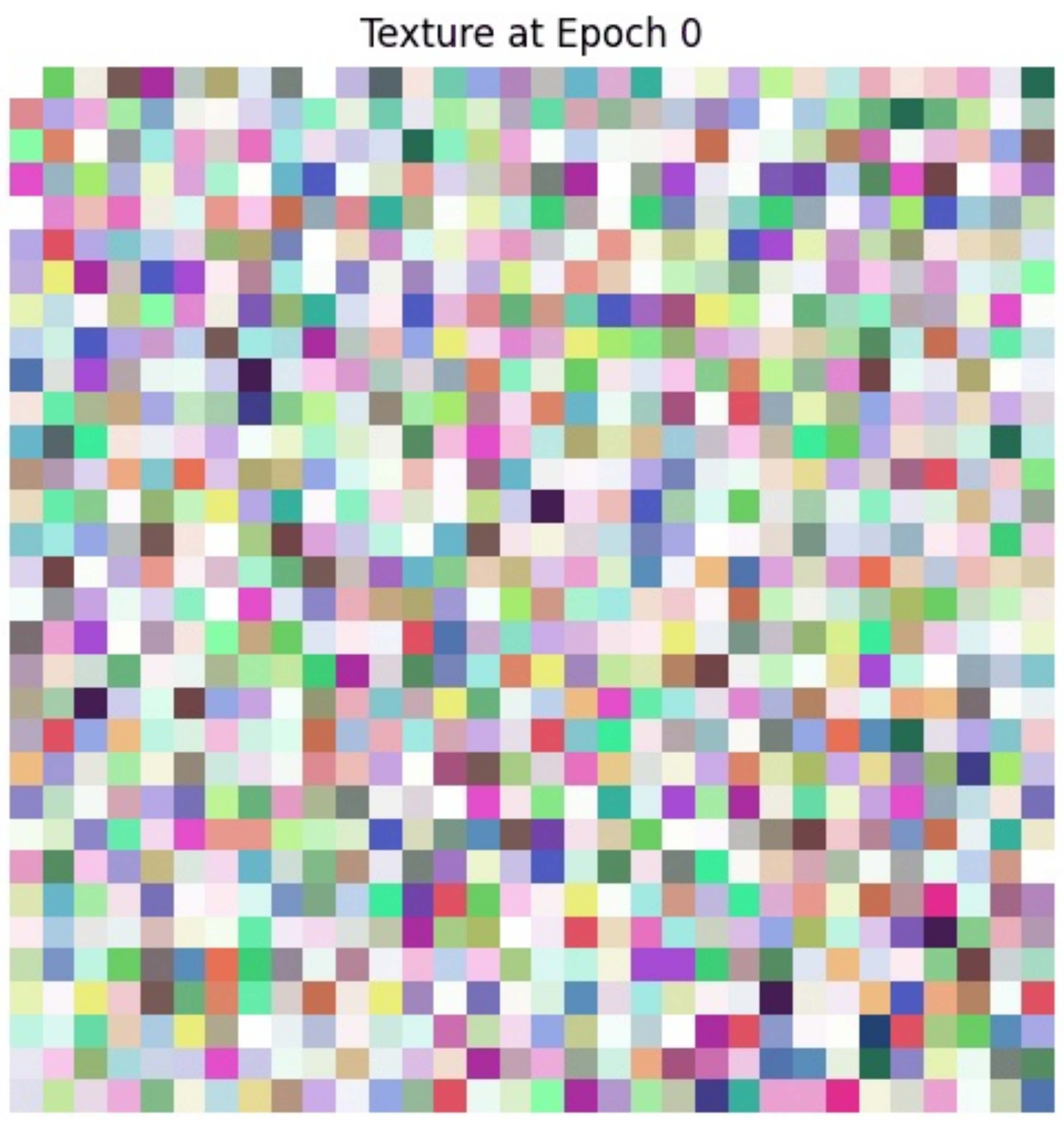
Example of calling foo() in a “forward pass”:

```
float a = 1.0;
float b = 2.4;
float result = foo(a, b);
float loss = 100 – result;

printf(“result is %f, loss was %f”, result, loss);
```

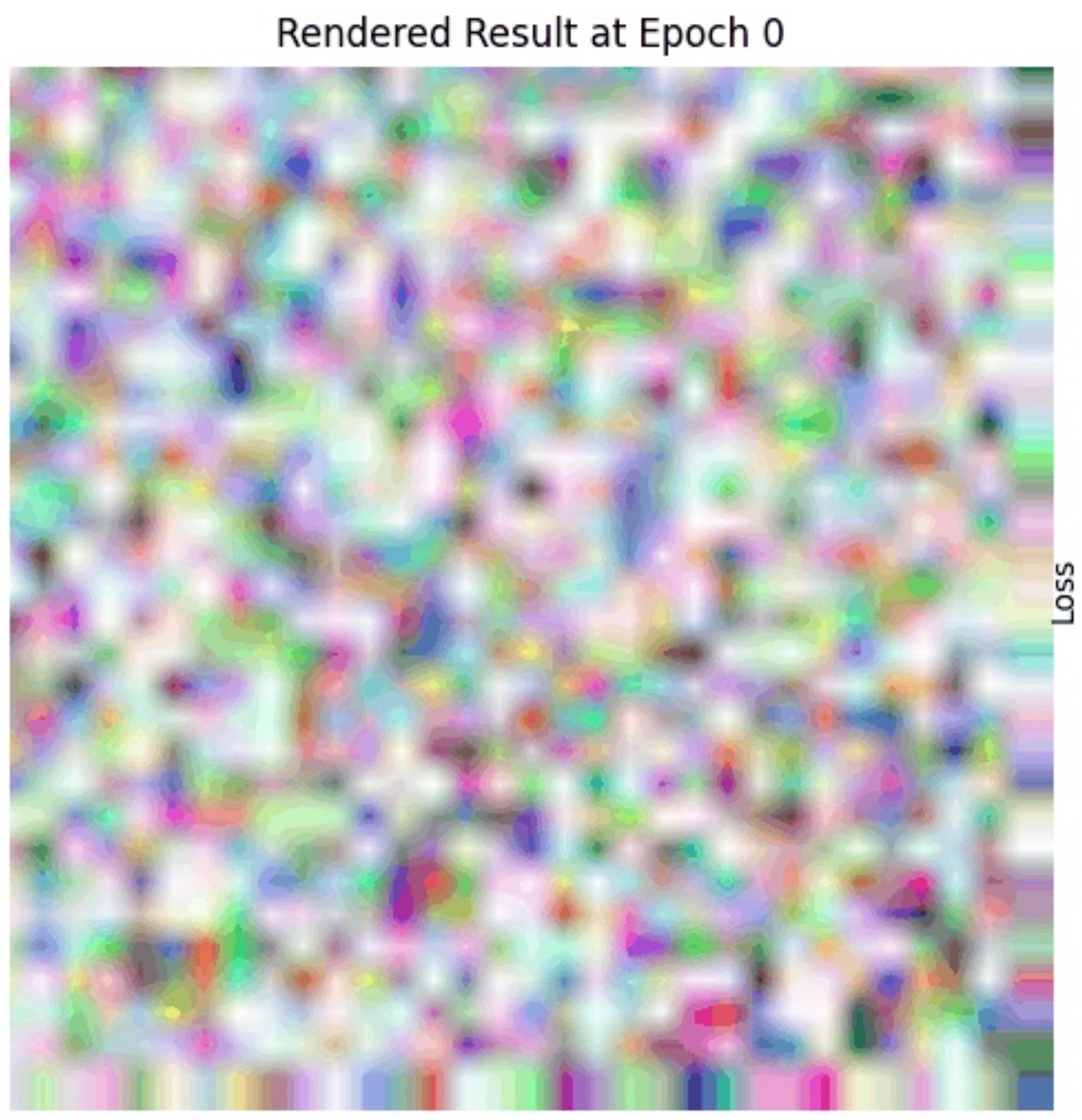
Example use of foo in a backwards pass to compute gradients:

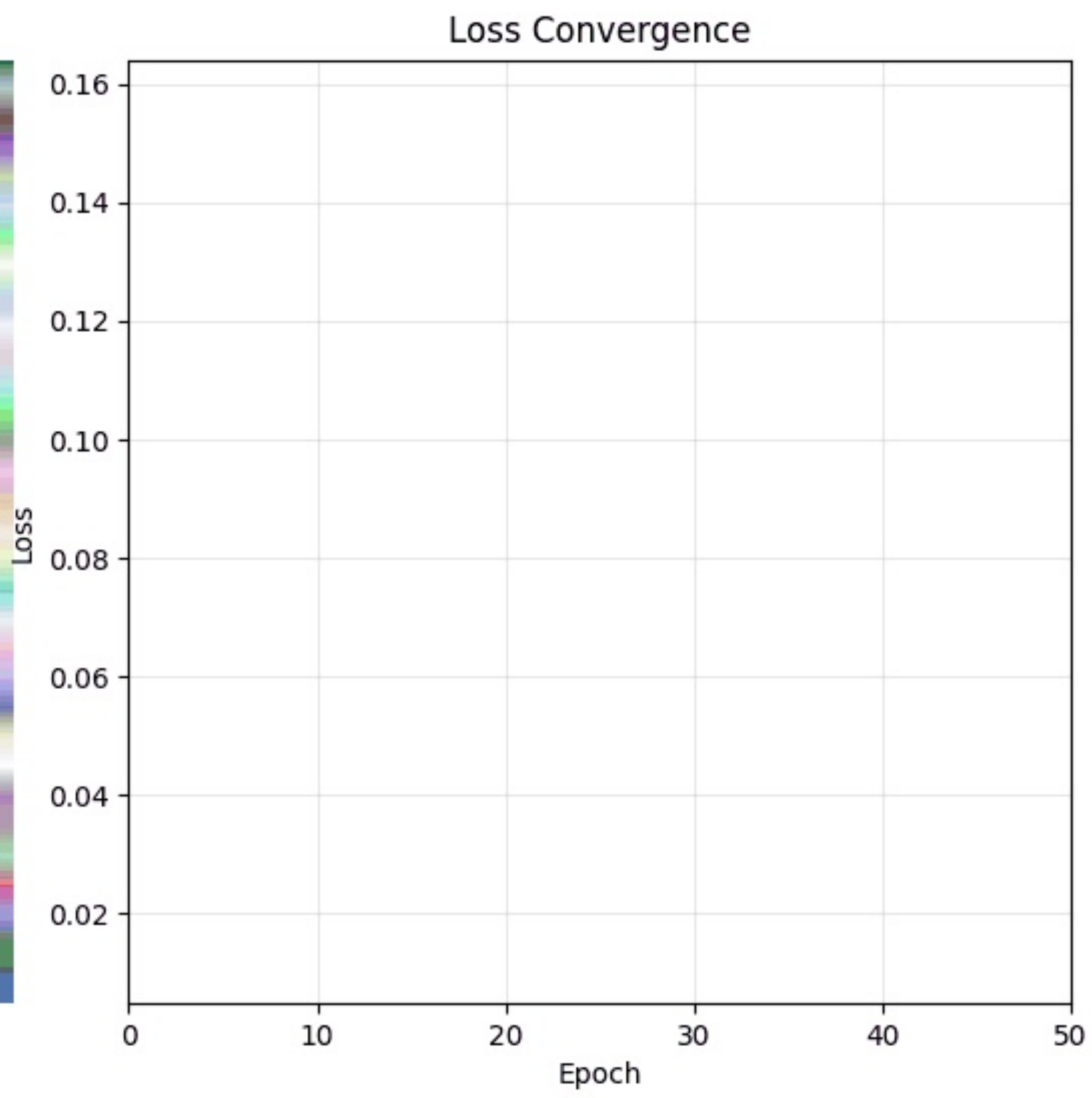
```
DifferentialPair<float> dp_a = diffPair(1.0);
DifferentialPair<float> dp_b = diffPair(2.4);

// Derivative of scalar L w.r.t the function foo’s output
float2 dL_dfoo = float2(1.0);

// compiler generates code for computing dFoo/da and dFoo/db
// and uses the input dL_dfoo to compute dL/da and dL/db
// dL/da=(dL/dfoo)(dfoo/da), dL/db=(dL/dfoo)(dfoo/db)
bwd_diff(foo)(dp_a, dp_b, dL_dfoo);

float dL_da = dp_a.d;
float dL_db = dp_b.d;

printf("If dL/dOutput = 1.0, then (dL/da, dL/db) at (1.0, 2.4) = (%f, %f)", dL_da, dL_db);
```

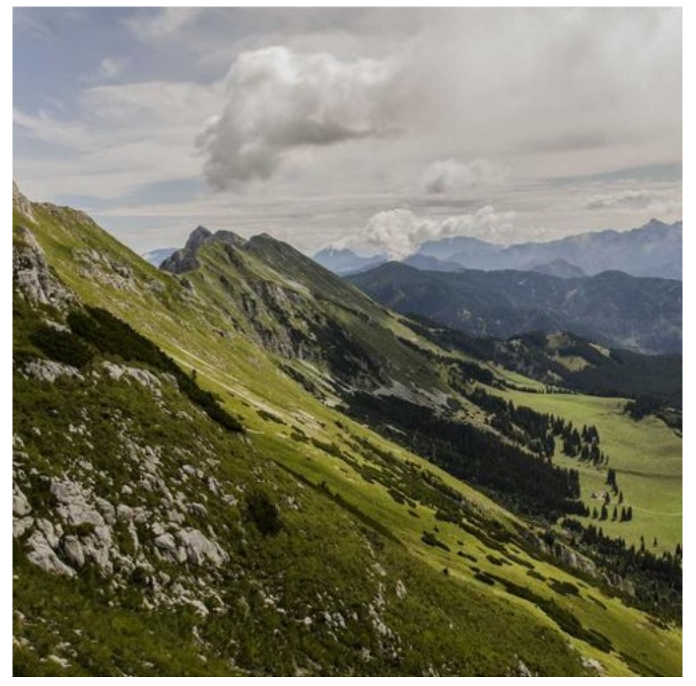
Example: optimization to recover texture values



$\mathbf{x}$

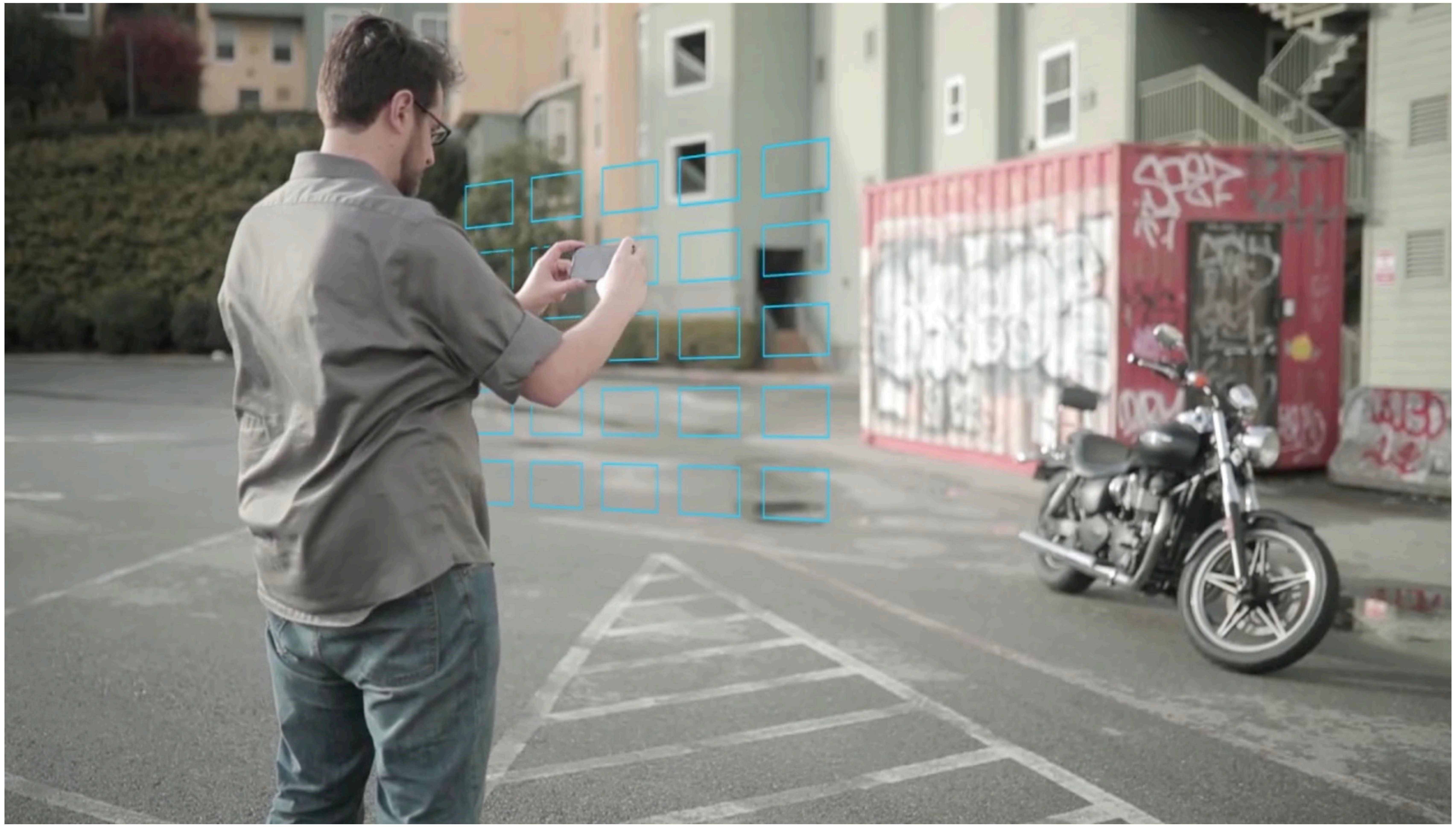




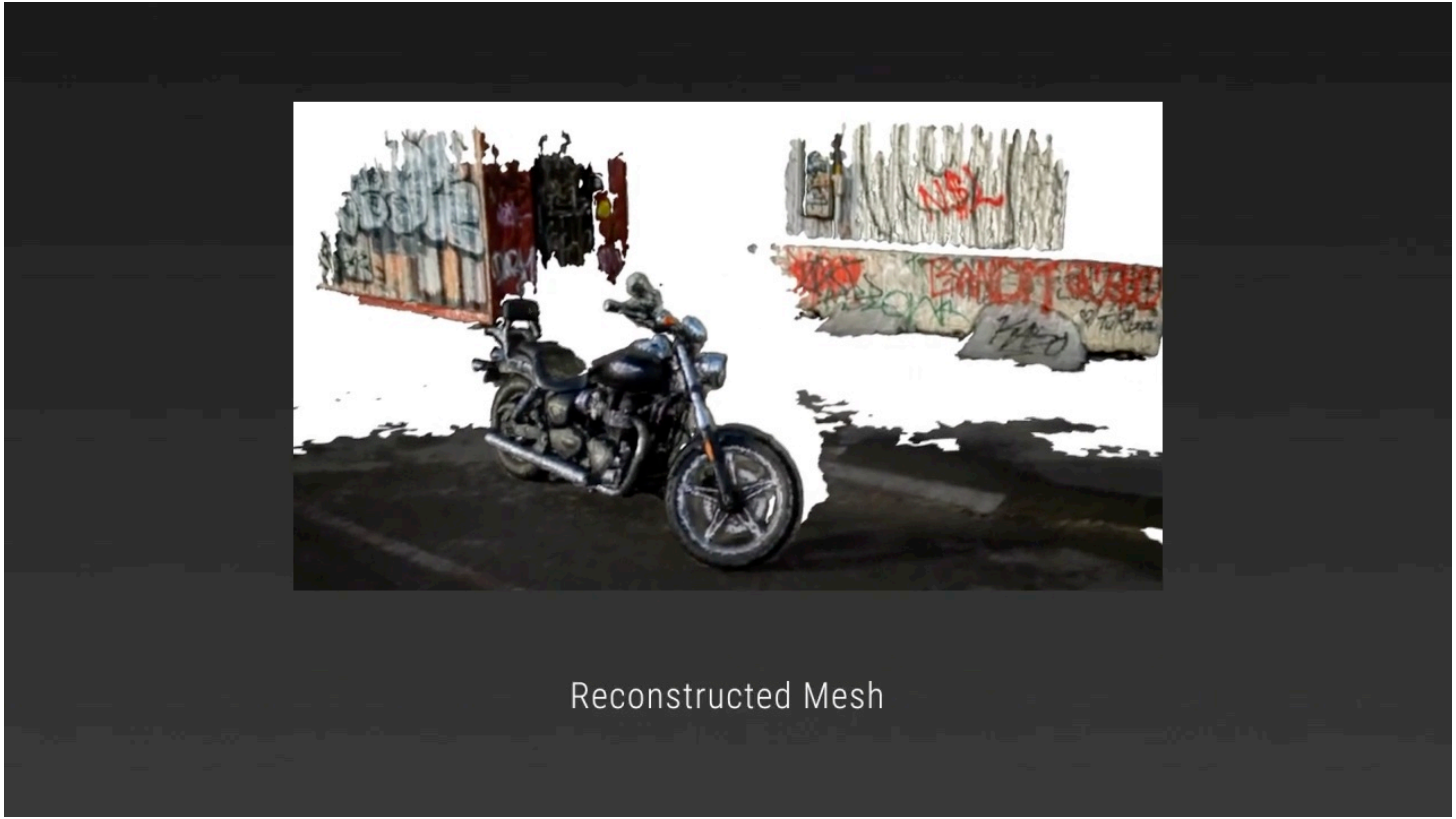


Target signal: $\text{scene}(u, v)$

Okay, finally back to our original problem of recovering scenes…



Estimating mesh geometry is tricky

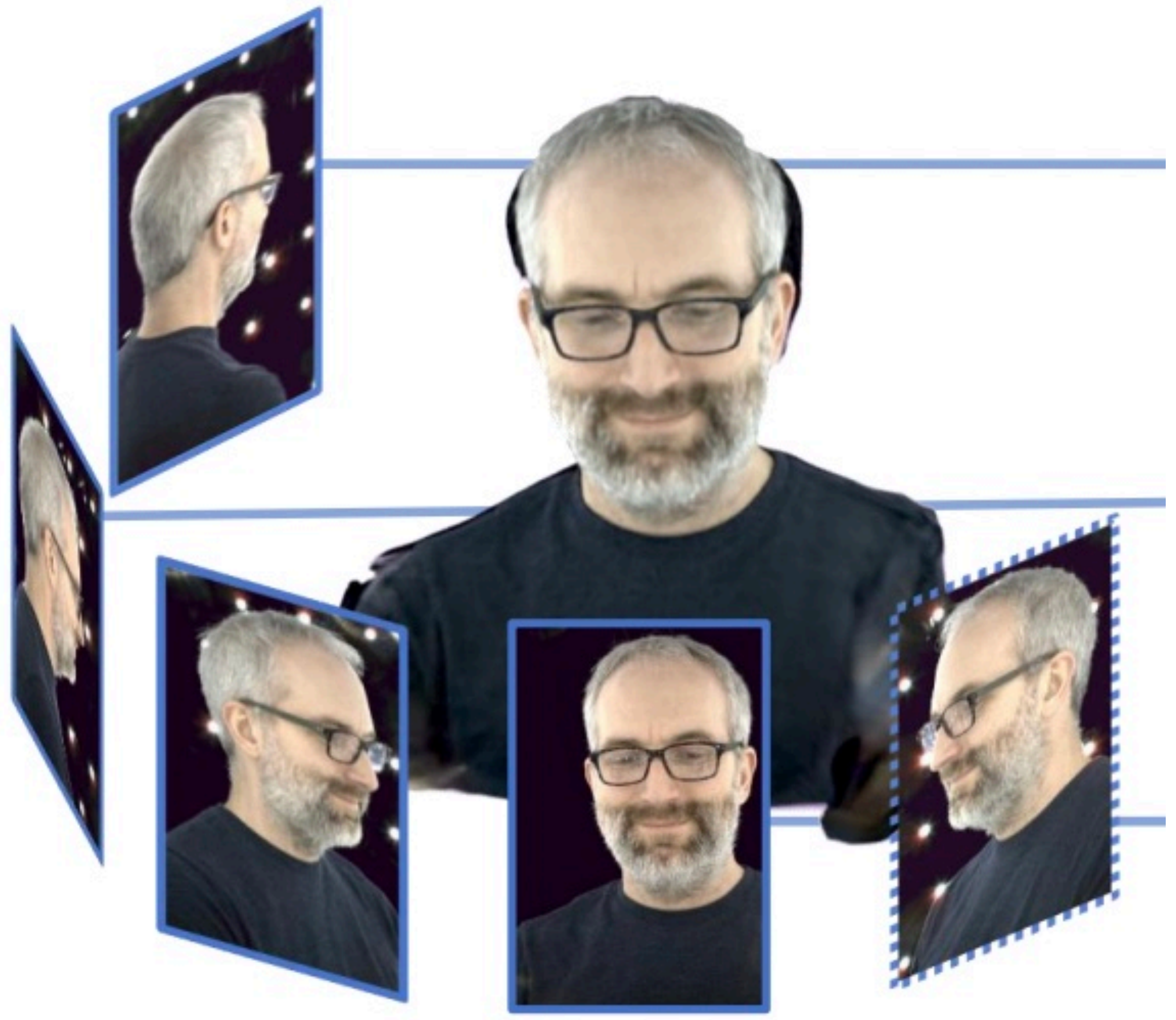


Credit: Mildenhall 2019

Renewed interest in volume rendering (circa 2018)

The idea: if the task we care about is scene reconstruction from photos (not efficient scene rendering)…

Let's move away from triangle-based representations. It is simpler (and more versatile when it's unclear what the surface geometry is anyway) to recover 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

Last time: simple volume rendering

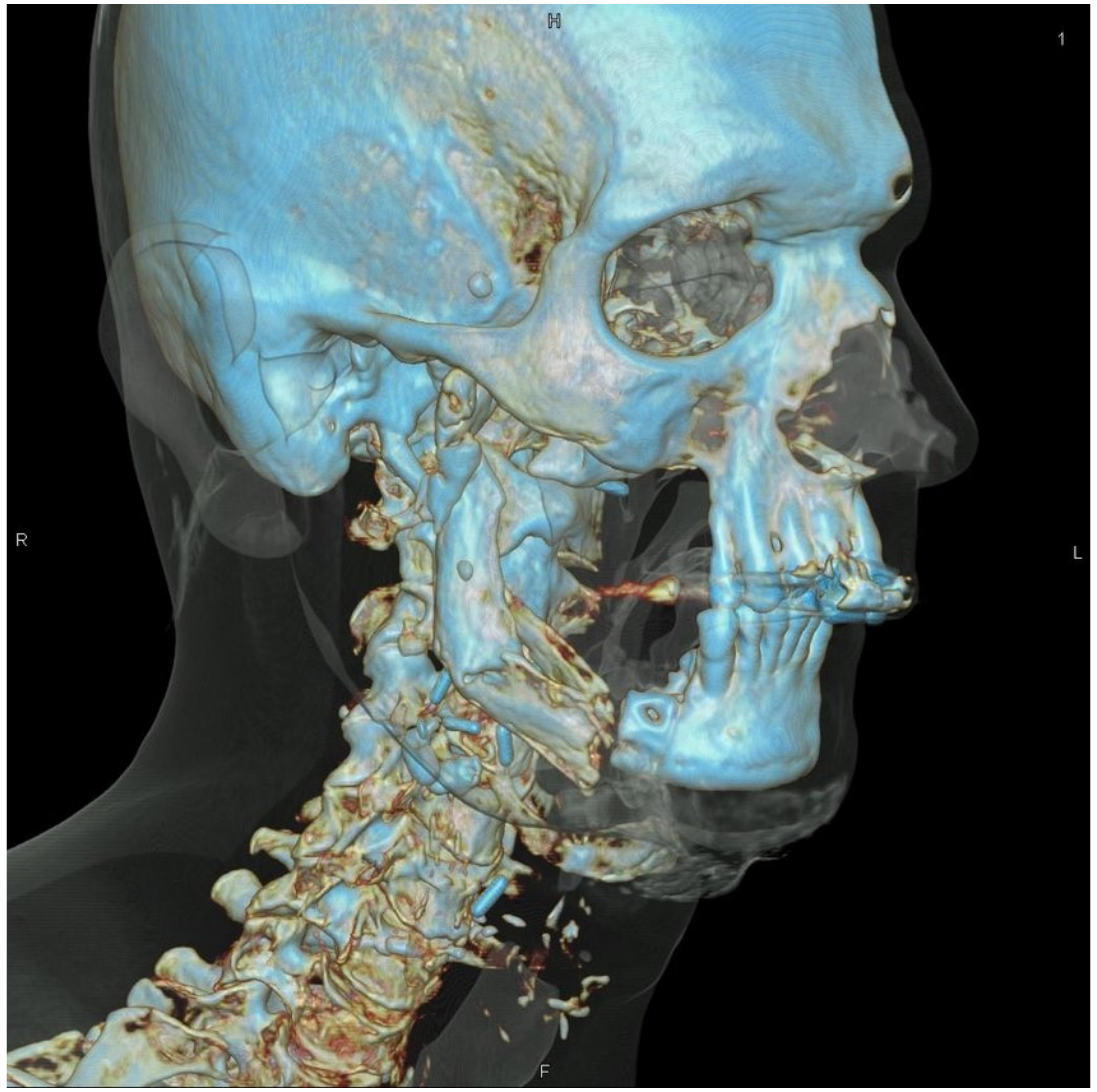
Consider representing a scene as a volume

Volume density and "reflectance" at all points in space

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

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

Think: radiance reflected off volume material at point p in direction ω. (Or radiance emitted by volume)



Volume rendered CT scan

Volume rendered scene

Credit: Taubmann et al. , Siemens Healthineers

Last time: rendering volumes

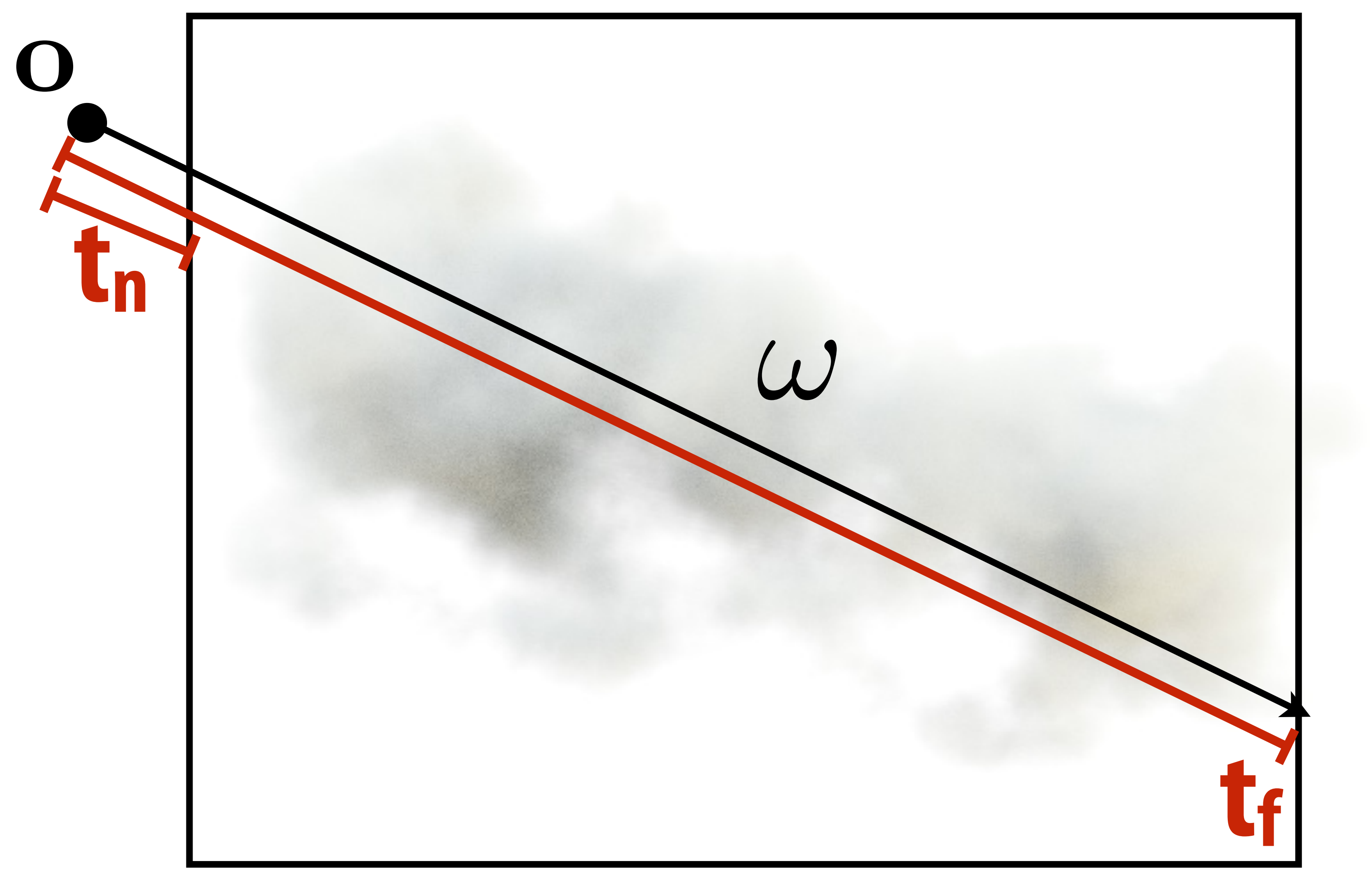
Given "camera ray" from point o in direction w….

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

And continuous volume with density and directional radiance.

$\sigma(\mathrm{p})$ ⟵ Volume density at all points in space.

$c(\mathrm{p}, \omega)$ ⟵ Radiance leaving volume point p in direction w (Due to light reflection off volume or emission)



Step through the volume to compute radiance along the ray.

Attenuation of radiance along r between r(t) and the ray original due to light being absorbed or scattered by the volume

Color, opacity of the volume at the current point (More precisely: radiance along -w at point r(t))

$$L(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\boxed{\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), -\omega))}\mathrm{d}t \quad \text{where } T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right)$$

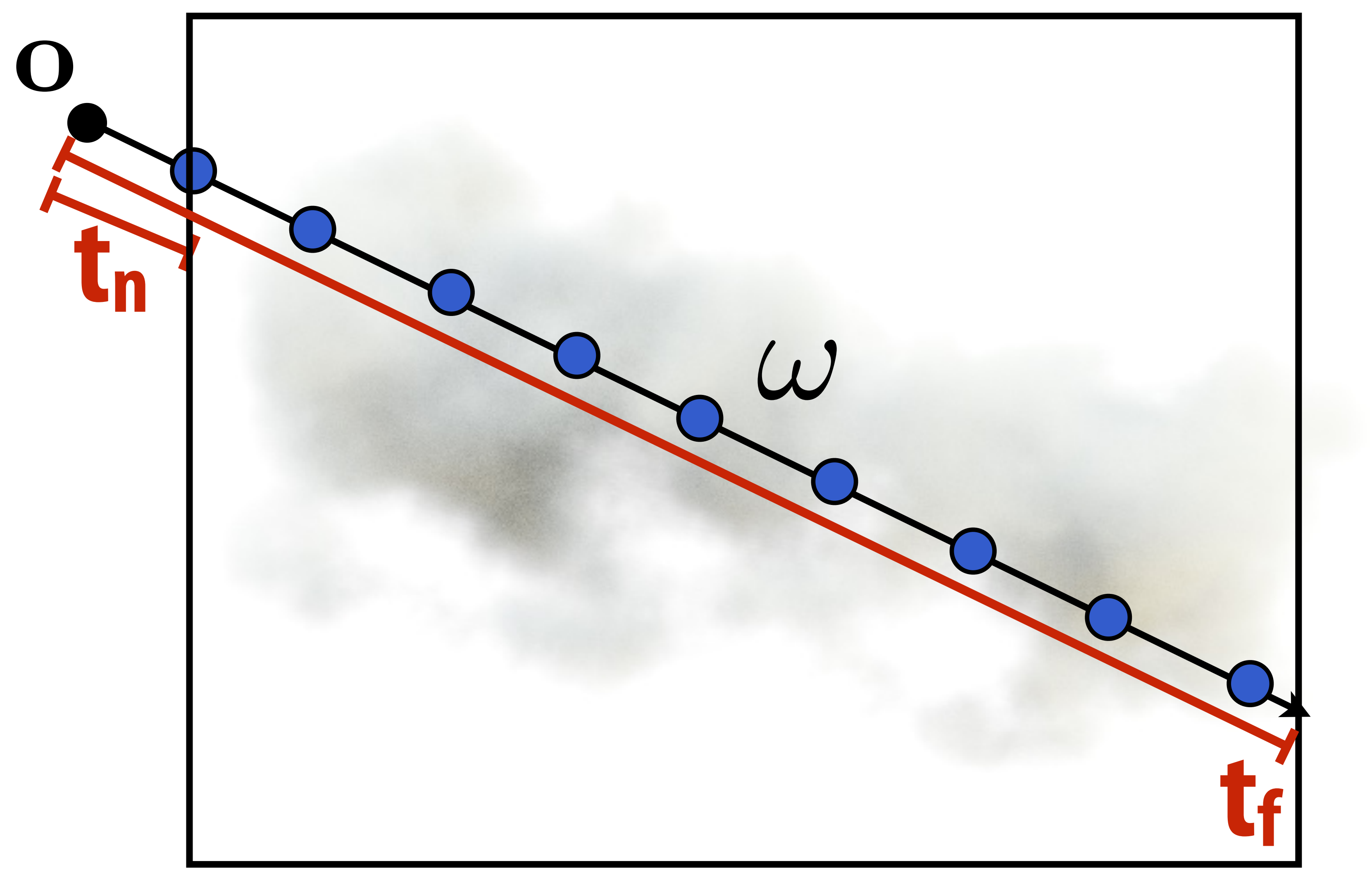
Last time: rendering volumes

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

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

And volume with density and directional radiance

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



Step through the volume and accumulate radiance along the ray:

```
L = 0.0                 // total radiance accumulated
thickness = 0.0         // total density traversed
num_steps = (t_f – t_n) / step_size
for i=0 to num_steps:
  p = o + (t_n + i * step_size) * w   // current point along ray
  density = sample_density(p)         // tri-lerp
  refl = sample_color(p, –w)          // tri-lerp
  thickness += density * step_size
  transmittance = exp(–thickness)

  // accumulate radiance contributed from current point
  // (accounting for attenuation)
  L += transmittance * density * refl
```

Computing gradients with respect to volume parameters

Recovering a volume that yields acquired images

Given a set of images of a subject with known camera positions…

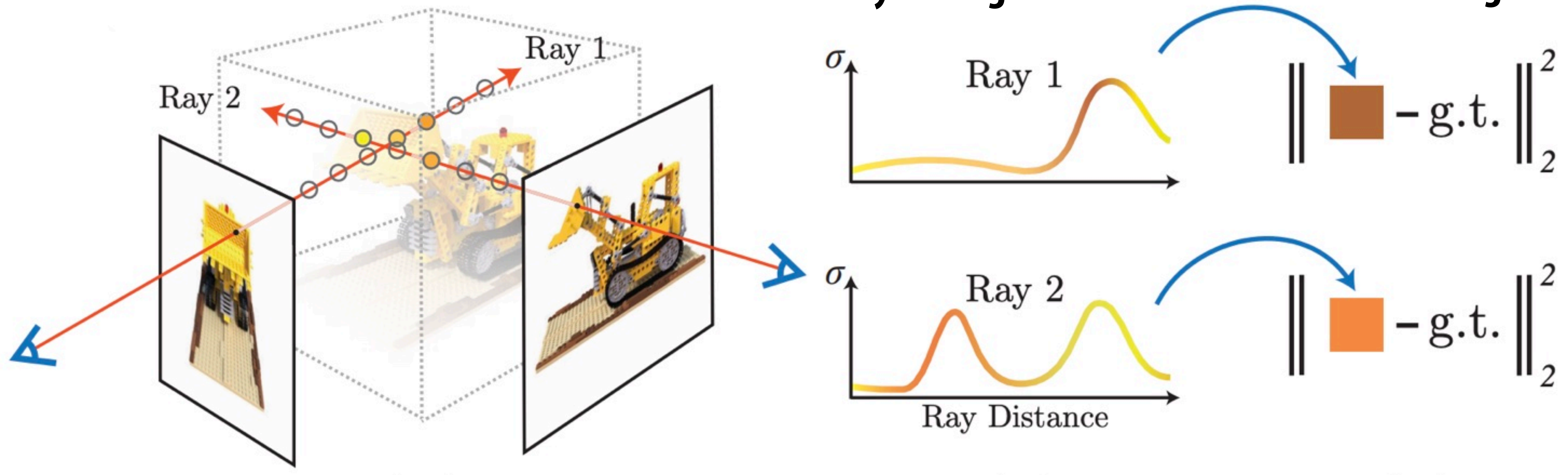
And a volume renderer that can render an image given a camera position and a volume

Recover the parameters of a 3D volume: *

$$\sigma(i,j,k),\ \mathbf{c}(i,j,k)$$

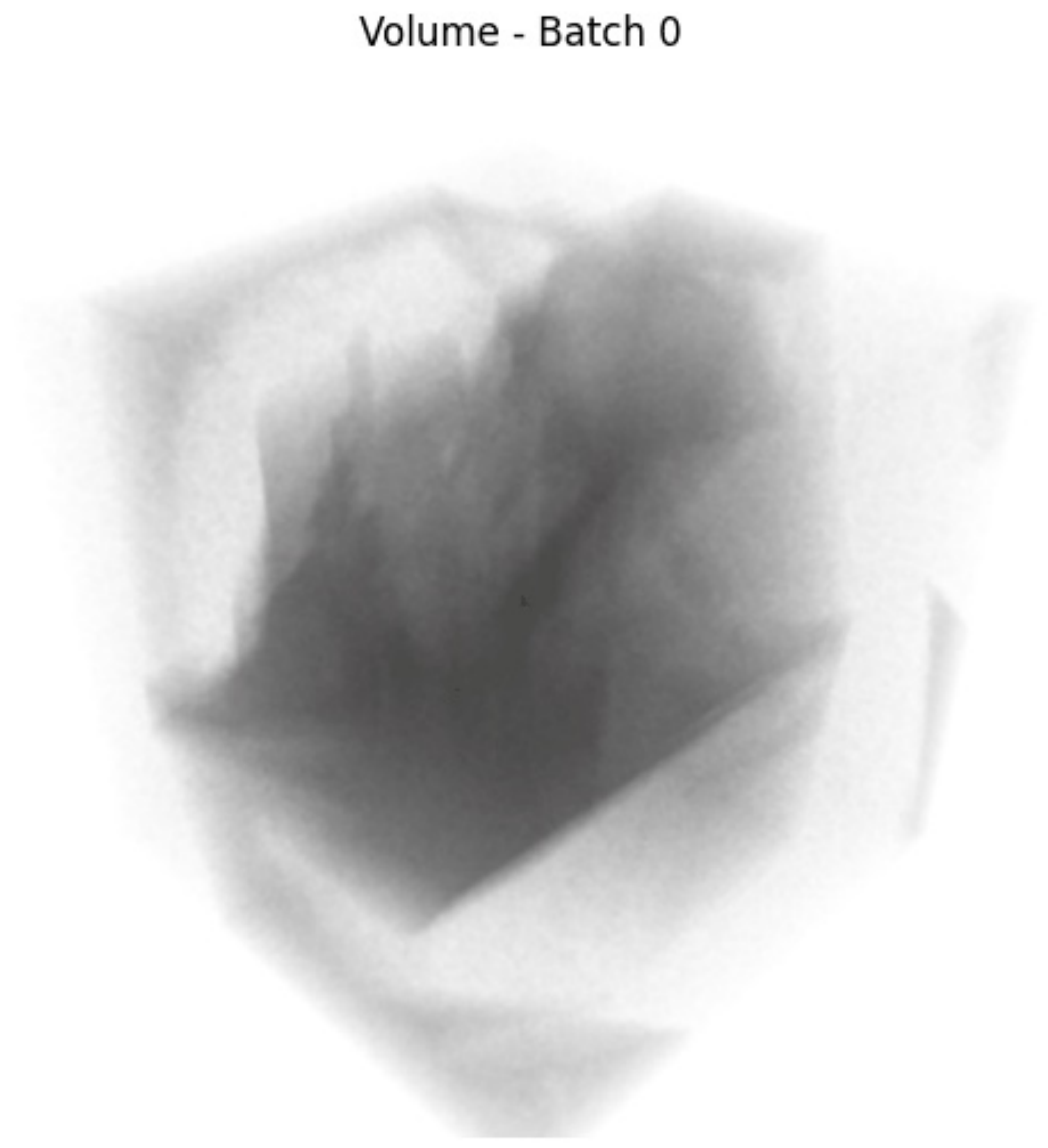
Compute radiance along ray through volume

Compare to actual image

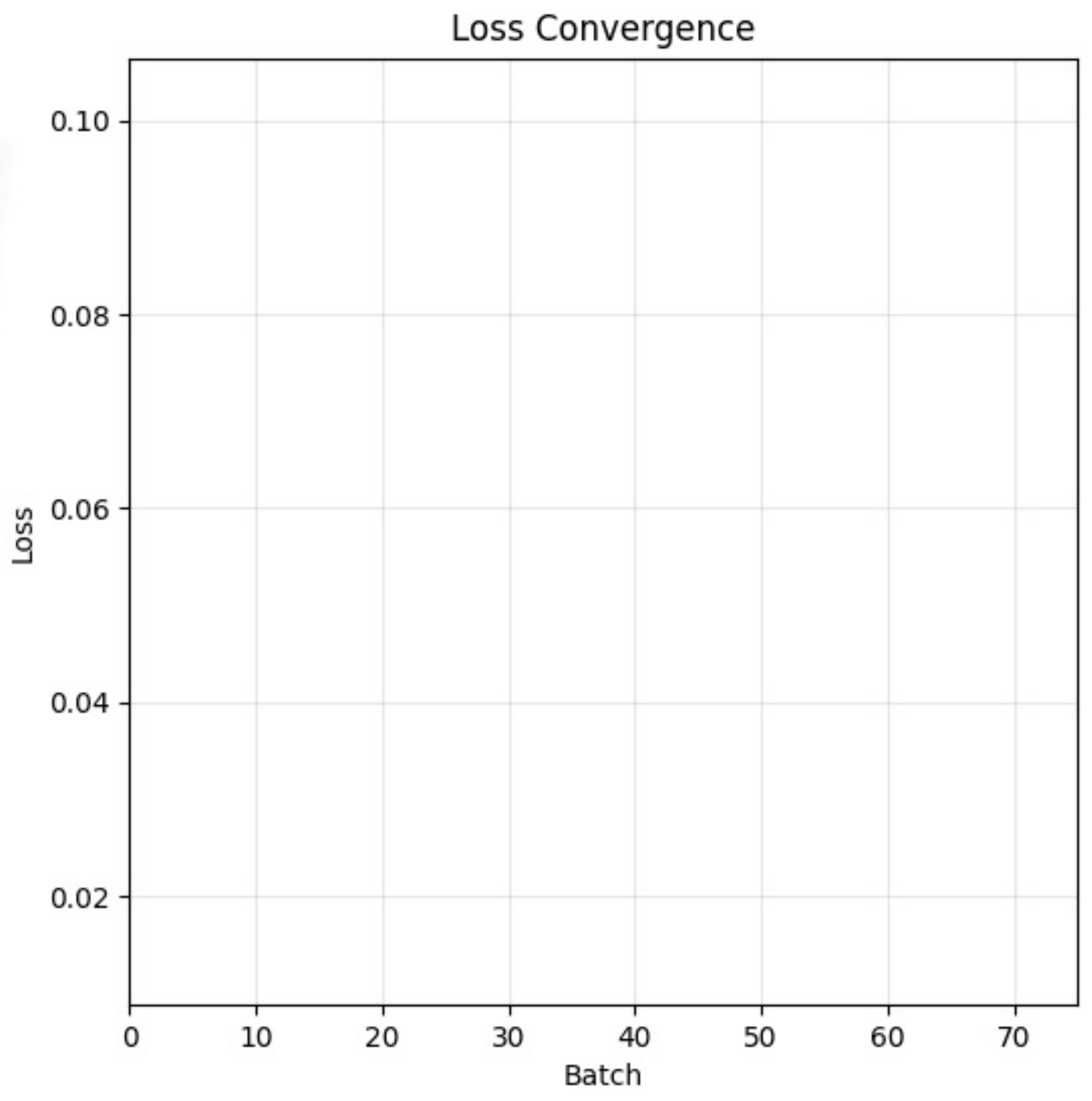

Ray 1
Ray 2
σ
Ray 1
σ
Ray 2
Ray Distance
$\left\| \blacksquare - \text{g.t.} \right\|_2^2$
$\left\| \blacksquare - \text{g.t.} \right\|_2^2$

*** In this simple example, assume that** $\mathbf{c}(\mathrm{p},\omega) = \mathbf{c}(\mathrm{p})$ **(No directional dependence of radiance reflected off volume)**

Recovering density and color of volume



View from one Dataset Camera - Batch 0



Visualization of gradient of loss

- Volume initialized to low density value everywhere

- This is a visualization of the gradient of the loss

- Positive gradient occurs in what should be empty space
  - Stepping in direction of negative gradient will reduce volume density.

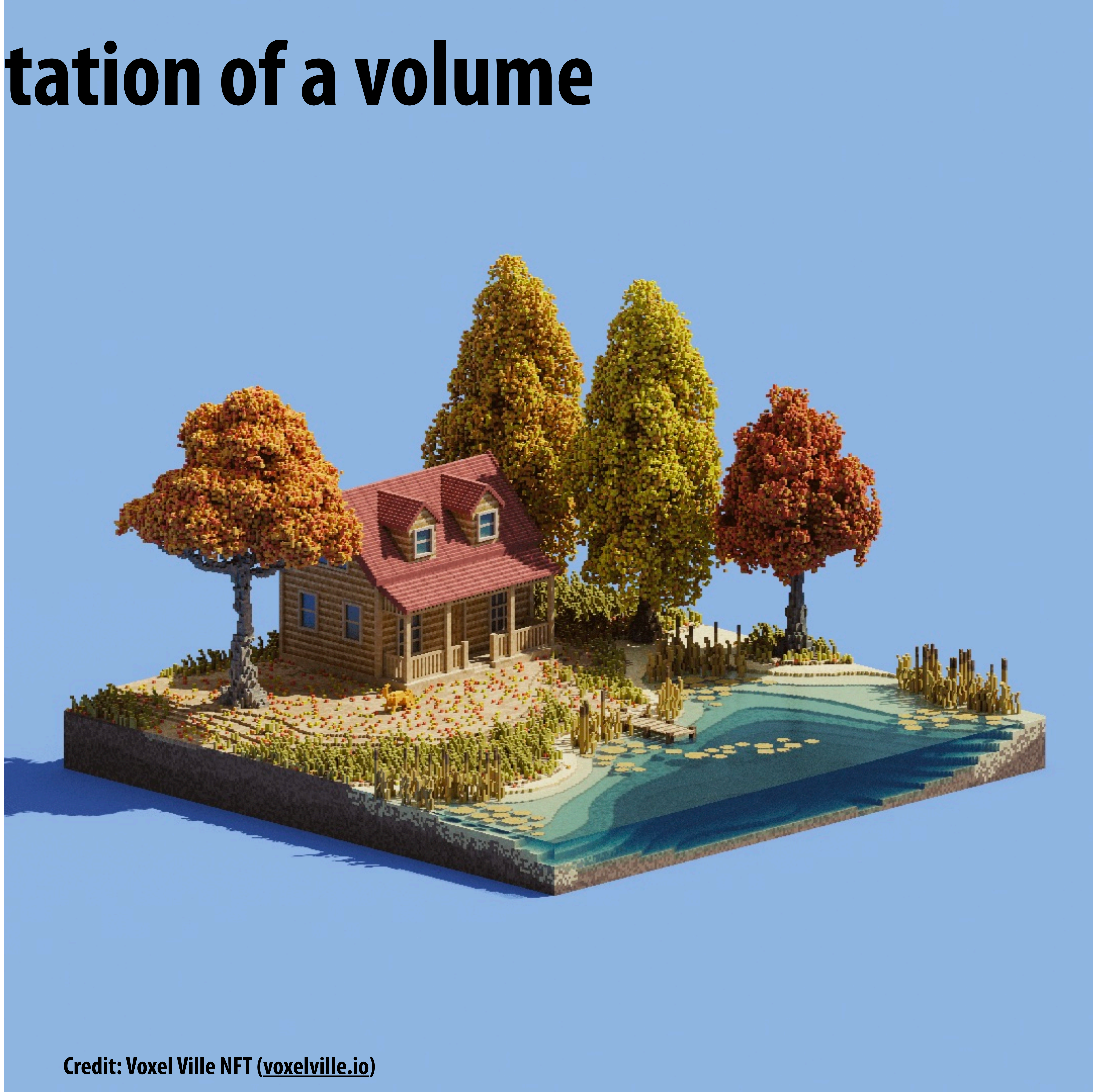
- Negative gradient occurs in the view frustum where geometry should be present. Density should increase there

Regular 3D grid representation of a volume

- Dense 3D grid
  - volume[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 radiance



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)

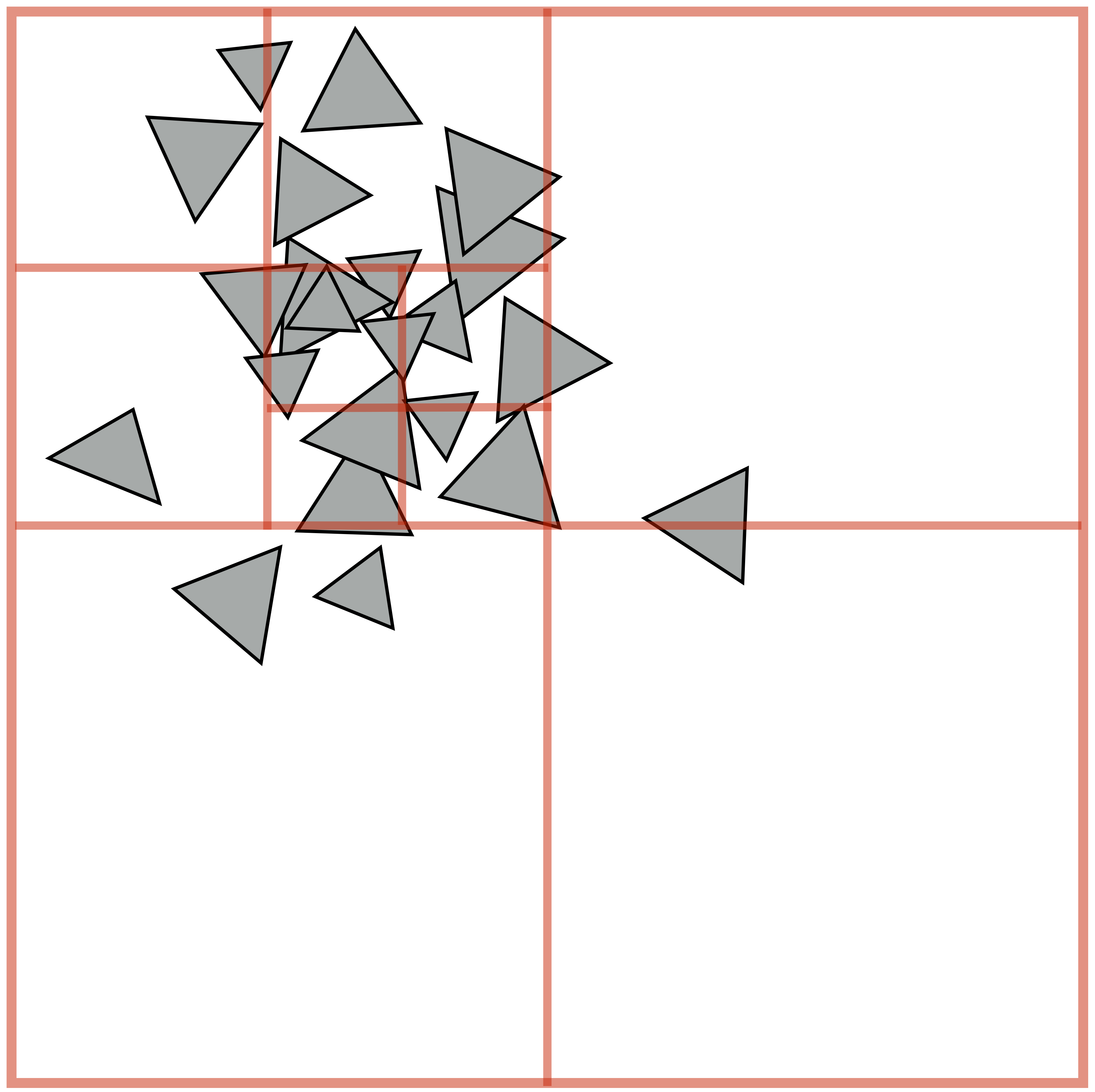
Recall quad-tree / octree

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

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

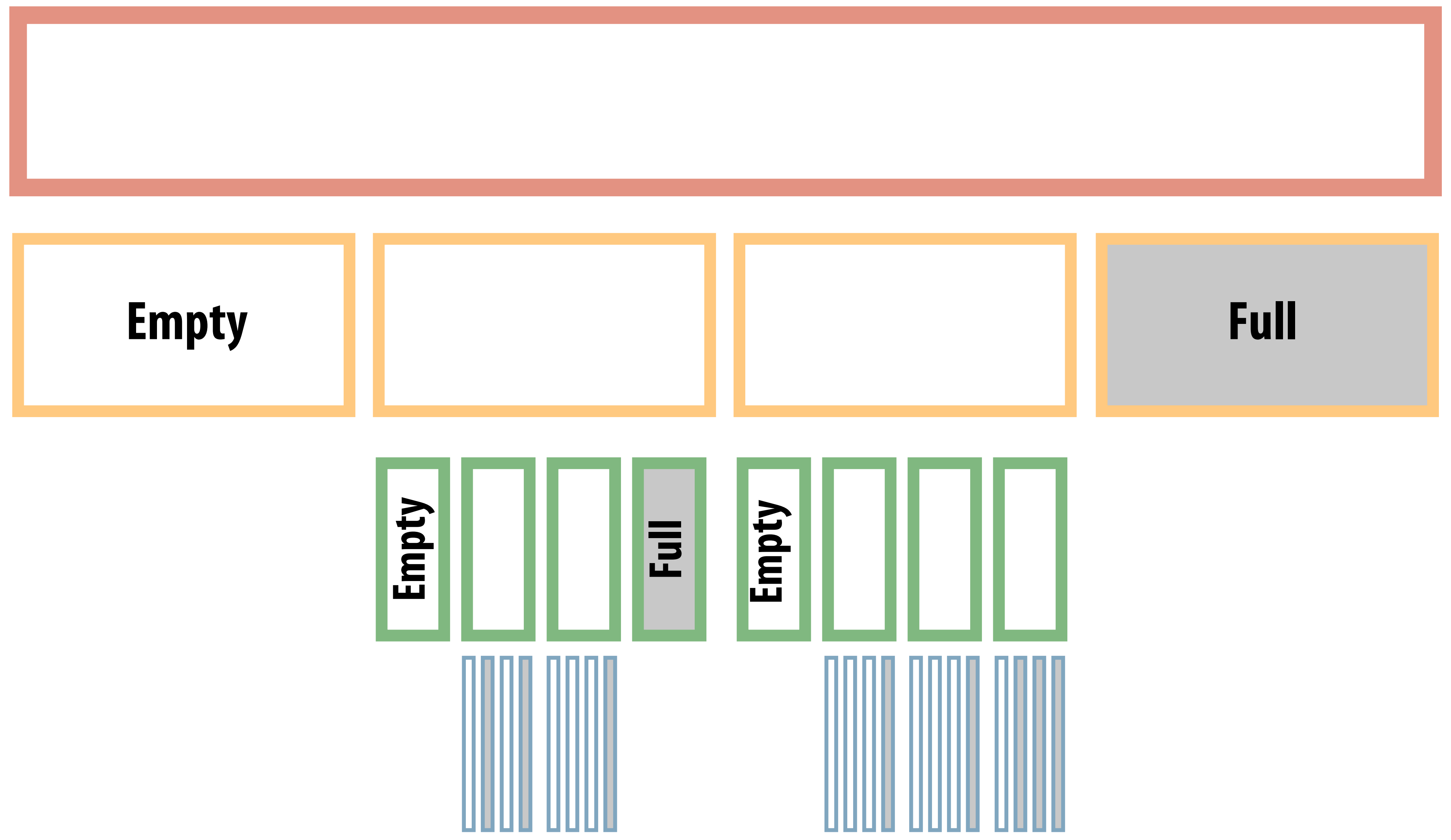
Like uniform grid: easy to build (don't have to choose partition planes)

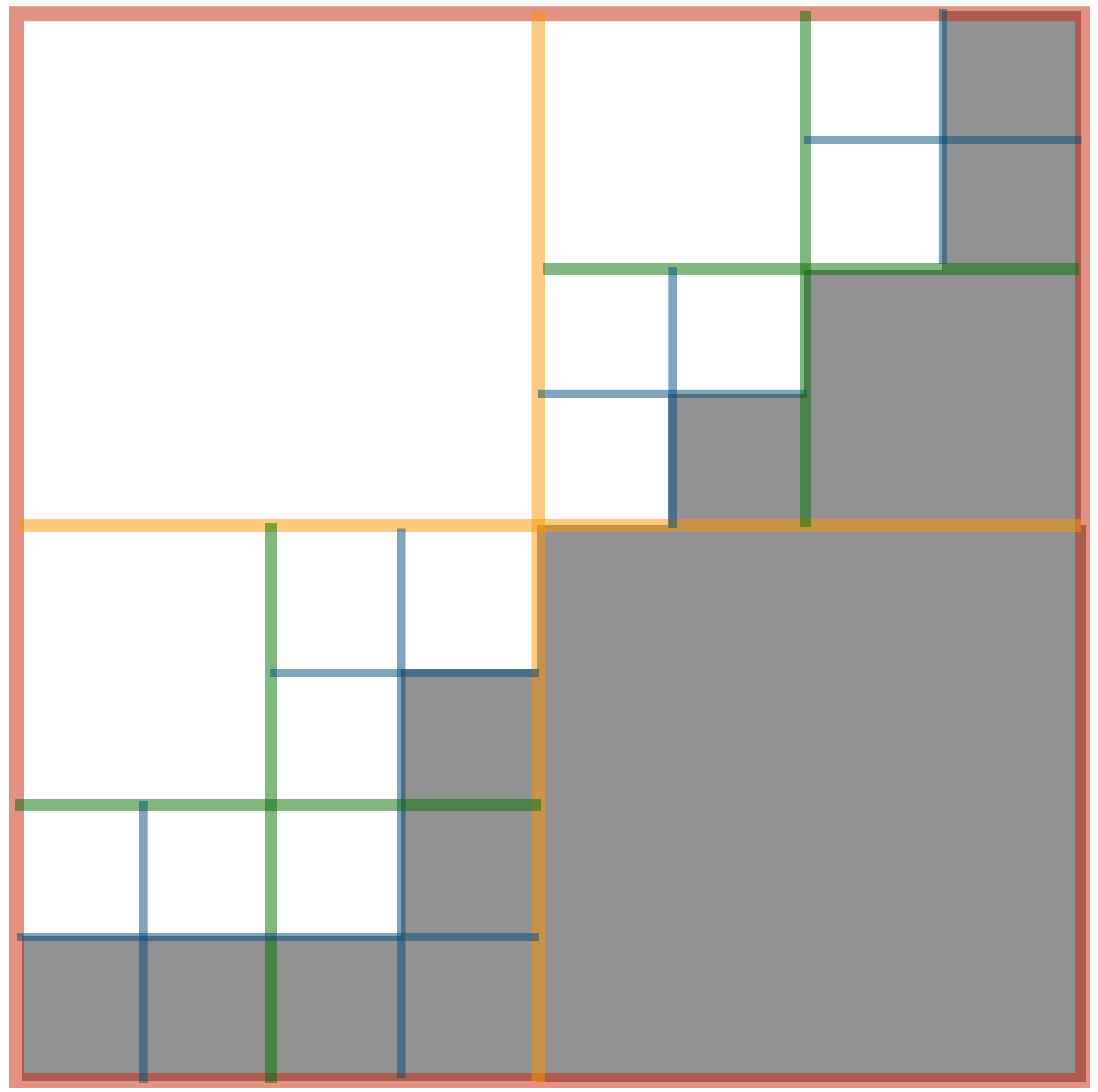
Has greater ability to adapt to location of scene geometry than uniform grid.



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

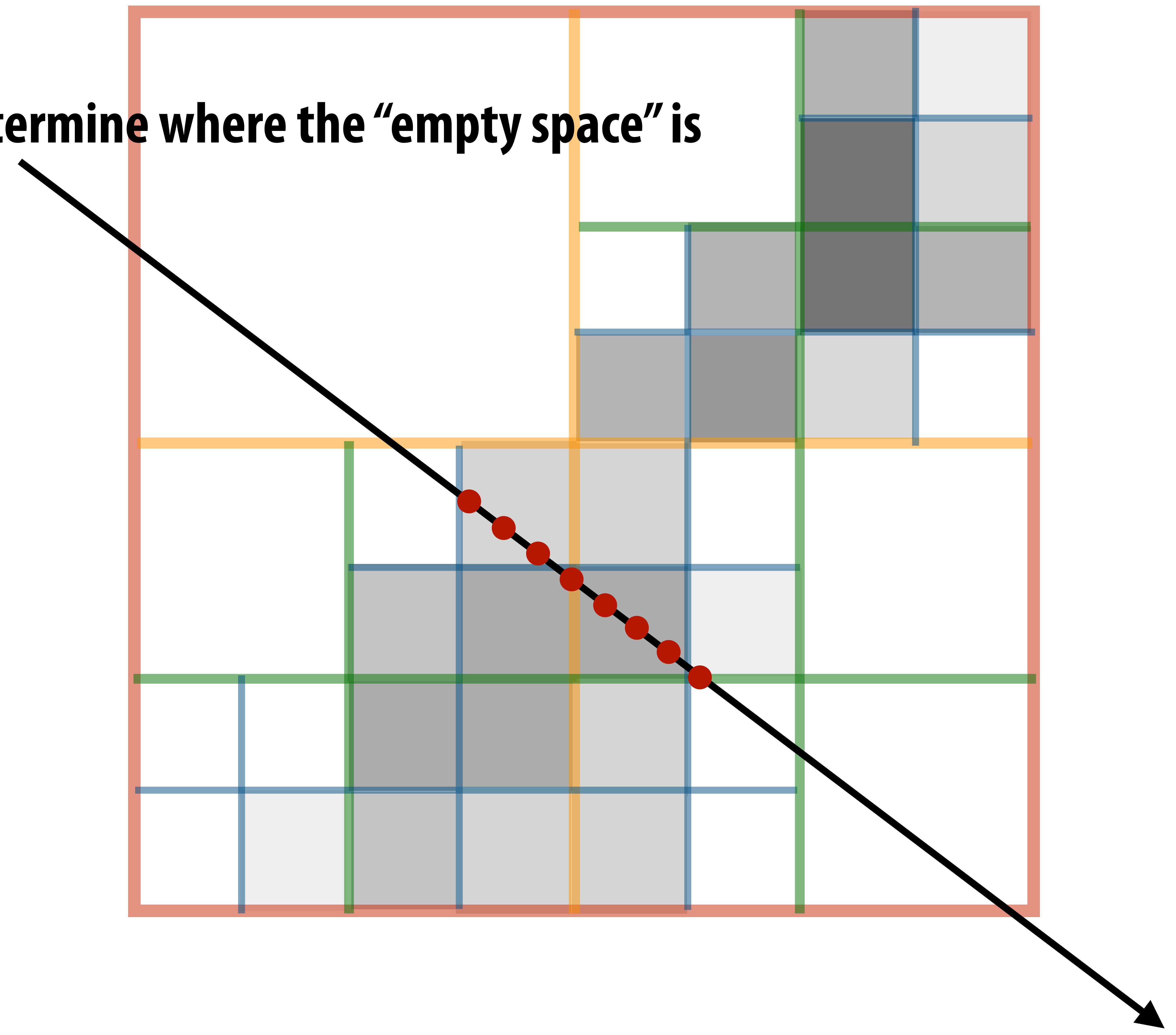
Interior nodes with no children → same "value" for all children in subtree

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

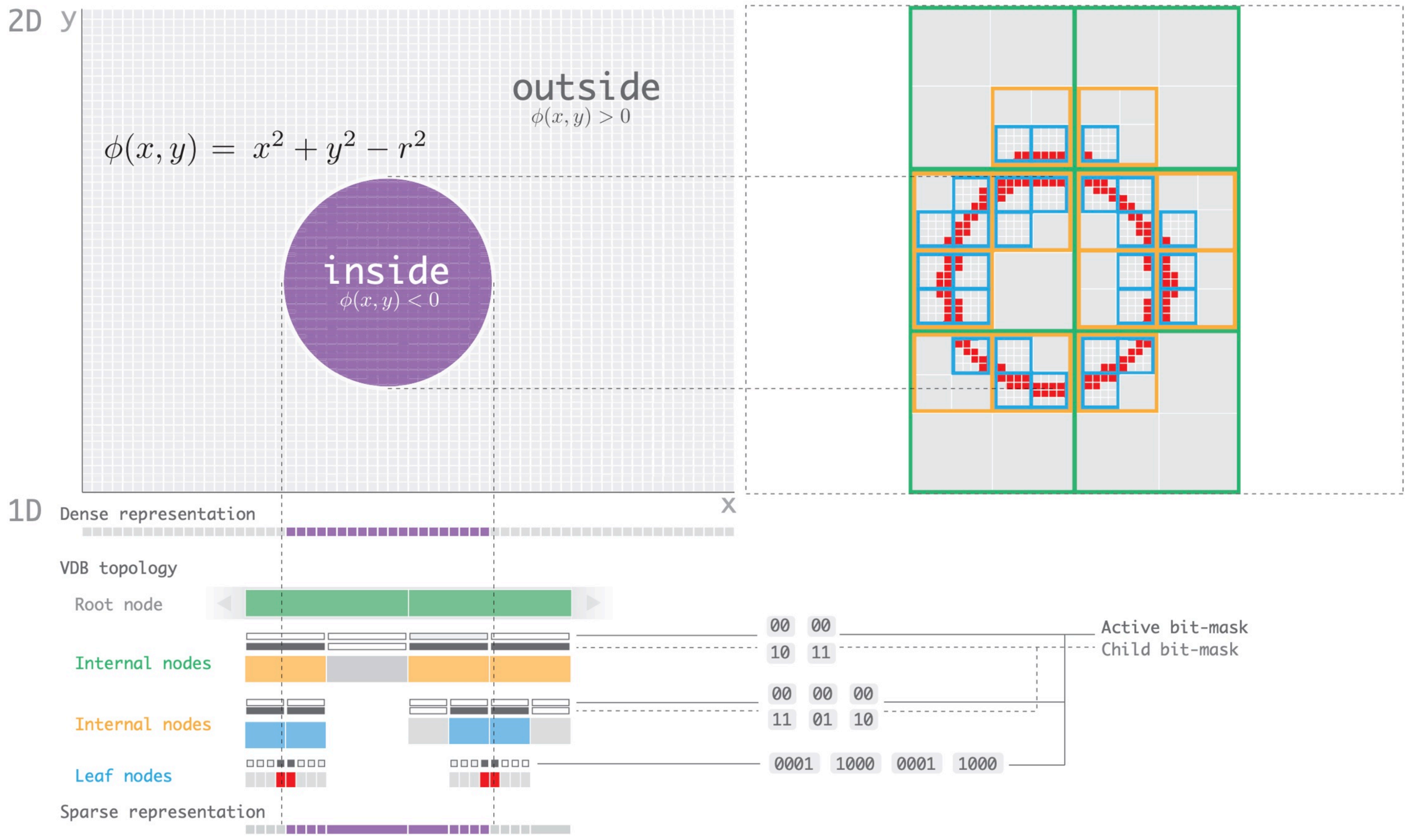
Ray can now "skip" through empty space

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

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

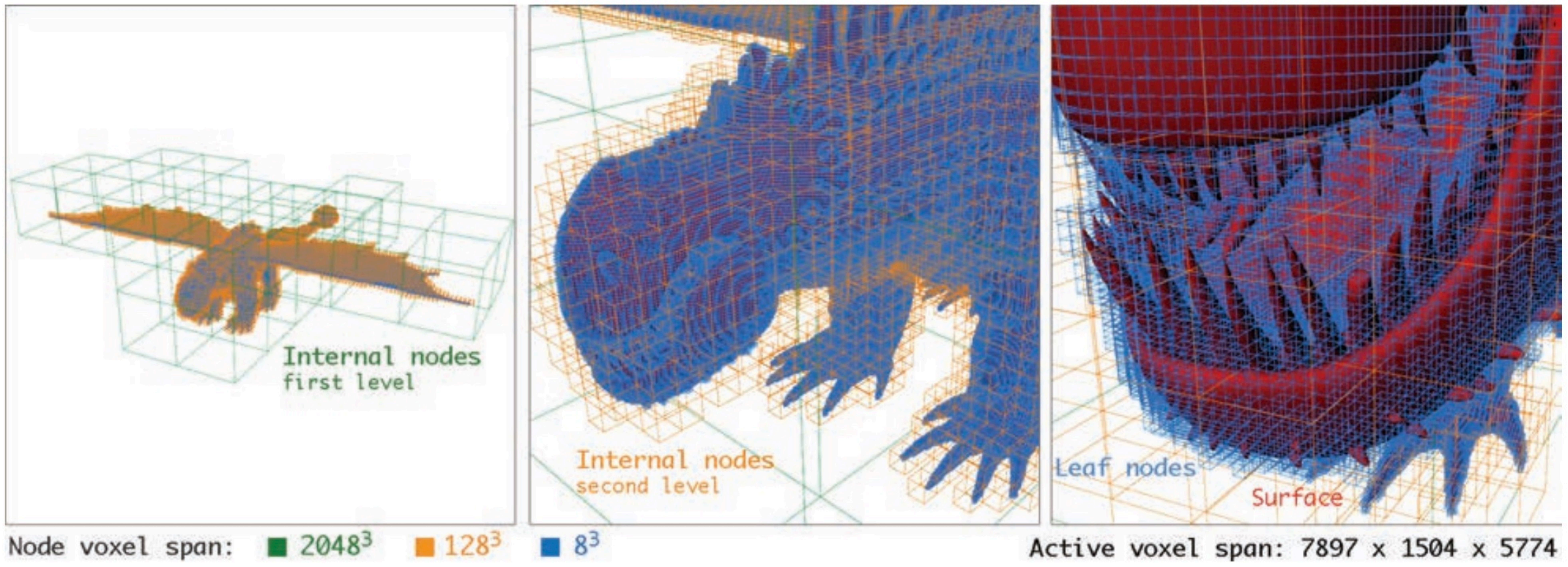

Internal nodes
first level
Internal nodes
second level
Leaf nodes
Surface
Node voxel span: 🟩 2048³ 🟧 128³ 🟦 8³
Active voxel span: 7897 x 1504 x 5774

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

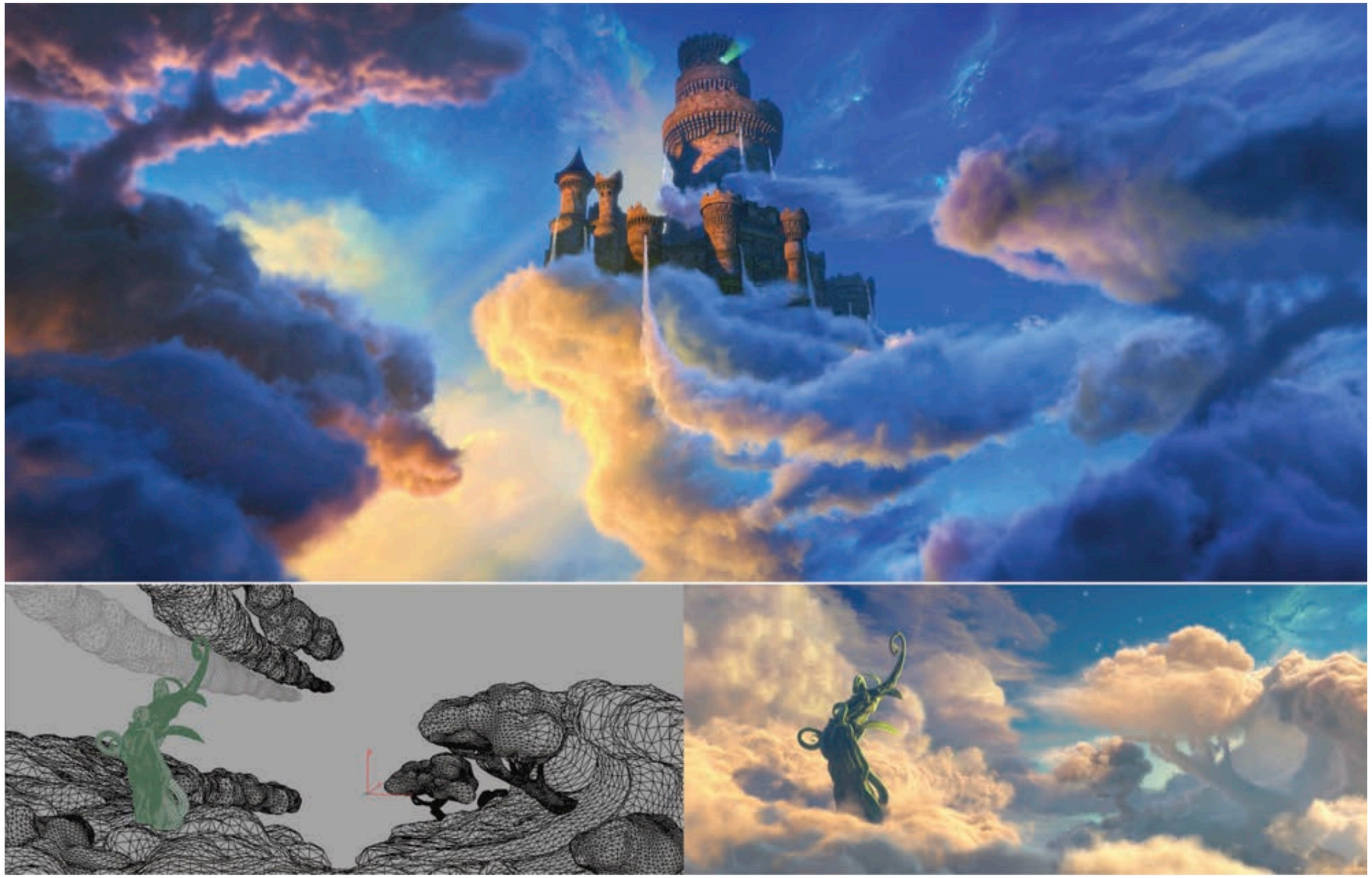


Fig. 1. Top: Shot from the animated feature *Puss in Boots*, showing high-resolution animated clouds generated using VDB [Miller et al. 2012]. Left: The clouds are initially modelled as polygonal surfaces, then scan-converted into narrow-band level sets, after which procedural noise is applied to create 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*.

Can you think of challenges of using sparse structures when attempting to recover a 3D scene representation?

Recurring theme in this course:
Choose the right representation for the task at hand

Now the task is recovering a continuous color and opacity field that represents a complex 3D scene

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

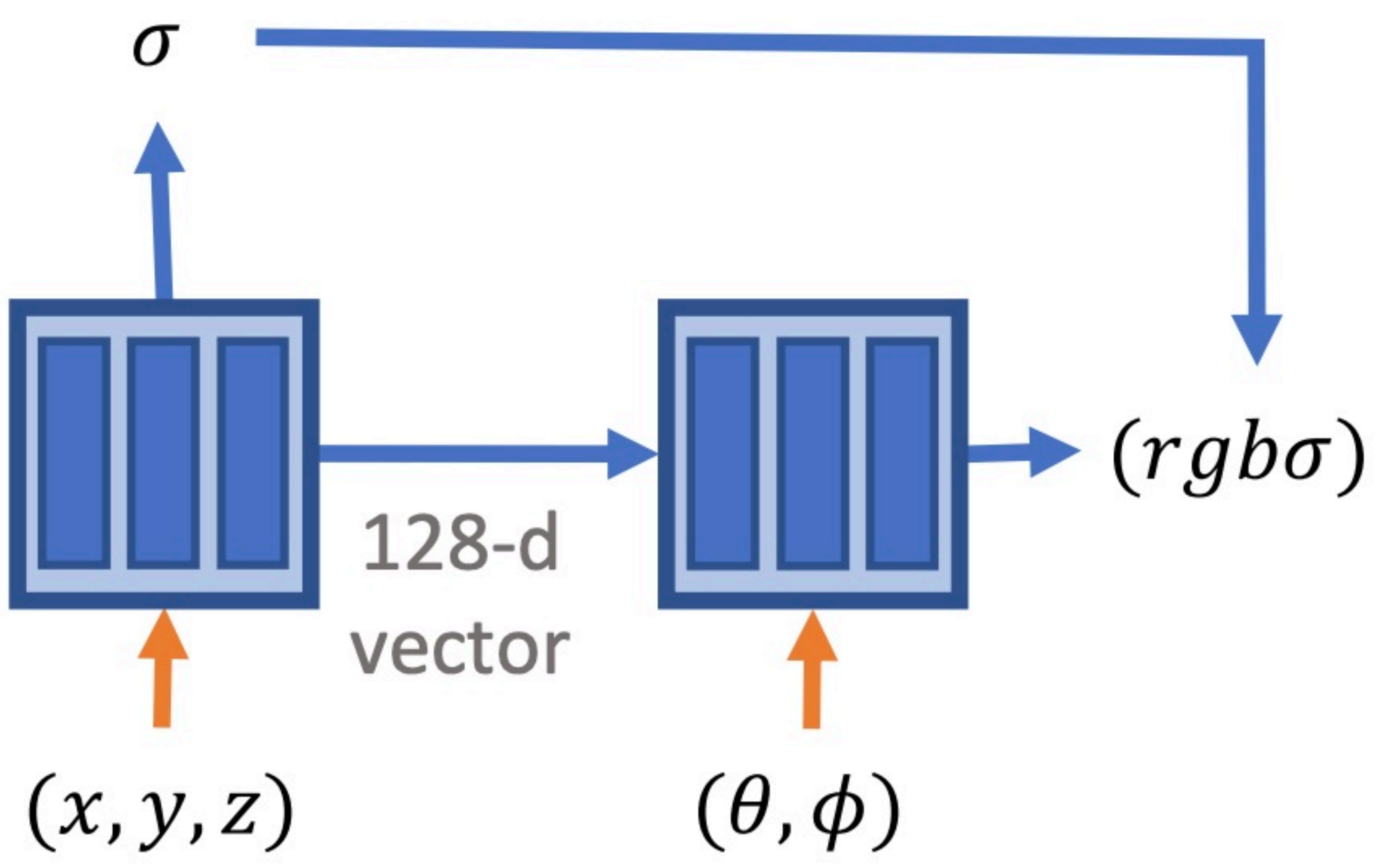
And that recovery process is optimization via gradient descent.

Learning (compressed) representations

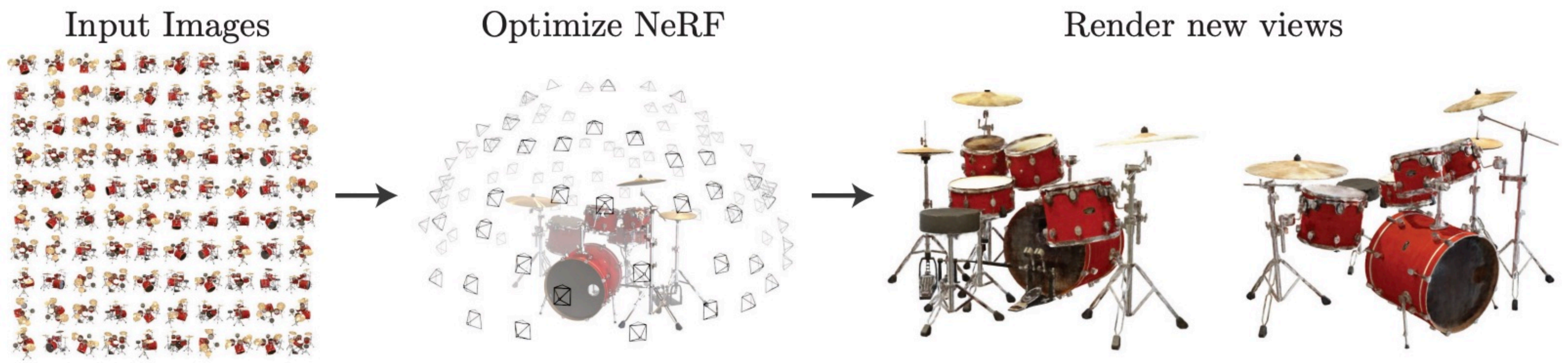
Rather than store an entire dense volume, let's just learn an approximation to the continuous function that matches observations from different viewpoints?

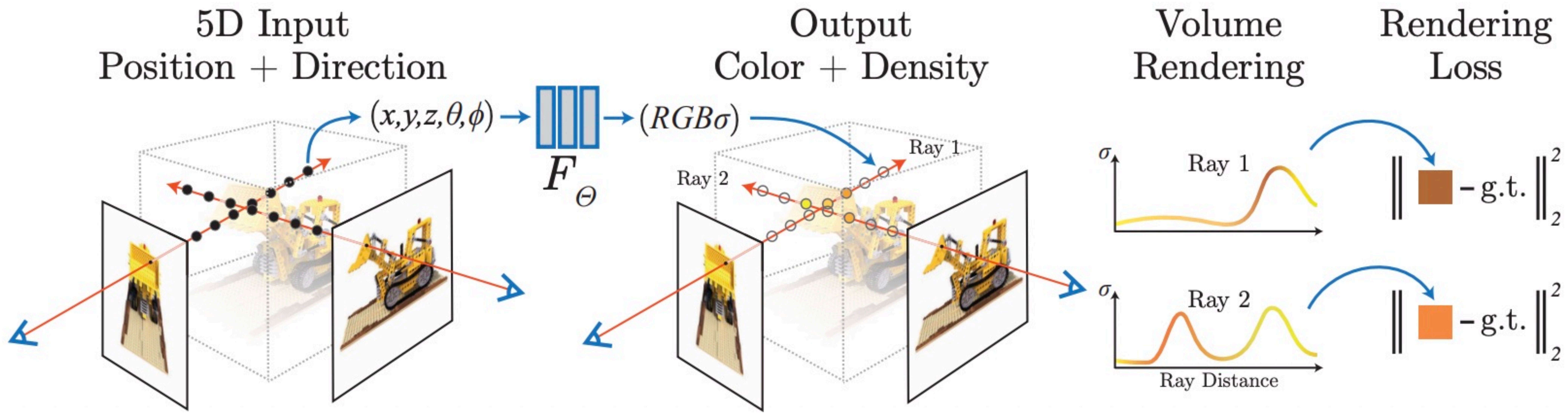
Let's represent that approximation using a deep neural network.

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



Recovering neural radiance fields (NeRF)





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

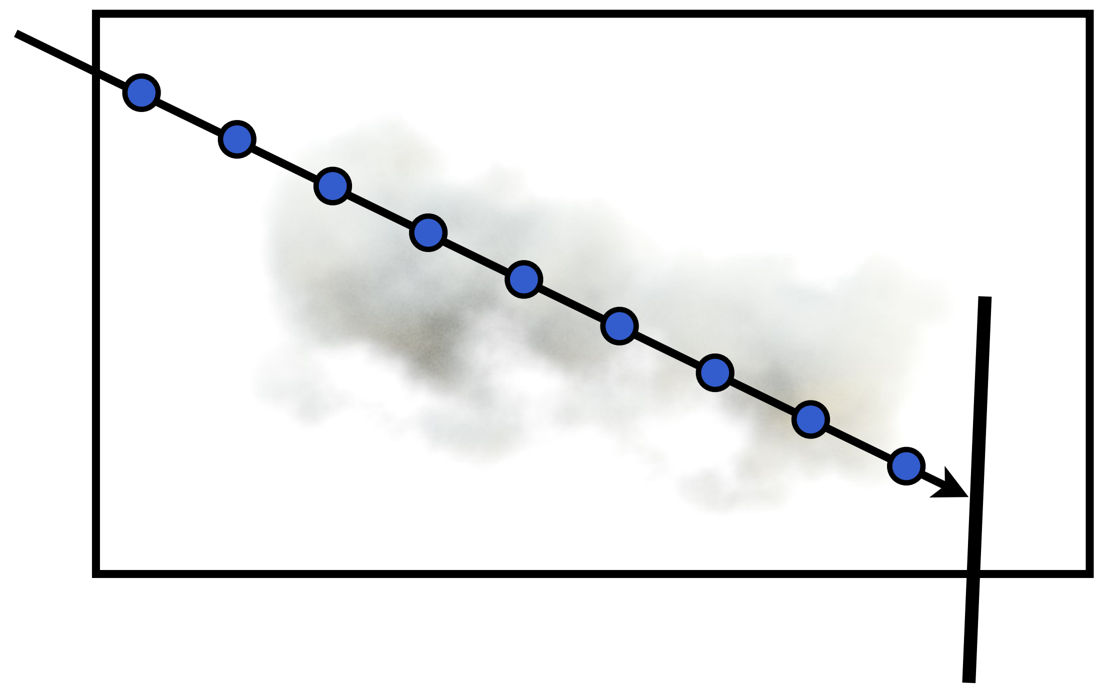
Great visual results!

Credit: Mildenhall 2023

What just happened?

- Continuous coordinate-based representation vs regular grid: DNN is optimized so its weights to produce high-resolution output where needed to match input image data

- Extremely compact representation: trades-off storage 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… we can only query the DNN for opacity)



- Compact representation: DNN can 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 optimize a good NeRF

Is NeRF a “good” representation?

- Ask yourself: what was the task?
  - Optimization (to recover DNN weights) and then rendering high-quality images
  - And doing so on “real world” complexity scenes (not simple surfaces) for which accurate mesh-based representations would be very complex!

- Extreme compactness of DNN representation (MLP) made optimization of high-resolution scenes with viewpoint dependent surfaces possible (scene parameters fit on single GPU)
  - Amount of compression possible while retaining high fidelity was generally surprising to many
  - Flexibility of MLP (fully connected DNN layers) allows optimization to “allocate” parameter capacity as needed to maintain high quality

- NeRF was a great success is showing that IT WAS POSSIBLE to use brute force optimization + a differentiable volume renderer to recover a model of a scene.

Improving rendering performance

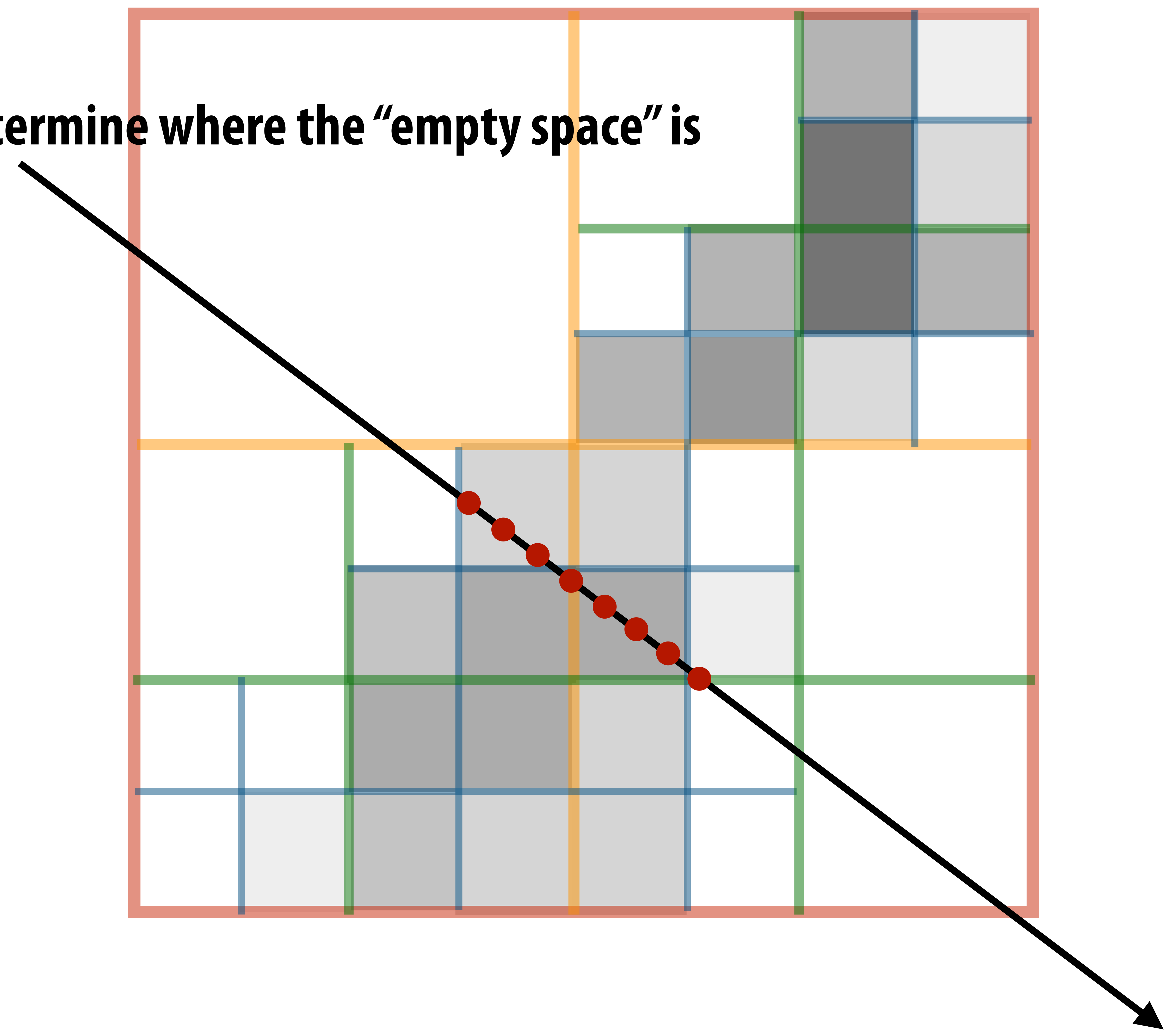
- **But from a performance perspective, NeRF was not so good of a representation.**
- **So let's use our graphics knowledge to move to representations that offer different points in the compression-compute trade-off space**

- **Main ideas:**
  - **Most of a scene is empty space, let's avoid stepping densely through empty space when unnecessary (aka. It's costly to evaluate the DNN during ray marching to find density = 0)**
  - **Shrink the size of the DNN**
  - **Avoid evaluating the DNN altogether when you can**

Recall: ray marching a sparse voxel grid

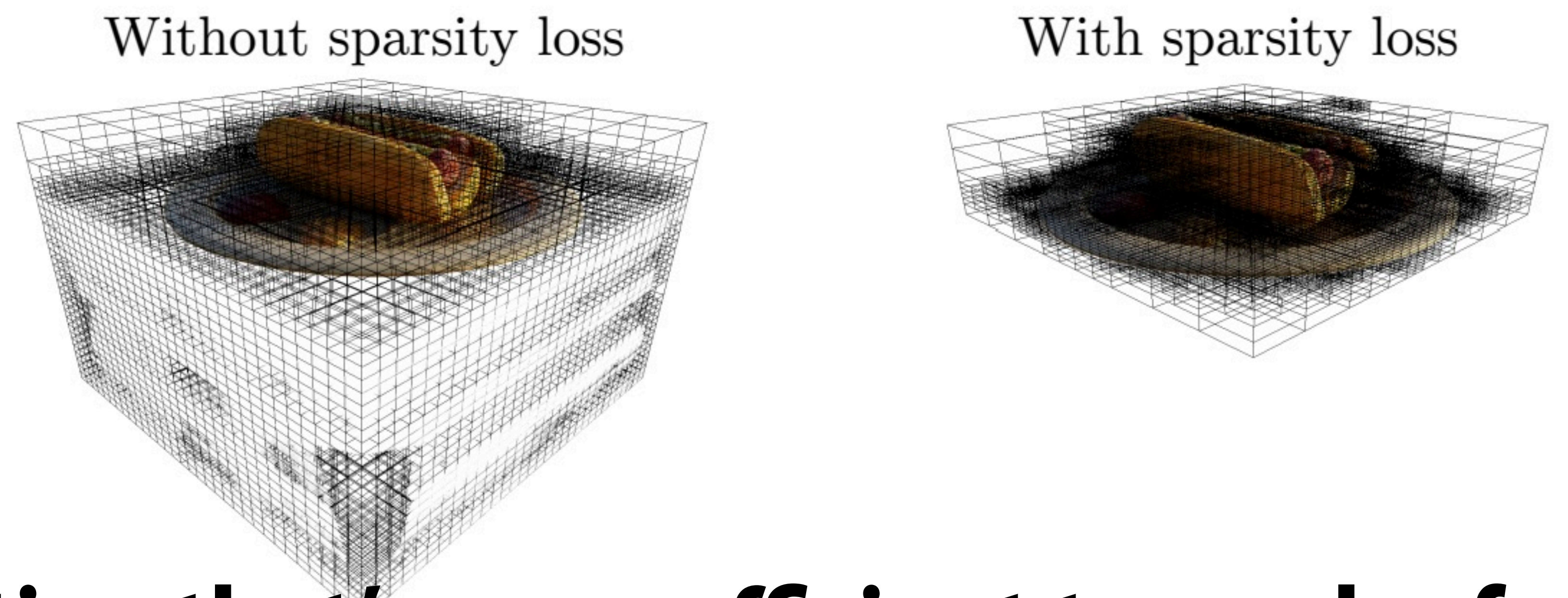
Ray can now "skip" through empty space

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

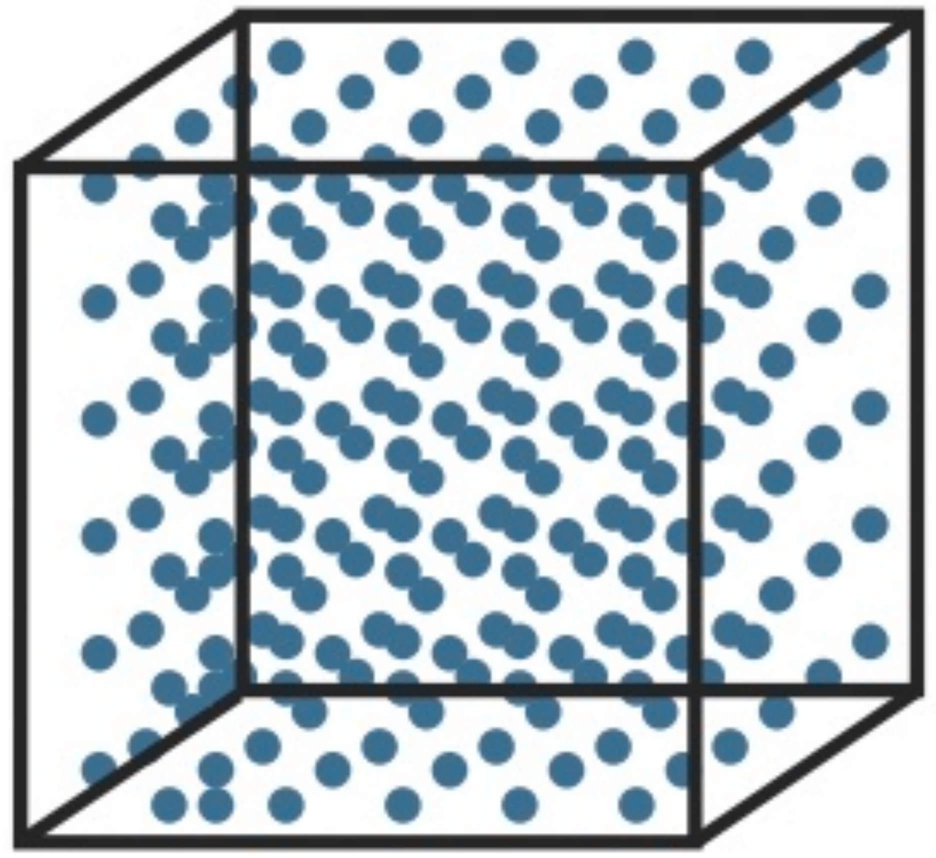


Let's just run NeRF optimization for a bit like before…

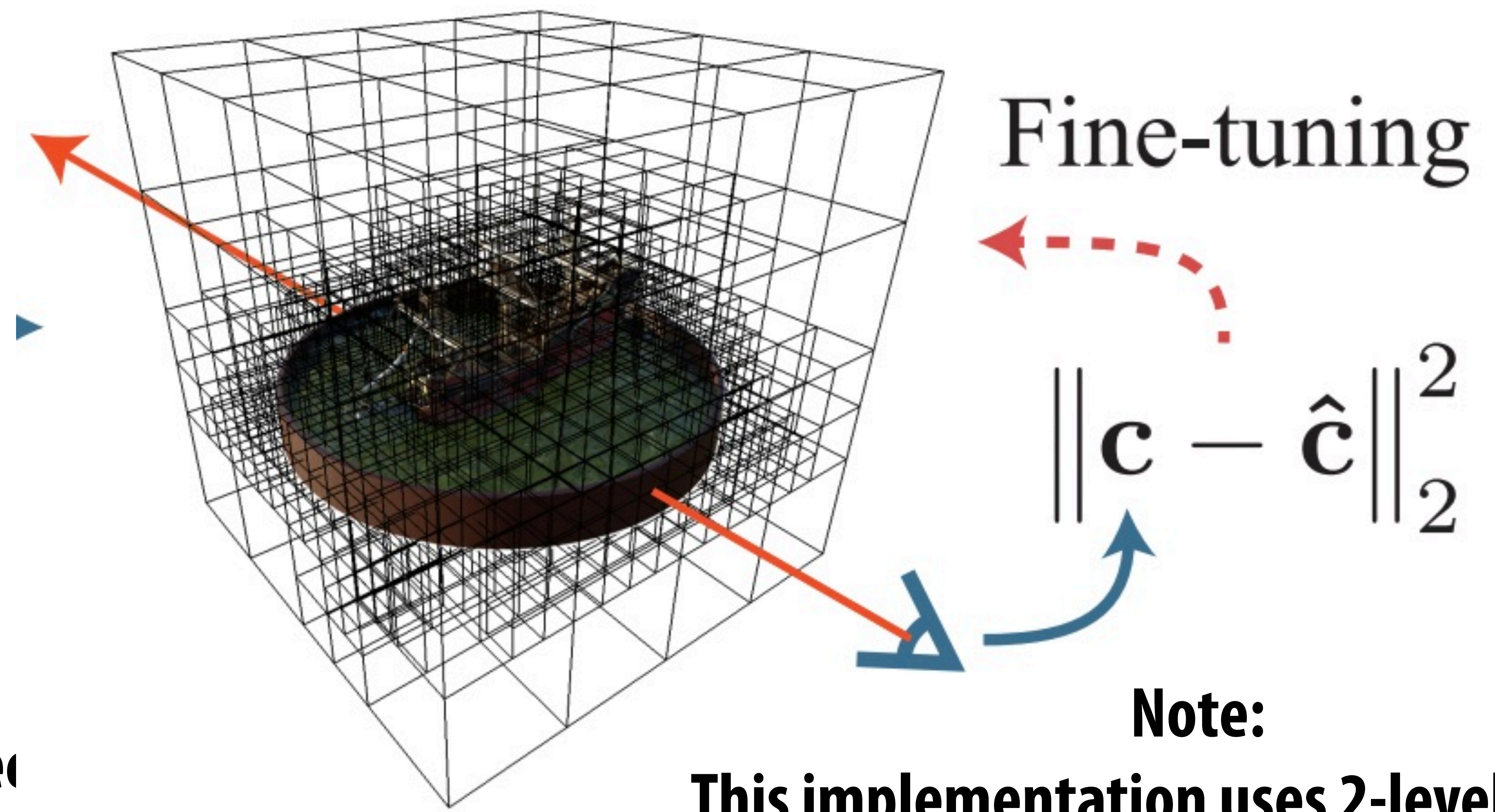
- Optimization will push some opacity values to 0
- DNN tells us where the empty space is!



- 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 a color/density representation at leaves



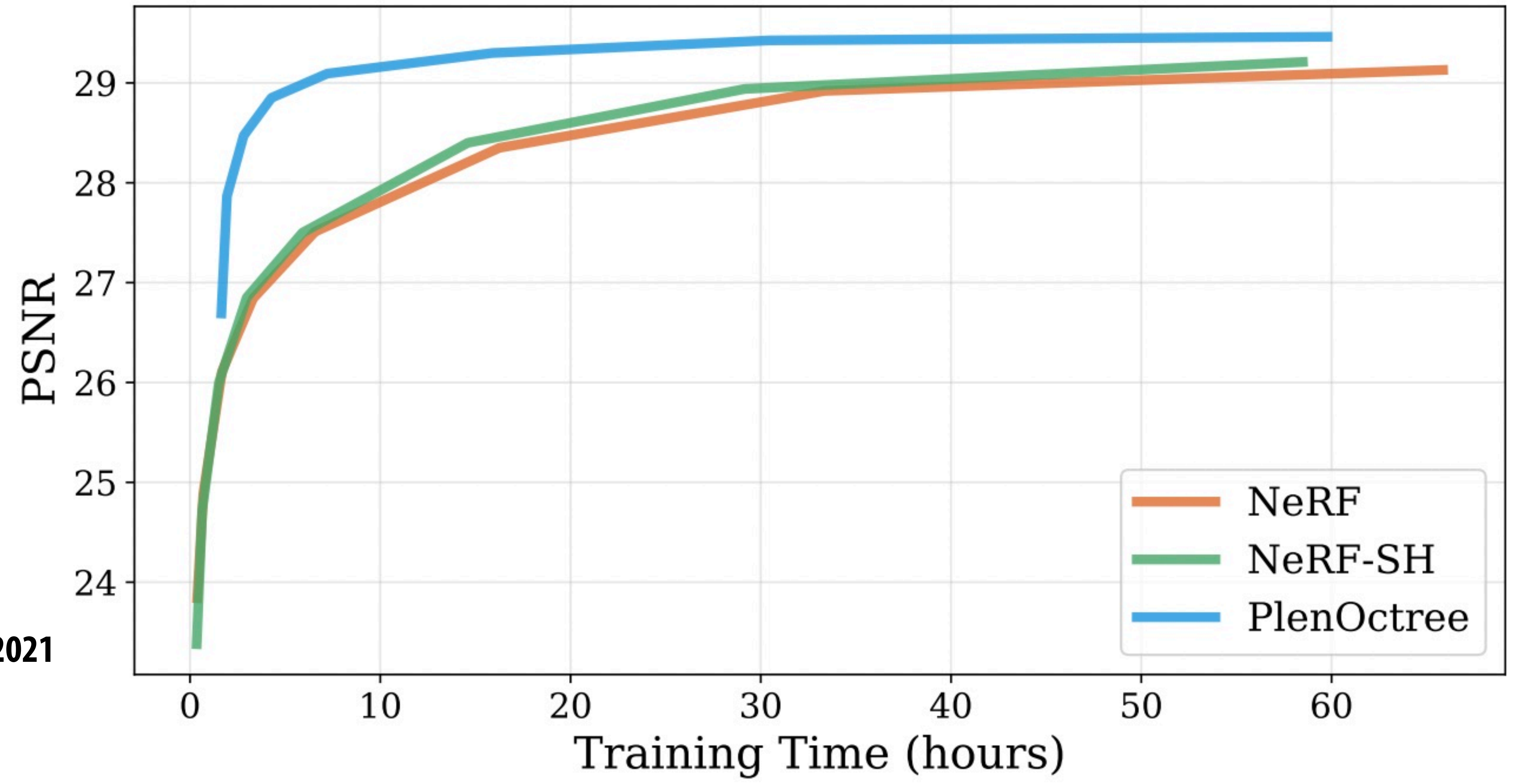
Use the initial MLP to densely sample volume
(Identify the empty space, use it to build a simple octre



Note:
This implementation uses 2-level octreee

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 spatial acceleration structure to replace the "big" DNN. Can use little DNNs at the leaves.
- That structure speeds up rendering (a lot), and it also speeds up "fine tuning" training, since the initial "big" DNN need not be trained to convergence

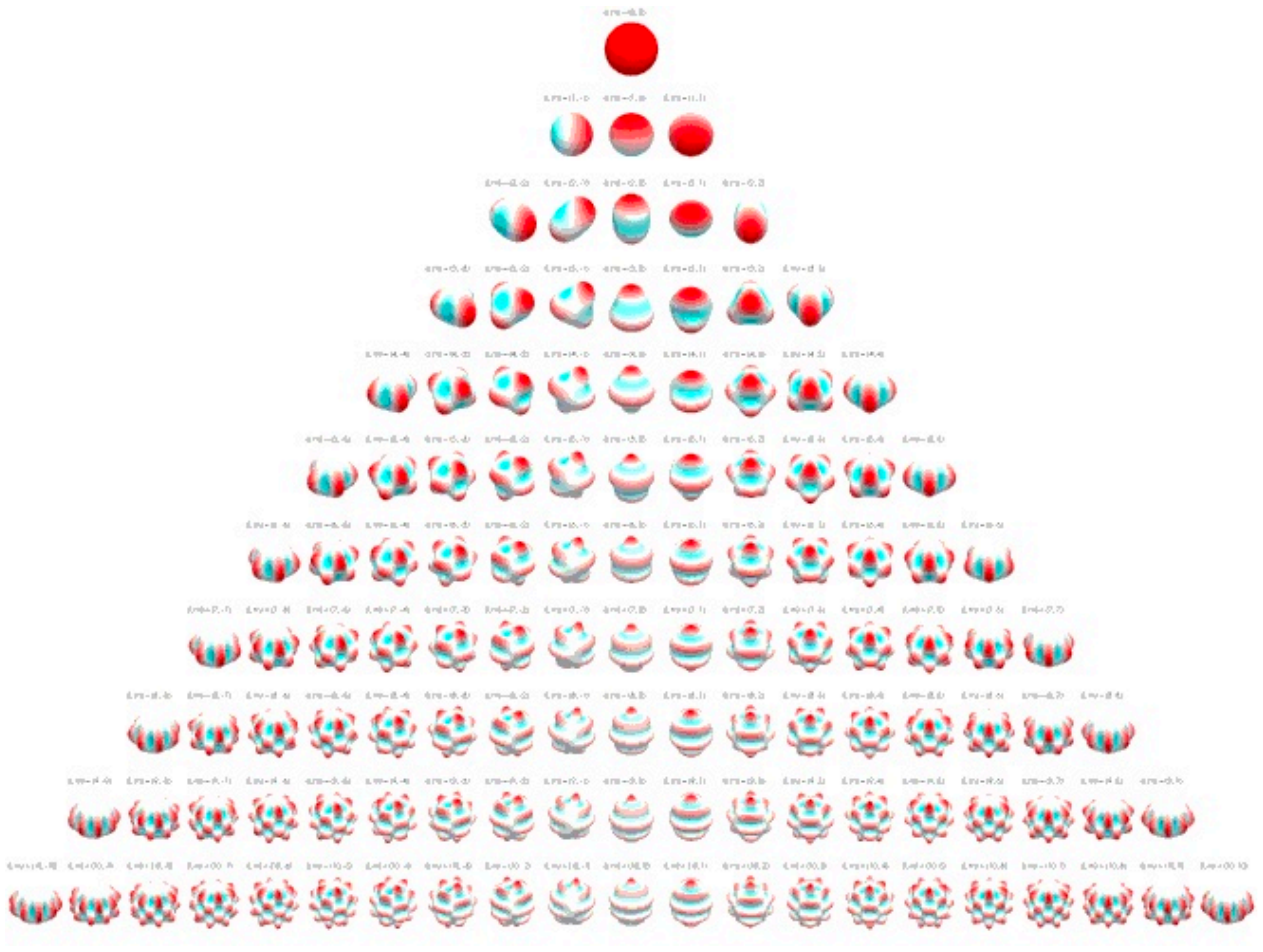


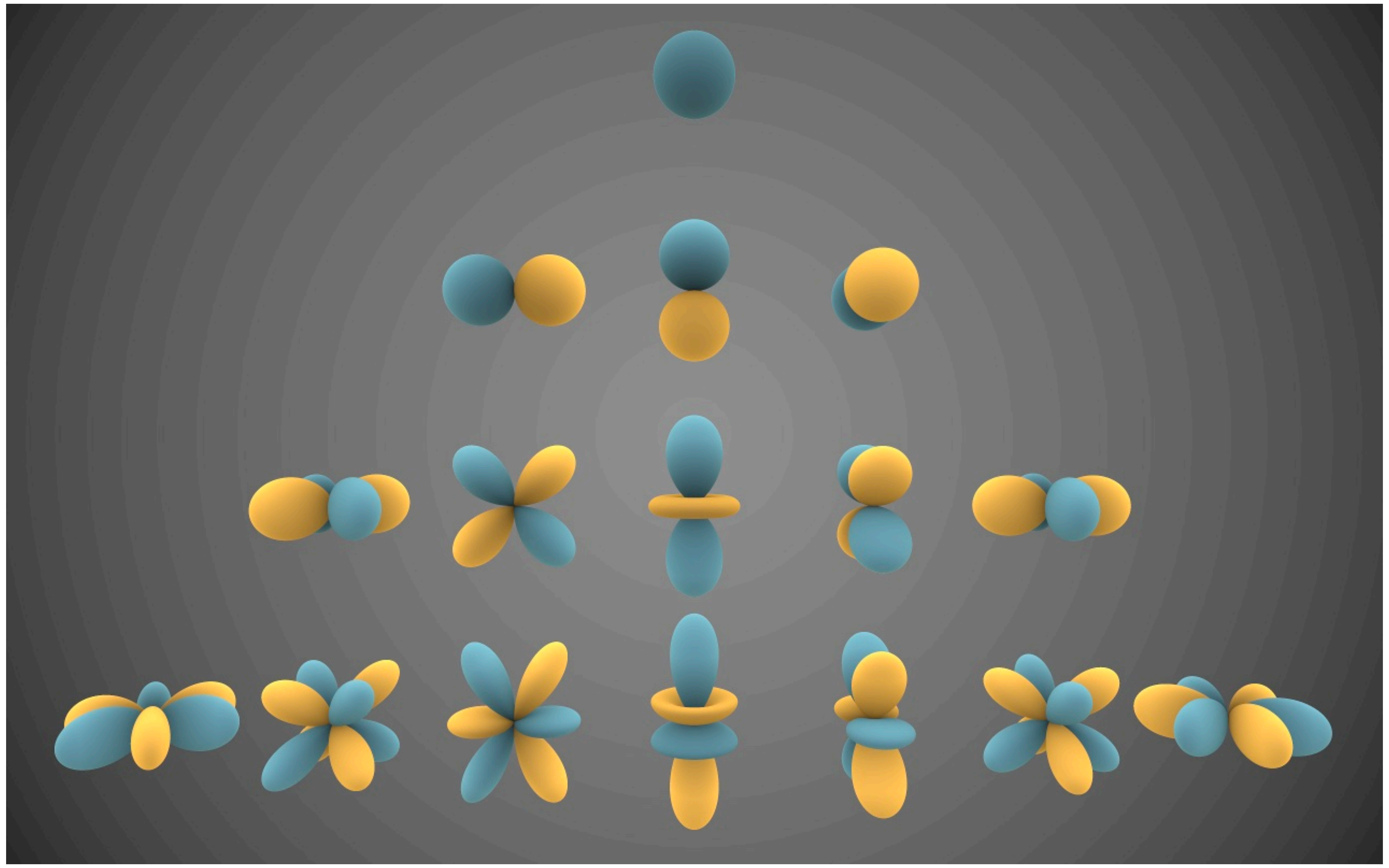
Credit: Yu 2021

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

Another idea: use spherical harmonic representation of radiance

- Useful basis for representing functions that varying smoothly w.r.t direction.
- Analogy: cosine basis on the sphere





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

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

Light probe locations in a game

Here: spherical harmonic probes sampled on a uniform grid

(game compactly stores a few SH coefficients at each point to represent indirect illumination)

Empty Actor
Empty Character
Empty Pawn
Point Light
Player Start
Cube
Sphere
Cylinder
Cone
Plane
Box Trigger
Sphere Trigger
Perspective
Lit
Show
10
10°
0.25
4
Attic (Editor)
Lighting
DDGIVolume1
DirectionalLight
ExponentialHeightFog
PostProcessVolume
SkyLight
Attic_benchmark
attic_converted_Section4_1
attic_objects_ball_star
attic_objects_ball_star2
attic_objects_ball_tennis
attic_objects_blanket_bear
190 actors (1 selected)
Details
World Settin
DDGIVolume1
+ Add Component
Blu
Search Components
DDGIVolume1 (Instance)
DDGIVolumeComponent (Inherited)
Search Details
Transform
GI
DDGIVolume Compone
DDGIVolume Comp
GI
wser

Location X -612.1 Y -357.

Rotation X 0.0° Y 0.0°

Scale X 13.20 Y 23.10

Rays Per Probe 720

Probe Counts X 12 Y 12

Update Priority 1.0

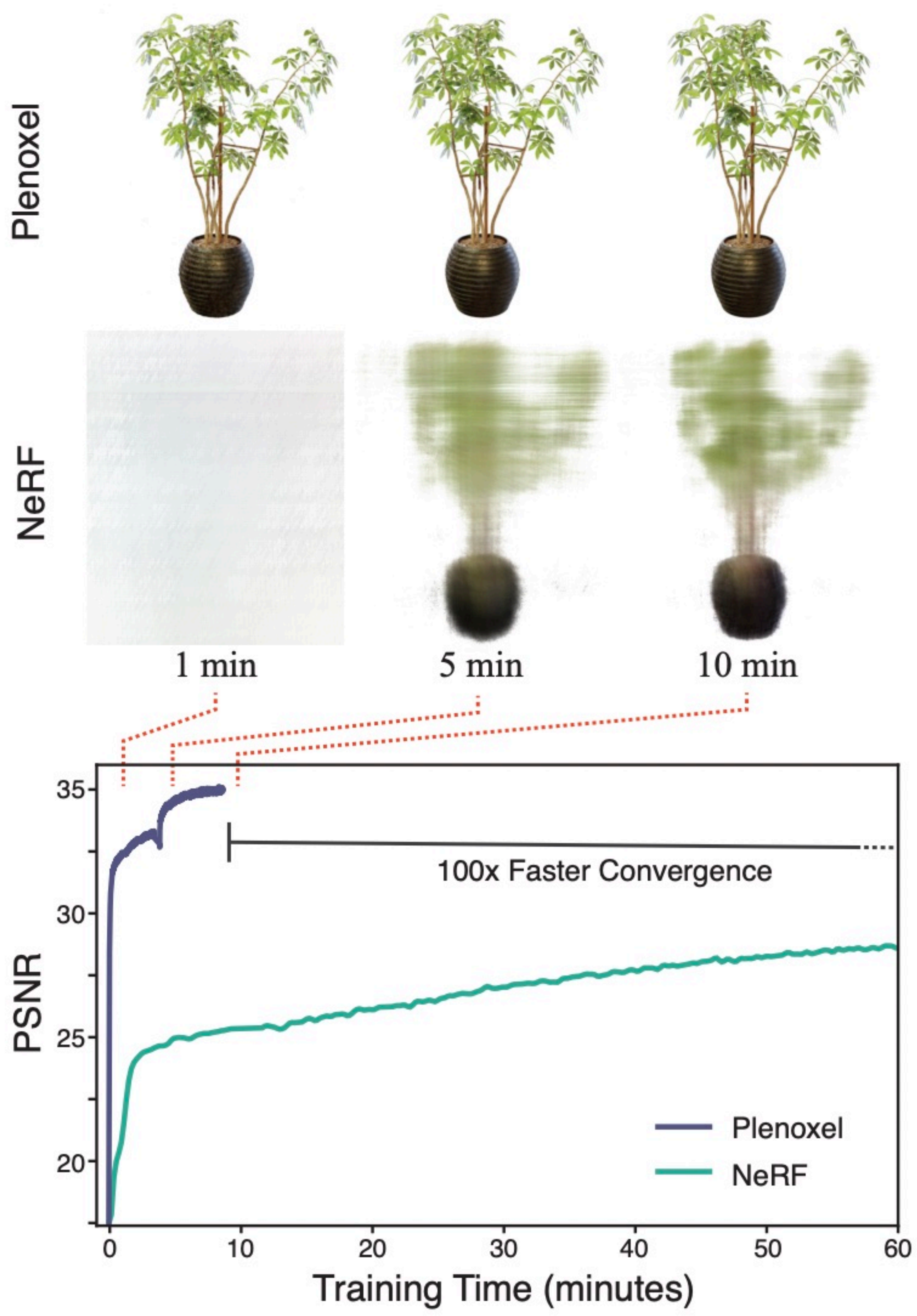
Probe Max Ray Di 100000.0

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 spherical harmonic light" lighting

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



Neural codes… better than a DNN at the leaves

- Rather than store a "per-leaf" DNN or per leaf SH coefficients, store a "code" z_i per leaf node *i*

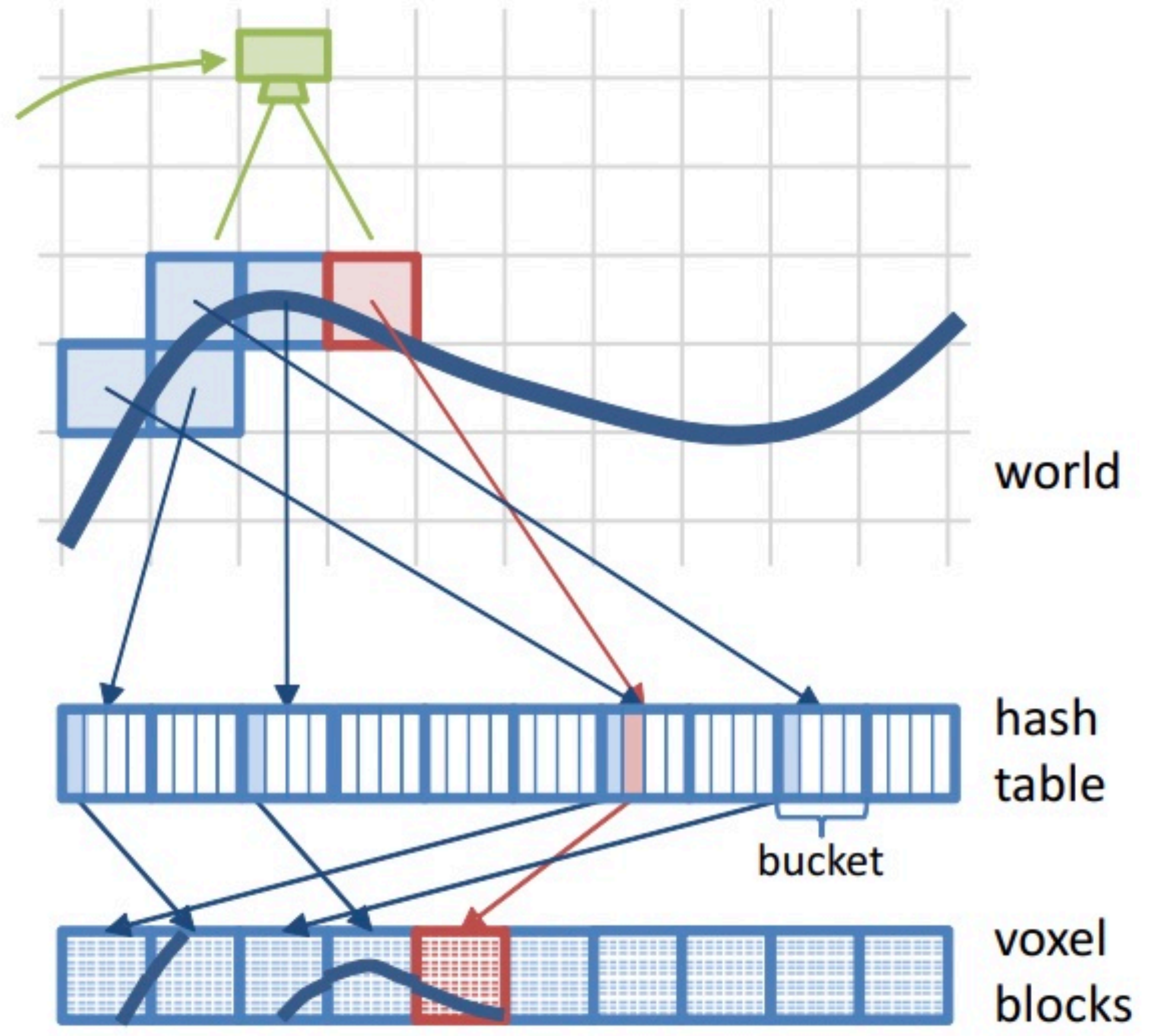
- Ray march through the octree like normal
  - Instead of evaluating DNN_i(x,y,z,phi,theta) for node i corresponding to the current sample point, or evaluating SH coeffients to get radiance… retrieve the neural code z_i
  - Use a DNN to "decode" the code into a radiance or opacity

- Decoder DNN is "small" (cheap to evaluate) since it is only decoding a code into an opacity/color, it doesn't have to represent all spatial occupancy information

Hashing: a parallel friendly approach to storing and retrieving sparse voxels

- Voxel hashing is a fast GPU data structure for supporting sparse voxel representations
  - "Give me data for voxel containing (x,y,z)"
  - Compact in space and "GPU friendly" for fast parallel lookup and update
- TL;DR — use hashing instead of trees
- Developed by the 3D reconstruction community for interactive GPU-accelerated 3D reconstruction

$$H(x,y,z) = (x \cdot p_1 \oplus y \cdot p_2 \oplus z \cdot p_3)\, mod\, n$$



Advanced topic: NVIDIA's instant neural graphics primitives (NGP)

- Combines two ideas:
  - Hierarchy of regular grids
  - Irregular hash data structures

Given position P:

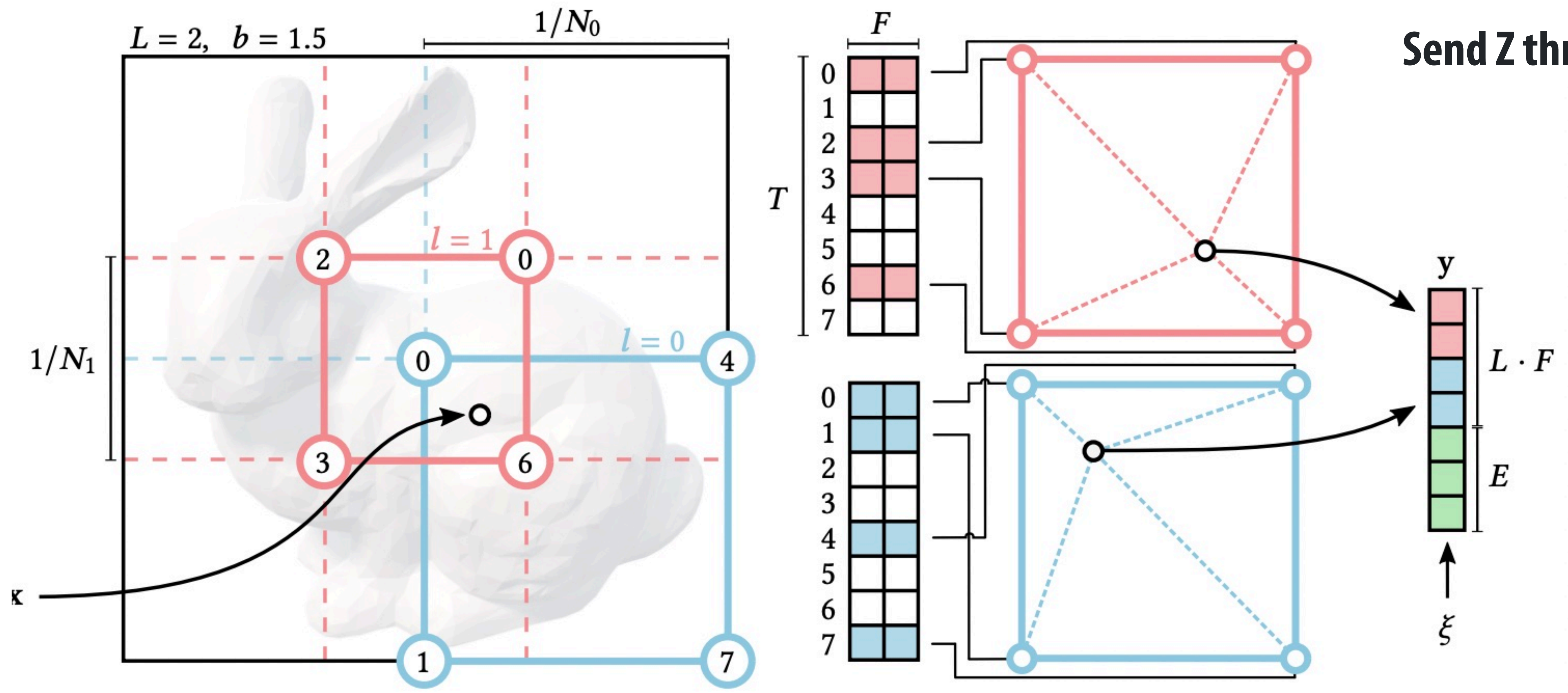
Compute indices of cell containing P on a bunch of different resolution grids (L grids)

At each grid resolution, turn indices into a hash code.

Use hash code to get F components of neural code Z

Concatenate all the codes to get Z (neural code of length L x F)

Send Z through an MLP to decode final value



What is cool:

1. Implementation elegance: no two-step process to find empty space, build structure, then proceed optimizing on another data structure
2. Sparse hash structure is fast… ignore collisions, if collisions happen, just let SGD sort out what the neural code should be.

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 full bag of DNN optimization tricks, etc)

$$\begin{tikzcd}(\mathrm{p}, \omega) \arrow[r] & \boxed{F_\theta(\mathrm{p}, \omega)} \arrow[r] & \begin{matrix}\sigma(\mathrm{p}) \\ c(\mathrm{p}, \omega)\end{matrix}\end{tikzcd}$$

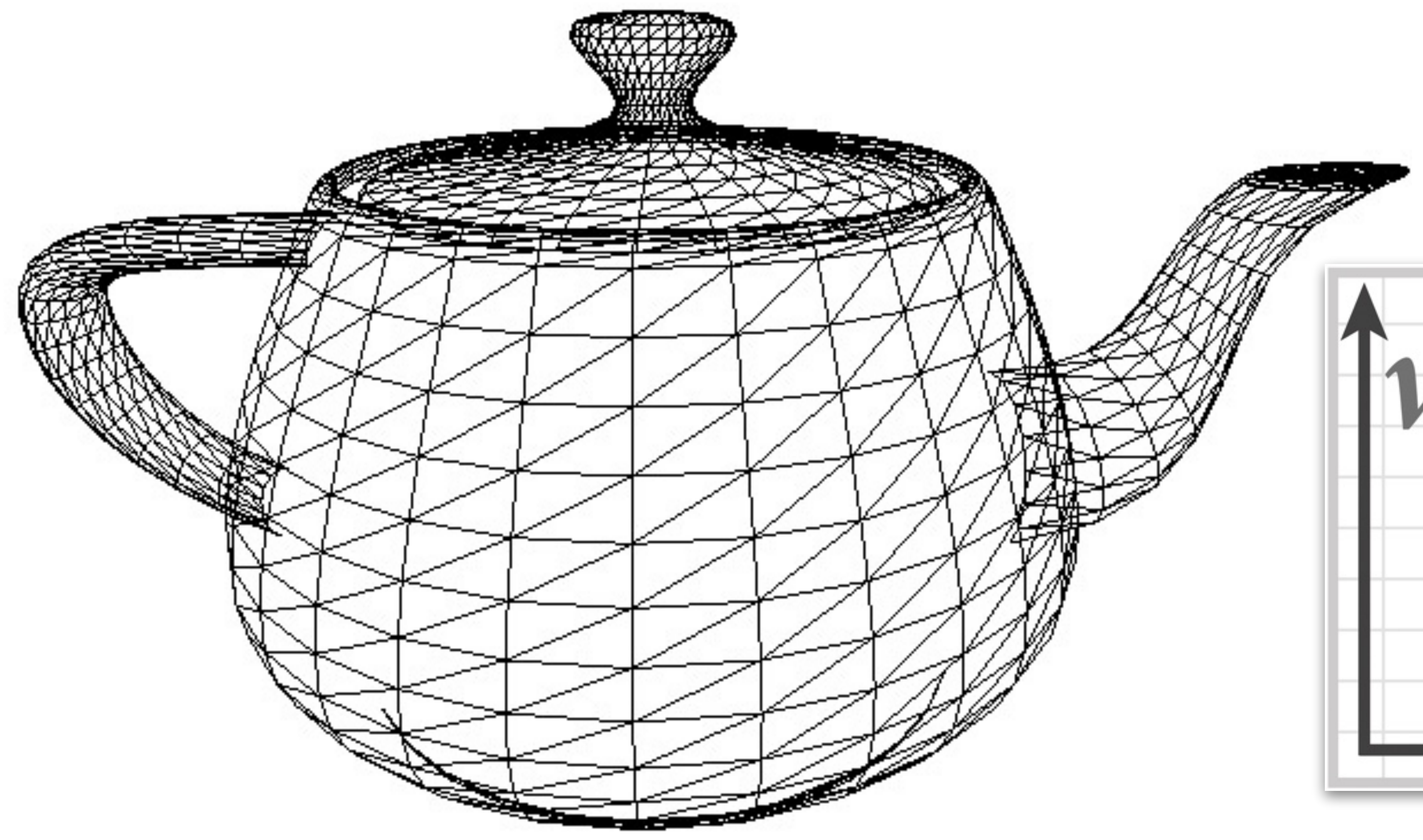
- Then move to a traditional sparse encoding of occupancy (sparse volumetric structure)
  - Now the "topology" of the irregular data structure is fixed
  - Representation of surface/appearance/etc is stored at the nodes of this structure (spherical harmonics, neural code, etc.)
  - Most of the heavy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

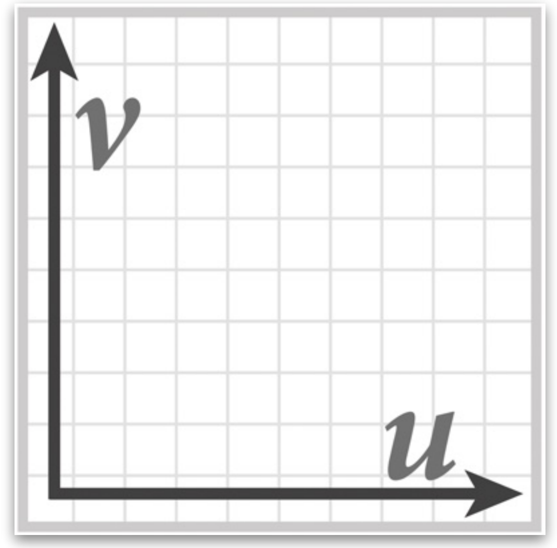
$$\begin{tikzcd}\mathrm{p} \arrow[r] & \boxed{\mathrm{lookup}(\mathrm{p})} \arrow[r] & \begin{matrix}\mathrm{z_p} \\ \sigma_p \\ \mathrm{c_p}(\omega) \\ \mathrm{SH}_p(\omega)\end{matrix}\end{tikzcd}$$

Traditional data structure

- 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

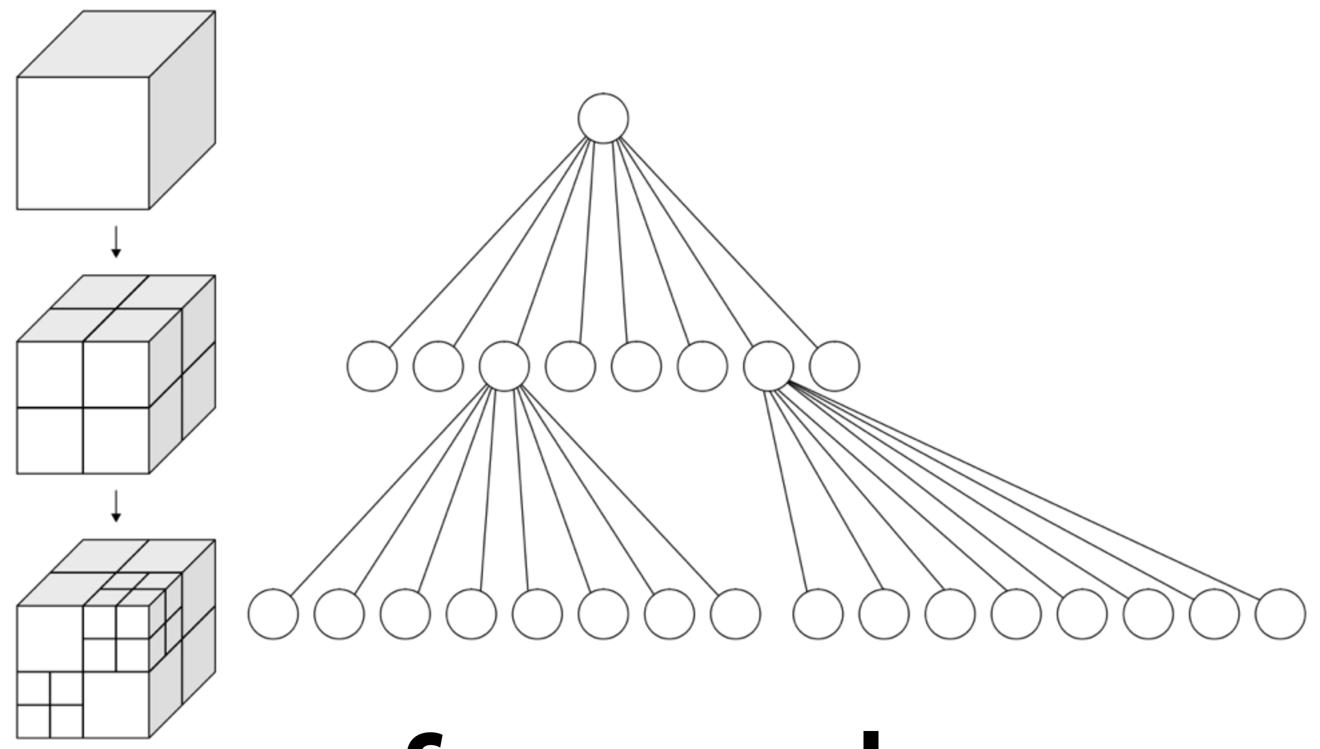
But there are many scene representations



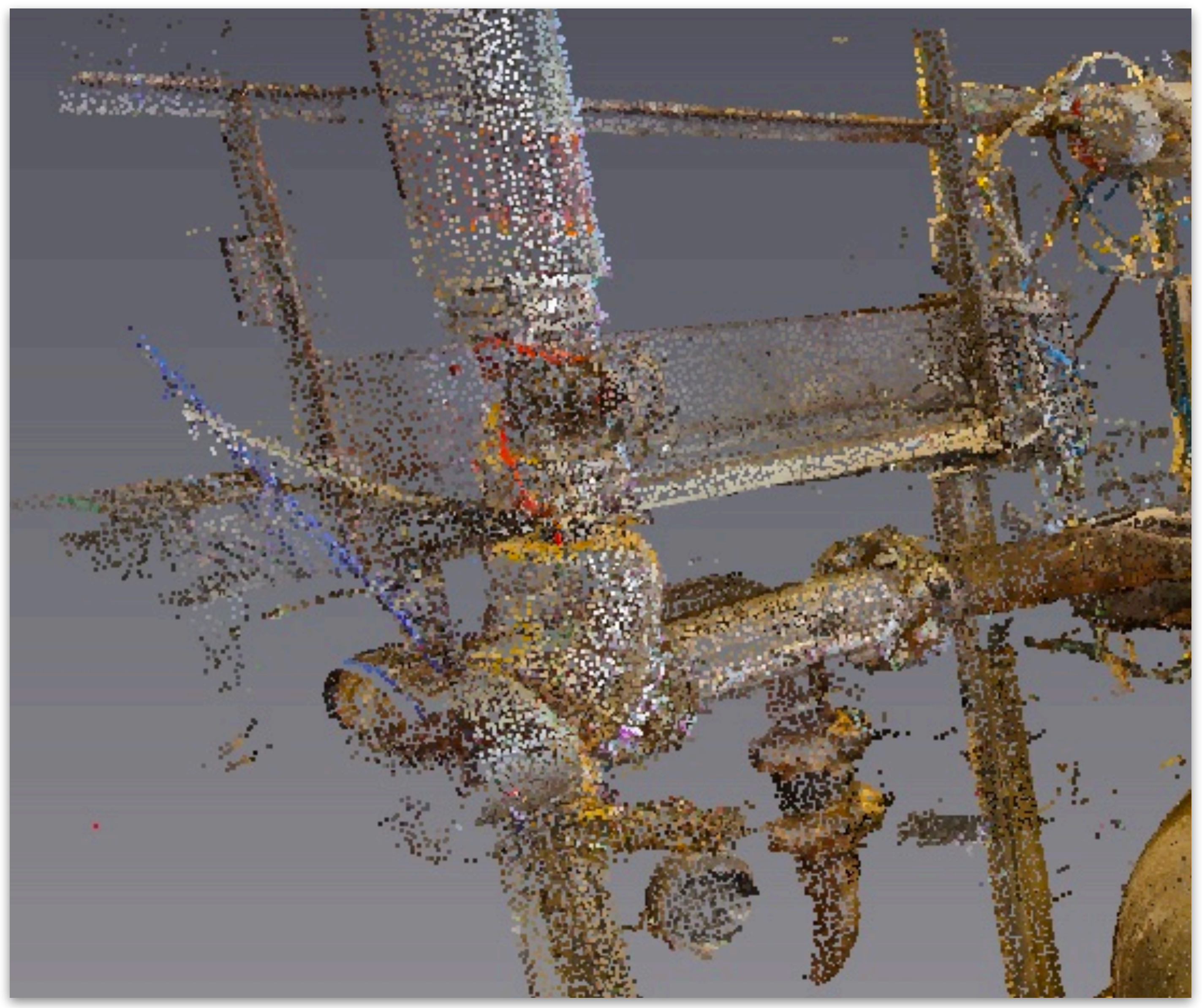


3D triangle mesh + texture map

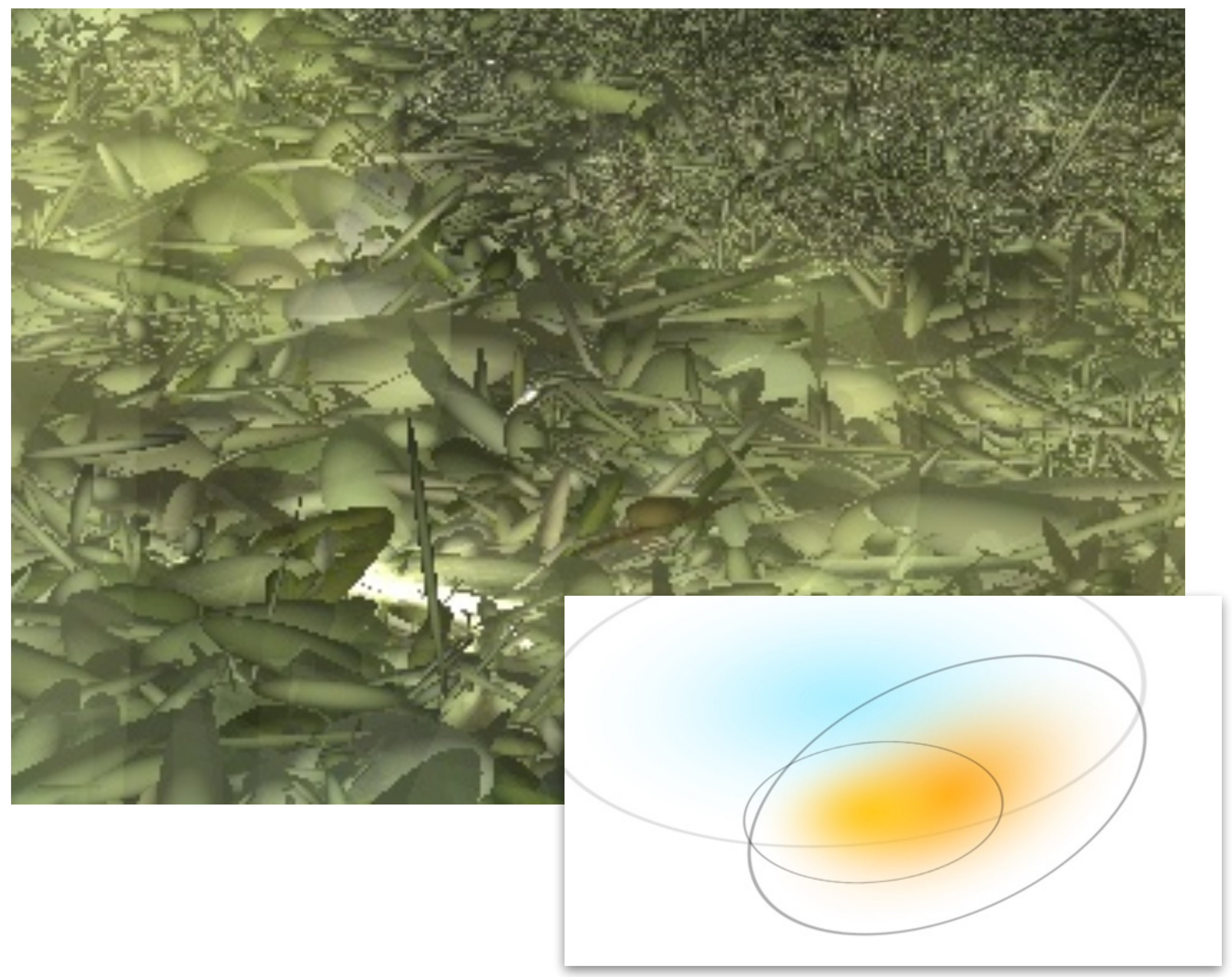
3D volume (voxels)



Sparse voxels

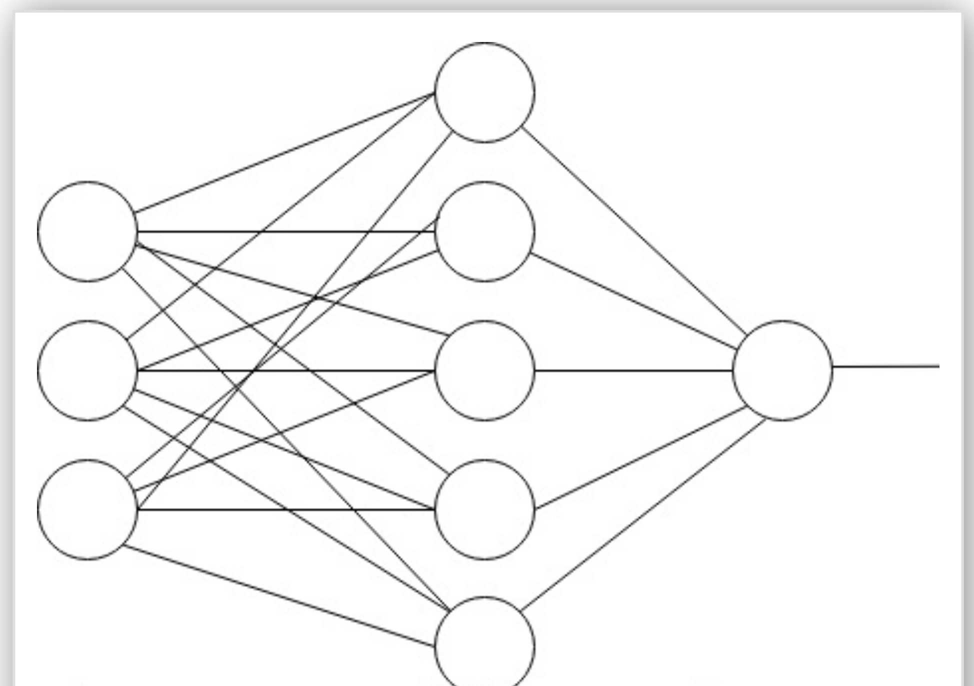


Point cloud (list of points)



Oriented 3D Gaussians

DNN (MLP)



Implicit representations like volumes and DNNs make it hard to know where the "empty space is" (hard to enumerate points on the surface)

So we had to "add in" extra support through spatial data structures like octrees, hash grids, etc.

Explicit representations are much better at the task of enumerating points on the surface (or equivalently, identifying where the empty space is)

Let's consider one explicit representation that can accurately represent the contents of real world scenes…
A list of 3D Gaussians

And conveniently, a simple rasterizer or a ray caster of 3D Gaussians is differentiable!
(The color at a pixel due to a Gaussian blob is just an exponential)

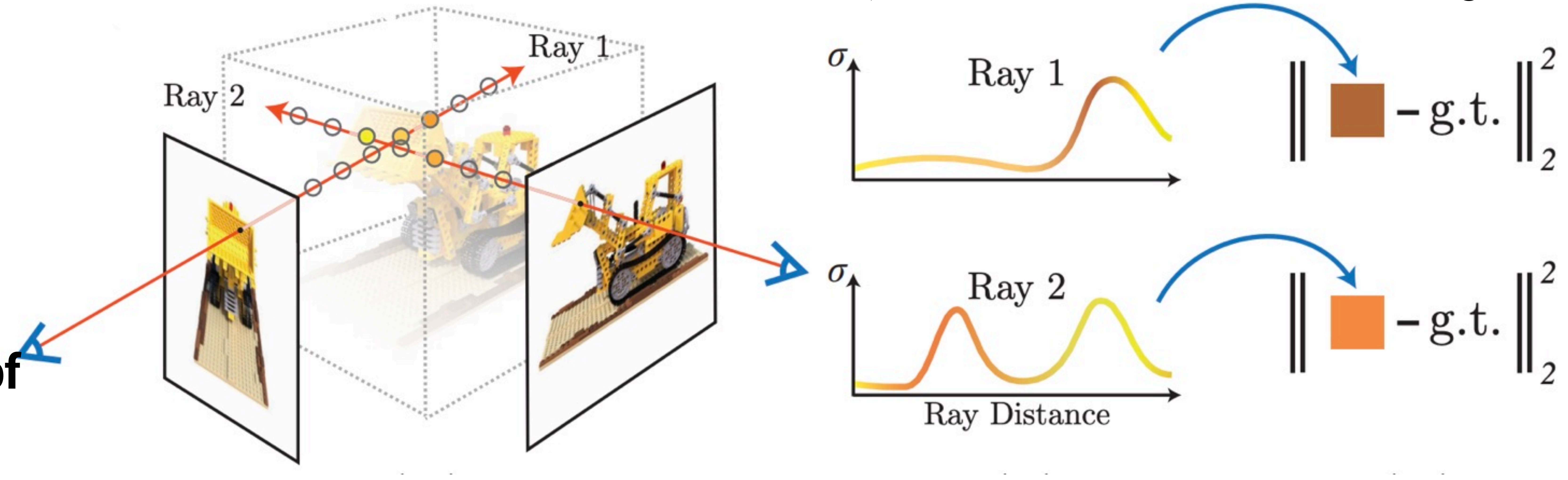
Optimization to recover parameters of 3D Gaussians, not voxel parameters, DNN weights, or neural codes

- Earlier in lecture: optimization produces color and opacity at each voxel, or DNN parameters, etc..

- 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)

Compute radiance along ray through scene

Compare to actual image



Ray 1
Ray 2
σ
Ray 1
$\left\| \blacksquare - \text{g.t.} \right\|_2^2$
σ
Ray 2
Ray Distance
$\left\| \blacksquare - \text{g.t.} \right\|_2^2$

Key idea: differentiable Gaussian splatting rendering to compute dL/d(color)d(radius)d(position)

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

Summary

- Volumes (continuous color/opacity fields) and 3D Gaussian points are representations of geometry and materials that lend themselves to simple differential rendering algorithms

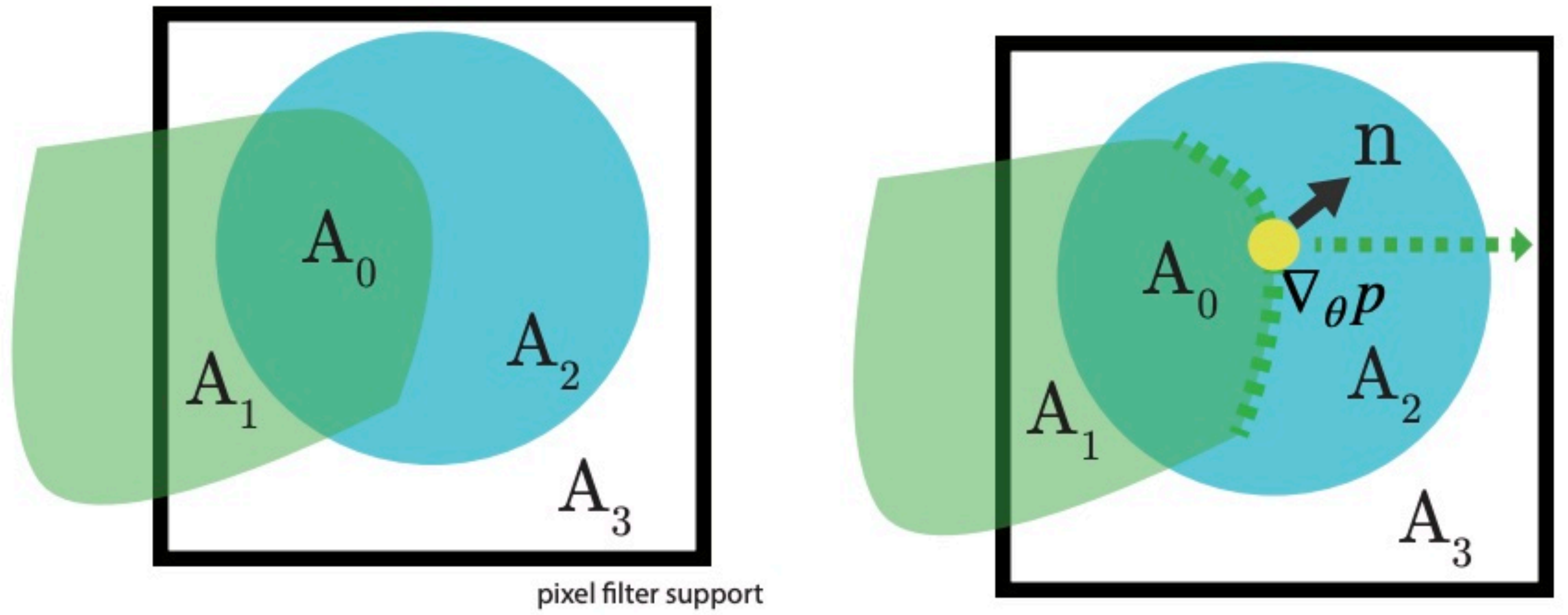
- Modern high-performance optimization techniques are amazingly effective at recovering the parameters of these representations.

- Together, these two observations have led to rapid progress in reconstructing scenes from (potentially sparse) set of photos

- Some of these solutions employ interesting combinations of neural structures (learned DNN weights, or neural "codes") and "traditional" graphics representations like spatial accelerations structures or compact bases for radiance.
  - Takeaway for graphics students in 2026: need to be a master of both domains!

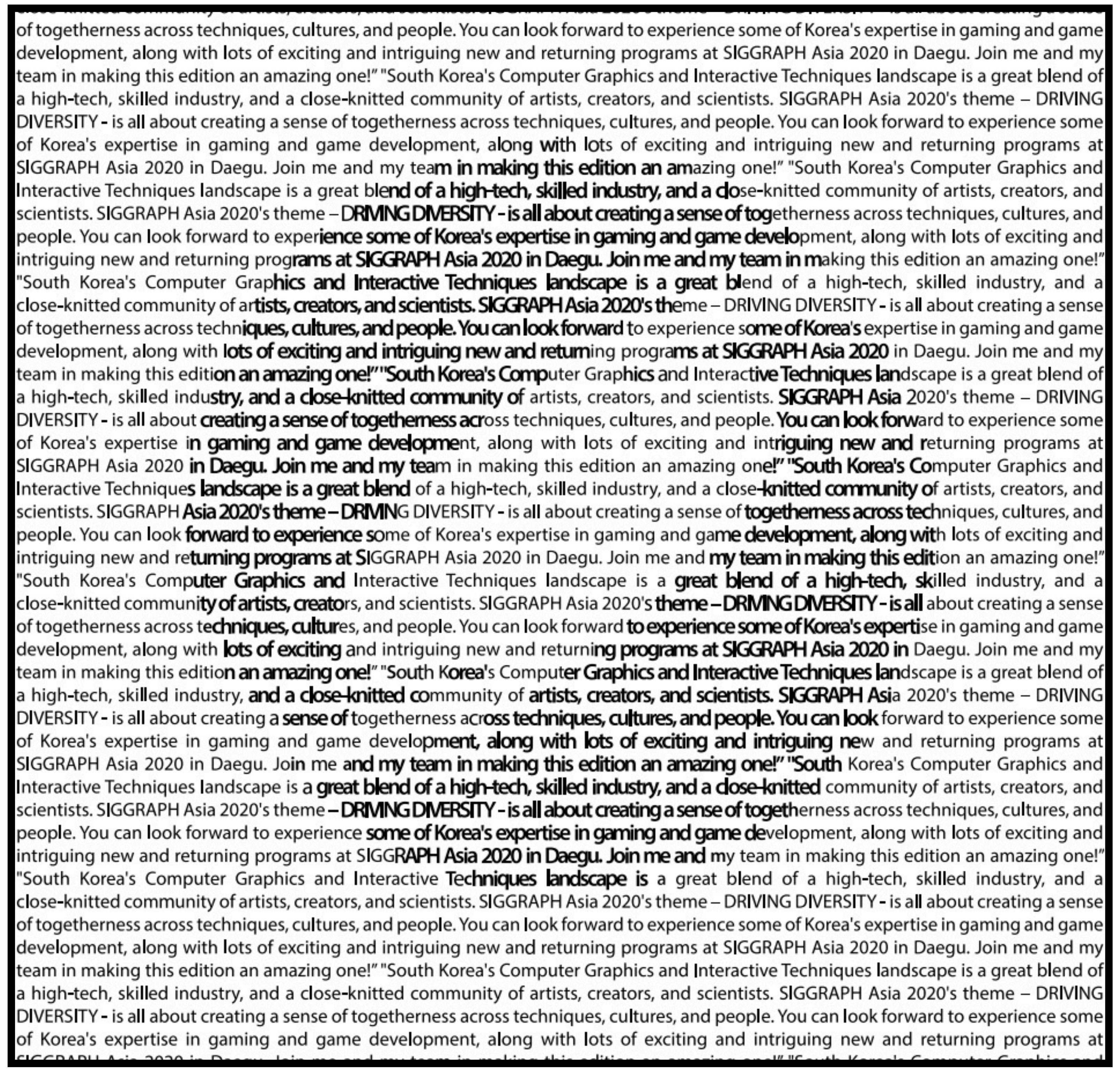
What about triangles and textures?

- What are the parameters of a mesh? (Vertex positions, number of vertices, connectivity, etc.)
- Computing the gradient of a rendering subject to these parameters is challenging.
    - Consider simple case of fixed vertex count and fixed topology: change in rendering output at a single sample point is discontinuous at object silhouettes as a function of vertex position changes (might see object A, then see object B if object A moves!)
    - But integral of radiance over a pixel (post resolve output) is not discontinuous… (fraction of pixel covered)



Example uses of differential rasterizers/ray tracers

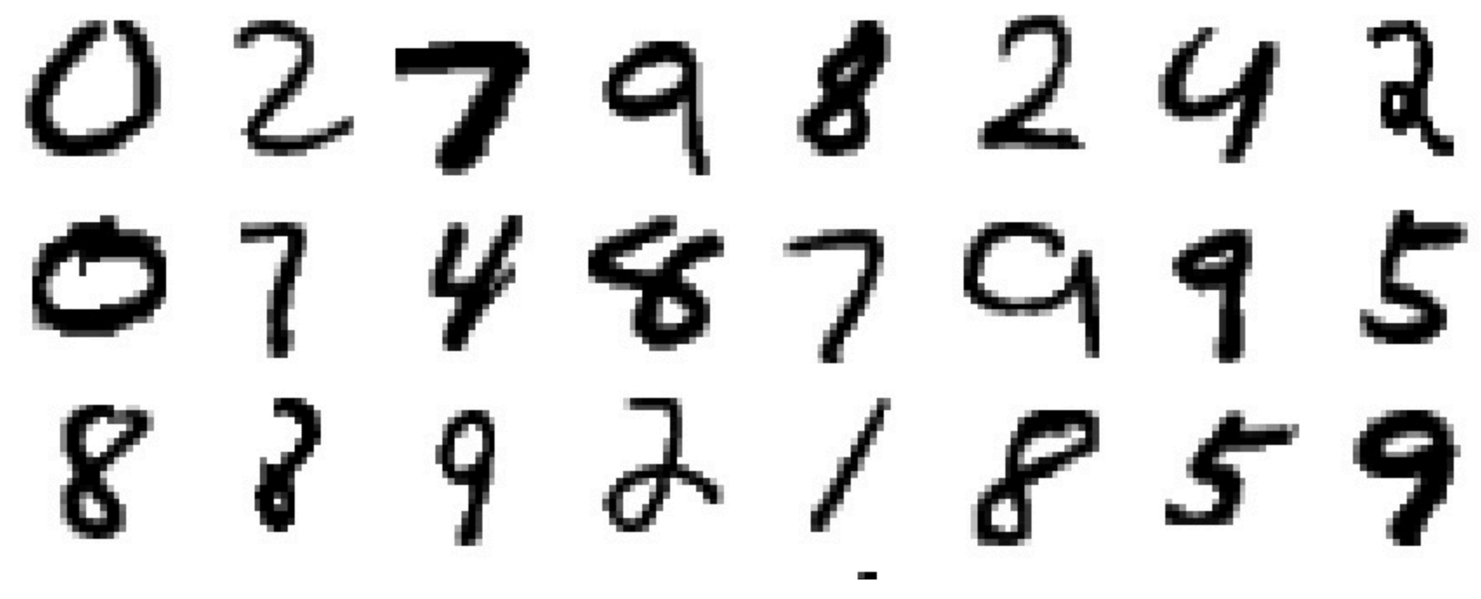
- **Optimize parameters of SVG file to get a certain look**

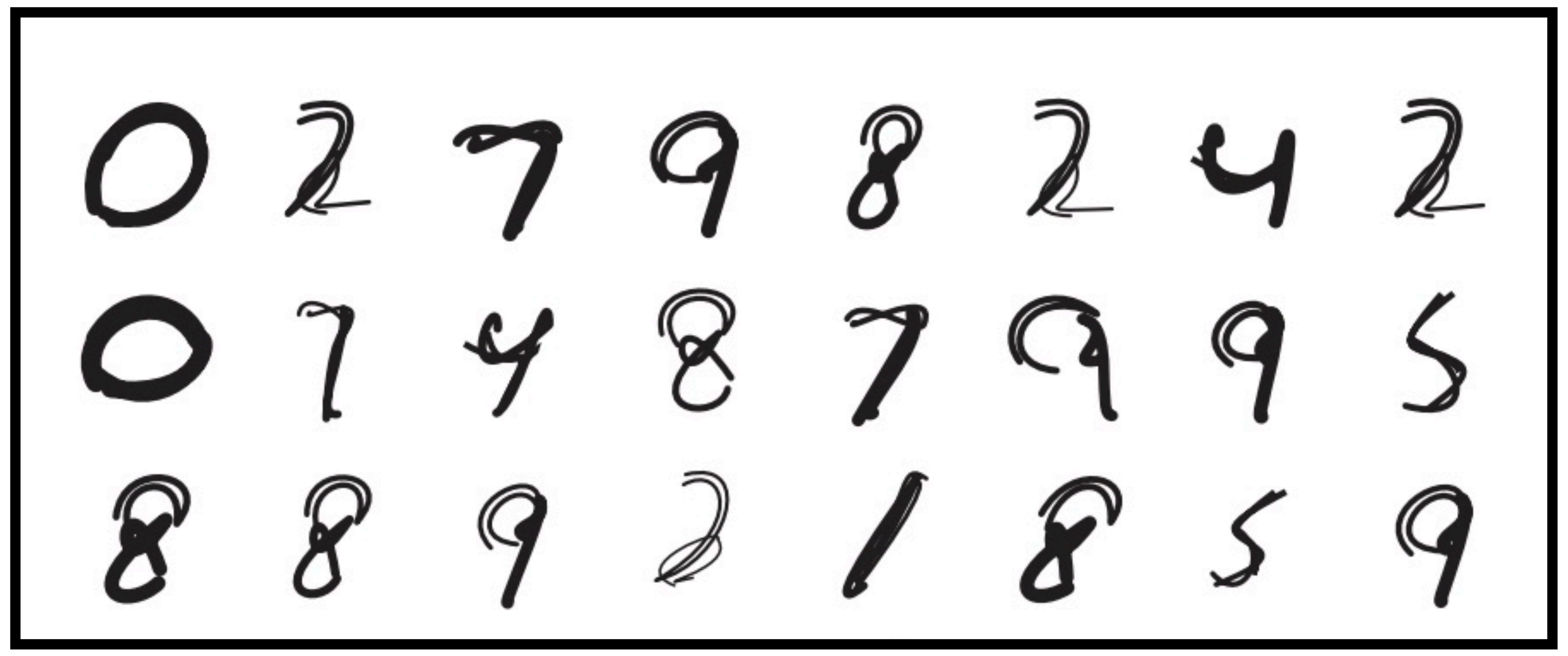


of togetherness across techniques, cultures, and people. You can look forward to experience some of Korea's expertise in gaming and game
development, along with lots of exciting and intriguing new and returning programs at SIGGRAPH Asia 2020 in Daegu. Join me and my
team in making this edition an amazing one!" "South Korea's Computer Graphics and Interactive Techniques landscape is a great blend of
a high-tech, skilled industry, and a close-knitted community of artists, creators, and scientists. SIGGRAPH Asia 2020's theme – DRIVING
DIVERSITY - is all about creating a sense of togetherness across techniques, cultures, and people. You can look forward to experience some
of Korea's expertise in gaming and game development, along with lots of exciting and intriguing new and returning programs at
SIGGRAPH Asia 2020 in Daegu. Join me and my team in making this edition an amazing one!" "South Korea's Computer Graphics and
Interactive Techniques landscape is a great blend of a high-tech, skilled industry, and a close-knitted community of artists, creators, and
scientists. SIGGRAPH Asia 2020's theme – DRIVING DIVERSITY - is all about creating a sense of togetherness across techniques, cultures, and
people. You can look forward to experience some of Korea's expertise in gaming and game development, along with lots of exciting and
intriguing new and returning programs at SIGGRAPH Asia 2020 in Daegu. Join me and my team in making this edition an amazing one!"
"South Korea's Computer Graphics and Interactive Techniques landscape is a great blend of a high-tech, skilled industry, and a
close-knitted community of artists, creators, and scientists. SIGGRAPH Asia 2020's theme – DRIVING DIVERSITY - is all about creating a sense
of togetherness across techniques, cultures, and people. You can look forward to experience some of Korea's expertise in gaming and game
development, along with lots of exciting and intriguing new and returning programs at SIGGRAPH Asia 2020 in Daegu. Join me and my
team in making this edition an amazing one!" "South Korea's Computer Graphics and Interactive Techniques landscape is a great blend of
a high-tech, skilled industry, and a close-knitted community of artists, creators, and scientists. SIGGRAPH Asia 2020's theme – DRIVING
DIVERSITY - is all about creating a sense of togetherness across techniques, cultures, and people. You can look forward to experience some
of Korea's expertise in gaming and game development, along with lots of exciting and intriguing new and returning programs at
SIGGRAPH Asia 2020 in Daegu. Join me and my team in making this edition an amazing one!" "South Korea's Computer Graphics and
Interactive Techniques landscape is a great blend of a high-tech, skilled industry, and a close-knitted community of artists, creators, and
scientists. SIGGRAPH Asia 2020's theme – DRIVING DIVERSITY - is all about creating a sense of togetherness across techniques, cultures, and
people. You can look forward to experience some of Korea's expertise in gaming and game development, along with lots of exciting and
intriguing new and returning programs at SIGGRAPH Asia 2020 in Daegu. Join me and my team in making this edition an amazing one!"
"South Korea's Computer Graphics and Interactive Techniques landscape is a great blend of a high-tech, skilled industry, and a
close-knitted community of artists, creators, and scientists. SIGGRAPH Asia 2020's theme – DRIVING DIVERSITY - is all about creating a sense
of togetherness across techniques, cultures, and people. You can look forward to experience some of Korea's expertise in gaming and game
development, along with lots of exciting and intriguing new and returning programs at SIGGRAPH Asia 2020 in Daegu. Join me and my
team in making this edition an amazing one!" "South Korea's Computer Graphics and Interactive Techniques landscape is a great blend of
a high-tech, skilled industry, and a close-knitted community of artists, creators, and scientists. SIGGRAPH Asia 2020's theme – DRIVING
DIVERSITY - is all about creating a sense of togetherness across techniques, cultures, and people. You can look forward to experience some
of Korea's expertise in gaming and game development, along with lots of exciting and intriguing new and returning programs at
SIGGRAPH Asia 2020 in Daegu. Join me and my team in making this edition an amazing one!" "South Korea's Computer Graphics and
Interactive Techniques landscape is a great blend of a high-tech, skilled industry, and a close-knitted community of artists, creators, and
scientists. SIGGRAPH Asia 2020's theme – DRIVING DIVERSITY - is all about creating a sense of togetherness across techniques, cultures, and
people. You can look forward to experience some of Korea's expertise in gaming and game development, along with lots of exciting and
intriguing new and returning programs at SIGGRAPH Asia 2020 in Daegu. Join me and my team in making this edition an amazing one!"
"South Korea's Computer Graphics and Interactive Techniques landscape is a great blend of a high-tech, skilled industry, and a
close-knitted community of artists, creators, and scientists. SIGGRAPH Asia 2020's theme – DRIVING DIVERSITY - is all about creating a sense
of togetherness across techniques, cultures, and people. You can look forward to experience some of Korea's expertise in gaming and game
development, along with lots of exciting and intriguing new and returning programs at SIGGRAPH Asia 2020 in Daegu. Join me and my
team in making this edition an amazing one!" "South Korea's Computer Graphics and Interactive Techniques landscape is a great blend of
a high-tech, skilled industry, and a close-knitted community of artists, creators, and scientists. SIGGRAPH Asia 2020's theme – DRIVING
DIVERSITY - is all about creating a sense of togetherness across techniques, cultures, and people. You can look forward to experience some
of Korea's expertise in gaming and game development, along with lots of exciting and intriguing new and returning programs at

Optimize "bold" parameter of SVG text to match image to right…

Optimize curve control points to match images of numbers.

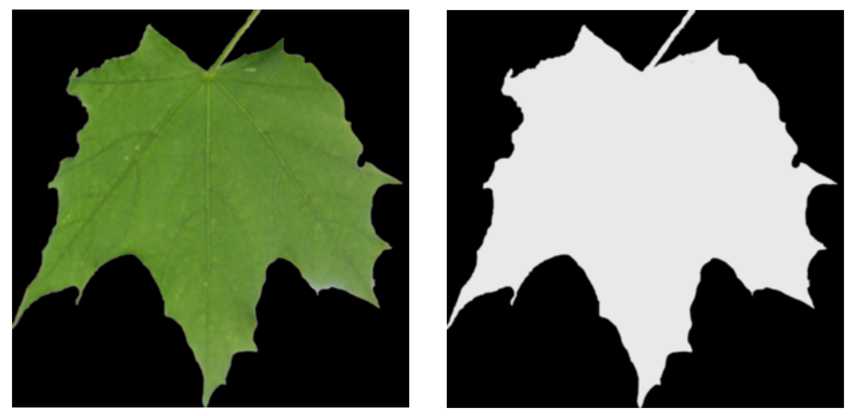




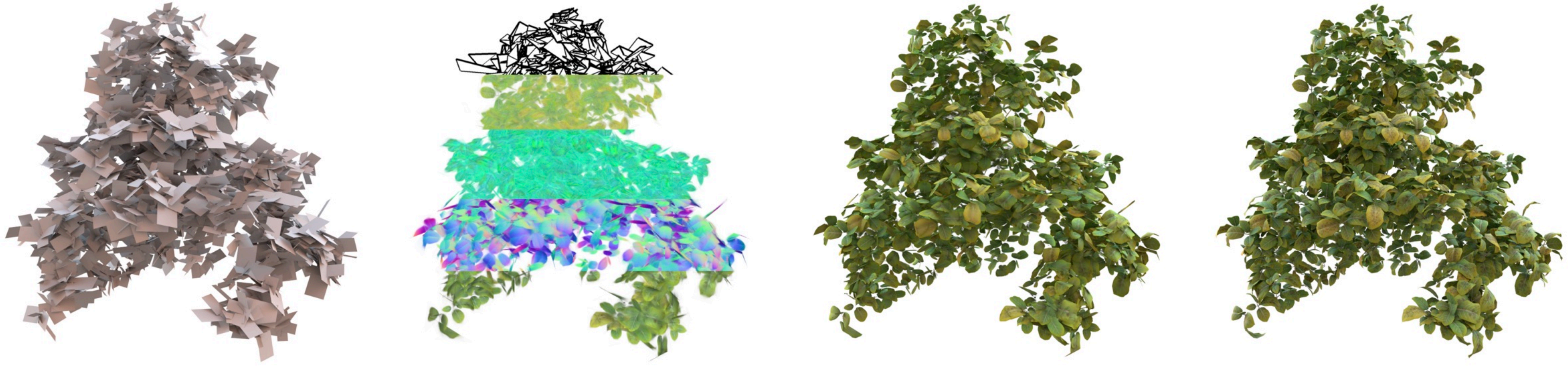
[Li et al. 2020 Differentiable Vector Graphics Rasterization for Editing and Learning]

Example uses of differential rasterizers/ray tracers

- Optimize vertex positions (at fixed vertex count) and also texture map pixels (alpha matte) to make the best low-poly representation of a mesh (when compared to renderings of a reference high poly mesh)



Example alpha texture for a leaf

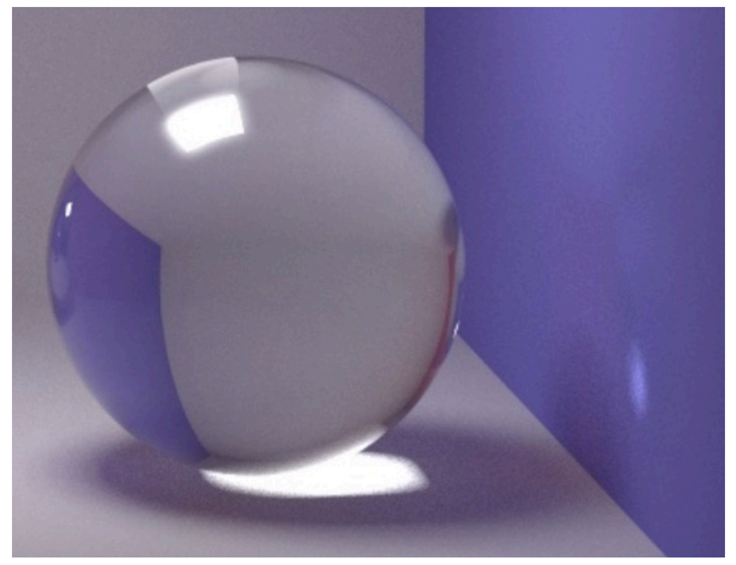


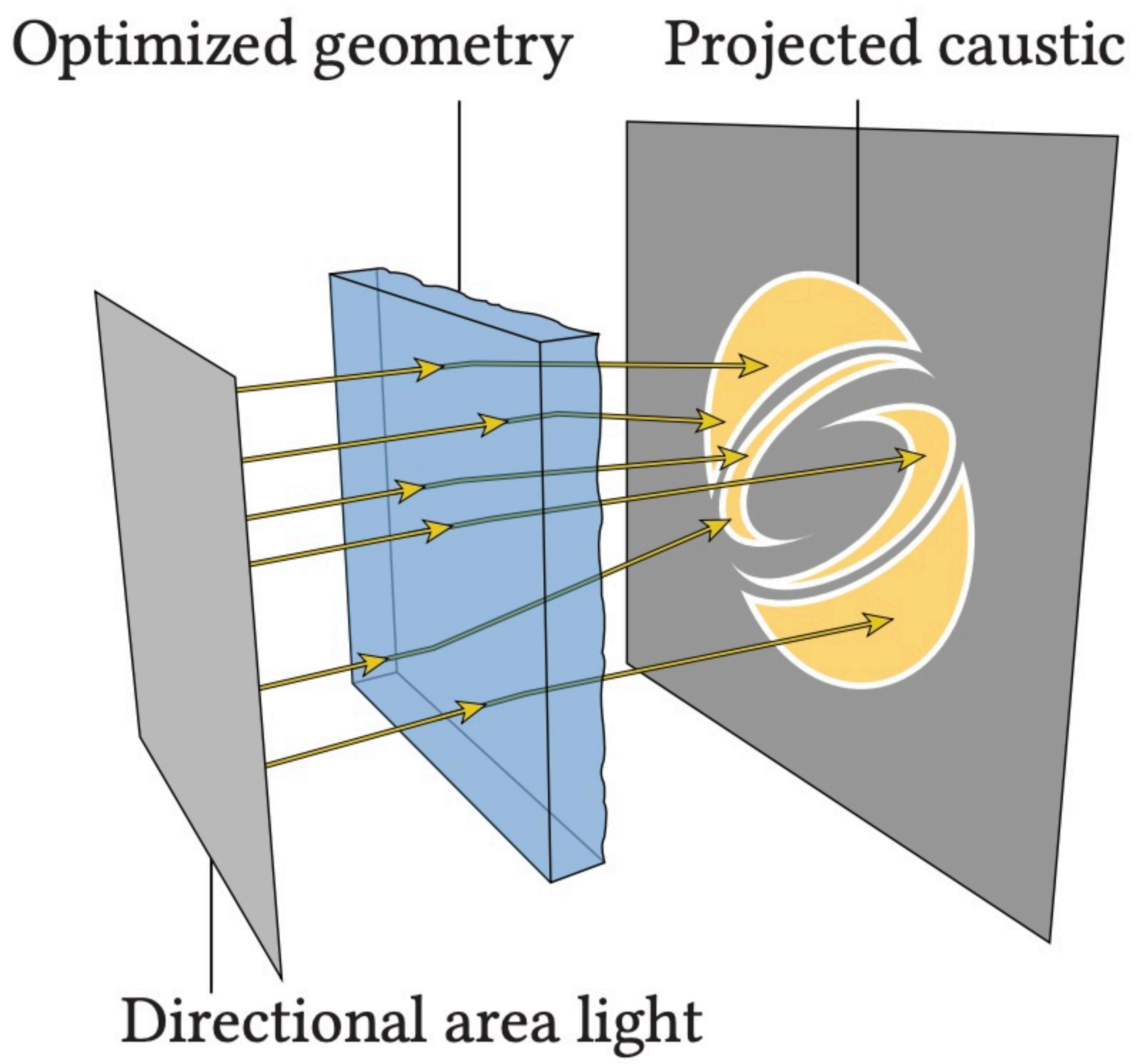
Initial guess (6.5k tris) Optimized parameters Our (6.5k tris) Reference (1.7M tris)

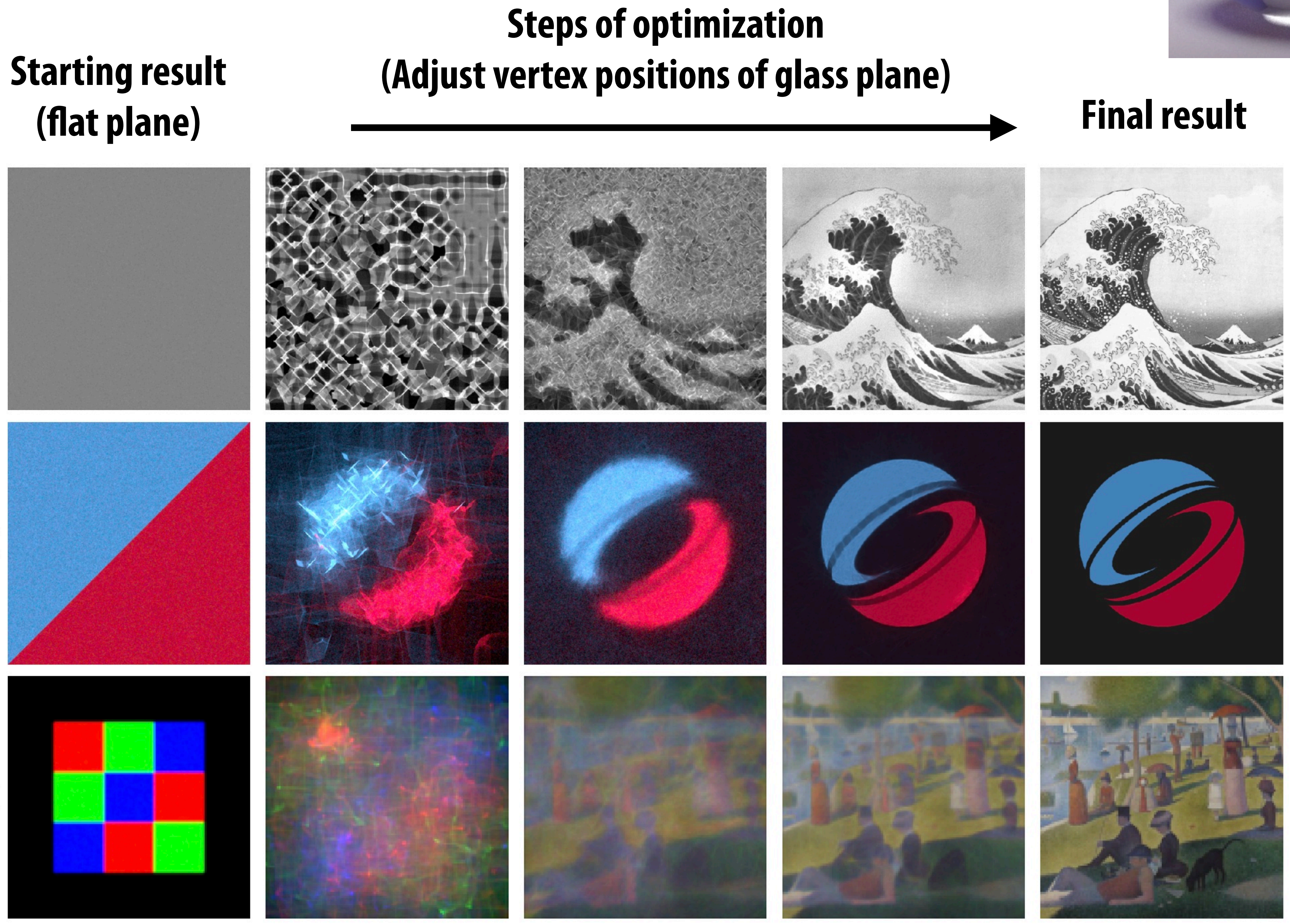
[Hasselgren et al. 2021 Appearance Driven automatic 3D Model Simplification]

Example uses of differential rasterizers/ray tracers

Optimize vertex positions so surface refracts light to make given image on a receiving plane.







[Nimier-David et al. 2019 - Mitsuba 2: A Retargetable Forward and Inverse Renderer]

Summary

- Renderers are “world simulators” that can use a variety of representations to model surfaces, materials, light, etc.

- Making those simulators differentiable opens up the possibility to invoke the amazing effectiveness of large-scale optimization to recover “good representations” by minimizing loss from a reference

- Depending on (1) task at hand (high-quality rendering, parameter recovery, scene editing, etc.) and (2) the properties of the scene you are trying to work with (complex foliage, smooth curves, fine scale hair/fur, flat walls) and (3) your storage/performance needs, different representations will be preferred.