Lecture 16:

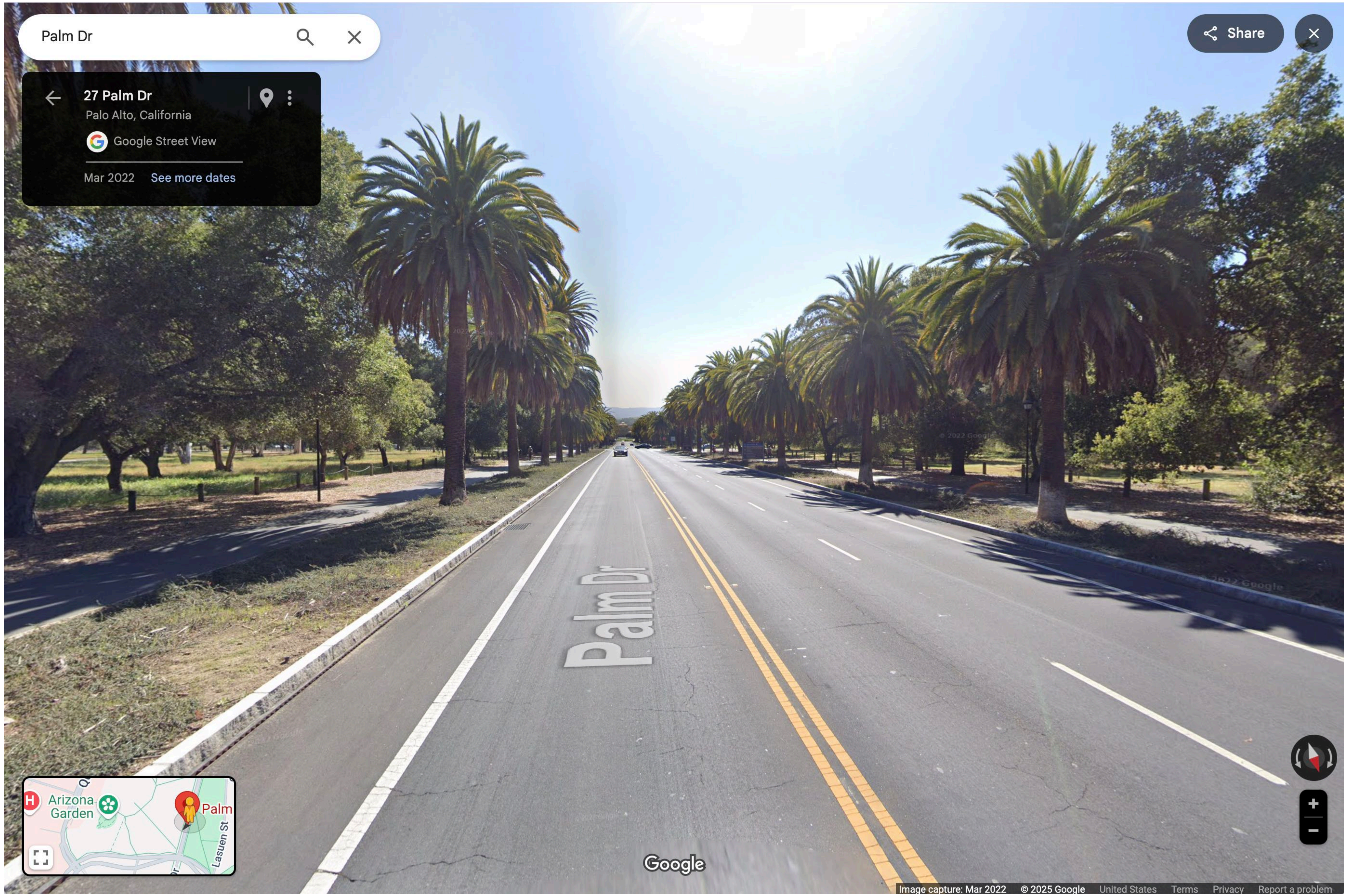
Recovering Scene Representations with Differentiable Rendering

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

A longstanding challenge in computer graphics…

- Acquiring high-quality 3D content for rendering
- Imagine I wanted to make a high-quality 3D model and associated texture maps depicting Josephine the graphics cat…

Google Street View

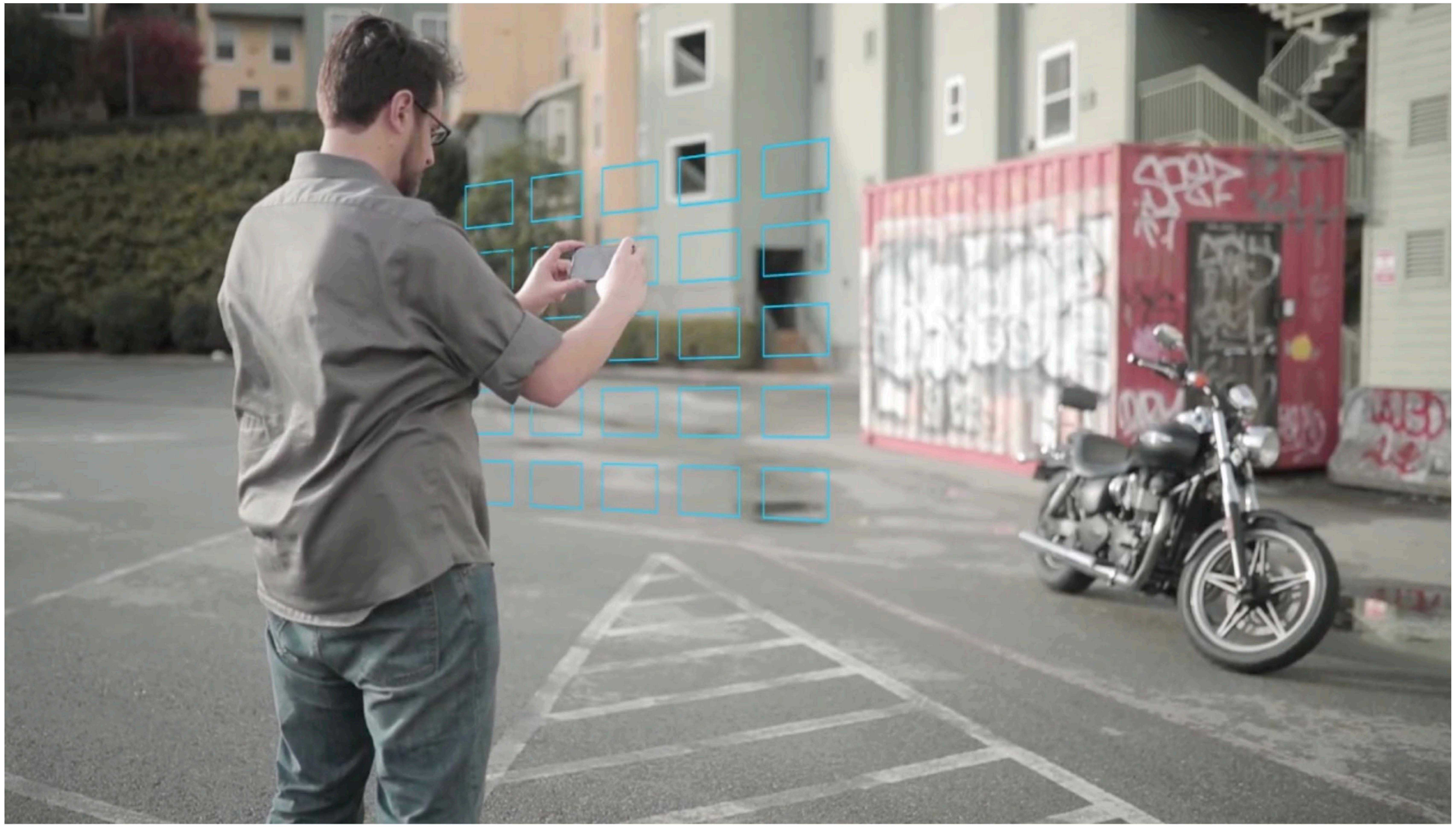


STOP
Turk St
Golden Gate Ave
Divisadero St
Webster St
Fell St

Credit: Tancik et al. 2022

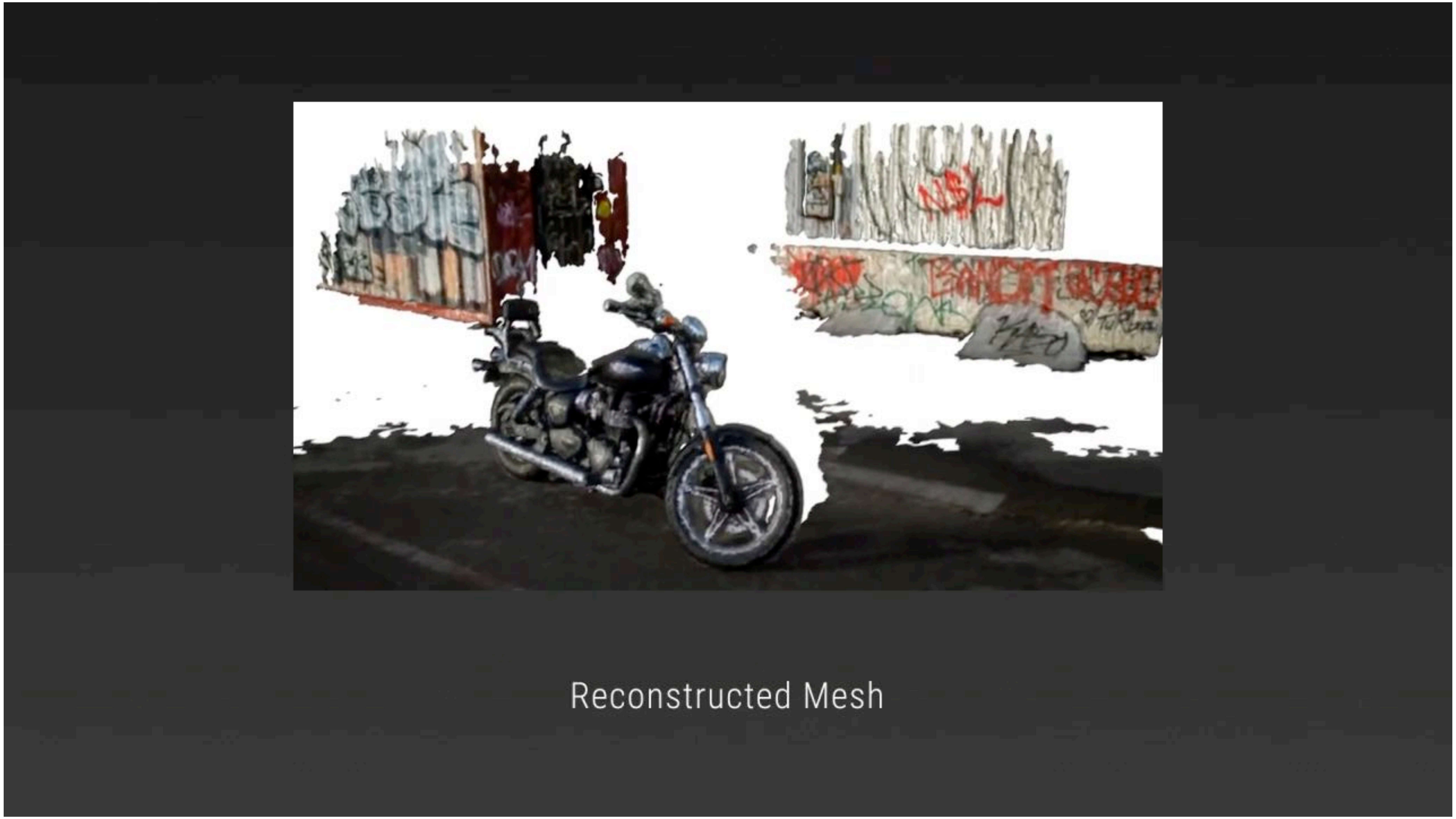
An interesting task

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



Credit: Mildenhall 2019

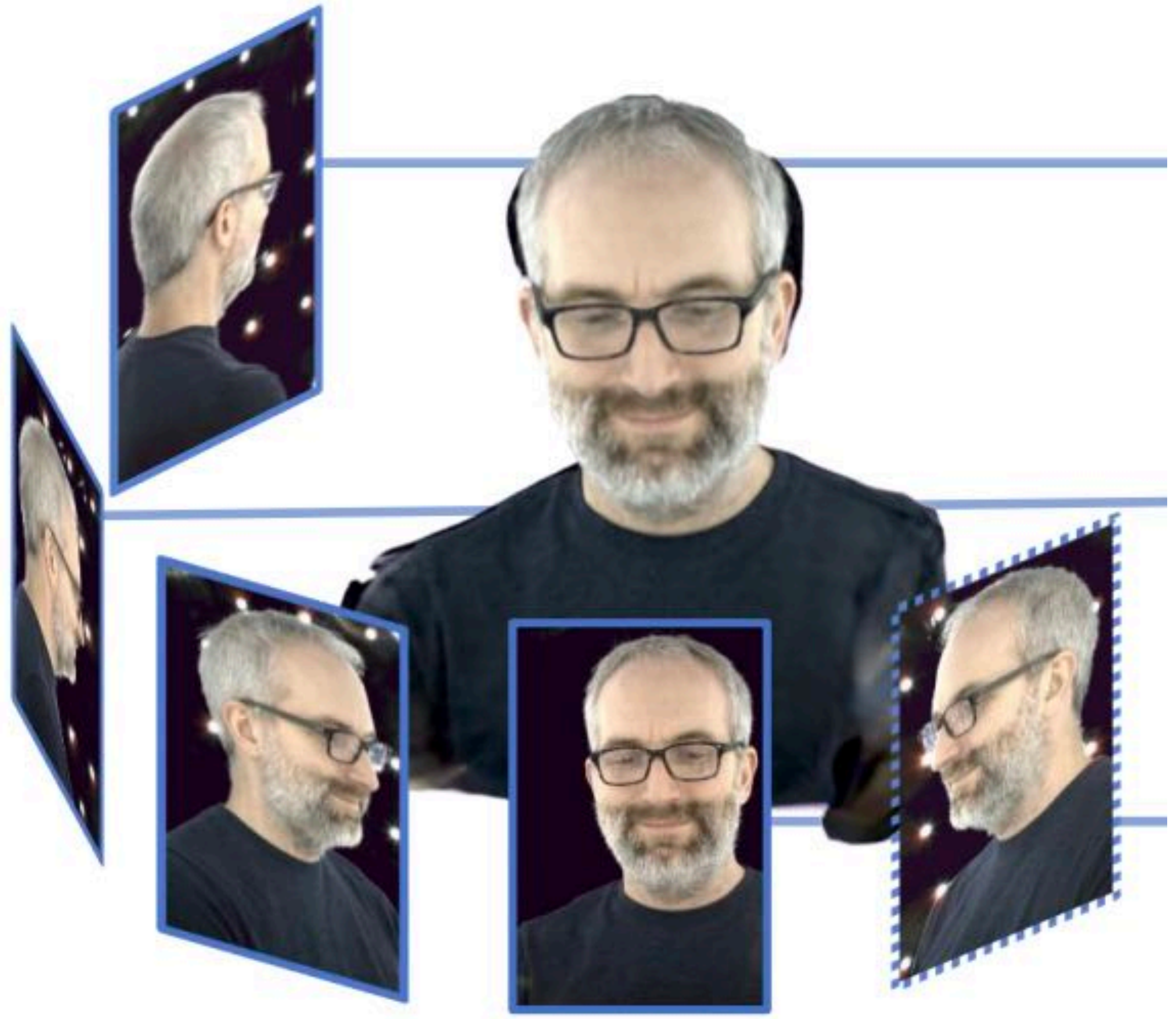
Estimating mesh geometry is tricky



Credit: Mildenhall 2019

Renewed interest in 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

Last time: rendering volumes

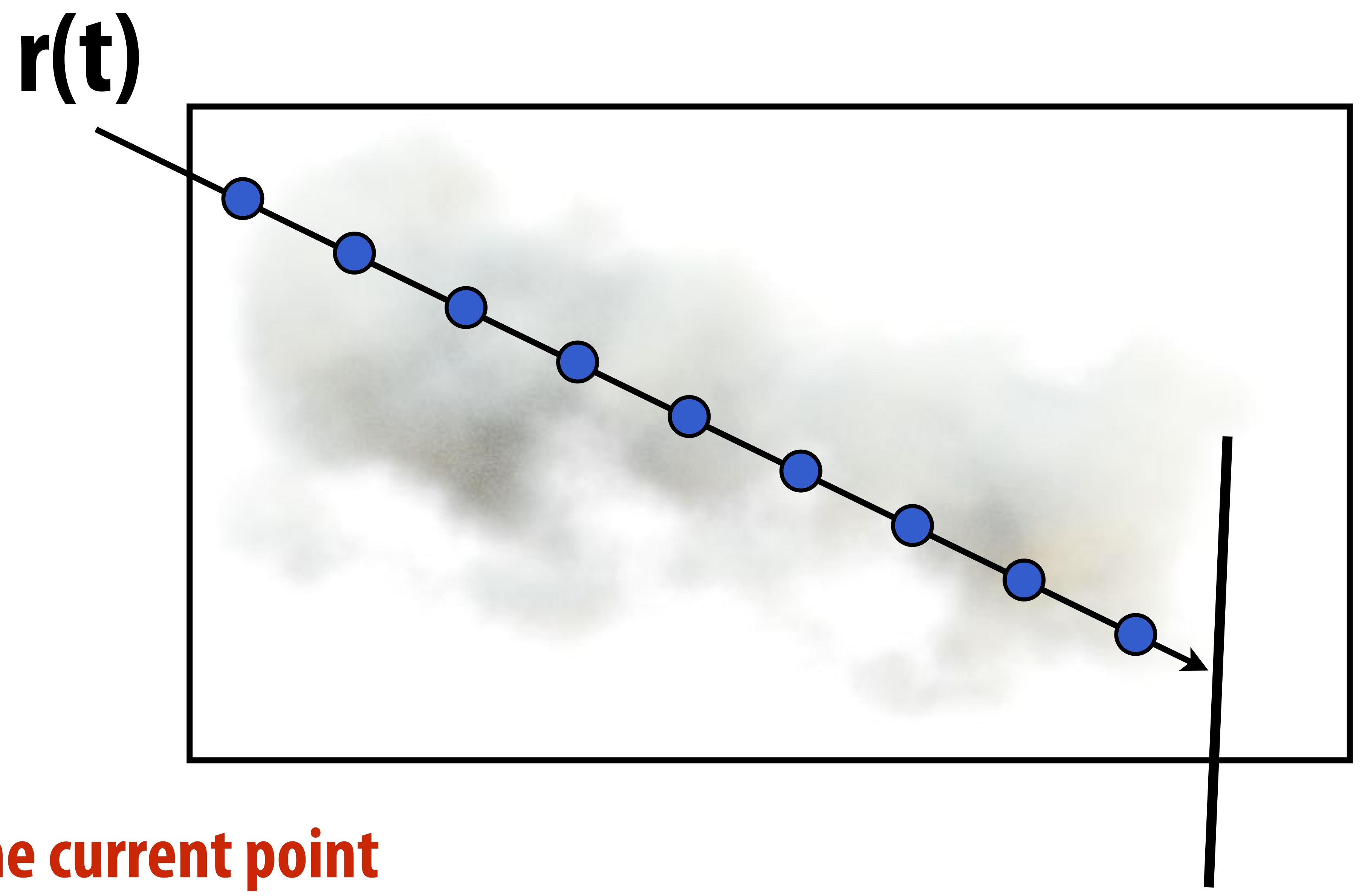
Given "camera ray" from point o in direction w….

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

And continuous volume with density and directional radiance.

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

$\longleftarrow$ Volume density and color at all points in space.



Step through the volume to compute radiance along the ray.

Attenuation of radiance along r between r(s) and "camera" due to out scattering or absorbion

Color, opacity of the volume at the current point (More precisely: radiance along r at point r(s) due to in-scattering or emission)

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

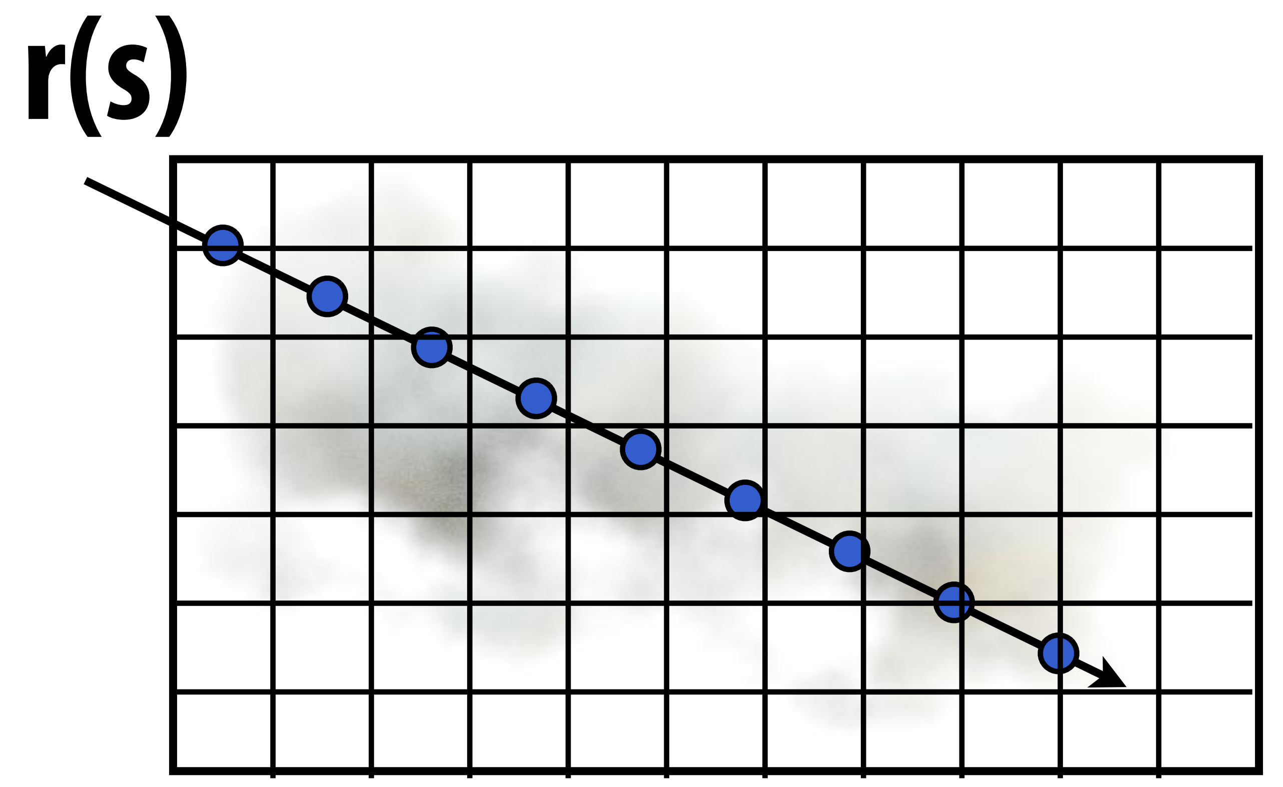
Recovering a volume that yields acquired images

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

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

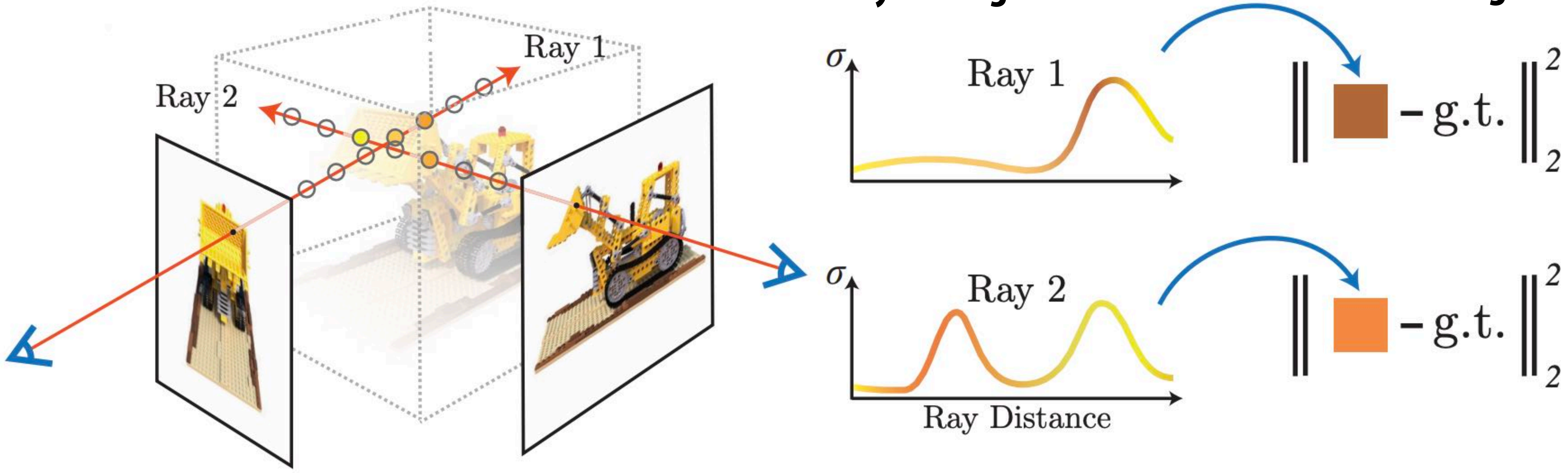
Idea: find volume parameters (opacity and color at each (i,j,k)
To make C(r) match the corresponding pixel in the photos.

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



Compute radiance along ray through volume

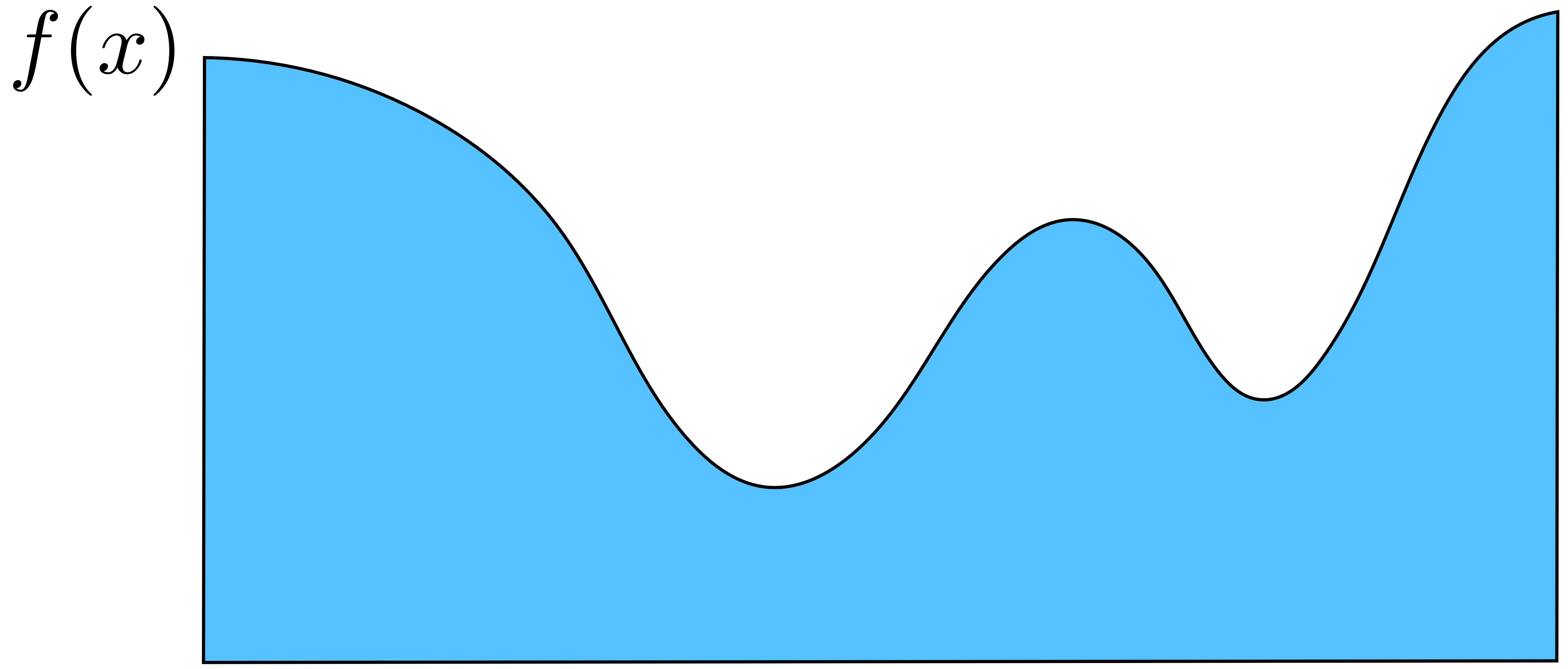
Compare to actual image



Mini intro to gradient-based optimization

Imagine we have a function $f(x)$

- How can we find the minimum of the function?

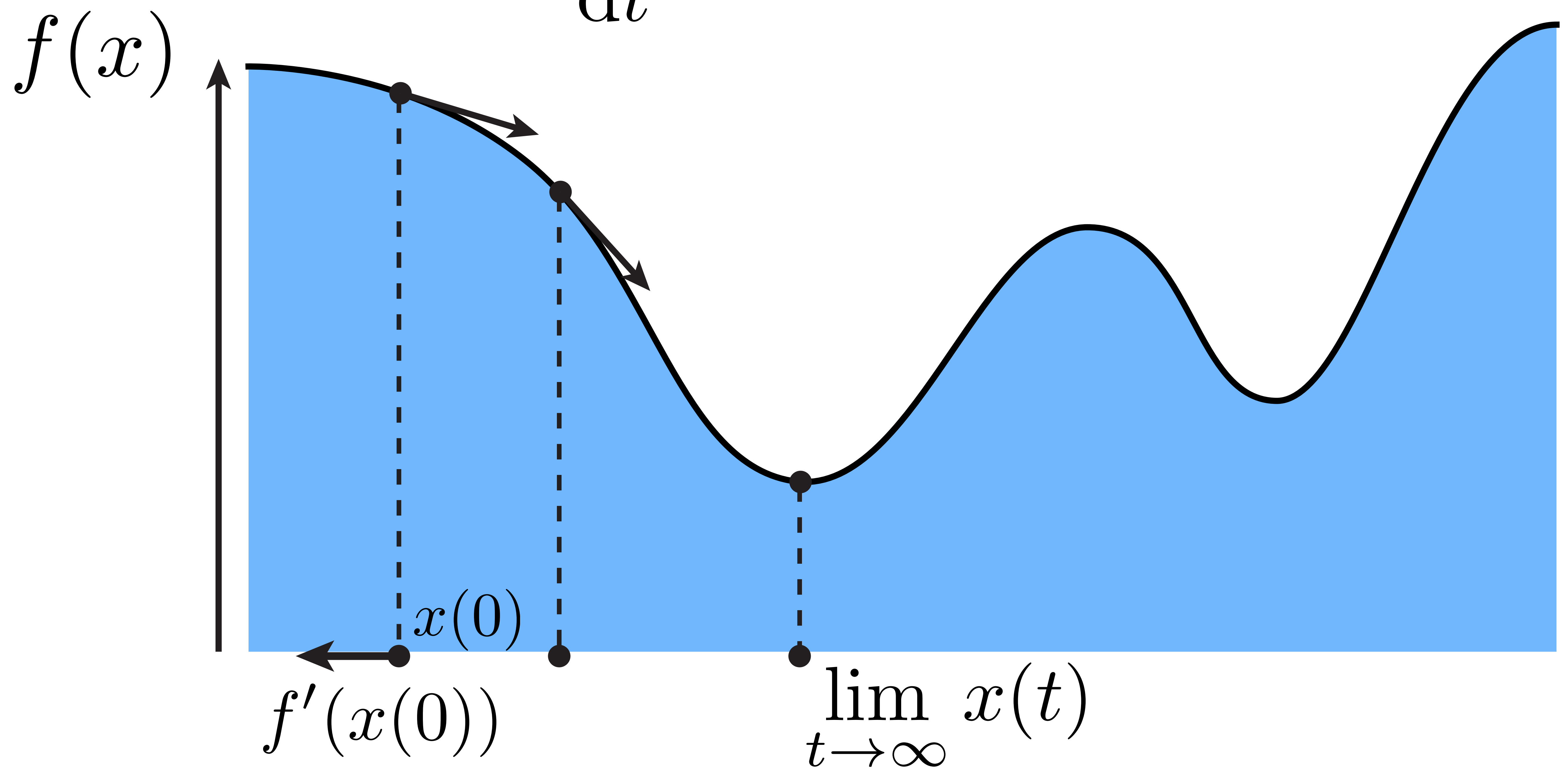


Descent methods

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)

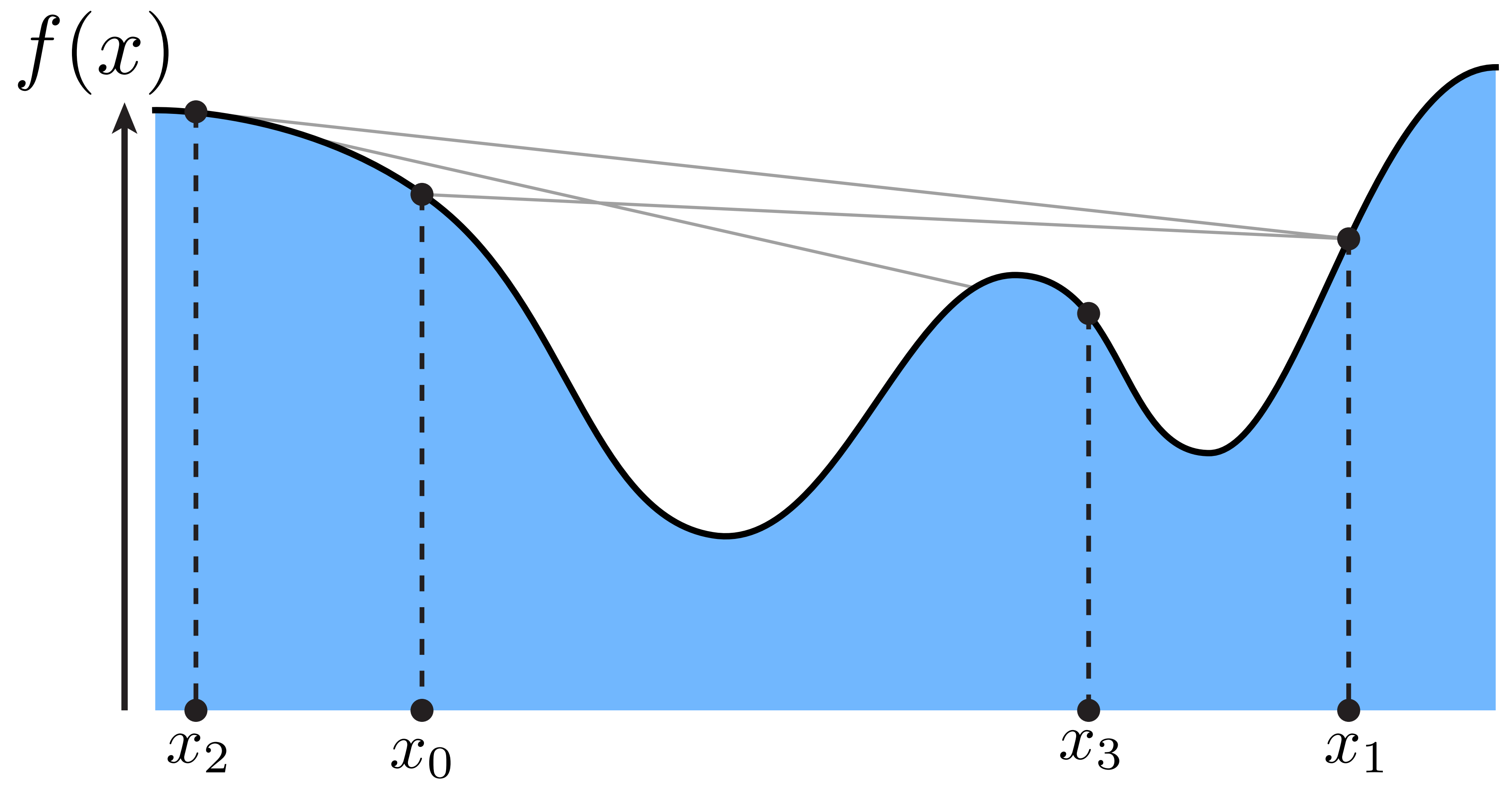
- "Walk downhill"
- One simple way: forward Euler:

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

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?**

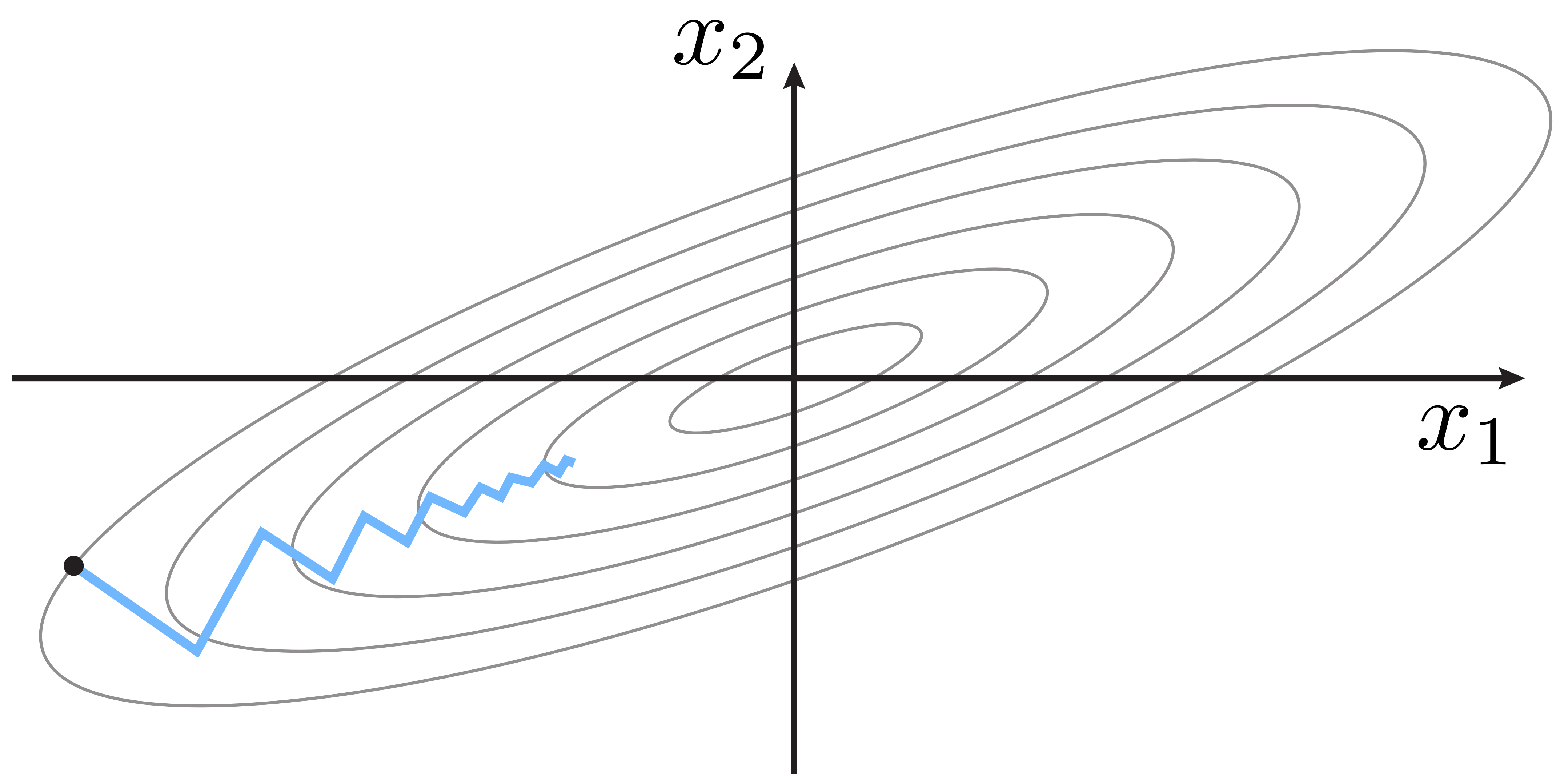
$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}(t) = -\nabla f(\mathbf{x}(t))$$

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

- **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**



Back to our problem of recovering a volume

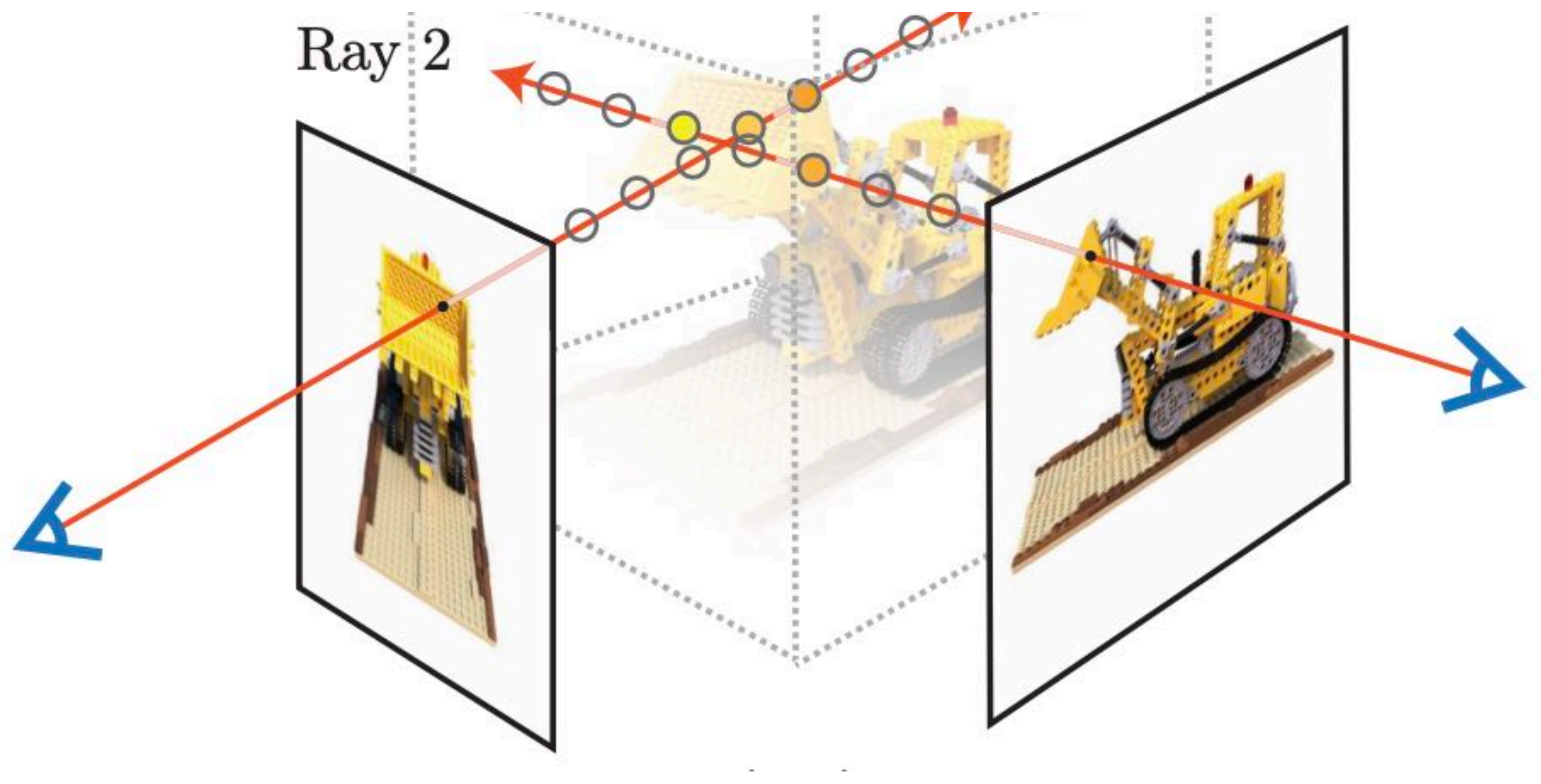
Recovering a volume that yields acquired images

Given a set of images of a subject with known camera positions…
What is the function we are trying to minimize?

$$f(\theta, \mathbf{r}) = \|C(\mathbf{r}) - I_j(x, y)\|_2^2$$

Where r is the ray corresponding to the center point of pixel (x,y) for a given image j, and C(r) is the color (radiance) of the scene observed along the ray given by marching r through the volume. (I'm using theta to denote color and opacity parameters of the volume)

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

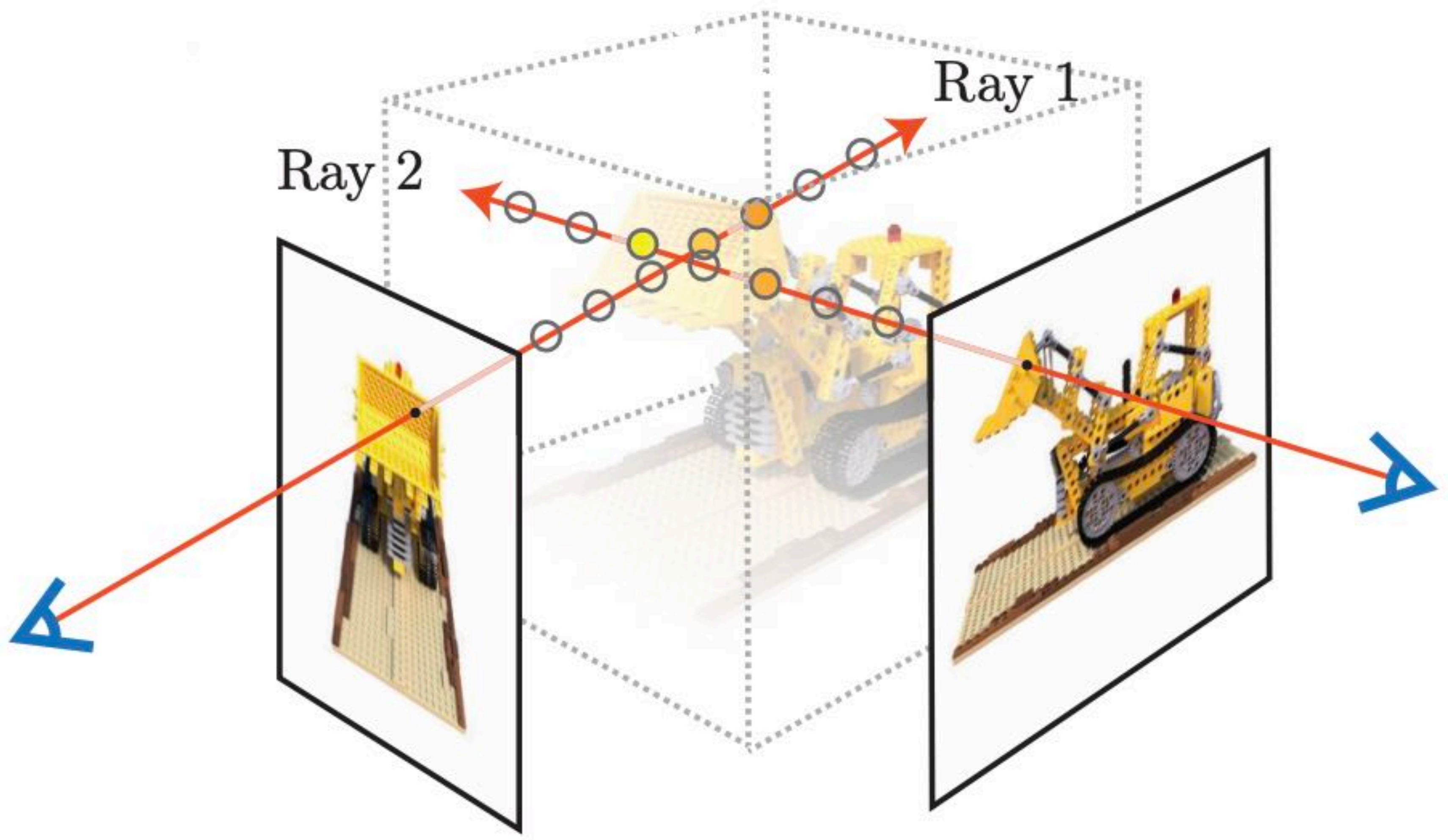


Notice that the rendering result $C(\mathbf{r})$ depends on the volume's color and opacity parameters.

So we want to minimize f subject to those parameters. Luckily f is easily differentiable! (It's a sum of exponentials!)

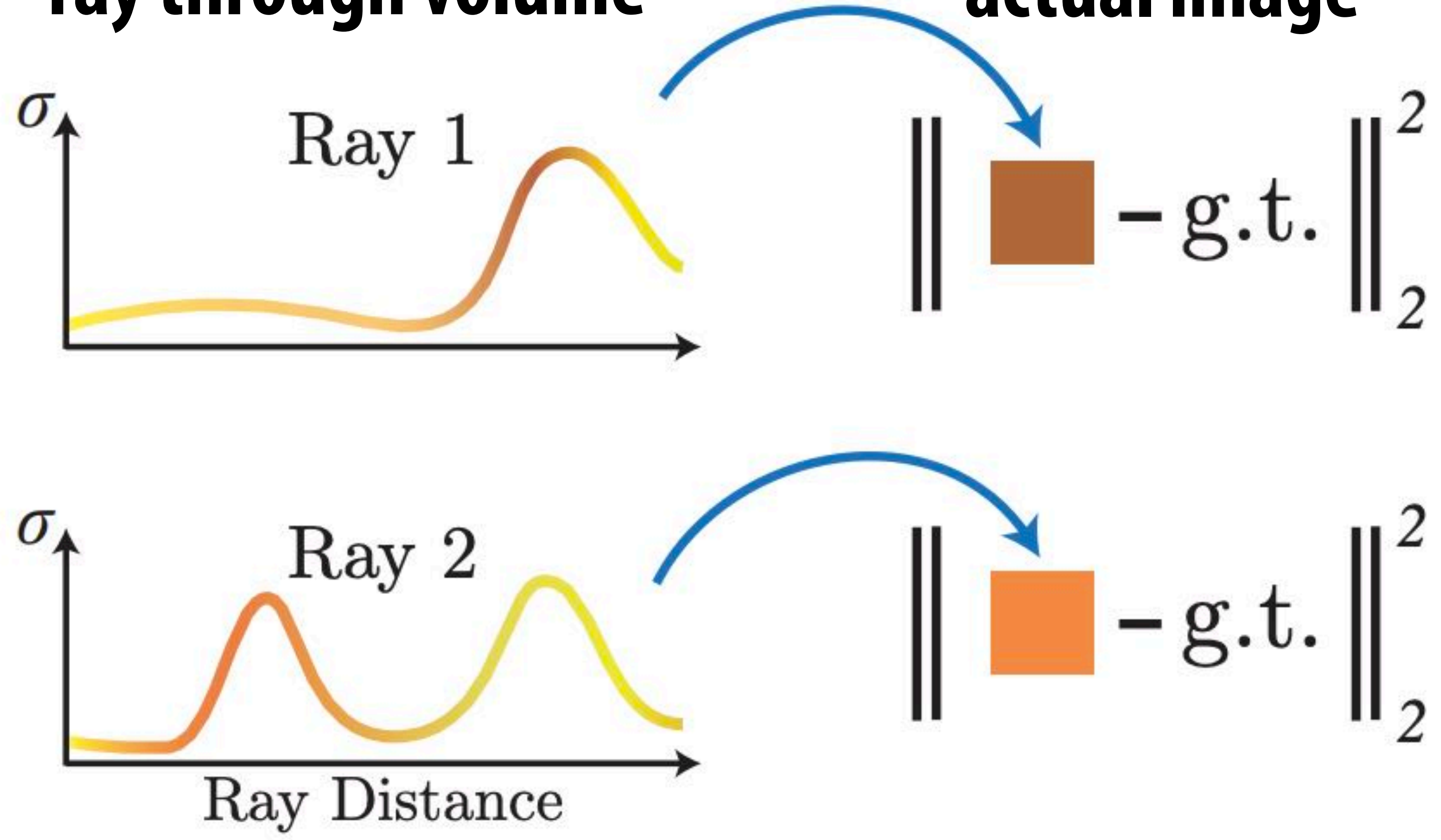
Recovering a volume that yields acquired images

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



Compute radiance along ray through volume

Compare to actual image



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

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

Problem: regular 3D grid representation of a volume has high storage cost

- **Dense 3D grid**
  - **V[i,j,k] = rgba**
  - **4096^3 cells ~ 128 GB**

- **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)

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 corresponds to renderings from various known viewpoints.

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

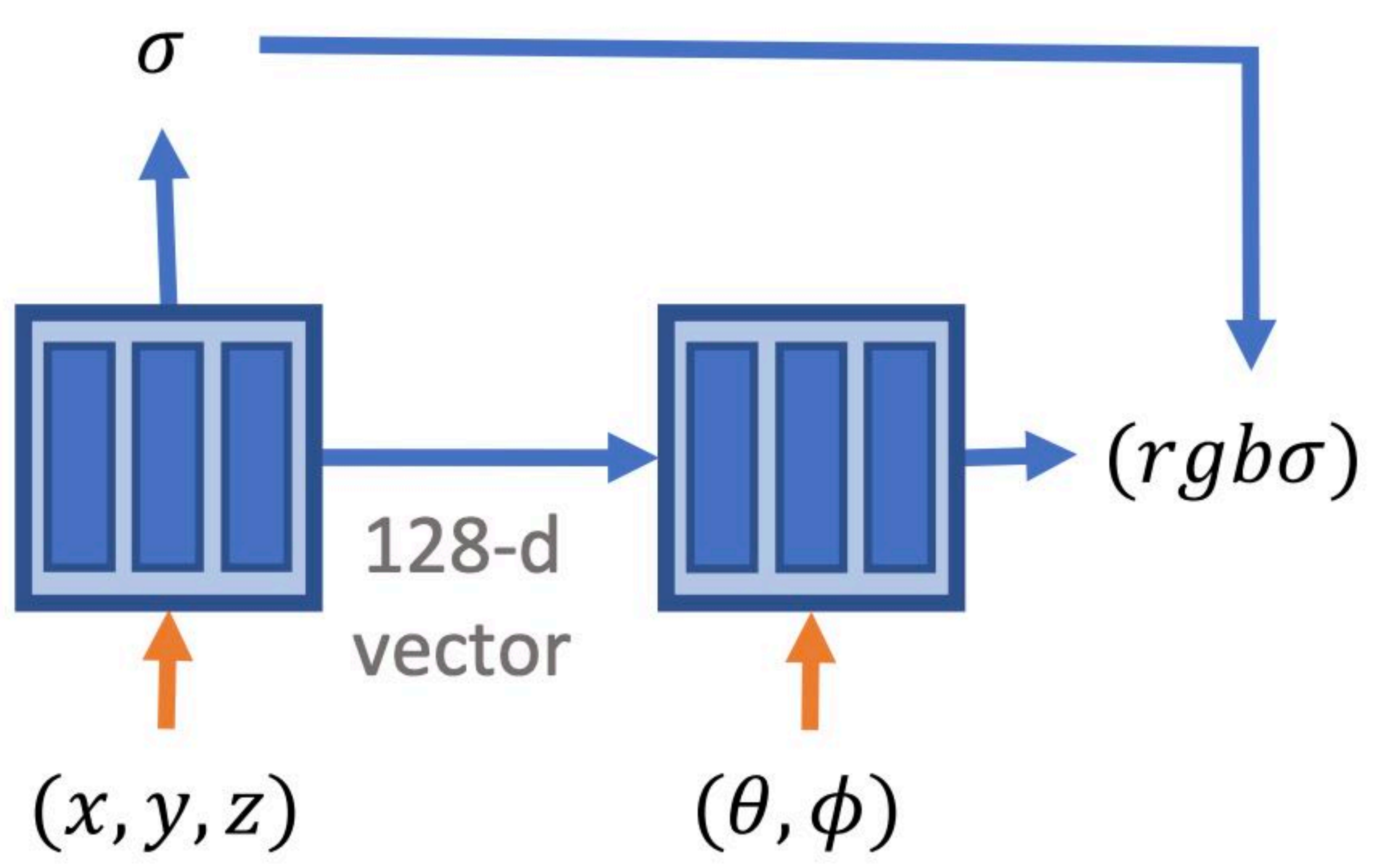
And that recovery process is optimization via gradient descent.
Technically… modern stochastic gradient descent (SGD).

Learning (compressed) representations

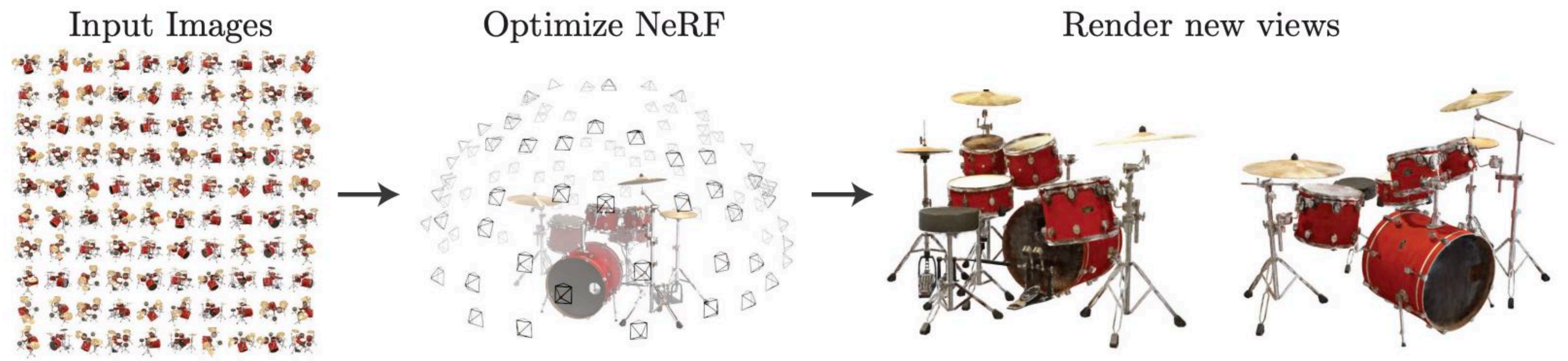
Rather than store an entire dense volume, let's just learn an approximation to the continuous function that matches observations from different viewpoints?

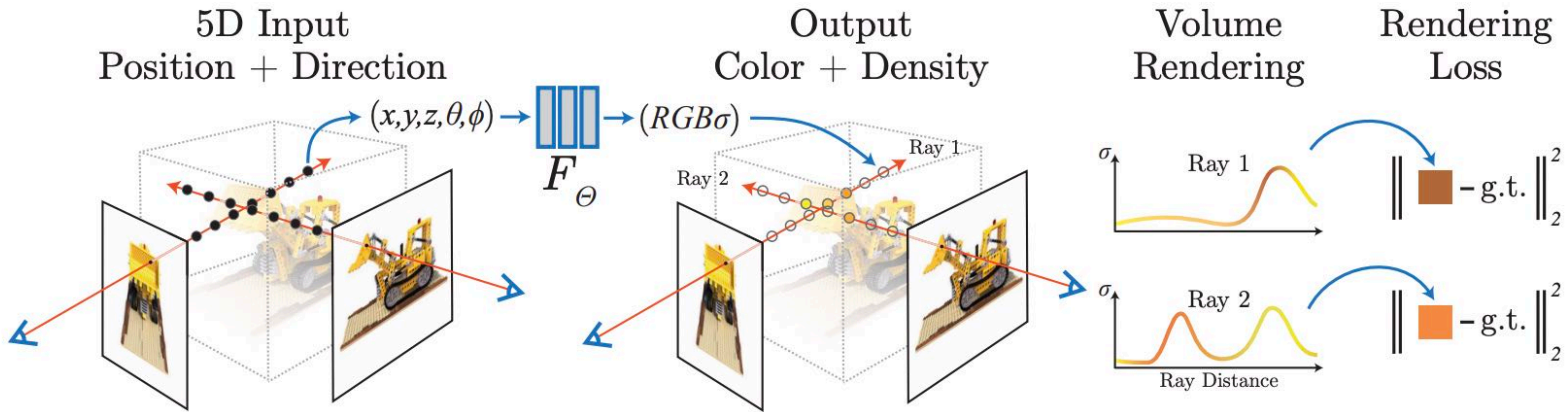
Let's represent that approximation using a deep neural network.

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



Learning neural radiance fields (NeRF)





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

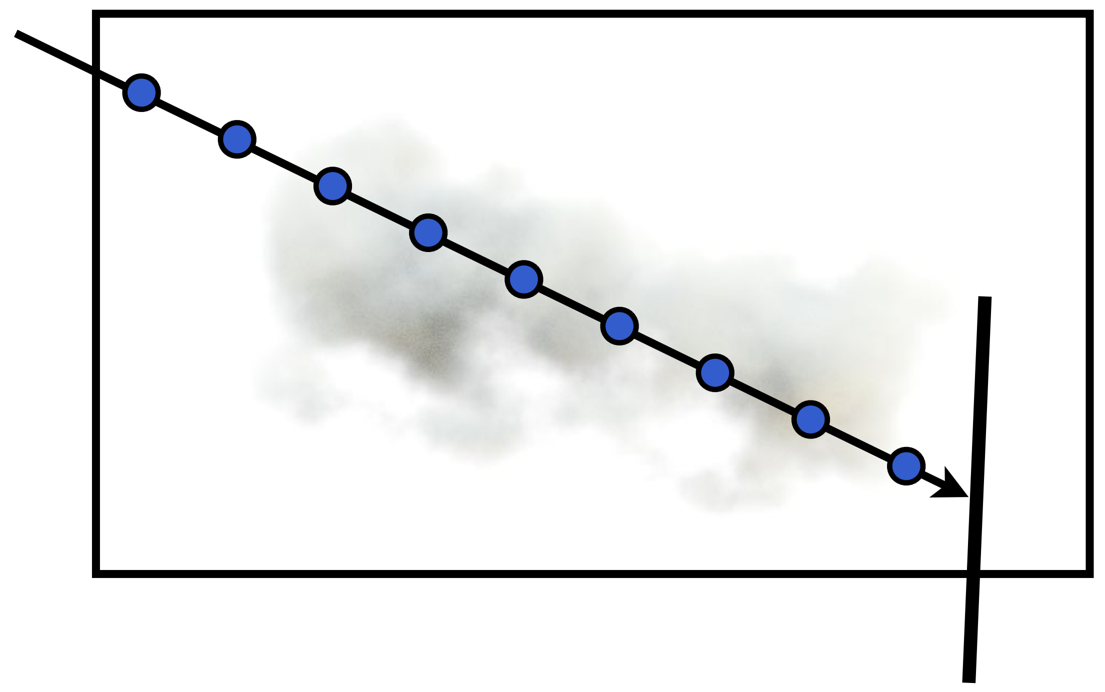
Great visual results!

Credit: Mildenhall 2023

What just happened?

- Continuous coordinate-based representation vs regular grid: DNN 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 possible (parameters fit on single GPU)
  - Amount 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

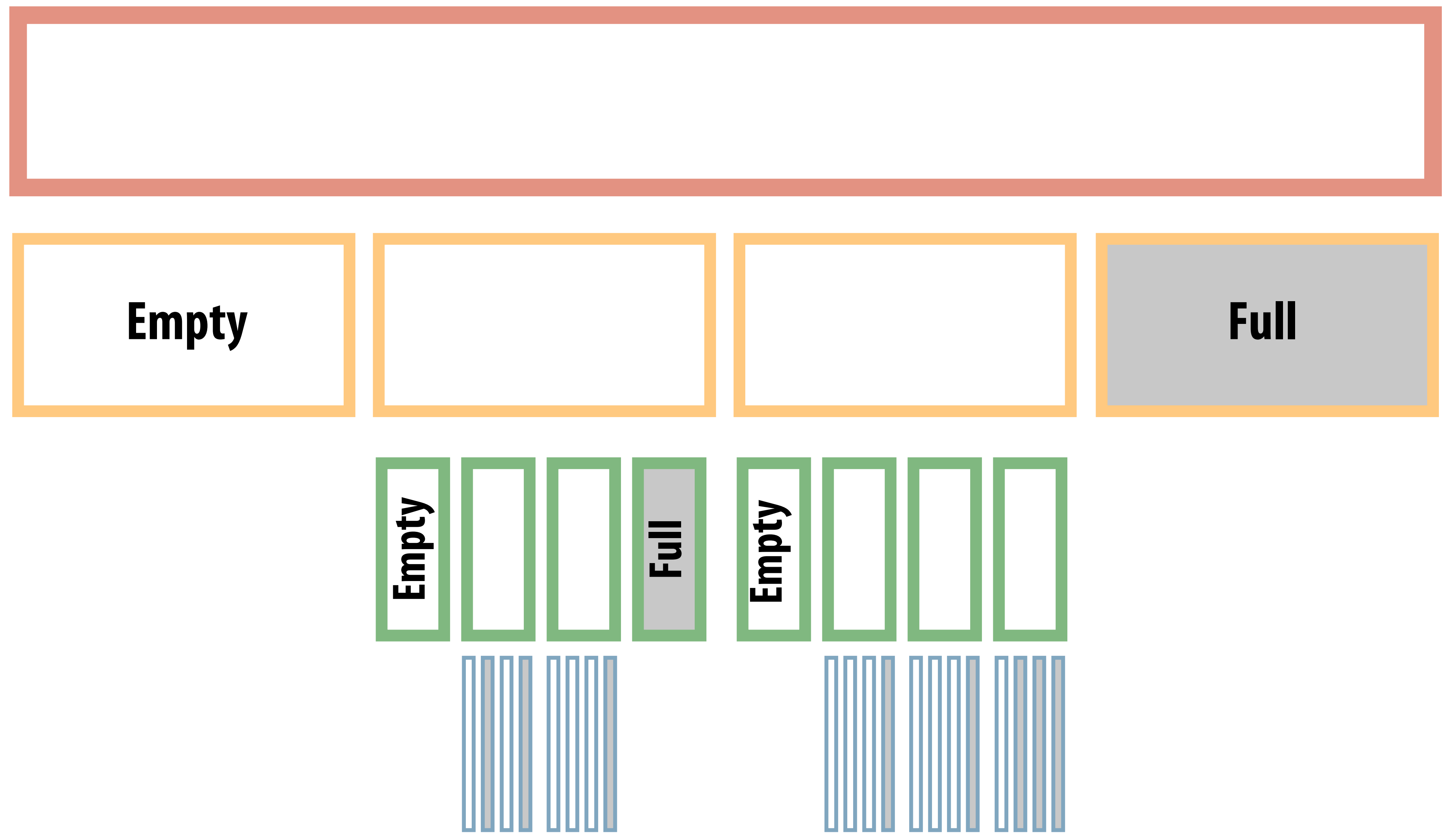
- NeRF was a great success is showing that IT WAS POSSIBLE to use brute force optimization + a differentiable volume renderer to recover a model of a scene.

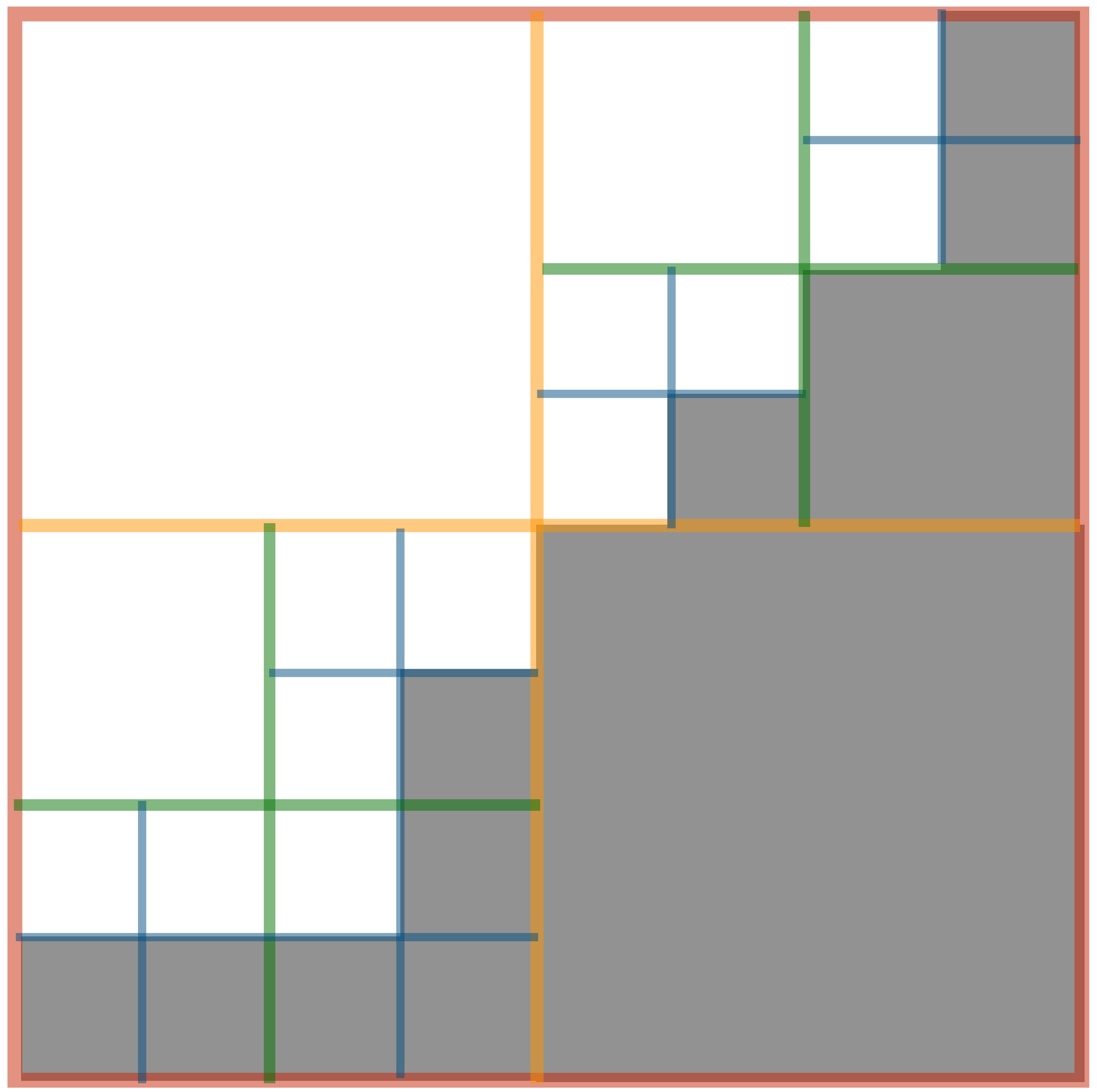
Improving rendering performance

- 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: quad-tree / octree data structures





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

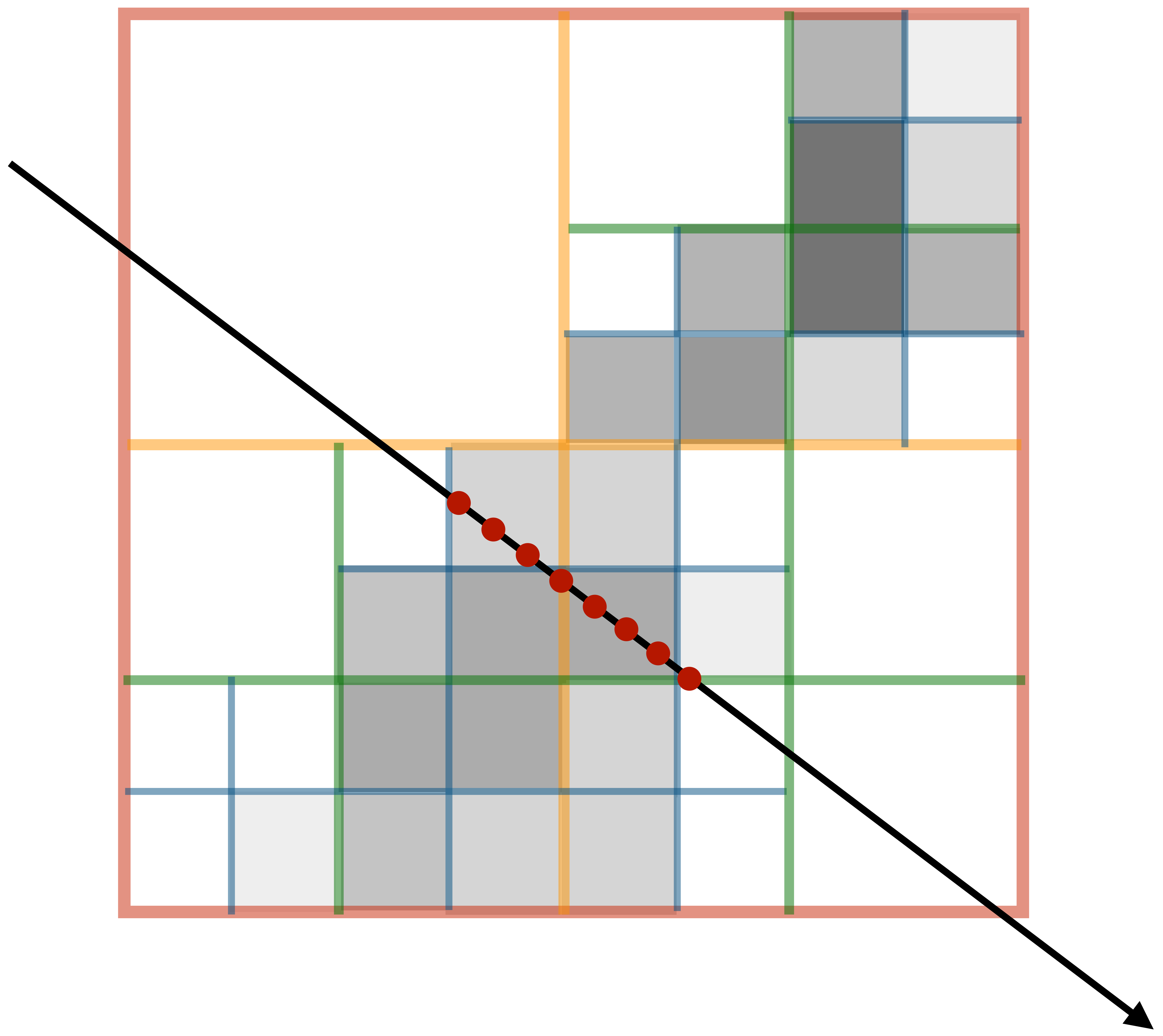
Interior nodes with no children → same "value" for all children in subtree

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

Recall: ray marching a sparse voxel grid

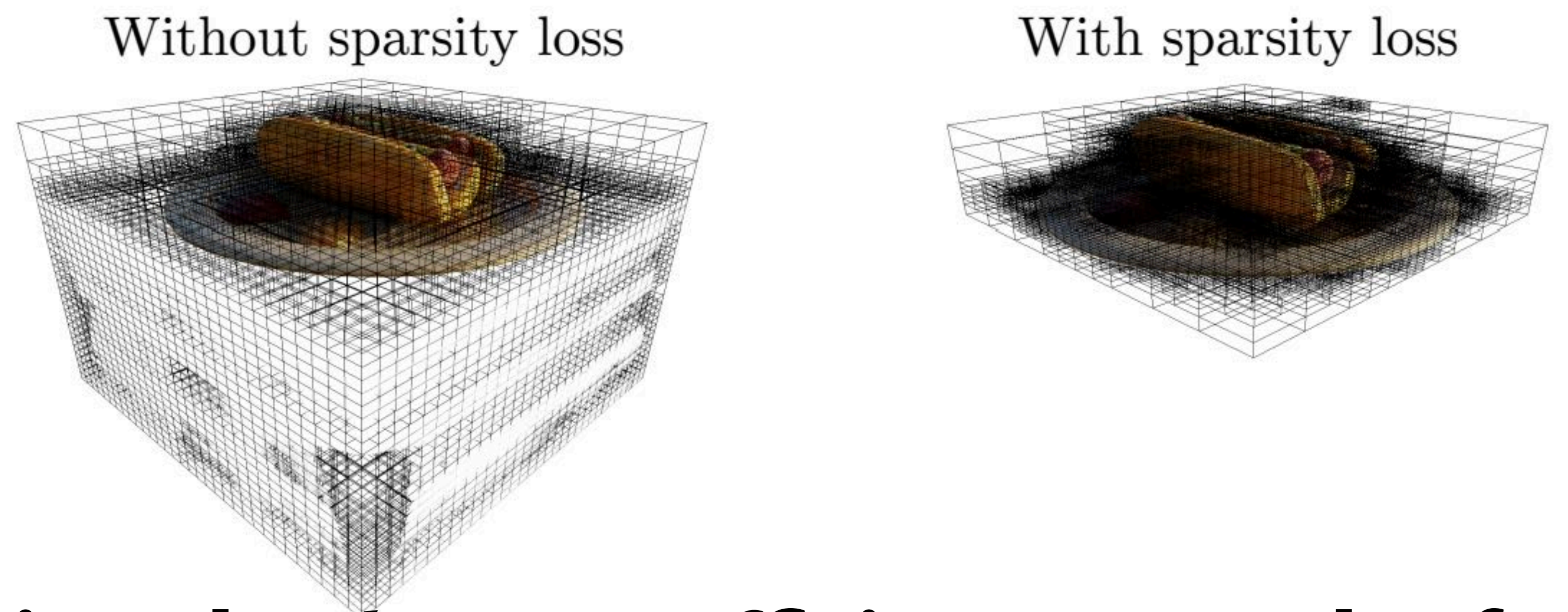
Ray can now "skip" through empty space

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

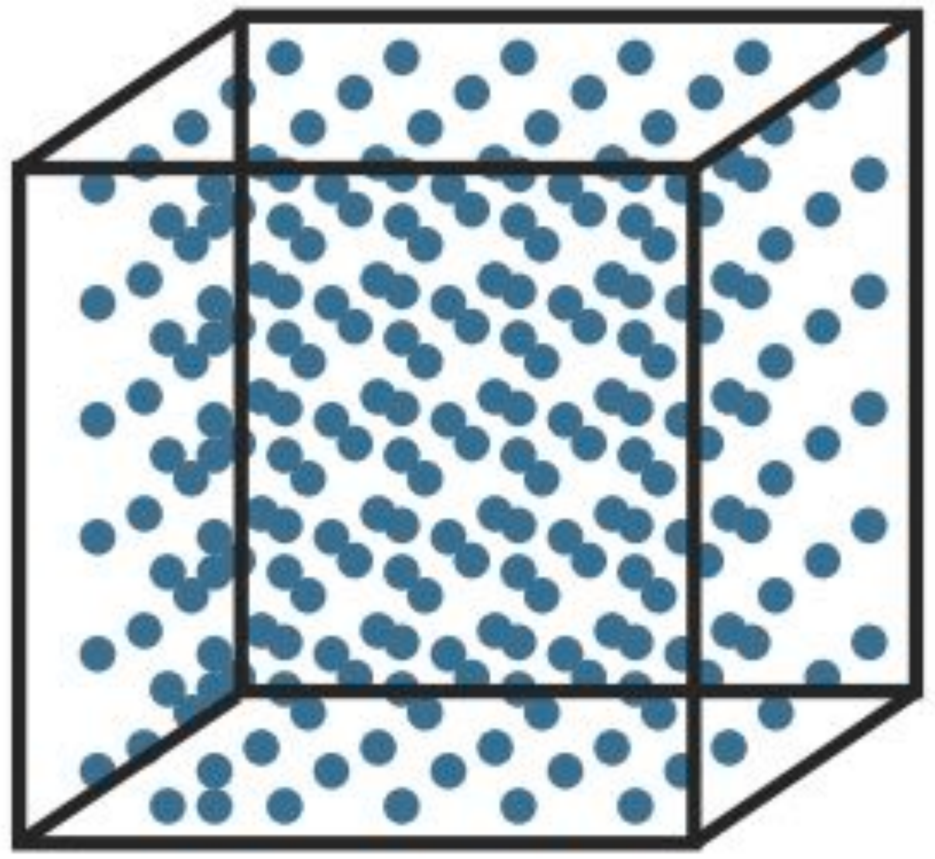


Let's just run NeRF optimization for a bit like before…

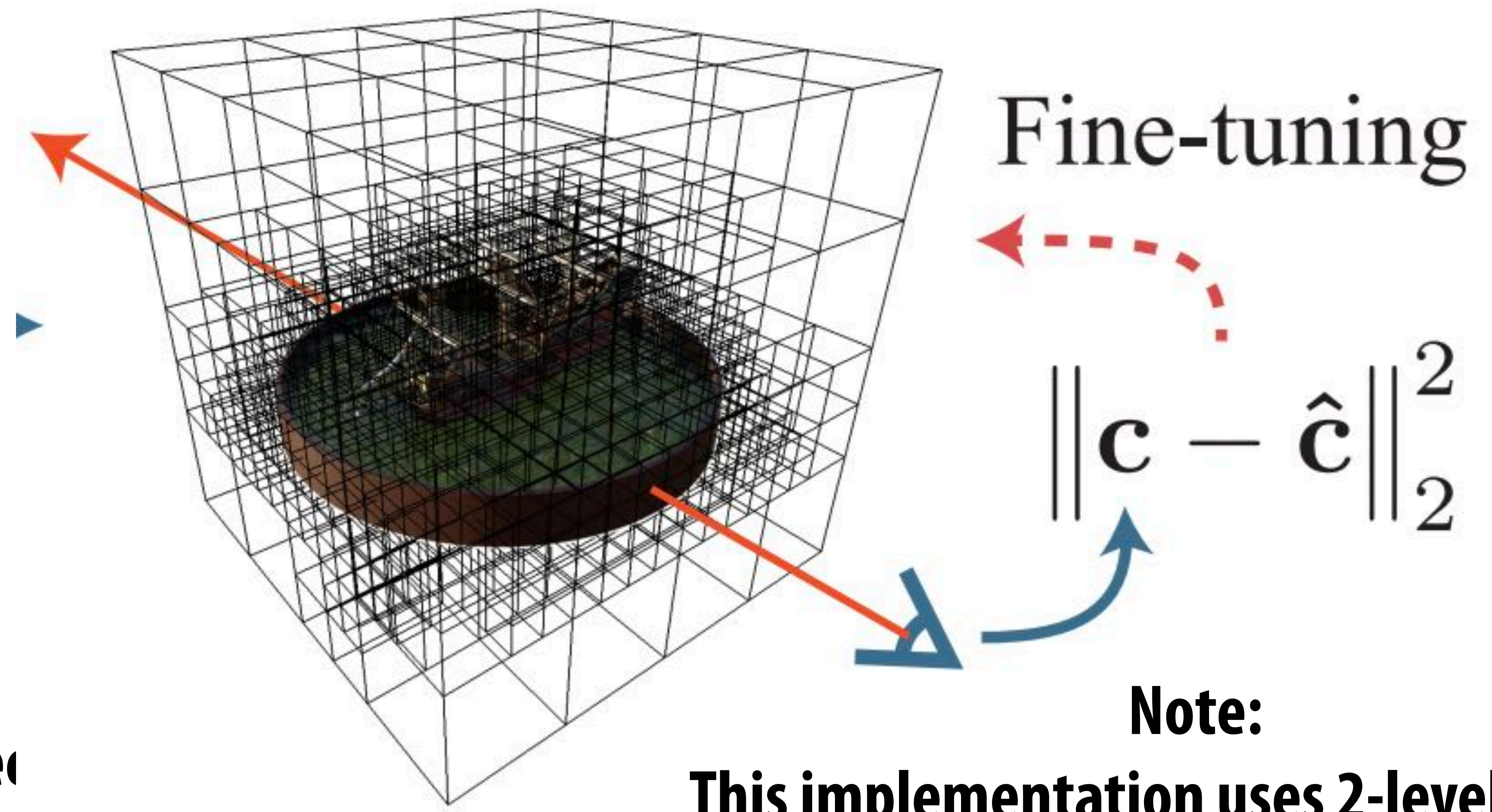
- Optimization will push some opacity values to 0
- DNN tells us where the empty space is!



- 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

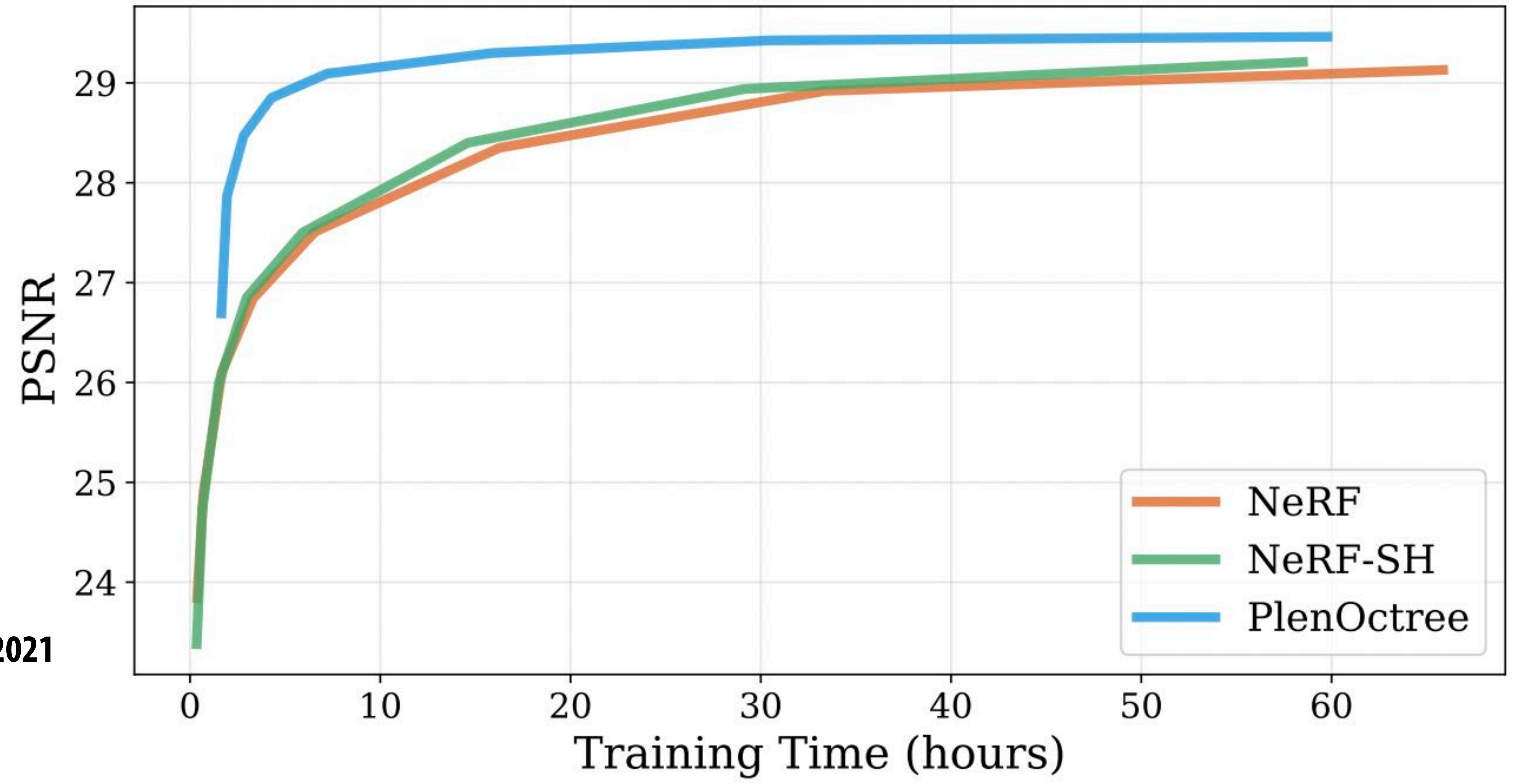


Note:
This implementation uses 2-level octreee

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

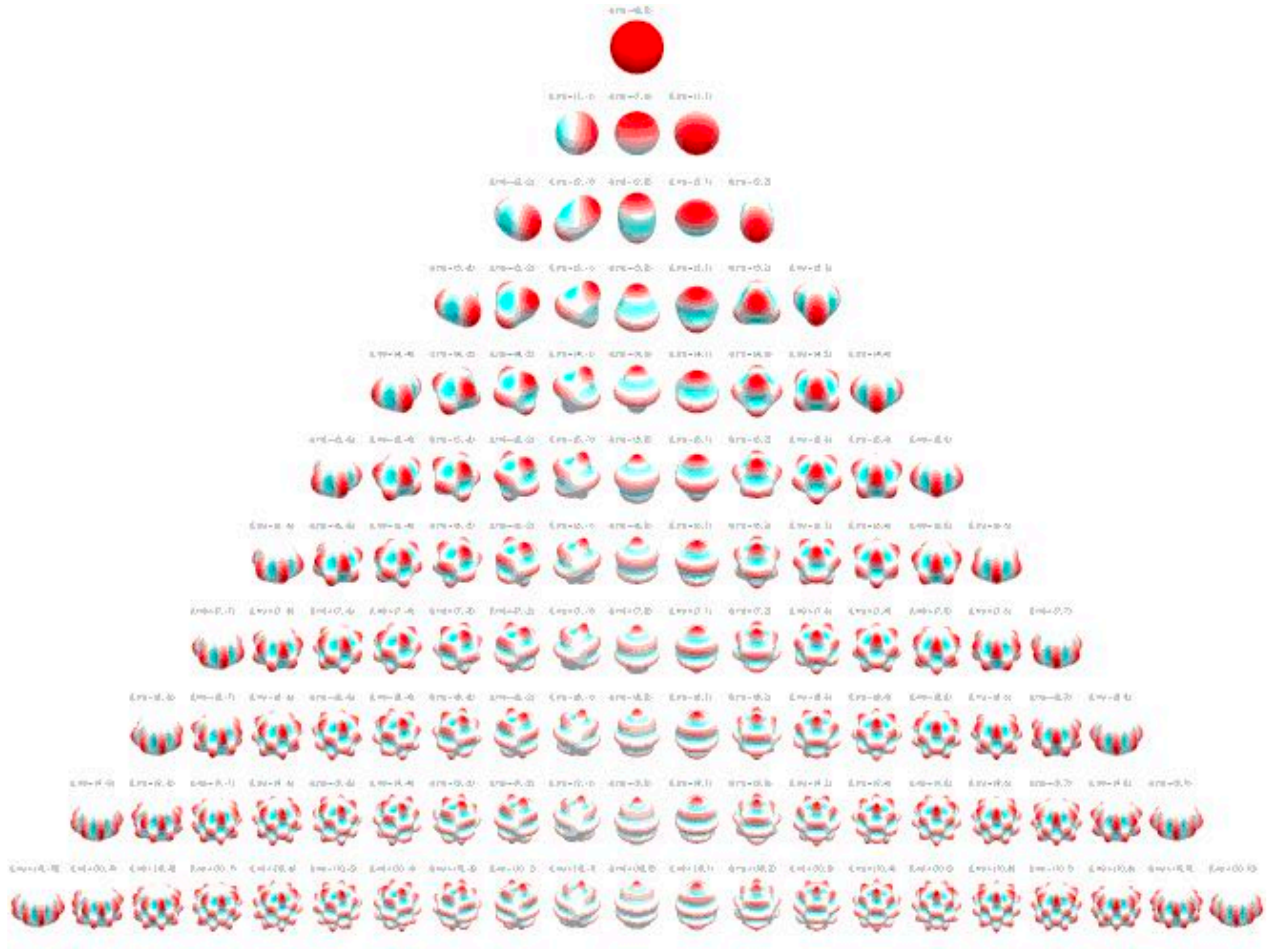
Credit: Yu 2021

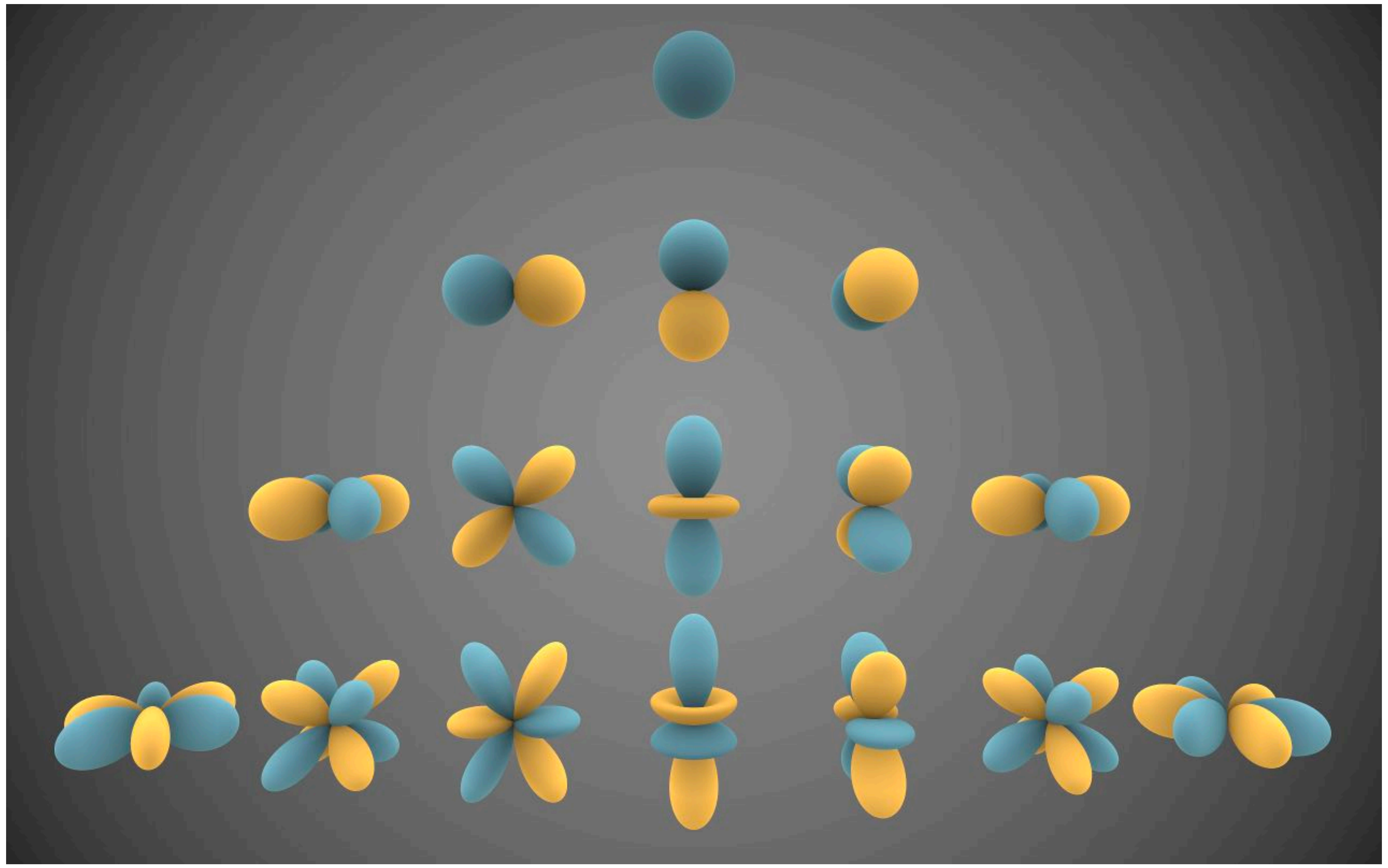


- Cost? Octree structure now 100's of MBs instead of a few MBs for MLP

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
wser
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
+ Add Component
Blu
Search Components
DDGIVolume1 (Instance)
DDGIVolumeComponent (Inherited)
Search Details
Transform
GI
DDGIVolume Compone
DDGIVolume Comp
GI

DDGIVolume1

Location X -612. 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

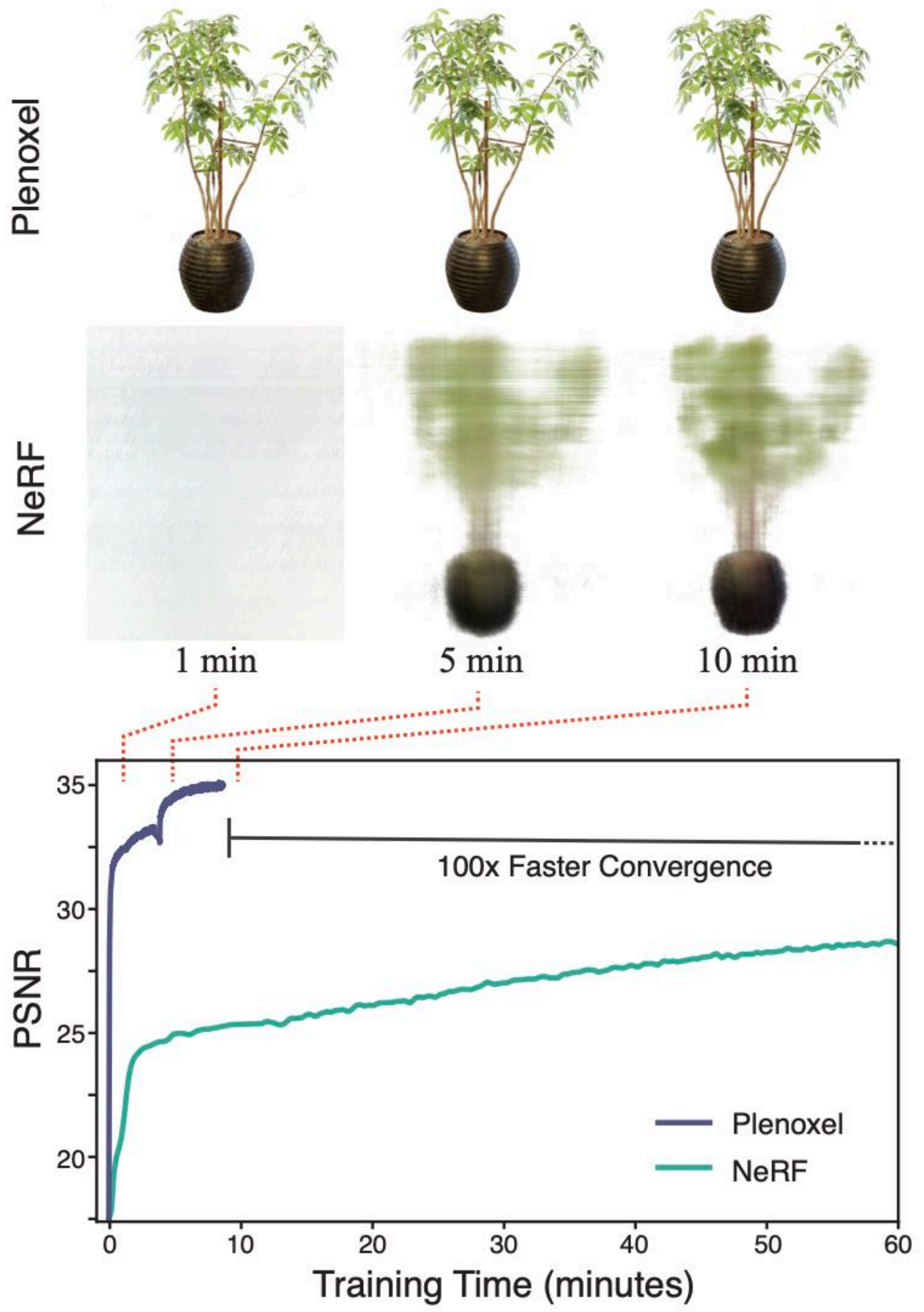
Probe Max Ray Di 100000.0

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***

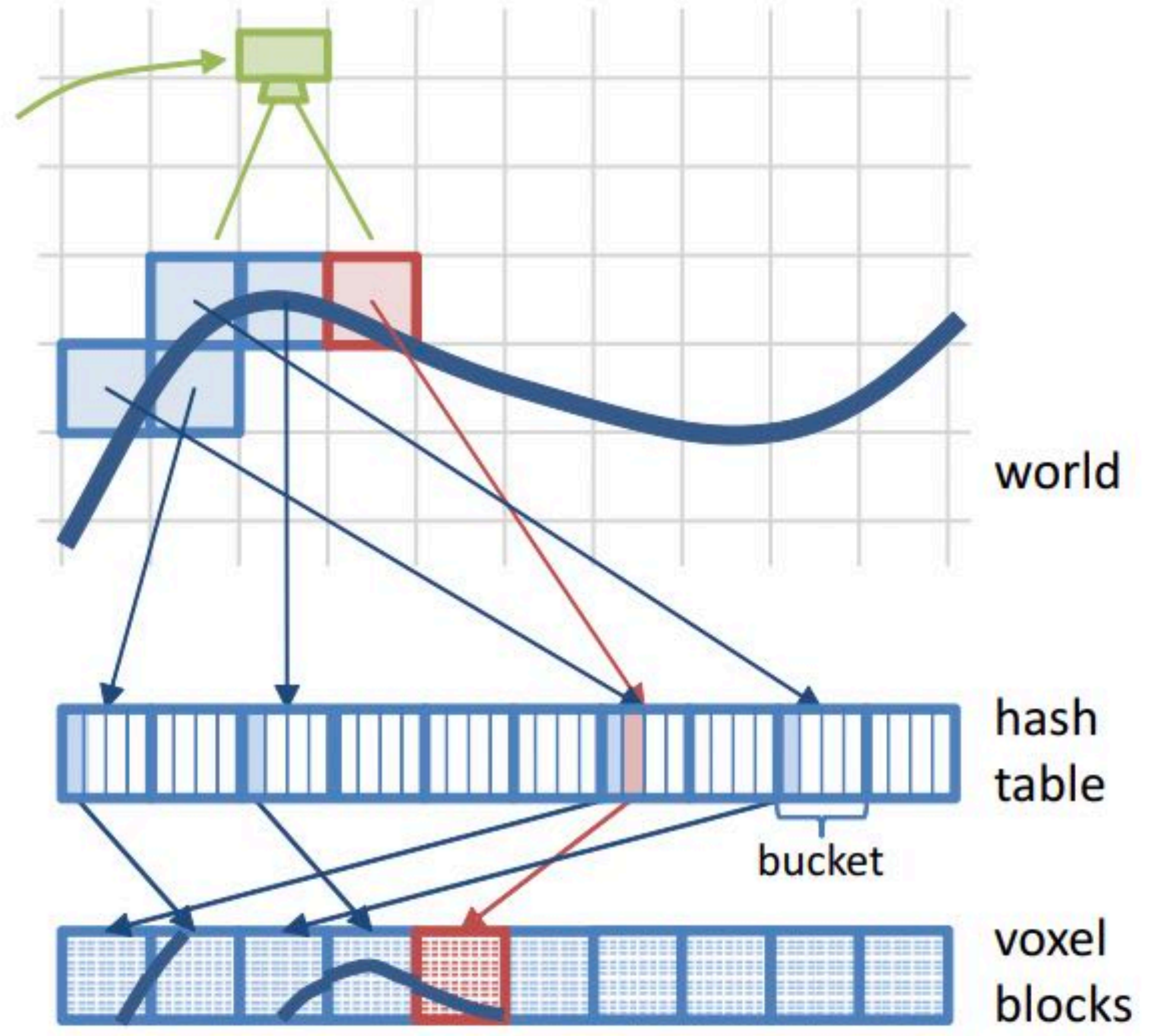
- **Ray march through the octree like normal**
  - **Instead of evaluating DNN_i(x,y,z,phi,theta) for node i corresponding to the current sample point, or evaluating SH coeffients to get radiance… retrieve the neural code z_i**
  - **Use a DNN to "decode" the code into a radiance or opacity**

- **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:

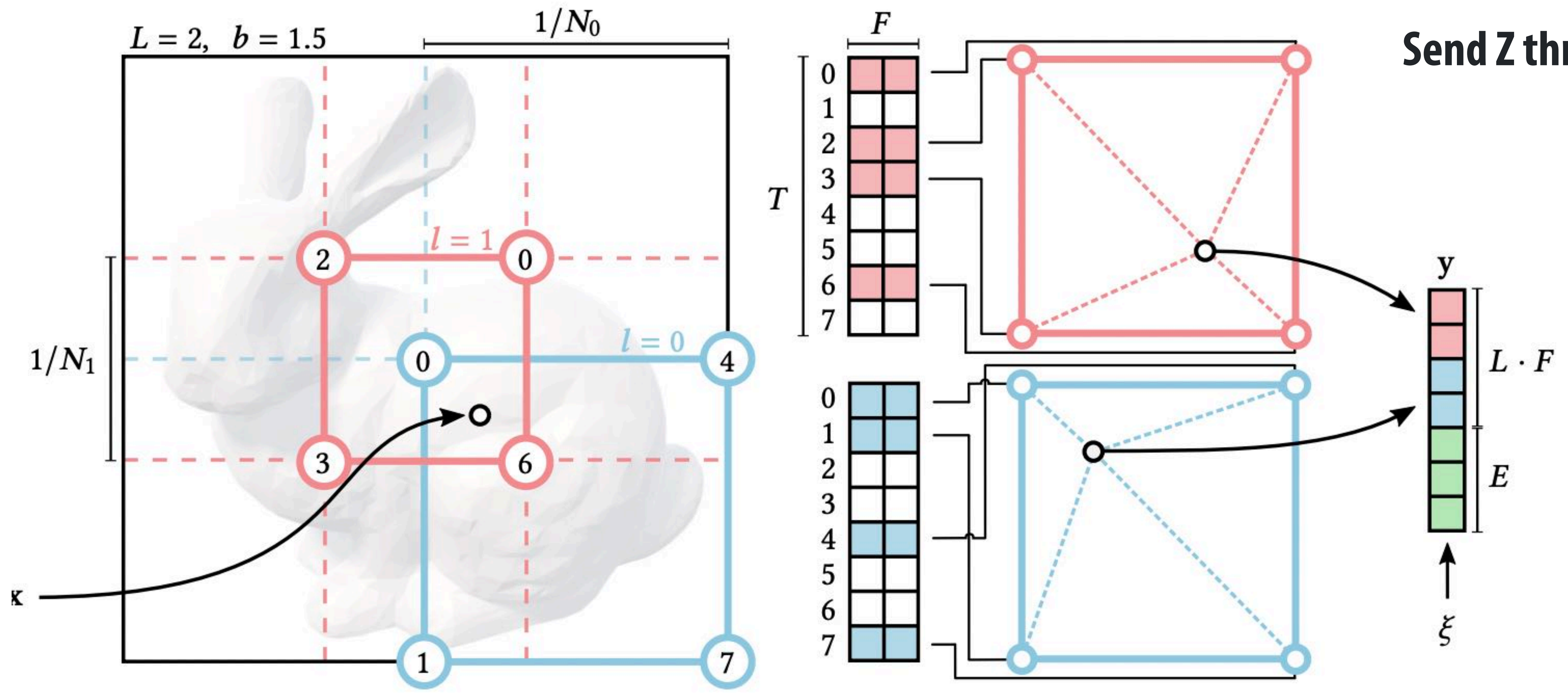
Compute indices of cell containing P on a bunch of different resolution grids (L grids)

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)**

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

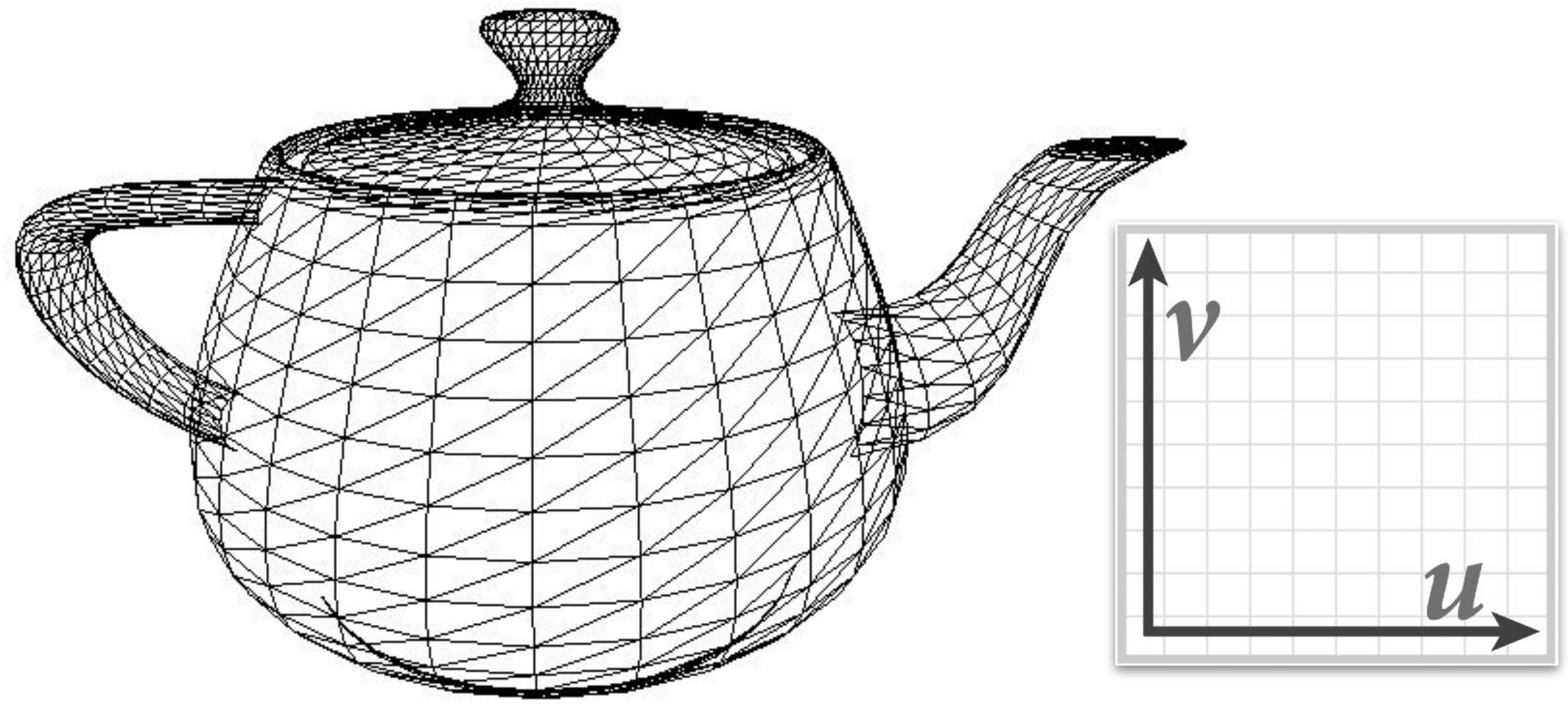
- **Then move to a traditional sparse encoding of occupancy (sparse volumetric structure)**
  - **Now the "topology" of the irregular data structure is fixed**
  - **Representation of surface/appearance/etc is stored at the nodes of this structure (spherical harmonics, neural code, etc.)**
  - **Most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

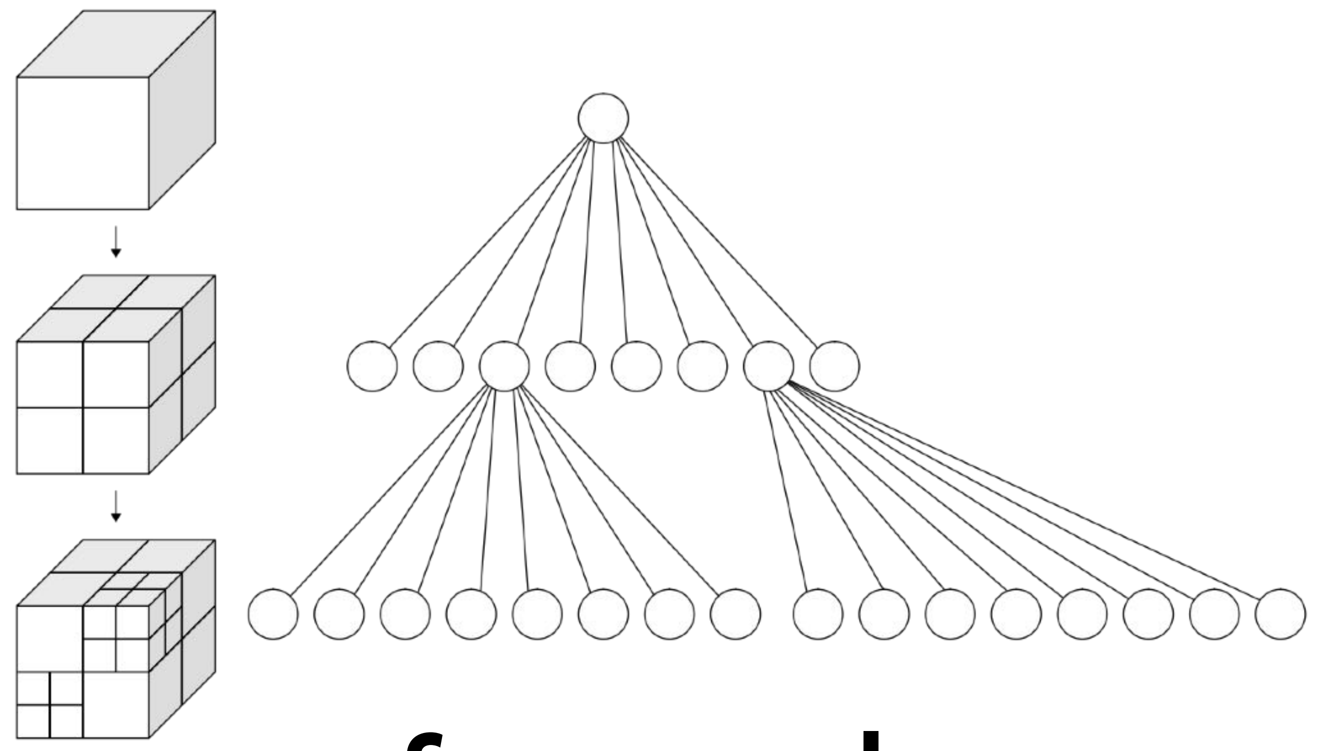
- **Continue optimization on the fixed, sparse representation**
  - **Leverages differential volume rendering on sparse structure**
  - **What we're now learning is how to represent/compress the local details**

But there are many scene representations

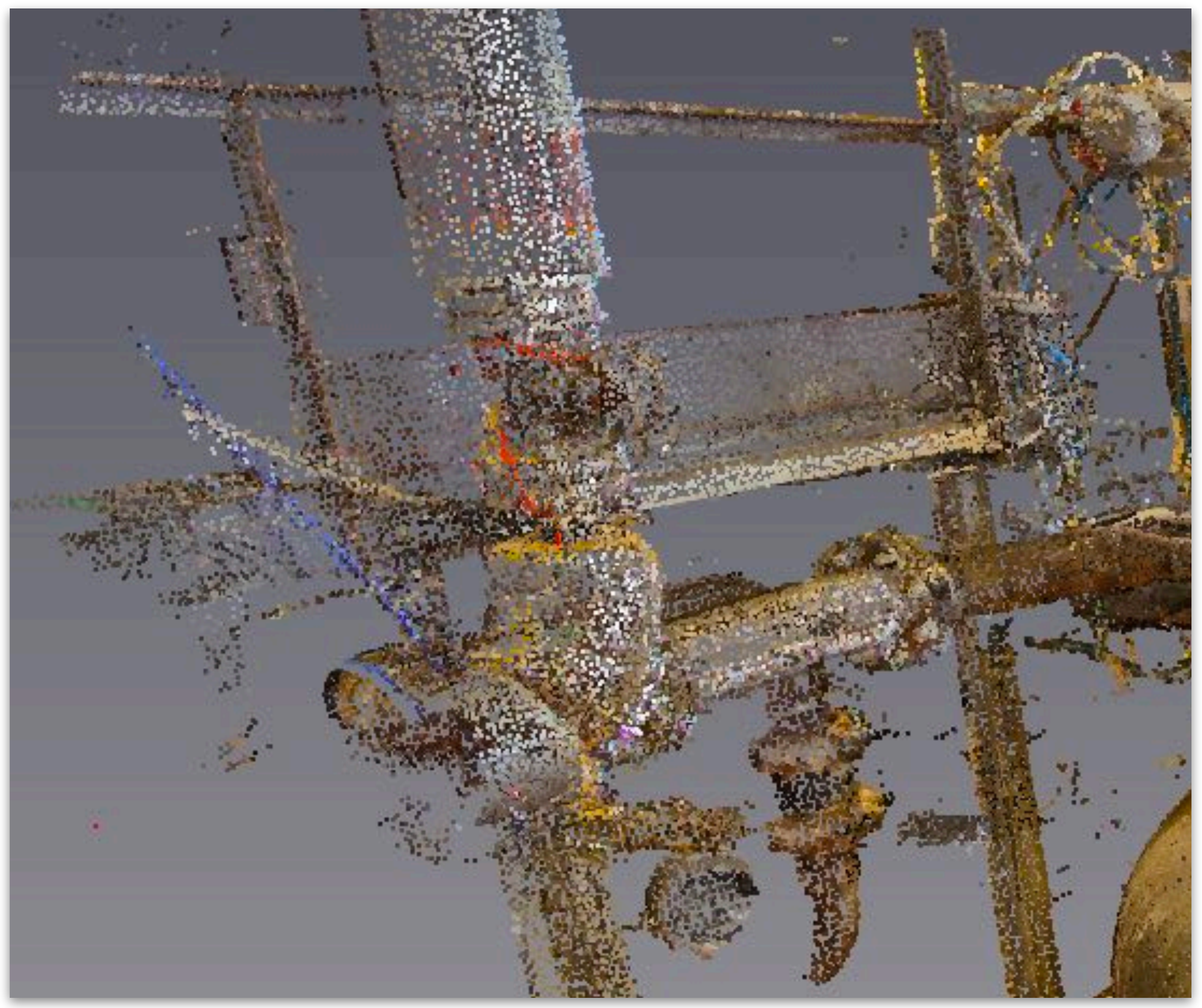


3D triangle mesh + texture map

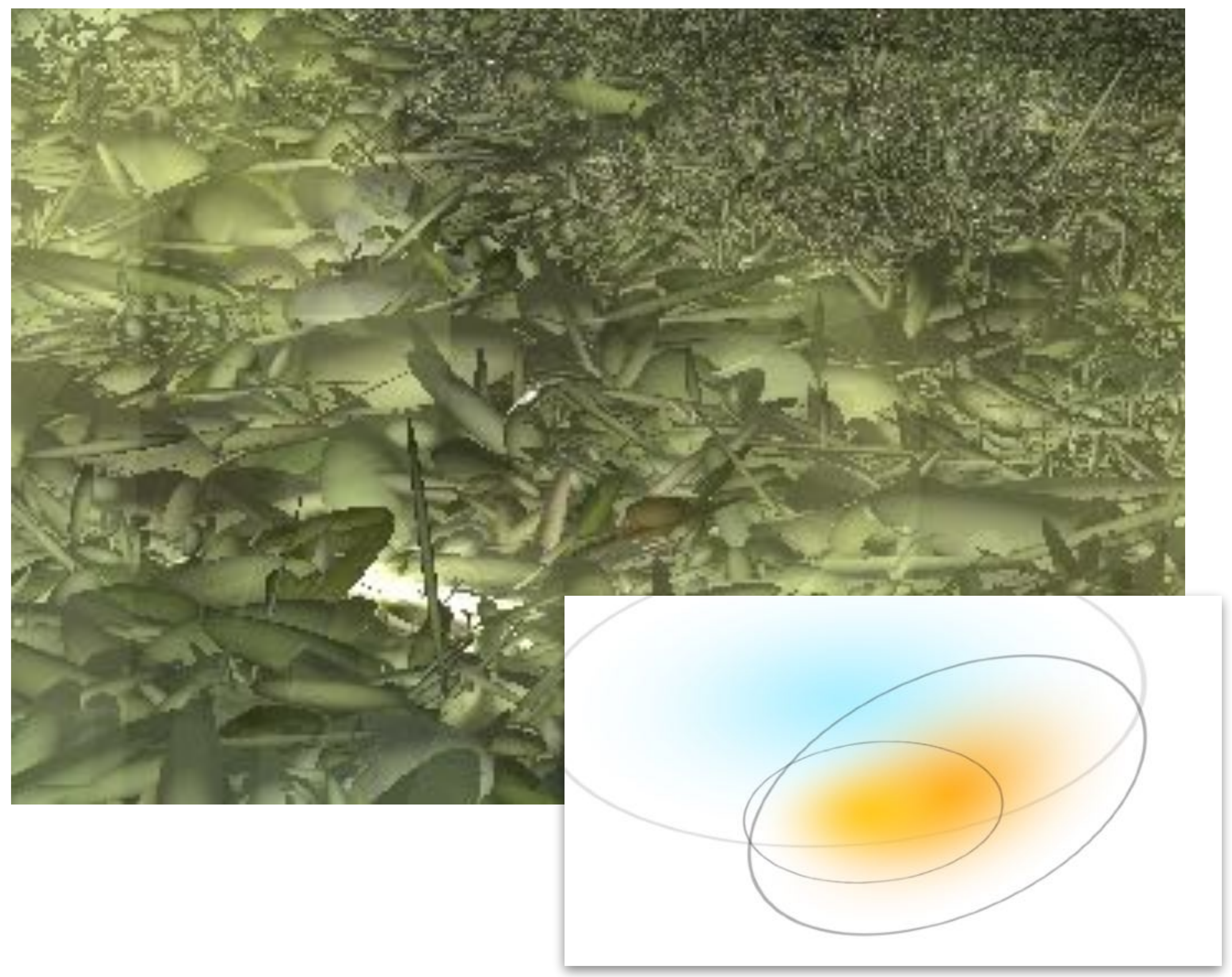
3D volume (voxels)



Sparse voxels

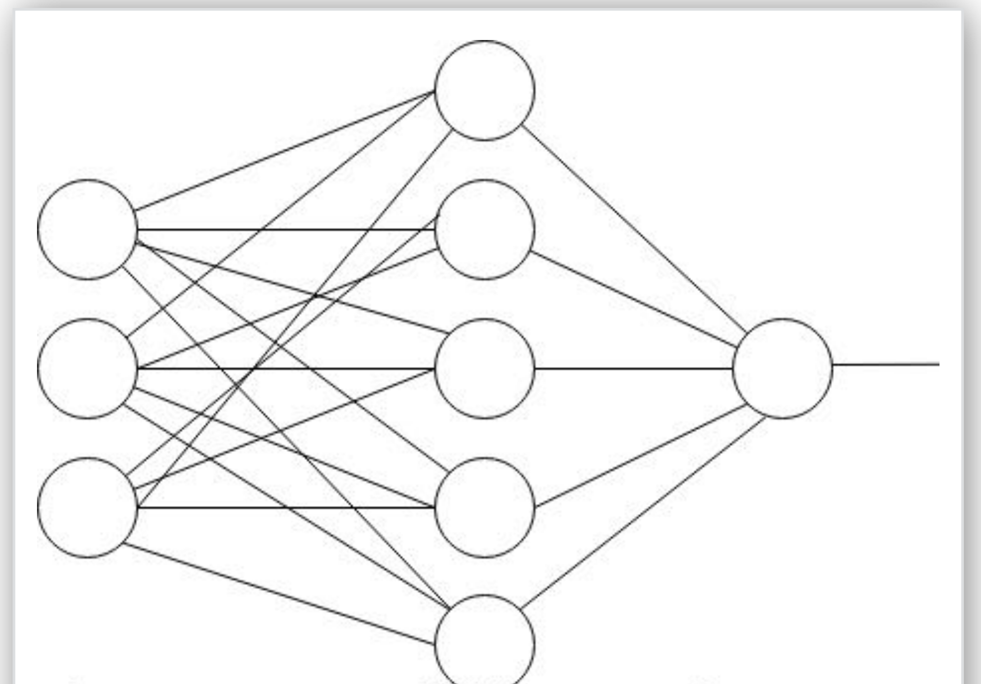


Point cloud (list of points)



Oriented 3D Gaussians

DNN (MLP)



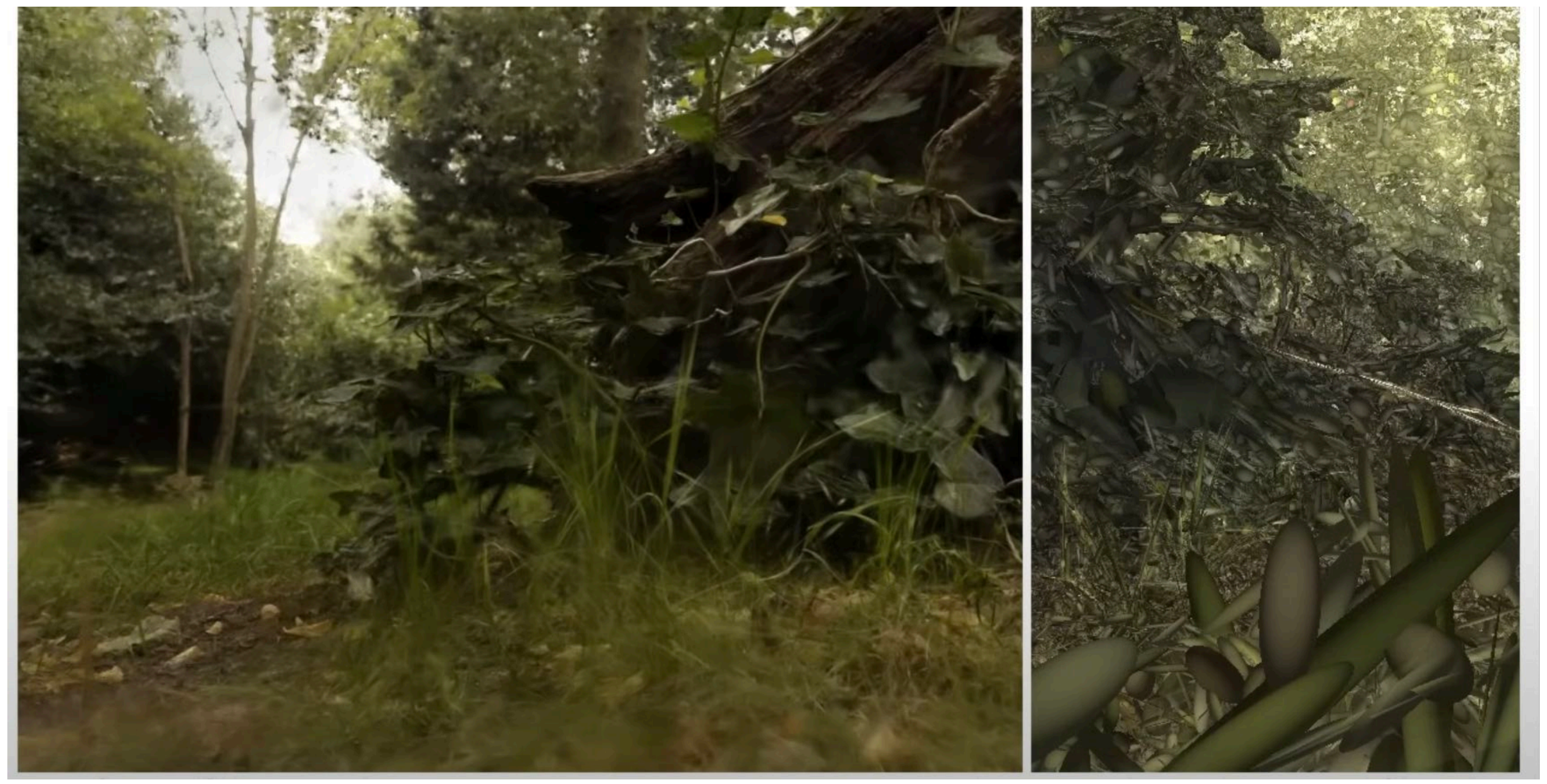
Implicit representations like volumes and DNNs make it hard to know where the "empty space is" (hard to enumerate points on the surface)

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)

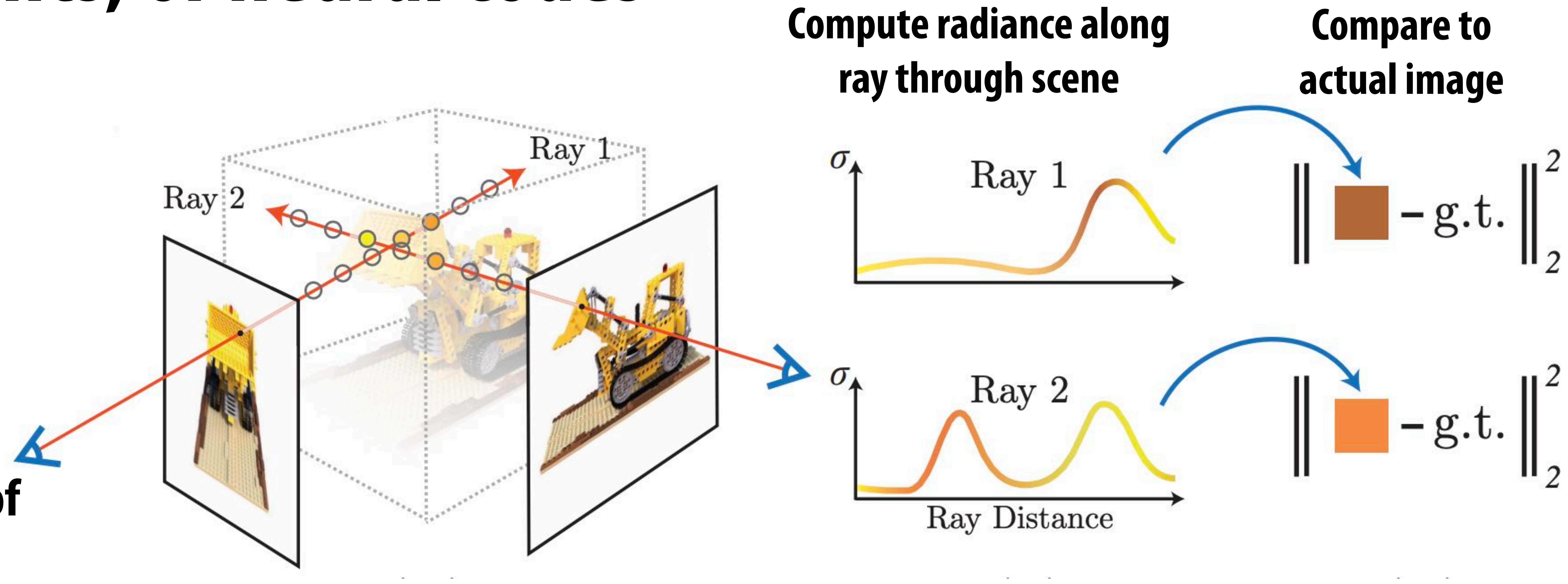
Let's consider one explicit representation that can accurately represent the contents of real world scenes…
A list of 3D Gaussians

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

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

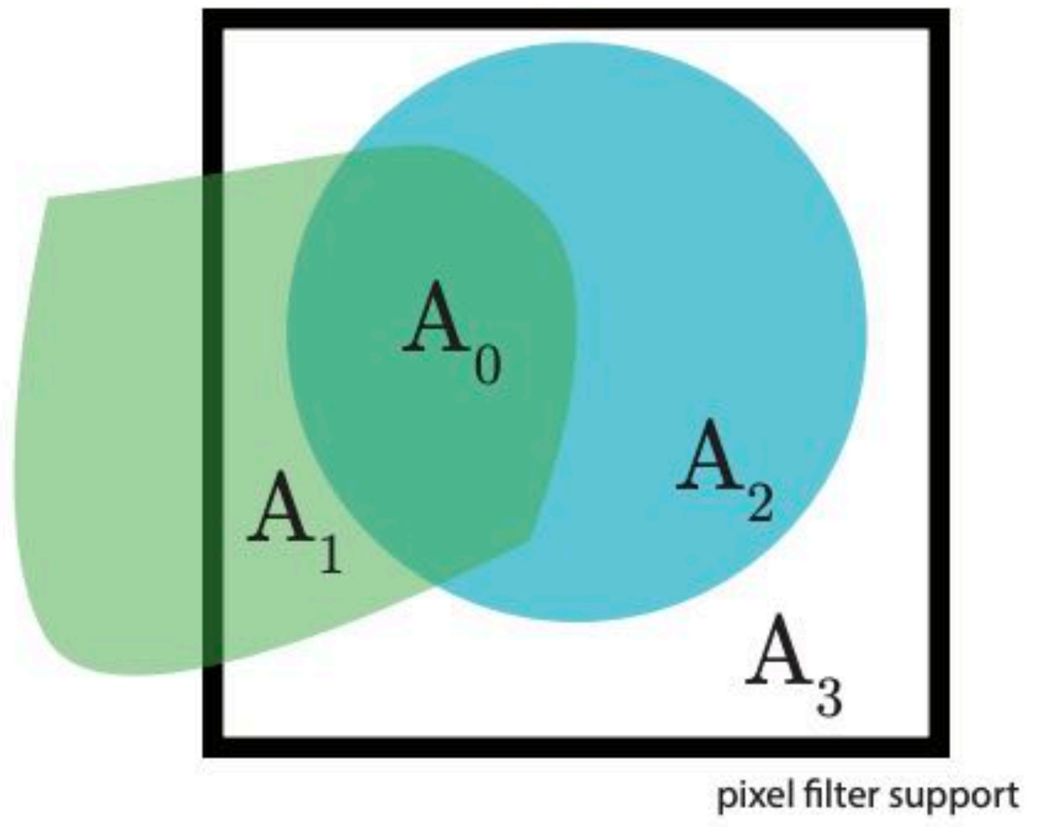
See "3D Gaussian Splatting for Real-Time Radiance Field Rendering" [Kerbl 2023]

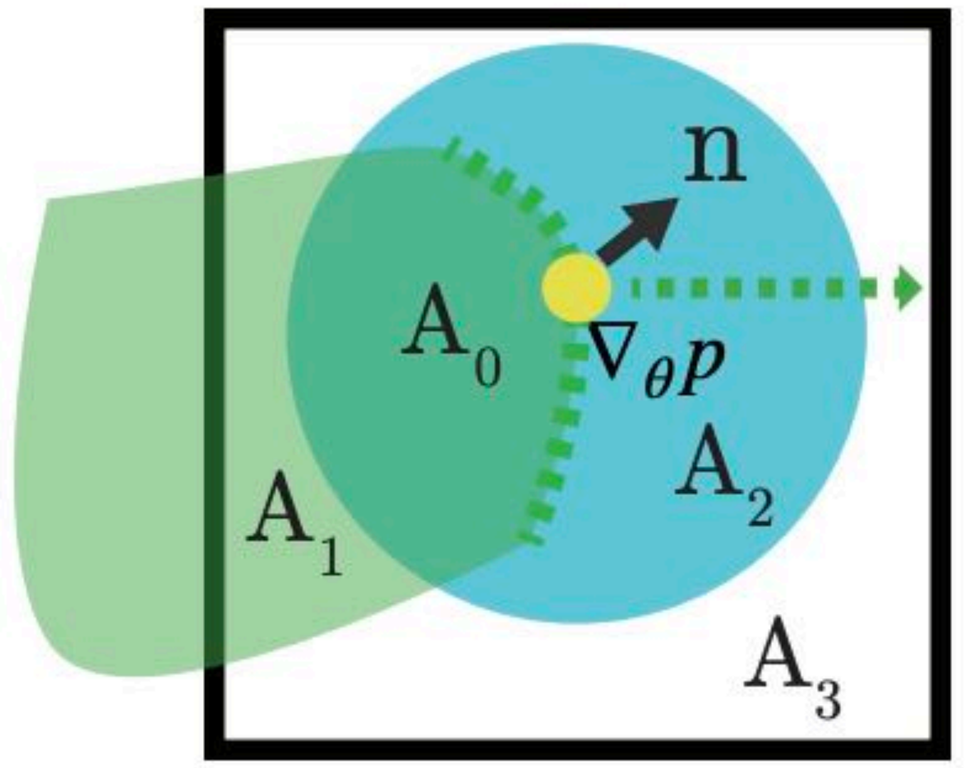
Summary

- Volumes (continuous color/opacity fields) ane 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 2025: 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)

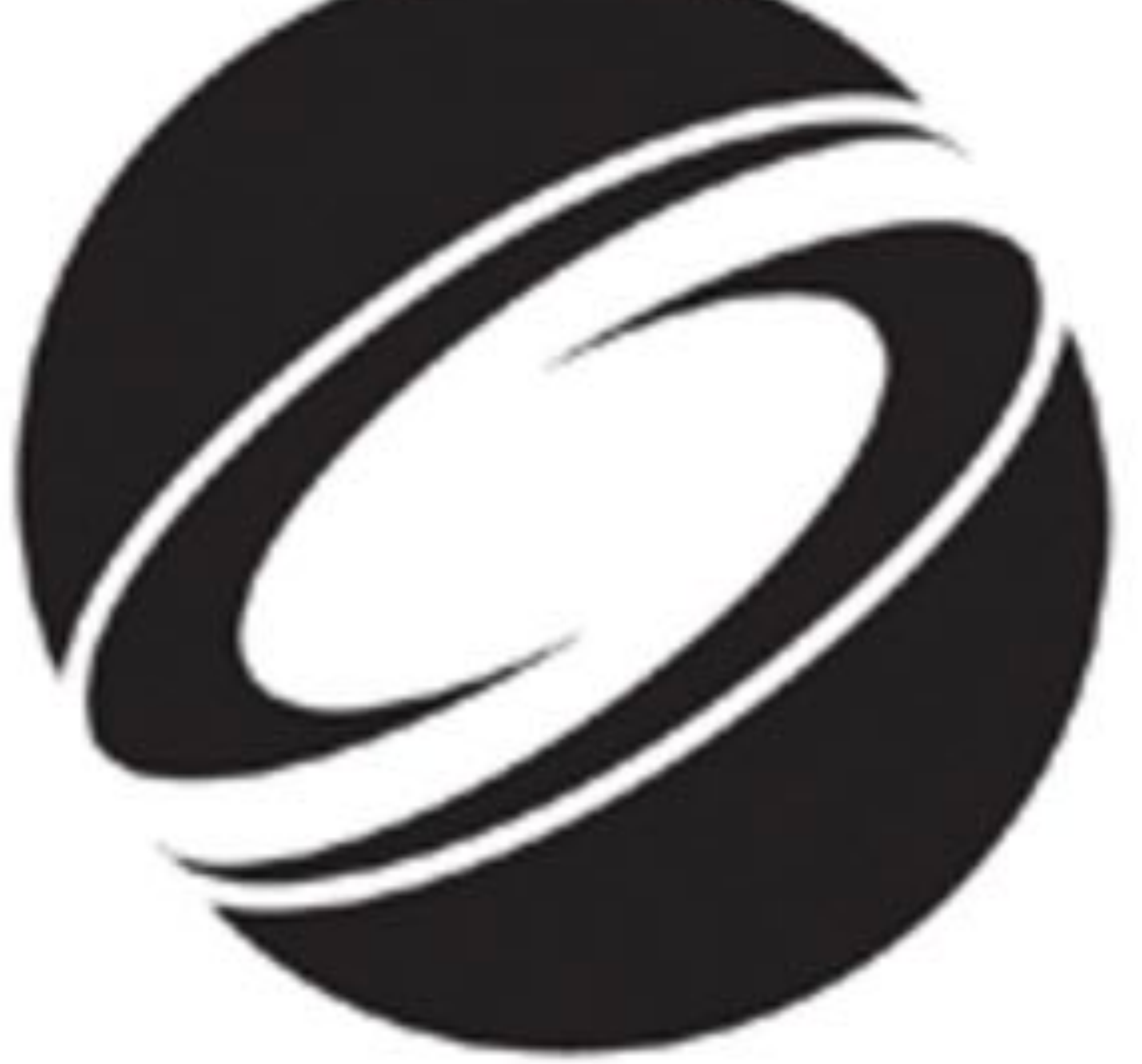


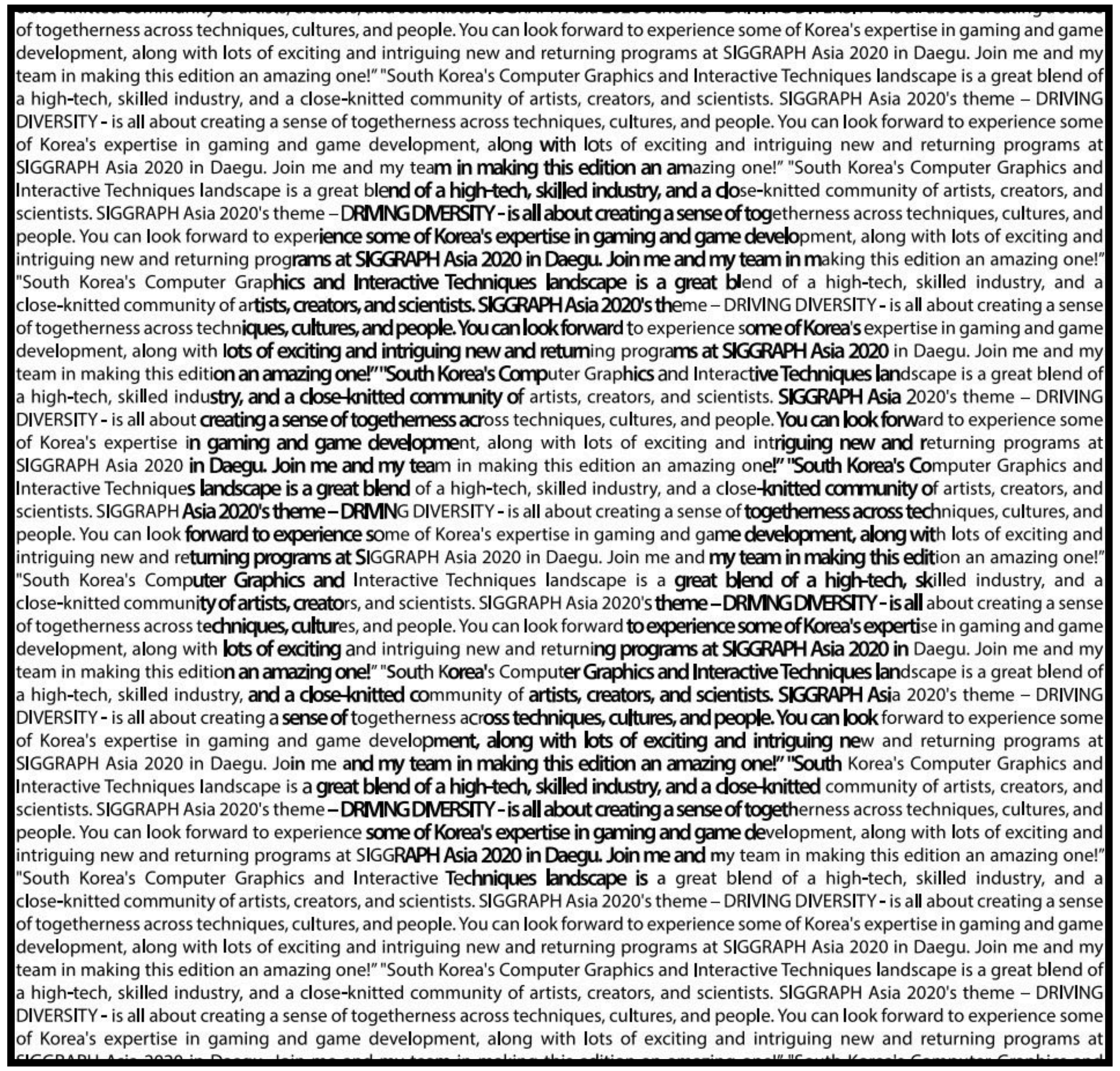


Example uses of differential rasterizers/ray tracers

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

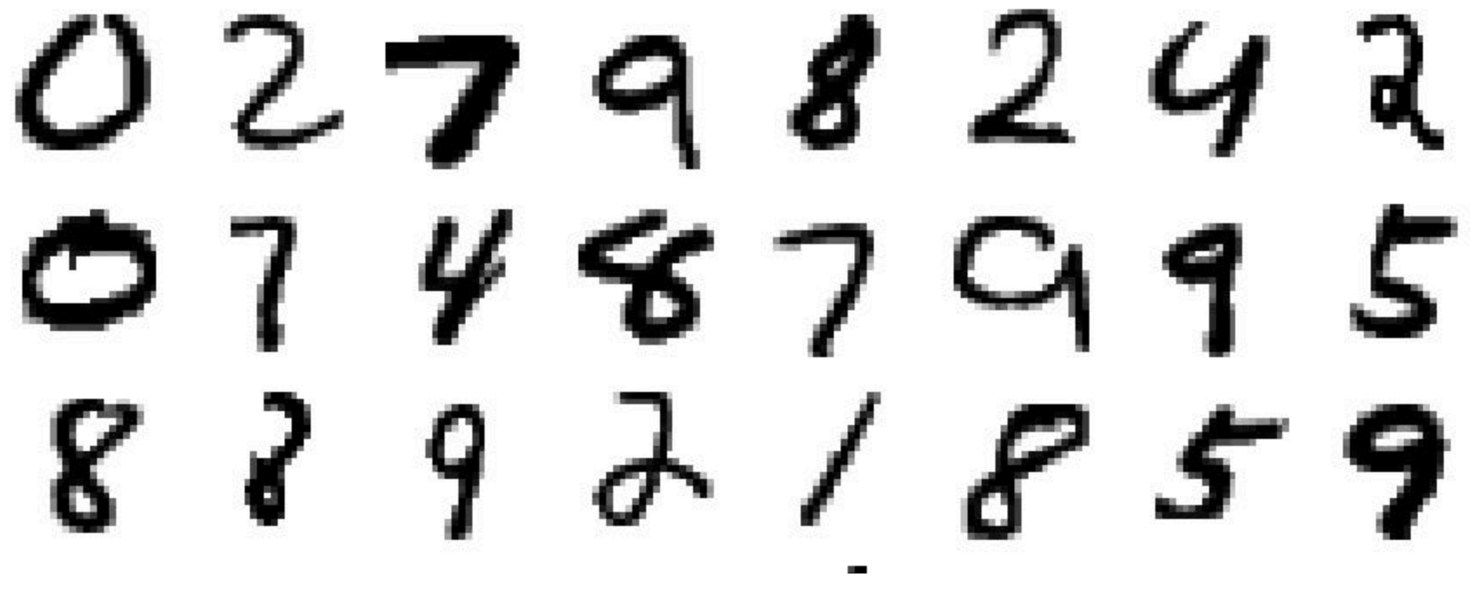
Optimize "bold" parameter of SVG text to match image to right…

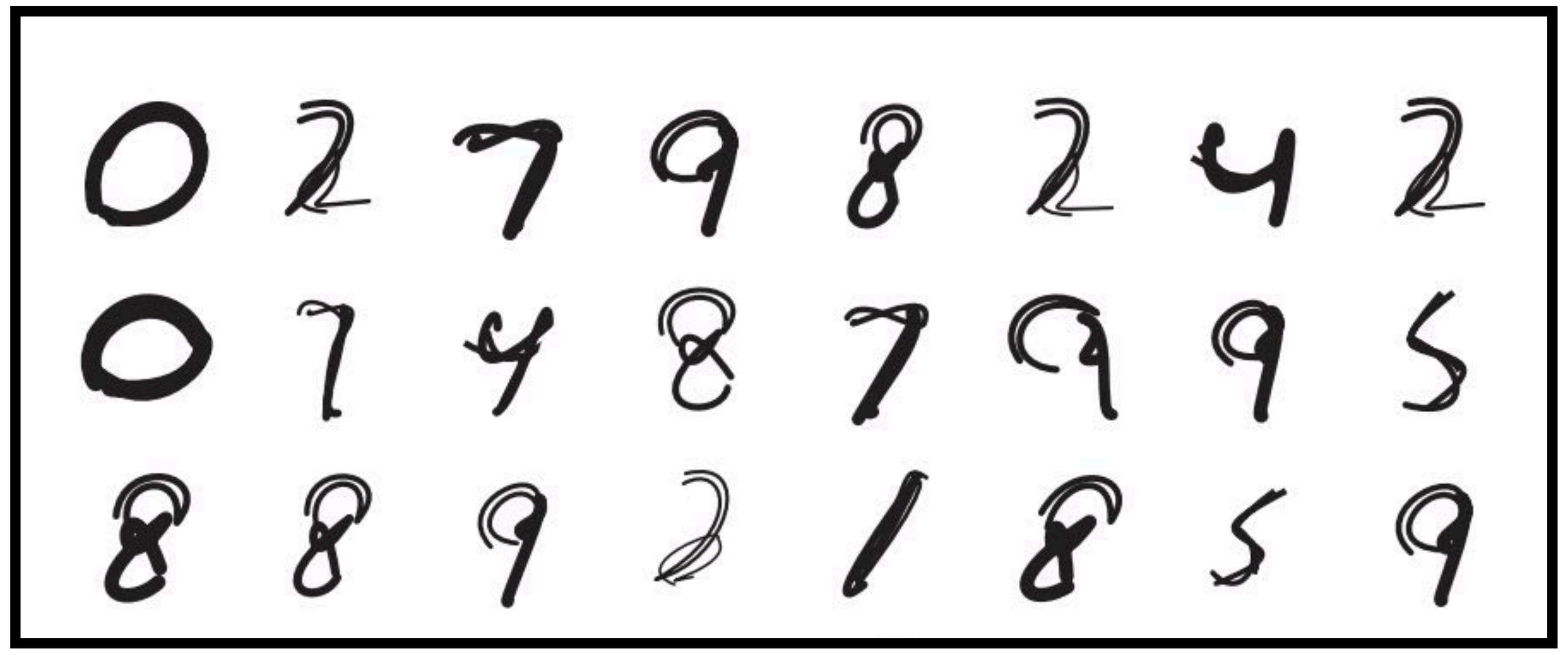




of togetherness across techniques, cultures, and people. You can look forward to experience some of Korea's expertise in gaming and game
development, along with lots of exciting and intriguing new and returning programs at SIGGRAPH Asia 2020 in Daegu. Join me and my
team in making this edition an amazing 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 curve control points to match images of numbers.

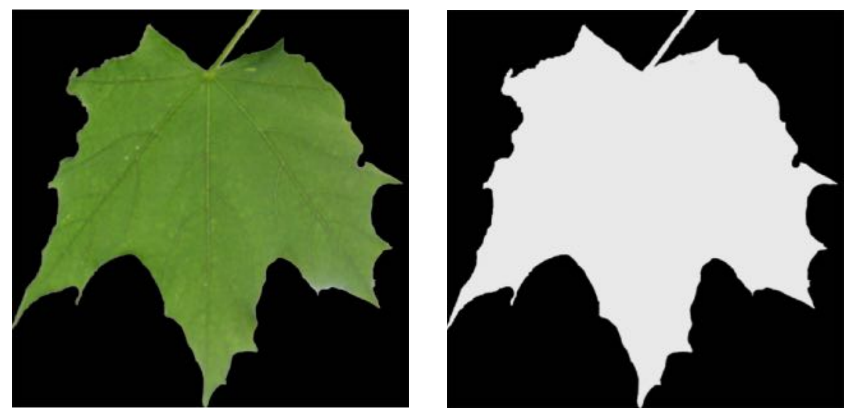




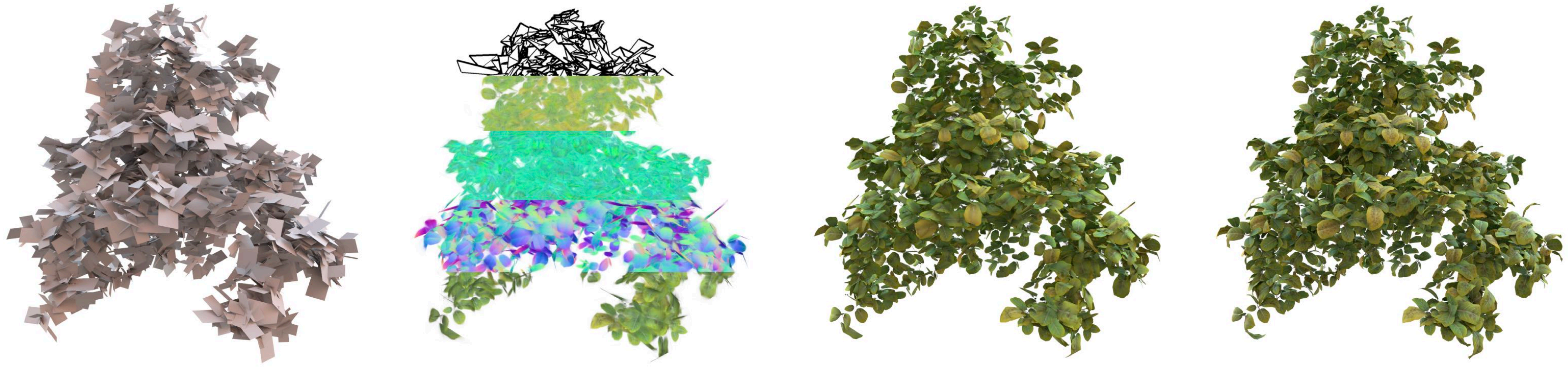
[Li et al. 2020 Differentiable Vector Graphics Rasterization for Editing and Learning]

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

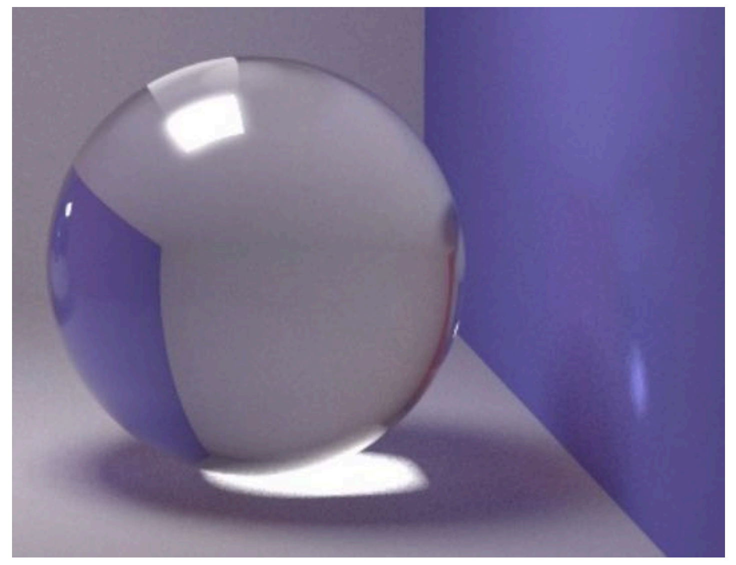


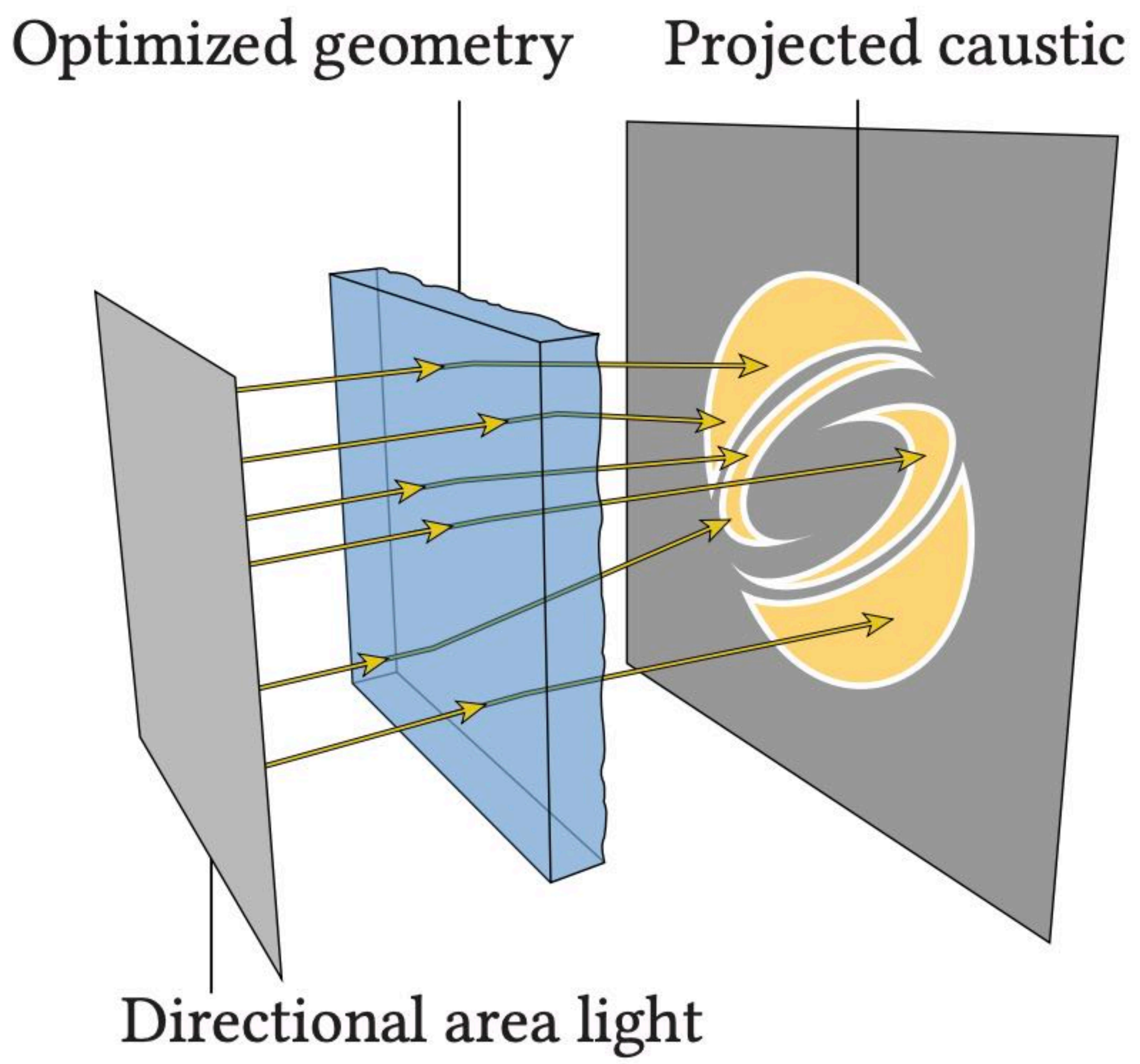
Initial guess (6.5k tris) Optimized parameters Our (6.5k tris) Reference (1.7M tris)

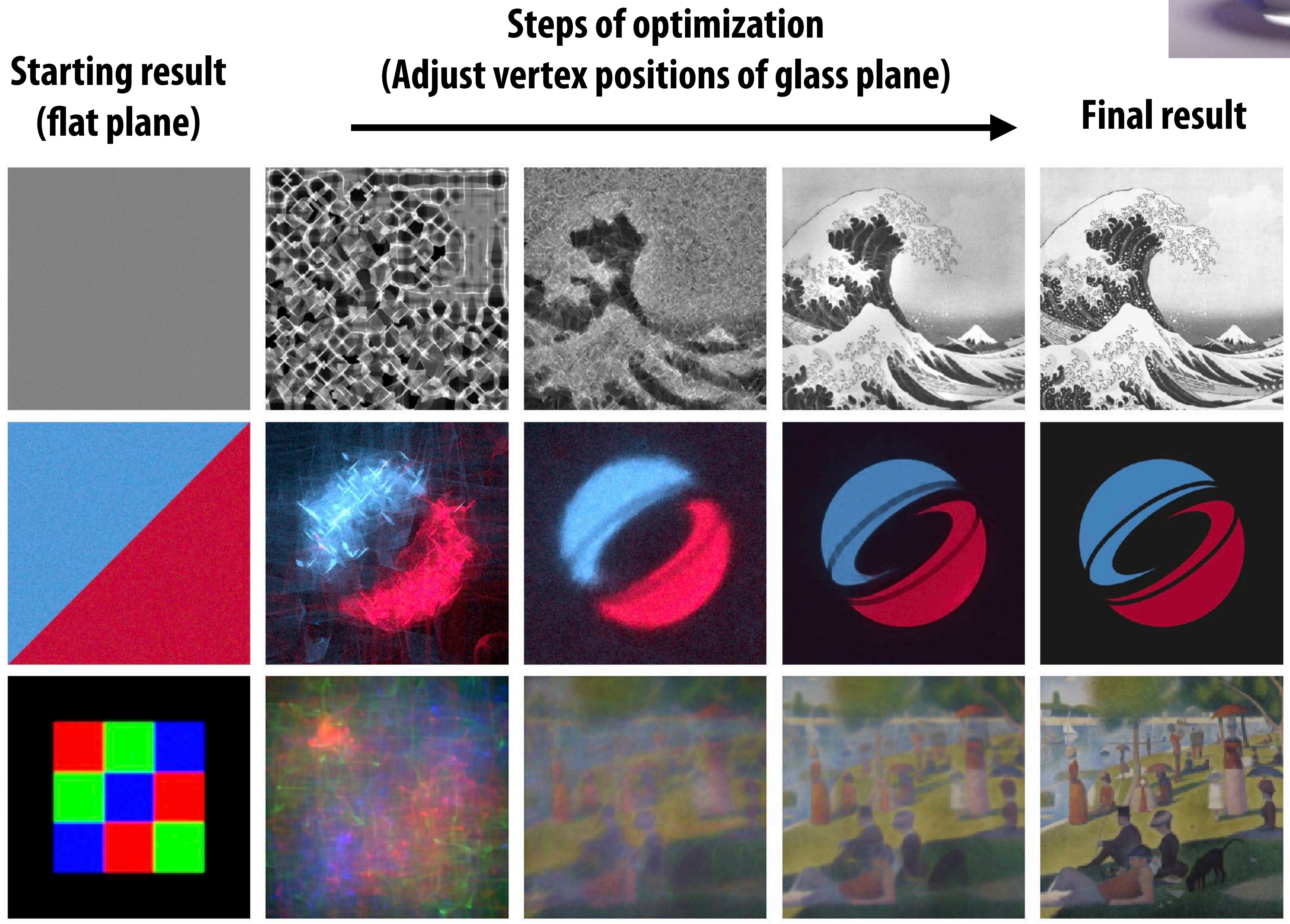
[Hasselgren et al. 2021 Appearance Driven automatic 3D Model Simplification]

Example uses of differential rasterizers/ray tracers

- Recall how we talked about "caustics" that occur when refraction causes light to focus
  - Use differential path tracer to 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.

Summary

- Thanks to Ben Mildenall, Ren Ng for discussions related to these slides