Lecture 13:

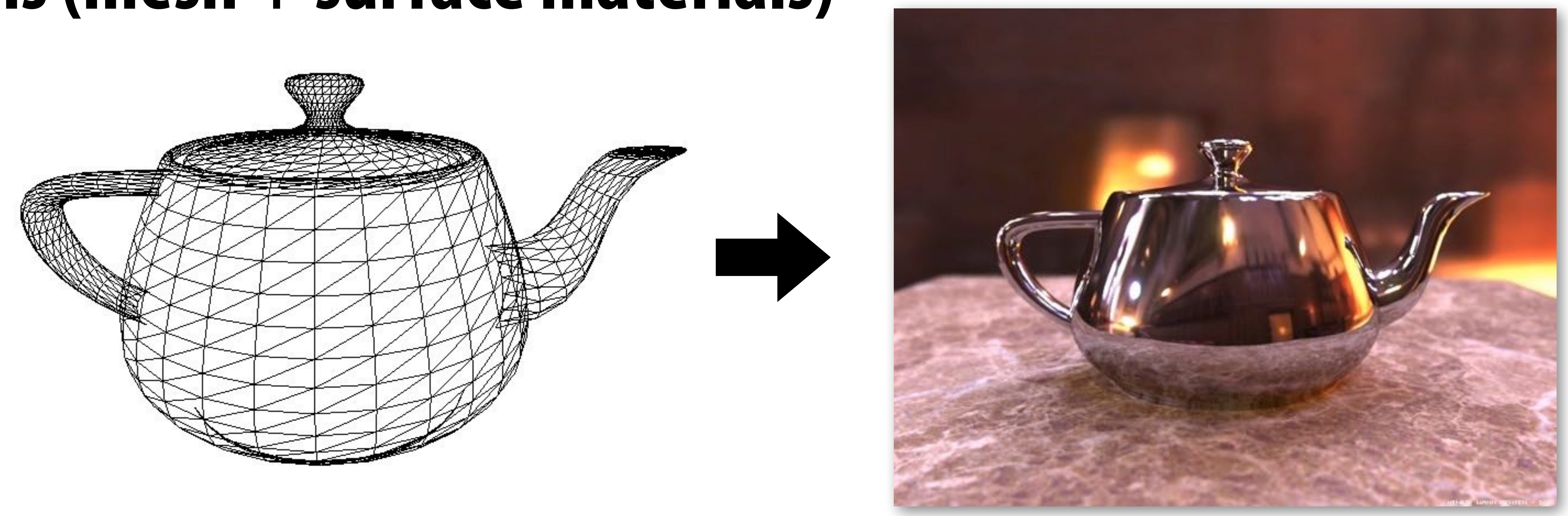
The NeRF Explosion

Visual Computing Systems
Stanford CS348K, Spring 2023

Many scene representations in graphics

Triangle-based 3D surface representations (mesh + surface materials)

(Rendering via ray-casting or 2D projection)

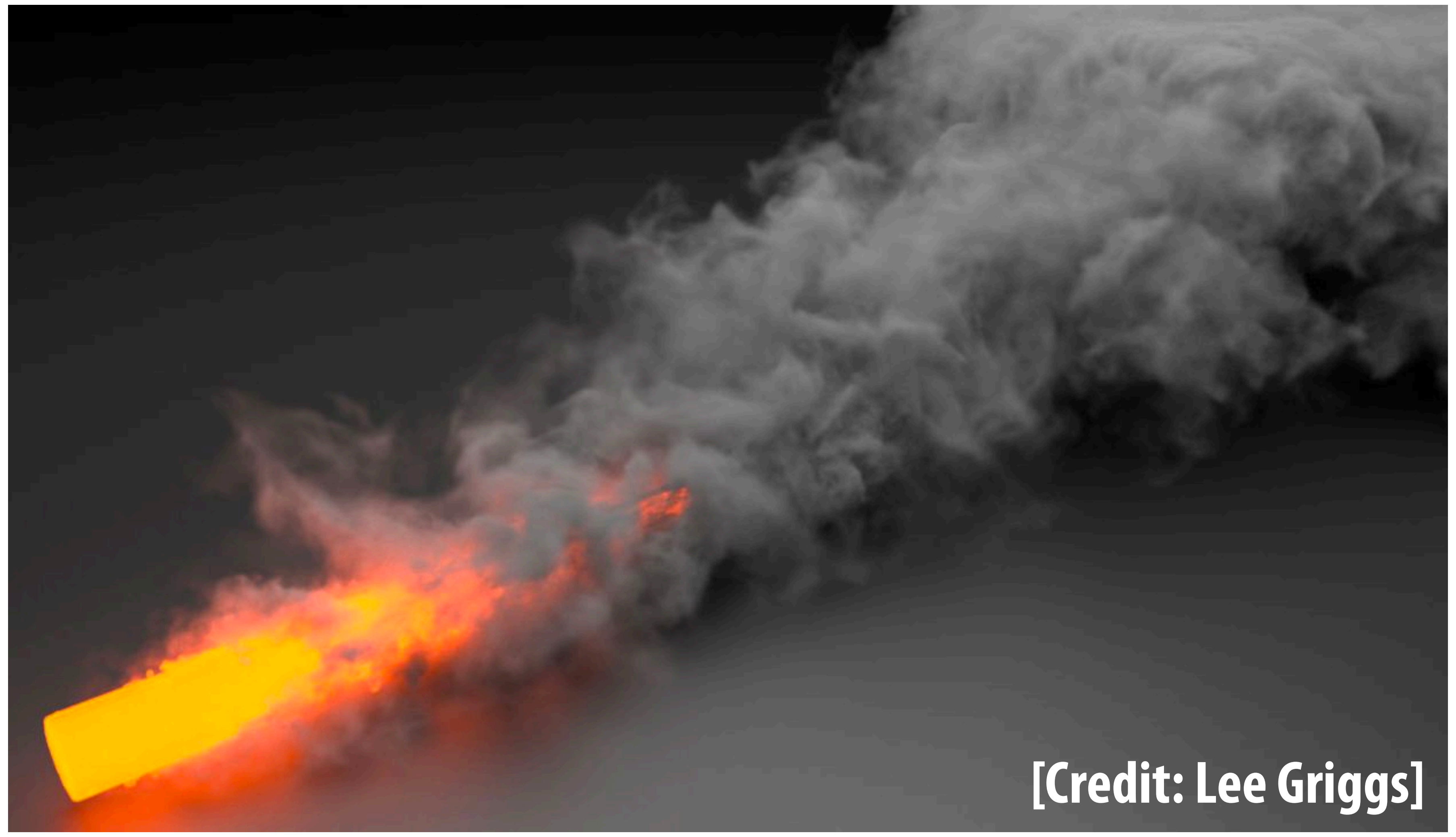


Depth-image based surface representations

(Novel view synthesis via depth-guided image warping, pixel re-projection, etc.)

[Andersen 16] Jump: VR video

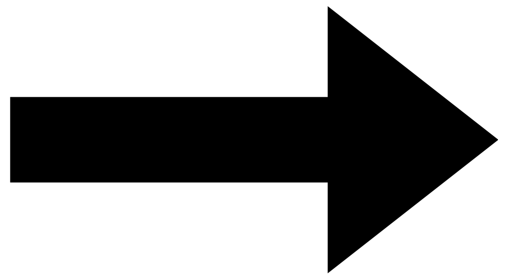
3D Volumes



And many more… e.g., Implicit Surfaces

Novel view synthesis problem

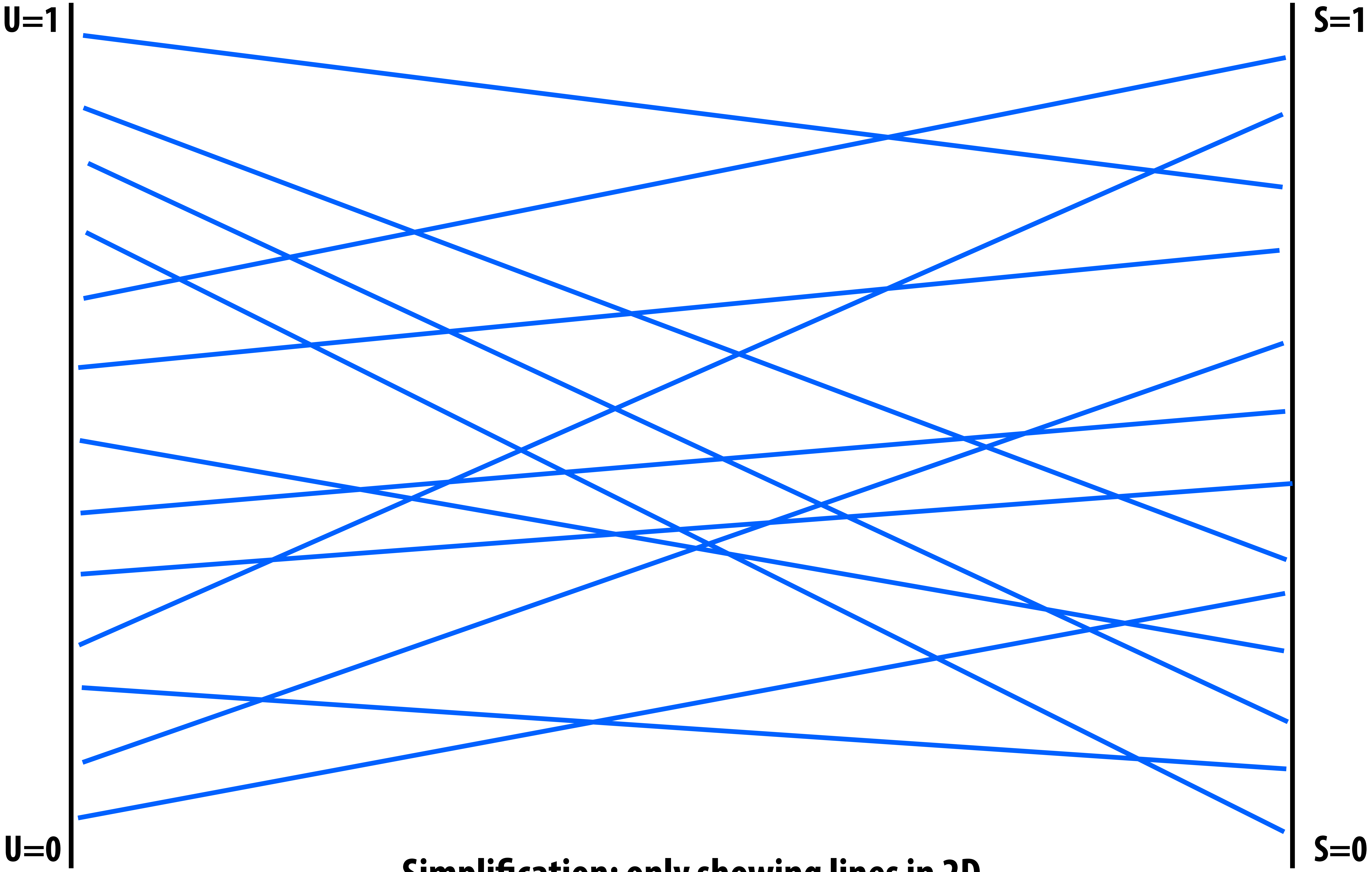
Input photos (from a fixed set of views)



Novel views
(camera position different from those in input photos)

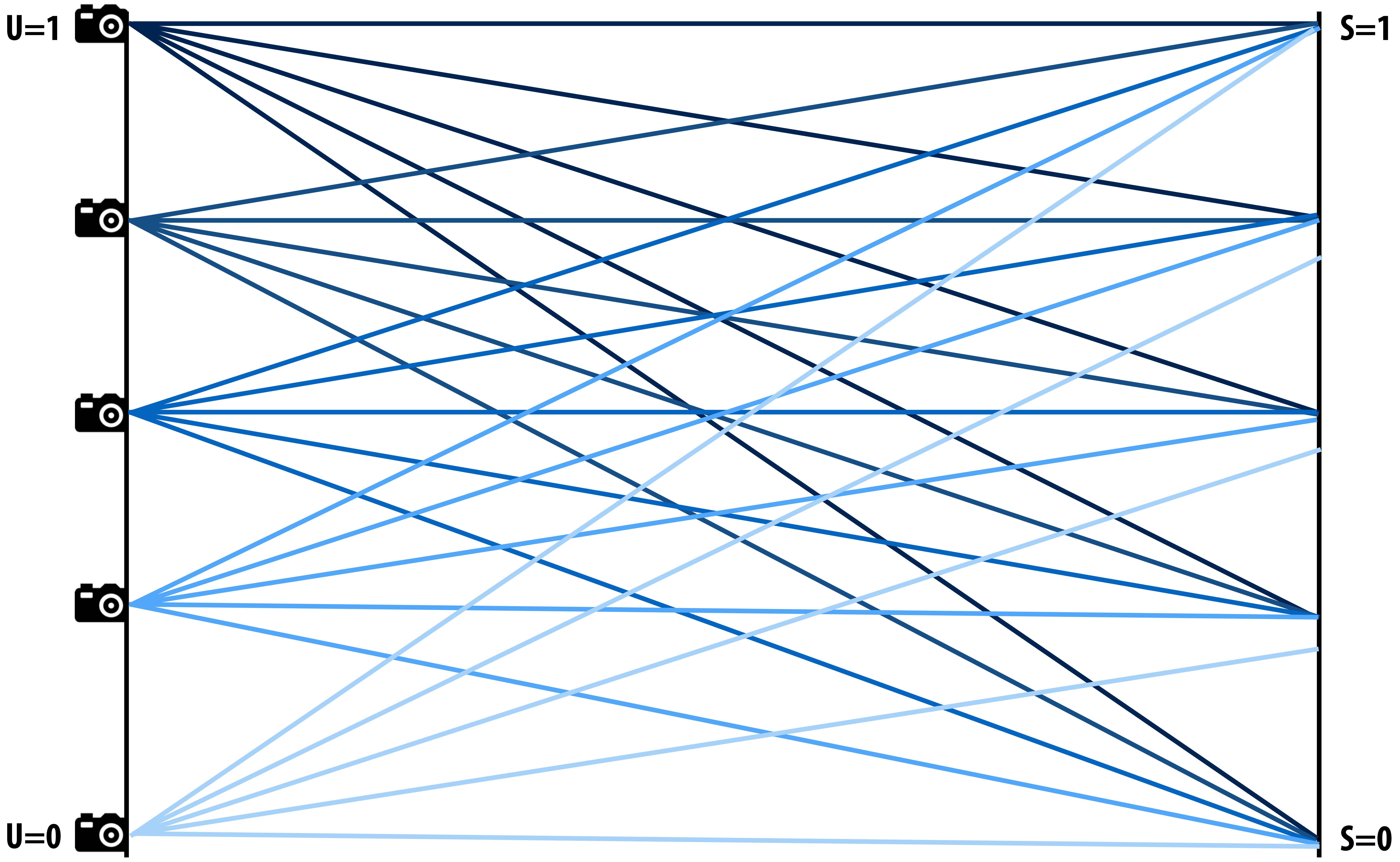
Fundamentals: the light field

Review: sampling the light field



Simplification: only showing lines in 2D
(full light field is 4D function)

Review: measuring the light field by taking many pictures



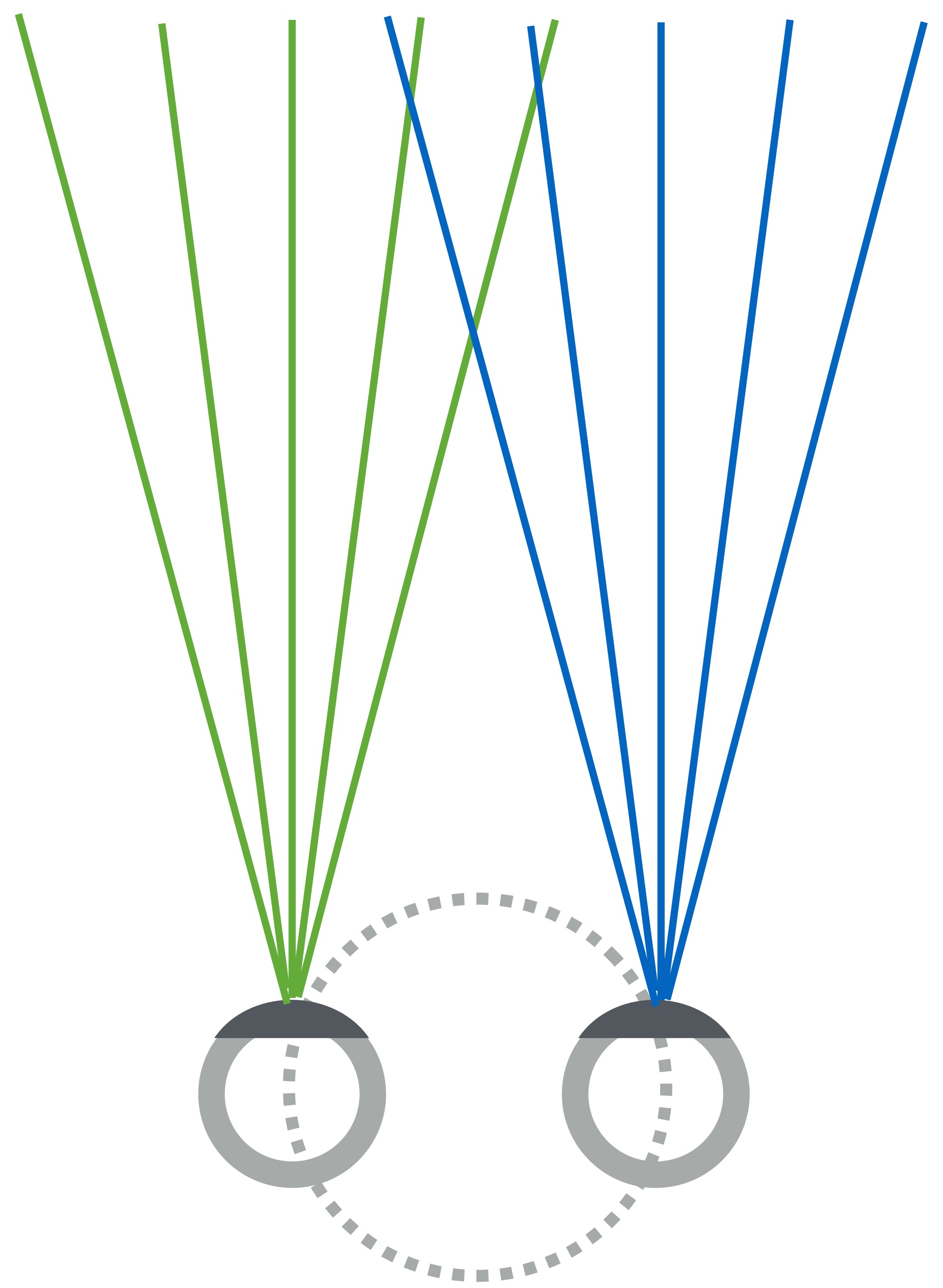
[Credit: Camera icon by Venkatesh Aiyulu from The Noun Project]

Acquiring light field content for VR

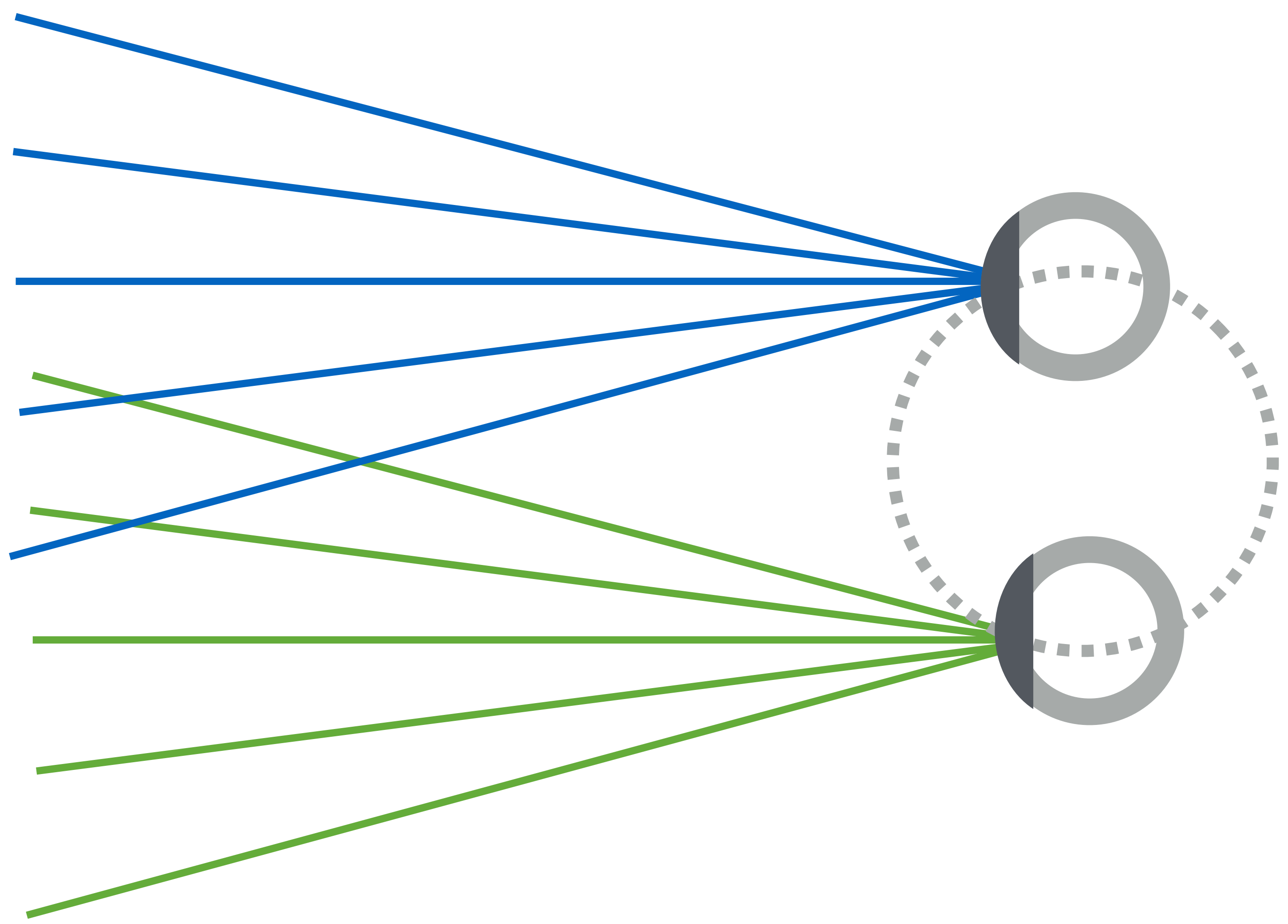
Google's Jump VR video:
Yi Halo Camera (17 cameras)

Facebook Manifold
(16 8K cameras)

Stereo, 360-degree viewing



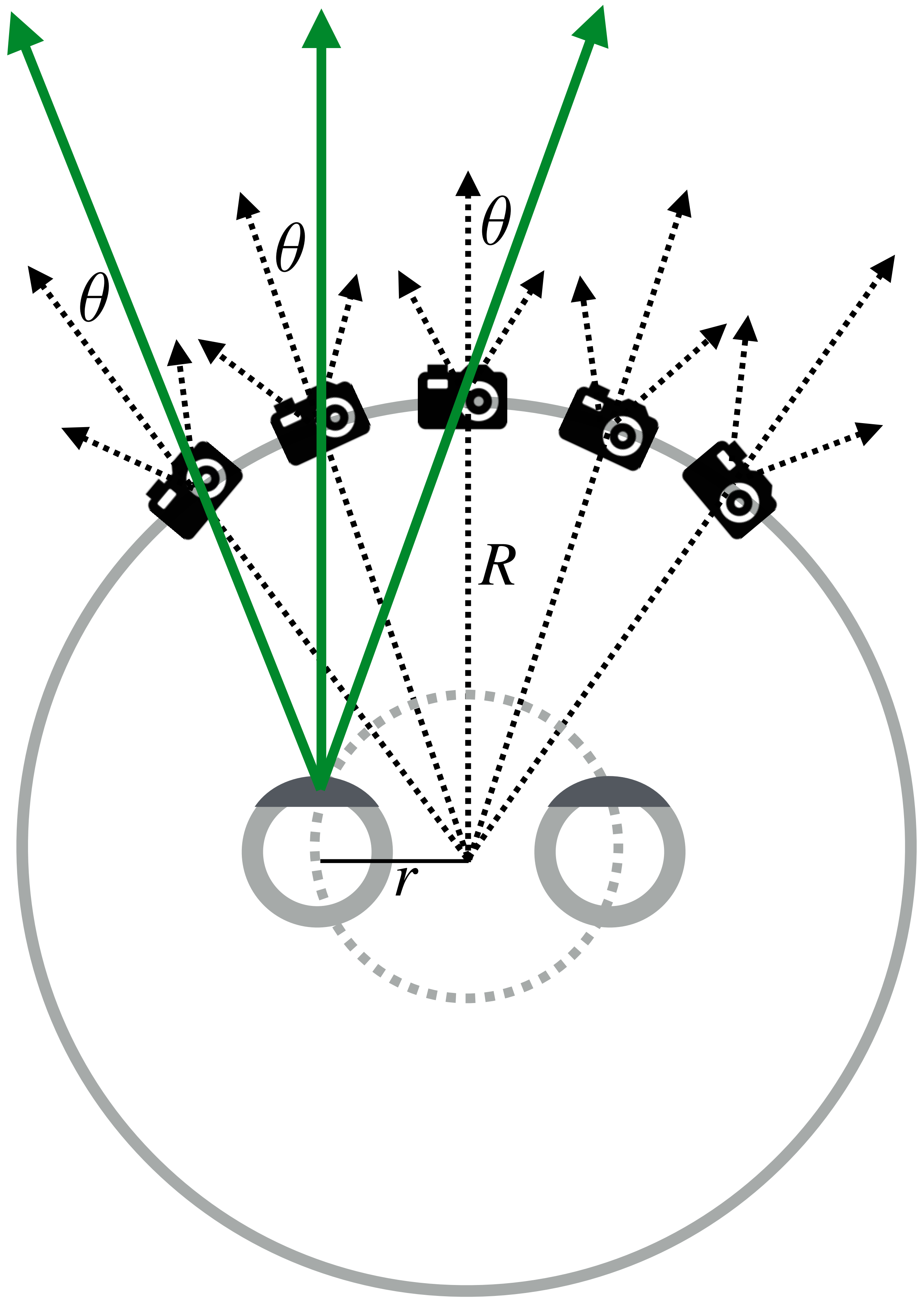
Stereo, 360-degree viewing



Measuring light arriving at left eye

Left eye

$$\sin\theta = r/R$$

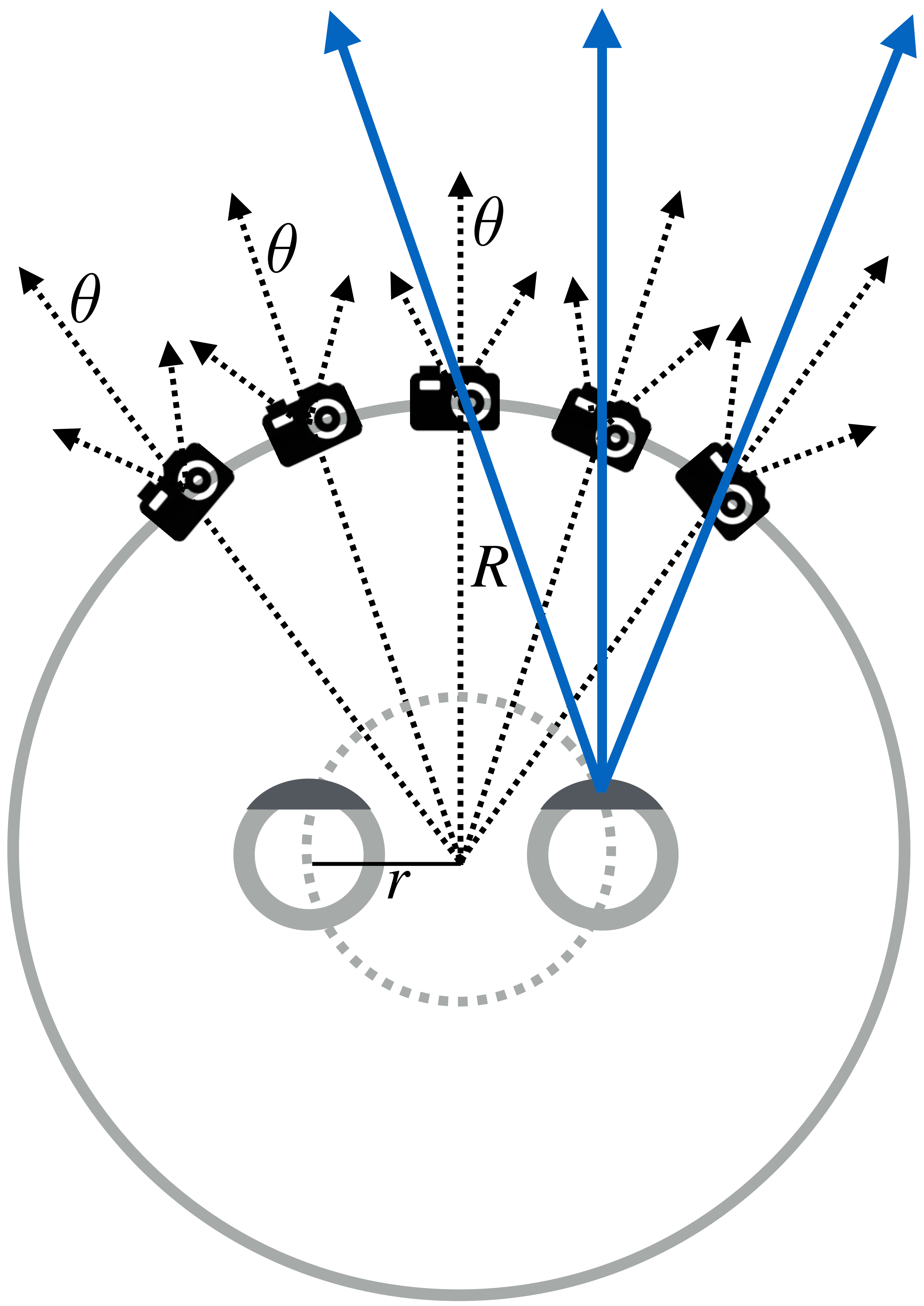


[Credit: Camera icon by Venkatesh Aiyulu from The Noun Project]

Measuring light arriving at right eye

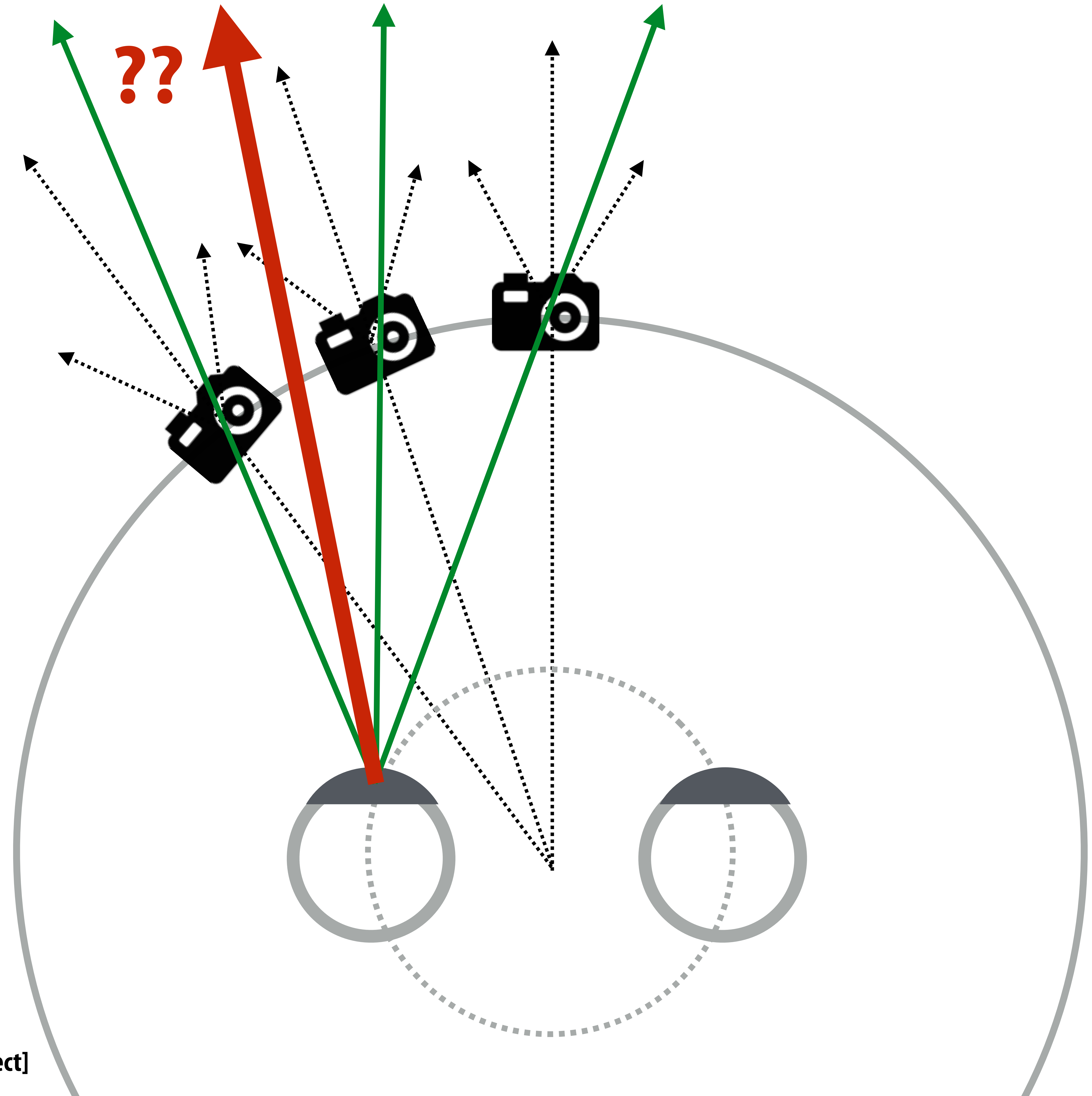
Right eye

$$\sin\theta = -r/R$$



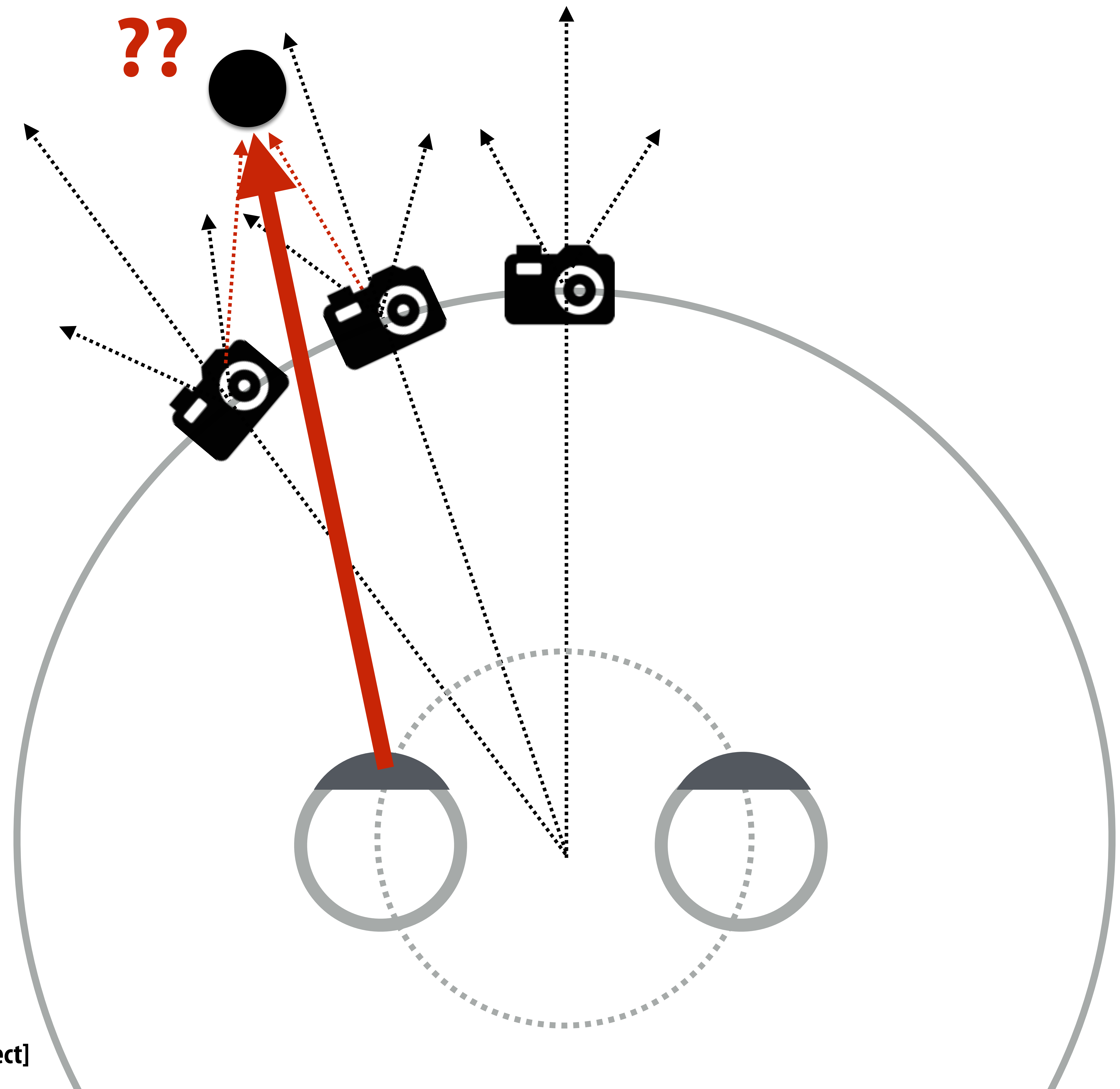
[Credit: Camera icon by Venkatesh Aiyulu from The Noun Project]

How to estimate rays at "missing" views?



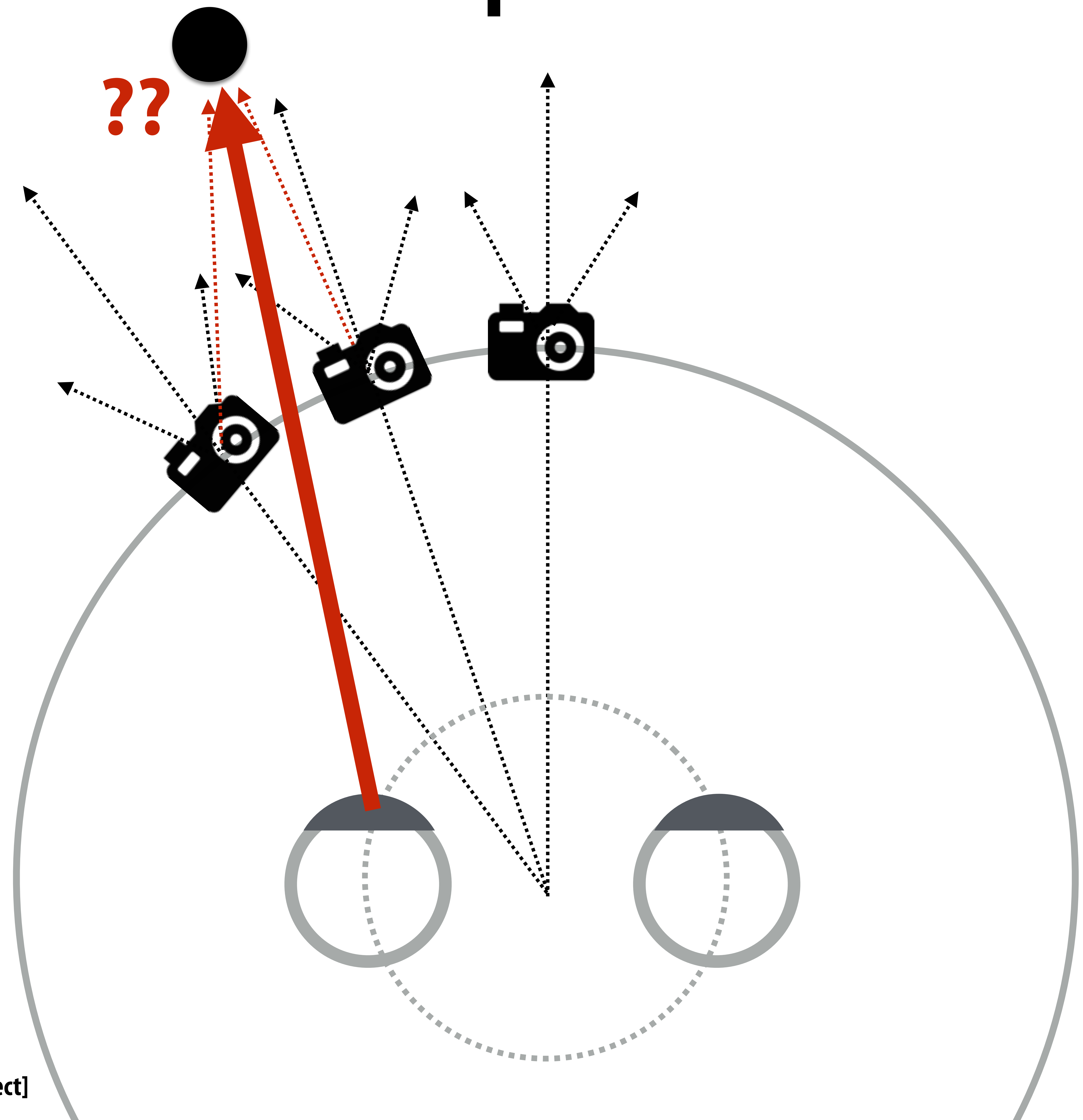
[Credit: Camera icon by Venkatesh Aiyulu from The Noun Project]

Interpolation to novel views depends on scene depth



[Credit: Camera icon by Venkatesh Aiyulu from The Noun Project]

Interpolation to novel views depends on scene depth



[Credit: Camera icon by Venkatesh Aiyulu from The Noun Project]

Computing depth of scene point from two images

Binocular stereo 3D reconstruction of point *P*: depth from disparity

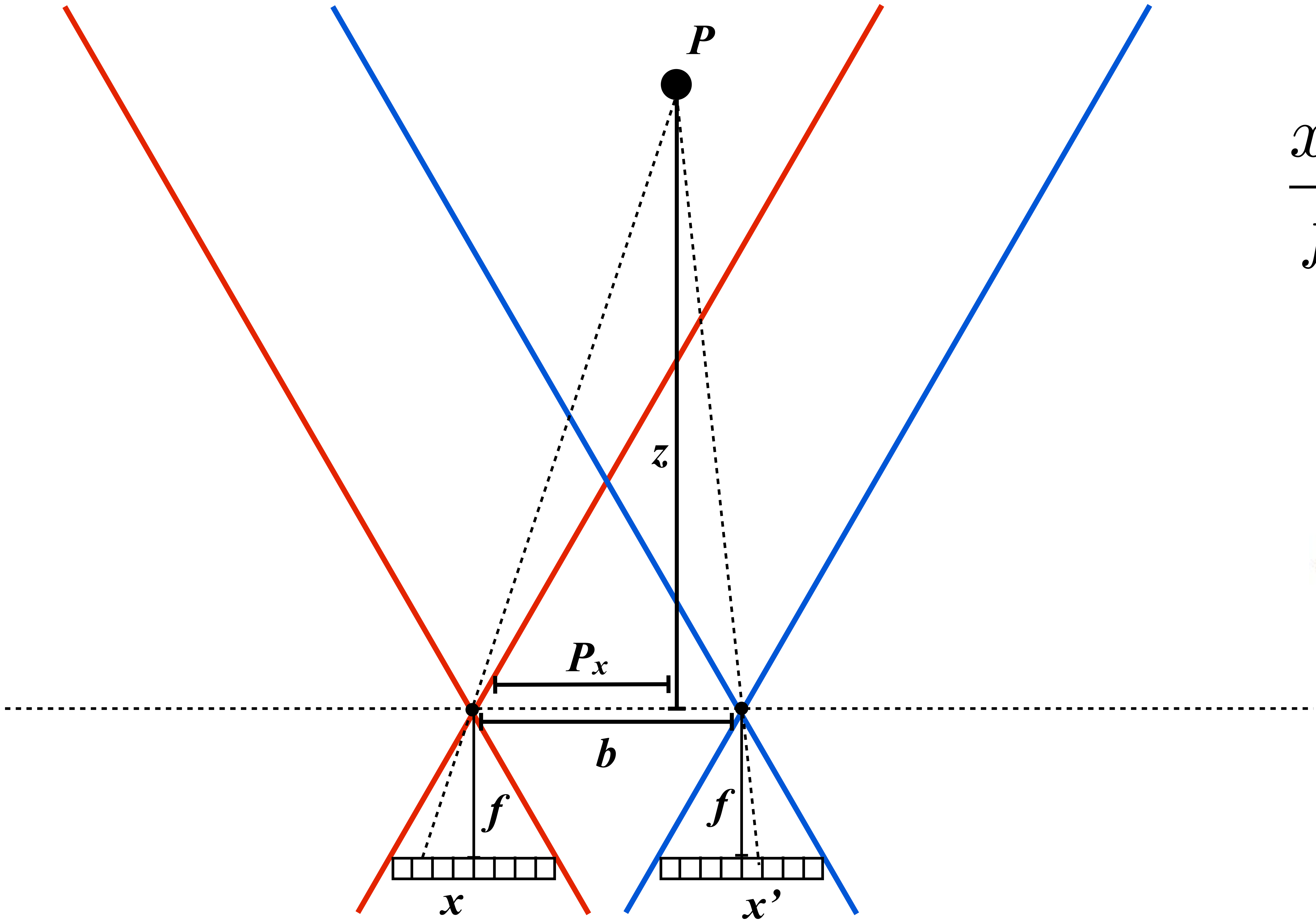
Known from device construction:

Focal length: f

Camera baseline: b

Measured:

Disparity: $d = x - x'$



$$\frac{x}{f} = \frac{P_x}{z}$$

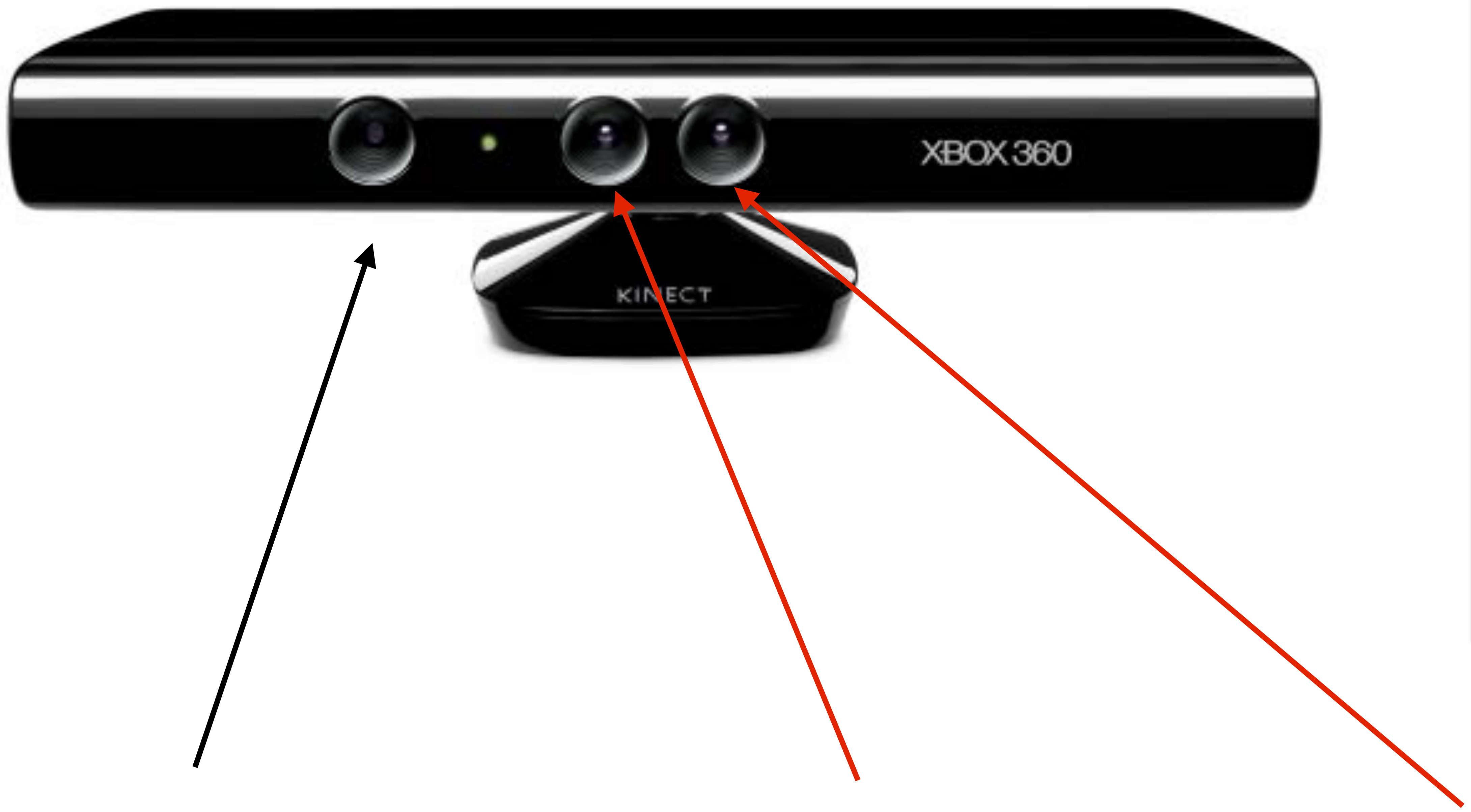
$$\frac{x'}{f} = \frac{P_x - b}{z}$$

$$z = \frac{bf}{d}$$

Simple reconstruction example: cameras aligned (coplanar sensors), separated by known distance *b* ("camera baseline"), same focal length *f*

"Disparity" is the distance between object's projected position in the two images: x - x'

Microsoft XBox 360 Kinect



Illuminant
(Infrared Laser + diffuser)

RGB CMOS Sensor
640x480 (w/ Bayer mosaic)

Monochrome Infrared
CMOS Sensor
(Aptina MT9M001)
1280x1024 **

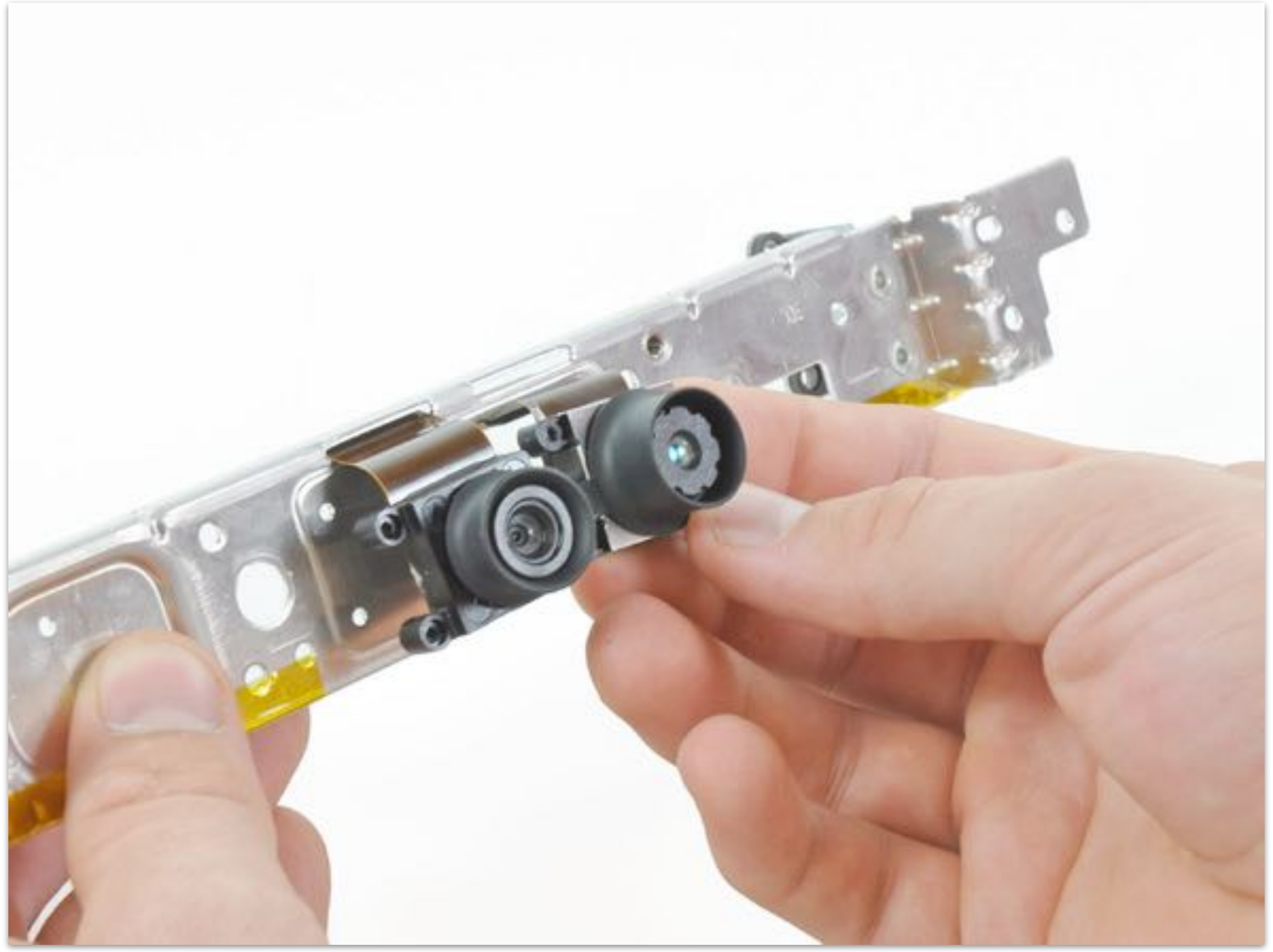
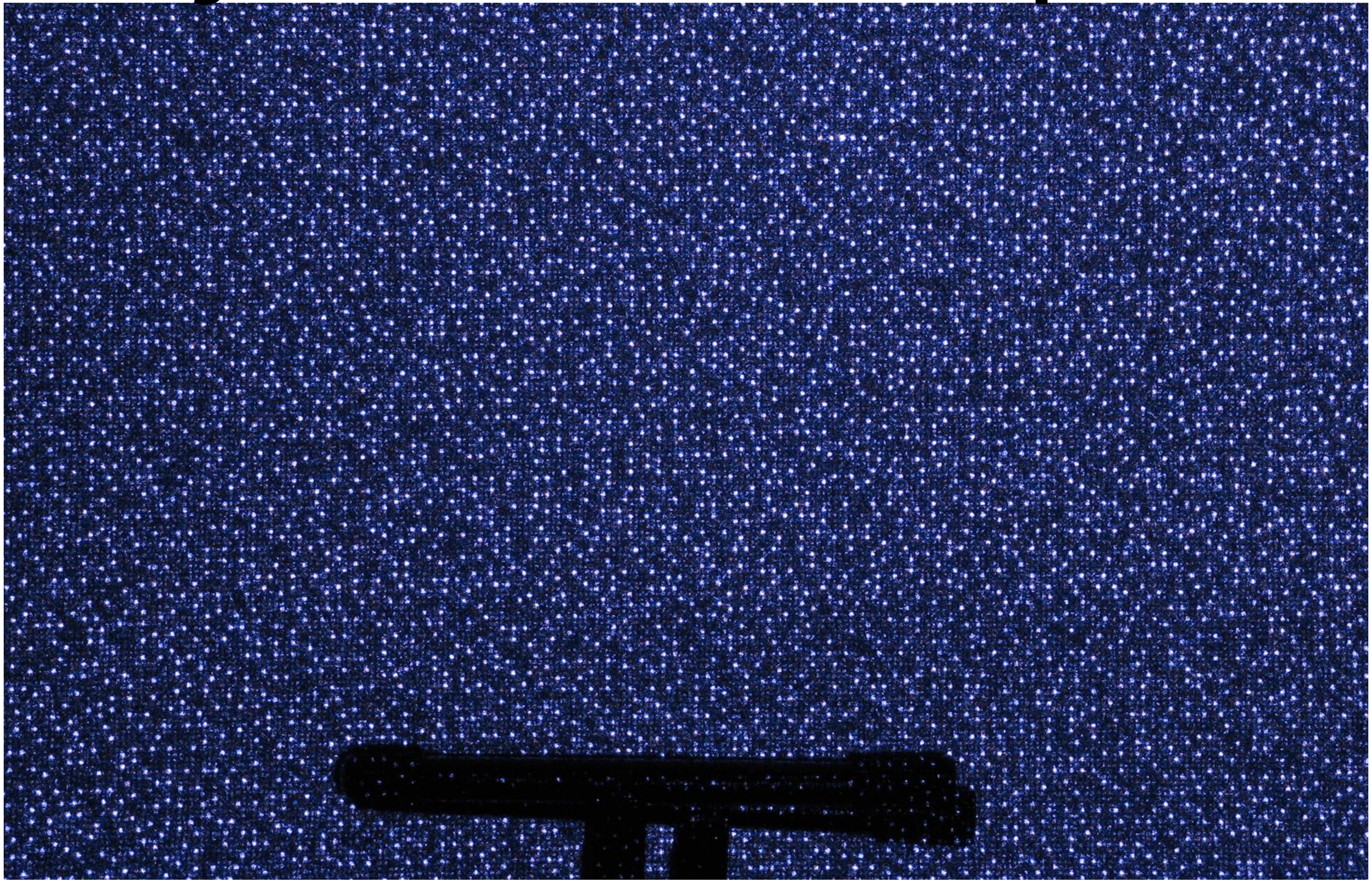


Image credit: iFixIt

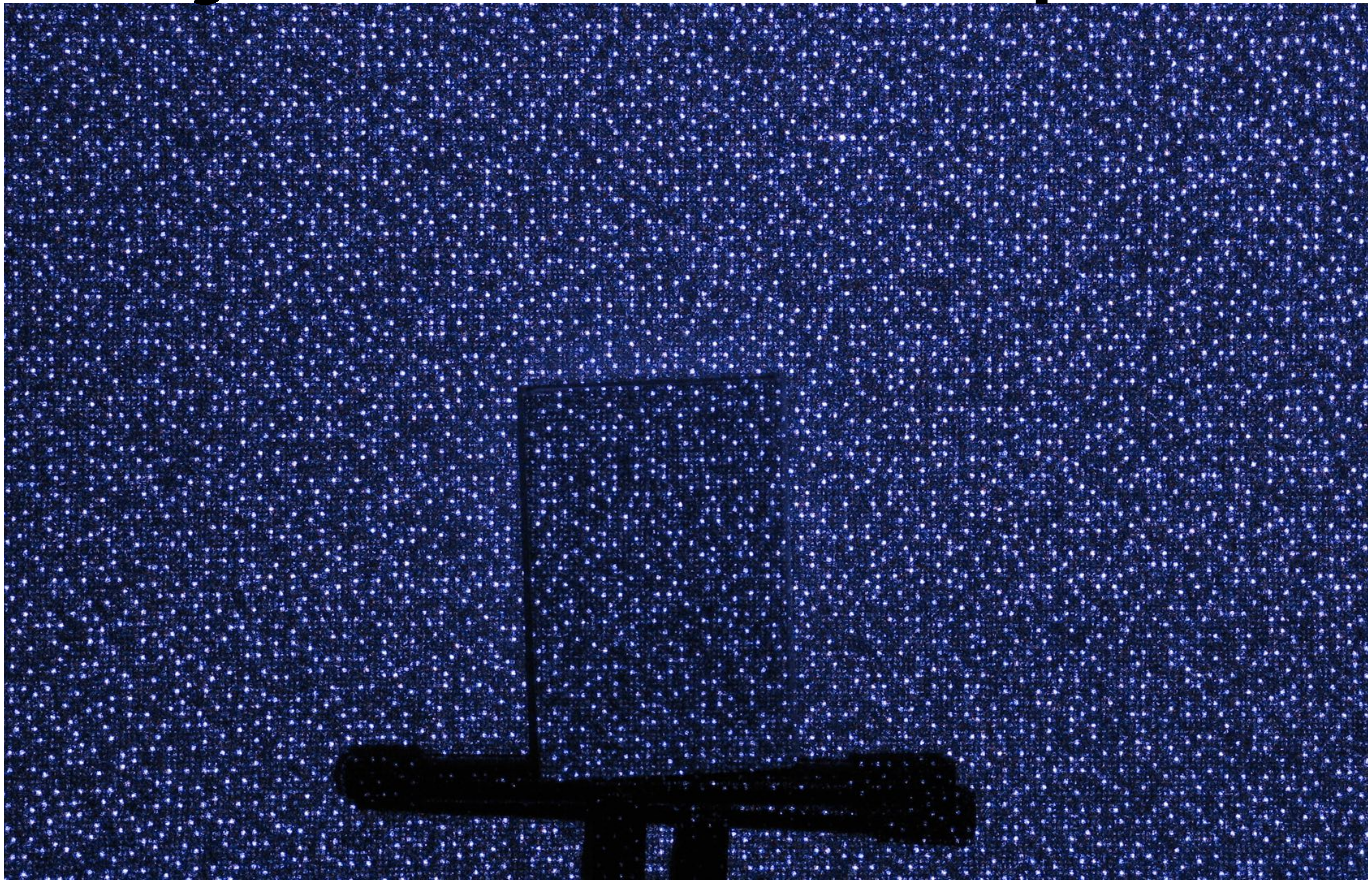
** Kinect returns 640x480 disparity image

Infrared image of Kinect illuminant output



Credit: www.futurepicture.org

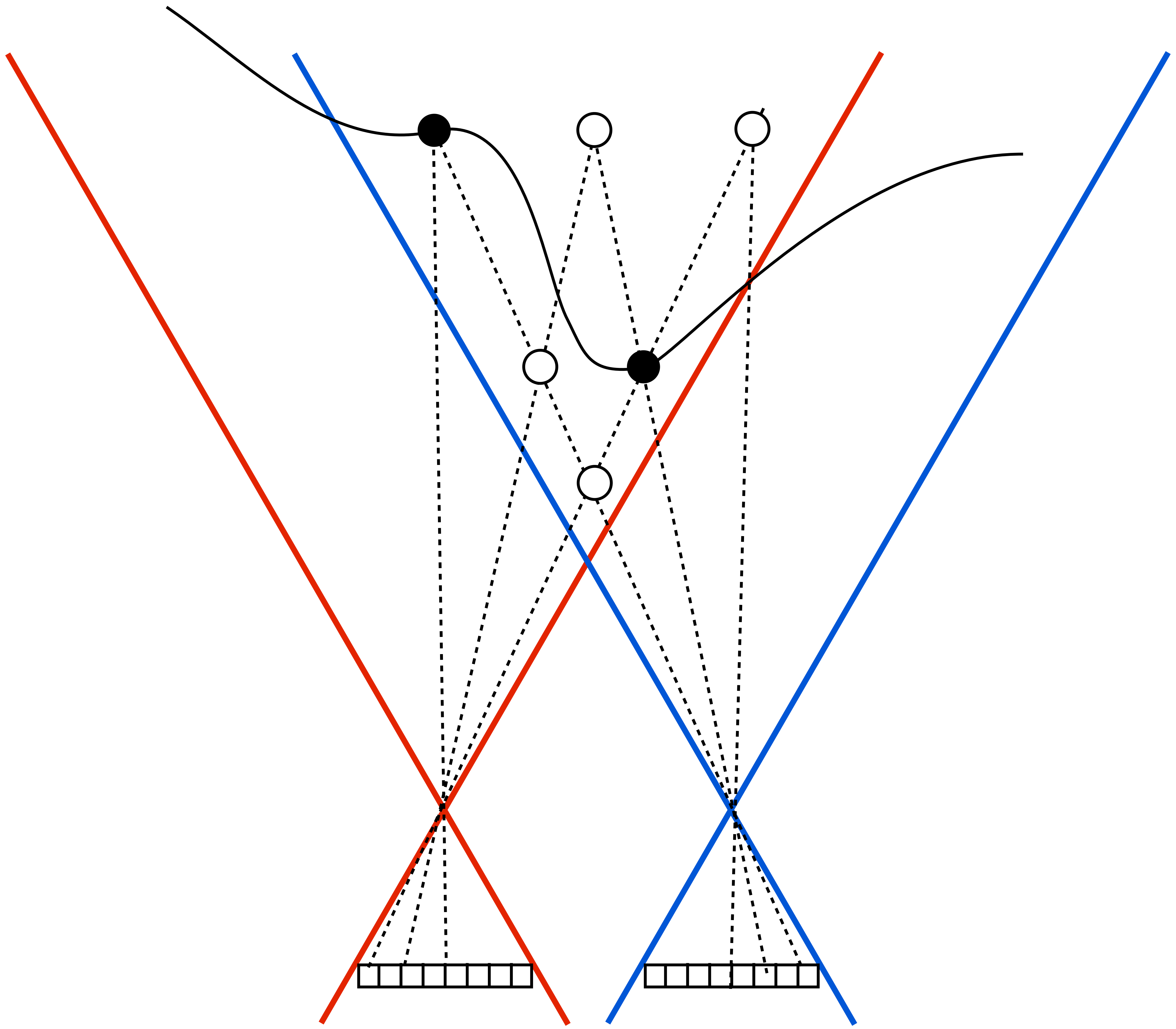
Infrared image of Kinect illuminant output



Credit: www.futurepicture.org

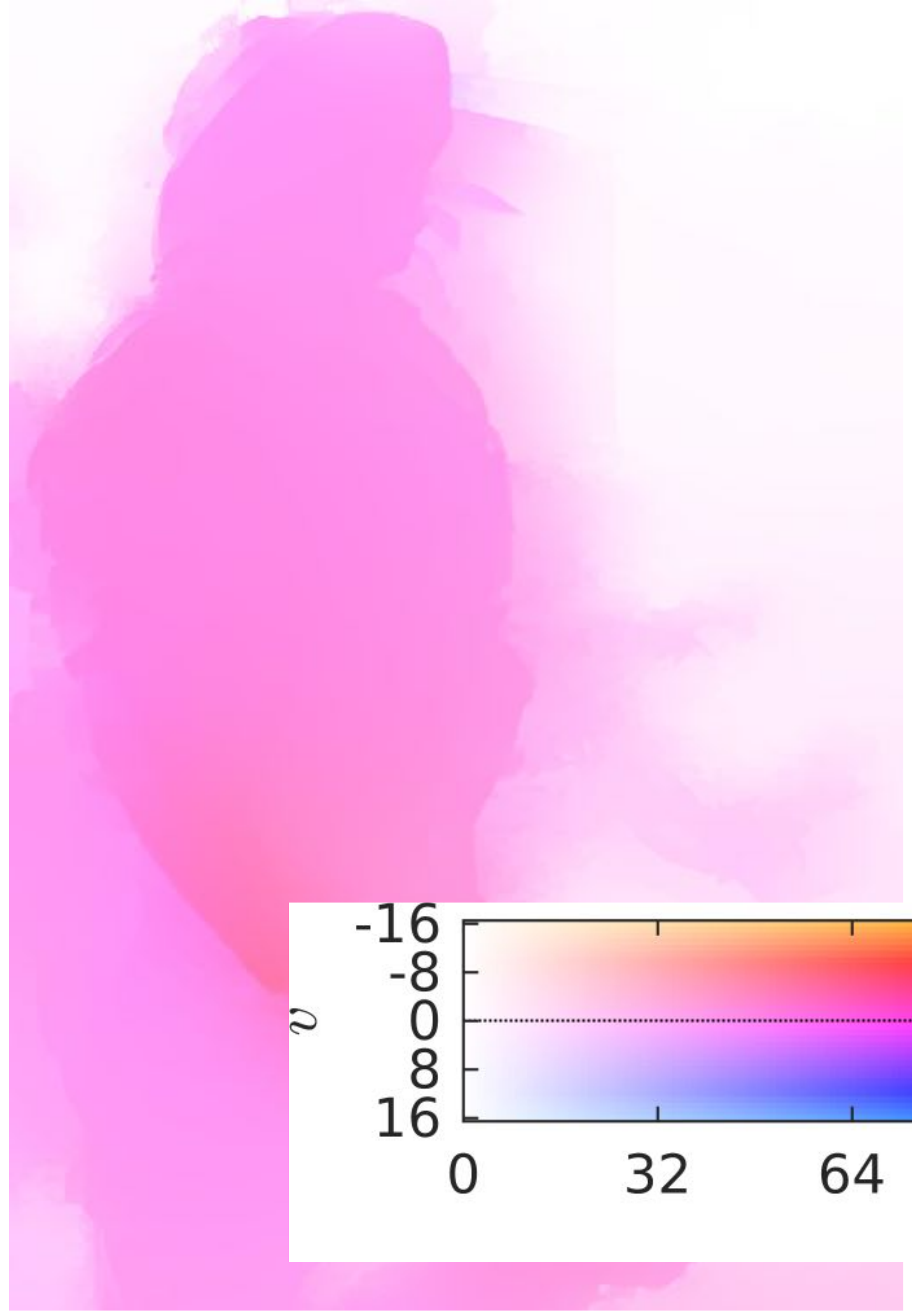
Correspondence problem

How to determine which pairs of pixels in image 1 and image 2 correspond to the same scene point?



Correspondence problem = compute "flow" between adjacent cameras

- For each pixel in frame from camera *i*, find closest pixel in camera *i+1*
- Google's Jump VR video pipeline uses a coarse-to-fine algorithm: align 32x32 blocks by searching over local window, then perform per-pixel alignment
  - Recall: H.264 motion estimation, HDR+ burst alignment (same correspondence challenge, but here we are aligning different perspectives taken the same time to estimate unknown scene depth, not estimating motion of camera or scene over time)
  - Additional tricks to ensure temporal consistency of flow over time (see papers)



2D Flow
(sat = u, hue = v)

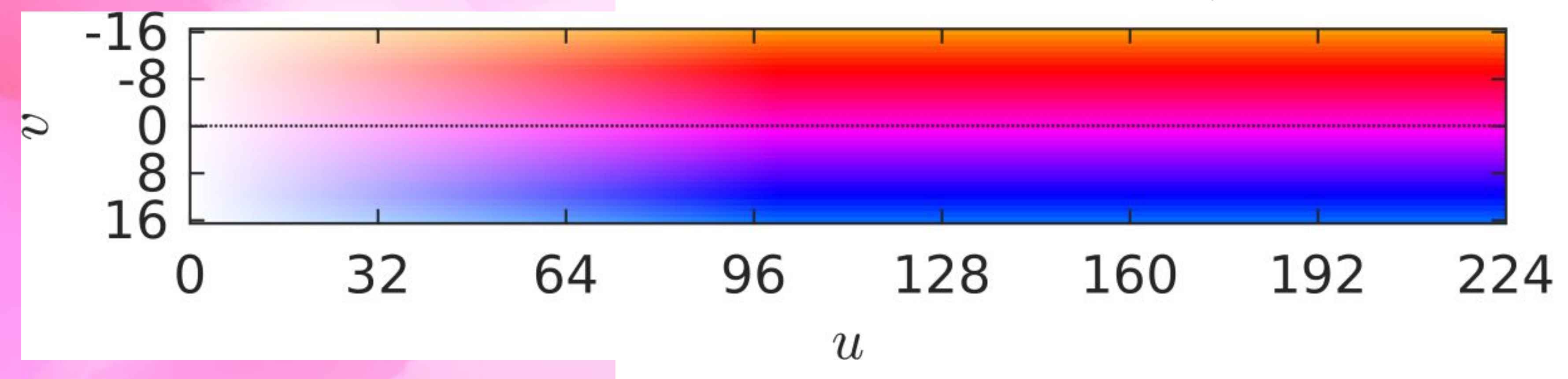
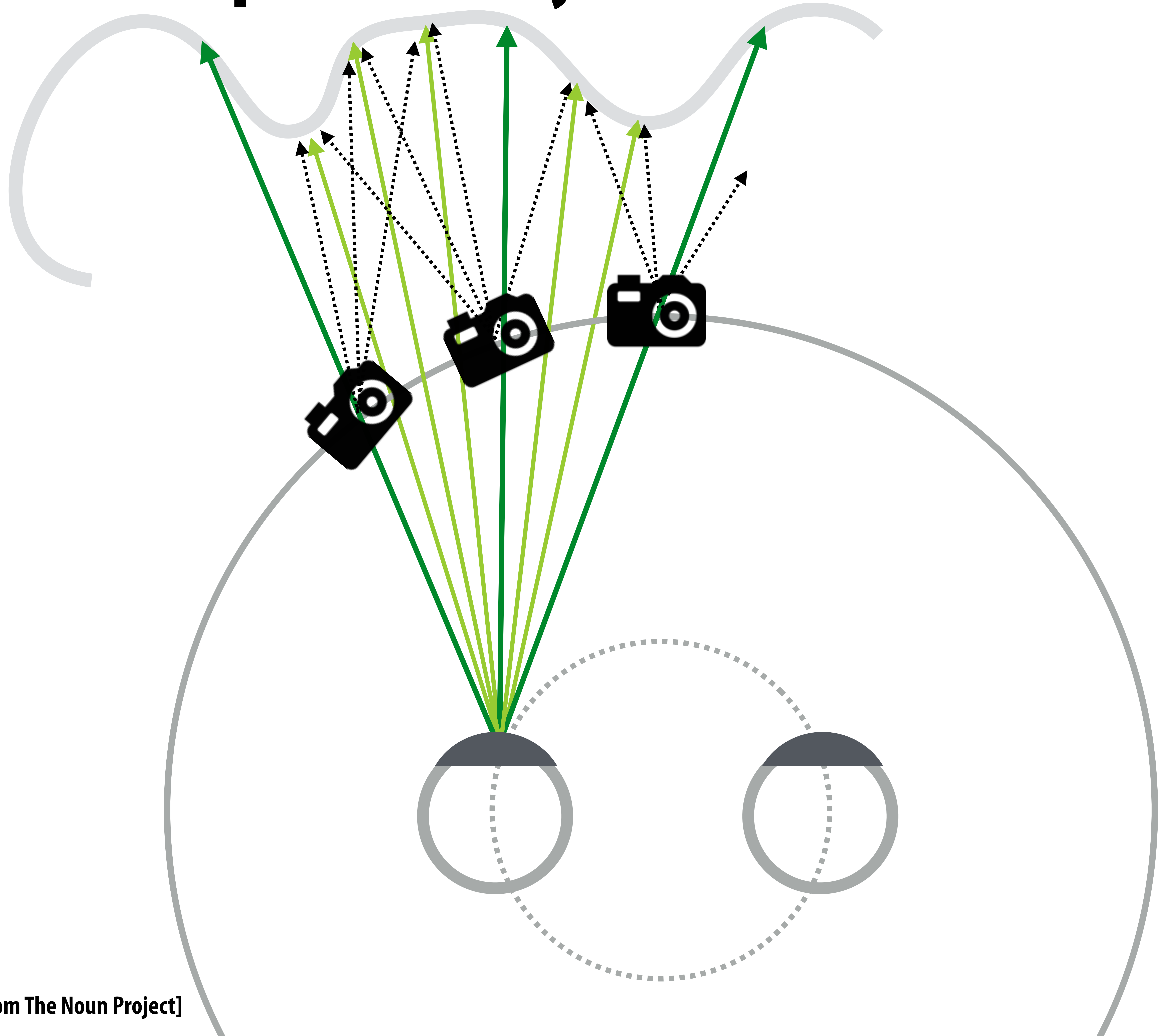


Image credit: Andersen et al. 2016

Left eye: with interpolated rays

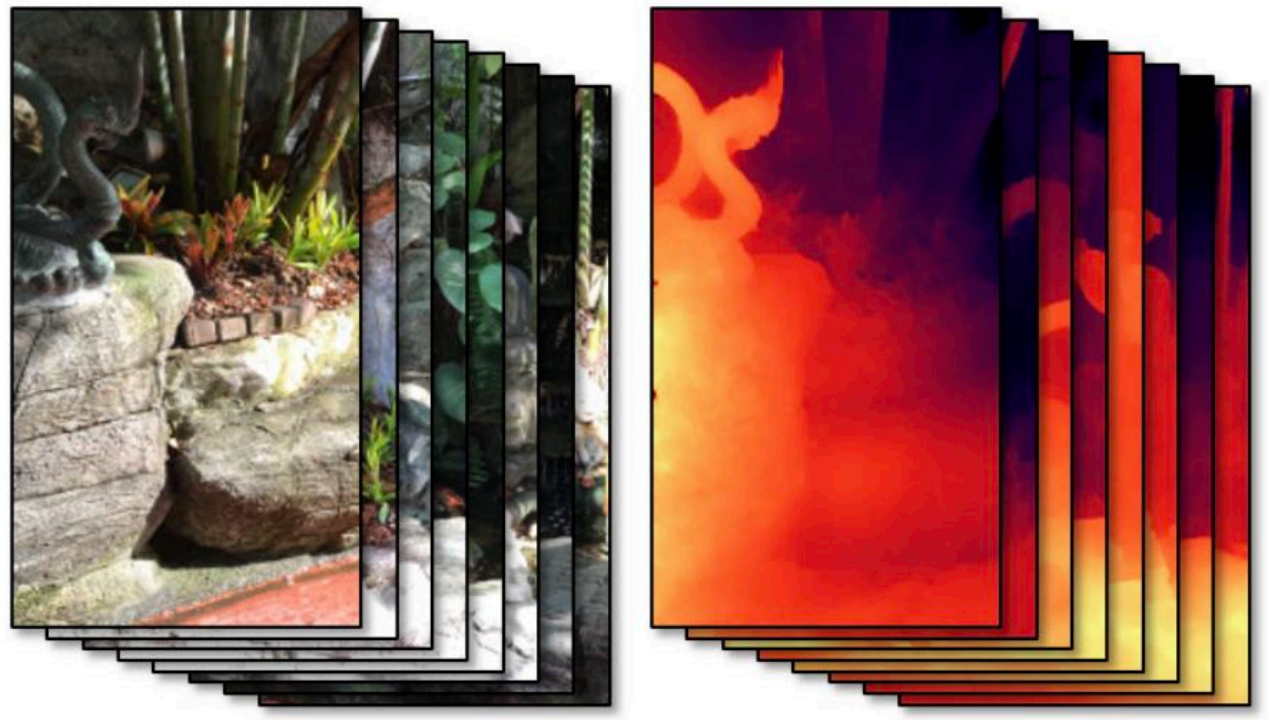


[Credit: Camera icon by Venkatesh Aiyulu from The Noun Project]

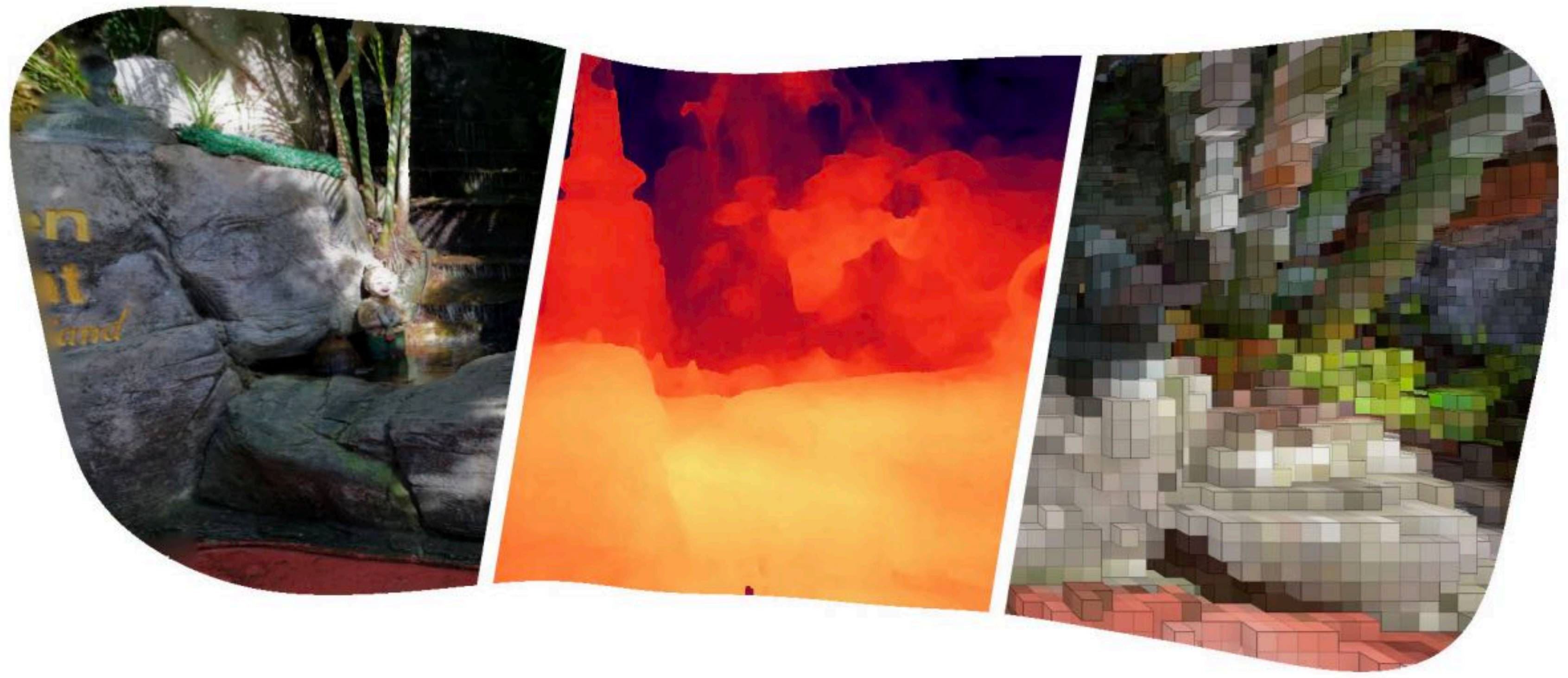
"Casual 3D photography"

- Acquisition: wave a smartphone camera around to acquire images of scene from multiple viewpoints
- Processing: construct 3D representation of scene from photos
  - Novel view synthesis performed by rendering a textured triangle mesh

Dual-camera Smartphone



Burst of photos + depth maps

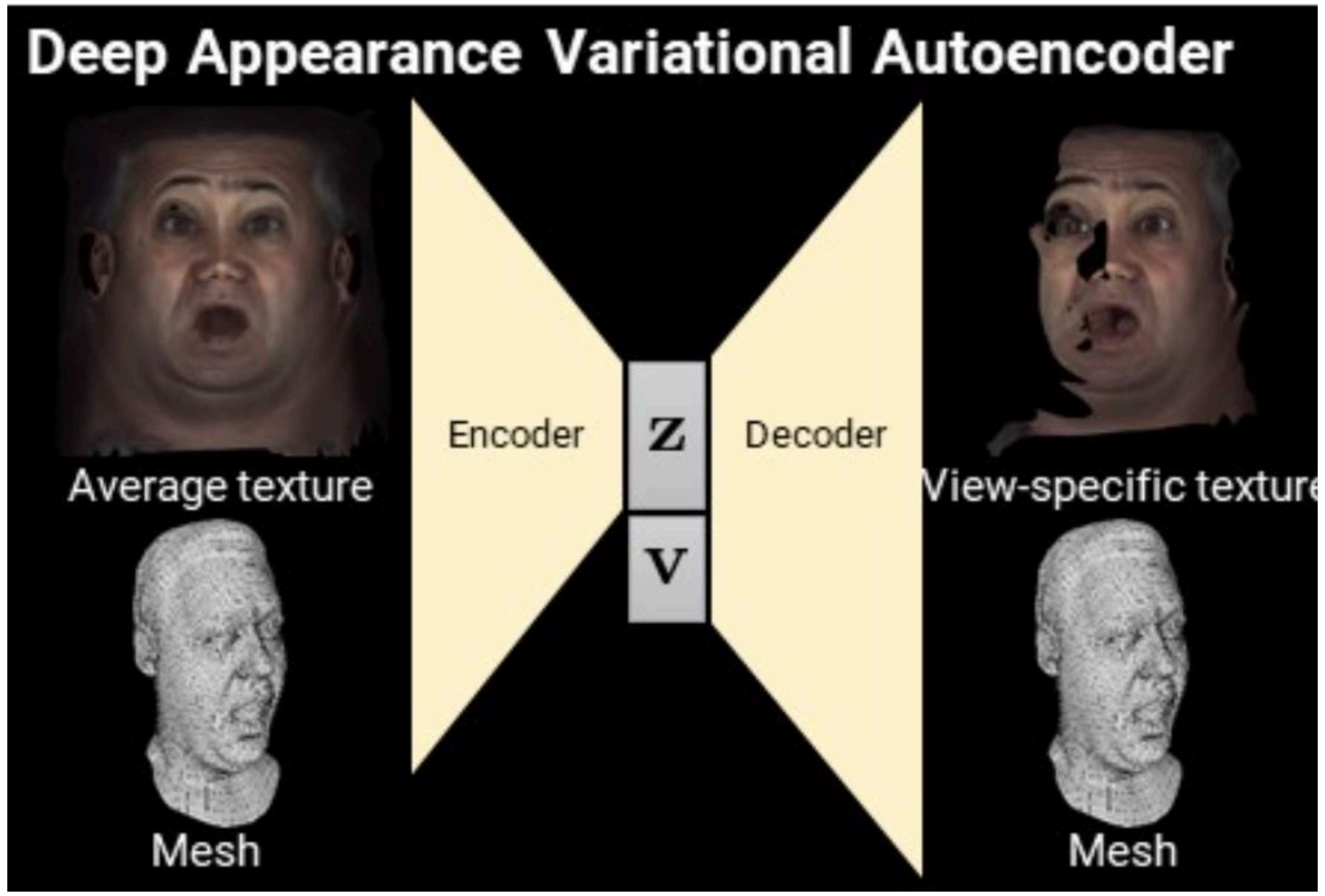


Stitch photos into depth panorama, create 3D mesh + textures, render during VR viewing

But it's hard to accurately estimate depth or geometry

Computer science in a nutshell:
Choose the right representation for the task at hand

Deep appearance models for face rendering [SIG18]



Given a bunch of views train a generative model (VAE) that given a view spits out a mesh and a view-dependent texture.

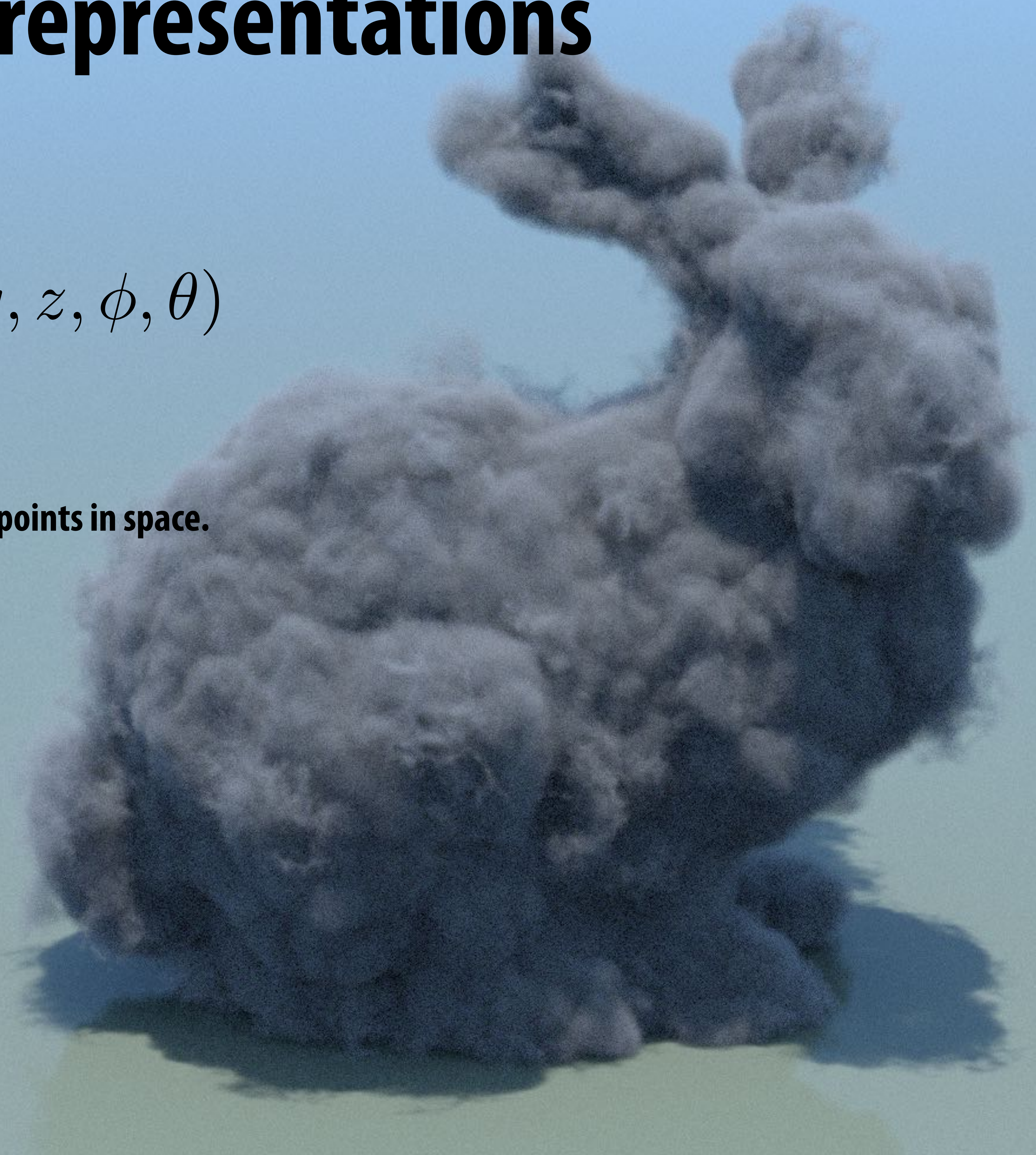
Starting to see an end-to-end argument emerge
1. Geometry estimation is hard.
2. And joint optimization of geometry and texture is good because we can learn to produce a view-dependent texture can compensates for position and topology errors in the output mesh.

Volumetric representations

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

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

Volume density and color at all points in space.



Representing rays

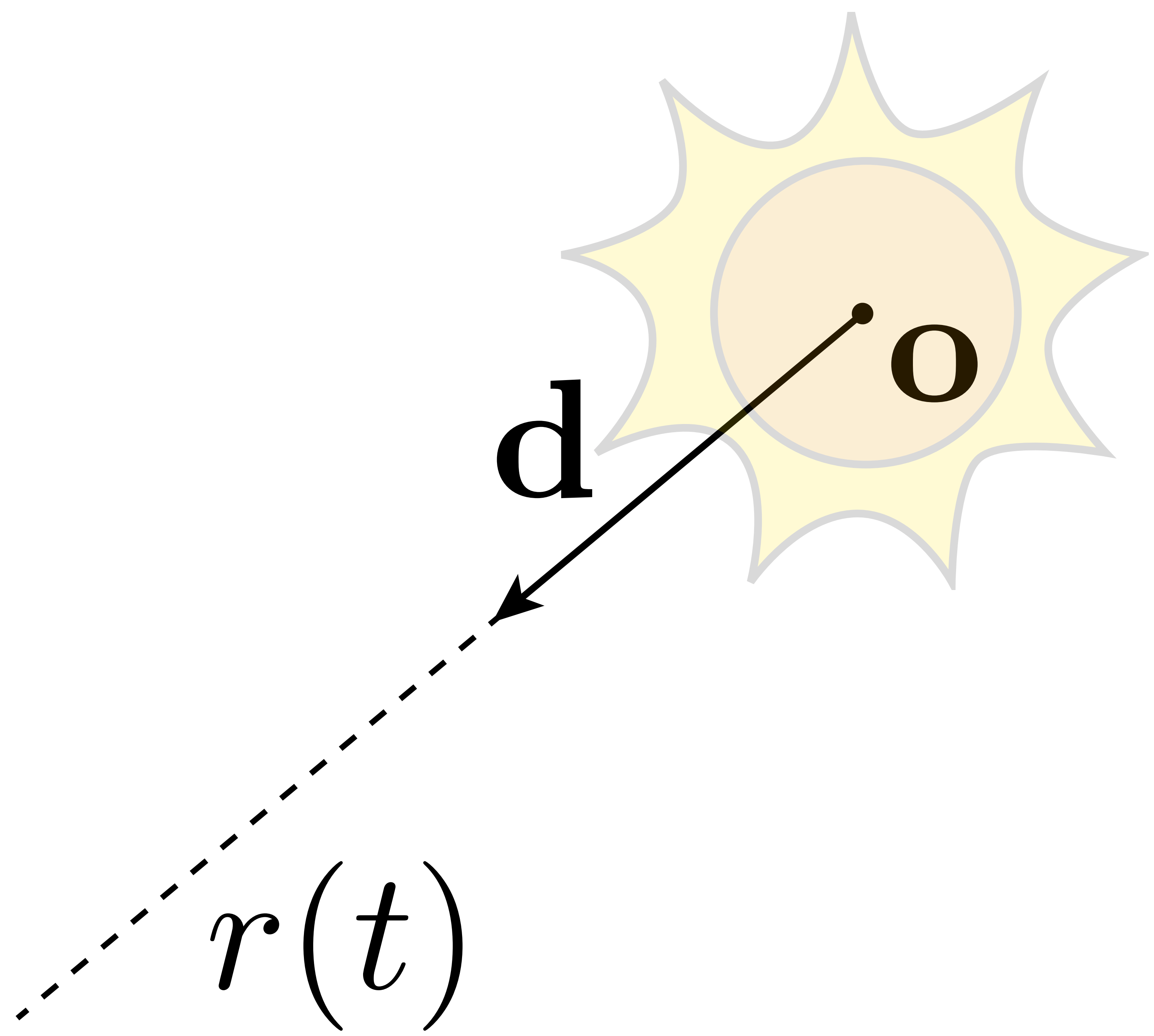
origin

unit direction

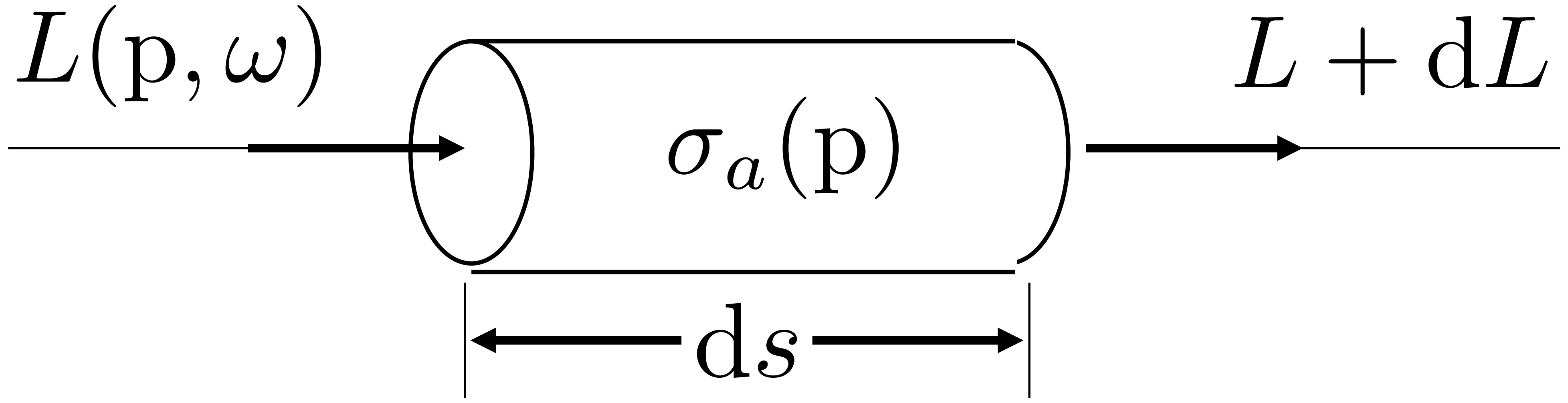
$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$$

point along ray

"Distance" or "time"



Absorption in a volume



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

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

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

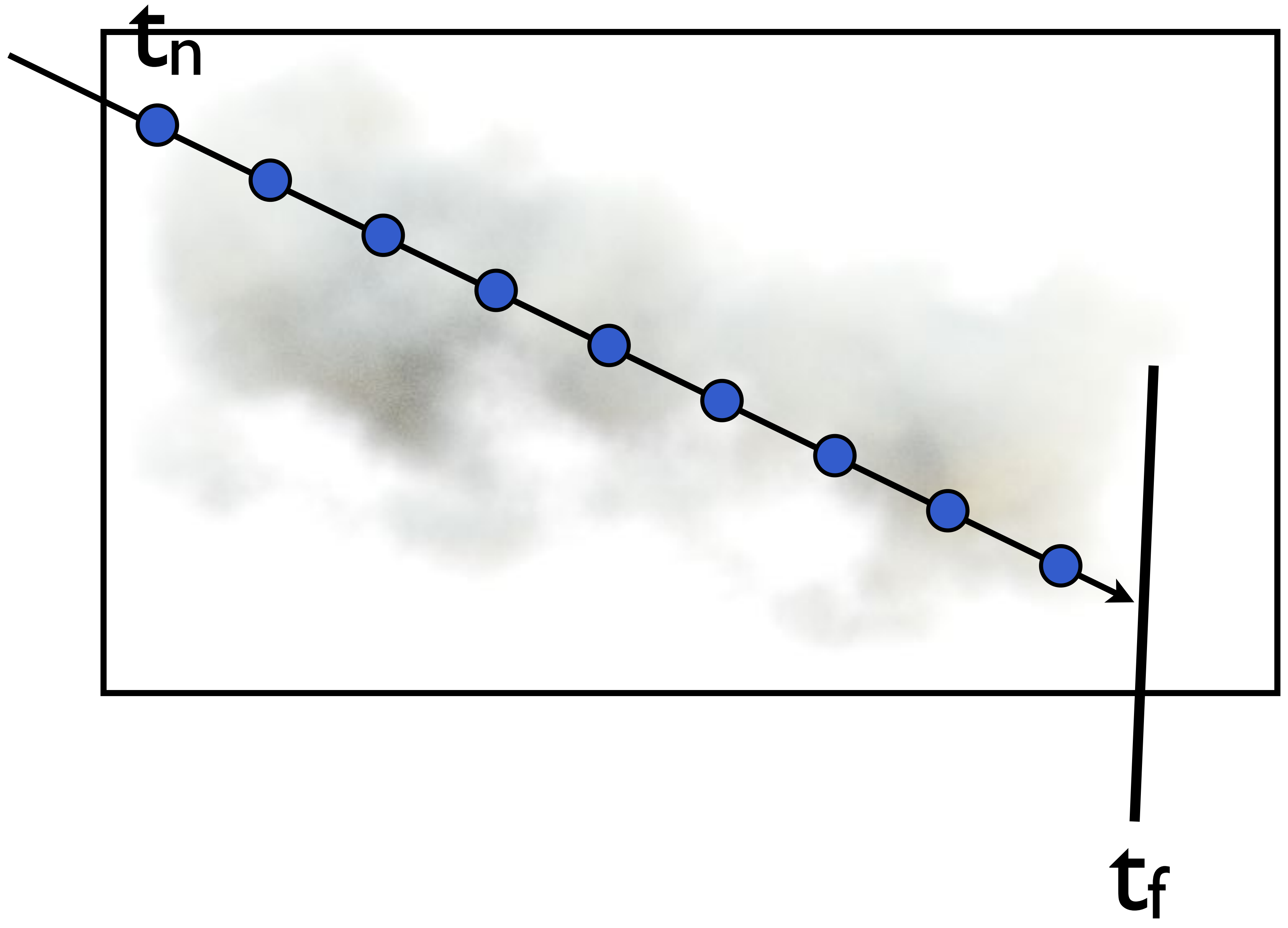
- $L(\mathrm{p}, \omega)$ **light energy (called "radiance") along a ray from *p* in direction *w***
- **Absorption cross section at point in space:** $\sigma_a(\mathrm{p})$
  - **Probability of being absorbed per unit length**
  - **Units: 1/distance**

Rendering volumes

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

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

Volume density and color at all points in space.
e.g., Values stored in a 3D grid



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

Regular 3D grid representation?

Consider storage requirements:
1024^3 cells

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

Now consider directional dependency on (ϕ, θ)
… much worse

Typical challenge of dense voxel representations: limited resolution

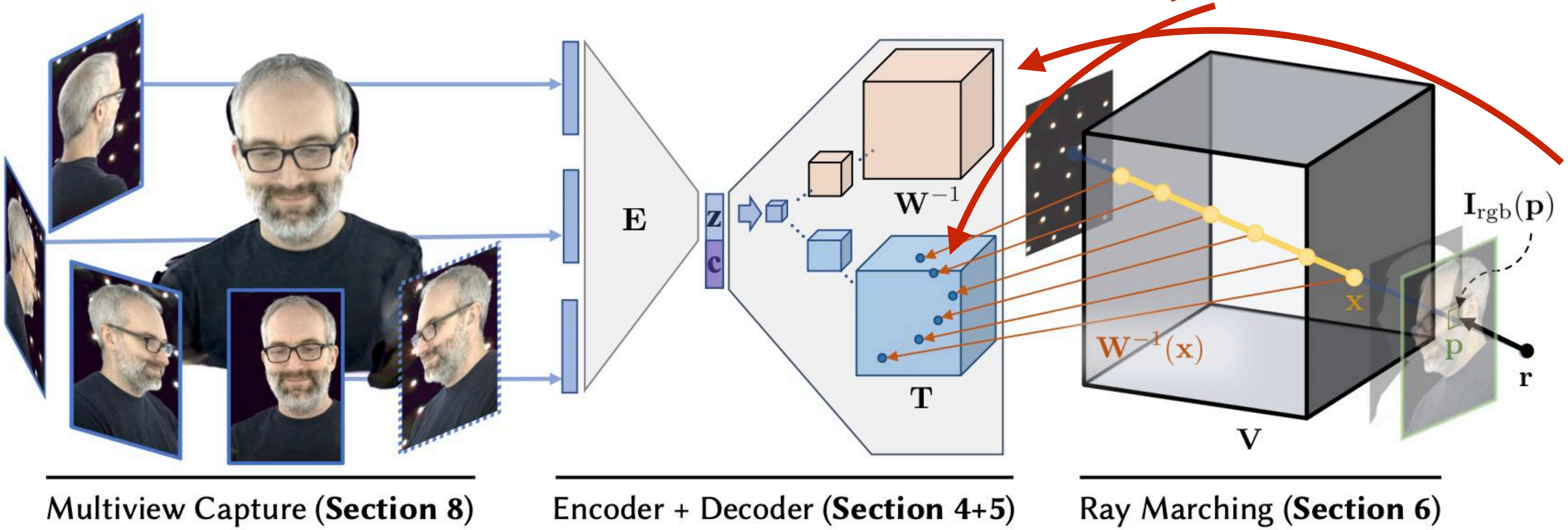
Credit: Voxel Ville NFT (voxelville.io)

Neural Volumes [SIGGRAPH 19]

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

One possible model for the volume function $\mathbf{V}(\mathbf{x}; \mathbf{z})$ at point $\mathbf{x}$ with state $\mathbf{z}$, is an implicit one with a series of fully connected layers with non-linearities. A benefit of this approach is that we're not restricted by voxel grid resolution or storage space. Unfortunately, in practice an MLP requires prohibitive size to produce high-quality reconstructions. We must also evaluate the MLP at every step along each ray in the ray-marching process (see §6), imposing an equally restrictive upper bound on the MLP complexity for real-time applications.

We can't represent the voxel grid with an MLP (not enough resolution), so we just have a convolutional decoder that operates on D³)

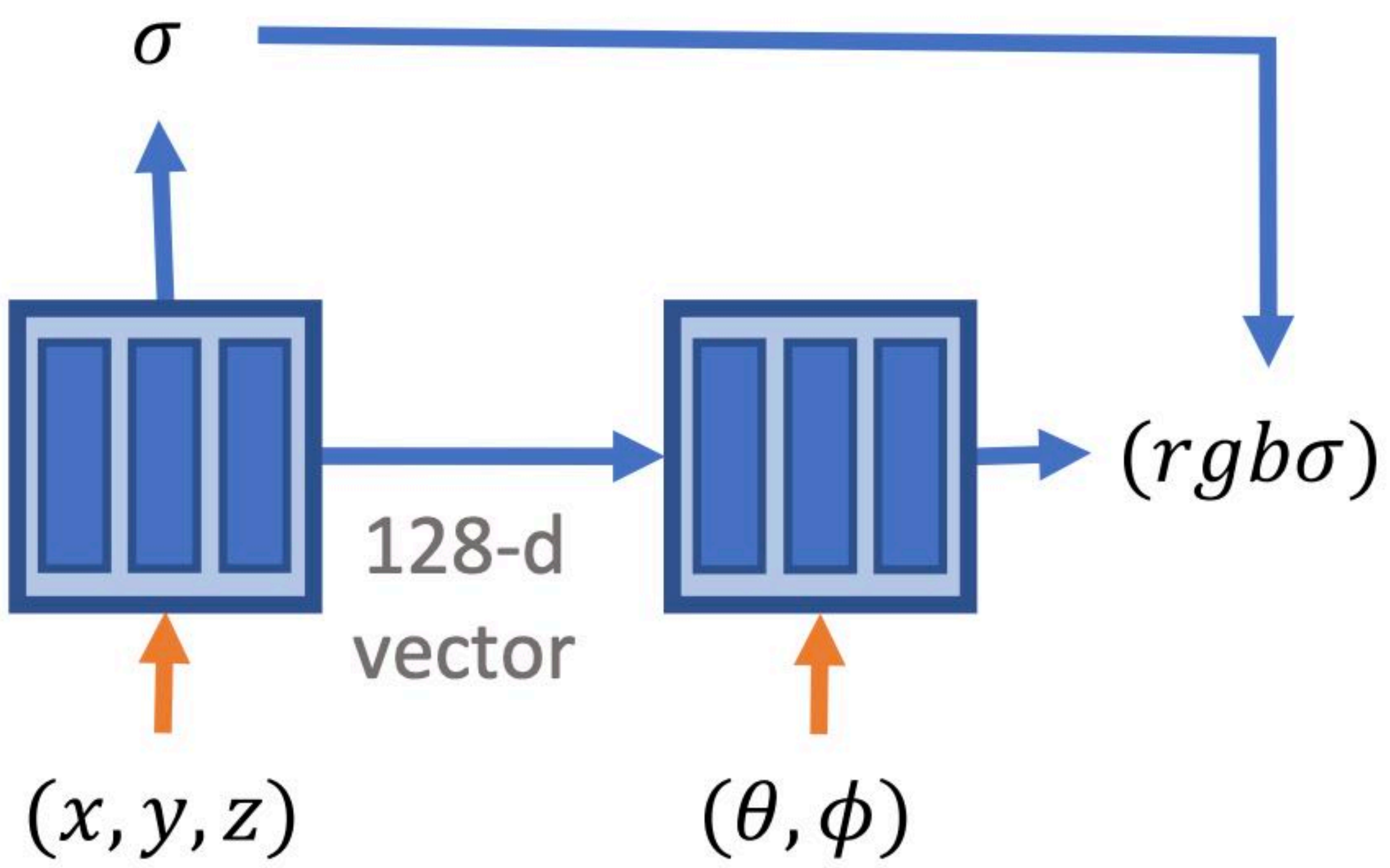


And since a 3D dense voxel grid is not high-enough res to avoid blockiness, we'll also output a warping function W-1 to adjust space when volume rendering to reduce rendering loss.

Learning (compressed) representations

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

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



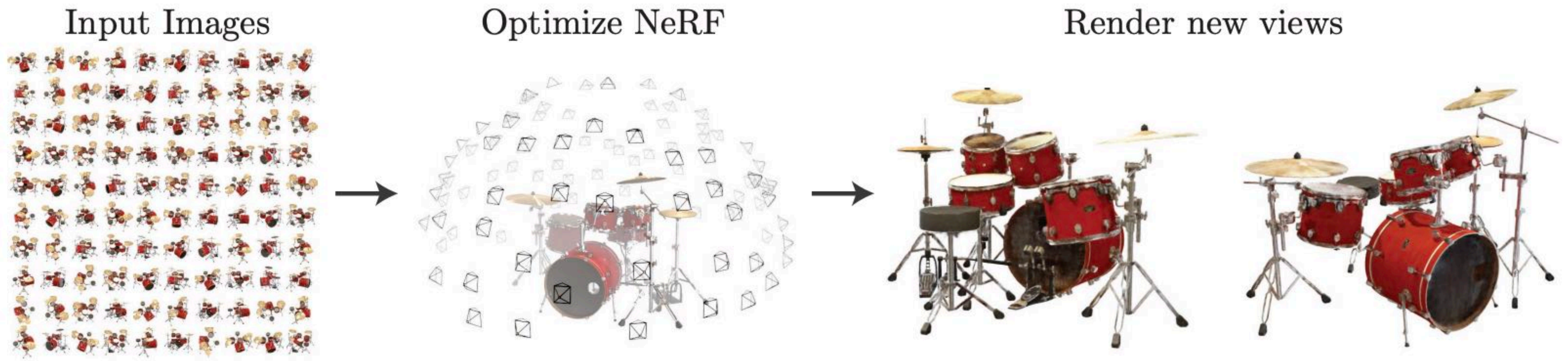
Learning better (more compressed) representations

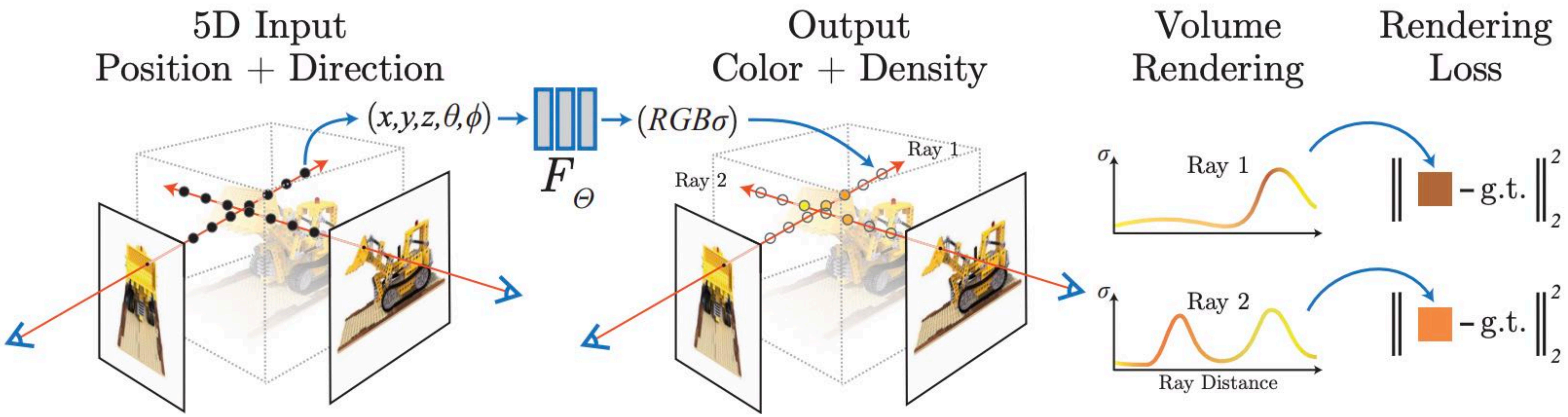
- **Why not just learn an approximation to the continuous function:**

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

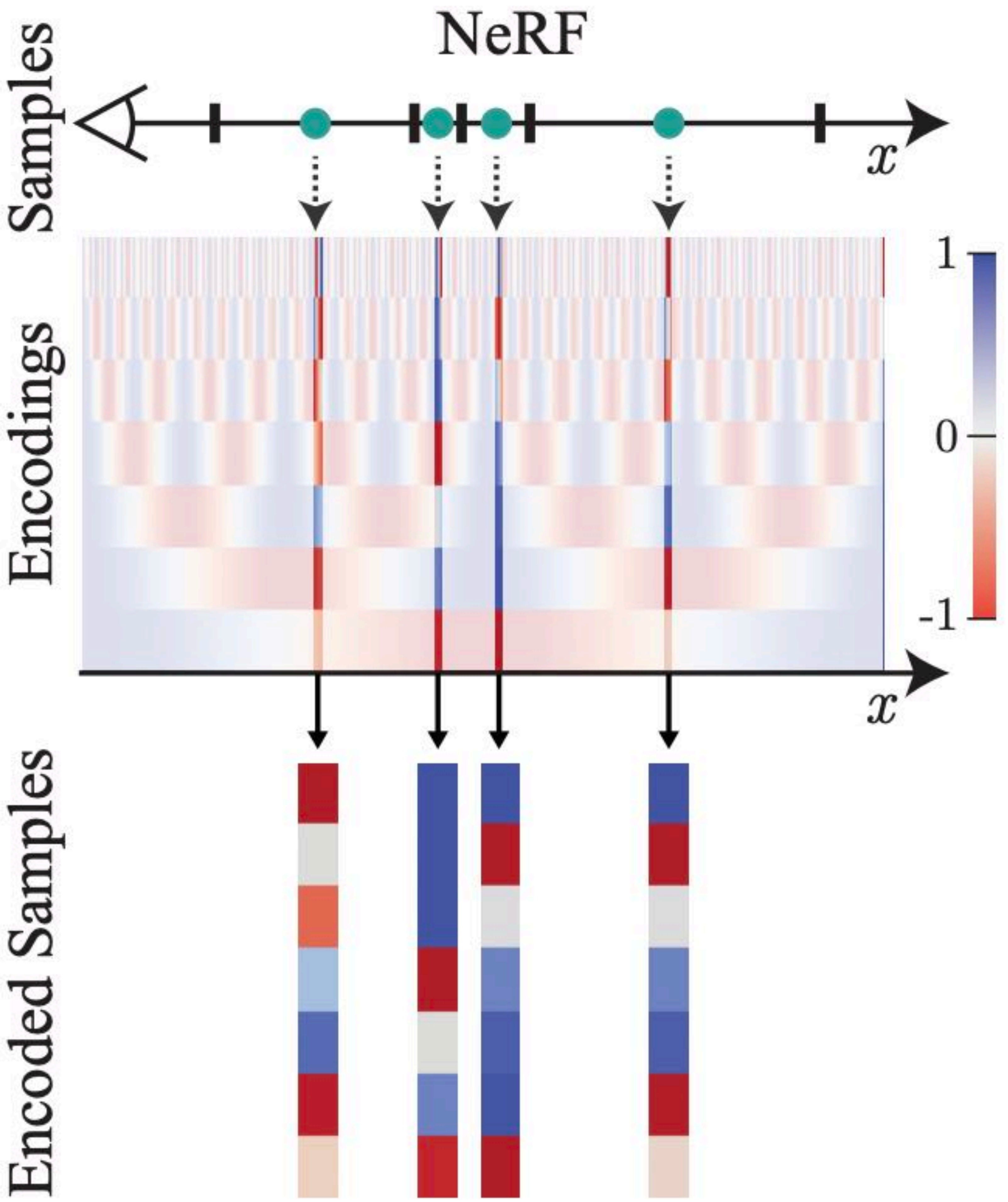
- **For all photos of the scene that we have, use $F_\theta(\mathrm{p}, \omega)$ to volume render the scene from the known viewpoint.**
- **Loss is difference between rendered view and actual photo.**
- **Update θ using standard optimization techniques (SGD)**

Learning neural radiance fields (NeRF)





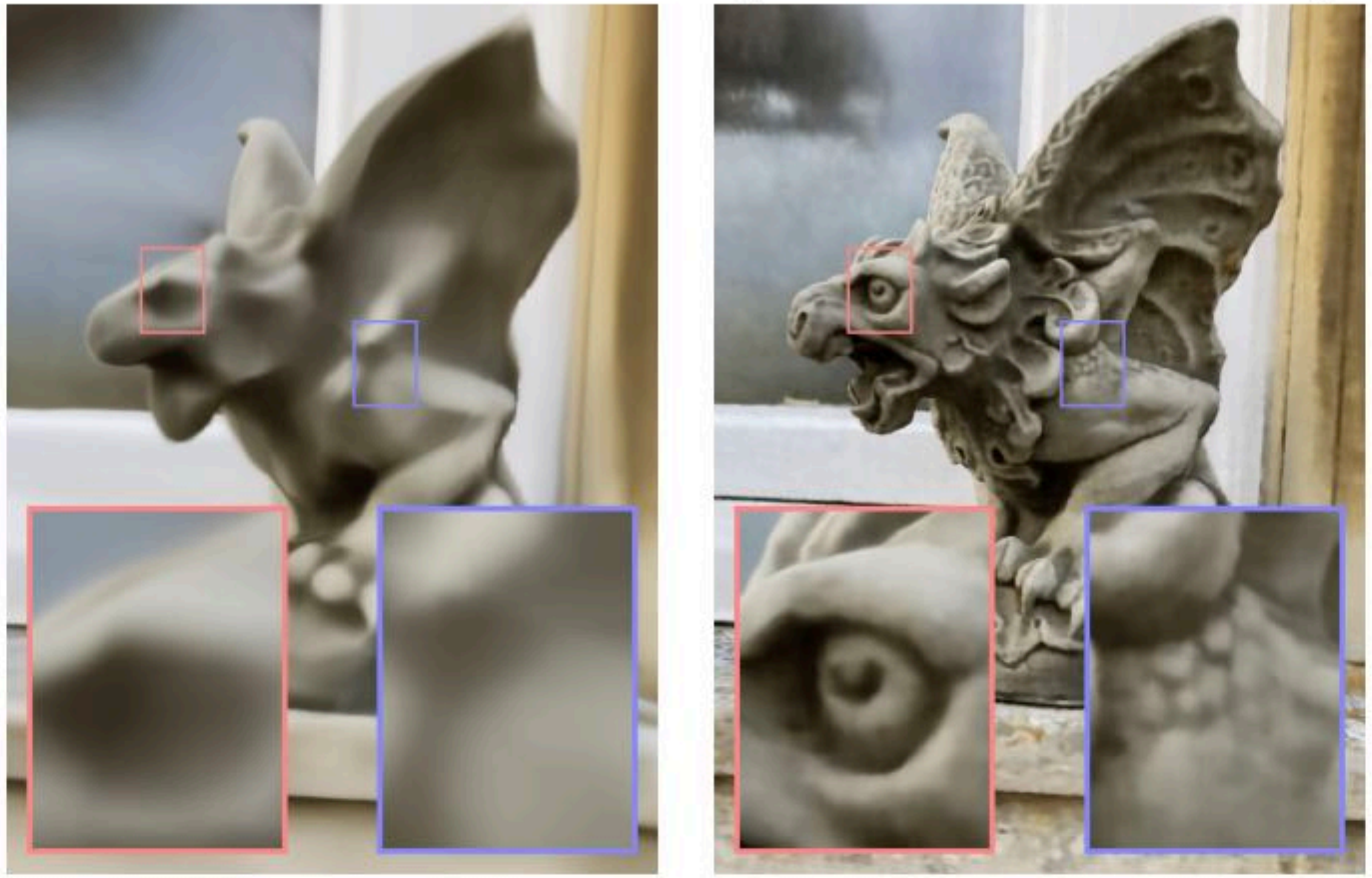
Frequency encoding of (x,y,z,phi,theta)



$$\gamma(\mathbf{x}) = \left[\sin(\mathbf{x}), \cos(\mathbf{x}), \ldots, \sin\left(2^{L-1}\mathbf{x}\right), \cos\left(2^{L-1}\mathbf{x}\right)\right]^{\mathrm{T}}$$

(a) No encoding

(b) Frequency [Mildenhall et al. 2020]



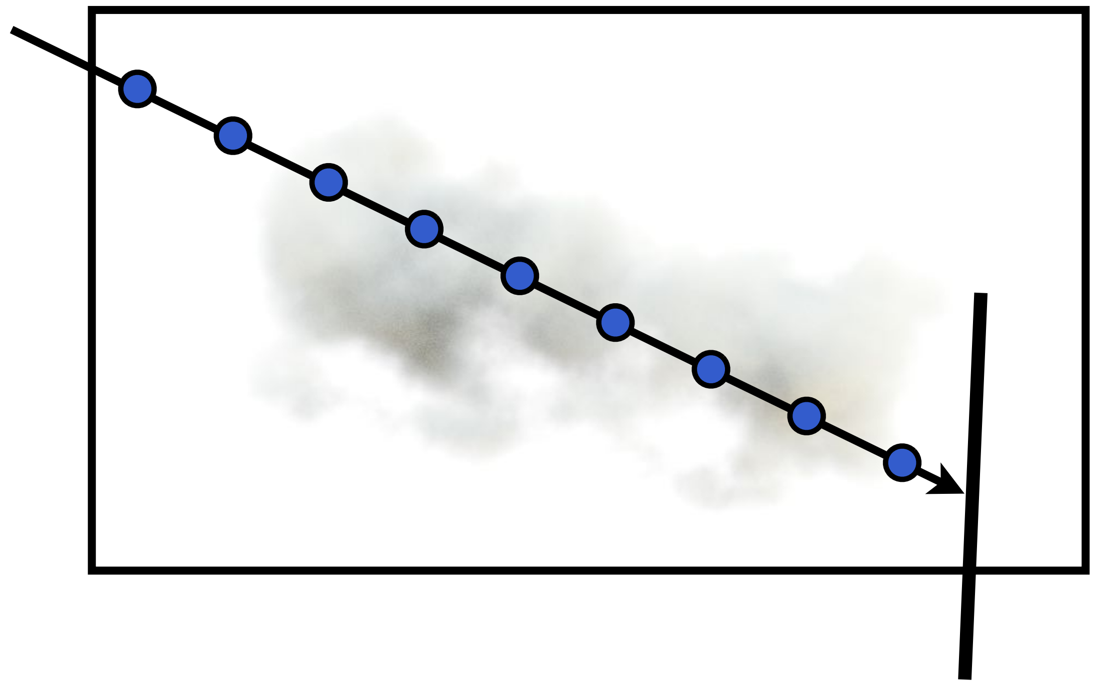
See: "On the spectral bias of neural networks" In: ICML (2018)

Key ideas of volumetric representations in this context

- Do not need to reconstruct/estimate triangle mesh surface geometry
- Volume rendering is easily differentiable, so easy to update $F_\theta(\mathrm{p}, \omega)$
- The DNN used to represent $F_\theta(\mathrm{p}, \omega)$ is a compact representation of this high-dimensional function.
  - Alternative representation than a dense voxel grid.

What just happened?

- Continuous coordinate-based representation vs regular grid: MLP "learns" how to use its weights to produce high-resolution output where needed… given input data
  - 3D grid: RGBA values at each grid location
  - MLP: fixed set of weights, weights can get "allocated" different regions of space via learning



- Compact representation: trades-off space for expensive rendering
  - Good: a few MBs = effectively very high resolution dense grid
  - Bad: must evaluate MLP every step
    - And it's a "big" MLP (8-layer x 256)
  - Bad: must step densely (because we don't know where the surface is)

MLP must do real work to associate weights with 5D locations

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

NeRF demos

Where we stand

- **Good:**
  - We can recover [surprisingly] high quality $F_\theta(\mathrm{p}, \omega)$ from a relatively sparse set of photos
  - What do you mean by "high quality"?
    - High quality novel view synthesis
    - Visualization of a occupancy reveals high quality depth maps

- **Bad: (high cost!)**
  - Long training times
  - Expensive MLP at rendering time (far from interactive rendering)

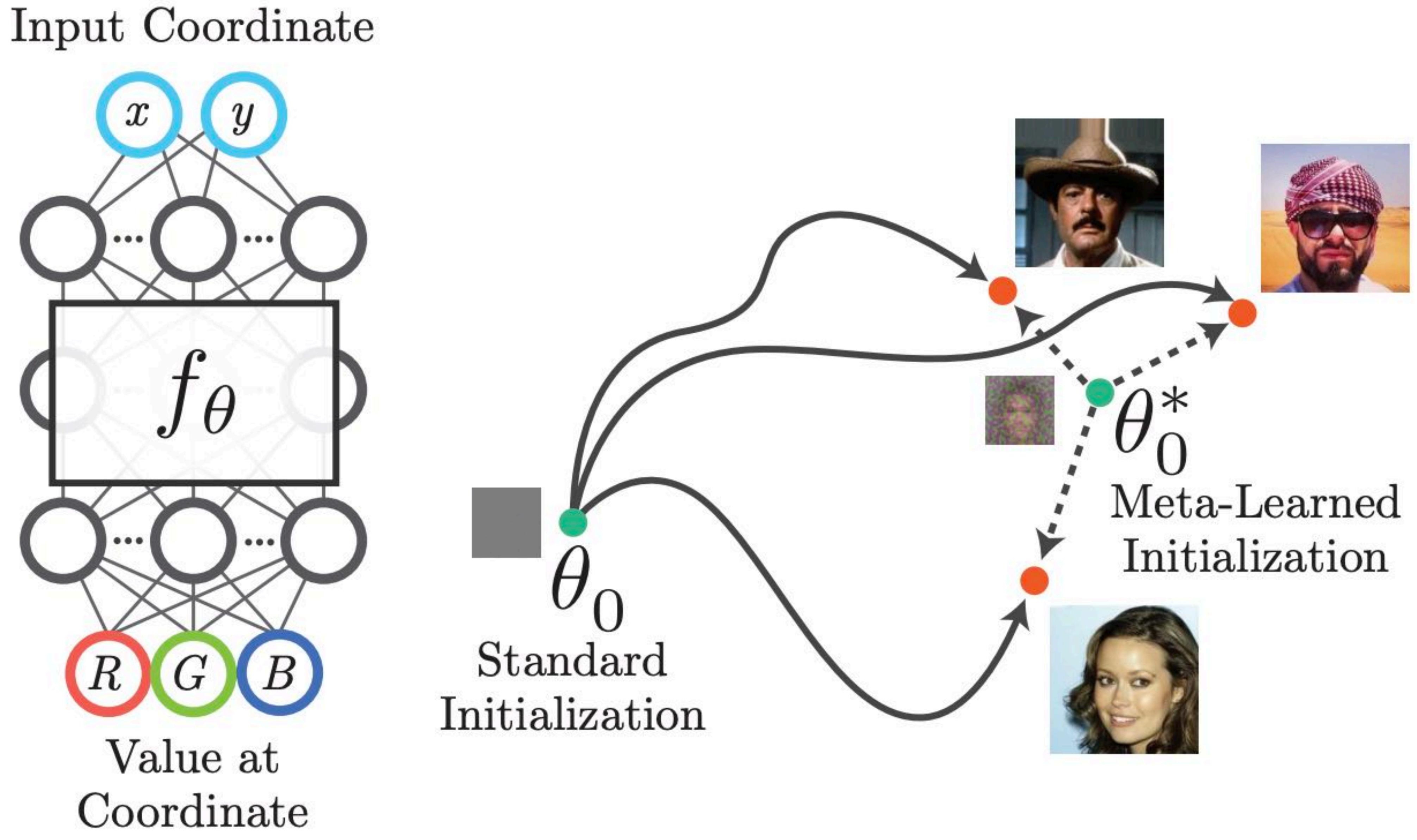
Sound familiar? 🤔

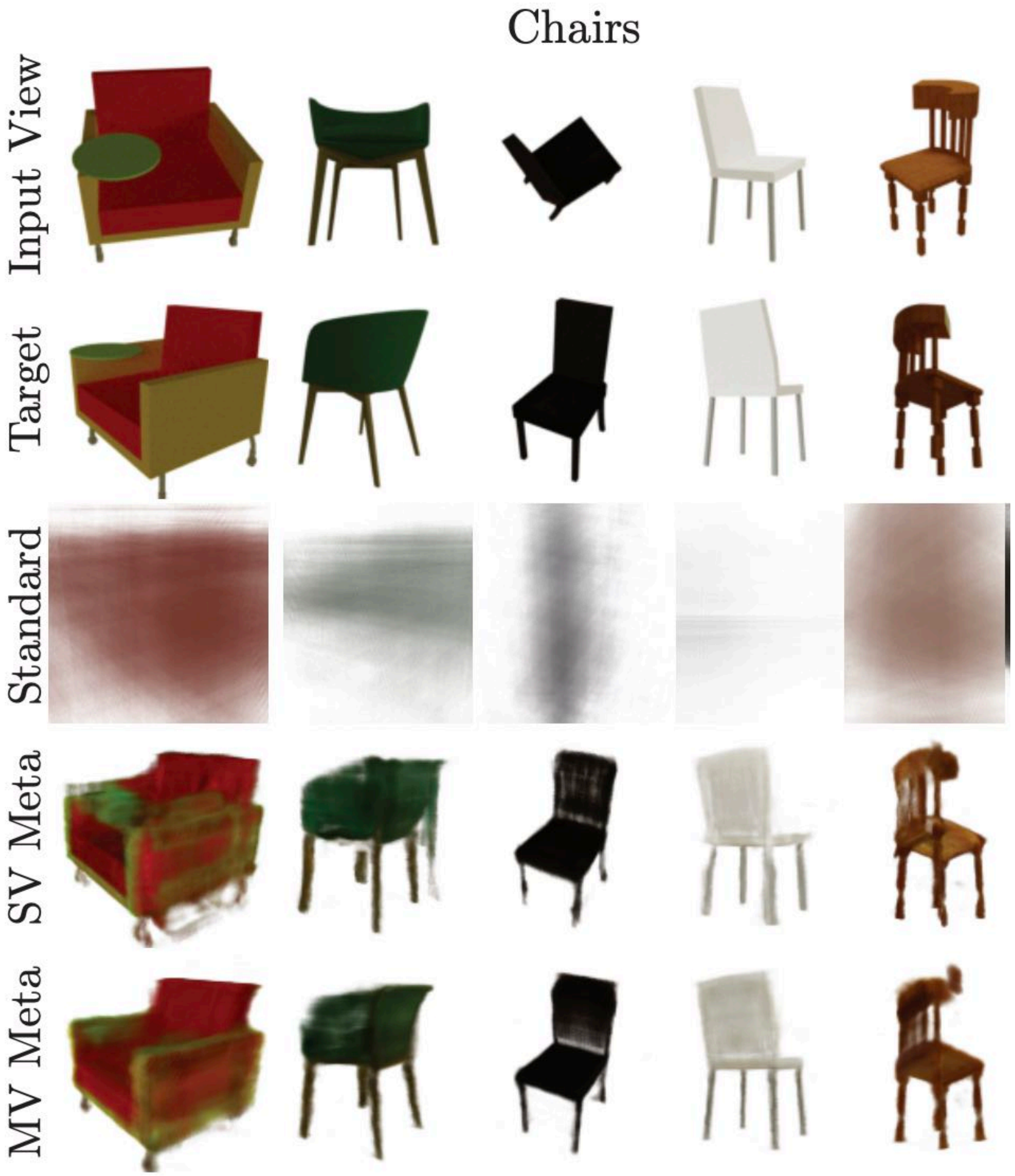
- Amazing new ML algorithm emerges
  - Amazing effectiveness of brute-force optimization, uses few structural priors…
  - Extremely high cost to train or evaluate

- What should we do?
  - Build faster DNN accelerator hardware?
  - Parallelize onto different machines?
  - ???

Improving NeRF training

- **One idea: improve training speed AND reduce numl**





Improving rendering performance

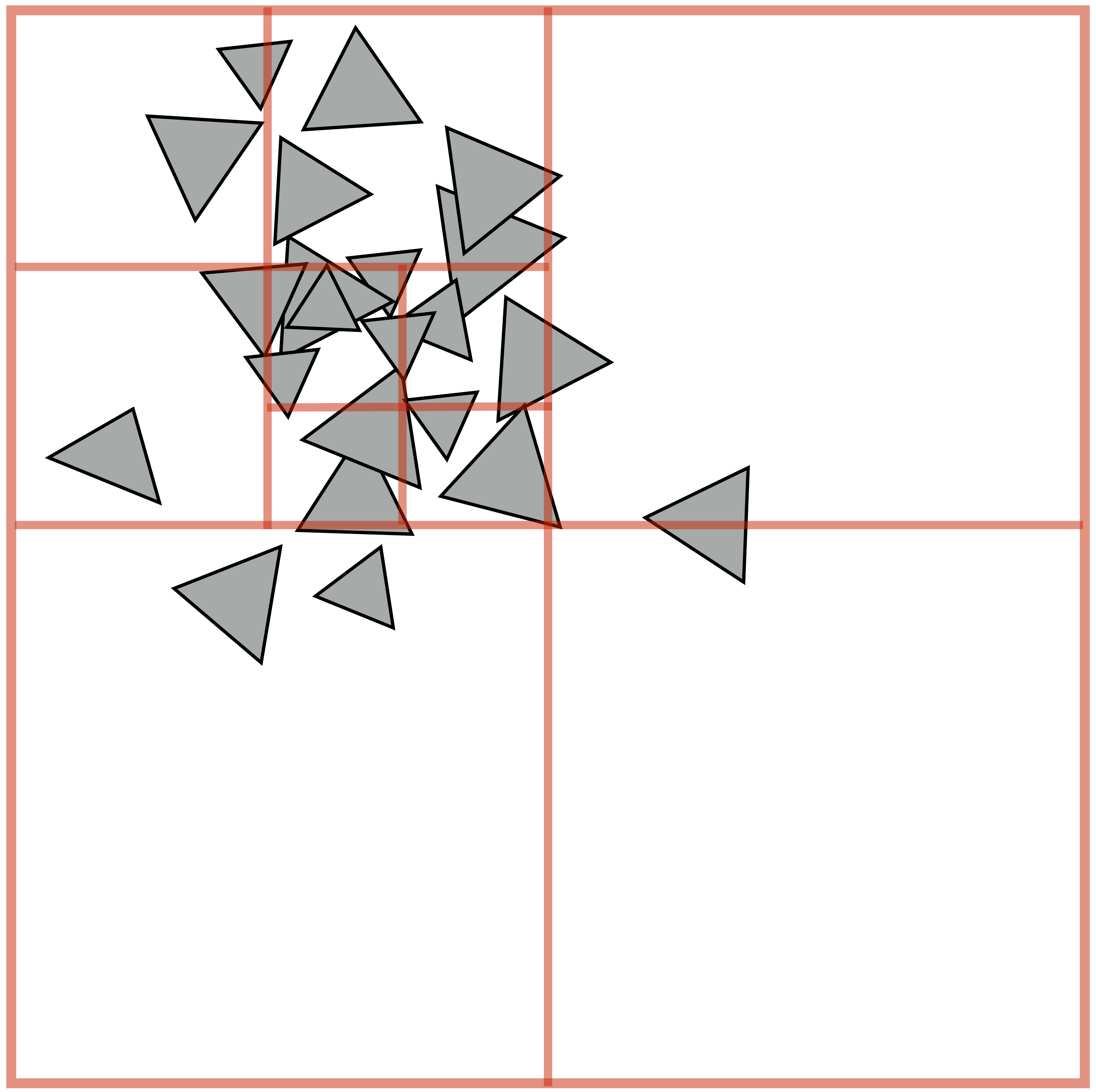
- **Main idea: move to a different point in the compression-compute trade-off space by moving back toward traditional computer graphics data structures**

- **Main ideas:**
  - **Avoid stepping densely through space during rendering (it is costly to evaluate the MLP and find density = 0)**
  - **Shrink the size of the MLP**
  - **Avoid evaluating the MLP when you can**

Background part 1: quad-tree / octree

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

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



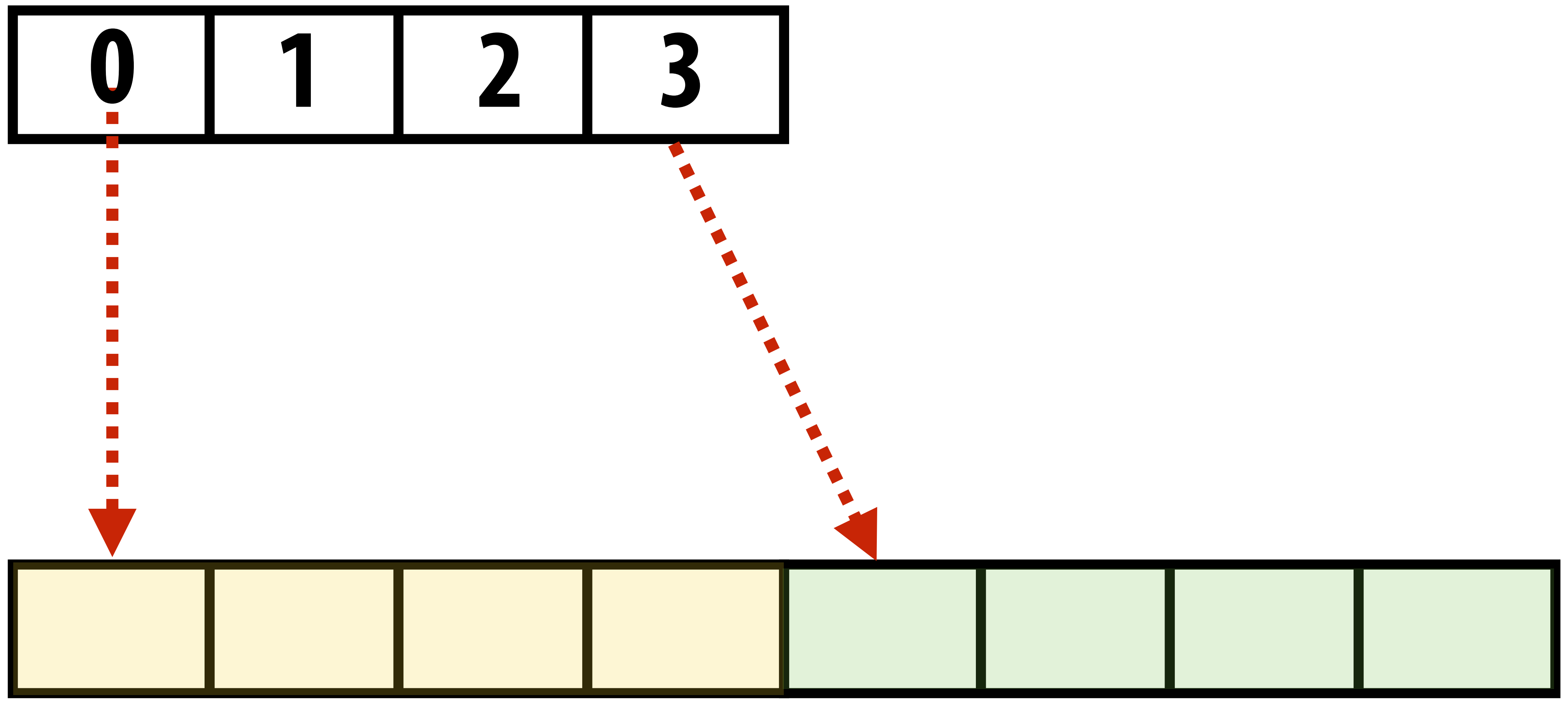
Like uniform grid: easy to build (don't have to choose partition planes)

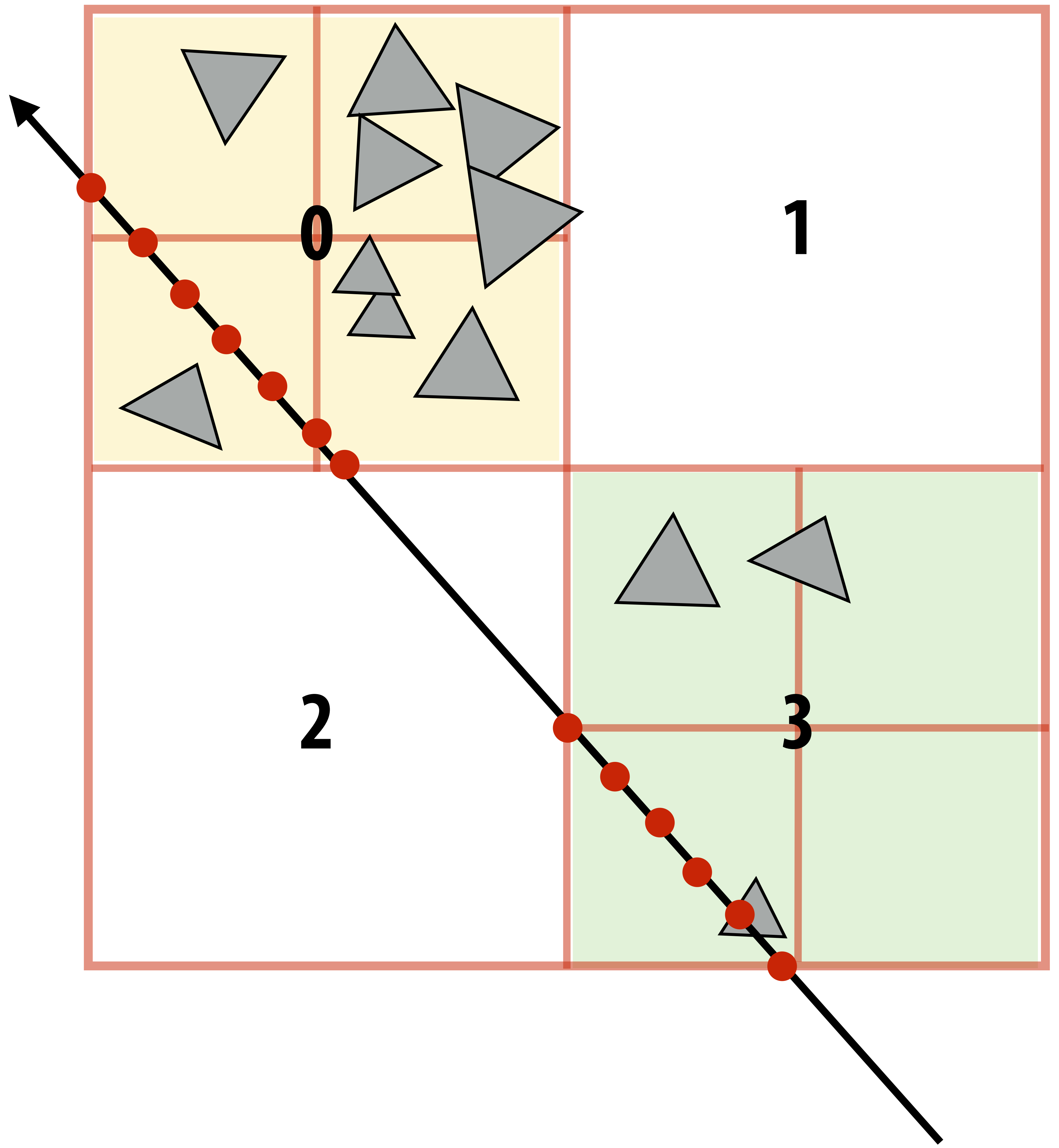
Has greater ability to adapt to location of scene geometry than uniform grid.

Simple two-level sparse quad tree

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

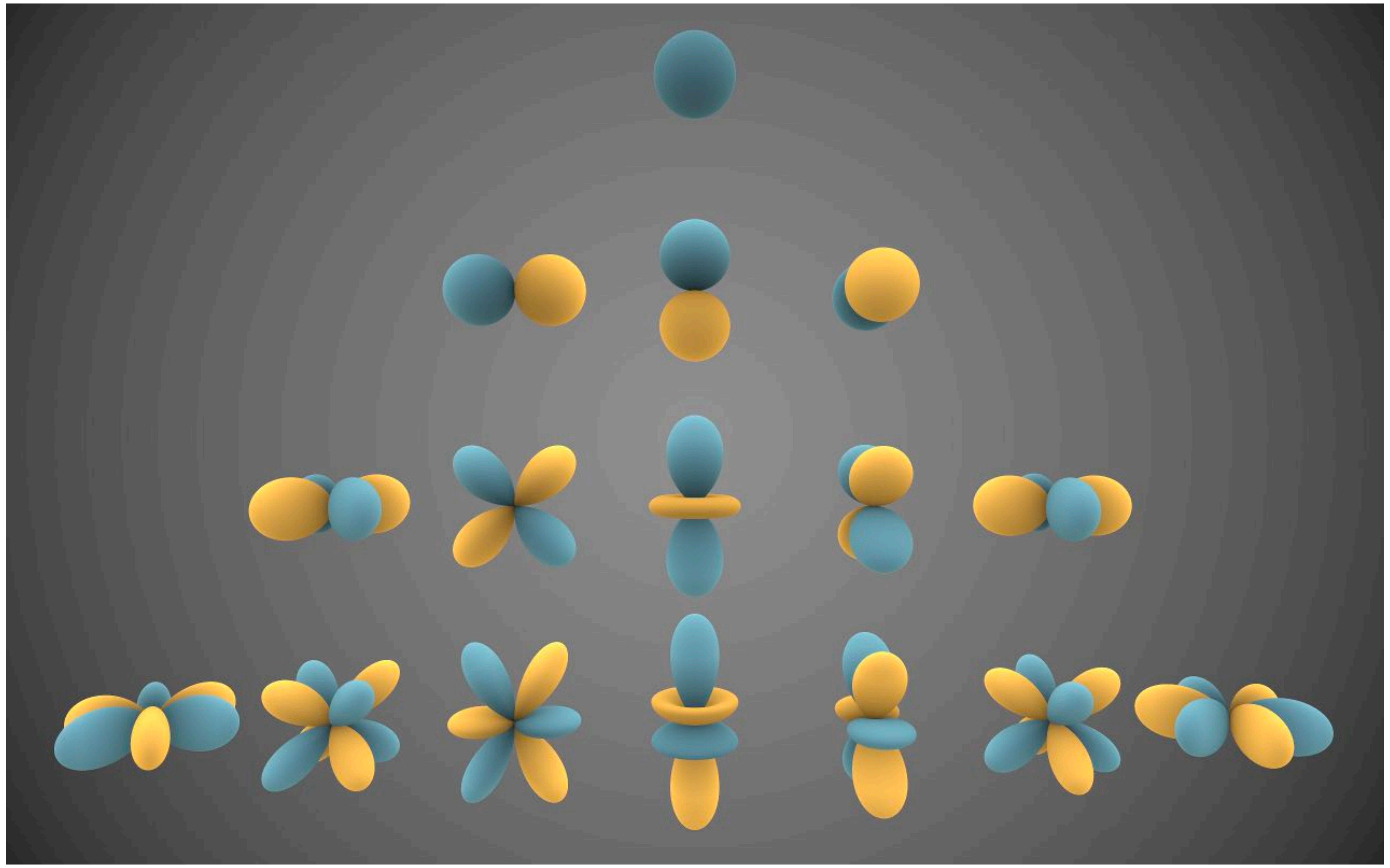
Octree: nodes have 8 children (partitions 3D space)





Background part 2: spherical harmonics

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



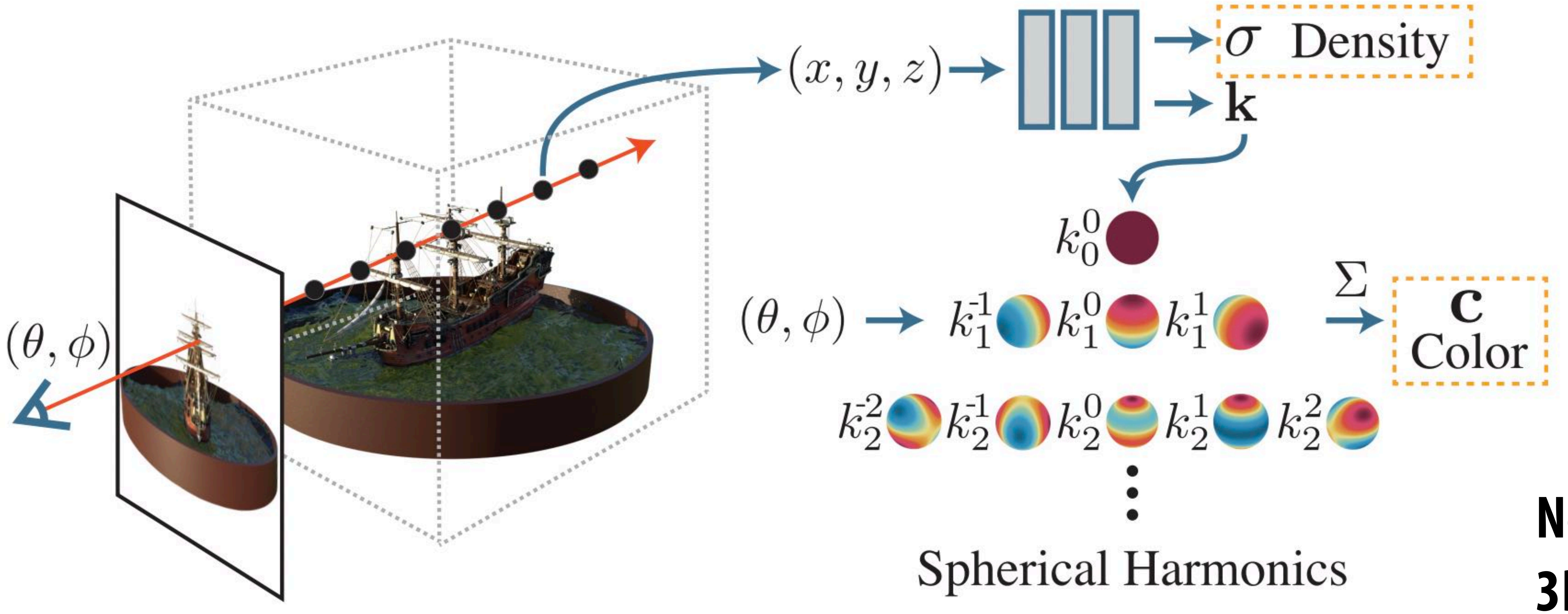
- Represent $c(\mathrm{p}, \omega)$ compactly by projecting into basis of SH.

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

Okay… so let's have F_θ just predict SH coefficients

(We can compute $c(\mathrm{p}, \omega)$ from the coefficients as needed)

PlenOctrees [ICCV 21]

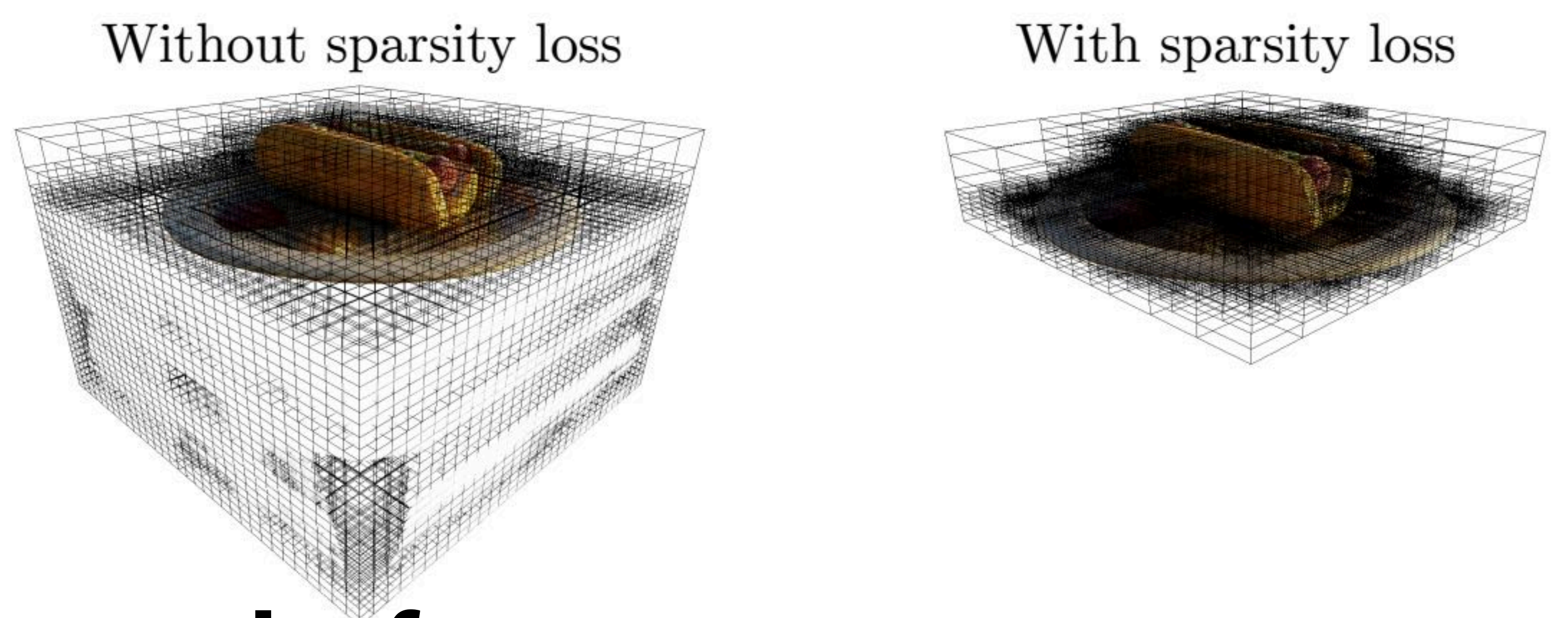


$$\boxed{F_\theta(\mathrm{p})} \rightarrow \begin{array}{l} \sigma(\mathrm{p}) \\ K(\mathrm{p}) \end{array}$$

(Vector of SH coefficients)

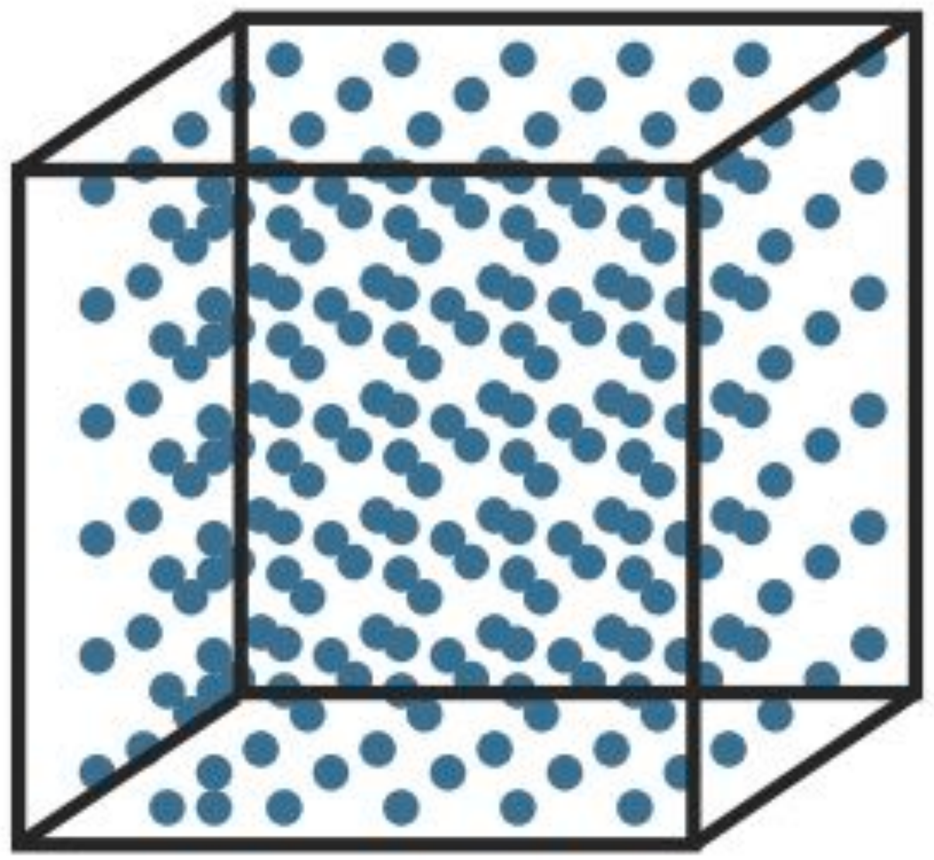
Note, now the MLP is just a function of 3D coordinates p.
(Not position and direction… the direction is handled by the SH representation)

Okay, let's just train for a bit…

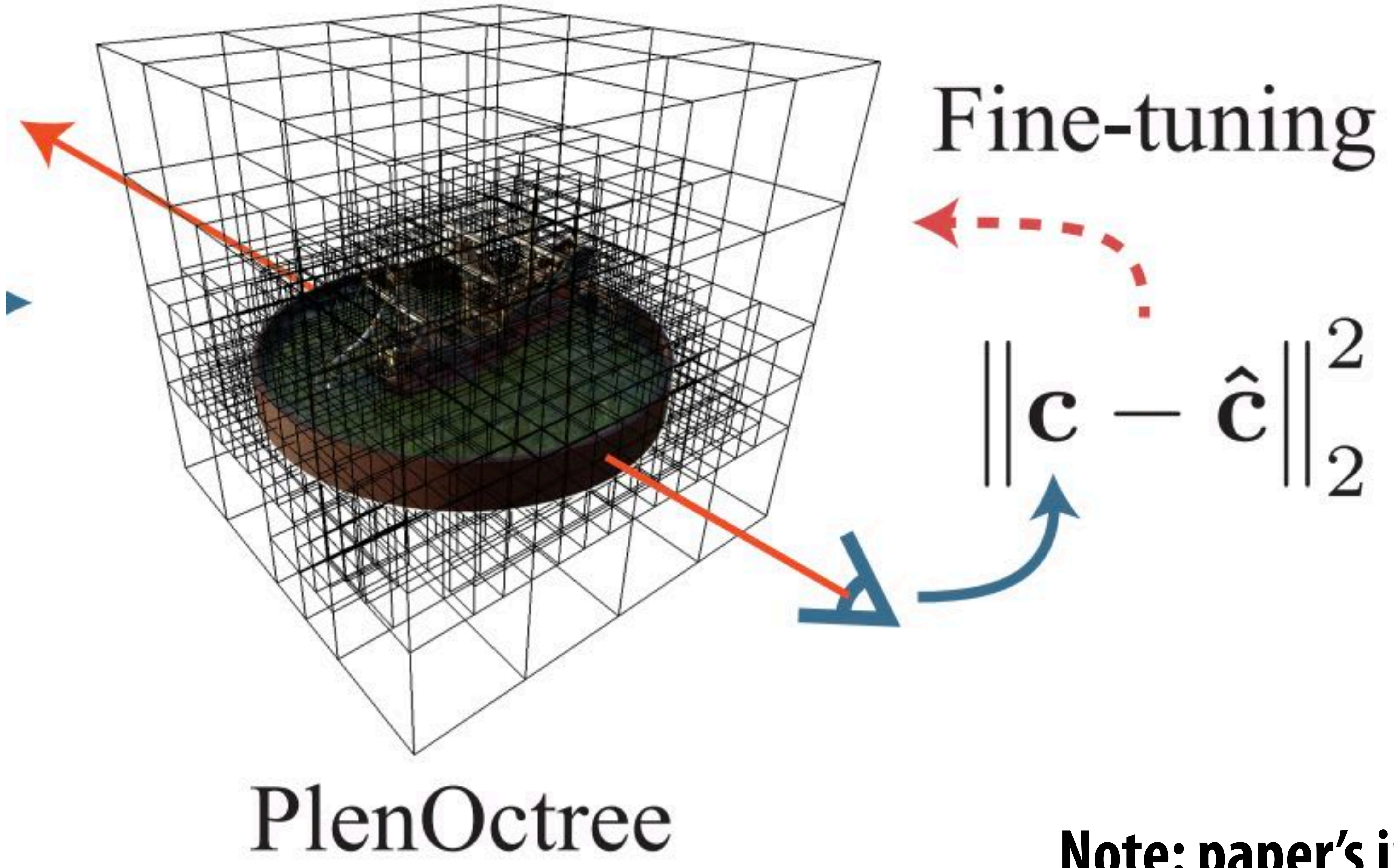


- Until the MLP tells us where the empty space clearly is.

- Then move to an octree representation that's more efficient to render from…
- With the octree structure *fixed*, we can continue to optimize SH coefficients and density at leaves with differential volume rendering and SGD



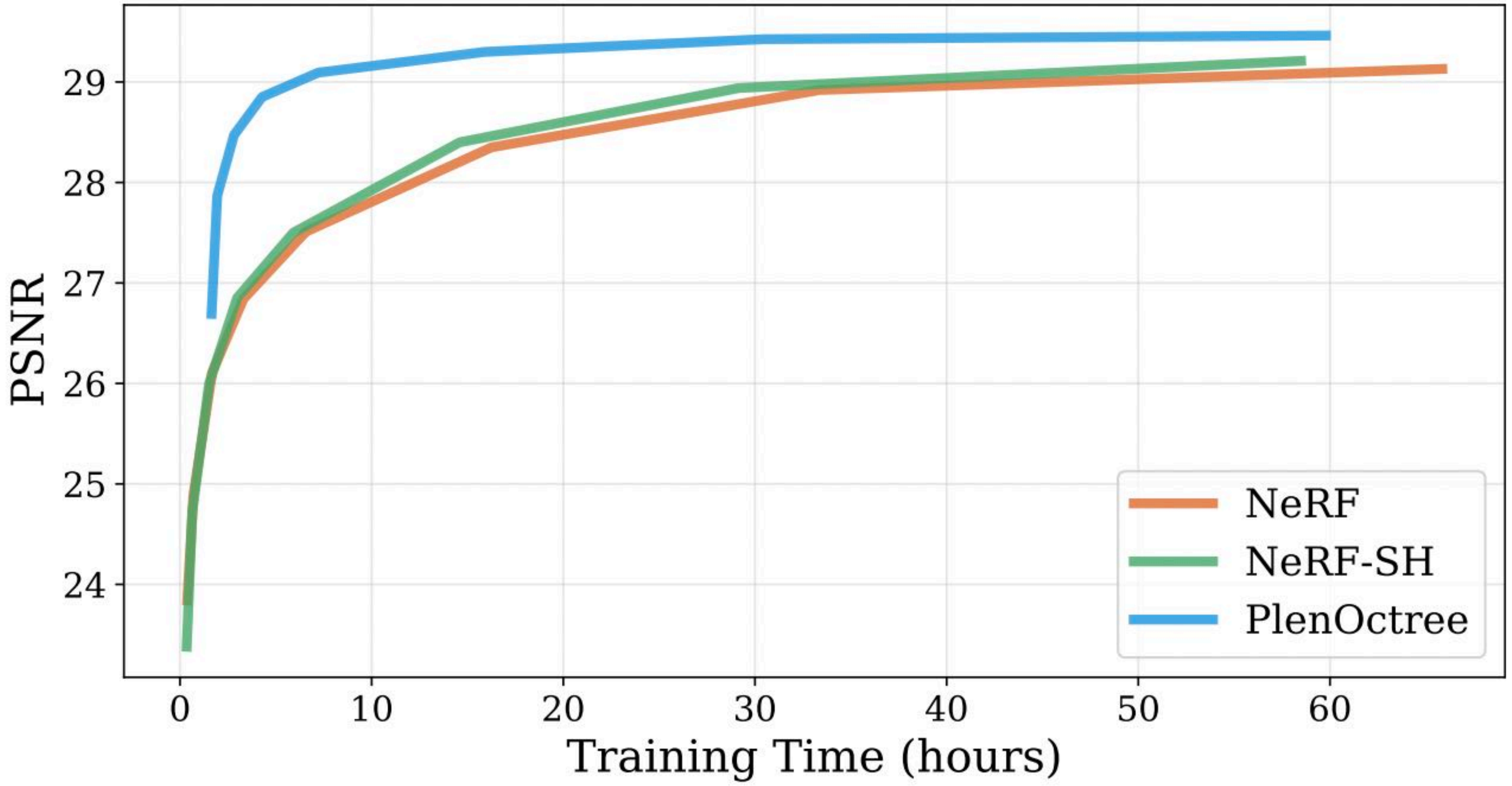
Use the initial MLP to densely sample volume
(Find the empty space that's used to build the octree)



Note: paper's implementation uses 2-level octree

What just happened?

- We performed initial training a la original NeRF
- Once we get a sense of where the empty space is, we switch to a traditional acceleration structure to replace the "big" MLP. This paper uses SH model at the leaves, but we could have instead used a little MLP at the leaves too.
- That structure speeds up rendering (a lot), and also speeds up "fine tuning" training, since the initial "big" MLP need not be trained to convergence



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

Yet another take (for SDFs not NeRF)

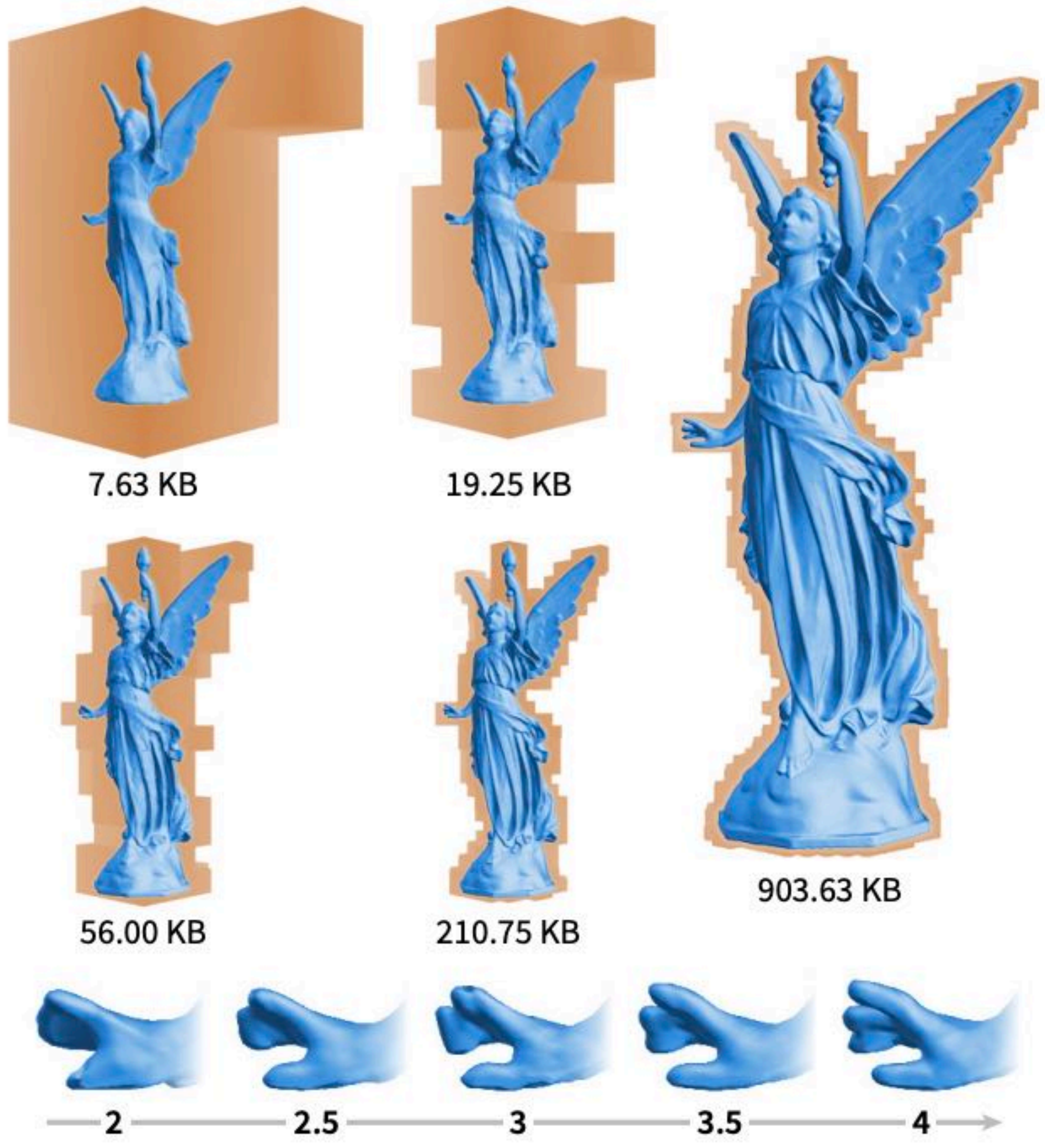
Neural Geometric Level of Detail
[CVPR 21]

- Sparse voxel octree, but store 32-D neural code stored at all empty cells, not just leaf
  - Data at level L is coarse representation of scene
  - Train one MLP per level

- Sampling the SDF at location p
  - For each level:
    - Get neural code z for level l at point p
    - Evaluate MLP_l(z) to get level l's signed distance
  - Blend distances for all levels

- MLP is small because heavyweight lifting comes from octree traversal and neural code.

- Details: clever tricks about how to sample points from the original surface representation during training

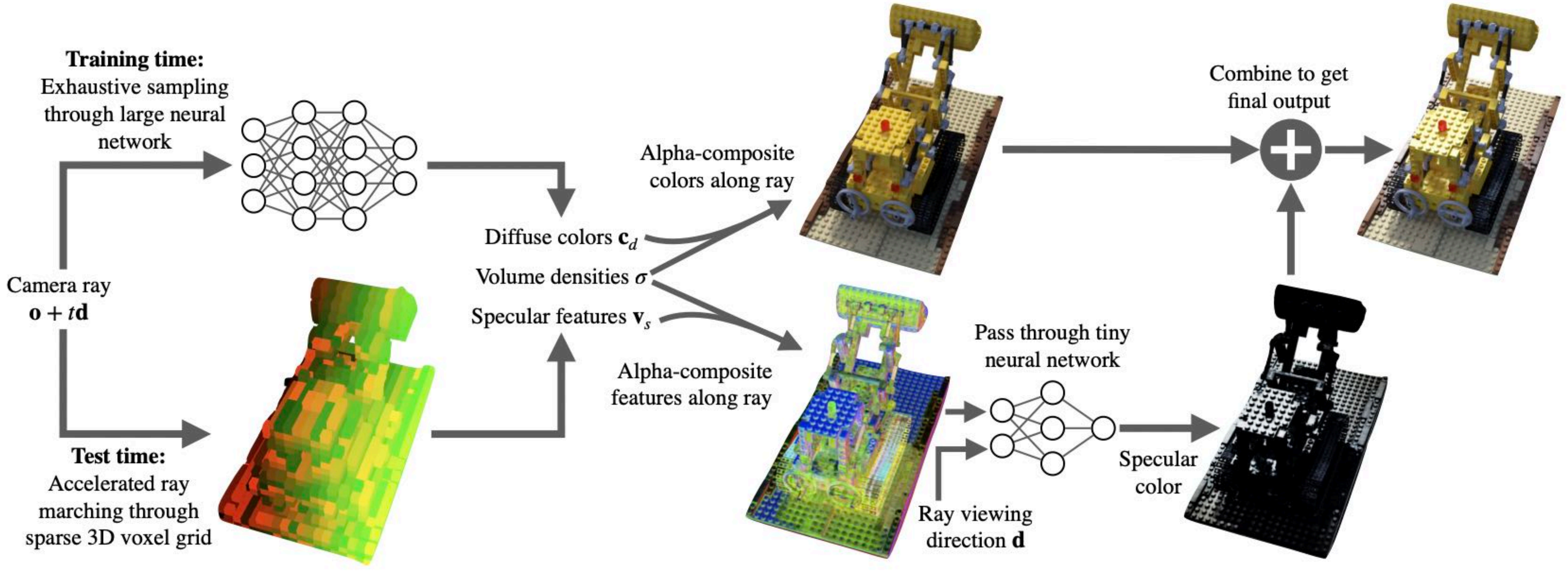


Another take (by other groups)

Baking Neural Radiance Fields for Real-Time View Synthesis [ICCV 21]

- Same idea: densely sample MLP to “bake” density into sparse octree representation
- Instead of SH, MLP outputs density, diffuse color, and specular (directional) “features”

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



- “Volume render” both the diffuse colors and the features, and use one MLP eval at the end to turn the diffuse color and features into a final color
  - Note: now 1 MLP eval per ray, instead of per step

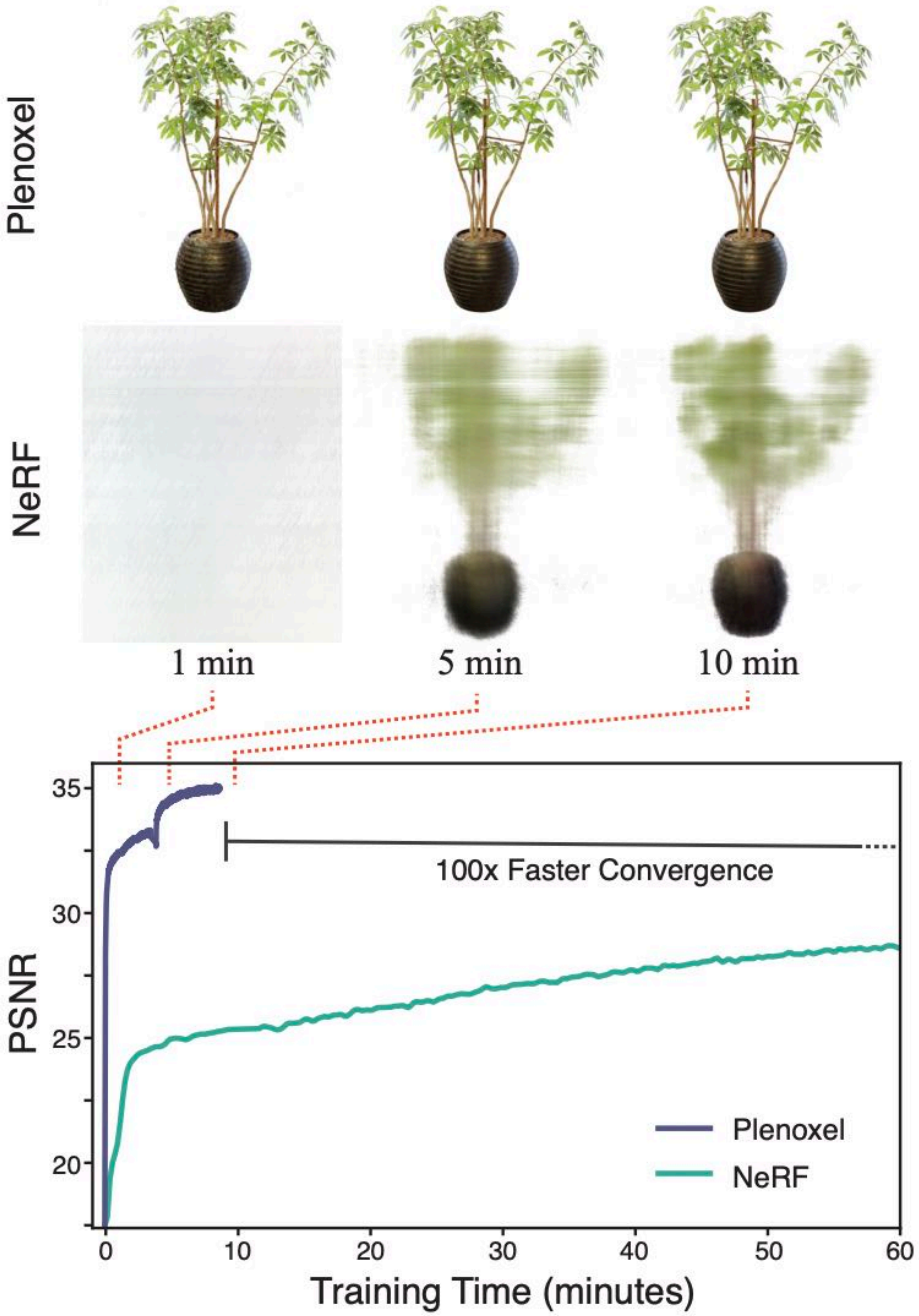
In the authors’s words: “Since we do not know where the scene geometry lies during optimization, it is crucial to use a compact representation that can represent highly-detailed geometry at arbitrary locations. However, after a NeRF has been trained, we argue that it is prudent to rethink this space-time tradeoff and “bake” the NeRF representation into a data structure that stores pre-computed values from the MLP to enable real-time rendering.”

Finally…back to where we began

Plenoxels [CVPR 22]

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

- Takeaway: conventional computer graphics representations are *much* more efficient representations to learn on
  - But learning can be a little challenging… see all the details in the paper about choice of optimizer, etc. etc.



Light probe locations in a game

Here: SH probes sampled on a uniform grid

The neural representation "template"

- **Train MLP to understand 3D occupancy (where the surface is)**
  - **Uses Little-to-no geometric priors (so need position encoding tricks, etc)**

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

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

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

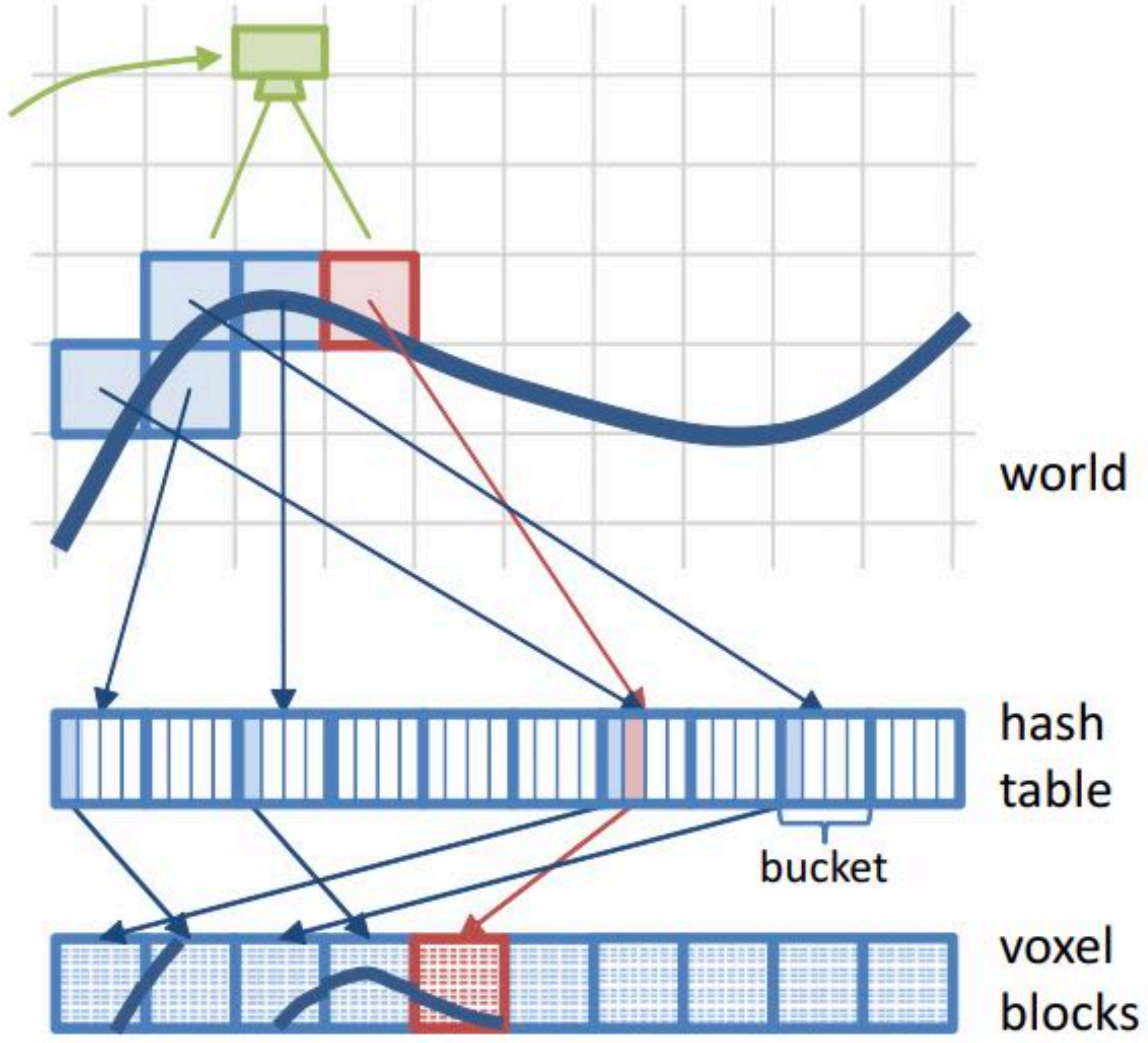
- **Continue optimization on the fixed, sparse representation**
  - **Leverages differential volume rendering on sparse structure**
  - **What we're now learning is how to represent/compress the local details**

More background: sparse voxel hashing

Real-time 3D Reconstruction at Scale using Voxel Hashing [TOG 13]

- Voxel hashing as a fast GPU data structure for 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

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



NVIDIA's instant neural graphics primitives (NGP)

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

Given position P:

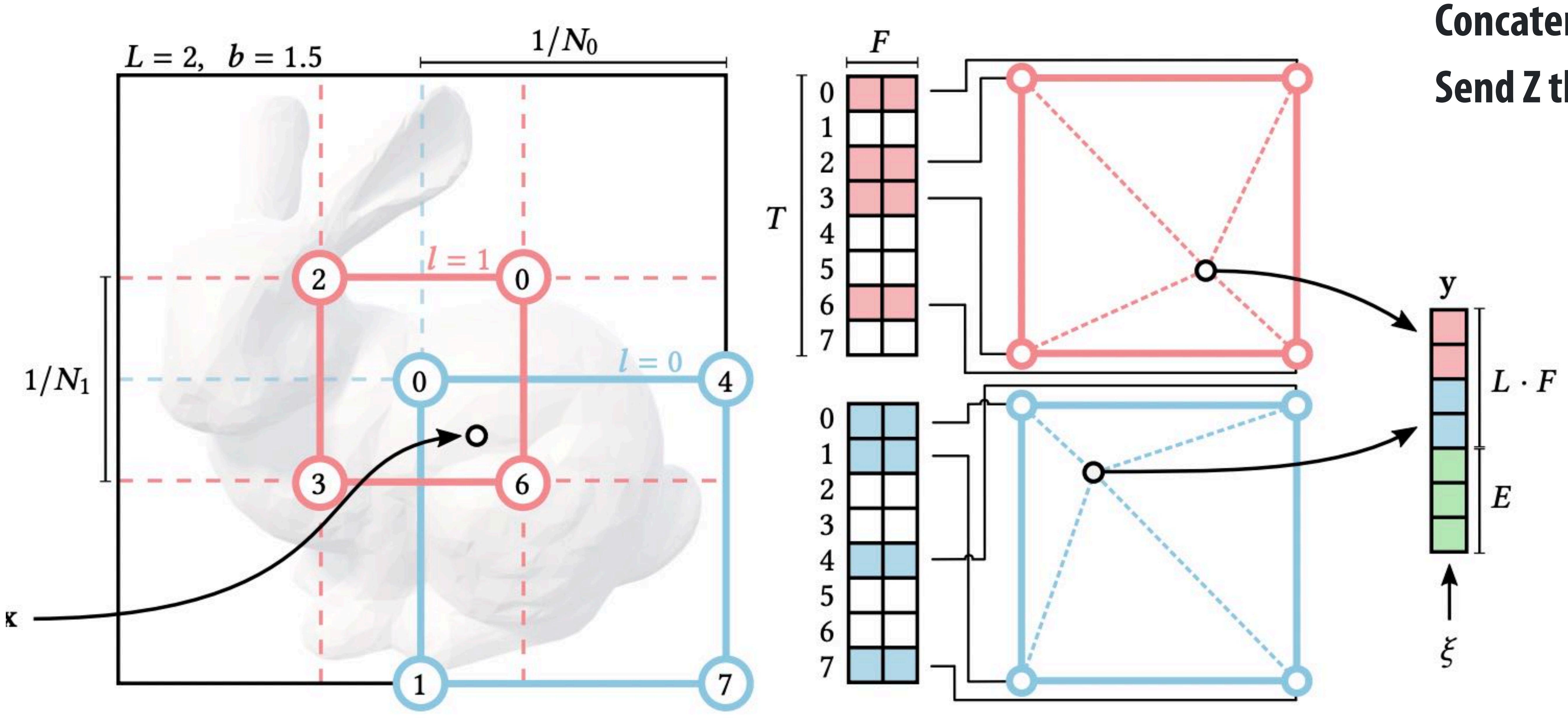
Compute indices of cell containing P on a bunch of different resolution grids (L grids)

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.

More demos

Discussion: what have we learned?

- Definitely amazing: “unreasonable effectiveness of optimization”
  - Credit: Ren Ng for this perspective

- There’s a huge art to getting optimization to work
  - I seriously doubt I could get these things to optimize
  - If I was a easy career Ph.D. student, I’d want to become very accomplished in the “art” of getting an optimizer to work for me

- Neural data compression just “makes sense” and is always a good thing (fundamental)

- A lot of the “key algorithmic ideas” from the NeRF explosion seem less fundamental to modern practice
  - Frequency encoding of position → neural code stored in data structures
  - Yes, removing structural bias is cool… but (that’s not specific to the MLP, and many of the recent works put strong priors back in to reduce cost or add robustness)