Lecture 12:

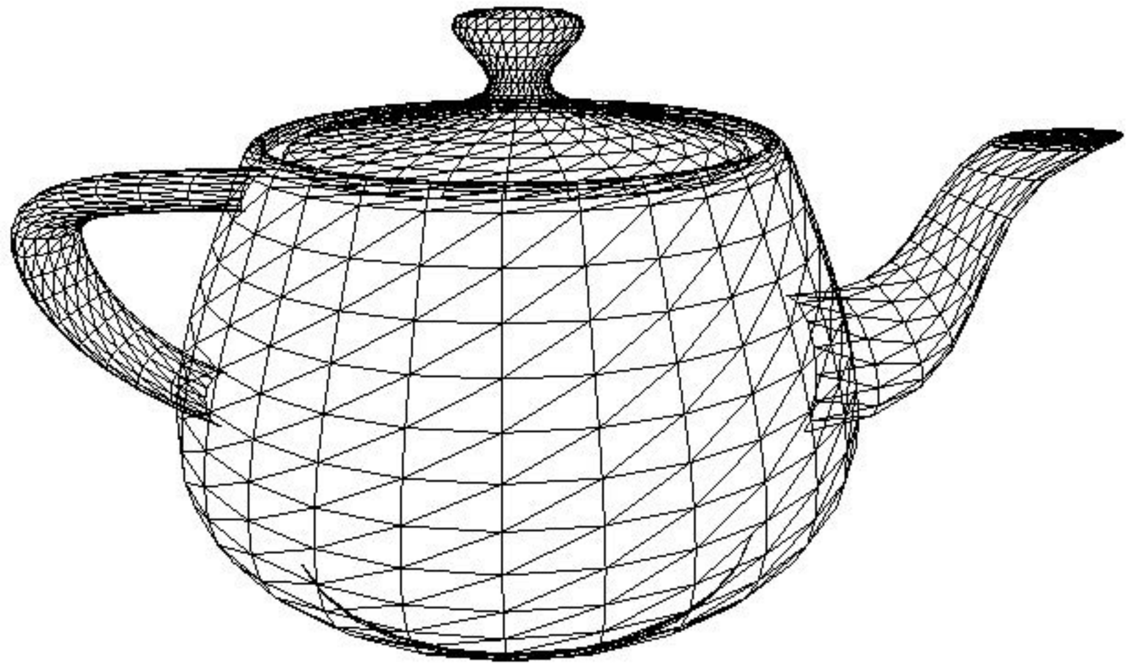
Background: the light field and rendering basics

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)

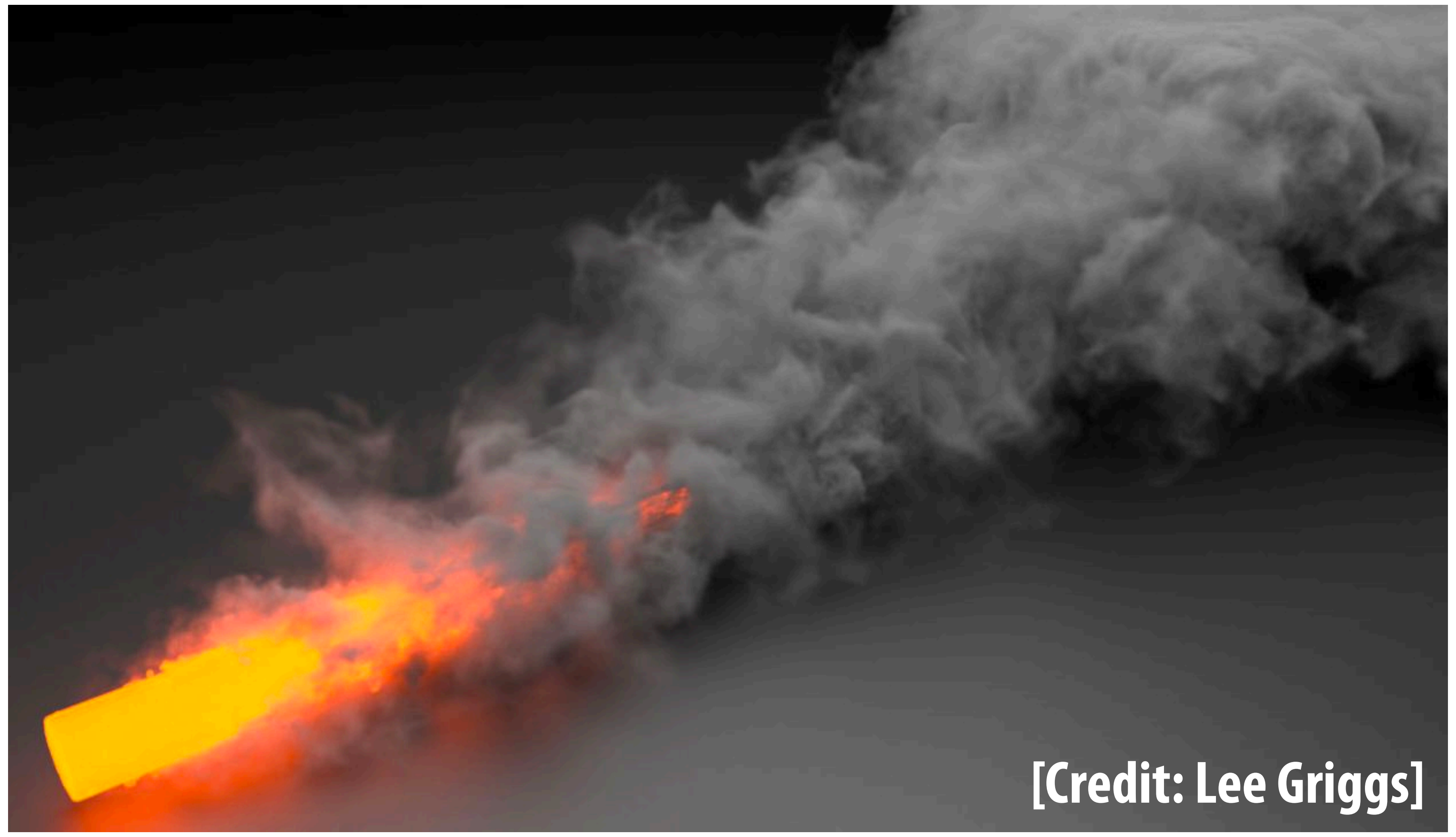


Depth-image based surface representations

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

[Andersen 16] Jump: VR video

3D Volumes



[Credit: Lee Griggs]

And many more… e.g., Implicit Surfaces

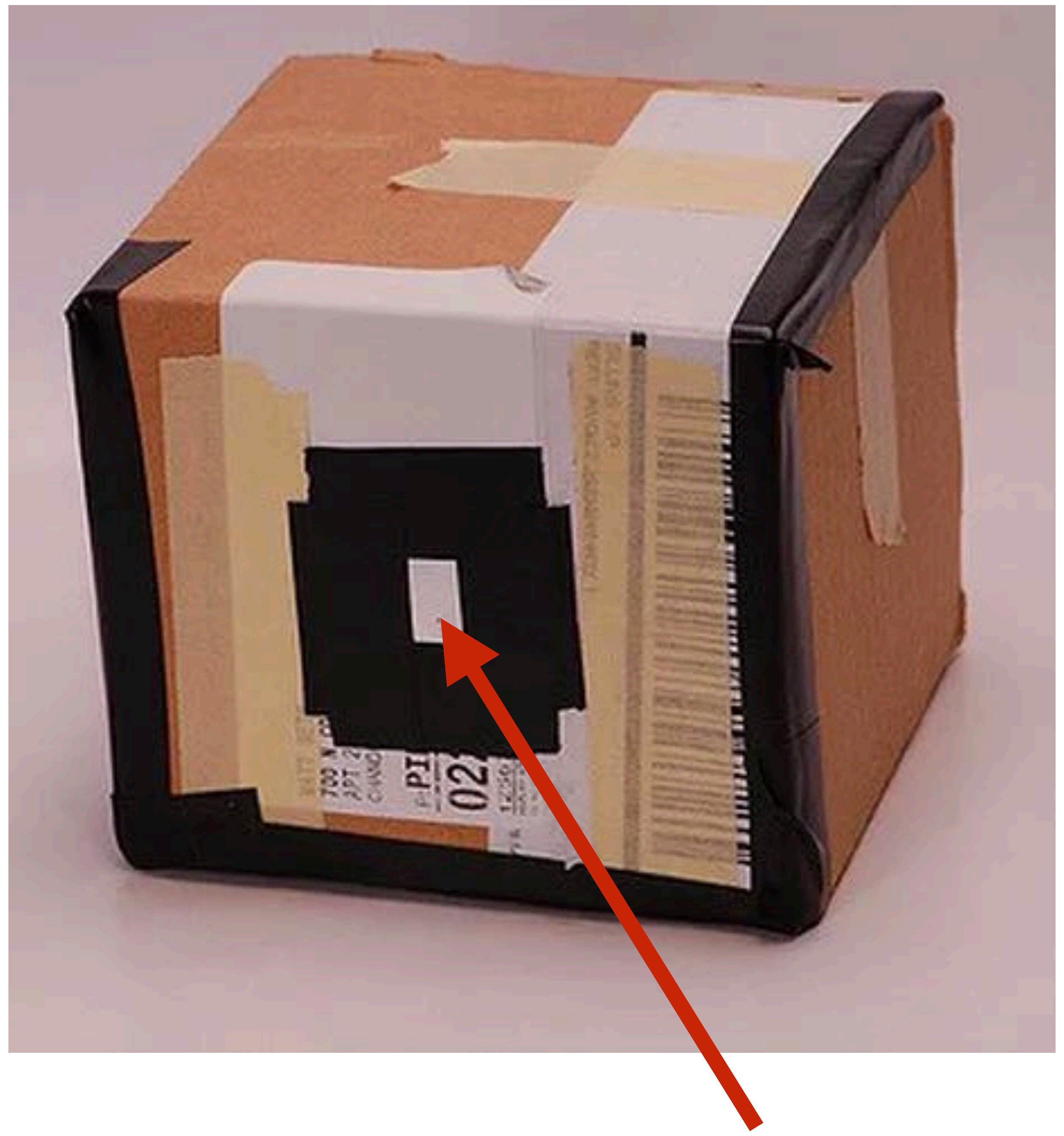
Novel view synthesis problem

Input photos (from a fixed set of views)

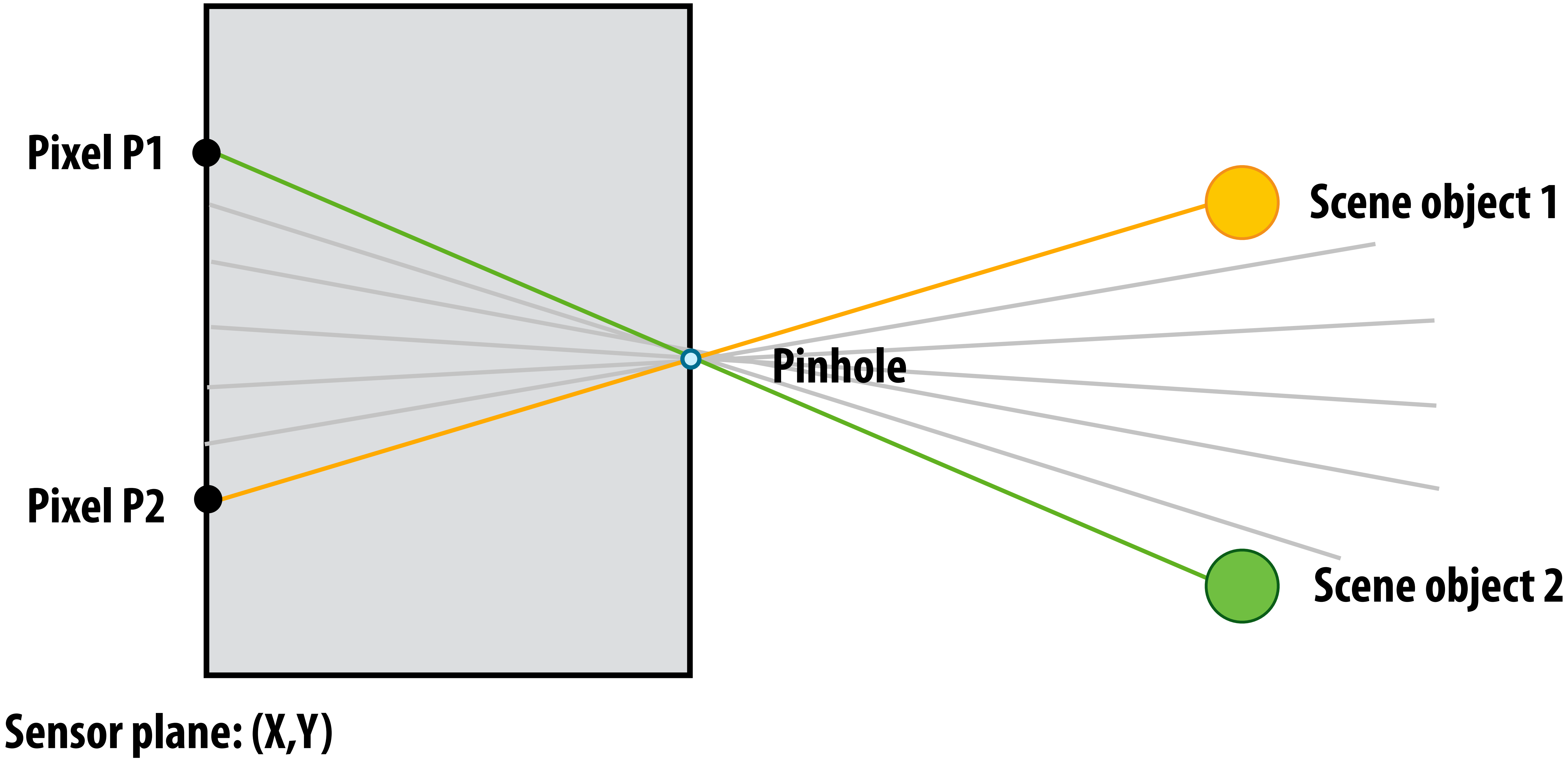
Novel views
(camera position different from those in input photos)

Fundamentals: the light field

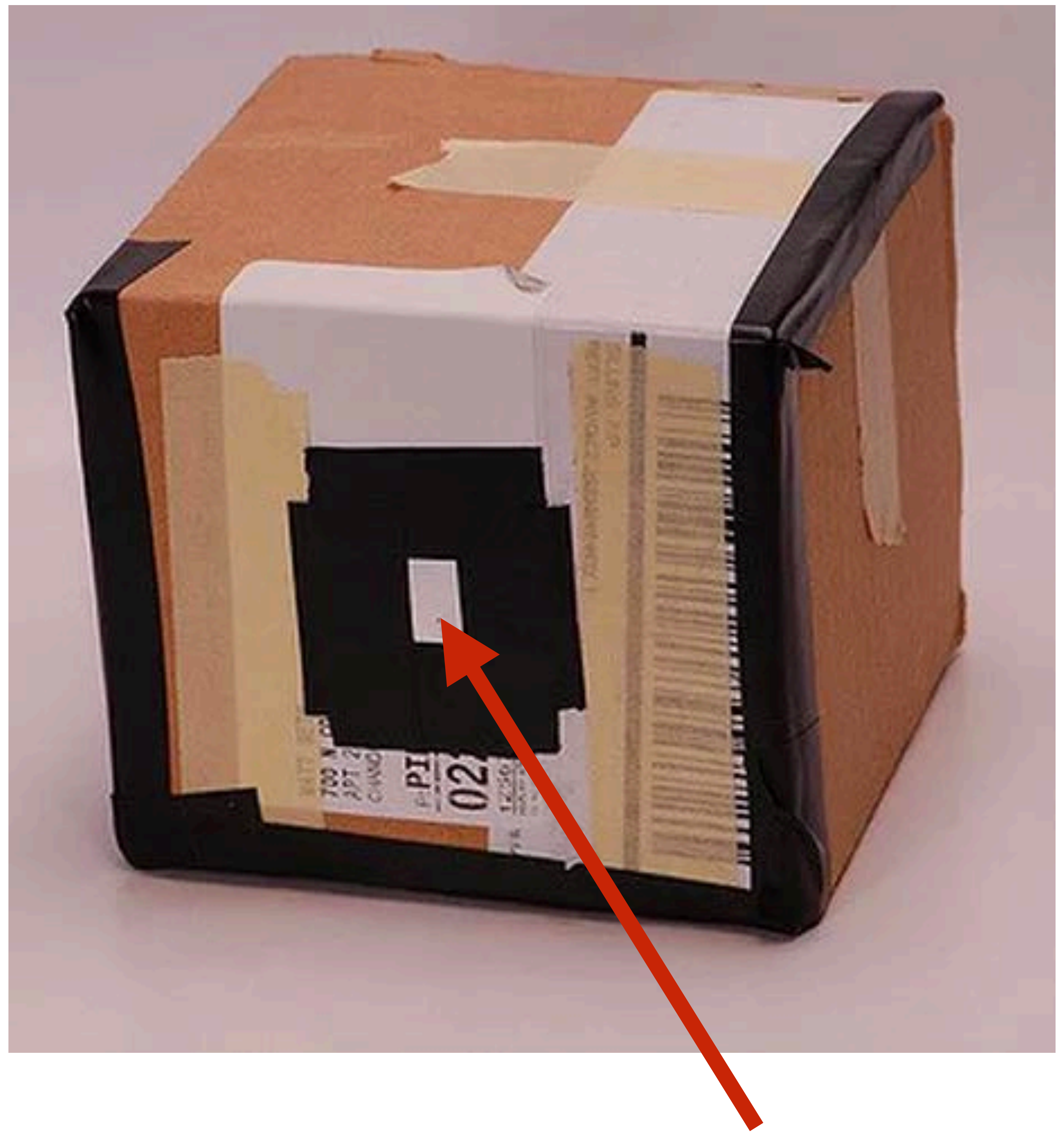
Recall basic pinhole camera



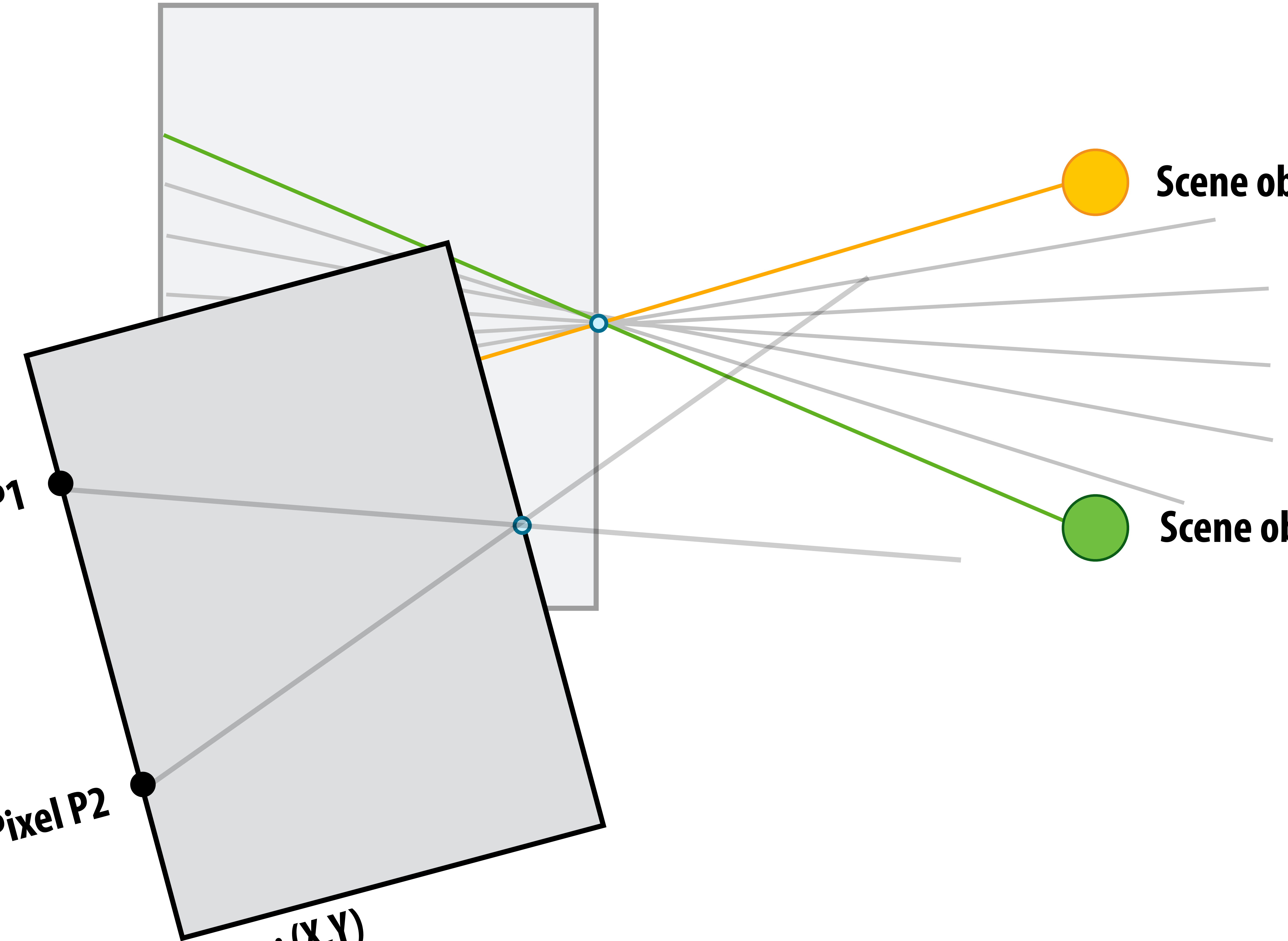
Pinhole



What about taking the pictures from a new viewpoint?



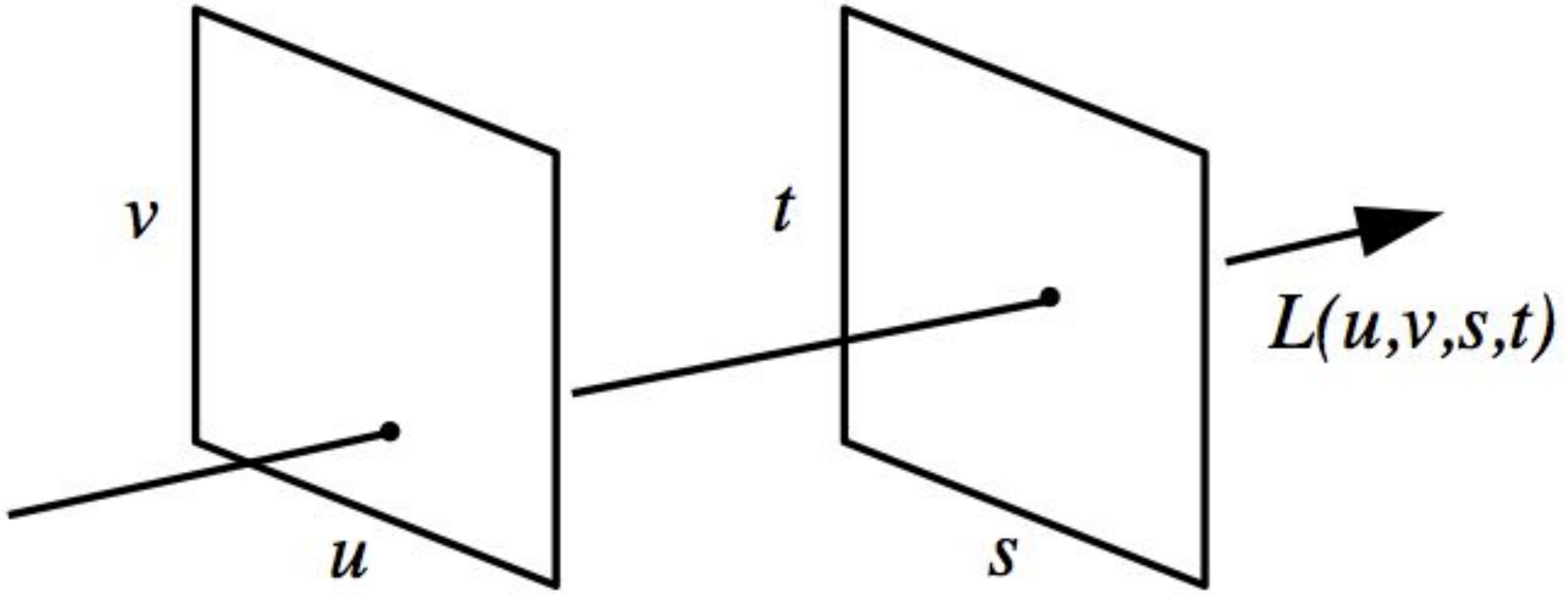
Pinhole



Light-field parameterization

[Levoy and Hanrahan 96]
[Gortler et al., 96]

Light field as a 4D function (represents light in free space: no occlusion)



[Image credit: Levoy and Hanrahan 96]

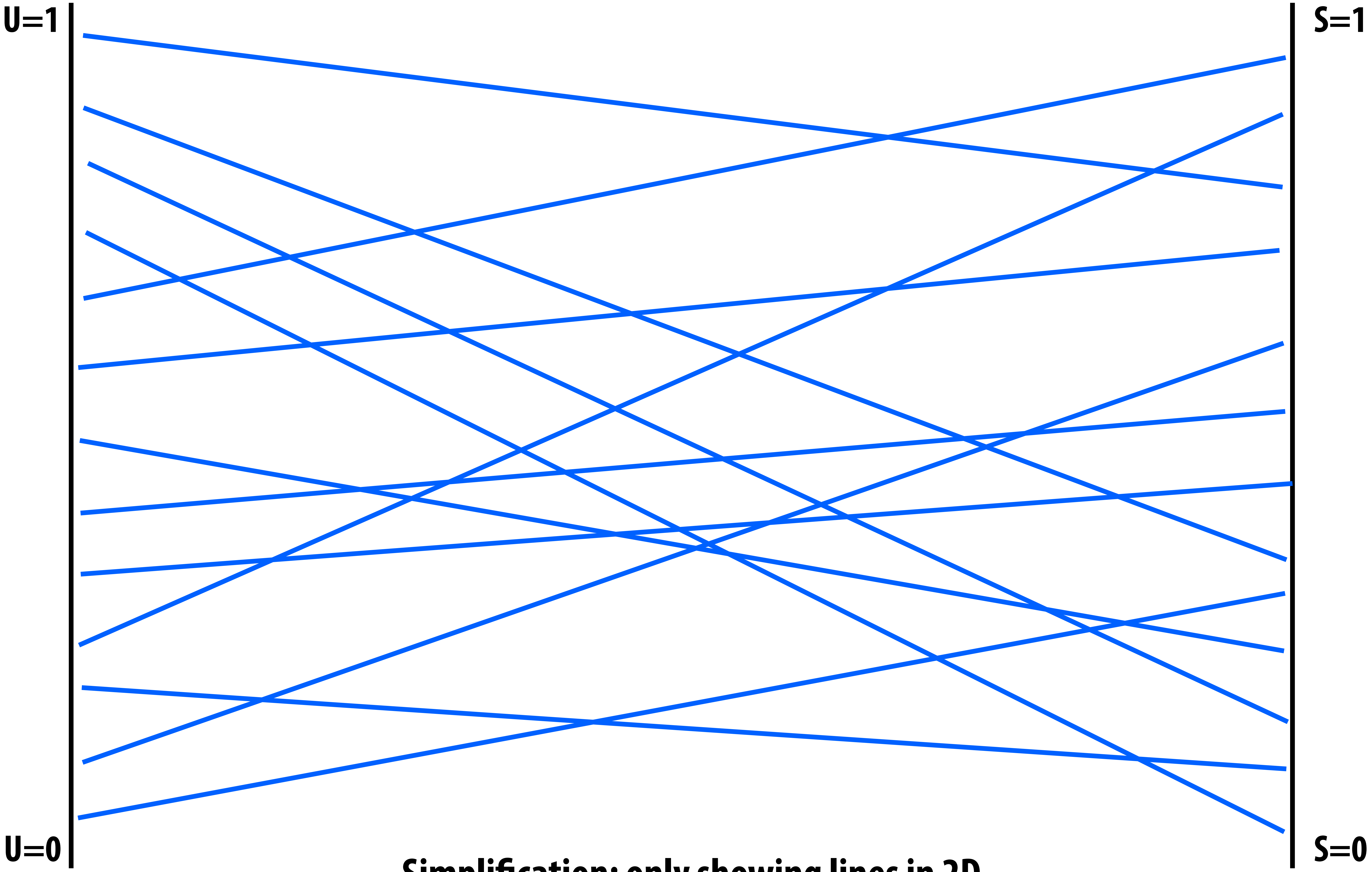
Efficient two-plane parameterization

Line described by connecting point on (u,v) plane with point on (s,t) plane

If one of the planes placed at infinity: point + direction representation

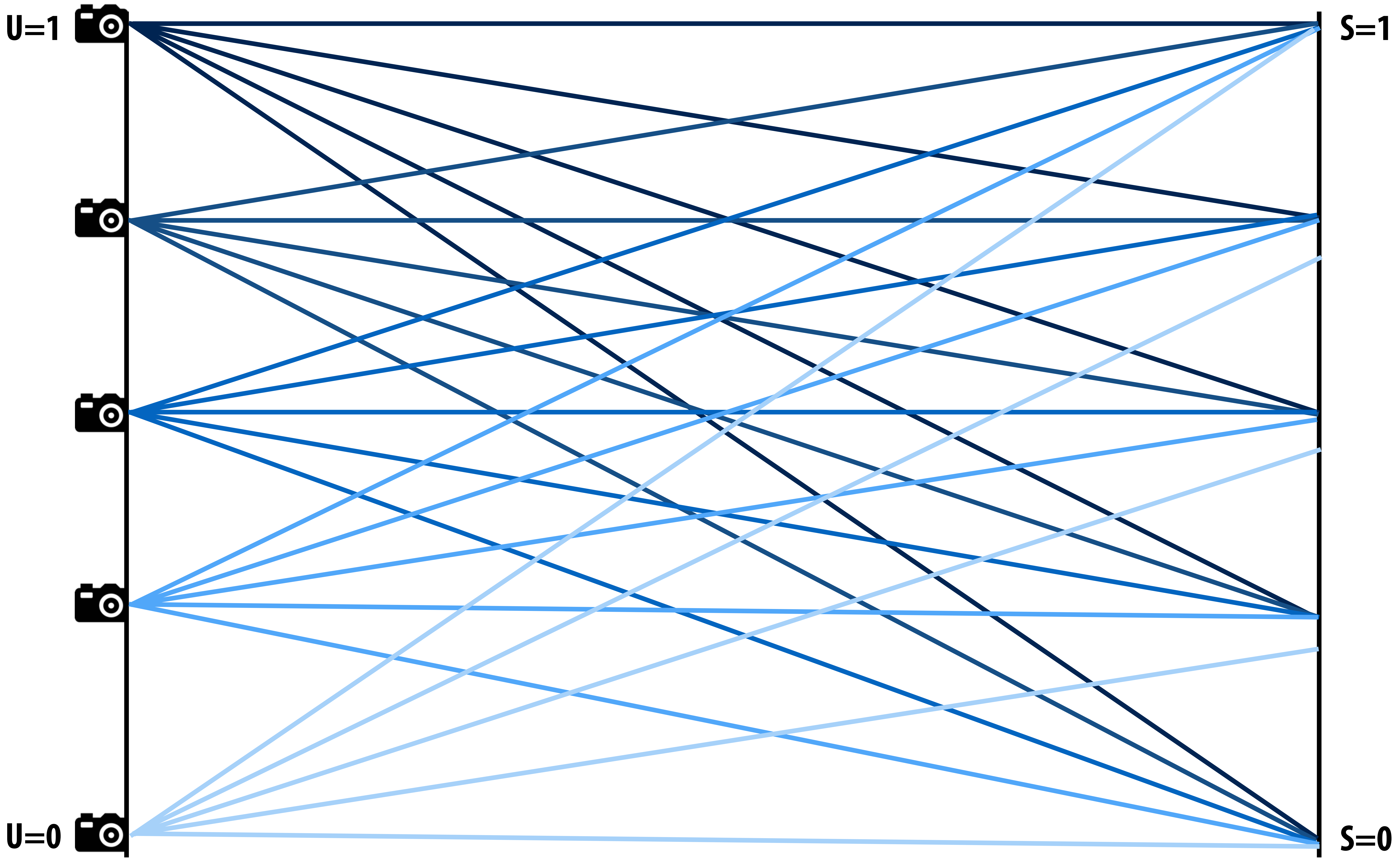
Levoy/Hanrahan refer to representation as a "light slab": beam of light entering one quadrilateral and exiting another

Sampling the light field



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

Measuring the light field by taking many pictures



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

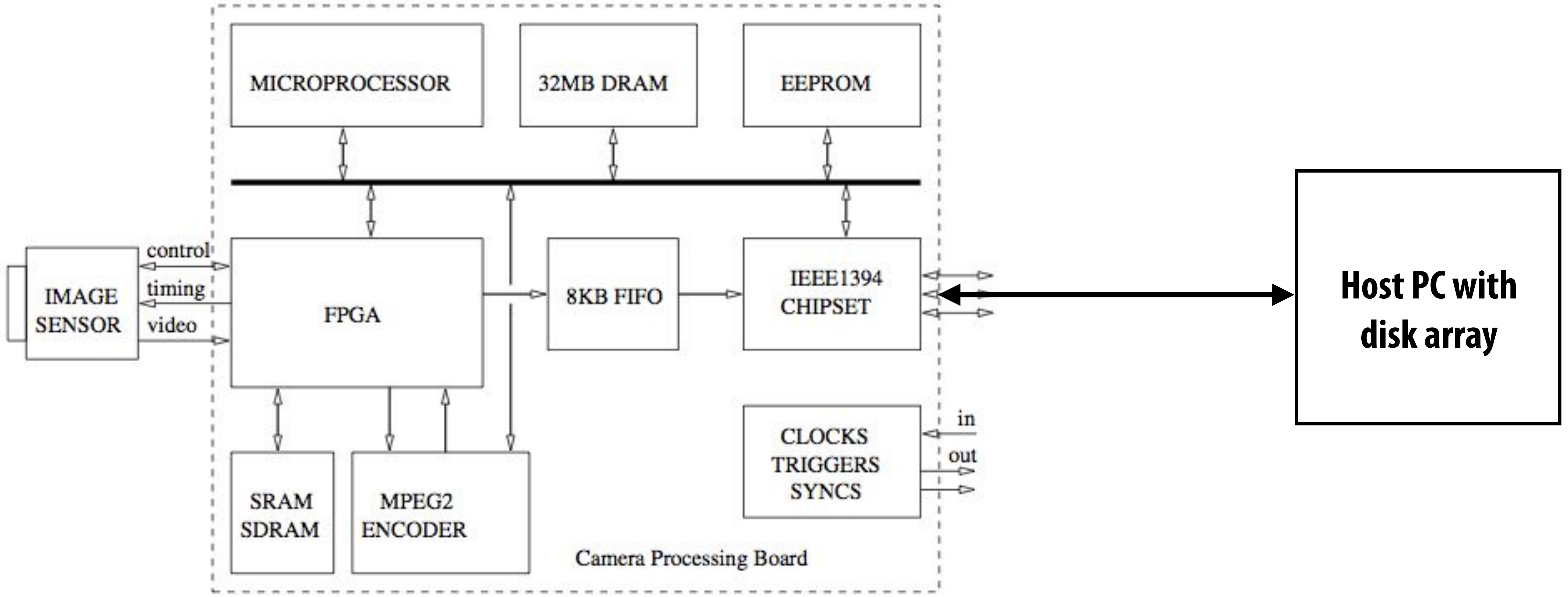
Stanford Camera Array

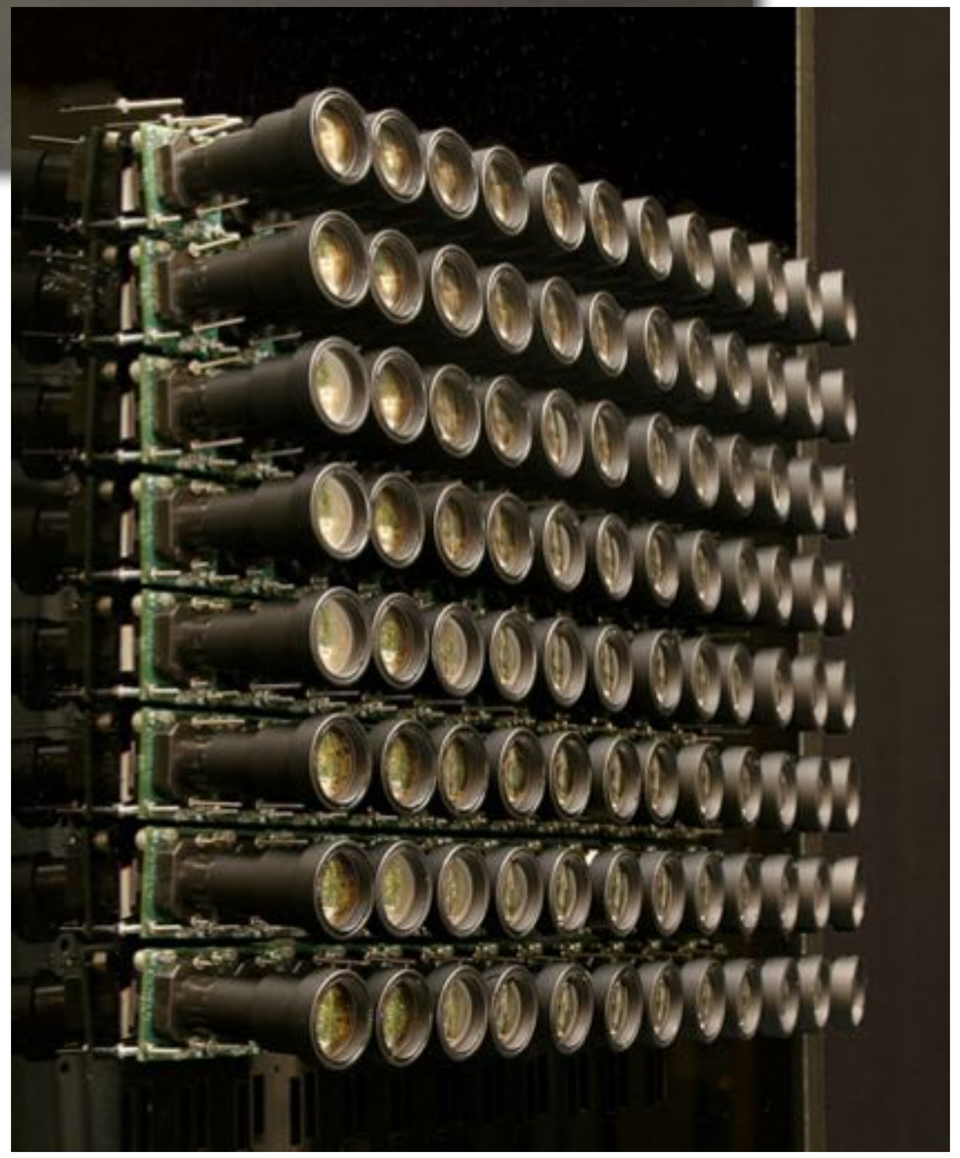
[Wilburn et al. 2005]

640 x 480 tightly synchronized, repositionable cameras

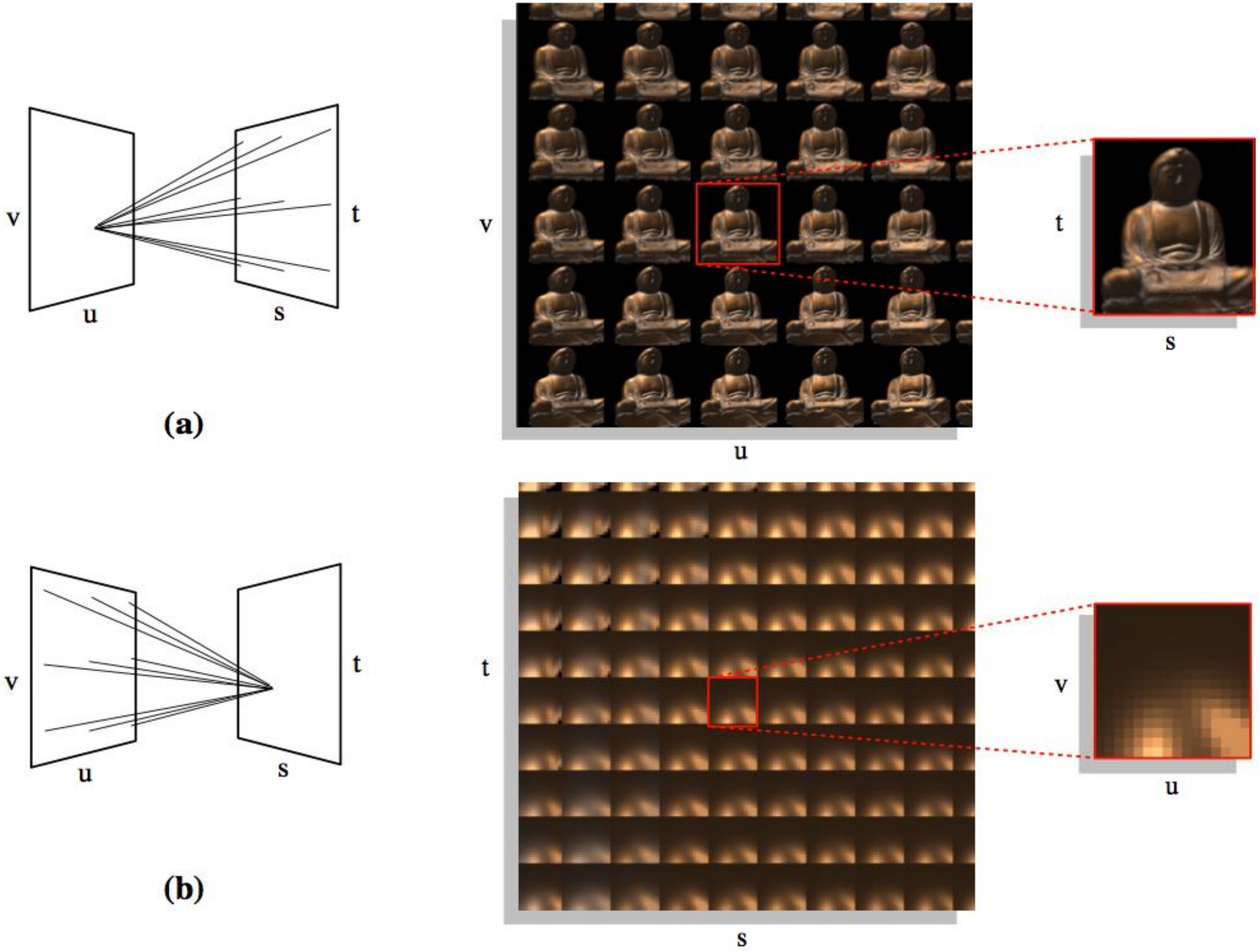
Custom processing board per camera

Tethered to PCs for additional processing/storage





Light field storage layouts



[Image credit: Levoy and Hanrahan 96]

Later light field cameras

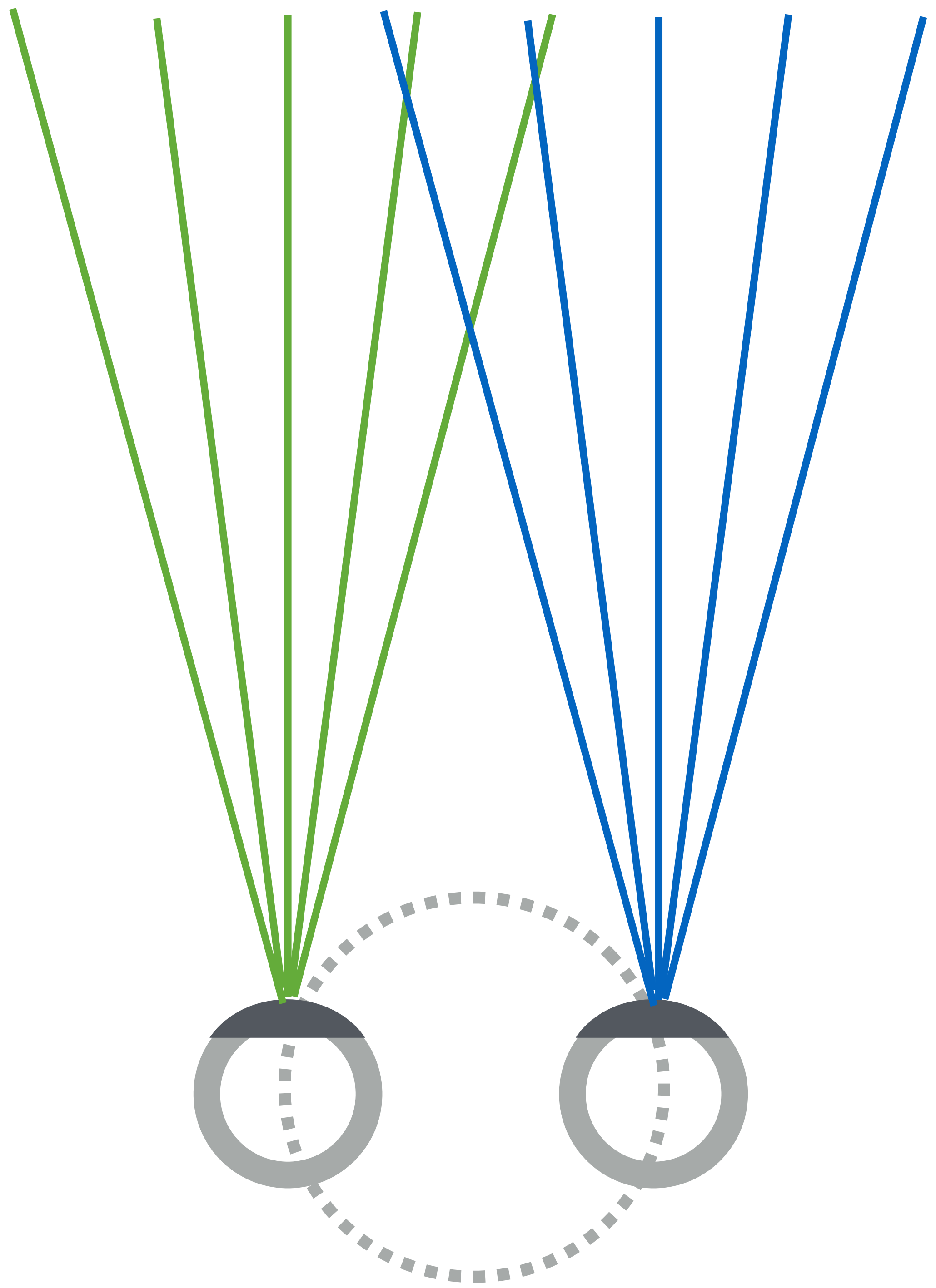
Lytro Illum

Acquiring light field content for VR

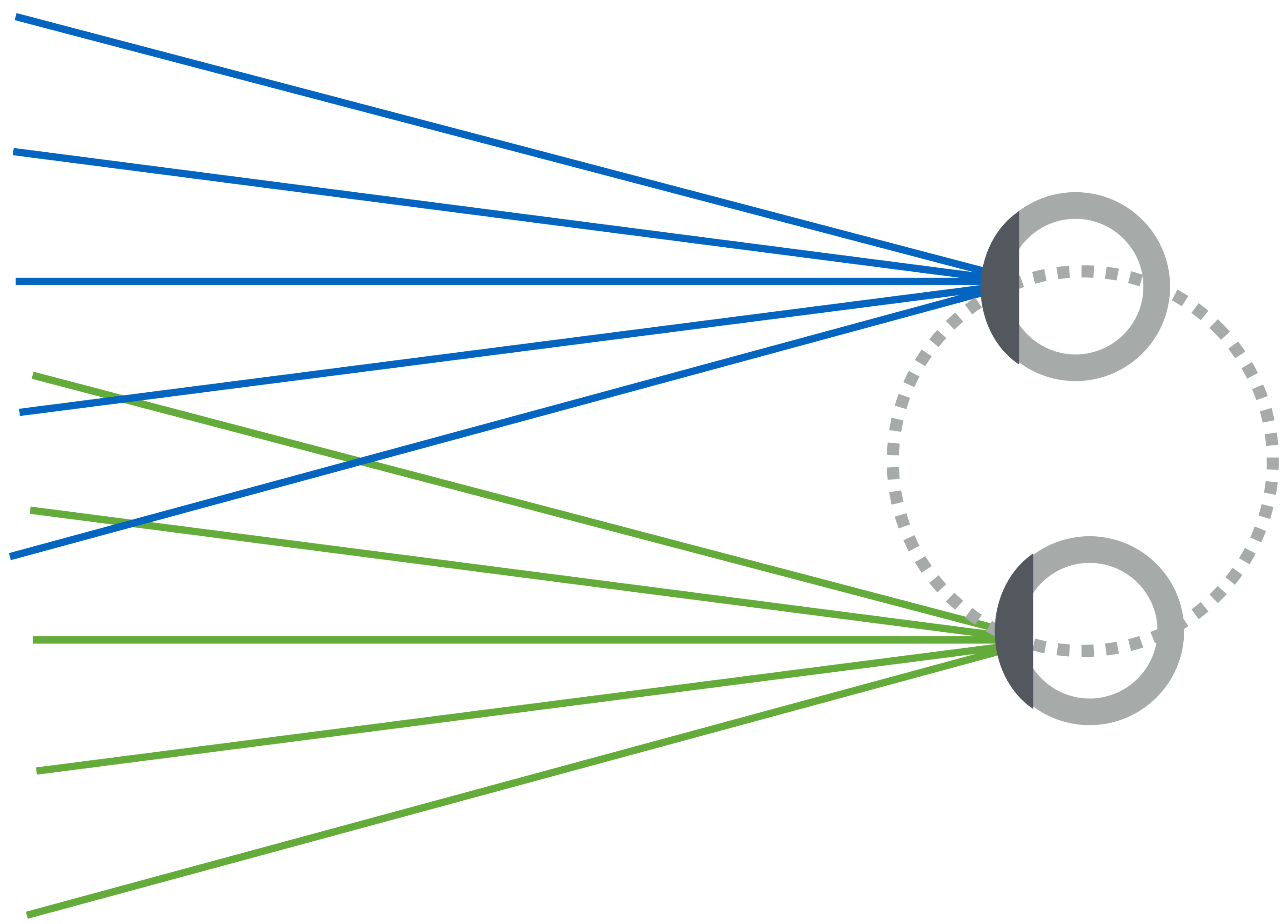
Google's Jump VR video:
Yi Halo Camera (17 cameras)

Facebook Manifold
(16 8K cameras)

Stereo, 360-degree viewing



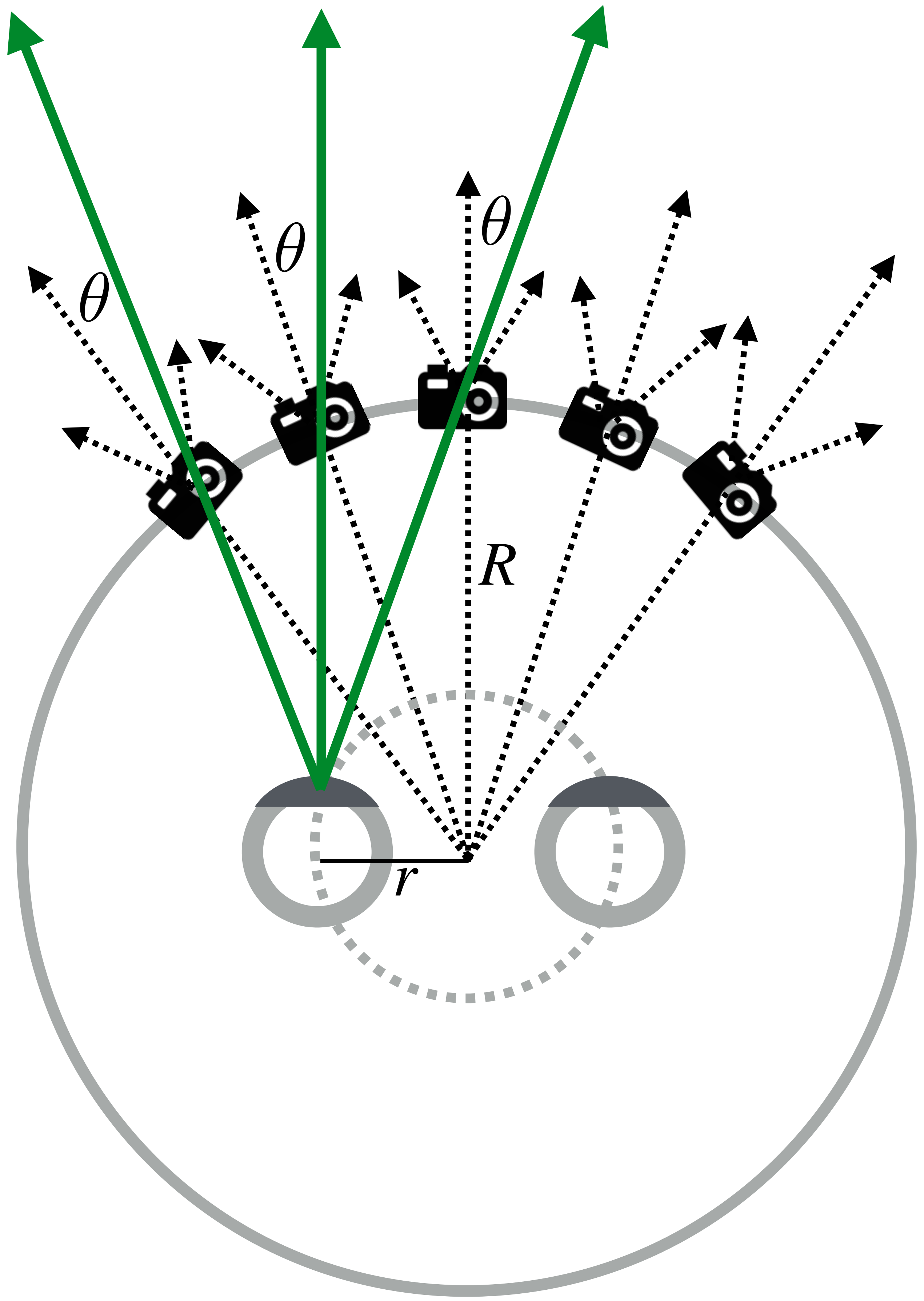
Stereo, 360-degree viewing



Measuring light arriving at left eye

Left eye

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

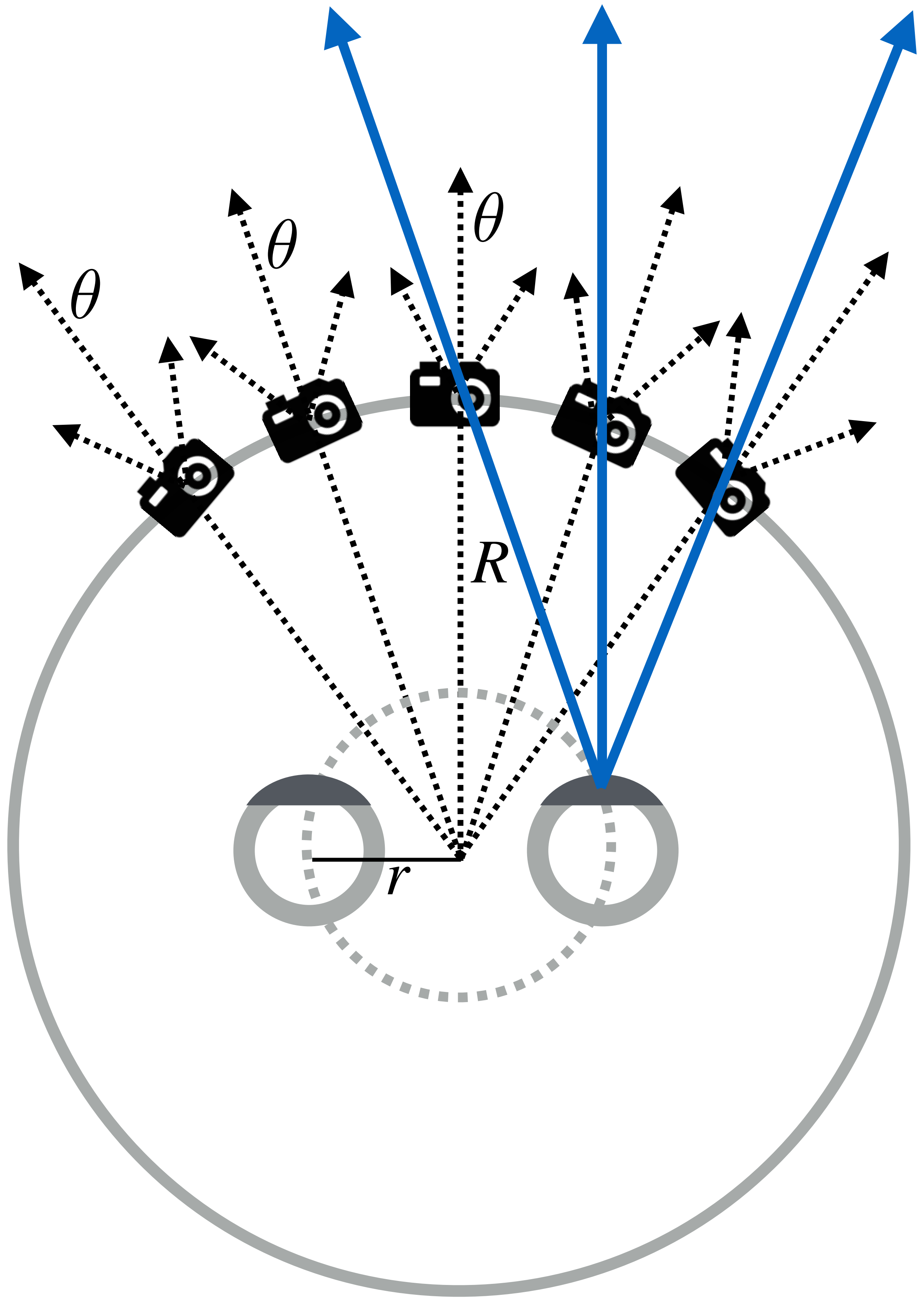


[Credit: Camera icon by Venkatesh Aiyulu from The Noun Project]

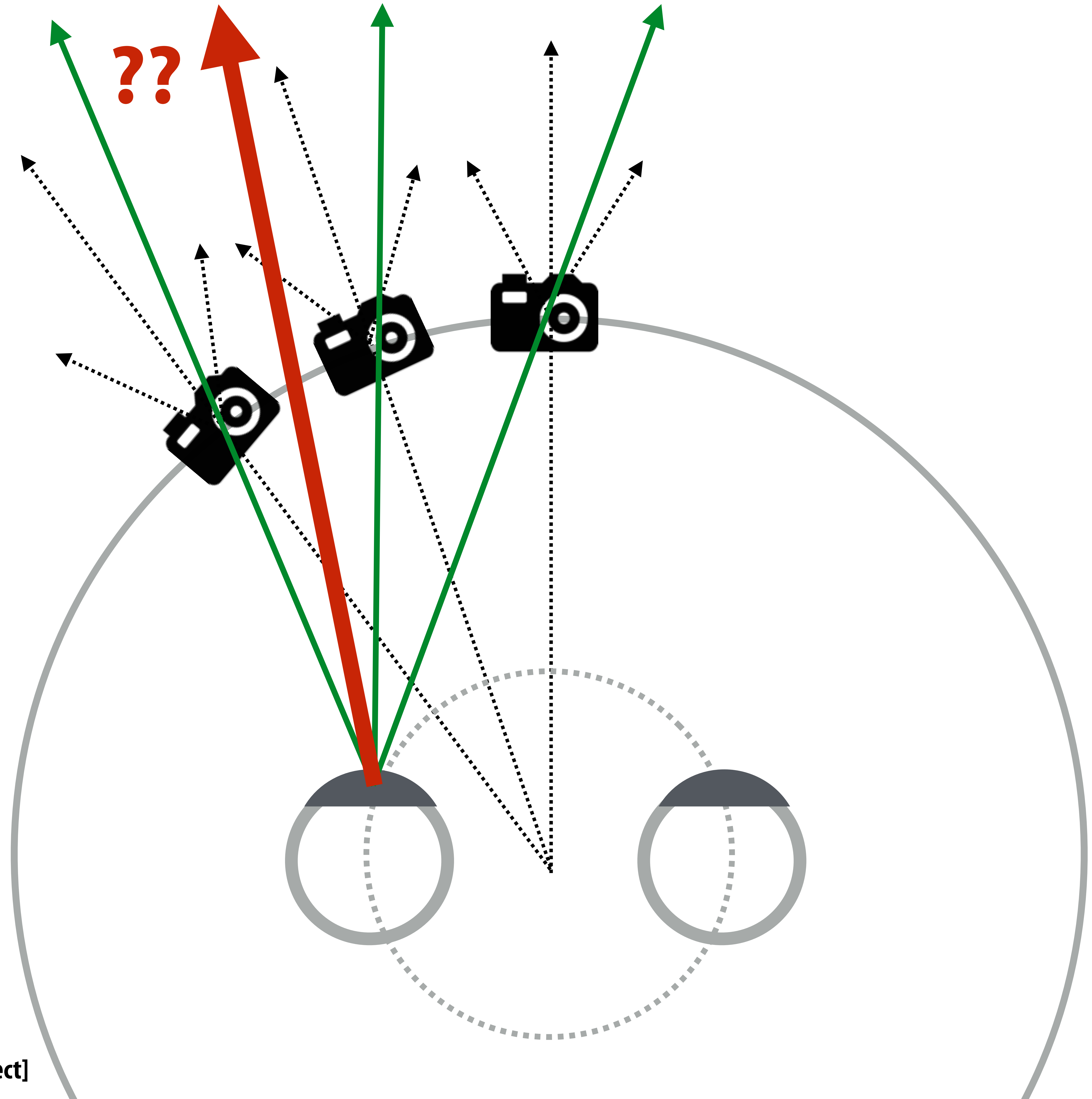
Measuring light arriving at right eye

Right eye

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

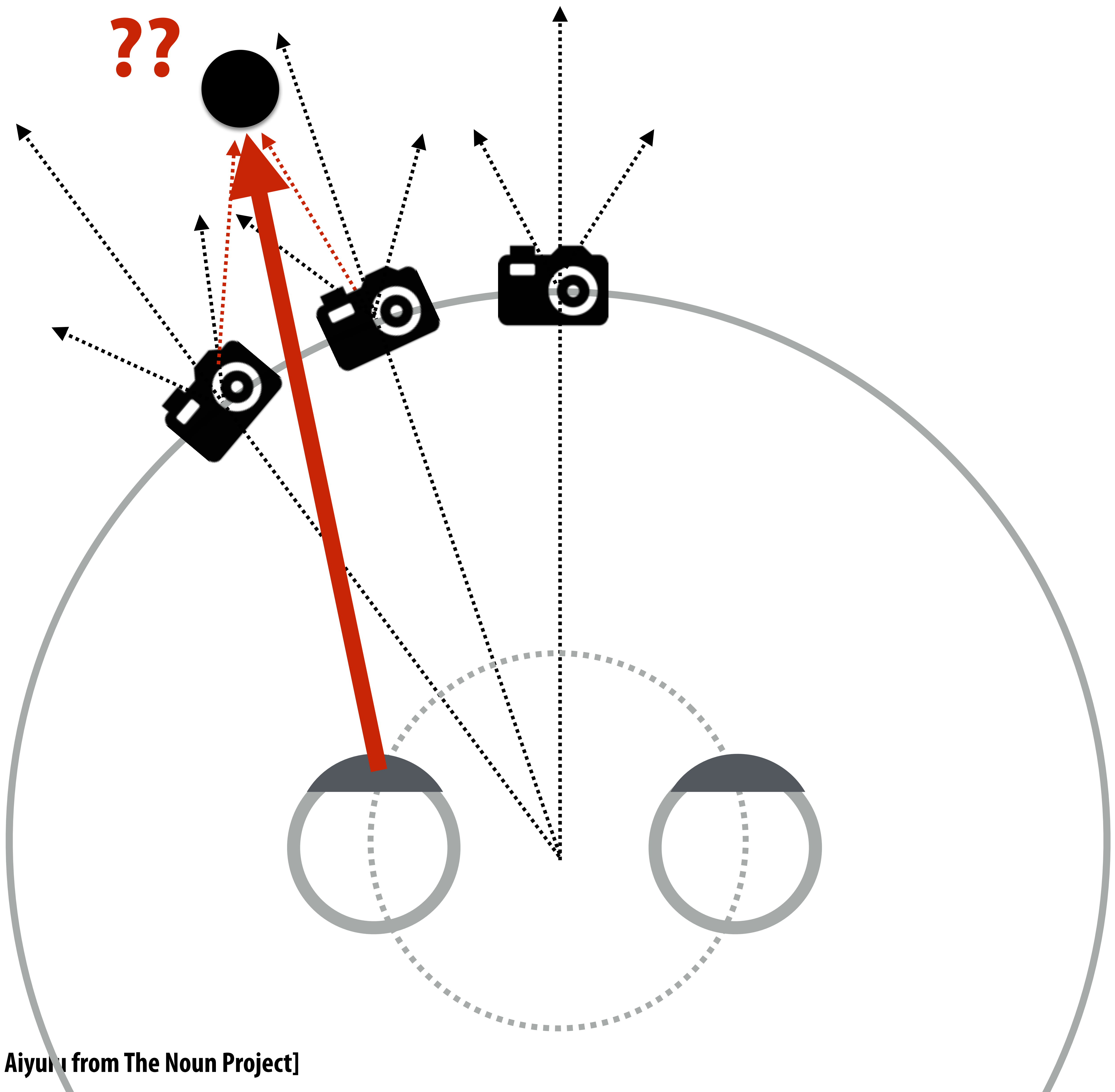


How to estimate rays at "missing" views?



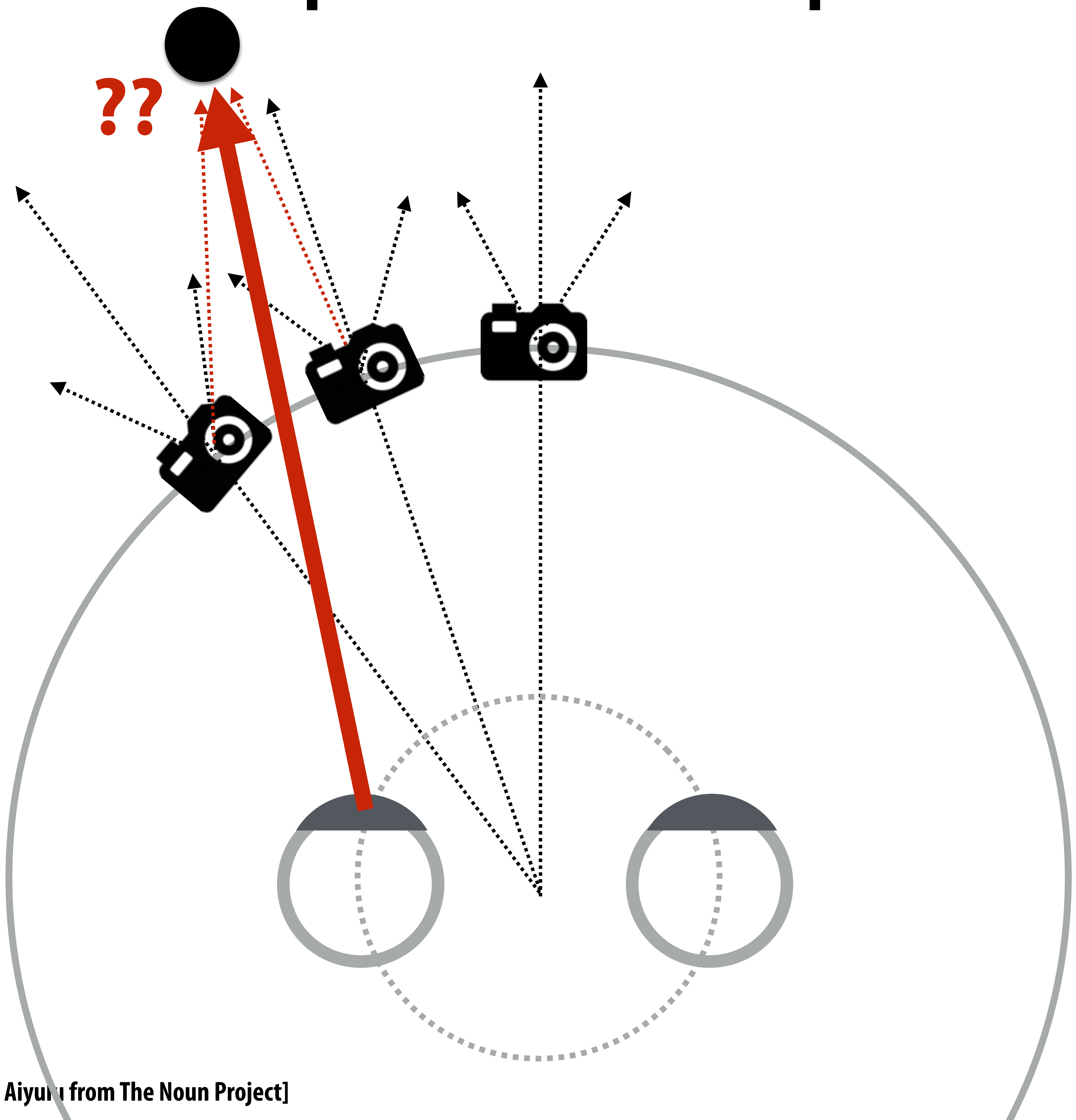
[Credit: Camera icon by Venkatesh Aiyulu from The Noun Project]

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

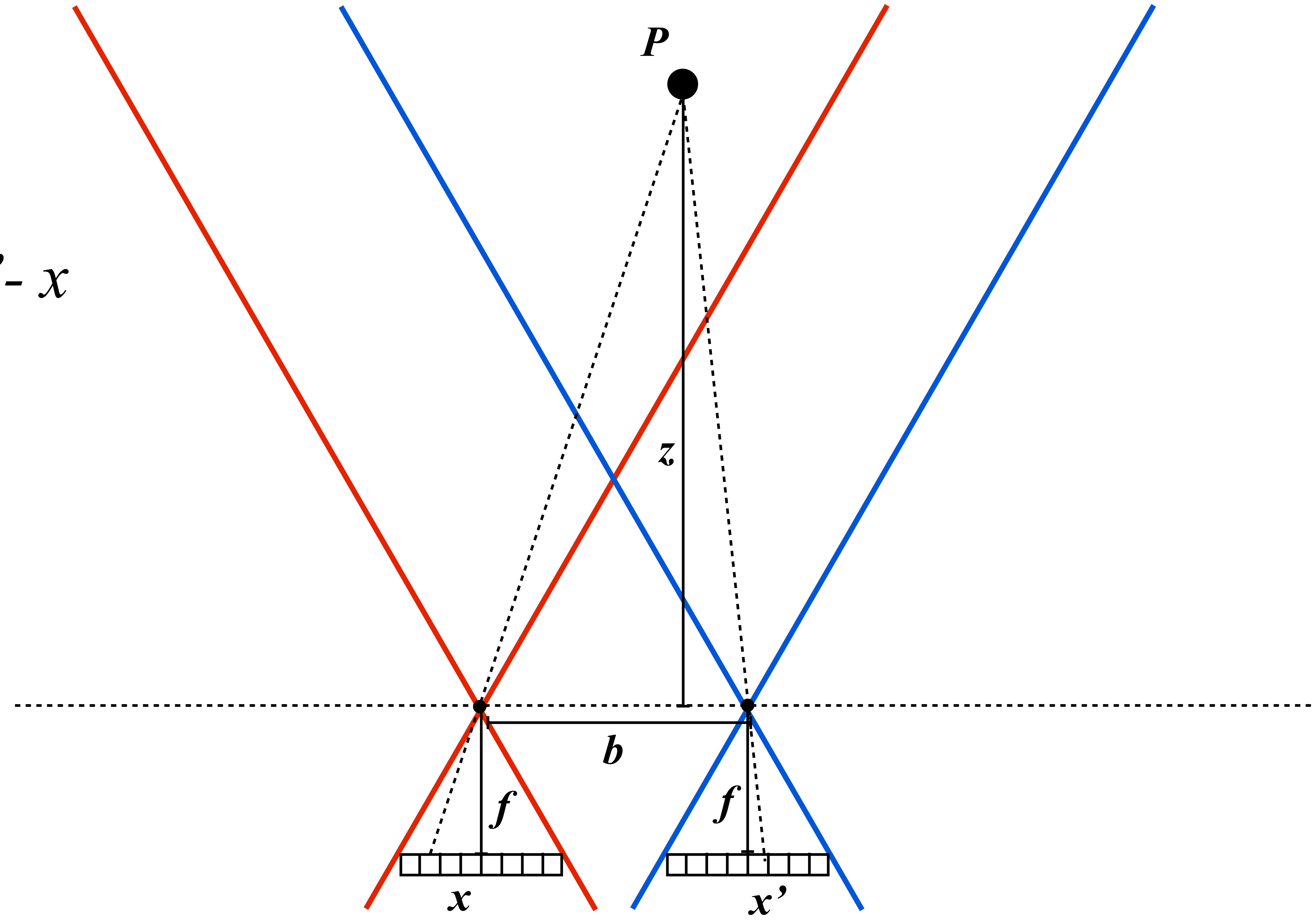
Binocular stereo 3D reconstruction of point *P*: depth from disparity

Focal length: f

Baseline: b

Disparity: $d = x' - x$

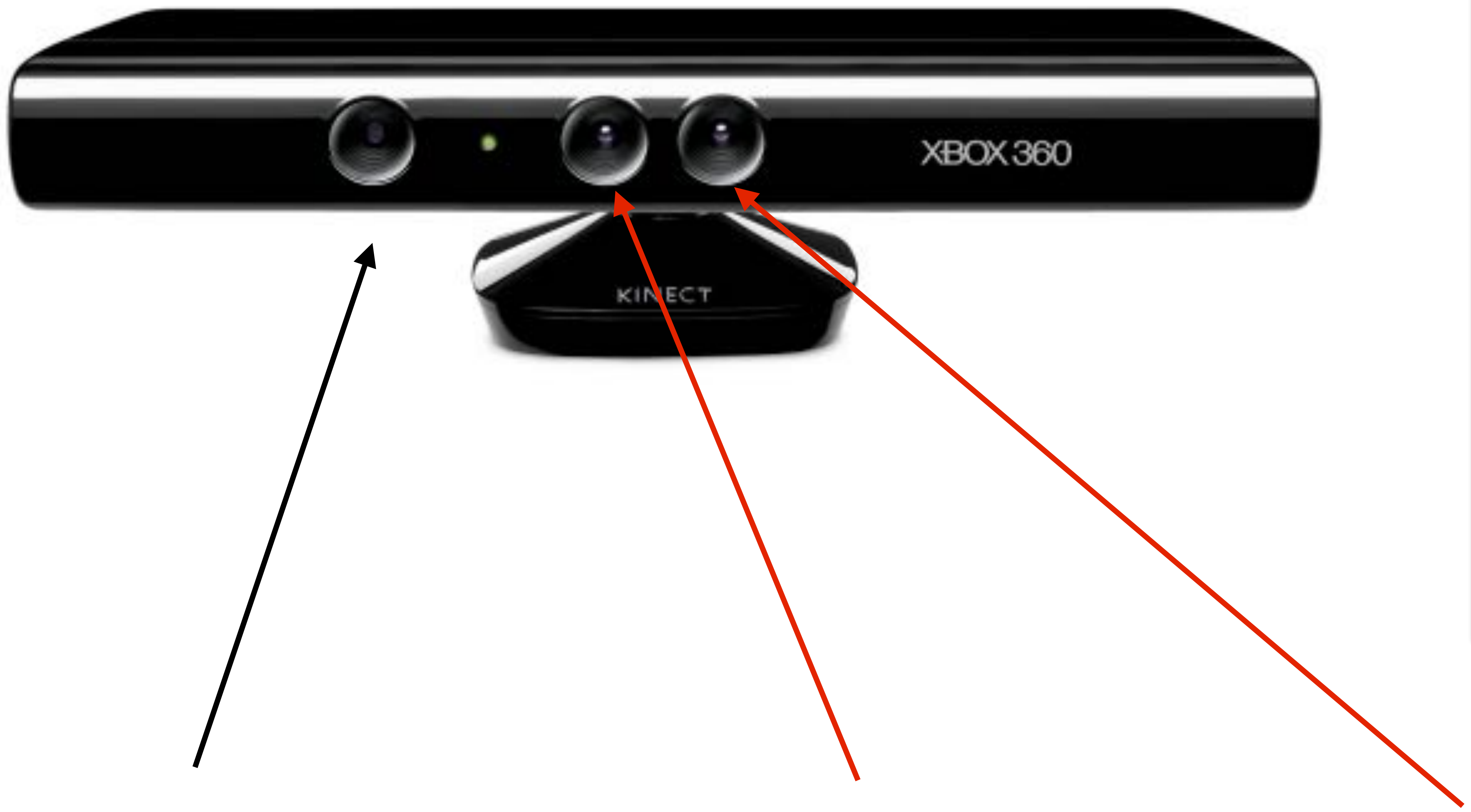
$$z = \frac{bf}{d}$$



Simple reconstruction example: cameras aligned (coplanar sensors), separated by known distance, same focal length

"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 **

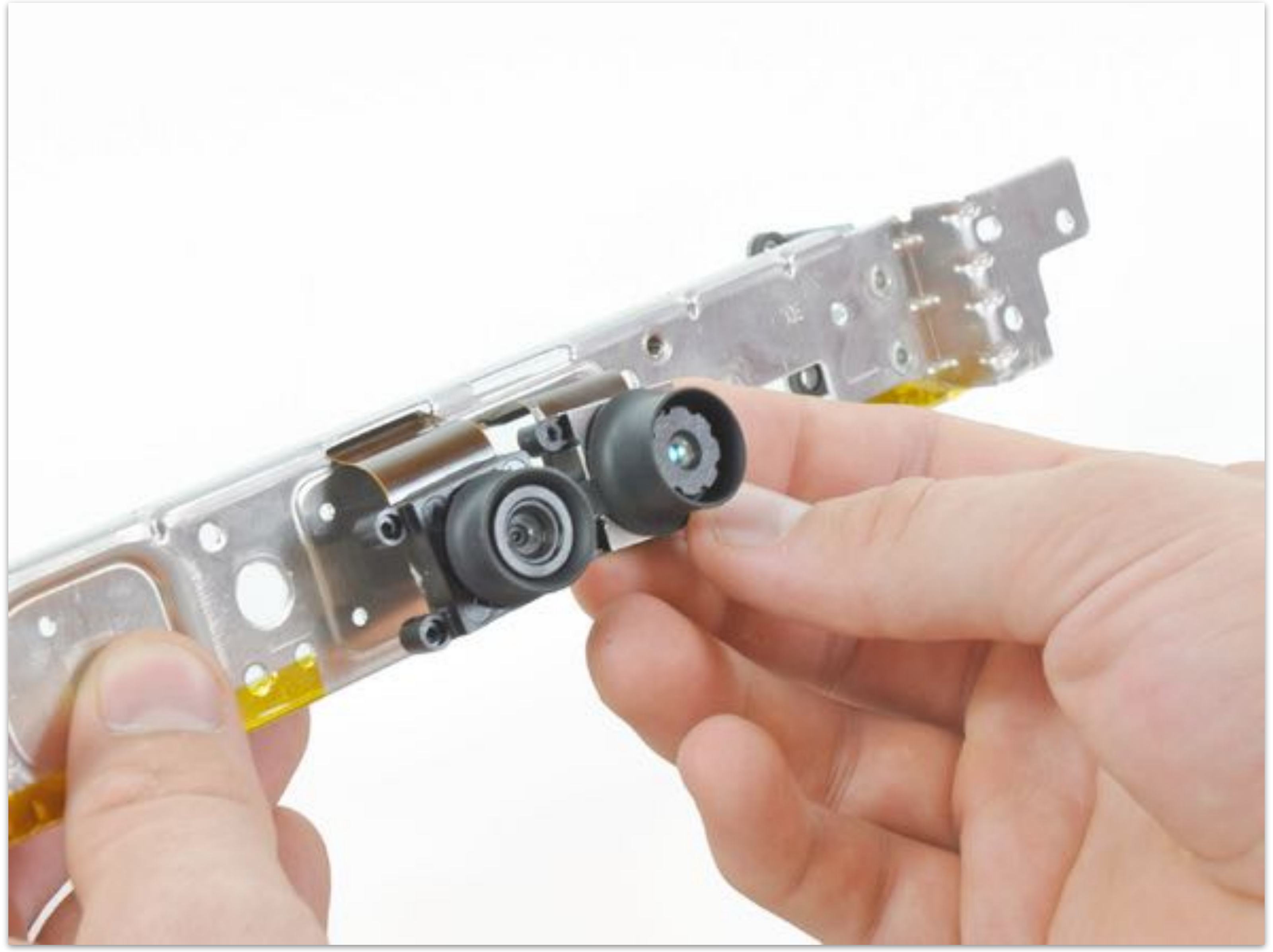
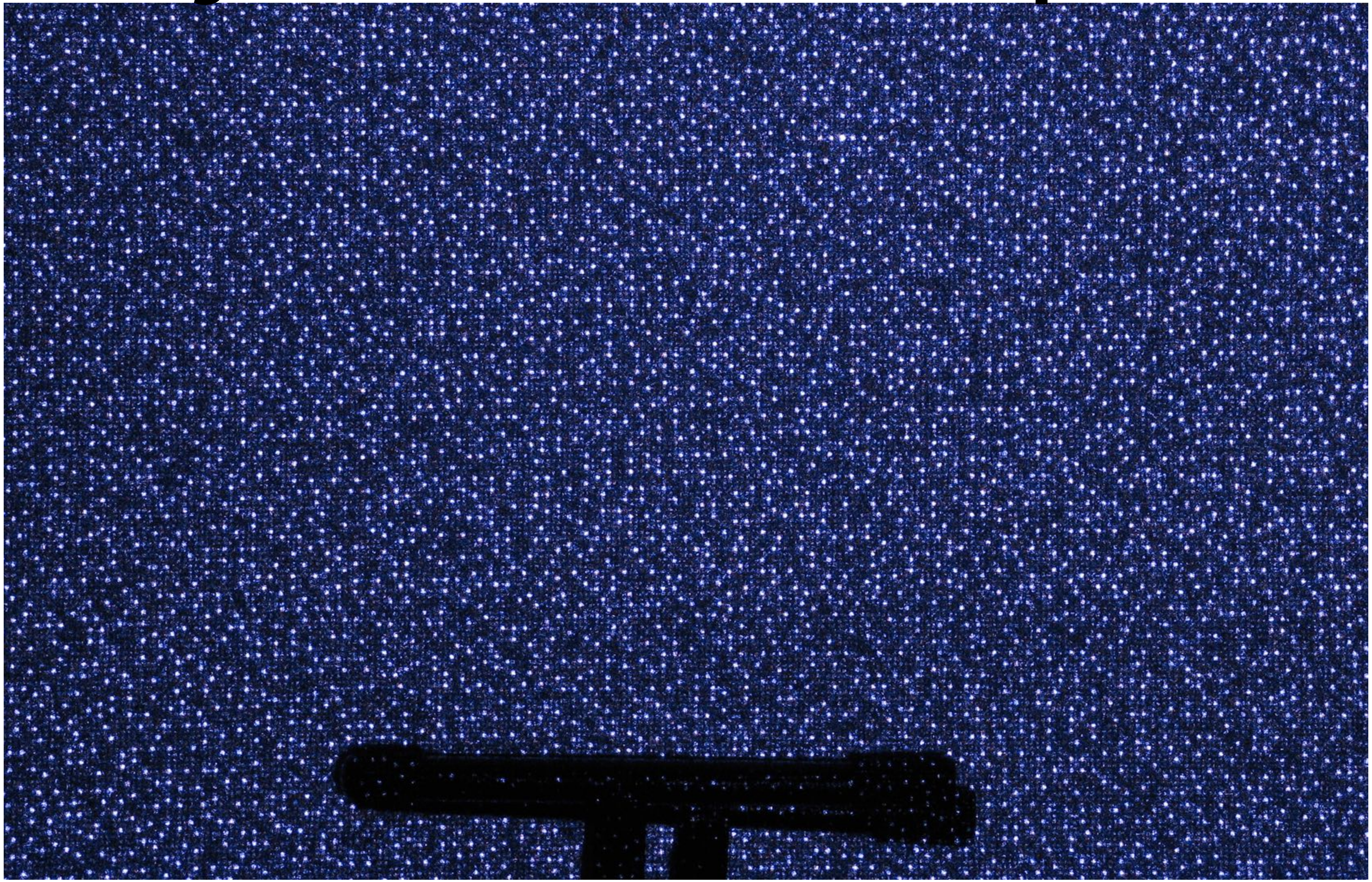


Image credit: iFixIt

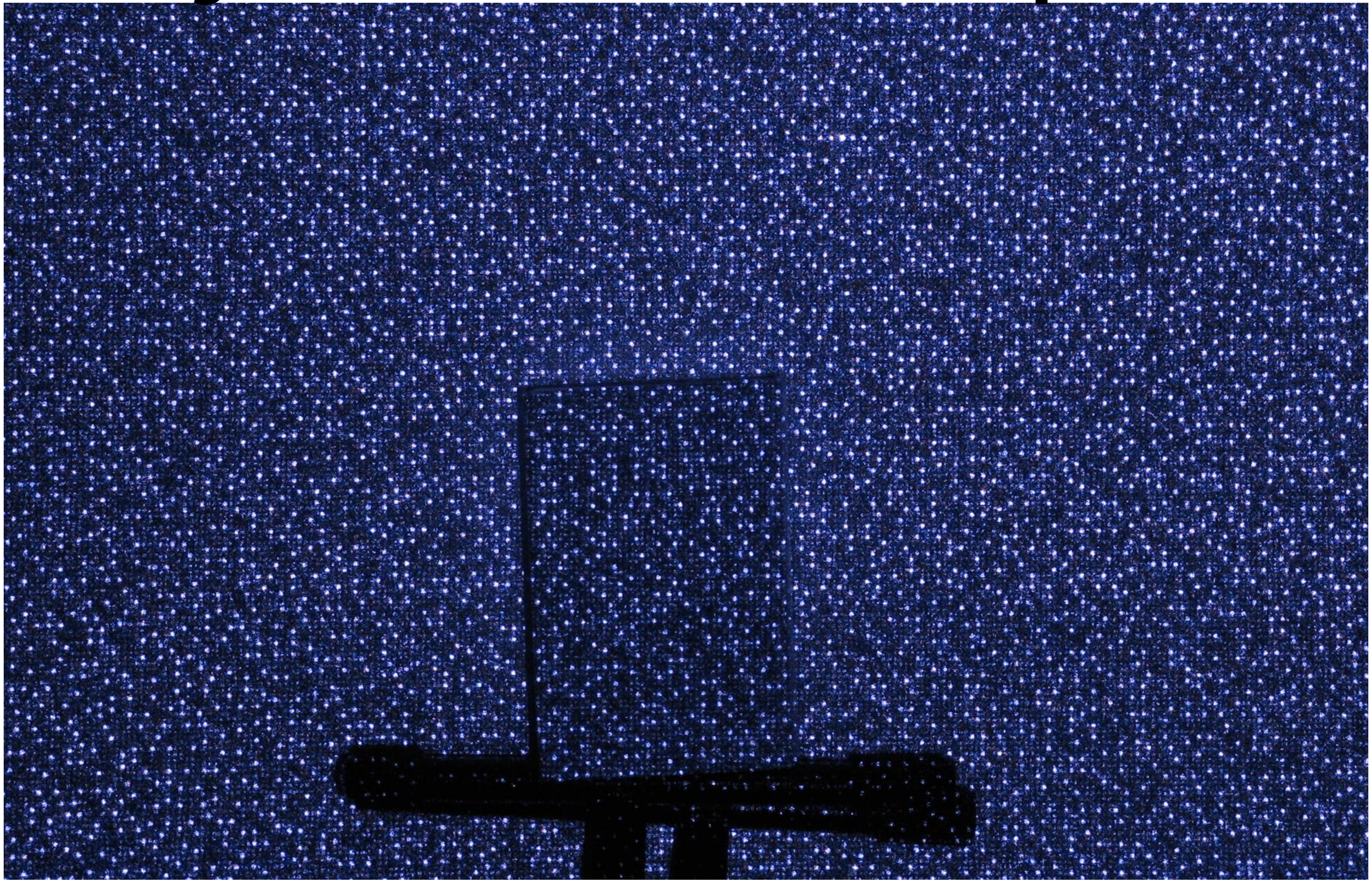
** Kinect returns 640x480 disparity image, suspect sensor is configured for 2x2 pixel binning down to 640x512, then crop

Infrared image of Kinect illuminant output



Credit: www.futurepicture.org

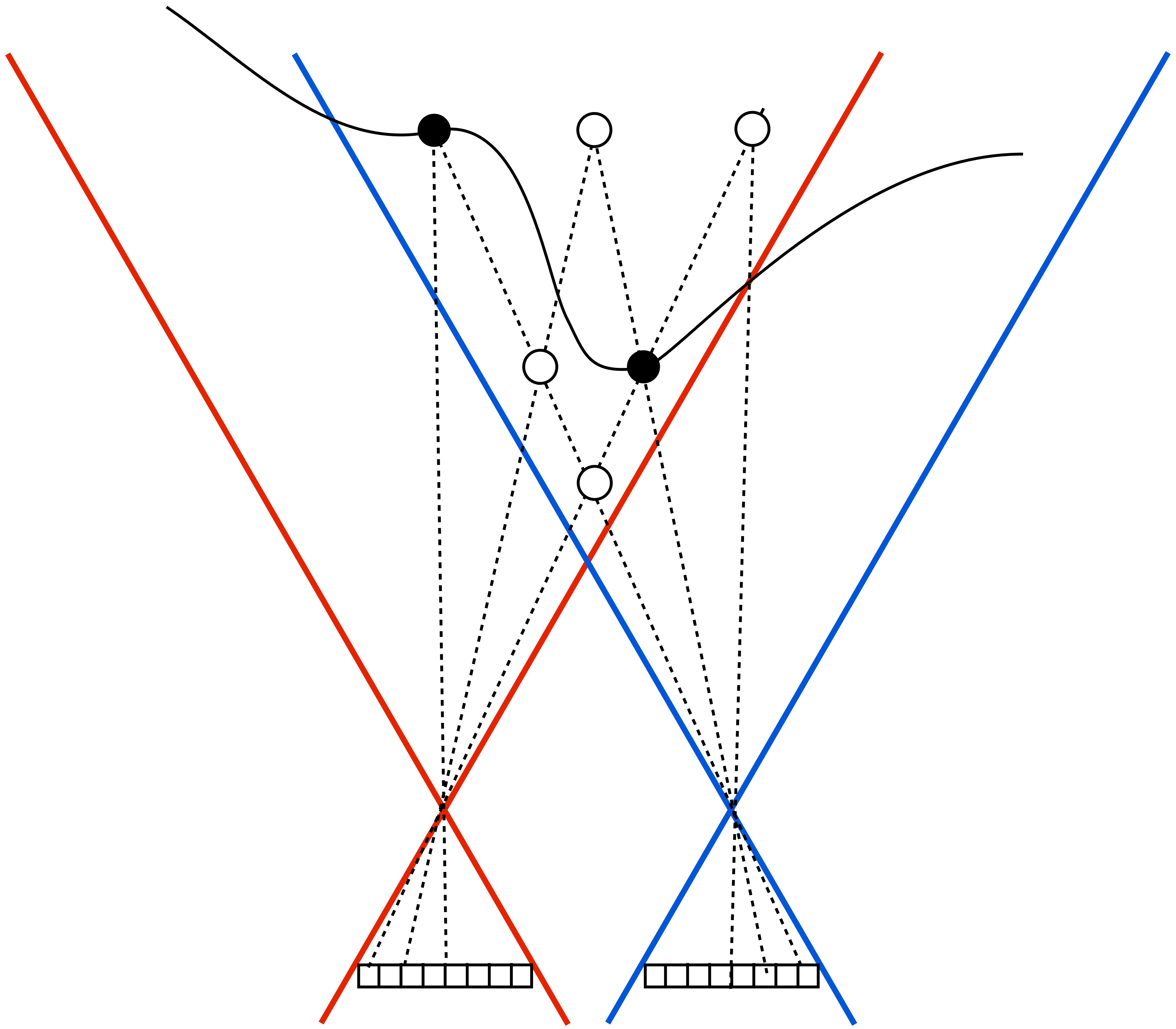
Infrared image of Kinect illuminant output



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?

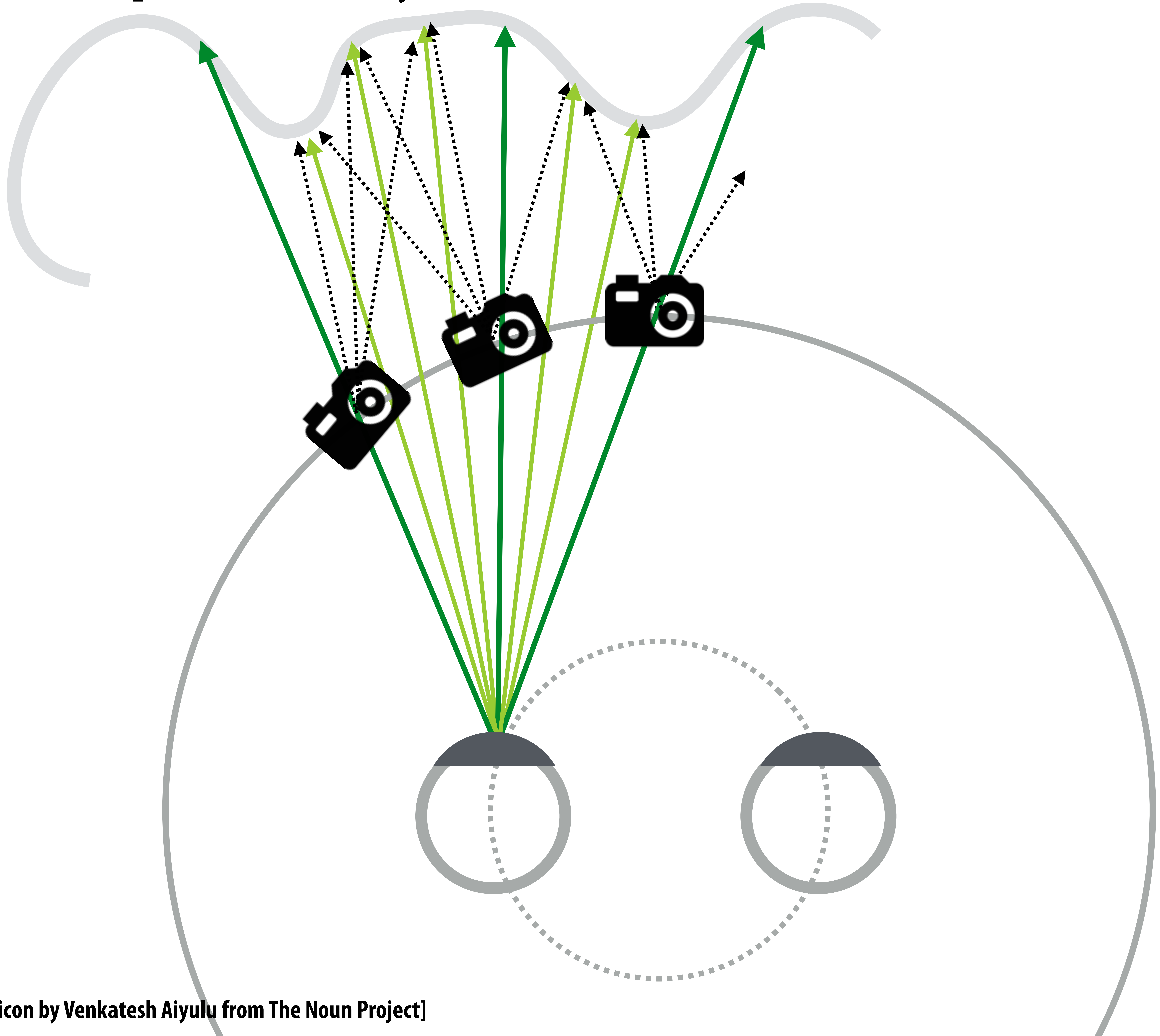


Correspondence problem = compute "flow" between adjacent cameras

- 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 at 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)

Image credit: Andersen et al. 2016

Left eye: with interpolated rays

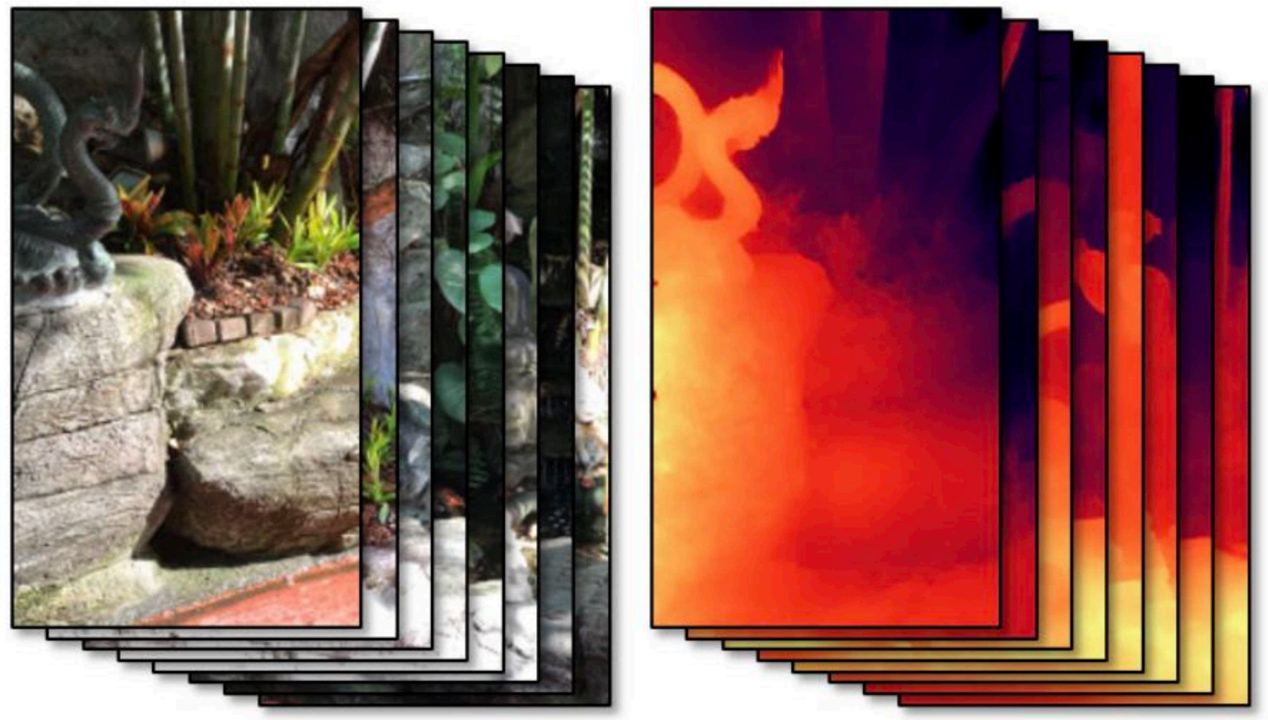


[Credit: Camera icon by Venkatesh Aiyulu from The Noun Project]

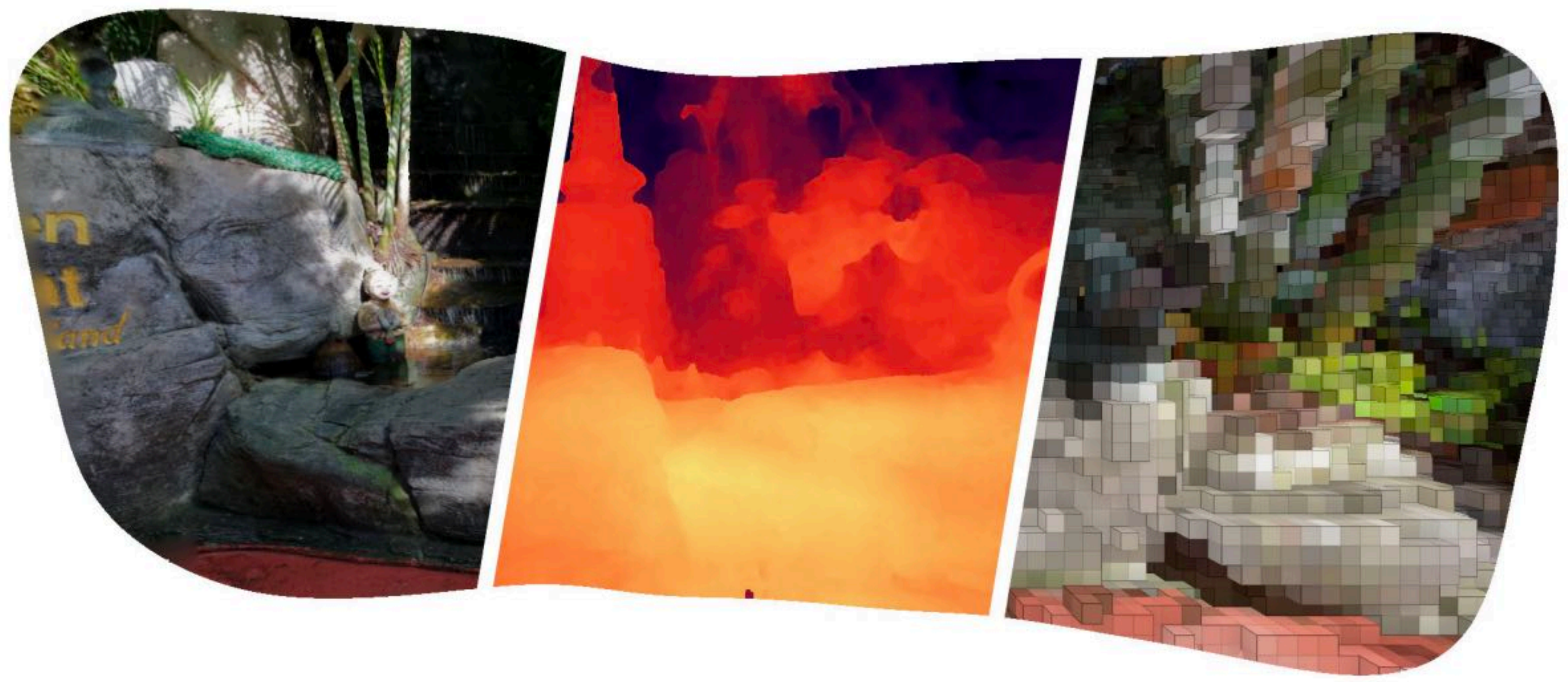
“Casual 3D photography”

- **Acquisition:** wave a smartphone camera around to acquire images of scene from multiple viewpoints
- **Processing:** construct 3D representation of scene from photos
  - Render 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

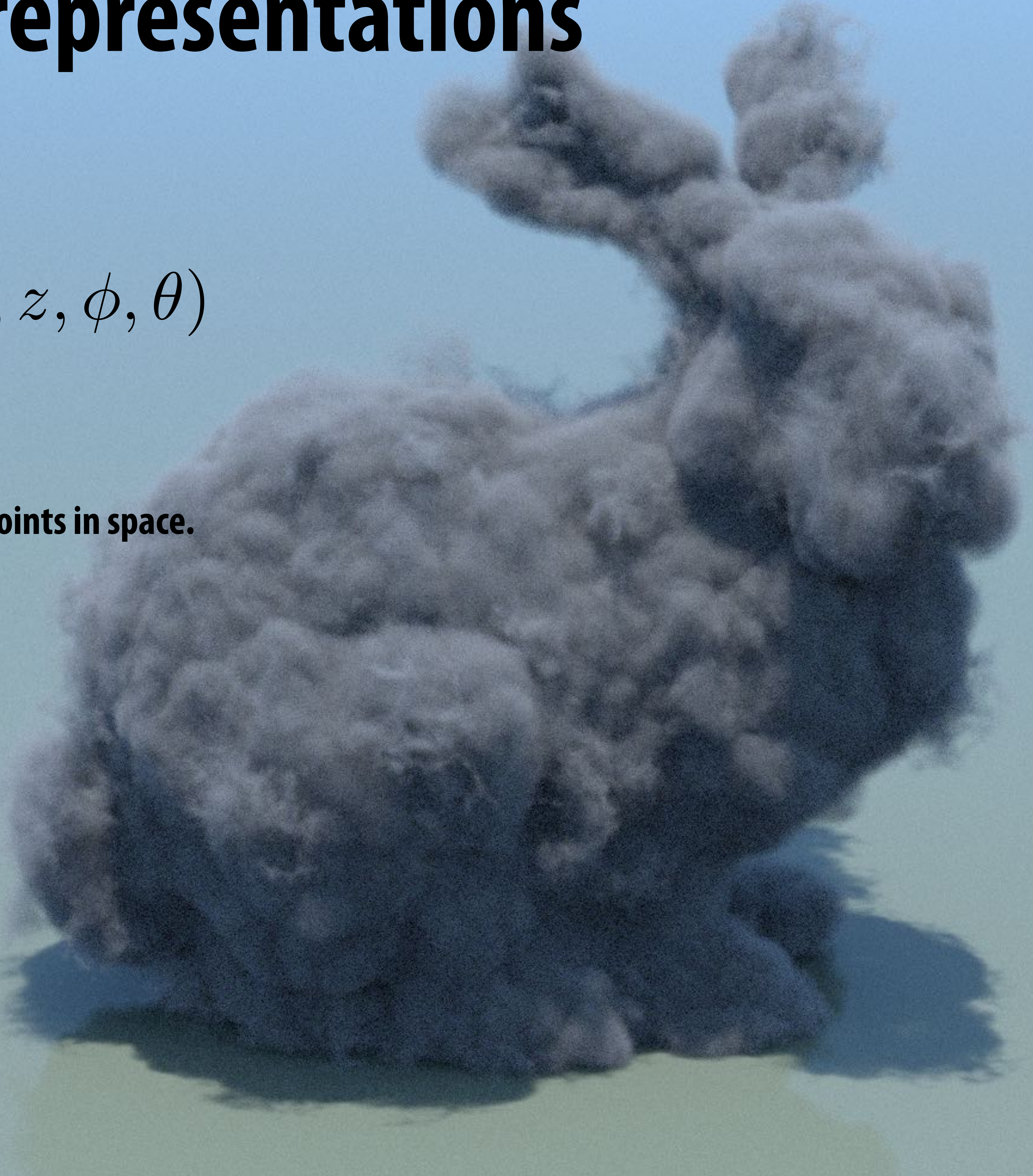
But it's hard to estimate depth or geometry

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

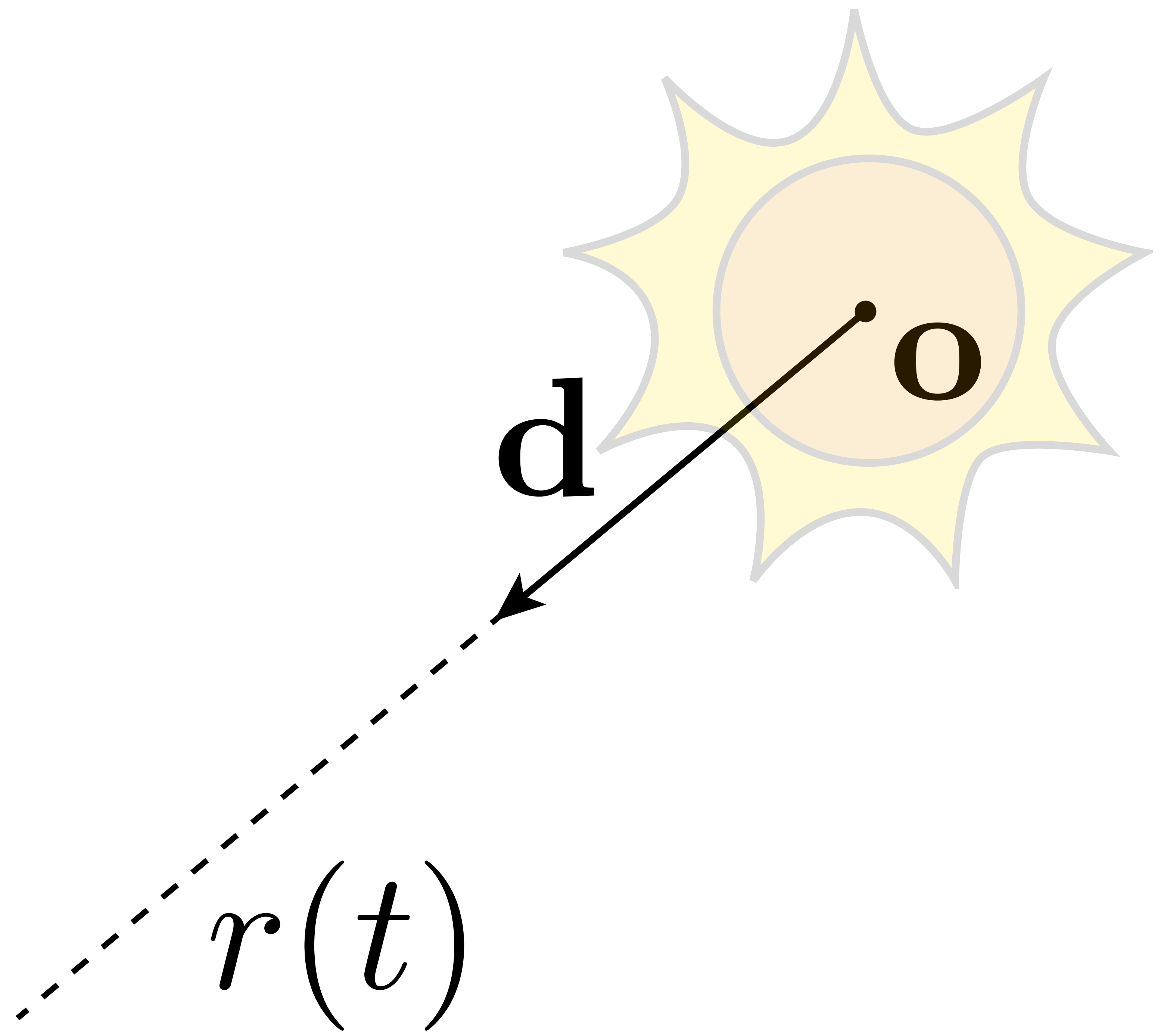
origin

unit direction

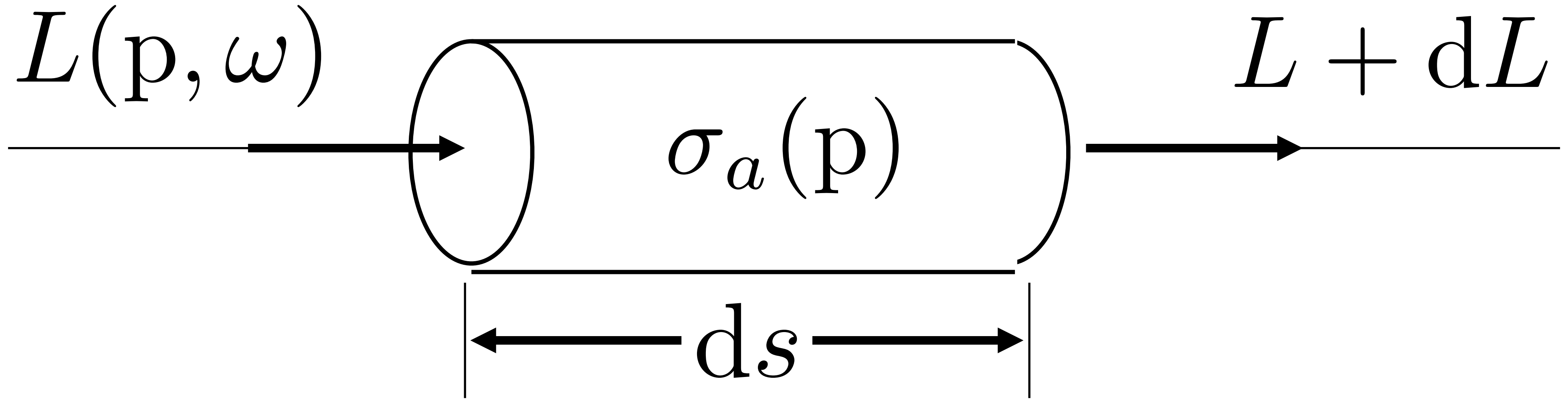
$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$$

point along ray

"Distance" or "time"



Absorption in a volume



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

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

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

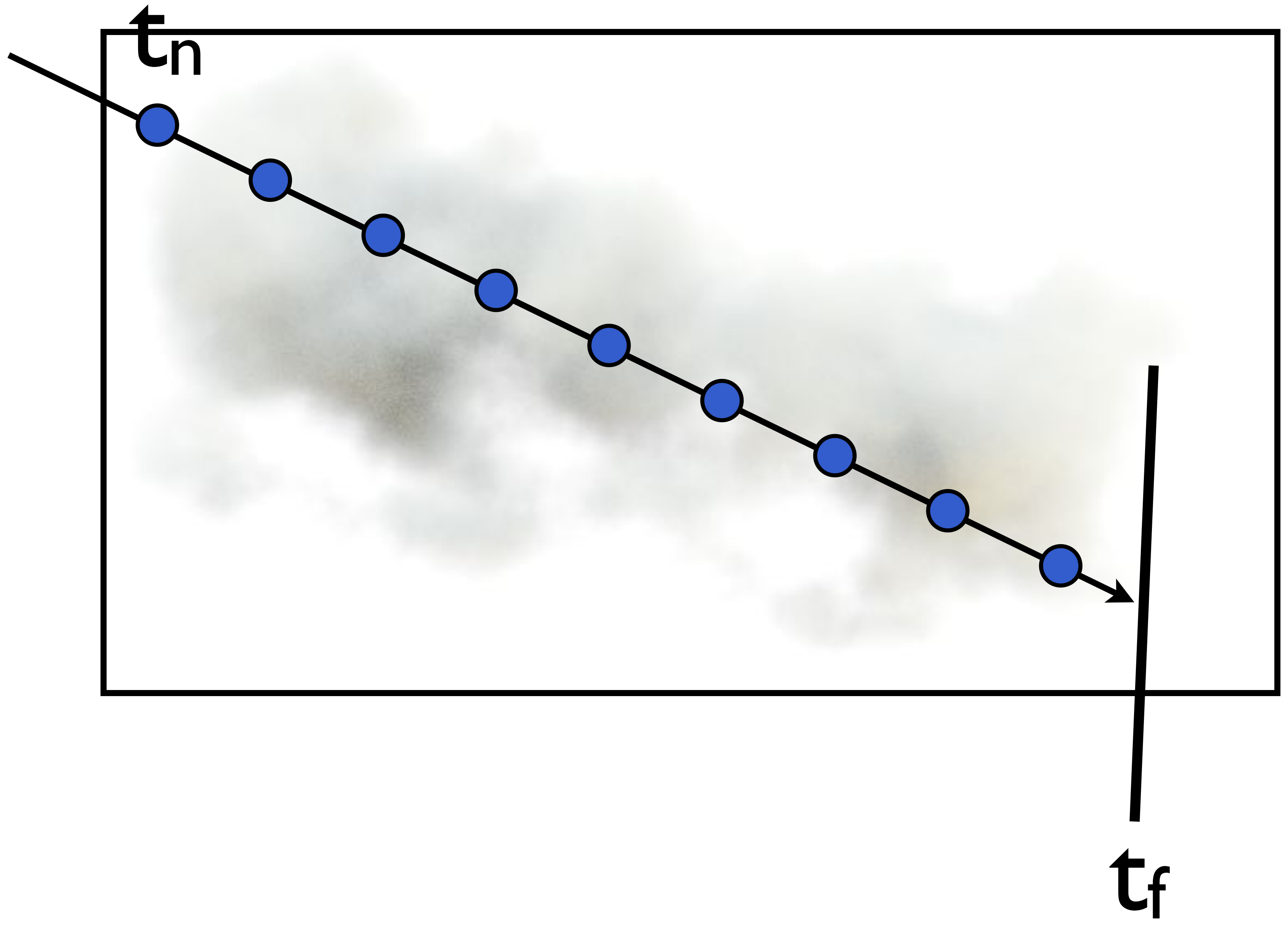
- $L(\mathrm{p}, \omega)$ **light energy (radiance) along a ray from p in direction w**
- **Absorption cross section at point in space:** $\sigma_a(\mathrm{p})$
  - **Probability of being absorbed per unit length**
  - **Units: 1/distance**

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

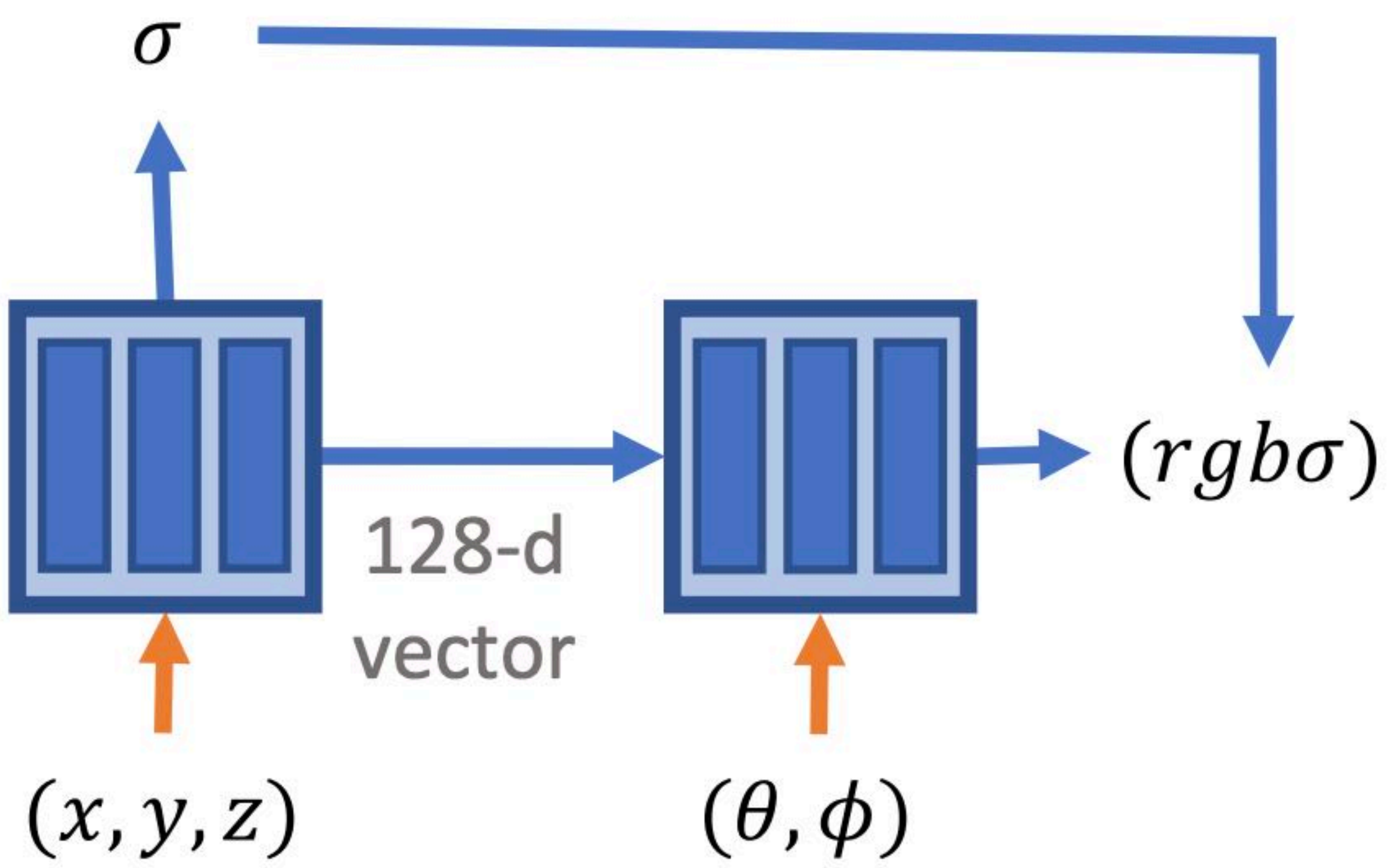
Typical challenge:
limited resolution

Credit: Voxel Ville NFT (voxelville.io)

Learning (compressed) representations

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

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



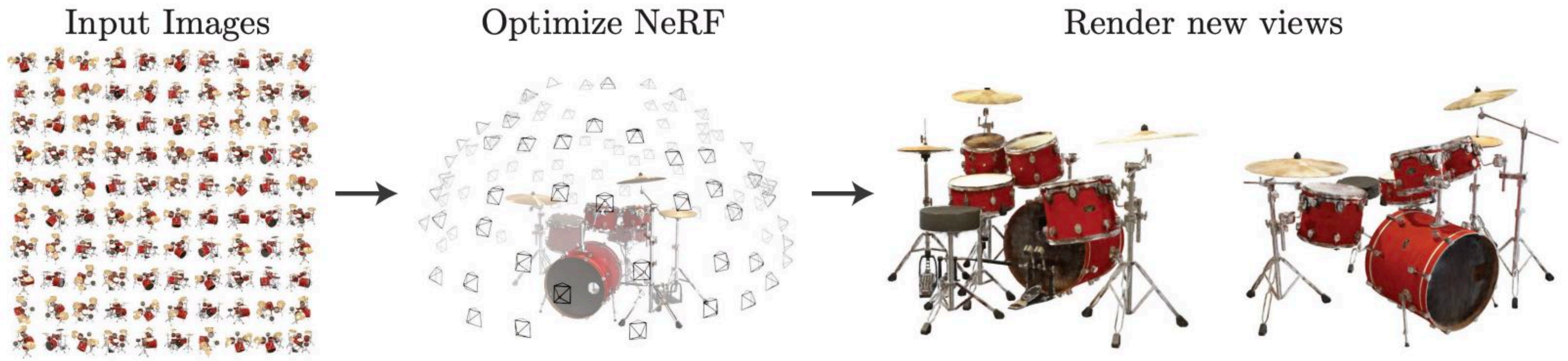
Learning better (more compressed) representations

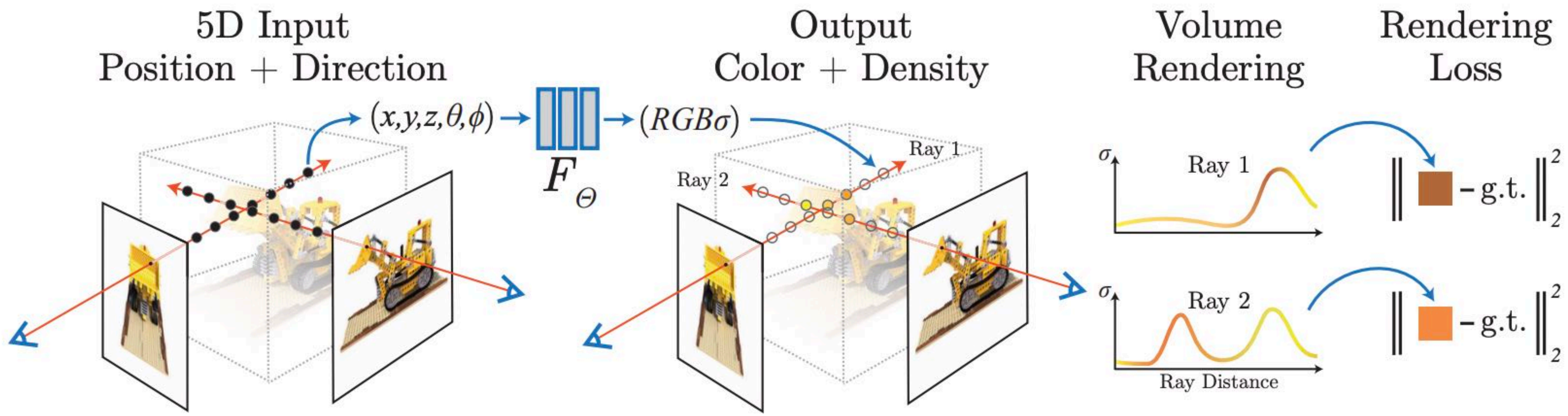
- **Why not just learn an approximation to the continuous function:**

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

- **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)



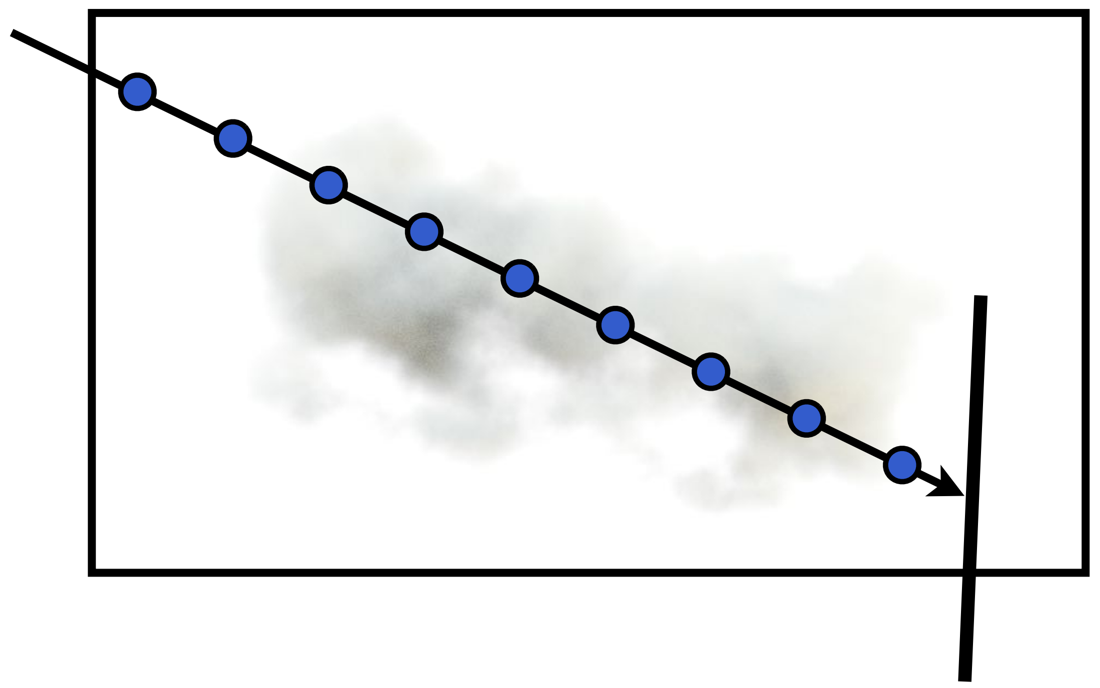


What just happened?

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

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

Demos

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.
  - Better representation than a dense voxel grid.