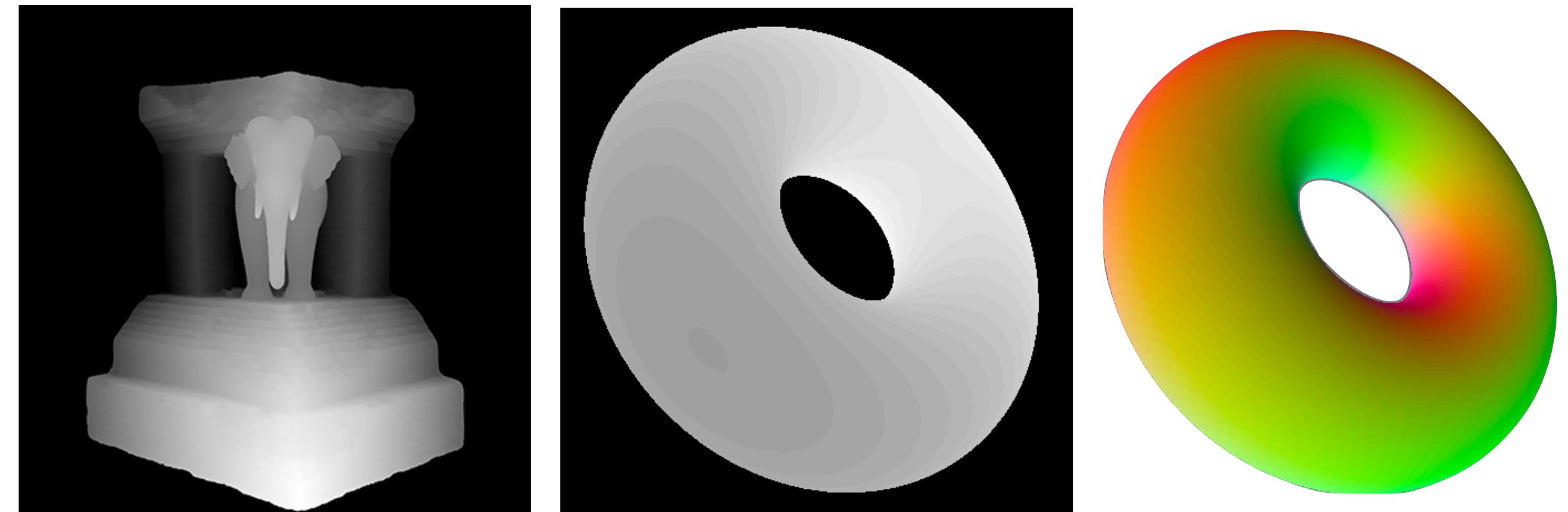
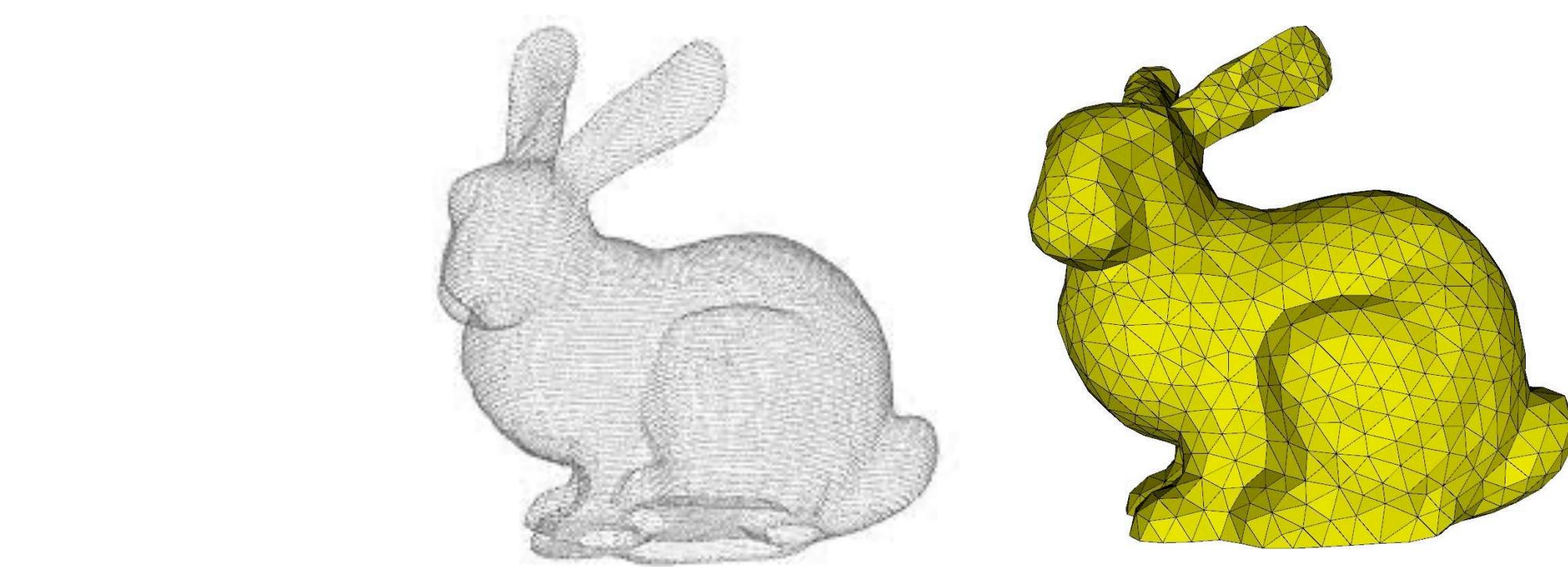


SCENE REPRESENTATIONS

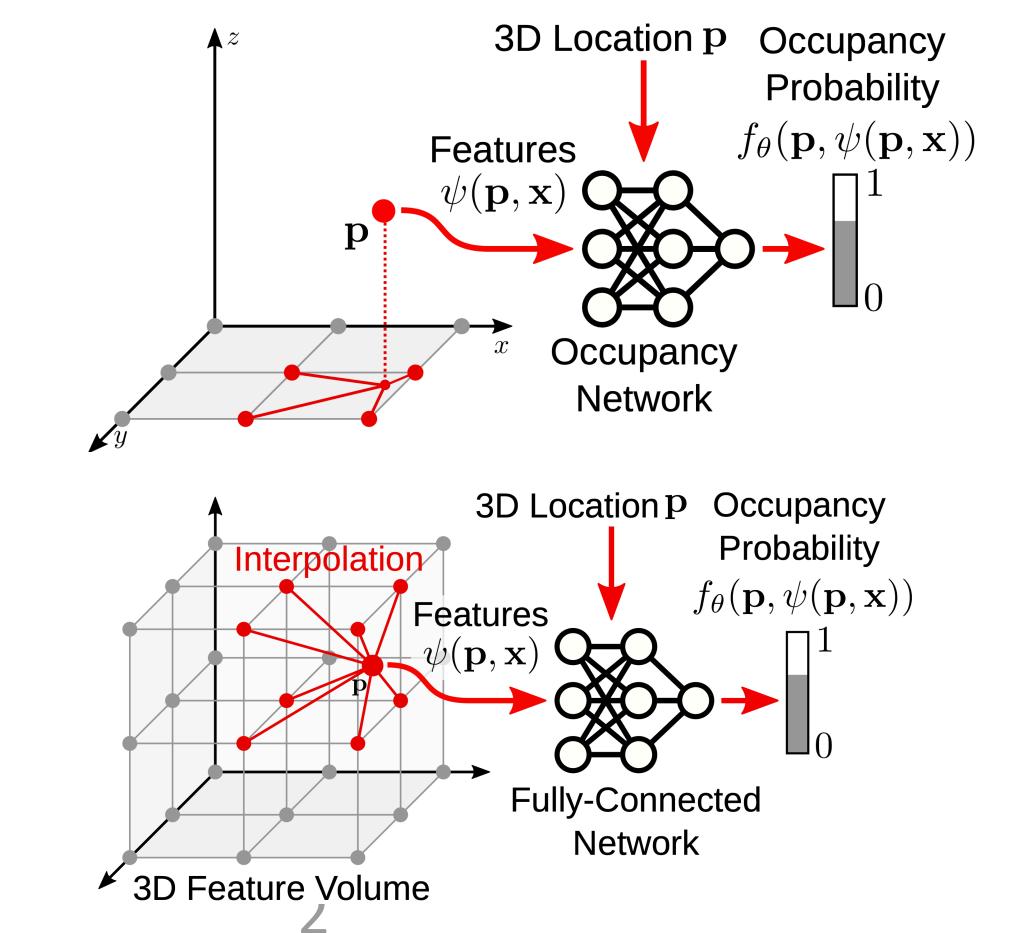
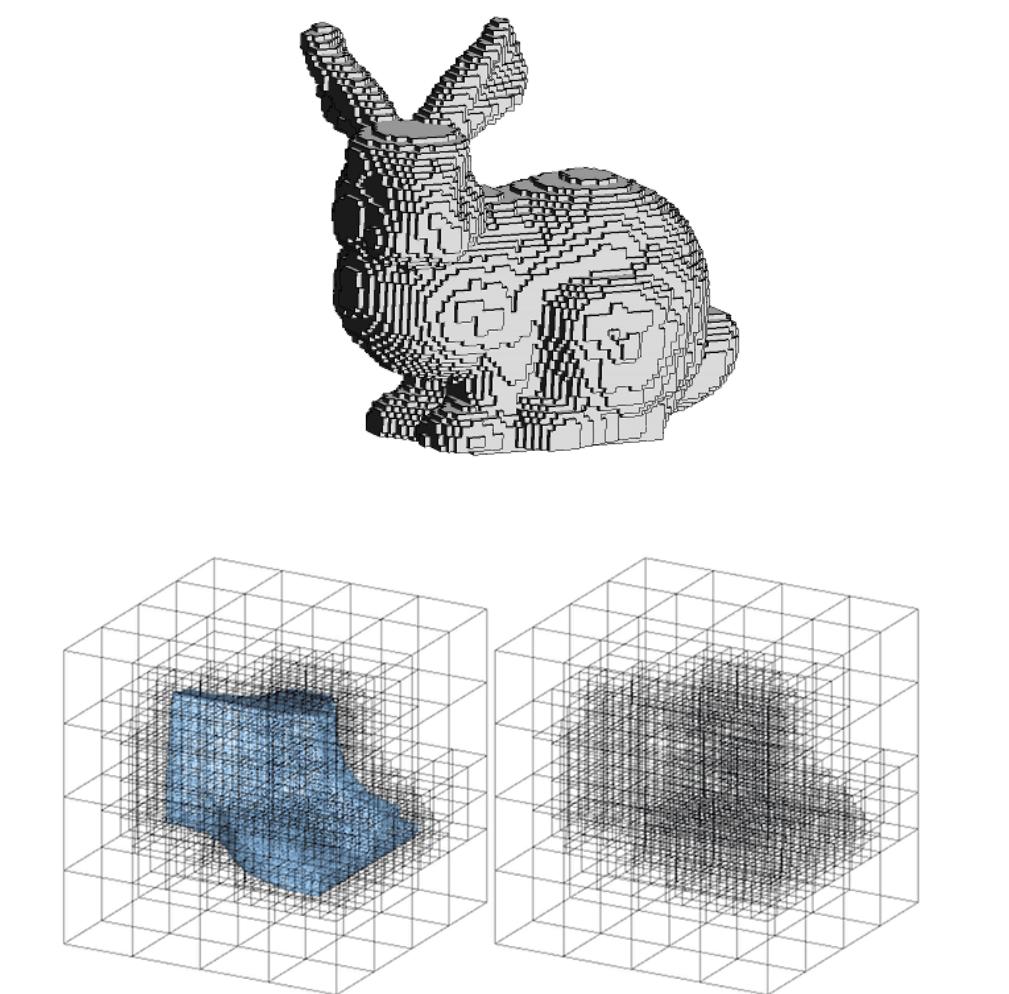
2.5D Representations



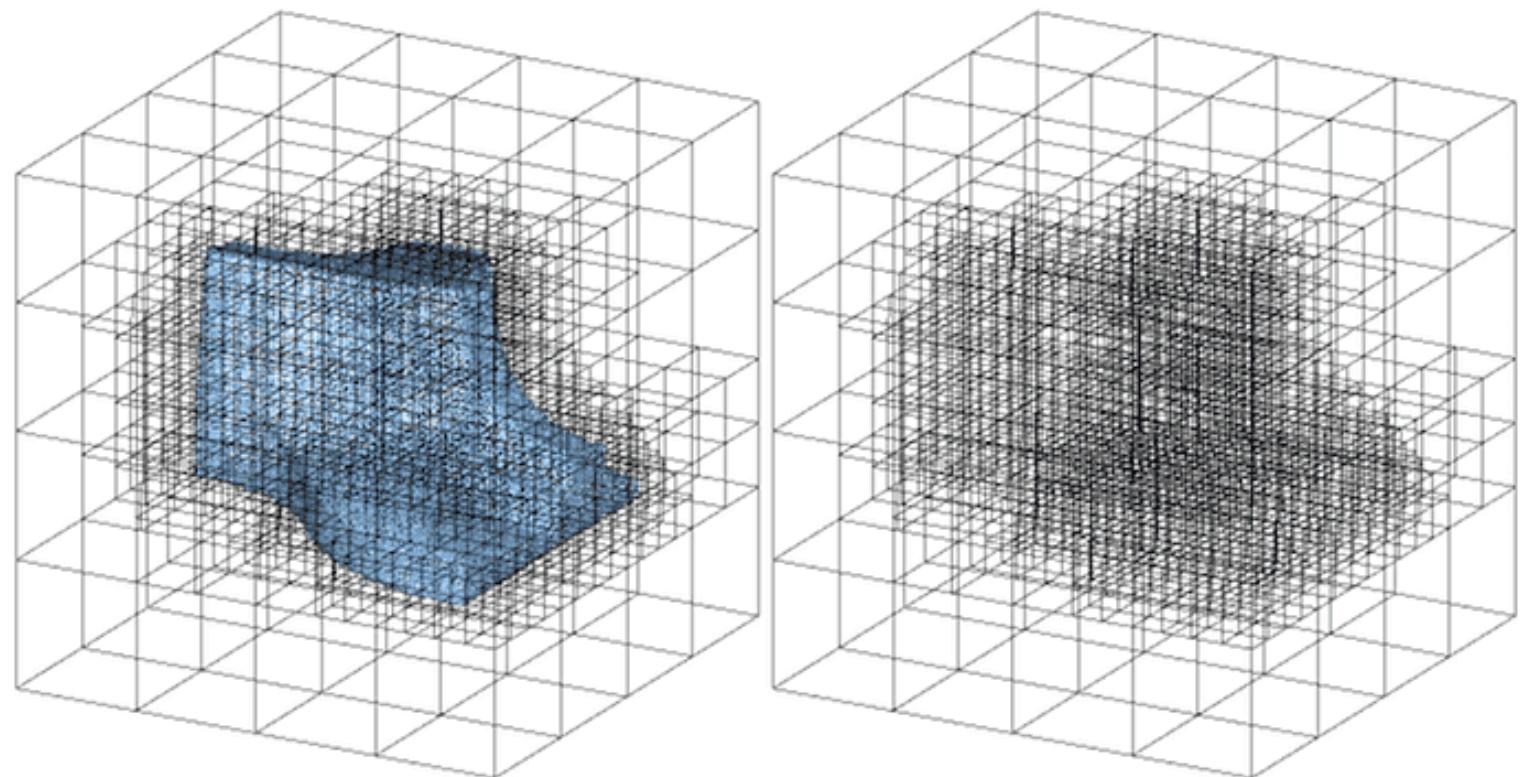
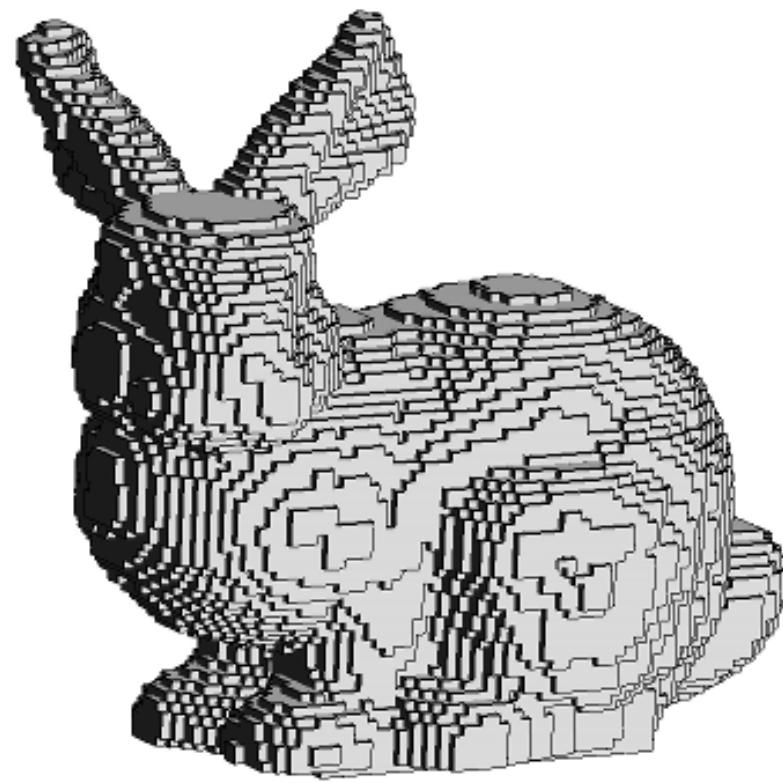
Surface Representations



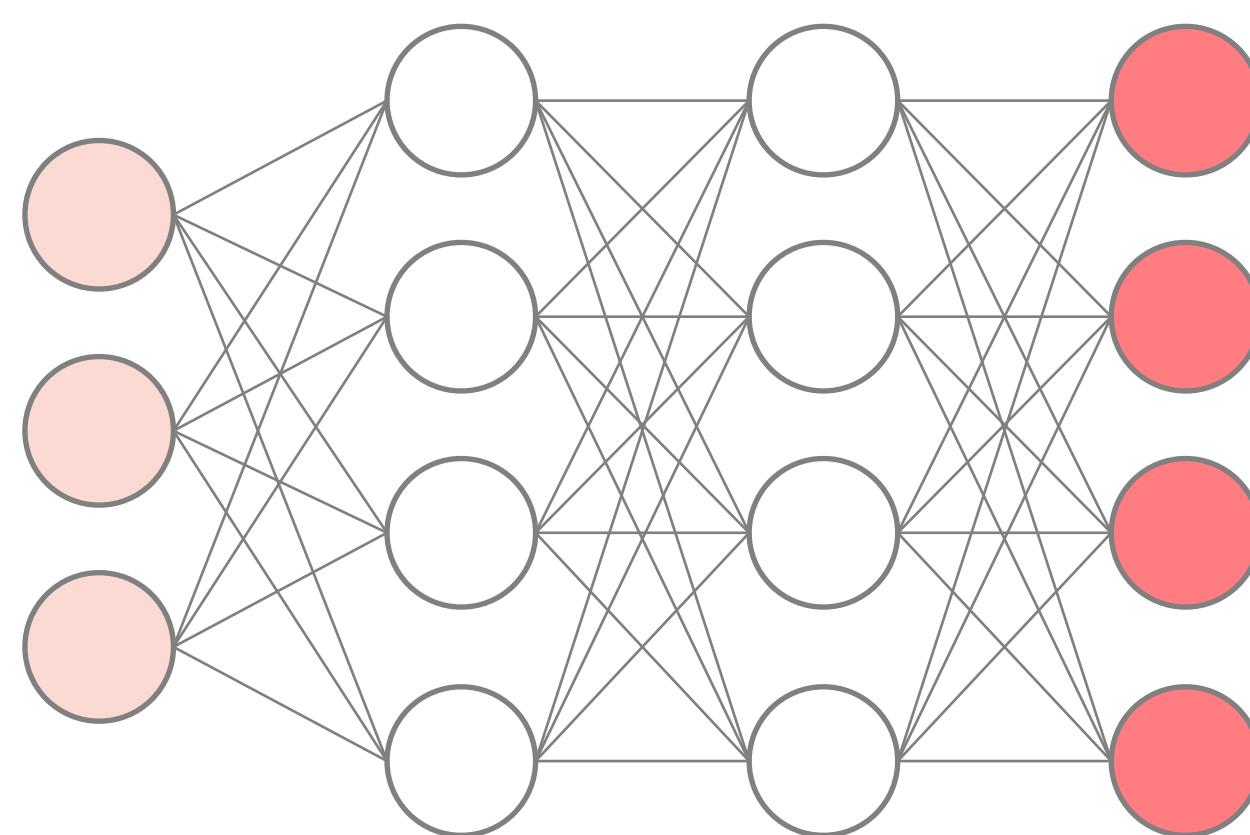
Field Parameterizations



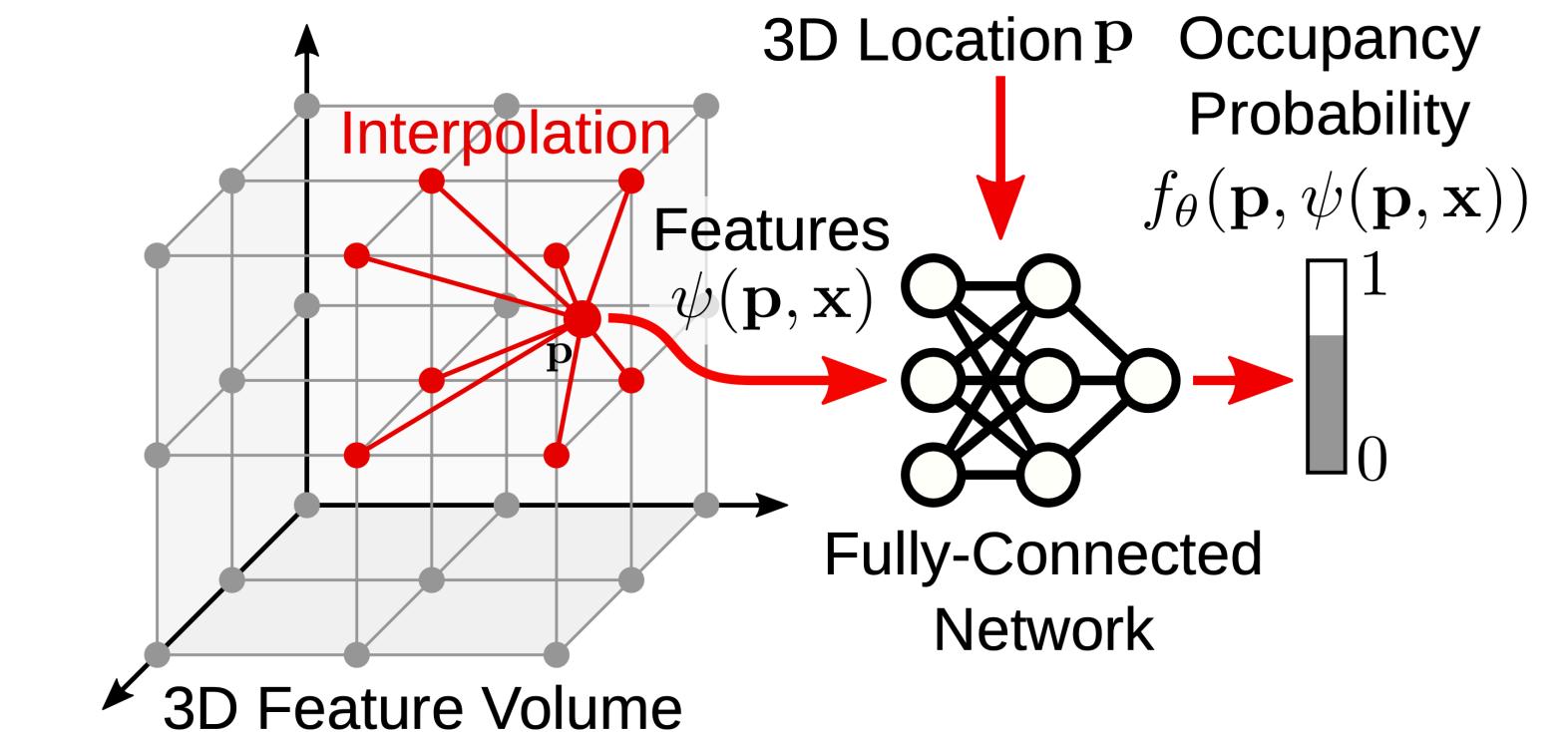
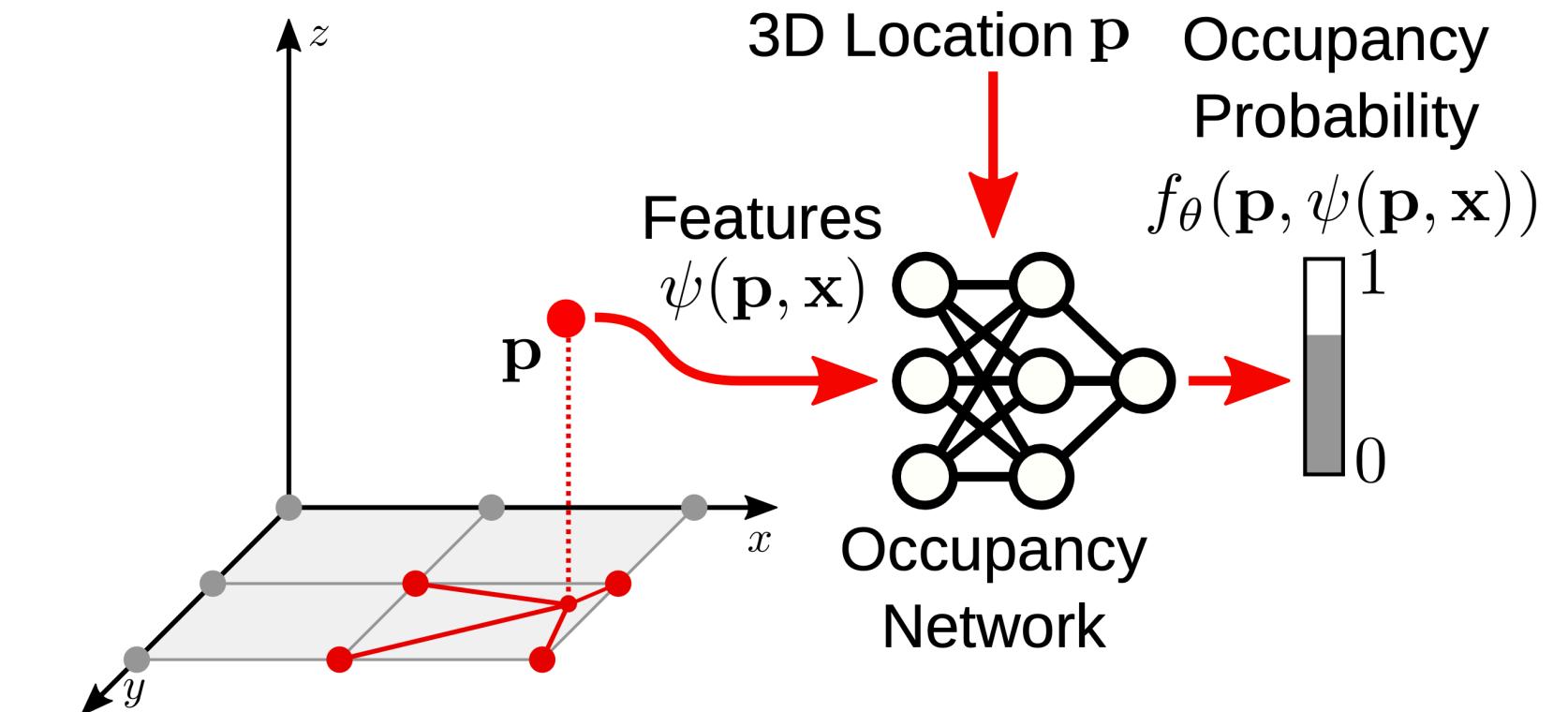
Parameterizing Fields



Discrete Parameterizations

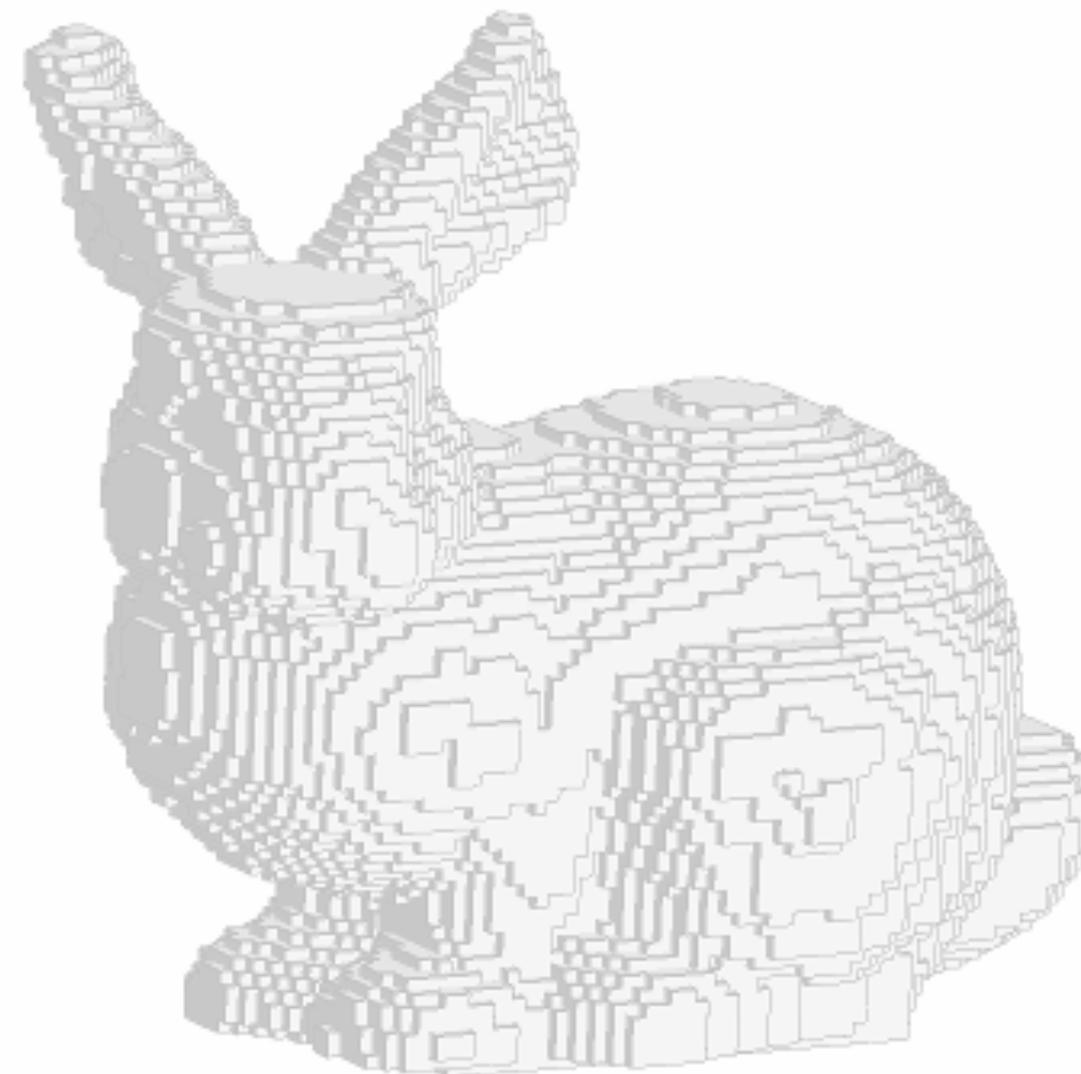


Continuous Parameterizations

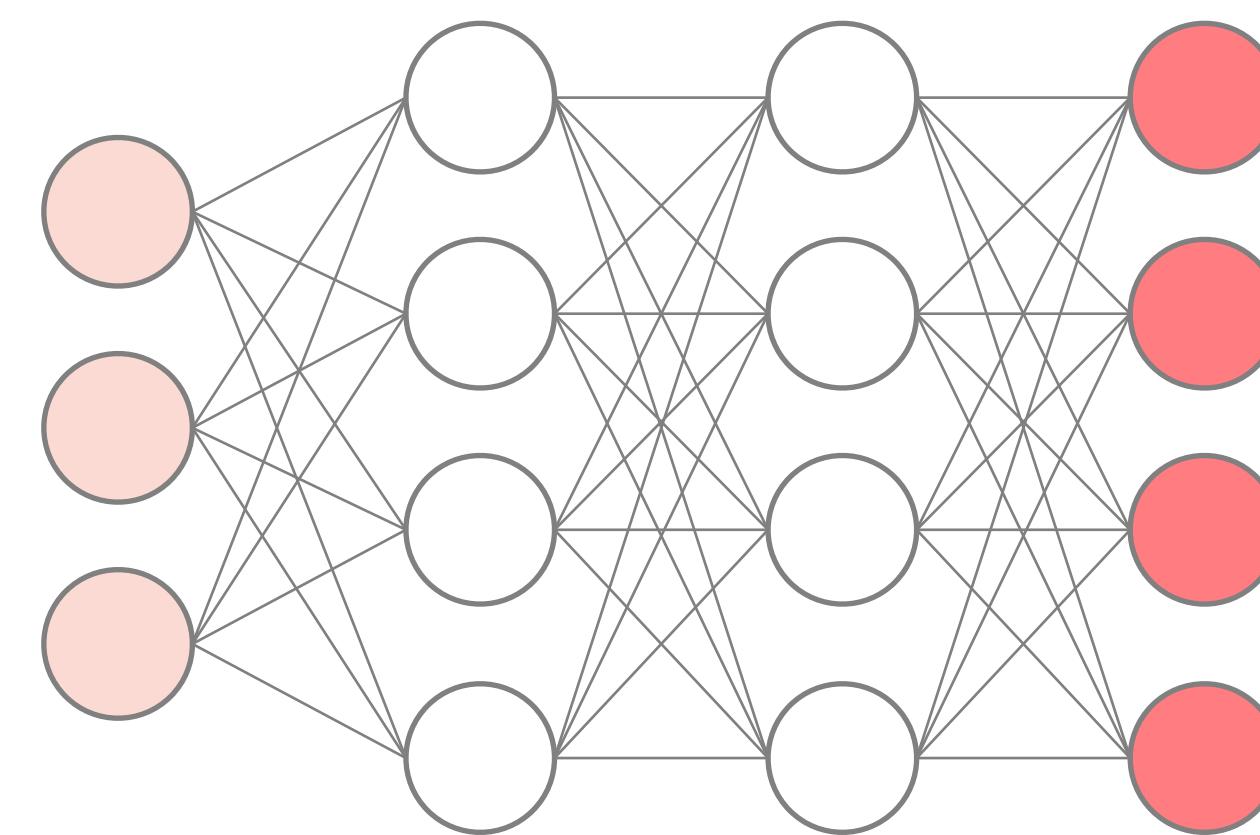


Hybrid Parameterizations

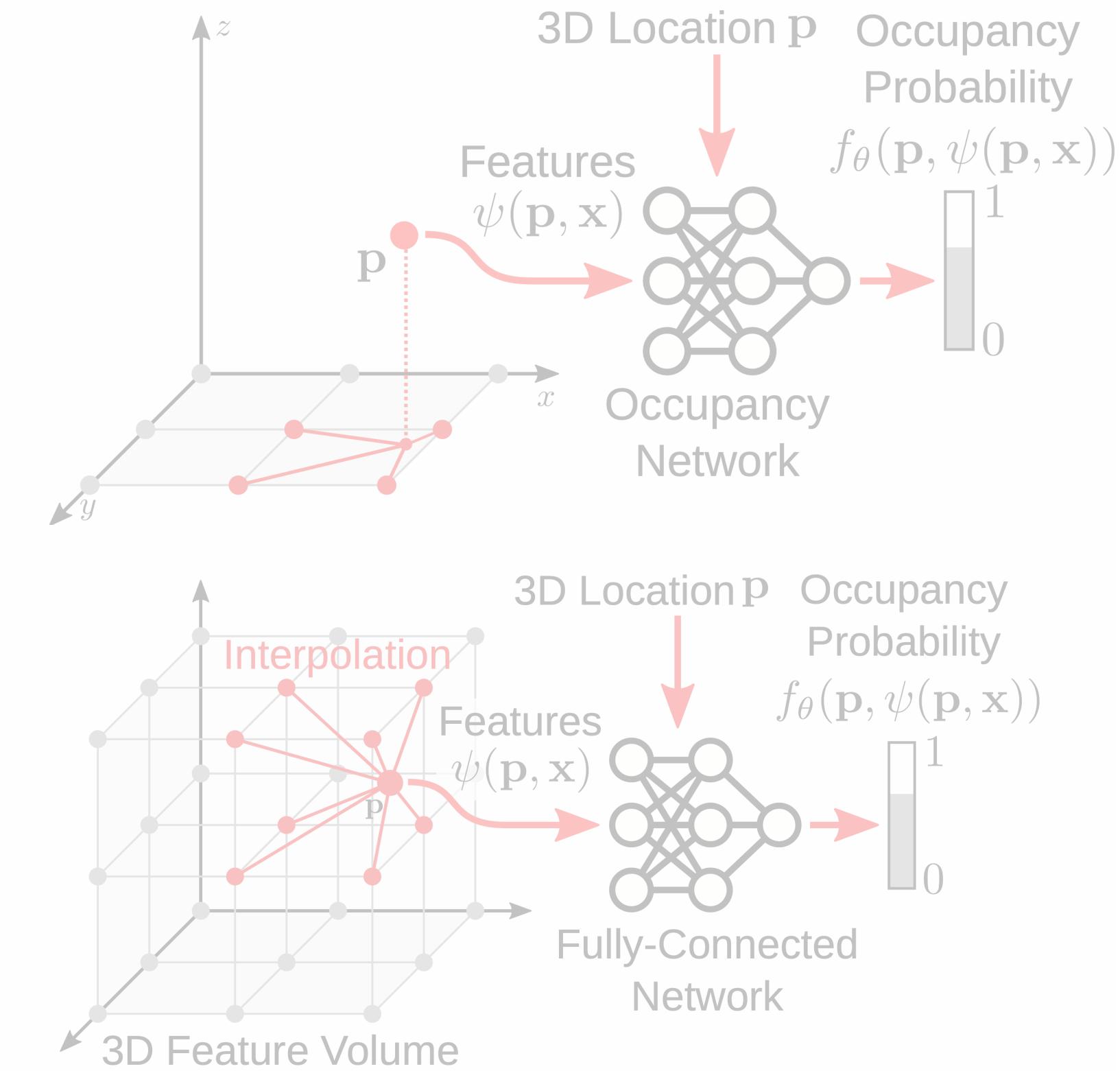
Neural Fields



Discrete Parameterizations

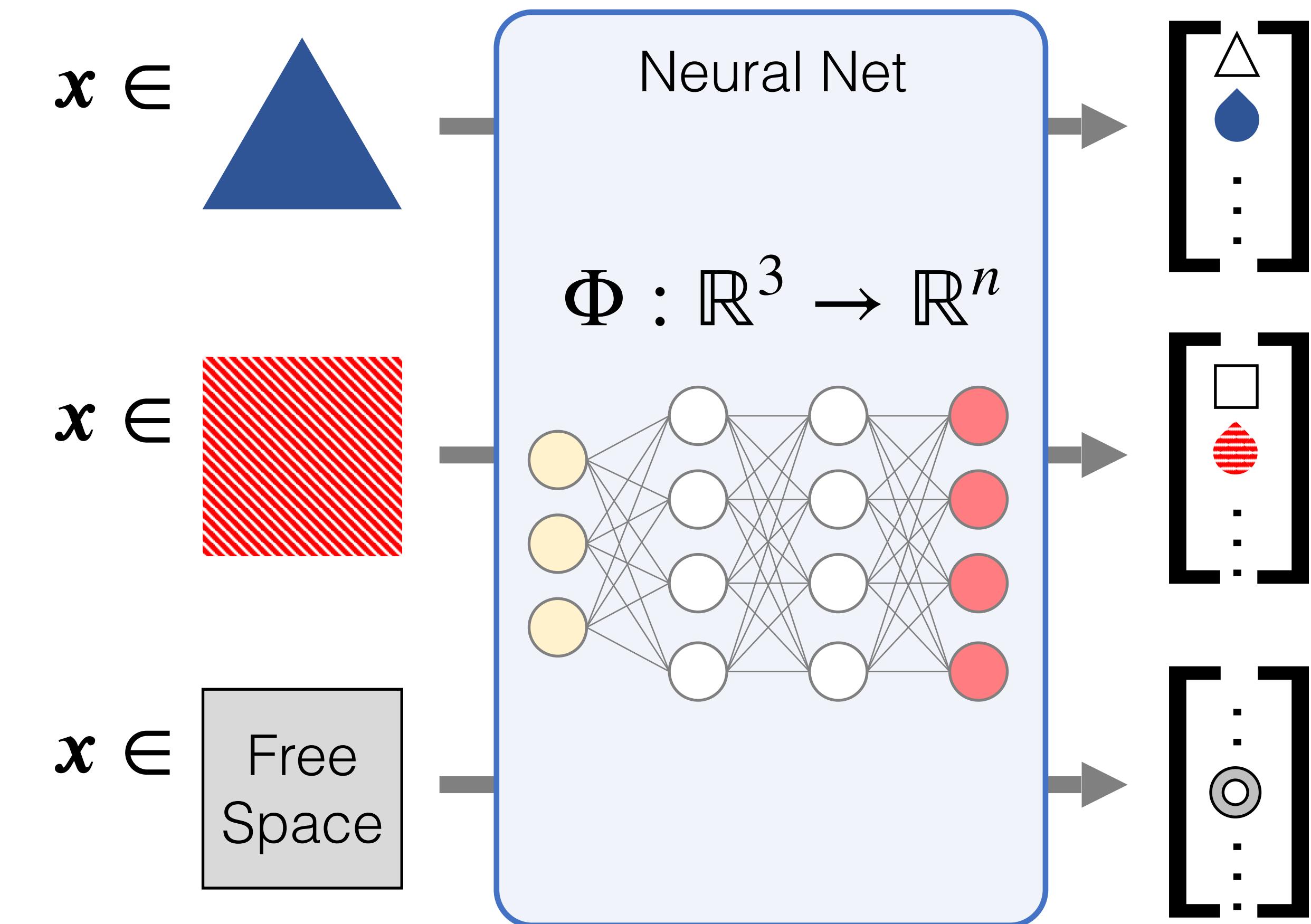
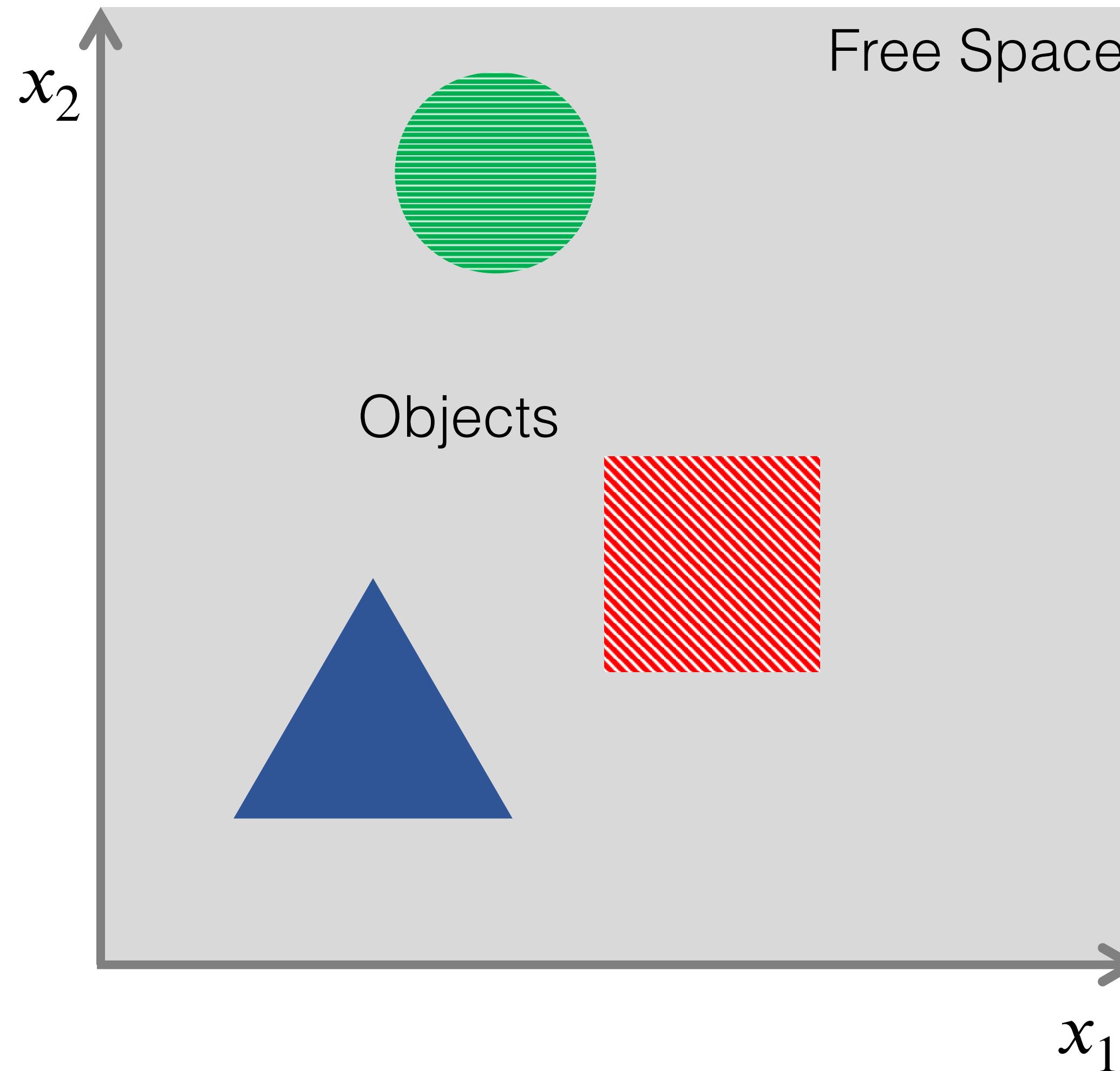


Neural Fields



Hybrid Parameterizations

Neural fields: Parameterize Field as Neural Network!



Occupancy Networks: Learning 3D Reconstruction in Function Space, Mescheder et al.

IM-Net: Learning Implicit Fields for Generative Shape Modeling, Chen et al.

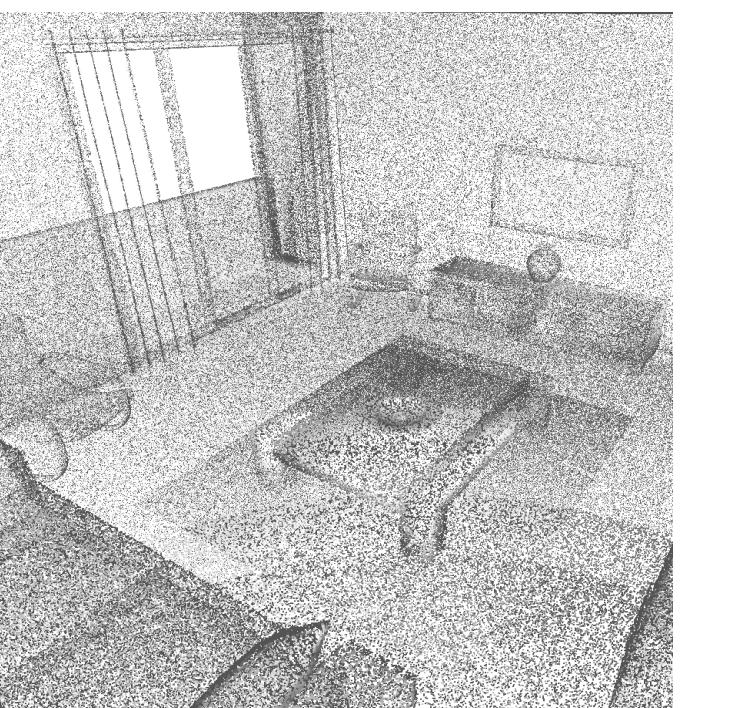
DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation, Park et al. 2019

Sitzmann et al: Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations, NeurIPS 2019.

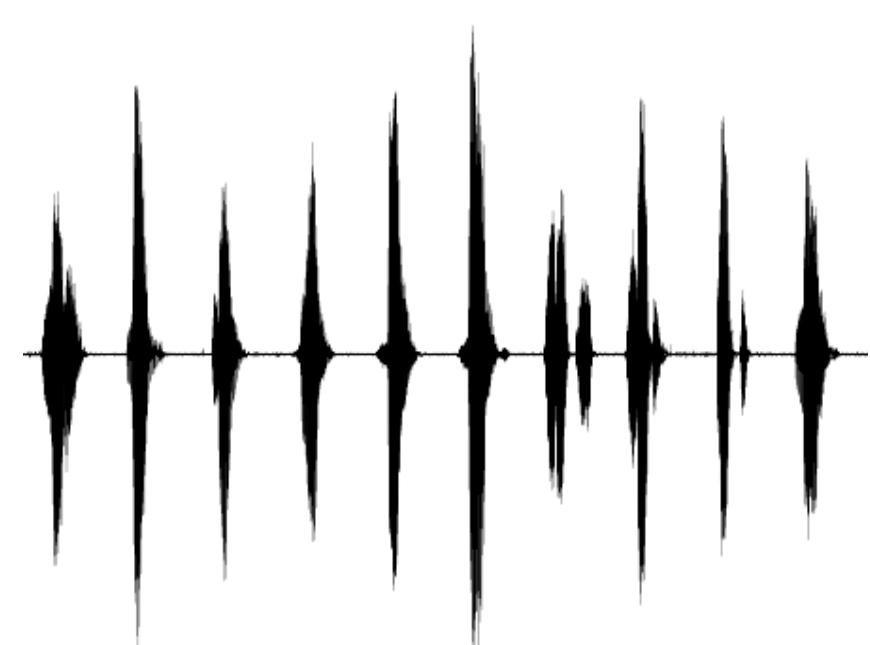
Images



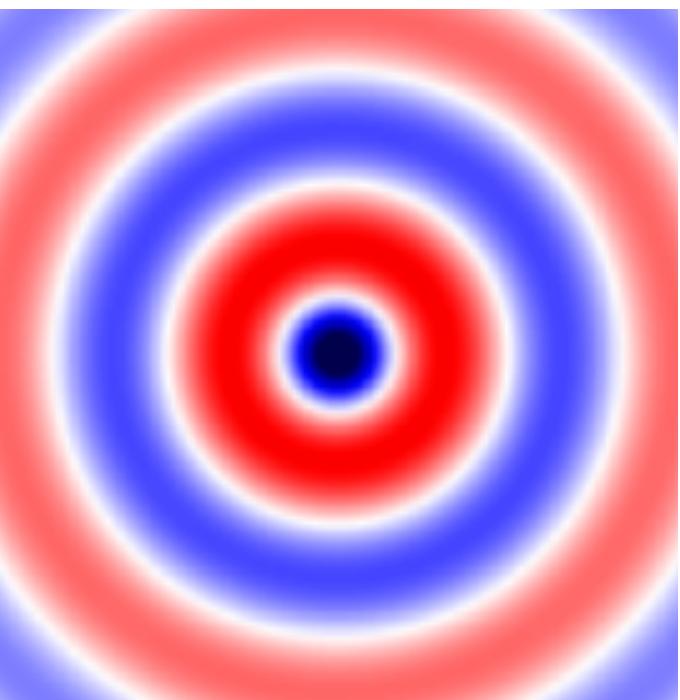
Shapes



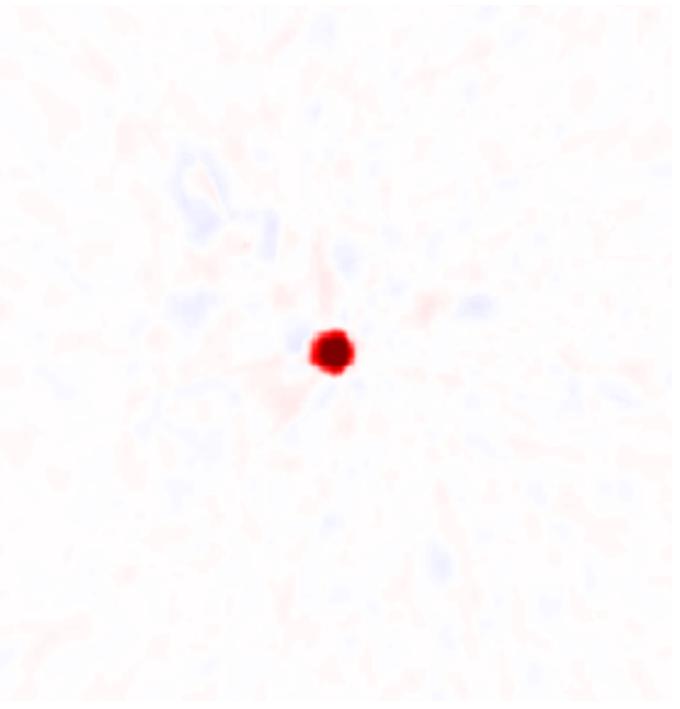
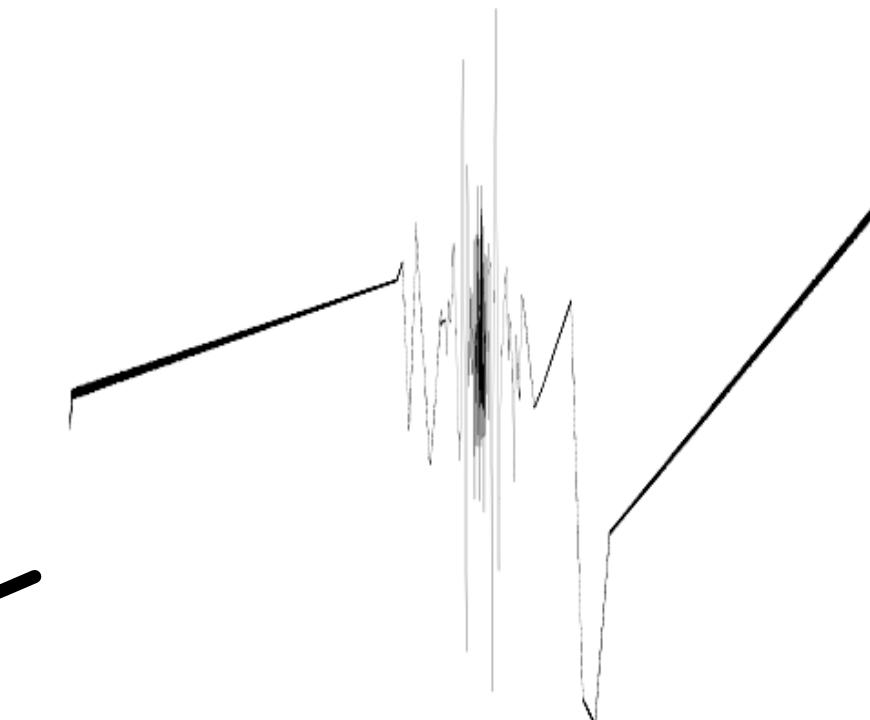
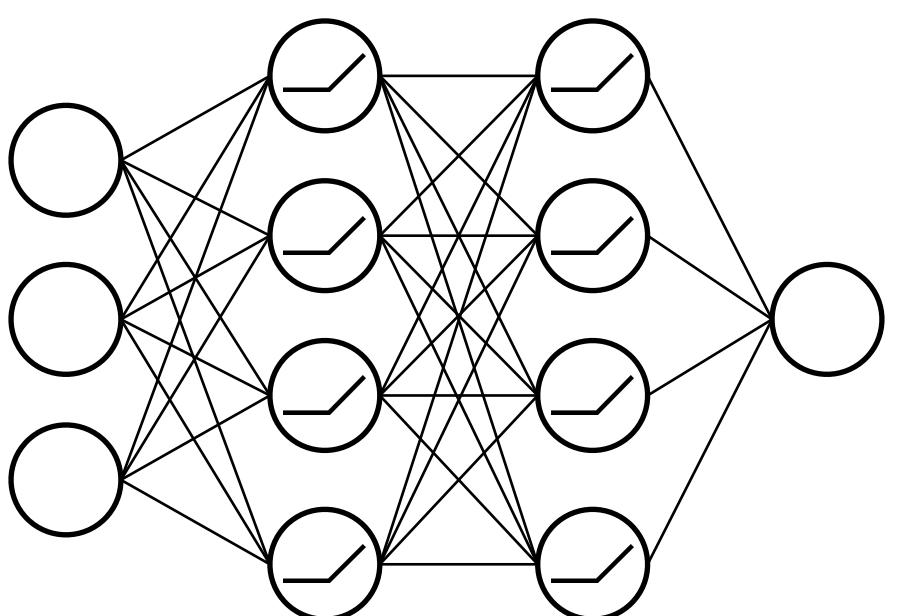
Audio



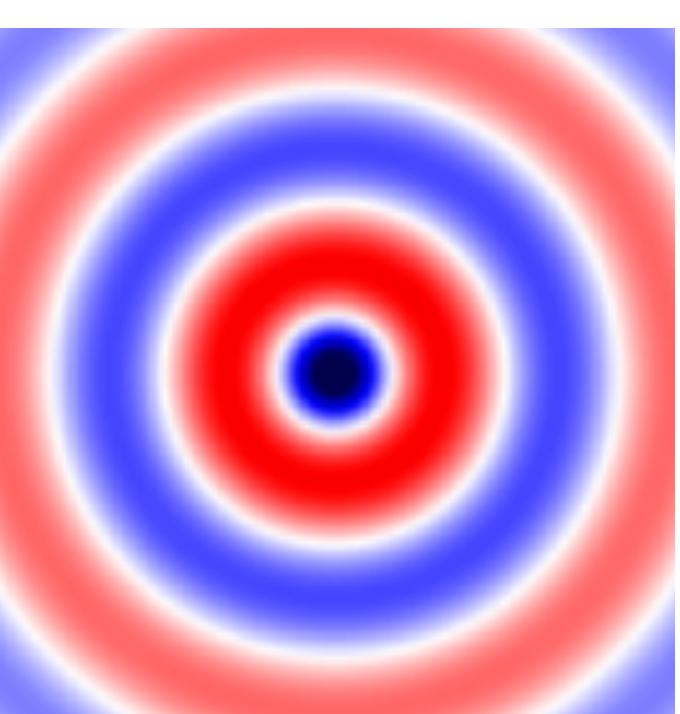
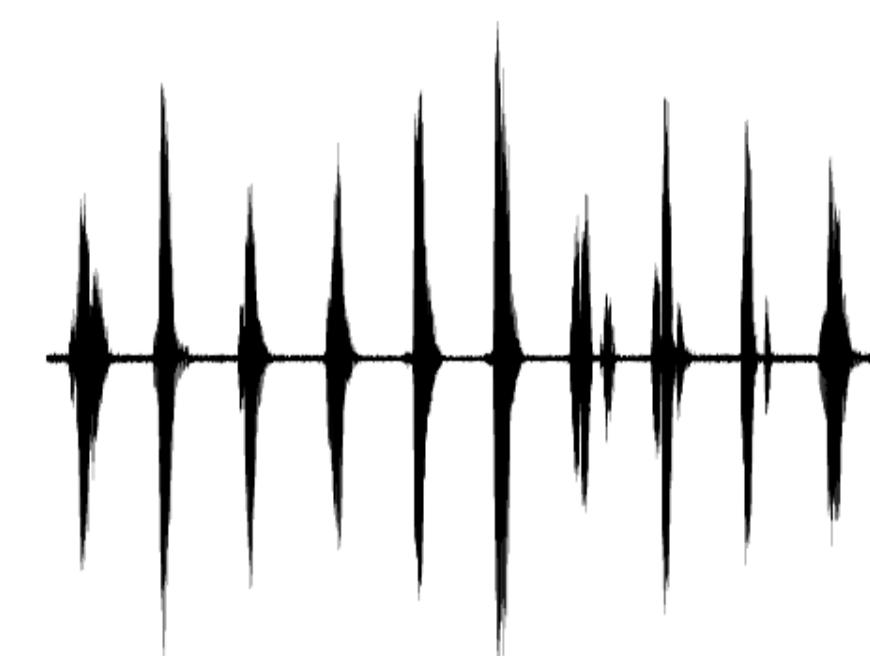
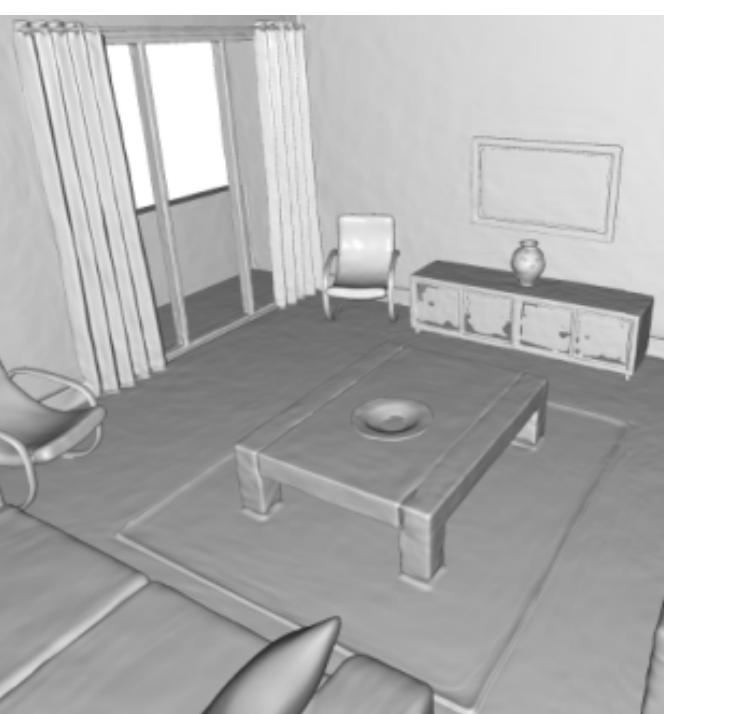
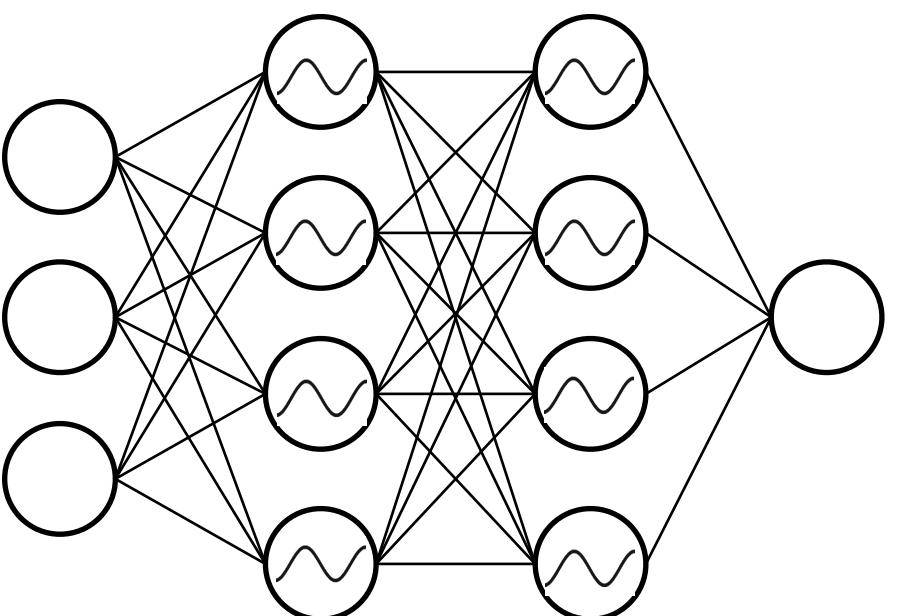
Quantities defined by a differential equation

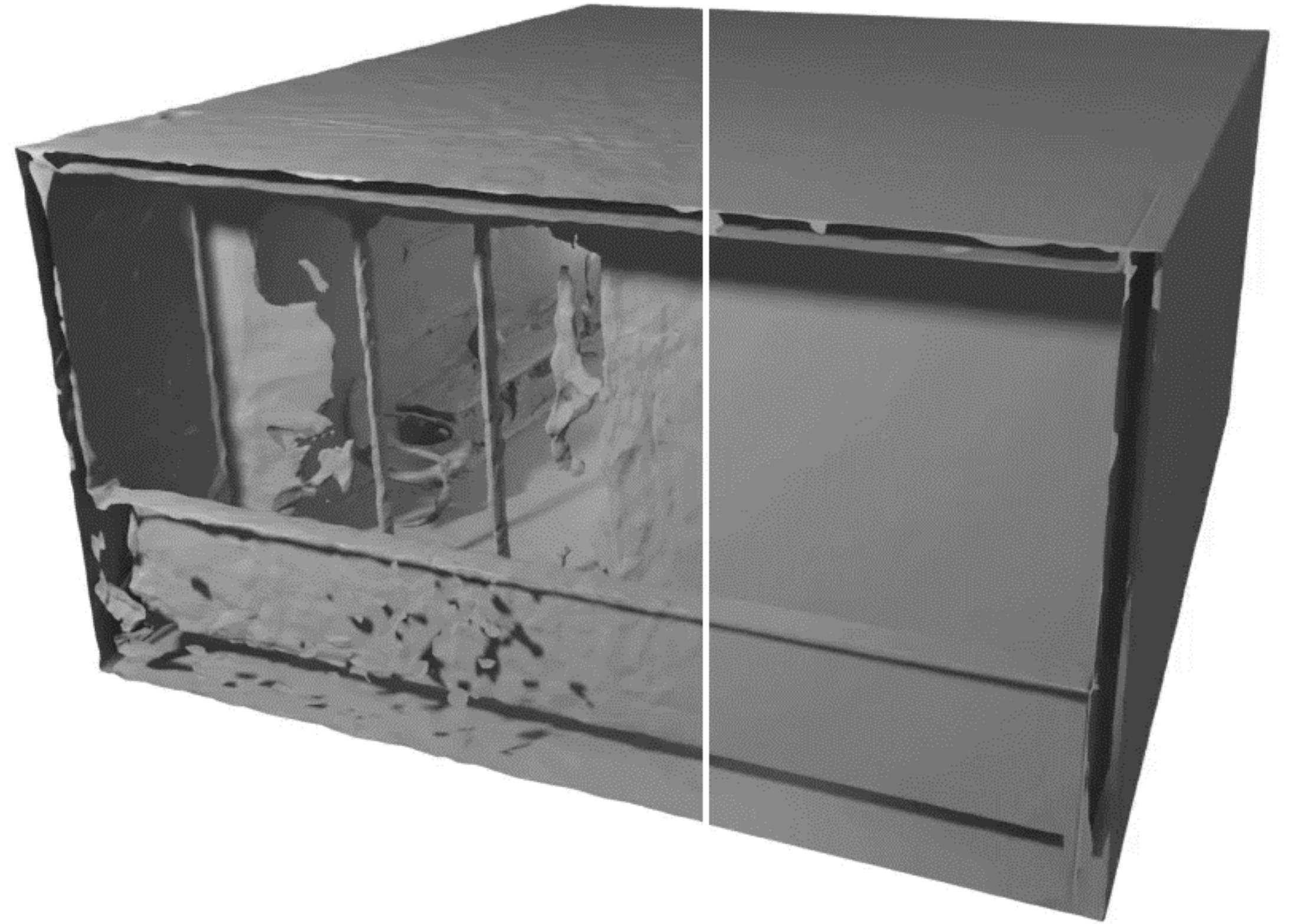


ReLU MLP



SIREN





~1MB compared to
110MB of full mesh!

Background: Kernel Regression

Given “training” set $\{(x_i, y_i)\}_i$, a kernel function makes predictions on a point \mathbf{x} by interpolating labels y_i in the training set according to pairwise weights between x_i to \mathbf{x} as measured by a kernel function:

$$f(\mathbf{x}) = \sum_{i=1}^n (\mathbf{K}^{-1} \mathbf{y})_i k(\mathbf{x}_i, \mathbf{x})$$

Where $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, k is kernel function with $k \geq 0$.

The Neural Tangent Kernel

$$f(\mathbf{x}) = \sum_{i=1}^n (\mathbf{K}^{-1}\mathbf{y})_i k(\mathbf{x}_i, \mathbf{x})$$

It turns out (Jacot et al. 2018) that in the limit of an infinitely wide MLP $f(x)$ with initialization θ_0 , test-time predictions are made according to a kernel:

$$k_{NTK}(\mathbf{x}_1, \mathbf{x}_2) = \langle \nabla_{\theta} f_{\theta_0}(\mathbf{x}_1), \nabla_{\theta} f_{\theta_0}(\mathbf{x}_2) \rangle$$

The Neural Tangent Kernel

$$f(\mathbf{x}) = \sum_{i=1}^n (\mathbf{K}^{-1}\mathbf{y})_i k(\mathbf{x}_i, \mathbf{x})$$

It turns out (Jacot et al. 2018) that in the limit of an infinitely wide MLP $f(x)$ with initialization θ_0 , test-time predictions are made according to a kernel:

$$k_{NTK}(\mathbf{x}_1, \mathbf{x}_2) = \langle \nabla_{\theta} f_{\theta_0}(\mathbf{x}_1), \nabla_{\theta} f_{\theta_0}(\mathbf{x}_2) \rangle$$

In other words: “Neural” implicit representations interpolate the training set according to a relatively simple rule. No “intelligence”, no AI, no magic.

The Neural Tangent Kernel

$$f(\mathbf{x}) = \sum_{i=1}^n (\mathbf{K}^{-1}\mathbf{y})_i k(\mathbf{x}_i, \mathbf{x})$$

It turns out (Jacot et al. 2018) that in the limit of an infinitely wide MLP $f(x)$ with initialization θ_0 , test-time predictions are made according to a kernel:

$$k_{NTK}(\mathbf{x}_1, \mathbf{x}_2) = \langle \nabla_{\theta} f_{\theta_0}(\mathbf{x}_1), \nabla_{\theta} f_{\theta_0}(\mathbf{x}_2) \rangle$$

What we can do is design the kernel and pick the space of (\mathbf{x}, \mathbf{y}) , the space in which our neural network will interpolate.

The Neural Tangent Kernel

$$f(\mathbf{x}) = \sum_{i=1}^n (\mathbf{K}^{-1}\mathbf{y})_i k(\mathbf{x}_i, \mathbf{x})$$

It turns out (Jacot et al. 2018) that in the limit of an infinitely wide MLP $f(x)$ with initialization θ_0 , test-time predictions are made according to a kernel:

$$k_{NTK}(\mathbf{x}_1, \mathbf{x}_2) = \langle \nabla_{\theta} f_{\theta_0}(\mathbf{x}_1), \nabla_{\theta} f_{\theta_0}(\mathbf{x}_2) \rangle$$

We do not know of an equivalent rule for transformers to date - work in progress!

The Neural Tangent Kernel

$$f(\mathbf{x}) = \sum_{i=1}^n (\mathbf{K}^{-1}\mathbf{y})_i k(\mathbf{x}_i, \mathbf{x})$$

It turns out (Jacot et al. 2018) that in the limit of an infinitely wide MLP $f(x)$ with initialization θ_0 , test-time predictions are made according to a kernel:

$$k_{NTK}(\mathbf{x}_1, \mathbf{x}_2) = \langle \nabla_{\theta} f_{\theta_0}(\mathbf{x}_1), \nabla_{\theta} f_{\theta_0}(\mathbf{x}_2) \rangle$$

What does this mean for neural fields?

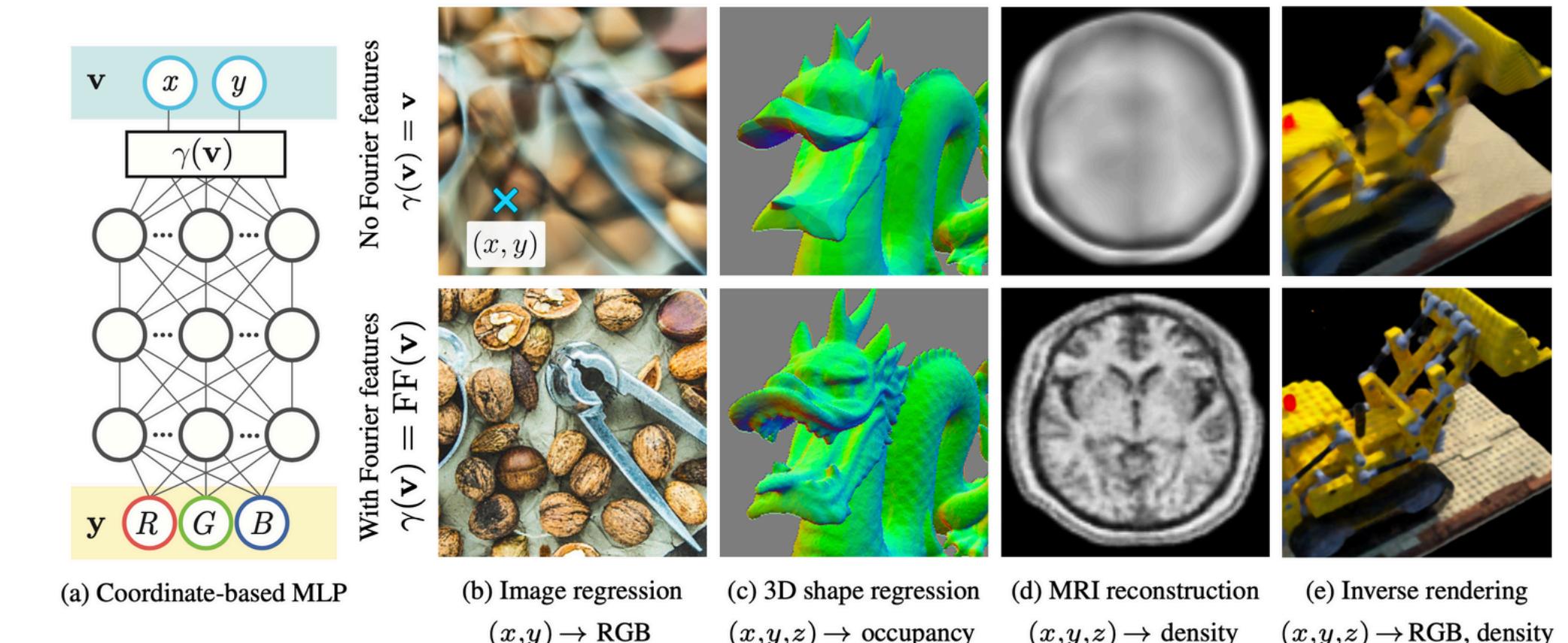
SIREN: Neural Implicit Representations With Periodic Activation Functions

Sitzmann & Martel et al. NeurIPS 2020



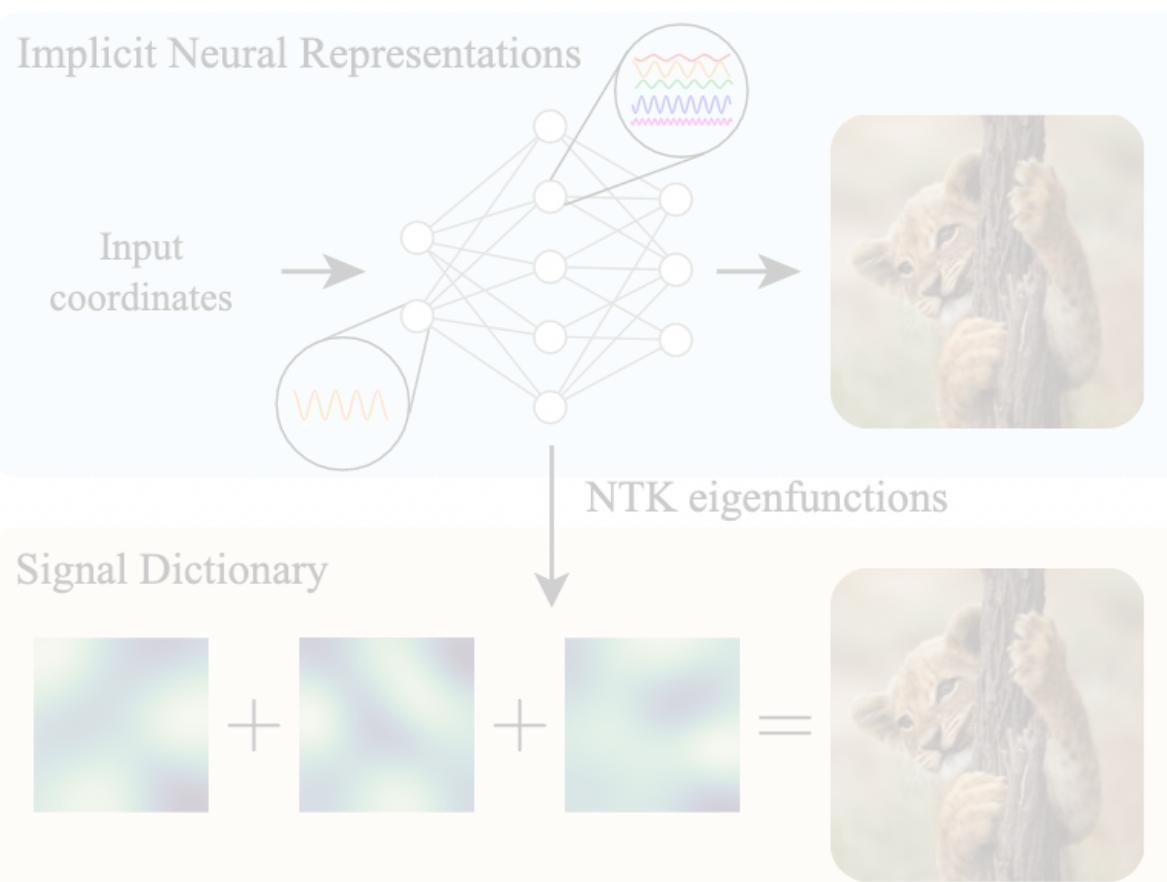
Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains

Tancik, Srinivasan, Mildenhall et al, NeurIPS 2020



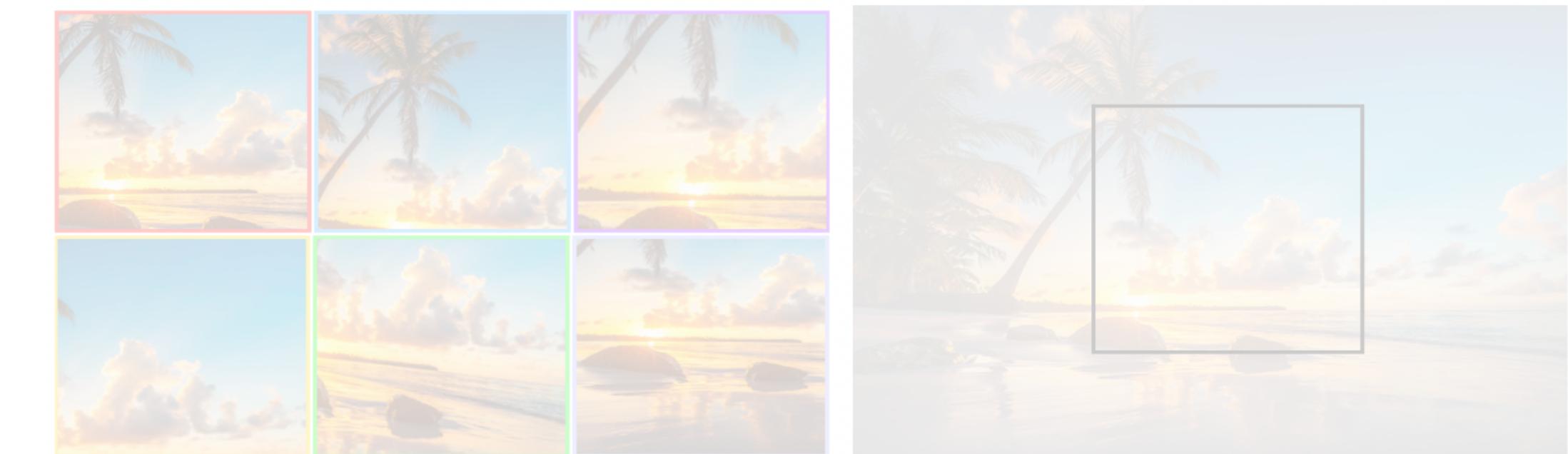
A Structured Dictionary Perspective on Implicit Neural Representations

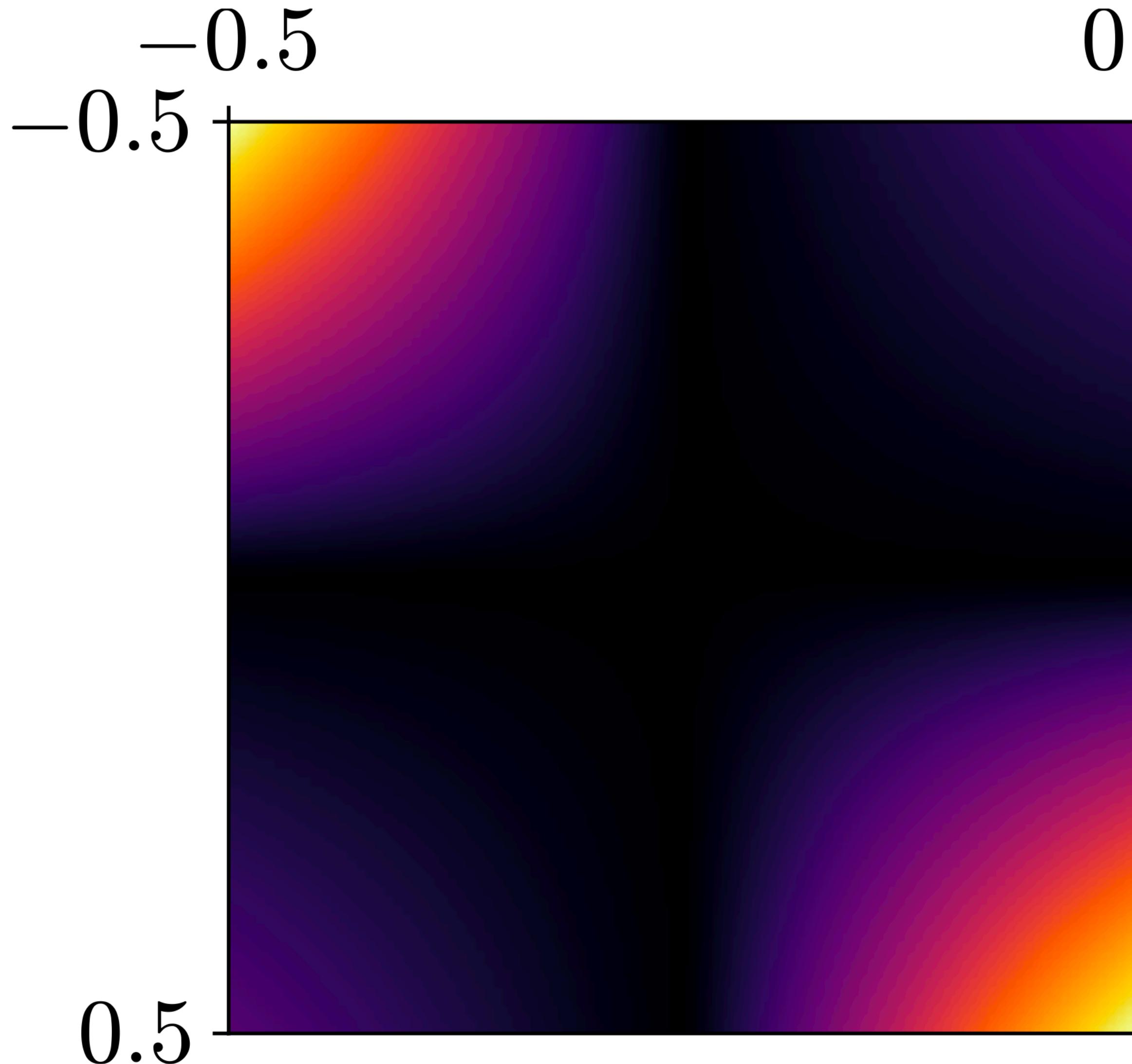
Yüce & Ortiz-Jiménez et al. CVPR 2022



Gaussian Activated Neural Radiance Fields for High Fidelity Reconstruction & Pose Estimation

Chng et al. ECCV 2022





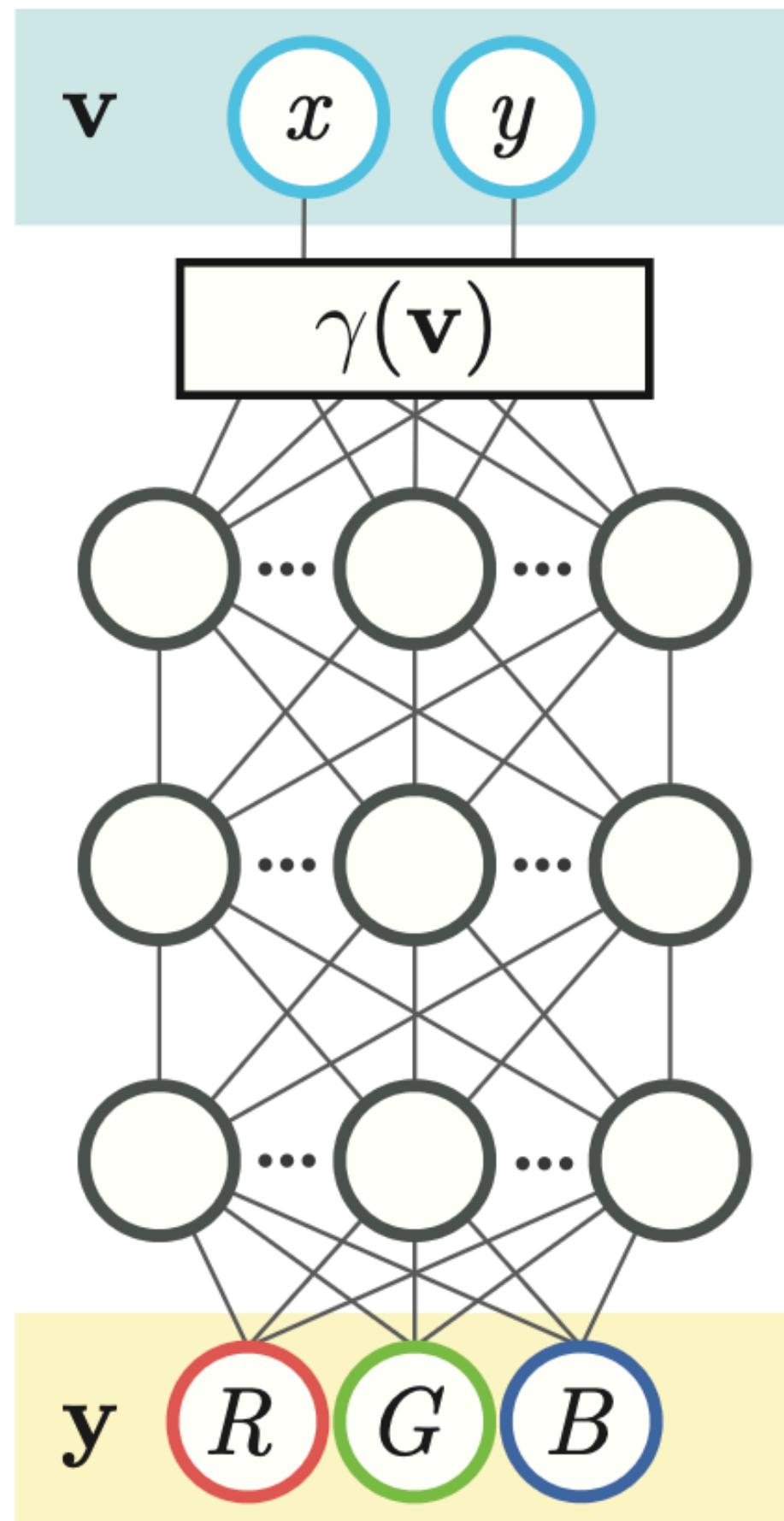
what we realise is relu doesn't
use local property

NTK kernel $k_{NTK}(x_i, x_j)$ for a 4-layer
ReLU MLP with one scalar input.

From “Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains”, Tancik, Srinivasan, Mildenhall et al.

NTK of ReLU neural net is quite non-local.

Feature Embeddings



$$\gamma(\mathbf{x}) = [\gamma_1(\mathbf{x}), \dots, \gamma_n(\mathbf{x})]$$

$$\gamma_{2i}(\mathbf{x}) = \sin(2^{i-1}\pi x_i)$$

$$\gamma_{(2i+1)}(\mathbf{x}) = \cos(2^{i-1}\pi x_i)$$

Sinusoidal Embeddings

Zhong et al. ICLR 2020

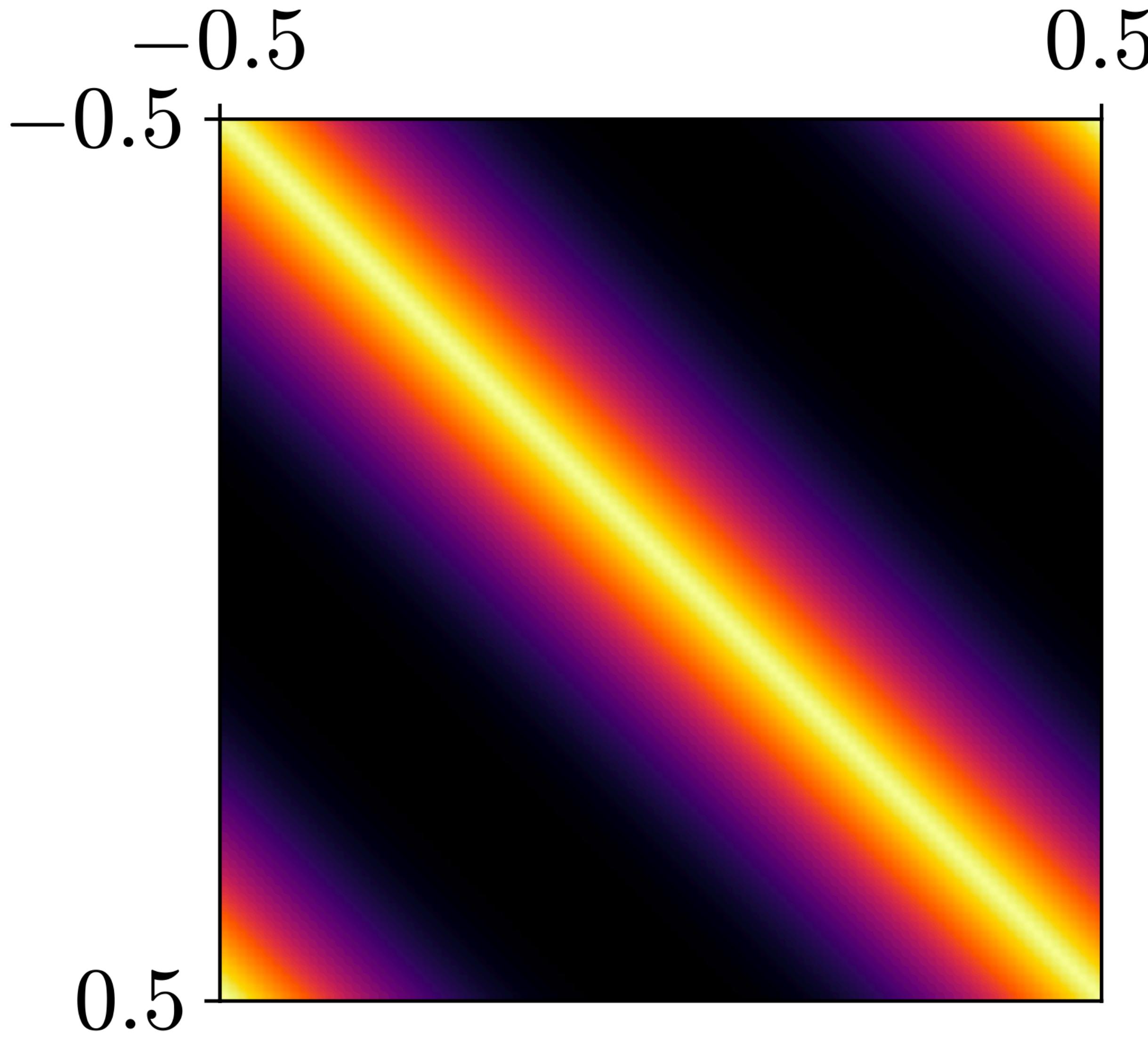
Mildenhall et al., ECCV 2020

$$\gamma(\mathbf{x}) = \exp\left(-\frac{\|t - x\|^2}{2\sigma^2}\right)$$

Gaussian Embeddings

Zheng et al., arXiv 2021

• • •



NTK kernel $k_{NTK}(\gamma(x_i), \gamma(x_j))$ for a 4-layer ReLU MLP with one scalar input and **Sinusoidal Feature Mapping**

$$\gamma(x) = [\cos(2\pi x), \sin(2\pi x)].^\top$$

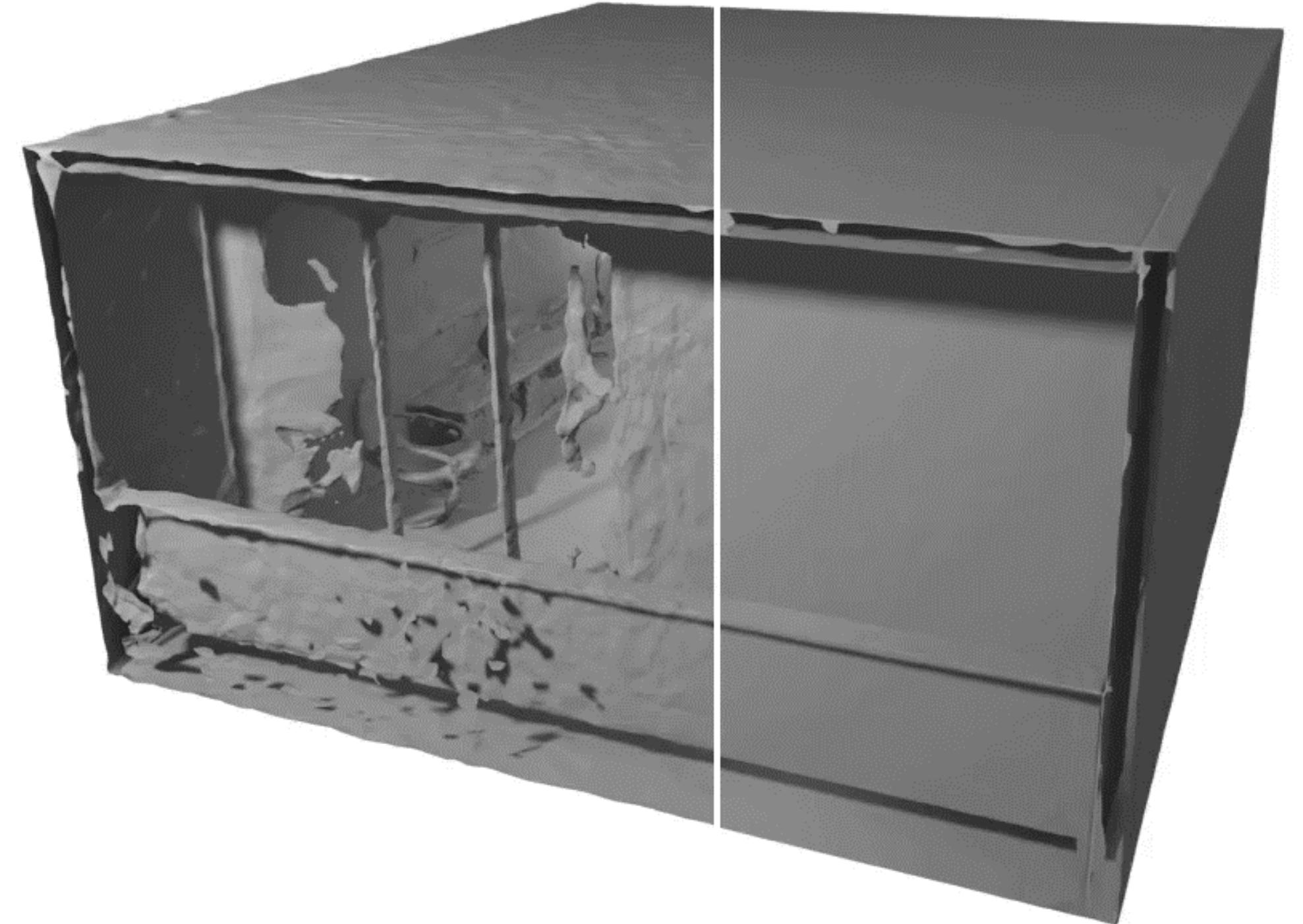
local kernel.

From “Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains”, Tancik, Srinivasan, Mildenhall et al.

We can tune the kernel function like in this case.

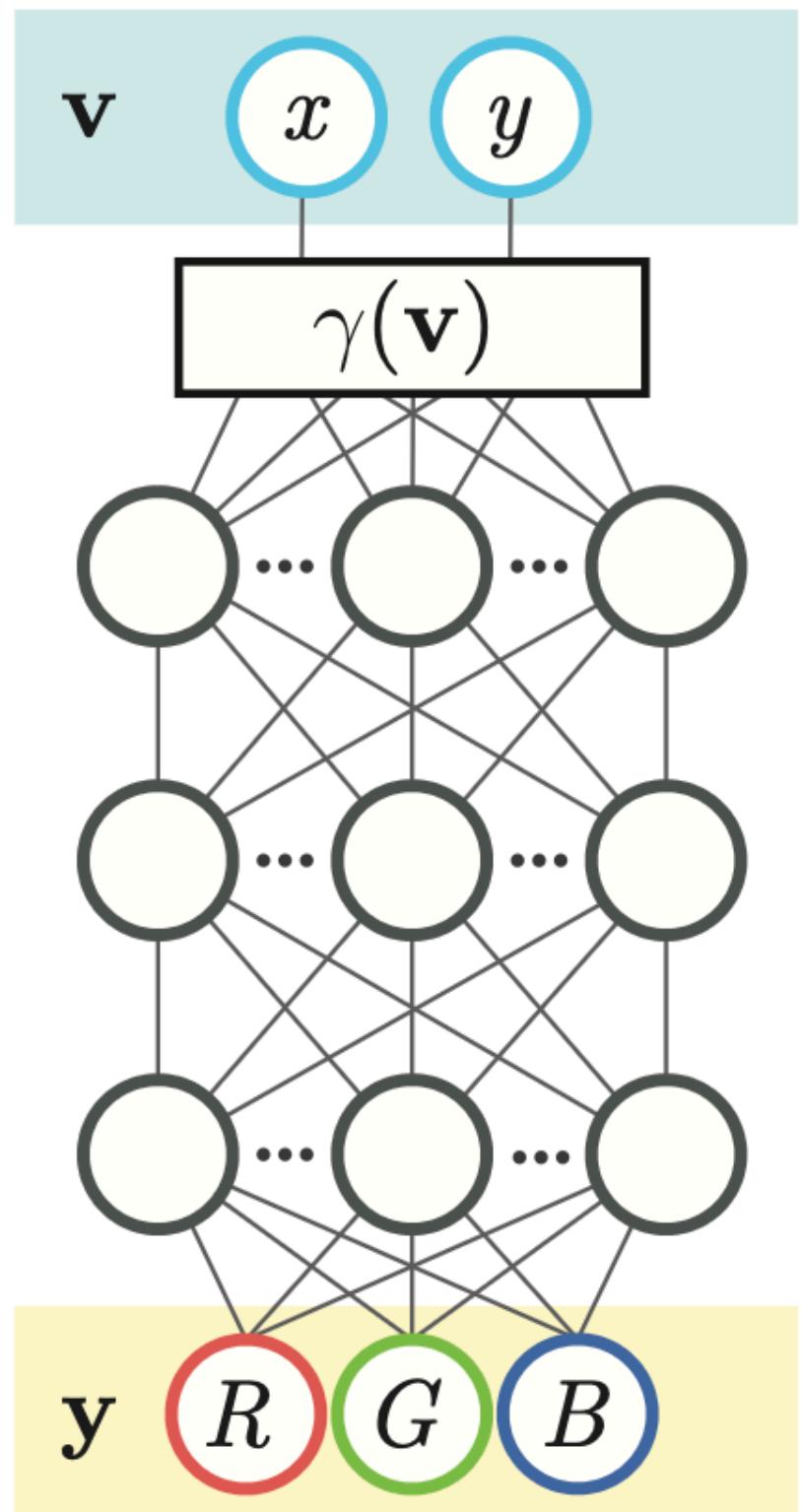
NTK is local (note diagonal).

SIREN



~1MB compared to
110MB of full mesh!

Fourier Features



(a) Coordinate-based MLP

No Fourier features
 $\gamma(\mathbf{v}) = \mathbf{v}$

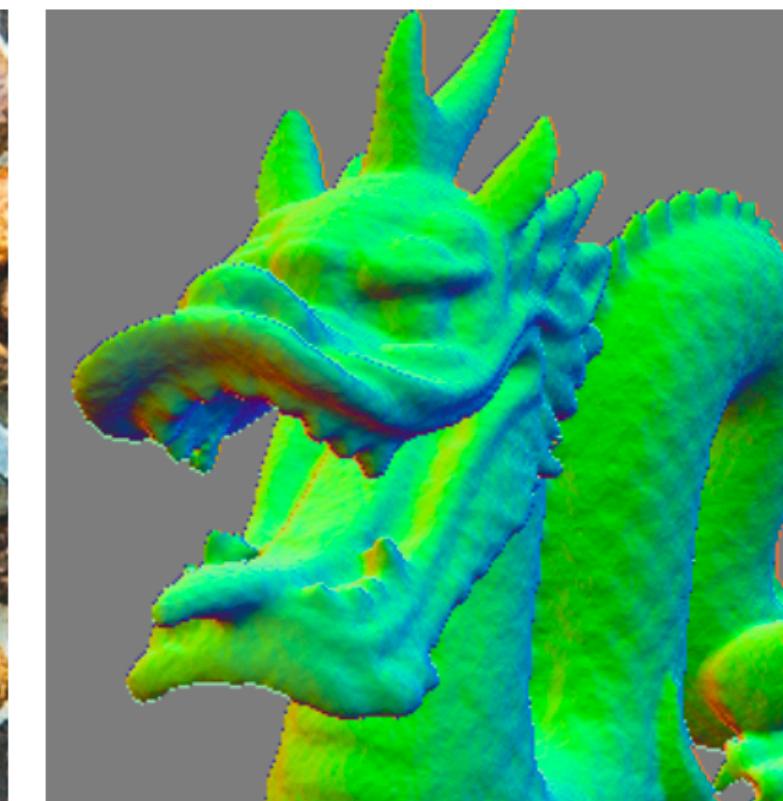
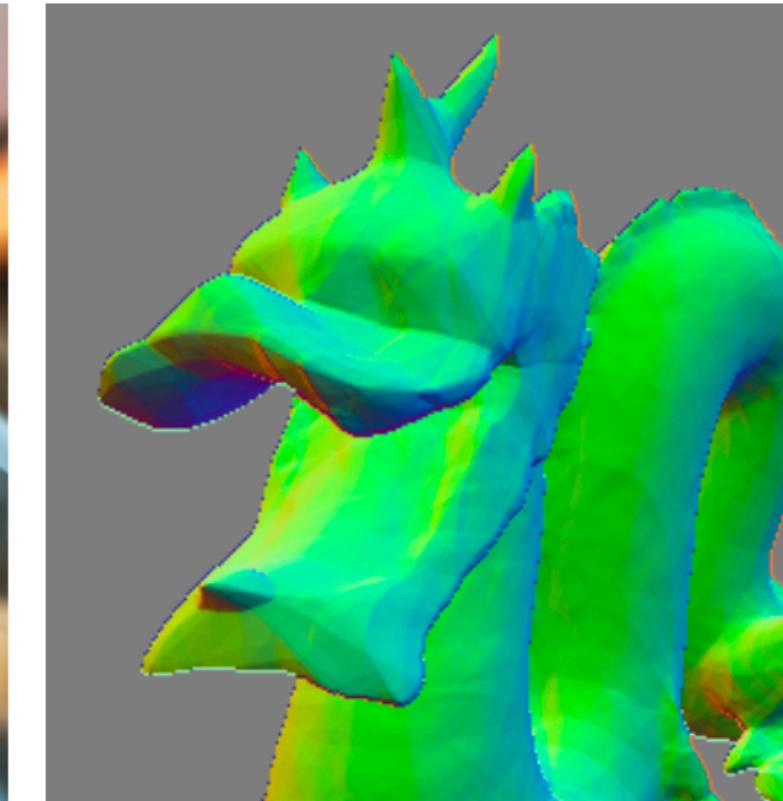


With Fourier features
 $\gamma(\mathbf{v}) = \text{FF}(\mathbf{v})$



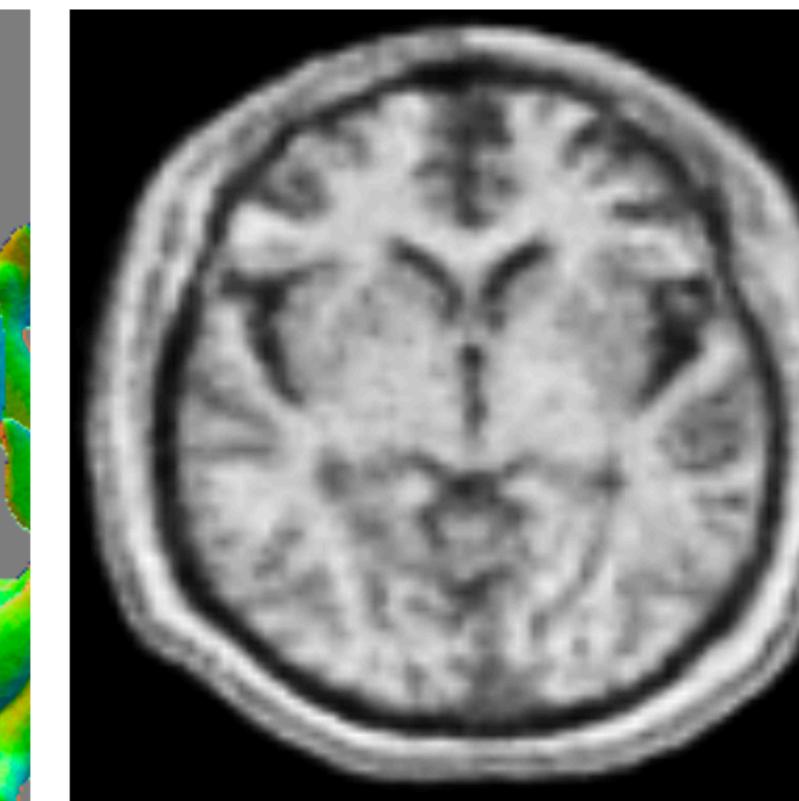
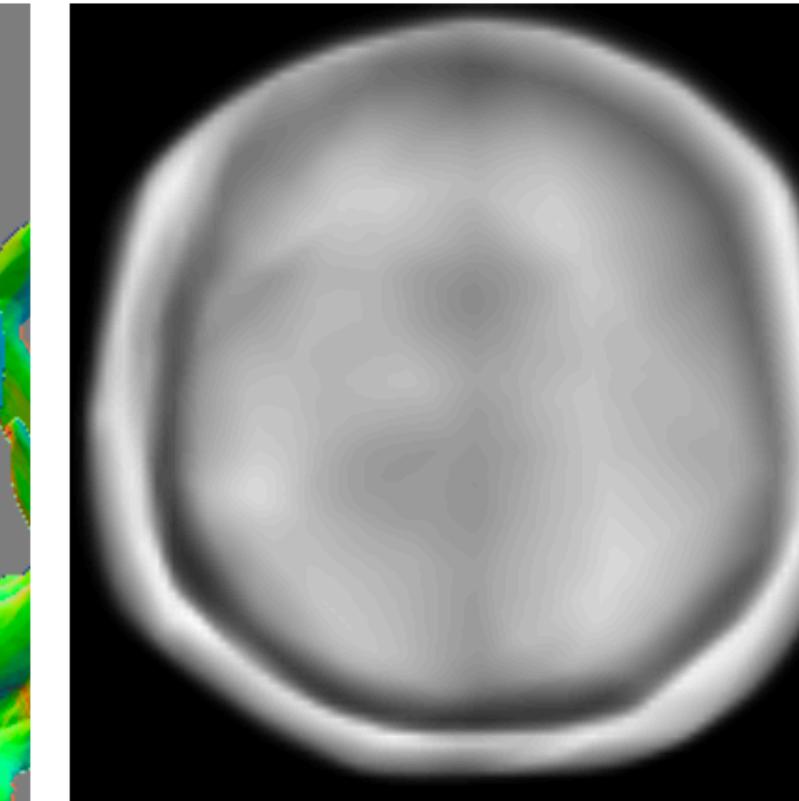
(b) Image regression

$$(x, y) \rightarrow \text{RGB}$$



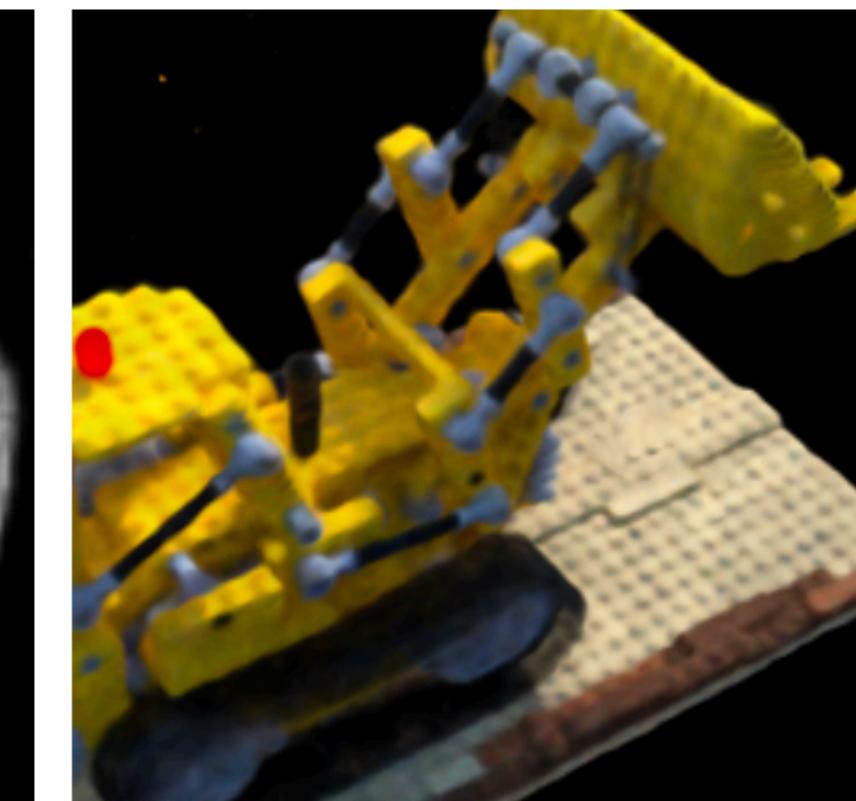
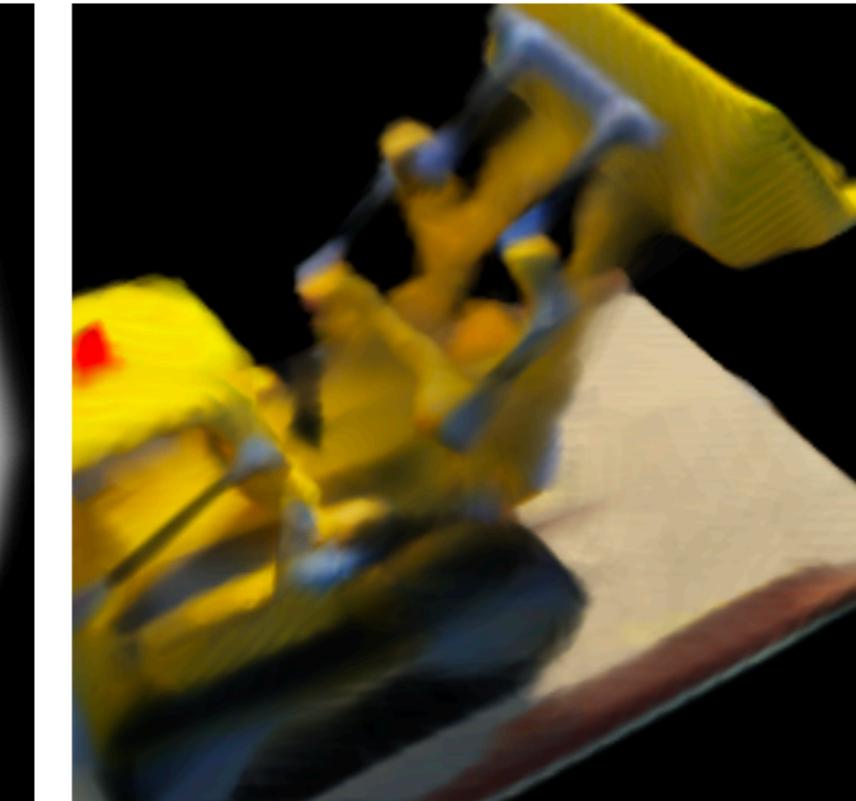
(c) 3D shape regression

$$(x, y, z) \rightarrow \text{occupancy}$$



(d) MRI reconstruction

$$(x, y, z) \rightarrow \text{density}$$



(e) Inverse rendering

$$(x, y, z) \rightarrow \text{RGB, density}$$

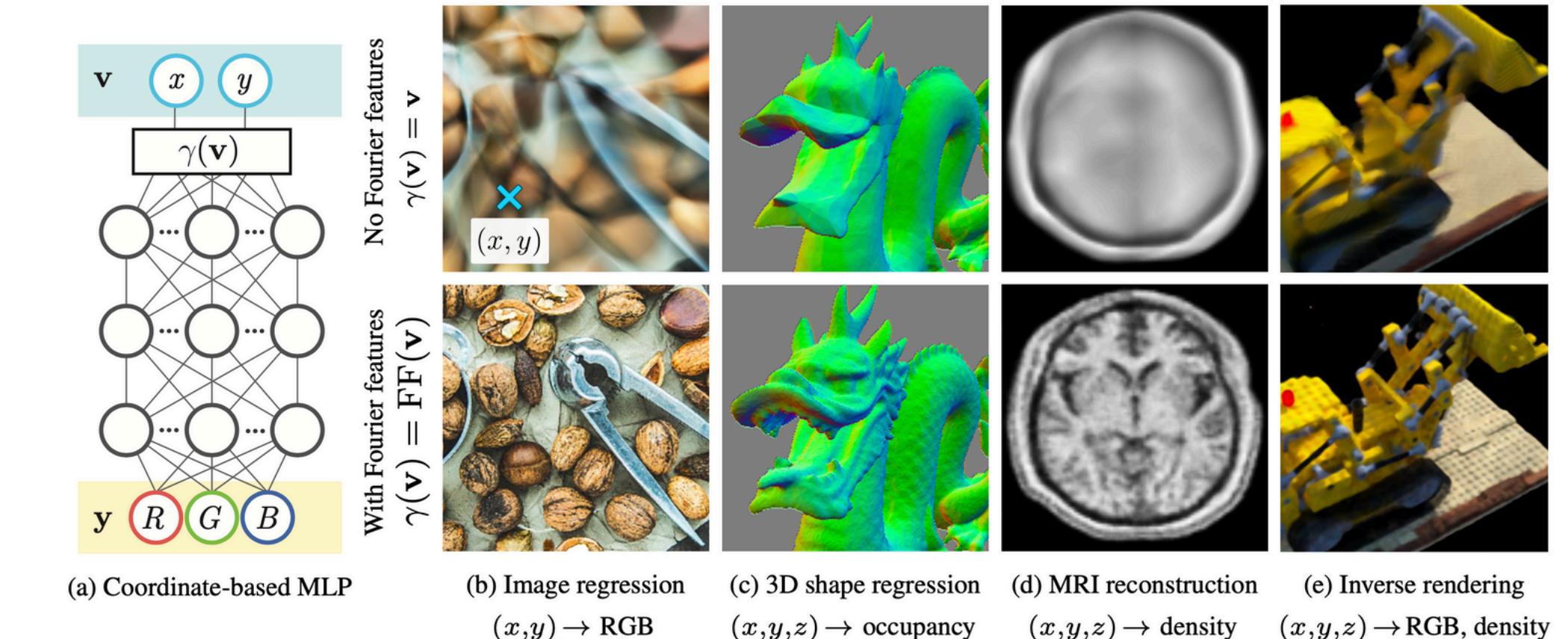
SIREN: Neural Implicit Representations With Periodic Activation Functions

Sitzmann & Martel et al. NeurIPS 2020



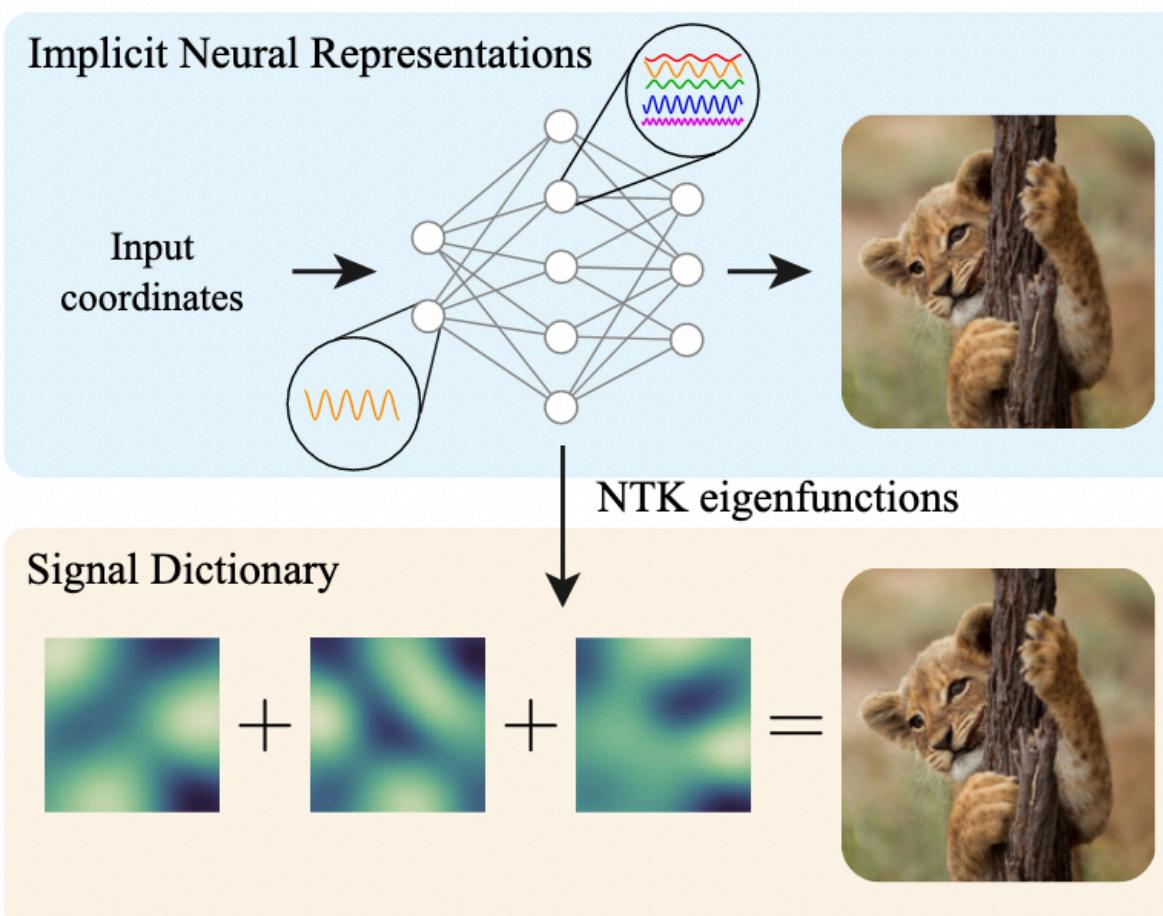
Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains

Tancik, Srinivasan, Mildenhall NeurIPS 2020



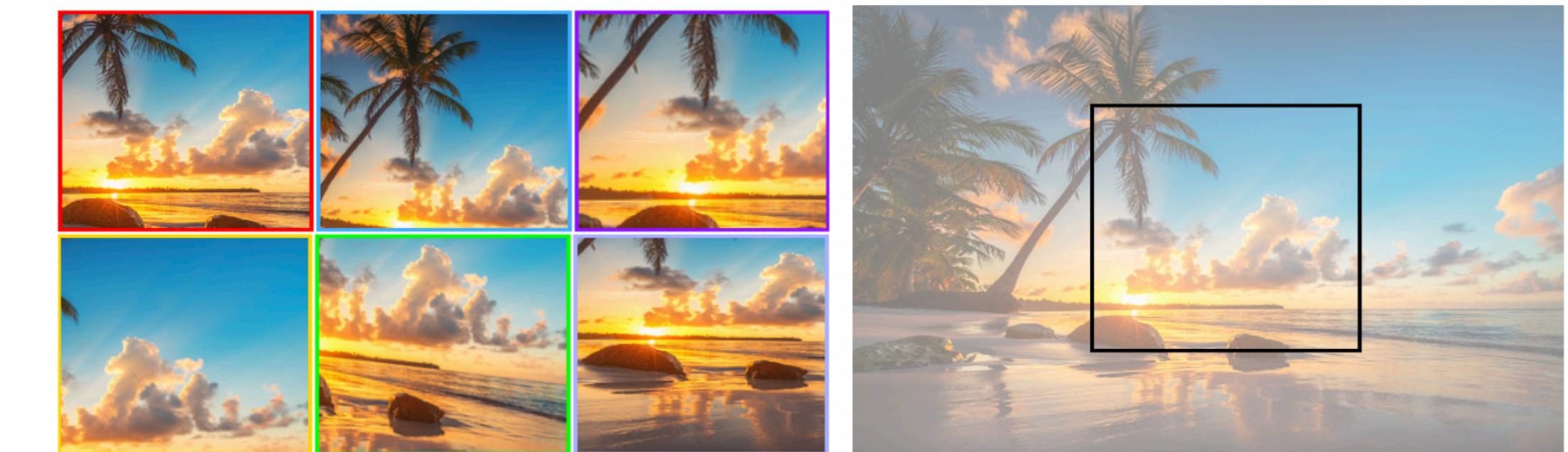
A Structured Dictionary Perspective on Implicit Neural Representations

Yüce & Ortiz-Jiménez et al. CVPR 2022



Gaussian Activated Neural Radiance Fields for High Fidelity Reconstruction & Pose Estimation

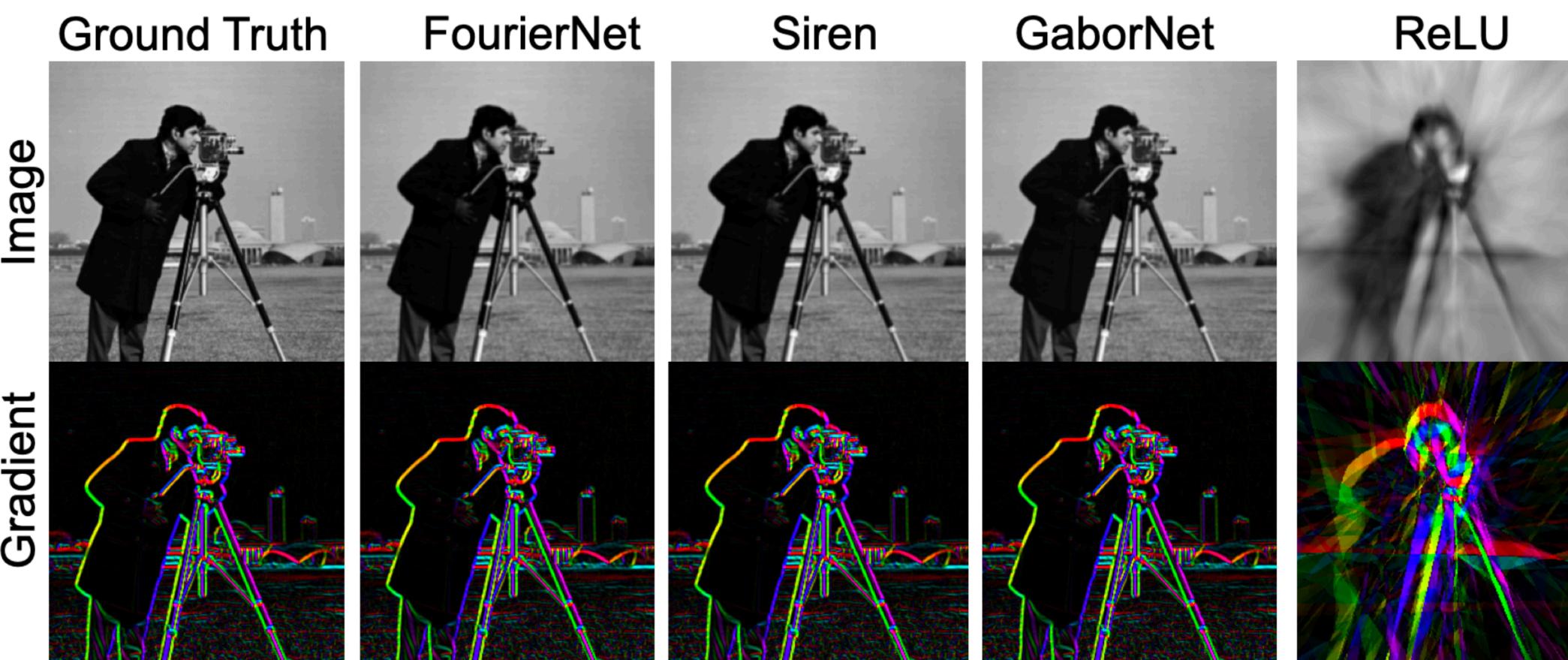
Chng et al. ECCV 2022



Neural (?) Fields with analytical (but not tractably computable) Fourier spectrum!

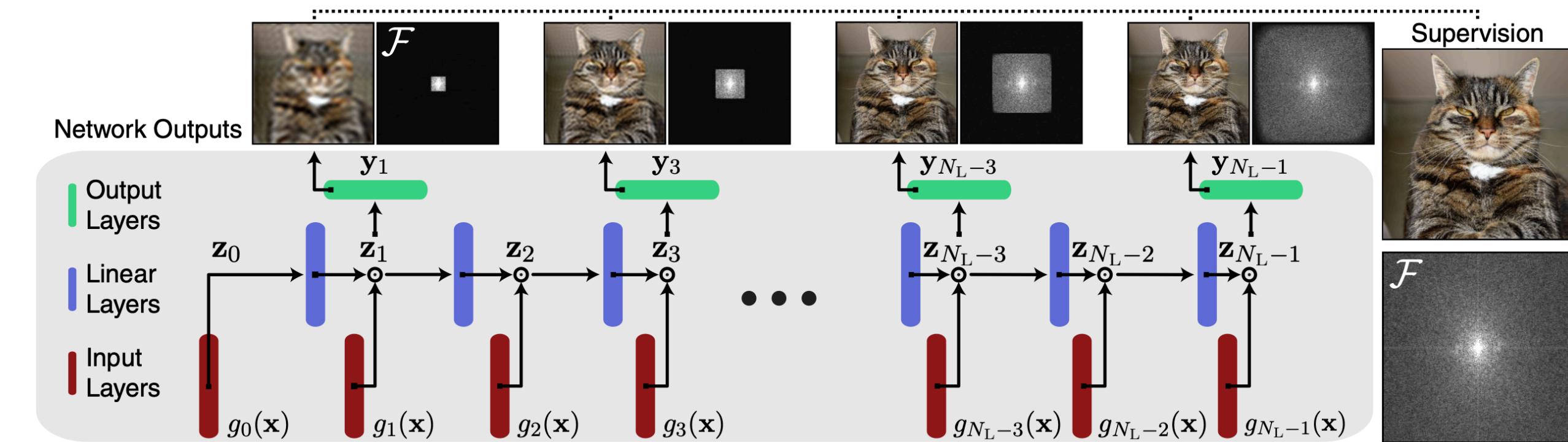
Multiplicative Filter Networks

Fathony et al. ICLR 2021



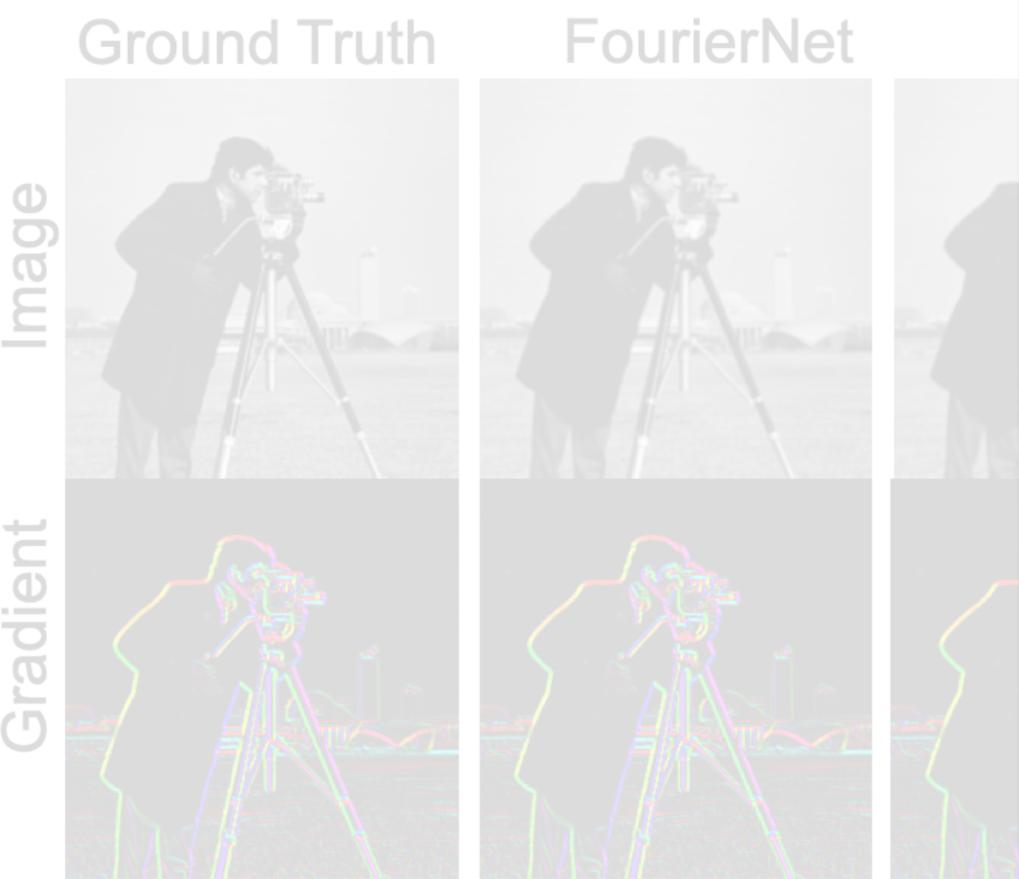
BACON: Band-limited coordinate networks for multiscale scene representation

Lindell et al. CVPR 2022



(but

Multiplicative Fathy et al.



arXiv:2111.11426v4 [cs.CV] 5 Apr 2022

EUROGRAPHICS 2022
D. Meneveaux and G. Patanè
(Guest Editors)

Volume 41 (2022), Number 2
STAR – State of The Art Report

rum!

Neural Fields in Visual Computing and Beyond

Yiheng Xie^{1,2} Towaki Takikawa^{3,4} Shunsuke Saito⁵ Or Litany⁴ Shiqin Yan¹ Numair Khan¹ Federico Tombari^{6,7}
James Tompkin¹ Vincent Sitzmann^{8†} Srinath Sridhar^{1†}

¹Brown University ²Unity Technologies ³University of Toronto ⁴NVIDIA ⁵Meta Reality Labs Research ⁶Google ⁷Technical University of Munich
⁸Massachusetts Institute of Technology [†]Equal advising

<https://neuralfields.cs.brown.edu/>

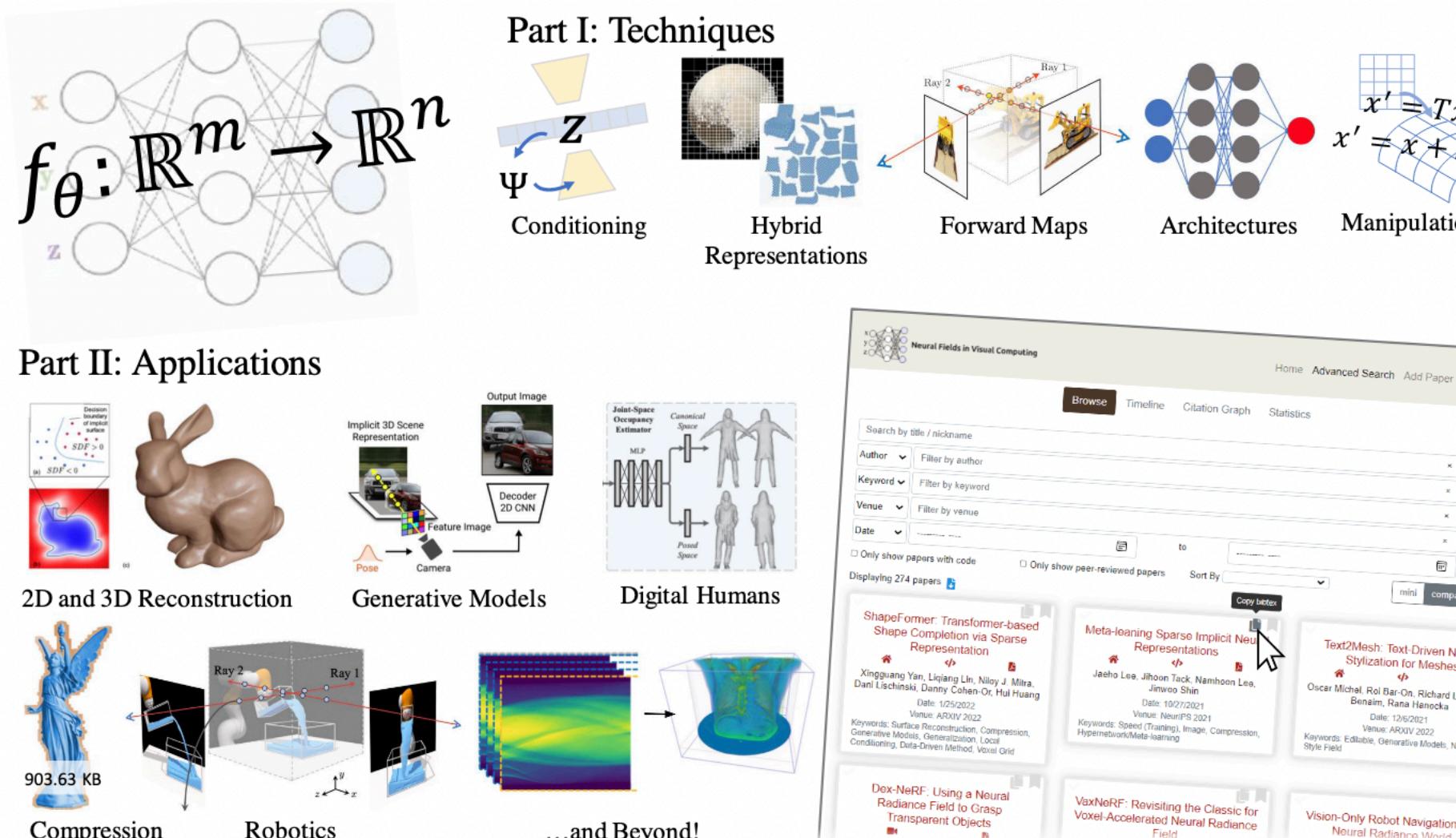
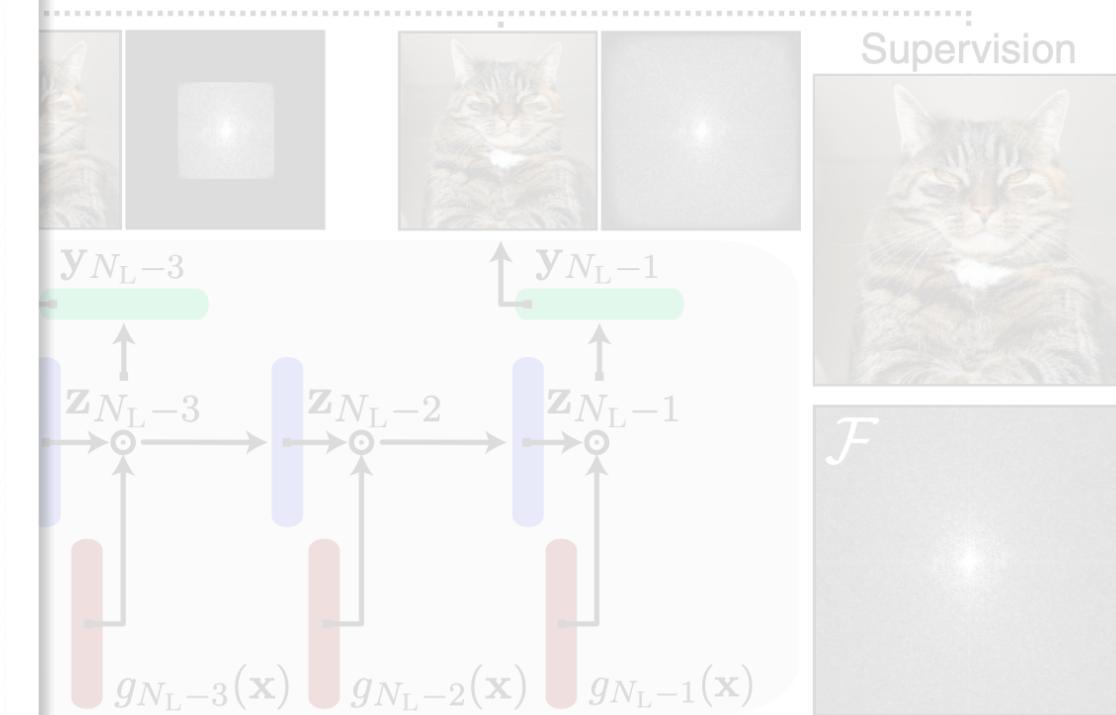


Figure 1: Contribution of this report. Following a survey of over 250 papers, we provide a review of (**Part I**) techniques in neural fields such as prior learning and conditioning, representations, forward maps, architectures, and manipulation, and of (**Part II**) applications in visual computing including 2D image processing, 3D scene reconstruction, generative modeling, digital humans, compression, robotics, and beyond. This report is complemented by a [community-driven website](https://neuralfields.cs.brown.edu/) with search, filtering, bibliographic, and visualization features.

ordinate networks for
representation
CVPR 2022

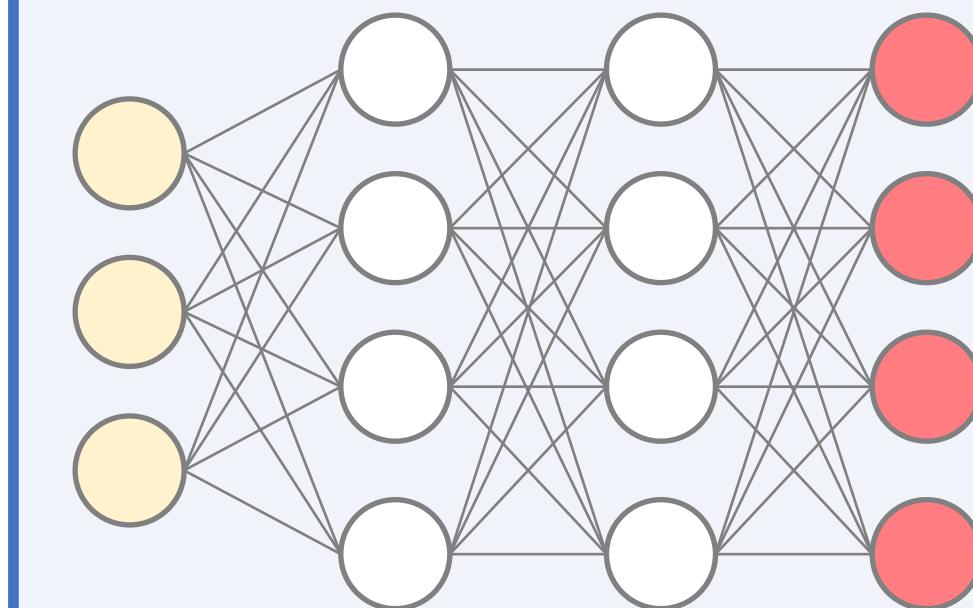


Neural Fields

- Storage memory **does not grow with spatial resolution or number of spatial dimensions.**
- **Slow** sampling: Each query takes a forward pass through the neural net. Means GPU-memory intensive forward passes.
- Does **not** expose locality: Can't identify set of parameters / direction in parameter space that encodes particular spatial location.
- **Inconvenient processing:** cannot run convolutions.
- But: automatic adaptive resolution. Over optimization, will converge to assign more compute to higher-frequency areas.

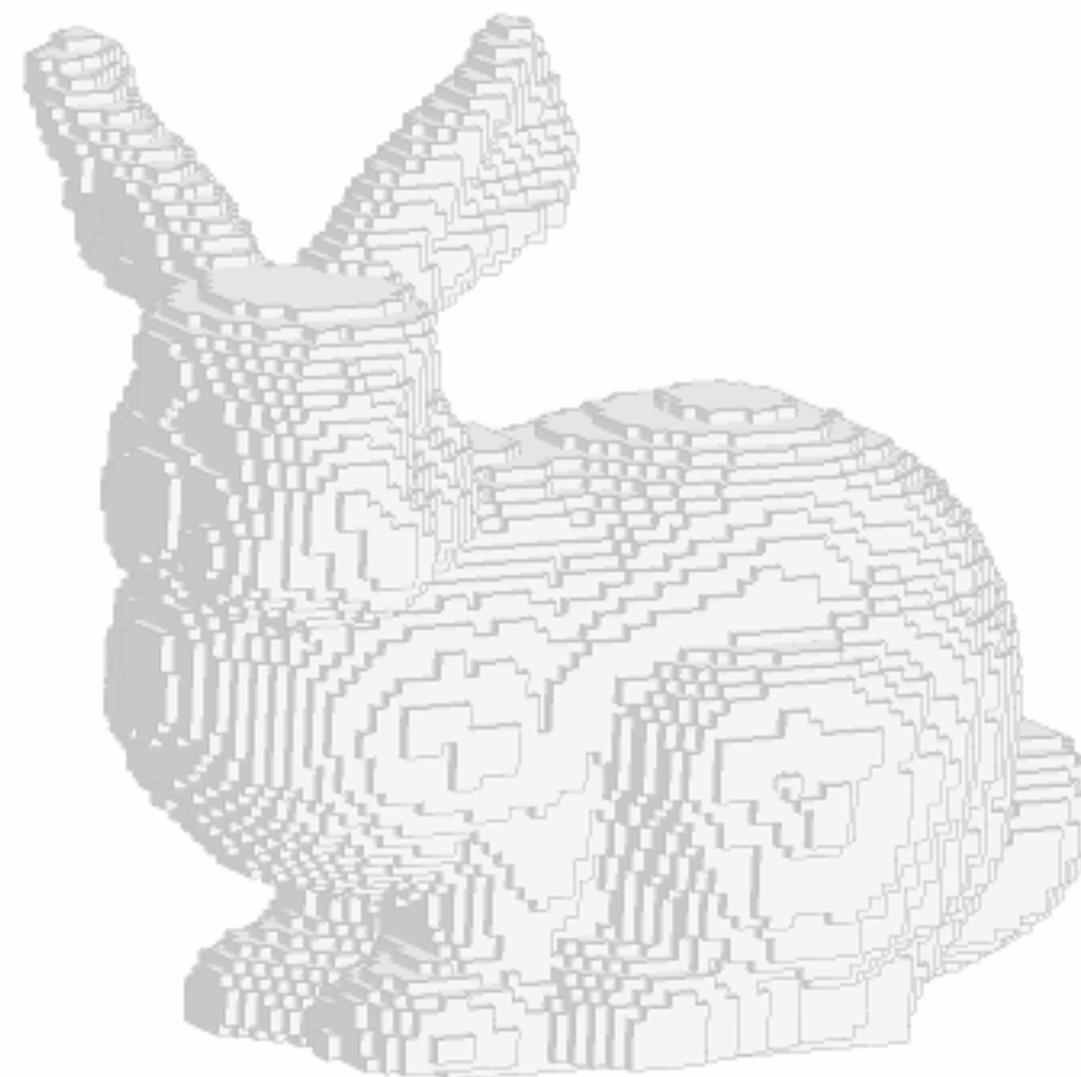
Neural Fields

$$\Phi : \mathbb{R}^3 \rightarrow \mathbb{R}^n$$

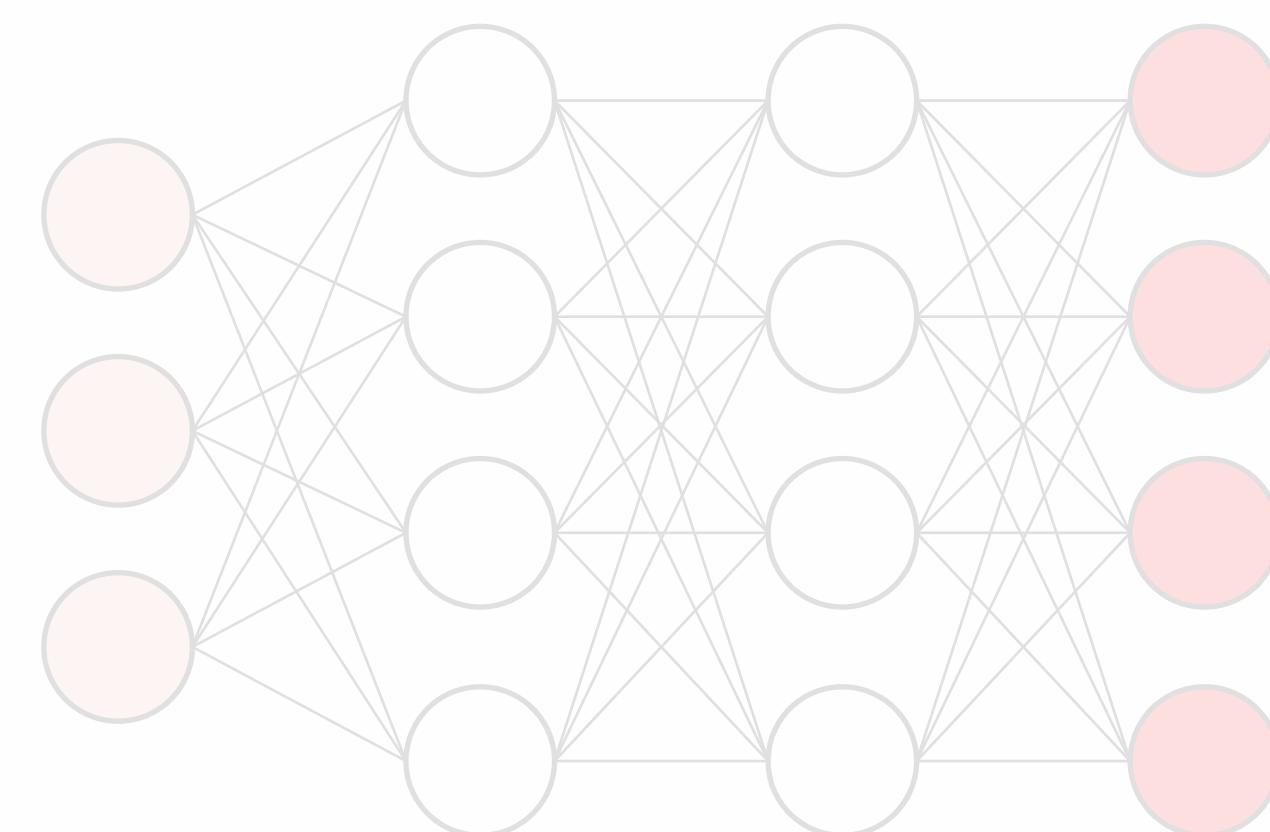


Note: Despite their name, has nothing to do with Machine Learning itself.
It's as “smart” as a voxelgrid.

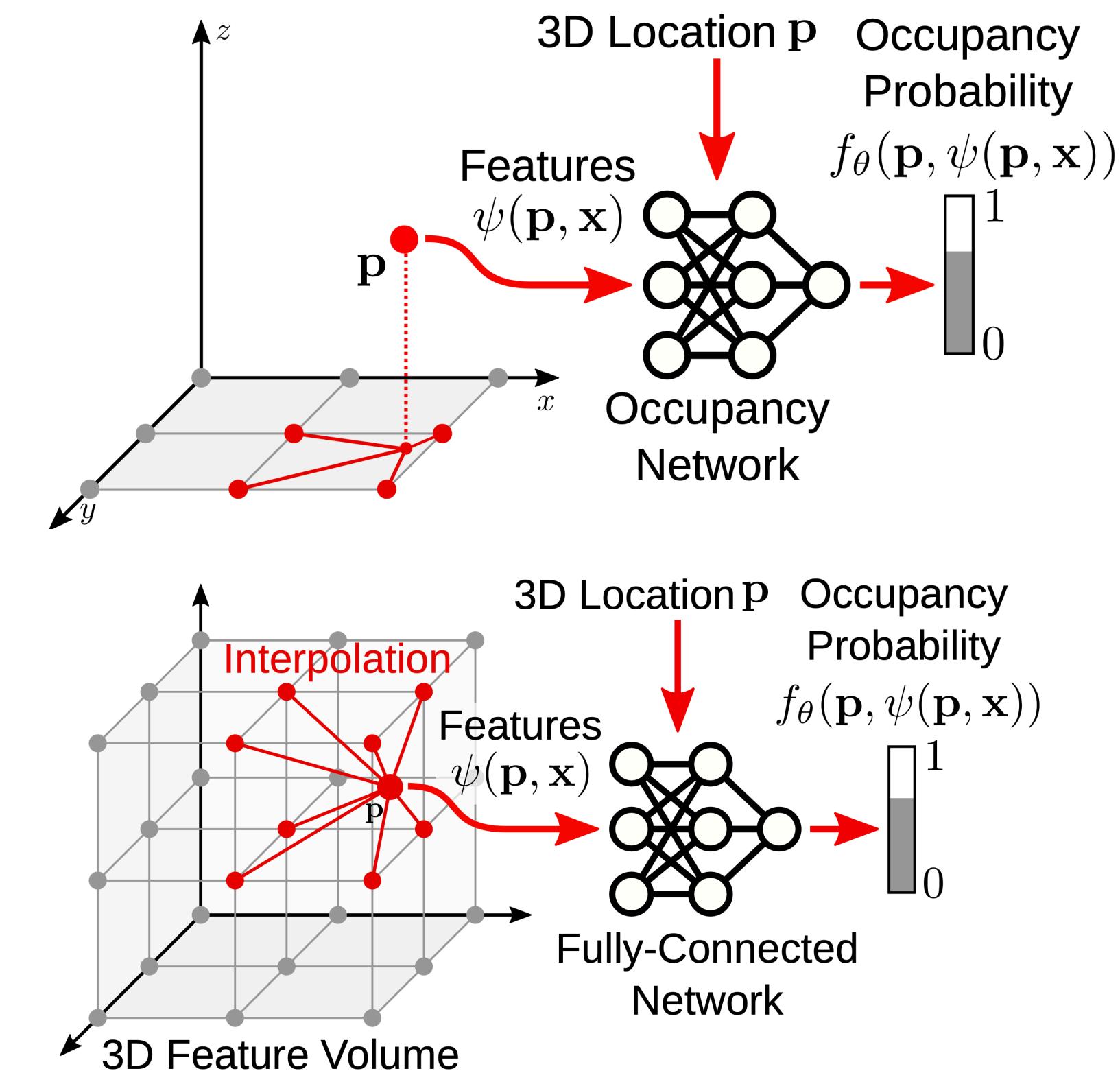
Hybrid discrete / continuous reps



Discrete Parameterizations

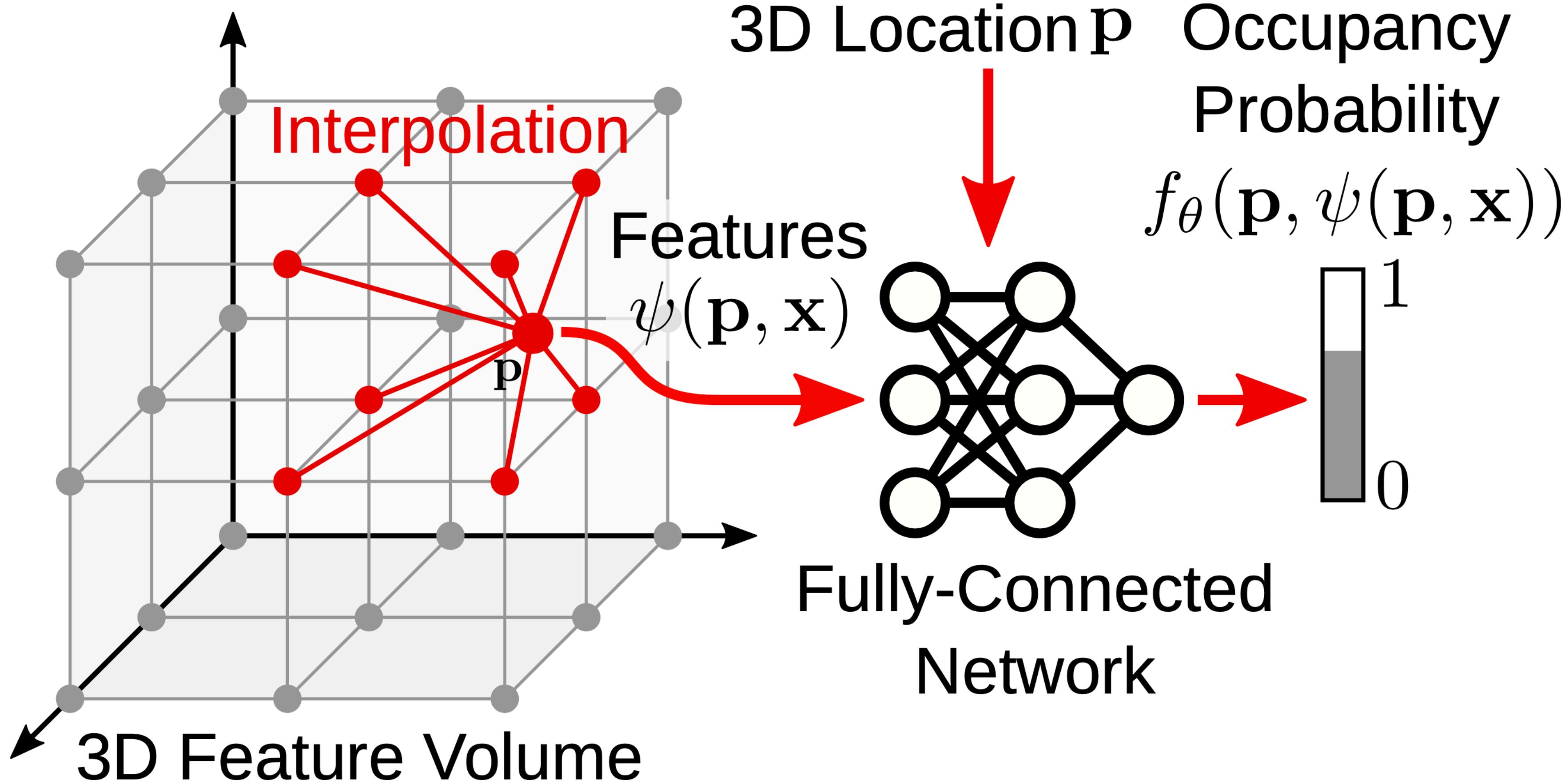


Continuous Parameterizations



Hybrid Parameterizations

Basic idea: Use Neural Net as Interpolation Kernel



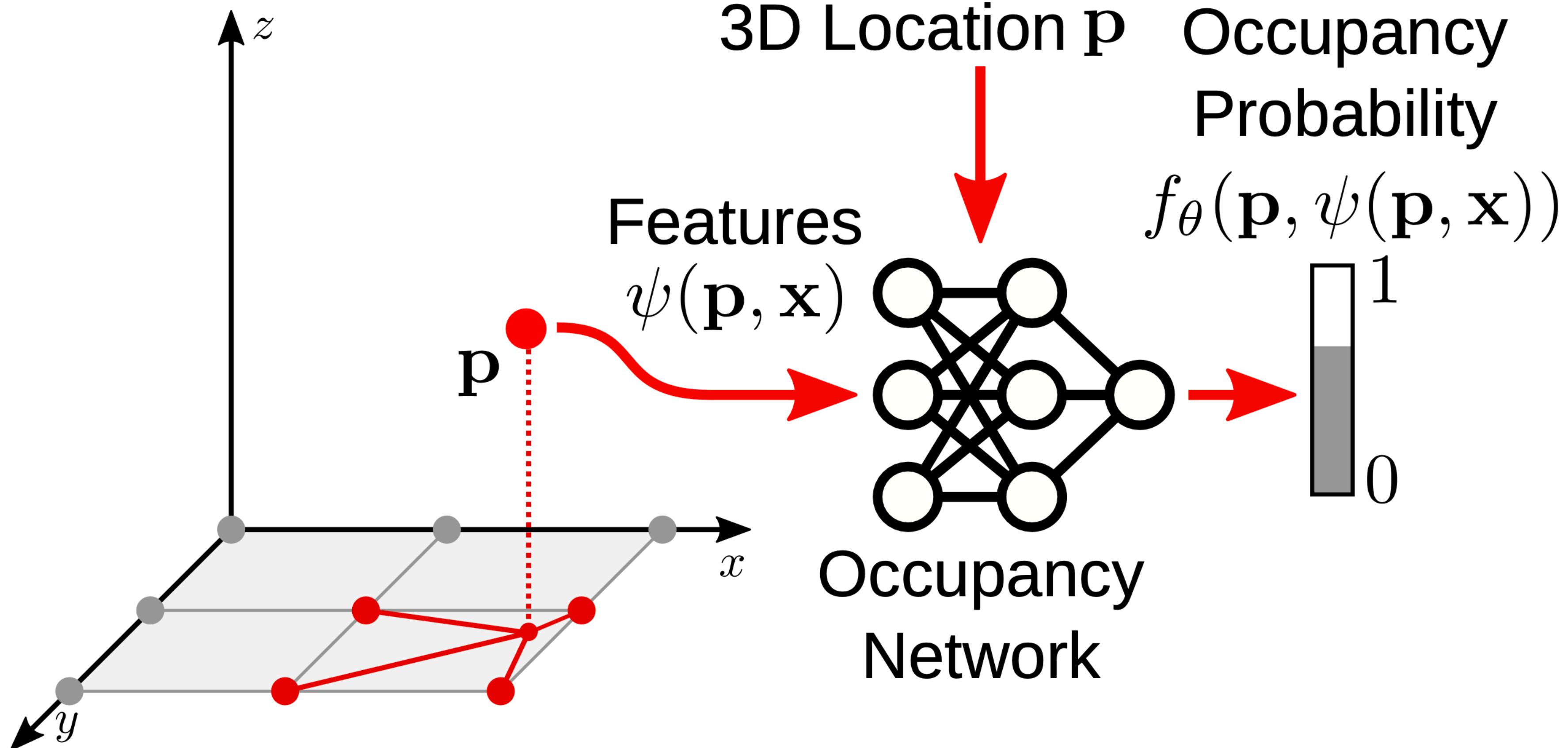
Convolutional Occupancy Networks [Peng et al. 2020]

Local Implicit Grid Representations for 3D Scenes [Jiang et al. 2020]

Implicit Functions in Feature Space for 3D Shape Reconstruction and Completion [Chibane et al. 2020]

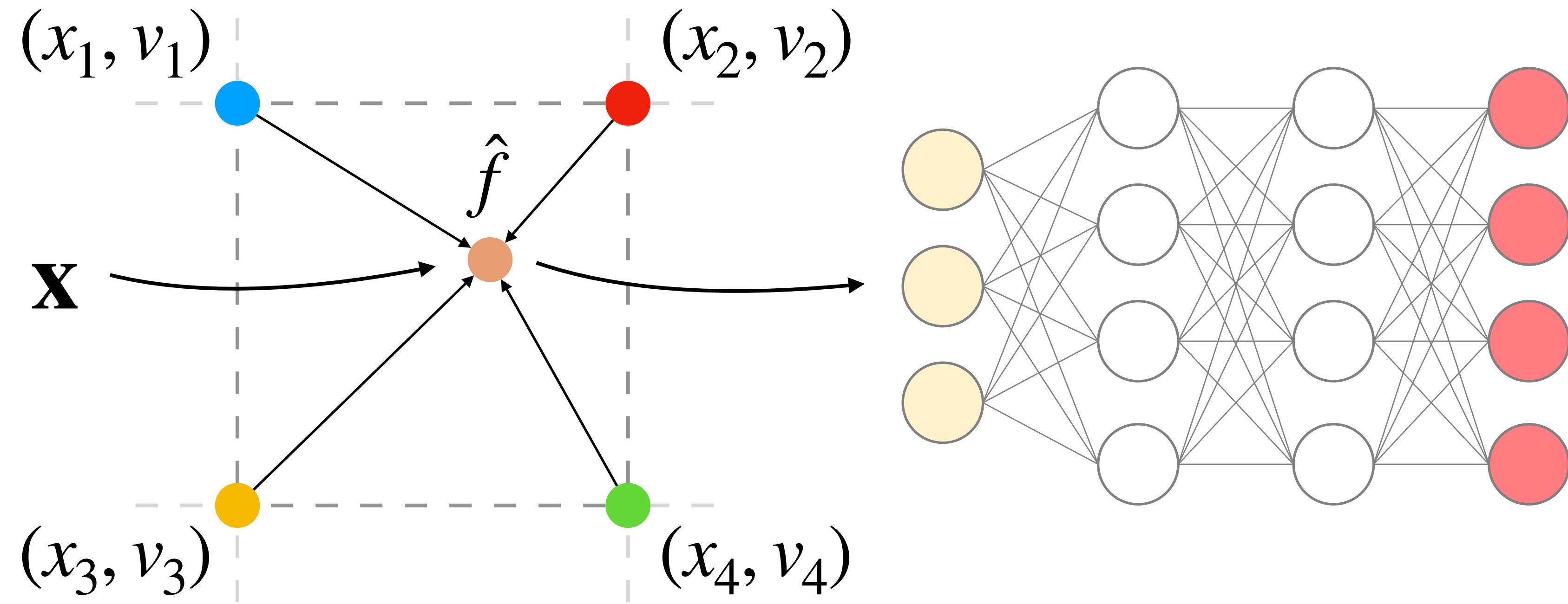
Deep Local Shapes: Learning Local SDF Priors for Detailed 3D Reconstruction [Chabra et al. 2020]

Ground plan & Orthographic Projection

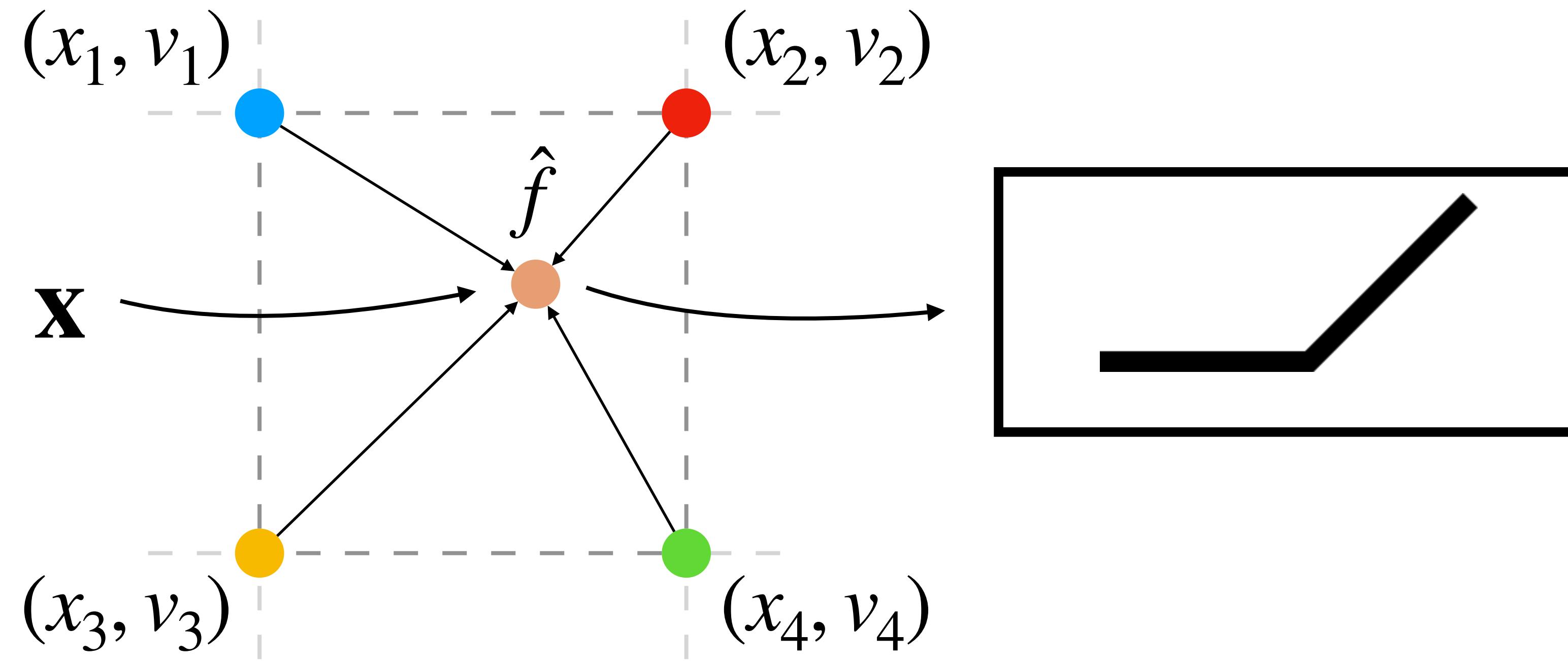


Any assumptions made on scene? Any limitations?

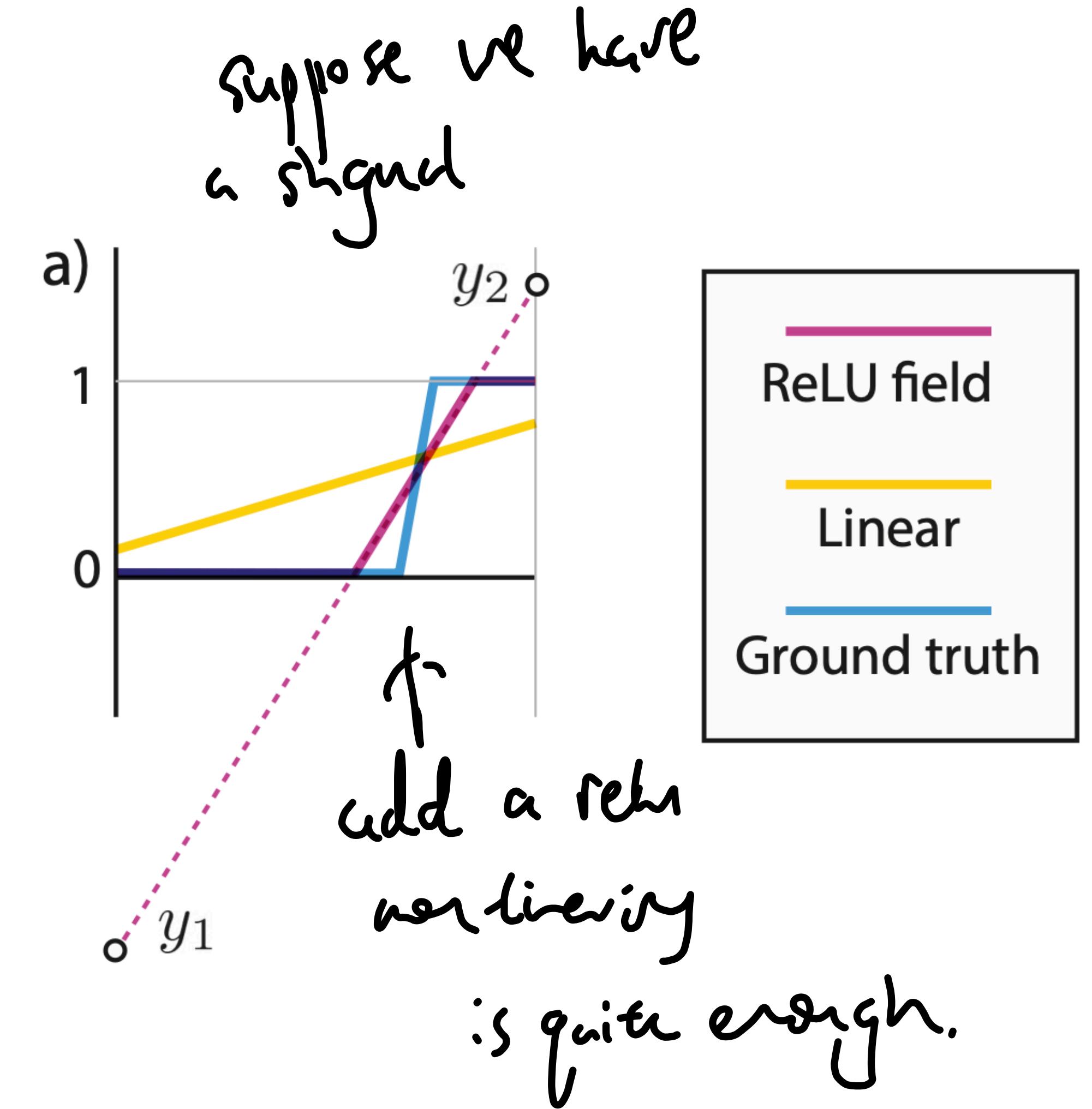
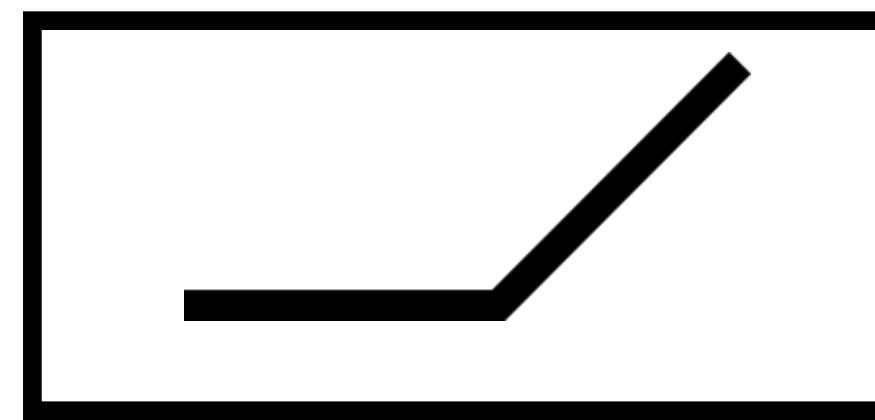
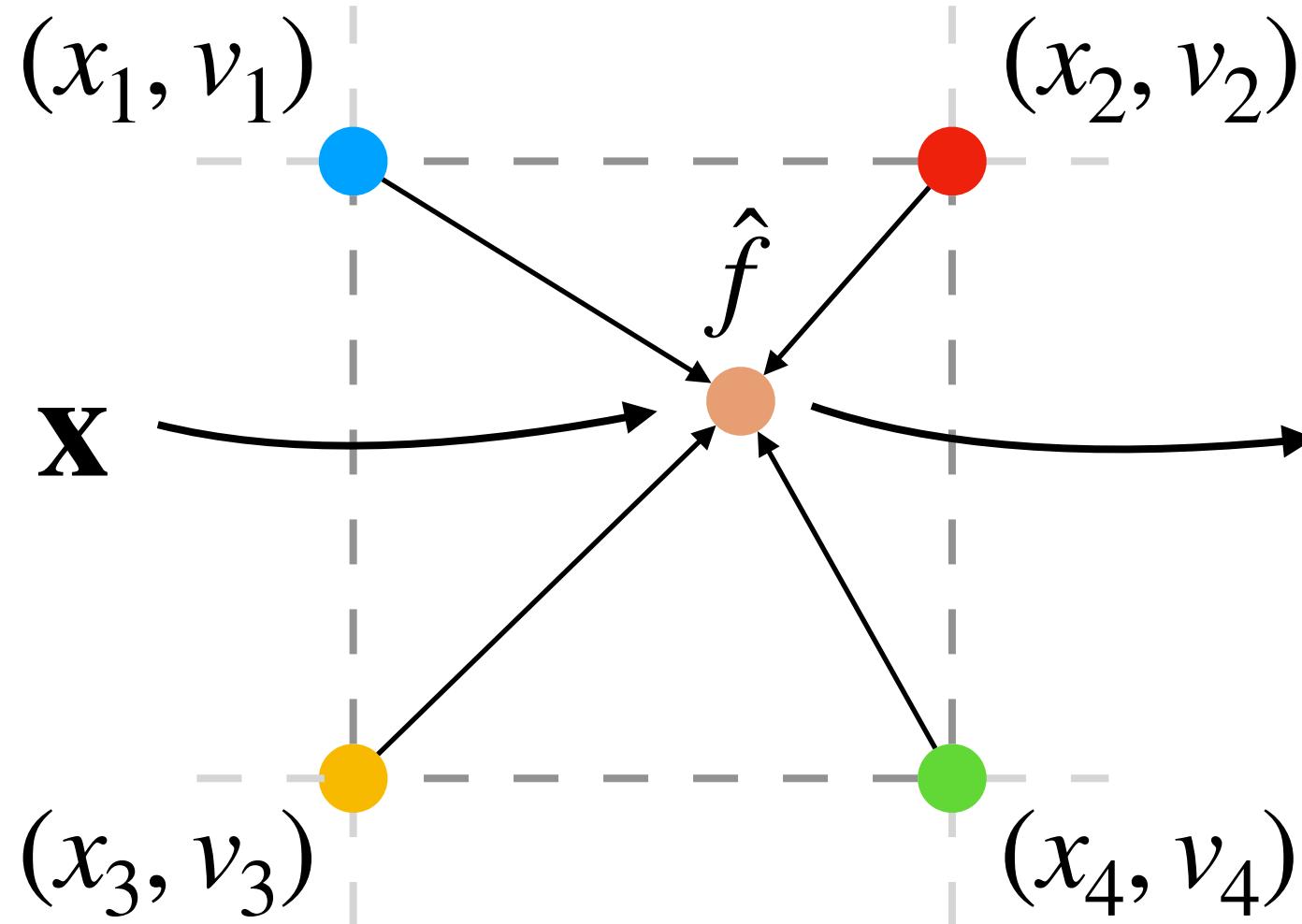
ReLU Fields: How much MLP do you need?



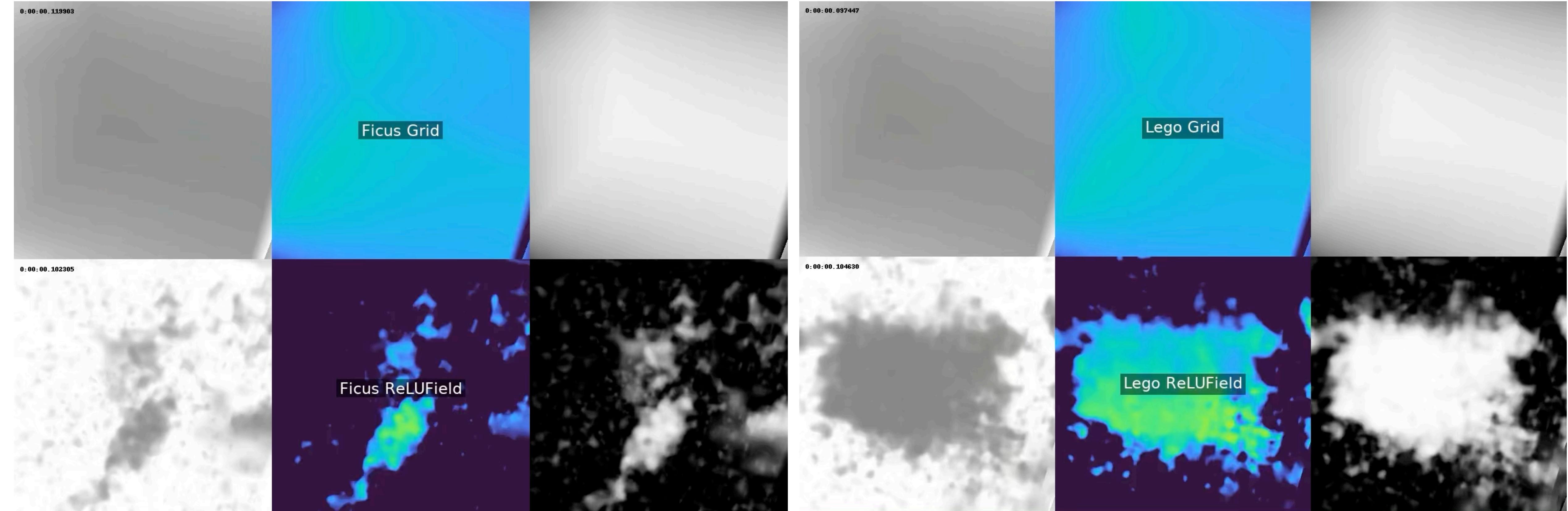
ReLU Fields: How much MLP do you need?



ReLU Fields: How much MLP do you need?



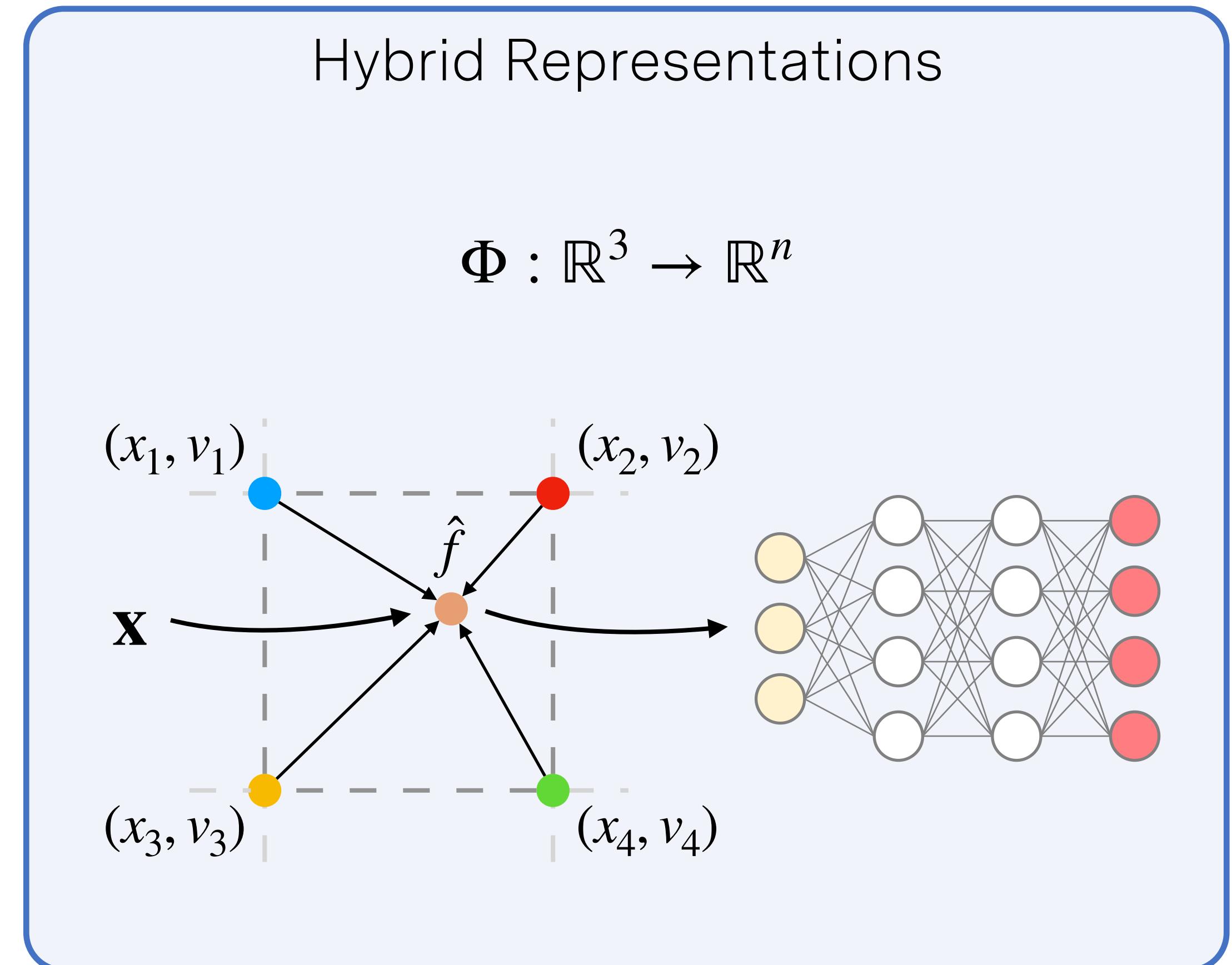
ReLU Fields: How much MLP do you need?



ReLU Fields: The Little Non-linearity That Could (Karnewar et al. 2022)

Hybrid Representations

- Very natural to trade off memory with compute.
- Neural network can locally “super resolve” discrete representation: Basically acts as interpolation kernel.
- Can be the best of both worlds: Fast and “continuous”.

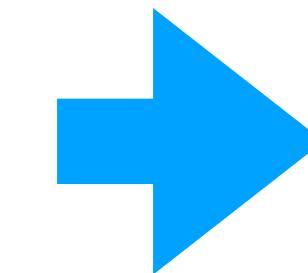
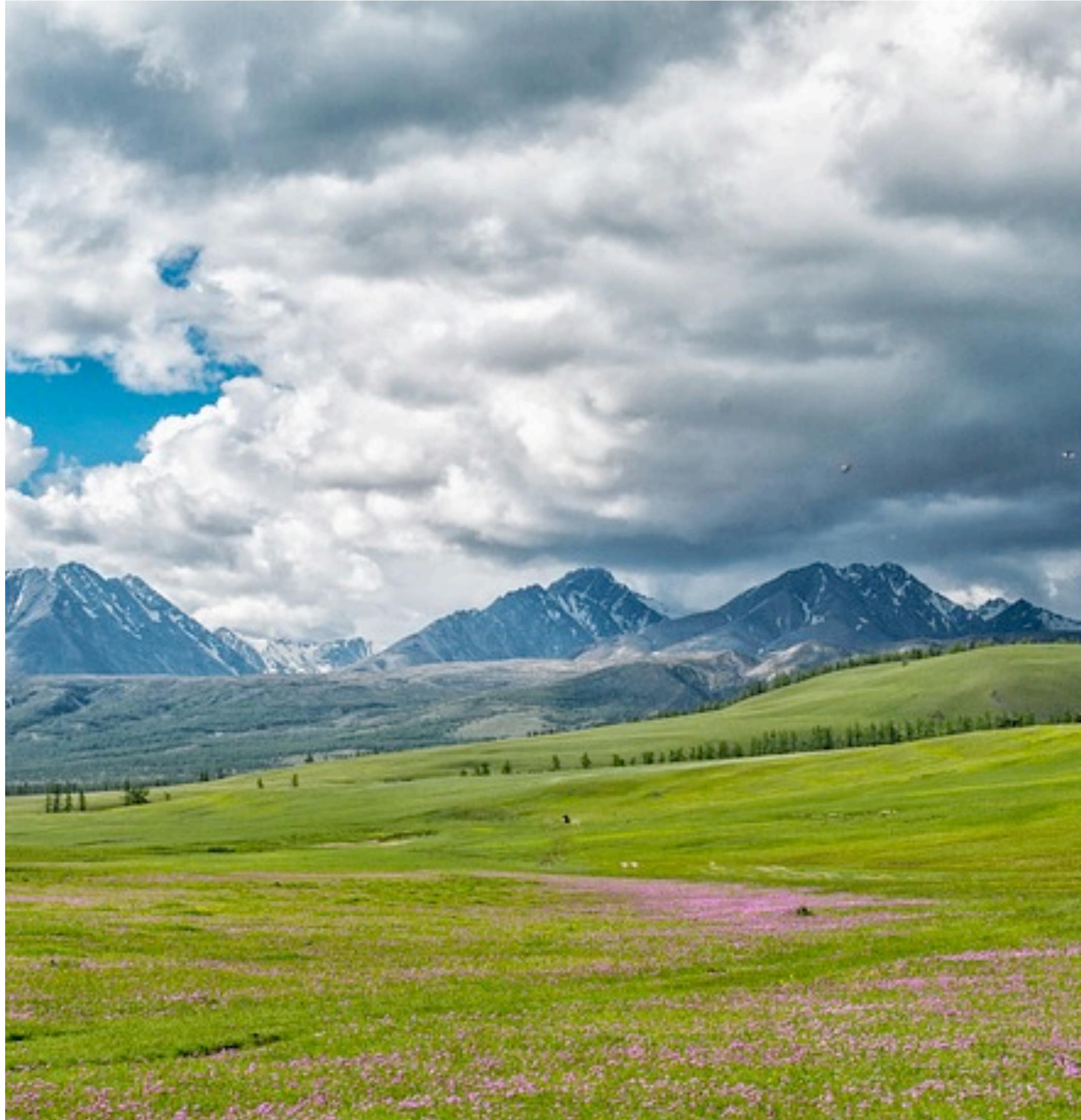


How can we represent unbounded scenes?



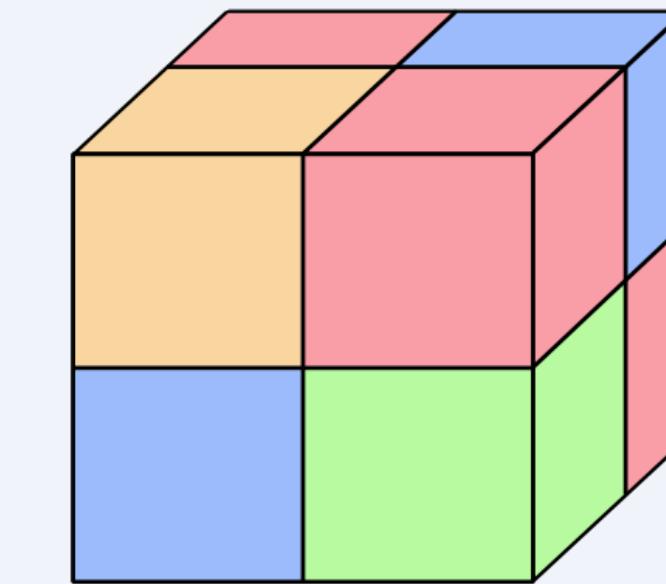
Image Source: Wikipedia

How can we represent unbounded scenes?

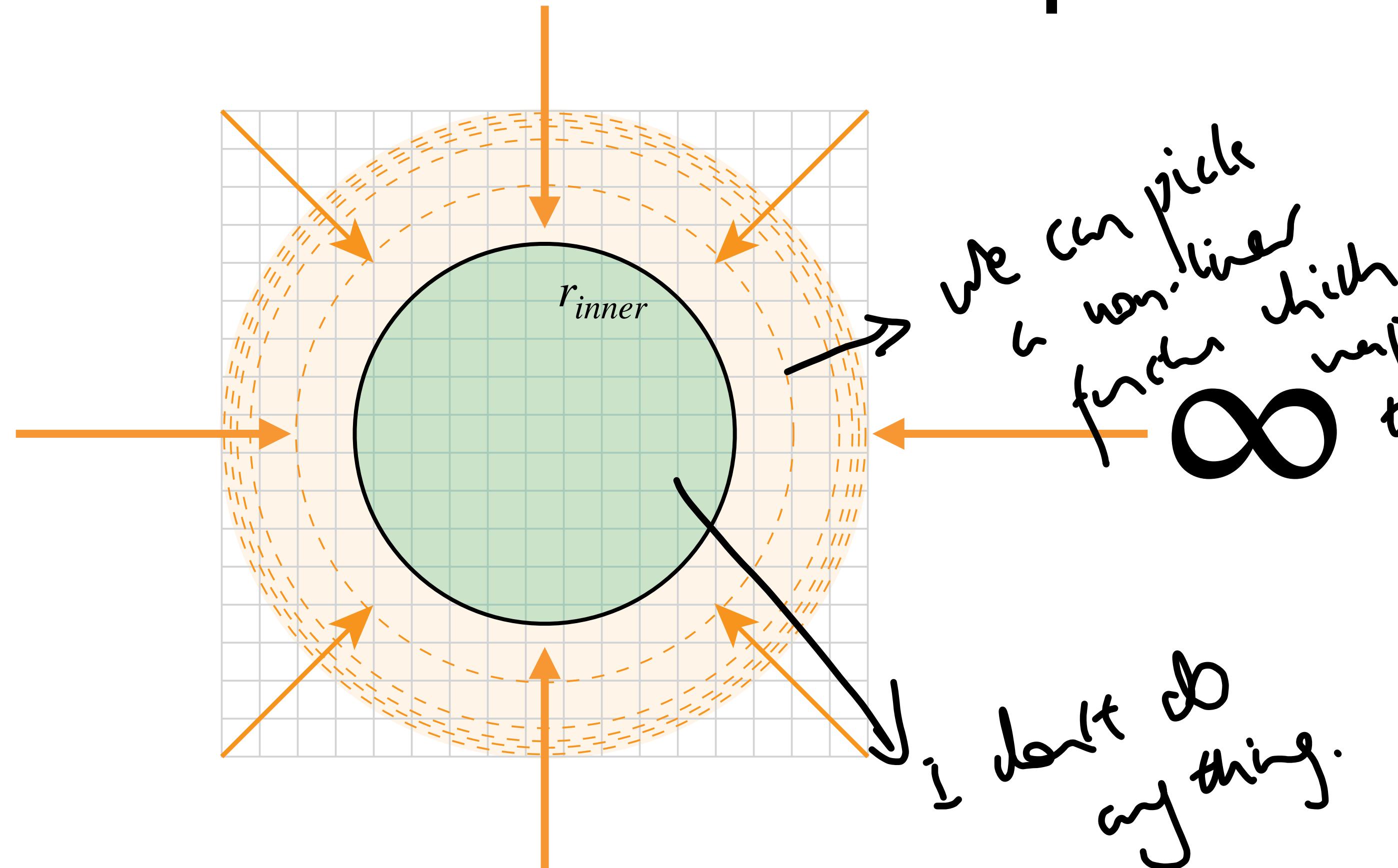


Scene
Representation

$$\Phi : \mathbb{R}^3 \rightarrow \mathbb{R}^n$$



Parameterizing Unbounded Scenes: Compactification

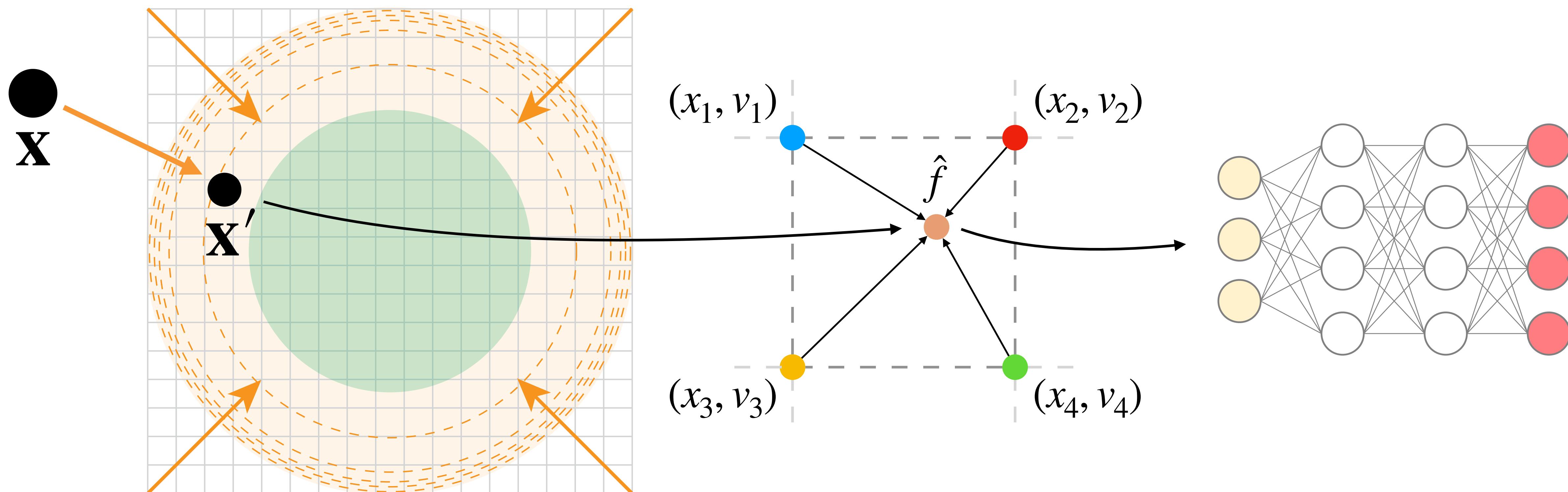


$$\mathbf{x}' = \frac{(1 + k) - k/\|\mathbf{u}\|}{\mathbf{u}/\|\mathbf{u}\|} r_{inner}$$
$$\mathbf{u} = \frac{\mathbf{x}}{r_{inner}}$$

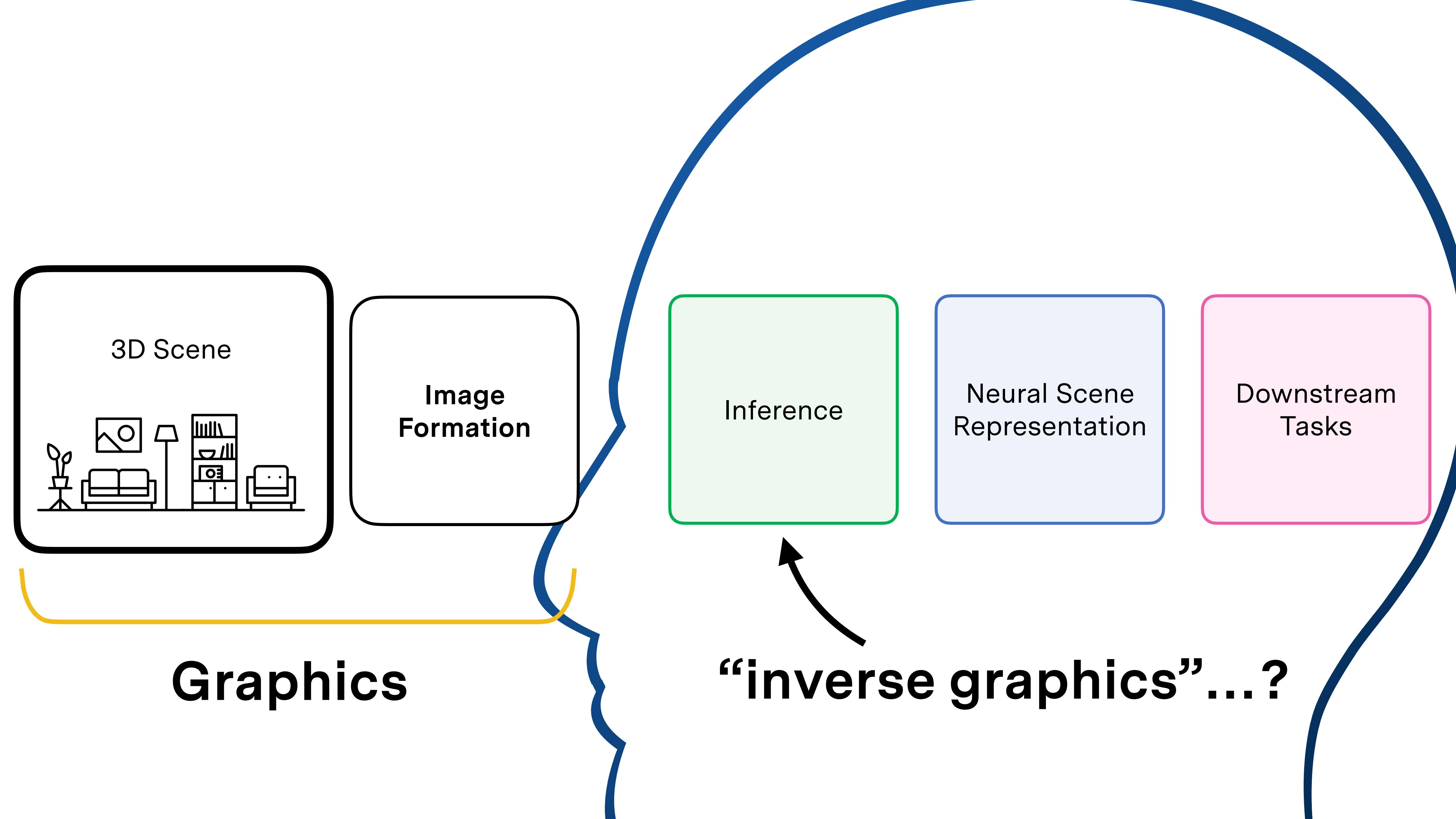
Example Compactification function from
“Mip-NeRF 360: Unbounded Anti-Aliased Neural
Radiance Fields”, Barron et al., CVPR 2022

Come up with nonlinear mapping that “squashes” space, such that infinity is mapped to some **finite** value. Aligns with intuition “further away, less resolution”.

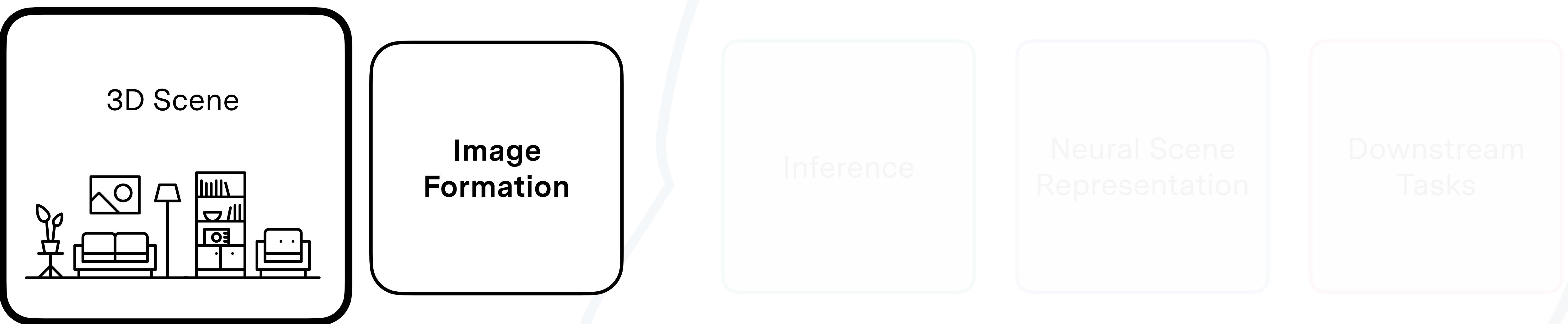
Parameterizing Unbounded Scenes: Compactification



RENDERING & Computer Graphics



Today: Light Transport & High-level of Physics-based Rendering



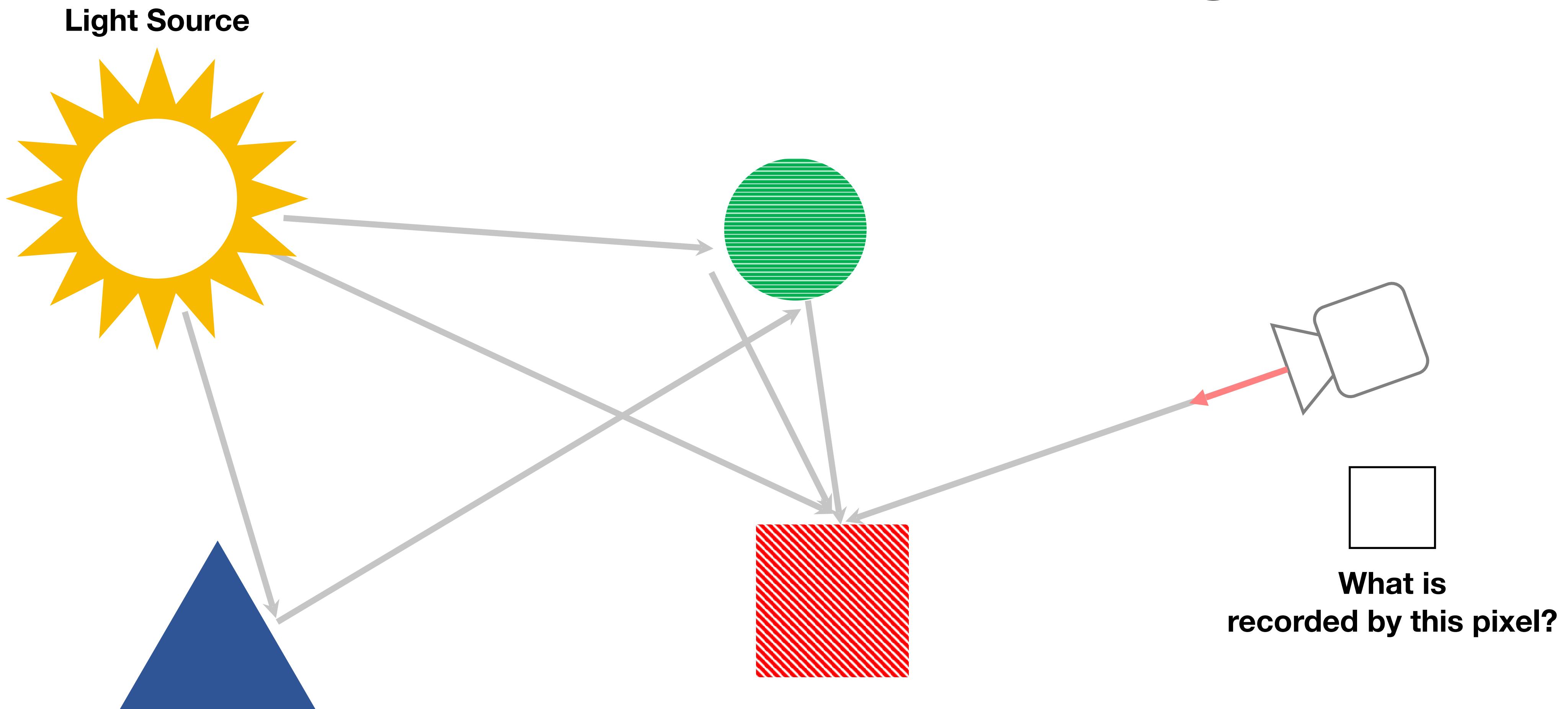
Why?

In order to reason about the 3D world from images, we need to understand how 3D properties such as materials, lighting, etc. relate to the measurements observed by a camera.

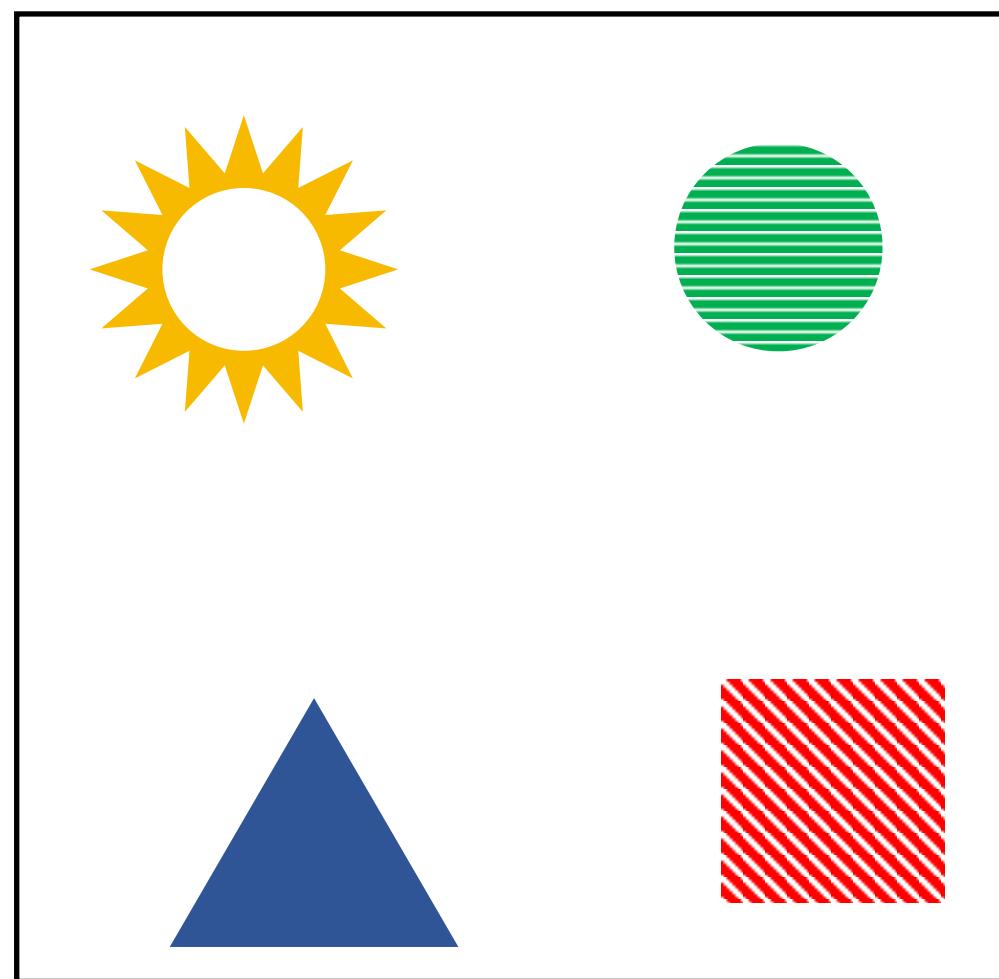
What you'll learn.

The rendering equation, simulating light transport via multi-bounce ray-tracing, bidirectional radiance distribution functions, rasterization, rendering.

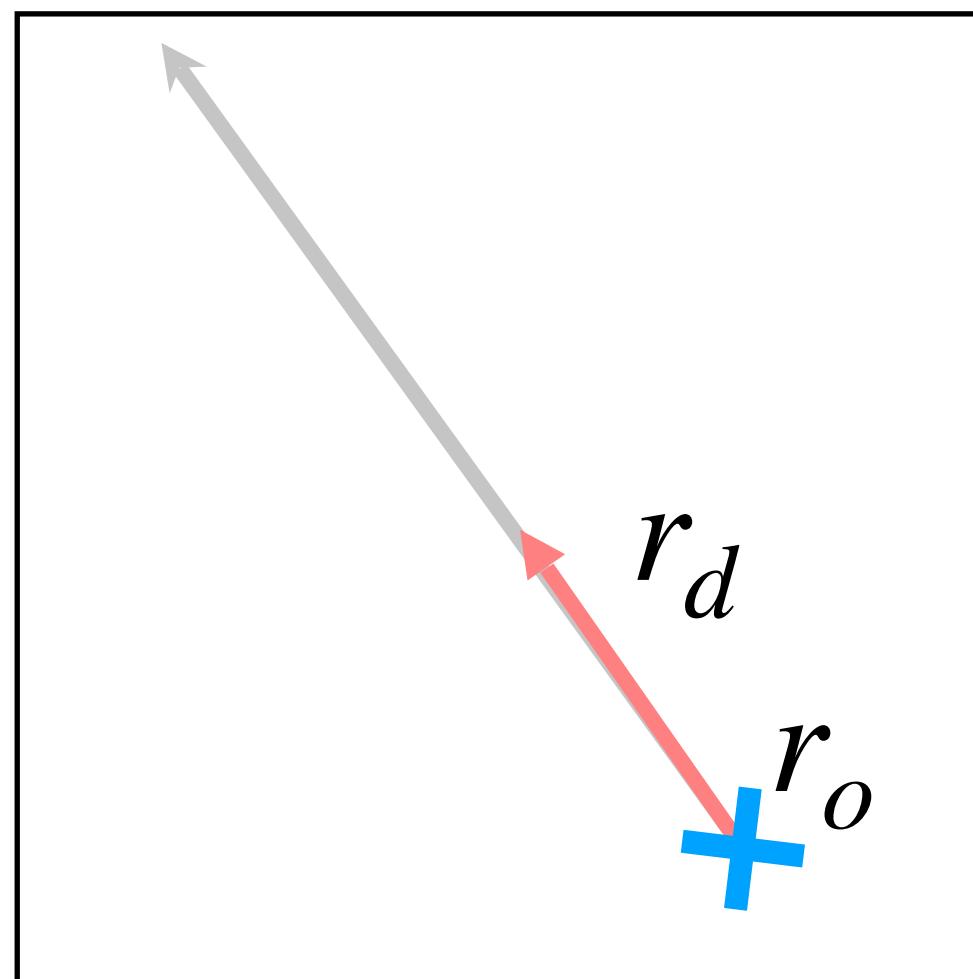
What is Rendering?



Function Signature of Rendering



Scene : Light sources,
materials, shapes, ...



Camera Ray



Radiance of the ray

Some Slides Adopted From

Stanford CS348I: Computer Graphics in the Era of AI
Profs. C. Karen Liu and Jiajun Wu

The Rendering equation

The moral of the story:

$$L_o(\mathbf{x}, \omega_o, \lambda) = L_e(\mathbf{x}, \omega_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \omega_i, \omega_o, \lambda) L_i(\mathbf{x}, \omega_i, \lambda) (\omega_i \cdot \mathbf{n}) d\omega_i$$

Radiance

- Radiance is a fundamental quantity of light.
- Radiance is density of photons at a point, traveling in the same direction, at given wavelength.
- Radiance is energy per time, per unit area, per unit solid angle, per wavelength.

$$L(\mathbf{x}, \omega, \lambda) : \mathbb{R}^3 \times \mathbb{S}^2 \times \mathbb{R} \mapsto \mathbb{R}^+$$

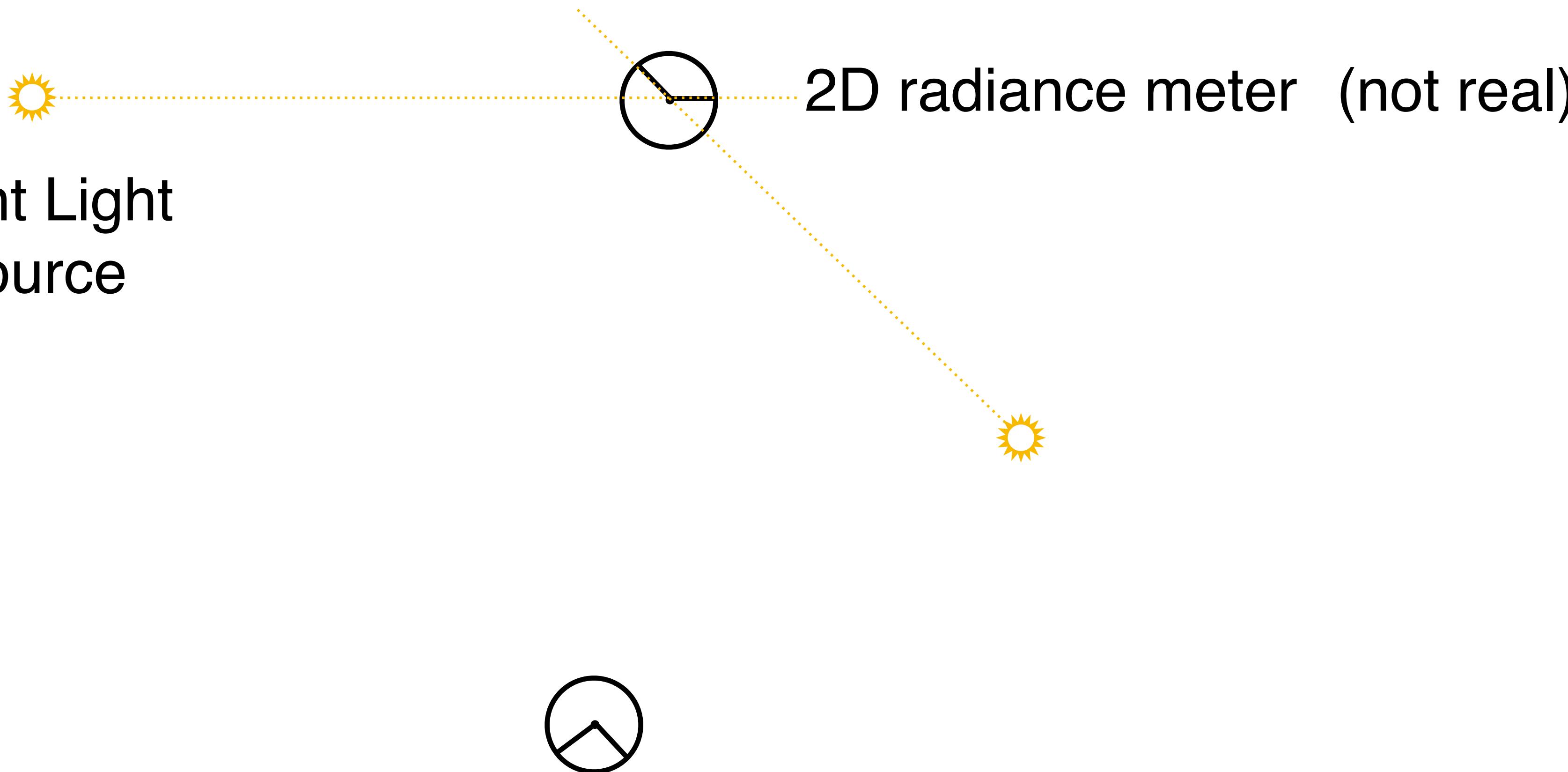
position direction wavelength

The diagram shows a mathematical function $L(\mathbf{x}, \omega, \lambda)$ with three input variables. The first variable, "position", is represented by a horizontal arrow pointing towards the function. The second variable, "direction", is represented by a vertical arrow pointing upwards towards the function. The third variable, "wavelength", is represented by a diagonal arrow pointing towards the function from the right. The function itself is written as $L(\mathbf{x}, \omega, \lambda) : \mathbb{R}^3 \times \mathbb{S}^2 \times \mathbb{R} \mapsto \mathbb{R}^+$.

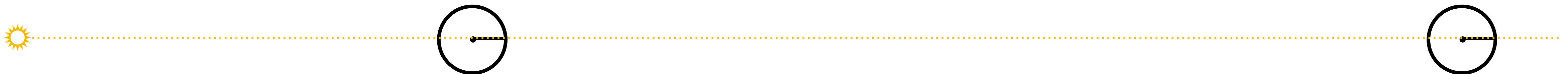
The “Light Field”: Radiance for every possible ray

Radiance intuition

Point Light
Source



Radiance intuition



Radiance intuition



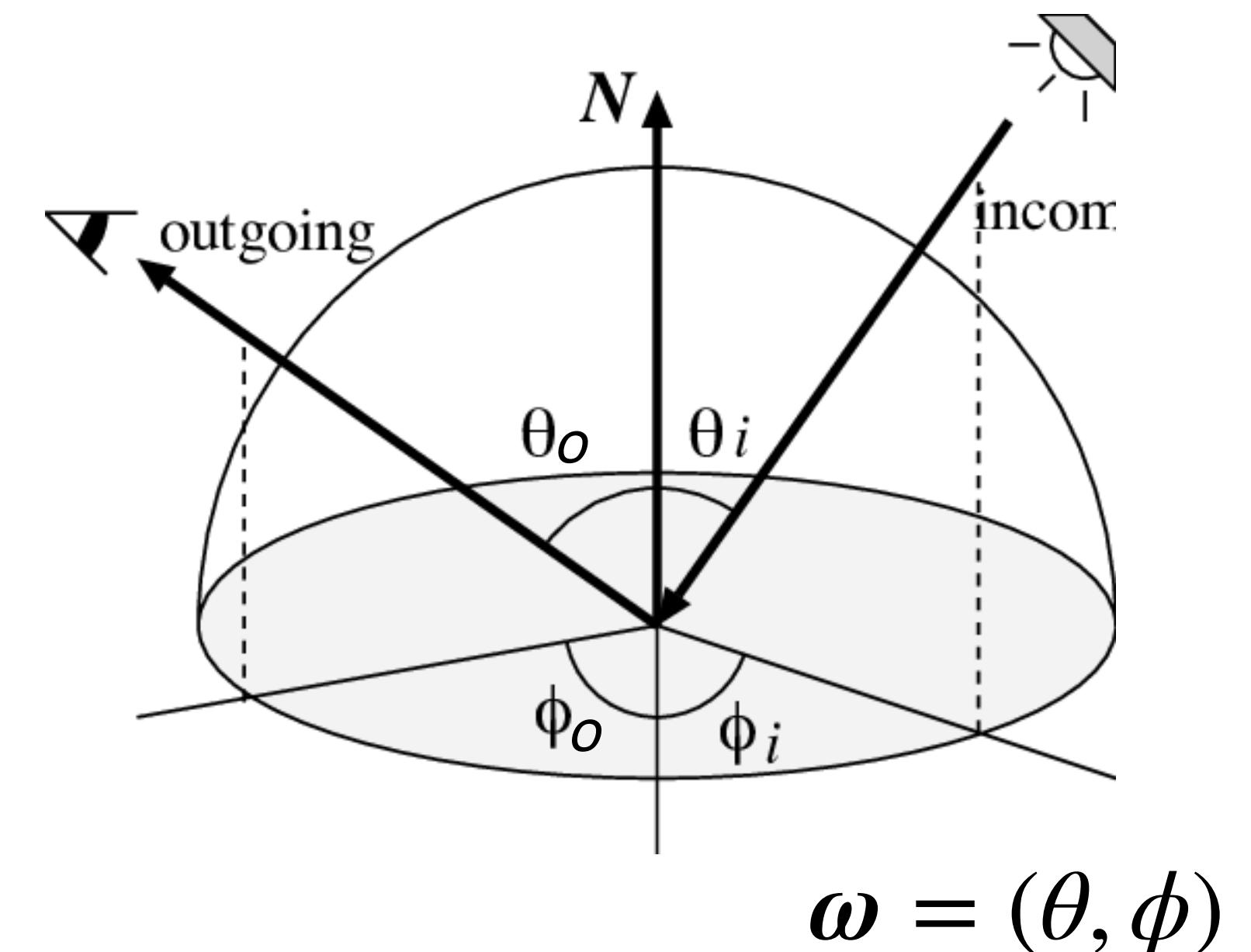
- Radiance properties
 - Radiance along an unblocked ray is constant.
 - Response of sensors is proportional to radiance of visible surface.
 - Radiance of reflected light is proportional to that of incoming light:

BRDF

- Constant of proportionality defines Bidirectional Reflectance Distribution Function (BRDF).

$$f(\omega_i, \omega_o) = \frac{\text{outgoing light in direction } \omega_o}{\text{incoming light in direction } \omega_i}$$

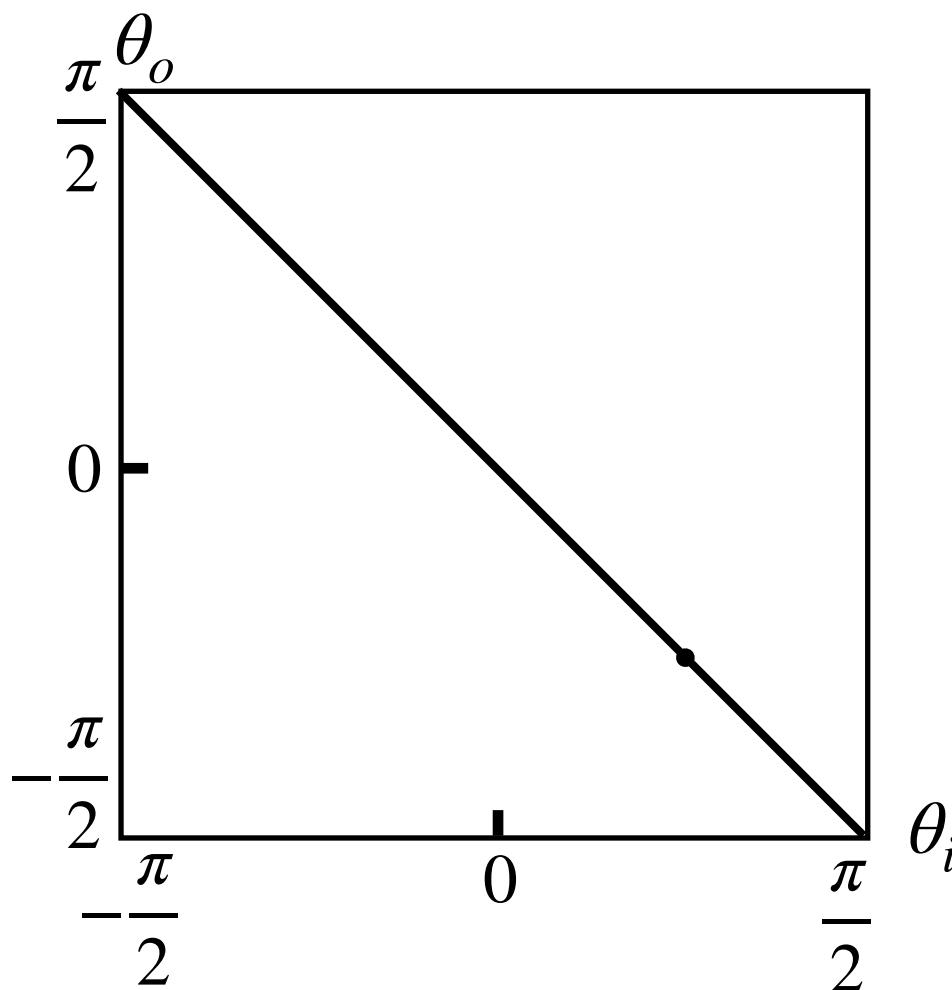
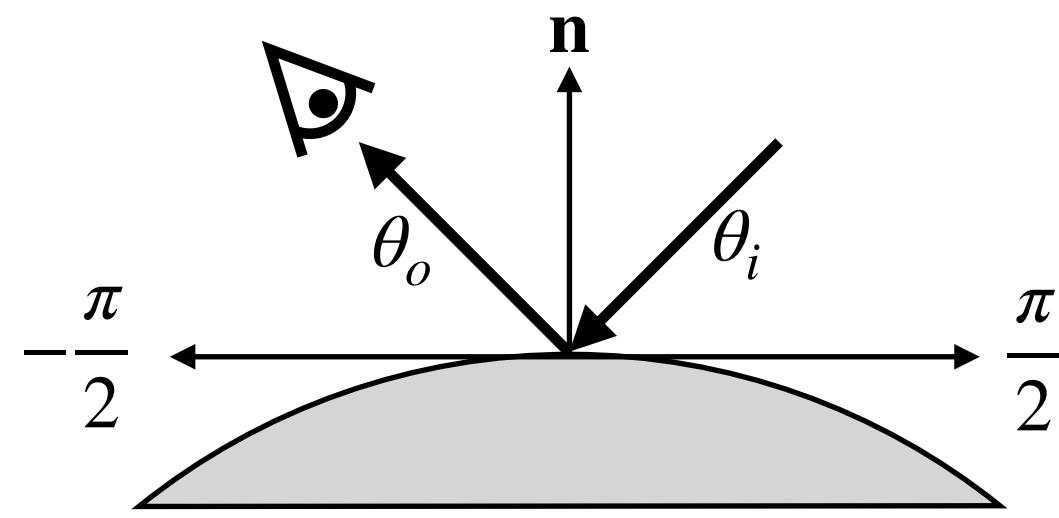
- BRDF describes how a material reflects light.
- Helmholtz reciprocity: $f(\omega_1, \omega_2) = f(\omega_2, \omega_1)$



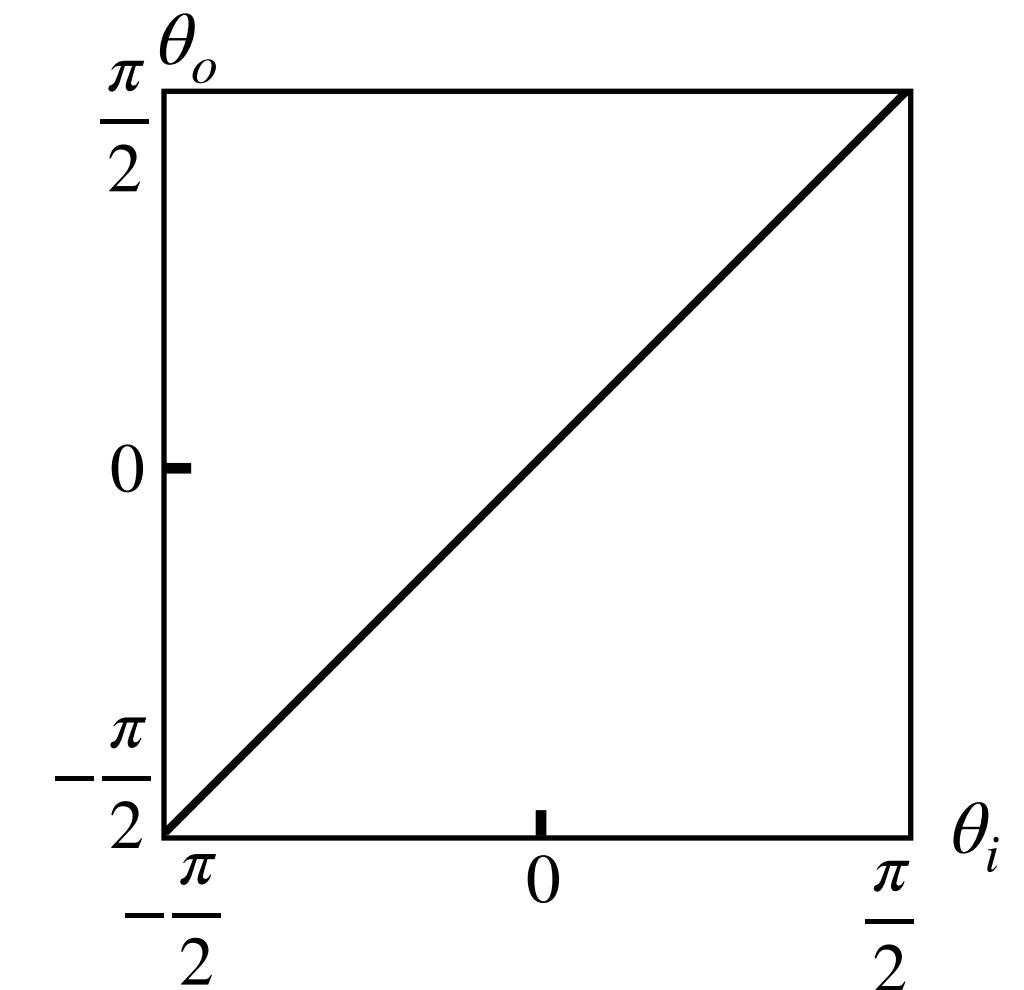
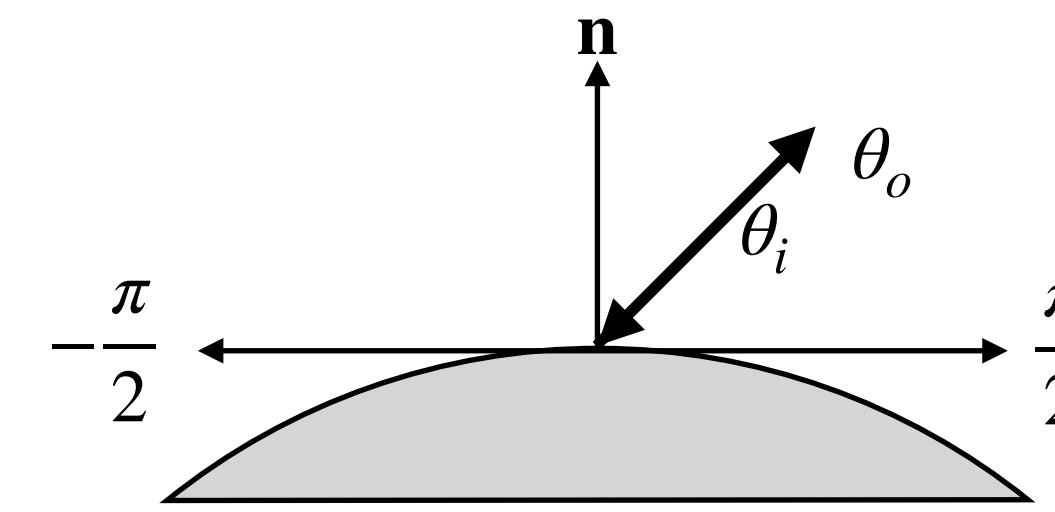
$$\omega = (\theta, \phi)$$

BRDF intuition

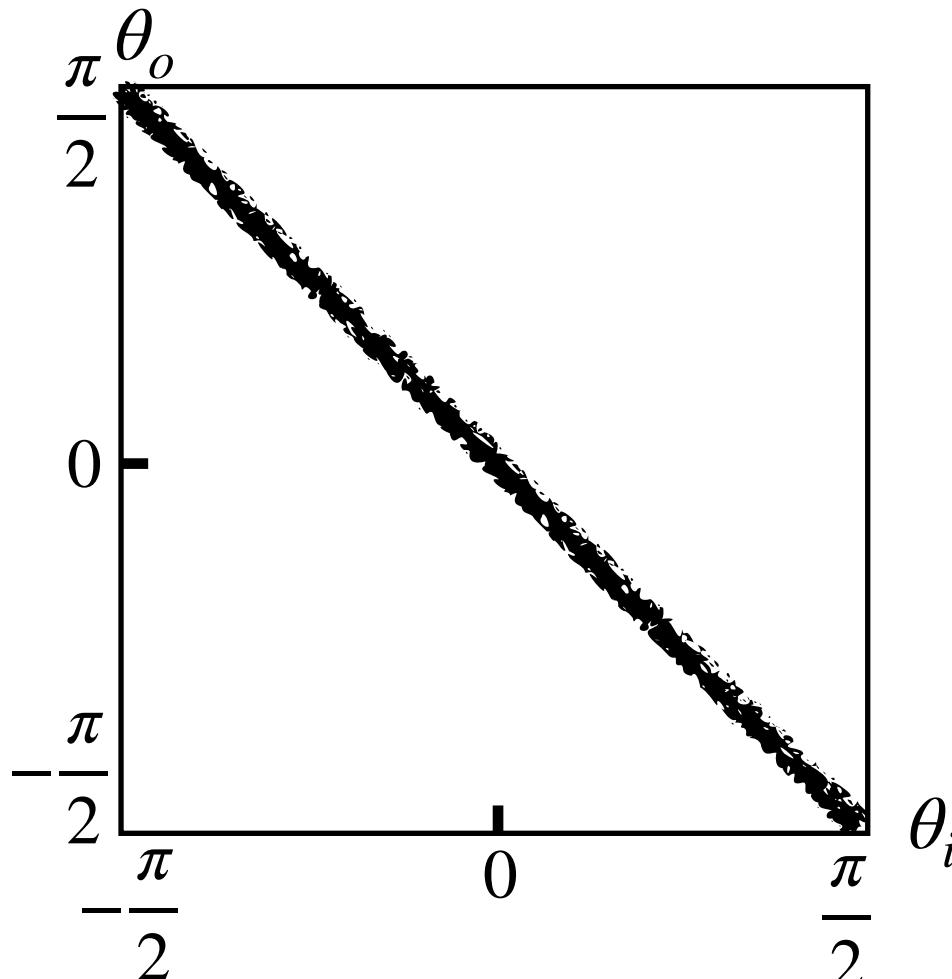
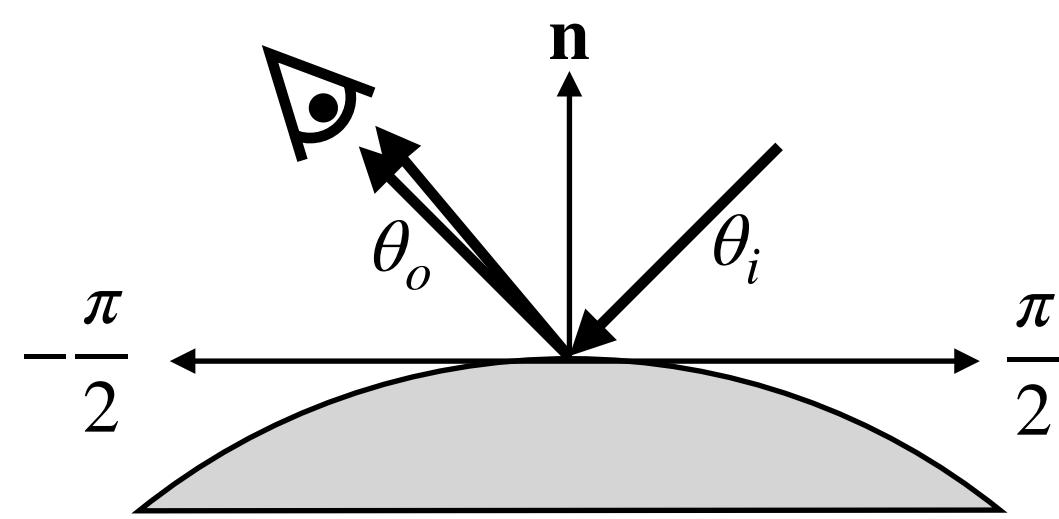
perfect mirror



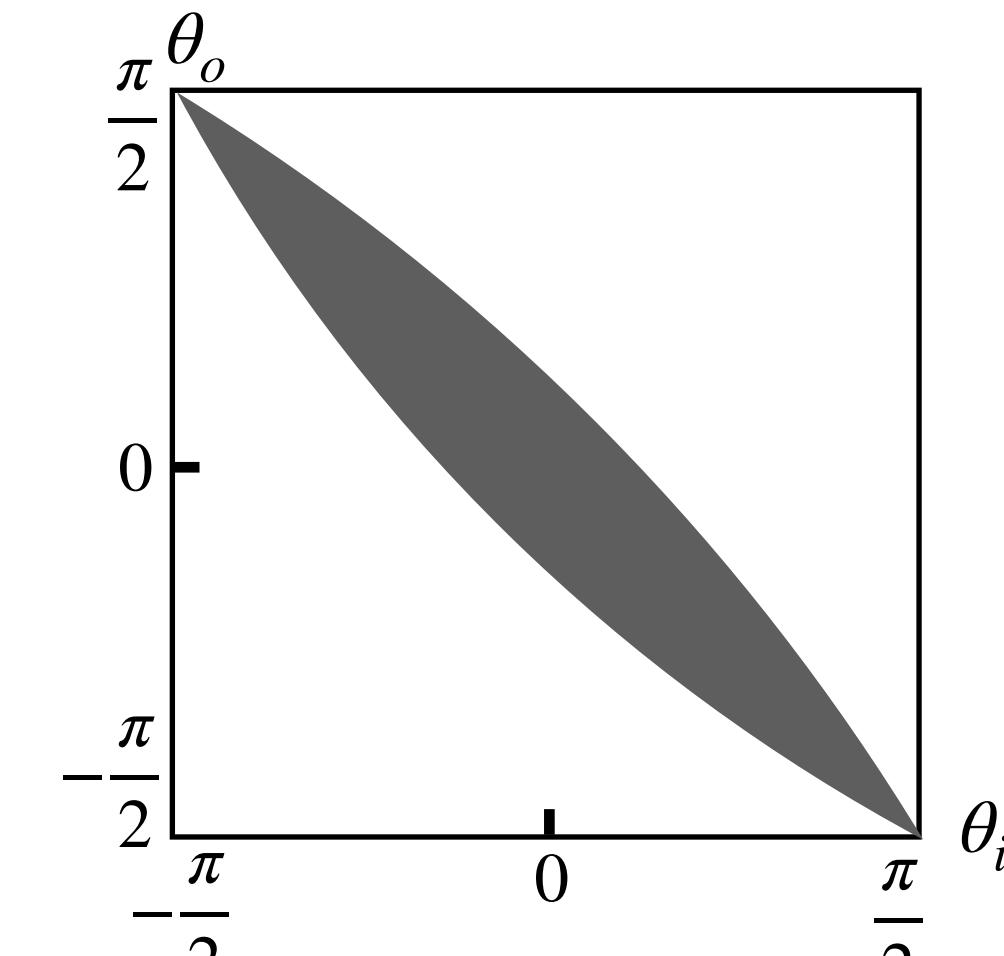
retro-reflector



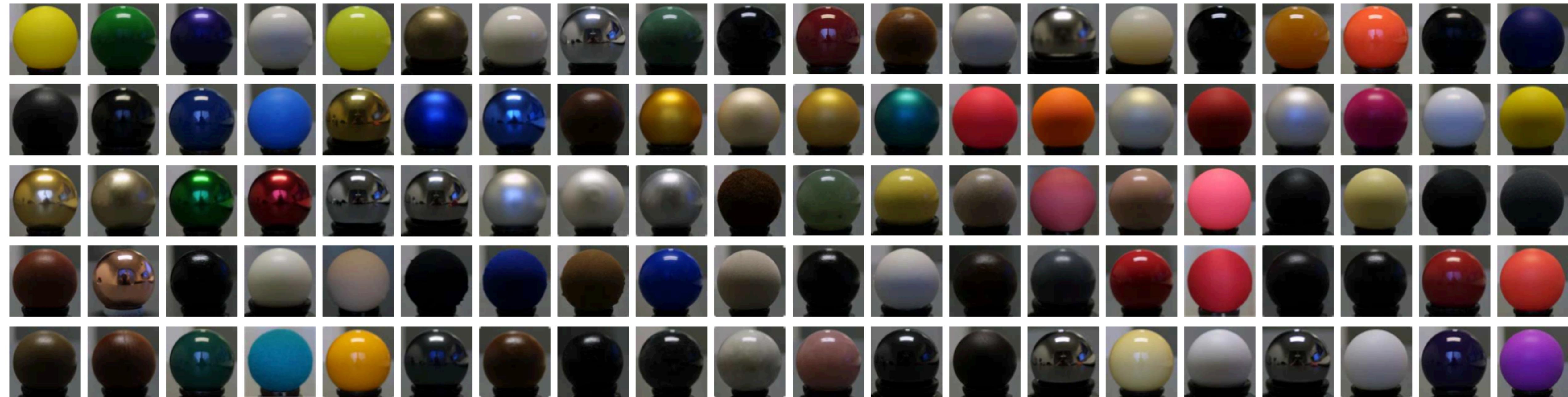
fairly shiny mirror



How does BRDF
for paper look like?

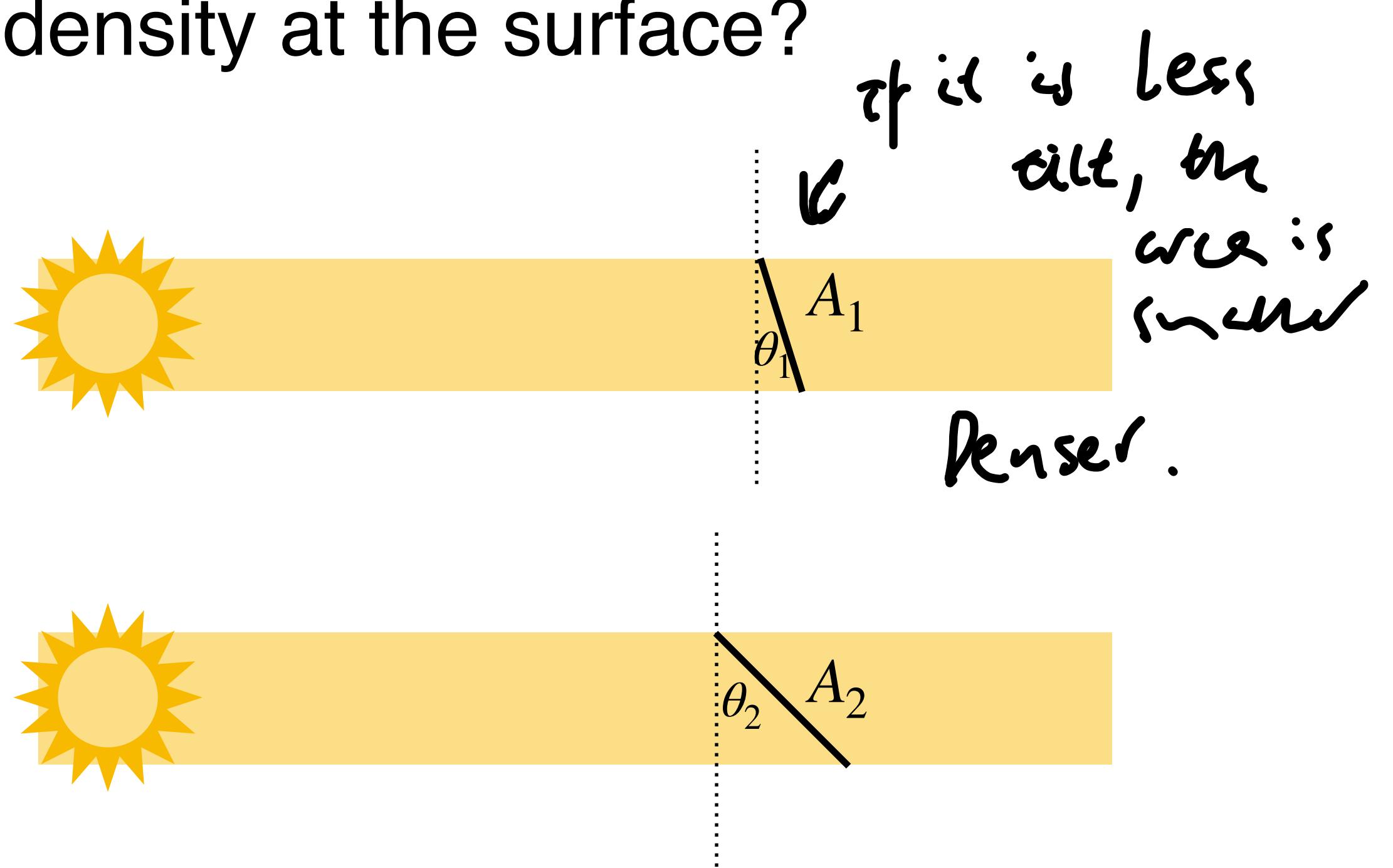
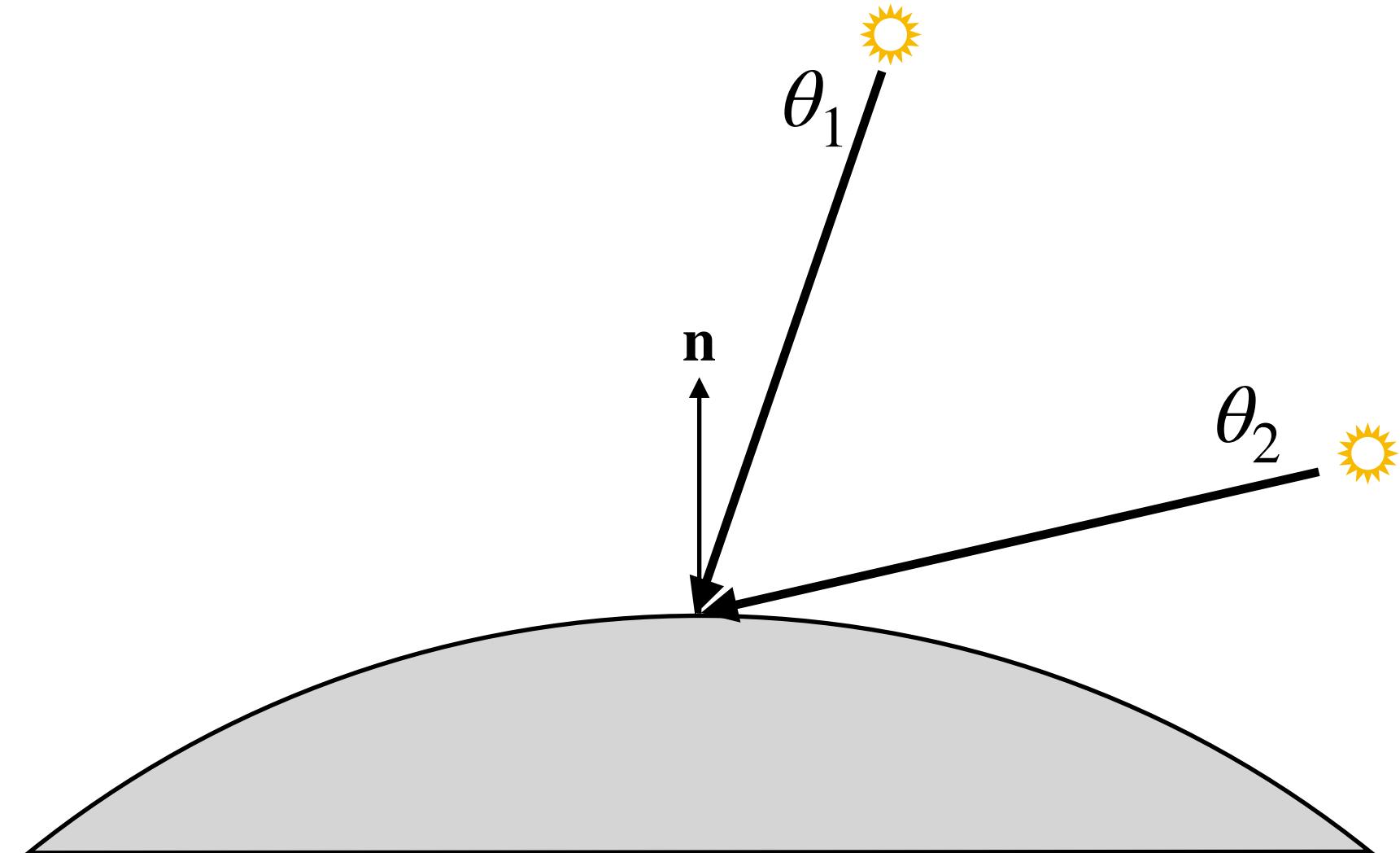


Different materials



Angle Weight of Photon density

Which incoming light results in more photon density at the surface?



Rendering equation

all possible light could be coming from.

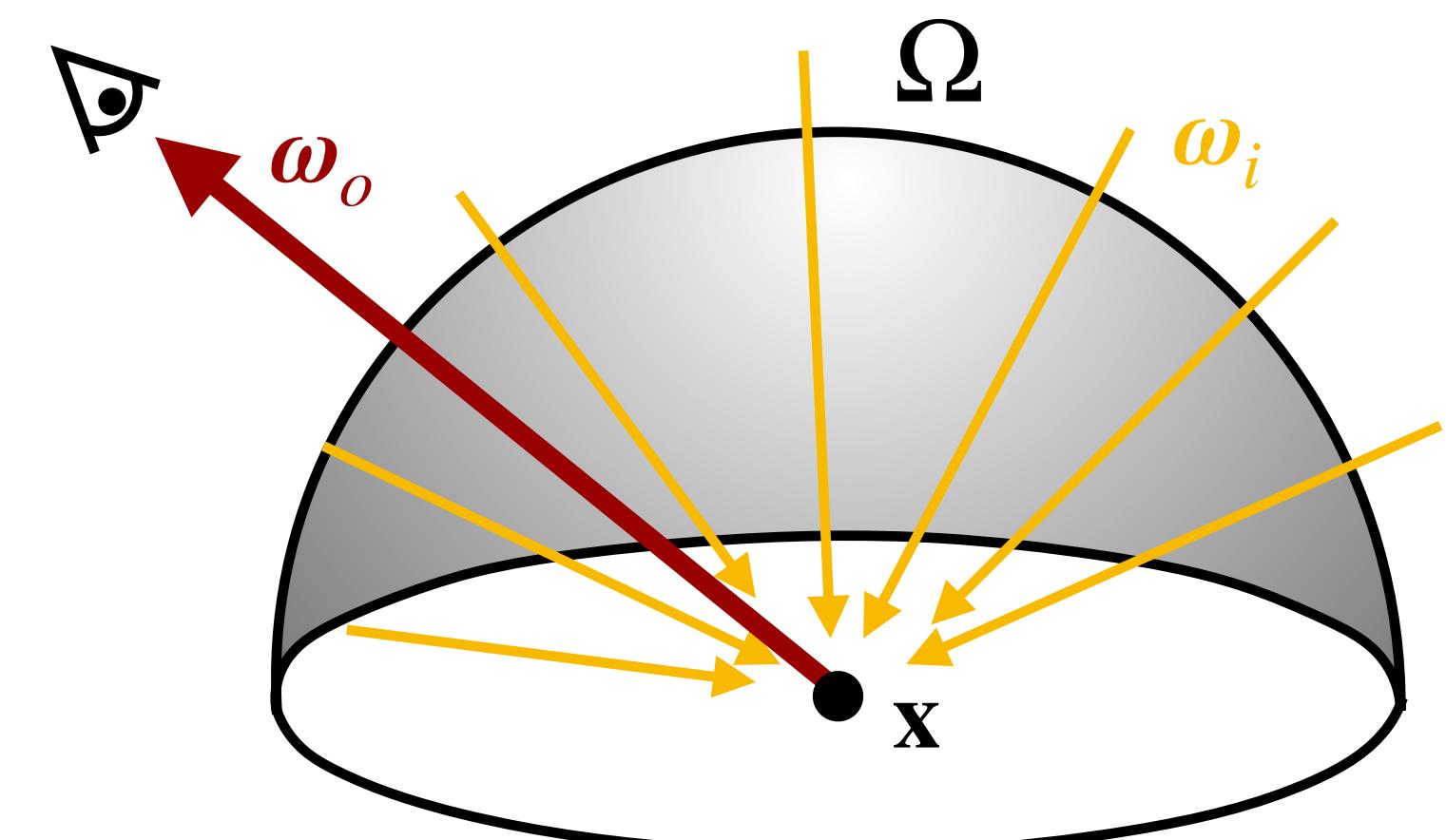
$$L_o(\mathbf{x}, \omega_o, \lambda) = L_e(\mathbf{x}, \omega_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \omega_i, \omega_o, \lambda) L_i(\mathbf{x}, \omega_i, \lambda) (\omega_i \cdot \mathbf{n}) d\omega_i$$

Diagram illustrating the Rendering Equation:

- emitted radiance**: $L_e(\mathbf{x}, \omega_o, \lambda)$
- outgoing radiance in ω_o** : $L_o(\mathbf{x}, \omega_o, \lambda)$
- fraction of incoming light that is reflected into ω_o** : $f(\mathbf{x}, \omega_i, \omega_o, \lambda)$
- incoming radiance**: $L_i(\mathbf{x}, \omega_i, \lambda)$
- angle weight of photon density**: $(\omega_i \cdot \mathbf{n}) d\omega_i$
- solid angle differential**: $d\omega_i$

Conservation of energy:

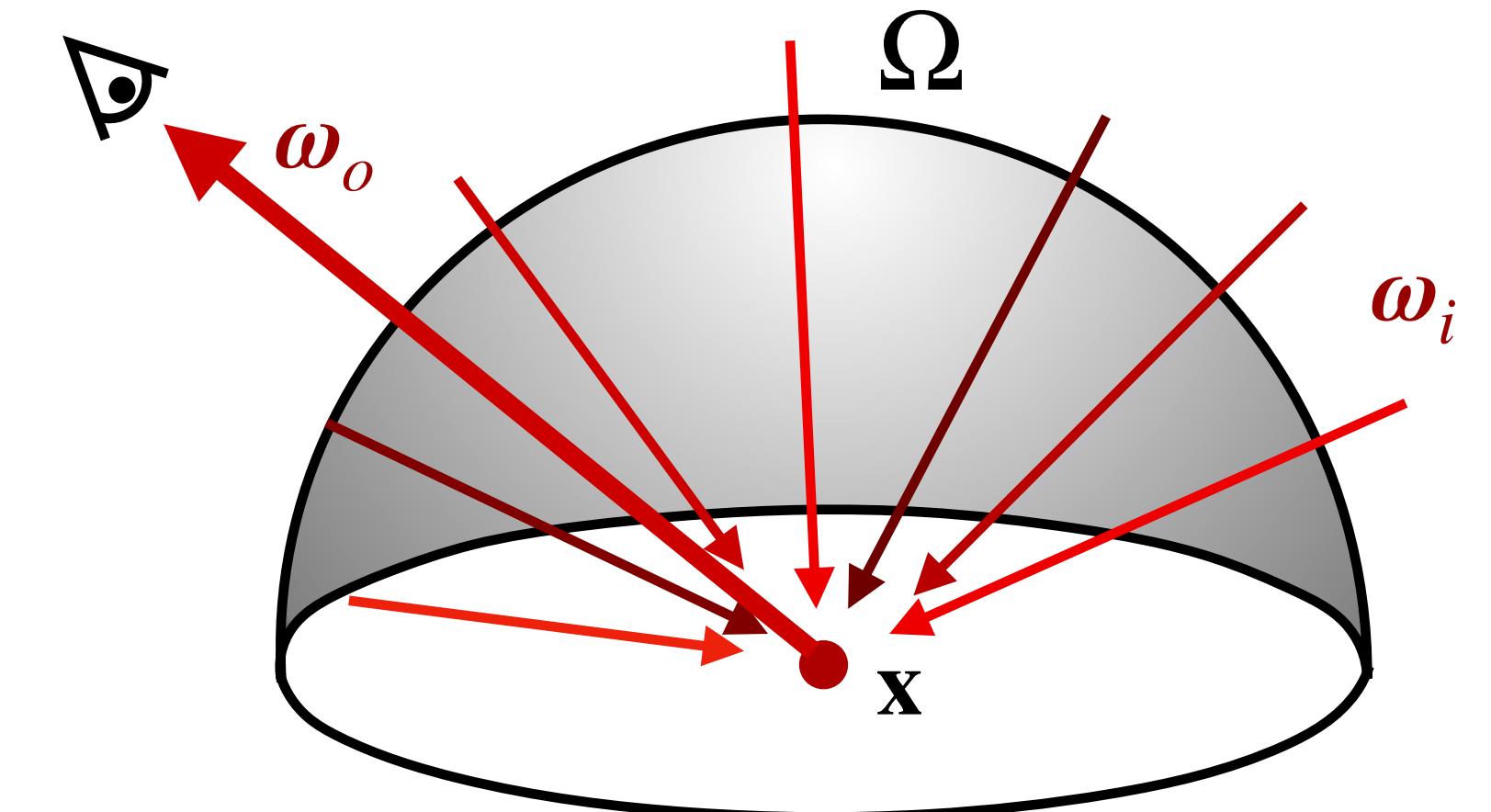
$$\int_{\omega_i \in \Omega} f(\omega_i, \omega_o) (\omega_i \cdot \mathbf{n}) d\omega_i \leq 1$$



Colors

$$L_o(\mathbf{x}, \omega_o, \lambda) = L_e(\mathbf{x}, \omega_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \omega_i, \omega_o, \lambda) L_i(\mathbf{x}, \omega_i, \lambda) (\omega_i \cdot \mathbf{n}) d\omega_i$$

$$L_o(\mathbf{x}, \omega_o, \lambda^R) = L_e(\mathbf{x}, \omega_o, \lambda^R) + \int_{\Omega} f(\mathbf{x}, \omega_i, \omega_o, \lambda^R) L_i(\mathbf{x}, \omega_i, \lambda^R) (\omega_i \cdot \mathbf{n}) d\omega_i$$



Colors

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda) L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda) (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$$

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda^R) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda^R) + \int_{\Omega} f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda^R) L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda^R) (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$$

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda^G) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda^G) + \int_{\Omega} f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda^G) L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda^G) (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$$

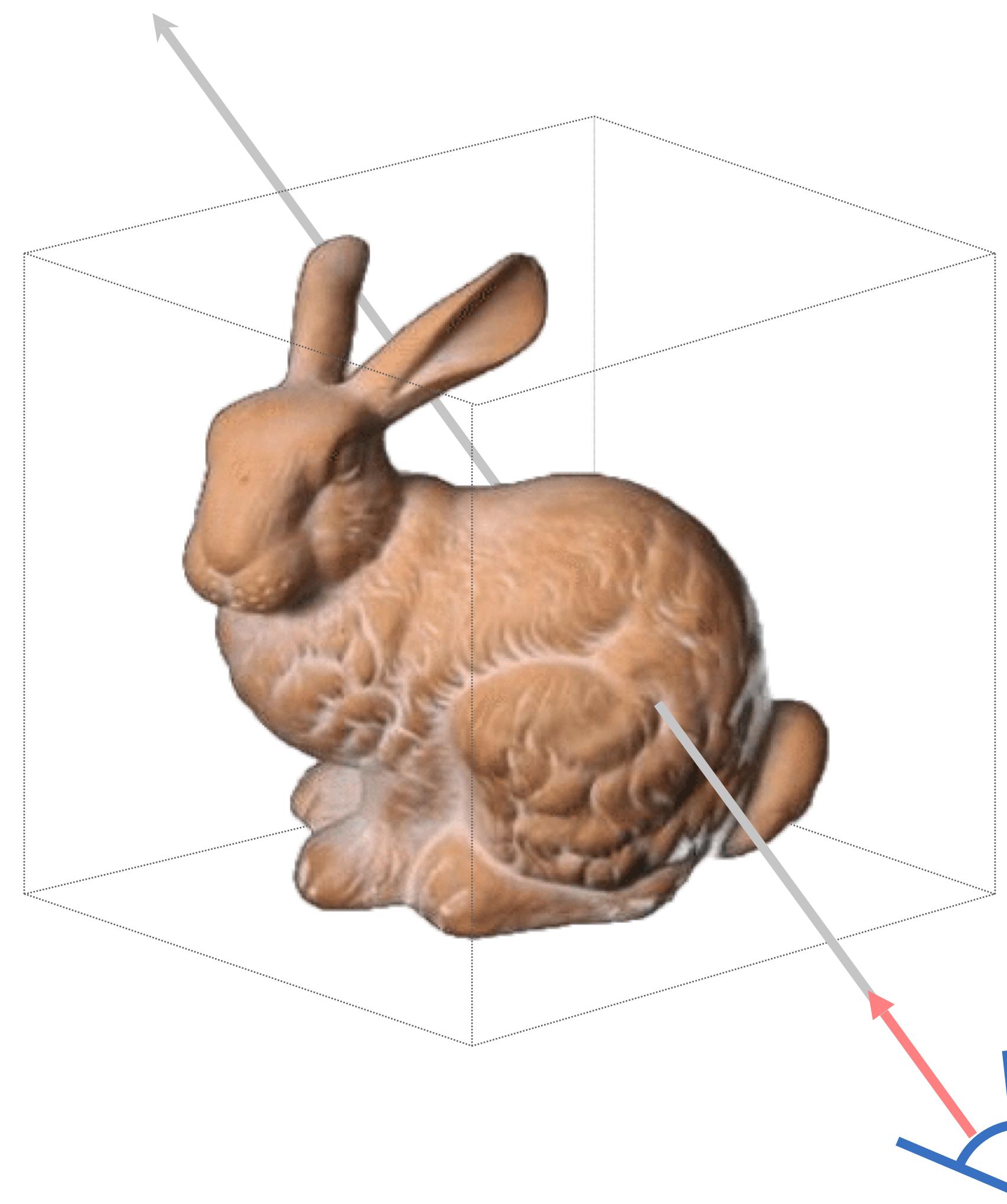
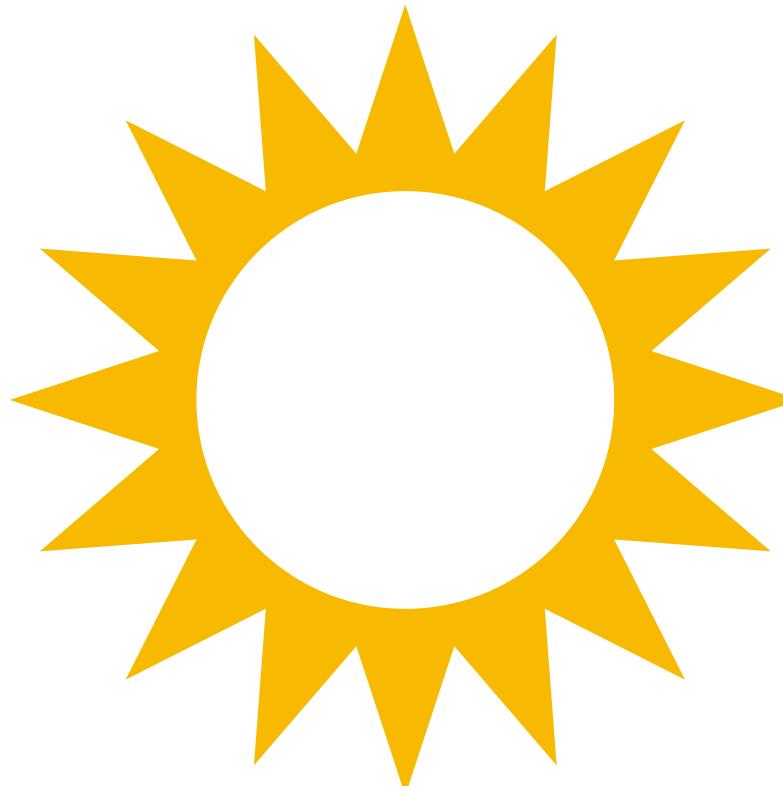
$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda^B) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda^B) + \int_{\Omega} f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda^B) L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda^B) (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$$

Colors

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o, \lambda) L_i(\mathbf{x}, \boldsymbol{\omega}_i, \lambda) (\boldsymbol{\omega}_i \cdot \mathbf{n}) d\boldsymbol{\omega}_i$$

$$L(\mathbf{x}, \boldsymbol{\omega}) : \mathbb{R}^3 \times \mathbb{S}^2 \mapsto \mathbb{R}^3$$

Ok, sure, but how do we actually render our bunny?

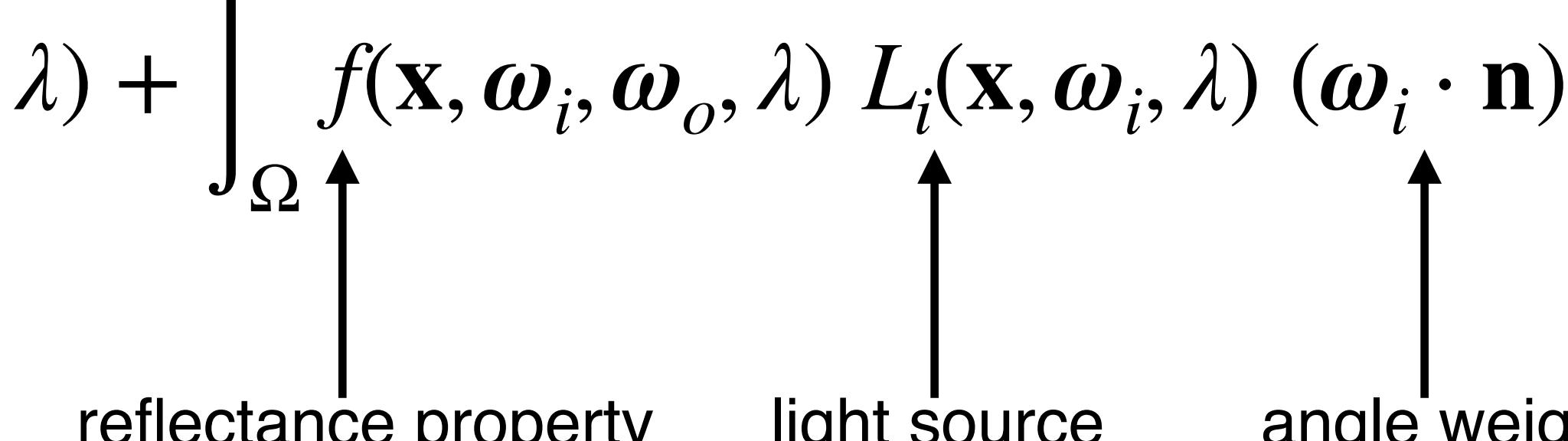


Rendering equation


$$\text{Rendering equation: } L_o(\mathbf{x}, \omega_o, \lambda) = L_e(\mathbf{x}, \omega_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \omega_i, \omega_o, \lambda) L_i(\mathbf{x}, \omega_i, \lambda) (\omega_i \cdot \mathbf{n}) d\omega_i$$

Shading

We assume surface color doesn't change even if we move our head.

$$L_o(\mathbf{x}, \omega_o, \lambda) = L_e(\mathbf{x}, \omega_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \omega_i, \omega_o, \lambda) L_i(\mathbf{x}, \omega_i, \lambda) (\omega_i \cdot \mathbf{n}) d\omega_i$$


reflectance property light source angle weight

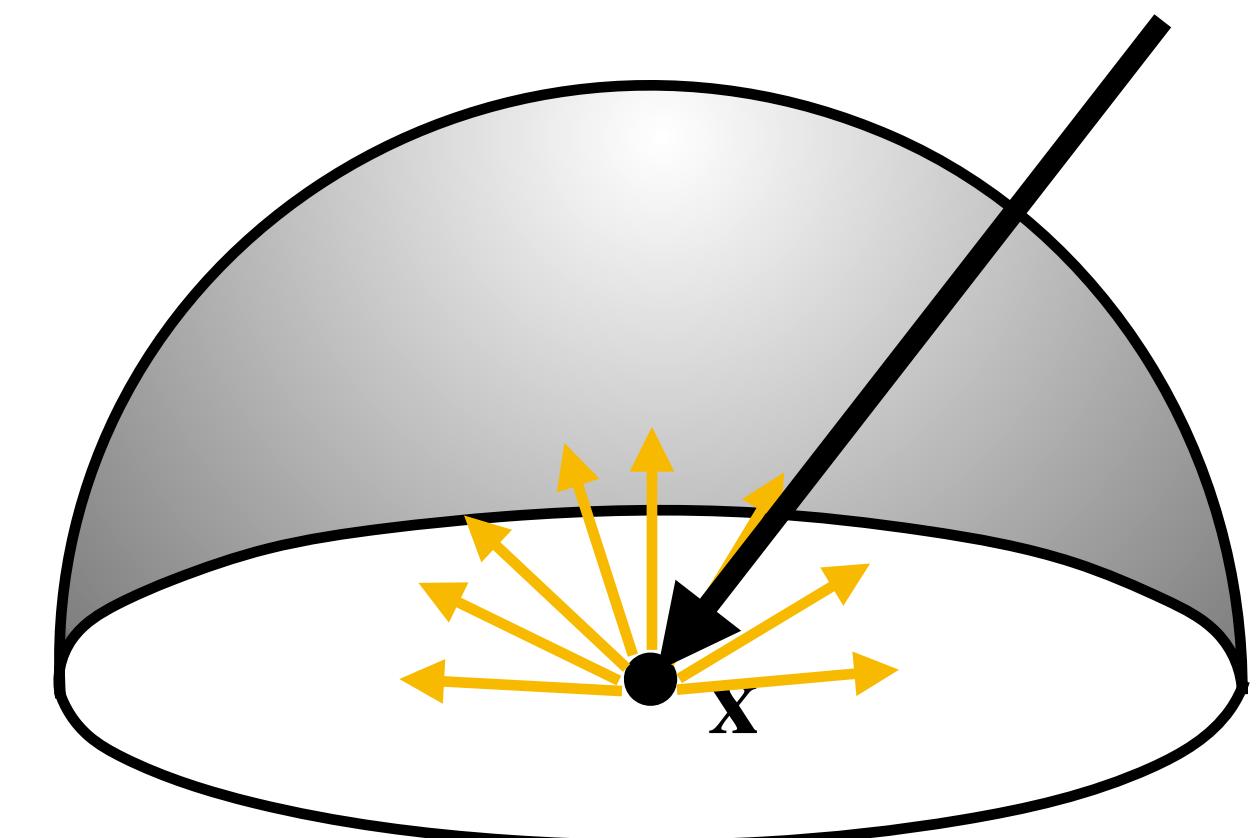
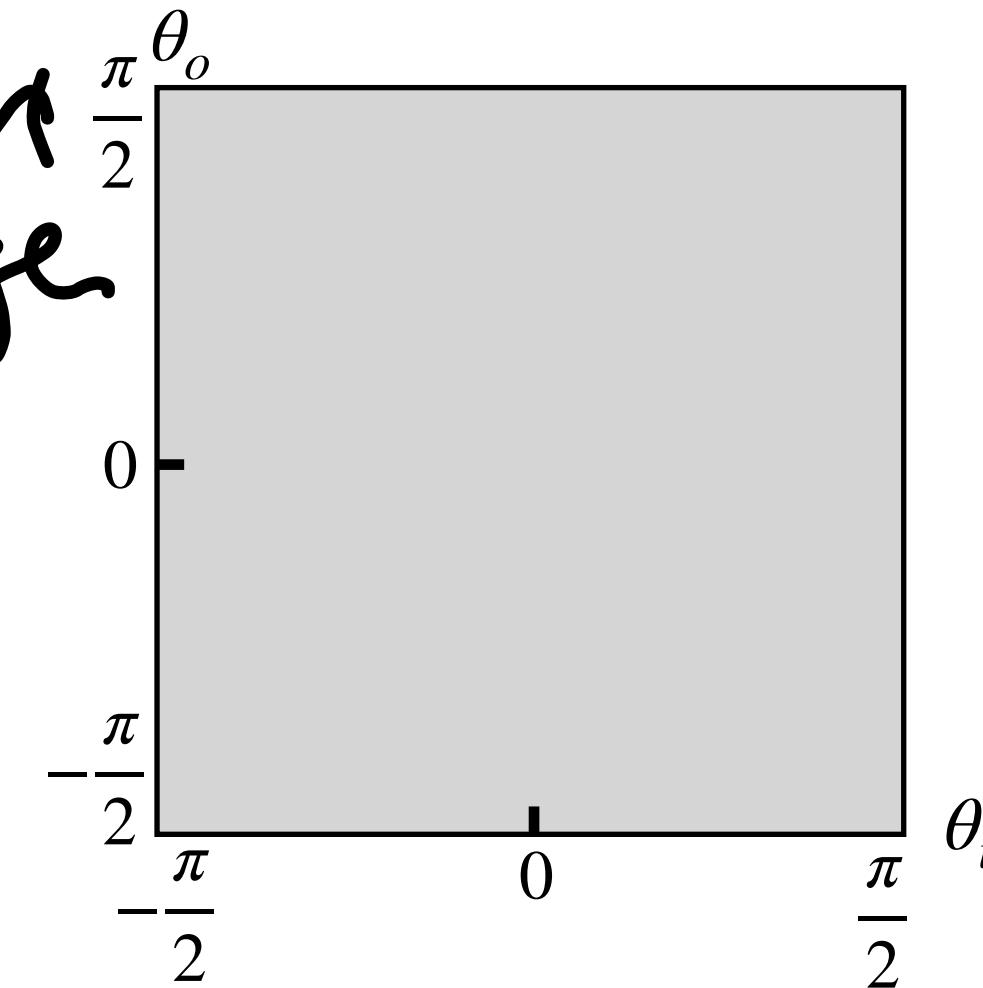
Lambertian surfaces

$$L_o(\mathbf{x}, \omega_o, \lambda) = L_e(\mathbf{x}, \omega_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \omega_i, \omega_o, \lambda) L_i(\mathbf{x}, \omega_i, \lambda) (\omega_i \cdot \mathbf{n}) d\omega_i$$

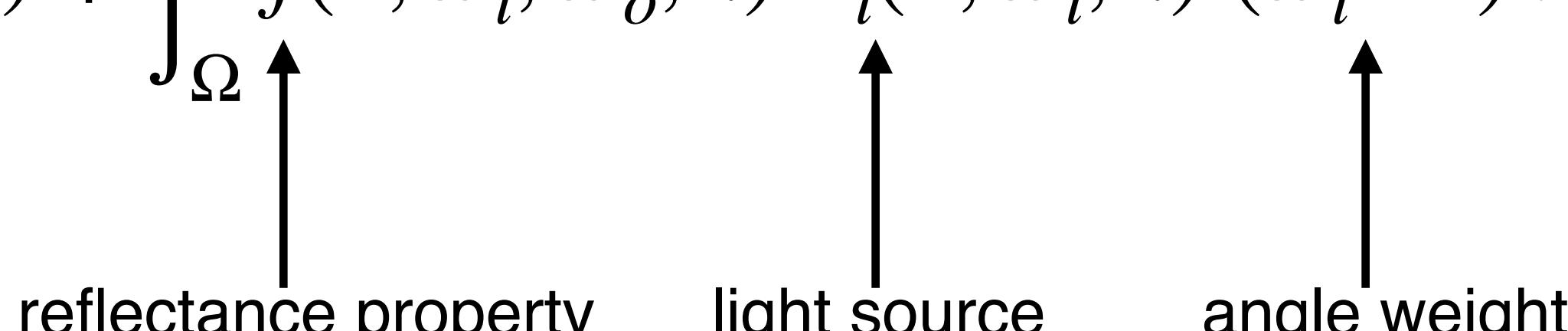
↑
reflectance property ↑
light source ↑
angle weight

What does the BRDF look like for
Lambertian surfaces?

→ Lambert's law
 $f(\mathbf{x}, \cdot, \cdot, \lambda) = \text{constant} = C_{surf}$



Lambertian surfaces

$$L_o(\mathbf{x}, \omega_o, \lambda) = L_e(\mathbf{x}, \omega_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \omega_i, \omega_o, \lambda) L_i(\mathbf{x}, \omega_i, \lambda) (\omega_i \cdot \mathbf{n}) d\omega_i$$


What does the BRDF look like for

Assumption: Lambertian surfaces

$$f(\mathbf{x}, \cdot, \cdot, \lambda) = \text{constant} = C_{surf}$$

$$L_o(\mathbf{x}, \omega_o, \lambda) = L_e(\mathbf{x}, \omega_o, \lambda) + C_{surf} \int_{\Omega} L_i(\mathbf{x}, \omega_i, \lambda) (\omega_i \cdot \mathbf{n}) d\omega_i$$

Assumption: Direct lights only

$$L_o(\mathbf{x}, \omega_o, \lambda) = L_e(\mathbf{x}, \omega_o, \lambda) + C_{surf} \sum_i L_i(\mathbf{x}, \omega_i, \lambda) (\omega_i \cdot \mathbf{n})$$

in point light source

Diffuse light

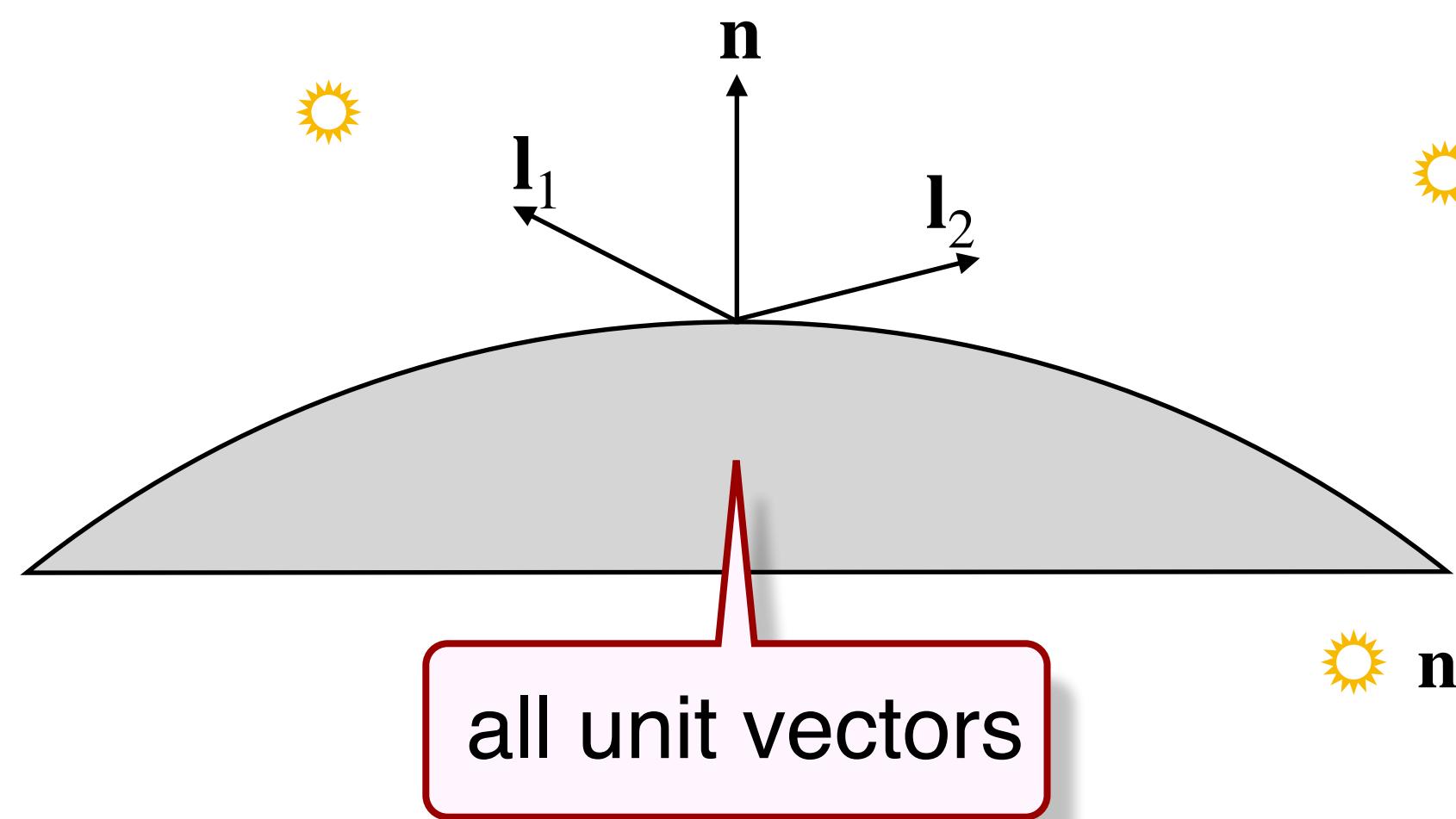
C_{surf} and C_{l_i} are 3x1 vectors representing RGB colors
• is elementwise multiplication

$$C = C_{surf} \circ \sum_{i=1}^N C_{l_i} \max(0, \mathbf{n} \cdot \mathbf{l}_i)$$

↑ surface color ↑ light color

$C_l \max(0, \mathbf{n} \cdot \mathbf{l})$.
the angle of surface.

Light locations matter,
but eye location doesn't!



What's the perceived color if we shine a blue light on a red diffuse surface?

- purple
- red
- black

$$L_o(\mathbf{x}, \omega_o, \lambda) = L_e(\mathbf{x}, \omega_o, \lambda) + C_{surf} \sum_i L_i(\mathbf{x}, \omega_i, \lambda) (\omega_i \cdot \mathbf{n})$$

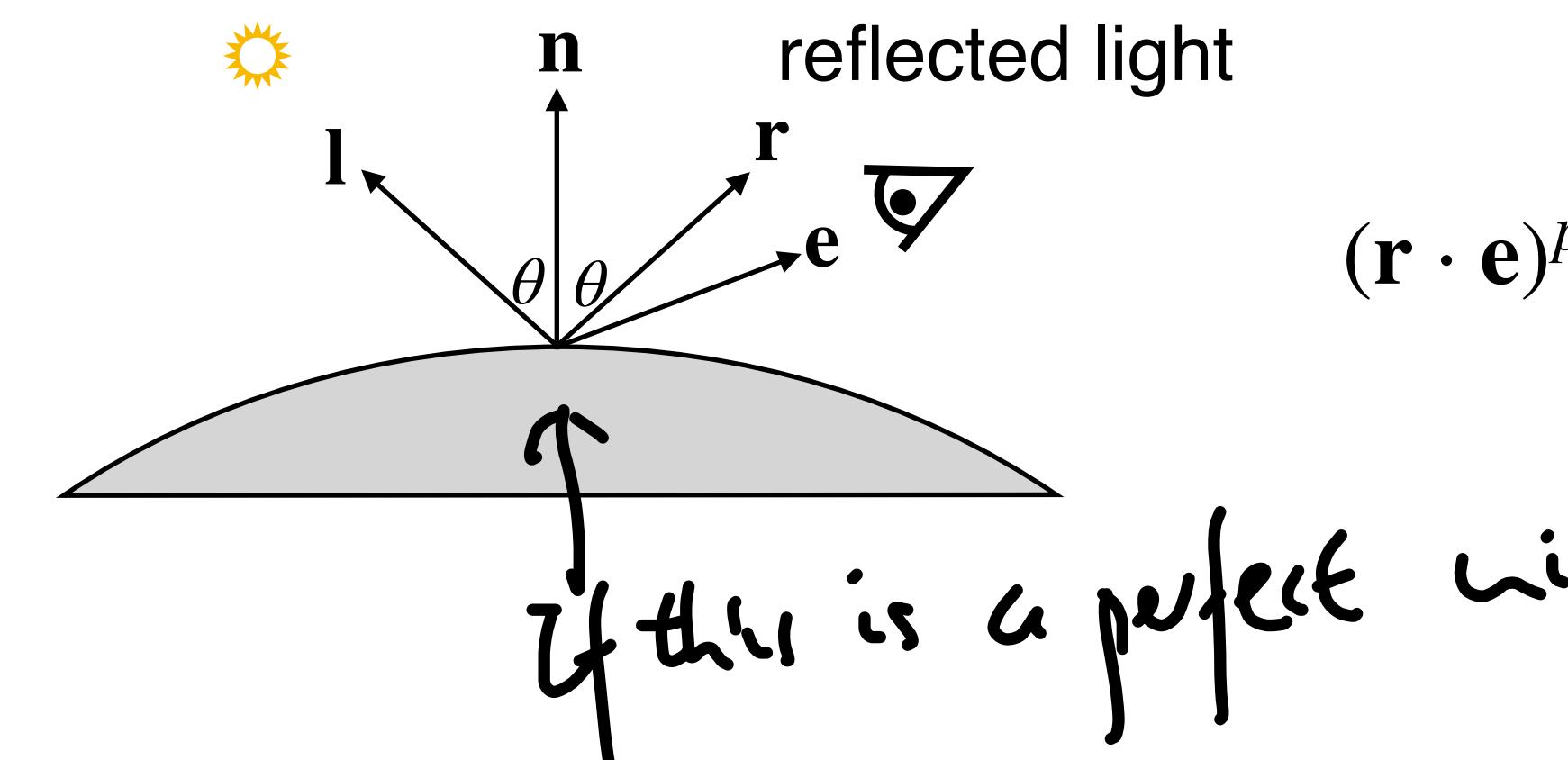
$\star \mathbf{n} \cdot \mathbf{l} < 0$

Ambient light and specular light

If we denote a point light source.
 we have a constant term. - light coming from everywhere

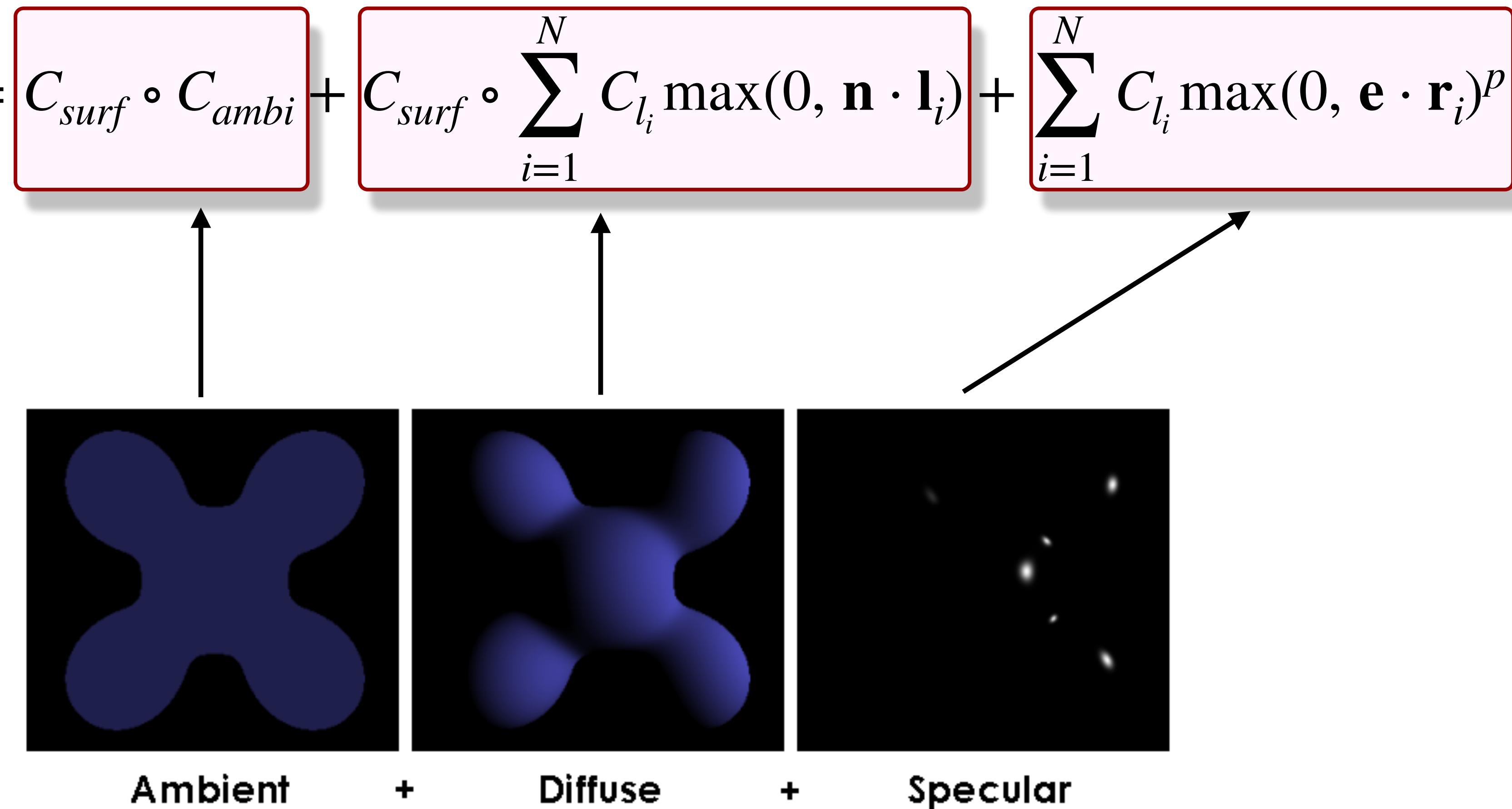
Assumption: Lambertian surfaces

Assumption: Direct lights only



Ambient term: approximates the average color of all surfaces in the scene to make up for indirect lights from every direction.

Phong illumination model



Apply shading

- Where to apply shading equation?

- Per polygon (Flat Shading): one normal

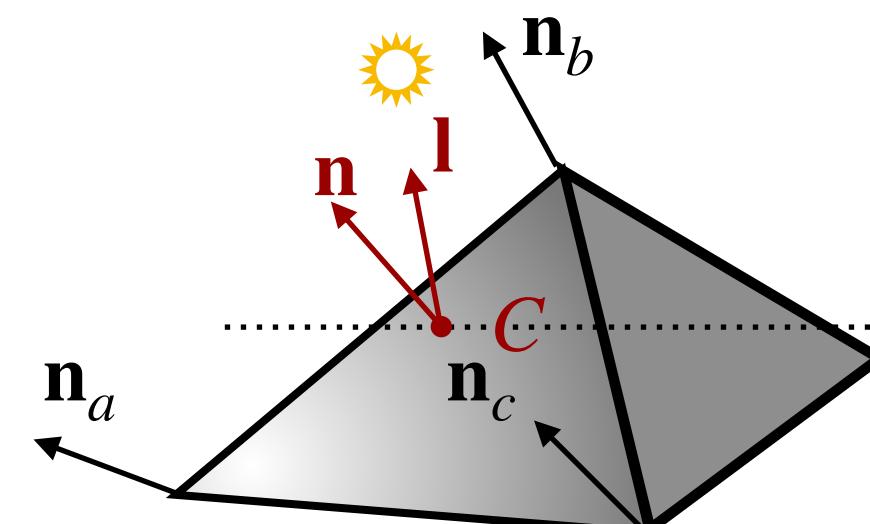
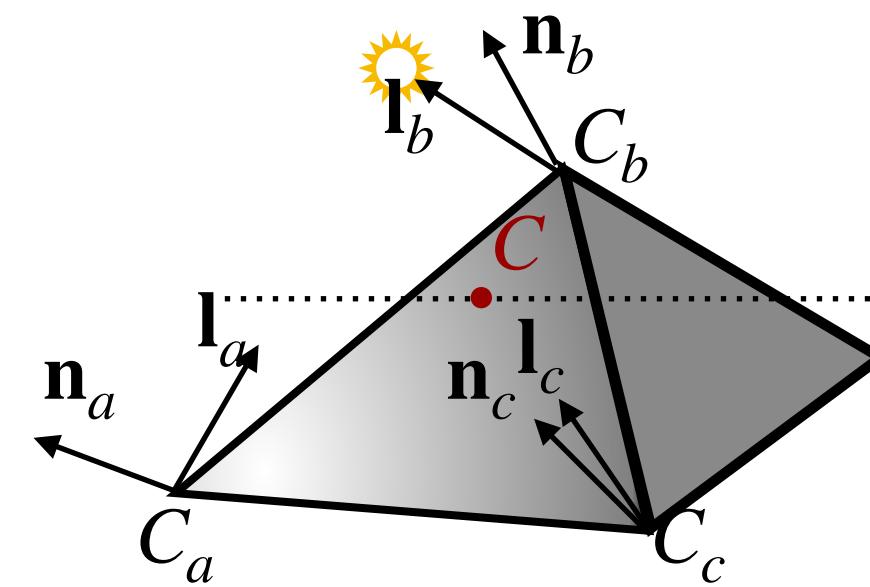
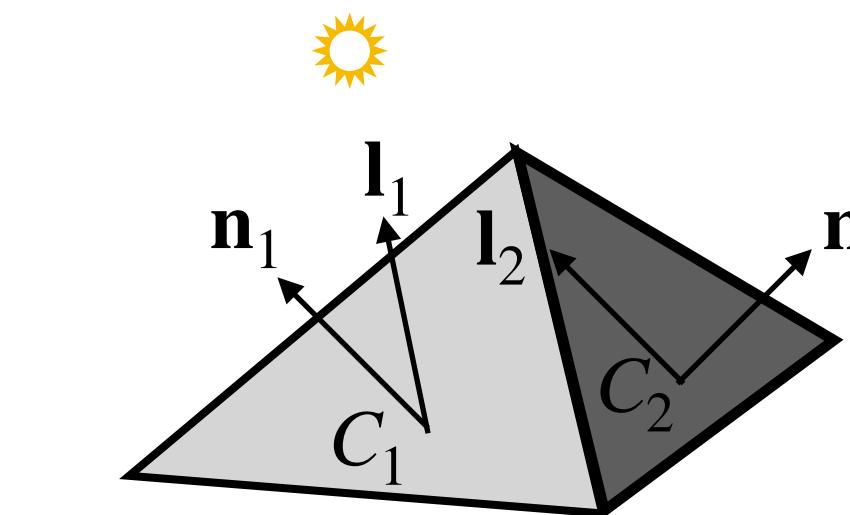
The process of computing the light vector per triangle.
the mapping from scene geometry to pixels

- Per vertex (Phong Interpolation):

compute color at vertices of meshes and interpolate color during **rasterization**.

- Per pixel (Phong interpolation):

interpolate normals of vertices during rasterization and compute color for each interpolated normal.



if we apply rendering, every phase has one color
linear interpolation



Rendering equation

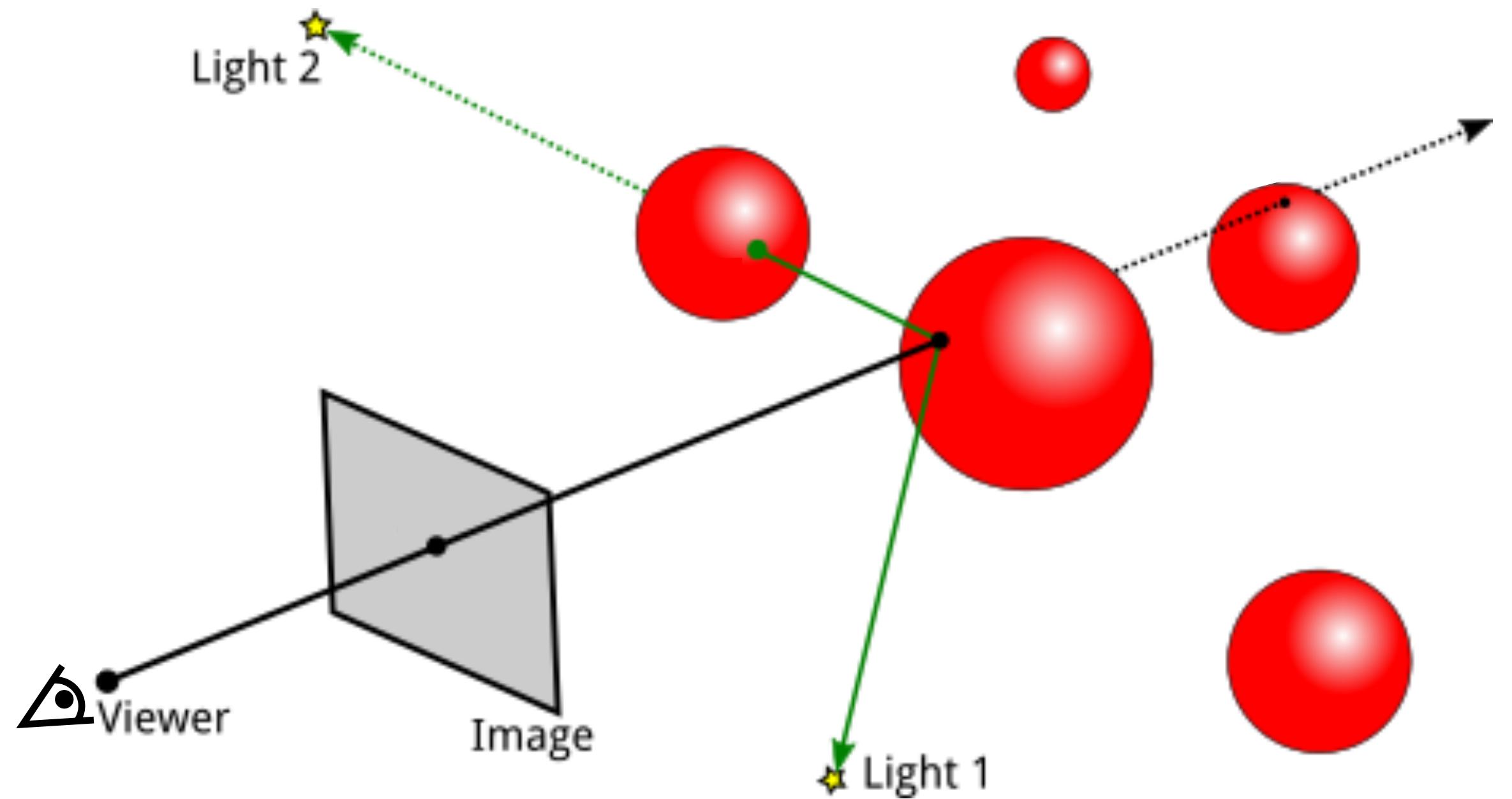


Rendering equation: $L_o(\mathbf{x}, \omega_o, \lambda) = L_e(\mathbf{x}, \omega_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \omega_i, \omega_o, \lambda) L_i(\mathbf{x}, \omega_i, \lambda) (\omega_i \cdot \mathbf{n}) d\omega_i$

Can't do shadows, refraction, indirect light and inter-reflections...

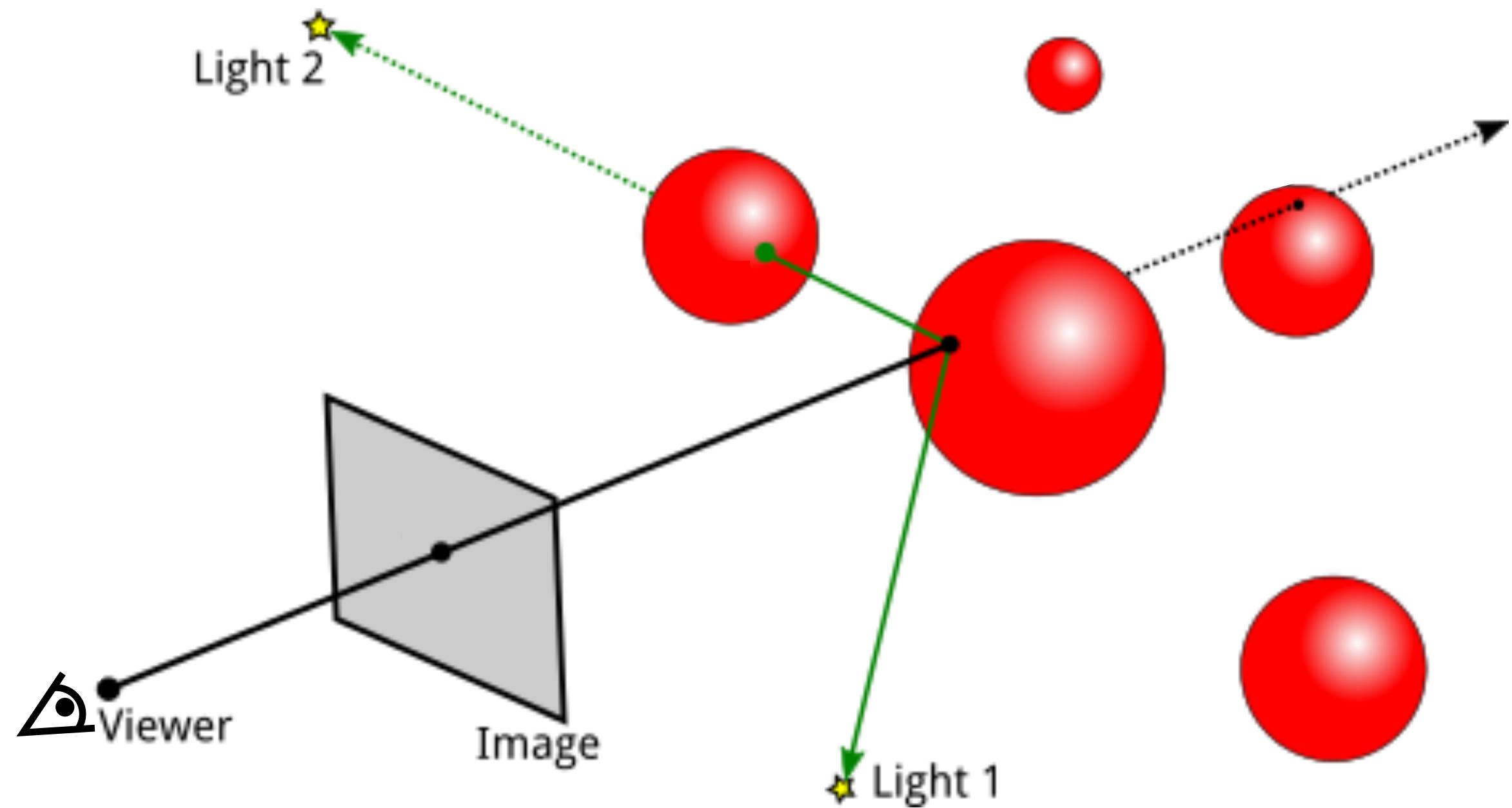
Phong model with **rasterization**: $C = C_{surf} \circ C_{ambi} + C_{surf} \circ \sum_{i=1}^N C_{l_i} \max(0, \mathbf{n} \cdot \mathbf{l}_i) + \sum_{i=1}^N C_{l_i} \max(0, \mathbf{e} \cdot \mathbf{r}_i)^p$

Ray tracing



for each pixel (x, y)
create ray R from eye through (x, y)
for each object O in scene
 if R intersects O and is closest so far
 record this intersection
 shade pixel (x, y) based on nearest intersection

Ray tracing



for each pixel (x, y)

create ray R from eye through (x, y)

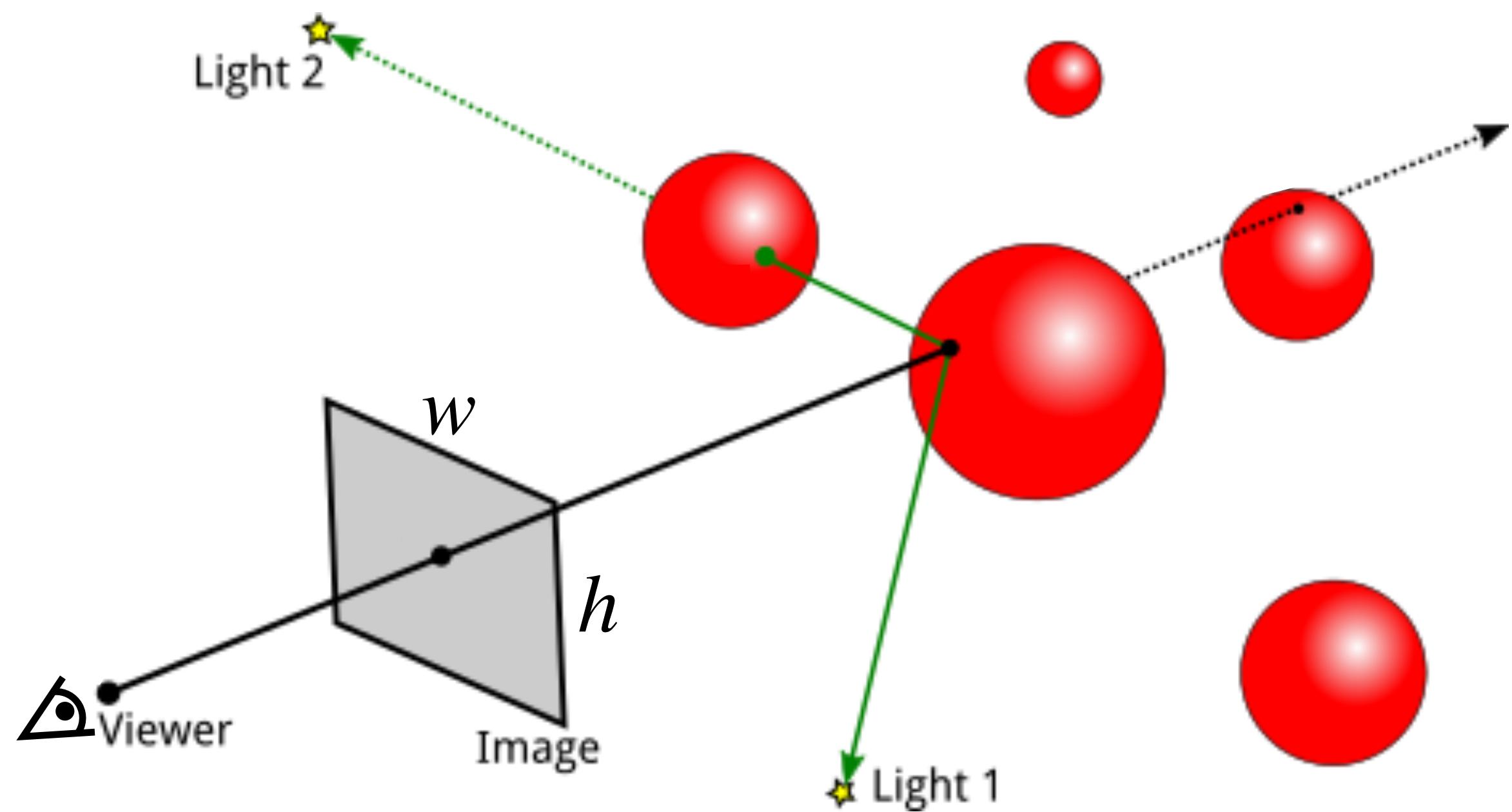
for each object O in scene

if R intersects O and is closest so far

record this intersection

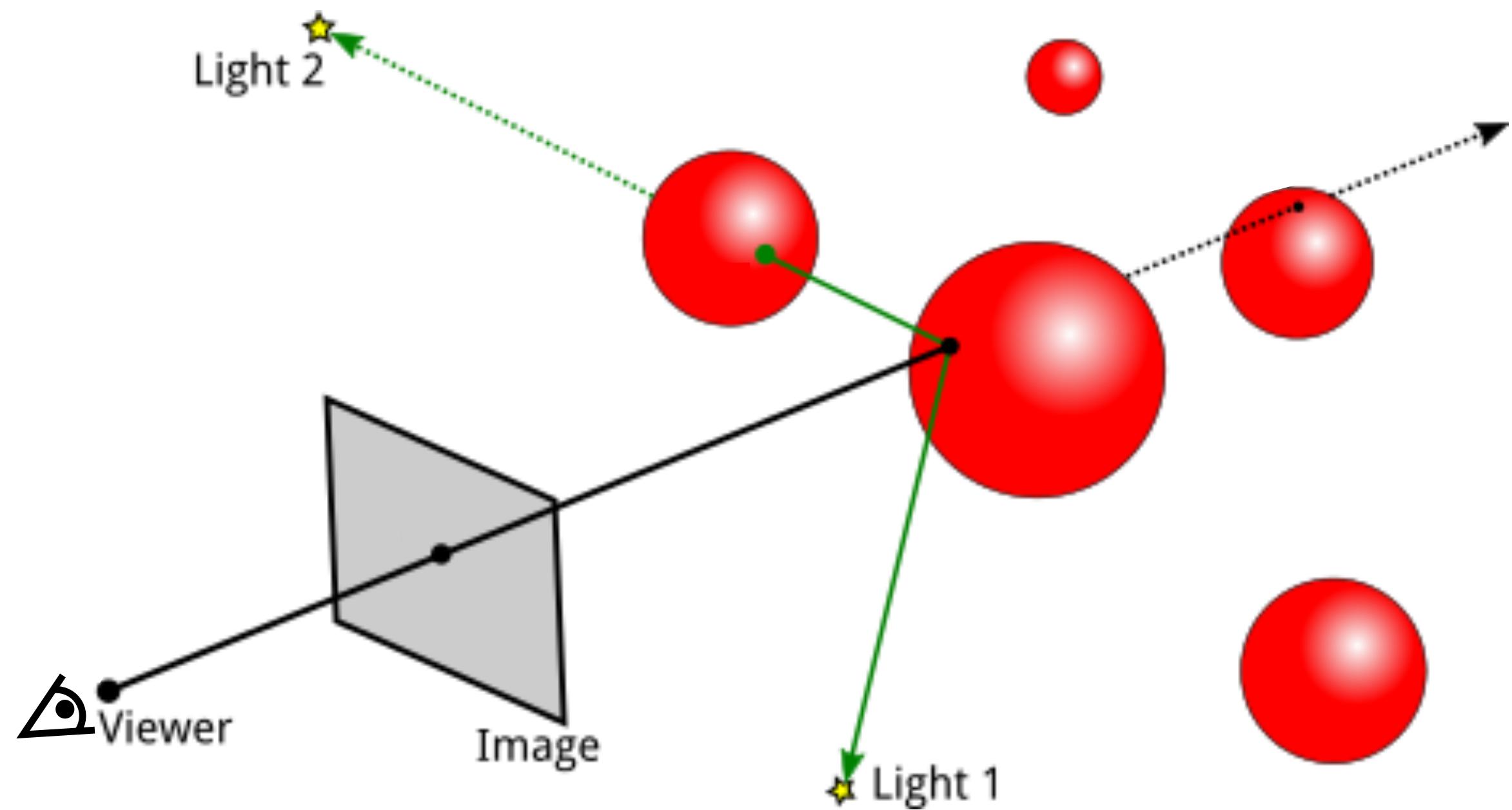
shade pixel (x, y) based on nearest intersection

Create an eye ray



$$\mathbf{X}(d) = d * \mathbf{R}_1^{C2W} \mathbf{K}_1^{-1} \tilde{\mathbf{x}}_1 + \mathbf{O}_1$$

Ray tracing



for each pixel (x, y)

create ray R from eye through (x, y)

for each object O in scene

if R intersects O and is closest so far

record this intersection

shade pixel (x, y) based on nearest intersection

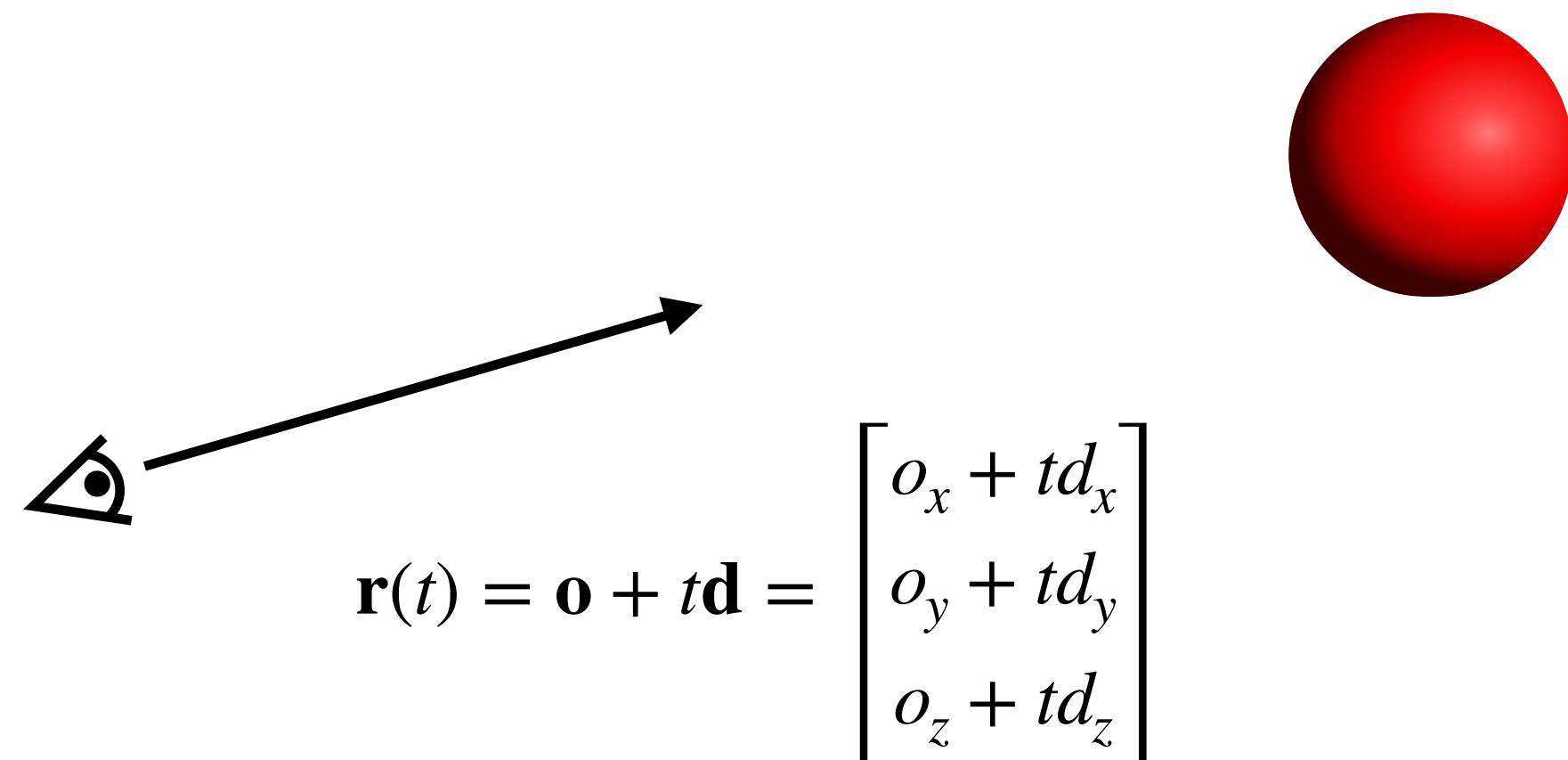
Find intersection

- How to find an intersection of a ray with a sphere?

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 - r^2 = 0$$

work out algebra to solve t

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

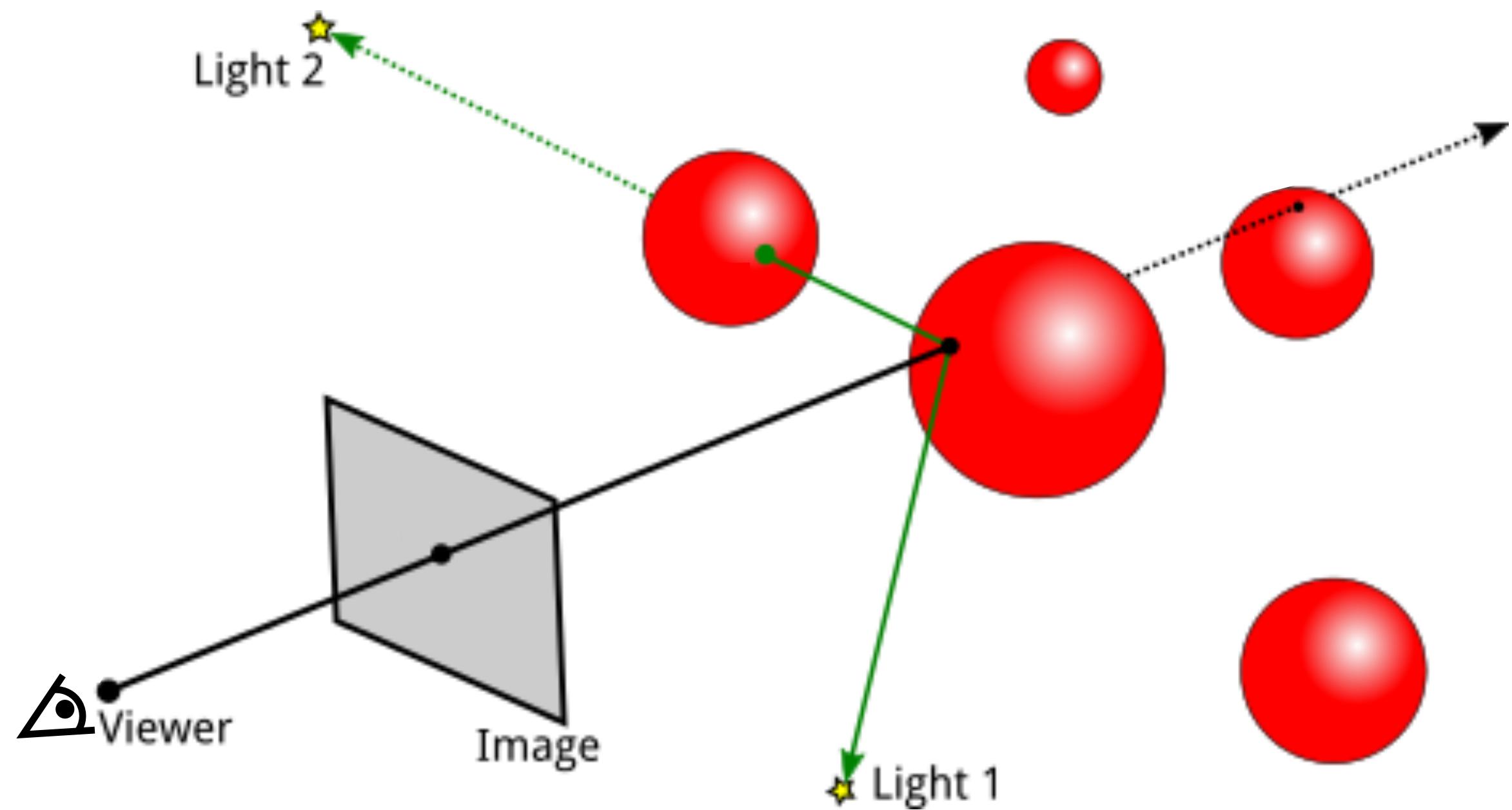


- What about a ray with a polygon?

Find intersection

- Intersect the ray with the plane containing the polygon.
- Find out if the intersection is inside of the polygon.

Ray tracing



for each pixel (x, y)

create ray R from eye through (x, y)

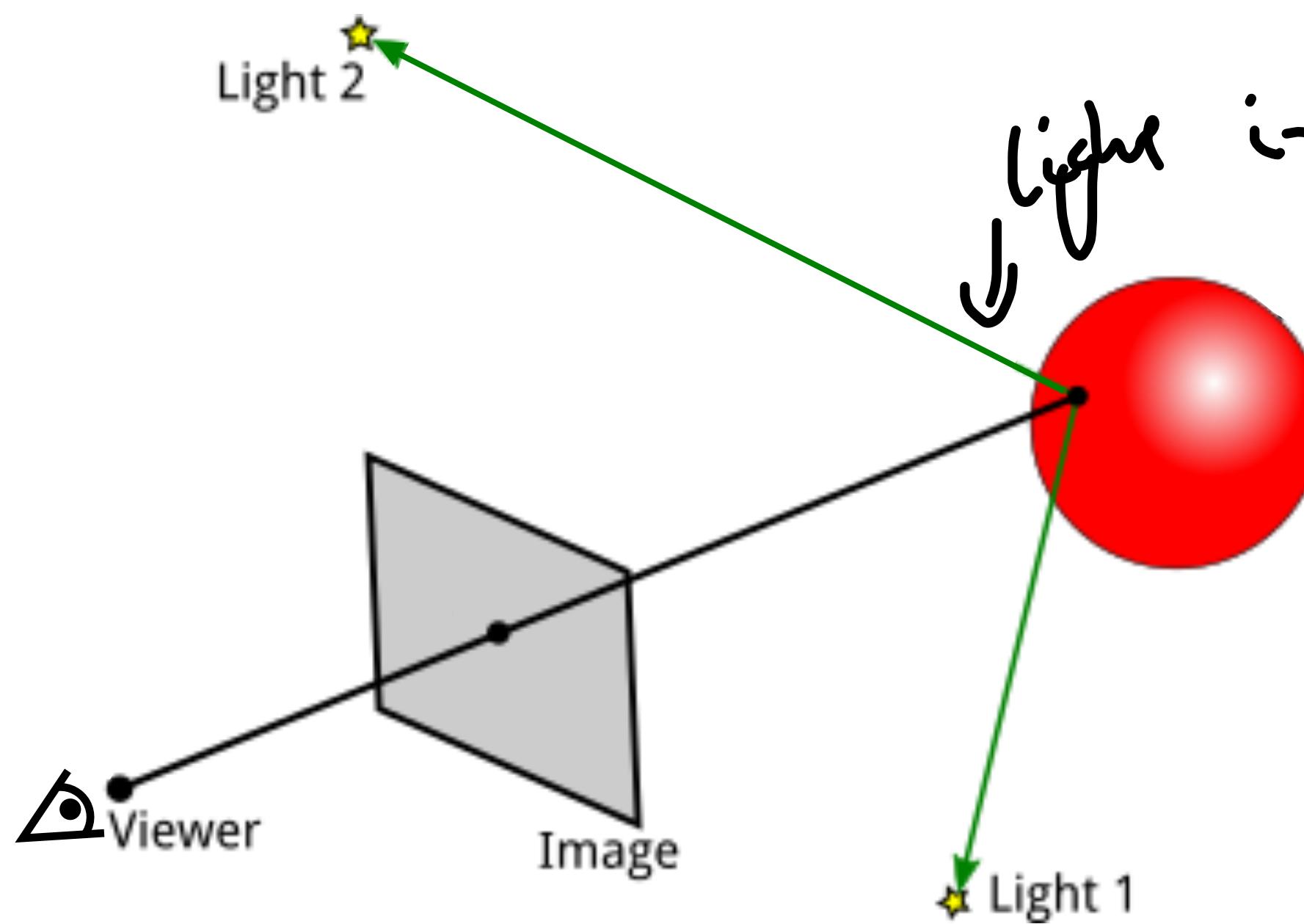
for each object O in scene

if R intersects O and is closest so far

record this intersection

shade pixel (x, y) based on nearest intersection

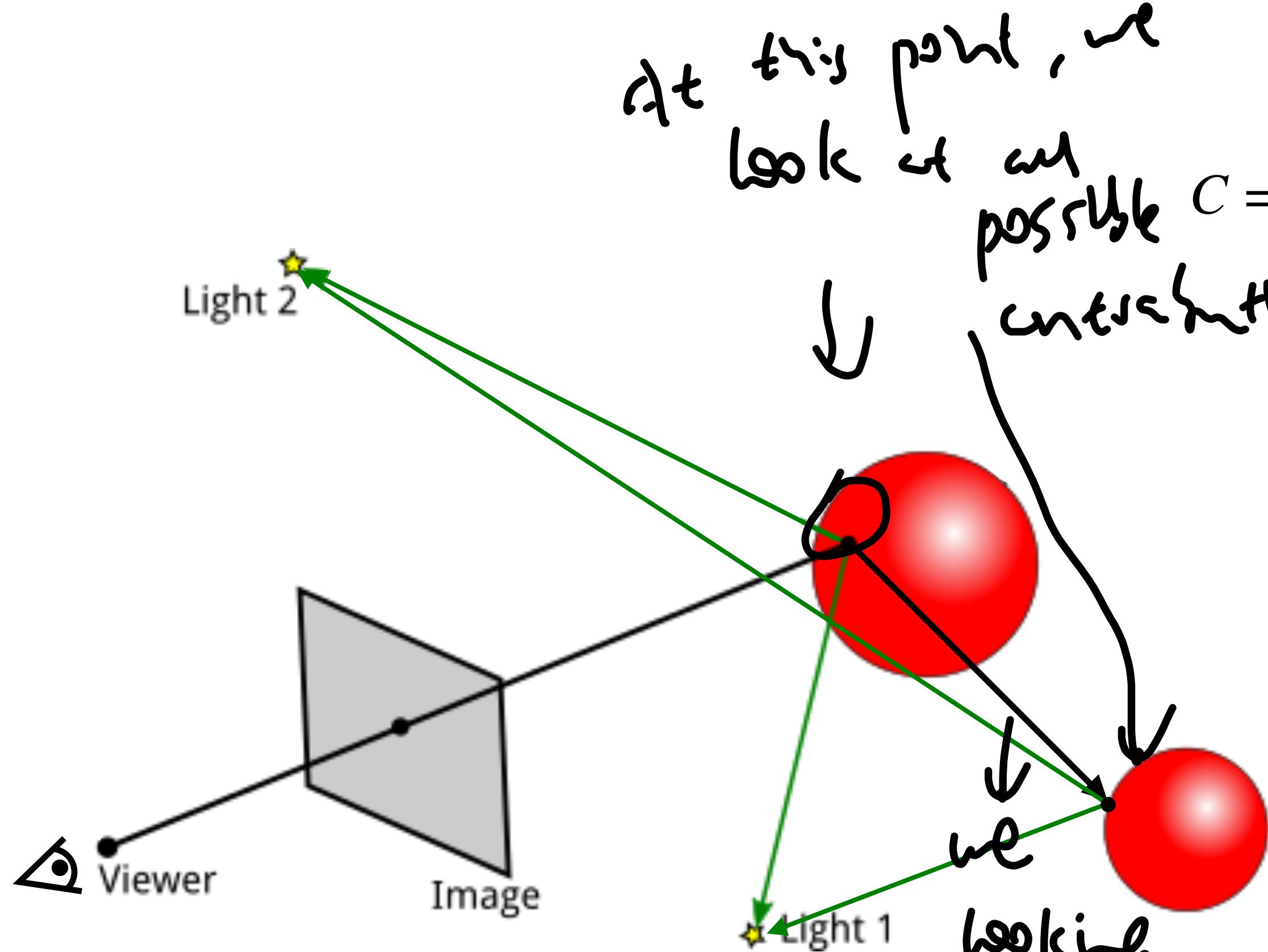
Compute the color of a pixel



$$C = C_{ambi} + \sum_{i=1}^2 C_{l_i} \circ C_{surf} (\mathbf{n} \cdot \mathbf{l}_i) + C_{spec}$$

1. first interactn: generate a reflected ray.

Reflection



Generate single, “most likely” direction

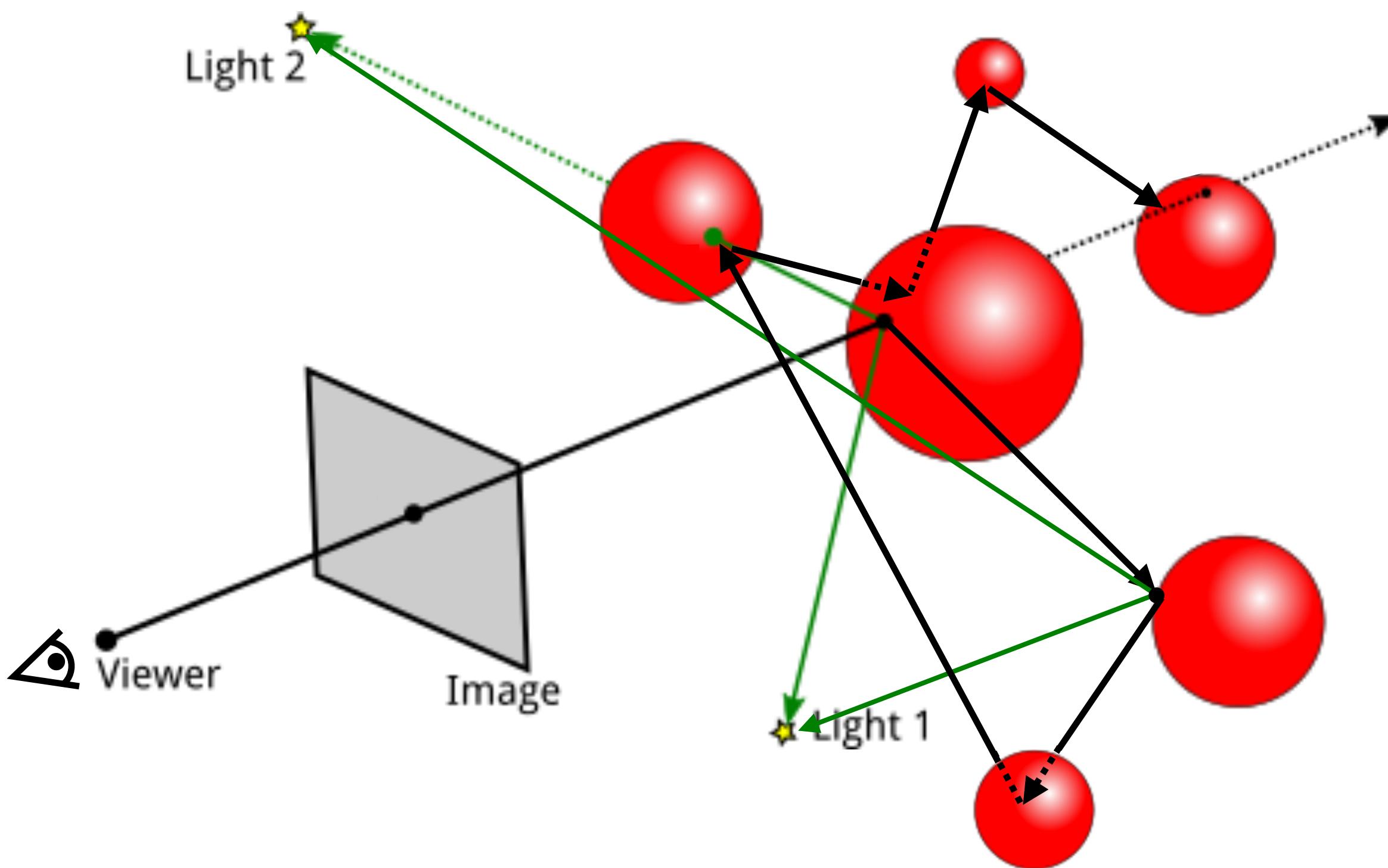
$$C = C_{ambi} + \sum_{i=1}^2 C_{l_i} \circ C_{surf} (\mathbf{n} \cdot \mathbf{l}_i) + C_{spec} + k_{refl}^{(1)} \cdot C_{refl}^{(1)}$$

$$C_{refl}^{(1)} = C_{ambi} + \sum_{i=1}^2 C_{l_i} \circ C_{surf}^{(1)} (\mathbf{n}^{(1)} \cdot \mathbf{l}_i^{(1)}) + C_{spec}^{(1)}$$

Repeat the process.

At this point, we look at all possible interactions of light.
we looking at BRDF and find the most possible ray it is going.
And repeat.

Recursive reflection



$$C = C_{ambi} + \sum_{i=1}^2 C_{l_i} \circ C_{surf} (\mathbf{n} \cdot \mathbf{l}_i) + C_{spec} + k_{refl}^{(1)} \cdot C_{refl}^{(1)}$$

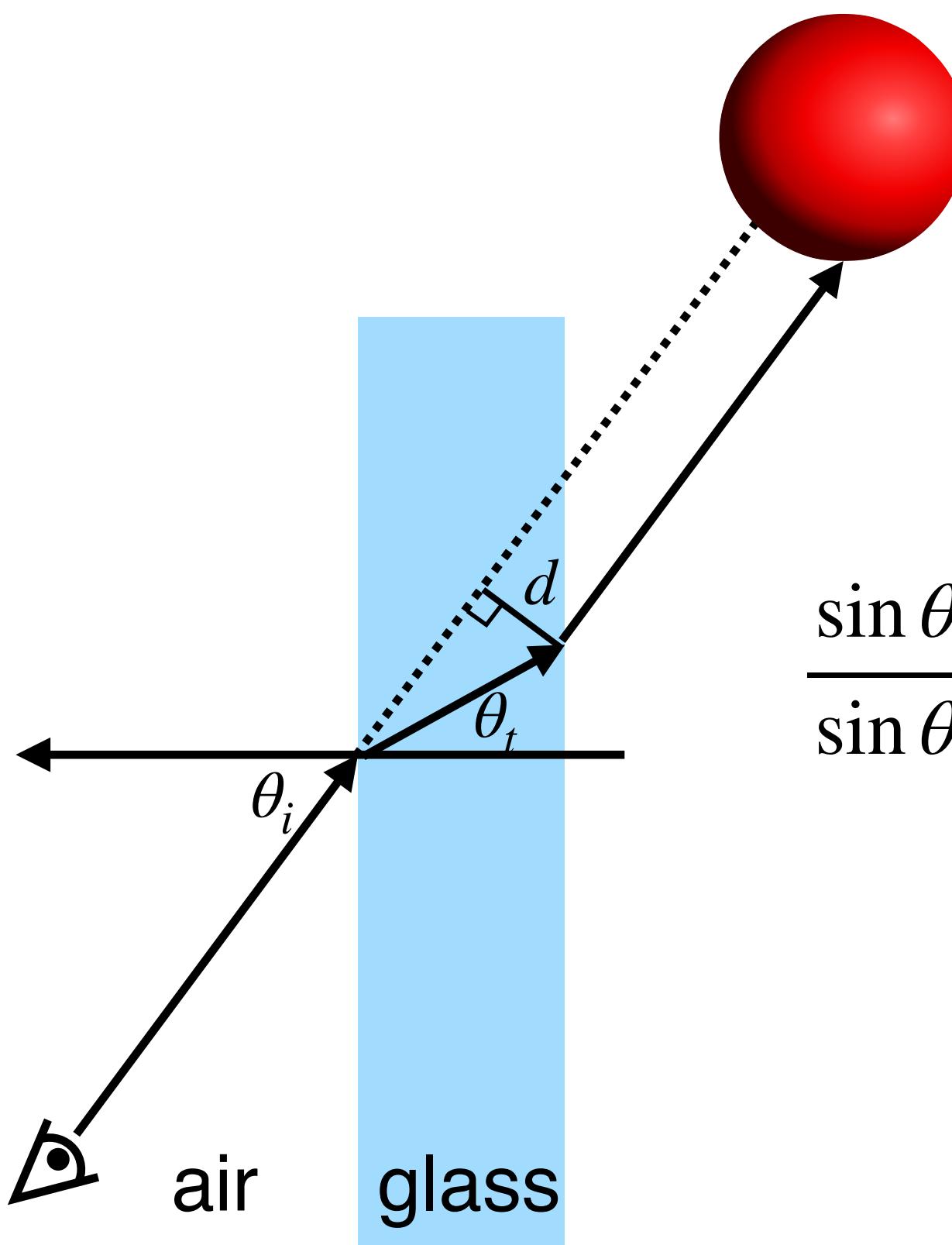
$$C_{refl}^{(1)} = C_{ambi} + \sum_{i=1}^2 C_{l_i} \circ C_{surf}^{(1)} (\mathbf{n}^{(1)} \cdot \mathbf{l}_i^{(1)}) + C_{spec}^{(1)} + k_{refl}^{(2)} \cdot C_{refl}^{(2)}$$

$$C_{refl}^{(2)} = C_{ambi} + \sum_{i=1}^2 C_{l_i} \circ C_{surf}^{(2)} (\mathbf{n}^{(2)} \cdot \mathbf{l}_i^{(2)}) + C_{spec}^{(2)} + k_{refl}^{(3)} \cdot C_{refl}^{(3)}$$

... When to stop recursion?

1. Hit a non-shiny surface.
2. When contribution to final color is small.
3. Max depth is reached.

Refraction



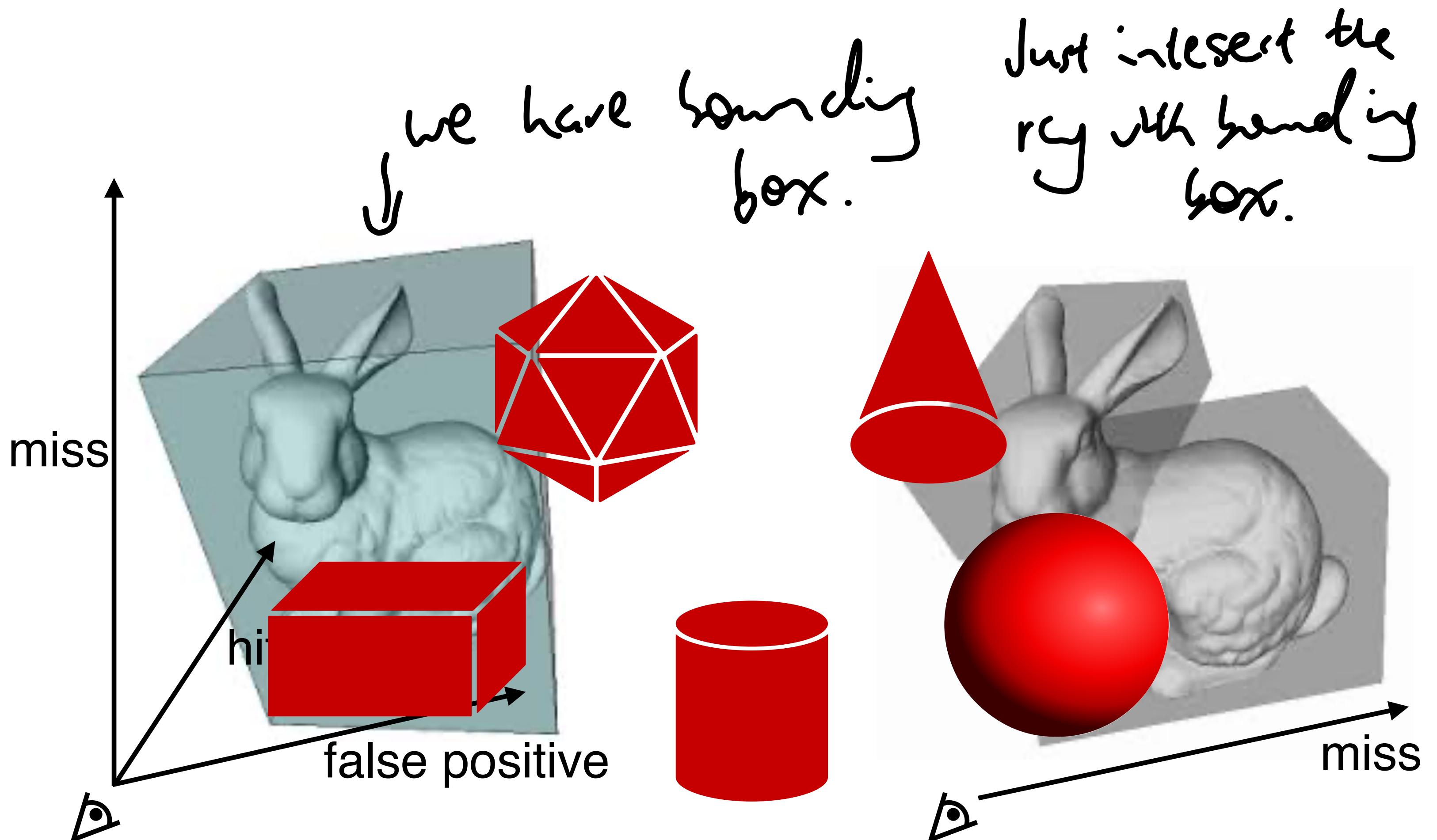
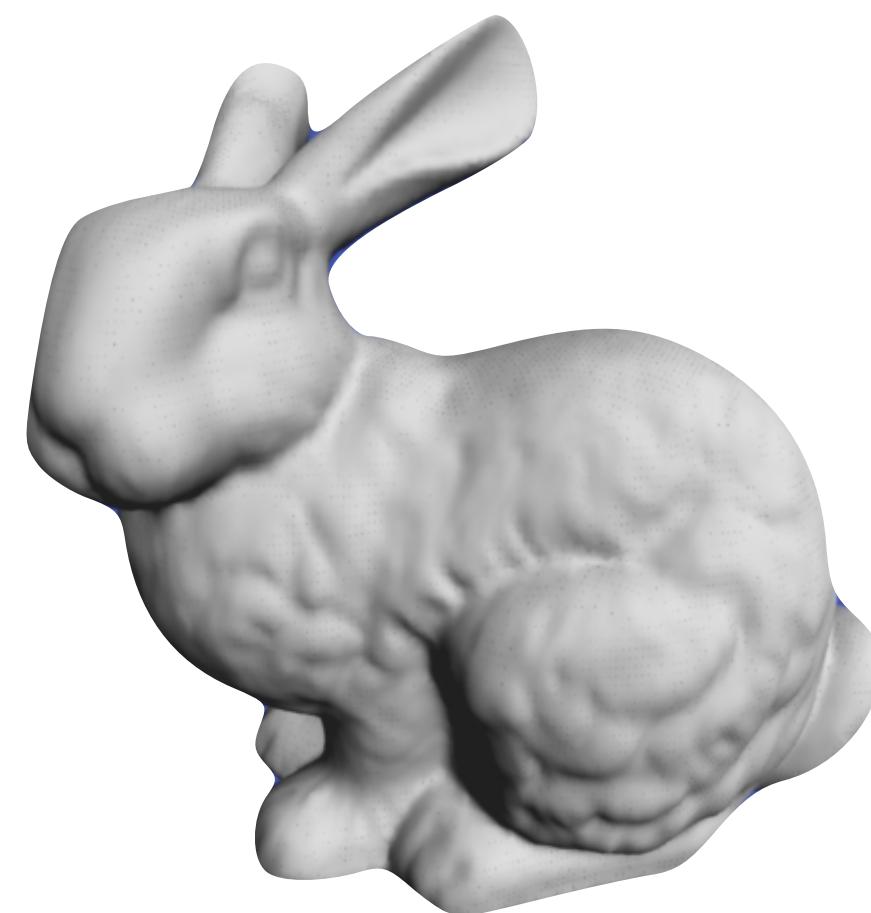
$$\frac{\sin \theta_i}{\sin \theta_t} = \frac{n_t}{n_i}$$

index of refraction (η) = $\frac{\text{speed of light thru vacuum}}{\text{speed of light thru medium}}$

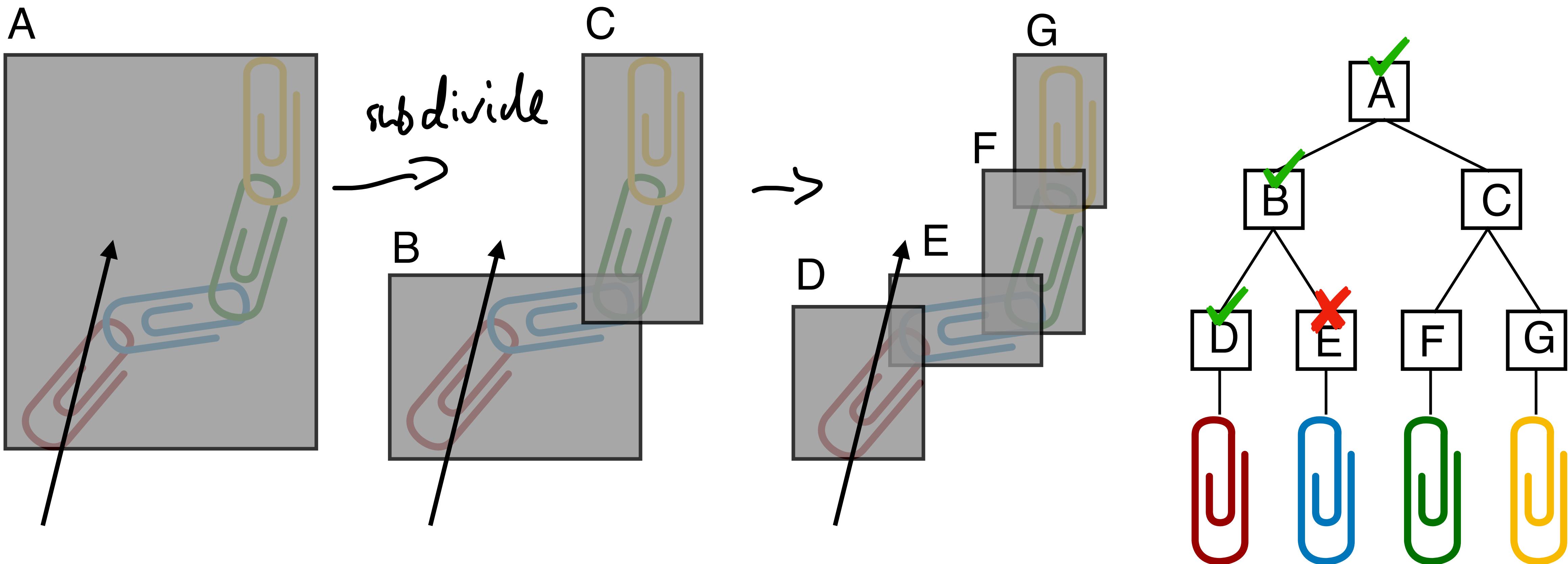


materials	η
vacuum	1
air	1.0003
water	1.33
ice	1.309
diamond	2.47

Ray tracing acceleration

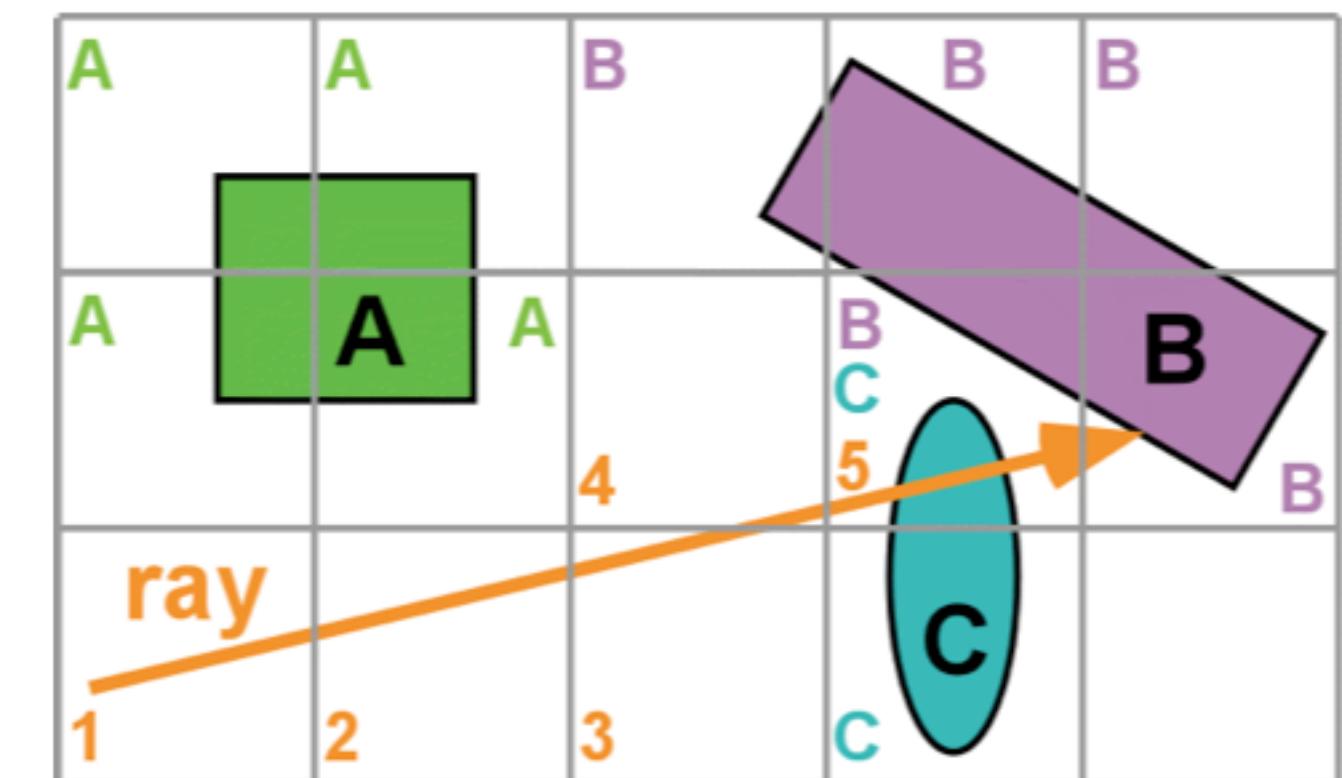


Bounding hierarchy

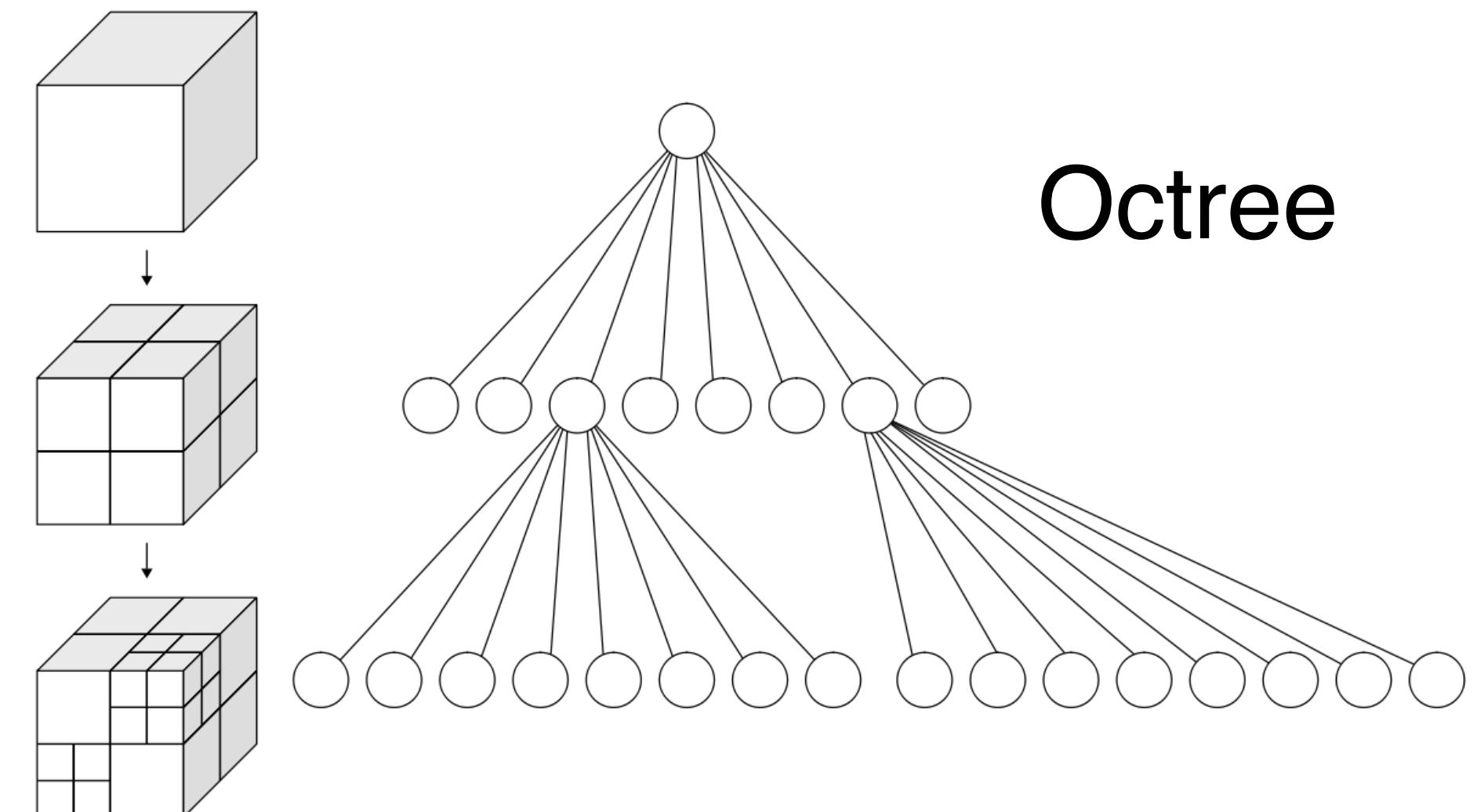


Create bounding hierarchy

- Use a tree-like structure to organize polygons into bounding hierarchy.
- For objects with similar size, **Spatial Partition** (or **Grids**) method works well.
 - Build an object list for each cell.
 - Shoot a ray thru the grid and check the object list of each cell the ray touches.
- For varying sized objects, build an **Octree** to store the objects in a binary hierarchy.

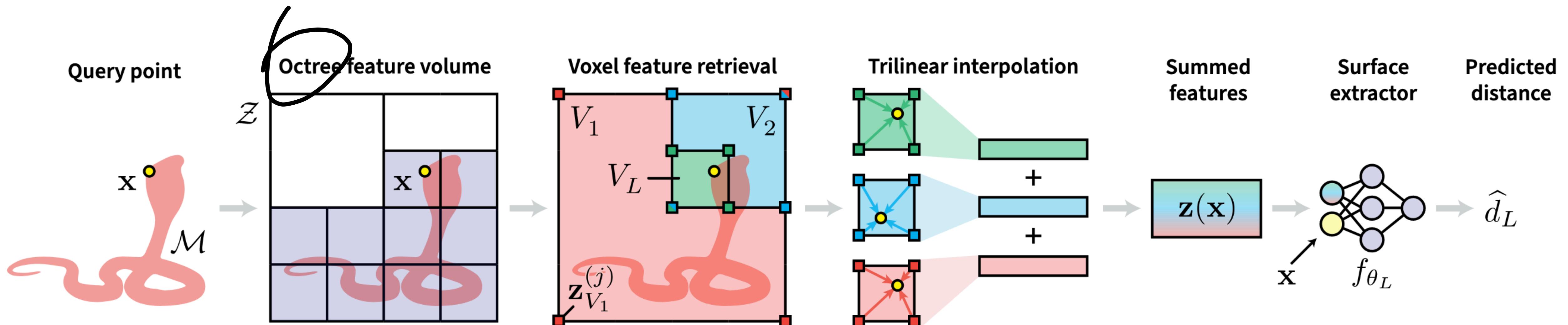


Grids

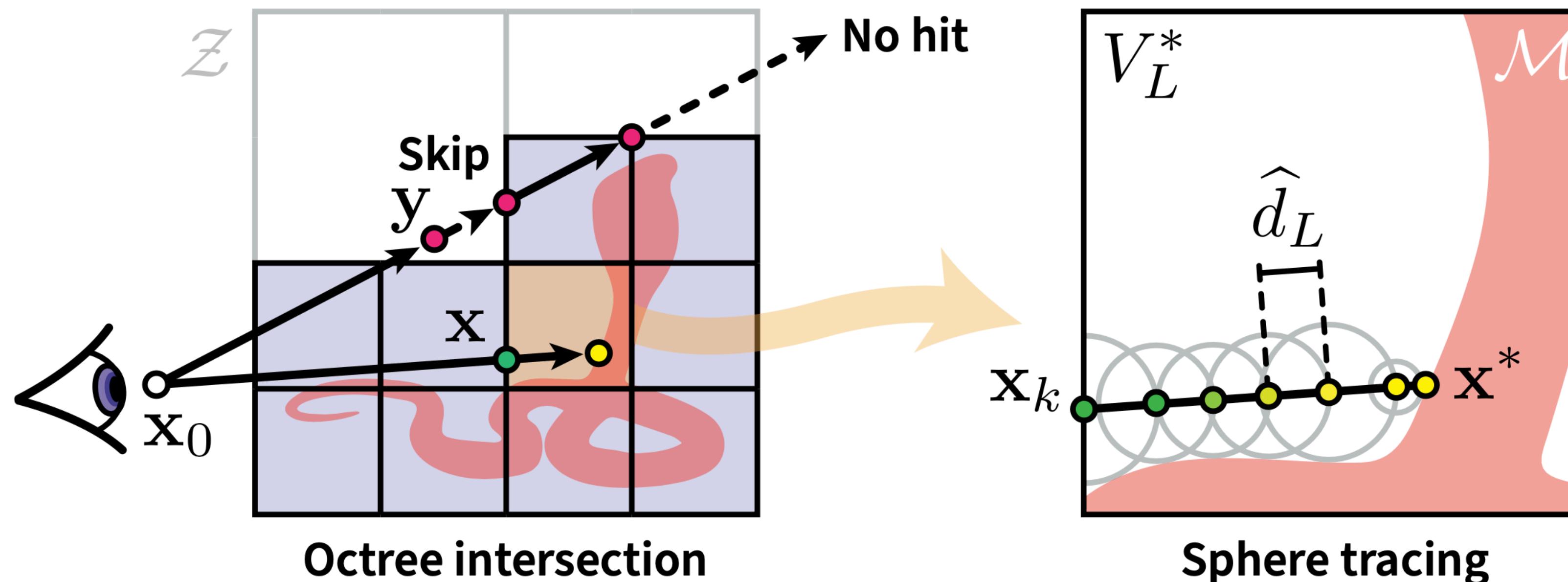


Octree

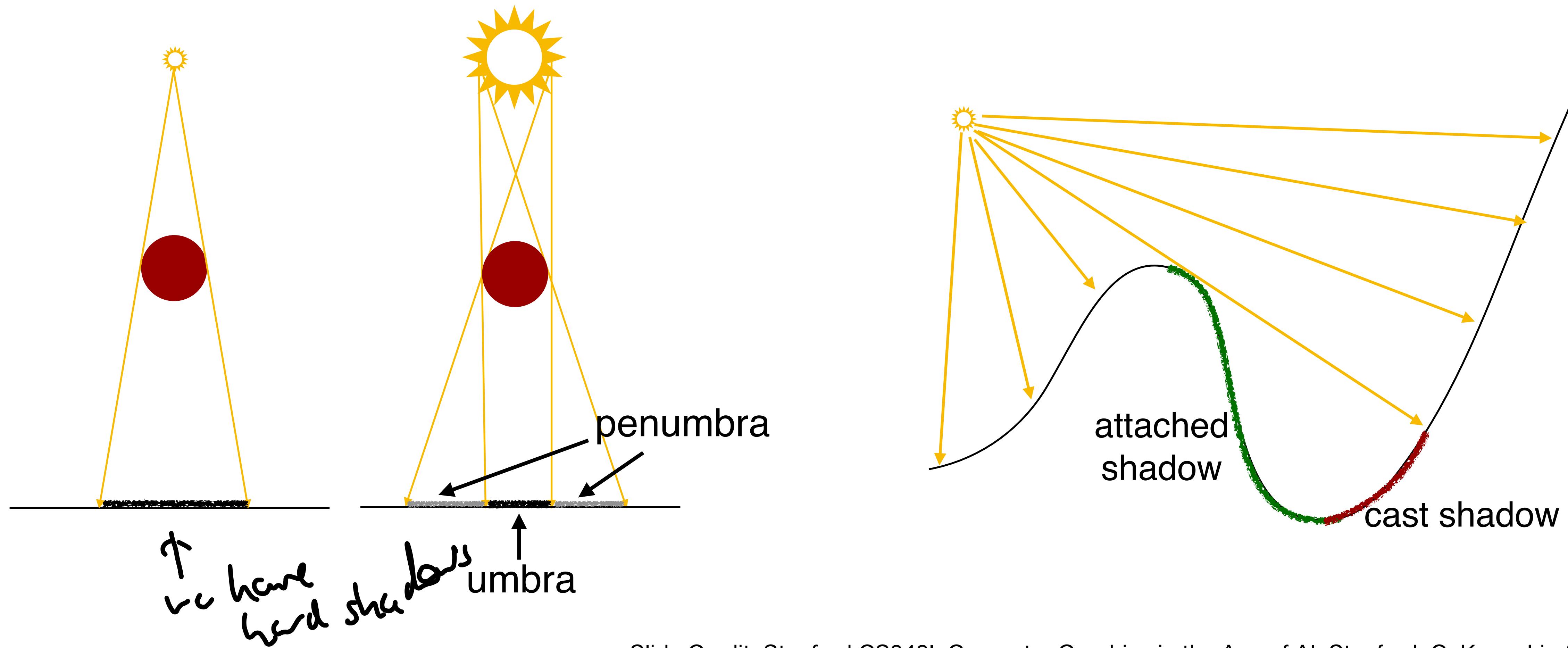
Neural Geometric Level of Detail



Neural Geometric Level of Detail

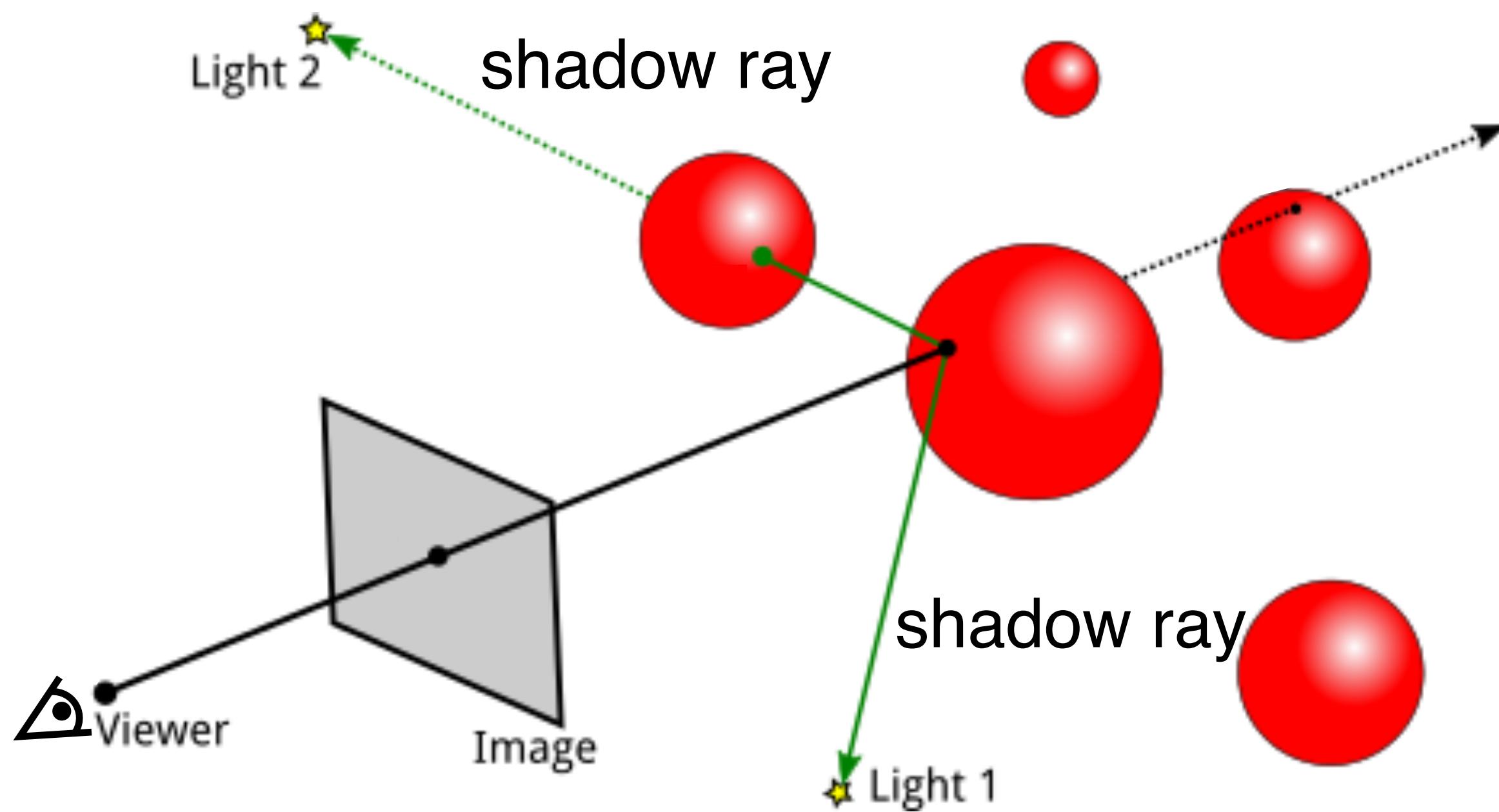


Shadows



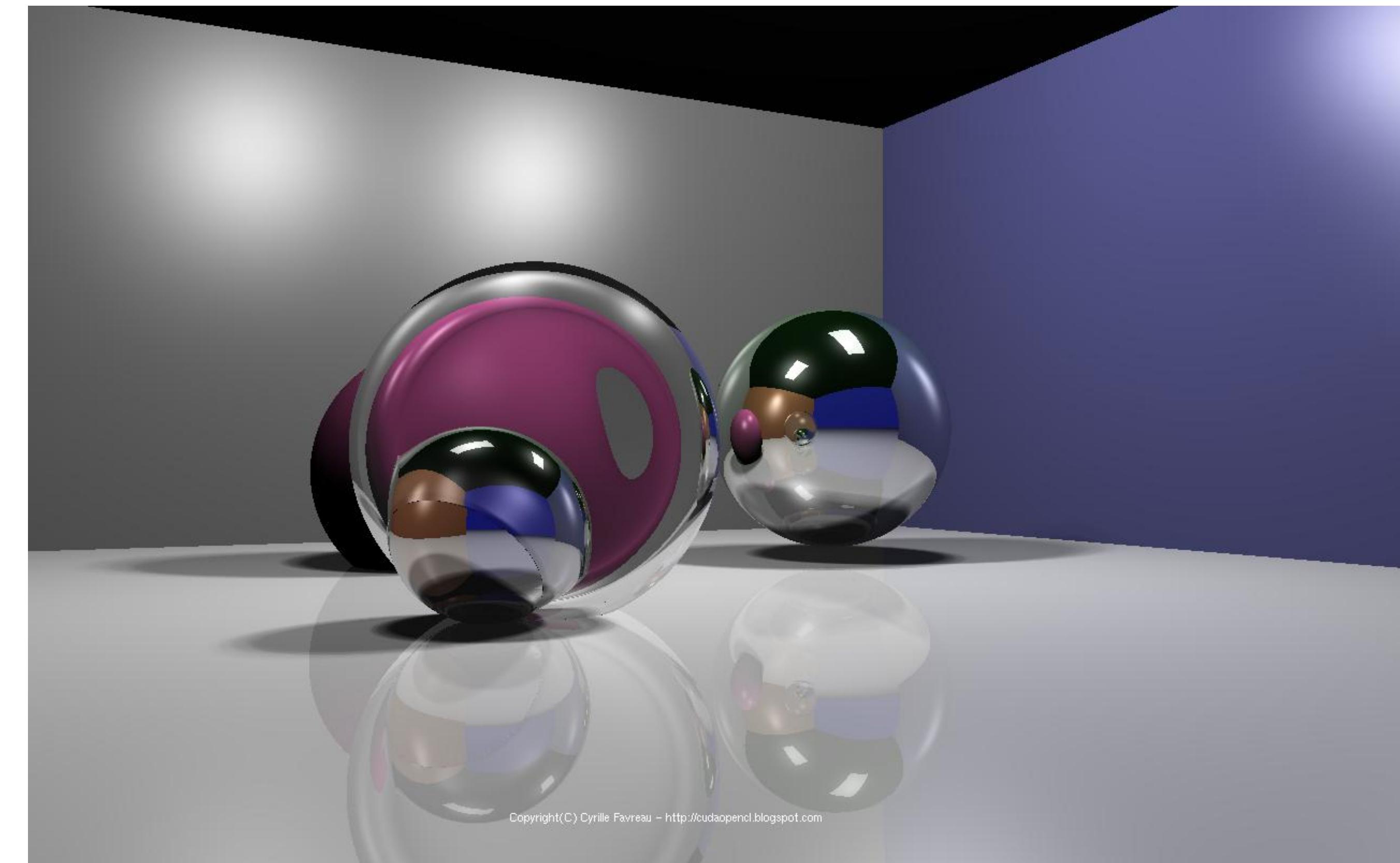
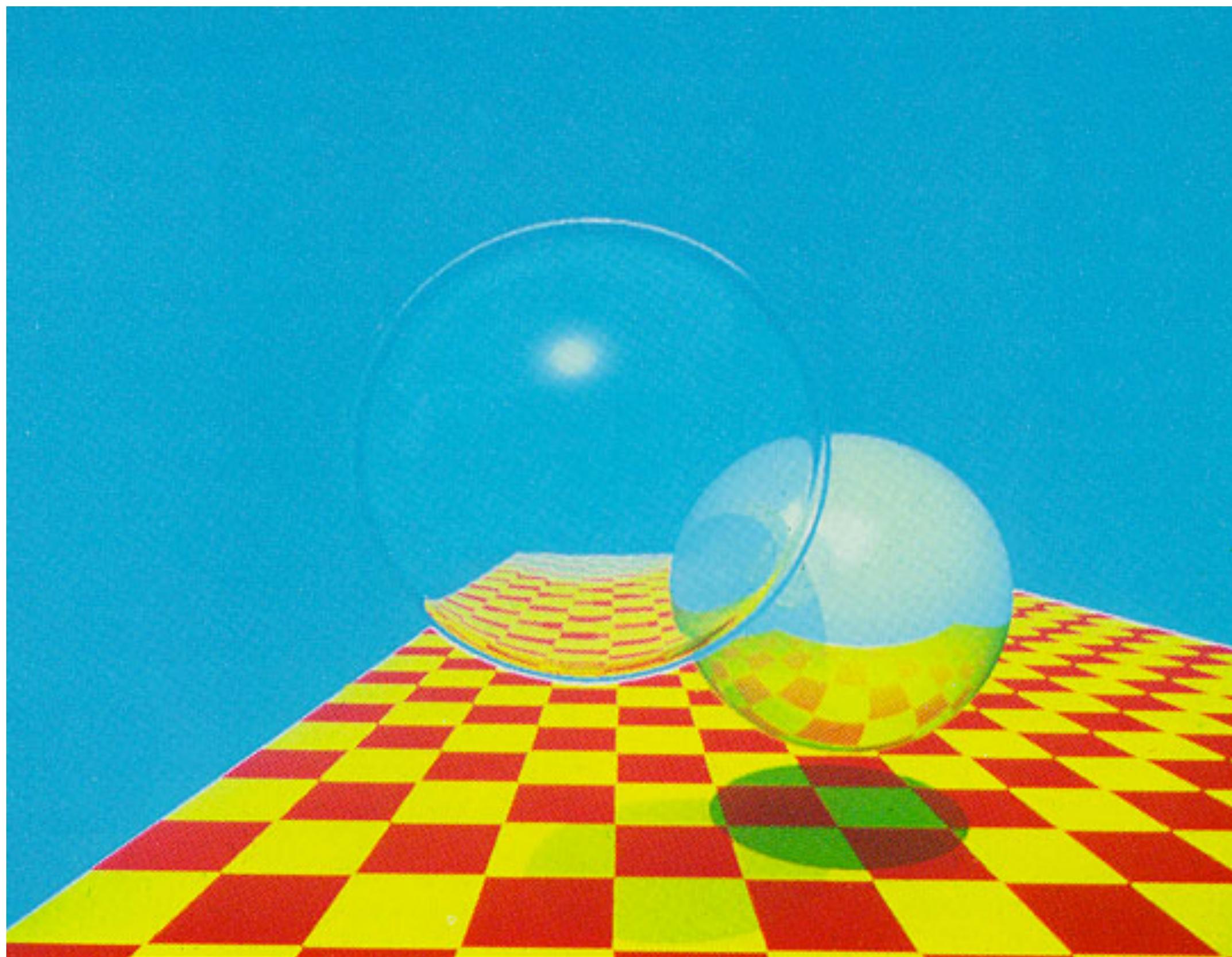
Shadow rays

Add shadows in ray tracing

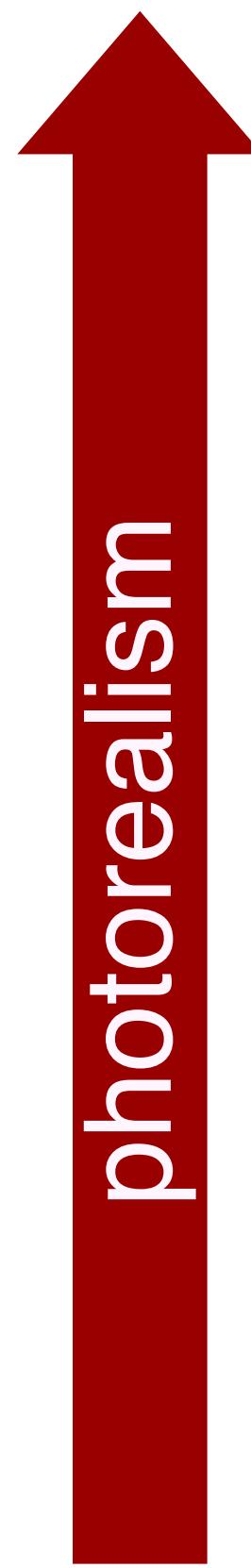


$$C = C_{ambi} + \sum_{i=1}^2 \text{visible}(\mathbf{l}_i, \mathbf{x}) \cdot C_{l_i} \circ C_{surf} \max(\mathbf{n} \cdot \mathbf{l}_i, 0) + C_{spec}$$

Ray tracing examples



Rendering equation

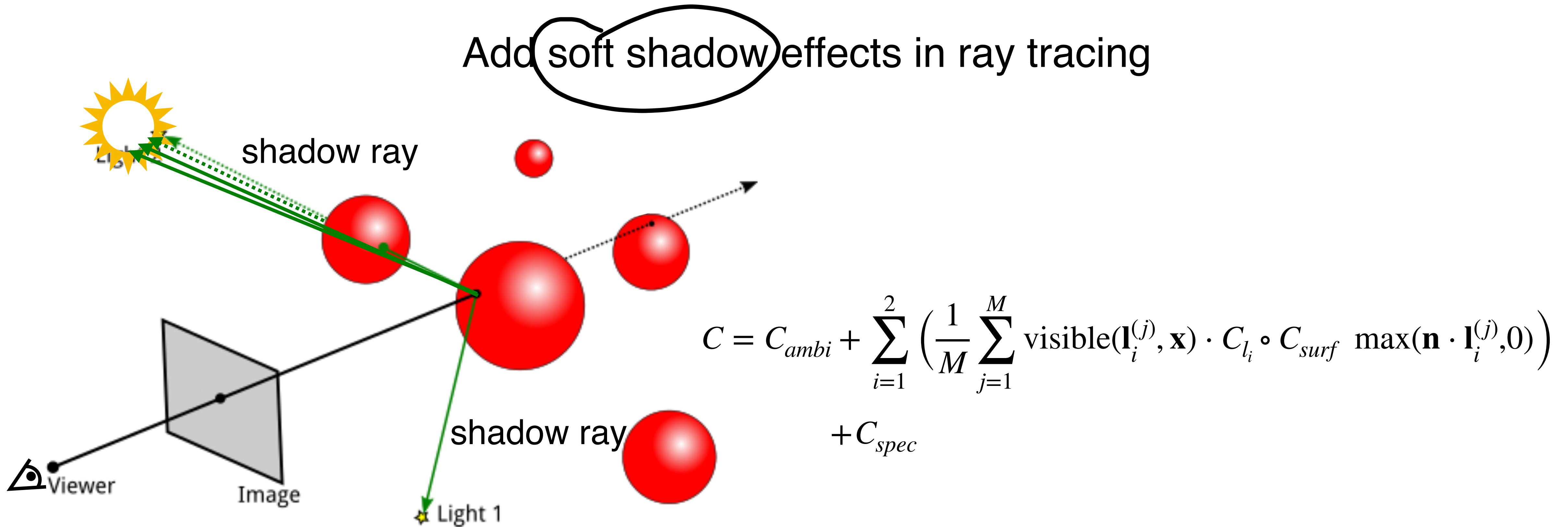


Rendering equation: $L_o(\mathbf{x}, \omega_o, \lambda) = L_e(\mathbf{x}, \omega_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \omega_i, \omega_o, \lambda) L_i(\mathbf{x}, \omega_i, \lambda) (\omega_i \cdot \mathbf{n}) d\omega_i$

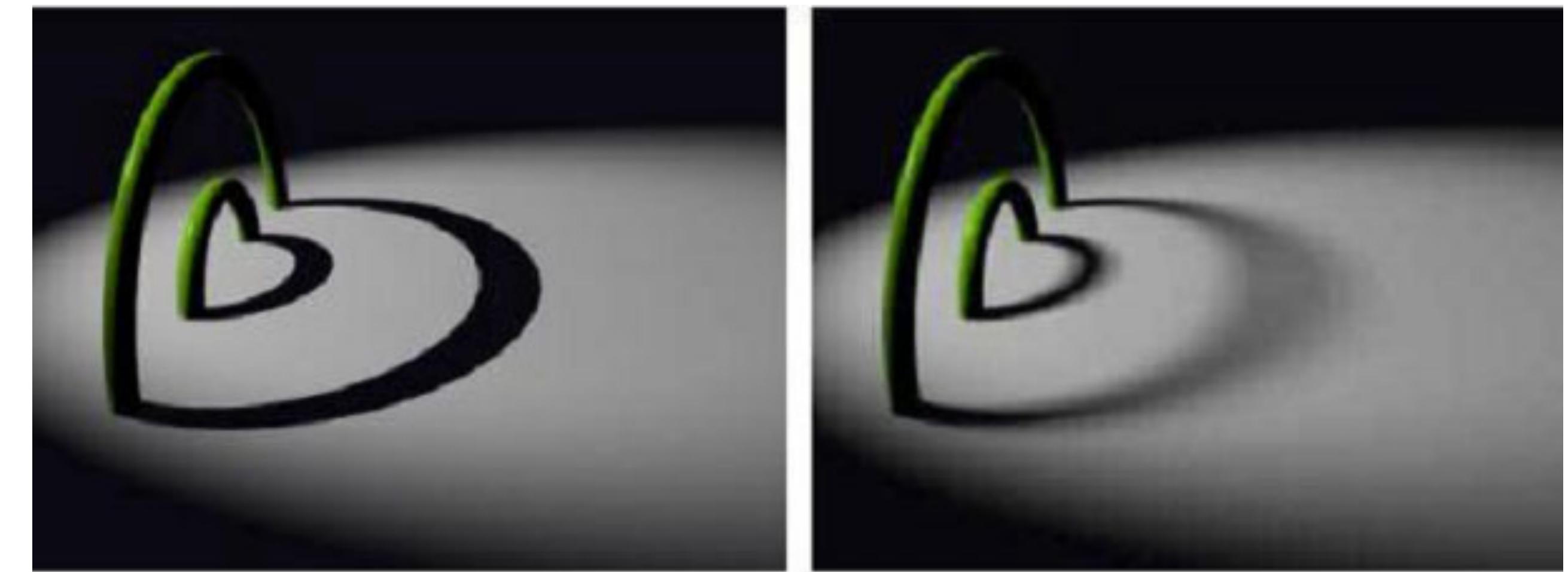
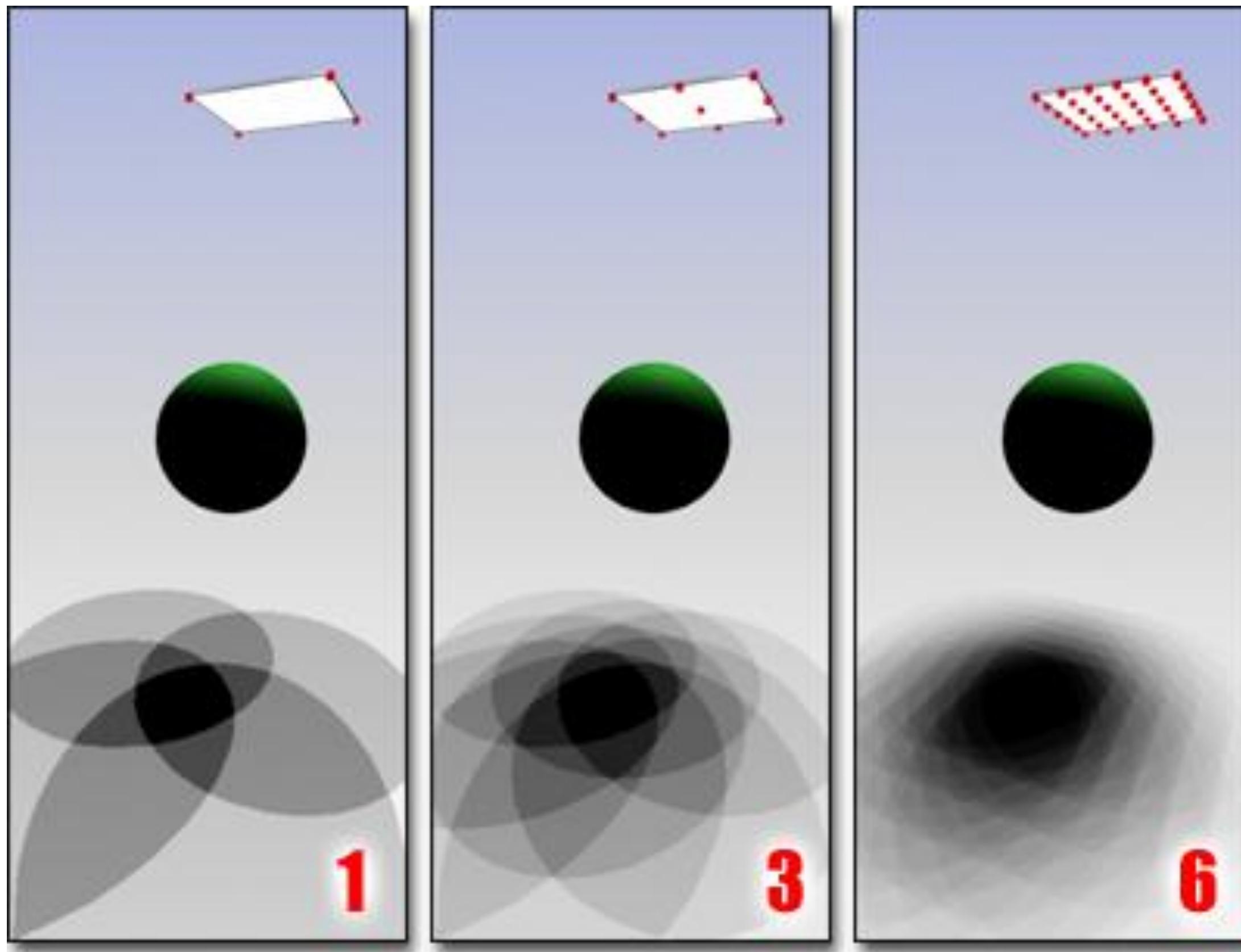
Phong model with **ray tracing**: Add reflection, refraction and hard shadows to basic Phong model.

Phong model with **rasterization**: $C = C_{surf} \circ C_{ambi} + C_{surf} \circ \sum_{i=1}^N C_{l_i} \max(0, \mathbf{n} \cdot \mathbf{l}_i) + \sum_{i=1}^N C_{l_i} \max(0, \mathbf{e} \cdot \mathbf{r}_i)^p$

Distribution ray tracing

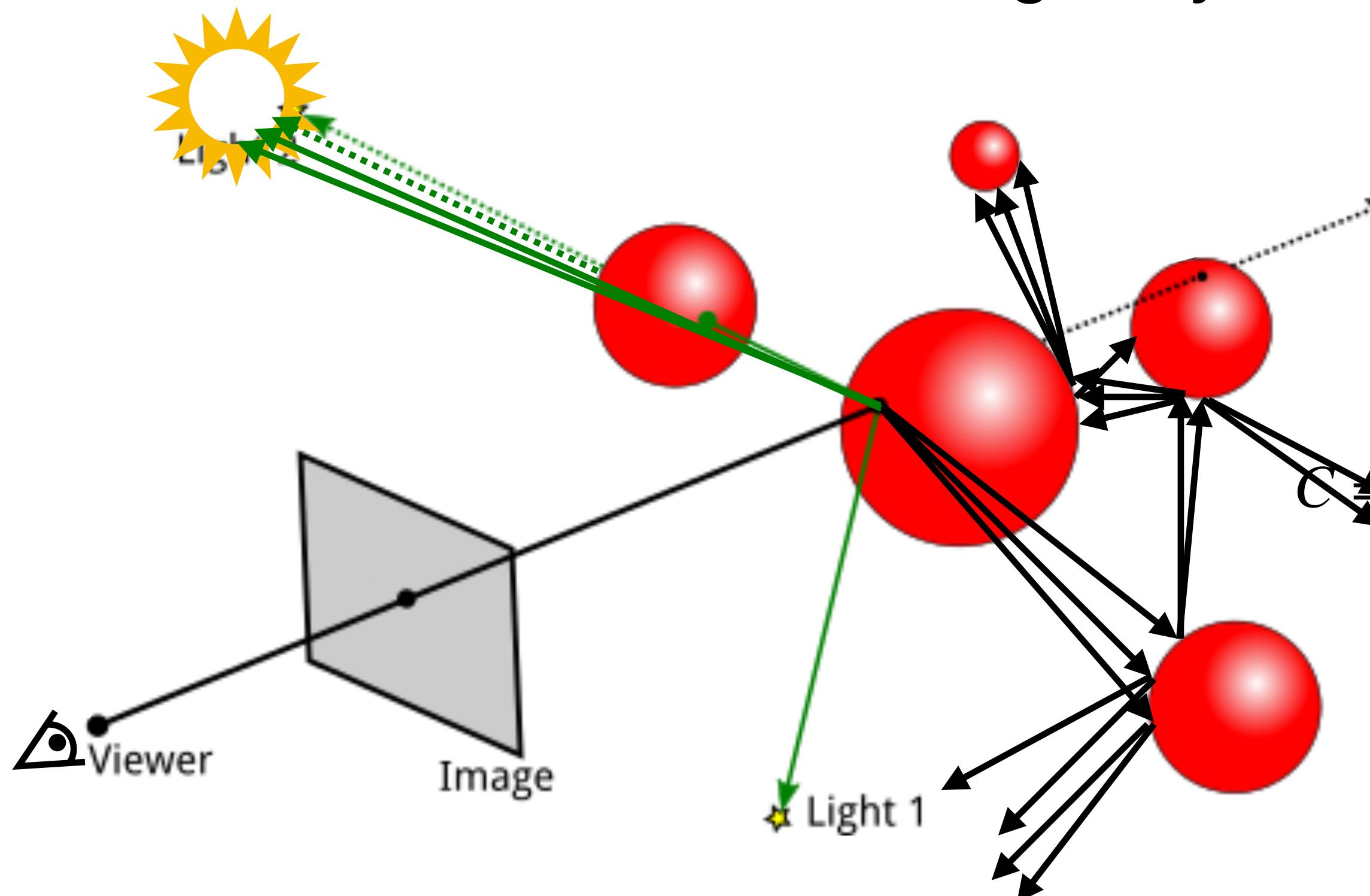


Distribution ray tracing examples



Distribution ray tracing

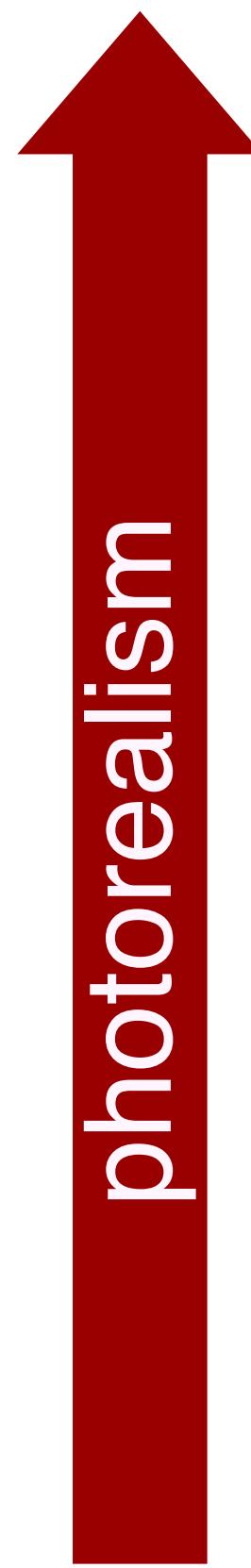
Add glossy reflection effects in ray tracing



$$C \leftarrow C_{ambi} + \sum_{i=1}^2 \left(\frac{1}{M} \sum_{j=1}^M \text{visible}(l_i^{(j)}, \mathbf{x}) \cdot C_{l_i} \circ C_{surf} \max(\mathbf{n} \cdot \mathbf{l}_i^{(j)}, 0) \right) \\ + C_{spec} + k_{refl}^{(1)} \cdot \sum_{k=1}^P C_{refl_p}^{(1)}$$

Exponentially expand
the reflection terms.

Rendering equation



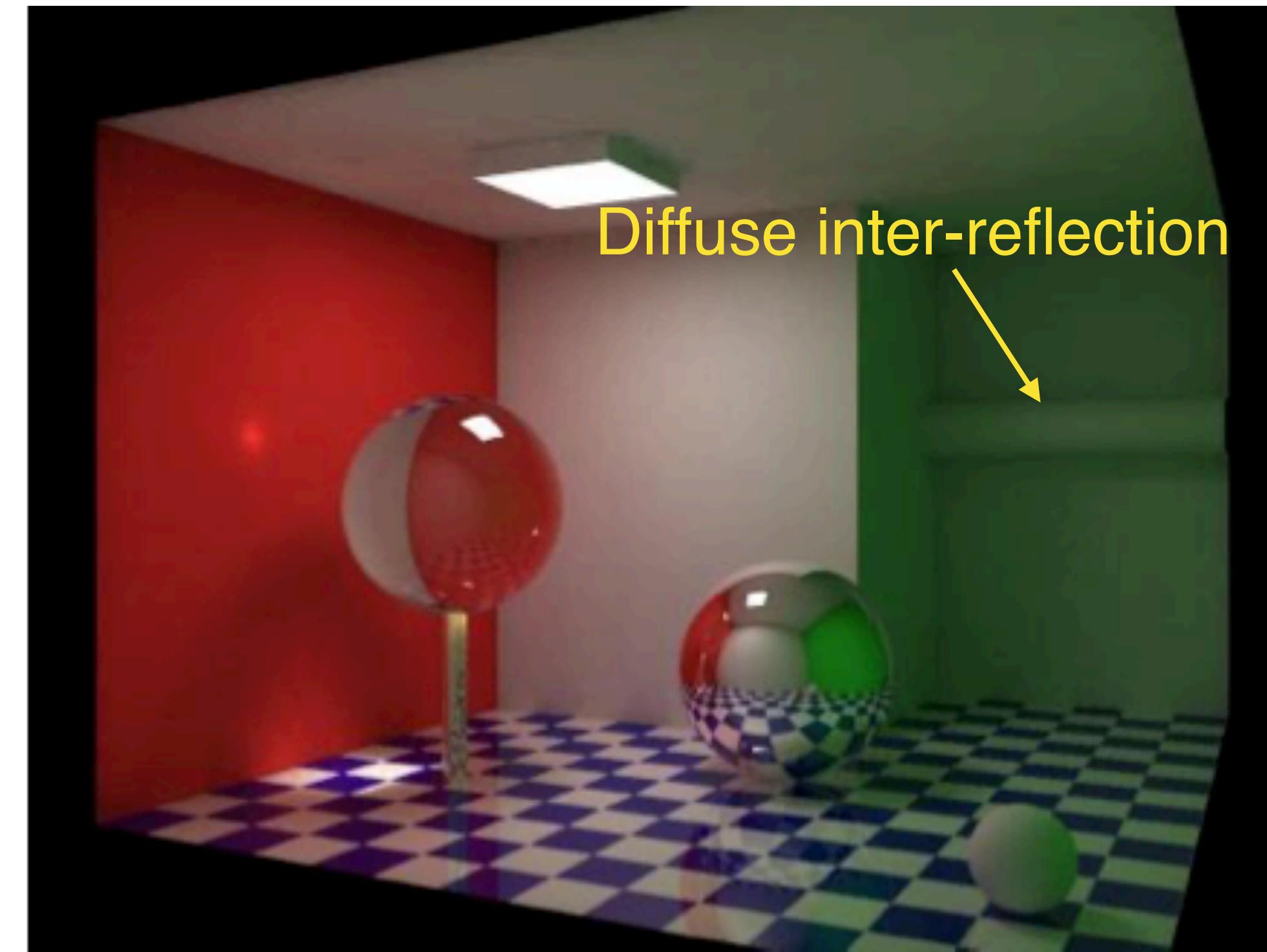
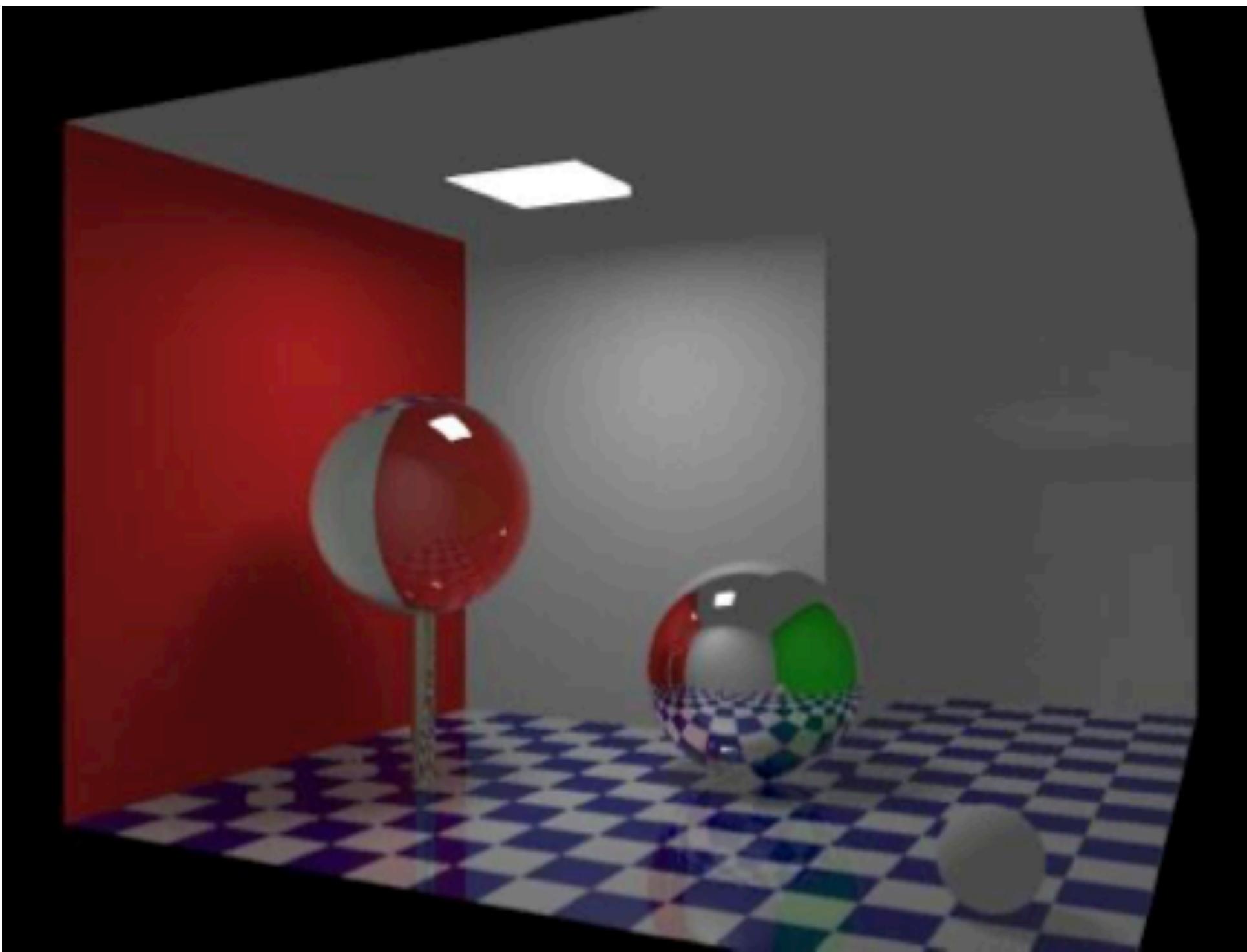
Rendering equation: $L_o(\mathbf{x}, \omega_o, \lambda) = L_e(\mathbf{x}, \omega_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \omega_i, \omega_o, \lambda) L_i(\mathbf{x}, \omega_i, \lambda) (\omega_i \cdot \mathbf{n}) d\omega_i$

Distribution ray tracing: Can approximate soft shadows and glossy (but not perfectly reflective) surfaces.

Phong model with **ray tracing**: Add reflection, refraction and hard shadows to basic Phong model.

Phong model with **rasterization**: $C = C_{surf} \circ C_{ambi} + C_{surf} \circ \sum_{i=1}^N C_{l_i} \max(0, \mathbf{n} \cdot \mathbf{l}_i) + \sum_{i=1}^N C_{l_i} \max(0, \mathbf{e} \cdot \mathbf{r}_i)^p$

Beyond ray tracing



Rendering equation



$$\text{Rendering equation: } L_o(\mathbf{x}, \omega_o, \lambda) = L_e(\mathbf{x}, \omega_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \omega_i, \omega_o, \lambda) L_i(\mathbf{x}, \omega_i, \lambda) (\omega_i \cdot \mathbf{n}) d\omega_i$$

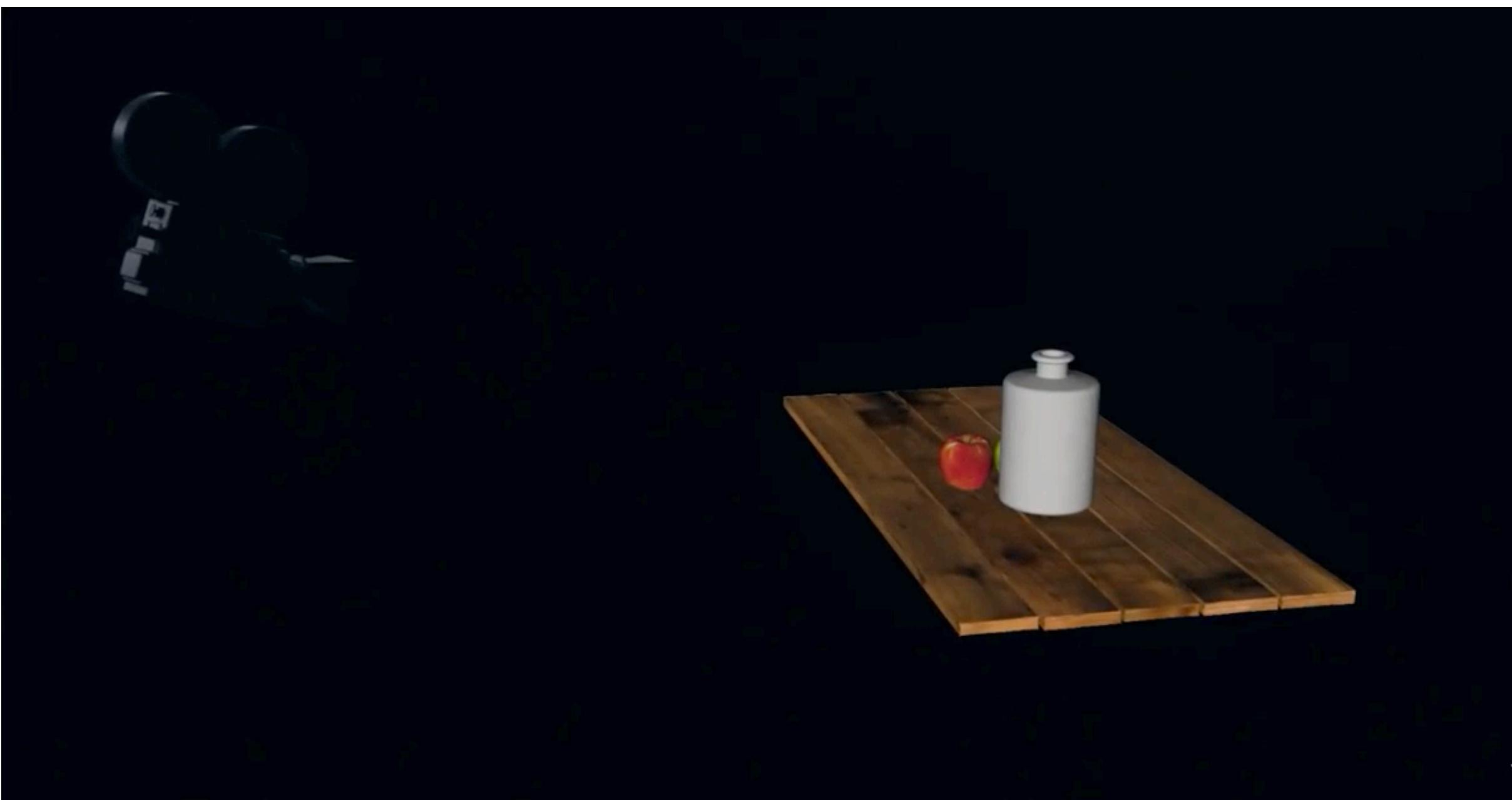
Path tracing: Close approximation of the solution to the rendering equation.

Distribution ray tracing: Can approximate soft shadows and glossy (but not perfectly reflective) surfaces.

Phong model with ray tracing: Add reflection, refraction and hard shadows to basic Phong model.

$$\text{Phong model with rasterization: } C = C_{surf} \circ C_{ambi} + C_{surf} \circ \sum_{i=1}^N C_{l_i} \max(0, \mathbf{n} \cdot \mathbf{l}_i) + \sum_{i=1}^N C_{l_i} \max(0, \mathbf{e} \cdot \mathbf{r}_i)^p$$

Path tracing



for each pixel p

 create ray r from eye through p

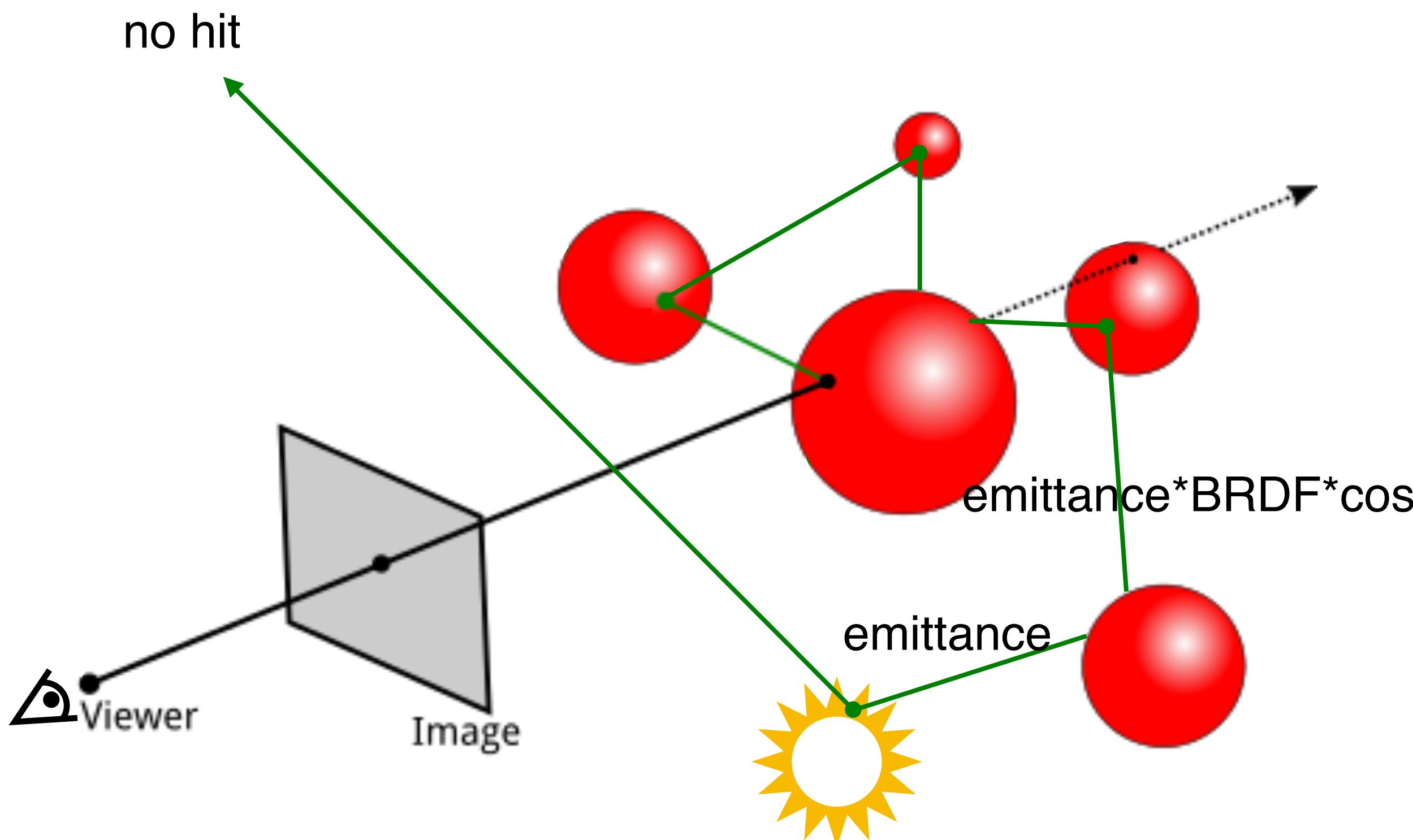
for each sample

$color += \text{PathTracer}(r)$

 render p in $color/\text{numSamples}$

- Send a ray through a pixel, find the first intersection on the surface of an object, and integrate over all the illuminance arriving to the intersection point.
- Compute illuminance by recursively projecting a ray from the surface to the scene in a bouncing path that terminates when a light source is intersected.
- The light is then sent backwards through the path and to the output pixel.

Path tracing



PathTracer(r)

if $r.\text{hitNothing}$

return Black

if $r.\text{hit.material.emmittance}$

return $r.\text{hit.material.emmittance}$

create a new ray r' from $r.\text{hit}$ in random direction

light = PathTracer(r')

$\cos = \text{dot}(r'.\text{direction}, r.\text{hit.normal})$

$\text{BRDF} = r.\text{hit.material.reflectance}$

return $\text{BRDF} * \text{light} * \cos$

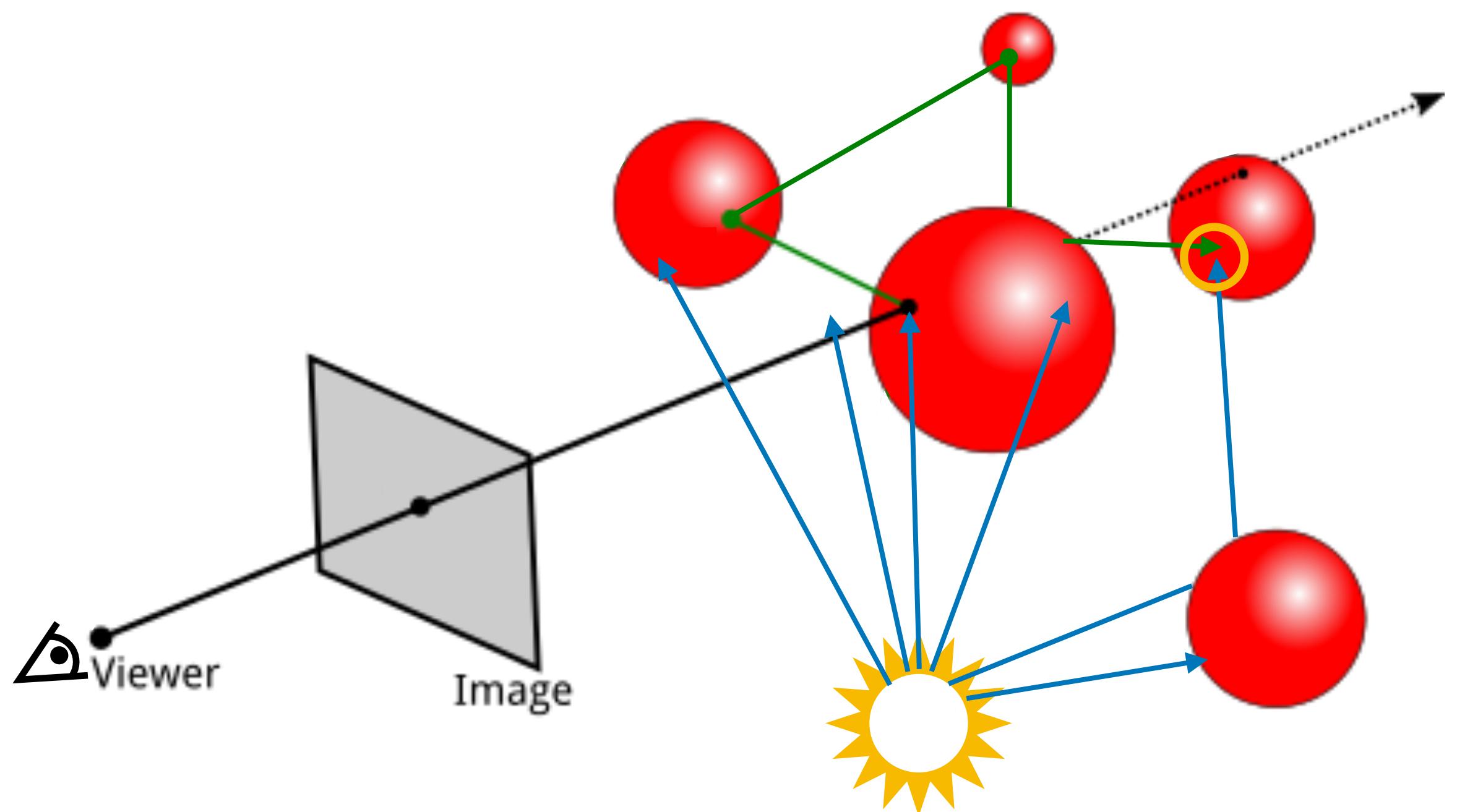


CHATEAU ARMIN

Château Arnim
Vins de Bourgogne

Château Arnim

Bidirectional path tracing (BDPT)



- Reflected light for a point can be approximated by two ways:
 - Shooting rays from the light sources into the scene. Billions of paths are created and bounced for a number of steps.
 - Gathering rays from a point on a surface. Bounce rays from the surface to the scene and terminate them at light sources.
- Bidirectional Path Tracing combines both **Shooting** and **Gathering** to obtain faster convergence:
 - Shooting and gathering paths are traced independently.
 - The head of the shooting path is connected to the tail of the gathering path.
 - Attenuate light at every bounce and back out into the pixel.

2spp





2spp denoised



4spp



4spp denoised

8spp





8spp denoised



50spp



50spp denoised



1000spp



50spp denoised



1000spp denoised



12000spp



12000spp denoised



50spp denoised



12000spp

Summary



Rendering equation: $L_o(\mathbf{x}, \omega_o, \lambda) = L_e(\mathbf{x}, \omega_o, \lambda) + \int_{\Omega} f(\mathbf{x}, \omega_i, \omega_o, \lambda) L_i(\mathbf{x}, \omega_i, \lambda) (\omega_i \cdot \mathbf{n}) d\omega_i$

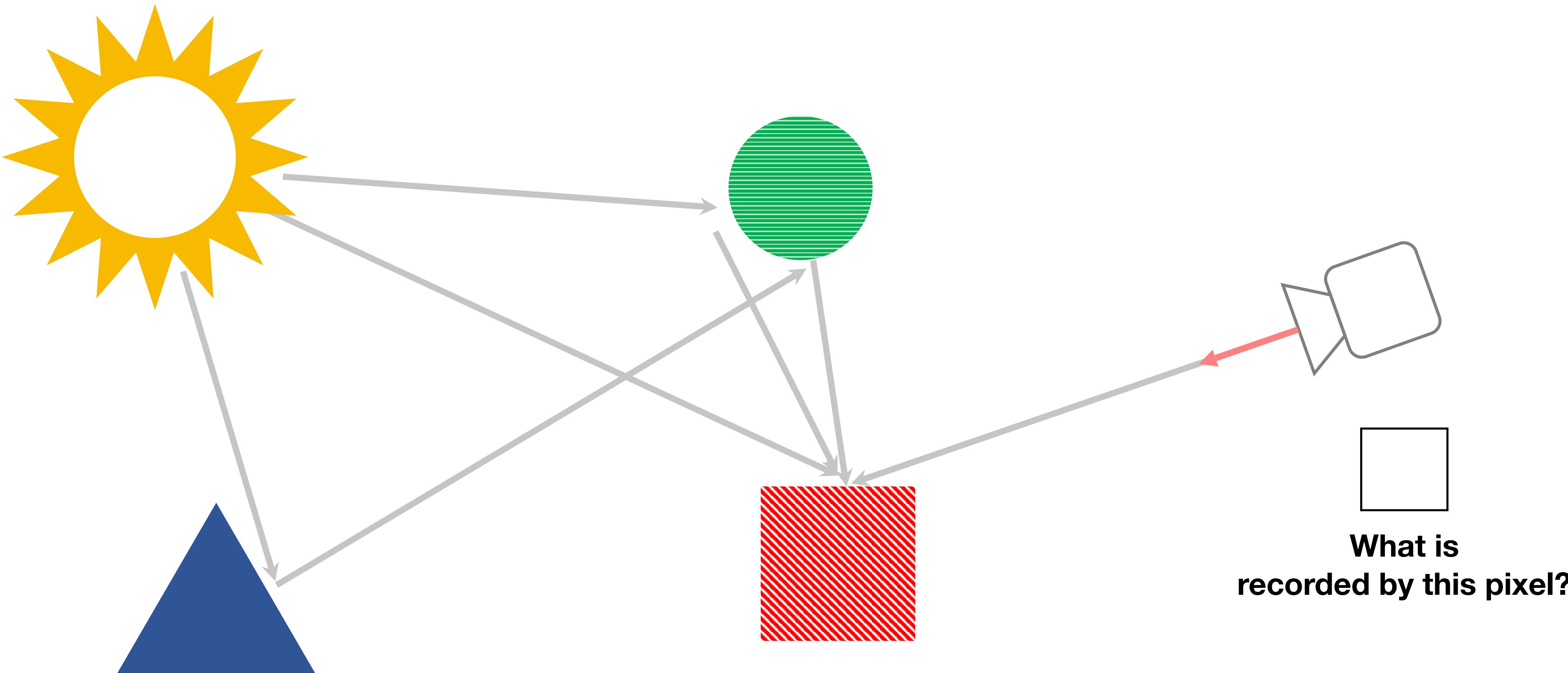
Path tracing: Close approximation of the solution to the rendering equation.

Distribution ray tracing: Can approximate soft shadows and glossy (but not perfectly reflective) surfaces.

Phong model with ray tracing: Add reflection, refraction and hard shadows to basic Phong model.

Phong model with rasterization: $C = C_{surf} \circ C_{ambi} + C_{surf} \circ \sum_{i=1}^N C_{l_i} \max(0, \mathbf{n} \cdot \mathbf{l}_i) + \sum_{i=1}^N C_{l_i} \max(0, \mathbf{e} \cdot \mathbf{r}_i)^p$

Other effects to be aware of



Depth-of-Field



aperture....f 1.8
shutter.....1/500
ISO.....100
distance...~3ft

aperture....f 4
shutter.....1/125
ISO.....100
distance...~3ft

aperture....f 8
shutter.....1/40
ISO.....125
distance...~3ft

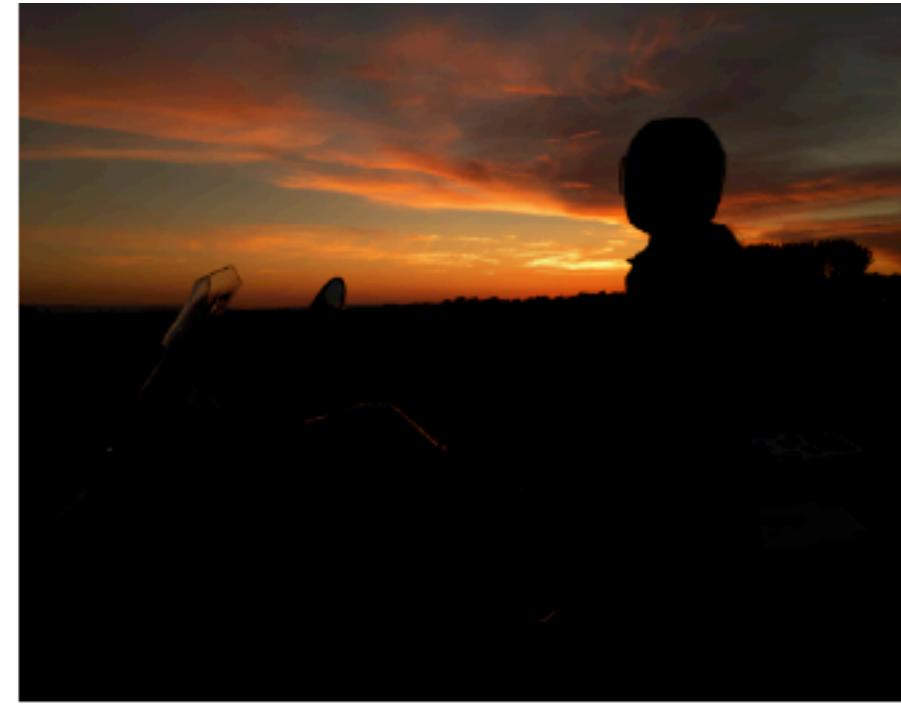
<http://photographywisdom.com>

Important Imaging Effects: Dynamic Range

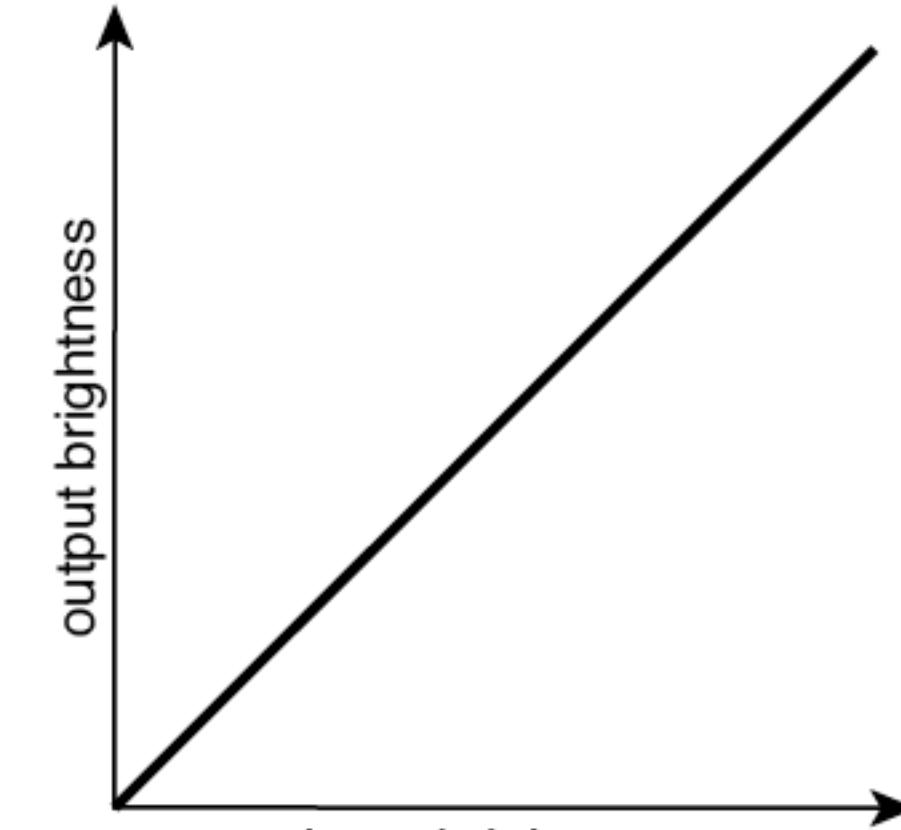


Wikipedia

Important Imaging Effects: Tone Mapping



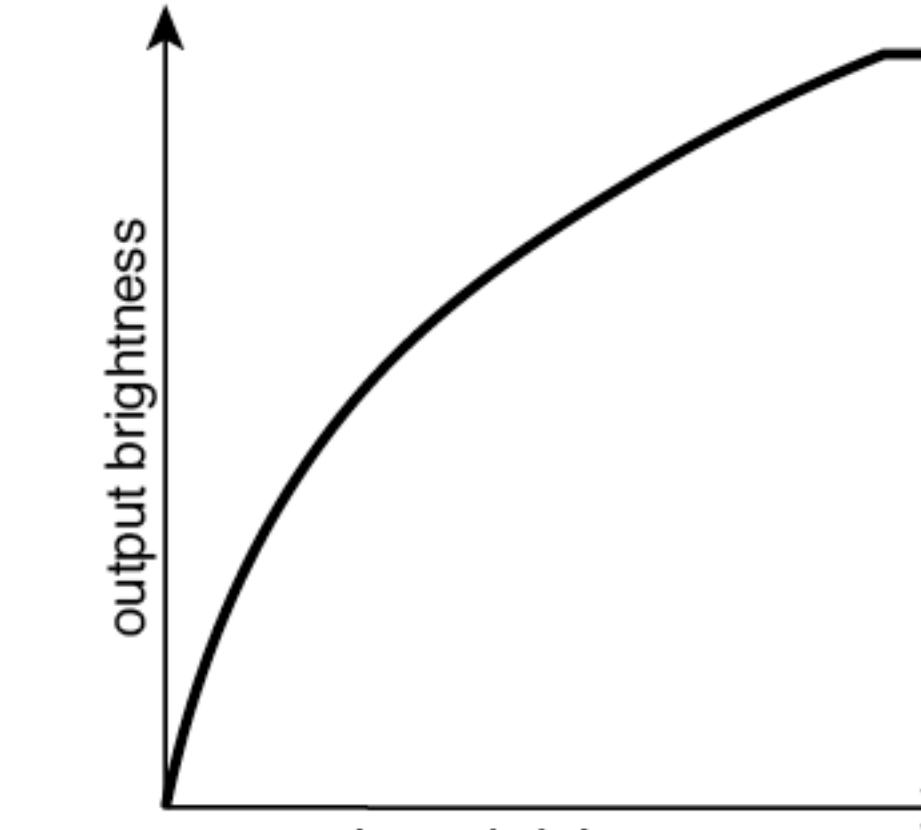
Linear RGB image



(a)



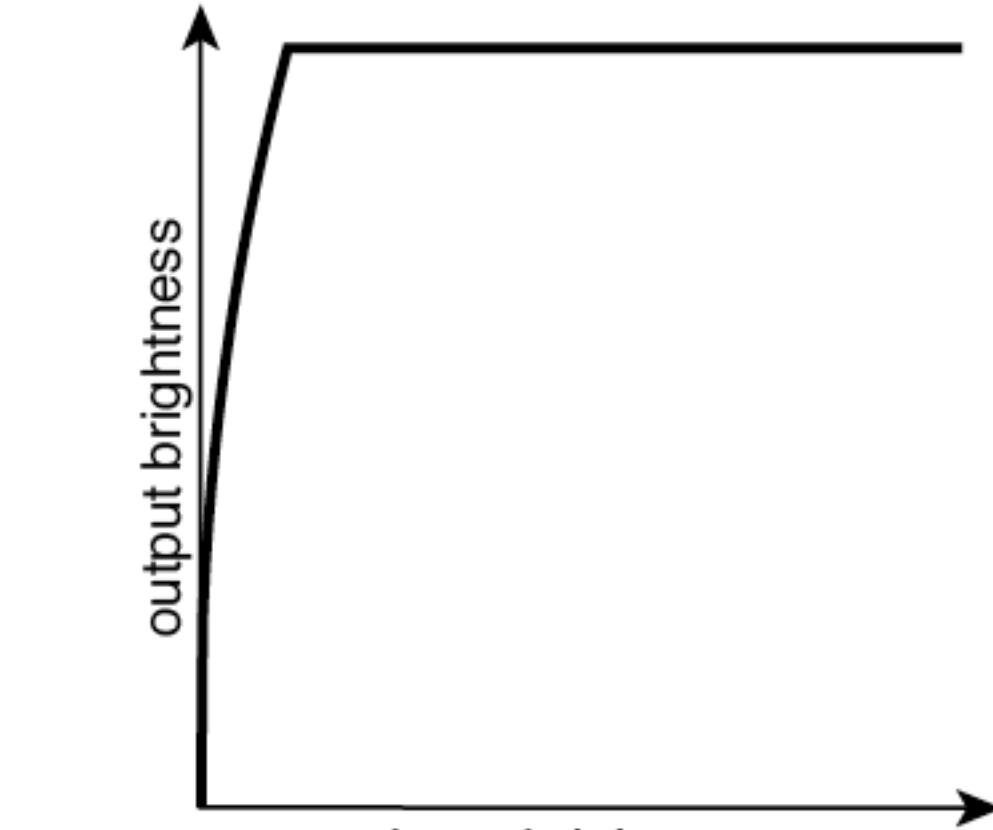
Highlights visible



(b)

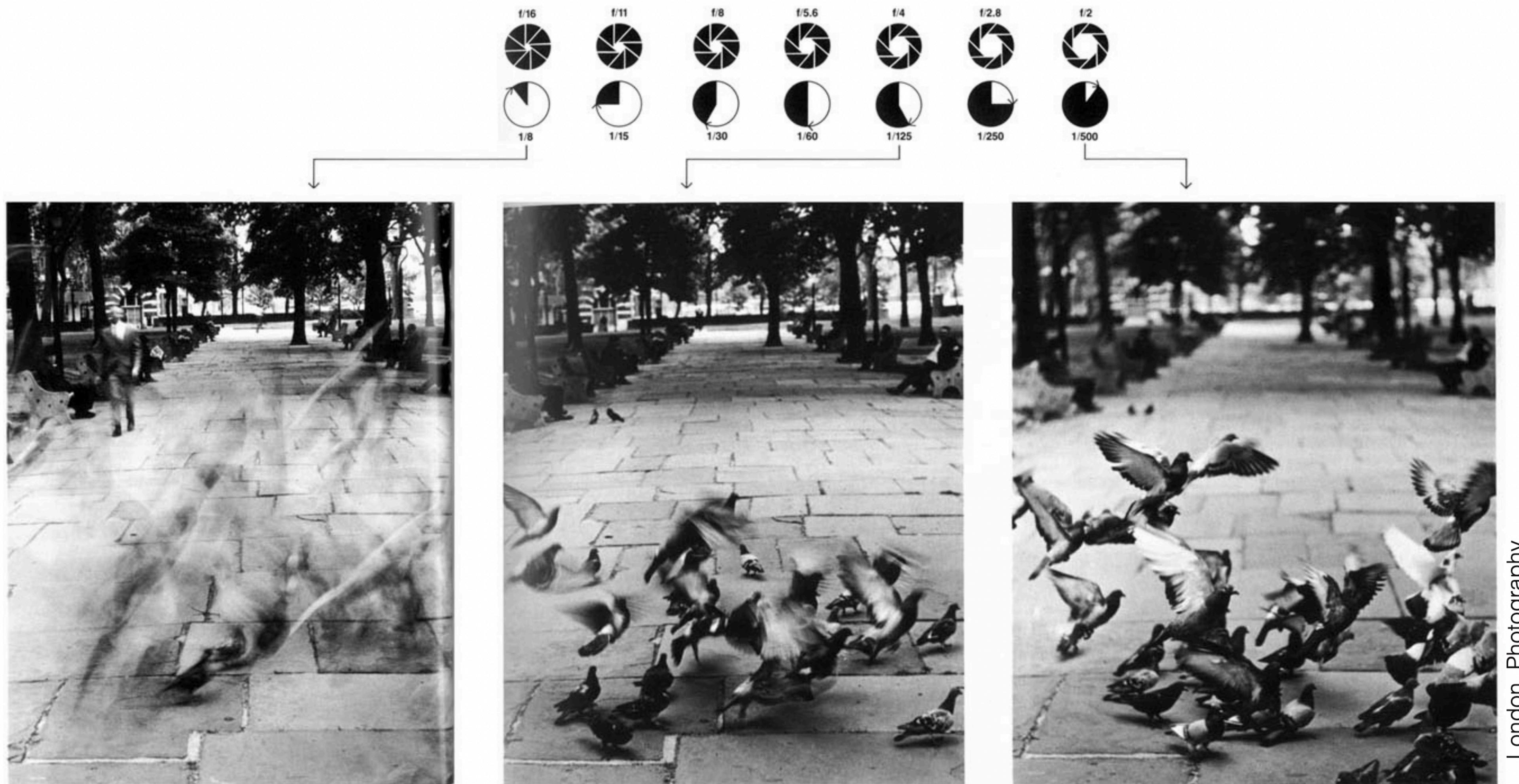


Shadows visible

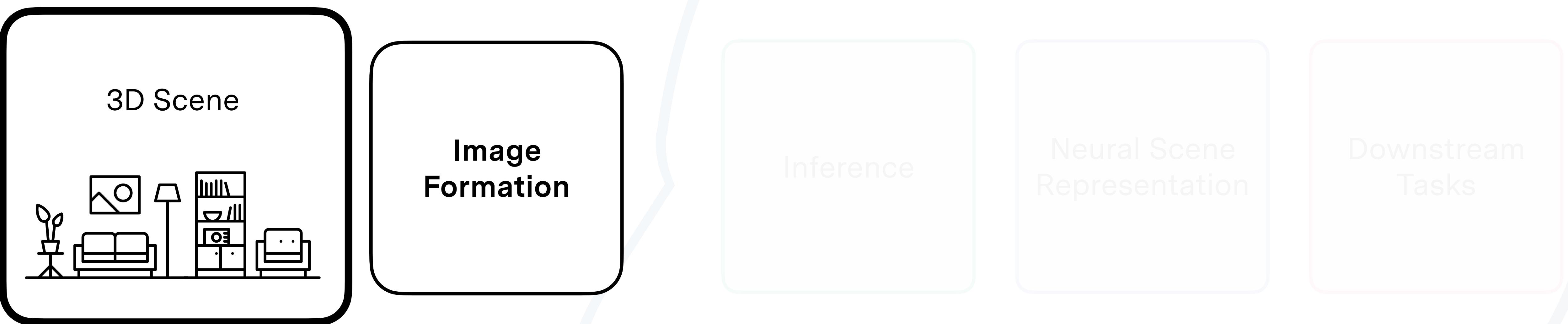


(c)

Important Imaging Effects: Motion Blur



Today: Light Transport & High-level of Physics-based Rendering



Why?

In order to reason about the 3D world from images, we need to understand how 3D properties such as materials, lighting, etc. relate to the measurements observed by a camera.

What you'll learn.

The rendering equation, simulating light transport via multi-bounce ray-tracing, bidirectional radiance distribution functions, rasterization, rendering.