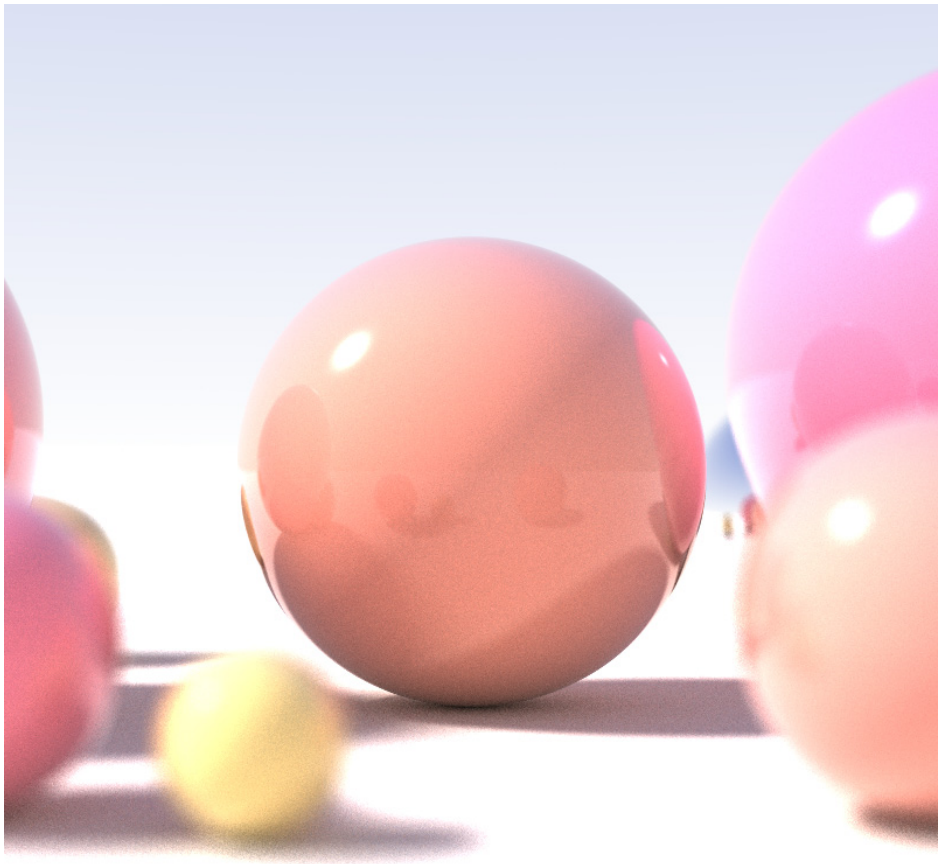
Differentiable Rendering



Evangelos Kalogerakis

574/674

What is rendering?

The process of generating a 2D image from a 3D representation

What is rendering?

The process of generating a 2D image from a 3D representation

Rasterization **Ray tracing**

What is rendering?

The process of generating a 2D image from a 3D representation

Rasterization

for each polygon

project it onto 2D plane

for each pixel in polygon

shade the pixel

Ray tracing

What is rendering?

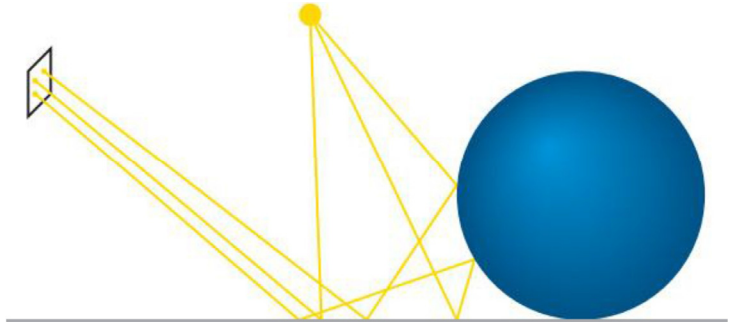
The process of generating a 2D image from a 3D representation

Rasterization

```
for each polygon
    project it onto 2D plane
    for each pixel in polygon
        shade the pixel
```

Ray tracing

```
for each pixel throw ray
    for each object
        calculate intersection
        shade the pixel
```



What is rendering?

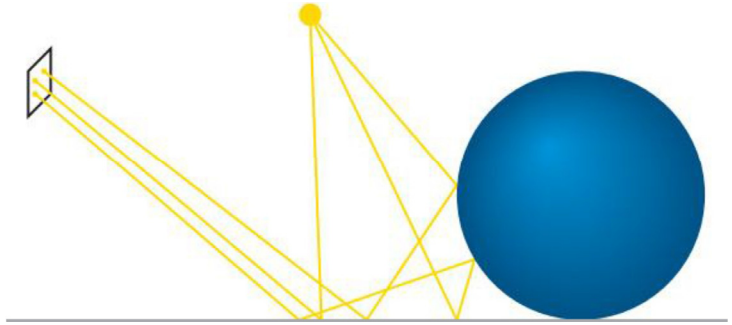
Both are heavily parallelized in GPU hardware!

Rasterization

```
for each polygon
    project it onto 2D plane
    for each pixel in polygon
        shade the pixel
```

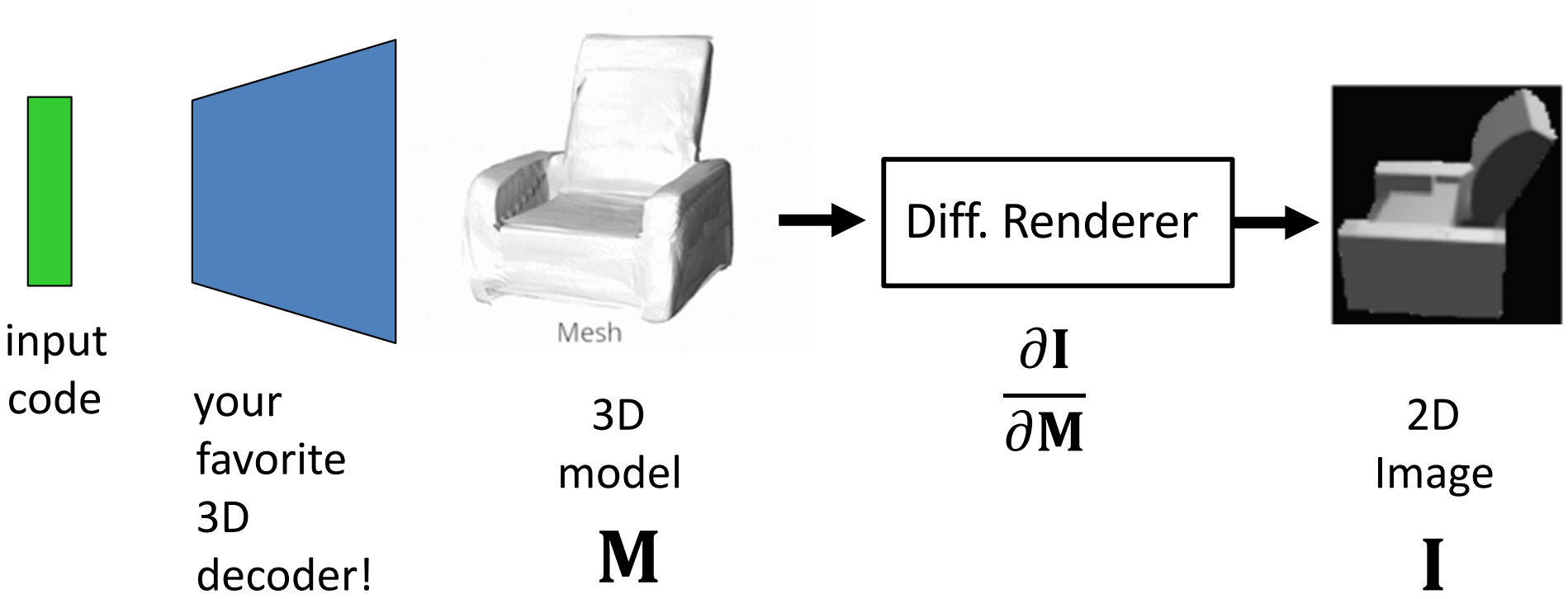
Ray tracing

```
for each pixel throw ray
    for each object
        calculate intersection
        shade the pixel
```



What is **differentiable** rendering?

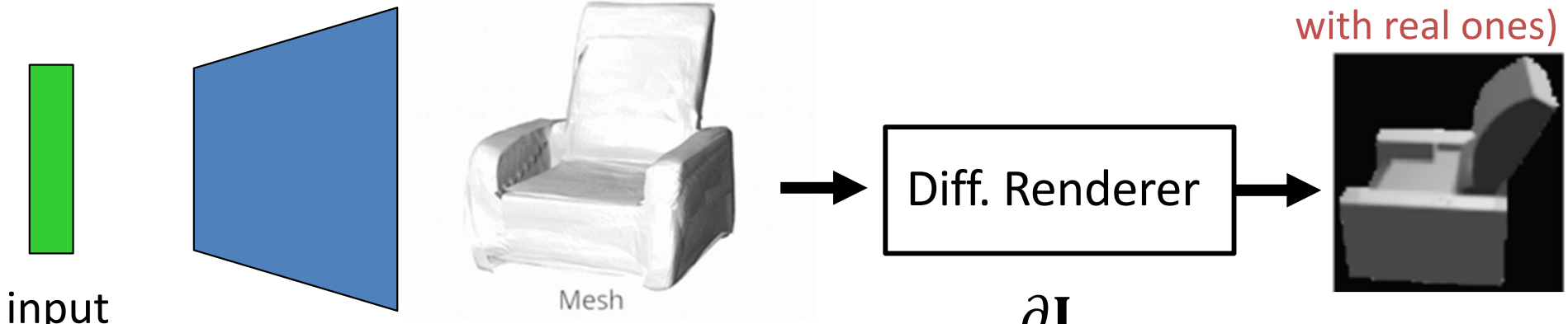
Allows incorporating rendering as a module in a NN!



What is **differentiable** rendering?

Allows incorporating rendering as a module in a NN!

Apply losses here!
(compare rendered images with real ones)



input code

your favorite 3D decoder!

3D model

$\mathbf{M}$

$\frac{\partial \mathbf{I}}{\partial \mathbf{M}}$

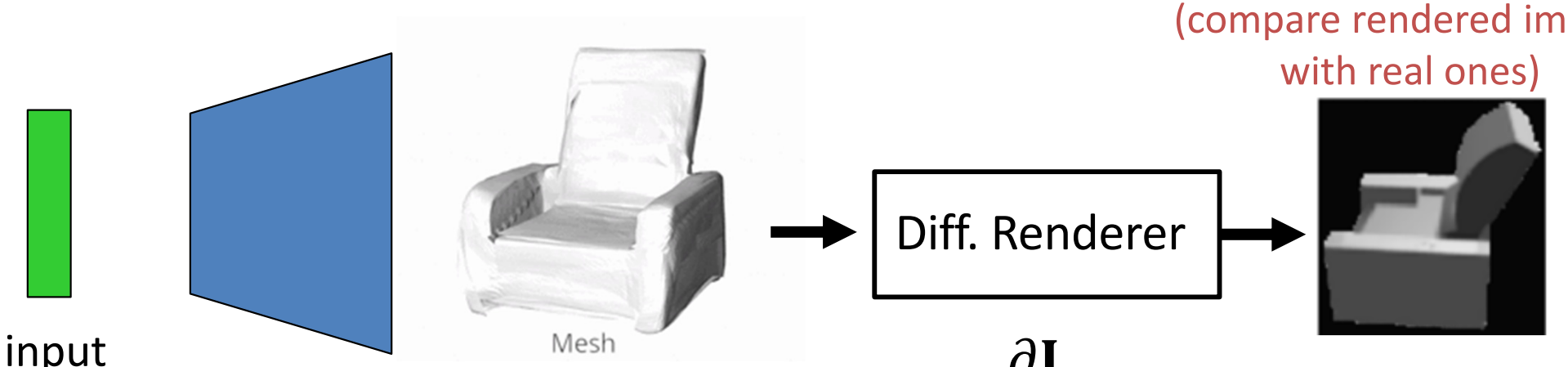
2D Image

$\mathbf{I}$

What is **differentiable** rendering?

Allows training 3D DL nets with losses coming from 2D image datasets (tons of them available!)

Apply losses here!
(compare rendered images with real ones)



input code

your favorite 3D decoder!

3D model

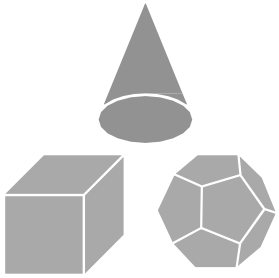
$\mathbf{M}$

$\frac{\partial \mathbf{I}}{\partial \mathbf{M}}$

2D Image

$\mathbf{I}$

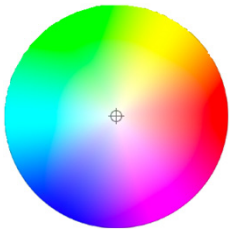
Rasterization pipeline



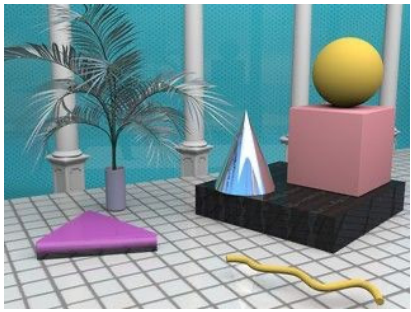
geometry

camera

lighting

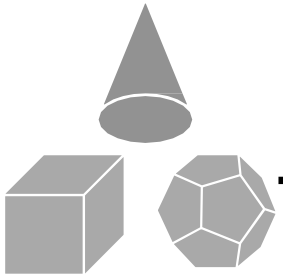


colors & materials



output image

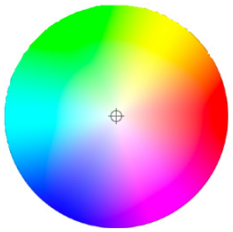
Rasterization pipeline



geometry

camera

lighting



colors & materials

Transformation

2D coordinates

Rasterization

fragments

depths

Z-buffering

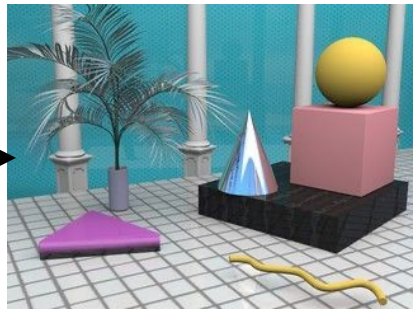
3D vertices,
normals

pixels

Vertex
shader

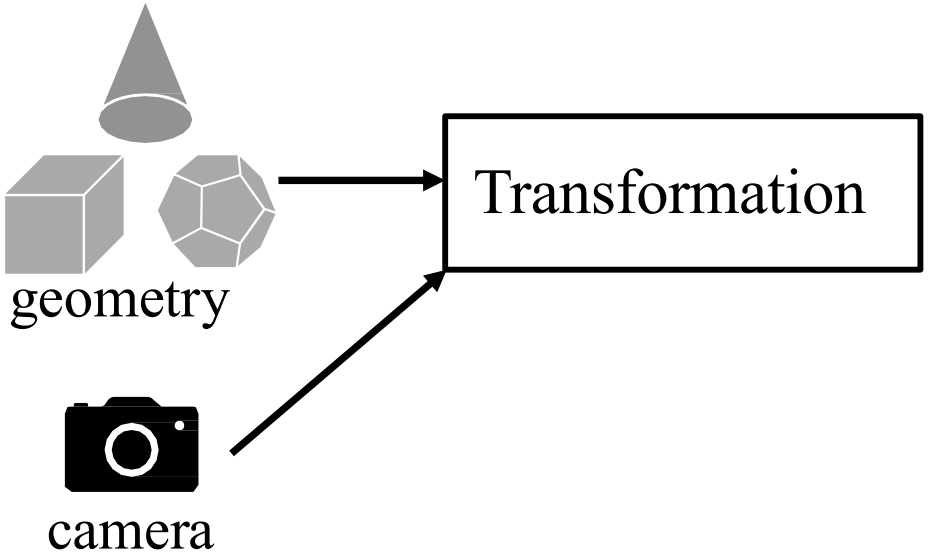
vertex colors

Fragment
shader



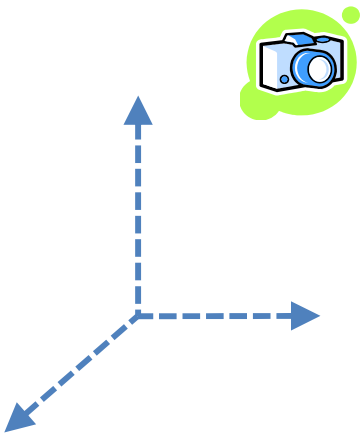
output image

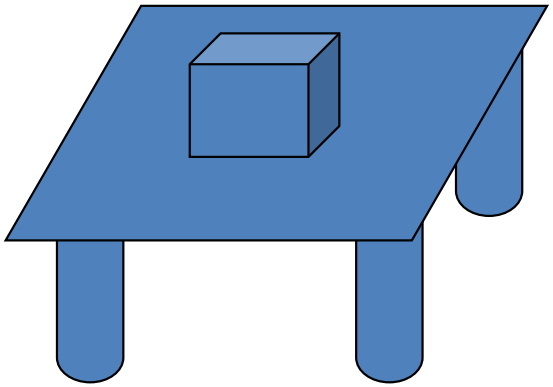
Rasterization pipeline



View transformation

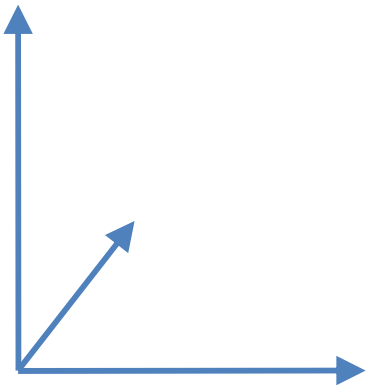
Position and orient a camera in your scene...





Change of coordinate systems

Suggest that an object is represented with respect to the world coordinate system (in blue)

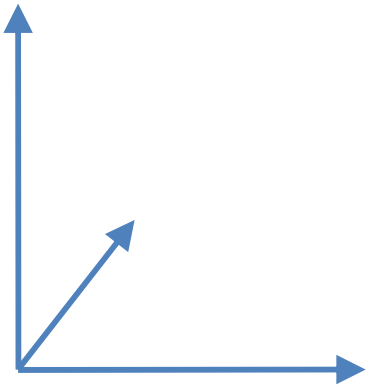


Original coordinate system

$\boldsymbol{p}=[x,y,z]^T$

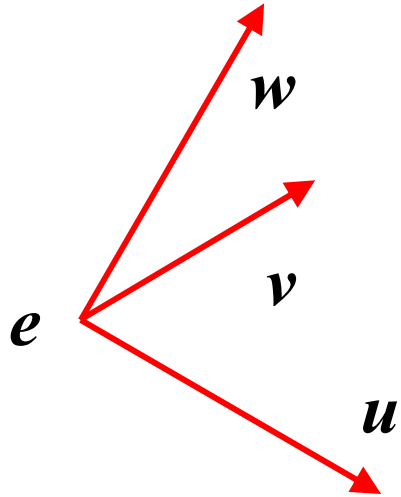
Change of coordinate systems

We want to change the coordinate system so that the camera is the center of our world...



Original coordinate system

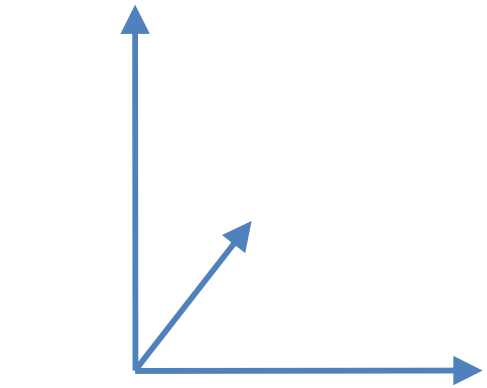
$\boldsymbol{p}=[x,y,z]^T$



New coordinate system
($\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}$ are orthonormal vectors,
$\boldsymbol{u}=[u_1,u_2,u_3]^T$
$\boldsymbol{v}=[v_1,v_2,v_3]^T$
$\boldsymbol{w}=[w_1,w_2,w_3]^T$
and $\boldsymbol{e}=[e_1,e_2,e_3]^T$ is the origin)

Change of coordinate systems

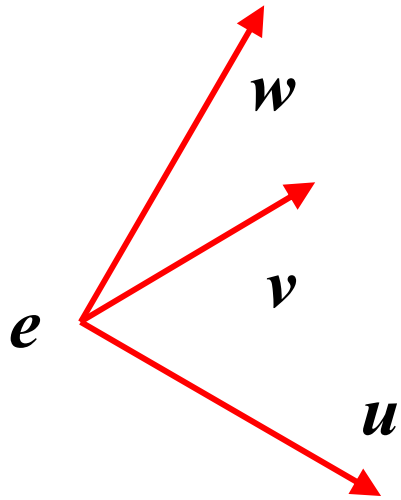
How do we compute the position of the point according to the new coordinate system?



Original coordinate system

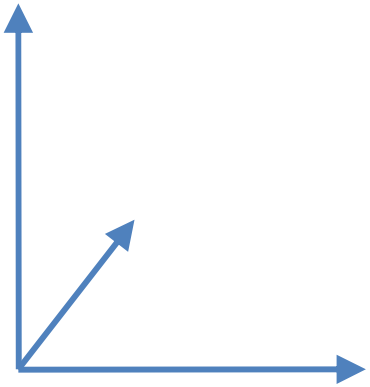
$\boldsymbol{p}=[x,y,z]^T$

$\boldsymbol{p}'=?$



New coordinate system
($\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}$ are orthonormal vectors,
$\boldsymbol{u}=[u_1,u_2,u_3]^T$
$\boldsymbol{v}=[v_1,v_2,v_3]^T$
$\boldsymbol{w}=[w_1,w_2,w_3]^T$
and $\boldsymbol{e}=[e_1,e_2,e_3]^T$ is the origin)

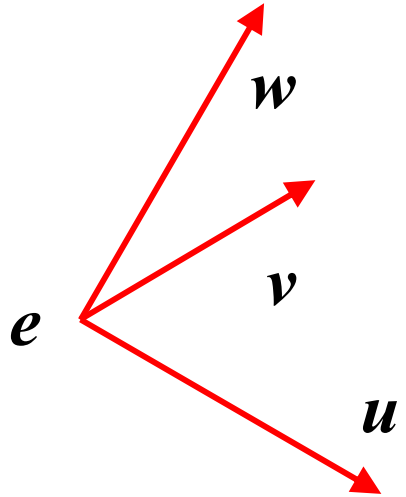
Change of coordinate systems



Original coordinate system

$\boldsymbol{p}=[x,y,z]^T$

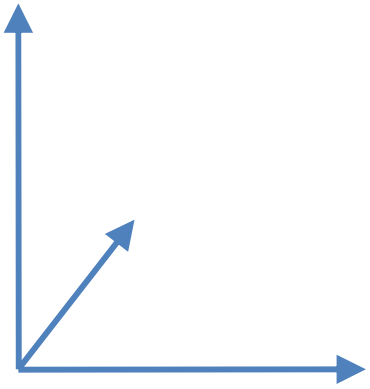
$\boldsymbol{p}'=?$



New coordinate system

$$\mathbf{p}' = \begin{bmatrix} u_1 & u_2 & u_3 & 0 \\ v_1 & v_2 & v_3 & 0 \\ w_1 & w_2 & w_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -e_1 \\ 0 & 1 & 0 & -e_2 \\ 0 & 0 & 1 & -e_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{p}$$

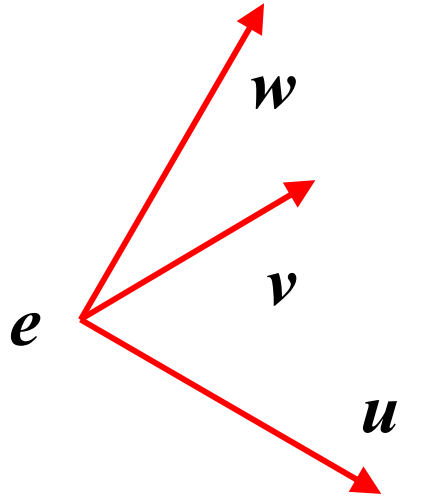
Change of coordinate systems



Original coordinate system

$\mathbf{p}=[x,y,z]^T$

$\mathbf{p}'=?$

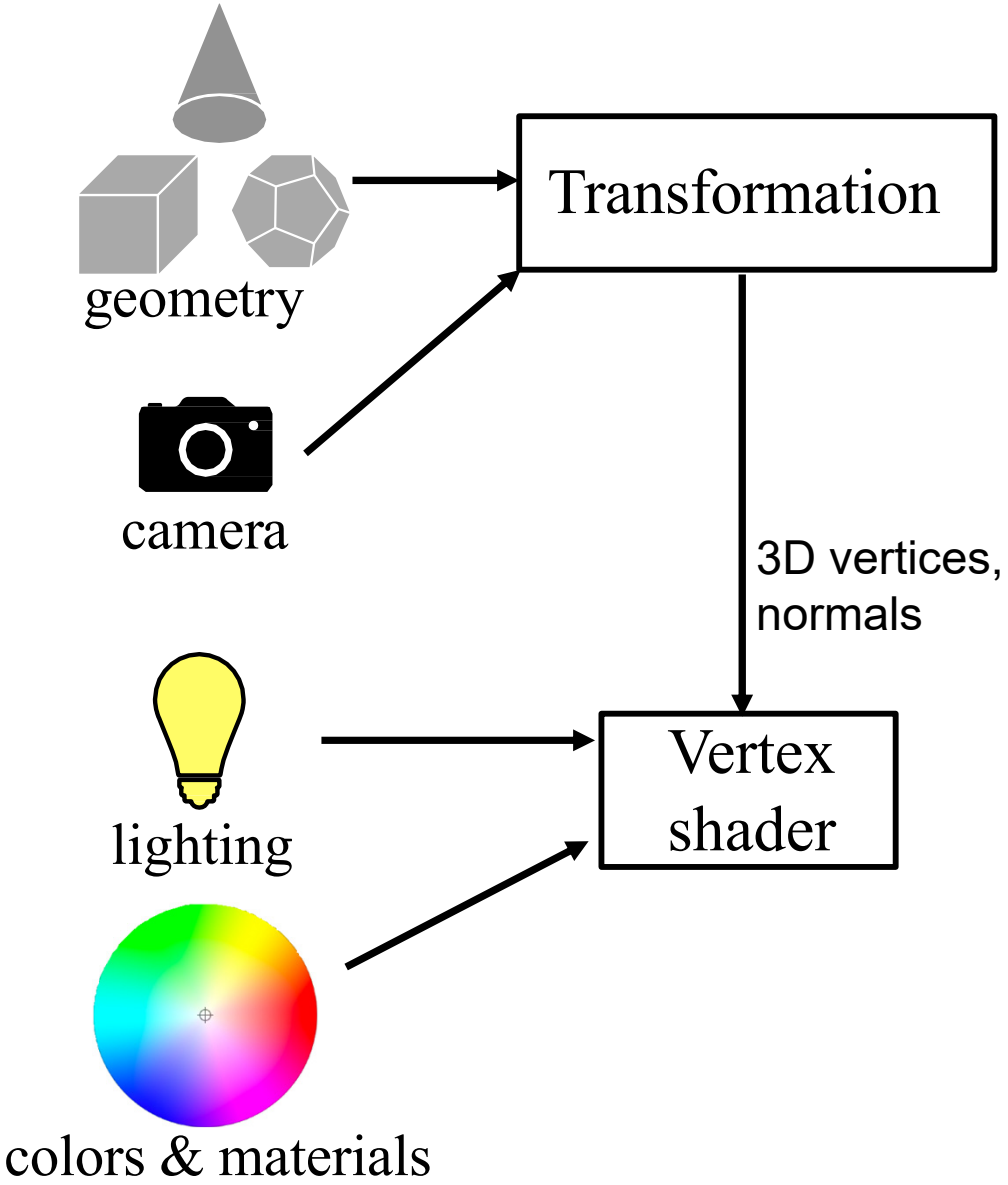


New coordinate system

$$\mathbf{p}' = \begin{bmatrix} u_1 & u_2 & u_3 & 0 \\ v_1 & v_2 & v_3 & 0 \\ w_1 & w_2 & w_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -e_1 \\ 0 & 1 & 0 & -e_2 \\ 0 & 0 & 1 & -e_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{p}$$

DIFFERENTIABLE

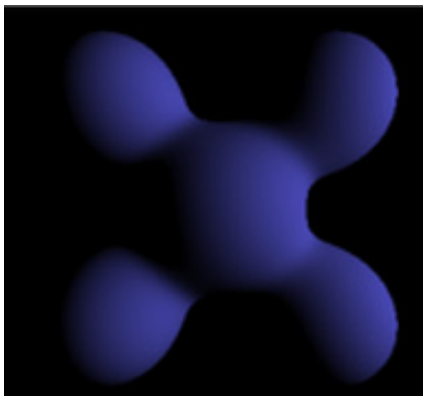
Rasterization pipeline

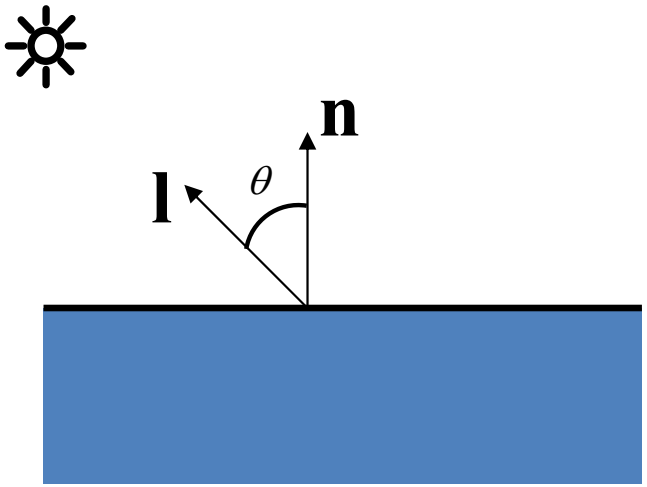


Diffuse / Lambertian Shading

Surface reflects light with equal intensity in all directions.

Obeys **Lambert's cosine law**: *"the amount of light the surface receives at a point depends on the angle between its normal and the vector from the surface point to the light"*



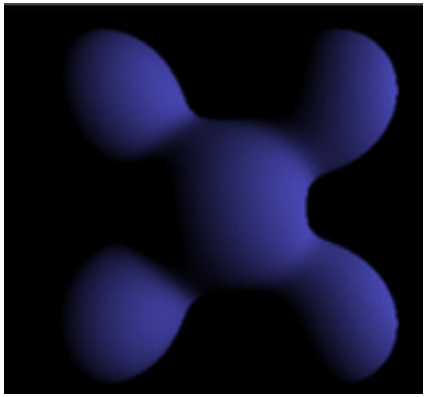


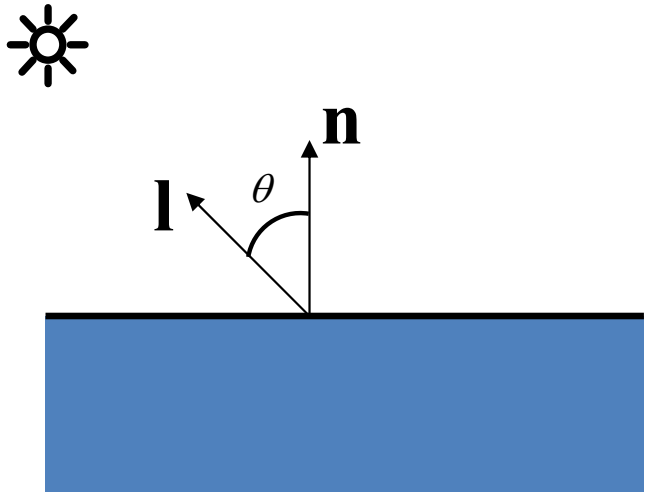
Diffuse / Lambertian Shading

Surface reflects light with equal intensity in all directions.

Obeys **Lambert's cosine law**: *"the amount of light the surface receives at a point depends on the angle between its normal and the vector from the surface point to the light"*

$$C_{diffuse} = C \cdot \max(\mathbf{n} \bullet \mathbf{l}, 0)$$

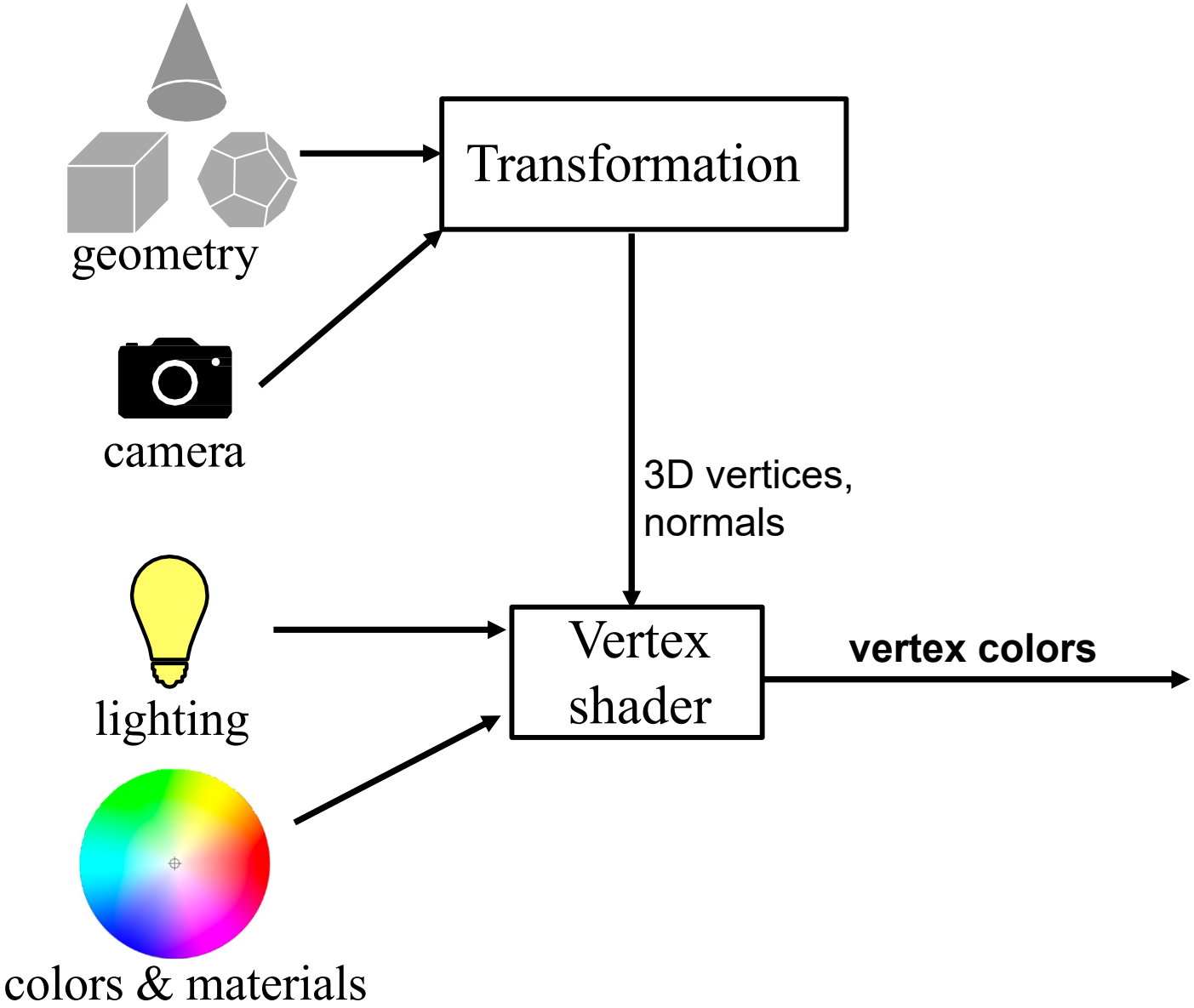




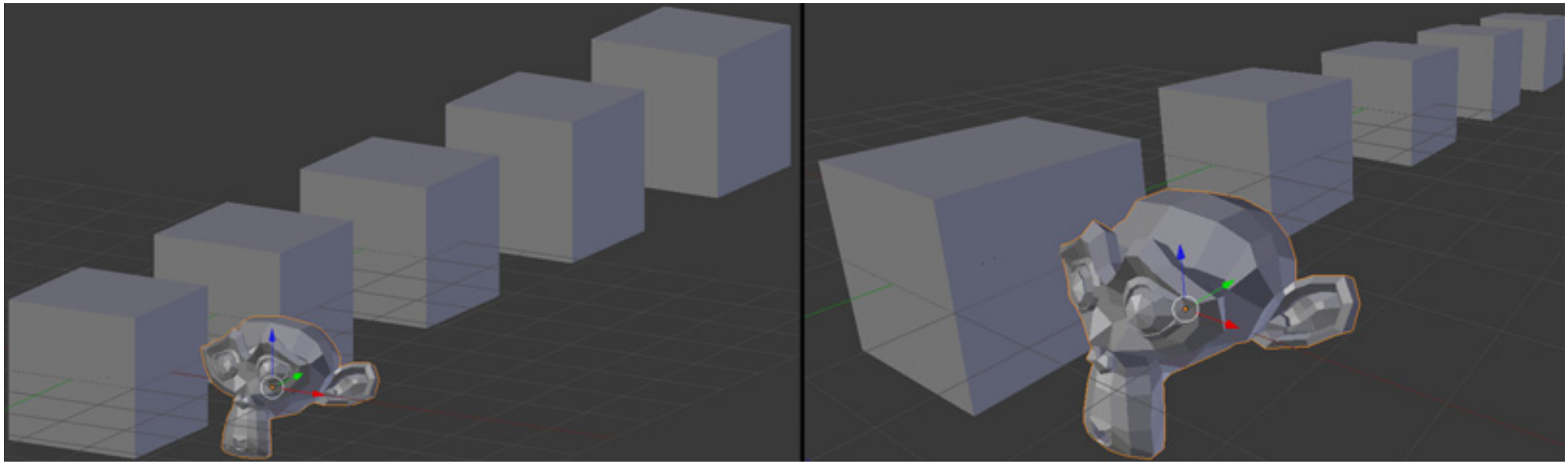
C: surface color

DIFFERENTIABLE

Rasterization pipeline



Two main types of projection

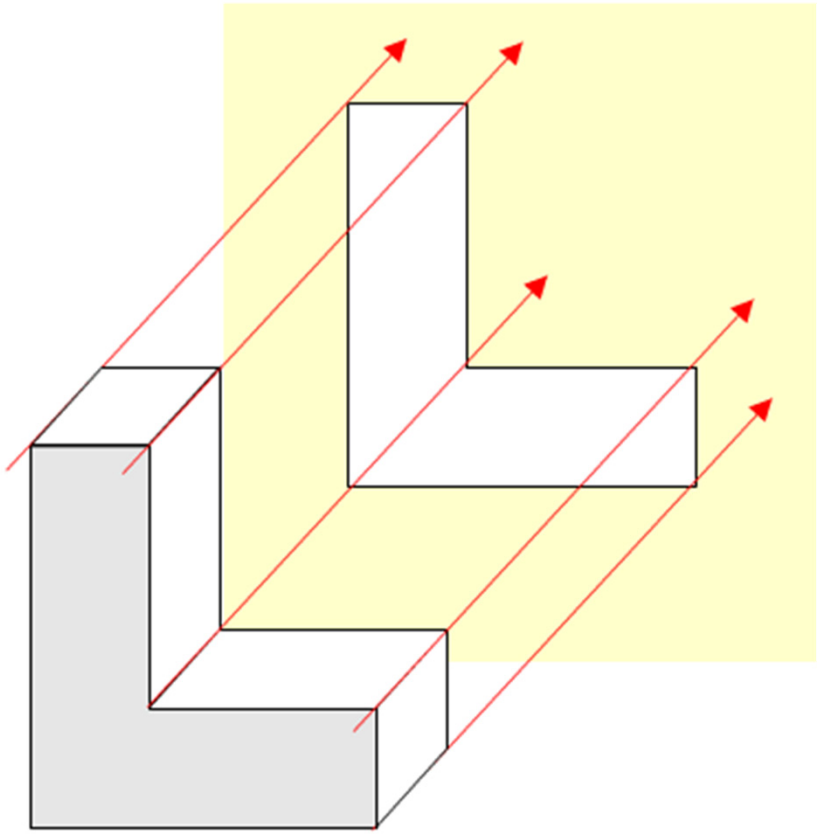


orthographic projection **perspective projection**

The **perspective projection** makes distant objects look smaller.

Orthographic Projection

Project everything along the Z axis to the $z=0$ plane.



$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

Orthographic Projection

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Orthographic Projection

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix}$$

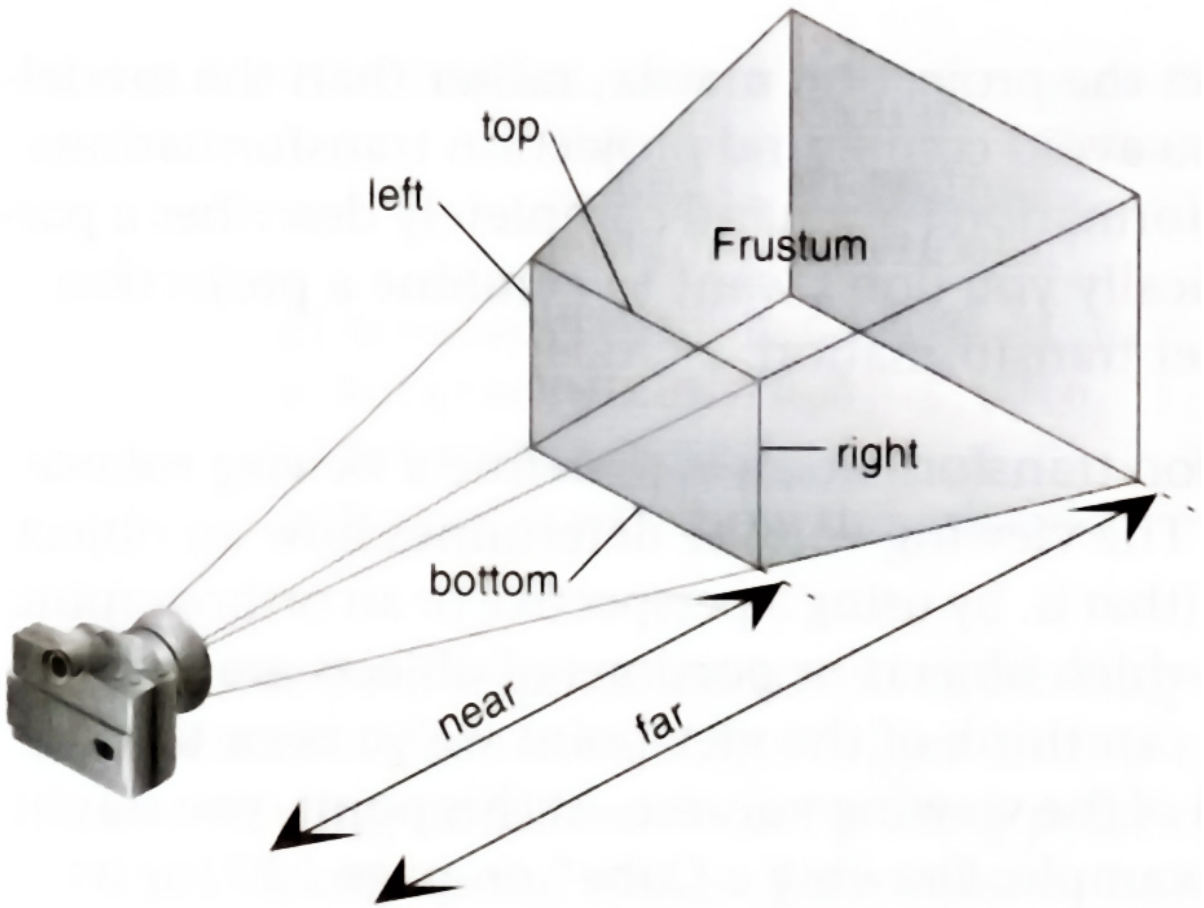
Orthographic Projection

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix}$$

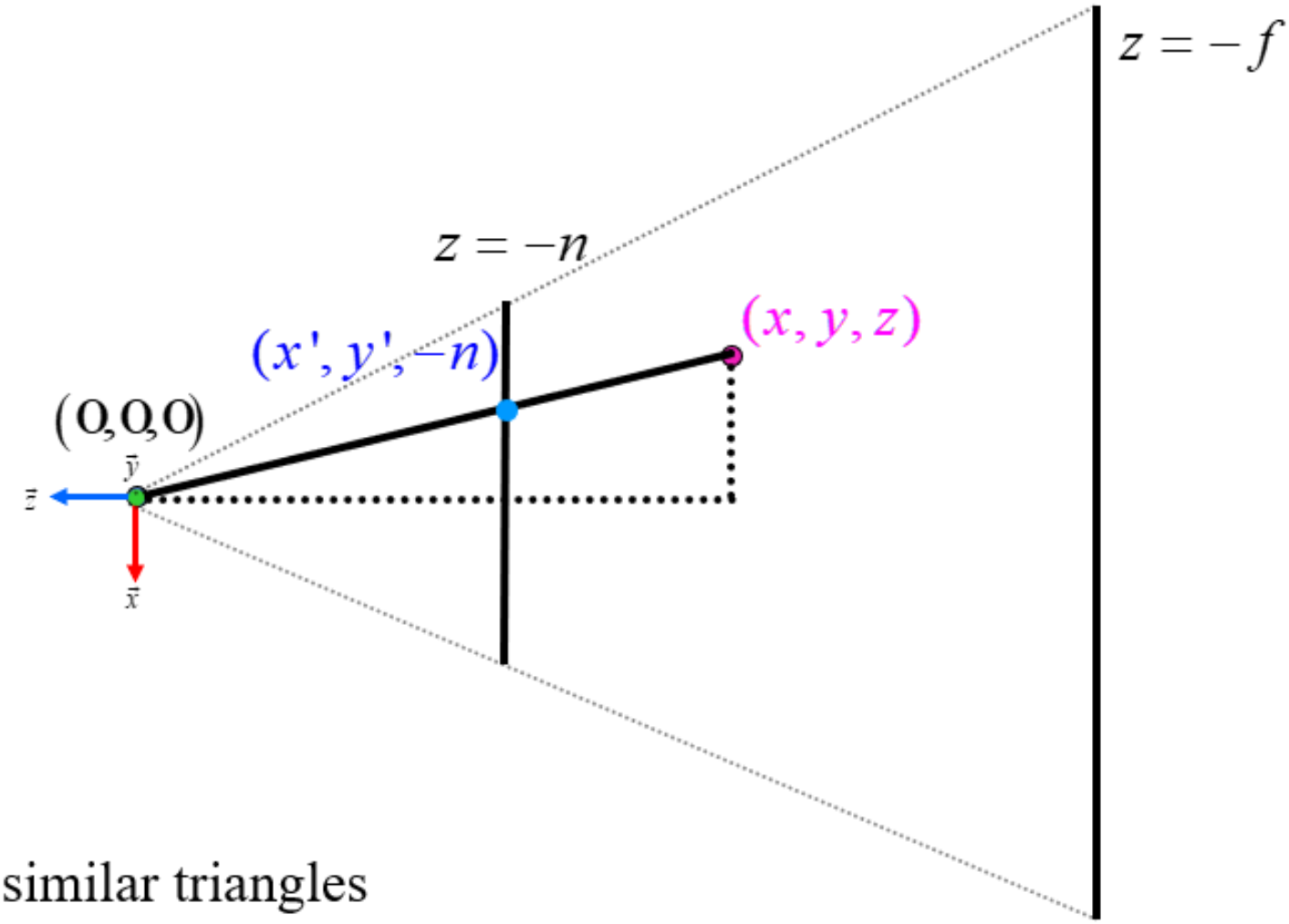
DIFFERENTIABLE

Perspective Projection

Viewing volume has the shape of a **frustum**.



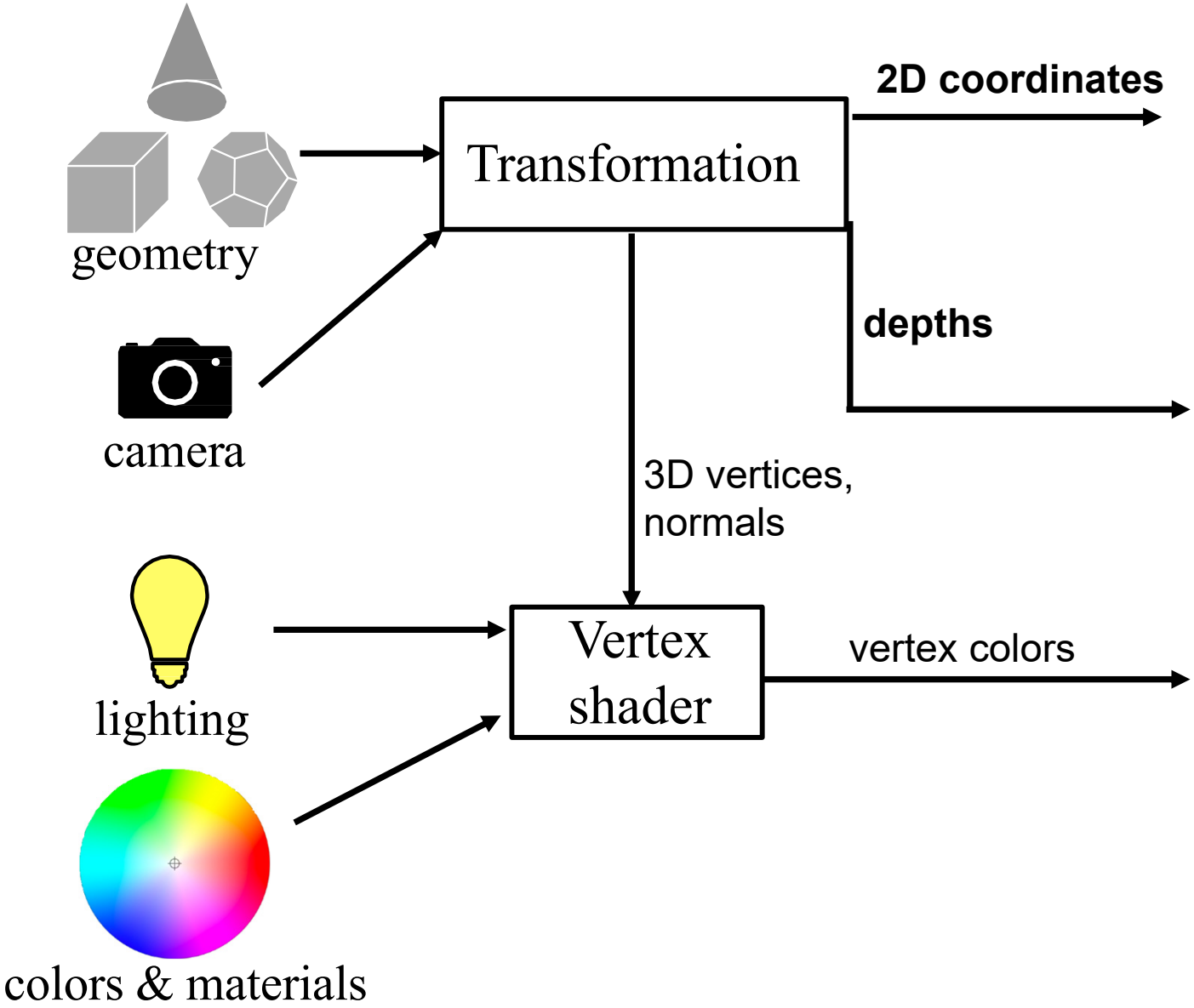
Overhead View of Frustum
- Perspective Projection



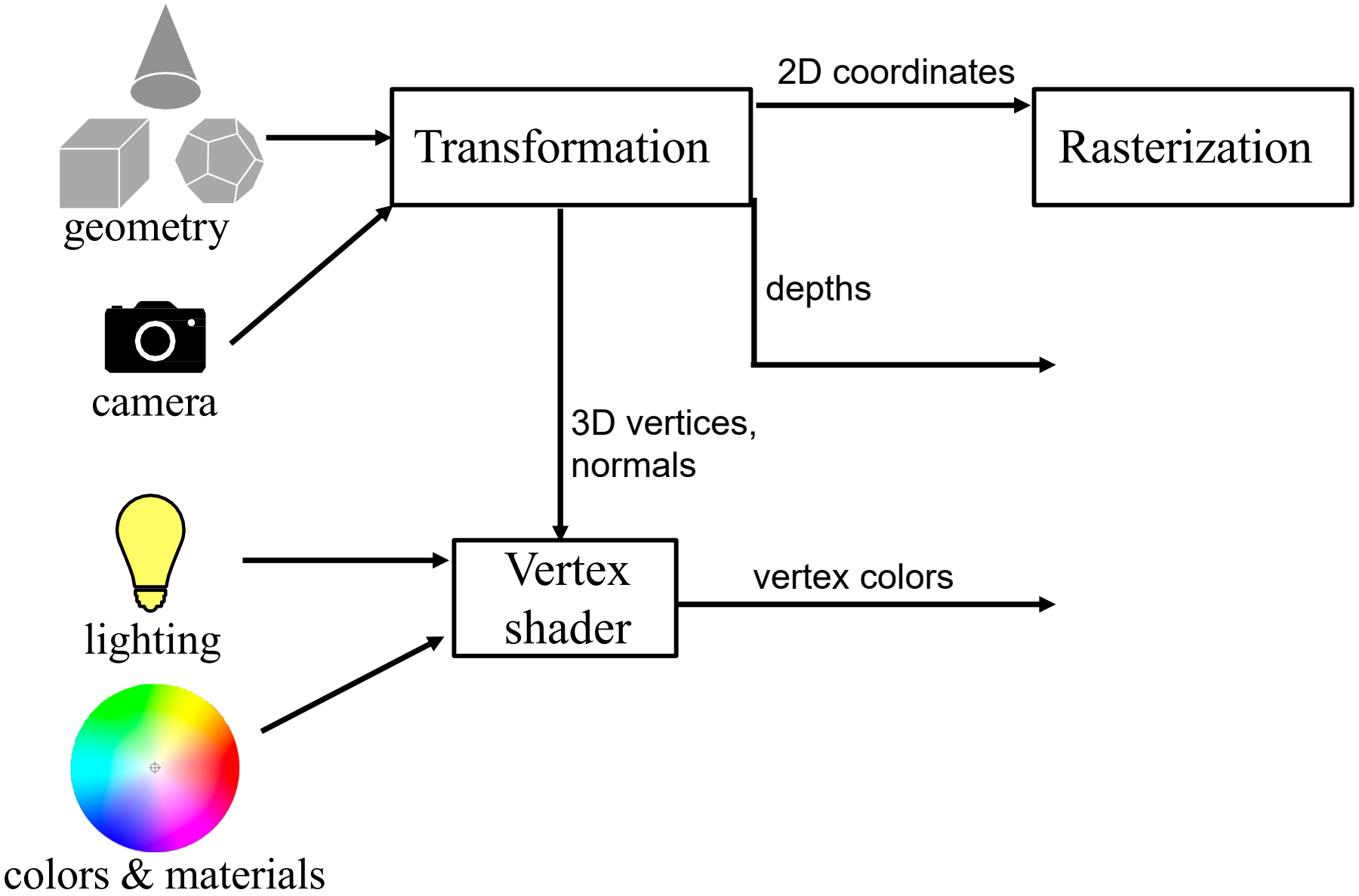
Use similar triangles

$$\frac{x}{z} = \frac{x'}{-n} \Rightarrow x' = -\frac{n \cdot x}{z} \qquad \frac{y}{z} = \frac{y'}{-n} \Rightarrow y' = -\frac{n \cdot y}{z}$$

Rasterization pipeline

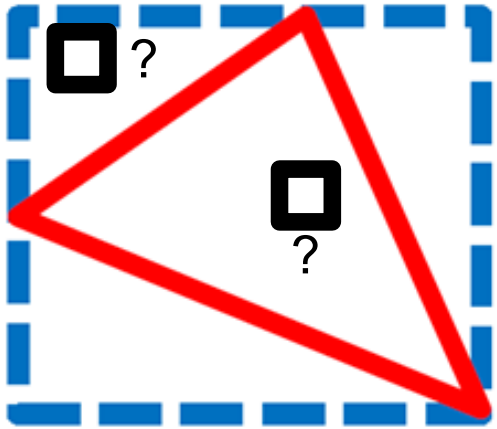


Rasterization pipeline



Rasterization

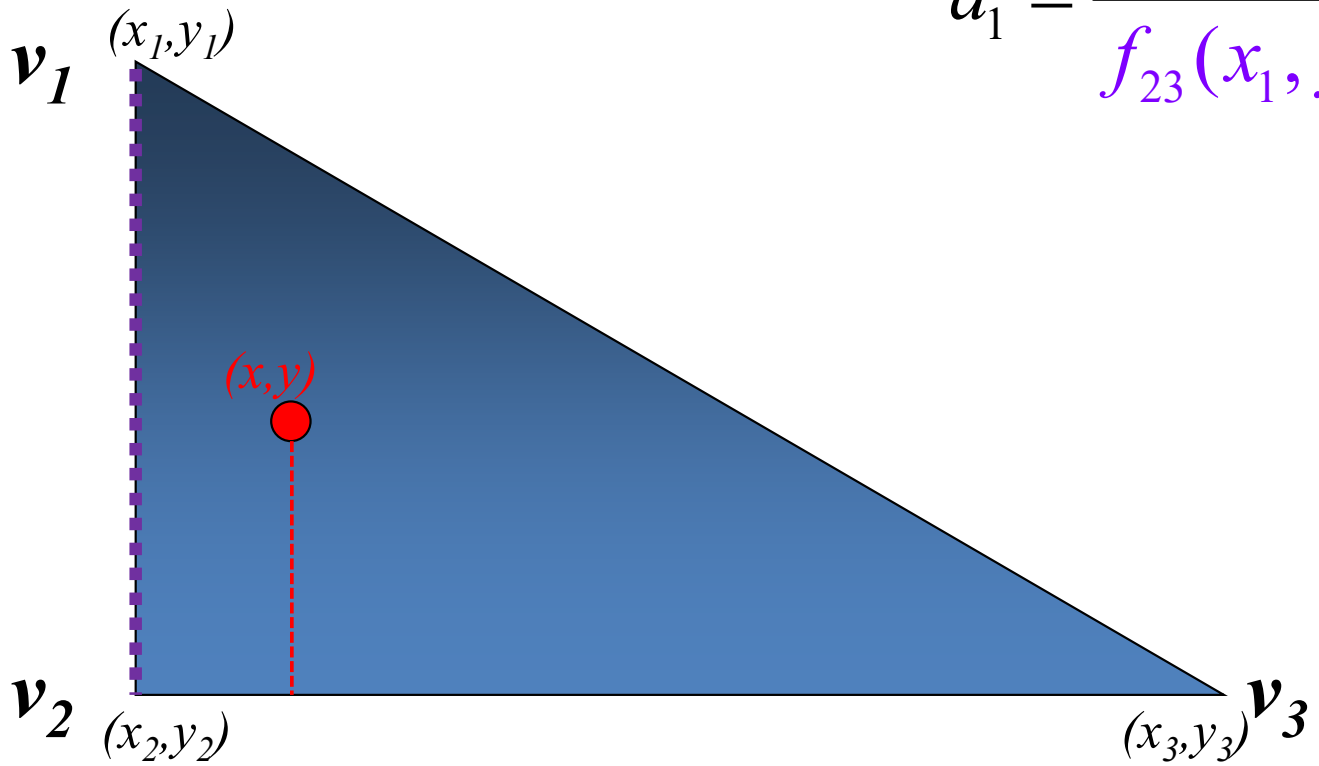
Check if each pixel (its center point) is inside a triangle or not ... **a binary test**



Traditional rasterization

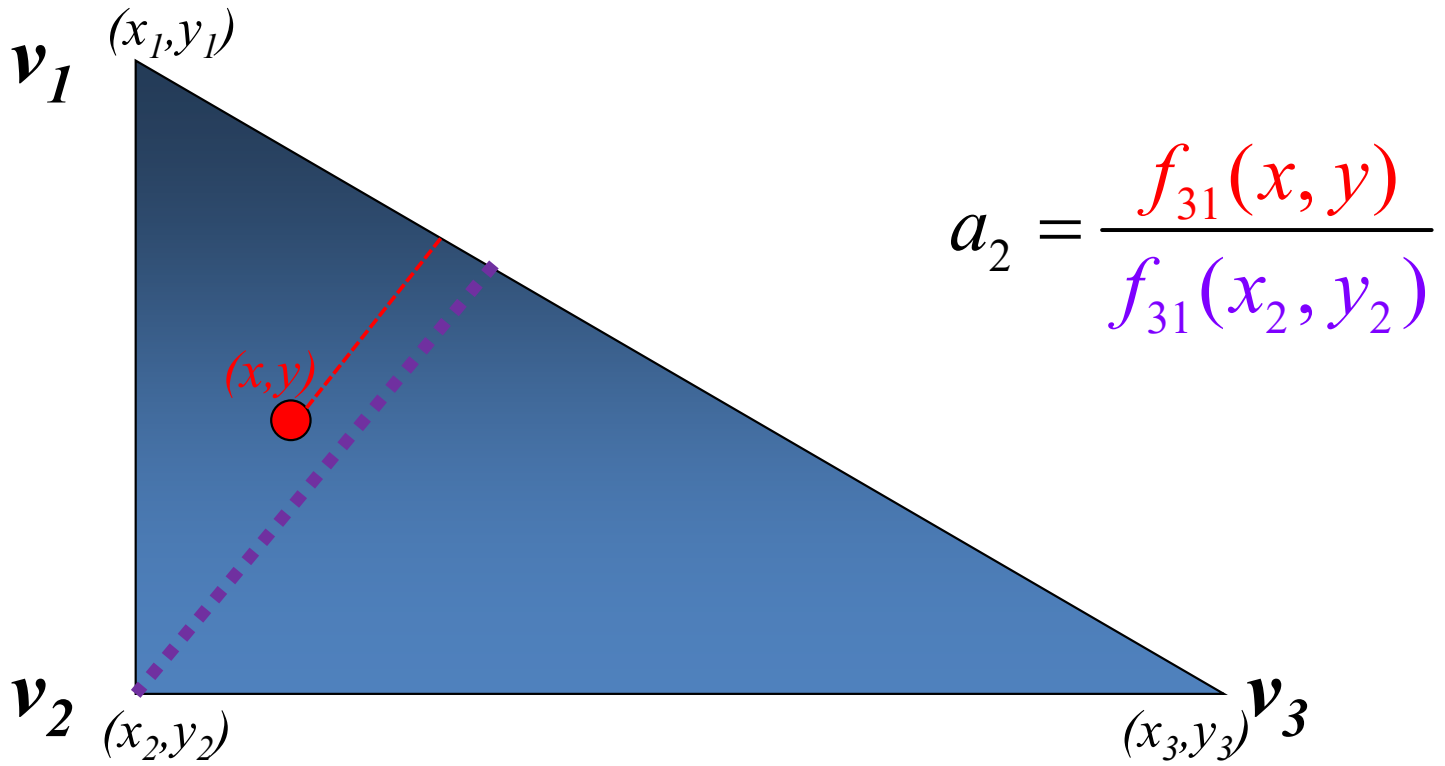
Check if a point is inside a triangle using **barycentric coordinates** and implicit line equations for edges:

$$a_1 = \frac{f_{23}(x, y)}{f_{23}(x_1, y_1)}$$



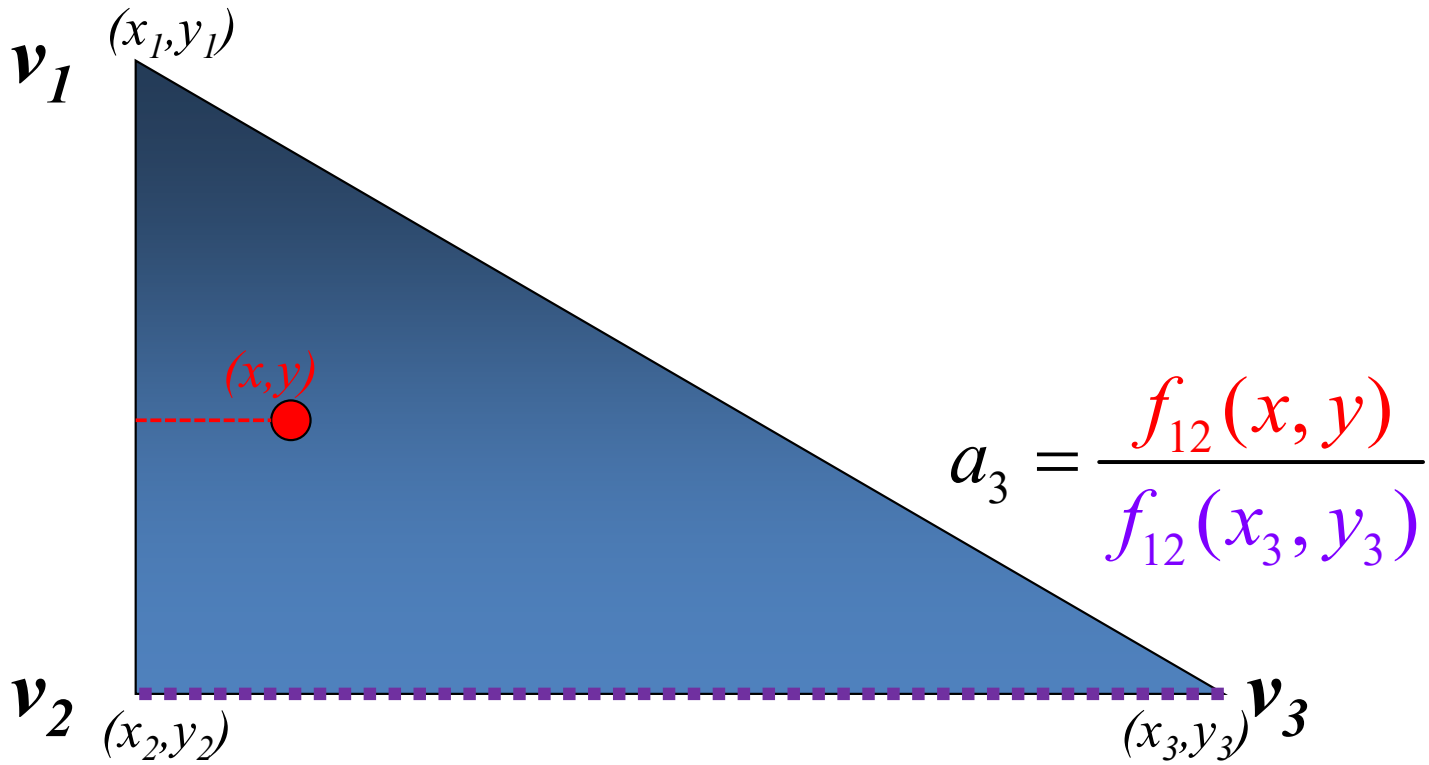
Traditional rasterization

Check if a point is inside a triangle using **barycentric coordinates** and implicit line equations for edges:



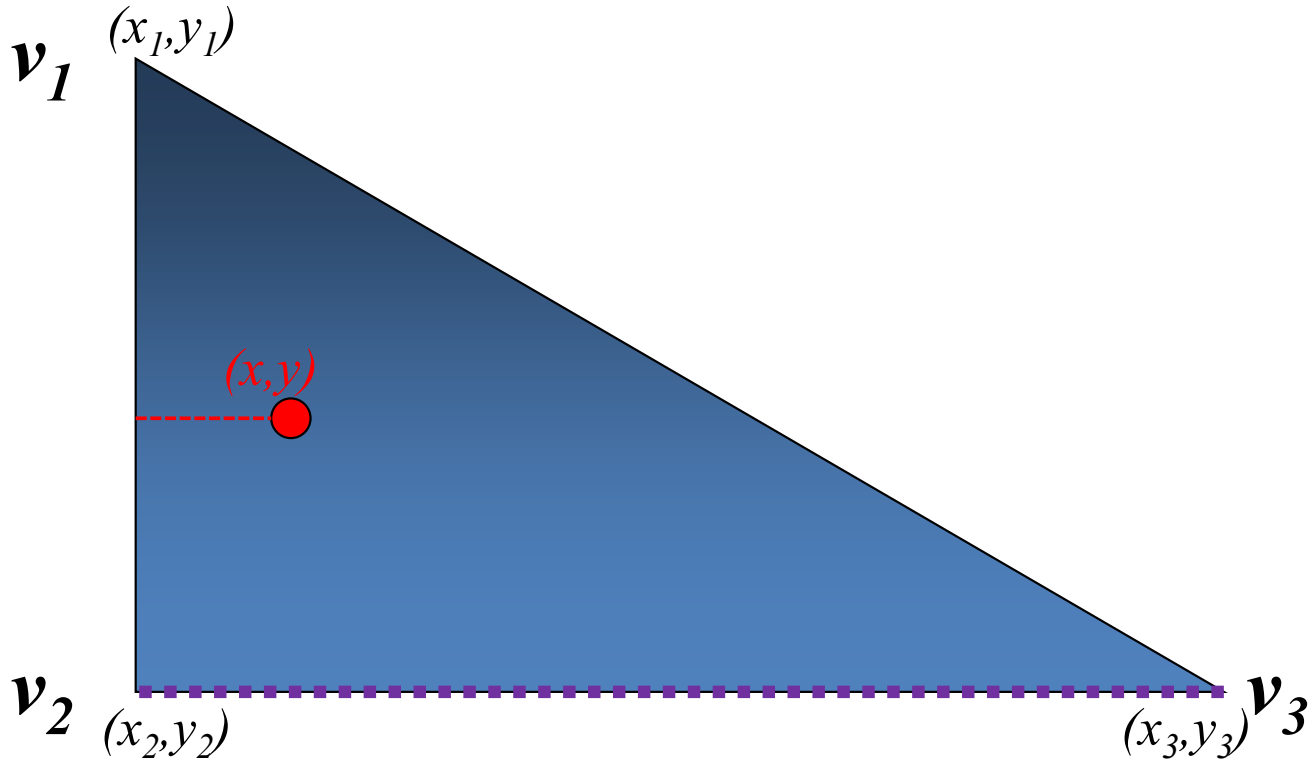
Traditional rasterization

Check if a point is inside a triangle using **barycentric coordinates** and implicit line equations for edges:



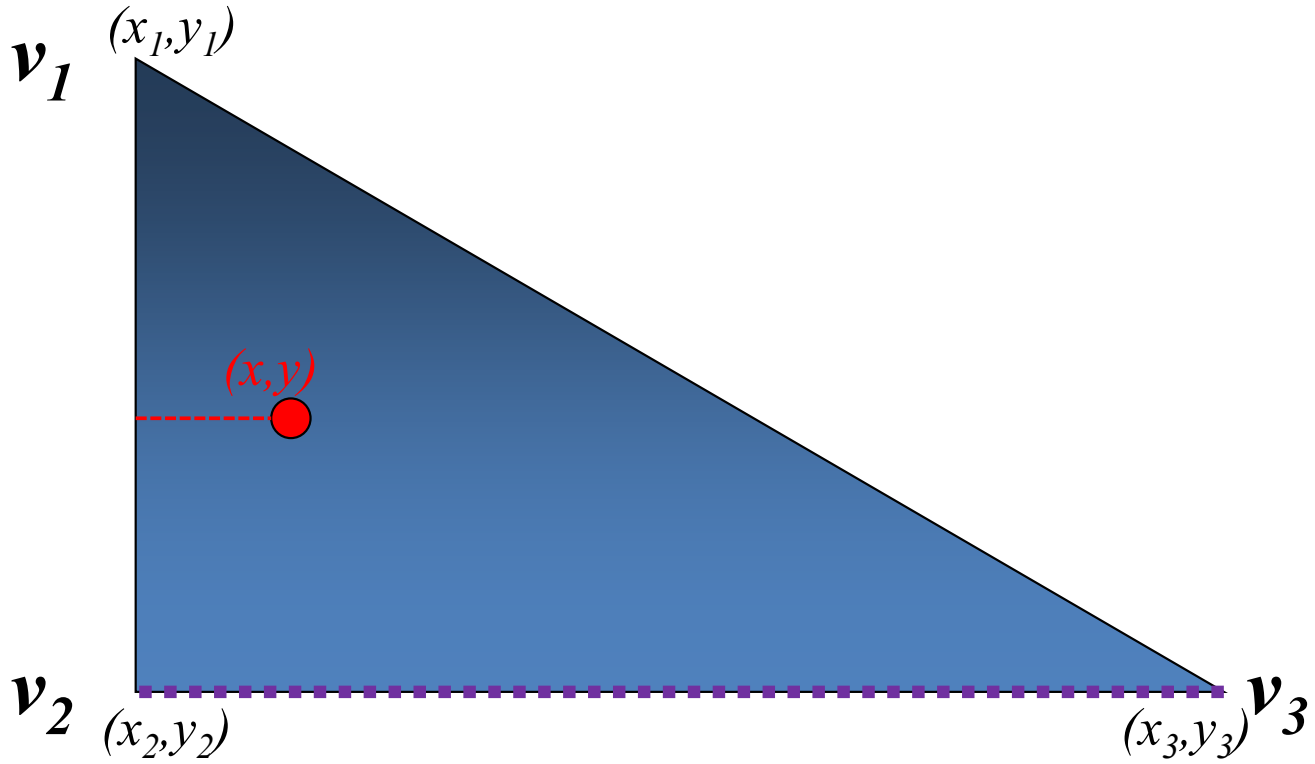
Traditional rasterization

If $0 < a_1 < 1,\ 0 < a_2 < 1,\ 0 < a_3 < 1$
the pixel is inside *(send the fragment to the next stage!)*



Traditional rasterization

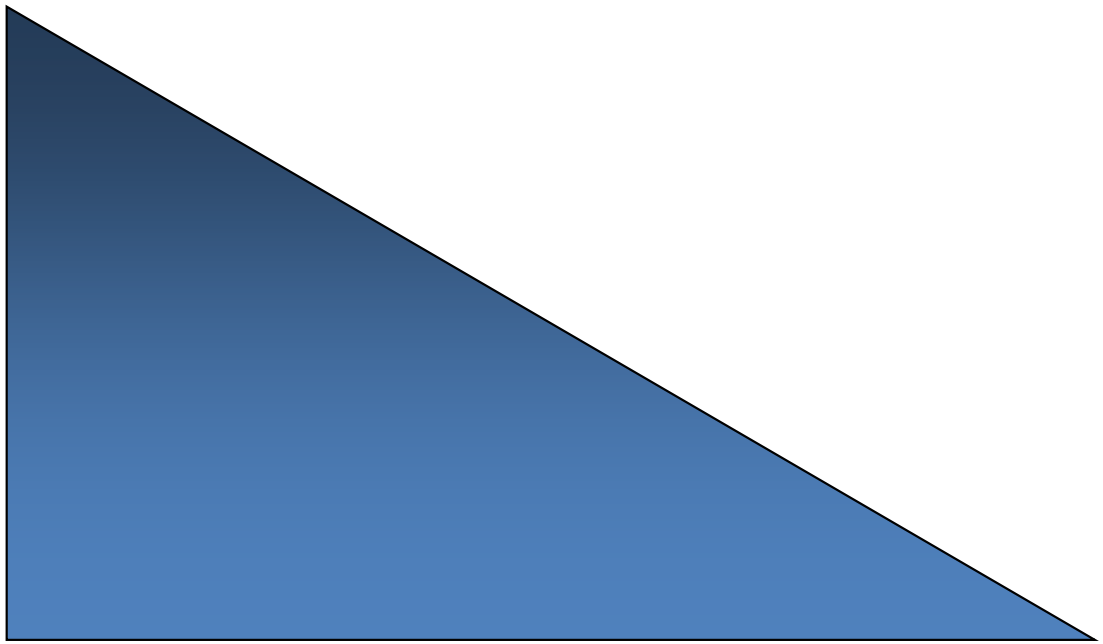
If $0 < a_1 < 1,\ 0 < a_2 < 1,\ 0 < a_3 < 1$
the pixel is inside *(send the fragment to the next stage!)*



NON-DIFFERENTIABLE

Soft Rasterizer

Instead of making a hard decision (pixel belongs to the triangle or not), **soften it**!



Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning, Liu et al., ICCV 2019

Soft Rasterizer

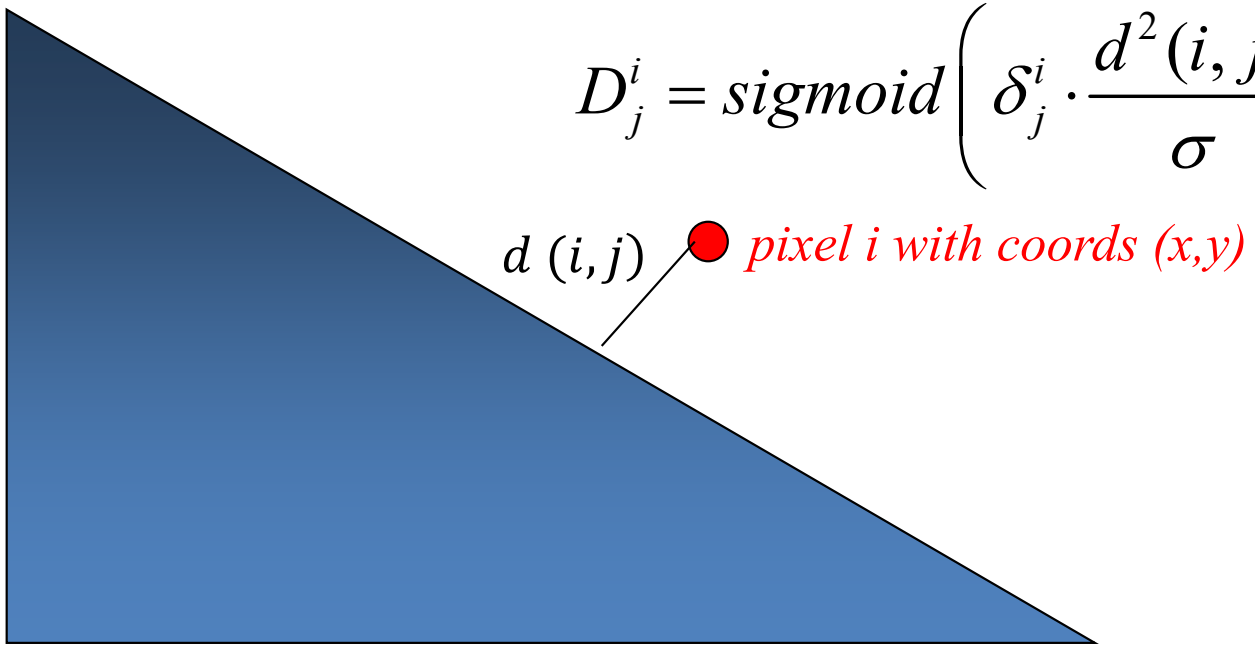
Instead of making a hard decision (pixel belongs to the triangle or not), **soften it**!

Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning, Liu et al., ICCV 2019

Soft Rasterizer

Given the Euclidean distance $d(i,j)$ of pixel i to triangle j, a binary indicator δ (values 1 / -1) returning whether the pixel belongs to the triangle or not, and a hyperparameter σ:

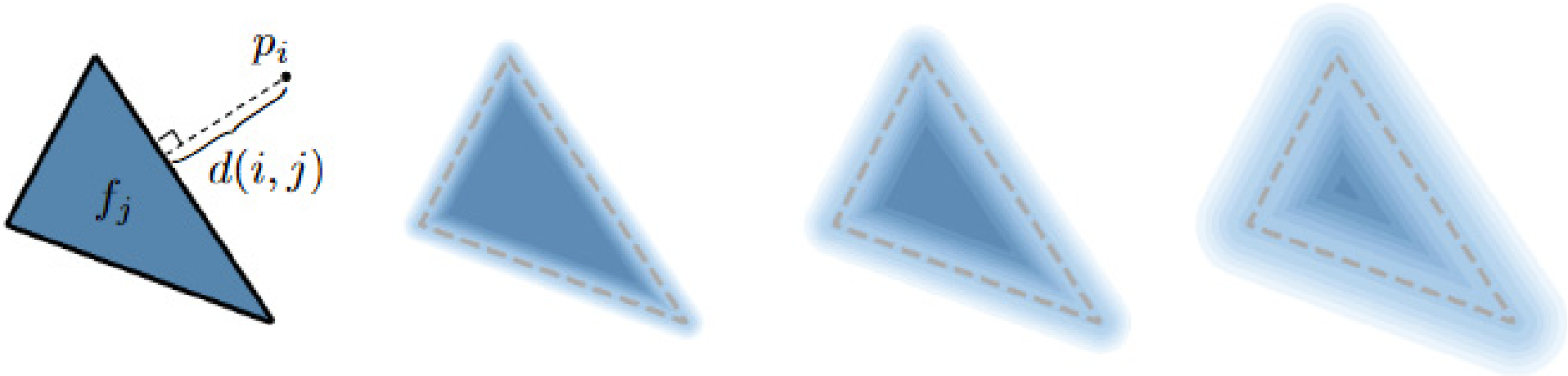
$$D_j^i = sigmoid\left(\delta_j^i \cdot \frac{d^2(i,j)}{\sigma}\right)$$



Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning, Liu et al., ICCV 2019

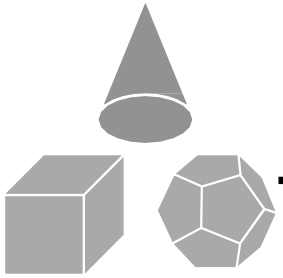
Soft Rasterizer

Effect of hyperparameter σ:



Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning, Liu et al., ICCV 2019

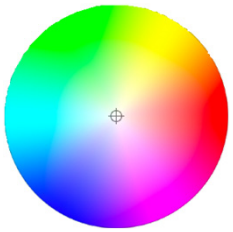
Rasterization pipeline



geometry

camera

lighting



colors & materials

Transformation

2D coordinates

Rasterization

fragments

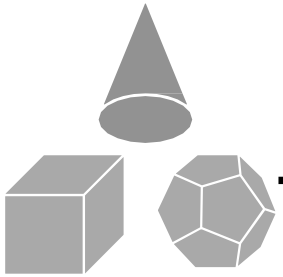
depths

3D vertices,
normals

Vertex
shader

vertex colors

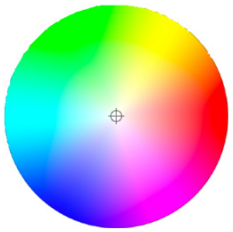
Rasterization pipeline



geometry

camera

lighting



colors & materials

Transformation

2D coordinates

Rasterization

fragments

depths

Z-buffering

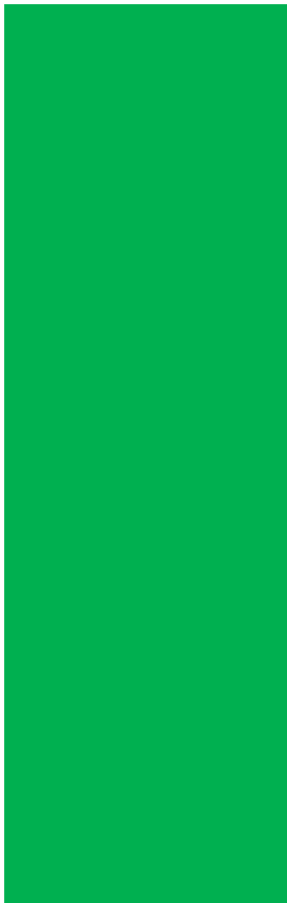
3D vertices,
normals

Vertex
shader

vertex colors

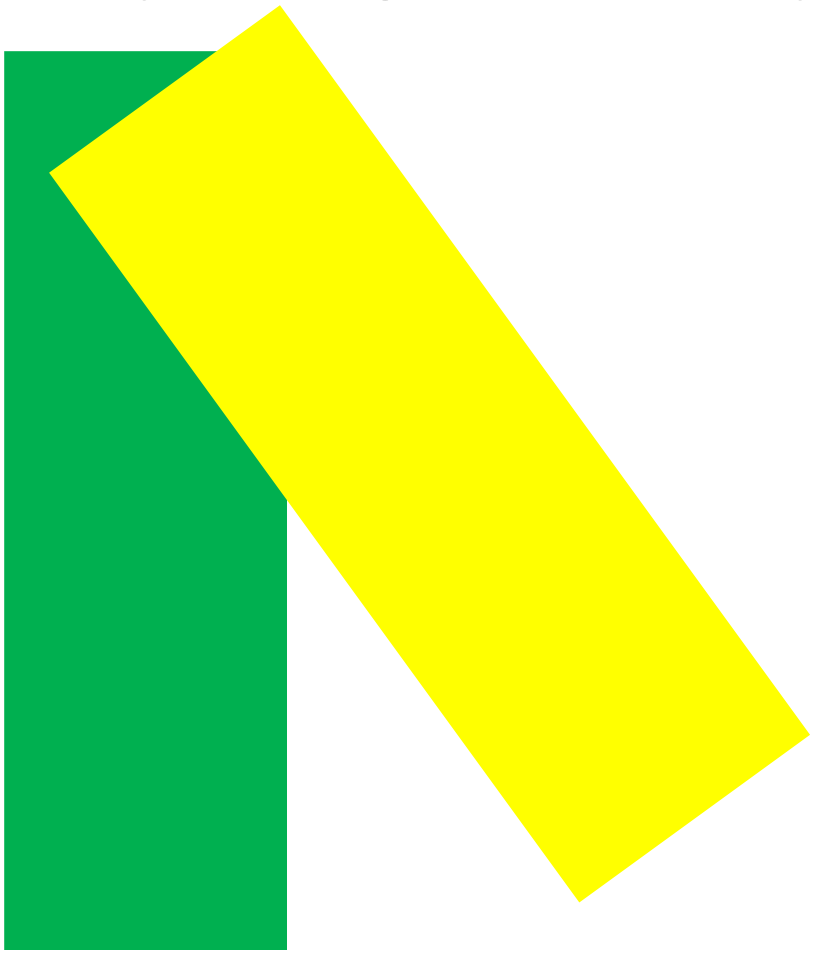
Depth test

After rasterizing each polygon, we obtain **fragments** (pixels + depth + colors)... we need to decide which fragments are in front of others depending on their depth



Depth test

After rasterizing each polygon, we obtain **fragments** (pixels + depth + colors)... we need to decide which fragments are in front of others depending on their depth

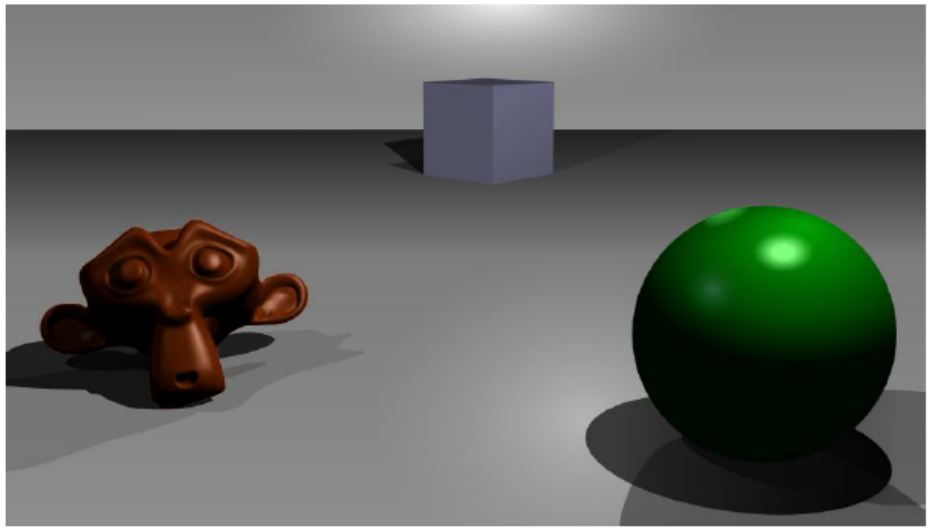


Depth test

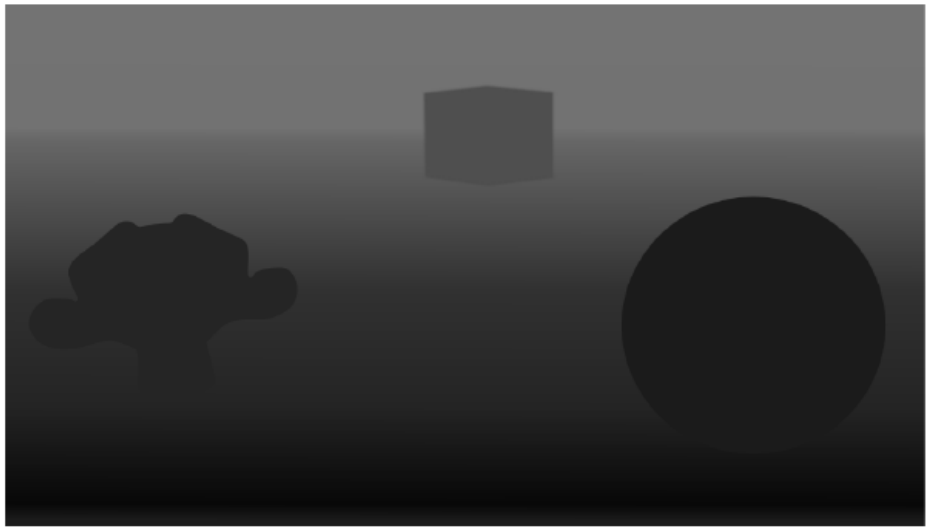
After rasterizing each polygon, we obtain **fragments** (pixels + depth + colors)... we need to decide which fragments are in front of others depending on their depth

The Z-buffer (or depth buffer)

Stores depth for each fragment *[we **do not throw out the z-coordinate** during our pipeline]*



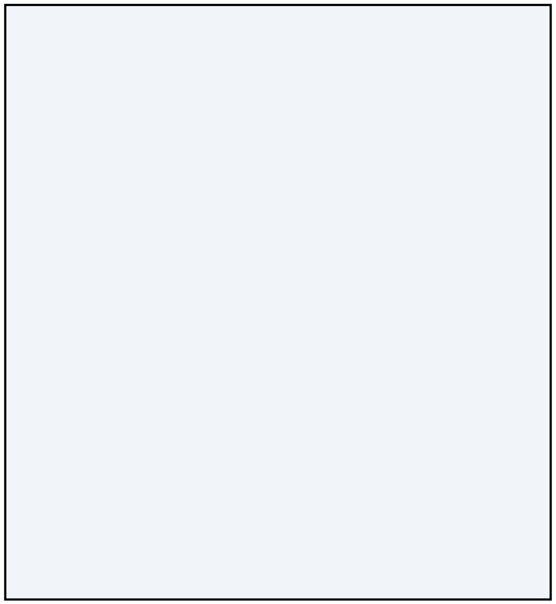
Color buffer



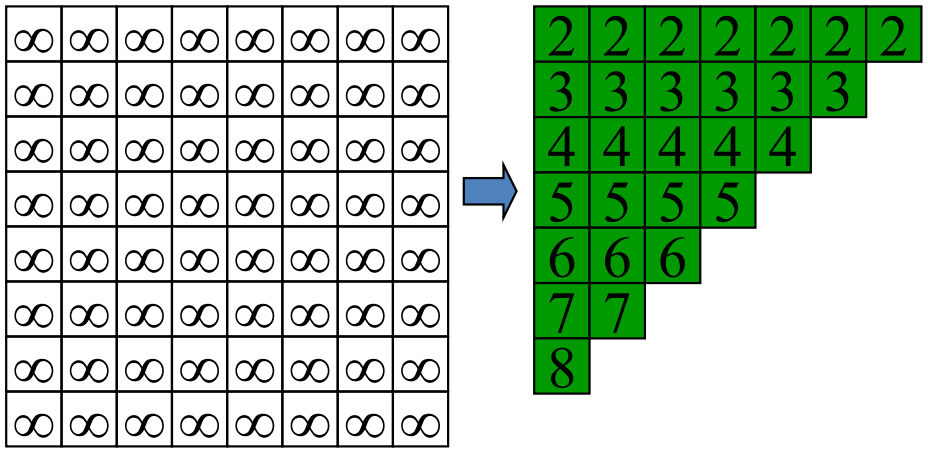
Depth buffer

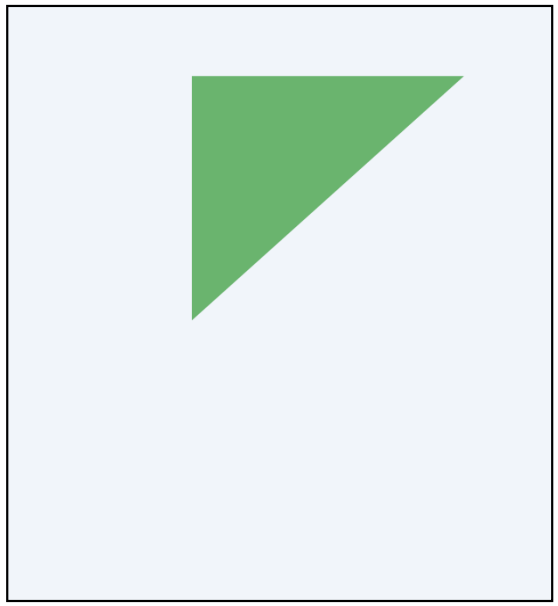
Z-Buffer Algorithm

∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞

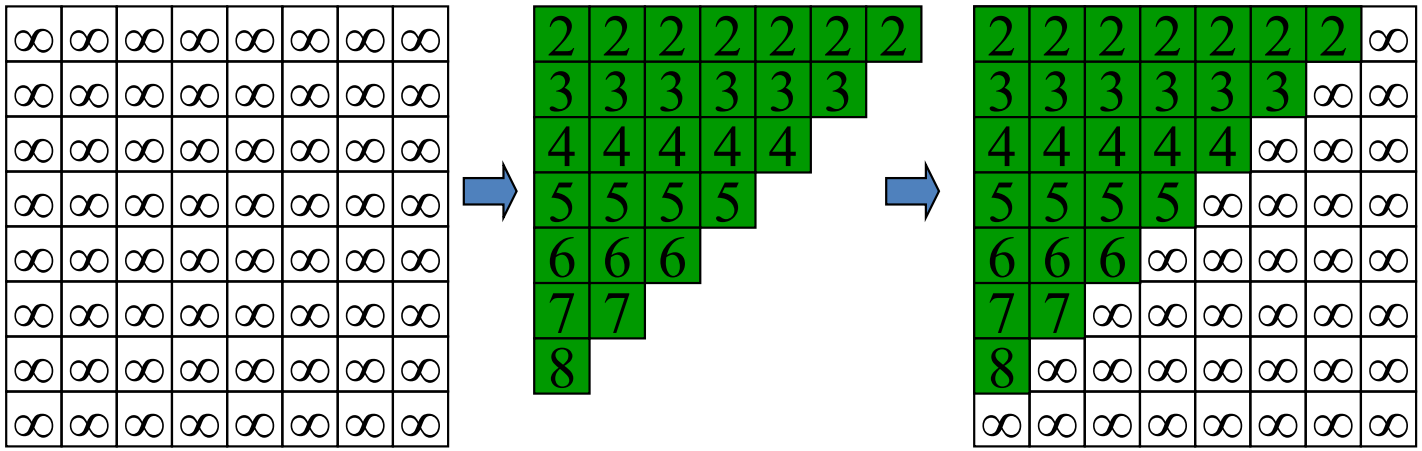


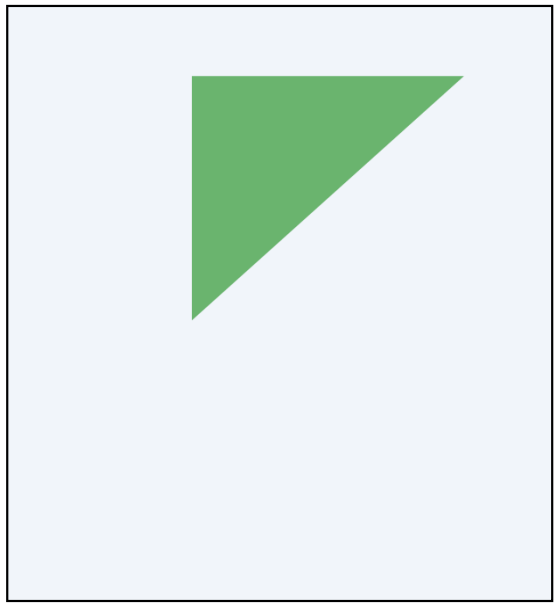
Z-Buffer Algorithm



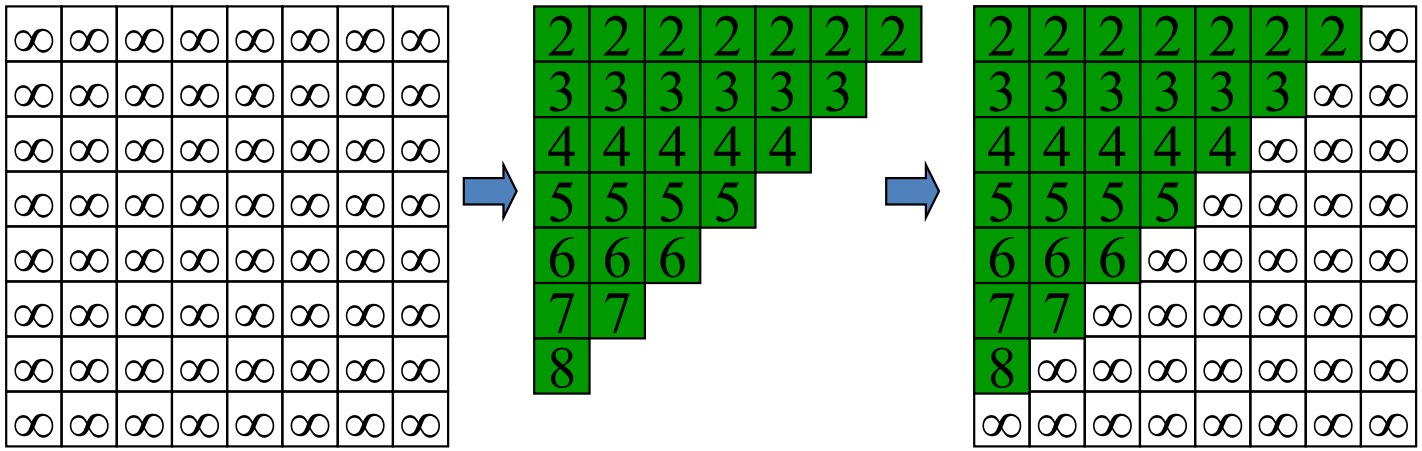


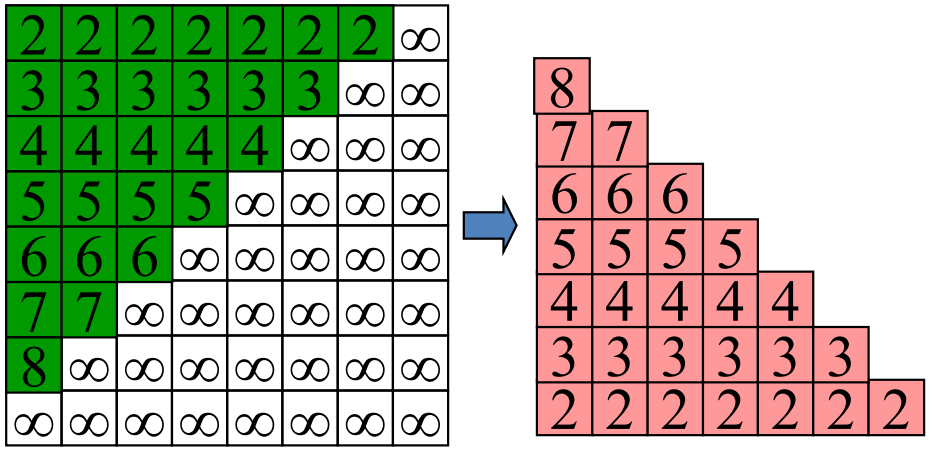
Z-Buffer Algorithm

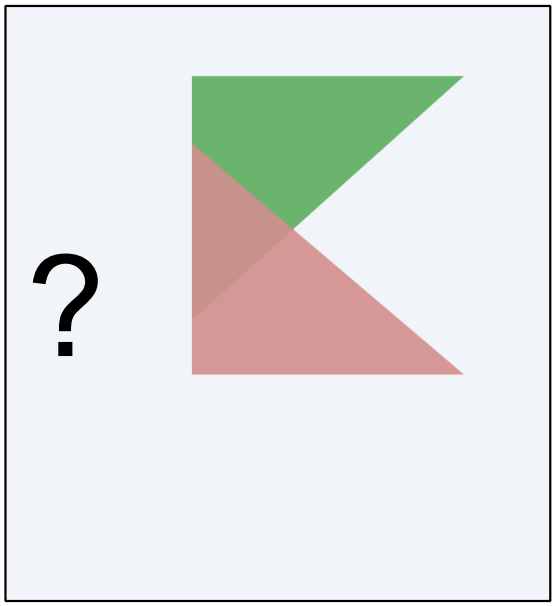




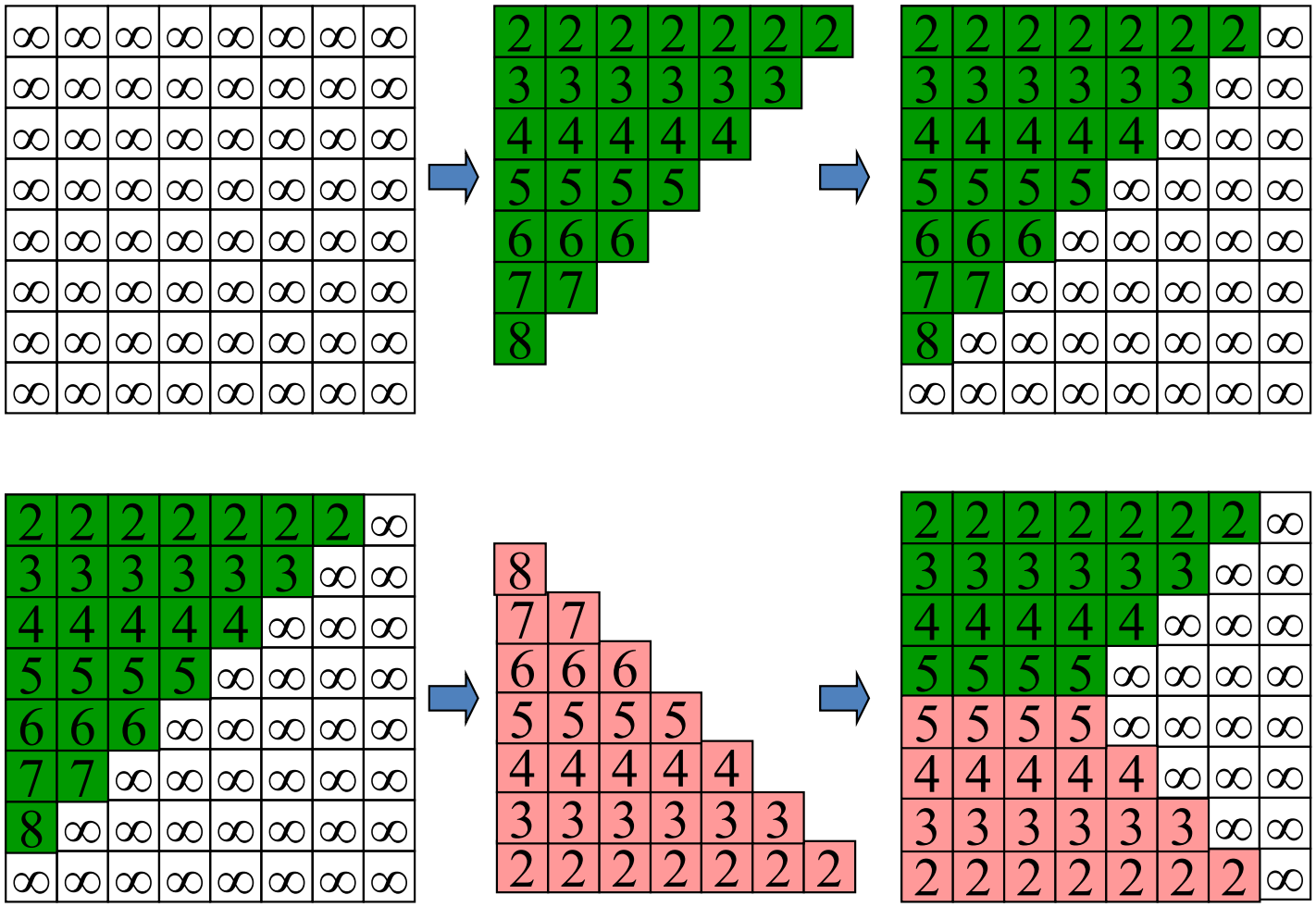
Z-Buffer Algorithm

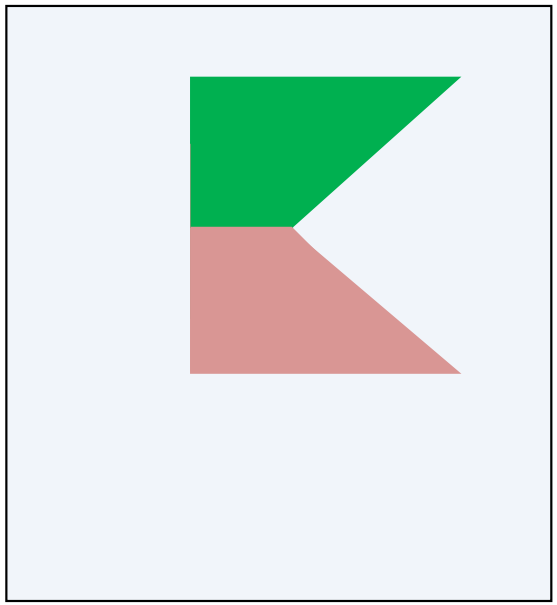






Z-Buffer Algorithm





The Z-Buffer Algorithm [Z-test]

For each rasterized primitive j (triangle, lines)

Is the **depth** z_i^j **of fragment** i about to be drawn **greater** than the **recorded depth** in the z-buffer?

If yes, throw it away!

Otherwise, draw pixel and **update depth!**

```
if ( z_i^j < z_buffer[x, y])
{
  z_buffer[x,y]= z_i^j;
  I[x,y]=c[x,y];
}
```

The Z-Buffer Algorithm [Z-test]

For each rasterized primitive j (triangle, lines)

Is the **depth** z_i^j **of fragment** i about to be drawn **greater** than the **recorded depth** in the z-buffer?

If yes, throw it away!

Otherwise, draw pixel and **update depth!**

```
if ( z_i^j  < z_buffer[x, y])
{
  z_buffer[x,y]= z_i^j;
  I[x,y]=c[x,y];
}
```

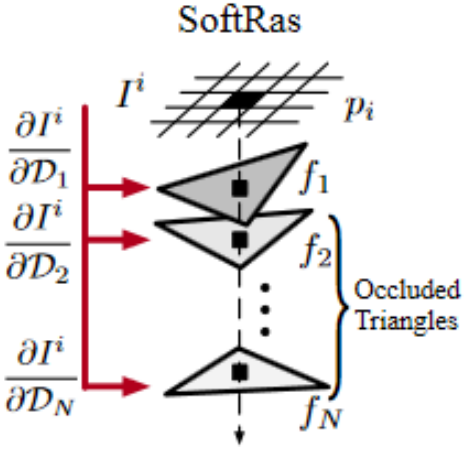
NON-DIFFERENTIABLE

Soft Rasterizer

Given the Euclidean distance $d(i,j)$ of pixel i to triangle j, a binary indicator δ (values -1 / 1) returning whether the pixel belongs to the triangle or not, and a hyperparameter σ:

$$D_j^i = sigmoid\left(\delta_j^i \cdot \frac{d^2(i,j)}{\sigma}\right)$$

Probability of pixel belong to triangle

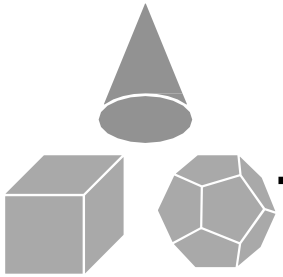


$$w_j^i = \frac{D_j^i \exp(-z_j^i / \gamma)}{\sum_k D_k^i \exp(-z_k^i / \gamma) + const}$$

Probability of pixel being drawing based on its inverse depth

Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning, Liu et al., ICCV 2019

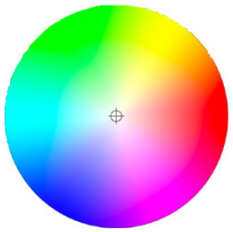
Traditional rasterization pipeline



geometry

camera

lighting



colors & materials

Transformation

2D coordinates

NON-DIFFERENTIABLE

Rasterization

fragments

depths

Z-buffering

3D vertices,
normals

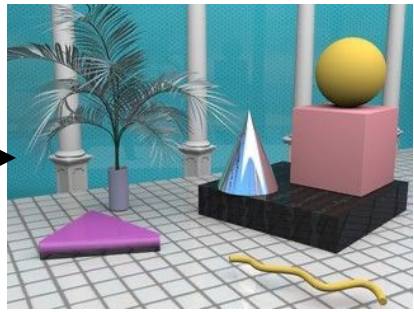
NON-DIFFERENTIABLE

pixels

Vertex
shader

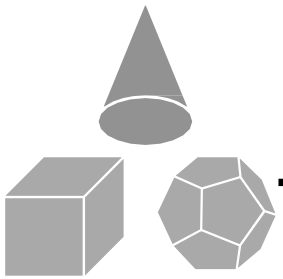
vertex colors

Fragment
shader



output image

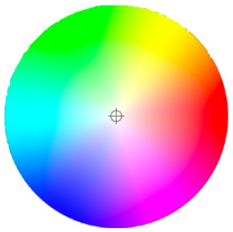
Soft Rasterizer



geometry

camera

lighting



colors & materials

Transformation

2D coordinates

Rasterization

DIFFERENTIABLE
fragments

depths

Z-buffering

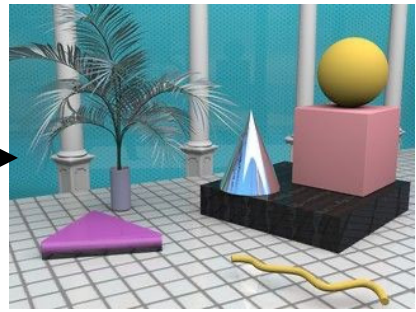
3D vertices,
normals

DIFFERENTIABLE
pixels

Vertex
shader

vertex colors

Fragment
shader

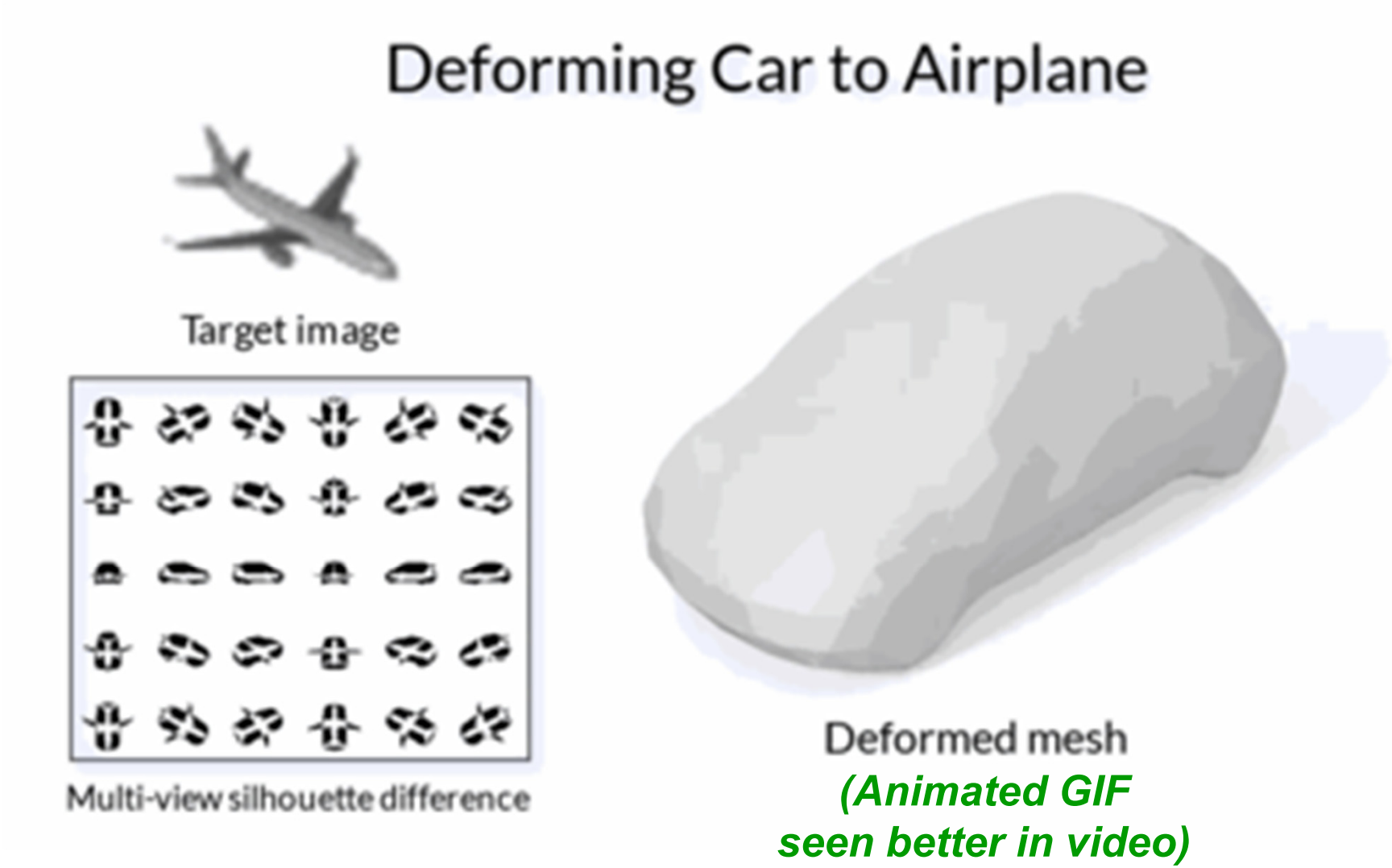


output image

Results: Single Image Reconstruction

Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning, Liu et al., ICCV 2019

Results: Mesh deformation



Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning, Liu et al., ICCV 2019

Differentiable rasterizers

Well suited for differentiable rendering of meshes

Yet, for 3D shape generation we need to generative mesh decoders…
need initialization with predefined templates (e.g. see Pixel2Mesh)

Reminder: Rendering pipelines

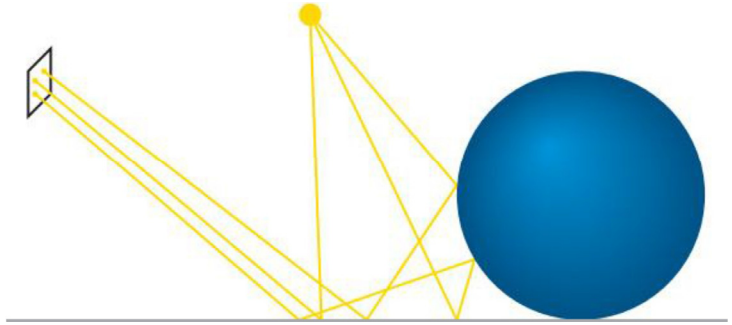
The process of generating a 2D image from a 3D representation

Rasterization

for each polygon
    project it onto 2D plane
    for each pixel in polygon
        shade the pixel

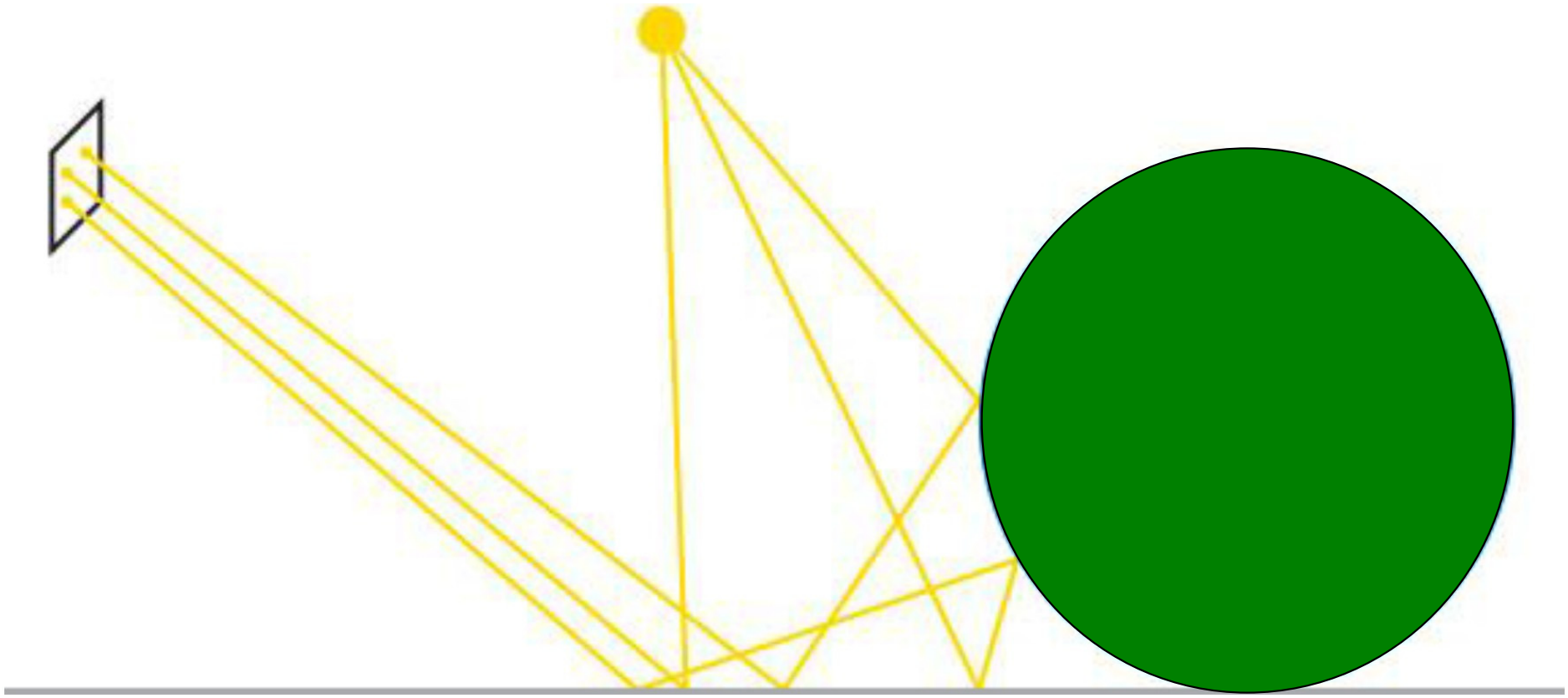
Ray tracing

for each pixel throw ray
    for each object
        calculate intersection
        shade the pixel

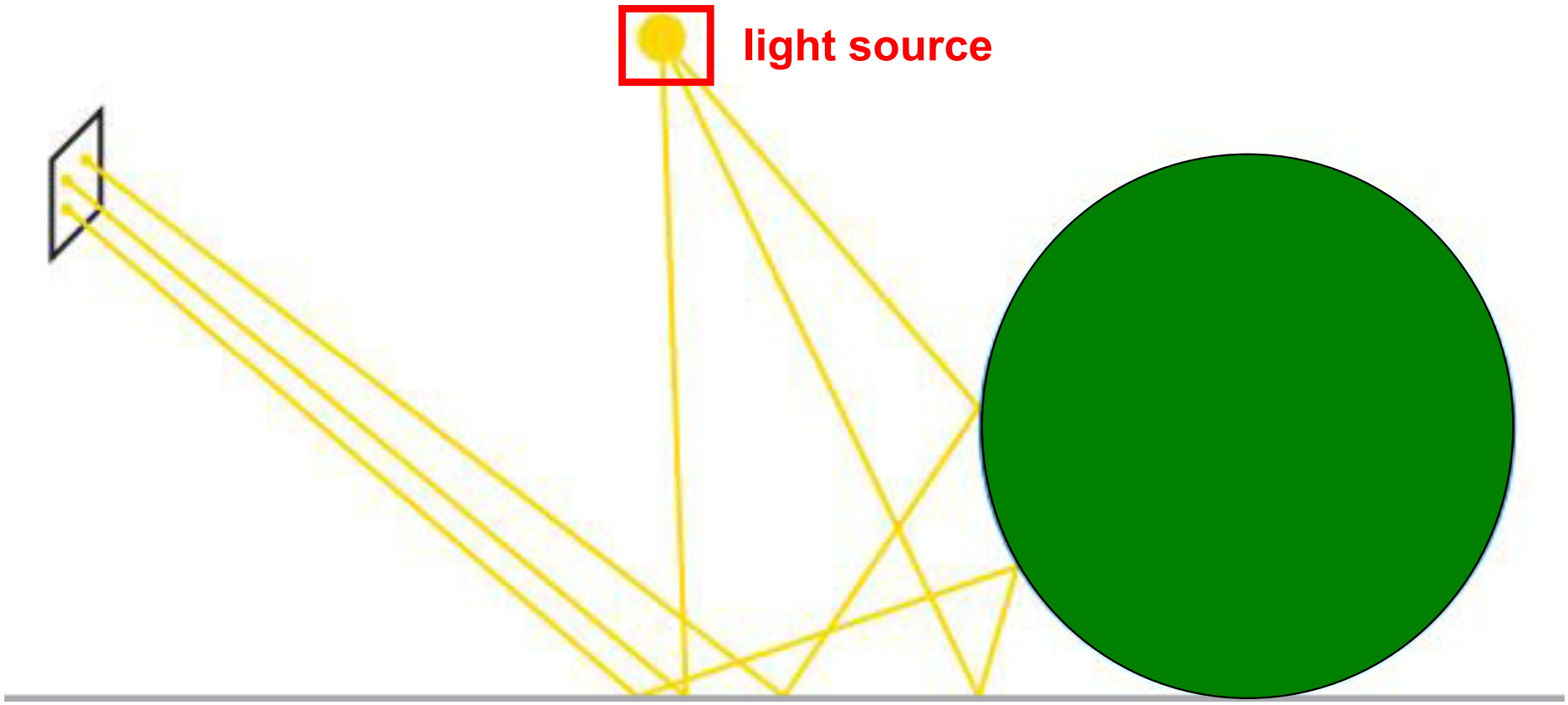


Paths of light

In nature, light sources emit rays of light that bounce on objects. Think of a "ray" as a stream of photons traveling along a path.

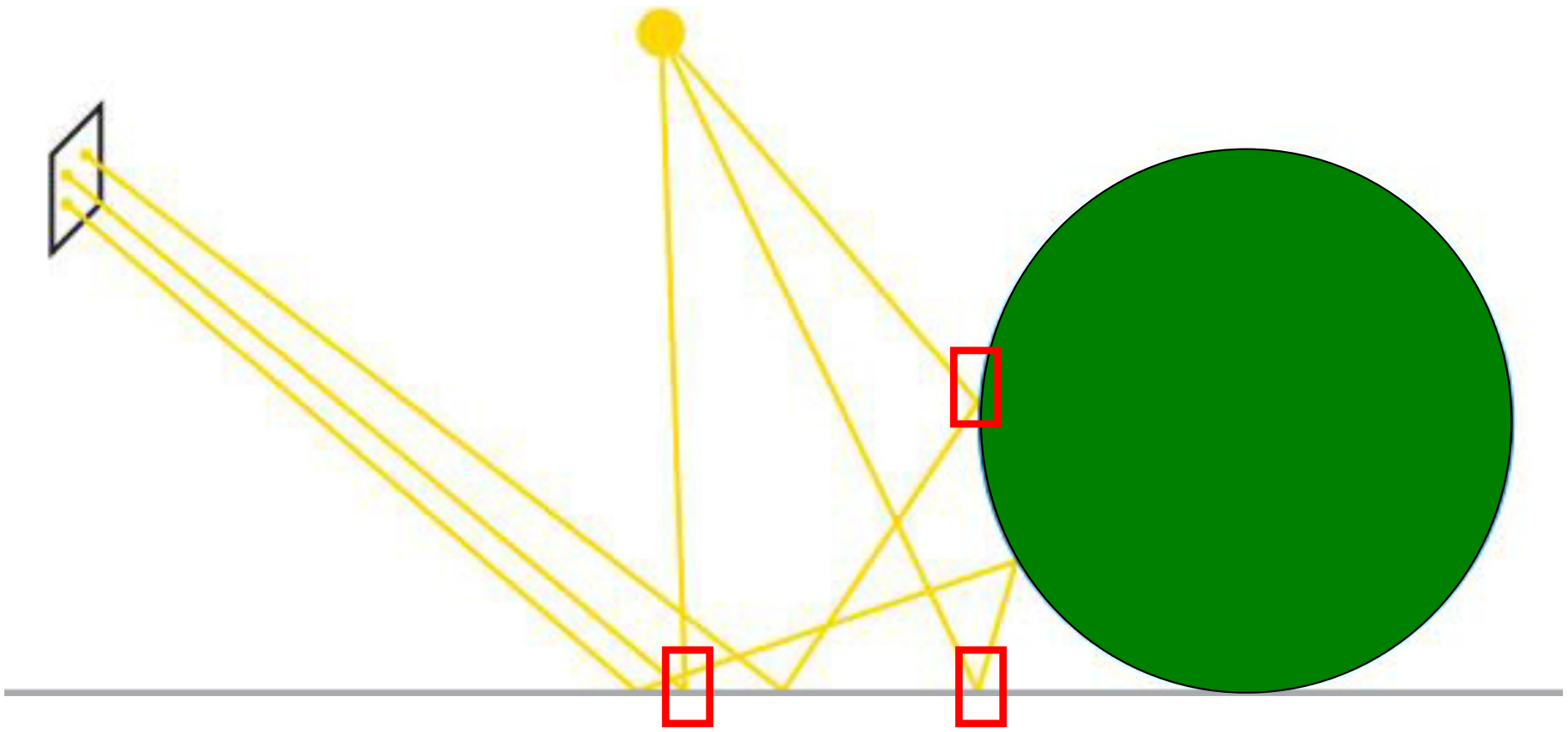


Paths of light



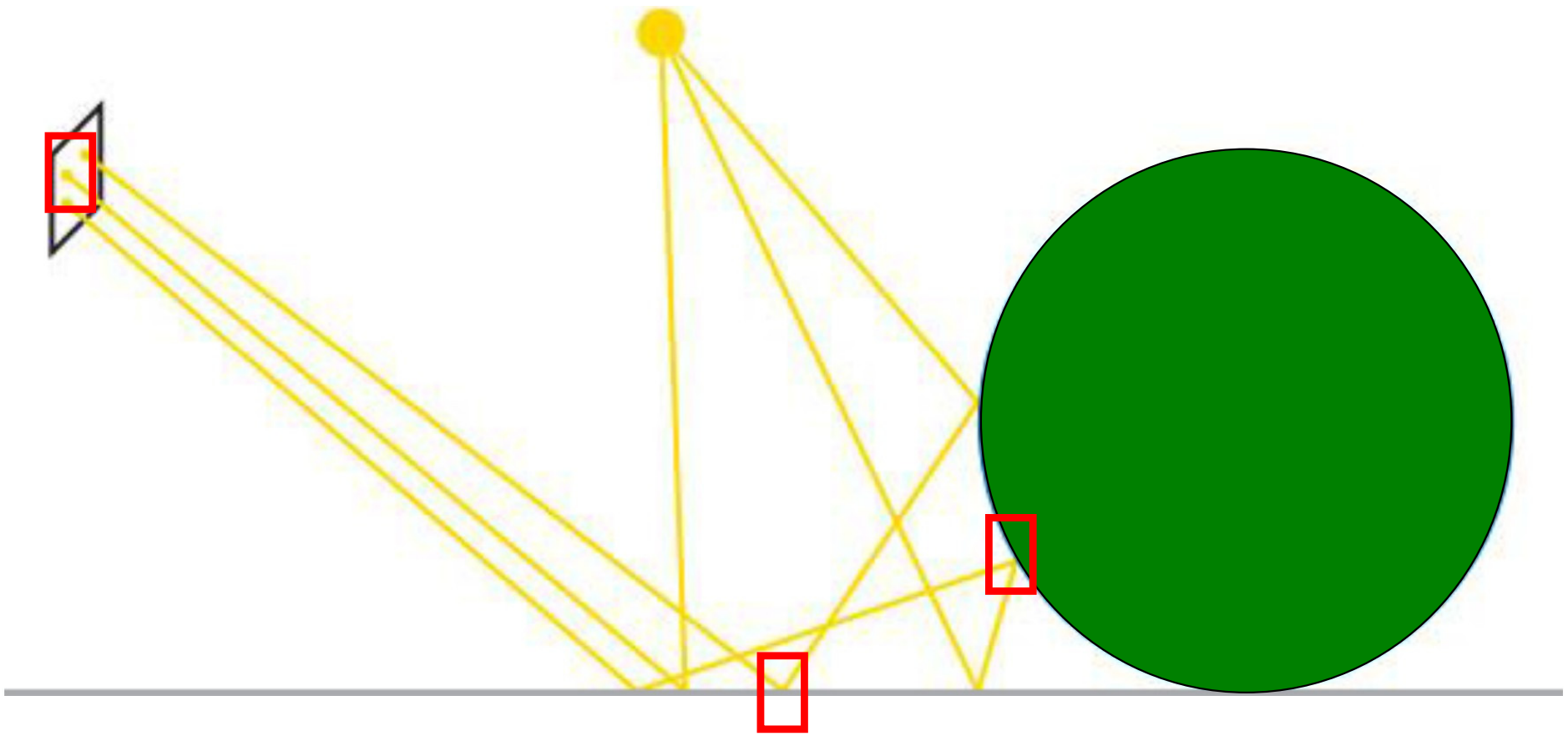
Paths of light

Rays interact with objects in a scene



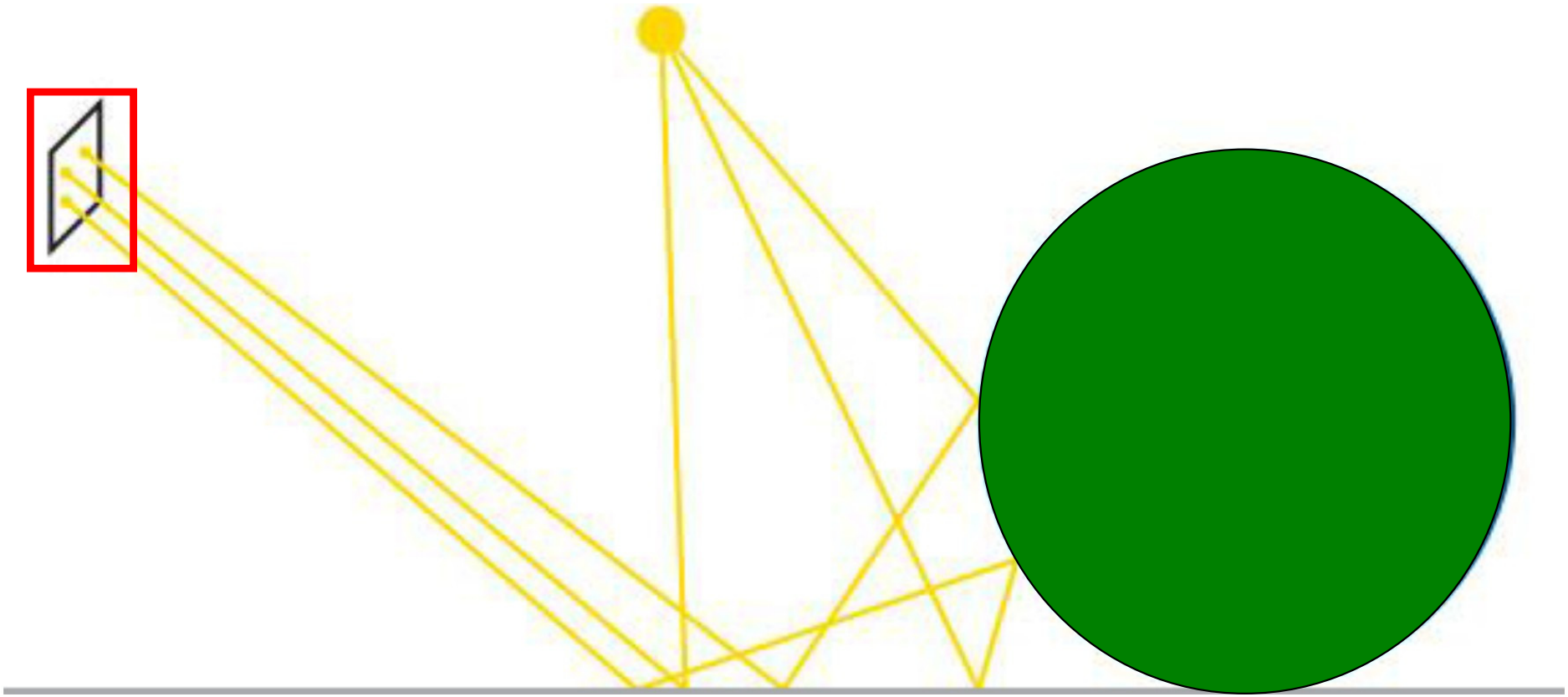
Paths of light

Rays interact with objects in a scene



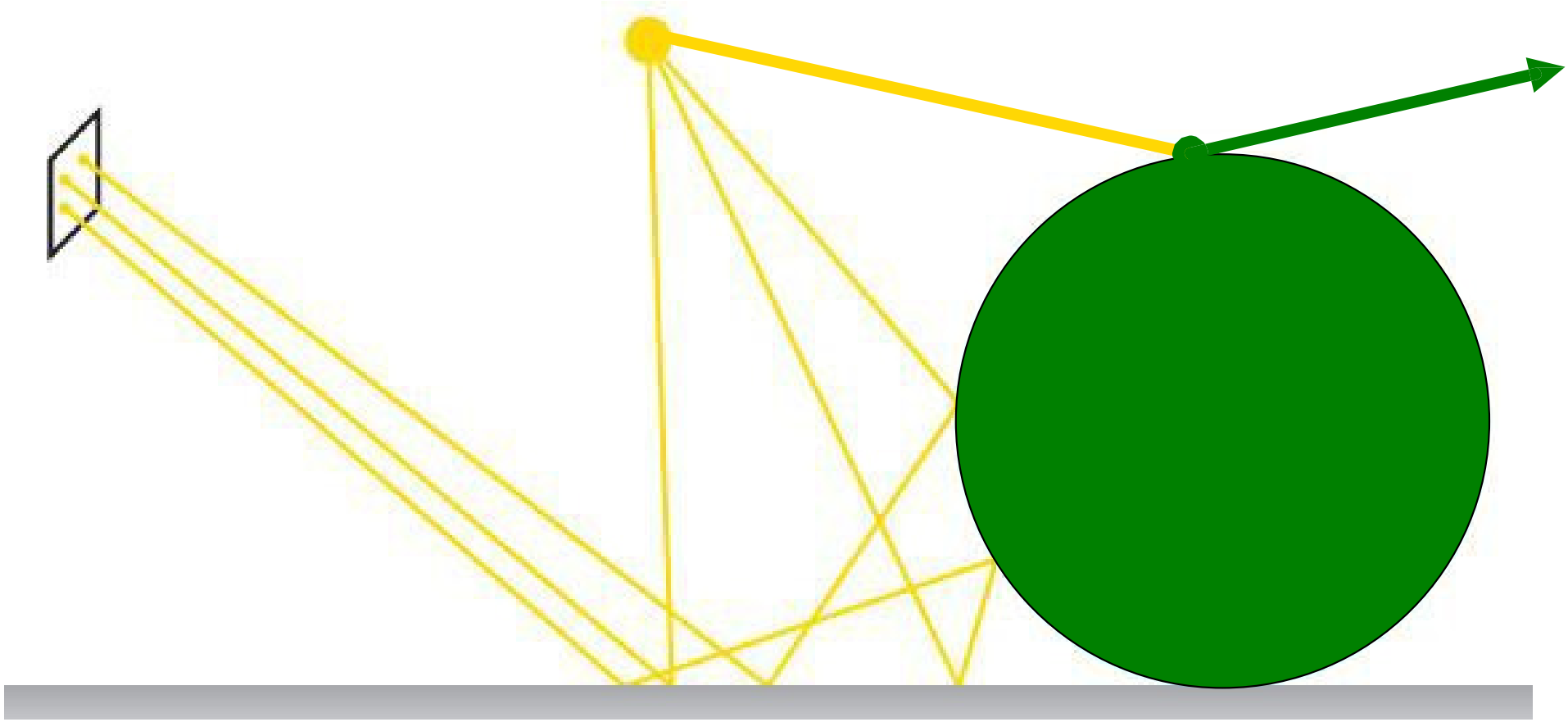
Paths of light

Some rays end up in our camera plane



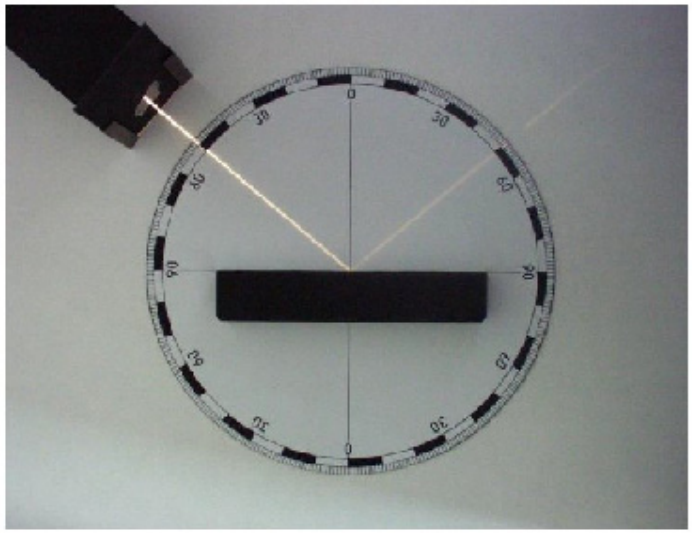
Is "forward" ray tracing good idea?

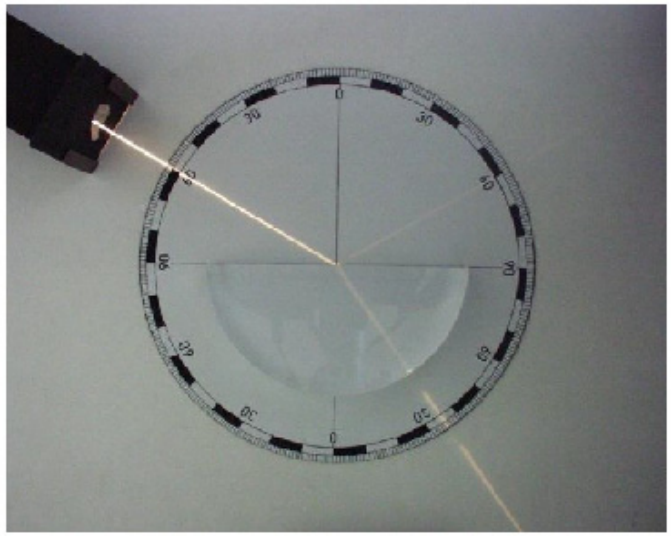
However, most rays never reach the camera! Huge unnecessary computation...



"Backward" ray tracing

Many light effects are reversible... Let's trace rays from the **camera to light** instead!

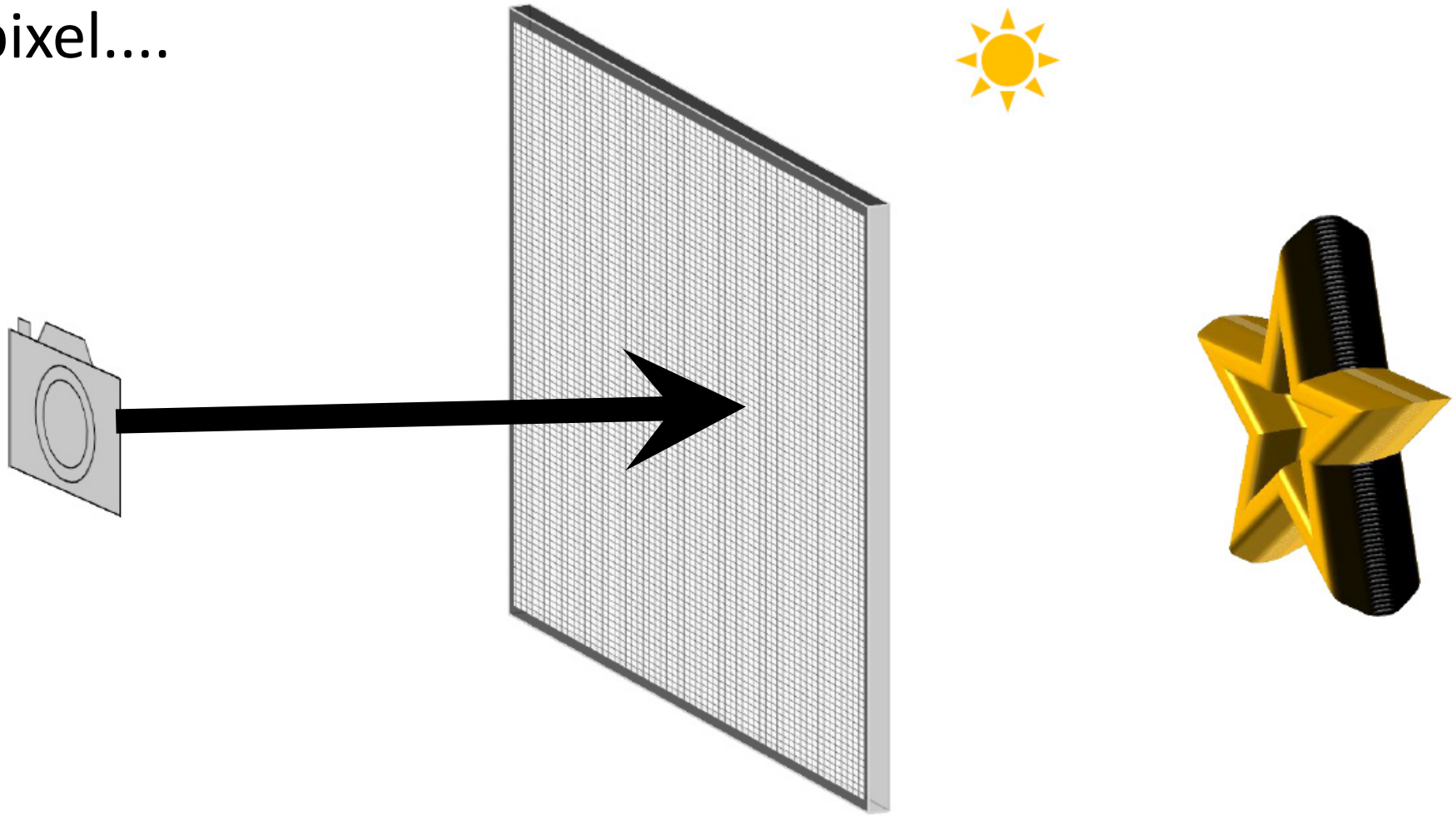




note: it is possible to combine forward and backward ray tracing (bidirectional path tracing)... particularly useful for indoor scenes

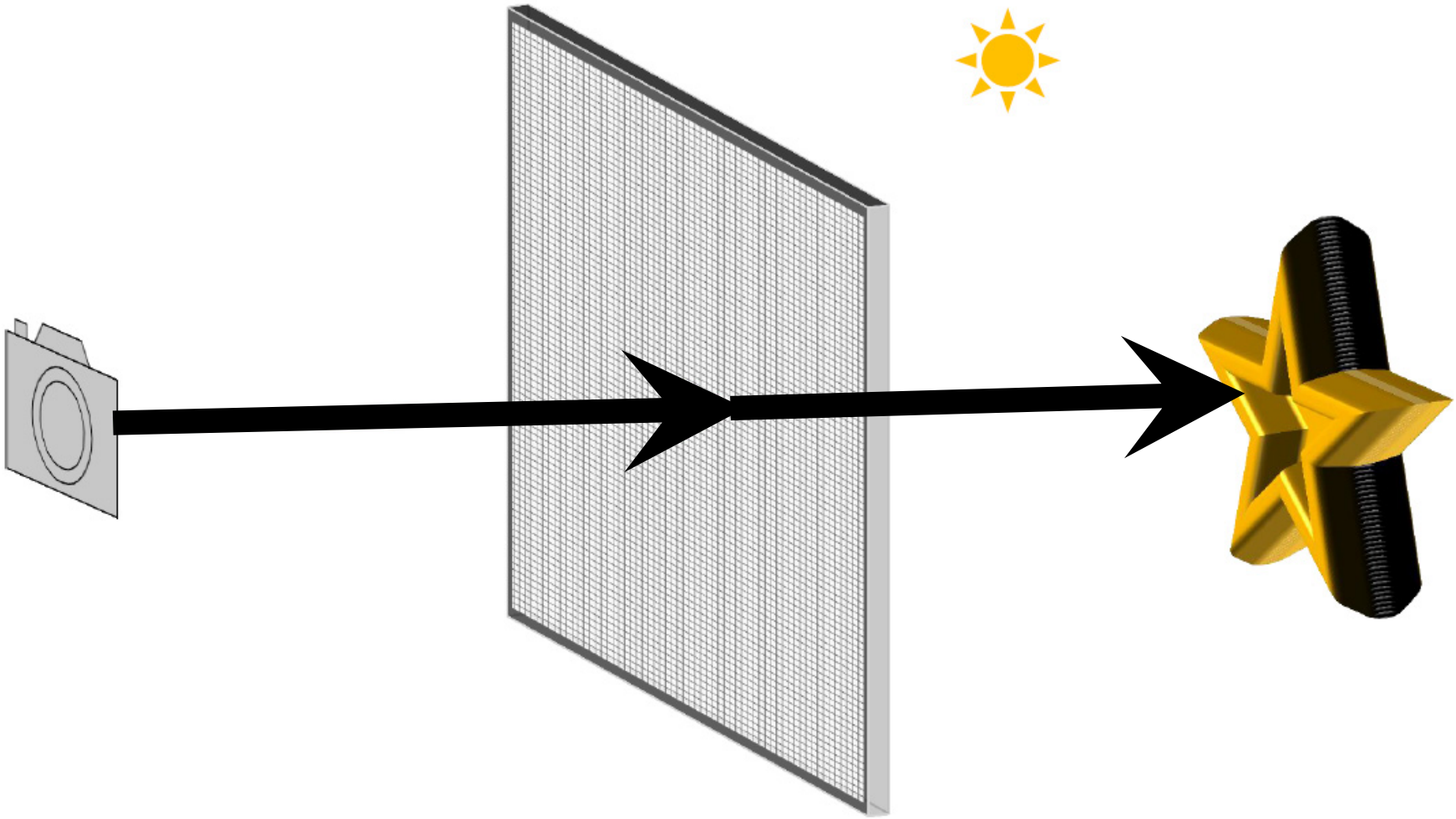
Ray tracing algorithm

Place your camera. Place an image array in front of it. Send a ray from camera towards the center of each pixel....



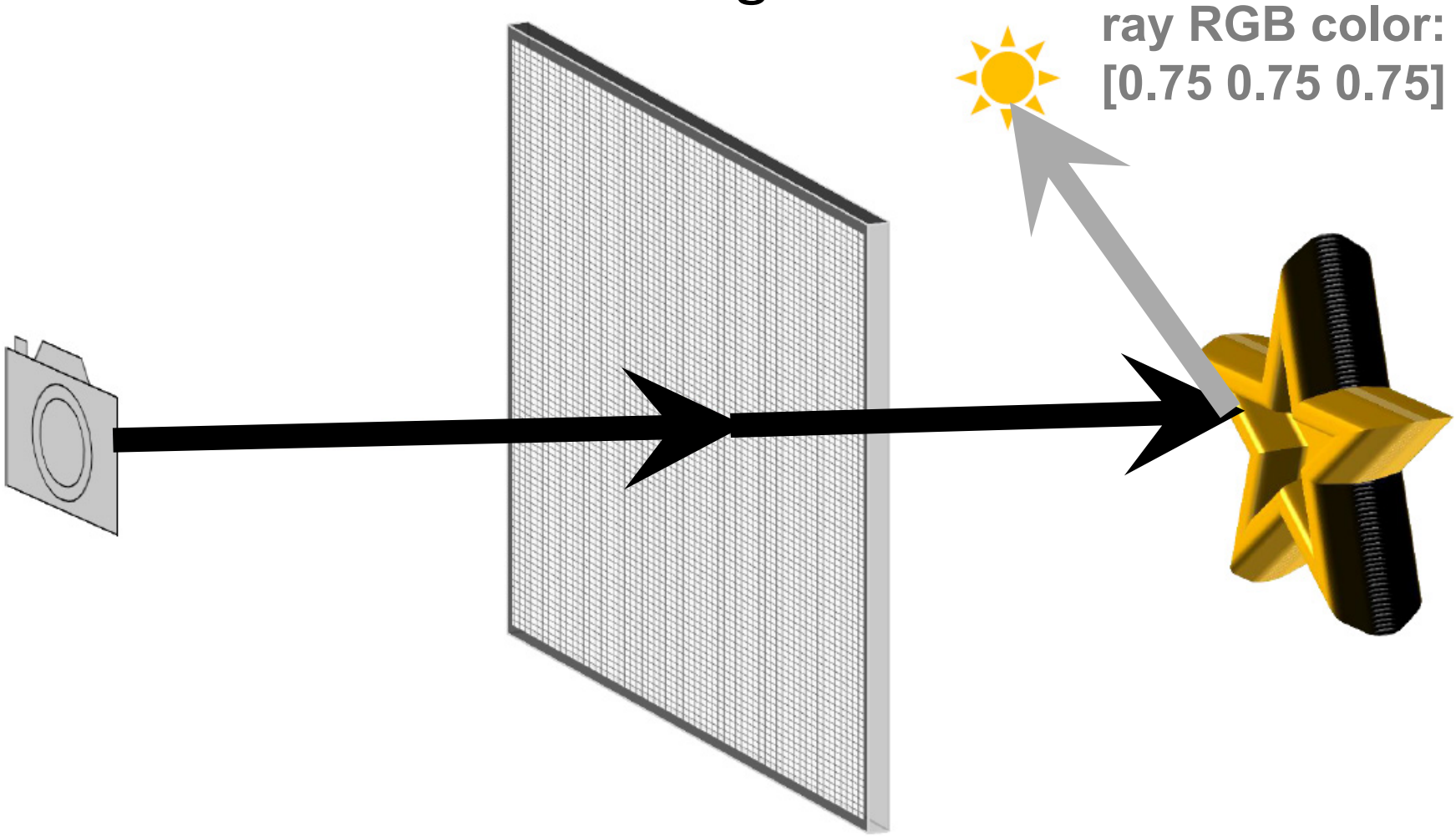
Ray tracing algorithm

Ray keeps going ... until it intersects with an object.



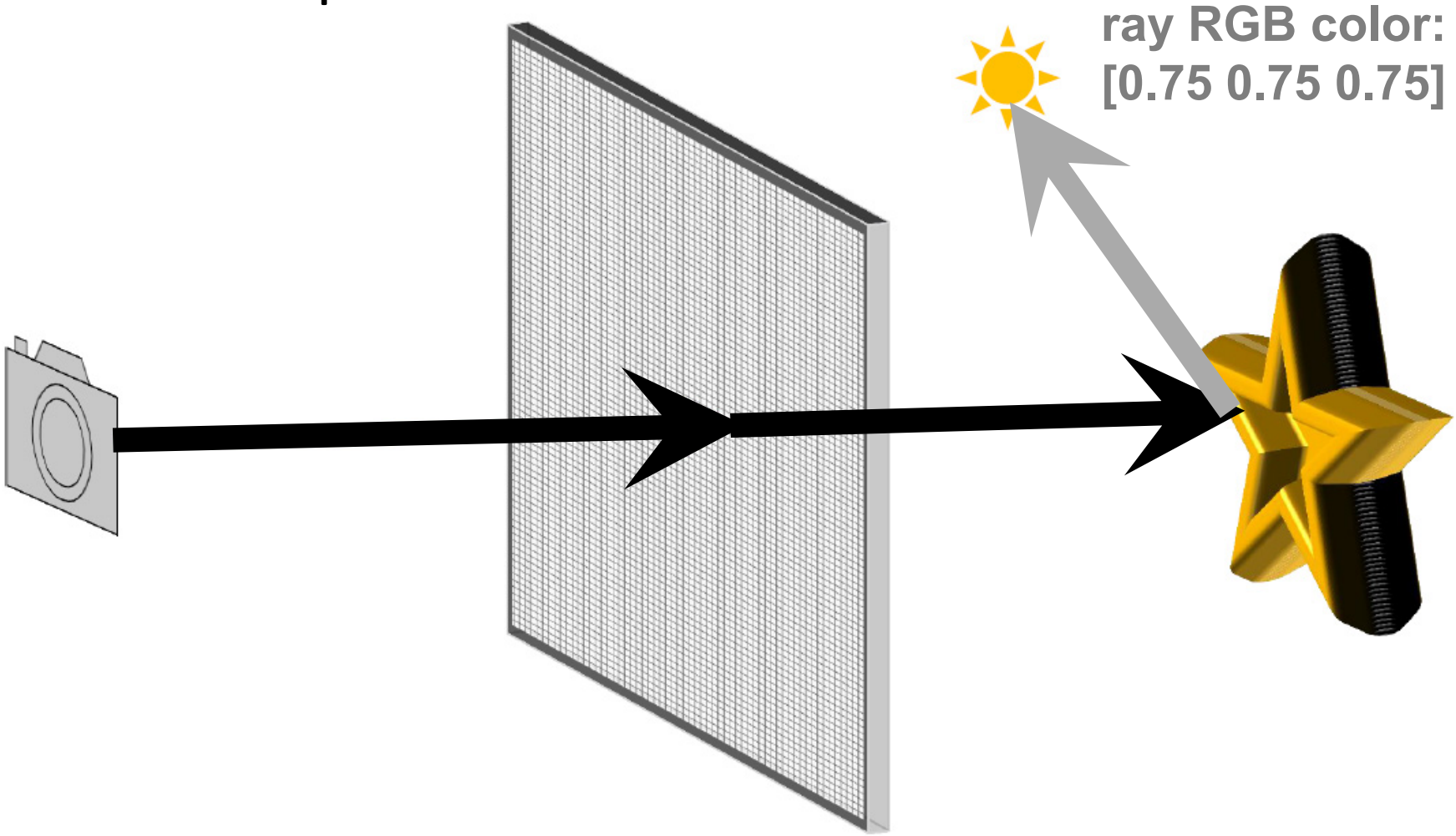
Ray tracing algorithm

Then compute a new **light ray** from the point of intersection towards each light.



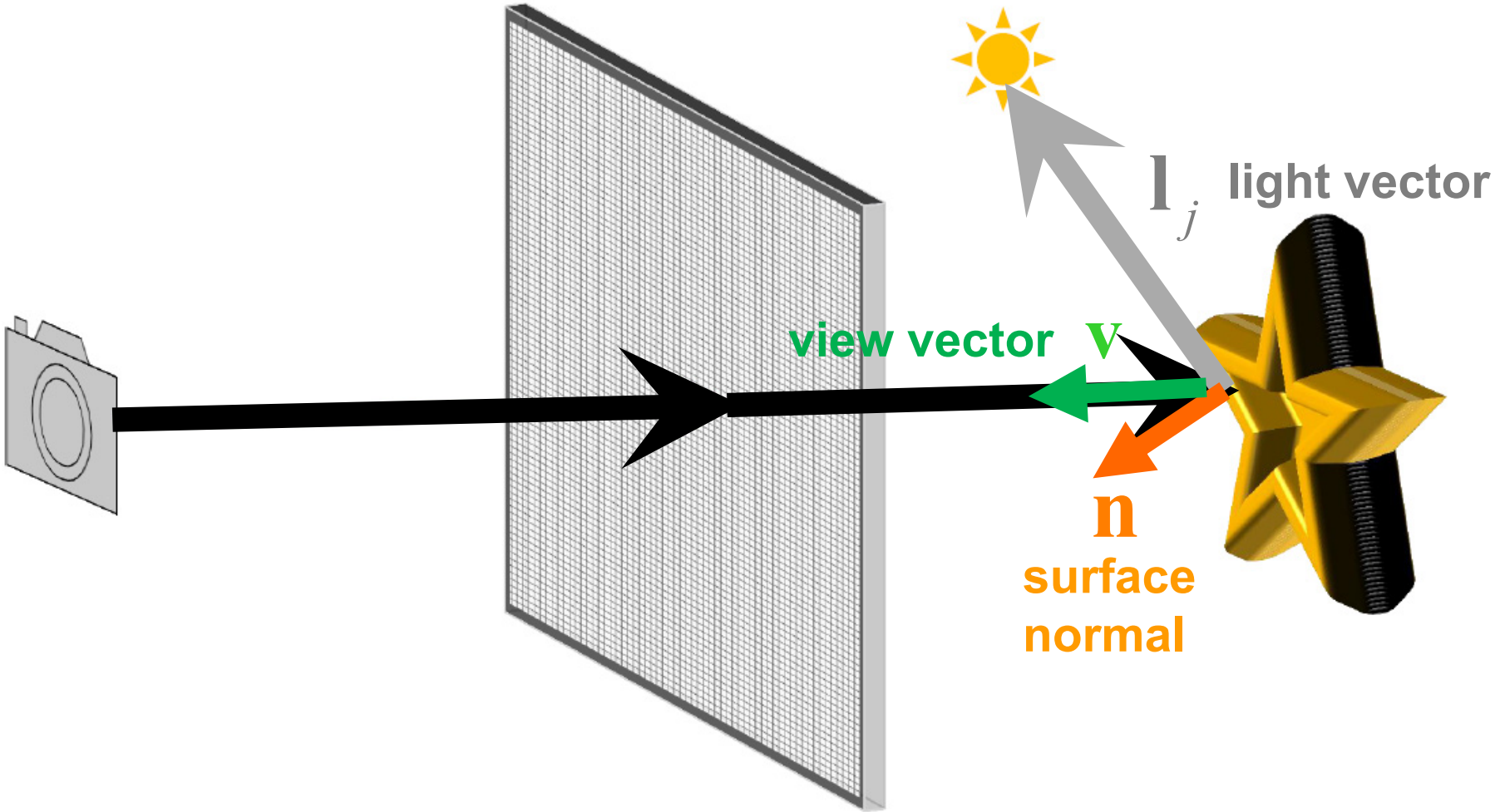
Ray tracing algorithm

Evaluate **shading model** based on the material of the intersection point...



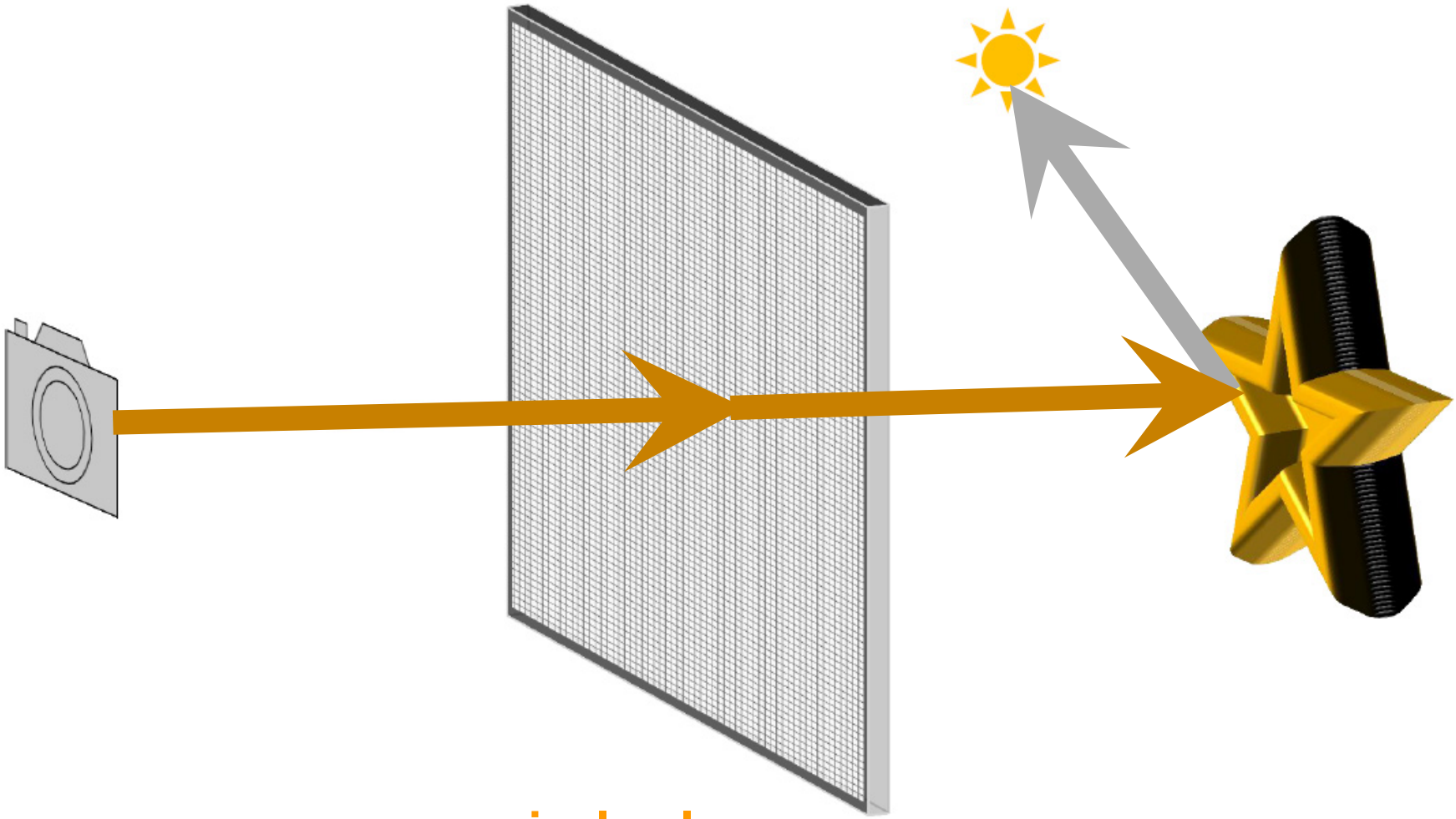
Ray tracing algorithm

Evaluate **shading model** based on the material of the intersection point...



Ray tracing algorithm

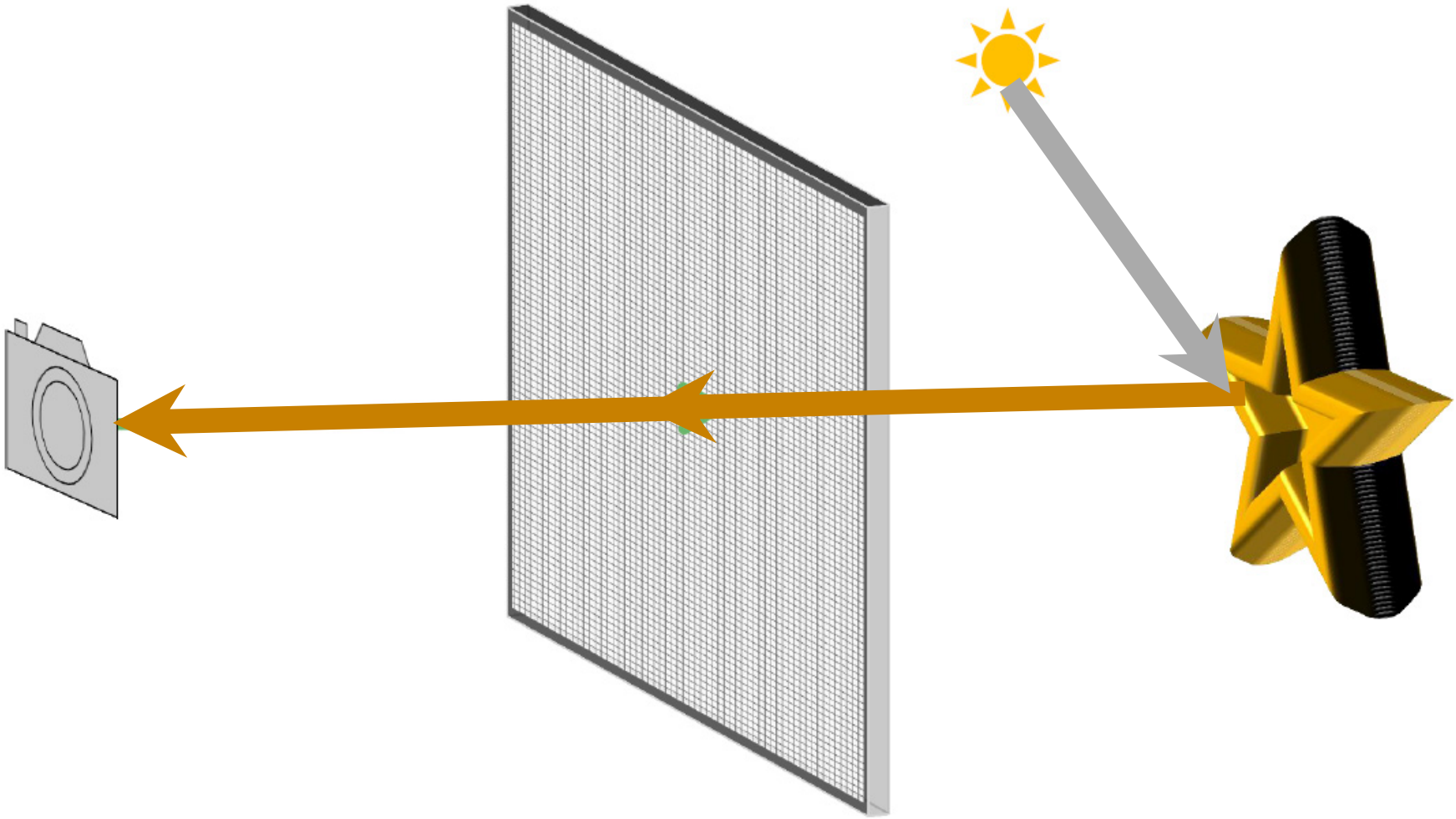
Update pixel color!



pixel color
e.g. [0.75 0.5 0.0]

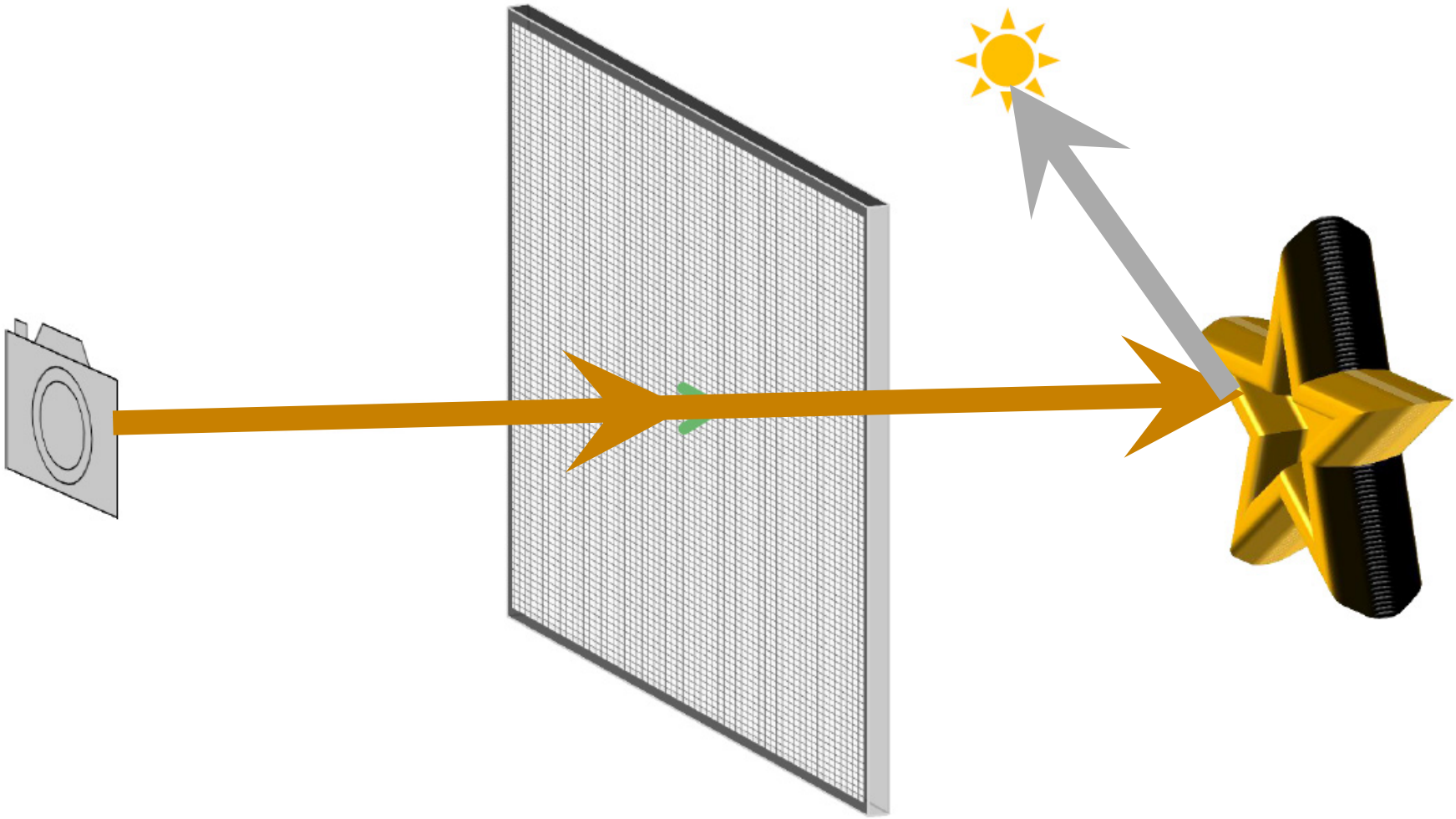
Ray tracing algorithm

This is the “**actual**” ray we traced in a “**backwards**” manner.



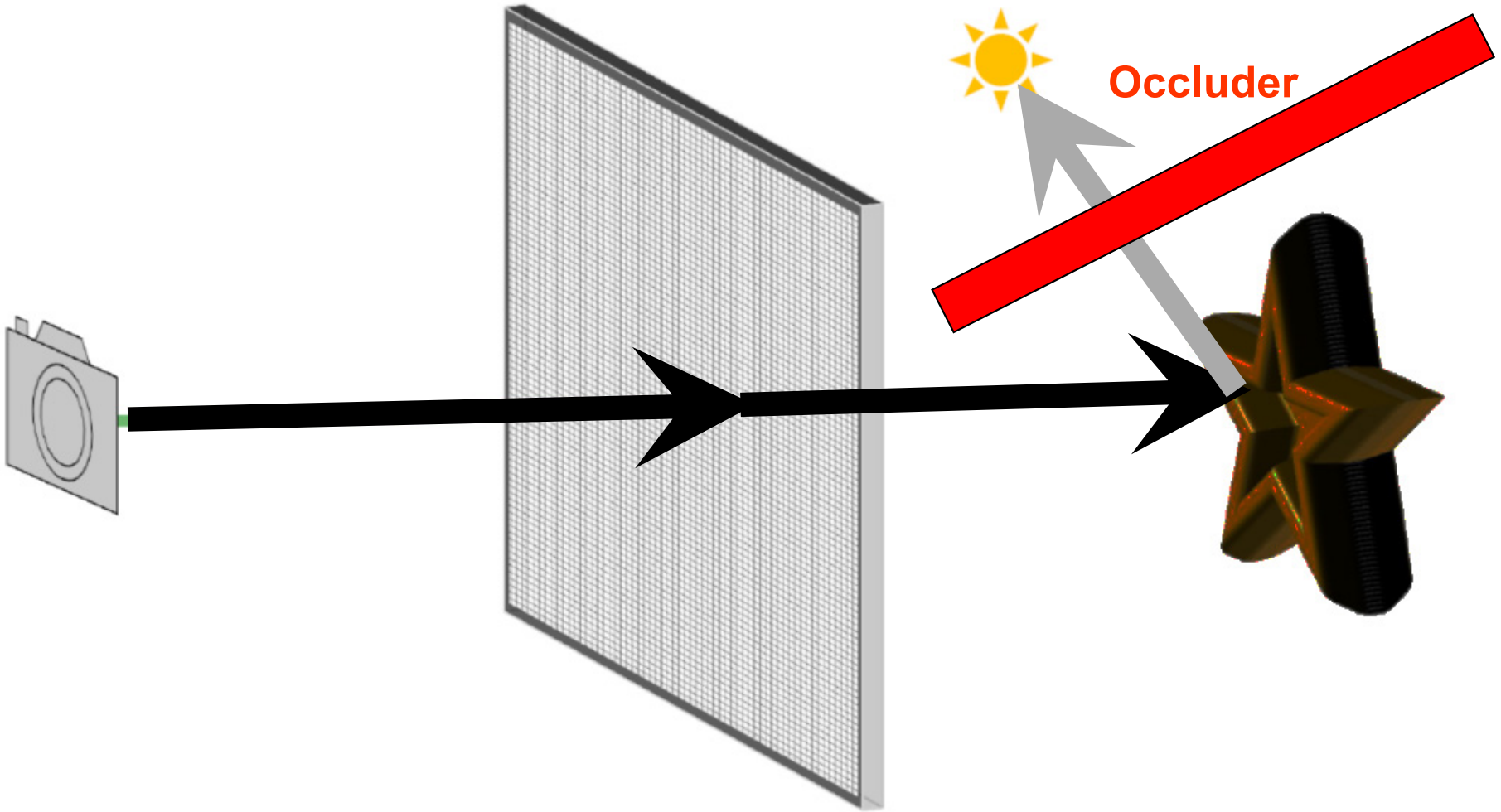
Ray tracing algorithm

As we discussed, ray tracing is **reversible**.



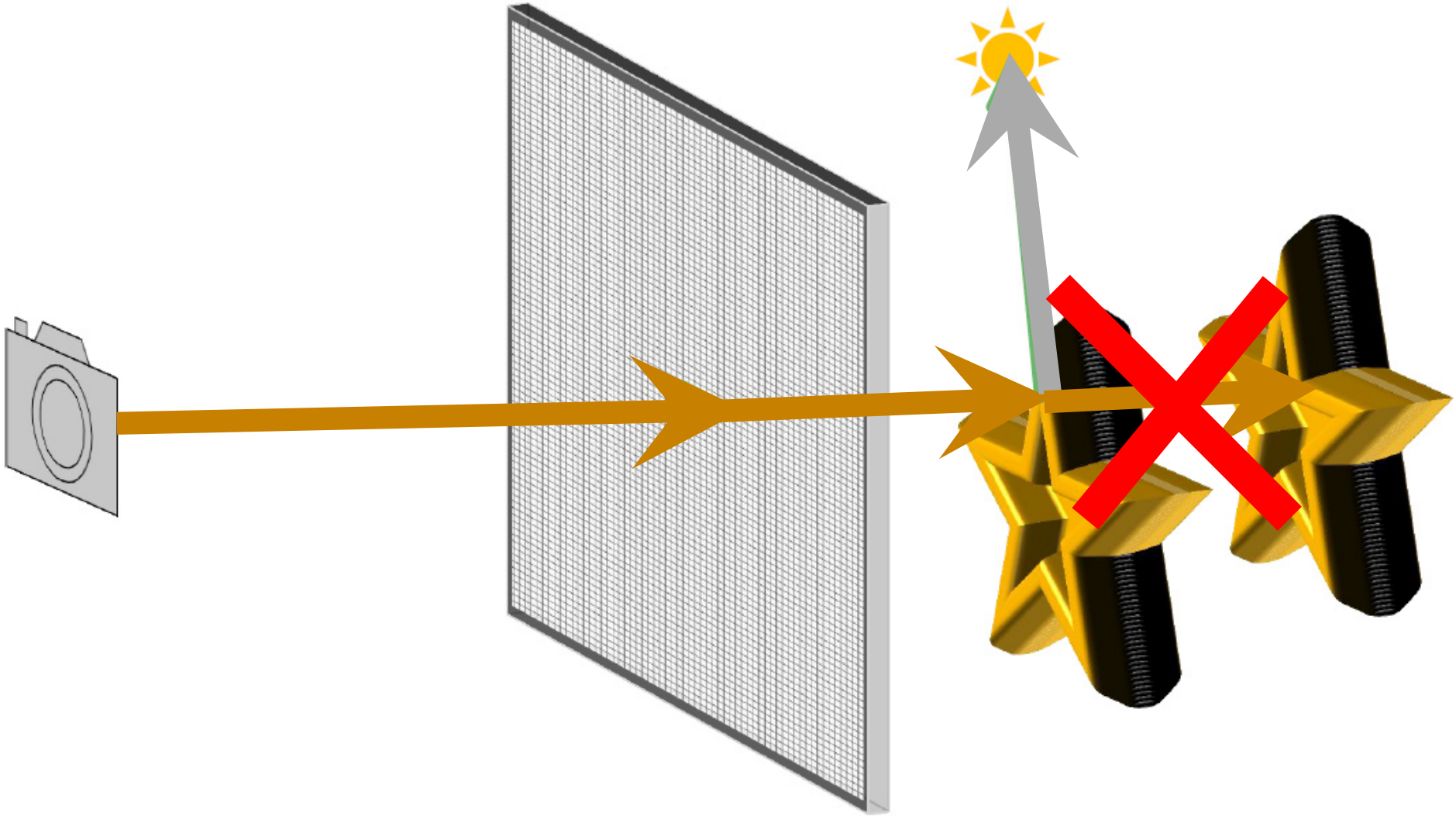
Shadow rays

If the light ray is blocked by another object, then the color of ray is set to black (or an ambient color)



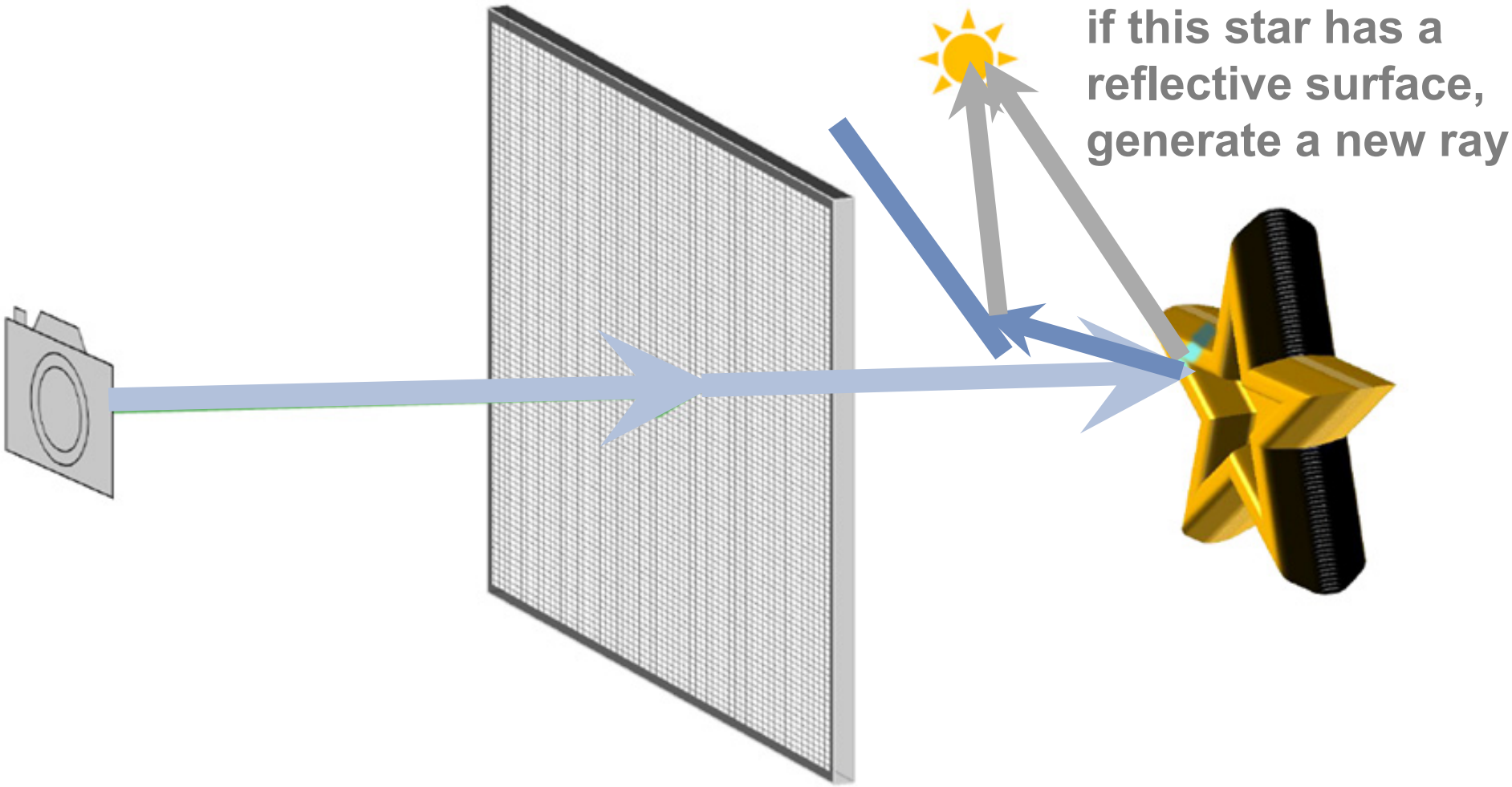
Multiple objects?

Consider only the closest intersection point during ray tracing!

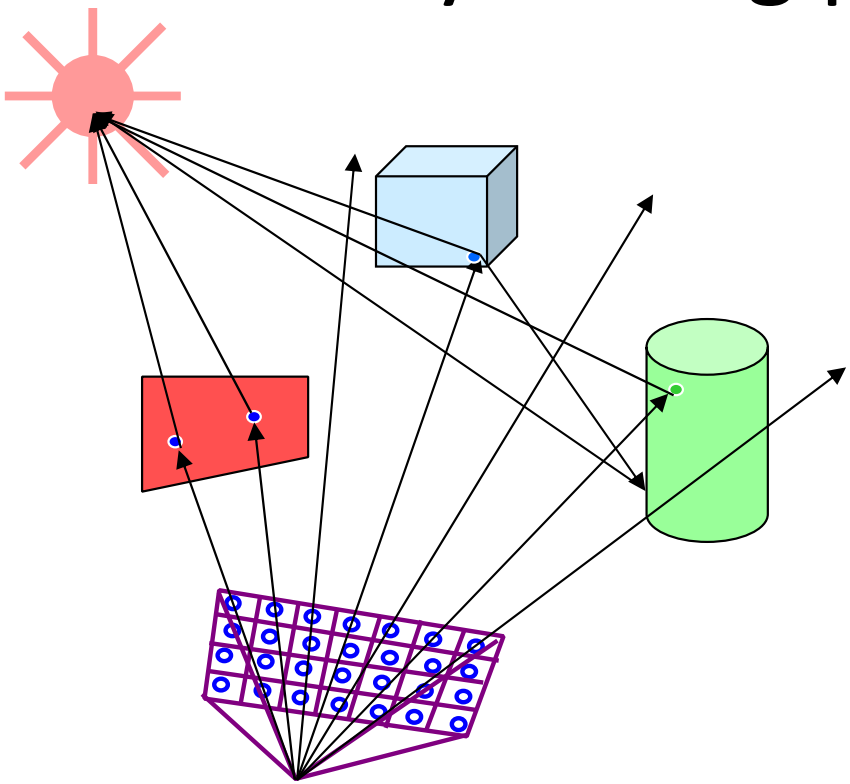


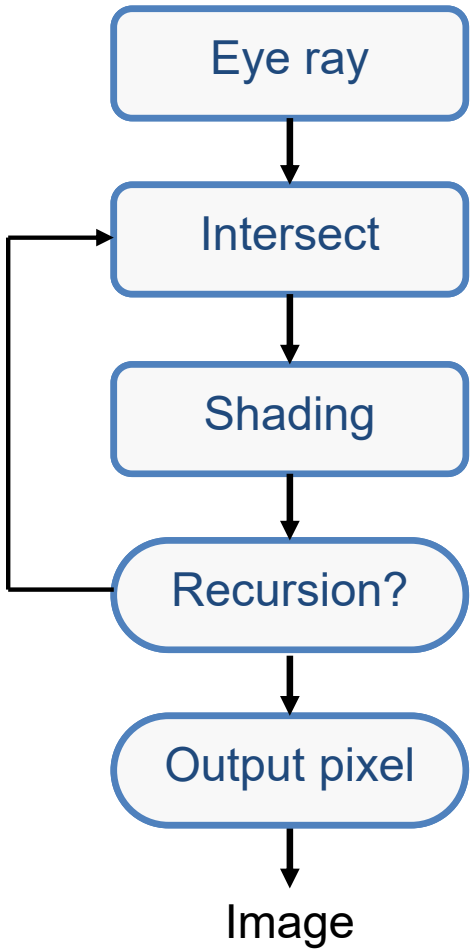
Reflections?

Basic idea: at each intersection point, **generate a new ray recursively**!



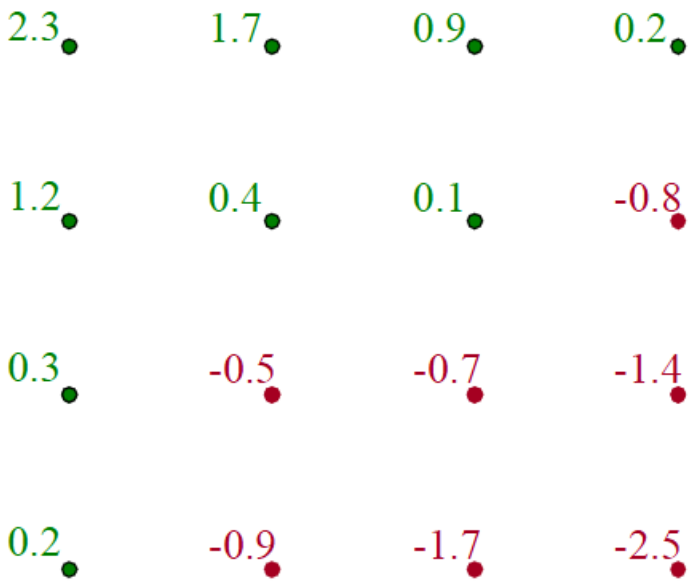
Ray tracing pipeline





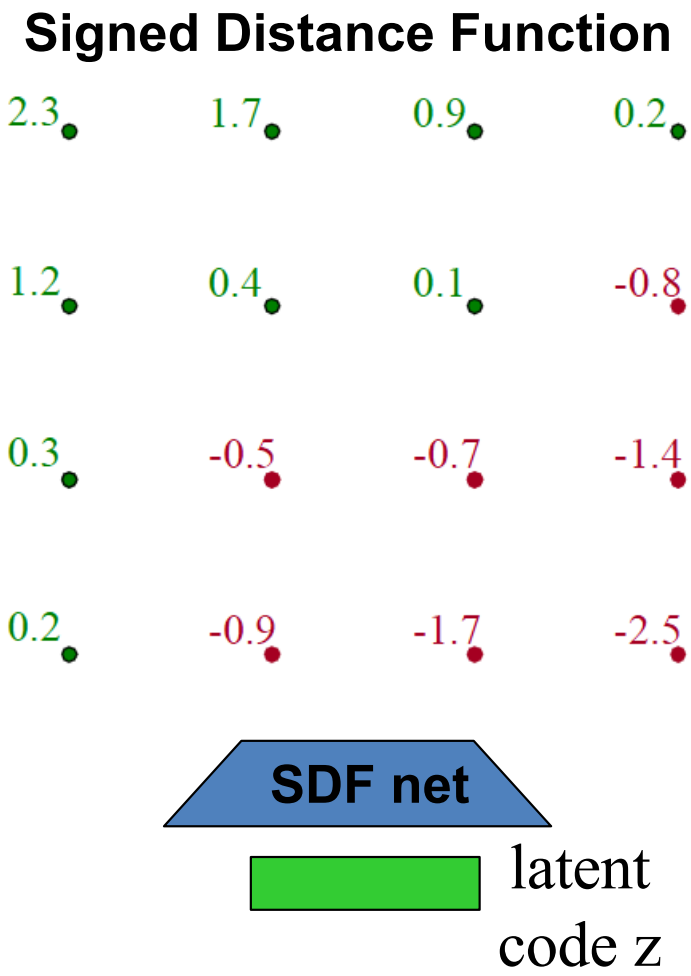
Differentiable Ray Marching for Implicit Surfaces

Signed Distance Function



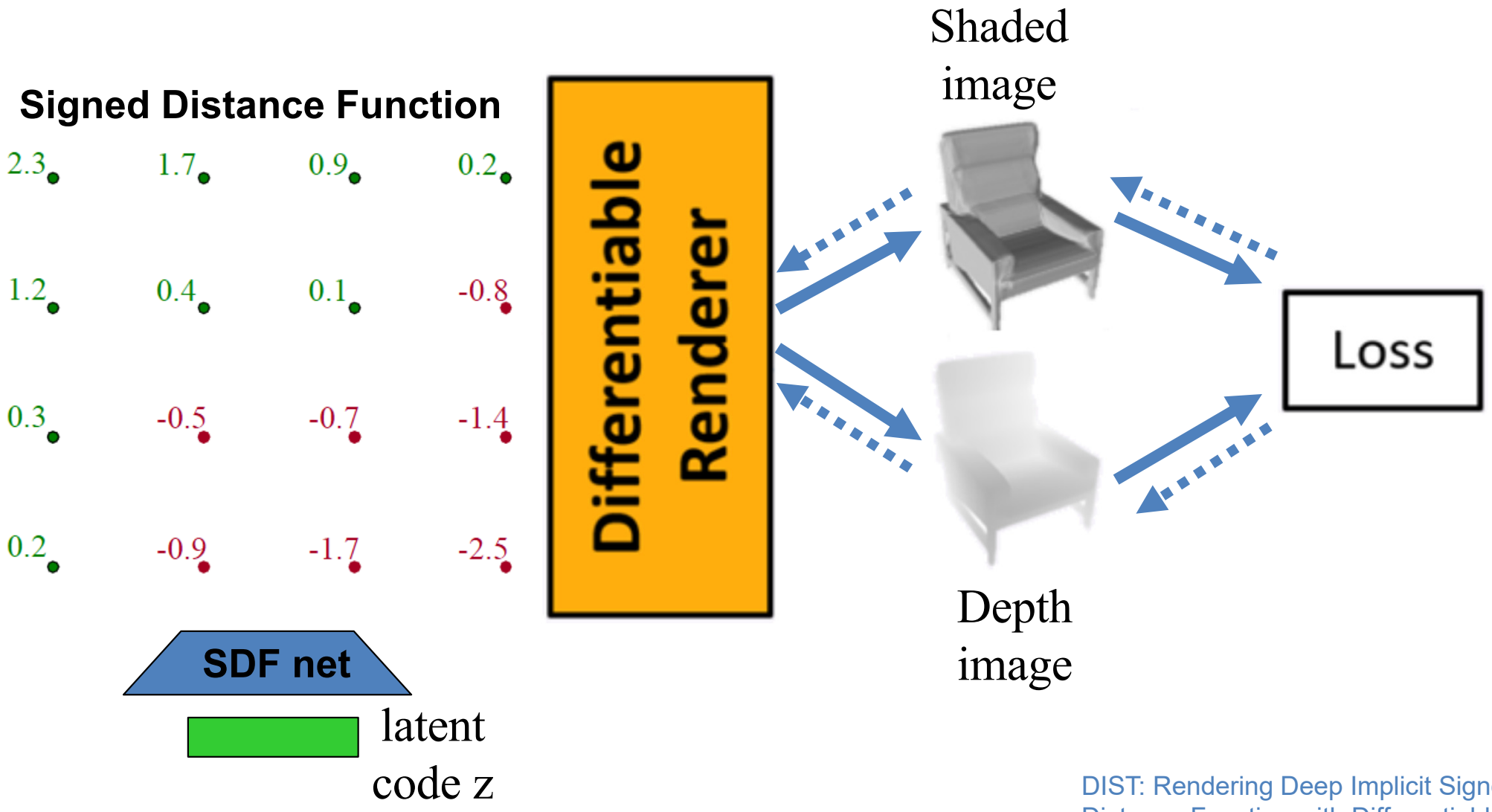
DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

Differentiable Ray Marching for Implicit Surfaces



DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

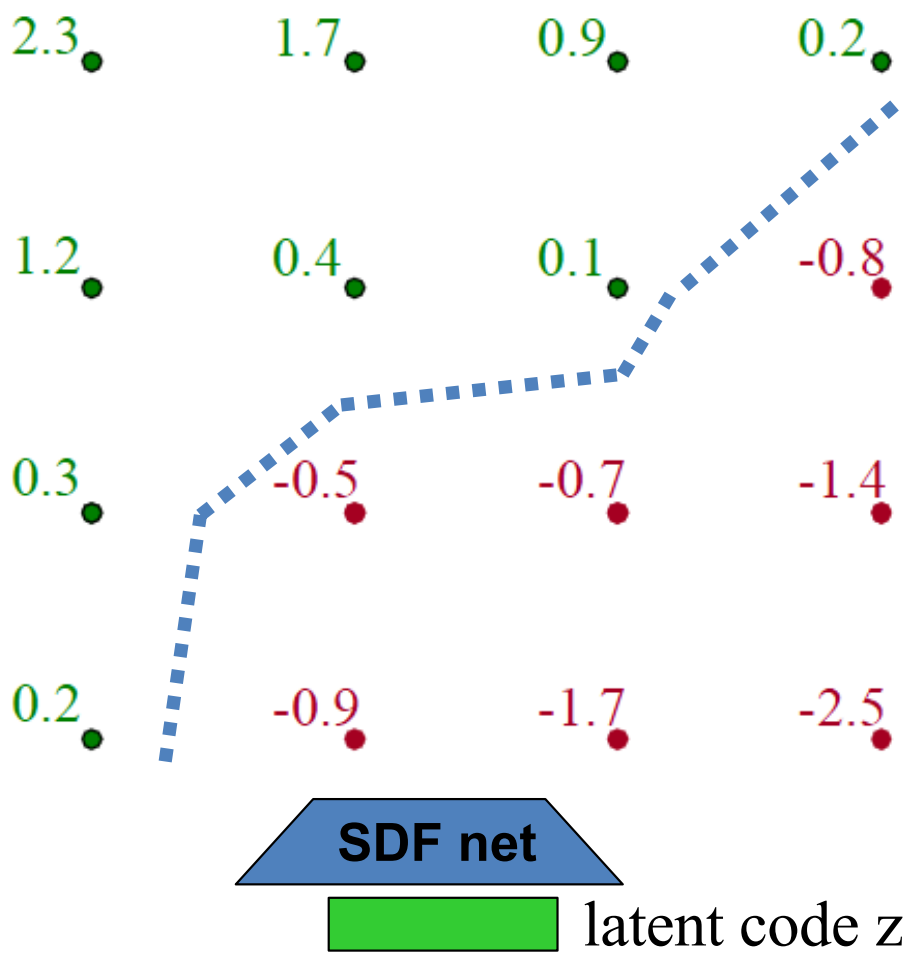
Differentiable Ray Marching for Implicit Surfaces



DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

How to perform ray tracing for implicits?

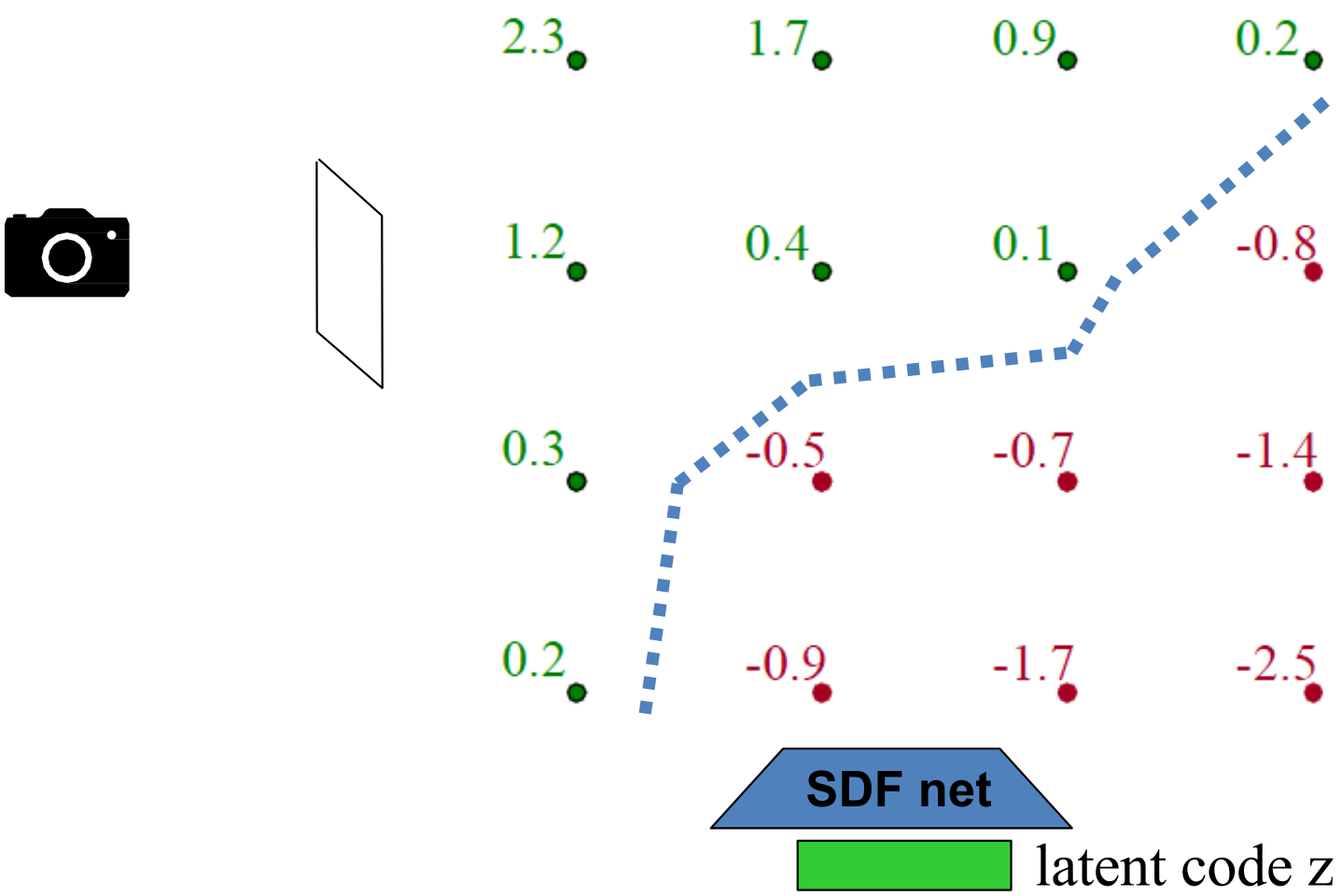
Given a continuous SDF function (here visualized as grid)...



DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

How to perform ray tracing for implicits?

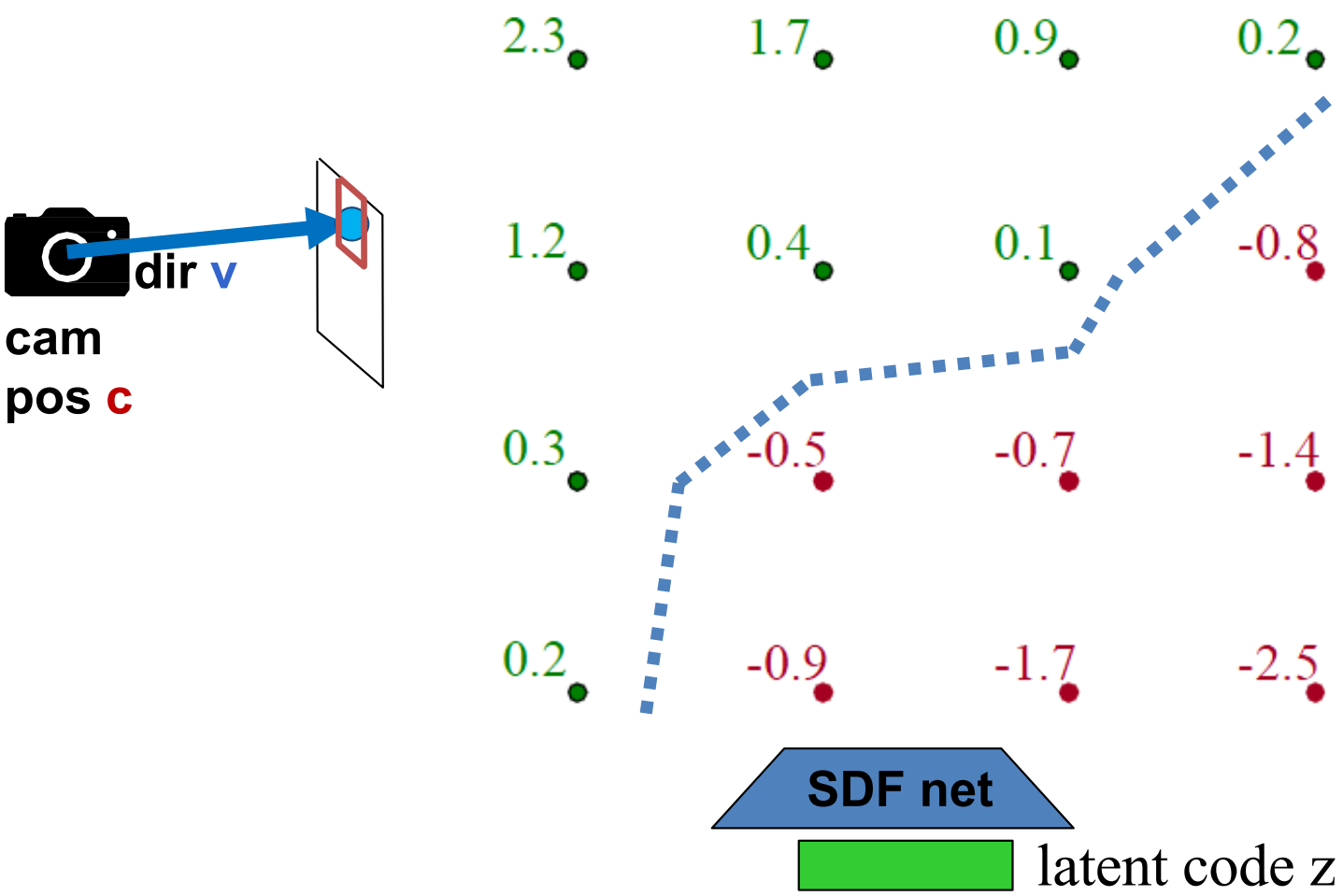
and a camera position and camera plane



DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

How to perform ray tracing for implicits?

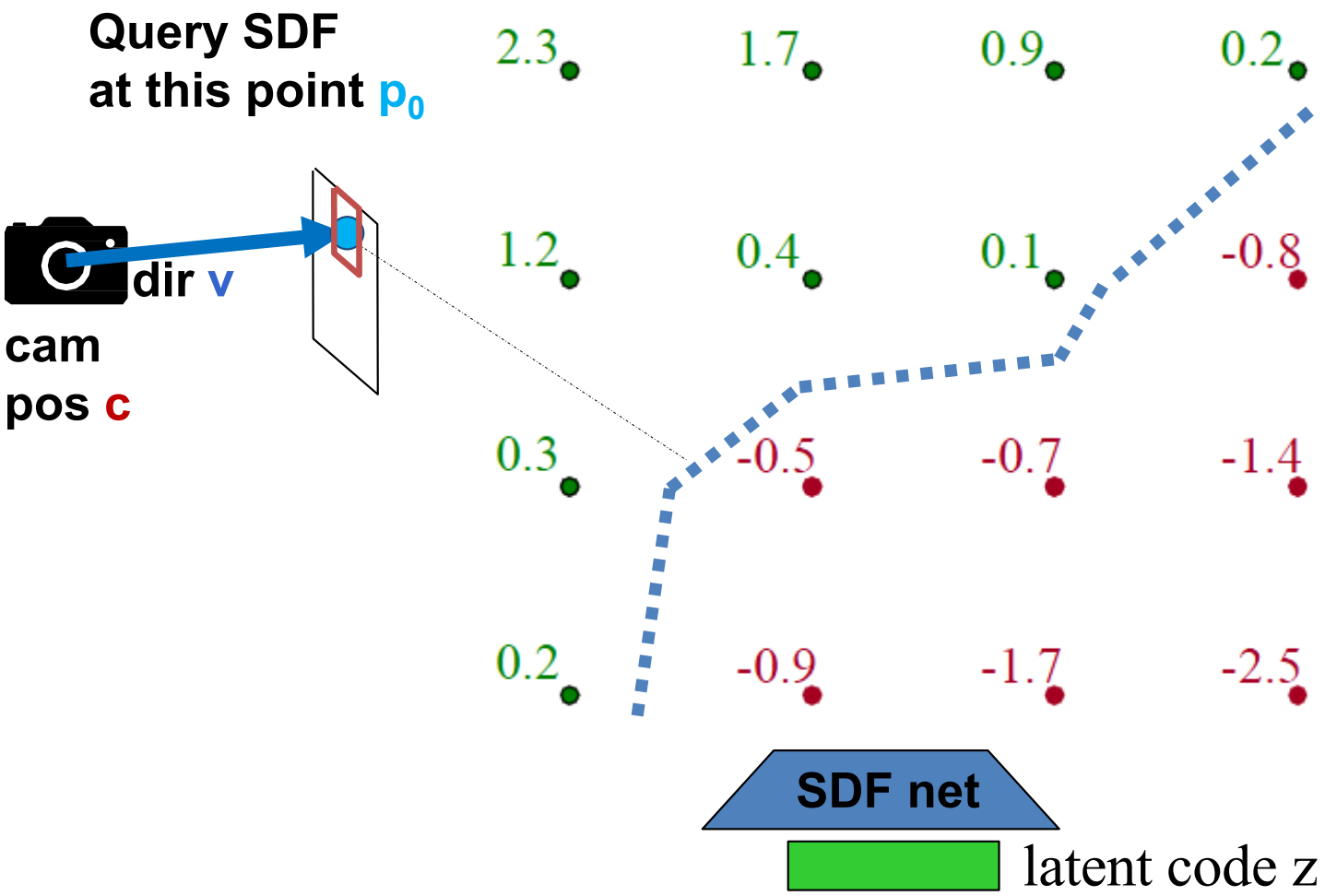
Shoot a ray for each pixel...



DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

How to perform ray tracing for implicits?

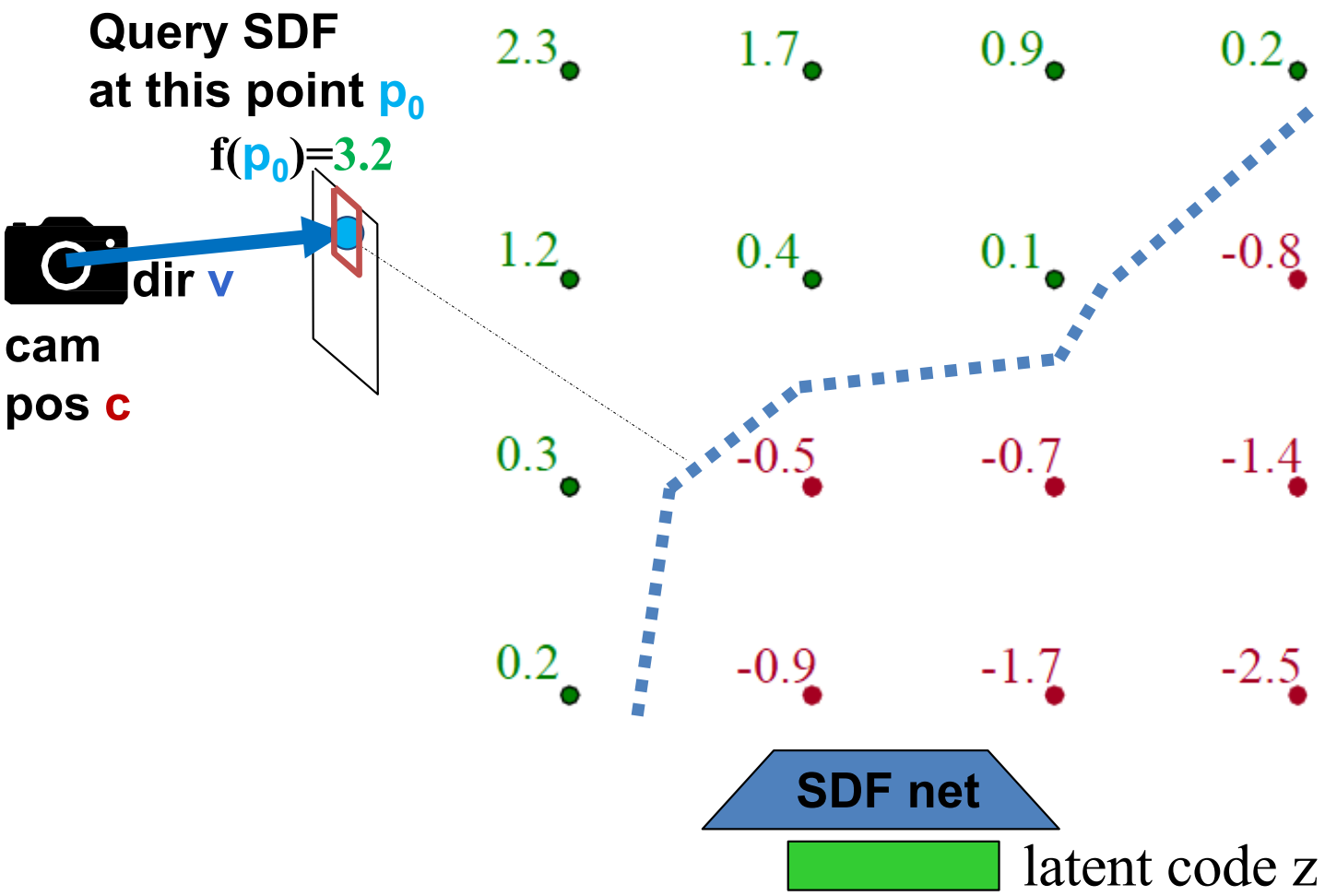
Shoot a ray for each pixel...



DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

How to perform ray tracing for implicits?

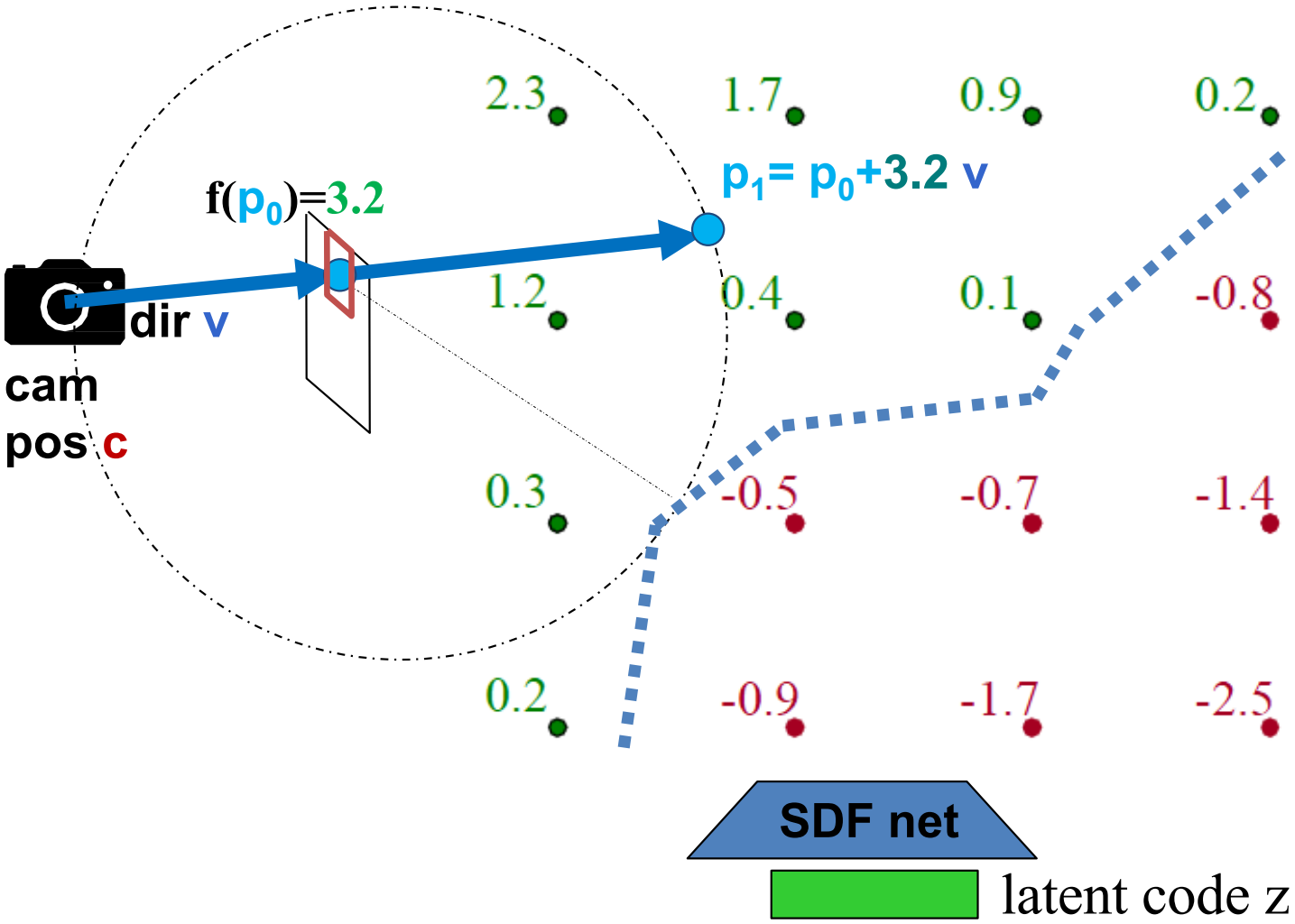
Shoot a ray for each pixel...



DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

How to perform ray tracing for implicits?

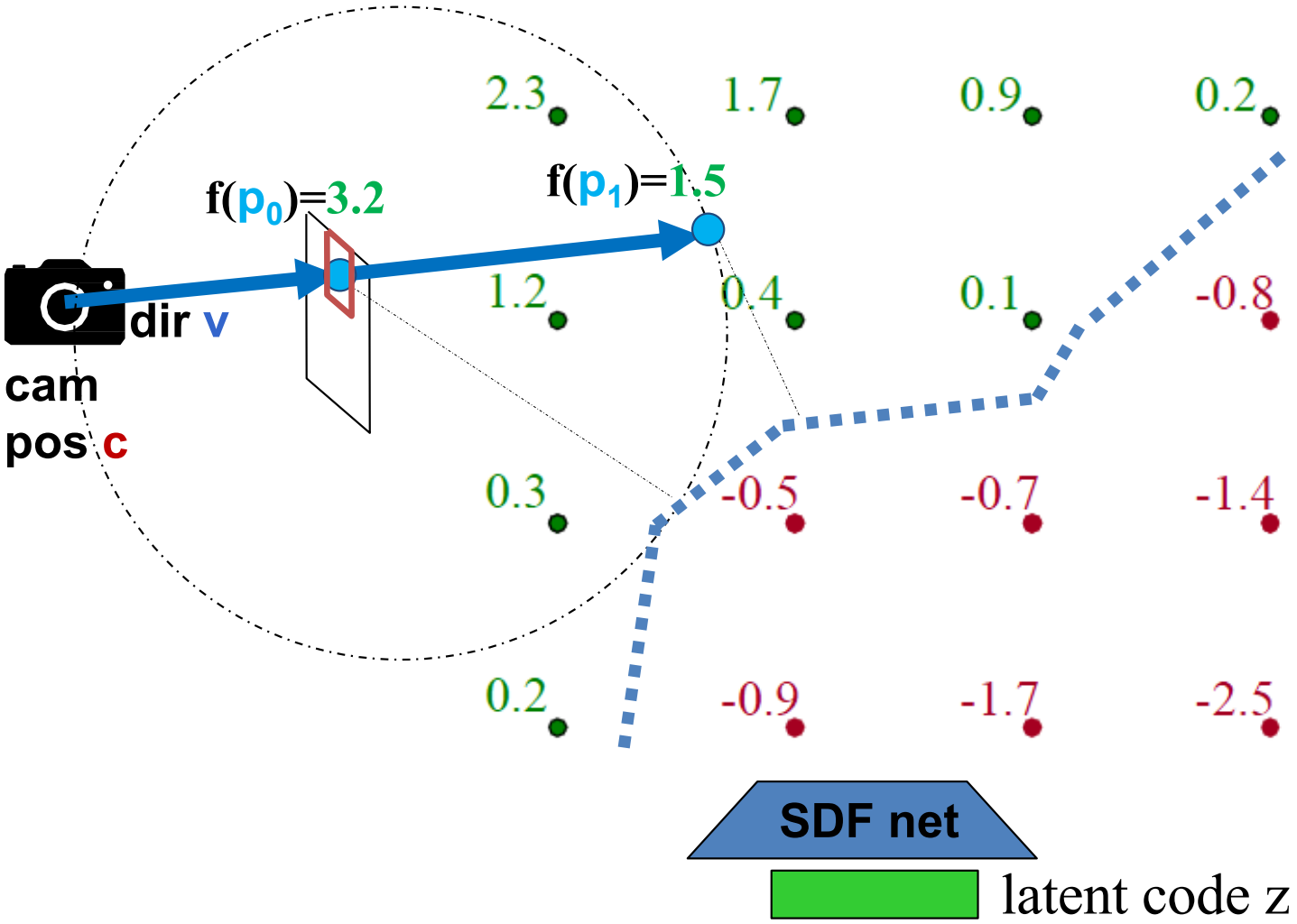
Follow the path of the ray taking steps according to the SDF ("sphere" ray tracing)



DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

How to perform ray tracing for implicits?

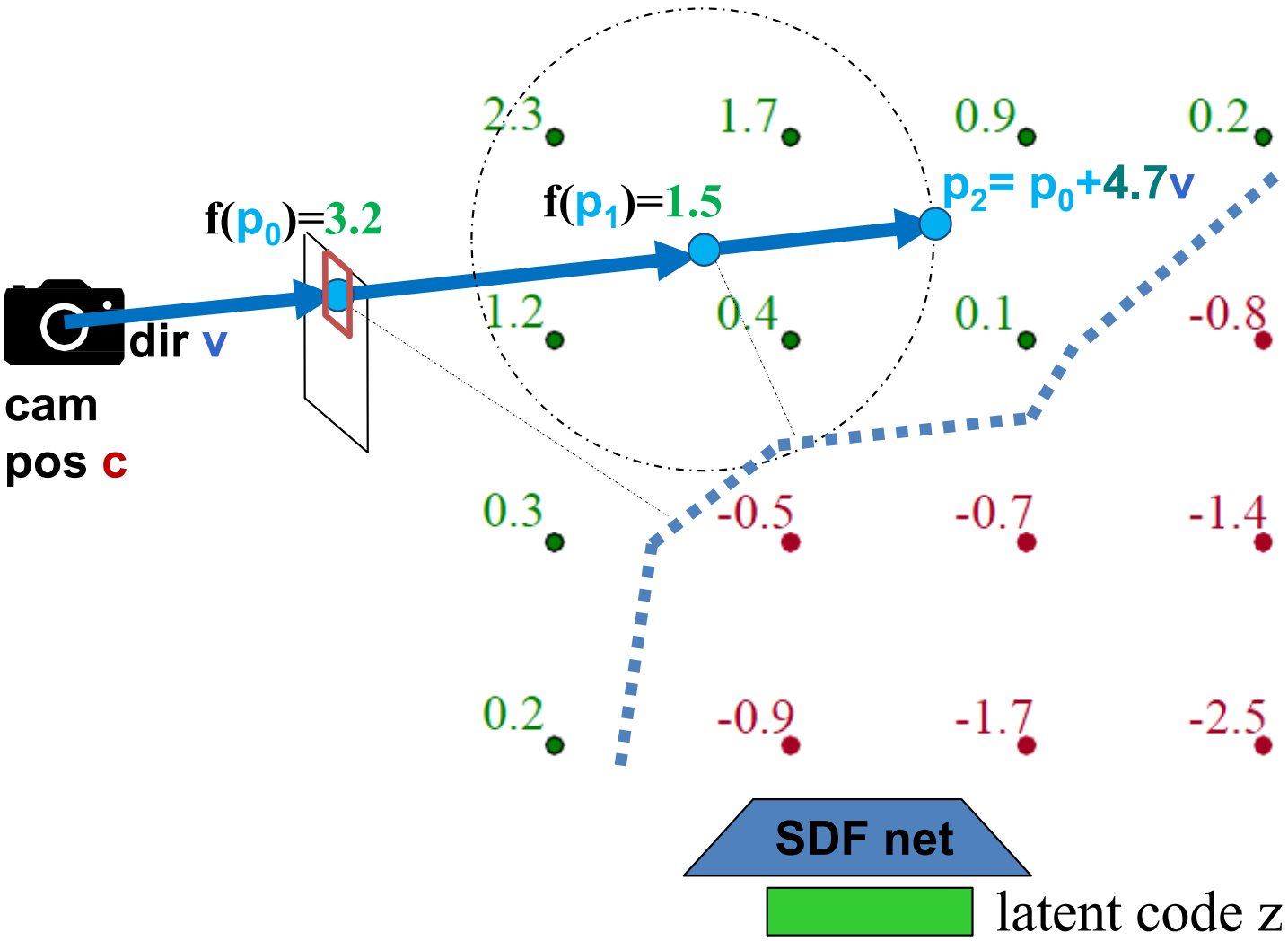
Follow the path of the ray taking steps according to the SDF ("sphere" ray tracing)



DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

How to perform ray tracing for implicits?

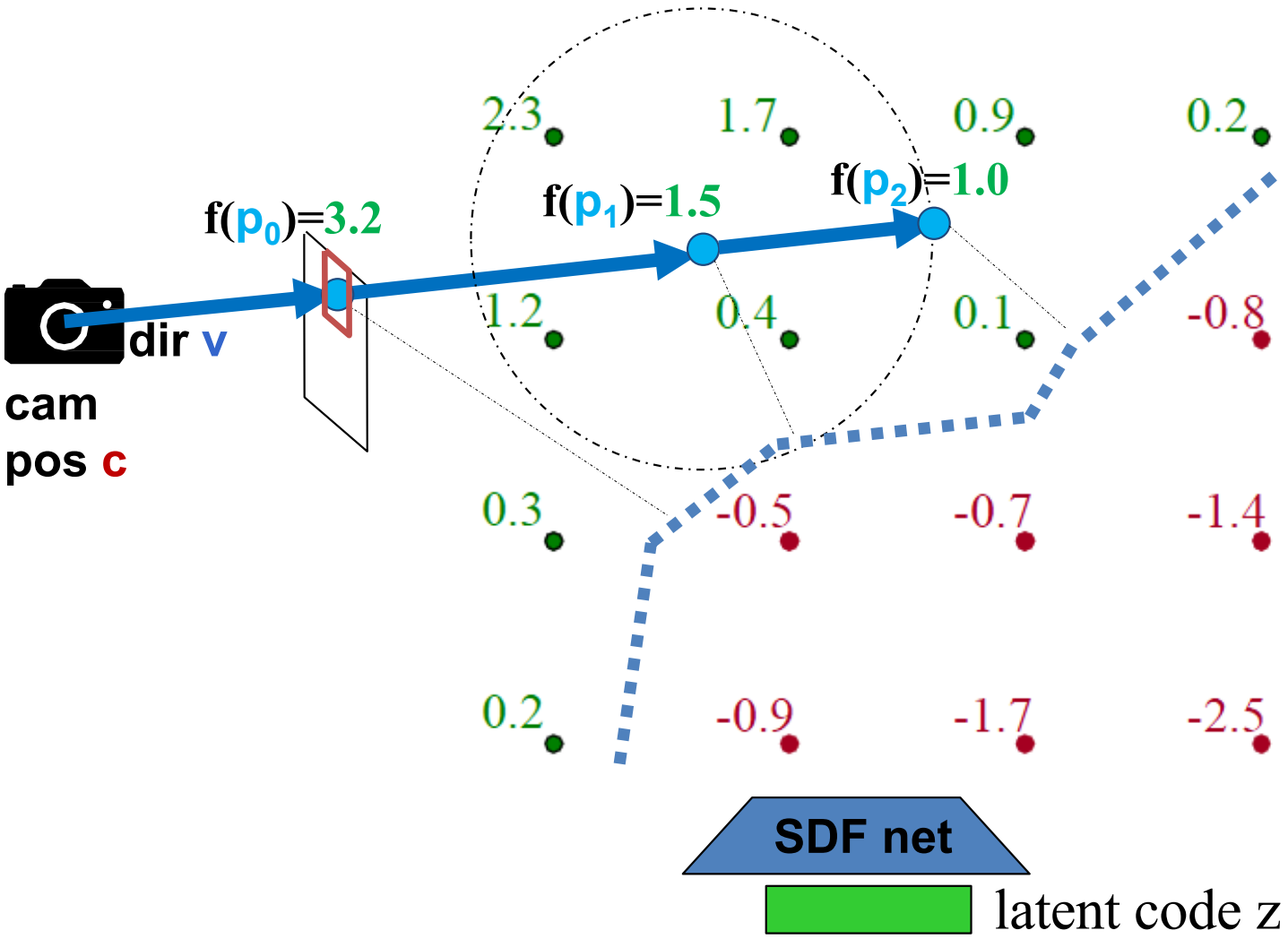
Follow the path of the ray taking steps according to the SDF ("sphere" ray tracing)



DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

How to perform ray tracing for implicits?

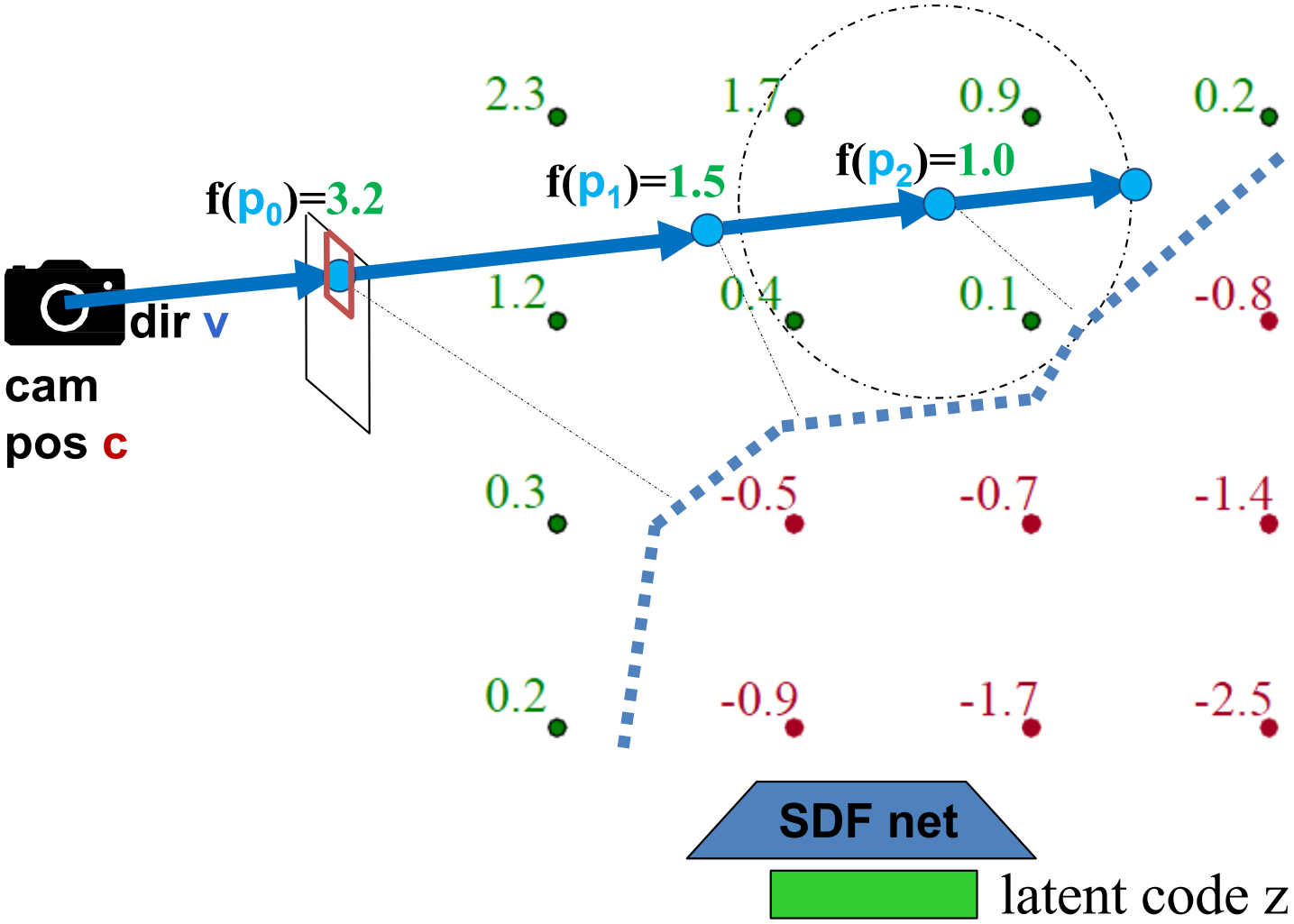
Follow the path of the ray taking steps according to the SDF ("sphere" ray tracing)



DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

How to perform ray tracing for implicits?

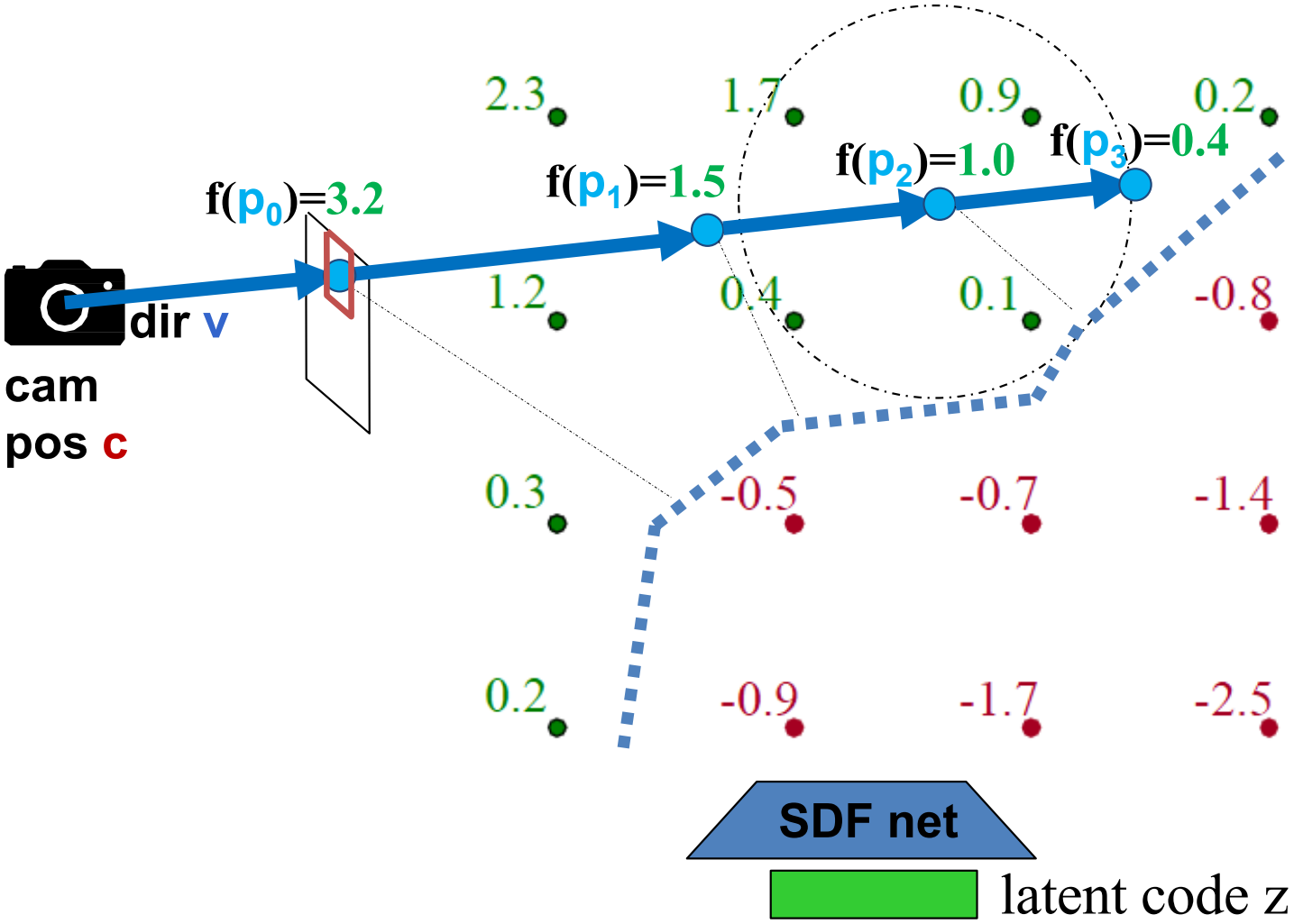
Follow the path of the ray taking steps according to the SDF ("sphere" ray tracing)



DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

How to perform ray tracing for implicits?

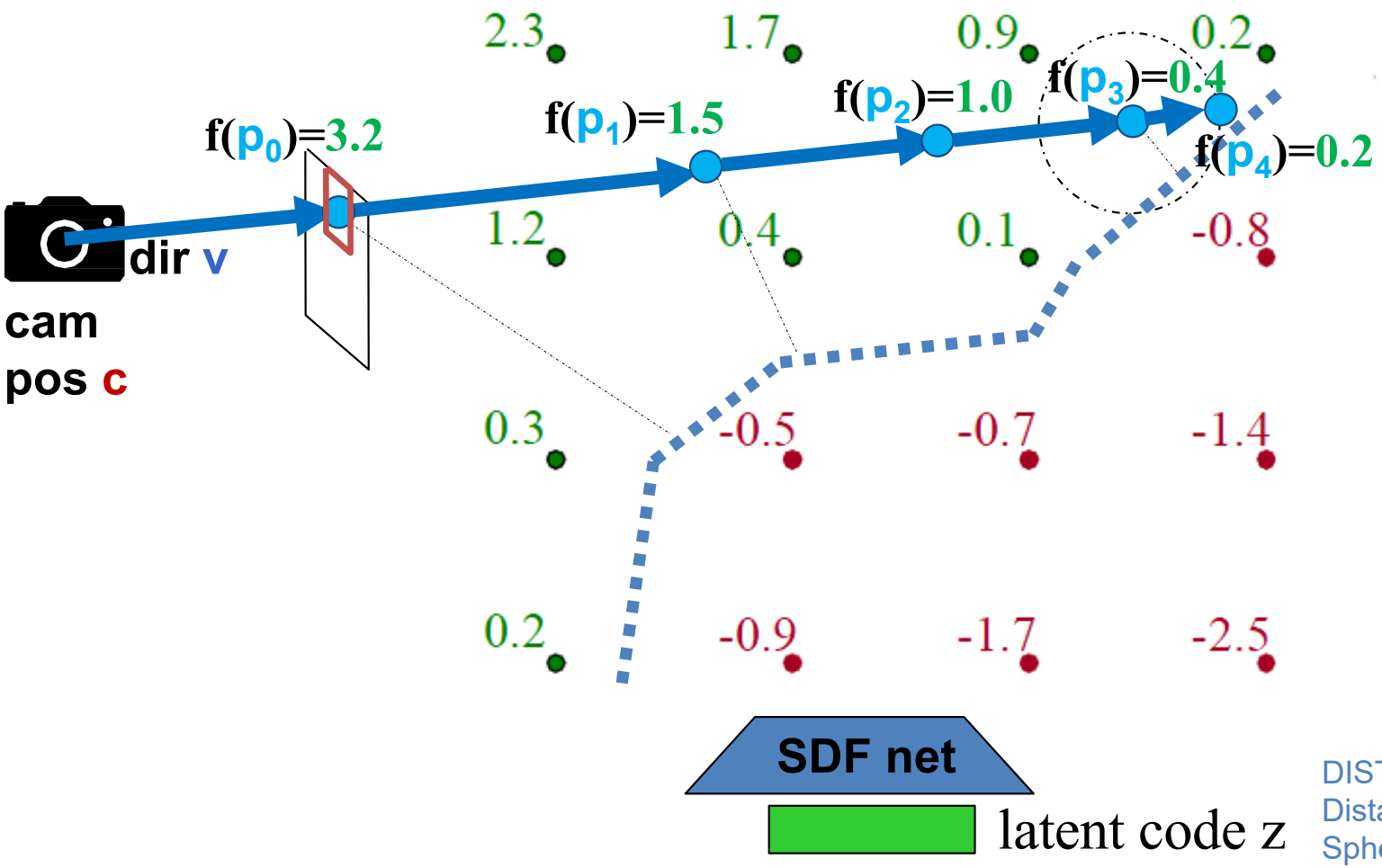
Follow the path of the ray taking steps according to the SDF ("sphere" ray tracing)



DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

How to perform ray tracing for implicits?

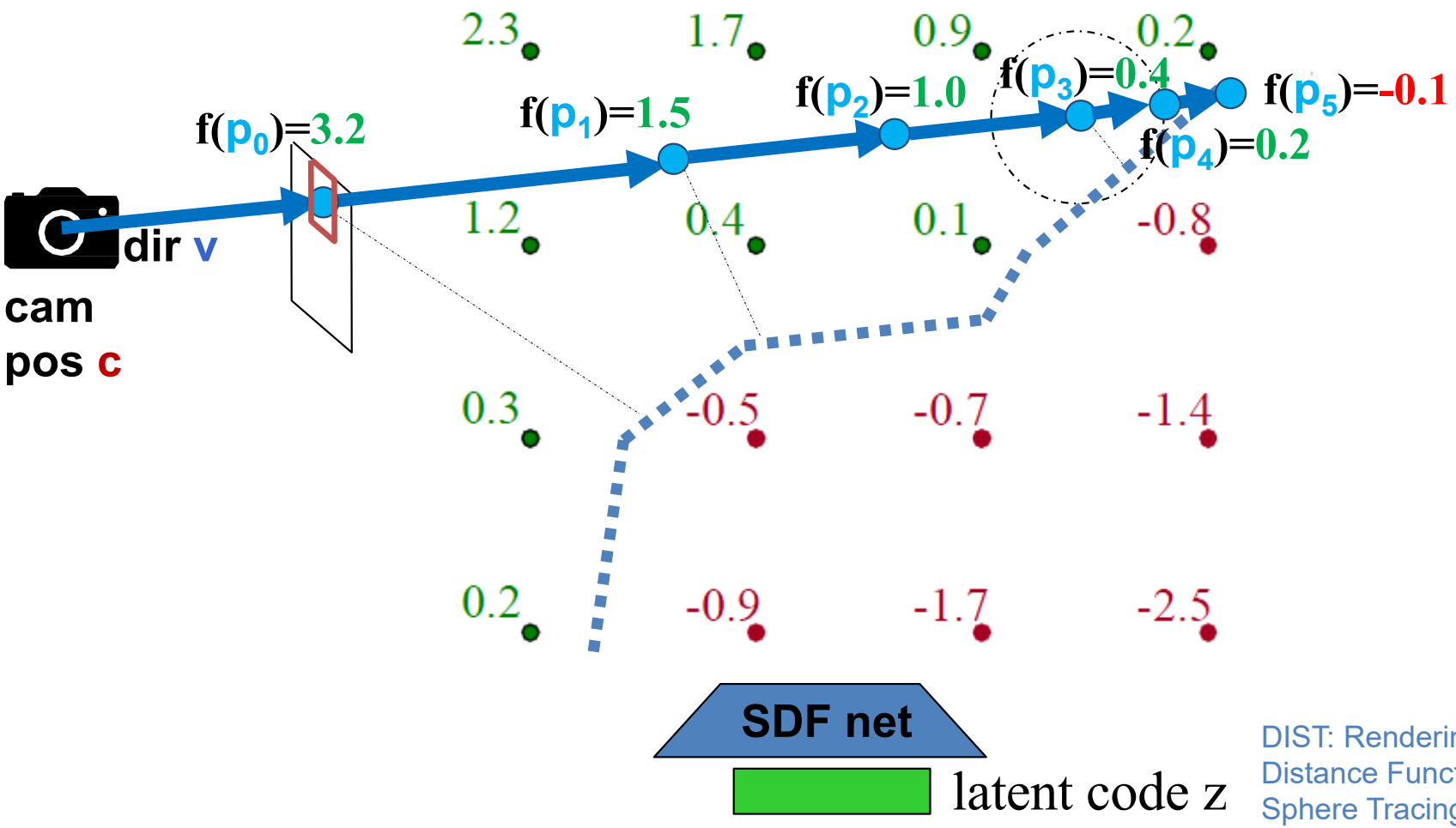
Follow the path of the ray taking steps according to the SDF ("sphere" ray tracing)



DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

How to perform ray tracing for implicits?

... stop when you are very close to the surface based on a threshold OR when the ray comes out of the grid limits (hits background)

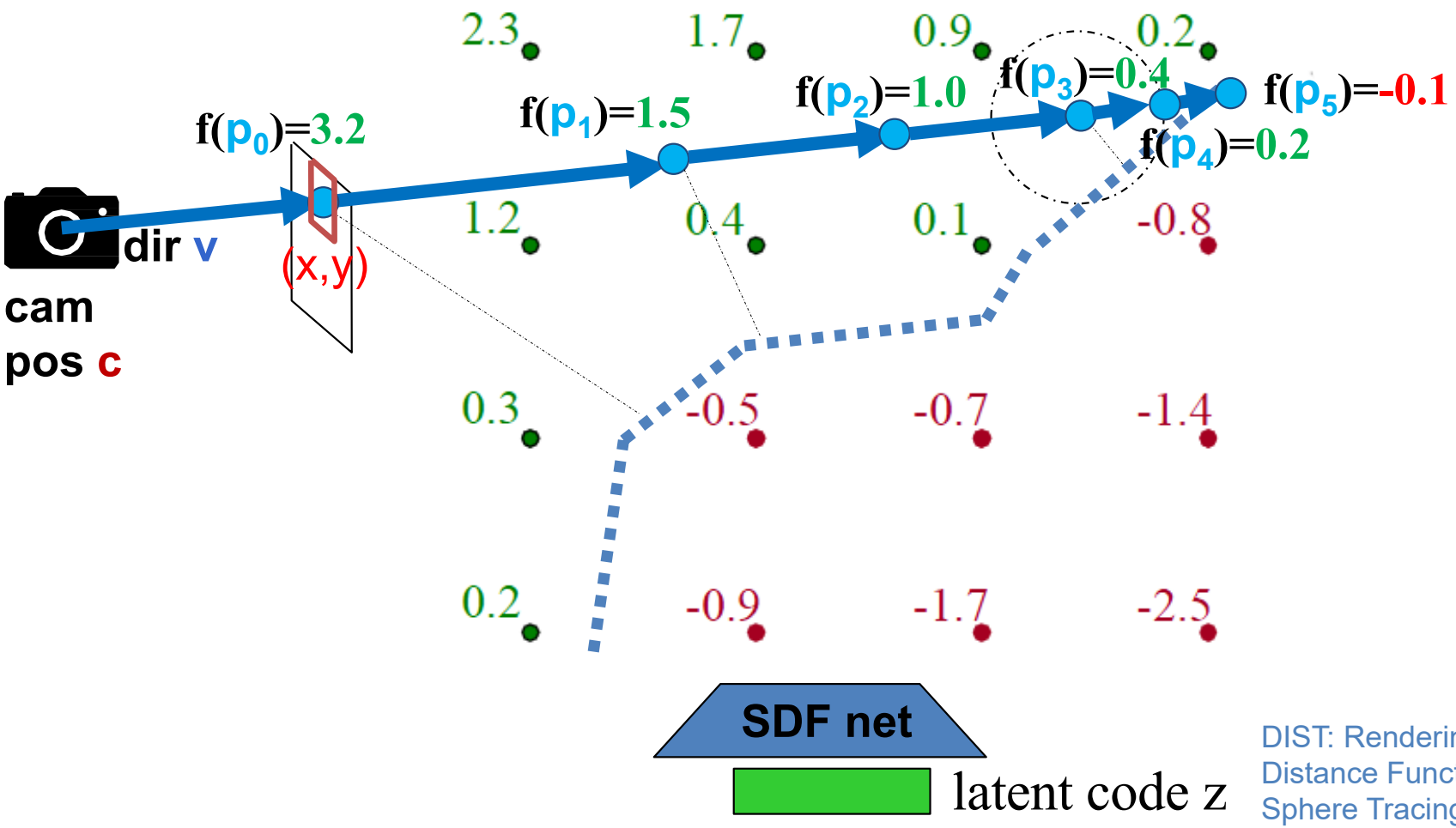


DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

How to perform ray tracing for implicits?

Distance traveled for that ray:

$$d(x, y) = \sum_{n=0}^{N} f(\mathbf{p_n})$$

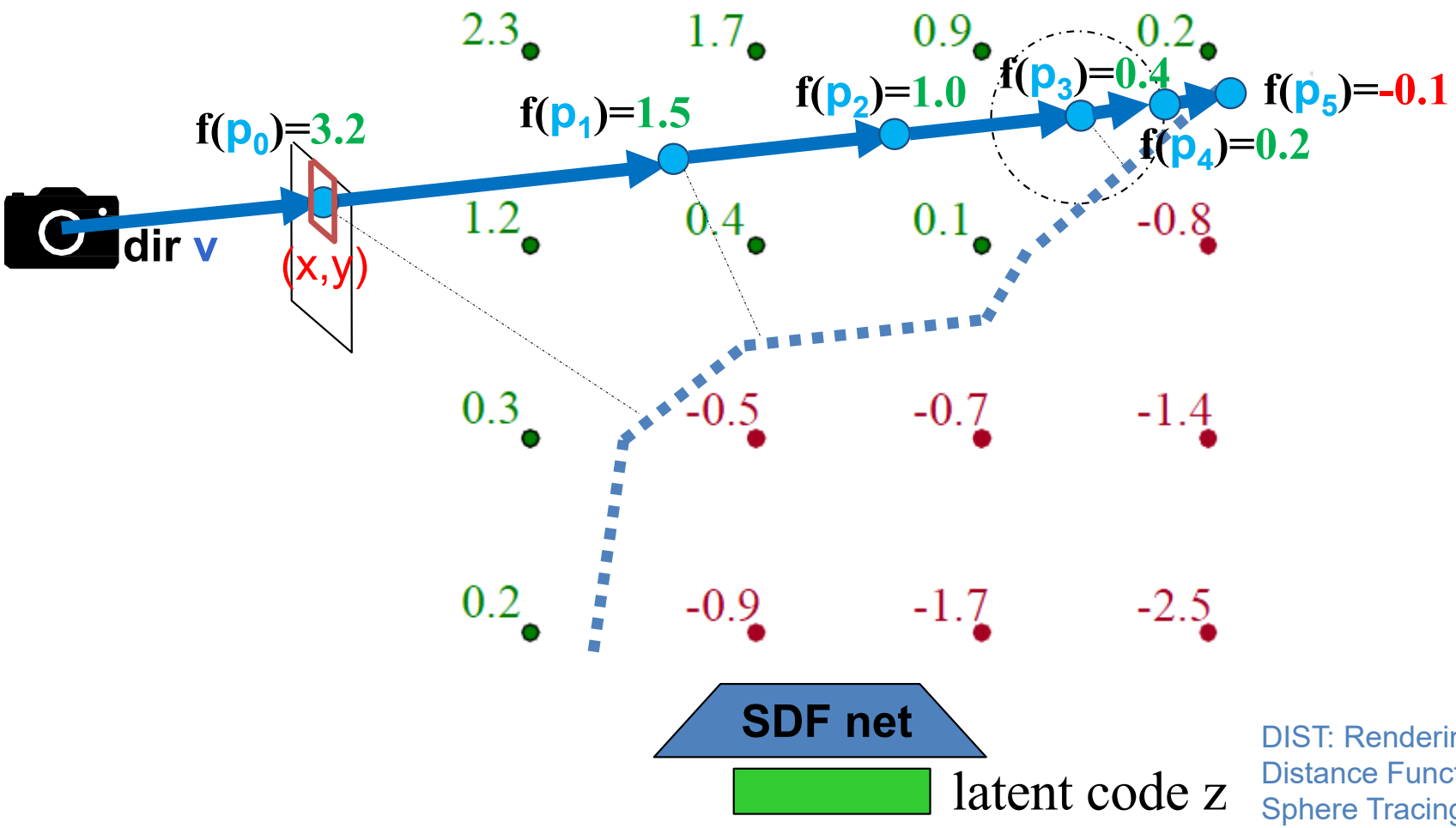


DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

Differentiable sphere ray tracing

Distance traveled is **differentiable**! ($\boldsymbol{\theta}$ are the network parameters)

$$\frac{\partial \mathrm{d}(\mathrm{x},\mathrm{y})}{\partial \boldsymbol{\theta}} = \sum_{n=0}^{N} \frac{\partial \mathrm{f}(\mathbf{p_n})}{\partial \boldsymbol{\theta}}$$

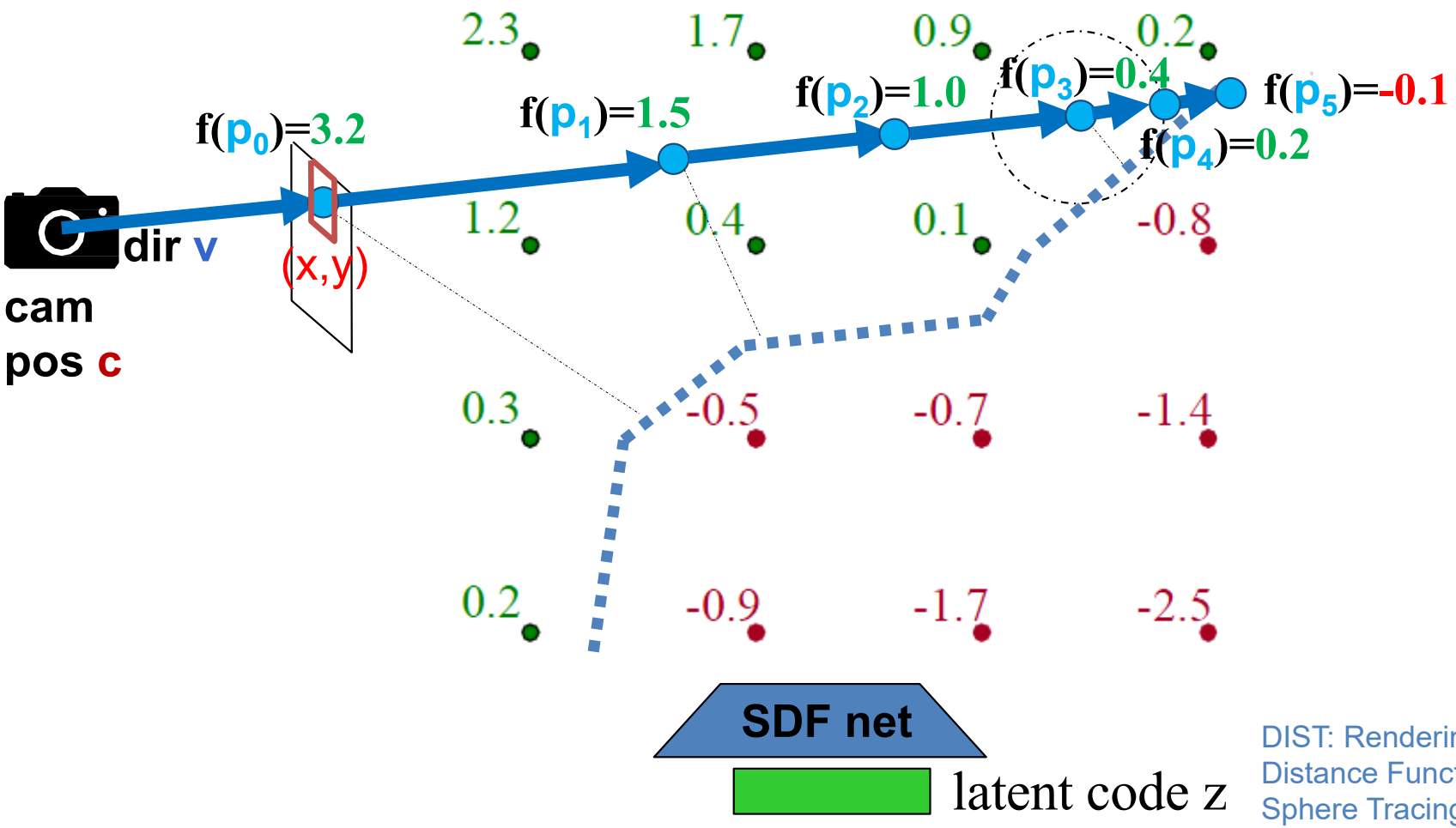


DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

How to perform ray tracing for implicits?

Intersection point can be computed from camera position **c**, ray direction **v**, and distance traveled:

$$\mathbf{p}(x, y) = \mathbf{c} + d(x, y)\,\mathbf{v}$$ (**differentiable**)



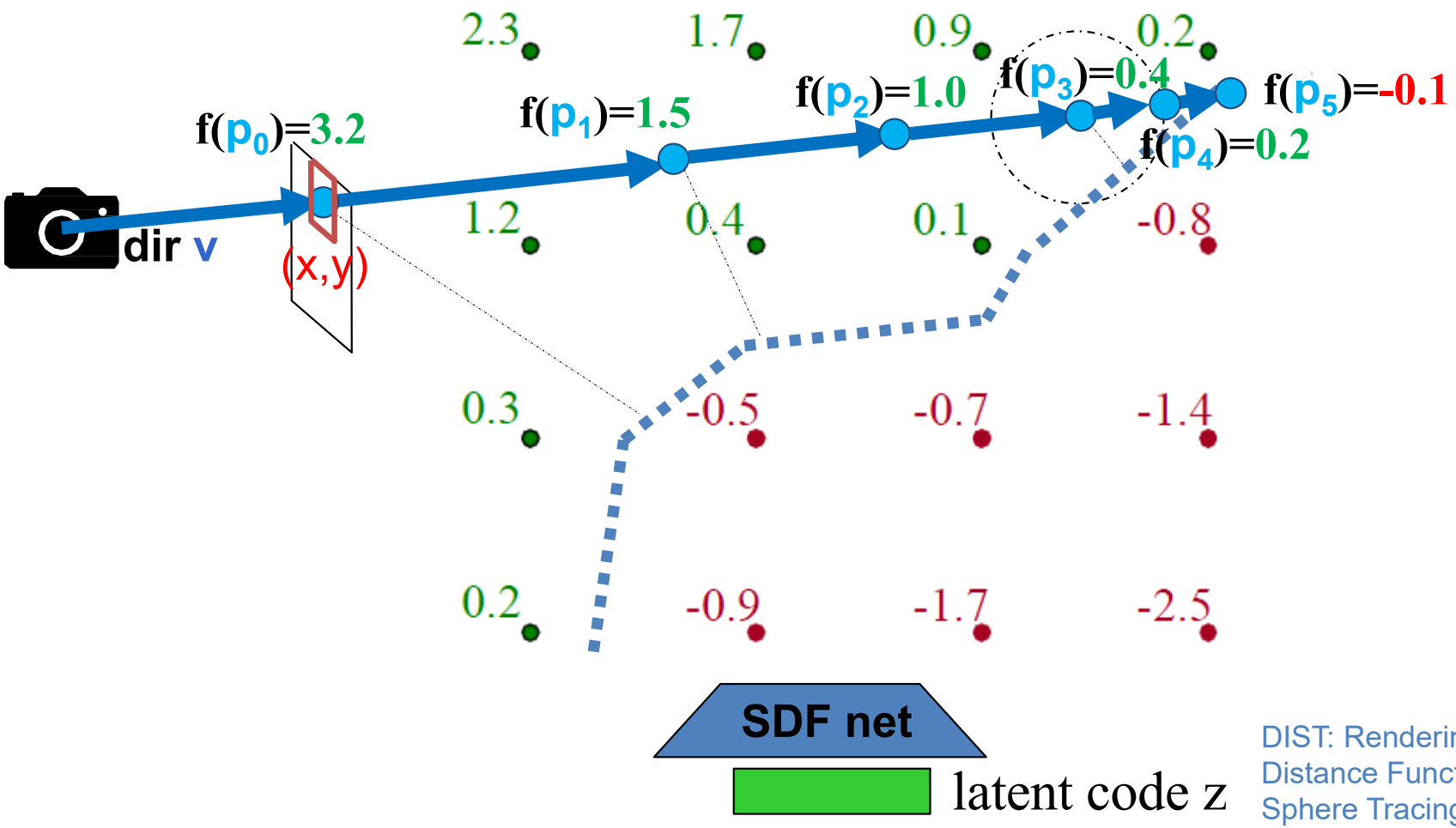
DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

Differentiable sphere ray tracing

Surface normal observed at pixel (x,y) ... also **differentiable**:

$$\mathbf{n}(x, y) = \frac{\nabla f(\mathbf{p}_N)}{||\nabla f(\mathbf{p}_N)||}$$

(... thus basic shading can also be **differentiable**)



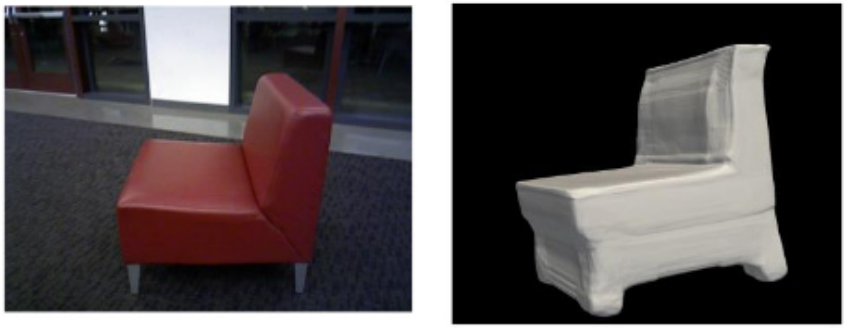
DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

A few results

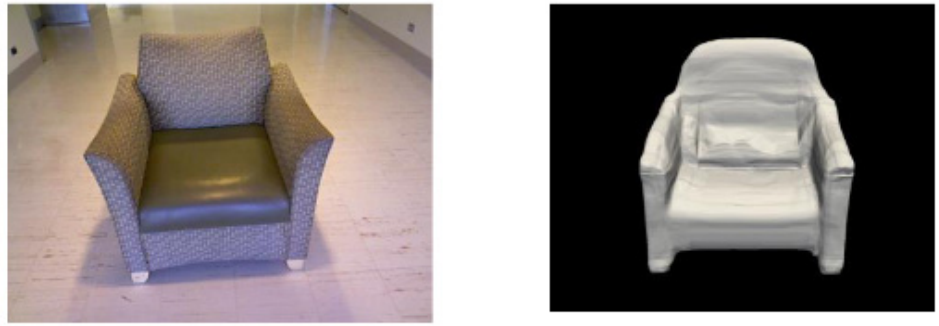
Synthetic #1

Synthetic #2

Real-world #1



Real-world #2



DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

A few results

Synthetic #1

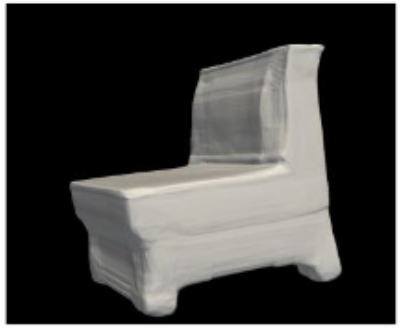
Synthetic #2

Appearance/texture is missed

Intersection points might be unreliable

(esp at early stages of training)

Real-world #1



Real-world #2

DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing, Liu et al., CVPR 2020

Neural Volumetric Rendering

Neural Volumetric

Rendering

querying the radiance value
along rays through 3D space

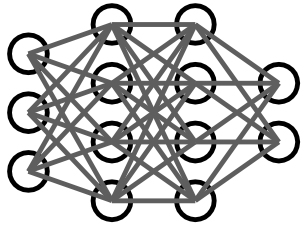
What color?

Neural **Volumetric** Rendering

continuous, differentiable
rendering model without
concrete ray/surface intersections

Neural Volumetric Rendering

using a neural network as a scene representation, rather than a voxel grid of data

(x, y, z) →  → Scene properties

Multi-layer Perceptron (Neural Network)

Given a set of sparse views of an object with known camera poses

Optimize a NeRF model

3D reconstruction viewable from any angle

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, Mildenhall et al. ECCV20

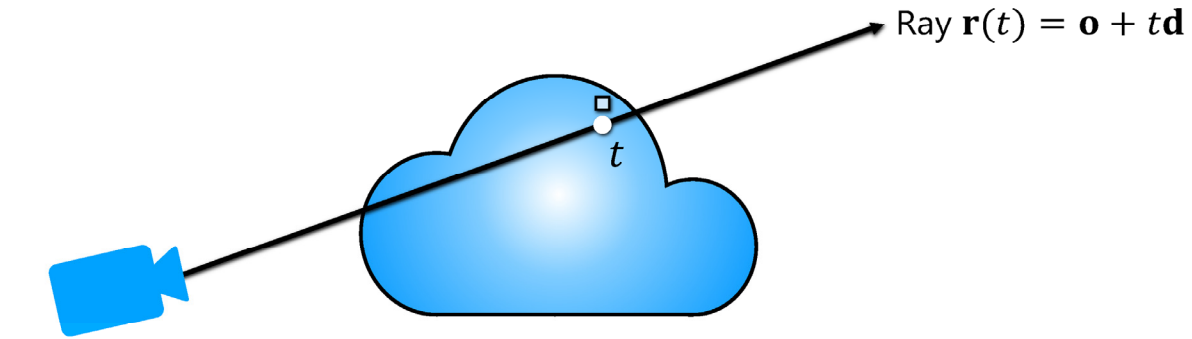
Volumetric formulation for NeRF

Scene is a cloud of colored fog

Max and Chen 2010, Local and Global Illumination in the Volume Rendering Integral

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, Mildenhall et al. ECCV20

Volumetric formulation for NeRF

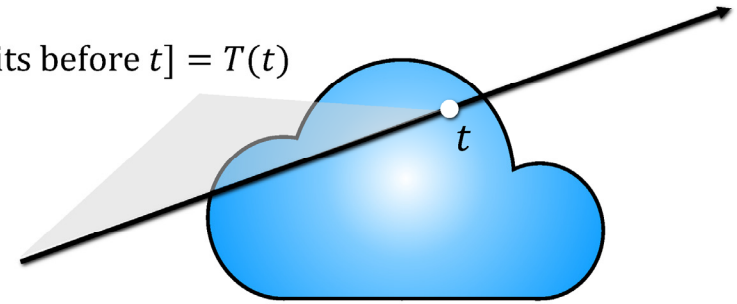


Consider a ray traveling through the scene, and a point at distance t along this ray. We look up its color $\mathbf{c}(t)$, and its opacity (alpha value) $\alpha(t)$

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, Mildenhall et al. ECCV20

Volumetric formulation for NeRF

$P[\text{no hits before } t] = T(t)$



But t may also be blocked by earlier points along the ray. $T(t)$: probability that the ray didn't hit any particles earlier.

$T(t)$ is called "transmittance"

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, Mildenhall et al. ECCV20

Volume rendering estimation: integrating color along a ray

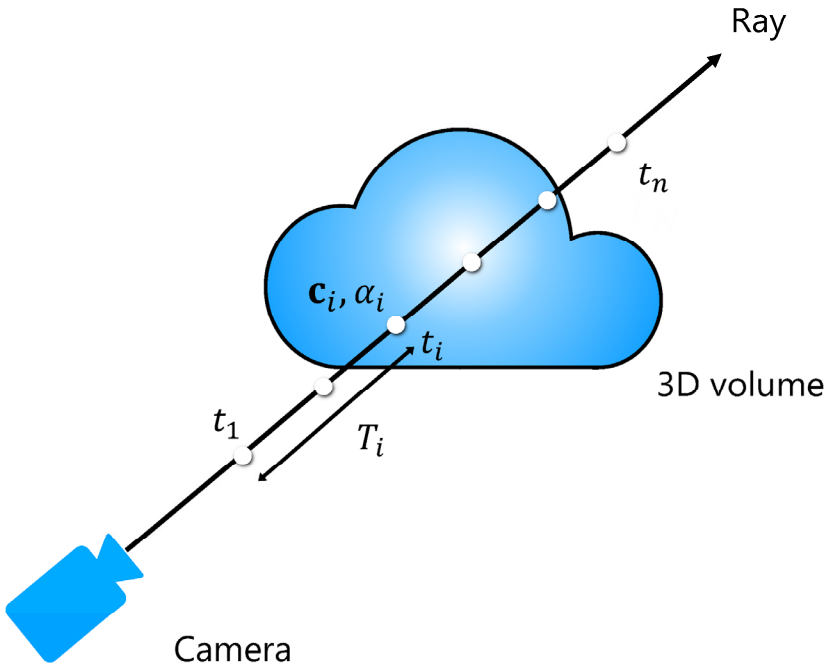
Rendering model for ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$:

$$\mathbf{c} \approx \sum_{i=1}^{n} T_i \alpha_i \mathbf{c}_i$$

final rendered color along ray

weights

colors



NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, Mildenhall et al. ECCV20

Volume rendering estimation: integrating color along a ray

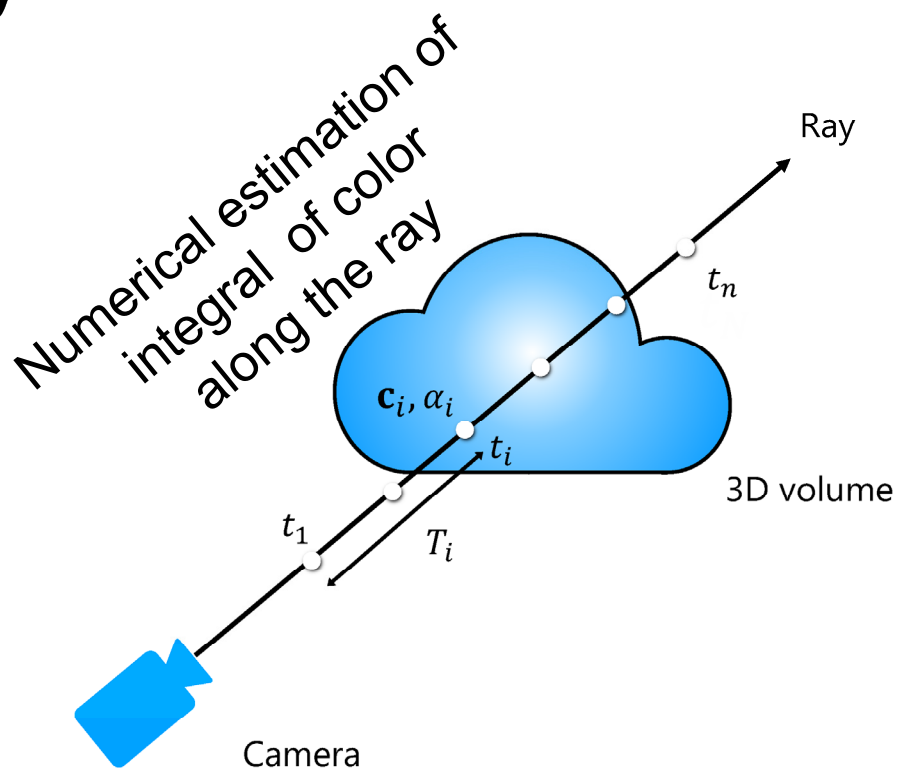
Rendering model for ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$:

$$\mathbf{c} \approx \sum_{i=1}^{n} T_i \alpha_i \mathbf{c}_i$$

final rendered color along ray

weights

colors



NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, Mildenhall et al. ECCV20

Volume rendering estimation: integrating color along a ray

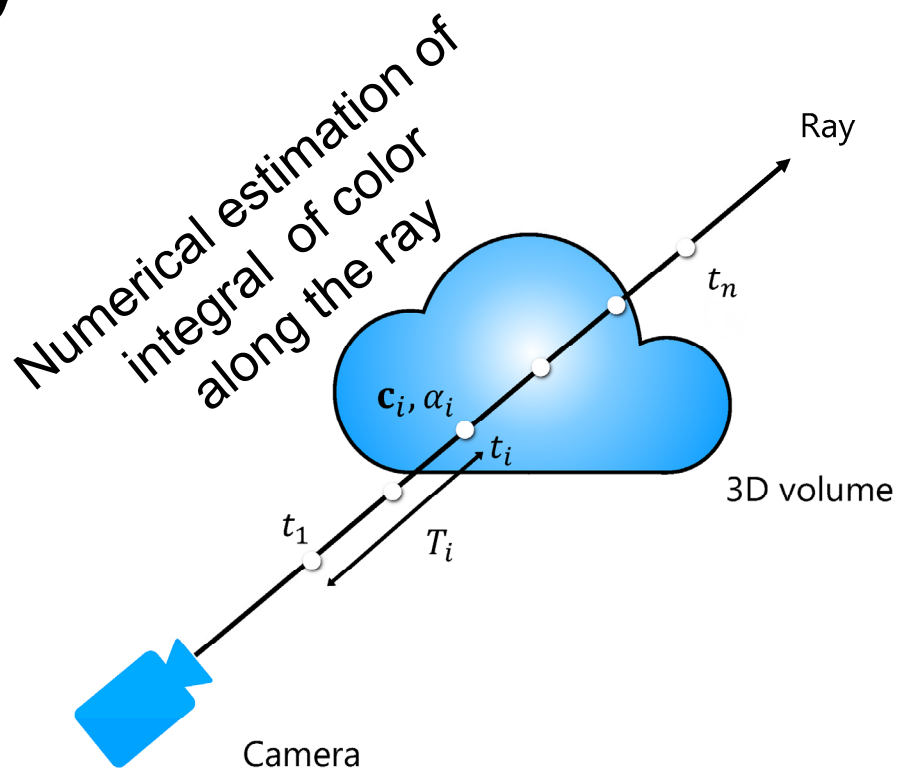
Rendering model for ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$:

$$\mathbf{c} \approx \sum_{i=1}^{n} T_i \alpha_i \mathbf{c}_i$$

final rendered color along ray

weights

colors



Computing the color for a set of rays through the pixels of an image yields a rendered image

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, Mildenhall et al. ECCV20

Volume rendering estimation: integrating color along a ray

Rendering model for ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$:

$$\mathbf{c} \approx \sum_{i=1}^{n} T_i \alpha_i \mathbf{c}_i$$

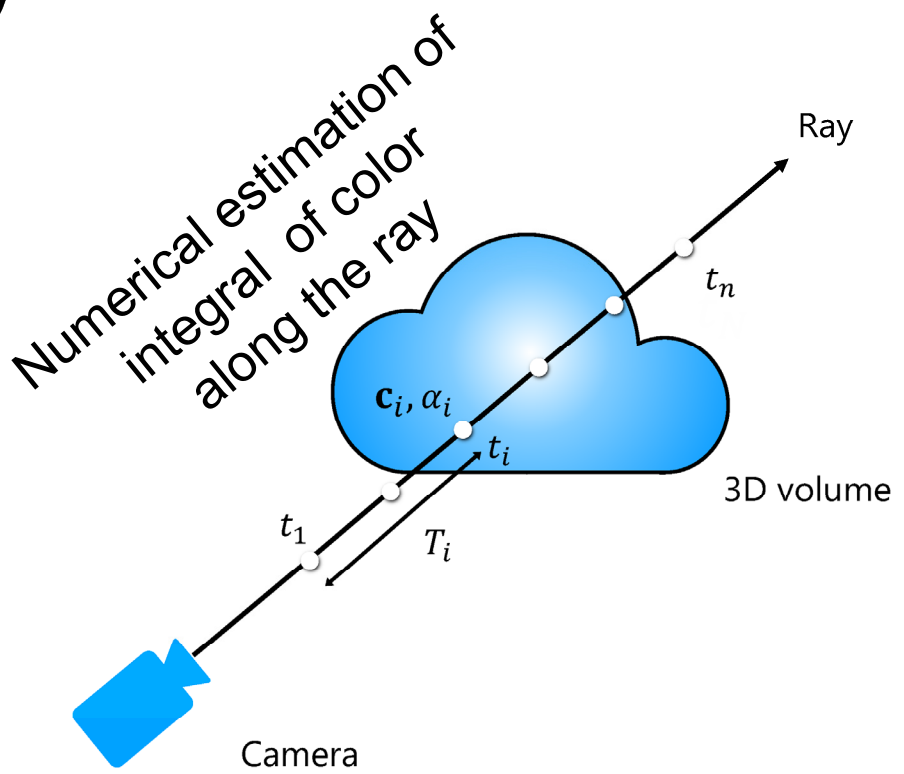
final rendered color along ray

weights

colors

How much light is blocked earlier along ray:

$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$$



NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, Mildenhall et al. ECCV20

Volume rendering estimation: integrating color along a ray

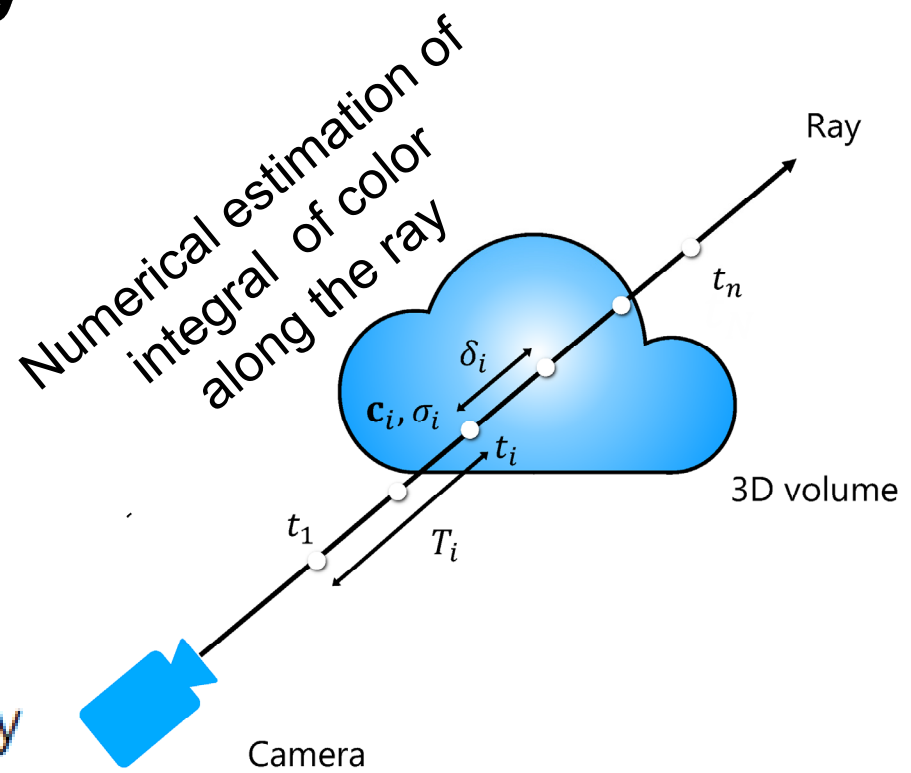
Rendering model for ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$:

$$\mathbf{c} \approx \sum_{i=1}^{n} T_i \alpha_i \mathbf{c}_i$$

final rendered color along ray — weights — colors

How much light is blocked earlier along ray:

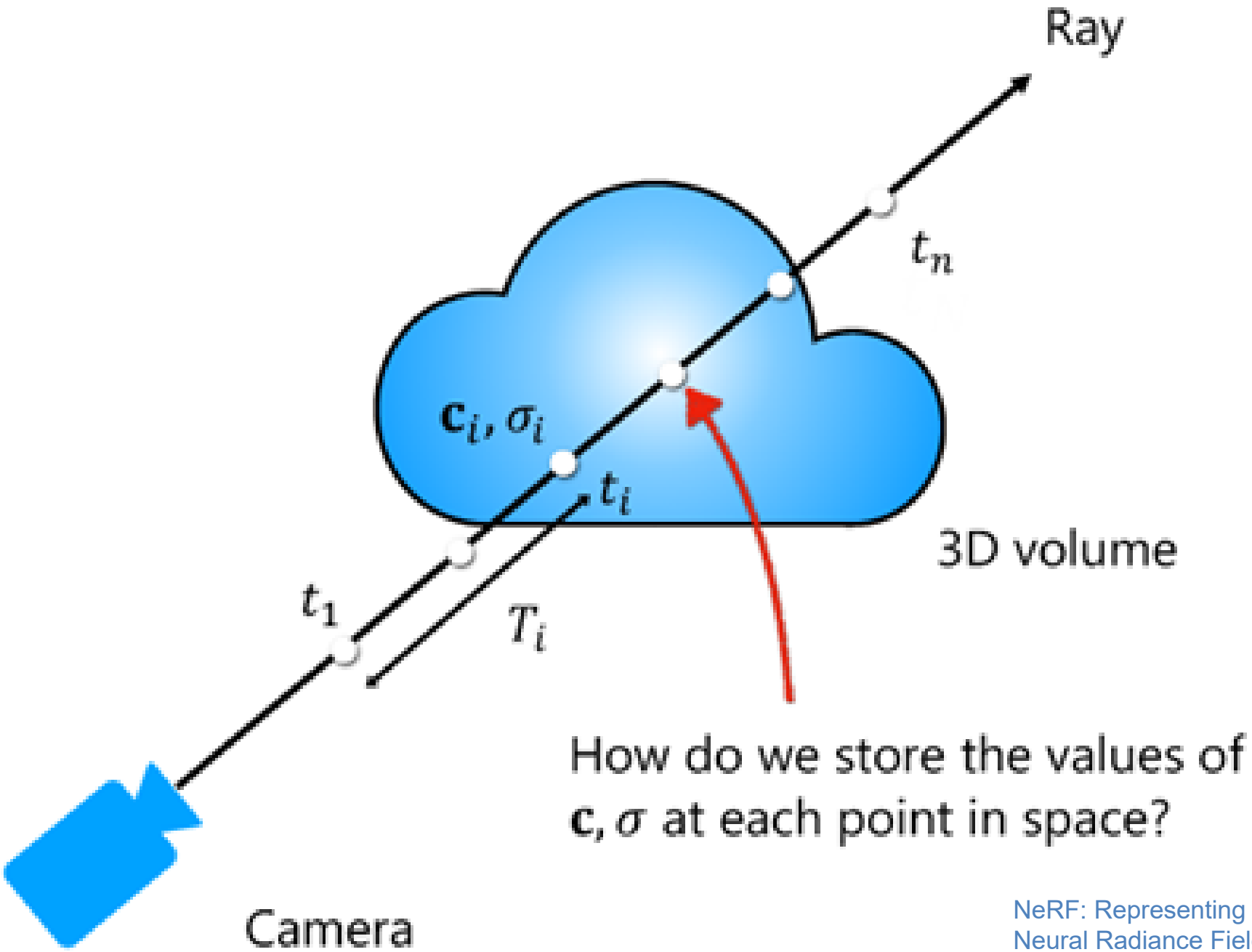
$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$$



α is derived from a stored volume density sigma (σ) multiplied by the distance between samples delta δ :

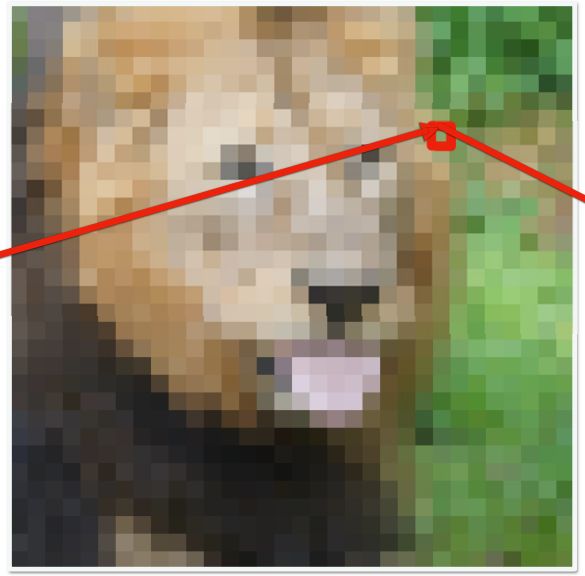
$$\alpha_i = 1 - \exp(-\sigma_i \delta_i)$$

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, Mildenhall et al. ECCV20



NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, Mildenhall et al. ECCV20

Toy problem: storing 2D image data

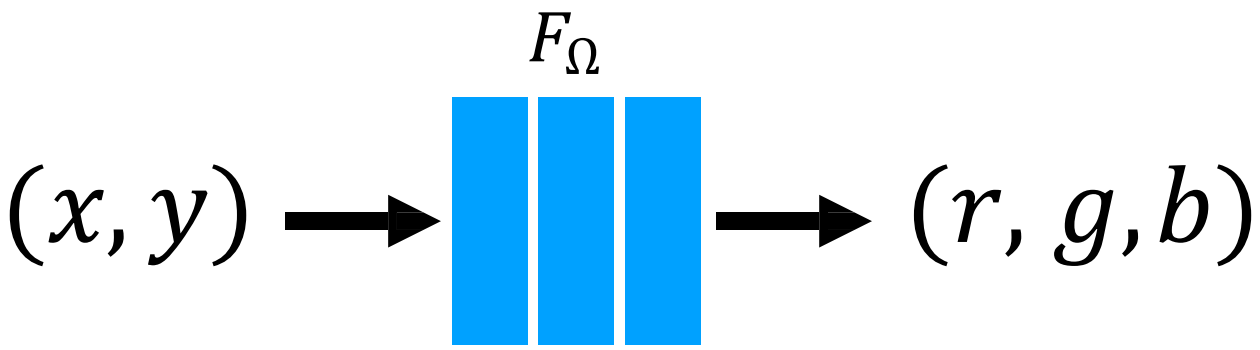


(x, y)

(r, g, b)

Usually we store an image as a
2D grid of RGB color values

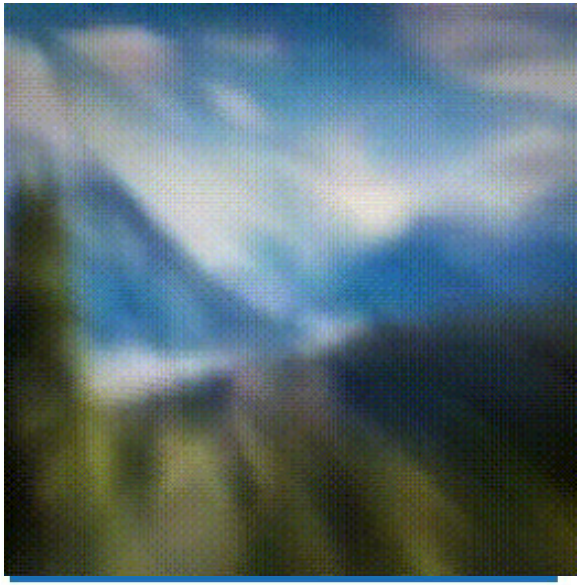
Toy problem: storing 2D image data



What if we train a simple fully-connected network (MLP) to do this instead?

Naive approach fails!

Ground truth image



Neural network output fit
with gradient descent

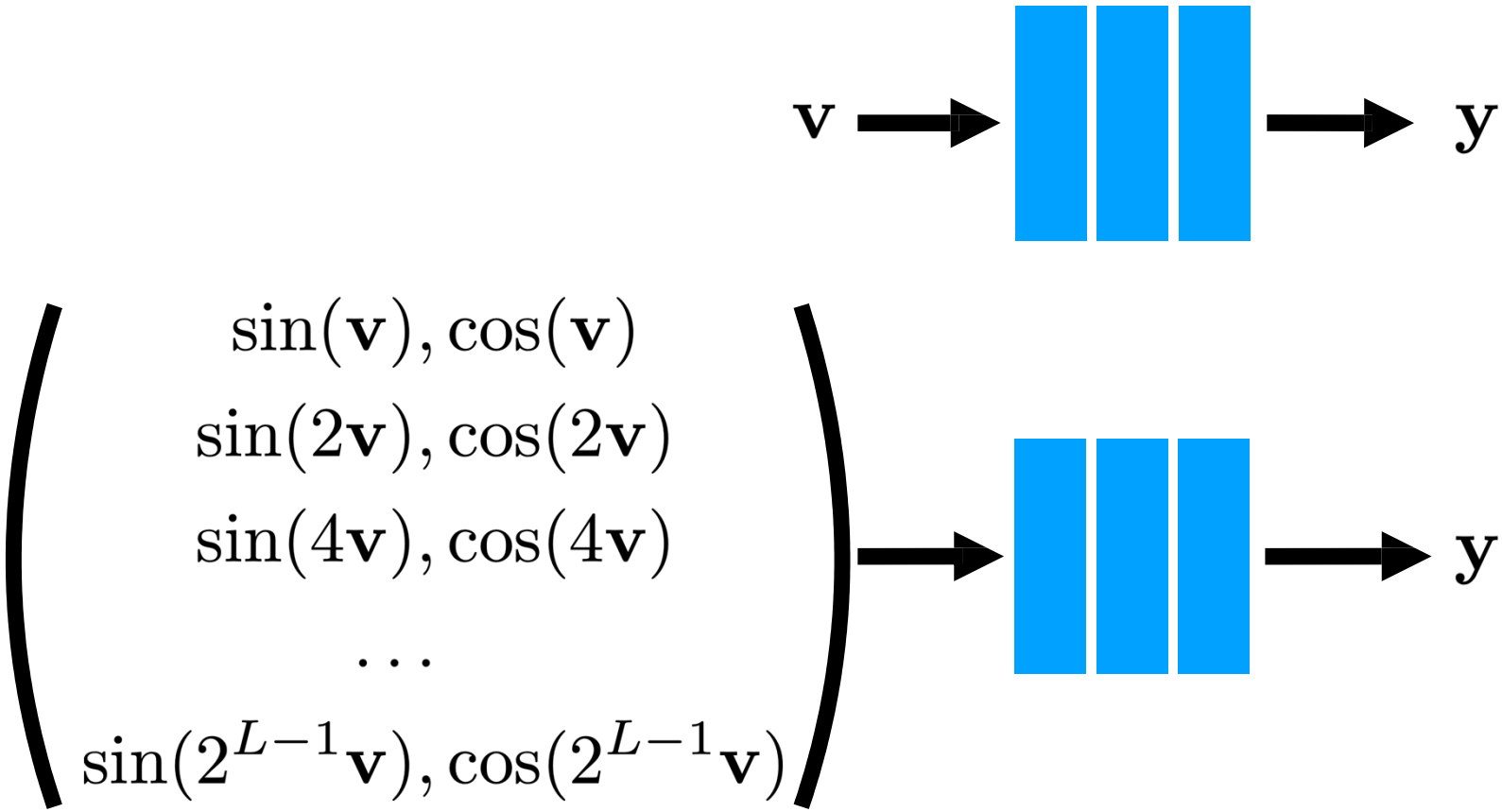
Problem:

"Standard" coordinate-based MLPs cannot represent high frequency functions

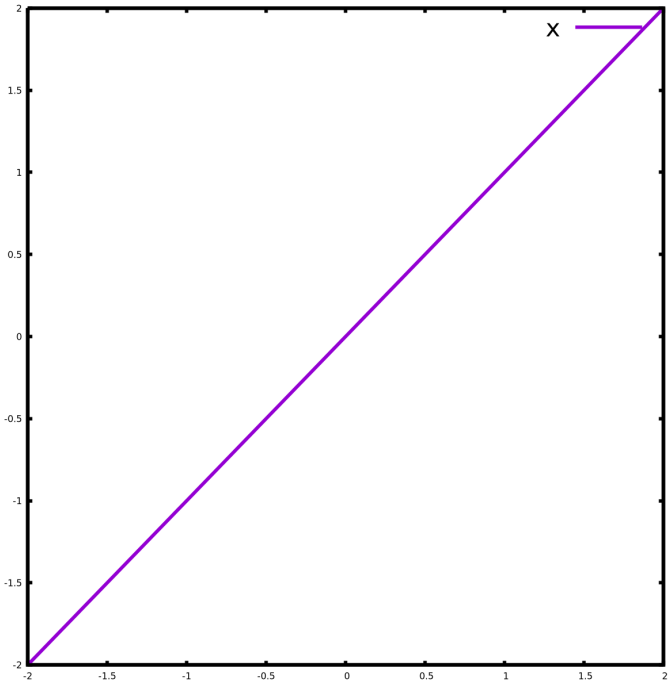
Solution:

Pass input coordinates through a high frequency mapping first

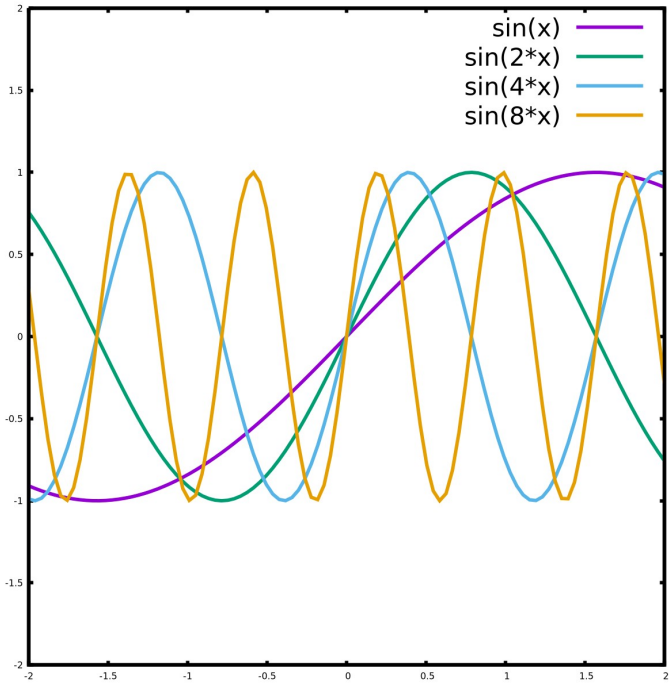
Example mapping: “positional encoding”



Positional encoding



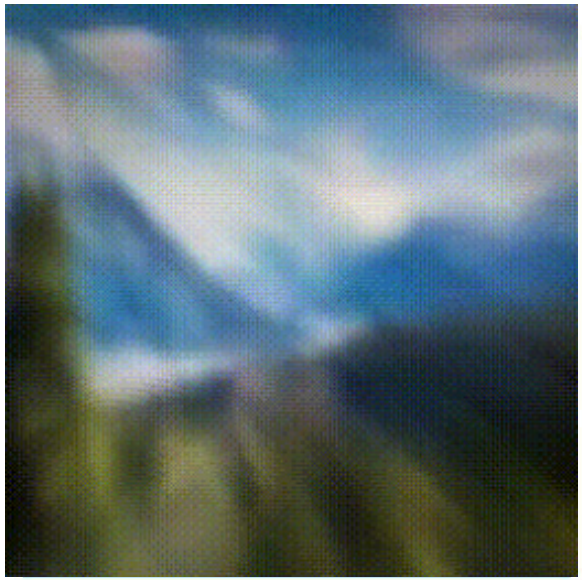
Raw encoding of a number ***x***



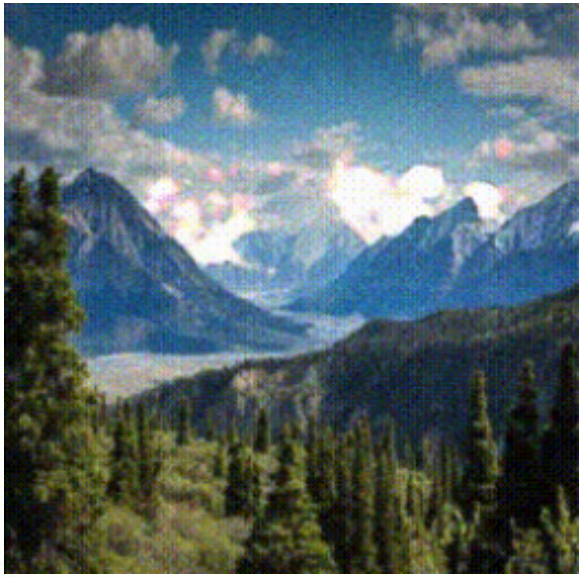
"Positional encoding" of a number ***x***

Problem solved!

Ground truth image



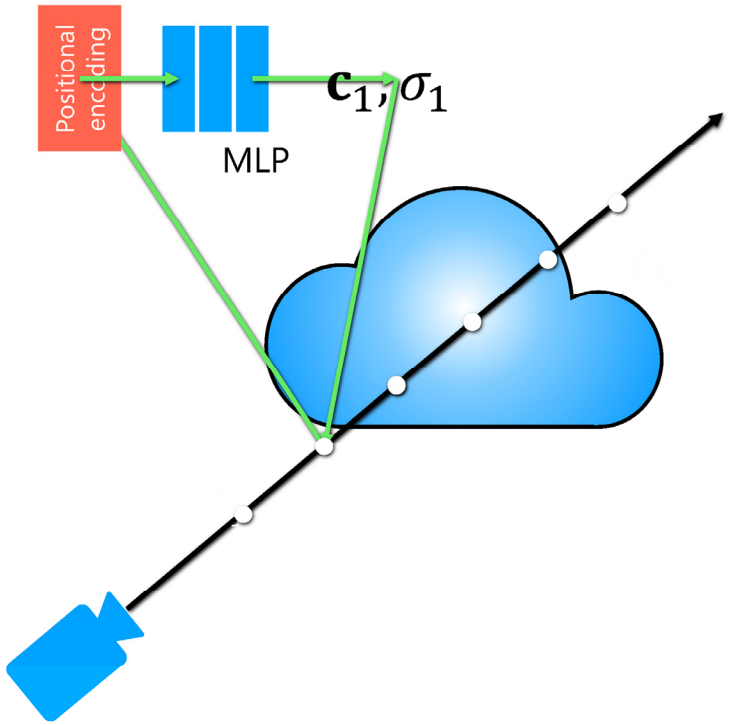
Neural network output without high frequency mapping



Neural network output with high frequency mapping

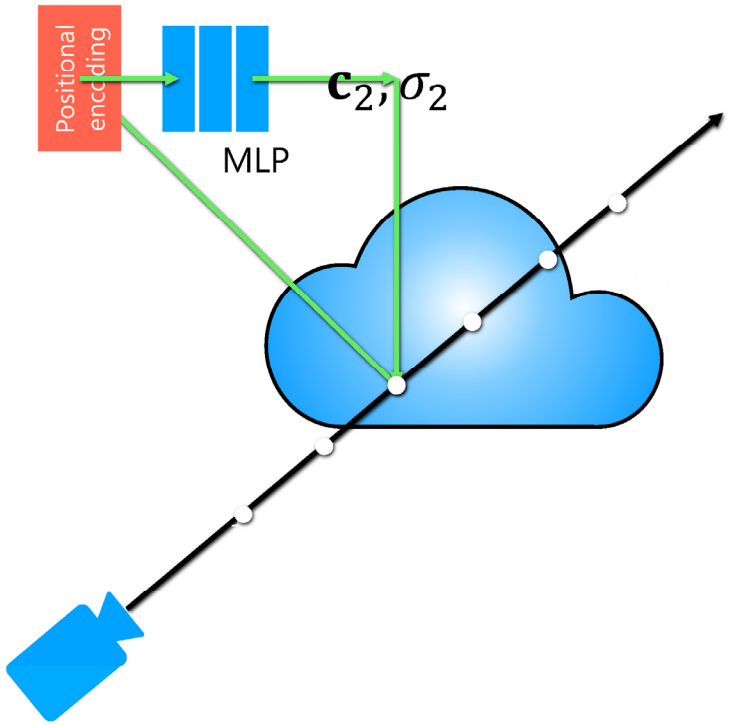
NeRF = volume rendering + coordinate-based network

How do we store the values of $\mathbf{c}, \sigma$ at each point in space



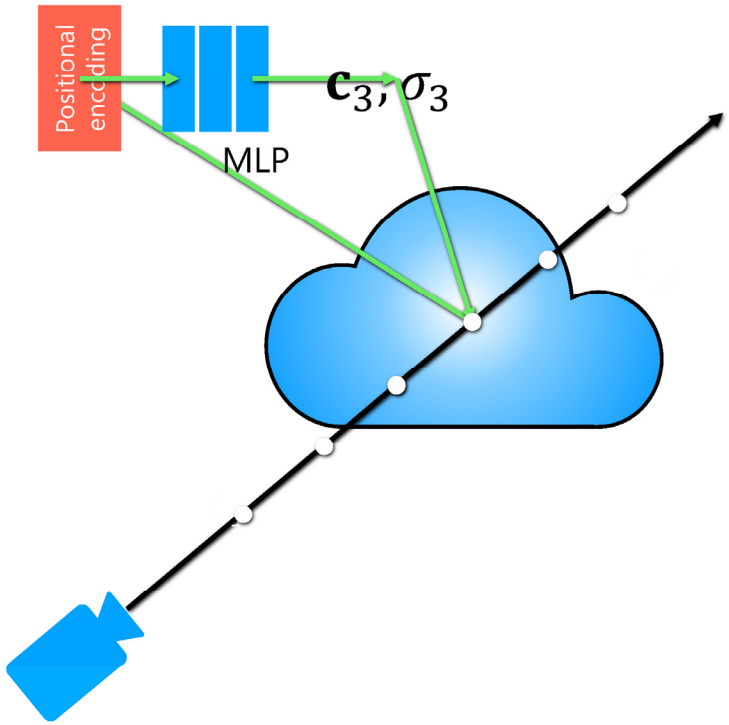
NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, Mildenhall et al. ECCV20

How do we store the values of $\mathbf{c}, \sigma$ at each point in space



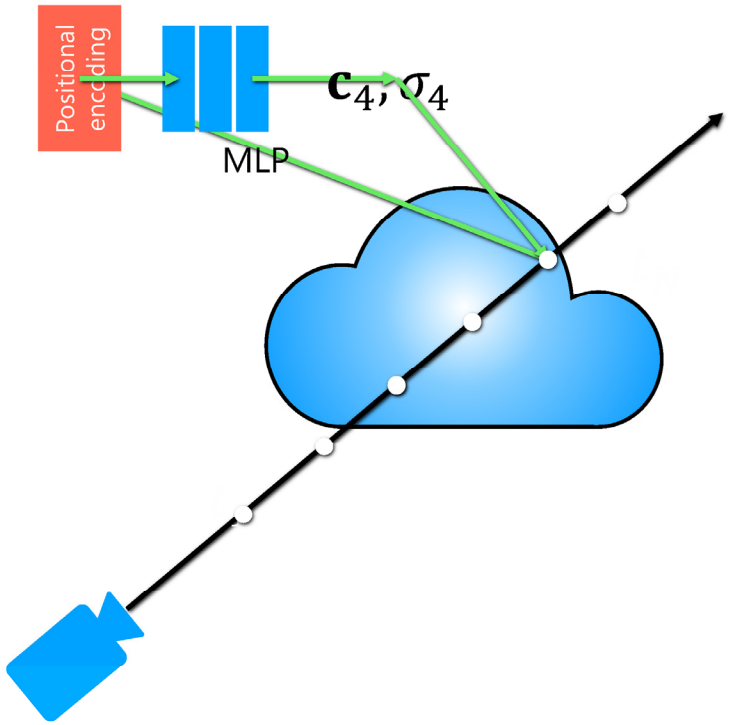
NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, Mildenhall et al. ECCV20

How do we store the values of $\mathbf{c}, \sigma$ at each point in space



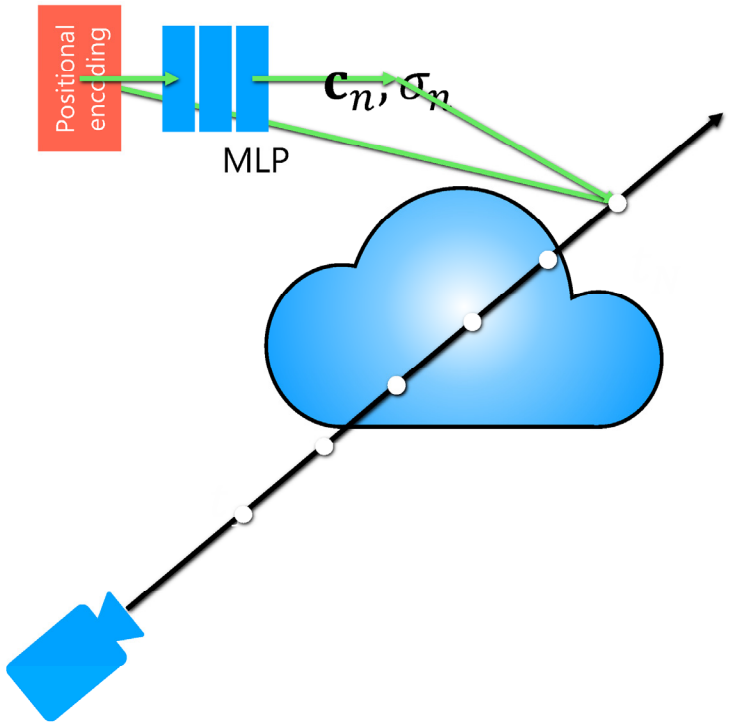
NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, Mildenhall et al. ECCV20

How do we store the values of $\mathbf{c}, \sigma$ at each point in space



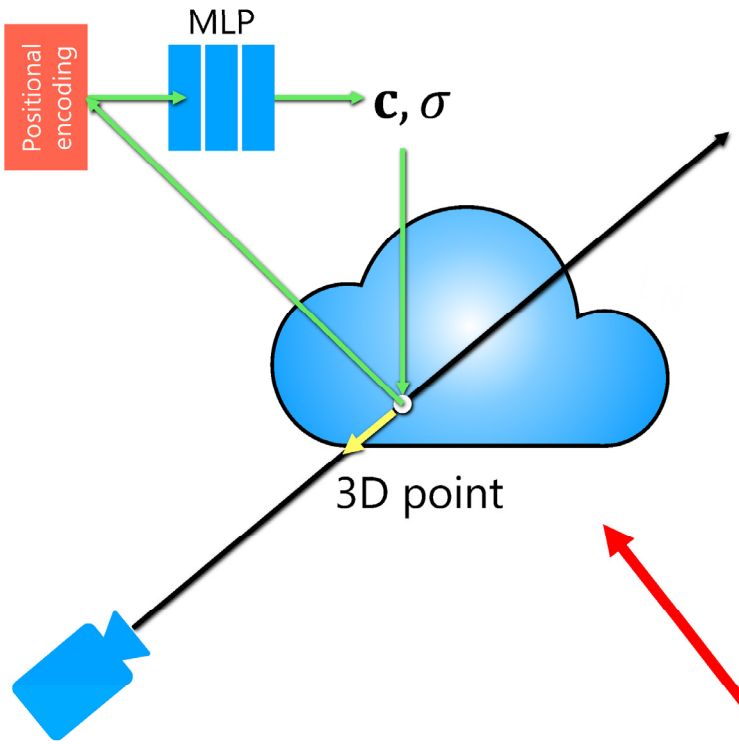
NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, Mildenhall et al. ECCV20

How do we store the values of $\mathbf{c}, \sigma$ at each point in space



NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, Mildenhall et al. ECCV20

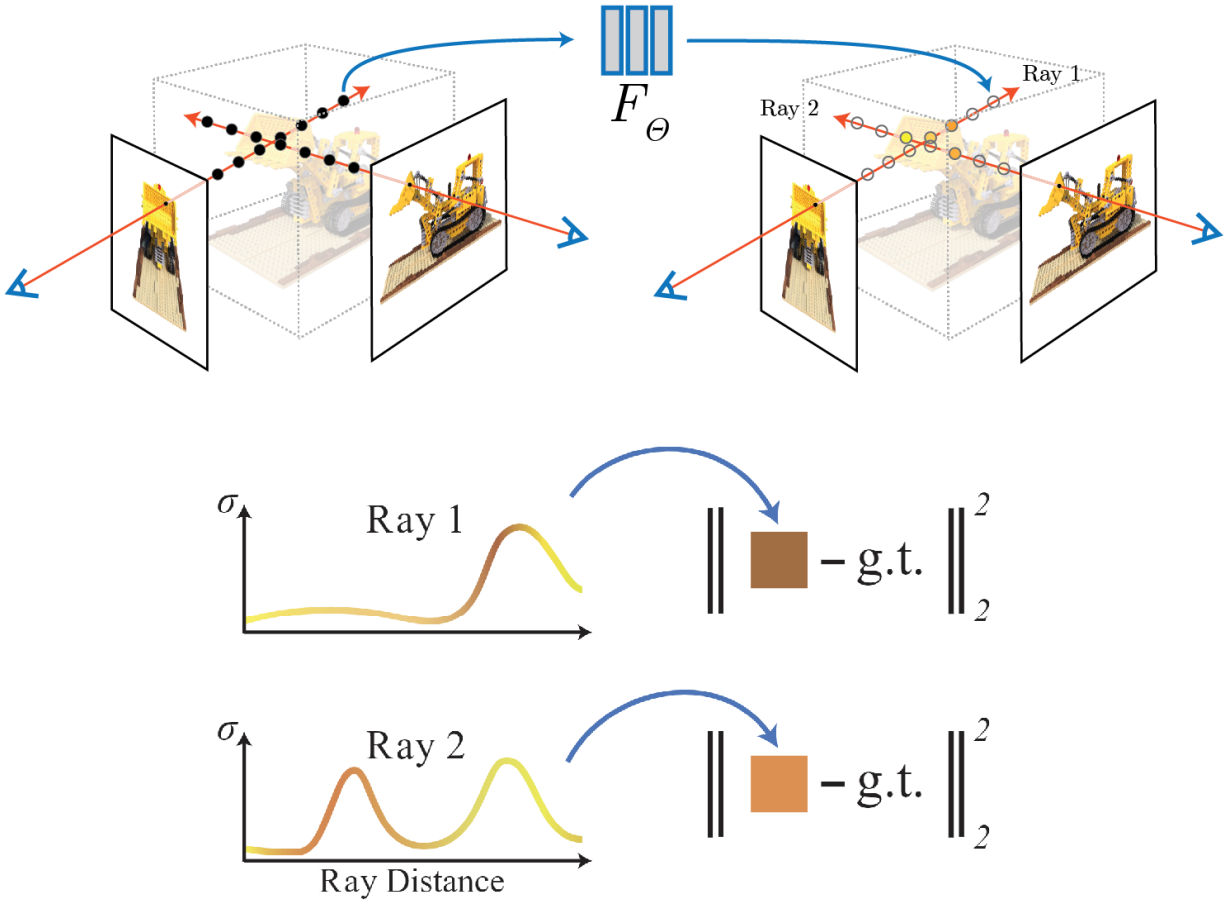
Extension: view-dependent field



Include the ray direction in the input to the MLP → allows for capturing and rendering view-dependent effects (e.g., shiny surfaces)

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, Mildenhall et al. ECCV20

Putting it all together



NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, Mildenhall et al. ECCV20

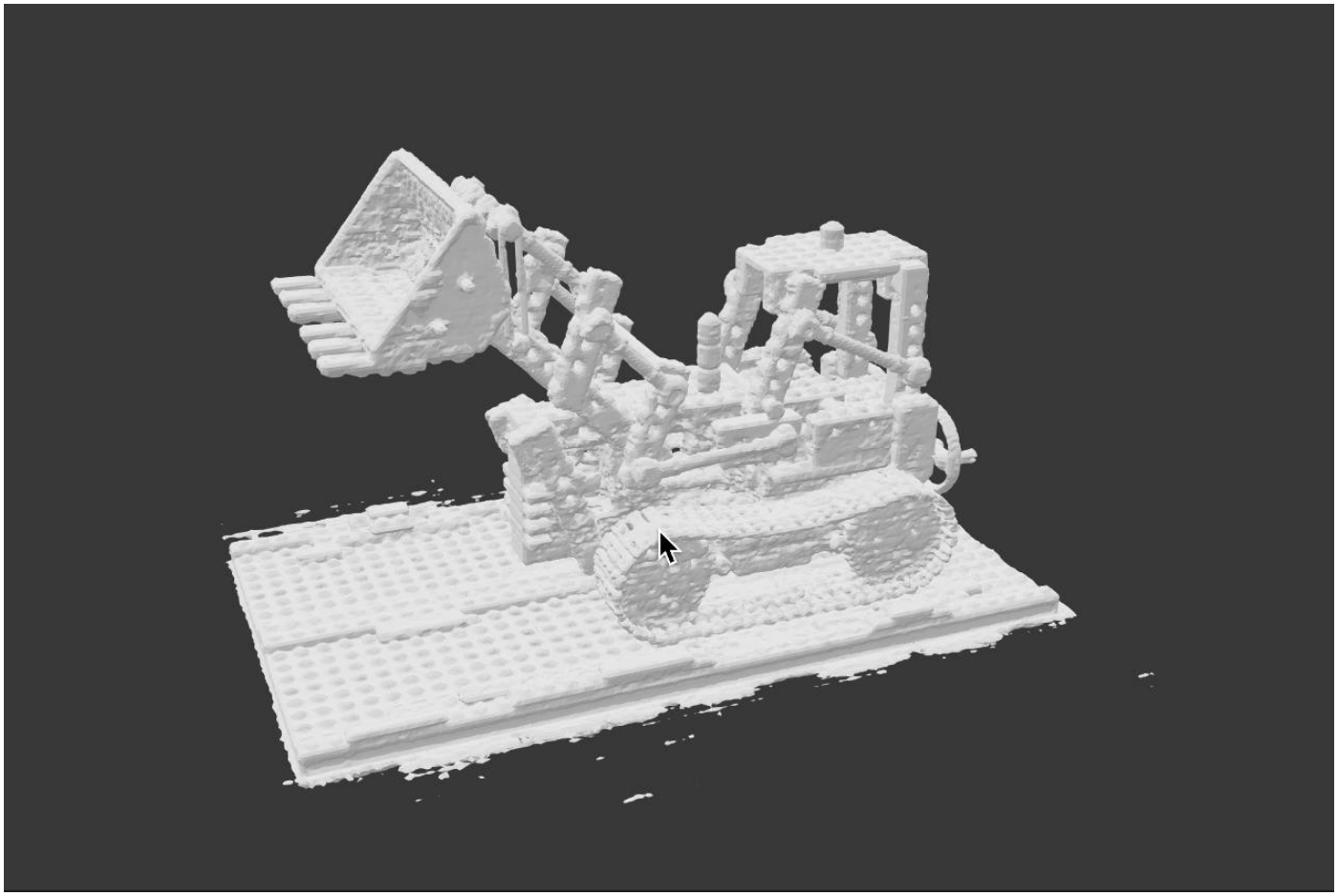
Results

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis

Ben Mildenhall* UC Berkeley
Pratul P. Srinivasan* UC Berkeley
Matthew Tancik* UC Berkeley
Jonathan T. Barron Google Research
Ravi Ramamoorthi UC San Diego
Ren Ng UC Berkeley

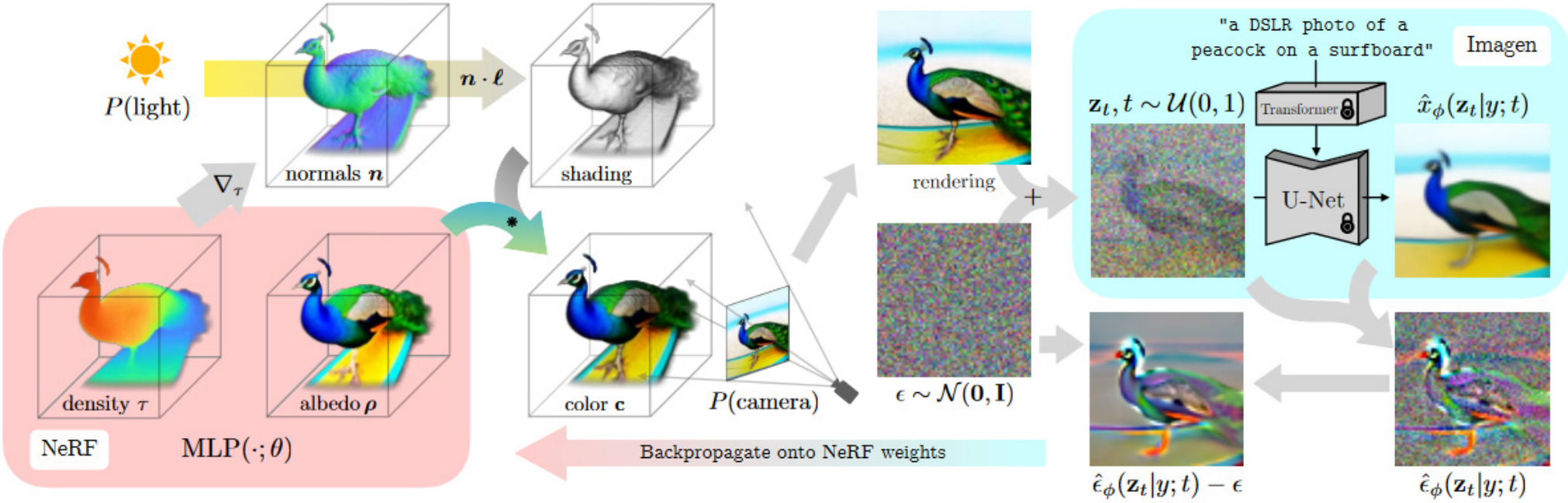
* Denotes Equal Contribution

NeRF encodes detailed scene geometry



DreamFusion!

https://dreamfusion3d.github.io/



Train NERF scene representation
guided by text-conditioned diffusion model

What we covered

1. Image and Shape Representations
2. Introduction to Neural Networks
3. 3D Deep Learning: the multi-view approach
4. 3D Deep Learning: the voxel-based approach
5. 3D Deep Learning: the point-based approach
6. 3D Deep Learning: the mesh-based approach
7. 3D Generative Models: GANs, VAEs, diffusion
8. 3D Generative models: Generating views, voxels, octrees, points, implicits, meshes
9. Neural Rendering

What we covered

1. Image and Shape Representations
2. Introduction to Neural Networks
3. 3D Deep Learning: the multi-view approach
4. 3D Deep Learning: the voxel-based approach
5. 3D Deep Learning: the point-based approach
6. 3D Deep Learning: the mesh-based approach
7. 3D Generative Models: GANs, VAEs, diffusion
8. 3D Generative models: Generating views, voxels, octrees, points, implicits, meshes
9. Neural Rendering

Many open research challenges / questions:

What is the right 3D representation to generate for each task?

Generate interactive 3D models (not static) – neural 4D reconstruction/animation

Zero-shot learning – test on categories not seen during training

Leverage text + 2D + 3D supervision, self-supervision

Tasks to finish

- Submit assignment 5 [this Fri, NO EXTENSIONS]
- Presentation for 574 [May 25, 5pm, NO EXTENSIONS]
- Project for 674 [May 25, 5pm, NO EXTENSIONS]