Lecture 23: 3D Vision

Reminder: A5

Recurrent networks, attention, Transformers

Due on **Tuesday 4/12**, 11:59pm ET

A6

Will cover image generation and visualization:

Generative Models: GANs and VAEs

Network visualization: saliency maps, adversarial examples, class visualizations

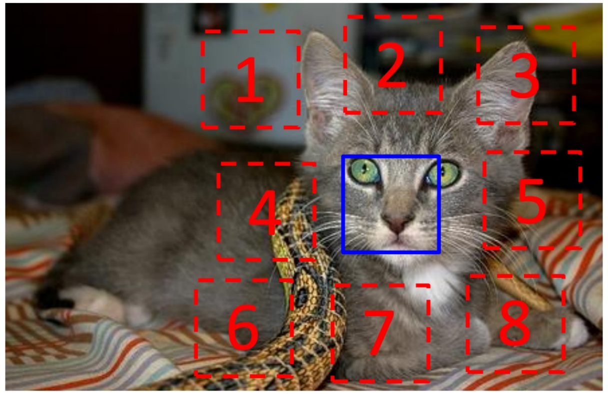
Style Transfer

Should be released tonight; due 2 weeks after release

YOU CANNOT USE LATE DAYS ON A6!!!!

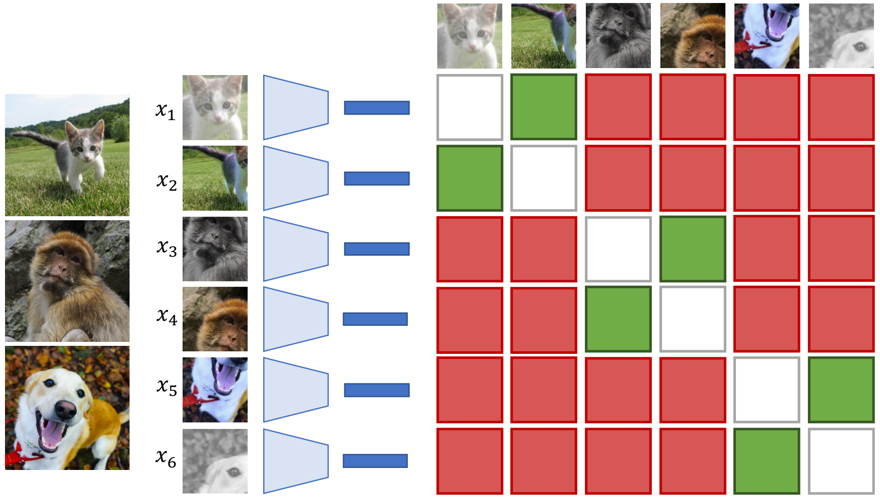
Last Time: Self-Supervised Learning

Context Prediction

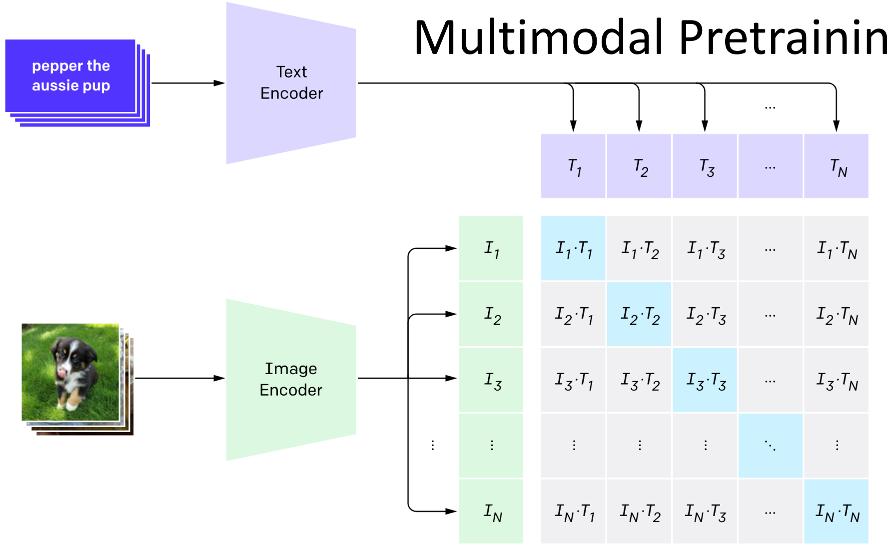


Colorization

Contrastive Learning



Multimodal Pretraining

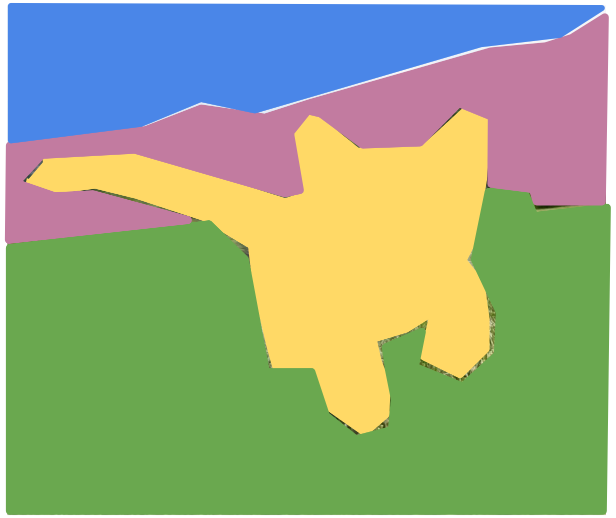


Previously: Predicting 2D Shapes of Objects

Classification

CAT

Semantic Segmentation

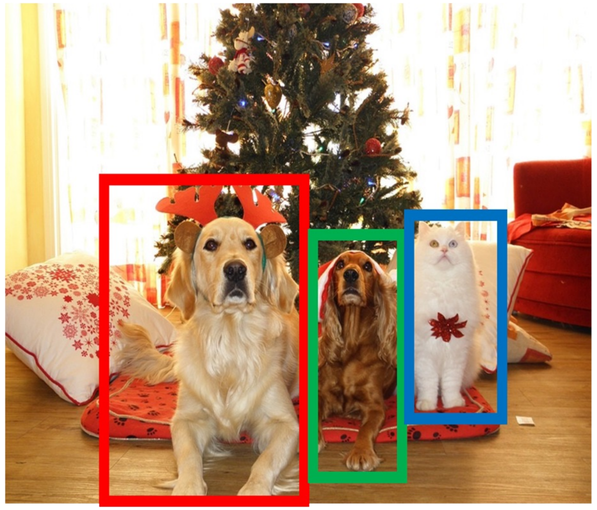


GRASS, CAT, TREE, SKY

No spatial extent

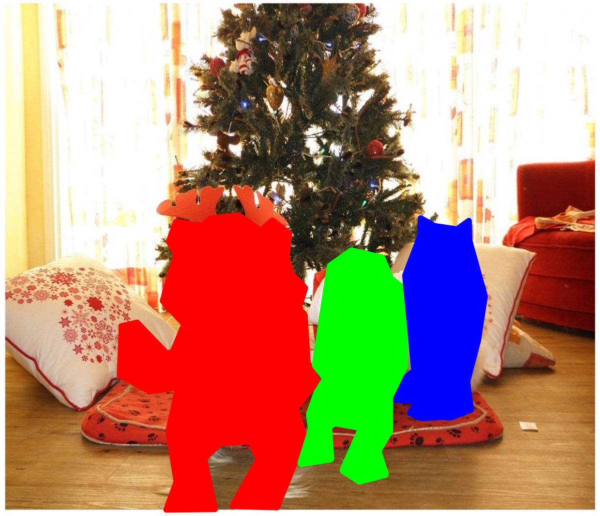
No objects, just pixels

Object Detection



DOG, DOG, CAT

Instance Segmentation

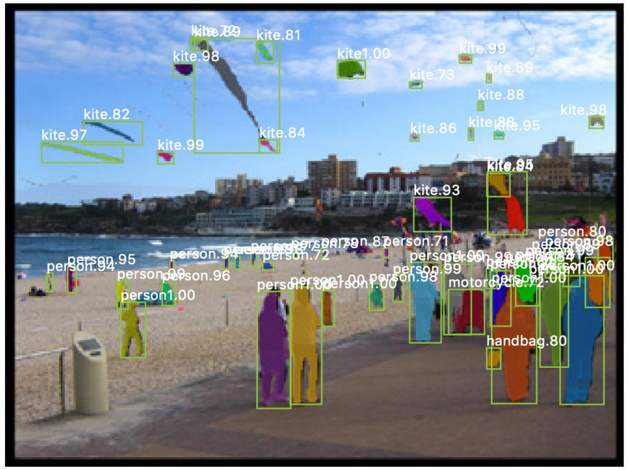


DOG, DOG, CAT

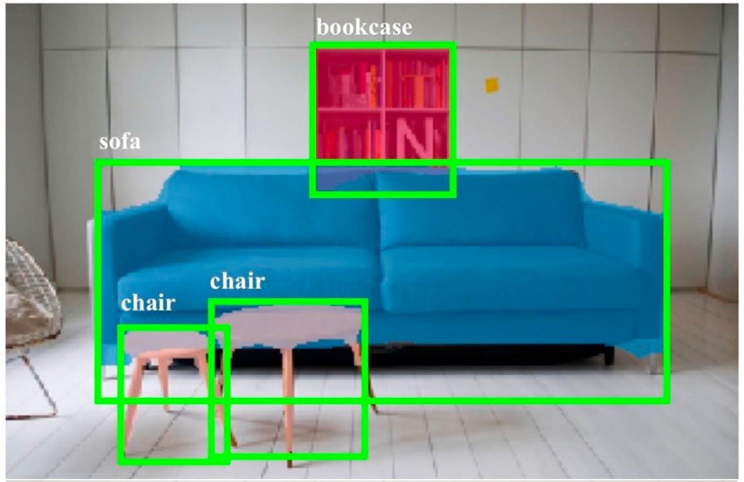
Multiple Objects

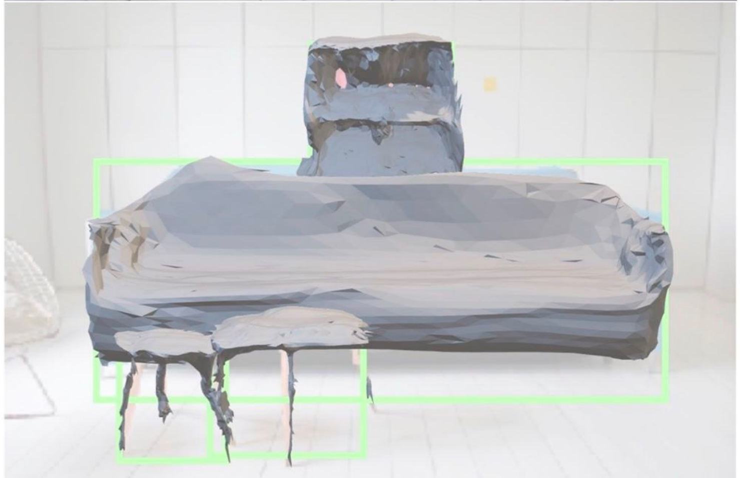
Today: Predicting **3D Shapes of Objects**

Mask R-CNN:
2D Image -> 2D shapes



Mesh R-CNN:
2D Image -> **3D** shapes





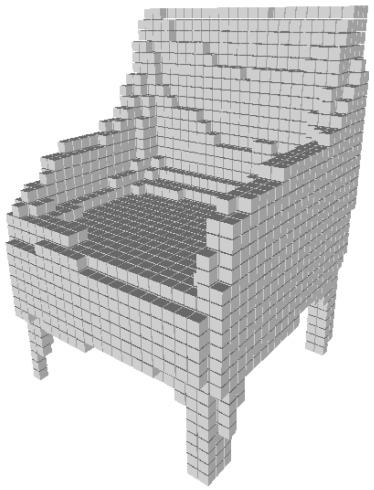
He, Gkioxari, Dollár, and Girshick, “Mask R-CNN”, ICCV 2017

Gkioxari, Malik, and Johnson, “Mesh R-CNN”, ICCV 2019

Focus on Two Problems today

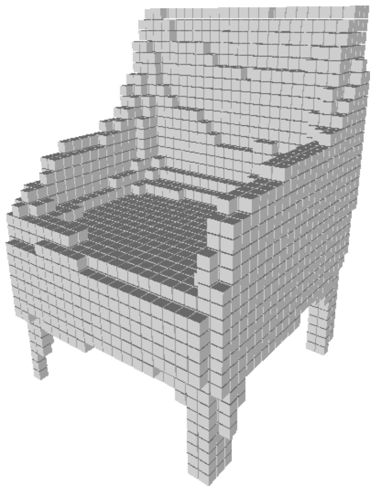
Predicting 3D Shapes from single image

Input Image



3D Shape

Processing 3D input data



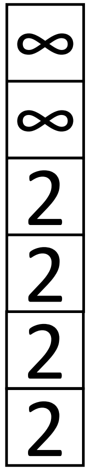
3D Shape

Chair

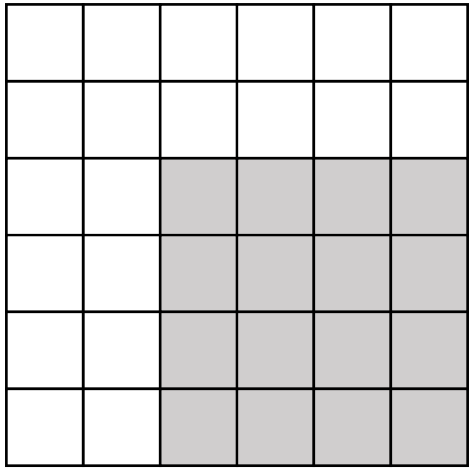
Many more topics in 3D Vision!

Computing correspondences
Multi-view stereo
Structure from Motion
Simultaneous Localization and Mapping (SLAM)
Self-supervised learning
Differentiable graphics
3D Sensors

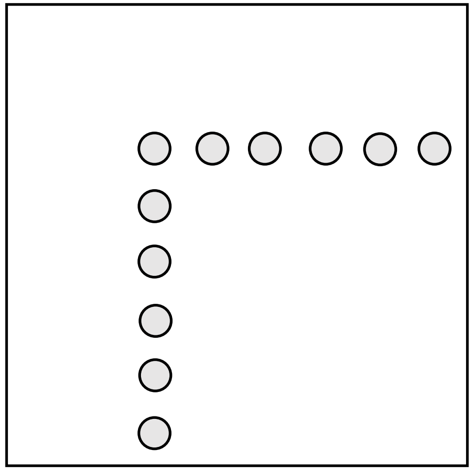
3D Shape Representations



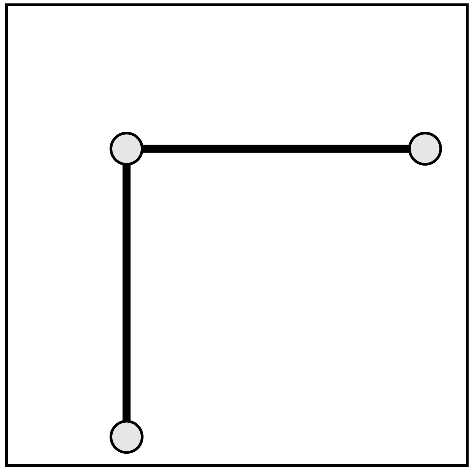
Depth Map



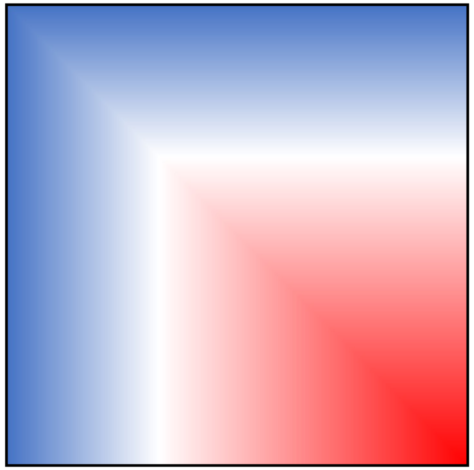
Voxel Grid



Pointcloud

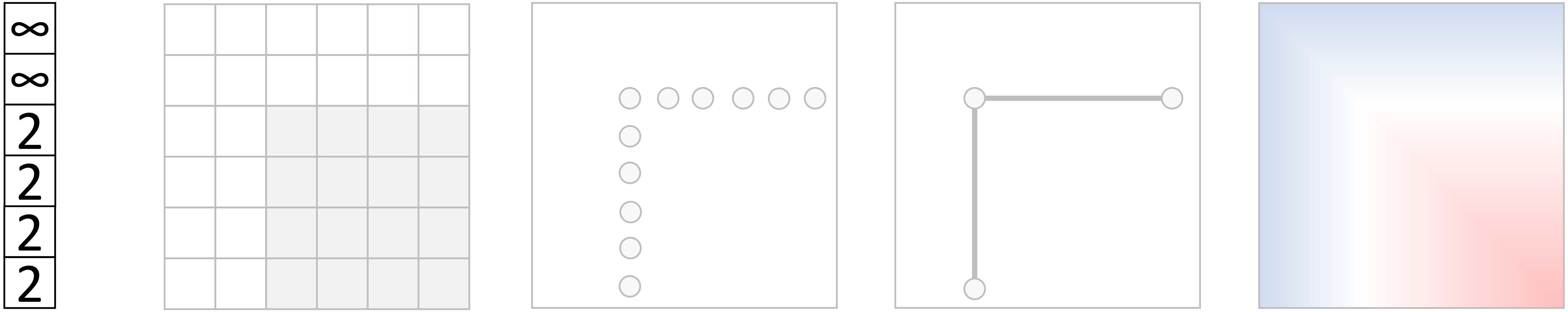


Mesh



Implicit Surface

3D Shape Representations



Depth Map

Voxel Grid

Pointcloud

Mesh

Implicit Surface

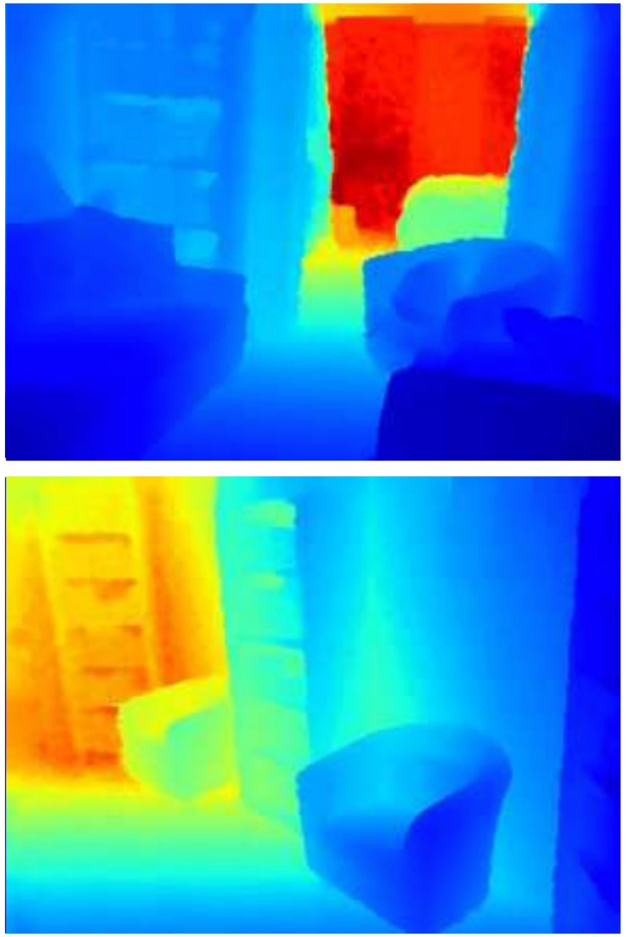
3D Shape Representations: Depth Map

For each pixel, **depth map** gives distance from the camera to the object in the world at that pixel

RGB image + Depth image
= RGB-D Image (2.5D)

This type of data can be recorded directly for some types of 3D sensors (e.g. Microsoft Kinect)

RGB Image: 3 x H x W

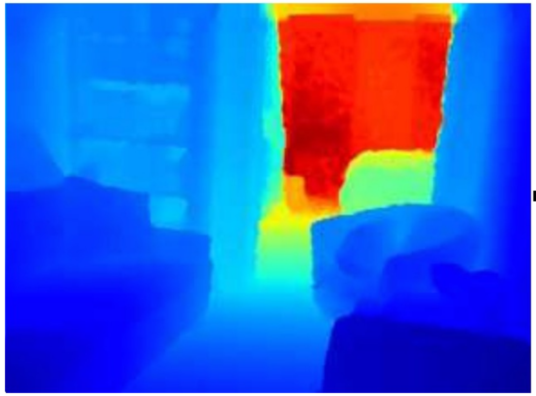


Depth Map: H x W

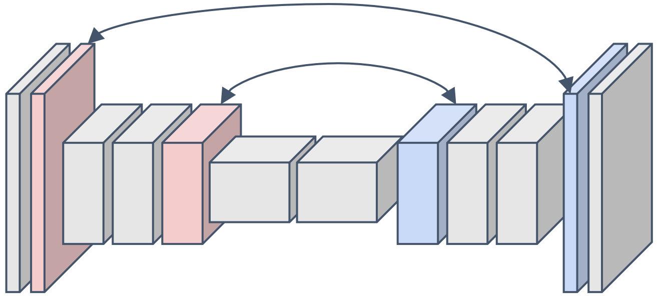
Eigen and Fergus, "Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture", ICCV 2015

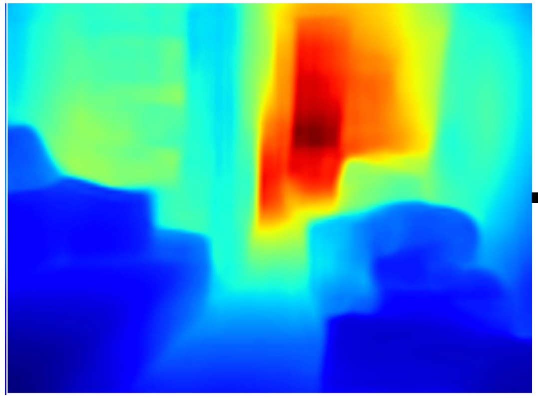
Predicting Depth Maps

Predicted Depth Image:
1 x H x W



Per-Pixel Loss
(L2 Distance)





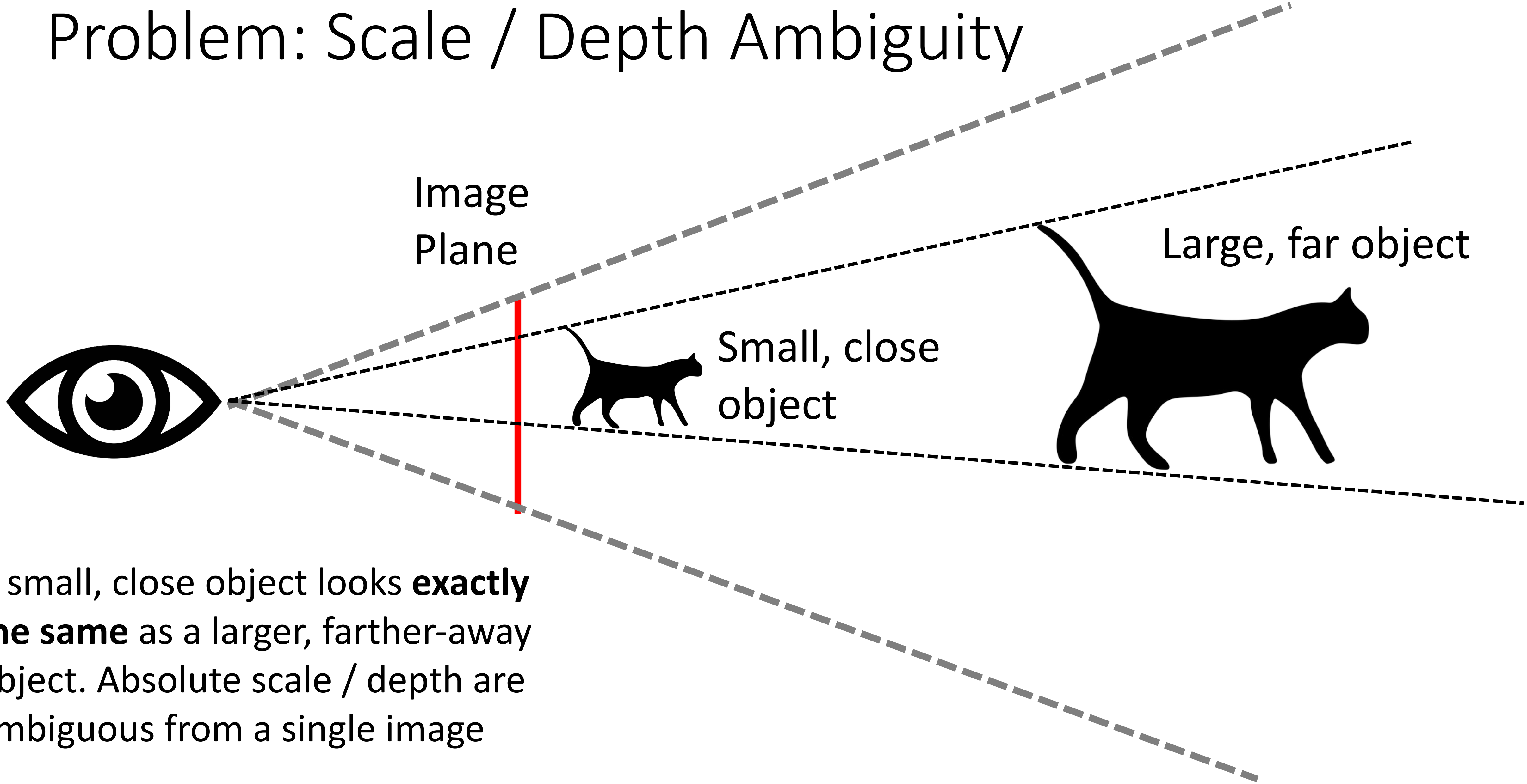
RGB Input Image:
3 x H x W

Fully Convolutional network

Predicted Depth Image:
1 x H x W

Eigen, Puhrsh, and Fergus, “Depth Map Prediction from a Single Image using a Multi-Scale Deep Network”, NeurIPS 2014
Eigen and Fergus, “Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture”, ICCV 2015

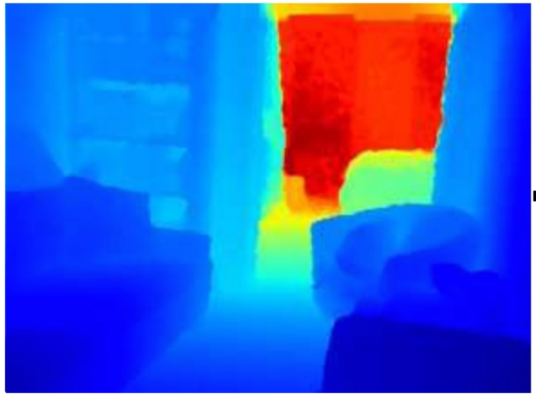
Problem: Scale / Depth Ambiguity



A small, close object looks **exactly the same** as a larger, farther-away object. Absolute scale / depth are ambiguous from a single image

Predicting Depth Maps

Predicted Depth Image:
1 x H x W

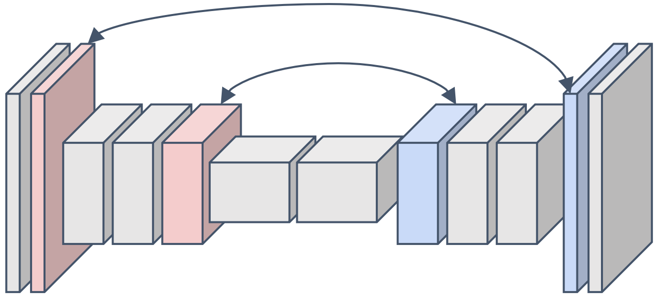


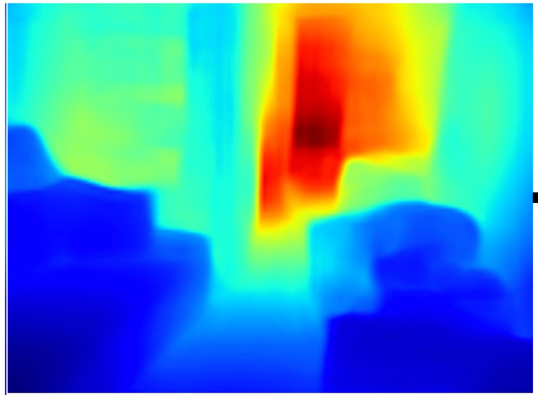
Scale invariant loss

$$D(y, y^*) = \frac{1}{2n^2}\sum_{i,j}\left((\log y_i - \log y_j) - (\log y_i^* - \log y_j^*)\right)^2$$

$$= \frac{1}{n}\sum_i d_i^2 - \frac{1}{n^2}\sum_{i,j} d_i d_j = \frac{1}{n}\sum_i d_i^2 - \frac{1}{n^2}\left(\sum_i d_i\right)^2$$

Per-Pixel Loss
(Scale invariant)





RGB Input Image:
3 x H x W

Fully Convolutional network

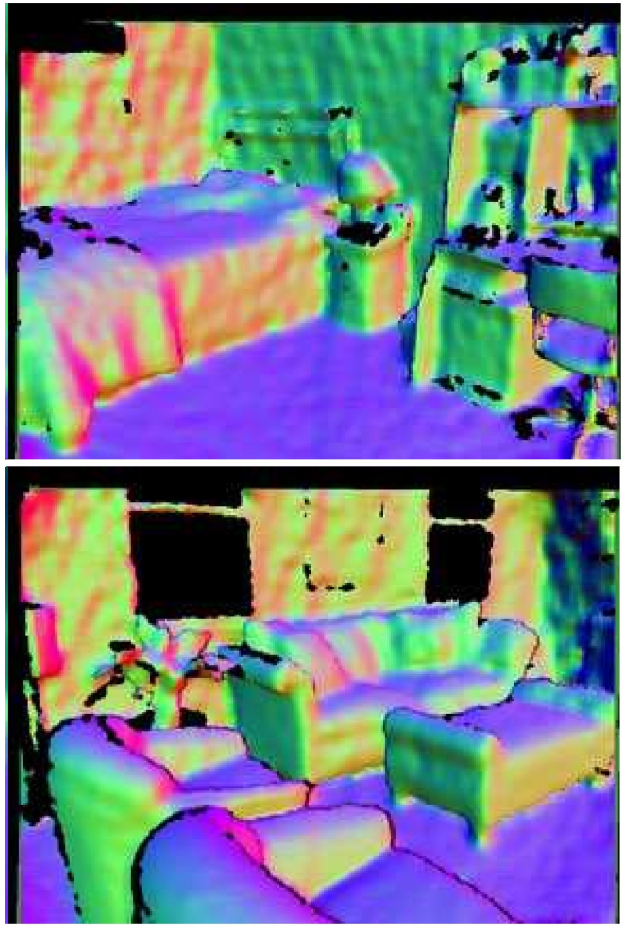
Predicted Depth Image:
1 x H x W

Eigen, Puhrsh, and Fergus, “Depth Map Prediction from a Single Image using a Multi-Scale Deep Network”, NeurIPS 2014
Eigen and Fergus, “Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture”, ICCV 2015

3D Shape Representations: Surface Normals

For each pixel, **surface normals** give a vector giving the normal vector to the object in the world for that pixel

RGB Image: 3 x H x W

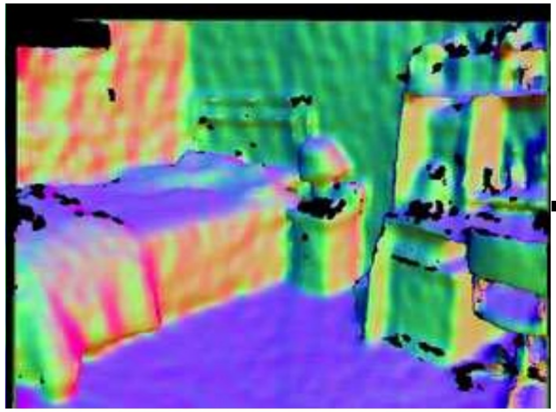


Normals: 3 x H x W

Eigen and Fergus, "Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture", ICCV 2015

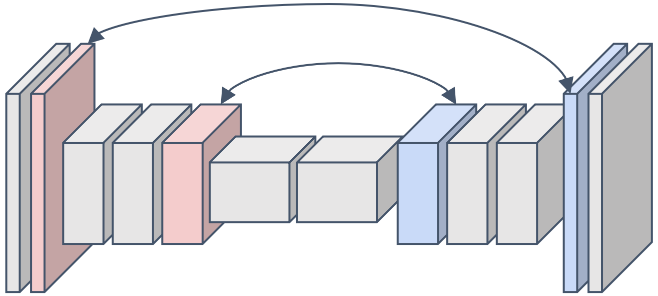
Predicting Normals

Ground-truth Normals:
3 x H x W

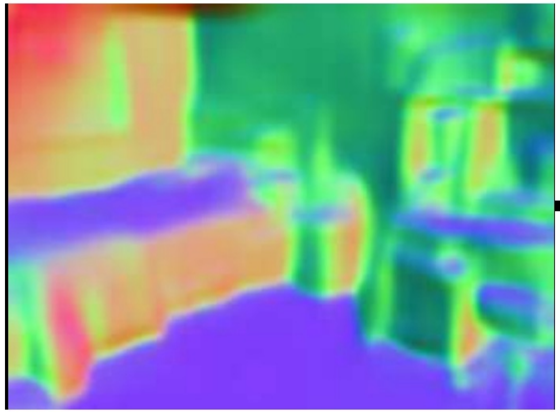


Per-Pixel Loss:
(x · y) / (|x| |y|)

RGB Input Image:
3 x H x W



Fully Convolutional network

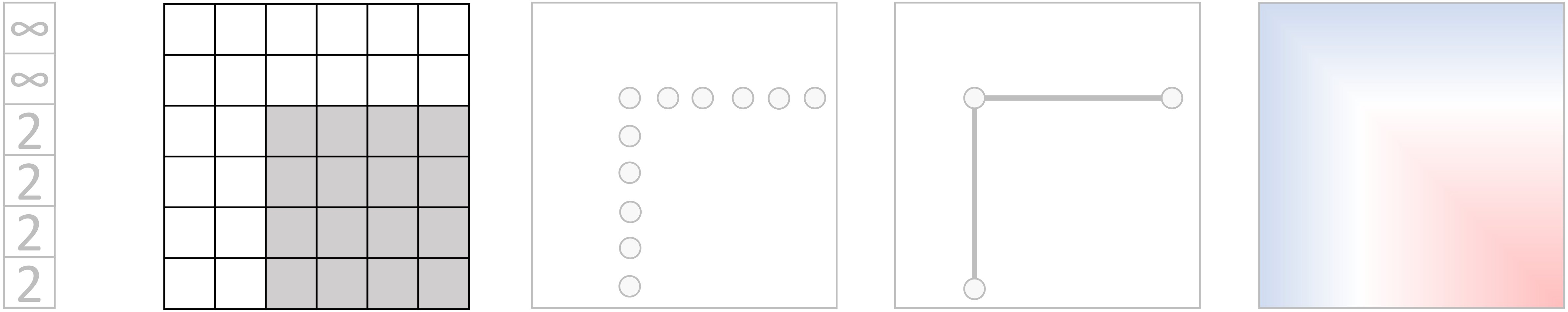


Predicted Normals:
3 x H x W

Recall:
x · y
= |x| |y| cos θ

Eigen and Fergus, “Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture”, ICCV 2015

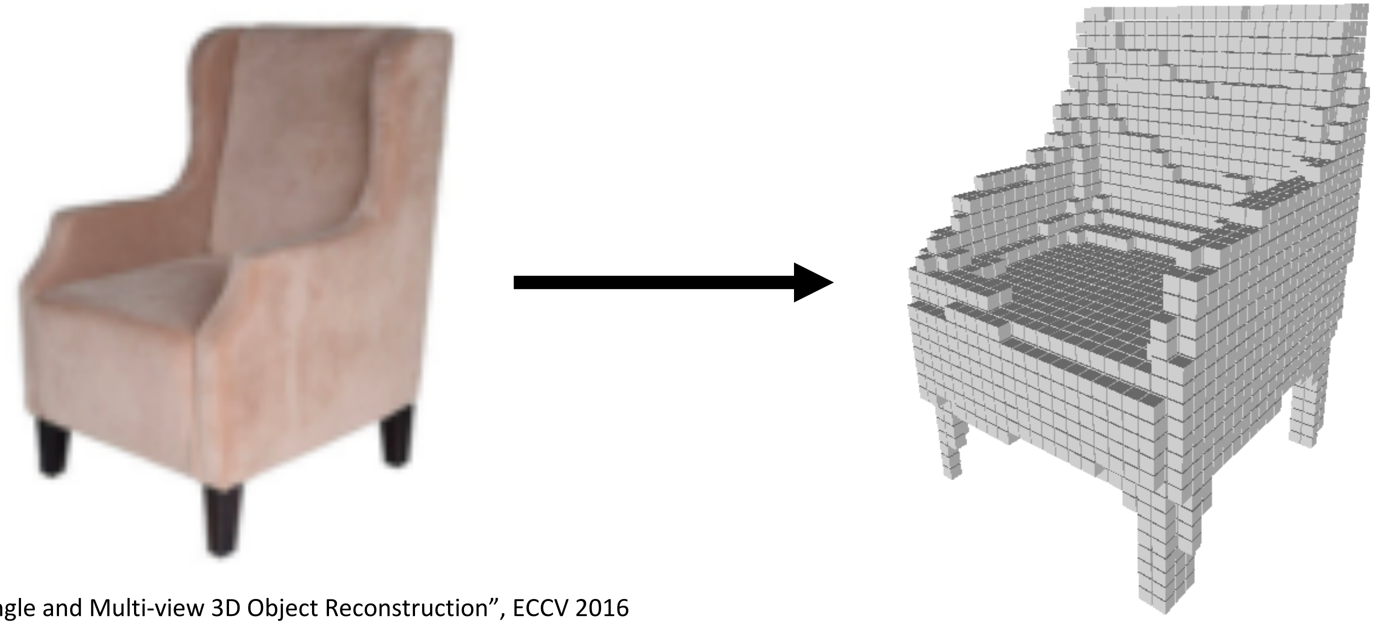
3D Shape Representations



Depth Map | Voxel Grid | Pointcloud | Mesh | Implicit Surface

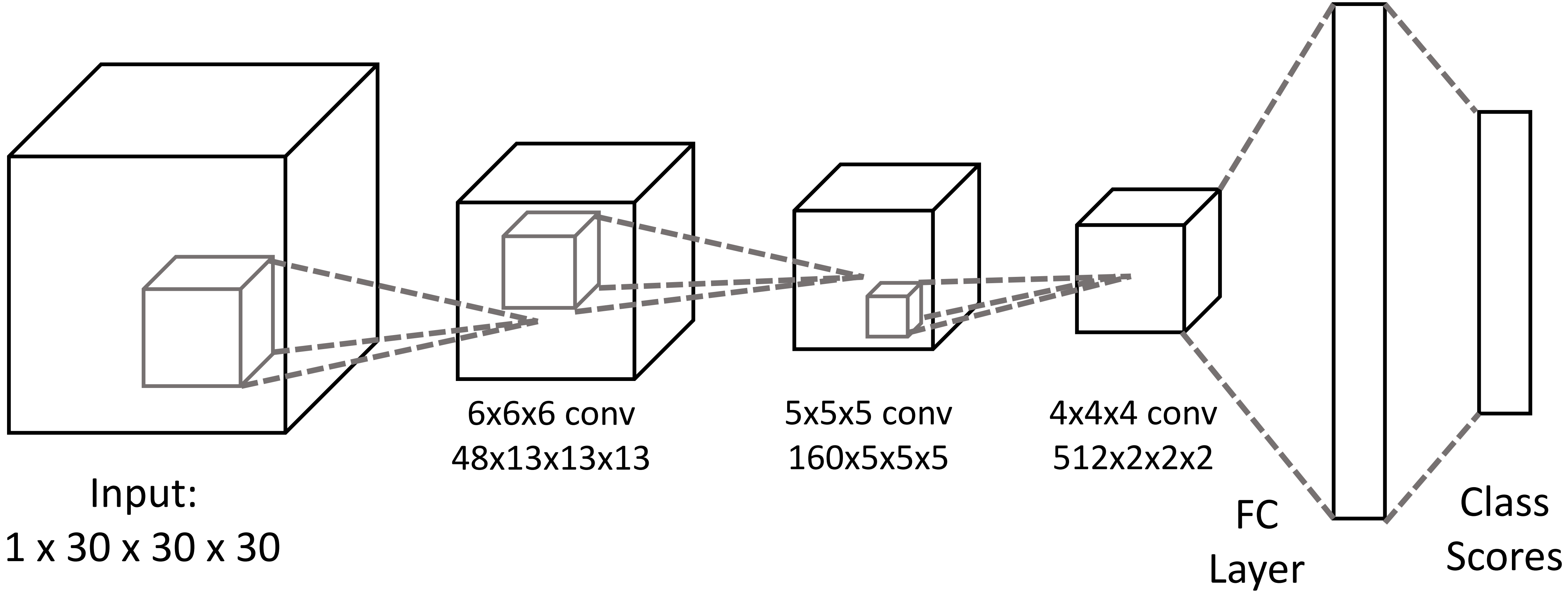
3D Shape Representations: Voxels

- Represent a shape with a V x V x V grid of occupancies
- Just like segmentation masks in Mask R-CNN, but in 3D!
- (+) Conceptually simple: just a 3D grid!
- (-) Need high spatial resolution to capture fine structures
- (-) Scaling to high resolutions is nontrivial!



Choy et al, "3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction", ECCV 2016

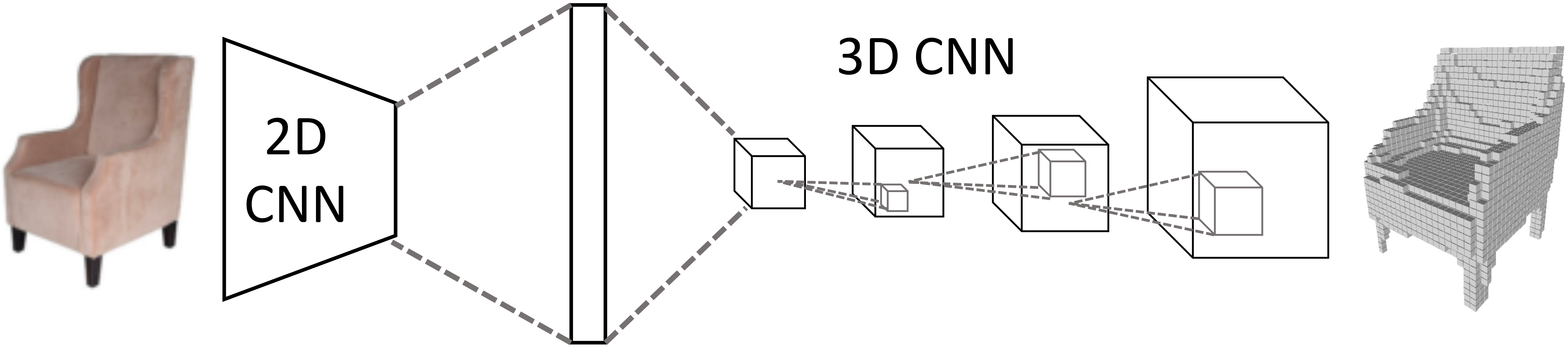
Processing Voxel Inputs: 3D Convolution



Train with classification loss

Wu et al, “3D ShapeNets: A Deep Representation for Volumetric Shapes”, CVPR 2015

Generating Voxel Shapes: 3D Convolution



Input image:
3 x 112 x 112

2D Features:
C x H x W

3D Features:
C’ x D’ x H’ x W’

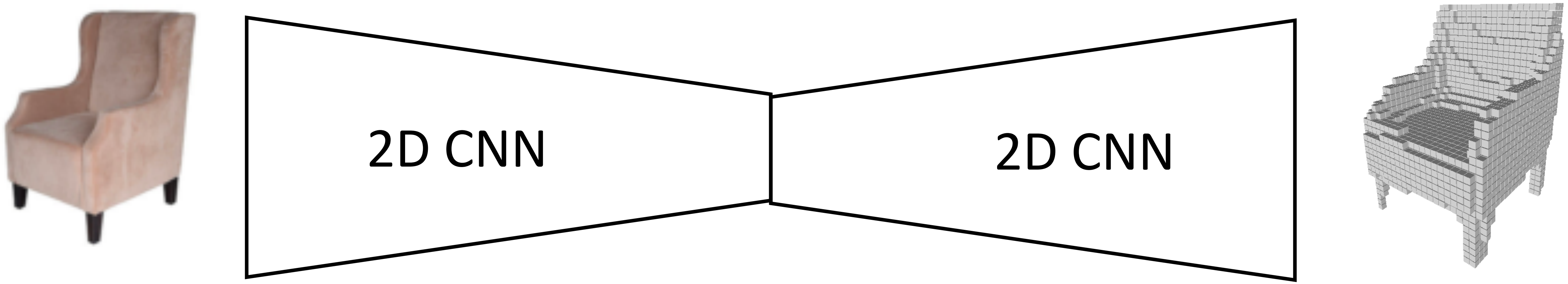
Voxels:
1 x V x V x V

Train with per-voxel cross-entropy loss

Choy et al, “3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction”, ECCV 2016

Generating Voxel Shapes: ”Voxel Tubes”

Final conv layer: V filters
Interpret as a “tube” of
voxel scores



Input image:
3 x 112 x 112

2D Features:
C x H x W

3D Features:
C’ x D’ x H’ x W’

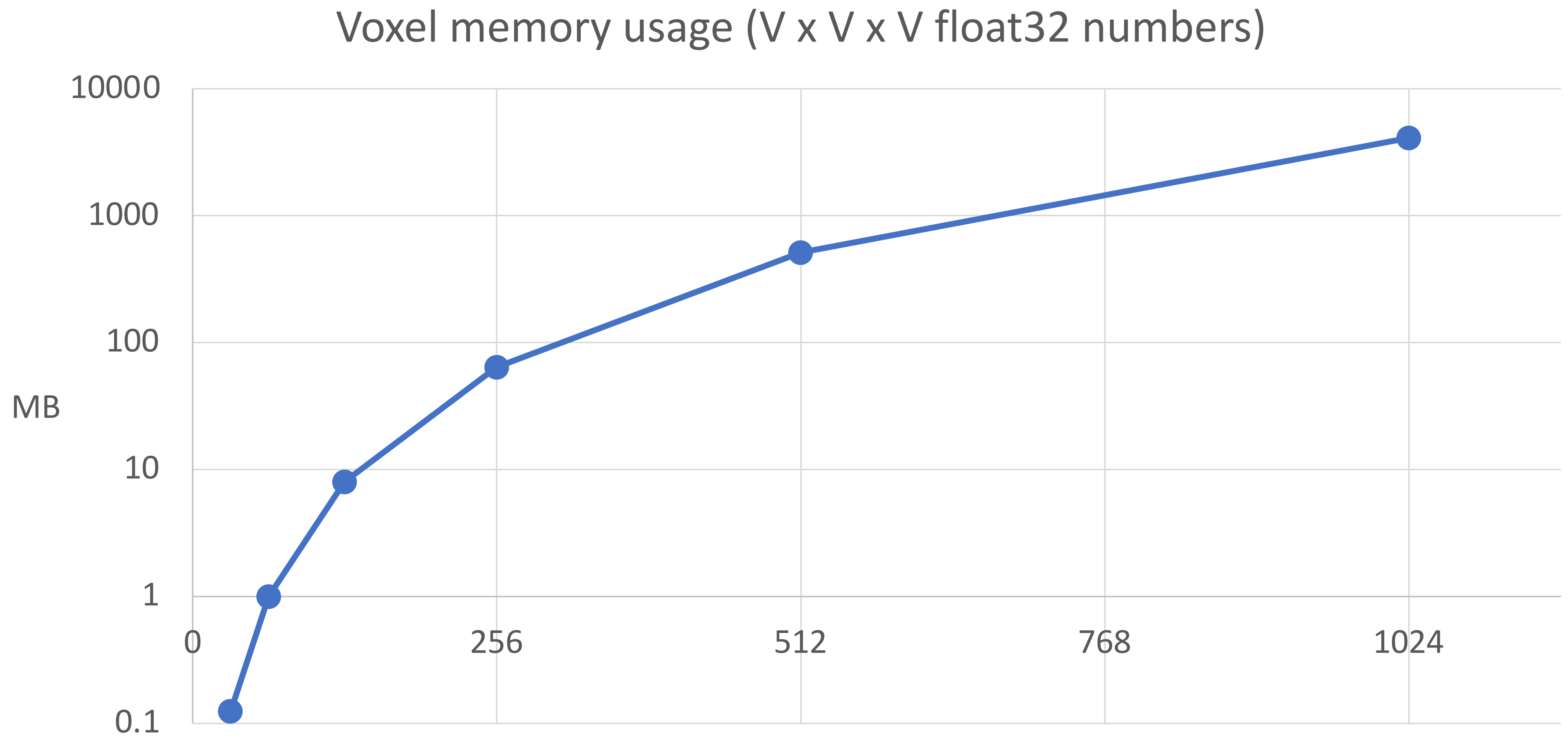
Voxels:
V x V x V

Train with per-voxel cross-entropy loss

Choy et al, “3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction”, ECCV 2016

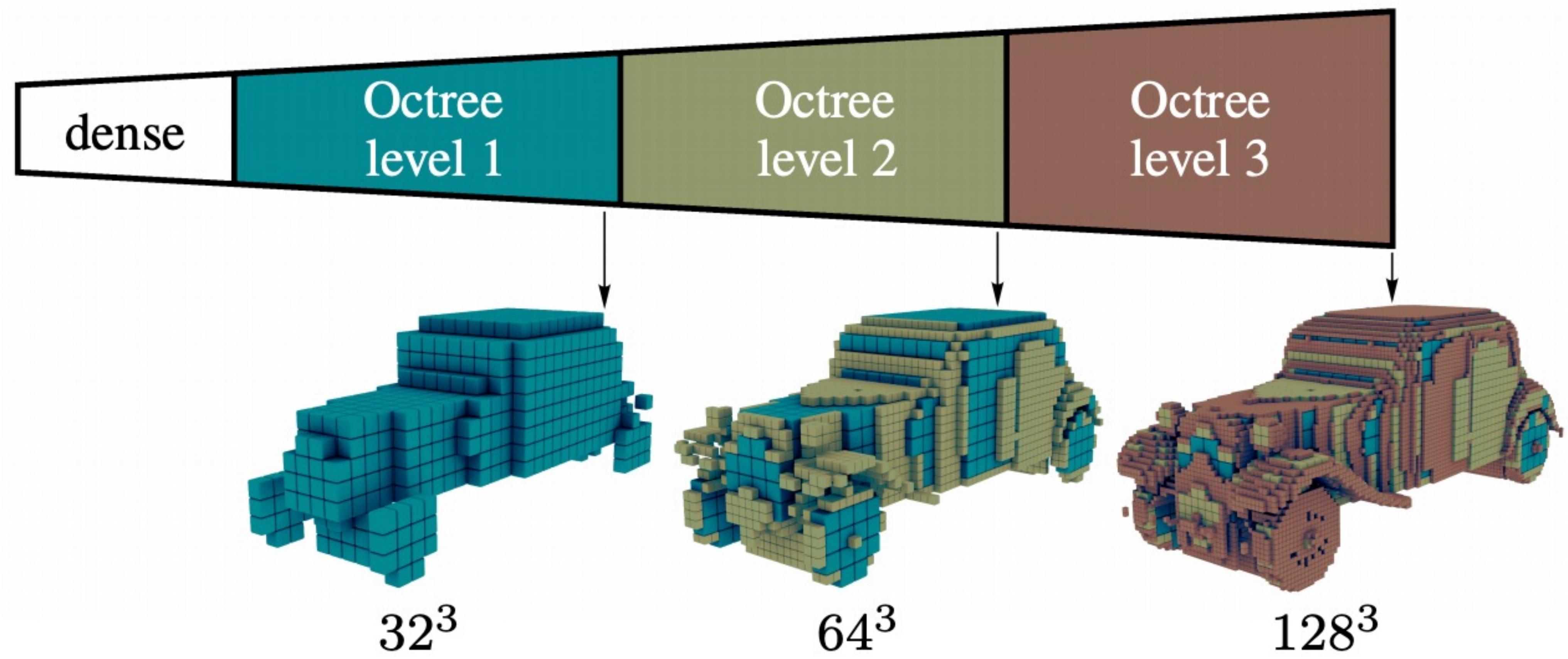
Voxel Problems: Memory Usage

Storing 1024^3 voxel grid takes 4GB of memory!



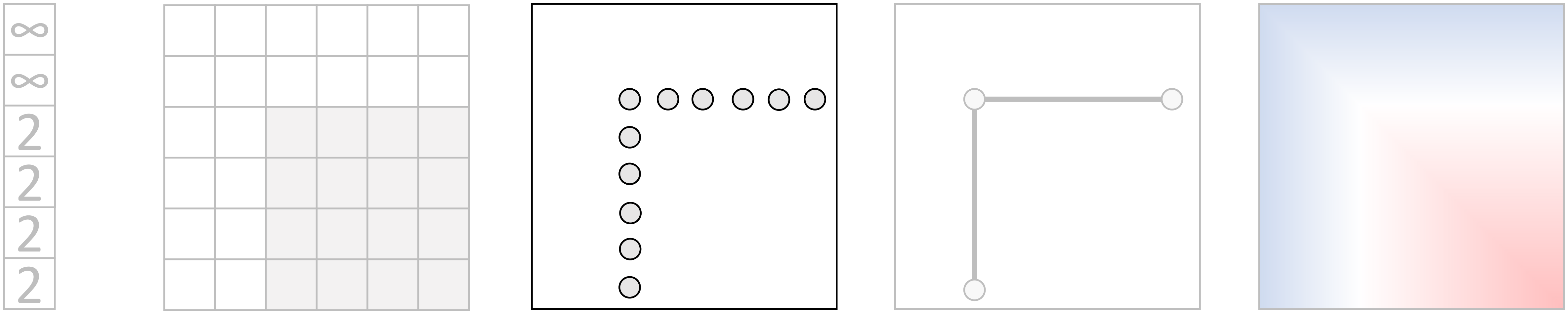
Scaling Voxels: Oct-Trees

Use voxel grids with heterogenous resolution!



Tatarchenko et al, "Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs", ICCV 2017

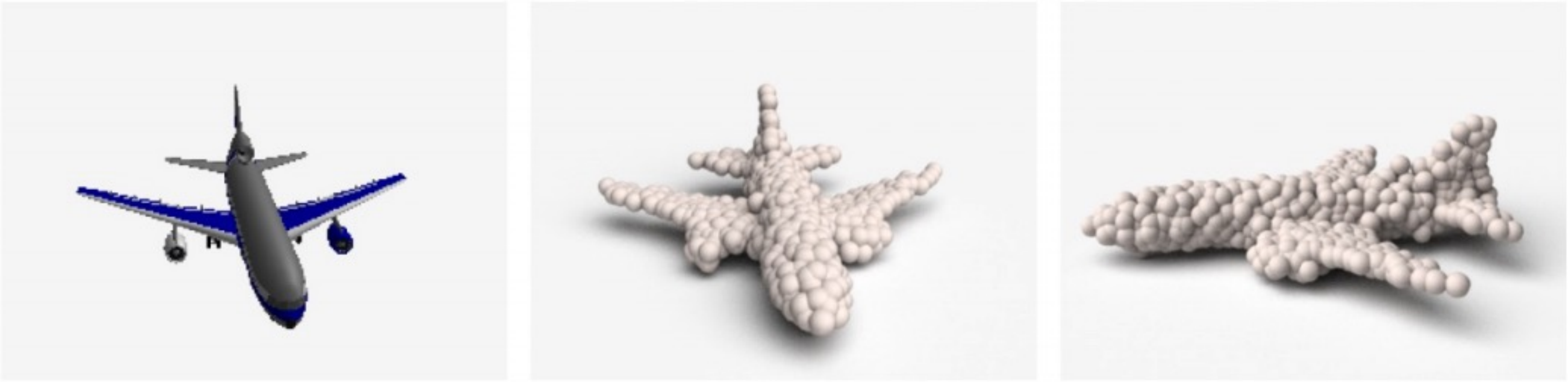
3D Shape Representations



Depth Map | Voxel Grid | **Pointcloud** | Mesh | Implicit Surface

3D Shape Representations: Point Cloud

- Represent shape as a set of P points in 3D space
- (+) Can represent fine structures without huge numbers of points
- ( ) Requires new architecture, losses, etc
- (-) Doesn't explicitly represent the surface of the shape: extracting a mesh for rendering or other applications requires post-processing



Fan et al, "A Point Set Generation Network for 3D Object Reconstruction from a Single Image", CVPR 2017

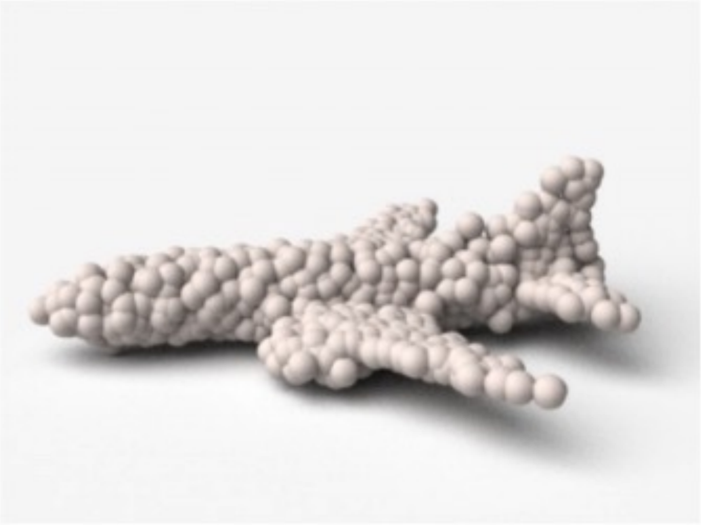
Processing Pointcloud Inputs: PointNet

Want to process pointclouds as **sets**: order should not matter

Run MLP on each point

Max-Pool

Fully Connected



Input pointcloud: P x 3

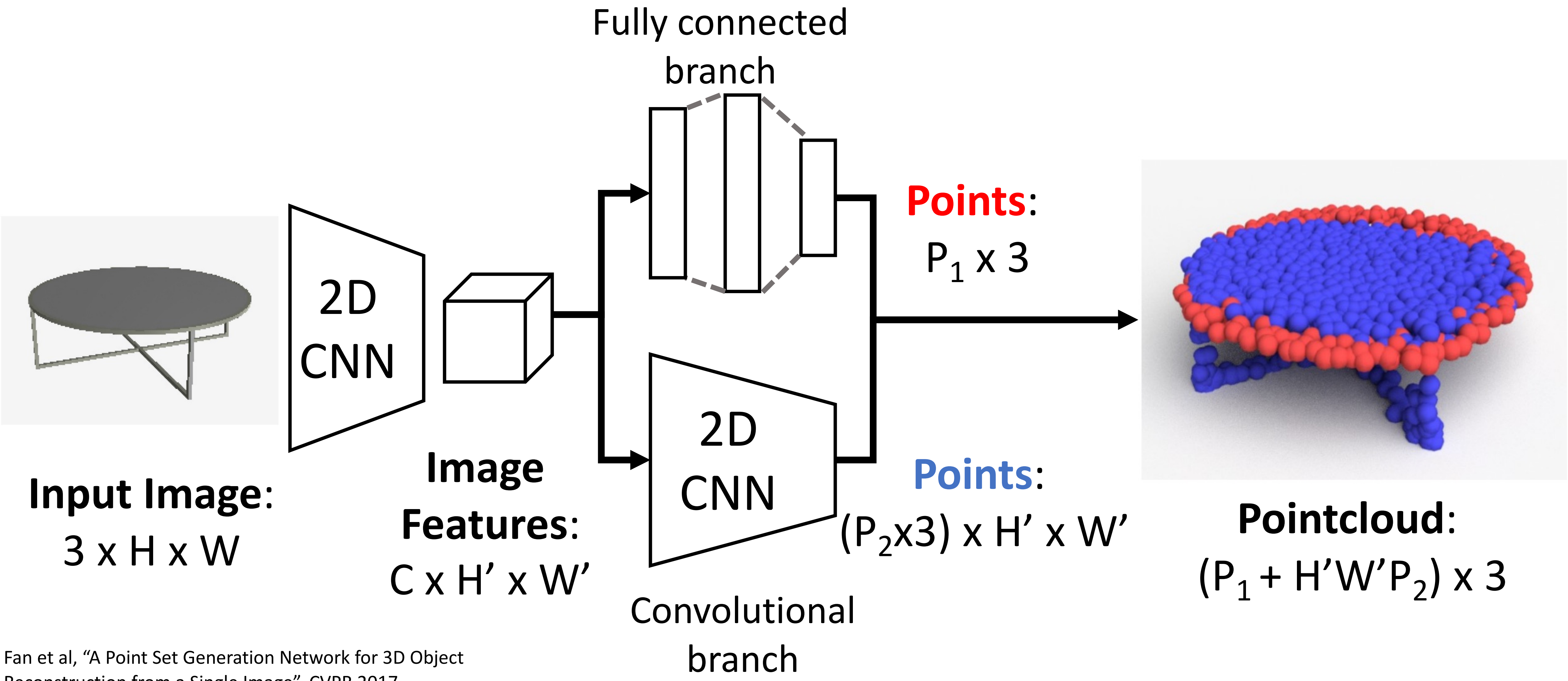
Point features: P x D

Pooled vector: D

Class score: C

Qi et al, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", CVPR 2017
Qi et al, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space", NeurIPS 2017

Generating Pointcloud Outputs



Fan et al, "A Point Set Generation Network for 3D Object Reconstruction from a Single Image", CVPR 2017

Predicting Point Clouds: Loss Function

We need a (differentiable) way to compare pointclouds **as sets**!

Fan et al, "A Point Set Generation Network for 3D Object Reconstruction from a Single Image", CVPR 2017

Predicting Point Clouds: Loss Function

We need a (differentiable) way to compare pointclouds **as sets**!

Chamfer distance is the sum of L2 distance to each point's nearest neighbor in the other set

$$d_{CD}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2^2 + \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2^2$$

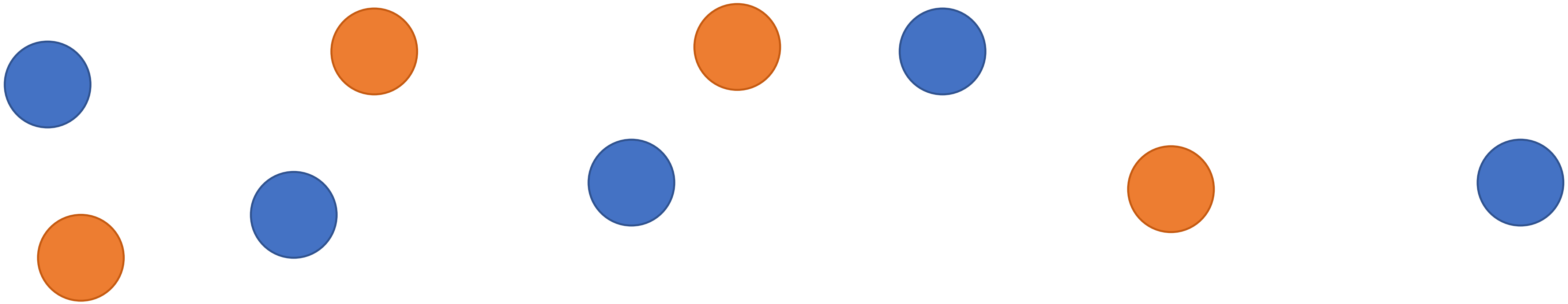
Fan et al, "A Point Set Generation Network for 3D Object Reconstruction from a Single Image", CVPR 2017

Predicting Point Clouds: Loss Function

We need a (differentiable) way to compare pointclouds **as sets**!

Chamfer distance is the sum of L2 distance to each point's nearest neighbor in the other set

$$d_{CD}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2^2 + \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2^2$$



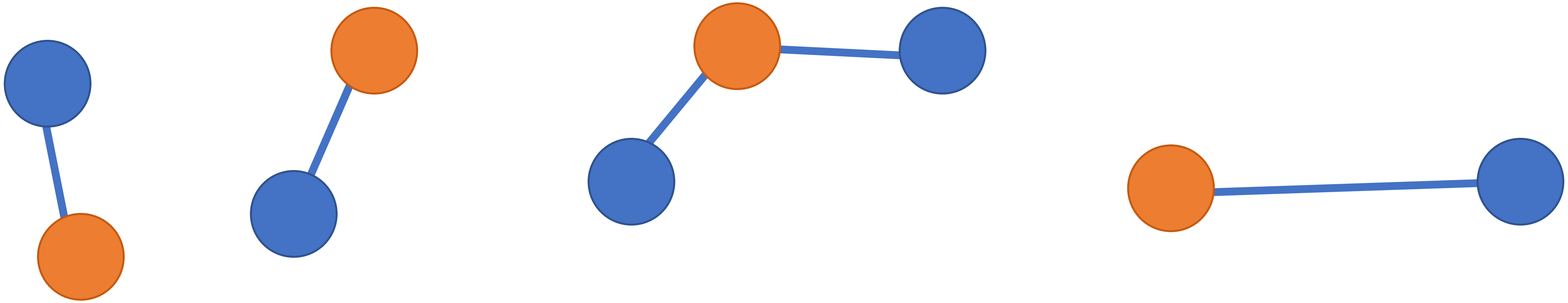
Fan et al, "A Point Set Generation Network for 3D Object Reconstruction from a Single Image", CVPR 2017

Predicting Point Clouds: Loss Function

We need a (differentiable) way to compare pointclouds **as sets**!

Chamfer distance is the sum of L2 distance to each point's nearest neighbor in the other set

$$d_{CD}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2^2 + \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2^2$$



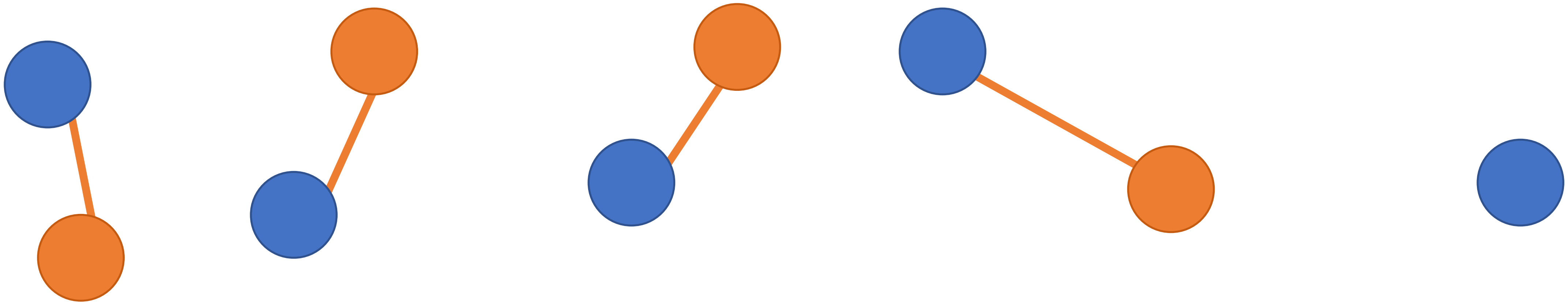
Fan et al, "A Point Set Generation Network for 3D Object Reconstruction from a Single Image", CVPR 2017

Predicting Point Clouds: Loss Function

We need a (differentiable) way to compare pointclouds **as sets**!

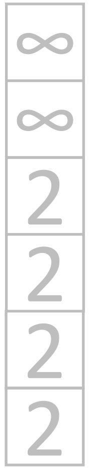
Chamfer distance is the sum of L2 distance to each point's nearest neighbor in the other set

$$d_{CD}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2^2 + \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2^2$$

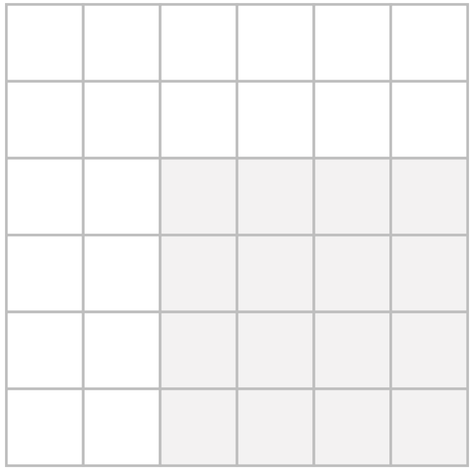


Fan et al, "A Point Set Generation Network for 3D Object Reconstruction from a Single Image", CVPR 2017

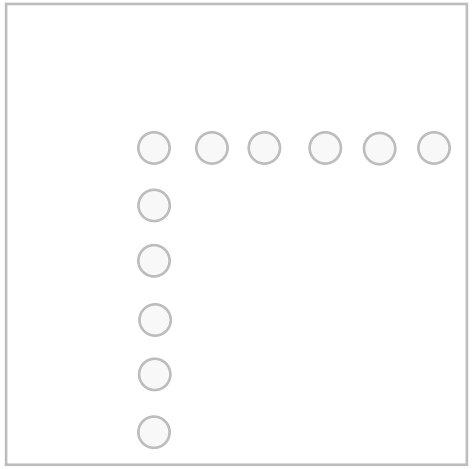
3D Shape Representations



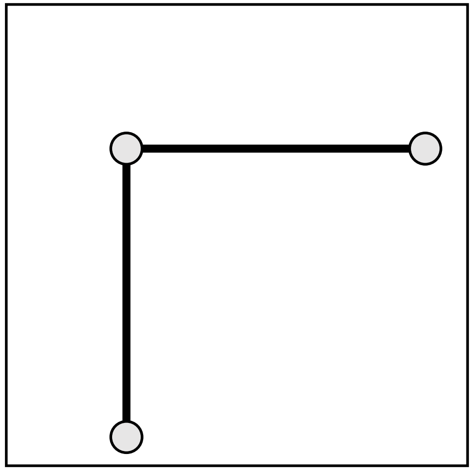
Depth Map



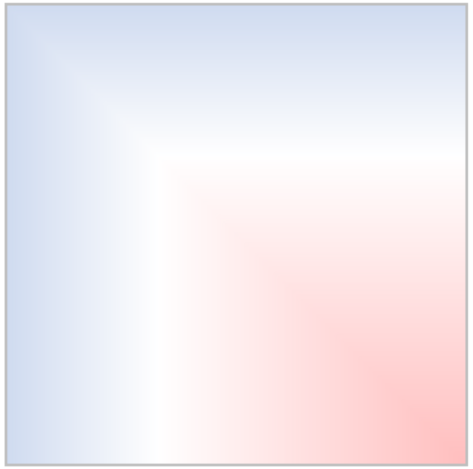
Voxel Grid



Pointcloud



Mesh



Implicit Surface

3D Shape Representations: Triangle Mesh

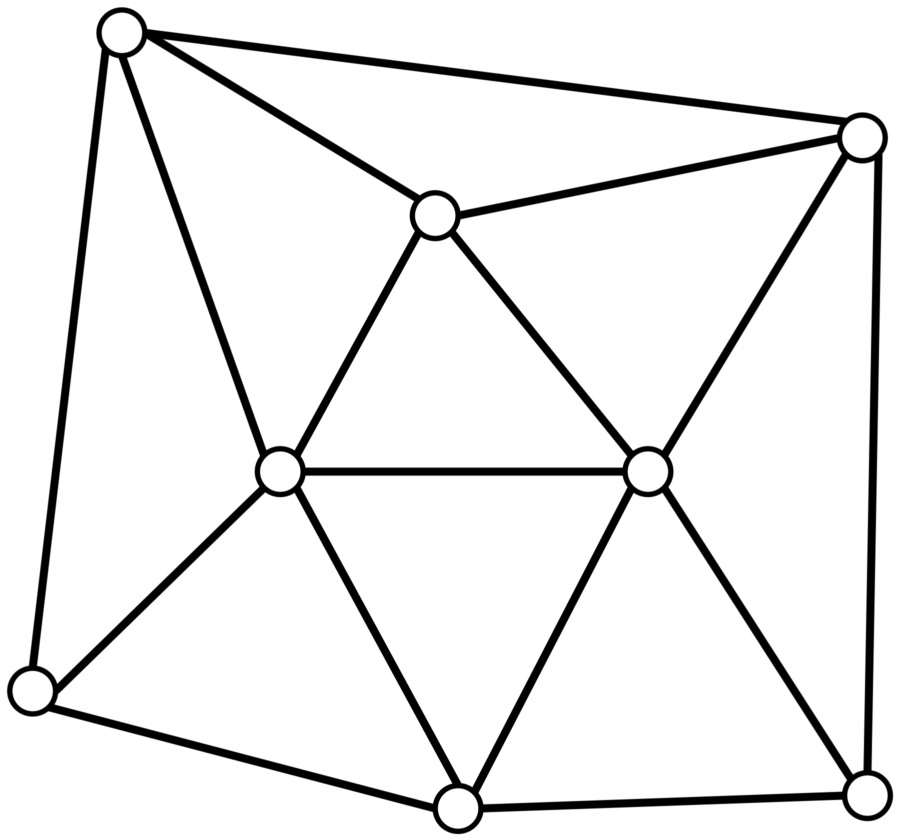
Represent a 3D shape as a set of triangles

Vertices: Set of V points in 3D space

Faces: Set of triangles over the vertices

(+) Standard representation for graphics

(+) Explicitly represents 3D shapes



3D Shape Representations: Triangle Mesh

Represent a 3D shape as a set of triangles

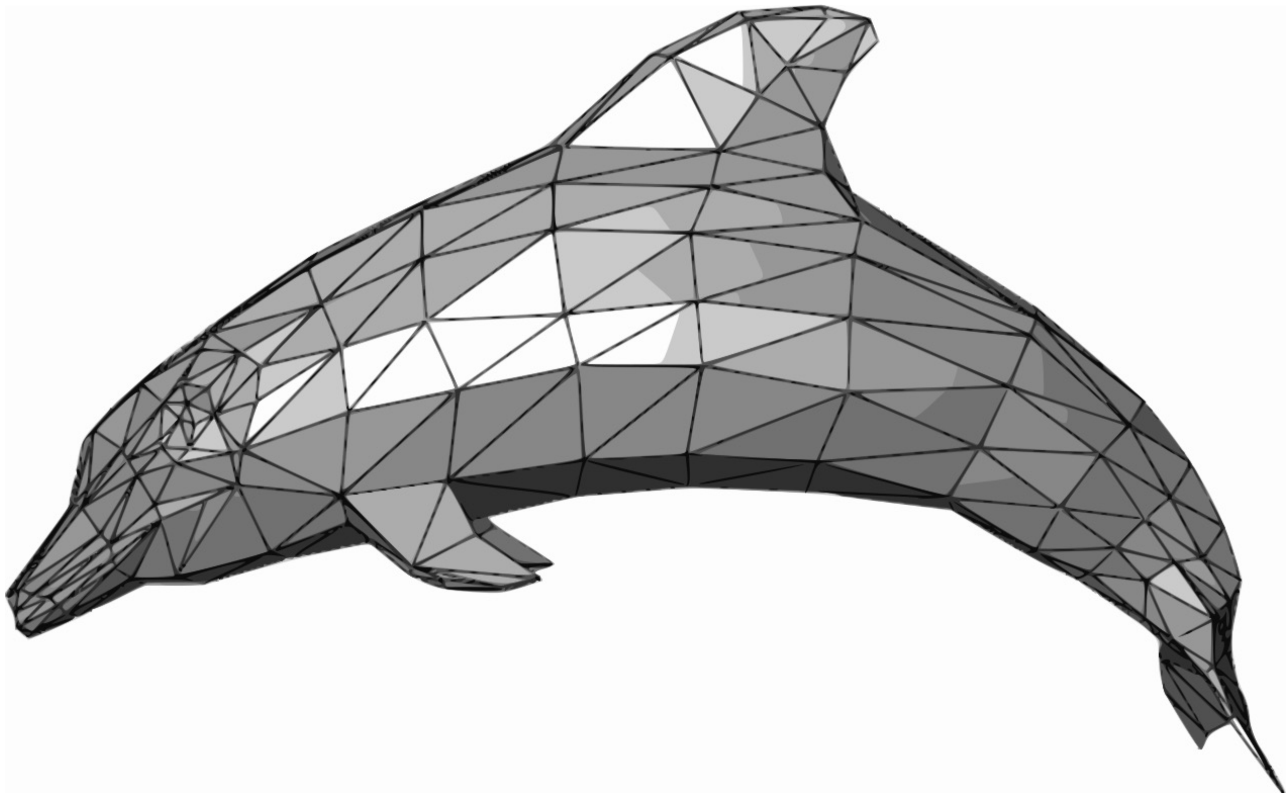
Vertices: Set of V points in 3D space

Faces: Set of triangles over the vertices

(+) Standard representation for graphics

(+) Explicitly represents 3D shapes

(+) Adaptive: Can represent flat surfaces very efficiently, can allocate more faces to areas with fine detail



Dolphin image is in the public domain

3D Shape Representations: Triangle Mesh

Represent a 3D shape as a set of triangles

Vertices: Set of V points in 3D space

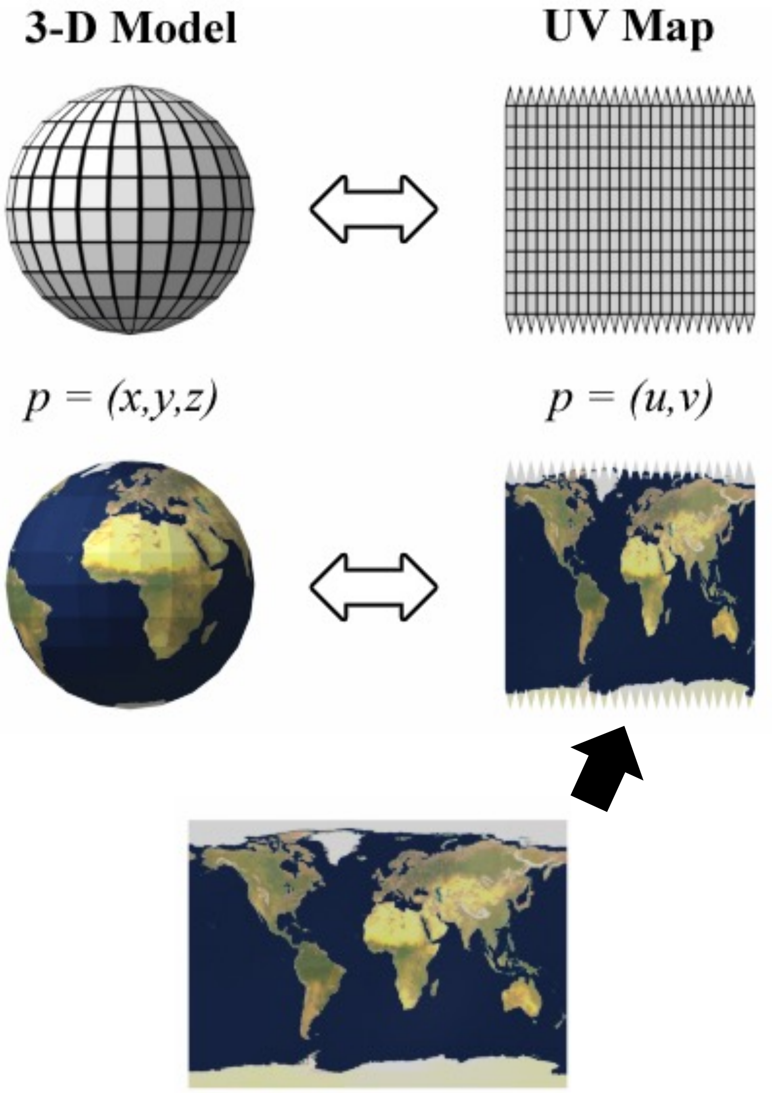
Faces: Set of triangles over the vertices

(+) Standard representation for graphics

(+) Explicitly represents 3D shapes

(+) Adaptive: Can represent flat surfaces very efficiently, can allocate more faces to areas with fine detail

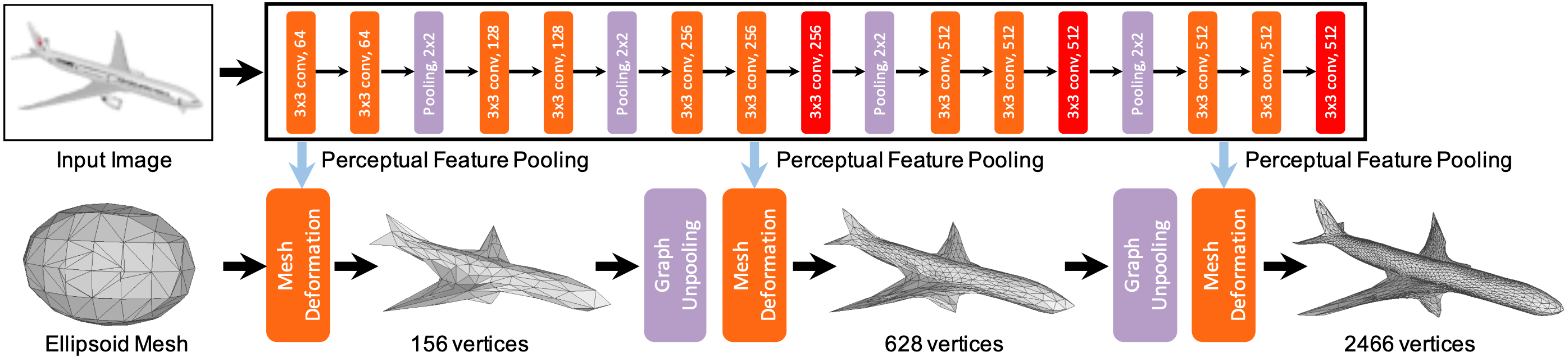
(+) Can attach data on verts and interpolate over the whole surface: RGB colors, texture coordinates, normal vectors, etc.



Predicting Meshes: Pixel2Mesh

Input: Single RGB Image of an object

Output: Triangle mesh for the object



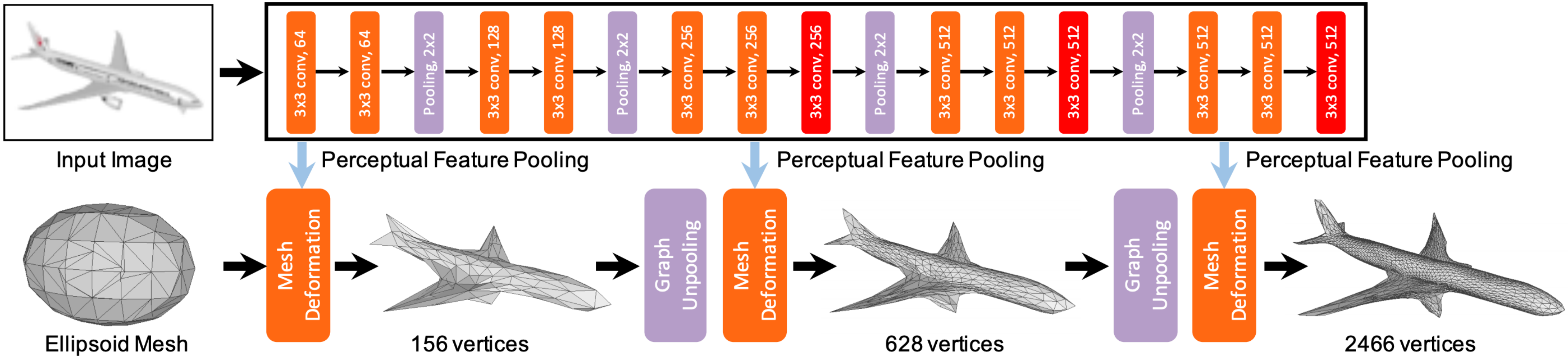
Wang et al, "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images", ECCV 2018

Predicting Meshes: Pixel2Mesh

Input: Single RGB Image of an object

Key ideas:
Iterative Refinement
Graph Convolution
Vertex Aligned-Features
Chamfer Loss Function

Output: Triangle mesh for the object



Wang et al, "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images", ECCV 2018

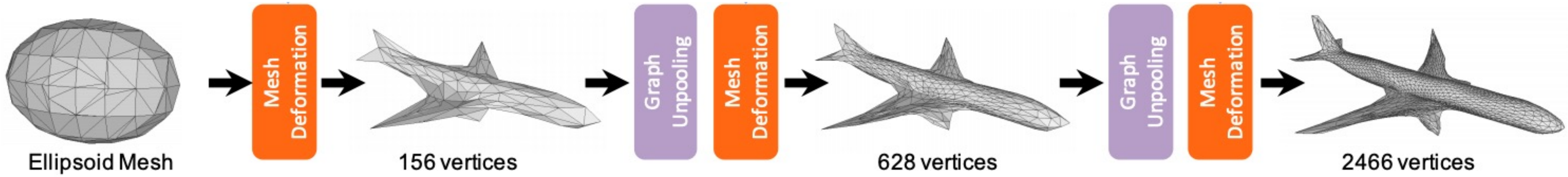
Predicting Triangle Meshes: Iterative Refinement

Idea #1: Iterative mesh refinement

Start from initial ellipsoid mesh

Network predicts offsets for each vertex

Repeat.



Wang et al, "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images", ECCV 2018

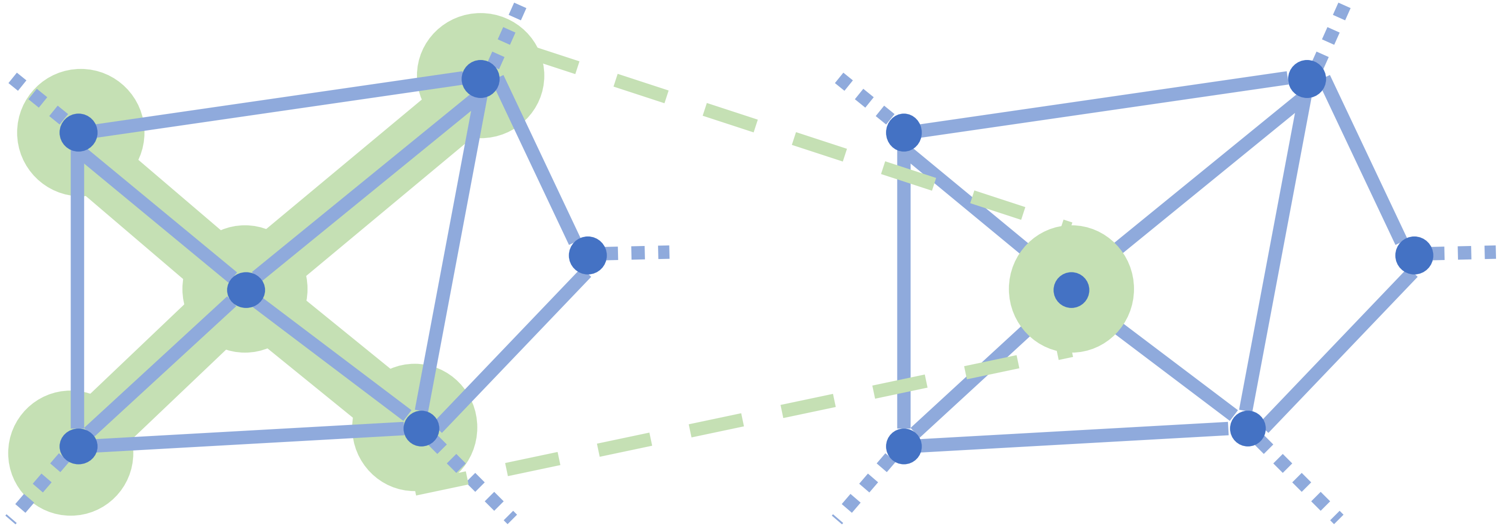
Predicting Triangle Meshes: Graph Convolution

$$f_i' = W_0 f_i + \sum_{j \in N(i)} W_1 f_j$$

Vertex v_i has feature f_i

New feature f_i' for vertex v_i depends on feature of neighboring vertices $N(i)$

Use same weights W_0 and W_1 to compute all outputs

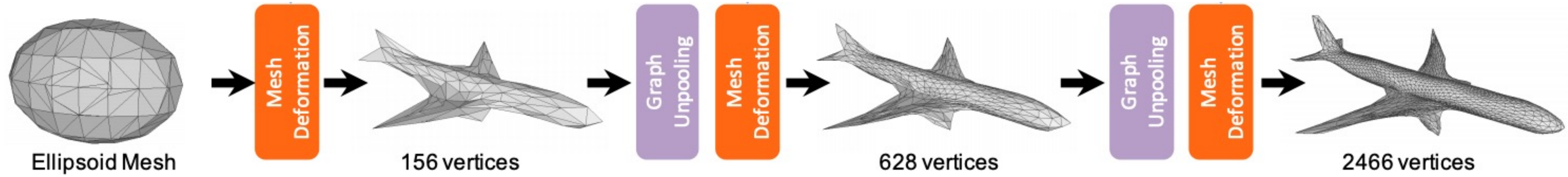


Input: Graph with a feature vector at each vertex

Output: New feature vector for each vertex

Predicting Triangle Meshes: Graph Convolution

Each of these blocks consists of a stack of **graph convolution layers** operating on edges of the mesh

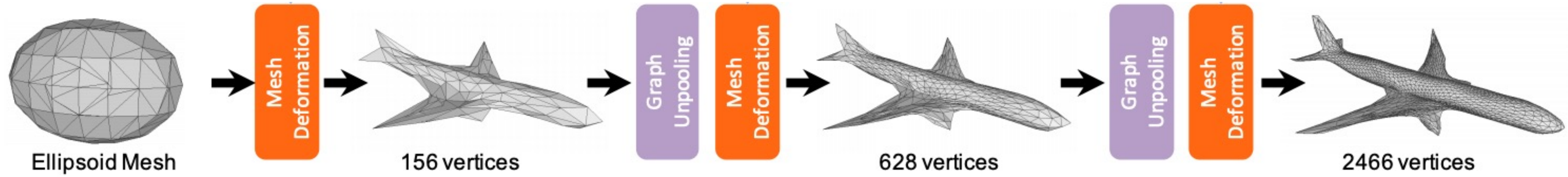


Wang et al, "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images", ECCV 2018

Predicting Triangle Meshes: Graph Convolution

Each of these blocks consists of a stack of **graph convolution layers** operating on edges of the mesh

Problem: How to incorporate image features?



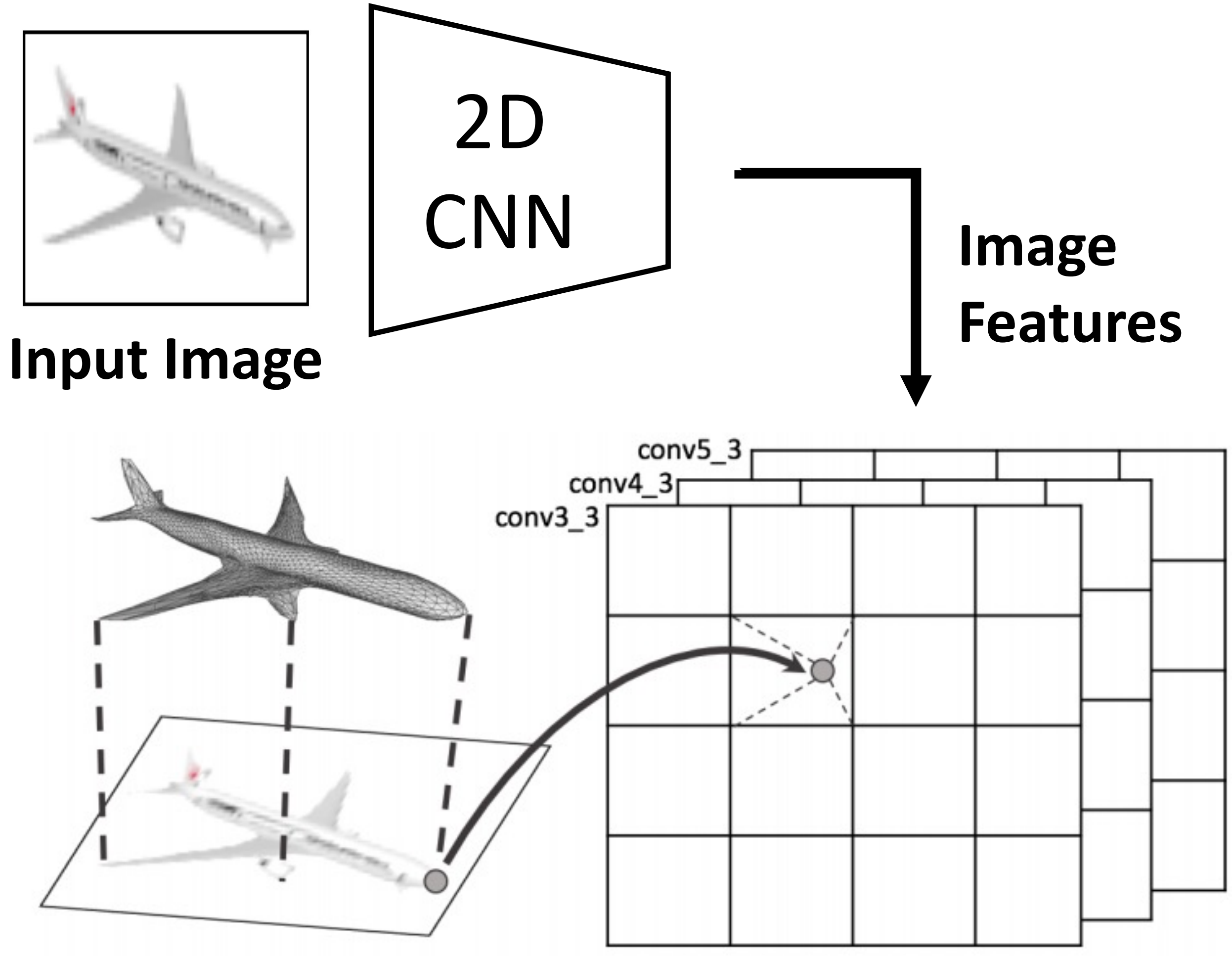
Wang et al, "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images", ECCV 2018

Predicting Triangle Meshes: Vertex-Aligned Features

Idea #2: Aligned vertex features

For each vertex of the mesh:

- Use camera information to project onto image plane
- Use bilinear interpolation to sample a CNN feature



Wang et al, "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images", ECCV 2018

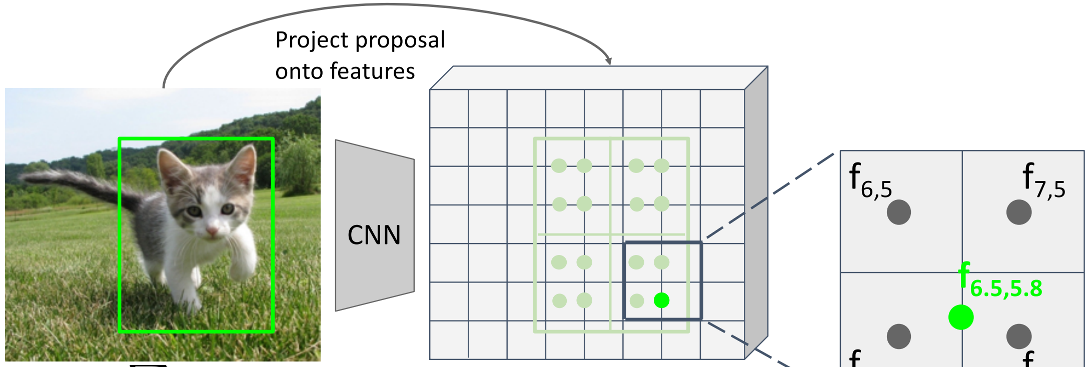
Predicting Triangle Meshes: Vertex-Aligned Features

Idea #2: Aligned vertex features

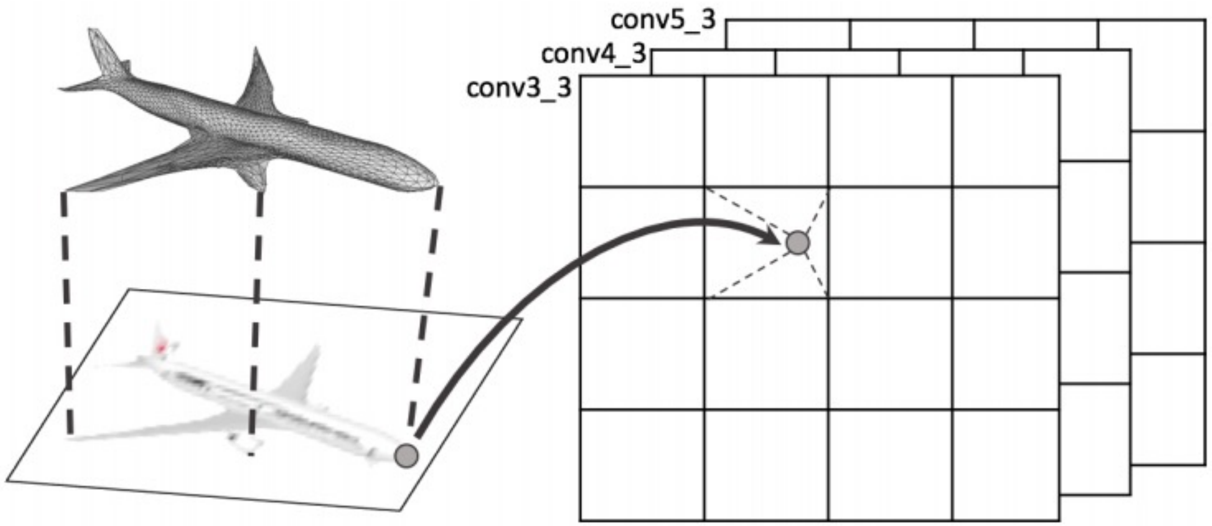
For each vertex of the mesh:

- Use camera information to project onto image plane
- Use bilinear interpolation to sample a CNN feature

Similar to RoI-Align operation from detection: maintains alignment between input image and feature vectors



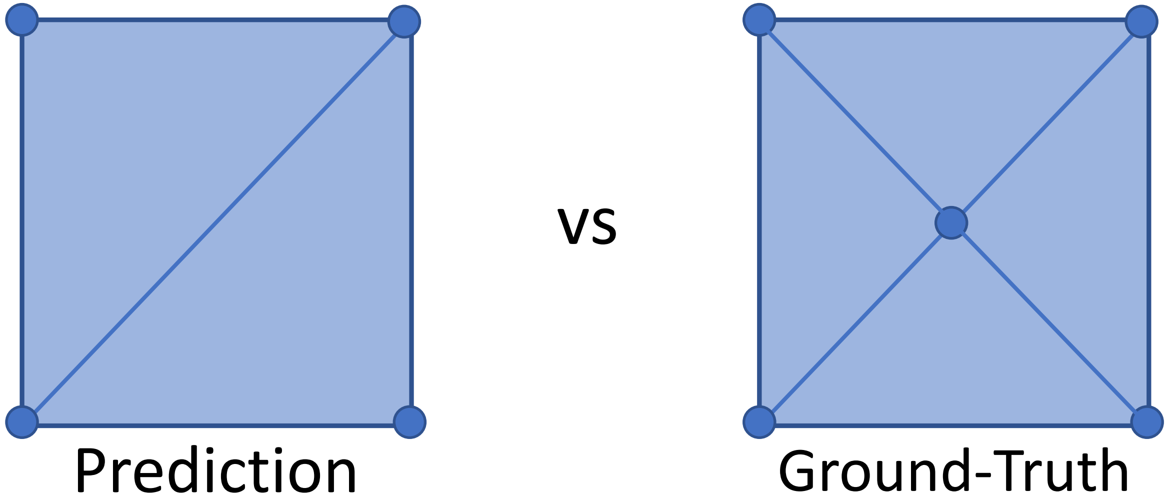
$$f_{x,y} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - i|) \max(0, 1 - |y - j|)$$



Wang et al, "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images", ECCV 2018

Predicting Meshes: Loss Function

The same shape can be represented with different meshes – how can we define a loss between predicted and ground-truth mesh?

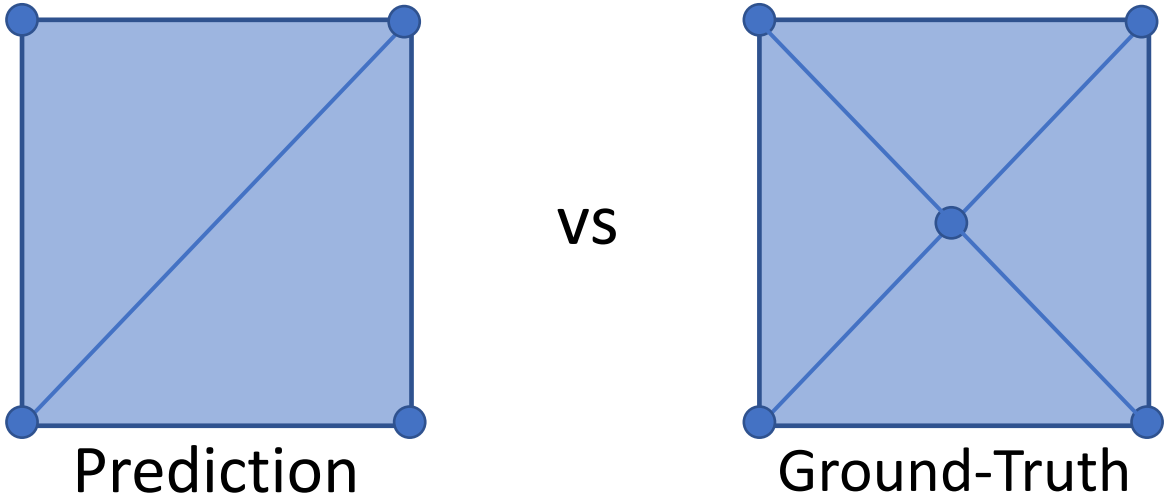


Wang et al, "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images", ECCV 2018

Predicting Meshes: Loss Function

The same shape can be represented with different meshes – how can we define a loss between predicted and ground-truth mesh?

Idea: Convert meshes to pointclouds, then compute loss

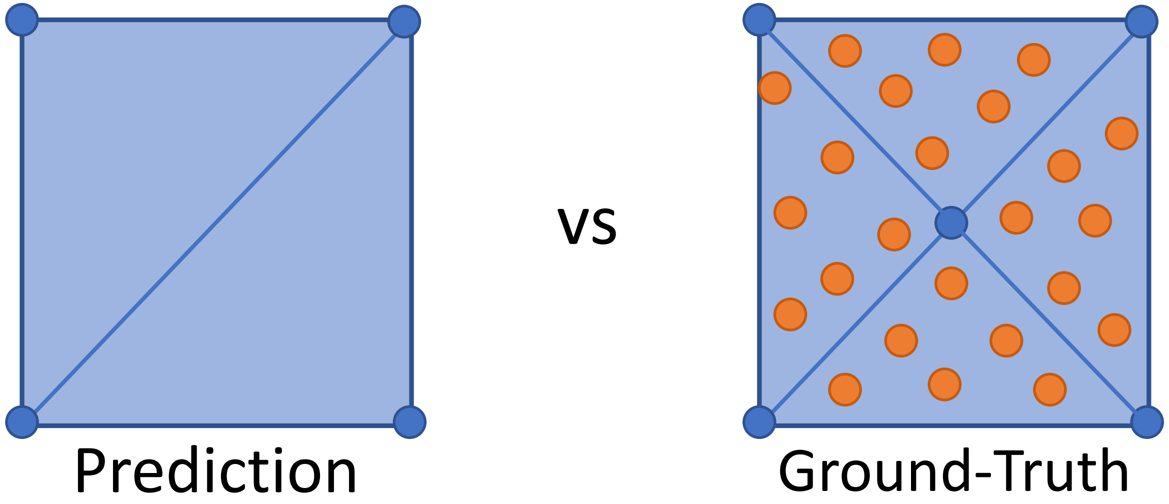


Wang et al, "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images", ECCV 2018

Predicting Meshes: Loss Function

The same shape can be represented with different meshes – how can we define a loss between predicted and ground-truth mesh?

Idea: Convert meshes to pointclouds, then compute loss



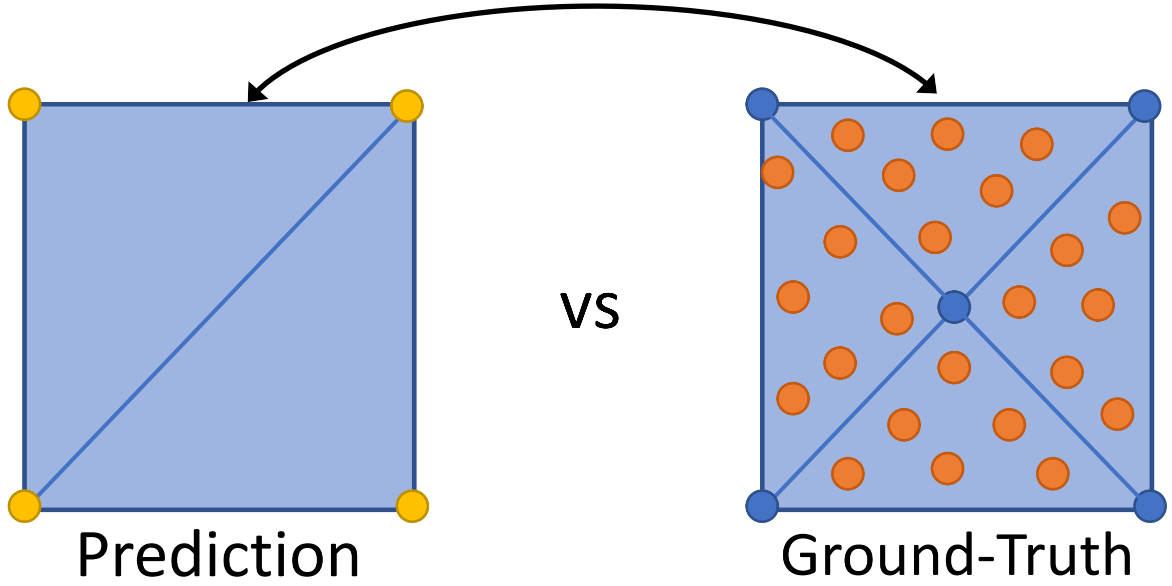
Sample points from the surface of the ground-truth mesh (offline)

Wang et al, “Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images”, ECCV 2018

Predicting Meshes: Loss Function

The same shape can be represented with different meshes – how can we define a loss between predicted and ground-truth mesh?

Loss = Chamfer distance between **predicted verts** and **ground-truth samples**



Sample points from the surface of the ground-truth mesh (offline)

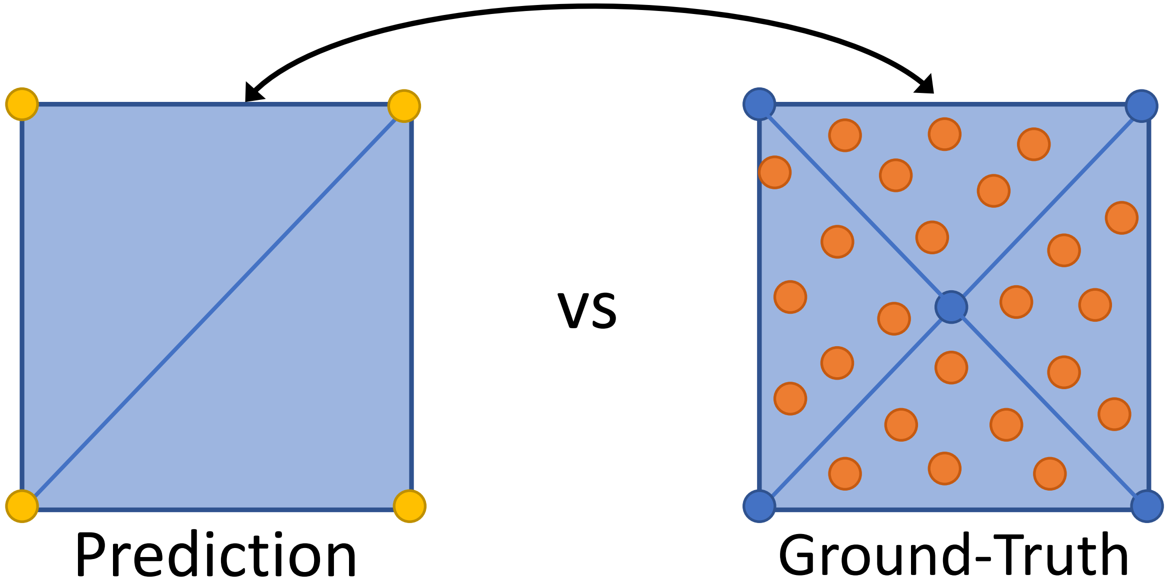
Wang et al, "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images", ECCV 2018

Predicting Meshes: Loss Function

The same shape can be represented with different meshes – how can we define a loss between predicted and ground-truth mesh?

Loss = Chamfer distance between **predicted verts** and **ground-truth samples**

Problem: Doesn't take the interior of predicted faces into account!



Sample points from the surface of the ground-truth mesh (offline)

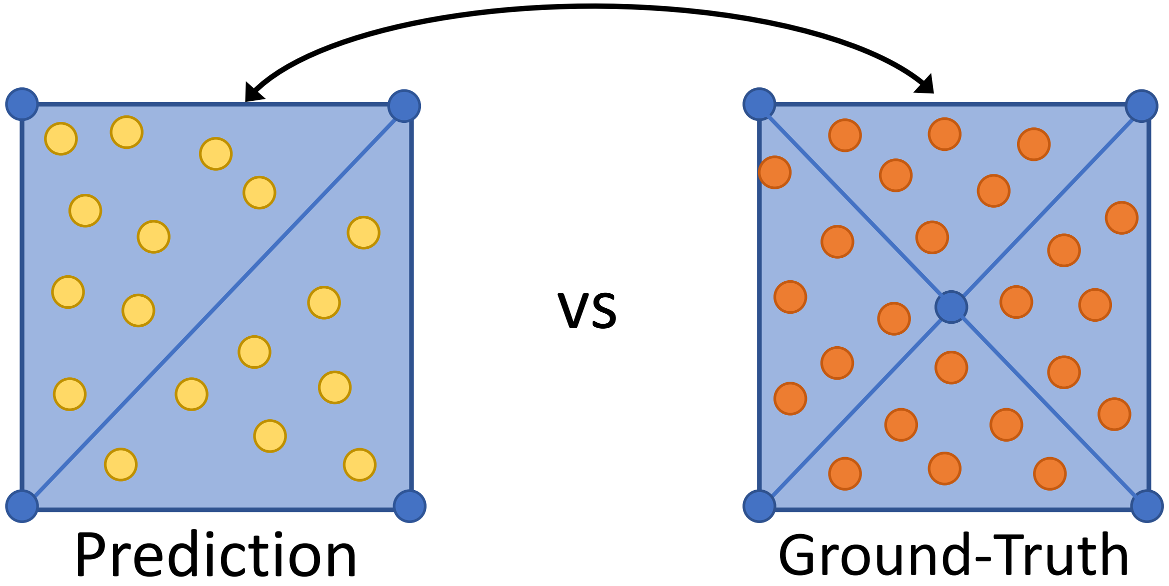
Wang et al, "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images", ECCV 2018

Predicting Meshes: Loss Function

The same shape can be represented with different meshes – how can we define a loss between predicted and ground-truth mesh?

Loss = Chamfer distance between **predicted samples** and **ground-truth samples**

Sample points from the surface of the predicted mesh (online!)



Sample points from the surface of the ground-truth mesh (offline)

Smith et al, "GEOMetrics: Exploiting Geometric Structure for Graph-Encoded Objects", ICML 2019

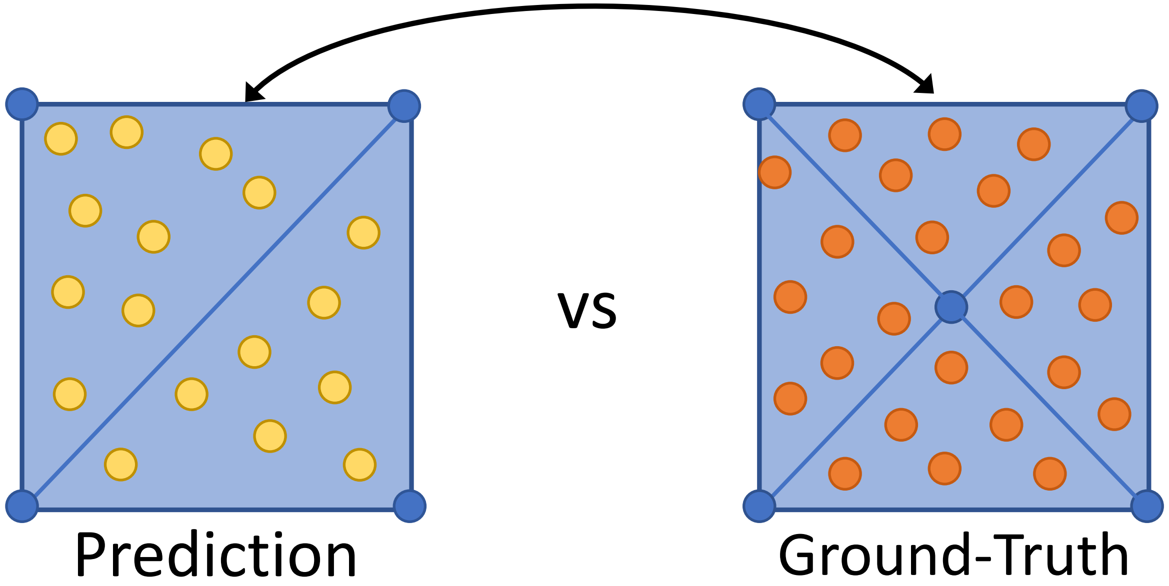
Predicting Meshes: Loss Function

Problem: Need to sample online! Must be efficient!

Problem: Need to backprop through sampling!

Loss = Chamfer distance between **predicted samples** and **ground-truth samples**

Sample points from the surface of the predicted mesh (online!)



Sample points from the surface of the ground-truth mesh (offline)

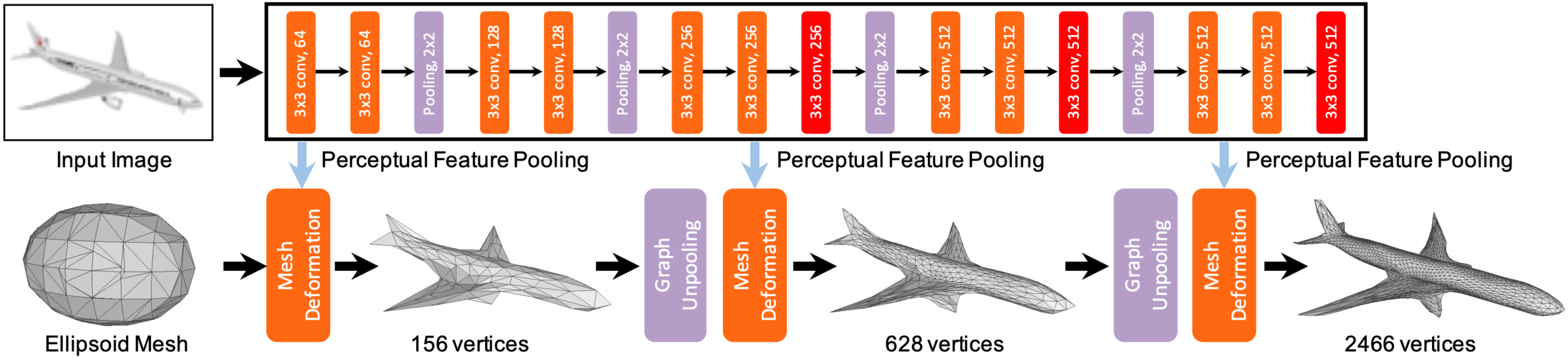
Smith et al, "GEOMetrics: Exploiting Geometric Structure for Graph-Encoded Objects", ICML 2019

Predicting Meshes: Pixel2Mesh

Input: Single RGB Image of an object

Key ideas:
Iterative Refinement
Graph Convolution
Vertex Aligned-Features
Chamfer Loss Function

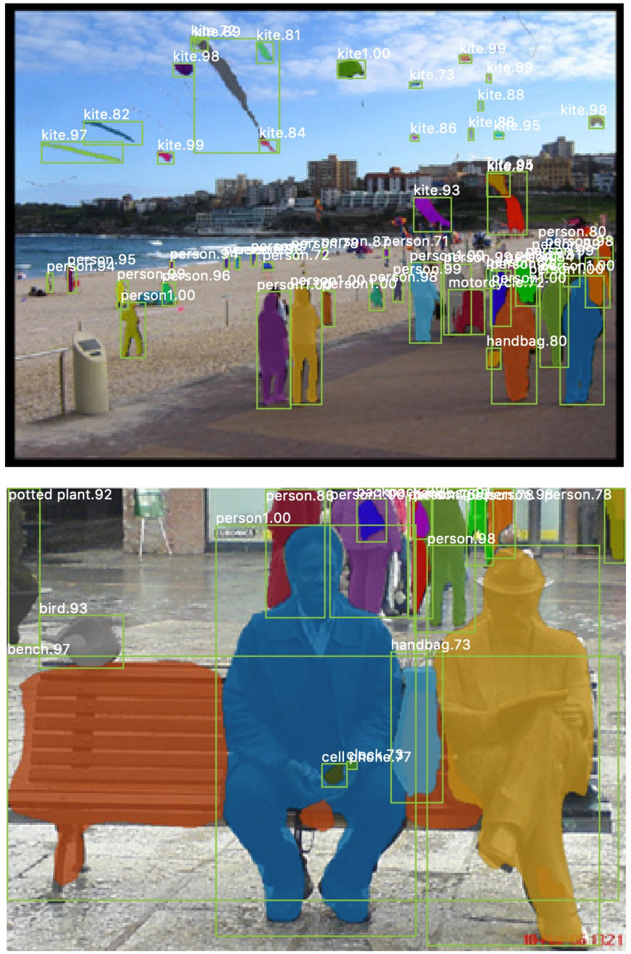
Output: Triangle mesh for the object



Wang et al, "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images", ECCV 2018

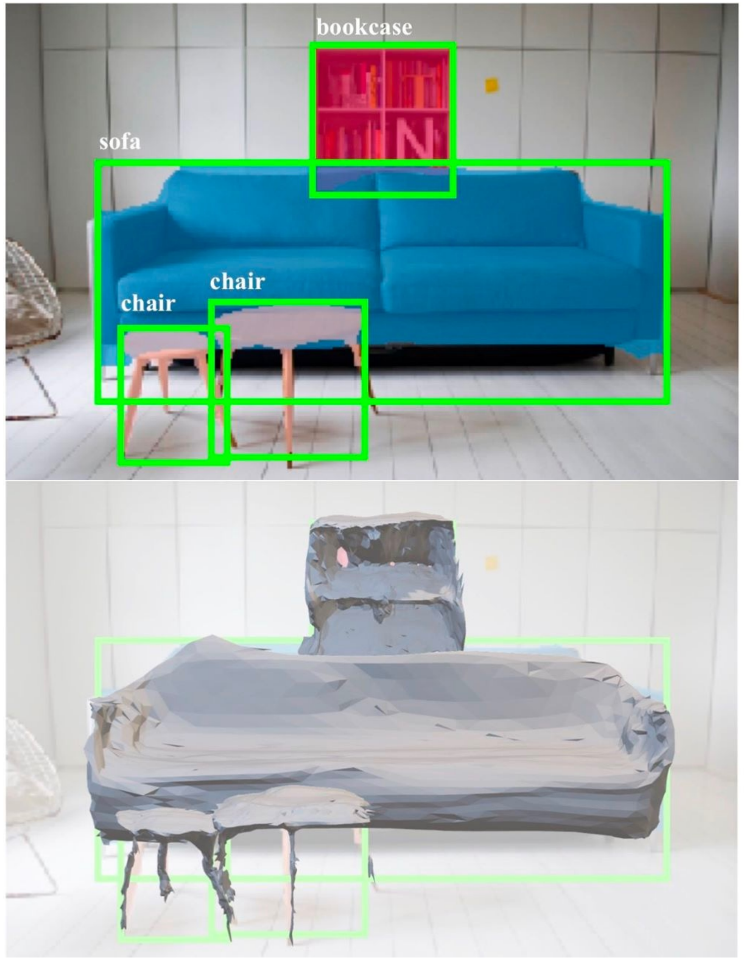
3D Shape Prediction: Mesh R-CNN

Mask R-CNN:
2D Image -> 2D shapes



He, Gkioxari, Dollár, and Girshick, “Mask R-CNN”, ICCV 2017

Mesh R-CNN:
2D Image -> **Triangle Meshes**



Gkioxari, Malik, and Johnson, “Mesh R-CNN”, ICCV 2019

Mesh R-CNN: Task

Input: Single RGB image

Output:

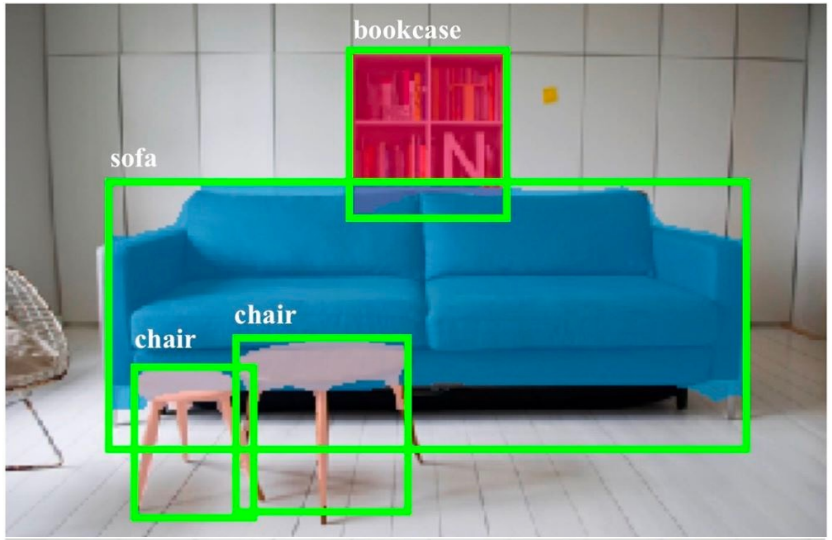
A set of detected objects

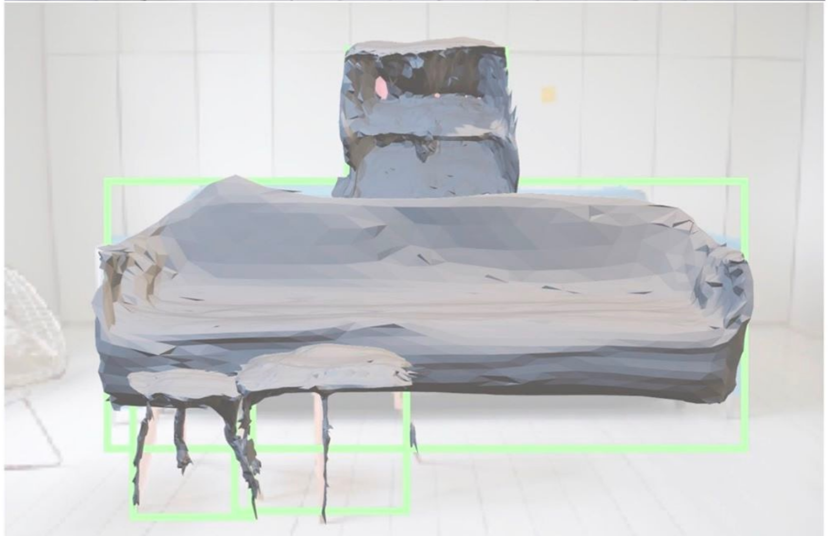
For each object:

- Bounding box
- Category label
- Instance segmentation
- 3D triangle mesh

Mask R-CNN

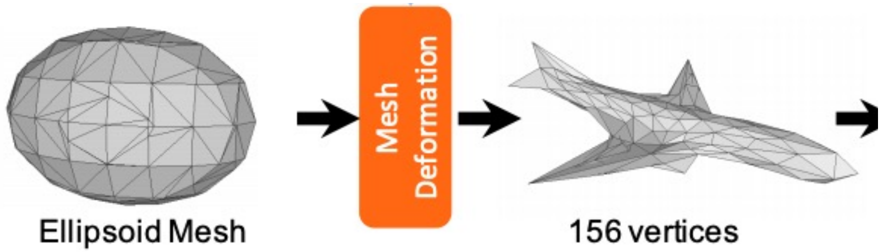
Mesh head





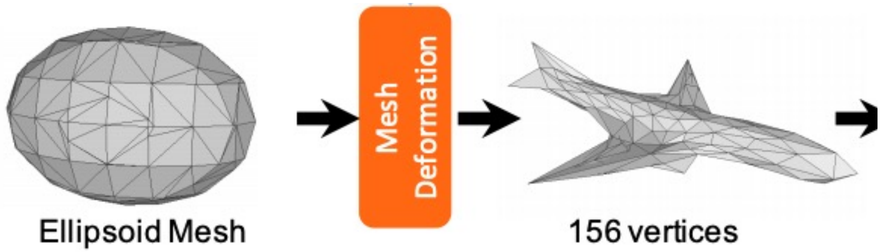
Mesh R-CNN: Hybrid 3D shape representation

Mesh deformation gives good results, but the topology (verts, faces, genus, connected components) fixed by the initial mesh

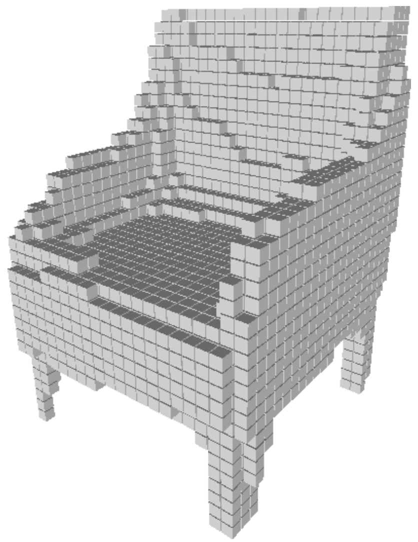


Mesh R-CNN: Hybrid 3D shape representation

Mesh deformation gives good results, but the topology (verts, faces, genus, connected components) fixed by the initial mesh



Our approach: Use voxel predictions to create initial mesh prediction!



Mesh R-CNN Pipeline

Input image

Mesh R-CNN Pipeline

Input image

→

2D object recognition

Mesh R-CNN Pipeline

Input image

2D object recognition

3D object voxels

Mesh R-CNN Pipeline

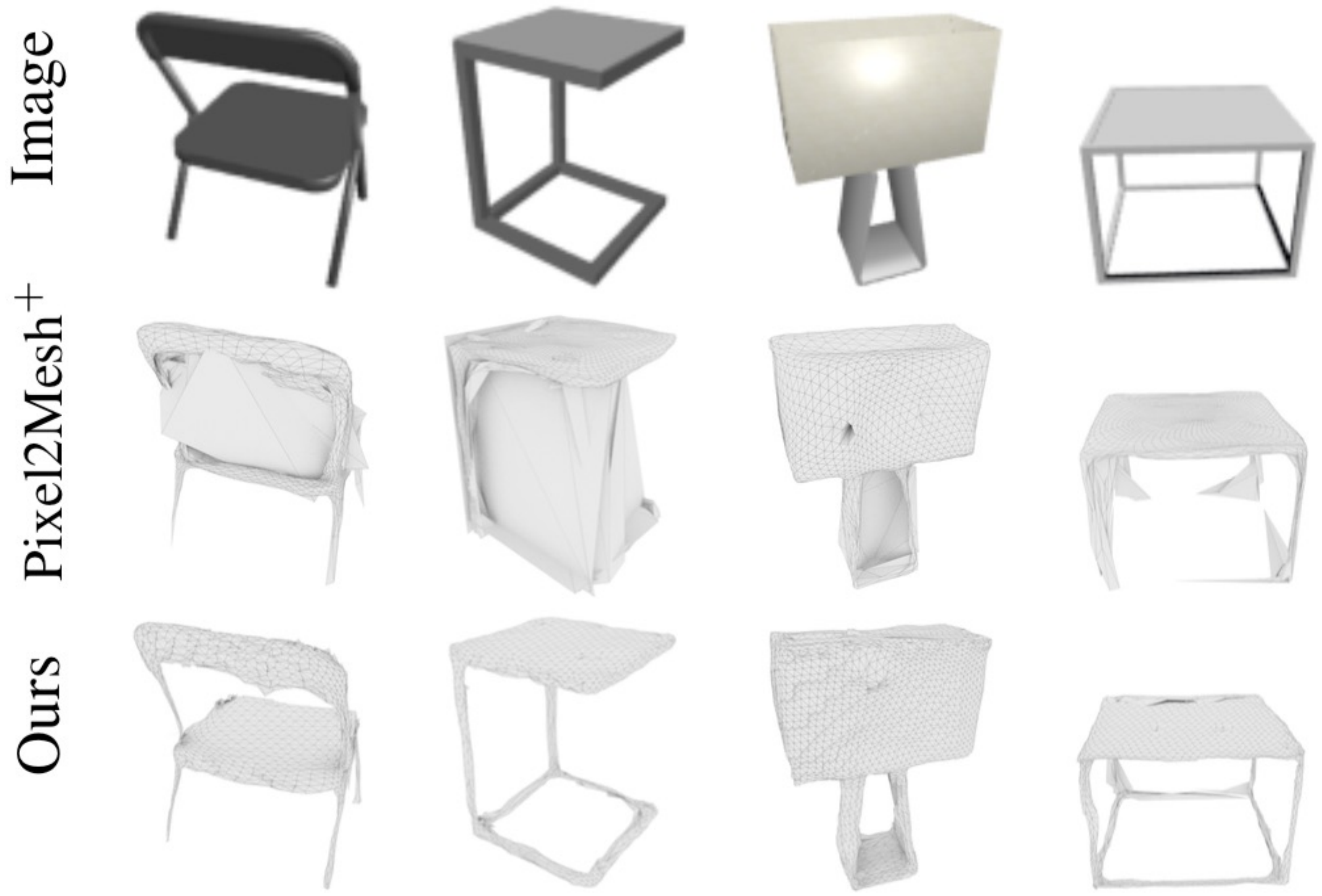
Input image

2D object recognition

3D object voxels

3D object meshes

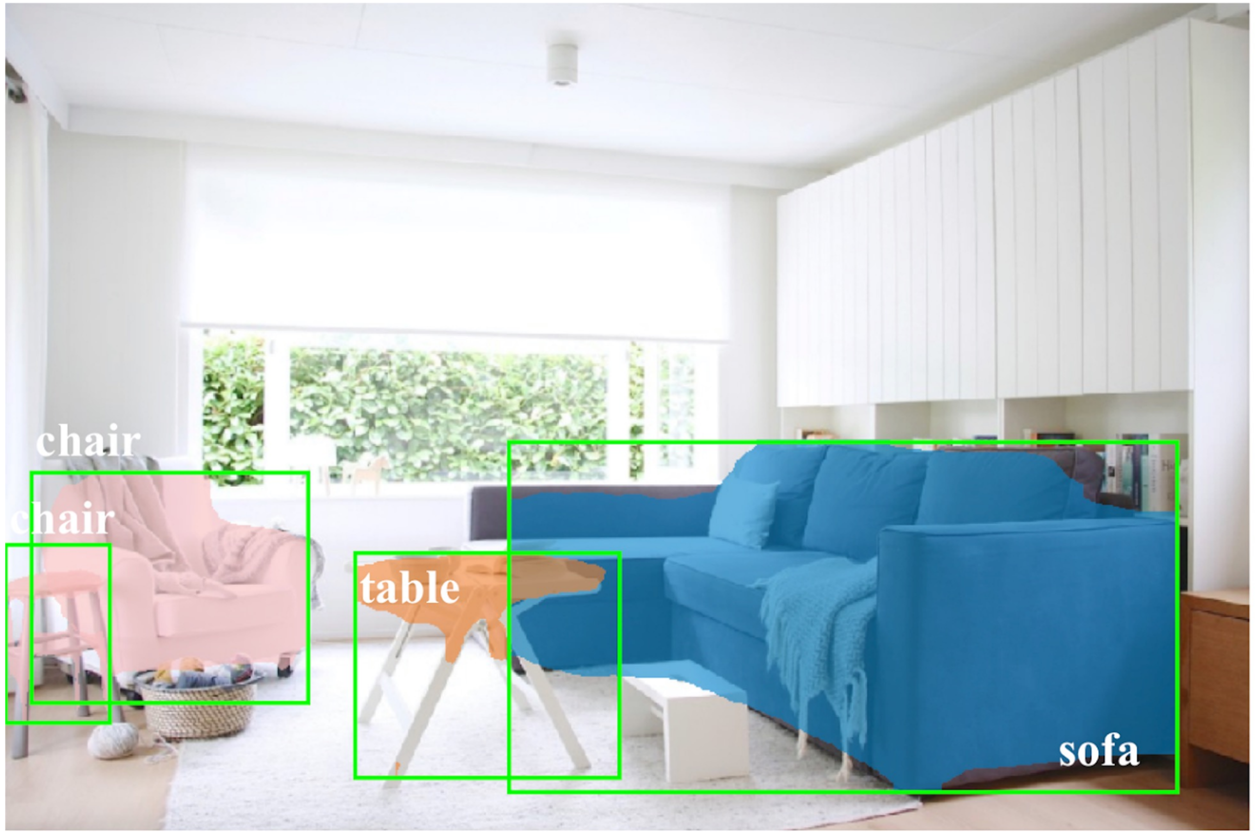
Mesh R-CNN: ShapeNet Results



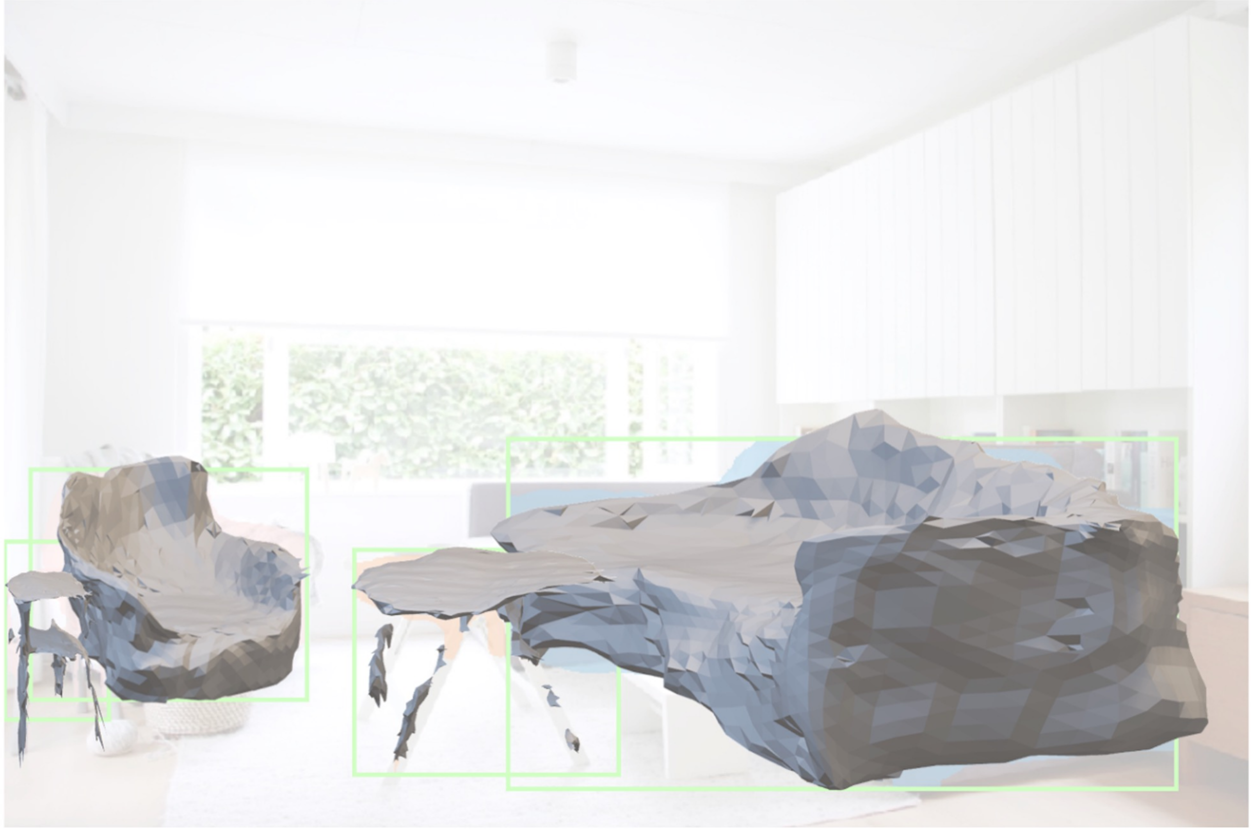
Mesh R-CNN: Pix3D Results

Mesh R-CNN: Pix3D Results

Predicting many objects per scene



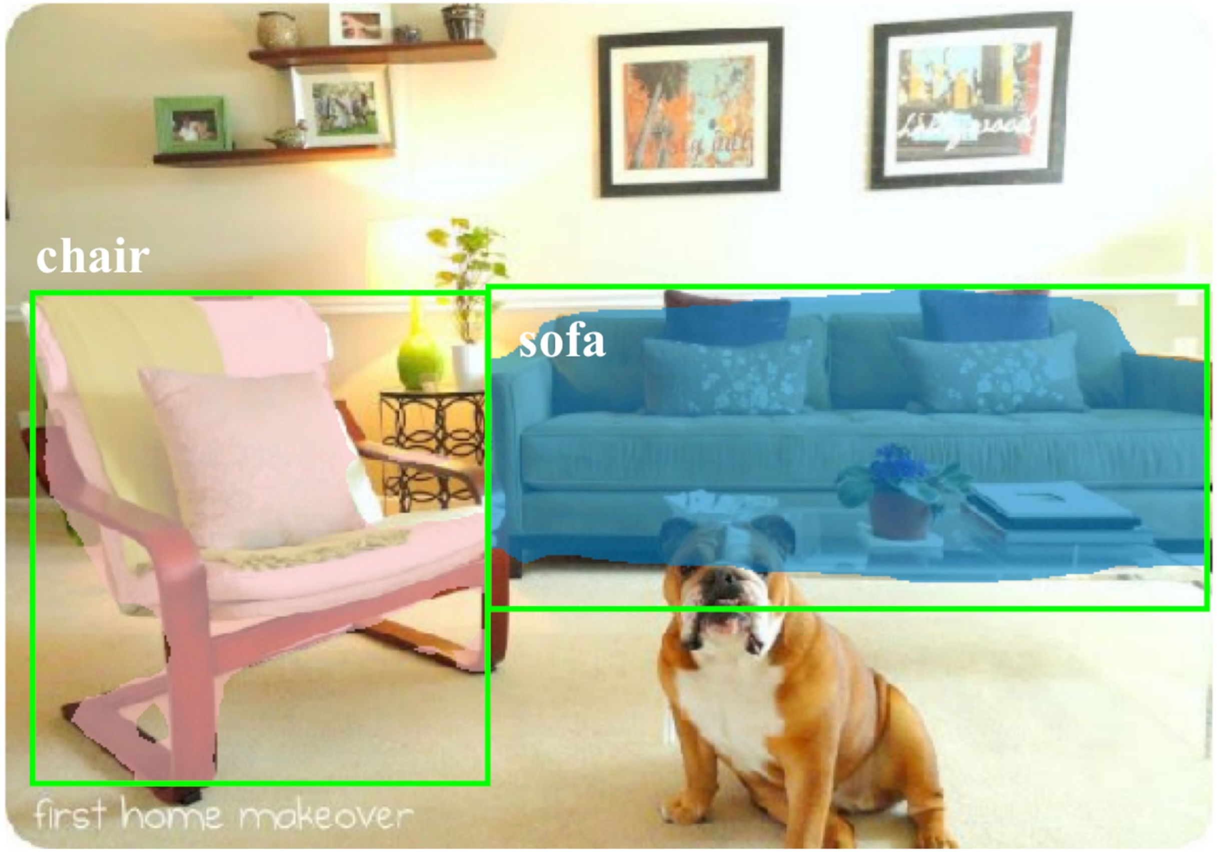
Box & Mask Predictions



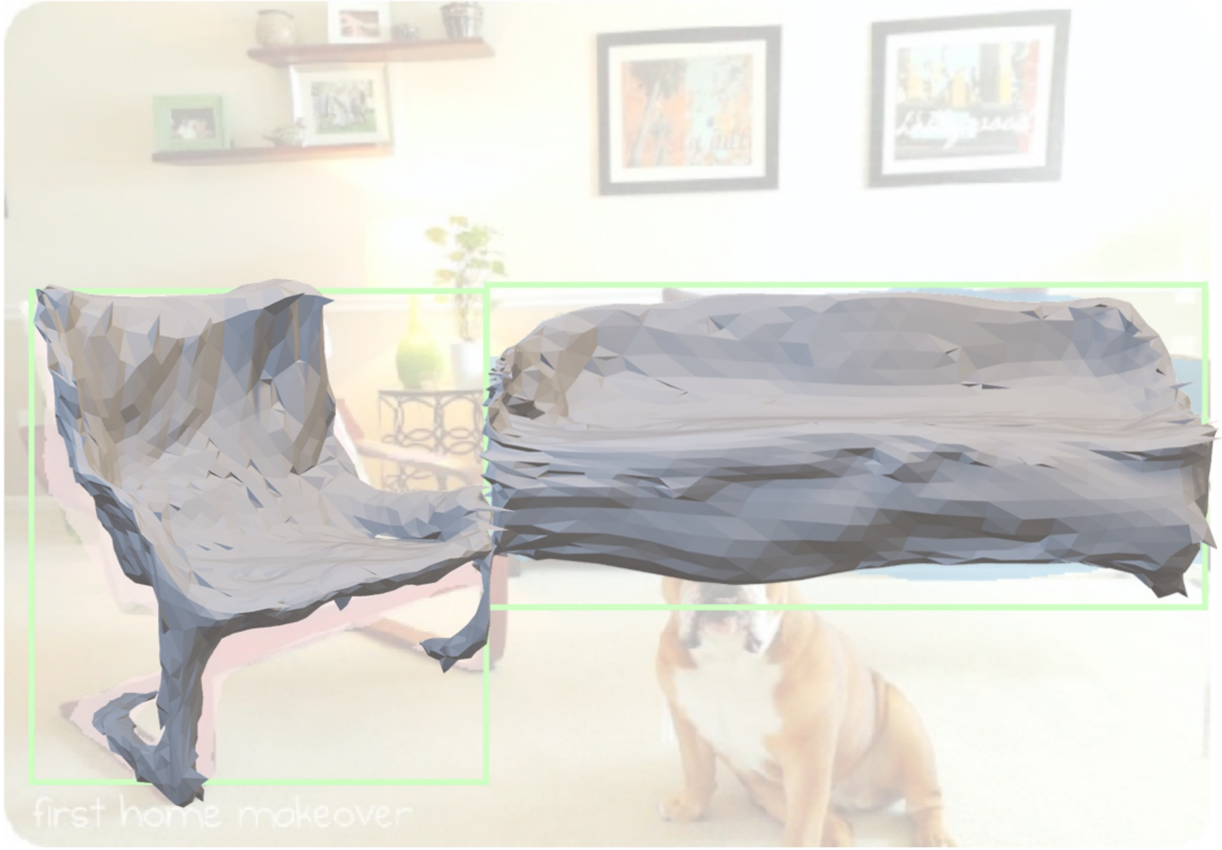
Mesh Predictions

Mesh R-CNN: Pix3D Results

Amodal completion: predict occluded parts of objects



Box & Mask Predictions



Mesh Predictions

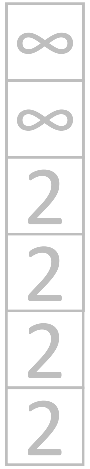
Mesh R-CNN: Pix3D Results

Segmentation failures propagate to meshes

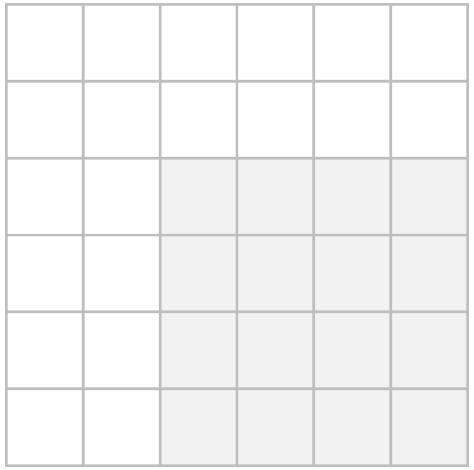
Box & Mask Predictions

Mesh Predictions

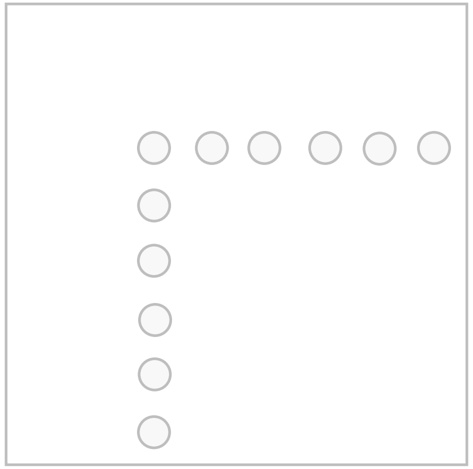
3D Shape Representations



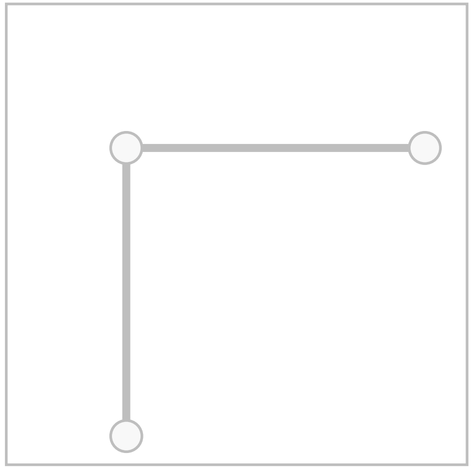
Depth Map



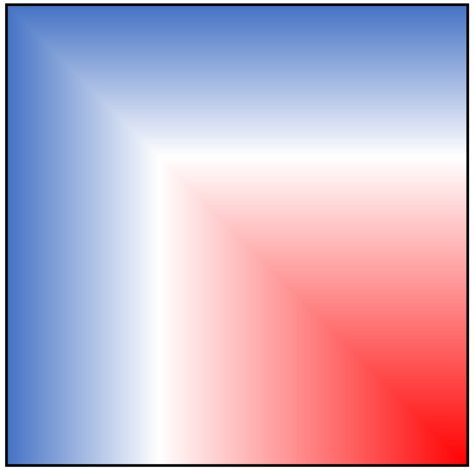
Voxel Grid



Pointcloud



Mesh

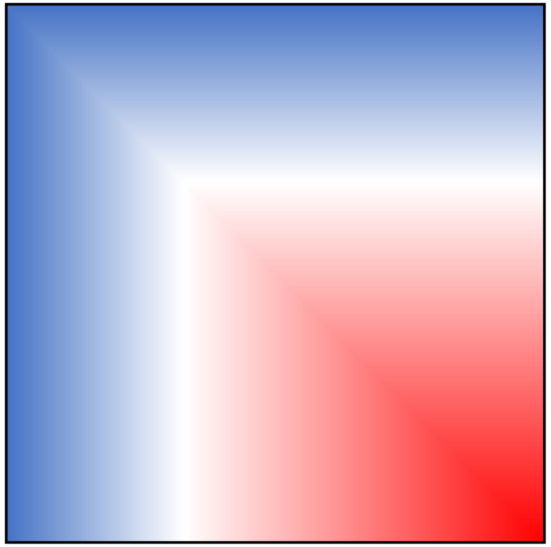


Implicit Surface

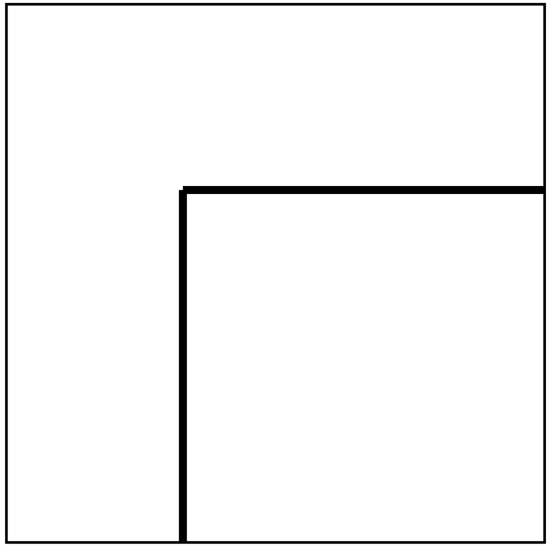
3D Shape Representations: Implicit Functions

Learn a function to classify arbitrary 3D points as inside / outside the shape $o : \mathbb{R}^3 \rightarrow \{0, 1\}$

The surface of the 3D object is the level set $\{x : o(x) = ½\}$



Implicit function



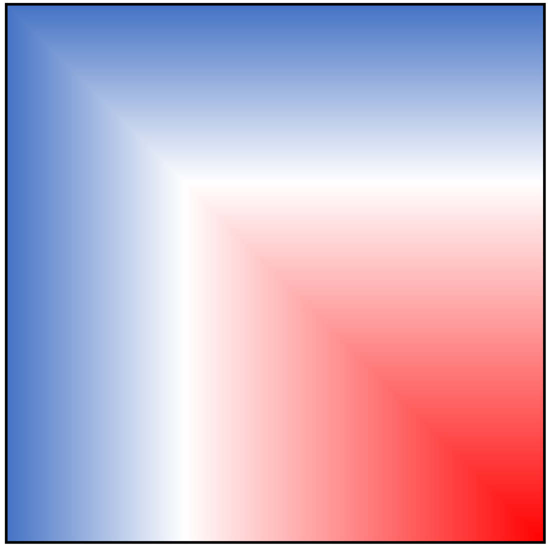
Explicit Shape

3D Shape Representations: Implicit Functions

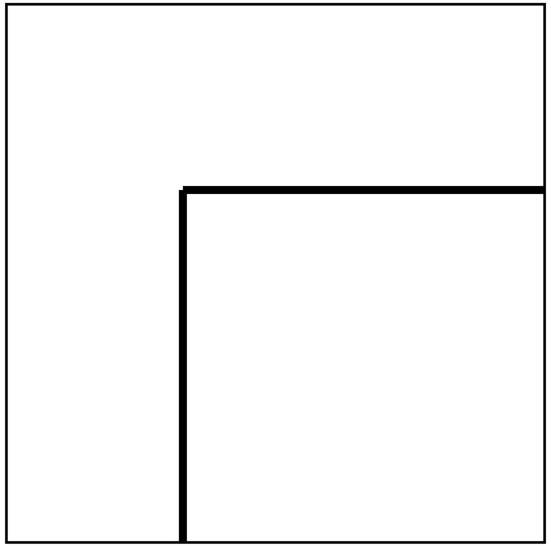
Learn a function to classify arbitrary 3D points as inside / outside the shape

$$o : \mathbb{R}^3 \to \{0, 1\}$$

The surface of the 3D object is the level set $\{x : o(x) = \frac{1}{2}\}$



Implicit function



Explicit Shape

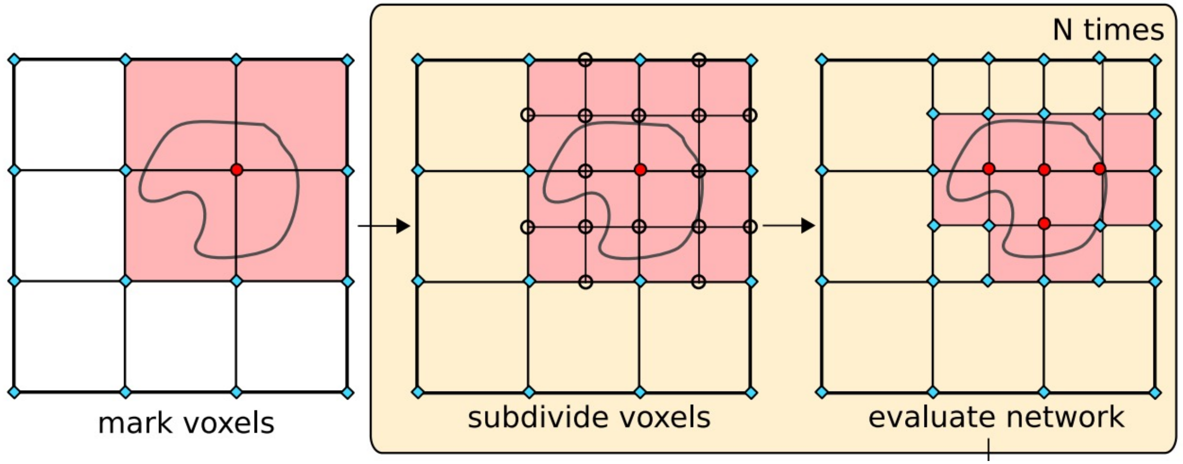
Same idea: **signed distance function (SDF)** gives the Euclidean distance to the surface of the shape; sign gives inside / outside

3D Shape Representations: Implicit Functions

Learn a function to classify arbitrary 3D points as inside / outside the shape

$$o : \mathbb{R}^3 \rightarrow \{0, 1\}$$

The surface of the 3D object is the level set $\{x : o(x) = ½\}$



Allows for multiscale outputs like Oct-Trees

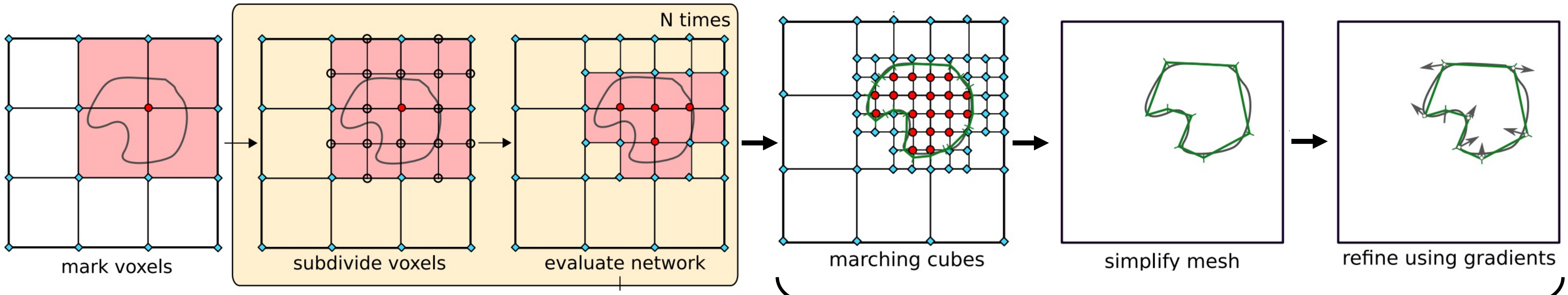
Mescheder et al, "Occupancy Networks: Learning 3D Reconstruction in Function Space", CVPR 2019

3D Shape Representations: Implicit Functions

Learn a function to classify arbitrary 3D points as inside / outside the shape

$$o : \mathbb{R}^3 \rightarrow \{0, 1\}$$

The surface of the 3D object is the level set $\{x : o(x) = ½\}$



Extracting explicit shape outputs requires post-processing

Mescheder et al, "Occupancy Networks: Learning 3D Reconstruction in Function Space", CVPR 2019

Neural Radiance Fields (NeRF) for View Synthesis

View Synthesis

Input: Many images of the same scene (with known camera parameters)

Image source: Mildenhall et al, "Representing Scenes as Neural Radiance Fields for View Synthesis", ECCV 2020

View Synthesis

Input: Many images of the same scene (with known camera parameters)

Output: Images showing the scene from novel viewpoints

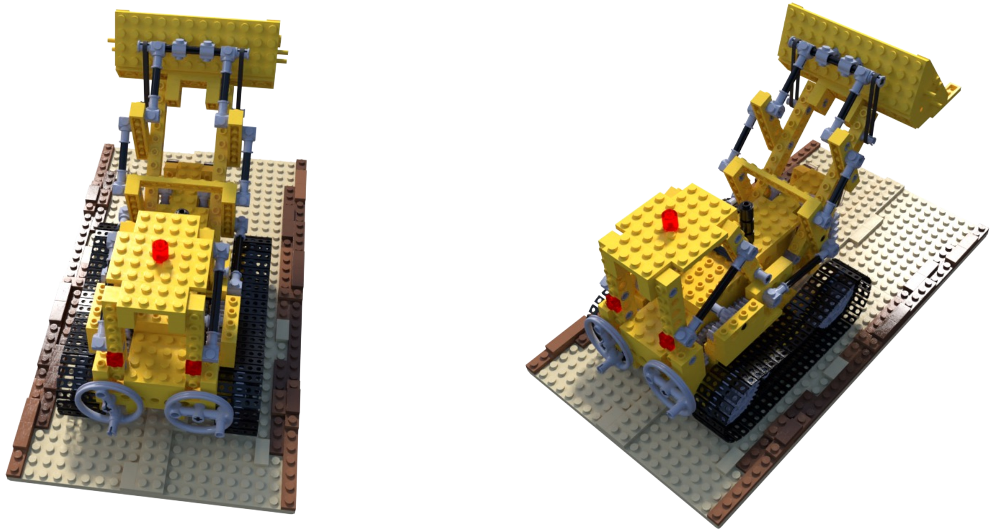
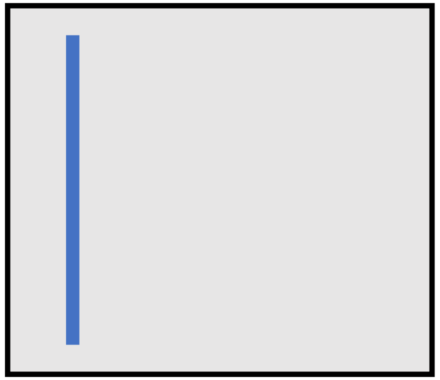


Image source: Mildenhall et al, "Representing Scenes as Neural Radiance Fields for View Synthesis", ECCV 2020

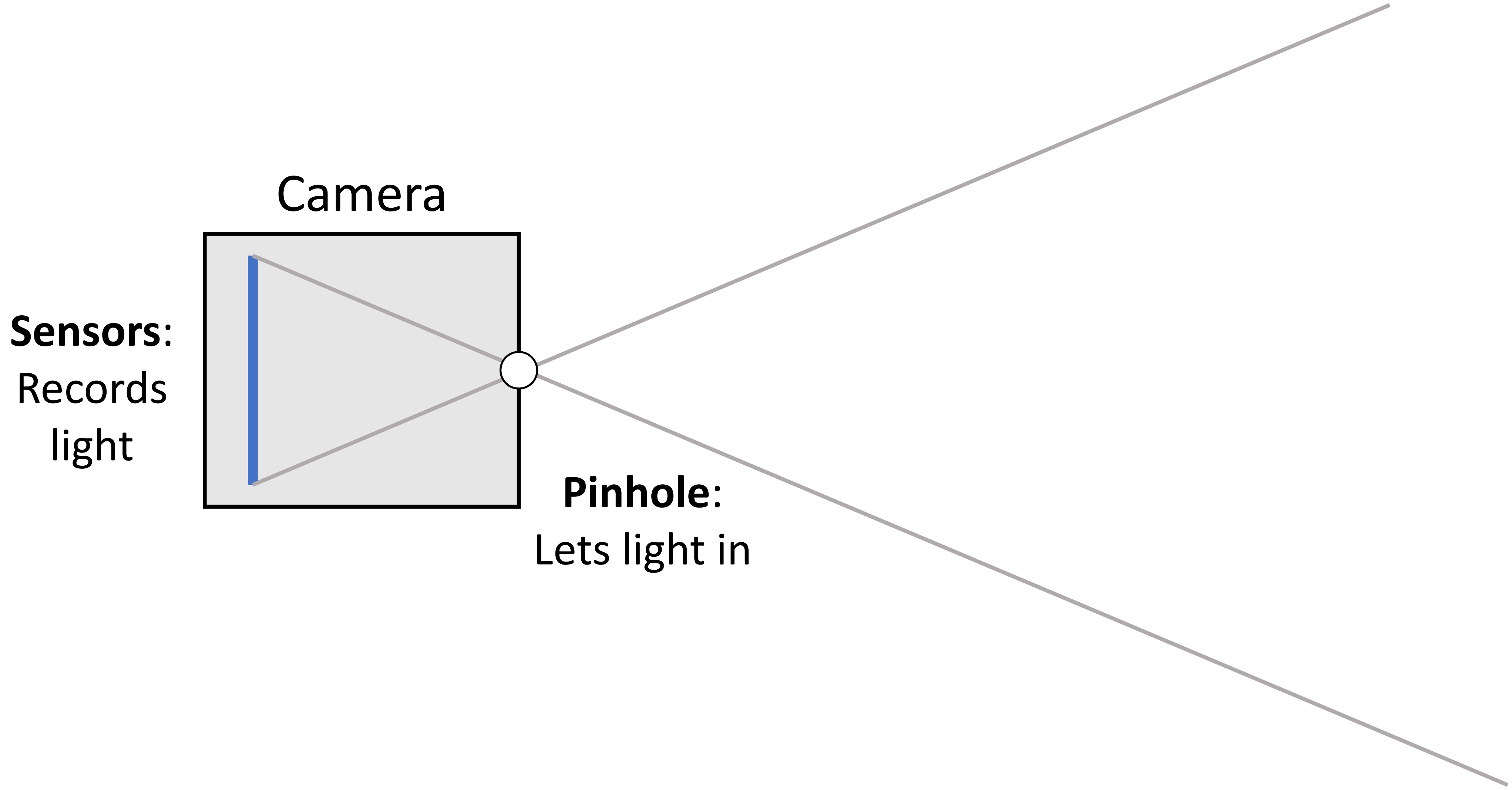
Stepping Back: Pinhole Camera Model

Camera

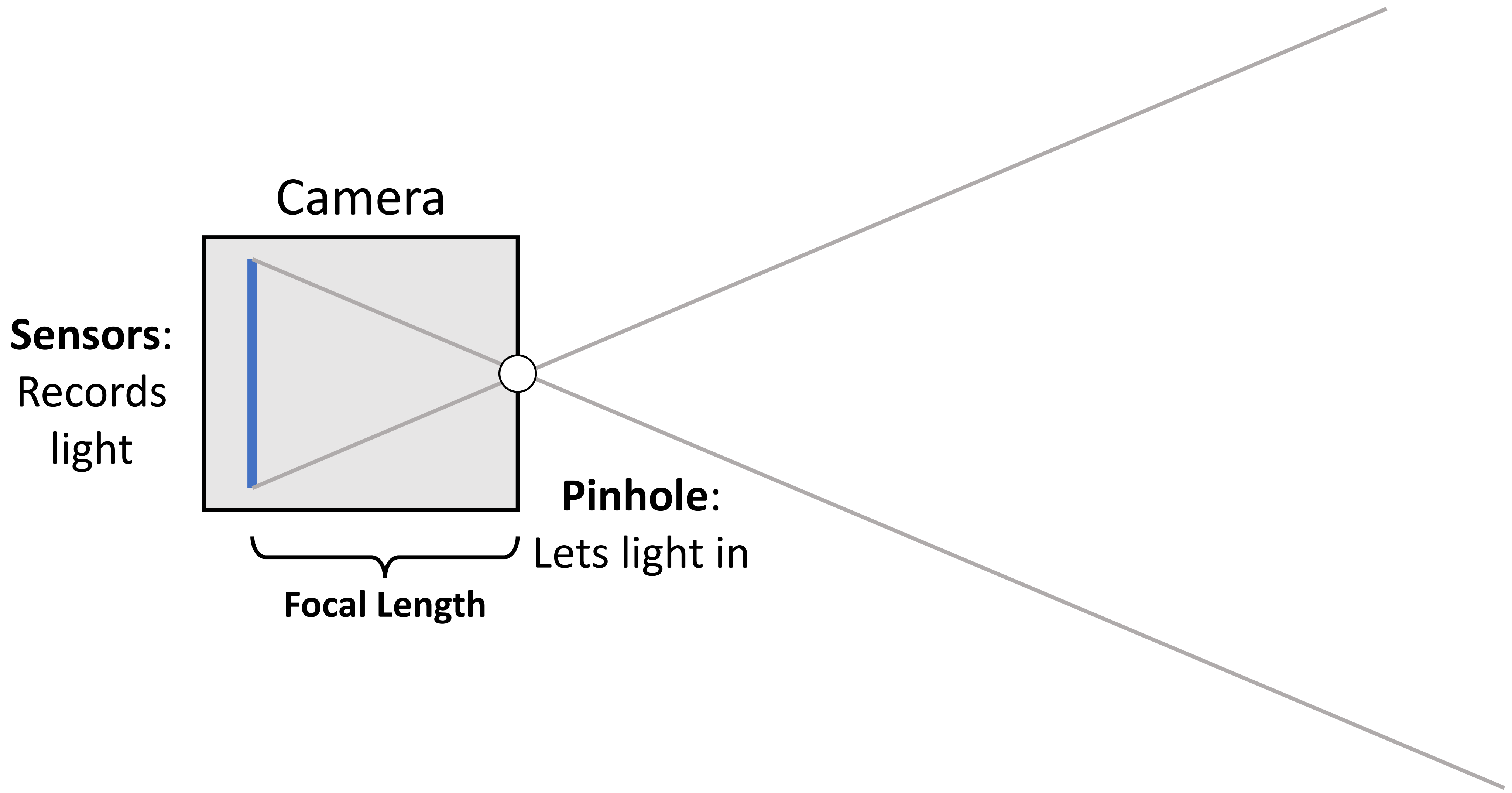
Stepping Back: Pinhole Camera Model



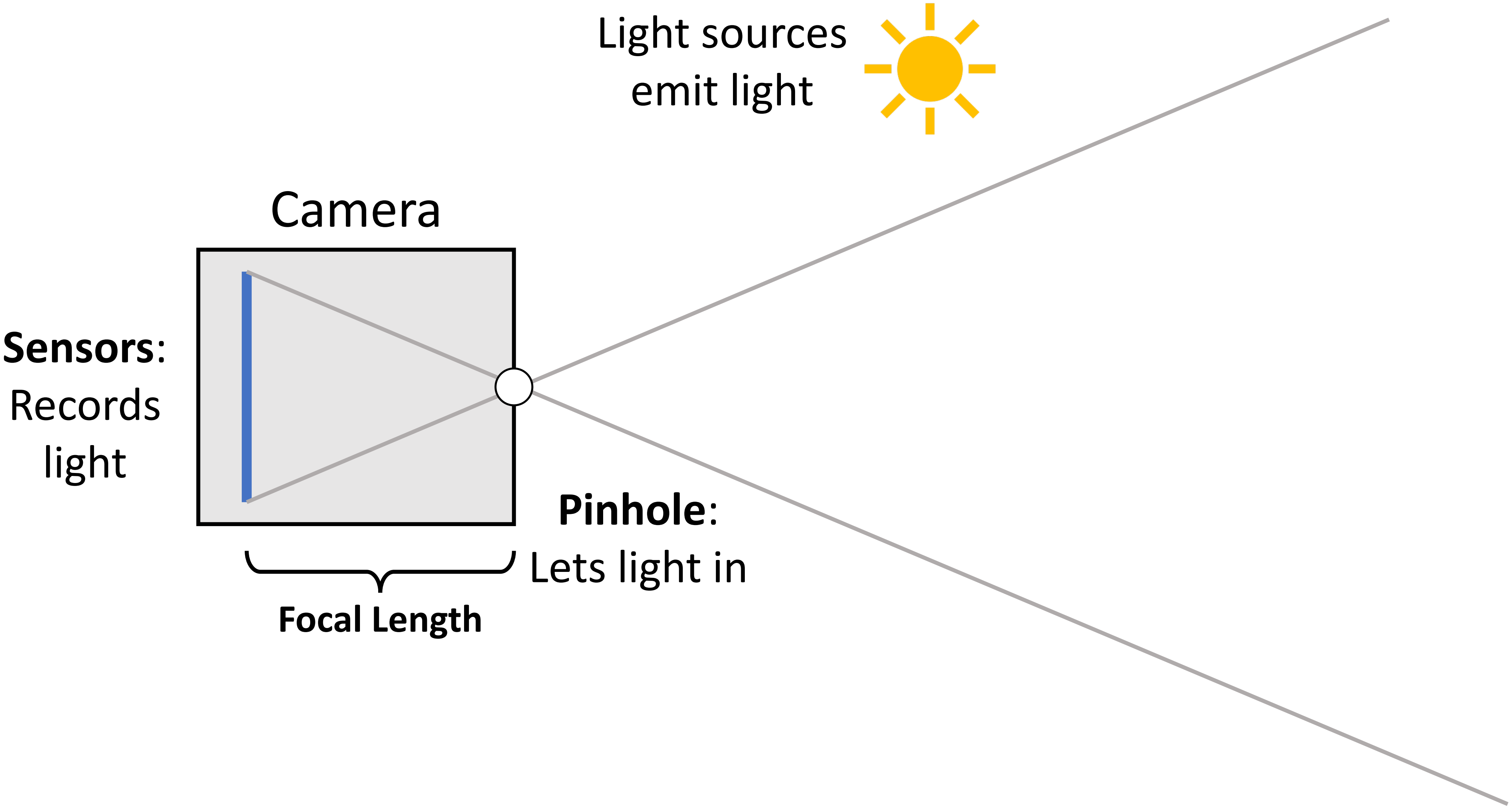
Stepping Back: Pinhole Camera Model



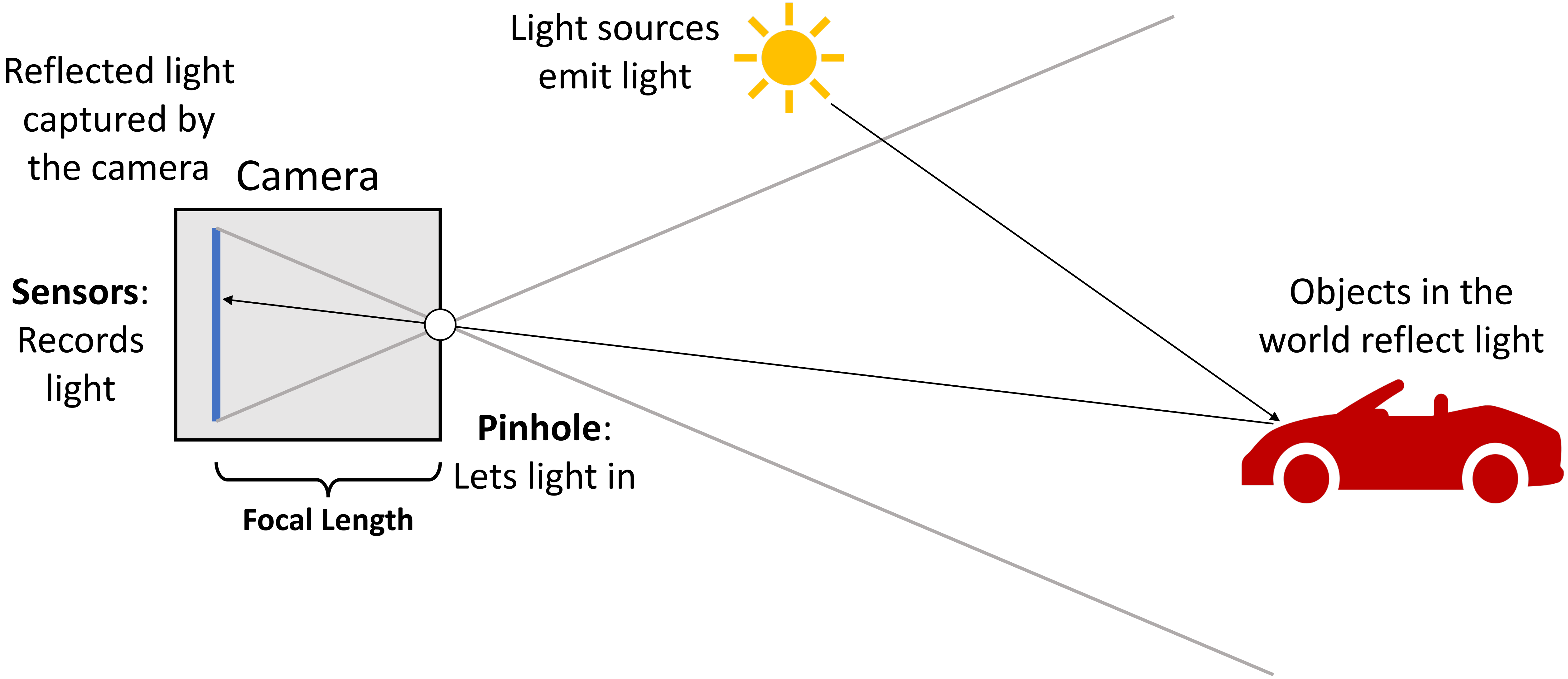
Stepping Back: Pinhole Camera Model



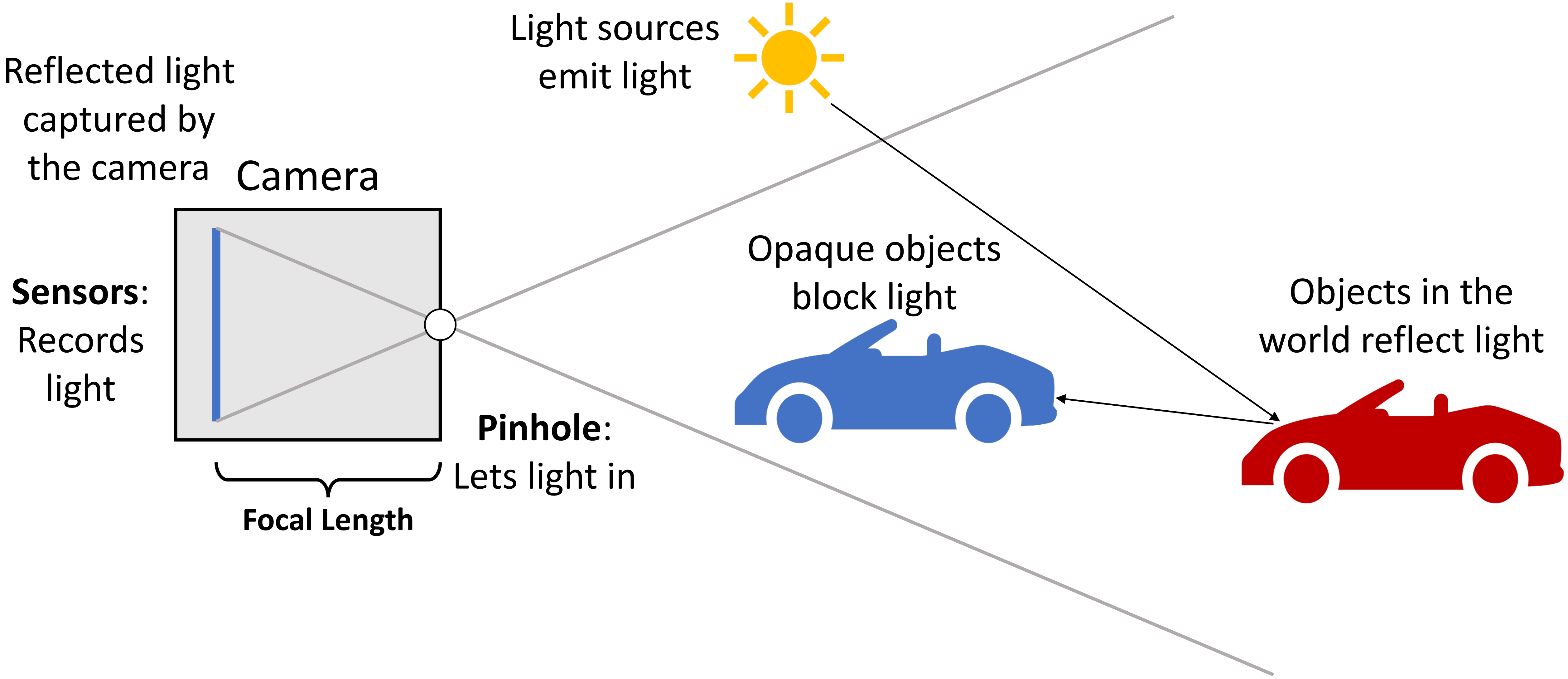
Stepping Back: Pinhole Camera Model



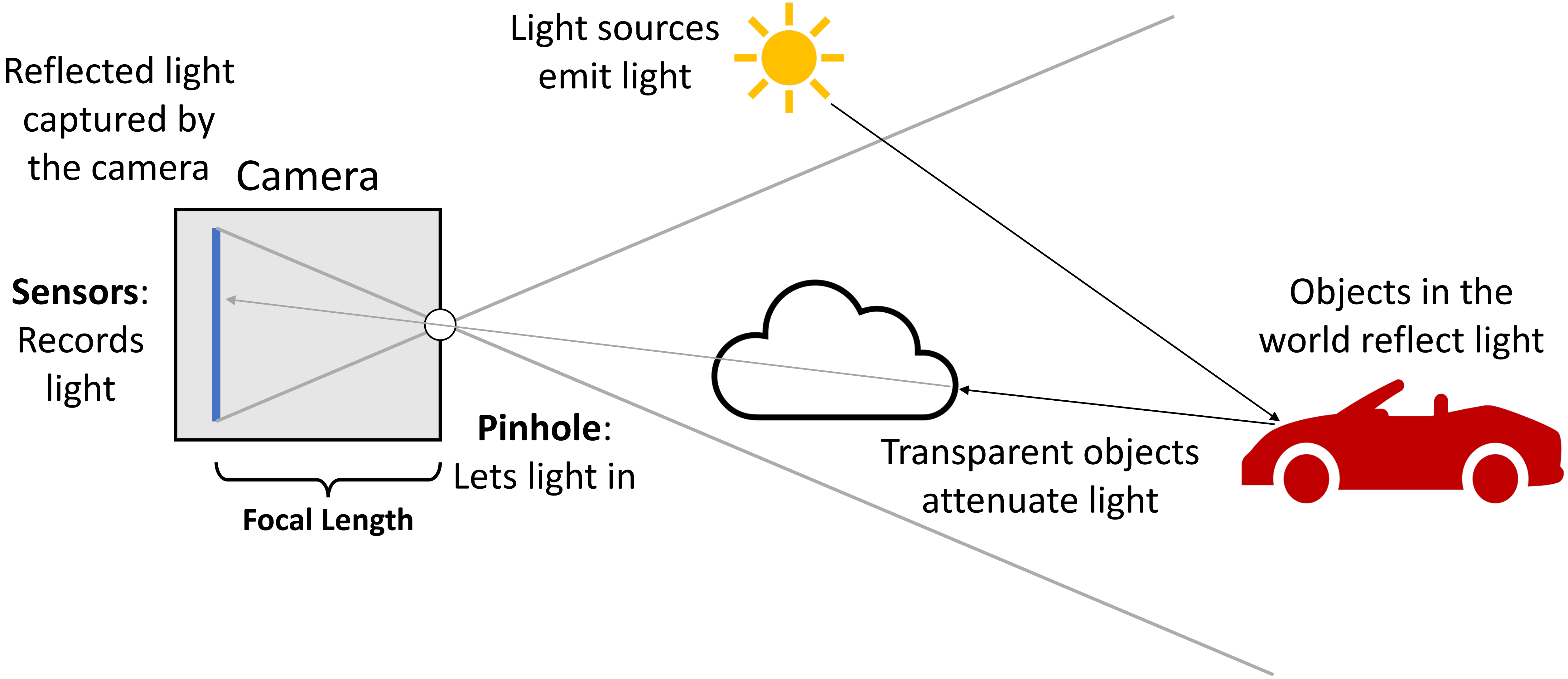
Stepping Back: Pinhole Camera Model



Stepping Back: Pinhole Camera Model



Stepping Back: Pinhole Camera Model



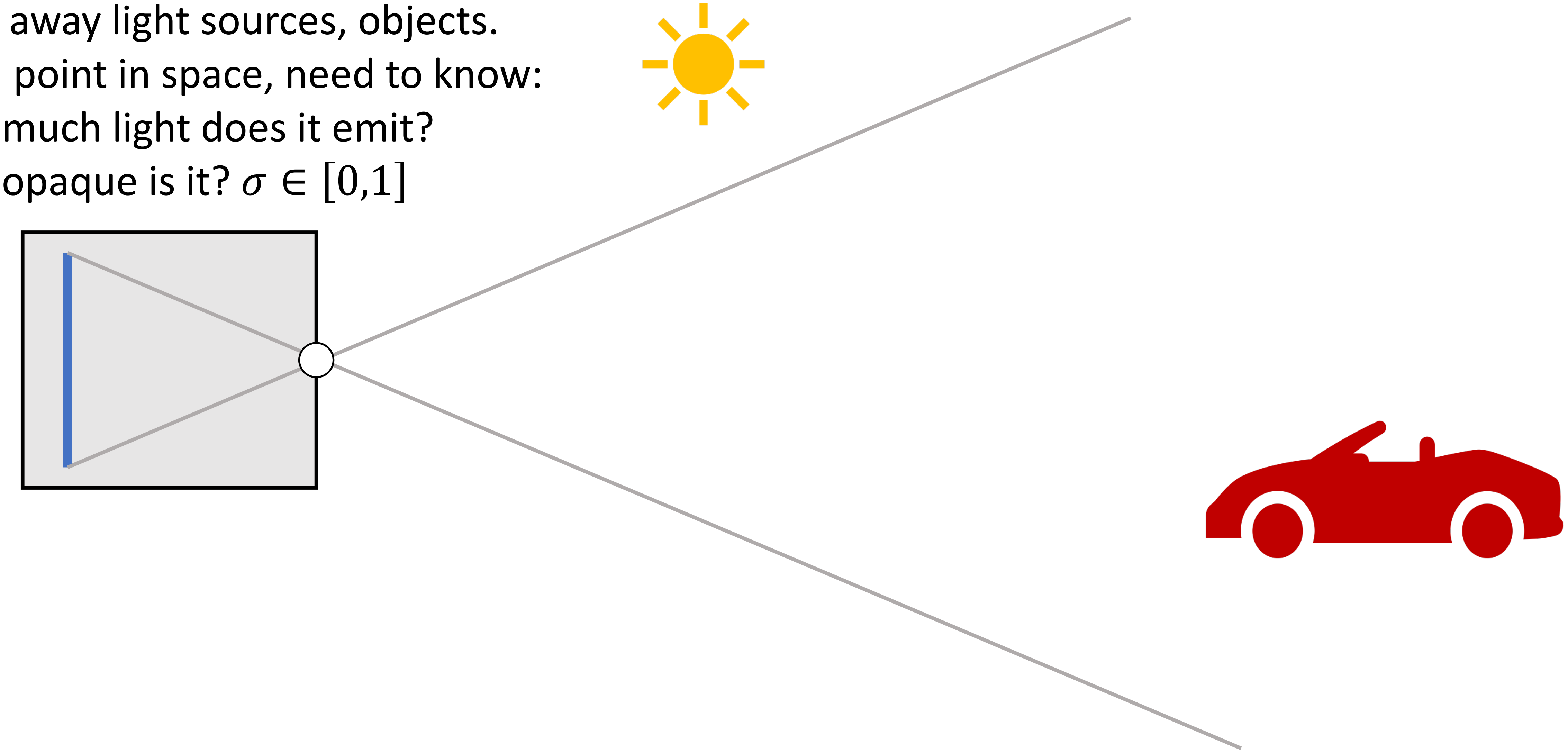
Volume Rendering

Abstract away light sources, objects.

For each point in space, need to know:

(1) How much light does it emit?

(2) How opaque is it? $\sigma \in [0,1]$



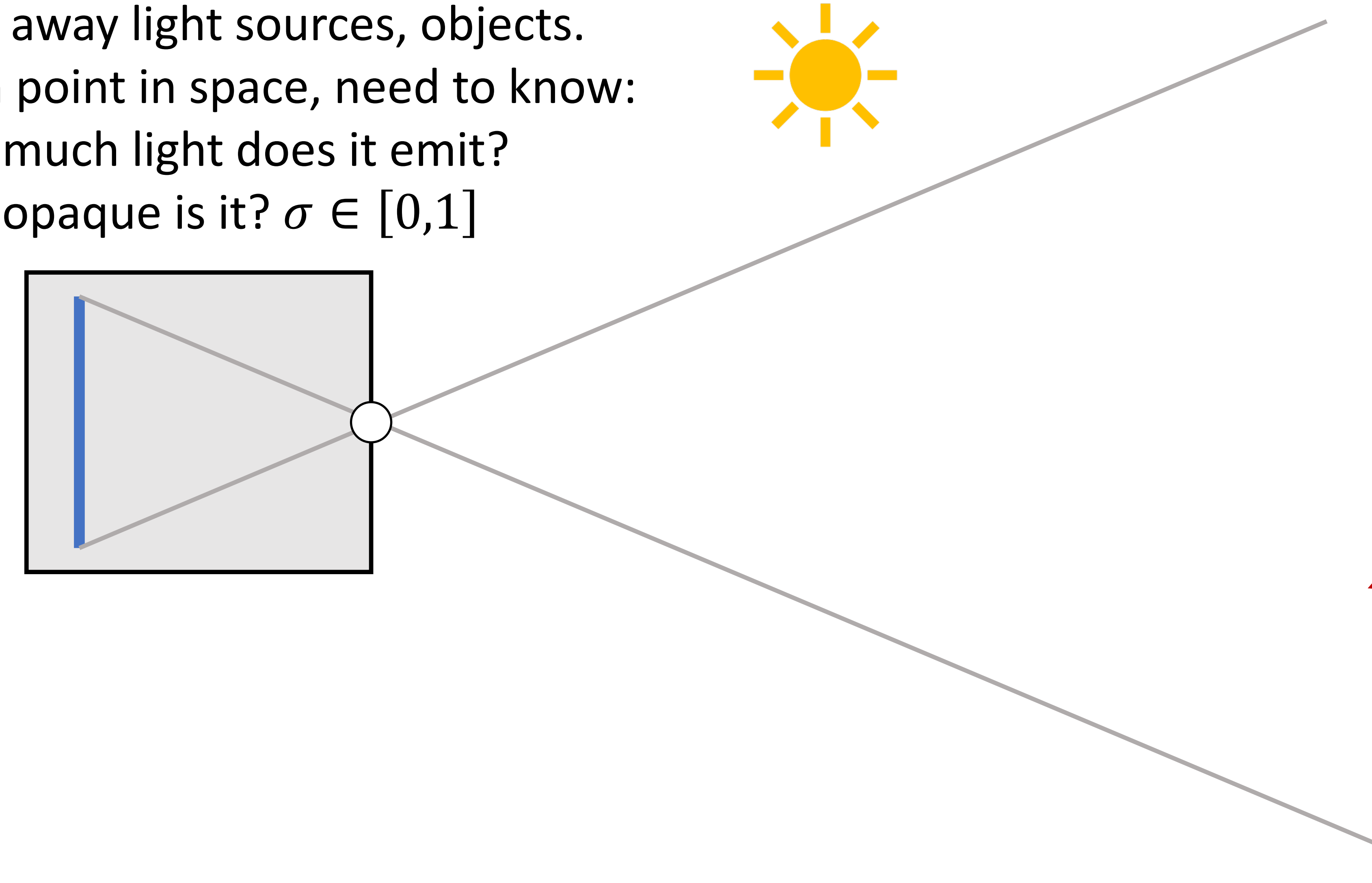
Volume Rendering

Abstract away light sources, objects.

For each point in space, need to know:

(1) How much light does it emit?

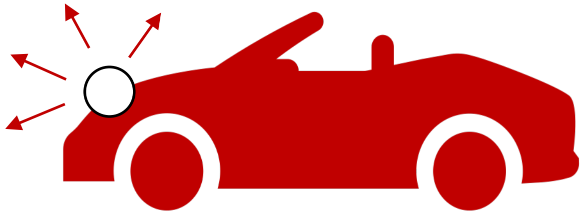
(2) How opaque is it? $\sigma \in [0,1]$



Point on car:

(1) Emits red light in hemisphere

(2) Complete opaque $\sigma = 1$



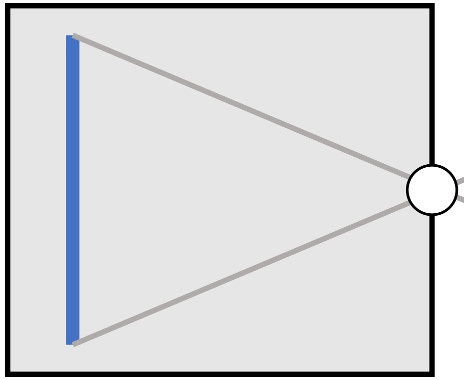
Volume Rendering

Abstract away light sources, objects.

For each point in space, need to know:

(1) How much light does it emit?

(2) How opaque is it? $\sigma \in [0,1]$



Point in empty space:

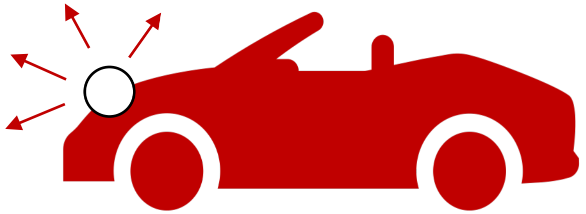
(1) Emits no light (black)

(2) Completely transparent $\sigma = 0$

Point on car:

(1) Emits red light in hemisphere

(2) Complete opaque $\sigma = 1$



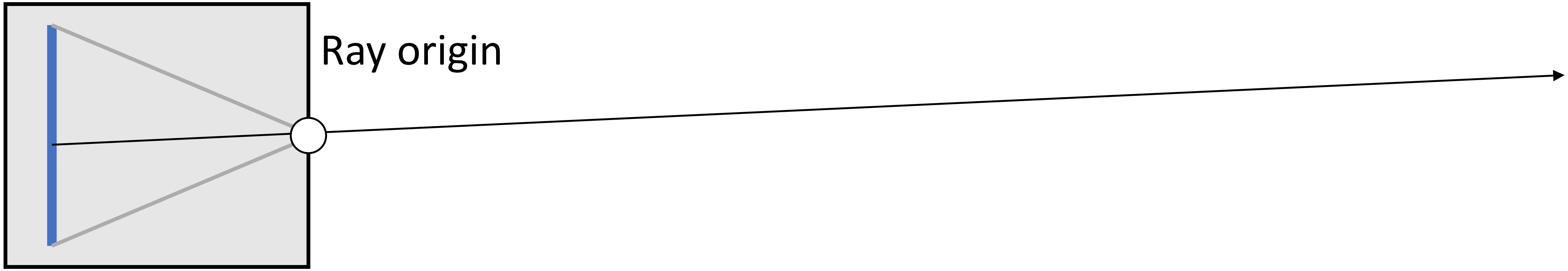
Volume Rendering

Abstract away light sources, objects.

For each point in space, need to know:

(1) How much light does it emit?

(2) How opaque is it? $\sigma \in [0,1]$



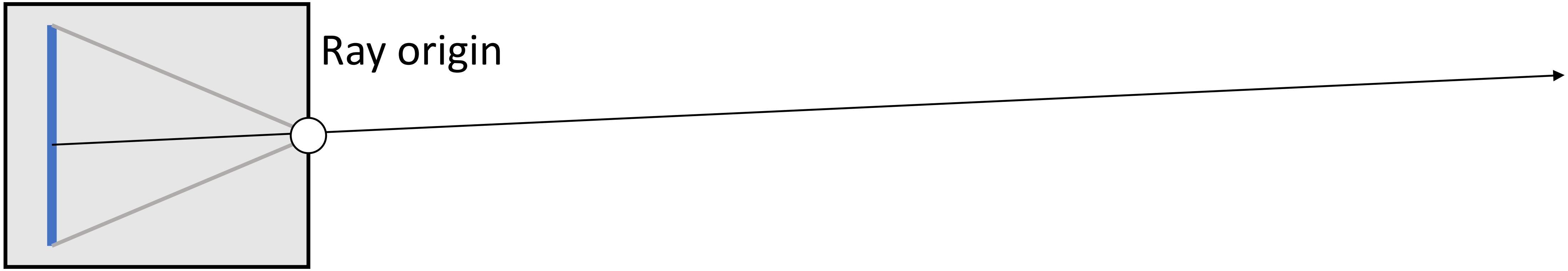
Parameterize each ray as origin plus direction: $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$

Volume Density is $\sigma(\boldsymbol{p}) \in [0,1]$

Color that a point **p** emits in direction **d** is $c(\boldsymbol{p}, \boldsymbol{d}) \in [0,1]^3$

Volume Rendering

Abstract away light sources, objects.
For each point in space, need to know:
(1) How much light does it emit?
(2) How opaque is it? $\sigma \in [0,1]$



Parameterize each ray as origin plus direction: $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$
Volume Density is $\sigma(\boldsymbol{p}) \in [0,1]$
Color that a point **p** emits in direction **d** is $c(\boldsymbol{p}, \boldsymbol{d}) \in [0,1]^3$

Color observed by the camera given by **volume rendering equation**:

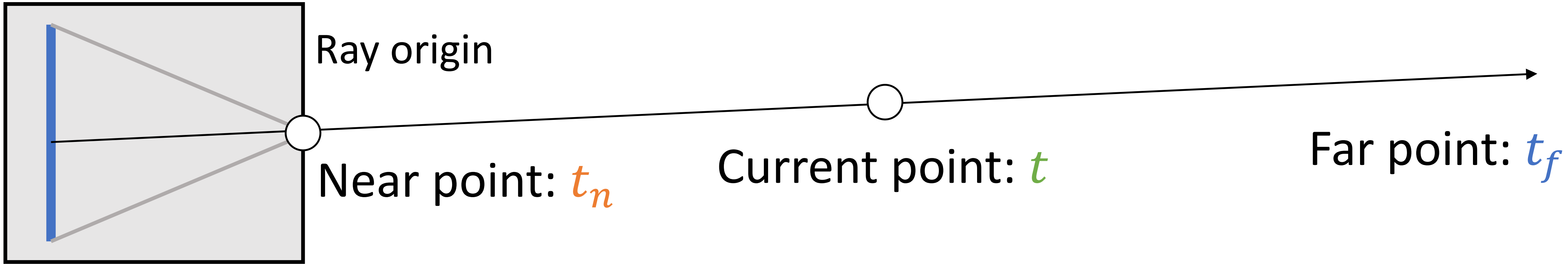
$$C(\boldsymbol{r}) = \int_{t_n}^{t_f} T(t)\sigma\big(\boldsymbol{r}(t)\big)\boldsymbol{c}(\boldsymbol{r}(t), \boldsymbol{d})dt$$

Volume Rendering

Color observed by the camera given by **volume rendering equation**:

$$C(\boldsymbol{r}) = \int_{t_n}^{t_f} T(t)\sigma\big(\boldsymbol{r}(t)\big)\boldsymbol{c}(\boldsymbol{r}(t), \boldsymbol{d})dt$$

Abstract away light sources, objects.
For each point in space, need to know:
(1) How much light does it emit?
(2) How opaque is it? $\sigma \in [0,1]$



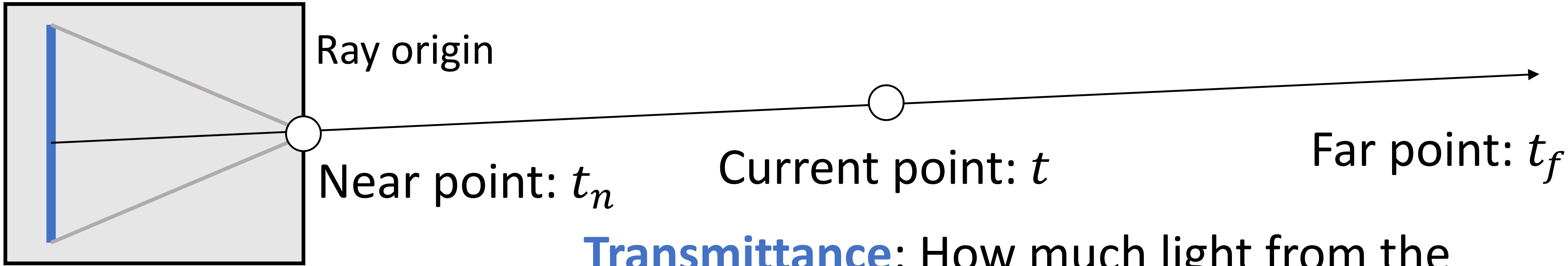
Parameterize each ray as origin plus direction: $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$
Volume Density is $\sigma(\boldsymbol{p}) \in [0,1]$
Color that a point **p** emits in direction **d** is $c(\boldsymbol{p}, \boldsymbol{d}) \in [0,1]^3$

Volume Rendering

Abstract away light sources, objects.
For each point in space, need to know:
(1) How much light does it emit?
(2) How opaque is it? $\sigma \in [0,1]$

Color observed by the camera given by **volume rendering equation**:

$$C(\boldsymbol{r}) = \int_{t_n}^{t_f} T(t)\sigma\big(\boldsymbol{r}(t)\big)\boldsymbol{c}(\boldsymbol{r}(t), \boldsymbol{d})dt$$



Parameterize each ray as origin plus direction: $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$
Volume Density is $\sigma(\boldsymbol{p}) \in [0,1]$
Color that a point **p** emits in direction **d** is $c(\boldsymbol{p}, \boldsymbol{d}) \in [0,1]^3$

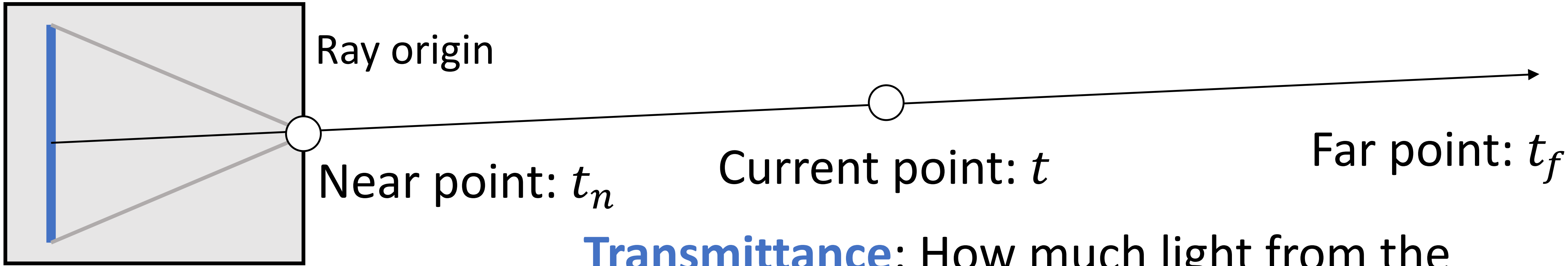
Transmittance: How much light from the current point will reach the camera?

Volume Rendering

Color observed by the camera given by **volume rendering equation**:

$$C(\boldsymbol{r}) = \int_{t_n}^{t_f} T(t)\sigma(\boldsymbol{r}(t))\boldsymbol{c}(\boldsymbol{r}(t), \boldsymbol{d})dt$$

Abstract away light sources, objects.
For each point in space, need to know:
(1) How much light does it emit?
(2) How opaque is it? $\sigma \in [0,1]$



Parameterize each ray as origin plus direction: $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$
Volume Density is $\sigma(\boldsymbol{p}) \in [0,1]$
Color that a point **p** emits in direction **d** is $c(\boldsymbol{p}, \boldsymbol{d}) \in [0,1]^3$

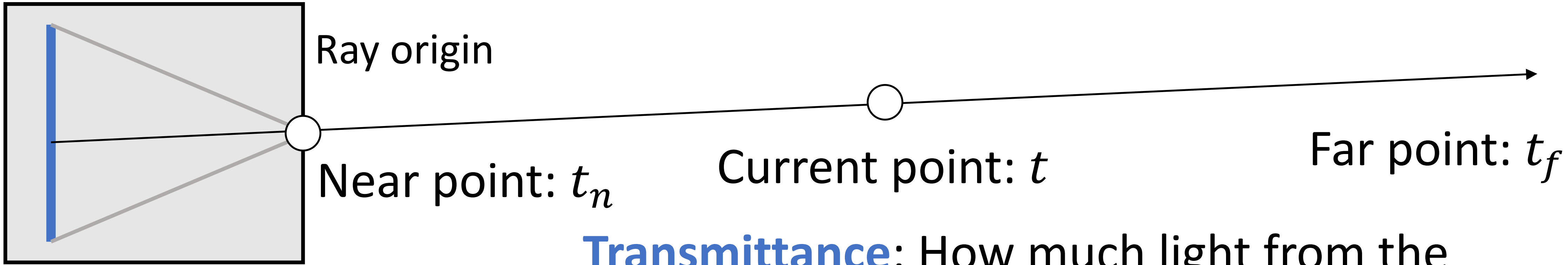
Transmittance: How much light from the current point will reach the camera?
Opacity: How opaque is the current point?

Volume Rendering

Abstract away light sources, objects.
For each point in space, need to know:
(1) How much light does it emit?
(2) How opaque is it? $\sigma \in [0,1]$

Color observed by the camera given by **volume rendering equation**:

$$C(\boldsymbol{r}) = \int_{t_n}^{t_f} T(t)\sigma(\boldsymbol{r}(t))\boldsymbol{c}(\boldsymbol{r}(t), \boldsymbol{d})dt$$



Parameterize each ray as origin plus direction: $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$
Volume Density is $\sigma(\boldsymbol{p}) \in [0,1]$
Color that a point **p** emits in direction **d** is $c(\boldsymbol{p}, \boldsymbol{d}) \in [0,1]^3$

Transmittance: How much light from the current point will reach the camera?
Opacity: How opaque is the current point?
Color: What color does the current point emit along the direction toward the camera?

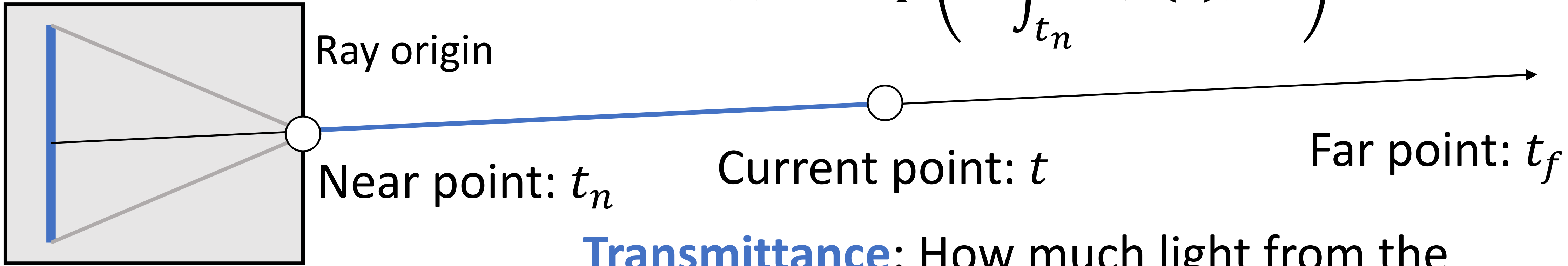
Volume Rendering

Abstract away light sources, objects.
For each point in space, need to know:
(1) How much light does it emit?
(2) How opaque is it? $\sigma \in [0,1]$

Color observed by the camera given by **volume rendering equation**:

$$C(\boldsymbol{r}) = \int_{t_n}^{t_f} T(t)\sigma(\boldsymbol{r}(t))\boldsymbol{c}(\boldsymbol{r}(t), \boldsymbol{d})dt$$

$$T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\boldsymbol{r}(s))ds\right)$$



Parameterize each ray as origin plus direction: $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$
Volume Density is $\sigma(\boldsymbol{p}) \in [0,1]$
Color that a point **p** emits in direction **d** is $c(\boldsymbol{p}, \boldsymbol{d}) \in [0,1]^3$

Transmittance: How much light from the current point will reach the camera?

Compute transmittance by accumulating volume density up to current point

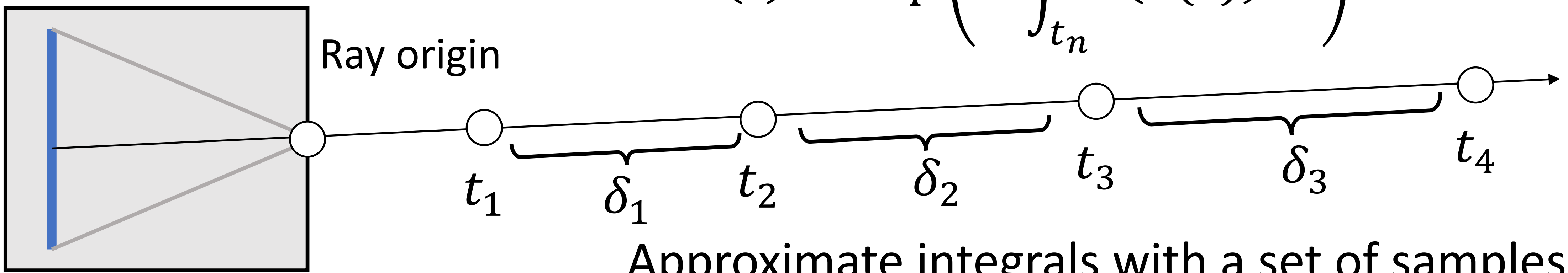
Volume Rendering

Abstract away light sources, objects.
For each point in space, need to know:
(1) How much light does it emit?
(2) How opaque is it? $\sigma \in [0,1]$

Color observed by the camera given by **volume rendering equation**:

$$C(\boldsymbol{r}) = \int_{t_n}^{t_f} T(t)\sigma(\boldsymbol{r}(t))\boldsymbol{c}(\boldsymbol{r}(t), \boldsymbol{d})dt$$

$$T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\boldsymbol{r}(s))ds\right)$$



Approximate integrals with a set of samples:

Parameterize each ray as origin plus direction: $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$
Volume Density is $\sigma(\boldsymbol{p}) \in [0,1]$
Color that a point **p** emits in direction **d** is $c(\boldsymbol{p}, \boldsymbol{d}) \in [0,1]^3$

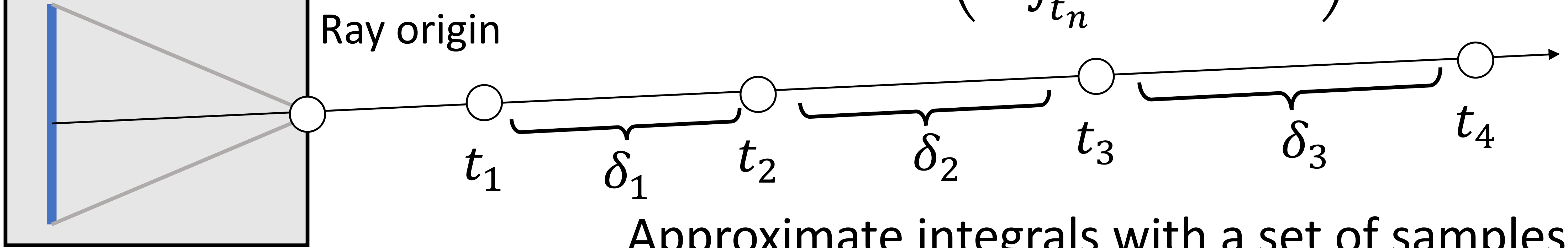
Volume Rendering

Abstract away light sources, objects.
For each point in space, need to know:
(1) How much light does it emit?
(2) How opaque is it? $\sigma \in [0,1]$

Color observed by the camera given by **volume rendering equation**:

$$C(\boldsymbol{r}) = \int_{t_n}^{t_f} T(t)\sigma(\boldsymbol{r}(t))\boldsymbol{c}(\boldsymbol{r}(t),\boldsymbol{d})dt$$

$$T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\boldsymbol{r}(s))ds\right)$$



Approximate integrals with a set of samples:

$$C(\boldsymbol{r}) \approx \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))\, \boldsymbol{c}_i$$

$$T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j\delta_j\right)$$

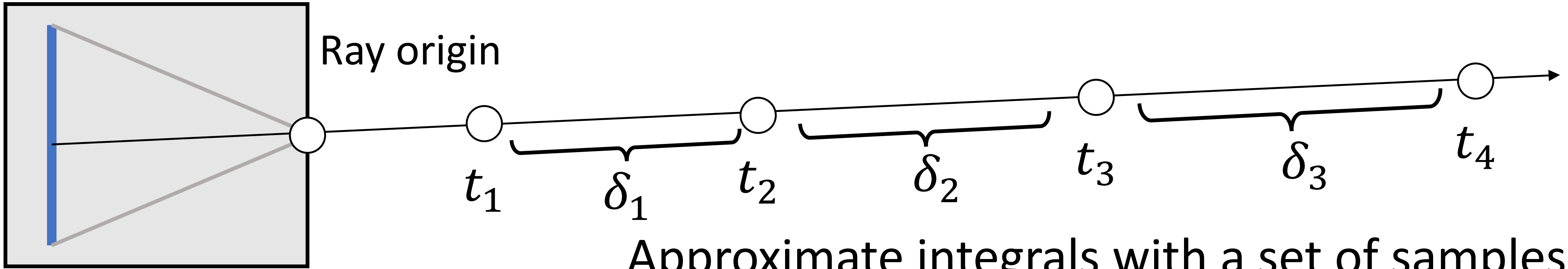
Parameterize each ray as origin plus direction: $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$
Volume Density is $\sigma(\boldsymbol{p}) \in [0,1]$
Color that a point **p** emits in direction **d** is $c(\boldsymbol{p},\boldsymbol{d}) \in [0,1]^3$

Mildenhall et al, "Representing Scenes as Neural Radiance Fields for View Synthesis", ECCV 2020

Neural Radiance Fields (NeRF)

Abstract away light sources, objects.
For each point in space, need to know:
(1) How much light does it emit?
(2) How opaque is it? $\sigma \in [0,1]$

Train a neural network to input position **p** and direction **d**, output $\sigma(\boldsymbol{p})$ and $c(\boldsymbol{p}, \boldsymbol{d})$



Parameterize each ray as origin plus direction: $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$
Volume Density is $\sigma(\boldsymbol{p}) \in [0,1]$
Color that a point **p** emits in direction **d** is $c(\boldsymbol{p}, \boldsymbol{d}) \in [0,1]^3$

Approximate integrals with a set of samples:

$$C(\boldsymbol{r}) \approx \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i \delta_i))\, \boldsymbol{c}_i$$

$$T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right)$$

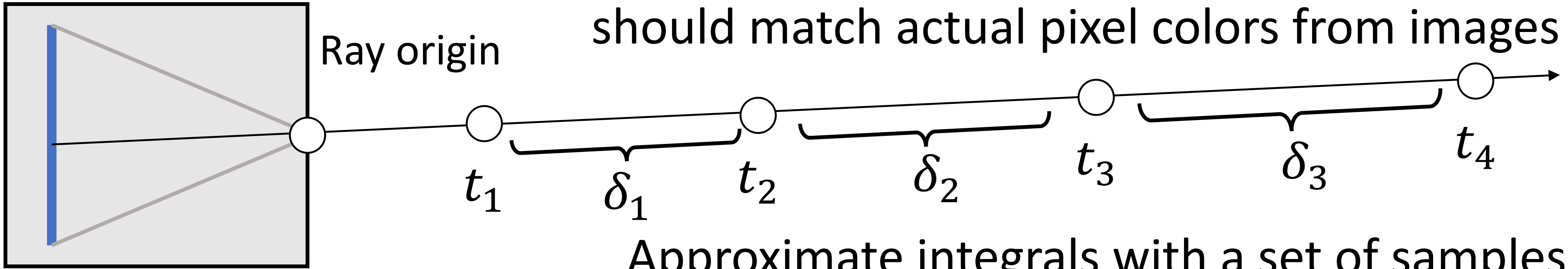
Mildenhall et al, "Representing Scenes as Neural Radiance Fields for View Synthesis", ECCV 2020

Neural Radiance Fields (NeRF)

Abstract away light sources, objects.
For each point in space, need to know:
(1) How much light does it emit?
(2) How opaque is it? $\sigma \in [0,1]$

Train a neural network to input position **p** and direction **d**, output $\sigma(\boldsymbol{p})$ and $c(\boldsymbol{p},\boldsymbol{d})$

Training loss: Estimated pixel colors $C(\boldsymbol{r})$ should match actual pixel colors from images



Parameterize each ray as origin plus direction: $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$
Volume Density is $\sigma(\boldsymbol{p}) \in [0,1]$
Color that a point **p** emits in direction **d** is $c(\boldsymbol{p},\boldsymbol{d}) \in [0,1]^3$

Approximate integrals with a set of samples:

$$C(\boldsymbol{r}) \approx \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))\,\boldsymbol{c}_i$$

$$T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j\delta_j\right)$$

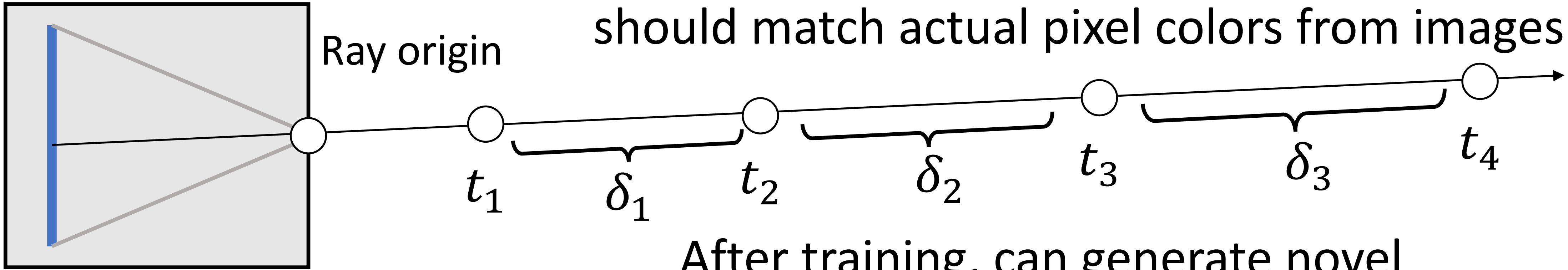
Mildenhall et al, "Representing Scenes as Neural Radiance Fields for View Synthesis", ECCV 2020

Neural Radiance Fields (NeRF)

Abstract away light sources, objects.
For each point in space, need to know:
(1) How much light does it emit?
(2) How opaque is it? $\sigma \in [0,1]$

Train a neural network to input position **p** and direction **d**, output $\sigma(\boldsymbol{p})$ and $c(\boldsymbol{p}, \boldsymbol{d})$

Training loss: Estimated pixel colors $C(\boldsymbol{r})$ should match actual pixel colors from images



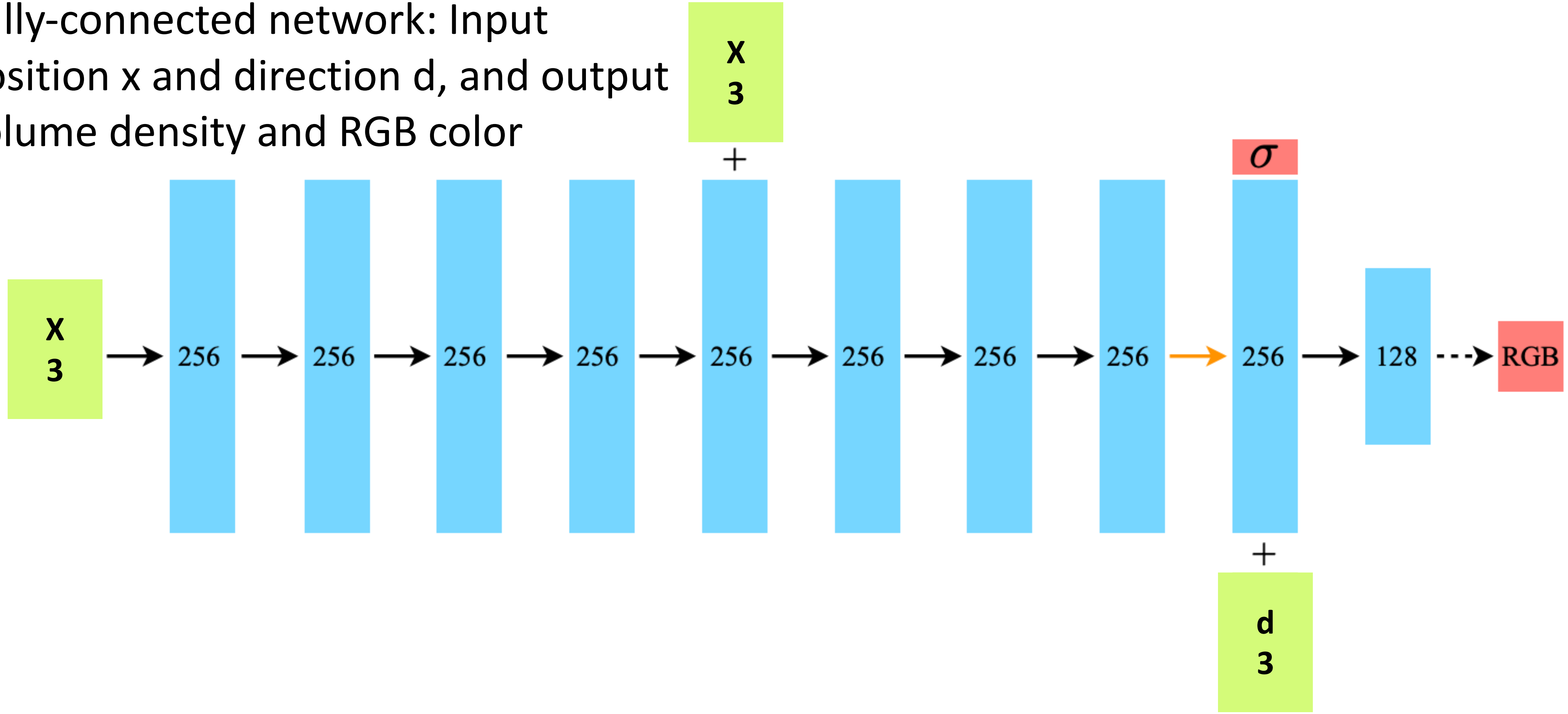
After training, can generate novel views of the scene by integrating along rays corresponding to new pixels

Parameterize each ray as origin plus direction: $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$
Volume Density is $\sigma(\boldsymbol{p}) \in [0,1]$
Color that a point **p** emits in direction **d** is $c(\boldsymbol{p}, \boldsymbol{d}) \in [0,1]^3$

Mildenhall et al, "Representing Scenes as Neural Radiance Fields for View Synthesis", ECCV 2020

Neural Radiance Fields (NeRF): Network Architecture

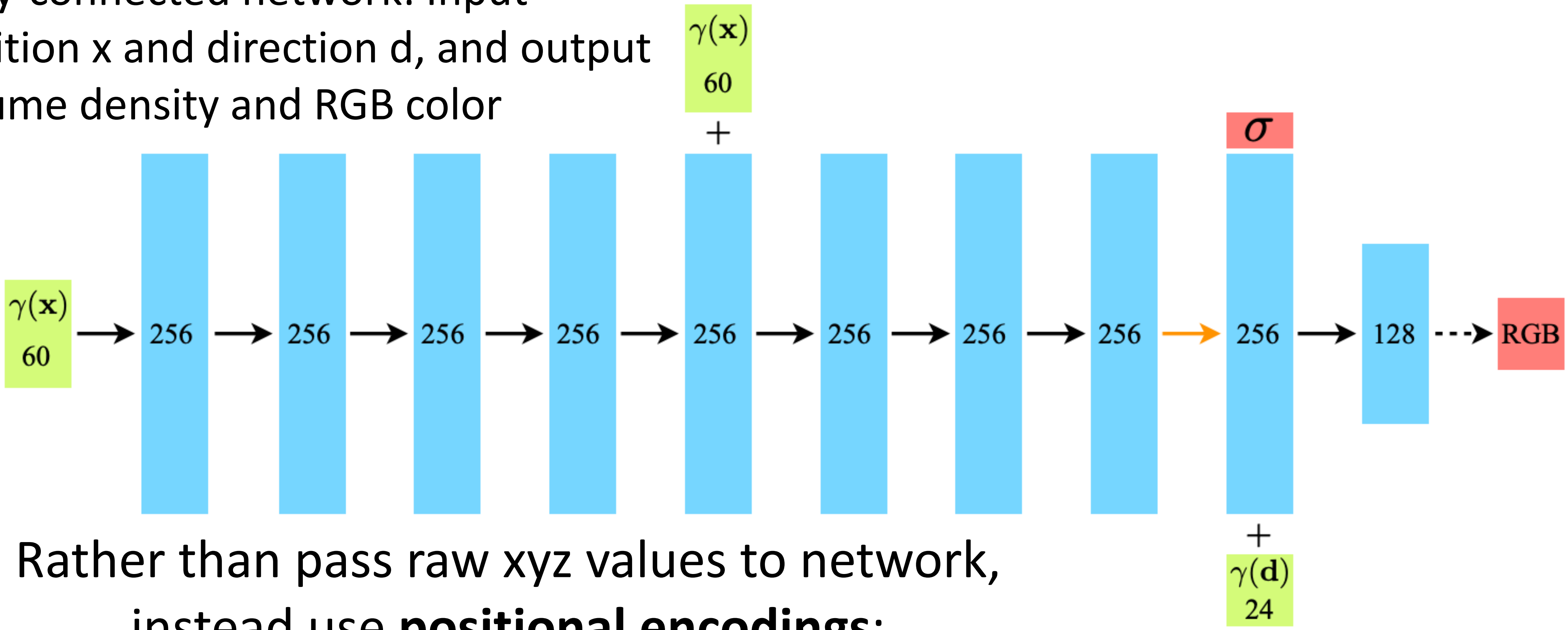
Fully-connected network: Input position x and direction d, and output volume density and RGB color



Mildenhall et al, "Representing Scenes as Neural Radiance Fields for View Synthesis", ECCV 2020

Neural Radiance Fields (NeRF): Network Architecture

Fully-connected network: Input position x and direction d, and output volume density and RGB color

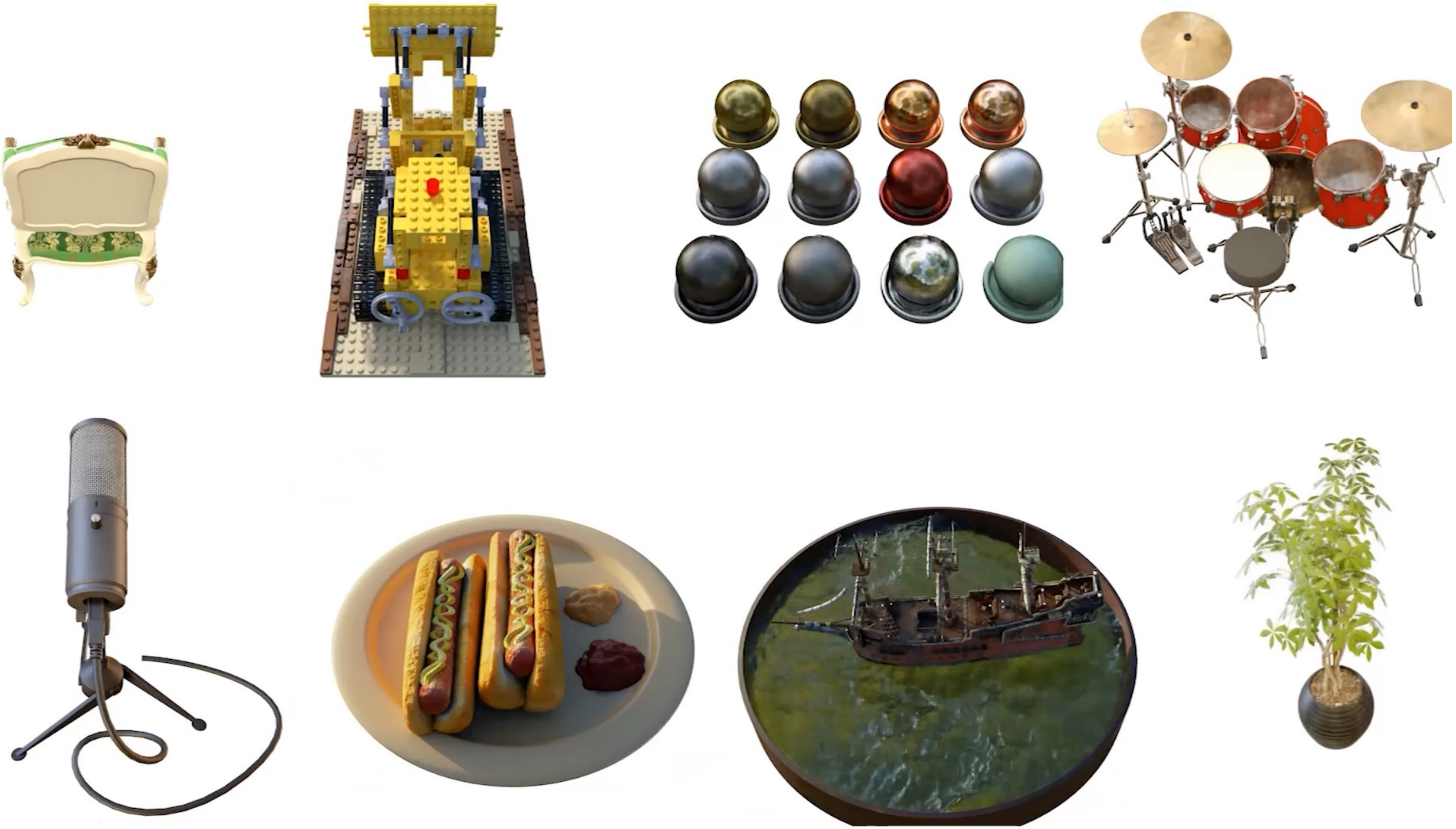


Rather than pass raw xyz values to network, instead use **positional encodings**:

$$\gamma(p) = \left(\sin(2^0\pi p), \cos(2^0\pi p), \ldots, \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p)\right)$$

Mildenhall et al, "Representing Scenes as Neural Radiance Fields for View Synthesis", ECCV 2020

Neural Radiance Fields: Very Strong Results!



Neural Radiance Fields: Very Strong Results!

Neural Radiance Fields: Very Strong Results!

Neural Radiance Fields: Can extract 3D geometry!

Neural Radiance Fields

Main Problem: Very slow!

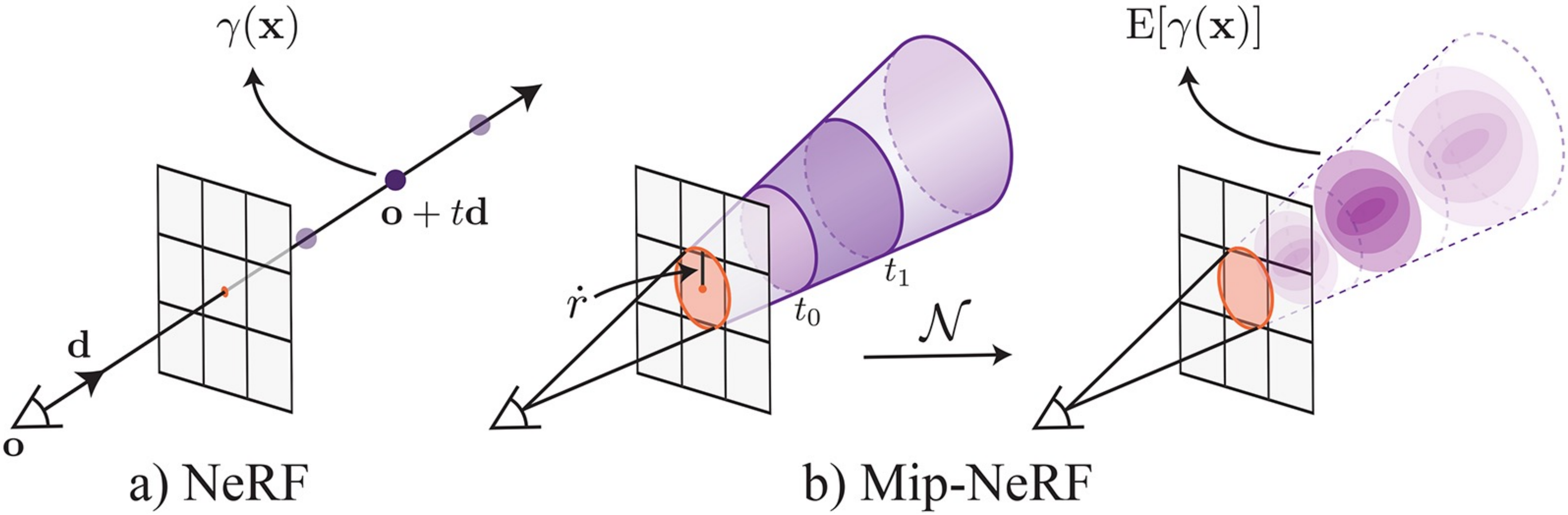
Training: 1-2 days on a V100 GPU, for just a single scene!

Inference: Sampling an image from a trained model:
(256 x 256 pixels) x (224 samples per pixel)
= 14.6M forward passes through MLP

Tons of follow-up work!

Mildenhall et al, “Representing Scenes as Neural Radiance Fields for View Synthesis”, ECCV 2020

Mip-NeRF: Model cones rather than rays



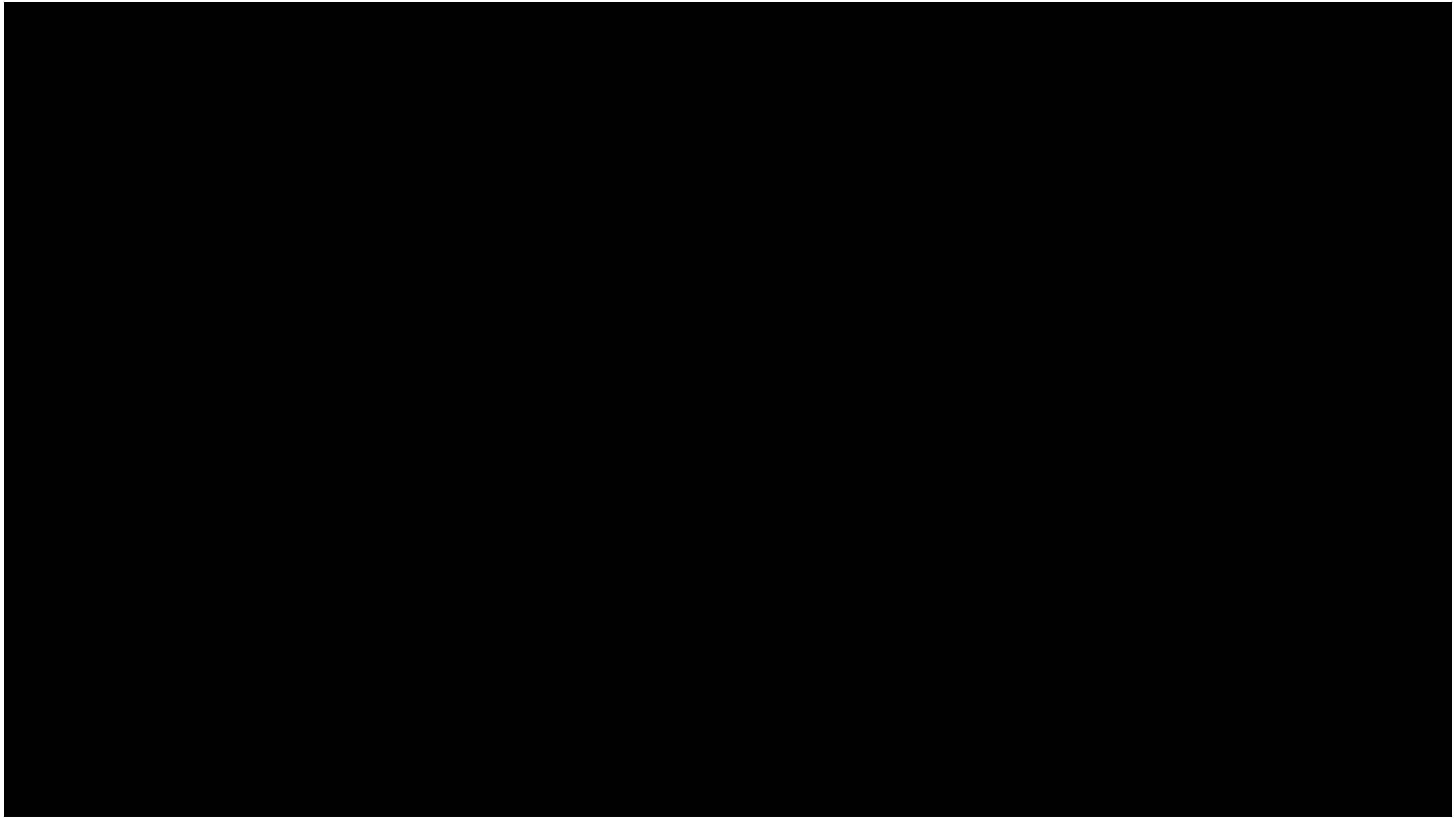
Barron et al, “Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields”, ICCV 2021

Dynamic NeRF: Deformable Scenes

(a) Capture Process (b) Input (c) Nerfie (d) Nerfie Depth

Park et al, "Nerfies: Deformable Neural Radiance Fields", ICCV 2021

RawNeRF: High-Dynamic Range Imagery



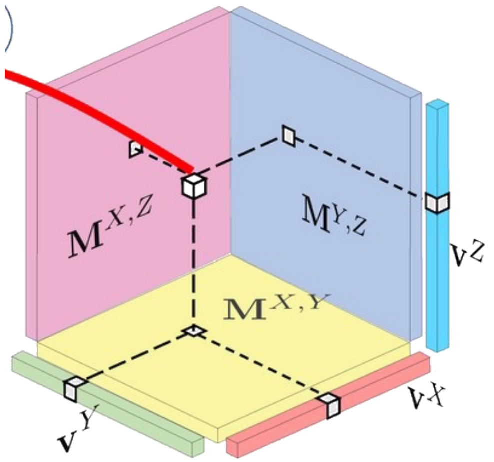
Mildenhall et al, “NeRF in the Dark: High Dynamic Range View Synthesis from Noisy Raw Images”, CVPR 2022

BlockNeRF: A Neighborhood of San Francisco

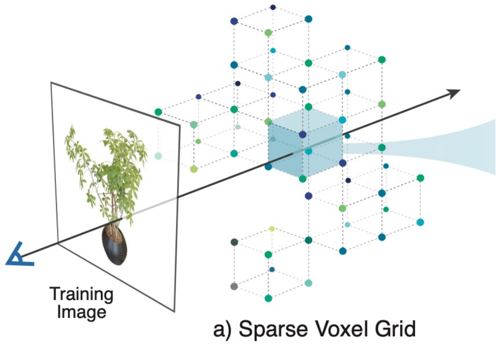
Tancik et al, “Block-NeRF: Scalable Large Scene Neural View Synthesis”, arXiv 2022

Training NeRF models in minutes!

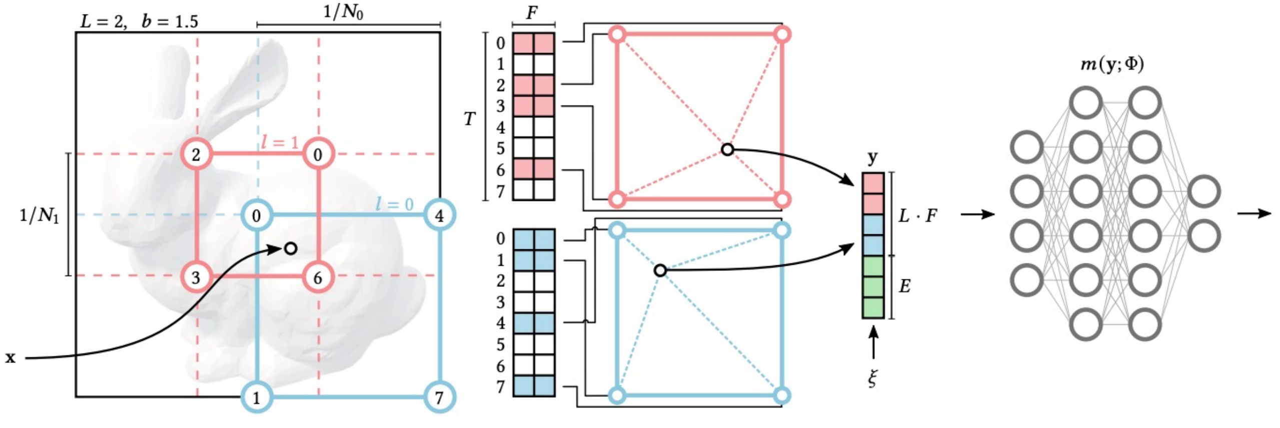
TensoRF



Plenoxels

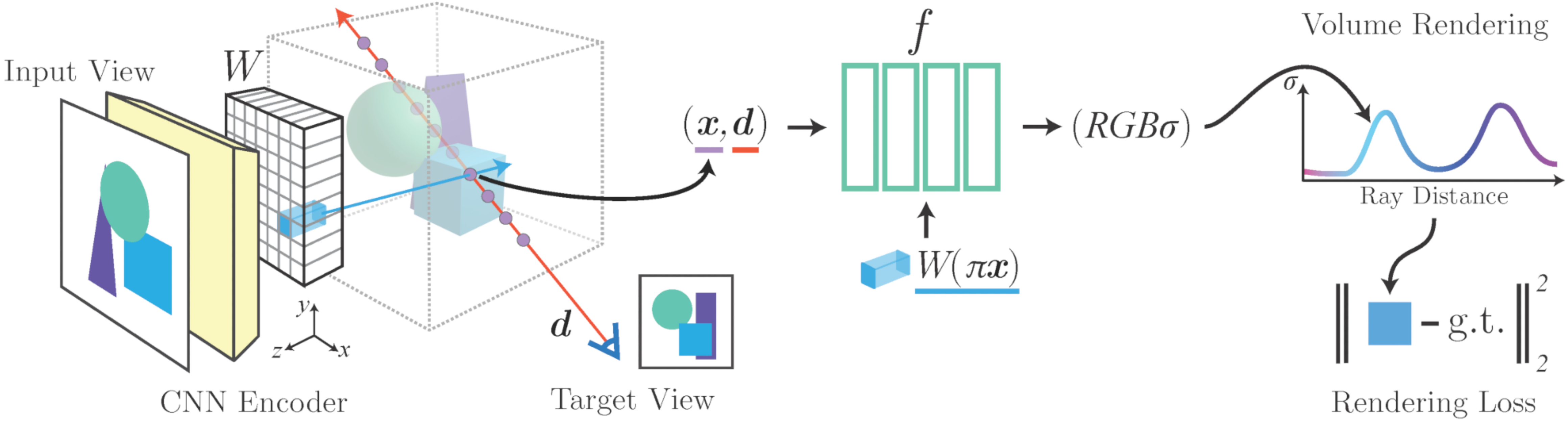


Instant-NGP



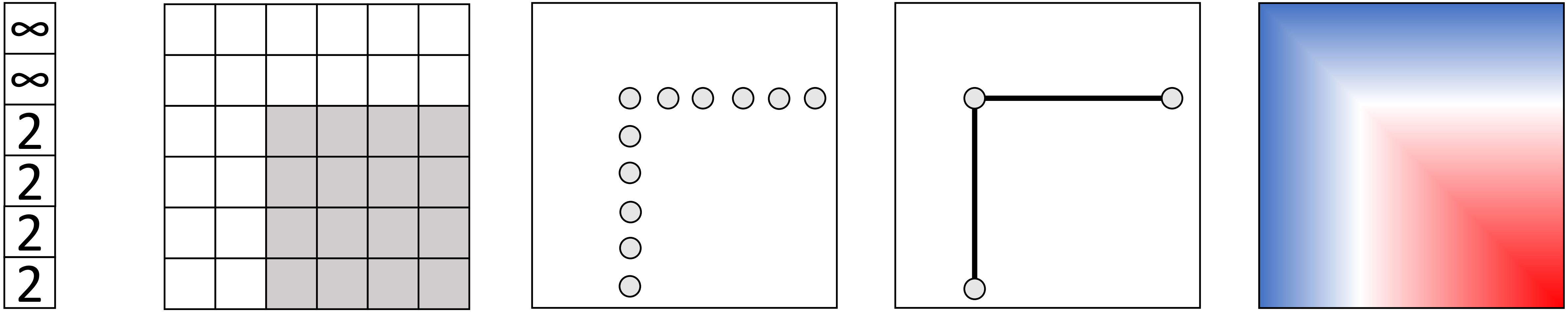
Yu et al, “Plenoxels: Radiance Fields without Neural Networks”, CVPR 2022
Muller et al, “Instant Neural Graphics Primitives with a Multiresolution Hash Encoding”, arXiv 2022
Chen et al, “TensoRF: Tensorial Radiance Fields”, arXiv 2022

Generalizable NeRF: Same model for many scenes



Yu et al, "pixelNeRF: Neural Radiance Fields from One or Few Images", CVPR 2021
Wang et al, "IBRNet: Learning Multi-View Image-Based Rendering", CVPR 2021

Summary: 3D Shape Representations



Depth Map | Voxel Grid | Pointcloud | Mesh | Implicit Surface

Summary: Neural Radiance Fields

Represent neural radiance fields with neural networks

Train using posed RGB images of a scene

Render novel views, extract 3D scene representations

One of the hottest topics in computer vision for past few years

Next Time:
Videos