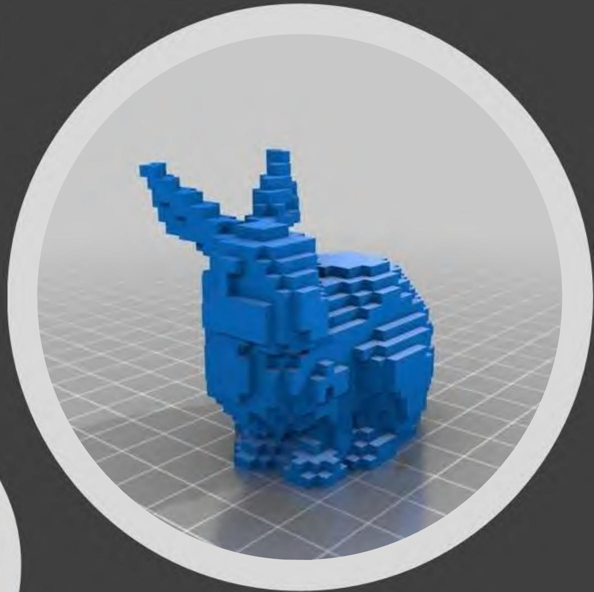
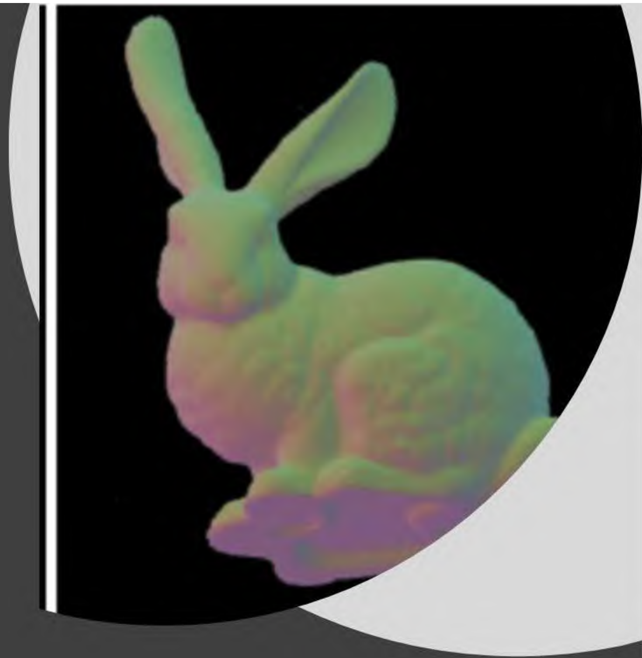
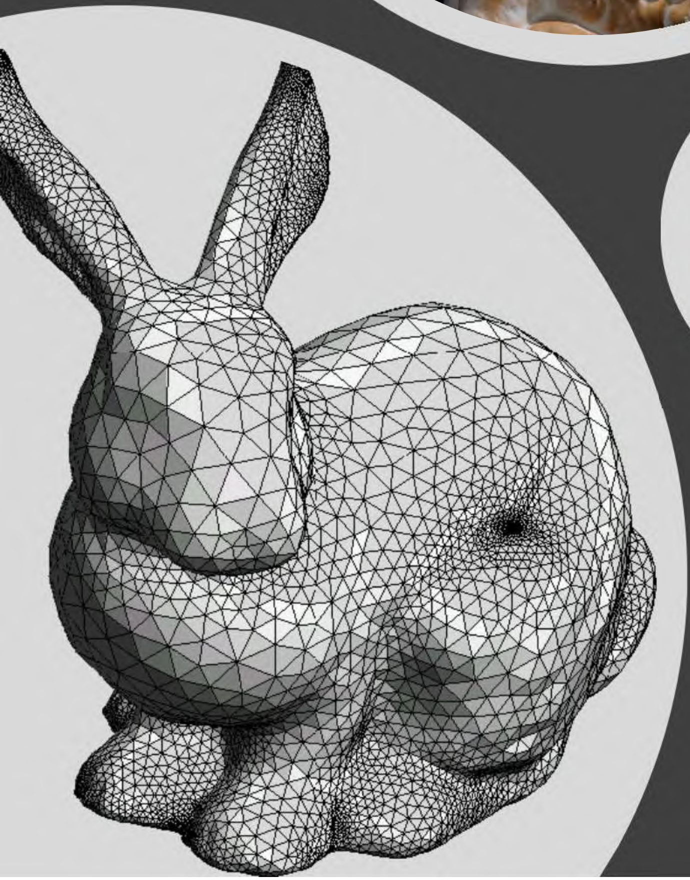
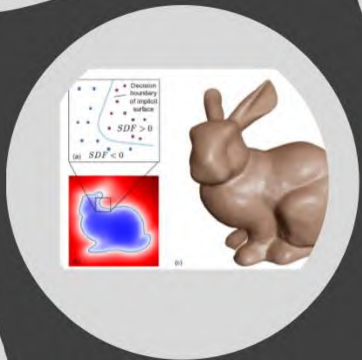
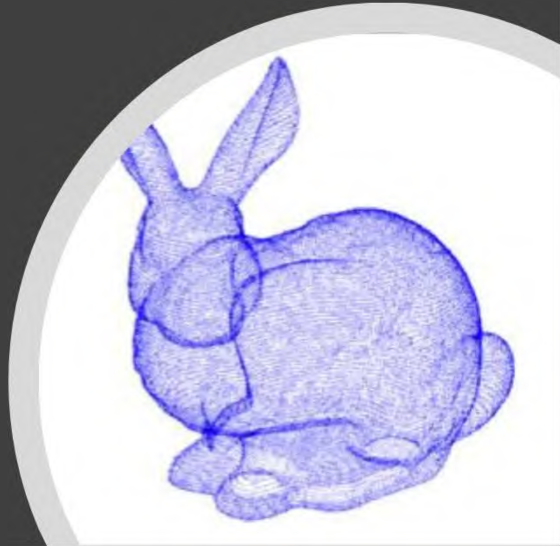




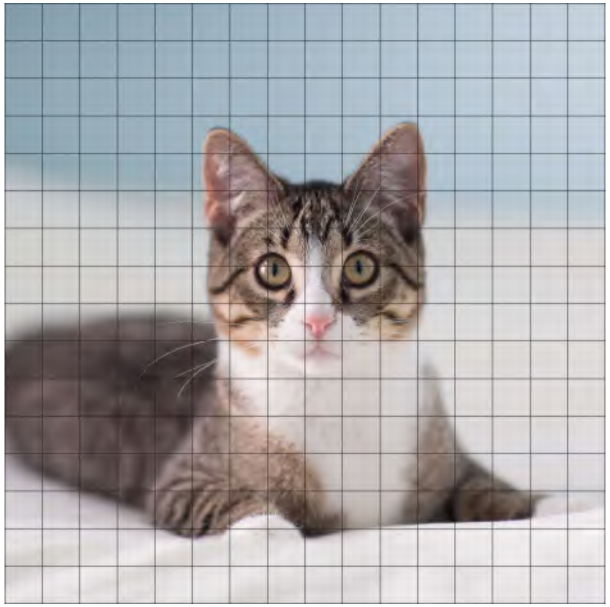




三维表示
3D Representations

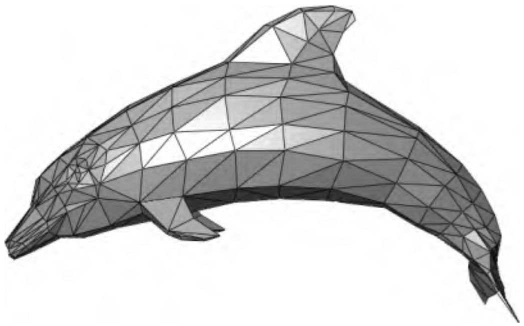


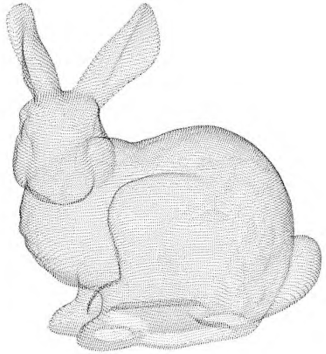
Representation is simple for 2D-land…

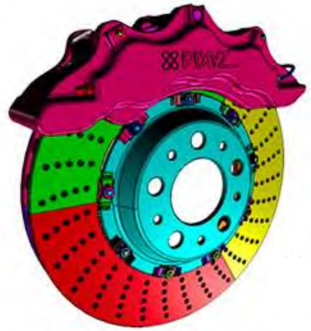


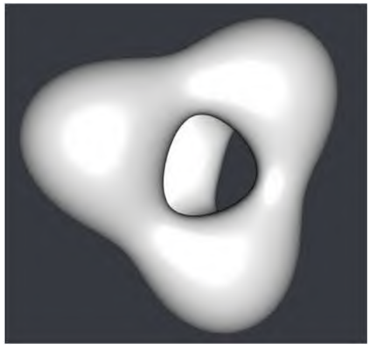
Unified representation across
applications, tools, and sensors

Meanwhile in 3D-land



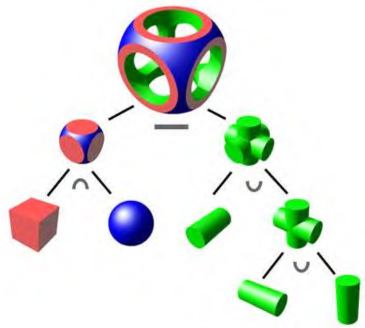






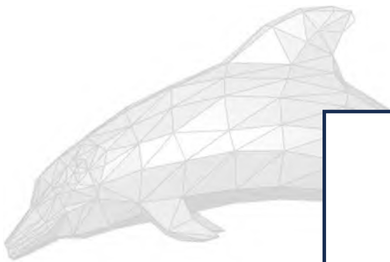
$$\{\mathbf{p} \mid f(\mathbf{p}) = 0\}$$

$$f(\mathbf{u}) = \mathbf{p} \in \mathbb{R}^3$$



and many more …

Meanwhile in 3D-land




What is the best representation?

It depends on applications, input, requirements

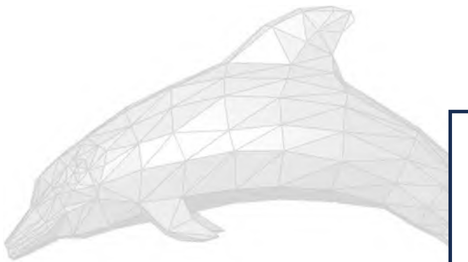
Reconstruction / Editing / Animation

y more …

$\{\mathbf{p} \mid f(\mathbf{p}) = 0\}$

$f(\mathbf{u}) = \mathbf{p} \in \mathbb{R}^3$

Meanwhile in 3D-land



Goals:

Understand different representations
(and basic implementation)

Limitations and Benefits

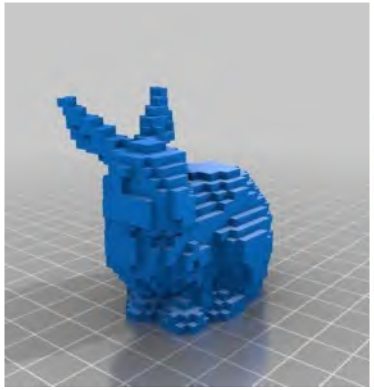
many more …

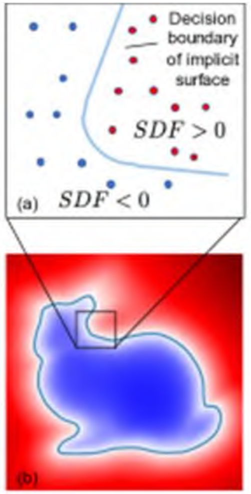
$\{\mathbf{p} \mid f(\mathbf{p}) = 0\}$

$f(\mathbf{u}) = \mathbf{p} \in \mathbb{R}^3$

Basic 3D representations

2.5D / Image Based Rendering

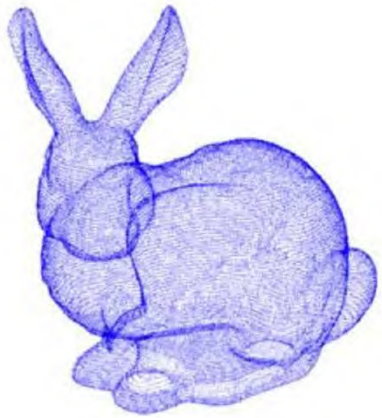




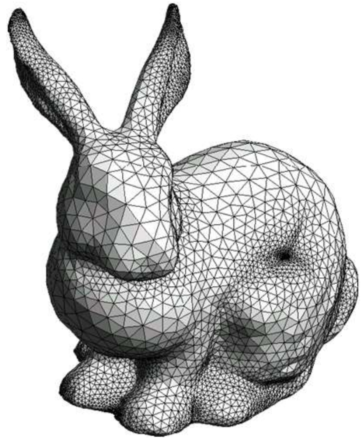
Volumetrics

Explicit

Implicit



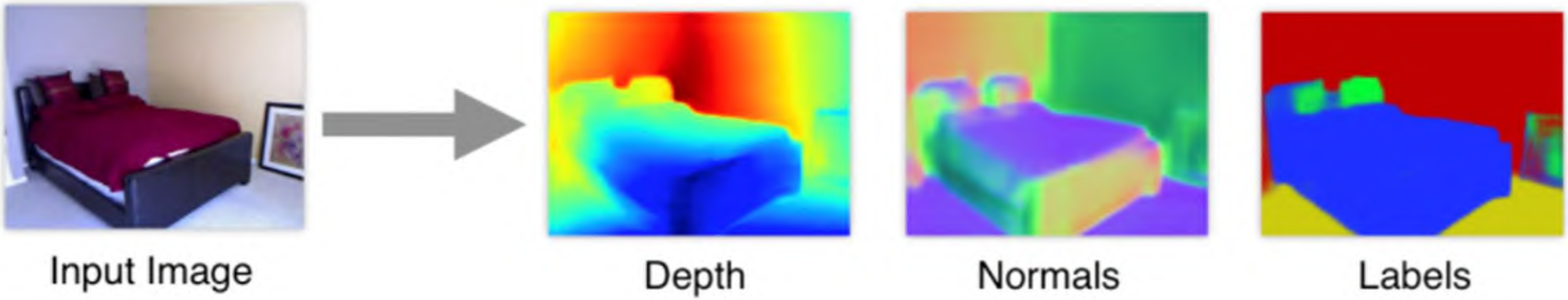
Point clouds



Meshes

2.5D — ex. Depth Maps

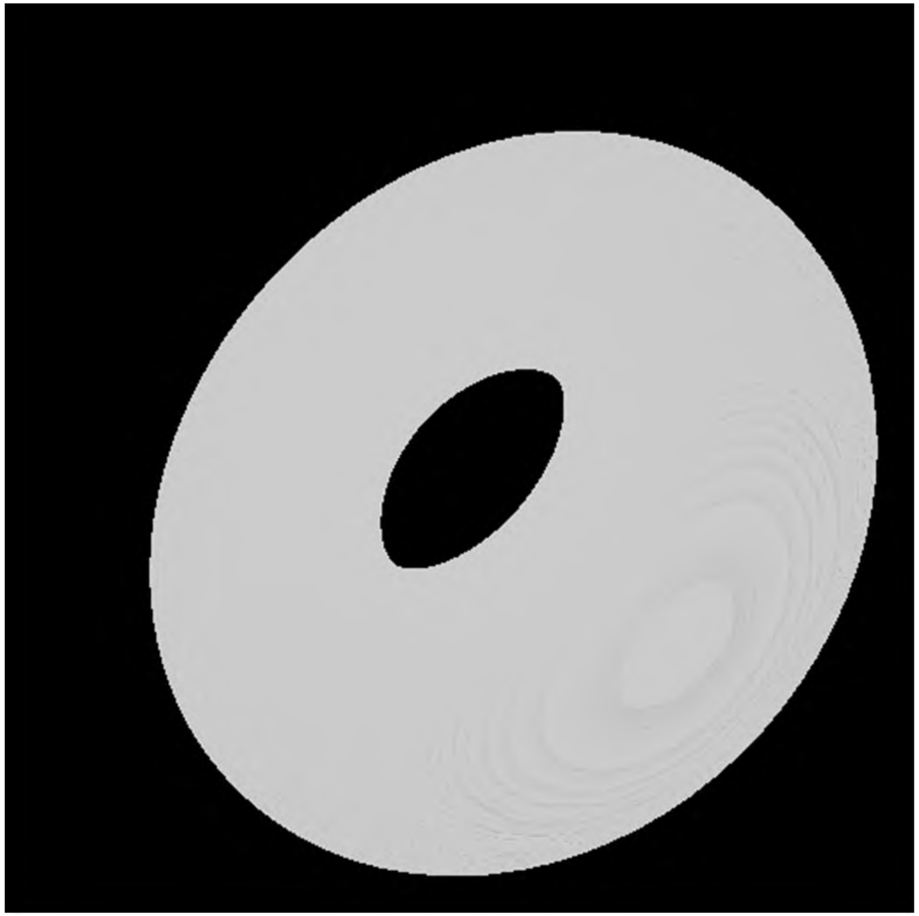
2.5D Representations



Eigen and Fergus ICCV '15

- Depth-map = **Per-pixel** depth or disparity value (H x W x 1)
- Normal map = per-pixel normal values (H x W x 3)
- Output is aligned with the input
- 2.5D = Not suitable to look around (no large baseline)
- Can think of each pixel as a colored 3D point

2.5D — Representing the Visible



Does not capture the ‘full’ 3D structure

Properties associated to the visible image pixels in the input view

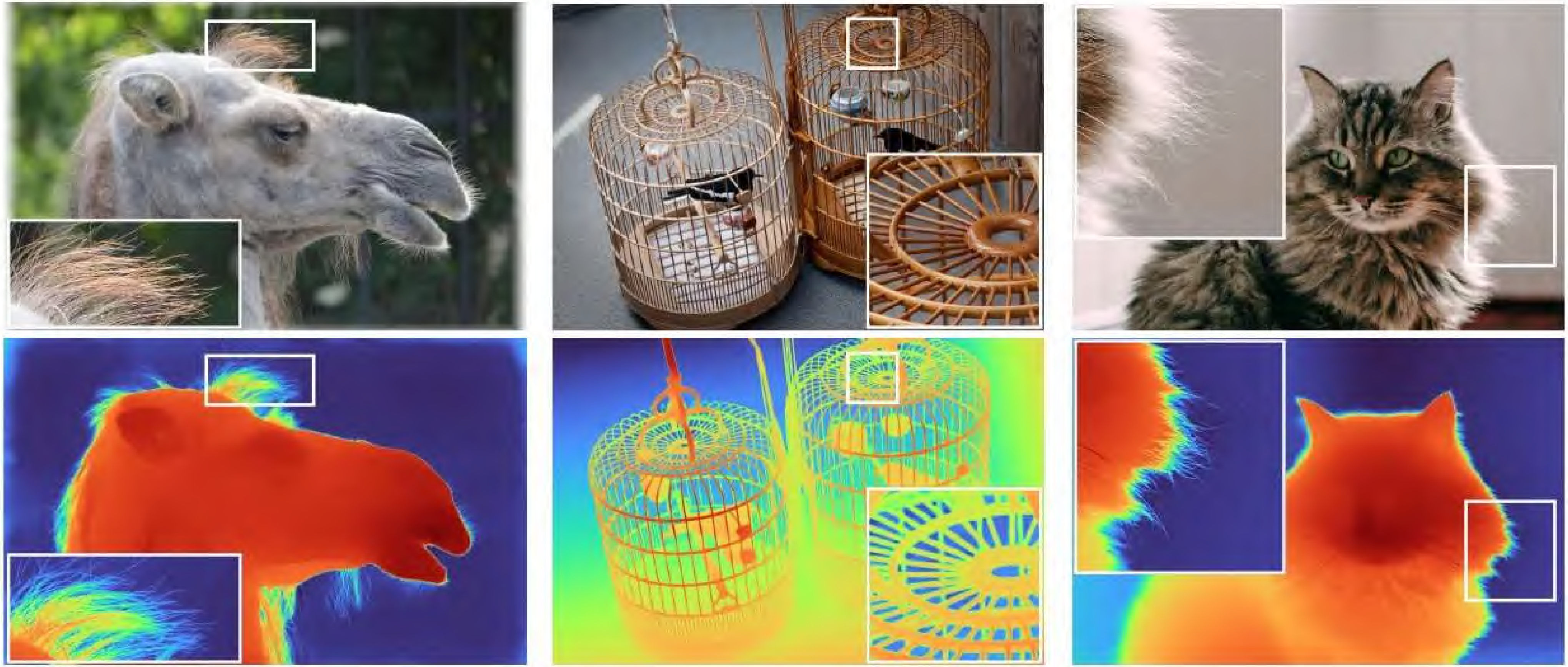
Easy way to obtain — Monocular Depth

- Predicted from a single image!
- Usually learn from many many image-depth pairs
- Predicts relative depth

$$(s, t) = \arg\min_{s,t} \sum_{i=1}^{M} (s\mathbf{d}_i + t - \mathbf{d}_i^*)^2$$

Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer. TPAMI 2022.

Even for sharp structures



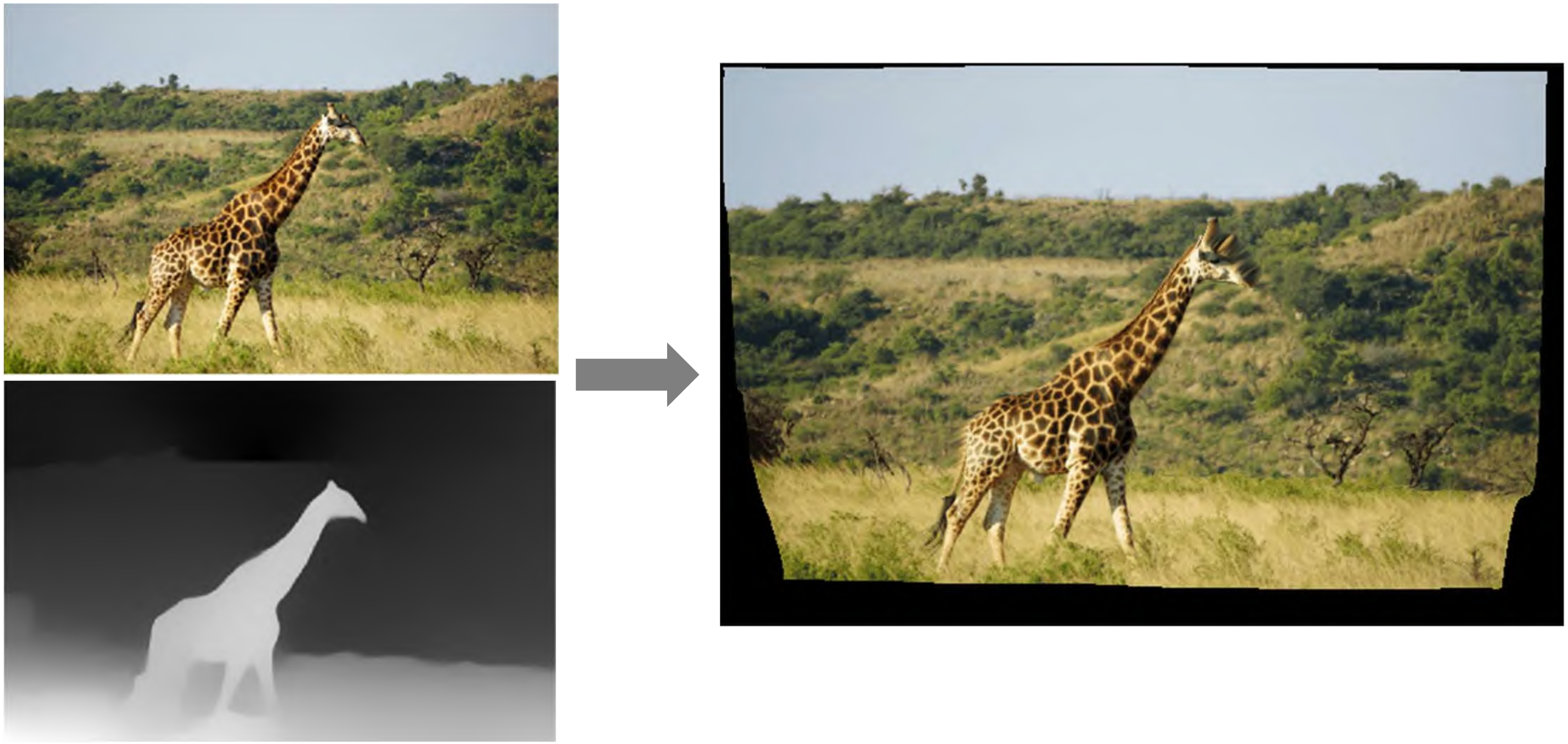
Depth Pro: Sharp Monocular Metric Depth in Less Than a Second. ArXiv 2024.

How to render?

Treat it as Colored 3D Points

All monocular depth works can’t produce perfect depth
But still useful for some applications

3D Photo: View synthesis from single image



3D Photo: View synthesis from single image

Synthesize the missing regions!

3D Photo via inpainting missing regions

Niklaus et al. ToG 2019

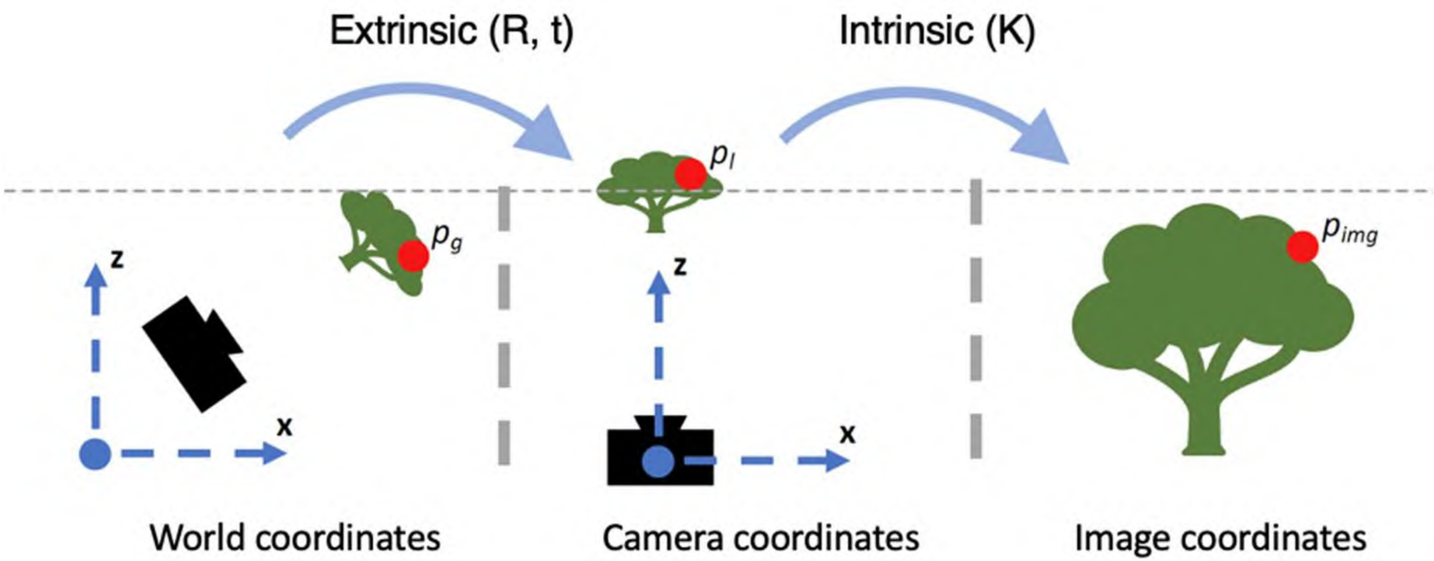
Shih et al. CVPR 2020

Inpaint RGB and D & Repeat ➔Infinite Nature Perpetual View Generation

Infinite Nature [Liu et al.] ICCV 2021

Recall: Depth -> 3D Point

- Lift the 2D plane to 3D
- Inverse the projection process (the last class)

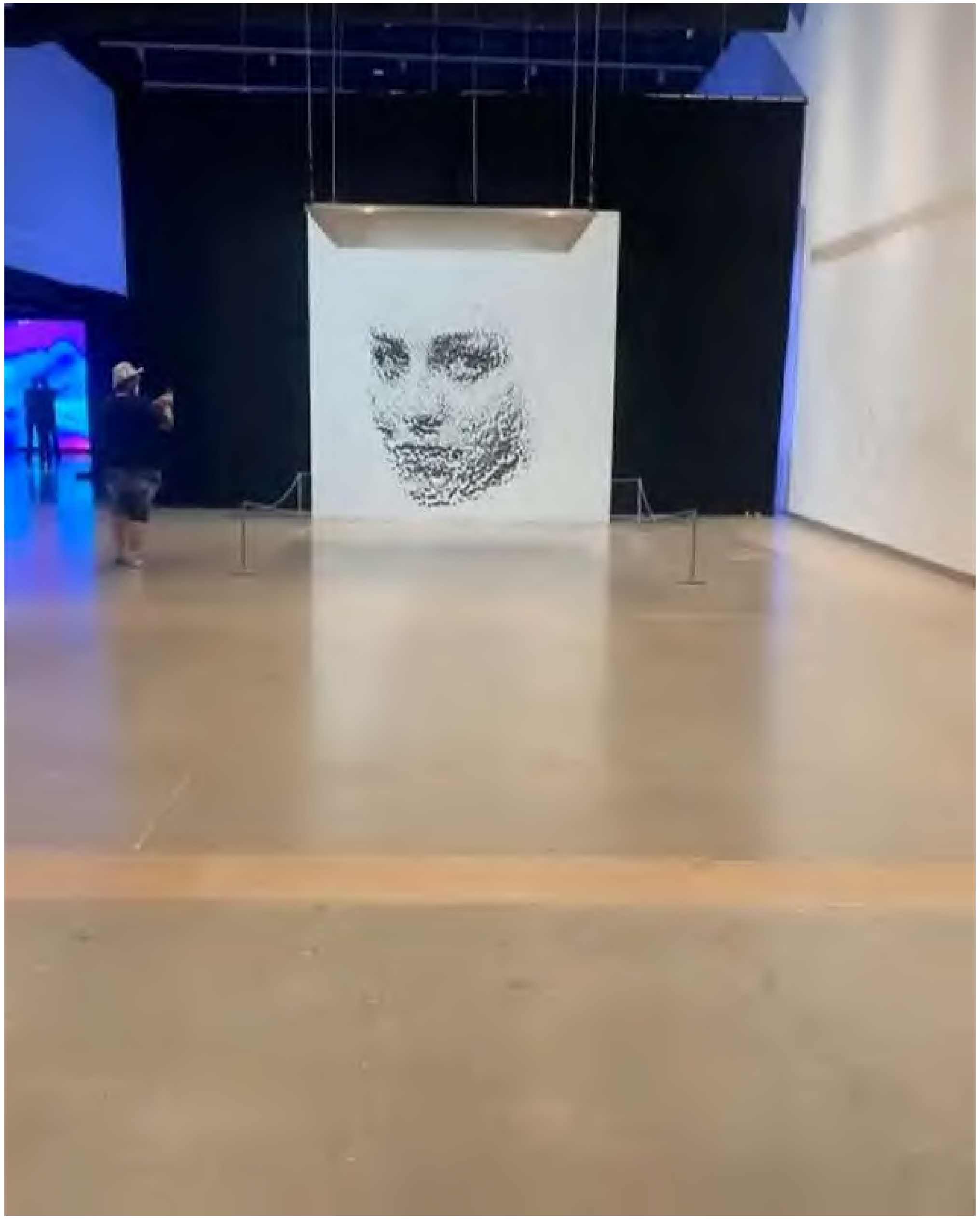


Point Clouds (PCL)

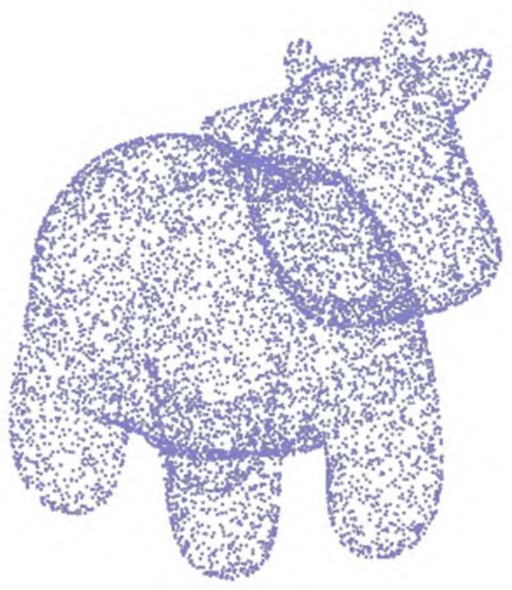
A basic point cloud = $\{(x_i, y_i, z_i),\ i \in [1, n]\}$
Can have other attributes (color, normal, …)

Obtained from

- Depth images / Lidar
- Single Image (with deep learning)
- Multiple RGB Images
- Scanner outputs
- Converted from a 3D mesh



Point Clouds

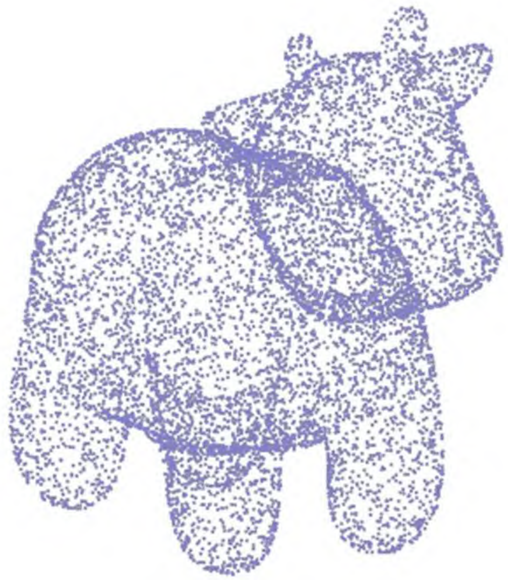


$$\xrightarrow{\text{Permutation } \sigma}$$

$$\{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_N\}$$

$$\{\mathbf{p}_{\sigma(1)}, \mathbf{p}_{\sigma(2)}, \cdots, \mathbf{p}_{\sigma(N)}\}$$

Point Clouds



$$\{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_N\}$$

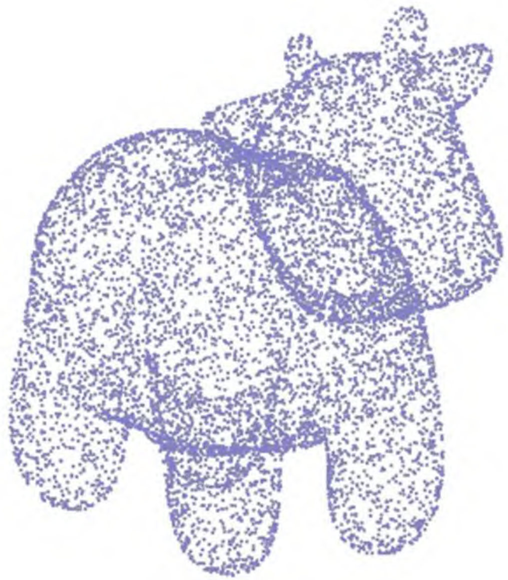
Unordered set of points

x1,y1,z1
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

Often represented as a NX3 array, but ordering does **not** matter (unlike images)

Need processing/generation methods that are permutation invariant (e.g. fully connected layer will not work)

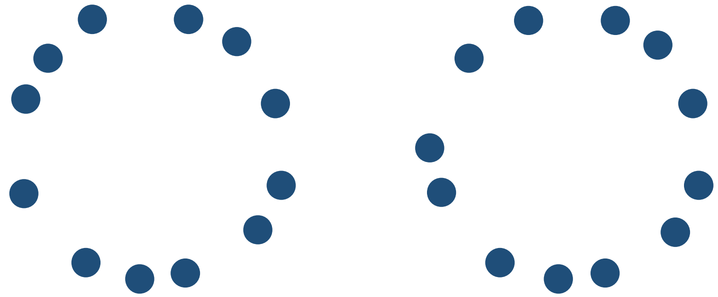
Point Clouds



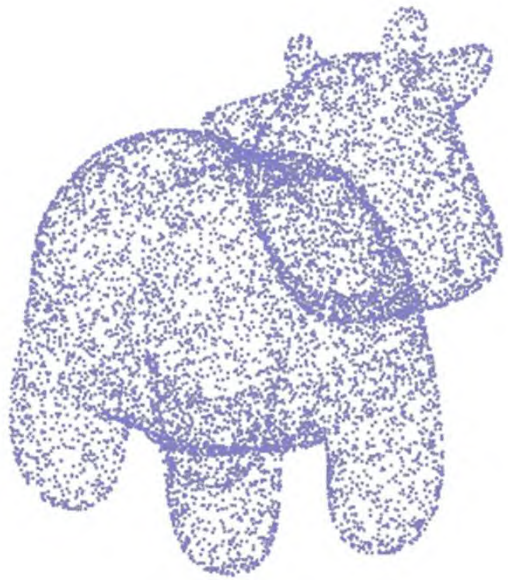
$$\{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_N\}$$

Unordered set of points

No explicit 'connectivity' information



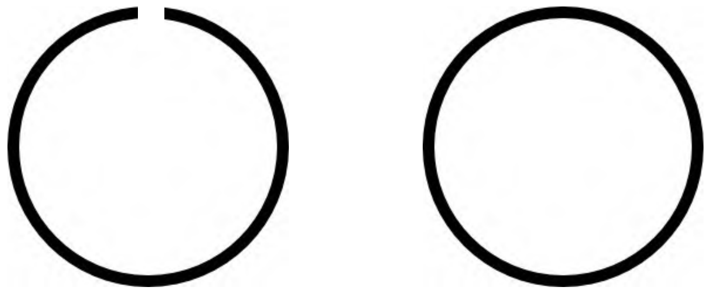
Point Clouds



$$\{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_N\}$$

Unordered set of points

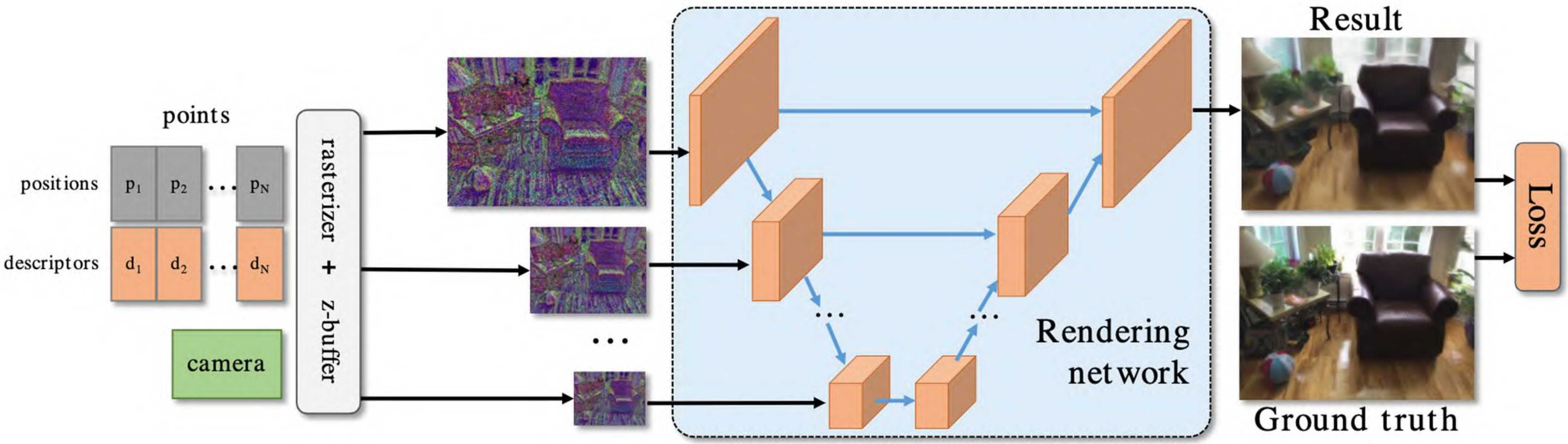
No explicit 'connectivity' information



(So it's more efficient to add edges (connectivity))


Dissolving Realities Vietnam #2 - Point Cloud Unity shader
^向上拉以精确地调整播放进度
0:00
0:00 / 0:59
Render (Colored) Point Clouds in Unity

Neural Rendering Point Clouds

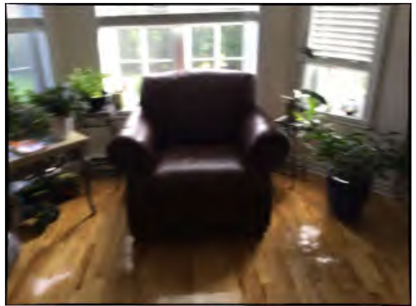


Setup: View-synthesis from available views
Memorize one specific scene

Neural Point Based Graphics, Aliev et al. ECCV 2020

Neural Rendering Point Clouds

Nearest available view

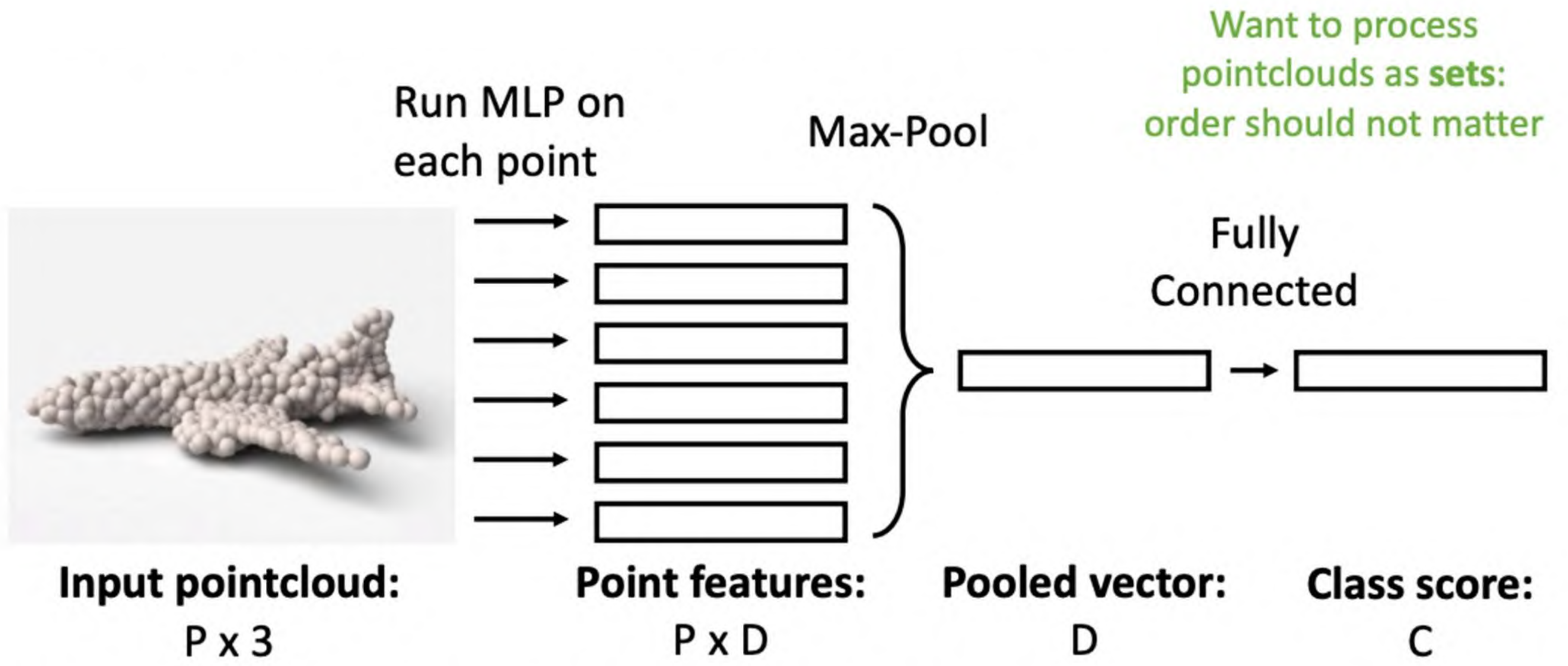


Pointcloud w/ color rendered

After neural rendering

Neural Point Based Graphics, Aliev et al. ECCV 2020

Neural Processing Point Clouds



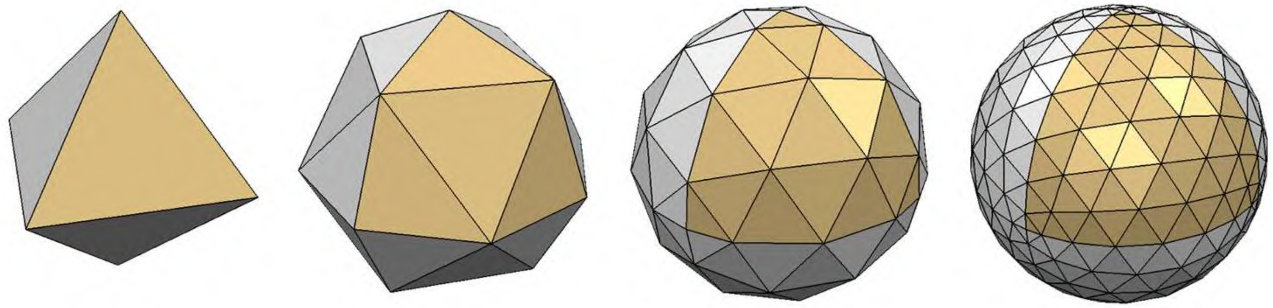
Qi et al, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", CVPR 2017
Qi et al, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space", NeurIPS 2017

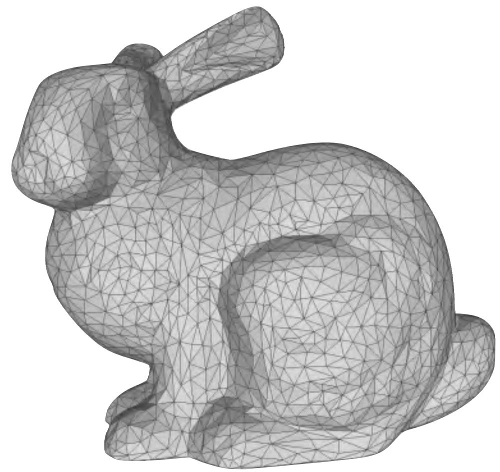
Meshes: Connected Point Clouds

- Point clouds are order-less set of points
- Permutation invariant
- Meshes vertices are ordered and connected

Polygon Meshes

- A mesh is a set of vertices with faces that defines the topology
- Mesh = {Vertices, Faces}
  - Vertices: N x 3
  - Faces: F x {3, 4, …} specifying the edges of a polygon
  - Triangle faces most common but tetrahedrons (tets) are also.
- Surface is explicitly modeled by the faces
- Most common modeling representation





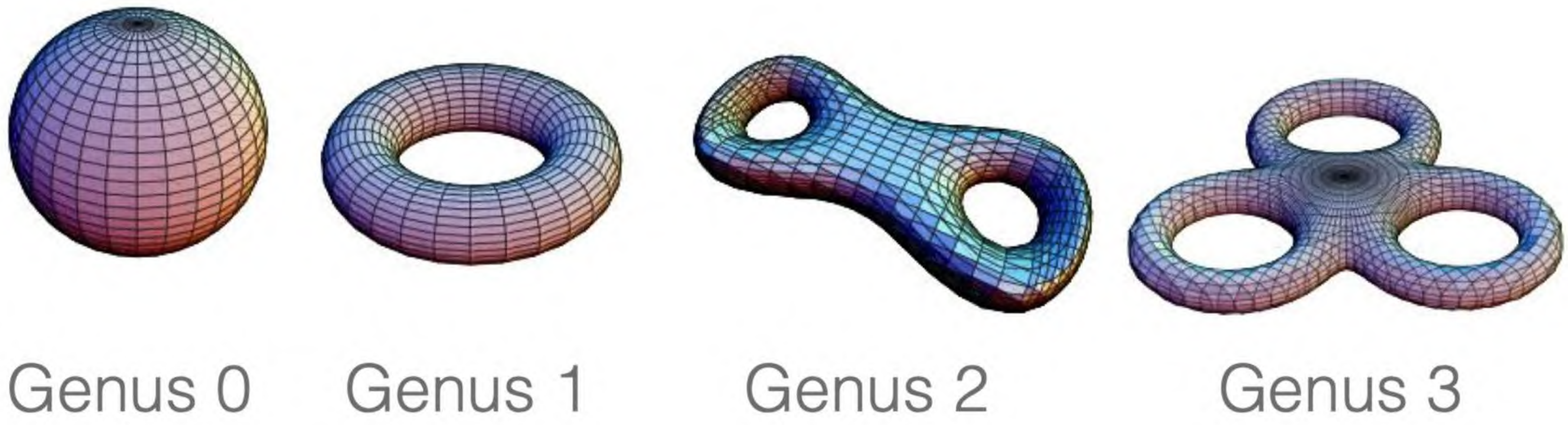
Topology: Genus

Makes it hard to *directly* predict meshes of arbitrary objects from images

Informally, the number of **holes** or **handles**

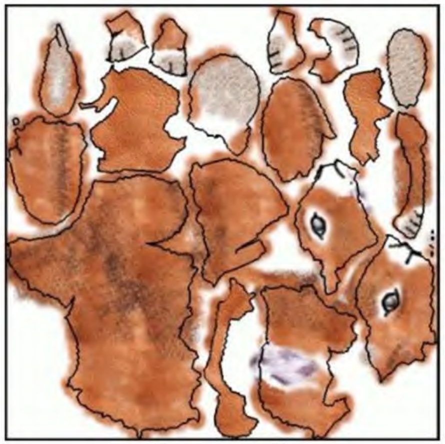
Meshes can represent arbitrary topology.

BUT two surfaces with different genuses are not homeomorphic (can't be transformed without cutting / gluing)



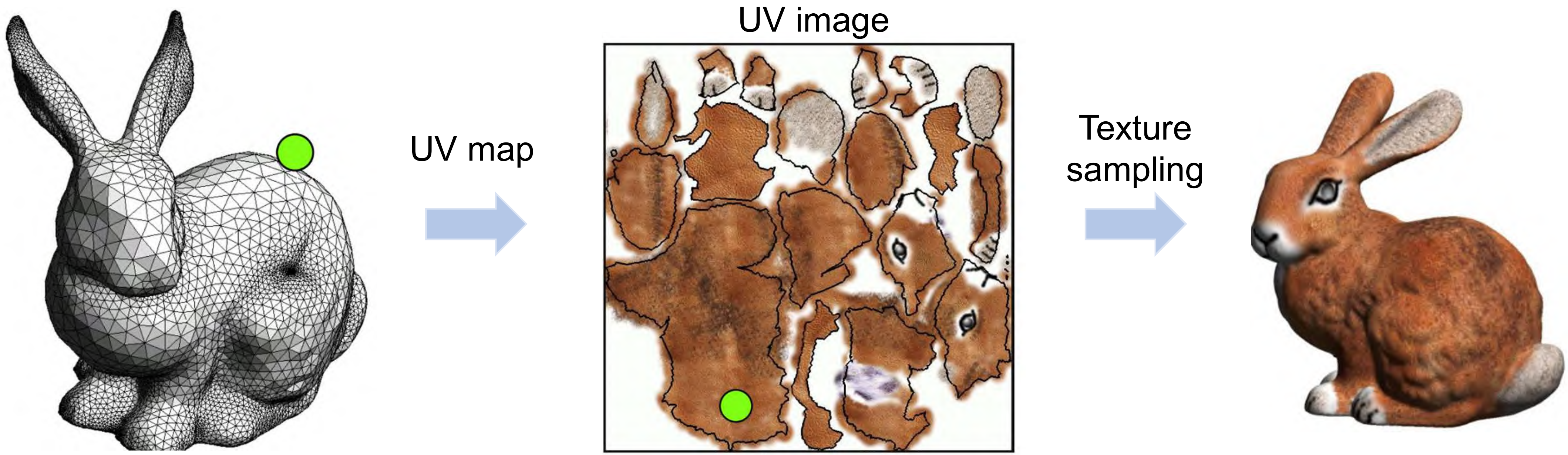
Meshes are great for texturing

UV Image

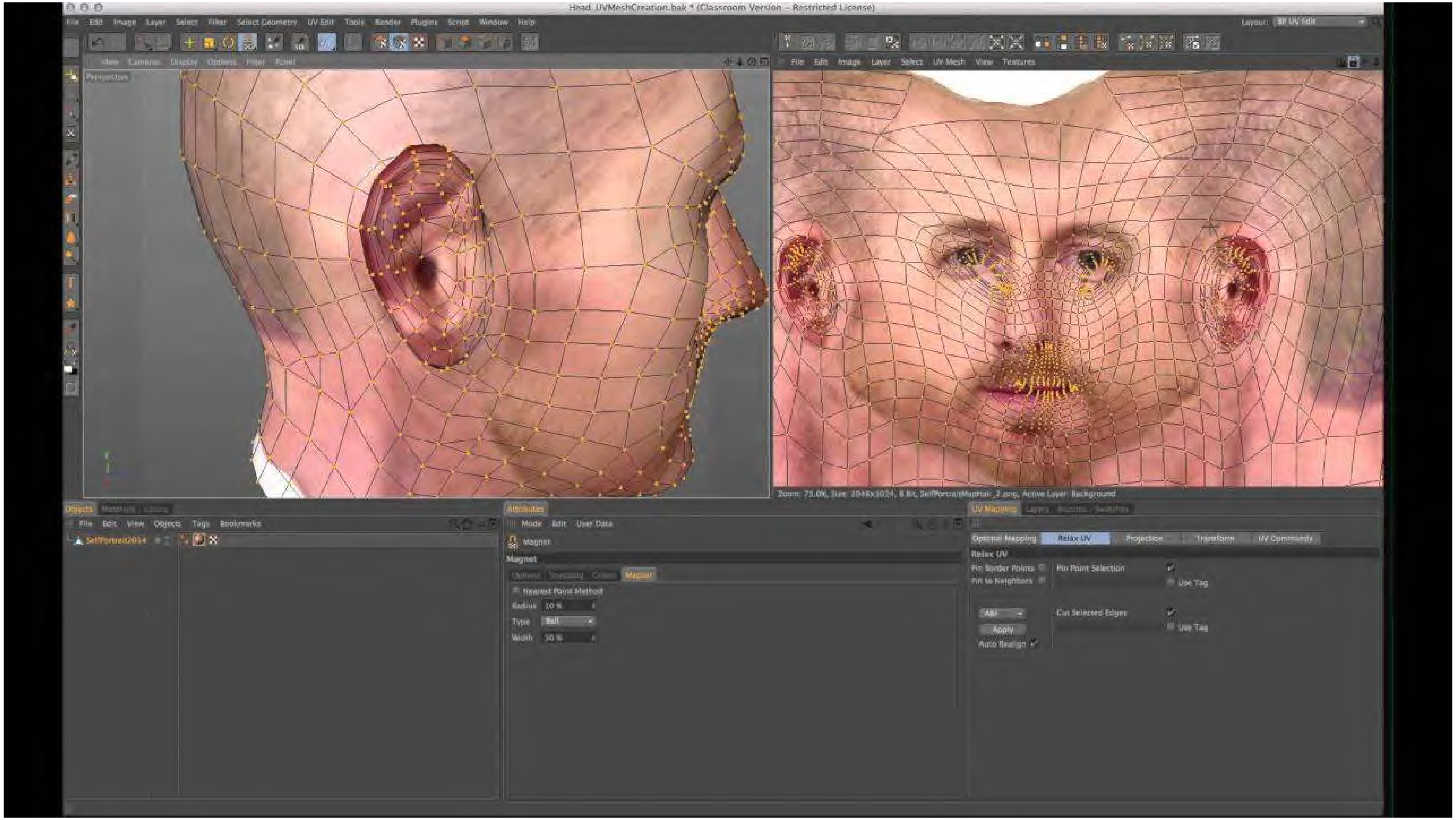


Every single vertex has a UV coordinate

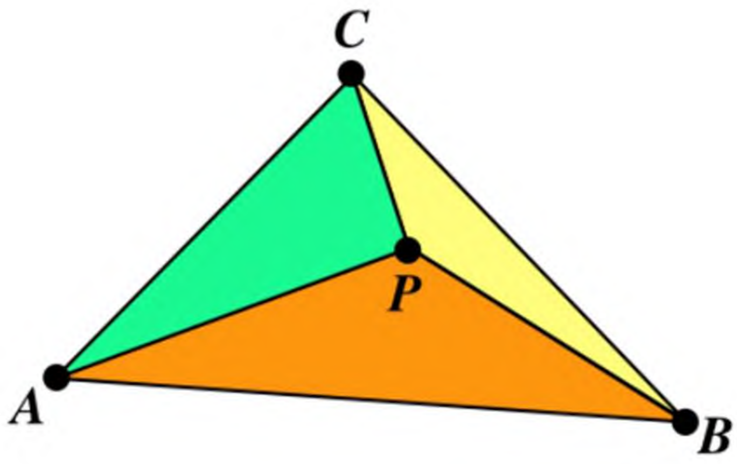
- Defined by UV mapping : (x,y,z) → (u,v)
- “texture coordinates”



Benefit of UV texturing: Continuous Color



Barycentric coordinates to get UV Coordinates



$$P = w_A \times A + w_B \times B + w_C \times C$$

$$w_A = \frac{\Delta PBC}{\Delta ABC} =$$

$$w_B = \frac{\Delta PCA}{\Delta ABC} =$$

$$w_C = \frac{\Delta PAB}{\Delta ABC} =$$

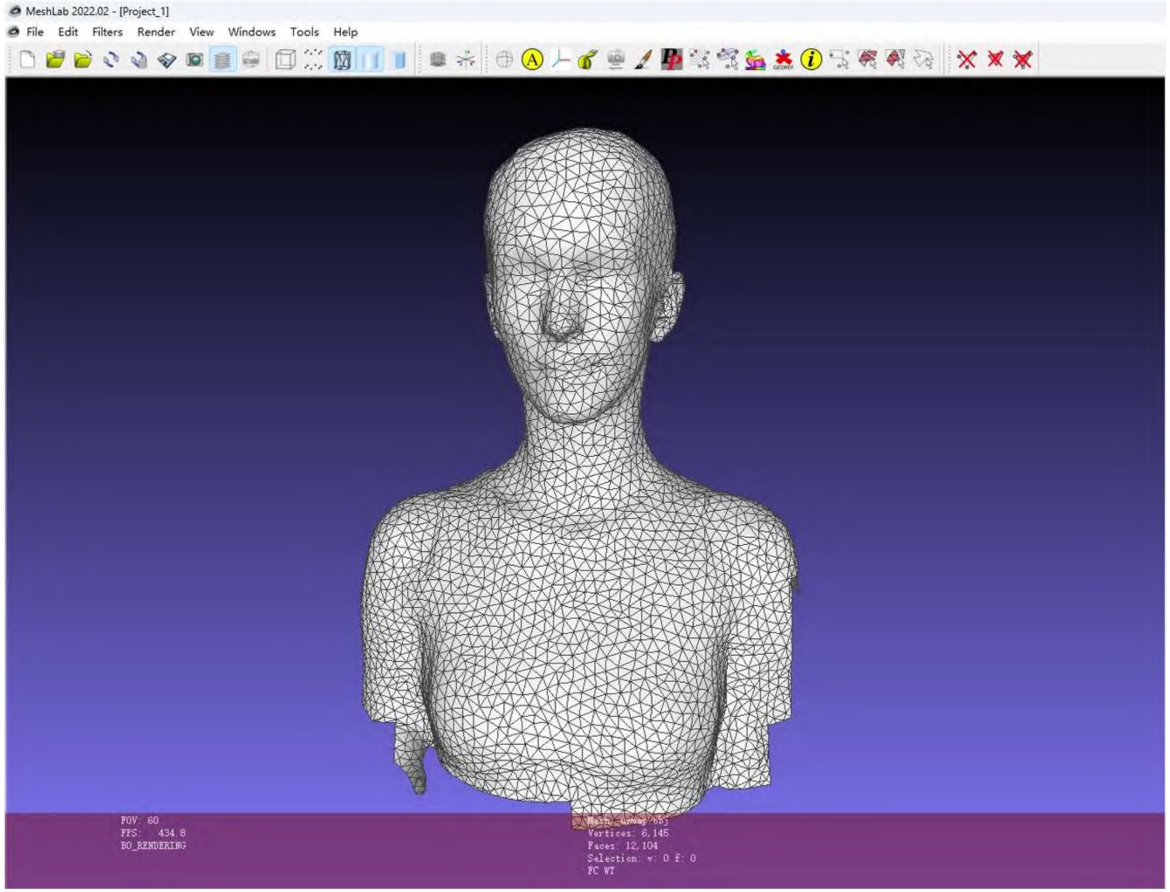
inside condition

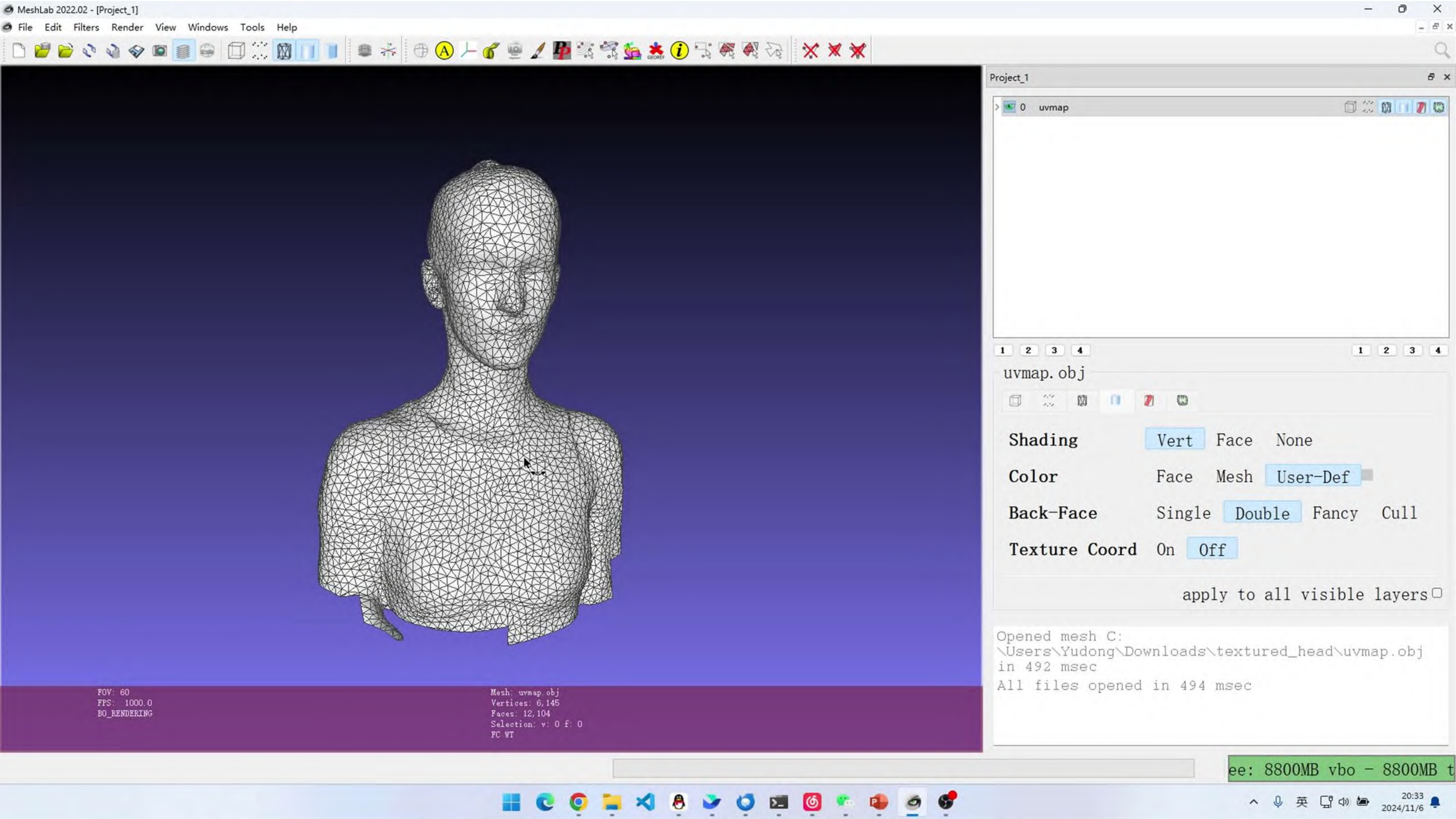
$0 \le w_A, w_B, w_C \le 1 \qquad w_A + w_B + w_C = 1$

Texturing Process

- Precompute the UV mapping
- Sample X many points on each triangle
- Figure out their UV coordinates (compute once)
- Get a UV image
- Sample the UV image

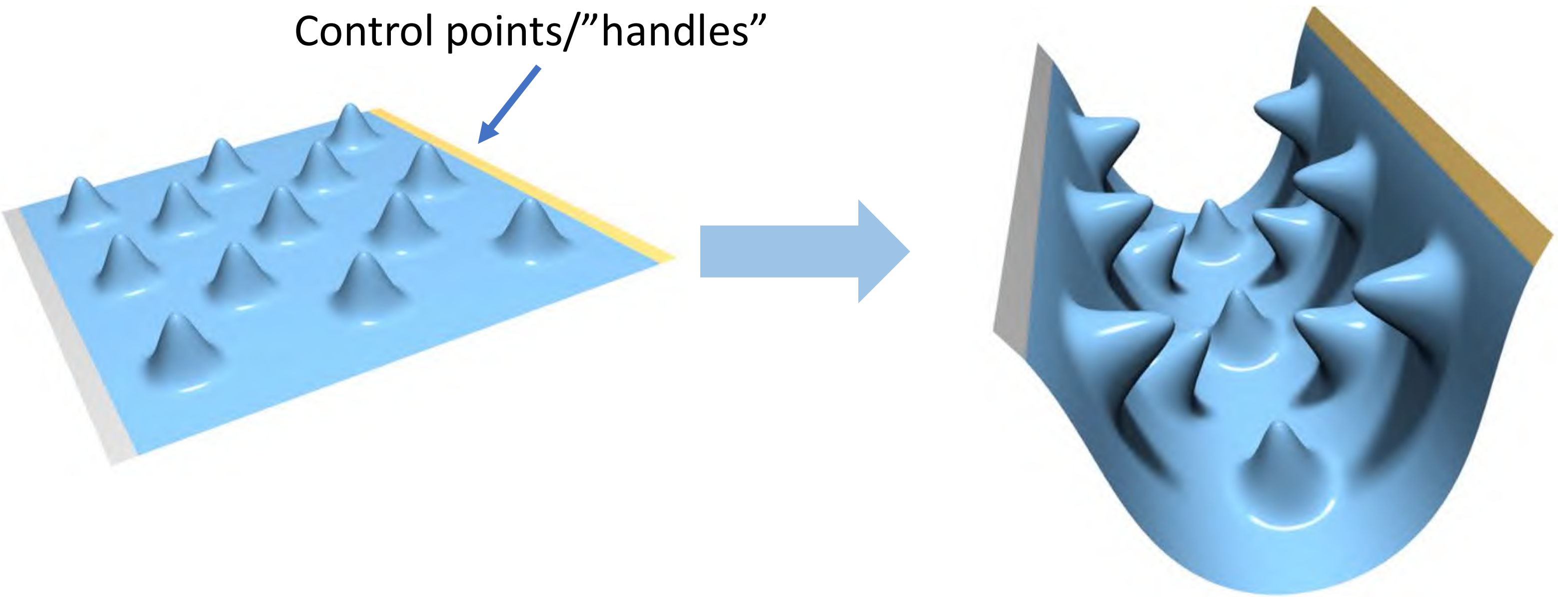
Easy For Rendering




MeshLab 2022.02 - [Project_1]
File Edit Filters Render View Windows Tools Help
Project_1
0 uvmap
uvmap.obj
Shading Vert Face None
Color Face Mesh User-Def
Back-Face Single Double Fancy Cull
Texture Coord On Off
apply to all visible layers
Opened mesh C:\Users\Yudong\Downloads\textured_head\uvmap.obj in 492 msec
All files opened in 494 msec
FOV: 60
FPS: 1000.0
BO_RENDERING
Mesh: uvmap.obj
Vertices: 6,145
Faces: 12,104
Selection: v: 0 f: 0
FC WT
ee: 8800MB vbo - 8800MB t
20:33
2024/11/6

Good For Deformation

- For: shape/character editing, sculpting, modelling
- Problem: Deform some 3D representation given some target



Good For Animation

Very Easy For Using

GUI Software

UNREAL ENGINE

blender

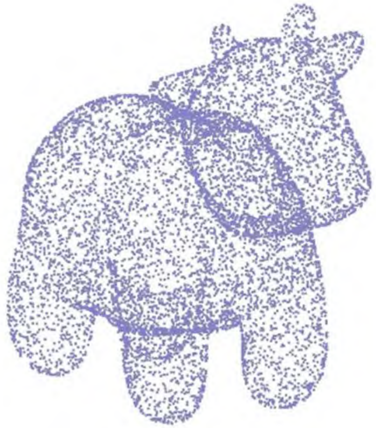
PyTorch3D

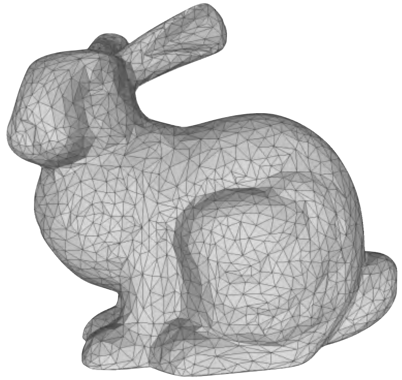
MeshLab

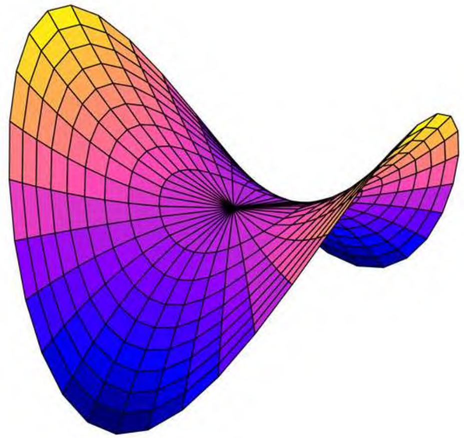
OpenGL

Taichi
Programming Language

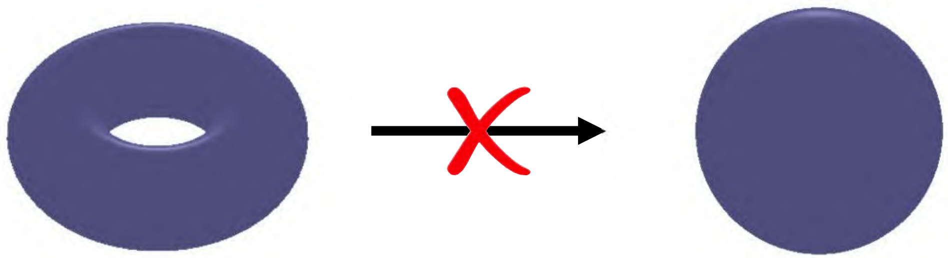
Coding Library

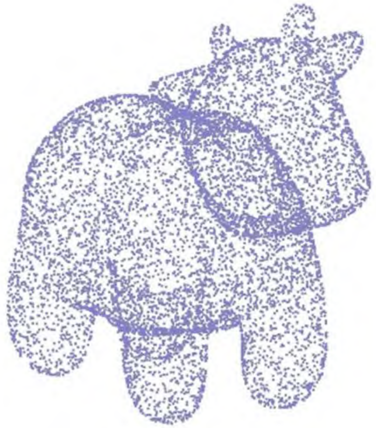


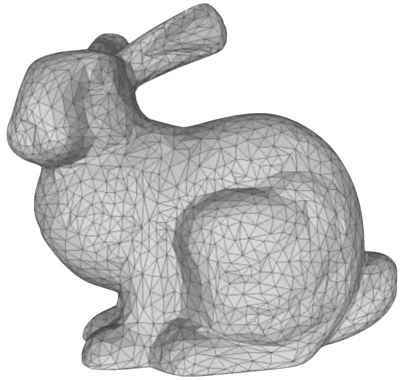


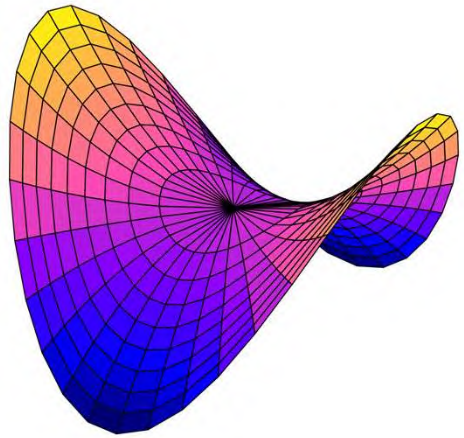


Surface Representations

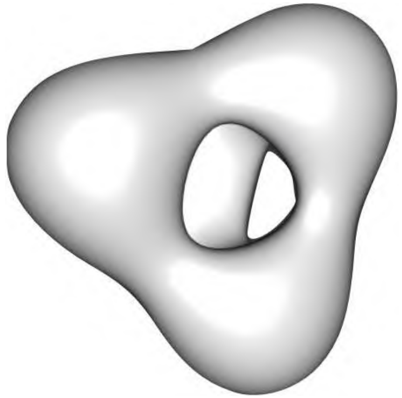








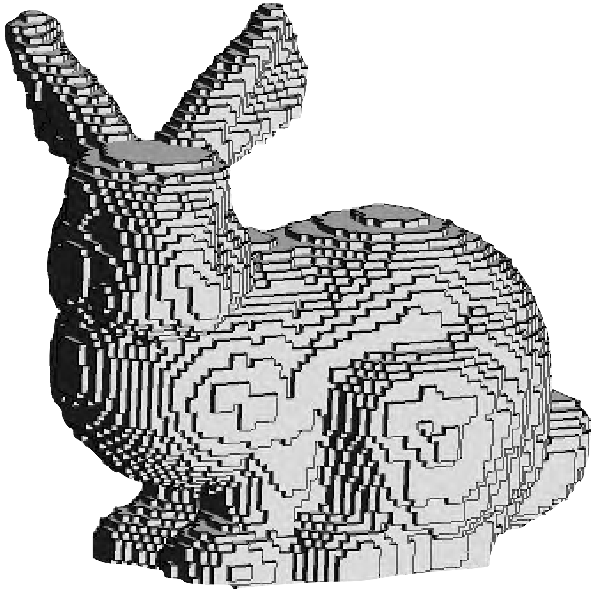
Surface Representations



Volume Representations

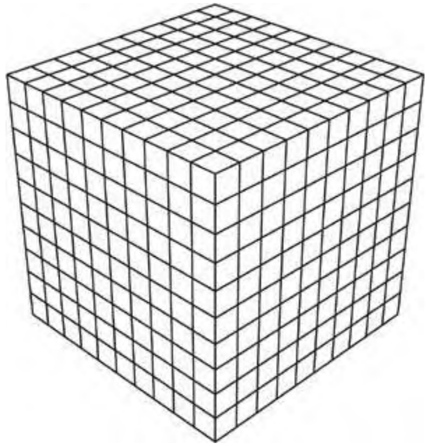
Volumetric Representations

So far we talked about points, lines, and surfaces
Volumetric representations model the entire space
Can be explicit & implicit



Voxels

Simple, but very memory expensive!!



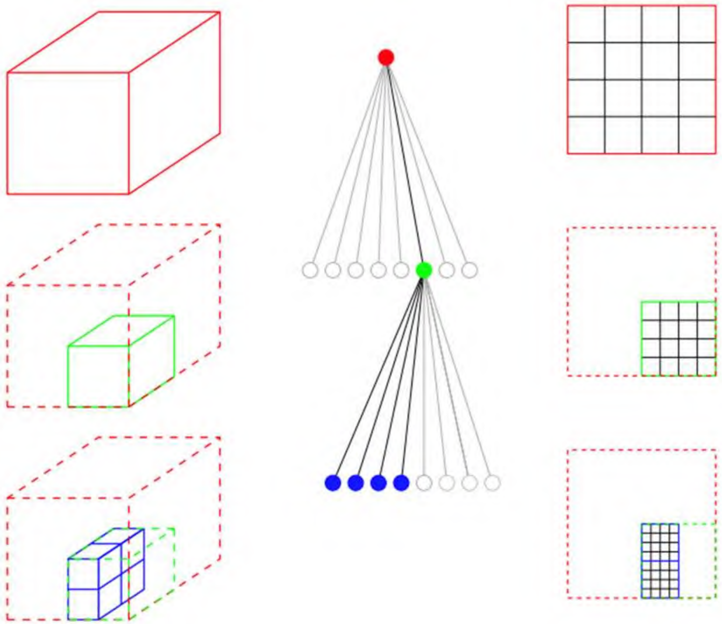
Discretize a 3-D space with some resolution: D x D x D

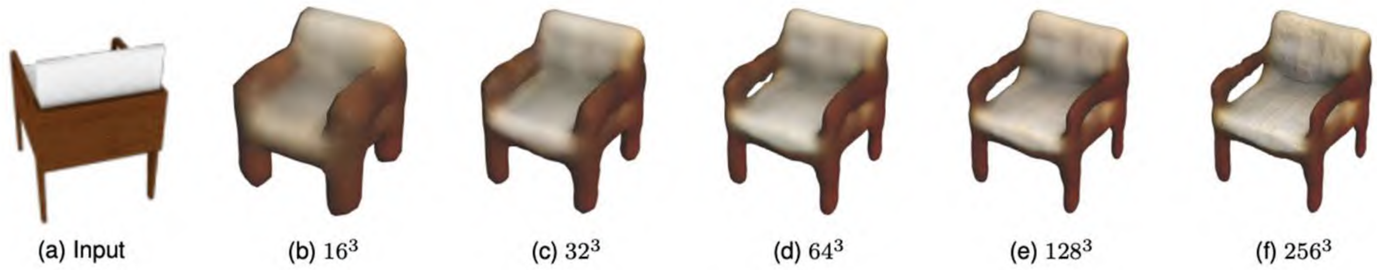
What is stored at each value can be:

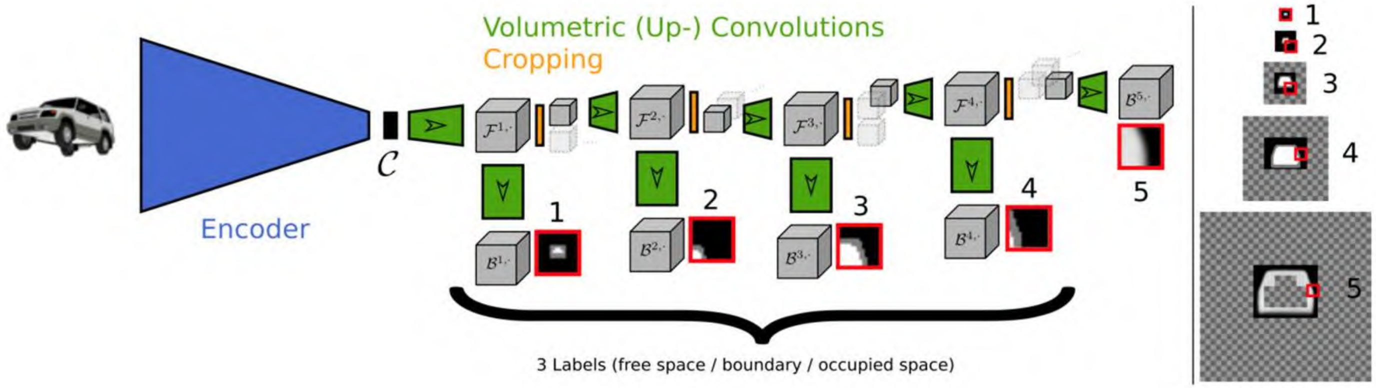
- 1 or 0 (occupancy)
- Signed distance
- Color/attributes

Was the easiest to get adopted into Deep Learning because of its proximity to pixels.

Oct-tree (八叉树)



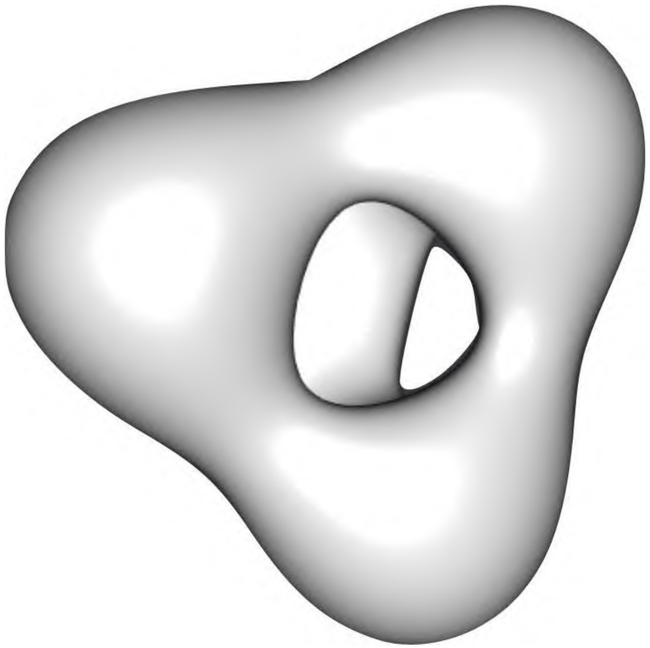




OctNet: Learning Deep 3D Representations at High Resolutions, Riegler et al. CVPR2017
Hierarchical Surface Prediction for 3D Object Reconstruction Häne, 3DV 2017

Volume for Surfaces

- You can use volumes to represent Surfaces
- the zero-crossing of a continuous function is the surface



$$\{\mathbf{p} \mid f(\mathbf{p}) = 0\}$$

```python
def f(p):
    '''
    Input:
        p: a point in R^3
    Output:
        v: A scalar value
    '''
```

Can be anything — an analytic function, or a neural network, or a voxel

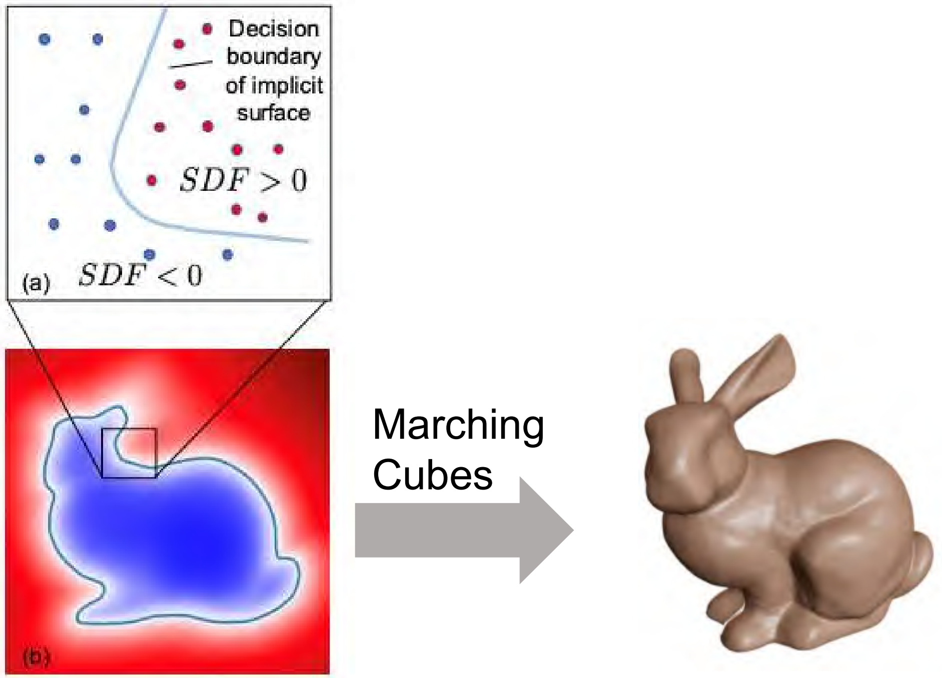
Implicit Volume

Instead of explicitly outputting a discretized volume D^3, learn a function:

$$f_\theta : \mathbb{R}^3 \times \mathcal{X} \to [0, 1]$$

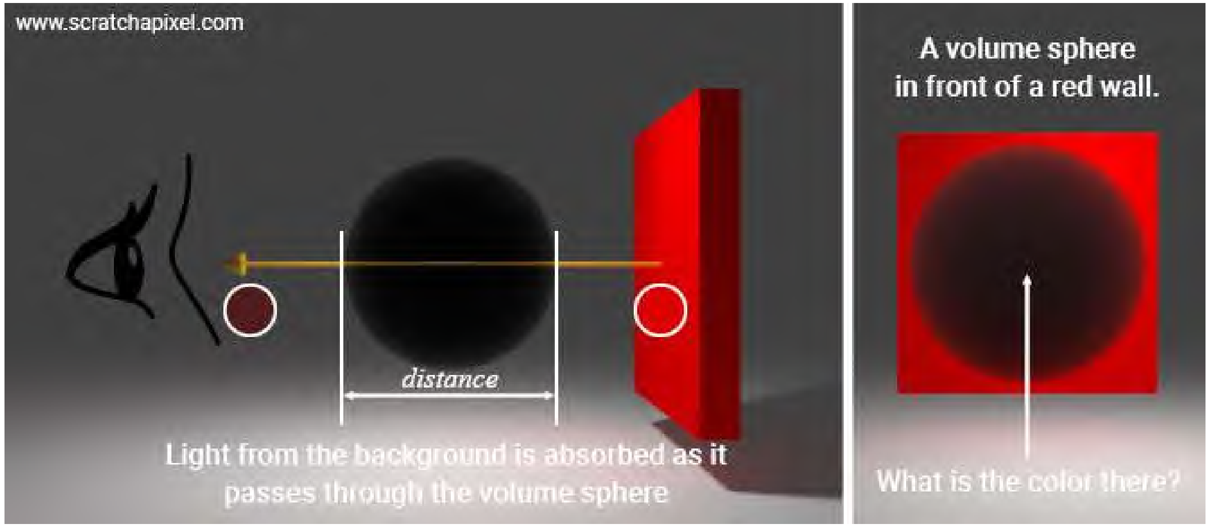
Output can be an occupancy {0, 1} or a real valued signed distance.
X is some N-dim image embedding

This f is often just a simple 3~8 layer MLP.
f(x,y,z, observation) = $\mathbb{R}$

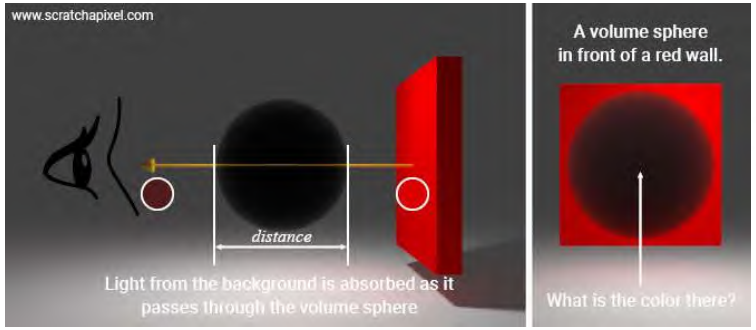


DeepSDF, CVPR'19,
OccupancyNet, CVPR'19,
Learning Implicit Fields for Generative Shape Modeling, CVPR'19

Rendering Volumes



Rendering Volumes



Given color and density (r, g, b, σ) , we calculate the color of every camera ray using:

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

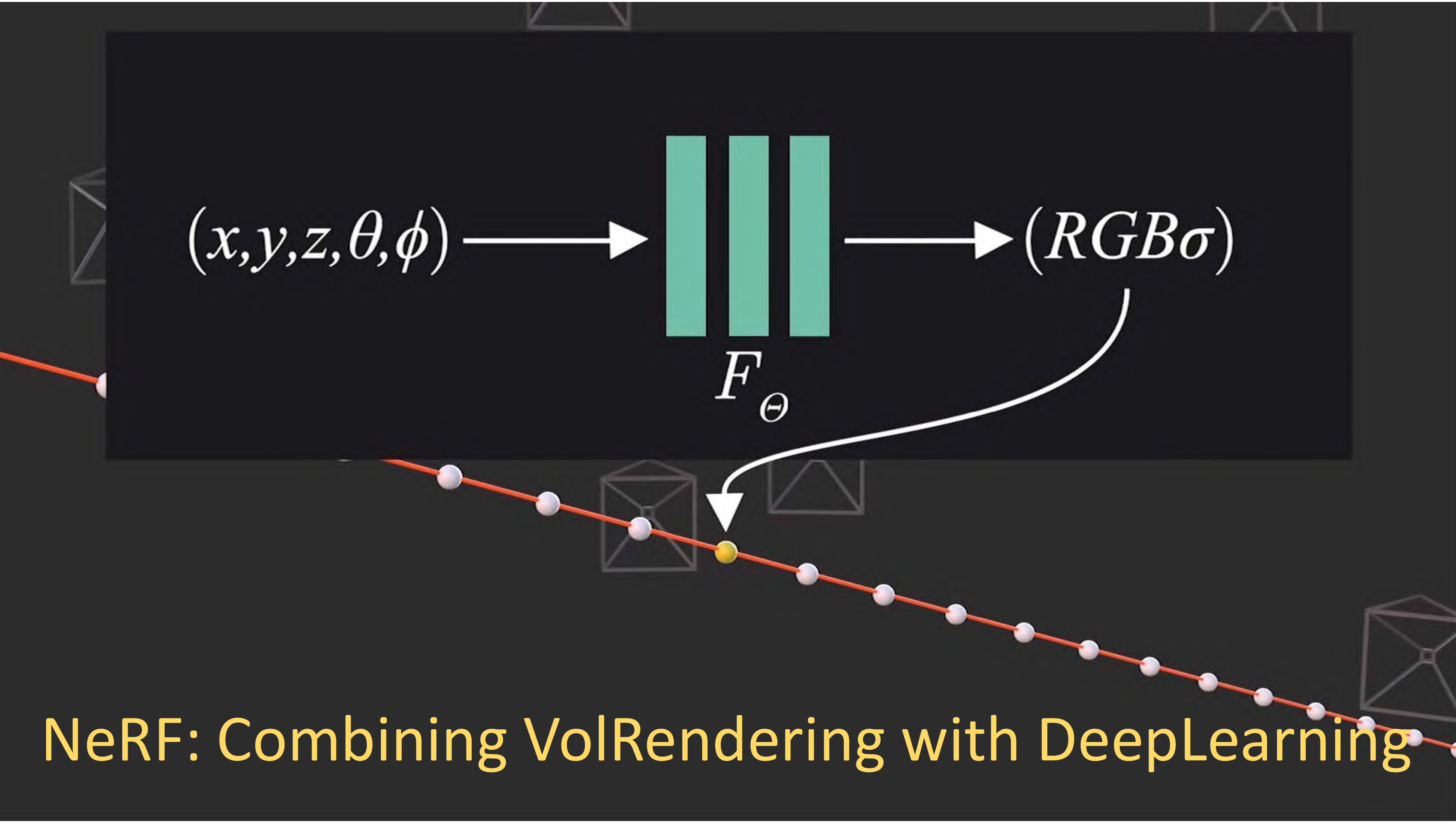
Camera ray: $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$

Near and far bounds

Color (r,g,b) at r(t)

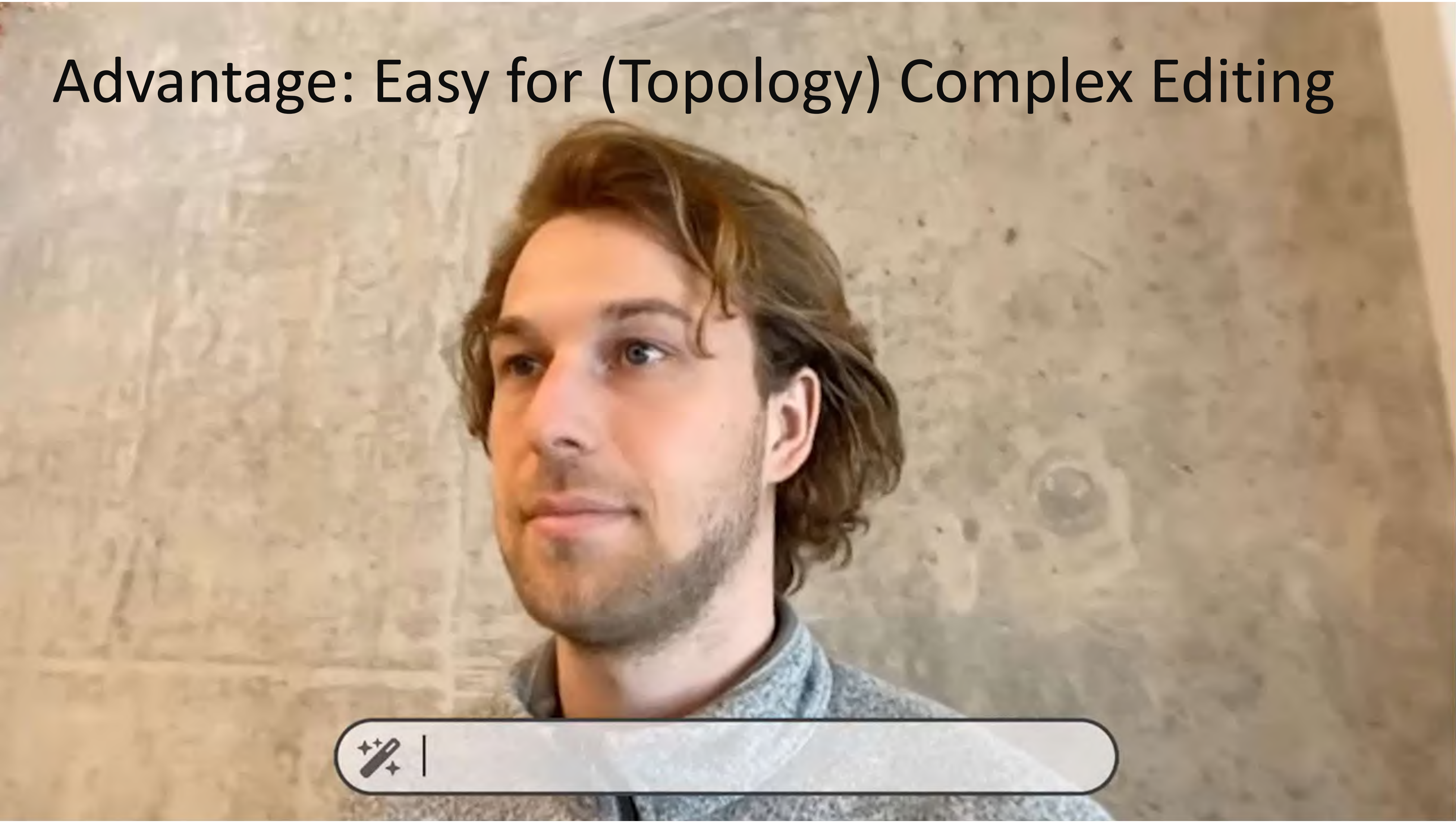
Volume density: Probability of a ray terminating at an infinitesimal particle at location r(t)

Accumulated transmittance along ray :
$T(t) = \exp(-\int_{t_n}^{t_f} \sigma(\mathbf{r}(s))ds)$

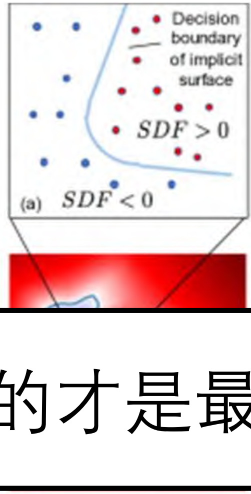

(x, y, z, θ, ϕ)
F_{Θ}
$(RGB\sigma)$
NeRF: Combining VolRendering with DeepLearning

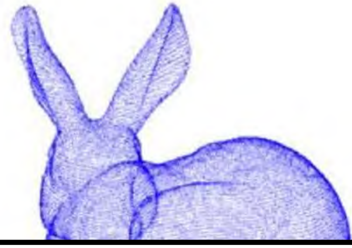
Advantage: Very Realistic Rendering

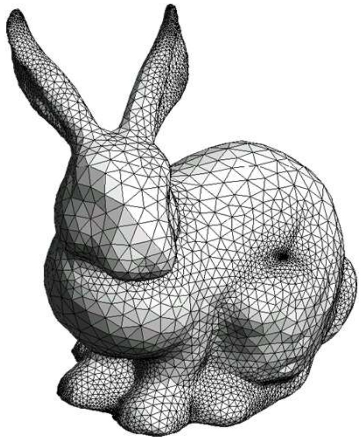
Advantage: Easy for (Topology) Complex Editing



Summary







合适的才是最好的

Volumetrics

2.5D / Image Based Rendering

- 👍 Can make pretty images
- 👍 Capture higher order lighting effects
- 👍 Robust off-the-shelf models (MiDas, VIT..)
- ✗ often not suitable for large baseline/360 view

Explicit / Implicit

- 👍 Easy to train with VolRend
- 👍 Topology free (editing)
- 👍 PhotoRealistic Rendering
- ✗ No surface*
- ✗ Hard to texture
- ✗ Exp: Memory intensive*
- ✗ Imp: Expensive to render

Point clouds

- 👍 Better memory
- 👍 Topology free
- ✗ No surface
- ✗ Can't print it

- 😳 Need to splat / holes
- 😳 Easy to get stuck in diffrend

Meshes

- 👍 Better memory
- 👍 Explicit surface
- 👍 Great for texture/light
- 👍 Great for deformation
- 👍 Most common in 3D artists/graphics
- ✗ Topology sensitive
- 😳 Easy to get stuck in diffrend

中国科学技术大学
University of Science and Technology of China

谢谢观看!