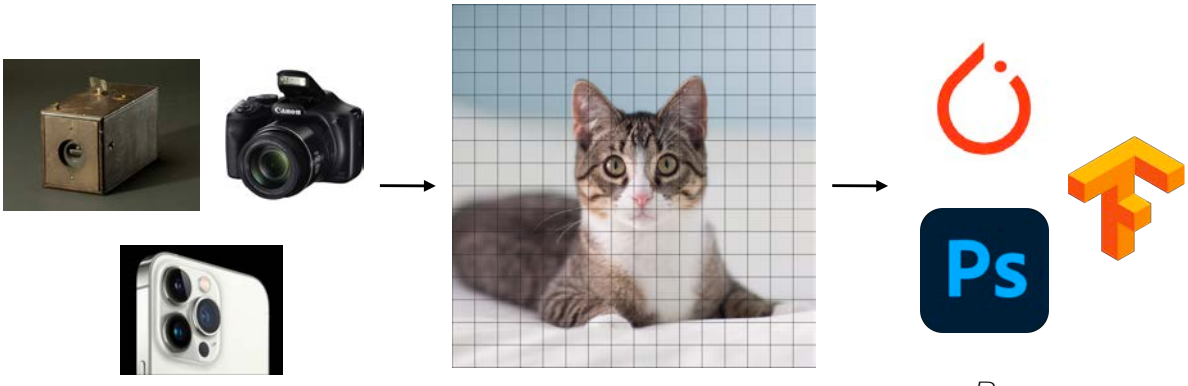
Representations for 3D

Based on Slides by Shubham Tulsiani, CMU 16-825, 2025

Life is nice in 2D-land…

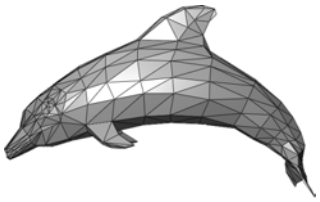
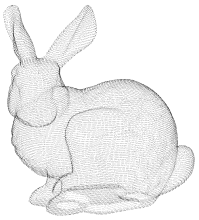
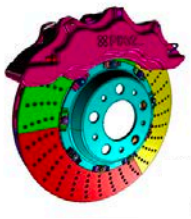


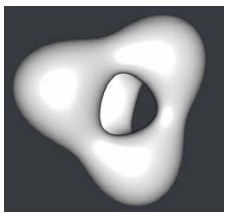
Acquisition

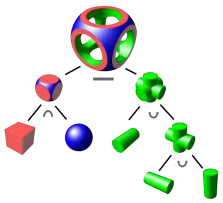
Process

Unified representation across applications, tools, and sensors

Meanwhile in 3D-land…





and many more …

Meanwhile in 3D-land…

But what is the best representation ?!

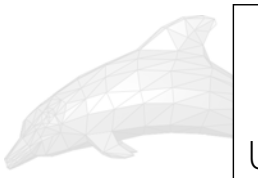
well, depends ..

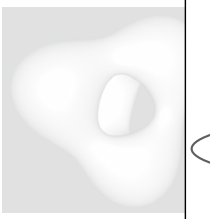
Acquisition vs Editing vs Animation vs Prediction vs Processing …

and many more …

$\{\mathbf{p} \mid f(\mathbf{p}) = 0\}$ $f(\mathbf{u}) = \mathbf{p} \in \mathbb{R}^3$

Meanwhile in 3D-land…





$\{\mathbf{p} \mid f(\mathbf{p}) = 0\}$ $f(\mathbf{u}) = \mathbf{p} \in \mathbb{R}$

…any more …

Goals:

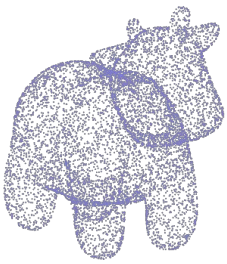
Understand representations computationally
(what do they look like in code?)

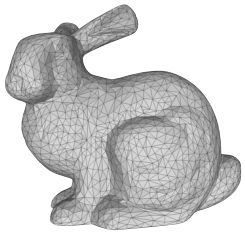
Limitations and Benefits

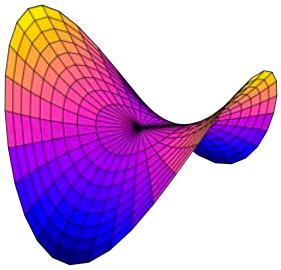
Conversions across representations

Outline

- Boundary Representations for Shapes
- Volumetric Representations for Shapes

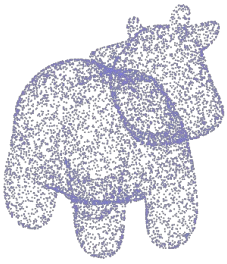




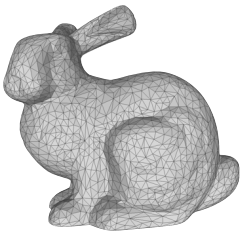


Boundary Representations
for Shapes

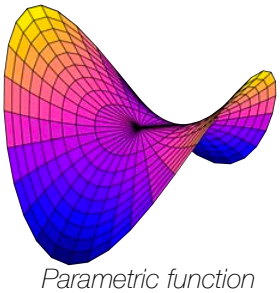
Volumetric Representations
for Shapes



Points

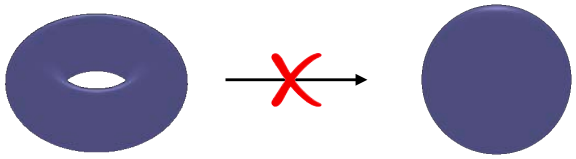


Mesh



Parametric function

Surface Representations



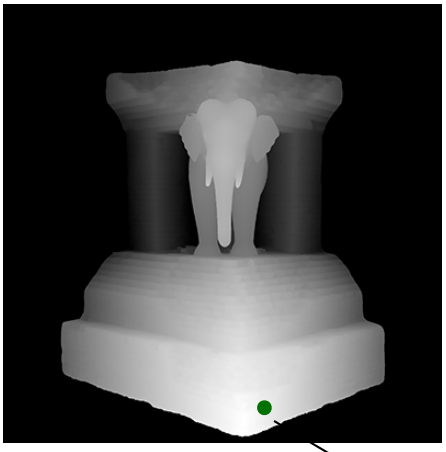
OBS: topological changes

Depth Maps

Depth Maps — Acquisition

Common modality across sensors

Depth Maps — Representation



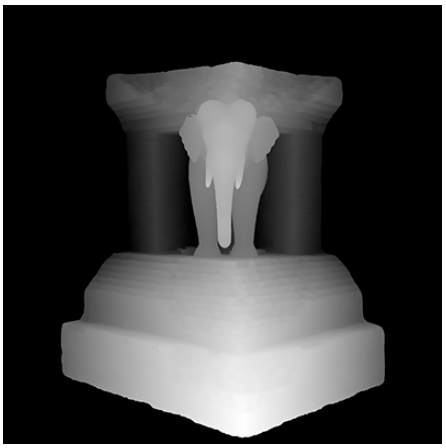
An **image** that represents how far each pixel **p** is

$$D[\mathbf{p}] \in \mathbb{R}^+$$

$$p = (u, v, d)$$

Image credits: Paul Bourke

Depth Maps — Tools

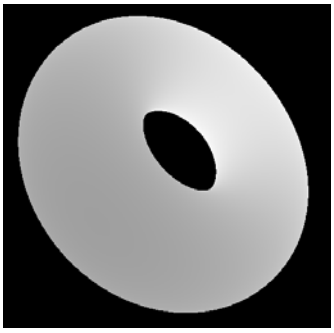


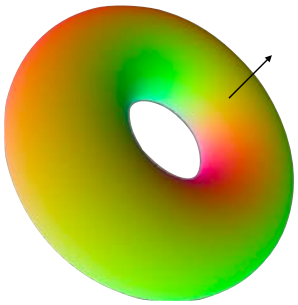
OBS: Range

The 'standard' image processing tools can be used

Image credits: Paul Bourke

Depth Maps and Normal Maps



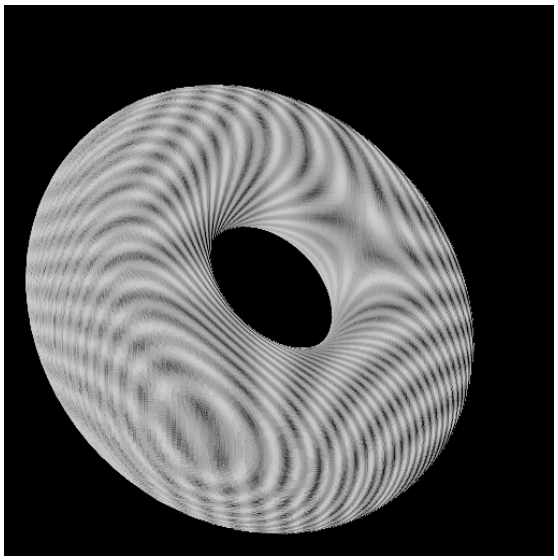


$N[\mathbf{p}] \in \mathbb{S}^2$

Can extend to capture other surface properties e.g. surface normals

OBS: Reconstruction from Normal Field

Depth Maps — Representing the Visible



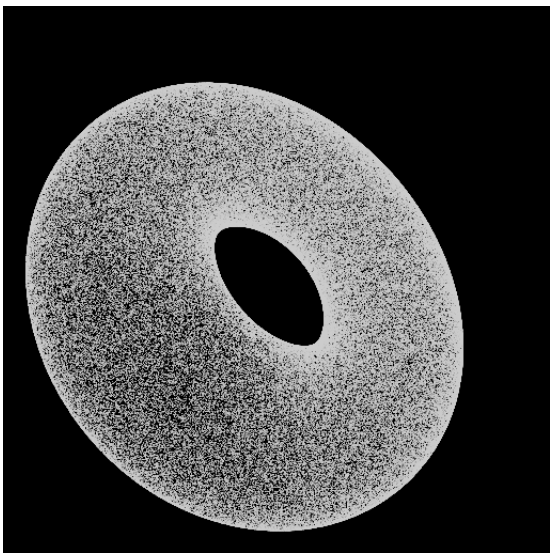
Does not capture the 'full' 3D structure

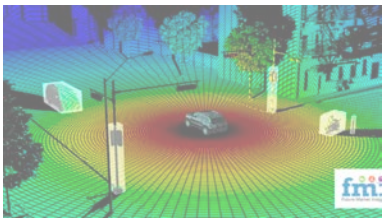
'2.5D' representations — properties associated to image pixels (camera)

can we relax this 'image-dependence' constraint?

Point Clouds

Point Clouds



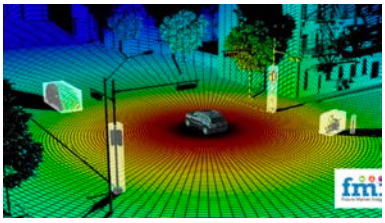


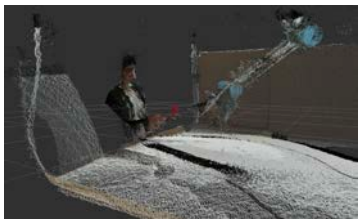
Point Clouds — Acquisition

Often acquired in an image-like manner from sensors but can be useful to 'forget' this

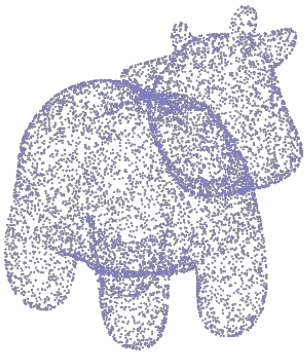
e.g. point clouds from multiple views can make a single point cloud

can help unify approaches independent of sensing modality

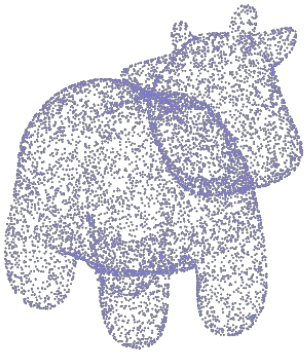




Point Clouds — Representation



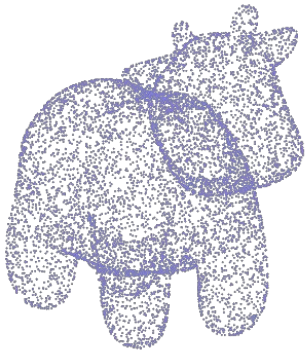
Permutation σ



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

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

Point Clouds – Data Structure



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

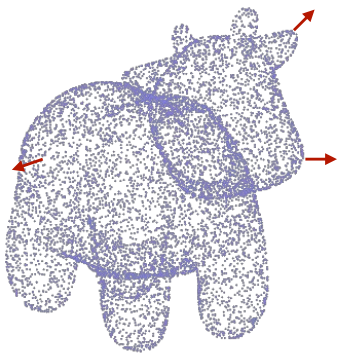
Often represented as a NX3 array, but ordering does **not** matter (unlike images)

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

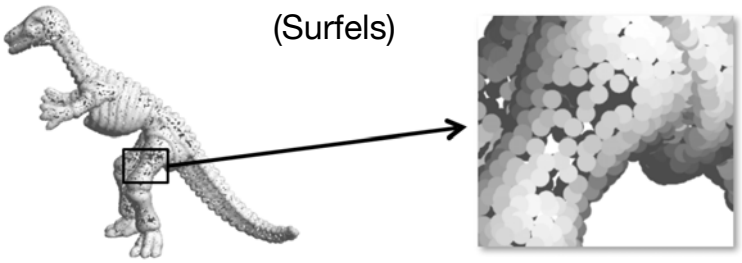
$$\{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_N\}$$

Unordered set of points

Oriented Point Clouds



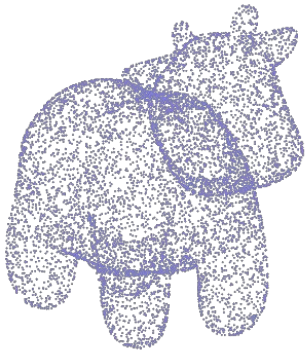
Can treat points as small flat disks instead of tiny spheres - helps in rendering under lighting



$$\{(\mathbf{p}_1, \mathbf{n}_1), (\mathbf{p}_2, \mathbf{n}_2), \cdots, (\mathbf{p}_N, \mathbf{n}_N)\}$$

Image credits: Hao Su

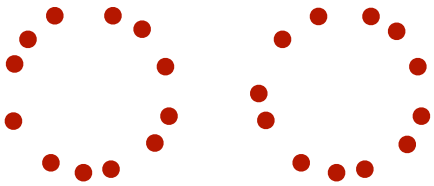
Point Clouds — Properties



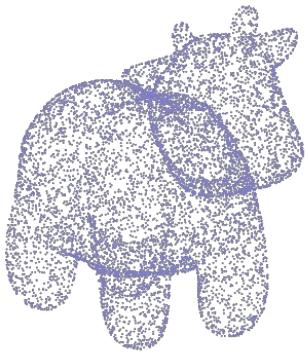
$$\{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_N\}$$

Unordered set of points

No explicit ‘connectivity’ information



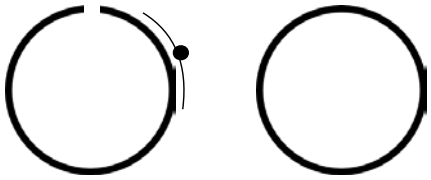
Point Clouds — Neighborhoods



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

Unordered set of points

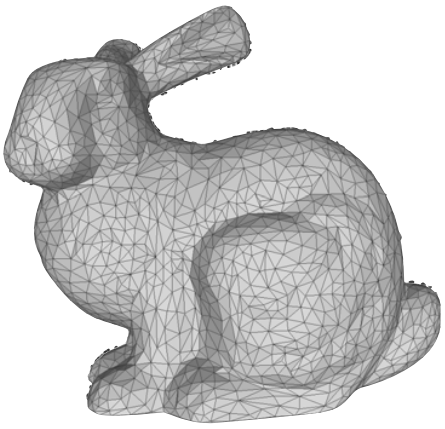
No explicit ‘connectivity’ information



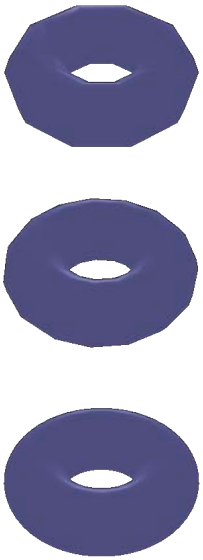
(yes, we can sample more - but more efficient to add edges)

Meshes

Meshes — Model



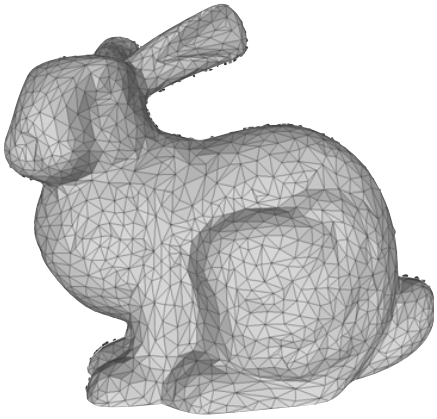
Vertices + Faces



Piecewise linear approximations of underlying surface

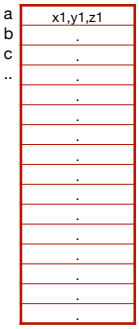
More elements allow better approximation

Meshes — Data Structures



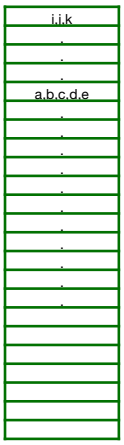
Vertices + Faces

Vertices



Positions of Vertices

Faces

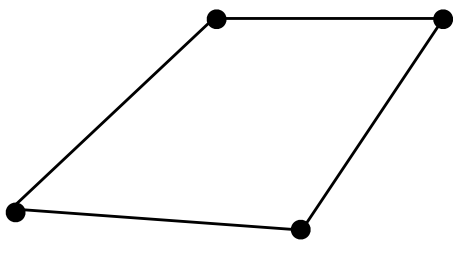


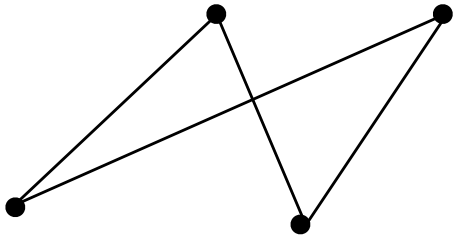
Connectivity
(indices of vertices that
make a 'face')

OBS: Others (Half-Edge, etc)

Meshes - Topological Structure

Ordering Matters in Face Indices





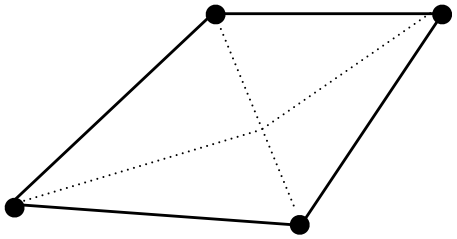
Meshes - Geometry

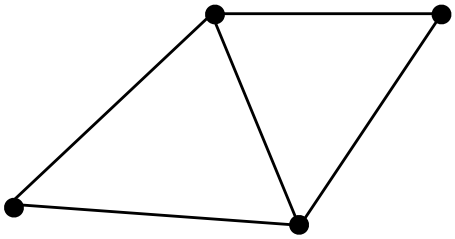
Faces

Vertices

i,j,k

Arbitrary Polygons can be non-planar





restrict to *Triangular* Meshes

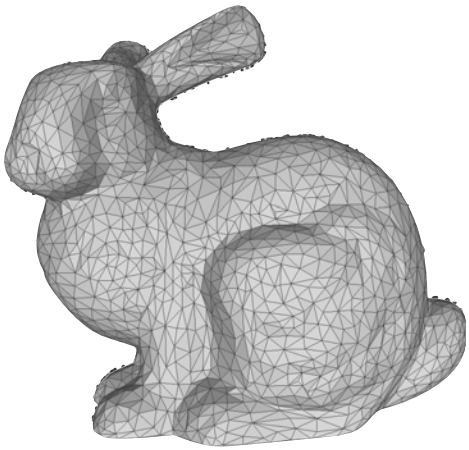
Vertices + Faces

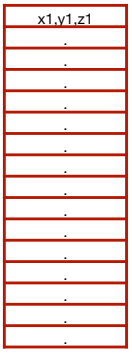
(indices of vertices that make a 'face')

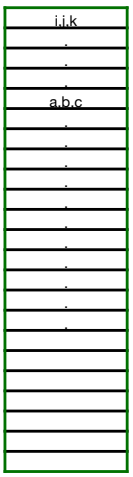
Triangular Meshes - Simplicial Structure

Faces

Vertices







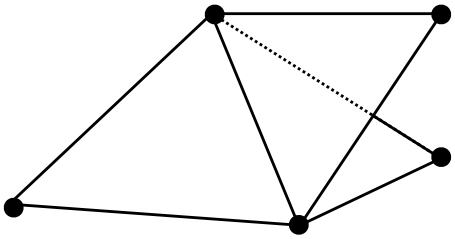
Vertices + Faces

Positions of Vertices

Connectivity
(indices of **three** vertices
that make a 'face')

Triangular Meshes - Discrete Manifolds (a)

Can still get non-manifold meshes

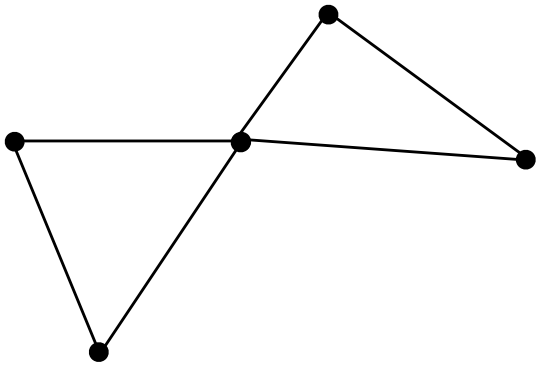


Restrict each edge to maximum 2 faces*

* can often get away without enforcing this

Triangular Meshes - Discrete Manifolds (b)

Can still get non-manifold meshes



Disconnected surface around vertex

* critical for mesh processing applications, less so for rendering

Triangular Meshes — Representation

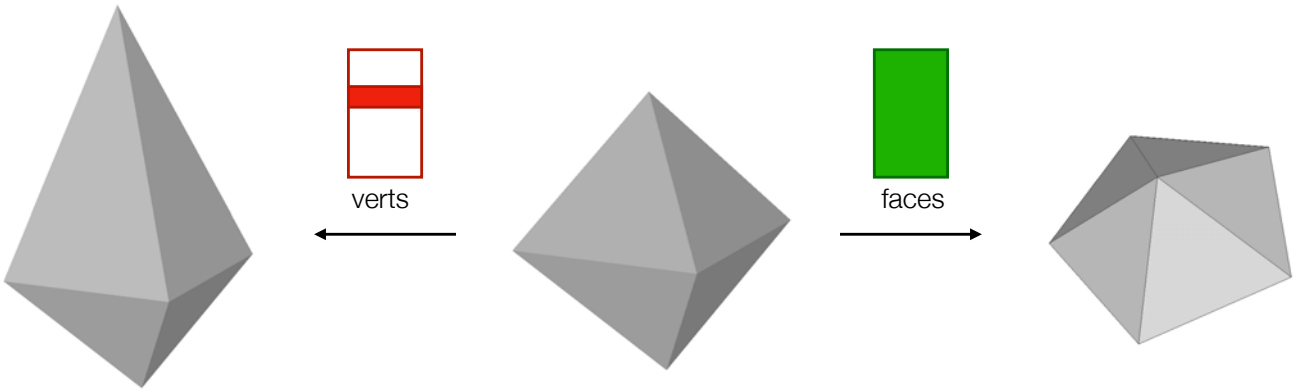
Efficient to compute ray-triangle intersections

(shape)

Easy to texture and render (common representation across graphics)

(attributes)

Triangular Meshes — Geometry Operations

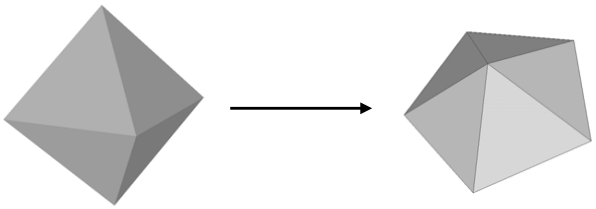


Modifying vertex positions can edit shape

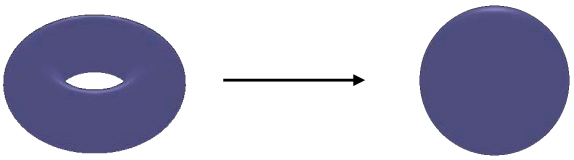
But other changes are non-trivial

Triangular Meshes — Topology Operations

Non-trivial even though both have 8 vertices and 10 faces



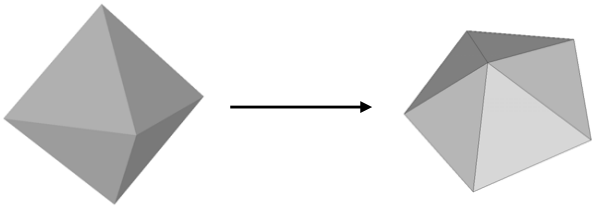
Even more challenging if topology changes



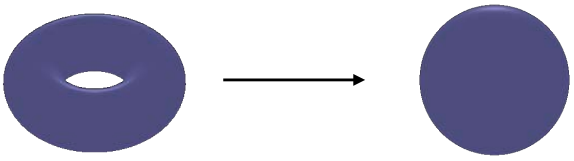
In general, designing methods that can predict **arbitrary connectivity** is challenging (e.g. image to mesh)

Triangular Meshes — Deformations

Possible under continuous deformations



Impossible under continuous deformations

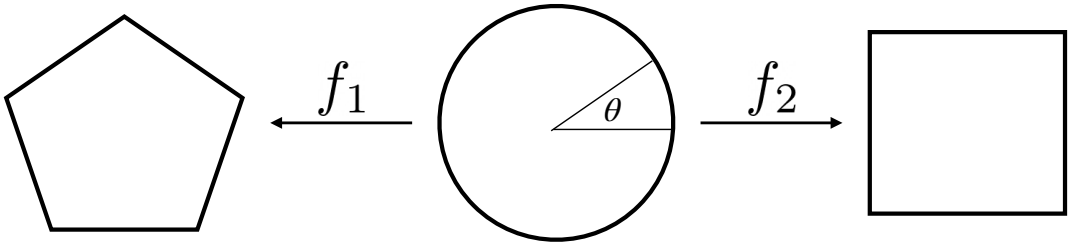


Both edits require similar operations for meshes (new Fs and updated Vs).

Is there an alternate representation where the former is easier?

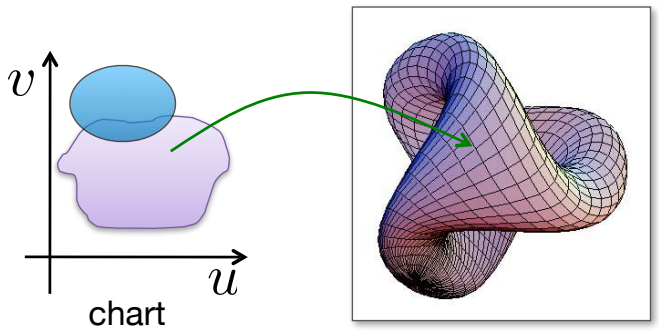
Parametric Surfaces

Parametric Surfaces (in 2D)



$$f(\theta) = \mathbf{p} \in \mathbb{R}^2;\ \theta \in \mathbb{S}^1$$

Parametric Surfaces (in 3D)



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

A continuous function ***f*** over a 2D manifold $\mathcal{M}$ can parametrize a surface

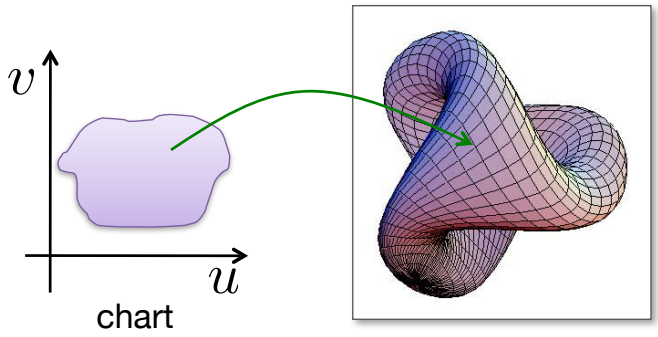
Atlas: { charts }

The connectivity/topology is constrained to be same as $\mathcal{M}$

Disentangles discretization (vertices/ faces) from shape representation

Image credits: Hao Su

Parametric Surfaces — Manifold Models



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

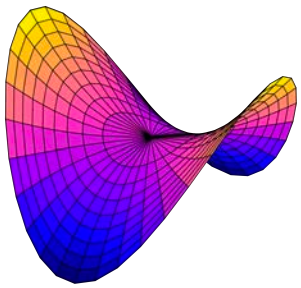
A continuous function ***f*** over a 2D manifold $\mathcal{M}$ can parametrize a surface

```
def f(u):
    '''
    Input:
        u: N-dimenstional vector from a 2D manifold
    Output:
        p: 3D point
    '''
```

Can be a fixed analytic function, parametrized family, or even a neural network!

Image credits: Hao Su

Parametric Surfaces — Analytic



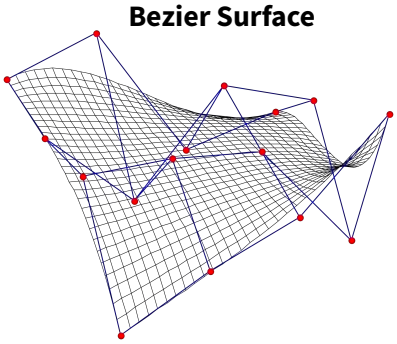
$$f((u,v)) = (u, v, v^2 - u^2)$$

```
def trefoil(u, v):
    x = r * np.sin(3 * u) / (2 + np.cos(v))
    y = r * (np.sin(u) + 2 * np.sin(2 * u)) / (2 + np.cos(v + np.pi * 2 / 3))
    z = r / 2 * (np.cos(u) - 2 * np.cos(2 * u)) * (2 + np.cos(v)) * (2 + np.cos(v + np.pi * 2 / 3)) / 4
```

Image credits: Wikipedia

Parametric Surfaces — Polynomial Patches

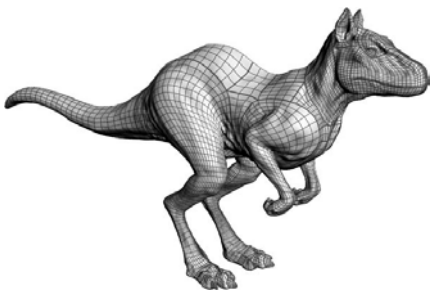
Bezier Surface



$$f((u,v)) = \sum_{i}^{N} \sum_{j}^{M} B_i^N(u)\ B_j^M(v)\ k_{i,j}$$

$$B_i^N(u) = \binom{N}{i} u^i (1-u)^{N-i}$$

M*N control points can determine a smooth surface in 3D



Commonly used for designing meshes

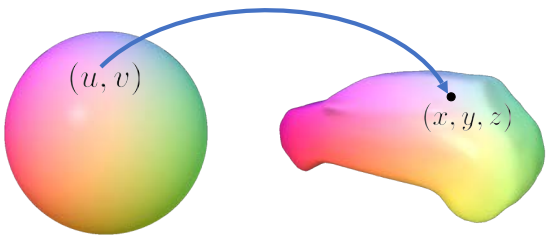
Image credits: Wikipedia

Parametric Surfaces — Neural Networks

$$f_\theta(\mathbf{u}) = \mathbf{p} \in \mathbb{R}^3;\ \mathbf{u} \in \mathbb{S}^2$$

Neural network with parameters θ

Parametric Surfaces — Properties



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

manifold $\mathcal{M}$ can parametrize a surface

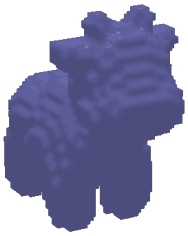
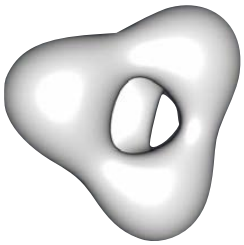
Easy to sample points on surface

Can do computation in the domain $\mathcal{M}$

Difficult to reason about global structure (e.g. is a query point **q** outside the shape?)

Difficult to analytically render (ray-surface intersection is hard for arbitrary **f**)

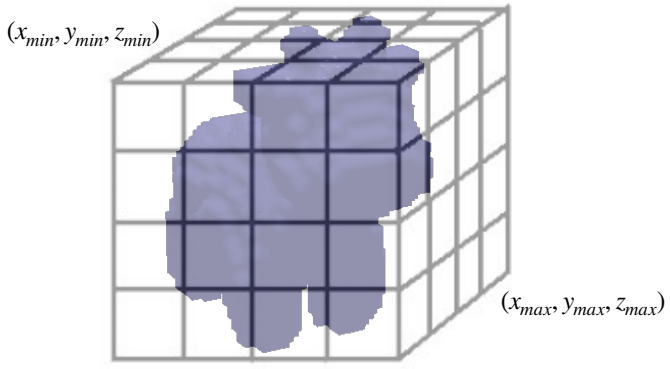
Boundary Representations
for Shapes

Volumetric Representations
for Shapes

3D Voxel Grids

Voxelized 3D — Representation



$V[x, y, z] \in [0,1]$

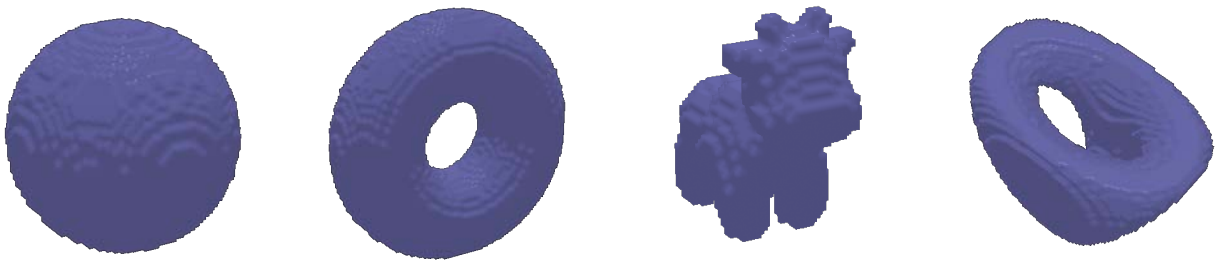
Discretized representation of the space demarcated via cuboid endpoints

A (W X H X D) grid representing occupancy (or probability) at each cell

Coding tip: disentangle grid coordinates vs spatial extent in world

An 'image-like' representation — can use convolutions for processing and generation

Voxelized 3D — Model (Binary Occupancy Function)



A generic way for representing shapes across topologies

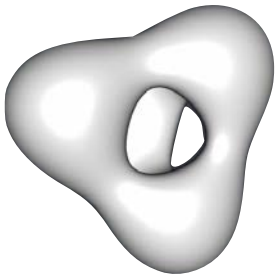
Voxelized 3D — The Curse of Dimensionality

32^3 64^3 128^3 256^3 mesh

Computationally challenging to scale resolution

Implicit Surfaces

Implicit Surfaces — Model (Extrinsic)



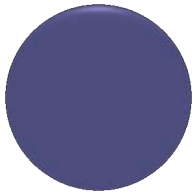
$$\{\mathbf{p} \mid f(\mathbf{p}) = 0\}$$

The zero crossing of a continuous function ***f*** can parametrize a surface

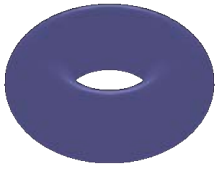
```
def f(p):
    '''
    Input:
        p: a point in R^3
    Output:
        v: A scalar value
    '''
```

Can be anything — a simple function, or a neural network.

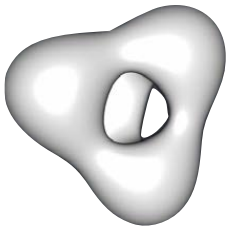
Implicit Surfaces — Algebraic Primitives



$$f(x, y, z) = x^2 + y^2 + z^2 - 1$$



$$f(x, y, z) = 4R^2(x^2 + y^2)$$
$$-(x^2 + y^2 + z^2 + R^2 - a^2)^2$$

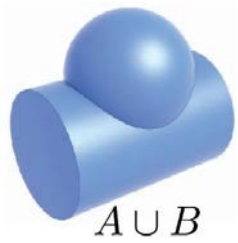


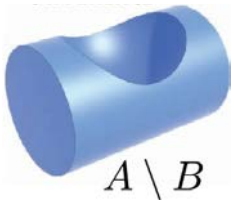
$$f(x, y, z) = x^4 + y^4 + z^4 + 2x^2y^2 + 2x^2z^2$$
$$+2y^2z^2 + 8xyz - 10x^2 - 10y^2 - 10z^2 + 20$$

Simple functions can offer concise representations for complex shapes

Implicit Surfaces — Constructive Solid Geometry (CSG)

Shape Algebra + Primitives





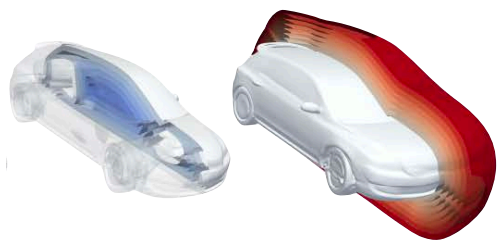
$$\cup_i f_i(\mathbf{p}) = \min_i f_i(\mathbf{p}) \qquad \cap_i f_i(\mathbf{p}) = \max_i f_i(\mathbf{p}) \qquad (f - g)(\mathbf{p}) = \max(f(\mathbf{p}), -g(\mathbf{p}))$$

Boolean operations are easy

(max and min depend on convention — is +ve outside or inside? Assumed outside here.)

Image credits: Keenan Crane

Implicit Surfaces — Neural Networks

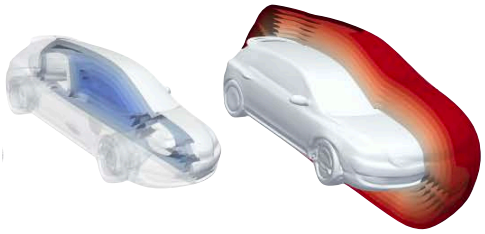


$$\{\mathbf{p} \mid f_\theta(\mathbf{p})\} = 0$$

Neural network with parameters θ

Image credits: IGR: Implicit Geometric Regularization for Learning Shapes. *Gropp et. al*

Implicit Surfaces - Signed Distance Function (SDF)



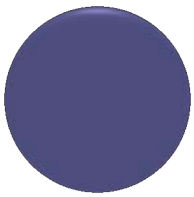
$$f(x) = \begin{cases} d(x, \partial\Omega) & \text{if } x \in \Omega \\ -d(x, \partial\Omega) & \text{if } x \notin \Omega. \end{cases}$$

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

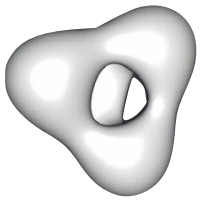
Additional condition that ***f*** represents
(signed) distance to surface from point **p**

$$d(x, \partial\Omega) = \inf_{y \in \partial\Omega} d(x, y),$$

Implicit Surfaces and SDFs



$$f(x, y, z) = x^2 + y^2 + z^2 - 1$$



$$f(x, y, z) = x^4 + y^4 + z^4 + 2x^2y^2 + 2x^2z^2$$
$$+2y^2z^2 + 8xyz - 10x^2 - 10y^2 - 10z^2 + 20$$

Which of these is an SDF?

Neither!

Implicit Surfaces - Properties of SDFs

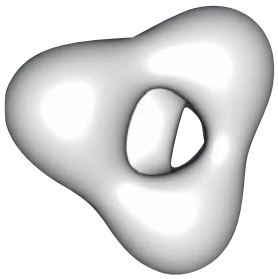
- If the SDF is differentiable, its gradient satisfies the Eikonal equation

$$|\nabla f| = 1.$$

- If the SDF is C^k for k ≥ 2, $d(x)$ is C^k on points close to the zero level set. In particular, on the boundary it satisfies

$$\nabla f(x) = N(x),$$

Implicit Surfaces — Summary

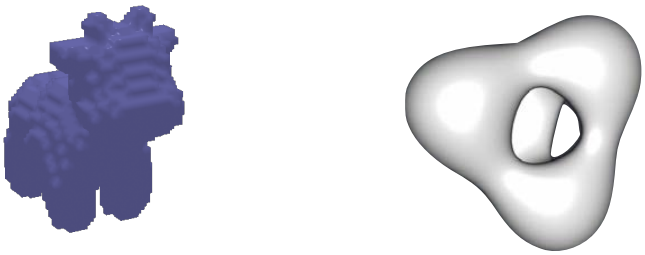


$\{\mathbf{p} \mid f(\mathbf{p}) = 0\}$

The zero crossing of a continuous function ***f*** can parametrize a surface

Easy to answer questions about occupancy

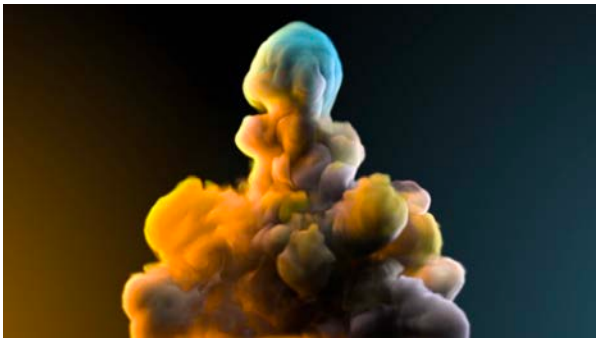
Difficult to reason about the surface (may not even exist given an arbitrary ***f*** !)

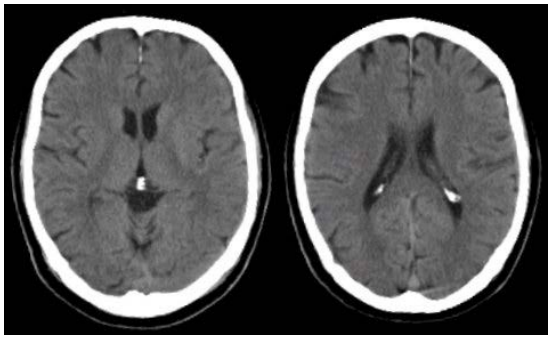


Volume Representations

Density Fields

Density Fields — Model



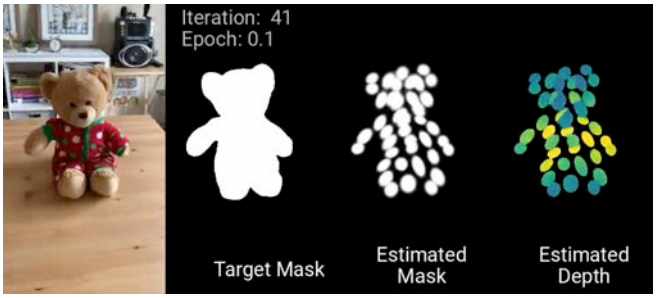


$$V[x, y, z] \in \mathbf{R}^+ \qquad f(\mathbf{p}) \in \mathbb{R}^+$$

No notion of a surface (e.g. $\boldsymbol{f}(\mathbf{p}) = 0.1$ everywhere can represent a semi-transparent fluid)

Will revisit in more detail (e.g. in NeRF)

Primitive-based Density Fields

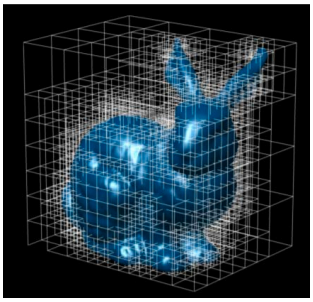


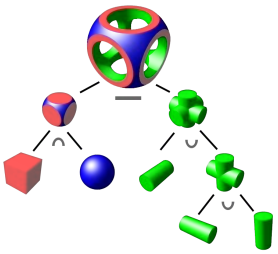
Hybrid representations for primitive-induced density fields (3DGS)

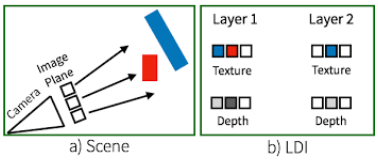
Will revisit in more detail (e.g. in Differentiable Gaussian Rendering)

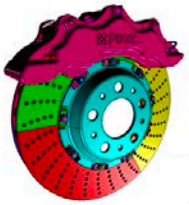
Image credits: *Left*: Local Deep Implicit Functions for 3D Shape, Genova et. al. *Right:* Fuzzy Metaballs, Keselman & Hebert

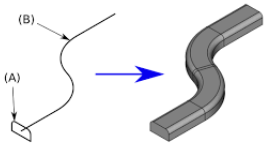
And many more …



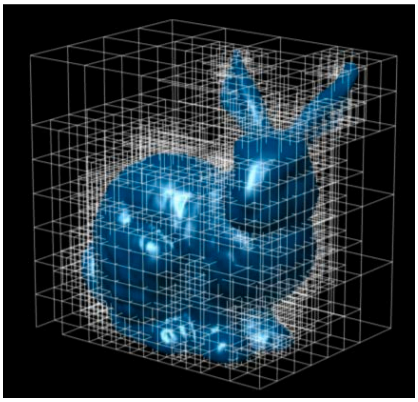








Octrees

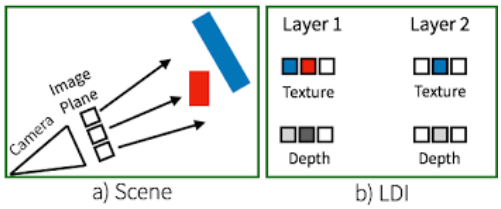


Voxels with Adaptive resolution

Surface-like efficiency for rendering and storage, volume-like efficiency for occupancy queries

Difficult to edit or predict sparsity structure

Layered Depth Images



a) Scene b) LDI

Layer 1 = Visible Content

Layer N+1 = Depth and Color of the 'next' surface point the ray hits after Layer N

Extends depth to capture 'full' 3D

Often very discontinuous

Surface

Layer 1

Layer 2

Layer 3

Multi-plane Images

Discrete Volumetric Representations with non-uniform spatial cells

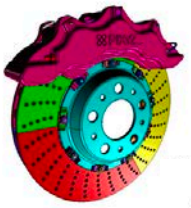
Image courtesy: Tinghui Zhou

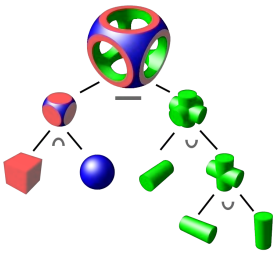
Extremely efficient to render - each 'layer' is just a plane at fixed depth

Limited resolution for far-away content

Uniform cells maybe preferable for object-centric 3D

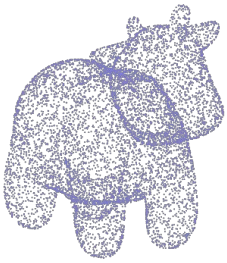
CAD Models and CSG Trees

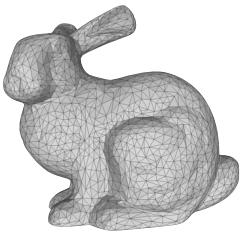


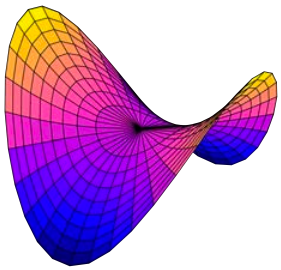


Compositions of primitive volumes or surfaces

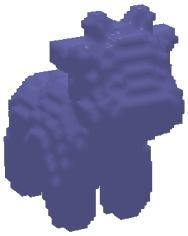
Popular across design applications, but challenging for learning

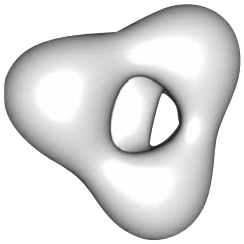




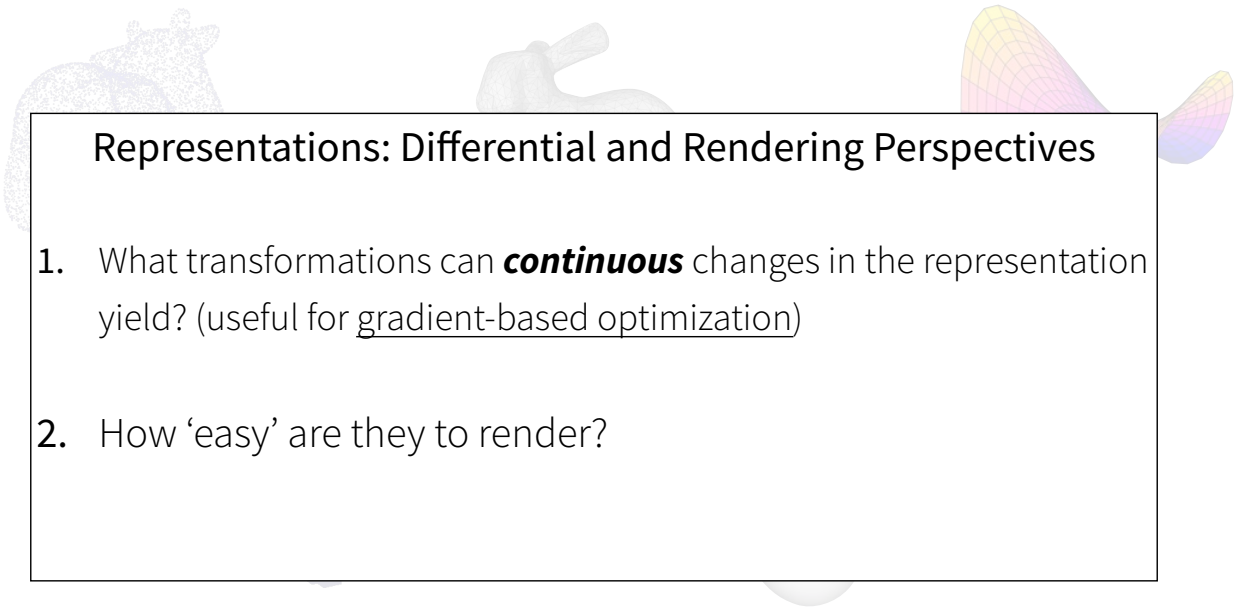


Boundary Representations for Shapes





Volumetric Representations for Shapes



Representations: Differential and Rendering Perspectives

1. What transformations can ***continuous*** changes in the representation yield? (useful for gradient-based optimization)

2. How ‘easy’ are they to render?

Volumetric Representations
for Shapes