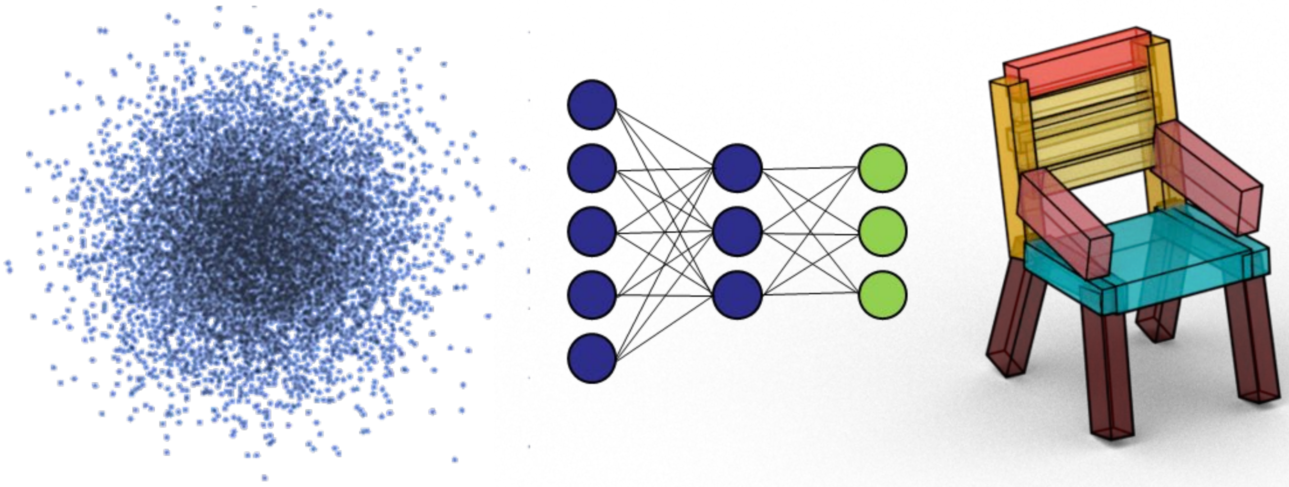
CS348n: Neural Representations and Generative Models for 3D Geometry



Leonidas Guibas
Computer Science Department
Stanford University

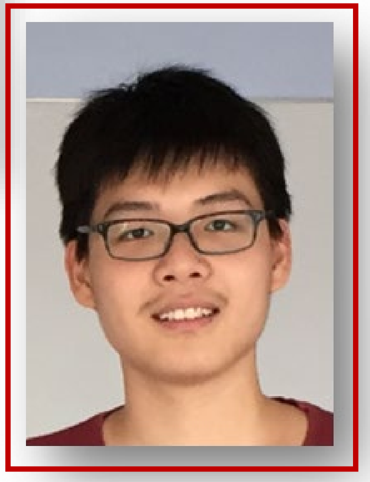
Leonidas Guibas Laboratory

Geometric Computing

Recap: Class Logistics

The Class Principals

- Leonidas (Leo) Guibas (CS & EE)
  - Instructor
- Kaichun Mo (CS)
  - Course Assistant (TA)
- Carrie Petersen (CS)
  - Admin



Also, a number of guest speakers …

Canvas for class videos, etc…

http://cs348n.stanford.edu

Class venue: Zoom, Clark S361, Gates 105, …

Course Requirements / Mechanics

- 3 programming assignments (1 week, 2 weeks, 2 weeks) using Google Cloud for Education
- 1 small project (BYI, but suggestions also provided, 3 weeks)
- 1 class presentation on research papers from the literature (topics covered in the previous class)
- teams of up to three students allowed

- we'll use Piazza (www.piazza.com) as the class discussion forum, and Gradescope (www.gradescope.com) for assignment submissions

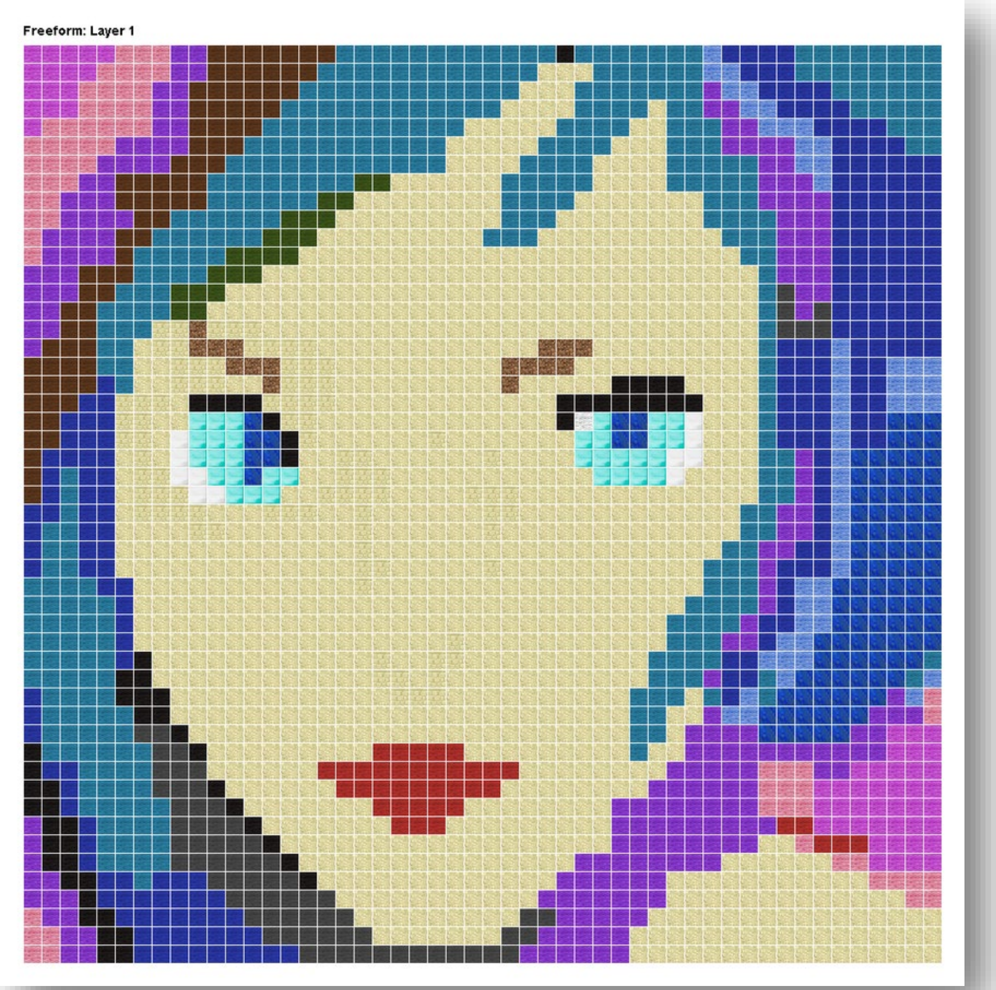
Other Tidbits

- Sign up for Piazza
- Sign up for your class presentation session:
  - Google form: https://forms.gle/xNzWptSzfngzmuGs7
- For access to class lecture slides:
  - Use credentials:
    - user: neural
    - passwd: creation
- Obtain Google Cloud coupons from CA

Geometry Representations
and
Geometry Processing

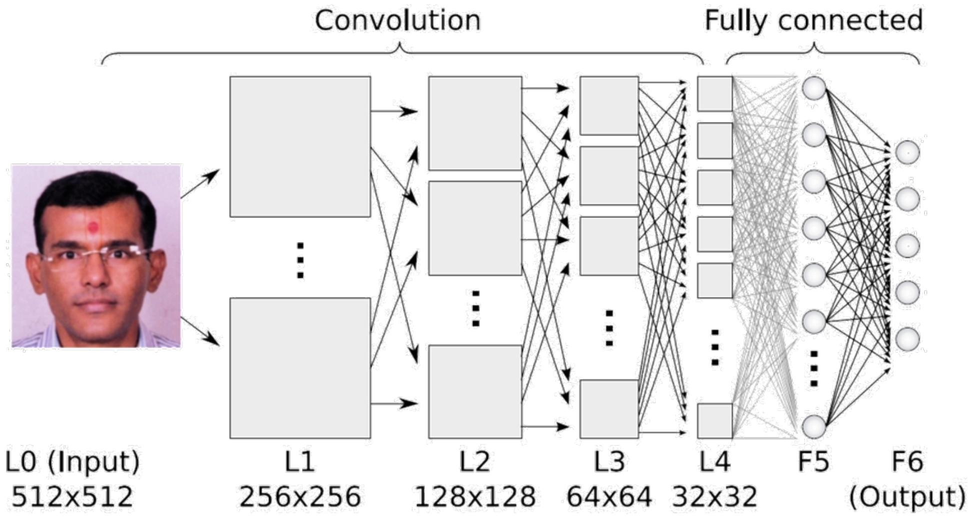
Mostly traditional …

Images/Videos Have Canonical Representations

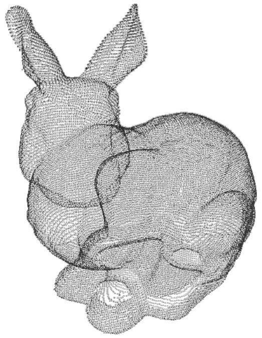


Pixel arrays

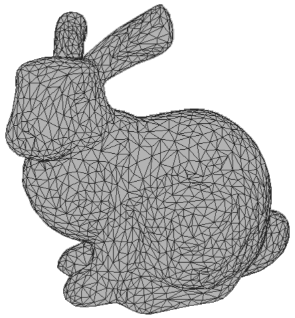
Regular representations
aid ML algorithms, e.g.
convolutional deep networks



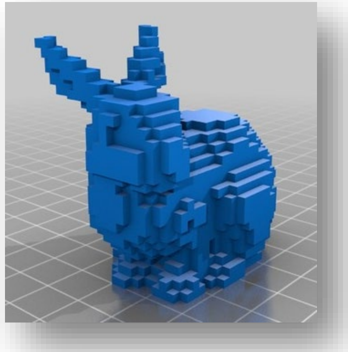
In 3D, There is Representation Diversity



Point Cloud



Mesh

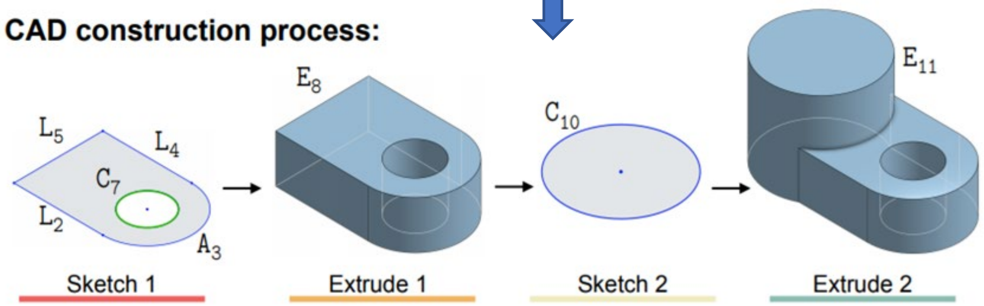


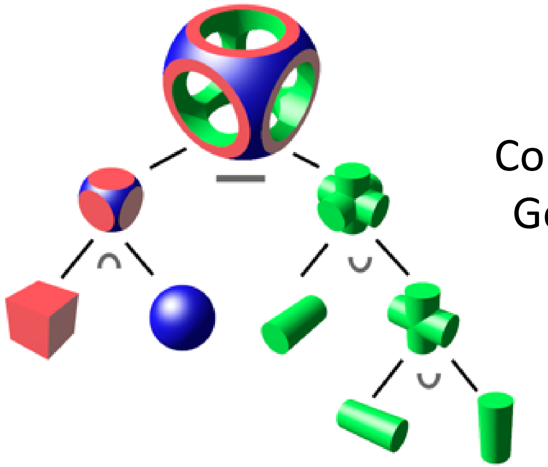
Voxels

Multiple View Images
RGB(D)

These are irregular representations – and the ones most commonly used in 3D apps

Sketch-Extrude

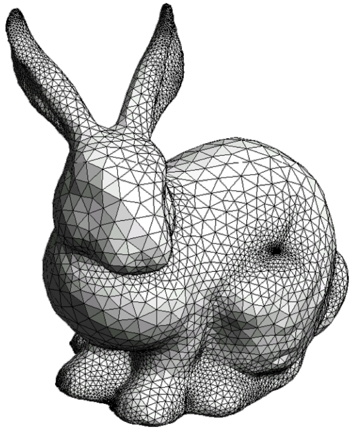


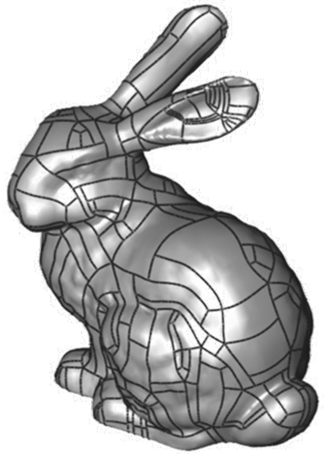


Constructive Solid Geometry (CSG)

3D: Boundary or Volumetric?

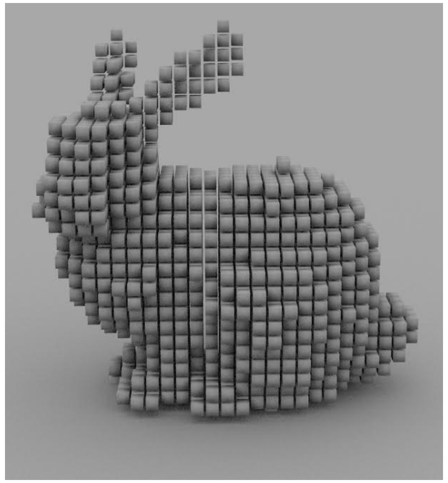
- B(oundary)-Reps

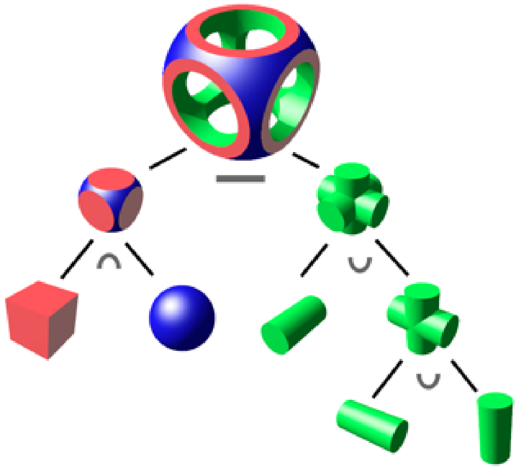




  - more efficient
  - closer to semantics, support local editing

- V(olume)-Reps

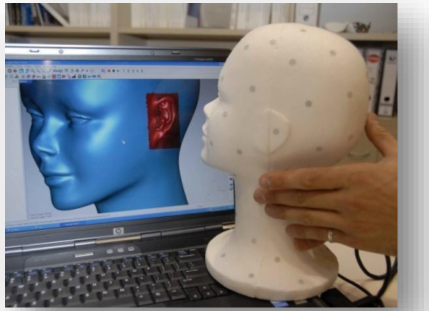


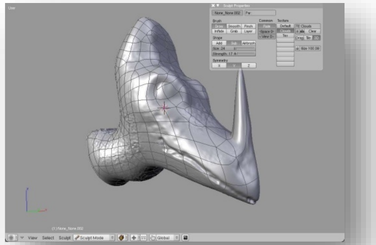
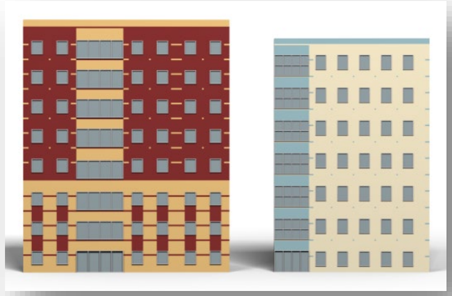


  - more regular
  - support unions and intersections

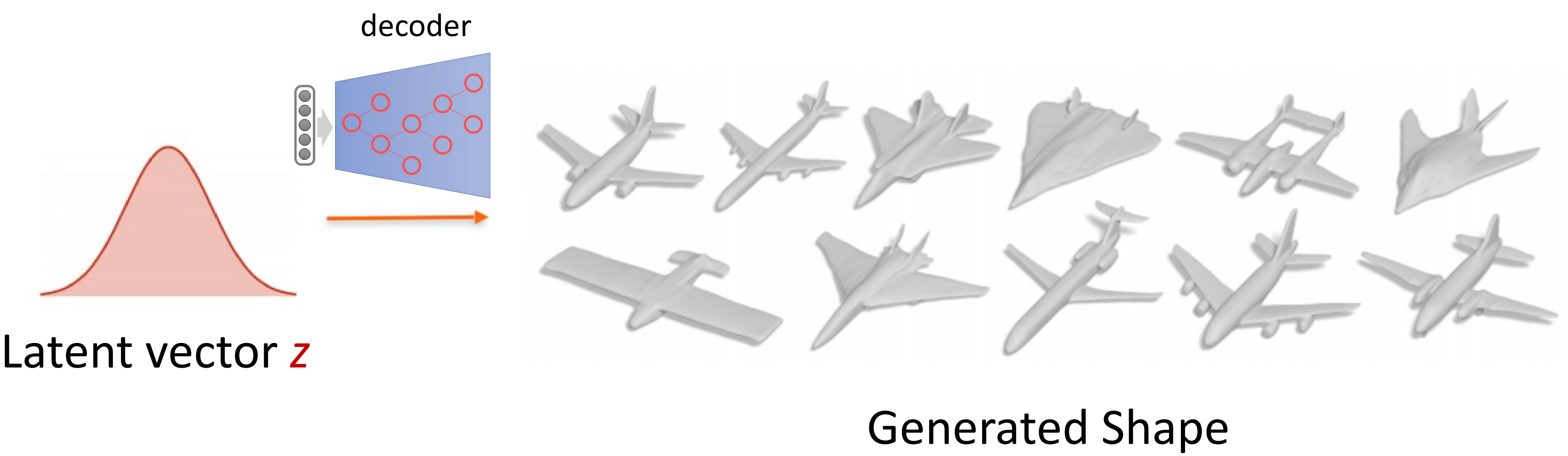
Many Reps, Because 3D has Many Sources

- Acquired real-world objects:
  - RGB or RGBD data, scanning
  - Points, meshes
- Modeling "by hand":
  - Higher-level representations, amendable to modification, control
  - Parametric surfaces, subdivision surfaces, implicits
- Procedural modeling
  - Algorithms, grammars
  - Primitives, Polygons, application-dependent elements
- Neural generators …

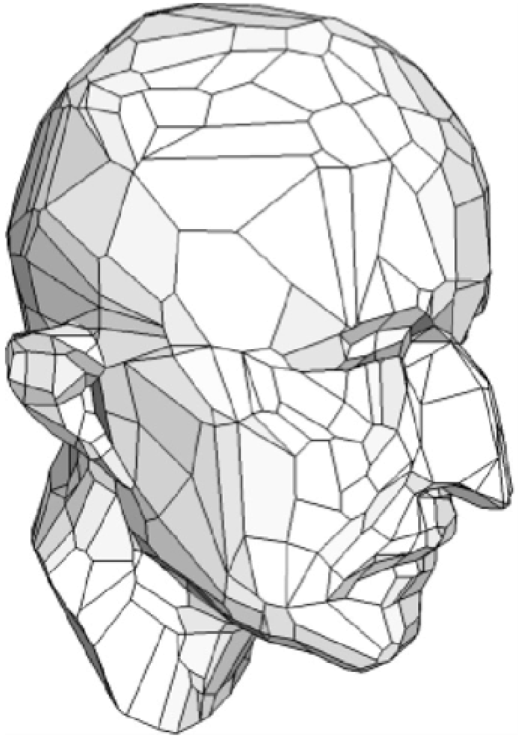

ML Decoding/Generation from Latent Vectors



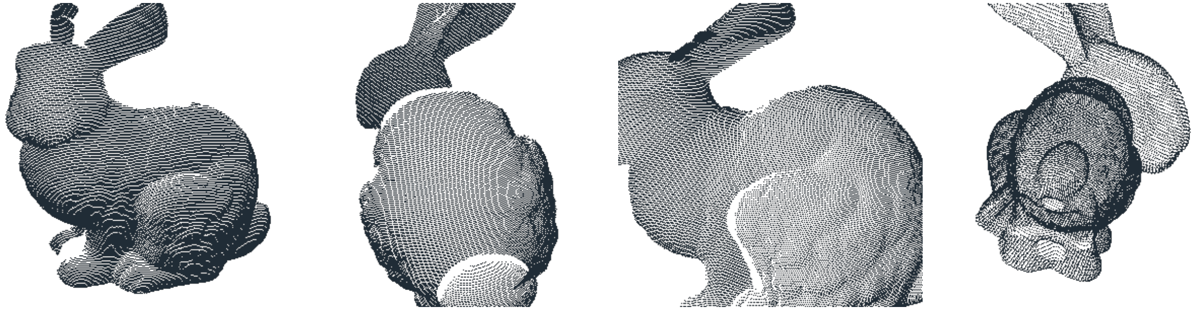
Generator/Decoder: generating shapes from latent vectors via deep networks – but in what format?

Representation Considerations

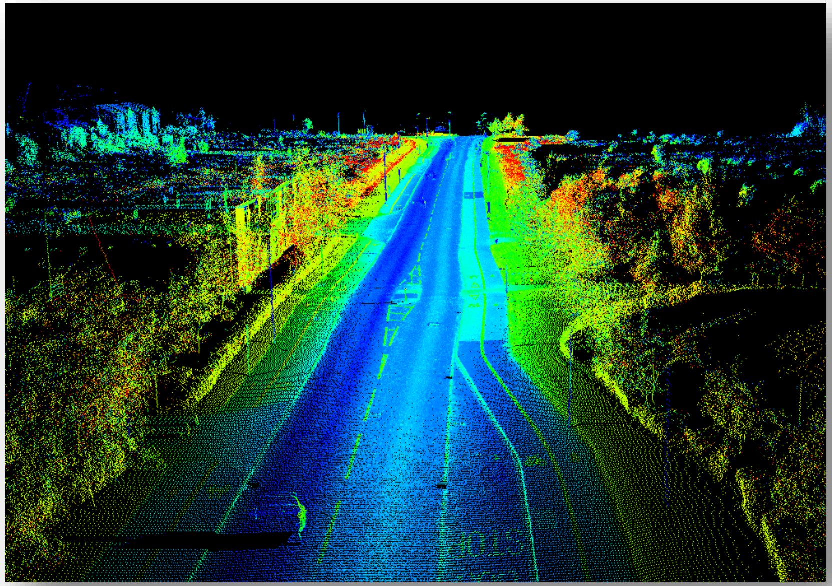
- How should we represent geometry?
  - Be derivable from sensor data
  - Support storage efficiency
  - Support editing:
    - Modification, simplification, smoothing, filtering, repair…
  - Support creativity:
    - Input metaphors, UIs…
  - Support rendering:
    - Rasterization, raytracing…
  - Support ML:
    - Share info across related shapes
    - How much manipulation can we do in the latent domain?



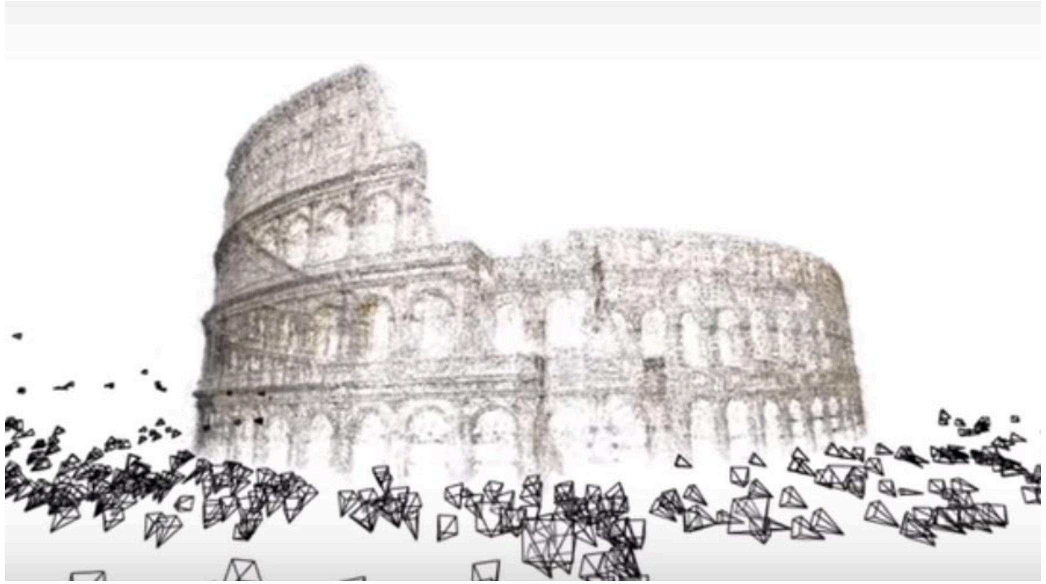
Point Clouds



3D Point Clouds from Many Sensors



Lidar point clouds (LizardTech)

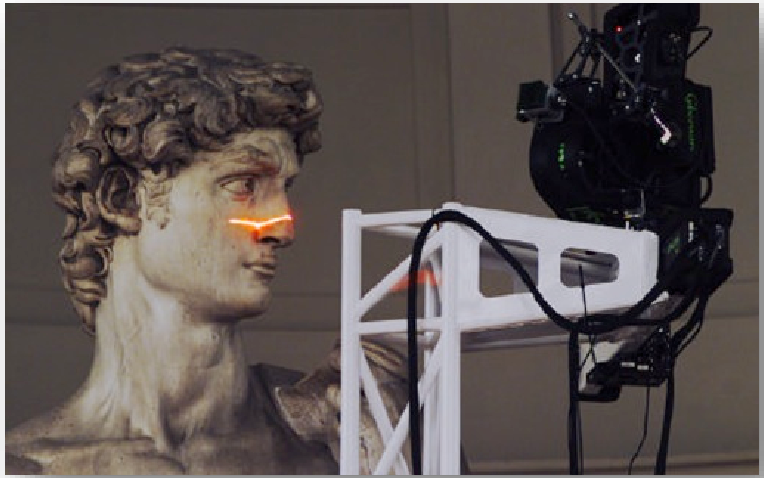


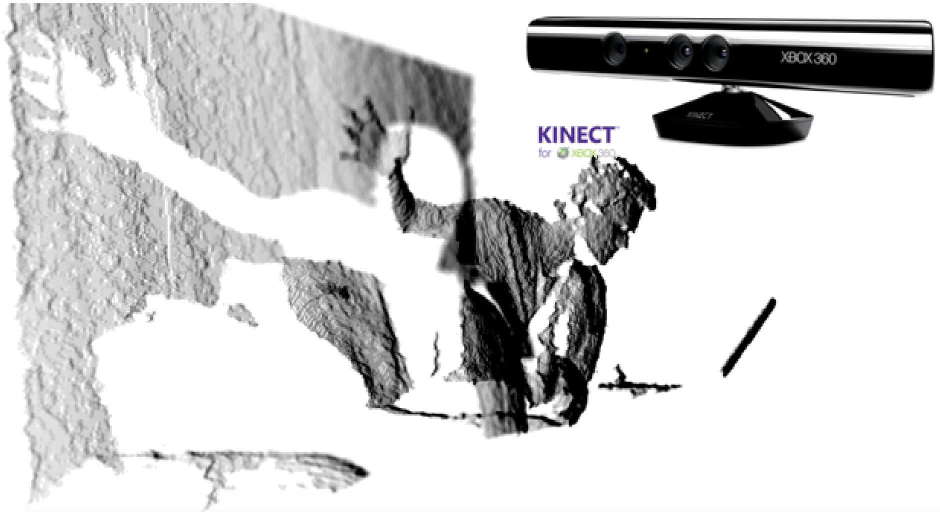
Structure from motion (Microsoft)

Depth camera (Intel, Microsoft, Google)

14

Output of Acquisition Process





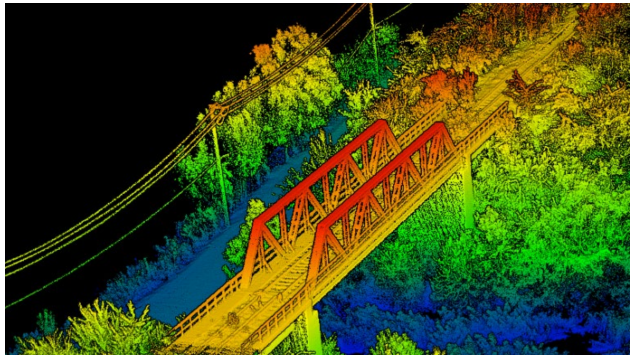
Triangulation, time-of-flight, structured light scanners

but also from classic computer algorithms like stereo

3D Point Cloud Data

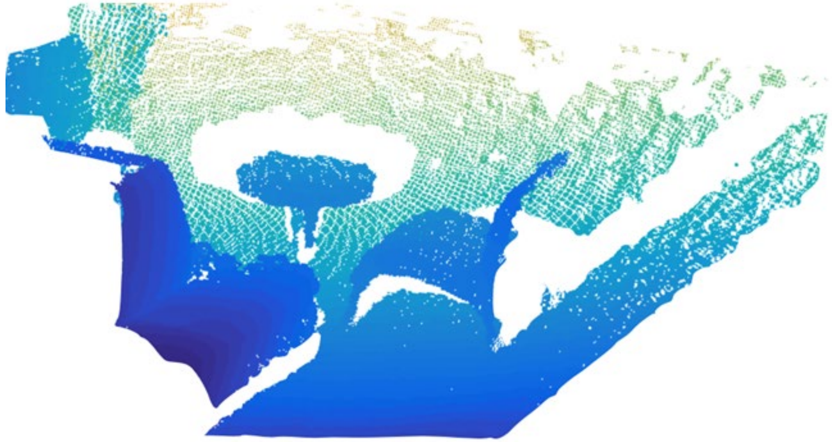
- Close to raw sensor data
- Representationally simple
- Irregular neighborhoods
- Variable density

LiDAR



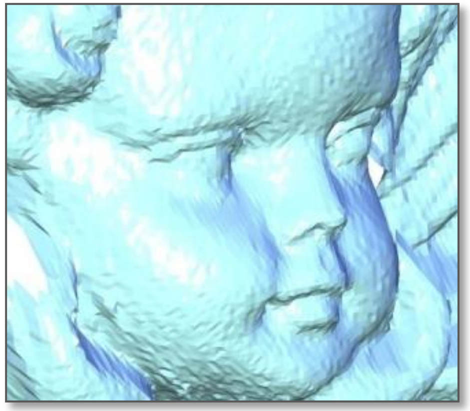
Depth Sensor

Point Cloud



Point Cloud Issues

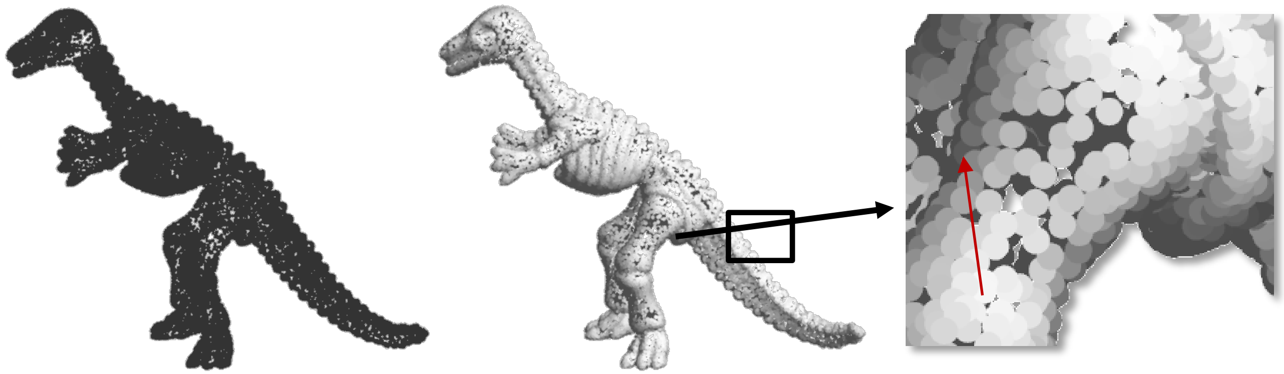
- Standard 3D data from a variety of sources/scanners -- but
  - Irregular, variable density
  - Potentially noisy



  - Can have holes
  - Registration of multiple images is required

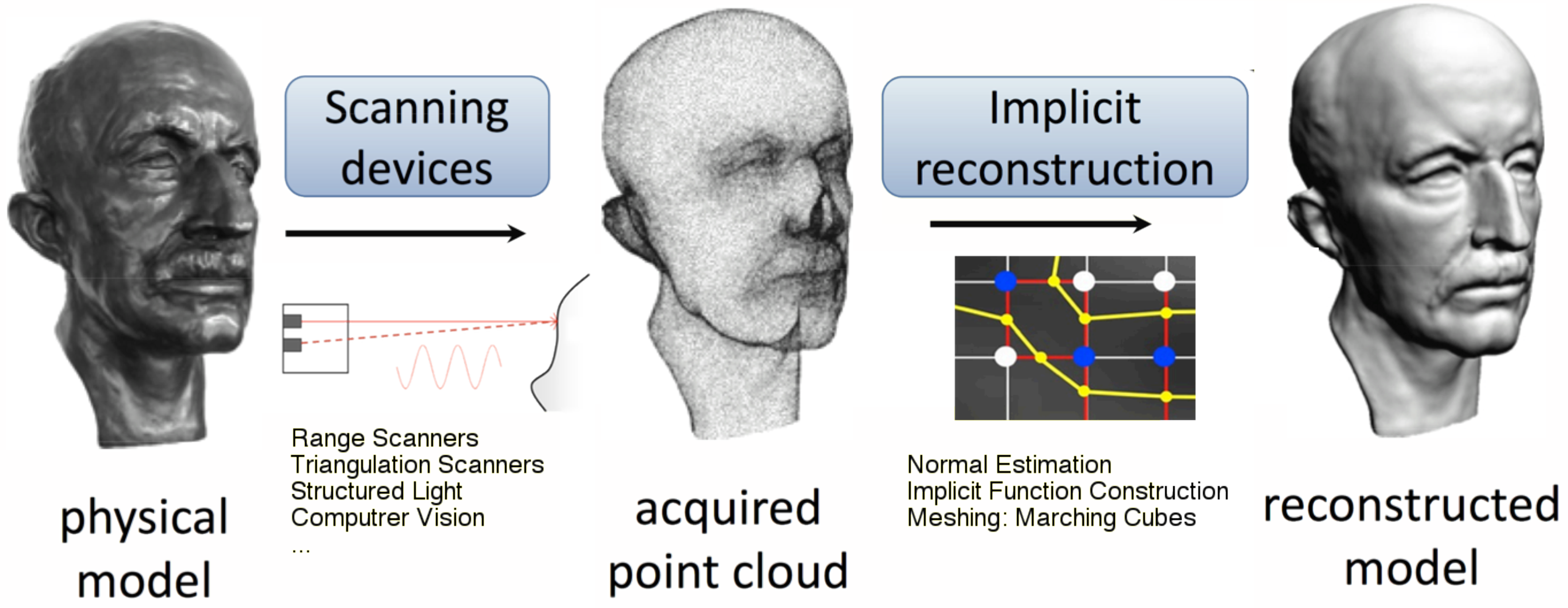
Point Clouds, Often an Intermediate Representation

- Points = unordered set of 3-tuples – no structure
- Frequently converted to other reps
  - Meshes, implicits, parametric surfaces
  - Easier to process, edit and/or render
- Efficient point processing / modeling requires spatial partitioning data structure
  - E.g., to figure out neighborhoods for normal estimation



shading needs normals!

3D Point Cloud Processing



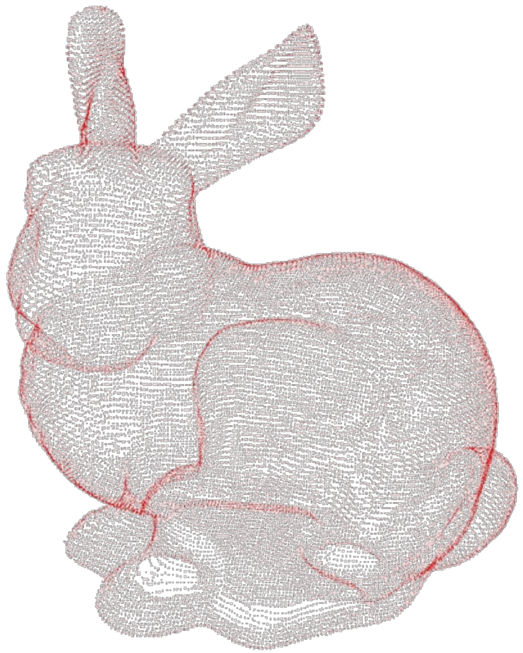
Traditional 3D Acquisition Pipeline

We'll see how to apply ML directly on Point Cloud Data

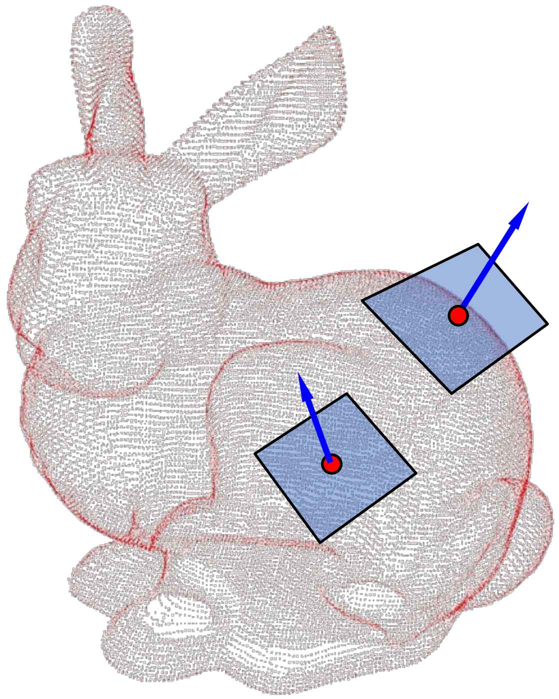
Stanford Bunny

Point Clouds

- Simplest representation: **only points,** no connectivity
- Collection of (x,y,z) coordinates, possibly with normals

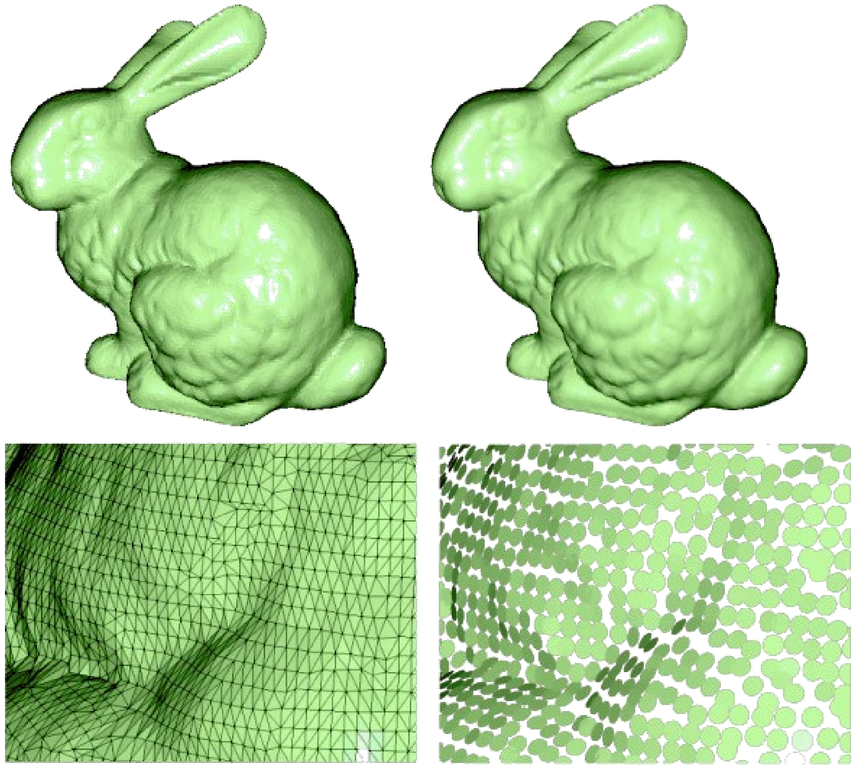


Stanford bunny



Point Clouds

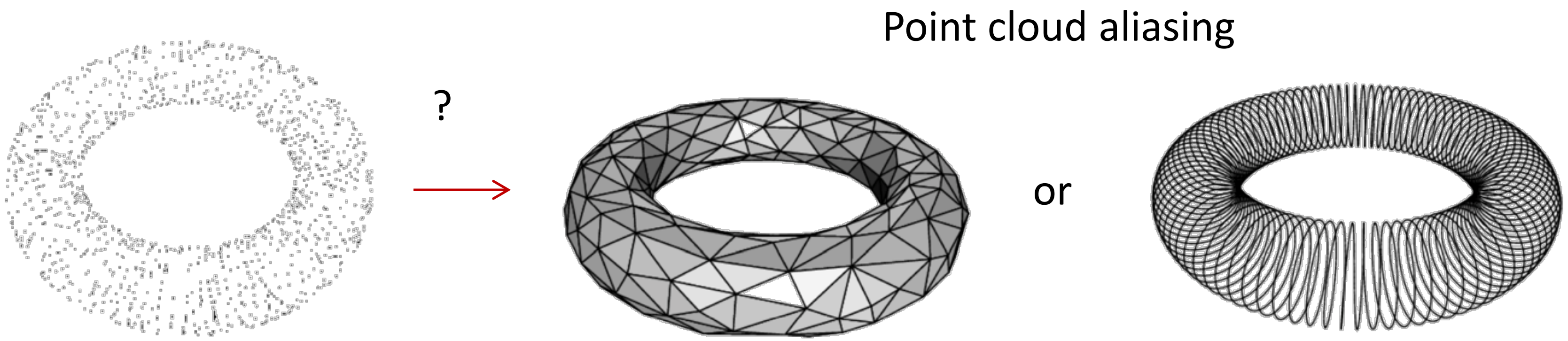
- Simplest representation: **only points,** no connectivity
- Collection of (x,y,z) coordinates, possibly with normals
- Points with orientation are called **surfels**



Filip van Bouwel

Point Clouds

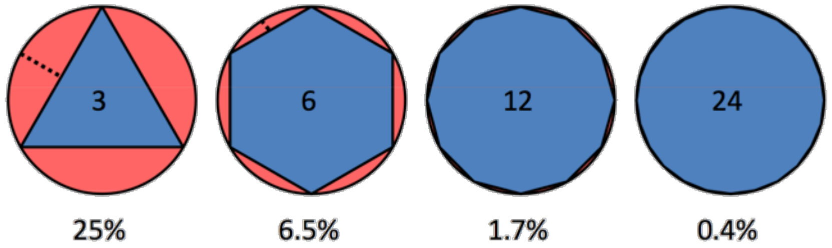
- Simplest representation: **only points,** no connectivity
- Collection of (x,y,z) coordinates, possibly with normals
- Points with orientation are called **surfels**
- Several limitations:
  - **no** simplification or subdivision
  - **no** direct smooth rendering
  - **no** topological information



Point Clouds

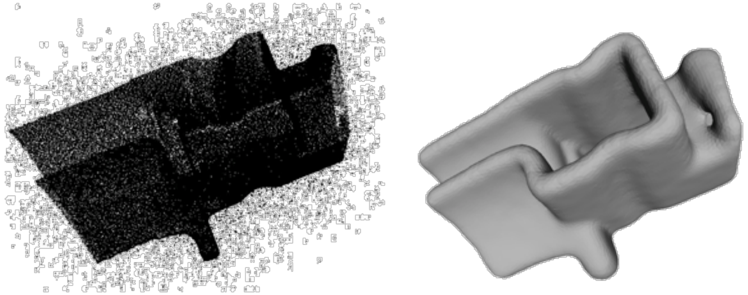
- Simplest representation: **only points,** no connectivity
- Collection of (x,y,z) coordinates, possibly with normals
- Points with orientation are called **surfels**
- Several limitations:
  - **no** simplification or subdivision
  - **no** direct smooth rendering
  - **no** topological information
  - weak approximation power: $O(h)$ for point clouds
    - need *square* number of points for the same approximation power as meshes

- Piecewise linear approximation
  - Error is $O(h^2)$



Point Clouds

- Simplest representation: **only points,** no connectivity
- Collection of (x,y,z) coordinates, possibly with normals
- Points with orientation are called **surfels**
- Several limitations:
  - **no** Simplification or subdivision
  - **no** direct smooth rendering
  - **no** topological information
  - **weak** approximation power
  - **noise** and **outliers**

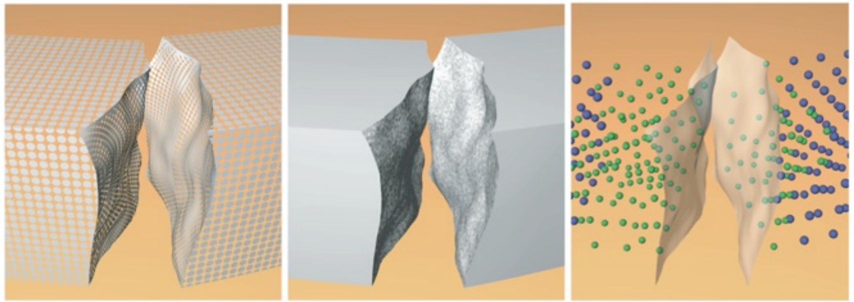


- But NNs can compensate for many of these limitations

Why Point Clouds?

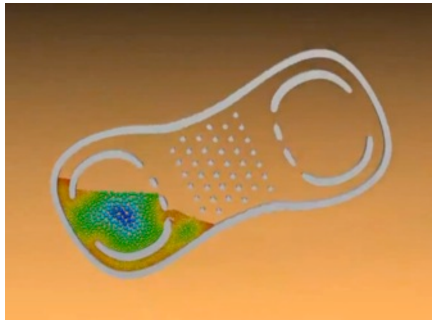
1) Typically, that's the only thing that's available from a large class of sensors
2) Isolation: sometimes, easier to handle (esp. in hardware).

Fracturing Solids



Meshless Animation of Fracturing Solids
Pauly et al., SIGGRAPH '05

Fluids



Adaptively sampled particle fluids,
Adams et al. SIGGRAPH '07

Point Cloud Processing

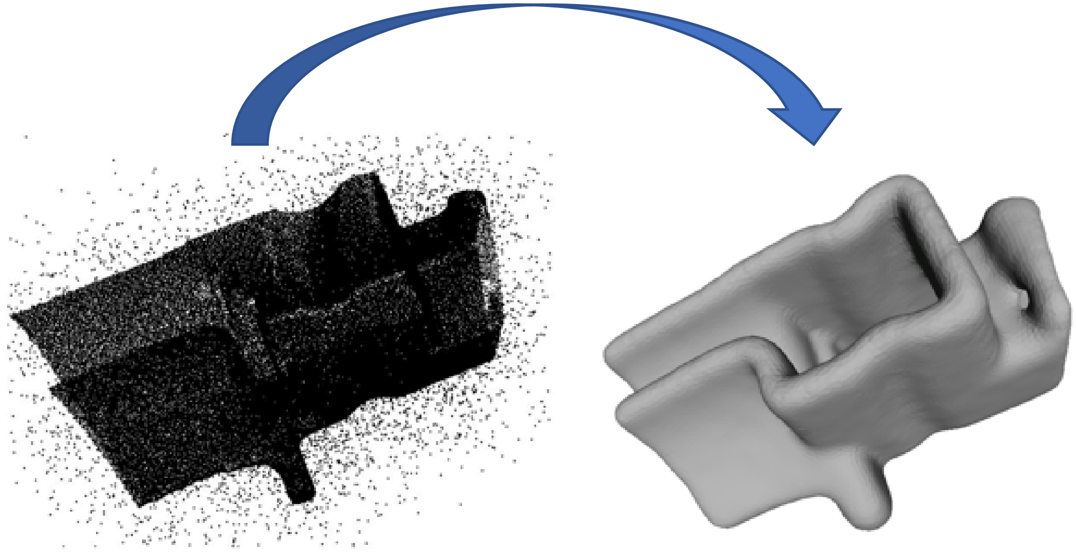
Geometry Processing

→ CS348a

3D Point Cloud Processing

Typically point cloud sampling of a shape is insufficient for most applications. Main stages in processing:

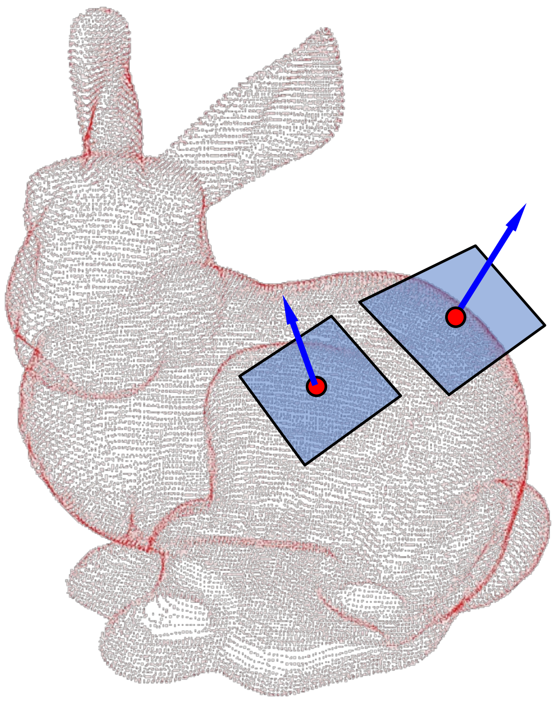
1. Outlier removal – throw away samples from non-surface areas
2. If we have multiple scans, align them
3. Smoothing – remove local noise
4. Estimate surface normals
5. Surface reconstruction
   - Implicit representation
   - Triangle mesh extraction



Normal Estimation and Outlier Removal

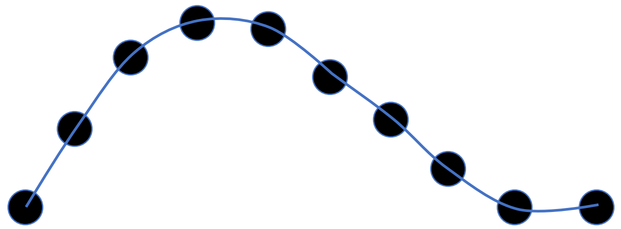
Fundamental problems in point cloud processing.

Although seemingly very different, can be solved with the same general approach – look at the “shape of neighborhoods” …



Normal Estimation

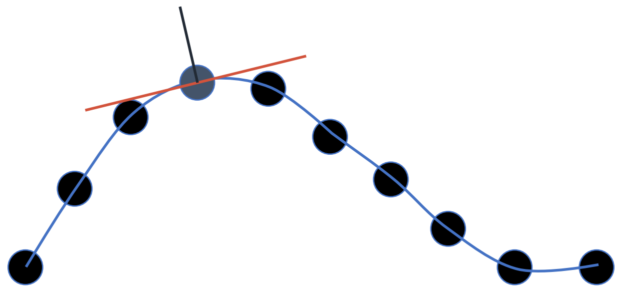
Assume we have a clean sampling of the surface. OK, start with a curve.



Our goal is to find the best approximation of the tangent direction, and thus of the normal to the curve.

Normal Estimation

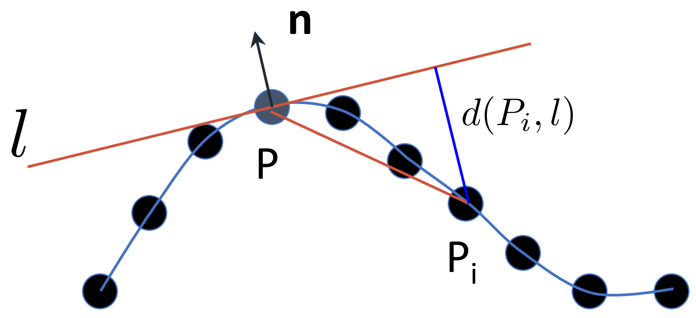
Assume we have a clean sampling of the surface. OK, start with a curve.



Our goal is to find the best approximation of the tangent direction, and thus of the normal to the line.

Normal Estimation

Assume we have a clean sampling of the surface. OK, start with a curve.



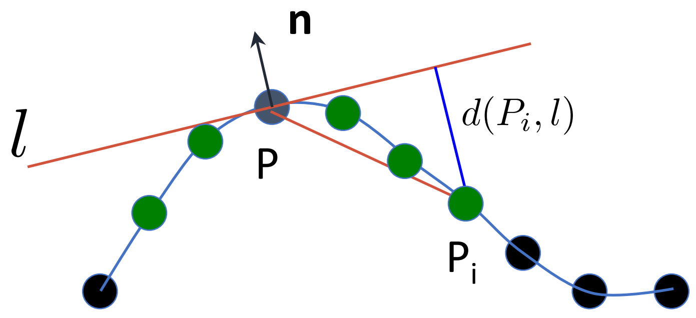
Goal: find best approximation of the normal at P.

Method: Given line l through P with normal **n**, for another point p_i:

$$d(p_i, l)^2 = \frac{((p_i - P)^T \mathbf{n})^2}{\mathbf{n}^T \mathbf{n}} = ((p_i - P)^T \mathbf{n})^2 \text{ if } \|\mathbf{n}\| = 1$$

Normal Estimation

Assume we have a clean sampling of the surface. OK, start with a curve.



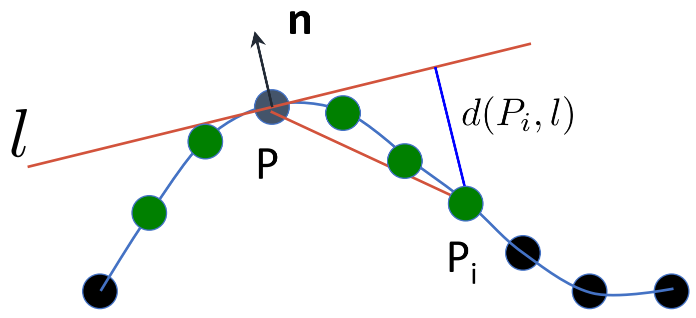
Goal: find best approximation of the normal at P.

Method: Find **n**, minimizing $\sum_{i=1}^{k} d(p_i, l)^2$ for a set of *k* points near P (e.g. *k* nearest neighbors of P).

$$\mathbf{n}_{\text{opt}} = \arg\min_{\|\mathbf{n}\|=1} \sum_{i=1}^{k} ((p_i - P)^T \mathbf{n})^2$$

Normal Estimation

Assume we have a clean sampling of the surface. OK, start with a curve.



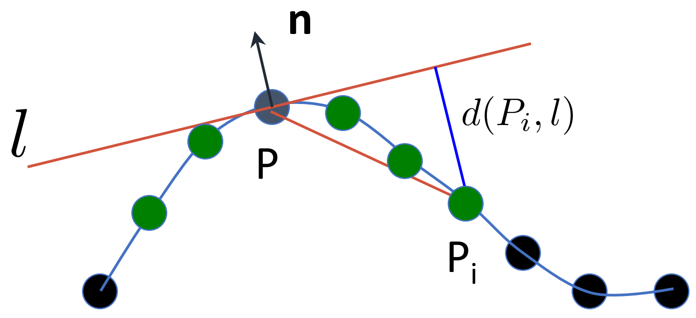
Using Lagrange multipliers:

$$\frac{\partial}{\partial \mathbf{n}}\left(\sum_{i=1}^{k}((p_i - P)^T\mathbf{n})^2\right) - \lambda\frac{\partial}{\partial \mathbf{n}}\left(\mathbf{n}^T\mathbf{n}\right) = 0$$

$$\sum_{i=1}^{k} 2(p_i - P)(p_i - P)^T\mathbf{n} = 2\lambda\mathbf{n}$$

Normal Estimation

Assume we have a clean sampling of the surface. OK, start with a curve.



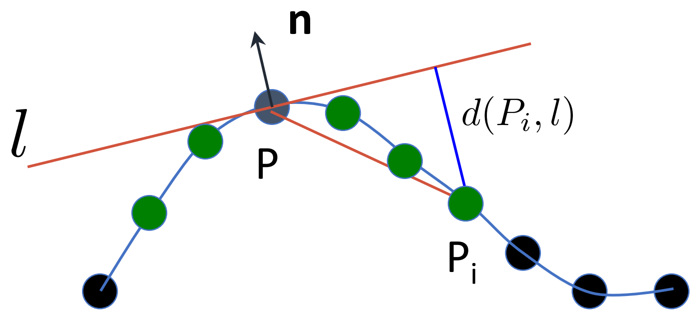
Using Lagrange multipliers:

$$\frac{\partial}{\partial \mathbf{n}}\left(\sum_{i=1}^{k}((p_i - P)^T\mathbf{n})^2\right) - \lambda\frac{\partial}{\partial \mathbf{n}}\left(\mathbf{n}^T\mathbf{n}\right) = 0$$

$$\left(\sum_{i=1}^{k}(p_i - P)(p_i - P)^T\right)\mathbf{n} = \lambda\mathbf{n} \quad \Rightarrow \quad C\mathbf{n} = \lambda\mathbf{n}$$

Normal Estimation

Assume we have a clean sampling of the surface. OK, start with a curve.



The normal **n** must be an eigenvector of the local covariance matrix:

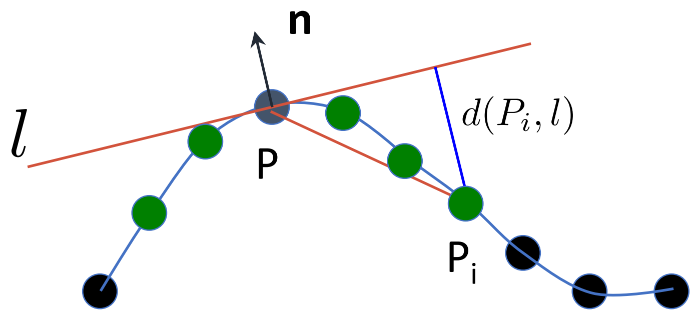
$$C\mathbf{n} = \lambda\mathbf{n} \qquad C = \sum_{i=1}^{k}(p_i - P)(p_i - P)^T$$

Moreover, since:

$$\mathbf{n}_{\text{opt}} = \arg\min_{\|\mathbf{n}\|=1} \sum_{i=1}^{k}((p_i - P)^T\mathbf{n})^2 = \arg\min_{\|\mathbf{n}\|=1} \mathbf{n}^T C\mathbf{n}$$

Normal Estimation

Assume we have a clean sampling of the surface. OK, start with a curve.



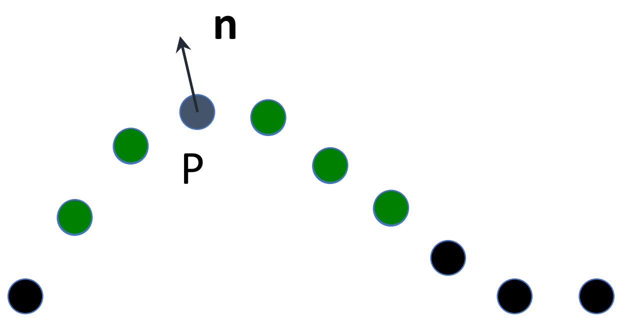
The normal **n** must be an eigenvector of the matrix:

$$C\mathbf{n} = \lambda\mathbf{n} \qquad C = \sum_{i=1}^{k}(p_i - P)(p_i - P)^T$$

So, $\mathbf{n}_{opt}$ must be the eigenvector corresponding to the **smallest eigenvalue** of C.

Normal Estimation

Method Outline (PCA):



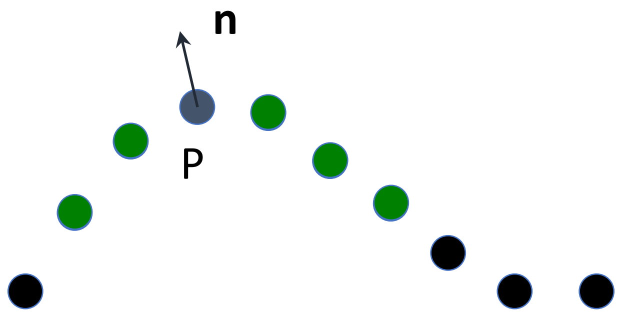
1. Given a point P in the point cloud, find its *k* nearest neighbors.
2. Compute the local covariance matrix

$$C = \sum_{i=1}^{k} (p_i - P)(p_i - P)^T$$

3. **n:** eigenvector corresponding to the smallest eigenvalue of *C*.

Normal Estimation

Method Outline (PCA-like):



1. Given a point P in the point cloud, find its *k* nearest neighbors.
2. Compute $C = \sum_{i=1}^{k} (p_i - P)(p_i - P)^T$
3. **n:** eigenvector corresponding to the smallest eigenvalue of *C*.

Variant on the theme: use $C = \sum_{i=1}^{k} (p_i - \overline{P})(p_i - \overline{P})^T, \quad \overline{P} = \frac{1}{k} \sum_{i=1}^{k} p_i$

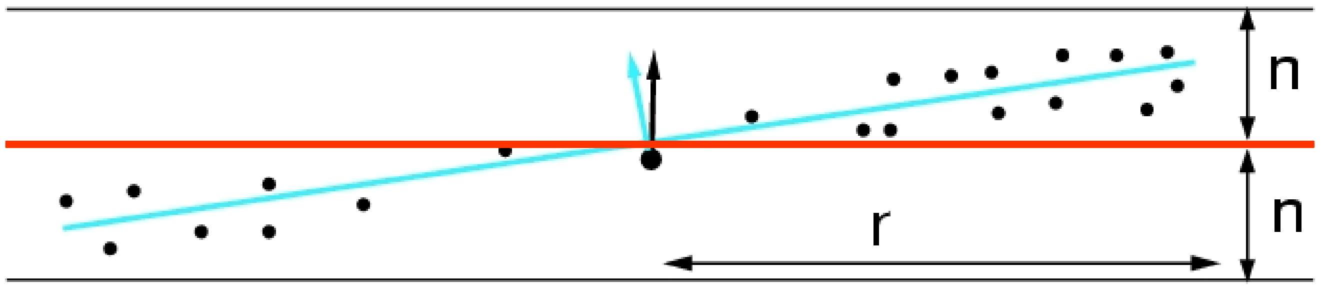
Normal Estimation

Critical parameter: *k*. Because of uneven sampling typically fix a radius *r*, and use all points **inside a ball of radius *r*.**

How to pick an optimal *r*?

Normal Estimation

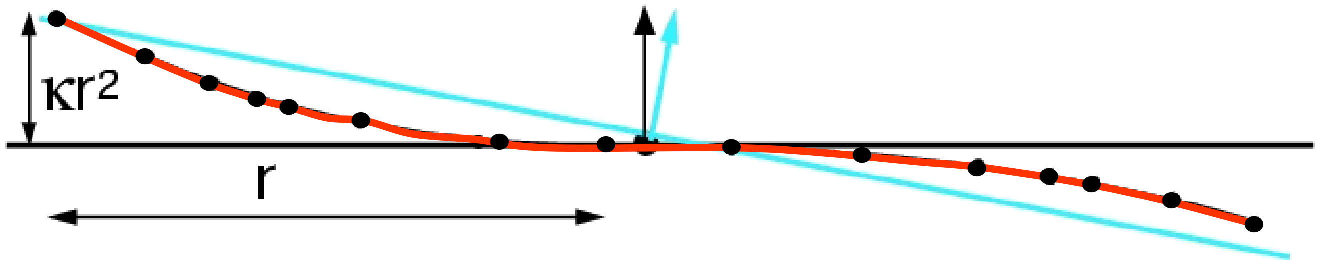
Collusive noise



Because of noise in the data, small *r* may lead to underfitting.

Normal Estimation

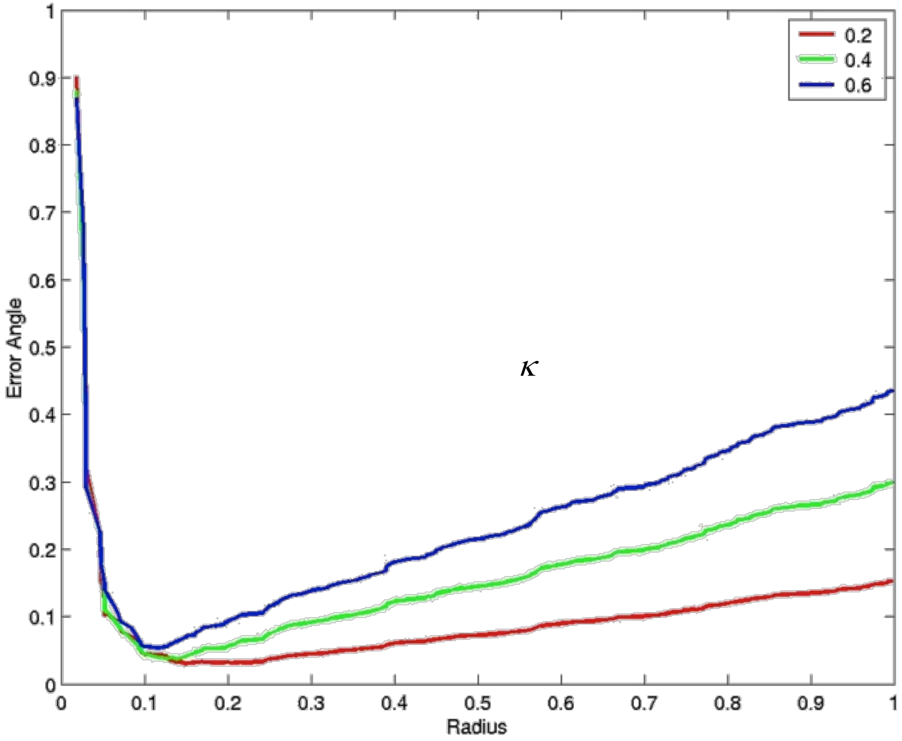
Curvature effect



Due to curvature, large *r* can lead to estimation bias.

Normal Estimation

κ



κ

κ = curvature

σ_n = noise

source: Mitra et al. '04

$$error \le \Theta(1)\kappa r + \Theta(1)\frac{\sigma_n}{\sqrt{\varepsilon\rho r^3}} + \Theta(1)\frac{\sigma_n^2}{r^2}$$

Estimation error under Gaussian noise for different values of curvature (2D)

Normal Estimation

A similar but involved analysis results in 3D,

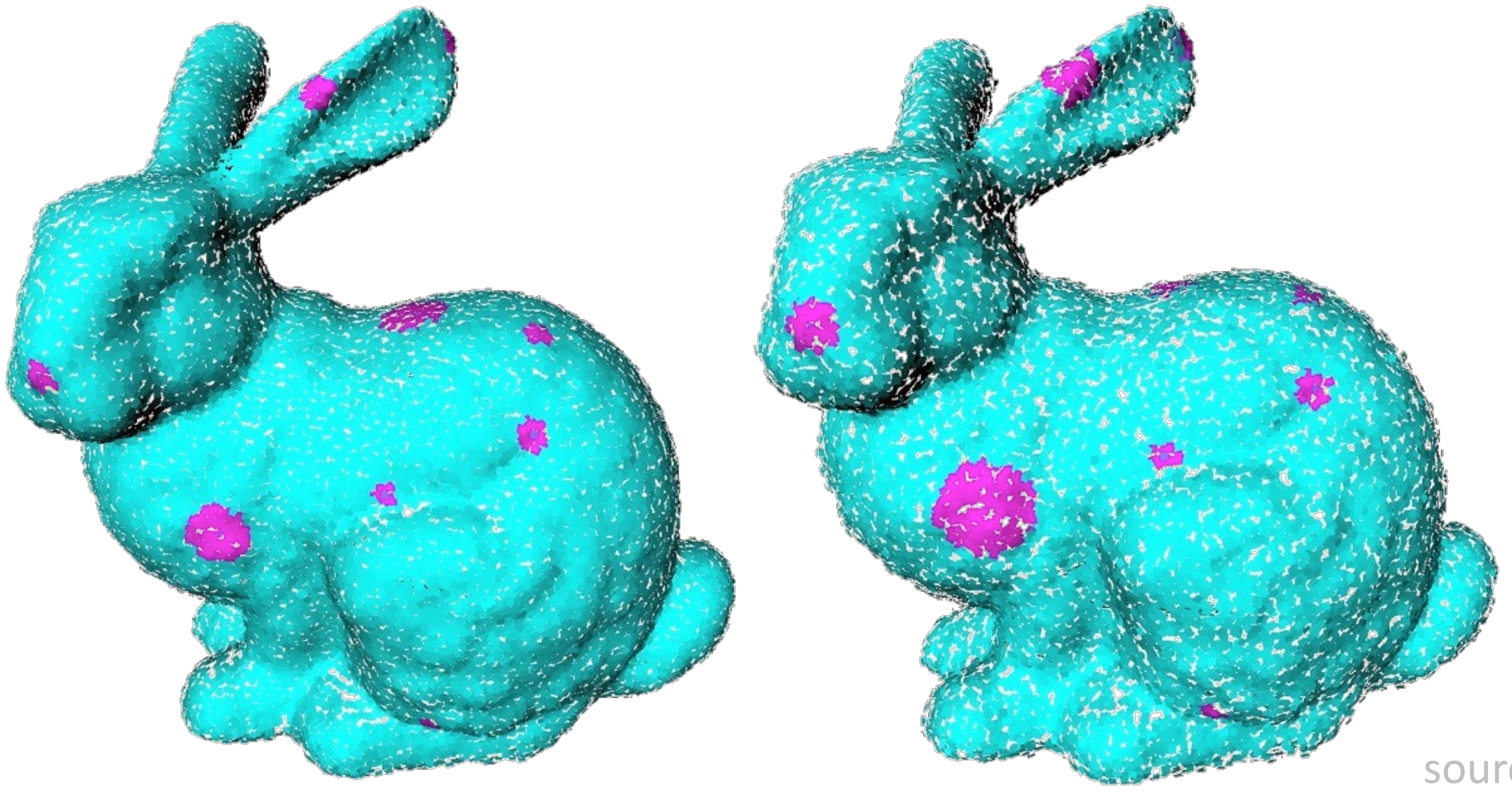
$$error \leq \Theta(1)\kappa r + \Theta(1)\sigma_n / (r^2 \sqrt{\varepsilon\rho}) + \Theta(1)\sigma_n^2 / r^2$$

A good choice of *r* is,

$$r = \left(\frac{1}{\kappa} \left(c_1 \frac{\sigma_n}{\sqrt{\varepsilon\rho}} + c_2 \sigma_n^2 \right) \right)^{1/3}$$

source: Mitra et al. '04

Normal Estimation – Optimal Neighborhood Size

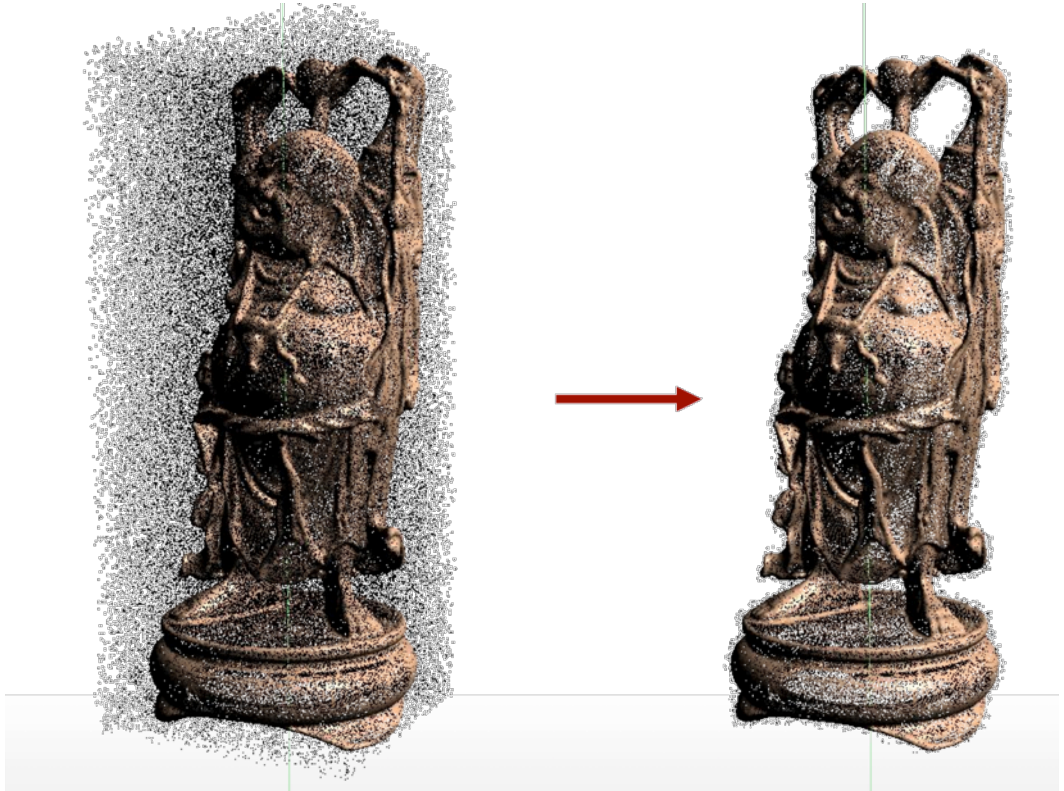


1x noise

2x noise

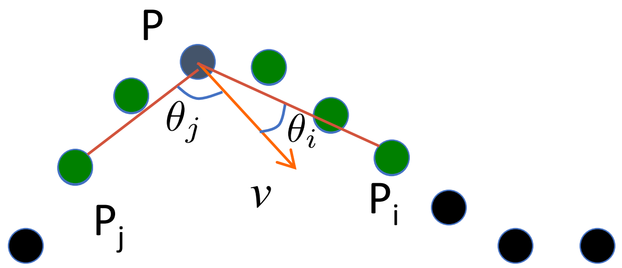
source: Mitra et al. ‘04

Outlier Removal



Goal: remove points that do not lie close to a surface.

Outlier Estimation



From the covariance matrix: $C = \sum_{i=1}^{k} (p_i - P)(p_i - P)^T$ we have:

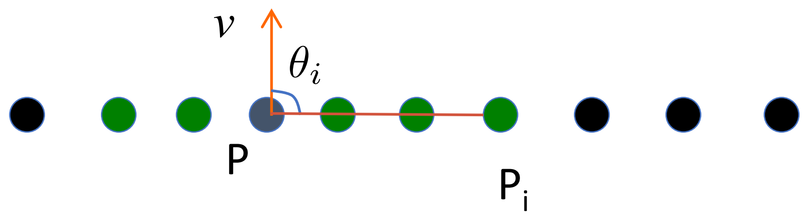
for any vector v, the Rayleigh quotient:

$$\frac{v^T C v}{v^T v} = \sum_{i=1}^{k} \left((p_i - P)^T v\right)^2 \text{ if } \|v\| = 1$$

$$= \sum_{i=1}^{k} \left(\|p_i - P\| \cos(\theta_i)\right)^2$$

Intuitively, v_{min}, maximizes the sum of angles to each vector $(p_i - P)$.

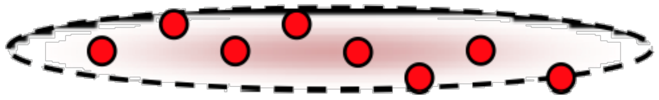
Outlier Estimation



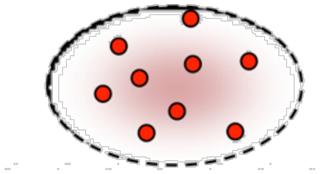
If all the points are on a line, then $\min_v \frac{v^T C v}{v^T v} = \lambda_{\min}(C) = 0$ and $\lambda_{\max}(C)$ is large.

There exists a direction along which the point cloud has no variability.

If points are scattered randomly, then: $\lambda_{\max}(C) \approx \lambda_{\min}(C)$

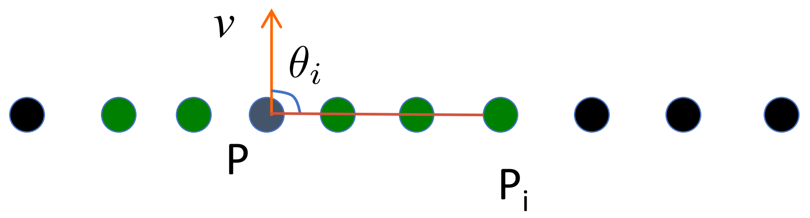


line or curve like (1D) $\frac{\lambda_1}{\lambda_2}$ small



$\frac{\lambda_1}{\lambda_2} \approx 1$ area like (2D)

Outlier Estimation



If all the points are on a line, then $\min_v \frac{v^T C v}{v^T v} = \lambda_{\min}(C) = 0$ and $\lambda_{\max}(C)$ is large.

There exists a direction along which the point cloud has no variability.

If points are scattered randomly, then: $\lambda_{\max}(C) \approx \lambda_{\min}(C)$

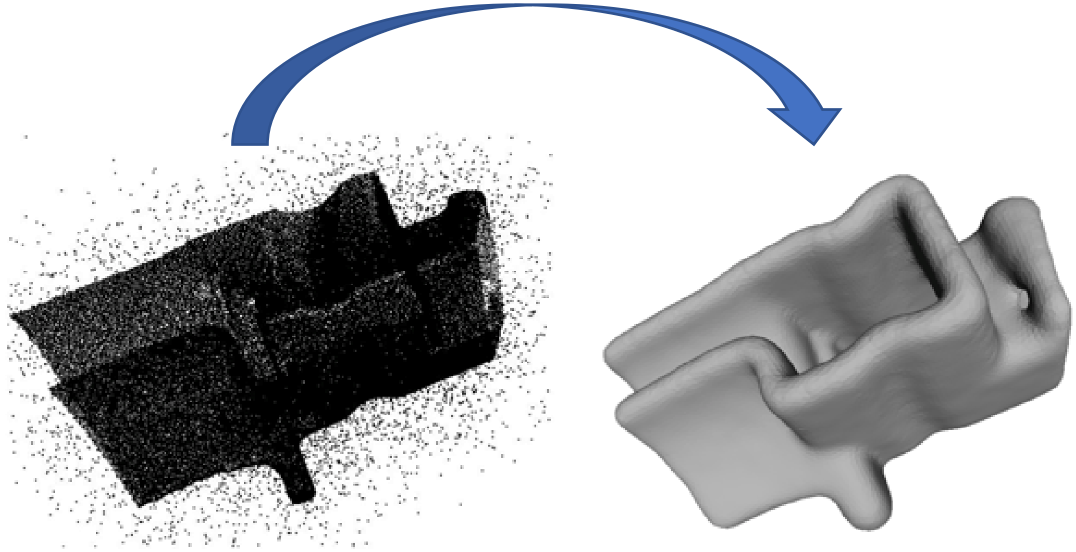
Thus, can remove points where $\frac{\lambda_1}{\lambda_2} > \epsilon$ for some threshold.

In 3D we expect two zero eigenvalues, so use $\frac{\lambda_2}{\lambda_3} > \epsilon$ for some threshold.

3D Point Cloud Processing

Typically point cloud sampling of a shape is insufficient for most applications. Main stages in processing:

1. Outlier removal – throw away samples from non-surface areas
2. If we have multiple scans, align them
3. Smoothing – remove local noise
4. Estimate surface normals
5. Surface reconstruction
   - Implicit representation
   - Triangle mesh extraction



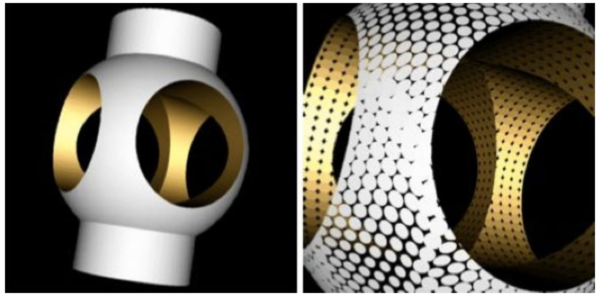
Boundary Surface Representations

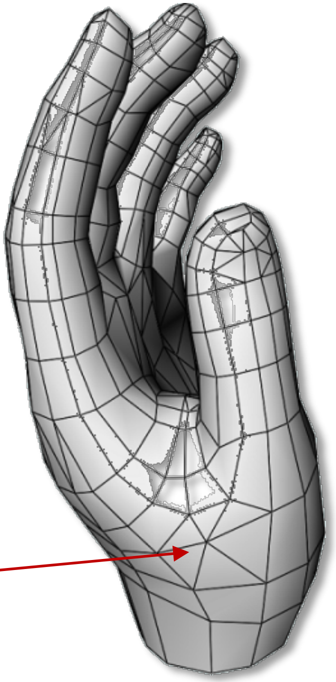
B-Reps

→ CS348a

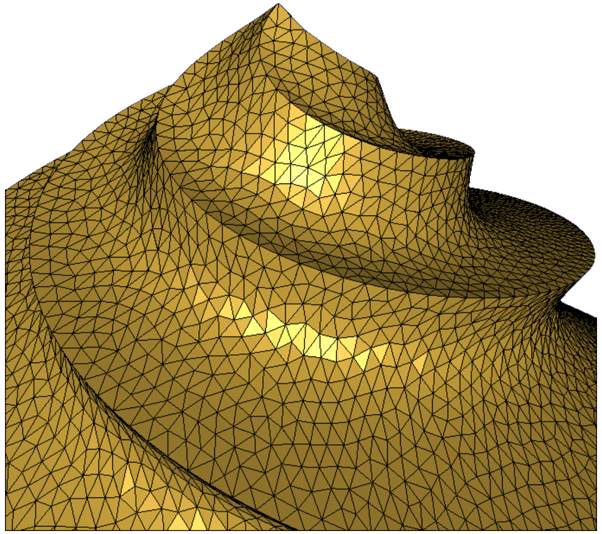
B-Reps: Low-Level Elements

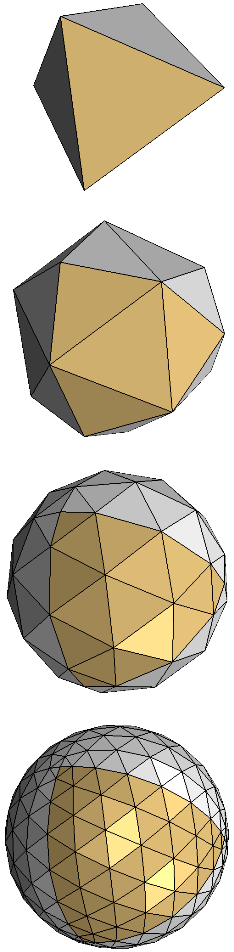
- Triangle meshes
- Quad meshes





quad meshes can be formed by grid-like quads, but there will almost always be extraordinary (singular) vertices



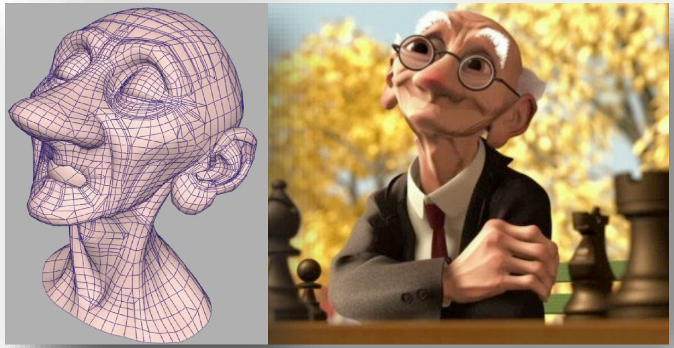


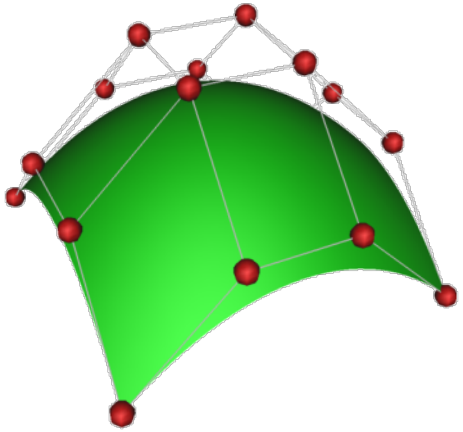
trade-offs

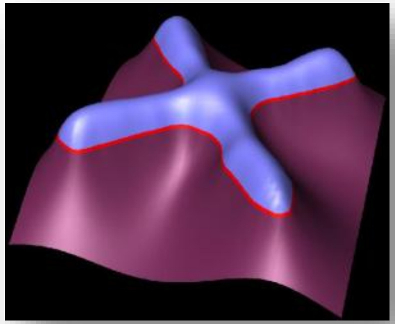
trade-offs

B-Reps: High-Level Surface Patches

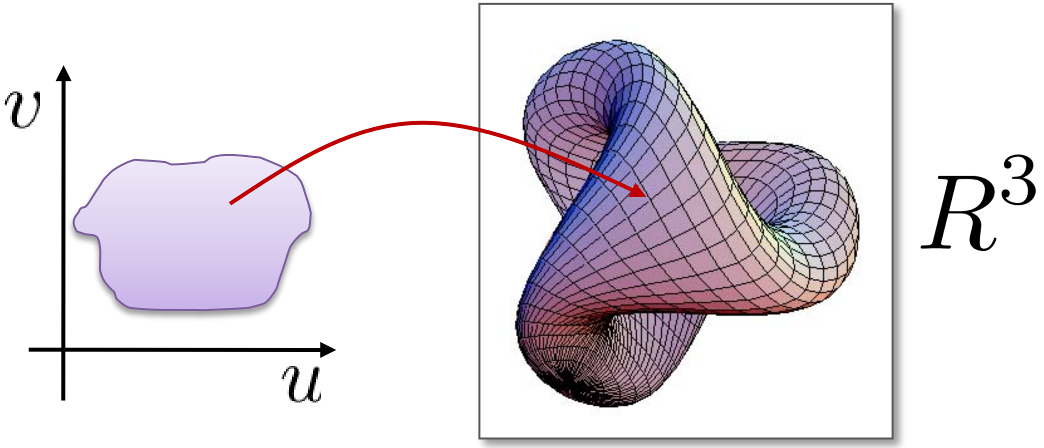
- Parametric surfaces
- Implicit functions
- Subdivision surfaces







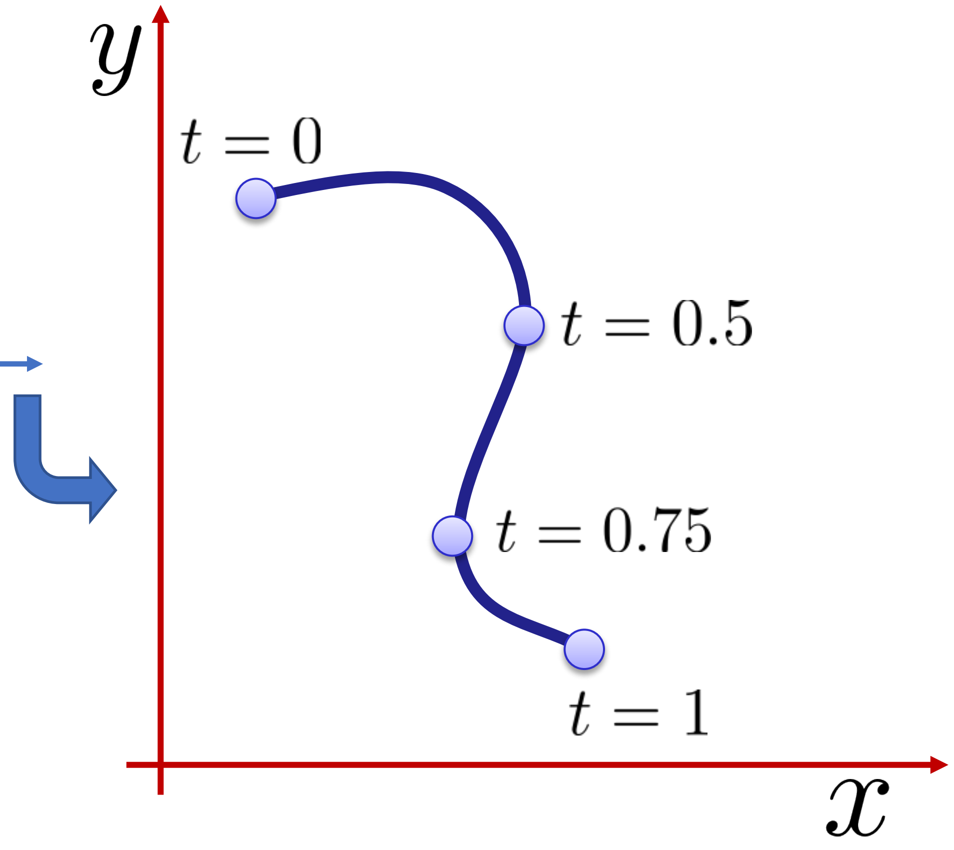
Parametric Curves and Surfaces



Parametric Representation: 1D Curves

- Range of a function $f : X \rightarrow Y, X \subseteq \mathbb{R}^m, Y \subseteq \mathbb{R}^n$
  - Planar curve: $m = 1, n = 2$

$$s(t) = (x(t), y(t))$$



  - Space curve: $m = 1, n = 3$

$$s(t) = (x(t), y(t), z(t))$$

1-D Parametric Curves

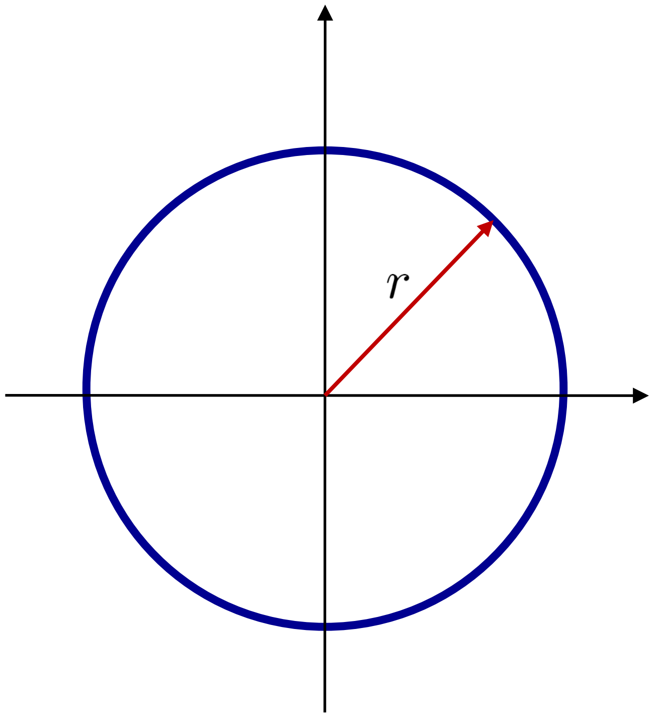
- Example: Explicit curve/circle in 2D

$$\mathbf{p} : \mathbb{R} \to \mathbb{R}^2$$

$$t \mapsto \mathbf{p}(t) = (x(t), y(t))$$

$$\mathbf{p}(t) = r\,(\cos(t), \sin(t))$$

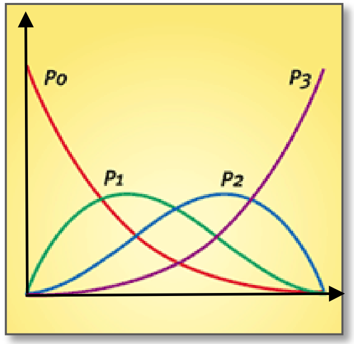
$$t \in [0, 2\pi)$$



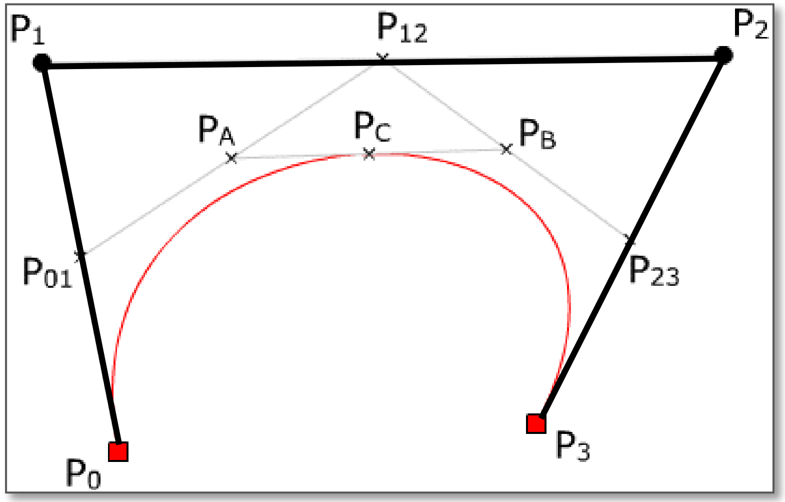
Parametric Curves: Splined Representations

- Bézier curves, splines (use multiple parametric functions)

$$s(t) = \sum_{i=0}^{n} \mathbf{p}_i B_i^n(t) \qquad B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$$



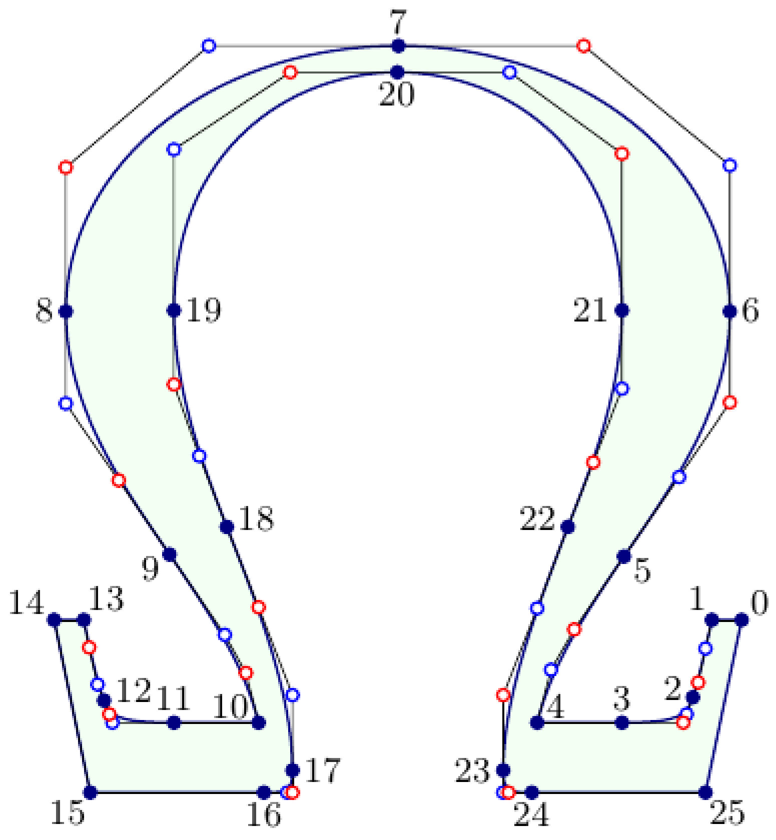
Basis functions



Cubic Bézier curve and associated control polygon

Define parametric arcs via control points

Modeling 2D Shapes with Spline Curves

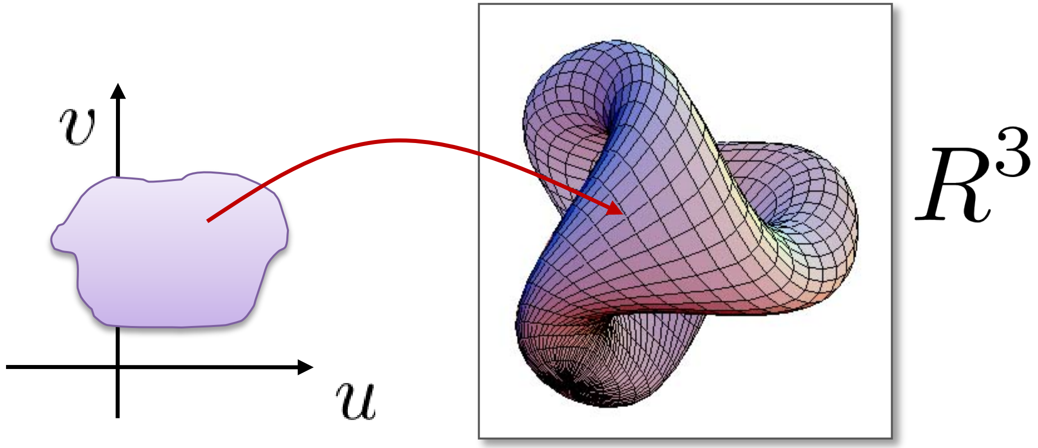


Joint many arcs to complete a closed boundary.

The fonts we use …

Parametric Representation: 2D Surfaces

- Range of a function $f : X \to Y, X \subseteq \mathbb{R}^m, Y \subseteq \mathbb{R}^n$
  - Surface in 3D: $m = 2, n = 3$

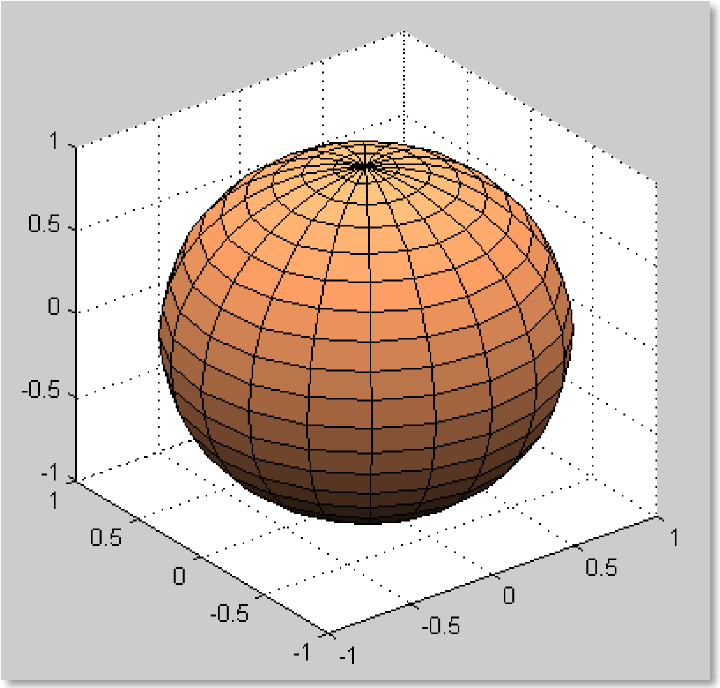


$$s(u, v) = (x(u, v), y(u, v), z(u, v))$$

Example Parametric Surface

- Sphere in 3D

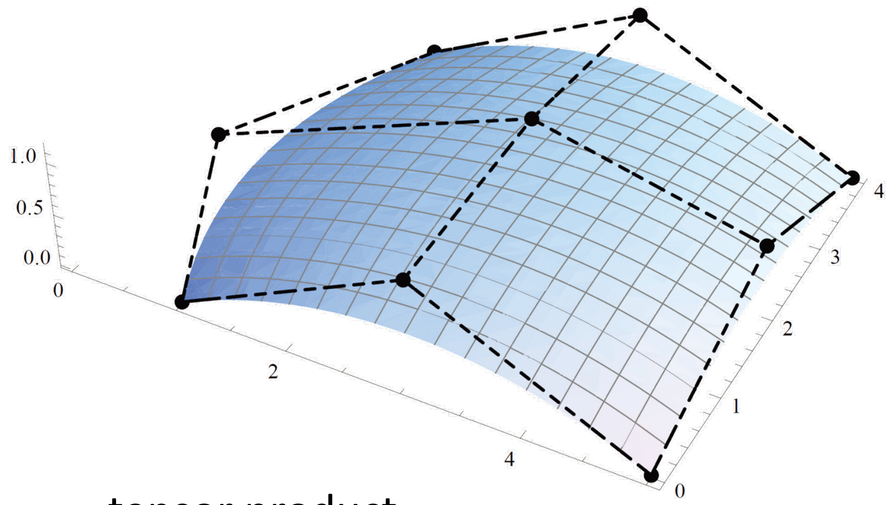
$$s : \mathbb{R}^2 \to \mathbb{R}^3$$



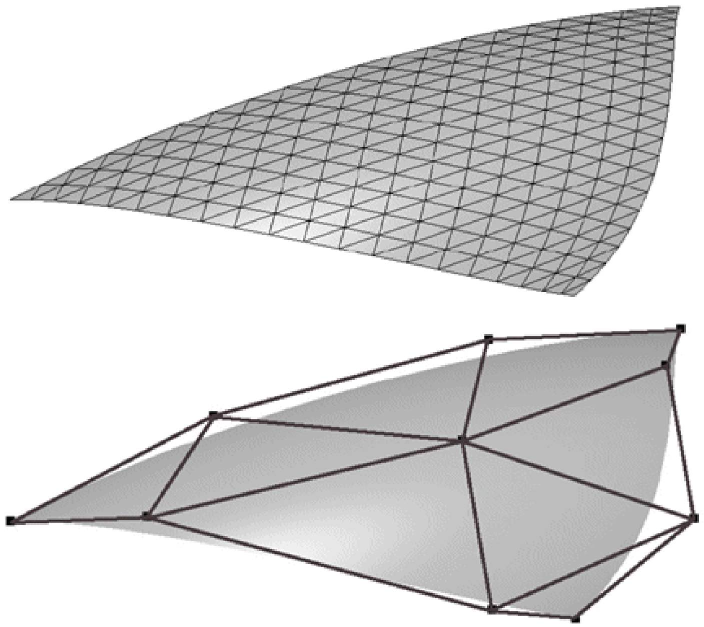
$$s(u,v) = r\left(\cos(u)\cos(v), \sin(u)\cos(v), \sin(v)\right)$$

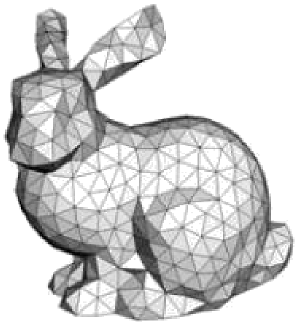
$$(u,v) \in [0, 2\pi) \times [-\pi/2, \pi/2]$$

Tensor Product vs. Triangular Patch Surfaces



tensor product,
regular quad mesh





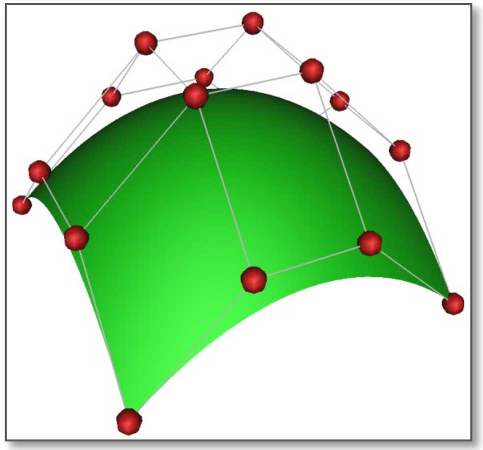
triangular patch,
triangular mesh

Different flavors of
control polyhedra

Tensor Product Parametric Surfaces

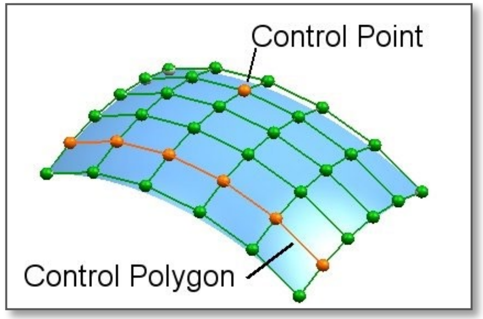
- Curve swept by another curve

$$s(u,v) = \sum_{i,j} \mathbf{p}_{i,j} B_i(u) B_j(v)$$



- Bézier surface:

$$s(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} \mathbf{p}_{i,j} B_i^m(u) B_j^n(v)$$



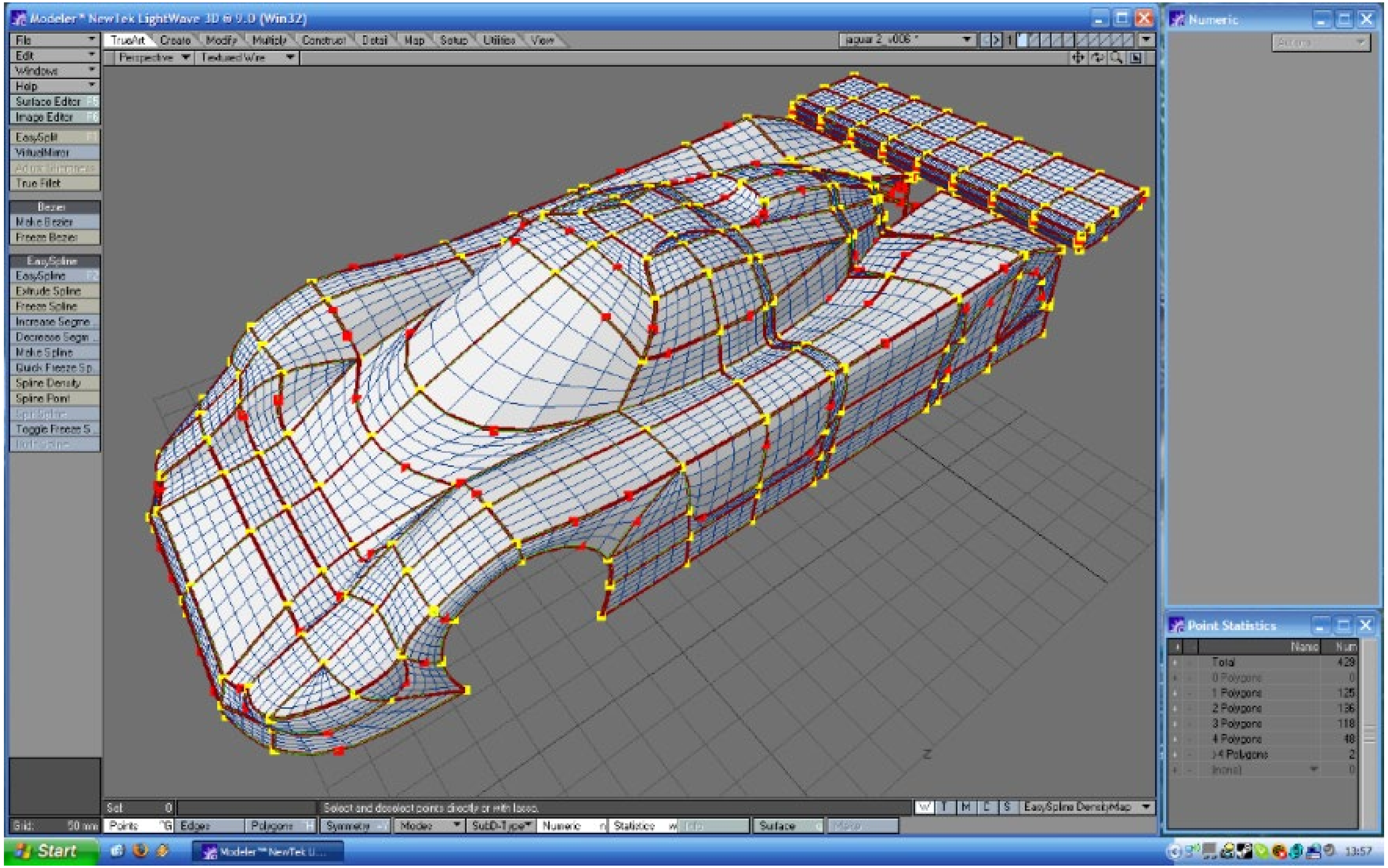
- Also : triangular patch surfaces, subdivision surfaces

Parametric Curves and Surfaces

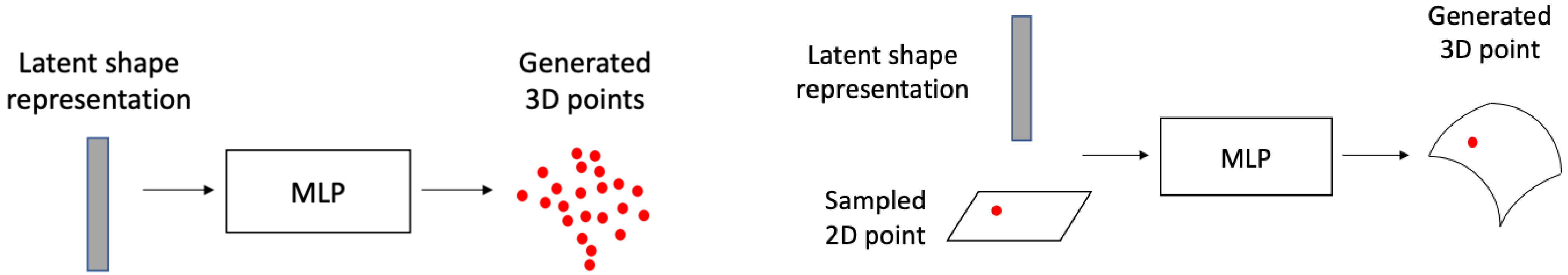
- Advantages
  - Easy to generate points on the curve/surface
  - Separate x/y/z components
  - Name each point

- Disadvantages
  - Hard to determine inside/outside
  - Hard to determine if a point is **on** the curve/surface
  - Hard to express more complex curves/surfaces!
    → therefore use piecewise parametric patches (e.g., mesh), requiring continuity constraints

Splined Surfaces for CAD



Preview: Parametric Decoder – AtlasNet



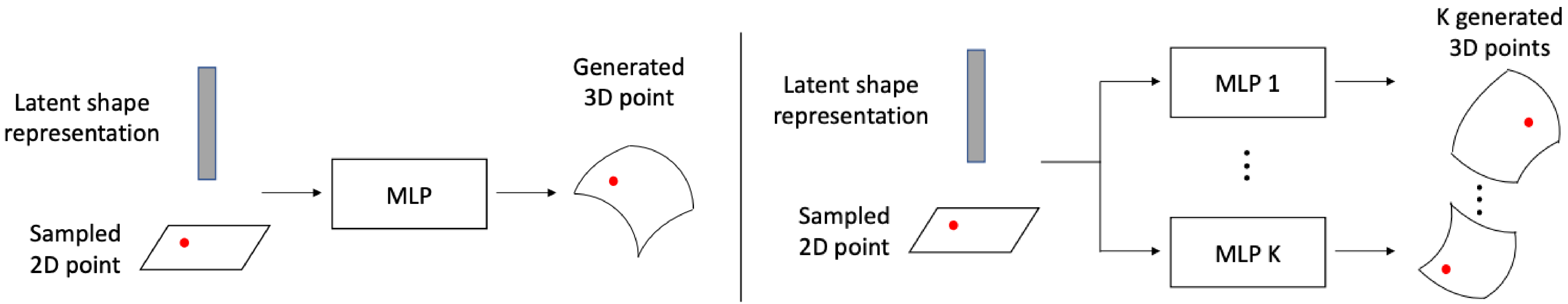
Given that the output points form a smooth surface, enforce such a parametrization in input. For each point (u, v) on the parameterization,

MLP([z, uv]) -> point

Also, you can get a **mesh!**

AtlasNet: A Papier-Machˆe Approach to Learning 3D Surface Generation, CVPR 2018

Preview: Parametric Decoder – AtlasNet

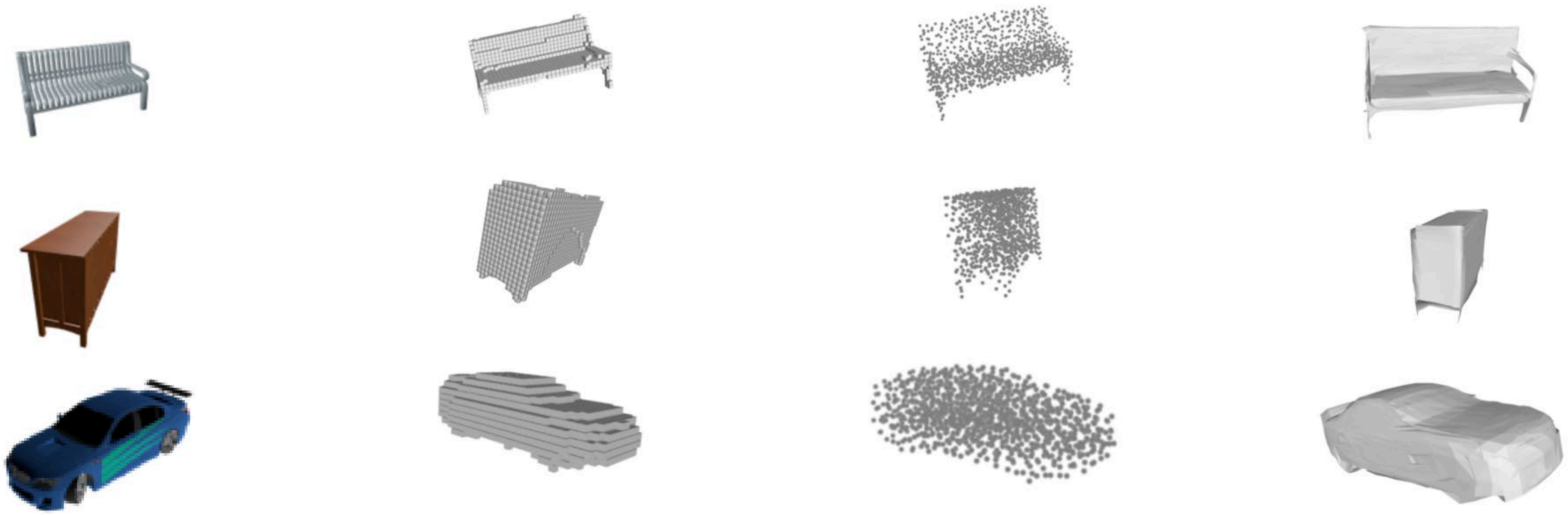


One parameterization (an **atlas**) is limited for objects with complex topology.
So, **more sheets.**

AtlasNet: A Papier-Machˆe Approach to Learning 3D Surface Generation, CVPR 2018

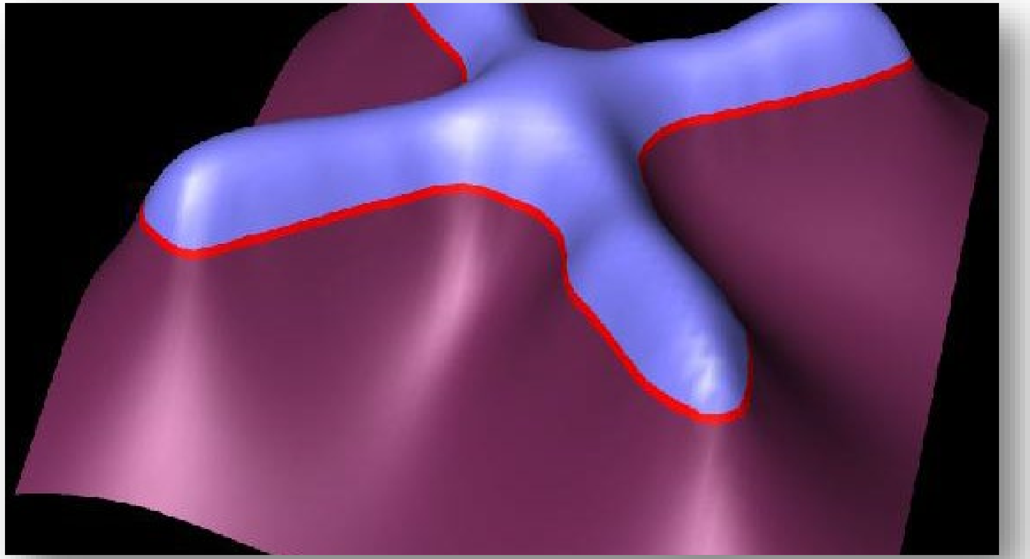
Comparison

Input image | Voxel | Point cloud | AtlasNet



AtlasNet: A Papier-Machˆe Approach to Learning 3D Surface Generation, CVPR 2018

Implicit Curves and Surfaces



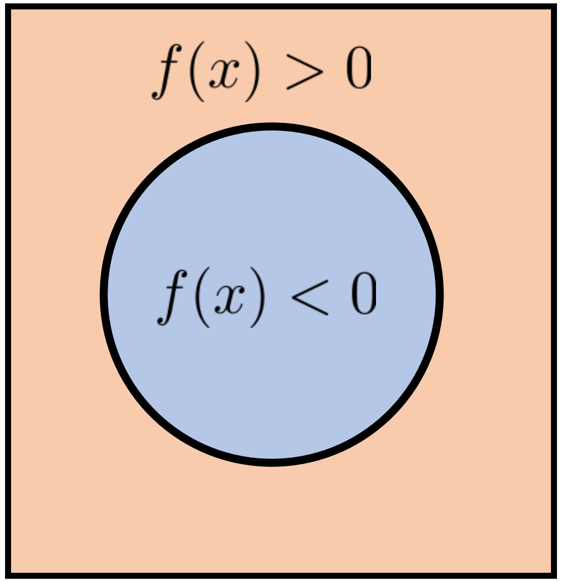
Implicit Curves and Surfaces

- Kernel of a scalar function $f : \mathbb{R}^m \rightarrow \mathbb{R}$
  - Curve in 2D: $S = \{x \in \mathbb{R}^2 | f(x) = 0\}$
  - Surface in 3D: $S = \{x \in \mathbb{R}^3 | f(x) = 0\}$
- Space partitioning

$\{x \in \mathbb{R}^m | f(x) > 0\}$ Outside

$\{x \in \mathbb{R}^m | f(x) = 0\}$ Curve/Surface

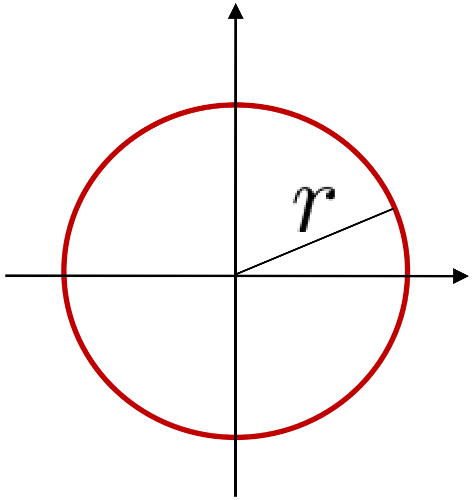
$\{x \in \mathbb{R}^m | f(x) < 0\}$ Inside



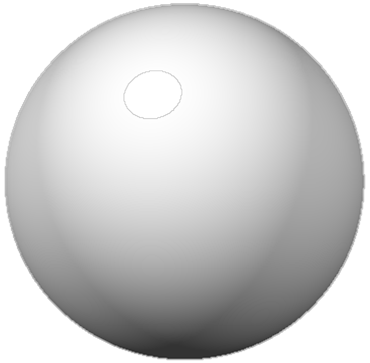
Implicit Curves and Surfaces

- Implicit circle and sphere

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



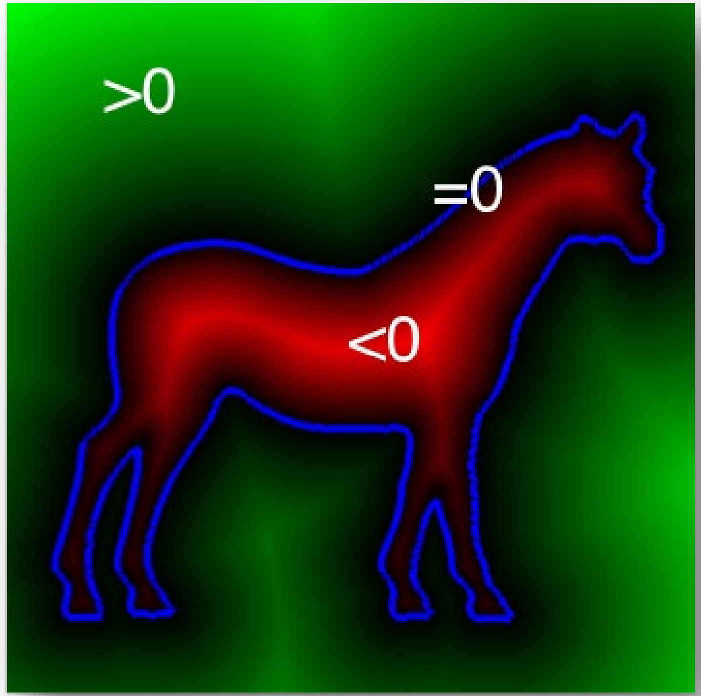
$$f(x, y, z) = x^2 + y^2 + z^2 - r^2$$



Implicit Curves and Surfaces

- Kernel of a scalar function $f : \mathbb{R}^m \rightarrow \mathbb{R}$
  - Curve in 2D: $S = \{x \in \mathbb{R}^2 | f(x) = 0\}$
  - Surface in 3D: $S = \{x \in \mathbb{R}^3 | f(x) = 0\}$

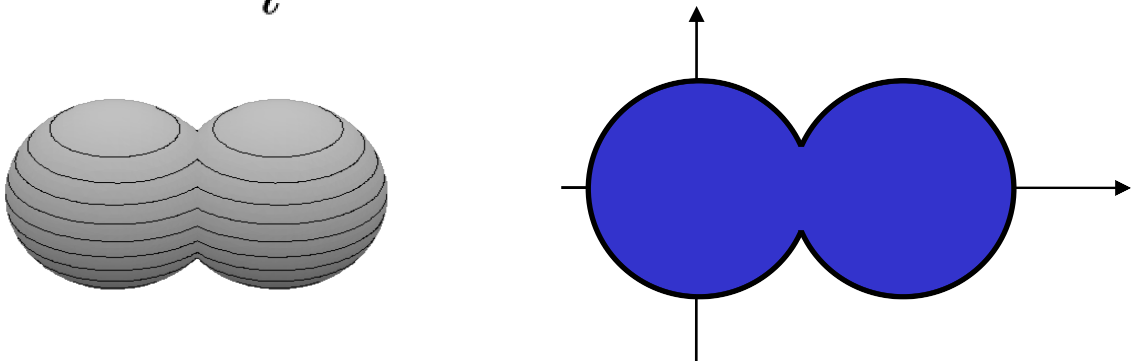
- Zero level set of
  signed distance function



Boolean Set Operations

- Union:

$$\bigcup_i f_i(x) = \min f_i(x)$$



- Intersection:

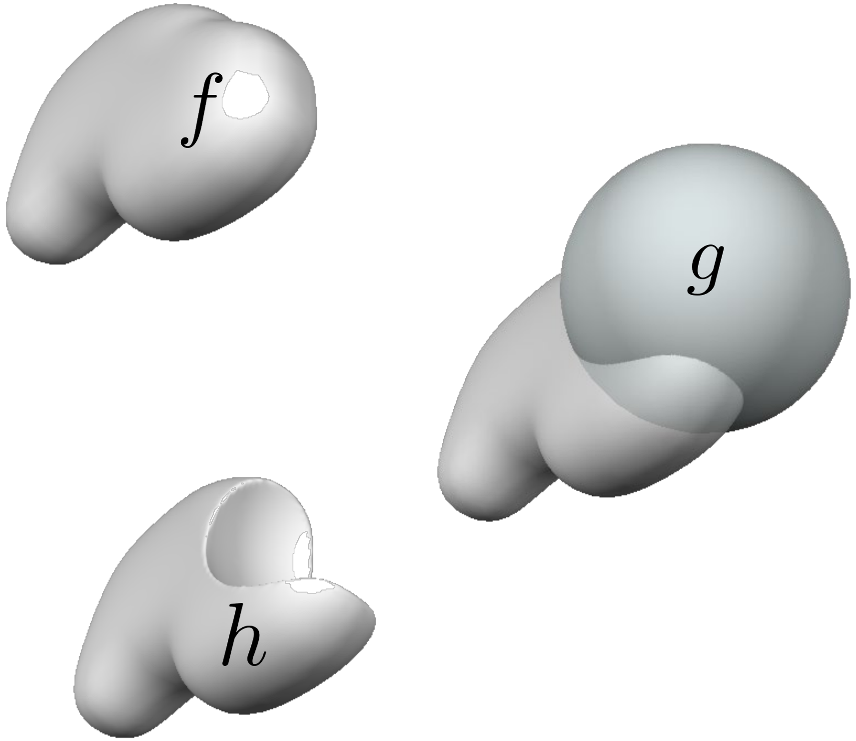
$$\bigcap_i f_i(x) = \max f_i(x)$$

More Boolean Set Operations

- Positive = outside, negative = inside
- Boolean subtraction:

	$f > 0$	$f < 0$
$g > 0$	$h > 0$	$h < 0$
$g < 0$	$h > 0$	$h > 0$

- Much easier than for parametric surfaces!



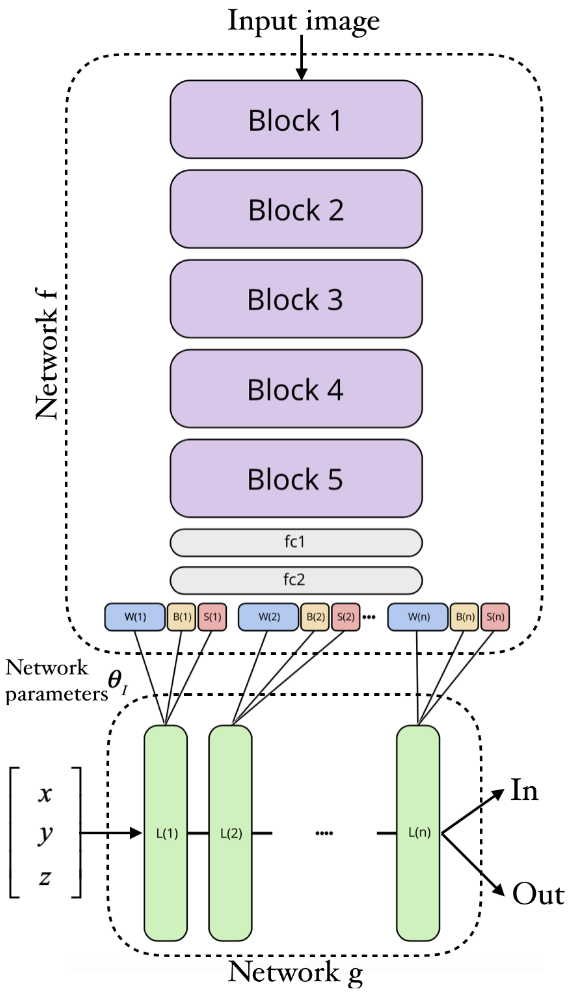
$$h = max(f, -g)$$

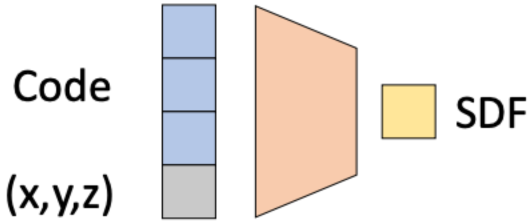
Implicit Curves and Surfaces

- Advantages
  - Easy to determine inside/outside
  - Easy to determine if a point is **on** the curve/surface, on what side of the surface

- Disadvantages
  - Hard to generate points on the curve/surface
  - Do not lend to (real-time) rendering

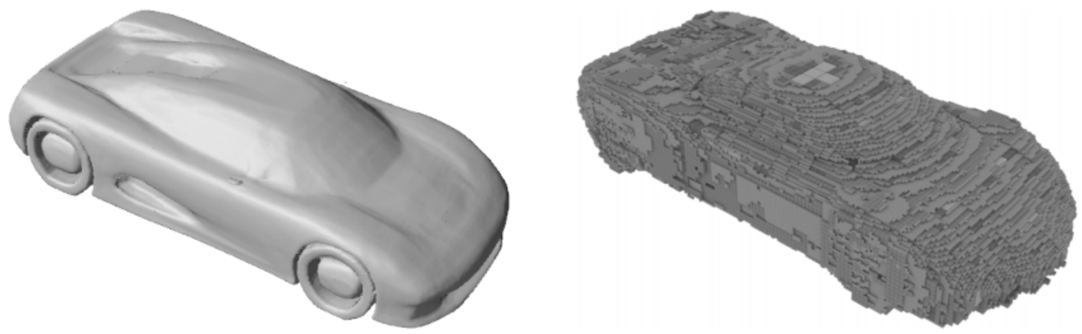
Preview: Neural AutoEncoding Occupancy or SDF





Decoder

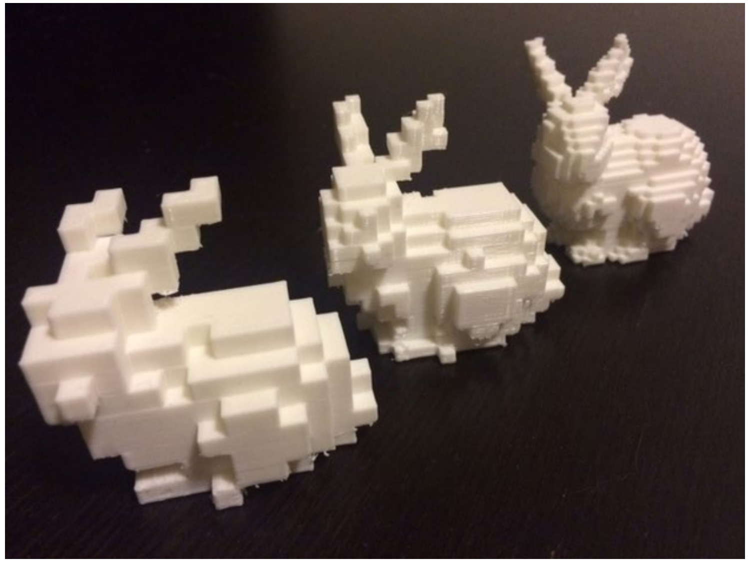
Comparison with Octree



(a) Ground-truth (b) Our Result (c) [22]-25 patch (d) [22]-sphere

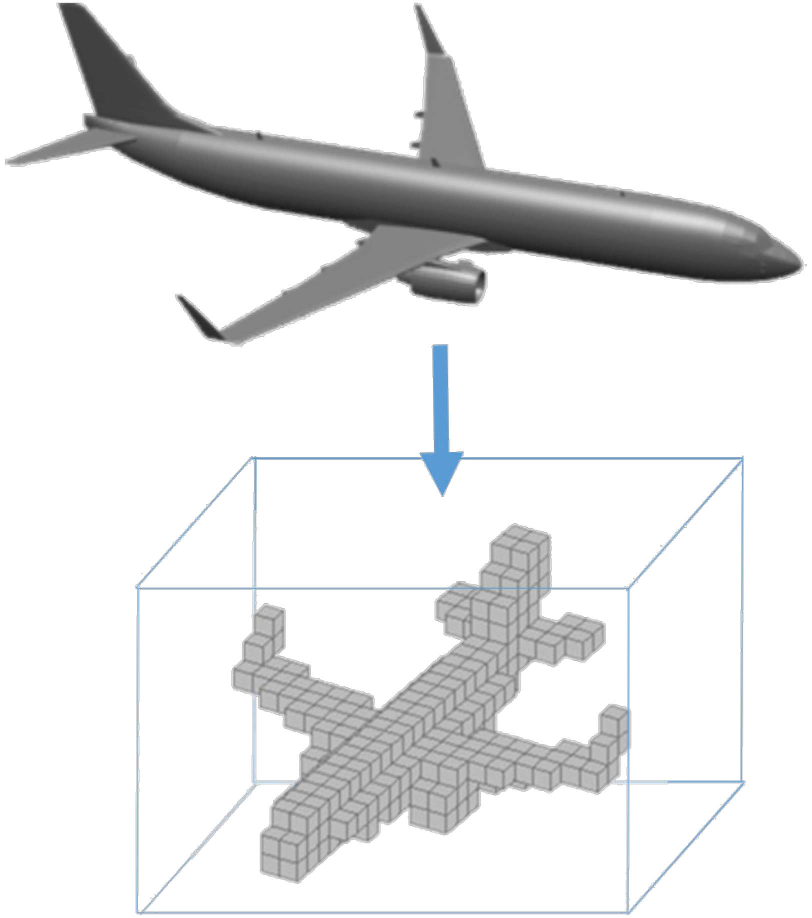
DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation, CVPR 2019
Deep Meta Functionals for Shape Representation, ICCV 2019

Volumetric Representations



V-Rep: Volumetric Grids

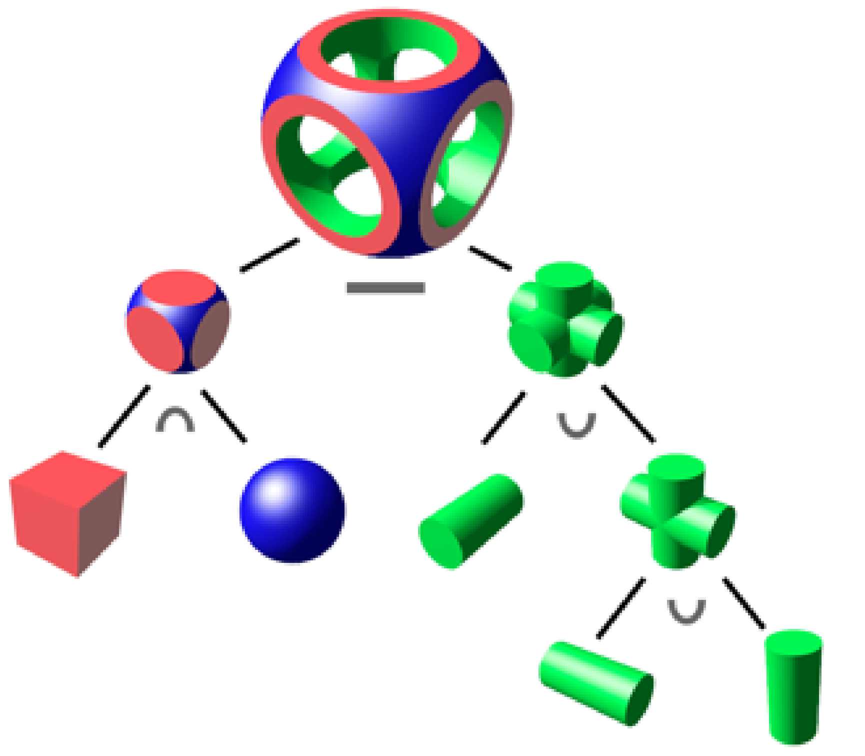
- Binary volumetric grids
- Can be produced by thresholding the distance function, or from the scanned points directly
- N^3 gets expensive fast



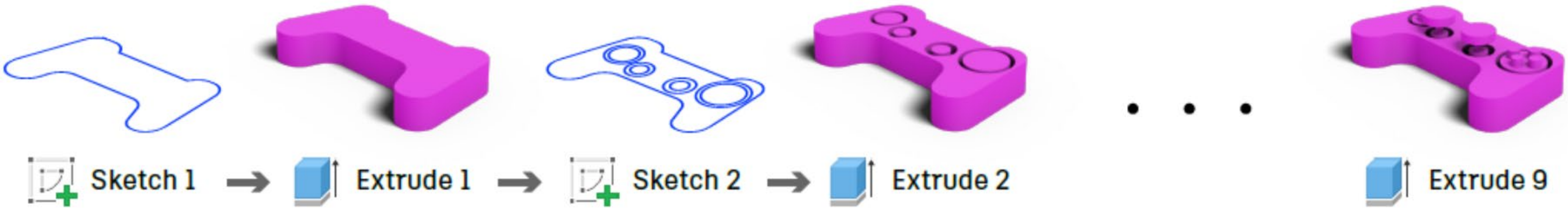
Also represents space of little informational value

V-Rep: CSG

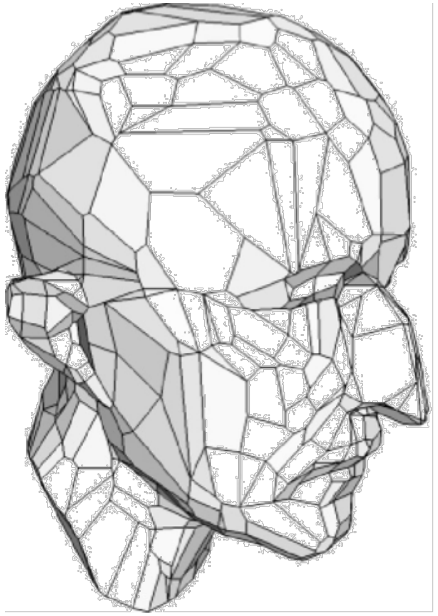
- Constructive Solid Geometry
- Boolean ops over geometric primitives (spheres, boxes, cylinders, cones, …)
- Often non-unique



Sketch-Extrude

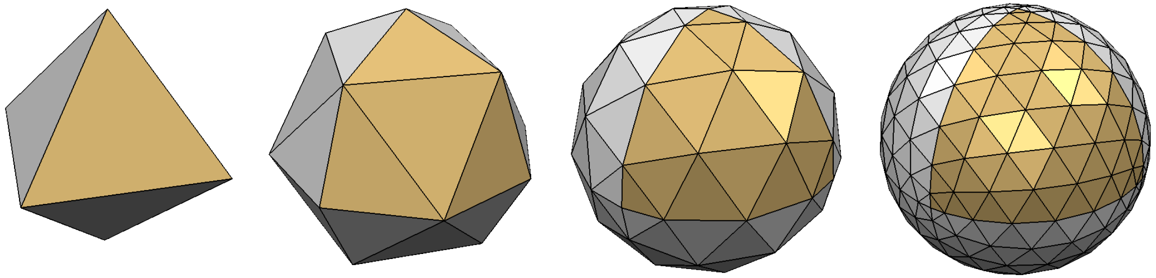


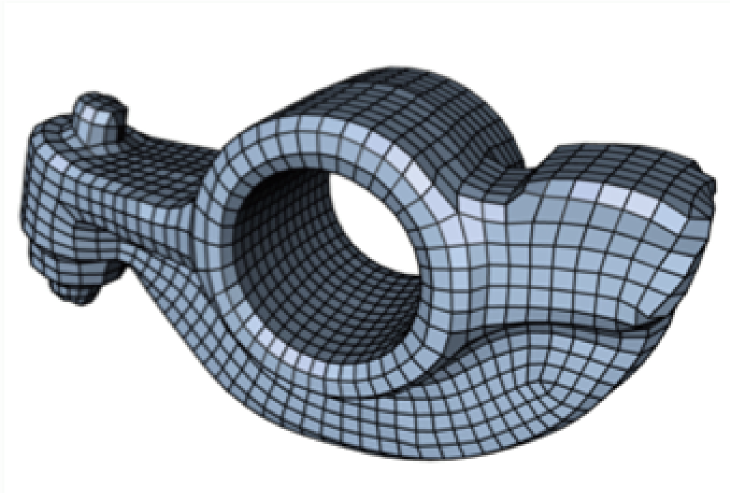
Polygonal Meshes



Polygonal Meshes

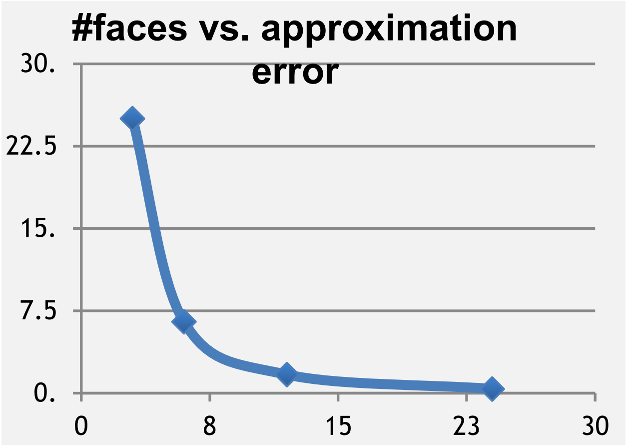
- Boundary representations of objects using polygonal primitives

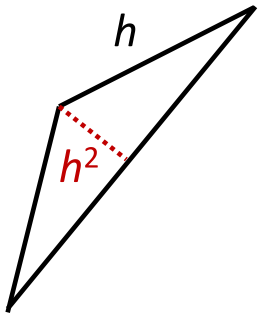


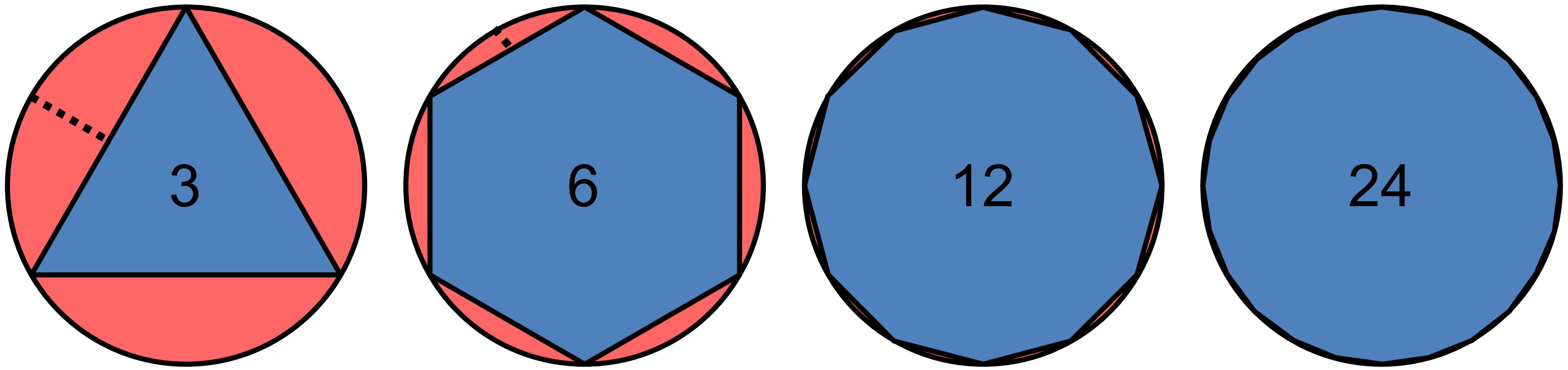


Meshes as Approximations of Smooth Surfaces

- Piecewise linear approximation
  - Error is $O(h^2)$ [$O(h)$ for points]



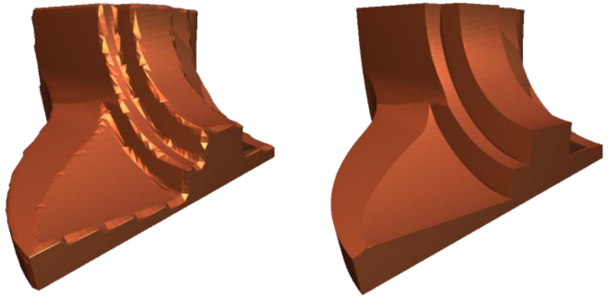


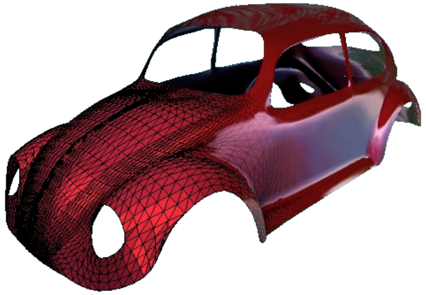


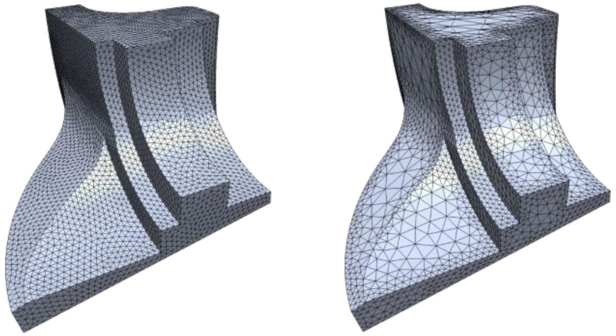
25% 6.5% 1.7% 0.4%

Polygonal Meshes

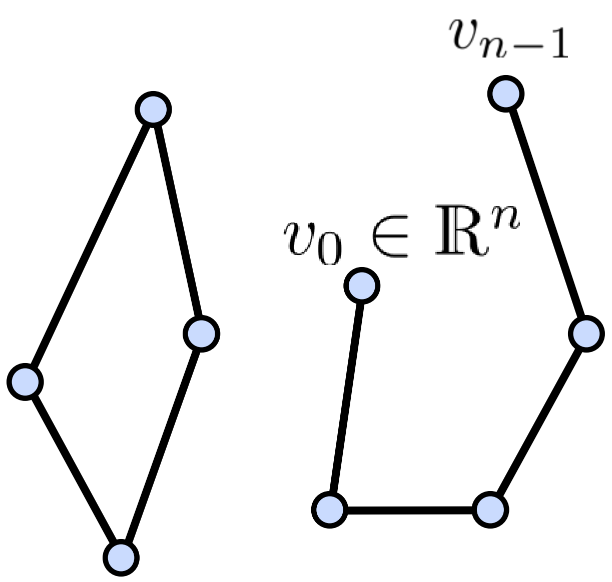
- Polygonal meshes are a good representation
  - approximation $O(h^2)$
  - arbitrary topology
  - adaptive refinement
  - efficient rendering







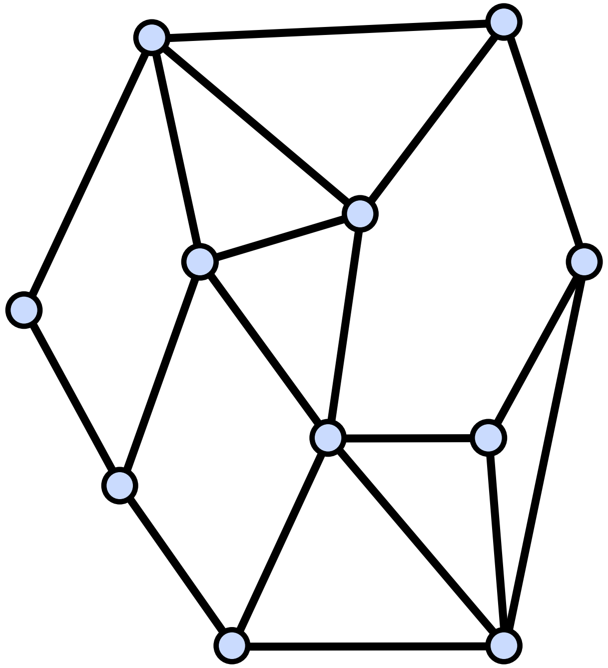
Planar Polygons



- Vertices: $v_0, v_1, \ldots, v_{n-1}$
- Edges: $\{(v_0, v_1), \ldots, (v_{n-2}, v_{n-1})\}$

- Closed: $v_0 = v_{n-1}$
- Planar: all vertices on a plane
- Simple: not self-intersecting

Polygonal Meshes (Complexes)

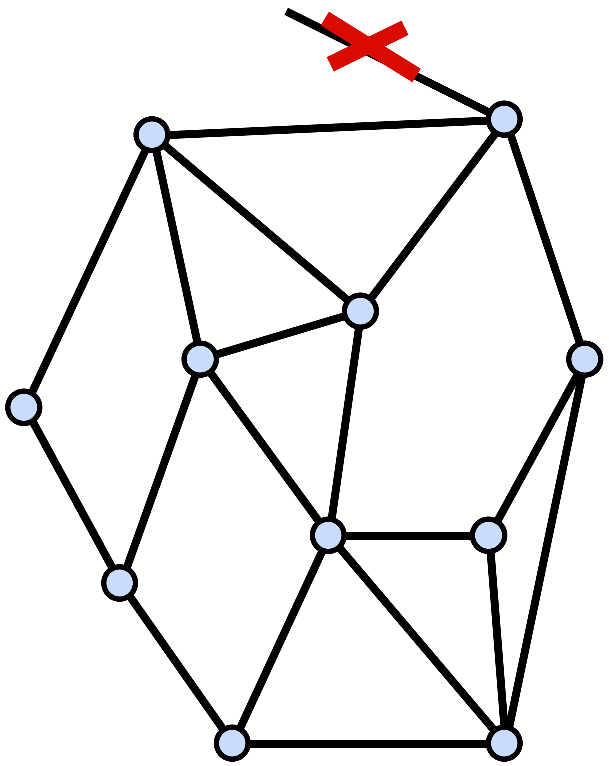


- A finite set M of closed, simple polygons Q_i is a polygonal mesh
- The intersection of two polygons in M is either empty, a vertex, or an edge

$$M = \langle V, E, F \rangle$$

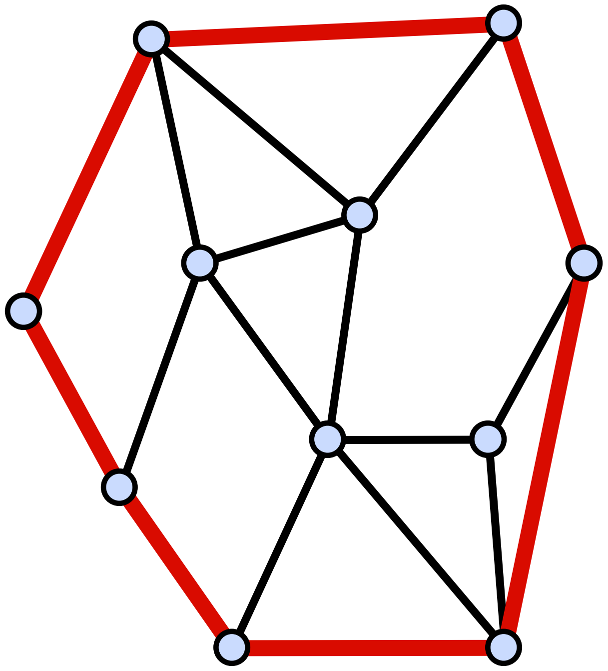
vertices (V), edges (E), faces (F)

Polygonal Mesh

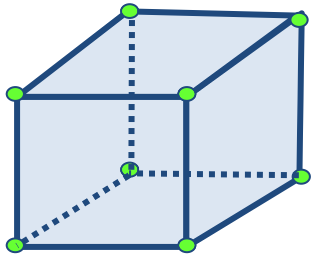


- A finite set M of closed, simple polygons Q_i is a polygonal mesh
- The intersection of two polygons in M is either empty, a vertex, or an edge
- Every edge belongs to at least one polygon

Polygonal Mesh

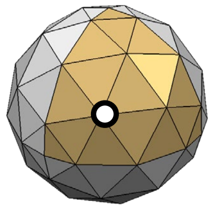


- **Boundary**: the set of all edges that belong to only one polygon
  - Either empty or forms closed loops
  - If empty, then the polygonal mesh is closed



Triangle Meshes

- Connectivity: vertices, edges, triangles
- Geometry: vertex positions

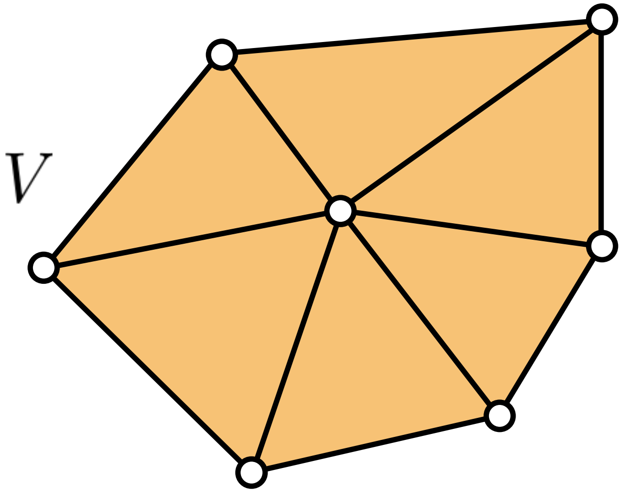


$$V = \{v_1, \ldots, v_n\}$$

$$E = \{e_1, \ldots, e_k\}, \quad e_i \in V \times V$$

$$F = \{f_1, \ldots, f_m\}, \quad f_i \in V \times V \times V$$

$$P = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}, \quad \mathbf{p}_i \in \mathbb{R}^3$$



Data Structures

- What should be stored?
  - Geometry: 3D coordinates
  - Connectivity
    - Adjacency relationships
  - Attributes
    - Normal, color, texture coordinates
    - Per vertex, face, edge

Continuous information

Discrete information

Simple Data Structures: Triangle List

- STL ("Standard Triangle Language”) format (used in CAD)
- Storage
  - Face: 3 positions
  - 4 bytes per coordinate
  - 36 bytes per face
    - on average: $f = 2v$ *(Euler)*
    - $72*v$ bytes for a mesh with v vertices
- No explicit connectivity information

Triangles			
0	x0	y0	z0
1	x1	y1	z1
2	x2	y2	z2
3	x3	y3	z3
4	x4	y4	z4
5	x5	y5	z5
6	x6	y6	z6
…	…	…	…

```
facet normal n_i n_j n_k
    outer loop
        vertex v1_x v1_y v1_z
        vertex v2_x v2_y v2_z
        vertex v3_x v3_y v3_z
    endloop
endfacet
```

Simple Data Structures: Triangle List

- Used in formats

  OBJ, OFF, WRL

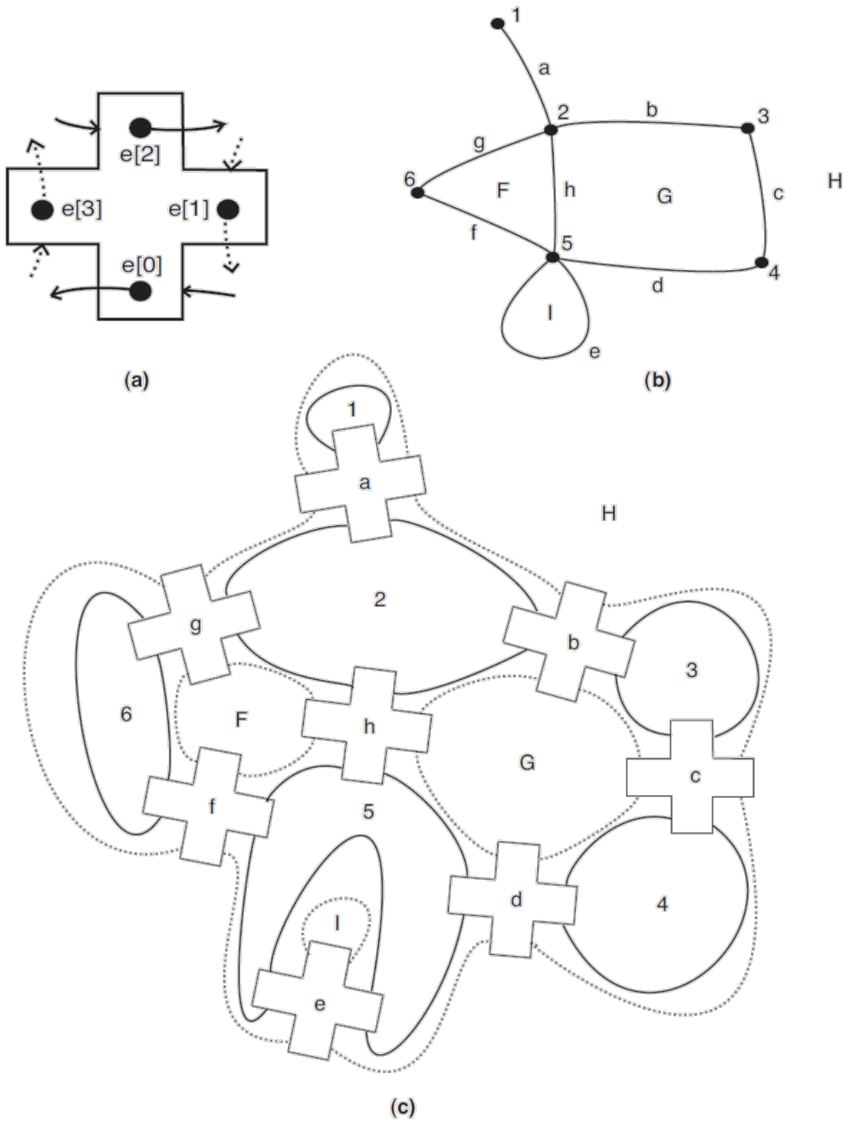
- Storage
  - Vertex: position
  - Face: vertex indices
  - 12 bytes per vertex
  - 12 bytes per face
  - 36**v* bytes for the mesh

- No explicit neighborhood info

Vertices			
v0	x0	y0	z0
v1	x1	x1	z1
v2	x2	y2	z2
v3	x3	y3	z3
v4	x4	y4	z4
v5	x5	y5	z5
v6	x6	y6	z6
…	⋮	⋮	⋮

Triangles			
t0	v0	v1	v2
t1	v0	v1	v3
t2	v2	v4	v3
t3	v5	v2	v6
…	⋮	⋮	⋮

Quad-Edge: Encoding Mesh Topology

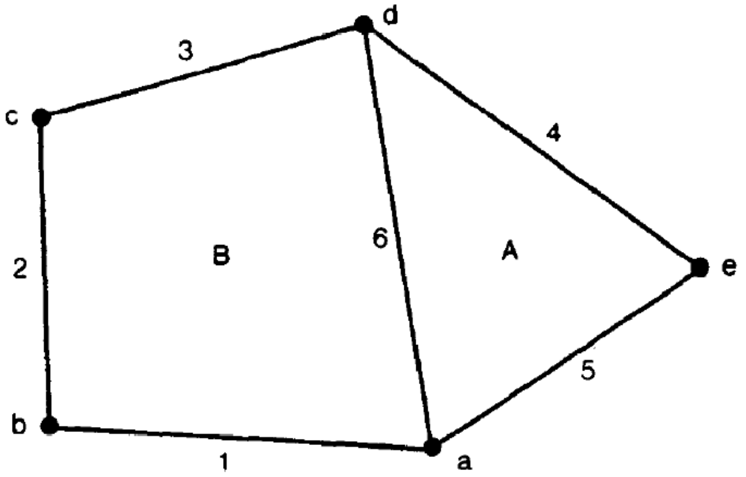


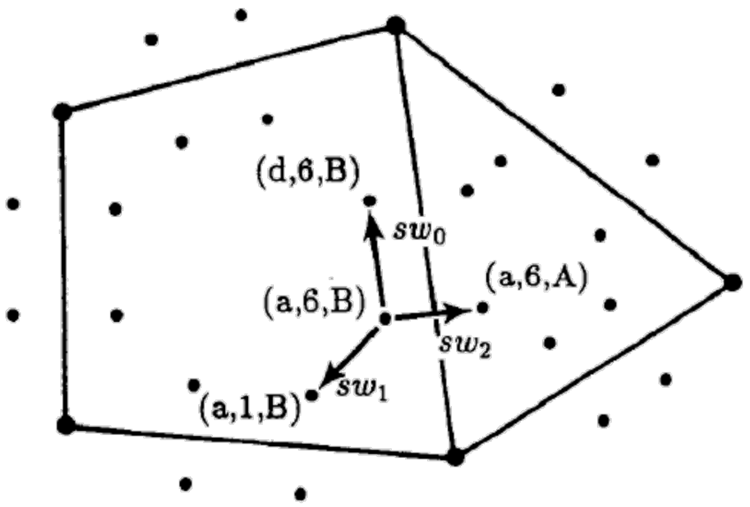
Edge-based
(many variants,
half-edge, etc)

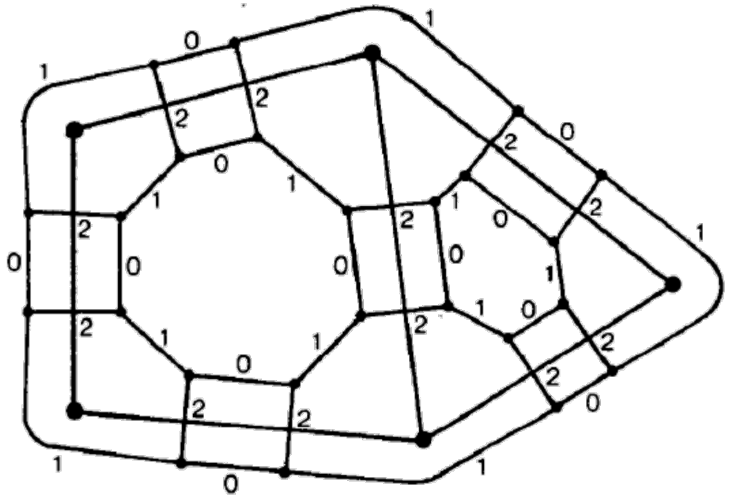
Topological traversal algorithms

Brisson: Cell-Tuple

(v,e,f)

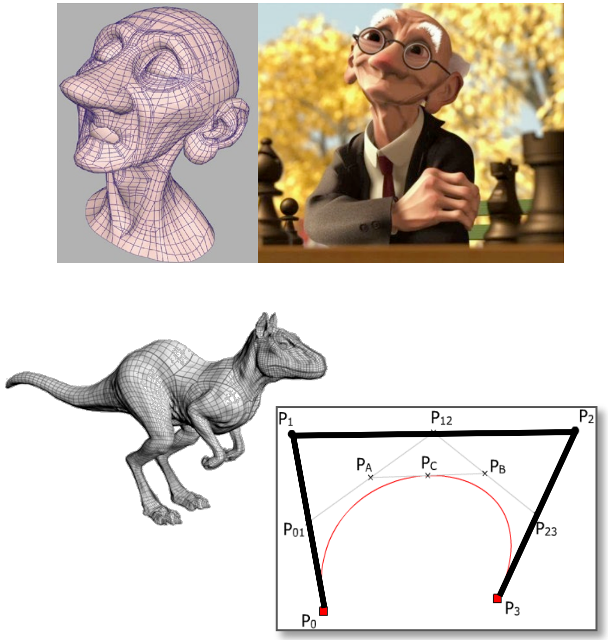
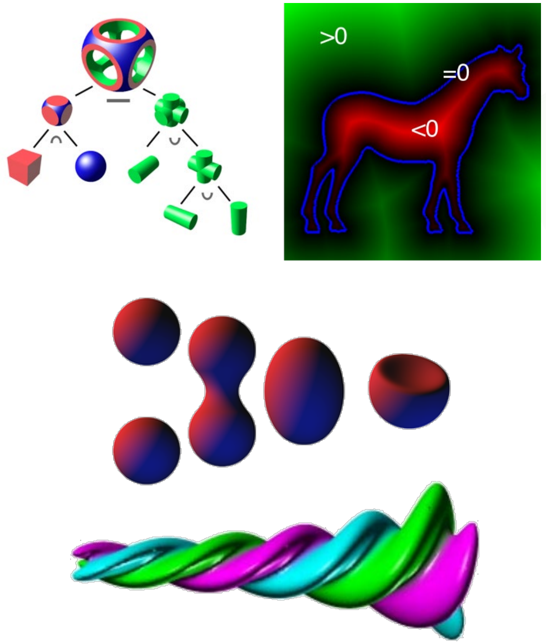
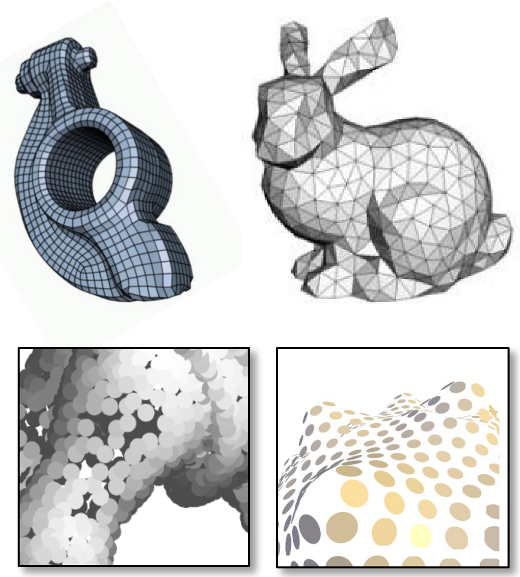






G_M

Summary

<table>
<tr><th>Parametric</th><th>Implicit</th><th>Discrete/Sampled</th></tr>
<tr>
<td>
</td>
<td>
</td>
<td></td>
</tr>
<tr>
<td>• Splines, tensor-product surfaces
• Subdivision surfaces</td>
<td>• Distance fields
• Metaballs/blobs</td>
<td>• Meshes
• Point set surfaces</td>
</tr>
</table>

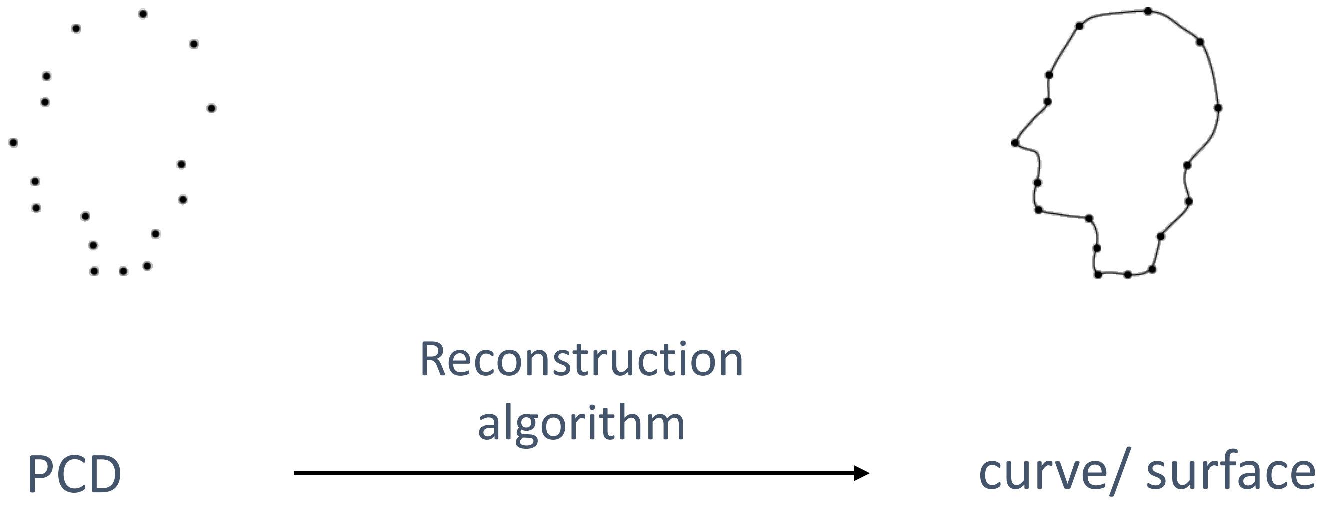
Representation Conversions

Points → Implicit
Implicit → Mesh
Mesh → Points

3D Point Cloud Reconstruction

Main Goal:

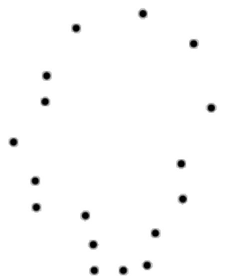
Construct a polygonal (e.g. triangle mesh) representation of the point cloud.



3D Point Cloud Reconstruction

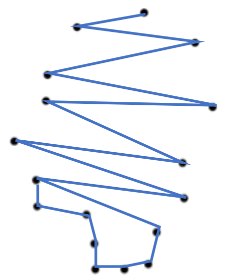
Main Problem:

Data is **unstructured.** E.g. in 2D the points are not **ordered.**



Reconstruction algorithm

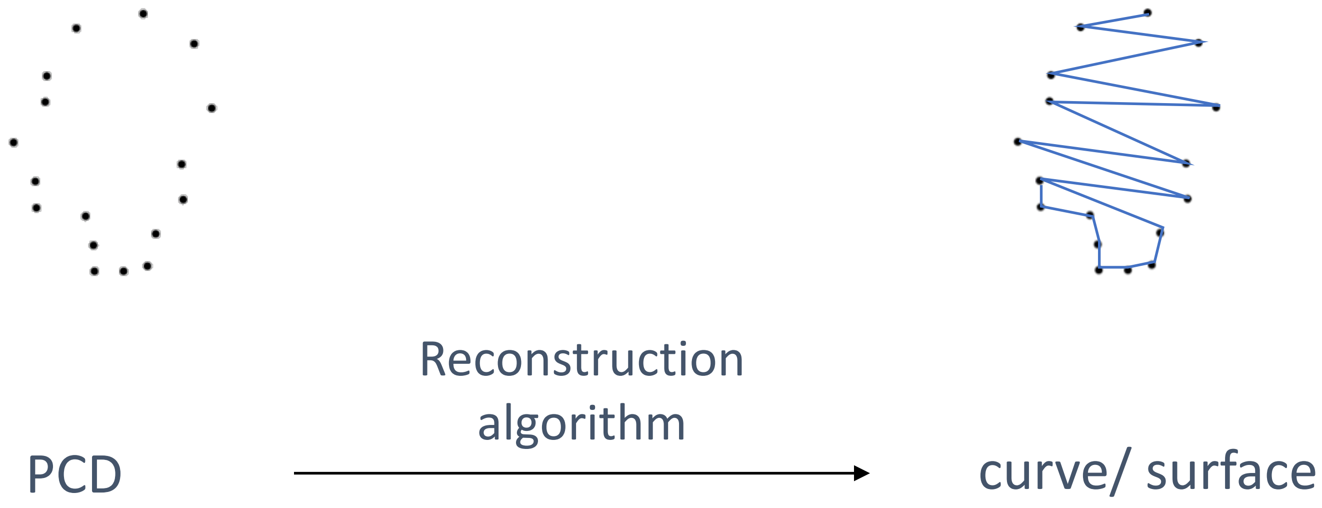
PCD → curve/ surface



3D Point Cloud Reconstruction

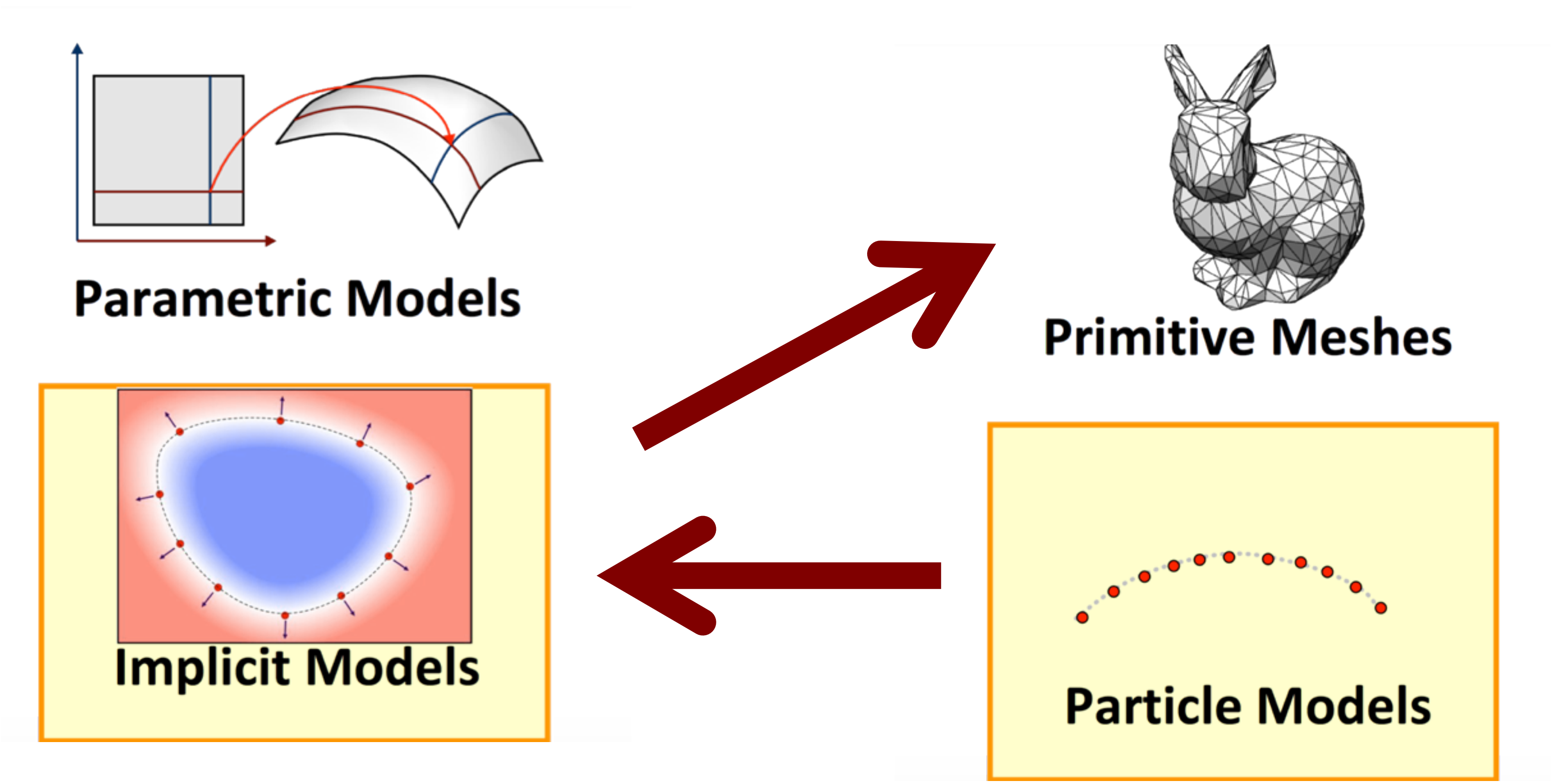
Main Problem:

Data is **unstructured.** E.g. in 2D the points are not **ordered.**
Inherently **ill-posed** (aka difficult) problem.



3D Point Cloud Reconstruction

Reconstruction through Implicit models.



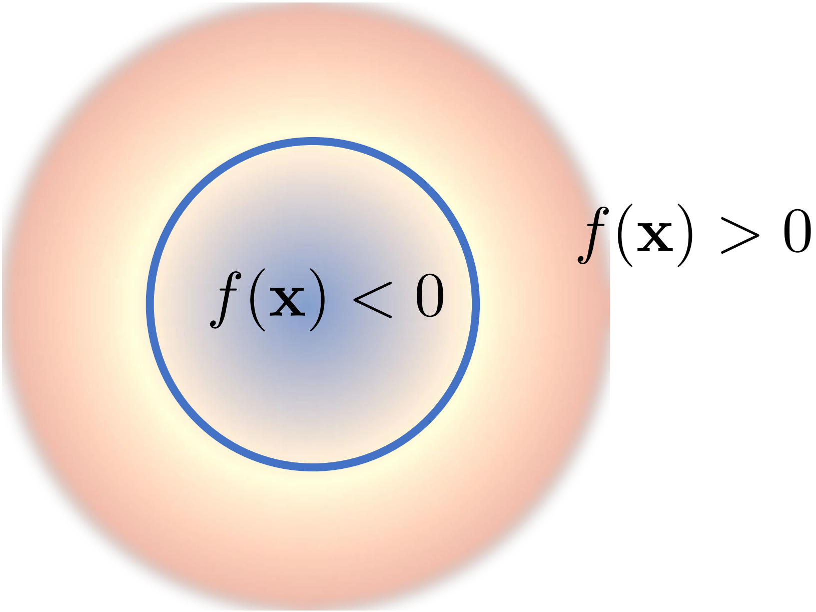
POINTS → IMPLICIT

Implicit Surface Reconstruction

Implicit Surfaces

Given a function f(**x**), the surface is defined as:

$$\{\mathbf{x}, \text{s.t.} f(\mathbf{x}) = 0\}$$



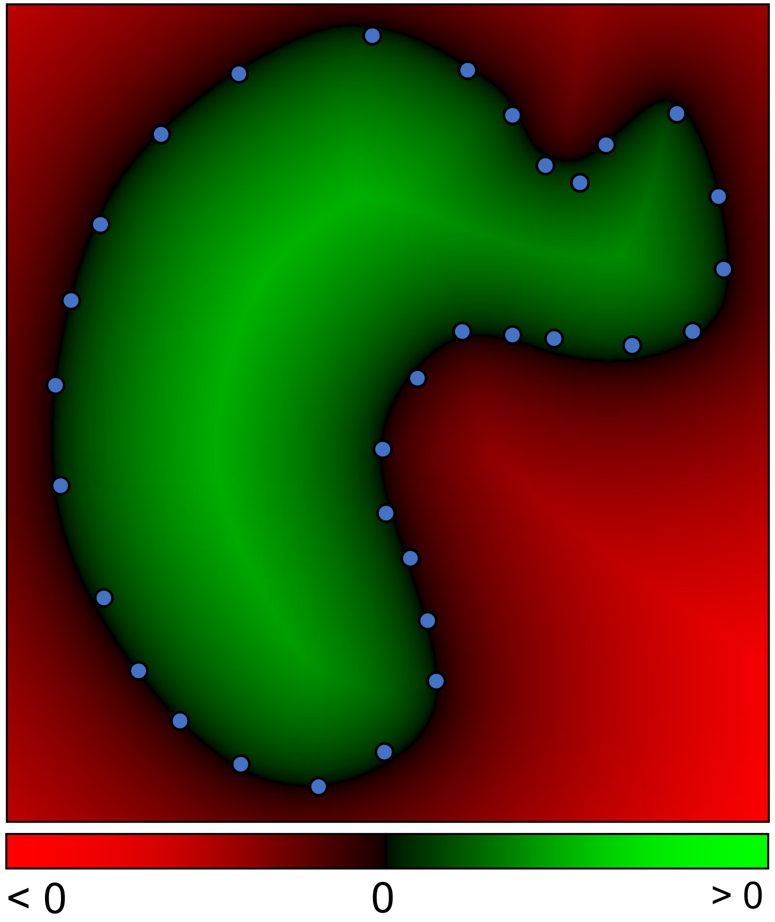
$$f(x, y) = x^2 + y^2 - r^2$$

Implicit Function Approach

- Given a point cloud
- Define a function $f : R^3 \rightarrow R$

with value > 0 outside the shape and < 0 inside

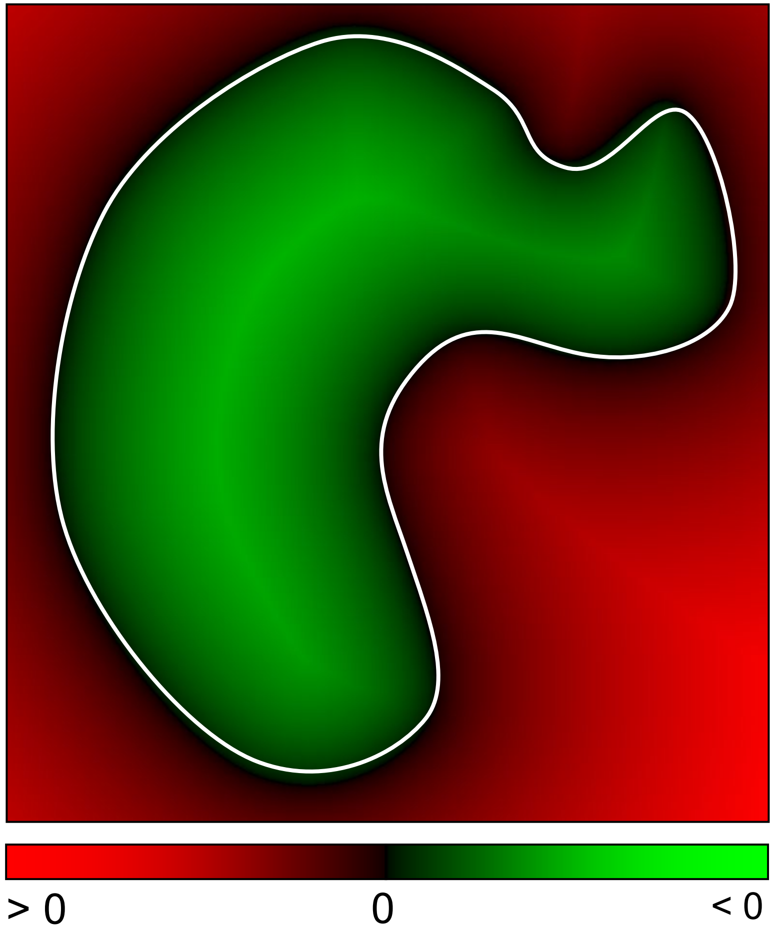
Example: signed distance function (SDF) to the shape surface



Implicit Function Approach

- Define a function $f : R^3 \rightarrow R$

  with value > 0 outside the shape and < 0 inside

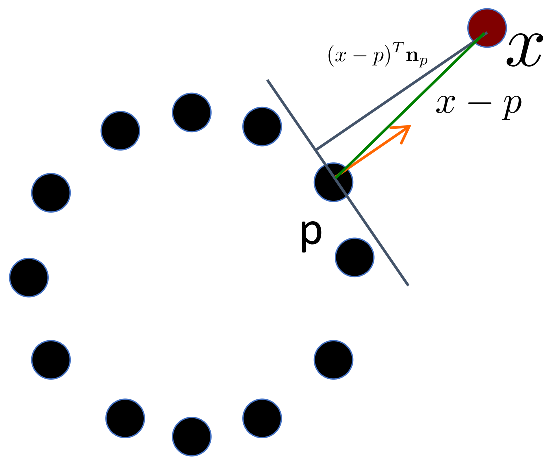


- Extract the zero-set

$$\{x : f(x) = 0\}$$

Implicit Surfaces

Converting from a point cloud to an implicit surface:



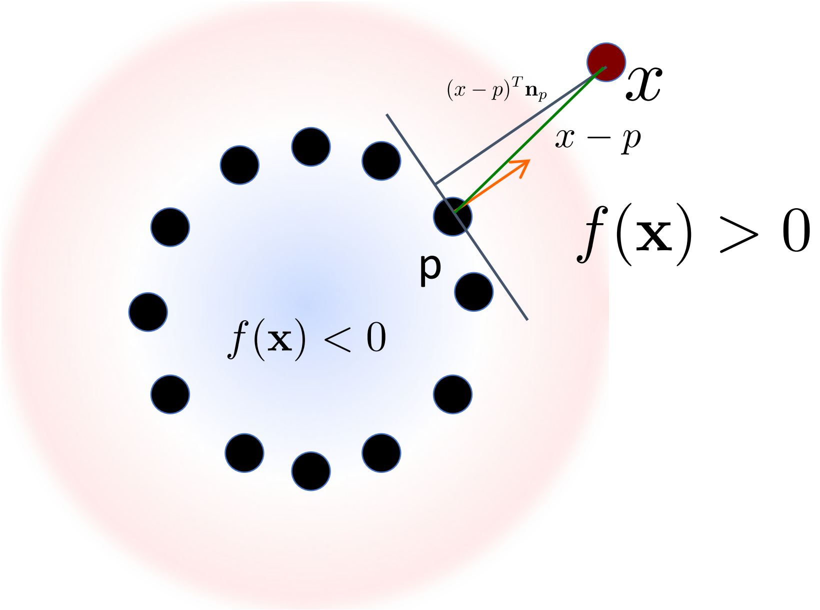
Simplest method:

1. Given a point x in space, find nearest point p in PCD.
2. Set $f(x) = (x-p)^T\mathbf{n}_p$ – signed distance to the tangent plane.

Hugues Hoppe: Surface reconstruction from unorganized points

Implicit Surfaces

Converting from a point cloud to an implicit surface:



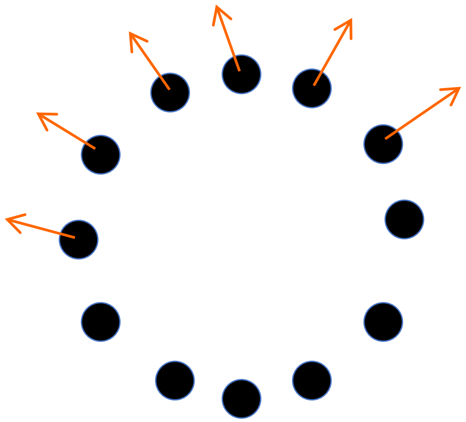
Simplest method:

1. Given a point x in space, find nearest point p in PCD.
2. Set $f(x) = (x-p)^T \mathbf{n}_p$ – signed distance to the tangent plane.

Hugues Hoppe: Surface reconstruction from unorganized points

Implicit Surfaces

Converting from a point cloud to an implicit surface:



Simplest method:

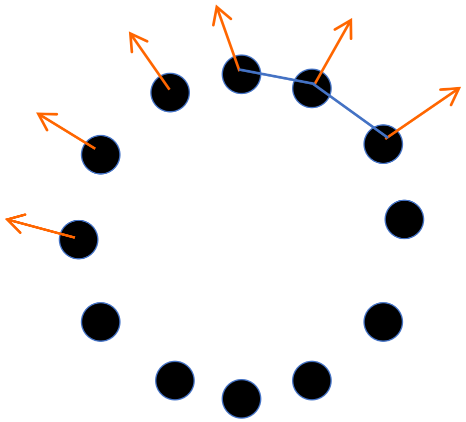
1. Given a point x in space, find nearest point p in PCD.
2. Set $f(x) = (x - p)^T \mathbf{n}_p$ – signed distance to the tangent plane.
3. Note: need consistently oriented normals.

    PCA only gives normals **up to orientation**

Hugues Hoppe: Surface reconstruction from unorganized points

Implicit Surfaces

Converting from a point cloud to an implicit surface:



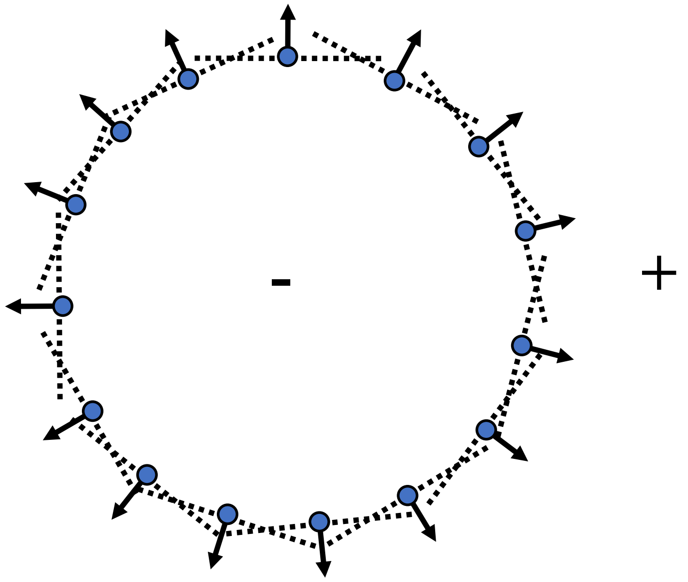
Simplest method:

1. Given a point x in space, find nearest point p in PCD.
2. Set $f(x) = (x - p)^T \mathbf{n}_p$ – signed distance to the tangent plane.
3. Note: need consistently oriented normals. In general, difficult problem, but can try to locally connect points and fix orientations.

Hugues Hoppe: Surface reconstruction from unorganized points

SDF from Points and Normals

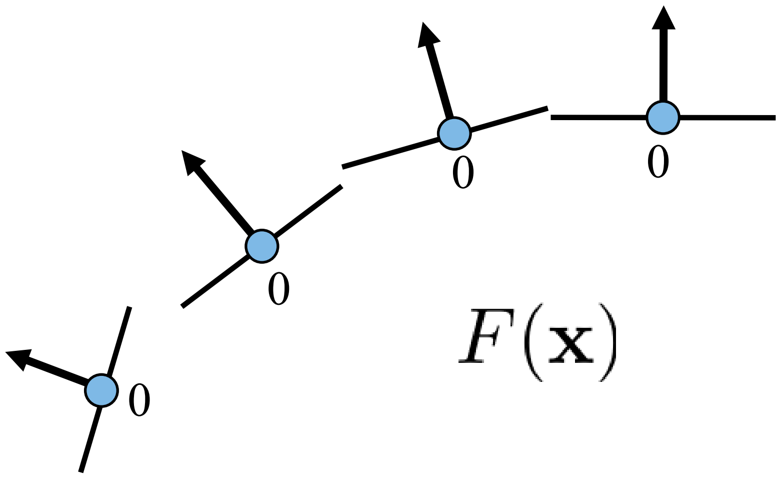
- Input: Points + Normals
- Normals help to distinguish between inside and outside
- Computed via locally fitting planes at the points (and consistently oriented)
- Previous method is very local and gives noisy results



“Surface reconstruction from unorganized points”, Hoppe et al., ACM SIGGRAPH 1992
http://research.microsoft.com/en-us/um/people/hoppe/proj/recon/

A More Global, Smooth SDF

- Find smooth implicit F
- Scattered data interpolation:
  - $F(\mathbf{p}_i) = 0$
  - F is smooth
  - Avoid trivial $F \equiv 0$

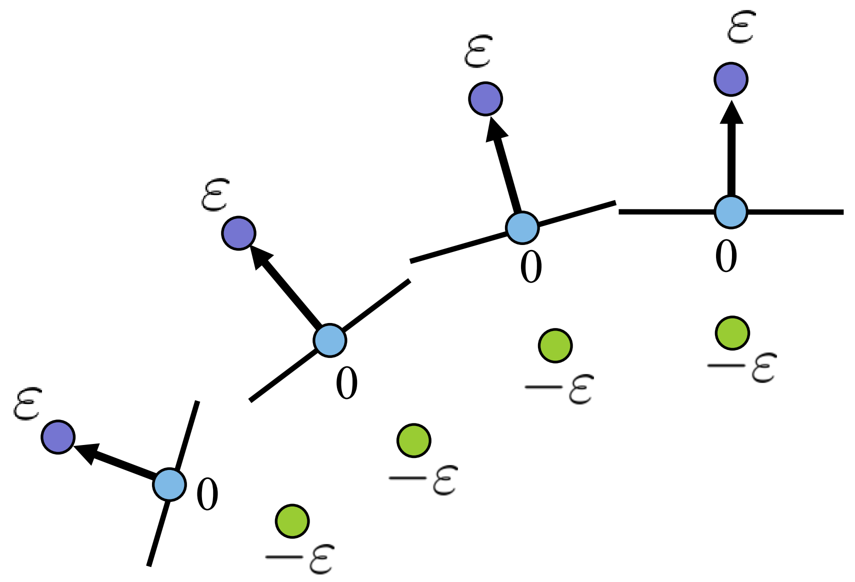


“Reconstruction and representation of 3D objects with radial basis functions”, Carr et al., ACM SIGGRAPH 2001

Smooth SDF

- Scattered data interpolation:
  - $F(\mathbf{p}_i) = 0$
  - F is smooth
  - Avoid trivial $F \equiv 0$

- Add off-surface constraints



$$F(\mathbf{p}_i + \varepsilon \mathbf{n}_i) = \varepsilon$$
$$F(\mathbf{p}_i - \varepsilon \mathbf{n}_i) = -\varepsilon$$

"Reconstruction and representation of 3D objects with radial basis functions", Carr et al., ACM SIGGRAPH 2001

Radial Basis Function Interpolation

- **RBF**: Weighted sum of shifted, smooth kernels

$$F(\mathbf{x}) = \sum_{i=0}^{N-1} w_i \, \varphi(\|\mathbf{x} - \mathbf{c}_i\|) \quad N = 3n$$

Scalar weights
Unknowns

Smooth kernels (basis functions) centered at constrained points.
For example:

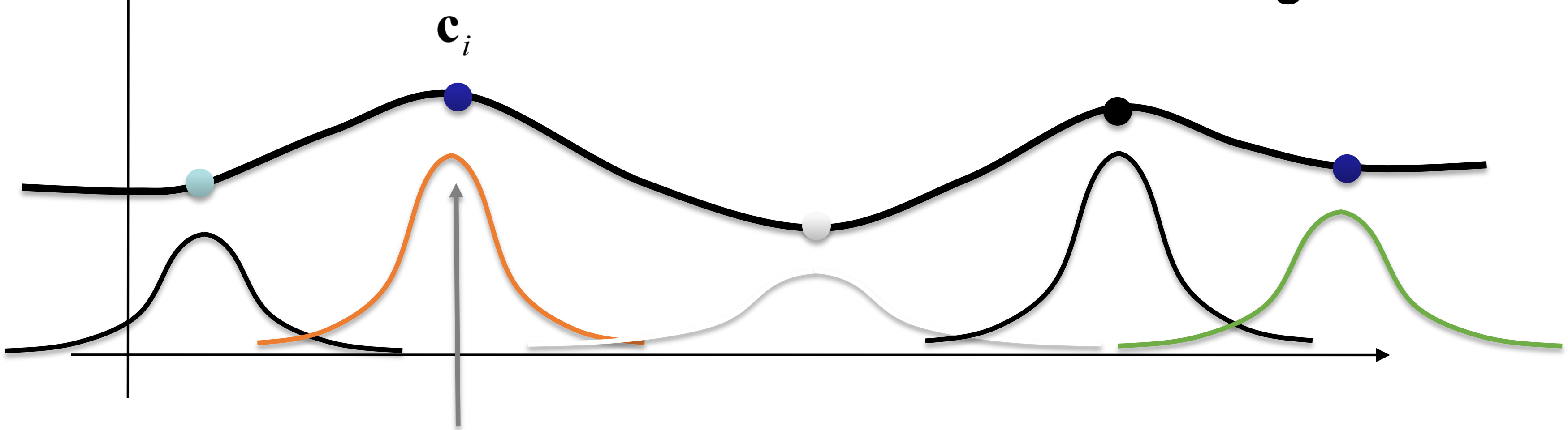
$$\varphi(r) = r^3$$

Radial Basis Function Interpolation

$$dist(\mathbf{x}) = \sum_i w_i \varphi_i(\mathbf{x}) = \sum_i w_i \varphi(\|\mathbf{x} - \mathbf{c}_i\|)$$

Kernel centers: on- and off-surface points

How do we find the weights?



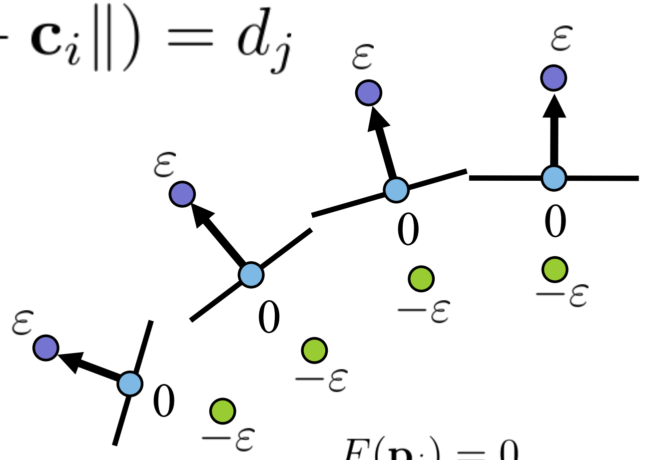
$$\varphi_i(\mathbf{x}) = \varphi(\|\mathbf{x} - \mathbf{c}_i\|)$$

Radial Basis Function Interpolation

- Interpolate the constraints:

$$\{\mathbf{c}_{3i}, \mathbf{c}_{3i+1}, \mathbf{c}_{3i+2}\} = \{\mathbf{p}_i,\ \mathbf{p}_i + \varepsilon\mathbf{n}_i,\ \mathbf{p}_i - \varepsilon\mathbf{n}_i\}$$

$$\forall j = 0, \ldots, N-1, \quad \sum_{i=0}^{N-1} w_i\, \varphi(\|\mathbf{c}_j - \mathbf{c}_i\|) = d_j$$



$$F(\mathbf{p}_i) = 0$$
$$F(\mathbf{p}_i + \varepsilon\mathbf{n}_i) = \varepsilon$$
$$F(\mathbf{p}_i - \varepsilon\mathbf{n}_i) = -\varepsilon$$

Radial Basis Function Interpolation

- Interpolate the constraints:

$$\{\mathbf{c}_{3i}, \mathbf{c}_{3i+1}, \mathbf{c}_{3i+2}\} = \{\mathbf{p}_i,\ \mathbf{p}_i + \varepsilon\mathbf{n}_i,\ \mathbf{p}_i - \varepsilon\mathbf{n}_i\}$$

- Symmetric linear system to get the weights:

$$\begin{pmatrix} \varphi(\|\mathbf{c}_0 - \mathbf{c}_0\|) & \dots & \varphi(\|\mathbf{c}_0 - \mathbf{c}_{N-1}\| \\ \vdots & \ddots & \vdots \\ \varphi(\|\mathbf{c}_{N-1} - \mathbf{c}_0\|) & \dots & \varphi(\|\mathbf{c}_{N-1} - \mathbf{c}_{N-1}\|) \end{pmatrix} \begin{pmatrix} w_0 \\ \vdots \\ w_{N-1} \end{pmatrix} = \begin{pmatrix} d_0 \\ \vdots \\ d_{N-1} \end{pmatrix}$$

3n equations

3n variables

RBF Kernels

- Triharmonic: $\varphi(r) = r^3$
  - Globally supported
  - Leads to dense symmetric linear system
  - C^2 smoothness
  - Works well for highly irregular sampling

RBF Kernels

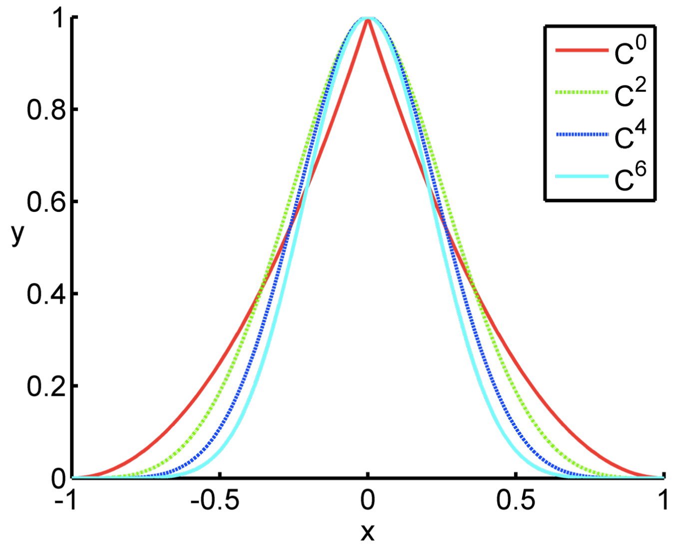
- Polyharmonic spline
  - $\varphi(r) = r^k \log(r), \;\; k = 2, 4, 6 \ldots$
  - $\varphi(r) = r^k, \;\; k = 1, 3, 5 \ldots$
- Multiquadratic

$$\varphi(r) = \sqrt{r^2 + \beta^2}$$

- Gaussian

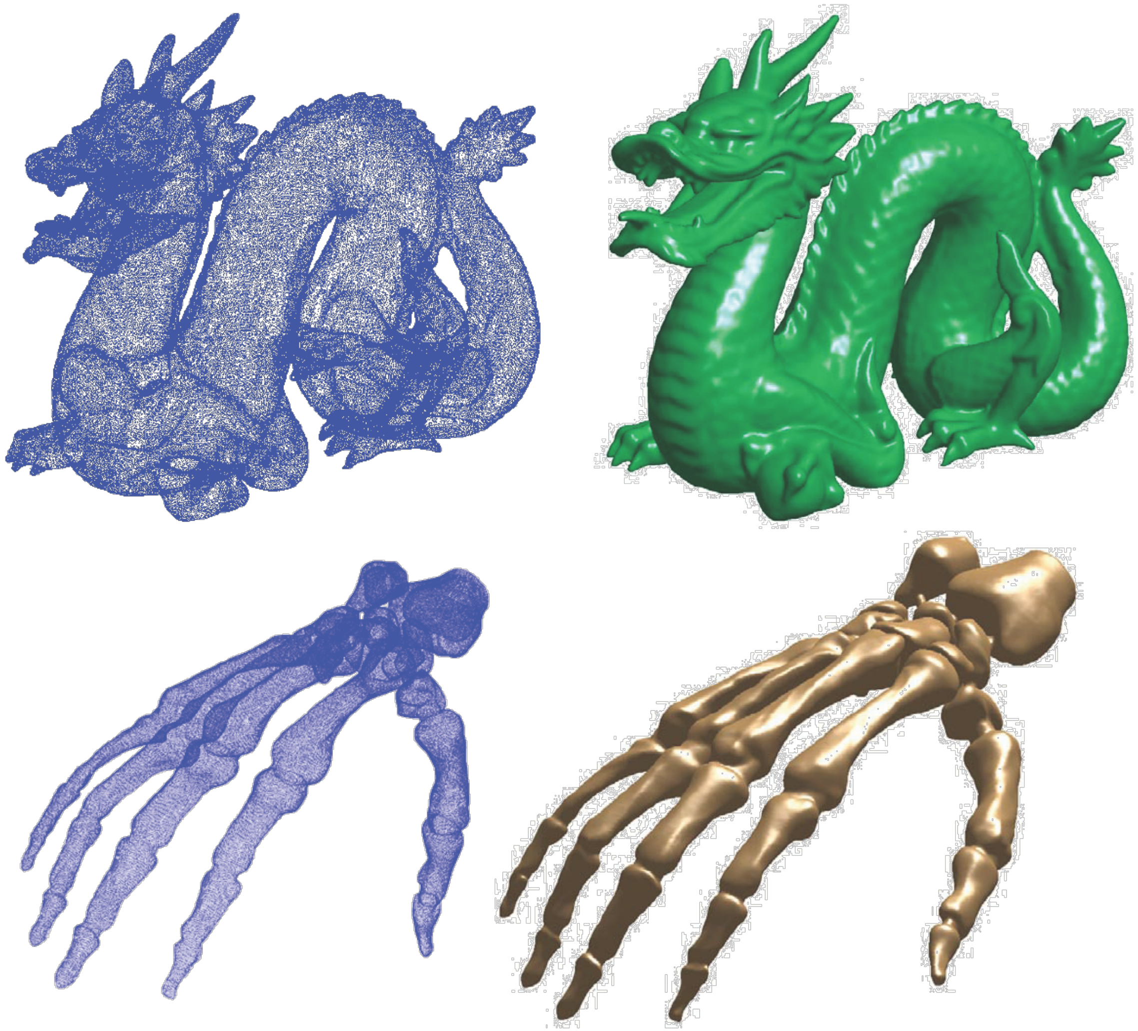
$$\varphi(r) = e^{-\beta r^2}$$

- B-Spline (compact support)



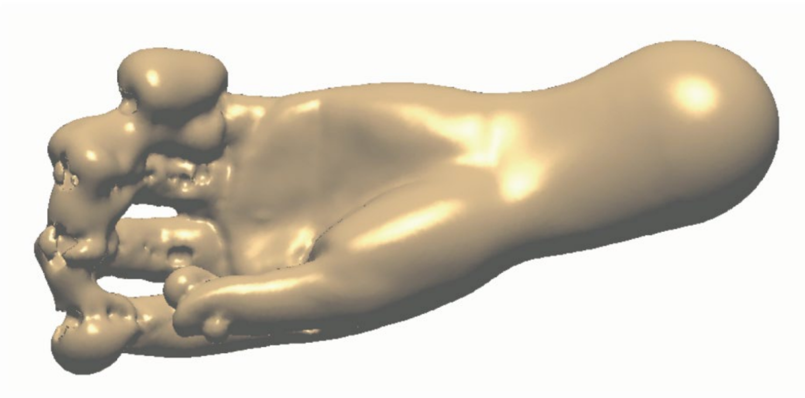
$$\varphi(r) = \text{piecewise-polynomial}(r)$$

RBF Reconstruction Examples

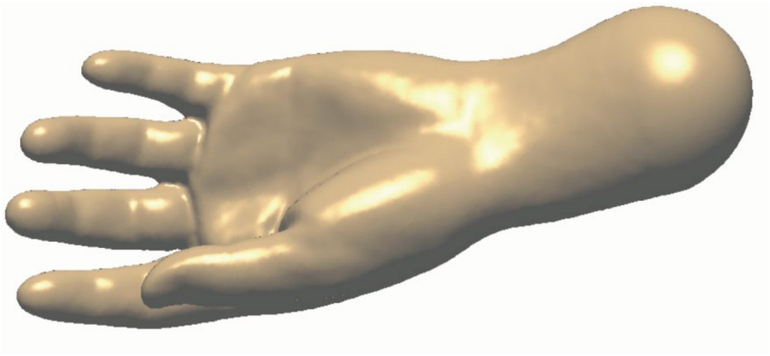


“Reconstruction and representation of 3D objects with radial basis functions”, Carr et al., ACM SIGGRAPH 2001

Off-Surface Points



Insufficient number/
badly placed off-surface points



Properly chosen off-surface points

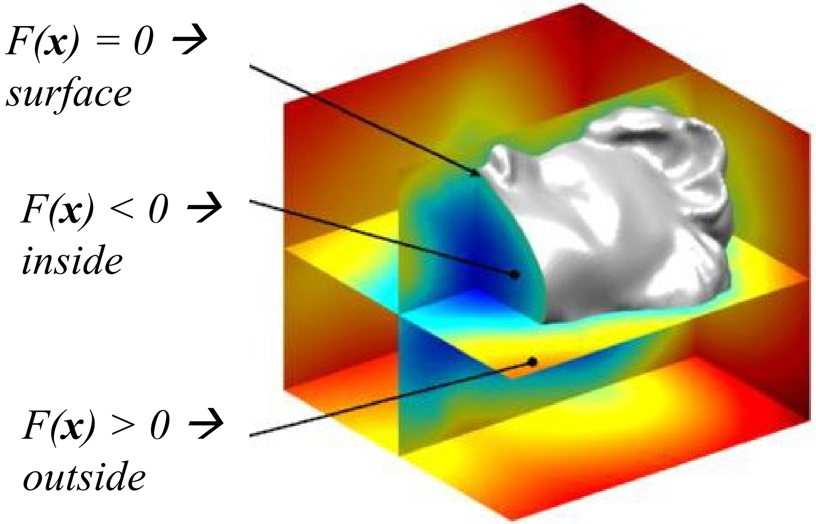
"Reconstruction and representation of 3D objects with radial basis functions", Carr et al., ACM SIGGRAPH 2001

IMPLICT → MESH

Marching Cubes

Extracting the Surface

- Wish to compute a manifold mesh of the level set

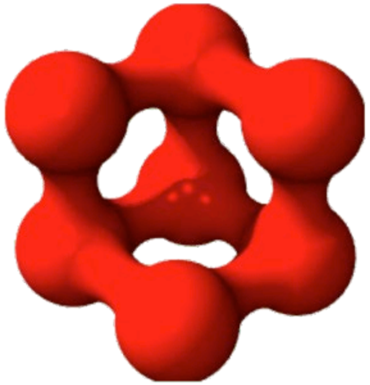


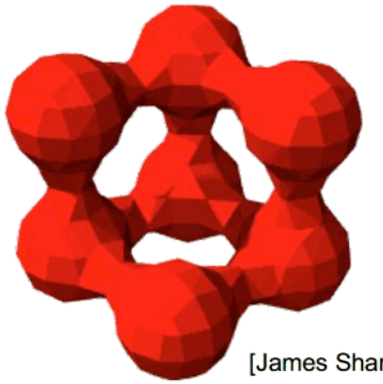
Marching Cubes

Converting from implicit to explicit representations.

Goal: Given an implicit representation: $\{\mathbf{x}, \text{s.t.} f(\mathbf{x}) = 0\}$

Create a triangle mesh that approximates the surface.

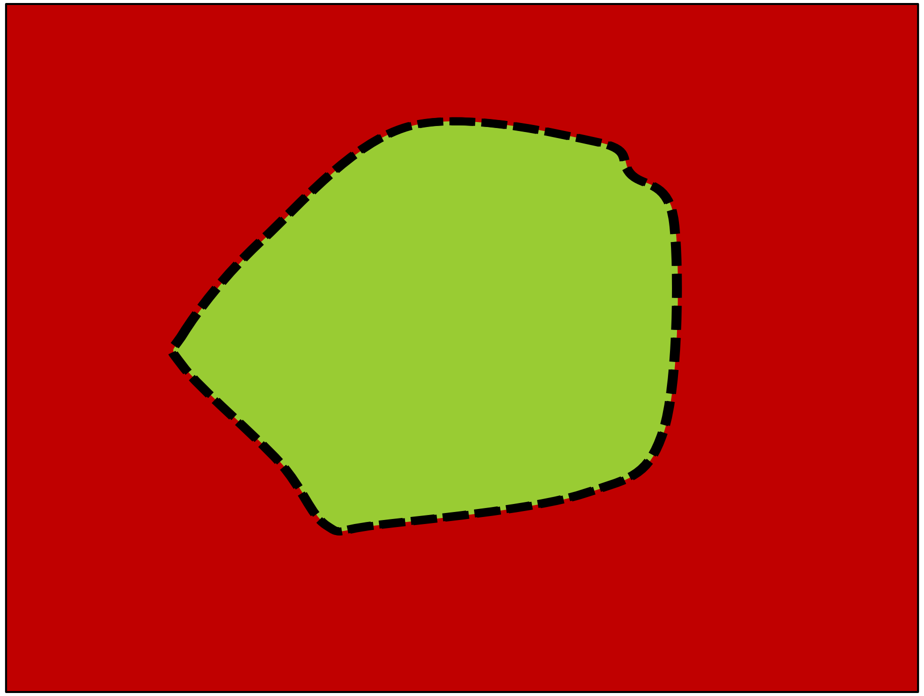




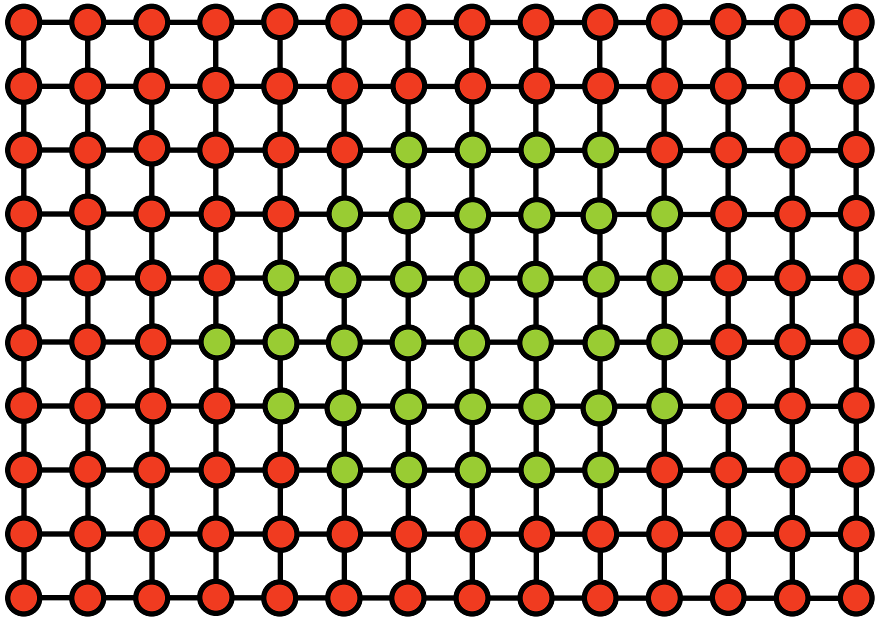
[James Sharman]

Lorensen and Cline, SIGGRAPH '87

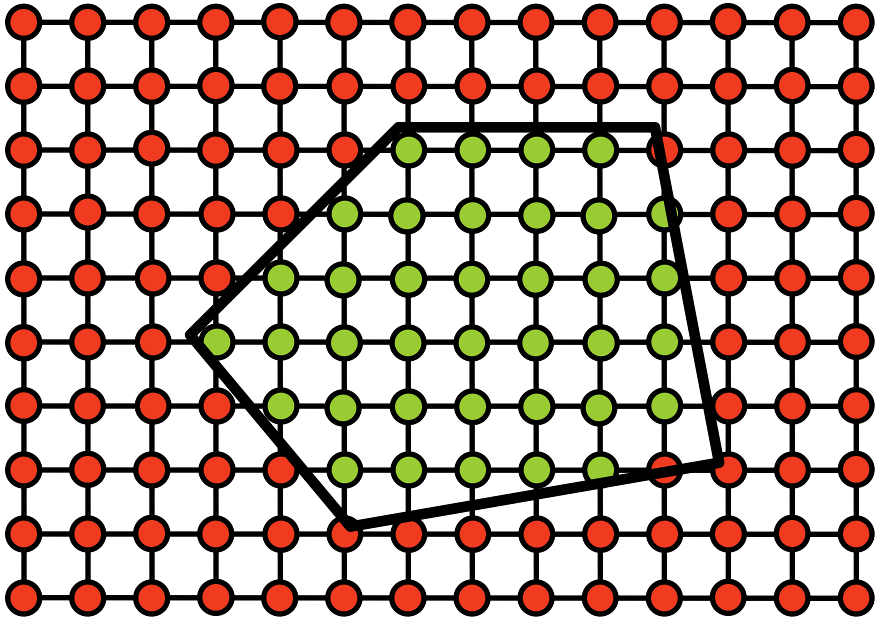
Sample the SDF



Sample the SDF



Sample the SDF

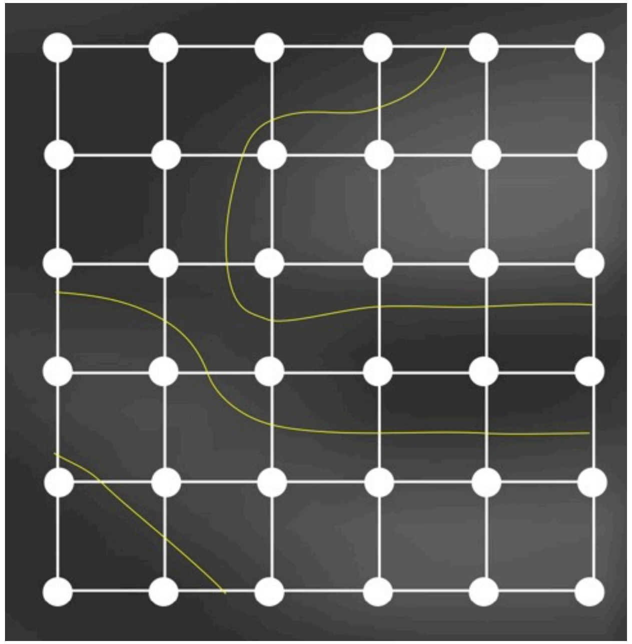


Marching Squares (2D)

Given a function: $f(x)$

- $f(\mathbf{x}) < 0$ inside
- $f(\mathbf{x}) > 0$ outside

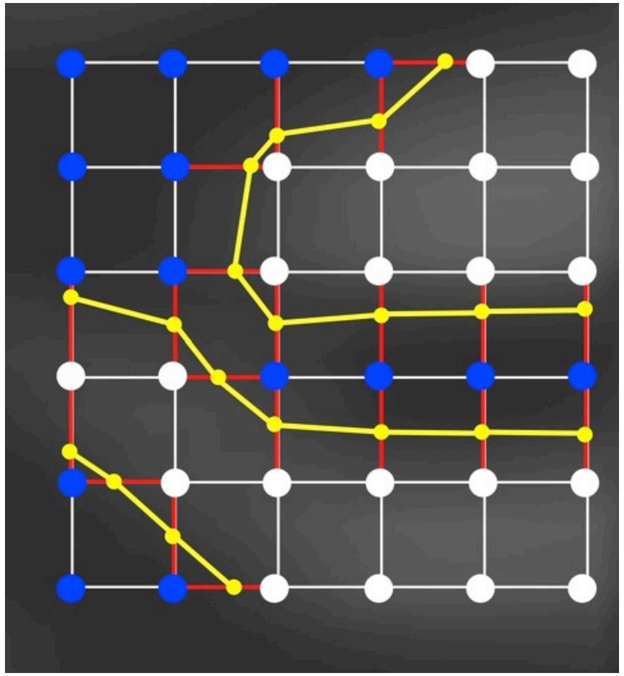
1. Discretize space.
2. Evaluate $f(x)$ on a grid.



Marching Squares (2D)

Given a function: $f(x)$

- $f(\mathbf{x}) < 0$ inside
- $f(\mathbf{x}) > 0$ outside



1. Discretize space.
2. Evaluate $f(x)$ on a grid.
3. Classify grid points (+/-)
4. Classify grid edges
5. Compute intersections
6. Connect intersections

Marching Squares (2D)

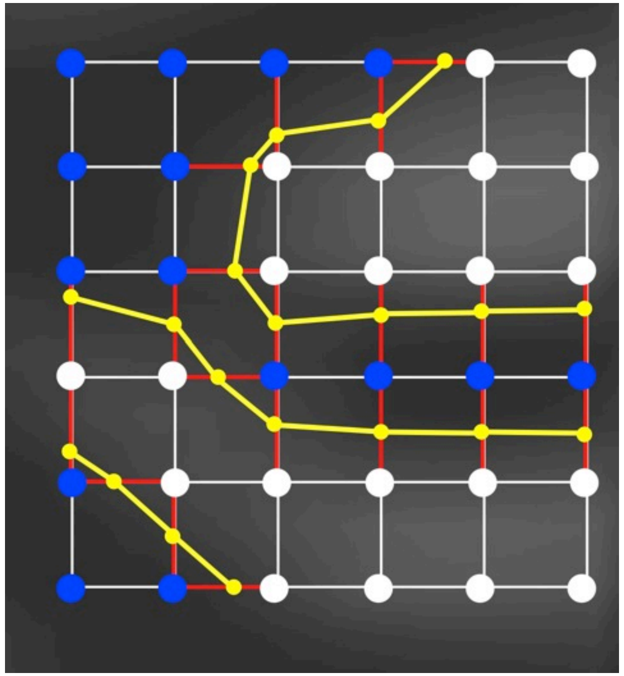
Computing the intersections:

- Edges with a sign switch contain intersections.

$$f(x_1) < 0, f(x_2) > 0 \Rightarrow$$
$$f(x_1 + t(x_2 - x_1)) = 0$$
$$\text{for some } 0 \leq t \leq 1$$

- Simplest way to compute t: assume f is linear between x1 and x2:

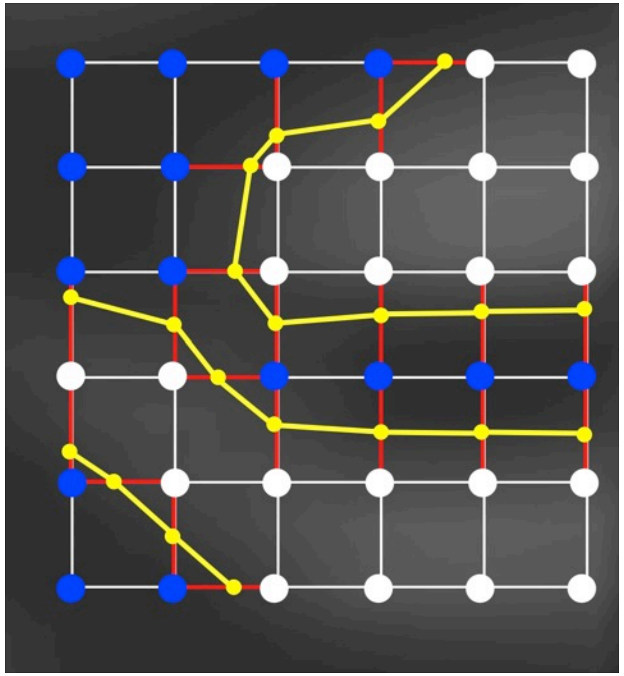
$$t = \frac{f(x_1)}{f(x_2) - f(x_1)}$$



Marching Squares (2D)

Connecting the intersections:

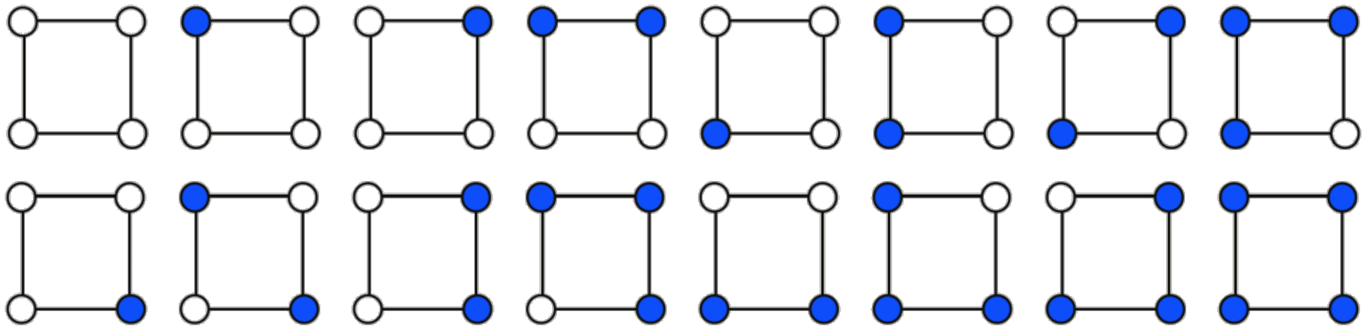
- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.



Marching Squares (2D)

Connecting the intersections:

- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.
- Group those leading to the same intersections

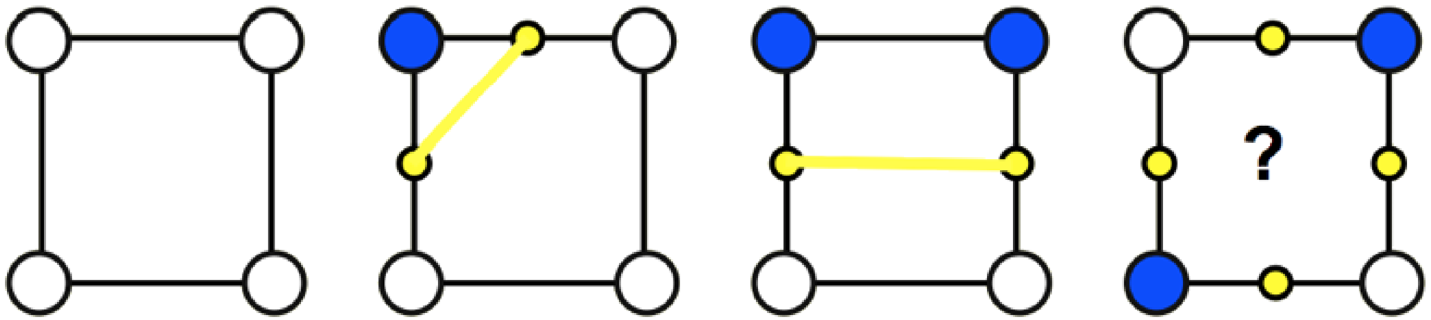


16 cases

Marching Squares (2D)

Connecting the intersections:

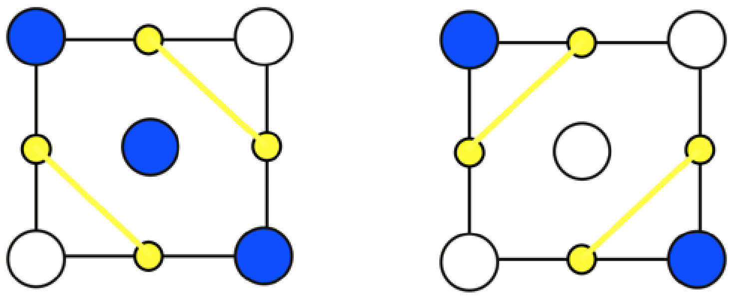
- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.
- Group those leading to the same intersections.
- Group equivalent after rotation.
- Connect intersections

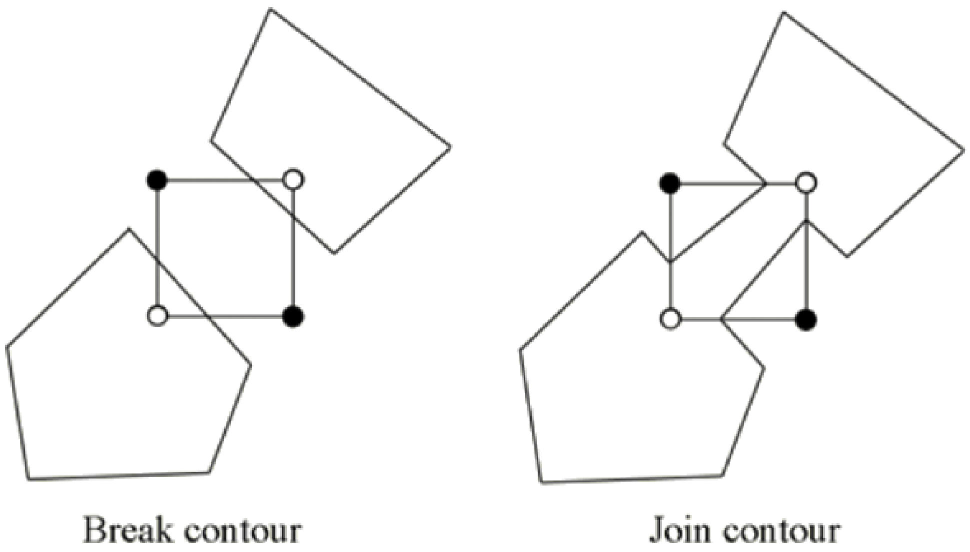


Marching Squares (2D)

Connecting the intersections:

Ambiguous cases:





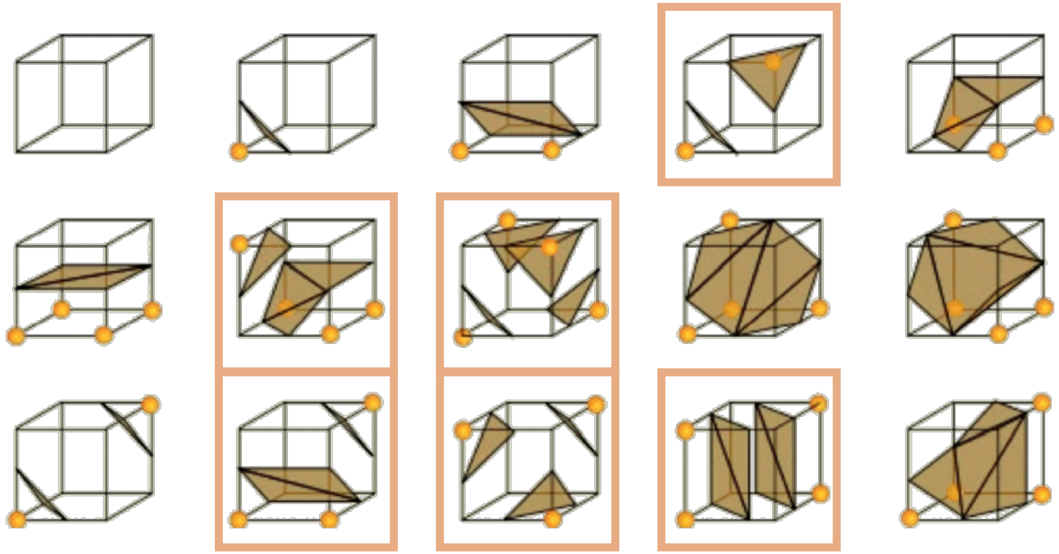
Two options:

1) Can resolve ambiguity by subsampling inside the cell.
2) If subsampling is impossible, pick one of the two possibilities.

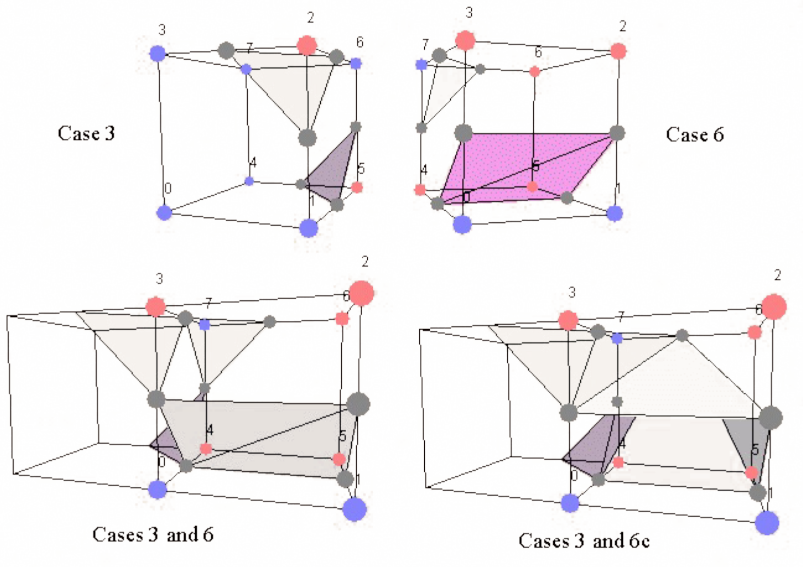
Marching Cubes (3D)

Same machinery: cells → **cubes** (voxels), lines → triangles

- 256 different cases - 15 after symmetries, 6 ambiguous cases
- More subsampling rules → 33 unique cases



the 15 cases



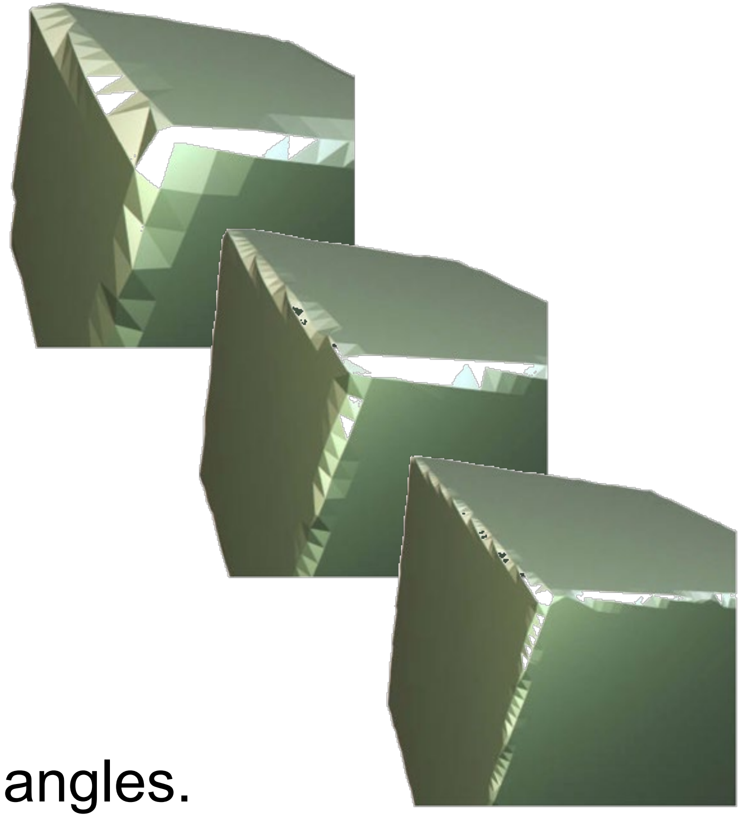
explore ambiguity to avoid holes!

Chernyaev, Marching Cubes 33,'95

Marching Cubes (3D)

Main Strengths:

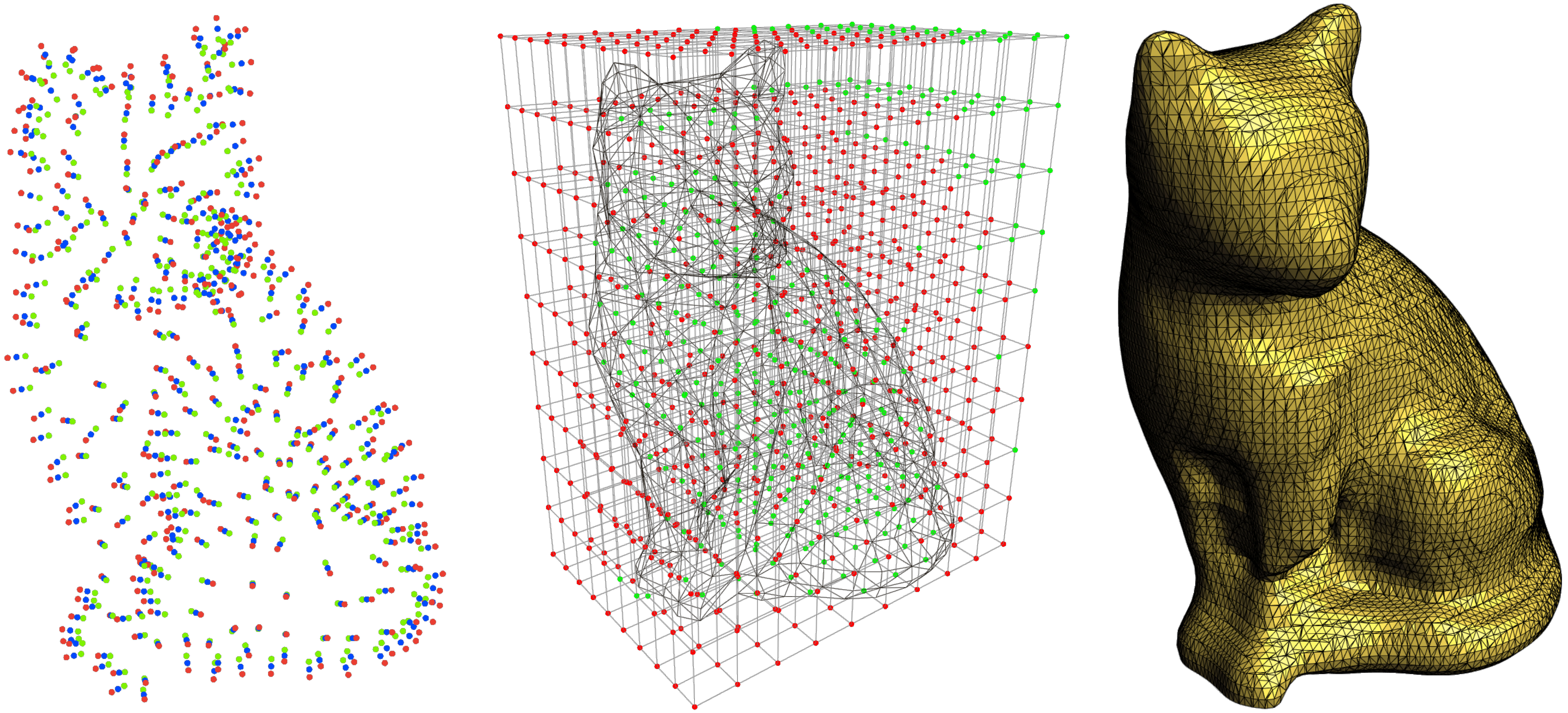
- Very multi-purpose.
- Extremely fast and parallelizable.
- Relatively simple to implement.
- Virtually parameter-free



Main Weaknesses:

- Can create badly shaped (skinny) triangles.
- Many special cases (implemented as big lookup tables).
- No sharp features.

Recap: Points→Implicit→Mesh

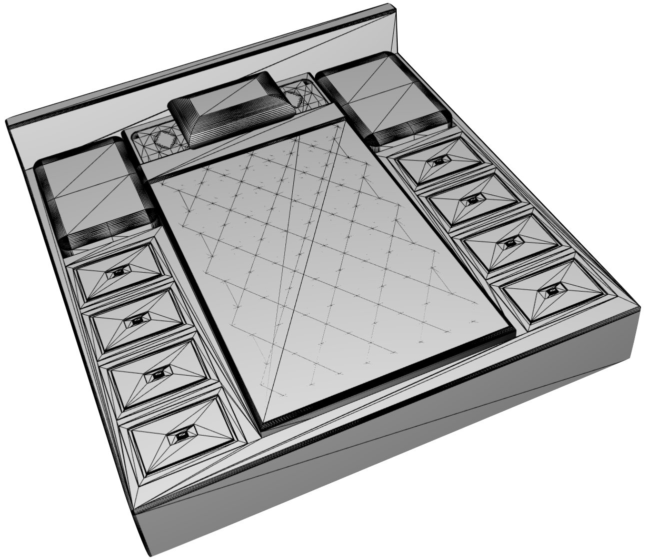


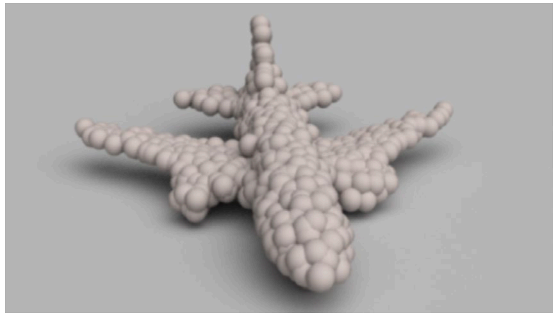
MESH-> POINT CLOUD

Sampling

From Surface to Point Cloud Why?

- Points are simple but expressive!
  - Few points can suffice
- Flexible, unstructured, few constraints
- Also: ML applications!

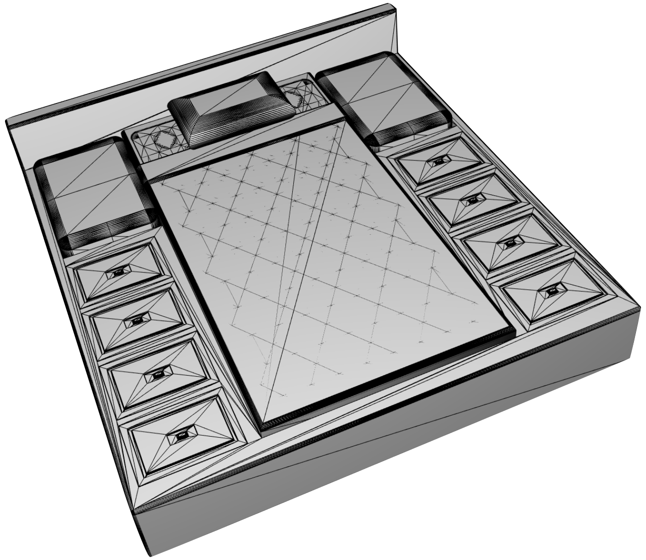


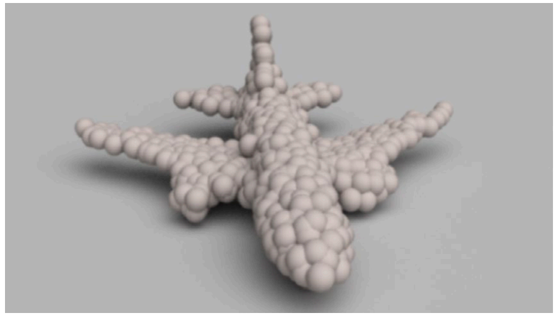


CAD meshes:
many components
bad triangles
connectivity problems

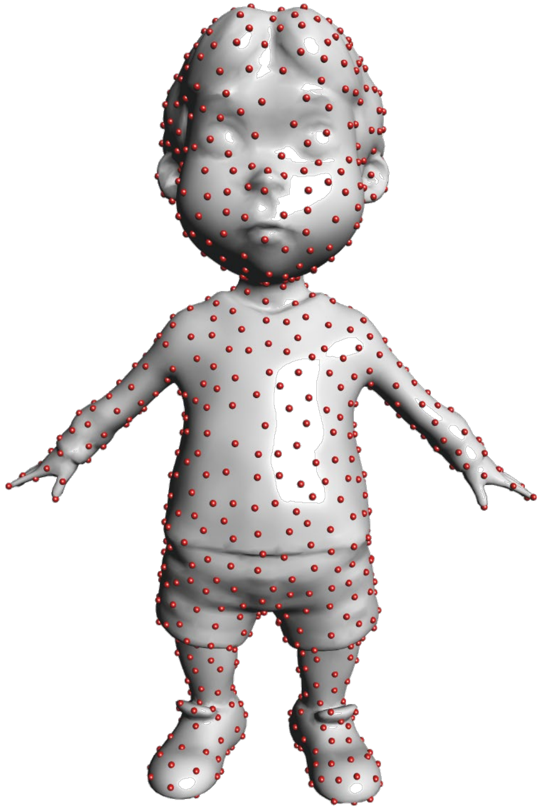
From Surface to Point Cloud Why?

- Points are simple but expressive!
  - Few points can suffice
- Flexible, unstructured, few constraints
- Also: ML applications!





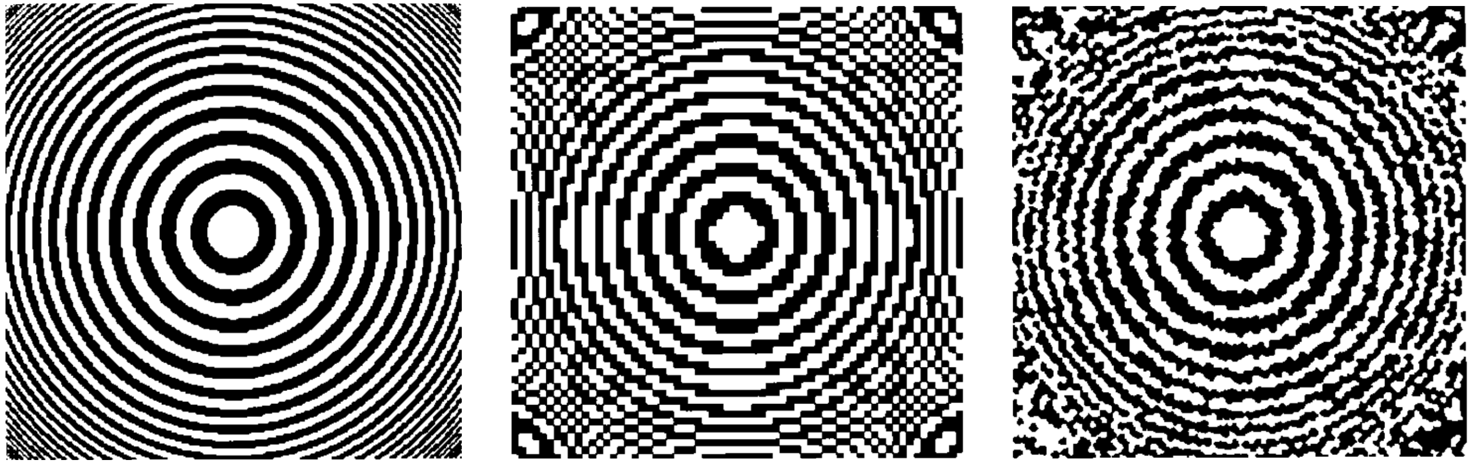
CAD meshes:
many components
bad triangles
connectivity problems



the problem:
sampling the mesh

Farthest Point Sampling

- Introduced for progressive transmission/acquisition of images
- Quality of approximation improves with increasing number of samples
  - as opposed eg. to raster scan
- Key Idea: repeatedly place next sample in the middle of the least-known area of the domain.



Gonzalez 1985, “Clustering to minimize the maximum intercluster distance”
Hochbaum and Shmoys 1985, “A best possible heuristic for the k-center problem”

Pipeline

1.Create an initial sample point set *S*

- Image corners + additional random point.

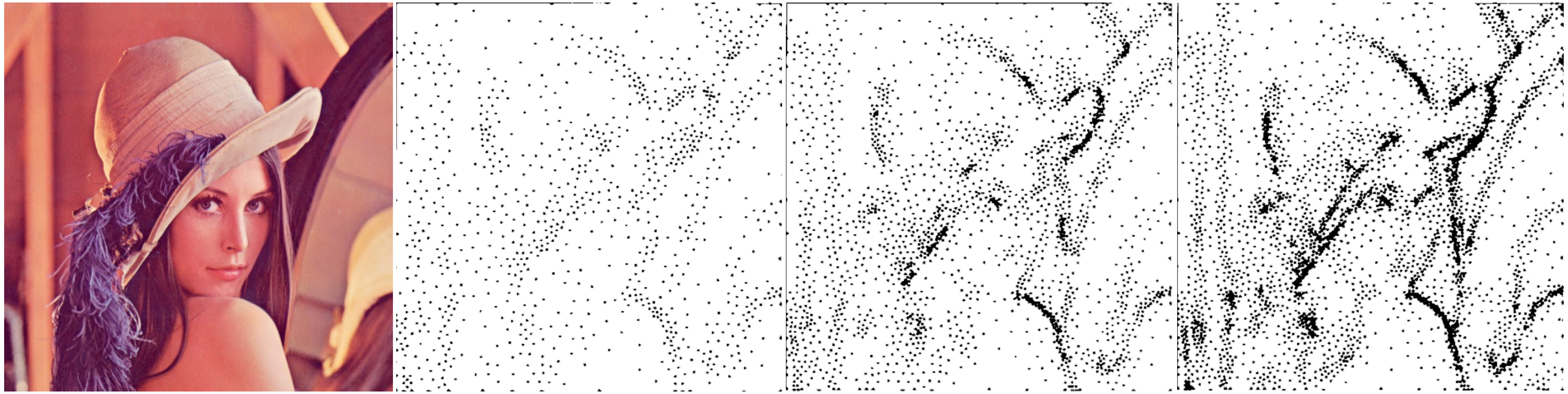
2. Find the point which is the farthest from all point in *S*

$$d(p,S) = \max_{q \in A}(d(q,S))$$
$$= \max_{q \in A}\left(\min_{0 \le i < N}(d(q,s_i))\right)$$

3. Insert the point to *S* and update the distances
4. While more points are needed, iterate

Farthest Point Sampling

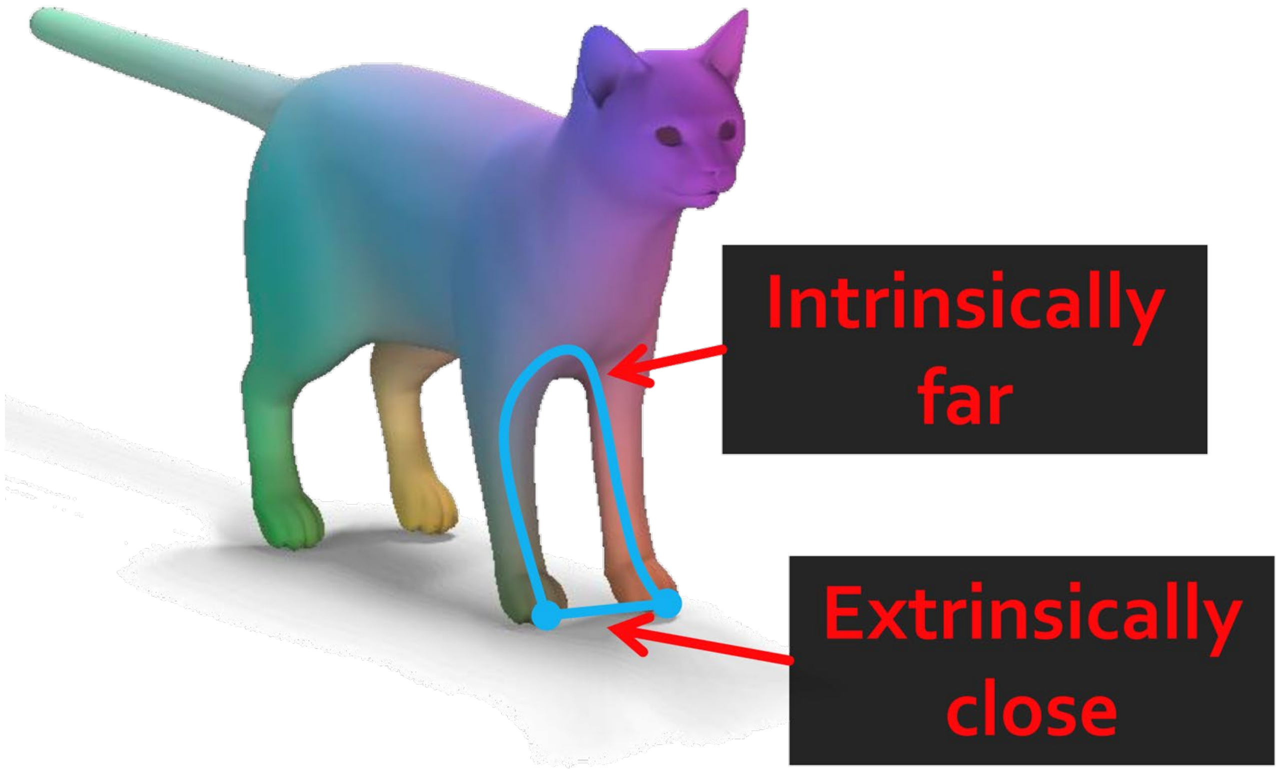
- Depends on a notion of distance on the sampling domain
- Can be made adaptive, via a weighted distance

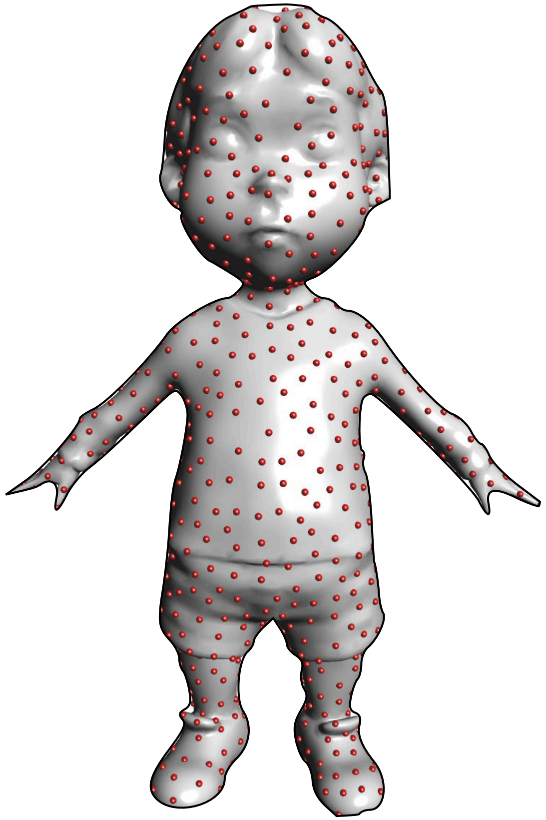


Eldar et al. 1997, "The Farthest Point Strategy for Progressive Image Sampling"

FPS on surfaces

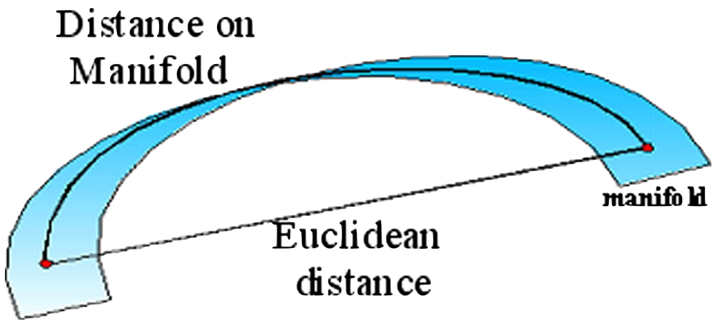
- What's an appropriate distance?

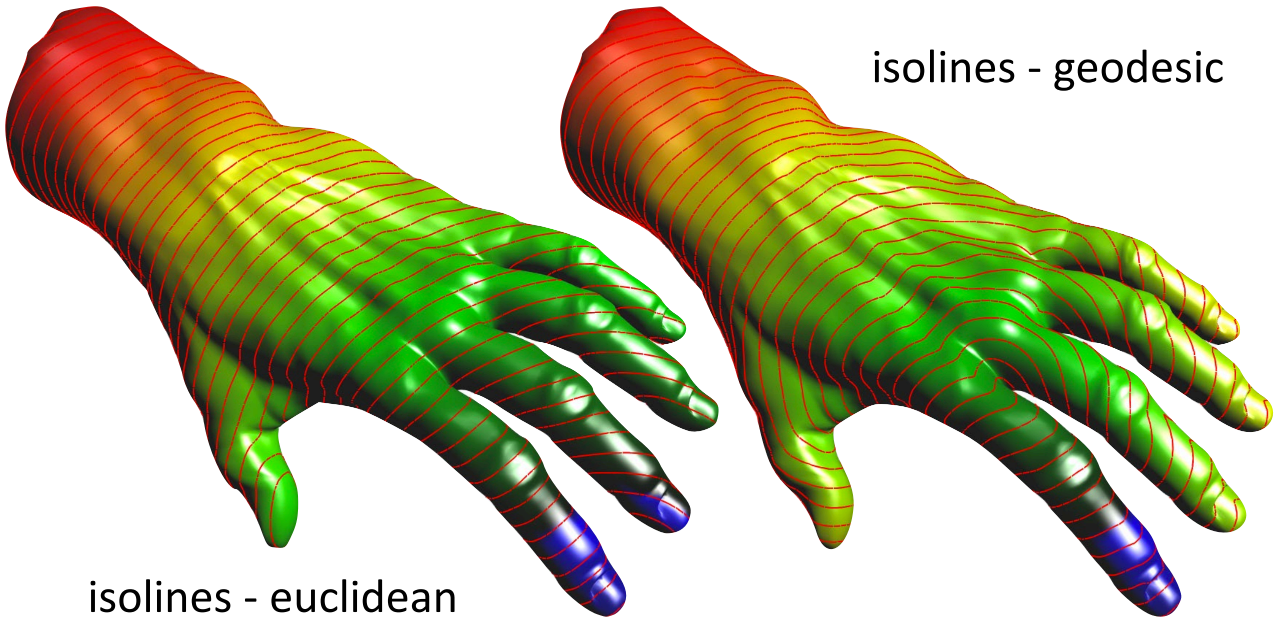




On-Surface Distances

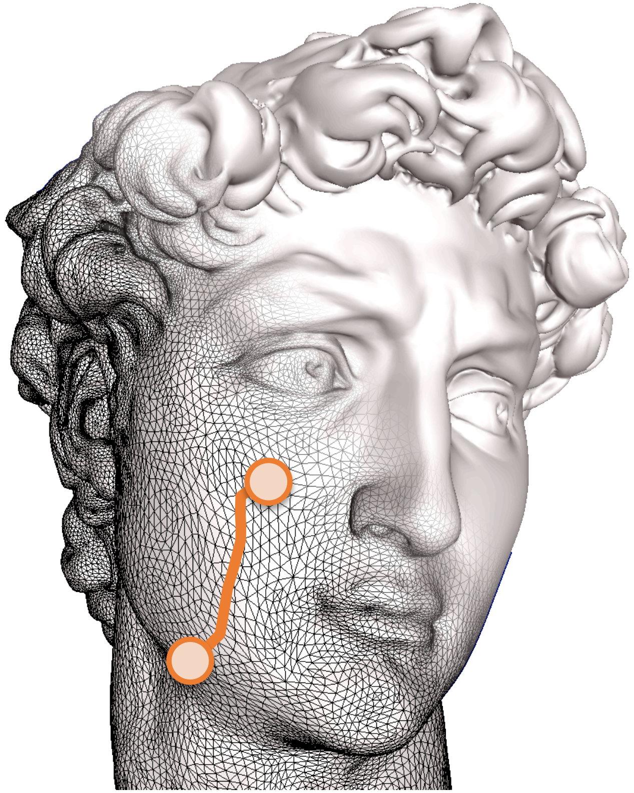
- Geodesics: Straightest and **locally shortest** curves





Discrete Geodesics

- Recall: a mesh is a graph!
- Approximate geodesics as paths along edges



v_0= initial vertex
d_i = current distance to vertex i
S = verticies with known optimal distance

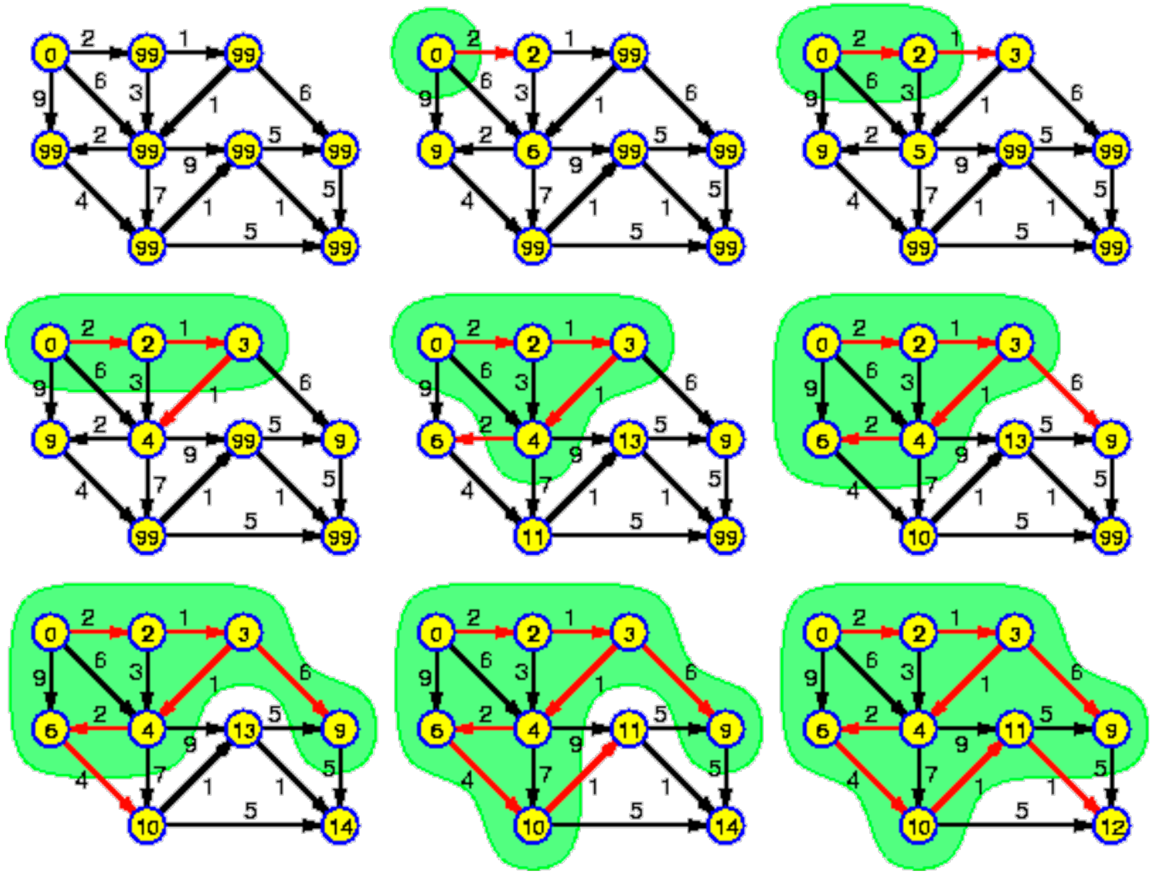
\# initialize
$d_0 = 0$
d_i = [inf for d in d_i]
$S = \{\}$

for each iteration k:
    \# update
    k = argmin(d_k), for v_k not in S
    S.append(v_k)
    for neighbors index v_l of v_k:
        d_l = min([d_l, d_k + d_{kl}])

Dijkstra's algorithm!

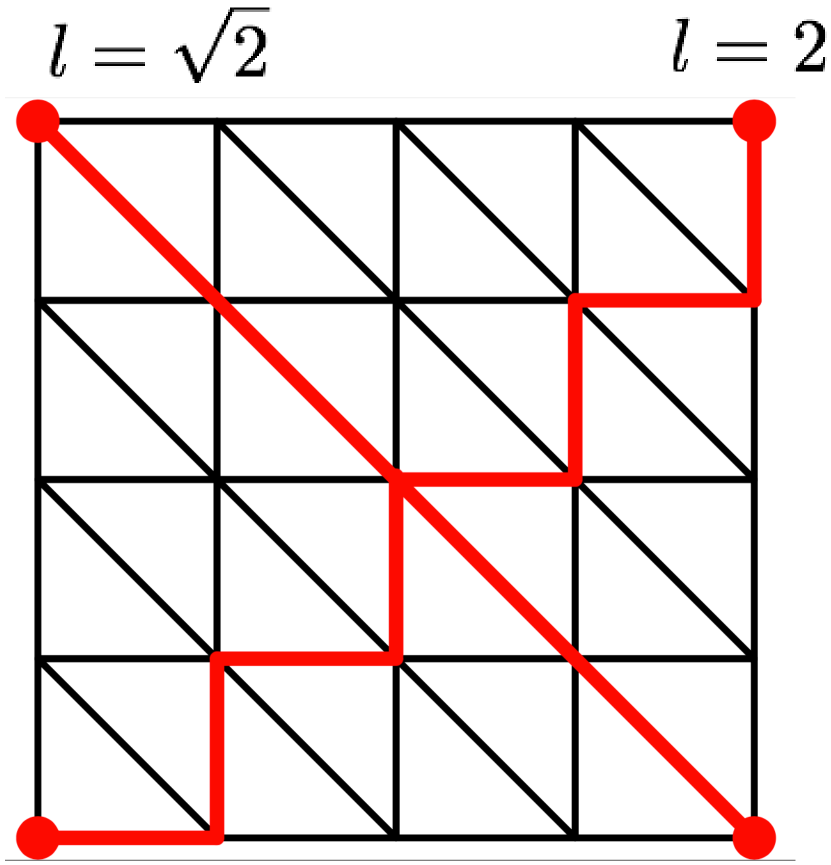
Dijkstra Geodesics

- Dikjstra as wave front propagation



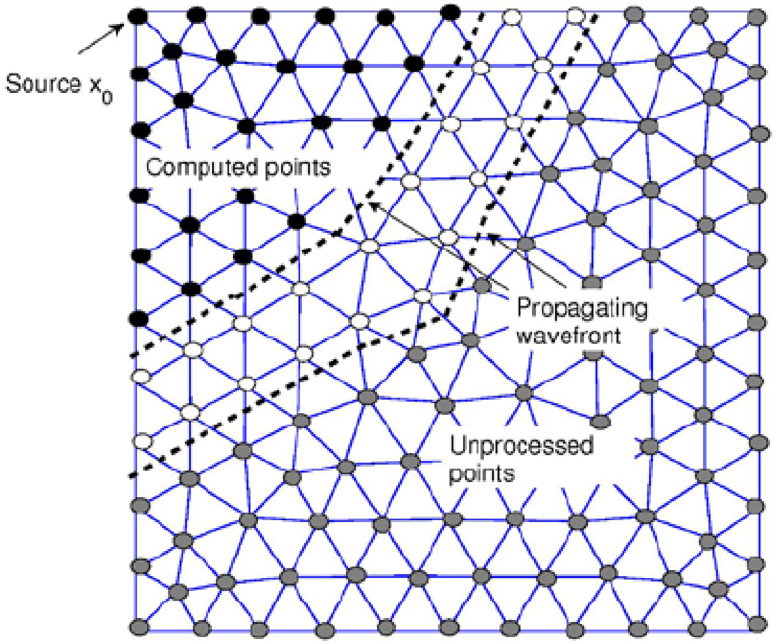
Dijkstra Geodesics

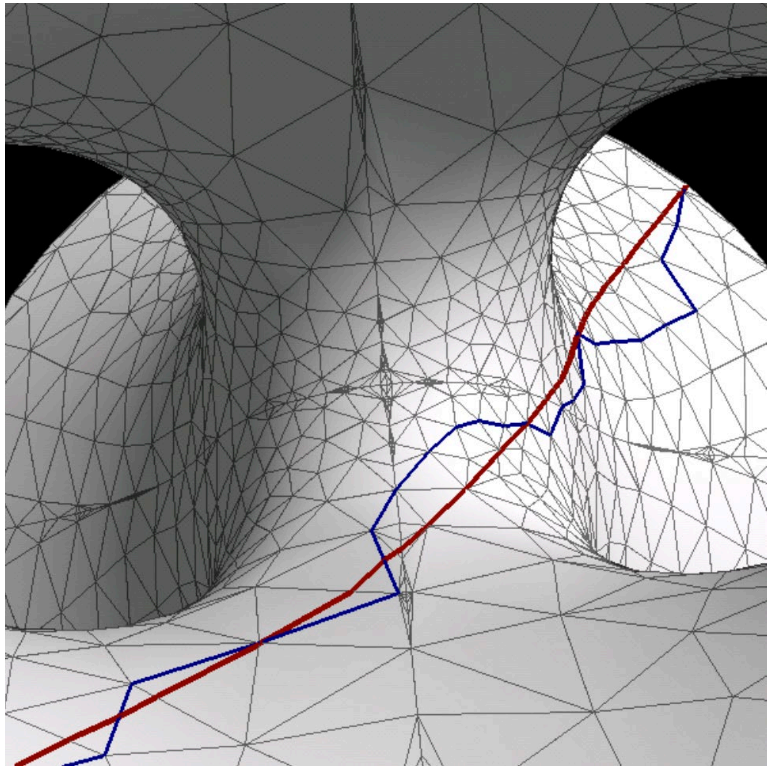
Can be asymmetric - no matter how fine the mesh!



Fast Marching Geodesics

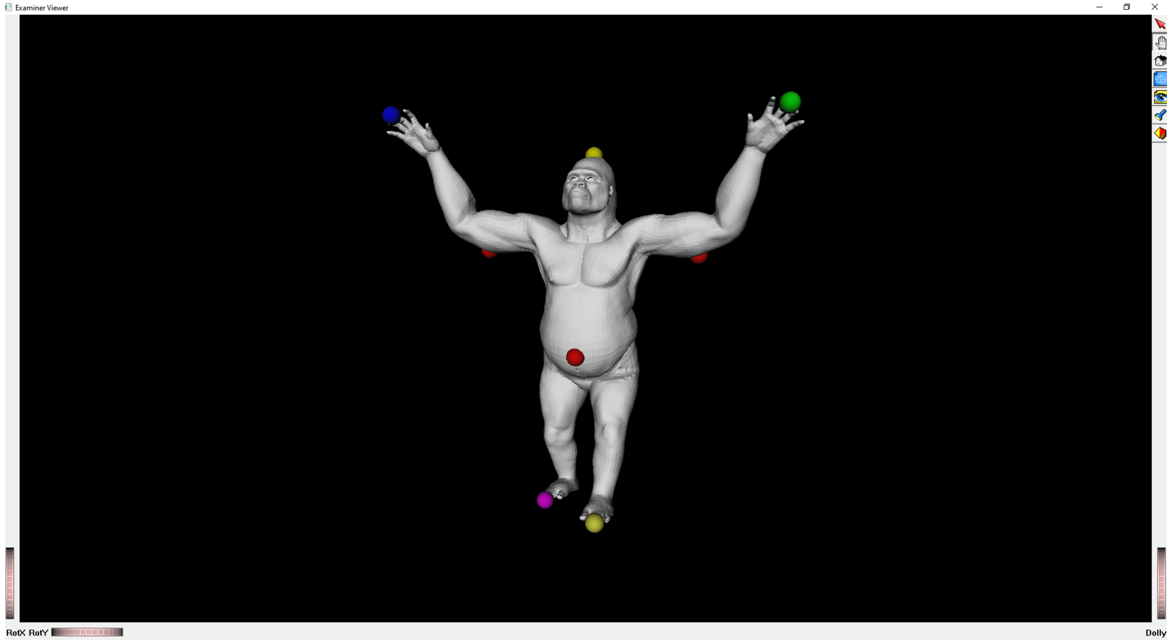
- A better approximation: allow fronts to cross triangles!

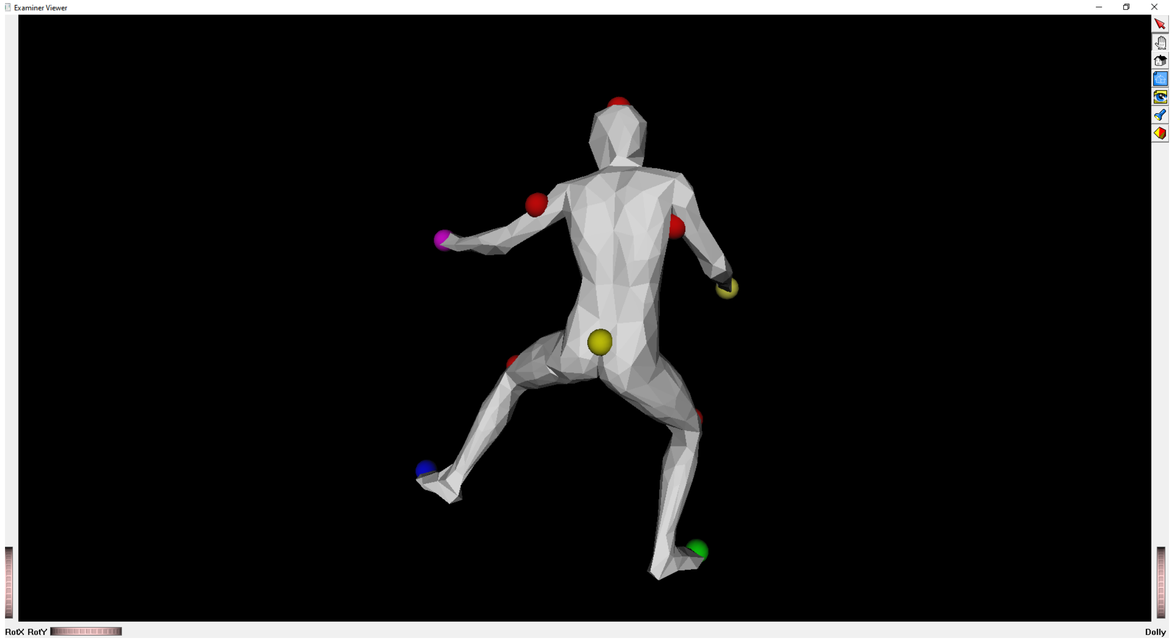




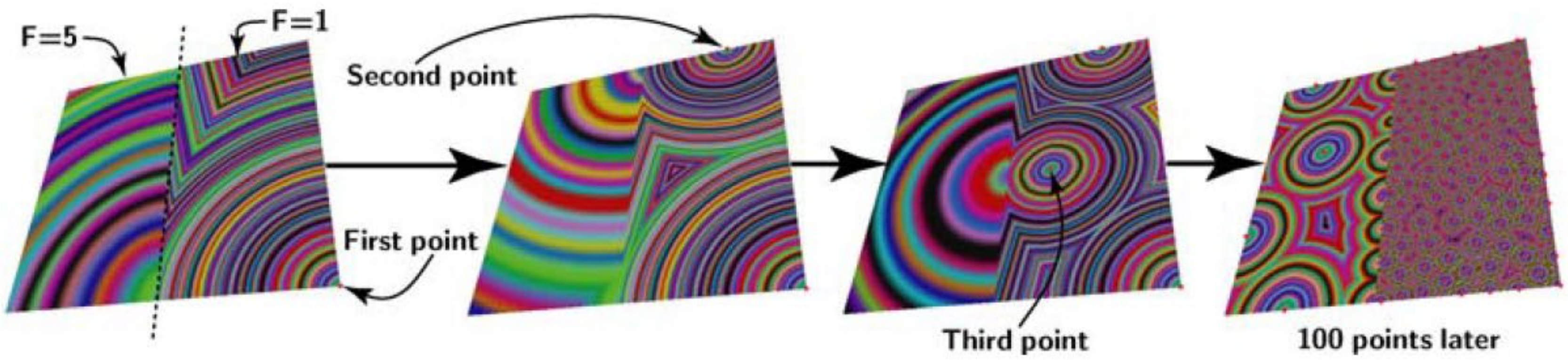
Kimmel and Sethian 1997, “Computing Geodesic Paths on Manifolds”

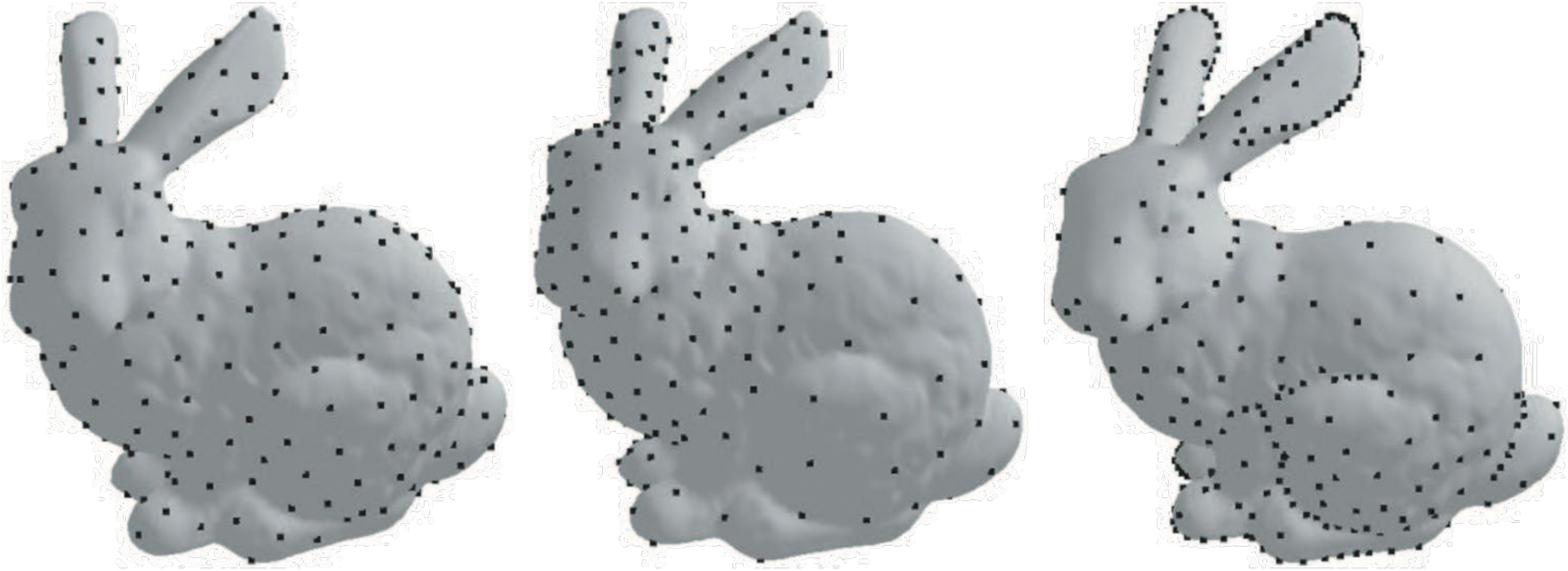
Initialization





Adaptive FPS





Peyré and Cohen 2003, Geodesic Remeshing Using Front Propagation

Faster Distance Approximations

Carne, Weischedel, and Wardetzky 2017,
The Heat Method for Distance Computation

Software

- Libigl http://libigl.github.io/libigl/tutorial/tutorial.html
  - MATLAB-style (flat) C++ library, based on indexed face set structure
- OpenMesh www.openmesh.org
  - Mesh processing, based on half-edge data structure
- CGAL www.cgal.org
  - Computational geometry
- MeshLab http://www.meshlab.net/
  - Viewing and processing meshes

Software

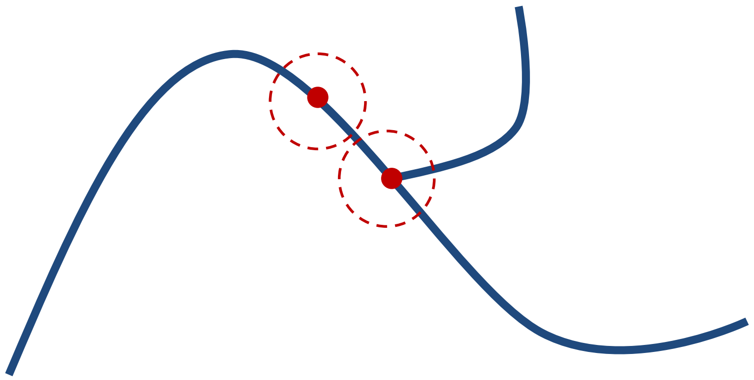
- Alec Jacobson’s GP toolbox
  - https://github.com/alecjacobson/gptoolbox
  - MATLAB, various mesh and matrix routines
- Gabriel Peyre’s Fast Marching Toolbox
  - https://www.mathworks.com/matlabcentral/fileexchange/6110-toolbox-fast-marching
  - On-surface distances (more next time!)
- OpenFlipper https://www.openflipper.org/
  - Various GP algorithms + Viewer

A Primer on Differential Geometry

Local Differential Notions

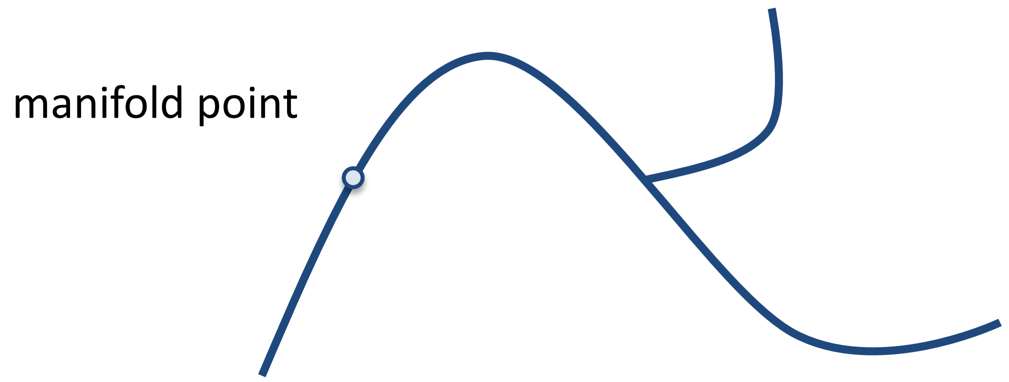
Differential Geometry Basics

- Geometry of manifolds
- Properties that can be discovered by local observation: point + a neighborhood



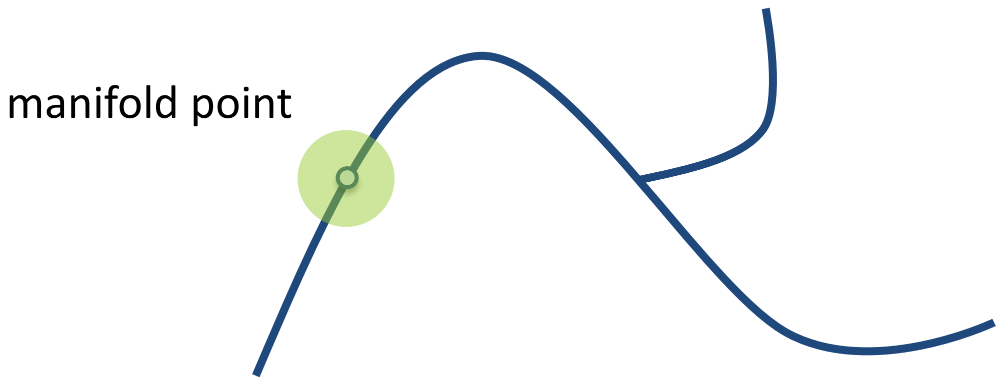
Differential Geometry Basics 2D

- Geometry of manifolds
- Properties that can be discovered by local observation: point + neighborhood



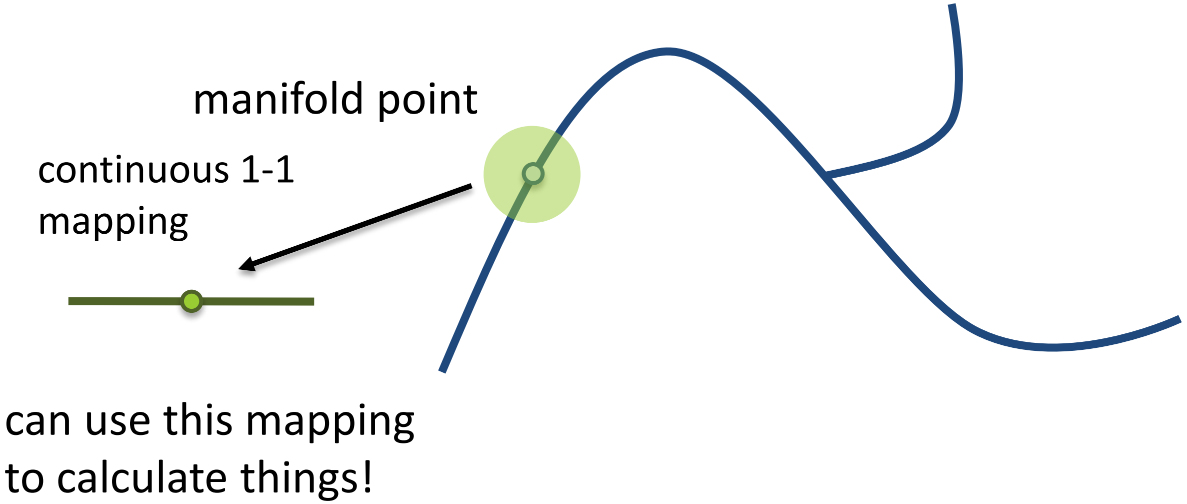
Differential Geometry Basics 2D

- Geometry of manifolds
- Properties that can be discovered by local observation: point + neighborhood



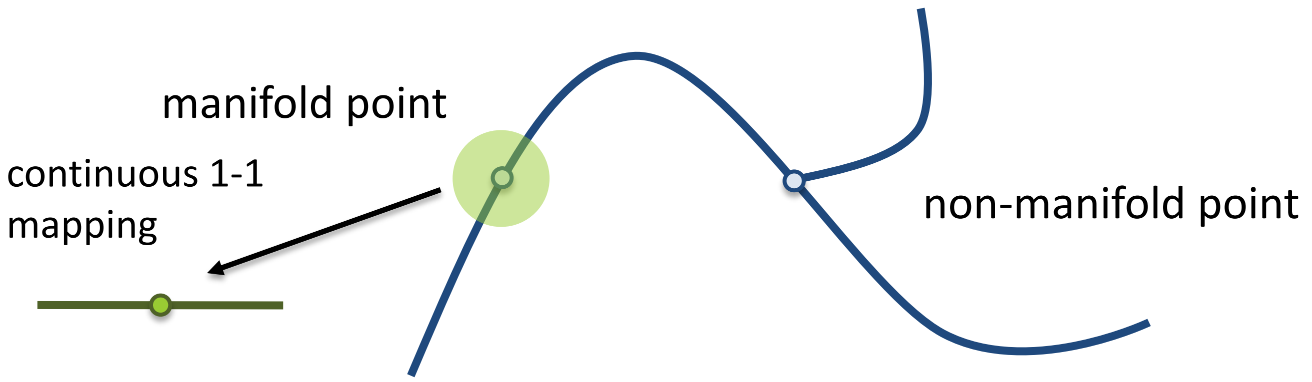
Differential Geometry Basics 2D

- Geometry of manifolds
- Properties that can be discovered by local observation: point + neighborhood



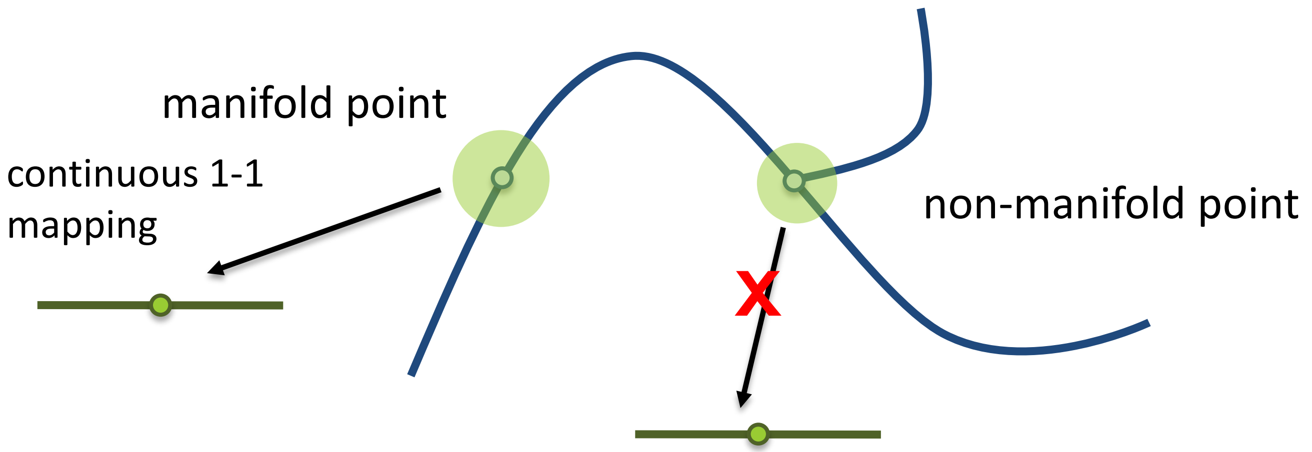
Differential Geometry Basics 2D

- Geometry of manifolds
- Properties that can be discovered by local observation: point + neighborhood



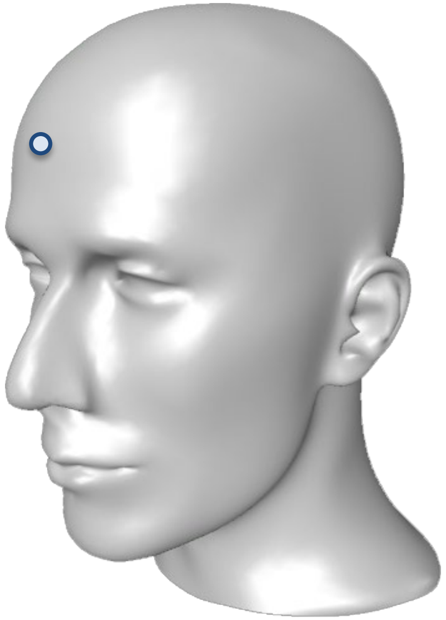
Differential Geometry Basics

- Geometry of manifolds
- Properties that can be discovered by local observation: point + neighborhood



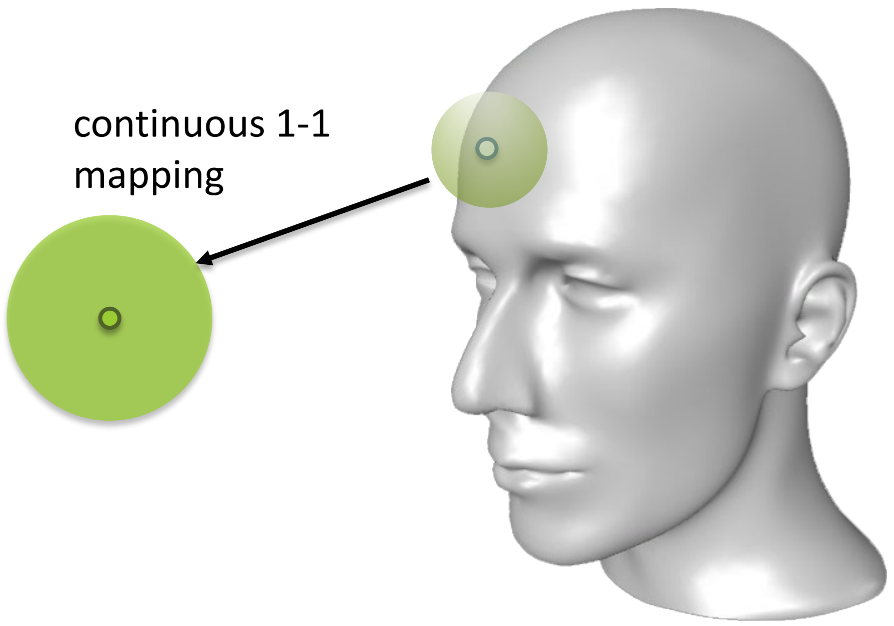
Differential Geometry Basics 3D

- Geometry of manifolds
- Properties that can be discovered by local observation: point + neighborhood



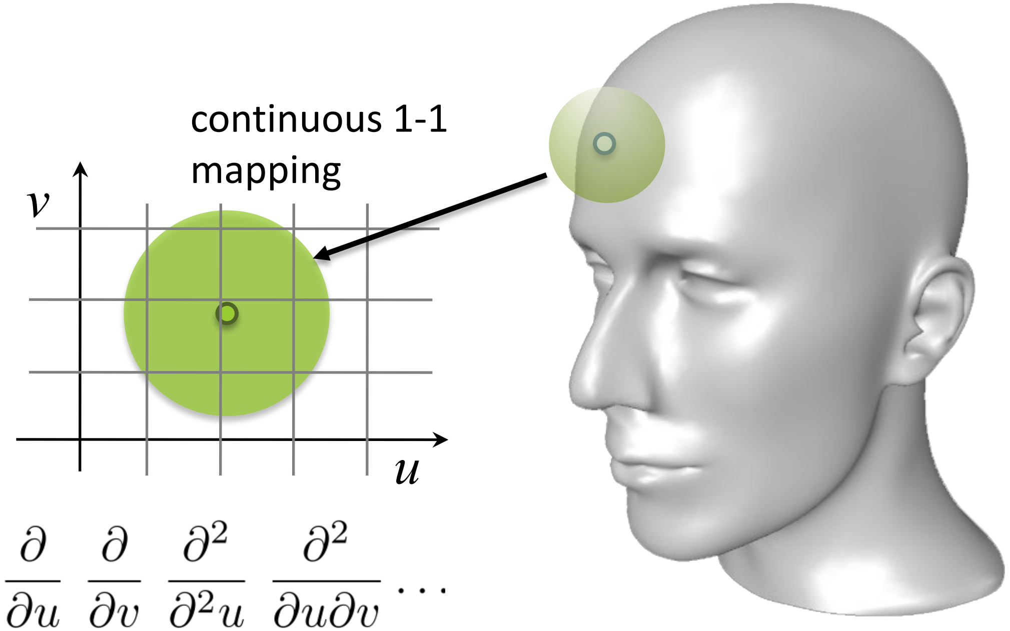
Differential Geometry Basics 3D

- Geometry of manifolds
- Properties that can be discovered by local observation: point + neighborhood



Differential Geometry Basics 3D

- Geometry of manifolds
- Properties that can be discovered by local observation: point + neighborhood



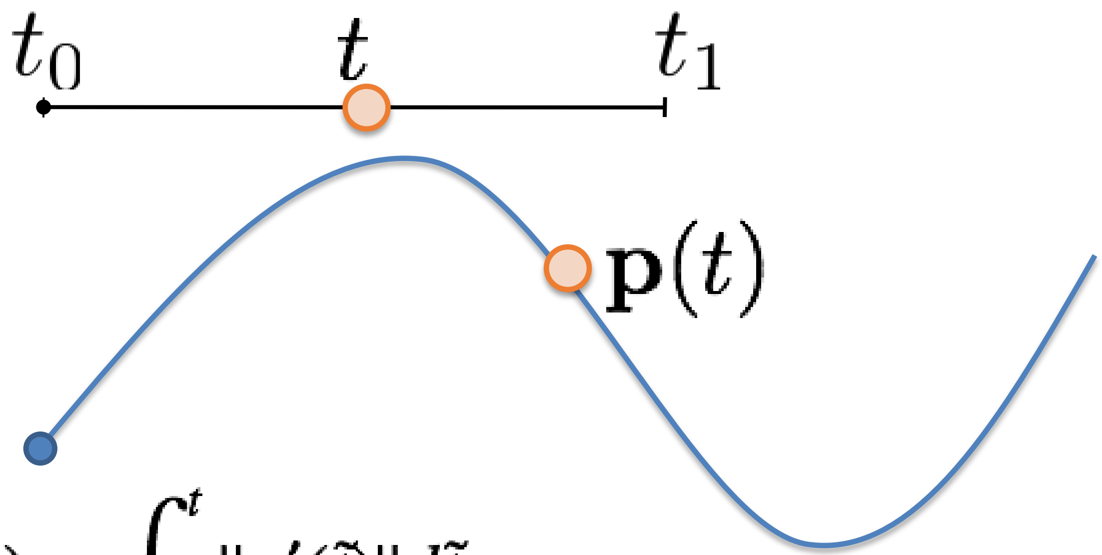
$$\frac{\partial}{\partial u} \quad \frac{\partial}{\partial v} \quad \frac{\partial^2}{\partial^2 u} \quad \frac{\partial^2}{\partial u \partial v} \quad \ldots$$

If a sufficiently smooth mapping can be constructed, we can look at its first and second derivatives

Local quantities:
Tangents, normals, curvatures, curve angles, distances

Parametric Curves

- 2D: $\mathbf{p}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}, \ t \in [t_0, t_1]$
- $\mathbf{p}(t)$ must be continuous



$$len\left(\mathbf{p}(t_0), \mathbf{p}(t)\right) = \int_{t_0}^{t} ||\mathbf{p}'(\tilde{t})|| d\tilde{t}$$

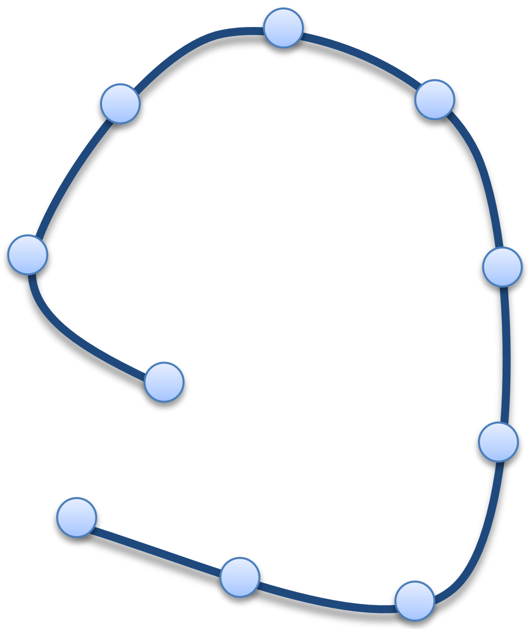
Arc Length Parameterization

- Equal pace of the parameter along the curve

- $len\,(\mathbf{p}(s_1), \mathbf{p}(s_2)) = |s_1 - s_2|$

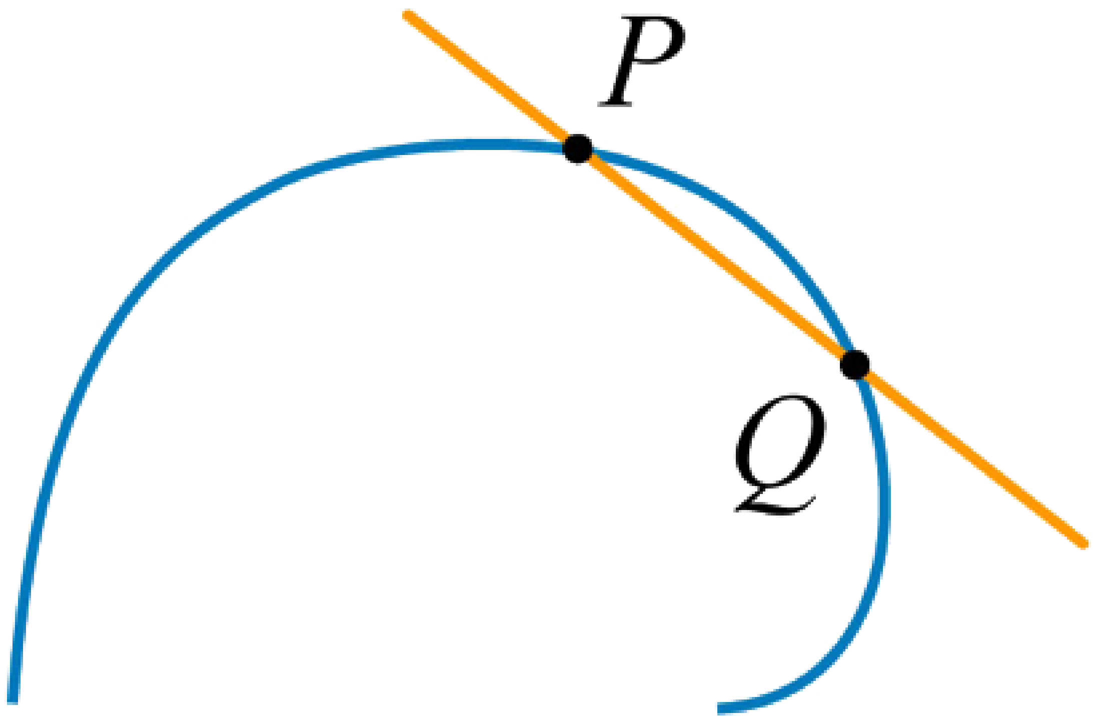
- Now parameter goes from 0 to L

$$\|\mathbf{p}'(s)\| = 1$$



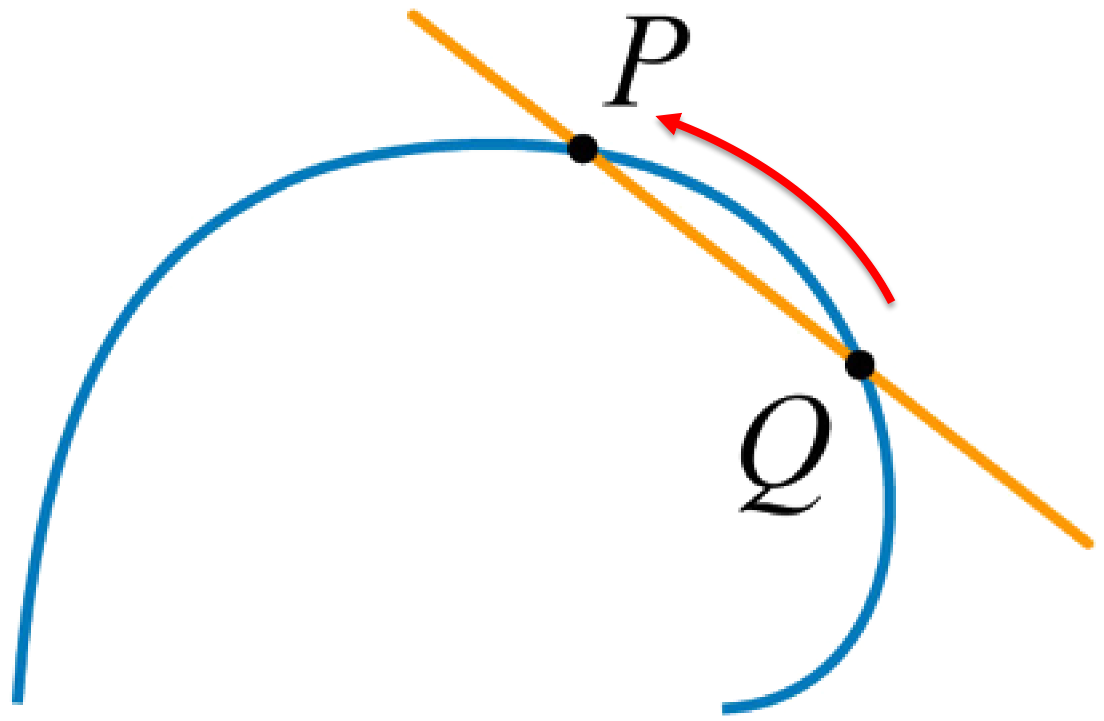
Secant

- A line through two points on the curve.



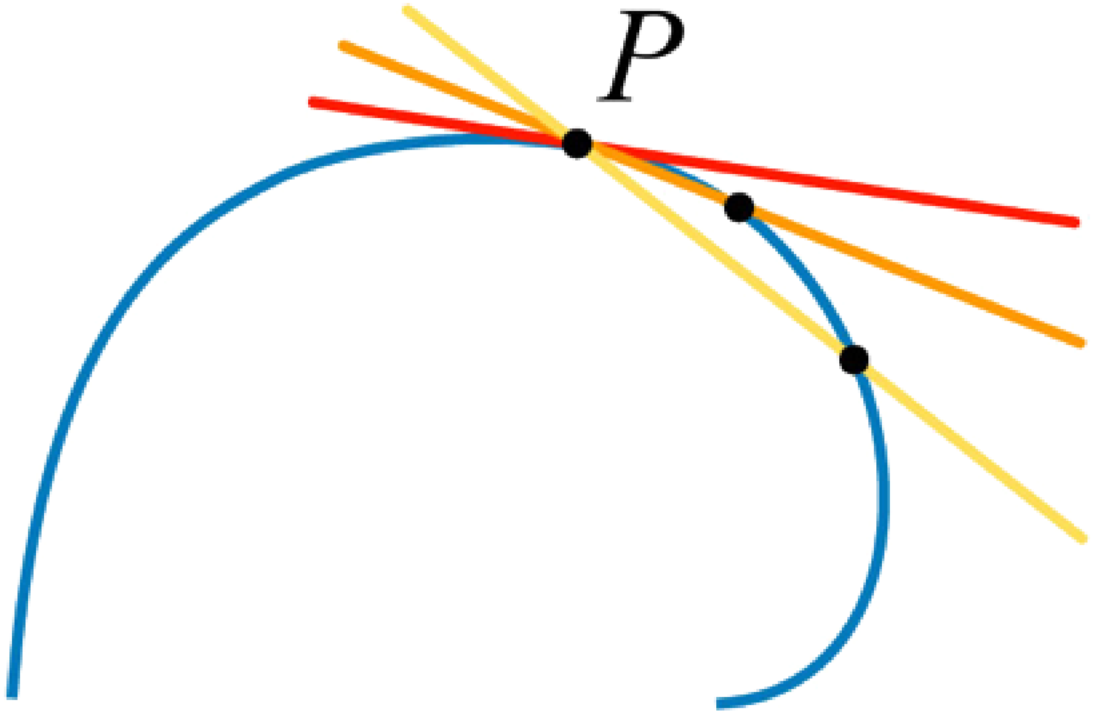
Secant

- A line through two points on the curve.



Tangent

- The limiting secant as the two points come together.



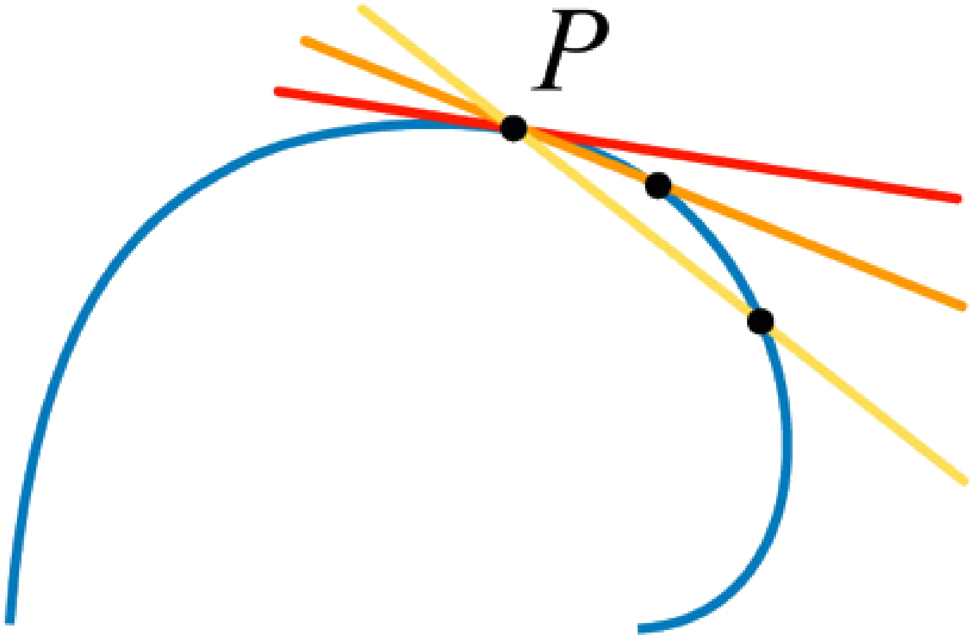
Secant and Tangent – Parametric Form

- Secant: $\mathbf{p}(t) - \mathbf{p}(s)$
- Tangent: $\mathbf{p}'(t) = (x'(t), y'(t), \ldots)^{\mathrm{T}}$
- If t is arc-length:

  $\|\mathbf{p}'(t)\| = 1$

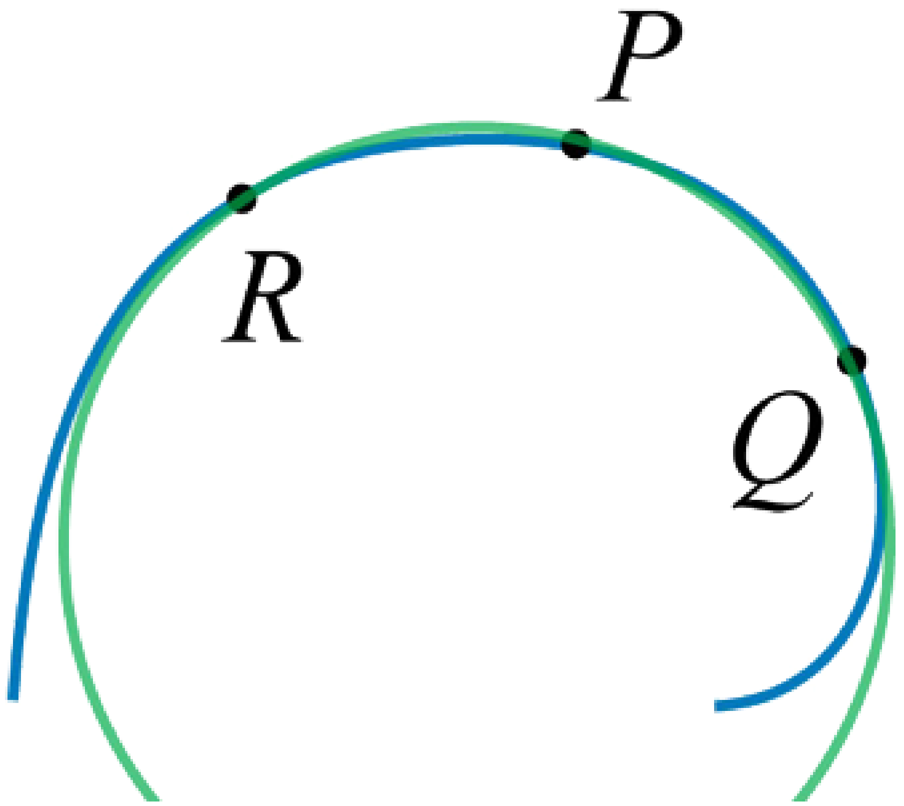
Recall

$$len(\mathbf{p}(t_0), \mathbf{p}(t)) = \int_0^t \|\mathbf{p}'(t)\| dt$$



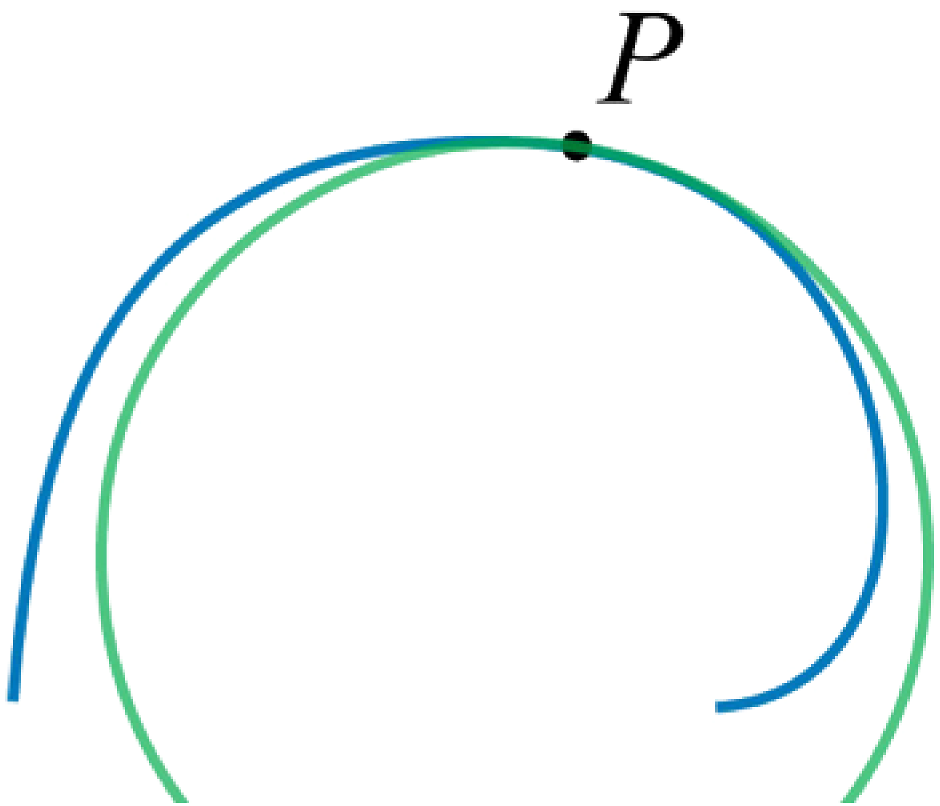
Circle of Curvature

- Consider the circle passing through three points on the curve…

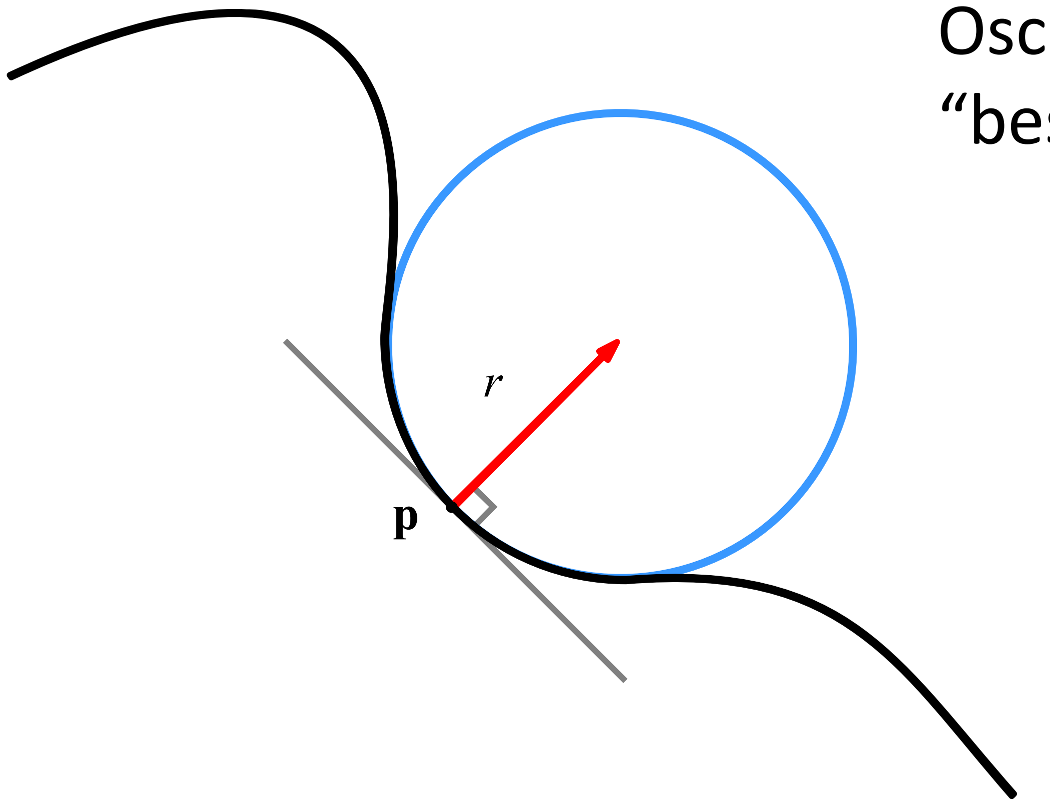


Circle of Curvature

- …the limiting circle as three points come together.



Tangent, normal, radius of curvature

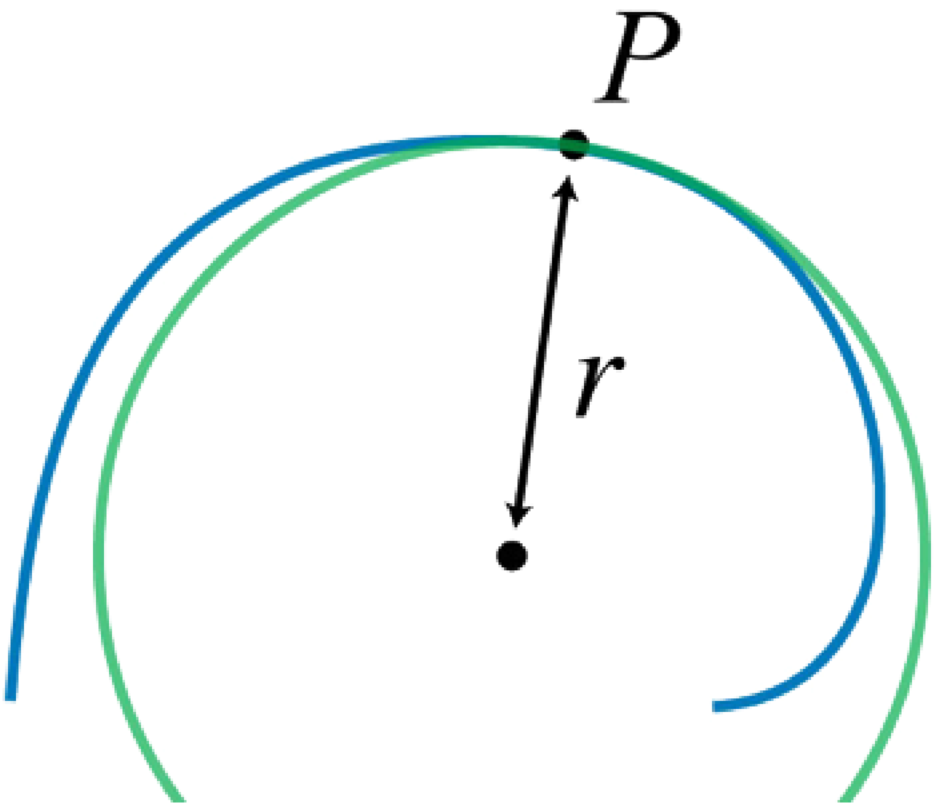


Osculating circle
“best fitting circle”

Radius of Curvature, $r = 1/\kappa$

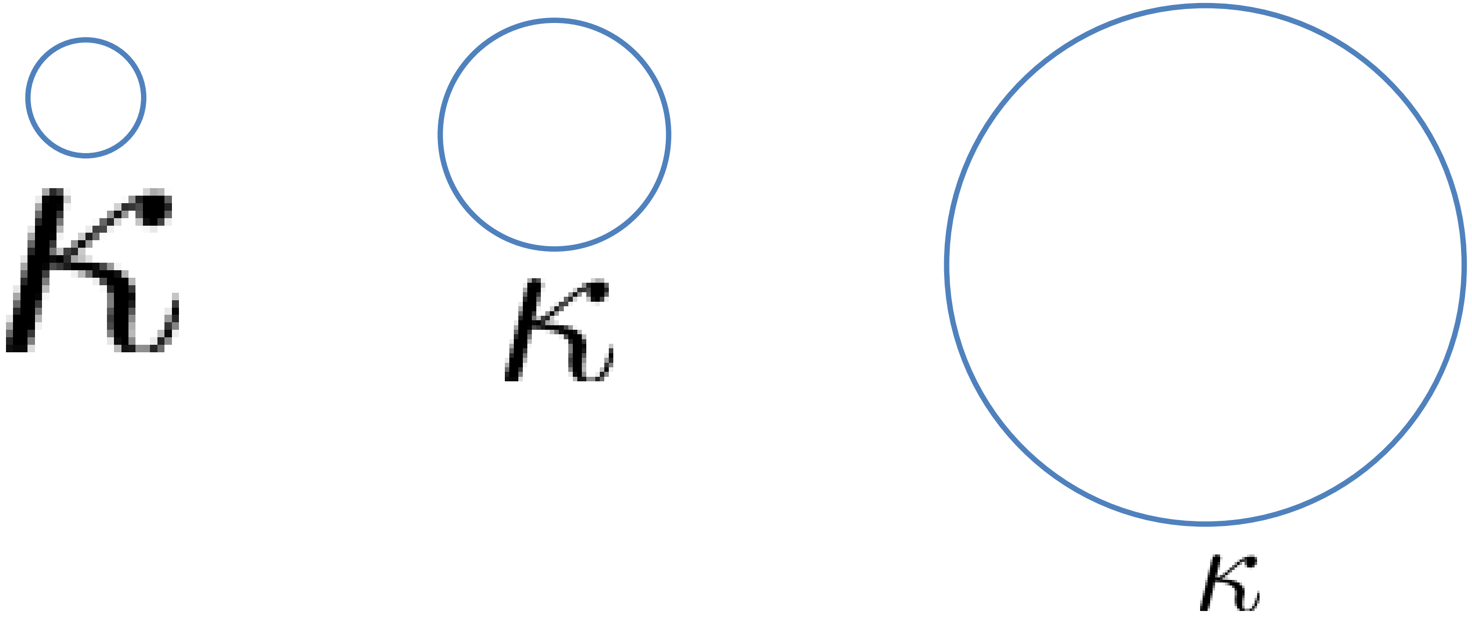
Curvature

$$\kappa = \frac{1}{r}$$



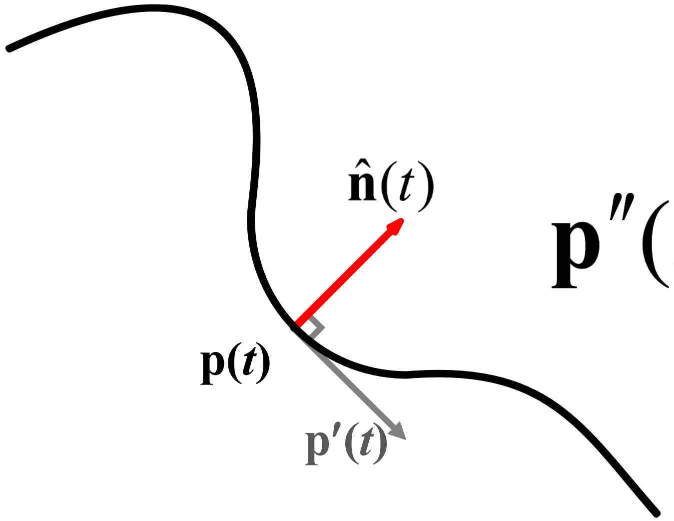
Curvature is Scale Dependent

$$\kappa = \frac{1}{r}$$

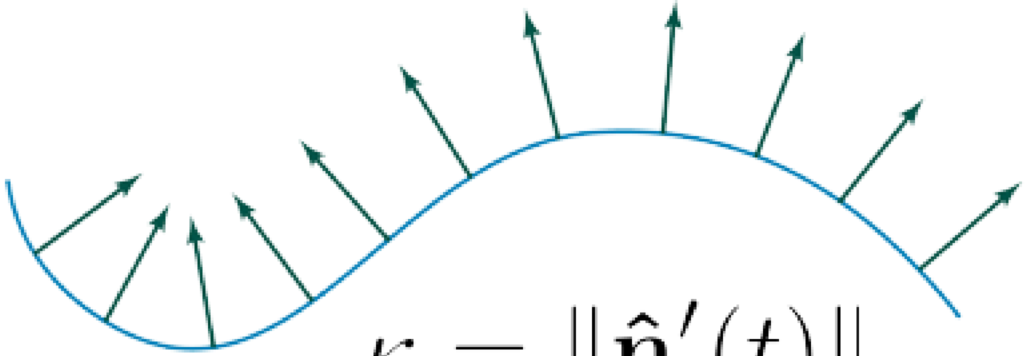


Curvature and Normal

- Assuming t is arc-length parameter:



$$\mathbf{p}''(t) = \kappa \hat{\mathbf{n}}(t)$$ normal to the curve



$$\kappa = \|\hat{\mathbf{n}}'(t)\|$$

Surfaces, Parametric Form

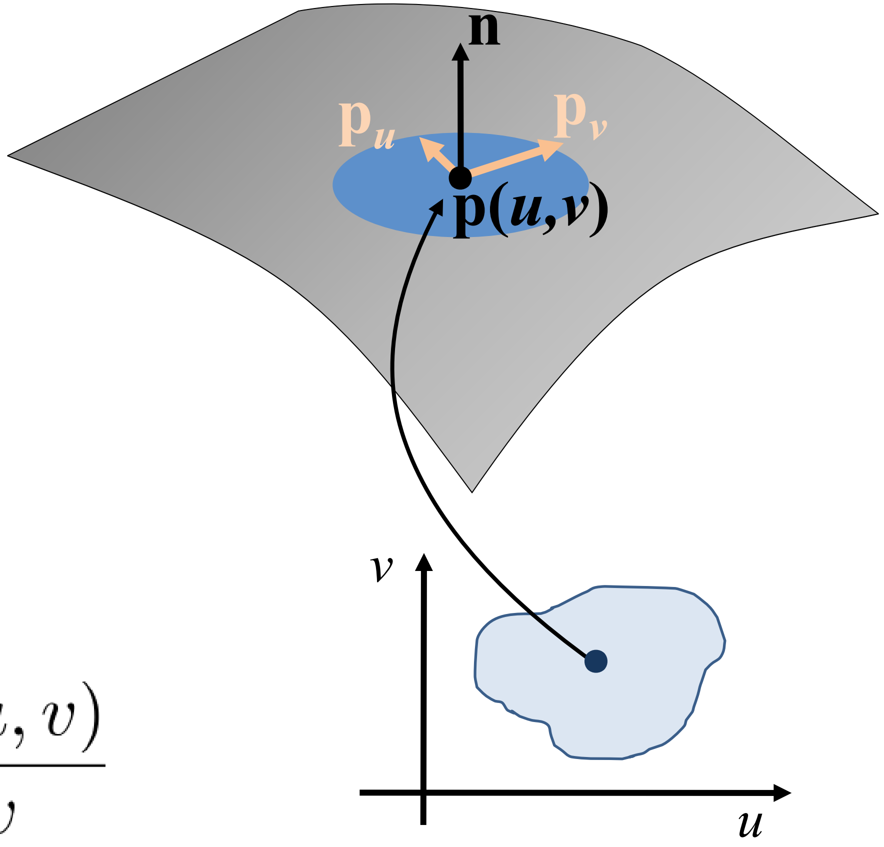
- Continuous surface

$$\mathbf{p}(u,v) = \begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{pmatrix}, \quad (u,v) \in \mathbb{R}^2$$

- Tangent plane at point $\mathbf{p}(u,v)$ is spanned by

$$\mathbf{p}_u = \frac{\partial \mathbf{p}(u,v)}{\partial u}, \quad \mathbf{p}_v = \frac{\partial \mathbf{p}(u,v)}{\partial v}$$

These vectors don't have to be orthogonal



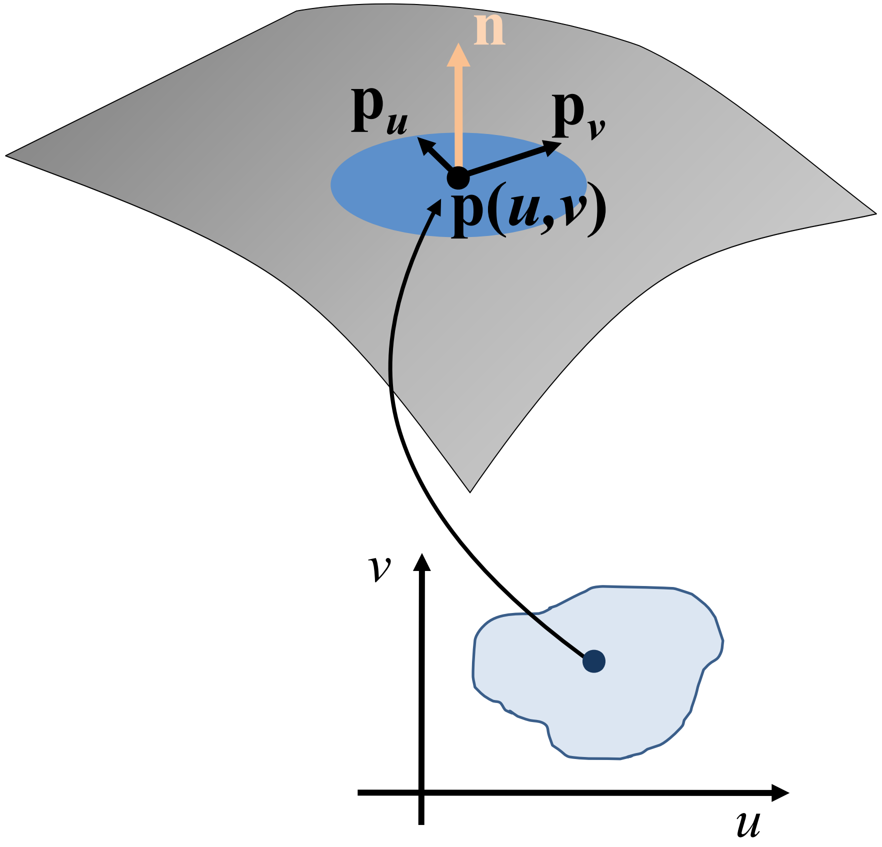
Surface Normals

- Surface normal:

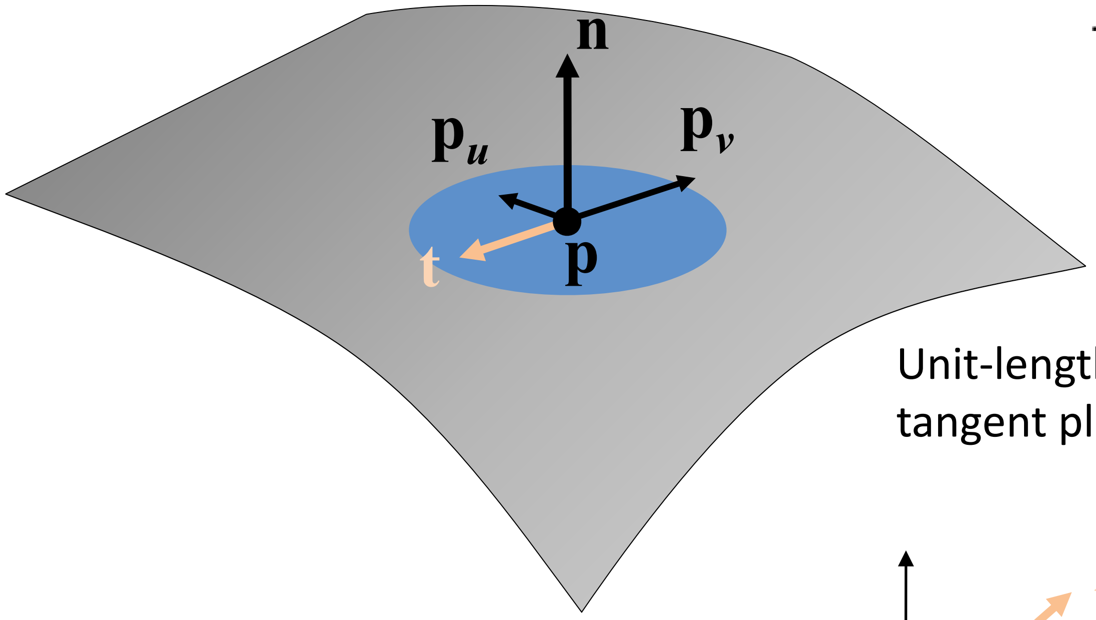
$$\mathbf{n}(u,v) = \frac{\mathbf{p}_u \times \mathbf{p}_v}{\|\mathbf{p}_u \times \mathbf{p}_v\|}$$

  - Assuming *regular* parameterization, i.e.,

$$\mathbf{p}_u \times \mathbf{p}_v \neq 0$$



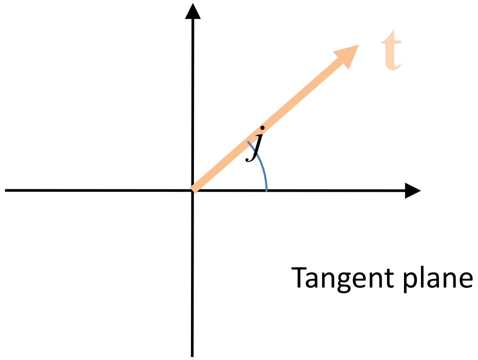
Normal Curvature



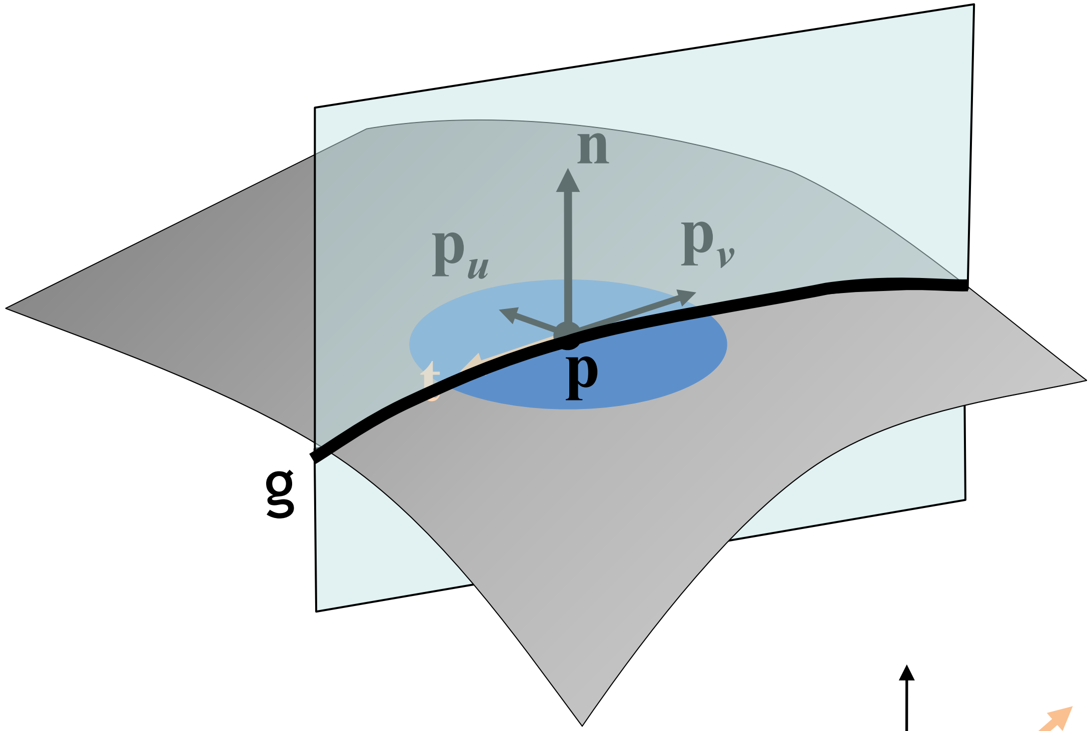
$$\mathbf{n}(u,v) = \frac{\mathbf{p}_u \times \mathbf{p}_v}{\|\mathbf{p}_u \times \mathbf{p}_v\|}$$

Unit-length direction **t** in the tangent plane (if $\mathbf{p}_u$ and $\mathbf{p}_v$ are orthogonal):

$$\mathbf{t} = \cos\varphi \frac{\mathbf{p}_u}{\|\mathbf{p}_u\|} + \sin\varphi \frac{\mathbf{p}_v}{\|\mathbf{p}_v\|}$$



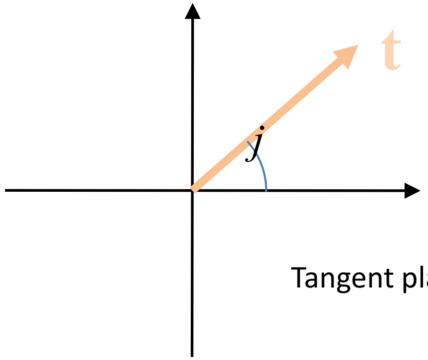
Normal Curvature



The curve γ is the intersection of the surface with the plane through **n** and **t** - a normal section.

Normal curvature:

$$\kappa_n(\varphi) = \kappa(\gamma(\mathbf{p}))$$



Surface Curvatures

- **Principal curvatures**
  - Minimal curvature
  - Maximal curvature

$$\kappa_1 = \kappa_{\min} = \min_{\varphi} \kappa_n(\varphi)$$

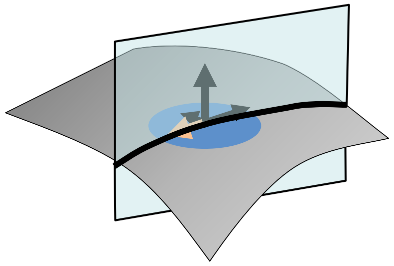
$$\kappa_2 = \kappa_{\max} = \max_{\varphi} \kappa_n(\varphi)$$

- **Mean curvature**

$$H = \frac{\kappa_1 + \kappa_2}{2} = \frac{1}{2\pi} \int_0^{2\pi} \kappa_n(\varphi) d\varphi$$

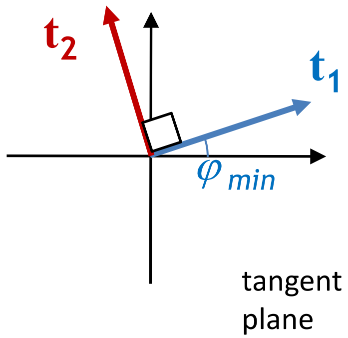
- **Gaussian curvature**

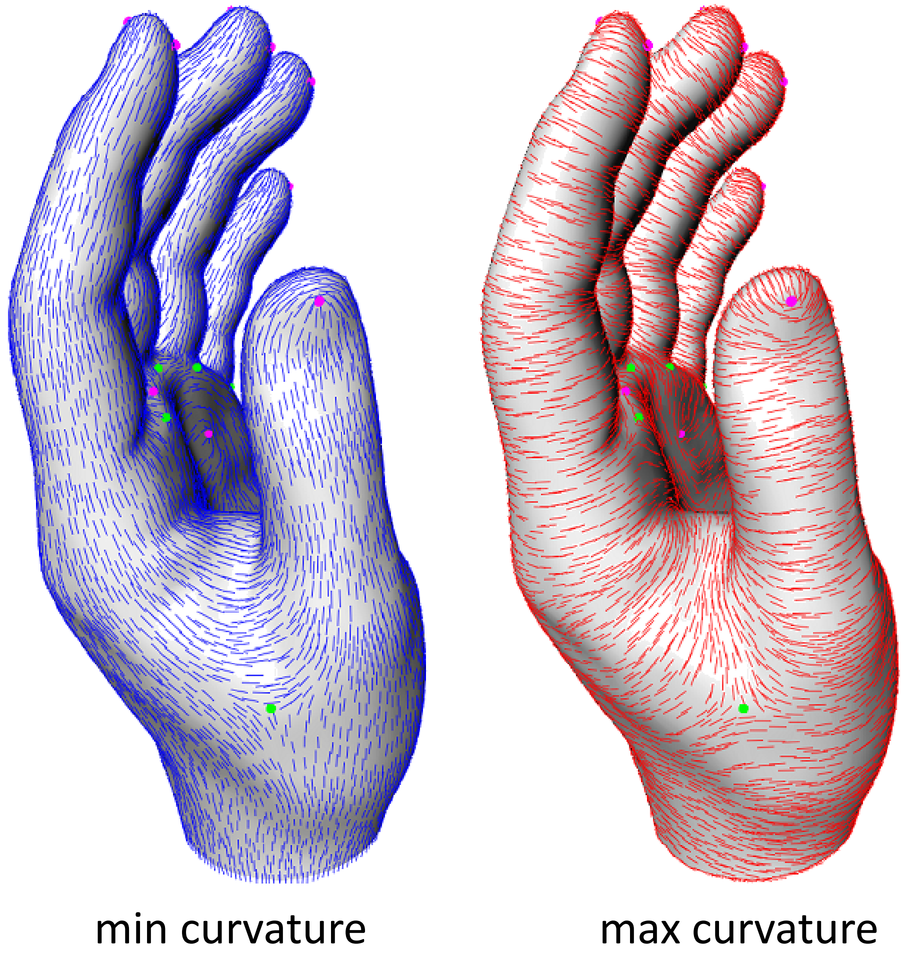
$$K = \kappa_1 \cdot \kappa_2$$



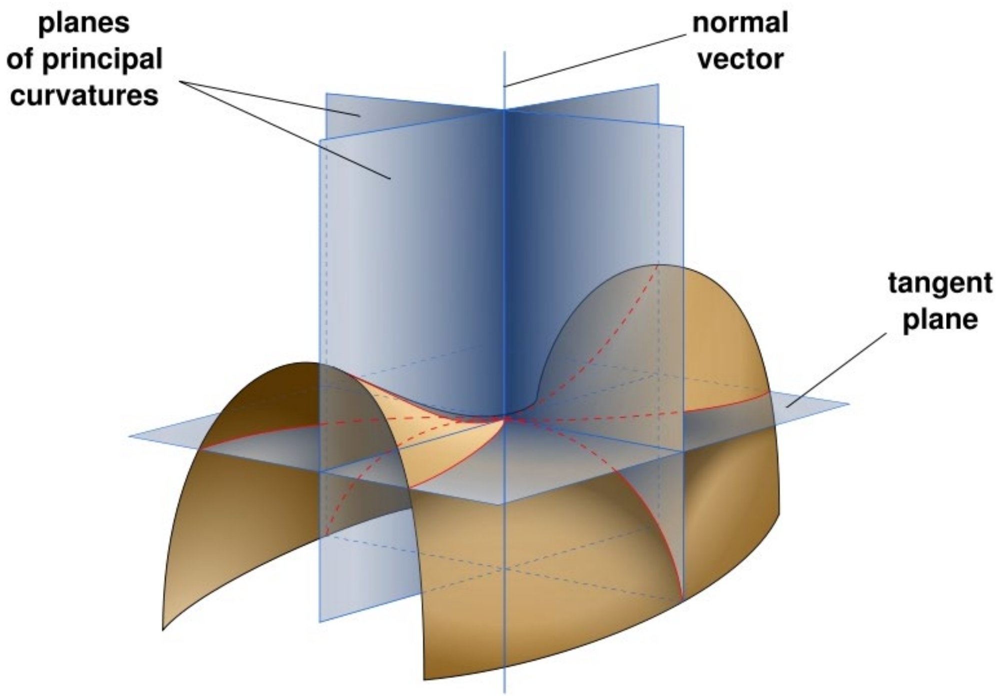
Principal Directions

- Principal directions: tangent vectors corresponding to φ_{max} and φ_{min}





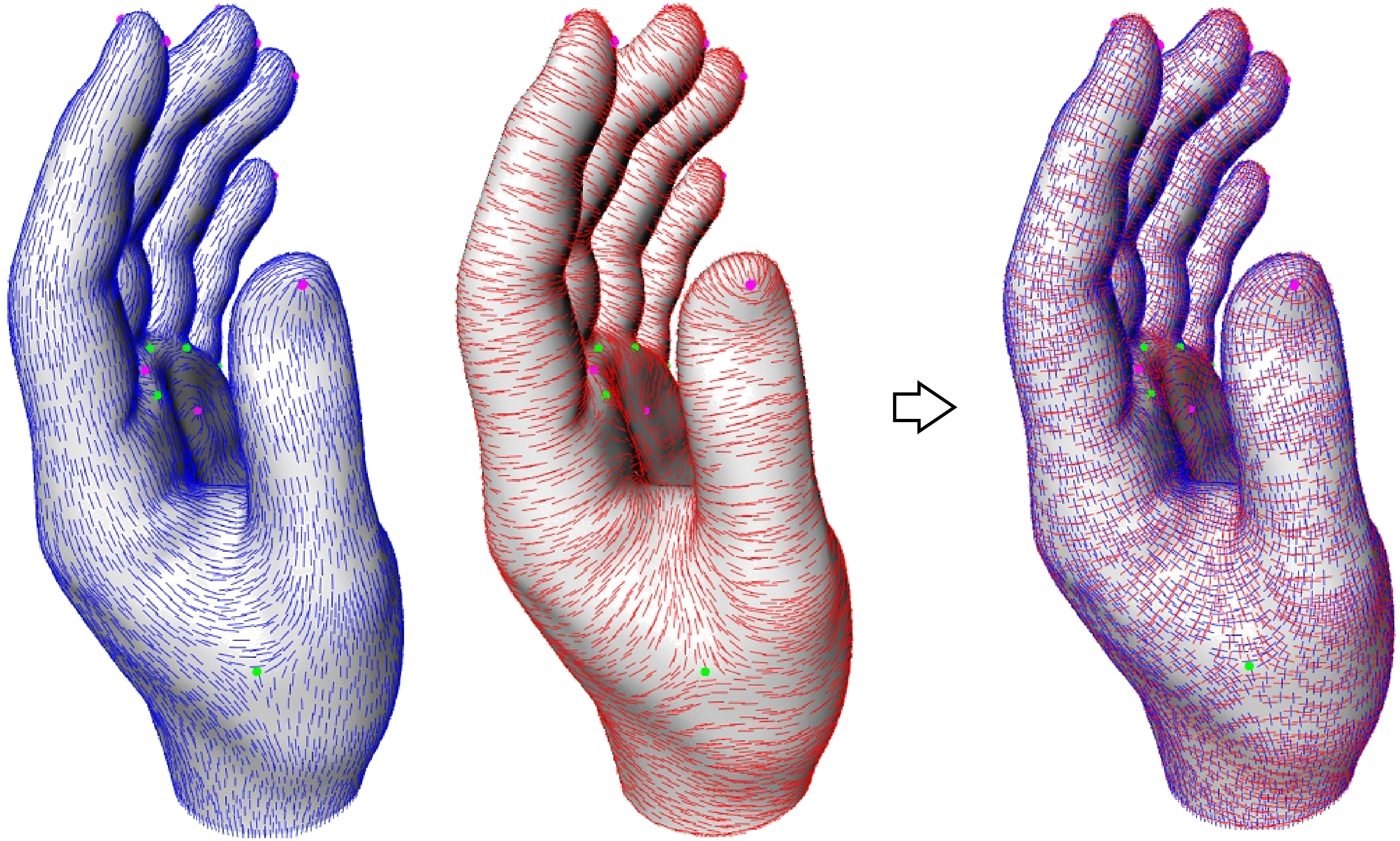
Principal Directions



Euler's Theorem: Planes of principal curvature are orthogonal and independent of parameterization.

$$\kappa_n(\varphi) = \kappa_1 \cos^2 \varphi + \kappa_2 \sin^2 \varphi, \quad \varphi = \text{angle with } \mathbf{t}_1$$

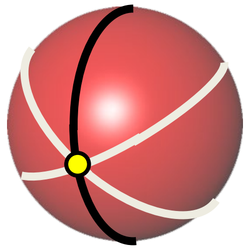
Principal Directions



Local Surface Shape By Curvatures

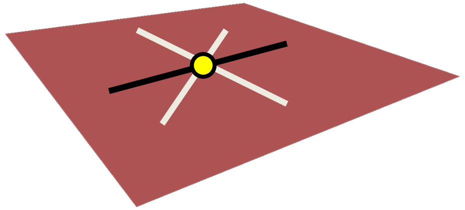
Isotropic:
all directions are principal directions

$K > 0,\ \kappa_1 = \kappa_2$



spherical (umbilical)

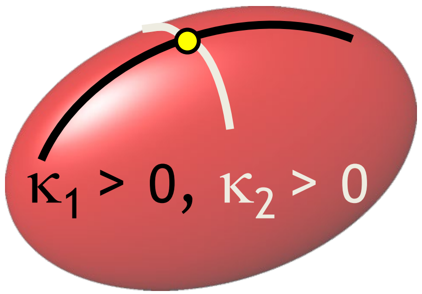
$K = 0$



planar

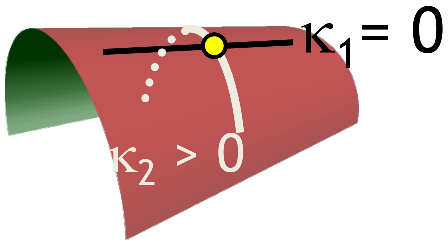
Anisotropic:
2 distinct principal directions

$K > 0$



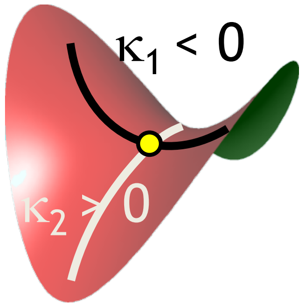
elliptic

$K = 0$



parabolic

$K < 0$



hyperbolic

That’s All