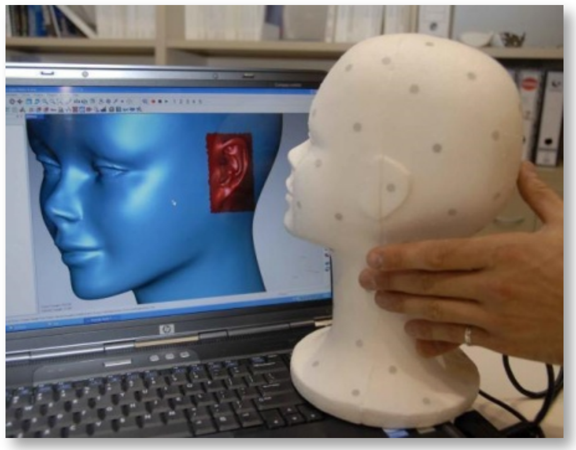
Shape Representations

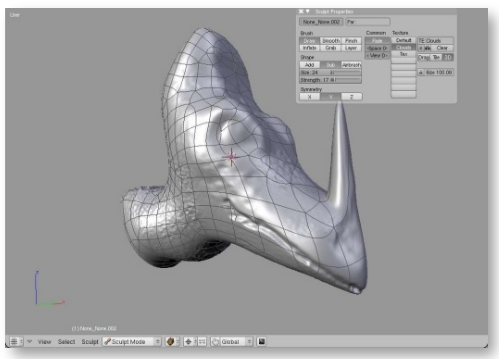
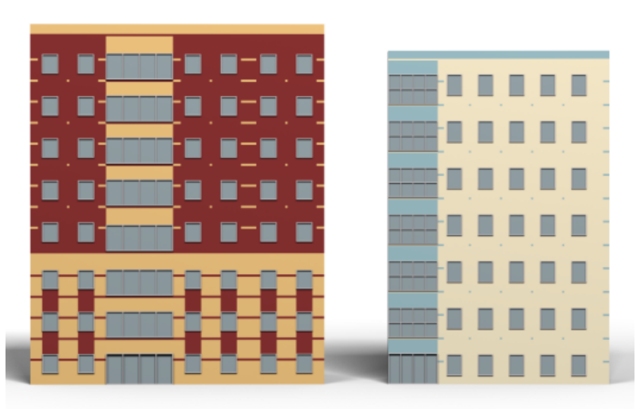
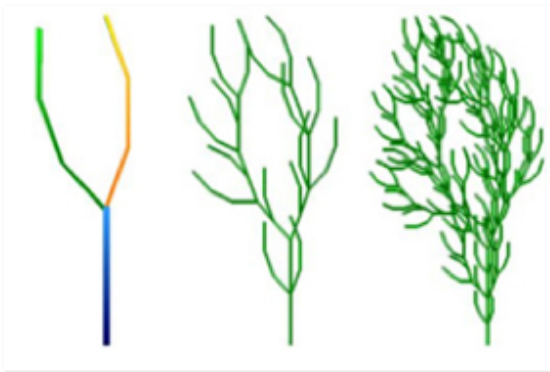
Instructor: Hao Su

Slides credits: Olga Diamanti, Olga Sorkine-Hornung, Daniele Panozzo, ETH Zurich, Maks Ovsjanikov, Mario Botsch

Shape Representation: Origin- and Application-Dependent

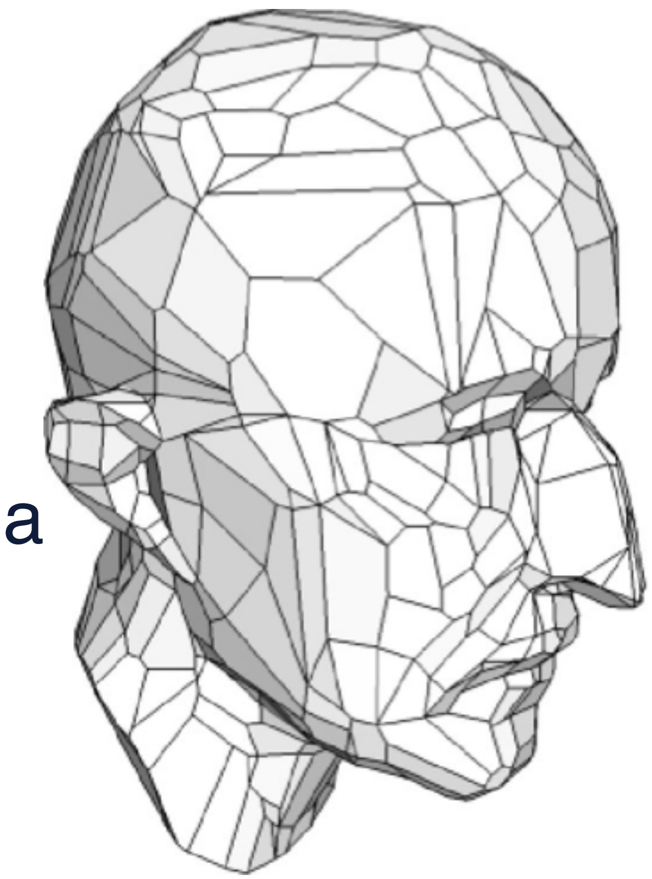
- Acquired real-world objects:
- Modeling "by hand":
- Procedural modeling
- …

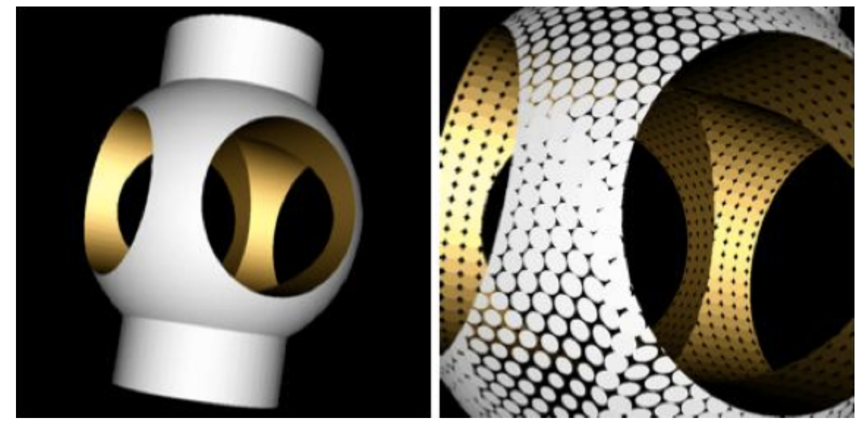
Representation Considerations

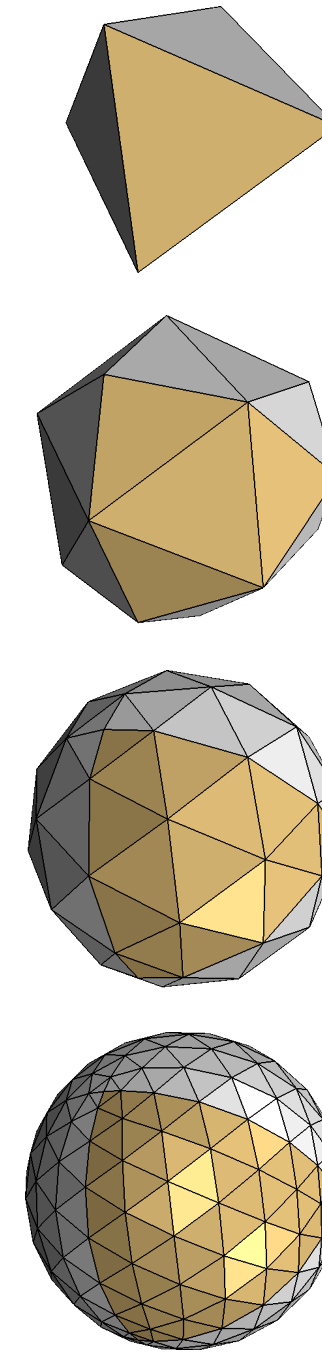
- How should we represent geometry?
  - Needs to be stored in the computer
  - Creation of new shapes
    - Input metaphors, interfaces…
  - What operations do we apply?
    - Editing, simplification, smoothing, filtering, repa
  - How to render it?
    - Rasterization, raytracing…

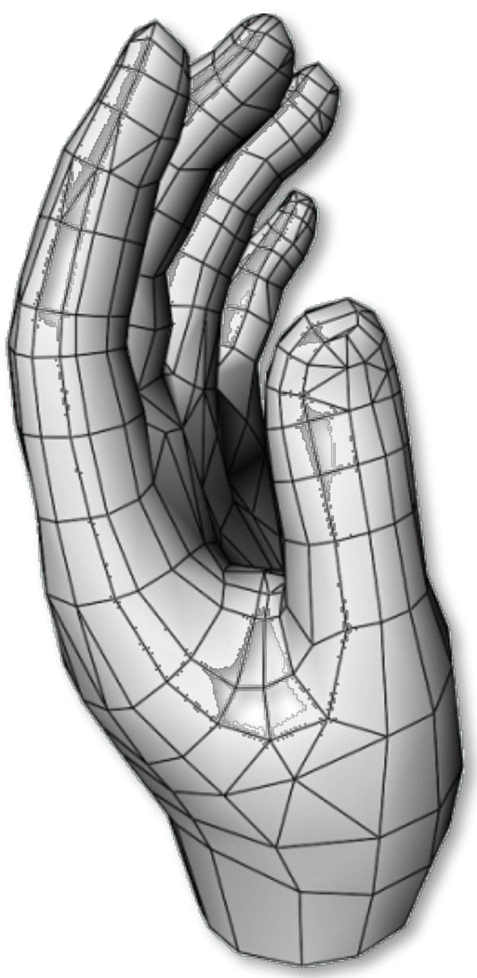


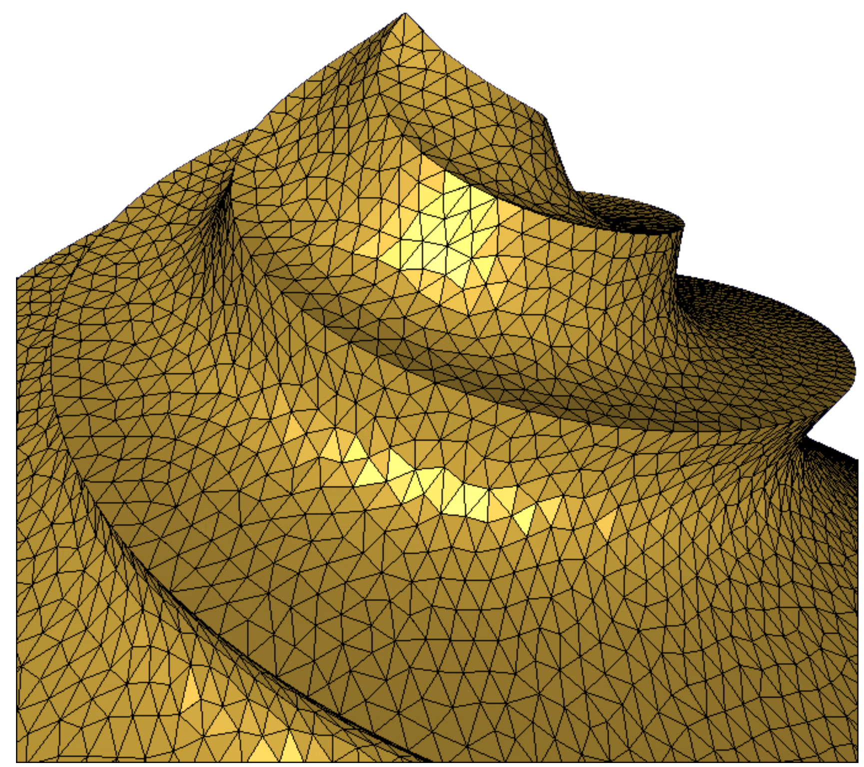
Shape Representations

- Points
- Polygonal meshes



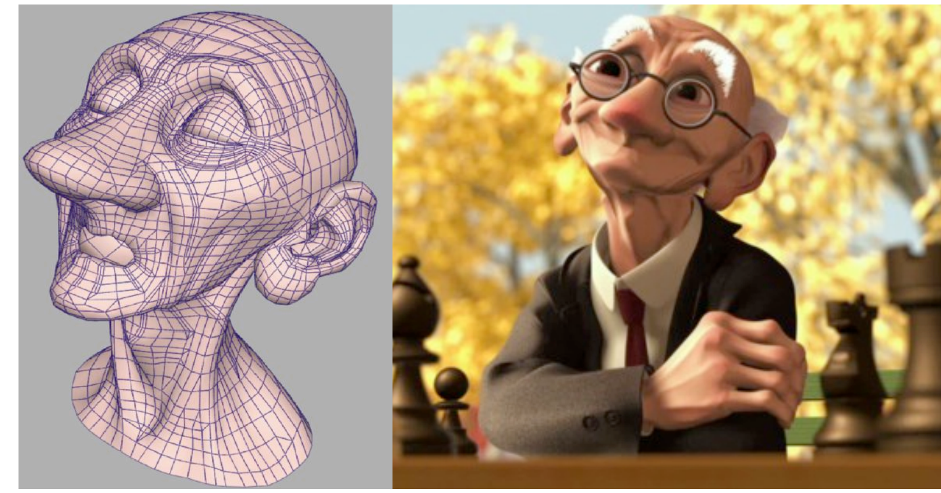


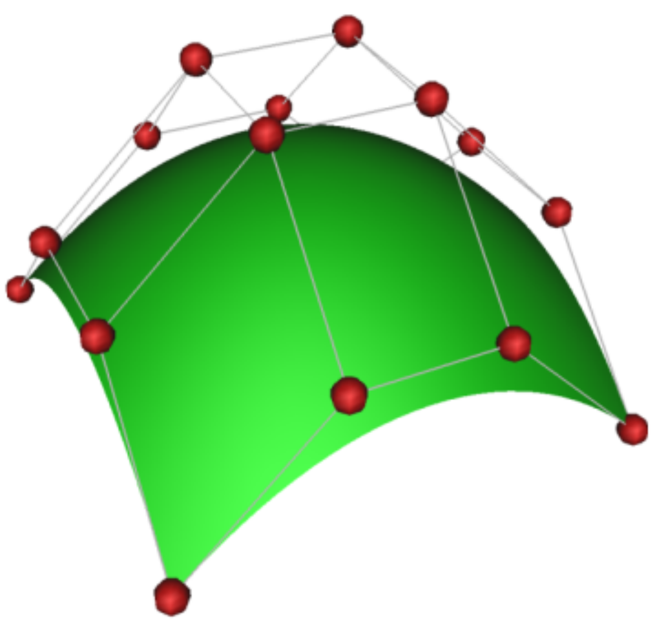


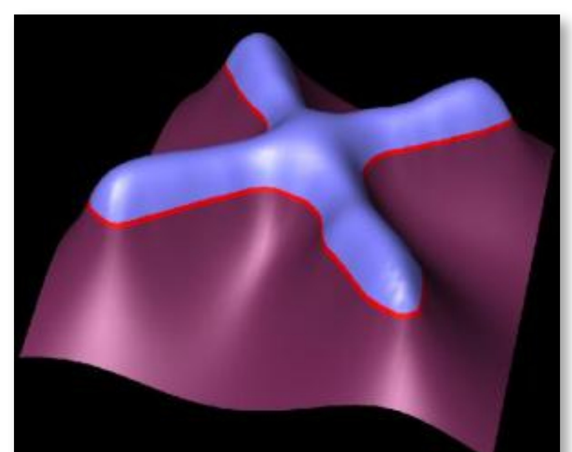


Shape Representations

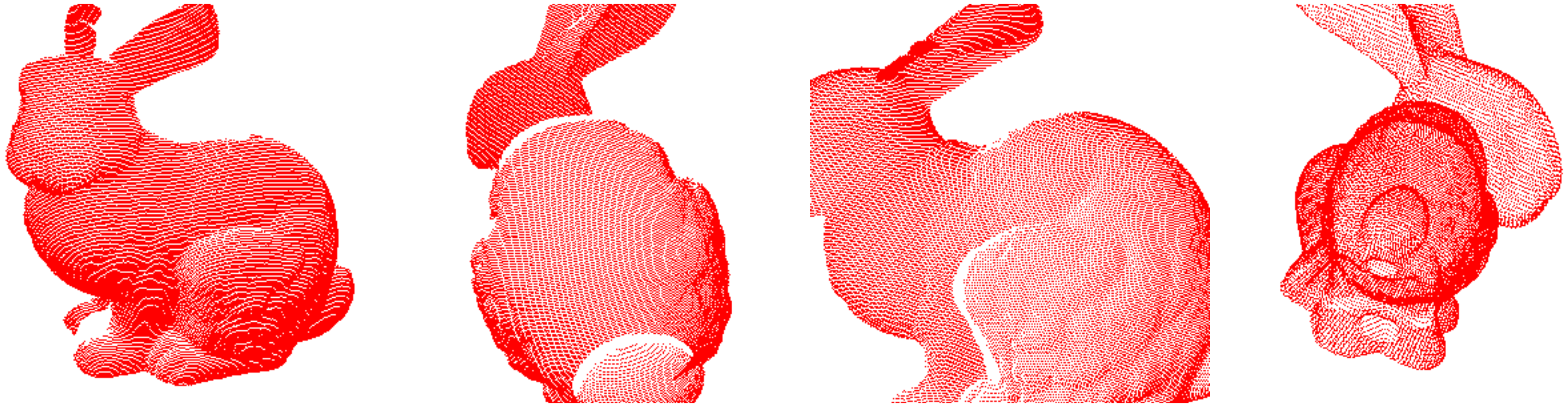
- Parametric surfaces
- Implicit functions
- Subdivision surfaces



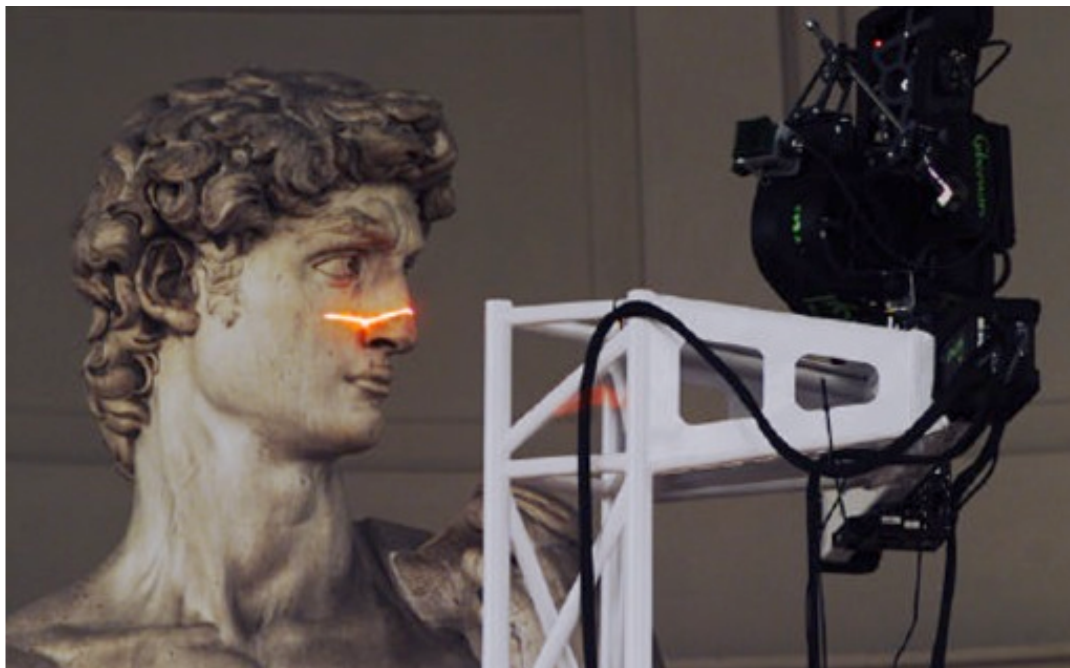


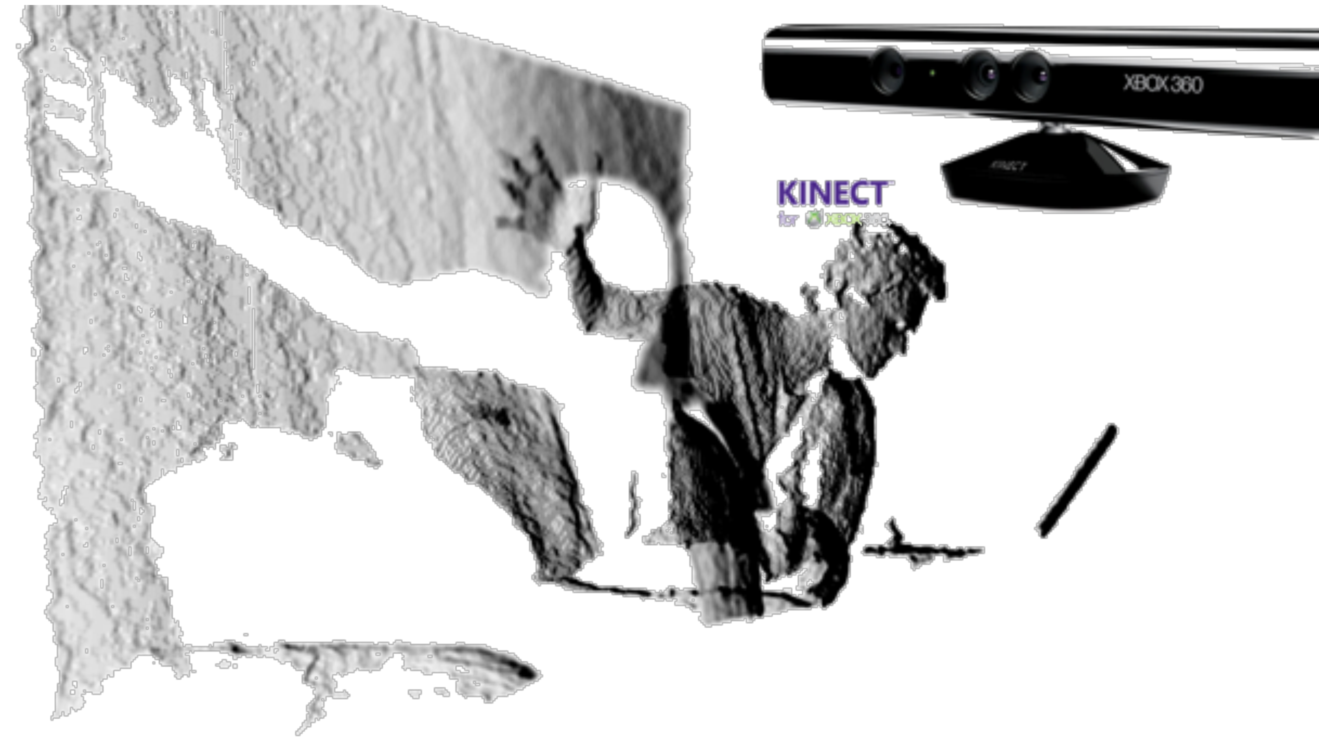


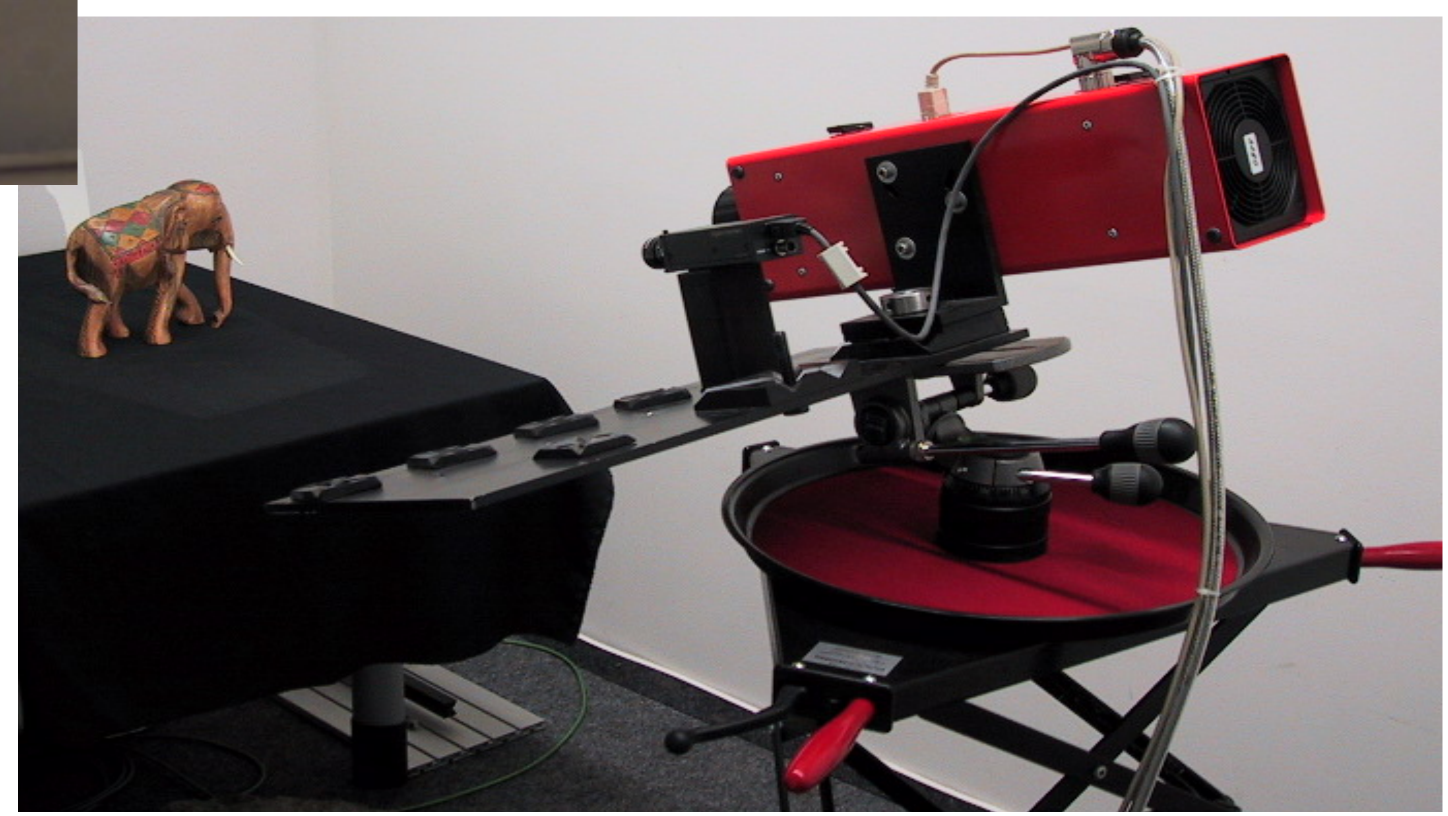
Points



Output of Acquisition

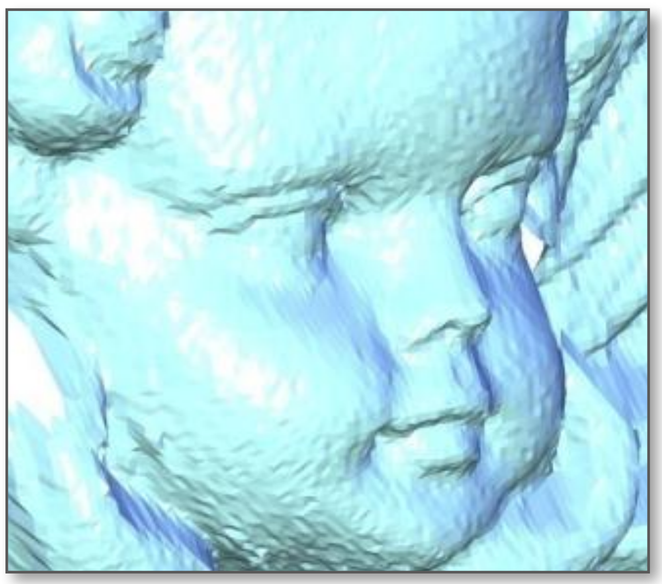


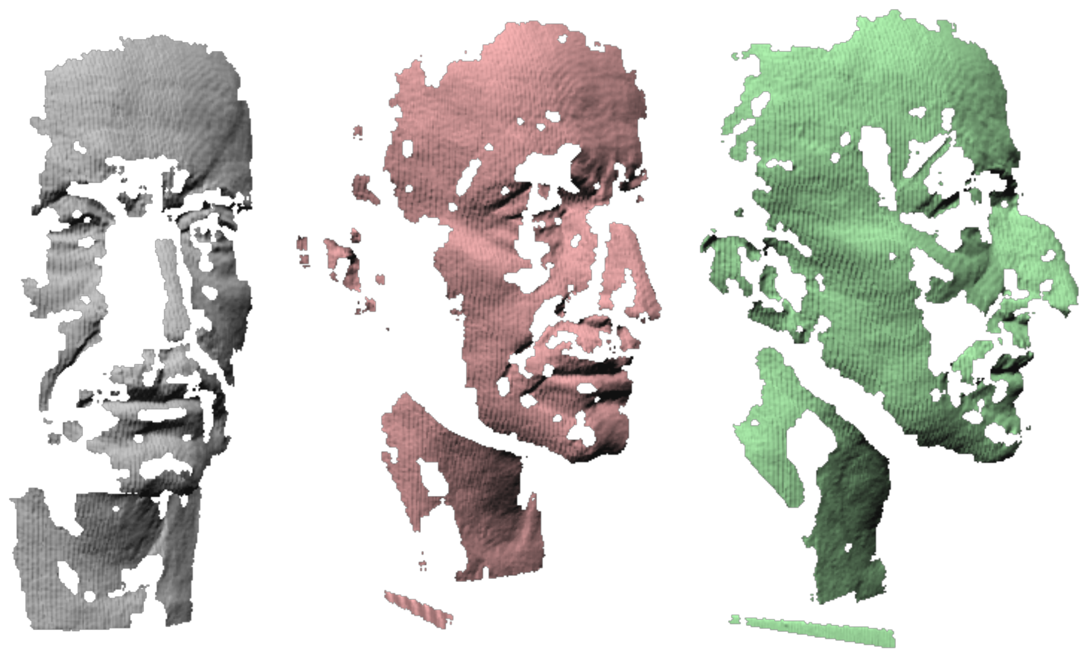




Points

- Standard 3D data from a variety of sources
  - Often results from scanners
  - Potentially noisy



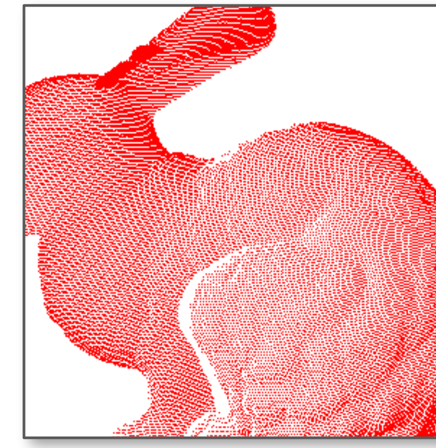


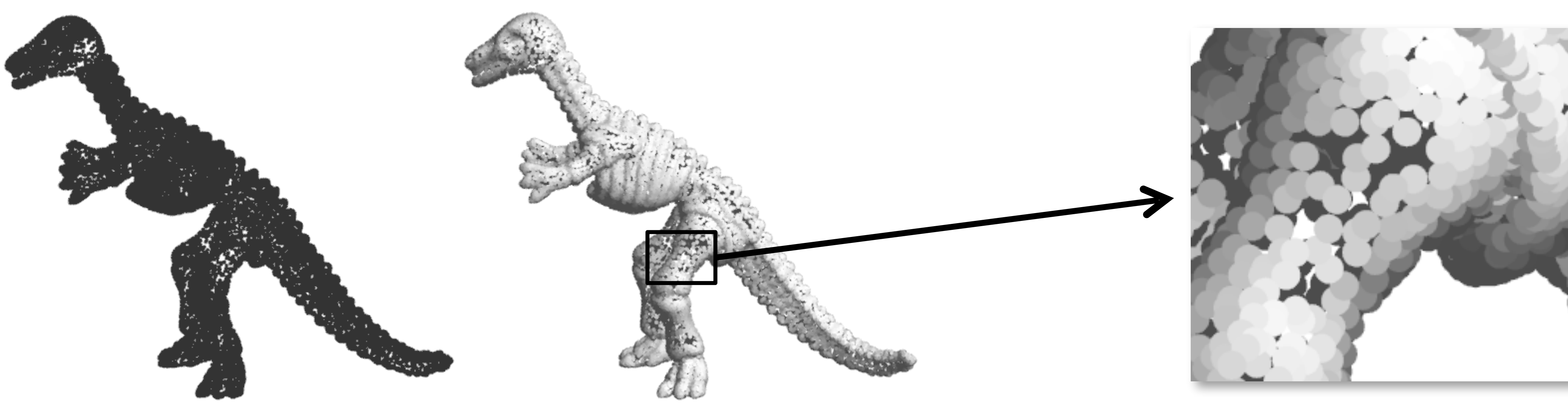
set of raw scans

  - Depth imaging (e.g. by triangulation)
  - Registration of multiple images

Points

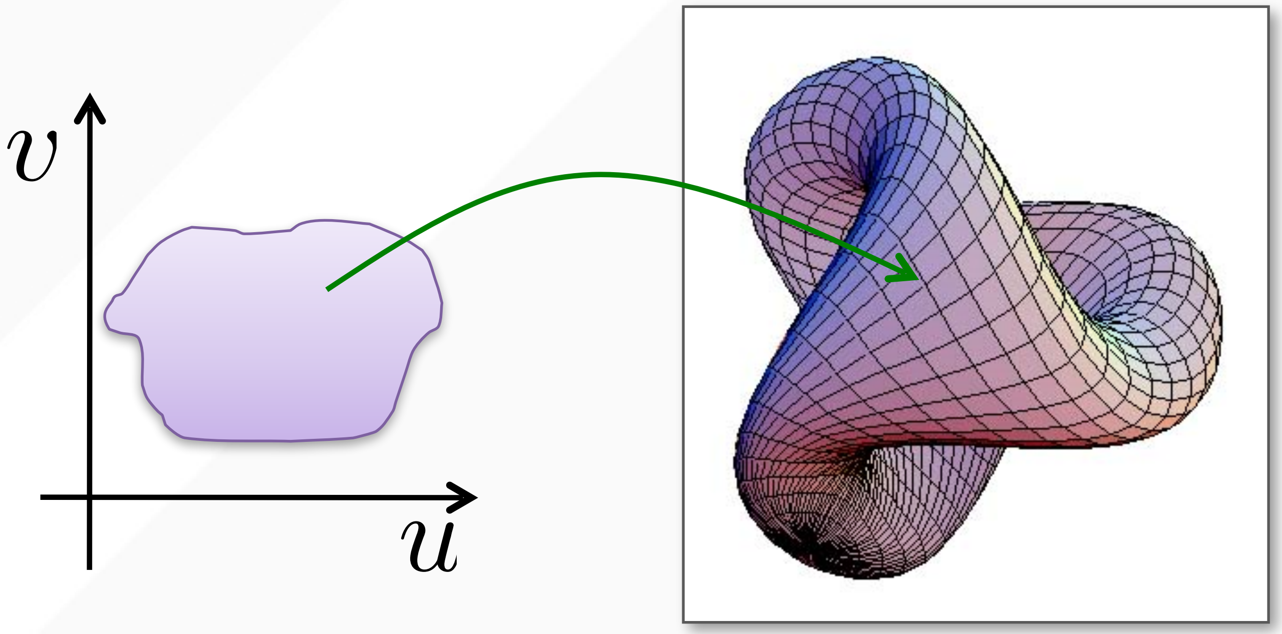
- Points = unordered set of 3-tuples
- Often converted to other reps
  - Meshes, implicits, parametric surfaces
  - Easier to process, edit and/or render
- Efficient point processing / modeling requires spatial partitioning data structure
  - Eg. **to figure out neighborhoods**





shading needs normals!

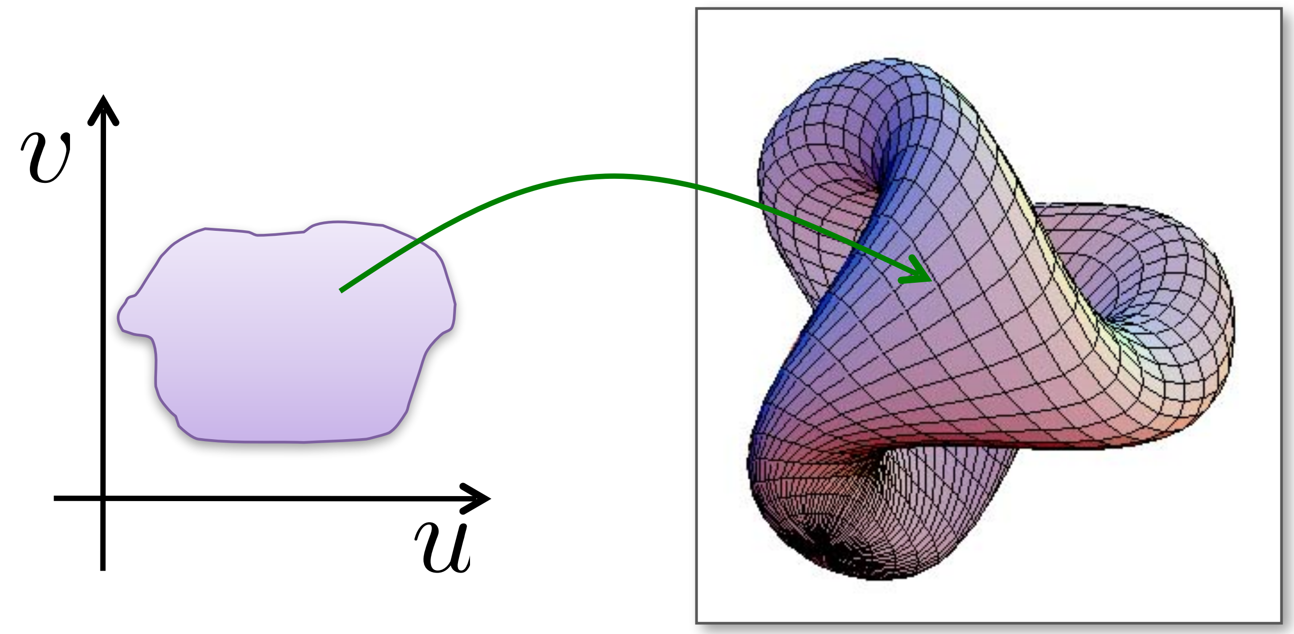
PARAMETRIC CURVES AND SURFACES



UC San Diego

Parametric Representation

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



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

Parametric Curves

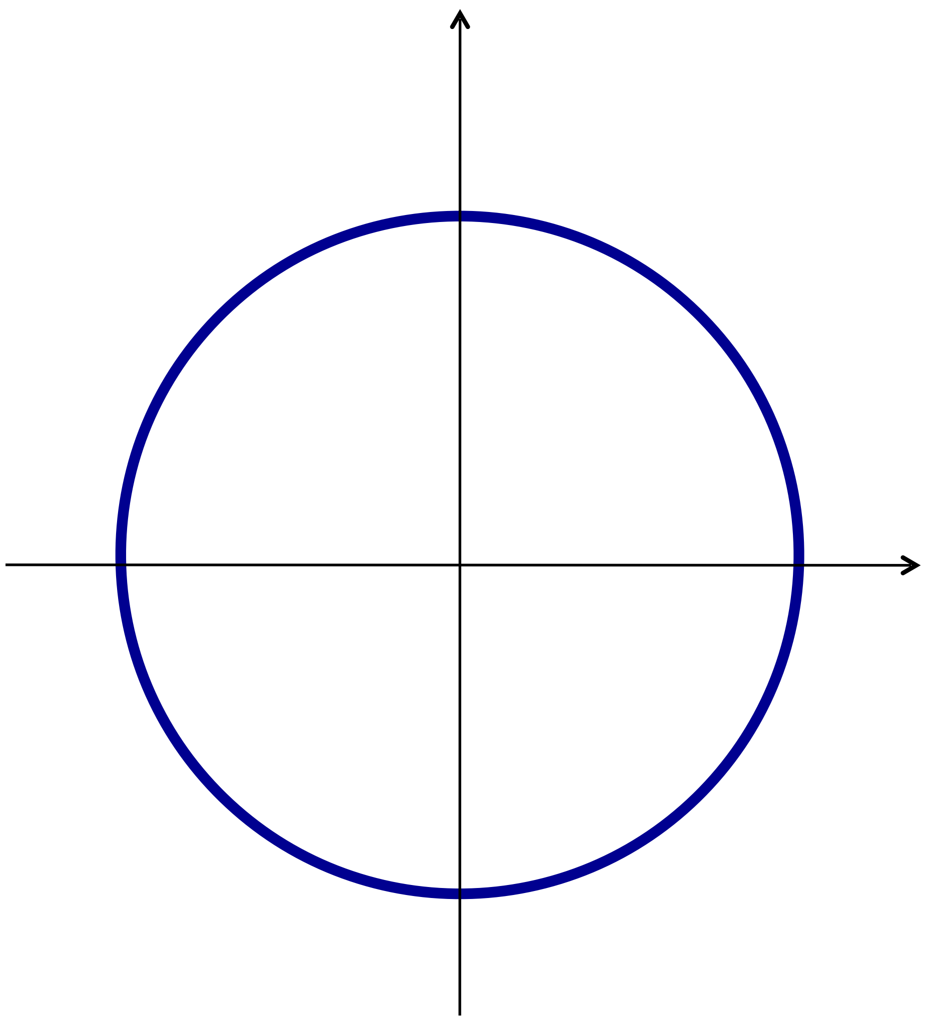
- Example: Explicit curve/circle in 2D

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

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

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

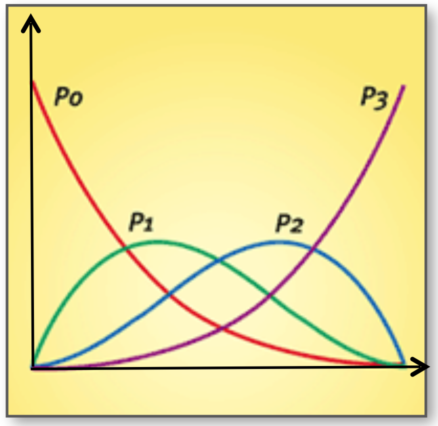
$$t \in [0, 2\pi)$$



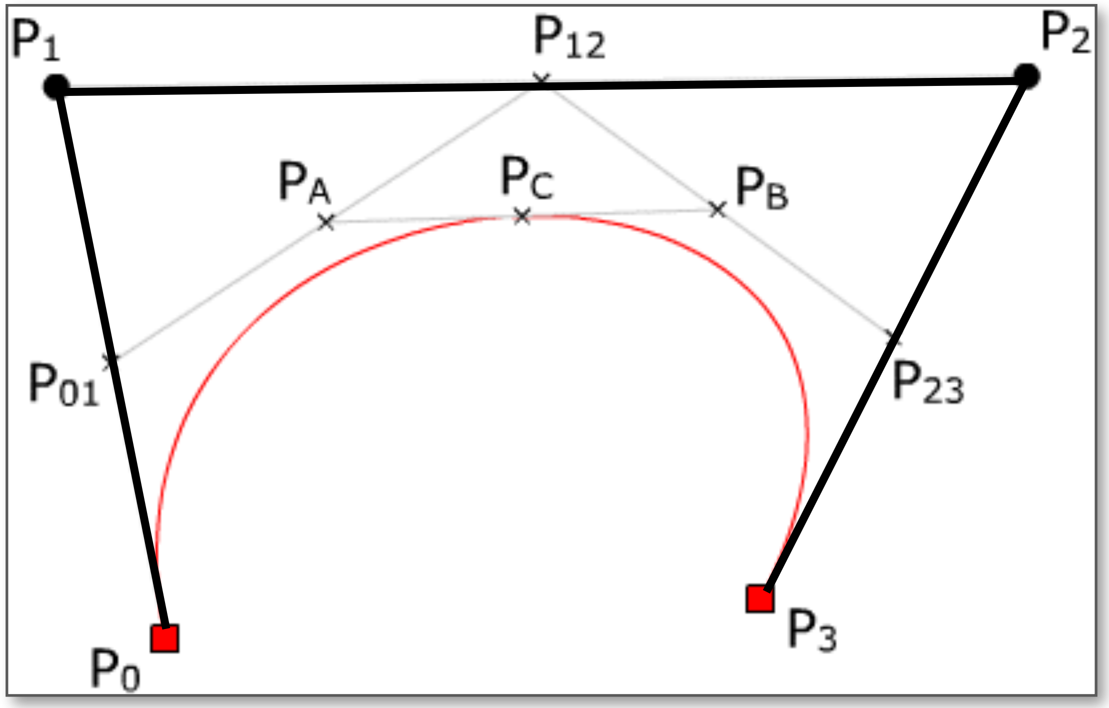
Parametric Curves

- Bezier curves, splines

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



Basis functions

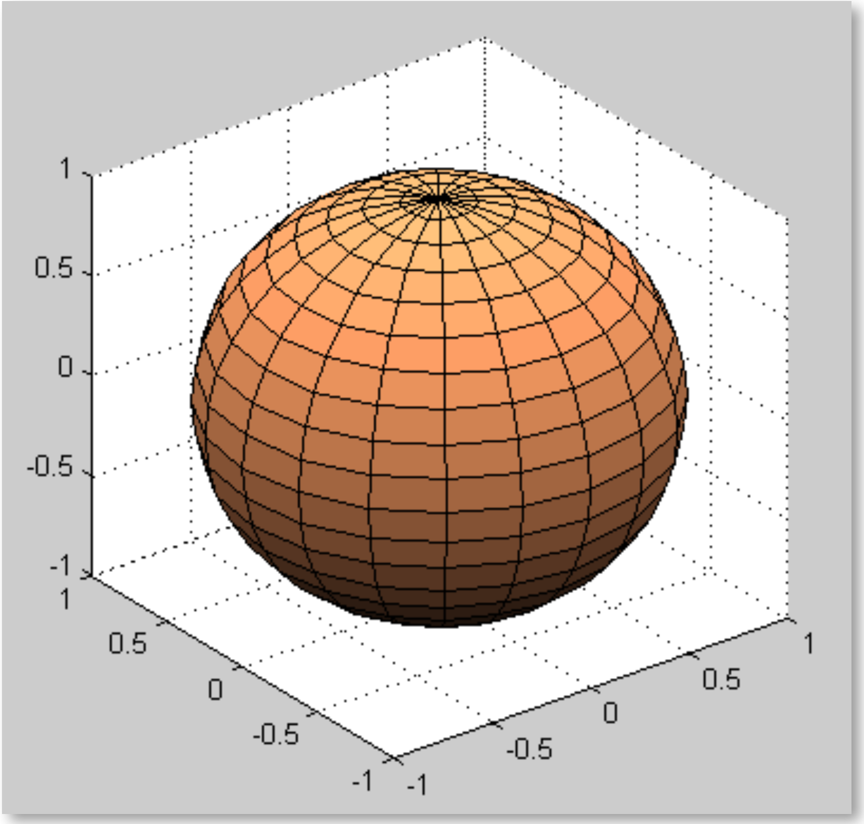


Curve and control polygon

Parametric Surfaces

- Sphere in 3D

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



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

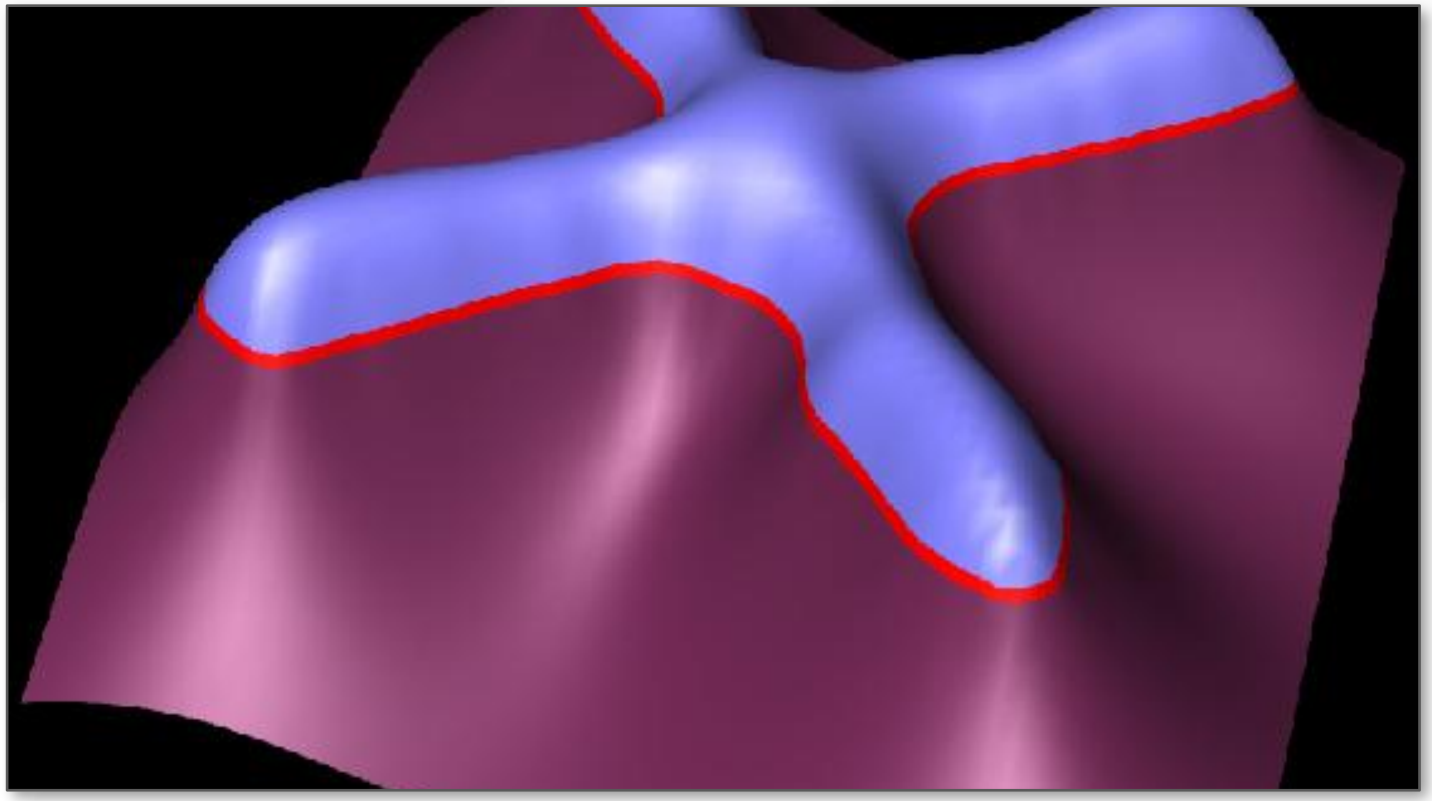
$$(u, v) \in [0, 2\pi) \times [-\pi/2, \pi/2]$$

Parametric Curves and Surfaces

- Advantages
  - Easy to generate points on the curve/surface
  - Separates x/y/z components

- Disadvantages
  - Hard to determine inside/outside
  - Hard to determine if a point is on the curve/surface
  - Hard to express more complex curves/surfaces!
    ➜cue: piecewise parametric surfaces (eg. mesh)

IMPLICIT CURVES AND SURFACES



Implicit Curves and Surfaces

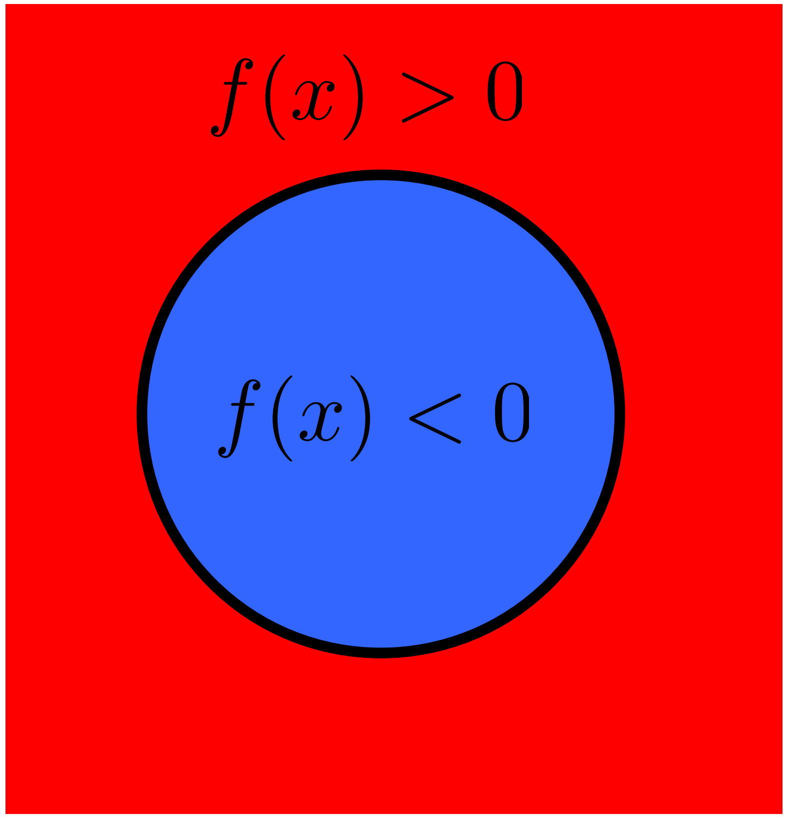
- Kernel of a scalar function $f : \mathbb{R}^m \to \mathbb{R}$
  - Curve in 2D: $S = \{x \in \mathbb{R}^2 | f(x) = 0\}$
  - Surface in 3D: $S = \{x \in \mathbb{R}^3 | f(x) = 0\}$

- Space partitioning

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

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

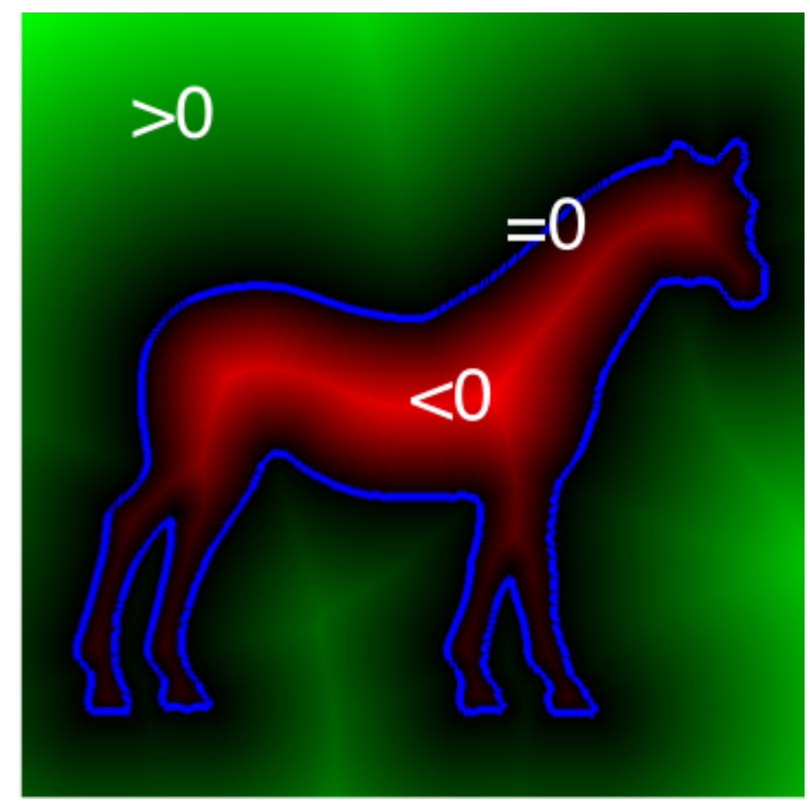
$\{x \in \mathbb{R}^m | f(x) < 0\}$ Inside



Implicit Curves and Surfaces

- 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: $S = \{x \in \mathbb{R}^3 | f(x) = 0\}$

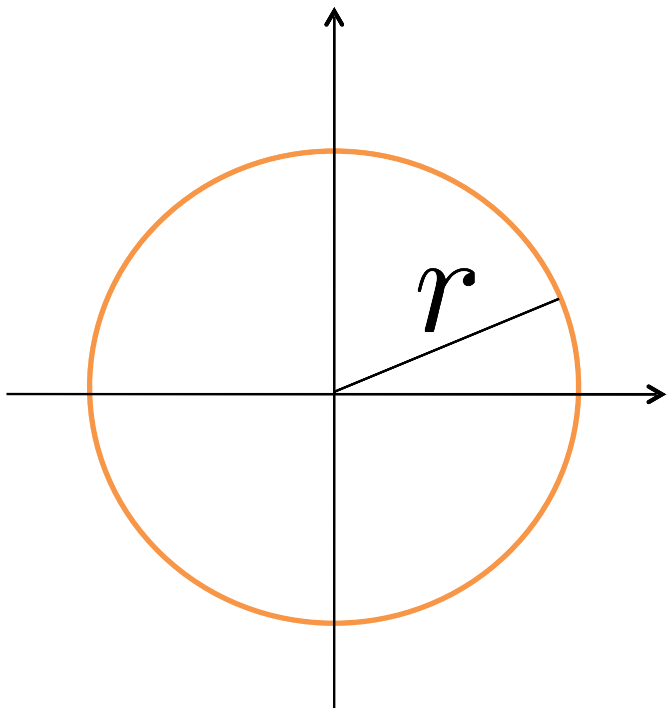
- Zero level set of signed distance function



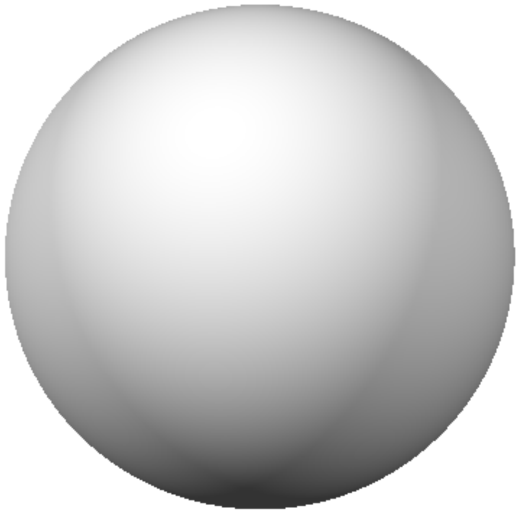
Implicit Curves and Surfaces

- Implicit circle and sphere

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

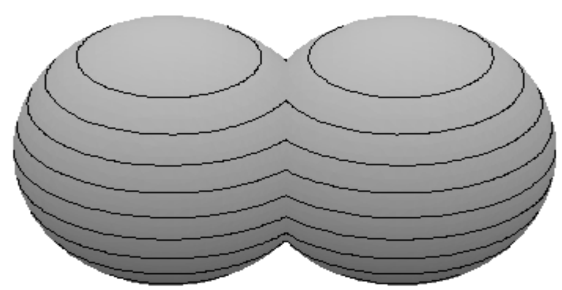


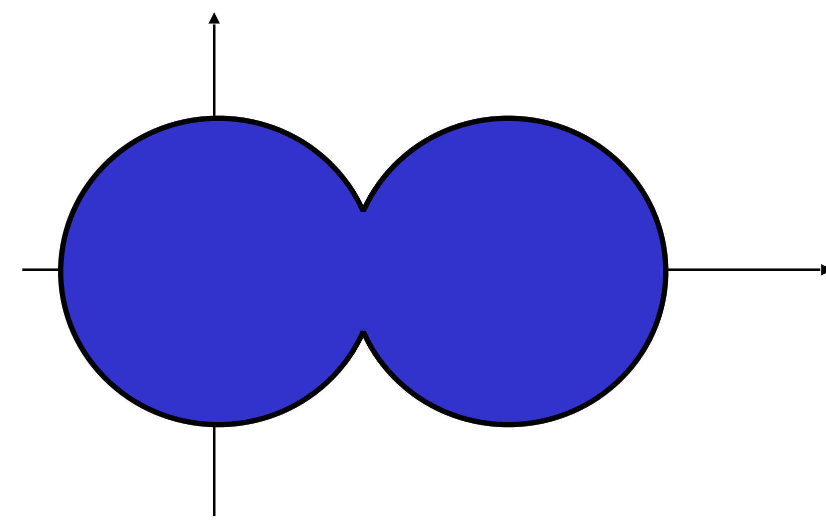
$$f(x, y, z) = x^2 + y^2 + z^2 - r^2$$



Boolean Set Operations

- Union: $$\bigcup_i f_i(x) = \min f_i(x)$$





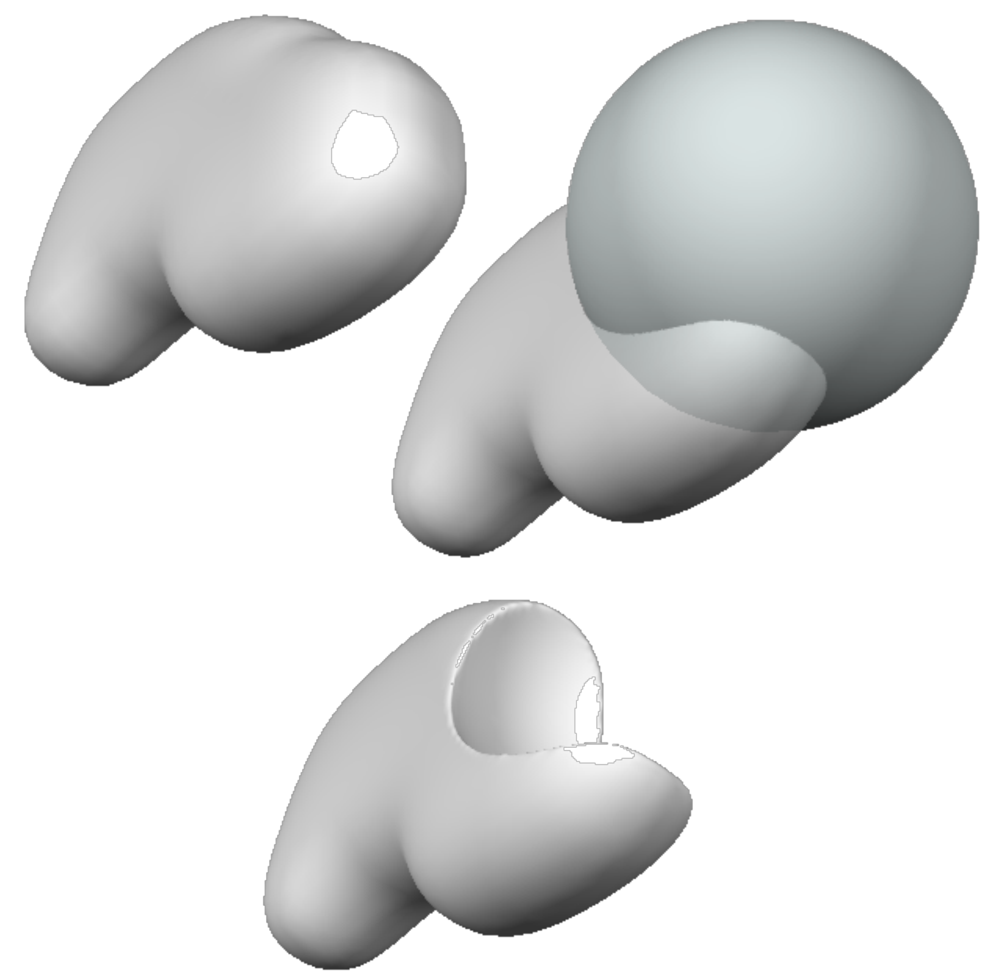
- Intersection: $$\bigcap_i f_i(x) = \max f_i(x)$$

Boolean Set Operations

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

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

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



- Much easier than for parametric surfaces!

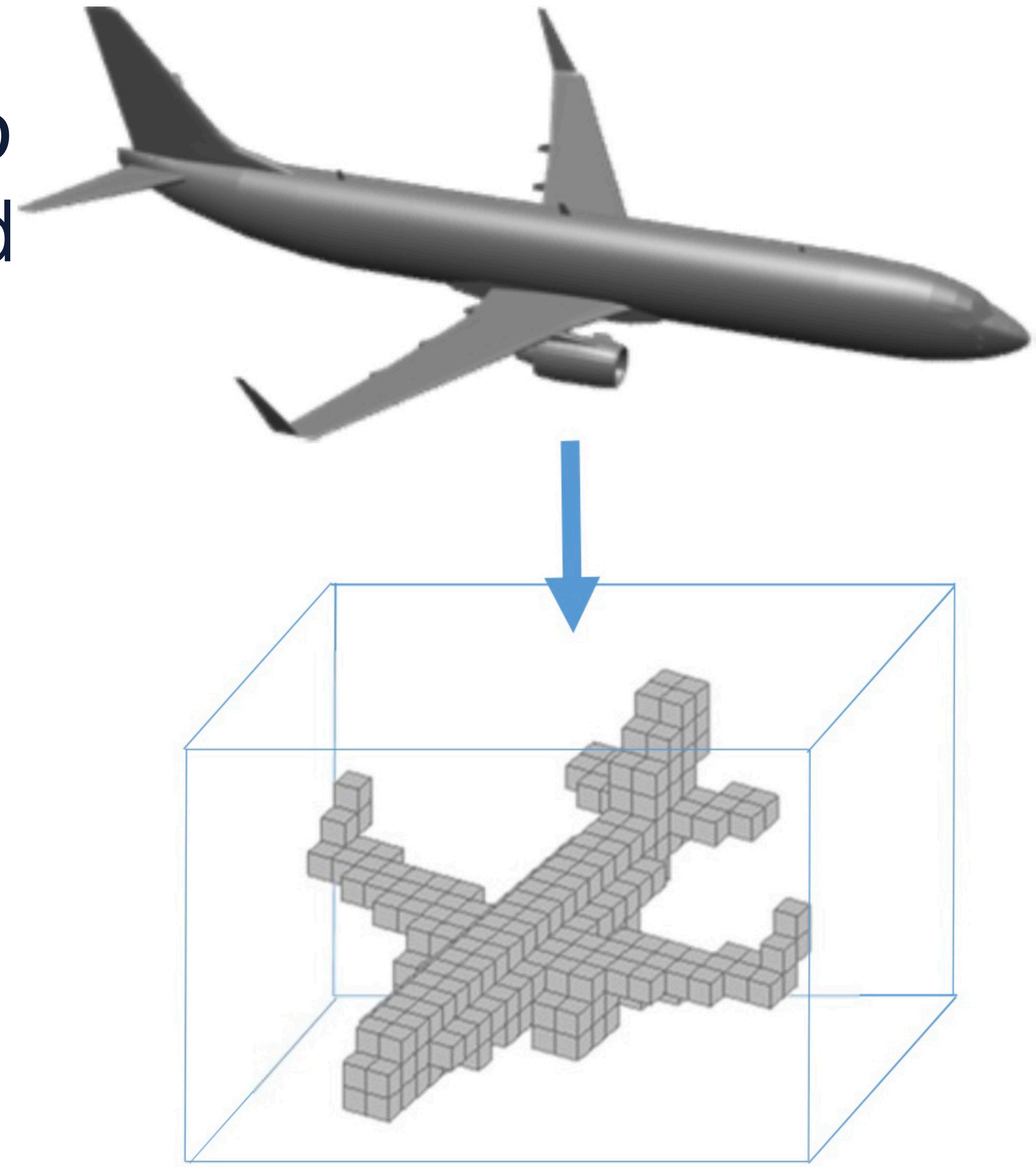
Implicit Curves and Surfaces

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

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

A related representation

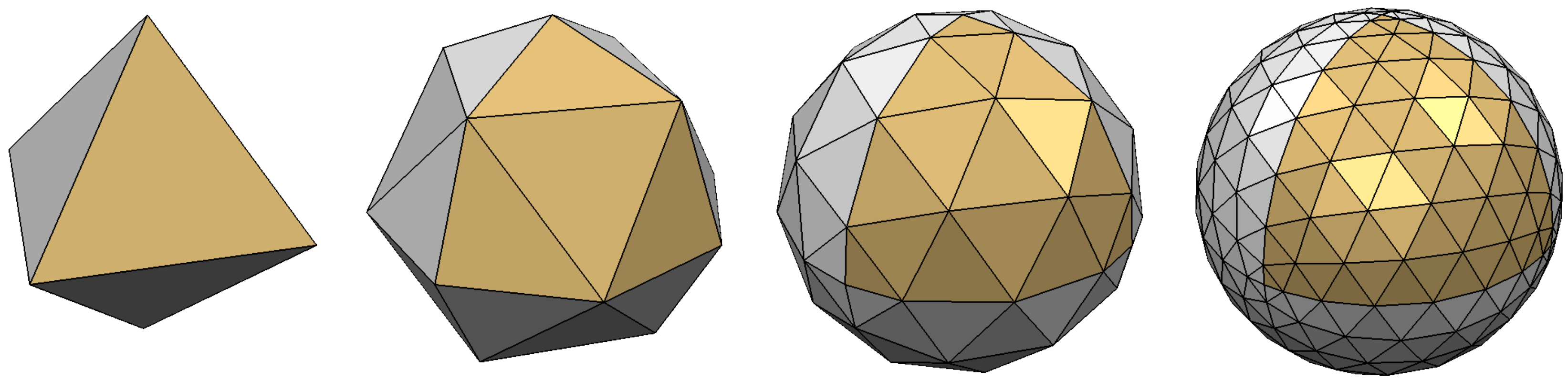
- Binary volumetric grids
- Can be produced by thresho
  or from the scanned points d

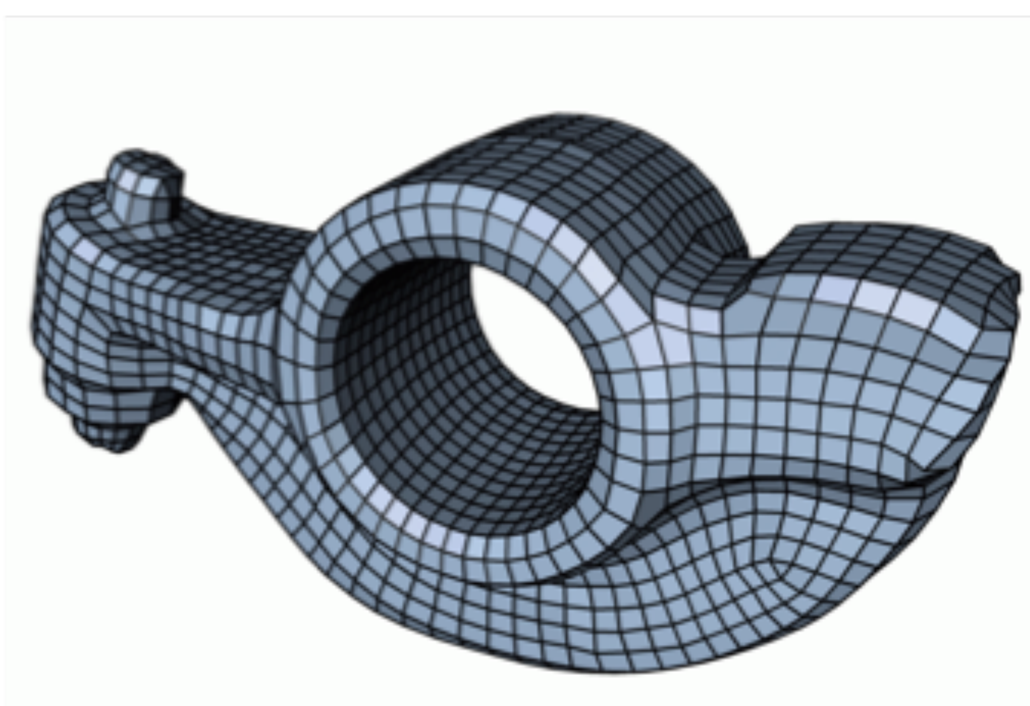


POLYGONAL MESHES

Polygonal Meshes

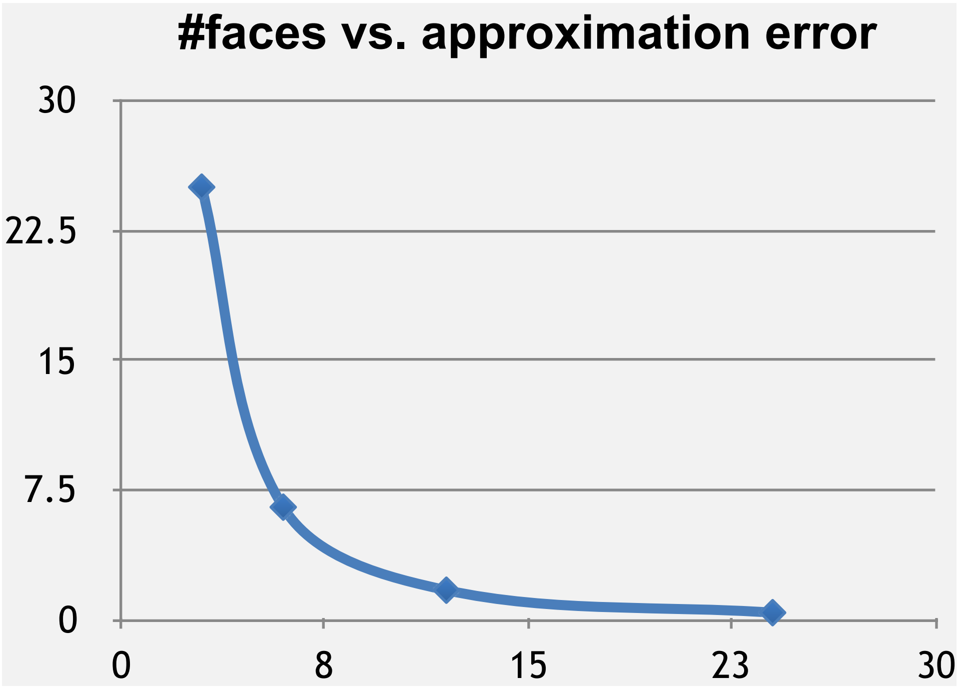
- Boundary representations of objects

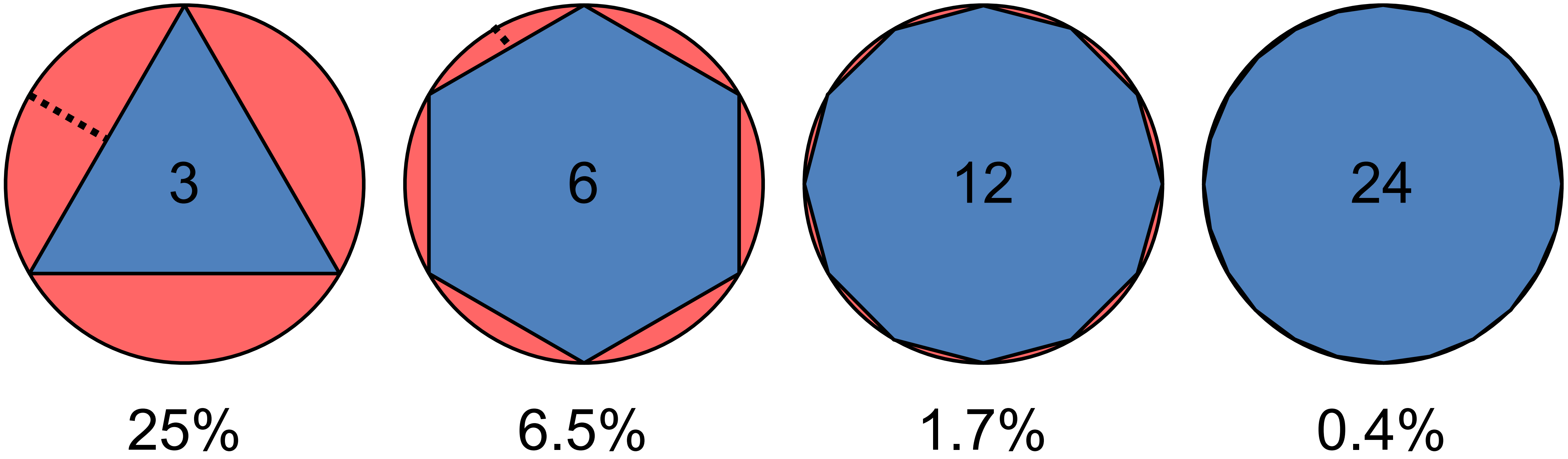




Meshes as Approximations of Smooth Surfaces

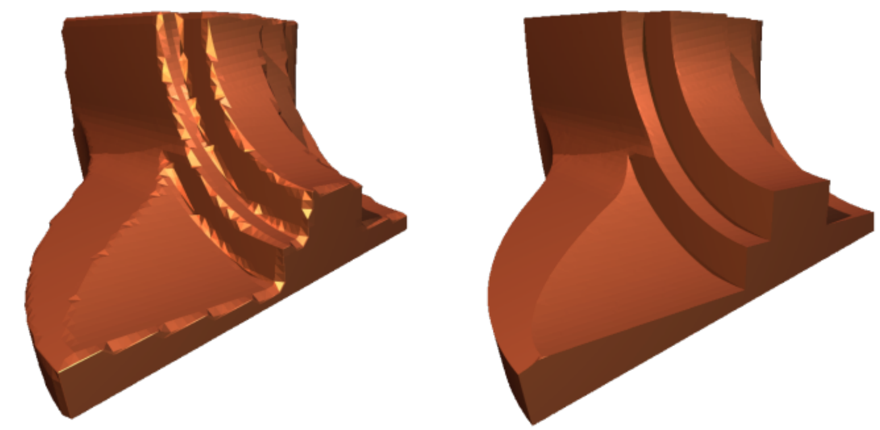
- Piecewise linear approximation
  - Error is $O(h^2)$

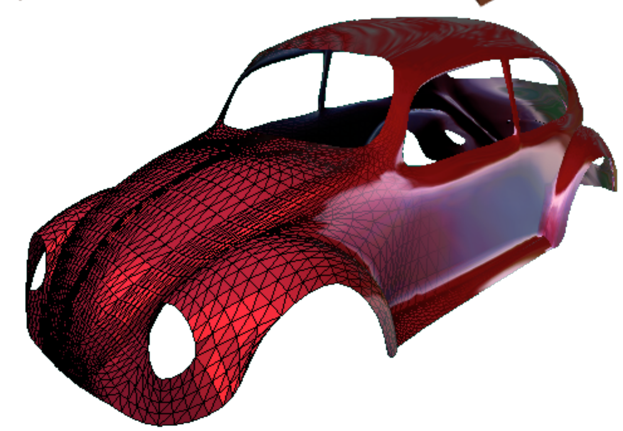


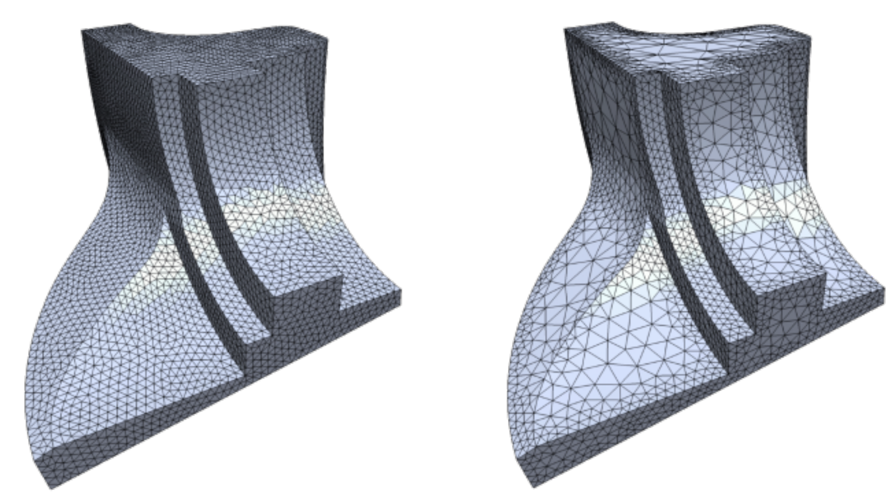


Polygonal Meshes

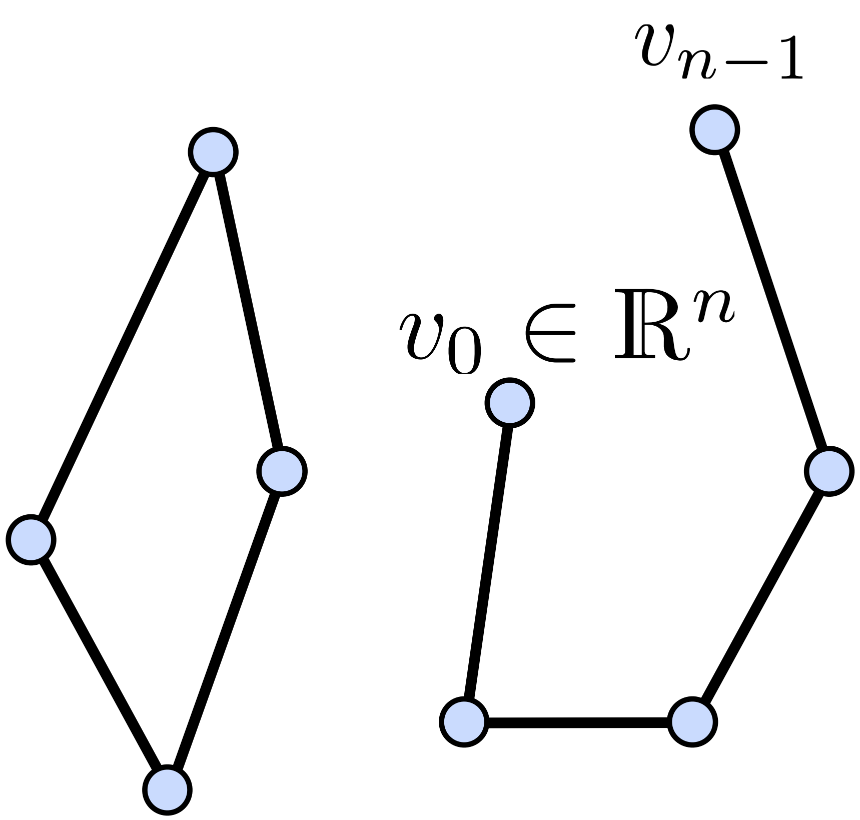
- Polygonal meshes are a good representation
  - approximation O(h^2)
  - arbitrary topology
  - adaptive refinement
  - efficient rendering







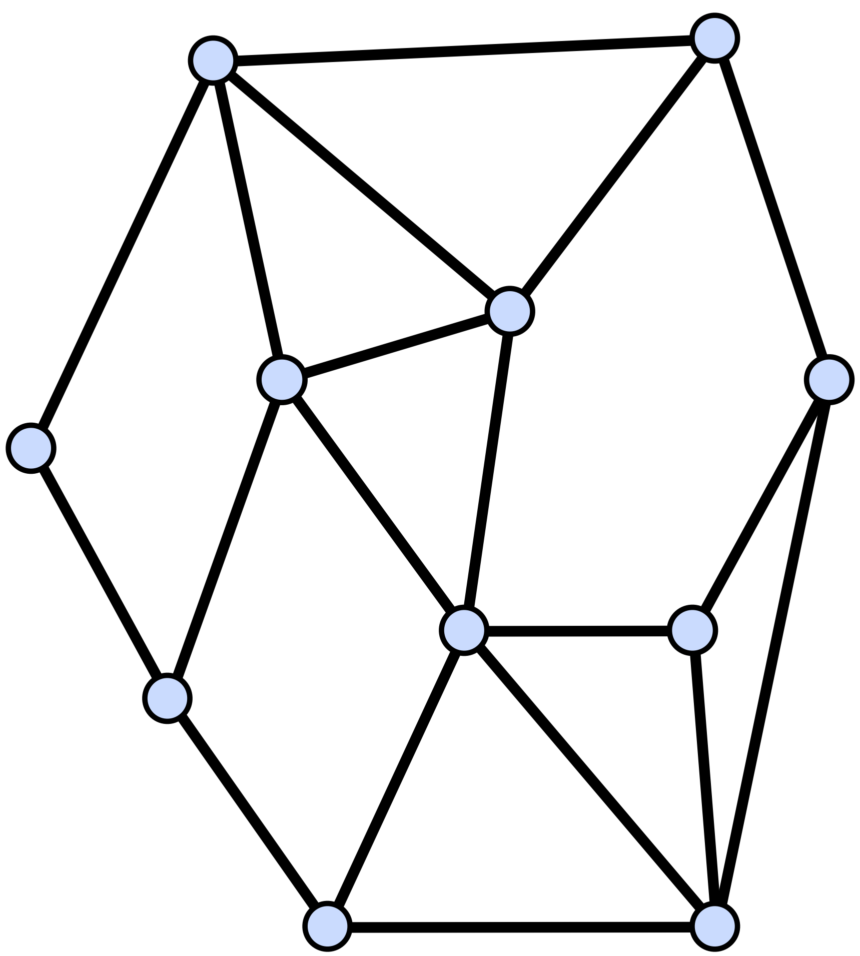
Polygon



- 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 Mesh

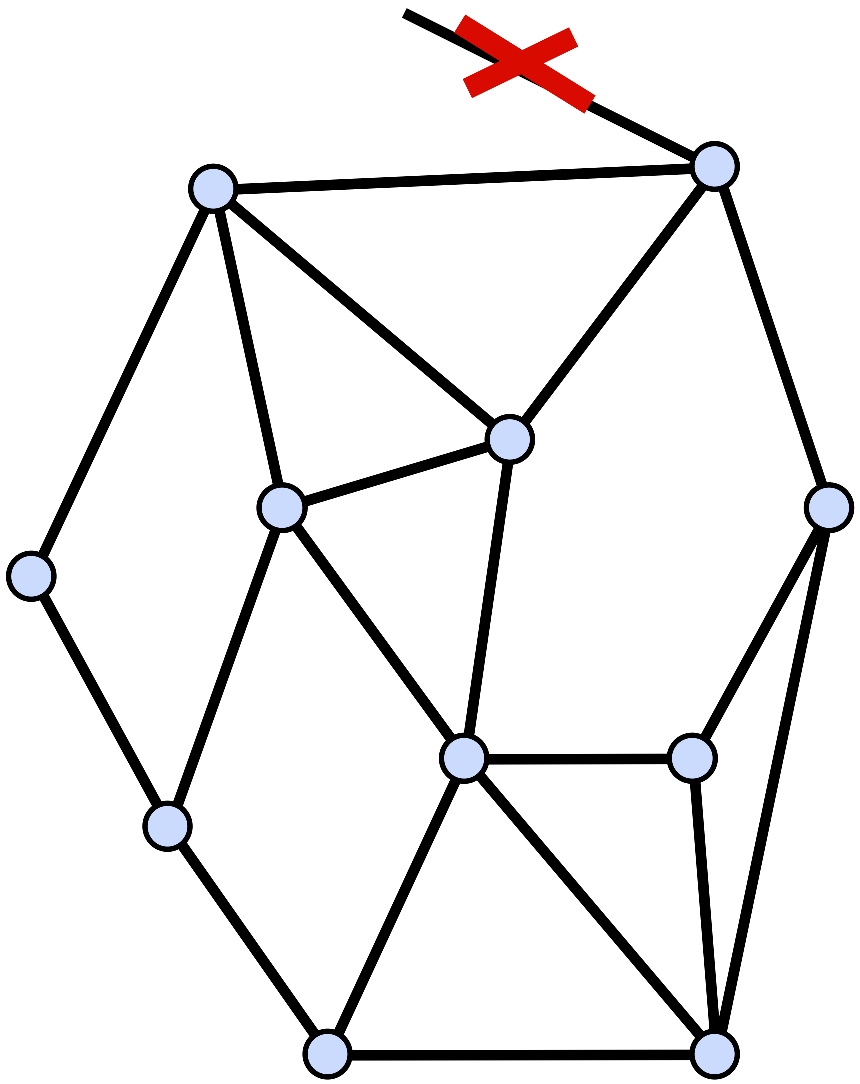


- A finite set M of closed, simple polygons Q_i is a polygonal mesh
- The intersection of two polygons in M is either empty, a vertex, or an edge

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

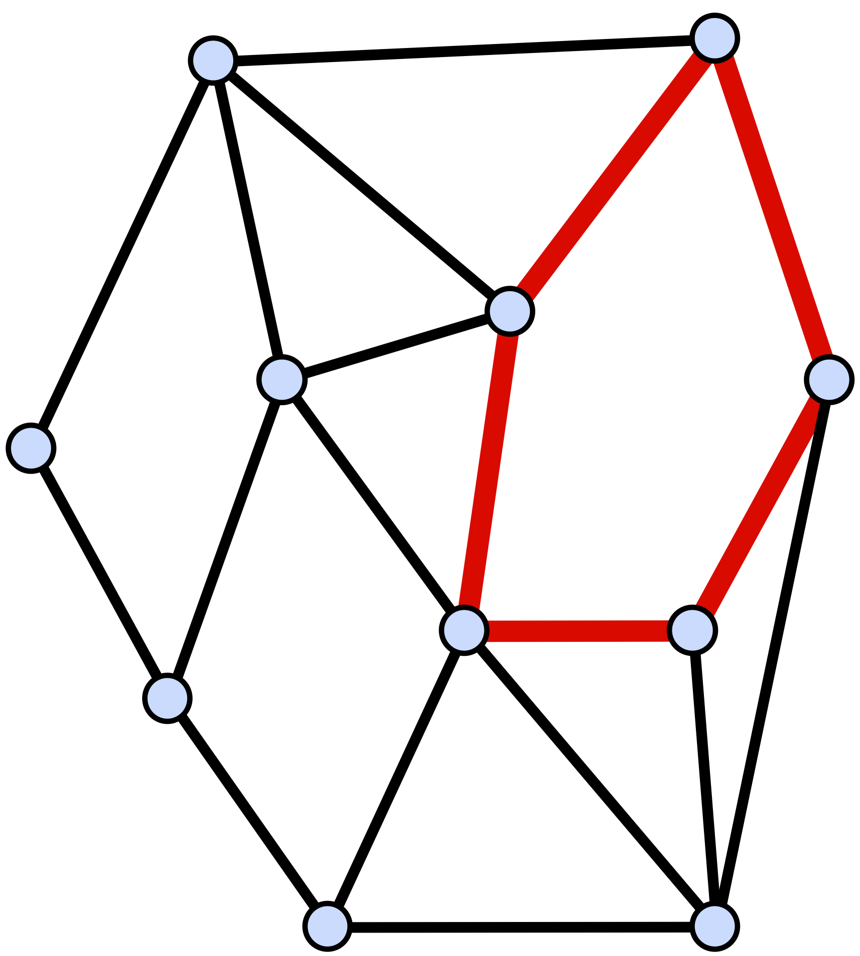
vertices edges faces

Polygonal Mesh



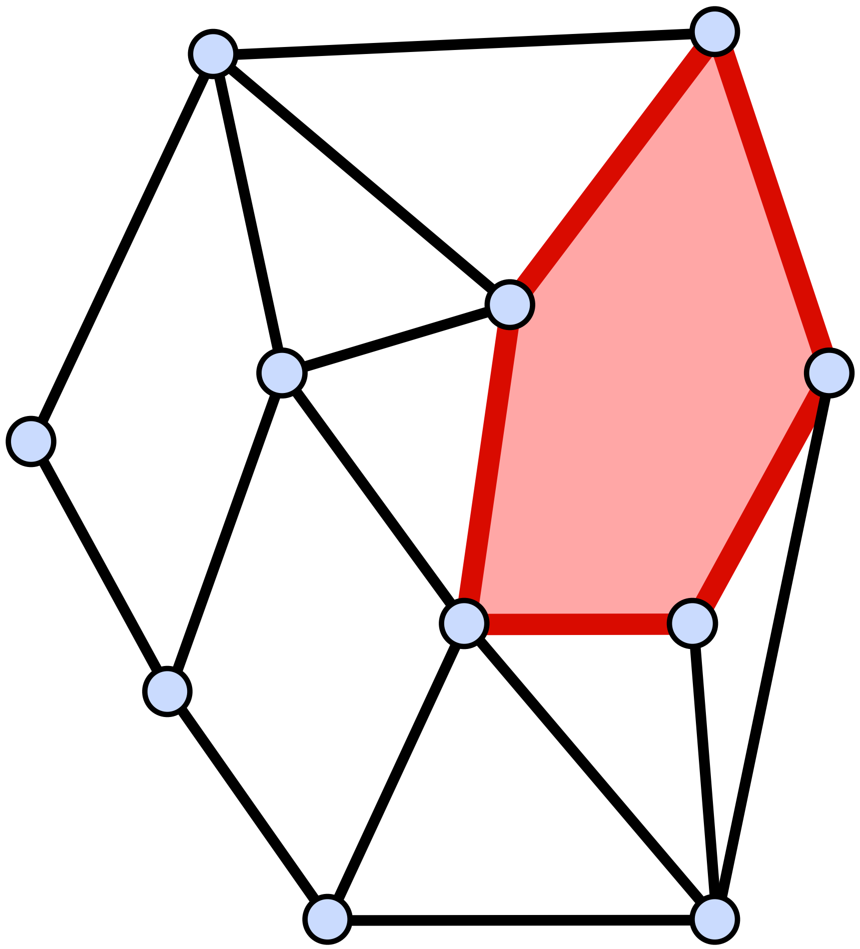
- A finite set *M* of closed, simple polygons Q_i is a polygonal mesh
- The intersection of two polygons in M is either empty, a vertex, or an edge
- Every edge belongs to at least one polygon

Polygonal Mesh



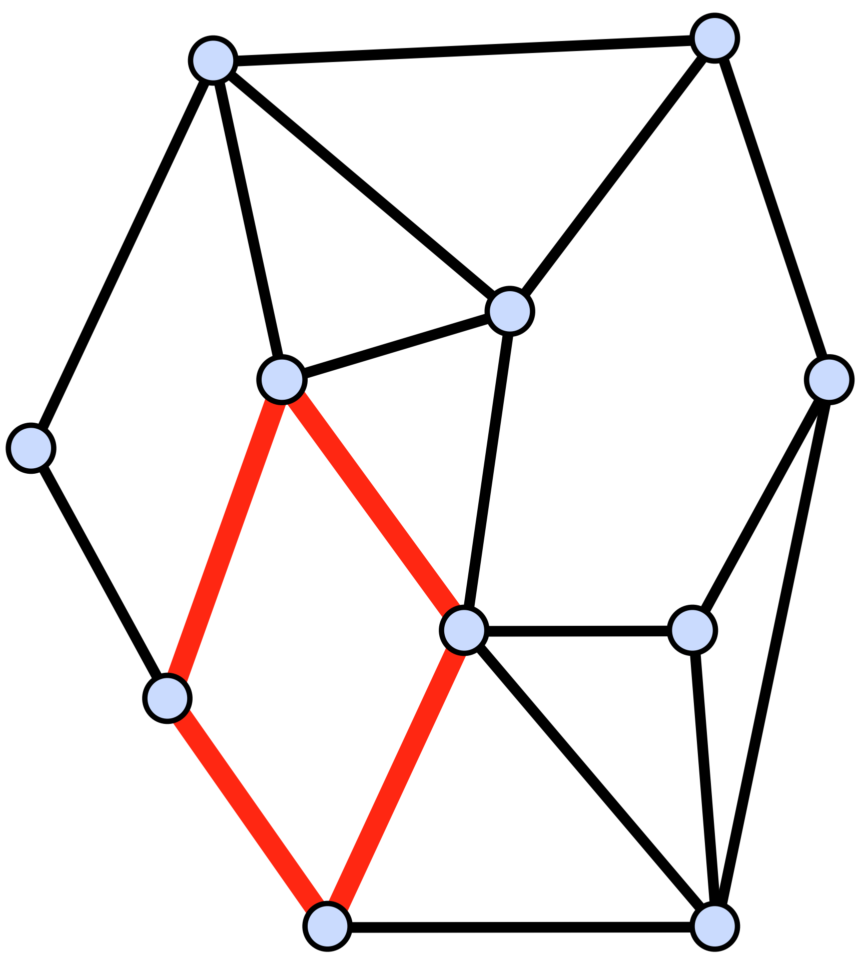
- A finite set M of closed, simple polygons Q_i is a polygonal mesh
- The intersection of two polygons in M is either empty, a vertex, or an edge
- Every edge belongs to at least one polygon
- Each Q_i defines a **face** of the polygonal mesh

Polygonal Mesh



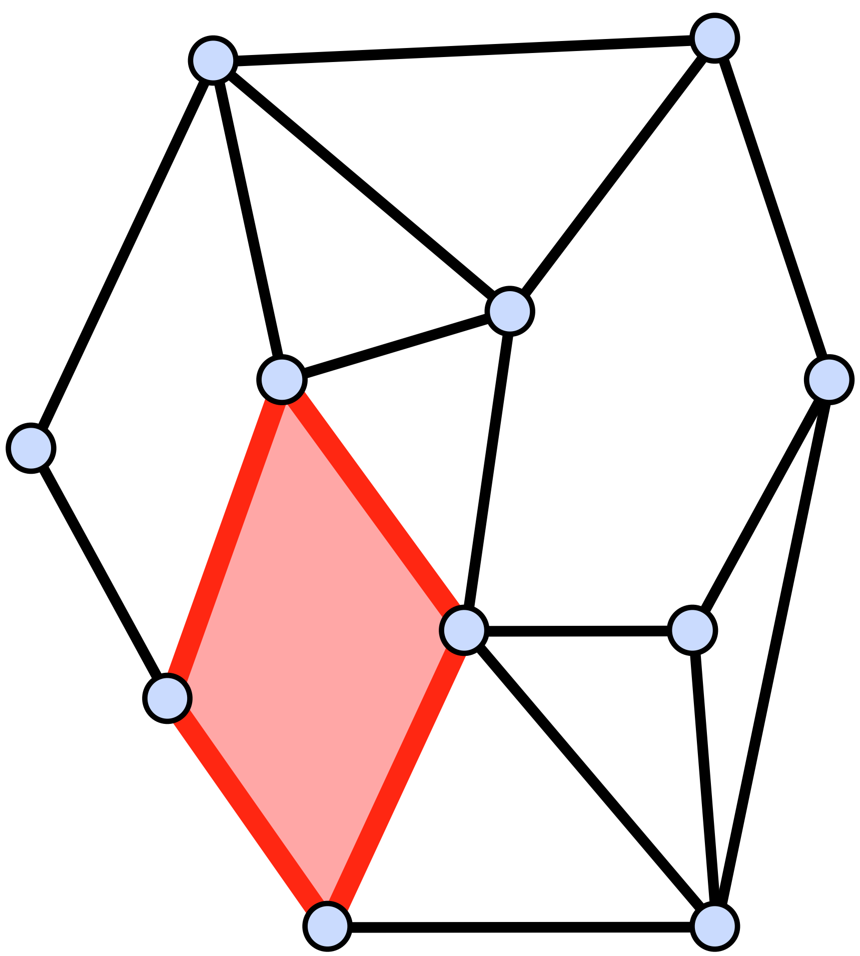
- A finite set M of closed, simple polygons Q_i is a polygonal mesh
- The intersection of two polygons in M is either empty, a vertex, or an edge
- Every edge belongs to at least one polygon
- Each Q_i defines a face of the polygonal mesh

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
- Each Q_i defines a **face** of the polygonal mesh

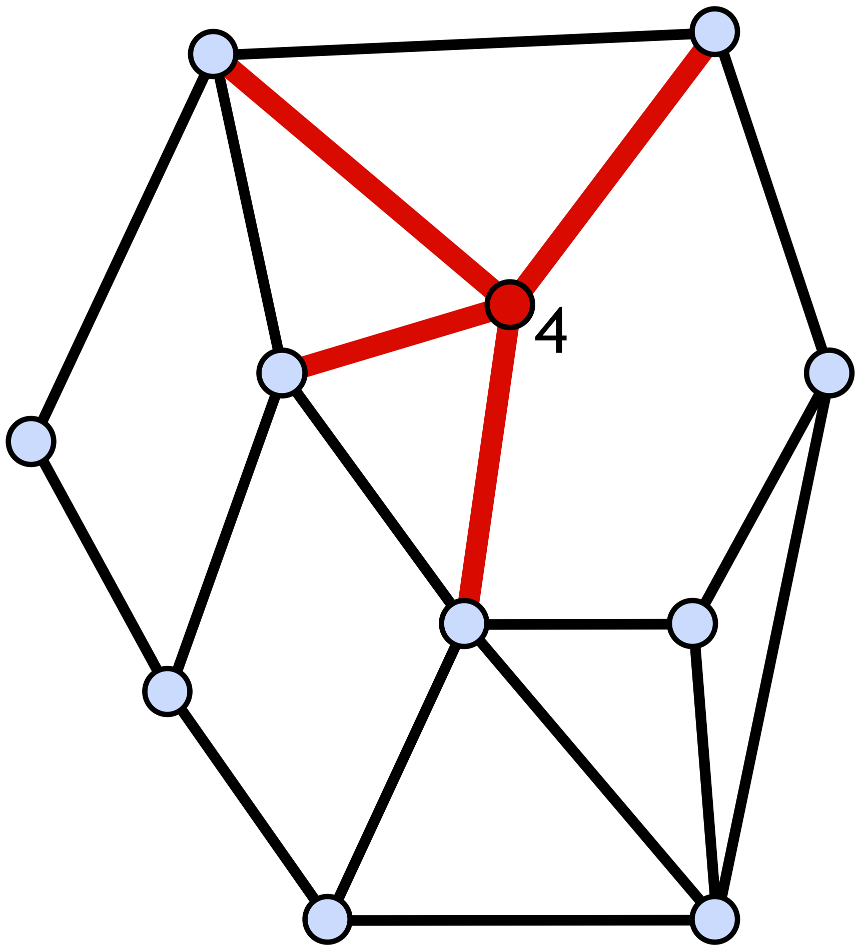
Polygonal Mesh



- 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
- Each Q_i defines a face of the polygonal mesh

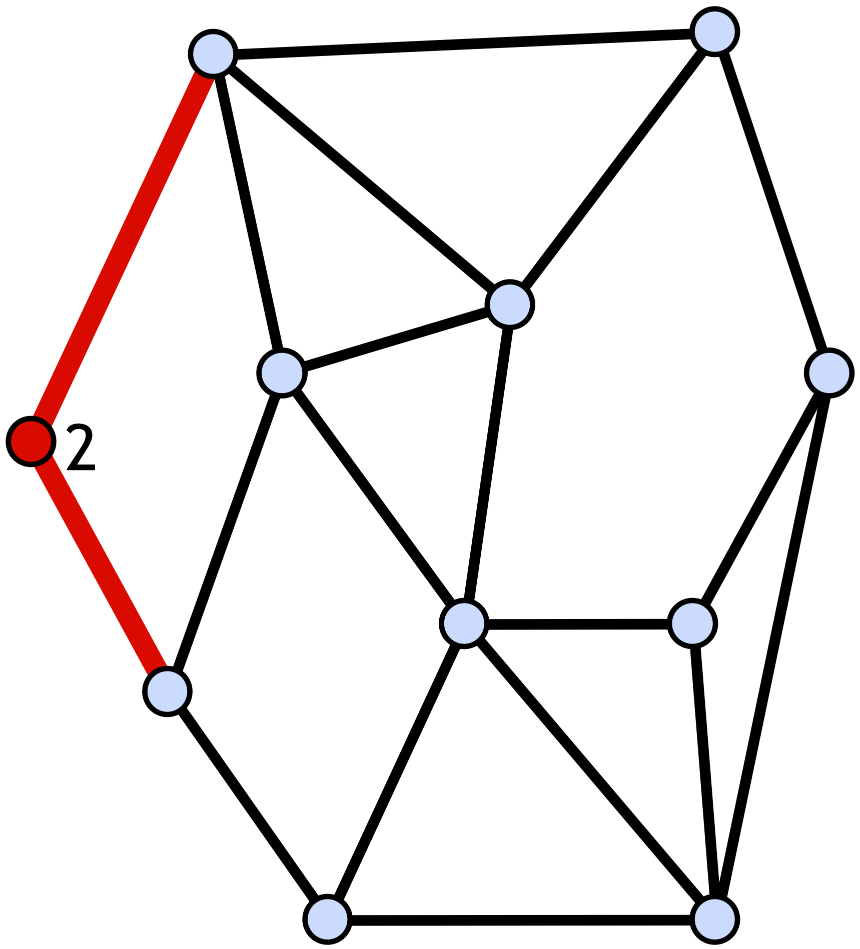
Polygonal Mesh

- Vertex **degree** or **valence** = number of incident edges

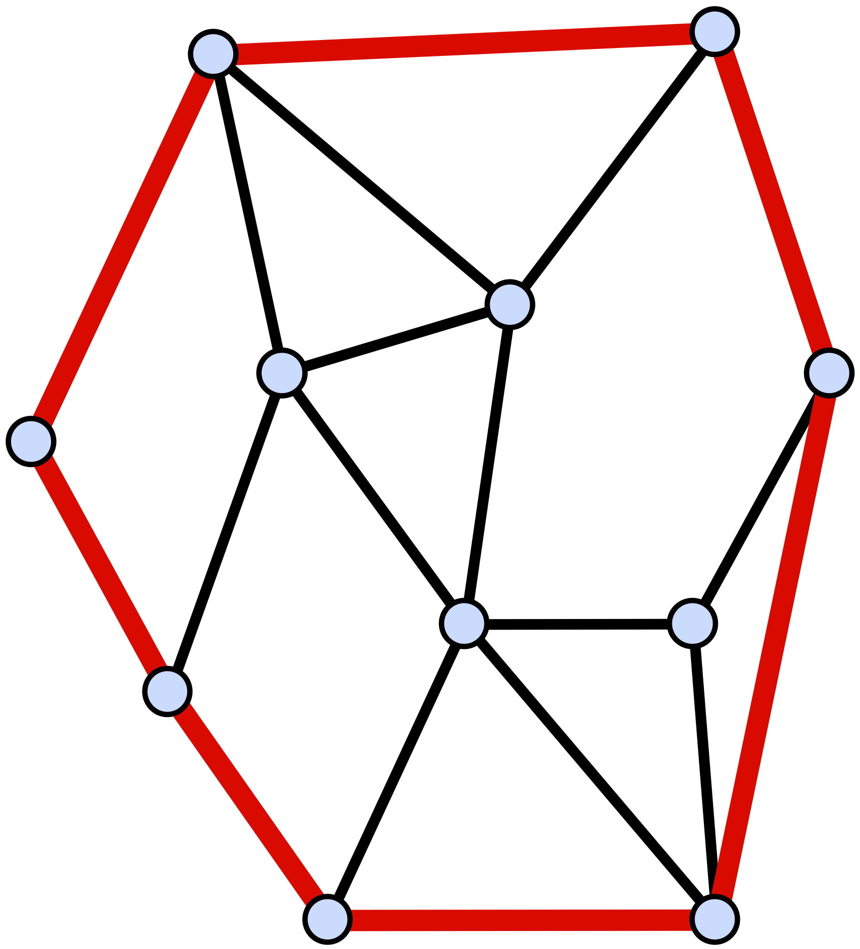


Polygonal Mesh

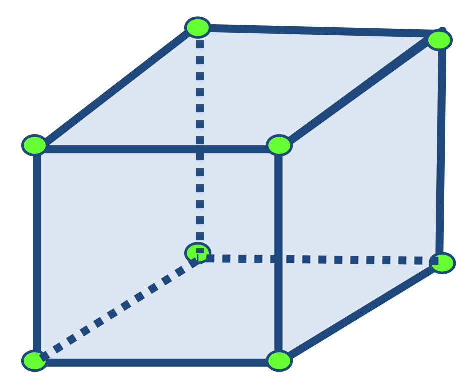
- Vertex degree or valence = number of incident edges



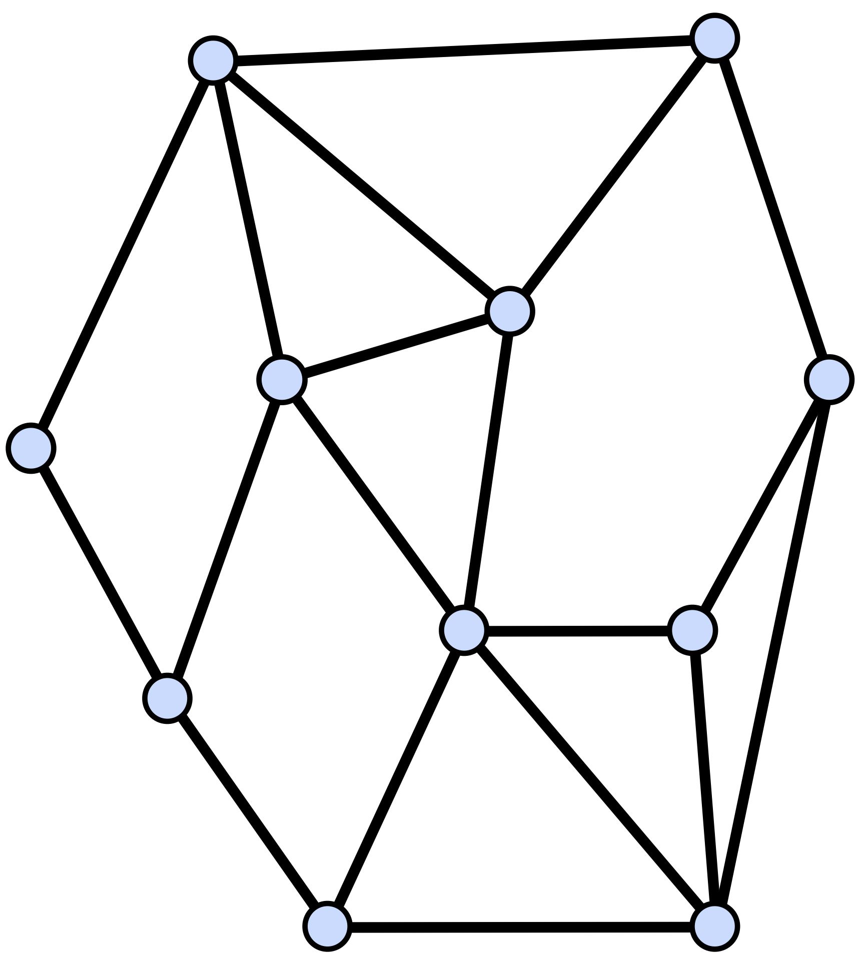
Polygonal Mesh



- 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

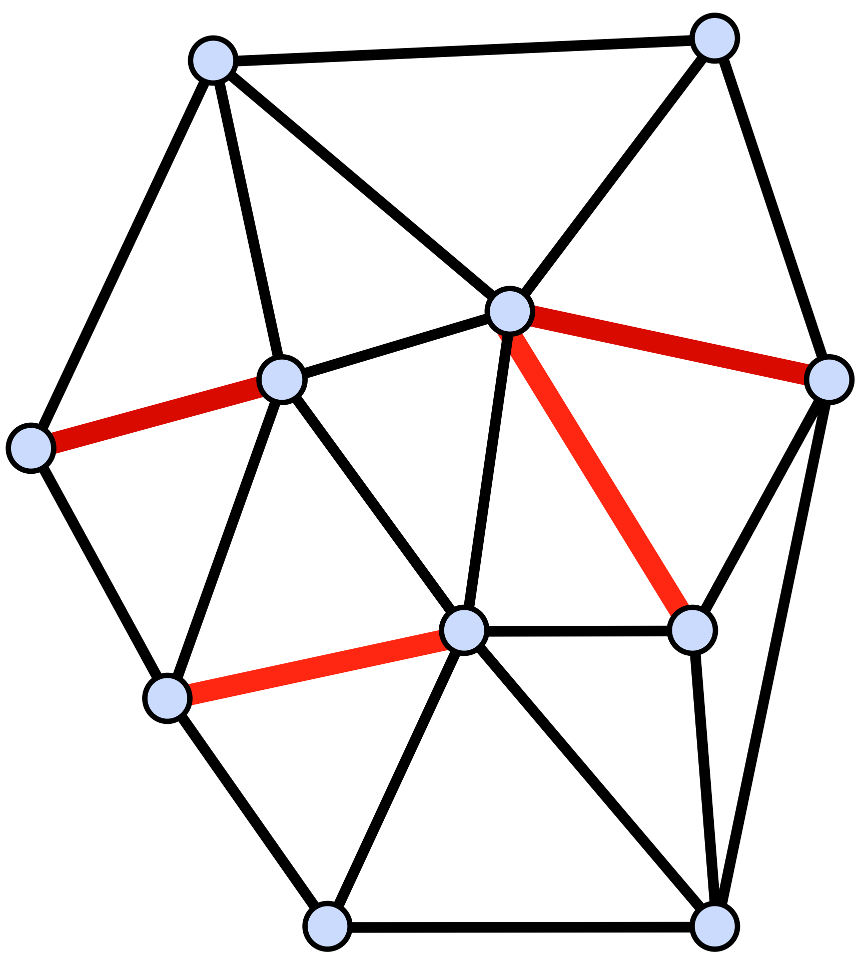


Triangulation



- Polygonal mesh where every face is a triangle
- Simplifies data structures
- Simplifies rendering
- Simplifies algorithms
- Each face planar and convex
- Any polygon can be triangulated

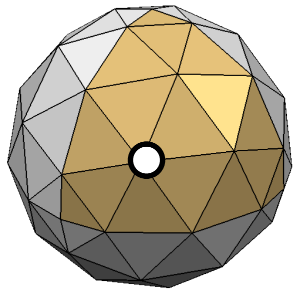
Triangulation



- Polygonal mesh where every face is a triangle
- Simplifies data structures
- Simplifies rendering
- Simplifies algorithms
- Each face planar and convex
- Any polygon can be triangulated

Triangle Meshes

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

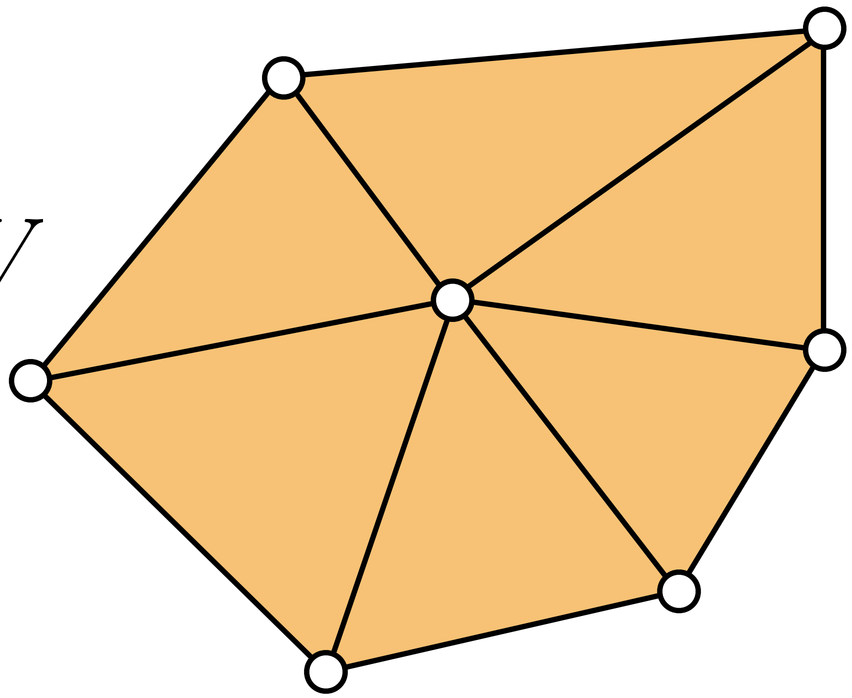


$$V = \{v_1, \ldots, v_n\}$$

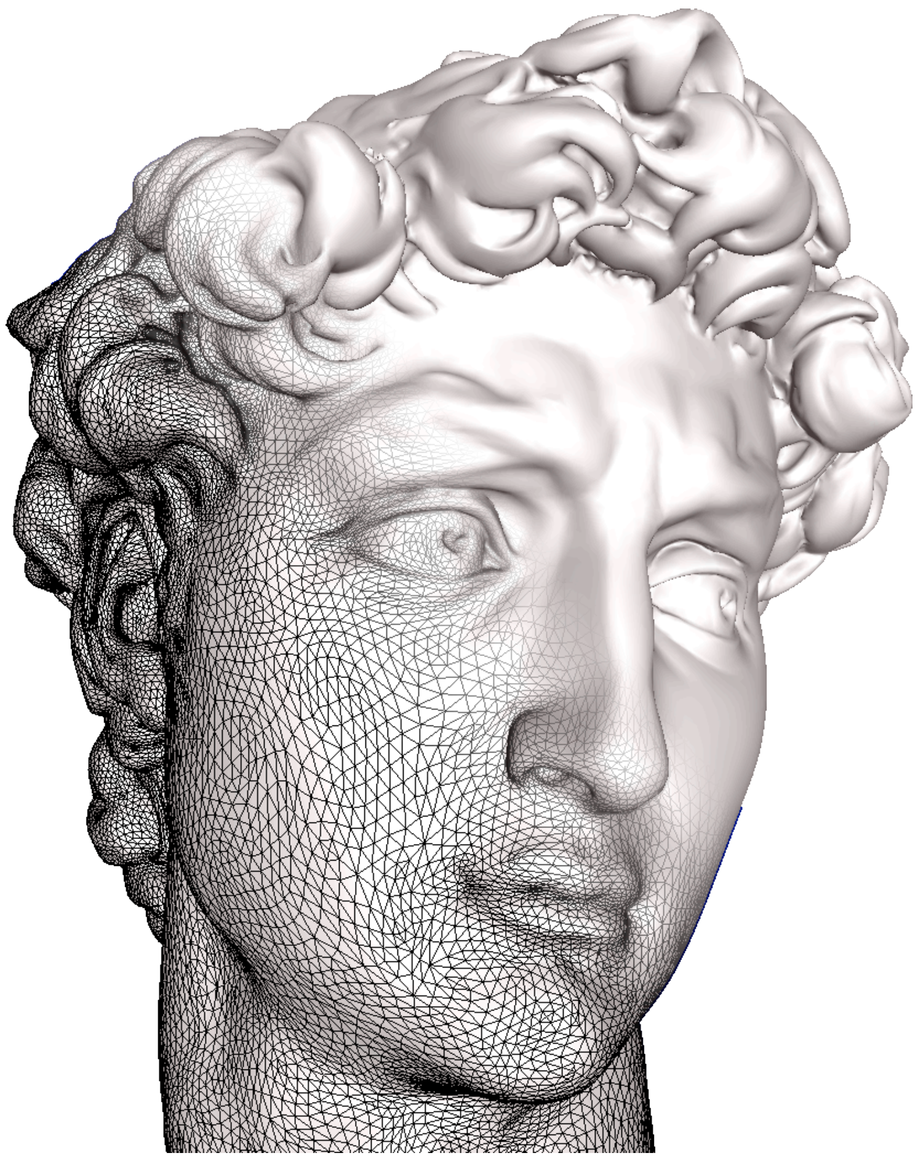
$$E = \{e_1, \ldots, e_k\}, \quad e_i \in V \times V$$

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

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



Data Structures



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

Simple Data Structures: Triangle List

- STL 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 connectivity information

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

Simple Data Structures:Indexed Face Set

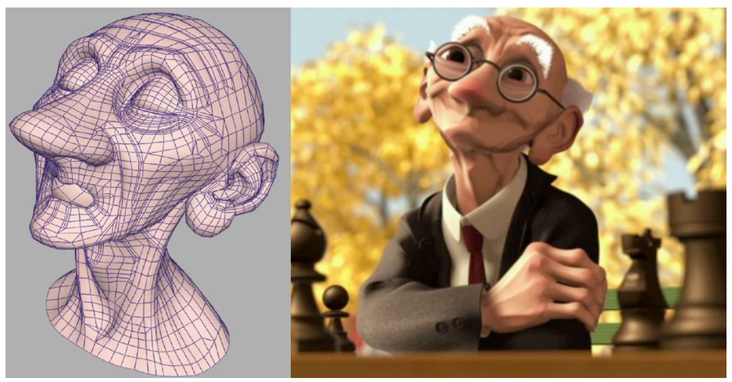
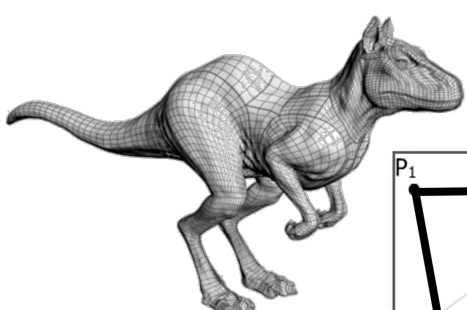
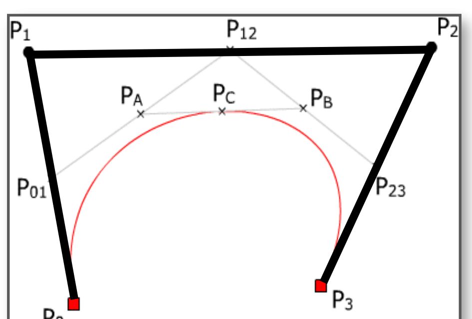
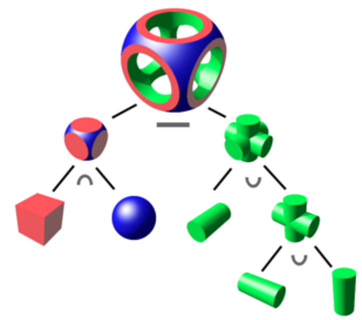
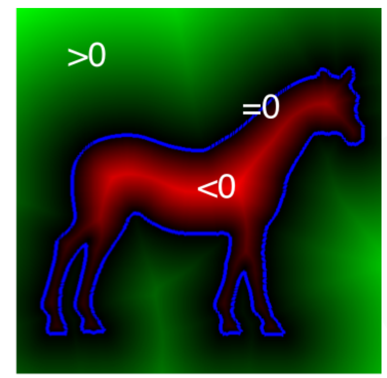
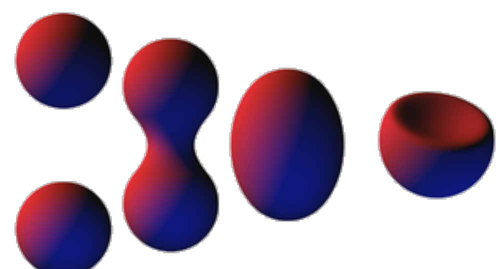
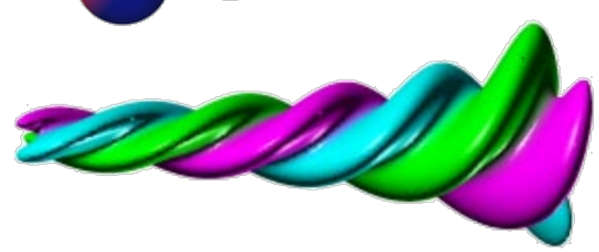
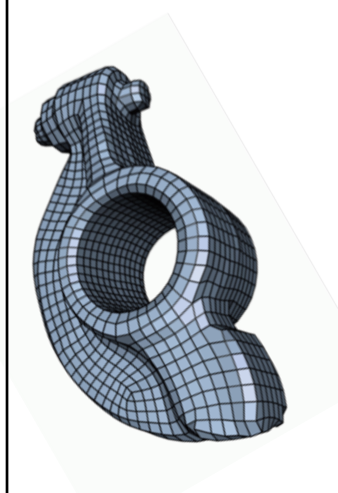
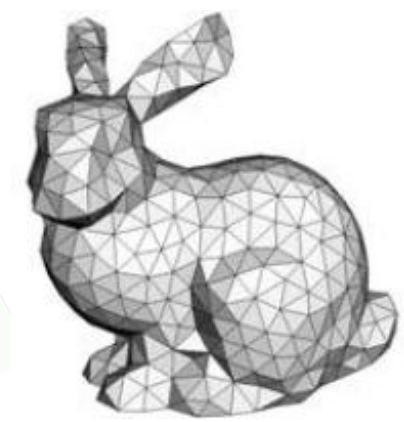
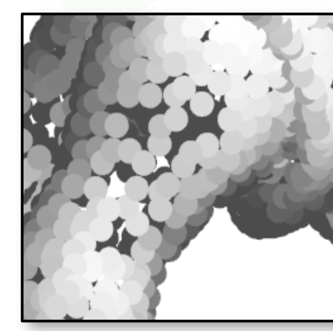
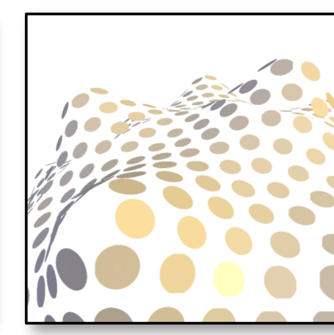
- Used in formats
- OBJ, OFF, WRL
- Storage
  - Vertex: position
  - Face: vertex indices
  - 12 bytes per vertex
  - 12 bytes per face
  - 36*v bytes for the mesh
- No explicit neighborhood info

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
...	...	...	...

queue: halfedge
datastructure!

Summary

Parametric	Implicit	Discrete/Sampled
   • Splines, tensor-product surfaces • Subdivision surfaces	    • Distance fields • Metaballs/blobs	    • Meshes • Point set surfaces

CONVERSIONS

Implicit → Mesh
Mesh → Points (next time!)

POINTS → IMPLICIT

Implicit Surface Reconstruction

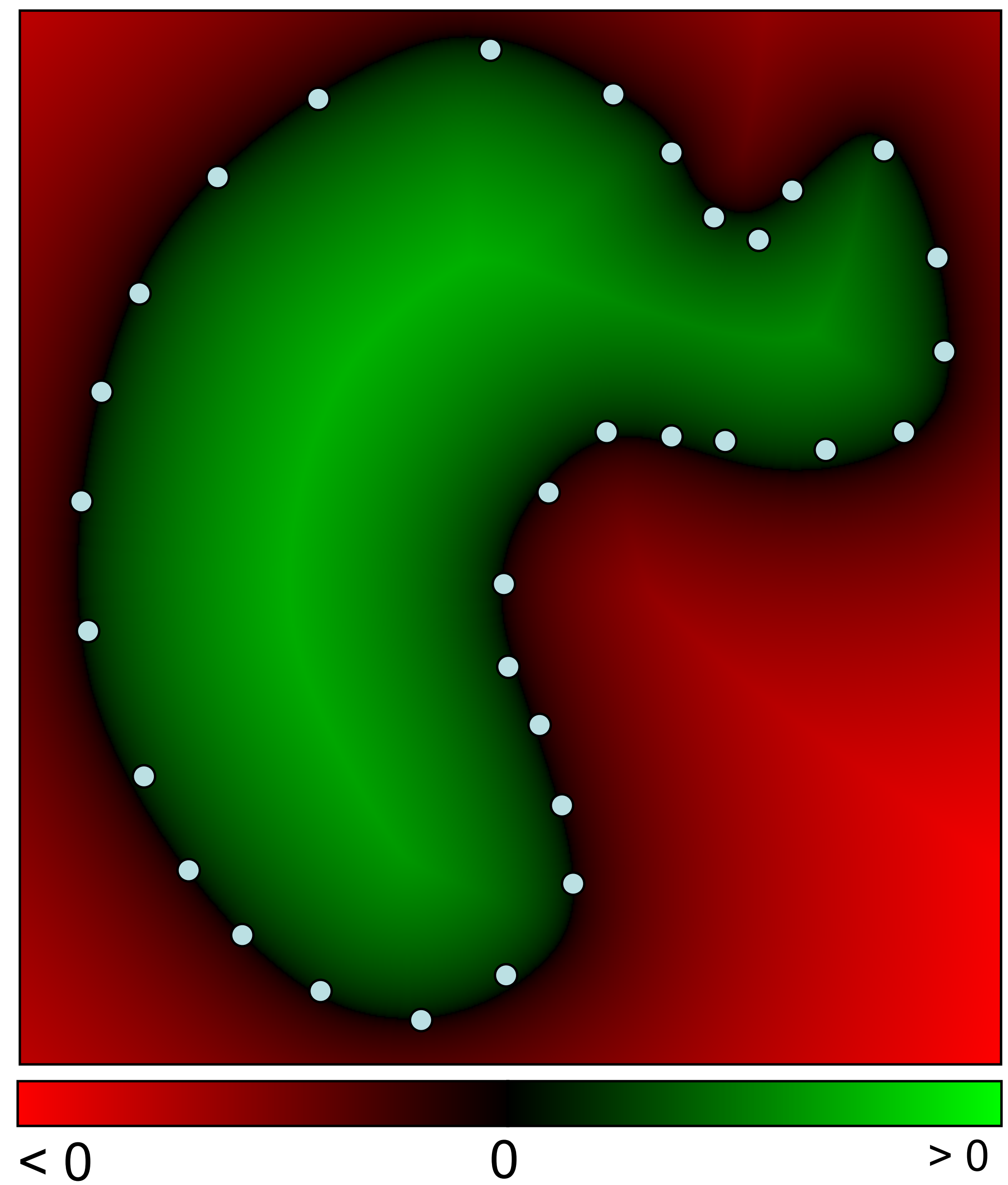
UC San Diego

Implicit Function Approach

- Define a function

$$f : R^3 \rightarrow R$$

with value < 0 outside the shape and > 0 inside

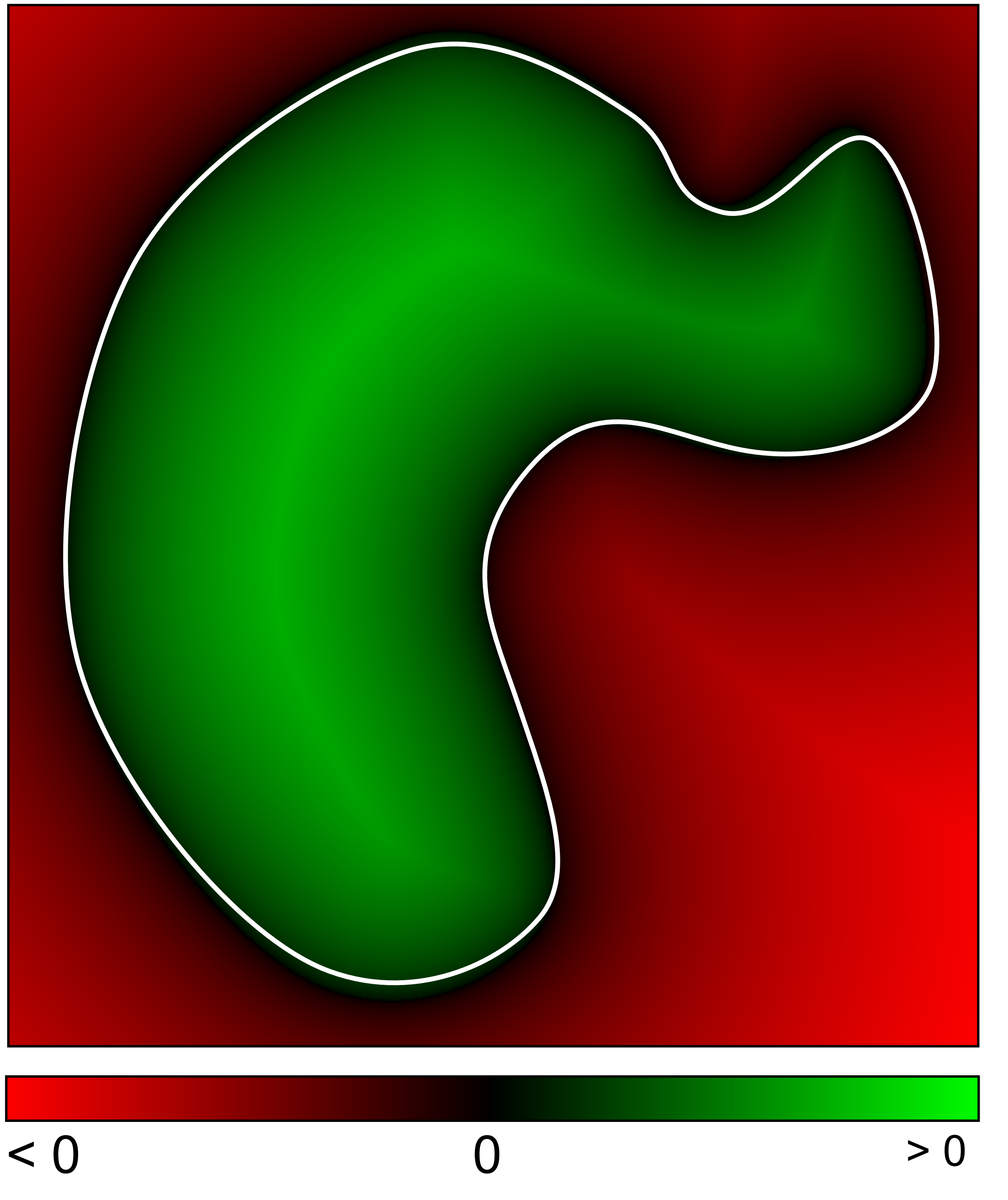


Implicit Function Approach

- Define a function

$$f : R^3 \rightarrow R$$

with value < 0 outside the shape and > 0 inside

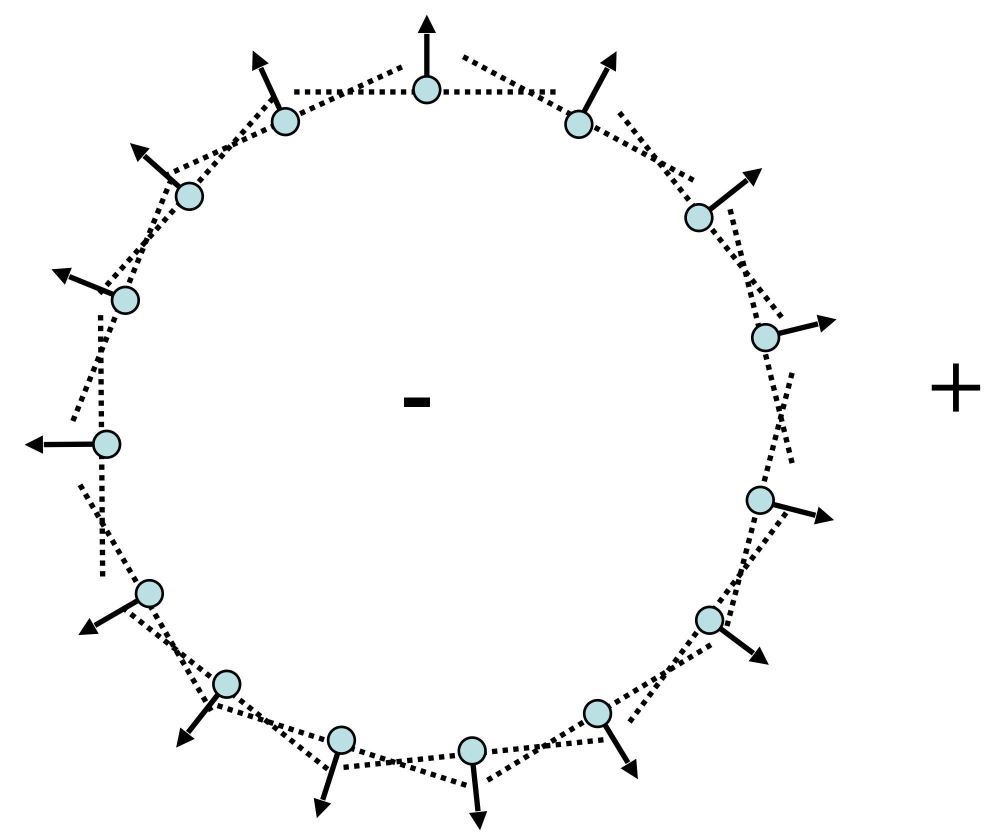


- Extract the zero-set

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

SDF from Points and Normals

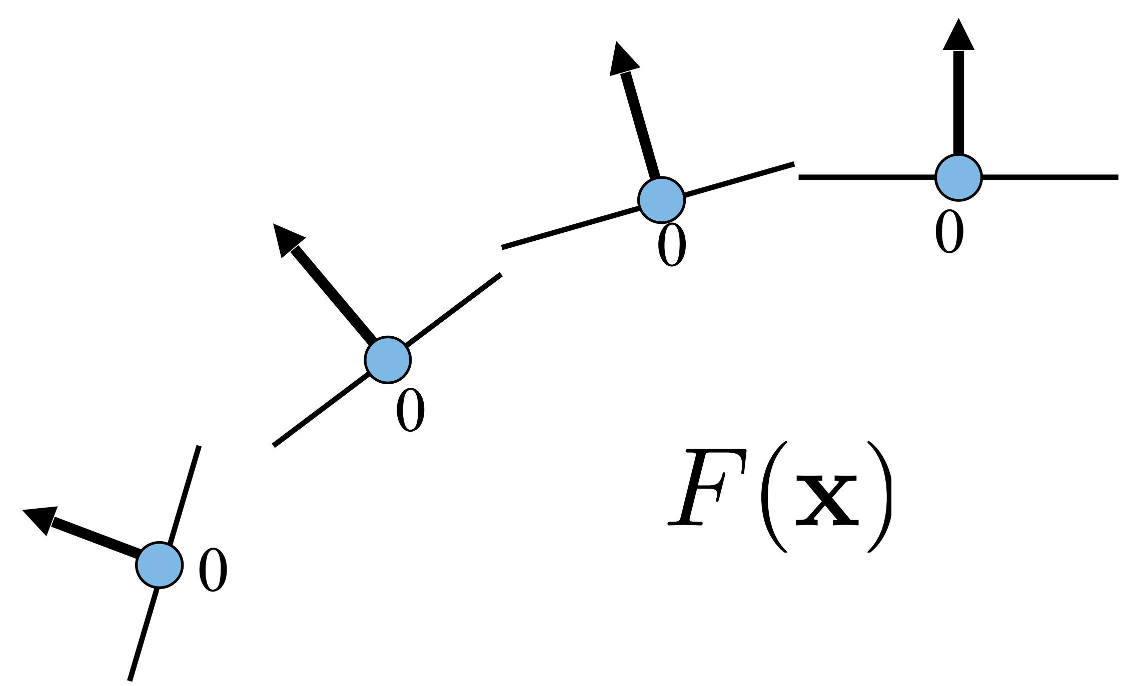
- Input: Points + Normals
- Normals help to distinguish between inside and outside
- Computed via locally fitting planes at the points



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

Smooth SDF

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

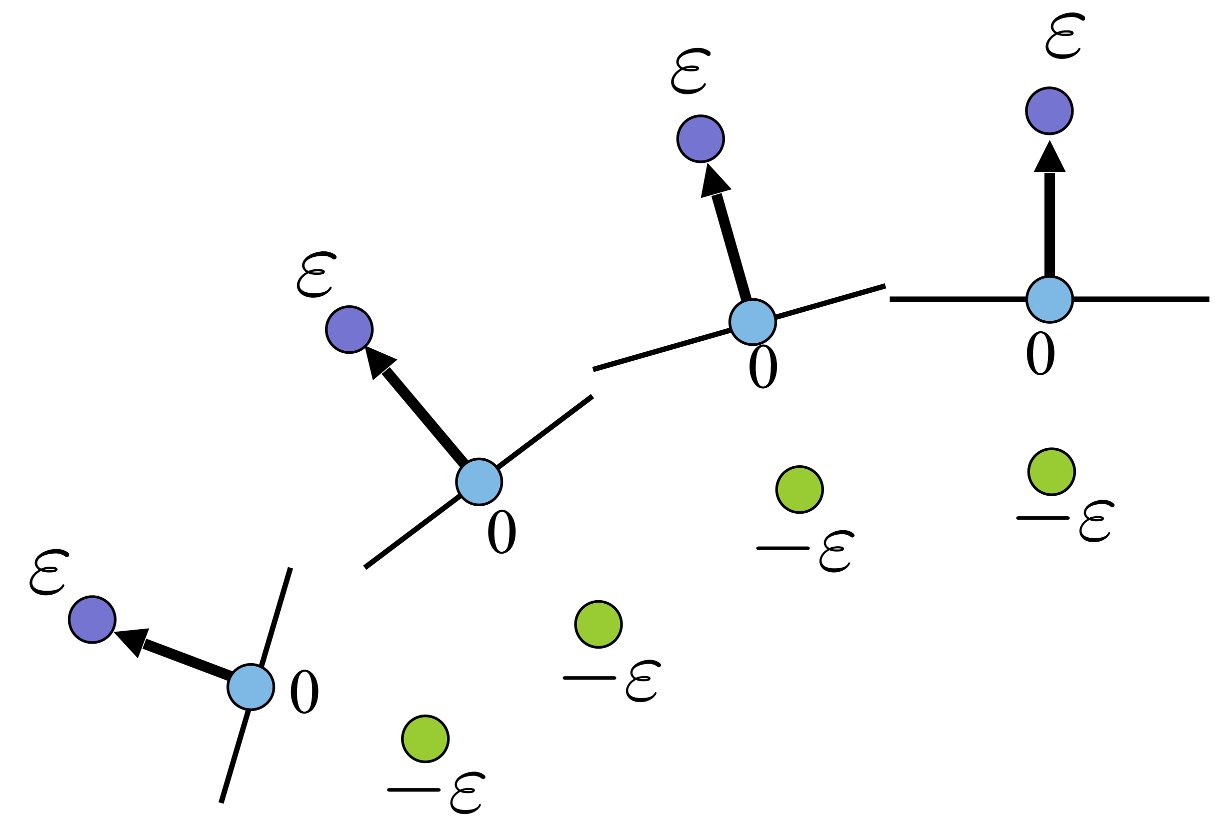


"Reconstruction and representation of 3D objects with radial basis functions", Carr et al., ACM SIGGRAPH 2001

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:

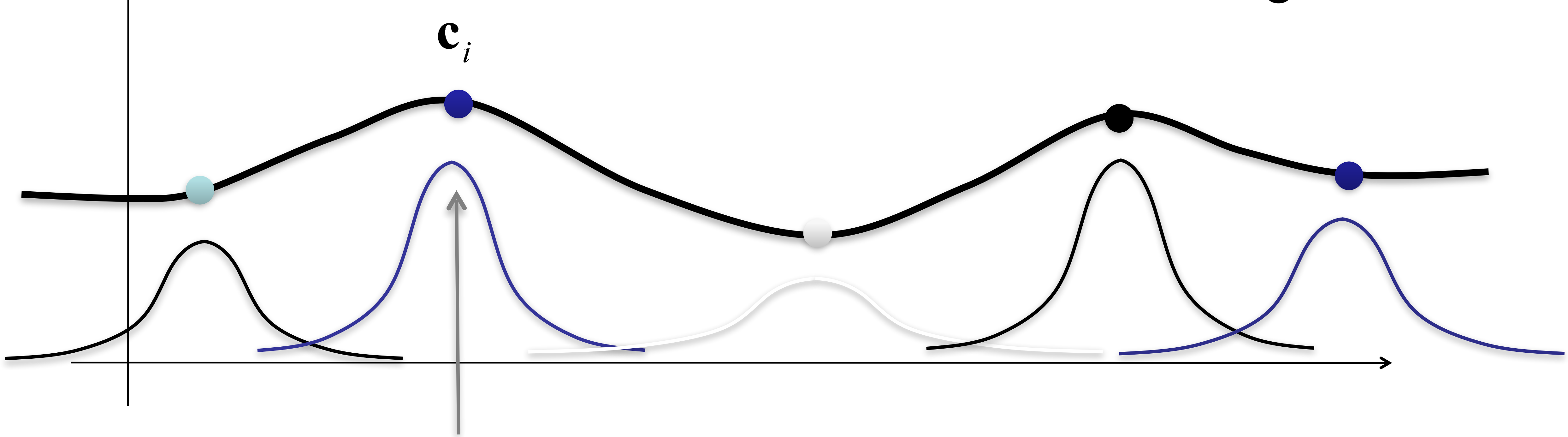
$$\varphi(r) = r^3$$

Radial Basis Function Interpolation

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

Kernel centers: on- and off-surface points

How do we find the weights?



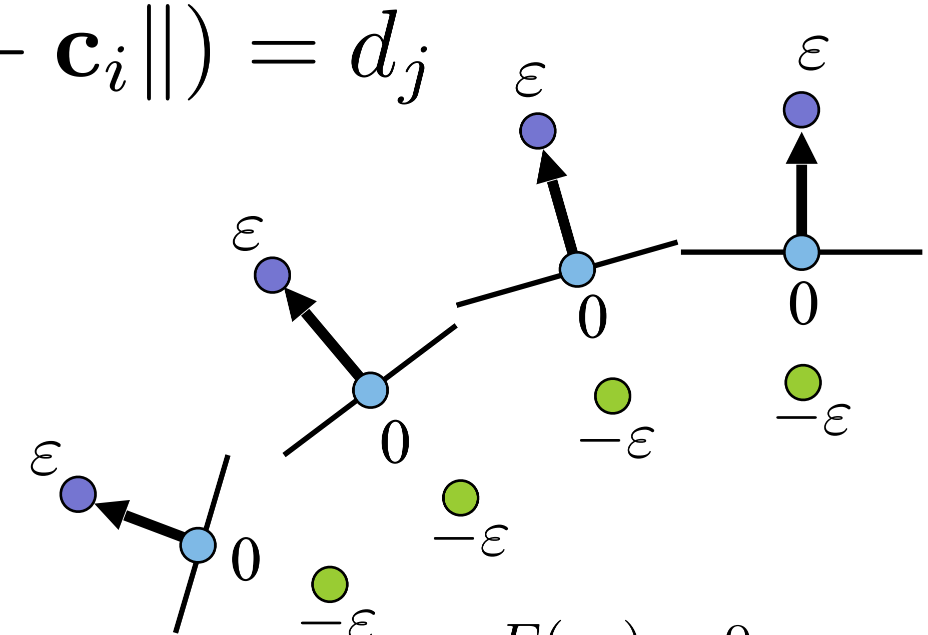
$$\varphi_i(\mathbf{x}) = \varphi\left(\left\|\mathbf{x} - \mathbf{c}_i\right\|\right)$$

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:
  - Globally supported $\varphi(r) = r^3$
  - Leads to dense symmetric linear system
  - C2 smoothness
  - Works well for highly irregular sampling

RBF Kernels

- Polyharmonic spline

$$\varphi(r) = r^k \log(r),\ \ k = 2, 4, 6 \ldots$$

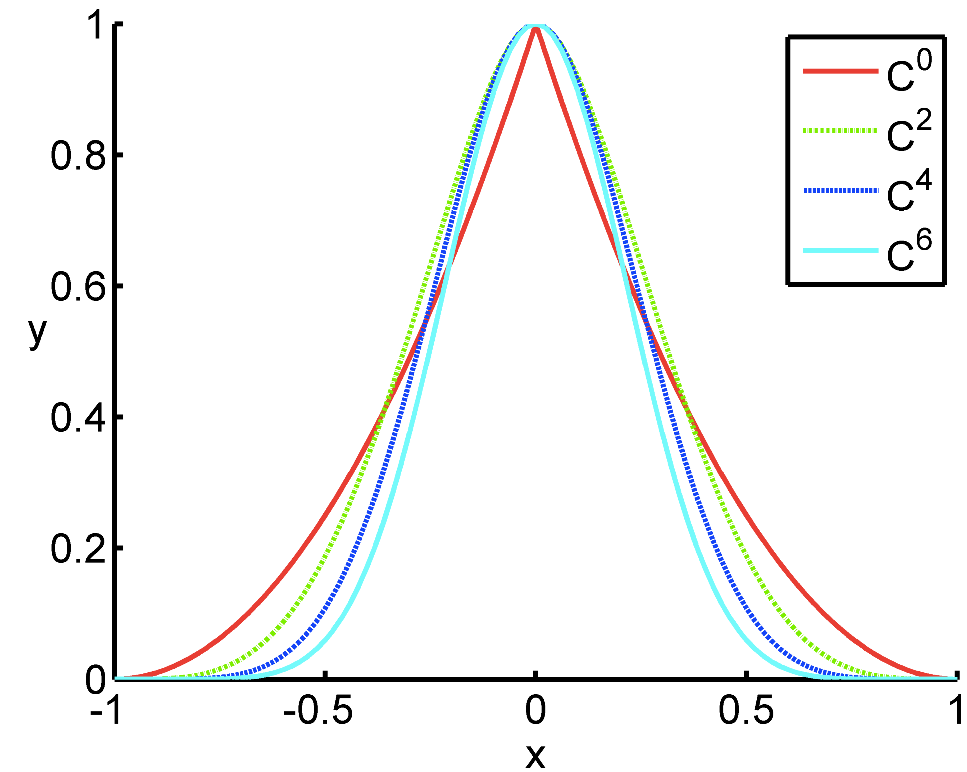
$$\varphi(r) = r^k,\ \ k = 1, 3, 5 \ldots$$

- Multiquadratic

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

- Gaussian

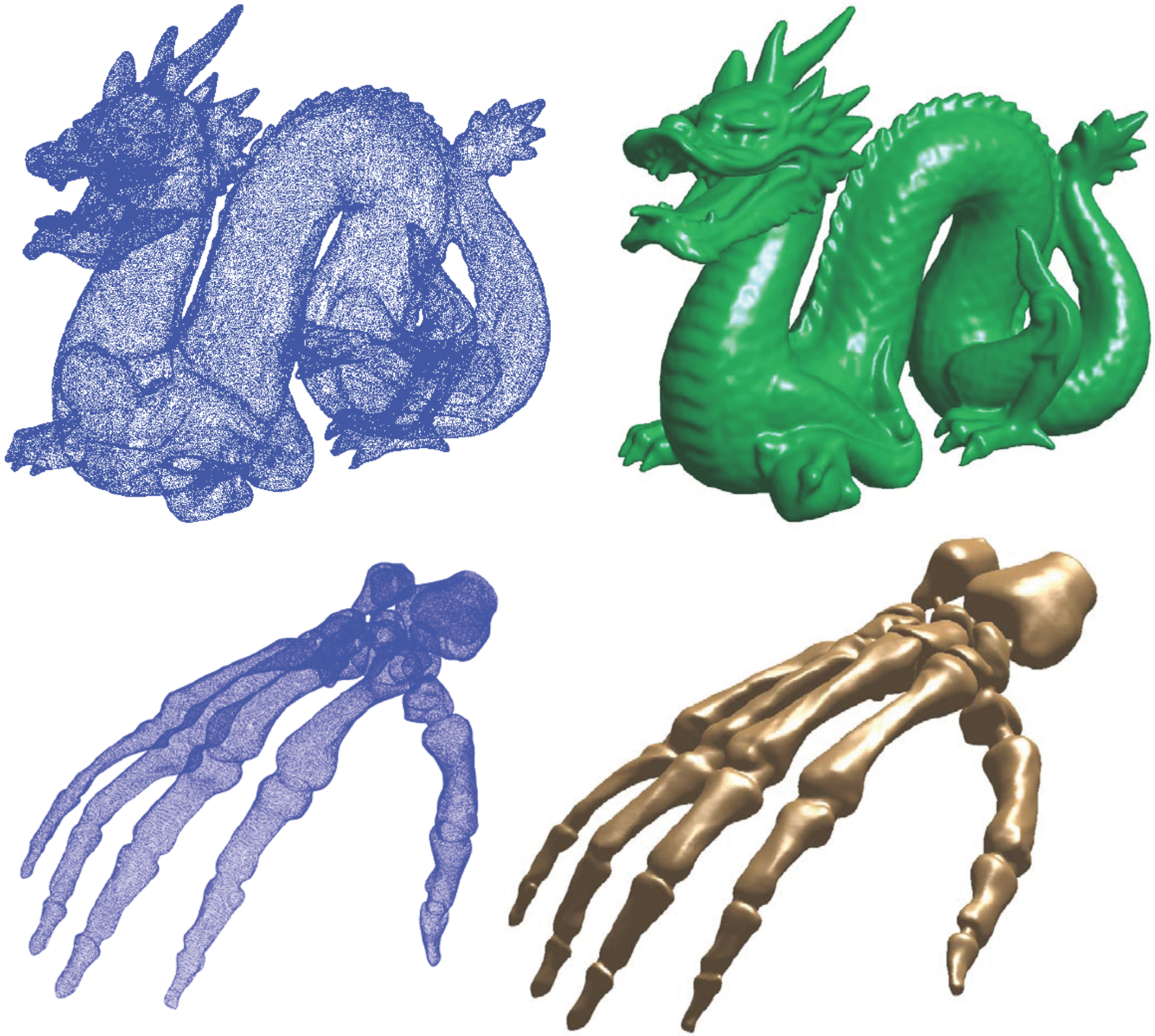
$$\varphi(r) = e^{-\beta r^2}$$



- B-Spline (compact support)

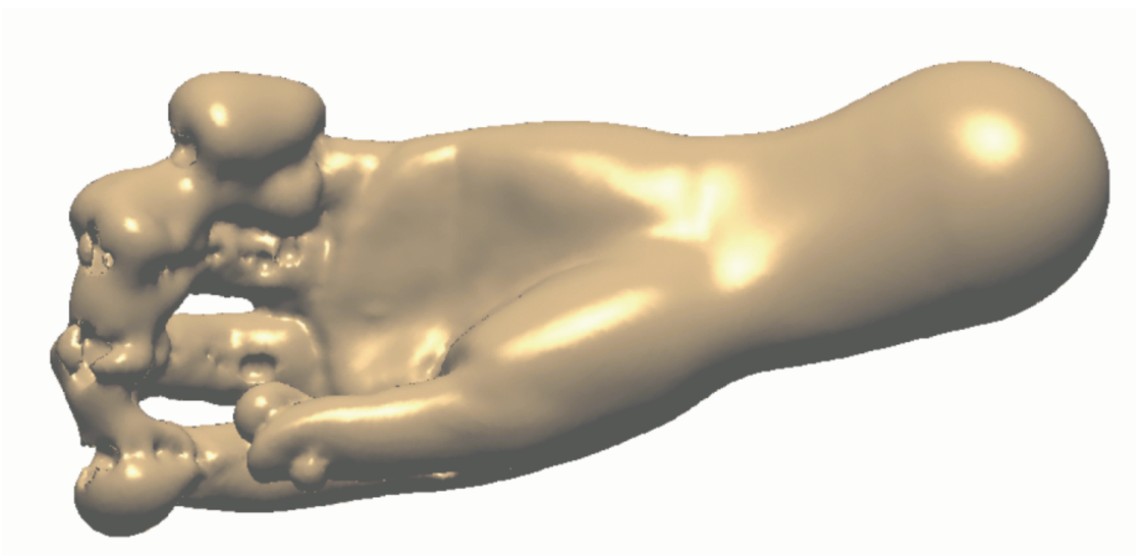
$$\varphi(r) = \text{piecewise-polynomial}(r)$$

RBF Reconstruction Examples

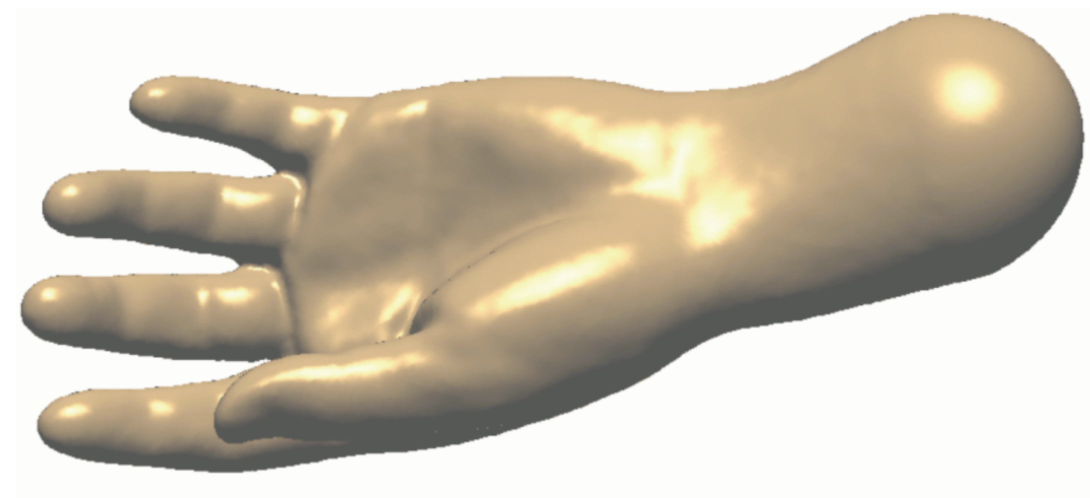


“Reconstruction and representation of 3D objects with radial basis functions”, Carr et al., ACM SIGGRAPH 2001

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

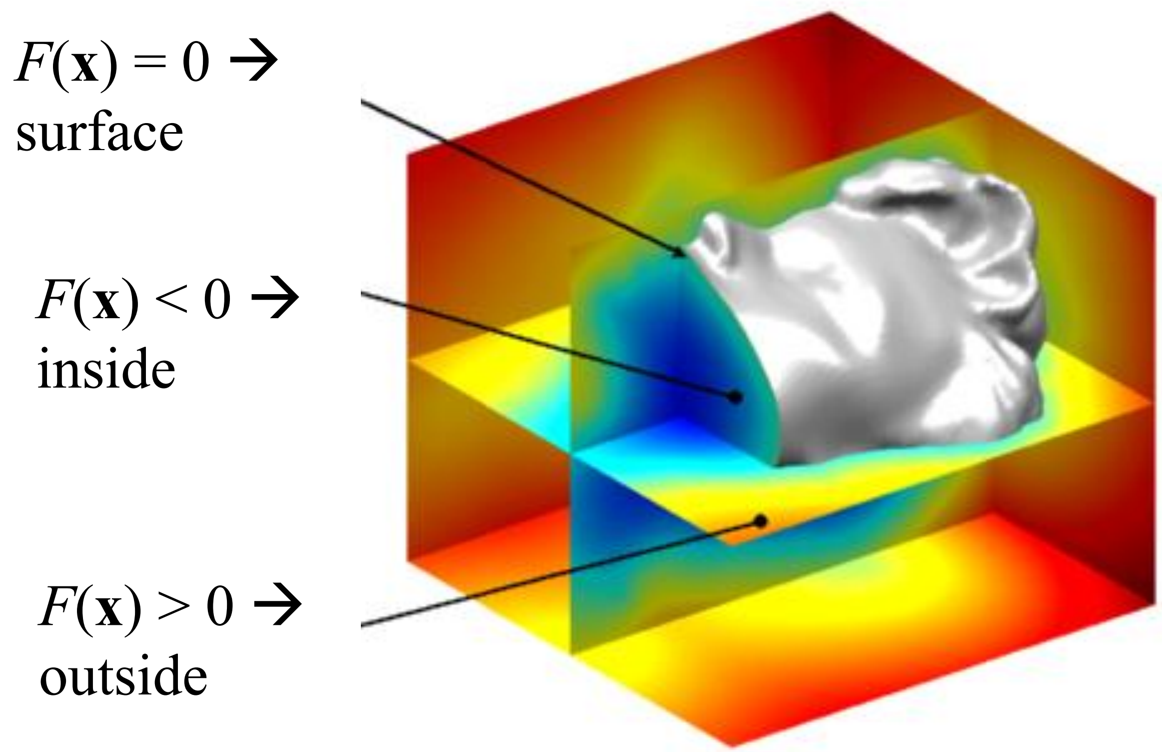
IMPLICIT → MESH

Marching Cubes

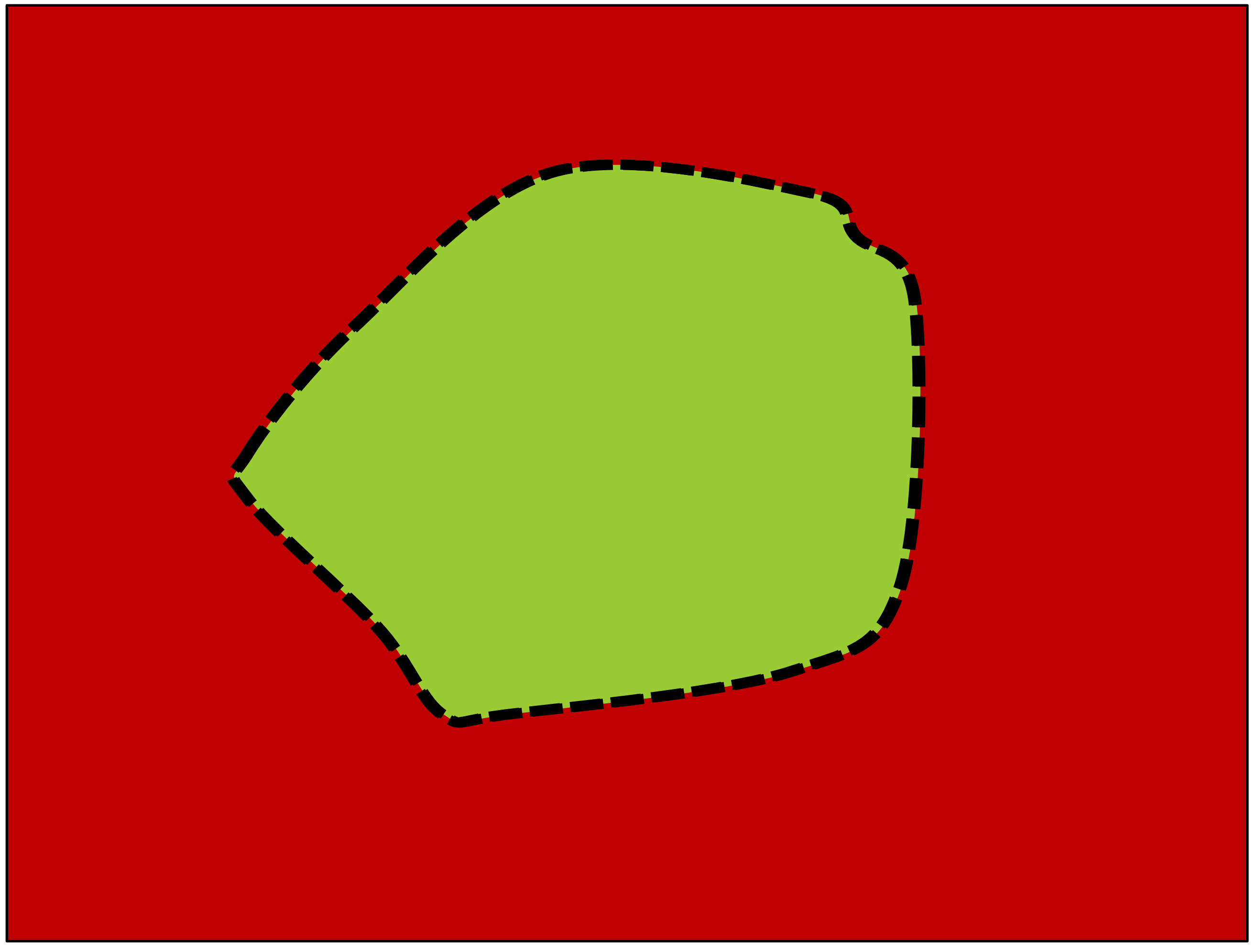
UC San Diego

Extracting the Surface

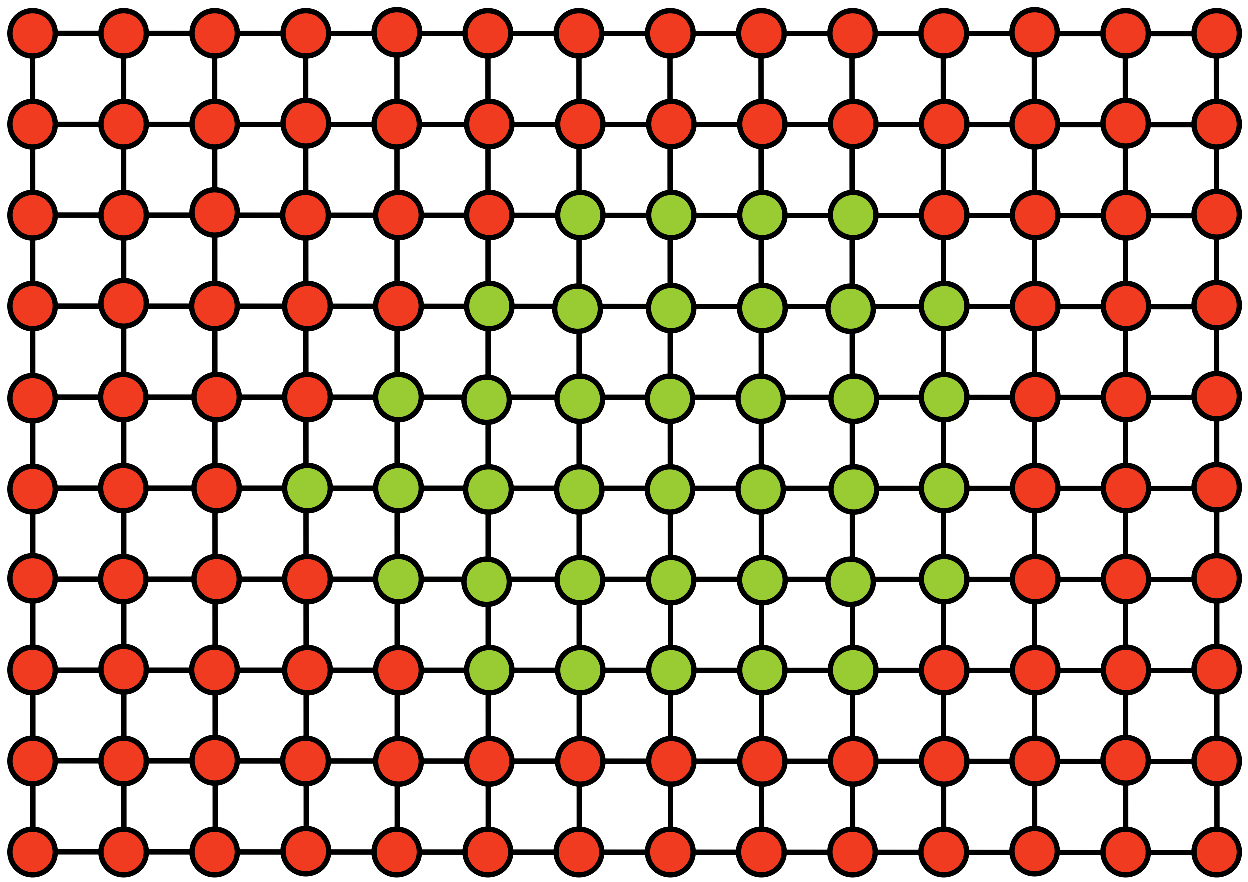
- Wish to compute a manifold mesh of the level set



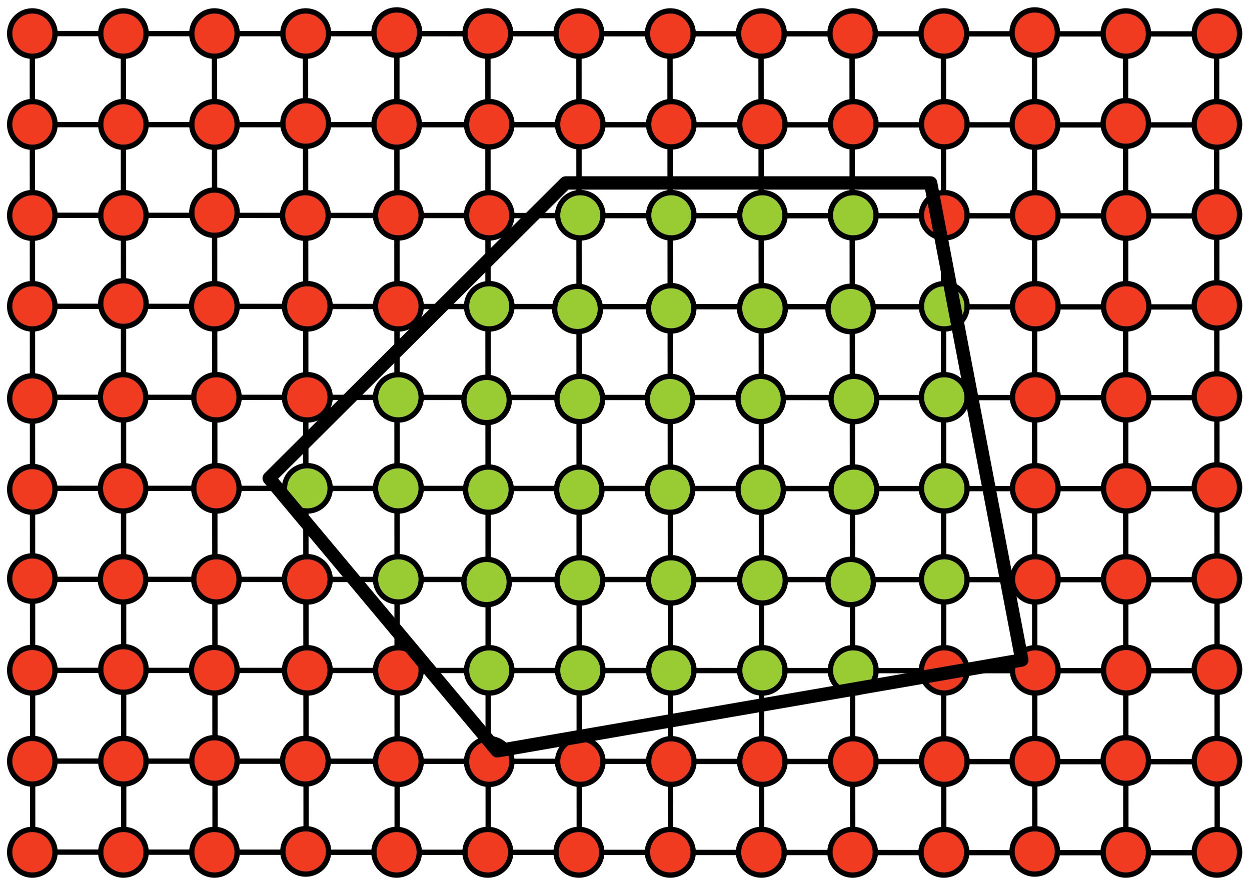
Sample the SDF



Sample the SDF



Sample the SDF

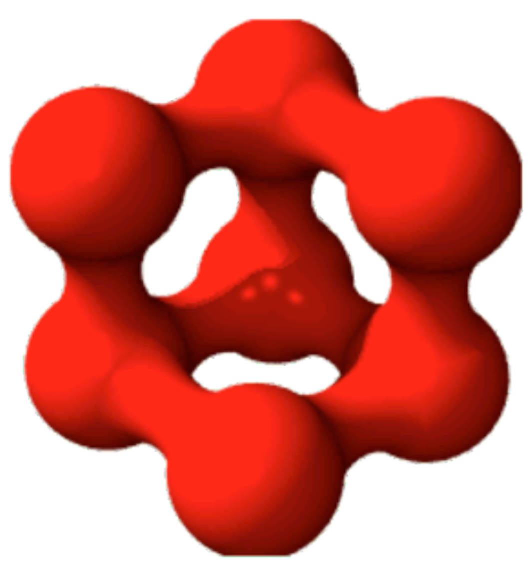


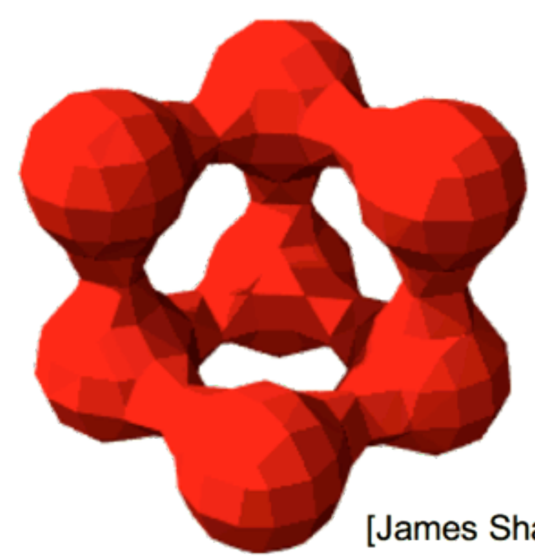
Marching Cubes

Converting from implicit to explicit representations.

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

Create a triangle mesh that approximates the surface.





[James Sharman]

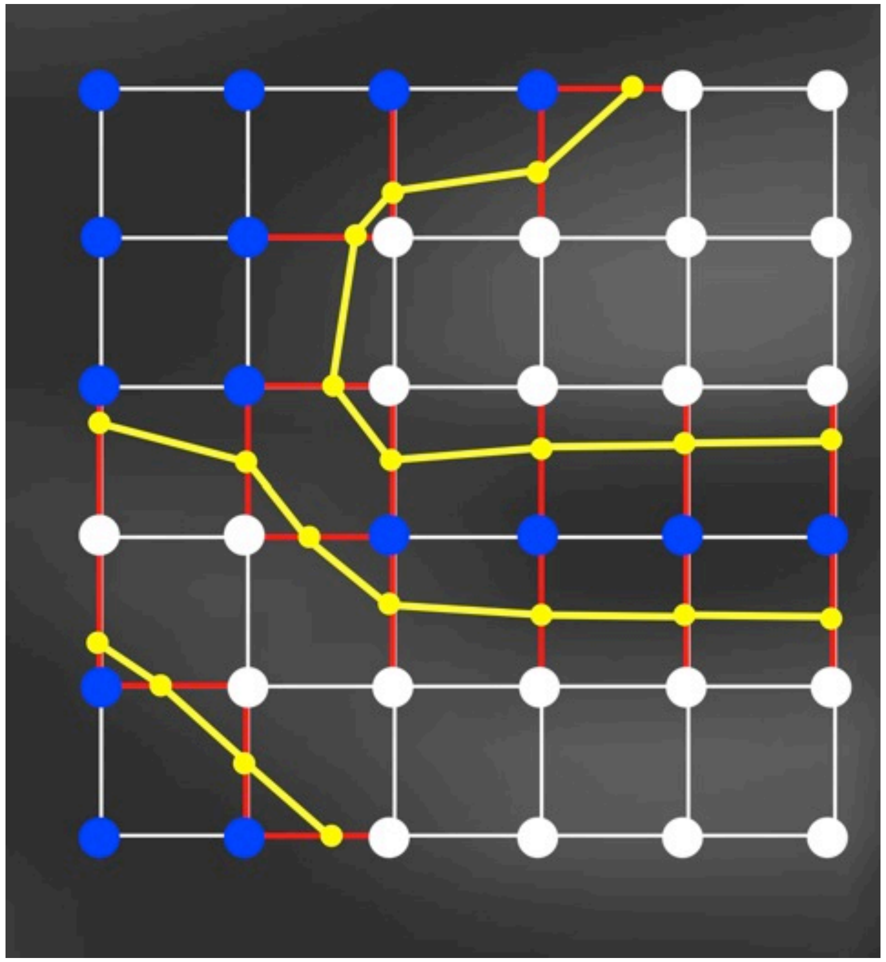
Lorensen and Cline, SIGGRAPH ‘87

Marching Squares (2D)

Given a function: $f(x)$

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

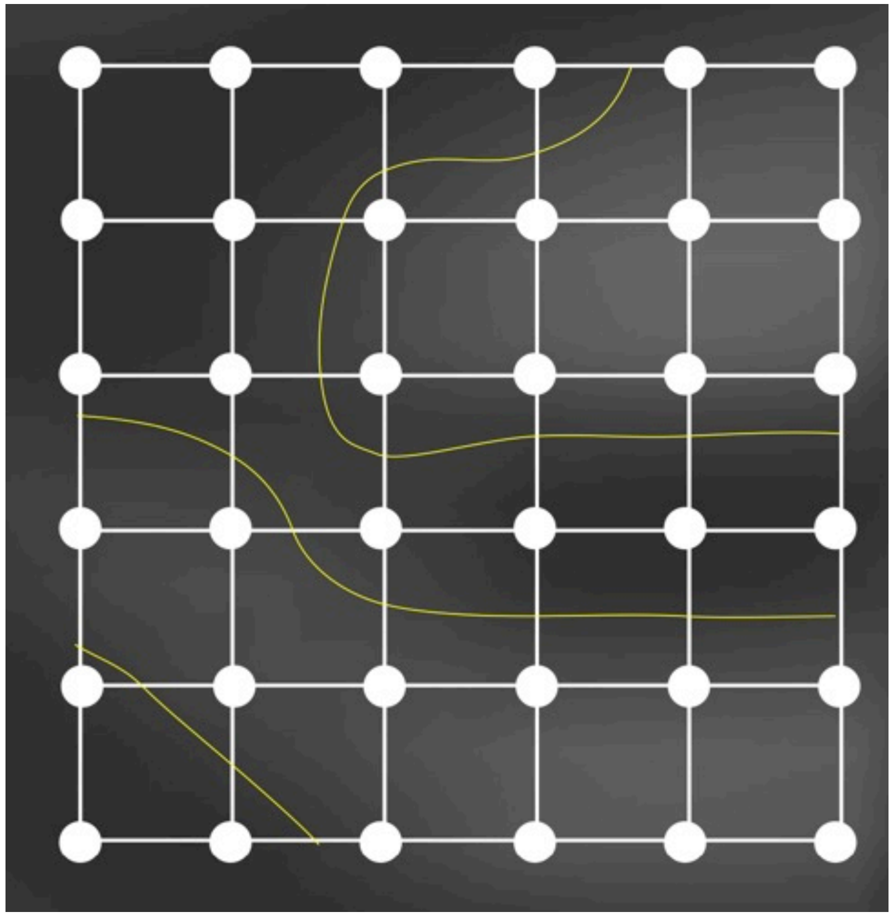
1. Discretize space.
2. Evaluate $f(x)$ on a grid.



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)

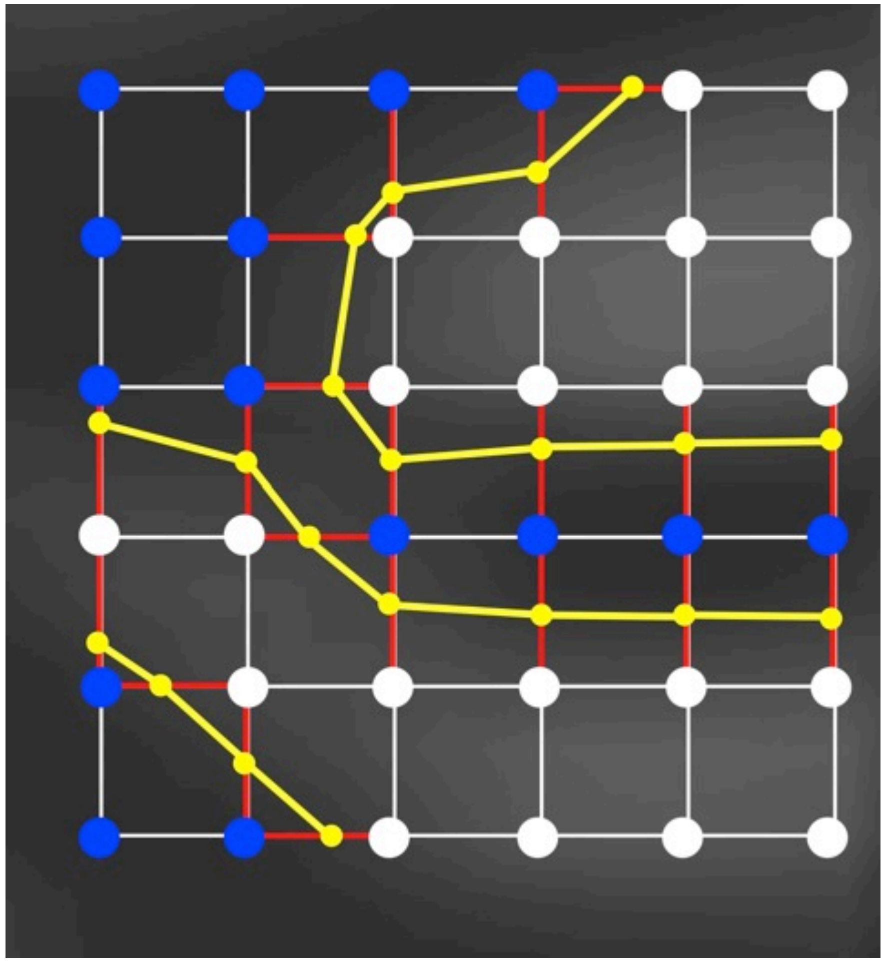
Computing the intersections:

- 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:

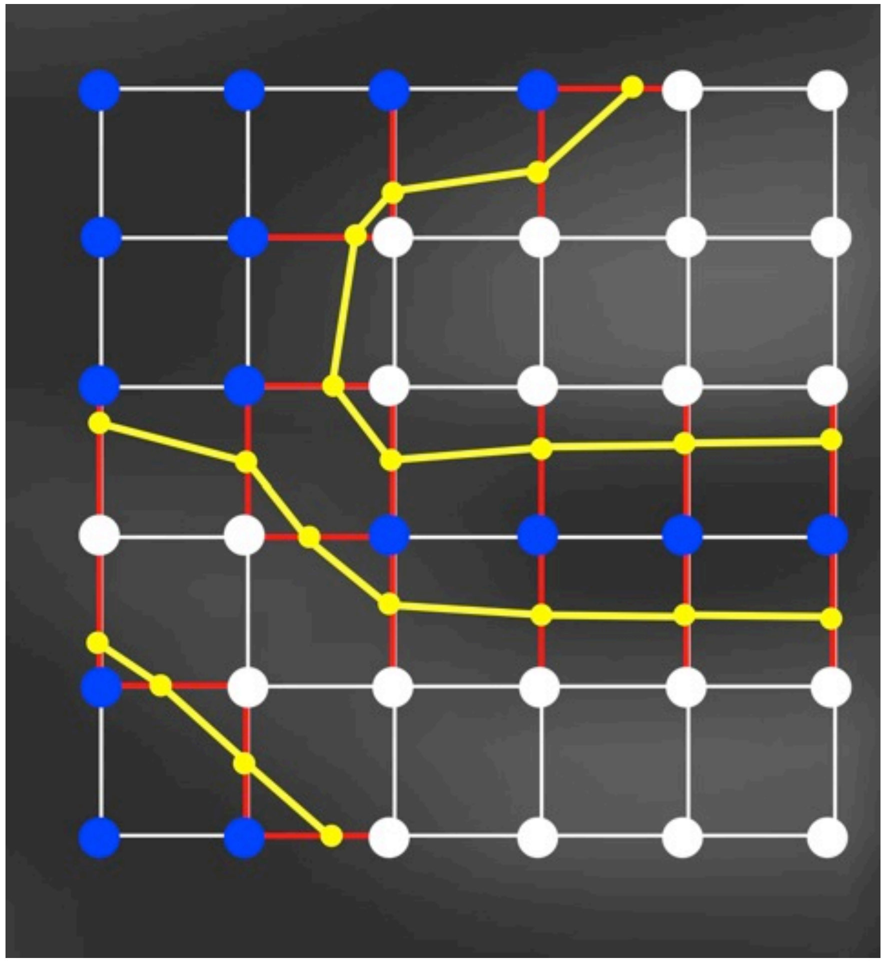
$$t = \frac{f(x_1)}{f(x_2) - f(x_1)}$$



Marching Squares (2D)

Connecting the intersections:

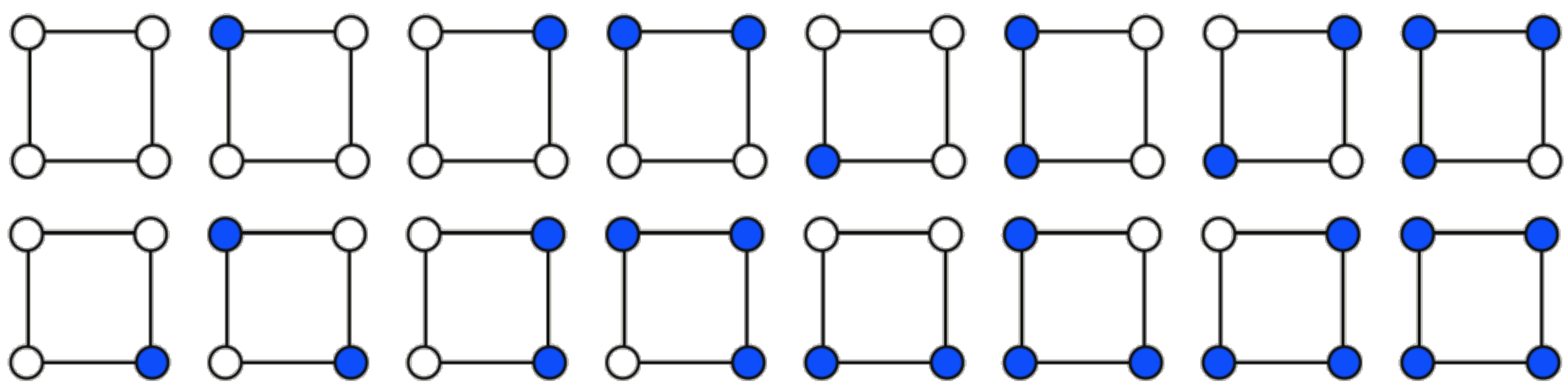
- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.



Marching Squares (2D)

Connecting the intersections:

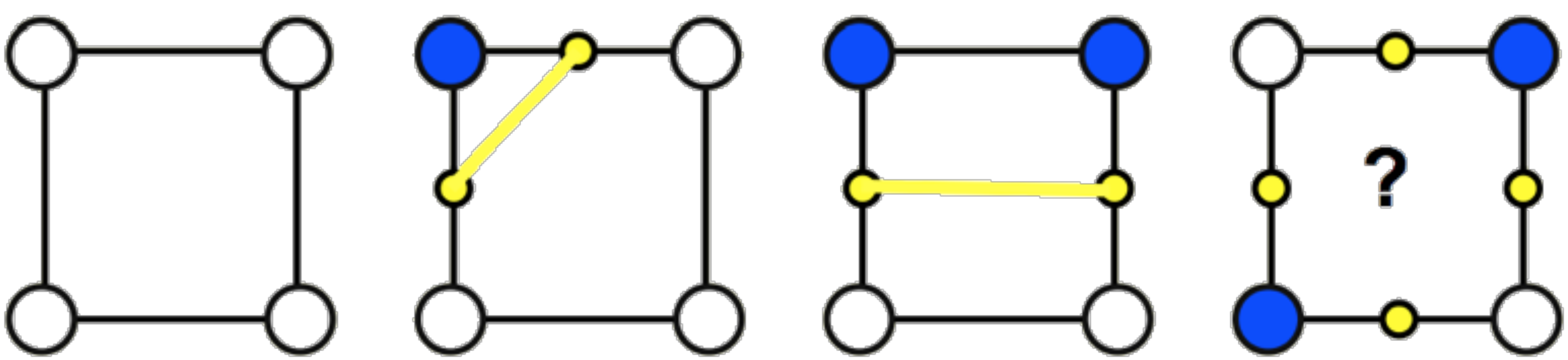
- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.
- Group those leading to the same intersections



Marching Squares (2D)

Connecting the intersections:

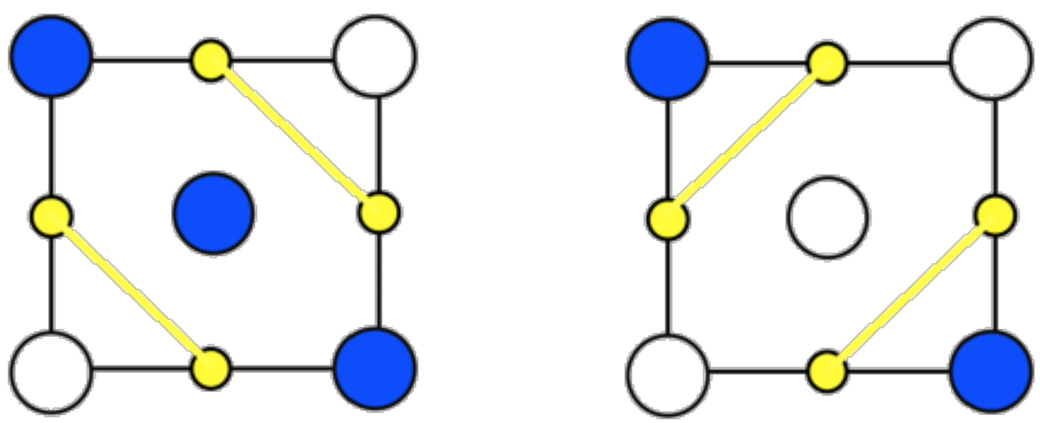
- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.
- Group those leading to the same intersections.
- Group equivalent after rotation.
- Connect intersections

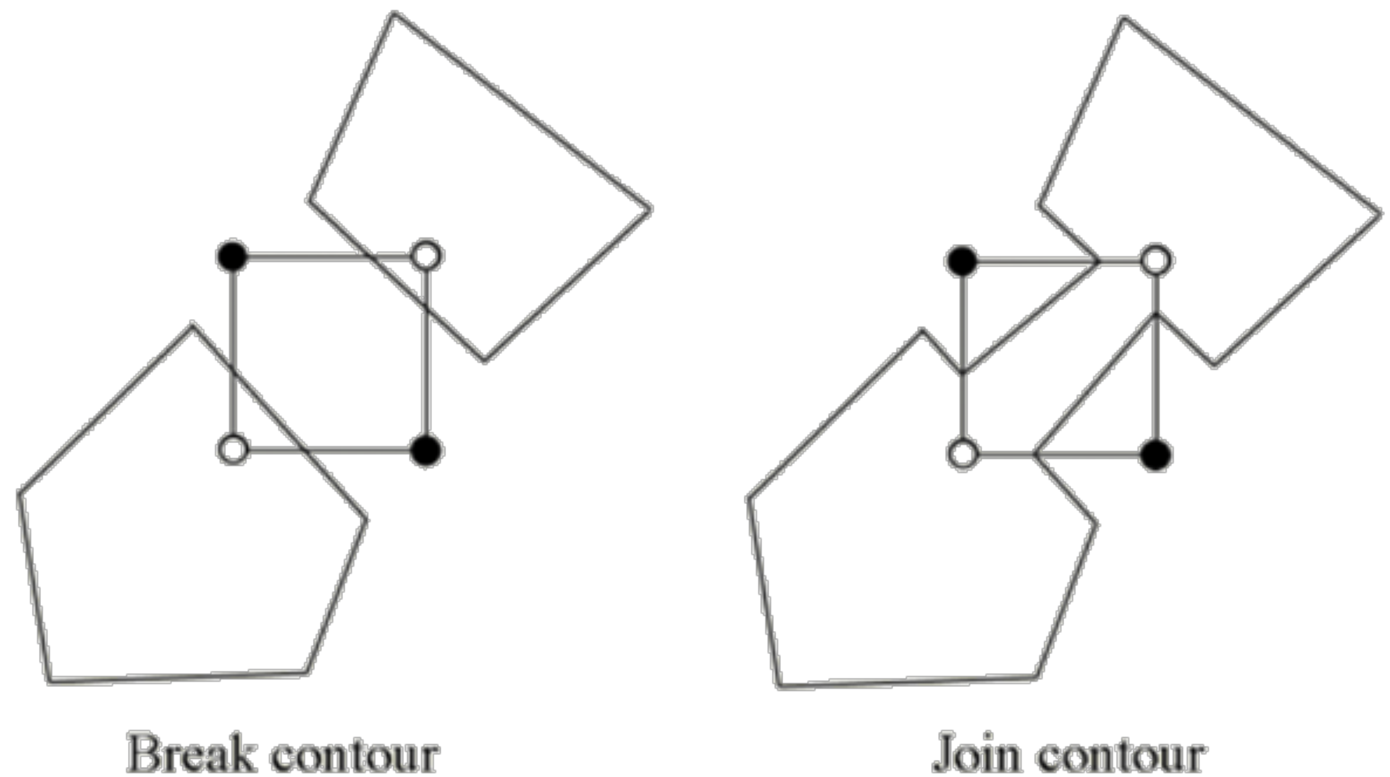


Marching Squares (2D)

Connecting the intersections:

Ambiguous cases:





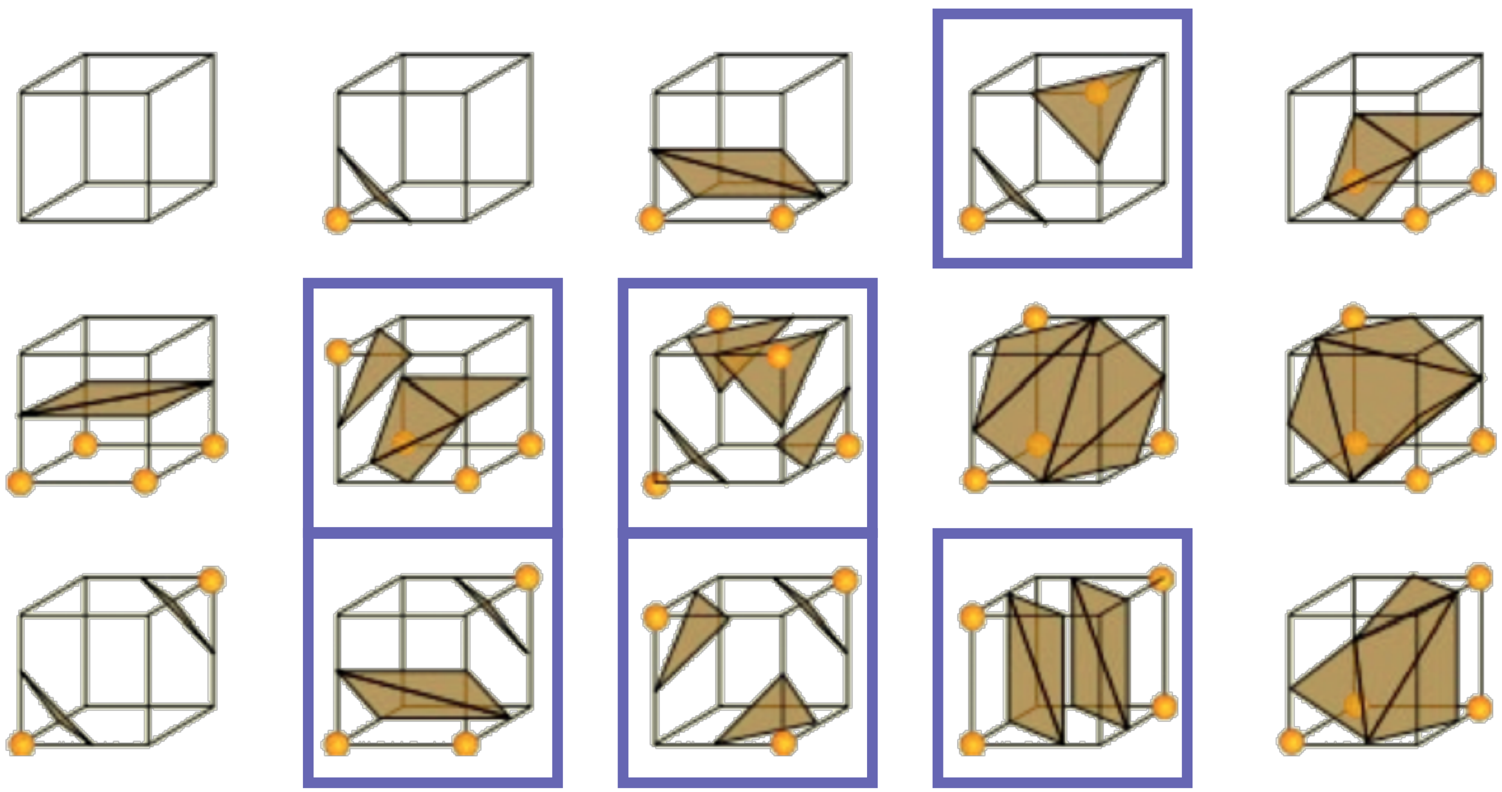
Two options:

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

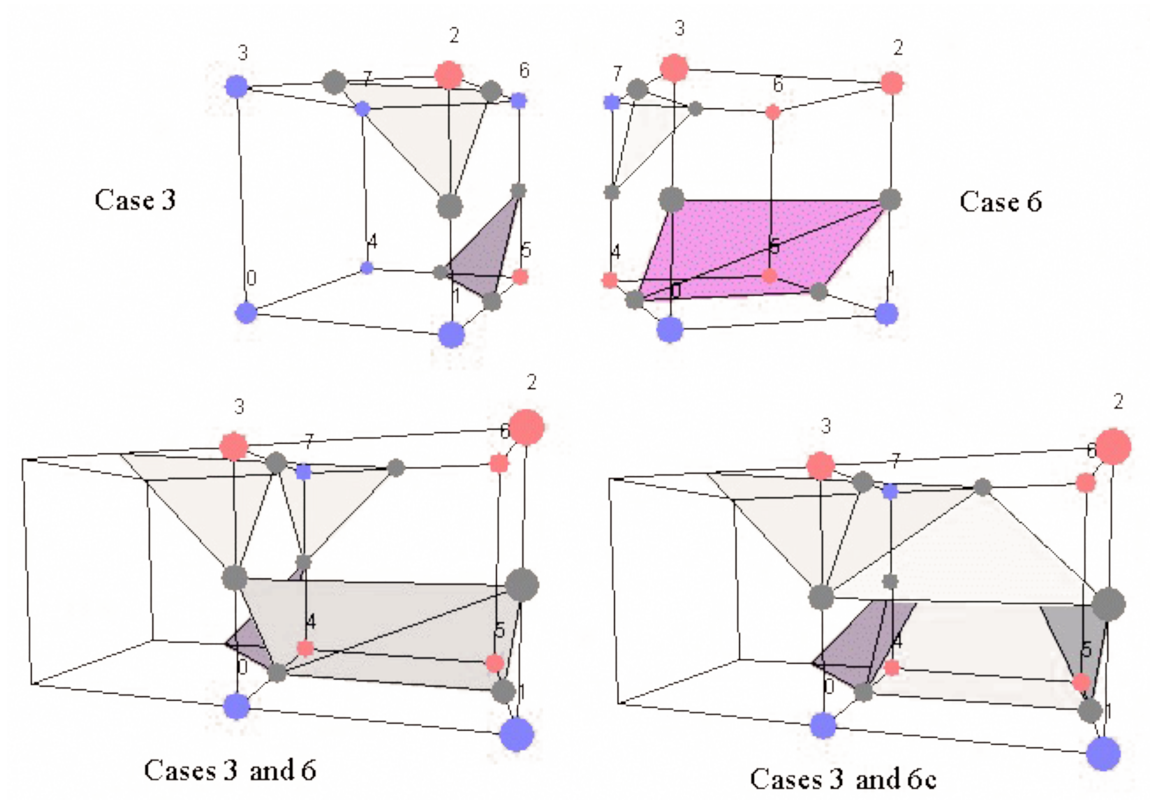
Marching Cubes (3D)

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

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



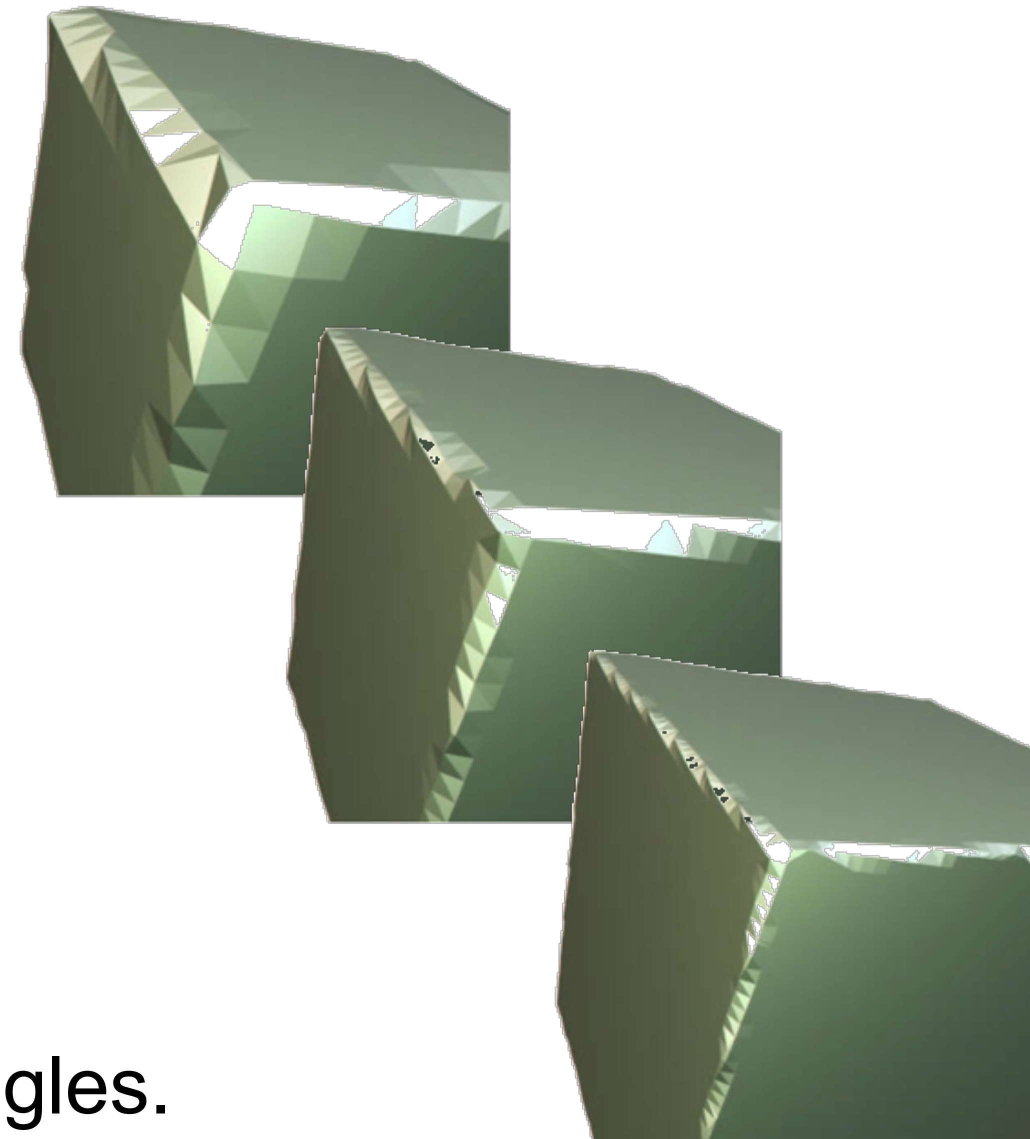
the 15 cases



explore ambiguity to avoid holes!

Chernyaev, Marching Cubes 33,’95

Marching Cubes (3D)



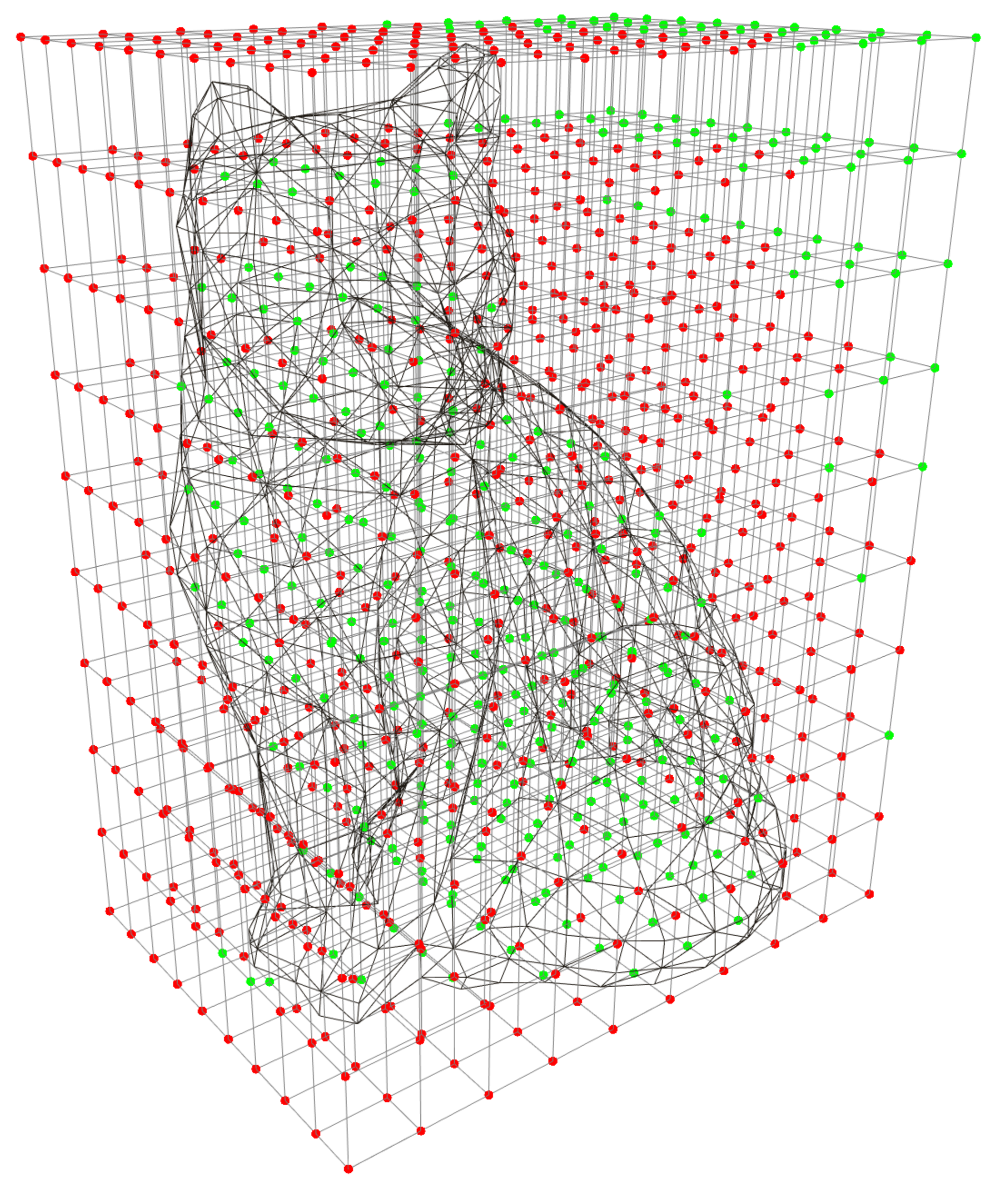
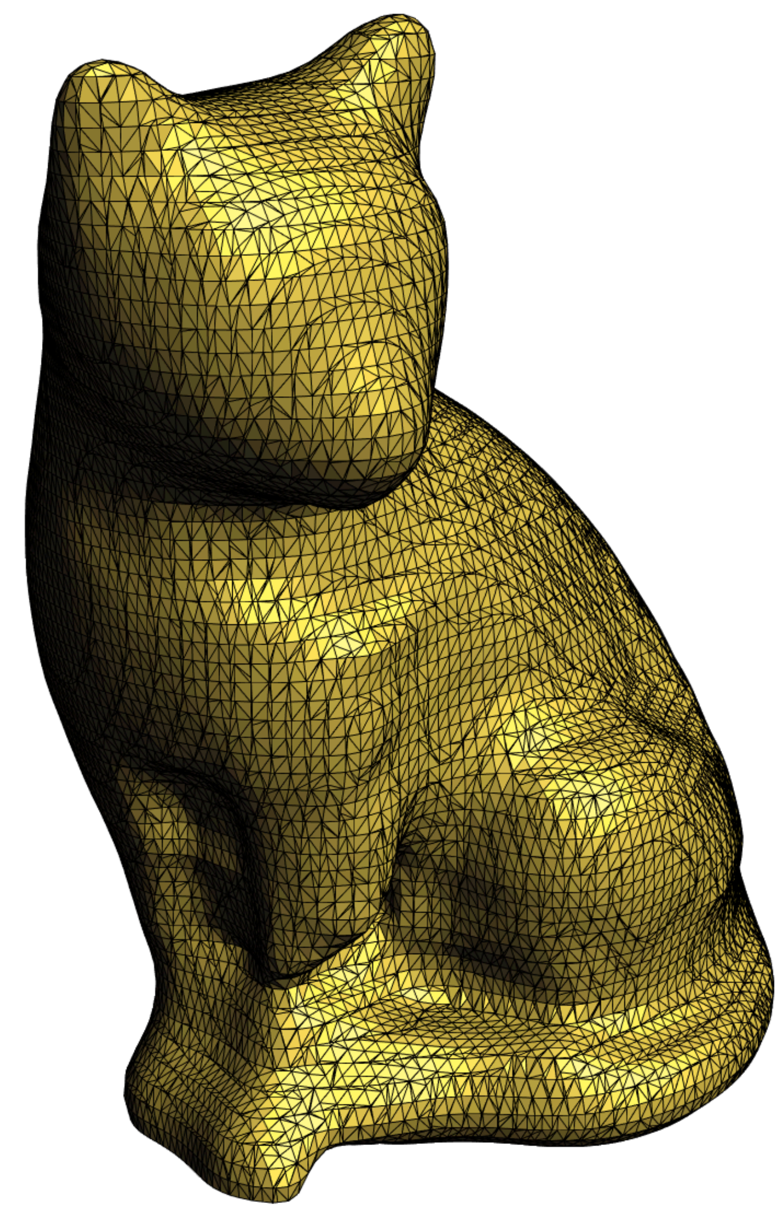
Main Strengths:

- 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

Next Time: Mesh → Point Cloud!

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

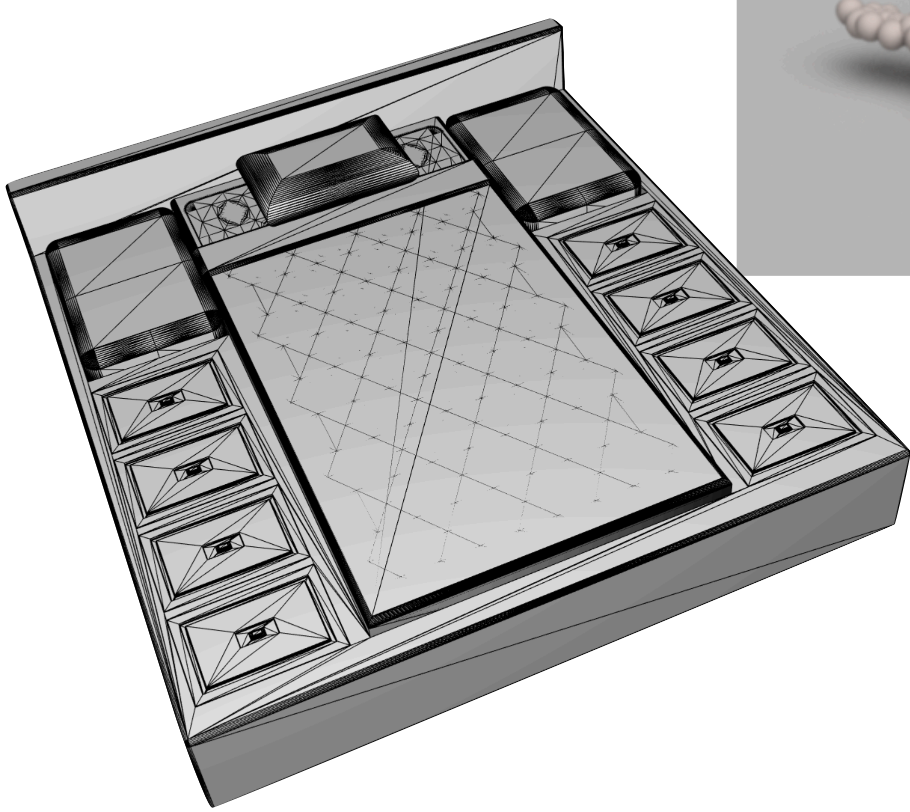
- Alec Jacobson's GP toolbox
  - https://github.com/alecjacobson/gptoolbox
  - MATLAB, various mesh and matrix routines
- Gabriel Peyre's Fast Marching Toolbox
  - https://www.mathworks.com/matlabcentral/fileexchange/6110-toolbox-fast-marching
  - On-surface distances (more next time!)
- OpenFlipper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

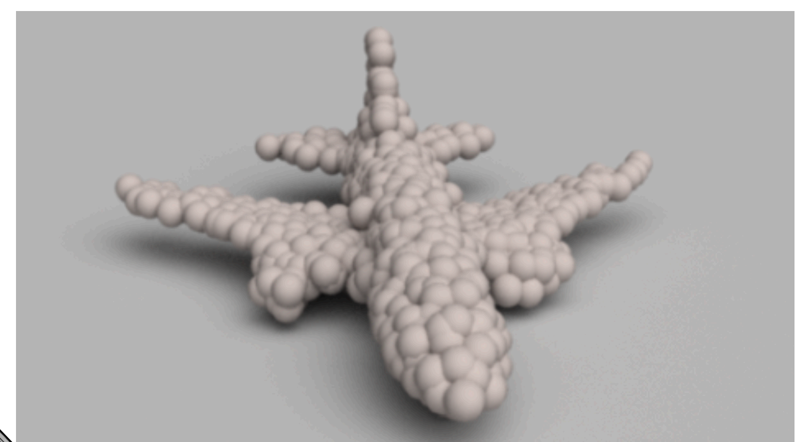
MESH-> POINT CLOUD

Sampling

From Surface to Point Cloud - Why?

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

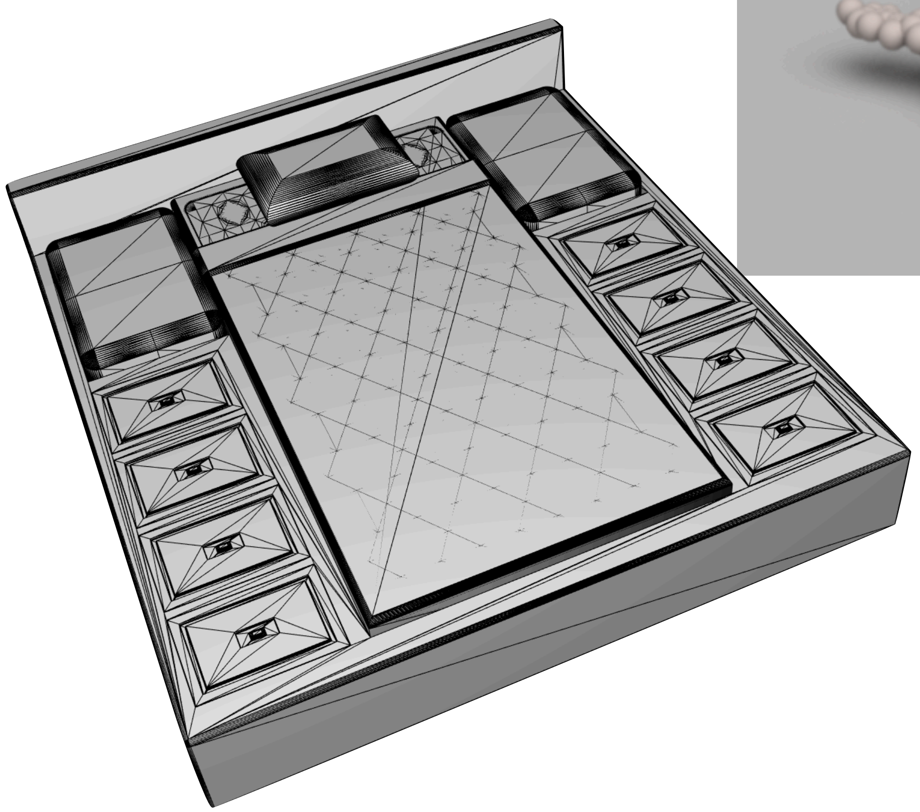


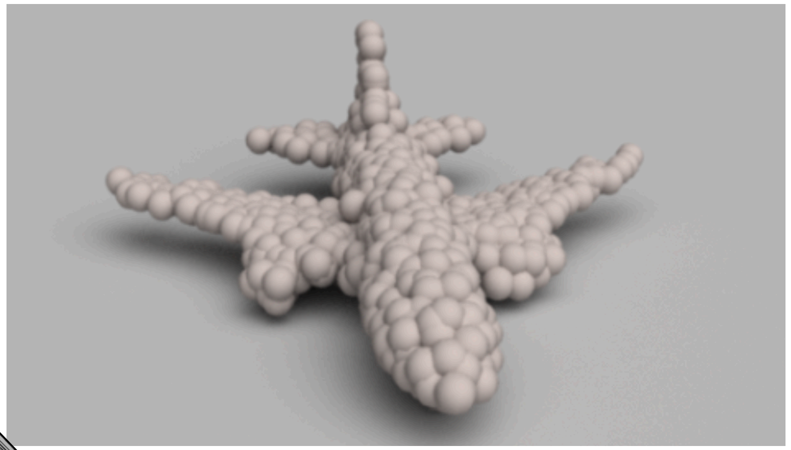


CAD meshes:
many components
bad triangles
connectivity problems

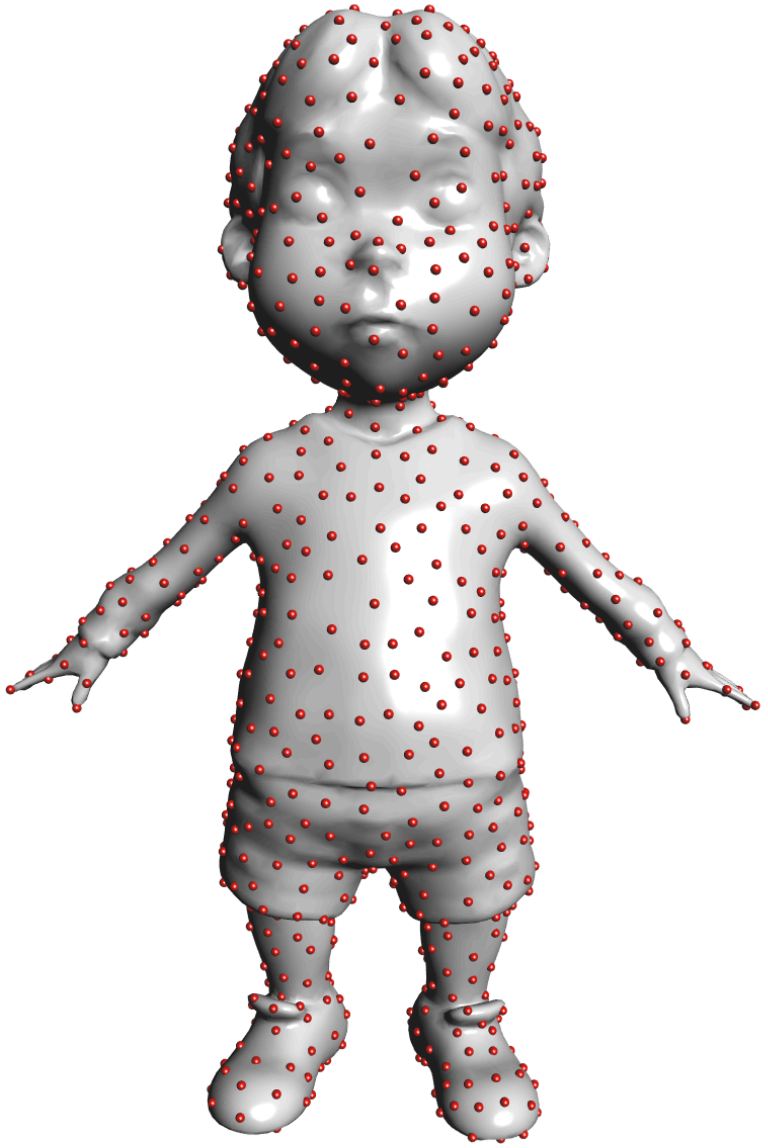
From Surface to Point Cloud - Why?

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





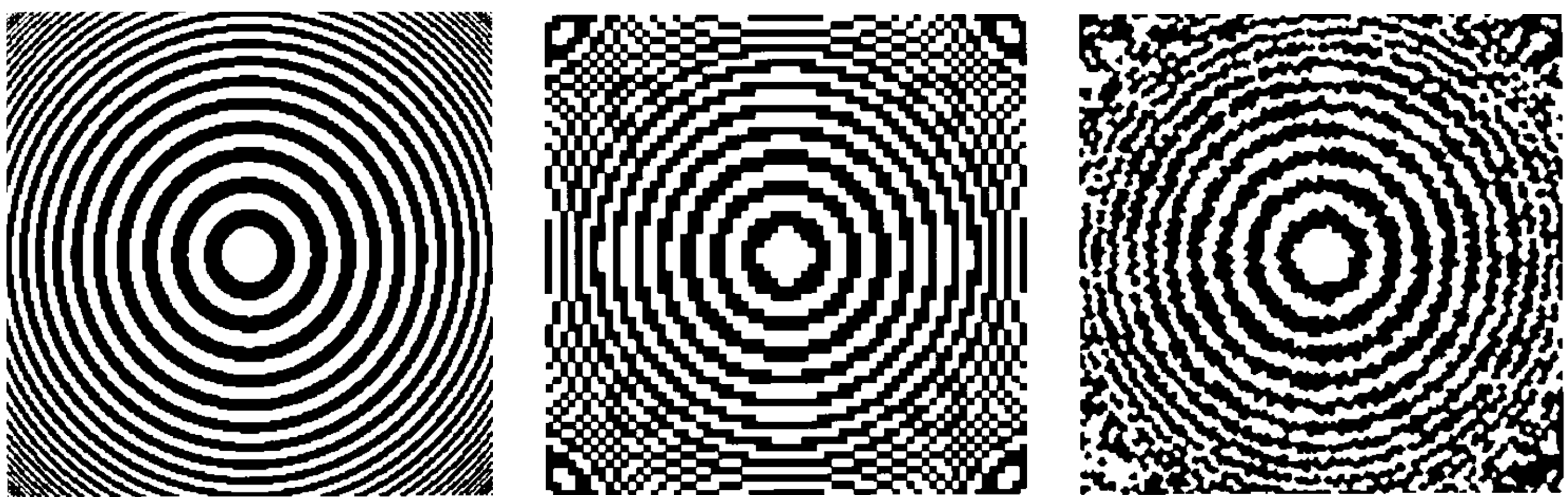
CAD meshes:
many components
bad triangles
connectivity problems



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

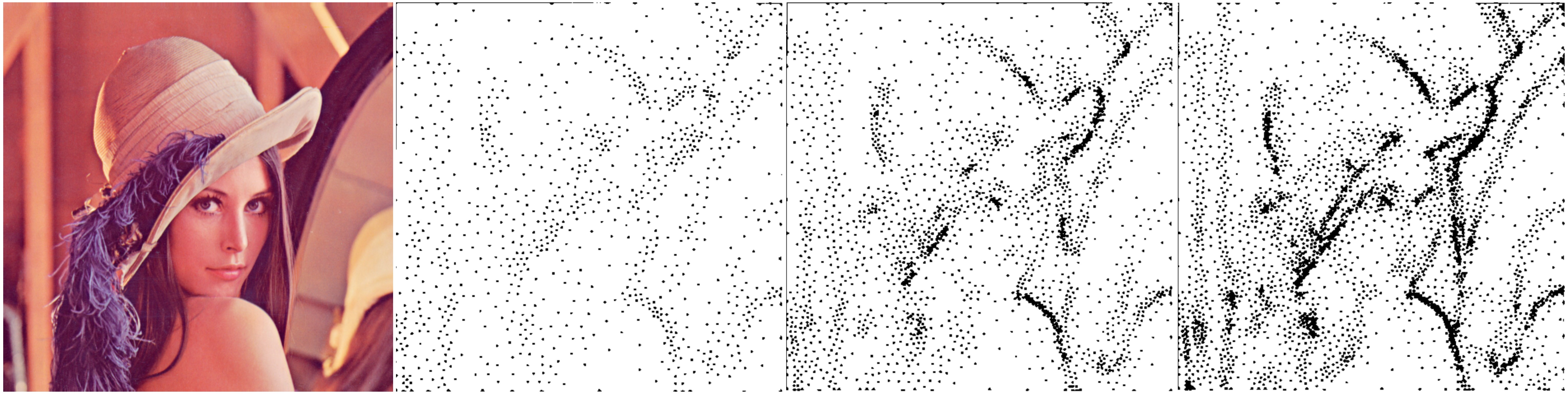
1. Create an initial sample point set *S*
   - Image corners + additional random point.
2. Find the point which is the farthest from all point in *S*

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

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

Farthest Point Sampling

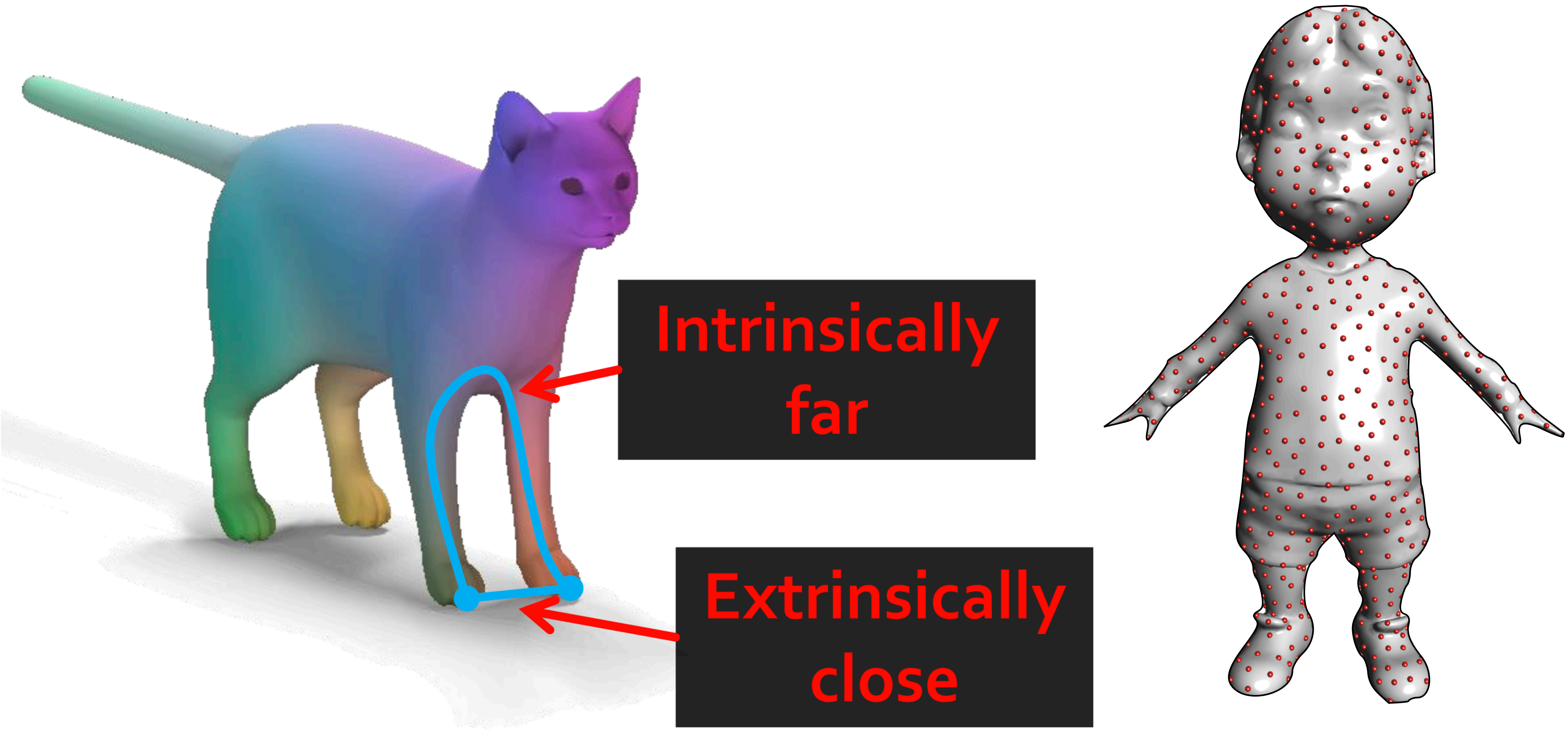
- Depends on a notion of distance on the sampling domain
- Can be made adaptive, via a weighted distance



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

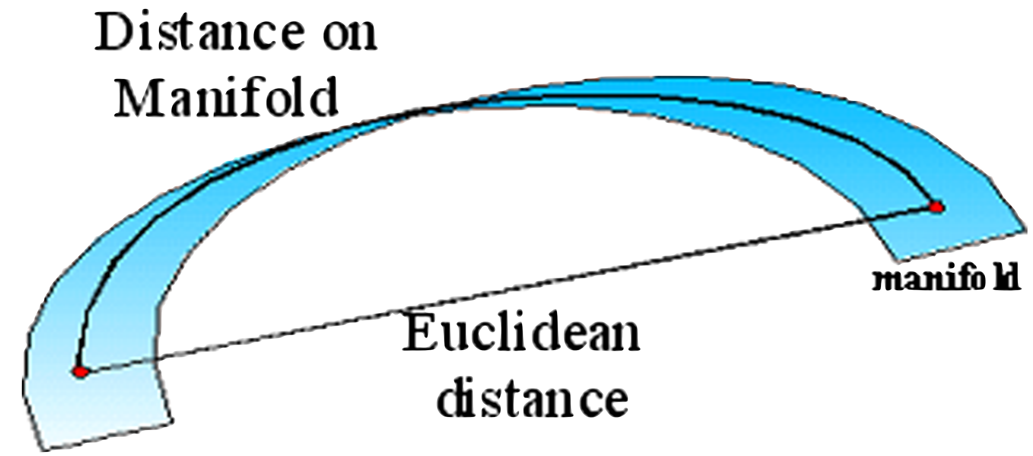
FPS on surfaces

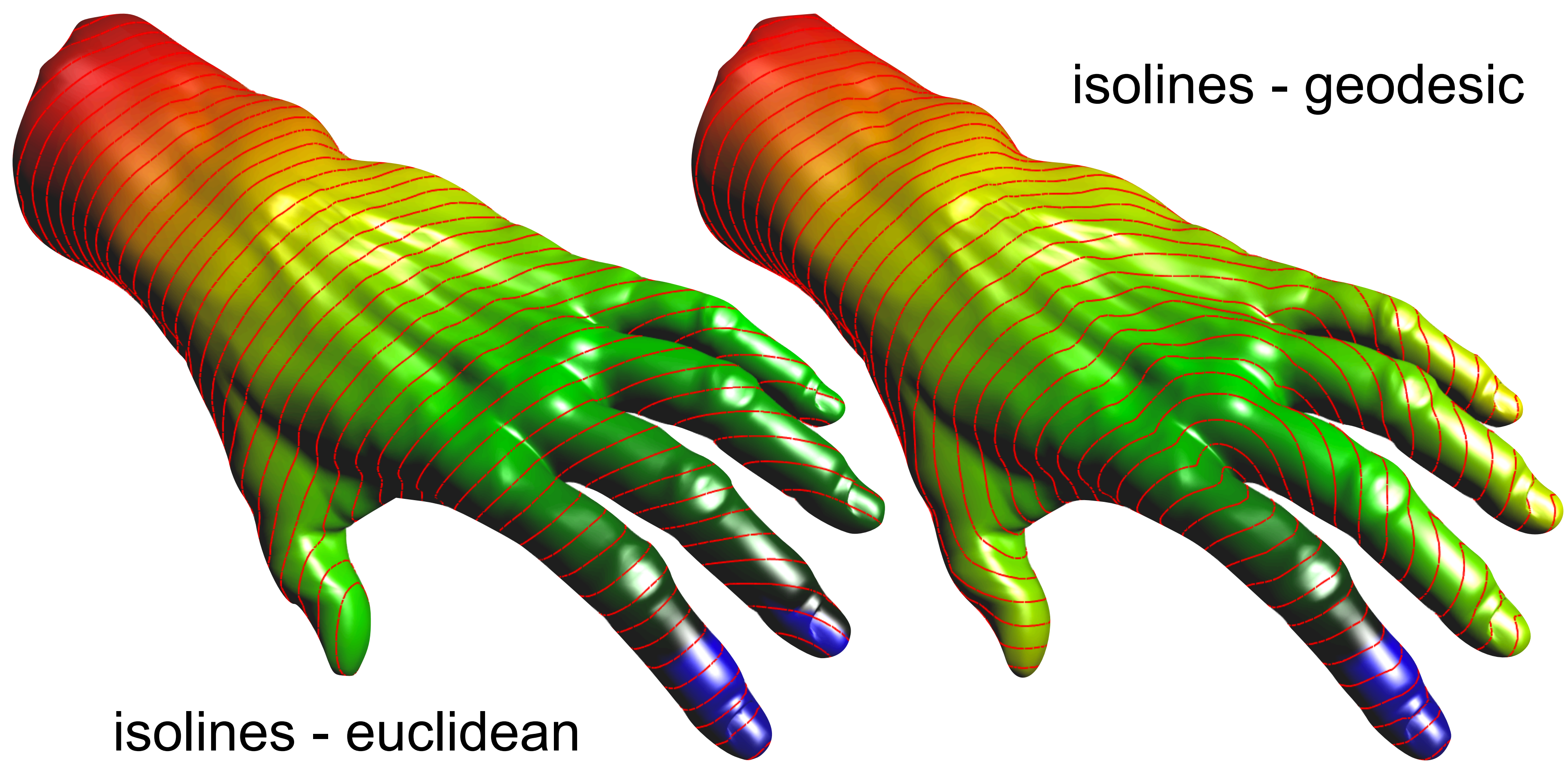
- What's an appropriate distance?



On-Surface Distances

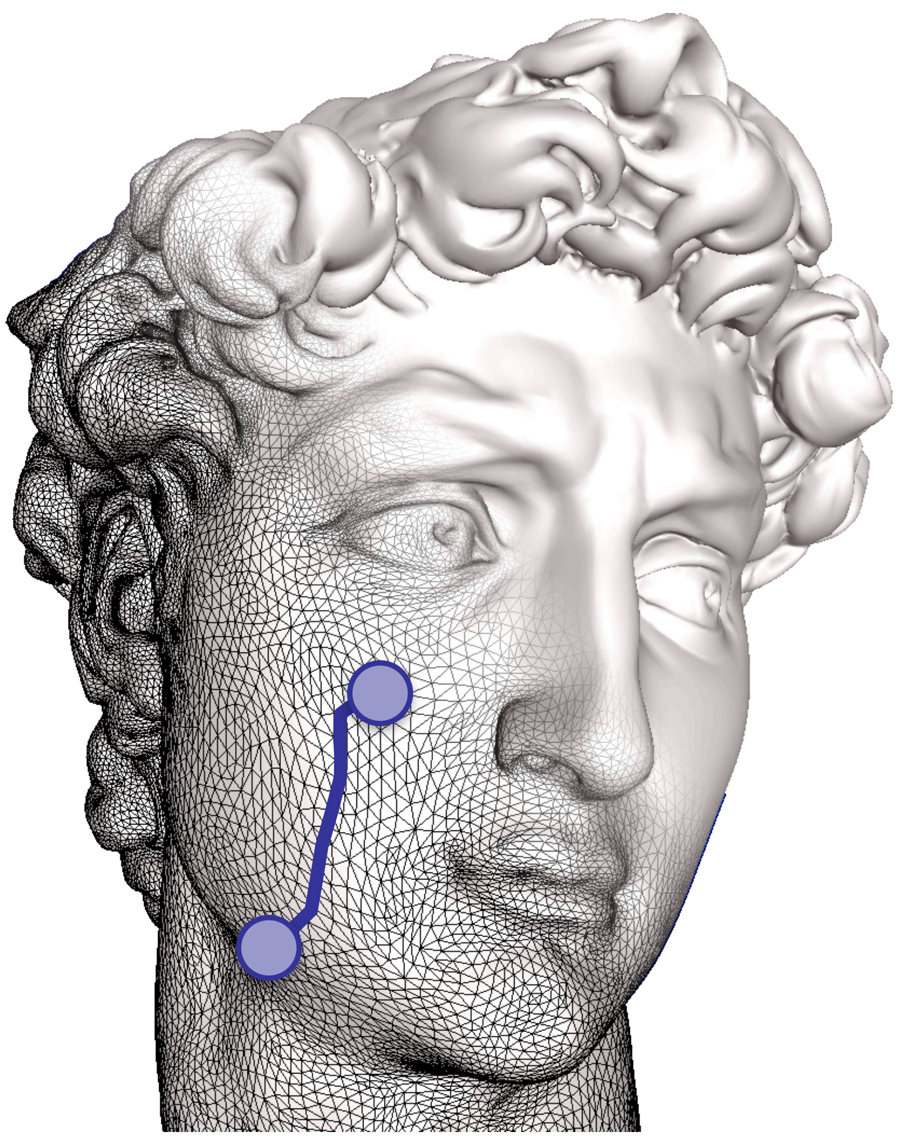
- Geodesics: Straightest and **locally shortest** curves





Discrete Geodesics

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

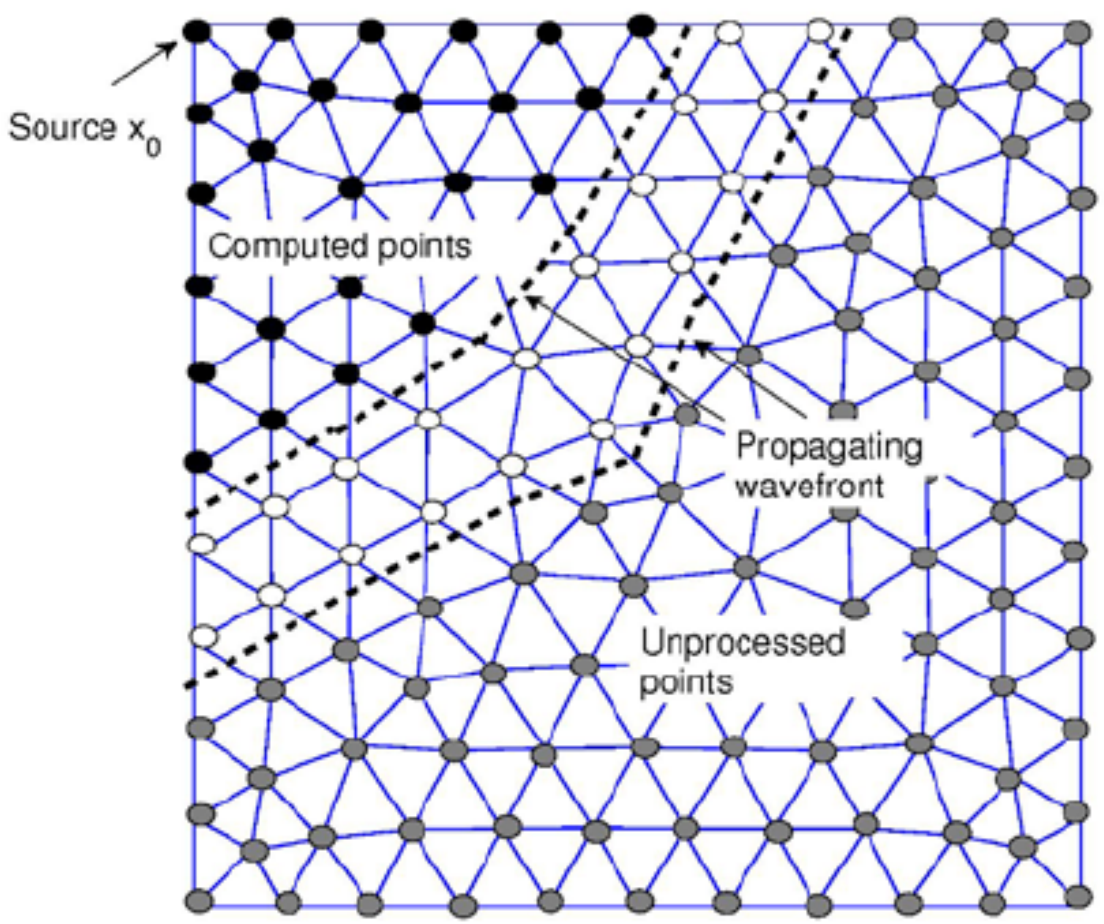


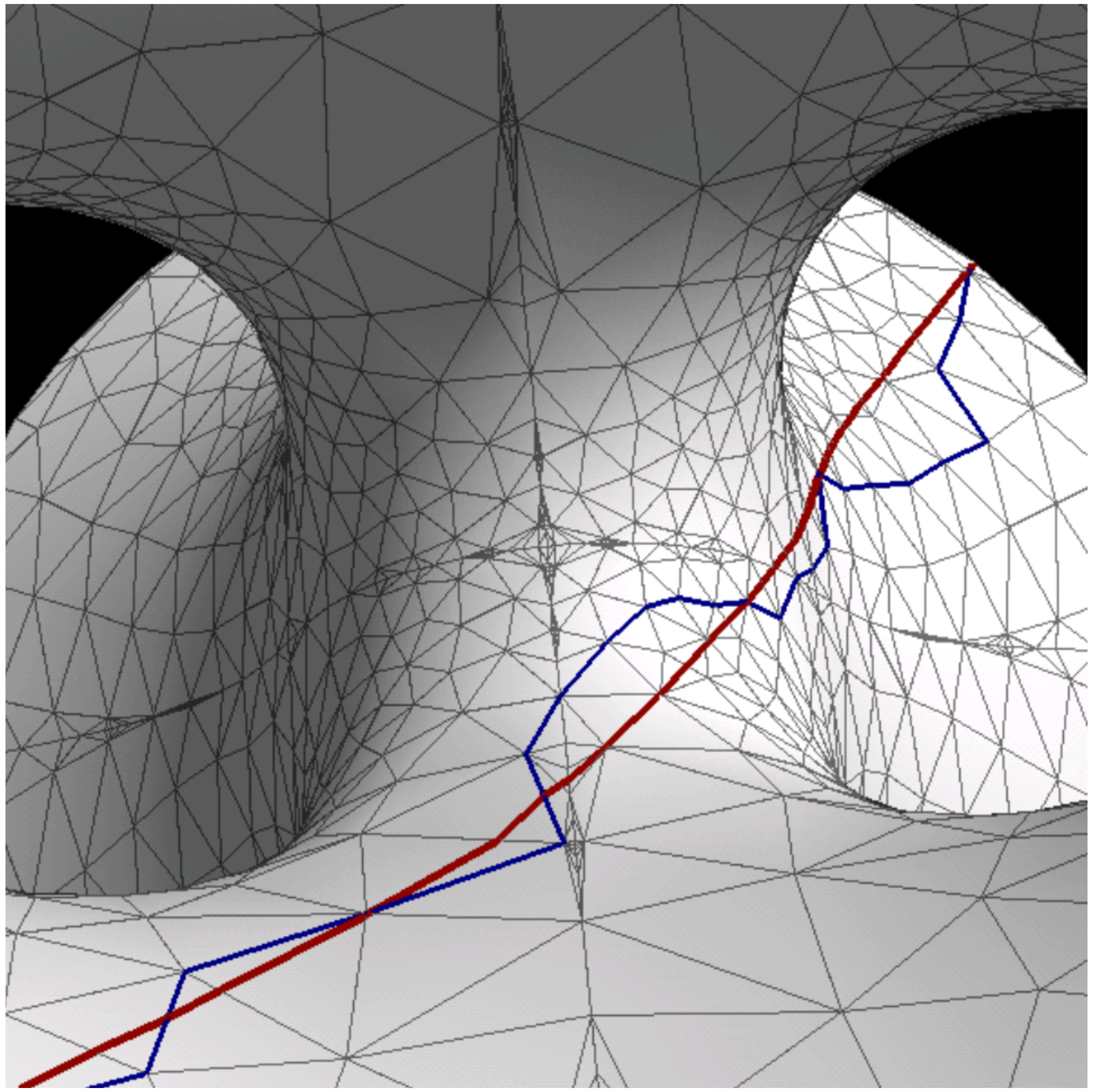
```
v_0 = initial vertex
d_i = current distance to vertex i
S = verticies with known optimal distance

# 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!

Fast Marching Geodesics

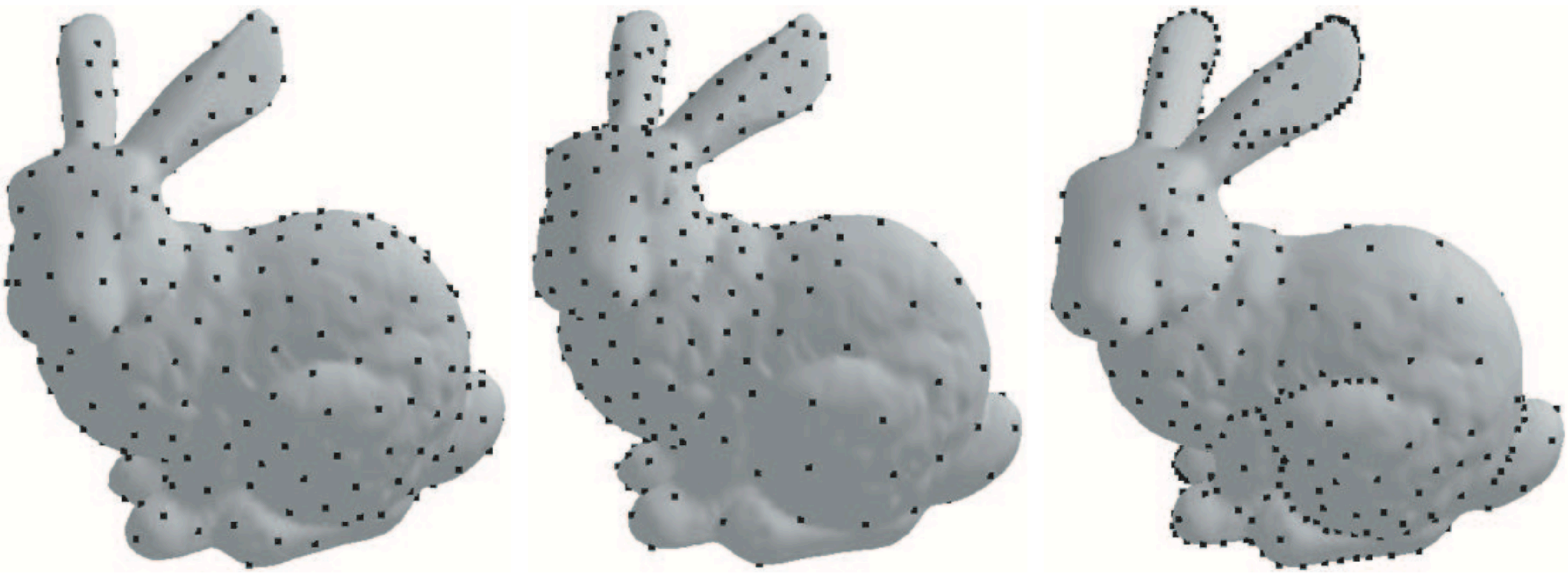
- A better approximation: allow fronts to cross triangles!





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

FPS on a Mesh



Peyré and Cohen 2003, Geodesic Remeshing Using Front Propagation