UC San Diego

Lecture 5:

Geometry Foundations

Instructor: Hao Su

Jan 23, 2018

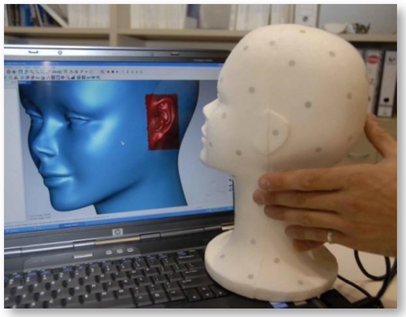
slides credits: Olga Diamanti, Olga Sorkine-Hornung, Daniele Panozzo, ETH Zurich, Maks Ovsjanikov, Mario Botsch

Agenda

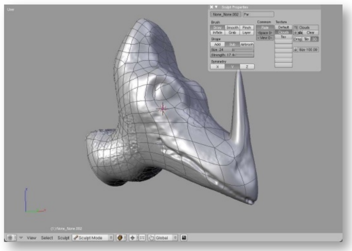
- Machine Learning on Extrinsic Geometry (3 weeks)
  - Overview of 3D Representations
  - Geometric foundation
  - Machine Learning on Different 3D Representations
    - Volumetric
    - Multi-view
    - Point cloud
    - Parametric

Shape Representation: Origin- and Application-Dependent

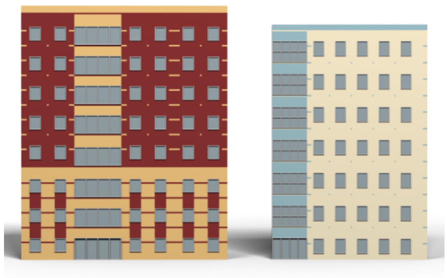
- Acquired real-world objects:

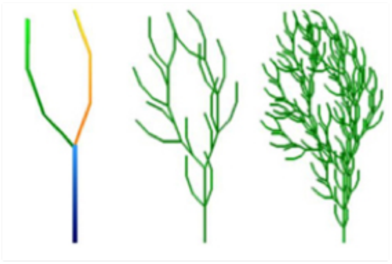


- Modeling "by hand":



- Procedural modeling

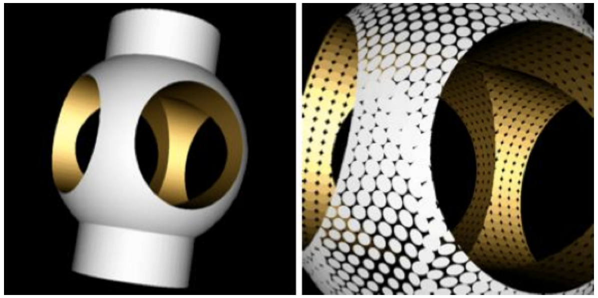


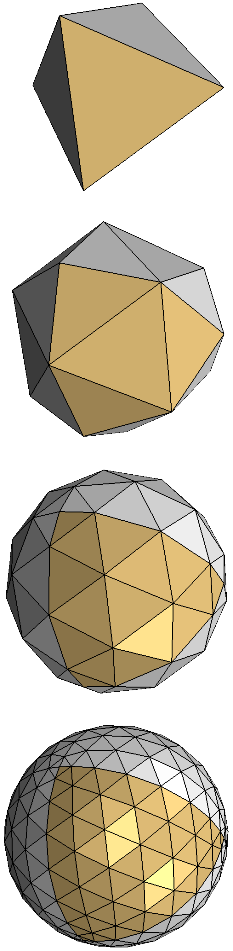


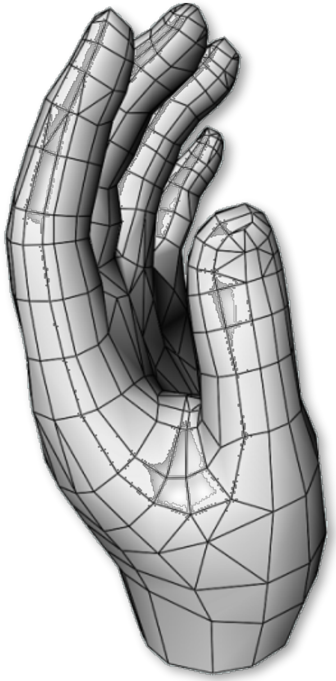
- …

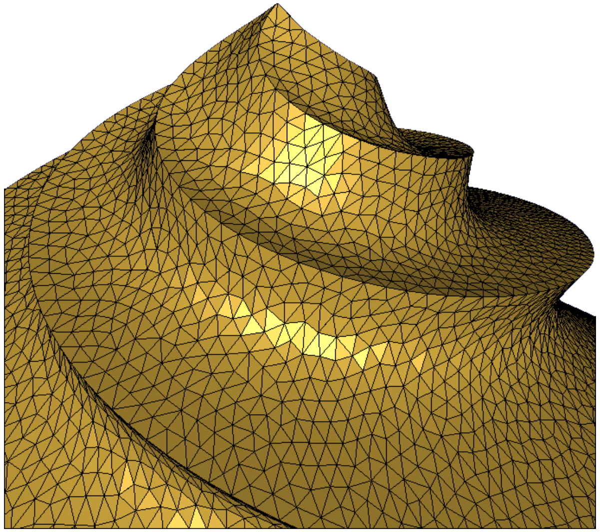
Shape Representations

- Points
- Polygonal meshes



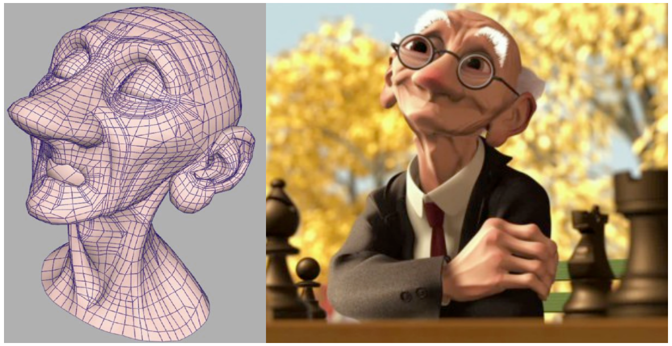


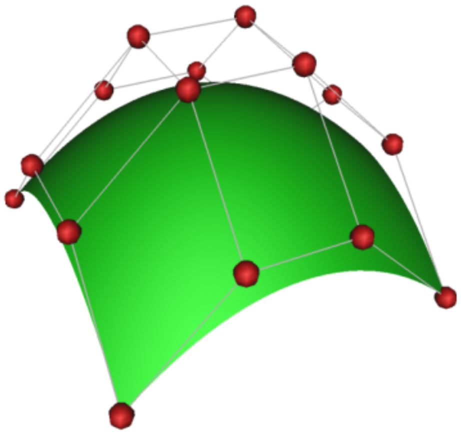


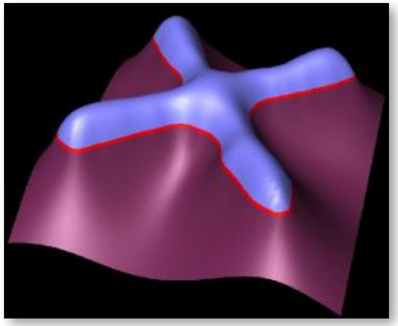


Shape Representations

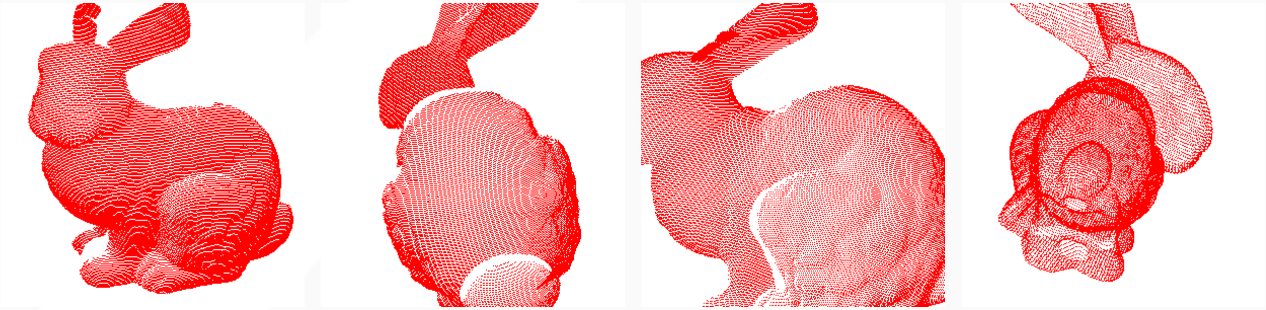
- Parametric surfaces
- Implicit functions
- Subdivision surfaces



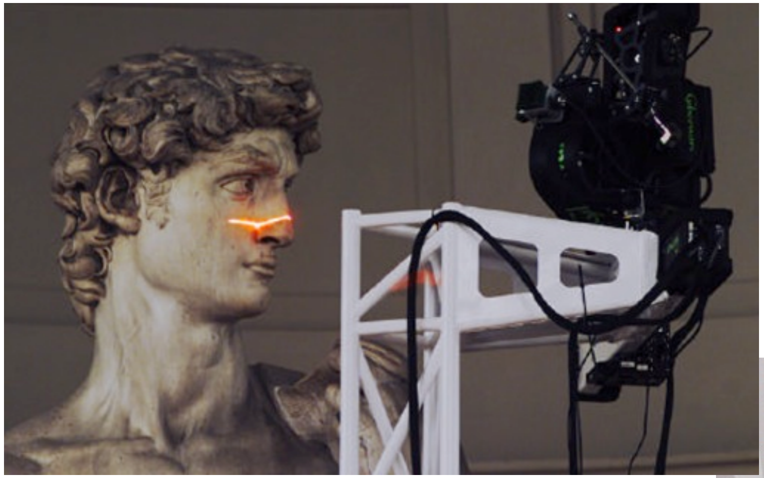


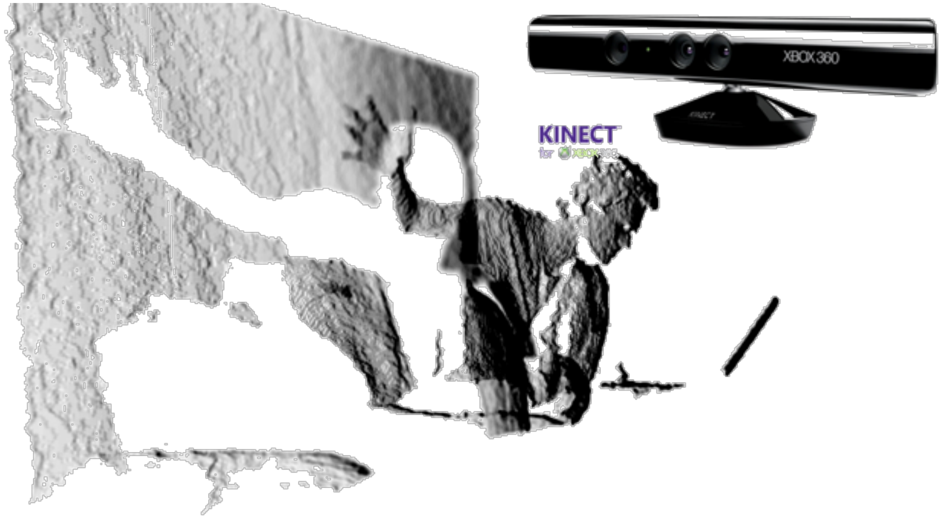


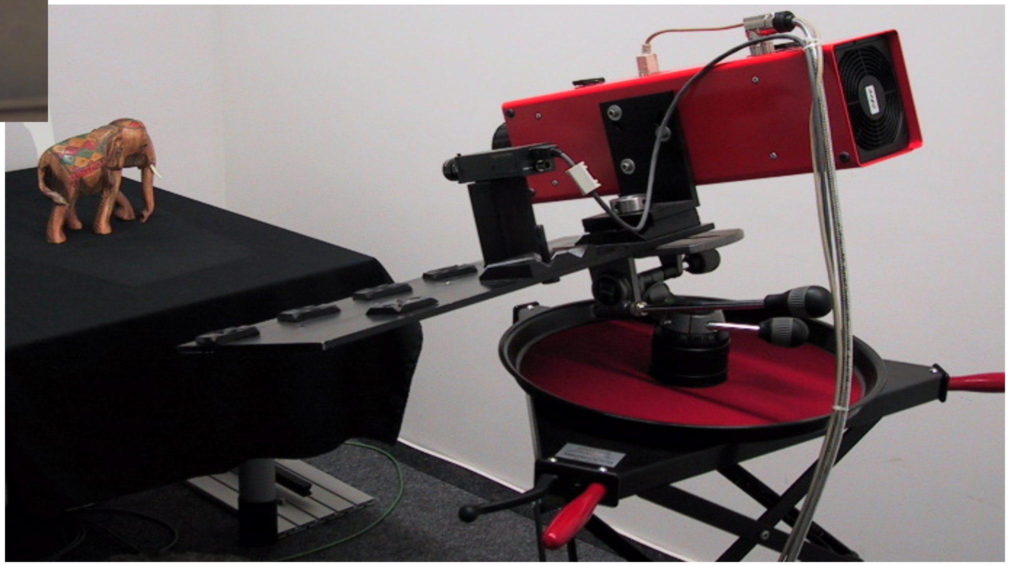
POINTS



Output of Acquisition

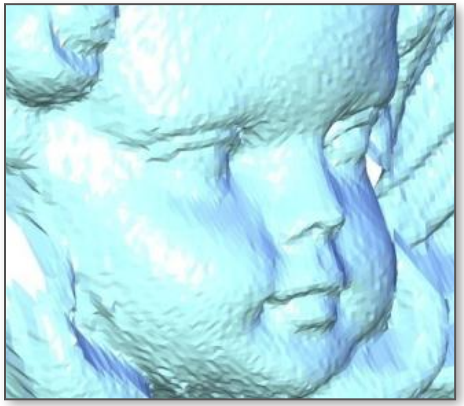


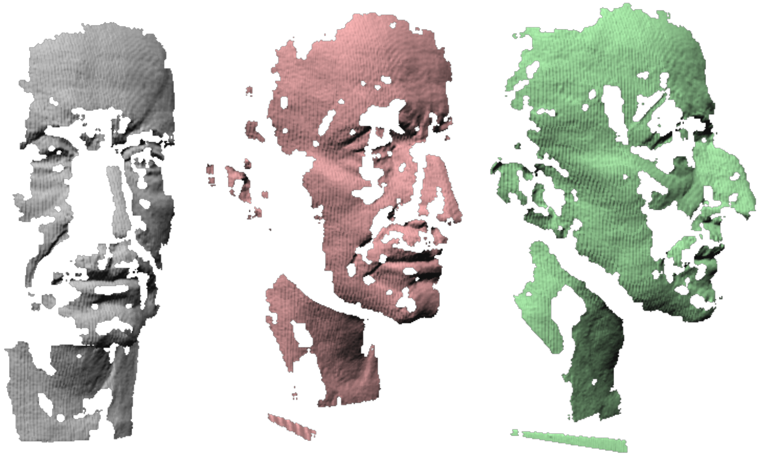




Points

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

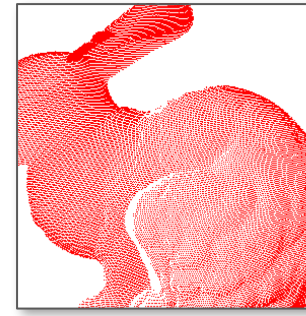




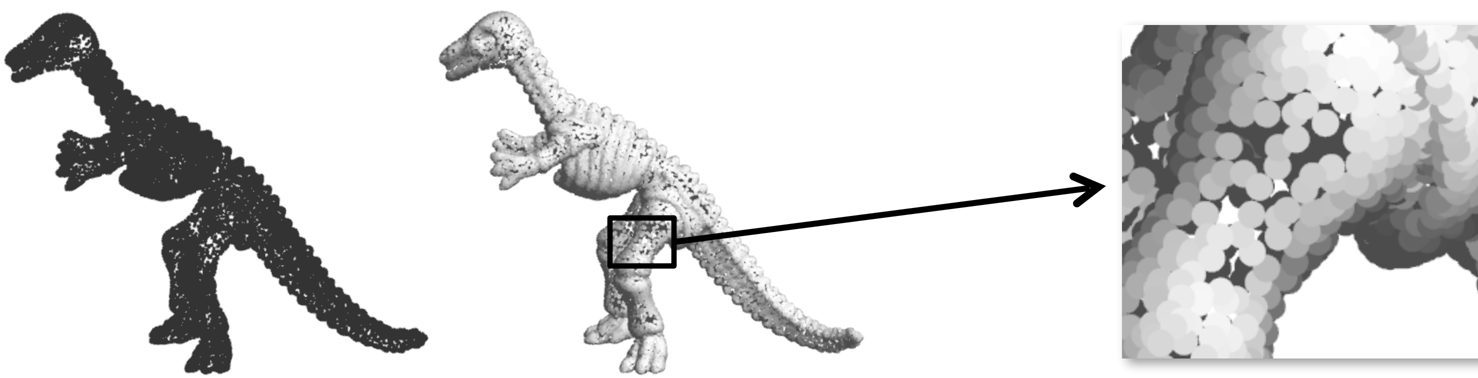
set of raw scans

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

Points

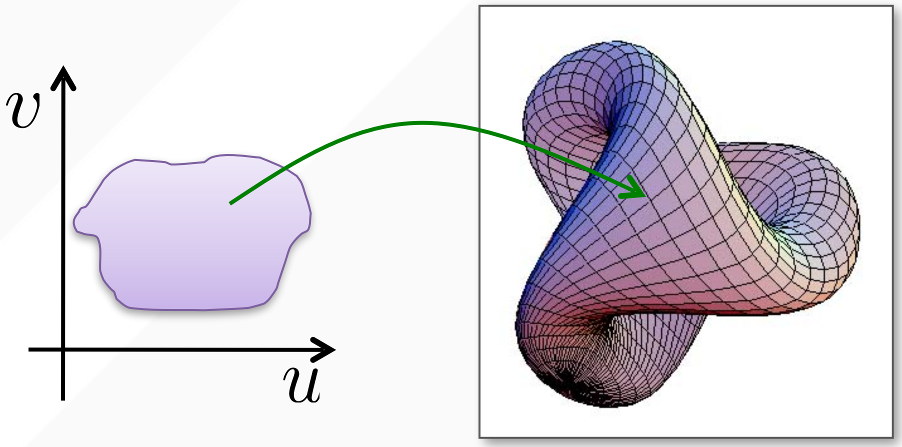


- Points = unordered set of 3-tuples
- Often converted to other reps
  - Meshes, implicits, parametric surfaces
  - Easier to process, edit and/or render
- Efficient point processing / modeling requires spatial partitioning data structure
  - Eg. **to figure out neighborhoods**



shading needs normals!

PARAMETRIC CURVES AND SURFACES



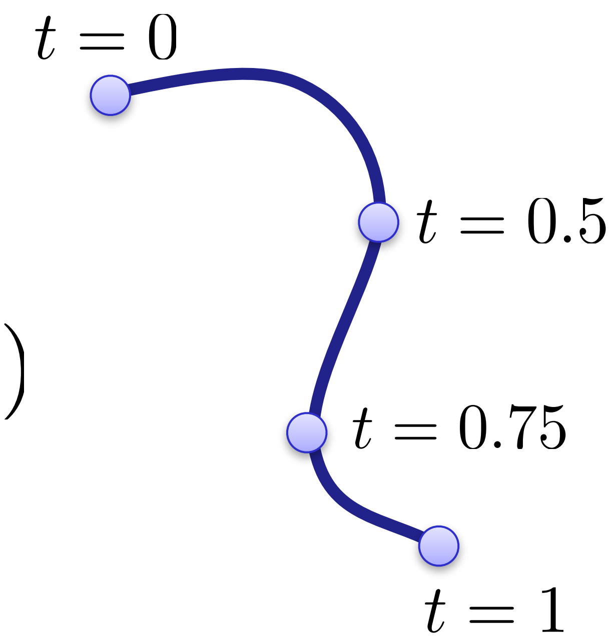
Parametric Representation

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

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

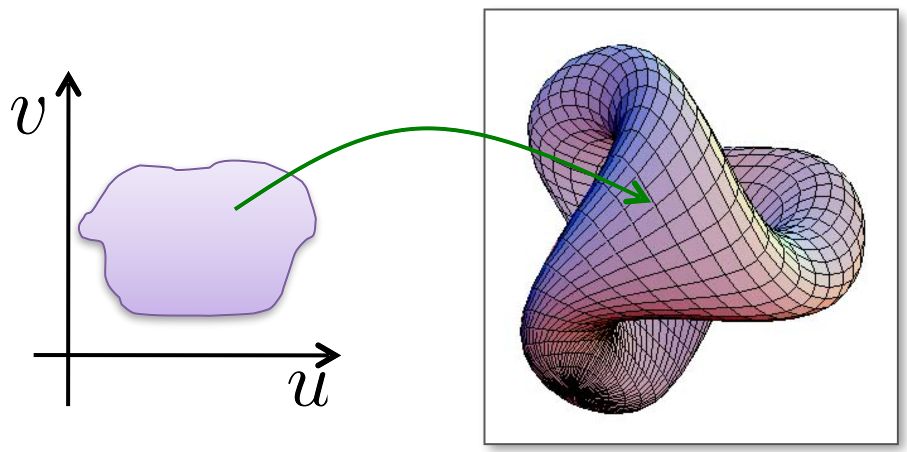
  - Space curve: $m = 1, n = 3$

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



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

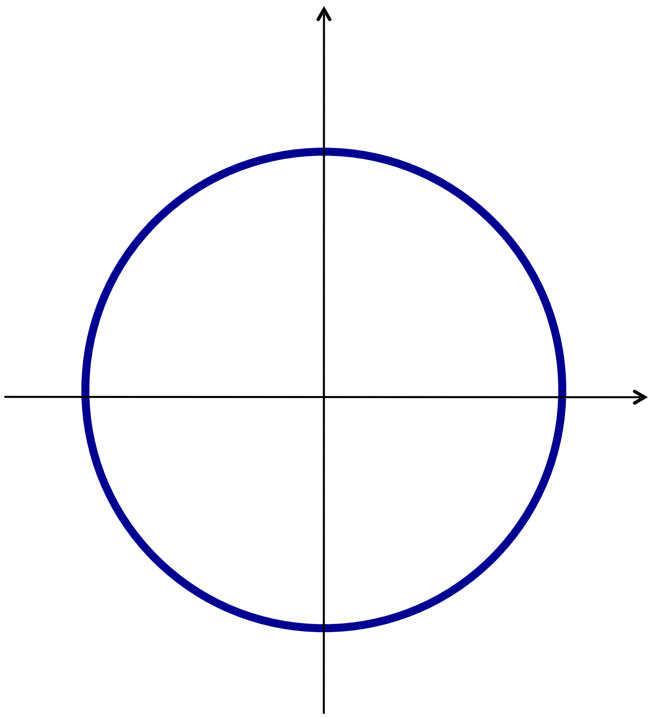
- Example: Explicit curve/circle in 2D

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

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

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

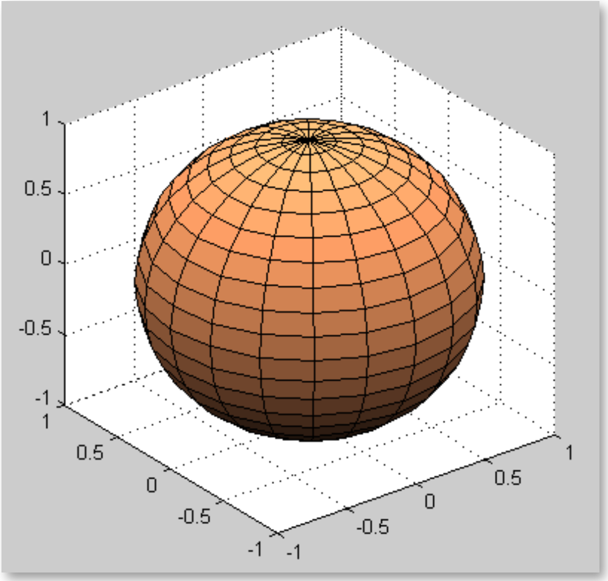
$$t \in [0, 2\pi)$$



Parametric Surfaces

- Sphere in 3D

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



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

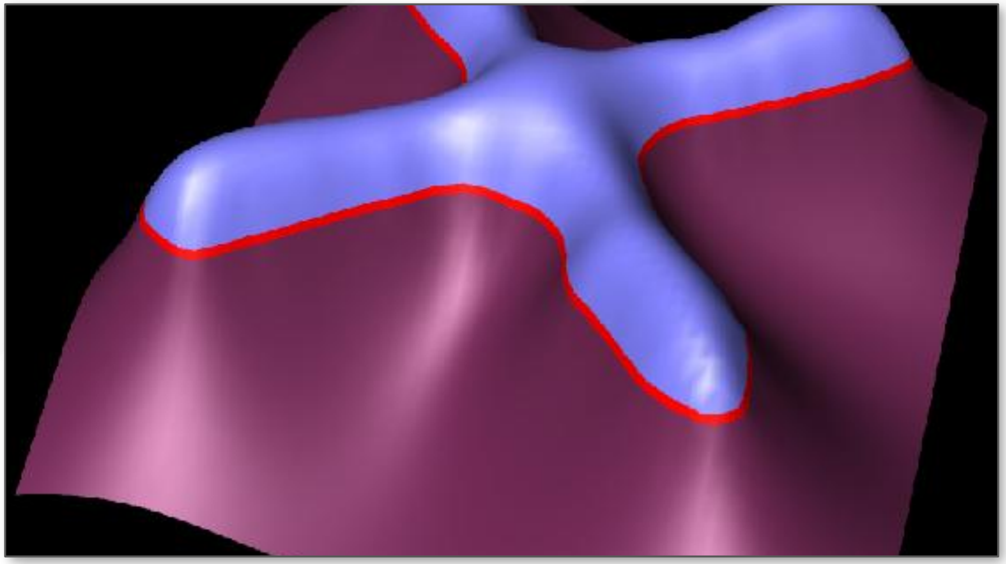
$$(u, v) \in [0, 2\pi) \times [-\pi/2, \pi/2]$$

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

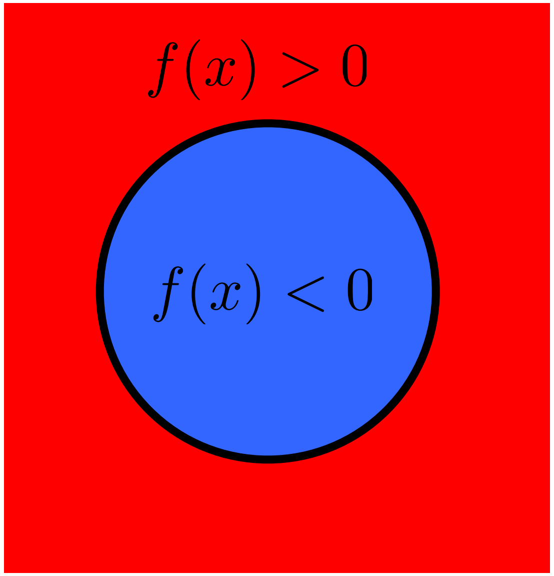
- Kernel of a scalar function $f : \mathbb{R}^m \to \mathbb{R}$
  - Curve in 2D: $S = \{x \in \mathbb{R}^2 | f(x) = 0\}$
  - Surface in 3D: $S = \{x \in \mathbb{R}^3 | f(x) = 0\}$

- Space partitioning

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

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

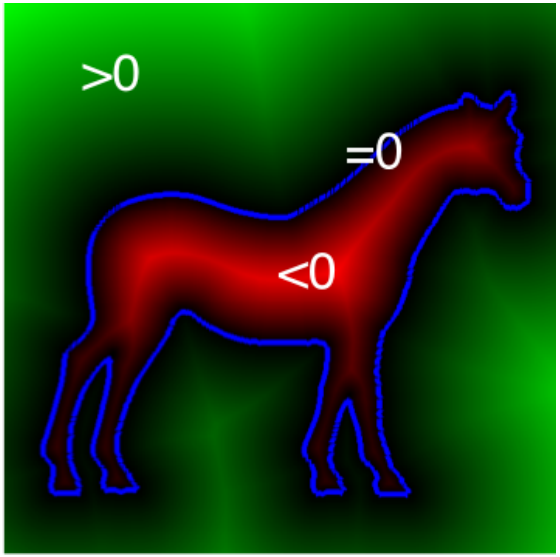
$\{x \in \mathbb{R}^m | f(x) < 0\}$ Inside



Implicit Curves and Surfaces

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

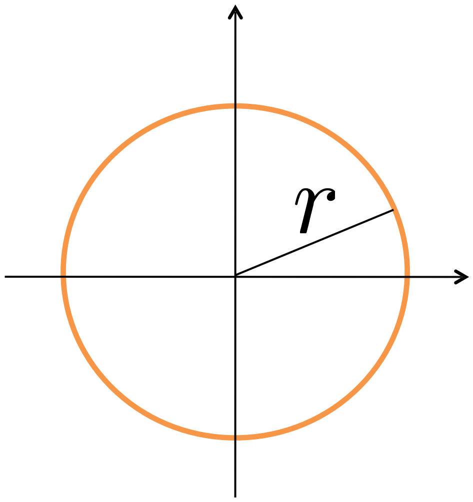
- Zero level set of signed distance function



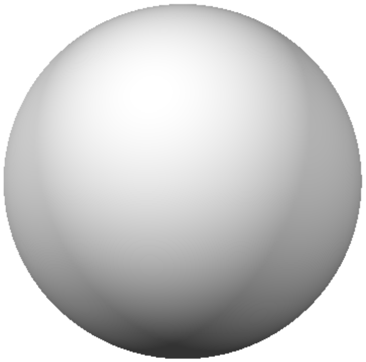
Implicit Curves and Surfaces

- Implicit circle and sphere

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

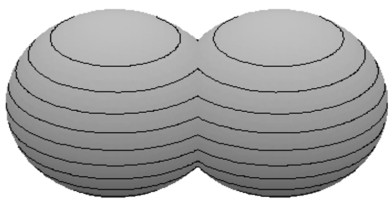


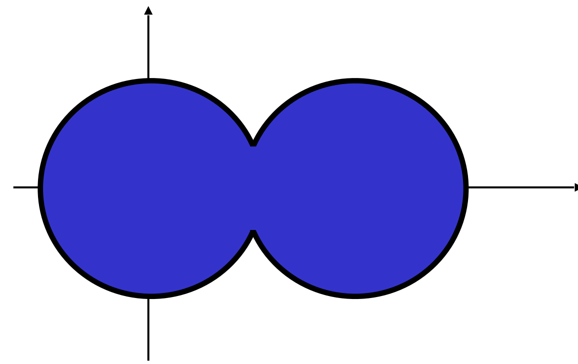
$$f(x, y, z) = x^2 + y^2 + z^2 - r^2$$



Boolean Set Operations

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





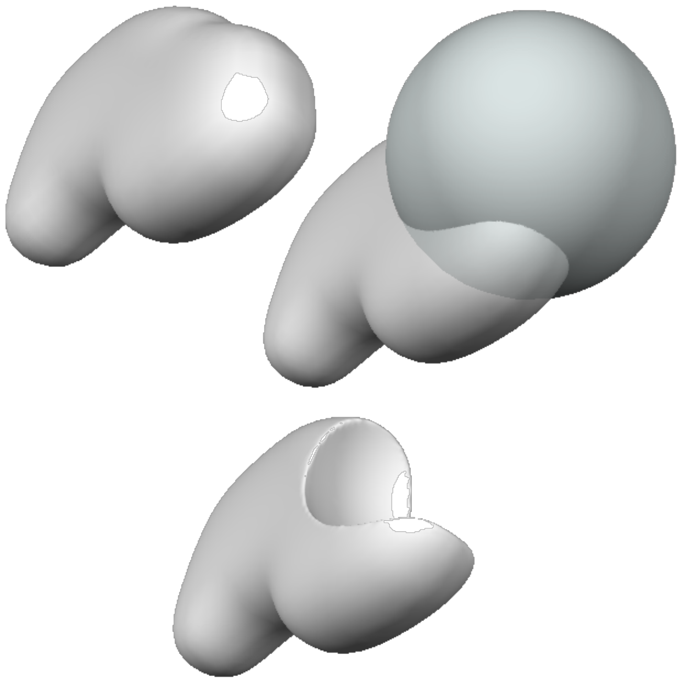
- Intersection: $$\bigcap_i f_i(x) = \max f_i(x)$$

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

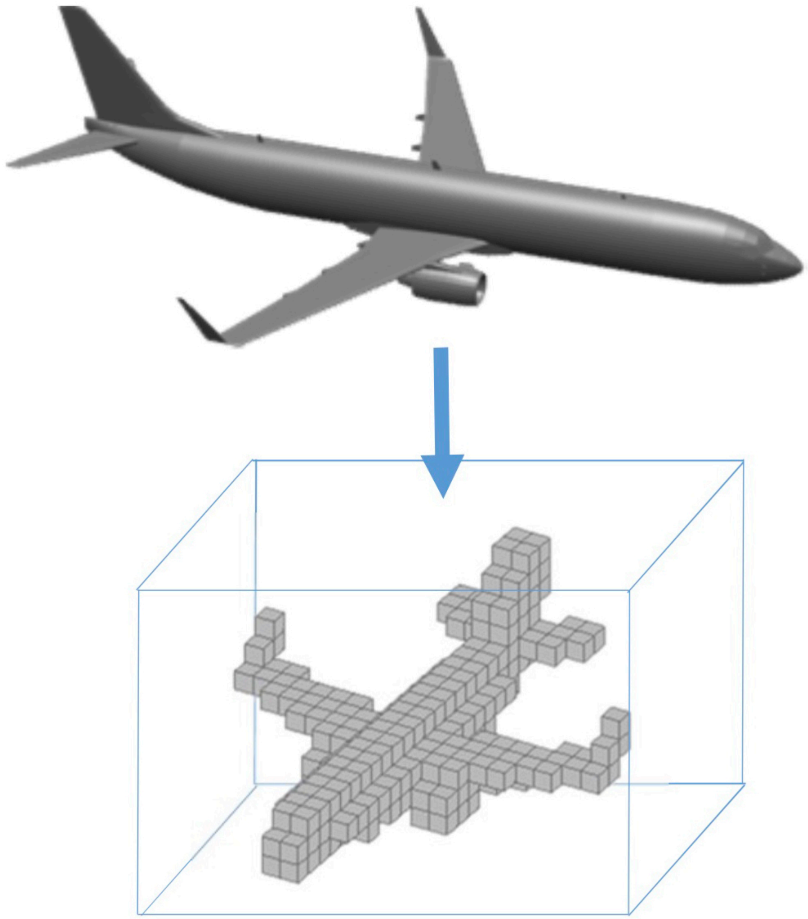
- Advantages
  - Easy to determine inside/outside
  - Easy to determine if a point is **on** the curve/surface

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

A related representation

- Binary volumetric grids

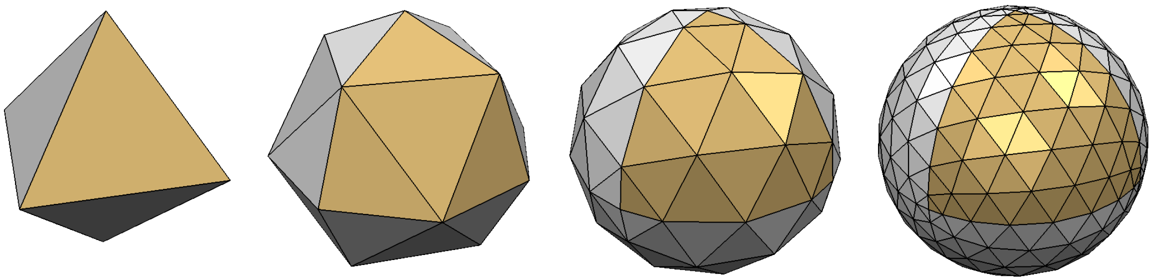
- Can be produced by thresholding the distance function, or from the scanned points directly

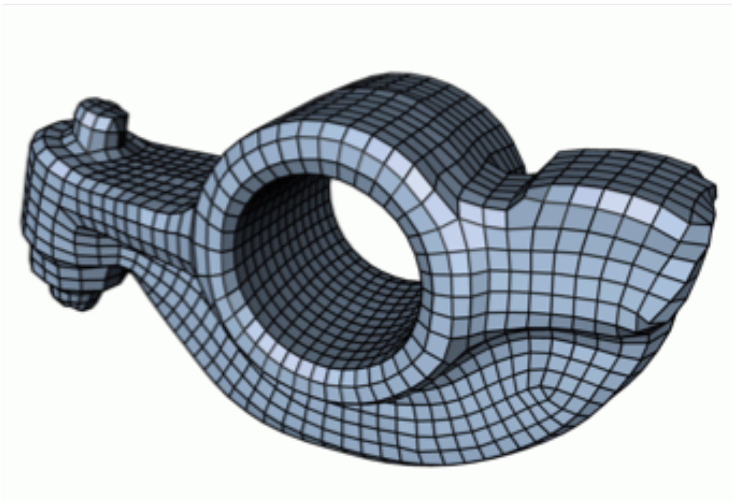


POLYGONAL MESHES

Polygonal Meshes

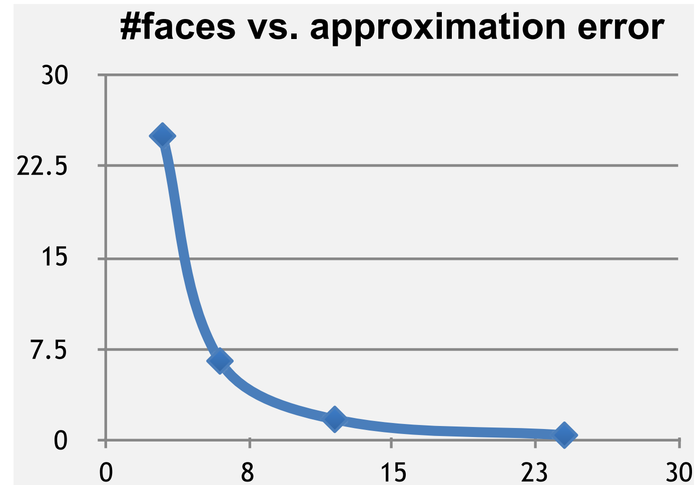
- Boundary representations of objects

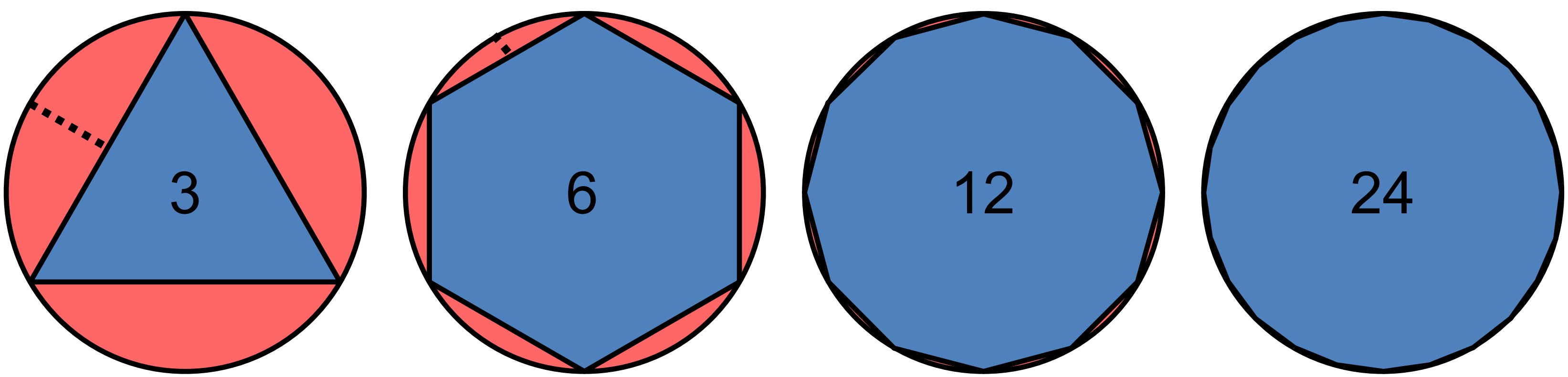




Meshes as Approximations of Smooth Surfaces

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

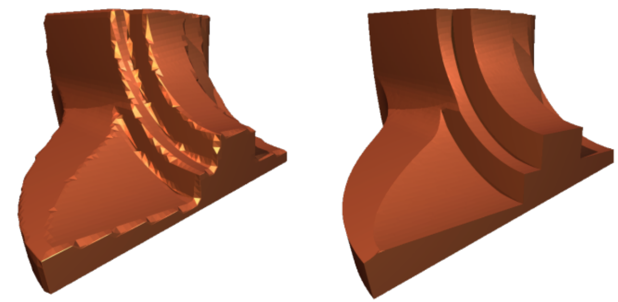


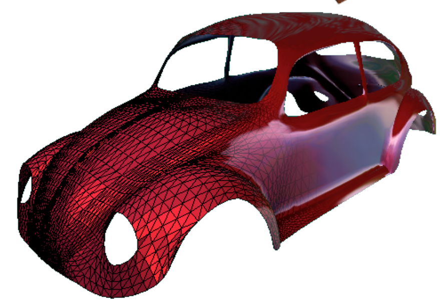


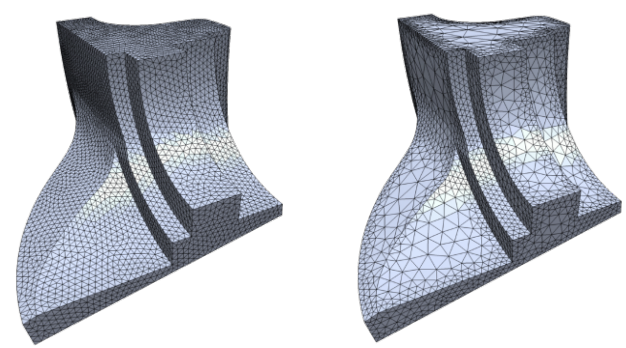
25% 6.5% 1.7% 0.4%

Polygonal Meshes

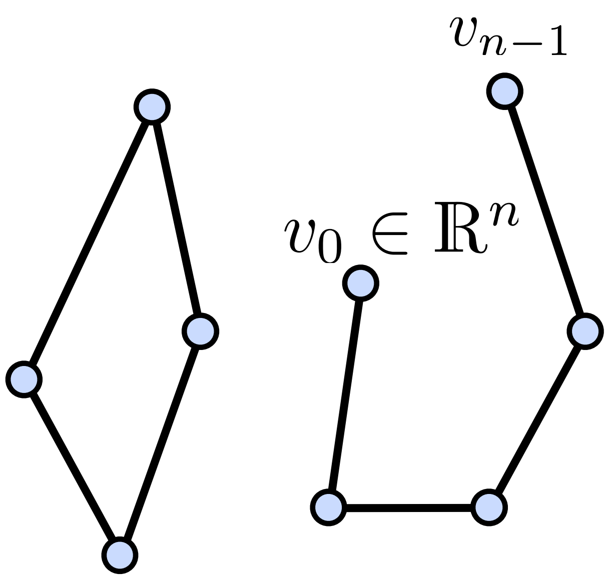
- Polygonal meshes are a good representation
  - approximation O(h^2)
  - arbitrary topology
  - adaptive refinement
  - efficient rendering







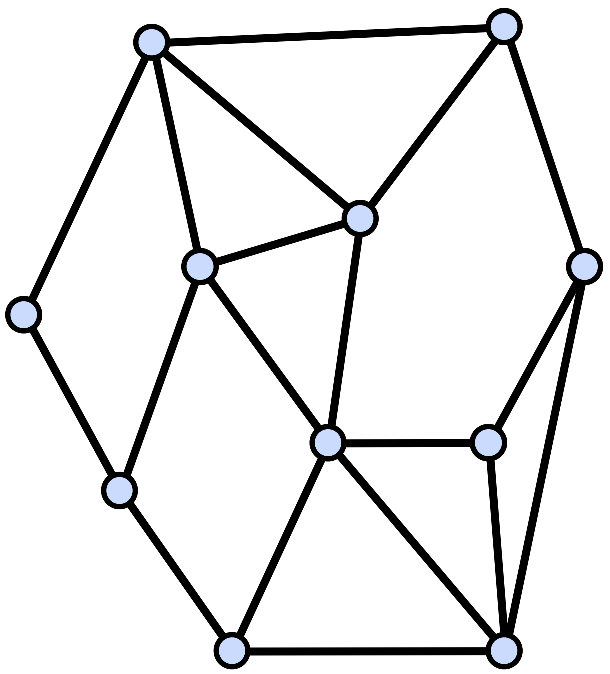
Polygon



- 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

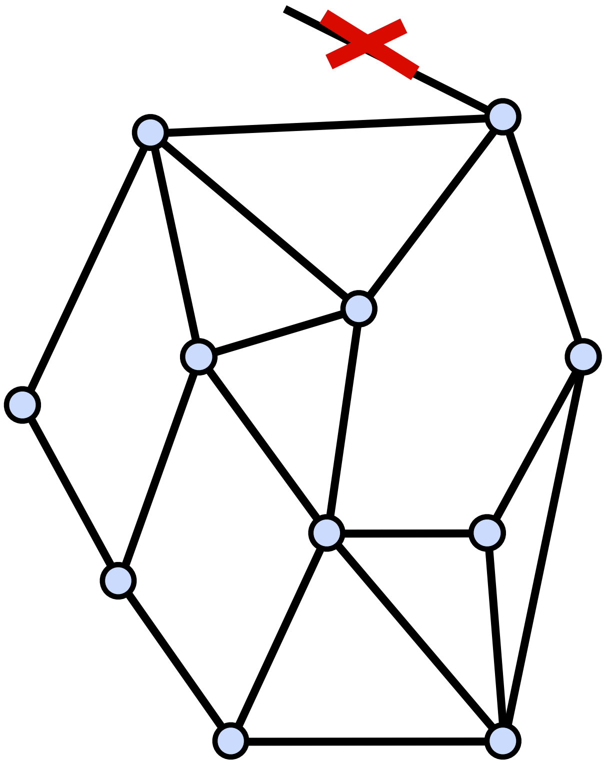


- A finite set M of closed, simple polygons Q_i is a polygonal mesh
- The intersection of two polygons in M is either empty, a vertex, or an edge

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

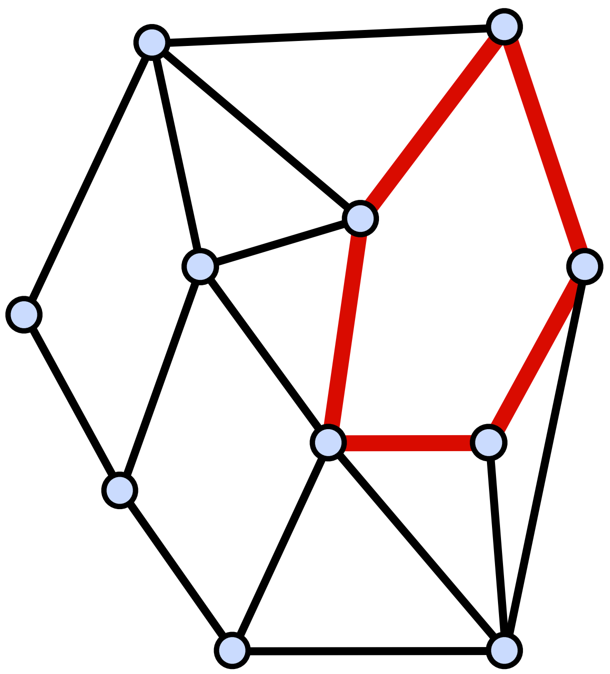
vertices edges faces

Polygonal Mesh



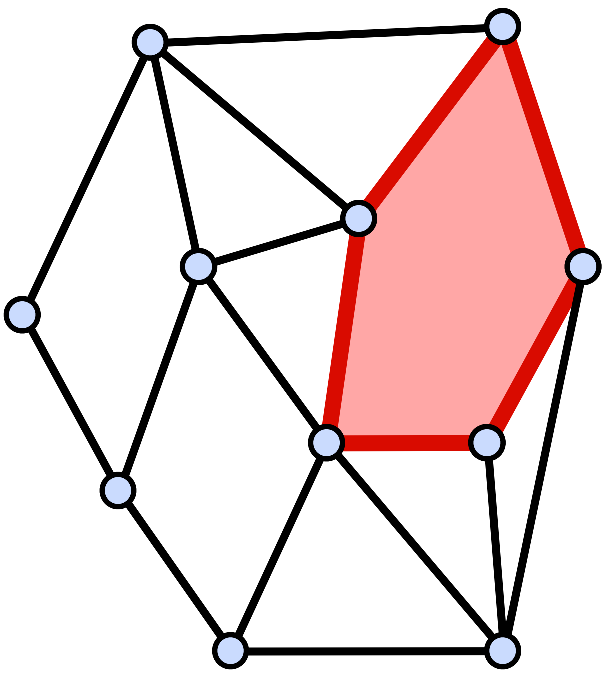
- A finite set *M* of closed, simple polygons Q_i is a polygonal mesh
- The intersection of two polygons in M is either empty, a vertex, or an edge
- Every edge belongs to at least one polygon

Polygonal Mesh



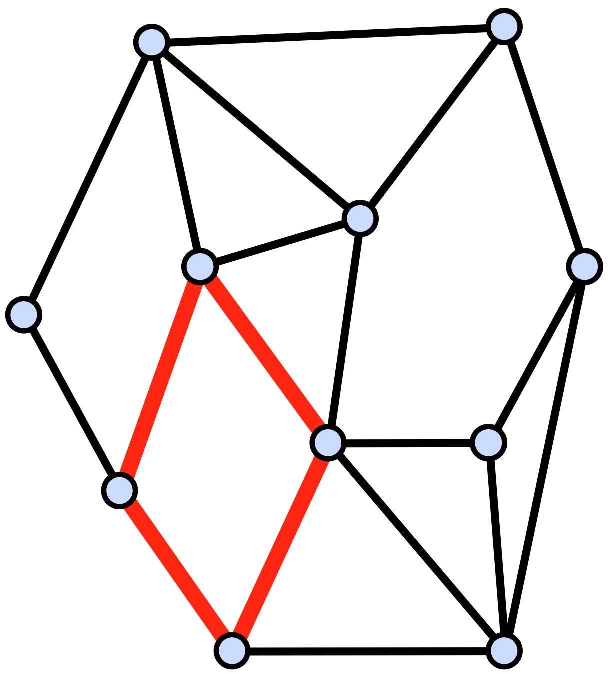
- A finite set *M* of closed, simple polygons Q_i is a polygonal mesh
- The intersection of two polygons in *M* is either empty, a vertex, or an edge
- Every edge belongs to at least one polygon
- Each Q_i defines a **face** of the polygonal mesh

Polygonal Mesh



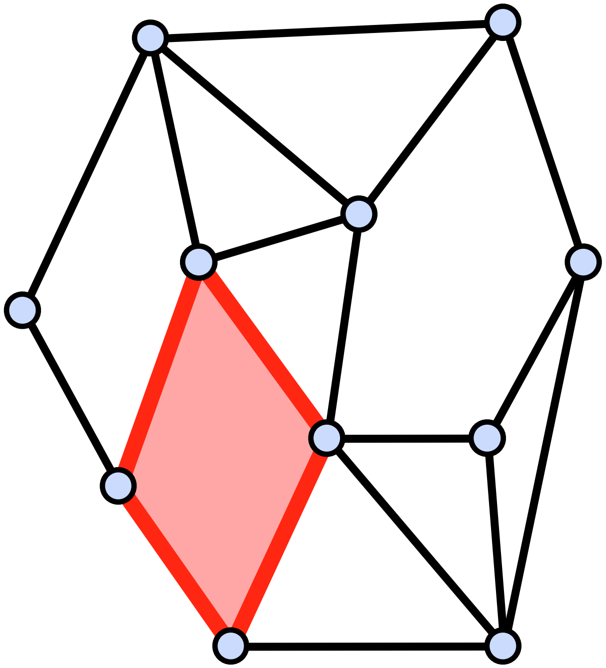
- A finite set M of closed, simple polygons Q_i is a polygonal mesh
- The intersection of two polygons in M is either empty, a vertex, or an edge
- Every edge belongs to at least one polygon
- Each Q_i defines a face of the polygonal mesh

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

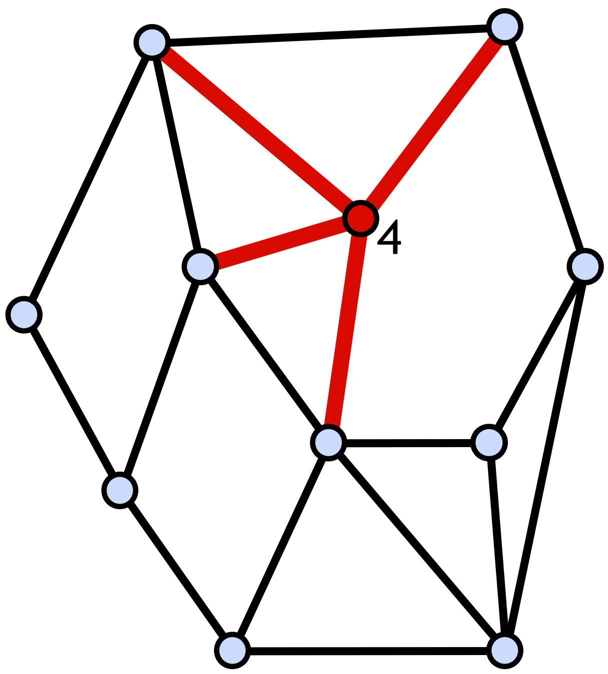
Polygonal Mesh



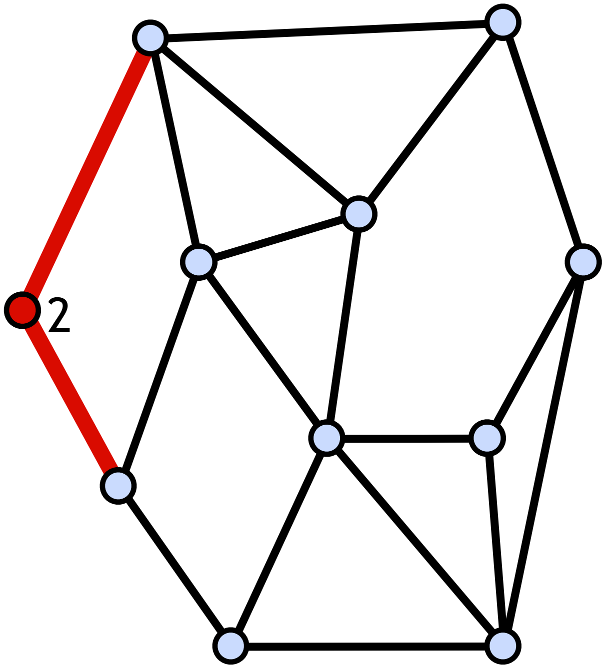
- A finite set M of closed, simple polygons Q_i is a polygonal mesh
- The intersection of two polygons in M is either empty, a vertex, or an edge
- Every edge belongs to at least one polygon
- Each Q_i defines a face of the polygonal mesh

Polygonal Mesh

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

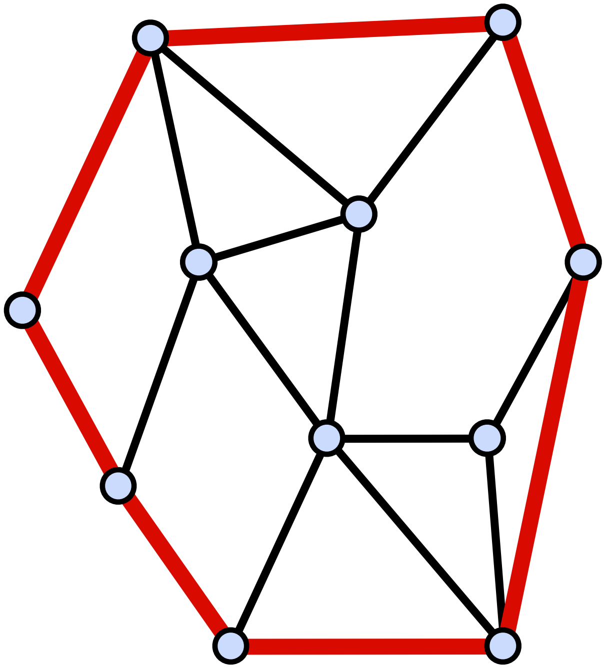


Polygonal Mesh

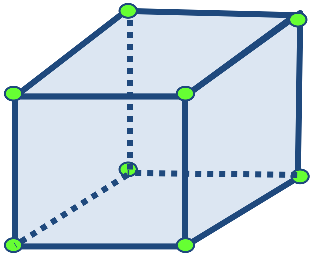


- Vertex degree or valence = number of incident edges

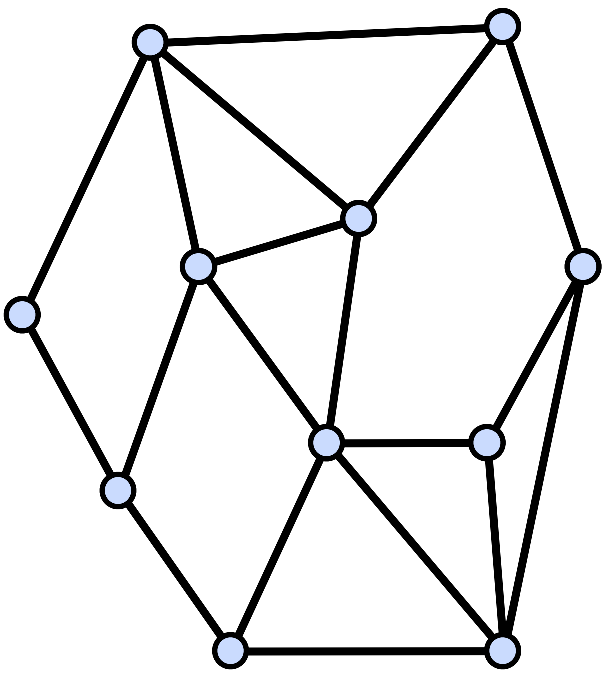
Polygonal Mesh



- 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



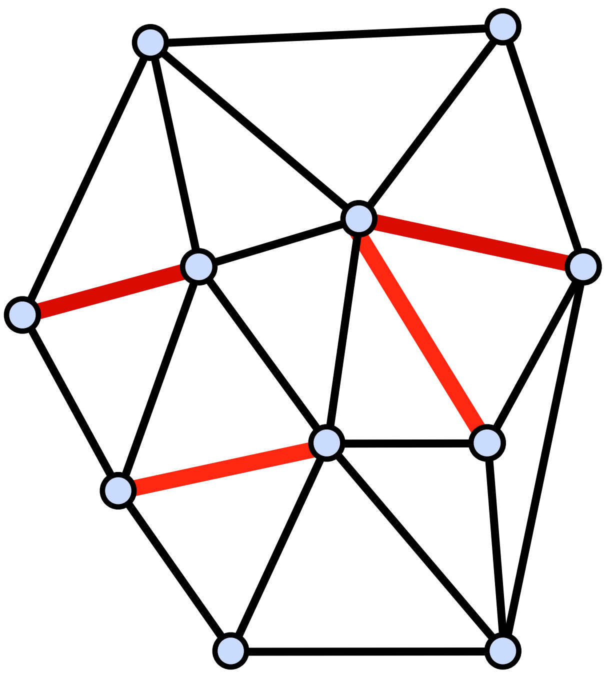
Triangulation



- 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

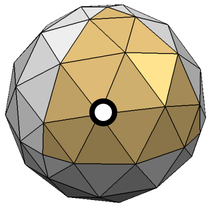


- Polygonal mesh where every face is a triangle

- 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

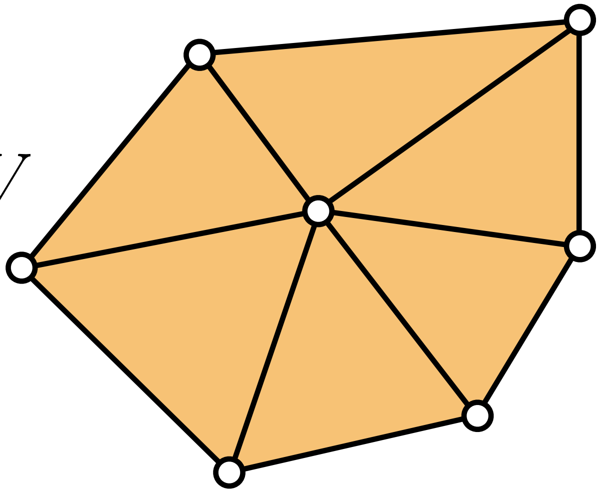


$$V = \{v_1, \ldots, v_n\}$$

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

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

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



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

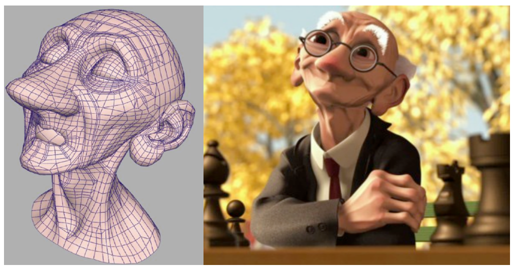
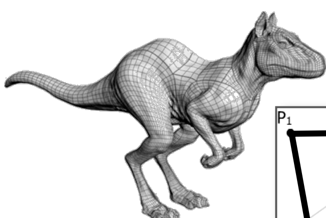
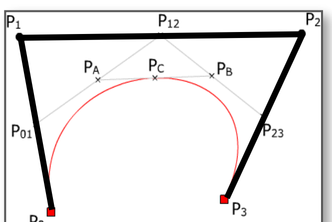
- Used in formats
- OBJ, OFF, WRL
- Storage
  - Vertex: position
  - Face: vertex indices
  - 12 bytes per vertex
  - 12 bytes per face
  - 36*v bytes for the mesh
- No explicit neighborhood info

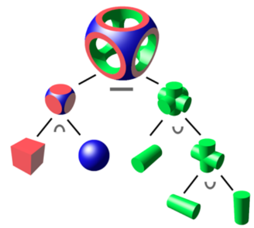
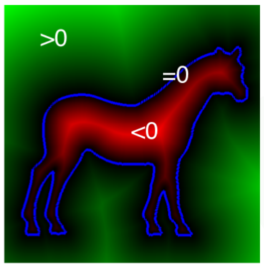
Vertices			
v0	x0	y0	z0
v1	x1	x1	z1
v2	x2	y2	z2
v3	x3	y3	z3
v4	x4	y4	z4
v5	x5	y5	z5
v6	x6	y6	z6
...	...	...	...

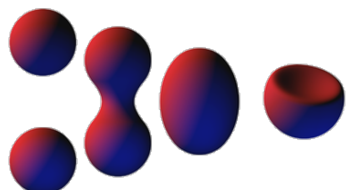
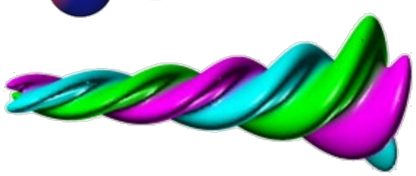
Triangles			
t0	v0	v1	v2
t1	v0	v1	v3
t2	v2	v4	v3
t3	v5	v2	v6
...	...	...	...

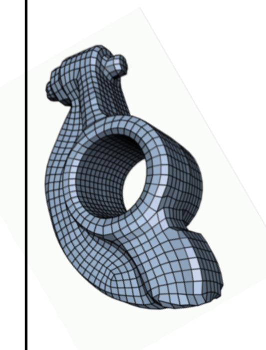
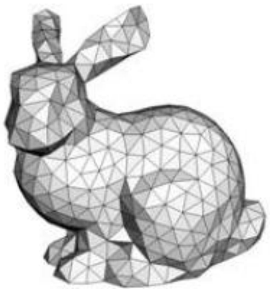
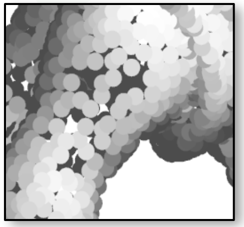
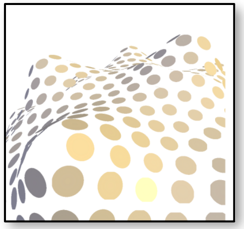
queue: halfedge datastructure!

Summary

<table>
<tr><th>Parametric</th><th>Implicit</th><th>Discrete/Sampled</th></tr>
<tr>
<td>




- Splines, tensor-product surfaces
- Subdivision surfaces
</td>
<td>



- Distance fields
- Metaballs/blobs
</td>
<td>





- Meshes
- Point set surfaces
</td>
</tr>
</table>

CONVERSIONS

Implicit → Mesh
Mesh → Points (next time!)

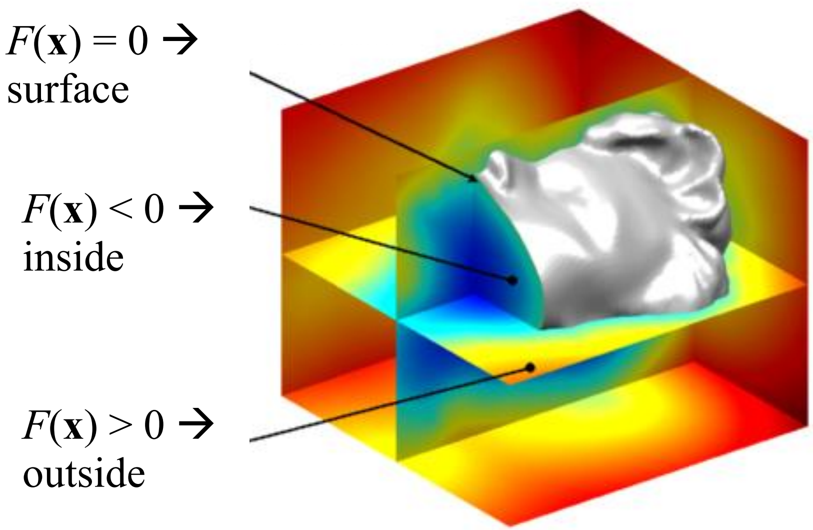
UC San Diego

IMPLICIT → MESH

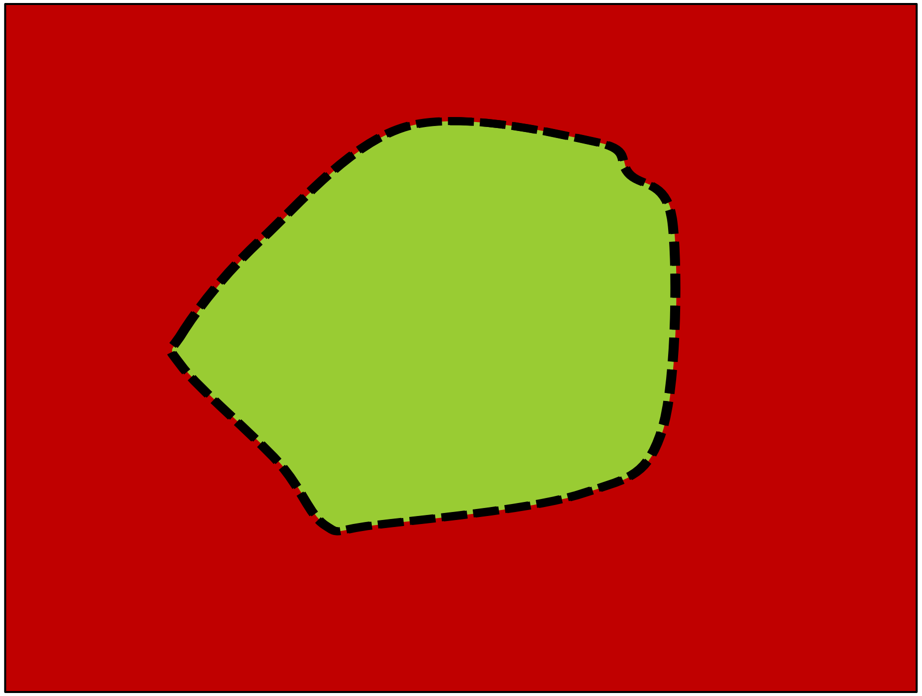
Marching Cubes

Extracting the Surface

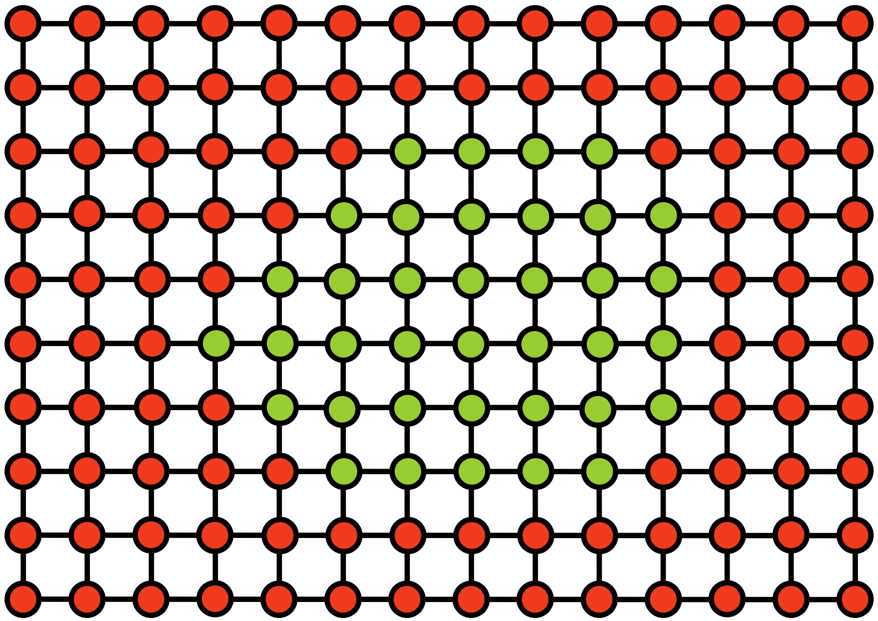
- Wish to compute a manifold mesh of the level set



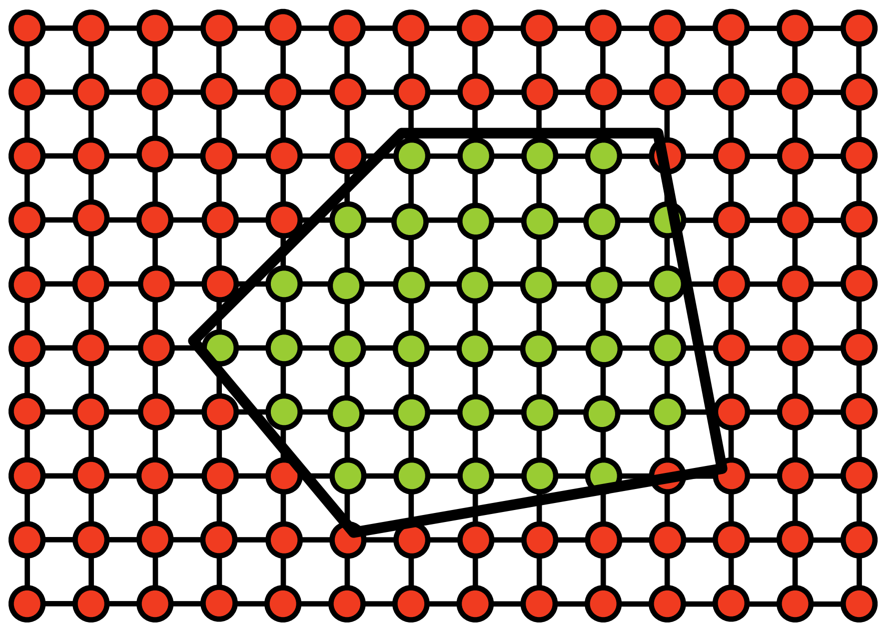
Sample the SDF



Sample the SDF



Sample the SDF

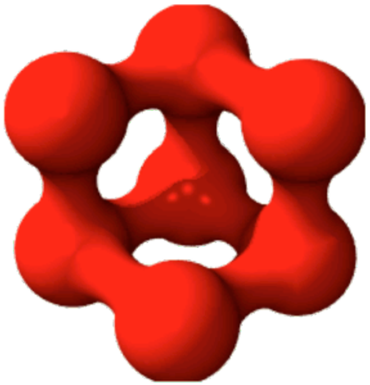


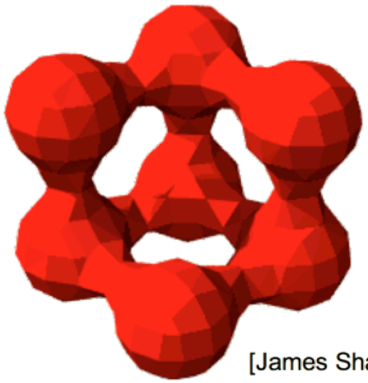
Marching Cubes

Converting from implicit to explicit representations.

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

Create a triangle mesh that approximates the surface.





[James Sharman]

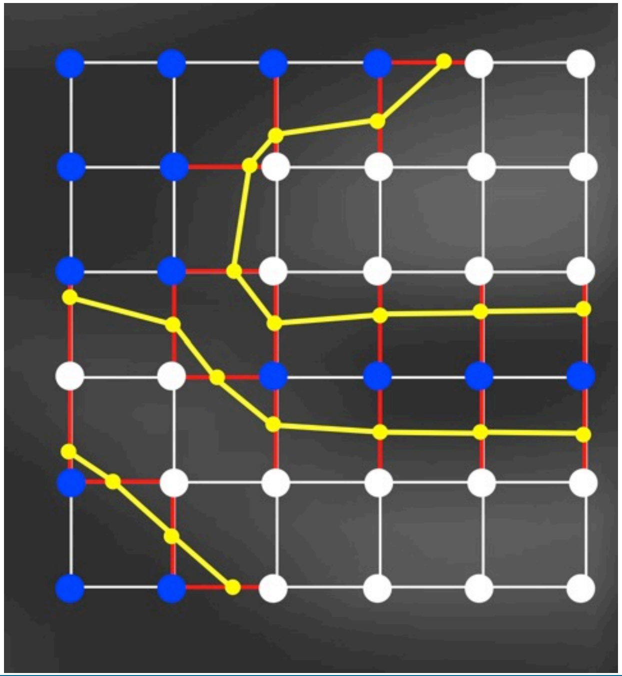
Lorensen and Cline, SIGGRAPH '87

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.

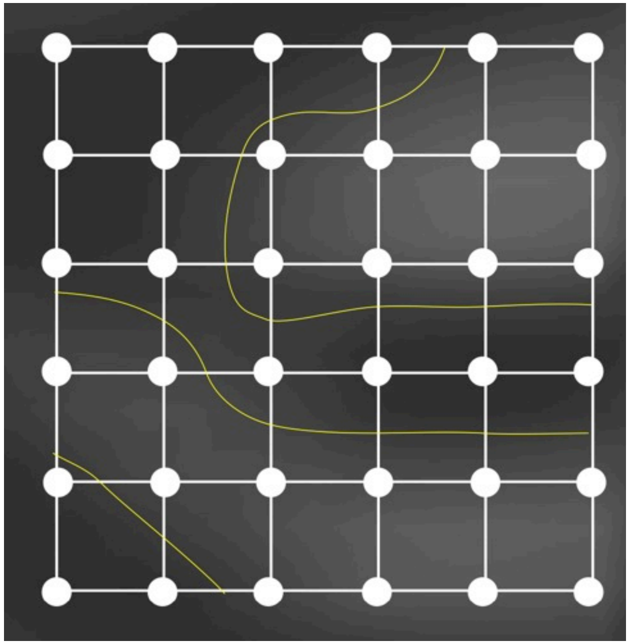


Marching Squares (2D)

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)

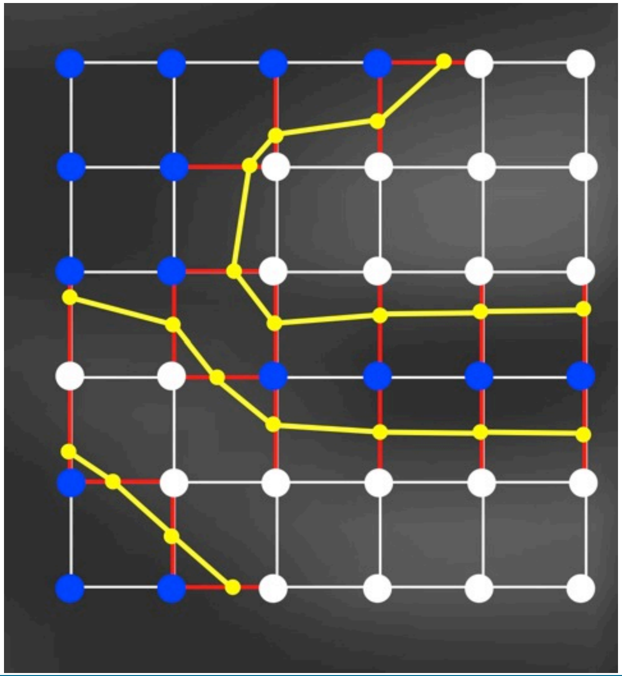
Computing the intersections:

- 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:

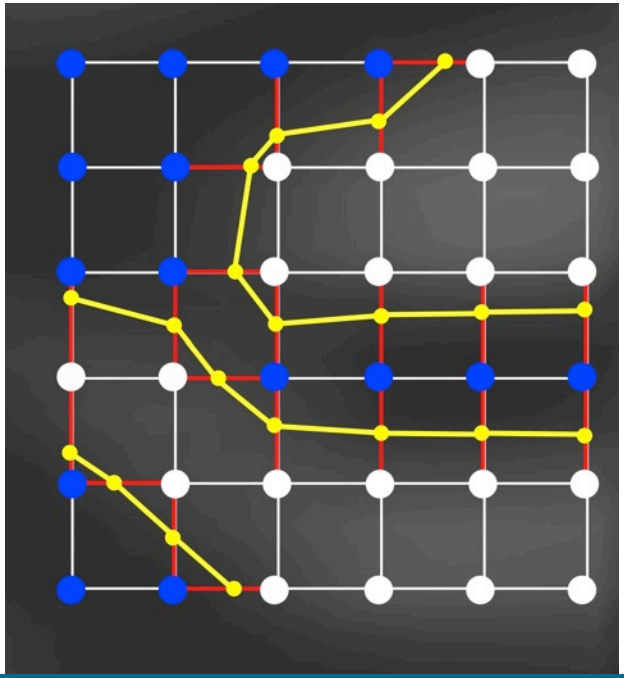
$$t = \frac{f(x_1)}{f(x_2) - f(x_1)}$$



Marching Squares (2D)

Connecting the intersections:

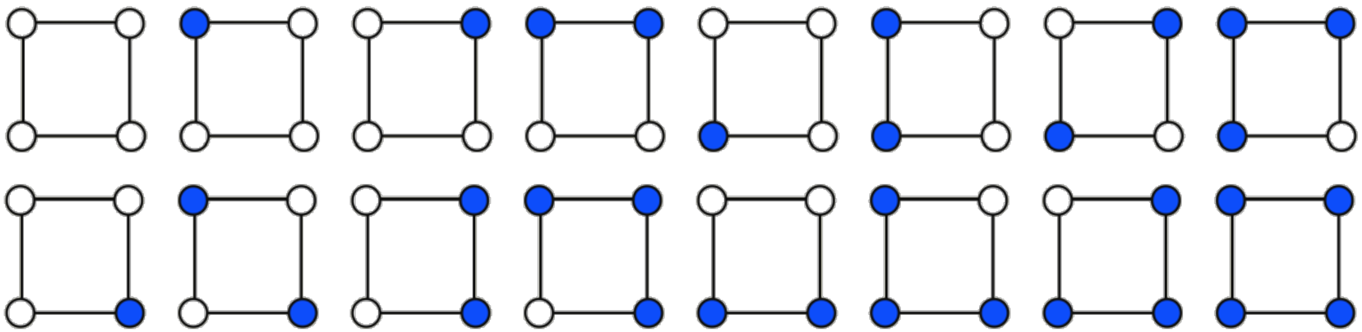
- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.



Marching Squares (2D)

Connecting the intersections:

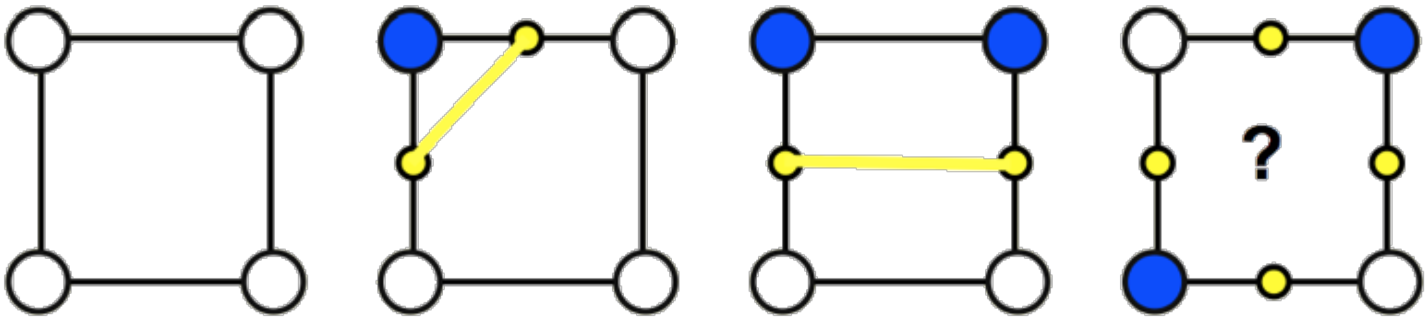
- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.
- Group those leading to the same intersections



Marching Squares (2D)

Connecting the intersections:

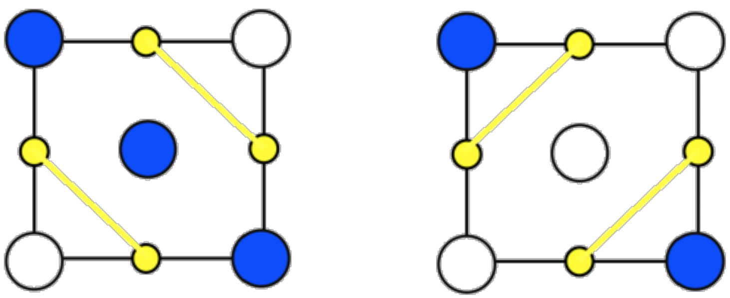
- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.
- Group those leading to the same intersections.
- Group equivalent after rotation.
- Connect intersections

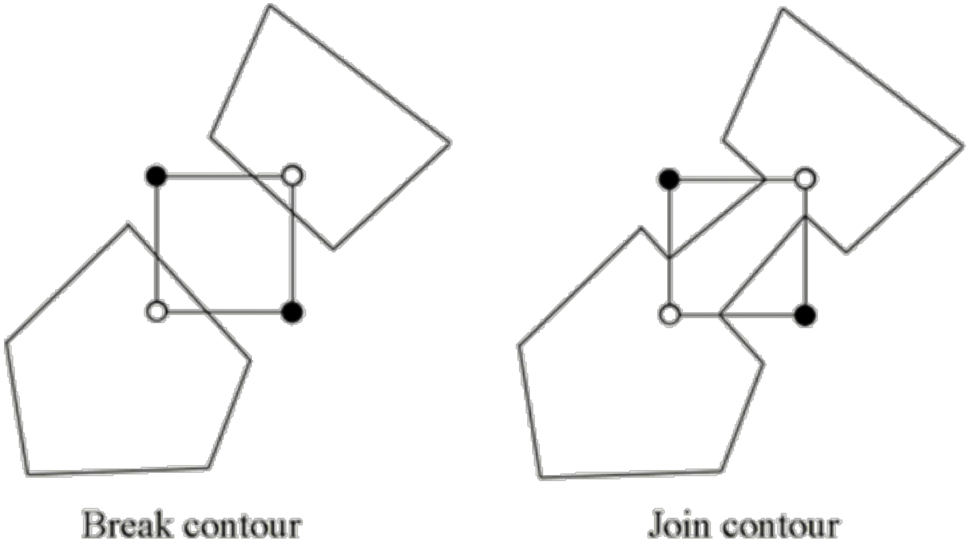


Marching Squares (2D)

Connecting the intersections:

Ambiguous cases:





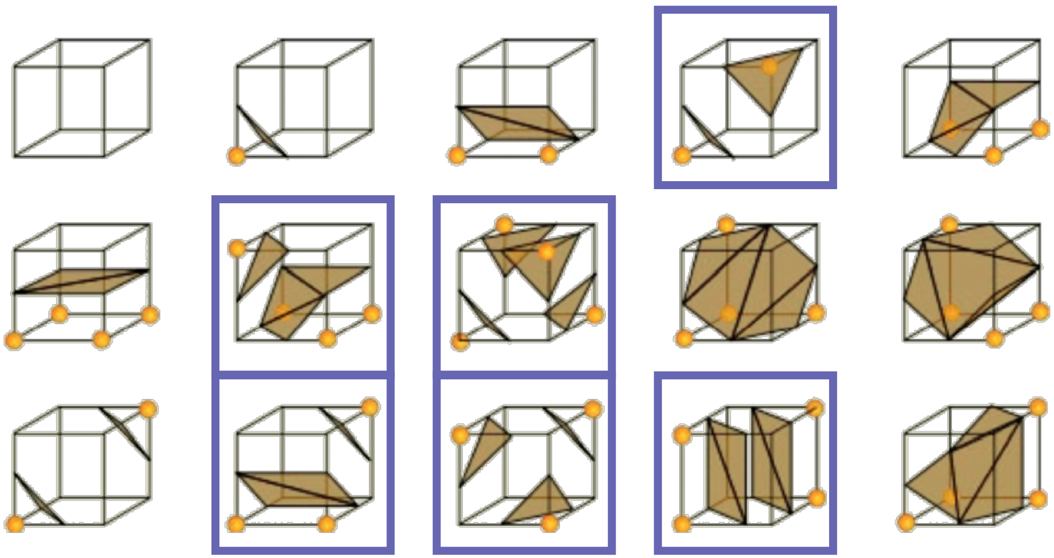
Two options:

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

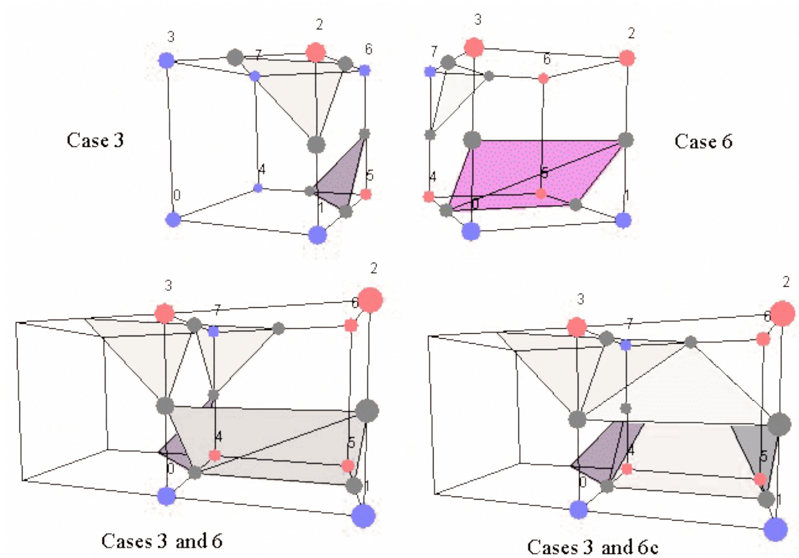
Marching Cubes (3D)

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

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

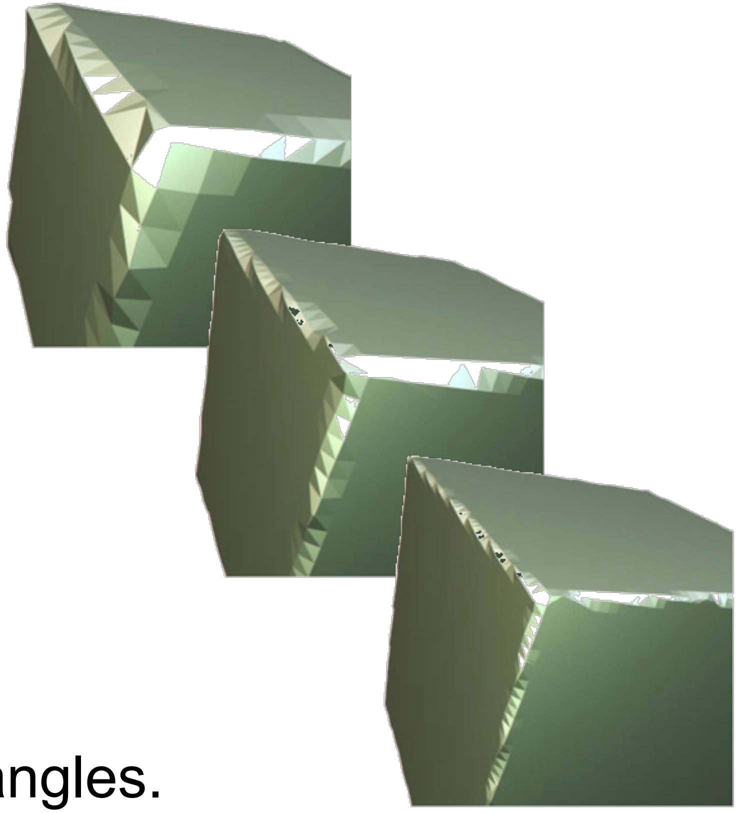


the 15 cases



explore ambiguity to avoid holes!

Marching Cubes (3D)



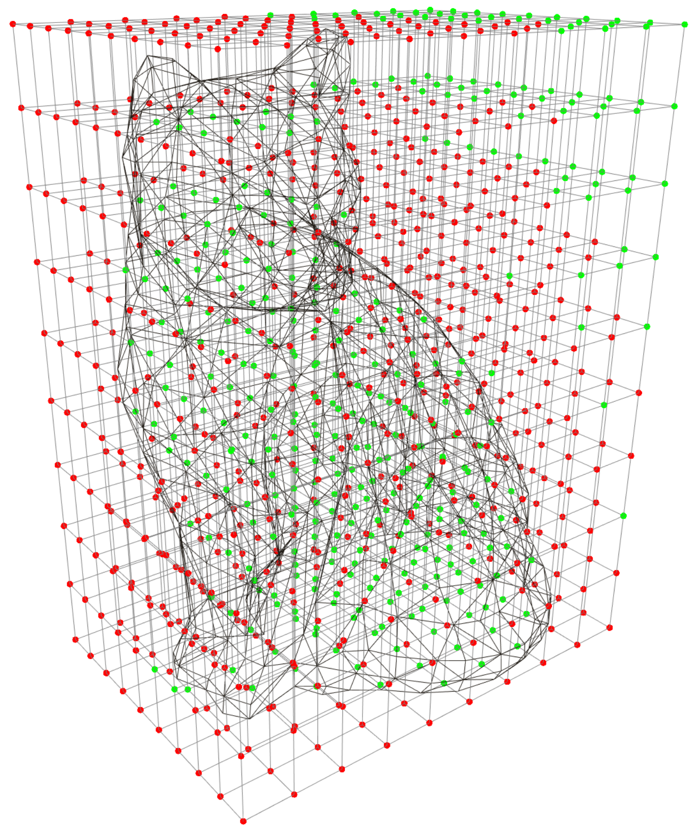
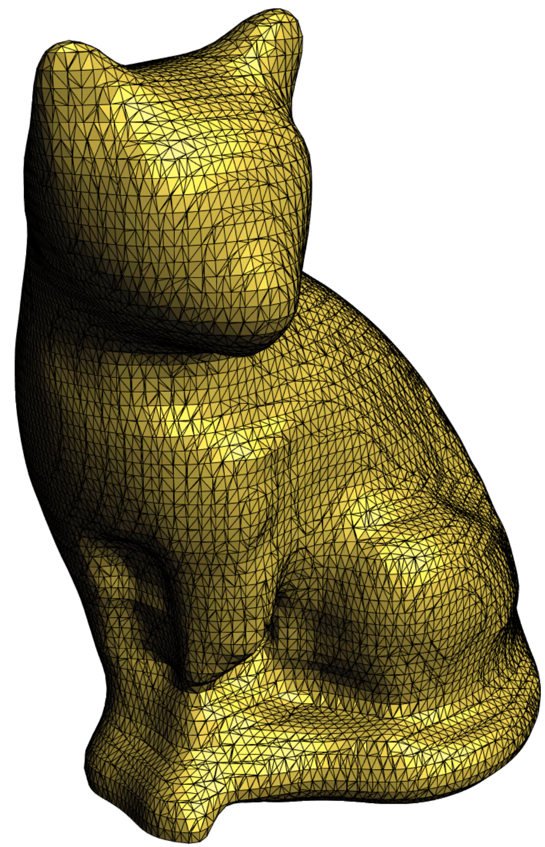
Main Strengths:

- 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

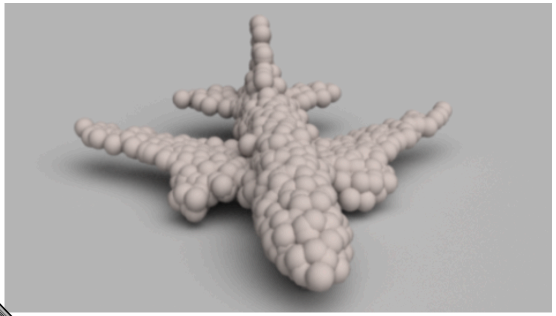
- Alec Jacobson's GP toolbox
  - https://github.com/alecjacobson/gptoolbox
  - MATLAB, various mesh and matrix routines
- Gabriel Peyre's Fast Marching Toolbox
  - https://www.mathworks.com/matlabcentral/fileexchange/6110-toolbox-fast-marching
  - On-surface distances (more next time!)
- OpenFlipper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

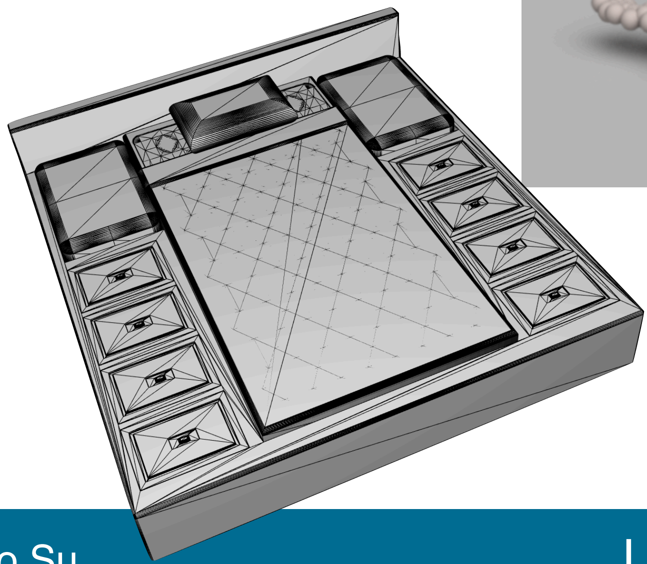
MESH-> POINT CLOUD

Sampling

From Surface to Point Cloud - Why?

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

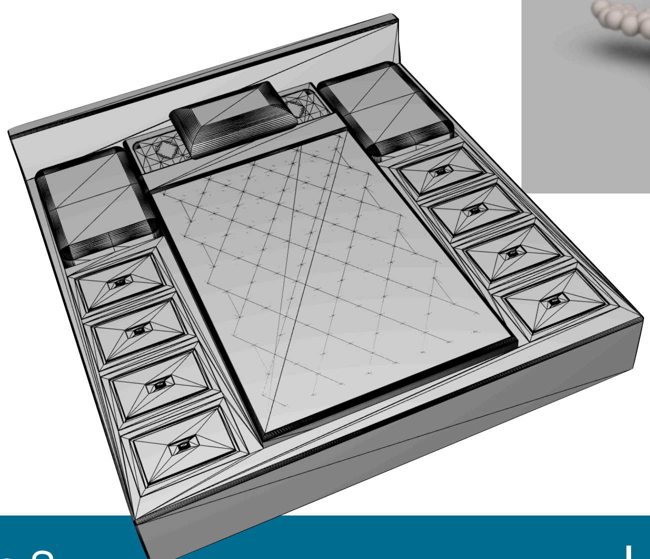


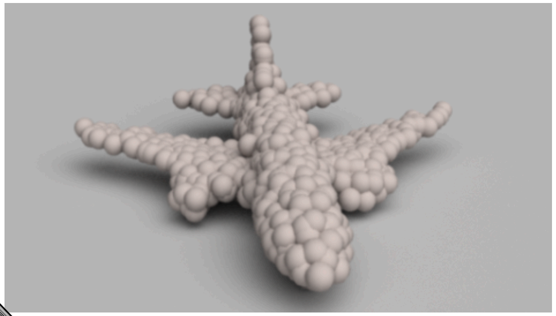


CAD meshes:
many components
bad triangles
connectivity problems

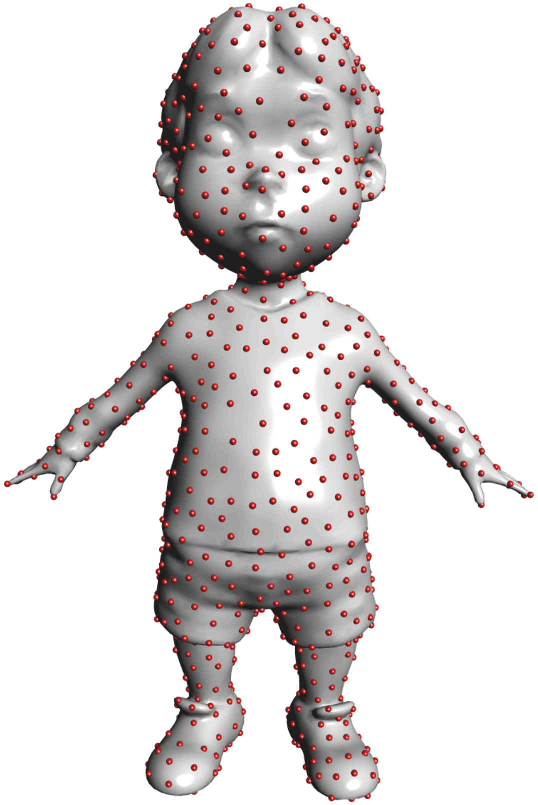
From Surface to Point Cloud - Why?

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





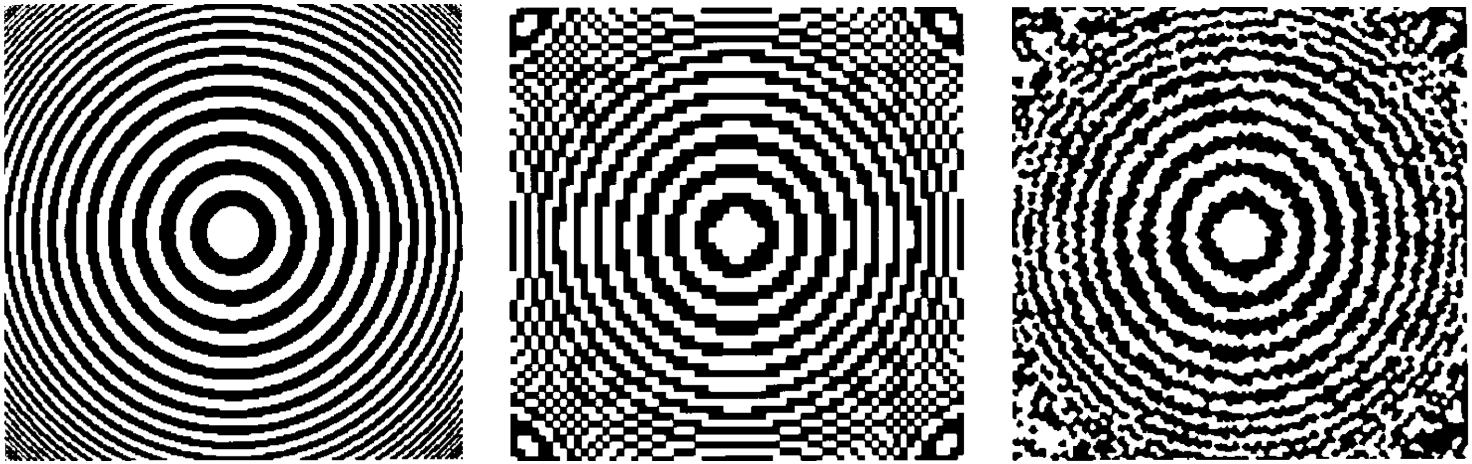
CAD meshes:
many components
bad triangles
connectivity problems



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

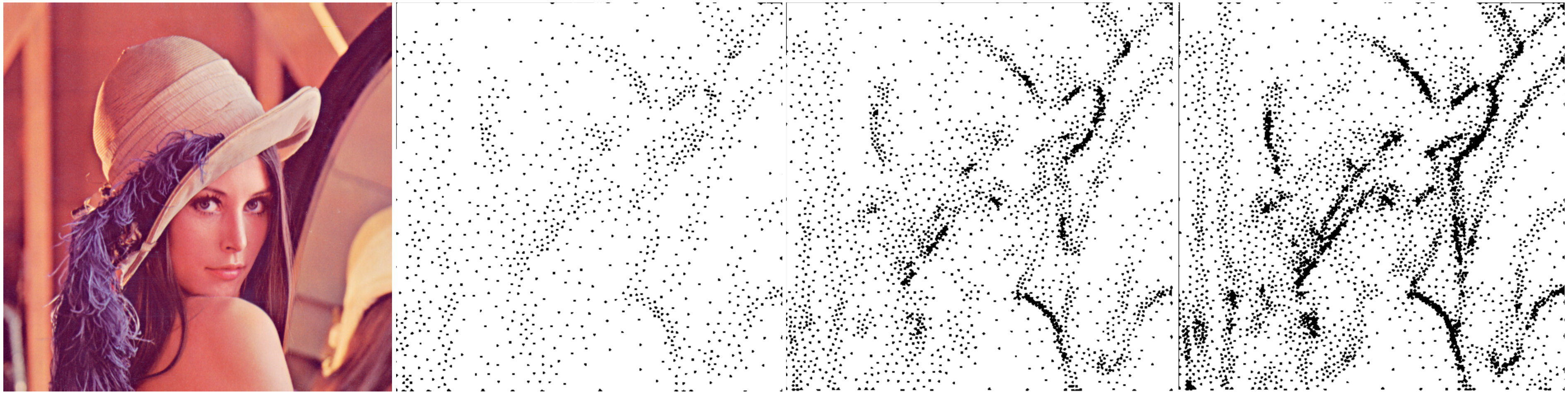
1. Create an initial sample point set *S*
    - Image corners + additional random point.
2. Find the point which is the farthest from all point in *S*

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

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

Farthest Point Sampling

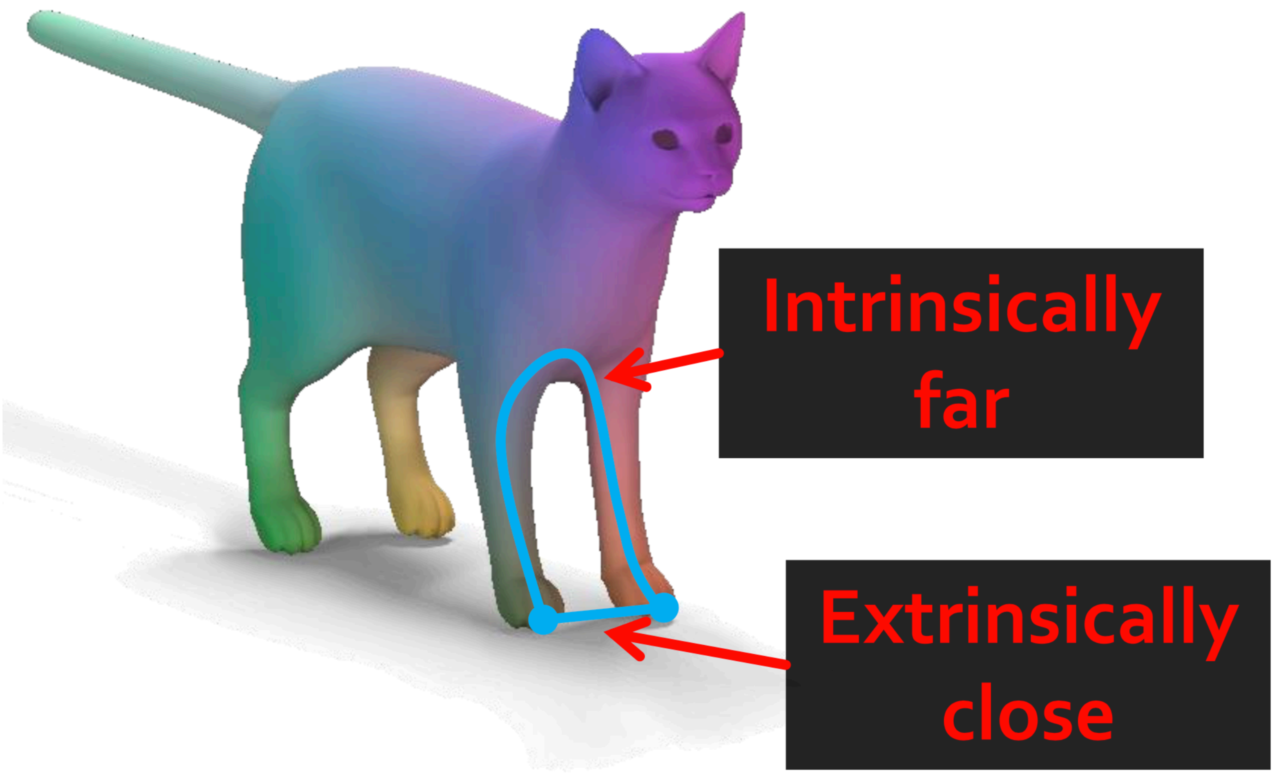
- Depends on a notion of distance on the sampling domain
- Can be made adaptive, via a weighted distance

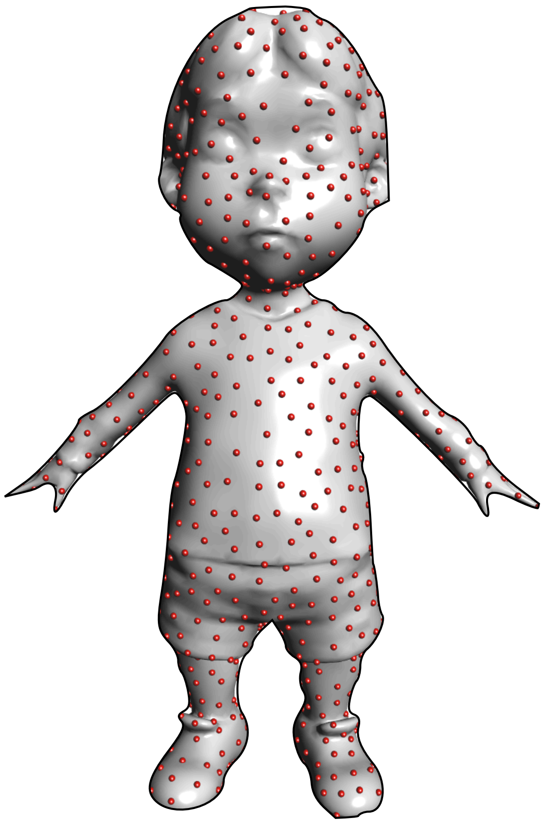


Eldar et al. 1997, "The Farthest Point Strategy for Progressive Image Sampling"

FPS on surfaces

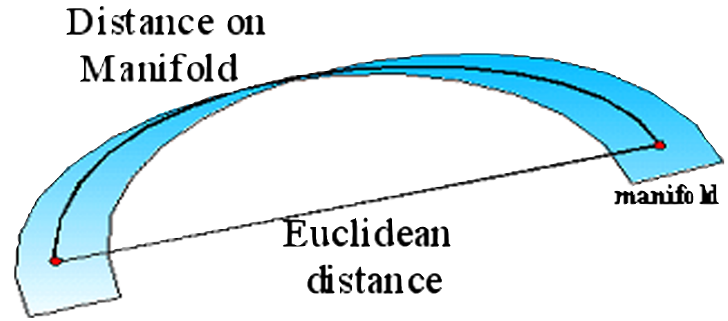
- What's an appropriate distance?

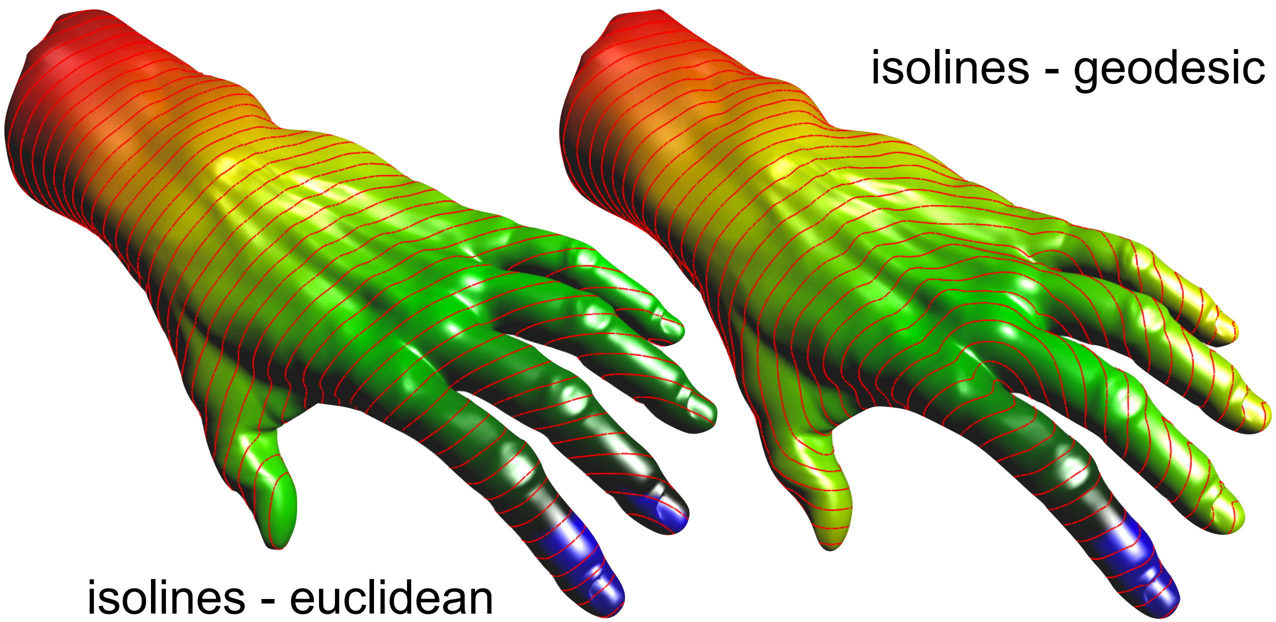




On-Surface Distances

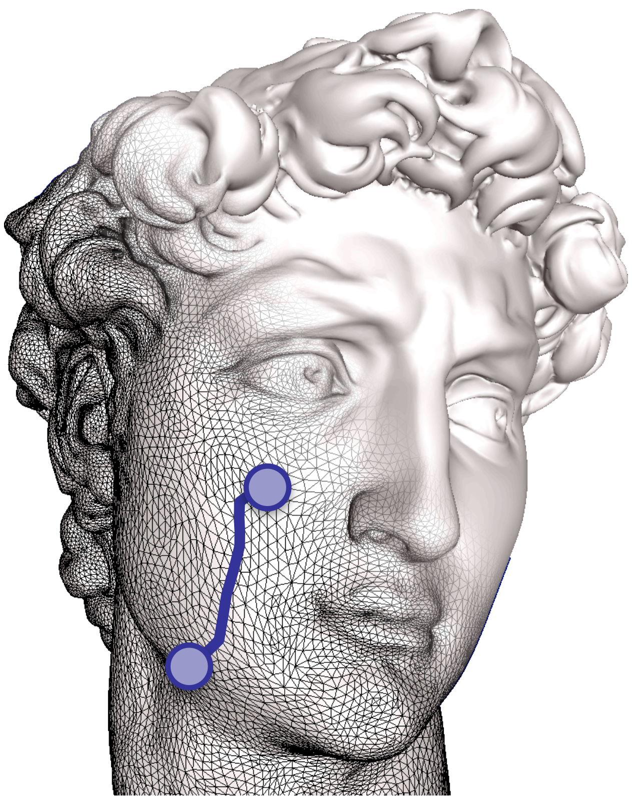
- Geodesics: Straightest and **locally shortest** curves





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]
$S = \{\}$

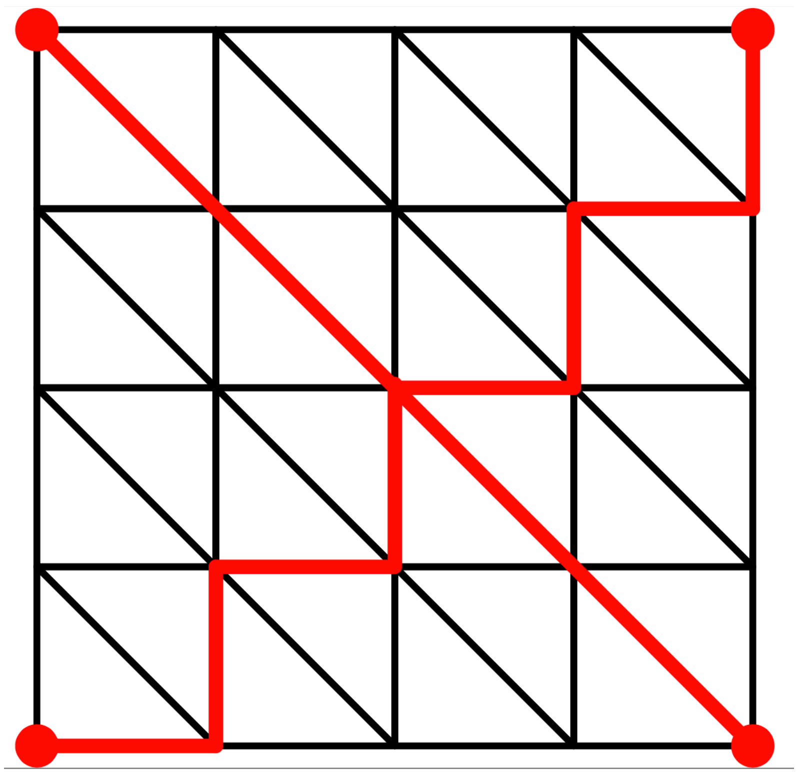
for each iteration k:
    # update
    k = argmin(d_k), for v_k not in S
    S.append(v_k)
    for neighbors index v_l of v_k:
        d_l = min([d_l, d_k + d_{kl}])

Dijkstra's algorithm!

Dijkstra Geodesics

Can be asymmetric - no matter how fine the mesh!

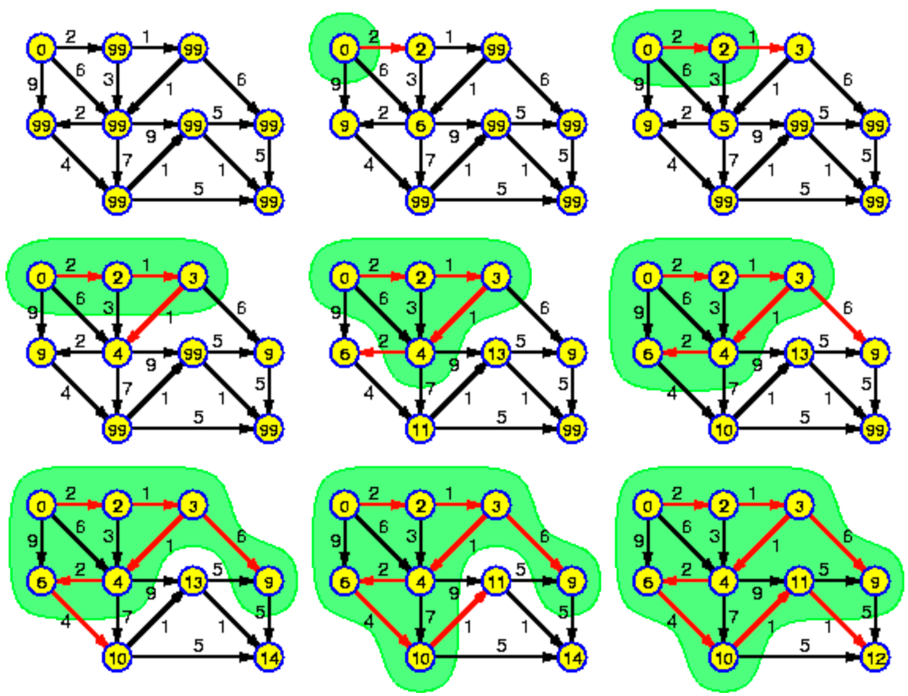
$l = \sqrt{2}$ $l = 2$



Dijkstra Geodesics

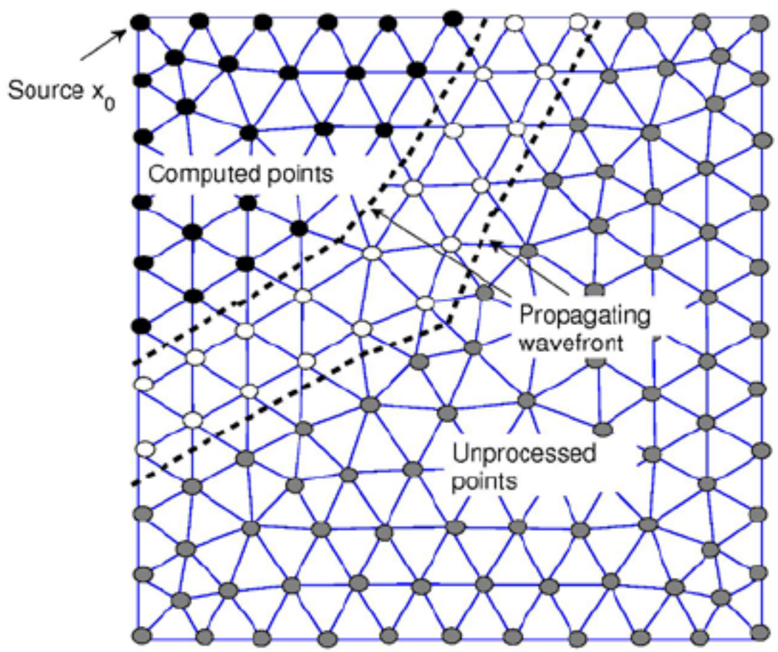
Can be asymmetric - no matter how fine the mesh!

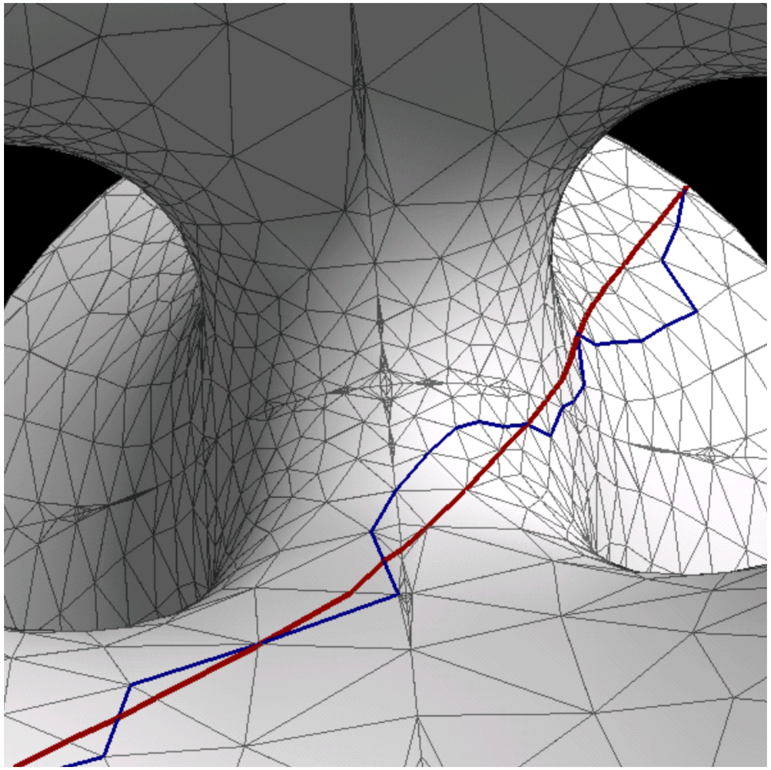
- Dikjstra as front propagation



Fast Marching Geodesics

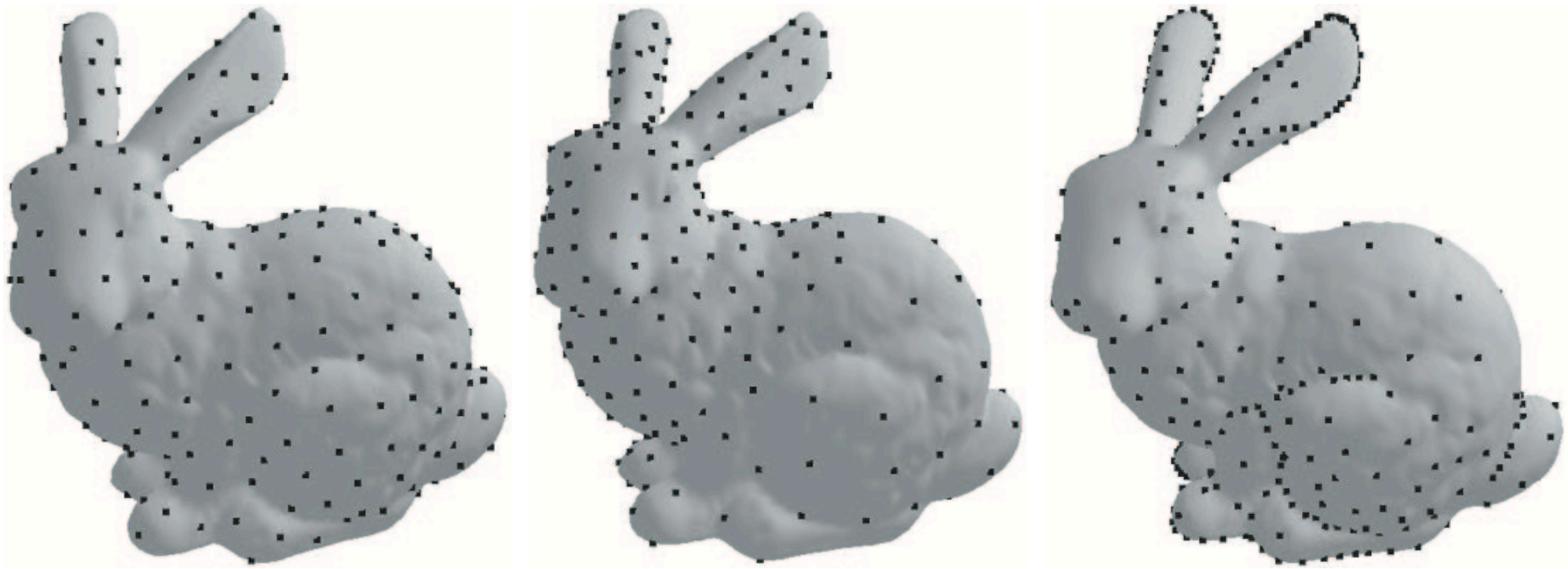
- A better approximation: allow fronts to cross triangles!





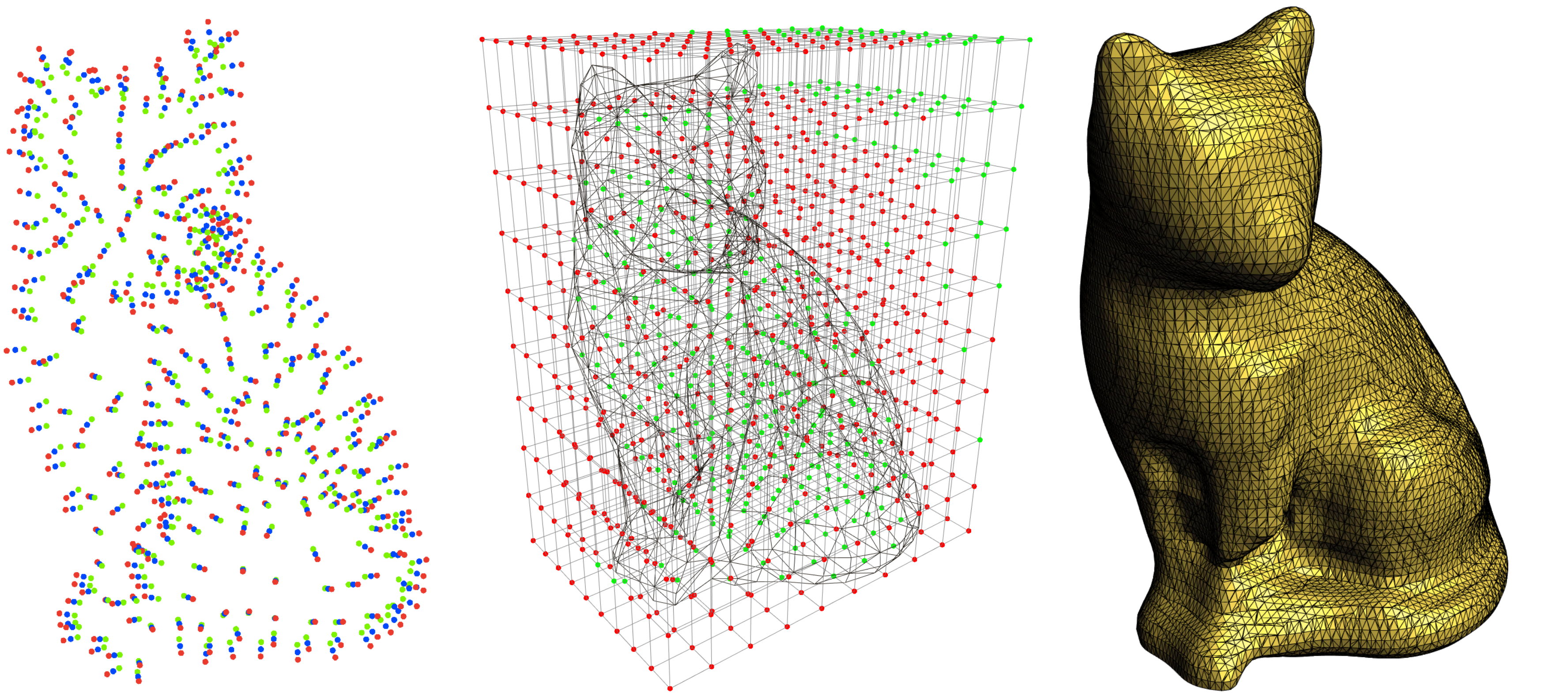
Kimmel and Sethian 1997, “Computing Geodesic Paths on Manifolds”

FPS on a Mesh

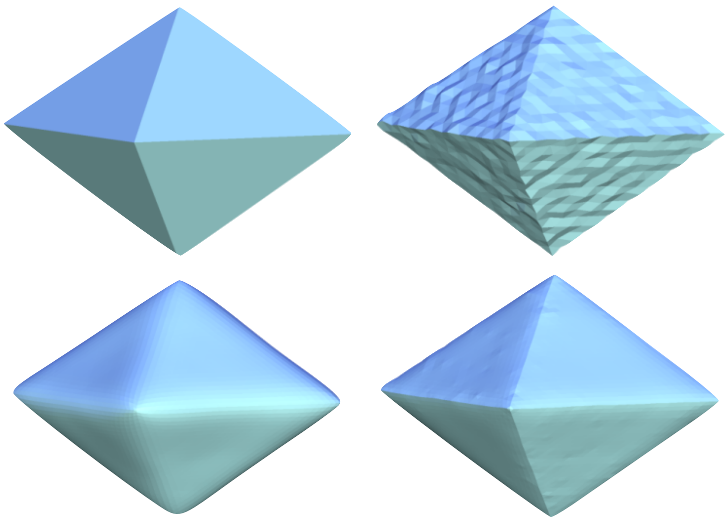


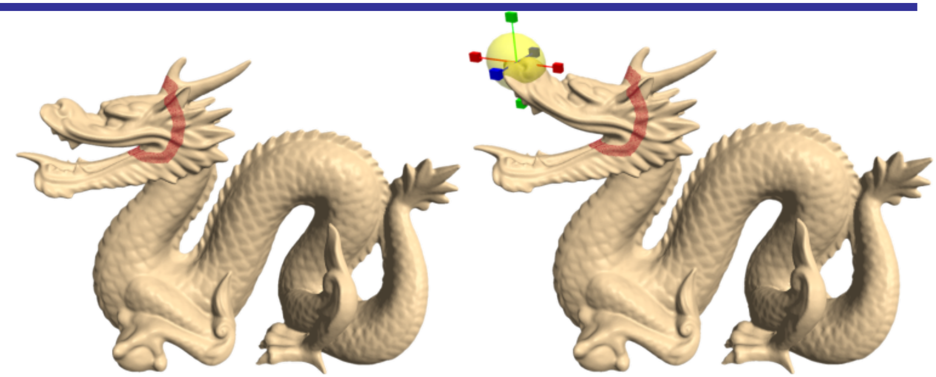
Peyré and Cohen 2003, Geodesic Remeshing Using Front Propagation

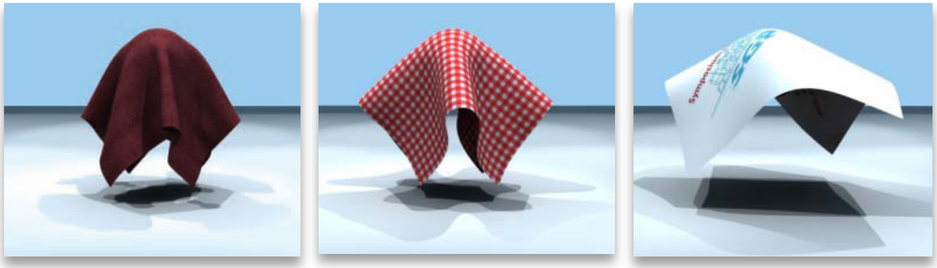
Recap: Conversions

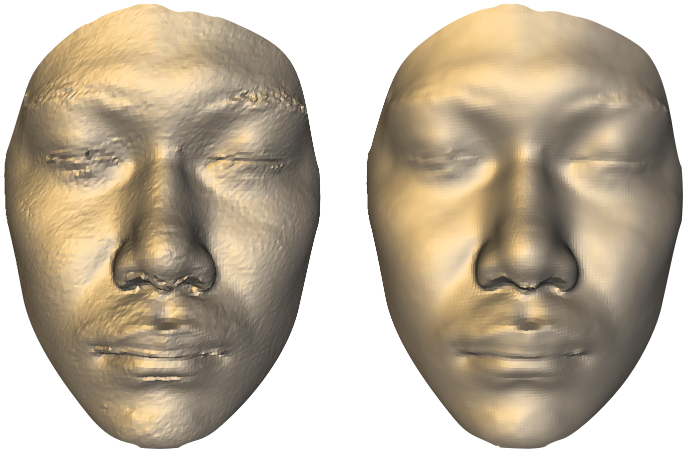


Geometry Foundations: Discrete Differential Geometry





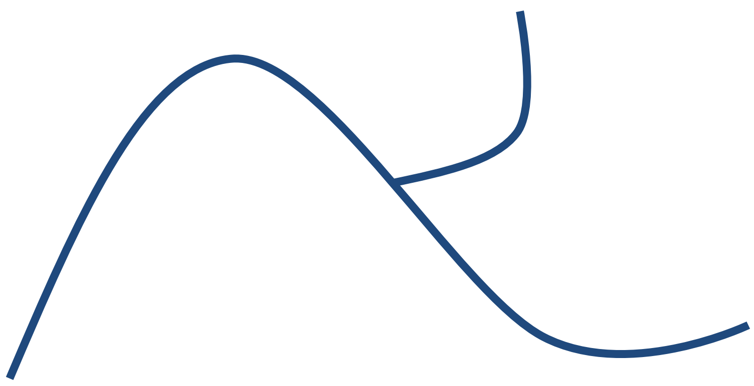




slides credits: , Daniele Panozzo

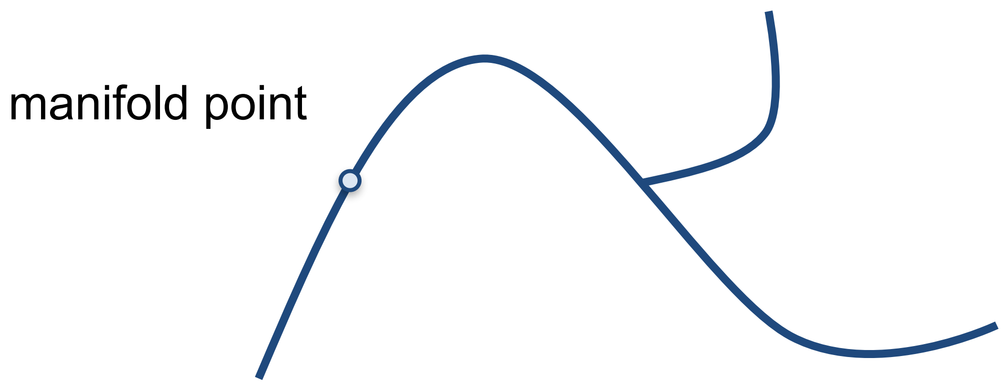
Differential Geometry Basics

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



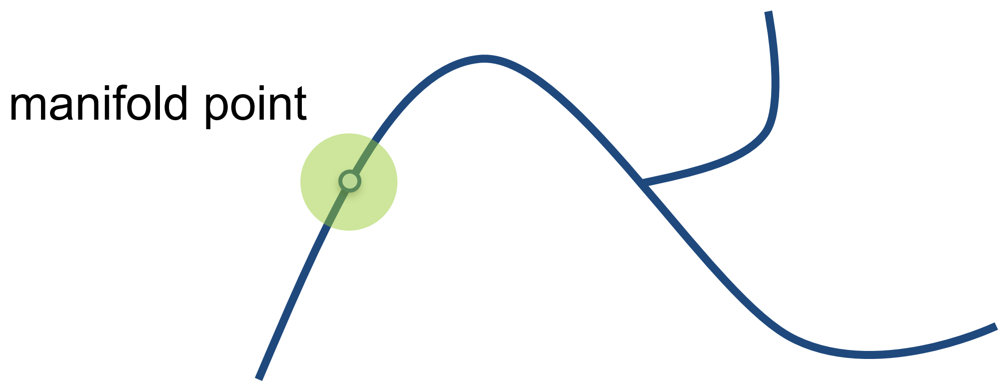
Differential Geometry Basics

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



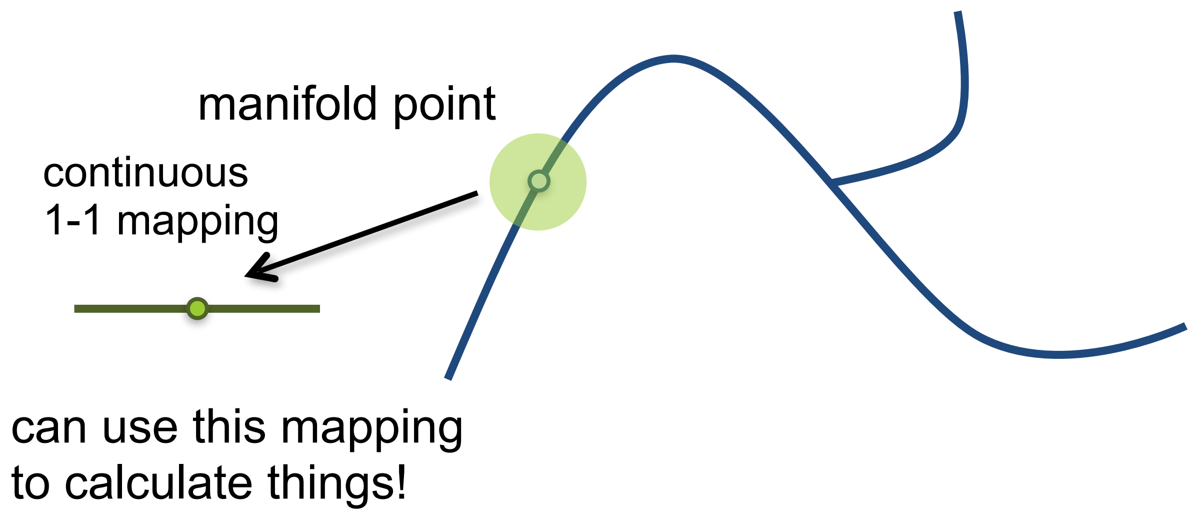
Differential Geometry Basics

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



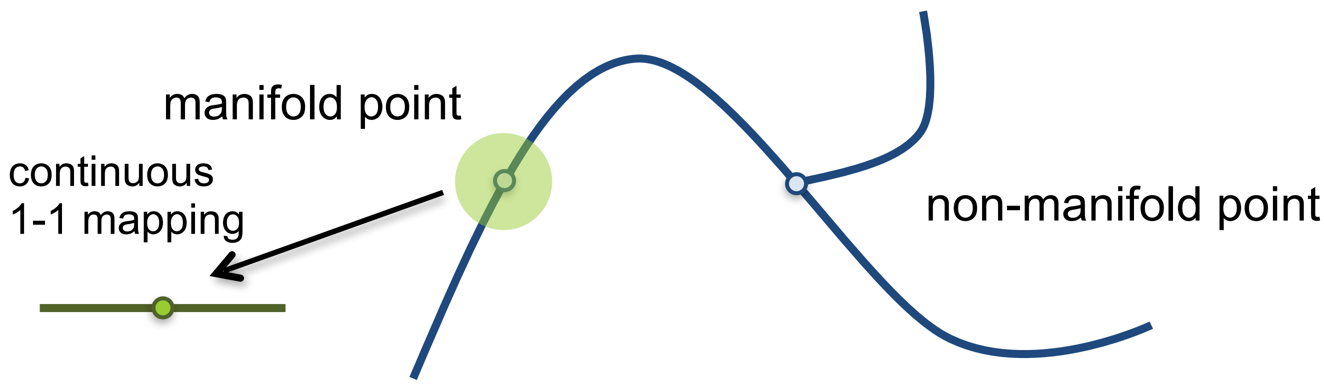
Differential Geometry Basics

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



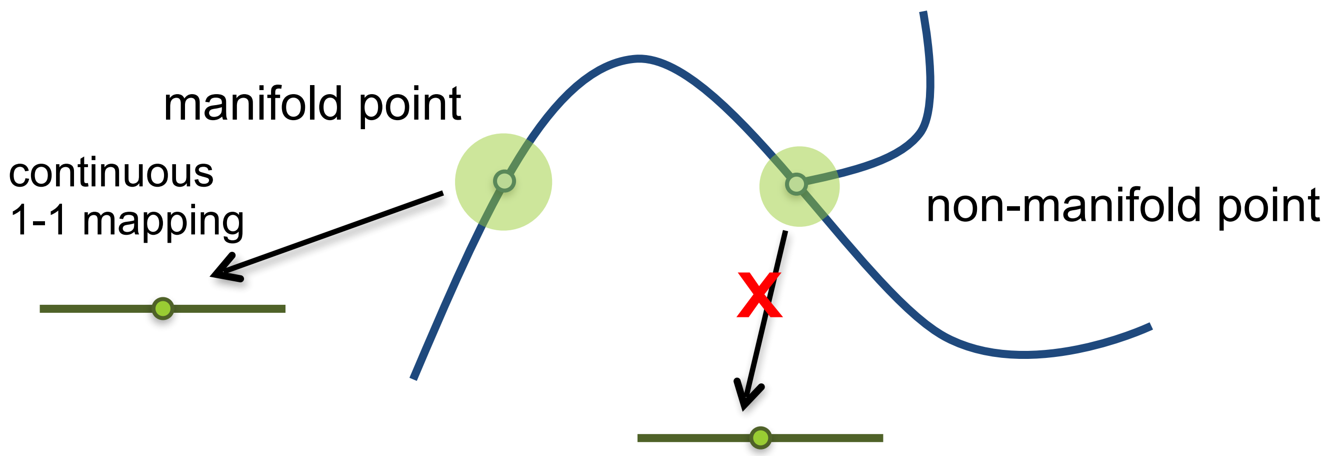
Differential Geometry Basics

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



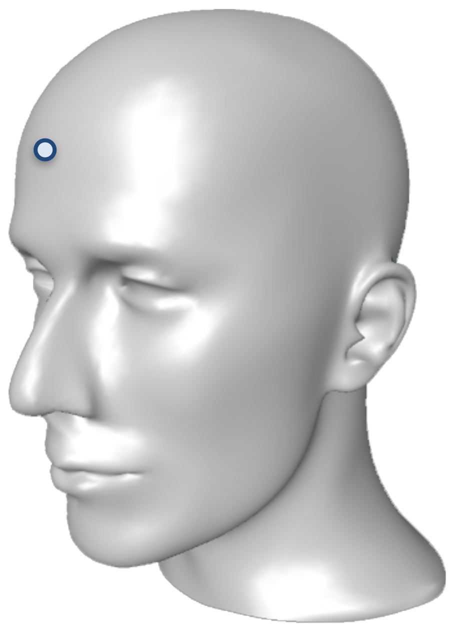
Differential Geometry Basics

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



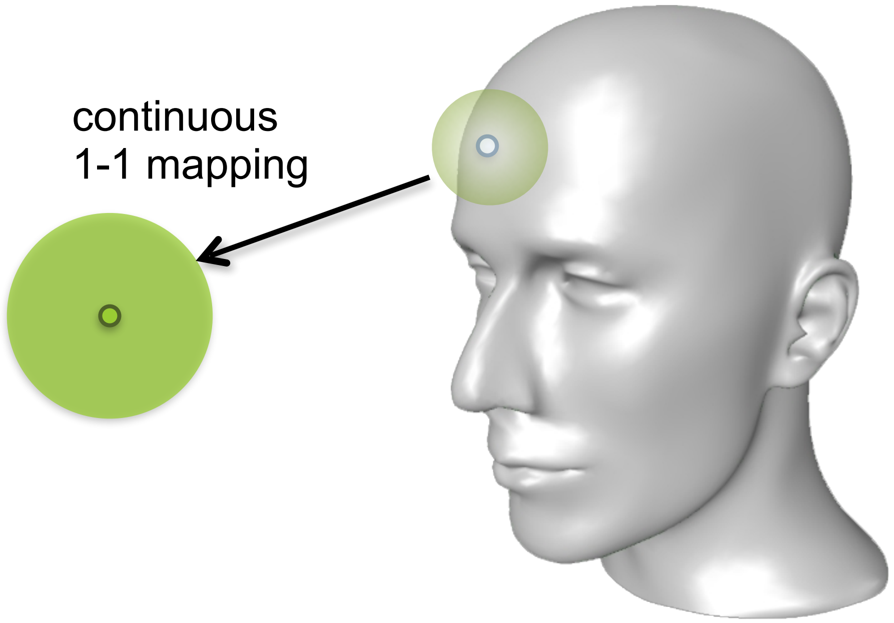
Differential Geometry Basics

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



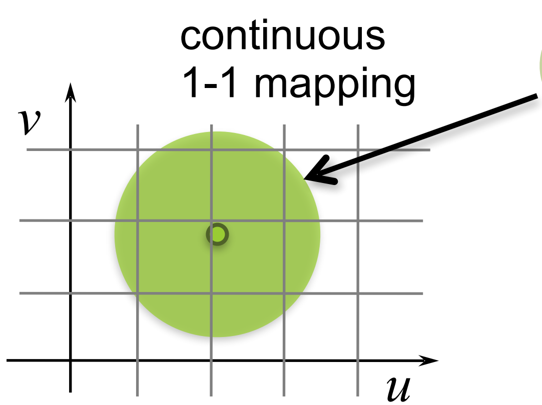
Differential Geometry Basics

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

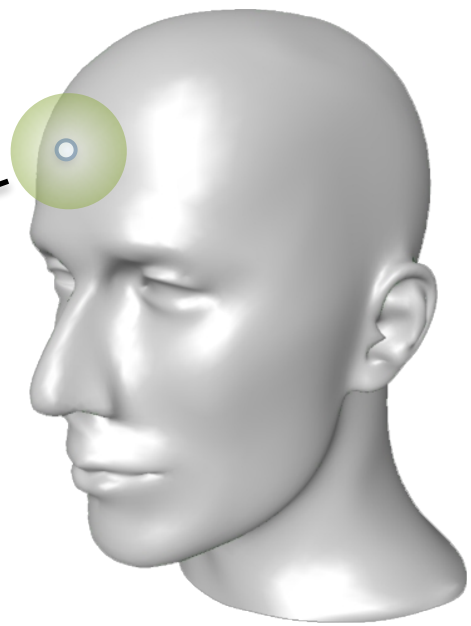


Differential Geometry Basics

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



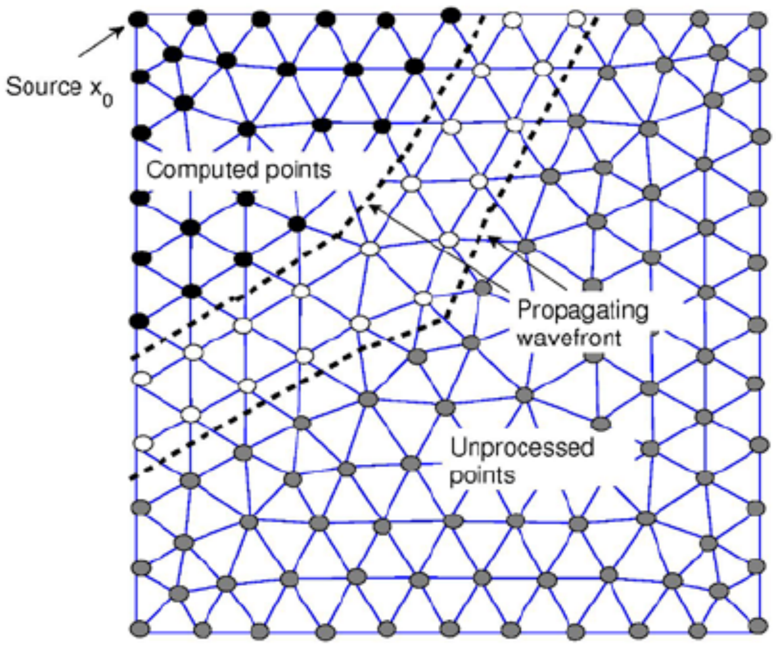
$$\frac{\partial}{\partial u} \quad \frac{\partial}{\partial v} \quad \frac{\partial^2}{\partial^2 u} \quad \frac{\partial^2}{\partial u \partial v} \quad \dots$$

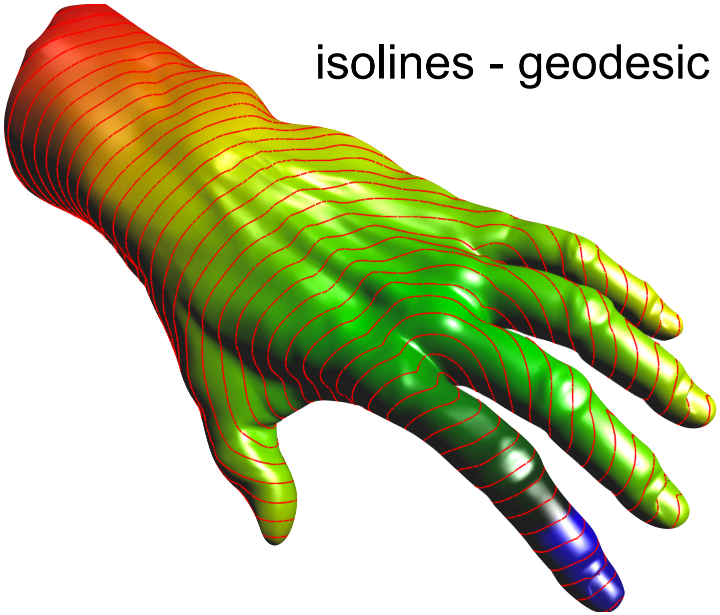


If a sufficiently smooth mapping can be constructed, we can look at its first and second derivatives

Tangents, normals, curvatures, curve angles, distances

Example: Local Distance



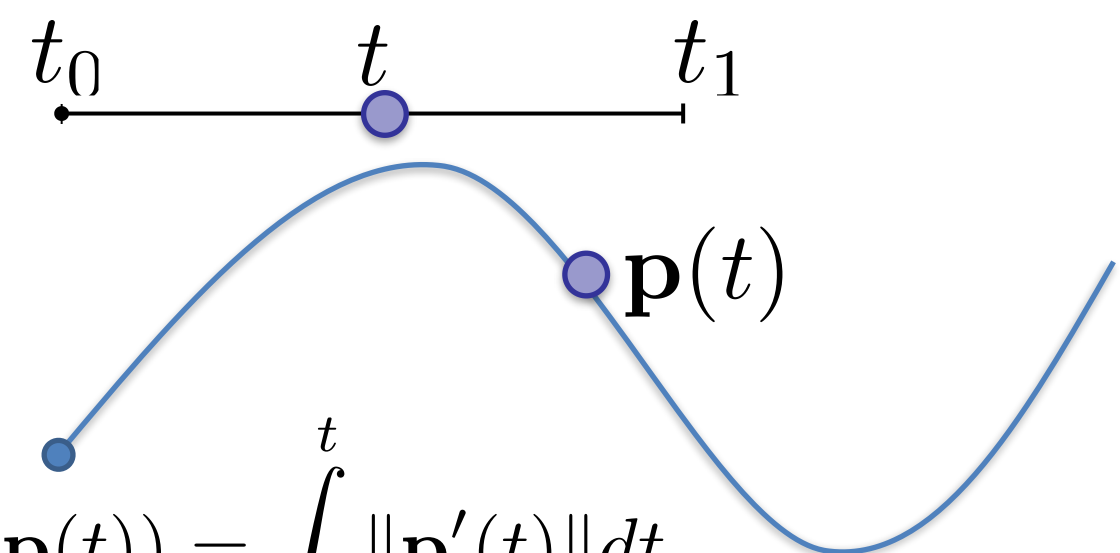


isolines - geodesic

another important example: curvature!

Curves

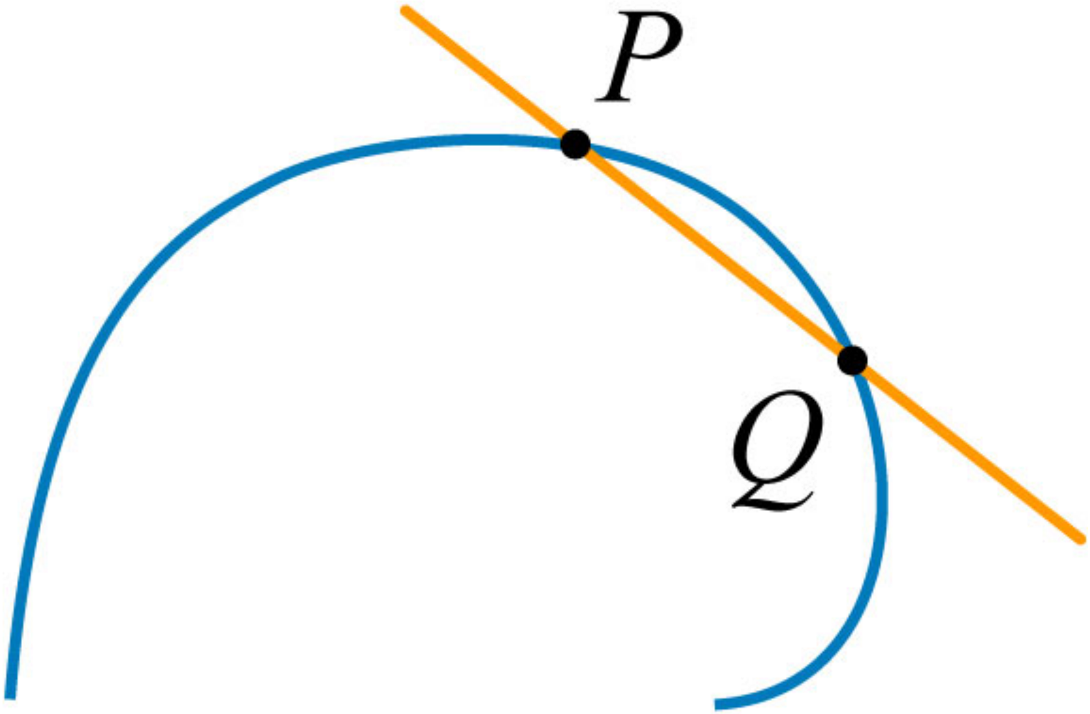
- 2D: $\mathbf{p}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}, \ t \in [t_0, t_1]$
- $\mathbf{p}(t)$ must be continuous



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

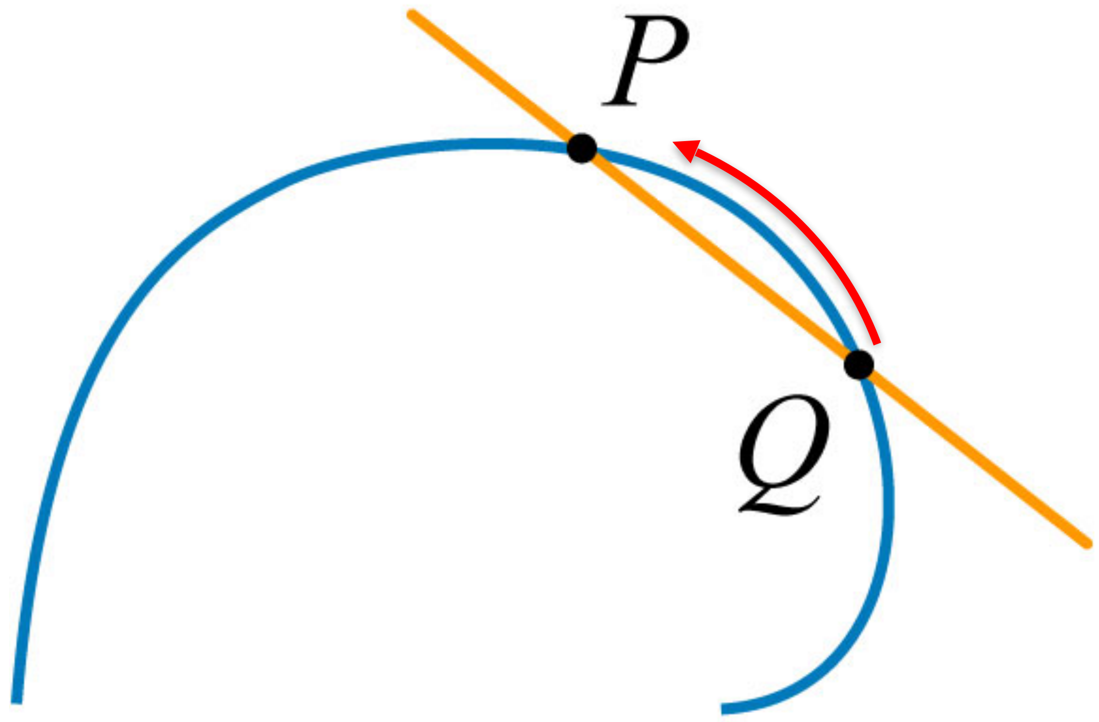
Secant

- A line through two points on the curve.



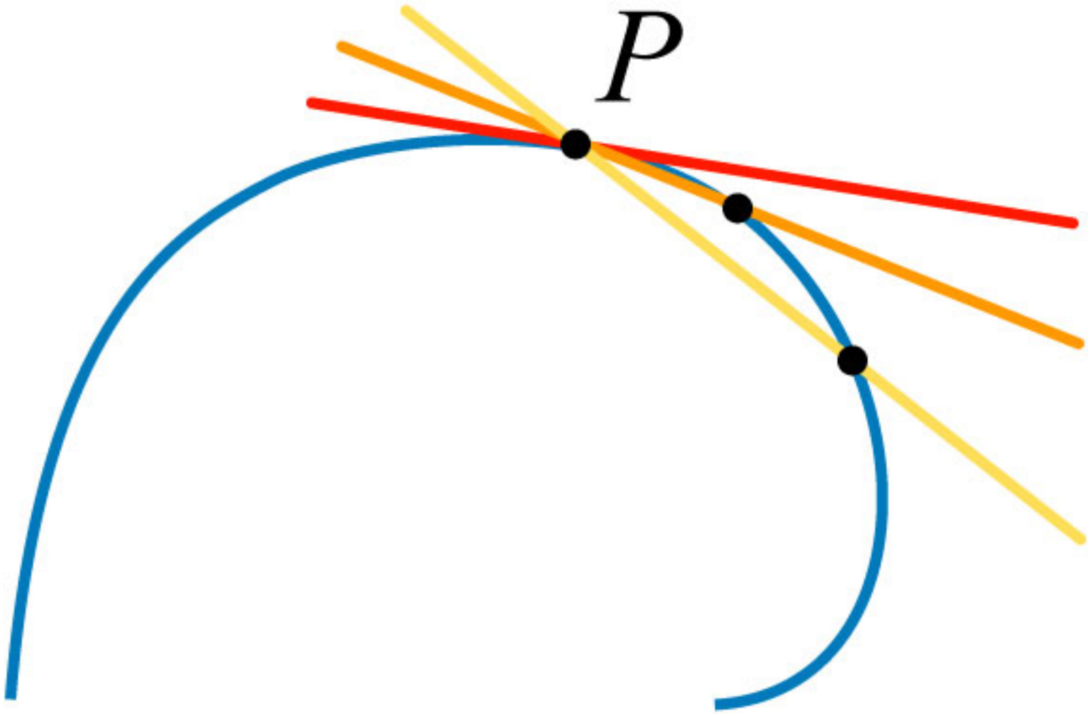
Secant

- A line through two points on the curve.



Tangent

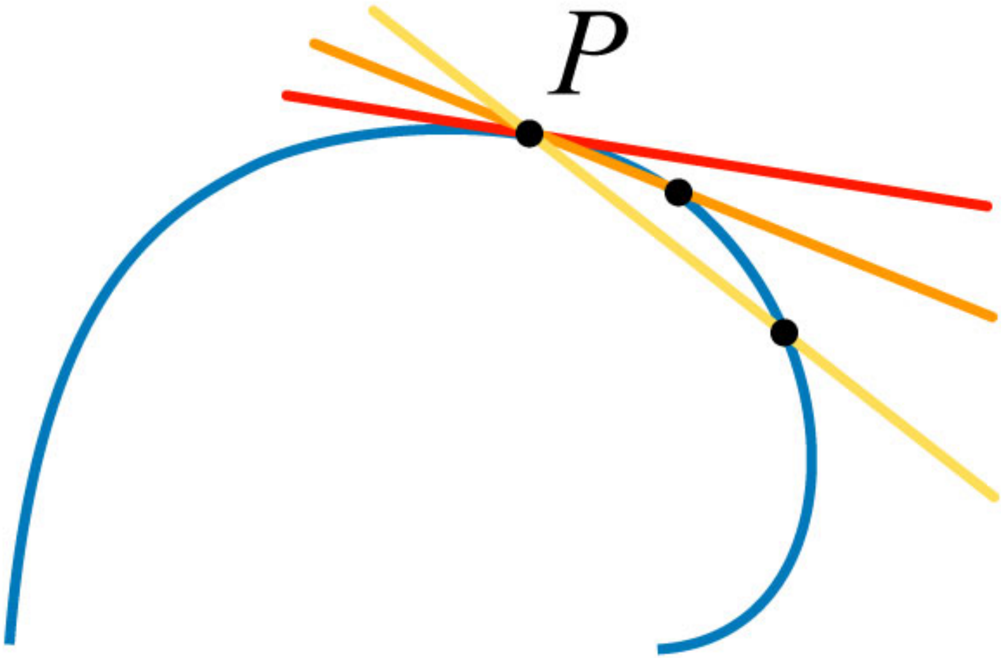
- The limiting secant as the two points come together.



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:

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



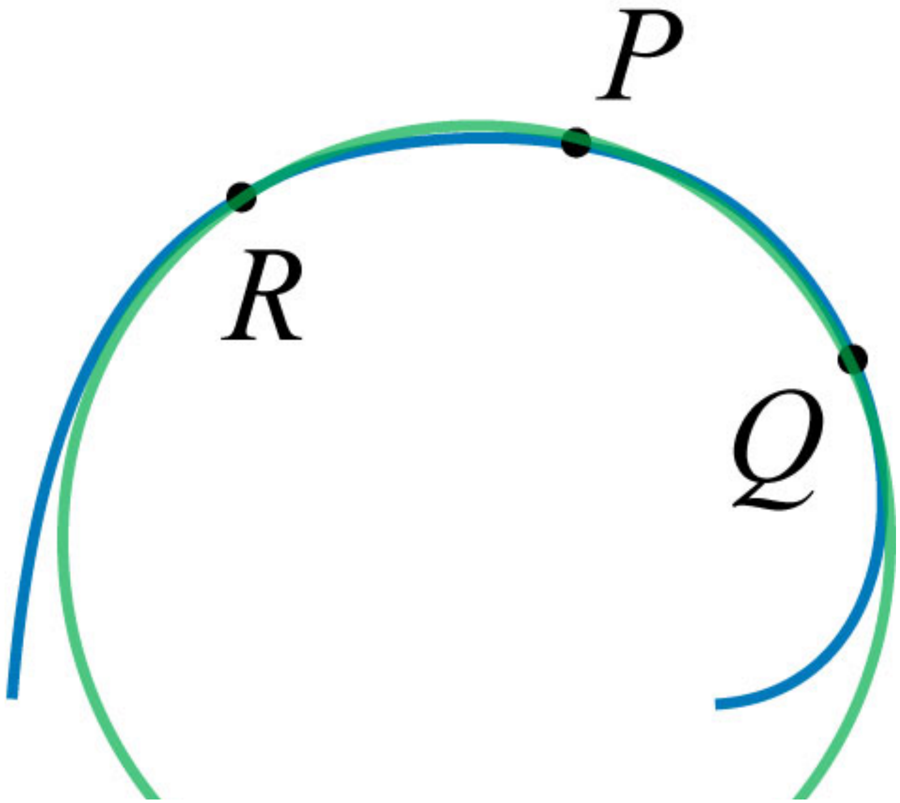
Recall

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

curve “geodesic”

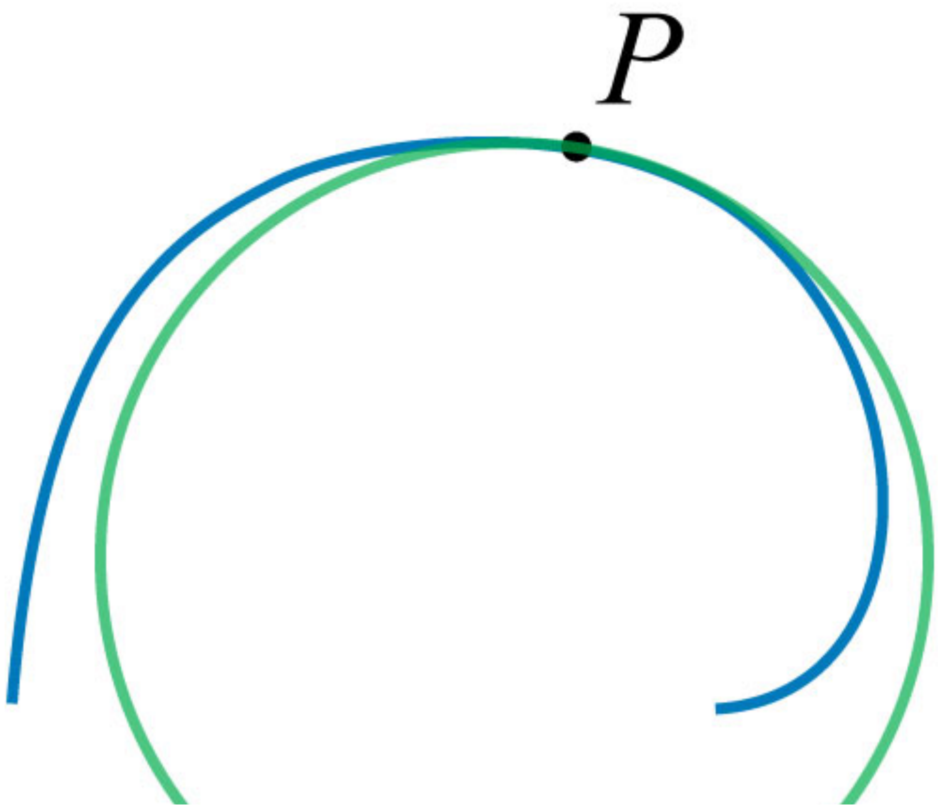
Circle of Curvature

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

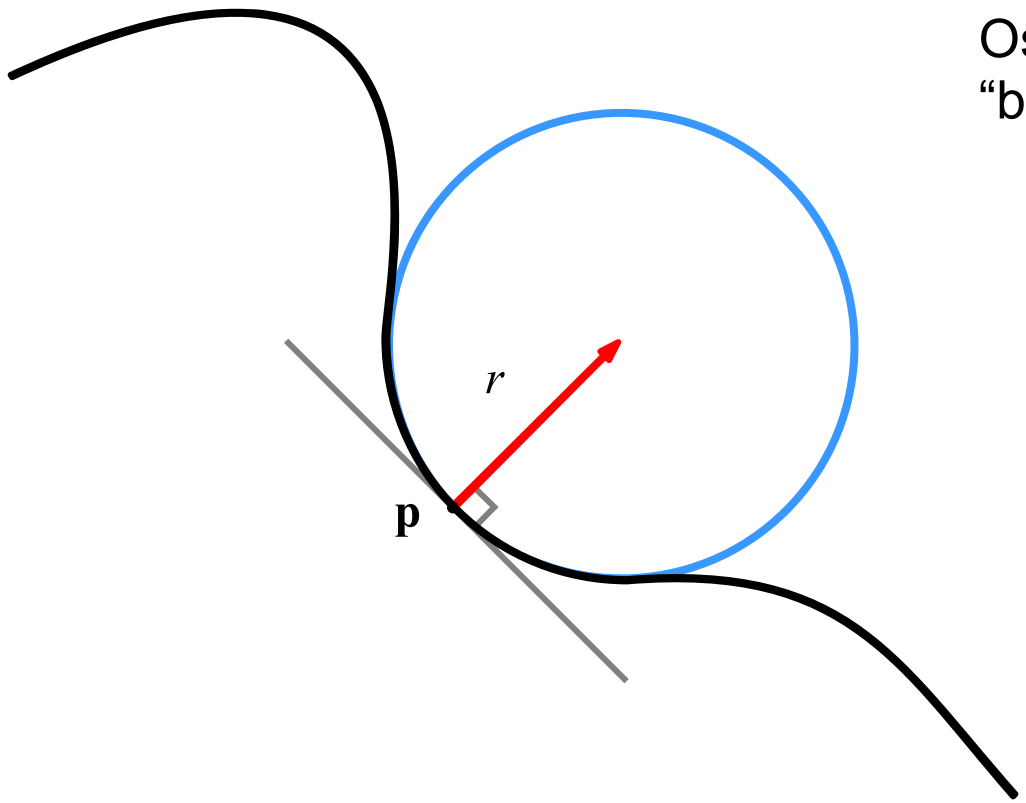


Circle of Curvature

- …the limiting circle as three points come together.



Tangent, normal, radius of curvature

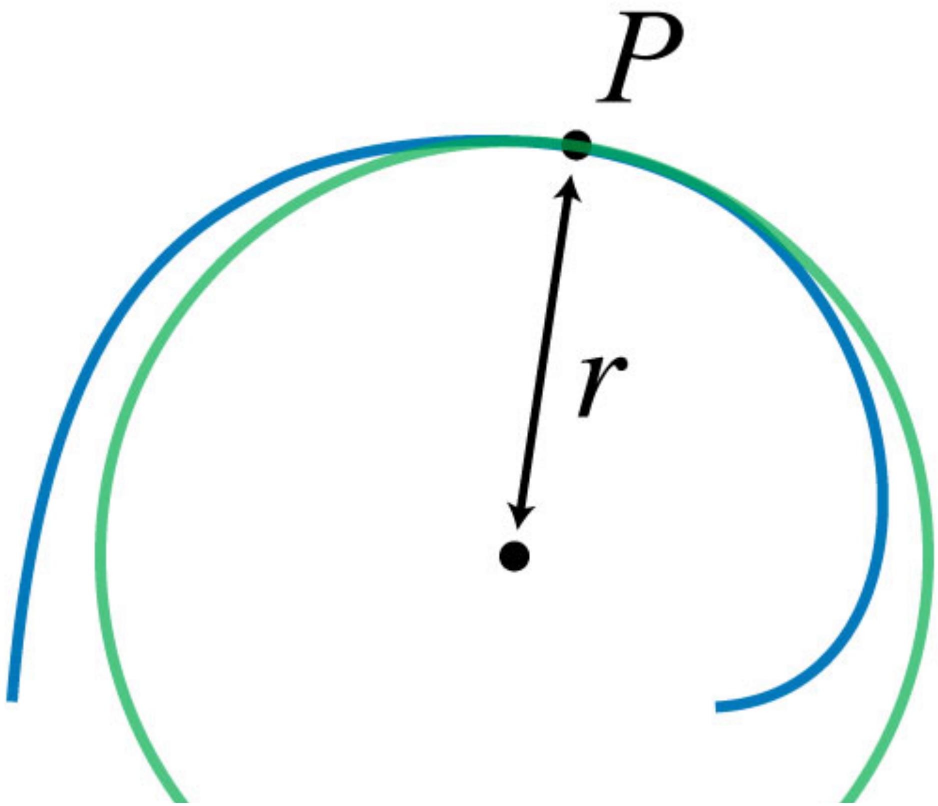


Osculating circle
"best fitting circle"

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

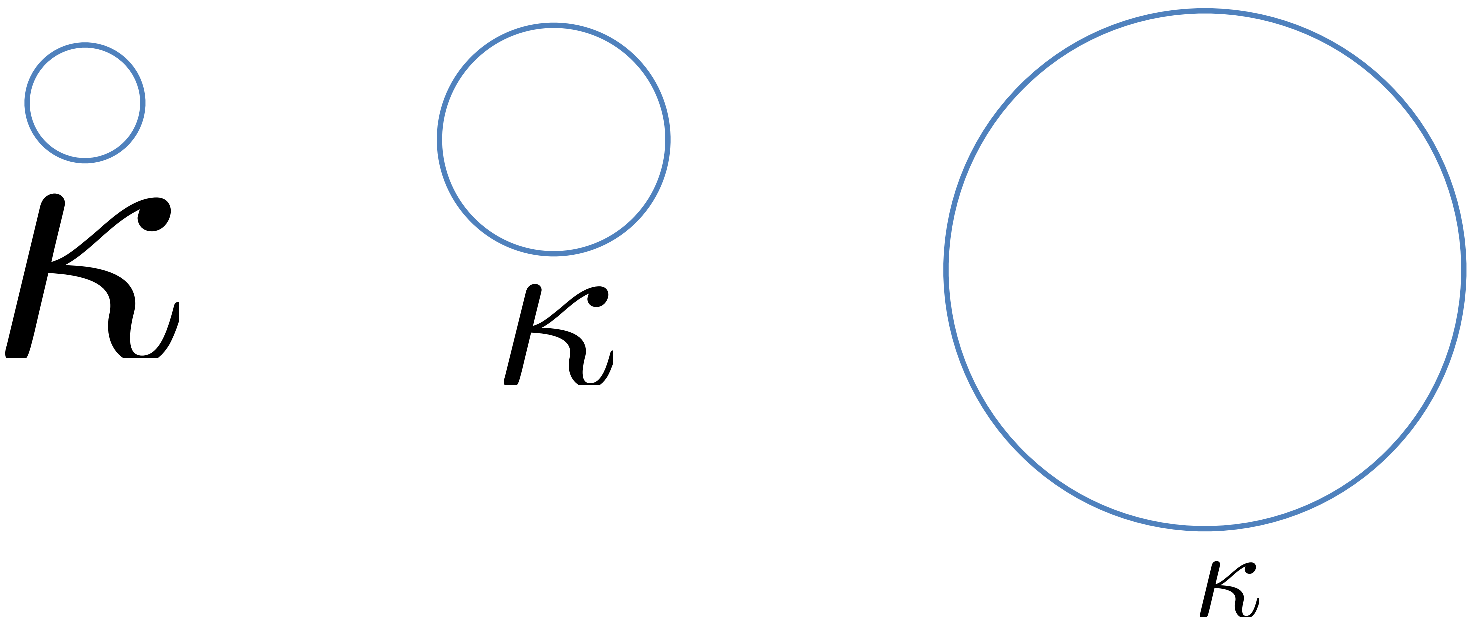
Curvature

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



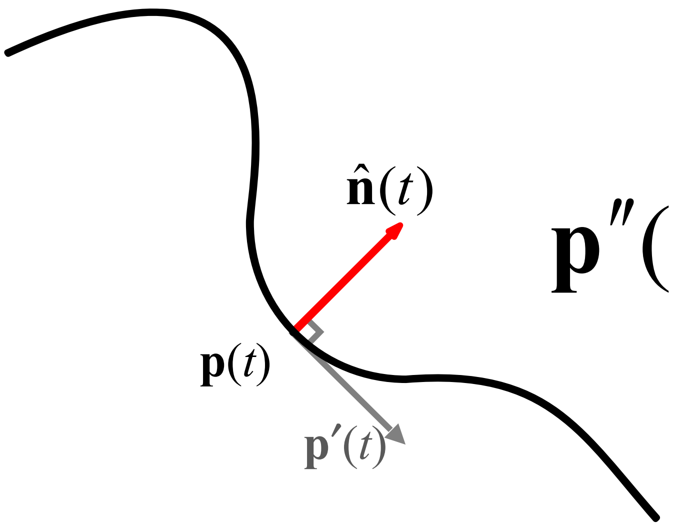
Curvature is scale dependent

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

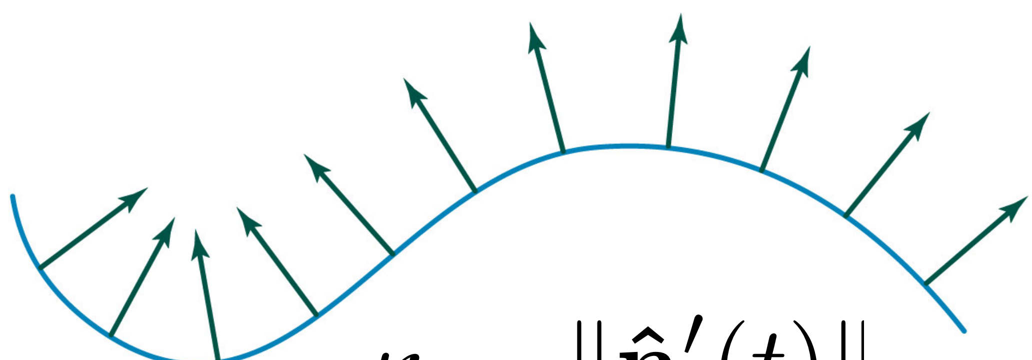


Curvature and Normal

- Assuming t is arc-length parameter:

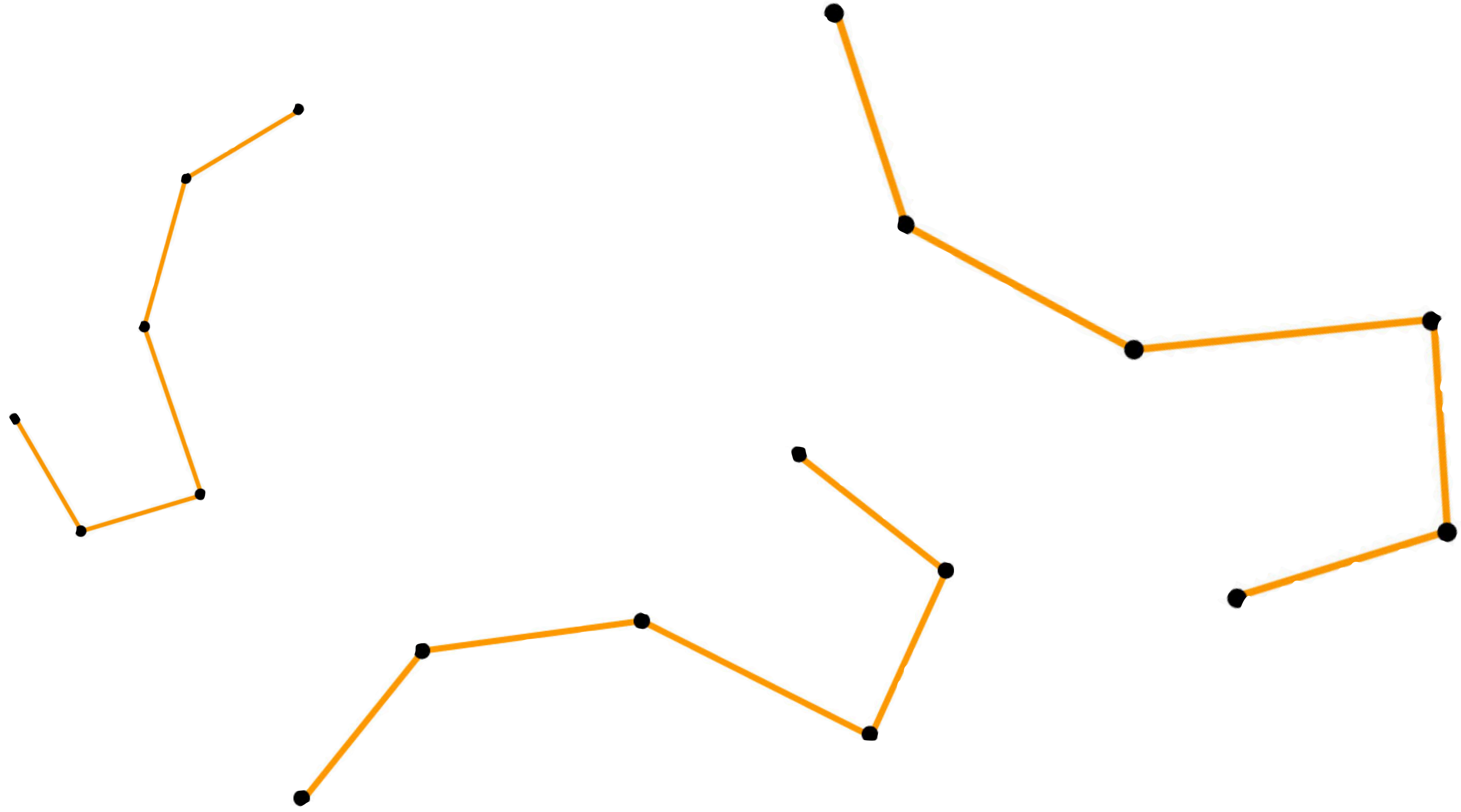


$$\mathbf{p}''(t) = \kappa\hat{\mathbf{n}}(t)$$ normal to the curve



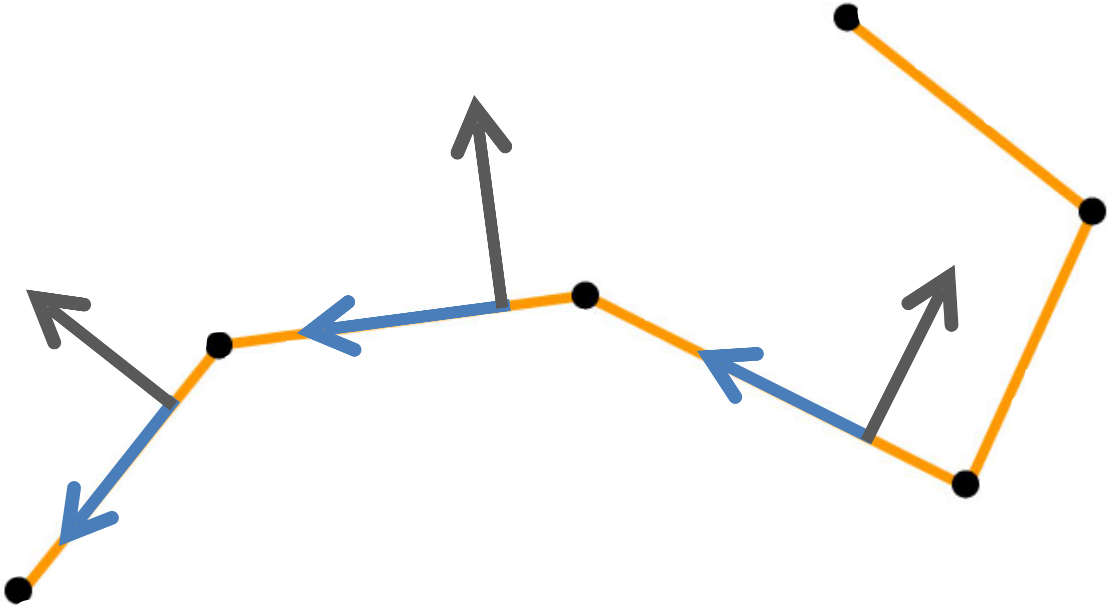
$$\kappa = \|\hat{\mathbf{n}}'(t)\|$$

Discrete Planar Curves



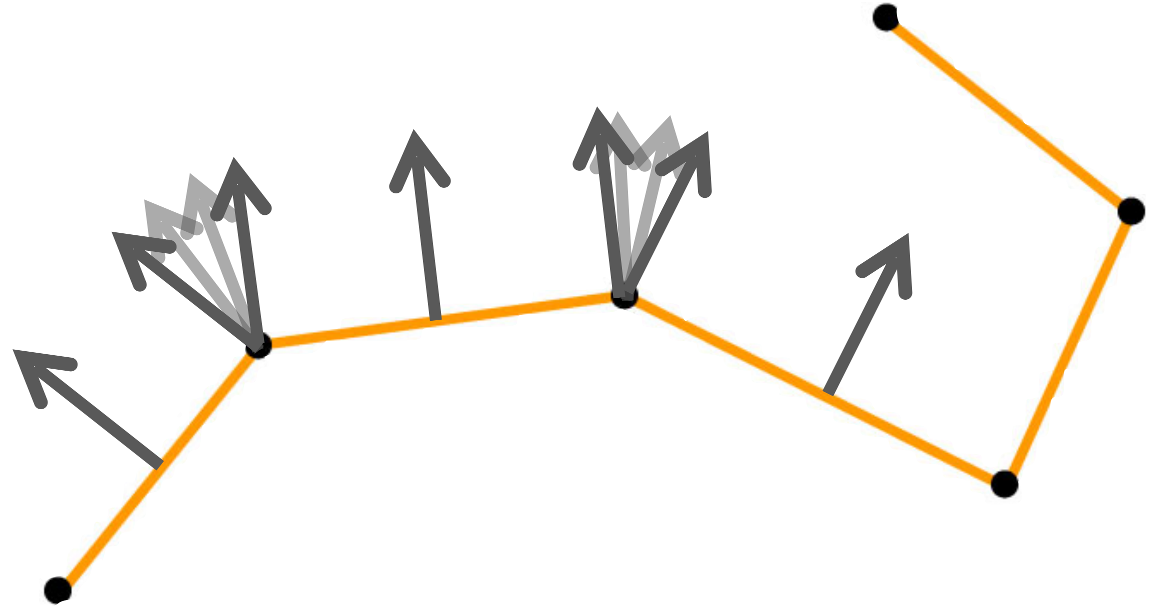
Tangents, Normals

- For any point on the edge, the tangent is simply the unit vector along the edge and the normal is the perpendicular vector



Tangents, Normals

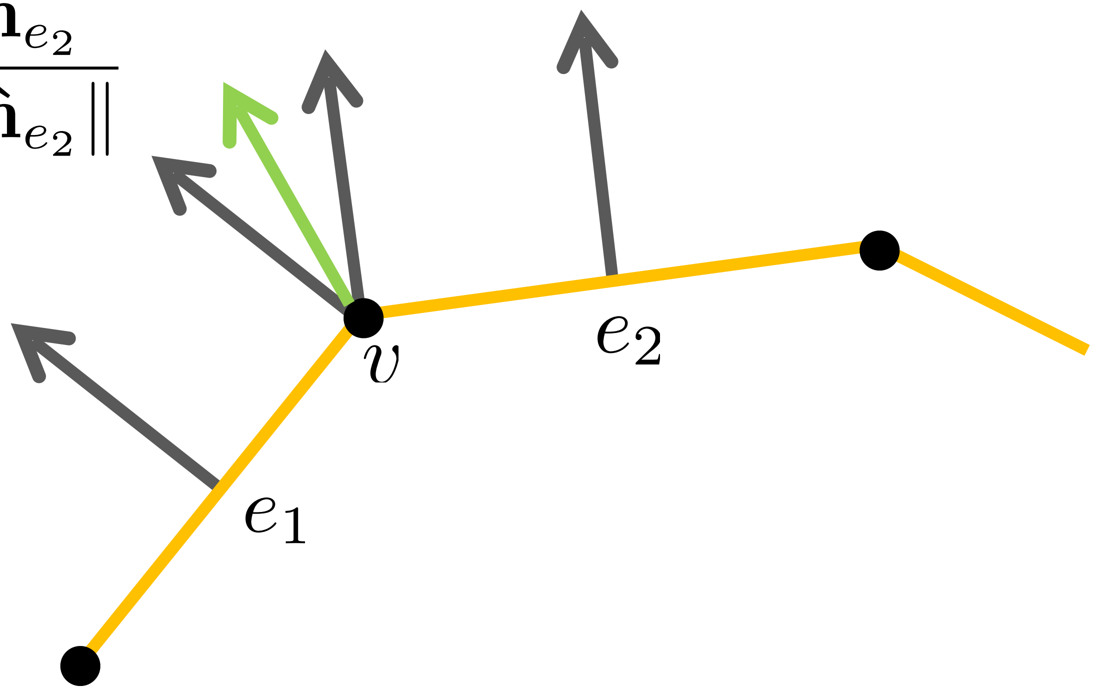
- For vertices, we have many options



Tangents, Normals

- Can choose to average the adjacent edge normals

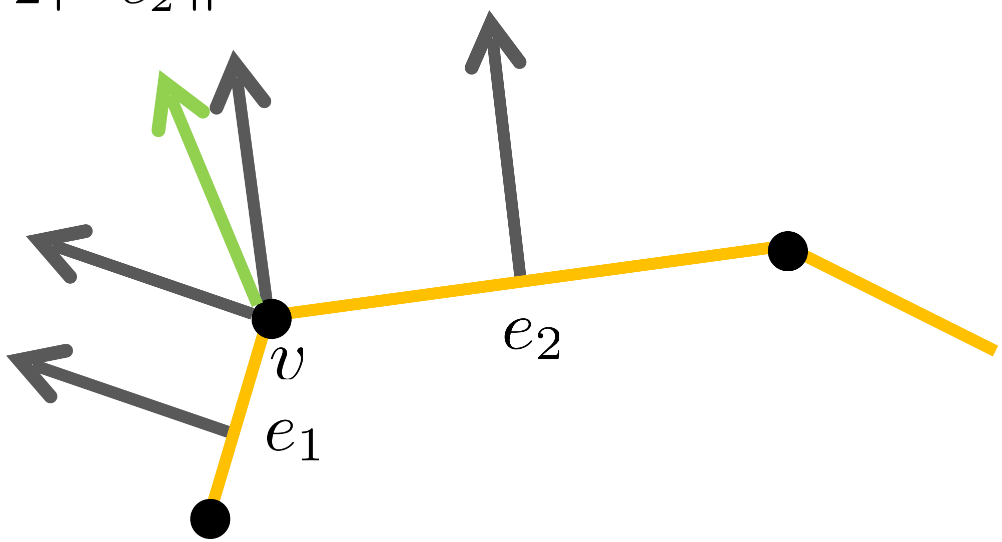
$$\hat{\mathbf{n}}_v = \frac{\hat{\mathbf{n}}_{e_1} + \hat{\mathbf{n}}_{e_2}}{\|\hat{\mathbf{n}}_{e_1} + \hat{\mathbf{n}}_{e_2}\|}$$



Tangents, Normals

- Weight by edge lengths

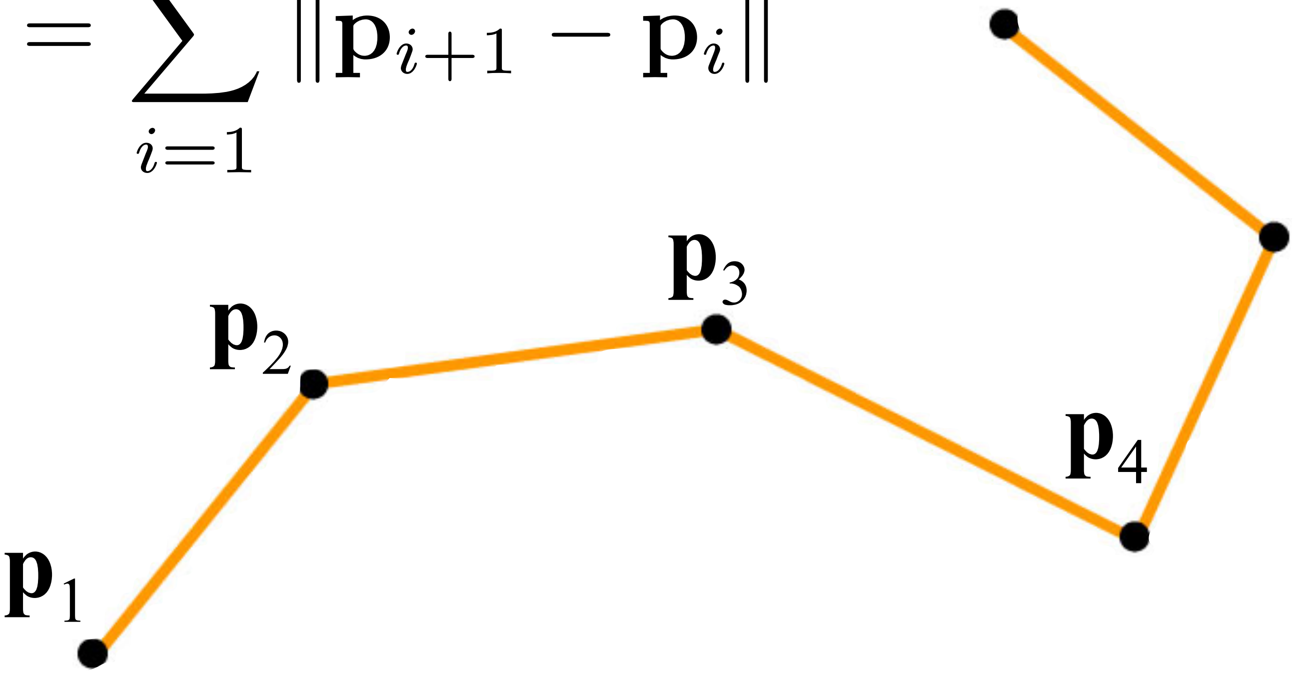
$$\hat{\mathbf{n}}_v = \frac{|e_1|\hat{\mathbf{n}}_{e_1} + |e_2|\hat{\mathbf{n}}_{e_2}}{\||e_1|\hat{\mathbf{n}}_{e_1} + |e_2|\hat{\mathbf{n}}_{e_2}\|}$$



The Length of a Discrete Curve

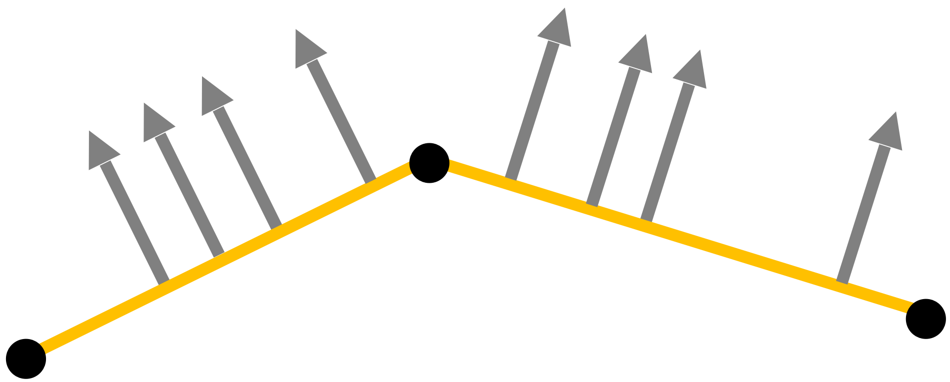
- Sum of edge lengths

$$\text{len}(p) = \sum_{i=1}^{n-1} \|\mathbf{p}_{i+1} - \mathbf{p}_i\|$$



Curvature of a Discrete Curve

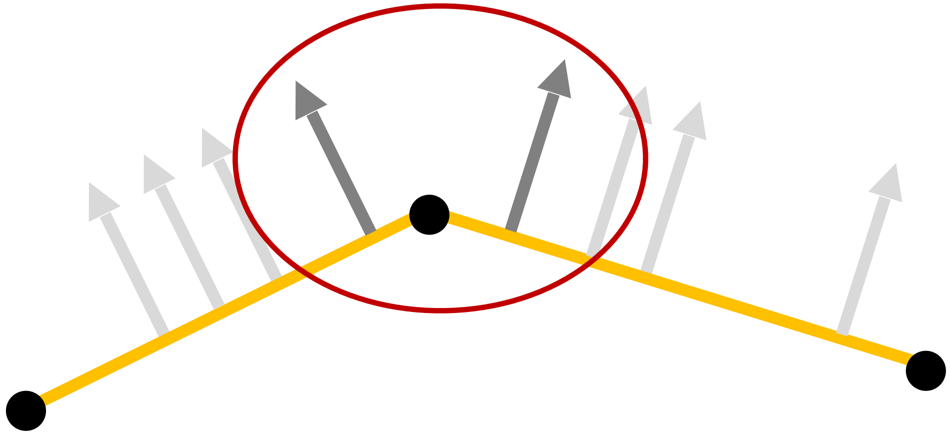
- Curvature is the change in normal direction as we travel along the curve



no change along each edge –
curvature is zero along edges

Curvature of a Discrete Curve

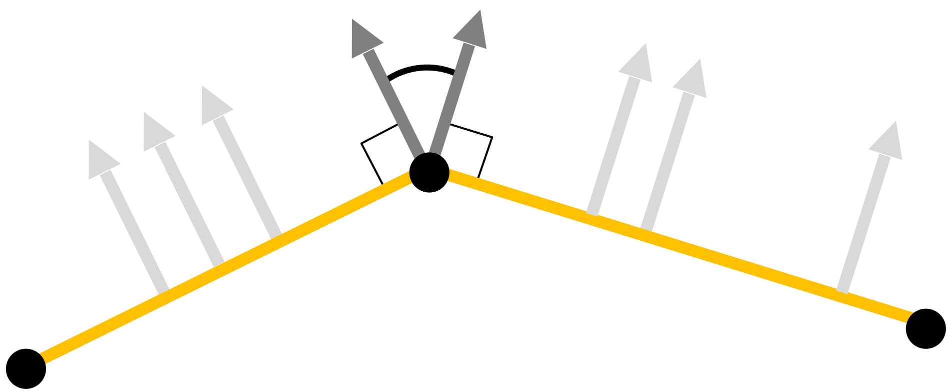
- Curvature is the change in normal direction as we travel along the curve



normal changes at vertices –
record the turning angle!

Curvature of a Discrete Curve

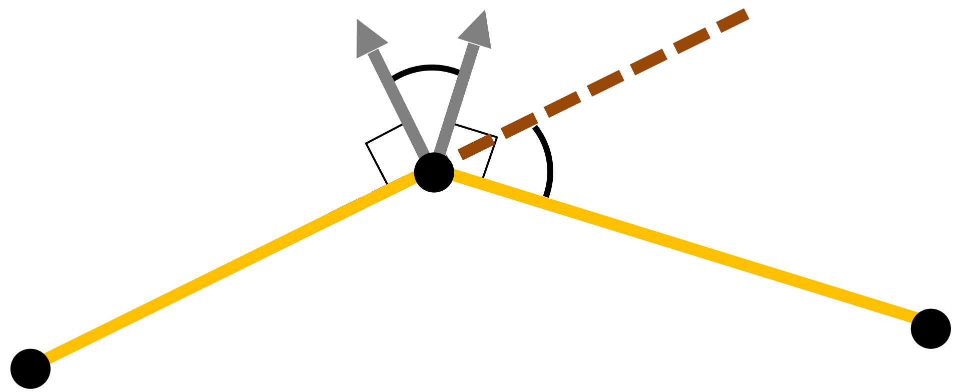
- Curvature is the change in normal direction as we travel along the curve



normal changes at vertices –
record the turning angle!

Curvature of a Discrete Curve

- Curvature is the change in normal direction as we travel along the curve



same as the turning angle
between the edges

Curvature of a Discrete Curve

- Zero along the edges
- Turning angle at the vertices
  = the change in normal direction