

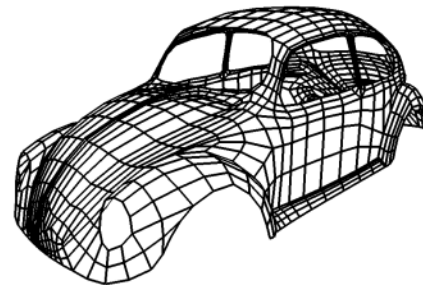
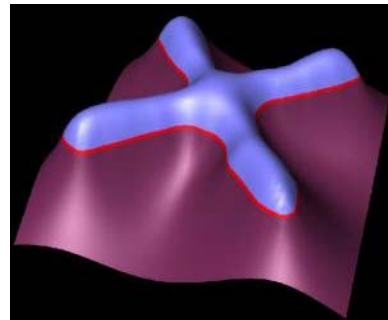
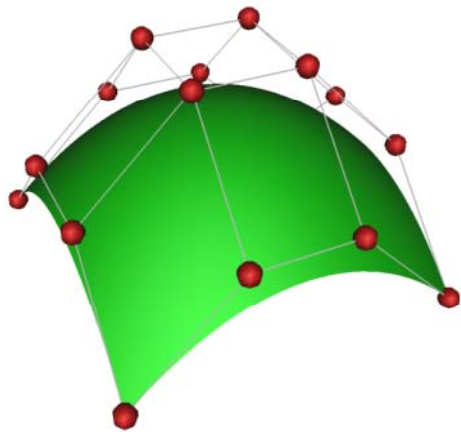
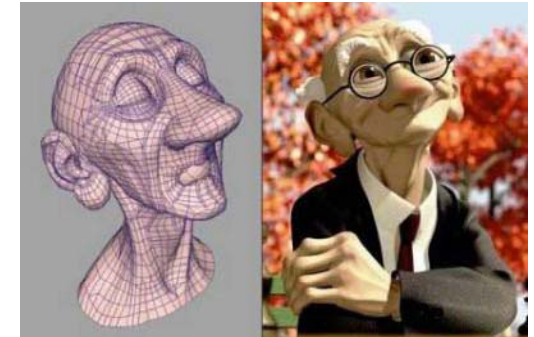
CS 523: Computer Graphics, Spring 2009

Shape Modeling

Shape Representations

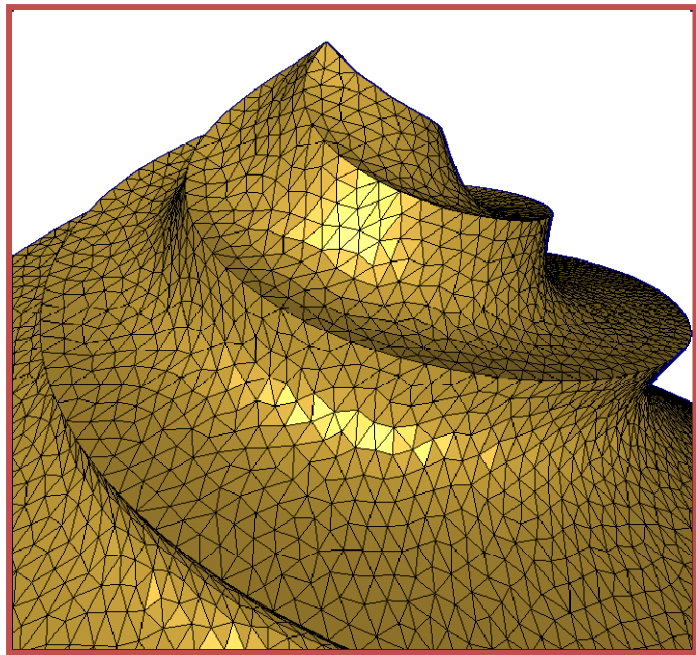
Course topics

- Shape representation
 - Points
 - Parametric surfaces
 - Implicits

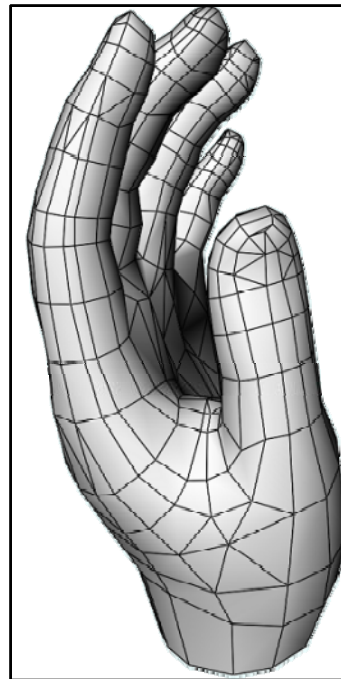


Course topics

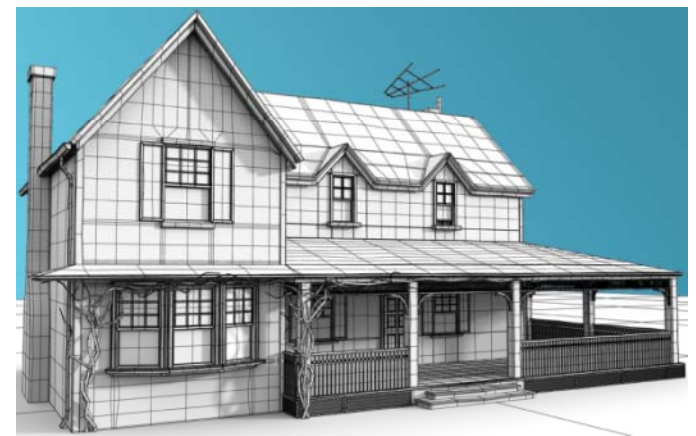
- Shape representation
 - Subdivision surfaces
 - Polygonal meshes



Andrew Nealen, Rutgers, 2009

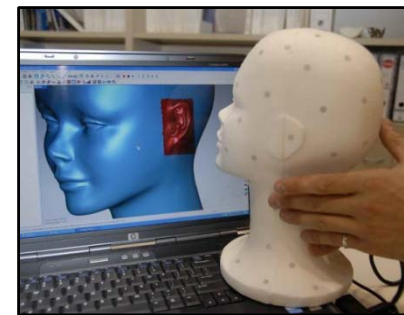
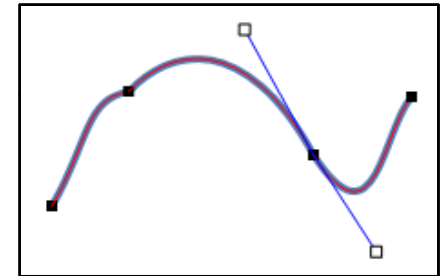
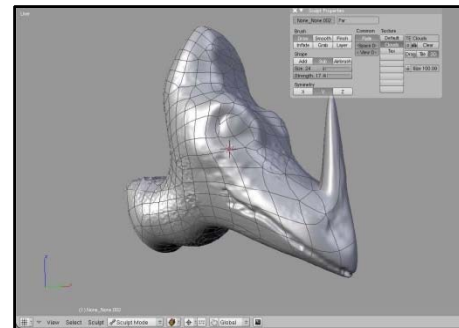


1/28/2009



Shape representation

- Where does the shape come from?
- Modeling “by hand”:
 - Higher-level representations, amendable to modification, control
 - Parametric surfaces, subdivision surfaces, implicits
- Acquired real-world objects:
 - Discrete sampling
 - Points, meshes



Points

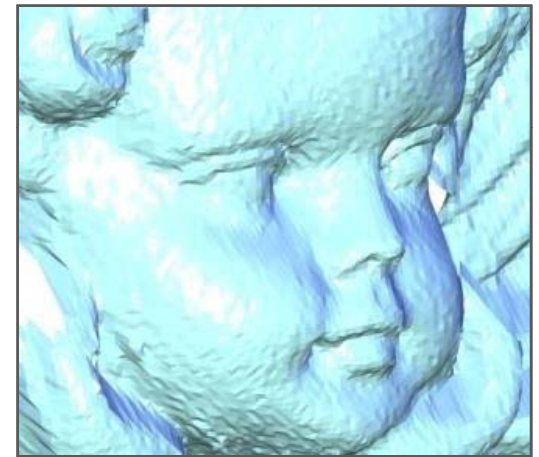
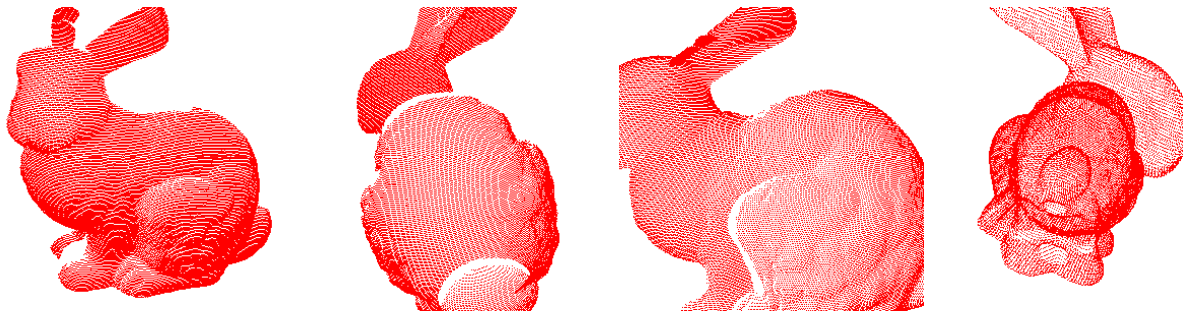
Shape acquisition

Sampling of real world objects

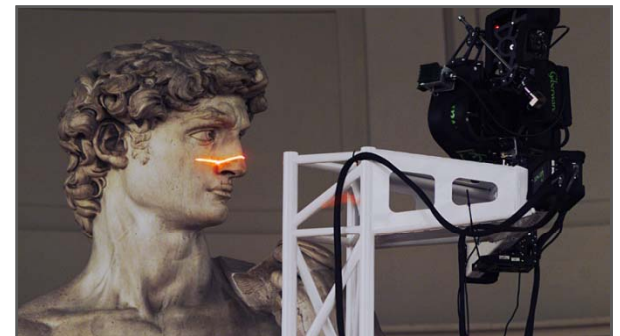


Points

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

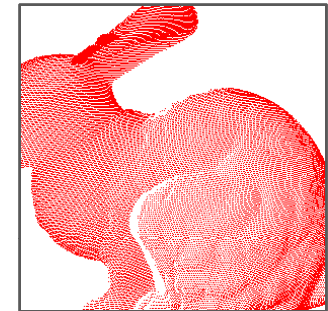


- Depth imaging (e.g. by triangulation)
- Registration of multiple images



Points

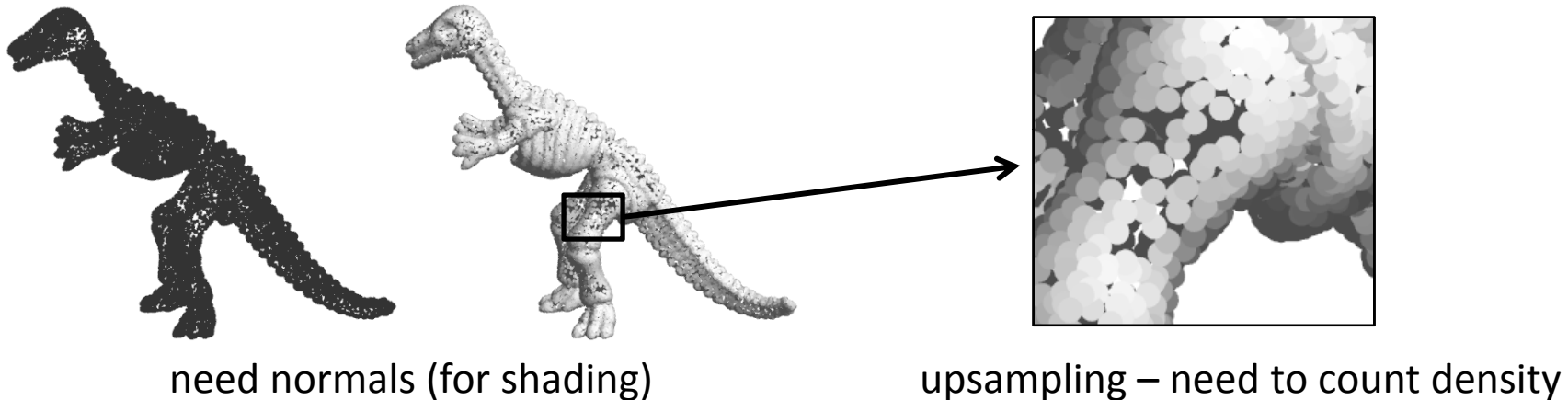
- points = unordered set of 3-tuples
- Often converted to other reps
 - Meshes, implicits, parametric surfaces
 - Easier to process, edit and/or render
- Efficient point processing and modeling requires a spatial partitioning data structure
 - To figure out neighborhoods



Points

Neighborhood information

- Why do we need neighbors?

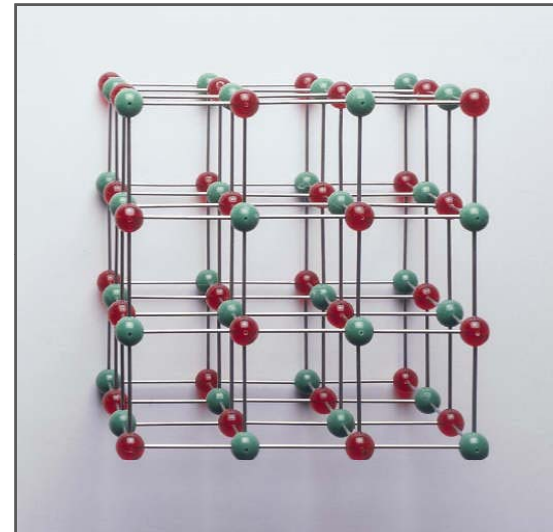


- Need sub-linear implementations of
 - k-nearest neighbors to point \mathbf{x}
 - In radius search $\|\mathbf{p}_i - \mathbf{x}\| < \varepsilon$

Spatial Data Structures

Commonly used for point processing

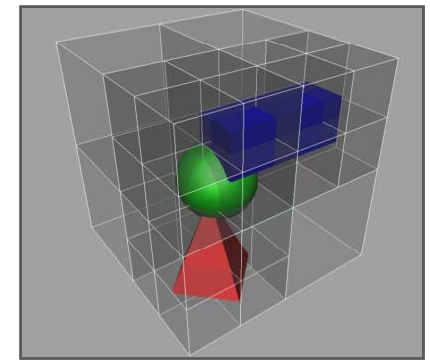
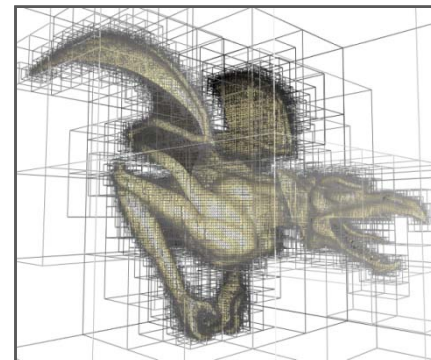
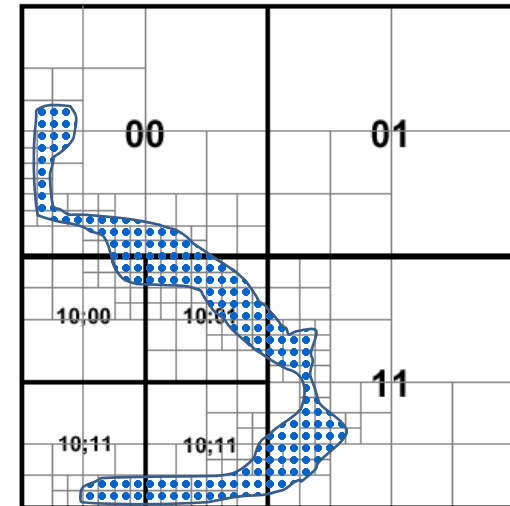
- Regular uniform 3D lattice
 - Simple point insertion by coordinate discretization
 - Simple proximity queries by searching neighboring cells
- Determining lattice parameters (i.e. cell dimensions) is nontrivial
- Generally unbalanced, i.e. many empty cells



Spatial Data Structures

Commonly used for point processing

- Octree
 - Splits each cell into 8 equal cells
 - Adaptive, i.e. only splits when too many points in cell
 - Proximity search by (recursive) tree traversal and distance to neighboring cells
 - Tree might not be balanced

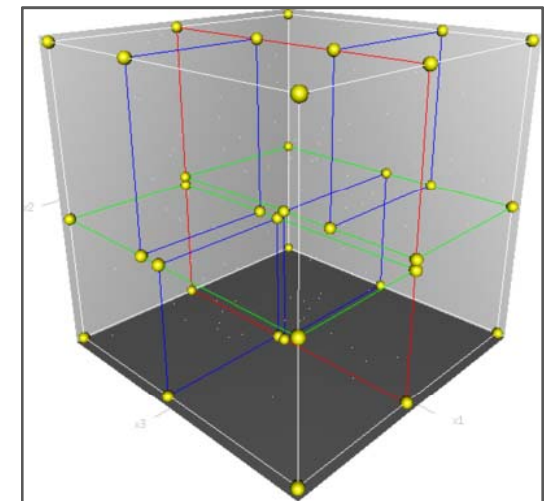
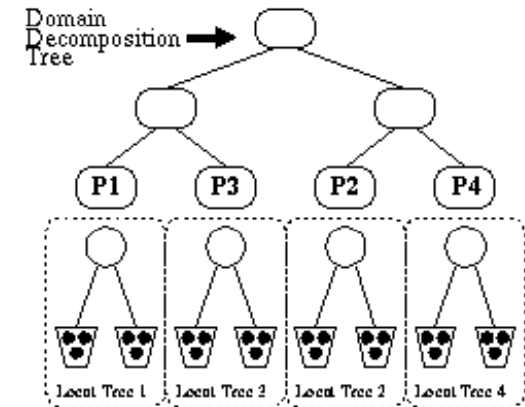
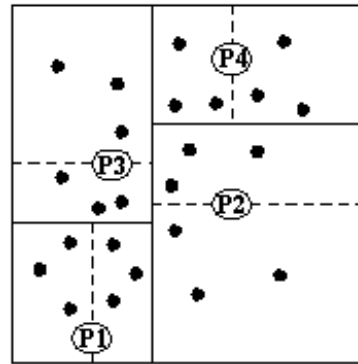


Spatial Data Structures

Commonly used for point processing

■ Kd-Tree

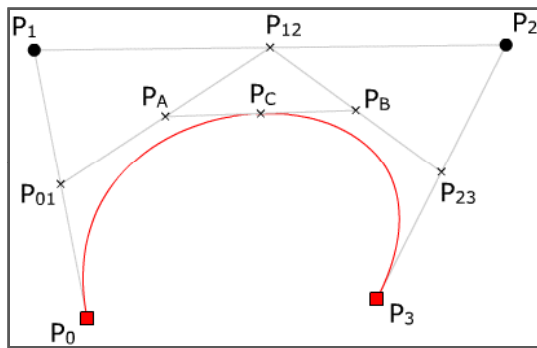
- Each cell is individually split along the median into two cells
- Same amount of points in cells
- Perfectly balanced tree
- Proximity search similar to the recursive search in an Octree
- More data storage required for inhomogeneous cell dimensions



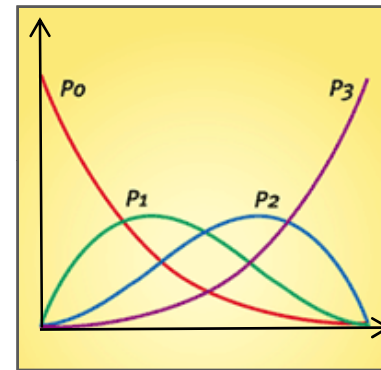
Parametric Curves and Surfaces

Parametric Curves and Surfaces

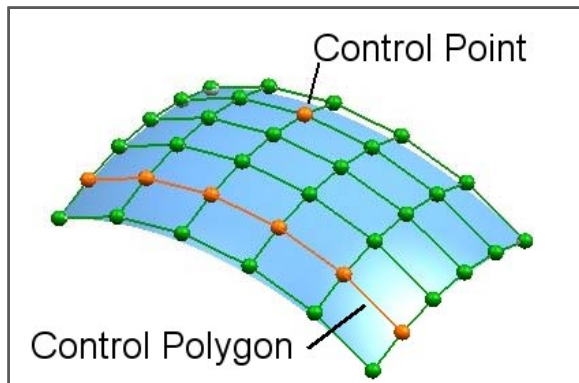
- Curves are 1-dimensional parameterizations



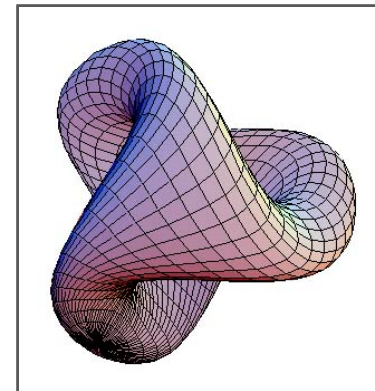
$$S(t) = \mathbf{x}_t$$



- Surfaces are 2-dimensional parameterizations



$$S(x, y) = \mathbf{x}_t$$



Parametric Curves and Surfaces

Examples

- Explicit curve/circle in 2D

$$\mathbf{p} : \mathbb{R} \rightarrow \mathbb{R}^d, d = 1, 2, 3, \dots$$

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

$$\mathbf{p}(t) = r \cdot (\cos(t), \sin(t), 0)$$

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

Parametric Curves and Surfaces

Examples

- Explicit surface/sphere in 3D

$$\mathbf{q} : \mathbb{R}^2 \rightarrow \mathbb{R}^d, d = 1, 2, 3, \dots$$

$$(u, v) \mapsto \mathbf{q}(u, v) = (x(u, v), y(u, v), z(u, v))$$

$$\mathbf{p}(u, v) = r \cdot (\cos(u) \cos(v), \sin(u) \cos(v), \sin(v))$$

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

Parametric Curves

Continuity and regularity

- Curve segment $\mathbf{p} : [a, b] \rightarrow R^d, d = 1, 2, 3, \dots$

- The same segment can be parameterized differently



$$\mathbf{p}_1 : [0, 1] \rightarrow R^3, \mathbf{p}(t) = tP_1 + (1 - t)P_2$$

$$\mathbf{p}_2 : [0, 1] \rightarrow R^3, \mathbf{p}(t) = t^2P_1 + (1 - t^2)P_2$$

- A parametric curve is n -times continuously differentiable if the image \mathbf{p} is n -times continuously differentiable (C^n)
- The derivative $\mathbf{p}'(t)$ at position t is a tangent vector
- A curve is regular when \mathbf{p} is differentiable and $\mathbf{p}'(t) \neq \mathbf{0}$

Parametric Curves

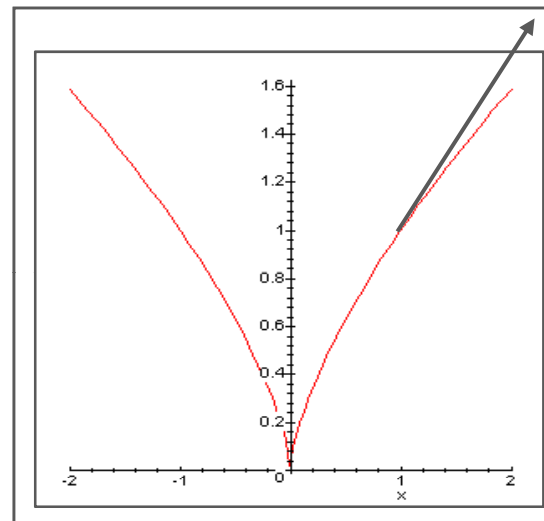
Continuity and regularity

- Example

$$\mathbf{p} : [-2, 2] \rightarrow \mathbb{R}^3, \mathbf{p}(t) = (t^3, t^2, 0)$$

$$\mathbf{p}'(t) = (3t^2, 2t, 0) \Rightarrow \mathbf{p}'(0) = \mathbf{0}$$

- \mathbf{p} is continuously differentiable, but not regular at position $t = 0$
- The regularity of a curve can be expressed as its visual smoothness
- The tangent vector can be interpreted as the velocity (compare to physics $\mathbf{v} = \mathbf{s}'$)



Parametric Curves

Arc length parameterization

- A curve is parameterized by arc length when

$$\|\mathbf{p}'(t)\| = 1, \quad t \in [a, b]$$

- Any regular curve can be parameterized by arc length
- For arc length parameterized curves:

$$T(s) := \mathbf{p}'(s) \quad \text{Tangent vector}$$

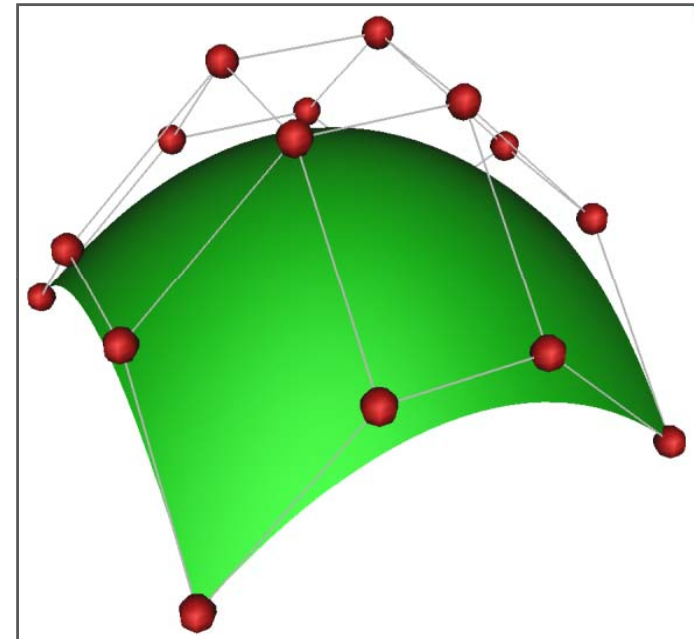
$$K(s) := \mathbf{p}''(s) \quad \text{Curvature vector}$$

$$\kappa(s) := \|\mathbf{p}''(s)\| \quad \text{Curvature (scalar)}$$

Parametric Surfaces

Tensor product surfaces

- Example: Bezier surfaces
 - Surface lies in convex hull of control points
 - Surface interpolates the four corner control points
 - Boundary curves are Bezier curves defined only by control points on the boundary
- Other: B-Spline patches, NURBS, etc...



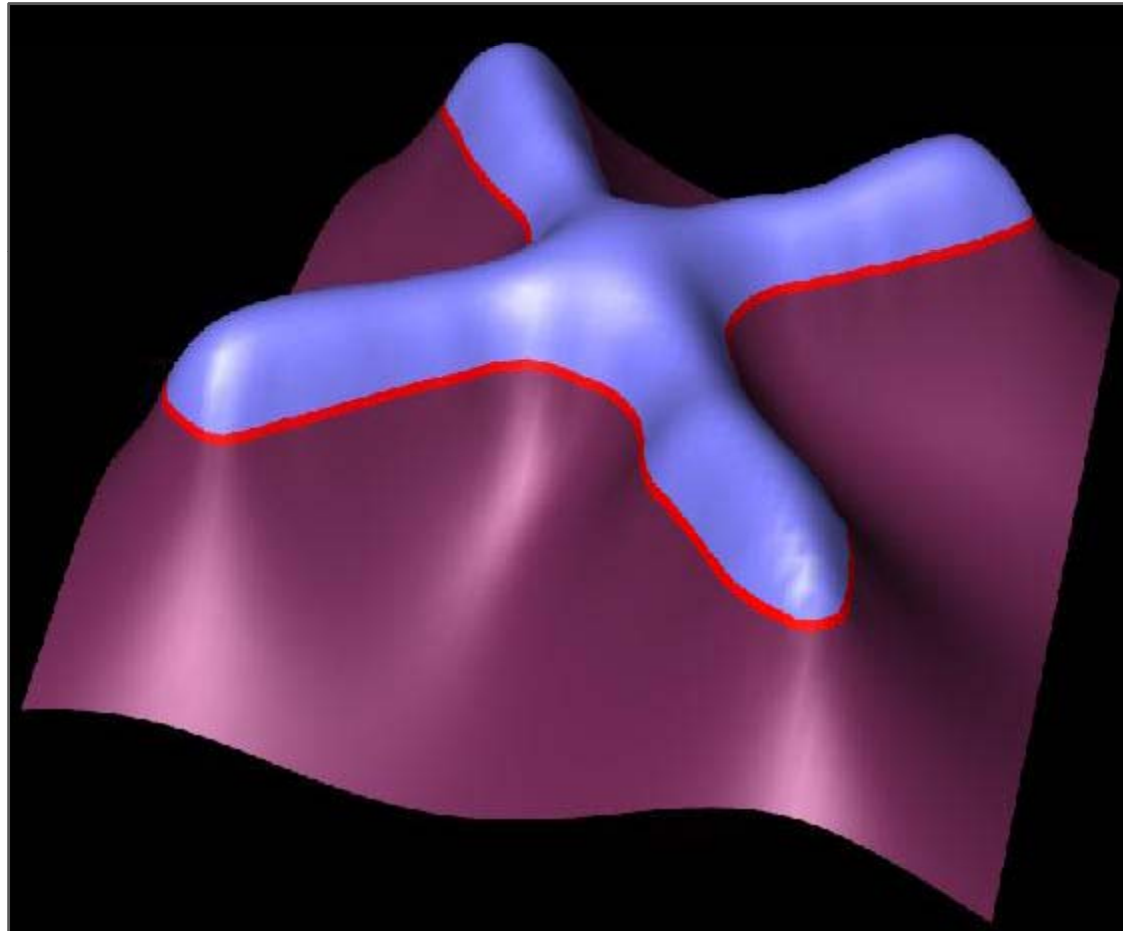
Parametric Curves and Surfaces

- 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

Implicit Curves and Surfaces

Implicit Curves and Surfaces

Illustration



Implicit Curves and Surfaces

Examples

- Implicit circle and sphere

$$f : \mathbb{R}^2 \rightarrow \mathbb{R}$$

$$K = \{\mathbf{p} \in \mathbb{R}^2 : f(\mathbf{p}) = 0\}$$

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

$$g : \mathbb{R}^3 \rightarrow \mathbb{R}$$

$$K = \{\mathbf{p} \in \mathbb{R}^3 : g(\mathbf{p}) = 0\}$$

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

Implicit Curves and Surfaces

Definition

- Definition $g : \mathbb{R}^3 \rightarrow \mathbb{R}$

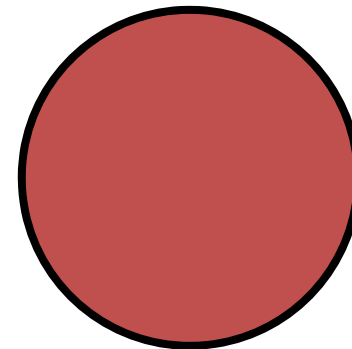
$$K = g^{-1}(0) = \{\mathbf{p} \in \mathbb{R}^3 : g(\mathbf{p}) = 0\}$$

- Space partitioning

$\{\mathbf{p} \in \mathbb{R}^3 : g(\mathbf{p}) < 0\}$ Inside

$\{\mathbf{p} \in \mathbb{R}^3 : g(\mathbf{p}) = 0\}$ Curve/Surface

$\{\mathbf{p} \in \mathbb{R}^3 : g(\mathbf{p}) > 0\}$ Outside



Implicit Curves and Surfaces

Gradient

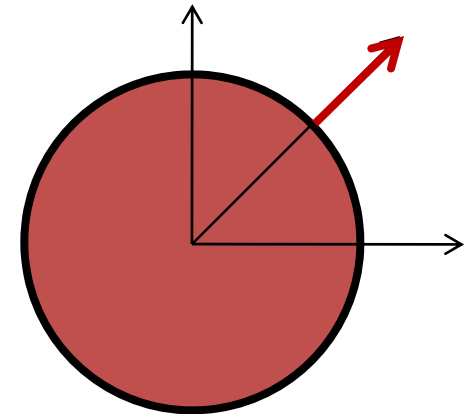
- The normal vector to the surface is given by the gradient of the (scalar) implicit function

$$\nabla g(x, y, z) = \left(\frac{\partial g}{\partial x}, \frac{\partial g}{\partial y}, \frac{\partial g}{\partial z} \right)^T$$

- Example

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

$$\nabla g(x, y, z) = (2x, 2y, 2z)^T$$

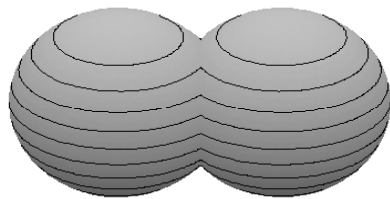


$$\nabla g(x, y, z) = (2, 2, 0)^T$$

Implicit Curves and Surfaces

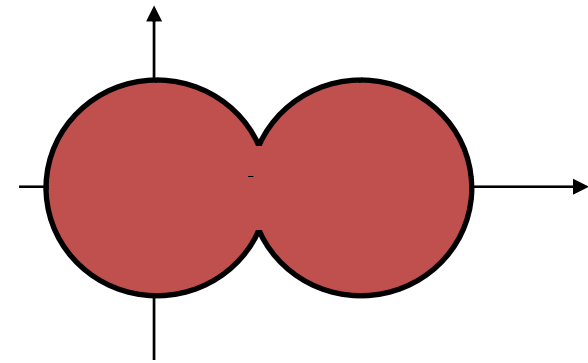
Smooth set operations

- Standard operations: union and intersection



$$\bigcup_i g_i(\mathbf{p}) = \min g_i(\mathbf{p})$$

$$\bigcap_i g_i(\mathbf{p}) = \max g_i(\mathbf{p})$$



- In many cases, smooth blending is desired
 - Pasko and Savchenko [1994]

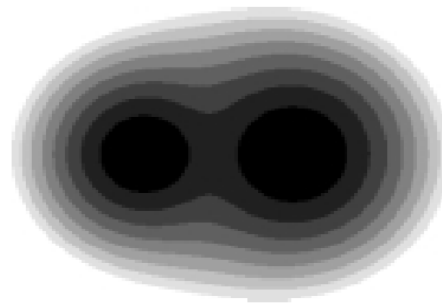
$$g \cup f = \frac{1}{1+\alpha} \left(g + f - \sqrt{g^2 + f^2 - 2\alpha gf} \right)$$

$$g \cap f = \frac{1}{1+\alpha} \left(g + f + \sqrt{g^2 + f^2 - 2\alpha gf} \right)$$

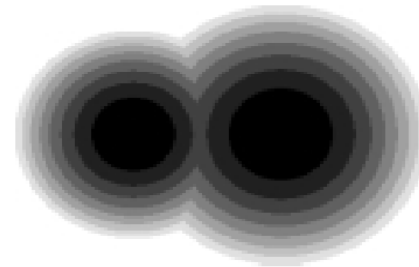
Implicit Curves and Surfaces

Smooth set operations

- Examples



$\alpha = 0$



$\alpha = 1$

- For $\alpha = 1$, this is equivalent to min and max

$$\lim_{\alpha \rightarrow 1} g \cup f = \frac{1}{2} \left(g + f - \sqrt{(g - f)^2} \right) = \frac{g + f}{2} - \frac{|g - f|}{2} = \min(g, f)$$

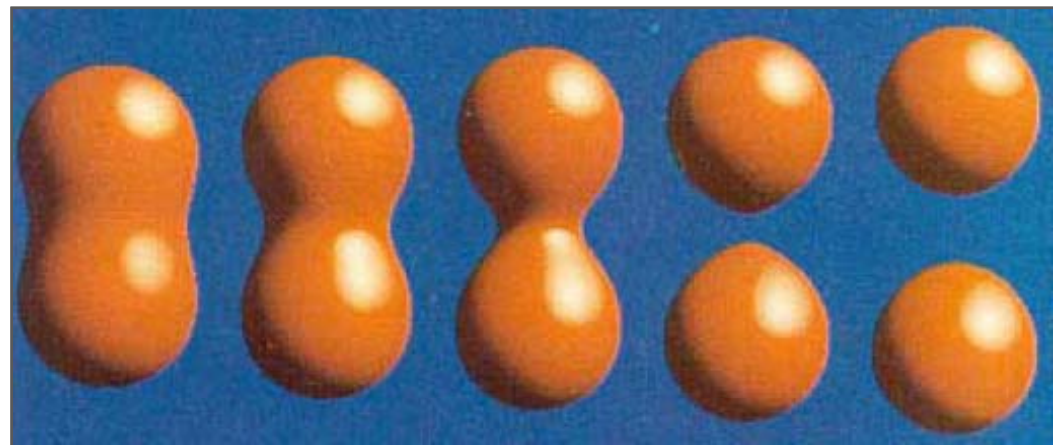
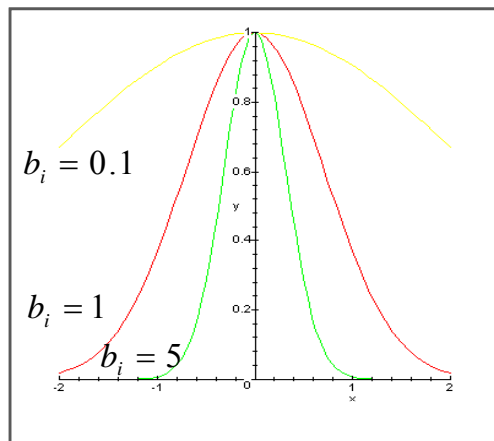
$$\lim_{\alpha \rightarrow 1} g \cap f = \frac{1}{2} \left(g + f + \sqrt{(g - f)^2} \right) = \frac{g + f}{2} + \frac{|g - f|}{2} = \max(g, f)$$



Implicit Curves and Surfaces

Blobs

- Suggested by Blinn [1982]
 - Defined implicitly by a potential function around a point \mathbf{p}_i :
$$g_i(\mathbf{p}) = a_i e^{-b_i \|\mathbf{p} - \mathbf{p}_i\|^2}$$
 - Set operations by simple addition/subtraction



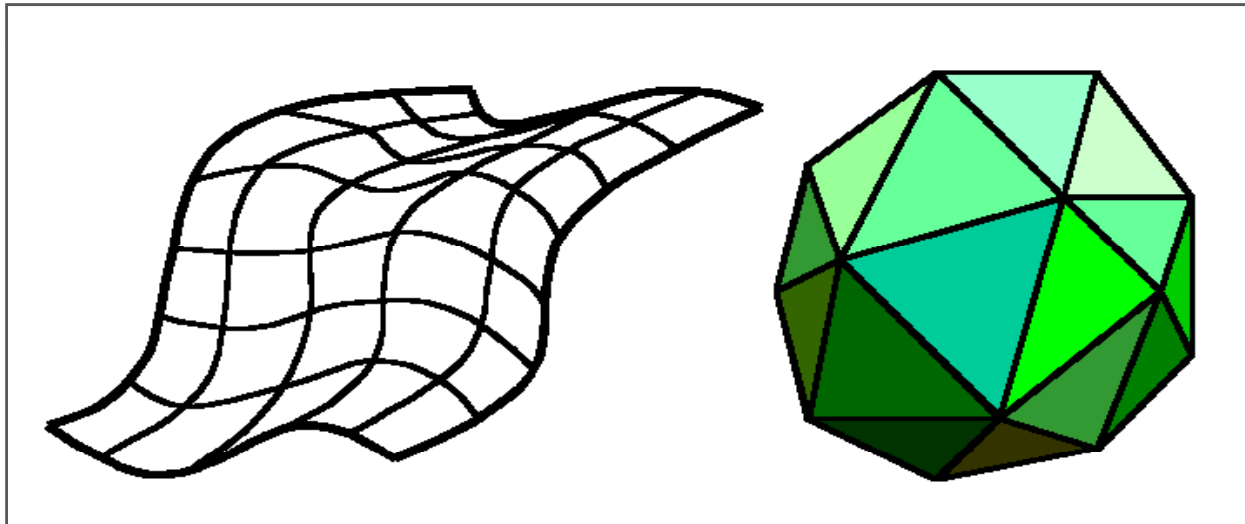
Implicit Curves and Surfaces

- 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

Polygonal Meshes

Polygonal Meshes

- Boundary representations of objects
 - Surfaces, polyhedrons, triangles, quadrilaterals



- How are these objects stored?

Definitions

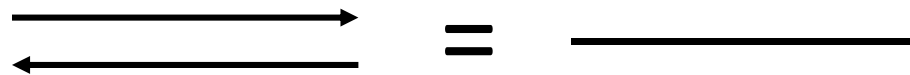
Geometric graph

- A Graph is a pair $G=(V,E)$
 - V is a nonempty set of n distinct vertices
 $\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_{n-1}$
 - E is a set of edges $(\mathbf{p}_i, \mathbf{p}_k)$
- If P is a (discrete) subset of \mathbb{R}^d with $d \geq 2$, then $G=(V,E)$ is a *geometric graph*
- The *degree* or *valence* of a vertex describes the number of edges incident to this vertex

Definitions

Edges

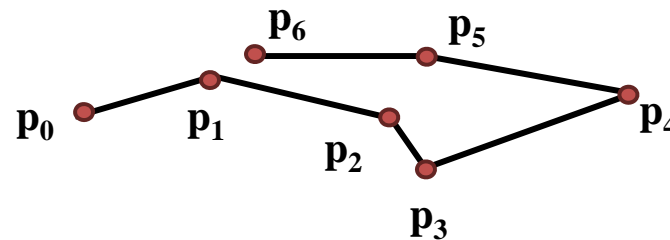
- Two edges are neighbors if they share a common vertex
- Edges are generally not oriented, and are noted as $(\mathbf{p}_i, \mathbf{p}_k)$
- Halfedges are edges with added orientation
- An edge is comprised of two halfedges



Definitions

Polygon

- A geometric graph $Q=(V,E)$ with $V=\{\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_{n-1}\} \subset \mathbb{R}^d$ with $d \geq 2$ and $E=\{(\mathbf{p}_0, \mathbf{p}_1), (\mathbf{p}_1, \mathbf{p}_2), \dots, (\mathbf{p}_{n-2}, \mathbf{p}_{n-1})\}$ is a *polygon*

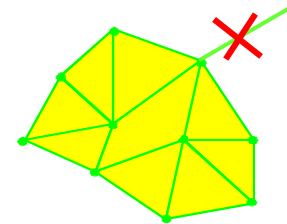


- A polygon is
 - Planar, if all vertices lie on a plane
 - Closed, if $\mathbf{p}_0 = \mathbf{p}_{n-1}$
 - Simple, if the polygon does not self-intersect

Definitions

Polygonal mesh

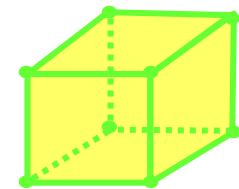
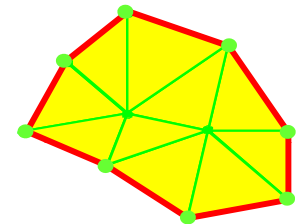
- A finite set M of closed, simple polygons Q_i is a polygonal mesh if:
 - The intersection of enclosed regions of any two polygons in M is empty
 - The intersection of two polygons in M is either empty, a vertex $v \in V$ or an edge $e \in E$
 - Every edge belongs to at least one polygon



Definitions

Polygonal mesh

- (Continued) The set of all edges that belong to only one polygon is termed the *boundary* of the polygonal mesh, and is either empty or forms closed loops
- If the set of edges that belong to only one polygon is empty, then the polygonal mesh is *closed*
- The set of all vertices and edges in a polygonal mesh form a graph



Definitions

Polyhedron

- A polygonal mesh is a polyhedron if
 - Each edge is part of two polygons (it is closed)
 - Every vertex $v \in V$ is part of finite, cyclic ordered set of polygons $\{Q_i\}$
 - The polygons incident to a vertex v can be ordered, such that Q_i and Q_j share an edge incident to v



- The union of all polygons forms a single connected component

Definitions

Manifold

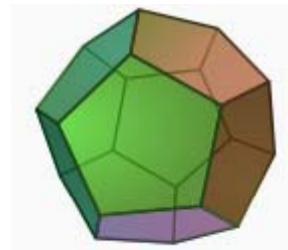
- A polygonal mesh is a 2-manifold if it is everywhere locally homeomorphic to a (half) Euclidean 2-ball (a disk)
 - A coffee cup is homeomorphic to a torus
- Examples for a non-manifold vertex and a non-manifold edge



Definitions

Polyhedron

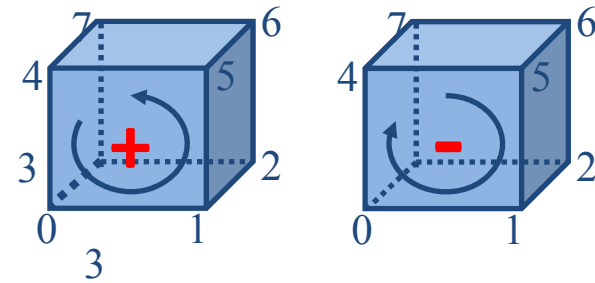
- The union of all polygonal areas is the *surface* of the polyhedron
- The polygonal areas of a polyhedron are also known as *faces*
- Every polyhedron partitions space into two areas; inside and outside the polyhedron



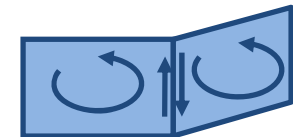
Definitions

Orientation

- Every face of a polygonal mesh is orientable
 - by defining “clockwise” (as opposed to “counterclockwise”). Two possible orientations
 - Defines the sign of the surface normal



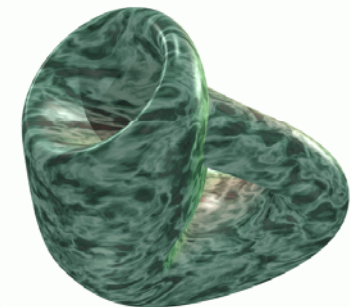
- Two neighboring facets are equally oriented, if the edge directions of the shared edge (induced by the face orientations) are opposing



Definitions

Orientability

- A polygonal mesh is orientable, if the incident faces to every edge can be equally oriented
 - If the faces are equally oriented for every edge, the mesh is *oriented*
- Notes
 - Every non-orientable closed mesh embedded in \mathbb{R}^3 intersects itself
 - The surface of a polyhedron is always orientable

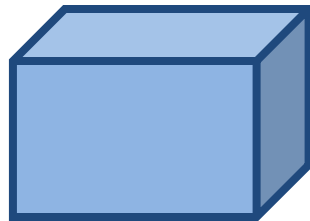


Euler-Poincaré Formula

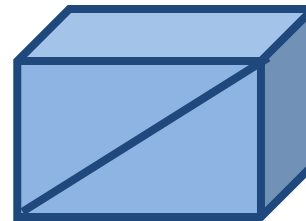
- Relation between #vertices, #edges and #faces of a polygonal mesh

- Example:

$v = \text{\#vertices}$
 $e = \text{\#edges}$
 $f = \text{\#faces}$



$v = 8$
 $e = 12$
 $f = 6$



$v = 8$
 $e = 12+1$
 $f = 6+1$

Euler-Poincaré Formula

- Theorem (Euler): The sum

$$\chi(M) = v - e + f$$

is constant for a given topology, no matter which mesh we choose

- If M has one boundary loop:

$$\chi(M) = v - e + f = 1$$

- If M is homeomorphic to a sphere:

$$\chi(M) = v - e + f = 2$$

Euler-Poincaré Formula

Usage

- Let's count the edges and faces in a closed triangle mesh:
 - Ratio of edges to faces: $e = 3/2 f$
 - each edge belongs to exactly 2 triangles
 - each triangle has exactly 3 edges
 - Ratio of vertices to faces: $f \sim 2v$
 - $2 = v - e + f = v - 3/2 f + f$
 - $2 - v = -f / 2$
 - Ratio of edges to vertices: $e \sim 3v$
 - Average degree of a vertex: 6
 - 2 vertices incident on each edge

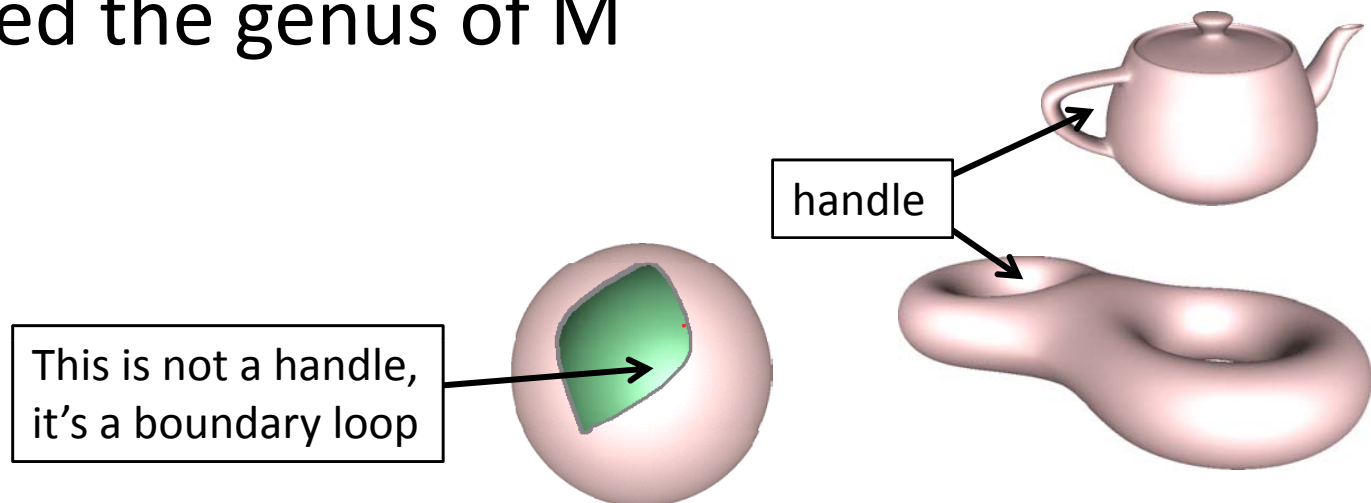
Euler-Poincaré Formula

Genus

- Theorem: if a polyhedron M is homeomorphic to a sphere with g handles (“holes”) then

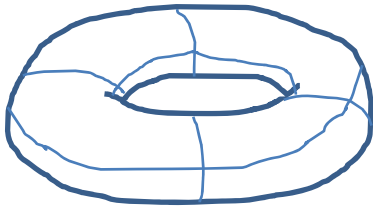
$$\chi(M) = v - e + f = 2(1 - g)$$

- g is called the genus of M



Euler-Poincaré Formula

Example: simple torus



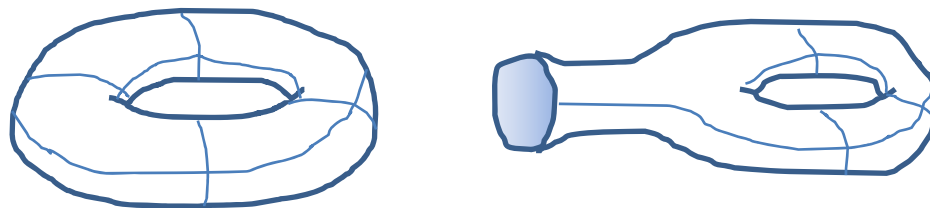
$$v - e + f = 2(1 - g)$$
$$8 - 16 + 8 = 2(1 - 1)$$

Euler-Poincaré Formula

Generalization

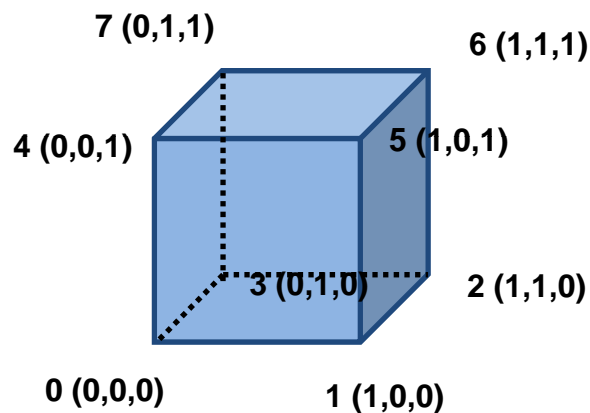
- Theorem: Let
 - v – # vertices
 - e – # edges
 - f – # faces
 - c – # connected components
 - h – # boundary loops
 - g – # handles (the genus)then:

$$v - e + f - h = 2(c - g)$$



Data structures for meshes

Indexed Face Set



Vertex list (Coordinate3)

0	0.0	0.0	0.0
1	1.0	0.0	0.0
2	1.0	1.0	0.0
3	0.0	1.0	0.0
4	0.0	0.0	1.0
5	1.0	0.0	1.0
6	1.0	1.0	1.0
7	0.0	1.0	1.0

Face list (IndexedFaceSet)

0	0	1	2	3
1	0	1	5	4
2	1	2	6	5
3	2	3	7	6
4	3	0	4	7
5	4	5	6	7

Data structures for meshes

Space requirements

- Coordinates/attributes

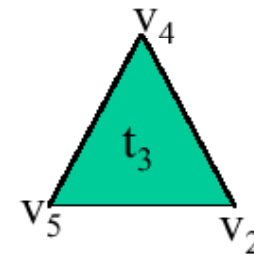
$3 \times 16 + k$ bits/vertex

vertex 1	x	y	z	c
vertex 2	x	y	z	c
vertex 3	x	y	z	c

- Connectivity

$3 \times \log_2(V)$ bits/triangle

Triangle 1	1	2	3
Triangle 2	3	2	4
Triangle 3	4	2	5
Triangle 4	7	5	6
Triangle 5	6	5	8
Triangle 6	8	5	1

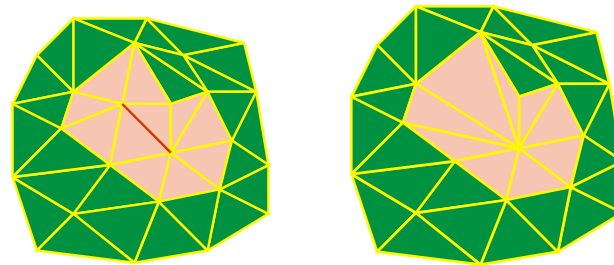


- When uncompressed, connectivity dominates
 - Reminder: $f = 2v...$ so after 256 vertices

Data structures for meshes

Indexed Face Set – Problems

- Information about neighbors is not explicit
 - Finding neighboring vertices/edges/faces etc. costs $O(v)$ time!
 - Local mesh modifications cost $O(v)$



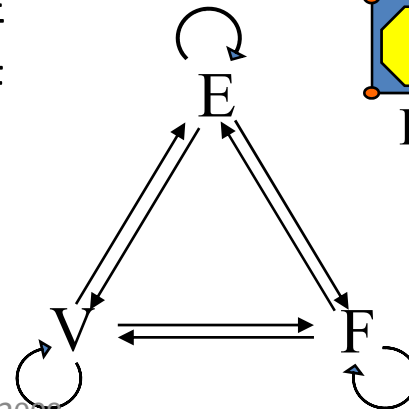
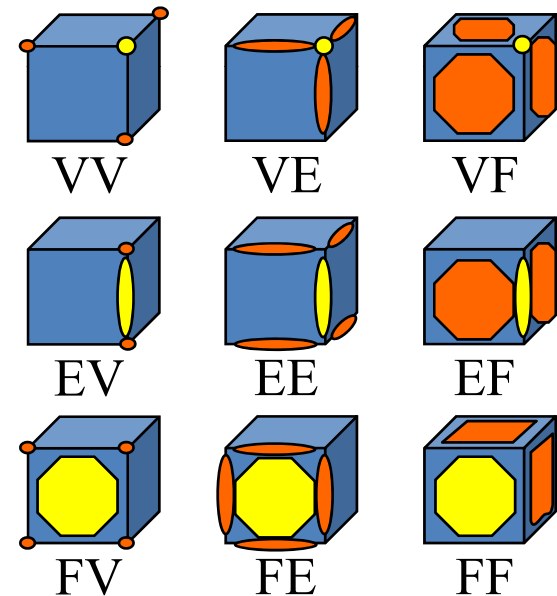
- Breadth-first search costs $O(k*v)$ where $k = \#$ found vertices

Data structures for meshes

Neighborhood relations [Weiler 1985]

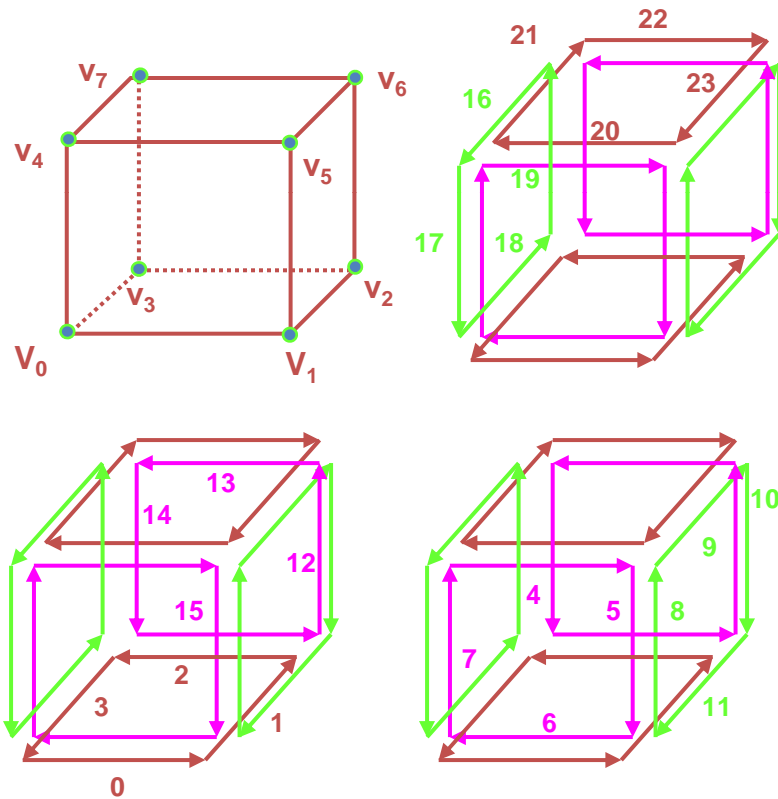
- All possible neighborhood relationships:

- | | | | |
|----|--------|----------|----|
| 1. | Vertex | – Vertex | VV |
| 2. | Vertex | – Edge | VE |
| 3. | Vertex | – Face | VF |
| 4. | Edge | – Vertex | EV |
| 5. | Edge | – Edge | EE |
| 6. | Edge | – Face | EF |
| 7. | Face | – Vertex | FV |
| 8. | Face | – Edge | FE |
| 9. | Face | – Face | FF |



Data structures for meshes

Half-edge data structure



Vertexlist

v	coord			he
0	0.0	0.0	0.0	0
1	1.0	0.0	0.0	1
2	1.0	1.0	0.0	2
3	0.0	1.0	0.0	3
4	0.0	0.0	1.0	4
5	1.0	0.0	1.0	9
6	1.0	1.0	1.0	13
7	0.0	1.0	1.0	16

Face

f	e
0	e0
1	e8
2	e4
3	e16
4	e12
5	e20

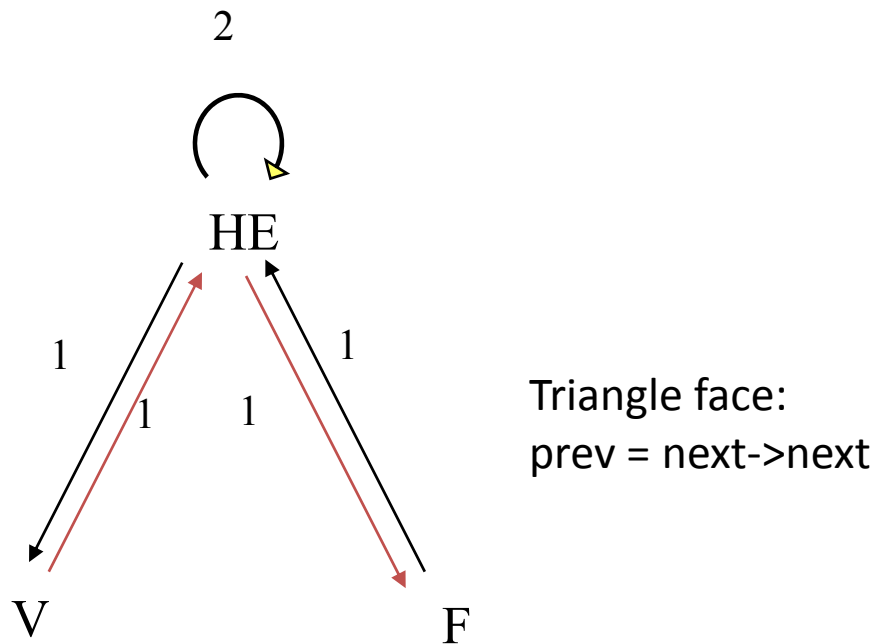
Half-Edgelist

he	vstart	next	prev	opp	he	vstart	next	prev	opp
0	0	1	3	6	12	2	13	15	10
1	1	2	0	11	13	6	14	12	22
2	2	3	1	15	14	7	15	13	19
3	3	0	2	18	15	3	12	14	2
4	4	5	7	20	16	7	17	19	21
5	5	6	4	8	17	4	18	16	7
6	1	7	5	0	18	0	19	17	3
7	0	4	6	17	19	3	16	18	14
8	1	9	11	5	20	5	21	23	4
9	5	10	8	23	21	4	22	20	16
10	6	11	9	12	22	7	23	21	13
11	2	8	10	1	23	6	20	22	9

Data structures for meshes

Half-edge data structure

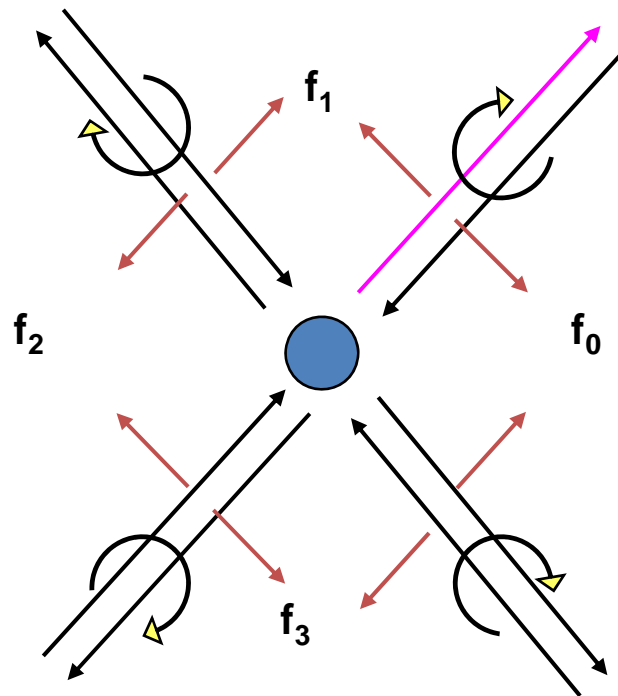
- Each atomic insertion into the data structure (i.e., vertex, edge or face insertion) requires constant space and time



Data structures for meshes

Half-edge data structure

- All basic queries take constant $O(1)$ time!
 - In particular, the query time is independent of the model size



Data structures for meshes

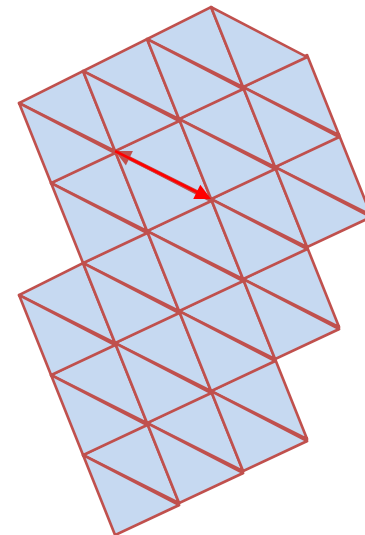
Half-edge data structure

- Example: efficient breadth-first search

```
//q: Queue (FIFO) of HalfEdges

HalfEdge he;
q.append(he);
if (he.opposite != null)
    q.append(he.opposite);

while (! q.isEmpty()) {
    he=q.first();
    // do work
    if (he.next.opposite != null)
        q.append(he.next.opposite);
    if (he.next.next.opposite != null)
        q.append(he.next.next.opposite)
}
```



Data structures for meshes

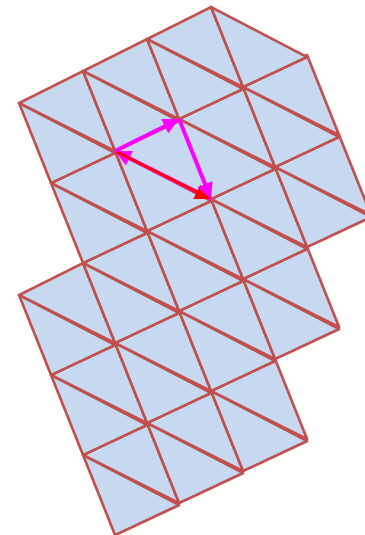
Half-edge data structure

- Example: efficient breadth-first search

```
//q: Queue (FIFO) of HalfEdges

HalfEdge he;
q.append(he);
if (he.opposite != null)
    q.append(he.opposite);

while (! q.isEmpty()) {
    he=q.first();
    // do work
    if (he.next.opposite != null)
        q.append(he.next.opposite);
    if (he.next.next.opposite != null)
        q.append(he.next.next.opposite)
}
```



Data structures for meshes

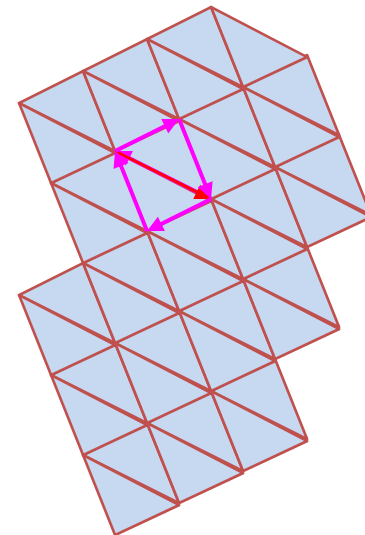
Half-edge data structure

- Example: efficient breadth-first search

```
//q: Queue (FIFO) of HalfEdges

HalfEdge he;
q.append(he);
if (he.opposite != null)
    q.append(he.opposite);

while (! q.isEmpty()) {
    he=q.first();
    // do work
    if (he.next.opposite != null)
        q.append(he.next.opposite);
    if (he.next.next.opposite != null)
        q.append(he.next.next.opposite)
}
```



Data structures for meshes

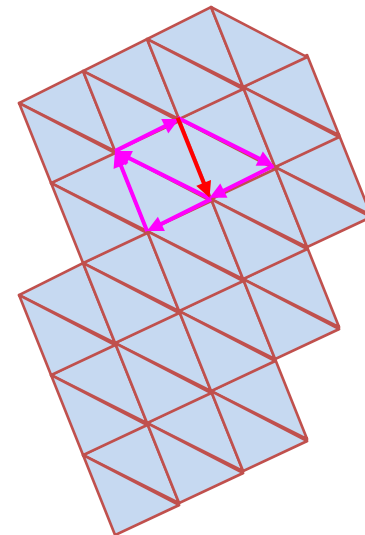
Half-edge data structure

- Example: efficient breadth-first search

```
//q: Queue (FIFO) of HalfEdges

HalfEdge he;
q.append(he);
if (he.opposite != null)
    q.append(he.opposite);

while (! q.isEmpty()) {
    he=q.first();
    // do work
    if (he.next.opposite != null)
        q.append(he.next.opposite);
    if (he.next.next.opposite != null)
        q.append(he.next.next.opposite)
}
```



Data structures for meshes

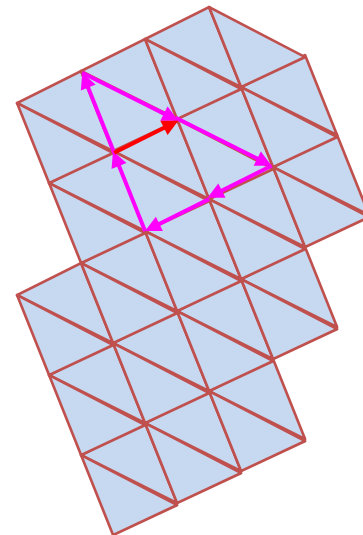
Half-edge data structure

- Example: efficient breadth-first search

```
//q: Queue (FIFO) of HalfEdges

HalfEdge he;
q.append(he);
if (he.opposite != null)
    q.append(he.opposite);

while (! q.isEmpty()) {
    he=q.first();
    // do work
    if (he.next.opposite != null)
        q.append(he.next.opposite);
    if (he.next.next.opposite != null)
        q.append(he.next.next.opposite);
}
```



Data structures for meshes

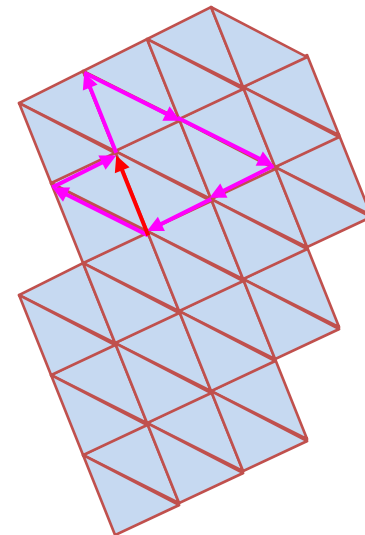
Half-edge data structure

- Example: efficient breadth-first search

```
//q: Queue (FIFO) of HalfEdges

HalfEdge he;
q.append(he);
if (he.opposite != null)
    q.append(he.opposite);

while (! q.isEmpty()) {
    he=q.first();
    // do work
    if (he.next.opposite != null)
        q.append(he.next.opposite);
    if (he.next.next.opposite != null)
        q.append(he.next.next.opposite)
}
```



Data structures for meshes

Criteria for design

- Maximal number of vertices (i.e., how large are the models?)
- Available memory size
- Required operations
 - Mesh updates (edge collapse, edge flip)
 - Neighborhood queries
- Distribution of operations (what are the most common/frequent ones?)
- How can we compare different data structures?