CS559: Computer Graphics

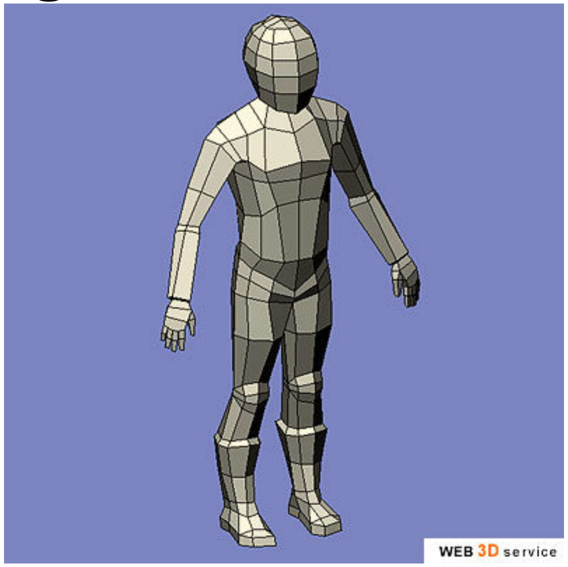
Lecture 24: Shape Modeling

Li Zhang

Spring 2010

Polygon Meshes

- A *mesh* is a set of polygons connected to form an object
- A mesh has several components, or geometric entities:
  - Faces
  - Edges
    - the boundary between faces
  - Vertices
    - the boundaries between edges,
    - or where three or more faces meet
  - Normals, Texture coordinates, colors, shading coefficients, etc



- What is the counterpart of a polygon mesh in curve modeling?

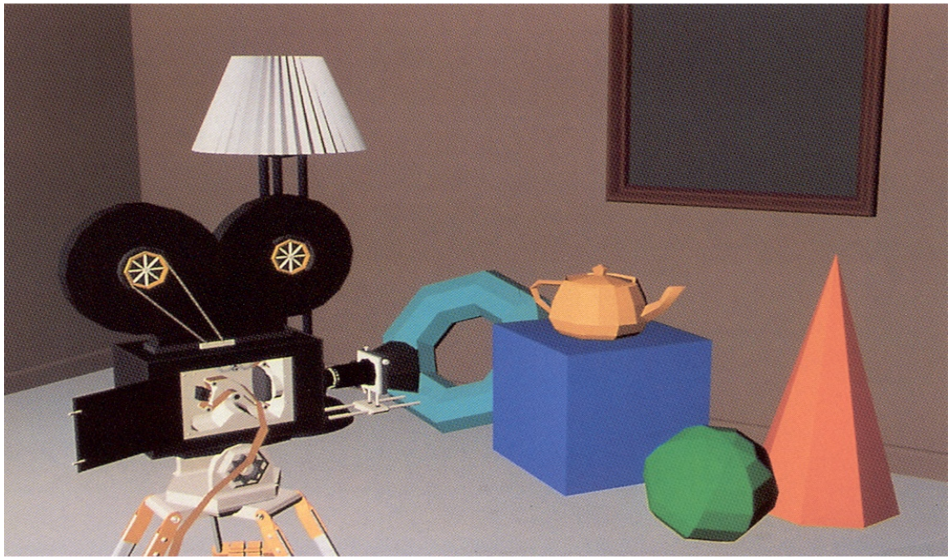
OpenGL and Vertex Indirection

```
struct Vertex {
   float coords[3];
}
struct Triangle {
   GLuint verts[3];
}
struct Mesh {
   struct Vertex vertices[m];
   struct Triangle triangles[n];
}

glEnableClientState(GL_VERTEX_ARRAY)
glVertexPointer(3, GL_FLOAT, sizeof(struct Vertex),
                  mesh.vertices);
glBegin(GL_TRIANGLES)
   for ( i = 0 ; i < n ; i++ )
   {
        glArrayElement(mesh.triangles[i].verts[0]);
        glArrayElement(mesh.triangles[i].verts[1]);
        glArrayElement(mesh.triangles[i].verts[2]);
   }
glEnd();
```

Normal Vectors in Mesh

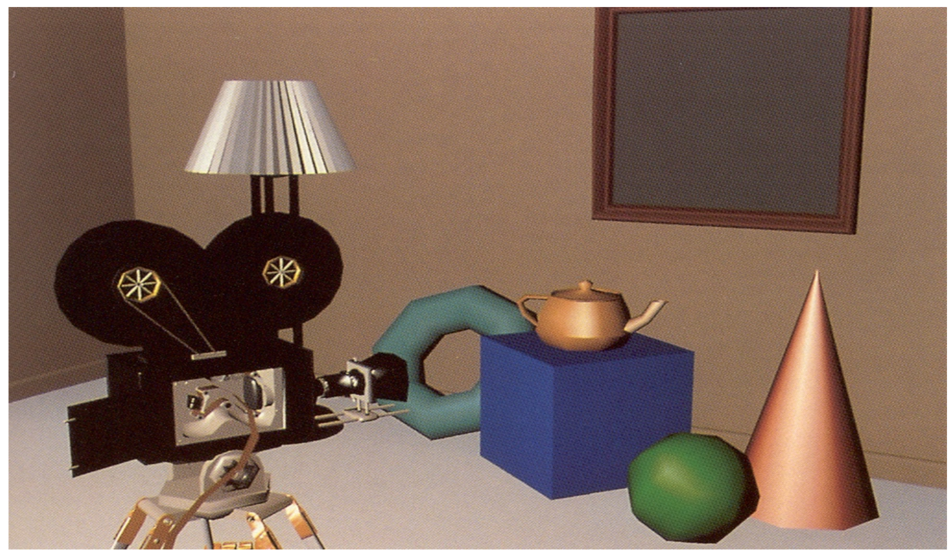
- Normal vectors give information about the true surface shape
- Per-Face normals:
  – One normal vector for each face, stored as part of face (Flat shading)



```
struct Vertex {
        float coords[3];
}
struct Triangle {
        GLuint verts[3];
        float normal[3];
}
struct Mesh {
        struct Vertex vertices[m];
        struct Triangle triangles[n];
}
```

Normal Vectors in Mesh

- Normal vectors give information about the true surface shape
- Per-Vertex normals:
  - A normal specified for every vertex (smooth shading)



```
struct Vertex {
        float coords[3];
        float normal[3];
}
struct Triangle {
        GLuint verts[3];
}
struct Mesh {
        struct Vertex vertices[m];
        struct Triangle triangles[n];
}
```

Other Data in Mesh

- Normal vectors give information about the true surface shape
- Per-Vertex normals:
  - A normal specified for every vertex (smooth shading)
- Per-Vertex Texture Coord

```
struct Vertex {
        float coords[3];
        float normal[3];
        float texCoords[2];
}
struct Triangle {
        GLuint verts[3];
}
struct Mesh {
        Vertex vertices[m];
        Triangle triangles[n];
}
```

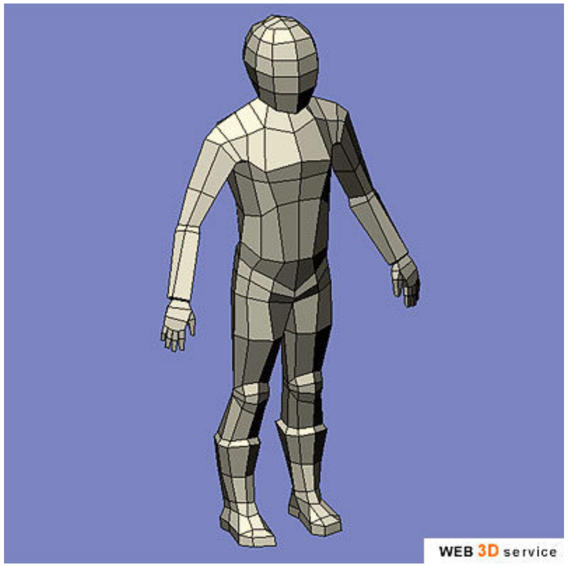
Other Data in Mesh

- Normal vectors give information about the true surface shape
- Per-Vertex normals:
  - A normal specified for every vertex (smooth shading)
- Per-Vertex Texture Coord, Shading Coefficients

```
struct Vertex {
        float coords[3];
        float normal[3];
        float texCoords[2], diffuse[3], shininess;
}
struct Triangle {
        GLuint verts[3];
}
struct Mesh {
        Vertex vertices[m];
        Triangle triangles[n];
}
```

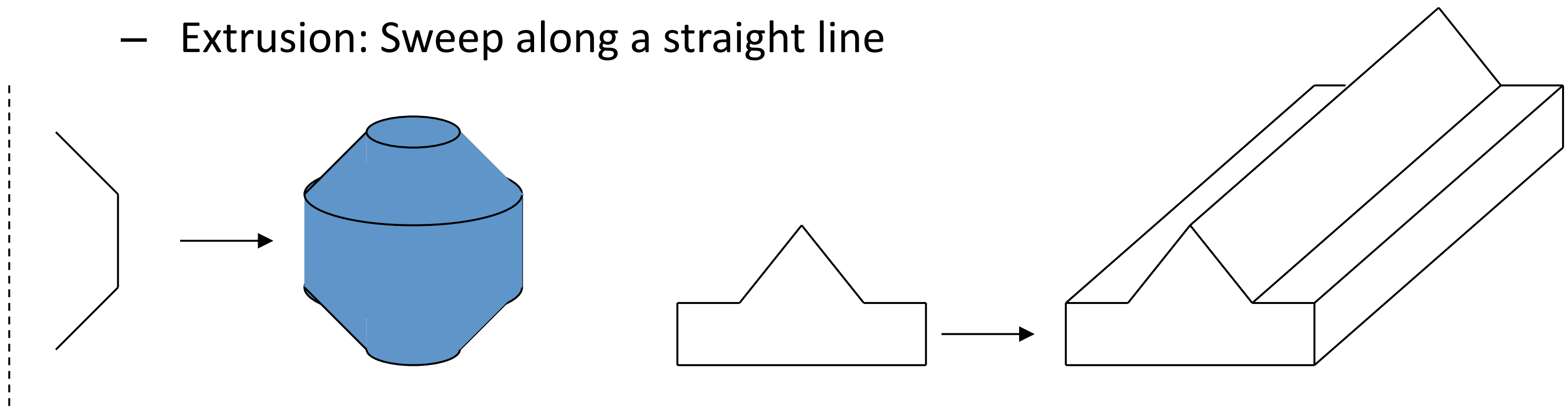
Issues with Polygons

- They are inherently an approximation
  - Things like silhouettes can never be perfect without very large numbers of polygons, and corresponding expense
  - Normal vectors are not specified everywhere
- Interaction is a problem
  - Dragging points around is time consuming
  - Maintaining things like smoothness is difficult
- Low level representation
  - Eg: Hard to increase, or decrease, the resolution
  - Hard to extract information like curvature



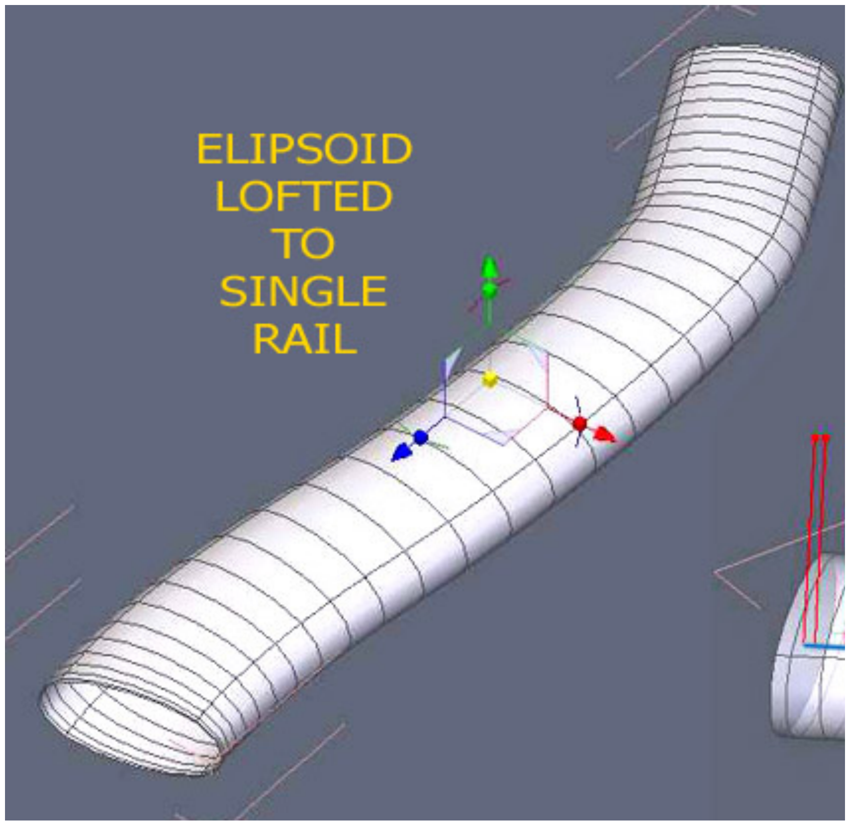
In Project 3, we use Sweep Objects

- Define a polygon by its edges
- Sweep it along a path
- The path taken by the edges form a surface - the sweep surface
- Special cases
  - Surface of revolution: Rotate edges about an axis
  - Extrusion: Sweep along a straight line



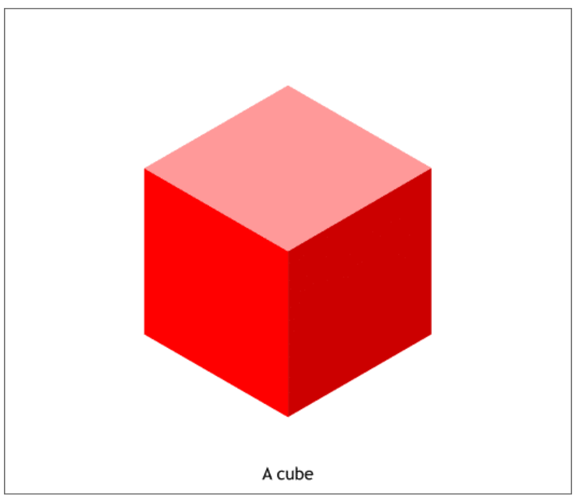
General Sweeps

- The path maybe any curve

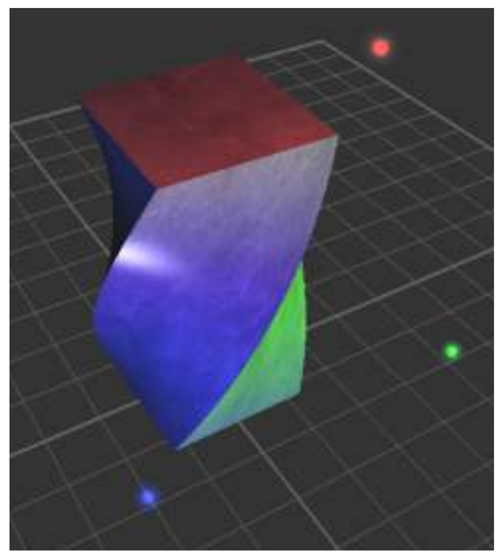


General Sweeps

- The path maybe any curve
- The polygon that is swept may be transformed as it is moved along the path
  - Scale, rotate with respect to path orientation, …



Cube

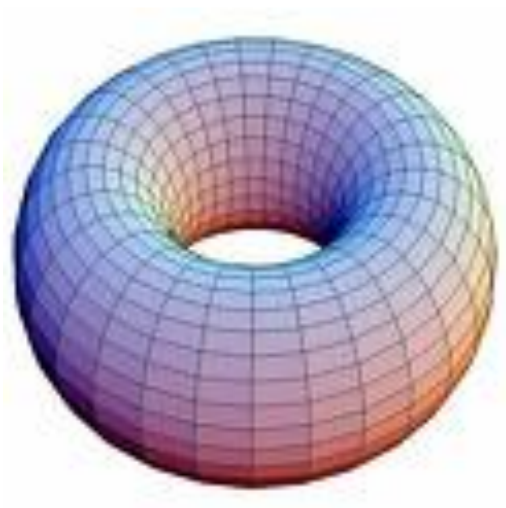


Twisted Cube

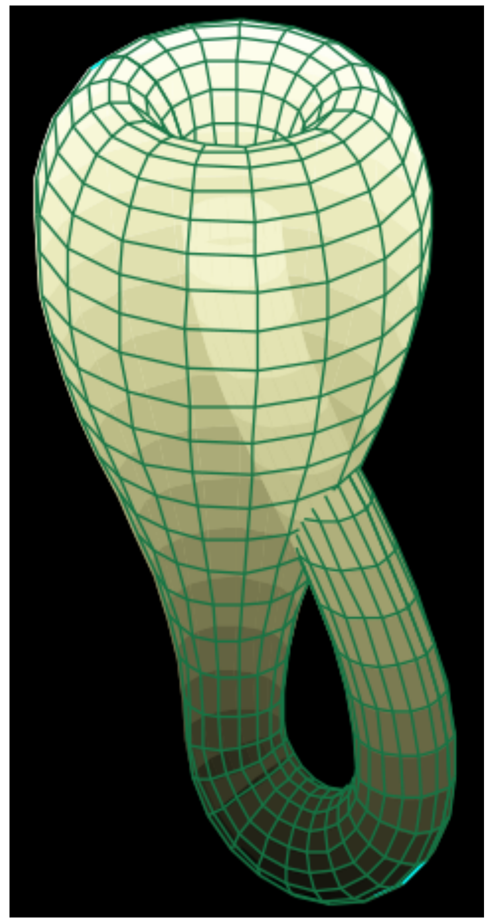
General Sweeps

- The path maybe any curve
- The polygon that is swept may be transformed as it is moved along the path
  - Scale, rotate with respect to path orientation, …
- One common way to specify is:
  - Give a poly-line (sequence of line segments) as the path
  - Give a poly-line as the shape to sweep
  - Give a transformation to apply at the vertex of each path segment
- Texture Coord?
- Difficult to avoid self-intersection

Klein Bottle

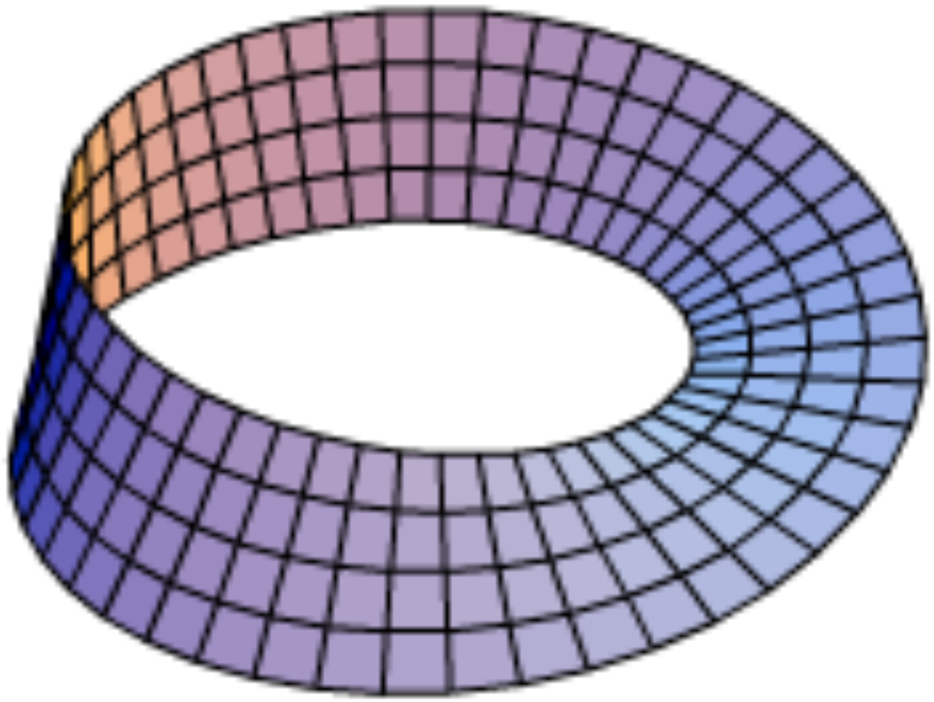


Torus



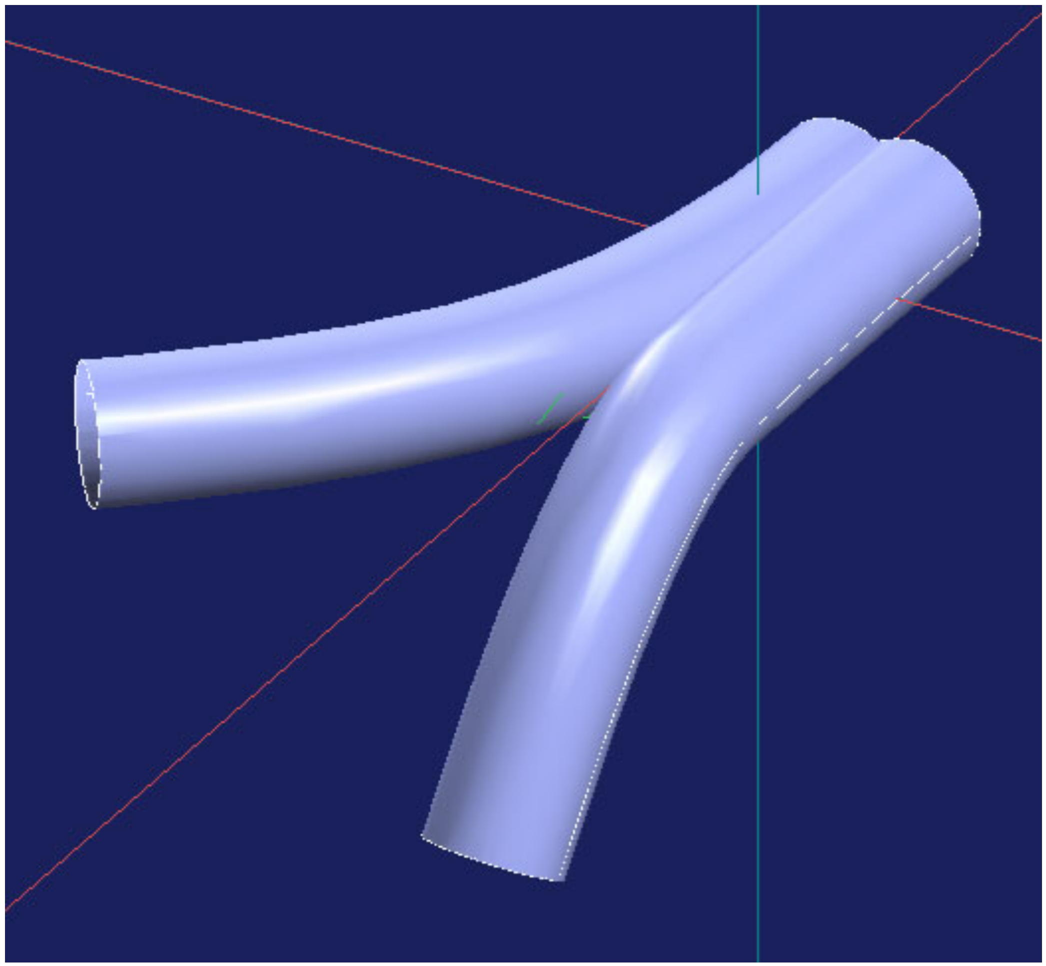
Klein Bottle

Mobious Strip



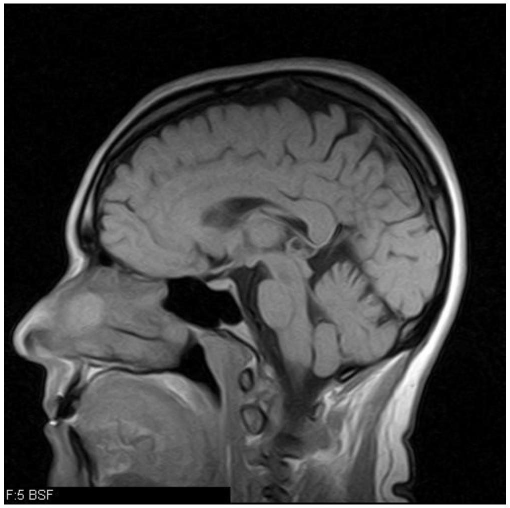
Non-orientable surfaces

Change Topology when Sweeping



Spatial Enumeration

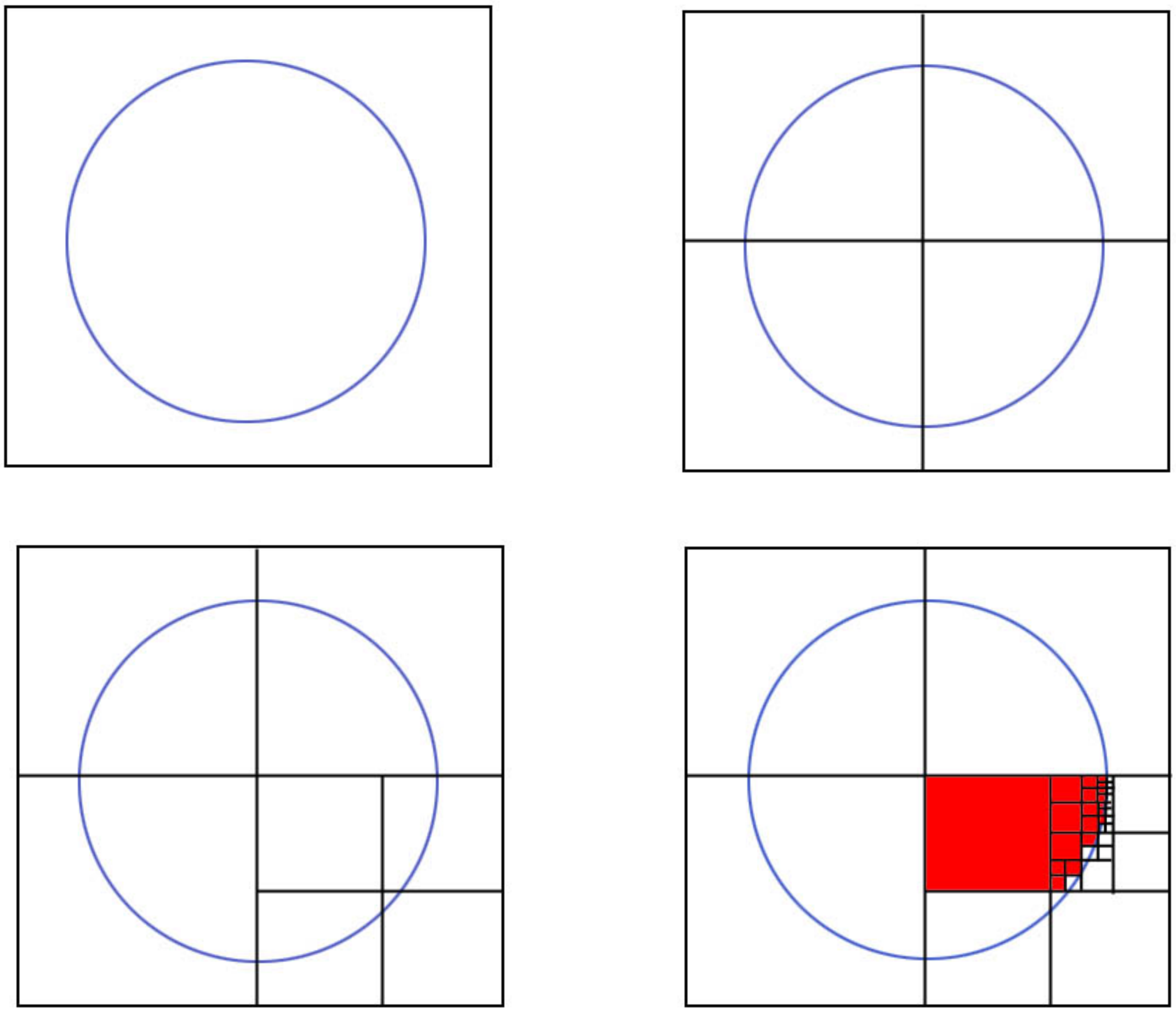
- Basic idea: Describe something by the space it occupies
  - For example, break the volume of interest into lots of tiny cubes
    - Data is associated with each voxel (volume element), binary or grayscale.
    - Works well for things like medical data (MRI or CAT scans, enumerates the volume)



Spatial Enumeration

- Basic idea: Describe something by the space it occupies
  - For example, break the volume of interest into lots of tiny cubes
    - Data is associated with each voxel (volume element), binary or grayscale.
    - Works well for things like medical data (MRI or CAT scans, enumerates the volume)
- Problem to overcome:
  - For anything other than small volumes or low resolutions, the number of voxels explodes
  - Note that the number of voxels grows with the *cube* of linear dimension

Quadtree Idea

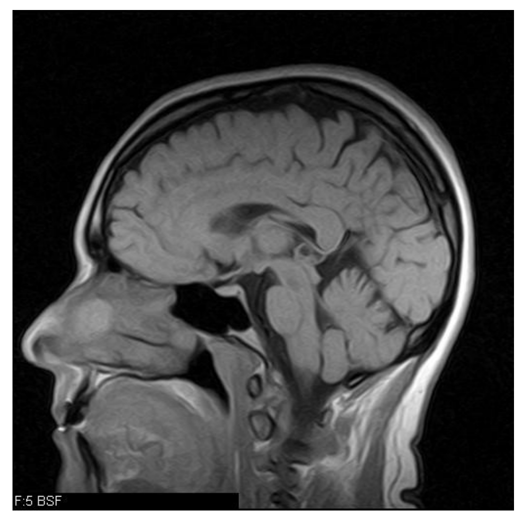


Octrees (and Quadtrees)

- Build a tree for adaptive voxel resolution
  - Large voxel for smooth regions
  - Small voxel for fine structures
- Quadtree is for 2D (four children for each node)
- Octree is for 3D (eight children for each node)

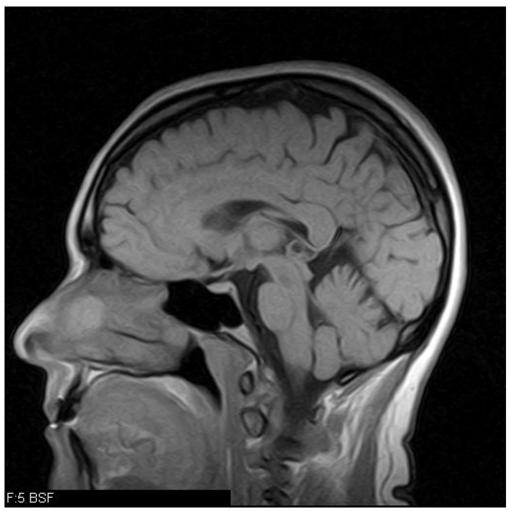
Rendering Octrees

- Volume rendering renders octrees and associated data directly
  - A special area of graphics, visualization, not covered in this class
- Can convert to polygons:
  - Find iso-surfaces within the volume and render those
  - Typically do some interpolation (smoothing) to get rid of the artifacts from the voxelization

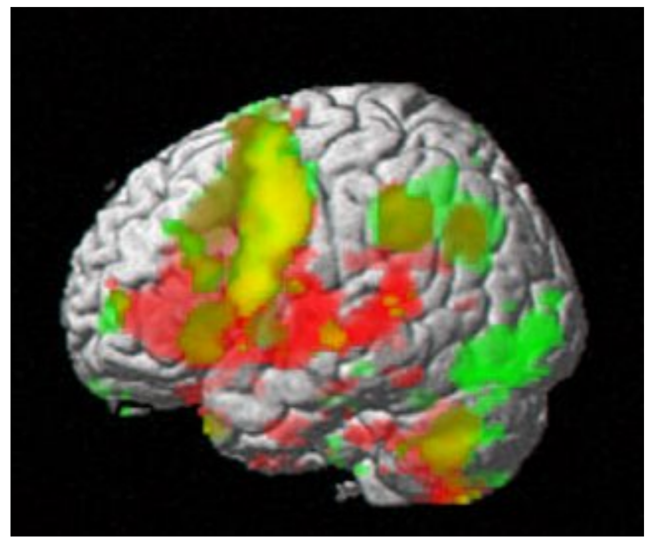


Rendering Octrees

- Typically render with colors that indicate something about the data



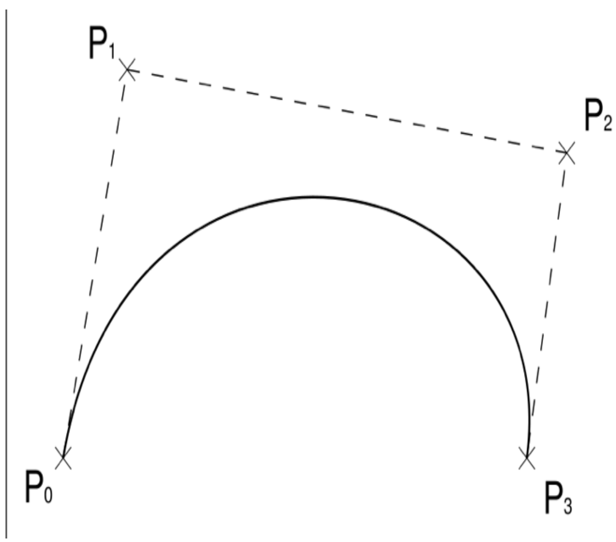
One MRI slice

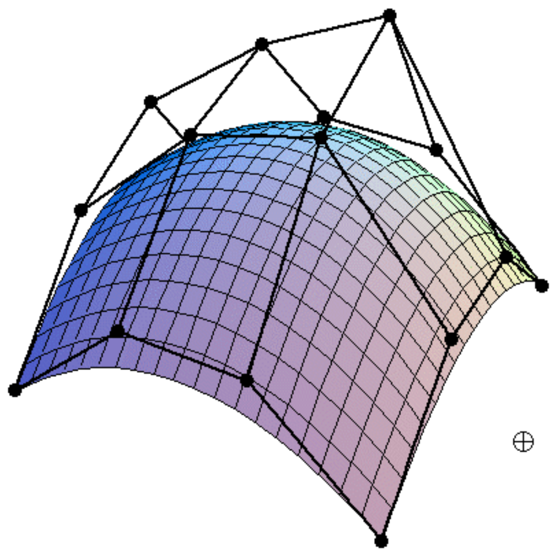


Surface rendering with color coded brain activity

Parametric surface

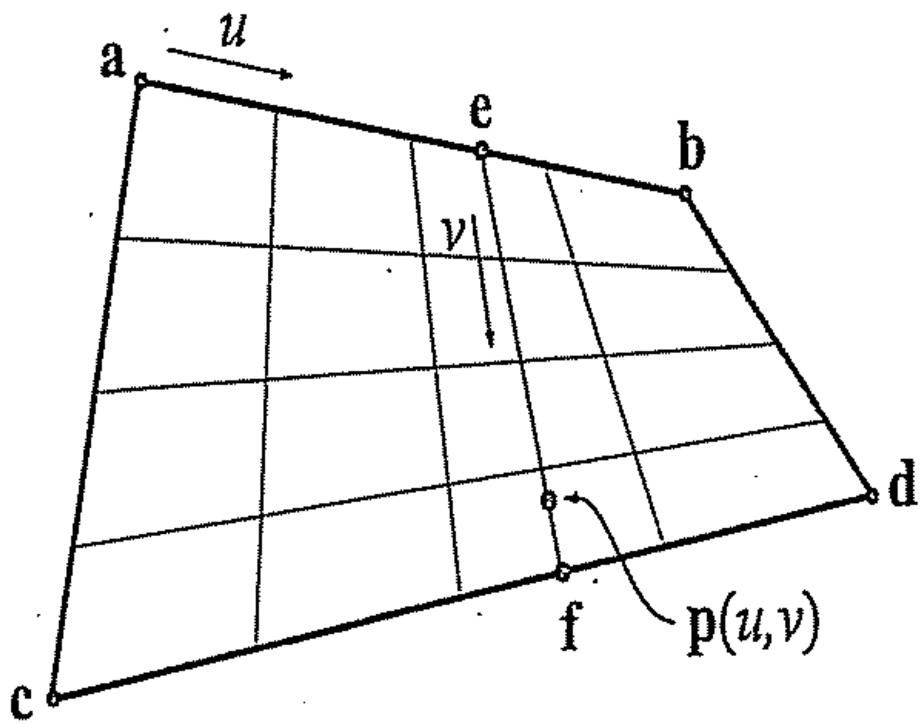
- Line Segments (1D) -> polygon meshes (2D)
- Cubic curves (1D) -> BiCubic Surfaces (2D)
  - Bezier curve -> Bezier surface





Bilinear Bezier Patch

- Define a surface that passes through a, b, c, d?



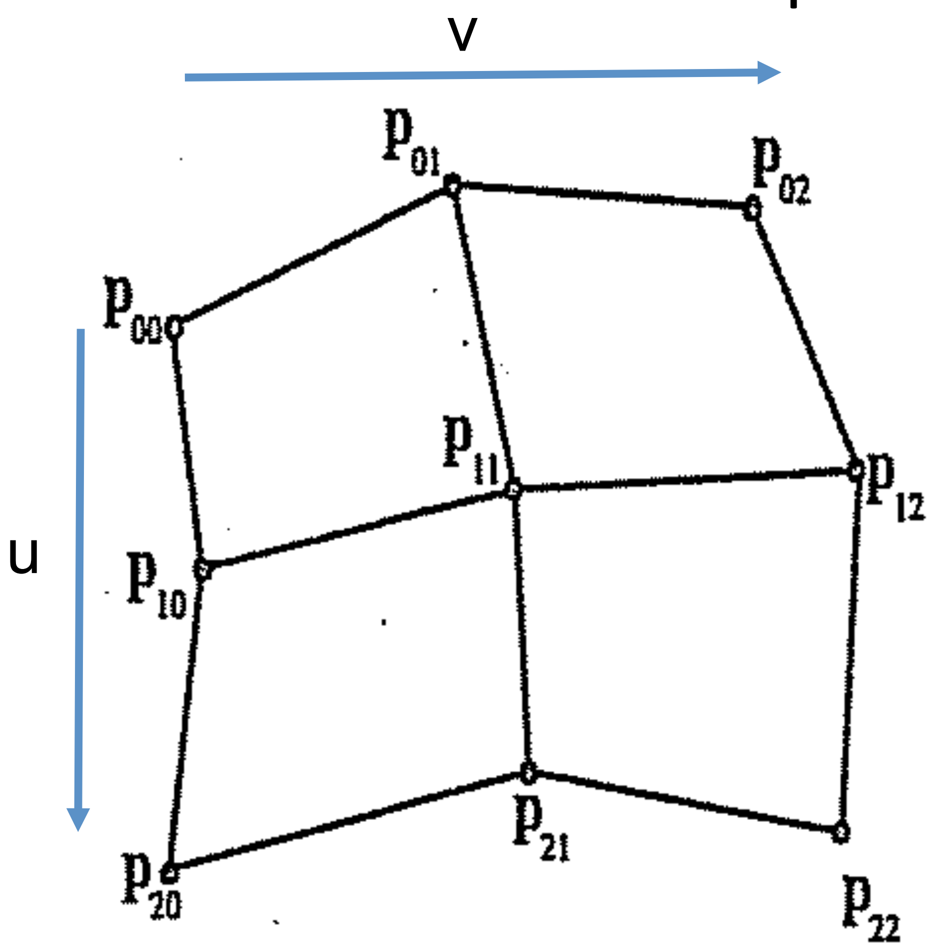
$$\mathbf{e} = (1-u)\mathbf{a} + u\mathbf{b},$$
$$\mathbf{f} = (1-u)\mathbf{c} + u\mathbf{d}.$$

Looks familiar?

$$\mathbf{p}(u,v) = (1-v)\mathbf{e} + v\mathbf{f}$$
$$= (1-u)(1-v)\mathbf{a} + u(1-v)\mathbf{b} + (1-u)v\mathbf{c} + uv\mathbf{d}.$$

Biquadratic Bezier Patch

- Define a surface that passes a 3x3 control lattice.



$$p(u,0) = (1-u)^2\, p_{00} + 2(1-u)u\, p_{10} + u^2\, p_{20}$$

$$p(u,1) = (1-u)^2\, p_{01} + 2(1-u)u\, p_{11} + u^2\, p_{21}$$

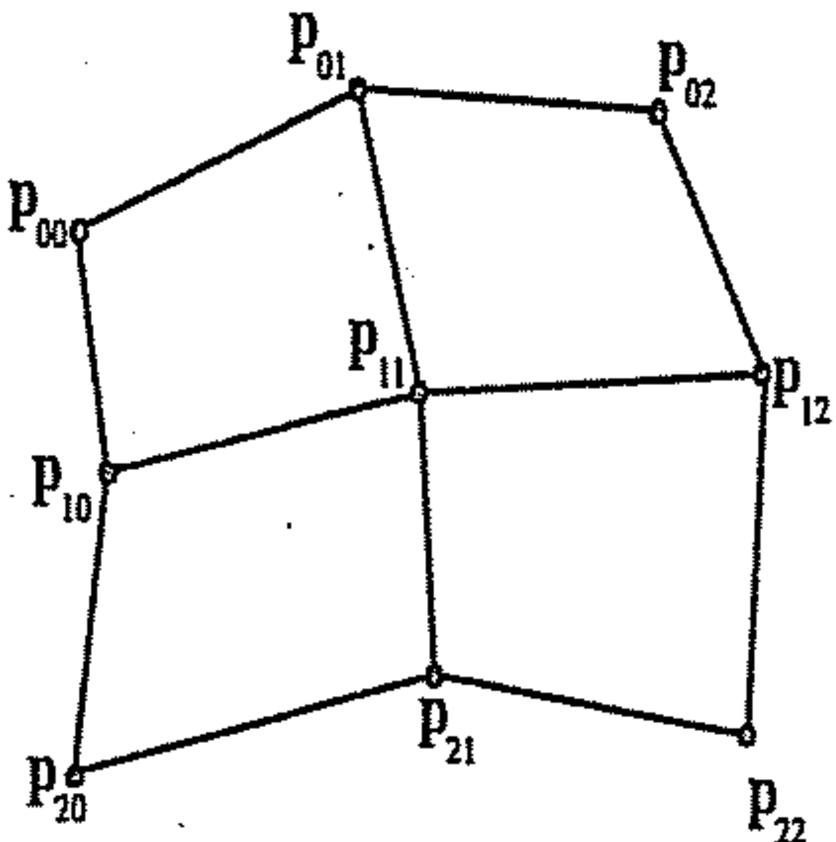
$$p(u,2) = (1-u)^2\, p_{02} + 2(1-u)u\, p_{12} + u^2\, p_{22}$$

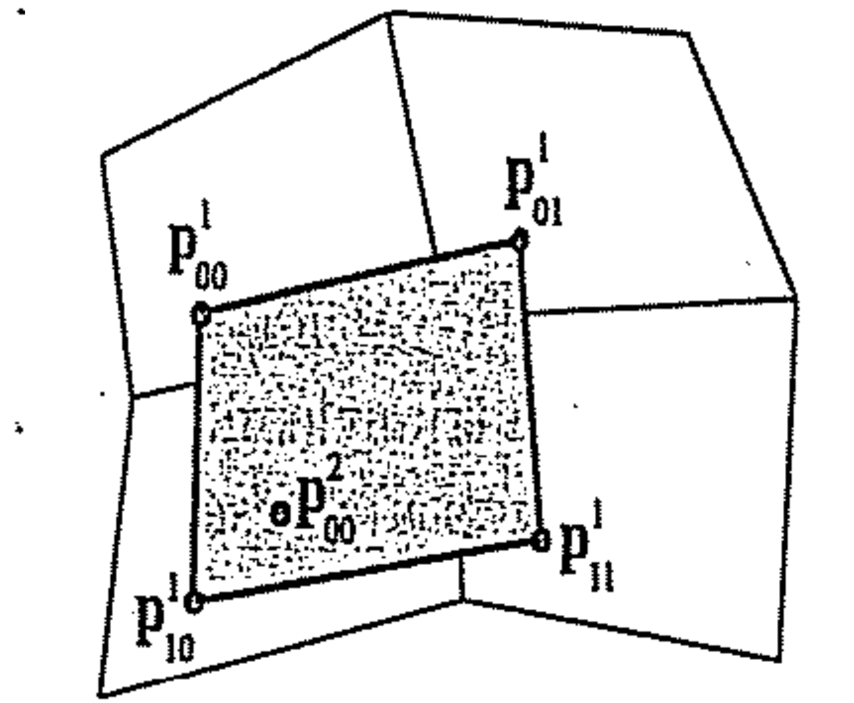
$$p(u,v) = (1-v)^2\, p(u,0) + 2(1-v)v\, p(u,1) + v^2\, p(u,2)$$

Bicubic Bezier Patch

- 4x4 control points?
- Demo: http://www.nbb.cornell.edu/neurobio/land/OldStudentProjects/cs490-96to97/anson/BezierPatchApplet/index.html
- Connecting Bezier Patches, demo on the same page.

De Casteljau algorithm in 2D





$p^1_{00}(u,v) = \text{Bilinear}(p_{00}, p_{10}, p_{01}, p_{11}; u, v)$

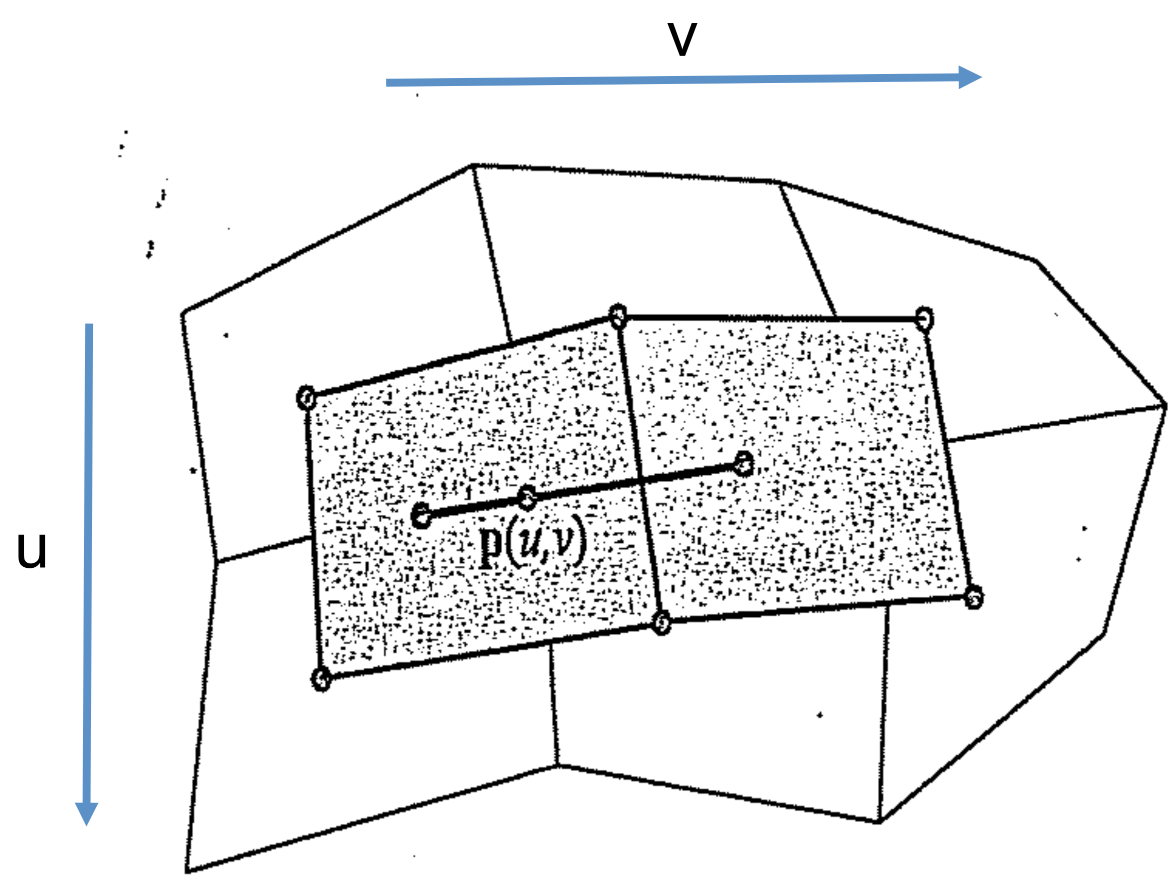
$p^1_{01}(u,v) = \text{Bilinear}(p_{01}, p_{11}, p_{02}, p_{12}; u, v)$

$p^1_{10}(u,v) = \text{Bilinear}(p_{10}, p_{20}, p_{11}, p_{21}; u, v)$

$p^1_{11}(u,v) = \text{Bilinear}(p_{11}, p_{21}, p_{12}, p_{22}; u, v)$

$p^1_{00}(u,v) = \text{Bilinear}(p_{00}, p_{10}, p_{01}, p_{11}; u, v)$

Different degree in different directions



General Formula for Bezier Patch

- If we have contronl points $p_{i,j}$ on a m by n lattice,

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

$$= \sum_{i=0}^{m}\sum_{j=0}^{n} \binom{m}{i}\binom{n}{j} u^i(1-u)^{m-i}v^j(1-v)^{n-j}\mathbf{p}_{i,j}$$

- Properties
  - Invariant to affine transform
  - Convex combination,
  - Used for intersection

$$\sum_{i=0}^{m}\sum_{j=0}^{n} B_i^m(u)B_j^n(v) = 1$$

General Formula for Bezier Patch

- If we have contronl points $p_{i,j}$ on a m by n lattice,

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

$$= \sum_{i=0}^{m}\sum_{j=0}^{n} \binom{m}{i} \binom{n}{j} u^i(1-u)^{m-i} v^j (1-v)^{n-j}\mathbf{p}_{i,j}$$

- Surface Normal

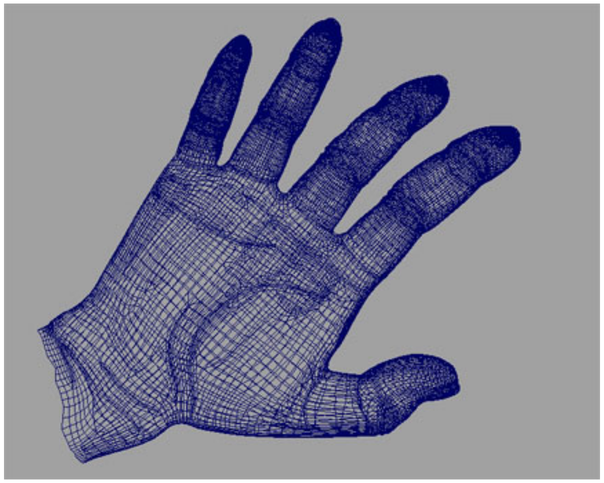
$$\frac{\partial \mathbf{p}(u,v)}{\partial u} = m \sum_{j=0}^{n}\sum_{i=0}^{m-1} B_i^{m-1}(u) B_j^n(v)[\mathbf{p}_{i+1,j} - \mathbf{p}_{i,j}]$$

$$\mathbf{n}(u,v) = \frac{\partial \mathbf{p}(u,v)}{\partial u} \times \frac{\partial \mathbf{p}(u,v)}{\partial v}.$$

$$\frac{\partial \mathbf{p}(u,v)}{\partial v} = n \sum_{i=0}^{m}\sum_{j=0}^{n-1} B_i^{m}(u) B_j^{n-1}(v)[\mathbf{p}_{i,j+1} - \mathbf{p}_{i,j}]$$

Issues with Bezier Patches

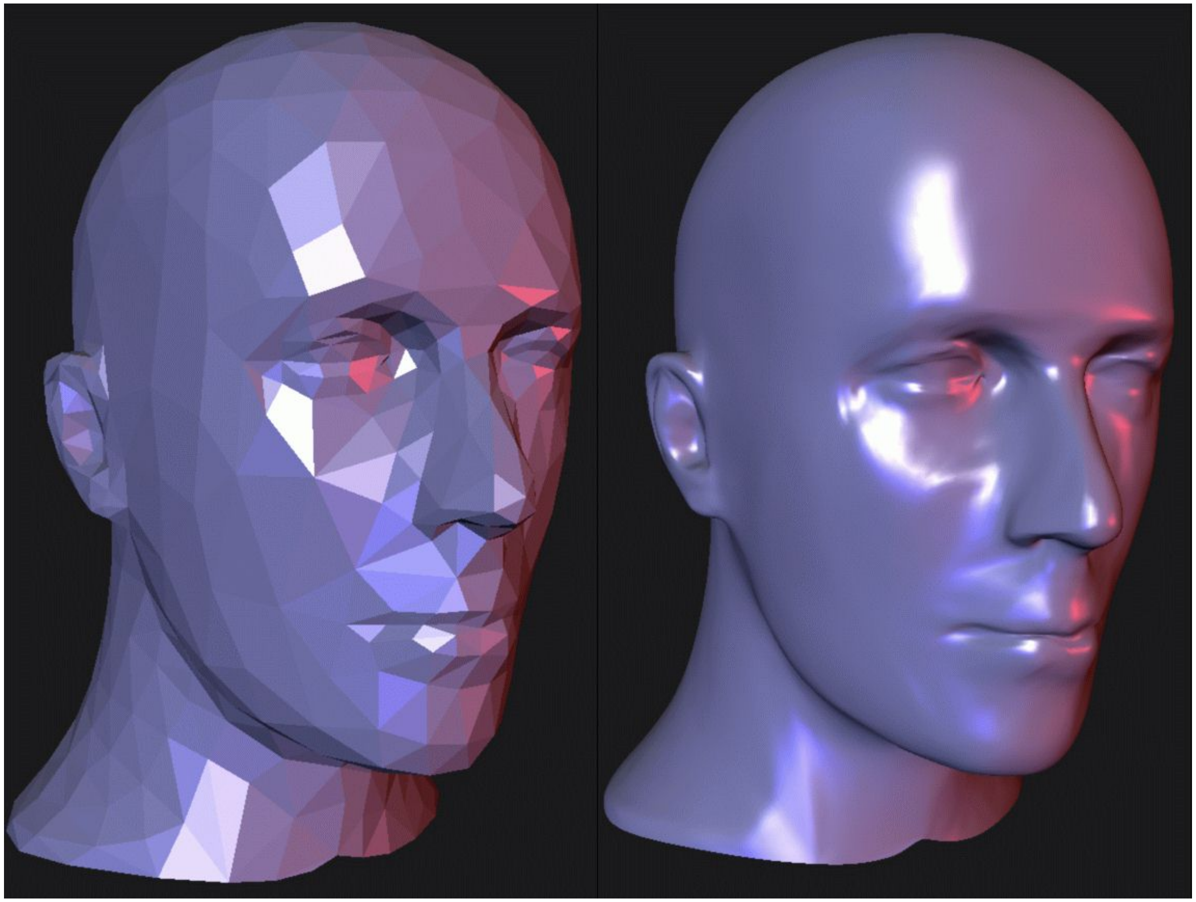
- With Bézier or B-spline patches, modeling complex surfaces amounts to trying to cover them with pieces of rectangular cloth.



- It's not easy, and often not possible if you don't make some of the patch edges degenerate (yielding triangular patches).
- Trying to animate that object can make continuity very difficult, and if you're not very careful, your model will show creases and artifacts near patch seams.
- Subdivision Surface is a promising solution.

Subdivision Surface

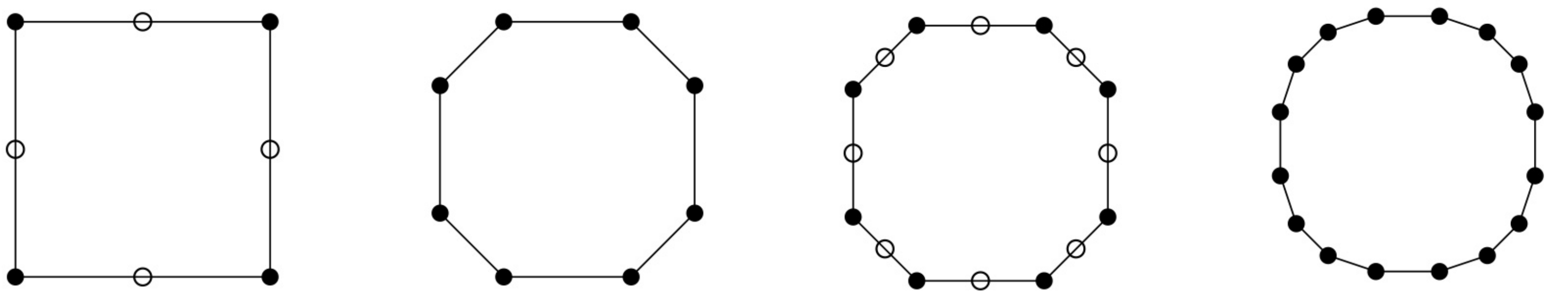
- From a coarse control mesh to smooth mesh with infinite resolution



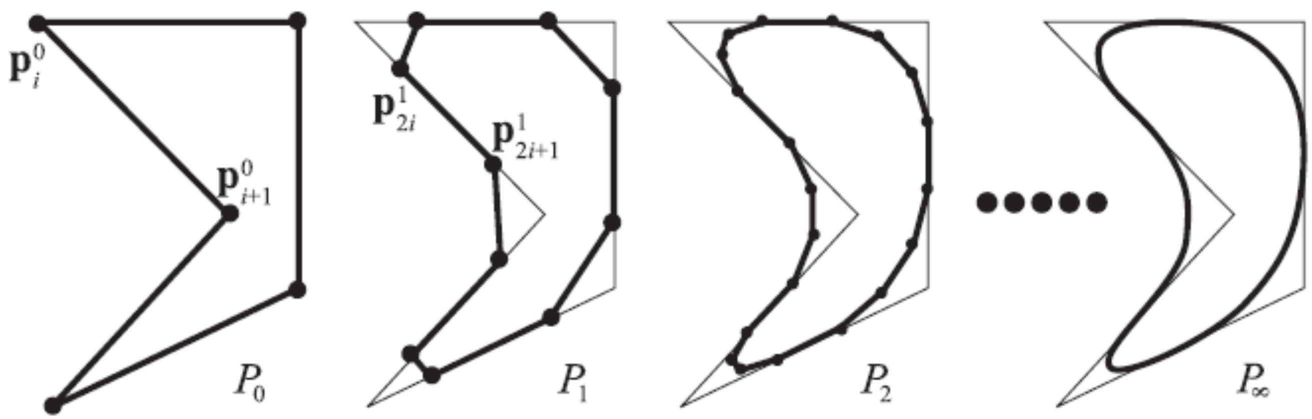
Example: Toy story 2

Subdivision Curve

We have seen this idea before

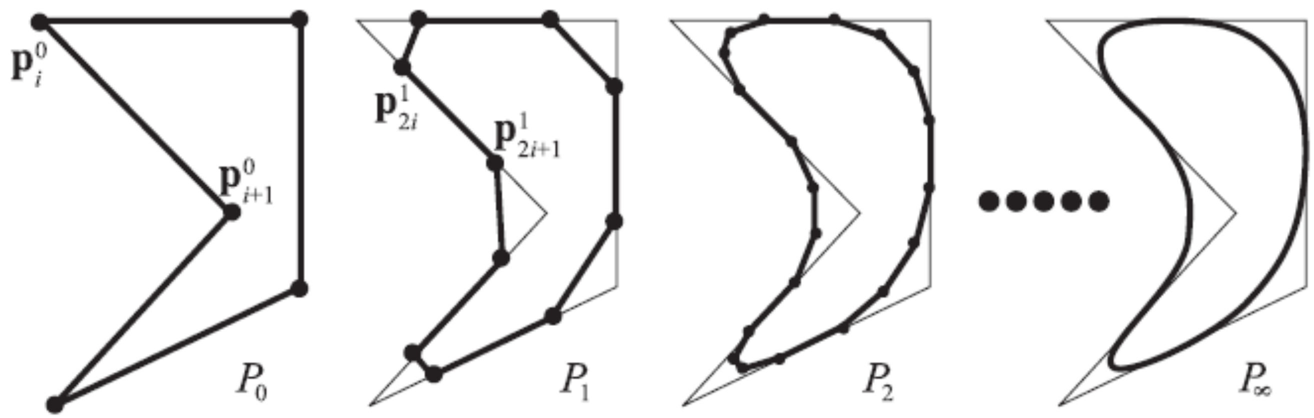


Shirley, Figure 15.15, The limiting curve is a quadratic Bezier Curve



RTR 3e, Figure 13.29, The limiting curve is a quadratic B-spline

Subdivision Curves: Approximating



Initial (Control) Curve: $P_0 = \{p_0^0, \ldots, p_{n-1}^0\},$

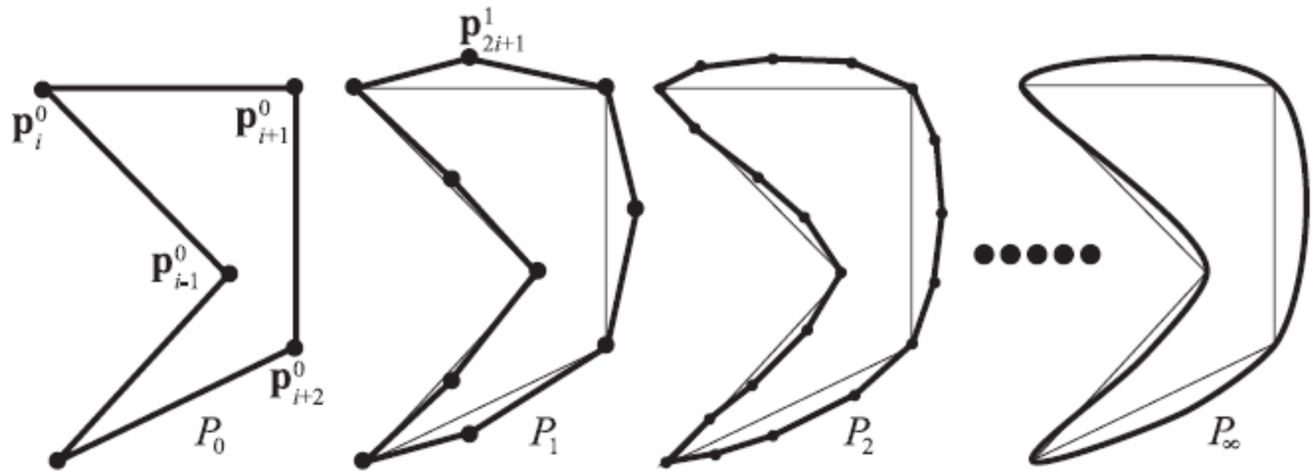
For each iteration k+1, add two vertices between: p_i^k and p_{i+1}^k

$$p_{2i}^{k+1} = \frac{3}{4}p_i^k + \frac{1}{4}p_{i+1}^k,$$

$$p_{2i+1}^{k+1} = \frac{1}{4}p_i^k + \frac{3}{4}p_{i+1}^k.$$

Approximating: Limit curve is very smooth (C2), but does not pass through control points

Subdivision Curves: Interpolating



Initial (Control) Curve: $P_0 = \{\mathrm{p}_0^0, \ldots, \mathrm{p}_{n-1}^0\}$,

For each iteration k+1, add two vertices between: p_i^k and p_{i+1}^k

$$\mathrm{p}_{2i}^{k+1} = \mathrm{p}_i^k,$$

$$\mathrm{p}_{2i+1}^{k+1} = (\frac{1}{2} + w)(\mathrm{p}_i^k + \mathrm{p}_{i+1}^k) - w(\mathrm{p}_{i-1}^k + \mathrm{p}_{i+2}^k).$$

Interpolating: for 0<w<1/8, limit curve is C1, and passes through control points

Subdivision Curves: Interpolating

- Handling Boundary Cases

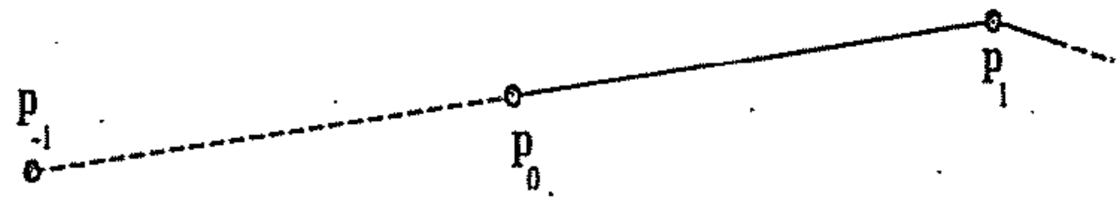
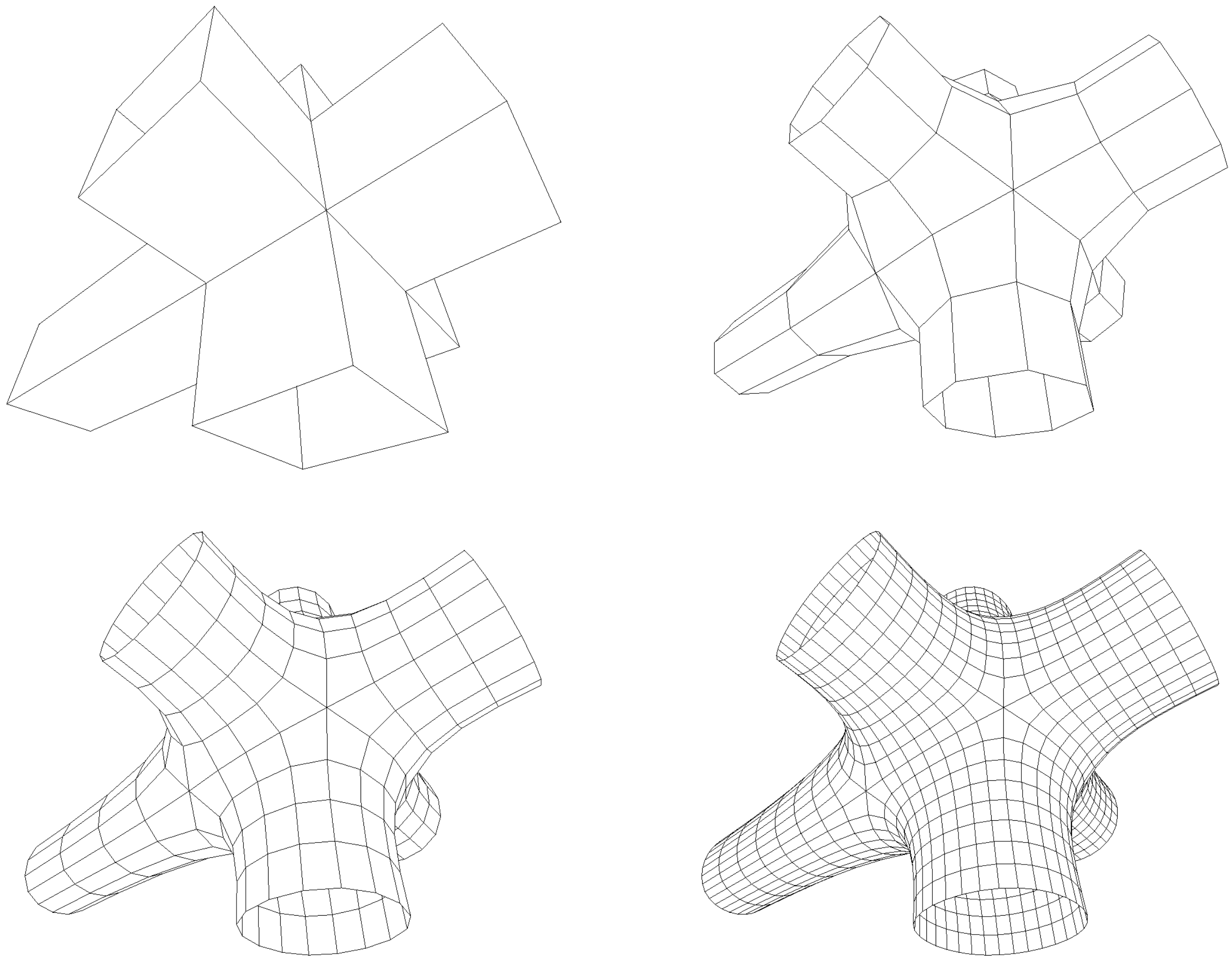


Figure 12.36. The creation of a reflection point, p_{-1}, for open polylines. The reflection point is computed as: $p_{-1} = p_0 - (p_1 - p_0) = 2p_0 - p_1$.

Subdivision Surfaces



Extend subdivision idea from curves to surfaces

RTR, 3e, figure 13.32

Basic Steps

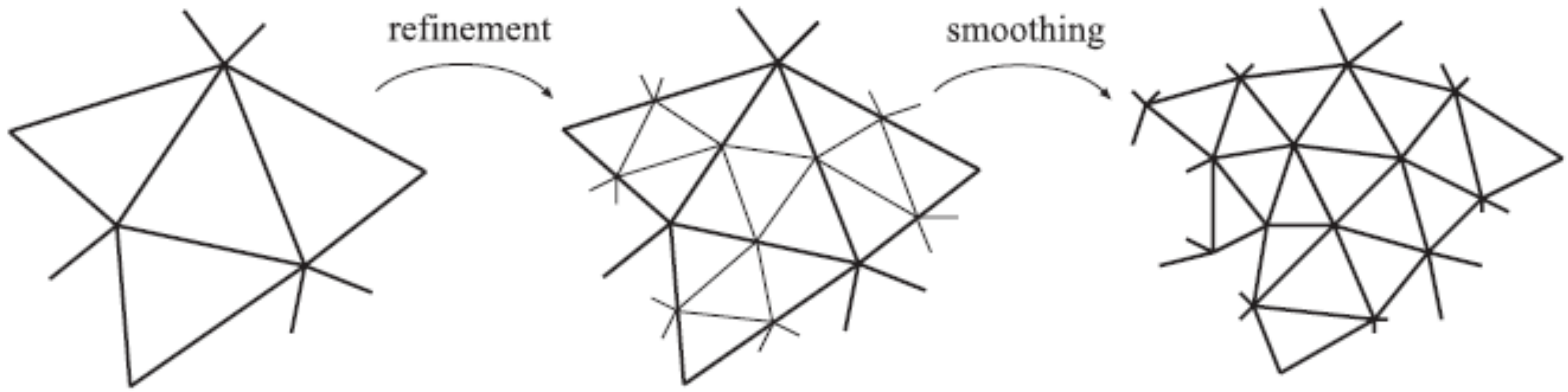
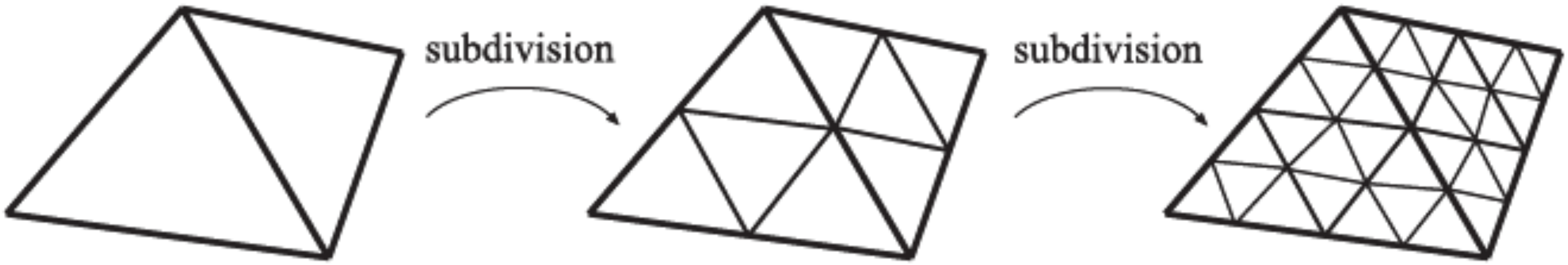


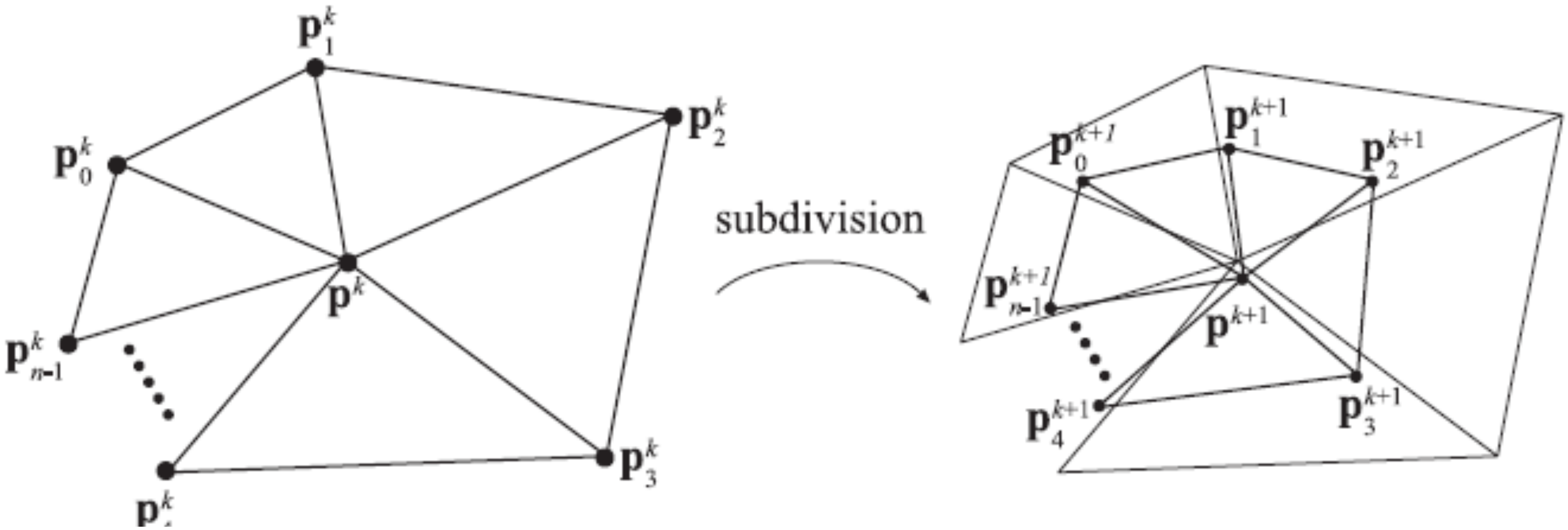
Figure 13.33: Subdivision as refinement and smoothing. The refinement phase creates new vertices and reconnects to create new triangles, and the smoothing phase computes new positions for the vertices.

Loop Subdivision



- Regular vertex: valence = 6
- Irregular vertex: valence != 6
- Irregular vertices can only be initial vertices.

Loop Subdivision

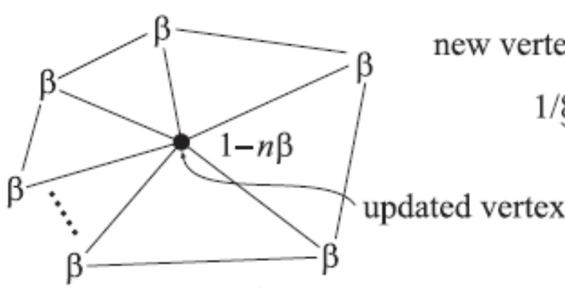


$$\mathrm{p}^{k+1} = (1 - n\beta)\mathrm{p}^k + \beta(\mathrm{p}_0^k + \cdots + \mathrm{p}_{n-1}^k),$$

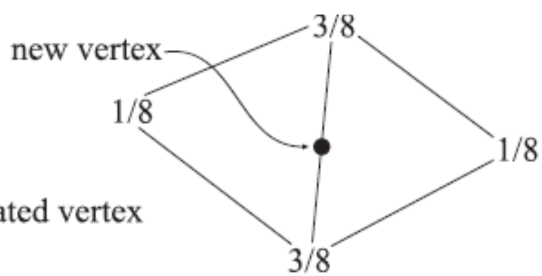
$$\mathrm{p}_i^{k+1} = \frac{3\mathrm{p}^k + 3\mathrm{p}_i^k + \mathrm{p}_{i-1}^k + \mathrm{p}_{i+1}^k}{8}, \quad i = 0 \ldots n-1.$$

$$\beta(n) = \frac{3}{n(n+2)}.$$

$$\beta(n) = \frac{1}{n}\left(\frac{5}{8} - \frac{(3 + 2\cos(2\pi/n))^2}{64}\right)$$



vertex mask



edge mask

Loop Subdivision

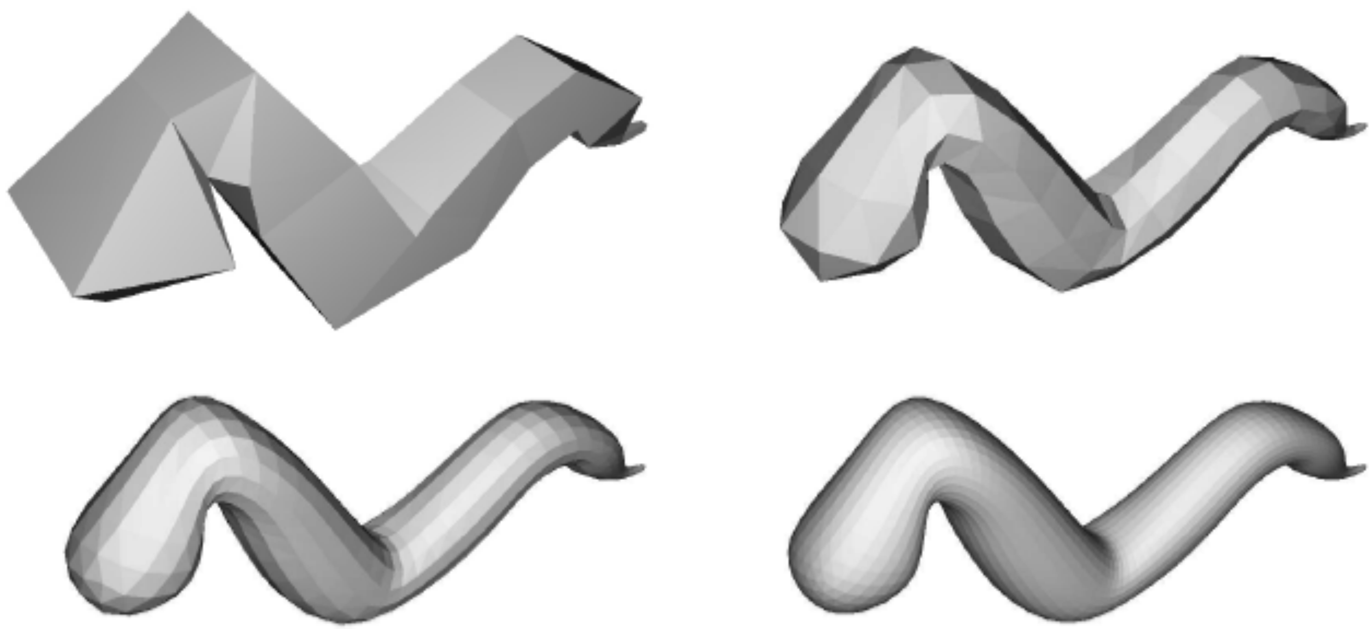


Figure 13.37: A worm subdivided three times with Loop's subdivision scheme.

C2 for regular vertices
C1 for irregular vertices

Limiting Surface

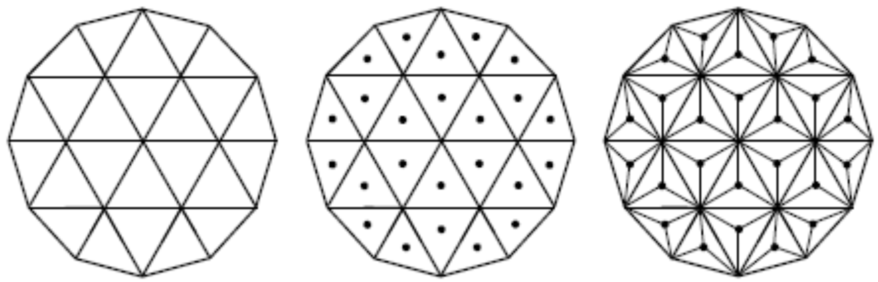
- Position and tangent of a vertex of the limiting surface can be computed directly

$$\mathrm{p}^{\infty} = (1 - n\beta)\mathrm{p}^k + \beta(\mathrm{p}_0^k + \cdots + \mathrm{p}_{n-1}^k),$$

$$\gamma(n) = \frac{1}{n + \frac{3}{8\beta(n)}}.$$

$$\mathrm{t}_u = \sum_{i=0}^{n-1} \cos(2\pi i/n)\mathrm{p}_i^k, \quad \mathrm{t}_v = \sum_{i=0}^{n-1} \sin(2\pi i/n)\mathrm{p}_i^k.$$

Sqrt(3) subdivision



Sqrt(3) subdivision

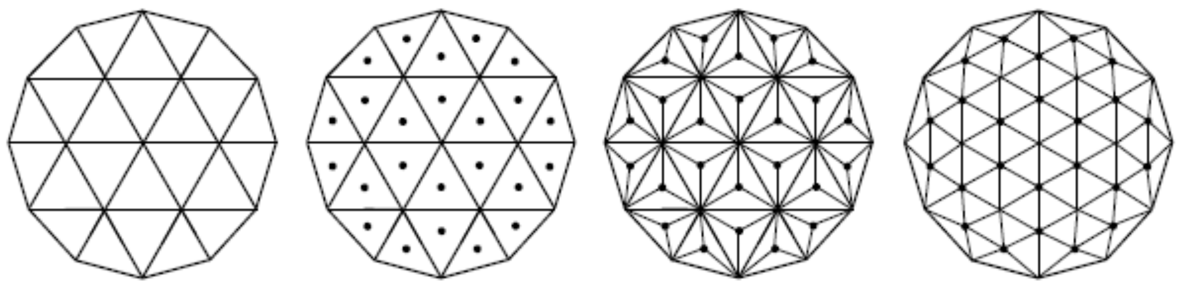
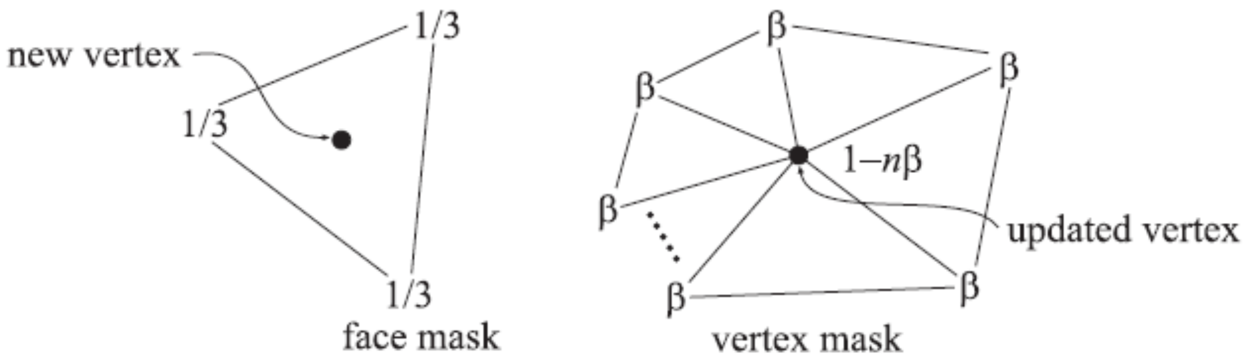


Figure 13.44: Illustration of the $\sqrt{3}$-subdivision scheme. A 1-to-3 split is performed instead of a 1-to-4 split as for Loop's and the modified butterfly schemes. First, a new vertex is generated at the center of each triangle. Then, this vertex is connected to the triangle's three vertices. Finally, the old edges are flipped. *(Illustration after Kobbelt [505].)*

Sqrt(3) subdivision



$$p_m^{k+1} = (p_a^k + p_b^k + p_c^k)/3$$

$$p^{k+1} = (1 - n\beta)p^k + \beta \sum_{i=0}^{n-1} p_i^k$$

$$\beta(n) = \frac{4 - 2\cos(2\pi/n)}{9n}$$

C2 for regular vertices
C1 for irregular vertices

Sqrt(3) subdivision

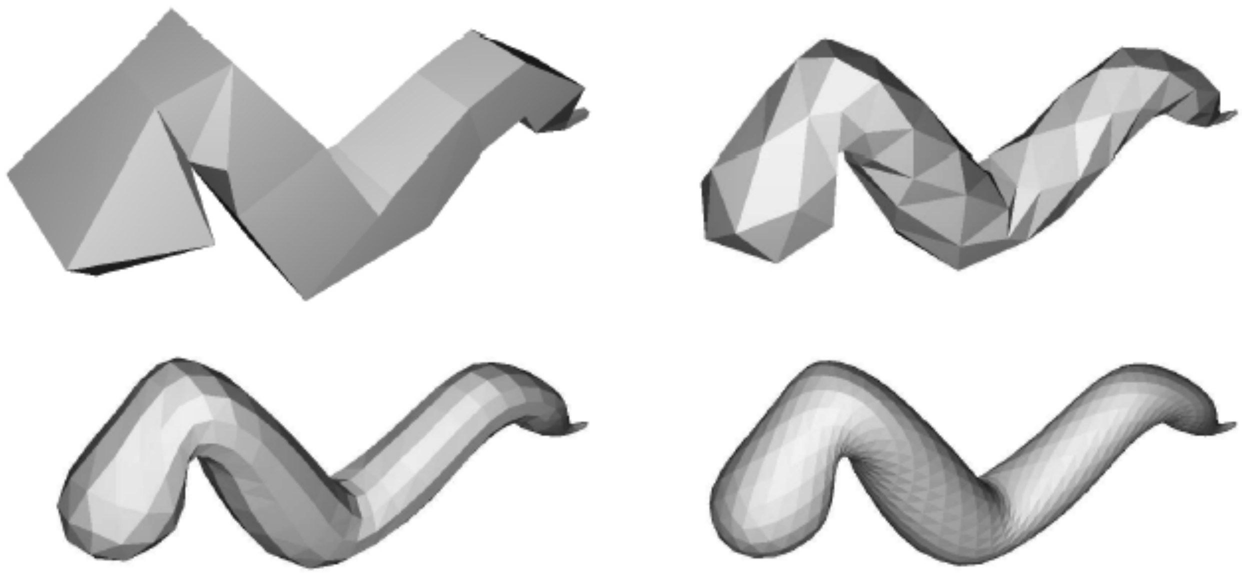


Figure 13.46: A worm is subdivided three times with the $\sqrt{3}$-subdivision scheme.

Sqrt(3) vs Loop

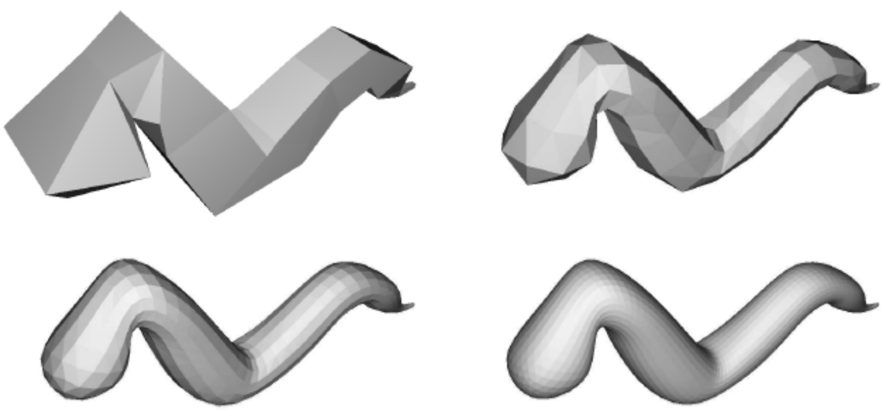


Figure 13.37: A worm subdivided three times with Loop's subdivision scheme.

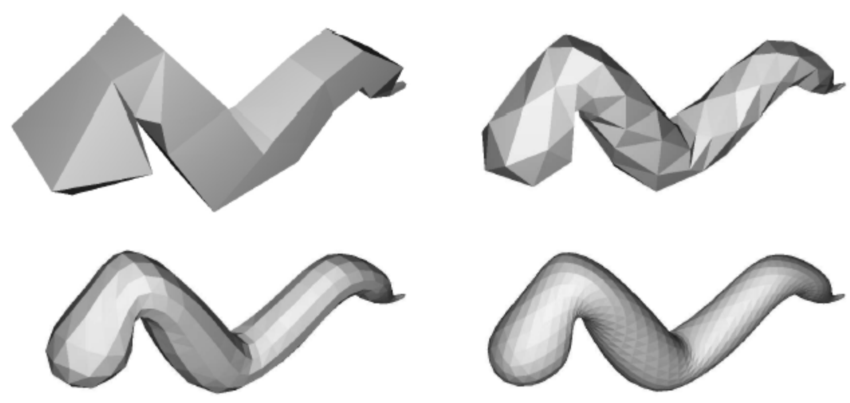


Figure 13.46: A worm is subdivided three times with the $\sqrt{3}$-subdivision scheme.

+ slower triangle growth rate
+ better for adaptive subdivision
- Edge flipping adds complexity
- Less intuitive at first few iterations