CS559: Computer Graphics

Lecture 33: Shape Modeling

Li Zhang

Spring 2008

Today

- Shape Modeling

- Reading
  - Real-Time Rendering, 2e, 12.2.1 (except Rational Bezier Patches)
    - Linux: /p/course/cs559-lizhang/public/readings/rtr-12-curves-surfaces.pdf
    - Windows: P:\course\cs559-lizhang\public\readings\rtr-12-curves-surfaces.pdf

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();
```

OpenGL and Vertex Indirection

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

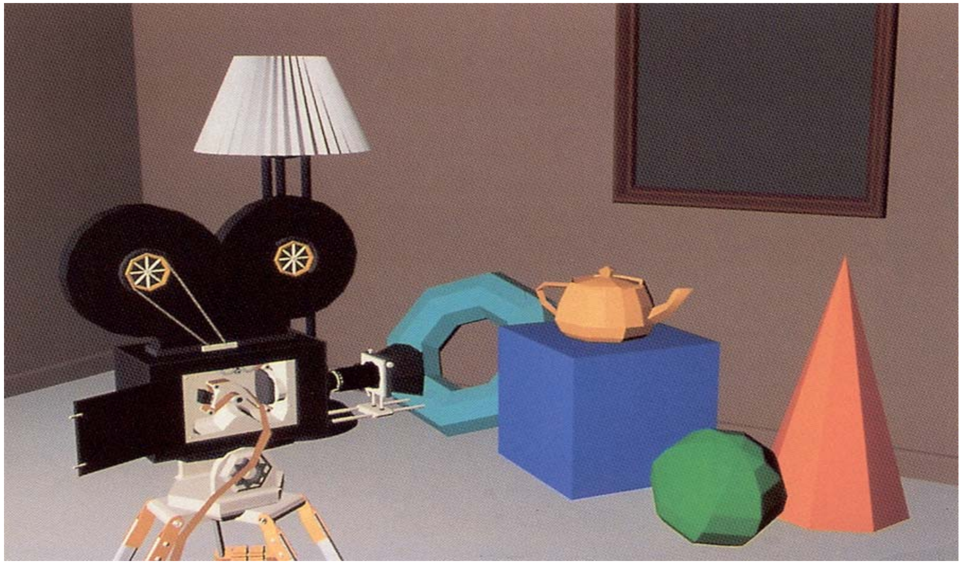
```
glEnableClientState(GL_VERTEX_ARRAY)
glVertexPointer(3, GL_FLOAT, sizeof(struct Vertex), mesh.vertices);

for ( i = 0 ; i < n ; i++ )
    glDrawElements(GL_TRIANGLES, 3, GL_UNSIGNED_INT,
                   mesh.triangles[i].verts);
```

- Minimizes amount of data sent to the renderer
- Fewer function calls, Faster!
- Other tricks to accelerate using array, see Red book, Ch 2 on vertex arrays

Normal Vectors in Mesh

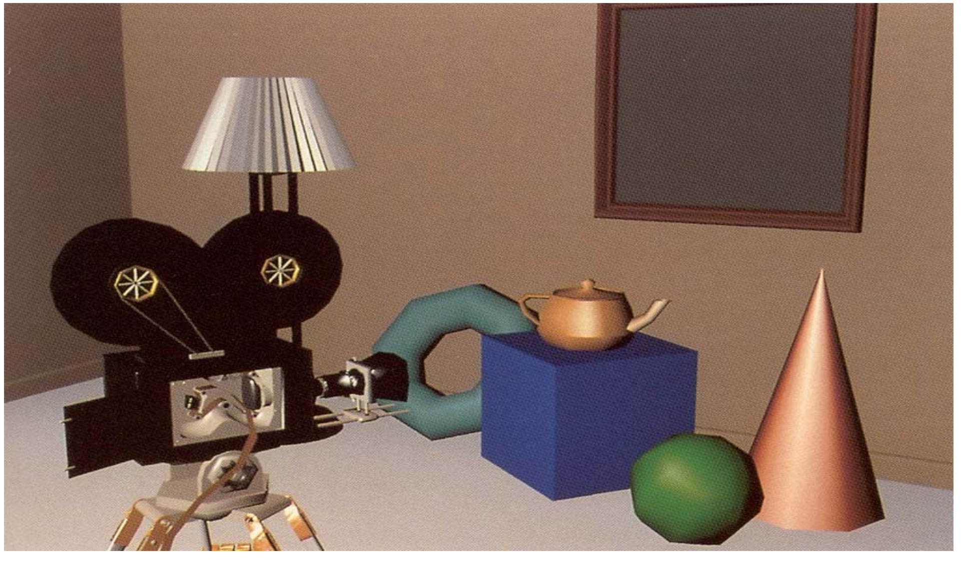
- Normal vectors give information about the true surface shape
- Per-Face normals:
  - One normal vector for each face, stored as part of face (Flat shading)



```
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];
}
```

Storing Other Information

- Colors, Texture coordinates and so on can all be treated like vertices or normals
- Lighting/Shading coefficients may be per-face, per-object, or per-vertex

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];
}
```

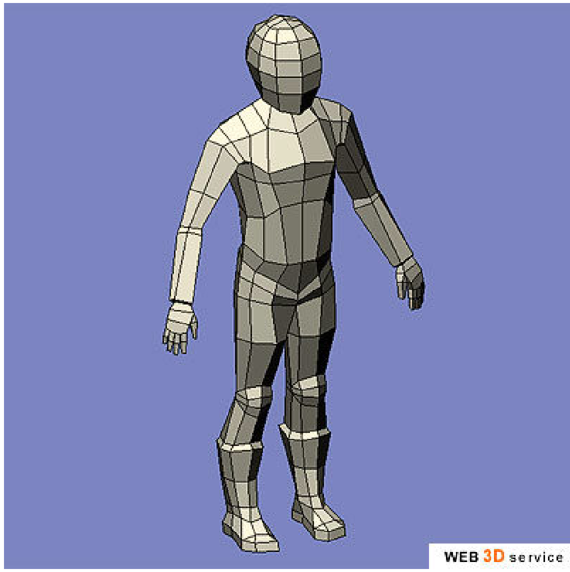
Other Data in Mesh

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

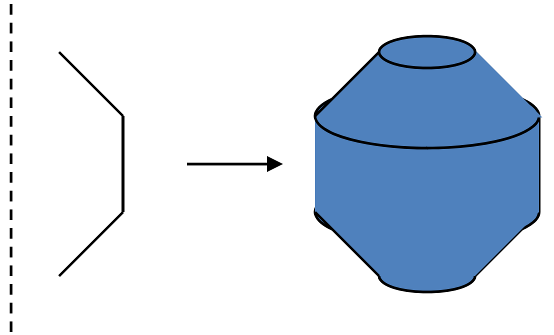
Issues with Polygons

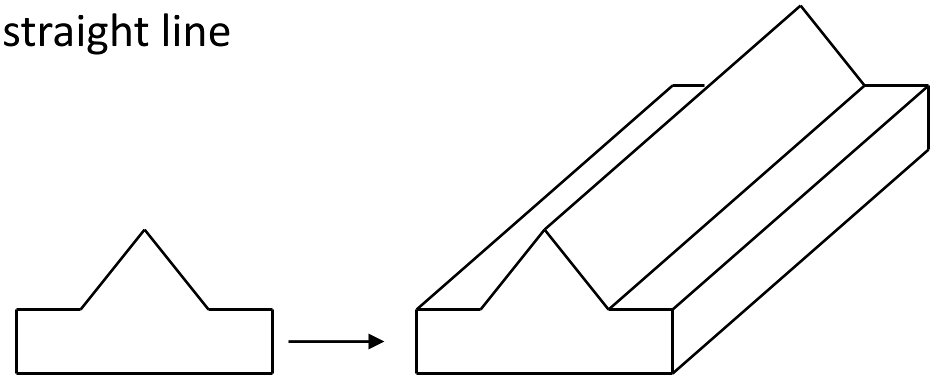
- They are inherently an approximation
  - Things like silhouettes can never be perfect without very large numbers of polygons, and corresponding expense
  - Normal vectors are not specified everywhere
- Interaction is a problem
  - Dragging points around is time consuming
  - Maintaining things like 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



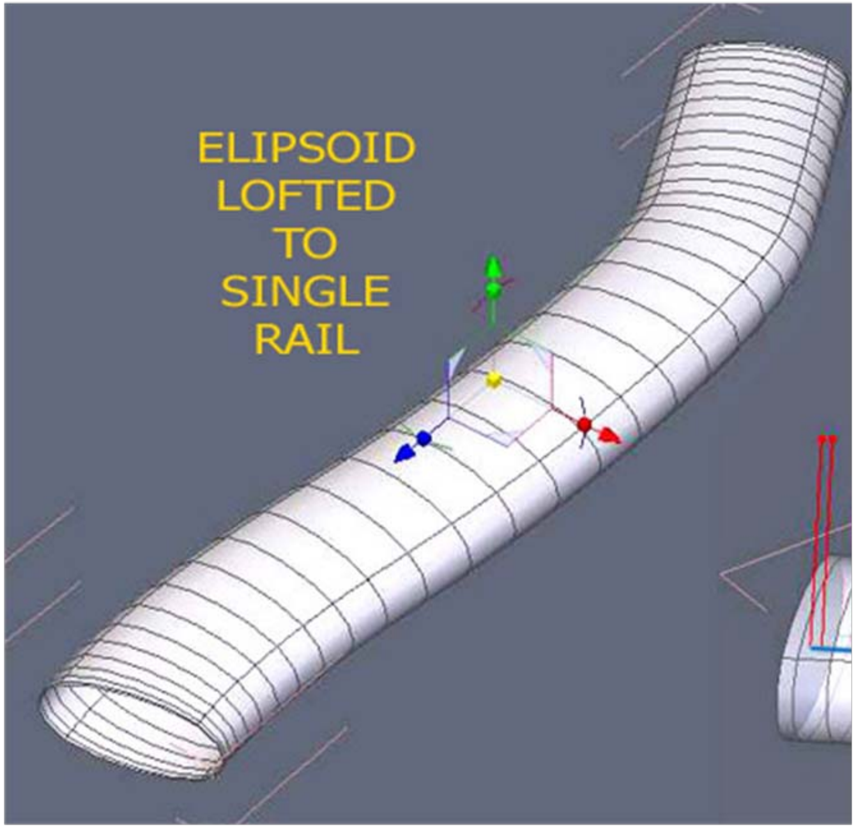


Rendering Sweeps

- Convert to polygons
  - Break path into short segments
  - Create a copy of the sweep polygon at each segment
  - Join the corresponding vertices between the polygons
  - May need things like end-caps on surfaces of revolution and extrusions
- Normals?
  - Normals come from sweep polygon and path orientation
- Texture Coord?
  - Sweep polygon defines one texture parameter, sweep path defines the other

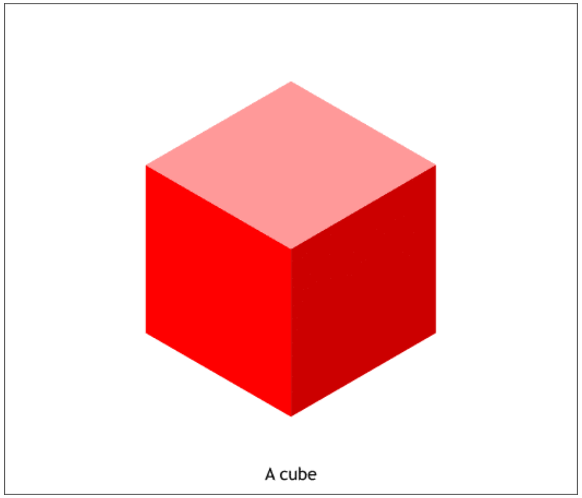
General Sweeps

- The path maybe any curve

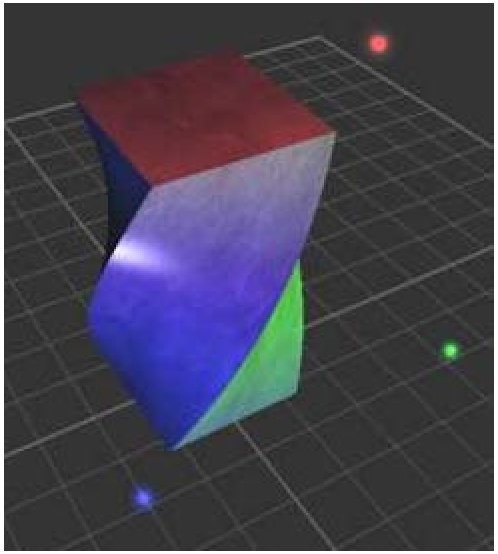


General Sweeps

- 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

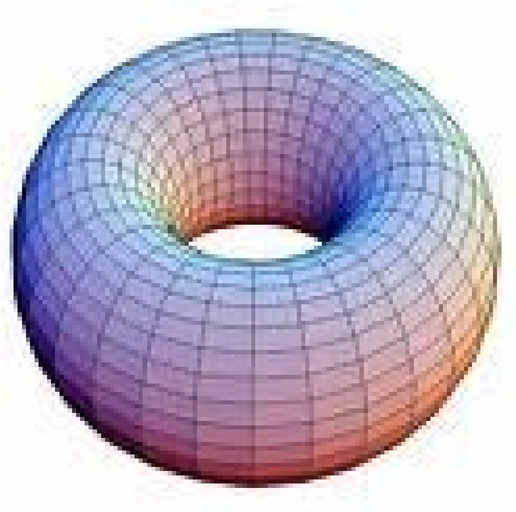
General Sweeps

- 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, …

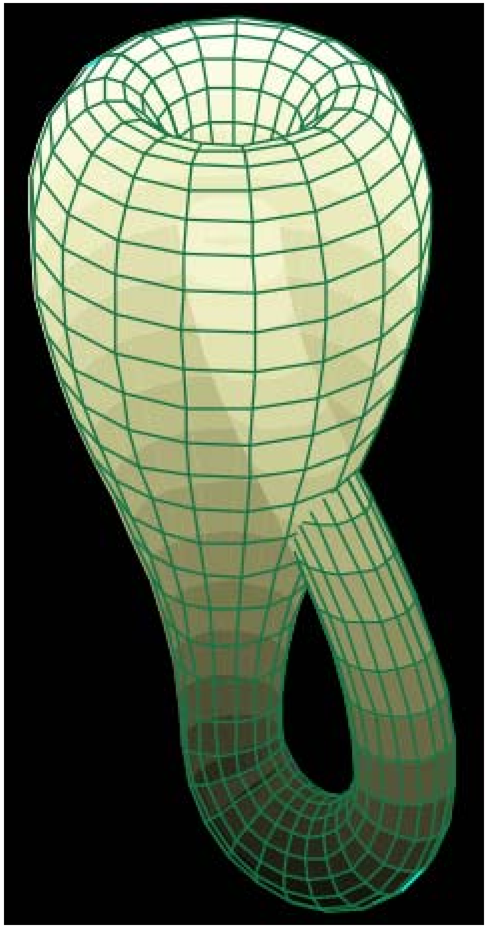
General Sweeps

- 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

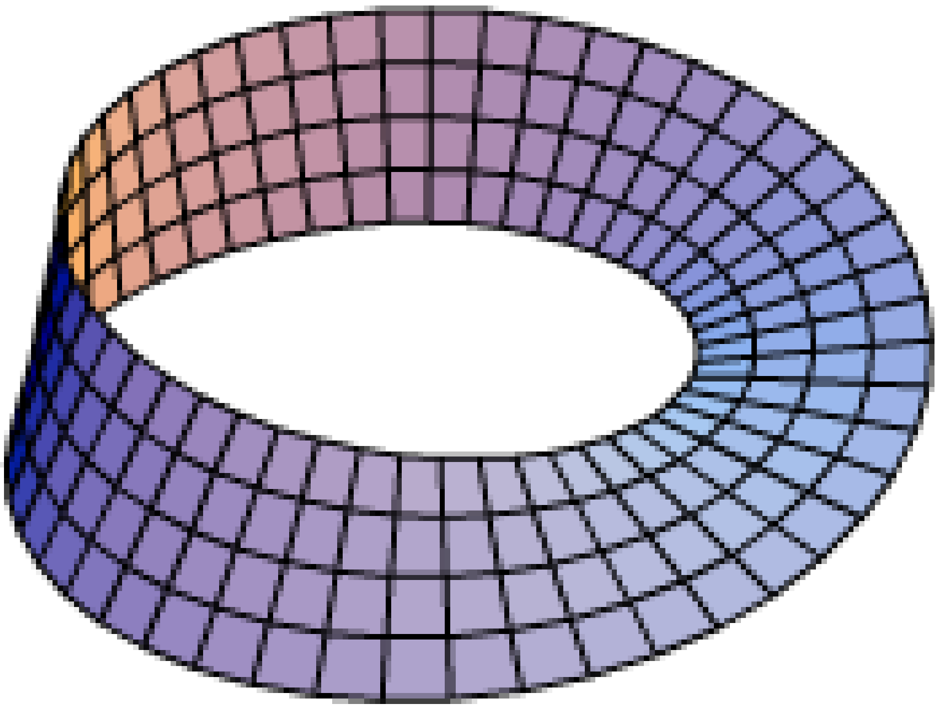


Torus



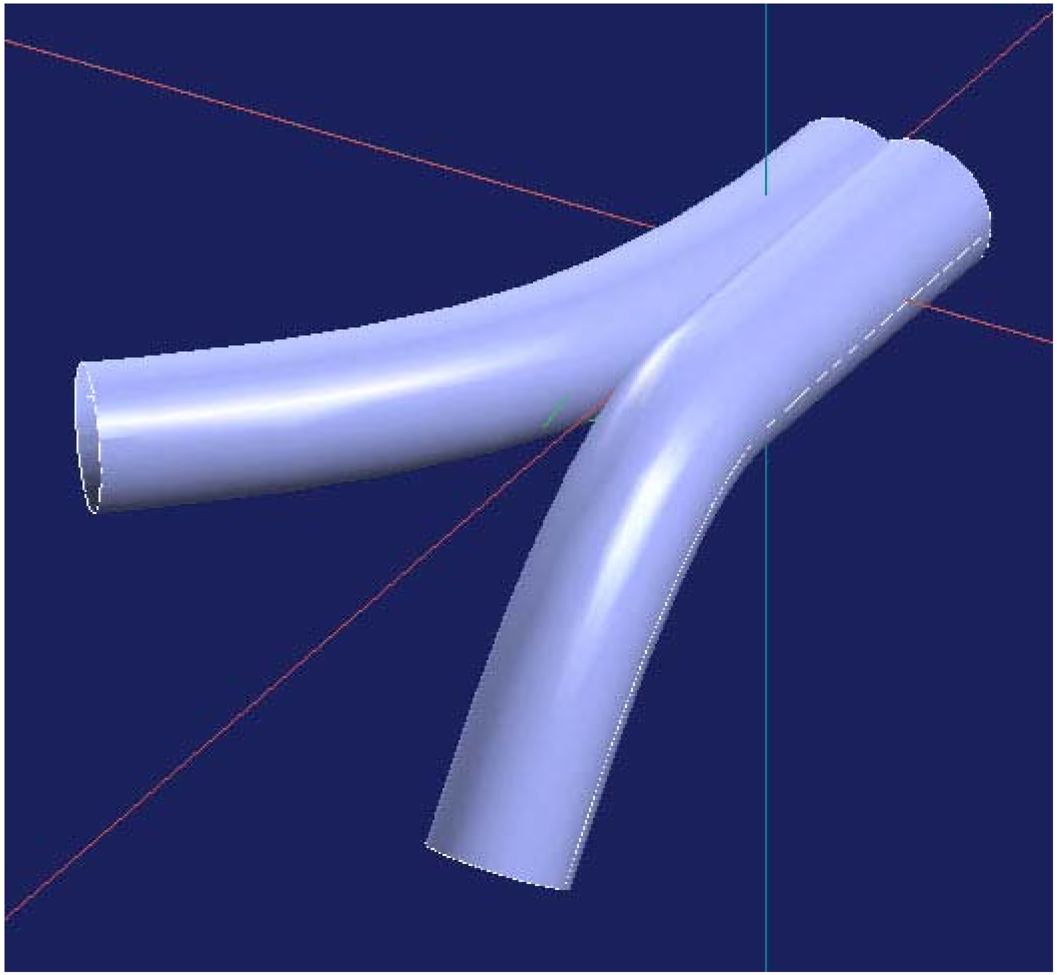
Klein Bottle

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)

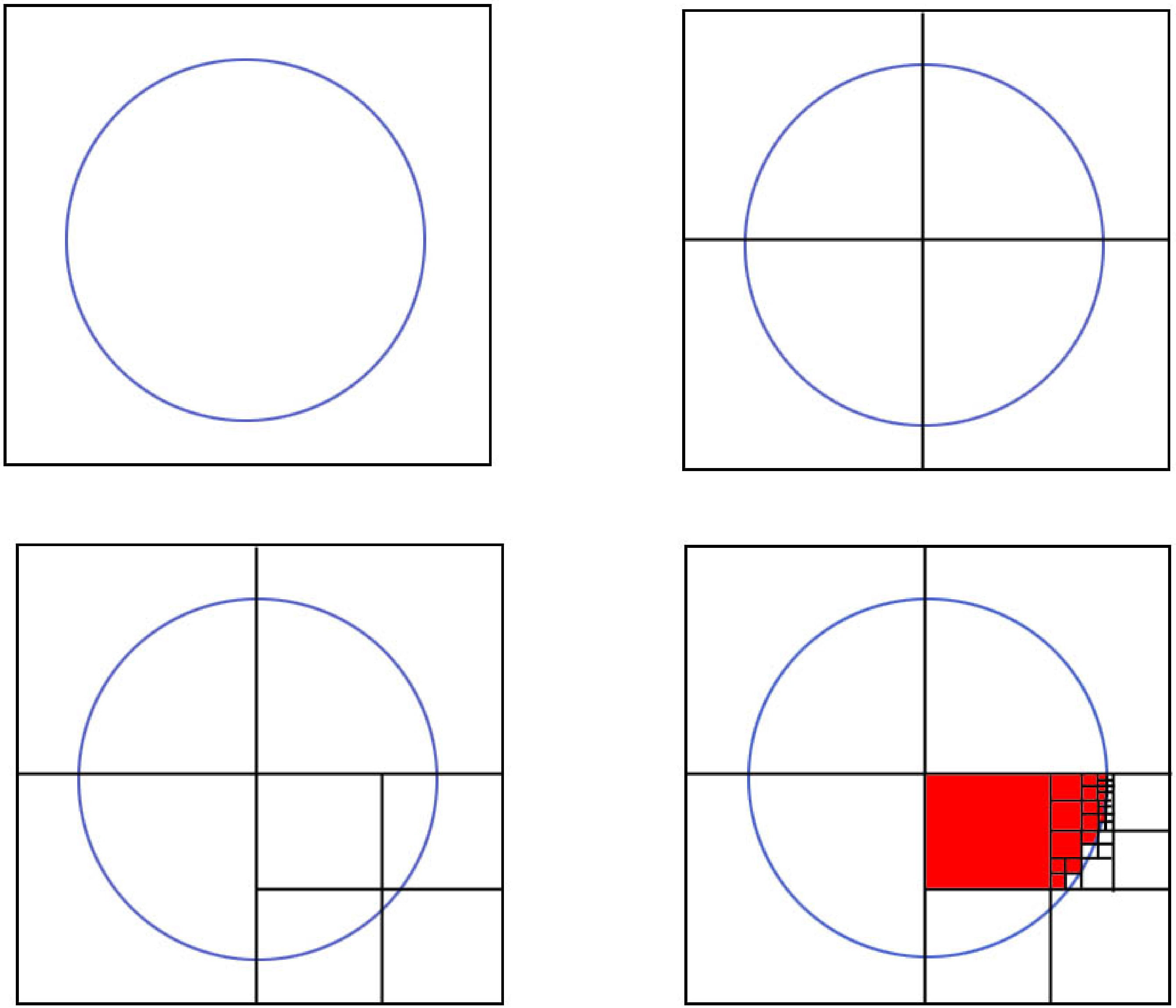
Spatial Enumeration

- 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

Octrees (and Quadtrees)

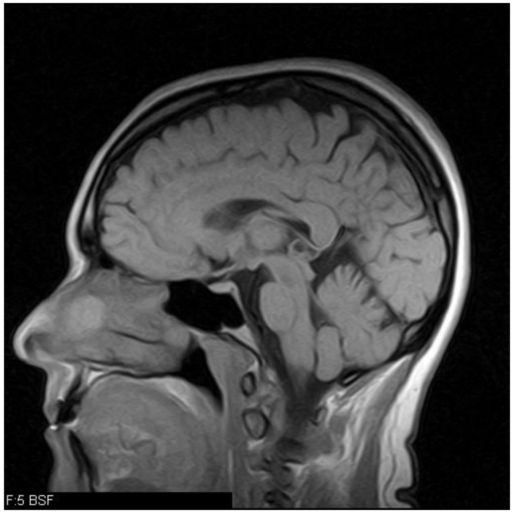
- Build a tree for adaptive voxel resolution
  - Large voxel for smooth regions
  - Small voxel for fine structures
- Quadtree is for 2D (four children for each node)
- Octree is for 3D (eight children for each node)

Quadtree example



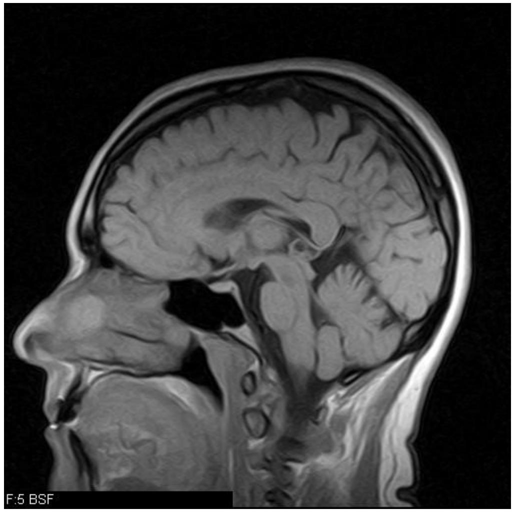
Rendering Octrees

- 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

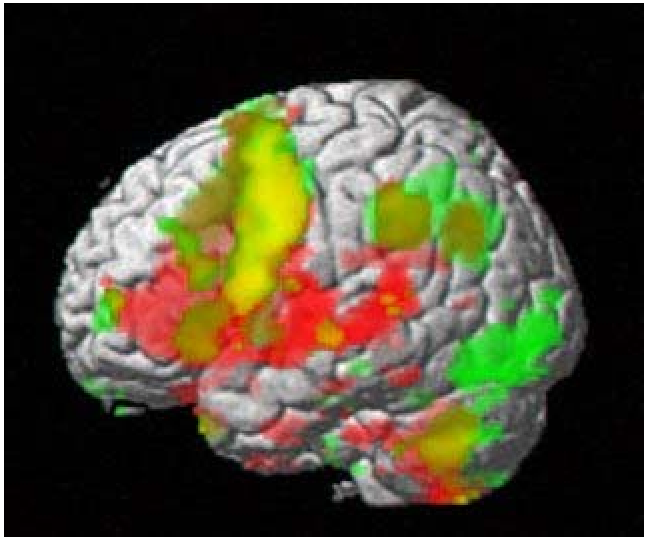


Rendering Octrees

- Typically render with colors that indicate something about the data



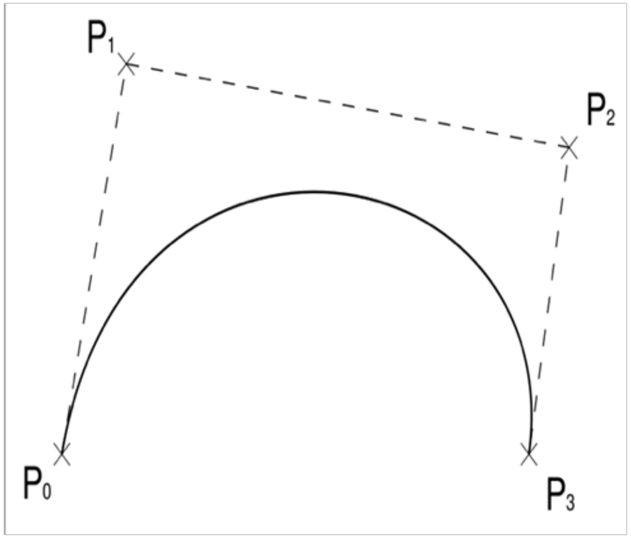
One MRI slice

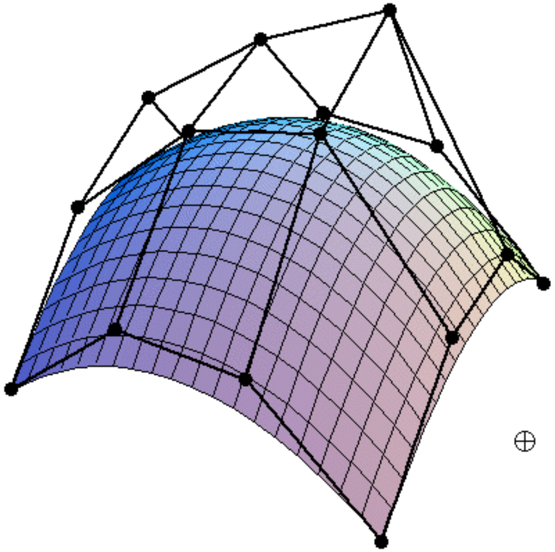


Surface rendering with color coded brain activity

Parametric surface

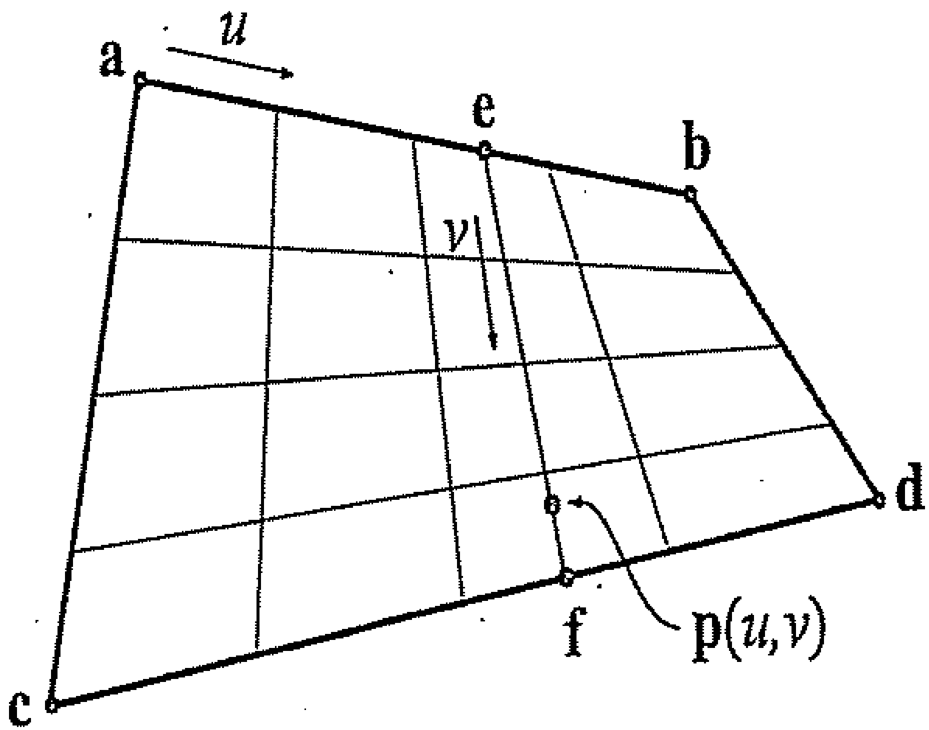
- Line Segments (1D) -> polygon meshes (2D)
- Cubic curves (1D) -> BiCubic Surfaces (2D)
  - Bezier curve -> Bezier surface





Bilinear Bezier Patch

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



$$e = (1-u)a + ub,$$

$$f = (1-u)c + ud.$$

Looks familiar?

$$p(u,v) = (1-v)e + vf$$

$$= (1-u)(1-v)a + u(1-v)b + (1-u)vc + uvd.$$