Supplement to Lecture 11

Bezier, B-Splines and Rendering in OpenGl

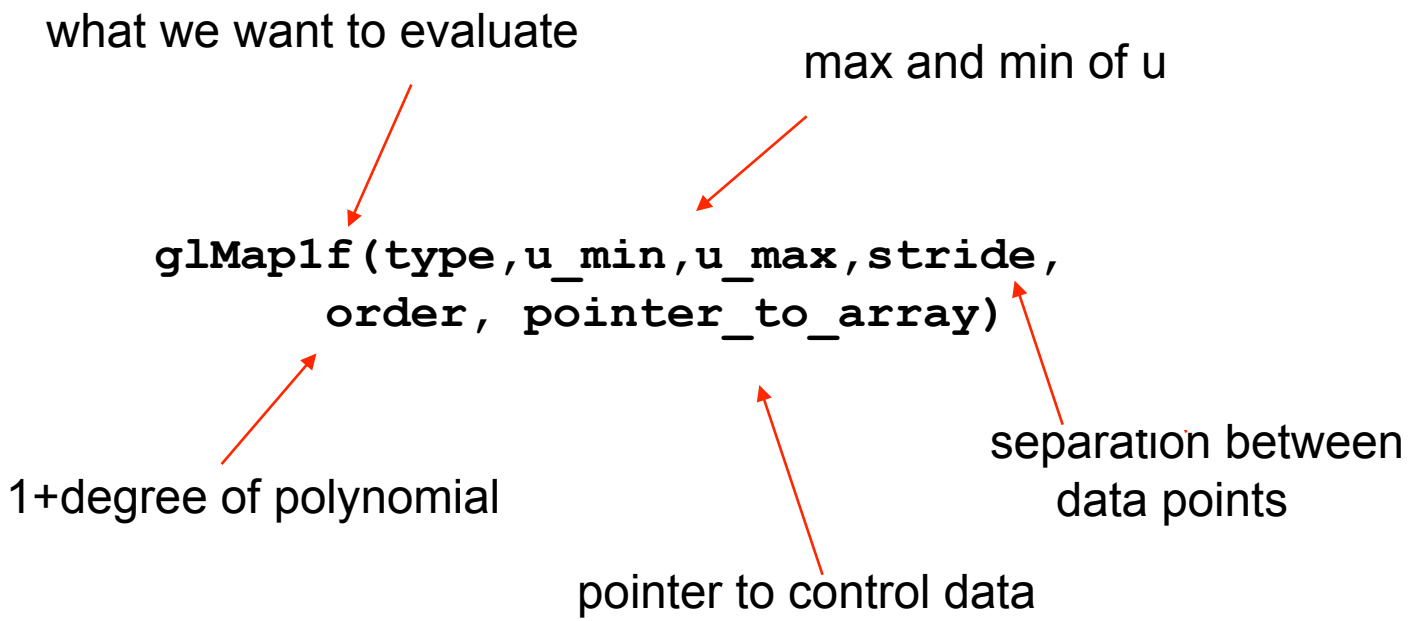
OpenGL Support

- Evaluators: a general mechanism for working with the Bernstein polynomials
  - Can use any degree polynomials
  - Can use in 1-4 dimensions
  - Automatic generation of normals and texture coordinates
  - NURBS supported in GLU
- Quadrics
  - GLU and GLUT contain polynomial approximations of quadrics

1-D Evaluators

- Evaluate a Bernstein polynomial of any degree at a set of specified values
- Can evaluate a variety of variables
  - Points along a 2, 3 or 4 dimensional curve
  - Colors
  - Normals
  - Texture Coordinates
- We can set up multiple evaluators that are all evaluated for the same value

Setting up an Evaluator



Each type must be enabled by **`glEnable(type)`**

Cubic Bezier Example

Consider an evaluator for a cubic Bezier curve over (0,1)

```
point data[ ]={……..}; * /3d data /*
glMap1f(GL_MAP_VERTEX_3,0.0,1.0,3,4,data);
```

data are 3D vertices

data are arranged as x,y,z,x,y,z……
three floats between data points in array

cubic

```
glEnable(GL_MAP_VERTEX_3);
```

Evaluation

- The function **`glEvalCoord1f(u)`** causes all enabled evaluators to be evaluated for the specified **`u`**
  - Can replace **`glVertex`**, **`glNormal`**, **`glTexCoord`**
- The values of **`u`** need not be equally spaced

Piecewise Linear Approx.

• Consider the previous evaluator that was set up for a cubic Bezier over (0,1)

• Suppose that we want to approximate the curve with a 100 point polyline

```
glBegin(GL_LINE_STRIP)
  for(i=0; i<100; i++)
    glEvalCoord1f( (float) i/100.0);
glEnd();
```

Equally spaced Points

Rather than use a loop, we can set up an equally spaced mesh (grid) and then evaluate it with one function call

```
glMapGrid(100, 0.0, 1.0);
```

sets up 100 equally-spaced points on (0,1)

```
glEvalMesh1(GL_LINE, 0, 99);
```

renders lines between adjacent evaluated points from point 0 to point 99

Bezier Surfaces

- Similar procedure to 1D but use 2D evaluators in **u** and **v**
- Set up with

```
glMap2f(type, u_min, umax, u_stride,
 u_order, v_min, v_max, v_stride,
 v_order, pointer_to_data)
```

- Evaluate with **glEvalCoord2f(u,v)**

Bicubic Bezier Surfaces

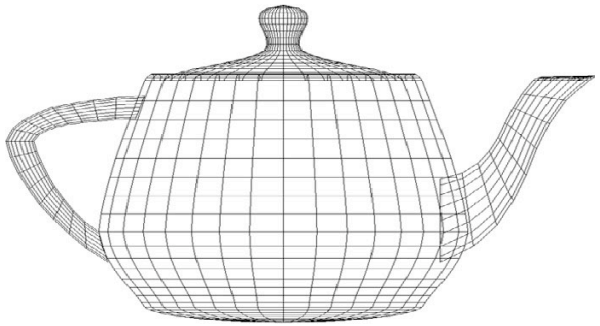
bicubic over (0,1) x (0,1)

```
point data[4][4]={.....};
glMap2f(GL_MAP_VERTEX_3, 0.0, 1.0, 3, 4,
    0.0, 1.0, 12, 4, data);
```

Note that in v direction data points are separated by 12 floats since array `data` is stored by rows

Rendering with Lines

must draw in both directions



```
for(j=0;j<100;j++) {
  glBegin(GL_LINE_STRIP);
    for(i=0;i<100;i++)
       glEvalCoord2f((float) i/100.0, (float) j/100.0);
  glEnd();
glBegin(GL_LINE_STRIP);
    for(i=0;i<100;i++)
       glEvalCoord2f((float) j/100.0, (float) i/100.0);
  glEnd();
}
```

Rendering with Quads

We can form a quad mesh and render with lines

```
for(j=0; j<99; j++) {
  glBegin(GL_QUAD_STRIP);
   for(i=0; i<100; i++) {
      glEvalCoord2f ((float) i/100.0,
          (float) j/100.0);
      glEvalCoord2f ((float)(i+1)/100.0,
          (float)j/100.0);
   }
   glEnd():
}
```

Uniform Meshes

- We can form a 2D mesh (grid) in a similar manner to 1D for uniform spacing

```
glMapGrid2(u_num, u_min, u_max,
 v_num, v_min, v_max)
```

- Can evaluate as before with lines or if want filled polygons

```
glEvalMesh2( GL_FILL, u_start,
 u_num, v_start, v_num)
```

Rendering with Filled Quads

- If we use filled polygons, we have to shade or we will see solid color uniform rendering
- Can specify lights and materials but we need normals
  - Let OpenGL find them

```
glEnable(GL_AUTO_NORMAL)
```

NURBS

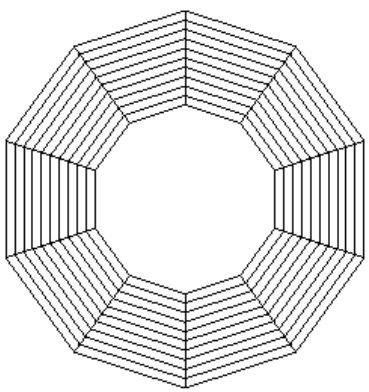
- OpenGL supports NURBS surfaces through the GLU library
- Why GLU?
  - Can use evaluators in 4D with standard OpenGL library
  - However, there are many complexities with NURBS that need a lot of code
  - There are five NURBS surface functions plus functions for trimming curves that can remove pieces of a NURBS surface

Quadrics in GLU & GLUT

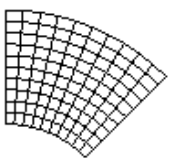
- Quadrics are in both the GLU and GLUT libraries
  - Both use polygonal approximations where the application specifies the resolution
  - Sphere: lines of longitude and lattitude
- GLU: disks, cylinders, spheres
  - Can apply transformations to scale, orient, and position
- GLUT: Platonic solids, torus, Utah teapot, cone

Quadric Objects in GLU

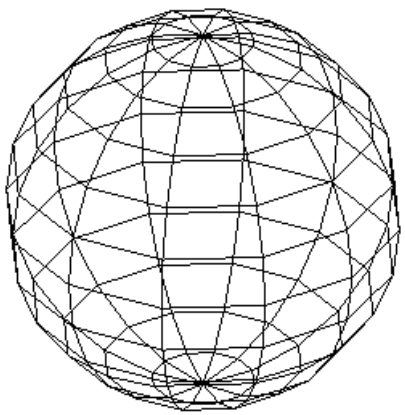
- GLU can automatically generate normals and texture coordinates
- Quadrics are objects that include properties such as how we would like the object to be rendered



disk



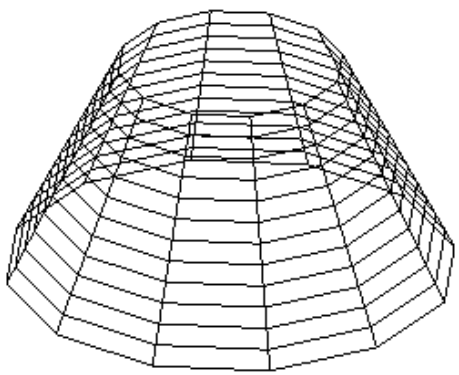
partial disk



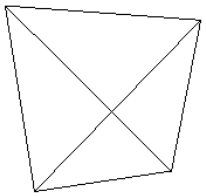
sphere

GLU Cylinder

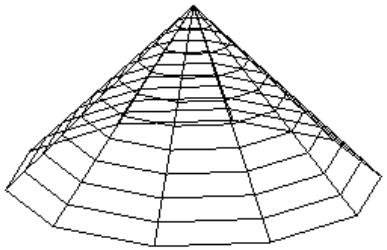
```
GLUquadricOBJ *p;
P = gluNewQuadric(); /*set up object */
gluQuadricDrawStyle(GLU_LINE);/*render
 style*/
gluCylinder(p, BASE_RADIUS, TOP_RADIUS,
        BASE_HEIGHT, sections, slices);
```



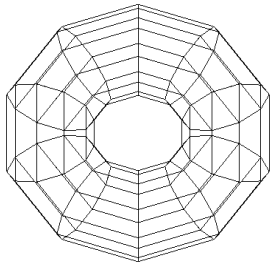
GLUT Objects



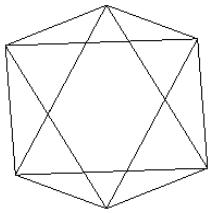
`glutWireTetrahedron()`



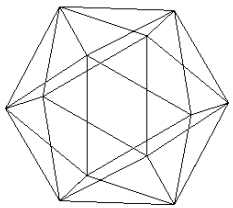
`glutWireCone()`



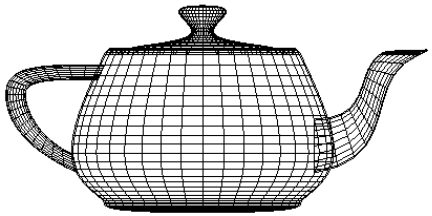
`glutWireTorus()`



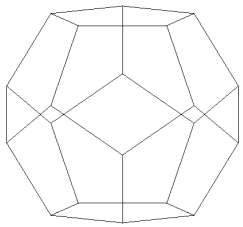
`glutWireOctahedron()`



`glutWireIcosahedron()`



`glutWireTeapot()`



`glutWireDodecahedron()`