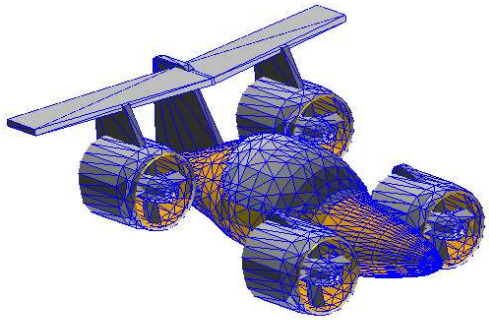
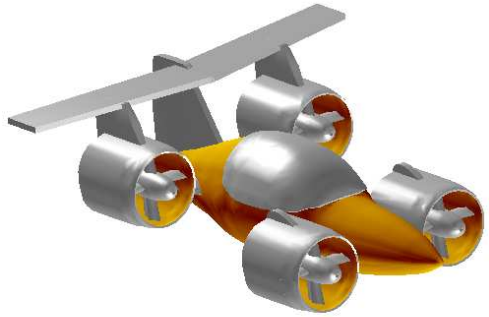
B-Splines and OpenGL/GLU





Spline Curves

- Successive linear blend
- Basis polynomials
- Recursive evaluation
- Properties
- Joining segments

Tensor-product-patch Spline Surfaces

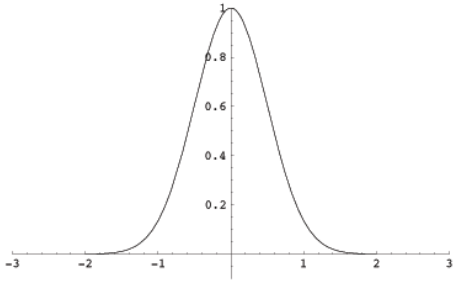
- Tensor product patches
- Recursive Evaluation
- Properties
- Joining patches

OpenGL and Glut Support

Splines

If give up on small support, get natural splines; every control point influences the whole curve.

If give up on interpolation, get cubic B-splines.



B-Splines

Need only one basis function, all $B_i(t)$ are obtained by shifts: $B_i(t) = B(t - i)$.
The basis function is piecewise polynomial:

$$B(t) = \begin{cases} 0 & t \leq -2 \\ \frac{1}{6}t^3 + 2t + \frac{4}{3} + t^2 & t \leq -1 \\ \frac{2}{3} - t^2 - \frac{1}{2}t^3 & t \leq 0 \\ \frac{2}{3} - t^2 + \frac{1}{2}t^3 & t \leq 1 \\ \frac{4}{3} - 2t + t^2 - \frac{1}{6}t^3 & t \leq 2 \\ 0 & 2 \leq t \end{cases}$$

B-Splines

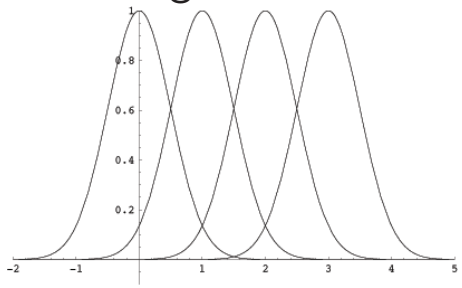
The curve with control points $p_0, p_1, p_2, \ldots p_n$ is computed using

$$p(t) = \sum_{i=0}^{n} p_i \; B(t-i)$$

The allowed range of t is from 1 to $n-1$; outside this interval our functions do not sum up to 1, which means in particular that if we move control points together in the same way, the curve outside the interval will not move rigidly.

B-Splines

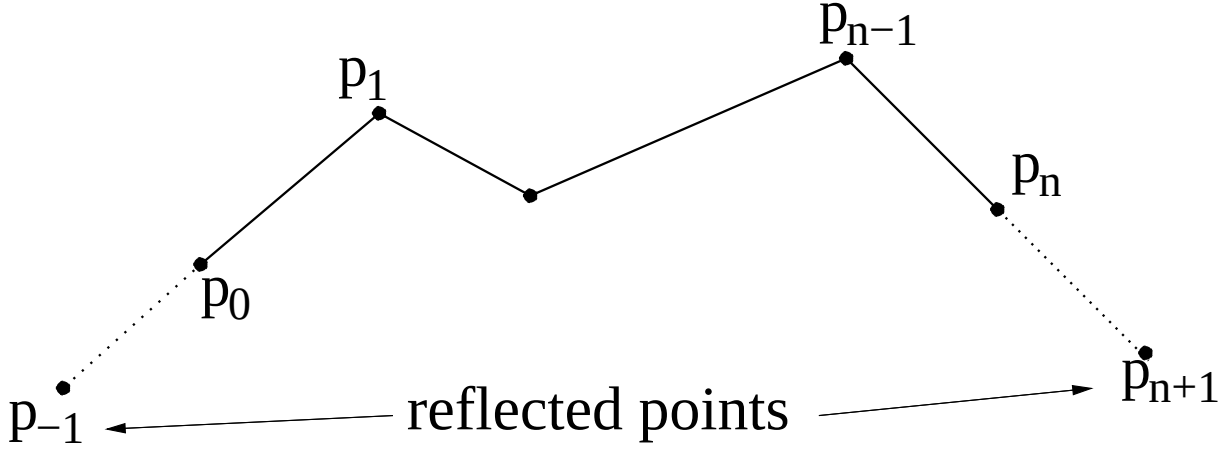
The minimal number of points required is 4;
this corresponds to the interval for t of length 1.



This is inconvenient - but we can always add control points by reflection.

B-Splines

Adding control points by reflection:



$$p_{-1} = 2p_0 - p_1; \qquad p_{n+1} = 2p_n - p_{n-1}$$

Drawing B-Splines

Hardware typically can draw only line segments. Need to approximate B-spline with piecewise linear curve. Simplest approach:

Choose small Δt.
Compute points $p(0), p(\Delta t), p(2\Delta t), \cdots$
Draw line segments connecting the points.

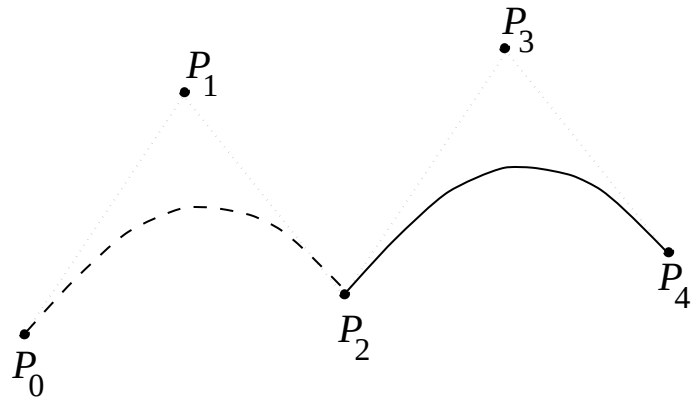
Not very efficient — have to evaluate a cubic polynomial (or several) at each point.

Can do better using a magic algorithm (subdivision). Next lecture!

Another Formulation: Discontinuities in Bézier Splines

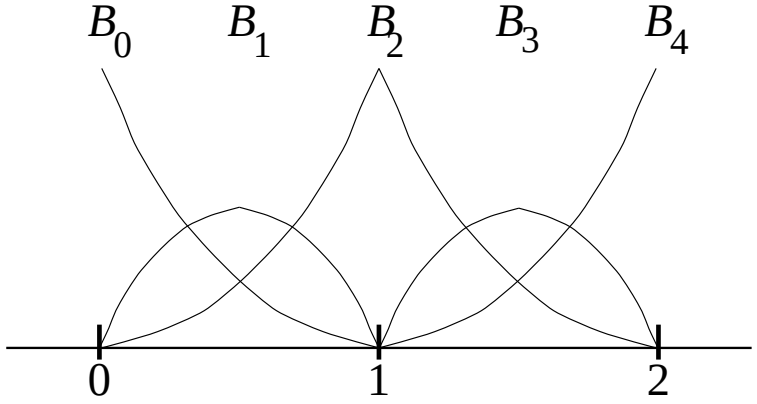
Bézier Discontinuities:

- Two Bézier segments can be completely disjoint
- Two segments join if they share last/first control point



Common Parameterization and Blending Functions

- Joined curves can be given common parameterization
  - Parameterize first segment with $0 \leq t < 1$
  - Parameterize nest segment with $1 \leq t \leq 2$, etc.
- Look at blending/basis polynomials under this parameterization
  - Combine those for common P_j into a single piecewise polynomial



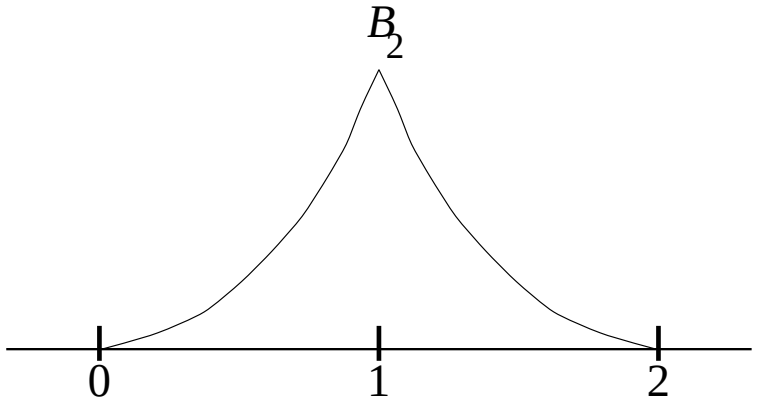
Combined Curve Segments

- Curve is $P(t) = P_0B_0(t) + P_1B_1(t) + P_2B_2(t) + P_3B_3(t) + P_4B_4(t)$, where

$$
\begin{aligned}
B_0(t) &= \begin{cases} (1-t)^2 & 0 \le t < 1 \\ 0 & 1 \le t \le 2 \end{cases} \\
B_1(t) &= \begin{cases} 2((1-t)t & 0 \le t < 1 \\ 0 & 1 \le t \le 2 \end{cases} \\
B_2(t) &= \begin{cases} t^2 & 0 \le t < 1 \\ (2-t)^2 & 1 \le t \le 2 \end{cases} \\
B_3(t) &= \begin{cases} 0 & 0 \le t < 1 \\ 2(2-t)(t-1) & 1 \le t \le 2 \end{cases} \\
B_4(t) &= \begin{cases} 0 & 0 \le t < 1 \\ (t-1)^2 & 1 \le t \le 2 \end{cases}
\end{aligned}
$$

Curve Discontinuities from Basis Discontinuities

- P_2 is scaled by $B_2(t)$, which has a discontinuous derivative
- The corner in the curve results from this discontinuity



Spline Continuity

Smoother Blending Functions:

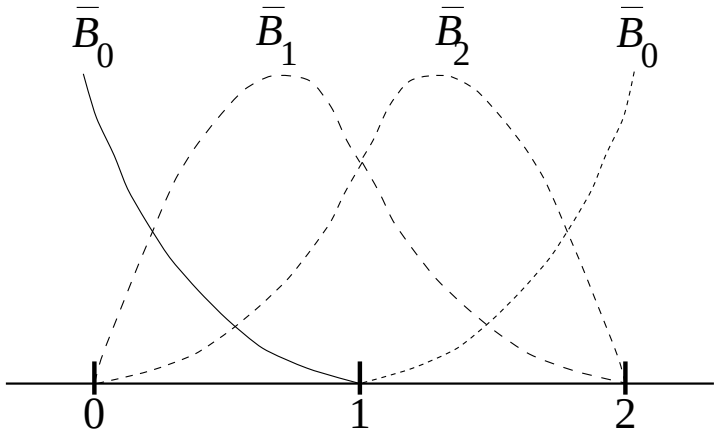
- Can $B_0(t), \ldots, B_4(t)$ be replaced by smoother functions?
  - Piecewise polynomials on $0 \leq t \leq 2$
  - Continuous derivatives
- Yes, but we lose one degree of freedom
  - Curve has no corner if segments share a common tangent
  - Tangent is given by the chords $\overline{P_1P_2}$, $\overline{P_2P_3}$
  - An equation constrains P_1, P_2, P_3
    $$P_3 - P_2 = P_2 - P_1 \Longrightarrow P_2 = \tfrac{P_1+P_3}{2}$$
- This equation leads to combinations:

$$P_0B_0(t) + P_1\left(B_1(t) + \frac{1}{2}B_2(t)\right) + P_3\left(\frac{1}{2}B_2(t) + B_3(t)\right) + P_4B_4(t)$$

Spline Basis:

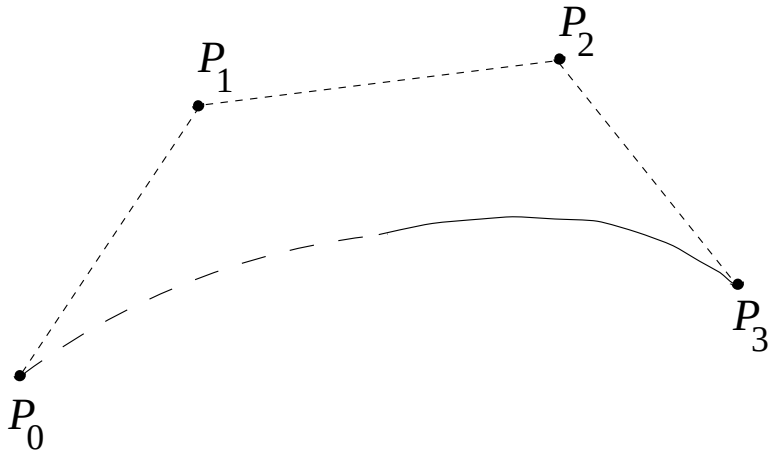
- Combined functions form a smoother *spline basis*

$$
\begin{aligned}
\overline{B}_0(t) &= B_0(t) \\
\overline{B}_1(t) &= \left(B_1(t) + \frac{1}{2} B_2(t) \right) \\
\overline{B}_2(t) &= \left(\frac{1}{2} B_2(t) + B_3(t) \right) \\
\overline{B}3_(t) &= B_4(t)
\end{aligned}
$$


$\bar{B}_0$
$\bar{B}_1$
$\bar{B}_2$
$\bar{B}_0$
0
1
2

Smoother Curves:

- Control points used with this basis produce smoother curves.

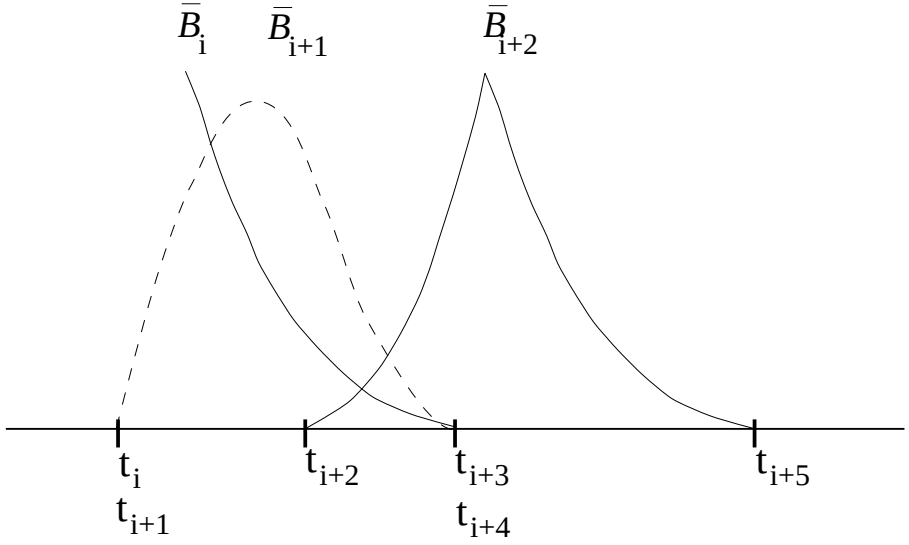


General B-Splines:

- Nonuniform B-splines (NUBS) generalize this construction
- A B-spline, $B_i^d(t)$, is a piecewise polynomial:
  - each of its segments is of degree $\leq d$
  - it is defined for all t
  - its segmentation is given by $knots\ t = t_0 \leq t_1 \leq \cdots \leq t_N$

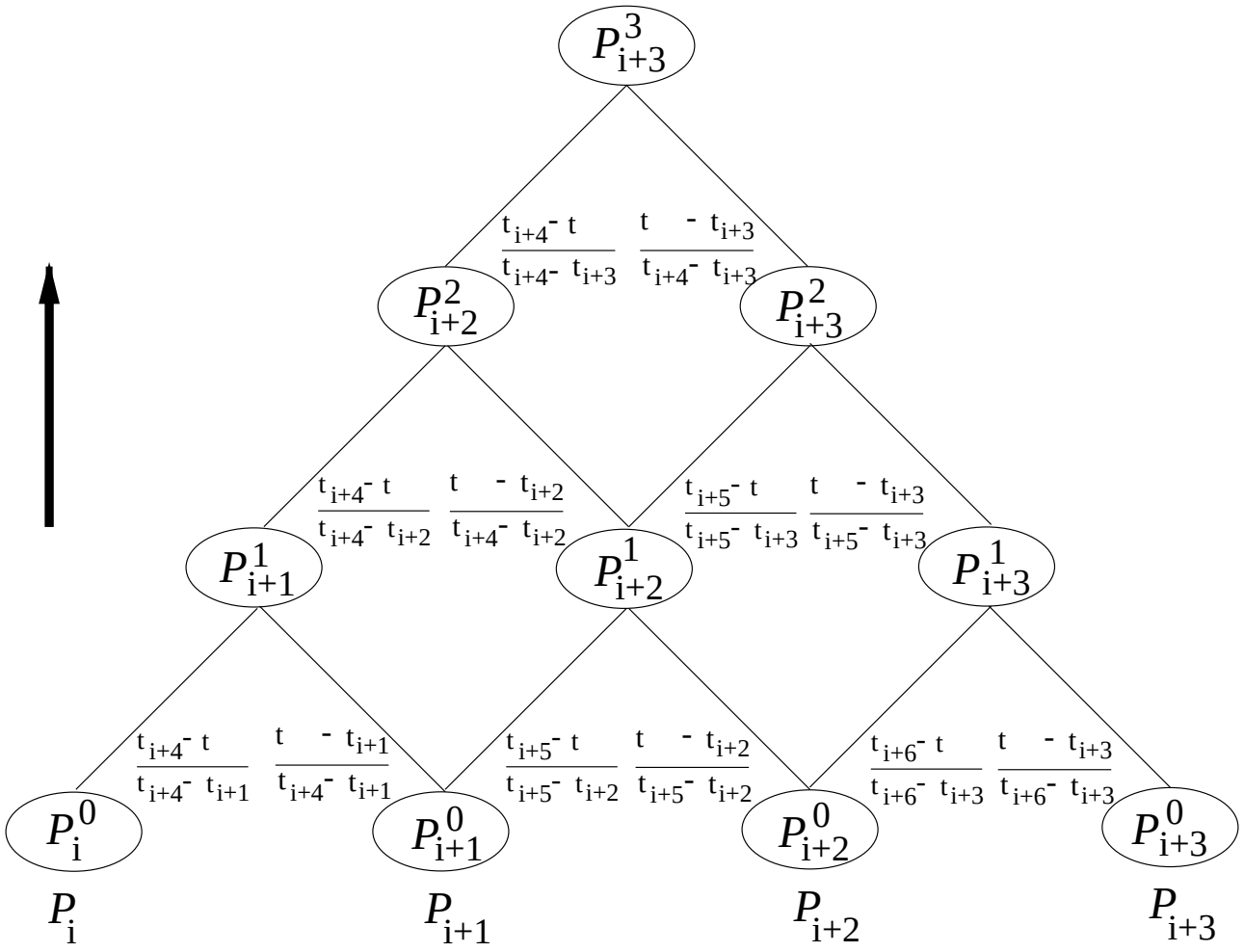
- it is zero for $T < T_i$ and $T > T_{i+d+1}$
- it may have a discontinuity in its $d - k + 1$ derivative at $t_j \in \{t_i, \ldots, t_{i+d+1}\}$, if t_j has multiplicity k
- it is nonnegative for $t_i < t < t_{i+d+1}$
- $B_i^d(t) + \cdots + B_{i+d}(t) = 1$ for $t_{i+d} \leq t < t_{i+d+1}$, and all other $B_j^d(t)$ are zero on this interval
- Bézier blending functions are the special case where all knots have multiplicity $d + 1$

Example (Quadratic):



Evaluation:

- There is an efficient, recursive evaluation scheme for any curve point
- It generalizes the triangle scheme (deCasteljau) for Bézier curves
- Example (for cubics and $t_{i+3} \leq t < t_{i+4}$):


P^3_{i+3}
$\frac{t_{i+4} - t}{t_{i+4} - t_{i+3}}$ $\frac{t - t_{i+3}}{t_{i+4} - t_{i+3}}$
P^2_{i+2}
P^2_{i+3}
$\frac{t_{i+4} - t}{t_{i+4} - t_{i+2}}$ $\frac{t - t_{i+2}}{t_{i+4} - t_{i+2}}$
$\frac{t_{i+5} - t}{t_{i+5} - t_{i+3}}$ $\frac{t - t_{i+3}}{t_{i+5} - t_{i+3}}$
P^1_{i+1}
P^1_{i+2}
P^1_{i+3}
$\frac{t_{i+4} - t}{t_{i+4} - t_{i+1}}$ $\frac{t - t_{i+1}}{t_{i+4} - t_{i+1}}$
$\frac{t_{i+5} - t}{t_{i+5} - t_{i+2}}$ $\frac{t - t_{i+2}}{t_{i+5} - t_{i+2}}$
$\frac{t_{i+6} - t}{t_{i+6} - t_{i+3}}$ $\frac{t - t_{i+3}}{t_{i+6} - t_{i+3}}$
P^0_i
P^0_{i+1}
P^0_{i+2}
P^0_{i+3}
P_i
P_{i+1}
P_{i+2}
P_{i+3}

Curves and Surfaces Programming using OpenGL and GLU

Quadrics support in GLU

Define a quadric object.

```
GLUquadricObj*p;
p=gluNewQuadric();
```

Specify a rendering Style of Quadric. Example as a wireframe.

```
gluQuadricDrawStyle(p,GLU_LINE);
```

Example a cylinder with its length along the y-axis

```
gluCylinder(p,BASE_RADIUS,BASE_RADIUS,BASE_HEIGHT,sample_circle,sample_height)
```

```
sample_circle = number of pieces of the base
```

```
sample_height = number of height pieces
```

Bézier Curves and Surfaces

Support is available through 1D, 2D, 3D, 4D *evaluators* to compute values for the polynomials used in Bézier and NURBS.

```
glMaplf(type,u_min,u_max,stride,order,point_array)

type = 3D points, 4D points, RGBA colors, normals, indexed colors,
1D to 4D texture coordinates

u_min <= parameter u  <=  u_max

stride = number of parameter values between curve segments

order = degree of polynomial + 1

control polygon = defined by point_array
```

Example an *evaluator* for a 3D cubic Bézier curve defined over (0,1) with a stride of 3 and order 4

```
point  data[]={...}
glMaplf{GL_MAP_VERTEX_3,0.0,1.0, 3,4,data};
```

Multiple evaluators can be active at the same time, and can be used to evaluate curves, normals, colors etc at the same time

To render the Bézier Curve over (0,1) with 100 line segments

```
glEnable{GL_MAP_VERTEX_3};
glBegin(GL_LINE_STRIP)
    for(i=0; i<100; i++) glEvalCoord1f((float) i/100.0);
glEnd();
```

The GLUT library has the teapot as an object. See pg 646-648, Chap 12 for display/render program a teapot using Bézier functions.

For lighting / shading using a NURBS surface, when additionally needs surface normals. These could be generated automatically, using

```
glEnable(GL_AUTO_NORMAL)
```

NURBS functions in GLU library

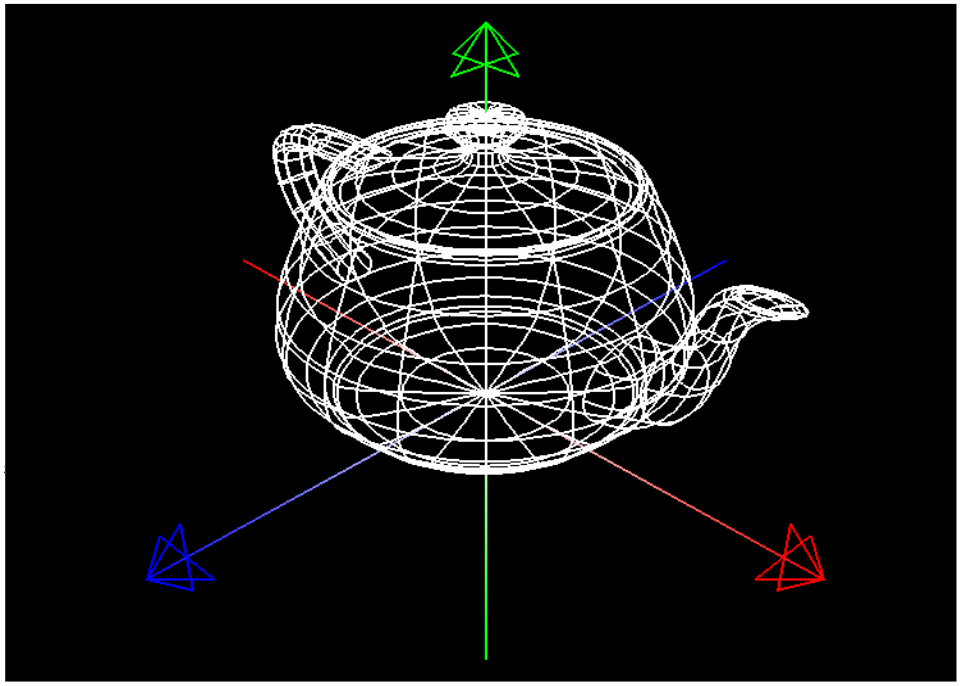
gluNewNurbsRenderer() - create a pointer to a NURBS object

gluNurbsProperty() - choose rendering values such as size of lines, polygons. Also enables a mode where the tesselated geometry can be retrieved through the callback interface

gluNurbsCallBack() - register the functions to call to retreive the tesselated geometric data or if you wish notification when an error is encountered

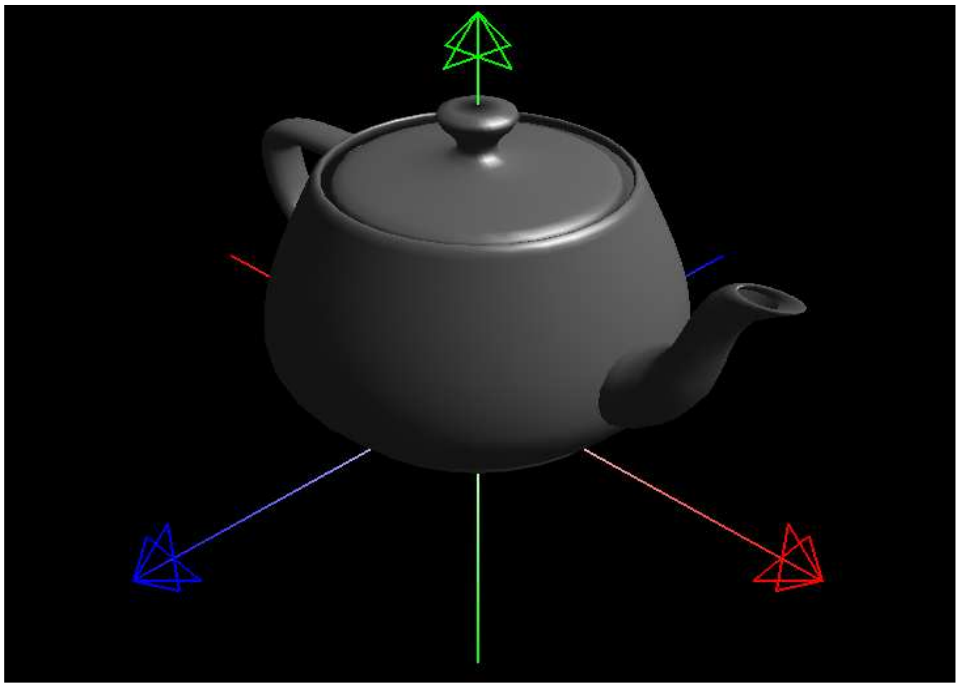
gluNurbsCurve() gluNurbsSurface() - to generate and render -specify control points, knot sequence, order, and/or normals, texture coordinates

Teapot using Bézier Patches: Wireframe Model



```
/* Enable evaluator */
glEnable( GL_MAP2_VERTEX_3 );

glColor3f( 1.0, 1.0, 1.0 );
glRotatef( -90.0, 1.0,  0.0, 0.0 );
glScalef( 0.25, 0.25, 0.25 );

/* Draw wireframe */
for ( k=0;  k< 32; k++ )
    {
     glMap2f( GL_MAP2_VERTEX_3, 0, 1, 3, 4, 0, 1, 12
                    &data[ k ][ 0 ][ 0 ][ 0 ] );

     for ( j = 0; j <= 4; j++ )
        {
         glBegin( GL_LINE_STRIP );
         for ( i = 0; i <= 20; i++ )
             glEvalCoord2f( ( GLfloat ) i / 20.0, ( GLfloat ) j / 4 );
         glEnd( );

         glBegin(GL_LINE_STRIP);
         for ( i = 0; i <=20; i++ )
             glEvalCoord2f( ( GLfloat ) j / 4.0, ( GLfloat ) i / 20 );
        glEnd( );
       }
   }
```

Teapot using Bézier Patches: Solid Model



```
/* Enable evaluators */
glEnable( GL_MAP2_VERTEX_3 );

/* Set up light */
GLfloat light_position[ ] = { 1.0, 1.0, 1.0, 0 };
GLfloat light_ambient[ ] = { 0.2, 0.2, 0.2, 1.0 };
GLfloat light_diffuse[ ] = { 0.6, 0.6, 0.6, 1.0 };
glLightfv( GL_LIGHT0, GL_POSITION, light_position );
glLightfv( GL_LIGHT0, GL_AMBIENT, light_ambient );
glLightfv( GL_LIGHT0, GL_DIFFUSE, light_diffuse ) ;
glEnable( GL_LIGHTING ); glEnable( GL_LIGHT0 );
glEnable( GL_AUTO_NORMAL );

/* Set up material properties */
GLfloat mat_ambient[ ] = {  0.2, 0.2, 0.2, 1.0 };
GLfloat mat_specular[ ] = {  1, 1, 1, 1.0 };
GLfloat mat_diffuse[ ] = { 0.6, 0.6, 0.6, 1.0 };
glMaterialfv( GL_FRONT_AND_BACK, GL_AMBIENT, mat_ambient );
glMaterialfv( GL_FRONT_AND_BACK, GL_DIFFUSE, mat_diffuse );
glMaterialfv( GL_FRONT_AND_BACK, GL_SPECULAR, mat_specular );
glMaterialf( GL_FRONT_AND_BACK, GL_SHININESS, 50.0 );

glRotatef( -90.0, 1.0,  0.0, 0.0 ); glScalef( 0.25, 0.25, 0.25 );

/* Draw solid model */
for ( k=0;  k< 32; k++ ) {
     glMapGrid2f( 8, 0.0, 1.0, 8, 0.0, 1.0 );
     ...
   }
```

Reading Assignment and News

Before the next class please review Chapter 10 and its practice exercises, of the recommended text. Also please see the midterm review questions.

(Recommended Text: Interactive Computer Graphics, by Edward Angel, Dave Shreiner, 6th edition, Addison-Wesley)

Please track Blackboard for the most recent Announcements and Project postings related to this course.

(http://www.cs.utexas.edu/users/bajaj/graphics2012/cs354/)