Today’s class

- Curve fitting
- Evaluators
- Surfaces
Curved lines

- Parametric form: \( P(t) = (x(t), y(t)) \) for \( 0 \leq t \leq 1 \)
- An arbitrary curve can be built up from different sets of parametric functions for different parts of the curve
- Continuity between sections
  - zero order - curves meet
  - first order - tangents are same at meeting point
  - second order - curvatures are same at meeting point
Curved lines (cont.)

- Control points describe a curve in an interactive environment
- If displayed curve passes through the control points the curve is said to **interpolate** the control points
- If displayed curve passes near the control points the curve is said to **approximate** the control points
Blending functions

- Given \( n \) sample points: \((x_0, y_0)\) through \((x_{n-1}, y_{n-1})\)
- Construct \( P(t) \) as a sum of terms, one term for each point:
  - \( x(t) = \sum x_i B_i(t) \)
  - \( y(t) = \sum y_i B_i(t) \)
- The functions \( B_i(t) \) are called **blending functions**
- For each value of \( t \) they determine how much the \( i^{th} \) sample point contributes to the curve
Interpolation

- For each sample point we would like $B_i(t) = 1$ and $B_j(t) = 0$ ($j \neq i$) for some value of $t$.

- This implies the curve $\mathbf{P}(t)$ will pass through each sample point.

- We want to arrange the blending functions such that each sample point has complete control of the curve in succession.
Lagrange interpolation

- Consider the case \( n=4 \)
- Want first point to have complete control at \( t=-1 \), second point at \( t=0 \), third point at \( t=1 \), and fourth point at \( t=2 \)
- Note the \textit{middle} region has \( 0 \leq t \leq 1 \)
- The blending functions are:

\[
B_0(t) = -\frac{1}{6}[t(t-1)(t-2)]
\]
\[
B_1(t) = \frac{1}{2}[(t+1)(t-1)(t-2)]
\]
\[
B_2(t) = -\frac{1}{2}[(t+1)t(t-2)]
\]
\[
B_3(t) = \frac{1}{6}[(t+1)t(t-1)]
\]
Drawing the curve

- Vary \( t \) in small increments between 0 and 1
- This will interpolate the curve between 2\(^{nd}\) and 3\(^{rd}\) control points
- To handle entire curve (more than 4 control points) draw curve between 2\(^{nd}\) and 3\(^{rd}\) points as above, then step up one control point and repeat for middle set
- For very first set of 4 control points will need to evaluate between \( t=-1 \) and \( t=0 \)
- For very last set of 4 control points will need to evaluate between \( t=1 \) and \( t=2 \)
Program for curve fitting

- curves.cpp is available online
- Program does three curve fitting techniques
  - Lagrange interpolation
  - Bézier curves
  - Uniform cubic B-spline
- Examine general program set-up and interpolation routine now
de Casteljau algorithm

- Consider 4 points $p_0$, $p_1$, $p_2$ and $p_3$
- First generation in-betweens:
  
  \[
  p_0^1(t) = (1-t)p_0 + tp_1 \\
  p_1^1(t) = (1-t)p_1 + tp_2 \\
  p_2^1(t) = (1-t)p_2 + tp_3
  \]

- Second generation in-betweens:
  
  \[
  p_0^2(t) = (1-t)p_0^1(t) + tp_1^1(t) \\
  p_1^2(t) = (1-t)p_1^1(t) + tp_2^1(t)
  \]

- Third generation in-betweens:
  
  \[
  p_0^3(t) = (1-t)p_0^2(t) + tp_1^2(t)
  \]
Bézier curves

- By direct substitution and expansion
  \[ p_0^3(t) = (1-t)^3 p_0 + 3(1-t)^2 t p_1 + 3(1-t)t^2 p_2 + t^3 p_3 \]

- \( p_0^3(t) \) is called the Bézier curve for the points \( p_0, p_1, p_2 \) and \( p_3 \)

- Blending functions:
  \[ B_0(t) = (1-t)^3 \]
  \[ B_1(t) = 3t(1-t)^2 \]
  \[ B_2(t) = 3t^2 (1-t) \]
  \[ B_3(t) = t^3 \]

- At \( t=0 \), \( B_0=1 \) and \( B_1=B_2=B_3=0 \)
- At \( t=1 \), \( B_0=B_1=B_2=0 \) and \( B_3=1 \)
- \( B_1 \) has a maximum at \( 1/3 \), \( B_2 \) at \( 2/3 \)
Observations on Bézier curves

- Pass through $p_0$ and $p_3$ only
- Can generate closed curves by specifying first and last control points to be the same
- A single control point can be specified two or more times if you want that point to exert more control of the curve in its region
Complicated Bézier curves

- More than 4 control points
- Piece together using smaller order curves with fewer control points
- Match endpoints for zero-order continuity
- At the endpoints, the tangent to the curve is along the line that connects the endpoint to the adjacent control point; obtain first-order continuity by picking collinear control points (last two of one section with first two of next section - with the endpoints the same)
Revisit program for curve fitting

- Examine Bézier curve fitting routine in `curves.cpp`
Splines

- An $m^{th}$ degree spline function is a piecewise polynomial of degree $m$ that has continuity of derivatives of order $m-1$ at each knot.
- Define the blending functions as shifted versions of the spline function: $B_i(t) = B(t-i)$.
Cubic splines

- Given \( p_0, p_1, \ldots, p_n, 0 \leq t \leq n \)
- Divide into \( n \) subintervals \([t_0, t_1], [t_1, t_2], \ldots, [t_{n-1}, t_n]\)
- Approximate points \( p_i \) by a curve \( P(t) \) which consists of a polynomial of degree 3 in each subinterval
- Points \( t_i \) are called \textbf{knots}
Uniform cubic B-spline

A cubic spline defined on equally spaced knots that is non-zero on the smallest possible interval

\[
P(t) = \begin{cases} 
0 & t \leq -2 \\
\frac{1}{6}(2 + t)^3 & -2 \leq t \leq -1 \\
\frac{1}{6}(2 + t)^3 - \frac{2}{3}(1 + t)^3 & -1 \leq t \leq 0 \\
\frac{2}{3}(1 - t)^3 + \frac{1}{6}(2 - t)^3 & 0 \leq t \leq 1 \\
\frac{1}{6}(2 - t)^3 & 1 \leq t \leq 2 \\
0 & 2 \leq t 
\end{cases}
\]
Observations on cubic B-splines

- At each point at most 4 B-splines are not zero
- At a knot only 3 consecutive points affect the curve, with weights 1/6, 2/3, and 1/6 so curve is approximating
- If a control point is specified 3 times then curve goes through point as $1/6 + 2/3 + 1/6 = 1$
Revisit program for curve fitting

- Examine B-spline fitting routine in `curves.cpp`
Bernstein polynomials

- The Bernstein polynomial of degree $n$ (order $n + 1$) is given by:

$$ B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i} $$

- Recall the Bézier curve function:

$$ p_0^3(t) = (1-t)^3 p_0 + 3(1-t)^2 t p_1 + 3(1-t)t^2 p_2 + t^3 p_3 $$

- Observe that it can be written:

$$ p_0^3(t) = B_0^3(t) p_0 + B_1^3(t) p_1 + B_2^3(t) p_2 + B_3^3(t) p_3 $$
Evaluators

- An OpenGL mechanism for specifying a curve or surface using only the control points
- Use Bernstein polynomials as the blending functions
- Can describe any polynomial or rational polynomial splines or surfaces to any degree, including:
  - B-splines
  - NURBS
  - Bézier curves and surfaces
Defining an evaluator

- `glMap1d (GL_MAP1_VERTEX_3, 0.0, STEPS, 3, 4, &points[j][0]);`
  - one-dimensional evaluator (a curve)
  - specifies x,y,z coordinates (need `glEnable(GL_MAP1_VERTEX_3);`)
  - parameter variables goes 0.0 to STEPS
  - 3 indicates number of values to advance in the data between one control point and the next
  - 4 is the order of the spline (degree + 1)
  - `&points[j][0]` is a pointer to first control point’s data
Evaluating a map

- `glEvalCoord1d (i);` evaluates a map at a given value of the parameter variable
- Value of `i` should be between 0.0 and STEPS
Example program

- evaluator.cpp is an example program using an evaluator
- It does a fit of the same data that the curve fitting program did
Surfaces

- The same curve fitting techniques can be extended to surfaces by applying the methods in two dimensions.
- For example, the Utah teapot is formally defined as a series of 32 bicubic Bézier patches using 306 control points.
Defining a 2-D evaluator

- `glMap2d (GL_MAP2_VERTEX_3, 0.0, 1.0, 12, 4, 0.0, 1.0, 3, 4, coords);`
  - two-dimensional evaluator (a surface)
  - specifies x,y,z coordinates (need `glEnable(GL_MAP2_VERTEX_3);`)
  - `u` parameter variables goes 0.0 to 1.0
  - 12 indicates number of values to advance in the data between one control point and the next in the `u` direction
  - 4 is the order of the spline (degree + 1)
  - The next four parameters handle the same information for the `v` direction
  - `coords` is a pointer to the array of control points
Set up a 2-D mesh

- `glMapGrid2d (STEPS, 0.0, 1.0, STEPS, 0.0, 1.0);`
- Specifies linear grid mappings between the integer grid coordinates and the floating-point evaluation map coordinates
- STEPS specifies the number of partitions in the u direction
- 0.0, 1.0 specify the u values for the first and last grid values
- The next 3 parameters repeat the information for the v direction
Compute the 2-D mesh

- `glEvalMesh2 (GL_FILL, 0, STEPS, 0, STEPS);`

- **First parameter** describes the type of mesh:
  - `GL_POINT` for points
  - `GL_LINE` for lines
  - `GL_FILL` for polygons

- **Next parameters** specify first and last integer grid values for the two directions
The `teapot.cpp` program shows the use of a two-dimensional OpenGL evaluator to produce the teapot.