Splines

- Cubic polynomial form for curve
  \[ p(u) = c_0 + c_1 u + c_2 u^2 + c_3 u^3 = \sum_{i=0}^{3} c_i u^i \]
- Each \( c_i \) is a column vector \([c_{ix}, c_{iy}, c_{iz}]^T\)
- Solve for \( c_i \) given control points
- Interpolation: 4 points
- Hermite curves: 2 endpoints, 2 tangents
- Bezier curves: 2 endpoints, 2 tangent points

B-Splines

- Use 4 points, but approximate only middle two
- Draw curve with overlapping segments
  0-1-2-3, 1-2-3-4, 2-3-4-5, 3-4-5-6, etc.
- Curve may miss all control points
- Smoother at joint points

Cubic B-Splines

- Need \( m+2 \) control points for \( m \) cubic segments
- Computationally 3 times more expensive
- \( C^2 \) continuous at each interior point
- Derive as follows:
  - Consider two overlapping segments
  - Enforce \( C^0 \) and \( C^1 \) continuity
  - Employ symmetry
  - \( C^2 \) continuity follows

Deriving B-Splines

- Consider points
  - \( p_0, p_{-1}, p_1, p_2 \)
  - \( p(0) \approx p_{-1}, p(1) \approx p_1 \)
  - \( p_{-2}, p_{-1}, p_1, p_2 \)
  - \( q(0) \approx p_{-2}, q(1) \approx p_1 \)
- Condition 1: \( p(0) = q(1) \)
  - Symmetry: \( p(0) = q(1) = 1/6 (p_{-2} + 4 p_{-1} + p_1) \)
- Condition 2: \( p'(0) = q'(1) \)
  - Geometry: \( p'(0) = q'(1) = 1/2 ((p_0 - p_{-1}) + (p_{-1} - p_2)) \)
  - Symmetry: \( p'(0) = q'(1) = 1/2 (p_0 - p_{-2}) \)
B-Spline Geometry Matrix

- Conditions at \( u = 0 \)
  - \( p(0) = c_0 = 1/6 \left(p_{i-2} + 4p_{i-1} + p_i\right) \)
  - \( p'(0) = c_1 = 1/2 \left(p_i - p_{i-2}\right) \)
- Conditions at \( u = 1 \)
  - \( p(1) = c_0 + c_1 + c_2 + c_3 = 1/6 \left(p_{i-1} + 4p_i + p_{i+1}\right) \)
  - \( p'(1) = c_1 + 2c_2 + 3c_3 = 1/2 \left(p_{i+1} - p_{i-1}\right) \)

\[
\begin{bmatrix}
  c_0 \\
  c_1 \\
  c_2 \\
  c_3 \\
\end{bmatrix} = M_S \begin{bmatrix}
  p_{i-2} \\
  p_{i-1} \\
  p_i \\
  p_{i+1} \\
\end{bmatrix}, \quad M_S = \frac{1}{6} \begin{bmatrix}
  1 & 4 & 1 & 0 \\
  -3 & 0 & 3 & 0 \\
  3 & -6 & 3 & 0 \\
  -1 & 3 & -3 & 1 \\
\end{bmatrix}
\]

Blending Functions

- Calculate cubic blending polynomials

\[
b(u) = M_S u = \frac{1}{6} \begin{bmatrix}
  (1 - u)^3 \\
  -4 + 6u^2 - 3u^3 \\
  1 + 3u + 3u^2 - 3u^3 \\
  u^3 \\
\end{bmatrix}
\]

- Note symmetries

Convex Hull

- For \( 0 \leq u \leq 1 \), have \( 0 \leq b_3(u) \leq 1 \)
- Recall:
  \( p(u) = \sum_{i} b_i(u)p_i = \sum \sum b_i(u)b_k(v)p_{ik} \)
- So each point \( p(u) \) lies in convex hull of \( p_k \)

Spline Basis Functions

- Total contribution \( B_i(u)p_i \) of \( p_i \) is given by

\[
B_i(u) = \begin{cases}
  0 & u < i - 2 \\
  b_0(u + 2) & i - 2 \leq u < i - 1 \\
  b_1(u + 1) & i - 1 \leq u < i \leq i \\
  b_2(u) & i \leq u < i + 1 \\
  b_3(u - 1) & i + 1 \leq u < i + 2 \\
  0 & i + 2 \leq u
\end{cases}
\]

Spline Surface

- As for Bezier patches, use 16 control points
- Start with blending functions

\[
p(u, v) = \sum \sum b_i(u)b_k(v)p_{ik}
\]

- Need 9 times as many splines as for Bezier

Assessment: Cubic B-Splines

- More expensive than Bezier curves or patches
- Smoother at join points
- Local control
  - How far away does a point change propagate?
- Contained in convex hull of control points
- Preserved under affine transformation
- How to deal with endpoints?
  - Closed curves (uniform periodic B-splines)
  - Non-uniform B-Splines (multiplicities of knots)
General B-Splines

- Generalize from cubic to arbitrary order
- Generalize to different basis functions
- Read: [Angel, Ch 10.8]
- Knot sequence $u_{\text{min}} = u_0 \leq ... \leq u_n = u_{\text{max}}$
- Repeated points have higher "gravity"
- Multiplicity 4 means point must be interpolated
- $\{0, 0, 0, 1, 2, ..., n-1, n, n, n, n\}$ solves boundary problem

Nonuniform Rational B-Splines (NURBS)

- Exploit homogeneous coordinates
  
  $p_i = \begin{bmatrix} x_i \\
  y_i \\
  z_i \\
  1 
\end{bmatrix} \simeq w_i \begin{bmatrix} x_i \\
  y_i \\
  z_i \\
  1 
\end{bmatrix} = q_i$

- Use perspective division to renormalize
  
  $p(u) = \frac{\sum_{i=0}^{n} B_i(u)w_i p_i}{\sum_{i=0}^{n} B_i(u)w_i}$

- Each component of $p(u)$ is rational function of $u$
- Points not necessarily uniform (NURBS)

NURBS Assessment

- Convex-hull and continuity props. of B-splines
- Preserved under perspective transformations
  - Curve with transformed points = transformed curve
- Widely used (including OpenGL)

Outline

- Cubic B-Splines
- Nonuniform Rational B-Splines (NURBS)
- Rendering by Subdivision
- Curves and Surfaces in OpenGL

Rendering by Subdivision

- Divide the curve into smaller subpieces
- Stop when "flat" or at fixed depth
- How do we calculate the sub-curves?
  - Bezier curves and surfaces: easy (next)
  - Other curves: convert to Bezier!

Subdividing Bezier Curves

- Given Bezier curve by $p_0, p_1, p_2, p_3$
- Find $l_0, l_1, l_2, l_3$ and $r_0, r_1, r_2, r_3$
- Subcurves should stay the same!
Construction of Bezier Subdivision

- Use algebraic reasoning

\[ l(0) = l_0 = p_0 \]
\[ l(1) = l_3 = p(1/2) = 1/8(p_0 + 3p_1 + 3p_2 + p_3) \]
\[ l'(0) = 3(l_1 - l_0) = p'(0) = 3/2 (p_1 - p_0) \]
\[ l'(1) = 3(l_3 - l_2) = p'(1/2) = 3/8(-p_0 + p_2 + p_3) \]
- Note parameter substitution \( v = 2u \) so \( dv = 2du \)

Geometric Bezier Subdivision

- Can also calculate geometrically

\[ l_1 = \frac{1}{2}(p_0 + p_1), r_2 = \frac{1}{2} (p_2 + p_3) \]
\[ l_2 = \frac{1}{2} (l_1 + \frac{1}{2} (p_1 + p_2)), r_1 = \frac{1}{2} (r_2 + \frac{1}{2}(p_1 + p_2)) \]
\[ l_3 = r_0 = \frac{1}{2} (l_2 + r_1), l_0 = p_0, r_3 = p_3 \]

Recall: Bezier Curves

- Recall \( u^T = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \)
- Express \( p(u) = c_0 + c_1u + c_2u^2 + c_3u^3 \)

\[ = u^T \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = u^T M_B \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} \]

\[ M_B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & -1 \end{bmatrix} \]

Subdividing Other Curves

- Calculations more complex
- Trick: transform control points to obtain identical curve as Bezier curve!
- Then subdivide the resulting Bezier curve
- Bezier: \( p(u) = u^T M_B p \)
- Other curve: \( p(u) = u^T M q, M \) geometry matrix
- Solve: \( q = M^{-1} M_B p \)

Example Conversion

- From cubic B-splines to Bezier:

\[ M_B^{-1} M_S = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & 0 \\ 0 & 4 & 2 & 0 \\ 0 & 2 & 4 & 0 \\ 0 & 1 & 4 & 1 \end{bmatrix} \]

- Calculate Bezier points \( p \) from \( q \)
- Subdivide as Bezier curve

Subdivision of Bezier Surfaces

- Slightly more complicated
- Need to calculate interior point
- Cracks may show with uneven subdivision
- See [Angel, Ch 10.9.4]
Outline
- Cubic B-Splines
- Nonuniform Rational B-Splines (NURBS)
- Rendering by Subdivision
- Curves and Surfaces in OpenGL

Curves and Surface in OpenGL
- Central mechanism is evaluator
- Defined by array of control points
- Evaluate coordinates at u (or u and v) to generate vertex
- Define Bezier curve: type = GL_MAP_VERTEX_3
  glMap1f(type, u0, u1, stride, order, point_array)
- Enable evaluator
glEnable(type)
- Evaluate Bezier curve
glEvalCoord1f(u)

Example: Drawing a Bezier Curve
- 4 control points
  GLfloat ctrlpoints[4][3] = {
    {-4.0, -4.0, 0.0}, {-2.0, 4.0, 0.0},
    {2.0, -4.0, 0.0}, {4.0, 4.0, 0.0}};
- Initialize
  void init()
  { ...
    glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4,
        &ctrlpoints[0][0]);
    glEnable(GL_MAP1_VERTEX_3);
  }

Drawing the Control Points
- To illustrate Bezier curve
  void display()
  { ...
    glPointSize(5.0);
    glColor3f(1.0, 1.0, 0.0);
    glBegin(GL_POINTS);
    for (i = 0; i < 4; i++)
      glVertex3fv(&ctrlpoints[i][0]);
    glEnd();
    glFlush();
  }

Evaluating Coordinates
- Use a fixed number of points, num_points
  void display()
  { ...
    glBegin(GL_LINE_STRIP);
    for (i = 0; i <= num_points; i++)
      glEvalCoord1f((GLfloat)i/(GLfloat)num_points);
    glEnd();
  }

Resulting Images
n = 5
n = 20
Bezier Surfaces
• Create evaluator in two parameters u and v
  \[
glMap2f(GL\_MAP2\_VERTEX\_3, \\
u_u, u_v, ustride, uorder, \\
v_v, vstride, vorder, point\_array);
\]
• Enable, also automatic calculation of normal
  \[
glEnable(GL\_MAP2\_VERTEX\_3); \\
glEnable(GL\_AUTO\_NORMAL);
\]
• Evaluate at parameters u and v
  \[
glEvalCoord2f(u, v);
\]

Grids
• Convenience for uniform evaluators
• Define grid (nu = number of u division)
  \[
glMapGrid2f(nu, u_0, u_1, nv, v_0, v_1);
\]
• Evaluate grid
  \[
glEvalMesh2(mode, i_0, i_1, k_0, k_1);
\]
  \(mode = GL\_POINT, GL\_LINE, or GL\_FILL\)
  \(i and k define subrange\)

Example: Bezier Surface Patch
• Use 16 control points
  \[
GLfloat ctrlpoints[4][4][3] = {...};
\]
• Initialize 2-dimensional evaluator
  \[
void init(void) \\
{ ...
  glMap2f(GL\_MAP2\_VERTEX\_3, 0, 1, 3, 4, \\
         0, 1, 12, 4, &ctrlpoints[0][0][0]);
  glEnable(GL\_MAP2\_VERTEX\_3);
  glEnable(GL\_AUTO\_NORMAL);
  glMapGrid2f(20, 0.0, 1.0, 20, 0.0, 1.0);
  }
\]

Evaluating the Grid
• Use full range
  \[
void display(void) \\
{ ... 
  glPushMatrix();
  glRotatef(85.0, 1.0, 1.0, 1.0);
  glEvalMesh2(GL\_FILL, 0, 20, 0, 20);
  glPopMatrix();
  glFlush();
  }
\]

NURBS Functions
• Higher-level interface
• Implemented in GLU using evaluators
• Create a NURBS renderer
  \[
theNurb = gluNewNurbsRenderer();
\]
• Set NURBS properties
  \[
  gluNurbsProperty(theNurb, GLU\_DISPLAY\_MODE, GLU\_FILL);
  gluNurbsCallback(theNurb, GLU\_ERROR, nurbsError);
\]
Displaying NURBS Surfaces

• Specify knot arrays for splines
  
  ```c
  GLfloat knots[8] = {0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0};
  gluBeginSurface(theNurb);
  gluNurbsSurface(theNurb,
      8, knots, 8, knots,
      4 * 3, 3, &ctlpoints[0][0][0],
      4, 4, GL_MAP2_VERTEX_3);
  gluEndSurface(theNurb);
  ```

• For more see [Red Book, Ch. 12]

Summary

• Cubic B-Splines
• Nonuniform Rational B-Splines (NURBS)
• Rendering by Subdivision
• Curves and Surfaces in OpenGL

Reminders

• Assignment 3 due tonight
• Assignment 4 out some time today
• Midterm will cover curves and surfaces
• Next Tuesday: Textures?
• Next Thursday: John Ketchpaw