Conic Sections via NURBS

- Obtained via projection of the 3D parabola onto a plane
- Note:
  - 3D Case: rational curve is a 4D object
  - 2D Case: rational curve is a 3D object
  - assign \( w \) to each control point


Conic Sections via NURBS:
A Circular Arc

- The two sides of the control polygon are of equal length
- The chord connecting the first and last control points meets each leg at an angle equal to half the angular extent of the desired arc (for instance, 30° for a 60° arc)
- The weight of the inner control point is equal to the cosine of \( \theta \)
- Knot vector is \( \{0.0, 0.0, 0.0, 1.0, 1.0, 1.0\} \)
Conic Sections via NURBS

Example:
4 arcs of 90°

Knot Insertion

• Issue: More control points mean more control
• How do we add more points and keep same curve?

Knot Insertion

• Basic Approach
  – Decide where we’d like to tweak the curve
  – Add a new knot
  – Find affected d-1 control points
  – Replace it with d new control points

Example:
New knot at u=2.6

Knot Insertion Algorithm

• Create new control point
  \[ Q_j = (\square_j) P_{j-1} + \square_j P_j \]
• Where \( \square \) is defined as
  \[ \square_j = \frac{t \square u_j}{u_j + d \square u_j} \]

Properties of Knot Insertion

• Increasing the multiplicity of a knot decreases the number of non-zero basis functions at this knot
• At a knot of multiplicity d, there will be only one non-zero basis function
• Corresponding point on the curve \( p(u) \) is affected by exactly one control point \( p_i \)
  • In fact \( p(u) = p_i \)
The de Boor Algorithm

- Generalization of de Casteljau's algorithm
- It provides a fast and numerically stable way for finding a point on a B-spline curve
- Observation: if a knot \( u \) is inserted \( d \) times to a B-spline, then \( p(u) \) is the point on the curve.
- Idea: We just simply insert \( u \) \( d \) times and the last point is \( p(u) \)!

See http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/B-spline/de-Boor.html
De Boor’s Algorithm

If \( u \) lies in \([u_i, u_{i+1})\) and \( u = u_i \), let \( h = d \)
If \( u = u_i \) and \( u_i \) is a knot of multiplicity \( s \), let \( h = d-s \)
Copy the affected control points \( p_{i+k-1}, \ldots, p_{i+k+d-1} \) to a new array and rename them as \( p_{i+k-d-1}, \ldots, p_{i+k+d-1} \)

for \( r = 1 \) to \( h \) do
for \( i = k-d+r \) to \( k-s \) do

\[
\begin{align*}
\alpha_i &= (u - u_i) / (u_{i+r} - u_i) \\
\beta_i &= 1 - \alpha_i \\
p_{i,r} &= (1 - \beta_i) p_{i+r} + \beta_i p_{i+r-1} 
\end{align*}
\]

\( p_{k+d,d} \) is the point \( p(u) \).

Example of De Boor’s Algorithm

Degree 3 B-spline curve (i.e., \( d = 3 \))
Defined by seven control points \( p_0, \ldots, p_6 \)
And knot vector:

\[
\begin{align*}
u &= 0.4 \\
u_0 &= 0.0 \quad u_1 = 0.1 \quad u_2 = 0.3 \quad u_3 = 0.5 \quad u_4 = 0.6 \quad u_5 = 1.0 
\end{align*}
\]

\[
\begin{align*}
\alpha_1 &= (u - u_1) / (u_2 - u_1) = 0.2 \\
\alpha_2 &= (u_3 - u) / (u_3 - u_2) = 0.53 \\
\alpha_3 &= (u - u_3) / (u_4 - u_3) = 0.4 \\
p_1 &= (1 - \beta_1) p_1 + \beta_1 p_0 = 0.3p_1 + 0.7p_0 \\
p_2 &= (1 - \beta_2) p_2 + \beta_2 p_1 = 0.47p_2 + 0.53p_1 \\
p_3 &= (1 - \beta_3) p_3 + \beta_3 p_2 = 0.3p_3 + 0.7p_2 \\
p_4 &= (1 - \beta_4) p_4 + \beta_4 p_3 = 0.3p_4 + 0.7p_3 \\
p_5 &= (1 - \beta_5) p_5 + \beta_5 p_4 = 0.3p_5 + 0.7p_4 \\
p_6 &= (1 - \beta_6) p_6 + \beta_6 p_5 = 0.3p_6 + 0.7p_5 \\
p_{3.2} &= \ldots \\
\end{align*}
\]

De Boor’s: Curves

De Boor’s Algorithm (cont)

for \( u := 0 \) to \( u_{max} \) do

for \( r := 1 \) to \( h \) do
for \( i := k-p+r \) to \( k-s \) do

\[
\begin{align*}
\alpha_i &= (u - u_i) / (u_{i+r} - u_i) \\
\beta_i &= 1 - \alpha_i \\
p_{i,r} &= (1 - \beta_i) p_{i+r} + \beta_i p_{i+r-1} 
\end{align*}
\]

\( p_{k+d,d} \) is the point \( p(u) \).

Similar but Different

De Casteljau’s:
- Dividing points are computed with a pair of numbers \((1 - u)\) and \( u \) that never change
- Can be used for curve subdivision
- Uses all control points

De Boor’s:
- These pairs of numbers are different and depend on the column number and control point number
- Intermediate control points are not sufficient \(d+1\) affected control points are involved in the computation

Oslo Algorithm

- A subdivision algorithm for B-splines, the basic idea:
- Take the curve with \( m+1 \) control points \( P_0, \ldots, P_m \)
- Insert a knot in any point \((0.3\text{ maybe?})\)
- As a result you will have 2 new points \( P_0' \) and \( P_m' \)
- Take curves with \( m+1 \) control points \( P_0', \ldots, P_r, \ldots, P_m' \)
- Apply procedure recursively on each part
Oslo Algorithm

Barycentric Coordinates

• By Ceva’s Theorem:
  – For any point K inside the triangle ABC
  – Consider the existence of masses \( w_A, w_B, \) and \( w_C \), placed at the vertices of the triangle
  – Their center of gravity (barycenter) will coincide with the point K.
• August Ferdinand Moebius (1790-1868) defined \( w_A, w_B, \) and \( w_C \) as the barycentric coordinates of K
  \[ K = w_A A + w_B B + w_C C \]

Properties of Barycentric Coordinates

• Not unique
• Can be generalized to negative masses
• Can be made unique by setting \( w_A + w_B + w_C = 1 \)
• \( w_A = 0 \) for points on BC
• \( w_B = 0 \) for points on AC
• \( w_C = 0 \) on AB

Calculating the Weights

• Given vertices A, B, C and Centroid K
• What are the weights, \( w_A, w_B, w_C \)?
  \[ x_K = w_A x_A + w_B x_B + w_C x_C \]
  \[ y_K = w_A y_A + w_B y_B + w_C y_C \]
• Substitute \( w_C = 1 - w_A - w_B \)
  \[ x_K = w_A x_A + w_B x_B + (1 - w_A - w_B) x_C \]
  \[ y_K = w_A y_A + w_B y_B + (1 - w_A - w_B) y_C \]

Calculating Weights (cont.)

• Solve for \( w_A \) and \( w_B \)
  \[ w_A = \frac{(x_B - x_C)(y_C - y_K) - (y_B - y_C)(x_C - x_K)}{(x_A - x_C)(y_B - y_C) - (y_A - y_C)(x_B - x_C)} \]
  \[ w_B = \frac{(x_A - x_C)(y_C - y_K) - (y_A - y_C)(x_C - x_K)}{(x_B - x_C)(y_A - y_C) - (y_B - y_C)(x_A - x_C)} \]
• \( w_C = 1 - w_A - w_B \)
Onto…

• Bézier Surfaces
• B-spline Surfaces
• NURBS Surfaces
• Faceting, Subdivision, Tessellation
• 3D Objects

Programming assignment 3

• Input PostScript-like file containing polygons
• Output B/W XPM
• Implement viewports
• Use Sutherland-Hodgman intersection for polygon clipping
• Implement scanline polygon filling. (You can not use flood filling algorithms)