Outline

- Conic Sections via NURBS
- Knot insertion algorithm
- The de Boor’s algorithm
  - for B-Splines
  - for NURBS
- Oslo Algorithm
- Barycentric Coordinates
- Discussion of homework #3

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

- Define the curve with three control points
- Weights of first/last control points are 1
- For center control point
  - \( w < 1 \) gives an ellipse
  - \( w > 1 \) gives a hyperbola
  - \( w = 1 \) gives a parabola
  - Knot vector is \([0.0, 0.0, 0.0, 1.0, 1.0, 1.0] \)


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 \( \frac{\theta}{2} \) 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: A Circle

- What if we need an arc of \( >180° \)?
- Idea:
  - Use multiple 90° or 120° arcs
  - stitch them together with knots
- Example:
  - 3 arcs of 120°
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
  \[ Q_j = (1-\alpha_j)P_{j-1} + \alpha_jP_j \]
• Where \( \alpha \) is defined as
  \[ \alpha_j = \frac{t - u_j}{u_{j+d} - u_j} \]

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 \( p(u) \) is \( p_i \)
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 \([u_0, u_{n+1})\) and \( u \neq u_i \), let \( h = d \)
If \( u = u_i \) and \( u_{i+1} \) is a knot of multiplicity \( s \), let \( h = d - s \)
Copy the affected control points \( \mathbf{p}_{k-s,0}^{a}, \mathbf{p}_{k-s+1,0}^{a}, \ldots, \mathbf{p}_{k-s+d-1,0}^{a} \)
to a new array and rename them as \( \mathbf{p}_{k-s,0}^{a}, \mathbf{p}_{k-s+1,0}^{a}, \ldots, \mathbf{p}_{k-s+d-1,0}^{a} \)

for \( r := 1 \) to \( h \) do

for \( j = k-d+r \) to \( k-s \) do

\[
\begin{align*}
    a_j &= (u - u_j) / (u_{j+1} - u_j) \\
    \mathbf{p}_{j} &= (1 - a_j) \mathbf{p}_{j+1} + a_j \mathbf{p}_{j+2}
\end{align*}
\]

\( \mathbf{p}_{k-d+a} \) is the point \( \mathbf{p}(u) \).

De Boor’s Algorithm (cont)

for \( u := 0 \) to \( u_{n+1} \) do

for \( r := 1 \) to \( h \) do

for \( j := k-p+r \) to \( k-s \) do

\[
\begin{align*}
    a_j &= (u - u_j) / (u_{j+1} - u_j) \\
    \mathbf{p}_{j} &= (1 - a_j) \mathbf{p}_{j+1} + a_j \mathbf{p}_{j+2}
\end{align*}
\]

\( \mathbf{p}_{k-d+a} \) is the point \( \mathbf{p}(u) \).

Example of de Boor’s Algorithm

Degree 3 B-spline curve (i.e., \( d = 3 \))
Defined by seven control points \( \mathbf{p}_0, \ldots, \mathbf{p}_6 \)
And knot vector:
\( u = 0.4 \)

<table>
<thead>
<tr>
<th>( u )</th>
<th>( u_0 )</th>
<th>( u_1 )</th>
<th>( u_2 )</th>
<th>( u_3 )</th>
<th>( u_4 )</th>
<th>( u_5 )</th>
<th>( u_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>0.3</td>
<td>0.5</td>
<td>0.7</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Intermediate control points are involved in

\( \mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_5, \mathbf{p}_6 \)

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

De Boor’s: Curves

Oslo Algorithm

- A subdivision algorithm for B-splines, the basic idea:
- Take the curve with \( m+1 \) control points \( \mathbf{P}_0 \) to \( \mathbf{P}_m \)
- Insert a knot in any point \((0.5 \text{ maybe?})\)
- As a result you will have 2 new points \( \mathbf{P}_{i,2} \) and \( \mathbf{P}_{i,0} \)
- Take curves with \( m+1 \) control points \( \mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_{m-1} \)
- Apply procedure recursively on each part
**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 Möbius (1790-1868) defined (1827) $w_A$, $w_B$, 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 $w_B$

$$w_A = \frac{(x_B - x_C)(y_C - y_K) - (x_C - x_K)(y_B - y_C)}{(x_A - x_C)(y_B - y_C) - (x_B - x_C)(y_A - y_C)}$$

$$w_B = \frac{(x_A - x_C)(y_C - y_K) - (x_C - x_K)(y_A - y_C)}{(x_B - x_C)(y_A - y_C) - (x_A - x_C)(y_B - y_C)}$$

- $w_C = 1 - w_A - w_B$
Programming Assignment 2

• Process command-line arguments
• Read in 3D curve points and tangents
• Compute Bezier control points for curves defined by each two curve points
• Use HW1 code to compute points on each Bezier curve
• Each Bezier curve should be a polyline
• Output points by printing them to the console as an IndexedLineSet with multiple polylines, and control points as spheres in Open Inventor format