Modelling Curves 2 - Bézier Splines
Bézier Splines

- Bézier Curves were discovered by Pierre Bézier.
- Approximating the shape of a control polygon.
- Mathematically: \[ P(t) = \sum_{i=0}^{n} B_i J_{n,i}(t) \] with \(0 \leq t \leq 1\)
- Where \(J_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i}\) and is called the Bernstein basis.
- \(\binom{n}{i} = \frac{n!}{i! (n-i)!}\)
Bézier Splines

\[ P(t) = \sum_{i=0}^{n} B_i J_{n,i}(t) \quad \text{with} \quad 0 \leq t \leq 1 \]

\[ J_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i} \quad \binom{n}{i} = \frac{n!}{i!(n-i)!} \]

\[ \sum_{i=0}^{n} J_{n,i}(t) = 1 \]

Partition of Unity

\[ \binom{n}{0} = \binom{n}{n} = 1 \]
\[ \binom{n}{i} = 0 \quad \text{for} \quad i \notin [0, n] \]
\[ \binom{n}{i} > 0 \quad \text{for} \quad i \in [0, n] \]

Positivity

Control polygon

Curve
Bézier Splines

- Cubic Bézier Splines

\[ P(t) = \sum_{i=0}^{3} B_i J_{3,i}(t) \quad \text{with } 0 \leq t \leq 1 \]

\[
\begin{align*}
J_{3,0}(t) &= (1 - t)^3 \\
J_{3,1}(t) &= 3t (1 - t)^2 \\
J_{3,2}(t) &= 3t^2 (1 - t) \\
J_{3,3}(t) &= t^3
\end{align*}
\]
Bézier Splines

• Cubic Bézier Splines

\[
P(t) = \begin{bmatrix}
(1 - t)^3 & 3t(1 - t)^2 & 3t^2(1 - t) & t^3
\end{bmatrix}
\begin{bmatrix}
B_0 \\
B_1 \\
B_2 \\
B_3
\end{bmatrix}
\]

\[
= \begin{bmatrix}
t^3 & t^2 & t & 1
\end{bmatrix}
\begin{bmatrix}
-1 & 3 & -3 & 1 \\
3 & -6 & 3 & 0 \\
-3 & 3 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
B_0 \\
B_1 \\
B_2 \\
B_3
\end{bmatrix}
\]
Bézier Splines

- Cubic Bézier Splines
Bézier Splines

- The basis functions are real.
- Degree of the polynomial defining the Bézier curve is one less than the number of defining control polygon points.
- The curve generally follows the shape of the control polygon.
- Endpoint Interpolation: The first and last point of the curve are coincident with the first and last point of the control polygon.

\[ \text{at } t = 0 \]
\[ i = 0, \quad J_{n,0} = \frac{n!}{n! 0!} (0)^0 (1 - 0)^{n-0} = 1 \]
\[ i \neq 0, \quad J_{n,i} = \frac{n!}{(n-i)! i!} (0)^i (1 - 0)^{n-i} = 0 \]
\[ \Rightarrow P(0) = \sum_{i=0}^{n} B_0 J_{n,i} = B_0 \]

\[ \text{at } t = 1 \]
\[ i = n, \quad J_{n,n} = \frac{n!}{0! n!} (1)^n (1 - 1)^{n-n} = 1 \]
\[ i \neq n, \quad J_{n,i} = \frac{n!}{(n-i)! i!} (1)^i (1 - 1)^{n-i} = 0 \]
\[ \Rightarrow P(1) = \sum_{i=0}^{n} B_n J_{n,i} = B_n \]
Bézier Splines

- Affine Invariance: Applying an affine transform to the curve is equivalent to applying the transformation to the control points.

\[
M \mathbf{x} = A \mathbf{x} + l
\]

\[
\Rightarrow M \mathbf{P}(t) = A \sum_{i=0}^{n} B_i \mathbf{J}_{n,i}(t) + l
\]

\[
= A \sum_{i=0}^{n} B_i \mathbf{J}_{n,i}(t) + l \sum_{i=0}^{n} \mathbf{J}_{n,i}(t)
\]

\[
= \sum_{i=0}^{n} (A B_i + l) \mathbf{J}_{n,i}(t) = \sum_{i=0}^{n} (M B_i) \mathbf{J}_{n,i}(t)
\]
Bézier Splines

- Convex Hull: The curve lies inside the convex hull of the control points.
- Given a set of points \( X = \{x_0, x_1, \ldots, x_n\} \) the convex hull of \( X \) is given by the set of points:
  \[
  CH(X) = \{a_0 x_0 + \ldots + a_n x_n \mid \sum_{i=0}^{n} a_i = 1, a_i \geq 0, a_i \in \mathbb{R}, x \in X\}
  \]
- Every point on the curve is of the form:
  \[
P(t) = \sum_{i=0}^{n} B_i J_{n,i}(t) \quad \text{with} \ 0 \leq t \leq 1
  \]
  and \( \sum_{i=0}^{n} J_{n,i}(t) = 1, J_{n,i}(t) \geq 0, J_{n,i}(t) \in \mathbb{R} \)
- So every point lies in the convex hull of \( B_i \)
Bézier Splines

- Symmetry: \( J_{n,i}(t) = J_{n,n-i}(1-t) \)

- The curve \( P(t) \) formed by the control points \( B_0, \ldots, B_n \) is the same as the curve \( P(1-t) \) formed by the control points \( B_n, \ldots, B_0 \).

- Parameter Domain Transformation:
  \[
  t = \frac{u - a}{b - a} \\
  \Rightarrow P(t) = \sum_{i=0}^{n} B_i J_{n,i}(t) = \sum_{i=0}^{n} B_i J_{n,i}\left(\frac{u-a}{b-a}\right)
  \]
Bézier Splines

- Variation Diminishing Property: The number of intersections of a given straight line with a planar Bézier curve is less than or equal to the number of intersections of that line with the control polygon.

http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/Bezier/Bezier-construct.html
Bézier Splines

• Tangent Vectors:

\[
\frac{dP(t)}{dt} = \sum_{i=0}^{n} \frac{d}{dt} B_i J_{n,i}(t)
\]

\[
\frac{dJ_{n,i}(t)}{dt} = \frac{d}{dt} \binom{n}{i} t^i (1-t)^{n-i} = \binom{n}{i} it^{i-1}(1-t)^{n-i} - \binom{n}{i} t^i (n-i)(1-t)^{n-i-1}
\]

\[
= n \binom{n-1}{i-1} t^{i-1}(1-t)^{n-i} - n \binom{n-1}{i} t^i (1-t)^{n-i-1}
\]

\[
= n(J_{n-1,i-1}(t) - J_{n-1,i}(t))
\]

\[
\Rightarrow \frac{dP(t)}{dt} = n \sum_{i=0}^{n} B_i (J_{n-1,i-1}(t) - J_{n-1,i}(t)) = n \sum_{i=1}^{n} B_i J_{n-1,i-1}(t) - n \sum_{i=0}^{n-1} B_i J_{n-1,i}(t)
\]

\[
= n \sum_{i=0}^{n} B_{i+1} J_{n-1,i}(t) - n \sum_{i=0}^{n-1} B_i J_{n-1,i}(t) = n \sum_{i=0}^{n-1} (B_{i+1} - B_i) J_{n-1,i}(t)
\]
Bézier Splines

- Tangent Vectors:

\[ P'(0) = n(B_1 - B_0) J_{n-1,0} = n(B_1 - B_0) \]
\[ P'(1) = n(B_n - B_{n-1}) J_{n-1,n-1} = n(B_n - B_{n-1}) \]

- i.e., tangent vectors at the ends of curve have the same direction as the first and last spans of the control polygon.

- Continuity: \( P(t) \) of degree \( n \), defined by control vertices \( B_i \)

\( Q(s) \) of degree \( m \), defined by control vertices \( C_i \)

For \( C^1 \) continuity: \( P'(1) = Q'(0) \)

\[ C_1 - C_0 = \frac{n}{m} (B_n - B_{n-1}) \]

\[ C_1 = \frac{n}{m} (B_n - B_{n-1}) + B_n \text{ because } C_0 = B_n \]

i.e., \( B_{n-1}, B_n = C_0, C_1 \) have to be collinear for the tangents to have the same direction.
Bézier Splines

- Control:
  - How does the shape of the curve change if we move one control point?
  - Each control point is associated to a basis function.
  - The basis functions effect the shape of the curve over a range of parameter values where the basis function is non-zero. In case of the Bernstein basis, this is the entire parameter range [0,1].

Bézier Splines

• Points on the curve: De Casteljau's Algorithm

\[ B_{i,j} = (1-u)B_{i-1,j} + uB_{i-1,j+1} \text{ where } 1 \leq i \leq n, \ 0 \leq j \leq n-i \]

http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/Bezier/de-casteljau.html
Bézier Splines

- Points on the curve: De Casteljau's Algorithm

```
decasteljau(int i, int j)
{
    if (i == 0) then return \( B_{0,j} \)
    else
        return (1-u)* decasteljau(i-1, j) + u* decasteljau(i-1, j+1)
}
```

- Is this a good way to implement the algorithm?
- Why bother with the algorithm at all?
- Subdivision and Degree Elevation.