Hermite and Catmull-Rom Curves

Hermite Curve

- 3D curve of polynomial bases
- Geometrically defined by position and tangents at end points
- No convex hull guarantees
- Supports tangent-continuous (C¹) composite curves

Algebraic Representation

- All of these curves are just parametric algebraic polynomials expressed in different bases
- Parametric cubic curve (in $\mathbb{R}^3$)
  \[ P(u) = au^3 + bu^2 + cu + d \]
  \[ P'(u) = 3au^2 + 2bu + c \]
- First derivative of curve
  \[ x = a u^3 + bu^2 + cu + d, \]
  \[ y = a u^3 + bu^2 + cu + d, \]
  \[ z = a u^3 + bu^2 + cu + d. \]
- Solving for the coefficients:
  \[ P(0) = d, \]
  \[ P'(0) = c, \]
  \[ P(1) = a + b + c + d, \]
  \[ P'(1) = 3a + 2b + c. \]

Hermite Curves

- 12 degrees of freedom (4 3-d vector constraints)
- Specify endpoints and tangent vectors at endpoints
  \[ P(0) = d, \]
  \[ P(1) = a + b + c + d, \]
  \[ P'(0) = c, \]
  \[ P'(1) = 3a + 2b + c. \]
- Solving for the coefficients:
  \[ a = 2p(0) - 2p(1) + p''(0) + p''(1), \]
  \[ b = -3p(0) + 3p(1) - 2p''(0) - p''(1), \]
  \[ c = p'(0), \]
  \[ d = p(0). \]
Hermite and Algebraic Forms

• Putting it all together produces the matrix formulation for the Hermite curve \( P(\mu) \)

\[
P(\mu) = \mathbf{a} u^3 + \mathbf{b} u^2 + \mathbf{c} u + \mathbf{d}
\]

\[
\mathbf{a} = 2 \mathbf{p}(0) - 2 \mathbf{p}(1) + \mathbf{p}'(0) + \mathbf{p}'(1)
\]

\[
\mathbf{b} = -3 \mathbf{p}(0) + 3 \mathbf{p}(1) - 2 \mathbf{p}'(0) - \mathbf{p}'(1)
\]

\[
\mathbf{c} = \mathbf{p}'(0)
\]

\[
\mathbf{d} = \mathbf{p}(0)
\]

\[
P(\mu) = (2\mu^3 - 3\mu^2 + 1)\mathbf{p}(0) + (-2\mu^3 + 3\mu^2 - \mu)\mathbf{p}'(0) + (\mu^3 - 2\mu^2 + \mu)\mathbf{p}'(1) + (\mu^3 - \mu^2)\mathbf{p}''(1)
\]

Hermite Curves

• Putting it all together

\[
P(\mu) = \mathbf{a} u^3 + \mathbf{b} u^2 + \mathbf{c} u + \mathbf{d}
\]

\[
\mathbf{a} = 2 \mathbf{p}(0) - 2 \mathbf{p}(1) + \mathbf{p}'(0) + \mathbf{p}'(1)
\]

\[
\mathbf{b} = -3 \mathbf{p}(0) + 3 \mathbf{p}(1) - 2 \mathbf{p}'(0) - \mathbf{p}'(1)
\]

\[
\mathbf{c} = \mathbf{p}'(0)
\]

\[
\mathbf{d} = \mathbf{p}(0)
\]

\[
P(\mu) = (2\mu^3 - 3\mu^2 + 1)\mathbf{p}(0) + (-2\mu^3 + 3\mu^2 - \mu)\mathbf{p}'(0) + (\mu^3 - 2\mu^2 + \mu)\mathbf{p}'(1) + (\mu^3 - \mu^2)\mathbf{p}''(1)
\]

Hermite Curves - Matrix Form

• Putting this in matrix form

\[
\mathbf{H} = [H_1(\mu) \quad H_2(\mu) \quad H_3(\mu) \quad H_4(\mu)]
\]

\[
\begin{bmatrix}
2 & -3 & 0 & 1 \\
-2 & 3 & 0 & 0 \\
1 & -2 & 1 & 0 \\
1 & -1 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\mu^3 \\
\mu^2 \\
\mu \\
1
\end{bmatrix}
\]

= \mathbf{GU}

• \( \mathbf{GU} \) is called the Hermite characteristic matrix

• Collecting the Hermite geometric coefficients into a geometry vector \( \mathbf{G} \),

\[
\mathbf{G} = [\mathbf{p}(0) \quad \mathbf{p}(1) \quad \mathbf{p}'(0) \quad \mathbf{p}'(1)]
\]

Hermite Basis

• Substituting for the coefficients and collecting terms gives

\[
P(\mu) = (2\mu^3 - 3\mu^2 + 1)\mathbf{p}(0) + (-2\mu^3 + 3\mu^2 - \mu)\mathbf{p}'(0) + (\mu^3 - 2\mu^2 + \mu)\mathbf{p}'(1) + (\mu^3 - \mu^2)\mathbf{p}''(1)
\]

• Then

\[
P(\mu) = H_1(\mu)\mathbf{p}(0) + H_2(\mu)\mathbf{p}(1) + H_3(\mu)\mathbf{p}'(0) + H_4(\mu)\mathbf{p}'(1)
\]

Hermite Curves - Blending Functions

\[
\mu = 0:
\]

- \( H_1 = 1, H_2 = H_3 = H_4 = 0 \)

\[
P(0) = \mathbf{p}0
\]

- \( H_1' = H_2' = H_4' = 0, H_3' = 1 \)

\[
P'(0) = \mathbf{T}0
\]

\[
\mu = 1:
\]

- \( H_1 = H_3 = H_4 = 0, H_2 = 1 \)

\[
P(1) = \mathbf{p}1
\]

- \( H_1' = H_2' = H_3' = 0, H_4' = 1 \)

\[
P'(1) = \mathbf{T}1
\]
Hermite to Bézier

- Mixture of points and vectors is awkward and unintuitive
- Specify tangents as differences of points

\[ p_0 = q_0; \quad p_3 = q_1; \]
\[ p_1 = q_0 + (1/3)t_0; \quad p_2 = q_1 - (1/3)t_1 \]

- note derivative is defined as 3 times offset

Beziers to Hermite

\[ q_0 = p_0; \quad q_1 = p_3; \]
\[ t_0 = 3(p_1 - p_0); \quad t_1 = 3(p_3 - p_2); \]

- note derivative is defined as 3 times offset

Issues with Bézier Curves

- Creating complex curves requires many control points
  - potentially a very high-degree polynomial with many wiggles
- Bézier blending functions have global support over the whole curve
  - move just one point, change whole curve
- Improved Idea: link (C\(_1\) ) lots of low degree (cubic) Bézier curves end-to-end

Continuity

Two types:
- Geometric Continuity, G\(_i\):
  - endpoints meet
  - tangent vectors’ directions are equal
- Parametric Continuity, C\(_i\):
  - endpoints meet
  - tangent vectors’ directions are equal
  - tangent vectors’ magnitudes are equal
- In general: C\(_i\) implies G\(_i\) but not vice versa
Parametric Continuity

- **Continuity** (recall from the calculus):
  - Two curves are $C_i$ continuous at a point $p$ iff the $i$-th derivatives of the curves are equal at $p$

![Parametric Continuity Diagram](image)

Continuity

- What are the conditions for $C^0$ and $C^1$ continuity at the joint of curves $x'$ and $x''$?
  - Tangent vectors at end points equal
  - End points equal

$Q'(1) = Q'(0), \quad \frac{dQ'}{dt}(1) = \frac{dQ'}{dt}(0)$

![Continuity Diagram](image)

Chaining Bézier curves

- No continuity built in
- Achieve $C^1$ using collinear control points around join points

![Chaining Bézier curves Diagram](image)

Catmull-Rom splines

- Our first example of an interpolating spline
- Like Bézier, equivalent to Hermite
  - In fact, all splines of this form are equivalent
- First example of a spline based on just an input point sequence
- Does not have convex hull property
- Only has $C^1$ continuity

![Catmull-Rom splines Diagram](image)
Catmull-Rom splines
• A sequence of Hermite/Bezier curves
• Would like to define tangents automatically
  – use adjacent control points
  – end tangents: user-defined or fit a parabola

Catmull-Rom splines
• Tangents are \((p_{k+1} - p_{k-1}) / 2\) for interior control points \((p_k)\)
• User specifies tangents at first \((T_0)\) and last \((T_N)\) input points
• Or fit parabola to first/last 3 points
  \[ q_0 = p_k \]
  \[ q_1 = p_{k+1} \]
  \[ t_0 = 0.5(p_{k+1} - p_{k-1}) \]
  \[ t_1 = 0.5(p_{k+2} - p_k) \]

Adding tension
• Adding tension to Catmull-Rom spline involves adjusting tangents at interior join points, \(p_i\)
  \[ t_0 = (1 - T)0.5(p_{k+1} - p_{k-1}) \]
  \[ t_1 = (1 - T)0.5(p_{k+2} - p_k) \]
• When \(T=0\), standard C-R spline
• When \(T=1\), tangent is zero

Adding tension
• Scale user-provided tangent vectors
  \[ T_0' = (1 - T) T_0 \]
  \[ T_N' = (1 - T) T_N \]

Programming Assignment 2
• Process command-line arguments
• Read in 3D input points and tangents
• Compute tangents at interior input points
• Modify tangents with tension parameter
• Compute Bezier control points for curves defined by each two input 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