Splines

Hermite Splines
Bezier Splines
Catmull-Rom Splines
Other Cubic Splines
[Angel Ch. 10]

Roller coaster

- Next programming assignment involves creating a 3D roller coaster animation
- We must model the 3D curve describing the roller coaster, but how?

Modeling Complex Shapes

- We want to build models of very complicated objects
- Complexity is achieved using simple pieces
  - polygons,
  - parametric curves and surfaces, or
  - implicit curves and surfaces
- This lecture: parametric curves

What Do We Need From Curves in Computer Graphics?

- Local control of shape (so that easy to build and modify)
- Stability
- Smoothness and continuity
- Ability to evaluate derivatives
- Ease of rendering

Curve Representations

- Explicit: \( y = f(x) \)
  - \( y = x^3 \)
  - \( y = mx + b \)
  - Must be a function (single-valued)
  - Big limitation—vertical lines?
- Parametric: \((x,y,z) = (f(u),g(u),h(u))\)
  - Easy to specify, modify, control
  - Extra "hidden" variable \( u \), the parameter \((x,y) = (\cos u, \sin u)\)
- Implicit (2D): \( f(x,y) = 0 \)
  - \( y \) can be a multiple valued function of \( x \)
  - Hard to specify, modify, control
  - \( x^2 + y^2 - r^2 = 0 \)

Parameterization of a Curve

- Parameterization of a curve: how a change in \( u \) moves you along a given curve in xyz space.
- Parameterization is not unique. It can be slow, fast, with continuous / discontinuous speed, clockwise (CW) or CCW...
Polynomial Interpolation

- An $n$-th degree polynomial fits a curve to $n+1$ points
  - called Lagrange Interpolation
  - result is a curve that is too wiggly, change to any control point affects entire curve (non-local)
  - this method is poor
- We usually want the curve to be as smooth as possible
  - minimize the wiggles
  - high-degree polynomials are bad

Lagrange interpolation, degree=15

Splines: Piecewise Polynomials

- A spline is a piecewise polynomial:
  Curve is broken into consecutive segments, each of which is a low-degree polynomial interpolating (passing through) the control points
- Cubic piecewise polynomials are the most common:
  - They are the lowest order polynomials that
    1. interpolate two points and
    2. allow the gradient at each point to be defined ($C^1$ continuity is possible).
  - Piecewise definition gives local control.
  - Higher or lower degrees are possible, of course.

Splines

- Types of splines:
  - Hermite Splines
  - Bezier Splines
  - Catmull-Rom Splines
  - Natural Cubic Splines
  - B-Splines
  - NURBS
- Splines can be used to model both curves and surfaces

Cubic Curves in 3D

- Cubic polynomial:
  - $p(u) = au^3 + bu^2 + cu + d$ with $a, b, c, d \in \mathbb{R}$
- Three cubic polynomials, one for each coordinate:
  - $x(u) = ax^3 + bx^2 + cx + d_x$
  - $y(u) = ay^3 + by^2 + cy + d_y$
  - $z(u) = az^3 + bz^2 + cz + d_z$
- In matrix notation:
  - $\begin{bmatrix} x(u) \\ y(u) \\ z(u) \end{bmatrix} = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} a_x \\ b_x \\ c_x \\ d_x \\ a_y \\ b_y \\ c_y \\ d_y \\ a_z \\ b_z \\ c_z \\ d_z \end{bmatrix}$
- Or simply:
  - $p = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \cdot A$

Cubic Hermite Splines

We want a way to specify the end points and the slope at the end points!
Deriving Hermite Splines

- Four constraints: value and slope (in 3-D, position and tangent vector) at beginning and end of interval [0, 1]:
  \[ p(0) = p_1 = (x_1, y_1, z_1) \]
  \[ p(1) = p_2 = (x_2, y_2, z_2) \]
  \[ p'(0) = \text{user constraints} \]
  \[ p'(1) = \text{user constraints} \]
- Assume cubic form: \( p(u) = au^3 + bu^2 + cu + d \)
- Four unknowns: \( a, b, c, d \)

\[ \begin{align*}
  p_1 &= p(0) = d \\
  p_2 &= p(1) = a + b + c + d \\
  p_1' &= p'(0) = c \\
  p_2' &= p'(1) = 3a + 2b + c
\end{align*} \]

The Cubic Hermite Spline Equation

- After inverting the 4x4 matrix, we obtain:
  \[
  \begin{bmatrix}
    x & y & z & 1 \\
    2 & -1 & 0 & 0 \\
    -3 & 3 & -1 & 0 \\
    1 & -2 & 1 & 0
  \end{bmatrix}
  \begin{bmatrix}
    x_1 & y_1 & z_1 \\
    x_2 & y_2 & z_2 \\
    x_1' & y_1' & z_1' \\
    x_2' & y_2' & z_2'
  \end{bmatrix}
  \]

- This form is typical for splines:
  - basis matrix and meaning of control matrix change with the spline type

Piecing together Hermite Splines

It's easy to make a multi-segment Hermite spline:
- each segment is specified by a cubic Hermite curve
- just specify the position and tangent at each “joint” (called knot)
- the pieces fit together with matched positions and first derivatives
- gives C1 continuity
Hermite Splines in Adobe Illustrator

### Hermite Splines
- Variant of the Hermite spline
- Instead of endpoints and tangents, four control points
  - Points P1 and P4 are on the curve
  - Points P2 and P3 are off the curve
  - $p(0) = P_1$, $p(1) = P_4$
  - $p'(0) = 3(P_2 - P_1)$, $p'(1) = 3(P_4 - P_3)$
- Basis matrix is derived from the Hermite basis (or from scratch)
- Convex Hull property: curve contained within the convex hull of control points
- Scale factor "3" is chosen to make "velocity" approximately constant

### The Bezier Spline Matrix
\[
\begin{bmatrix}
2 & 2 & 1 & 1 \\
-3 & -2 & -1 & 0 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
x_1 & y_1 & z_1 \\
x_2 & y_2 & z_2 \\
x_3 & y_3 & z_3 \\
x_4 & y_4 & z_4 \\
\end{bmatrix}
\]

### Bezier Blending Functions
\[ p(t) = \begin{bmatrix} (1-t)^3 & 3(1-t)^2 t & 3(1-t)t^2 & t^3 \end{bmatrix}^T \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} \]

### DeCasteljau Construction
Efficient algorithm to evaluate Bezier splines. Similar to Horner rule for polynomials. Can be extended to interpolations of 3D rotations.

### Catmull-Rom Splines
- Roller-coaster (next programming assignment)
- With Hermite splines, the designer must arrange for consecutive tangents to be collinear, to get $C^1$ continuity. Similar for Bezier. This gets tedious.
- Catmull-Rom: an interpolating cubic spline with built-in $C^1$ continuity.
- Compared to Hermite/Bezier: fewer control points required, but less freedom.
Constructing the Catmull-Rom Spline

Suppose we are given \( n \) control points in 3-D: \( p_1, p_2, \ldots, p_n \).

For a Catmull-Rom spline, we set the tangent at \( p_i \) to \( s \times (p_{i+1} - p_{i-1}) \) for \( i = 2, \ldots, n-1 \), for some \( s \) (often \( s=0.5 \)).

\( s \) is tension parameter: determines the magnitude (but not direction!) of the tangent vector at point \( p_i \).

What about endpoint tangents? Use extra control points \( p_0, p_{n+1} \).

Now we have positions and tangents at each knot. This is a Hermite specification. Now, just use Hermite formulas to derive the spline.

Note: curve between \( p_i \) and \( p_{i+1} \) is completely determined by \( p_{i-1}, p_i, p_{i+1}, p_{i+2} \).

Splines with More Continuity?

- So far, only \( C^1 \) continuity.
- How could we get \( C^2 \) continuity at control points?
- Possible answers:
  - Use higher degree polynomials: degree 4 = quartic, degree 5 = quintic, ... but these get computationally expensive, and sometimes wiggly
  - Give up local control \rightarrow natural cubic splines
    A change to any control point affects the entire curve
  - Give up interpolation \rightarrow cubic B-splines
    Curve goes near, but not through, the control points

Comparison of Basic Cubic Splines

<table>
<thead>
<tr>
<th>Type</th>
<th>Local Control</th>
<th>Continuity</th>
<th>Interpolation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hermite</td>
<td>YES</td>
<td>C1</td>
<td>YES</td>
</tr>
<tr>
<td>Bezier</td>
<td>YES</td>
<td>C1</td>
<td>YES</td>
</tr>
<tr>
<td>Catmull-Rom</td>
<td>YES</td>
<td>C1</td>
<td>YES</td>
</tr>
<tr>
<td>Natural</td>
<td>NO</td>
<td>C2</td>
<td>YES</td>
</tr>
<tr>
<td>B-Splines</td>
<td>YES</td>
<td>C2</td>
<td>NO</td>
</tr>
</tbody>
</table>

Summary:
Cannot get \( C^2 \), interpolation and local control with cubics

Natural Cubic Splines

- If you want 2nd derivatives at joints to match up, the resulting curves are called natural cubic splines
- It’s a simple computation to solve for the cubics’ coefficients. (See Numerical Recipes in C book for code.)
- Finding all the right weights is a global calculation (solve tridiagonal linear system)

B-Splines

- Give up interpolation:
  - the curve passes near the control points
  - best generated with interactive placement (because it’s hard to guess where the curve will go)
- Curve obeys the convex hull property
- \( C^2 \) continuity and local control are good compensation for loss of interpolation
B-Spline Basis

• We always need 3 more control points than the number of spline segments

\[
M_B = \begin{bmatrix}
-1 & 3 & -3 & 1 \\
3 & -6 & 3 & 0 \\
-3 & 0 & 3 & 0 \\
1 & 4 & 1 & 0 \\
\end{bmatrix}
\]

\[
G_B = \begin{bmatrix}
p_{i-3} \\
p_{i-2} \\
p_{i-1} \\
p_i \\
\end{bmatrix}
\]

Other Common Types of Splines

• Non-uniform Splines
• Non-Uniform Rational Cubic curves (NURBS)
• NURBS are very popular and used in many commercial packages

How to Draw Spline Curves

• Basis matrix equation allows same code to draw any spline type
• Method 1: brute force
  - Calculate the coefficients
  - For each cubic segment, vary \( u \) from 0 to 1 (fixed step size)
  - Plug in \( u \) value, matrix multiply to compute position on curve
  - Draw line segment from last position to current position
• What’s wrong with this approach?
  - Draws in even steps of \( u \)
  - Even steps of \( u \) does not mean even steps of \( x \)
  - Line length will vary over the curve
  - Want to bound line length
    - too long: curve looks jagged
    - too short: curve is slow to draw

• Method 2: recursive subdivision - vary step size to draw short lines

\[
\text{Subdivide}(u_0, u_1, \text{maxlinelength})
\]

\[
\text{umid} = (u_0 + u_1)/2
\]

\[
x_0 = F(u_0)
\]

\[
x_1 = F(u_1)
\]

\[
\text{if } |x_1 - x_0| > \text{maxlinelength}
\]

\[
\text{Subdivide}(u_0, \text{umid}, \text{maxlinelength})
\]

\[
\text{Subdivide}(\text{umid}, u_1, \text{maxlinelength})
\]

\[
\text{else } \text{drawline}(x_0, x_1)
\]

• Variant on Method 2 - subdivide based on curvature
  - replace condition in “if” statement with straightness criterion
  - draws fewer lines in flatter regions of the curve

Summary

• Piecewise cubic is generally sufficient
• Define conditions on the curves and their continuity

• Most important:
  - basic curve properties (what are the conditions, controls, and properties for each spline type)
  - generic matrix formula for uniform cubic splines \( p(u) = u B G \)
  - given a definition, derive a basis matrix (do not memorize the matrices themselves)