Announcements

Office hours for Jernej Barbic have changed:

New office hours:

Tuesday (same as before)
1 p.m. – 2 p.m. in the cluster

Thursday (changed)
1:45 p.m. - 2:45 p.m. in the cluster
Announcements (Contd.)

A link (URL) to the *libpicio* library has been added to the course website.

*Libpicio* library:

- read and write TIFF, XPM, JPEG formats
- used in assignment #1
Roller coaster

• Next programming assignment involves creating a 3D roller coaster animation
• We must model the 3D curve describing the roller coaster, but how?
• How to make the simulation obey the laws of gravity?
Modeling Complex Shapes

- We want to build models of very complicated objects
- An equation for a sphere is possible, but how about an equation for a telephone, or a face, or a cloud?
- 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:** $y = f(x)$
  
  \[
  y = mx + b \quad y = x^2
  \]
  
  - Must be a function (single-valued):
  - Big limitation—vertical lines?

- **Parametric:** $(x, y) = (f(u), g(u))$
  
  \[
  (x, y) = (\cos u, \sin u)
  \]
  
  + Easy to specify, modify, control
  - Extra “hidden” variable $u$, the *parameter*

- **Implicit:** $f(x, y) = 0$
  
  \[
  x^2 + y^2 - r^2 = 0
  \]
  
  + $Y$ can be multiple valued function of $x$
  - Hard to specify, modify, control
Parameterization of a Curve

- *Parameterization* of a curve: how a change in u moves you along a given curve in xyz space.
- There are an infinite number of parameterizations of a given curve. Slow, fast, speed continuous or discontinuous, 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 (nonlocal) – *this method is poor*

- We usually want the curve to be as smooth as possible
  - minimize the wiggles
  - high-degree polynomials are bad
Splines: Piecewise Polynomials

- A spline is a *piecewise polynomial* - many low degree polynomials are used to interpolate (pass through) the control points

- *Cubic piecewise* polynomials are the most common:
  - piecewise definition gives local control
  - lowest order polynomials that interpolate two points and allow the gradient at each point to be defined - $C^1$ continuity is possible
  - Higher or lower degrees are possible, of course
Splines (contd.)

- Types of splines:
  - Hermite Splines
  - Bezier Splines
  - Catmull-Rom Splines
  - Natural Cubic Splines
  - B-Splines
  - NURBS

- Splines can be used to model surfaces
Piecewise Polynomials

- Spline: lots of little polynomials pieced together
- Want to make sure they fit together nicely

C₀ continuity

Continuous in position

C₀ & C₁ continuity

Continuous in position and tangent vector

C₀ & C₁ & C₂ continuity

Continuous in position, tangent, and curvature
Cubic Curves in 3D

- Cubic polynomial
  - $x(u) = au^3 + bu^2 + cu + d = uA$
  - where $u = [u^3 \ u^2 \ u \ 1]^T$, $a = [a \ b \ c \ d]^T$
- Three cubic polynomials, one for each coordinate
  - $x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$
  - $y(u) = a_y u^3 + b_y u^2 + c_y u + d_y$
  - $z(u) = a_z u^3 + b_z u^2 + c_z u + d_z$
- In matrix notation

$$[x(u) \ y(u) \ z(u)] = [u^3 \ u^2 \ u \ 1] \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix}$$

- Or simply $x = uA$
An Illustrative Example

• Cubic Hermite Splines

That is, we want a way to specify the end points and the slope at the end points!
Deriving Hermite Splines

- **Four constraints:** value and slope (or in 3-D, position and tangent vector) at beginning and end of interval \([0, 1]\)
  
  \[
  \begin{align*}
  x(0) &= x_1 \\
  x(1) &= x_2 \\
  x'(0) &= x_1' \\
  x'(1) &= x_2'
  \end{align*}
  \]

  primes on left side denote derivative; primes on right denote slope constants

- **Assume cubic form:** \(x(u) = au^3 + bu^2 + cu + d\)

- **Four unknowns:** \(a, b, c, d\)
The Cubic Hermite Spline Equation

- Using some algebra, we obtain:

\[
\begin{bmatrix}
  x & y & z \\
\end{bmatrix} = \begin{bmatrix}
  u^3 & u^2 & u & 1 \\
\end{bmatrix} \begin{bmatrix}
  2 & -2 & 1 & 1 \\
  -3 & 3 & -2 & -1 \\
  0 & 0 & 1 & 0 \\
  1 & 0 & 0 & 0 \\
\end{bmatrix} \begin{bmatrix}
  x_1 & y_1 & z_1 \\
  x_2 & y_2 & z_2 \\
  \frac{dx_1}{du} & \frac{dy_1}{du} & \frac{dz_1}{du} \\
  \frac{dx_2}{du} & \frac{dy_2}{du} & \frac{dz_2}{du} \\
\end{bmatrix}
\]

The point that gets drawn is the basis, and the control matrix is what the user gets to pick.

- This form typical for splines
  - basis matrix and meaning of control matrix change with the spline type
Four Basis Functions for Hermite splines

\[
p(u) = \begin{bmatrix} 2u^3 - 3u^2 + 1 \\ -2u^3 + 3u^2 \\ u^3 - 2u^2 + u \\ u^3 - u^2 \end{bmatrix}^\top \begin{bmatrix} p_1 \\ p_2 \\ \nabla p_1 \\ \nabla p_2 \end{bmatrix}
\]

4 Basis Functions

Every cubic Hermite spline is a linear combination (blend) of these 4 functions.
Piecing together Hermite Curves

- It's easy to make a multi-segment Hermite spline
  - each piece is specified by a cubic Hermite curve
  - just specify the position and tangent at each "joint"
  - the pieces fit together with matched positions and first derivatives
  - gives C1 continuity

- Given a list of points & tangents, you can build a piecewise cubic that passes through all the points by calculating the Hermite cubic for each segment

- The points that the curve has to pass through are called knots or knot points
Bezier Curves*

- Another variant of the same game
- Instead of endpoints and tangents, four control points
  - Points P1 and P4 are on the curve
  - Points P2 and P3 are off the curve
  - \( X(0) = P1, X(1) = P4, \)
  - \( X'(0) = 3(P2-P1), X'(1) = 3(P4 - P3) \)
- Variant of the Hermite spline
  - Basis matrix derived from the Hermite basis (or from scratch)
- Convex Hull property
  - Curve contained within convex hull of control points
- Gives more uniform control knobs (series of points) than Hermite
- Scale factor (3) is chosen to make “velocity” approximately constant
The Bezier Spline Matrix*

\[
\begin{bmatrix}
    x & y & z
\end{bmatrix} =
\begin{bmatrix}
    u^3 & u^2 & u & 1
\end{bmatrix}
\begin{bmatrix}
    2 & -2 & 1 & 1 \\
    -3 & 3 & -2 & -1 \\
    0 & 0 & 1 & 0 \\
    1 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
    x_1 & y_1 & z_1 \\
    x_2 & y_2 & z_2 \\
    x_3 & y_3 & z_3 \\
    x_4 & y_4 & z_4
\end{bmatrix}
\]

Hermite basis  Bezier to Hermite  Bezier control vector

\[
= \begin{bmatrix}
    u^3 & u^2 & u & 1
\end{bmatrix}
\begin{bmatrix}
    -1 & 3 & -3 & 1 \\
    3 & -6 & 3 & 0 \\
    -3 & 3 & 0 & 0 \\
    1 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
    x_1 & y_1 & z_1 \\
    x_2 & y_2 & z_2 \\
    x_3 & y_3 & z_3 \\
    x_4 & y_4 & z_4
\end{bmatrix}
\]

Bezier basis  Bezier control vector

Computer Graphics 15-462
Beziers Blending Functions

$p(t) = \begin{bmatrix} (1-t)^3 \\ 3t(1-t)^2 \\ 3t^2(1-t) \\ t^3 \end{bmatrix}^T \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}$

Also known as the order 4, degree 3 Bernstein polynomials
Nonnegative, sum to 1
The entire curve lies inside the polyhedron bounded by the control points
It's easy to subdivide Bezier curves*

Each half is a Bezier curve, therefore it is easy to draw them by subdivision
Catmull-Rom Splines

- Use for the 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.
Catmull-Rom Splines, contd.

- Given $n$ control points in 3-D: $p_1, p_2, \ldots, p_n$:
  - Tangent at $p_i$ given by $s^*(p_{i+1} - p_{i-1})$ for $i=2 \ldots n-1$, for some $s$
  - $s$ is tension parameter: determines the magnitude (but not direction!) of the required tangent at point $p_i$
  - What about endpoint tangents? (several good answers: extrapolate, or use extra control points $p_0, p_{n+1}$)
  - Now we have positions and tangents at each knot – a Hermite specification.
  - Curve between $p_i$ and $p_{i+1}$ is determined by $p_{i-1}, p_i, p_{i+1}, p_{i+2}$
Catmull-Rom Spline Matrix

\[
\begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix} =
\begin{bmatrix}
  u^3 & u^2 & u & 1
\end{bmatrix}
\begin{bmatrix}
  -s & 2-s & s-2 & s \\
  2s & s-3 & 3-2s & -s \\
  -s & 0 & s & 0 \\
  0 & 1 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
  x_1 & y_1 & z_1 \\
  x_2 & y_2 & z_2 \\
  x_3 & y_3 & z_3 \\
  x_4 & y_4 & z_4
\end{bmatrix}
\]

- Derived similarly to Hermite and Bezier
- Parameter s is typically set to s=1/2.
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**
  - Can’t get C2, interpolation and local control with cubics - very sad.
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)

*Note: The asterisk (*) indicates a starred item that might be optional or of lesser importance.*
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
- C2 continuity and local control are good compensation for loss of interpolation
B-Spline Basis*

- We always need 3 more control points than spline pieces

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

\[ G_{Bs_i} = \begin{bmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{bmatrix} \]
Other common types of splines*

- Non-uniform Splines
- Non-Uniform Rational Cubic curves (NURBS)
How to Draw Spline Curves

- Basis matrix eqn. 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 \( u \) ≠ 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
Drawing Splines, 2

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

  \[\text{Subdivide}(u_0, u_1, \text{maxlinelength})\]
  \[\quad u_{\text{mid}} = (u_0 + u_1)/2\]
  \[\quad x_0 = F(u_0)\]
  \[\quad x_1 = F(u_1)\]
  \[\quad \text{if } |x_1 - x_0| > \text{maxlinelength}\]
  \[\quad \text{Subdivide}(u_0, u_{\text{mid}}, \text{maxlinelength})\]
  \[\quad \text{Subdivide}(u_{\text{mid}}, u_1, \text{maxlinelength})\]
  \[\quad \text{else drawline}(x_0, x_1)\]

- **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
In Summary...

• Summary:
  - piecewise cubic is generally sufficient
  - define conditions on the curves and their continuity

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