Interpolation and Curves and Motion on a Path
Problem statement

• How can we construct smooth paths?
  – Define smoothness in terms of geometry
  – What is the input?
  – Where does the input come from?
    • Pathfinding data
    • Animator specified
Interpolation

- **Interpolation**: The process of inserting in a series an intermediate number or quantity ascertained by calculation from those already known.

- Examples
  - Computing midpoints
  - Curve fitting
Interpolation terms

• Order (of the polynomial)
  – Linear
  – Quadratic
  – Cubic

• Dimensions
  – Bilinear (data in a 2D grid)
  – Trilinear (data in a 3D grid)
Linear interpolation

• Given two points, $P_0$ and $P_1$ in 2D
• Parametric line equation:

$$P = P_0 + t (P_1 - P_0) ; \text{OR}$$

$$\begin{cases} X = P_0.x + t (P_1.x - P_0.x) \\ Y = P_0.y + t (P_1.y - P_0.y) \end{cases}$$

• $t = 0 \rightarrow$ Beginning point $P_0$
• $t = 1 \rightarrow$ End point $P_1$
Linear interpolation

• Rewrite the parametric equation

\[ P = (1-t)P_0 + t P_1; \text{ OR} \]

\[ \begin{align*}
X &= (1-t)P_0.x + t P_1.x \\
Y &= (1-t)P_0.y + t P_1.y
\end{align*} \]

• Formula is equivalent to a weighted average
• \( t \) is the weight (or percent) applied to \( P_1 \)
• \( 1 - t \) is the weight (or percent) applied to \( P_0 \)

• \( t = 0.5 \rightarrow \text{Midpoint between } P_0 \text{ and } P_1 = P_{\text{mid}} \)
• \( t = 0.25 \rightarrow 1^{\text{st}} \text{ quartile } = \text{midpoint between } P_0 \text{ and } P_{\text{mid}} \)
• \( t = 0.75 \rightarrow 3^{\text{rd}} \text{ quartile } = \text{midpoint between } P_{\text{mid}} \text{ and } P_1 \)
Bilinear interpolation process

- Given 4 points (Q's)
- Interpolate in one dimension
  - $Q_{11}$ and $Q_{21}$ give $R_1$
  - $Q_{12}$ and $Q_{22}$ give $R_2$
- Interpolate with the results
  - $R_1$ and $R_2$ give $P$
Bilinear interpolation application

• Resizing an image
Curve Fitting

Interpolation vs. approximation

Polynomial complexity

Continuity: first degree (tangential)

Local vs. global control: local

Information requirements: tangents needed?
Interpolation vs. Approximation
Polynomial complexity: Cubic

Low complexity
reduced computational cost

With a point of inflection, the curve can match arbitrary tangents at end points

Therefore, choose a cubic polynomial
Continuity orders

$C^{-1}$ discontinuous

$C^0$ continuous

$C^1$: first derivative is continuous

$C^2$: first and second derivatives are continuous
Local vs. Global control

Local Control: moving one control point only changes the curve over a finite bounded region

Global Control: moving one control point changes the entire curve; distant sections may change only slightly.
Information requirements

- just the points
- tangents
- interior control points
- just beginning and ending tangents
Cubic Bezier Curve

- Find the point \( x \) on the curve as a function of parameter \( u \):
de Casteljau Algorithm

• Describe the curve as a recursive series of linear interpolations

• Intuitive, but not the most efficient form
de Casteljau Algorithm

- Cubic Bezier curve has four points (though the algorithm works with any number of points)
de Casteljau Algorithm

\[ q_0 = \text{Lerp}(u, p_0, p_1) \]
\[ q_1 = \text{Lerp}(u, p_1, p_2) \]
\[ q_2 = \text{Lerp}(u, p_2, p_3) \]

Lerp = linear interpolation
de Casteljau Algorithm

\[ r_0 = \text{Lerp}(u, q_0, q_1) \]
\[ r_1 = \text{Lerp}(u, q_1, q_2) \]
de Casteljau Algorithm

\[ x = \text{Lerp}(u, r_0, r_1) \]
Bezier Curve
Beziers Curve Animation

http://en.wikipedia.org/wiki/B%C3%A9zier_curve
Cubic Bezier

\[ P = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} -1.0 & 3.0 & -3.0 & 1.0 \\ 3.0 & -6.0 & 3.0 & 0.0 \\ -3.0 & 3.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} \begin{bmatrix} p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3} \end{bmatrix} \]

Curve runs through \( P_i \) and \( P_{i+3} \) with starting tangent \( P_i P_{i+1} \) and ending tangent \( P_{i+2} P_{i+3} \)
Controlling Motion along $p=P(u)$

Step 1. vary $u$ to create points on the curve

Step 2. Reparameterization by arc length

$u = U(s)$ where $s$ is distance along the curve

Step 3. Speed control

for example, ease-in / ease-out

$s = ease(t)$ where $t$ is time
Reparameterizing by Arc Length

Analytic
Forward differencing
  Supersampling
  Adaptive approach
Numerically
  Adaptive Gaussian
Reparameterizing by Arc Length - supersample

1. Calculate a bunch of points at small increments in $u$
2. Compute summed linear distances as approximation to arc length
3. Build table of (parametric value, arc length) pairs

Notes
1. Often useful to normalize total distance to 1.0
2. Often useful to normalize parametric value for multi-segment curve to 1.0
<table>
<thead>
<tr>
<th>index</th>
<th>u</th>
<th>Arc Length (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>0.05</td>
<td>0.080</td>
</tr>
<tr>
<td>2</td>
<td>0.10</td>
<td>0.150</td>
</tr>
<tr>
<td>3</td>
<td>0.15</td>
<td>0.230</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>20</td>
<td>1.00</td>
<td>1.000</td>
</tr>
</tbody>
</table>
Speed Control

Time-distance function
- Ease-in
- Ease-out
- Sinusoidal
- Cubic polynomial
- Constant acceleration
- General distance-time functions
Time Distance Function

\[ s = S(t) \]

\[ t \rightarrow \text{the global time variable} \]
Ease-in/Ease-out Function

\[ s = S(t) \]

Normalize distance and time to 1.0 to facilitate reuse.
Ease-in: Sinusoidal

\[ s = ease(t) = \left( \sin(t\pi - \pi/2) + 1 \right) / 2 \]
Ease-in: Single Cubic

\[ s = ease(t) = -2t^3 + 3t^2 \]
Ease-in: Constant Acceleration

\[ v = v_0 \cdot \frac{t}{t_1} \quad 0.0 < t < t_1 \]
\[ v = v_0 \quad t_1 < t < t_2 \]
\[ v = v_0 \left( 1.0 - \frac{t - t_2}{1 - t_2} \right) \quad t_2 < t < 1.0 \]
Ease-in: Constant Acceleration
Motion on a curve – solution steps

1. Construct a space curve that interpolates the given points with piecewise first order continuity
   \[ p = P(u) \]

2. Construct an arc-length-parametric-value function for the curve
   \[ u = U(s) \]

3. Construct time-arc-length function according to given constraints
   \[ s = S(t) \]

   \[ p = P(U(S(t))) \]