Interpolation and Motion
Interpolation and Basic Techniques

• Abstract out the important features of the animation?

• A very subjective thing to do $\rightarrow$ the appropriate feature for one kind of animation will not be the appropriate feature for another kind of animation.

• Our focus $\rightarrow$ various techniques in which the animator is responsible for specifying most of the information (low level of abstraction).
Interpolation and Basic Techniques

1. Interpolation
2. Motion along a curve (arc length)
3. Interpolation of rotations (quaternions)
4. Path following
Interpolation

• At the foundation of almost all animation is the interpolation of values.

• The simplest case is interpolating the position of a point in space.

• Even this is non-trivial to do correctly and requires some discussion of several issues:
  – the appropriate parameterization of position,
  – the appropriate interpolating function,
  – and maintaining the desired control of the interpolation over time.
Interpolation

• Often, an animator has a list of values associated with a given parameter at specific frames (called *key frames* or *keys*) of the animation.

![Diagram showing key frames](image)

• The question to be answered is how best to generate the values of the parameter for the frames between the key frames.

• The parameter to be interpolated may be
  – a coordinate of the position of an object,
  – a joint angle of an appendage of a robot,
  – the transparency attribute of an object,
  – any other parameter
Interpolation

• Our focus → How to choose the most appropriate interpolation technique and apply it in the production of an animated sequence.

• One of the first decisions to make is whether the given values represent actual values that the parameter should have at the key frames (interpolation), or whether they are meant merely to control the interpolating function and do not represent actual values the parameter will assume (approximation).
Interpolation

- Other issues that influence which interpolation technique to use include how smooth the resulting function needs to be (i.e. continuity),

\[
\text{a) positional discontinuity at the point} \quad \text{b) positional continuity but not tangential continuity at the point}
\]

\[
\text{c) positional and tangential continuity but not curvature continuity at the point} \quad \text{d) positional, tangential, and curvature continuity at the point}
\]
Interpolation

• how much computation you can afford to do (order of interpolating polynomial, and whether local or global control of the interpolating function is required.

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.
Spline curves & control points

- **Spline**
  - Created by mathematical technique that fits a curve to a set of given points called Control Points: Curve fitting
  - Artist places the points, and program creates a curve based on the points

- **Polyline**
  - series of straight lines through the points

- **B-splines**
  - blend position of control points without passing through any of them

- **Non-uniform rational B-splines (NURB)**
Bezier curve

- Named after French engineer Pierre Bezier
- Direct control over the location of the curve with anchor points
- Over the curvature between the points with two off-the curve control points
- Difficult: off-path control points

- Catmull-Rom Spline
  - Interpolating spline
  - Restricts the curvature choice but places all points on the curve
  - Make drawing and editing more straightforward

*Often used in animation path movement*
Curves

- Hermite
- Bezier
- Catmull-Rom
- Blended parabolas
- B-splines, NURBS
Hermite
Bezier
B-Spline / NURBS
Often used in animation path movement
Blended Parabolas

*Often used in animation path movement*
Interpolation

• Appendix B.4


Interpolation

- Let’s assume an interpolating technique has been chosen and that a function $P(t)$ has been selected which, for a given value of $t$, will produce a value, i.e., $p = P(t)$.

- If position is being interpolated then three functions are used in the following manner. The $x$, $y$ and $z$ coordinates for the positions at the key frames are specified.

![Diagram with points A, B, C, D with times 0, 10, 35, 60 respectively.]
Interpolation

• Key frames are associated with specific values of the time (t) parameter.

• The x, y and z coordinates are considered independently.

• For example, the points (x,t) are used as control points for the interpolating curve so that $X=P_x(t)$

• Similarly, $Y=P_y(t)$ and $Z=P_z(t)$ are formed so that at any time, t, a position (x,y,z) can be produced.
Interpolation

• Varying the parameter of interpolation, in this case $t$, by a constant amount, does not mean that the resulting values, in this case Euclidean position, will vary by a constant amount.

• This means that just because $t$ changes at a constant rate, the interpolated value will not necessarily, in fact seldom, have a constant speed.
Interpolation → Parametric Motion

• This animation shows a ball moving along a parametrically defined cubic curve:
  \[ P(t) = a \cdot t^3 + b \cdot t^2 + c \cdot t + d \]
  by uniformly varying the parameter (i.e. \( t = 0.0, 0.1, 0.2, \) etc.).

• Note that it travels further between frames at the start of the curve than at the end of the curve: **Equi-distant values in parametric space do not result in equi-distant points in Euclidean space.**
Interpolation

• In order to ensure a constant speed for the interpolated value, the interpolating function has to be parameterized by arc length, i.e., distance along the curve of interpolation.

• Some type of reparameterization by arc length should be performed for most applications.

• Usually this reparameterization can be approximated without adding undue overhead or complexity to the interpolating function.
1. Interpolation
2. **Motion along a curve (arc length)**
3. Interpolation of rotations (quaternions)
4. Path following
Reparameterization by Arc Length

• Assume that we have a spline defined by $P(t)$.

• No guarantee that the variance in $t$ is directly related to
  the distance traveled along the curve.
  – That is to say, that the distance traveled from $P(0)$ to $P(.2)$ is not
    necessarily twice as far as the distance traveled along the curve
    from $P(.2)$ to $P(.3)$.

• In order to establish this relationship we have to
  reparameterize the interpolating spline by arc length or
  some scalar of arc length.

• There are various ways to approach this.
Reparameterization by Arc Length

Analytically, arc length is defined as:

\[ L = \int_{u_1}^{u_2} \left| \frac{dP}{du} \right| du \]

We use **Gaussian quadrature** (see “Numerical Recipes”) to reduce the integral to

\[ L = \sum_{i=1}^{n} w_i f(u_i) \]

- \( n \) is the number of sample points,
- \( w_i \) are the weight values,
- \( u_i \) are the sampled values.

See Mortenson, "Geometric Modeling", pp. 299-300

The \( u \)'s can be normalized to the range zero to one and tables of weights and sample values can be found in tables (see Appendix B).
Reparameterization by Arc Length

• Most of the time, the curves that arise in computer animation applications are not analytically reparameterizable by arc length.

• They must be reparameterized numerically.

• A simple, but somewhat inaccurate, approach to this reparameterization is to precompute a table of values which relates the original parameter with an arc length parameter.
Reparameterization by Arc Length

- The number of entries in the table depends on the accuracy with which the arc length must be computed. This is determined by the application.

- The function is evaluated at \( n \) equidistant parameter values (equidistant in parameter space, e.g., \( t = 0.0, 0.01, 0.02, 0.03, \) etc.). \( N \) should be sufficiently large to ensure that the resulting arc lengths are within tolerance; this will become clear as the technique is described.

<table>
<thead>
<tr>
<th>Index</th>
<th>Parametric Entry</th>
<th>Arc Length</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>4</td>
<td>0.20</td>
<td>0.320</td>
</tr>
<tr>
<td>5</td>
<td>0.25</td>
<td>0.400</td>
</tr>
<tr>
<td>6</td>
<td>0.30</td>
<td>0.500</td>
</tr>
<tr>
<td>7</td>
<td>0.35</td>
<td>0.600</td>
</tr>
<tr>
<td>8</td>
<td>0.40</td>
<td>0.720</td>
</tr>
<tr>
<td>9</td>
<td>0.45</td>
<td>0.800</td>
</tr>
<tr>
<td>10</td>
<td>0.50</td>
<td>0.860</td>
</tr>
<tr>
<td>11</td>
<td>0.55</td>
<td>0.900</td>
</tr>
<tr>
<td>12</td>
<td>0.60</td>
<td>0.920</td>
</tr>
<tr>
<td>13</td>
<td>0.65</td>
<td>0.932</td>
</tr>
<tr>
<td>14</td>
<td>0.70</td>
<td>0.944</td>
</tr>
<tr>
<td>15</td>
<td>0.75</td>
<td>0.959</td>
</tr>
<tr>
<td>16</td>
<td>0.80</td>
<td>0.972</td>
</tr>
<tr>
<td>17</td>
<td>0.85</td>
<td>0.984</td>
</tr>
<tr>
<td>18</td>
<td>0.90</td>
<td>0.994</td>
</tr>
<tr>
<td>19</td>
<td>0.95</td>
<td>0.998</td>
</tr>
<tr>
<td>20</td>
<td>1.00</td>
<td>1.000</td>
</tr>
</tbody>
</table>

How to build this table?
\[ u = 0.00, 0.05, 0.10, 0.15, \ldots, 1.0 \]

**Table G(\( u \))**

<table>
<thead>
<tr>
<th>Index</th>
<th>Parametric Entry</th>
<th>Arc Length</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>4</td>
<td>0.20</td>
<td>0.320</td>
</tr>
<tr>
<td>5</td>
<td>0.25</td>
<td>0.400</td>
</tr>
<tr>
<td>6</td>
<td>0.30</td>
<td>0.500</td>
</tr>
<tr>
<td>7</td>
<td>0.35</td>
<td>0.600</td>
</tr>
<tr>
<td>8</td>
<td>0.40</td>
<td>0.720</td>
</tr>
<tr>
<td>9</td>
<td>0.45</td>
<td>0.800</td>
</tr>
<tr>
<td>10</td>
<td>0.50</td>
<td>0.860</td>
</tr>
<tr>
<td>11</td>
<td>0.55</td>
<td>0.900</td>
</tr>
<tr>
<td>12</td>
<td>0.60</td>
<td>0.920</td>
</tr>
<tr>
<td>13</td>
<td>0.65</td>
<td>0.932</td>
</tr>
<tr>
<td>14</td>
<td>0.70</td>
<td>0.944</td>
</tr>
<tr>
<td>15</td>
<td>0.75</td>
<td>0.959</td>
</tr>
<tr>
<td>16</td>
<td>0.80</td>
<td>0.972</td>
</tr>
<tr>
<td>17</td>
<td>0.85</td>
<td>0.984</td>
</tr>
<tr>
<td>18</td>
<td>0.90</td>
<td>0.994</td>
</tr>
<tr>
<td>19</td>
<td>0.95</td>
<td>0.998</td>
</tr>
<tr>
<td>20</td>
<td>1.00</td>
<td>1.000</td>
</tr>
</tbody>
</table>

\( G(0.0) = 0 \)
\( G(0.05) = \text{dist. between } P(0.00) \text{ and } P(0.05) \)
\( G(0.10) = G(0.05) + \text{dist. between } P(0.05) \text{ and } P(0.10) \)
\( G(0.15) = G(0.10) + \text{dist. between } P(0.10) \text{ and } P(0.15) \)
\[ \ldots \]
\( G(1.00) = G(0.95) + \text{dist. between } P(0.95) \text{ and } P(1.00) \)

Arc Length in the table is dist/max. dist.
User wants to know the distance (arc length) from the beginning of the curve \( u=0.00 \) to the point \( u=0.73 \).

\[
i = (\text{int}) \left( \frac{\text{given parametric value}}{\text{distance between entries}} \right) = (\text{int}) \left( \frac{0.73}{0.05} \right) = 14
\]

\[
L = \text{ArcLength}[i] + \frac{(\text{Given Value} - \text{Value}[i])}{(\text{Value}[i+1] - \text{Value}[i])} \cdot (\text{ArcLength}[i+1] - \text{ArcLength}[i])
\]

\[
= 0.944 + \frac{0.73 - 0.70}{0.75 - 0.70} \cdot (0.959 - 0.944)
\]

\[
= 0.953
\]
How to use this table? (2)

User wants to know the value of $u$ given the arc length; For example, $L = 0.75$, $u =$ ?

$$0.75 = 0.72 + t(0.8 - 0.72)$$

$$t = \frac{0.75 - 0.72}{0.8 - 0.72} = \frac{0.03}{0.08} = 0.375$$

$$u = 0.4 + t(0.45 - 0.4)$$

$$u = 0.4 + 0.375(0.45 - 0.4) = 0.41875$$
Reparameterization by Arc Length

- Using the parameter values and corresponding values of the function, a table is built which records accumulated linear distances between the computed points.

- Entries are then made in the table which record normalized distances along the curve.

- These entries are monotonic, increasing from zero to one, and represent the arc length parameterization (a scaled version of the arc length).

- This table can be precomputed before being used in the animation.

- This table can be used to determine the functional parameter needed to produce a point along the curve that corresponds to arc length and effectively parameterizes the function by arc length.
Arc Length Motion

- This animation shows a ball moving along a parametrically defined cubic curve: \( P(t) = a*t^3 + b*t^2 + c*t + d \)
- The parameter, \( t \), was uniformly varied from 0 to 1 generating 1000 samples.
- A table of linearly approximated normalized distances at those points was constructed.
- Then a final table of 20 points was extracted from the list to be as equally spaced as possible.
- No interpolation between samples was performed, although that would have made the final points more equally spaced. Note the ball travels pretty much the same amount between frames throughout the curve.
Parametric Motion X Arc Length Motion
Reparameterization by Arc Length

• An alternative technique is computationally more efficient but programmatically more difficult. This technique uses adaptive Gaussian quadrature to determine the value of the function at the point along the curve that corresponds to the given arc length.

Subdivision Criteria

Lengths $A+B-C$ above error tolerance

One extra subdivision can help

Lengths $A+B-C$ erroneously reports that the error is within tolerance
Controlling Motion Along a Curve

- slow
- fast
- slow
Ease-in/ease-out

Note the difference between interpolating position along a curve and the speed along which the interpolation proceeds (consider the interpolating parameter to be 'time').

For example you can do linear interpolation in space but cubic interpolation of the distance with respect to the time parameter.

At a given time, t, s(t) is the desired distance to have travelled.

Arc-length table gives corresponding parameter u for that distance.
1. \( s(t) \) should be monotonic in \( t \)
i.e. traversed without going backwards in \( t \)

2. \( s(t) \) should be continuous.
No jumps from one point to the next on the curve.

Normalizing (0-1 range) makes it easier to use in conjunction with arc length and other functions.
Sine Interpolation

a) Sine curve segment to use as ease-in/ease-out control

map parameter values [0,+1] into domain of section of curve. [-\(\pi/\), \(+\pi/2\)]

\[ s(t) = ease(t) = \sin\left( \frac{t\pi}{2} - \frac{\pi}{2} \right) + 1 \]

\[ t = 0.25 \rightarrow s(t) = 0.1465 \]
\[ t = 0.75 \rightarrow s(t) = 0.8535 \]
Sinusoidal Pieces

• Another method is to have the user specify times t1 and t2.

• A sinusoidal curve is used for velocity to implement an acceleration from time 0 to t1.

• A sinusoidal curve is also used for velocity to implement deceleration from time t2 to 1.

• Between times t1 and t2, a constant velocity is used.

• This is done by taking a parameter t in the range 0 to 1 and remapping it into that range according to the above velocity curves to get a new parameter rt.

• So as t varies uniformly from 0 to 1, rt will accelerate from 0, then maintain a constant parametric velocity and then decelerate back to 1.
Sinusoidal Pieces

- Distance (arc length)
- Linear segment
- Sinusoidal segments
Sinusoidal Pieces

\[ \frac{k_1}{\pi/2} + k_2 - k_1 + \frac{1.0 - k_2}{\pi/2} \]

**Constant Speed**

\[
\begin{align*}
\frac{1.0 - k_2}{\pi/2} \\
\frac{k_2 - k_1}{\pi/2} \\
\frac{k_1}{\pi/2}
\end{align*}
\]

\[ s \]

\[ t \]

\[ k_1 \quad k_2 \quad 1.0 \]

a) Ease-in/ease-out curve as it is initially pieced together

b) curve segments scaled into useful values with points marking segment junctions.
Sinusoidal Pieces

f is the total distance travelled to normalize the distance scale each segment by dividing by f.

\[ \text{east}(t) = \begin{cases} \left( k_1 \cdot \frac{2}{\pi} \cdot \sin \left( \left( \frac{t}{k_1} \cdot \frac{\pi}{2} - \frac{\pi}{2} \right) + 1 \right) \right) / f & t \leq k_1 \\ \left( \frac{k_1}{\pi/2} + t - k_1 \right) / f & k_1 \leq t \leq k_2 \\ \left( \frac{k_1}{\pi/2} + k_2 - k_1 + \left( 1 - k_2 \right) \cdot \frac{2}{\pi} \sin \left( \frac{t - k_2}{1.0 - k_2} \cdot \frac{\pi}{2} \right) \right) / f & k_2 \leq t \end{cases} \]

where \( f = k_1 \cdot 2/\pi + k_2 - k_1 + (1.0 - k_2) \cdot 2/\pi \)
float ease(float t, float k1, float k2)
{
    float t1,t2;
    float f,s;

    f = k1*2/3.1415926535 + k2 - k1 + (1.0-k2)

    if (t < k1) {
        s = k1*(2/PI)*(sin((t/k1)*PI/2-PI/2)+1);
    }
    else if (t < k2) {
        s = (2*k1/PI + t-k1);
    }
    else {
        s = 2*k1/PI + k2-k1 + ((1-k2)*(2/PI))*sin(((t-k2)/(1.0-k2))*PI/2);
    }

    return (s/f);
}
Constant Acceleration:

Parabolic Ease-In/Ease-Out

• An alternative approach and one that avoids the transcendental function evaluation, or corresponding table look-up and interpolation, is to establish basic assumptions about the acceleration and, from there, integrate to get the resulting interpolation function.
Constant Acceleration:

Parabolic Ease-In/Ease-Out

• The default case of no ease-in/ease-out would produce a velocity curve that is a horizontal straight line of $v_0$ as it goes from 0 to 1.

• The distance covered would be $\text{ease}(1) = v_0 \times 1$.

• To implement an ease-in/ease-out function, assume constant acceleration and deceleration at the beginning and end of the motion, and zero acceleration during the middle of the motion.
Parabolic Ease-In/Ease-Out

\[ a = acc \quad 0 < t < t1 \]
\[ a = 0.0 \quad t1 < t < t2 \]
\[ a = dec \quad t2 < t < 1.0 \]
Parabolic Ease-In/Ease-Out

- Integrate to get velocity. This produces a linear ramp for accelerating and decelerating velocities.

\[
\begin{align*}
    v &= v_0 \cdot \frac{t}{t_1} \\
    v &= v_0 \\
    v &= v_0 \left(1.0 - \frac{t-t_2}{1.0-t_2}\right)
\end{align*}
\]

- \(0.0 < t < t_1\)  
- \(t_1 < t < t_2\)  
- \(t_2 < t < 1.0\)
Constant Acceleration:

Parabolic Ease-In/Ease-Out

- More intuitive if user specifies $t_1$ and $t_2$, and the system can solve for the maximum velocity

$$v_0 = \frac{2.0}{(t_2 - t_1 + 1.0)}$$

$v = v_0 \cdot \frac{t}{t_1}$ 
$v = v_0$ 
$v = v_0 \left(1.0 - \frac{t - t_2}{t_1 - t_2}\right)$  
$0.0 < t < t_1$ 
$t_1 < t < t_2$ 
$t_2 < t < 1.0$
Constant Acceleration:

Parabolic Ease-In/Ease-Out

Distance function with parabolic sections at both ends.

\[
\begin{align*}
    d &= v_0 \cdot \frac{t^2}{2 \cdot t_1} & 0.0 < t < t_1 \\
    d &= v_0 \cdot \frac{t_1}{2} + v_0 \cdot (t - t_1) & t_1 < t < t_2 \\
    d &= v_0 \cdot \frac{t_1}{2} + v_0 \cdot (t_2 - t_1) + \left( v_0 - \frac{v_0 \cdot (t - t_2)}{1 - t_2} \right) \cdot (t - t_2) & t_2 < t < 1.0
\end{align*}
\]
Constant Acceleration:

Parabolic Ease-In/Ease-Out

- *Distance* in this case is in parametric space, or t-space, and is the distance covered by the output value of $t$.

- For a given range of $t = [0,1]$, the user specifies times to control acceleration and deceleration: $t_1$ and $t_2$.

- Acceleration occurs from time 0 to time $t_1$.

- Deceleration occurs from time $t_2$ to time 1.
Parabolic Ease-In/Ease-Out

double ease(double t, double t1, double t2)
{
    double v0, a1, a2;

    v0 = 2/(1+t2-t1); /* constant velocity attained */
    if (t<t1)
    {
        d = v0*t*t/(2*t1);
    }
    else
    {
        d = v0*t1/2;
        if (t<t2)
        {
            d += (t-t1)*v0;
        }
        else
        {
            d += (t2-t1)*vo;
            d += (t-t*t/2-t2+t2*b/2)*vo/(1-t2);
        }
    }
    return (d);
}
Ease-in/Ease-out Motion

- This animation uses the same procedure as the arc length animation with one exception.

- Before searching through the table of values to find the entry with the closest normalized distance, the parameter is passed through a constant-acceleration ease function which recomputes the value of the parameter.

- Note how, as the animation starts, the ball slowly increases its speed, maintains a constant speed, and then slows down at the end of the curve.

- Acceleration takes place until $t=0.4$; deceleration starts at $t=0.7$. 
General Distance-Time Functions

Some non-intuitive results of user-specified values on the velocity-time curve.

- User specified velocities
- Possible solution to enforce total distance covered equal to one

- User specified velocities
- Possible solution to enforce total distance covered (signed area under the curve) equal to one
Curve Fitting to Position-Time Pairs

Distance Time Functions

a) starts and ends abruptly

b) backs up

c) stalls

d) smoothly starts and stops

e) starts part way along the curve and gets to the end before t=1.0

f) waits awhile before starting and doesn’t get all the way to the end
Curve Fitting to Position-Time Pairs

Specifying motion constraints

- Distance-time constraints specified
- Resulting curve

- Velocity-distance-time constraints specified
- Resulting curve
Position-time constraints

- P1: time = 0
- P2: time = 10
- P3: time = 35
- P4: time = 50
- P5: time = 55
- P6: time = 60
Sine Acceleration/Deceleration

with Intermediate Constant Speed

• Another method is to have the user specify times $a$ and $b$, similar to the constant acceleration technique described above. However, in this case a segment of the sine curve is used to control the velocity acceleration and deceleration. Between times $a$ and $b$, a constant velocity is used.
Sine Acceleration/Deceleration with Intermediate Constant Speed

- The development of this is similar to that for constant acceleration. A velocity curve is constructed from sin segments with interior constant velocity. The constant velocity, v0, is unknown.
Sine Acceleration/Deceleration with Intermediate Constant Speed

- The distance covered can be computed as a function of $v_0$. A normalized distance of one can be used and $v_0$ can be solved for. From this the equations are integrated and equations for the distance covered can be derived (and is left as an exercise for the reader).