Introduction to Animation

Computer Graphics
CMU 15-462/15-662
Increasing the complexity of our models

...but what about motion?
First Animation

(Shahr-e Sukhteh, Iran 3200 BCE)
History of Animation

(tomb of Khnumhotep, Egypt 2400 BCE)
History of Animation

Leonardo da Vinci (1510)
History of Animation

Claude Monet, “Woman with a Parasol” (1875)
History of Animation

(Phenakistoscope, 1831)
First Film

- Originally used as scientific tool rather than for entertainment
- Critical technology that accelerated development of animation

Eadweard Muybridge, “Sallie Gardner” (1878)
First Animation on Film

Emile Cohl, “Fantasmagorie” (1908)
First Feature-Length Animation

Lotte Reiniger, “Die Abenteuer des Prinzen Achmed” (1926)
First Hand-Drawn Feature-Length Animation

Disney, “Snow White and the Seven Dwarves” (1937)
Hand-Drawn Animation - Present Day

Studio Ghibli, “Ponyo” (2008)
First Computer-Generated Animation

- New technology, also developed as a scientific tool
- Again turbo-charged the development of animation

John Whitney, “Catalog” (1961)
First Digital-Computer-Generated Animation

Ivan Sutherland, “Sketchpad” (1963)
First 3D Computer Animation

William Fetter, “Boeing Man” (1964)
Early Computer Animation

Nikolay Konstantinov, “Kitty” (1968)
Early Computer Animation

Ed Catmull & Fred Park, “Computer Animated Faces” (1972)
First Attempted CG Feature Film

First CG Feature Film

Computer Animation - Present Day

Sony Pictures Animation, “Cloudy With a Chance of Meatballs” (2009)
Zoetrope - Solid Animation
Zoetrope - 3D Printed Animation

John Edmark — BLOOMS
Perception of Motion

- Original (but debunked) theory: persistence of vision ("streaking")
- The eye is not a camera! More modern explanation:
  - beta phenomenon: visual memory in brain—not eyeball
  - phi phenomenon: brain anticipates, giving sense of motion

credit: Akiyoshi Kitaoka

beta

phi
Generating Motion (Hand-Drawn)

- Senior artist draws keyframes
- Apprentice draws inbetweens
- Tedious / labor intensive (opportunity for technology!)
How do we describe motion on a computer?
Basic Techniques in Computer Animation

- Artist-directed (e.g., keyframing)
- Data-driven (e.g., motion capture)
- Procedural (e.g., simulation)
Keyframing

- Basic idea:
  - specify important events only
  - computer fills in the rest via interpolation/approximation
- “Events” don’t have to be position
- Could be color, light intensity, camera zoom, ...

![Diagram of keyframes and interpolated frames]
How do you interpolate data?
Spline Interpolation

- Mathematical theory of interpolation arose from study of thin strips of wood or metal ("splines") under various forces

(Good summary in Levin, "The Elastica: A Mathematical History")
Interpolation

- Basic idea: “connect the dots”
- E.g., piecewise linear interpolation
- Simple, but yields rather rough motion (infinite acceleration)
Piecewise Polynomial Interpolation

- Common interpolant: piecewise polynomial “spline”

Basic motivation: get better continuity than piecewise linear!
Splines

- In general, a spline is any piecewise polynomial function.
- In 1D, spline interpolates data over the real line:

\[ (t_i, f_i), \quad i = 0, \ldots, n \]

"knots" values
\[ t_i < t_{i+1} \]

- "Interpolates" just means that the function exactly passes through those values:

\[ f(t_i) = f_i \quad \forall i \]

- The only other condition is that the function is a polynomial when restricted to any interval between knots:

\[
\text{for } t_i \leq t \leq t_{i+1}, \quad f(t) = \sum_{j=1}^{d} c_i t^j =: p_i(t)
\]
What’s so special about cubic polynomials?

- Splines most commonly used for interpolation are cubic \((d=3)\)
- Piecewise cubics give exact solution to elastic spline problem under assumption of small displacements
- More precisely: among all curves interpolating set of data points, minimizes norm of second derivative (not curvature)
- Food for thought: who cares about physical splines? We’re on a computer! And are interpolating phenomena in time
- Motivation is perhaps pragmatic: e.g., simple closed form, decent continuity
- Plenty of good reasons to choose alternatives (e.g., NURBS for exact conics, clothoid to prevent jerky motion, ...)
- Also...
Runge Phenomenon

- Tempting to use higher-degree polynomials, in order to get higher-order continuity
- Can lead to oscillation, ultimately worse approximation:
Fitting a Cubic Polynomial to Endpoints

- Consider a single cubic polynomial
  \[ p(t) = at^3 + bt^2 + ct + d \]
- Suppose we want it to match given endpoints:

Many solutions!
Cubic Polynomial - Degrees of Freedom

- Why are there so many different solutions?
- Cubic polynomial has four degrees of freedom (DOFs), namely four coefficients \((a,b,c,d)\) that we can manipulate/control
- Only need two degrees of freedom to specify endpoints:

\[ p(t) = at^3 + bt^2 + ct + d \]

\[ p(0) = p_0 \quad \Rightarrow \quad d = p_0 \]

\[ p(1) = p_1 \quad \Rightarrow \quad a + b + c + d = p_1 \]

- Overall, four unknowns but only two equations
- Not enough to uniquely determine the curve!
Fitting Cubic to Endpoints and Derivatives

What if we also match derivatives at endpoints?

\[ p(t) = at^3 + bt^2 + ct + d \]

\[ p(0) = p_0 \quad \Rightarrow d = p_0 \]
\[ p(1) = p_1 \quad \Rightarrow a + b + c + d = p_1 \]
\[ p'(0) = u_0 \quad \Rightarrow c = u_0 \]
\[ p'(1) = u_1 \quad \Rightarrow 3a + 2b + c = u_1 \]
Splines as Linear Systems

- This time, we have four equations in four unknowns
- Could also express as a matrix equation:

\[
\begin{bmatrix}
0 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 \\
3 & 2 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
a \\
b \\
c \\
d
\end{bmatrix}
= 
\begin{bmatrix}
p_0 \\
p_1 \\
u_0 \\
u_1
\end{bmatrix}
\]

- Often, this is the game we will play:
  - each condition on spline leads to a linear equality
  - hence, if we have m degrees of freedom, we need m (linearly independent!) conditions to determine spline
Natural Splines

- Now consider piecewise spline made of cubic polynomials \( p_i \)
- For each interval, want polynomial “piece” \( p_i \) to interpolate data (e.g., keyframes) at both endpoints:
  \[
p_i(t_i) = f_i, \quad p_i(t_{i+1}) = f_{i+1}, \quad i = 0, \ldots, n - 1
\]
- Want tangents to agree at endpoints (“\( C^1 \) continuity”):
  \[
p'(t_{i+1}) = p'_{i+1}(t_{i+1}), \quad i = 0, \ldots, n - 2
\]
- Also want curvature to agree at endpoints (“\( C^2 \) continuity”):
  \[
p''(t_{i+1}) = p''_{i+1}(t_{i+1}), \quad i = 0, \ldots, n - 2
\]
- How many equations do we have at this point?
  \[-2n + (n-1) + (n-1) = 4n-2\]
- Pin down remaining DOFs by setting curvature to zero at endpoints (this is what makes the curve “natural”)

Spline Desiderata

- In general, what are some properties of a “good” spline?
  - INTERPOLATION: spline passes exactly through data points
  - CONTINUITY: at least twice differentiable everywhere
  - LOCALITY: moving one control point doesn’t affect whole curve

- How does our natural spline do?
  - INTERPOLATION: yes, by construction
  - CONTINUITY: $C^2$ everywhere
  - LOCALITY: no, coefficients depend on global linear system

- Many other types of splines we can consider

- Spoiler: there is “no free lunch” with cubic splines: can’t simultaneously get all three properties
Review: Hermite/Bézier Splines

- Discussed briefly in introduction to geometry
- Each cubic “piece” specified by endpoints and tangents:

![Diagram of Hermite and Bézier splines]

- Equivalently: by four points (Bézier form); just take difference!
- Commonly used for 2D vector art (Illustrator, Inkscape, SVG, ...)
- Can we get tangent continuity?
- Sure: set both tangents to same value on both sides of knot!
  - E.g., $f_1$ above, but not $f_2$
Properties of Hermite/Bézier Spline

- More precisely, want endpoints to interpolate data:
  \[ p_i(t_i) = f_i, \quad p_i(t_{i+1}) = f_{i+1}, \quad i = 0, \ldots, n - 1 \]

- Also want tangents to interpolate some given data:
  \[ p'_i(t_i) = u_i, \quad p'_i(t_{i+1}) = u_{i+1}, \quad i = 0, \ldots, n - 1 \]

- How is this different from our natural spline’s tangent condition?

- There, tangents didn’t have to match any prescribed value—they merely had to be the same. Here, they are given.

- How many conditions overall?
  - \[ 2n + 2n = 4n \]

- What properties does this curve have?
  - INTERPOLATION and LOCALITY, but not \( C^2 \) CONTINUITY
Catmull-Rom Splines

- Sometimes makes sense to specify tangents (e.g., illustration)
- Often more convenient to just specify values
- Catmull-Rom: specialization of Hermite spline, determined by values alone
- Basic idea: use difference of neighbors to define tangent
  \[ u_i := \frac{f_{i+1} - f_{i-1}}{t_{i+1} - t_{i-1}} \]
- All the same properties as any other Hermite spline (locality, etc.)
- Commonly used to interpolate motion in computer animation.
- Many, many variants, but Catmull-Rom is usually good starting point
# Spline Desiderata, Revisited

<table>
<thead>
<tr>
<th></th>
<th>INTERPOLATION</th>
<th>CONTINUITY</th>
<th>LOCALITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>natural</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Hermite</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>???</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>
B-Splines

- Get better continuity and local control by sacrificing interpolation.

- B-spline basis defined recursively:

  \[
  B_{i,1}(t) := \begin{cases} 
  1, & \text{if } t_i \leq t < t_{i+1} \\
  0, & \text{otherwise}
  \end{cases}
  \]

  \[
  B_{i,k}(t) := \frac{t-t_i}{t_{i+k-1}-t_i} B_{i,k-1}(t) + \frac{t_{i+k}-t}{t_{i+k}-t_{i+1}} B_{i+1,k-1}(t)
  \]

- B-spline itself is then a linear combination of bases:

  \[
  f(t) := \sum_i a_i B_{i,d}
  \]
## Spline Desiderata, Revisited

<table>
<thead>
<tr>
<th></th>
<th>INTERPOLATION</th>
<th>CONTINUITY</th>
<th>LOCALITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>natural</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Hermite</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>B-splines</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>
Ok, I get it: splines are great.
But what exactly are we interpolating?
Simple example: camera path

- Animate position, direction, “up” direction of camera
  - each path is a function $f(t) = (x(t), y(t), z(t))$
  - each component $(x,y,z)$ is a spline
Character Animation

- Scene graph/kinematic chain: scene as tree of transformations
- E.g. in our “cube man,” configuration of a leg might be expressed as rotation relative to body
- Animate by interpolating transformations
- Often have sophisticated “rig”:

Even w/ computer “tweening,” a lot of work to animate!
Inverse Kinematics

- Important technique in animation & robotics
- Rather than adjust individual transformations, set “goal” and use algorithm to come up with plausible motion:

Many algorithms—basic idea: numerical optimization/descent
Skeletal Animation

- Previous characters looked a lot different from “cube man”!
- Often use “skeleton” to drive deformation of continuous surface
- Influence of each bone determined by, e.g., weighting function:

(Many other possibilities—still active area of R&D)
Blend Shapes

- Instead of skeleton, interpolate directly between surfaces
- E.g., model a collection of facial expressions:

Simplest scheme: take linear combination of vertex positions
- Spline used to control choice of weights over time
Coming up next...

- Even with “computer-aided tweening,” animating everything by hand takes a lot of work!
- Will see how data, physical simulation can help