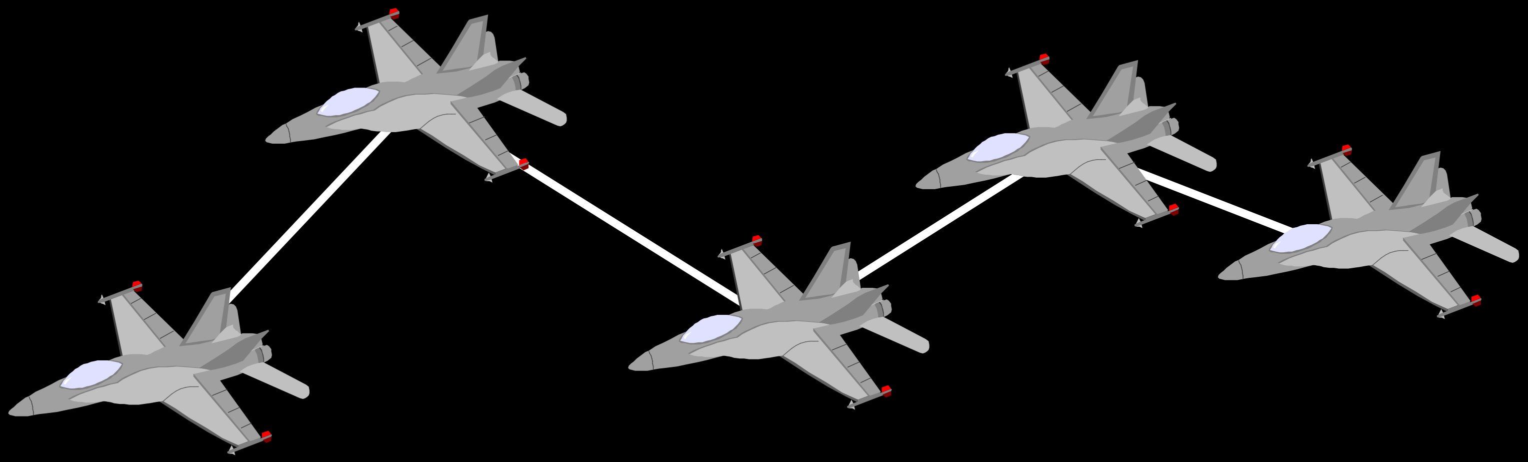
Kyeframing by Interpolation



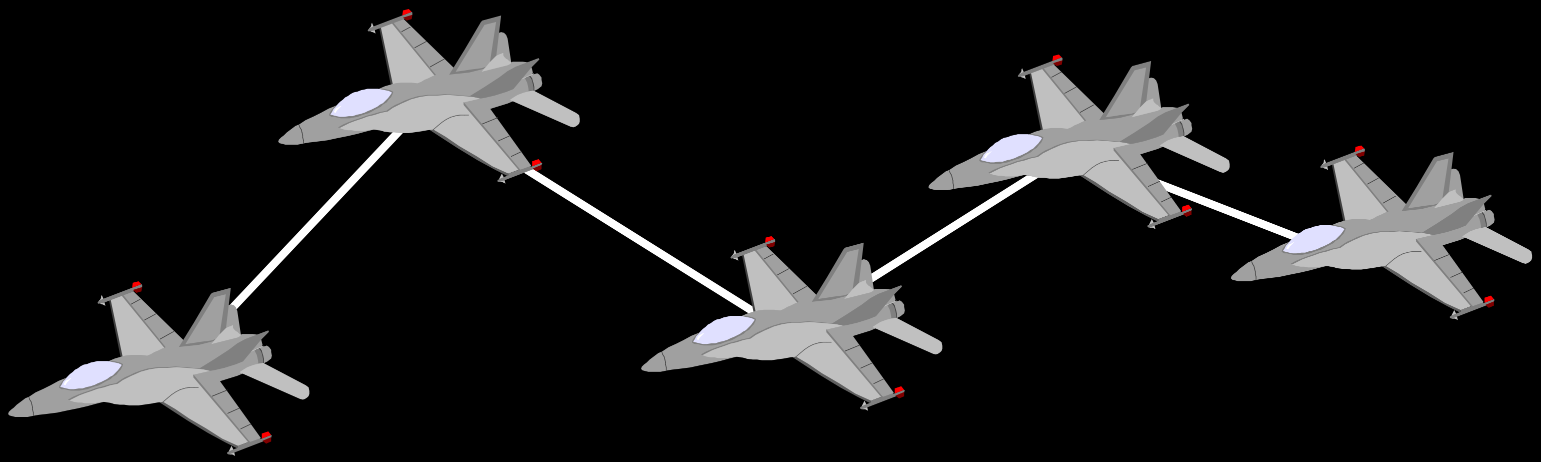
Wen-Chieh (Steve) Lin

National Chiao-Tung University

Parent, Computer Animation, Chapter 3, Appendix B

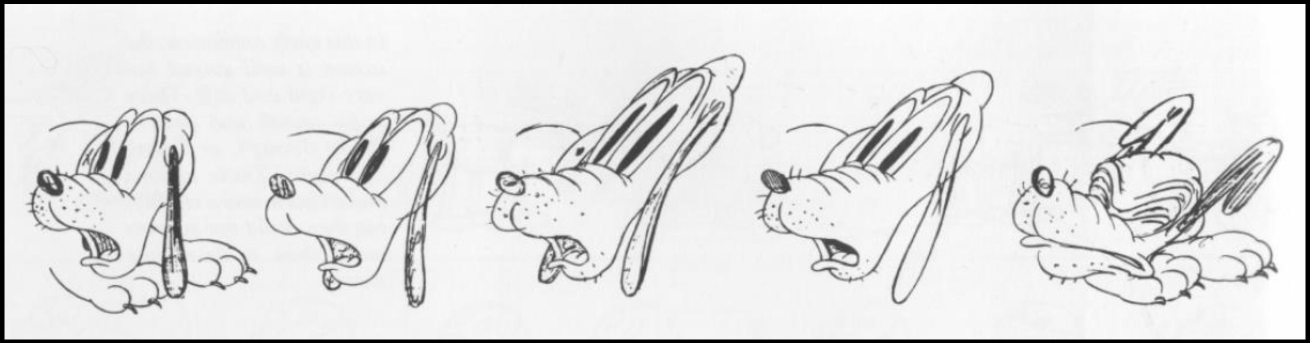
Keyframing

- Animator specifies the important key frames
- Computer generates the in-betweens automatically using interpolation



What is the key?

- Difficult to interpolate hand-drawn images



- Different approach in computer animation
  - Each keyframe is described by a set of parameters
  - Sequence of keyframes = points in high-dimensional space
- Compute inbetweenings by interpolating these points

Example: Keys in Pixar Characters

Keyframing Procedures

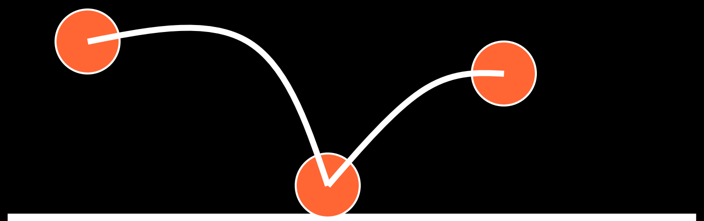
- Specify the key frames
  – rigid transformation, forward/inverse kinematics
- Specify the type of interpolation
  – linear, cubic, parametric curves
- Specify the speed profile of the interpolation
  – constant velocity, ease-in/out, etc.
- Computer generates the inbetween frames

Keyframe Animation: Pros and Cons

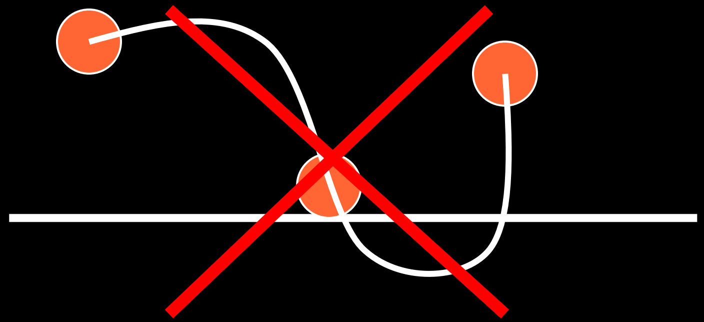
- Good control over motion
- Eliminates much of the labor in traditional animation, but still very labor-intensive
- Impractical for complex scenes
  - water, smoke
  - grass in the wind
  - crowds

Interpolation

- How to interpolate between key frames?



desired result

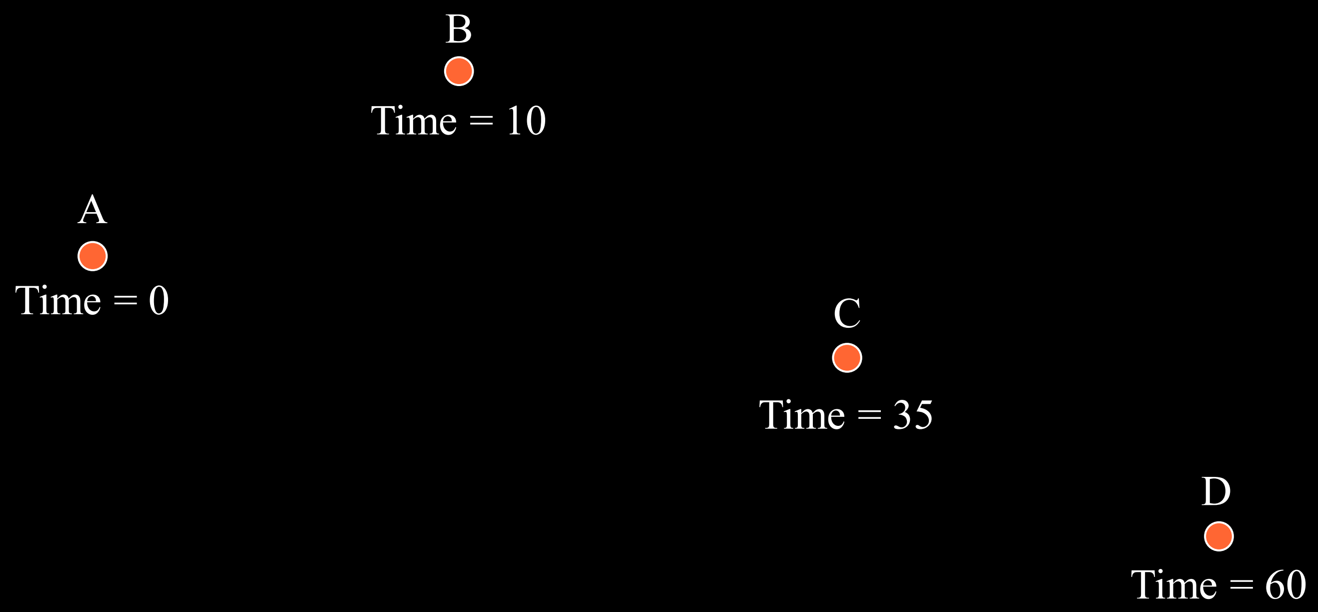


undesired result

- Need a smooth interpolation with user control

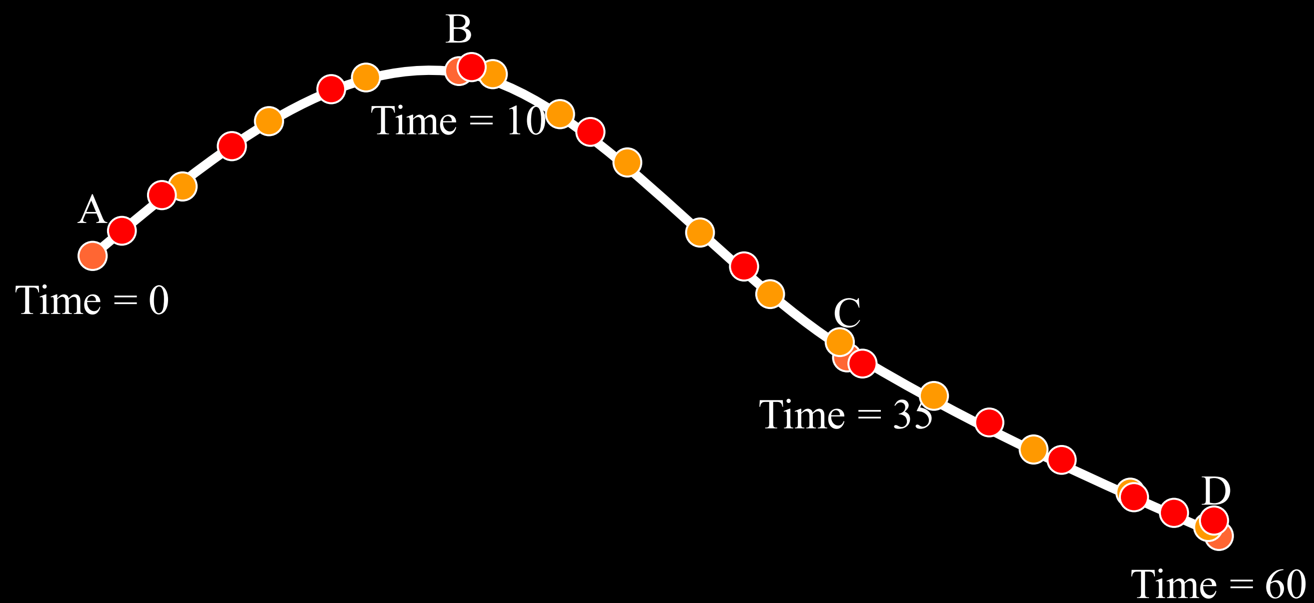
Problem

- Generate a path through points at designated times with smooth motion



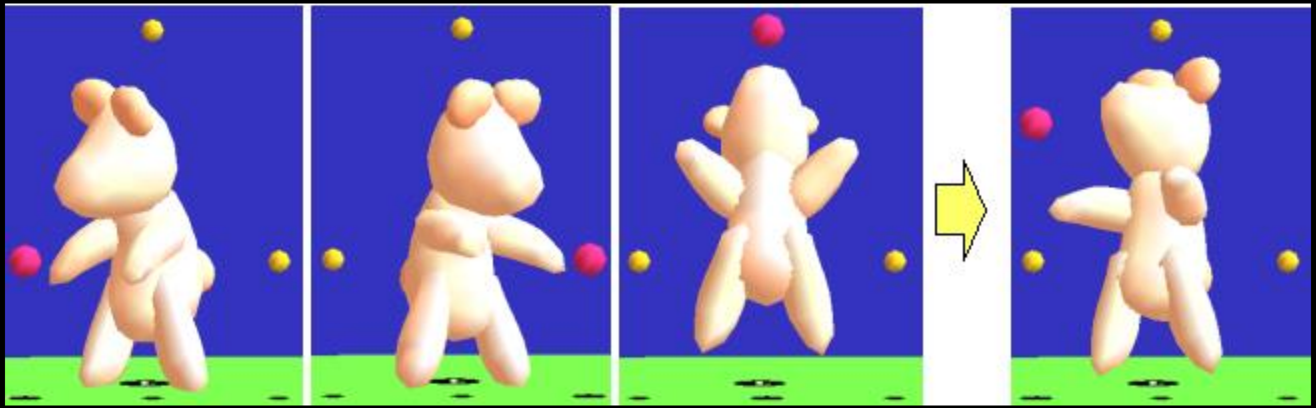
Solution

- Generate a space curve
- Distribute points evenly along curve
- Speed control: vary points temporally



Example: Spatial Keyframing

- Takeo Igarashi, Tomer Moscovich, John F. Hughes, “Spatial Keyframing for Performance-driven Animation,” SCA 2005



- Associate a key pose with a 3D position
- Interpolate in pose space
- Video

Review: Interpolating and Approximating Curves

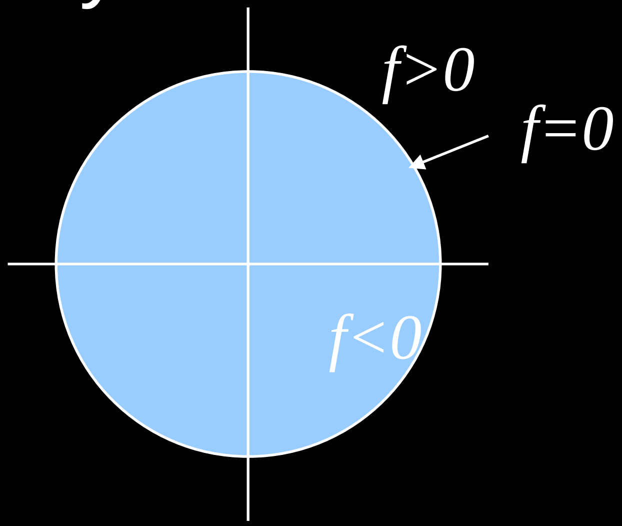
- Curve representation
- Basic techniques in interpolation and approximation

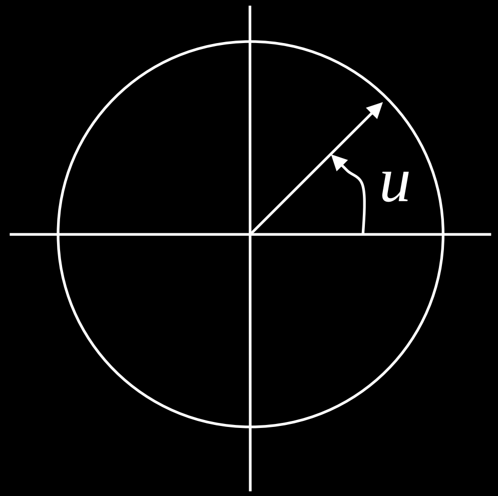
Types of Curve Representation

- Explicit $y = f(x)$
  - Good for generate points
  - For each input, there is a unique output
- Implicit $f(x,y) = 0$
  - Good for testing if a point is on a curve
  - Bad for generating a sequence of points
- Parametric $x = f(\mathrm{u}),\ y = g(\mathrm{u})$
  - Good for generating a sequence of points
  - Can be used for multi-valued function of x

Example: Representing Unit Circle

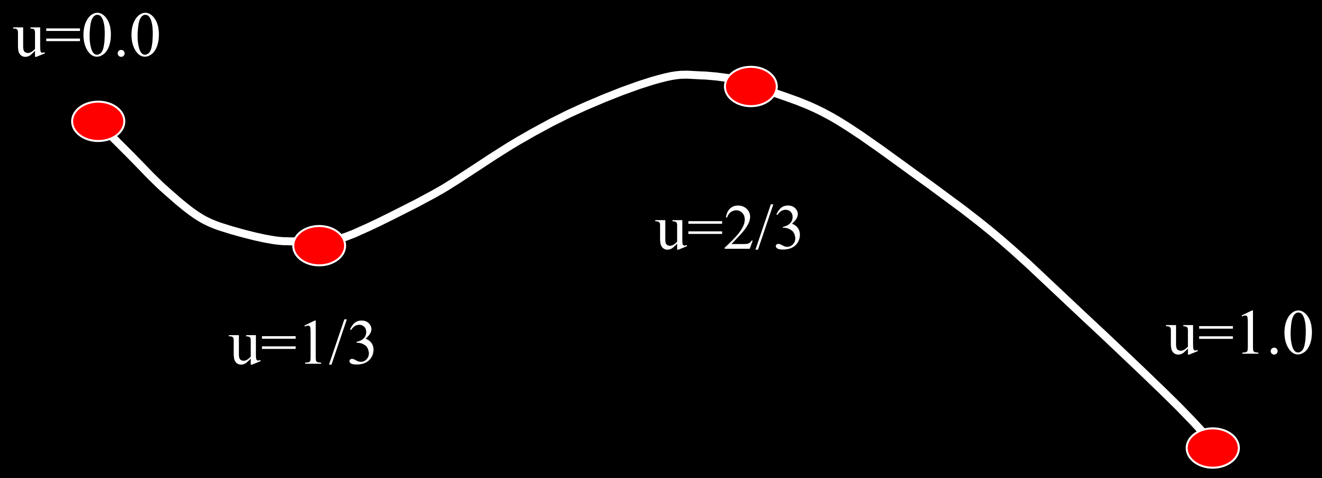
- Cannot be represented explicitly as a function of *x*
- Implicit form:
  $f(x,y)=x^2+y^2=1$
- Parametric form:
  $x=cos(u)$, $y=sin(u)$, $0<u<2\pi$





3-D Space Curve Parameterization

- Parametric form: $P = P(u) = (x,y,z)$
- $x = P_x(u)$, $y = P_y(u)$, $z = P_z(u)$



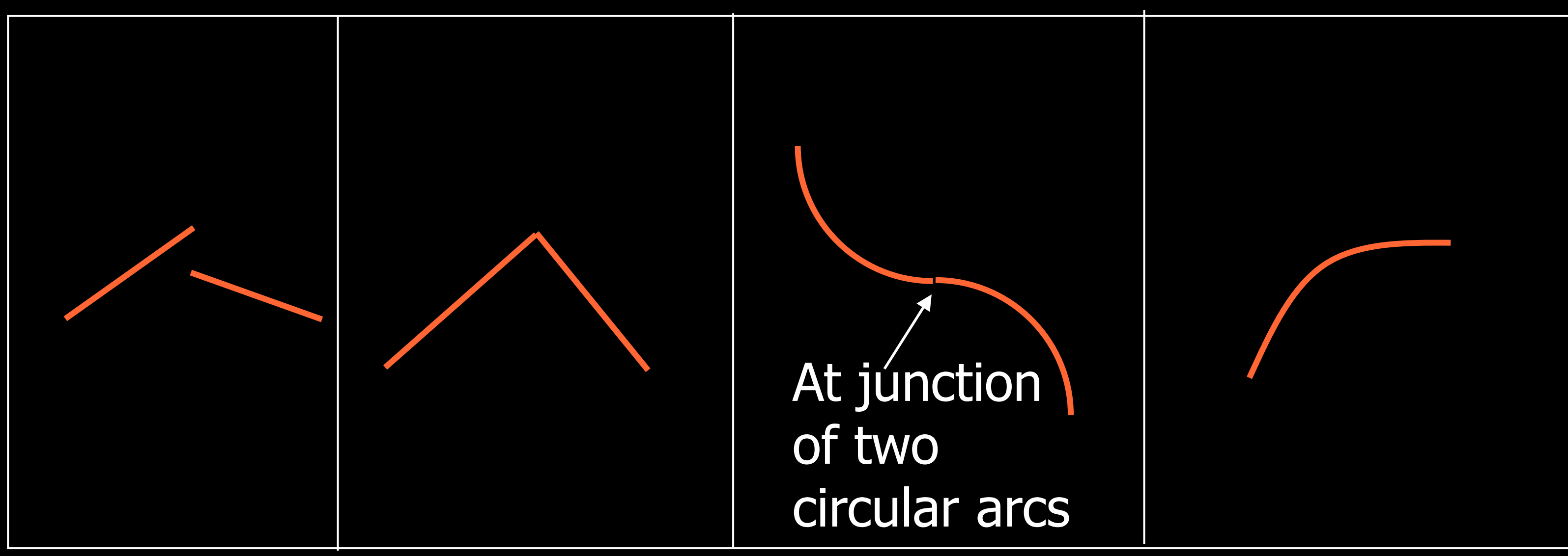
Space-curve $P = P(u)$ $0.0 <= u <= 1.0$

Appropriate Space Curve

- Interpolation vs. Approximation
  - match data vs. design
- Complexity
  - tradeoff between efficiency and flexibility
  - cubic polynomial is sufficient in general
- Continuity
- Global vs. Local control

Continuity

none | position (C0, 0th order) | tangent (C1, 1st order) | curvature (C2, 2nd order)



Global vs. Local Control

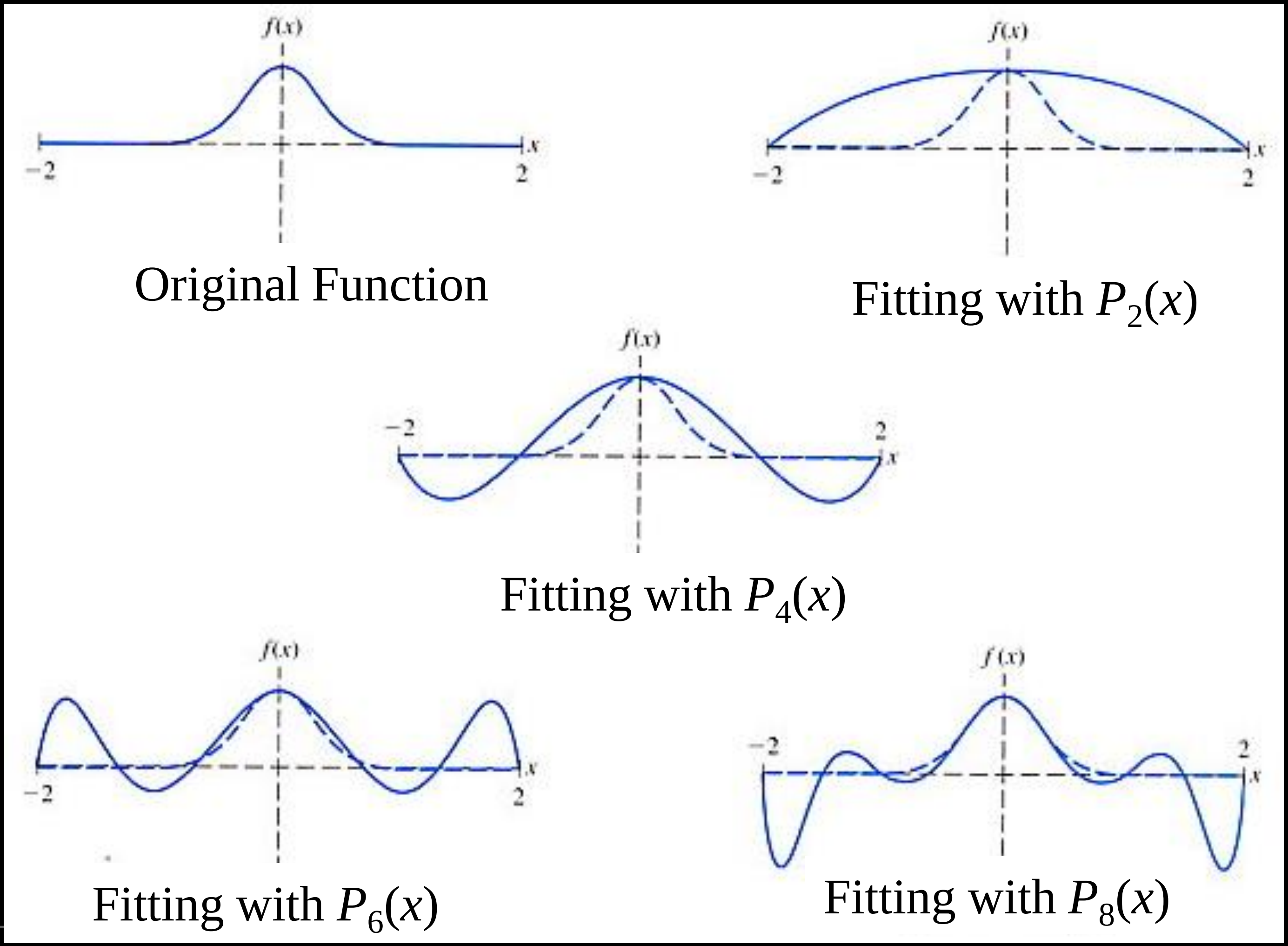
- Does a small change affect the whole curve or just a small segment?
- Local control is usually more intuitive and provided by composite curves

Global Control: Polynomial Interpolation

- An n-th degree polynomial fits a curve to $n+1$ points
  - Example: fit a second-degree curve to three points
    - $x(u)= au^2 + bu + c$
    - control points to interpolate:
      $(u_1, x_1), (u_2, x_2), (u_3, x_3)$
    - solve for coefficients (a, b, c):
      3 linear equations, 3 unknowns
- Called Lagrange Interpolation

Interpolating Data with a High-Degree Polynomial is Bad!

- Often causes undesirable wiggling in a flat region!



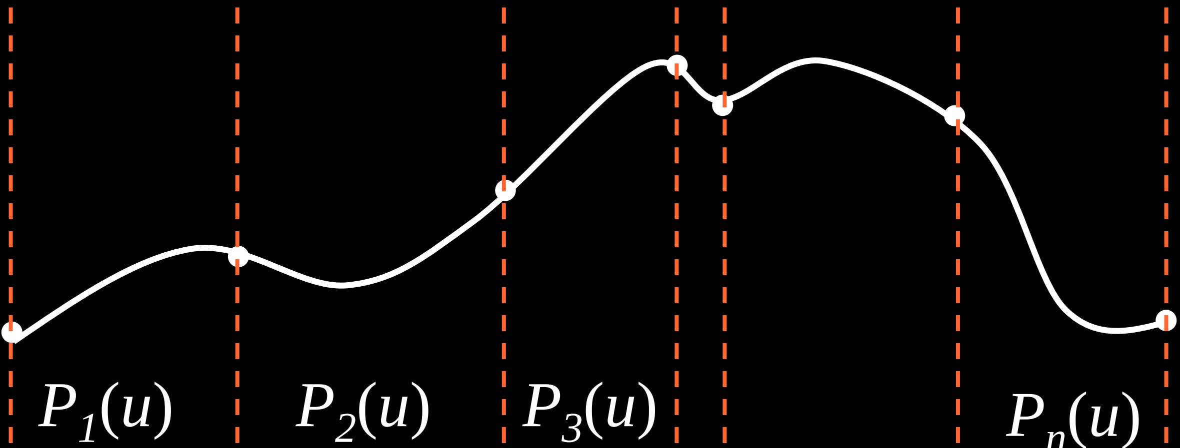
Polynomial Interpolation (cont.)

- 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

Local Control: Composite Segments

- Divide a curve into multiple segments
- Represent each in a parametric form
- Maintain continuity between segments
  - position
  - tangent
  - curvature



Splines: Piecewise Polynomials

- A spline is a piecewise polynomial
  - many low degree polynomials are used to interpolate (pass through) the control points
- A springy wire minimizes its bending energy (subject to constraints)
- Bending energy approximated by the integral of squared curvature
  - minimize this and the curve looks real
  - 2nd derivative approximates curvature

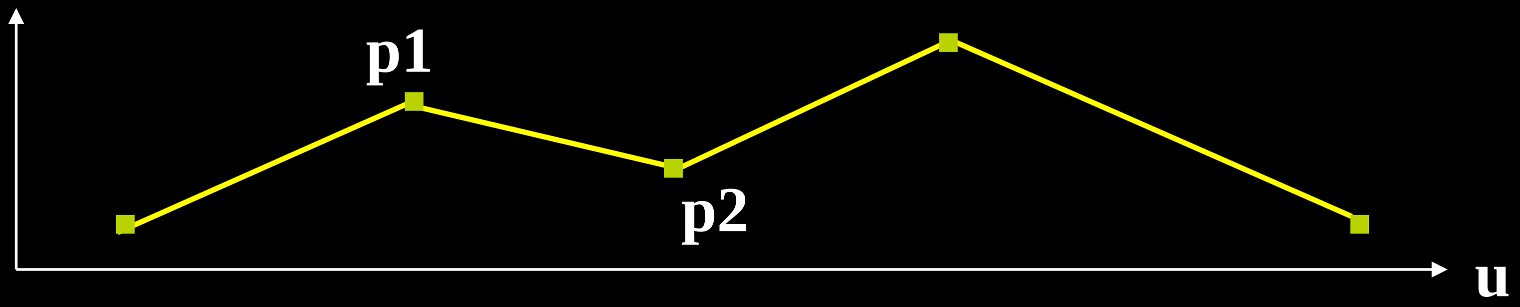
Splines: Piecewise Polynomials (cont.)

- Intuitively: try to make curvature zero everywhere
  - If you can't, distribute bends as uniformly as possible

- Cubic polynomials are the most common:
  - lowest order polynomials that interpolate two points and allow the gradient at each point to be defined - C1 continuity is possible
  - Higher or lower degrees are possible, of course

A Linear Piecewise Polynomial

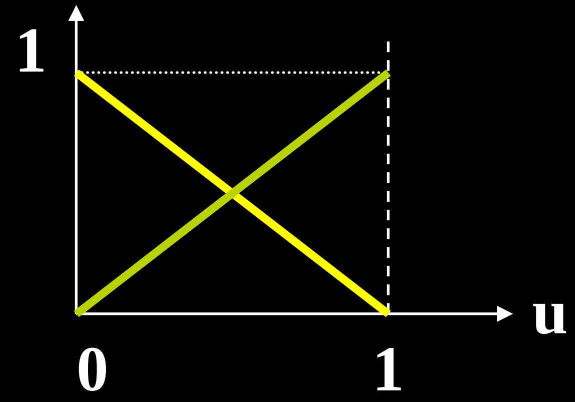
A simple example:



Each segment is of the form: ***(this is a vector equation)***

$$P(u) = (1-u)p_1 + up_2$$

Two basis (blending) functions



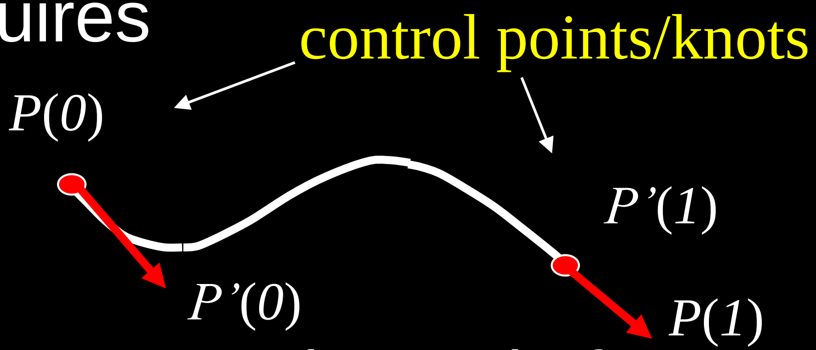
Hermite Interpolation

- Hermite Curves—cubic polynomial

$$P(u) = (P_x(u), P_y(u), P_z(u))$$

$$P_x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$$

- Really represents 3 equations in 3-D space
- Hermite interpolation requires
  - endpoints
  - derivatives at endpoints



- To create a composite curve, use the end of one as the beginning of the other and share the tangent vector

Hermite Curve Formation

- Cubic polynomial and its derivative

$$P_x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$$

$$P_x^{'}(u) = 3a_x u^2 + 2b_x u + c_x$$

- Given $P_x(0)$, $P_x(1)$, $P'_x(0)$, $P'_x(1)$, solve for *a, b, c, d*
  - 4 equations are given for 4 unknowns

$$P_x(0) = d_x$$
$$P_x(1) = a_x + b_x + c_x + d_x$$
$$P_x^{'}(0) = c_x$$
$$P_x^{'}(1) = 3a_x + 2b_x + c_x$$

Hermite Curve Formation (cont.)

- Problem: solve for *a, b, c, d*

$$P_x(0) = d_x$$
$$P_x(1) = a_x + b_x + c_x + d_x$$
$$P_x^{'}(0) = c_x$$
$$P_x^{'}(1) = 3a_x + 2b_x + c_x$$

- Solution:

$$a_x = 2(P_x(0) - P_x(1)) + P_x^{'}(0) + P_x^{'}(1)$$
$$b_x = 3(P_x(1) - P_x(0)) - 2P_x^{'}(0) - P_x^{'}(1)$$
$$c_x = P_x^{'}(0)$$
$$d_x = P_x(0)$$

Hermite Curve Formation (cont.)

- *x* component of Hermite curve can be represented as

$$P_x(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} a_x \\ b_x \\ c_x \\ d_x \end{bmatrix}$$

$$= \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_x(0) \\ P_x(1) \\ P'_x(0) \\ P'_x(1) \end{bmatrix}$$

Parametric Curves in Matrix Form

$$P(u) = au^3 + bu^2 + cu + d$$

$$P(u) = U^T MB$$

$U^T = [u^3, u^2, u, 1]$ is the parameter

M is the cofficient matrix

B is the geometric information

$$\mathrm{M} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad \mathrm{B} = \begin{bmatrix} P_x(0) & P_y(0) & P_z(0) \\ P_x(1) & P_y(1) & P_z(1) \\ P_x'(0) & P_y'(0) & P_z'(0) \\ P_x'(1) & P_y'(1) & P_z'(1) \end{bmatrix} \xrightarrow{\text{ith segment in composite curves}} \mathrm{B} = \begin{bmatrix} p_i \\ p_{i+1} \\ p'_i \\ p'_{i+1} \end{bmatrix}$$

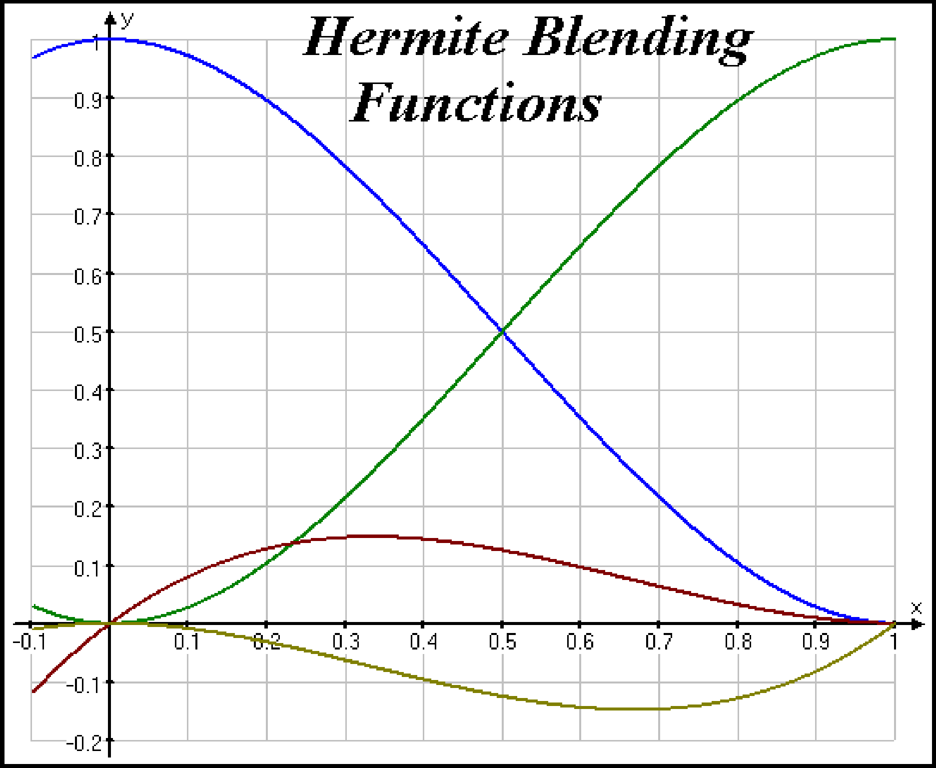
Blending Functions of Hermite Splines

- Each cubic Hermite spline is a linear combination of 4 blending functions

geometric information

$$P(u) = \boxed{U^T M} B$$

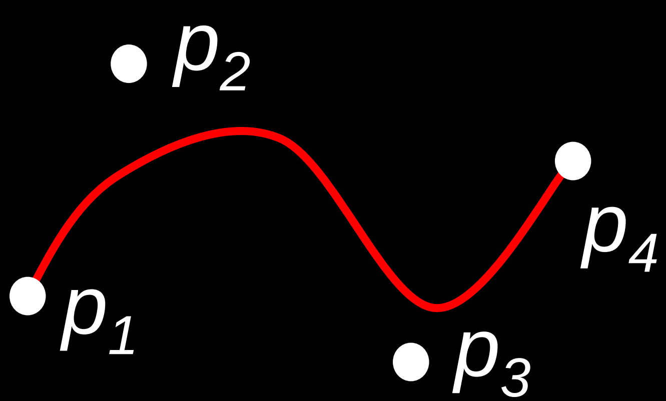
$$P(u) = \begin{bmatrix} 2u^3 - 3u^2 + 1 \\ -2u^3 + 3u^2 \\ u^3 - 2u^2 + u \\ u^3 - u^2 \end{bmatrix}^T \begin{bmatrix} p_1 \\ p_2 \\ p'_1 \\ p'_2 \end{bmatrix}$$



$$P_x(u) = (2u^3 - 3u^2 + 1)p_{1x} + (-2u^3 + 3u^2)p_{2x} + (u^3 - 2u^2 + u)p'_{1x} + (u^3 - u^2)p'_{2x}$$

Bezier Curves

- Another variant of the same game
- Instead of endpoints and tangents, four control points
  - points p_1 and p_4 are on the curve
  - points p_2 and p_3 are off the curve
  - $P(0) = p_1$, $P(1) = p_4$,
  - $P'(0) = 3(p_2 - p_1)$, $P'(1) = 3(p_4 - p_3)$

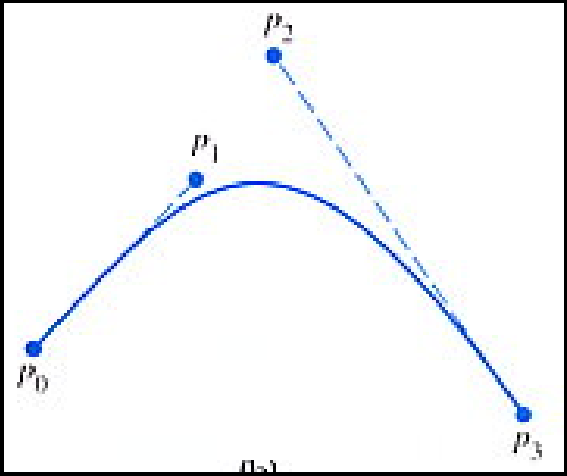


$$P(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}$$

Bezier Curves (cont.)

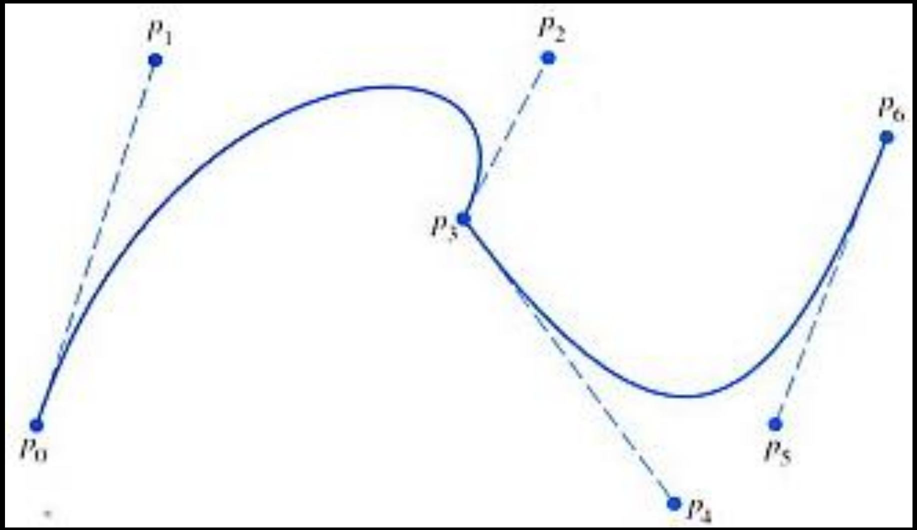
- Variant of the Hermite spline
  - basis matrix derived from the Hermite basis
- Gives more uniform control knobs (series of points) than Hermite
- The slope at u = 0 is the slope of the secant line between p_0 and p_1

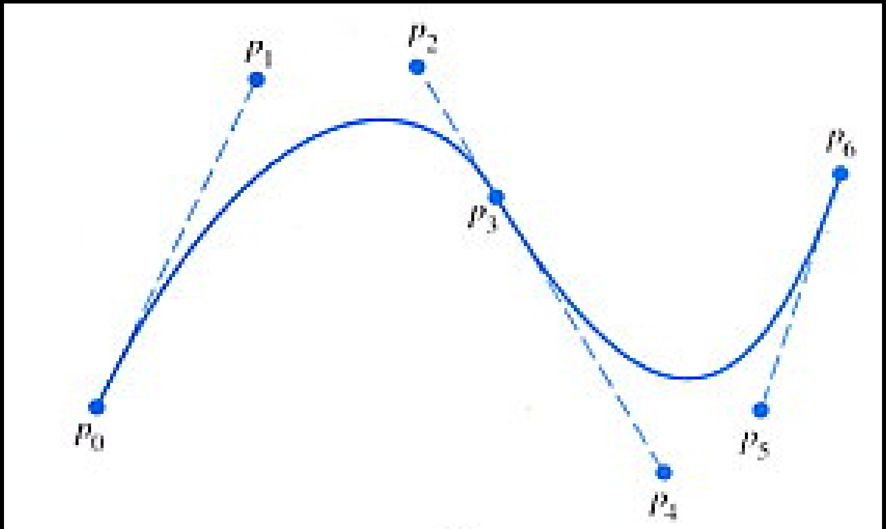
$$P'(0) = 3(p_1 - p_0)$$
$$dx/du = 3(x_1 - x_0)$$
$$dy/du = 3(y_1 - y_0)$$
$$dy/dx = (y_1 - y_0)/(x_1 - x_0)$$



Composing Bezier Curves

- Control points at consecutive segments need be collinear to avoid a discontinuity of slope





Catmull-Rom Splines

- 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 (cont.)

- 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..n-1, for some *s*
  - Curve between p_i and p_{i+1} is determined by p_{i-1}, p_i, p_{i+1}, p_{i+2}

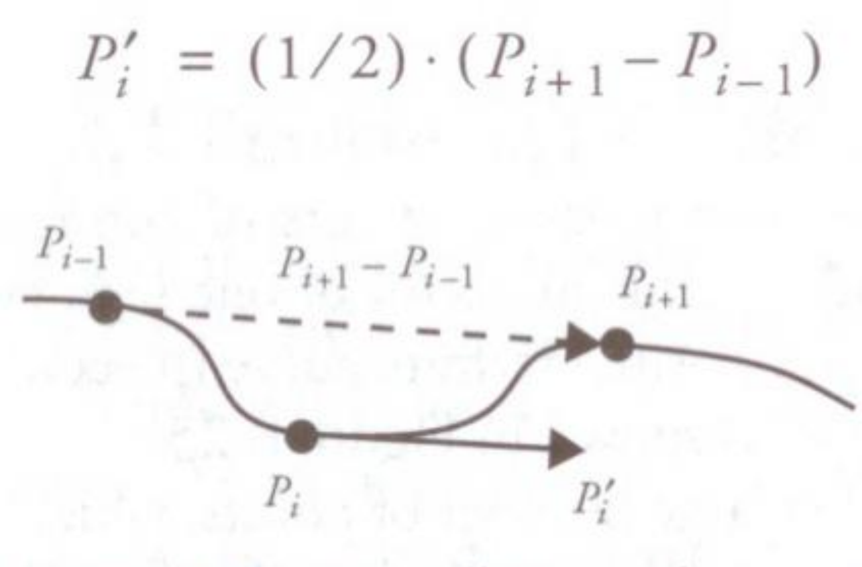


Figure B.26 Catmull-Rom spline

Catmull-Rom Splines (cont.)

- 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..n-1, for some *s*
  - Curve between p_i and p_{i+1} is determined by p_{i-1}, p_i, p_{i+1}, p_{i+2}
  - 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.

Catmull-Rom Spline Matrix

- Derived similarly to Hermite and Bezier
- s is tension parameter; typically s=1/2

$$P(u) = U^T \begin{bmatrix} -s & 2-s & s-2 & s \\ 2s & s-3 & 3-2s & -s \\ -s & 0 & s & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}$$

Catmull-Rom Spline Matrix Derivation

- P(0) = p_2, P'(0) = $s(p_3 - p_1)$,
- p(1) = p_3, P'(1) = $s(p_4 - p_2)$

$$P(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P(0) \\ P(1) \\ P'(0) \\ P'(1) \end{bmatrix}$$

$$= U^T \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s & 0 & s & 0 \\ 0 & -s & 0 & s \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}$$

Splines and Other Interpolation Forms

- See Computer Graphics textbooks
- Review
  - Appendix B.4 in Parent

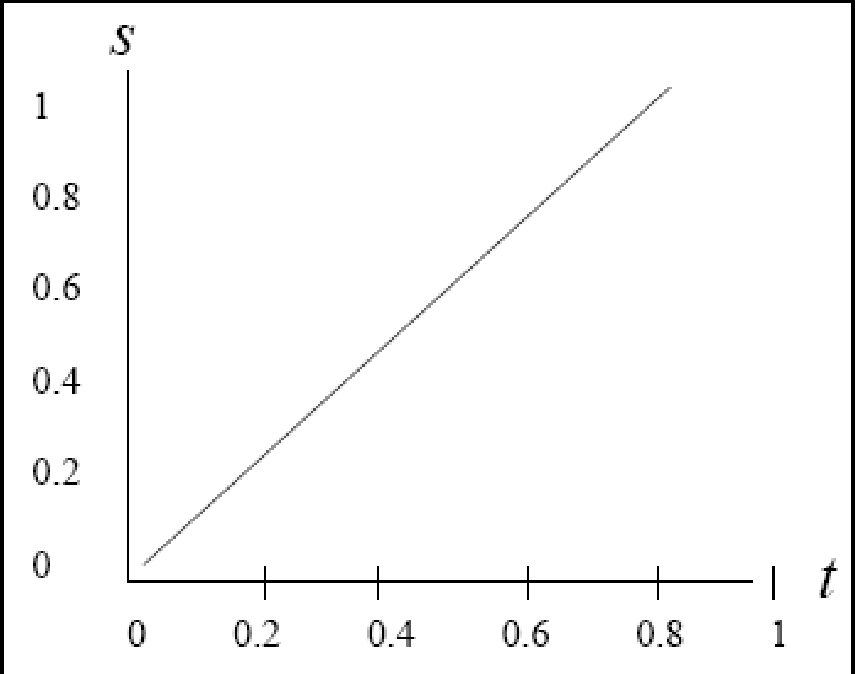
Now What?

- We have key frames or points
- We have a way to specify the space curve
- Now we need to specify velocity to traverse the curve

Speed Curves

Speed Control

- Maps parameters such as time to arc length
- Simplest form is constant velocity along the path

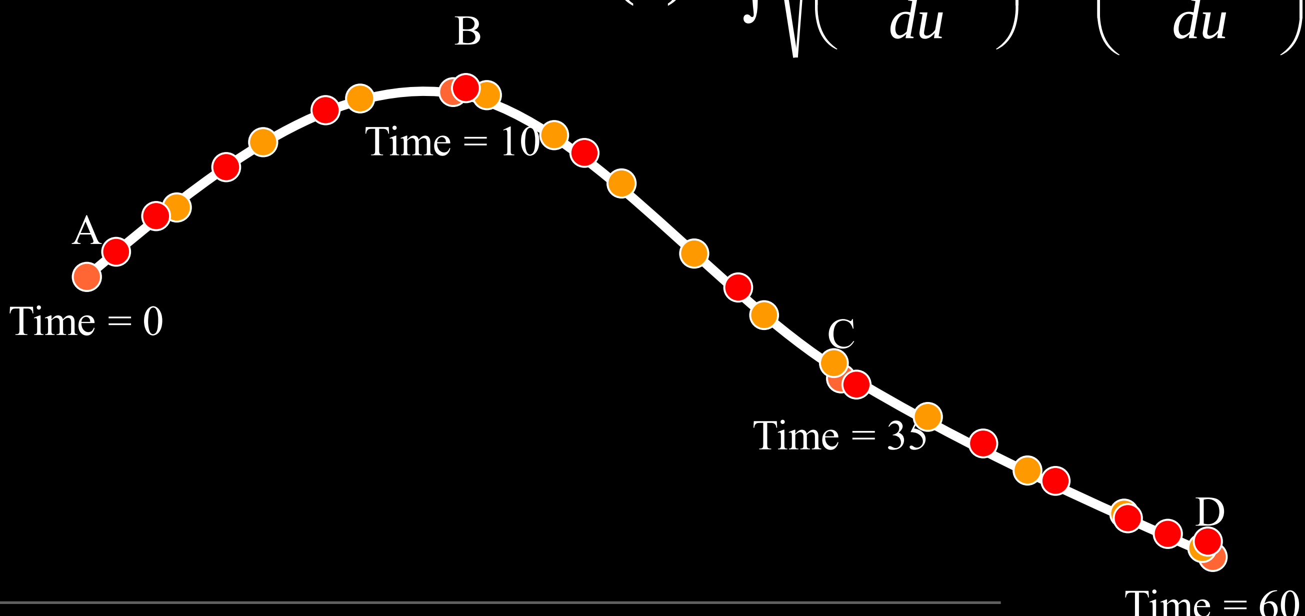


s is the arc length along the space curve; *t* is time

Non-uniformity in Parametrization

- Generally, equally spaced samples in parameter space are not equally spaced along the curve $u_2 - u_1 \neq s(u_2) - s(u_1)$

$$s(u) = \int \sqrt{\left(\frac{dP_x(u)}{du}\right)^2 + \left(\frac{dP_y(u)}{du}\right)^2 + \left(\frac{dP_z(u)}{du}\right)^2}\, du$$



Arc Length Reparameterization

- Reparametrize the space curve by arc length
- Problem
  - Given a parametric curve and two parameter values u_1 and u_2, find $arclength(u_1,u_2)$
  - Given an arc length s, and parameter value u_1, find u_2 such that $arclength(u_1,u_2) = s$
- Not possible analytically for most curves, e.g., B-splines

Finite Differences

- Sample the curve at small intervals of the parameter
- Compute the distance between samples
- Build a table of arc length for the curve

u	Arc Length
0.0	0.00
0.1	0.08
0.2	0.19
0.3	0.32
0.4	0.45
…	…

Arc Length Reparameterization Using Lookup Table

- Given an arc length, find the parametric value
  - Find the entry in the table closest to this *u*
  - Or take the u before and after it and interpolate linearly

u	Arc Length
0.0	0.00
0.1	0.08
0.2	0.19
0.3	0.32
0.4	0.45
…	…

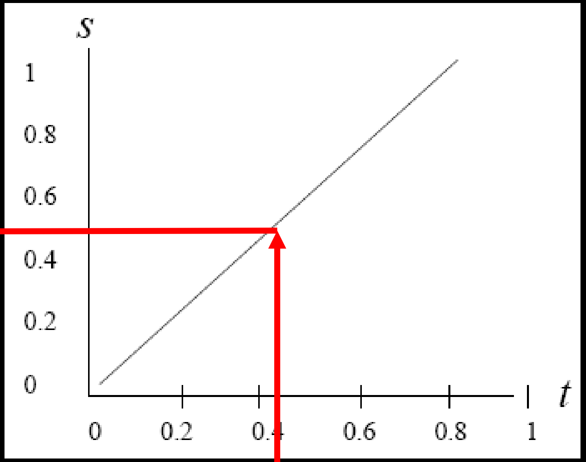
Speed Control

Arc Length Table

u	Arc Length
0.0	0.00
0.1	0.08
0.2	0.19
0.3	0.32
0.4	0.45
…	…

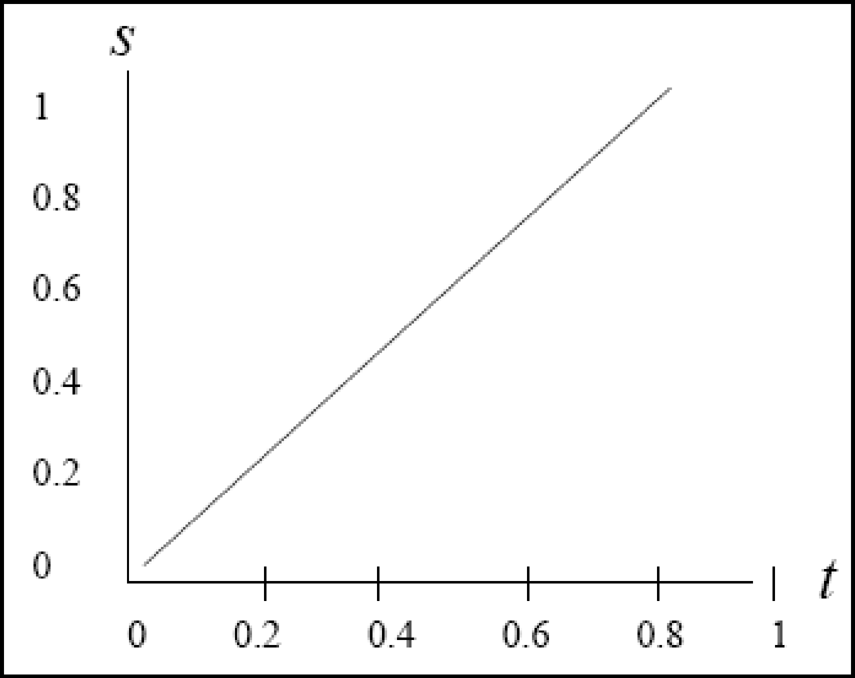
$P(u^*)$ $u*$ $s*$

Speed Curve



$t*$

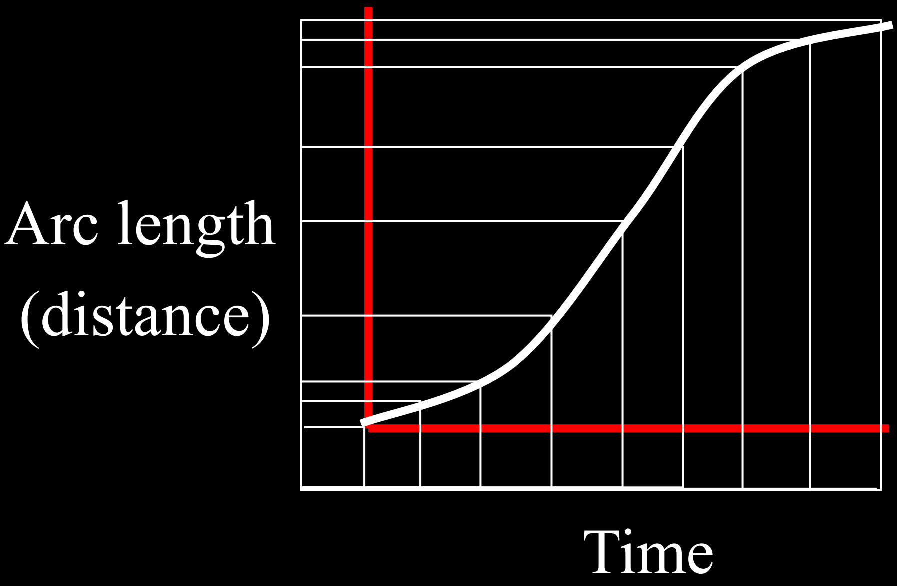
Constant Speed Curve



- Moving at 1 m/s if meters and seconds are the units
- Too simple to be what we want

Ease-in Ease-out Curve

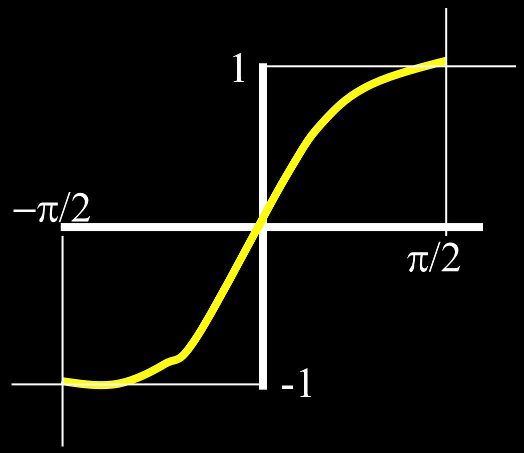
- Assume that the motion starts and stops at the beginning and end of the motion curve



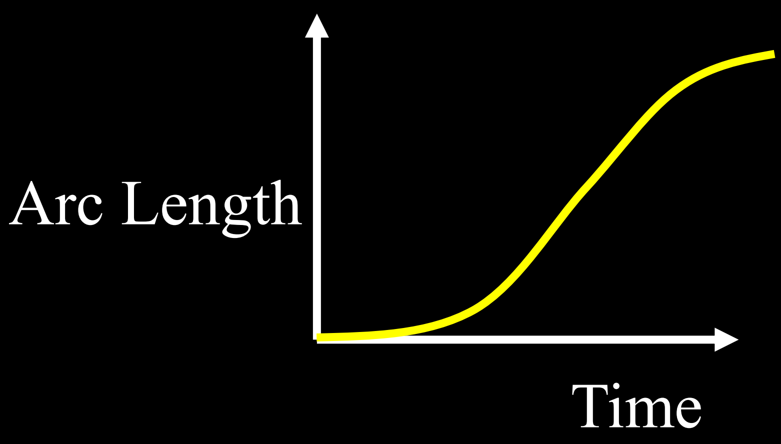
Equally spaced samples in time specify arc length required for that frame

Sine Interpolation

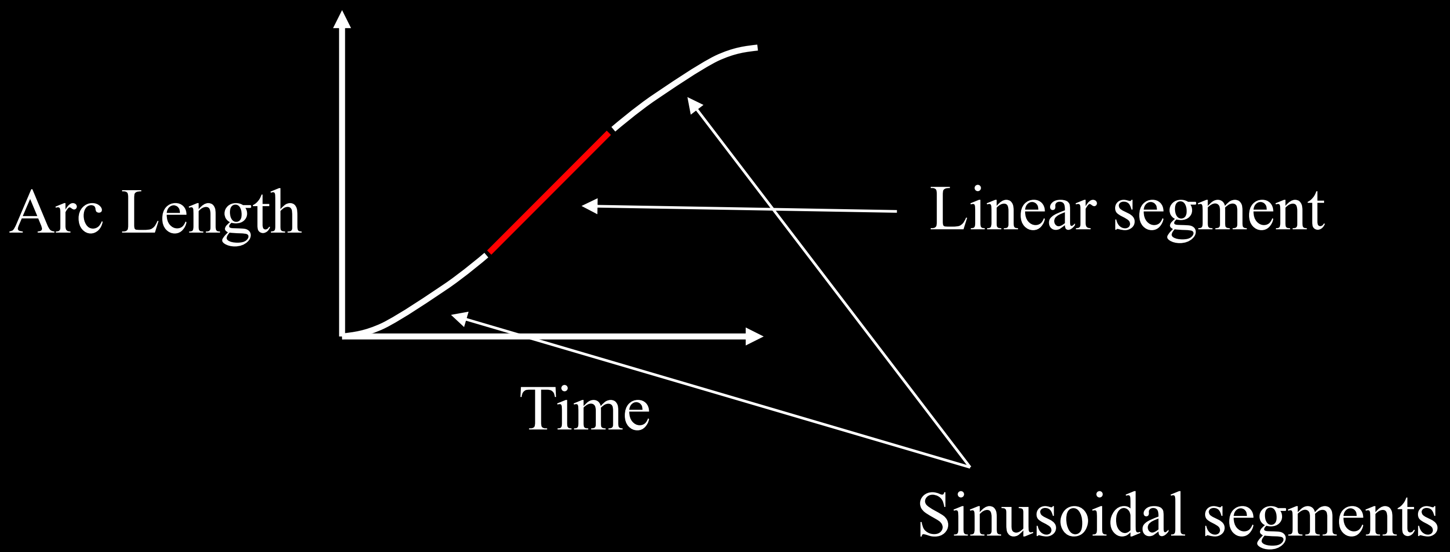
$$s(t) = \frac{1}{2}\sin(t\pi - \frac{\pi}{2}) + \frac{1}{2}, \qquad 0 \le t \le 1$$



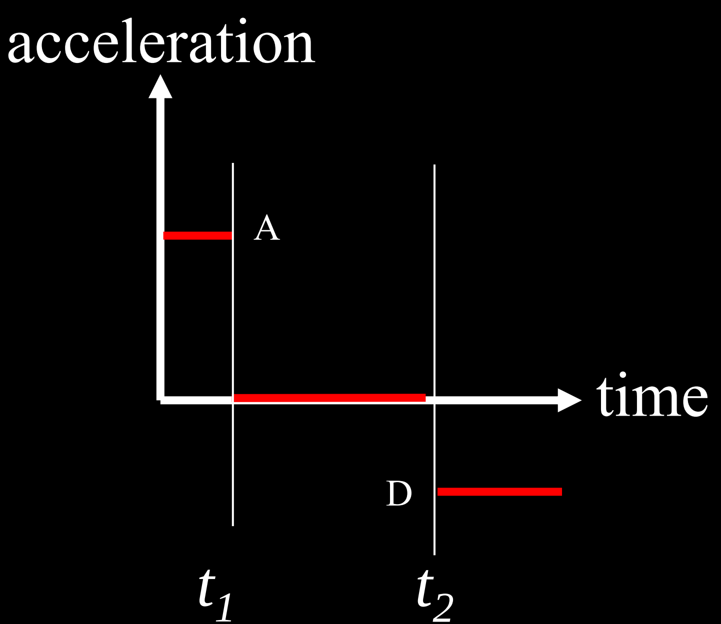
$$\sin(\alpha), \quad -\frac{\pi}{2} \le \alpha \le \frac{\pi}{2}$$

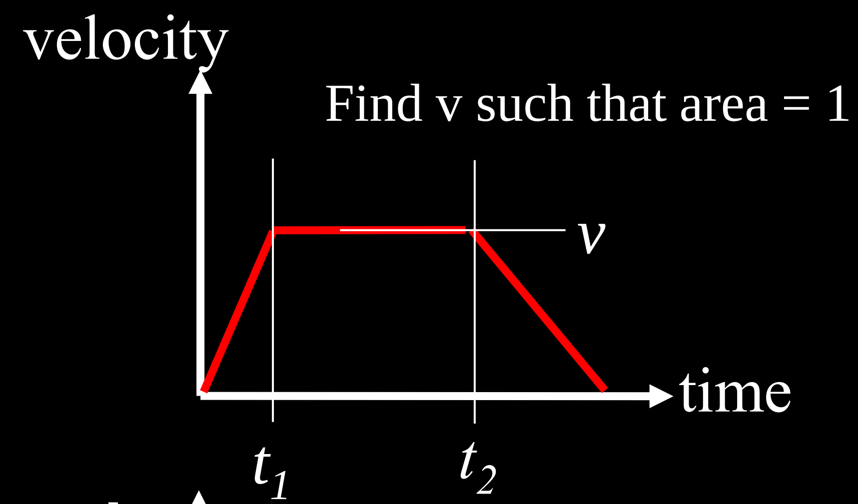


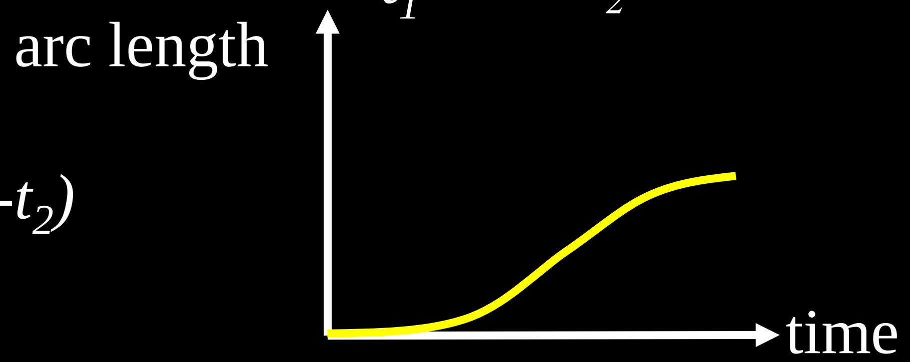
Piecing Curves Together for Ease In/Out



Integrating to avoid the sine function







$$V_1 = V_2 \rightarrow A*t_1 = - D*(1-t_2)$$