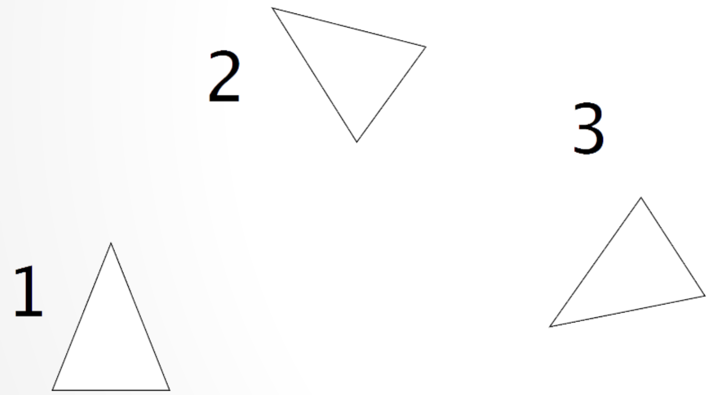
Computer Animation and Games I

CM50244

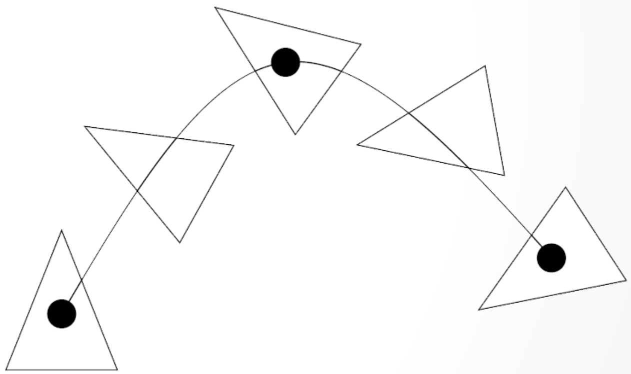
Interpolation

Recall Keyframe Animation

- Animator draws character at “extreme” poses
- Fill in in-betweens



Keyframes



Animation

Overview

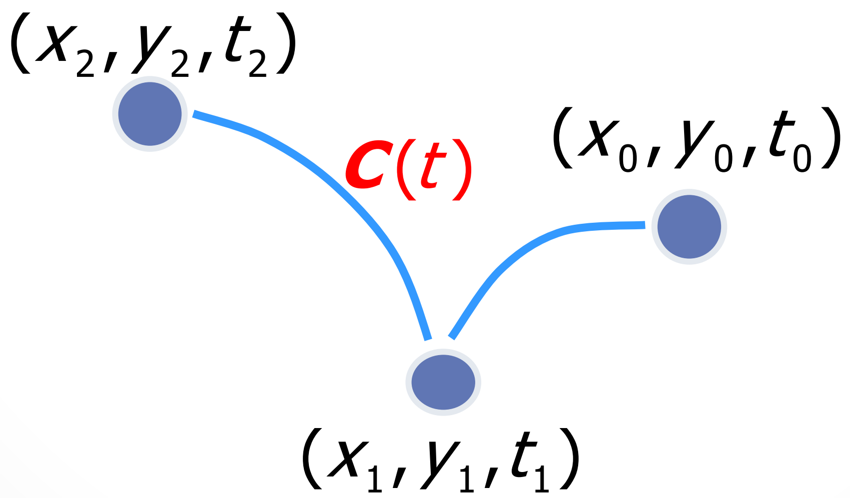
- Point Interpolation
- Rotation Interpolation

Overview

- **Point Interpolation**
- Rotation Interpolation

Interpolating Points

- Given points: $(x_i, y_i, t_i),\ i = 0, \ldots, n$
- find curve $\boldsymbol{C}(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}$ such that $\boldsymbol{C}(t_i) = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$



Parametric Polynomial Curves

- Functions $x(t)$ and $y(t)$ are polynomials of t

$$\boldsymbol{C}(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}$$

where

$$x(t) = a_0 + a_1 t + a_2 t^2 + \ldots$$

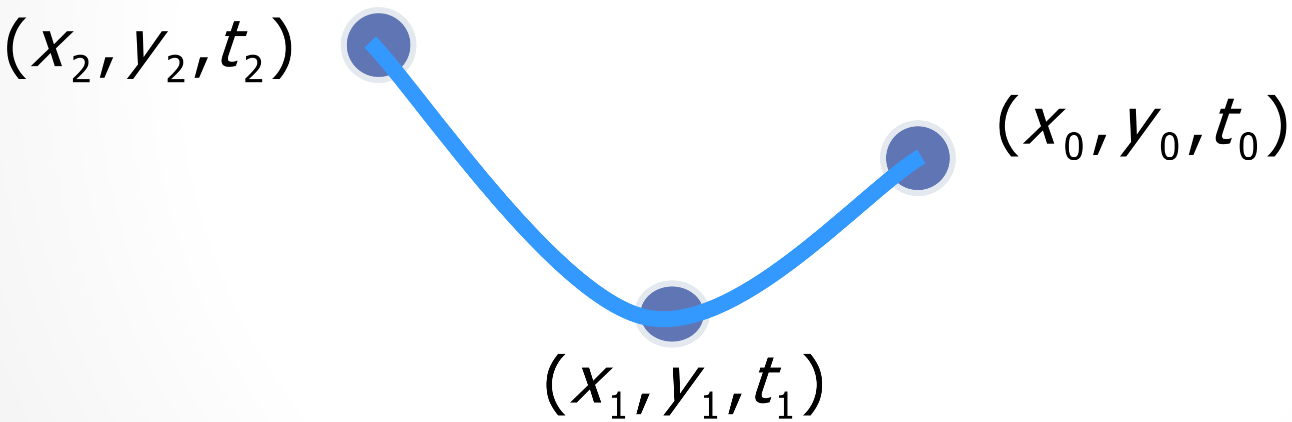
$$y(t) = b_0 + b_1 t + b_2 t^2 + \ldots$$

- The degree of the polynomial is determined by the highest order of t, which is the same for $x(t)$ and $y(t)$

Polynomial Interpolation

- 3 points can be interpolated by polynomial of degree 2, i.e.,

$$x(t) = a_0 + a_1 t + a_2 t^2$$
$$y(t) = b_0 + b_1 t + b_2 t^2$$



- The number of coefficients (unknowns) equals the number of constraints!

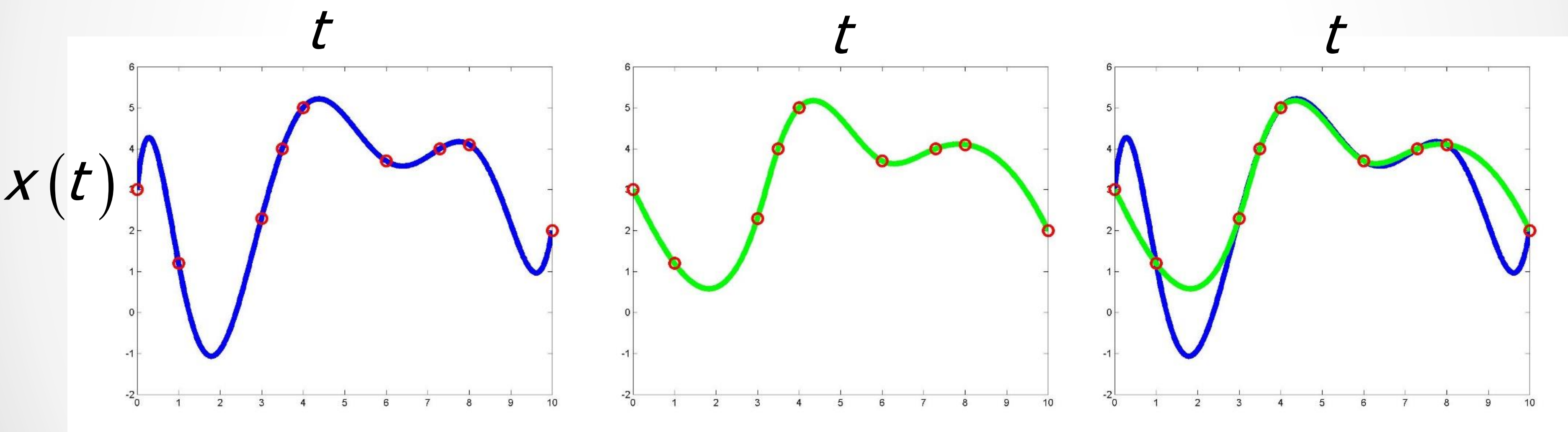
Determine Polynomial Degrees

- How about interpolating n+1 points?

- n+1 points can be interpolated using a polynomial of n degrees
  - n+1 constraints to solve for n+1 coefficients for x and y coordinate respectively!

Piecewise Polynomial Interpolation

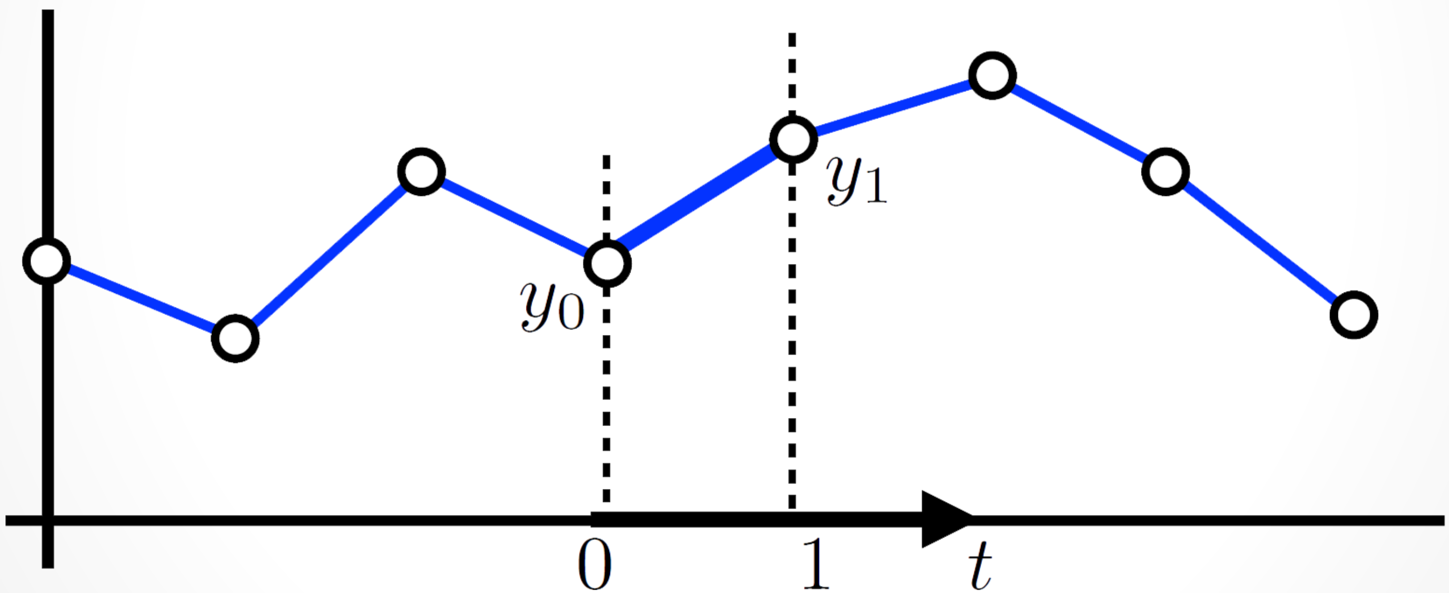
- Polynomials of small degree are fine but high degree polynomials are too wiggly. Piecewise polynomial interpolation produces nicer interpolation.



8-degree polynomial　　piecewise polynomial　　comparison

Piecewise Interpolation

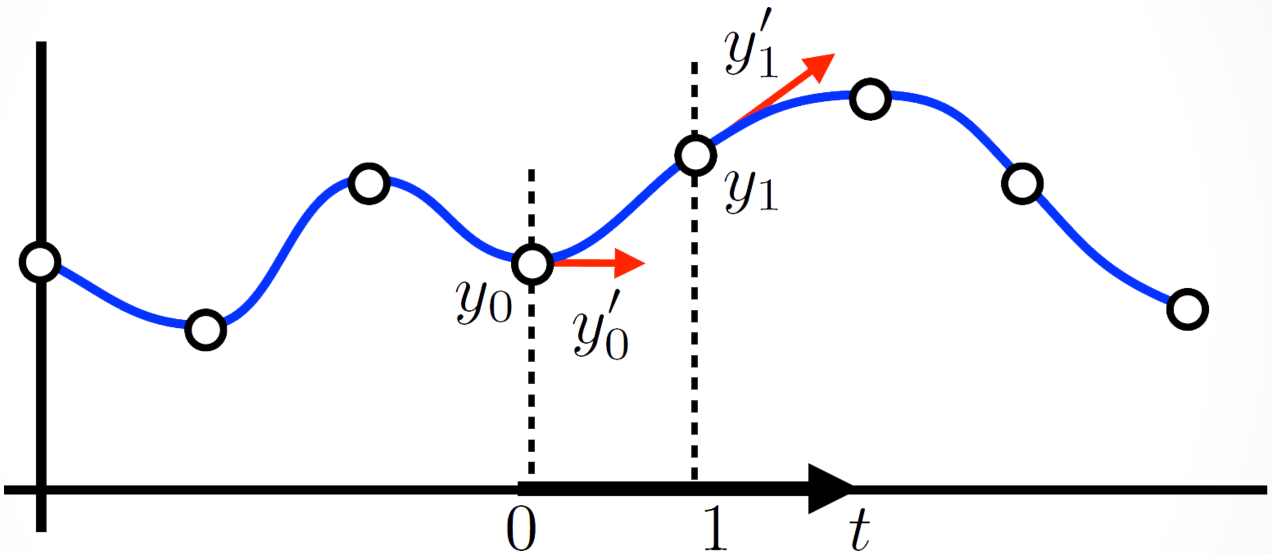
- A simple approach to interpolate points is to connect the neighboring points using line segment (polynomial of degree 1)



Note that the **value** is continuous, but the **gradient** is not

Piecewise Interpolation

- To get **gradient continuity**, we will have to use polynomials of higher degrees (parabola, cubic, …)

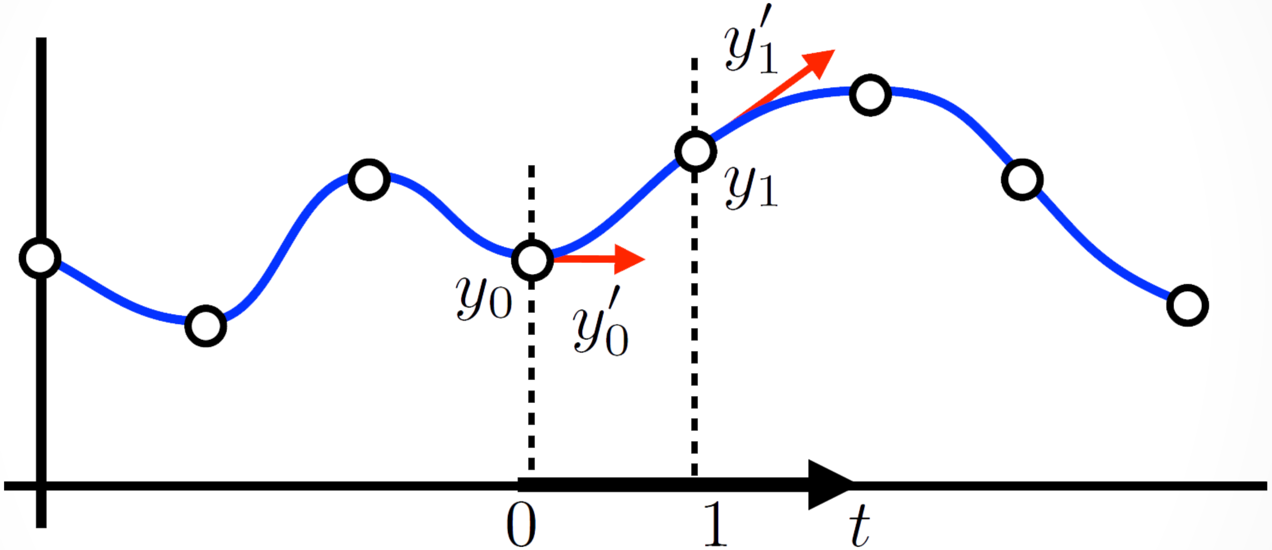


- For simplicity, we use polynomial in explicit form

$$y = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + \dots$$

Interpolating Gradients

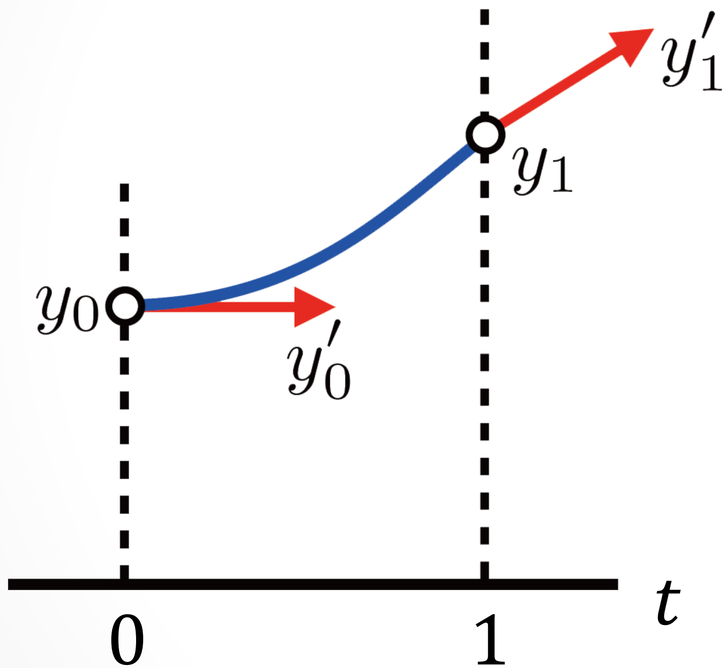
- Other than function values (positions), we need to specify gradients at start point and end point



- We need to make sure neighboring curve segments have the same gradient at the point they share

Polynomial Fitting

- Specify function **value** and **gradient** at start point and end point respectively

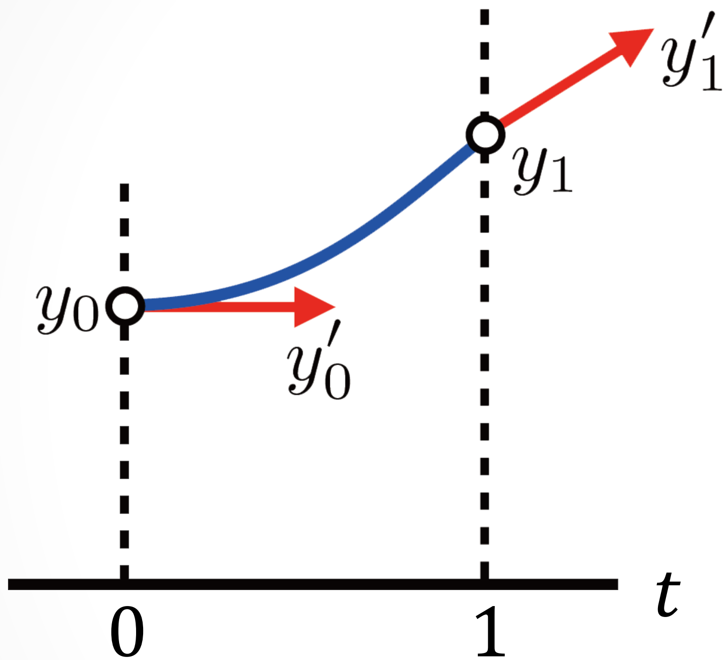


$$y(0) = y_0$$
$$y(1) = y_1$$
$$\frac{dy}{dt}(0) = y_0'$$
$$\frac{dy}{dt}(1) = y_1'$$

- **Q:** what is the order of polynomial for which we can specify 2 values and 2 gradients? (4 constraints)

Polynomial Fitting

- **A: Cubic.** $y = a_0 + a_1 t + a_2 t^2 + a_3 t^3$

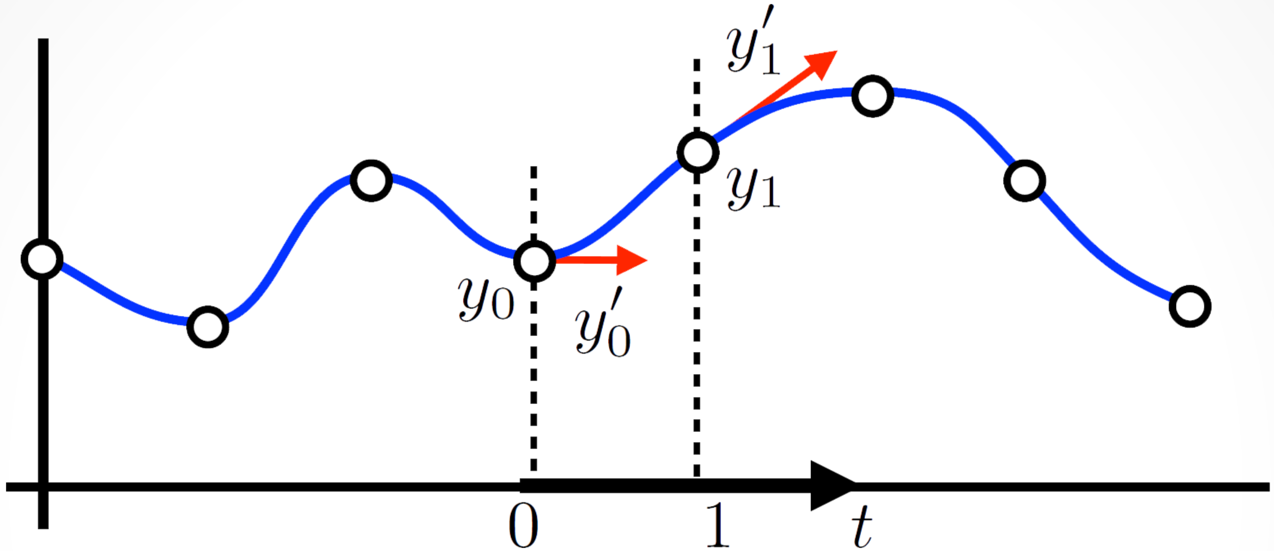


$$y(0) = y_0$$
$$y(1) = y_1$$
$$\frac{dy}{dt}(0) = y_0'$$
$$\frac{dy}{dt}(1) = y_1'$$

- Cubic has 4 coefficients (unknowns) a_0, a_1, a_2, a_3, which can be determined by these 4 constraints

- Cubic curves specified this way are called **Hermite Curves**

Continuity of the Whole Curve



- We can fit cubic Hermite curves to each piece, such that the value and the gradient are the same at the endpoints
- Now the value and the gradient are **continuous** (no jumps) for the whole curve

Overview

- Point Interpolation
- **Rotation Interpolation**

Interpolating Rotations

- Given two rotations, how to interpolate in-between?
    - Euler angles
    - Axis-angle
    - Quaternions

Flawed Solution on Rotation Matrices

- Simple idea: Linearly interpolate each entry
- Example: $\boldsymbol{M}_0$ is identity and $\boldsymbol{M}_1$ is 90-deg rotation around X-axis

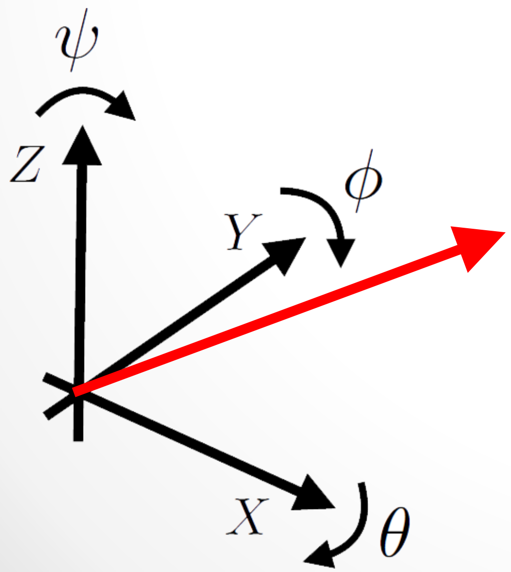
$$\text{Interpolate}\left(\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}, \begin{bmatrix}1 & 0 & 0\\0 & 0 & 1\\0 & -1 & 0\end{bmatrix}\right) = \begin{bmatrix}1 & 0 & 0\\0 & 0.5 & 0.5\\0 & -0.5 & 0.5\end{bmatrix}$$

- Is the result a rotation matrix?

The result R is not a rotation matrix. For example, check that $R^T R$ does not equal identity.

Euler Angles

- How do we generate an arbitrary rotation matrix? (rotation about arbitrary axis passing through origin)
- Simple idea: use an ordered combination of rotations about the X, Y and Z axes



$$\mathbf{R} = \mathbf{R}_z(\psi)\mathbf{R}_y(\phi)\mathbf{R}_x(\theta)$$

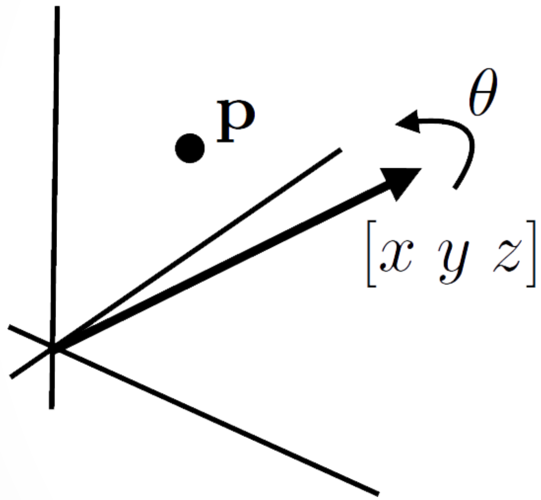
e.g. first X, then Y then Z
(all variations used)
NB: order matters!

Interpolating Euler Angles

- Linearly interpolate rotation angles corresponding to the X, Y and Z axes
- Unnatural interpolation:
  - A rotation of 90-degrees first around the z-axis and then around the y-axis has the effect of a 120-degree rotation around the axis (1, 1, 1).
  - But rotation of 30-degrees around the z- and y-axis does not have the effect of a 40-degree rotation around the axis (1, 1, 1).

Axis-Angle Rotations

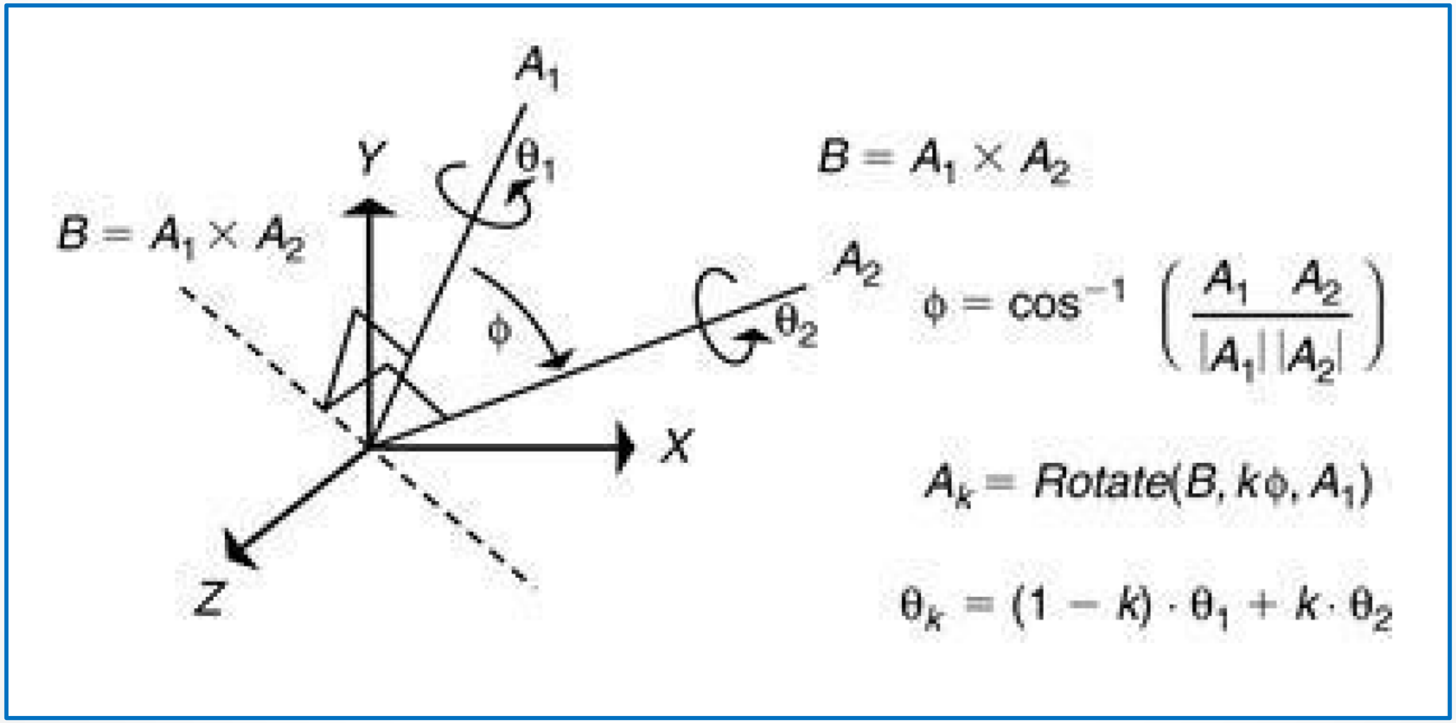
- We can represent any rotation directly by a rotation axis and angle



Consider rotating a point **P**
by an angle θ
about an axis direction $[x,y,z]$

Axis-angle Interpolation

- Interpolate axis and angle respectively



Quaternions

- **Unit Quaternions** $\mathbf{q} = q_0 + iq_1 + jq_2 + kq_3$

$$|\mathbf{q}| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1$$

- This represent rotations in a manner similar to axis-angle representation:

angle: $\theta = 2\cos^{-1} q_0$ axis: $[\ q_1\ q_2\ q_3\ ]$

Quaternion Interpolation

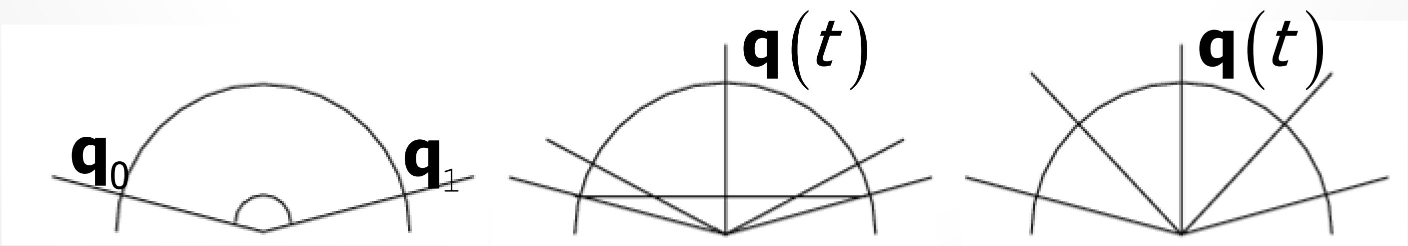
- **L**inear int**erp**olation of quaternion representation

$$\mathrm{lerp}(\mathbf{q}_0, \mathbf{q}_1, t) = \mathbf{q}(t) = \mathbf{q}_0(1-t) + \mathbf{q}_1(t)$$

- **S**pherical **l**inear int**erp**olation (slerp) along the arc lines instead of straight lines

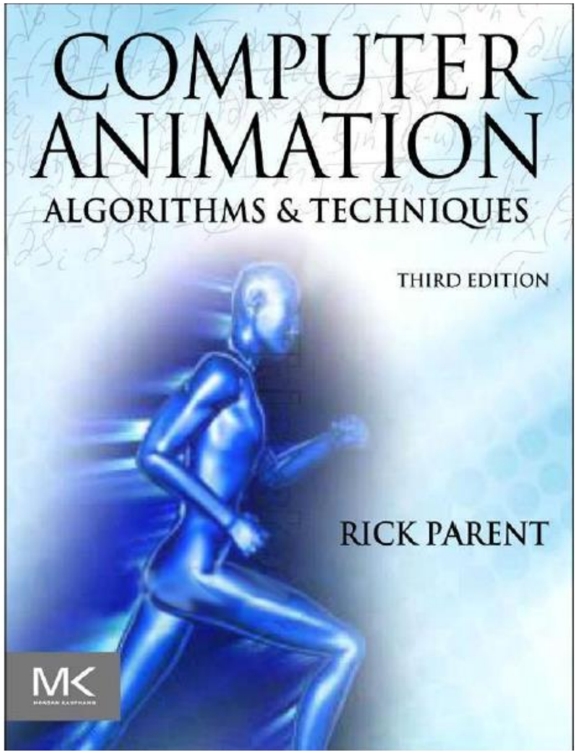
$$\mathrm{slerp}(\mathbf{q}_0, \mathbf{q}_1, t) = \mathbf{q}(t) = \frac{\mathbf{q}_0 \sin((1-t)\omega) + \mathbf{q}_1 \sin(t\omega)}{\sin(\omega)}$$

where $\omega = \cos^{-1}(\mathbf{q}_0 \bullet \mathbf{q}_1)$



keyframes lerp slerp

More about Interpolation



Chapter 3