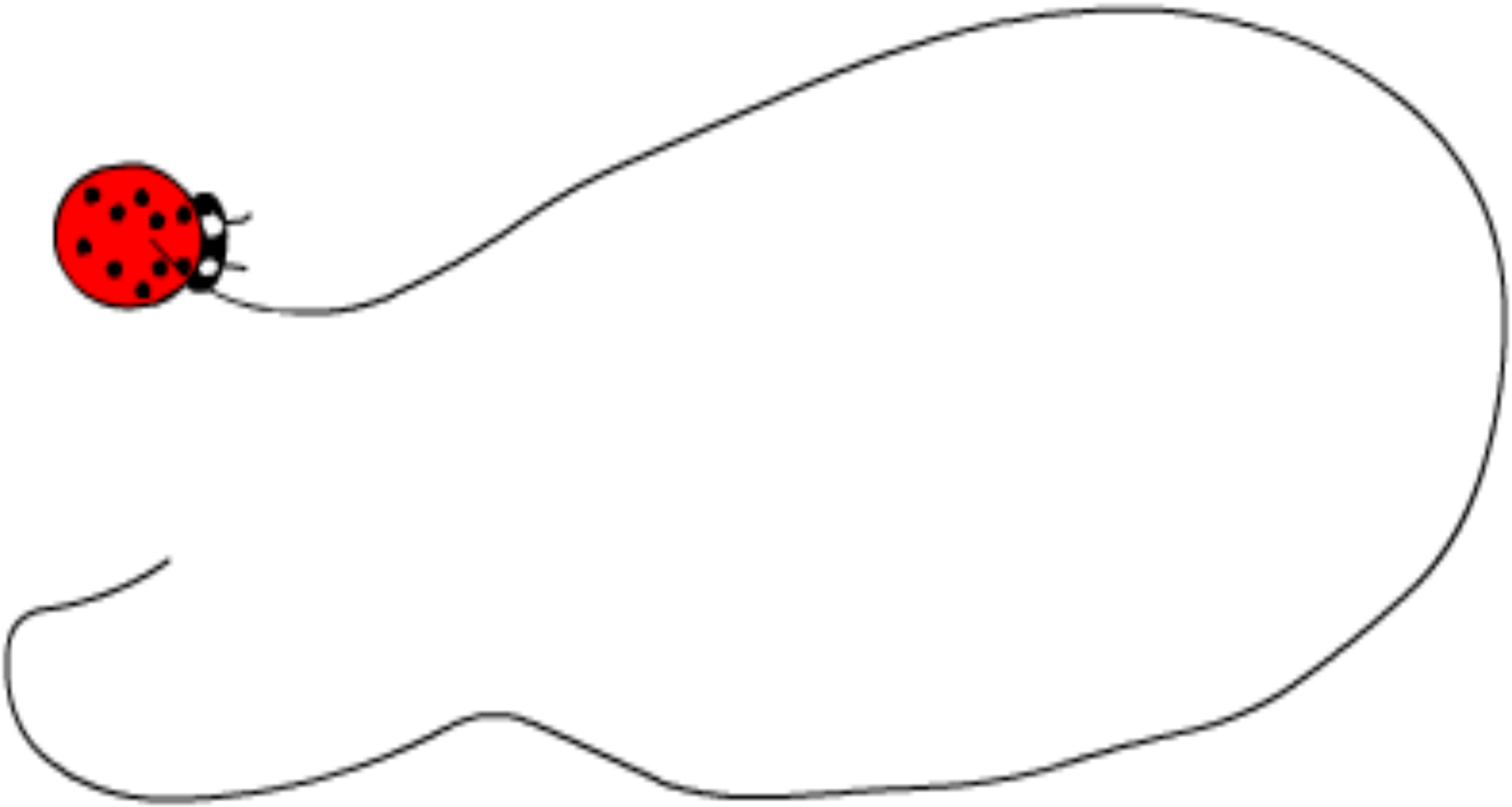
Animation Curves and Splines 2



Animation Homework

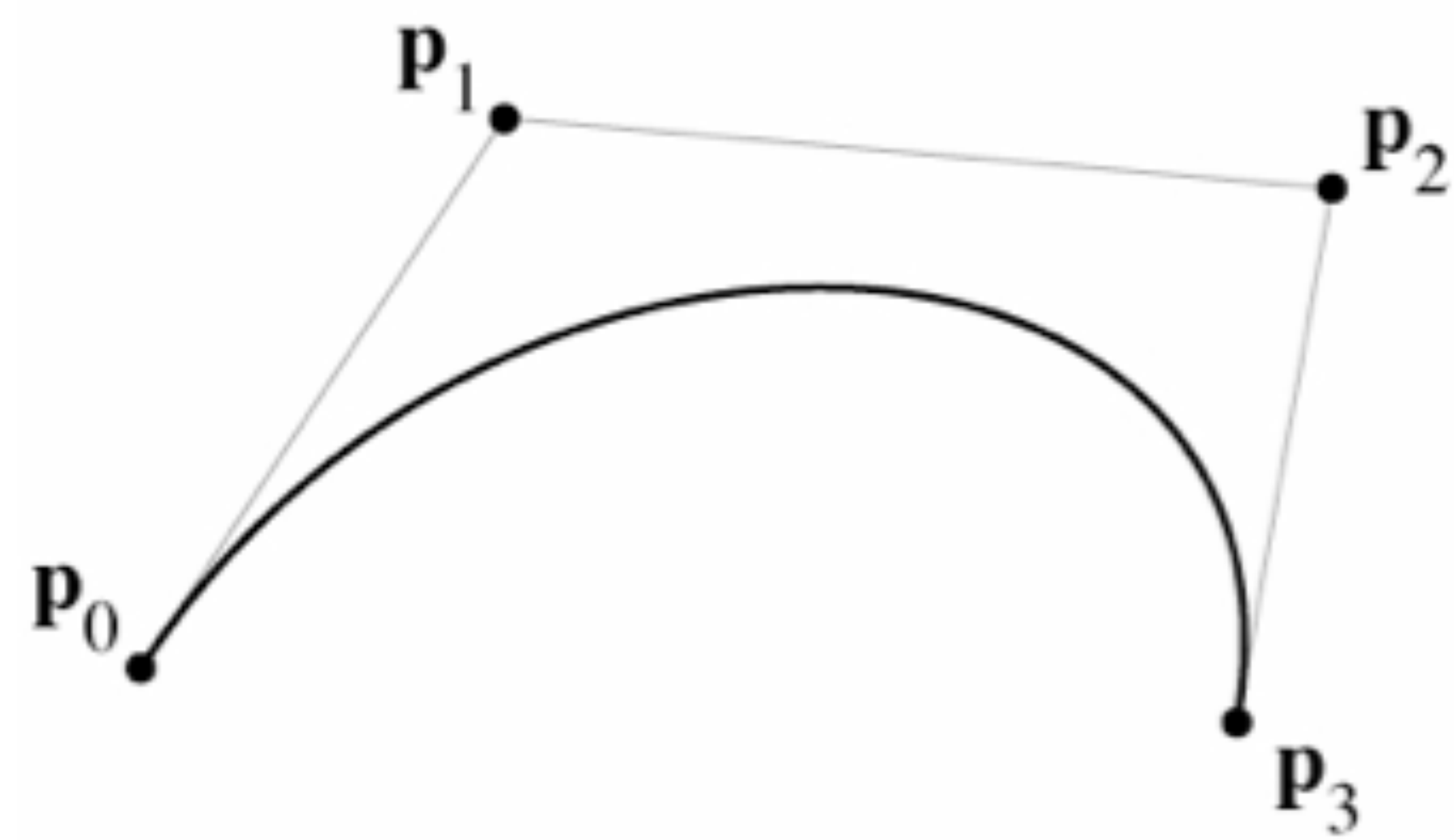
- Set up a simple avatar
  - E.g. cube/sphere (or square/circle in 2D)
- Specify some key frames (positions/orientations)
- Associate a time with each key frame
- Create smooth animations for the avatar movement
  - does it pass through or just near the key frames and positions?
- Save the canned animations for later use
- Provide a couple of examples
  - A crawling bug, a bouncing ball, etc...
- Use unity...

Thursday Animation & Unity

Bezier Curves

Bezier Curves

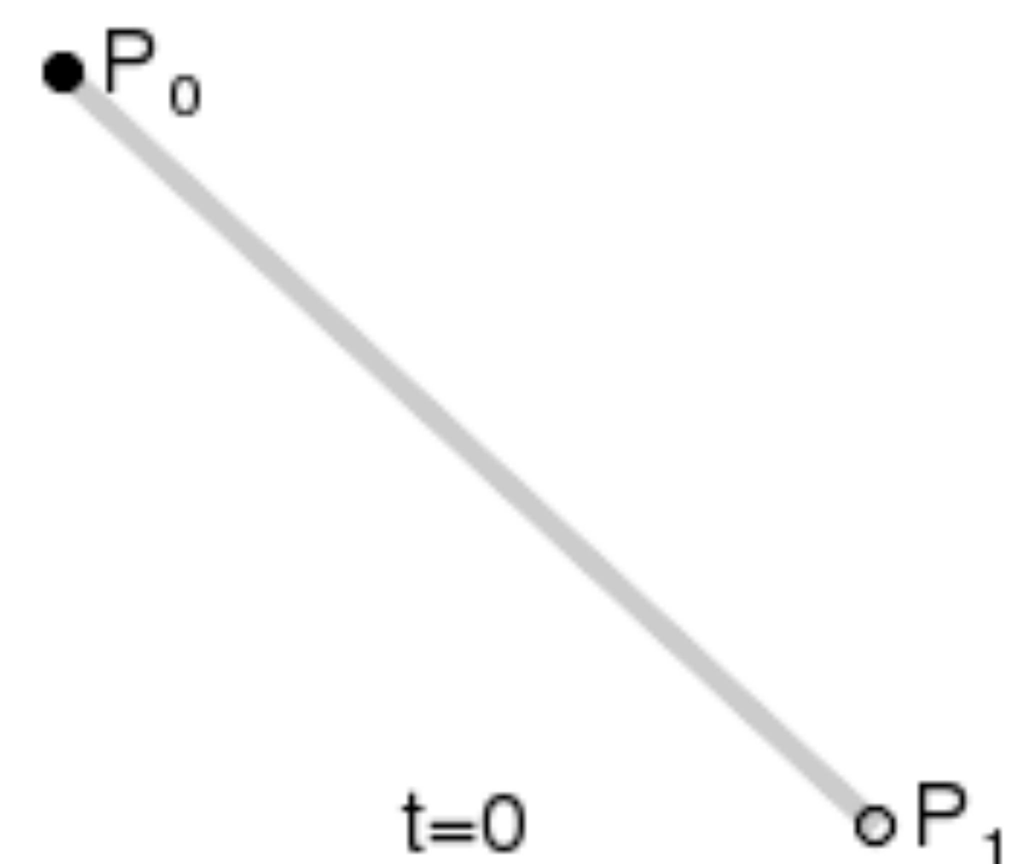
- Specify control points instead of derivatives



Bezier Curves

- A Bezier curve is defined by a set of control points $P_0, \ldots, P_n$ where n determines the order of a Bezier curve
- Start from a linear Bezier curve based on two control points, and build recursively
- A linear Bezier curve is a straight line

$$\mathbf{P}_{\mathbf{P_0},\mathbf{P_1}}(t) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1$$

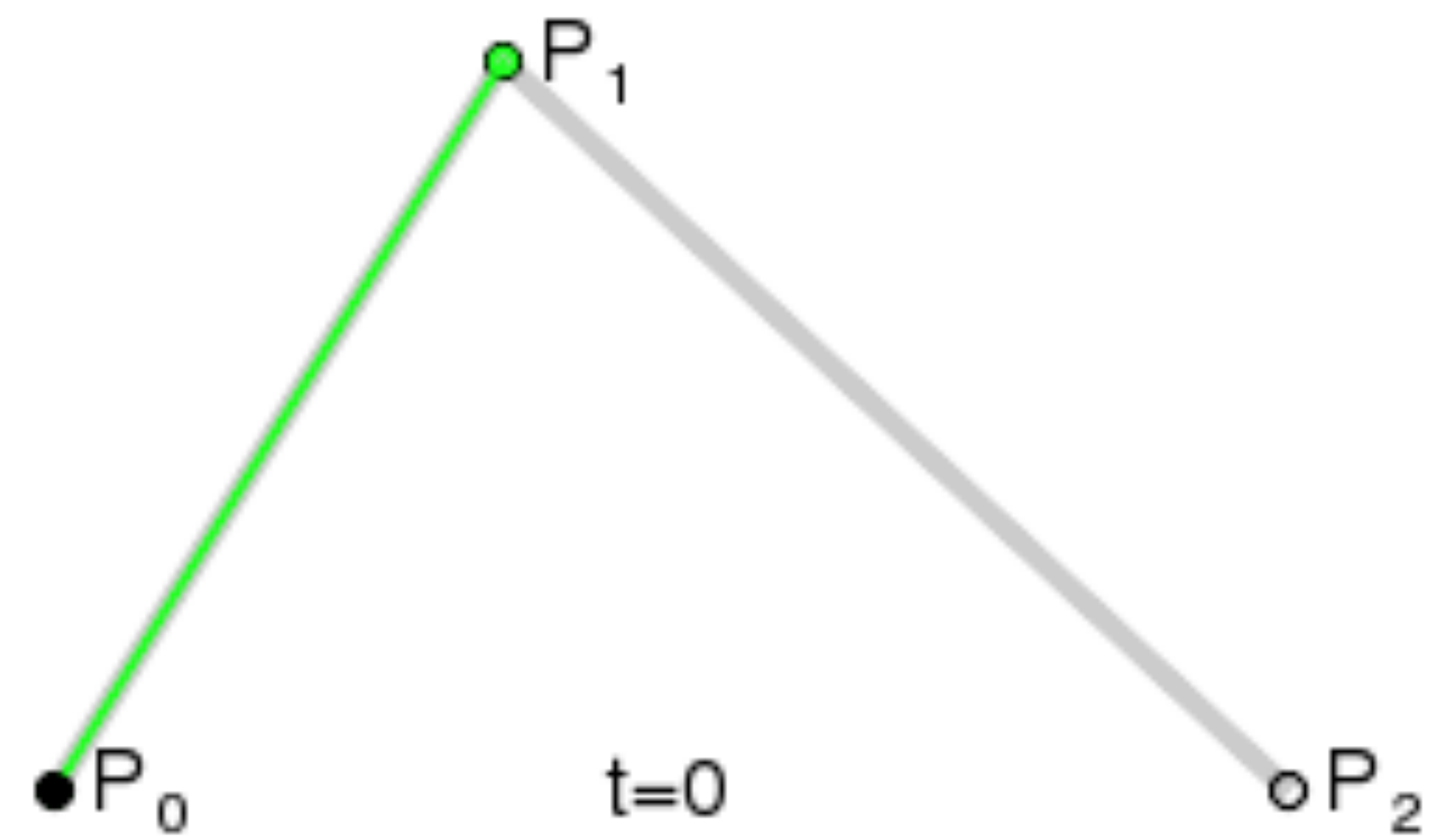


Bezier Curves

- A quadratic Bezier curve is determined by 3 control points $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$

$$\mathbf{P}_{\mathbf{P}_0,\mathbf{P}_1,\mathbf{P}_2}(t) = (1-t)\mathbf{P}_{\mathbf{P}_0,\mathbf{P}_1}(t) + t\mathbf{P}_{\mathbf{P}_1,\mathbf{P}_2}(t)$$

- $\mathbf{P}_{\mathbf{P}_0,\mathbf{P}_1,\mathbf{P}_2}(t)$ is linearly interpolated on the green line, and the endpoints of the green line lie on linear Bezier curves determined by $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_1$, $\mathbf{P}_2$ respectively



- The basis functions are quadratic functions

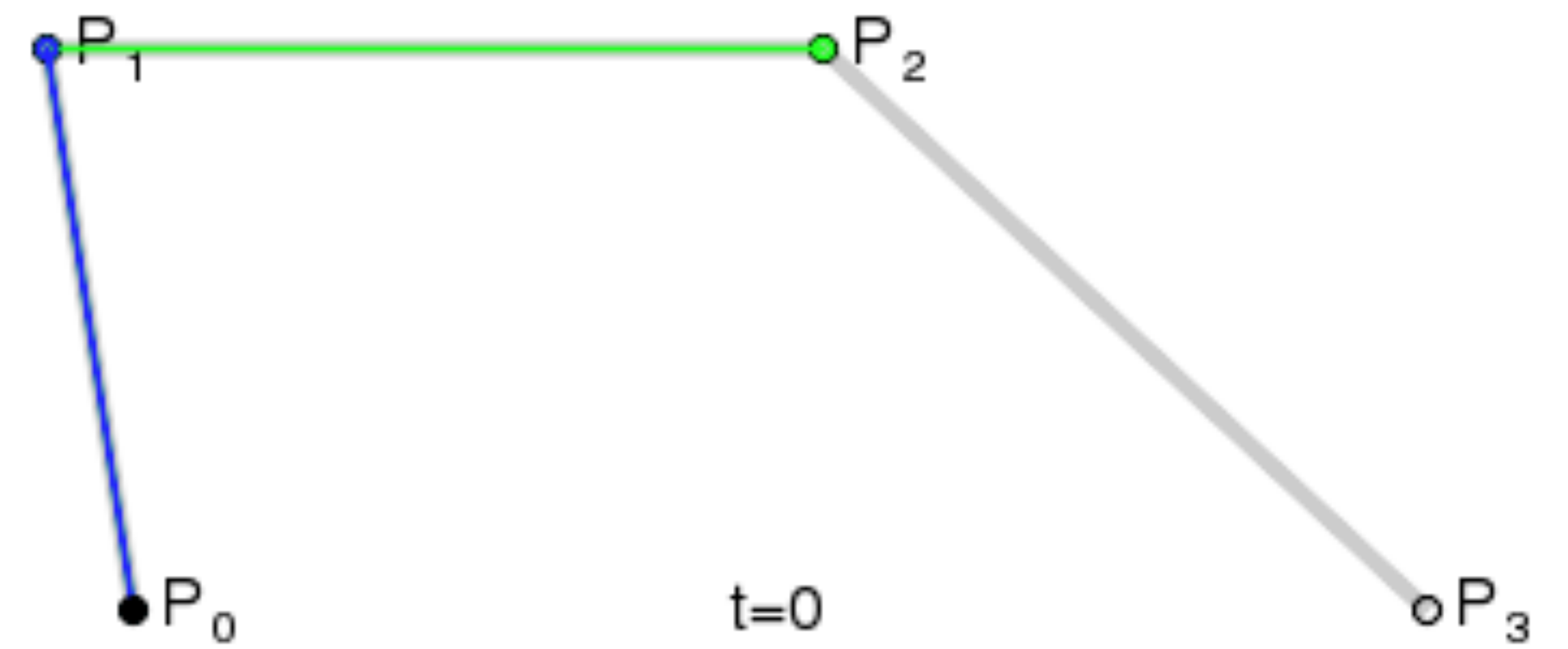
$$\mathbf{P}_{\mathbf{P}_0,\mathbf{P}_1,\mathbf{P}_2}(t) = (1-t)^2\mathbf{P}_0 + 2t(1-t)\mathbf{P}_1 + t^2\mathbf{P}_2$$

Bezier Curves

- A cubic Bezier curve is determined by 4 control points

$$\mathbf{P}_{\mathbf{P}_0,\mathbf{P}_1,\mathbf{P}_2,\mathbf{P}_3}(t) = (1-t)\mathbf{P}_{\mathbf{P}_0,\mathbf{P}_1,\mathbf{P}_2}(t) + t\mathbf{P}_{\mathbf{P}_1,\mathbf{P}_2,\mathbf{P}_3}(t)$$

- $\mathbf{P}_{\mathbf{P}_0,\mathbf{P}_1,\mathbf{P}_2,\mathbf{P}_3}(t)$ is linearly interpolated on the blue line, and the path of the two end points of the blue line are quadratic Bezier curves determined by $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$ and $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$ respectively



- The basis functions are cubic functions

$$\mathbf{P}_{\mathbf{P}_0,\mathbf{P}_1,\mathbf{P}_2,\mathbf{P}_3}(t) = (1-t)^3\mathbf{P}_0 + 3t(1-t)^2\mathbf{P}_1 + 3t^2(1-t)\mathbf{P}_2 + t^3\mathbf{P}_3$$

Bezier Curves

- Higher order Bezier curves are defined recursively

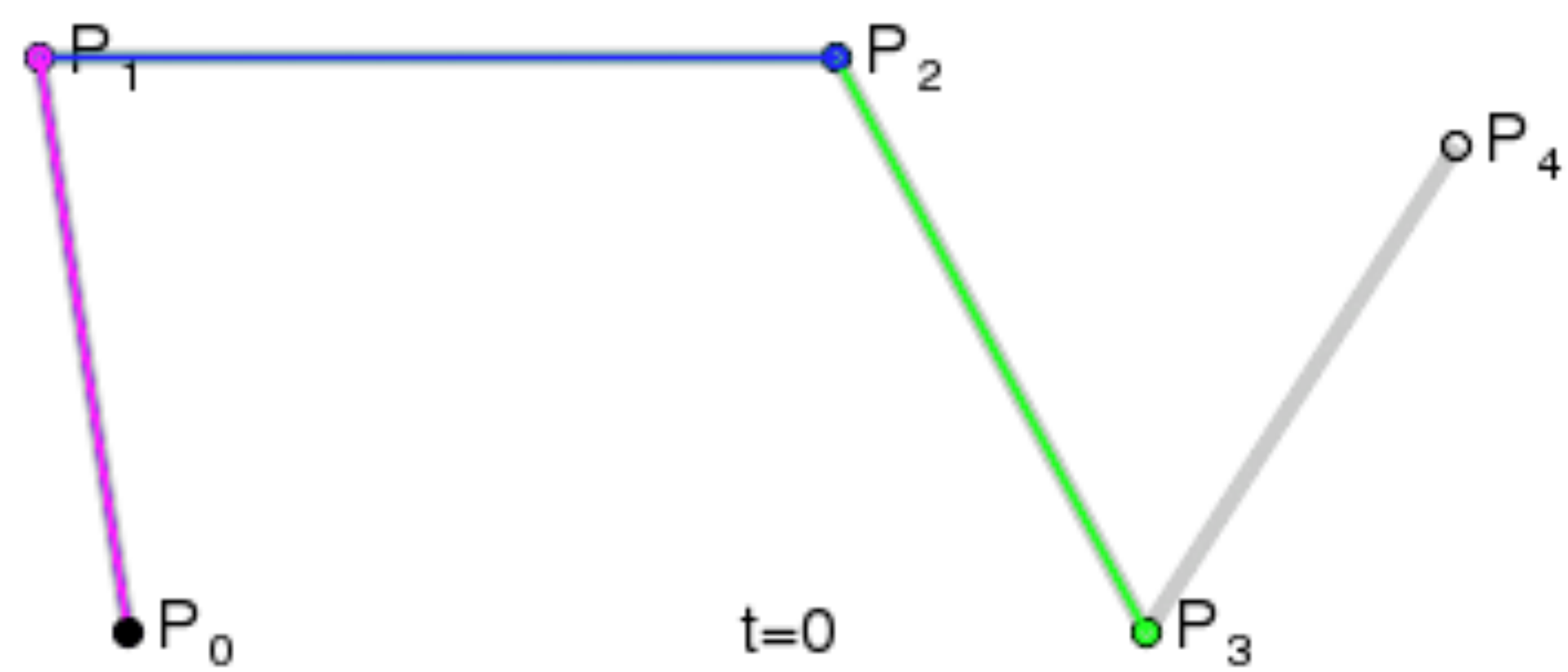
$$\mathbf{P}_{\mathbf{P_0},\mathbf{P_1},\ldots,\mathbf{P_n}}(t) = (1-t)\mathbf{P}_{\mathbf{P_0},\mathbf{P_1},\ldots,\mathbf{P_{n-1}}}(t) + t\mathbf{P}_{\mathbf{P_1},\mathbf{P_2},\ldots,\mathbf{P_n}}(t)$$

- The basis functions of an n^{th} order Bezier curve are n^{th} order polynomials

$$\mathbf{P}_{\mathbf{P_0},\mathbf{P_1},\ldots,\mathbf{P_n}}(t) = \sum_{i=0}^{n} B_i^n(t)\mathbf{P}_i$$

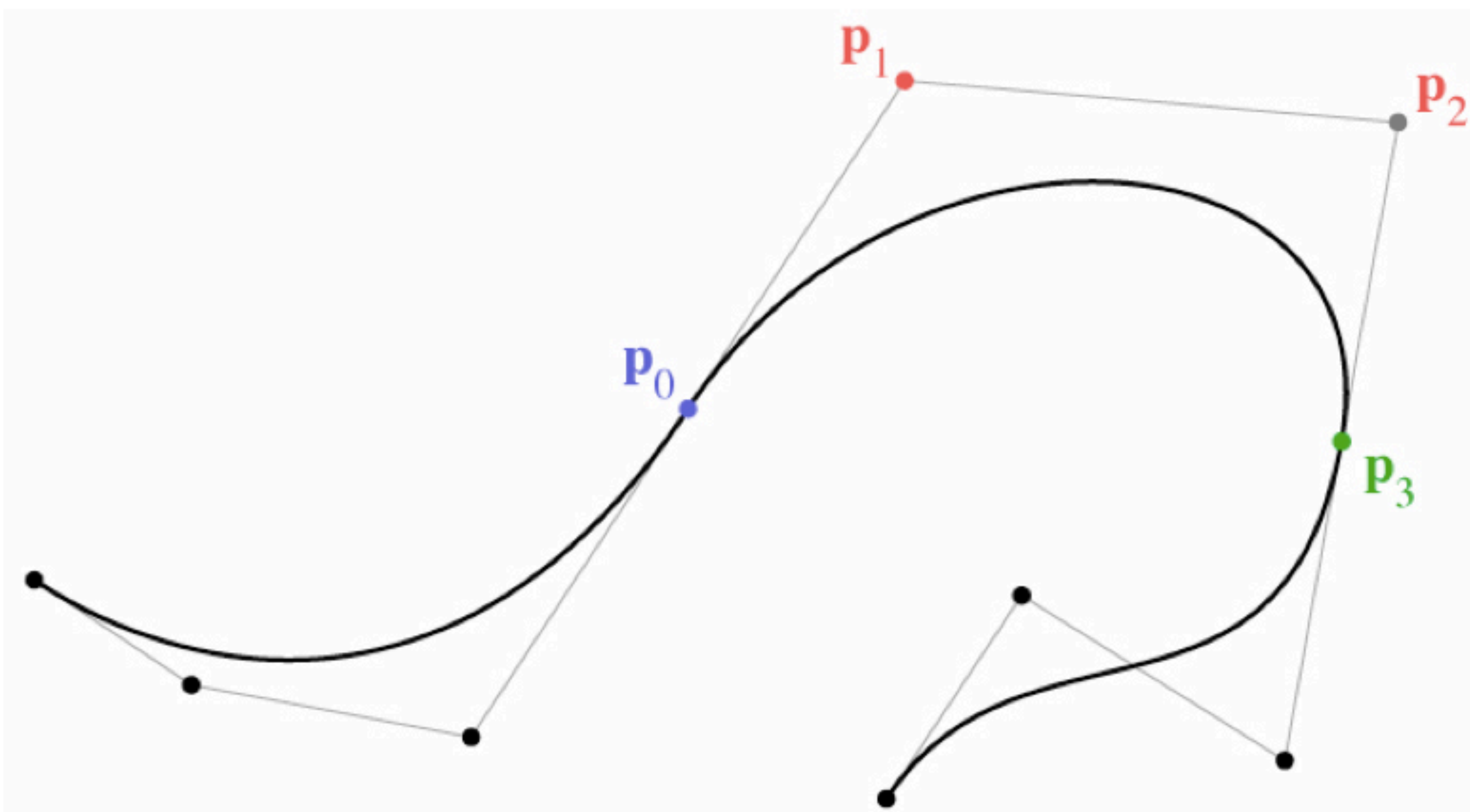
where $B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$

- High order Bezier curves are more expensive to evaluate
- When a complex shape is needed, low order Bezier curves (usually cubic) are connected together

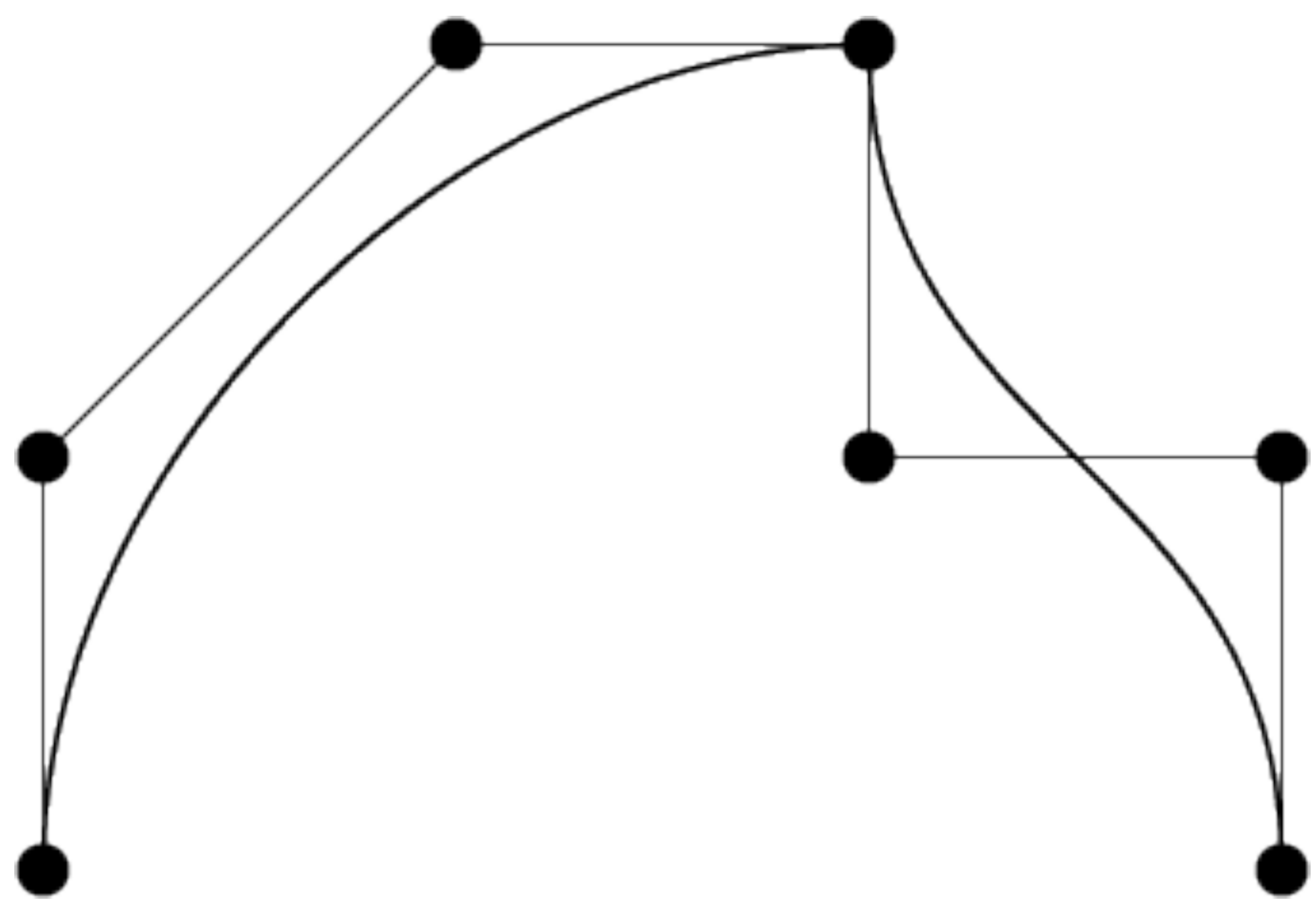


Connecting Cubic Bezier Curves

- Want velocity to be continuous
- Need co-linear control points across the junctions

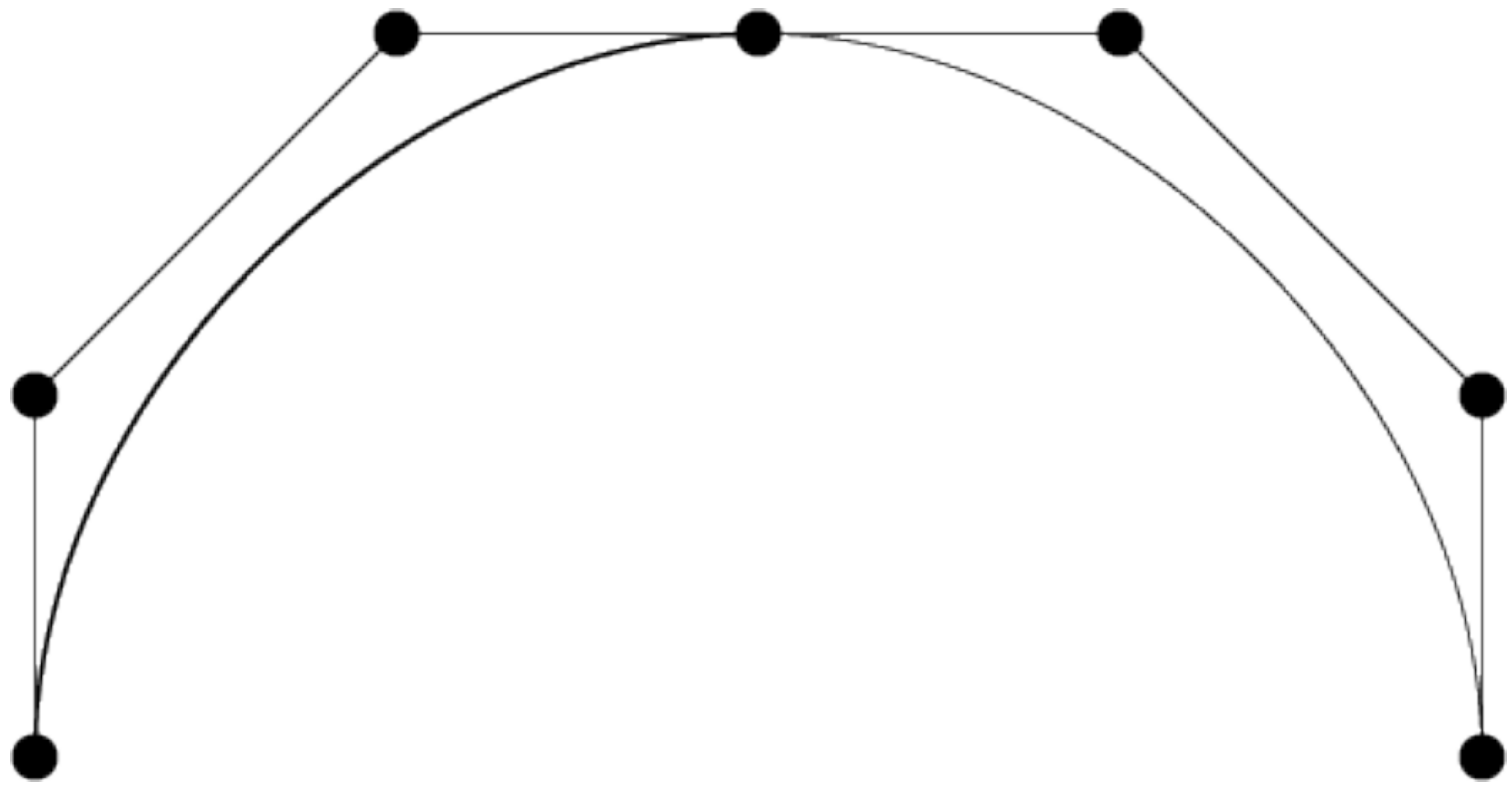


C^0 Continuity for Bezier Curves



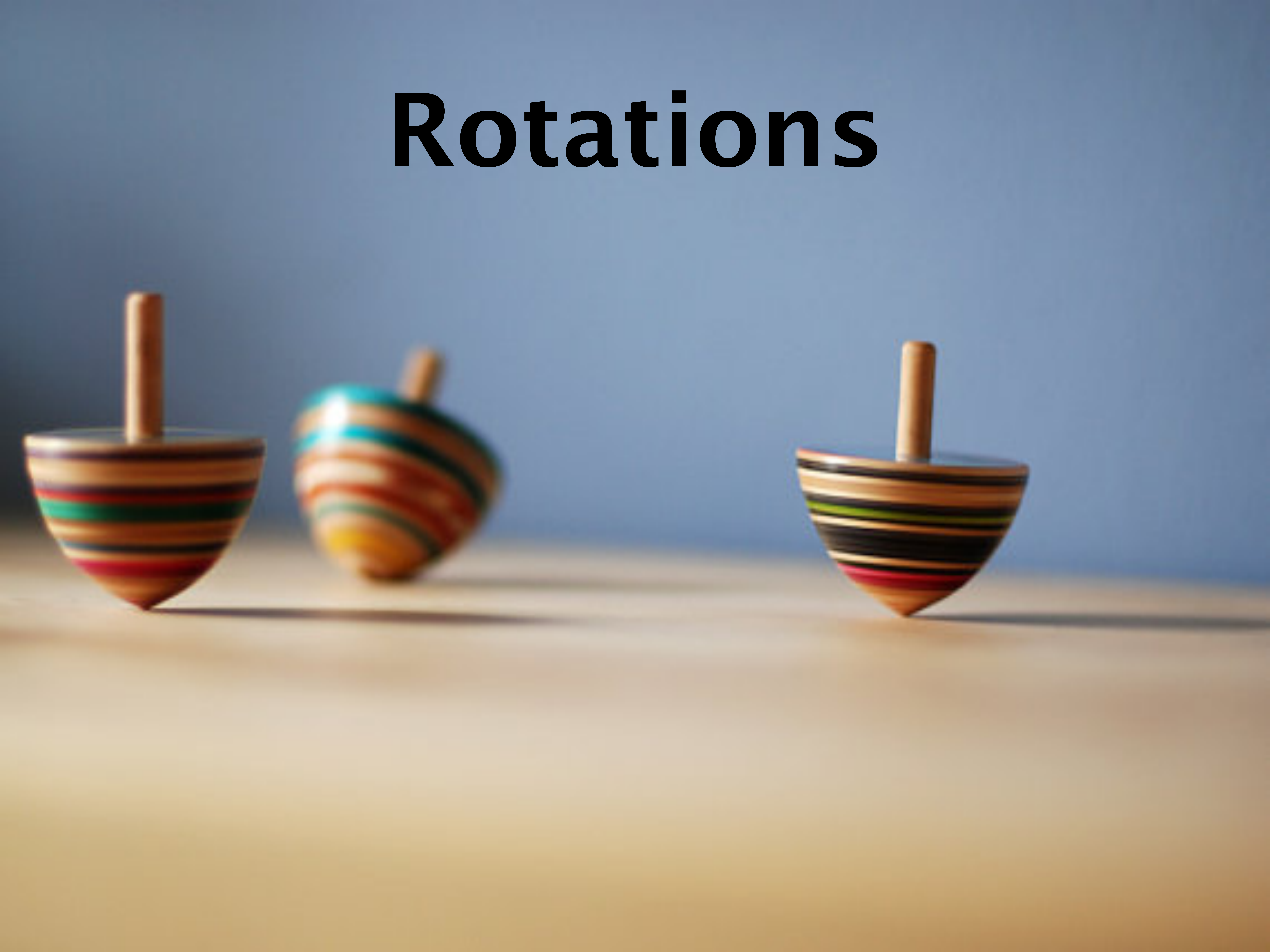
- Points are specified continuously
- But tangents are specified discontinuously

C^1 Continuity for Bezier Curves



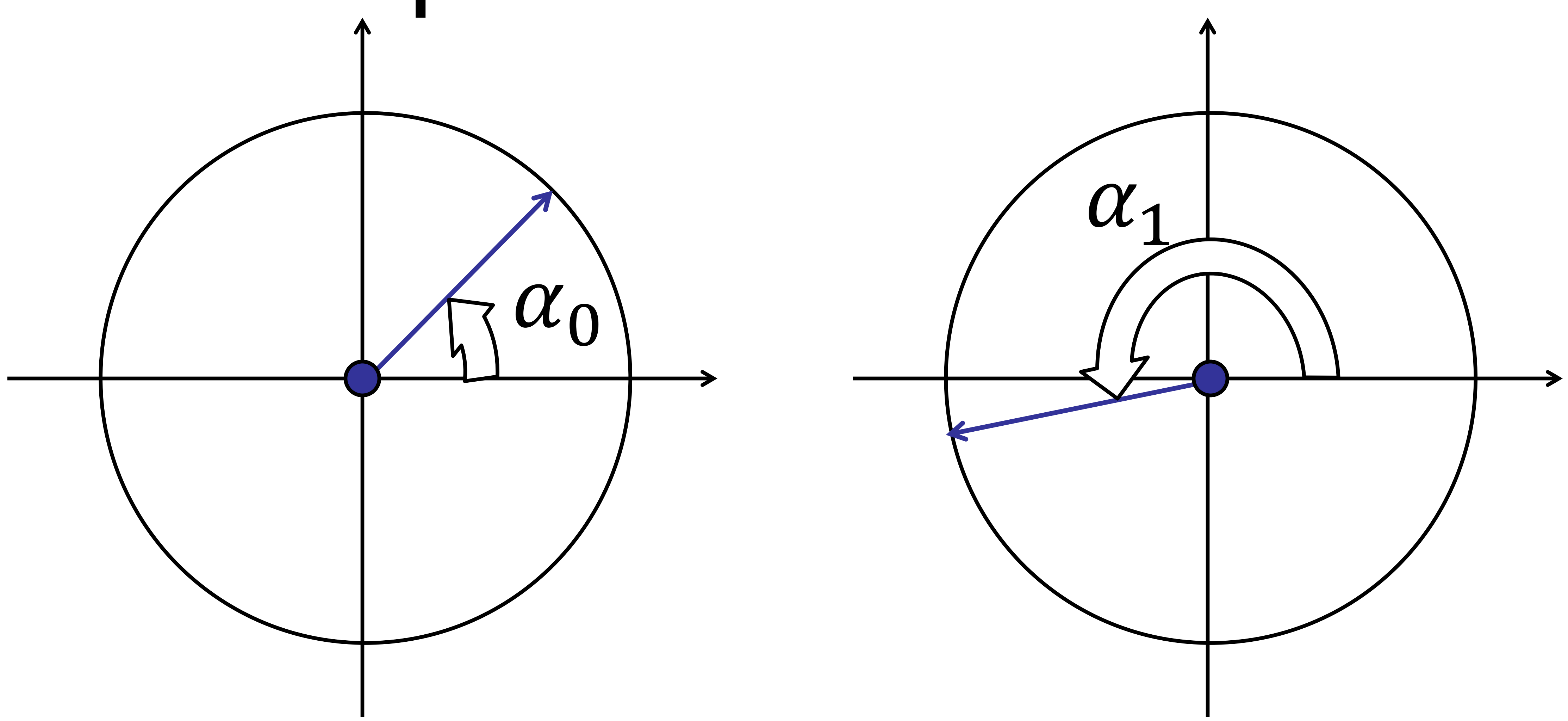
- **Tangents are specified continuously as well**

Rotations



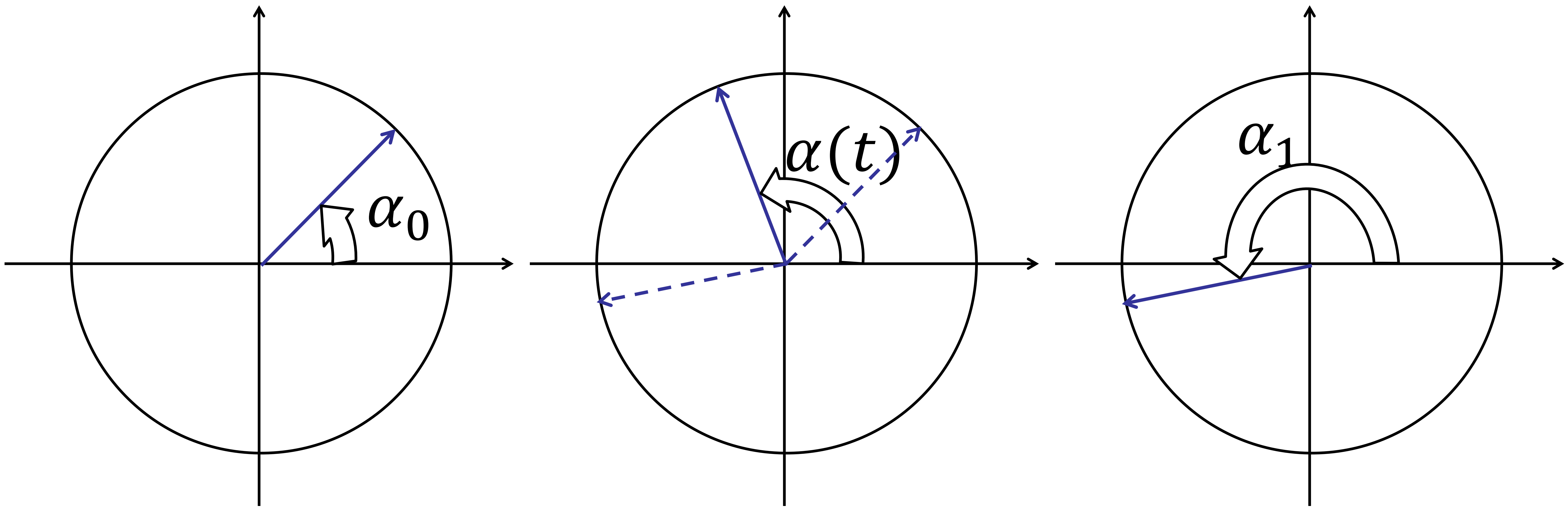
Rotation Interpolation

- Besides translation, the motion of a rigid object also includes rotation
- Consider an axis of rotation (out of the page) and angles of rotation α with respect to that axis



- Would like the object to smoothly change from one orientation to the other

Rotation Interpolation



Linear interpolation of Angles: $\alpha(t) = (1-t)\alpha_0 + t\alpha_1$

- Need to be mindful of angle limits
- Not necessarily the shortest path
  - Suppose we interpolate from $\alpha_0 = 1°$ to $\alpha_1 = 359°$
  - This rotates almost a full circle, although the two angles are nearly the same

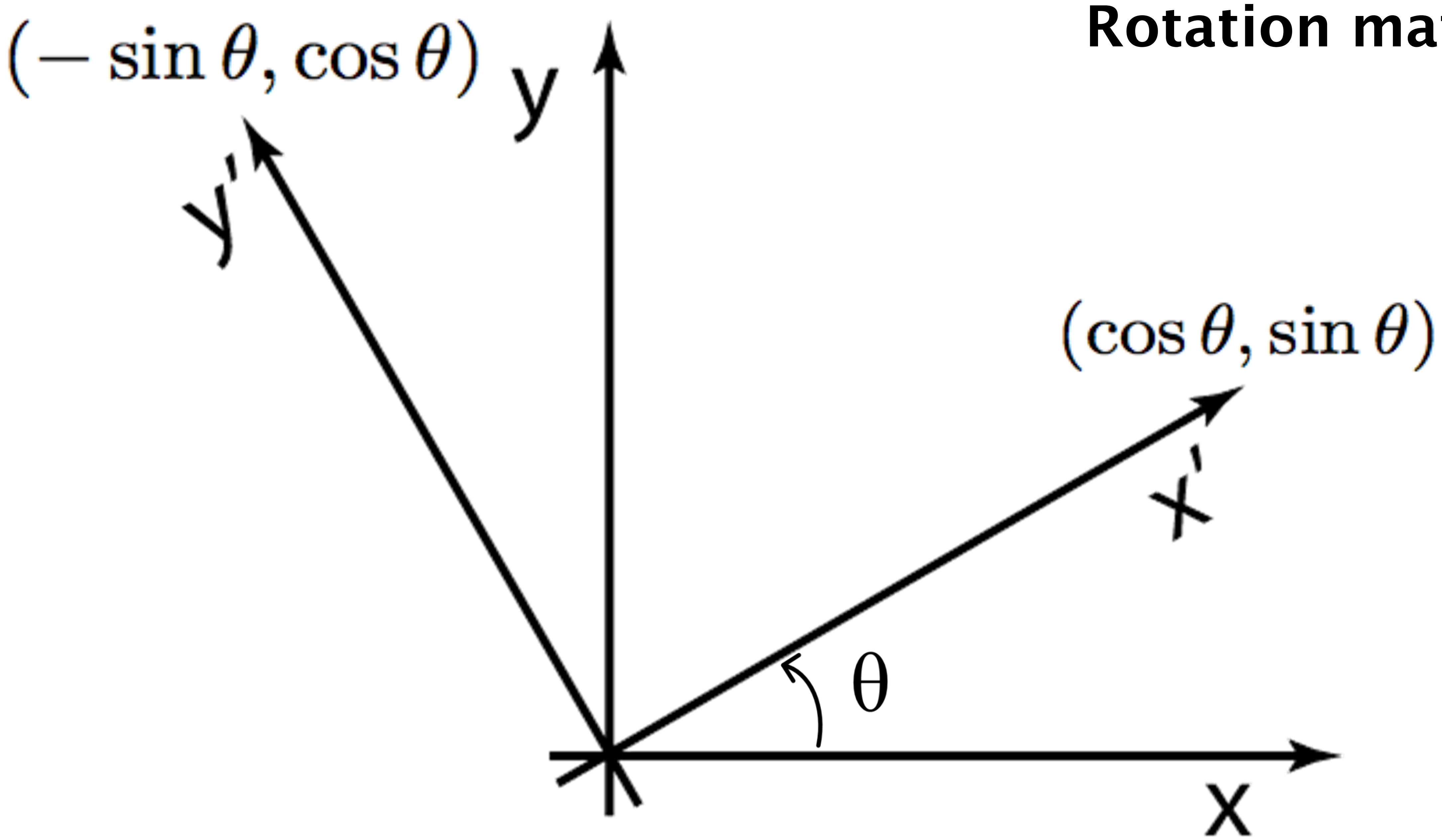
Recall: 2D Rotation Matrix

The columns of the matrix are the new locations of the x and y axes

$$\begin{bmatrix} m_{xx} \\ m_{yx} \end{bmatrix} = \begin{bmatrix} m_{xx} & m_{xy} \\ m_{yx} & m_{yy} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} m_{xy} \\ m_{yy} \end{bmatrix} = \begin{bmatrix} m_{xx} & m_{xy} \\ m_{yx} & m_{yy} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

So set the columns to the desired new locations of the x and y axes

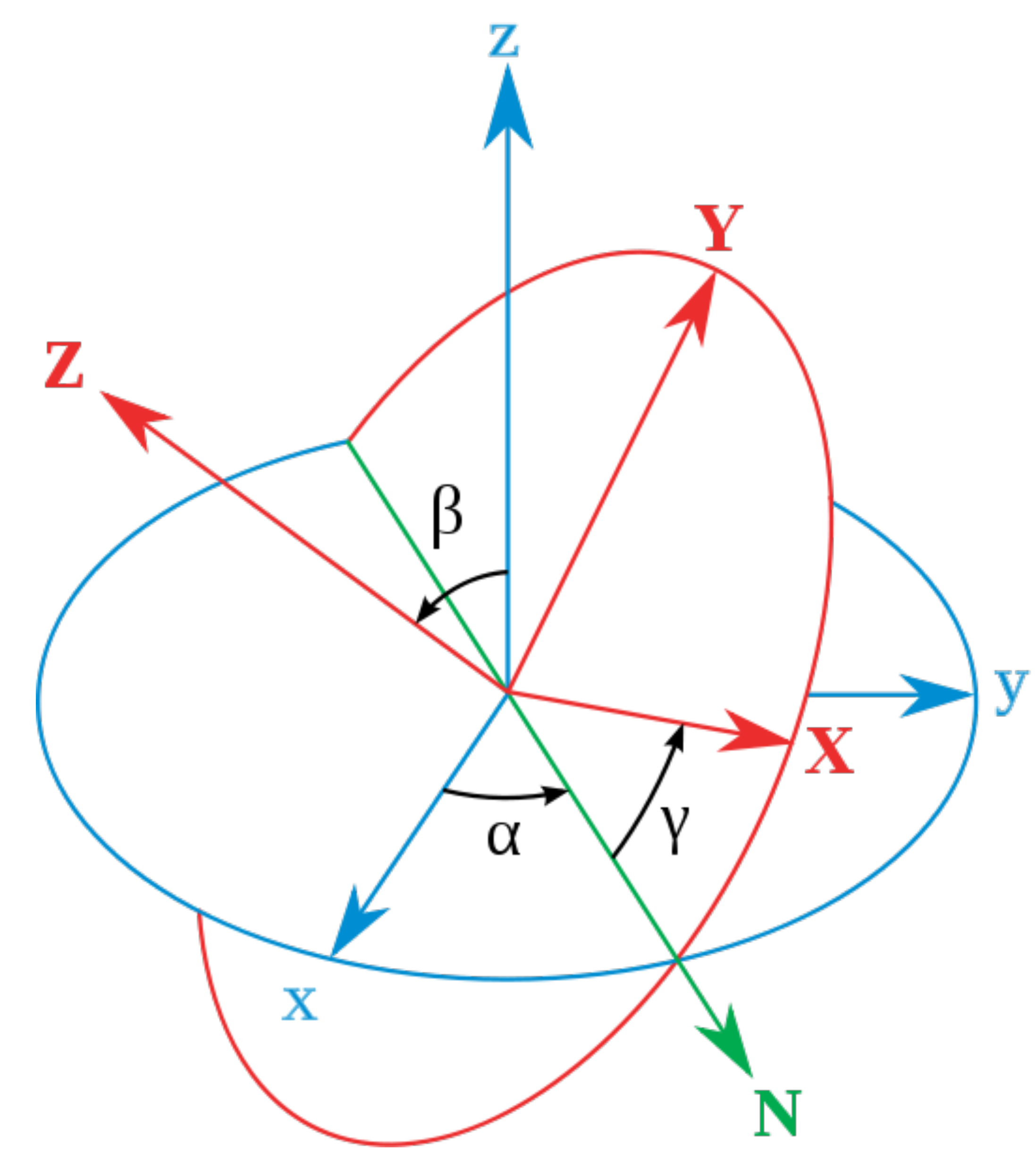
Rotation matrix = $\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$



3D is similar…

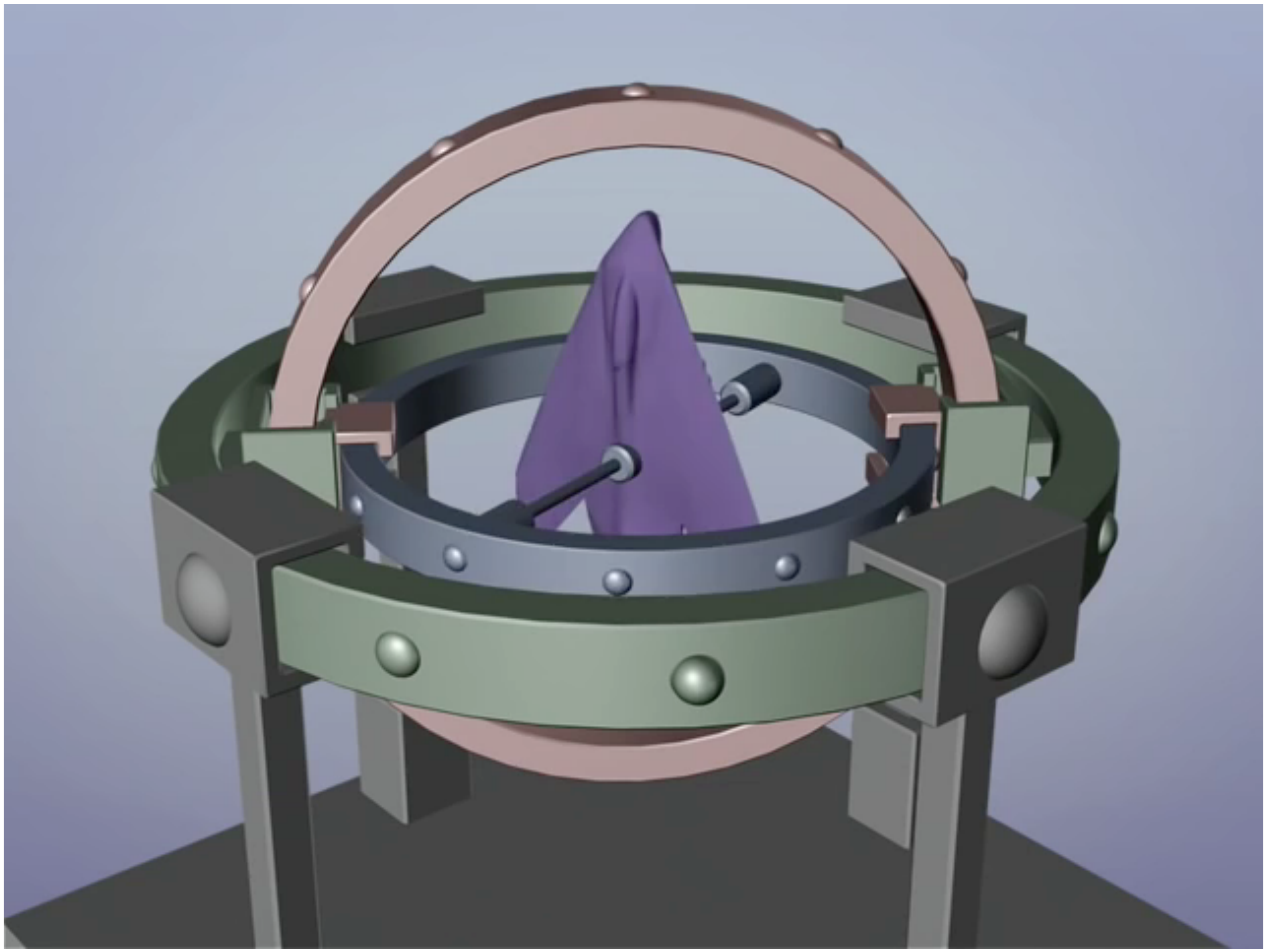
Euler Angles

- Euler angles:
  - from the original axes (xyz)
  - α: a rotation around the z-axis $(x'y'z')$
  - β: a rotation around the N-axis $(x''y''z'')$
  - γ: a rotation around the Z-axis (XYZ)
  - Three angles are applied in this fixed sequence
  - N-axis and Z-axis are both moving axes
- Interpolate rotation using Euler angles
  - Could parameterize each of the angles: α(t), β(t), γ(t)



Euler Angles

- Euler angles can lead to artifacts: gimbal lock



Question #1

LONG FORM:

- Summarize how Euler Angles work, and explain gimbal lock.
- List 5 things you plan to do research on in order to learn more about designing your game (One sentence for each idea).

SHORT FORM

- Form clusters (of about size 5-ish?) with at least one <u>novice</u> gamer in each cluster.
- Take turns answering questions that person might have, and otherwise giving him/her advice.
- Write down the best piece of advice you heard.

Quaternions

Quaternions

- Quaternions are an extension of complex numbers

$$q = s + xi + yj + zk$$

- The <u>conjugate</u> of a quaternion is defined as

$$q^* = s - xi - yj - zk$$

- They are added and subtracted (term by term) as usual.

- Multiplication is defined as follows: (using the table)

$$\begin{aligned} q_1 q_2 = (&s_1 s_2 - x_1 x_2 - y_1 y_2 - z_1 z_2, \\ &s_1 x_2 + x_1 s_2 + y_1 z_2 - z_1 y_2, \\ &s_1 y_2 - x_1 z_2 + y_1 s_2 + z_1 x_2, \\ &s_1 z_2 + x_1 y_2 - y_1 x_2 + z_1 s_2) \end{aligned}$$

×	1	i	j	k
1	1	i	j	k
i	i	−1	k	$-j$
j	j	$-k$	−1	i
k	k	j	$-i$	−1

Unit Quaternions as Rotations

- A quaternion can be expressed as a scalar/vector pair:

  $q = (s, \vec{v})$ where $\vec{v} = (x, y, z)$

- A unit (length) quaternion can be obtained by dividing through all the elements by: $\|q\| = \sqrt{s^2 + x^2 + y^2 + z^2}$

- Each unit quaternion corresponds to a rotation
  - For a rotation around a 3D axis $\hat{n}$ of angle θ, the corresponding unit quaternion is

    $$(\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\hat{n})$$

  - A quaternion multiplied by a nonzero scalar still corresponds to the same rotation (since normalization removes the scalar)

Unit Quaternions as Rotations

- A unit quaternion (s, x, y, z) is equivalent a rotation matrix:

$$\begin{pmatrix} 1-2y^2-2z^2 & 2xy-2sz & 2xz+2sy \\ 2xy+2sz & 1-2x^2-2z^2 & 2yz-2sx \\ 2xz-2sy & 2sx+2yz & 1-2x^2-2y^2 \end{pmatrix}$$

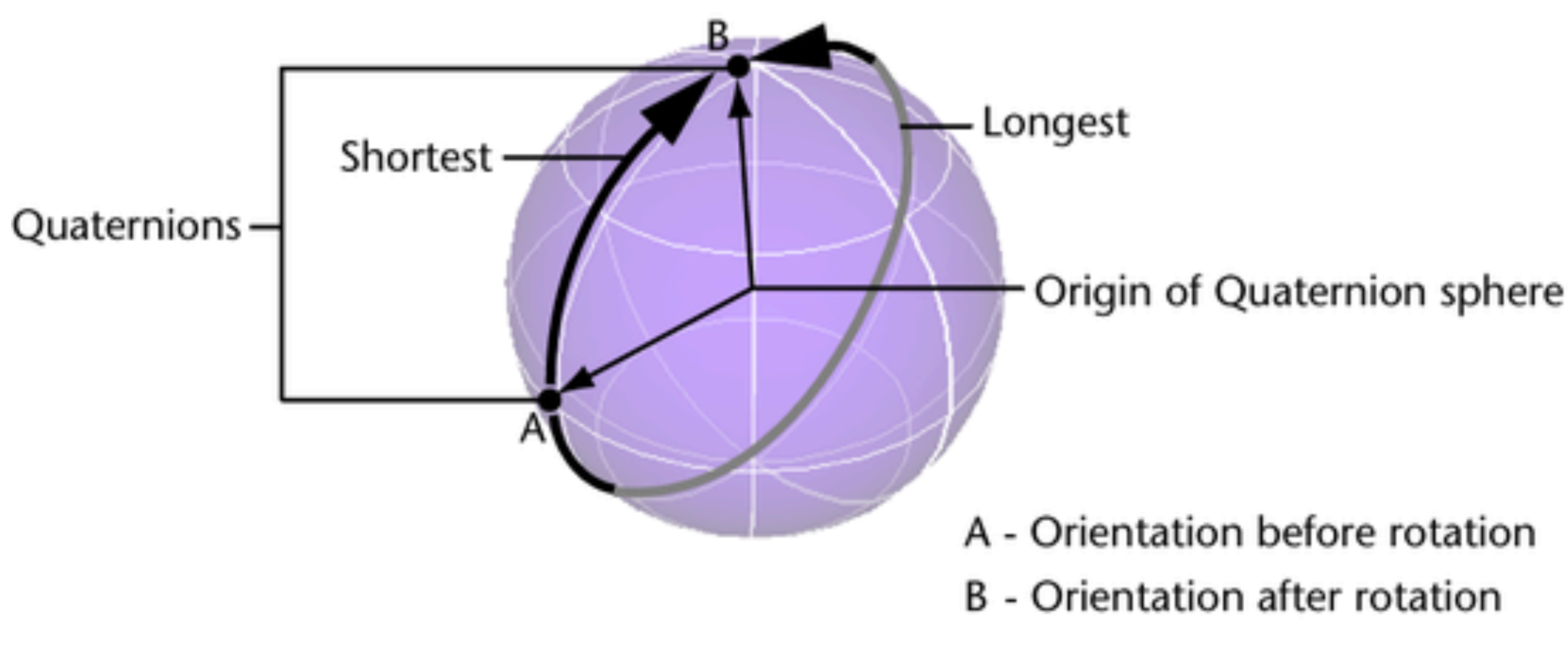
- Rotating a vector by a unit quaternion is faster than using a rotation matrix (a vector can be viewed as a non-unit quaternion with the scalar part set equal to zero)

$$\mathrm{Rotate}(\vec{u}) = q\vec{u}q^{-1}$$

- The inverse of a unit quaternion is simply its conjugate q^*
- (The inverse of a non-unit quaternion is $q^{-1} = q^*/\|q\|^2$)

Interpolating Unit Quaternions

- Unit quaternions can be viewed as points lying on a 4-D unit sphere: (s, x, y, z)
- Interpolating between these points means tracing out a curve on the surface of this 4-D sphere
- This framework allows us to take the shortest path as represented by an arc on the 4D sphere



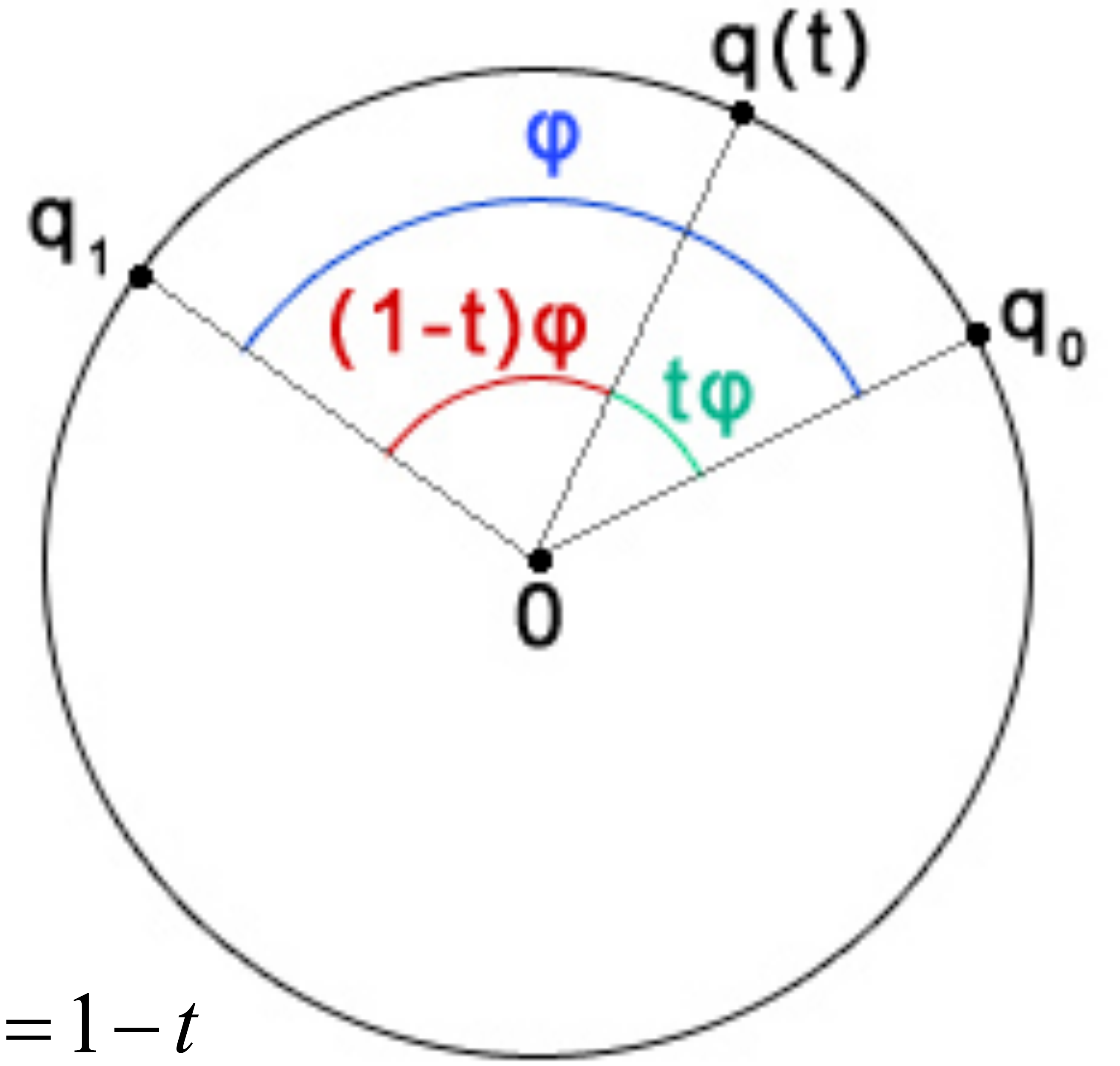
Quaternion rotation interpolation

Note: The 3D unit sphere is for illustrating the idea. The unit quaternion sphere is 4D!

SLERP

- **S**pherical **L**inear Int**erp**olation
  - Linearly interpolate between points q_0 and q_1 on the unit sphere:

$$q(t) = \frac{\sin(1-t)\varphi}{\sin\varphi} q_0 + \frac{\sin t\varphi}{\sin\varphi} q_1$$



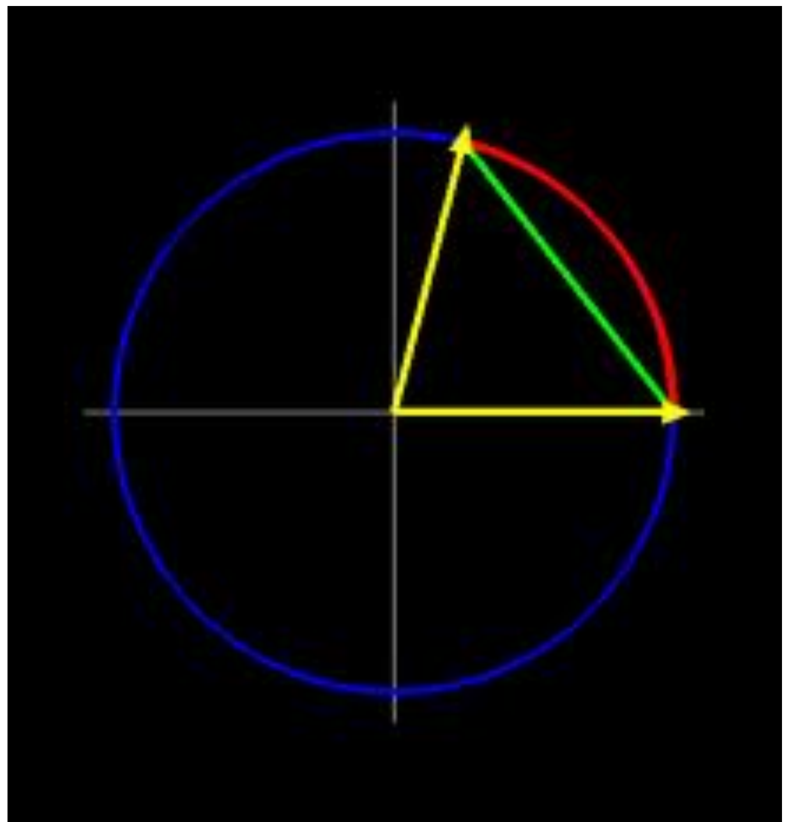
- According to L'Hôpital's rule,

$$\lim_{\varphi\to 0}\frac{\sin t\varphi}{\sin\varphi} = \lim_{\varphi\to 0}\frac{(\sin t\varphi)'}{(\sin\varphi)'} = \lim_{\varphi\to 0}\frac{t\cos t\varphi}{\cos\varphi} = t$$

$$\lim_{\varphi\to 0}\frac{\sin(1-t)\varphi}{\sin\varphi} = \lim_{\varphi\to 0}\frac{(\sin(1-t)\varphi)'}{(\sin\varphi)'} = \lim_{\varphi\to 0}\frac{(1-t)\cos(1-t)\varphi}{\cos\varphi} = 1-t$$

- Therefore, as φ goes to zero, we get linear interpolation

$$q(t) = (1-t)q_0 + tq_1$$



Angle Between Unit Quaternions

- The angle φ between two unit quaternions on a 4D sphere is calculated using:

$$\varphi = \arccos(q_0 \cdot q_1)$$

  with a typical dot-product: $q_0 \cdot q_1 = s_0 s_1 + x_0 x_1 + y_0 y_1 + z_0 z_1$

- φ is guaranteed to be between $[0, \pi]$
- However, it still does not guarantee the shortest path, because q and $-q$ correspond to the same rotation!
- So, if $q_0 \cdot q_1$ is negative, we negate either q_0 or q_1 before applying SLERP to guarantee the shortest path

SLERP for Unit Quaternions

- A quaternion $q = (s, \vec{v})$ can be defined in exponential form

$$q = \|q\| e^{\hat{n}\alpha} = \|q\| \left(\cos\alpha + \hat{n}\ \sin\alpha\right)$$

where α and the unit vector $\hat{n}$ are defined via:

$$s = \|q\| \cos\alpha\,, \qquad \vec{v} = \hat{n}\|\vec{v}\| = \|q\|\hat{n}\sin\alpha$$

- $\hat{n}$ is the rotation axis, and α equals half of the rotation angle
- The power of a quaternion is then: $q^t = \|q\|^t e^{\hat{n}\alpha t}$
- Note that the power affects the rotation angle α, but not the rotation axis $\hat{n}$

- Finally, SLERP for unit quaternions is expressed as:

$$SLERP(q_0, q_1; t) = q_0 ({q_0}^{-1} q_1)^{\,t}$$

Question #2

LONG FORM:
- Briefly explain why we use quaternions for rotations.
- Explain why moving rigid bodies use both a linear and an angular velocities.
- Answer the Short Form questions as well…

SHORT FORM
- Can you think of a particularly visually interesting rigid body used in games, movies, or television?
- Very briefly describe it.
- Explain how it makes use of translations and/or rotations for visual effects.