Animating Transformations

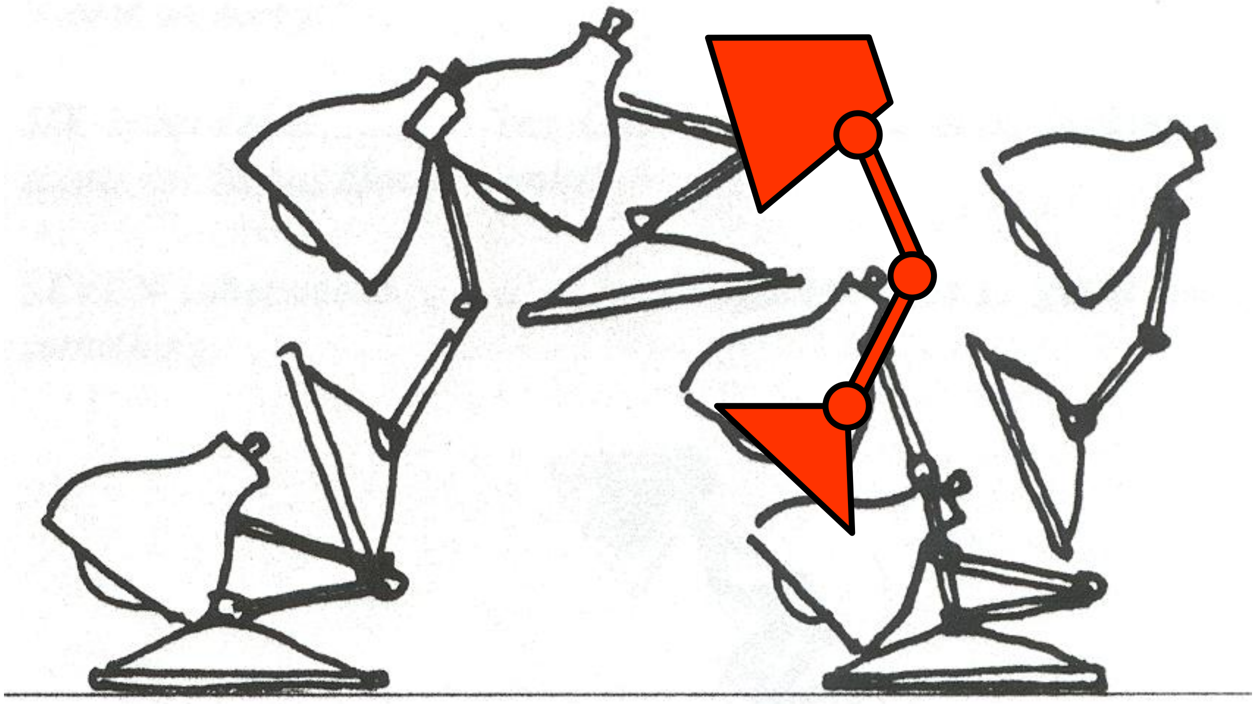
Michael Kazhdan

(601.457/657)

Recall

Keyframe Animation:

- Interpolate variables describing keyframes to determine poses for character “in-between”



Lasseter `87

Keyframe Animation

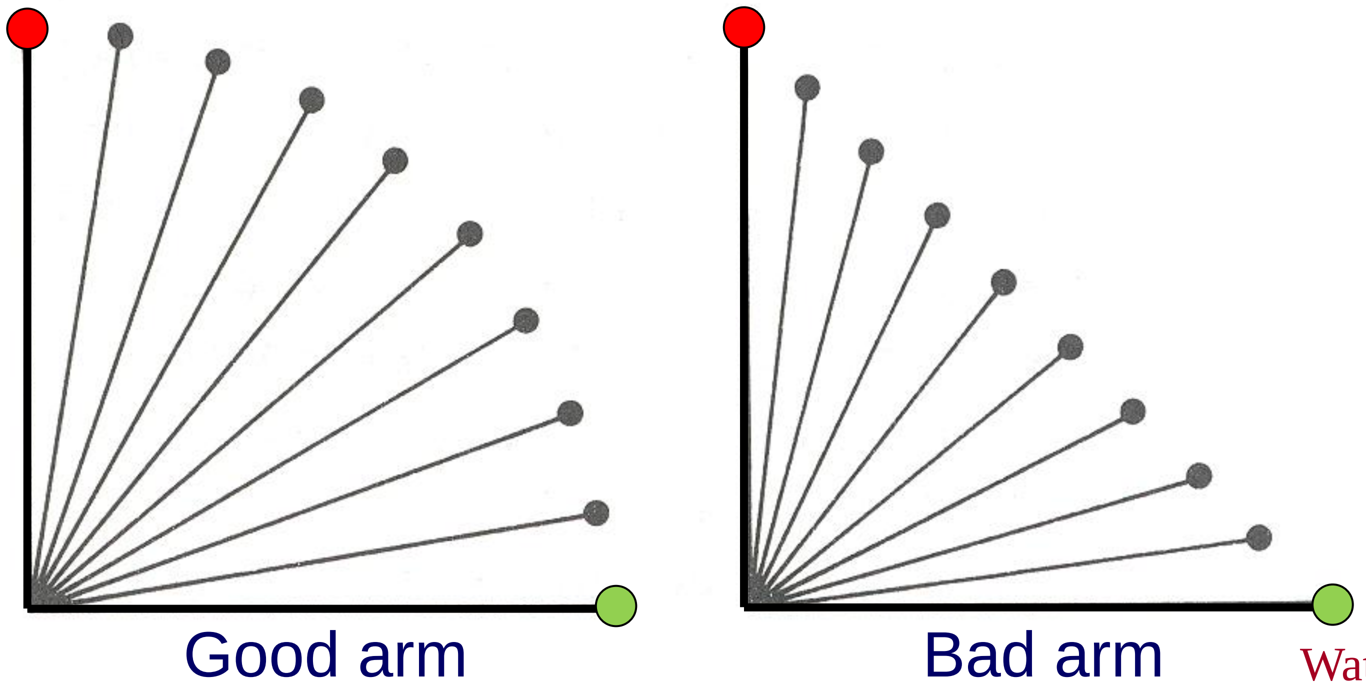
Q: Why interpolate/blend joint parameters instead of interpolating/blending vertices directly?

A: For translations, it doesn't make a difference (assuming the blend is translation equivariant).

Keyframe Animation

Q: Why interpolate/blend joint parameters instead of interpolating/blending vertices directly?

A: For rotations, it could lead to geometric distortion.



Watt & Watt

Keyframe Animation

Q: Why interpolate/blend joint parameters instead of interpolating/blending vertices directly?

A: For rotations, it could lead to geometric distortion.

For articulating objects, transformations are a combination of translation and rotation

- Translations are straight-forward:
  Use your favorite spline to fit a curve through/near the translational offsets
- How do we interpolate/approximate rotations?

Overview

- Orthogonal Transformations, Rotations, and SVD
- Interpolating/Approximating Points
  - Vectors
  - Unit-Vectors
- Interpolating/Approximating Transformations
  - Matrices
  - Rotations
    - » SVD Factorization
    - » Euler Angles

Orthogonal Transformations

What are orthogonal transformations?

An *orthogonal matrix* $\mathbf{O} \in \mathbb{R}^{n\times n}$ is a linear transformation that preserves angles:

$$\langle \mathbf{v}, \mathbf{w} \rangle = \langle \mathbf{O}\mathbf{v}, \mathbf{O}\mathbf{w} \rangle$$

Recall that the (standard) dot-product between two vectors can be expressed as a matrix multiplication:

$$\langle \mathbf{v}, \mathbf{w} \rangle = \mathbf{v}^\top \mathbf{w}$$

Orthogonal Transformations

<u>What are orthogonal transformations?</u>

An *orthogonal matrix* $\mathbf{O} \in \mathbb{R}^{n\times n}$ is a linear transformation that preserves angles:

$$\langle \mathbf{v}, \mathbf{w} \rangle = \langle \mathbf{O}\mathbf{v}, \mathbf{O}\mathbf{w} \rangle$$

$\Rightarrow$ If $\mathbf{O}$ is an orthogonal matrix:

$$\begin{aligned} \mathbf{v}^\top \mathbf{w} &= (\mathbf{O}\mathbf{v})^\top (\mathbf{O}\mathbf{w}) \\ &= \mathbf{v}^\top \mathbf{O}^\top \mathbf{O}\mathbf{w} \end{aligned}$$

Since this is true for all $\mathbf{v}$ and $\mathbf{w}$, this means that:

$$\mathbf{O}^\top \cdot \mathbf{O} = \text{identity} \quad \Leftrightarrow \quad \mathbf{O}^\top = \mathbf{O}^{-1}$$

Orthogonal Transformations

What are orthogonal transformations?

$$\mathbf{O}^\top \cdot \mathbf{O} = \text{identity}$$

For matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$:

- The determinant of the product is:

$$\det(\mathbf{A} \cdot \mathbf{B}) = \det(\mathbf{A}) \cdot \det(\mathbf{B})$$

- The determinant of the transpose is:

$$\det(\mathbf{A}) = \det(\mathbf{A}^\top)$$

It follows that:

$$\det(\mathbf{O}^\top \cdot \mathbf{O}) = \det(\text{identity})$$

$$\det(\mathbf{O}) \cdot \det(\mathbf{O}) = 1$$

$$\Downarrow$$

$$\det(\mathbf{O}) = \pm 1$$

Rotations

What are rotations?

A *rotation matrix* $\mathbf{O} \in \mathbb{R}^{n \times n}$ is:

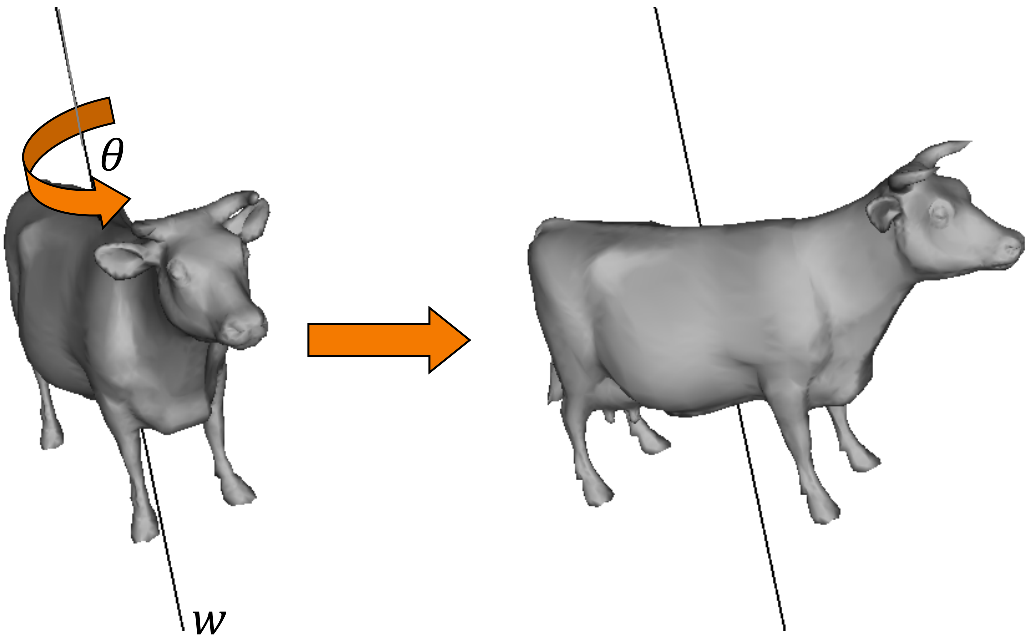
1. An orthogonal matrix
2. That preserves orientation (i.e. $\det(\mathbf{O}) = 1$).

Rotations

<u>What are rotations?</u>

A rotation in 3D can also be specified by:

- its axis of rotation w (with $\|w\| = 1$) and
- its angle of rotation θ



Rotations

What are rotations?

A rotation in 3D can also be specified by:

- its axis of rotation w (with $\|w\| = 1$) and
- its angle of rotation θ

Properties:

- The rotation corresponding to (θ, w) is the same as the rotation corresponding to $(-\theta, -w)$.
- The inverse of a rotation corresponding to (θ, w) is $(-\theta, w)$.
- Given rotations corresponding to (θ_1, w) and (θ_2, w), the product of the rotations corresponds to $(\theta_1 + \theta_2, w)$.

How do we define the product of rotations corresponding to (θ_1, w_1) and (θ_2, w_2)?

SVD

Any $n \times n$ matrix $\mathbf{M}$ can be expressed in terms of its <u>S</u>ingular <u>V</u>alue <u>D</u>ecomposition as:

$$\mathbf{M} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$$

where:

- $\mathbf{U}$ and $\mathbf{V}$ are $n \times n$ orthogonal matrices
- $\mathbf{D}$ is an $n \times n$ diagonal matrix (i.e. off-diagonals are 0)
  - » Typically the diagonal entries are:
    - – Non-negative
    - – Decreasing

SVD

Applications:

- Procrustes method
- Finding the (pseudo-)inverse of a matrix
- Compression

SVD

Finding the Inverse of a Matrix:

If we have an $n \times n$ invertible matrix $\mathbf{M}$, we can use the SVD to compute the inverse of $\mathbf{M}$.

Expressing $\mathbf{M}$ in terms of its SVD gives:

$$\mathbf{M} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$$

where:

- $\mathbf{U}$ and $\mathbf{V}$ are $n \times n$ orthogonal matrix,
- $\mathbf{D}$ is an $n \times n$ diagonal matrix

SVD

Finding the Inverse of a Matrix :

$$\mathbf{M} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$$

We can express $\mathbf{M}^{-1}$ as:

$$\mathbf{M}^{-1} = (\mathbf{U}\mathbf{D}\mathbf{V}^\top)^{-1} = (\mathbf{V}^\top)^{-1}\mathbf{D}^{-1}\mathbf{U}^{-1}$$
$$= \mathbf{V}\mathbf{D}^{-1}\mathbf{U}^\top$$

Since:

- $\mathbf{U}$ is an orthogonal transformation, $\mathbf{U}^{-1} = \mathbf{U}^\top$.
- $\mathbf{V}$ is an orthogonal transformation, $\mathbf{V}^{-1} = \mathbf{V}^\top$.

SVD

Solving Linear Systems:

$$\mathbf{M}^{-1} = \mathbf{V}\mathbf{D}^{-1}\mathbf{U}^{\top}$$

Since $\mathbf{D}$ is a diagonal matrix:

$$\mathbf{D} = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 & 0 \\ 0 & \lambda_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \lambda_{n-1} & 0 \\ 0 & 0 & \cdots & 0 & \lambda_n \end{pmatrix} \quad \Rightarrow \quad \mathbf{D}^{-1} = \begin{pmatrix} \frac{1}{\lambda_1} & 0 & \cdots & 0 & 0 \\ 0 & \frac{1}{\lambda_2} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\lambda_{n-1}} & 0 \\ 0 & 0 & \cdots & 0 & \frac{1}{\lambda_n} \end{pmatrix}$$

Note that this is not necessarily an *efficient* way to invert a matrix.

Overview

- Orthogonal Transformations, Rotations, and SVD
- Interpolating/Approximating Points
  - Vectors
  - Unit-Vectors
- Interpolating/Approximating Transformations
  - Matrices
  - Rotations
    - » SVD Factorization
    - » Euler Angles

Vectors

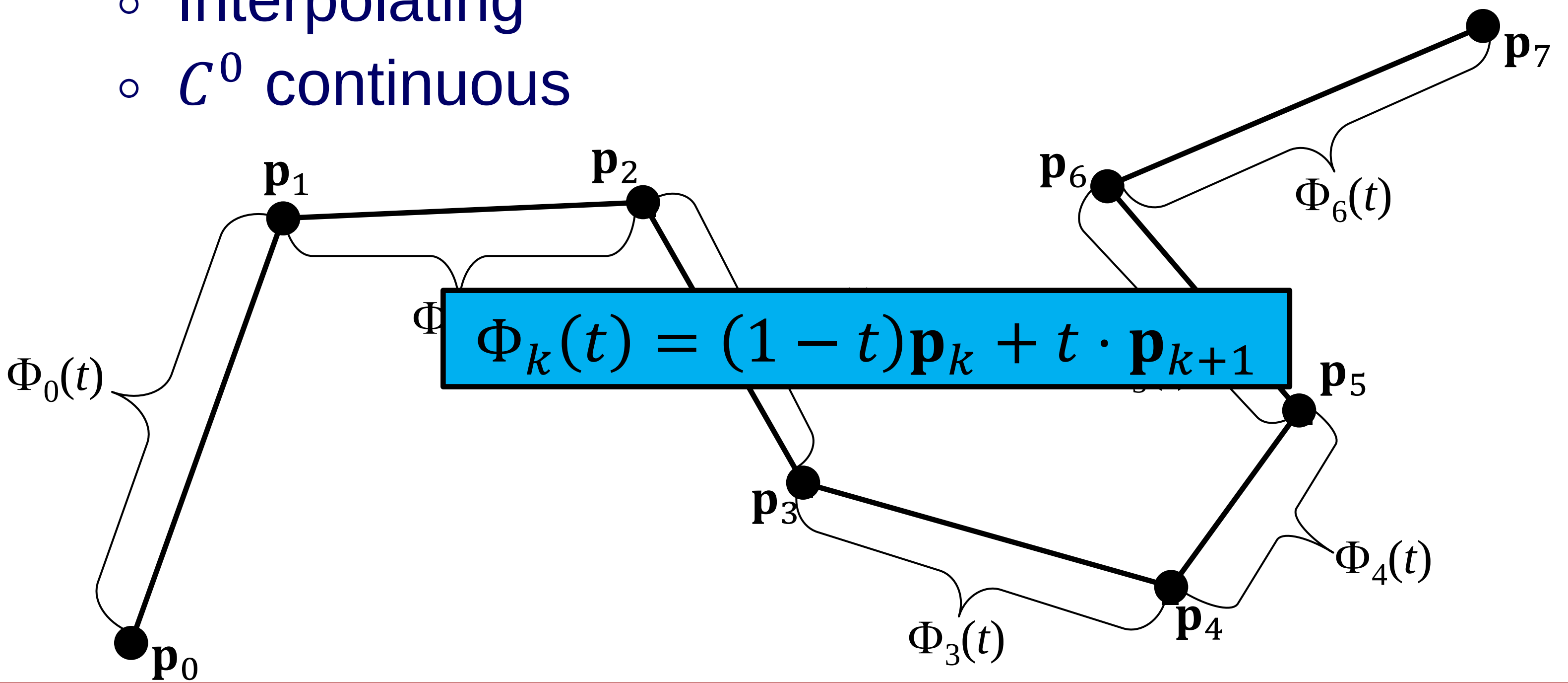
Given a collection of n control points $\{\mathbf{p}_0, \ldots, \mathbf{p}_{n-1}\}$, define a curve $\Phi(t)$ that approximates/interpolates the points.

Vectors

Given a collection of n control points $\{\mathbf{p}_0, \ldots, \mathbf{p}_{n-1}\}$, define a curve $\Phi(t)$ that approximates/interpolates the points.

<u>Linear Interpolation</u>:

- Interpolating
- C^0 continuous



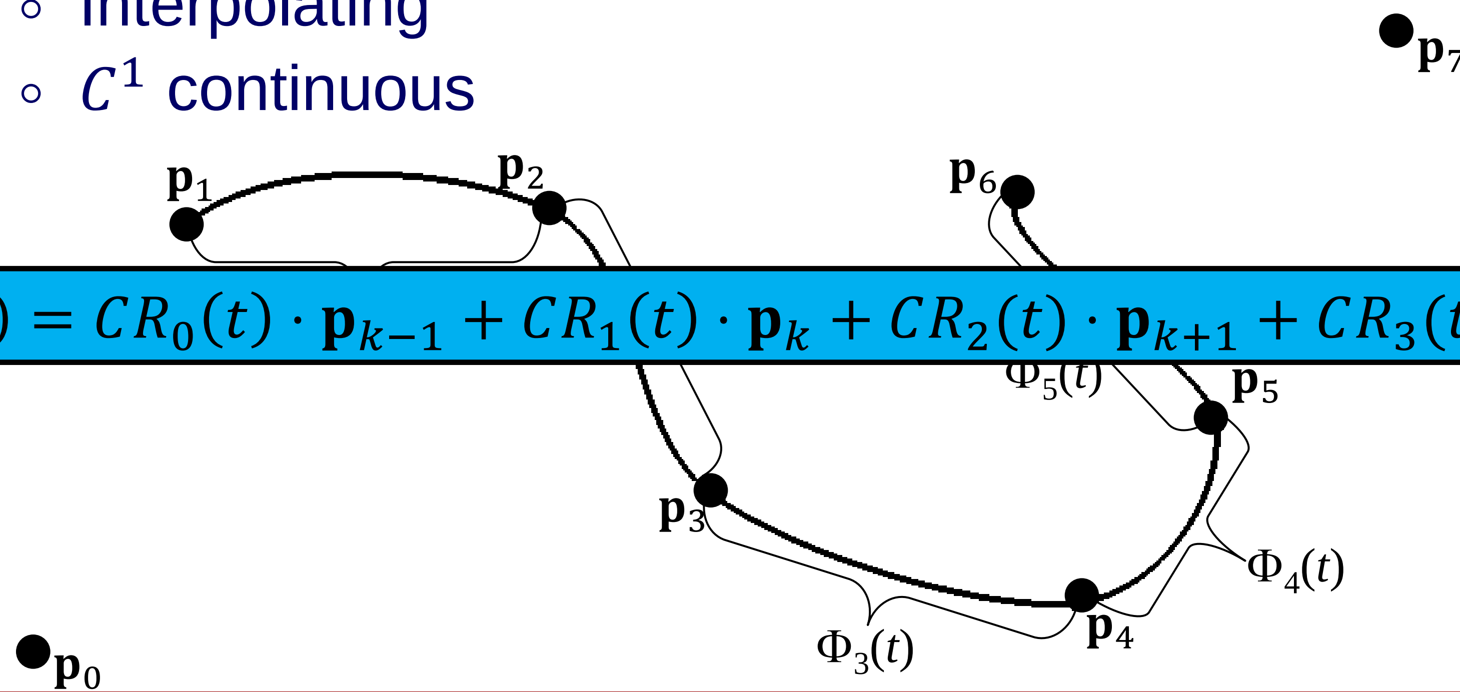
$$\Phi_k(t) = (1-t)\mathbf{p}_k + t \cdot \mathbf{p}_{k+1}$$

Vectors

Given a collection of n control points $\{\mathbf{p}_0, \ldots, \mathbf{p}_{n-1}\}$, define a curve $\Phi(t)$ that approximates/interpolates the points.

Catmull-Rom Splines (Cardinal Splines w/ $s = {}^{1}/{}_{2}$):

- Interpolating
- C^1 continuous



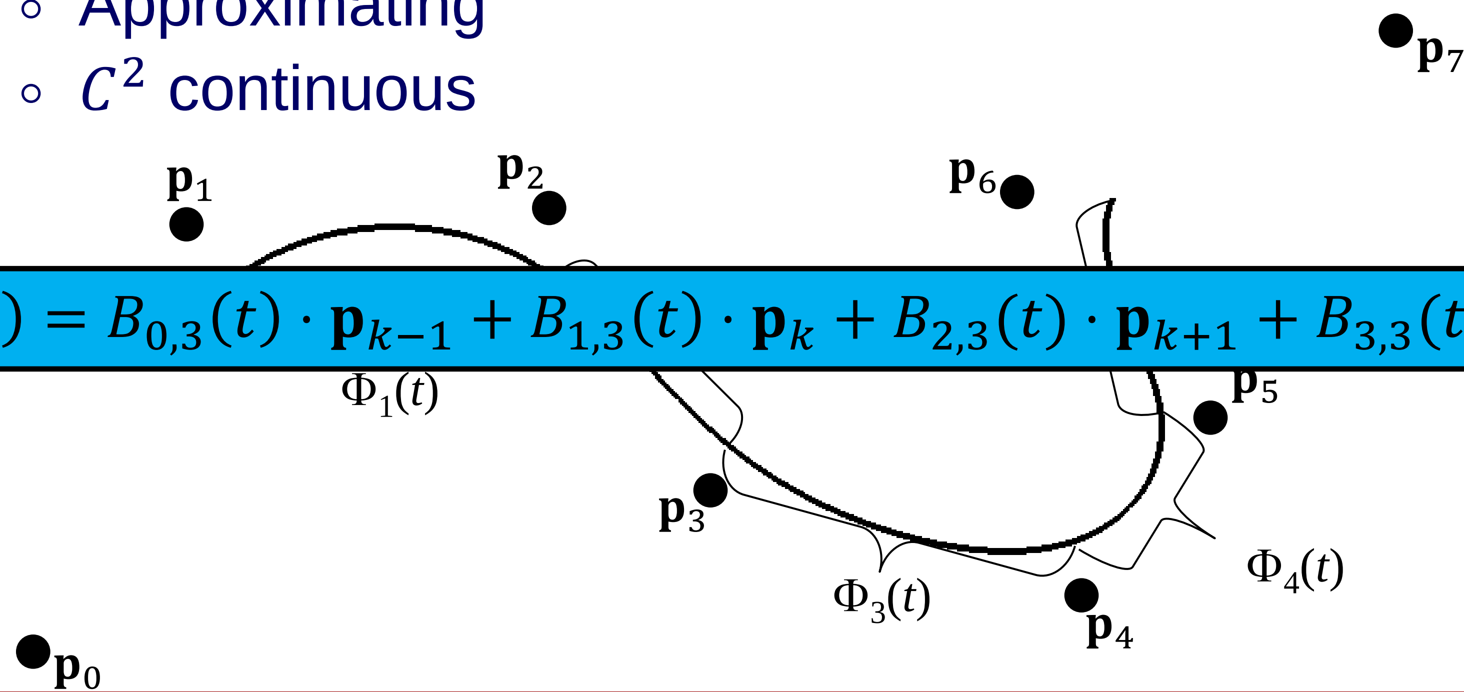
$$\Phi_k(t) = CR_0(t) \cdot \mathbf{p}_{k-1} + CR_1(t) \cdot \mathbf{p}_k + CR_2(t) \cdot \mathbf{p}_{k+1} + CR_3(t) \cdot \mathbf{p}_{k+2}$$

Vectors

Given a collection of n control points $\{\mathbf{p}_0, \ldots, \mathbf{p}_{n-1}\}$, define a curve $\Phi(t)$ that approximates/interpolates the points.

<u>Uniform Cubic B-Splines</u>:

- Approximating
- C^2 continuous



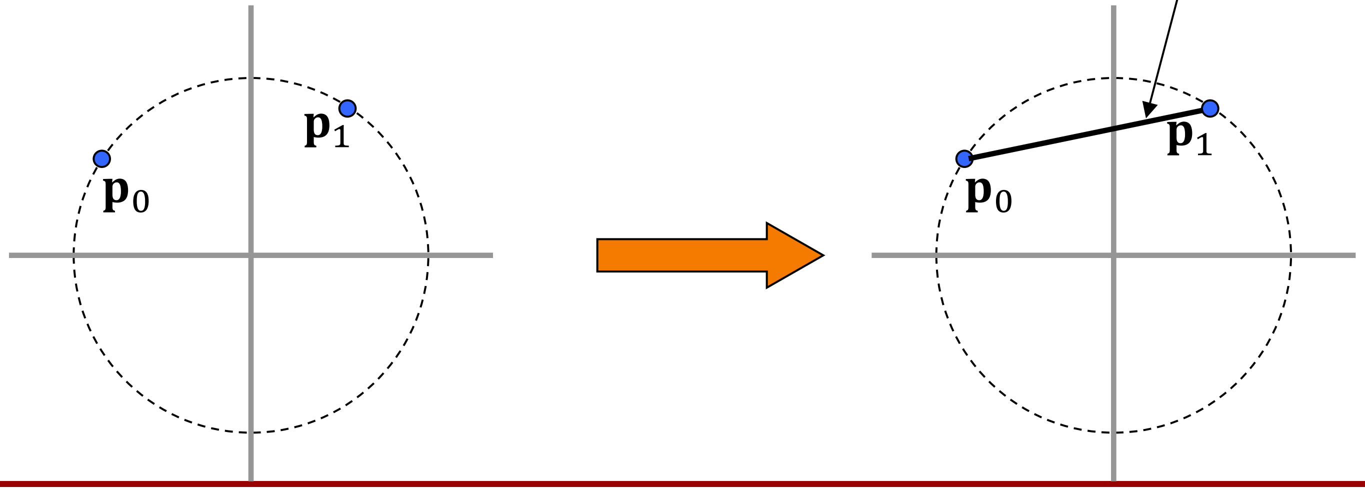
$$\Phi_k(t) = B_{0,3}(t) \cdot \mathbf{p}_{k-1} + B_{1,3}(t) \cdot \mathbf{p}_k + B_{2,3}(t) \cdot \mathbf{p}_{k+1} + B_{3,3}(t) \cdot \mathbf{p}_{k+2}$$

Unit-Vectors

What if we add the constraint that the points $\{\mathbf{p_0}, \ldots, \mathbf{p}_{n-1}\}$ and the curve $\Phi(t)$ have to lie on the unit circle/sphere ($\|\mathbf{p}_i\| = 1$, $\|\Phi(t)\| = 1$)?

We can't interpolate/approximate the points as before, because the in-between points don't have to lie on the unit circle/sphere!

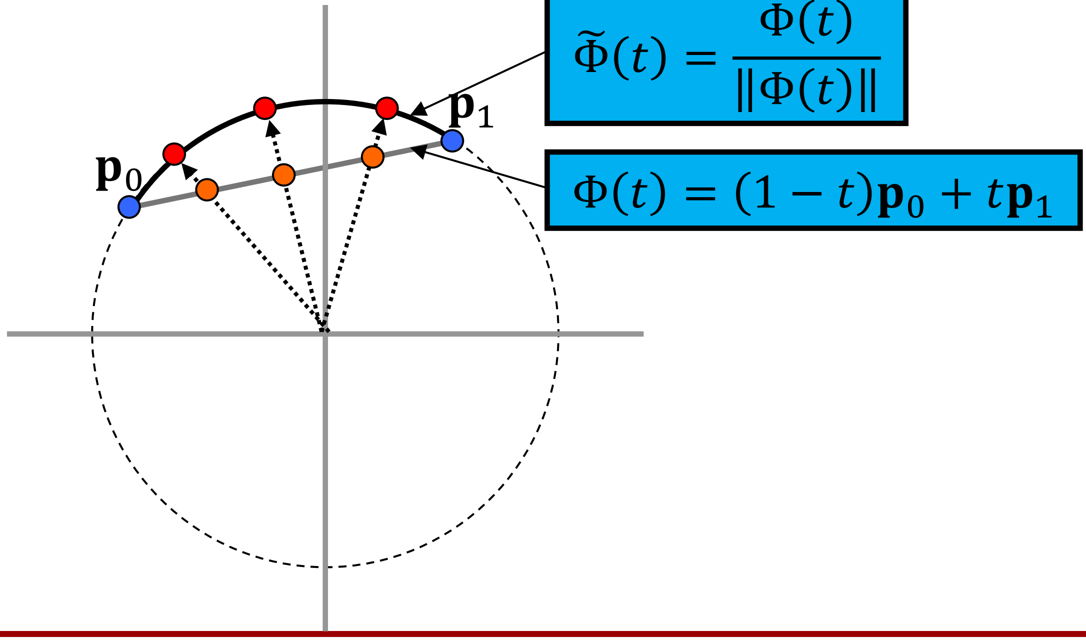
$\boxed{\Phi(t) = (1-t)\mathbf{p}_0 + t\mathbf{p}_1}$



Unit-Vectors

What if we add the constraint that the points $\{\mathbf{p_0},\dots,\mathbf{p}_{n-1}\}$ and the curve $\Phi(t)$ have to lie on the unit circle/sphere ($\|\mathbf{p}_i\| = 1$, $\|\Phi(t)\| = 1$)?

We can normalize the in-between points by sending them to the *closest* circle/sphere point:



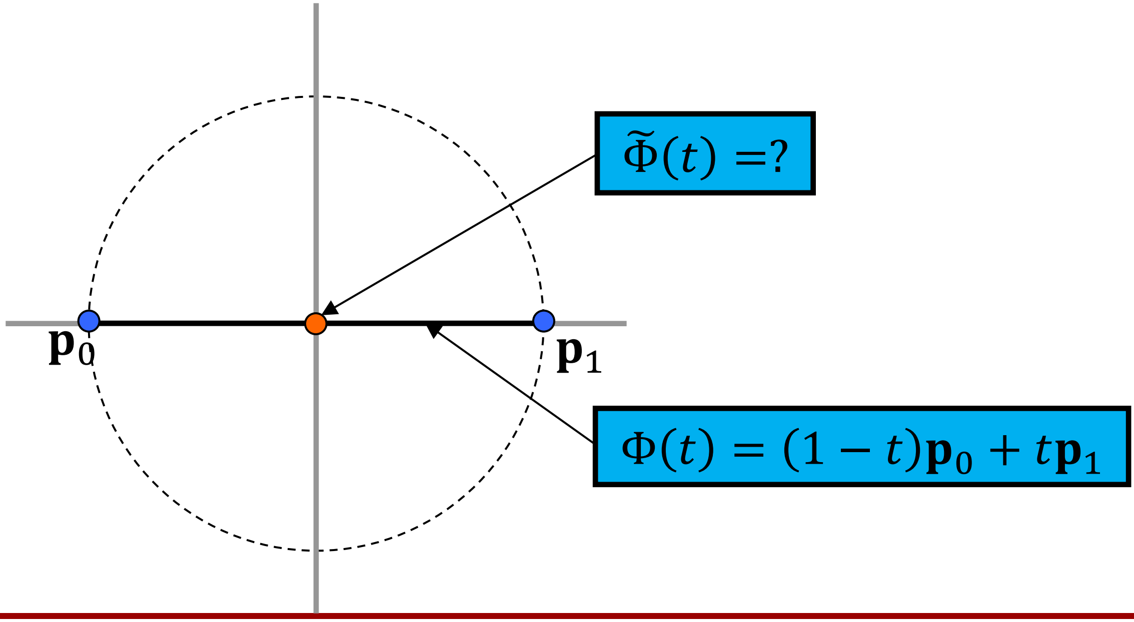
Curve Normalization

Limitations:

Curve Normalization

Limitations:

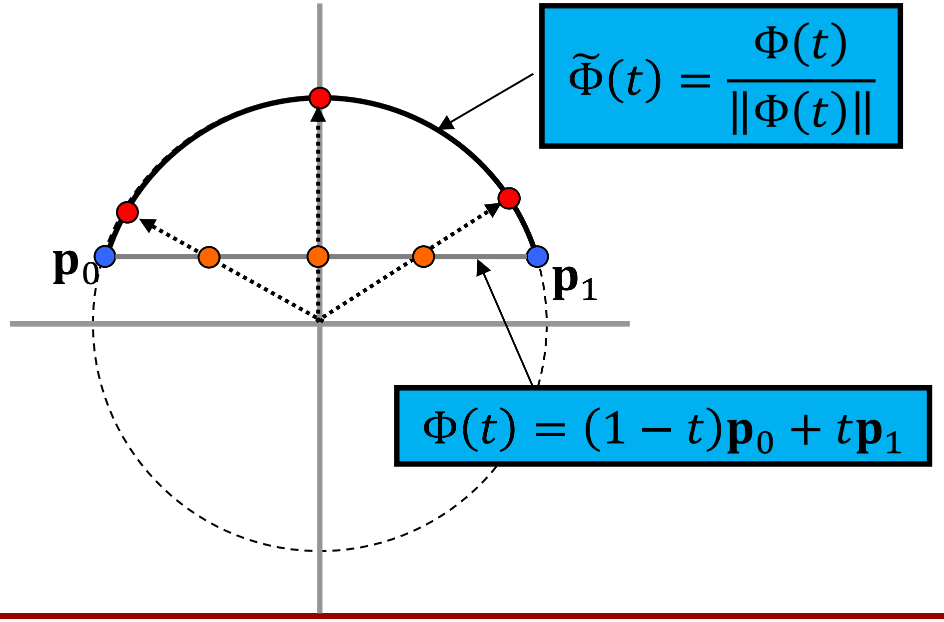
- The normalized curve is not always well defined.



Curve Normalization

<u>Limitations:</u>

- The normalized curve is not always well defined.
- Just because points are uniformly distributed on the original curve, does not mean that they will be uniformly distributed on the normalized one.

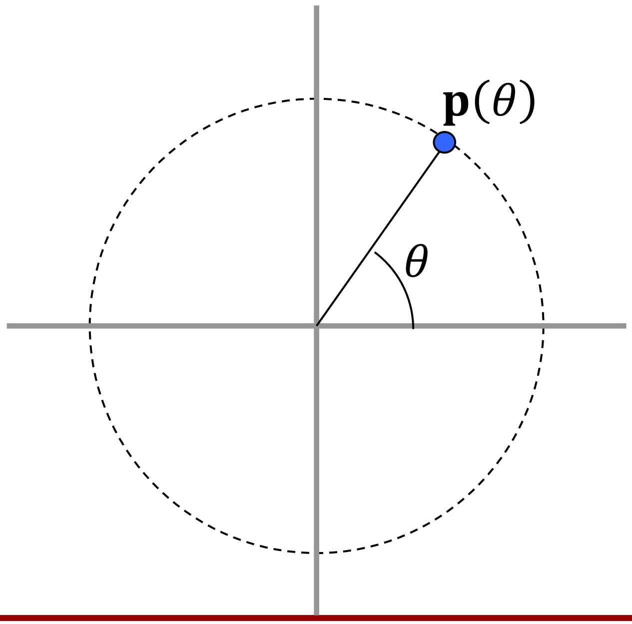


Curve Parameterization

- Define a parameterization of the circle/sphere.
- Compute the parameters of the end-points;
- Blend the parameters and evaluate.

SLERP (Spherical Linear Interpolation):

- Parameterize: $(\cos\theta, \sin\theta)$



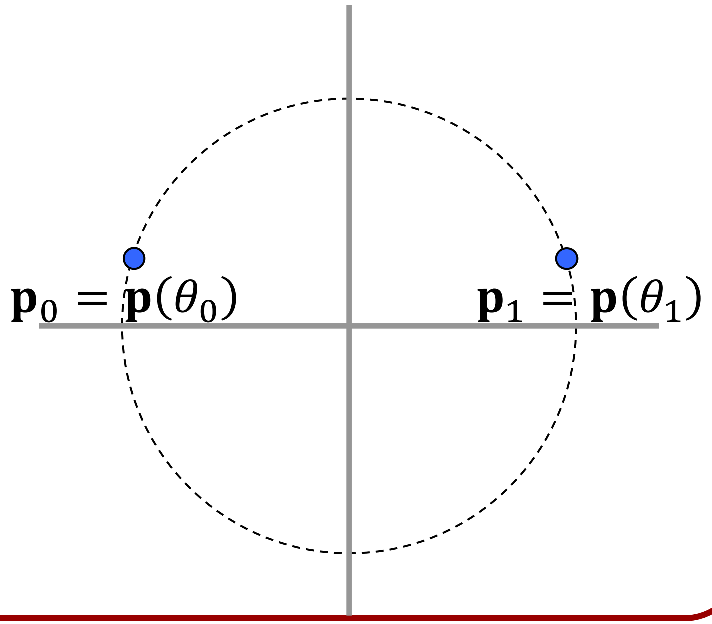
Curve Parameterization

- Define a parameterization of the circle/sphere.
- Compute the parameters of the end-points;
- Blend the parameters and evaluate.

SLERP (Spherical Linear Interpolation):

- Parameterize: $(\cos\theta\,,\sin\theta)$
- Compute:

$$\mathbf{p}_0 = (\cos\theta_0, \sin\theta_0)$$
$$\mathbf{p}_1 = (\cos\theta_1\,, \sin\theta_1)$$



Curve Parameterization

- Define a parameterization of the circle/sphere.
- Compute the parameters of the end-points;
- Blend the parameters and evaluate.

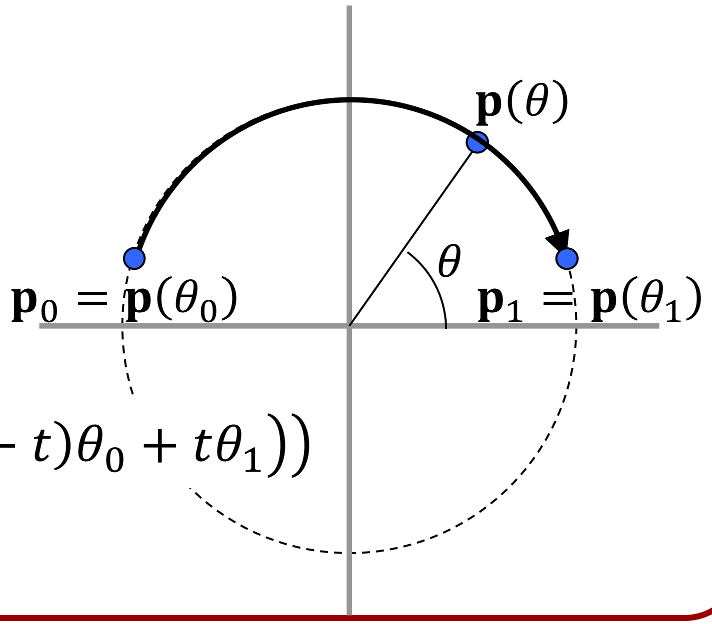
SLERP (Spherical Linear Interpolation):

- Parameterize: $(\cos\theta, \sin\theta)$
- Compute:

$$\mathbf{p}_0 = (\cos\theta_0, \sin\theta_0)$$
$$\mathbf{p}_1 = (\cos\theta_1, \sin\theta_1)$$

- Set:

$$\Phi(t) = \big(\cos\big((1-t)\theta_0 + t\theta_1\big), \sin\big((1-t)\theta_0 + t\theta_1\big)\big)$$



Curve Parameterization

Note:

- Parameter may not be unique.
- A shortest path in parameter space does not have to define a shortest path in the space of rotations.

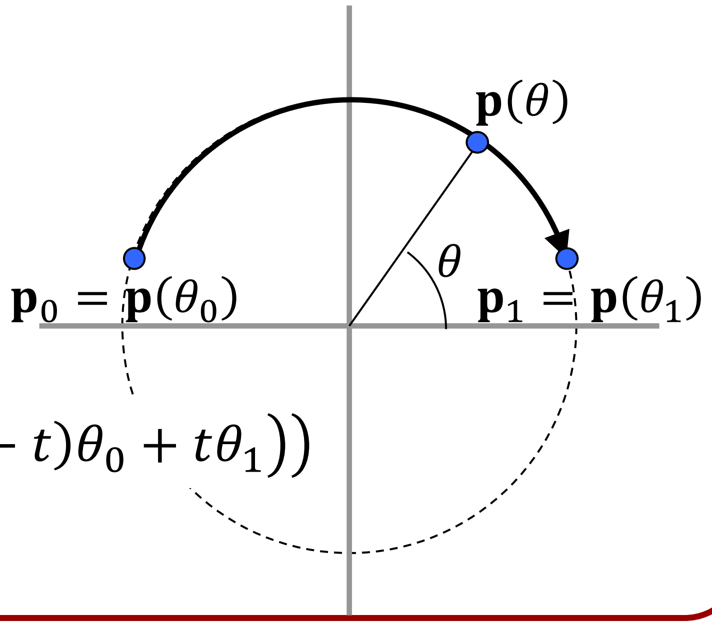
SLERP (Spherical Linear Interpolation):

- Parameterize: $(\cos\theta, \sin\theta)$
- Compute:

$$\mathbf{p}_0 = (\cos\theta_0, \sin\theta_0)$$
$$\mathbf{p}_1 = (\cos\theta_1, \sin\theta_1)$$

- Set:

$$\Phi(t) = \big(\cos\big((1-t)\theta_0 + t\theta_1\big), \sin\big((1-t)\theta_0 + t\theta_1\big)\big)$$



Overview

Interpolating/Approximating

- Orthogonal Transformations, Rotations, and SVD
- Interpolating/Approximating Points
  - Vectors
  - Unit-Vectors
- Interpolating/Approximating Transformations
  - Matrices
  - Rotations
    - » SVD Factorization
    - » Euler Angles

Matrices

Given a collection of n matrices $\{\mathbf{M}_0, \ldots, \mathbf{M}_{n-1}\}$, define a curve $\Phi(t)$ that approximates/interpolates the matrices.

Matrices

Given a collection of n matrices $\{\mathbf{M}_0, \ldots, \mathbf{M}_{n-1}\}$, define a curve $\Phi(t)$ that approximates/interpolates the matrices.

<u>As with vectors:</u>

- Linear Interpolation:

$$\Phi_k(t) = (1-t)\mathbf{M}_k + t \cdot \mathbf{M}_{k+1}$$

- Catmull-Rom Interpolation:

$$\Phi_k(t) = CR_0(t) \cdot \mathbf{M}_{k-1} + CR_1(t) \cdot \mathbf{M}_k + CR_2(t) \cdot \mathbf{M}_{k+1} + CR_3(t) \cdot \mathbf{M}_{k+2}$$

- Uniform Cubic B-Spline Approximation:

$$\Phi_k(t) = B_{0,3}(t) \cdot \mathbf{M}_{k-1} + B_{1,3}(t) \cdot \mathbf{M}_k + B_{2,3}(t) \cdot \mathbf{M}_{k+1} + B_{3,3}(t) \cdot \mathbf{M}_{k+2}$$

Rotations

What if we add the constraint that the matrices $\{\mathbf{M}_0, \ldots, \mathbf{M}_{n-1}\}$ and the values of the curve $\Phi(t)$ have to be rotations?

We can't interpolate/approximate the matrices as before, because the in-between matrices don't have to be rotations!

We could try to normalize, by mapping every matrix $\Phi(t)$ to the nearest rotation.

Challenge

Given a matrix $\mathbf{M}$, what is the closest rotation $\mathbf{R}$?

Normalization: SVD Factorization

Given a matrix $\mathbf{M}$, what is the closest rotation $\mathbf{R}$?

Singular Value Decomposition (SVD) allows us to express $\mathbf{M}$ as a diagonal matrix, multiplied on the left/right by orthogonal transformations $\mathbf{O}_1/\mathbf{O}_2$:

$$\mathbf{M} = \mathbf{O}_1 \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \mathbf{O}_2$$

Because the λ_i are positive, the closest orthogonal transform $\mathbf{O}$ to $\mathbf{M}$ is:

$$\mathbf{O} = \mathbf{O}_1 \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{O}_2$$

Normalization: SVD Factorization

Given a matrix $\mathbf{M}$, what is the closest rotation $\mathbf{R}$?

Singular Value Decomposition (SVD) allows us to

ex

le

In standard SVD factorization, the diagonal values are non-negative, and ordered from largest to smallest.

The orthogonal transformations $\mathbf{O}_1$ and $\mathbf{O}_2$ are not necessarily rotations.

To get a rotation, we must make the product have determinant 1.

Because the λ_i are positive, the closest orthogonal transform $\mathbf{O}$ to $\mathbf{M}$ is:

$$\mathbf{O} = \mathbf{O}_1 \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{O}_2$$

Normalization: SVD Factorization

Given a matrix $\mathbf{M}$, what is the closest rotation $\mathbf{R}$?

<u>Singular Value Decomposition (SVD)</u> allows us to

ex

le

In standard SVD factorization, the diagonal values are non-negative, and ordered from largest to smallest.

The orthogonal transformations $\mathbf{O}_1$ and $\mathbf{O}_2$ are not necessarily rotations.

To get a rotation, we must make the product have determinant 1.

Because the λ_i are positive and decreasing, the closest orthogonal transform $\mathbf{O}$ to $\mathbf{M}$ is:

$$\mathbf{R} = \mathbf{O}_1 \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(\mathbf{O}_1 \cdot \mathbf{O}_2) \end{pmatrix} \mathbf{O}_2$$

Parametrization: Euler Angles

Every rotation matrix $\mathbf{R}$ can be parameterized by:

- a rotation about the z-axis, multiplied by
- a rotation about the y-axis, multiplied by
- a rotation about the z-axis:

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

The angles $(\theta,\phi,\psi) \in [0,2\pi) \times [0,\pi] \times [0,2\pi)$ are called the Euler angles.

Parametrization: Euler Angles

Instead of blending matrices and then normalizing, we can blend the rotation parameters:

- For each $\mathbf{M}_k$, compute the Euler angles $(\theta_k, \phi_k, \psi_k)$

Parametrization: Euler Angles

Instead of blending matrices and then normalizing, we can blend the rotation parameters:

- For each $\mathbf{M}_k$, compute the Euler angles $(\theta_k, \phi_k, \psi_k)$
- Interpolate/Approximate the Euler angles:

Parametrization: Euler Angles

Instead of blending matrices and then normalizing, we can blend the rotation parameters:

- For each $\mathbf{M}_k$, compute the Euler angles $(\theta_k, \phi_k, \psi_k)$
- Interpolate/Approximate the Euler angles:
  - » Linear Interpolation:
    - $\theta_k(t) = (1-t)\theta_k + t \cdot \theta_{k+1}$
    - $\phi_k(t) = (1-t)\phi_k + t \cdot \phi_{k+1}$
    - $\psi_k(t) = (1-t)\psi_k + t \cdot \psi_{k+1}$

Parametrization: Euler Angles

Instead of blending matrices and then normalizing, we can blend the rotation parameters:

- For each $\mathbf{M}_k$, compute the Euler angles $(\theta_k, \phi_k, \psi_k)$
- Interpolate/Approximate the Euler angles:
  - » Linear Interpolation
  - » Catmull-Rom Interpolation:
    - $\theta_k(t) = CR_0(t) \cdot \theta_{k-1} + CR_1(t) \cdot \theta_k + CR_2(t) \cdot \theta_{k+1} + CR_3(t) \cdot \theta_{k+2}$
    - $\phi_k(t) = CR_0(t) \cdot \phi_{k-1} + CR_1(t) \cdot \phi_k + CR_2(t) \cdot \phi_{k+1} + CR_3(t) \cdot \phi_{k+2}$
    - $\psi_k(t) = CR_0(t) \cdot \psi_{k-1} + CR_1(t) \cdot \psi_k + CR_2(t) \cdot \psi_{k+1} + CR_3(t) \cdot \psi_{k+2}$

Parametrization: Euler Angles

Instead of blending matrices and then normalizing, we can blend the rotation parameters:

- For each $\mathbf{M}_k$, compute the Euler angles $(\theta_k, \phi_k, \psi_k)$
- Interpolate/Approximate the Euler angles:
  - » Linear Interpolation
  - » Catmull-Rom Interpolation
  - » Uniform Cubic B-Spline Approximation:
    - $\theta_k(t) = B_{0,3}(t) \cdot \theta_{k-1} + B_{1,3}(t) \cdot \theta_k + B_{2,3}(t) \cdot \theta_{k+1} + B_{3,3}(t) \cdot \theta_{k+2}$
    - $\phi_k(t) = B_{0,3}(t) \cdot \phi_{k-1} + B_{1,3}(t) \cdot \phi_k + B_{2,3}(t) \cdot \phi_{k+1} + B_{3,3}(t) \cdot \phi_{k+2}$
    - $\psi_k(t) = B_{0,3}(t) \cdot \psi_{k-1} + B_{1,3}(t) \cdot \psi_k + B_{2,3}(t) \cdot \psi_{k+1} + B_{3,3}(t) \cdot \psi_{k+2}$

Parametrization: Euler Angles

Instead of blending matrices and then normalizing, we can blend the rotation parameters:

- For each $\mathbf{M}_k$, compute the Euler angles $(\theta_k, \phi_k, \psi_k)$
- Interpolate/Approximate the Euler angles:
  - » Linear Interpolation
  - » Catmull-Rom Interpolation
  - » Uniform Cubic B-Spline Approximation
- Set the value of the in-between matrix to:

$$\Phi_k(t) = \mathbf{R}_z\big(\theta_k(t)\big)\mathbf{R}_y\big(\phi_k(t)\big)\mathbf{R}_z\big(\psi_k(t)\big)$$

Translations and Rotations

We represent translations and rotations independently, allowing us to:

- Blend the translations with splines
- Blend rotations with splines and
  - Normalization, or
  - Parametrization

There are other approaches that treat translations and rotations in a unified manner:

Dual Quaternions for Rigid Transformation Blending, [Kavan *et al*., 2006]