



Animating Transformations

Michael Kazhdan
(600.357 / 600.457)

HB A.6

FvDFH 21.1.3



Overview

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



Rotations

What are rotations?



Rotations

What are rotations?

- A rotation R is a linear transformation that has determinant equal to one and preserves the angle between any two vectors:

$$\langle v, w \rangle = \langle R(v), R(w) \rangle$$



Rotations

What are rotations?

- A rotation R is a linear transformation that has determinant equal to one and preserves the angle between any two vectors:

$$\langle v, w \rangle = \langle R(v), R(w) \rangle$$

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

$$\langle v, w \rangle = v^t w$$



Rotations

What are rotations?

- A rotation R is a linear transformation that has determinant equal to one and preserves the angle between any two vectors:

$$\langle v, w \rangle = \langle R(v), R(w) \rangle$$

This implies that:

$$v^t w = (Rv)^t (Rw)$$



Rotations

What are rotations?

- A rotation R is a linear transformation that has determinant equal to one and preserves the angle between any two vectors:

$$\langle v, w \rangle = \langle R(v), R(w) \rangle$$

This implies that:

$$\begin{aligned} v^t w &= (Rv)^t (Rw) \\ &= v^t (R^t R) w \end{aligned}$$



Rotations

What are rotations?

- A rotation R is a linear transformation that has determinant equal to one and preserves the angle between any two vectors:

$$\langle v, w \rangle = \langle R(v), R(w) \rangle$$

This implies that:

$$\begin{aligned} v^t w &= (Rv)^t (Rw) \\ &= v^t (R^t R) w \end{aligned}$$

Since this is true for all v and w , this means that:

$$R^t R = \text{Identity} \quad \longleftrightarrow \quad R^t = R^{-1}$$



Rotations

What are rotations?

- A rotation R is a linear transformation that has determinant equal to one and preserves the angle between any two vectors.
- A rotation R is a linear transformation that has determinant equal to one and whose transpose is equal to its inverse.



Rotations

What are rotations?

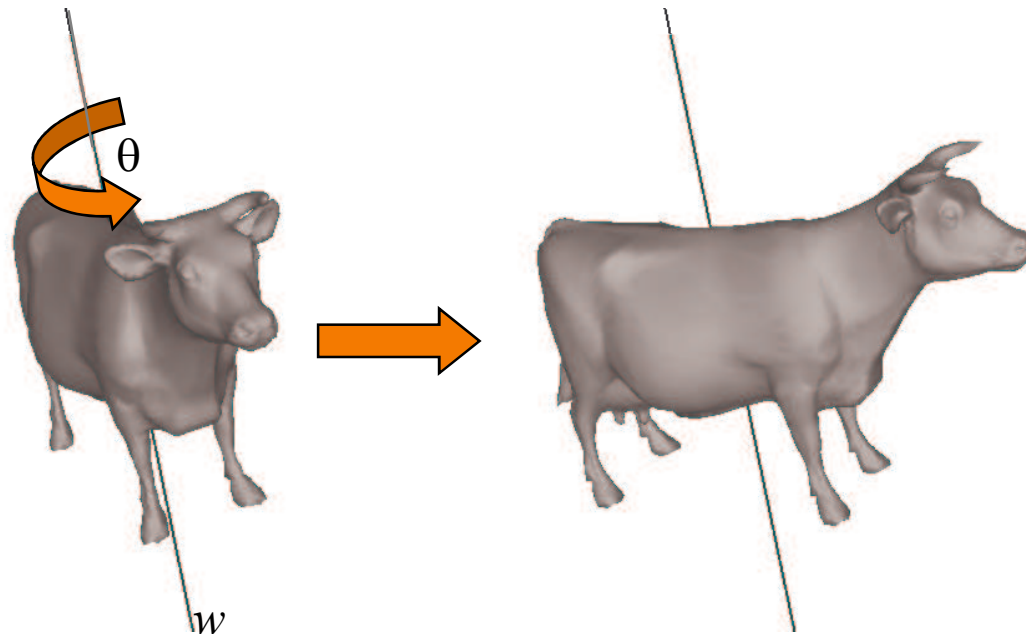
- A rotation R is a linear transformation that has determinant equal to one and preserves the angle between any two vectors.
- A rotation R is a linear transformation that has determinant equal to one and whose transpose is equal to its inverse.
- A rotation in 3D can be specified by a 3x3 matrix.



Rotations

What are rotations?

- A rotation in 3D can also be specified by:
 - its axis of rotation w ($\|w\|=1$) and
 - its angle of rotation θ





Rotations

What are rotations?

- A rotation in 3D can also be specified by:
 - its axis of rotation w ($\|w\|=1$) and
 - its angle of rotation θ

Properties:

- The rotation corresponding to (θ, w) is the same as the rotation corresponding to $(-\theta, -w)$.



Rotations

What are rotations?

- A rotation in 3D can also be specified by:
 - its axis of rotation w ($\|w\|=1$) and
 - its angle of rotation θ

Properties:

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



Rotations

What are rotations?

- A rotation in 3D can also be specified by:
 - its axis of rotation w ($\|w\|=1$) and
 - its angle of rotation θ

Properties:

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



Rotations

What are rotations?

- A rotation in 3D can also be specified by:
 - its axis of rotation w ($\|w\|=1$) and
 - its angle of rotation θ

Properties:

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

ra

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



SVD

Any $m \times n$ matrix M can be expressed in terms of its Singular Value Decomposition as:

$$M = UDV^t$$

where:

- U is an $n \times n$ rotation matrix,
- V is an $m \times m$ rotation matrix, and
- D is an $m \times n$ diagonal matrix (i.e off-diagonals are all 0).



SVD

Applications:

- Compression
- Model Alignment
- Matrix Inversion
- Solving Over-Constrained Linear Equations



SVD

Matrix Inversion:

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



SVD

Matrix Inversion:

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

Expressing M in terms of its SVD gives:

$$M = UDV^t$$

where:

- U is an $n \times n$ rotation matrix,
- V is an $n \times n$ rotation matrix, and
- D is an $n \times n$ diagonal matrix (i.e off-diagonals are all 0).



SVD

Matrix Inversion:

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

We can express M^{-1} as:

$$M^{-1} = (UDV^t)^{-1}$$



SVD

Matrix Inversion:

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

We can express M^{-1} as:

$$M^{-1} = (UDV^t)^{-1} = (V^{-1})^t D^{-1}U^{-1}$$



SVD

Matrix Inversion:

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

We can express M^{-1} as:

$$\begin{aligned} M^{-1} &= (UDV^t)^{-1} = (V^{-1})^t D^{-1} U^{-1} \\ &= VD^{-1}U^t \end{aligned}$$

Since:

- U is a rotation, $U^{-1}=U^t$.
- V is a rotation, $V^{-1}=V^t$.



SVD

Matrix Inversion:

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

We can express M^{-1} as:

$$M^{-1} = (UDV^t)^{-1} = (V^{-1})^t D^{-1} U^{-1}$$
$$= VD^{-1}U^t$$

Since:

- D is a diagonal matrix:

$$D = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 & 0 \\ 0 & \lambda_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \lambda_{n-1} & 0 \\ 0 & 0 & \cdots & 0 & \lambda_n \end{pmatrix} \longrightarrow D^{-1} = \begin{pmatrix} 1/\lambda_1 & 0 & \cdots & 0 & 0 \\ 0 & 1/\lambda_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1/\lambda_{n-1} & 0 \\ 0 & 0 & \cdots & 0 & 1/\lambda_n \end{pmatrix}$$



SVD

Solving Over-Constrained Linear Equations:

If we have m equations in n unknowns, with $m > n$, the problem is over-constrained and there is no general solution.

$$\begin{pmatrix} a_{11} & \cdots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}$$



SVD

Solving Over-Constrained Linear Equations:

If we have m equations in n unknowns, with $m \geq n$, the problem is over-constrained and there is no general solution.

$$\begin{pmatrix} a_{11} & \cdots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}$$

However, using SVD, we can find the values of $\{x_1, \dots, x_n\}$ that get us as close to $\{y_1, \dots, y_m\}$ as possible.



SVD

Solving Over-Constrained Linear Equations:

If we express the matrix A in terms of its SVD:

$$A = UDV^t$$

then we can set the matrix A^* to be:

$$A^* = VD^*U^t$$

where D^* is the diagonal matrix with:

$$D_{ij}^* = \begin{cases} 1/D_{ij} & \text{if } D_{ij} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

This is called the pseudo-inverse of A .

That is, we invert A as much as possible.



SVD

Solving Over-Constrained Linear Equations:

If we set:

$$(x_1 \cdots x_n)^t = A^* (y_1 \cdots y_m)^t$$

this gives us the values of $\{x_1, \dots, x_n\}$ that most nearly solve the initial equation:

$$A(x_1 \cdots x_n)^t = (y_1 \cdots y_m)^t$$



Overview

- 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 $\{p_0, \dots, p_{n-1}\}$,
define a curve $\Phi(t)$ that approximates/interpolates
the points.

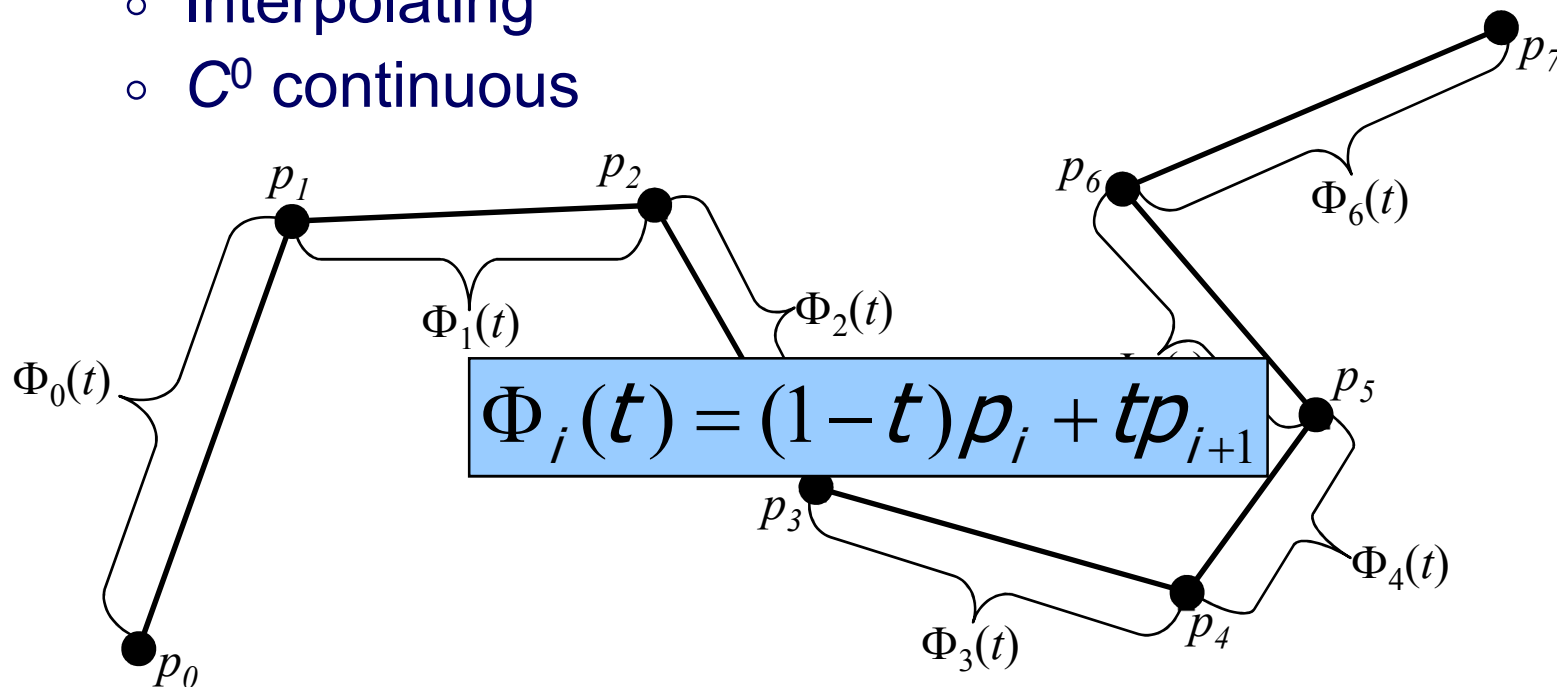


Vectors

Given a collection of n control points $\{p_0, \dots, p_{n-1}\}$, define a curve $\Phi(t)$ that approximates/interpolates the points.

Linear Interpolation:

- Interpolating
- C^0 continuous





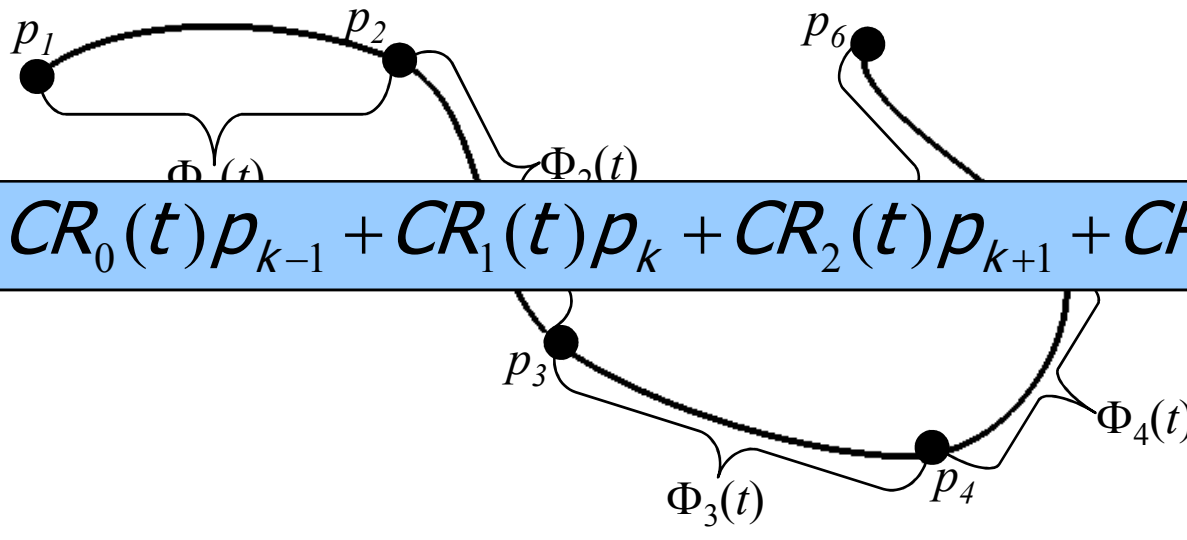
Vectors

Given a collection of n control points $\{p_0, \dots, p_{n-1}\}$, define a curve $\Phi(t)$ that approximates/interpolates the points.

Catmull-Rom Splines (Cardinal Splines with $t=0$):

- Interpolating
- C^1 continuous

● p_7



$$\Phi_i(t) = CR_0(t)p_{k-1} + CR_1(t)p_k + CR_2(t)p_{k+1} + CR_3(t)p_{k+2}$$

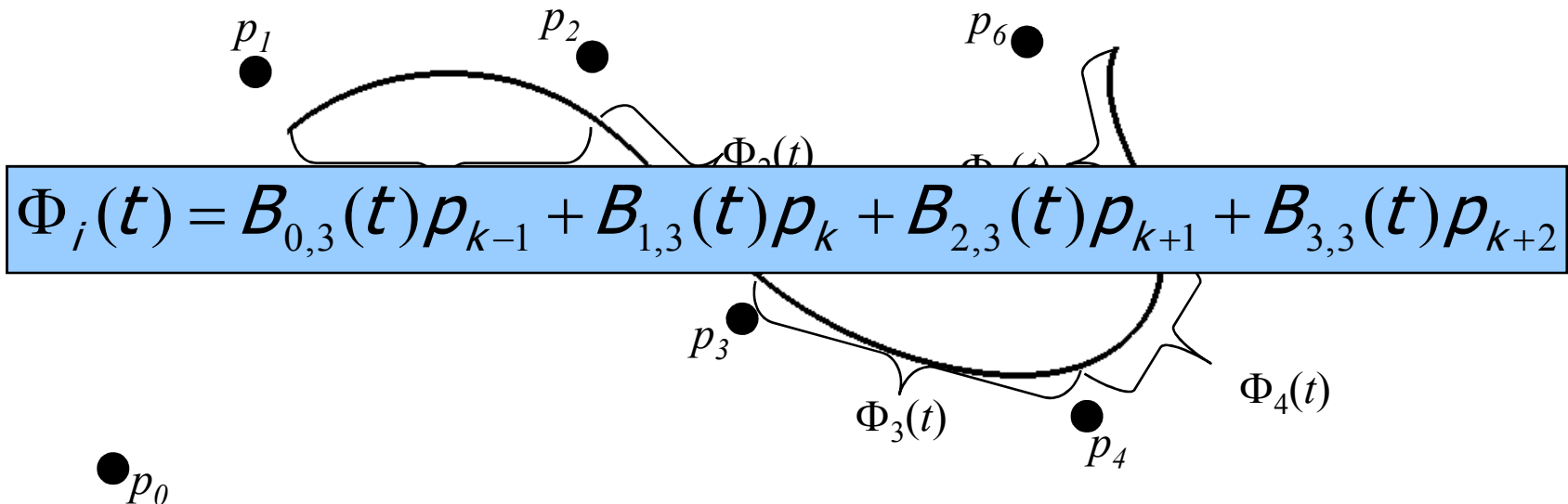


Vectors

Given a collection of n control points $\{p_0, \dots, p_{n-1}\}$,
define a curve $\Phi(t)$ that approximates/interpolates
the points.

Uniform Cubic B-Splines:

- Approximating
- C^2 continuous





Unit-Vectors

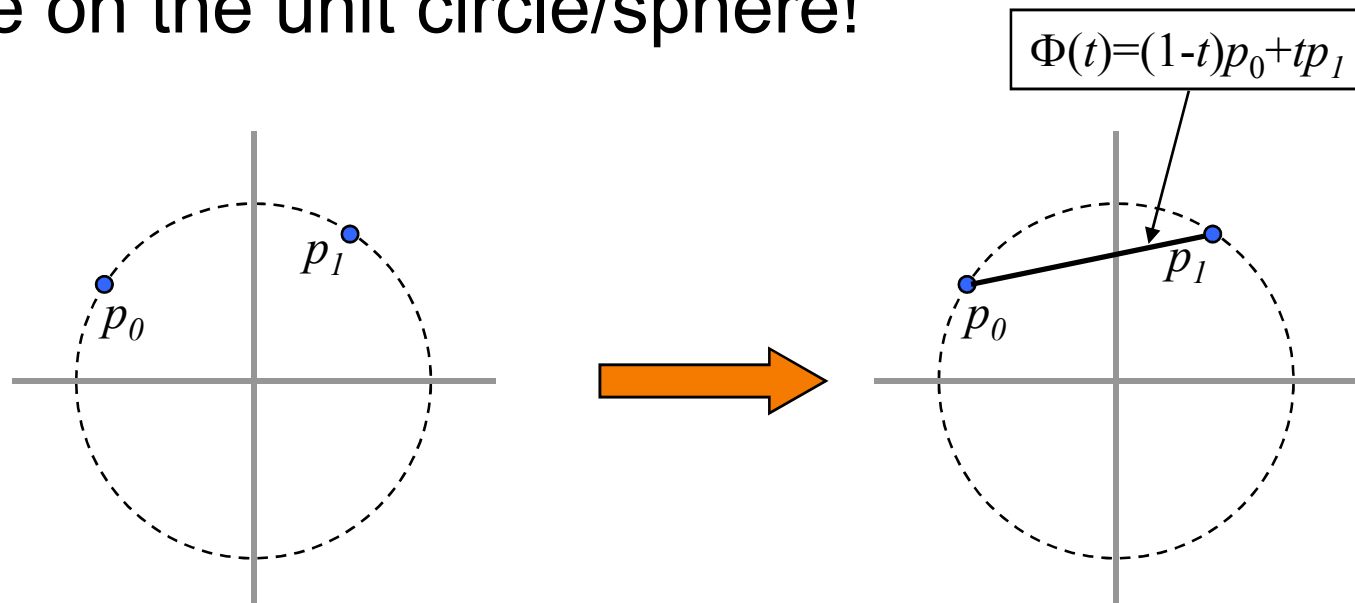
What if we add the additional constraint that the points $\{p_0, \dots, p_{n-1}\}$ and the curve $\Phi(t)$ have to lie on the unit circle/sphere ($\|p_i\|=1$, $\|\Phi(t)\|=1$)?



Unit-Vectors

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

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

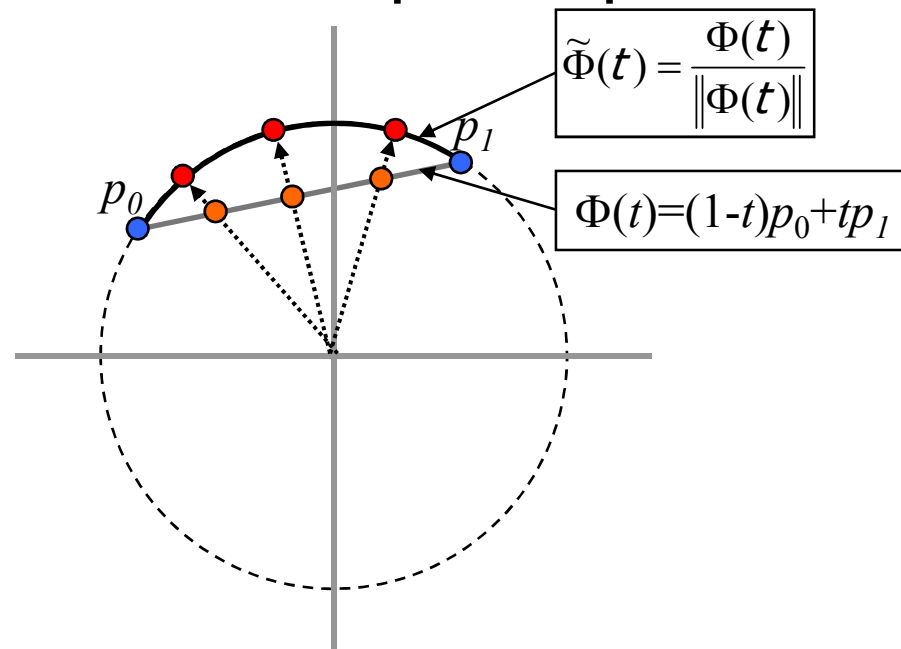




Unit-Vectors

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

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



Curve Normalization



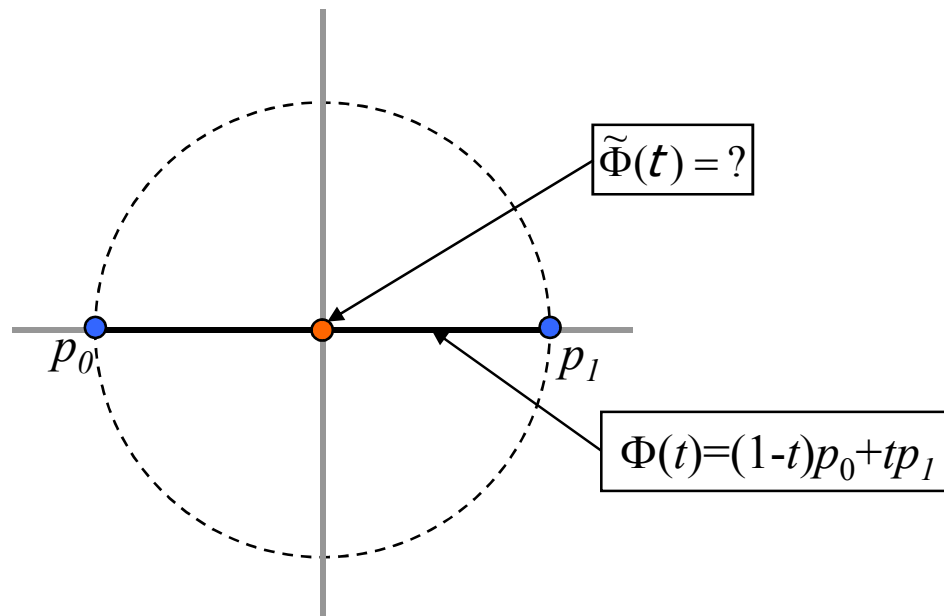
Limitations:



Curve Normalization

Limitations:

- The normalized curve is not always well defined.

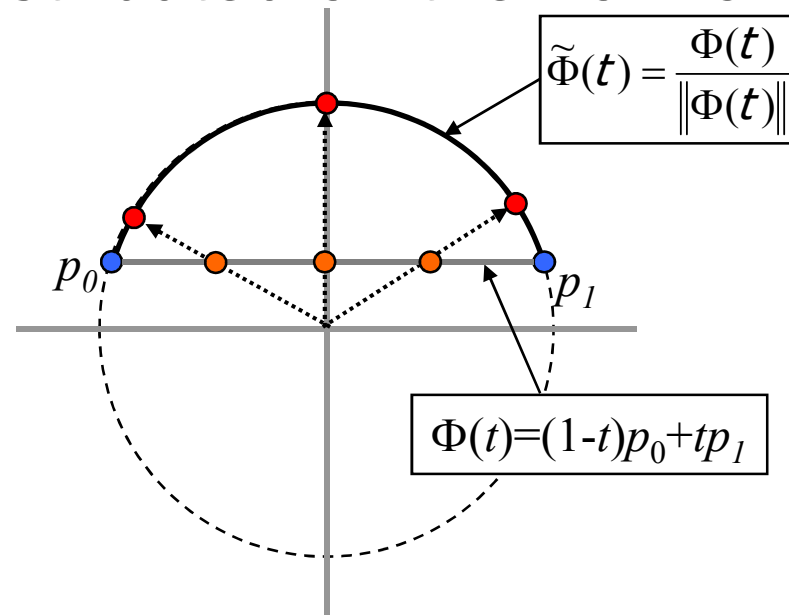




Curve Normalization

Limitations:

- 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 Normalization

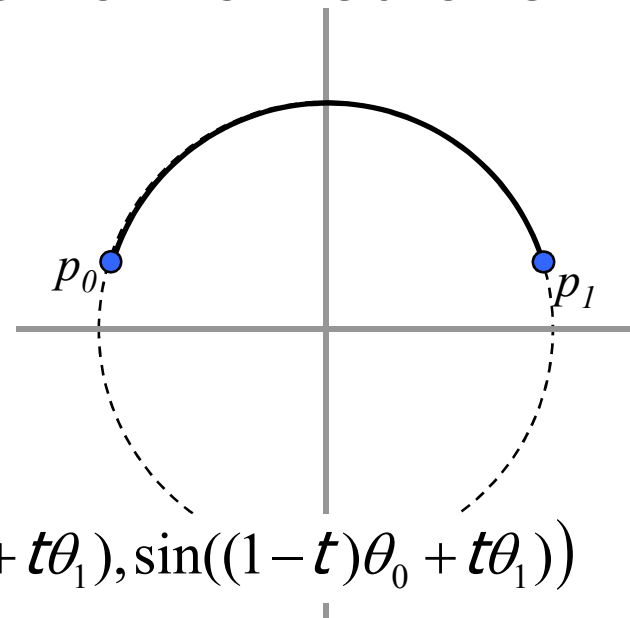
Limitations:

- 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.

SLERP:

If we set:

- $p_0 = (\cos\theta_0, \sin\theta_0)$
- $p_1 = (\cos\theta_1, \sin\theta_1)$



We can set: $\Phi(t) = (\cos((1-t)\theta_0 + t\theta_1), \sin((1-t)\theta_0 + t\theta_1))$



Overview

Interpolating/Approximating

- 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 $\{M_0, \dots, M_{n-1}\}$, define a curve $\Phi(t)$ that approximates/interpolates the matrices.



Matrices

Given a collection of n matrices $\{M_0, \dots, M_{n-1}\}$, define a curve $\Phi(t)$ that approximates/interpolates the matrices.

As with vectors:

- Linear Interpolation:

$$\Phi_i(t) = (1-t)M_i + tM_{i+1}$$

- Catmull-Rom Interpolation:

$$\Phi_i(t) = CR_0(t)M_{k-1} + CR_1(t)M_k + CR_2(t)M_{k+1} + CR_3(t)M_{k+2}$$

- Uniform Cubic B-Spline Approximation:

$$\Phi_i(t) = B_{0,3}(t)M_{k-1} + B_{1,3}(t)M_k + B_{2,3}(t)M_{k+1} + B_{3,3}(t)M_{k+2}$$



Rotations

What if we add the additional constraint that the matrices $\{M_0, \dots, M_{n-1}\}$ and the values of the curve $\Phi(t)$ have to be rotations?



Rotations

What if we add the additional constraint that the matrices $\{M_0, \dots, M_{n-1}\}$ 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!



Rotations

What if we add the additional constraint that the matrices $\{M_0, \dots, M_{n-1}\}$ 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 M , how do you find the rotation matrix R that is closest to M ?



SVD Factorization

Given a matrix M , how do you find the rotation matrix R that is closest to M ?

Singular Value Decomposition (SVD) allows us to express any M as a diagonal matrix, multiplied on the left and right by the rotations R_1 and R_2 :

$$M = R_1 \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} R_2$$



SVD Factorization

Given a matrix M , how do you find the rotation matrix R that is closest to M ?

Singular Value Decomposition (SVD) allows us to express any M as a diagonal matrix, multiplied on the left and right by the rotations R_1 and R_2 :

To be fully correct, you need to ensure that the product of $\text{sgn}(\lambda_i)$ is 1. If not, you need to flip the sign of the $\text{sgn}(\lambda_i)$ where $|\lambda_i|$ is smallest.

The closest rotation R to M is then just the rotation:

$$R = R_1 \begin{pmatrix} \text{sgn}(\lambda_1) & 0 & 0 \\ 0 & \text{sgn}(\lambda_2) & 0 \\ 0 & 0 & \text{sgn}(\lambda_3) \end{pmatrix} R_2$$



Euler Angles

Every rotation matrix R can be expressed as:

- some rotation about the x -axis, multiplied by
- some rotation about the y -axis, multiplied by
- some rotation about the z -axis:

$$R(\theta, \phi, \psi) = R_x(\theta)R_y(\phi)R_z(\psi)$$

The angles (θ, ϕ, ψ) are called the Euler angles.



Euler Angles

Instead of blending matrices and then normalizing using SVD, we can blend the Euler angles:

- For each M_k , compute the Euler angles $(\theta_k, \phi_k, \psi_k)$



Euler Angles

Instead of blending matrices and then normalizing using SVD, we can blend the Euler angles:

- For each M_k , compute the Euler angles $(\theta_k, \phi_k, \psi_k)$
- Interpolate/Approximate the Euler angles:



Euler Angles

Instead of blending matrices and then normalizing using SVD, we can blend the Euler angles:

- For each M_k , compute the Euler angles $(\theta_k, \phi_k, \psi_k)$
- Interpolate/Approximate the Euler angles:

» **Linear Interpolation:**

$$\theta_k(t) = (1-t)\theta_k + t\theta_{k+1}$$

$$\phi_k(t) = (1-t)\phi_k + t\phi_{k+1}$$

$$\psi_k(t) = (1-t)\psi_k + t\psi_{k+1}$$



Euler Angles

Instead of blending matrices and then normalizing using SVD, we can blend the Euler angles:

- For each M_k , compute the Euler angles $(\theta_k, \phi_k, \psi_k)$
- Interpolate/Approximate the Euler angles:
 - » Linear Interpolation
 - » Catmull-Rom Interpolation:

$$\theta_k(t) = CR_0(t)\theta_{k-1} + CR_1(t)\theta_k + CR_2(t)\theta_{k+1} + CR_3(t)\theta_{k+2}$$

$$\phi_k(t) = CR_0(t)\phi_{k-1} + CR_1(t)\phi_k + CR_2(t)\phi_{k+1} + CR_3(t)\phi_{k+2}$$

$$\psi_k(t) = CR_0(t)\psi_{k-1} + CR_1(t)\psi_k + CR_2(t)\psi_{k+1} + CR_3(t)\psi_{k+2}$$



Euler Angles

Instead of blending matrices and then normalizing using SVD, we can blend the Euler angles:

- For each 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:

$$\theta_k(t) = B_{0,3}(t)\theta_{k-1} + B_{1,3}(t)\theta_k + B_{2,3}(t)\theta_{k+1} + B_{3,3}(t)\theta_{k+2}$$

$$\phi_k(t) = B_{0,3}(t)\phi_{k-1} + B_{1,3}(t)\phi_k + B_{2,3}(t)\phi_{k+1} + B_{3,3}(t)\phi_{k+2}$$

$$\psi_k(t) = B_{0,3}(t)\psi_{k-1} + B_{1,3}(t)\psi_k + B_{2,3}(t)\psi_{k+1} + B_{3,3}(t)\psi_{k+2}$$



Euler Angles

Instead of blending matrices and then normalizing using SVD, we can blend the Euler angles:

- For each 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:

$$\Phi_i(t) = R_x(\theta_i(t))R_y(\phi_i(t))R_z(\psi_i(t))$$