Animating Transformations

Michael Kazhdan

(601.457/657)
Recall

Keyframe Animation:

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

- In-betweening (rotation)
  - If you interpolate vertex positions (e.g. instead of angles) the geometry may get distorted.

![Good arm](image1.png) ![Bad arm](image2.png)
Recall

• In-betweening
  ◦ If you interpolate vertex positions (e.g. instead of angles) the geometry may get distorted.
  ◦ 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 translations
    » 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 transformation $O$ is a linear transformation that preserves angles:
  $$\langle v, w \rangle = \langle O(v), O(w) \rangle$$

Recall that the (standard) dot-product between two vectors can be expressed as a matrix multiplication:
  $$\langle v, w \rangle = v^T w$$
Orthogonal Transformations

What are orthogonal transformations?

• An orthogonal transformation $O$ is a linear transformation that preserves angles:
  \[ \langle v, w \rangle = \langle O(v), O(w) \rangle \]

This implies that:

\[
\begin{align*}
    v^T w &= (Ov)^T (Ow) \\
    &= v^T O^T Ow
\end{align*}
\]

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

\[ O^T O = \text{identity} \iff O^T = O^{-1} \]
Orthogonal Transformations

What are orthogonal transformations?

• An orthogonal transformation $O$ is a linear transformation that preserves angles:
  $$\langle v, w \rangle = \langle O(v), O(w) \rangle$$

• An orthogonal matrix $O$ is a matrix whose transpose is its inverse.

• In 3D an orthogonal transformation can be specified by a $3 \times 3$ matrix.
Rotations

What are rotations?

A rotation is an orthogonal transformation that preserves orientation (i.e. has determinant +1).
Rotations

What are rotations?

• A rotation in 3D can also be specified by:
  ◦ its axis of rotation $\mathbf{w}$ ($\|\mathbf{w}\| = 1$) and
  ◦ its angle of rotation $\theta$
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 $\theta$

Properties:

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

Properties:

◦ The rotation corresponding to \((\theta, w)\) is the same as the rotation corresponding to \((-\theta, -w)\).
◦ How do we define the product of rotations corresponding to \((\theta_1, w_1)\) and \((\theta_2, w_2)\)?
◦ Given a rotation corresponding to \((\theta, w)\), the rotation raised to the power \(\alpha\) corresponds to \((\alpha \theta, w)\).
SVD

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

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

where:

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

Applications:
- Aligning point-sets
- 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
Finding the Inverse of a Matrix:

\[ M = U D V^T \]

We can express \( M^{-1} \) as:

\[ M^{-1} = (UDV^T)^{-1} = (V^T)^{-1}D^{-1}U^{-1} \]

\[ = VD^{-1}U^T \]

Since:

- \( U \) is an orthogonal transformation, \( U^{-1} = U^T \).
- \( V \) is an orthogonal transformation, \( V^{-1} = V^T \).
SVD

Solving Linear Systems:

\[ M^{-1} = VD^{-1}U^\top \]

Since \( D \) is a diagonal matrix:

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

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 $\{p_0, \ldots, p_{n-1}\}$, 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.

**Linear Interpolation:**

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

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

- Interpolating
- $C^1$ continuous

\[
\Phi_k(t) = CR_0(t) \cdot p_{k-1} + CR_1(t) \cdot p_k + CR_2(t) \cdot p_{k+1} + CR_3(t) \cdot p_{k+2}
\]
Vectors

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

**Uniform Cubic B-Splines:**
- Approximating
- $C^2$ continuous

\[
\Phi_k(t) = B_{0,3}(t) \cdot p_{k-1} + B_{1,3}(t) \cdot p_k + B_{2,3}(t) \cdot p_{k+1} + B_{3,3}(t) \cdot p_{k+2}
\]
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!

\[
\Phi(t) = (1 - t)\mathbf{p}_0 + t\mathbf{p}_1
\]
Unit-Vectors

What if we add the constraint that the points \( \{ p_0, \ldots, p_{n-1} \} \) 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:

\[
\Phi(t) = \frac{\Phi(t)}{\| \Phi(t) \|}
\]

\[
\tilde{\Phi}(t) = (1 - t)p_0 + tp_1
\]
Curve Normalization

Limitations:
Curve Normalization

Limitations:

• The normalized curve is not always well defined.

\[ \Phi(t) = (1 - t)p_0 + tp_1 \]
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.

\[
\phi(t) = (1 - t)p_0 + tp_1
\]

\[
\tilde{\phi}(t) = \frac{\phi(t)}{\|\phi(t)\|}
\]
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:
  \[ p_0 = (\cos \theta_0, \sin \theta_0) \]
  \[ p_1 = (\cos \theta_1, \sin \theta_1) \]
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:**
  \[
  \Phi(t) = (\cos((1 - t)\theta_0 + t\theta_1), \sin((1 - t)\theta_0 + t\theta_1))
  \]
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:
  \[ p_0 = (\cos \theta_0, \sin \theta_0) \]
  \[ p_1 = (\cos \theta_1, \sin \theta_1) \]
- Set:
  \[ \Phi(t) = (\cos((1-t)\theta_0 + t\theta_1), \sin((1-t)\theta_0 + t\theta_1)) \]

Note:
- Parameter may not be unique.
- There may not be a good parameterization.
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.

As with vectors:

- **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 \( \{M_0, \ldots, M_{n-1}\} \) 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 $\mathbf{R}$?
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 $\lambda_i$ are positive, the closest orthogonal transform $\mathbf{O}$ to $\mathbf{M}$ is:

$$
\mathbf{O} = \mathbf{O}_1 \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix} \mathbf{O}_2
$$
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 $\lambda_i$ are positive, the closest orthogonal transform $\mathbf{O}$ to $\mathbf{M}$ is:

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

In standard SVD factorization, the diagonal values are positive, 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 need to make the product have determinant $1$. 
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 $\mathbf{O}_1$ and $\mathbf{O}_2$:

$$ \mathbf{M} = \mathbf{O}_1 \boldsymbol{\lambda} \mathbf{O}_2 $$

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

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

In standard SVD factorization, the diagonal values are positive, 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 need to make the product have determinant 1.
Euler Angles

Every rotation matrix $\mathbf{R}$ can be expressed as:

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

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

The angles $(\theta, \phi, \psi)$ are called the Euler angles.
Euler Angles

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

- For each $\mathbf{M}_k$, 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:
    - $\theta_k(t) = (1 - t)\theta_k + t \cdot \theta_{k+1}$
    - $\phi_k(t) = (1 - t)\phi_k + t \cdot \phi_{k+1}$
    - $\psi_k(t) = (1 - t)\psi_k + t \cdot \psi_{k+1}$
Euler Angles

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

- 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:
    - $\theta_k(t) = CR_0(t) \cdot \theta_{k-1} + CR_1(t) \cdot \theta_k + CR_2(t) \cdot \theta_{k+1} + CR_3(t) \cdot \theta_{k+2}$
    - $\phi_k(t) = CR_0(t) \cdot \phi_{k-1} + CR_1(t) \cdot \phi_k + CR_2(t) \cdot \phi_{k+1} + CR_3(t) \cdot \phi_{k+2}$
    - $\psi_k(t) = CR_0(t) \cdot \psi_{k-1} + CR_1(t) \cdot \psi_k + CR_2(t) \cdot \psi_{k+1} + CR_3(t) \cdot \psi_{k+2}$
Euler Angles

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

- 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:

$$\begin{align*}
\theta_k(t) &= B_{0,3}(t) \cdot \theta_{k-1} + B_{1,3}(t) \cdot \theta_k + B_{2,3}(t) \cdot \theta_{k+1} + B_{3,3}(t) \cdot \theta_{k+2} \\
\phi_k(t) &= B_{0,3}(t) \cdot \phi_{k-1} + B_{1,3}(t) \cdot \phi_k + B_{2,3}(t) \cdot \phi_{k+1} + B_{3,3}(t) \cdot \phi_{k+2} \\
\psi_k(t) &= B_{0,3}(t) \cdot \psi_{k-1} + B_{1,3}(t) \cdot \psi_k + B_{2,3}(t) \cdot \psi_{k+1} + B_{3,3}(t) \cdot \psi_{k+2}
\end{align*}$$
Euler Angles

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

- 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:

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

Note that to blend rigid transformations, we want to do the standard blend of the translation component and the constrained blend of the rotation.