Animating Transformations

Michael Kazhdan

(601.457/657)
Recall

Keyframe Animation:

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

• In-betweening
  ◦ Interpolate/approximate transformations in the scene graph, **not the positions**

Watt & Watt
Recall

• In-betweening
  ◦ Interpolate/approximate transformations in the scene graph, **not the positions**
  ◦ 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 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:

\[ v^t w = (Ov)^t (Ow) \]
\[ = v^t O^t Ow \]

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 transformation $O$ is a linear transformation whose transpose is its inverse.

• A 3D 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 $w$ ($\|w\| = 1$) 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 $m \times n$ matrix $M$ can be expressed in terms of its Singular Value Decomposition as:

$$M = UDV^t$$

where:

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

» Typically the diagonal entries are:
  - Non-negative
  - Decreasing
SVD

Applications:

- Aligning point-sets
- Solving linear systems
- Solving over-constrained linear systems
- Compression
SVD

Solving Linear Systems:

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

Expressing \( M \) in terms of its SVD gives:

\[ M = UDV^t \]

where:

- \( U \) is an \( n \times n \) orthogonal matrix,
- \( V \) is an \( n \times n \) orthogonal matrix,
- \( D \) is an \( n \times n \) diagonal matrix
SVD

Solving Linear Systems:

\[ M = UDV^t \]

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:

\[ M^{-1} = V D^{-1} U^t \]

Since \( D \) is a diagonal matrix:

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

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

**Linear Interpolation:**
- Interpolating
- $C^0$ continuous

\[
\Phi_k(t) = (1 - t)p_k + t \cdot p_{k+1}
\]
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 \( \{p_0, \ldots, p_{n-1}\} \) and the curve \( \Phi(t) \) have to lie on the unit circle/sphere (\( \|p_i\| = 1, \|\Phi(t)\| = 1 \))?

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

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

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

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

Limitations:
Curve Normalization

Limitations:

• The normalized curve is not always well defined.

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

\[ \Phi(t) = \text{?} \]
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:
  \[ p_0 = (\cos \theta_0, \sin \theta_0) \]
  \[ p_1 = (\cos \theta_1, \sin \theta_1) \]
- Set:
  \[ \Phi(t) = (\cos((1-t)\theta_0 + t\theta_1), \sin((1-t)\theta_0 + t\theta_1)) \]
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 $\{M_0, \ldots, M_{n-1}\}$, define a curve $\Phi(t)$ that approximates/interpolates the matrices.
Matrices

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

As with vectors:

- **Linear Interpolation:**
  \[ \Phi_k(t) = (1 - t)M_k + t \cdot M_{k+1} \]

- **Catmull-Rom Interpolation:**
  \[ \Phi_k(t) = CR_0(t) \cdot M_{k-1} + CR_1(t) \cdot M_k + CR_2(t) \cdot M_{k+1} + CR_3(t) \cdot M_{k+2} \]

- **Uniform Cubic B-Spline Approximation:**
  \[ \Phi_k(t) = B_{0,3}(t) \cdot M_{k-1} + B_{1,3}(t) \cdot M_k + B_{2,3}(t) \cdot M_{k+1} + B_{3,3}(t) \cdot M_{k+2} \]
Rotations

What if we add the constraint that the matrices \( \{M_0, ..., M_{n-1}\} \) 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 $M$, what is the closest rotation $R$?
SVD Factorization

Given a matrix $M$, what is the closest rotation $R$?

Singular Value Decomposition (SVD) allows us to express $M$ as a diagonal matrix, multiplied on the left/right by orthogonal transformations $O_1/O_2$:

$$M = O_1 \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} O_2$$

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

$$O = O_1 \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} O_2$$
SVD Factorization

Given a matrix $M$, what is the closest rotation $R$?

Singular Value Decomposition (SVD) allows us to express $M$ as a diagonal matrix, multiplied on the left/right by orthogonal transformations $O_1/O_2$:

$$M = O_1 \lambda O_2$$

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

$$O = O_1 \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} O_2$$

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

The orthogonal transformations $O_1$ and $O_2$ are not necessarily rotations.

To get a rotation, we need to make the product have determinant is 1.
SVD Factorization

Given a matrix $M$, what is the closest rotation $R$?

Singular Value Decomposition (SVD) allows us to express $M$ as a diagonal matrix, multiplied on the left/right by orthogonal transformations $O_1/O_2$:

$$M = O_1 \lambda_1 0 0 0 \lambda_2 0 0 0 \lambda_3 O_2$$

Because the $\lambda_i$ are positive and decreasing, the closest rotation $R$ to $M$ is the rotation:

$$R = O_1 \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \text{det}(O_1 \cdot O_2) \end{pmatrix} O_2$$

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

The orthogonal transformations $O_1$ and $O_2$ are not necessarily rotations.

To get a rotation, we need to make the product have determinant $1$. 
Euler Angles

Every rotation matrix $R$ 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:

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

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 $M_k$, 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 $M_k$, 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 $M_k$, 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:
    - $\theta_k(t) = B_{0,3}(t) \cdot \theta_{k-1} + B_{1,3}(t) \cdot \theta_k + B_{2,3}(t) \cdot \theta_{k+1} + B_{3,3}(t) \cdot \theta_{k+2}$
    - $\phi_k(t) = B_{0,3}(t) \cdot \phi_{k-1} + B_{1,3}(t) \cdot \phi_k + B_{2,3}(t) \cdot \phi_{k+1} + B_{3,3}(t) \cdot \phi_{k+2}$
    - $\psi_k(t) = B_{0,3}(t) \cdot \psi_{k-1} + B_{1,3}(t) \cdot \psi_k + B_{2,3}(t) \cdot \psi_{k+1} + B_{3,3}(t) \cdot \psi_{k+2}$
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:
  \[
  \Phi_k(t) = R_z(\theta_k(t))R_y(\phi_k(t))R_x(\psi_k(t))
  \]

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