Animation and Rotations

Michael Kazhdan

(600.457)
Outline

- Keyframe animation
- Articulated figures
- Interpolating rotations

Luxo Junior
Keyframe Animation

- Define character poses at specific time steps called “keyframes”
Keyframe Animation

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

- In-betweening (translation):
  - Cubic spline interpolation - maybe good enough
    » May not follow physical laws

Recall: Convex hull containment
Keyframe Animation

- In-betweening (translation):
  - Cubic spline interpolation - maybe good enough
    » May not follow physical laws
Outline

• Keyframe animation
• Articulated figures
• Interpolating rotations
Articulated Figures

- Character poses described by set of rigid bodies connected by “joints”

Scene Graph

Angel Figures 8.8 & 8.9
Articulated Figures

- Well-suited for humanoid characters

Rose et al. `96
Example: Walk Cycle

- Articulated figure:
Example: Walk Cycle

• Hip joint orientation:
Example: Walk Cycle

• Knee joint orientation:
Example: Walk Cycle

- Ankle joint orientation:
Example: Walk Cycle

http://www.ischool.utexas.edu/~luna73/architecture/
Outline

• Keyframe animation
• Articulated figures
• Interpolating rotations
Interpolating Rotations

- In-betweening (rotation)
  - Interpolate angles, not positions, between keyframes

Watt & Watt

Good arm

Bad arm
Rotations

What are rotations?
Rotations

What are rotations?

- A rotation $R$ is a linear transformation that has determinant one and preserves angles:
  $$\langle v, w \rangle = \langle R(v), R(w) \rangle$$

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 one and preserves angles:
  $$\langle v, w \rangle = \langle R(v), R(w) \rangle$$

This implies that:

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

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

$$R^t R = \text{identity} \quad \iff \quad R^t = R^{-1}$$
Rotations

What are rotations?

• A rotation $R$ is a linear transformation that has determinant one and preserves angles:
  $$\langle v, w \rangle = \langle R(v), R(w) \rangle$$

• A rotation $R$ is a linear transformation that has determinant equal to one and whose transpose is its inverse.

• A 3D rotation can be specified by a $3 \times 3$ 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 $\theta$
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 \( (-\theta, -w) \).
◦ 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 \( (\theta, w) \), 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 \( (\theta, w) \), the rotation raised to the power \( \alpha \) corresponds to \( (\alpha \theta, w) \).
Outline

• Keyframe animation
• Articulated figures
• Interpolating rotations
  ◦ SVD
  ◦ Interpolating/Approximating Points
  ◦ Interpolating/Approximating Transformations
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
- $D$ is an $m \times n$ diagonal matrix (i.e. off-diagonals are 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 $M$.

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,
- $D$ is an $n \times n$ diagonal matrix
**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:

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

$$= VD^{-1}U^t$$

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:

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

Since $D$ is a diagonal matrix:

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

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

Since $D$ is a diagonal matrix:

$$D = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix}$$

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

- Keyframe animation
- Articulated figures
- Interpolating rotations
  - Rotations and SVD
  - Interpolating/Approximating Points
  - Interpolating/Approximating Transformations
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 additional constraint that the points \( \{ p_0, \ldots, p_{n-1} \} \) and the curve \( \Phi(t) \) have to lie on the unit circle/sphere \( (\| p_i \| = 1, \| \Phi(t) \| = 1) \)?
Unit-Vectors

What if we add the additional 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 additional 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.
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: \((\cos \theta, \sin \theta)\)
- 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:
  \[
  p_0 = (\cos \theta_0, \sin \theta_0), \quad p_1 = (\cos \theta_1, \sin \theta_1)
  \]

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

• Keyframe animation
• Articulated figures
• Interpolating rotations
  ○ Rotations and SVD
  ○ Interpolating/Approximating Points
  ○ Interpolating/Approximating Transformations
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:**
  \[
  \Phi_k(t) = CR_0(t) \cdot p_{k-1} + CR_1(t) \cdot p_k + CR_2(t) \cdot p_{k+1} + CR_3(t) \cdot p_{k+2}
  \]

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

What if we add the additional 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 $M$, how do you find the rotation matrix $R$ that is closest to $M$?
Normalization: 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 $R_2$:

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

The closest rotation $R$ to $M$ is the rotation:

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

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

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 the rotation:
Parameterization: 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 $(\theta, \phi, \psi)$ are called the **Euler angles**.
Parameterization: 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 $(\theta_k, \phi_k, \psi_k)$
Parameterization: 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:
Parameterization: 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:
  
  - $\theta_k(t) = (1 - t)\theta_k + t \cdot \theta_{k+1}$
  - $\phi_k(t) = (1 - t)\phi_k + t \cdot \phi_{k+1}$
  - $\psi_k(t) = (1 - t)\psi_k + t \cdot \psi_{k+1}$
Parameterization: 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:
    - $\theta_k(t) = CR_0(t) \cdot \theta_{k-1} + CR_1(t) \cdot \theta_k + CR_2(t) \cdot \theta_{k+1} + CR_3(t) \cdot \theta_{k+2}$
    - $\phi_k(t) = CR_0(t) \cdot \phi_{k-1} + CR_1(t) \cdot \phi_k + CR_2(t) \cdot \phi_{k+1} + CR_3(t) \cdot \phi_{k+2}$
    - $\psi_k(t) = CR_0(t) \cdot \psi_{k-1} + CR_1(t) \cdot \psi_k + CR_2(t) \cdot \psi_{k+1} + CR_3(t) \cdot \psi_{k+2}$
Parameterization: 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:
    - $\theta_k(t) = B_{0,3}(t) \cdot \theta_{k-1} + B_{1,3}(t) \cdot \theta_k + B_{2,3}(t) \cdot \theta_{k+1} + B_{3,3}(t) \cdot \theta_{k+2}$
    - $\phi_k(t) = B_{0,3}(t) \cdot \phi_{k-1} + B_{1,3}(t) \cdot \phi_k + B_{2,3}(t) \cdot \phi_{k+1} + B_{3,3}(t) \cdot \phi_{k+2}$
    - $\psi_k(t) = B_{0,3}(t) \cdot \psi_{k-1} + B_{1,3}(t) \cdot \psi_k + B_{2,3}(t) \cdot \psi_{k+1} + B_{3,3}(t) \cdot \psi_{k+2}$
Parameterization: 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 rotation to the rotation at the in-between parameters:
  \[
  \Phi_k(t) = R_x(\theta_k(t))R_y(\phi_k(t))R_z(\psi_k(t))
  \]

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