Topics in Articulated Animation

Animation
Articulated models:
- rigid parts
- connected by joints

They can be animated by specifying the joint angles (or other display parameters) as functions of time.

Character Representation
Character Models are rich, complex
- hair, clothes (particle systems)
- muscles, skin (FFD’s etc.)

Focus is rigid-body Degrees of Freedom (DOFs)
- joint angles

Reading
Shoemake, “Quaternions Tutorial”
Simple Rigid Body → Skeleton

Kinematics and dynamics

Kinematics: how the positions of the parts vary as a function of the joint angles.
Dynamics: how the positions of the parts vary as a function of applied forces.

Key-frame animation

- Each joint specified at various key frames (not necessarily the same as other joints)
- System does interpolation or in-betweening

Doing this well requires:
- A way of smoothly interpolating key frames: splines
- A good interactive system
- A lot of skill on the part of the animator

Representing a Skeleton

Model & connect each bone
- corresponds to most mocap data
- difficult to formulate joint limits
- not very efficient either
  - explicit constraints for joints
  - many wasted DOFs
**Efficient Skeleton: Hierarchy**

Implicitize joint constraints
- each bone relative to parent
- easy to limit joint angles
- very efficient
  - # angles = # DOFs
  - no constraints to enforce
  - leverages graphics libraries and hardware

---

**Operations on Hierarchies**

Specify poses

Draw the character in a given pose

Compute positions and orientations on body

*for example...*

---

**Computing a Sensor Position**

Forward kinematics
- uses vector-matrix multiplication
- transformation matrix is composition of all joint transforms between sensor/effector and root

\[ v_w = T(x_h, y_h, z_h) R(\theta, \phi, \sigma) \quad TR(\theta, \phi, \sigma) \quad TR(\theta) \quad TR(\theta) \quad v_s \]

---

**Joints = Rotations**

To specify a pose, we specify the joint-angle rotations

Each joint can have up to three rotational DOFs
**Euler angles**

An Euler angle is a rotation about a single Cartesian axis. Create multi-DOF rotations by concatenating Eulers.

Can get three DOF by concatenating:

- Euler-X
- Euler-Y
- Euler-Z

**Singularities**

What is a singularity?

- continuous subspace of parameter space all of whose elements map to same rotation

Why is this bad?

- induces gimbal lock - two or more axes align, results in loss of rotational DOFs (i.e. derivatives)

**Singularities in Action**

An object whose orientation is controlled by Euler rotation XYZ(\(\theta, \phi, \sigma\)).

- (0,0,0) : Okay
- (0, \(\pm 90^\circ\), 0) : X and Z axes align

**Eliminates a DOF**

In this configuration, changing \(\theta\) (X Euler angle) and \(\sigma\) (Z Euler angle) produce the same result.

No way to rotate around world X axis!
### Resulting Behavior

No applied force or other stimuli can induce rotation about world X-axis

The object locks up!!

### Singularities in Euler Angles

Cannot be avoided (occur at 0° or 90°)

Difficult to work around

But, only affects three DOF rotations

### Other Properties of Euler Angles

Several important tasks are easy:
- interactive specification (slides, etc.)
- joint limits
- Euclidean interpolation (Hermites, Beziers, etc.)
  - May be funky for tumbling bodies
  - fine for most joints

### Quaternions

But… singularities are unacceptable for IK, optimization

Traditional solution: Use unit quaternions to represent rotations
- $S^1$ has same topology as rotation space (a sphere), so no singularities
History of Quaternions

Invented by Sir William Rowan Hamilton in 1843

\[ H = w + ix + jy + kz \]
where \( i^2 = j^2 = k^2 = ijk = -1 \)

I still must assert that this discovery appears to me to be as important for the middle of the nineteenth century as the discovery of fluxions [the calculus] was for the close of the seventeenth.

Hamilton

[quaternions] … although beautifully ingenious, have been an unmixed evil to those who have touched them in any way.

Thompson

Quaternion as a 4 vector

\[ q = \begin{pmatrix} w \\ x \\ y \\ z \end{pmatrix} \]

Axis-angle rotation as a quaternion

\[ q = \begin{pmatrix} \cos(\theta/2) \\ \sin(\theta/2)x \\ \sin(\theta/2)y \\ \sin(\theta/2)z \end{pmatrix} \]

Unit Quaternions

\[ q = \begin{pmatrix} w \\ x \\ y \\ z \end{pmatrix} \]

\[ w = \sqrt{1 - (x^2 + y^2 + z^2)} \]

\[ |q| = 1 \]

\[ x^2 + y^2 + z^2 + w^2 = 1 \]
Quaternion Product

\[
\begin{pmatrix} w_1 \\ v_1 \end{pmatrix} \begin{pmatrix} w_2 \\ v_2 \end{pmatrix} = \begin{pmatrix} w_1 w_2 - v_1 \cdot v_2 \\ w_1 v_2 + w_2 v_1 + v_1 \times v_2 \end{pmatrix}
\]

\[
\begin{pmatrix} w_1 \\ v_1 \end{pmatrix} \begin{pmatrix} w_2 \\ v_2 \end{pmatrix} \neq \begin{pmatrix} w_1 \\ v_1 \end{pmatrix} \begin{pmatrix} w_2 \\ v_2 \end{pmatrix}
\]

Quaternion Conjugate

\[
q^* = \begin{pmatrix} w_1 \\ v_1 \end{pmatrix} = \begin{pmatrix} w_1 \\ -v_1 \end{pmatrix}
\]

\[
(p^*)^* = p
\]

\[
(pq)^* = q^* p^*
\]

\[
(p + q)^* = p^* + q^*
\]

Quaternion Inverse

\[
q^{-1} q = 1
\]

\[
q^{-1} = q^*/|q| = \begin{pmatrix} w \\ -v \end{pmatrix}/|q| = \begin{pmatrix} w \\ -v \end{pmatrix}/(w^2 + v \cdot v)
\]

Quaternion Rotation

\[
q p q^{-1} = \begin{pmatrix} w^2 - p \cdot v \cdot v + (p \cdot v) + v(p \cdot v) \\
wp - p \times v 
\end{pmatrix}
\]

What about a quaternion product \( q q ? \)
Matrix Form

\[ q = \begin{pmatrix} w \\ x \\ y \\ z \end{pmatrix} \]

\[ M = \begin{pmatrix}
1 - 2y^2 - 2z^2 & 2xy + 2wz & 2xz - 2wy \\
2xy - 2wz & 1 - 2x^2 - 2z^2 & 2yz + 2wx \\
2xz + 2wy & 2yz - 2wx & 1 - 2x^2 - 2y^2
\end{pmatrix} \]

Quatranions: What Works

Simple formulae for converting to rotation matrix

Continuous derivatives - no singularities

“Optimal” interpolation - geodesics map to shortest paths in rotation space

Nice calculus (corresponds to rotations)

Quatranion constraints

Restricting the rotation cone

\[ \frac{1 - \cos(\theta)}{2} = q_z^2 + q_x^2 \]

Restricting the rotation twist around an axis

\[ \tan(\theta/2) = \frac{q_{wz}}{q_x} \]

What Hierarchies Can and Can’t Do

Advantages:

- Reasonable control knobs
- Maintains structural constraints

Disadvantages:

- Doesn’t always give the “right” control knobs
  - e.g. hand or foot position - re-rooting may help
- Can’t do closed kinematic chains (keep hand on hip)
- Other constraints: do not walk through walls
Procedural Animation

Transformation parameters as functions of other variables

Simple example:
- a clock with second, minute and hour hands
- hands should rotate together
- express all the motions in terms of a “seconds” variable
- whole clock is animated by varying the seconds parameter

\[
m = \frac{s}{60} \\
h = \frac{m}{60} \\
\theta_s = \frac{2\pi s}{60} \\
\theta_m = \frac{2\pi m}{60} \\
\theta_h = \frac{2\pi h}{12}
\]

Models as Code: draw-a-bug

```c
void draw_bug(float walk_phase_angle, float xpos, float ypos, float zpos){
    pushmatrix;
    translate(xpos, ypos, zpos);
    for each leg:
        pushmatrix;
        translate(leg pos relative to body);
        draw_bug_leg(theta1 & theta2 for that leg);
        popmatrix;
    popmatrix;
}

void draw_bug_leg(float theta1, float theta2){
    glPushMatrix();
    glRotatef(theta1,0,0,1);
    draw_leg_segment(SEGMENT1_LENGTH);
    glTranslatef(SEGMENT1_LENGTH,0,0);
    glRotatef(theta2,0,0,1);
    draw_leg_segment(SEGMENT2_LENGTH);
    glPopMatrix();
}
```

Hard Example

In the figure below, what expression would you use to calculate the arm’s rotation angle to keep the tip on the star-shaped wheel as the wheel rotates???