Topics in Articulated Animation

Reading

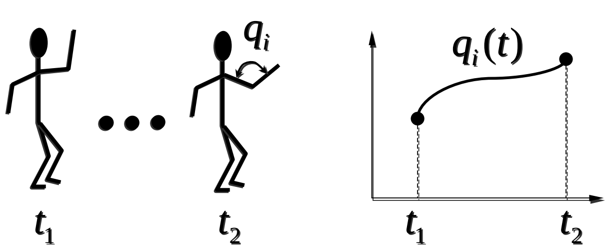
Shoemake, “Quaternions Tutorial”

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.*)

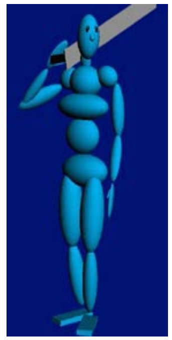
Focus is rigid-body Degrees of Freedom (DOFs)

- joint angles

Simple Rigid Body → Skeleton

Copyright © Squaresoft 1999

vs.



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.

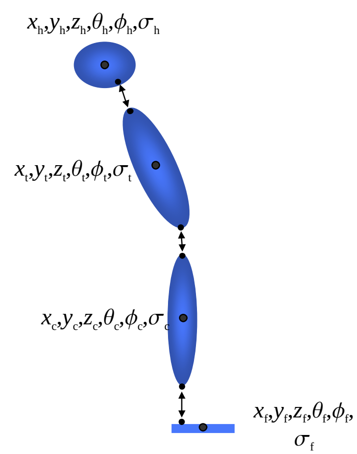
Key-frame animation

- 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

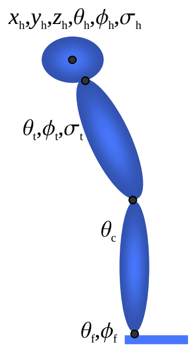
Representing a Skeleton

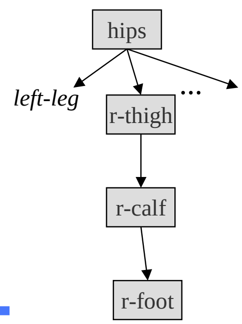


Model & connect each bone

- 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

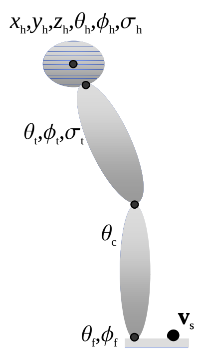
Specify poses

Draw the character in a given pose

Compute positions and orientations on body

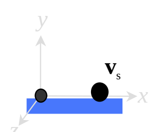
for example...

Computing a Sensor Position



Forward kinematics

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



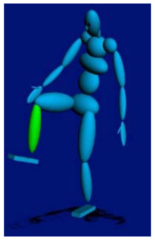
$$\mathbf{v}_w = \mathbf{T}(x_h, y_h, z_h)\mathbf{R}(\theta_h, \phi_h, \sigma_h)\ \mathbf{TR}(\theta_t, \phi_t, \sigma_t)\ \mathbf{TR}(\theta_c)\ \mathbf{TR}(\theta_f, \phi_f)\ \mathbf{v}_s$$

Joints = Rotations

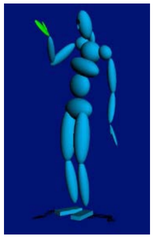
To specify a pose, we specify the joint-angle rotations

Each joint can have up to three rotational DOFs

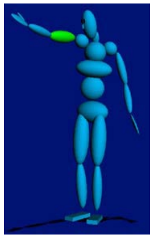
1 DOF: knee



2 DOF: wrist



3 DOF: arm



Euler angles

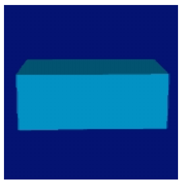
An Euler angle is a rotation about a single Cartesian axis

Create multi-DOF rotations by concatenating Eulers

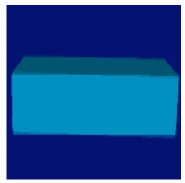
Can get three DOF by concatenating:

Euler-X

Euler-Y



Euler-Z



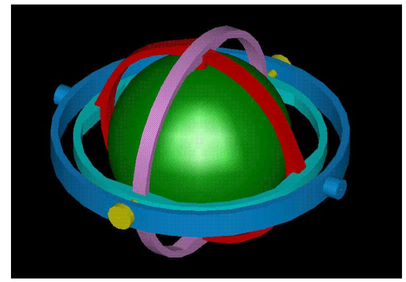
Singularities

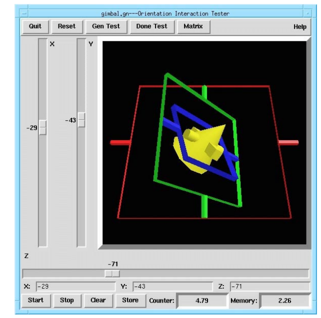
What *is* a singularity?

- 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)



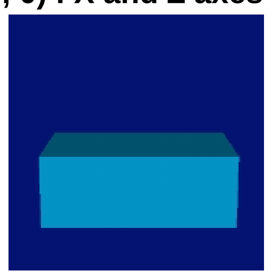


Singularities in Action

An object whose orientation is controlled by Euler rotation XYZ(θ,ϕ,σ)

(0,0,0) : Okay

(0, ± 90^o, 0) : X and Z axes align

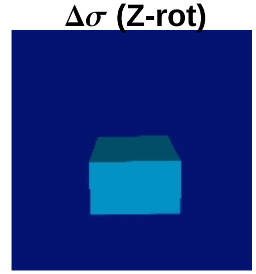


Eliminates a DOF

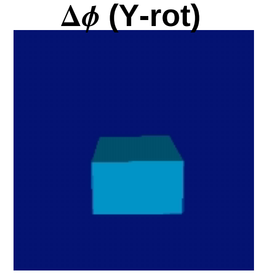
In this configuration, changing θ (X Euler angle) and σ (Z Euler angle) produce the same result.

No way to rotate around world X axis!

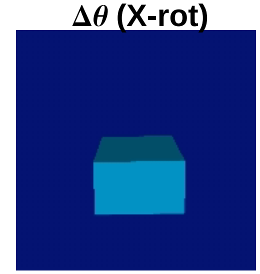
$\Delta\sigma$ (Z-rot)



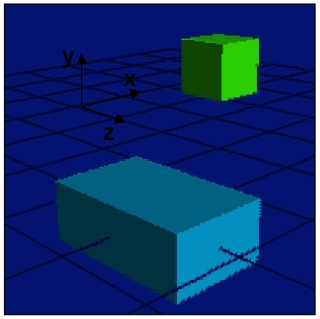
$\Delta\phi$ (Y-rot)



$\Delta\theta$ (X-rot)



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 (sliders, *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^3 has same topology as rotation space (a sphere), so no singularities

History of Quaternions

Invented by Sir William Rowan Hamilton in 1843

$$H = w + \mathbf{i}x + \mathbf{j}y + \mathbf{k}z$$
$$\text{where } \mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$$

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.

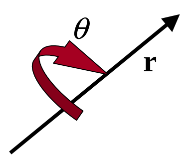
Thompson

Quaternion as a 4 vector

$$\mathbf{q} = \begin{pmatrix} w \\ x \\ y \\ z \end{pmatrix} = \begin{pmatrix} w \\ \mathbf{v} \end{pmatrix}$$

Axis-angle rotation as a quaternion

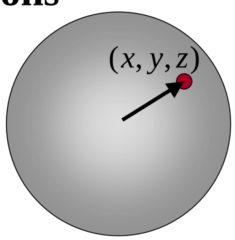
$$\mathbf{q} = \begin{pmatrix} w \\ x \\ y \\ z \end{pmatrix} = \begin{pmatrix} w \\ \mathbf{v} \end{pmatrix}$$



$$\mathbf{q} = \begin{pmatrix} \cos(\theta/2) \\ \sin(\theta/2)\mathbf{r} \end{pmatrix}$$

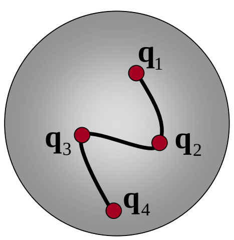
Unit Quaternions

$$\mathbf{q} = \begin{pmatrix} w \\ x \\ y \\ z \end{pmatrix}$$



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

$$|\mathbf{q}| = 1$$
$$x^2 + y^2 + z^2 + w^2 = 1$$



Quaternion Product

$$\begin{pmatrix} w_1 \\ \mathbf{v}_1 \end{pmatrix}\begin{pmatrix} w_2 \\ \mathbf{v}_2 \end{pmatrix} = \begin{pmatrix} w_1 w_2 - \mathbf{v}_1 \cdot \mathbf{v}_2 \\ w_1 \mathbf{v}_2 + w_2 \mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2 \end{pmatrix}$$

$$\begin{pmatrix} w_1 \\ \mathbf{v}_1 \end{pmatrix}\begin{pmatrix} w_2 \\ \mathbf{v}_2 \end{pmatrix} \neq \begin{pmatrix} w_2 \\ \mathbf{v}_2 \end{pmatrix}\begin{pmatrix} w_1 \\ \mathbf{v}_1 \end{pmatrix}$$

Quaternion Conjugate

$$\mathbf{q}^* = \begin{pmatrix} w_1 \\ \mathbf{v}_1 \end{pmatrix}^* = \begin{pmatrix} w_1 \\ -\mathbf{v}_1 \end{pmatrix}$$

$$(\mathbf{p}^*)^* = \mathbf{p}$$

$$(\mathbf{pq})^* = \mathbf{q}^* \mathbf{p}^*$$

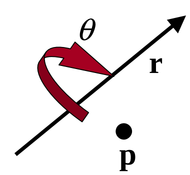
$$(\mathbf{p} + \mathbf{q})^* = \mathbf{p}^* + \mathbf{q}^*$$

Quaternion Inverse

$$\mathbf{q}^{-1}\mathbf{q} = 1$$

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

Quaternion Rotation



$$\begin{aligned} \mathbf{qpq}^{-1} &= \begin{pmatrix} w \\ \mathbf{v} \end{pmatrix}\begin{pmatrix} 0 \\ \mathbf{p} \end{pmatrix}\begin{pmatrix} w \\ -\mathbf{v} \end{pmatrix} \\ &= \begin{pmatrix} w \\ \mathbf{v} \end{pmatrix}\begin{pmatrix} \mathbf{p} \cdot \mathbf{v} \\ w\mathbf{p} - \mathbf{p} \times \mathbf{v} \end{pmatrix} \\ &= \begin{pmatrix} w\mathbf{p} \cdot \mathbf{v} - w\mathbf{p} \cdot \mathbf{v} = 0 \\ w(w\mathbf{p} - \mathbf{pv}) + (\mathbf{p} \cdot \mathbf{v})\mathbf{v} + \mathbf{v}(w\mathbf{p} - \mathbf{p} \times \mathbf{v}) \end{pmatrix} \end{aligned}$$

What about a quaternion product $\mathbf{q}_1\mathbf{q}_2$?

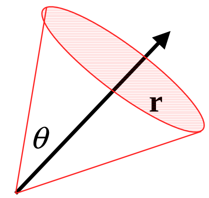
Matrix Form

$$\mathbf{q} = \begin{pmatrix} w \\ x \\ y \\ z \end{pmatrix}$$

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

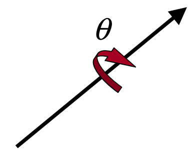
Quaternion constraints

Restricting the rotation cone



$$\frac{1-\cos(\theta_x)}{2} = q_y{}^2 + q_z{}^2$$

Restricting the rotation twist around an axis



$$\tan(\theta/2) = \frac{q_{axis}}{q_w}$$

Quaternions: 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)

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

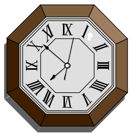
$$m = \frac{s}{60}$$

$$h = \frac{m}{60}$$

$$\theta_s = \frac{2\pi s}{60}$$

$$\theta_m = \frac{2\pi m}{60}$$

$$\theta_h = \frac{2\pi h}{12}$$



Models as Code: draw-a-bug

```
void draw_bug(walk_phase_angle, xpos, ypos zpos){
  pushmatrix
  translate(xpos,ypos,zpos)
  calculate all six sets of leg angles based on
    walk phase angle.
  draw bug body
  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???