Advanced Computer Graphics
CSE 190 [Spring 2015], Lecture 17
Ravi Ramamoorthi
http://www.cs.ucsd.edu/~ravir

To Do
- Assignment 3 milestone due May 29
- This lecture about animation: Inverse Kinematics
  - Imaging, Texture Synthesis next week

Course Outline
- 3D Graphics Pipeline
  - Rendering (Creating, shading images from geometry, lighting, materials)
  - Modeling (Creating 3D Geometry)

The Story So Far
scene → image

Animation
scene(t) → image(t)

Forward Kinematics
- Root body
  - Position set by global transform
- Root joint: position, rotation
- Other bodies relative to root
  - Inboard toward the root
  - Outboard away from the root
- Tree structure: loop joints break “tree-ness”
Inboard and Outboard Joints

- Inboard body
- Outboard body

Body
- Inboard joint
- Outboard joint (may be several)

Bodies

- Bodies arranged in a tree
- For now, assume no loops
- Body’s parent (except root)
- Body’s child (may have many children)

Joints

- Interior Joints (typically not 6 DOF)
  - Pin – rotate about one axis
  - Ball – arbitrary rotation
  - Prism – translate along one axis

Pin Joints

- Translate inboard joint to local origin
- Apply rotation about axis
- Translate origin to location of joint on outboard body

Ball Joints

- Translate inboard point to local origin
- Apply rotation about arbitrary axis
- Translate origin to location of joint on outboard body
**Prism Joint**
- Translate inboard joint to local origin
- Translate along axis
- Translate origin to location of joint on outboard

**Forward Kinematics**
- Composite transformations up the hierarchy

**Inverse Kinematics**
- Given
  - Root transformation
  - Initial configuration
  - Desired end point location
- Find
  - Interior parameter settings

**2 Segment Arm in 2D**

\[
\begin{align*}
 p_x &= l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \\
 p_y &= l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2)
\end{align*}
\]

**Direct IK**
- Analytically solve for parameters (not general)

\[
\begin{align*}
 \theta_2 &= \cos^{-1} \left( \frac{p_x^2 + p_y^2 - l_1^2 - l_2^2}{2l_1 l_2} \right) \\
 \theta_1 &= \frac{-p_x l_2 \sin(\theta_2) + p_y (l_1 + l_2 \cos(\theta_2))}{p_x l_2 \sin(\theta_2) + p_y (l_1 + l_2 \cos(\theta_2))}
\end{align*}
\]
**Difficult Issues**

- Multiple configurations distinct in config space
- Or connected in config space

**Infeasible Regions**

**Numerical Solution**

- Start in some initial config. (previous frame)
- Define error metric (goal pos – current pos)
- Compute Jacobian with respect to inputs
- Use Newton’s or other method to iterate
- General principle of goal optimization

**Back to 2 Segment Arm**

**Jacobians and Configuration Space**

The Jacobian (of \( p \) w.r.t. \( \theta \))

\[
J = \begin{bmatrix}
\frac{\partial p_x}{\partial \theta_1} & \frac{\partial p_x}{\partial \theta_2} \\
\frac{\partial p_y}{\partial \theta_1} & \frac{\partial p_y}{\partial \theta_2}
\end{bmatrix}
\]

\[
\frac{\partial p}{\partial \theta} = J \cdot \begin{bmatrix}
\frac{\partial c_1}{\partial \theta_1} \\
\frac{\partial c_1}{\partial \theta_2}
\end{bmatrix}
\]

**Solving for Joint Angles**

Solving for \( c_1 \) and \( c_2 \)

\[
c = \begin{bmatrix}
c_1 \\
\end{bmatrix}
\]

\[
dp = \begin{bmatrix}
dp_x \\
\end{bmatrix}
\]

\[
dp = J^{-1} \cdot dp
\]

\[
c = J^{-1} \cdot dp
\]
**Issues**

- Jacobian not always invertible
  - Use an SVD and pseudo-inverse
- Iterative approach, not direct
  - The Jacobian is a linearization, changes
- Practical implementation
  - Analytic forms for prism, ball joints
  - Composing transformations
  - Or quick and dirty: finite differencing
  - Cyclic coordinate descent (each DOF one at a time)

**Prism and Ball Joints**

**More on Ball Joints**

- **Ball Joints (moving axis)**
  \[ \mathbf{d} = \frac{1}{2} \mathbf{r}^{T} \mathbf{r} \mathbf{x} = [r] \mathbf{p} = [\mathbf{r}] \mathbf{p} \]

- **Ball Joints (fixed axis)**
  \[ \mathbf{d} = \mathbf{P} (\mathbf{r}^{T} \mathbf{x} = -\mathbf{p} - \mathbf{p} \mathbf{\Omega}) \]

**Multiple Links**

- IK requires Jacobian
  - Need generic method for building one
- Can’t just concatenate matrices

**Composing Transformations**

- **Transformation from body to world**
  \[ X_{i-1} = R_{i-1} X_{j-1-1} = X_{i-1} X_{j-1-2} \cdots \]

- **Rotation from body to world**
  \[ R_{i-1} = R_{i-1-1} R_{i-1-2} \cdots \]

**Inverse Kinematics: Final Form**

- **Jacobian for chain of links**
  \[ J = R_{0-2b} J_{3}(\theta_{3}, \mathbf{p}_{3}) R_{0-2a} J_{2a}(\theta_{2a}, \mathbf{X}_{2a-3} \cdot \mathbf{p}_{3}) R_{0-1} J_{2a}(\theta_{2a}, \mathbf{X}_{2a-3} \cdot \mathbf{p}_{3}) J_{1}(\theta_{1}, \mathbf{X}_{1-3} \cdot \mathbf{p}_{3}) \]

- **Position vector**
  \[ \mathbf{d} = \begin{bmatrix} d_{3} \\ d_{2a} \\ d_{2b} \\ d_{3b} \end{bmatrix} \]

- **Joint positions**
  \[ \mathbf{d} = J \cdot \mathbf{d} \]
A Cheap Alternative

- Estimate Jacobian (or parts of it) w. finite diffs.
- Cyclic coordinate descent
  - Solve for each DOF one at a time
  - Iterate till good enough / run out of time

More complex systems

- More complex joints (prism and ball)
- More links
- Other criteria (center of mass or height)
- Hard constraints (e.g., foot plants)
- Unilateral constraints (e.g., joint limits)
- Multiple criteria and multiple chains
- Loops
- Smoothness over time
  - DOF determined by control points of curve (chain rule)

Practical Issues

- How to pick from multiple solutions?
- Robustness when no solutions
- Contradictory solutions
- Smooth interpolation
  - Interpolation aware of constraints

Prior on “good” configurations