CS 4732:
Computer Animation

Kinematic Linkages

Robert W. Lindeman
Associate Professor
Interactive Media & Game Development
Department of Computer Science
Worcester Polytechnic Institute
gogo@wpi.edu

Kinematics

- The world is inherently relative
  - E.g., how do we define the planets?
- Most of this is hierarchical
  - Moon relative to the Earth, Earth to Sun
  - Things are on top of, connected to, etc.
- Here we are interested in animating objects whose motion is relative to other objects
  - Many things are naturally hierarchical

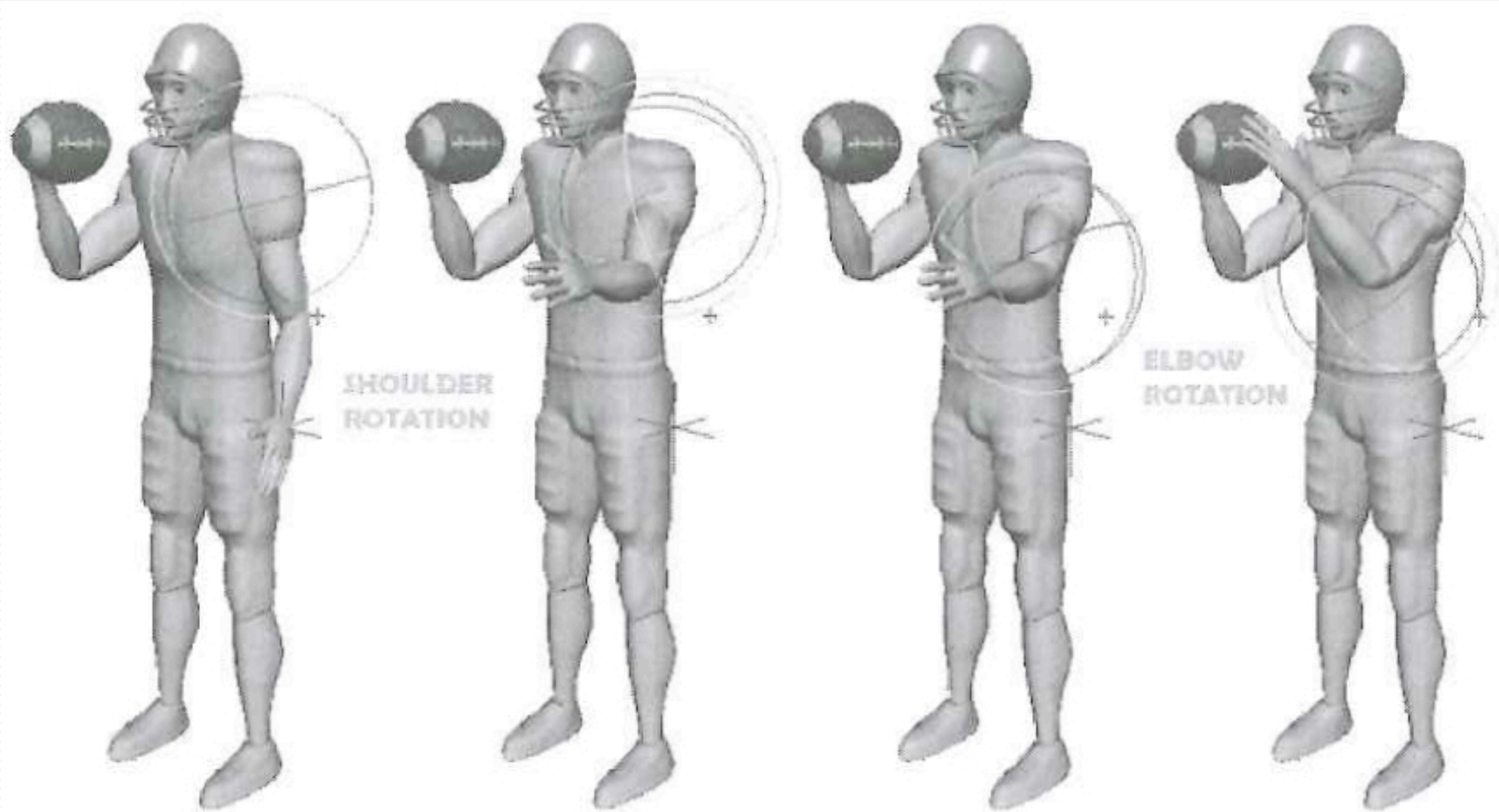
Motion Hierarchies

- Sequence of relative motions
- *Linked appendages* or *Linkages*
- These motions are typically restricted
  - Position of moon can be specified with 1 DOF
  - Called *reduced dimensionality*
- Determining position parameters over time is called *kinematics*
- Limbed motion is most common use

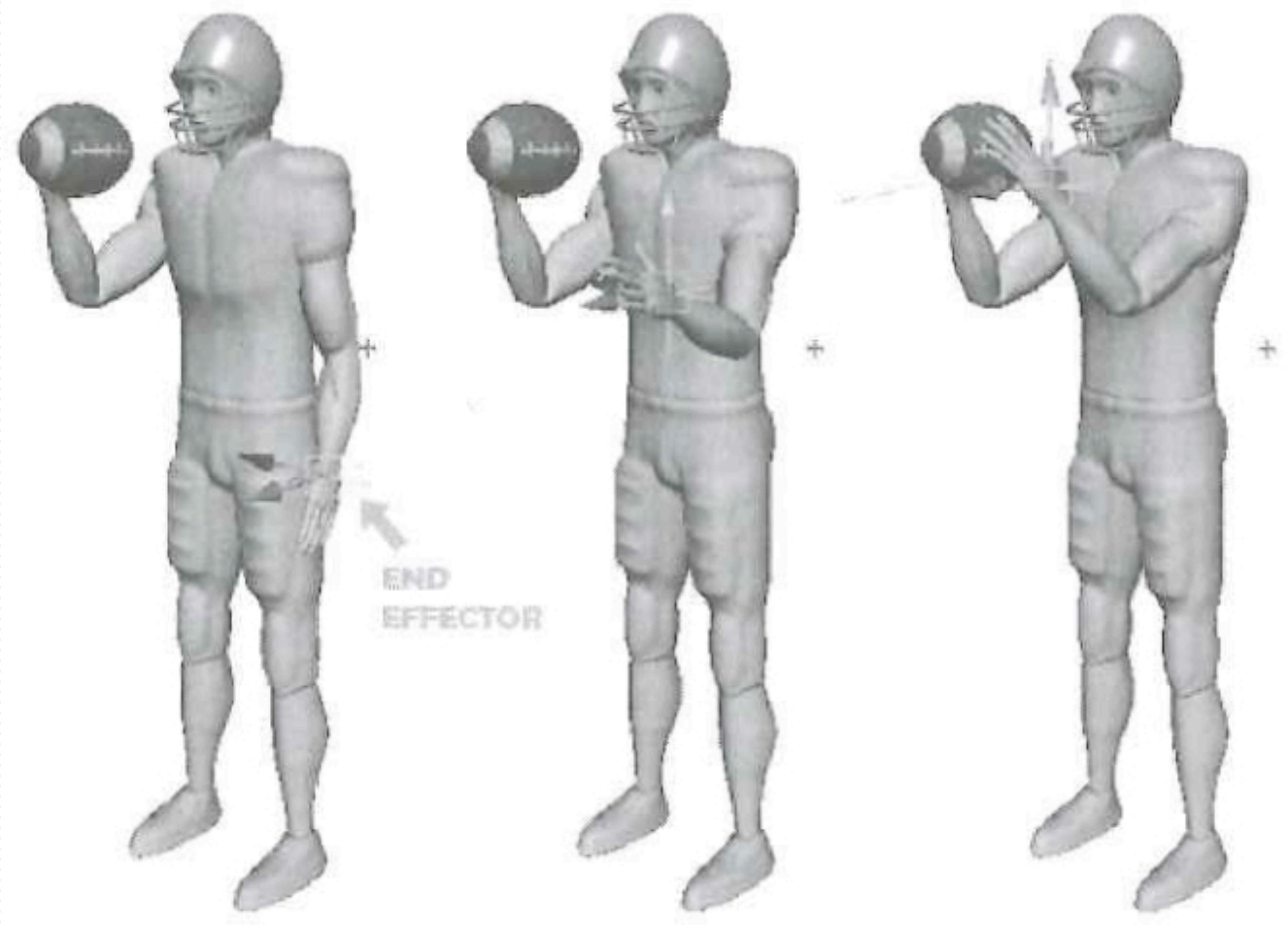
Two Main Flavors of Kinematics

- Forward kinematics
  - Animator specifies rotational parameters at joints
  - End effector position is well defined
- Inverse kinematics
  - Animator specifies the position of the end effector
  - System solves for joint angles
  - Many solutions are possible!

Forward Kinematics Example



Inverse Kinematics Example

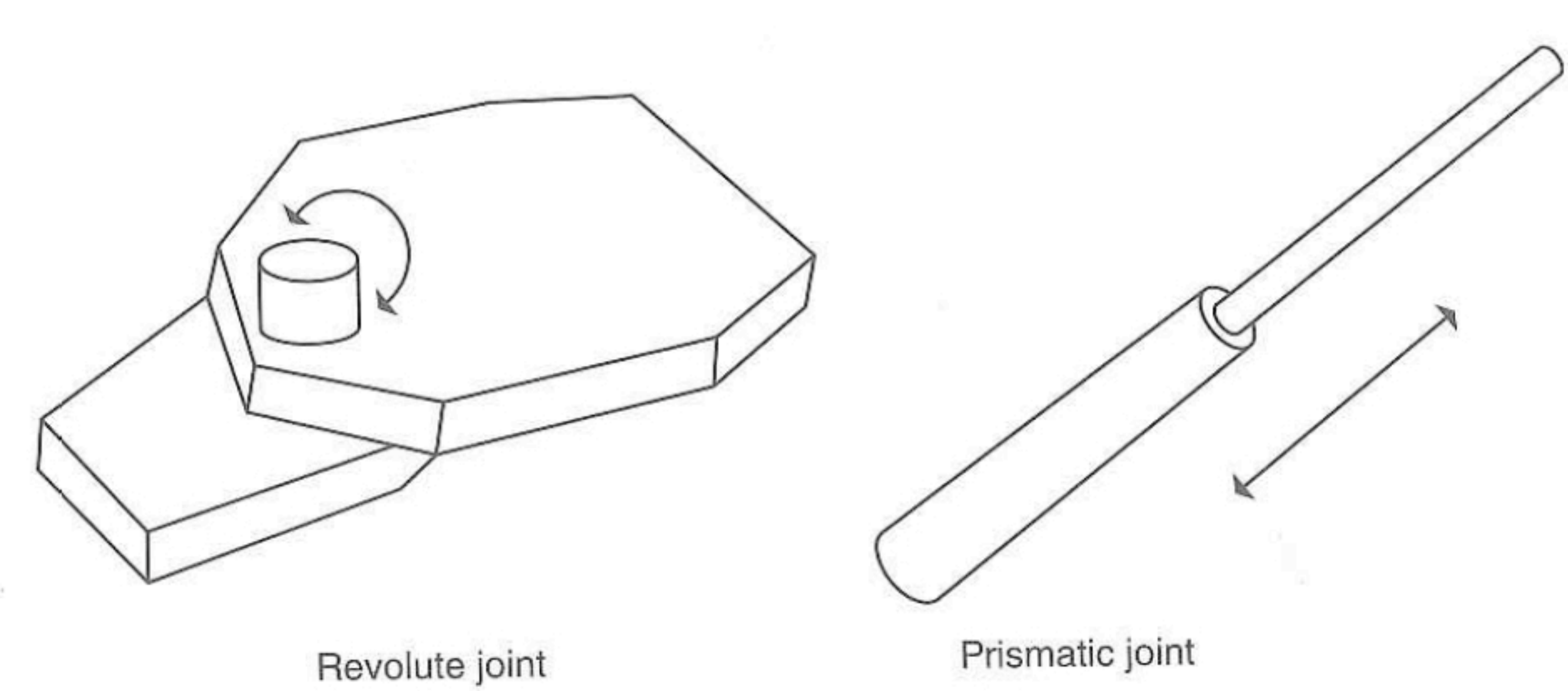


Hierarchical Modeling

- Tree-like hierarchy of relative location constraints
  - Moons, planets, suns, galaxies, etc.
- Many models use end-to-end connections
  - Also called *articulated figures*
- Much of this comes from robotics
  - Linkages are called ***manipulators***
  - Objects are called ***links***
  - Connections are called ***joints***
  - Free end is called the ***end effector***
  - Object coordinate system is called the ***frame***

Joints

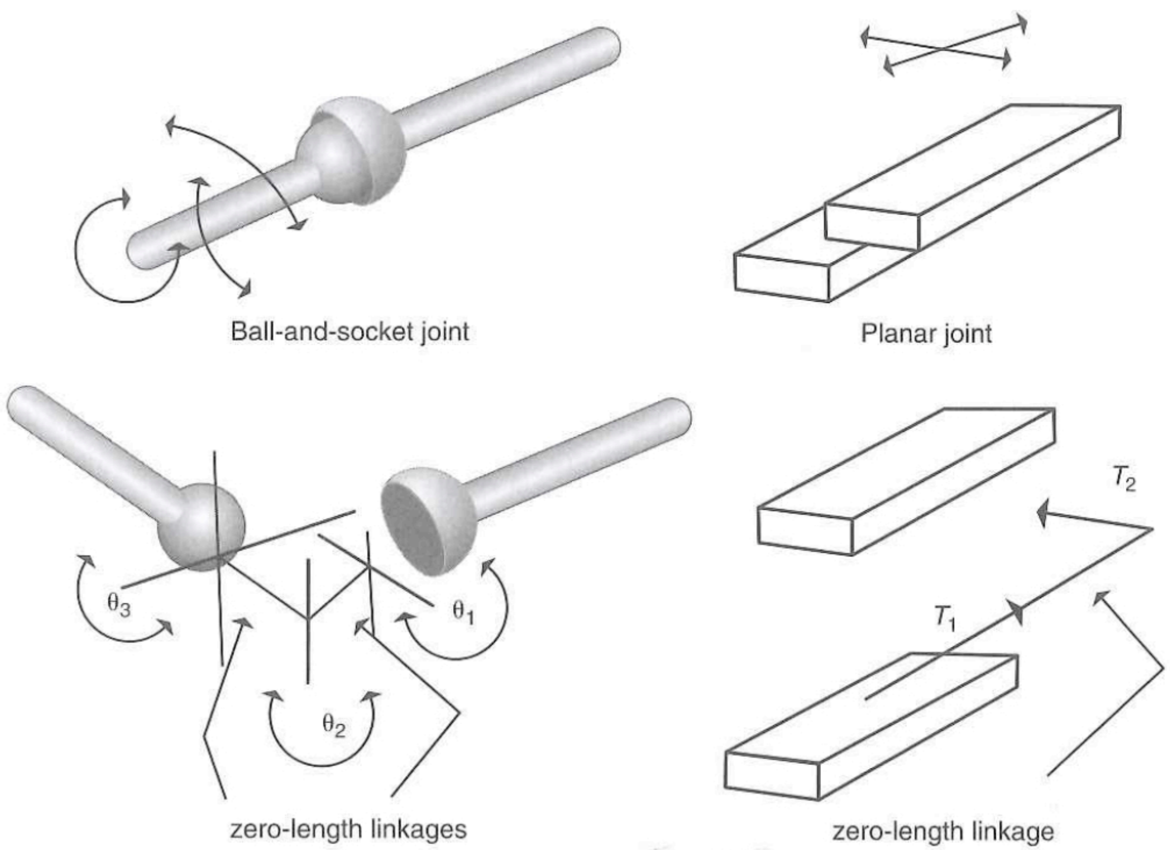
- Robotics usually deals with *revolute* or *prismatic* joints



Joints (cont.)

- Each direction of movement is called a *Degree of freedom* (*DOF*)
- Simple joints have one DOF
  - E.g., a hinge
- Complex joints have multiple DOFs
  - E.g., a ball-and-socket joint
- We usually model these as multiple one-DOF joints
  - But you can also just use, e.g., a quaternion

Complex Joints



Ball-and-socket joint modeled as 3 one-degree joints with zero-length links

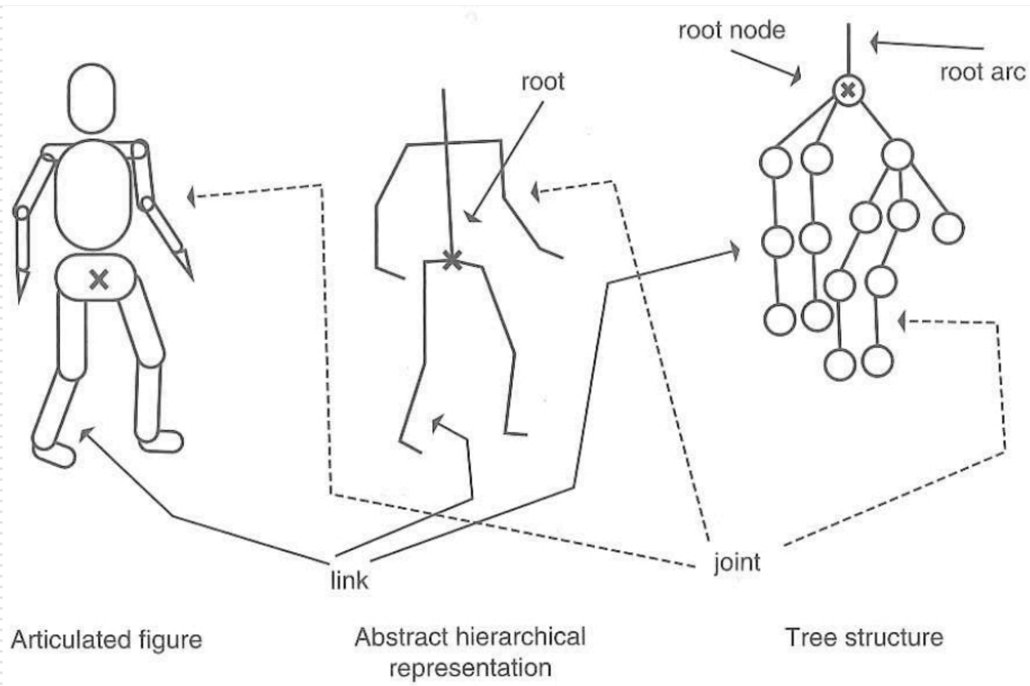
Planar joint modeled as 2 one-degree prismatic joints with zero-length links

Hierarchical Data Structure

- Articulated figures can be defined using a tree of nodes and arcs (edges)
- *Root node* is specified in world coordinates
  - All other nodes are relative
- "Up the hierarchy" means closer to root
- A *parent* is above a *child* node
- A *leaf* is an end effector
- A *node* is an object, and an *arc* is a joint

Hierarchical Data Structure (cont.)

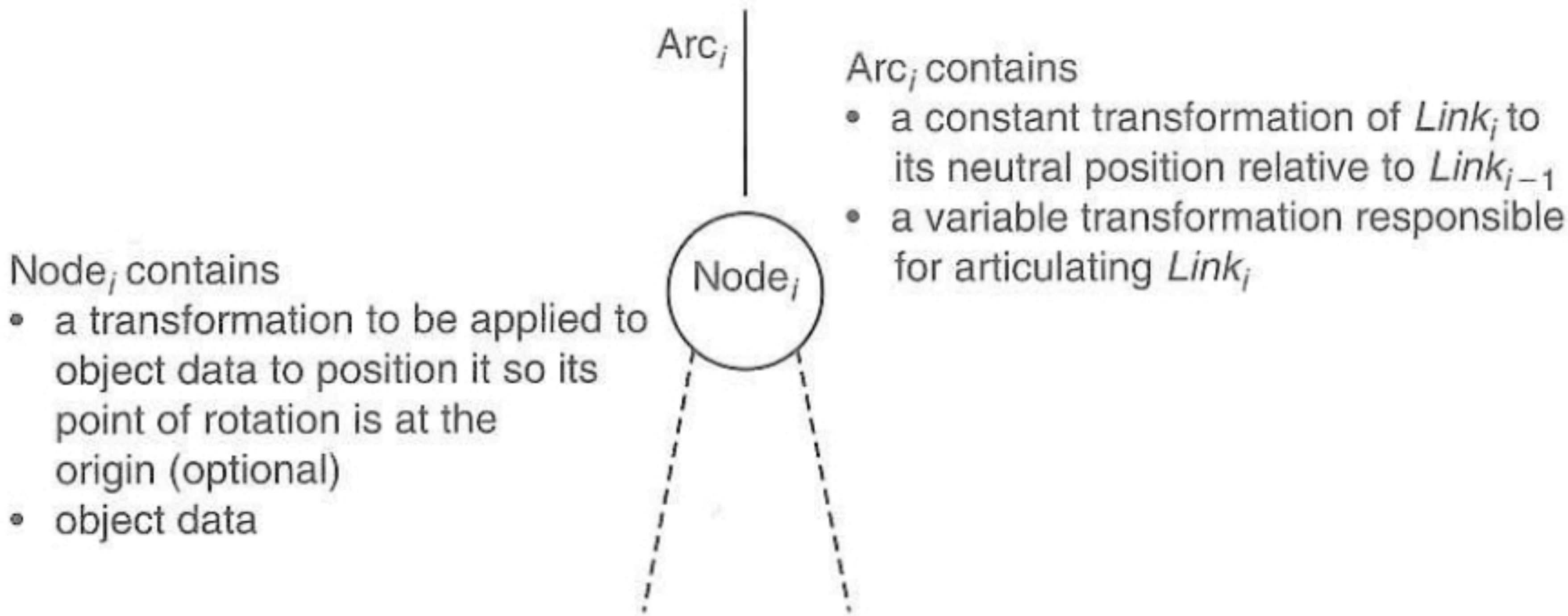
- Root arc defines the location of the root node in world space
  - And therefore all nodes in the tree!



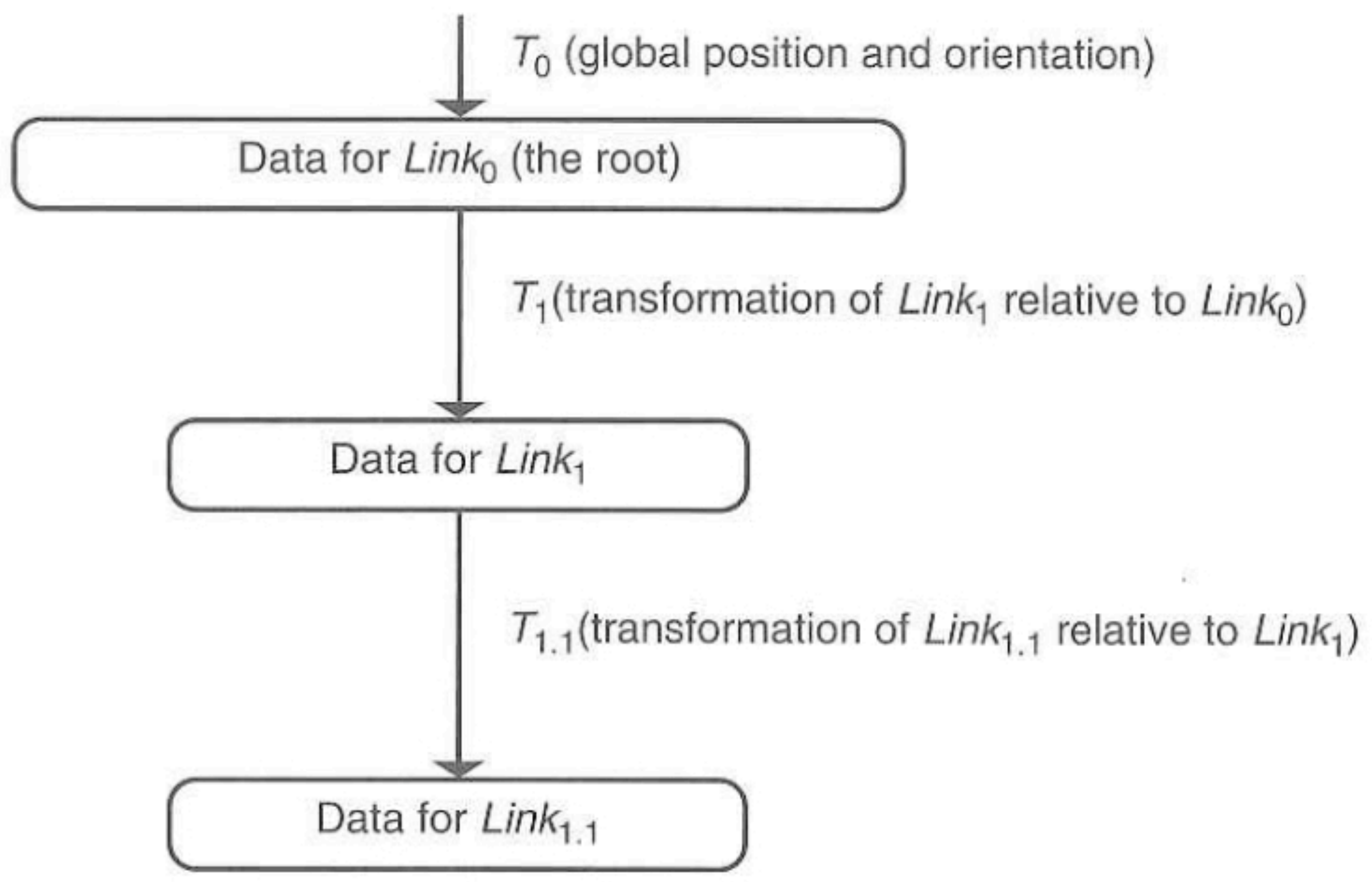
Anatomy of a Nodes & Arcs

- A node contains all the static information to get it ready for articulation
  - Information about geometry, etc. for drawing
  - An optional transformation to position it at the desired center of rotation
- An arc contains a static and a dynamic part
  - Static transformation in parent space to the location of the node
    - This is the link's "neutral" position relative to its parent
  - Dynamic transformation describing the actual joint articulation

Anatomy of a Nodes & Arcs (cont.)



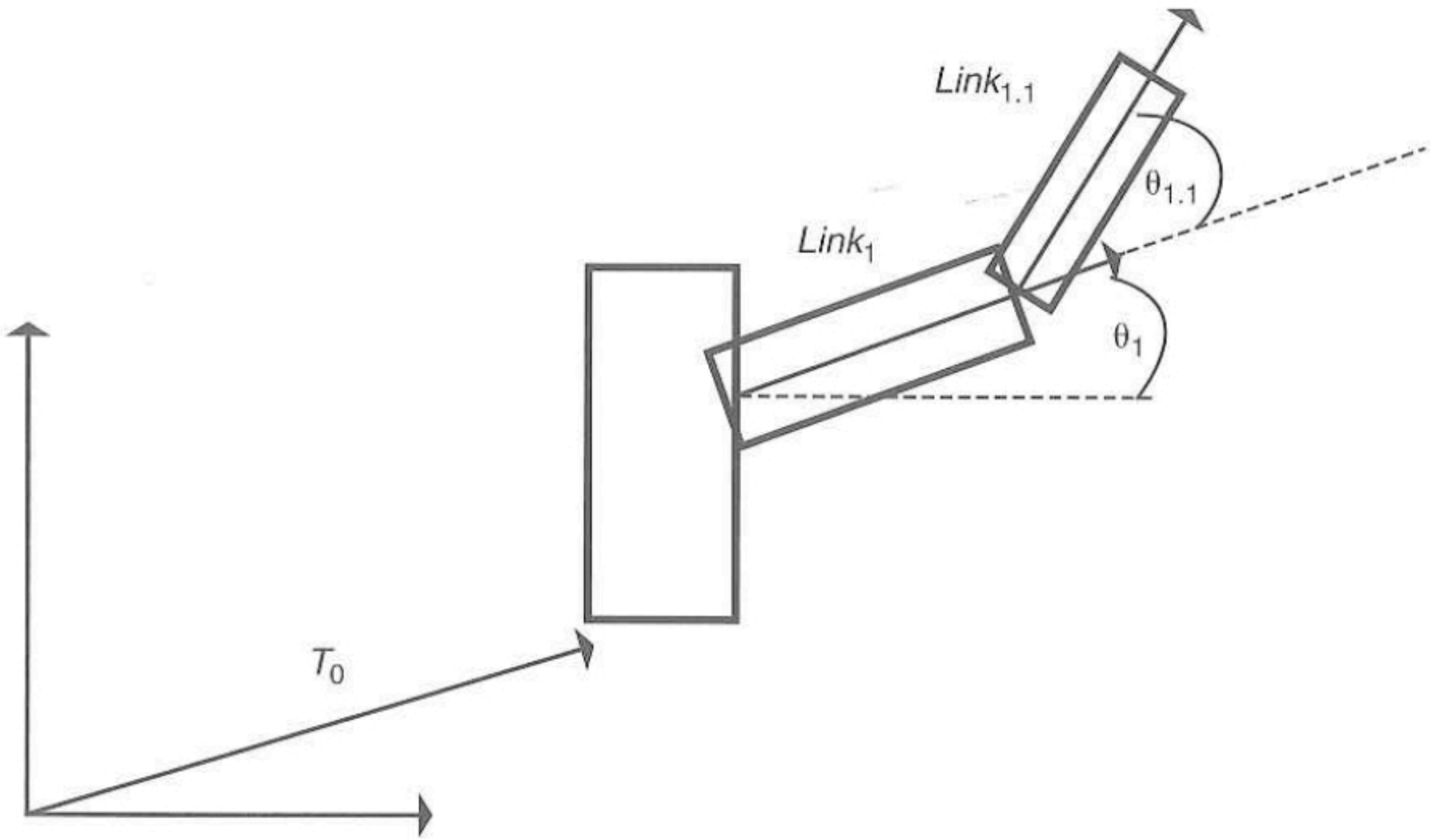
Hierarchy Example



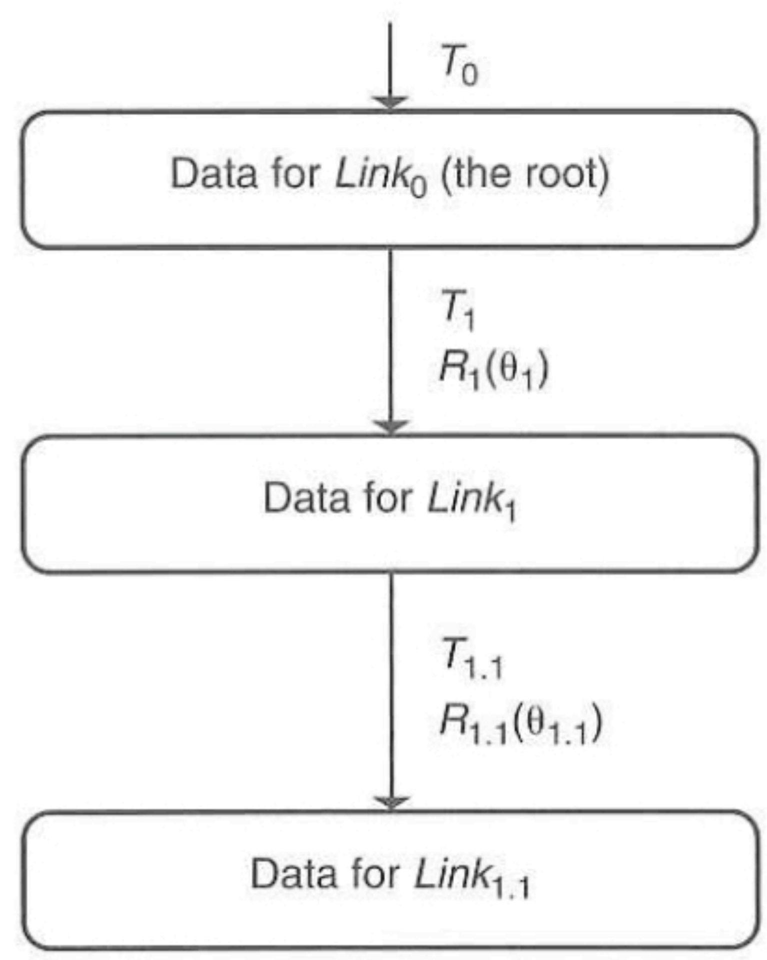
How to Apply Transformations

- Each node has an arc relative to its parent
  - How would you represent this in OpenGL?

How to Apply Transformations (cont.)

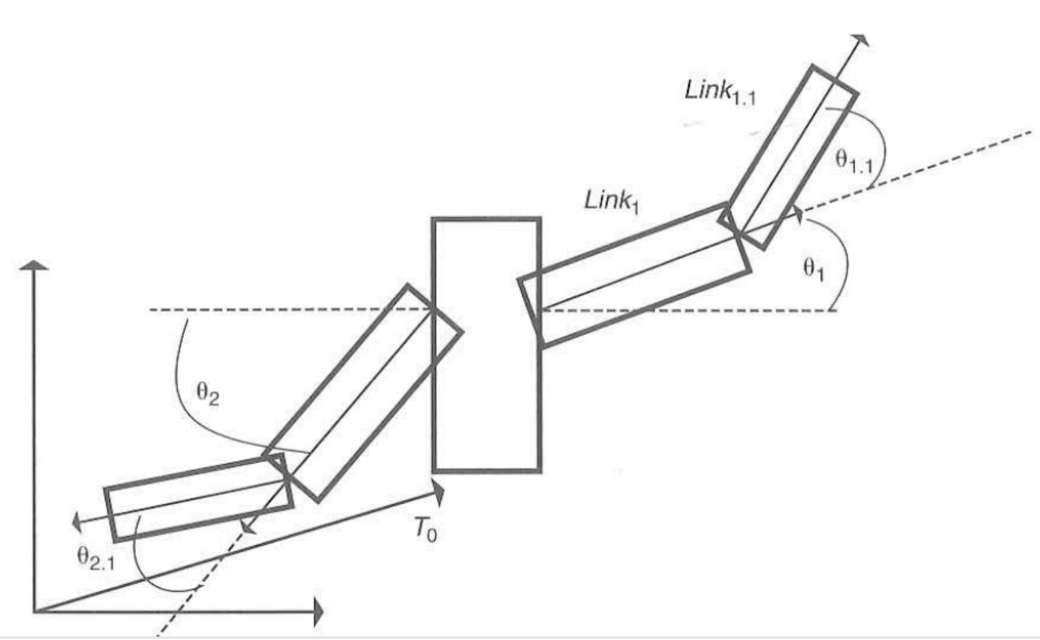


How to Apply Transformations (cont.)

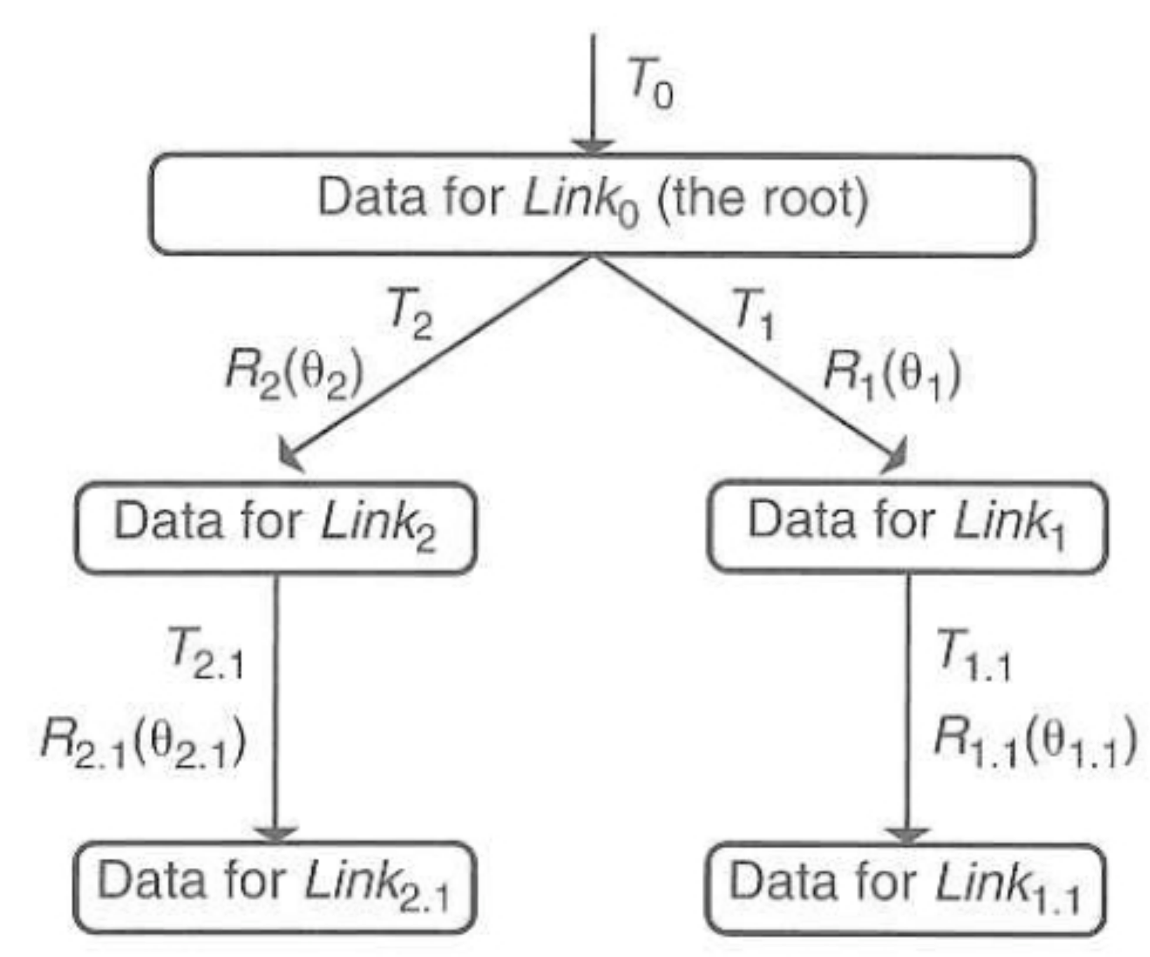


How to Apply Transformations (cont.)

- The rotational transformation is applied *before* the arc's constant transformation
- If there is an optional transformation, it is applied *before* the rotation



Multiple Appendages



Applying Forward Kinematics

- We need to walk the tree in a depth-first manner
  - Apply the transforms
  - Draw geometry
  - Save the current state
  - Recursively traverse each child arc, restoring the saved state before diving in
- A *pose vector* is used to define each DOF in the tree
- These DOFs are set using anything you like!
  - Such as key framing, physics, etc.
- How would this look in code?

Inverse Kinematics

- Animator specifies
  - Position (+ orientation) of the end effector
  - Starting pose vector
- Computer finds
  - The final pose vector (joint angles)
- May be 0, 1, or many solutions
- If no solutions, problem is probably *overconstrained*
- Too many solutions: underconstrained

Inverse Kinematics (cont.)

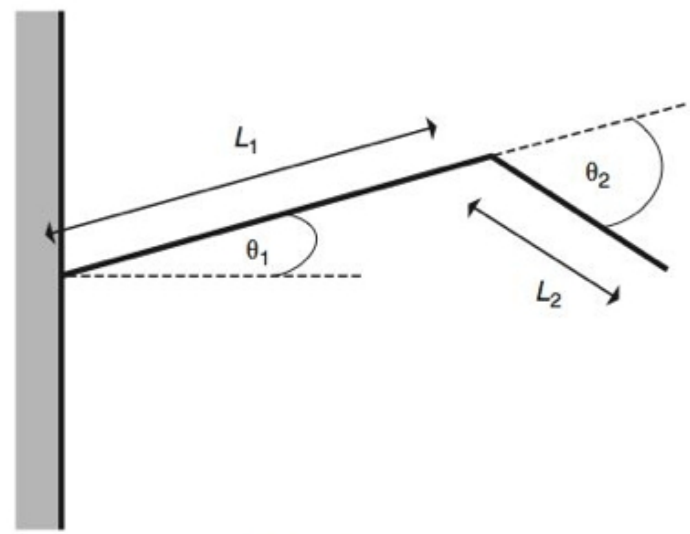
- The reachable workspace is the volume the end effector can reach.
- After finding the final pose vector, interpolate
- May have to break the motion down into several intermediate poses to get good control
- If the motion is simple, just interpolate
- If it is complex, then we use the *Jacobian*

The Jacobian

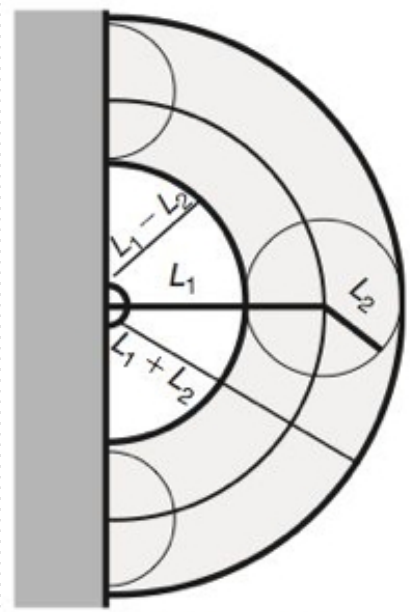
- An incremental approach
- Employs a matrix of values that relates changes in joints to end effector position and orientation
- End effector is iteratively nudged until the final configuration is attained within a given tolerance
- The Jacobian is just one iterative approach

Simple Systems: Analytic Solutions

- Consider a 2D example, with links L_1 & L_2
- If one end of L_1 is fixed to a base
  - Any location closer than $|L_1 - L_2|$ or farther than $|L_1 + L_2|$ is unreachable



Configuration



Reachable workspace

Simple Systems: Analytic Solutions (cont.)

- If we place the end effector at an (*X*, *Y*)
- The two angles θ_1 & θ_2 can be computed using the distance and the law of cosines (see book)
- In this simple case, there are only two solutions
- As you increase the number of DOFs, things get very complex, very fast!

Incremental Solutions

- For most things we want to do, analytic solutions are not available.
- Instead, we carry out a series of changes that move us incrementally closer
- Several methods exist, and most involve a matrix of partial derivatives
  - This is called **the Jacobian**
- So, the Jacobian is a way of representing the change in position/orientation of all the degrees of freedom in a kinematic chain

Formulating the Jacobian

- Consider the six equations:

$$y_1 = f_1(x_1, x_2, x_3, x_4, x_5, x_6)$$
$$y_2 = f_2(x_1, x_2, x_3, x_4, x_5, x_6)$$
$$y_3 = f_3(x_1, x_2, x_3, x_4, x_5, x_6)$$
$$y_4 = f_4(x_1, x_2, x_3, x_4, x_5, x_6)$$
$$y_5 = f_5(x_1, x_2, x_3, x_4, x_5, x_6)$$
$$y_6 = f_6(x_1, x_2, x_3, x_4, x_5, x_6)$$

- Using the chain rule, we can rewrite as the change in outputs based on changes in inputs:

$$dy_i = \frac{\partial f_i}{\partial x_1} dx_1 + \frac{\partial f_i}{\partial x_2} dx_2 + \frac{\partial f_i}{\partial x_3} dx_3 + \frac{\partial f_i}{\partial x_4} dx_4 + \frac{\partial f_i}{\partial x_5} dx_5 + \frac{\partial f_i}{\partial x_6} dx_6$$

Formulating the Jacobian (cont.)

- Or in vector notation:

$$dY = \frac{\partial F}{\partial X} dX$$

- The Jacobian can be thought of as mapping the velocities of *X* to the velocities of *Y*
- At any point in time, the Jacobian is a function of the current values of x_i

Formulating the Jacobian (cont.)

- To apply the Jacobian to a linked appendage
  - The input variables, x_i, become the joint values
  - The output variables, y_i, become the end effector position and orientation

$$Y = [p_x p_y p_z \alpha_x \alpha_y \alpha_z]^T$$

  - This relates the velocities of the joint angles, $\dot{\theta}$, to the velocities of the end effector position and orientation, $\dot{Y}$

$$V = \dot{Y} = J(\theta)\dot{\theta}$$

Formulating the Jacobian (cont.)

- V is the vector of linear and rotational velocities
  - Represents the desired change in the end effector

$$V = \dot{Y} = J(\theta)\dot{\theta}$$

- The desired change will be based on the difference between the current position/ orientation and the goal configuration

$$V = [v_x v_y v_z \omega_x \omega_y \omega_z]^T$$

Formulating the Jacobian (cont.)

- $\dot{\theta}$ is a vector of joint value velocities, or the changes to the joint parameters, which are the unknowns of the equation

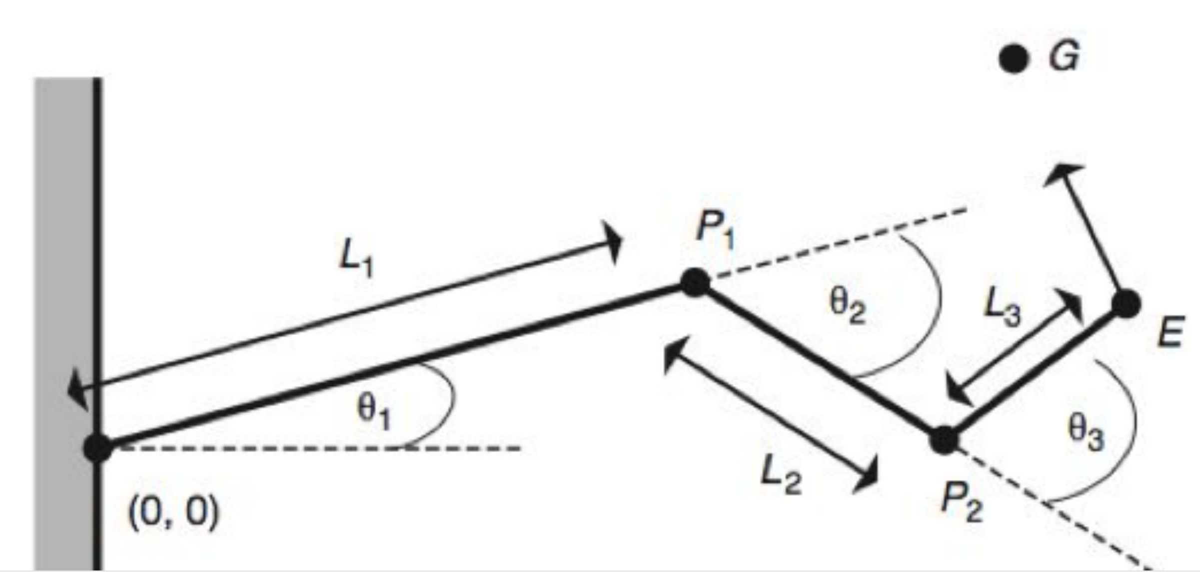
$$\dot{\theta} = [\dot{\theta}_1 \dot{\theta}_2 \ldots \dot{\theta}_n]^T$$

- *J*, the Jacobian, is a matrix that relates the two and is a function of the current pose

$$J = \begin{bmatrix} \frac{\partial p_x}{\partial \theta_1} & \frac{\partial p_x}{\partial \theta_2} & \ldots & \frac{\partial p_x}{\partial \theta_n} \\ \frac{\partial p_y}{\partial \theta_1} & \frac{\partial p_y}{\partial \theta_2} & \ldots & \frac{\partial p_y}{\partial \theta_n} \\ \ldots & \ldots & \ldots & \ldots \\ \frac{\partial \alpha_z}{\partial \theta_1} & \frac{\partial \alpha_z}{\partial \theta_2} & \ldots & \frac{\partial \alpha_z}{\partial \theta_n} \end{bmatrix}$$

An Example

- Consider simple planar linkage
  - The joint axes are coming out of the screen
  - We want to find θ_1, θ_2 & θ_3

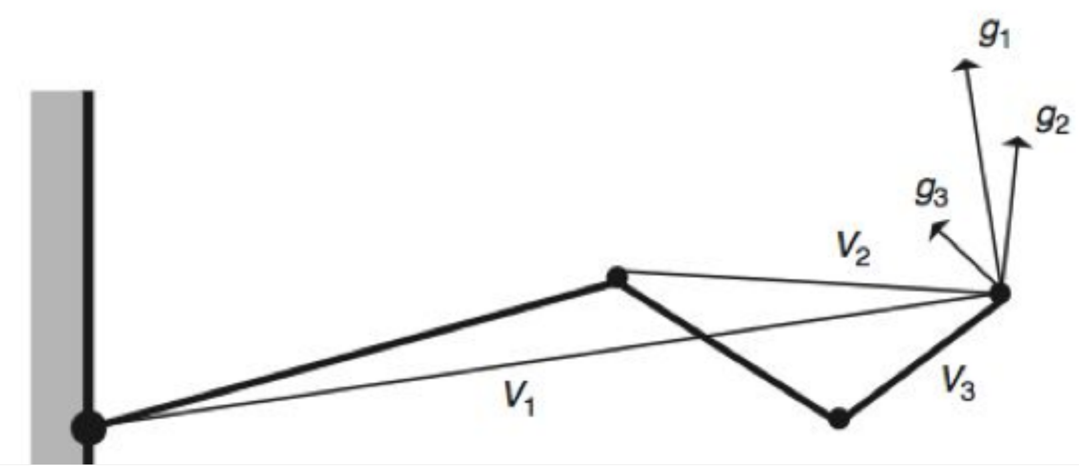


- V is the desired change:

$$V = \begin{bmatrix} (G-E)_x \\ (G-E)_y \\ (G-E)_z \end{bmatrix}$$

An Example (cont.)

- The effect of an incremental rotation, g_i, can be determined by crossing the joint axis and the vector from the joint to the end effector, V_i
  - These form the columns of the Jacobian



$$J = \begin{bmatrix} ((0,0,1) \times E)_x & ((0,0,1) \times (E - P_1)_x & ((0,0,1) \times (E - P_2)_x \\ ((0,0,1) \times E)_y & ((0,0,1) \times (E - P_1)_y & ((0,0,1) \times (E - P_2)_y \\ ((0,0,1) \times E)_z & ((0,0,1) \times (E - P_1)_z & ((0,0,1) \times (E - P_2)_z \end{bmatrix}$$

Solutions Using the Inverse Jacobian

- Once the Jacobian has been computed, we need to solve the equation:

$$V = J\dot{\theta}$$

- In the case that *J* is a square matrix, the inverse of the Jacobian, J^{-1}, is used to compute the joint angle velocities given the end effector velocities

$$J^{-1}V = \dot{\theta}$$

Solutions Using the Inverse Jacobian (cont.)

- If no inverse Jacobian exists, the system is said to be singular for the given joint angles
  - A linear combination of joint angle velocities cannot be formed to produce desired end effector velocities
- This can be solved if there are more DOFs than constraints to be satisfied
  - Leads to:
    - A potentially infinite number of solutions
    - A non-square Jacobian

Solutions Using the Inverse Jacobian (cont.)

- Lack of a square Jacobian can be remedied using a *pseudoinverse*, J^+
  - If there are more columns than rows in J

$$J^+ = (J^T J)^{-1} J^T$$

  - If there are more rows than columns in J

$$J^+ = J^T (J J^T)^{-1}$$

- Book gives details on further steps

Cyclic Coordinate Descent (CCD)

- Consider each joint sequentially, from the outermost inwards
  - Choose an angle that best gets the end effector to the goal position
- Let's look in more detail
- http://freespace.virgin.net/hugo.elias/models/m_ik2.htm

More References

- http://freespace.virgin.net/hugo.elias/models/m_ik2.htm
- http://grail.cs.washington.edu/projects/styleik/