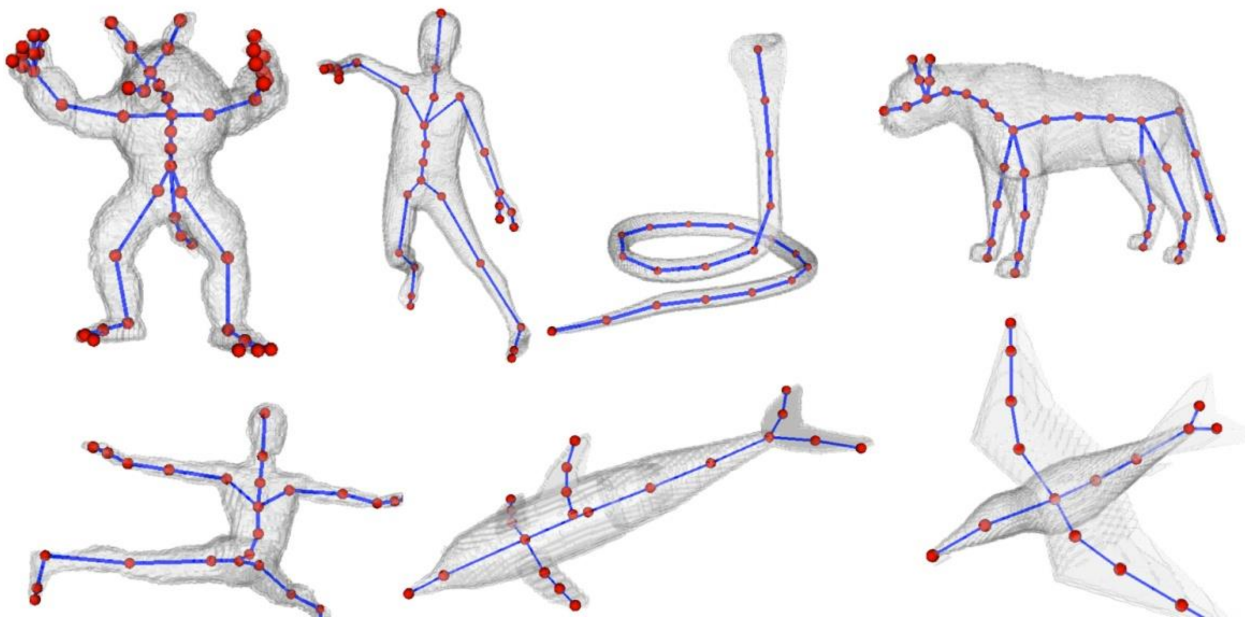
Articulated Characters

Skeleton

- A skeleton is a framework of <u>rigid body</u> “bones” connected by articulated joints
- Used as an (invisible?) armature to position and orient geometry (usually surface triangles)



Joints

- Joints connect rigid bodies together, while allowing for relative motion between them
- Different types: hinge, ball-and-socket, saddle joint, sliding…
- A Joint has 0-6 degrees of freedom (DoF)
  - A 0-DoF joint rigidly connects two bodies into a single rigid body
  - A full 6-DoF joint doesn't do anything, and each of the bodies are free to move entirely independently
  - Typical joints are somewhere in-between (i.e. from 1-5 DoF)

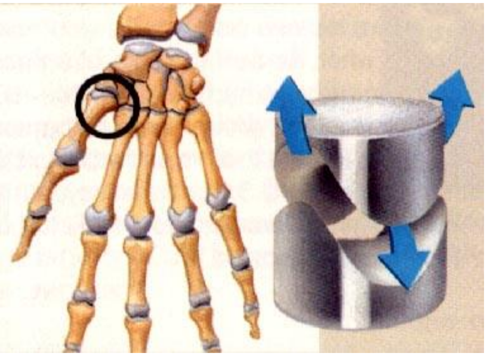
Joints

- Rotational Joints
  - 1-DoF: a rotation matrix $Rot_{4x4}(\mathbf{n},\theta)$ defined by axis **n** and angle θ
  - 2-DoF: multiplication of two sequential rotation matrices about different axes
  - 3-DoF: multiplication of 3 sequential rotation matrices about 3 different axes
    - Like Euler Angles, has the same problem of Gimbal lock
    - Better to specify a 3D rotation about an arbitrary axis (quaternion-style)
- Translational Joint
  - Can be specified to translate along any axis: (1-DoF, 2-DoF, 3-DoF)
  - Translation matrix $T_{4x4}(\mathbf{v})$ is defined by the translation vector **v**
- Compound Joint
  - Combines rotational and translational joints together

Examples

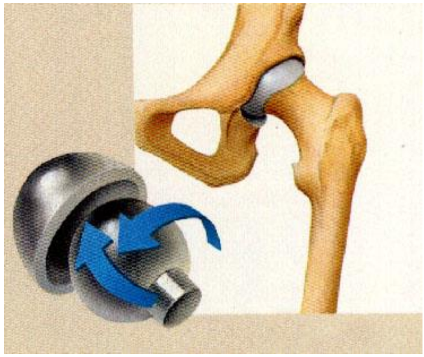
- rotation along one axis
- DoF=1

Hinge



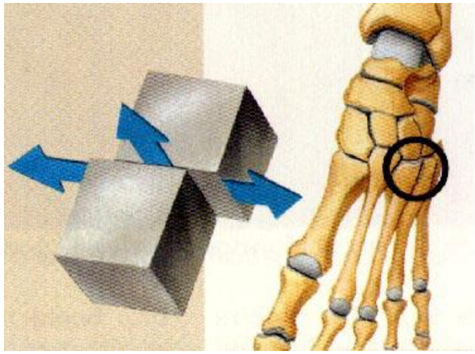
- back and forth & up and down motion
- no “rotation”
- DoF=2

Saddle



- rotation along all 3 axes
- DoF=3

Ball-and-socket



- sliding in a plane
- DoF=2

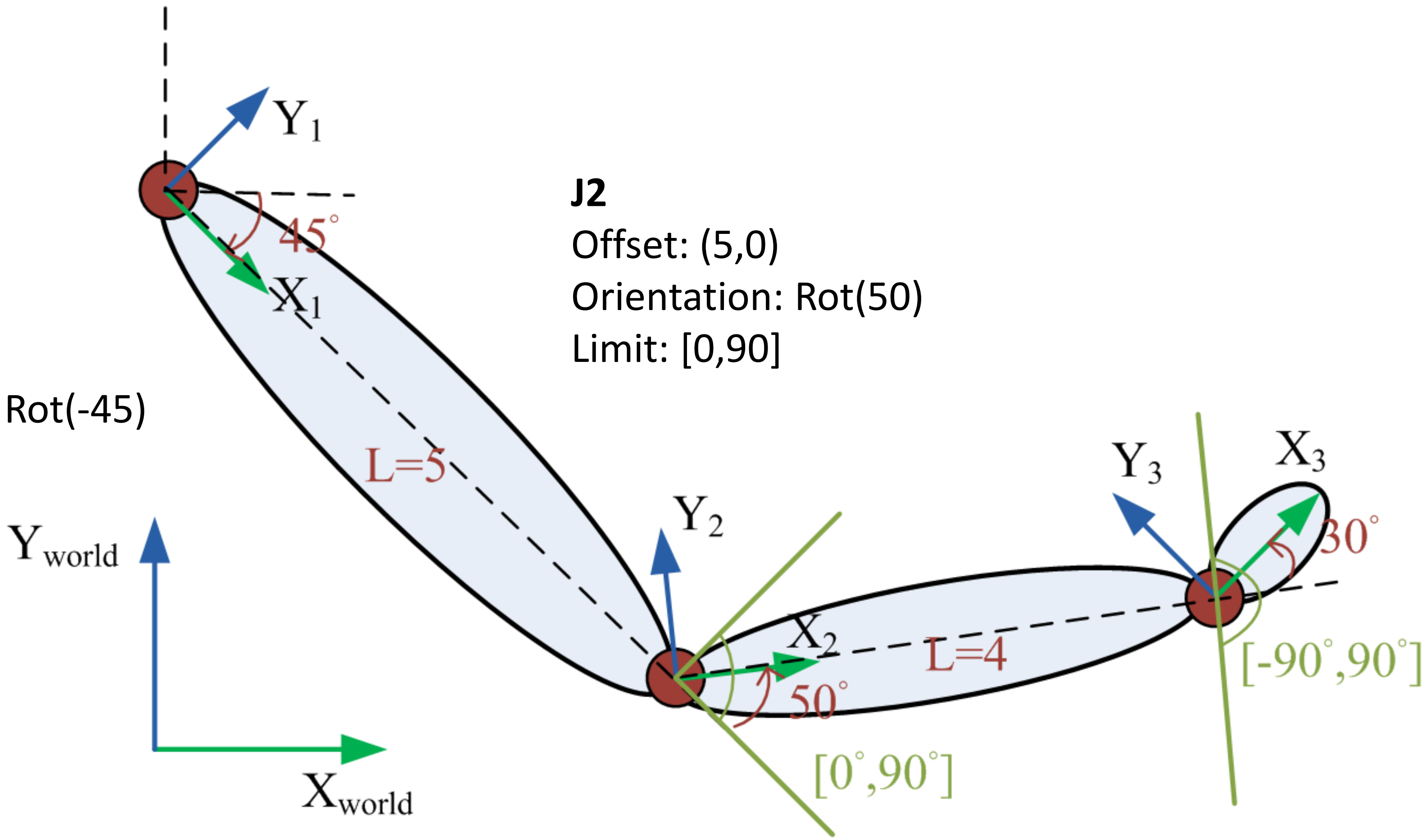
Sliding

Image from http://www.shockfamily.net/skeleton/JOINTS.HTML

Joint Parameters

- Offset
  - A fixed translational displacement in the space of the parent body/joint, which acts as a pivot point for the joint's movement
- Orientation
  - Orientation of a joint's local coordinate system defined in the space of its parent body/joint (a matrix or quaternion)
  - If using homogeneous coordinates, offset and orientation can be defined together in one 4x4 matrix
- Joint Limits
  - The minimum and maximum limits for each DOF that can be enabled or disabled independently
  - E.g. the human elbow can bend to about +150 degrees and hyperextend back as much as –10 degrees

Joint Parameters



J1
Offset: (0,6)
Orientation: Rot(-45)
Limit: None

J2
Offset: (5,0)
Orientation: Rot(50)
Limit: [0,90]

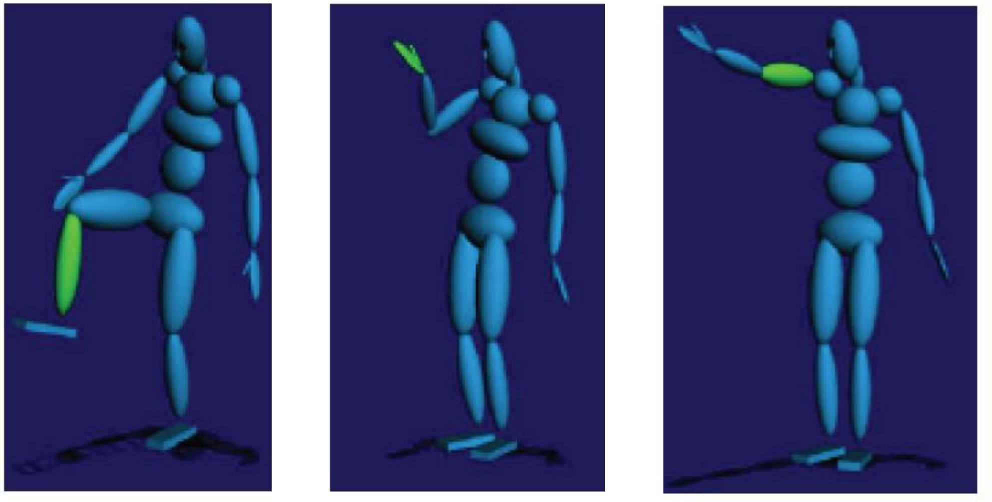
J3
Offset: (4,0)
Orientation: Rot(30)
Limit: [-90,90]

Joint Hierarchies

Joints in Character Animation

- Joints are organized in a hierarchy
- The root is the position of the "base" of the skeleton
  - typically the backbone or pelvis
  - the root has all 6 DoF so it can be placed anywhere with any orientation
- Typically, other joints have only rotational DoFs
  - but in reality they have prismatic (translational) components as well

3 translational and
48 rotational DoFs



1 DoF: Knee 2 DoFs: Wrist 3 DoFs: Shoulder

Image from http://www.cc.gatech.edu/classes/AY2012/cs4496_spring/

State Vector

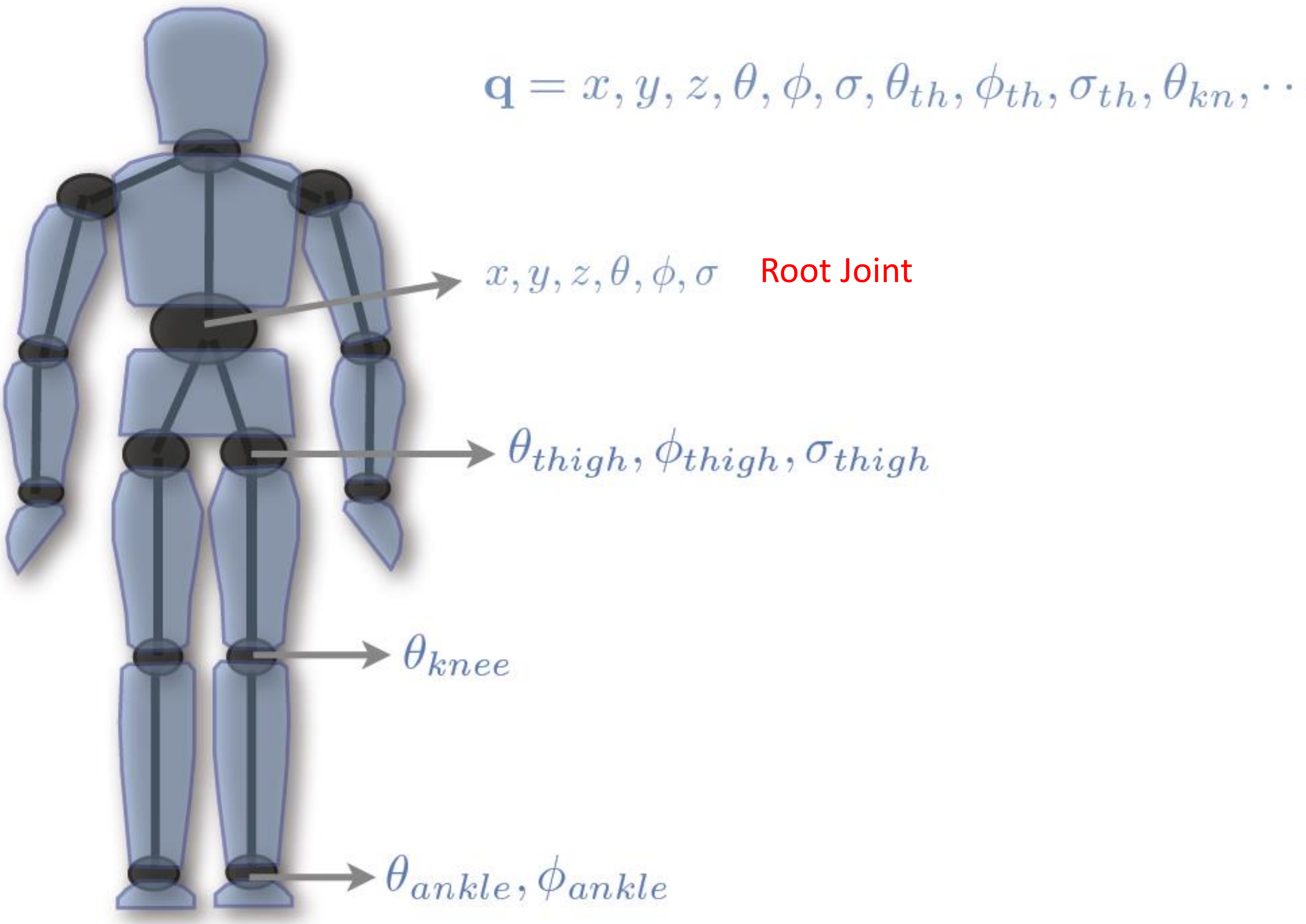
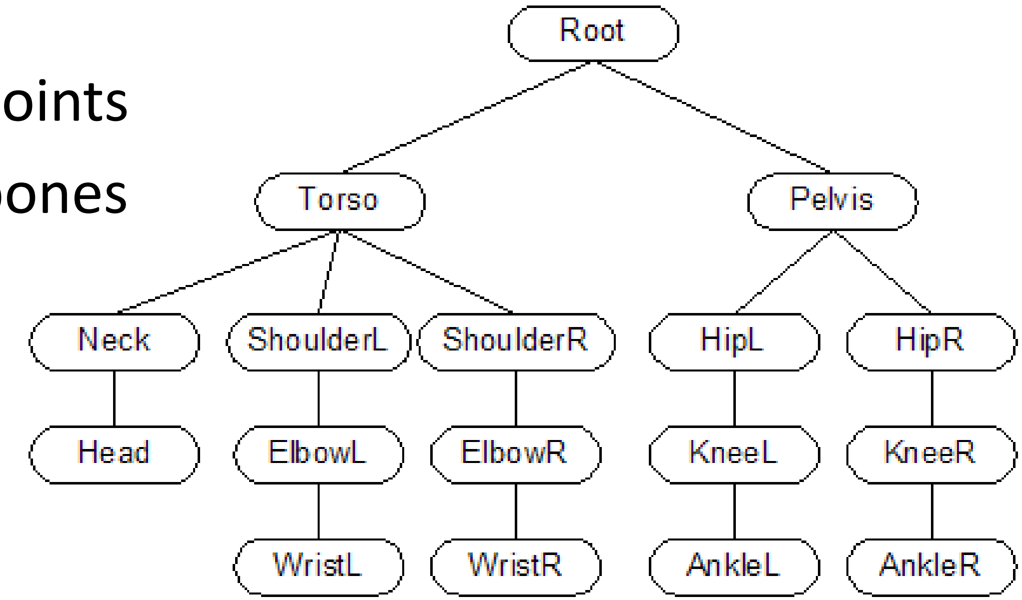
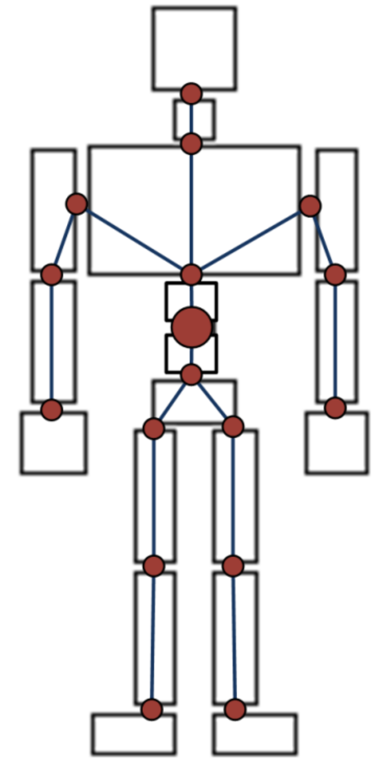


Image from http://www.cc.gatech.edu/classes/AY2012/cs4496_spring/

Hierarchical Representation

- The articulated skeleton can be described by a tree
  - Nodes: joints
  - Edges: bones



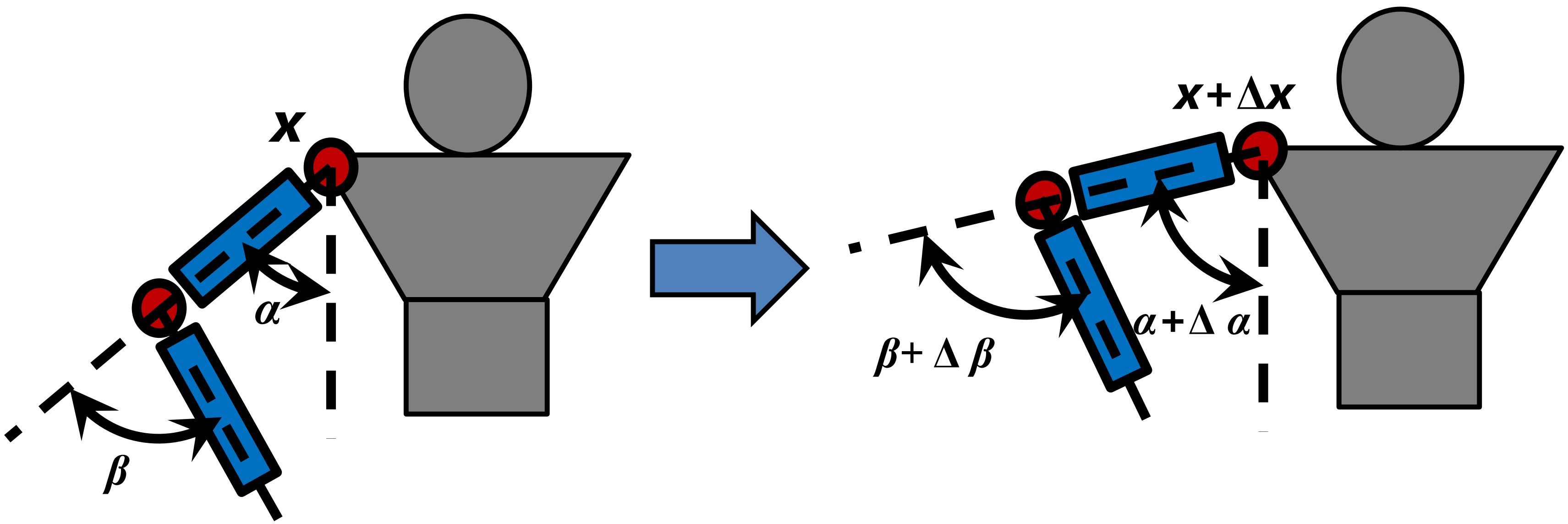


- The transformation of a joint is defined <u>relative</u> to its parent joint/body in the hierarchy

Forward Kinematics

Forward Kinematics

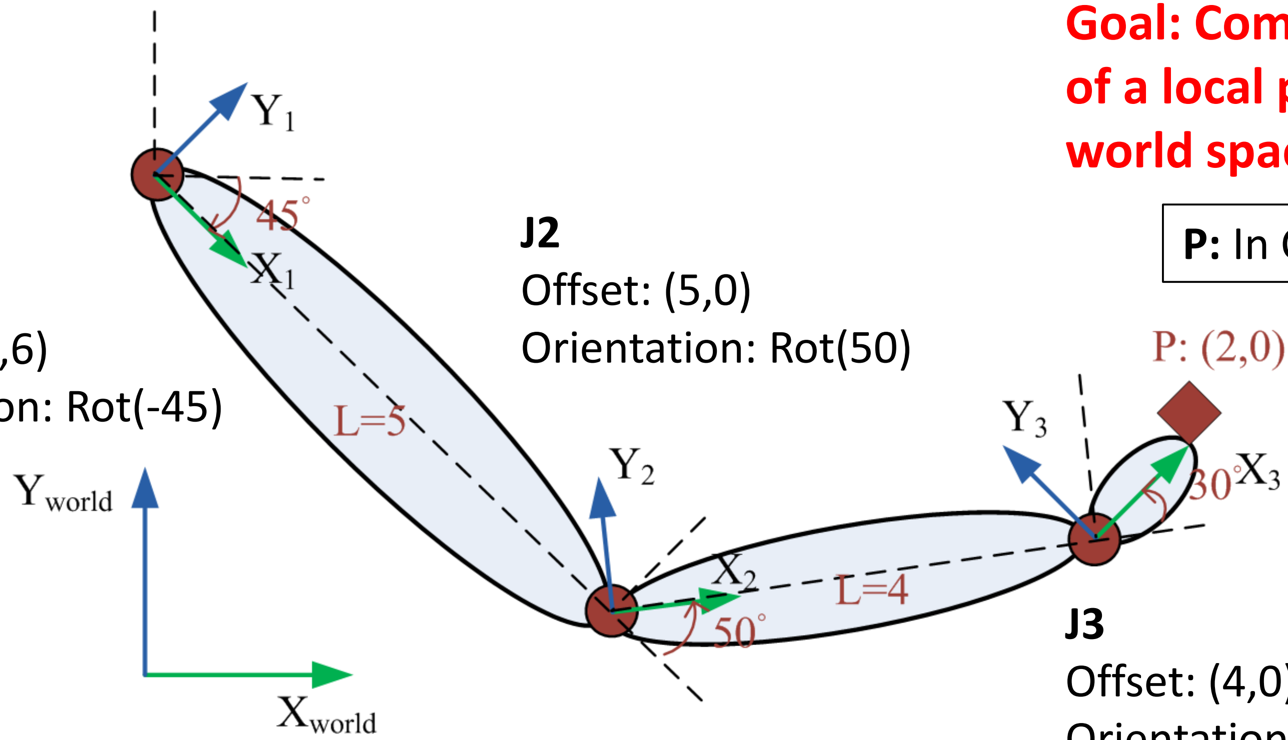
- Specify the base position/joint along with the other joint angles to prescribe motion
- To compute the final position and orientation of a joint in the world coordinate system, the transformations of all parent joints are combined together
  - Changing a parent joint affects *all* of its child bodies/joints
  - Changing a child joint does not affect any of its parent bodies/joints



Forward Kinematics

Goal: Compute the position of a local point P in the world space

P: In C.S. $\{X_3,Y_3\}$: (2,0)



J1
Offset: (0,6)
Orientation: Rot(-45)

J2
Offset: (5,0)
Orientation: Rot(50)

J3
Offset: (4,0)
Orientation: Rot(30)

In C.S. $\{x_{world},y_{world}\}$

In C.S. $\{x_1,y_1\}$

In C.S. $\{x_2,y_2\}$

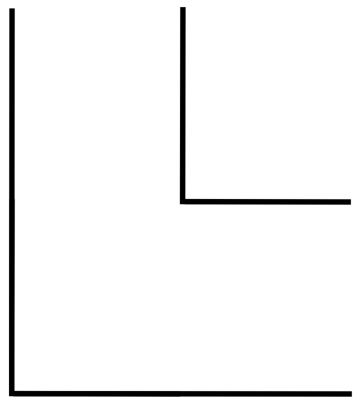
In C.S. $\{x_3,y_3\}$

$$P_{world}=T(0,6)R(-45)T(5,0)R(50)T(4,0)R(30)P_{local}$$

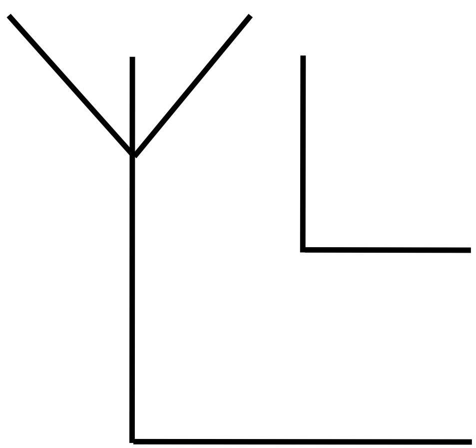
Recall: Transformations in World & Object Space

<u>World Space</u>: Read right to left

T(1,1)

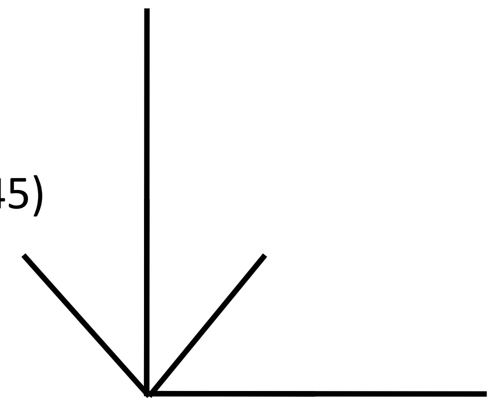


R(45) T(1,1)

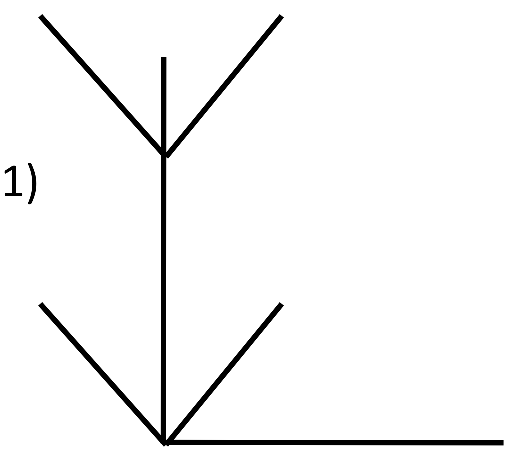


<u>Object Space</u>: Read left to right

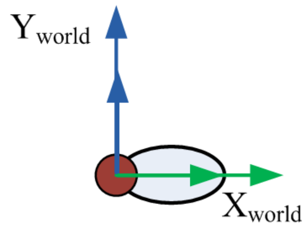
R(45)



R(45) T(1,1)

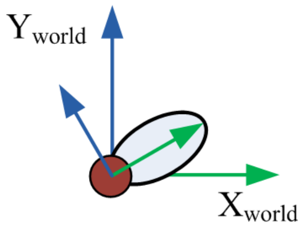


In the View of World Space



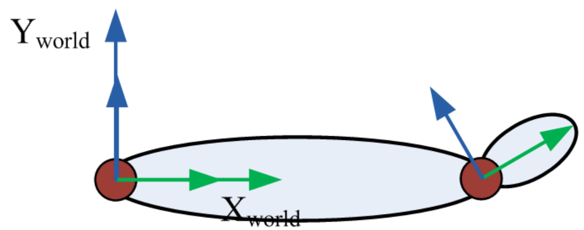
$$P_{world}=T(0,6)R(-45)T(5,0)R(50)T(4,0)R(30)P_{local}$$

In the View of World Space



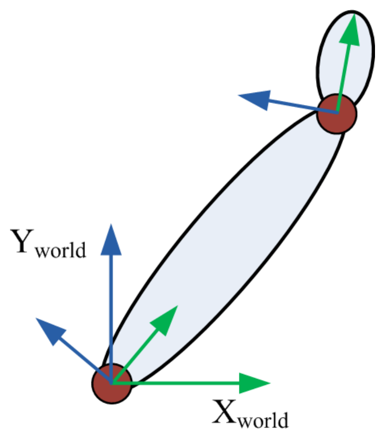
$$P_{world}=T(0,6)R(-45)T(5,0)R(50)T(4,0)\boxed{R(30)}P_{local}$$

In the View of World Space



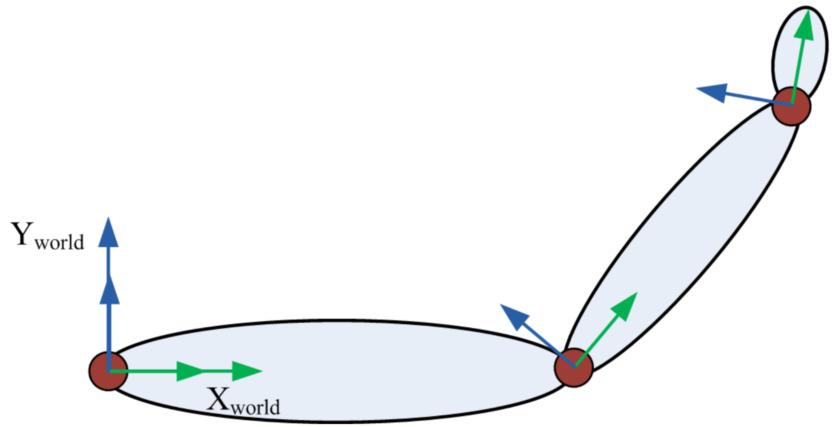
$$P_{world}=T(0,6)R(-45)T(5,0)R(50)\boxed{T(4,0)R(30)}P_{local}$$

In the View of World Space



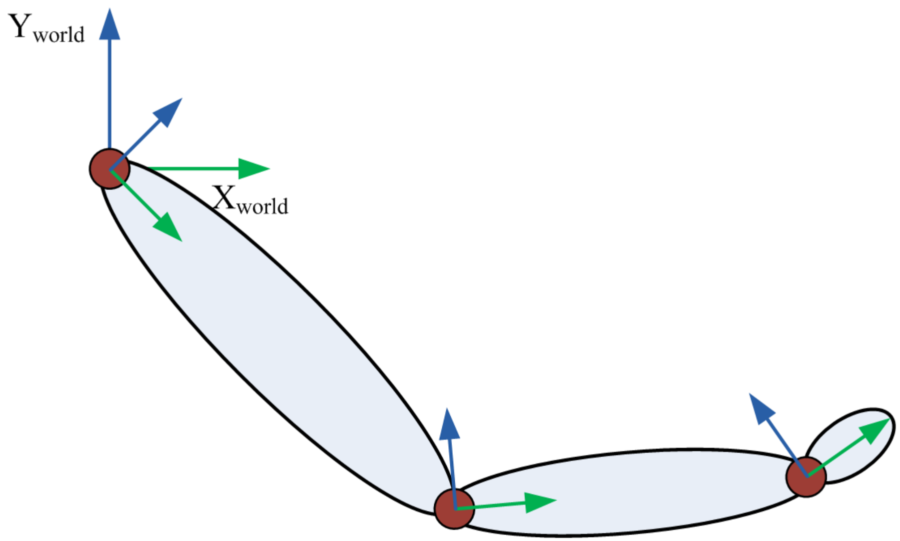
$$P_{world}=T(0,6)R(-45)T(5,0)\boxed{R(50)T(4,0)R(30)}P_{local}$$

In the View of World Space



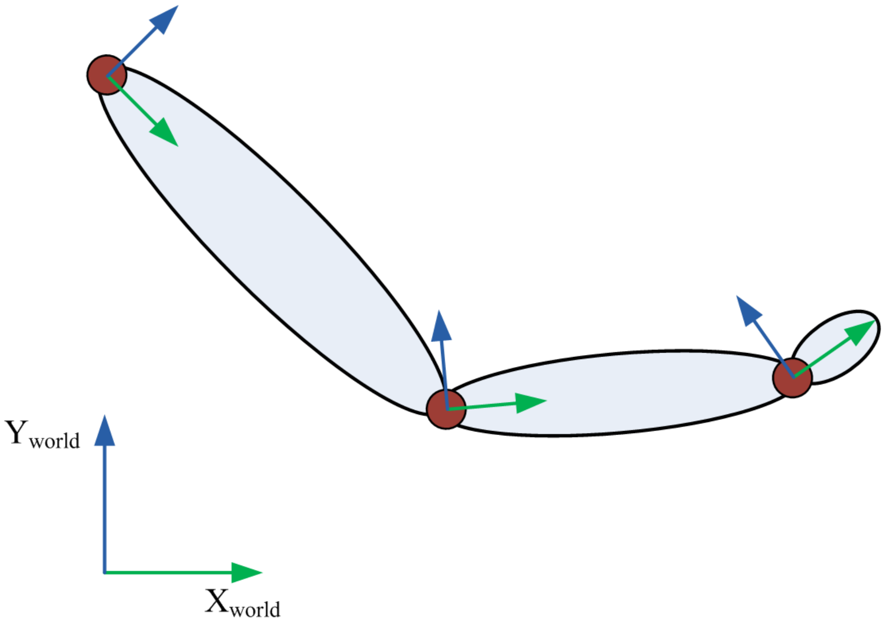
$$P_{world}=T(0,6)R(-45)\boxed{T(5,0)R(50)T(4,0)R(30)}P_{local}$$

In the View of World Space



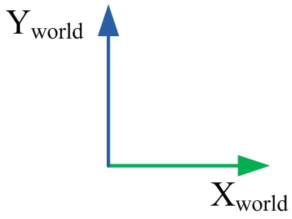
P_{world}=T(0,6)R(-45)T(5,0)R(50)T(4,0)R(30)P_{local}

In the View of World Space



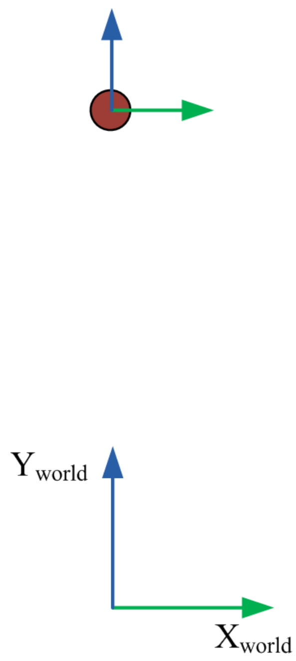
$$P_{world}=T(0,6)R(-45)T(5,0)R(50)T(4,0)R(30)P_{local}$$

In the View of Object Space



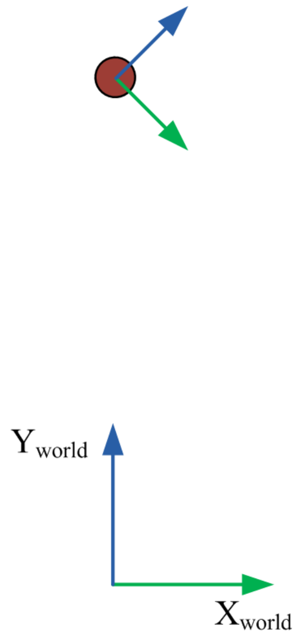
$$P_{world}=T(0,6)R(-45)T(5,0)R(50)T(4,0)R(30)P_{local}$$

In the View of Object Space



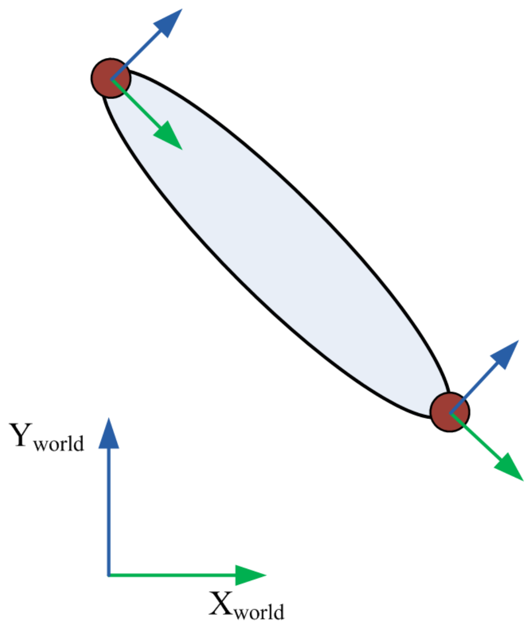
$$P_{world} = T(0,6)R(-45)T(5,0)R(50)T(4,0)R(30)P_{local}$$

In the View of Object Space



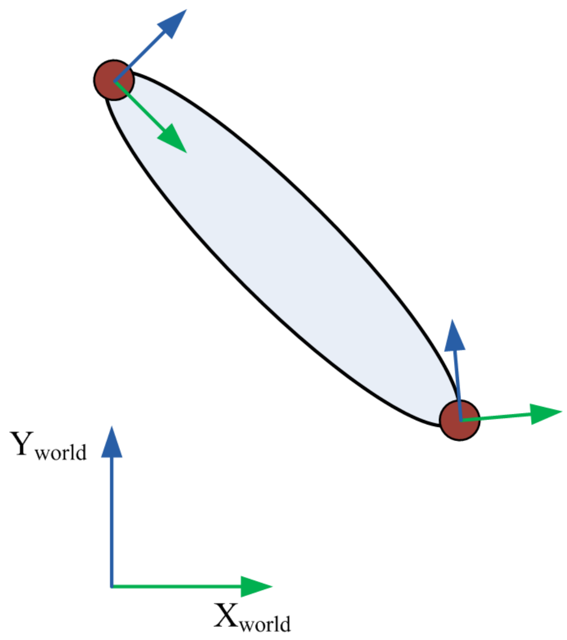
$$P_{world}=T(0,6)R(-45)T(5,0)R(50)T(4,0)R(30)P_{local}$$

In the View of Object Space



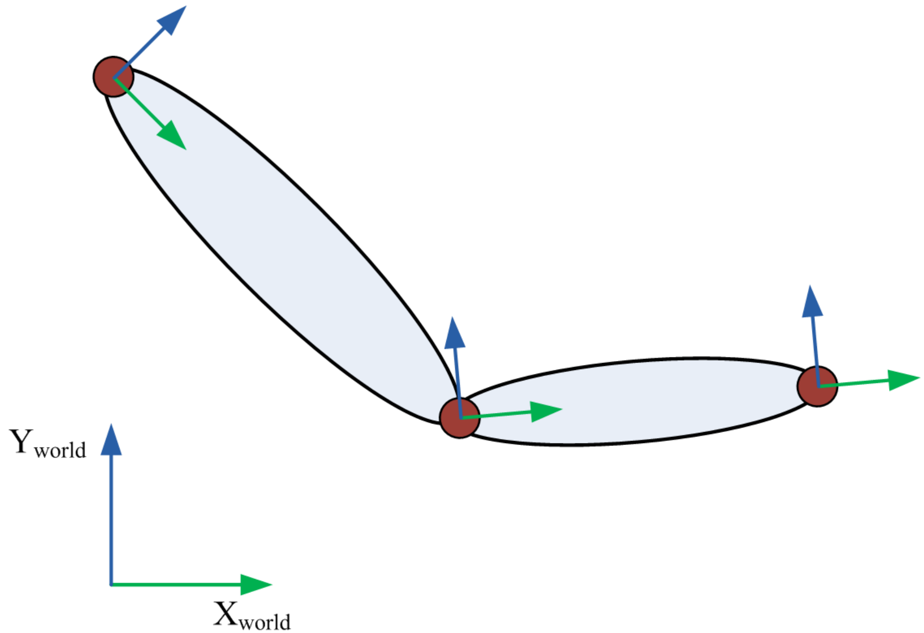
$$P_{world} = \boxed{T(0,6)R(-45)T(5,0)}R(50)T(4,0)R(30)P_{local}$$

In the View of Object Space



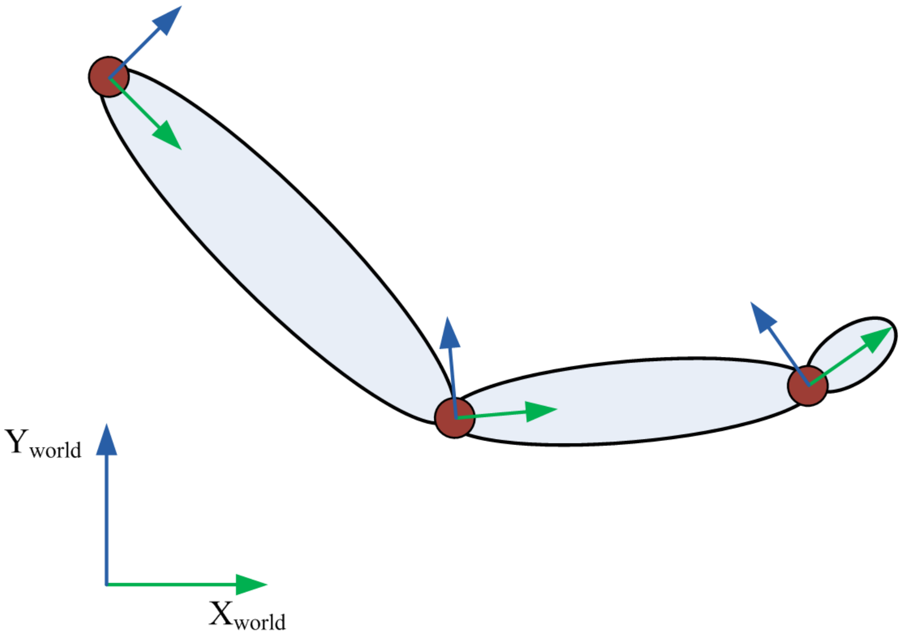
$$P_{world} = \boxed{T(0,6)R(-45)T(5,0)R(50)}T(4,0)R(30)P_{local}$$

In the View of Object Space



P_{world}=T(0,6)R(-45)T(5,0)R(50)T(4,0)R(30)P_{local}

In the View of Object Space



$$P_{world}=T(0,6)R(-45)T(5,0)R(50)T(4,0)R(30)P_{local}$$

Pros and Cons

- The hierarchy is simple and efficient:
  - Specify the motion of the root node first
  - Then specify the poses of children one layer at a time
- Especially suitable for modeling motions in open space (without constraints)
  - E.g. flying birds, swimming fish…
- Has difficulties with interactions with the environment
  - Making a foot stay on the ground, fingers stay in contact with a cup, etc.

Forward Kinematics Equations

- Given values for the joint DoF

$$\boldsymbol{\theta}=[\theta_1, \theta_2, \ldots, \theta_n]^T$$

- Compute the end effectors in world space

$$\mathbf{e}=[\mathbf{e_1}, \mathbf{e_2}, \ldots, \mathbf{e_m}]^T$$

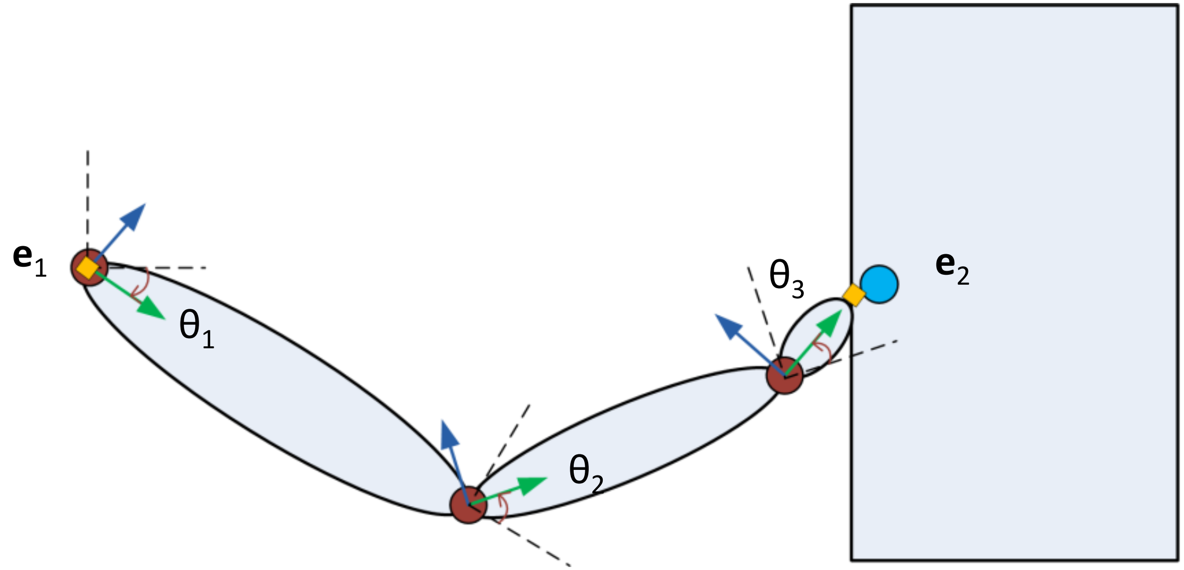
- Forward Kinematics defines and uses the function

$$\mathbf{e}=F(\boldsymbol{\theta})$$

Given $\boldsymbol{\theta}=[\theta_1, \theta_2, \theta_3]^T$

Find $\mathbf{e}=[\mathbf{e_1}, \mathbf{e_2}]^T$

$(\mathbf{e}_1,\mathbf{e}_2) = F(\theta_1, \theta_2, \theta_3)$



Inverse Kinematics

Inverse Kinematics Equations

- Given the values for the end effectors in world space

$$\mathbf{e}=[\mathbf{e_1}, \mathbf{e_2}, \ldots, \mathbf{e_m}]^T$$

- Compute the joint angles

$$\boldsymbol{\theta}=[\theta_1, \theta_2, \ldots, \theta_n]^T$$

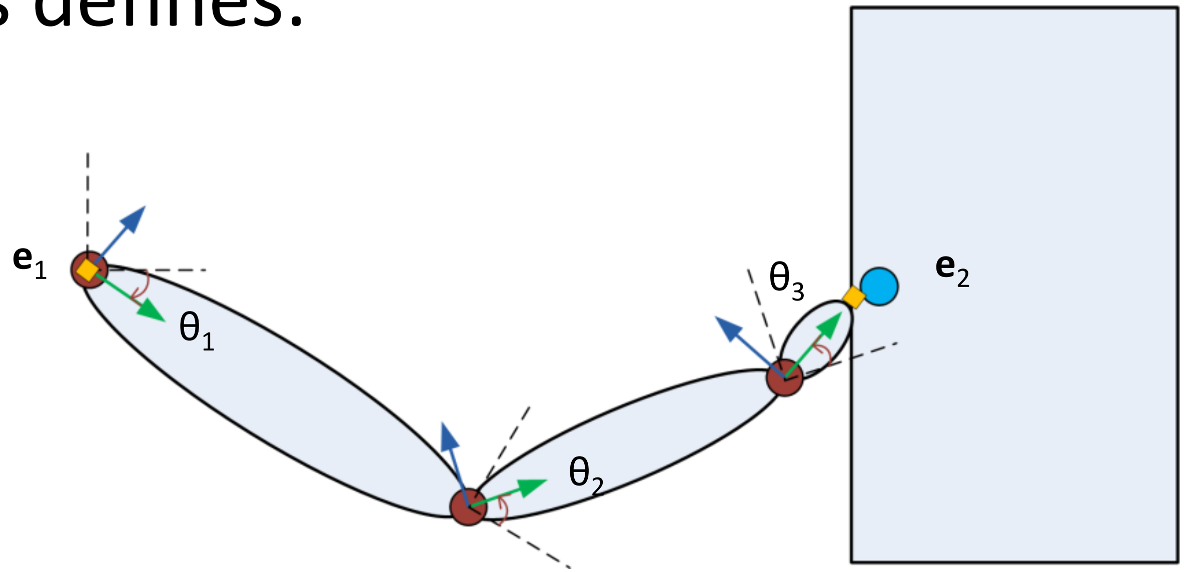
- Inverse Kinematics defines:

$$\boldsymbol{\theta} = G(\mathbf{e})$$

Given $\mathbf{e}=[\mathbf{e_1}, \mathbf{e_2}]^T$

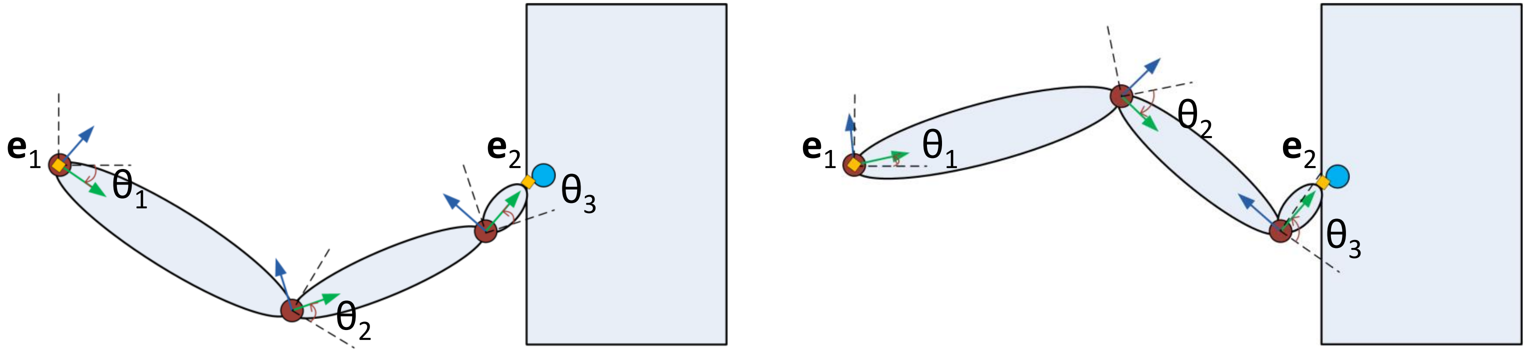
Find $\boldsymbol{\theta}=[\theta_1, \theta_2, \theta_3]^T$

$(\theta_1, \theta_2, \theta_3) = G(\mathbf{e}_1,\mathbf{e}_2)$



Inverse Kinematics

- Finding a solution for IK can be hard
  - one unique solution
  - infinite solutions (underconstrained)
  - no solution (overconstrained)
- Cannot be solved analytically in most cases
- Usually requires numerical methods
  - Jacobian iterative method
  - Optimization based methods



Two solutions $[\theta_1, \theta_2, \theta_3]^T$ for constrained $[\mathbf{e_1}, \mathbf{e_2}]^T$

Linearization

- Use the secant approximation on $\mathbf{e}=F(\boldsymbol{\theta})$:

$$\mathbf{e}-\mathbf{e}_0 = dF/d\boldsymbol{\theta}\,(\boldsymbol{\theta}-\boldsymbol{\theta}_0)$$

- Here $\mathbf{J}=dF/d\boldsymbol{\theta}$ is the Jacobian matrix of partial derivatives
- $\mathbf{J}$ defines the instantaneous changes in the end effectors $\mathbf{e}$ relative to infinitesimal changes in the angles $\boldsymbol{\theta}$
- since $\mathbf{e}=F(\boldsymbol{\theta})$ is nonlinear, $\mathbf{J}$ is only valid as an approximation near the current configuration $\boldsymbol{\theta}_0$

- Algorithm: replace $\mathbf{e}$ with $\mathbf{e}_{target}$ and solve for $\boldsymbol{\theta}$

Jacobian Matrix

- For $\mathbf{e}=[\mathbf{e_1}, \mathbf{e_2}, \ldots, \mathbf{e_m}]^T$ and $\boldsymbol{\theta}=[\theta_1, \theta_2, \ldots, \theta_n]^T$,

$$J = \begin{bmatrix} \frac{\partial F_1}{\partial \theta_1} & \frac{\partial F_1}{\partial \theta_2} & \cdots & \frac{\partial F_1}{\partial \theta_n} \\ \frac{\partial F_2}{\partial \theta_1} & \frac{\partial F_2}{\partial \theta_2} & \cdots & \frac{\partial F_2}{\partial \theta_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial F_m}{\partial \theta_1} & \frac{\partial F_m}{\partial \theta_2} & \cdots & \frac{\partial F_m}{\partial \theta_n} \end{bmatrix}$$

- The column for the *j*th joint can be computed numerically as:

$$\mathbf{J}_j = \frac{F(\boldsymbol{\theta} + \mathbf{b}_j \delta\theta_j) - F(\boldsymbol{\theta})}{\delta\theta_j} \quad \text{where} \quad \mathbf{b}_j = [0,0,\ldots,\overset{jth}{1},\ldots,0,0]^T$$

- If the *j*th joint is a rotational joint with a single degree of freedom then the element corresponding to the *i*th end effector in the *j*th column of **J** can be computed analytically as:

$$\mathbf{J}_{ij} = \mathbf{v}_j \times (\mathbf{s}_i - \mathbf{p}_j)$$

where $\mathbf{v}_j$ is the unit vector pointing along the axis of rotation, $\mathbf{p}_j$ is the position of the joint, and $\mathbf{s}_i$ is the position of the *i*th end effector

Iterative Solver

Input: $\mathbf{e_0}$ – current end effector positions

$\mathbf{e_t}$ – target end effector positions

$\boldsymbol{\theta_0}$ – current DoFs

<u>Algorithm</u>:

while($|\mathbf{e_t} - \mathbf{e_0}|$ > threshold){

compute **J**; // Take many small steps and recompute **J** at each step

$\delta\mathbf{e} = \mathbf{e_t} - \mathbf{e_0}$; // Can scale this down to aim for smaller steps

Solve $\mathbf{J}\,\delta\boldsymbol{\theta} = \delta\mathbf{e}$ to find $\delta\boldsymbol{\theta}$;

update the DoF with a small step of $\alpha\delta\boldsymbol{\theta}$: i.e., $\boldsymbol{\theta_0} \mathrel{+}= \alpha\delta\boldsymbol{\theta}$;

update end effectors: $\mathbf{e_0} = F(\boldsymbol{\theta_0})$;

}

Solving $\mathbf{J}\,\delta\boldsymbol{\theta} = \delta\mathbf{e}$

- $\mathbf{J}$ is not guaranteed to be invertible, and is typically not even a square matrix
- If $\mathbf{J}$ is overdetermined, multiply both sides by $\mathbf{J}^T$, solve $\mathbf{J}^T\mathbf{J}\,\delta\boldsymbol{\theta} = \mathbf{J}^T\delta\mathbf{e}$ for the least-squares solution (Householder is better!)
  - this gives the unique solution too, if it exists (of course, there are better methods if the solution is unique)
- If $\mathbf{J}$ is underdetermined, use pseudo inverse $\mathbf{J}^+$ to obtain $\delta\boldsymbol{\theta} = \mathbf{J}^+\delta\mathbf{e}$ for the minimum norm solution
  - The general solution of $\delta\boldsymbol{\theta} = \mathbf{J}^+\delta\mathbf{e}$ can be written as

    $\delta\boldsymbol{\theta} = \mathbf{J}^+\delta\mathbf{e} + (\mathbf{I} - \mathbf{J}^+\mathbf{J})\boldsymbol{\gamma}$
  - The second term $(\mathbf{I} - \mathbf{J}^+\mathbf{J})$ represents the orthogonal projection to the null space of $\mathbf{J}$
  - For any $\boldsymbol{\beta} = (\mathbf{I} - \mathbf{J}^+\mathbf{J})\boldsymbol{\gamma}$ we have $\mathbf{J}\,\boldsymbol{\beta} = 0$, which means $\boldsymbol{\beta}$ will only affect the interior joints and causes no motion of the end effectors
  - This null space term can be used for some secondary goals, e.g. finding the most natural positions for the joints, balance, etc.
    - think about all the “styles” of walking
- Take CS205!

Pros and Cons

- Modeling poses of characters interacting with other objects
  - E.g., make sure fingers are exactly in contact with a cup
- Cannot get a solution if the target position is impossible (over-constrained)
  - Try to find a solution as close as possible in some sense, but this could look bad
- Hard to pick the best solution for an under-constrained system (null space matters!)
  - Requires additional constraints or optimizing some quantities

Question #1

LONG FORM:

- Summarize forward and inverse kinematics.
- Describe your ideas, so far, for your game.

SHORT FORM:

- Give a name to each game we discuss in class.

Puppeteering

Puppeteering

- Specifying animation curves and splines for every degree of freedom (root and all angles) by hand can be tedious
  - And hard to make look realistic
- Inverse kinematics can help
  - But still leaves the null space degrees of freedom unspecified
- It would be better if one could input motion in a higher level fashion
  - similar to how a puppeteer *guides* a puppet
- One solution to this is <u>motion capture</u>, where human movements are captured by various types of cameras

Motion Capture

- Attach a number of markers to a person
- Light is emitted from a number of cameras and reflected back to the cameras
- Compute the 3D location of the markers by inferring depth values
- From the marker locations, determine rigid body positions and joint angles
  - and thus the character motion

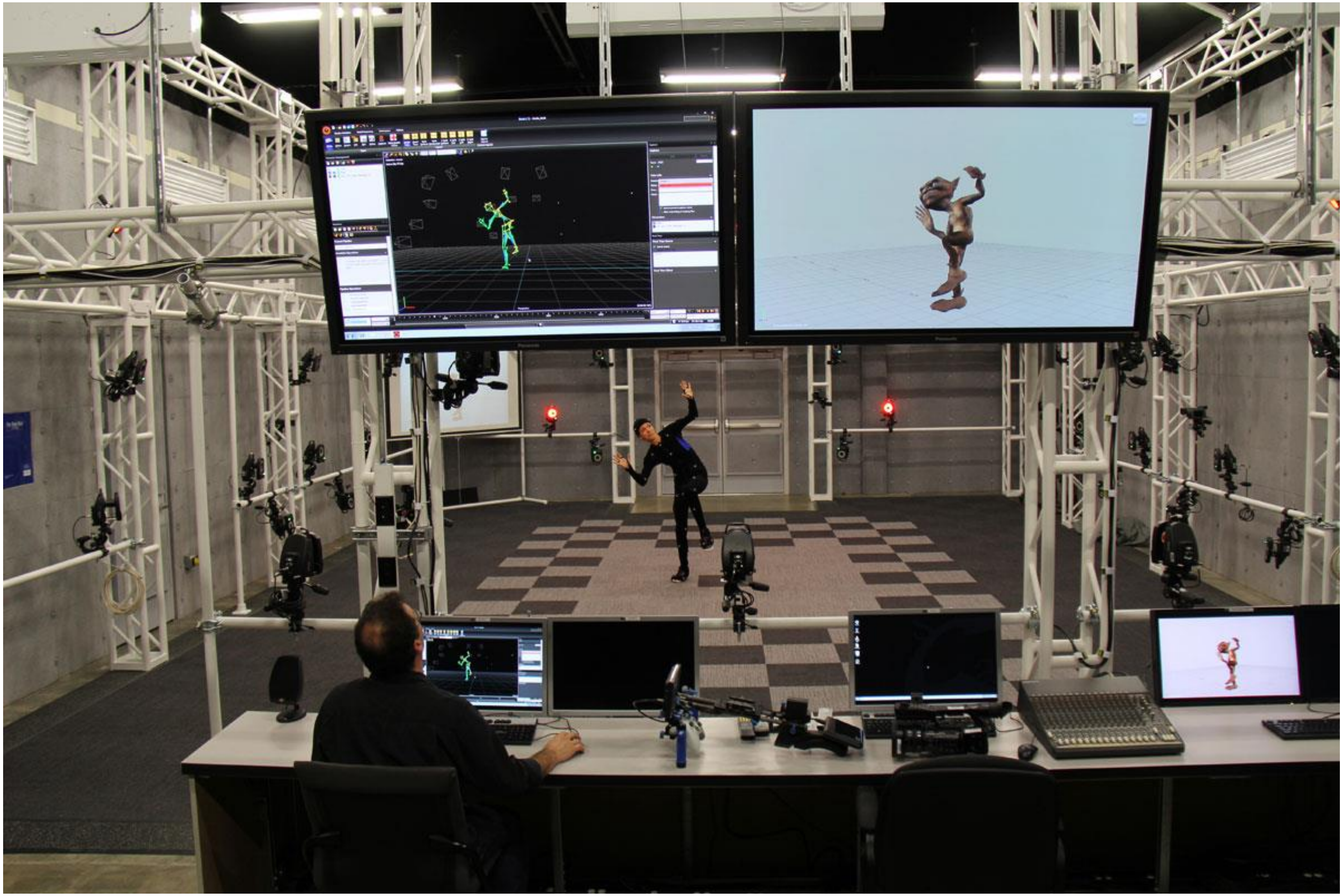
https://www.nbc.com/saturday-night-live/video/motion-capture/n13489

Cameras

- Multiple cameras are set up to capture the performance space…

Motion Capture Stage

- Stages can be quite extravagant…



Facial Motion Capture

- A special camera attached to the actors head is often used

Facial Motion Capture

- Extensive secondary effects, including simulation, can be incorporated on top of the motion capture