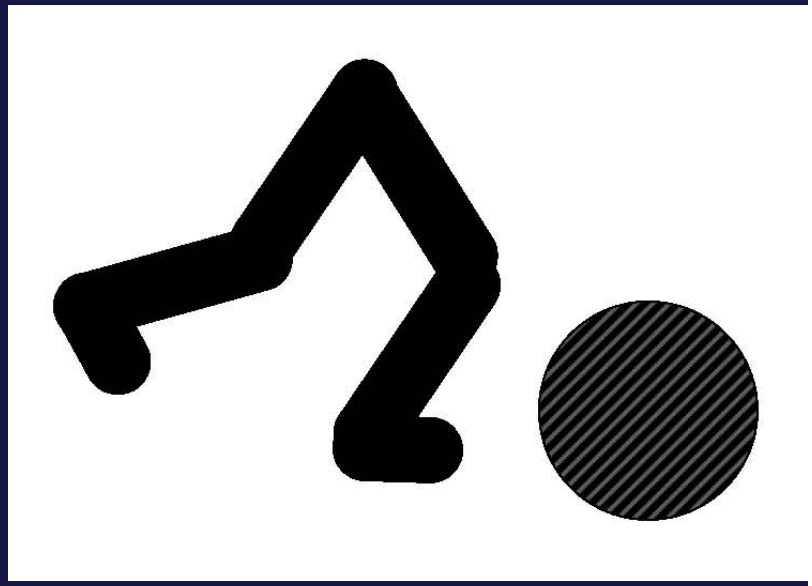
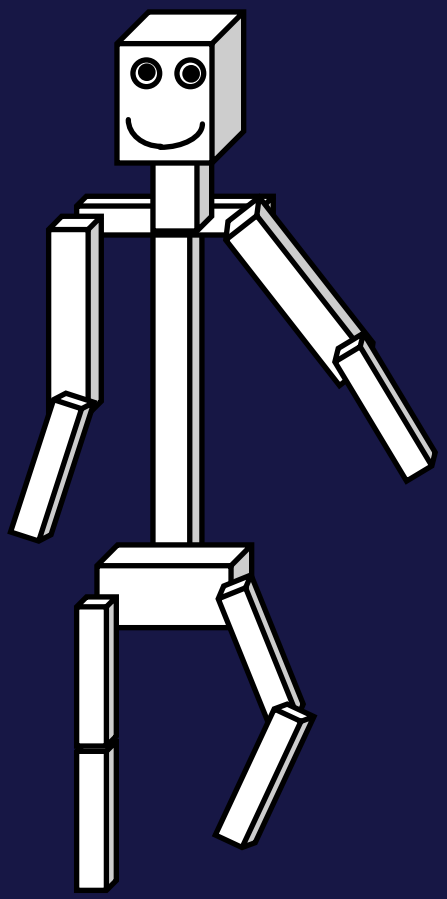
Kinematics and Orientations

- Hierarchies
- Forward Kinematics
- Transformations (review)
- Euler angles
- Quaternions
- Yaw and evaluation for assignment 2

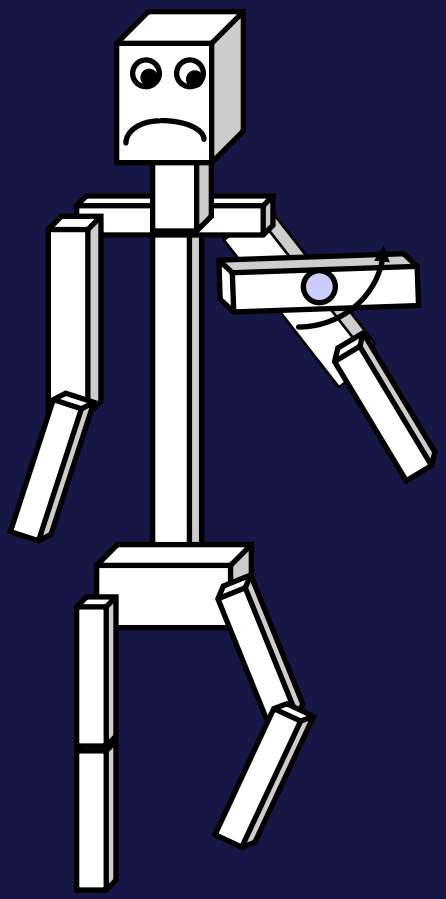


Building a character



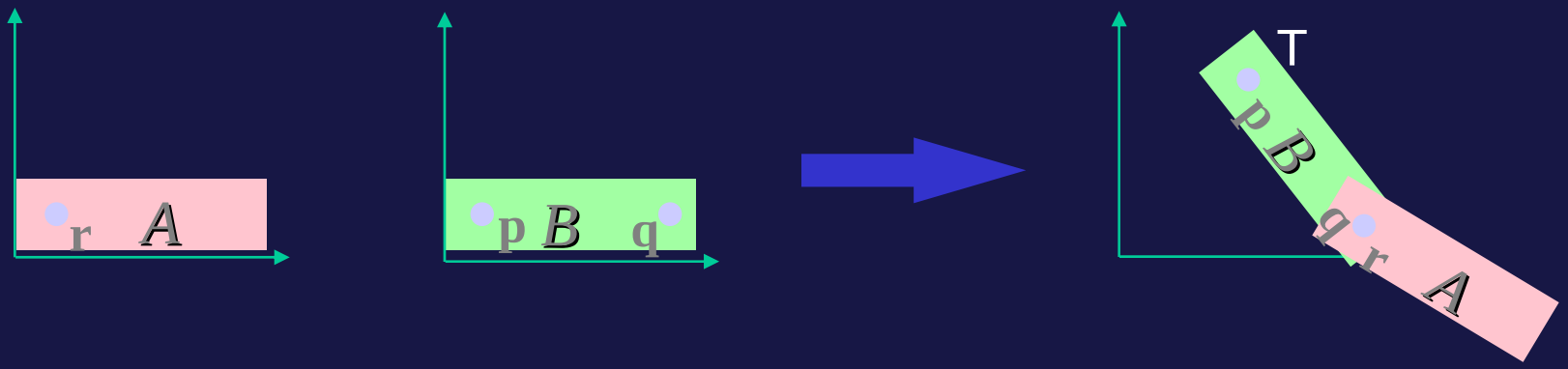
- Just translate, rotate, and scale each body part to get the right size, shape, position, and orientation.
- Looks great--until you try to make it move.

The Right Control Knobs



- As soon as you want to change something, the model falls apart
- Reason: the thing you're modeling is *constrained* but your model doesn't know it
- What we need:
  - some sort of representation of *structure*
  - a set of "control knobs" (parameters) that make it easy to move our stick person through *legal configurations*
- Key is to structure the transformations in the right way: using a hierarchy

Making an Articulated Model

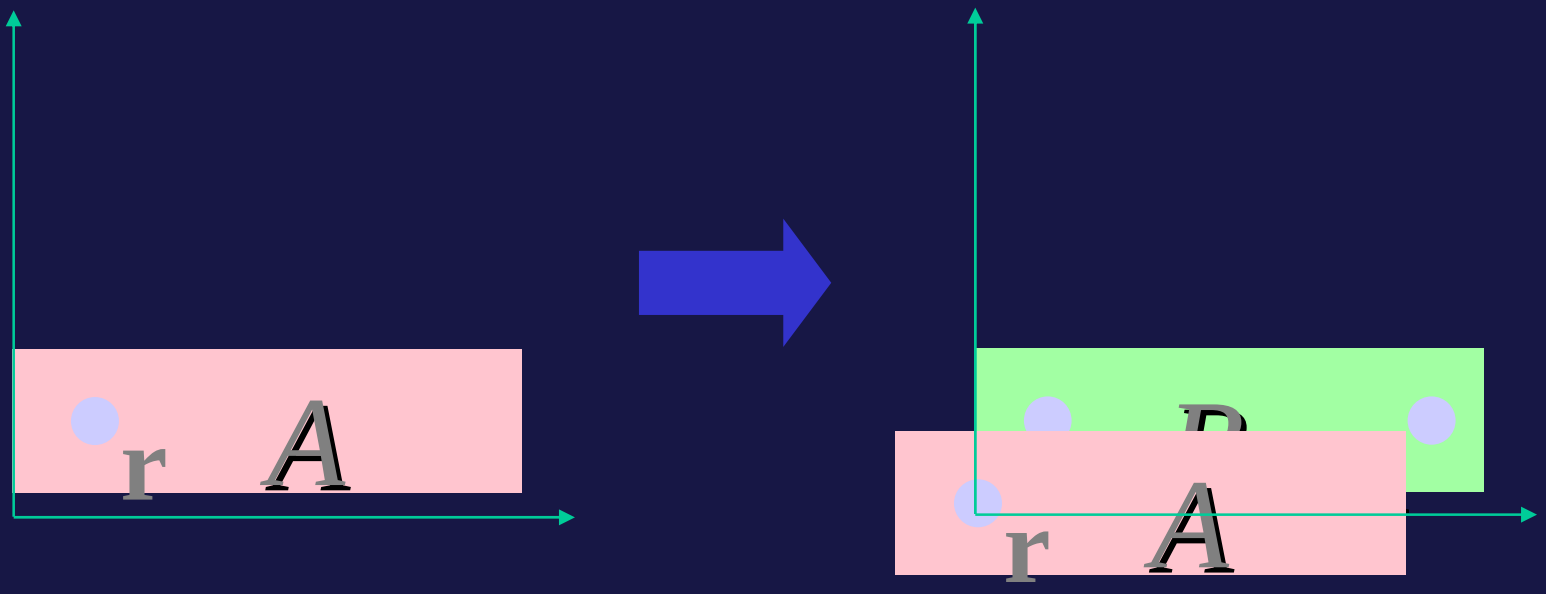


- A minimal 2-D jointed object:
  - Two pieces, *A* (“forearm”) and *B* (“upper arm”)
  - Attach point q on *B* to point r on *A* (“elbow”)
  - Desired control knobs:
    - T: shoulder position (point at which p winds up)
    - u: shoulder angle (*A* and *B* rotate together about p)
    - v: elbow angle (*A* rotates about r, which stays attached to q)

Making an Arm, step 1

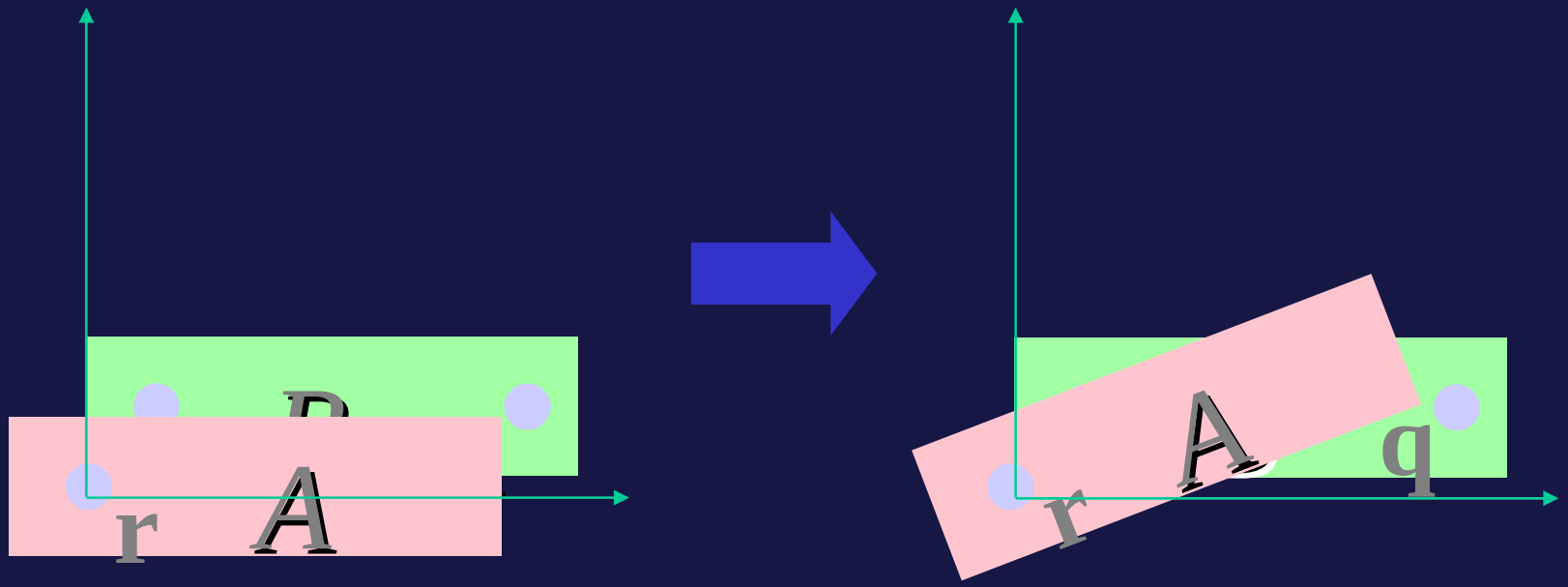
- Start with *A* and *B* in their untransformed configurations (*B* is hiding behind *A*)
- First apply a series of transformations to *A*, leaving *B* where it is…

Making an Arm, step 2



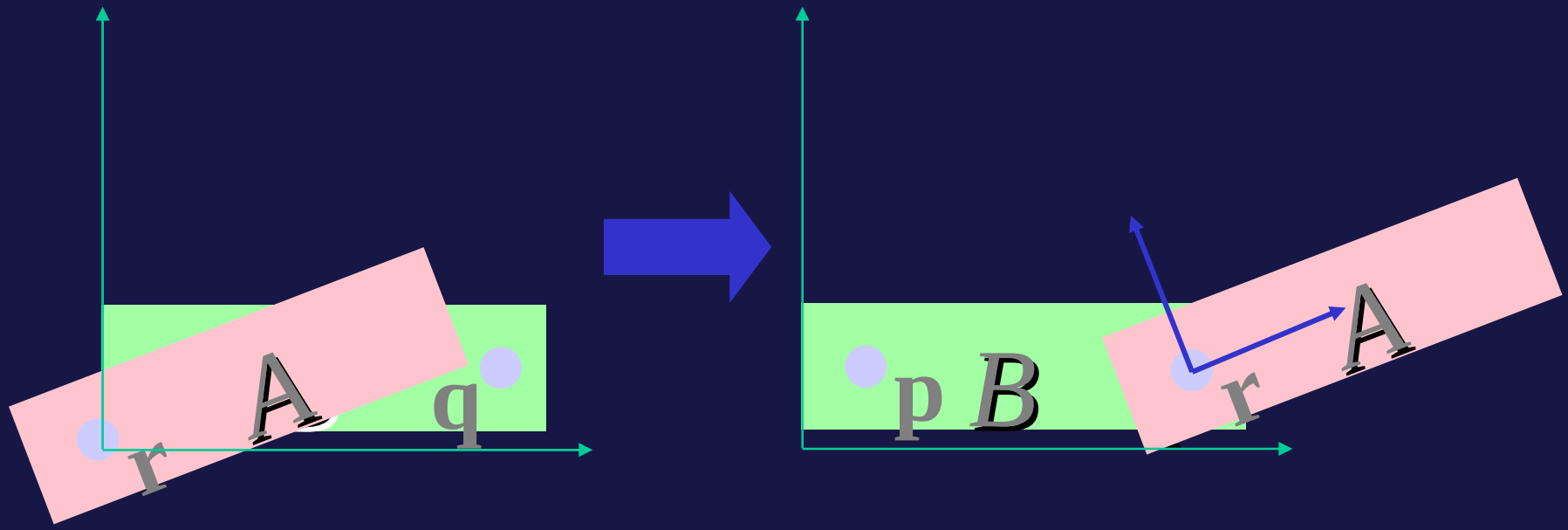
- Translate by -r, bringing r to the origin
- *B* is now peeking out from behind *A*

Making an Arm, step 3



Next, we rotate *A* by v (the “elbow” angle)

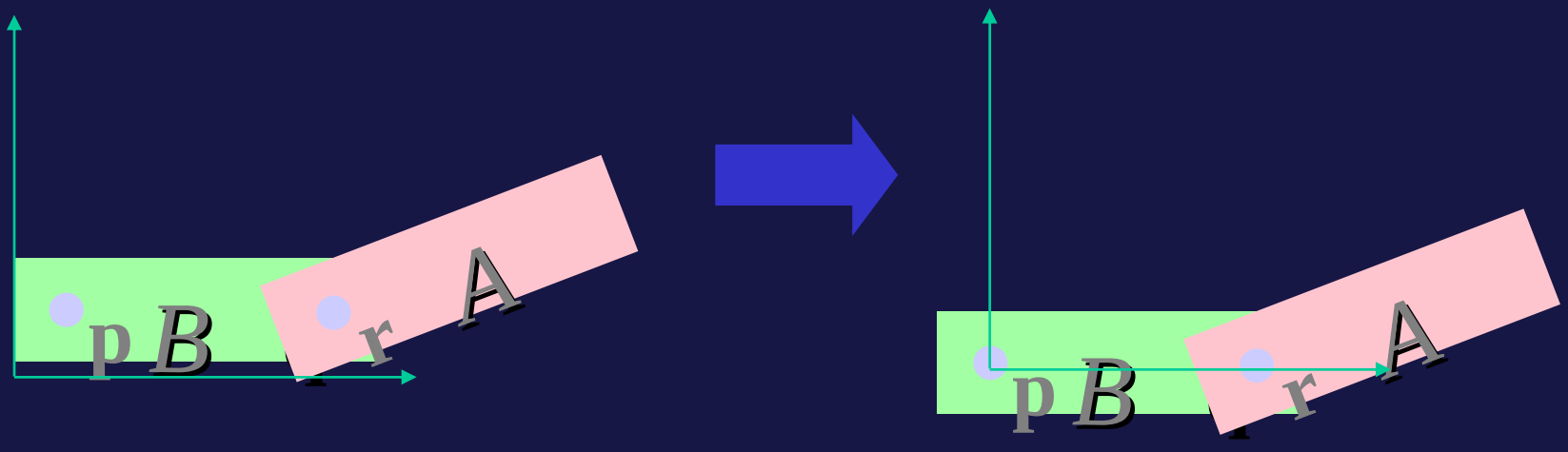
Making an Arm, step 4



Translate *A* by q, bringing r and q together to form the elbow joint

We can regard q as the origin of the *lower arm coordinate system*, and regard A as being in this coordinate system.

Making an Arm, step 5

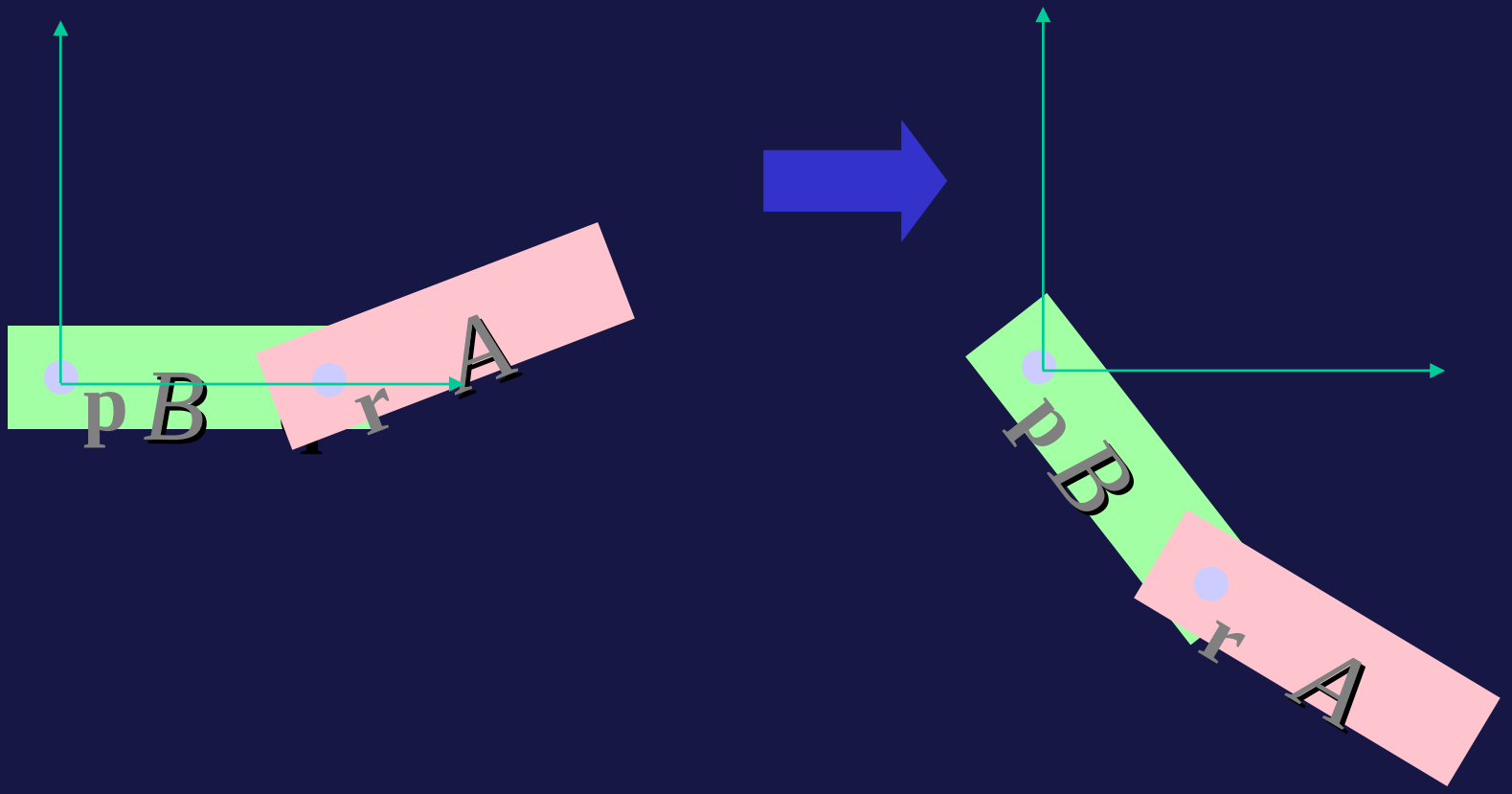


From now on, each transformation applies to *both* *A* and *B* (This is important!)

First, translate by -p, bringing p to the origin

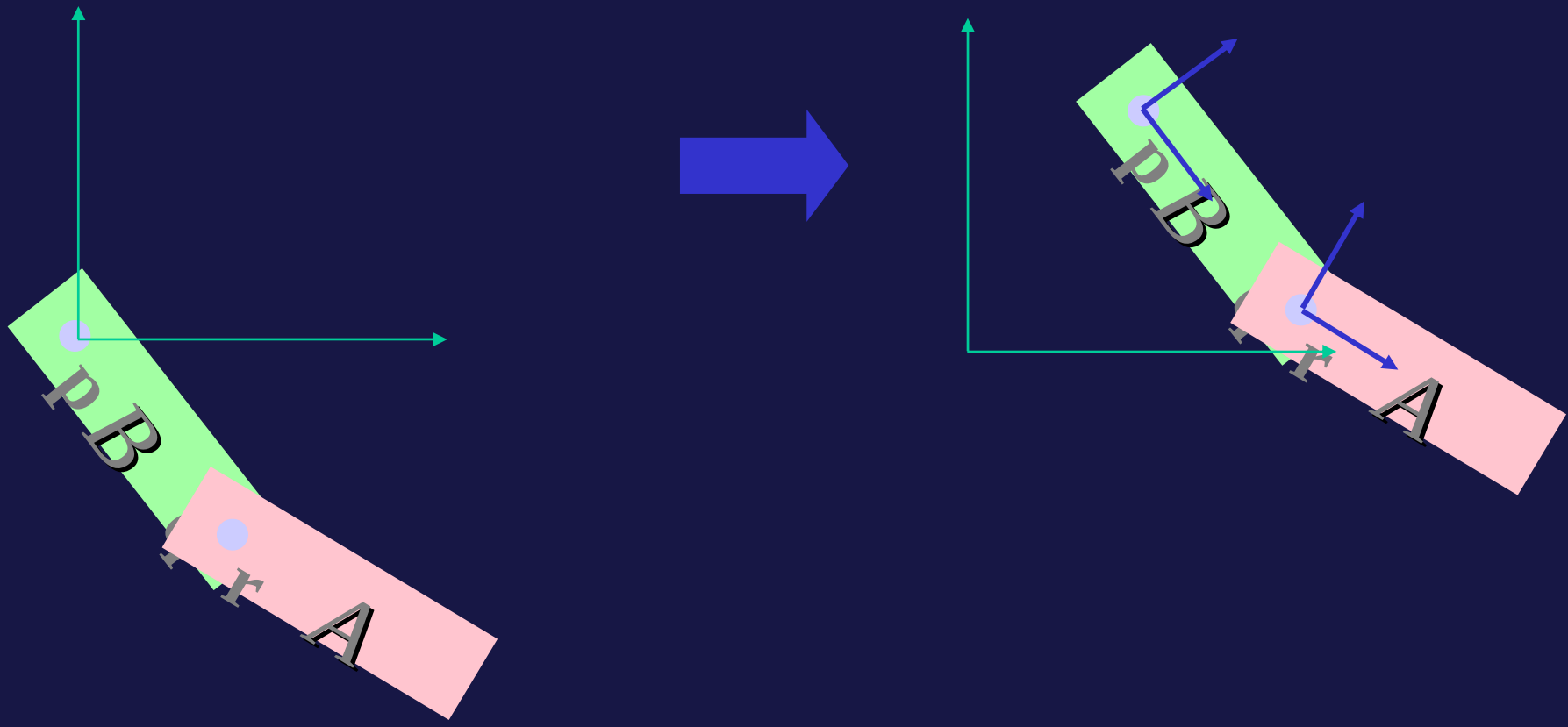
A and *B* both move together, so the elbow doesn't separate

Making an Arm, step 6



Then, we rotate by u, the “shoulder” angle
Again, *A* and *B* rotate together

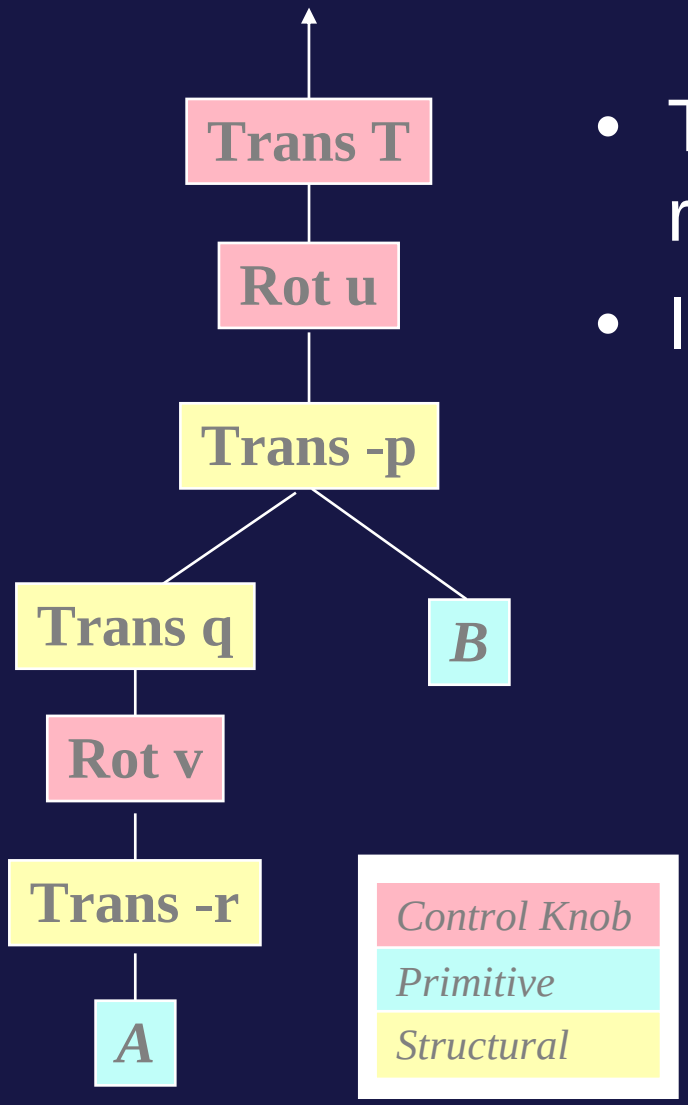
Making an Arm, step 7



Finally, translate by T, bringing the arm where we want it

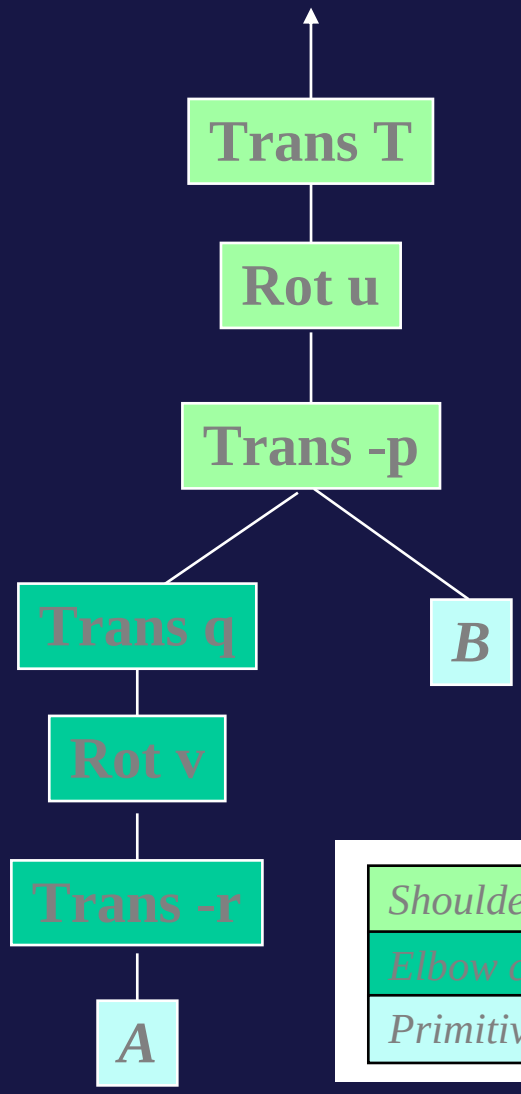
p is at origin of *upper arm coordinate system*

Transformation Hierarchy



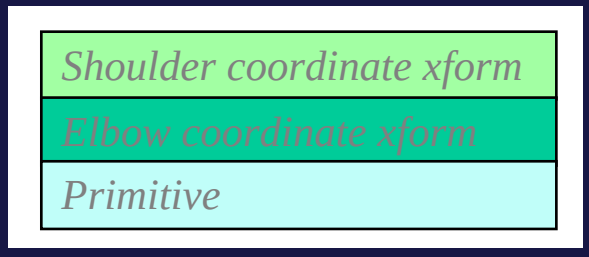
- The build-an-arm sequence, represented as a tree
- Interpretation:
  - Leaves are geometric primitives
  - Internal nodes are transformations
  - Transformations apply to everything under them

Transformation Hierarchy

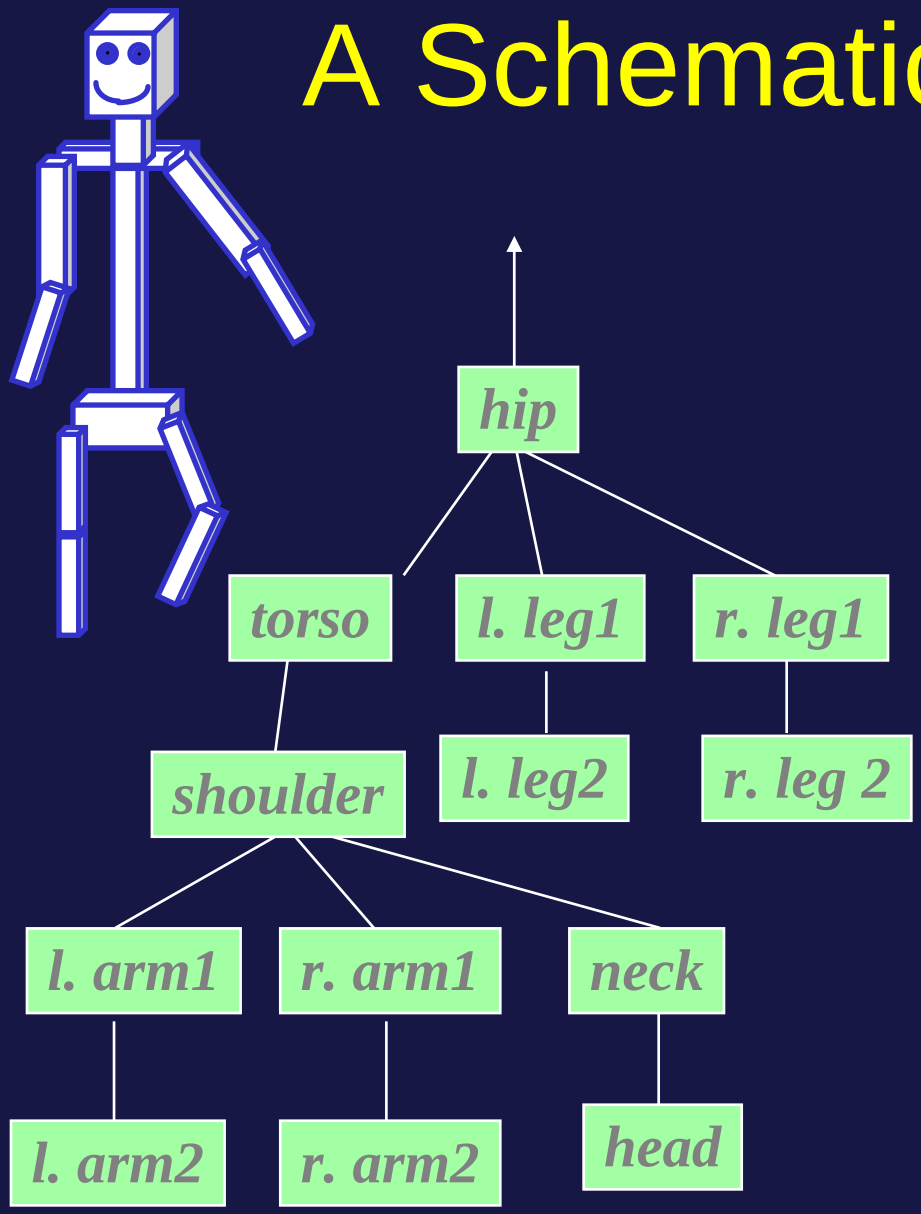


Another point of view:

- The shoulder coordinate transformation moves everything below it with respect to the shoulder:
  - B
  - A and its transformation
- The elbow coordinate transformation moves A with respect to the shoulder coordinate transform



A Schematic Humanoid



- Each node represents
  - rotation(s)
  - geometric primitive(s)
  - structural transformations
- The root can be anywhere. We chose the pelvis *(can re-root)*
- Control knob for each joint angle, plus global position and orientation for the root
- This is how amc file in 2nd assignment works

What Hierarchies Can and Can't Do

Advantages:

- Reasonable control knobs
- Maintains structural constraints

Disadvantages:

- Doesn't always give the "right" control knobs trivially
  - e.g. hand or foot position - re-rooting may help
- Can't do closed kinematic chains easily (keep hand on hip)
- Missing other constraints: do not walk through walls

So What Have We Done?

Forward Kinematics

Given the model and the joint angles, where is the end effector? Compute this so you know where to draw primitives

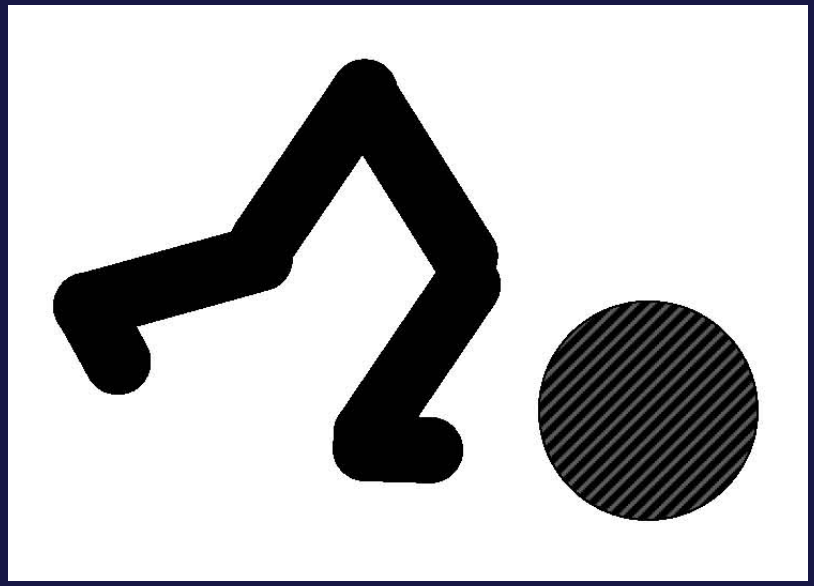
Inverse Kinematics

Given a desired location of the end effector, what are the required joint angles to put it there? Required to place the end effector near to an object in the real world.

Kinematics is easy, IK is hard because of redundancy.

Kinematics and Orientations

- Hierarchies
- Forward Kinematics
- Transformations (review)
- Euler angles
- Quaternions
- Washing out yaw for assignment 2



Transformations (Review)

- Translation, scaling, and rotation:

$$P' = T + P \qquad \text{Translation}$$

$$P' = SP \qquad \text{Scaling}$$

$$P' = RP \qquad \text{Rotation}$$

- treat all transformations the same so that they can be easily combined (streamline software and hardware)
- P is a point of the model
- Transformation is for animation, viewing, modeling
- P' is where it should be drawn

Translation

$$\begin{bmatrix} x+t_x \\ y+t_y \\ z+t_z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

new point in space

transformation matrix

point in space

Scaling

$$\begin{bmatrix} xS_x \\ yS_y \\ zS_z \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotation

In the upper left 3x3 submatrix

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

X axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Y axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Z axis

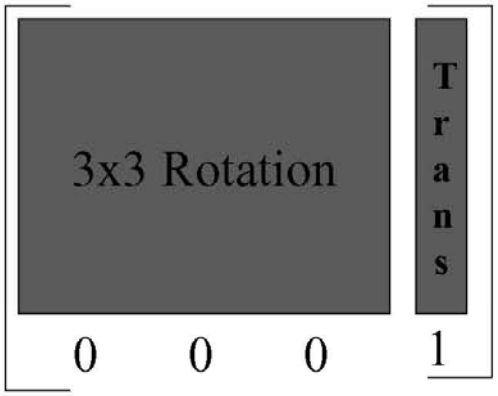
Composite Transformations

- We can now treat transformations as a series of matrix multiplications

$$P' = M_1 M_2 M_3 M_4 M_5 M_6 P$$

$$M = M_1 M_2 M_3 M_4 M_5 M_6$$

$$P' = MP$$



Interpolation

Trivial for translation: t=k * t1 + (1-k) * t2

Easy for rotation in 1D

Not so easy for 3D rotation

Interpolating Rotations

The upper left 3x3 submatrix of a transformation matrix is the rotation matrix

Maybe we can just interpolate the entries of that matrix to get the inbetween rotations?

Problem:

- Rows and columns are orthonormal (unit length and perpendicular to each other)
- Linear interpolation doesn't maintain this property, leading to nonsense for the inbetween rotations

Interpolating Rotation

Example:

- interpolate linearly from a positive 90 degree rotation about y to a negative 90 degree rotation about y

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Linearly interpolate each component and halfway between, you get this…

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

No longer a rotation matrix---not orthonormal
Makes no sense!

Orientation Representations

Direct interpolation of transformation matrices is not acceptable…

Where does that leave us?

How best do we represent orientations of an object and interpolate orientation to produce motion over time?

- Rotation Matrices
- Fixed Angle
- Euler Angle
- Axis Angle
- Quaternions

Fixed Angle Representation

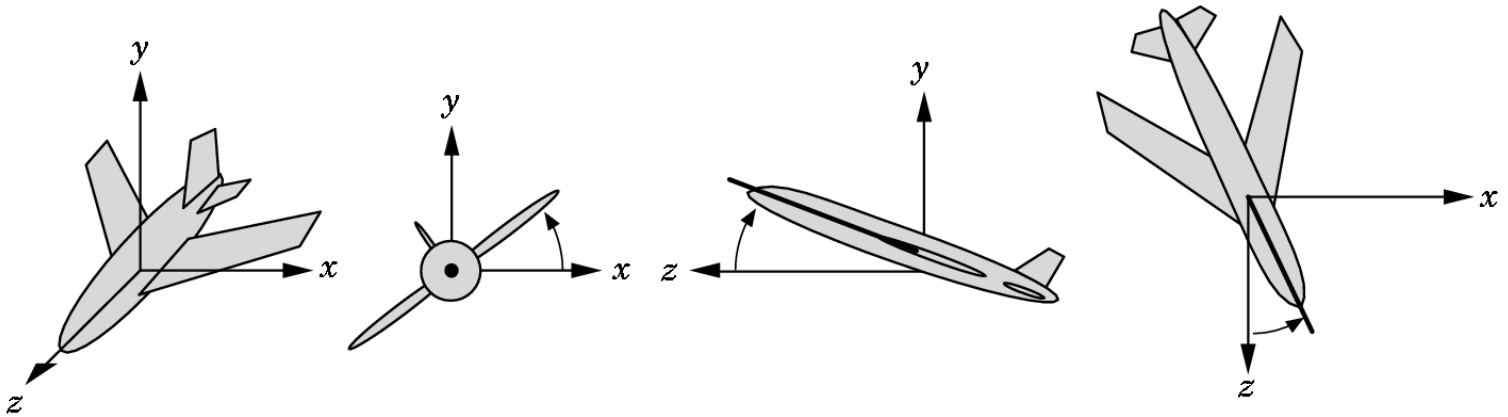
- Angles used to rotate about fixed axes
- Orientations are specified by a set of 3 ordered parameters that represent 3 ordered rotations about fixed axes, i.e. first about x, then y, then z
- Many possible orderings, don't have to use all 3 axes, but can't do the same axis back to back

Euler Angles

- Same as fixed axis, except now, the axes move with the object
- roll, pitch, yaw of an aircraft
- Euler Angle rotations about moving axes written in reverse order are the same as the fixed axis rotations.

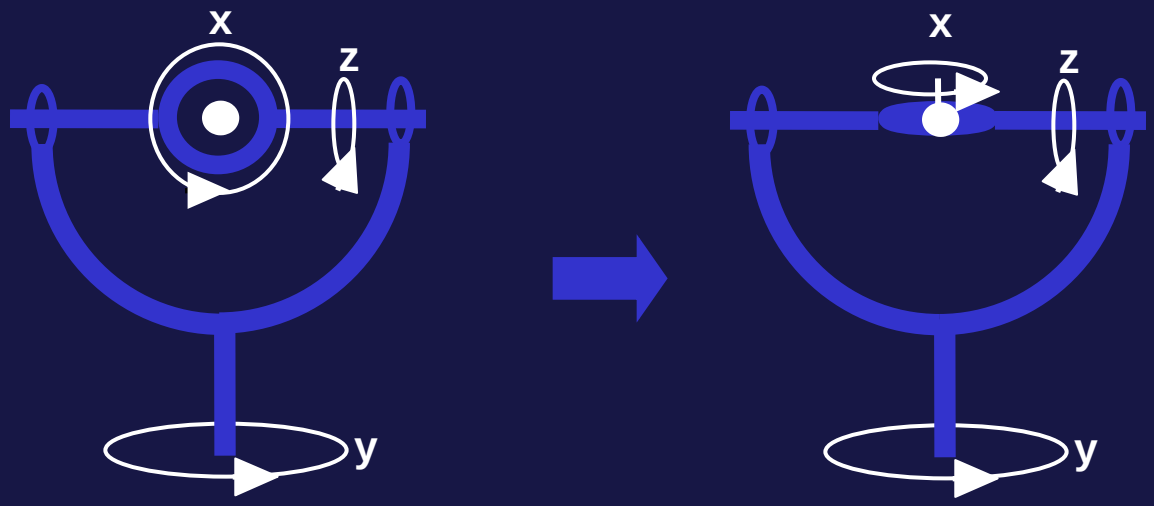
$$R_x(\alpha)R_y(\beta)R_z(\gamma)P = R_z(\gamma)R_y(\beta)R_x(\alpha)P$$

Euler Fixed



Roll Pitch Yaw

Gimbal Lock



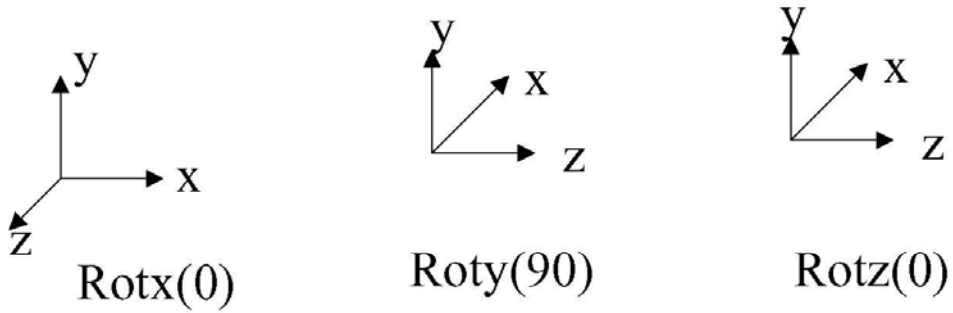
Gimbal **Locked Gimbal**

A *Gimbal* is a hardware implementation of Euler angles (used for mounting gyroscopes, expensive globes)

Gimbal lock is a basic problem with representing 3-D rotations using Euler angles

Gimbal Lock—Shown another way

- A 90 degree rotation about the y axis aligns the first axis of rotation with the third.



- Incremental changes in x,z produce the same results – lost a degree of freedom

Interpolating Rotations

- Q: What kind of a compound rotation do you get by successively turning about each of the three axes of the rotation at a constant rate?

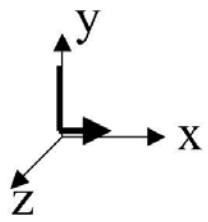
- A: not the one you want.

Example

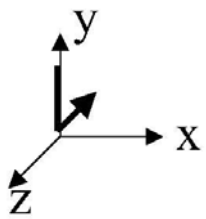
- Especially a problem if interpolating say…

(0,90,0) ⟶ (90, 45, 90)

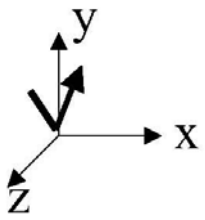
Just a 45 degree rotation from one orientation to the next, so we expect 90, 22.5, 90, but get 45, 67.5, 45



Initial Orientation
(object space)



(0,90,0)



(90, 45, 90)

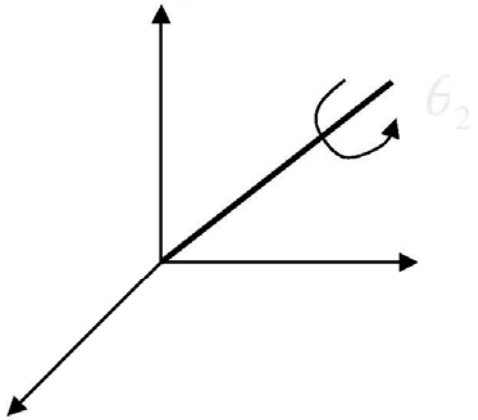
Axis Angle

Euler's Rotation Theorem:

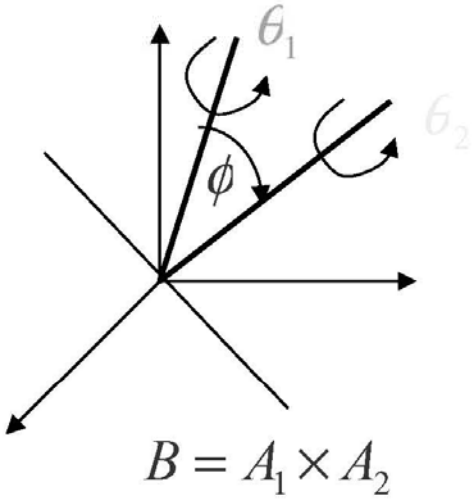
Any orientation can be represented by a 4-tuple

- angle, vector(x,y,z) where the angle is the amount to rotate by and the vector is the axis to rotate about

- Can interpolate the angle and axis separately



Axis Angle Interpolation



$$B = A_1 \times A_2$$

$$\phi = \cos^{-1}\left(\frac{A_1 \bullet A_2}{|A_1||A_2|}\right)$$

$$A_k = R_B(k\phi)A_1$$

$$\theta_1 = (1-k)\theta_1 + k\theta_2$$

Axis Angle

- Can interpolate the angle and axis separately
- No gimbal lock
- But, can't efficiently compose rotations…must convert to matrices first

Quaternions

- Good interpolation
- Can be multiplied (composed)
- No gimbal lock

Quaternions

- 4-tuple of real numbers
  - s,x,y,z or [s,v]
  - s is a scalar
  - v is a vector
- Same information as axis/angle but in a different form

$$q = Rot_{\theta(x,y,z)} = \left[\cos(\theta/2), \sin(\theta/2) \bullet (x, y, z)\right]$$

Quaternion Math

Addition:

$$[s_1, v_1] + [s_{2,} v_2] = [s_1 + s_2, v_1 + v_2]$$

Multiplication:

$$[s_1, v_1] \cdot [s_{2,} v_2] = [s_1 \cdot s_2 - v_1 \bullet v_{2,} s_1 \cdot v_2 + s_2 \cdot v_1 \times v_{2]}]$$

Multiplication is not commutative but is associative
(just like transformation matrices, as you would expect)

$$q_1 q_2 \neq q_2 q_1$$

$$(q_1 q_2) q_3 = q_1 (q_2 q_3)$$

Quaternion Math

A point in space is represented as $[0, v]$

$[1,(0,0,0)]$ multiplicative identity

$$q^{-1} = (1/\|q\|)^2 \cdot [s,-v]$$

$$\textit{where} \quad \|q\| = \sqrt{s^2 + x^2 + y^2 + z^2}$$

$q \cdot q^{-1} = [1,(0,0,0)]$ the unit length quaternion (and multiplicative identity)

Quaternion Rotation

To rotate a vector, v using quaternions

- –represent the vector as [0,v]
- –represent the rotation as a quaternion, q

$$q = Rot_{\theta,(x,y,z)} = [\cos(\theta/2), \sin(\theta/2) \cdot (x, y, z)]$$

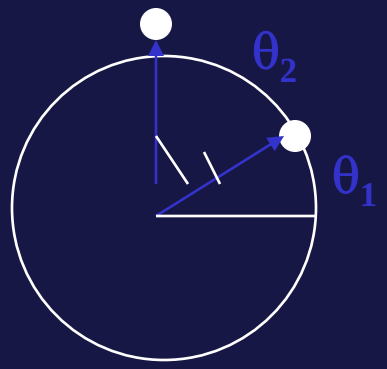
$$v' = Rot_q(v) = q \cdot v \cdot q^{-1}$$

Can compose rotations as well

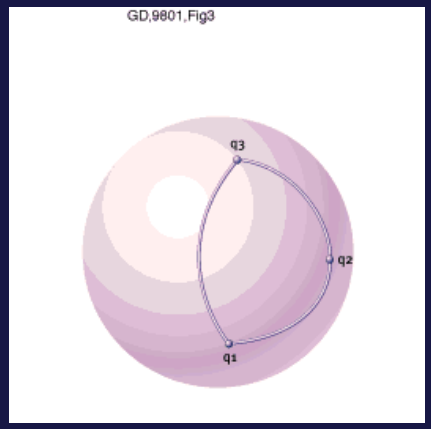
Looks good so far…we can easily specify and compose rotations!

Quaternion Interpolation

We can think of rotations as lying on an n-D unit sphere



1-angle (θ) rotation
(unit circle)

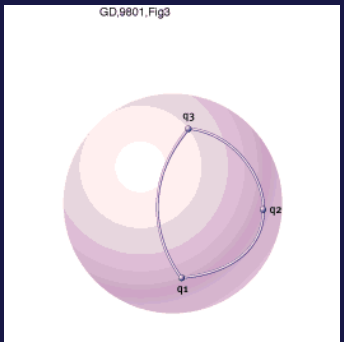


2-angle (θ-ϕ) rotation
(unit sphere)

Interpolating rotations means moving on n-D sphere

Quaternion Interpolation

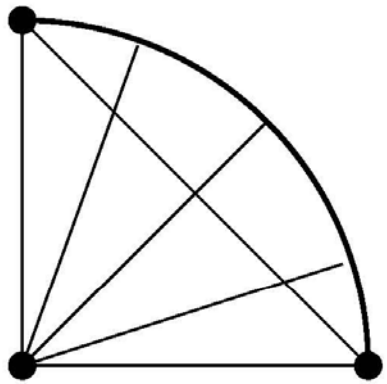
- Interpolating quaternions produces better results than Euler angles
- A quaternion is a point on the 4-D unit sphere
  - interpolating rotations requires a unit quaternion at each step - another point on the 4-D sphere
  - move with constant angular velocity along the great circle between the two points
  - Spherical Linear intERPolation (SLERPing)
- Any rotation is given by 2 quaternions, so pick the shortest SLERP
- To interpolate more than two points:
  - solve a non-linear variational constrained optimization (numerically)
- Further information: Ken Shoemake in the Siggraph '85 proceedings (*Computer Graphics,* V. 19, No. 3, P.245)



2-angle (θ-ϕ) rotation (unit sphere)

Quaternion Interpolation

- Direct linear interpolation does not work
  - Linearly interpolated intermediate points are not uniformly spaced when projected onto the circle



- Use a special interpolation technique
  - Spherical linear interpolation
  - viewed as interpolating over the surface of a sphere

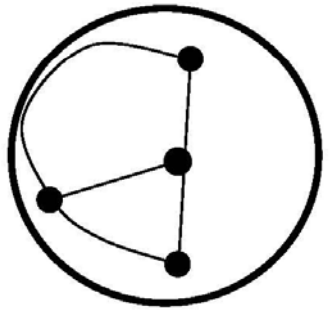
$$slerp(q1, q2, u)$$
$$= ((\sin((1-u)\cdot\theta))/(\sin\theta))\cdot q_1 + (\sin(u\cdot\theta))/(\sin\theta)\cdot q_2$$

- Normalize to regain unit quaternion

Two Representations of a Rotation

A quaternion and its negation [-s,-v] represent the same rotation:

$$
\begin{aligned}
-q &= Rot_{-\theta,-(x,y,z)} \\
&= [\cos(-\theta/2), \sin(-\theta/2)\cdot -(x,y,z)] \\
&= [\cos(\theta/2), -\sin(\theta/2)\cdot -(x,y,z)] \\
&= [\cos(\theta/2), \sin(\theta/2)\cdot (x,y,z)] \\
&= Rot_{\theta,(x,y,z)} \\
&= q
\end{aligned}
$$



Have to go the short way around!

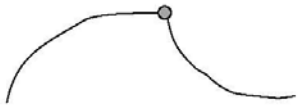
$$\cos(\theta) = q_1 \bullet q_2 = s_1 s_2 + v_1 \bullet v_2$$

$$\text{if } \cos(\theta) > 0 \Rightarrow q_1 \rightarrow q_2 \text{ shorter}$$

$$\text{else } q_1 \rightarrow -q_2 \text{ shorter}$$

Quaternion Interpolation

- As in linear interpolation in Euclidean space, we can have first order discontinuity



Solution is to formulate a cubic curve interpolation—see book for details

Quaternion Rotation

The rotation matrix corresponding to a quaternion,q, is

$$
\begin{aligned}
q &= Rot_{\theta,(x,y,z)} \\
&= [\cos(\theta/2), \sin(\theta/2)\cdot(x,y,z)] \\
&= [s,a,b,c]
\end{aligned}
$$

$$
\begin{bmatrix}
1-2b^2-2c & 2ab+2sc & 2ac-2sb \\
2ab-2sc & 1-2a^2-2c^2 & 2bc+2sc \\
2ac+2sb & 2bc-2sa & 1-2a^2-2b^2
\end{bmatrix}
$$

Rotations in Reality

- We can convert to/from any of these representations
  - but the mapping is not one-to-one
- Choose the best representation for the task
  - input: Euler angles
  - interpolation: quaternions
  - composing rotations: quaternions, orientation matrix
  - drawing: orientation matrix

Problems with Interpolation

- Splines don't always do the right thing
- Classic problems
  - Important constraints may break between keyframes
    - » feet sink through the floor
    - » hands pass through walls
  - 3D rotations
    - » Euler angles don't always interpolate in a natural way
- Solutions:
  - More keyframes!
  - Quaternions help fix rotation problems

Washing out Yaw

- Amc file format in Euler angles
- You want roll and pitch + delta yaw not roll, pitch, yaw
- Convert to yaw, roll, pitch Euler angles

  http://vered.rose.utoronto.ca/people/david_dir/GEMS/GEMS.html

- Compute delta yaw between frames
- Re-animate by incrementing yaw by delta yaw. Compute new rotation matrix.
- Compute delta x,z and increment to animate

Evaluation function?

Lots of possibilities here:

- joint angles
- weighted joint angles (visually big ones)
- root height, roll, pitch
- scattered points (a la Kovar and Gleicher)
- contact state
- what else?