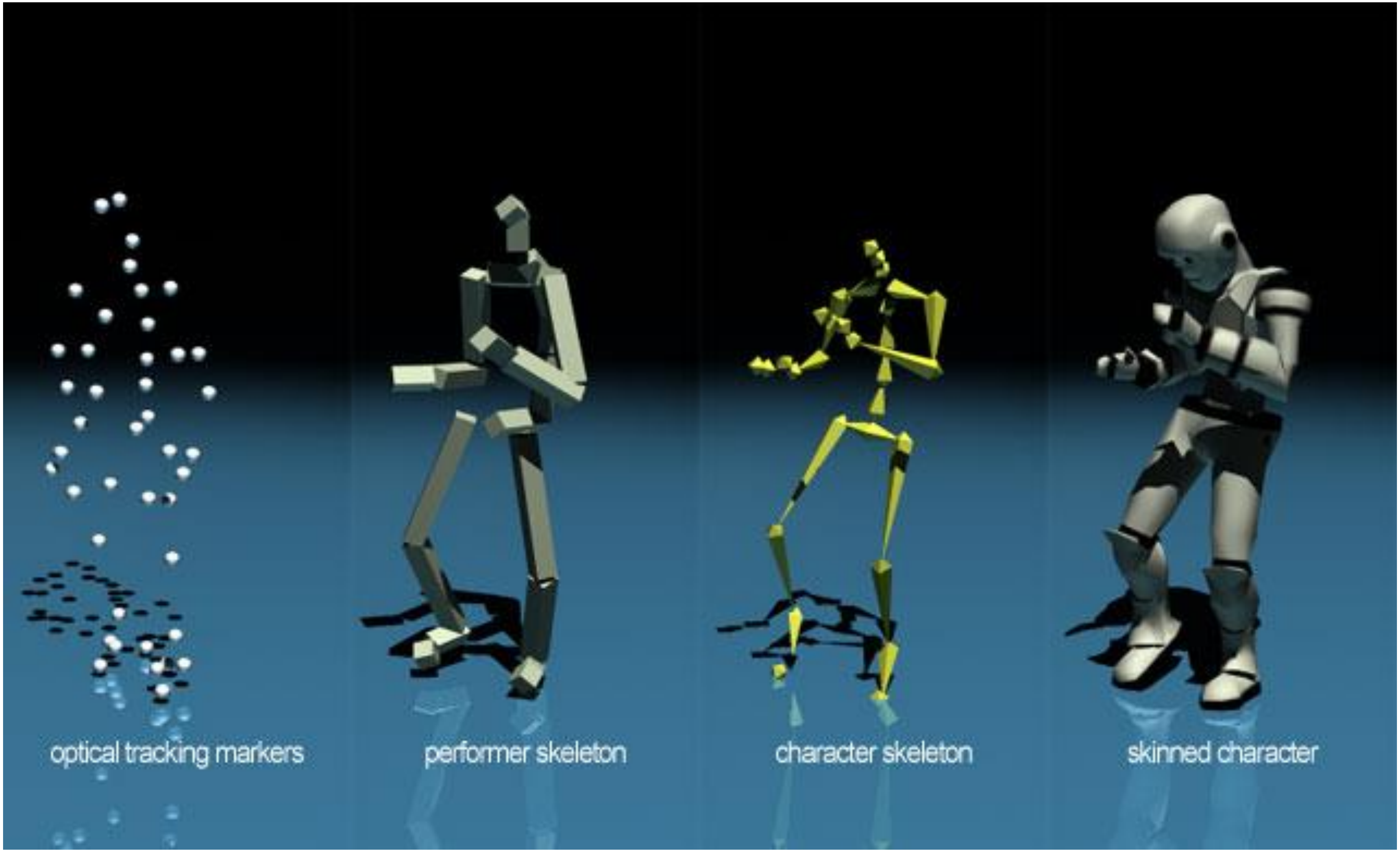
Motion Capture & Simulation

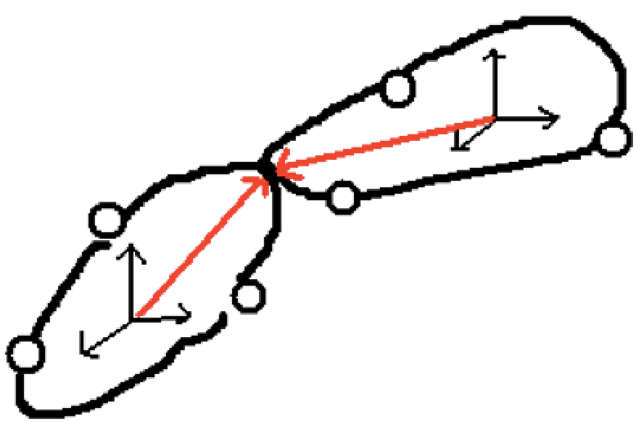
Motion Capture

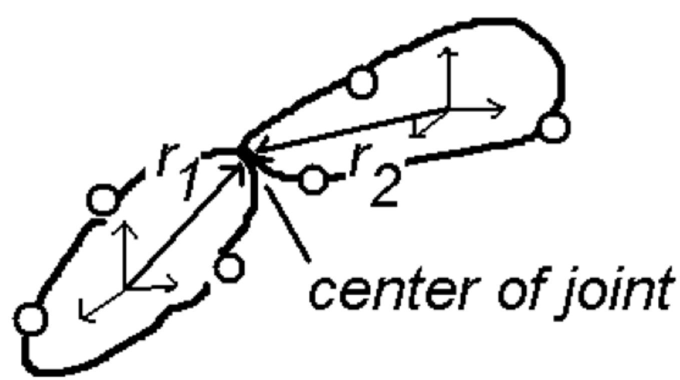
Character Reconstructions



Joint Angles

- Need 3 points to compute a rigid body coordinate frame
  - 1st point gives 3D translation, 2nd point gives 2 angles, 3rd point gives the last angle
  - Label markers by hand, so the system knows which three points to use for each rigid body/bone
  - If markers disappear (become occluded), have to re-label them after they show back up (extensive manual intervention)
- Once both rigid bodies associated with a joint are identified, the joint angle can be determined
- Note: Due to errors, neighboring bones may disagree in the location of the joint





MoCap Hardware

Mechanical Motion Capture

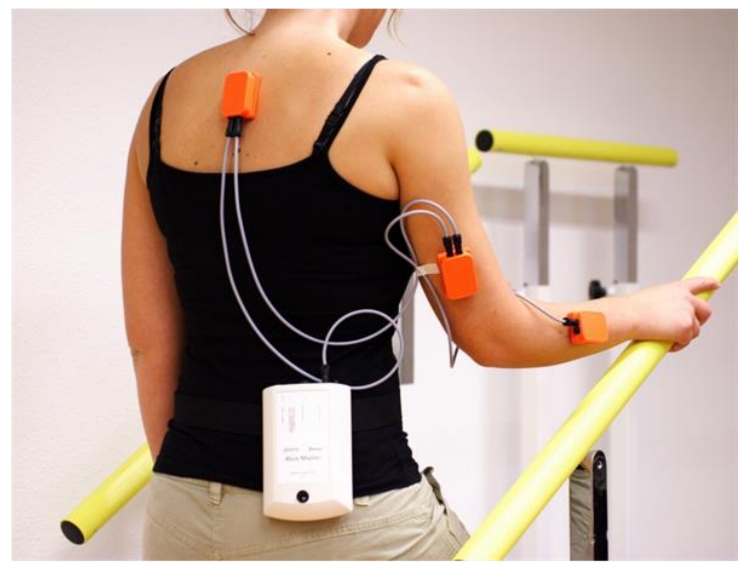
- Actors wear sensorized mechanical joints that directly measure the rotation of human joints
- Reduces manual intervention, but expensive and cumbersome

Wiimote

- Wiimote optical sensor images LEDs
  - Distance between the LEDs on the is bar fixed
  - Distance between <u>imaged</u> LEDs varies with depth
  - These two distances allow one to calculate how far away the remote is from the LED bar
- The angle of the remote is calculated from the angle the imaged LEDs make on the optical sensor

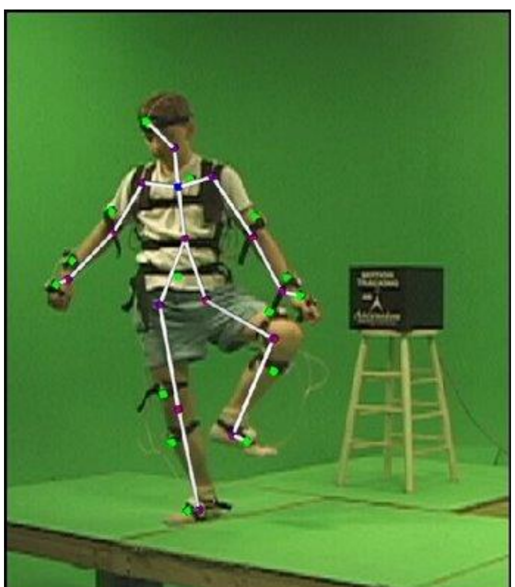
Inertial Tracking

- Wiimote (accelerometer) and Wiimotion Plus (gyroscope)
- Actors can also wear accelerometers and gyroscopes
- Accelerometers measure linear acceleration
  - Solve the usual system of ODEs with known accelerations to obtain velocity and position (errors cause drift)
- Gyroscope sensors measure the angular velocity
  - Solve the usual ODE to obtain orientation (errors cause drift)



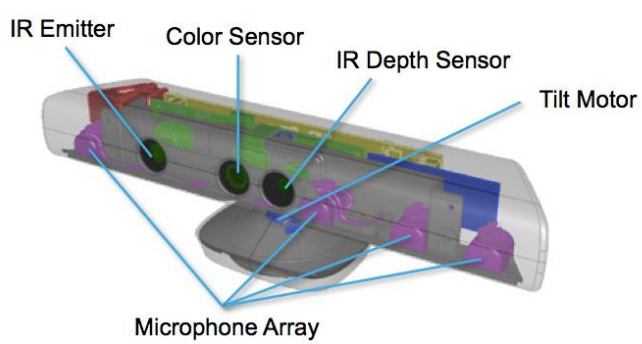
Magnetic Motion Capture

- Place emitters that produce three orthonormal magnetic fields
- Actors wear magnetic field detectors/receivers
  - Receivers detect the field strength, which gets weaker based on distance
- In addition to the accelerometer and gyroscope, the PSMove uses a magnetometer (and the Earth's magnetic field) to correct for drift from solving ODEs
  - Also uses a separate optical camera that measures the location, size, and orientation of the sphere (which becomes an ellipse when projected)



Marker-less Motion Capture

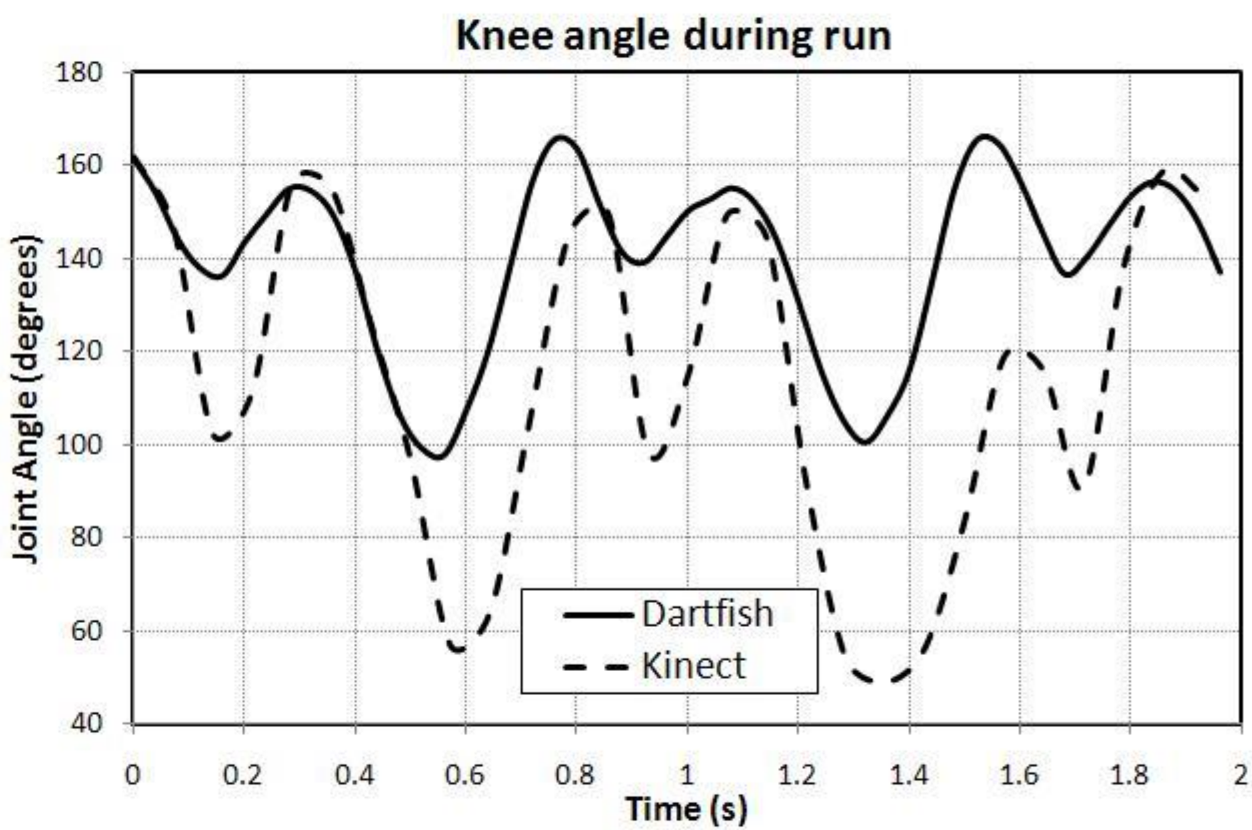
- Actors wear nothing!
- Use computer vision to reconstruct the person
  - Do a scan of a person ahead of time, and identify key points
  - Match key points between the video and the scan
- Microsoft Kinect - inexpensive depth sensor (from structured light)



MoCap Data

Mocap Output

- Root position/orientation vs. time
- Angles vs. time for all joints
- For example…



Using Mocap

- Mocap data can be mapped to a CG avatar anywhere in the scene by adding an arbitrary translation to the root position vs. time graph
  - Subsequently the avatar moves as the mocap data prescribes
    - Ignoring errors in the process
- But what if, for example, we have mocap data for walking straight and prefer to follow a curved path?
  - Could capture mocap data for every possible curved path, but that requires a lot of data
  - Similarly, could capture mocap data for every possible speed of movement
  - One also needs data for transitions between different radii of curvature, transitions between different speeds, and all combinations of curvature and speeds
  - Even more data is required for characters of different heights, since the angles and angles versus time will be different
  - Etc.
- Thus, many methods exist for using mocap data which doesn't exactly fit the task at hand

Mocap Examples

https://www.youtube.com/watch?v=BKkKTcHL5ow

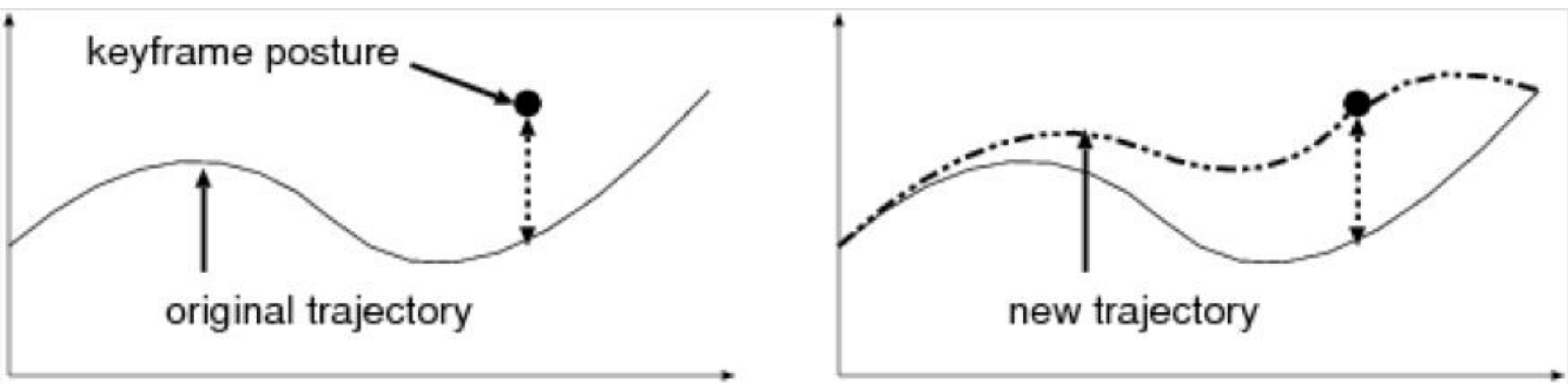
Modifying MoCap Data

Inverse Kinematics

- Suppose a foot misses the ground, or a hand fails to touch a cup
- Select a key frame in the mocap that has such an issue
- Use IK to adjust the end effector to meet the desired goal
- Treat the null space by minimizing the change in joint angles required to meet the goal
- Finally use this new key frame data to (smoothly?) modify the surrounding mocap data (angle vs. time graphs)

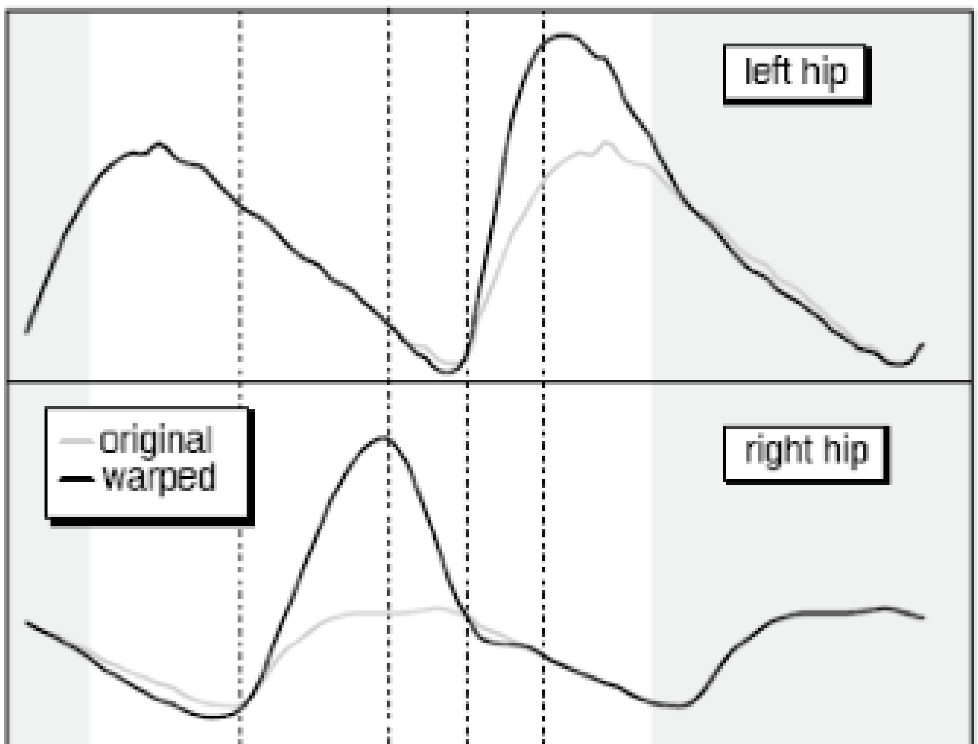
Inverse Kinematics

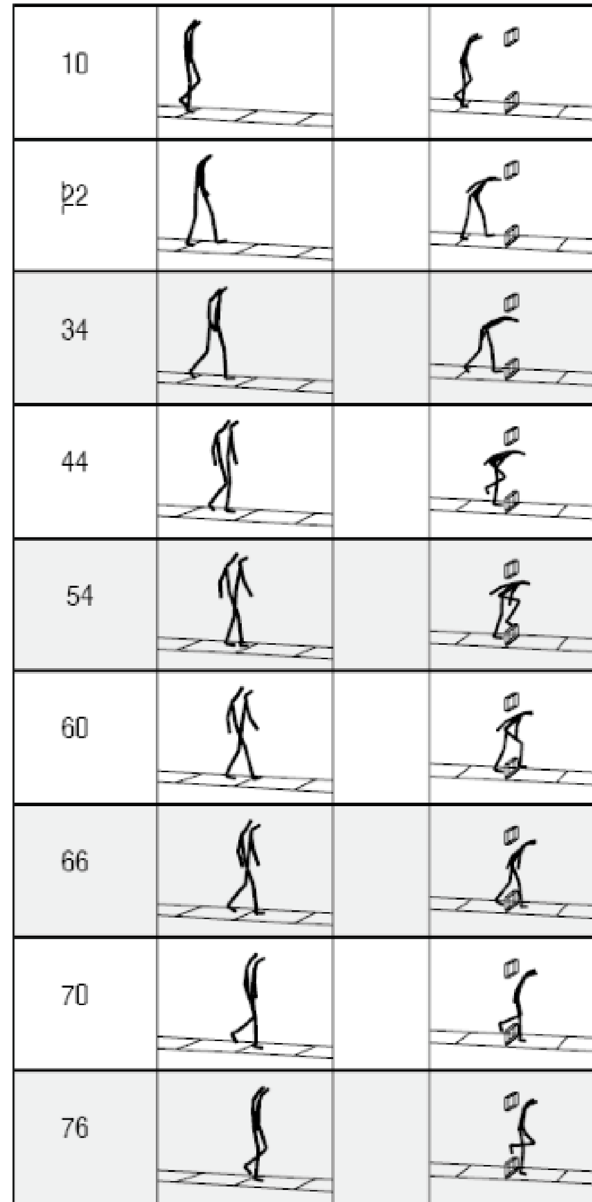
- Use various 1D interpolation techniques (e.g. splines) to ensure that the new function:
  - goes through the new key pose
  - modifies the original function as little as possible
  - remains smooth
  - Etc.



Motion Editing/Warping

- More generally one can specify new key frames to serve any purpose (motion editing)
- And then modify the 1D functions of angle vs. time to respect the newly edited key frames (motion warping)





Spacetime Constraints

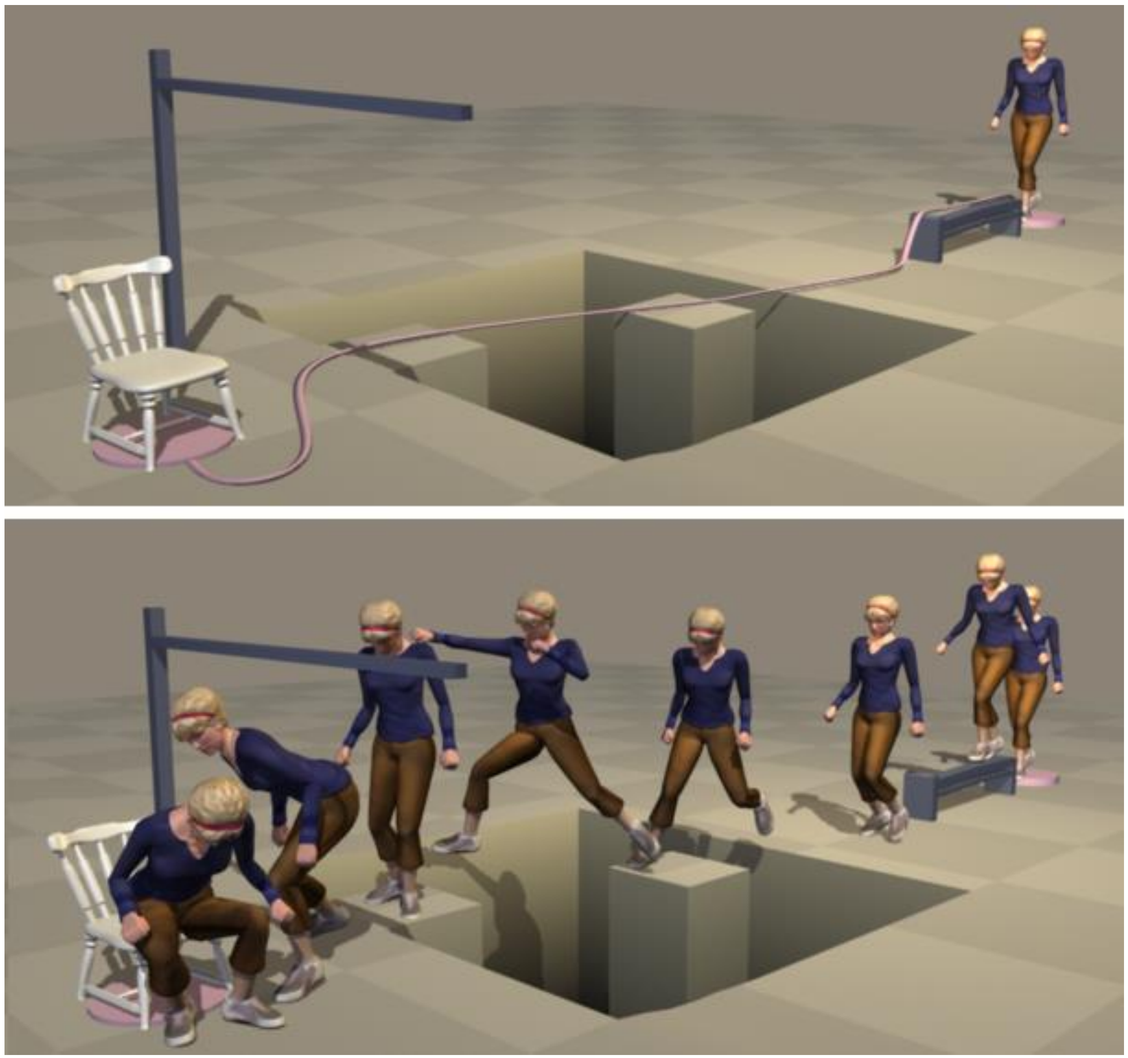
- Instead of specifying key frames, specify a number of constraints or goals throughout the time of the motion
- Then use constrained space/time optimization to find a new motion
- The optimization penalizes missing goals as well as the distance from the initial mocap data
- Besides goals for end effectors, more general constraints may include energy minimization, grace, style, efficiency, quality, etc.
- Expensive!

a given motion	$\mathbf{m}_0(t)$
an unknown motion	$\mathbf{m}(t) = \mathbf{m}_0(t) + \mathbf{d}_{dis}(t)$
a set of constraints	$f_i(\mathbf{q}^{ti}) = c_i \quad i = 1 \ldots k$
a function to be minimized	$\mathbf{g}(\mathbf{m}) = \int_t \mathbf{d}_{dis}(t)^2$

MoCap Transitions

Motion Transitions

- Typically, an avatar executes many different kinds of motion
- Need ways to transition from one motion type to another

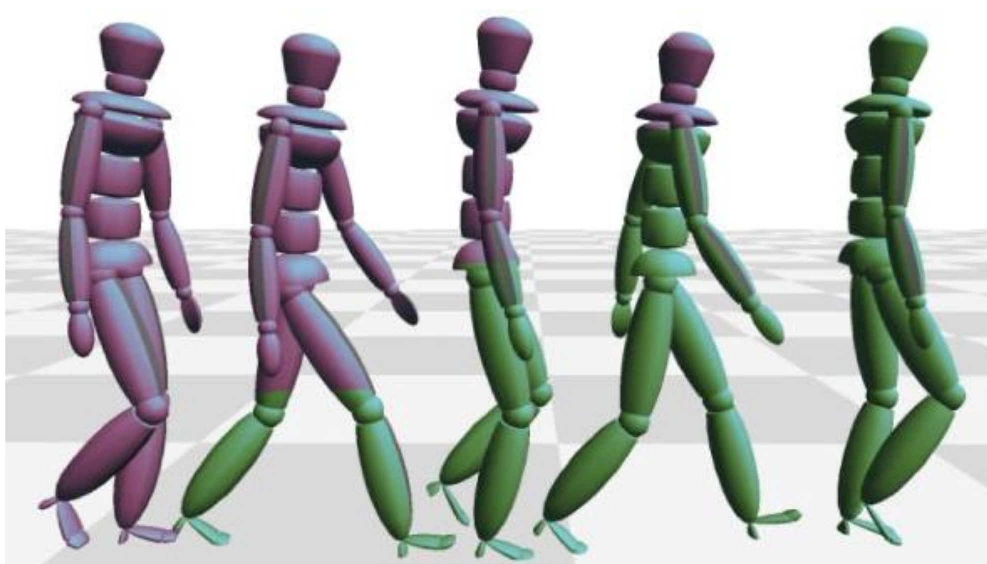


Motion Blending

- Given two motions, say walking and running, how does one transition from walking to running?
- Use interpolation between the two motions:

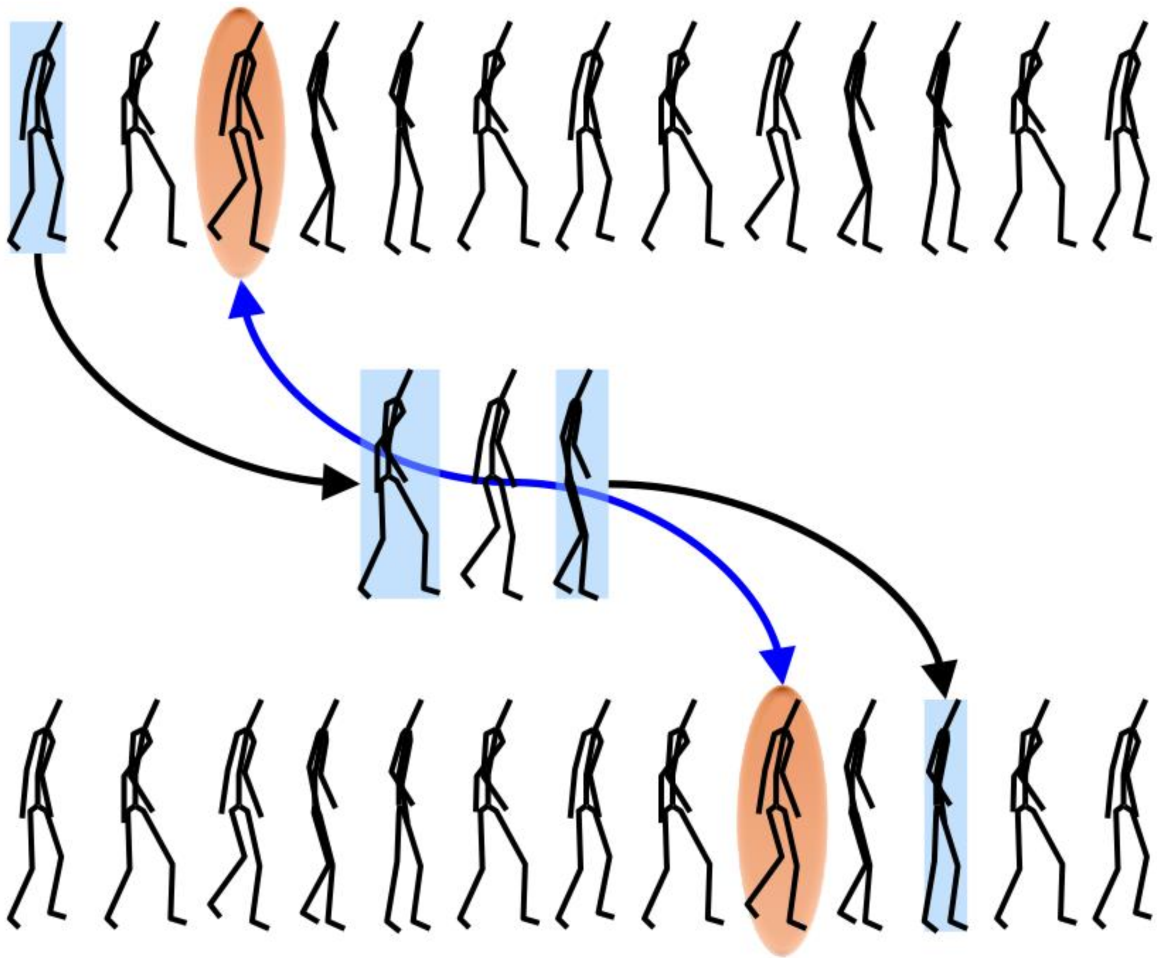
$$f(t) = A(t)(1 - s(t)) \; + \; B(t)s(t)$$

  - A(t) is the angle vs. time graph for walking
  - B(t) is the angle vs. time graph for running
  - f(t) is a new angle versus time graph that transitions from walking to running
  - s(t) starts at 0 (walking) and at some point transitions to a value of 1 (running)
- Works ok in some cases
  - Timings can be off, can look odd



Motion Graphs

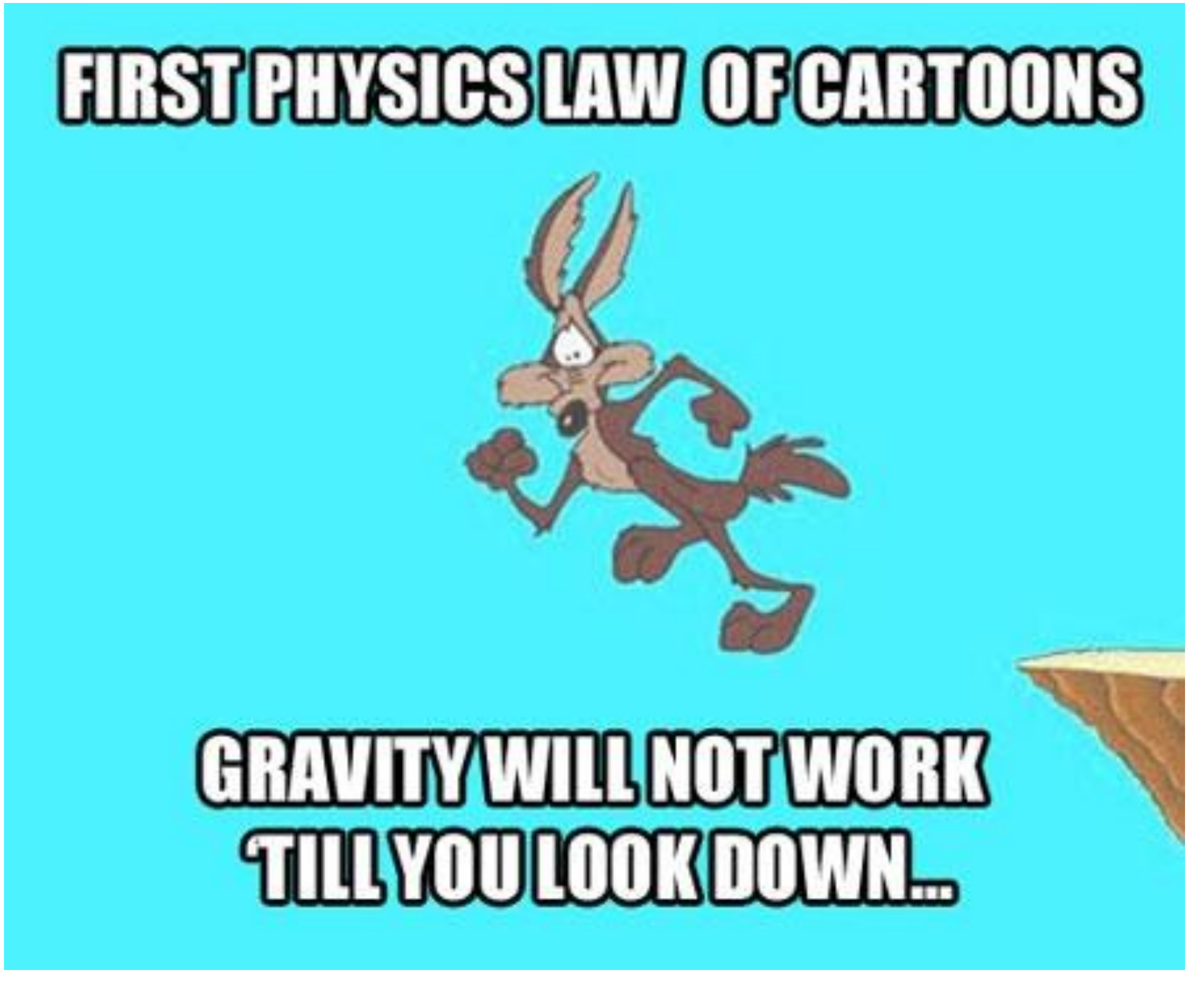
- Automatically look for acceptable transition points between two separate motions
- Then cut, splice, and blend…



MoCap & Simulation

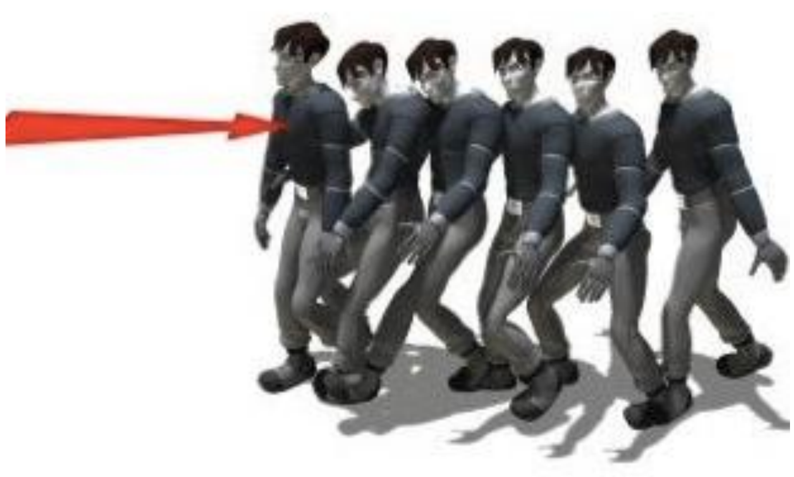
Simulation

- Kinematically following mocap data gives the character infinite mass and inertia
- A little bit of physics can be added for realism…



Interrupting Motion Capture

- If a character is given a small push, one can briefly simulate a reaction to collisions, etc., treating the character with finite mass
- After such a treatment, the character's state no longer matches that of the motion capture functions of angle vs. time
- Need to somehow recover
- Search for the closest motion capture frame to the current state and perform a motion graph transition



PD Control

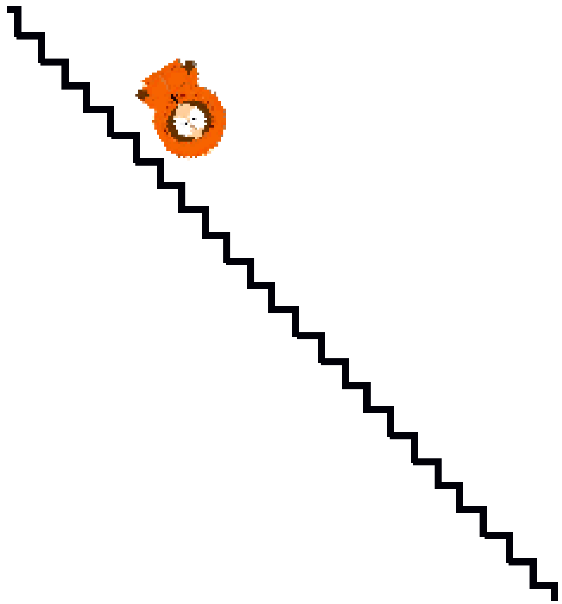
- When an object is perturbed, keep targeting the current motion capture frame using a mass spring style formulation
- This gives a somewhat plausible transition back to the mocap angle vs. time functions from a perturbed state
- Apply a torque proportional to the difference between the current state and the target state
  - Along with an additional damping torque proportional to the difference between the time derivatives ("angular velocities")

$$\mathrm{T} = c_P(q - q_d) \; + c_D(\dot{q} - \dot{q}_d)$$

- The larger the difference in the states, the bigger the torque
- The coefficients can be adjusted to more tightly or loosely follow the motion capture data

Rag Dolls

- The avatar can be simulated as an <u>articulated rigid body</u>
- Whether only for a brief collision, or for a longer period of time (e.g. falling down stairs)



- The articulated rigid body simulation would typically not be influenced by the motion capture data
  - However, one could add a PD Control torque to the articulated rigid body simulation (as a non-physical, non-momentum conserving torque) in order to somewhat include the effects of mocap data in the simulation

Question #1

LONG FORM:

- Summarize motion capture technology.
- Answer the short form questions.

SHORT FORM:

- Identify the main avatar in your game.
- How will the way it <u>looks</u> be compelling or relevant to the game?
- How will the way it <u>moves</u> be compelling or relevant to the game?

Articulated Rigid Bodies

Davy Jones

More than just characters…

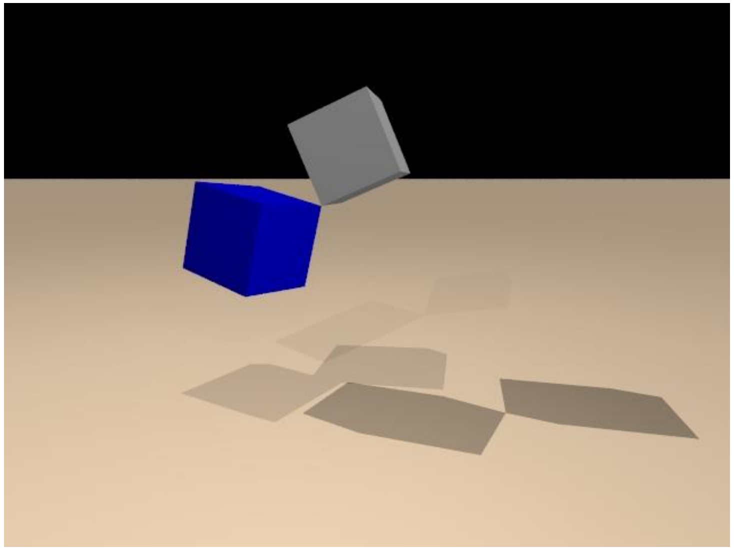
More than just characters…

Maximal vs. Reduced Coordinates

- Broadly speaking, there are two ways of going about simulating articulated rigid bodies
- Maximal (Cartesian) Coordinates
  - Simulate the full six degrees of freedom for each rigid body
  - Use some auxiliary method to enforce joint constraints
  - Contact, collision, friction, etc. are straightforward to include
- Reduced (Generalized) Coordinates
  - Remove any degrees of freedom that are constrained away by joints
    - write new differential equations for the position of the root and the joint angles)
  - Faster to simulate, since there are less degrees of freedom
  - Less intuitive equations, and less intuitive to incorporate contact, collision, friction and other unanticipated forces (but great for robots, since they’re not supposed to have unanticipated forces)

Enforcing Constraints

- Consider constraining two separate rigid bodies together at a point
- Each of the rigid bodies describes this point in its own object space
- Thus we know the world space location of the two points that are supposed to be coincident
- The goal is to constrain them to actually be coincident
- For example one could attach a zero length spring connecting these two points, and simulate the effect of the spring forces on the rigid bodies



Fixing Drift

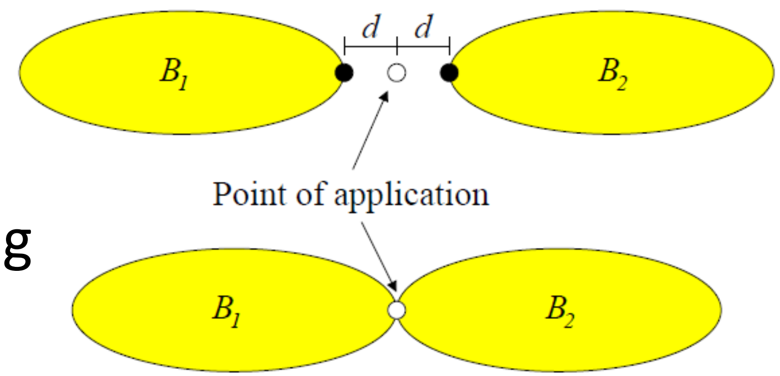
- Unless the zero length spring has infinite stiffness, which is difficult to simulate, the two points will drift somewhat apart
- This drift can be fixed at render time by projecting the current state of the rigid bodies into an acceptable joint state
  - For example, each of the two bodies can be translated along the segment connecting their constrained points until those points are coincident
    - In the case of a joint hierarchy one should only translate the rigid body further from the root
  - A similar procedure can be employed for constraints on angles (which can be enforced using angular springs)
- Note: this may cause other visual issues such as interpenetrations
- One could maintain this new projected state as the new simulation state as well
  - Although that then violates conservation of linear and angular momentum

Impulses for ARBs

Impulses

- Instead of using a spring to constrain the rigid bodies, one could use collision-style impulses
- Recall, an equal and opposite impulse is applied to each body using their impulse factors:

  $u_1^{new} = u_1 + K_1 j$ and $u_2^{new} = u_2 - K_2 j$
- The relative velocity $u_{rel} = u_1 - u_2$ should be identically zero after applying the impulse
- Solve $u_{rel} + K_T j = 0$ to find the impulse j
- The impulse is typically applied at the midpoint of the segment connecting the two points one is trying to keep coincident
- Unlike a spring that will fix the velocity and position, the impulse only fixes the velocity
- Thus, any position errors persist (creating drift that needs to be addressed)



Impulses for Drift

- During the position update, one could apply impulses to velocities in order to obtain velocities that close the gap
  - That is, evolving the rigid bodies using these velocities results in the two points moving to become coincident (eliminating drift)
  - Still use $u_1^{new} = u_1 + K_1 j$ and $u_2^{new} = u_2 - K_2 j$
  - The new relative velocity u_{rel}^{new} is not set to zero, but rather to the correct velocity to close the gap
  - Solve $u_{rel} + K_T j = u_{rel}^{new}$ to find the impulse j
  - Points on the rigid body move on nonlinear paths, since rigid bodies rotate and translate
  - Thus, finding the correct relative velocity and impulse is difficult
  - Requires a nonlinear solver, iterations, etc., but is doable…
- After updating the position to remove drift, new impulses are needed in order to make the relative velocity zero as well (as per the last slide)

Angular Impulses for ARBs

Angular Impulse

Recall: Equations for one body with collision location r_p with respect to its center of mass:

$$M\bar{v}^{new} = M\bar{v} + j$$

$$I\omega^{new} = I\omega + r_p^* j$$

- Let the point of application (and thus r_p) go to infinity while j goes to zero with $j_\tau = r_p^* j$ remaining finite:

$$M\bar{v}^{new} = M\bar{v} + 0$$

$$I\omega^{new} = I\omega + j_\tau$$

- This angular impulse only effects the angular momentum and not the linear momentum:

$$M\bar{v}^{new} = M\bar{v} + j$$

$$I\omega^{new} = I\omega + r_p^* j + j_\tau$$

Angular Impulse

- The pointwise velocity gets additionally modified by the angular impulse:

$$u_p^{new} = \bar{v}^{new} + \omega^{new*} r_p = \bar{v}^{new} + r_p^{*T} \omega^{new}$$

$$u_p^{new} = \bar{v} + \frac{j}{M} + r_p^{*T}\left(\omega + I^{-1} r_p^* j + I^{-1} j_\tau\right)$$

$$u_p^{new} = u_p + Kj + r_p^{*T} I^{-1} j_\tau$$

- Similarly for the angular velocity:

$$\omega^{new} = \omega + I^{-1} r_p^* j + I^{-1} j_\tau$$

- This gives us 2 vector valued equations for u_p^{new} and ω^{new} in 2 vector valued unknowns j and j_τ
- Thus we can target any velocity and angular velocity, constraining the joint together while also controlling its angle/rotation

Angular Impulse

- In addition to the relative velocity $u_{rel} = u_1 \; - u_2$ at the joint center, also consider the relative angular velocity $\omega_{rel} = \omega_1 \; - \omega_2$
- Equal and opposite impulses applied to each body:

$$u_1^{new} = u_1 + K_1 j + \; {r_1^*}^T I_1^{-1} j_\tau$$
$$u_2^{new} = u_2 - K_2 j - \; {r_2^*}^T I_2^{-1} j_\tau$$

- Equal and opposite angular impulses applied to each body:

$$\omega_1^{new} = \omega_1 + I_1^{-1} r_1^* j + I_1^{-1} j_\tau$$
$$\omega_2^{new} = \omega_2 - I_2^{-1} r_2^* j - I_2^{-1} j_\tau$$

- These equations can be combined and rewritten as:

$$u_{rel}^{new} = u_{rel} + K_T j + (\, {r_1^*}^T I_1^{-1} + \; {r_2^*}^T I_2^{-1})\; j_\tau$$
$$\omega_{rel}^{new} = \omega_{rel} + (I_1^{-1} r_1^* + I_2^{-1} r_2^*) j + (I_1^{-1} + \; I_2^{-1})\; j_\tau$$

- So given a desired relative velocity u_{rel}^{new} and a desired relative angular velocity ω_{rel}^{new}, we can solve for the impulse j and angular impulse j_τ