Kinematics

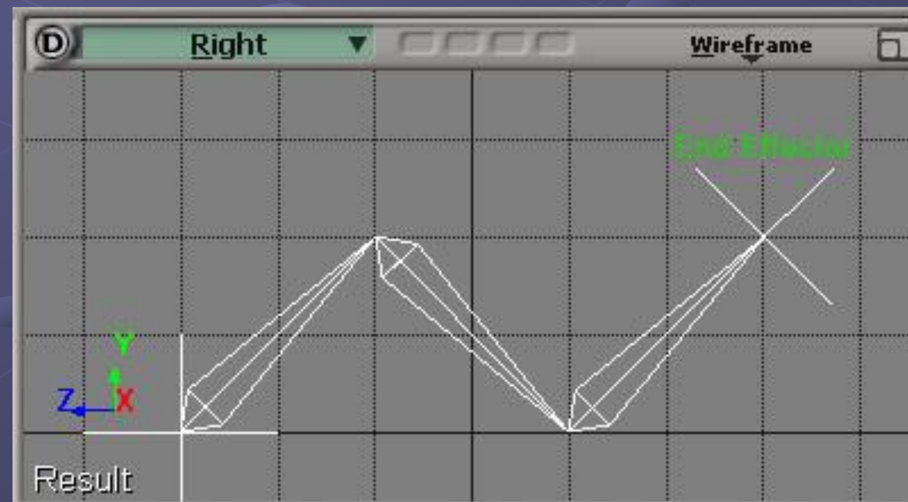
- Forward Kinematics (review)
- Inverse Kinematics

Forward Kinematics Review

Forward Kinematics (FK)

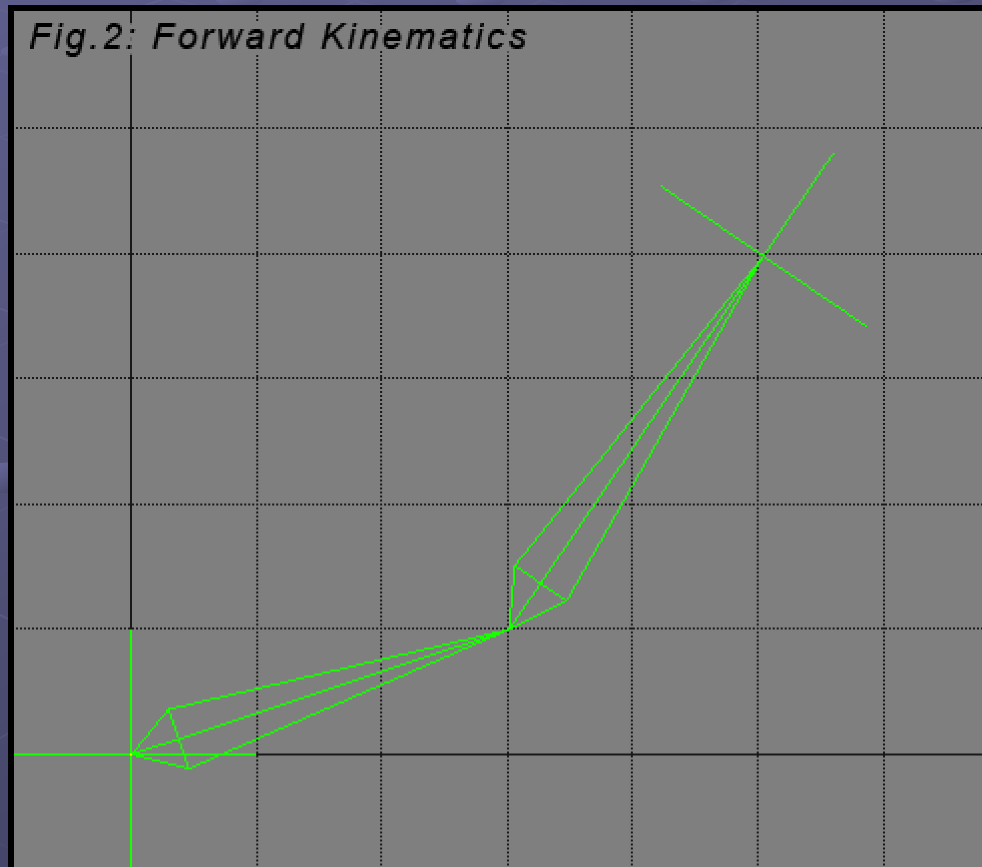
Incrementally manipulating each of the component parts of a flexible, jointed object to achieve an overall, desired pose.

Mathematically is concerned with finding the position of the end effector, given the angle of the joints and the length of each articulated segment.

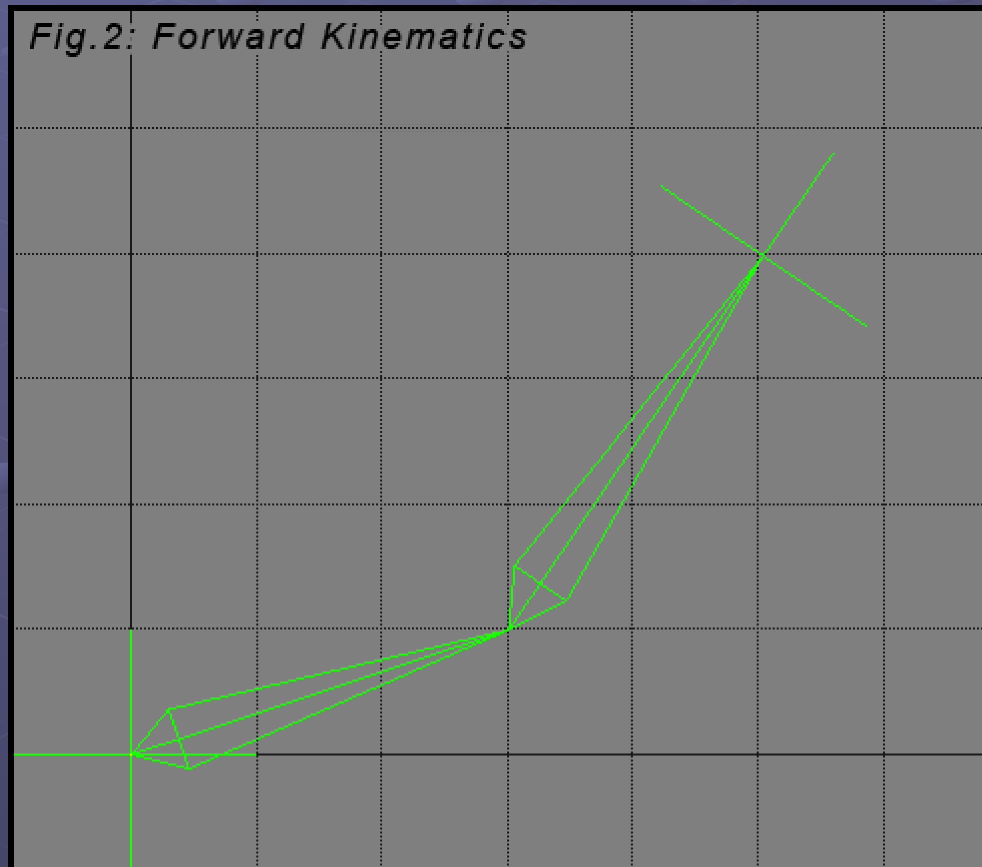


Slides courtesy Aryel Beck

FK Equation



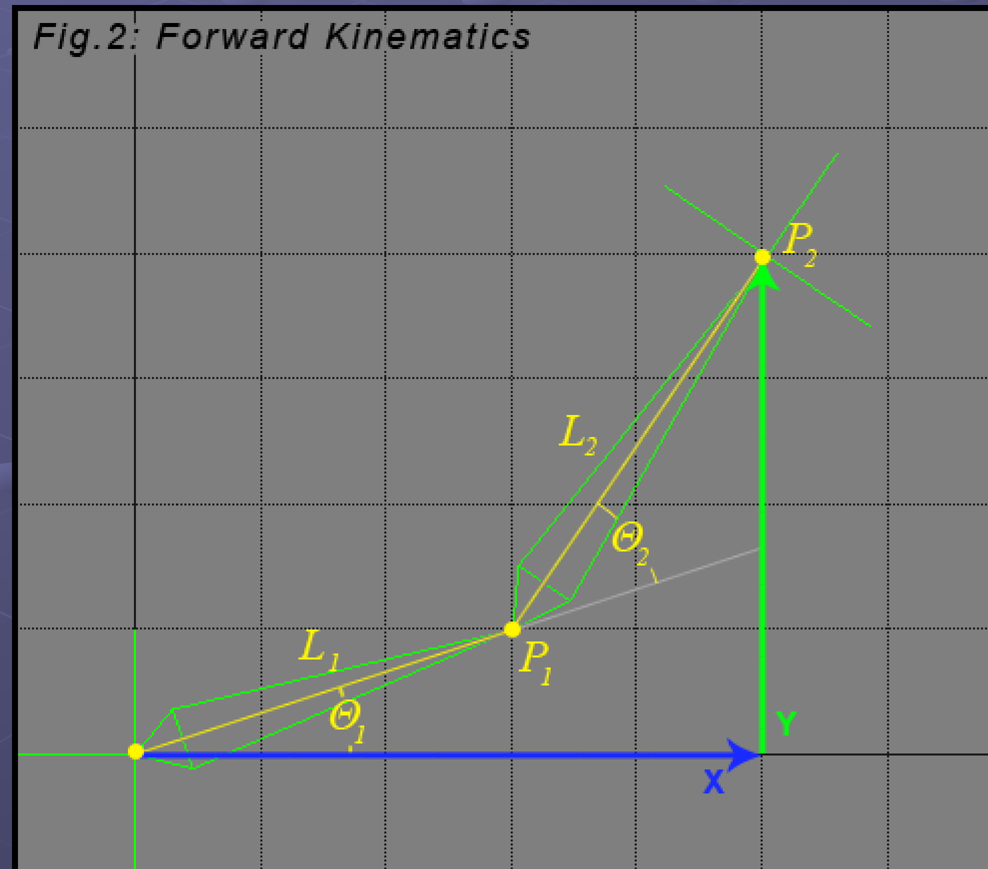
FK Equation



FK Equation

Expression for P1:

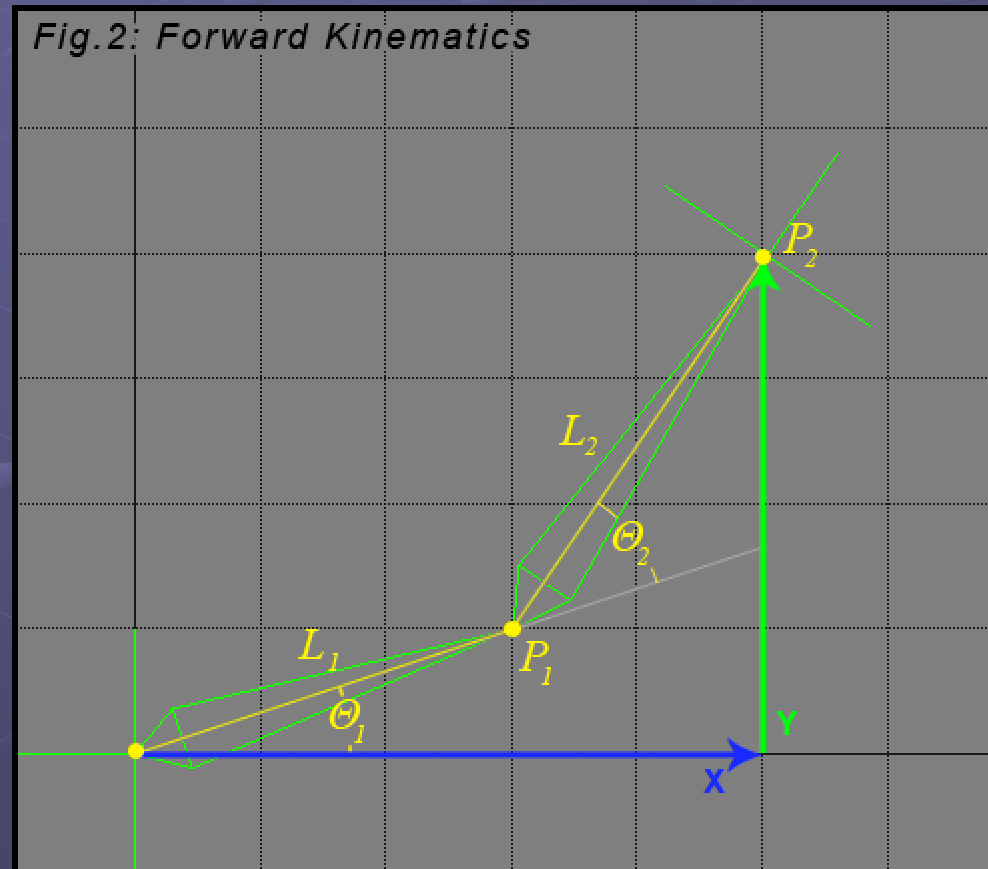
$$Px_1 = L_1 \times \cos(\Theta_1)$$
$$Py_1 = L_1 \times \sin(\Theta_1)$$



FK Equation

Expression for P2:

$$Px_2 = Px_1 + (L_2 \times \cos(\Theta_1 + \Theta_2))$$
$$Py_2 = Py_1 + (L_2 \times \sin(\Theta_1 + \Theta_2))$$



FK Equation

Expanded, FK Solution for P2:

$$Px_2 = L_1 \times \cos(\Theta_1) + (L_2 \times \cos(\Theta_1 + \Theta_2)$$

$$Py_2 = L_1 \times \sin(\Theta_1) + (L_2 \times \sin(\Theta_1 + \Theta_2)$$

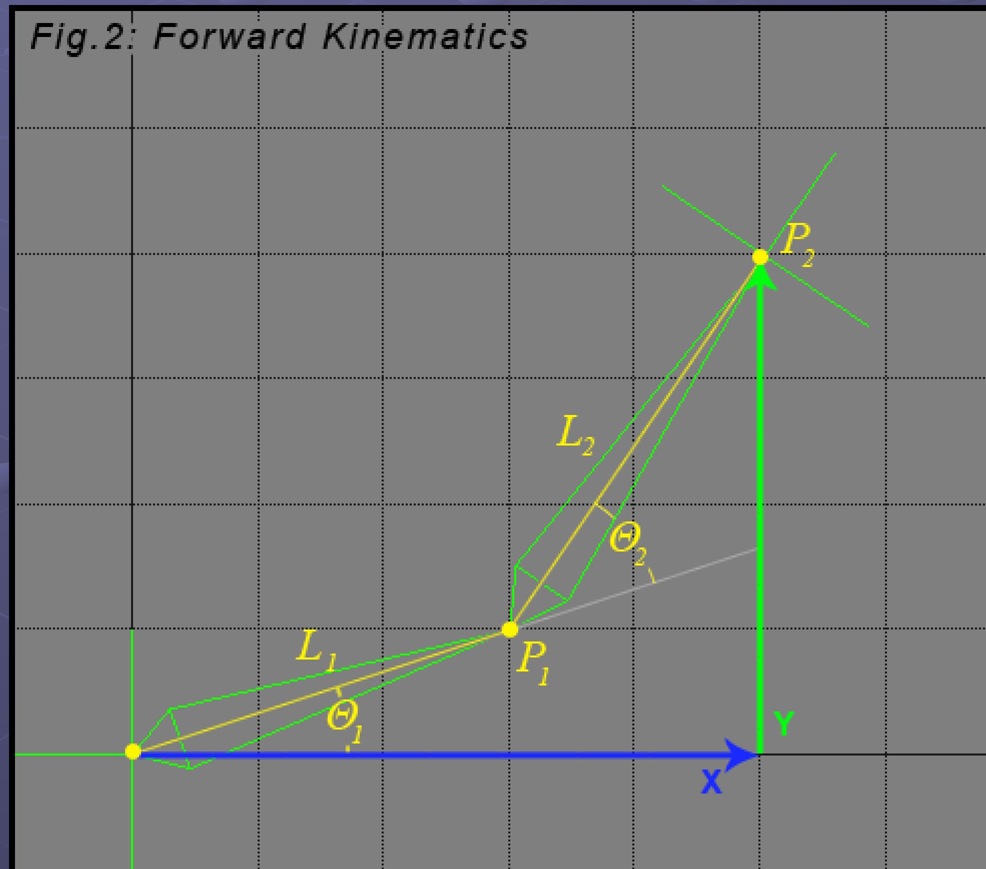


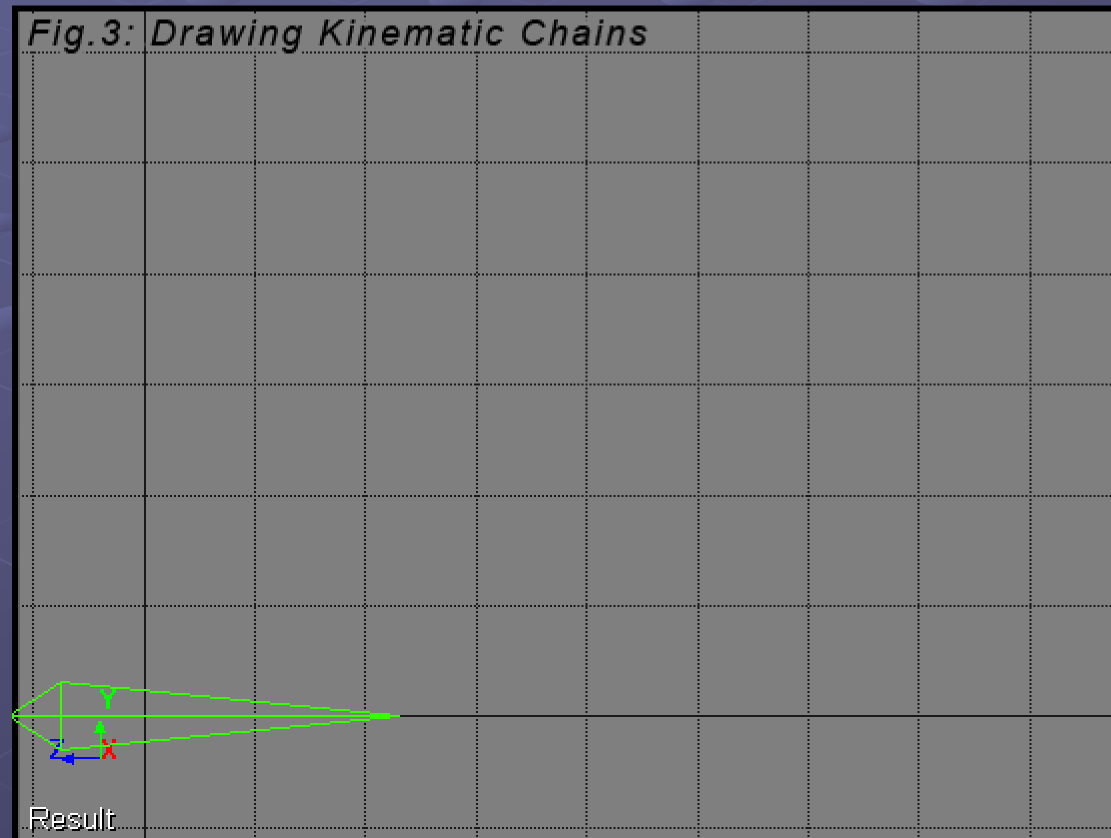
Fig.2: Forward Kinematics

Drawing Kinematic Chains

- Drawing kinematic chains requires that the links are drawn from the outermost link to the innermost.
- The positioning of each link requires translations and rotations from each other link prior to it.

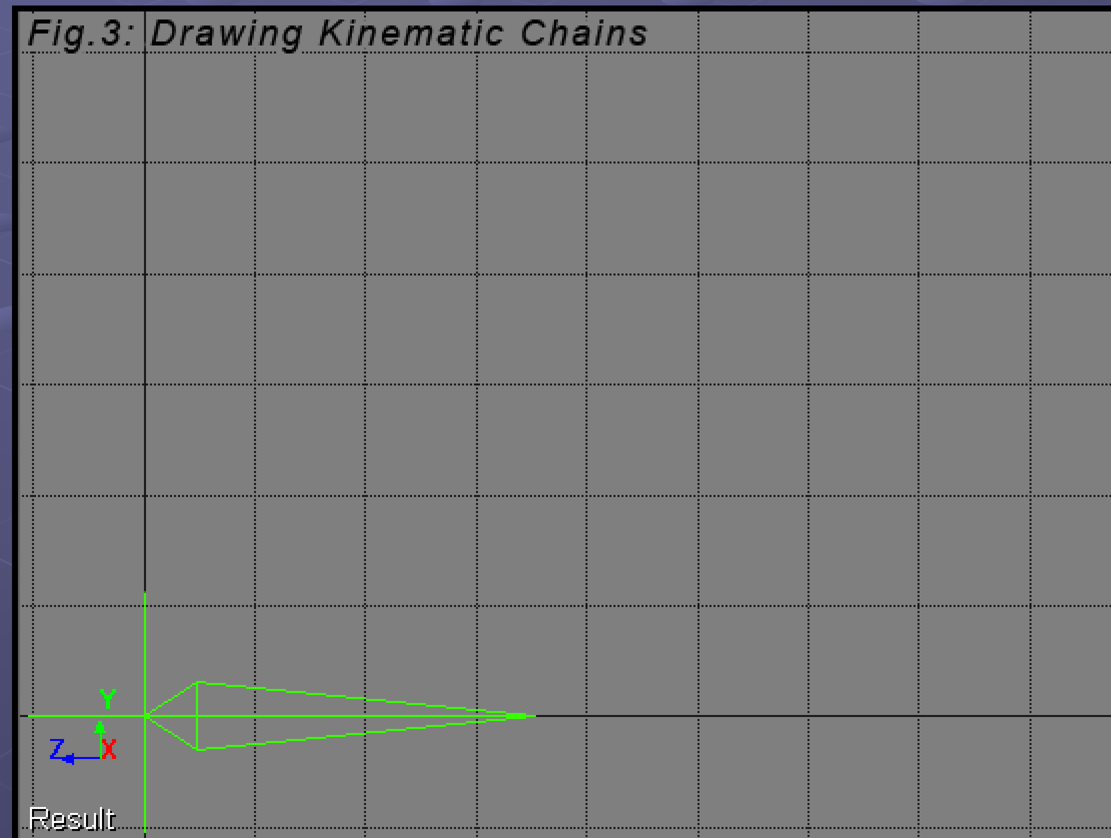
Drawing Kinematic Chains

- Starting with the effector's object:



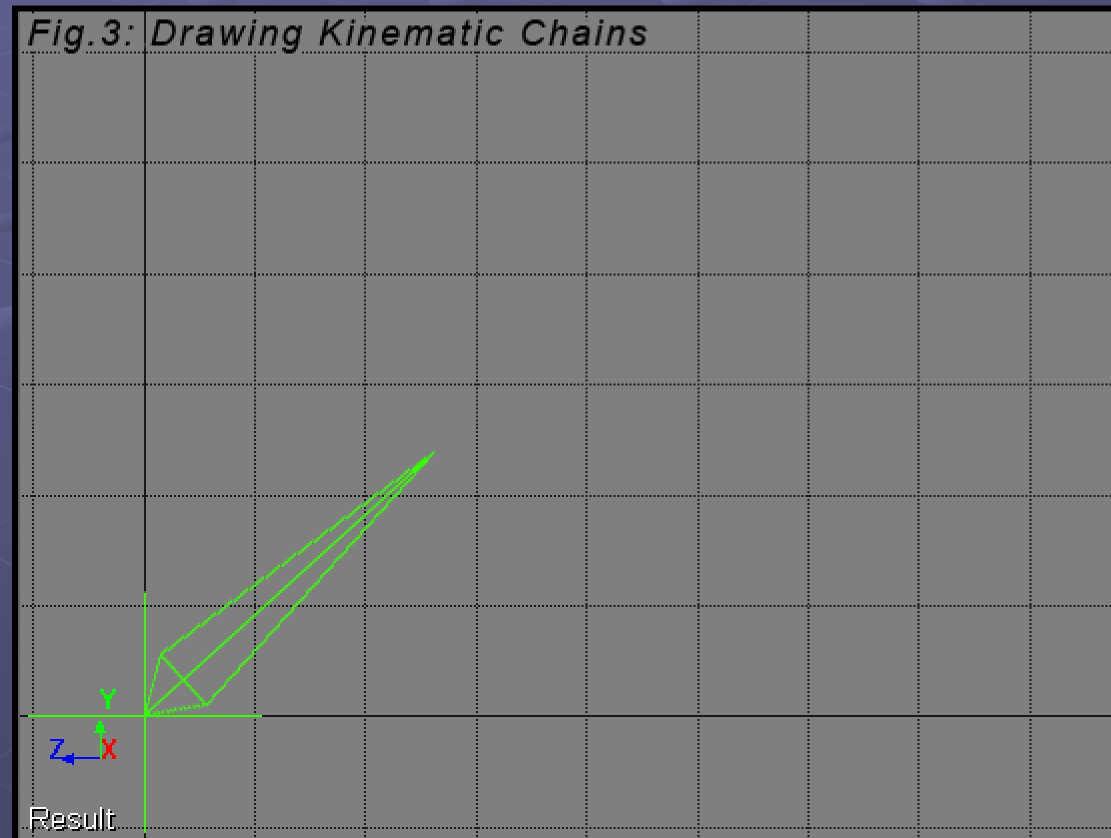
Drawing Kinematic Chains

- Starting with the effector's object:
  - Translate by length



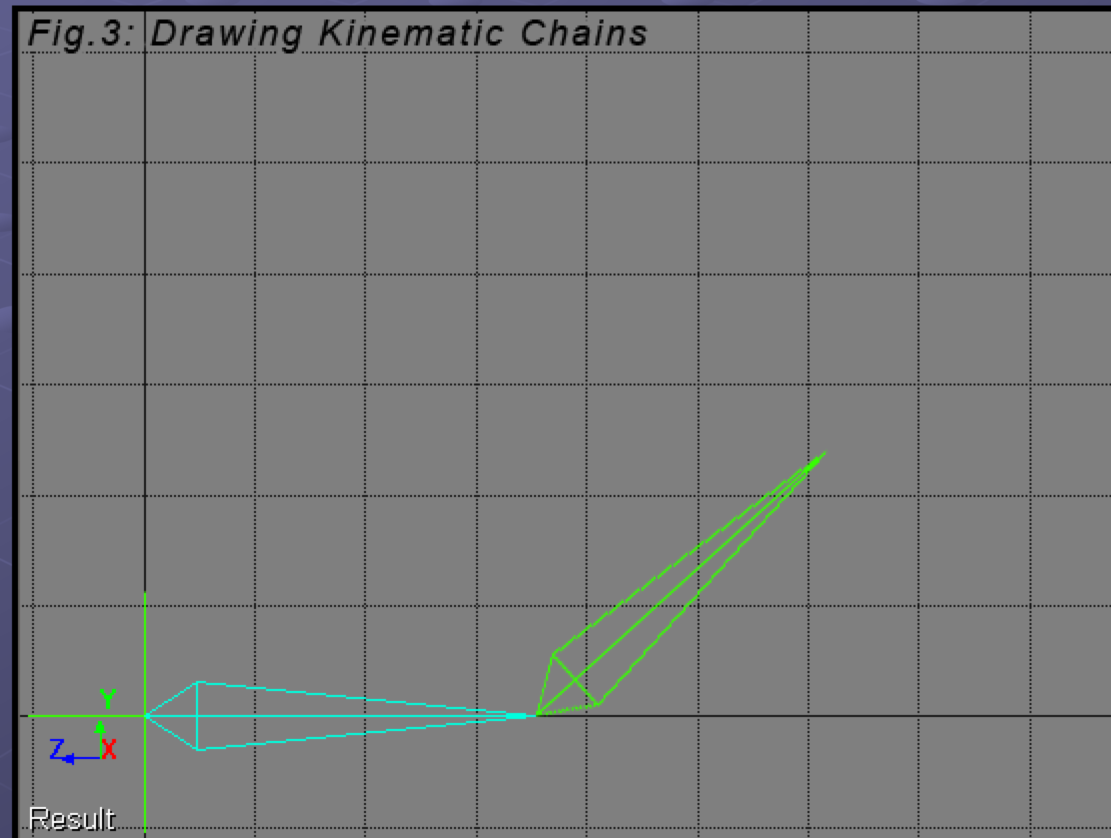
Drawing Kinematic Chains

- Starting with the effector's object:
  - Translate by length
  - Rotate by angle



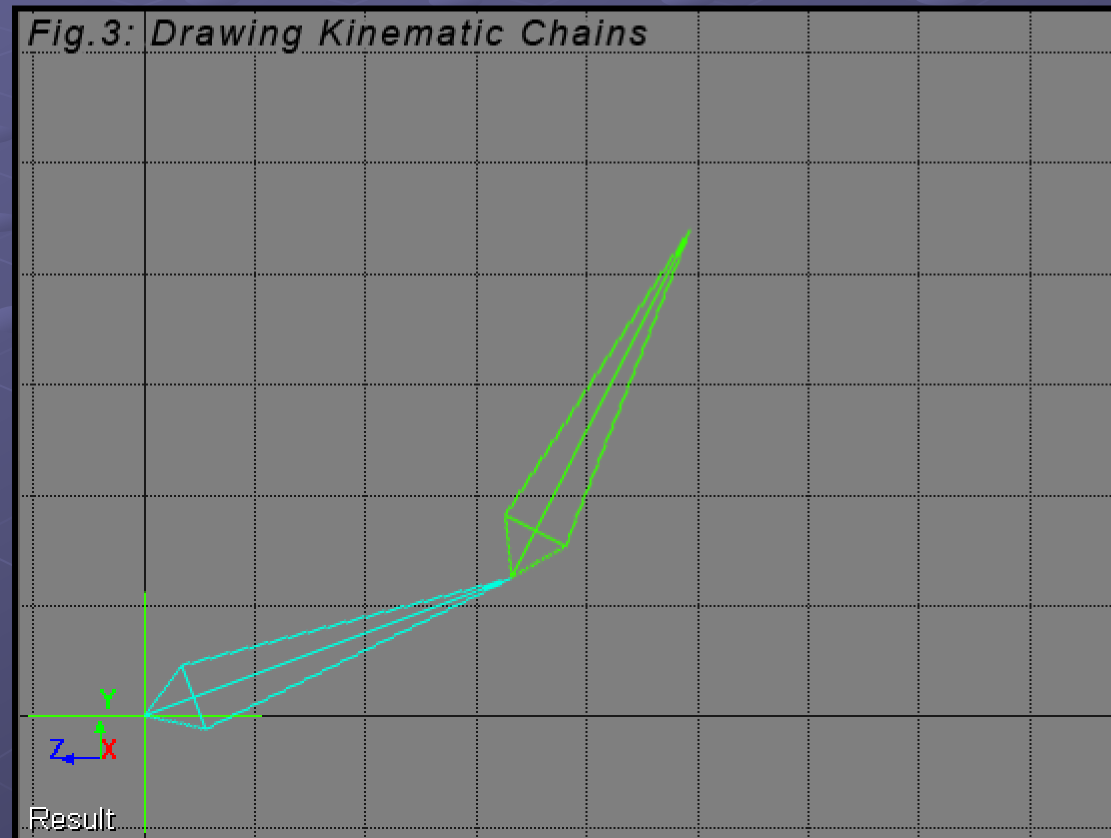
Drawing Kinematic Chains

- Starting with the effector's object:
  - Now translate by the length of the next link…



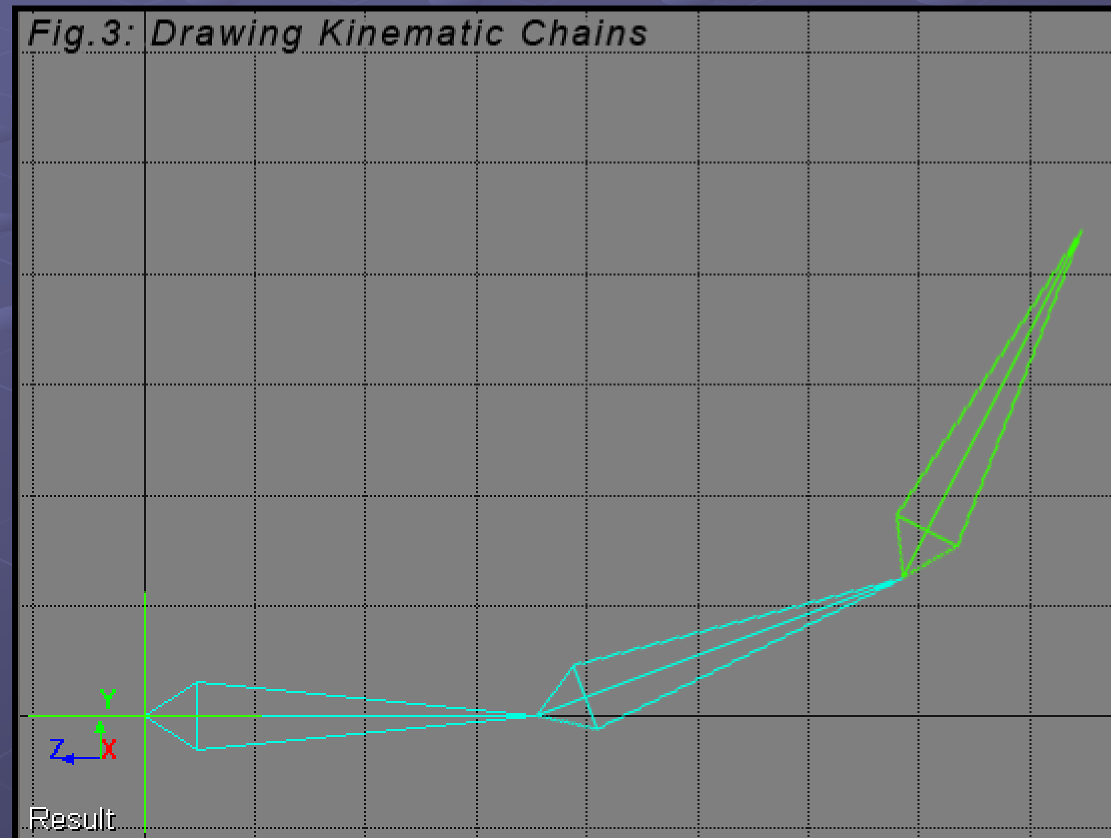
Drawing Kinematic Chains

- Starting with the effector's object:
  - Now translate by the length of the next link…
  - … and rotate the entire chain by the angle of that link



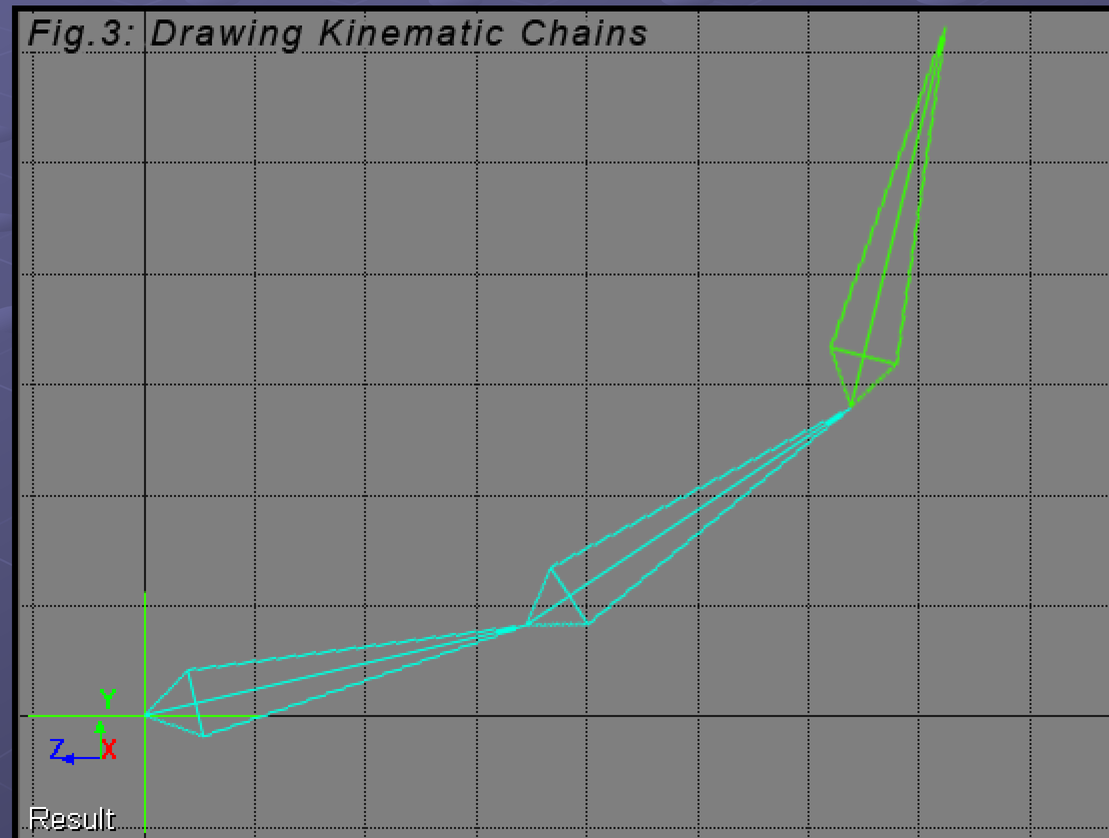
Drawing Kinematic Chains

- Starting with the effector's object:
  - Translate again by the length of the next link in the chain…



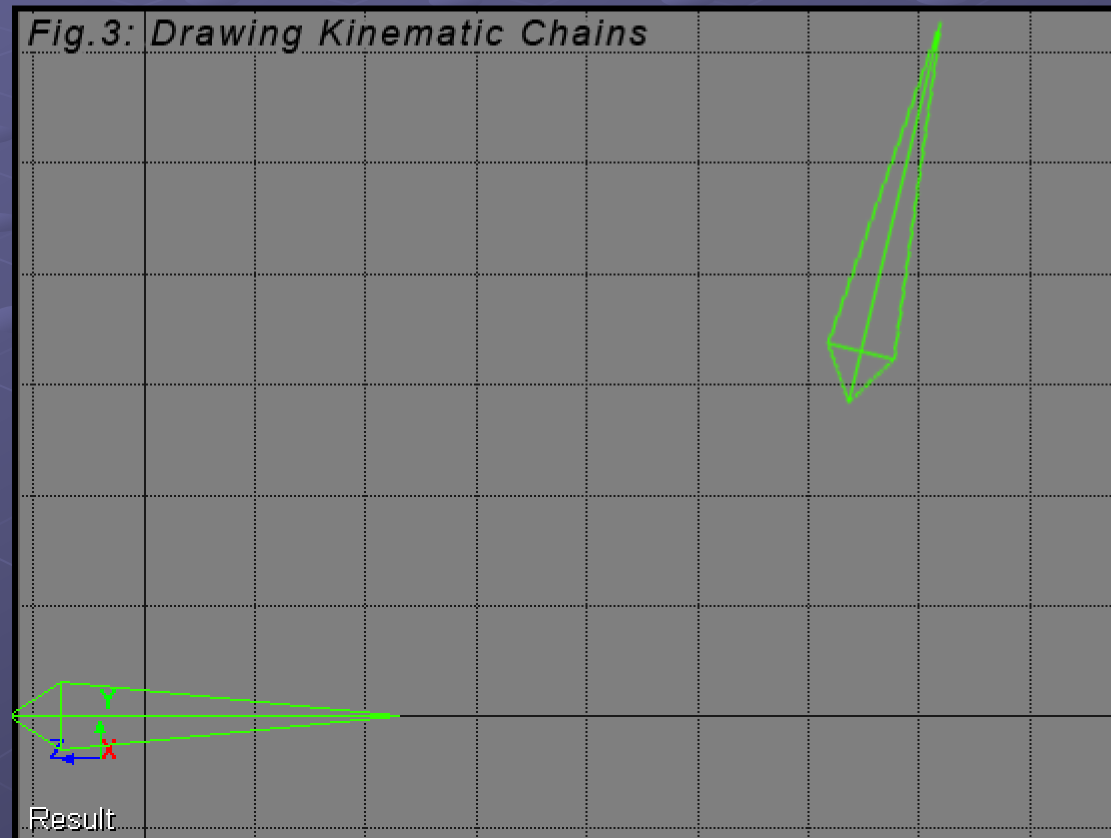
Drawing Kinematic Chains

- Starting with the effector's object:
  - Translate again by the length of the next link in the chain…
  - … and rotate the entire chain by that link's angle



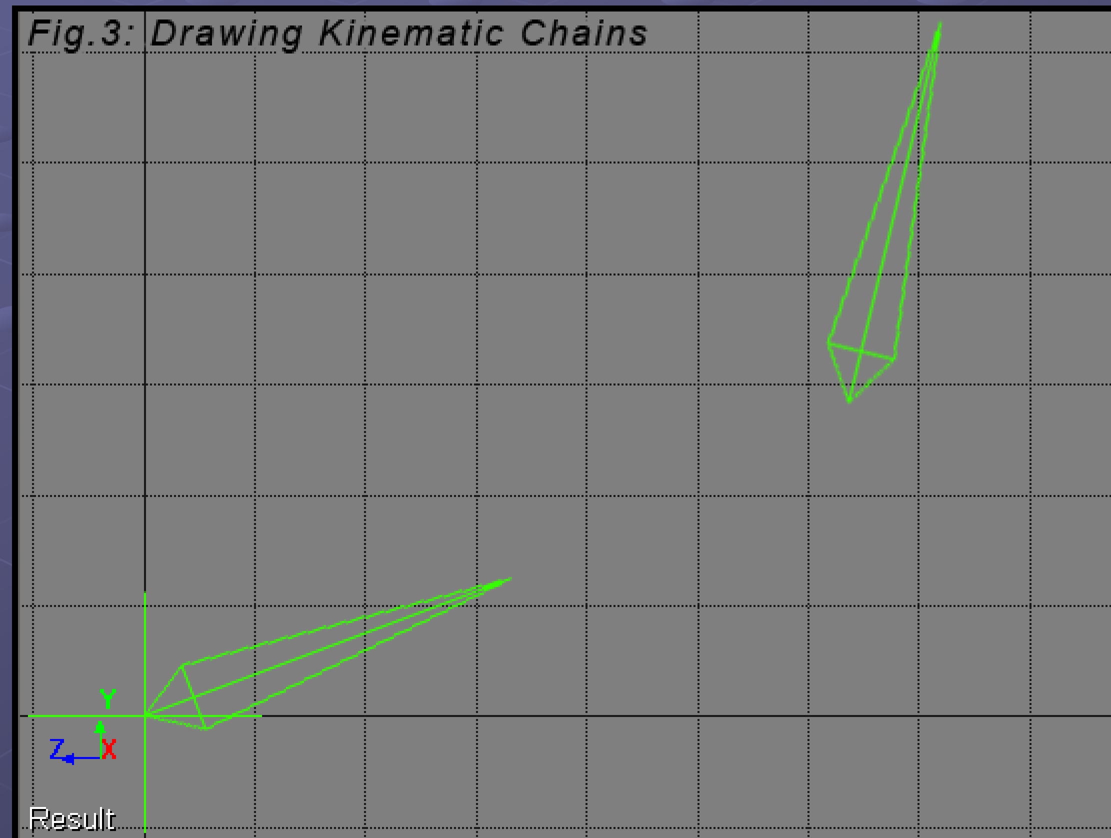
Drawing Kinematic Chains

- Starting with the NEXT link's object:



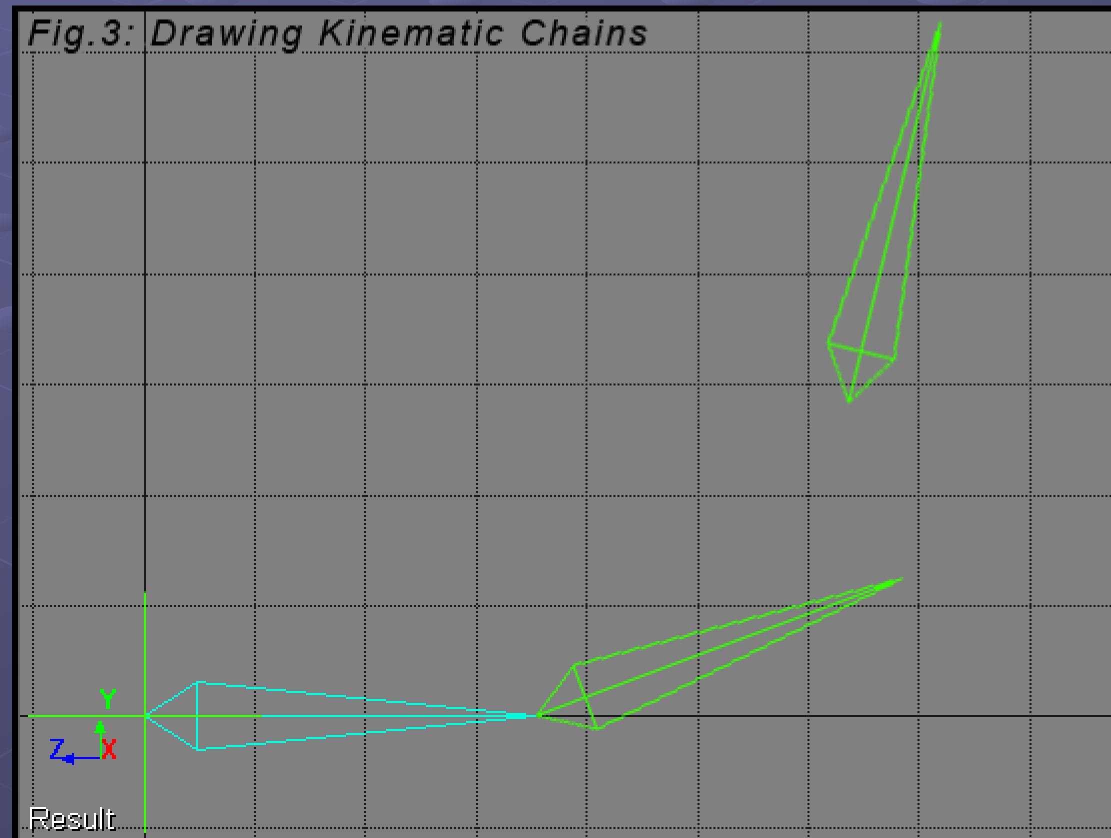
Drawing Kinematic Chains

- Starting with the NEXT link's object:
  - Translate by the object's length…
  - Rotate by the object's angle.



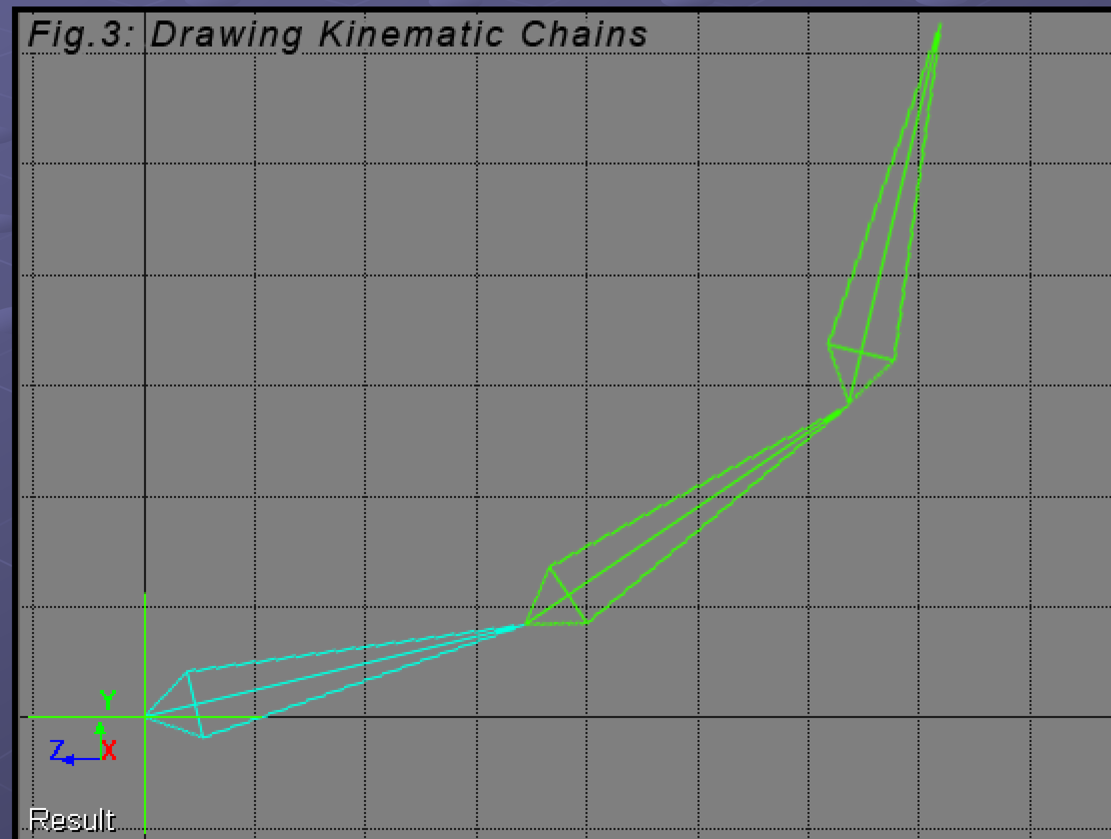
Drawing Kinematic Chains

- Starting with the NEXT link's object:
  - Translate by the next object's length…



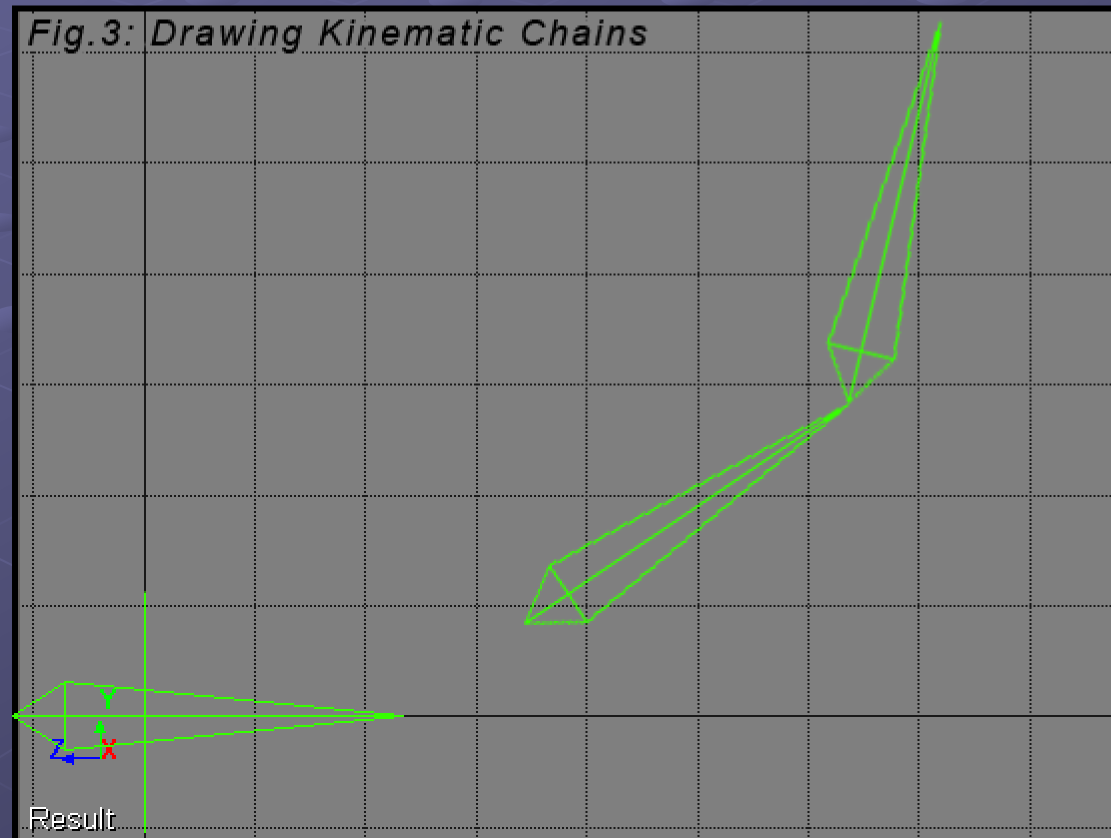
Drawing Kinematic Chains

- Starting with the NEXT link's object:
  - Translate by the next object's length…
  - And rotate the entire chain by that object's angle.



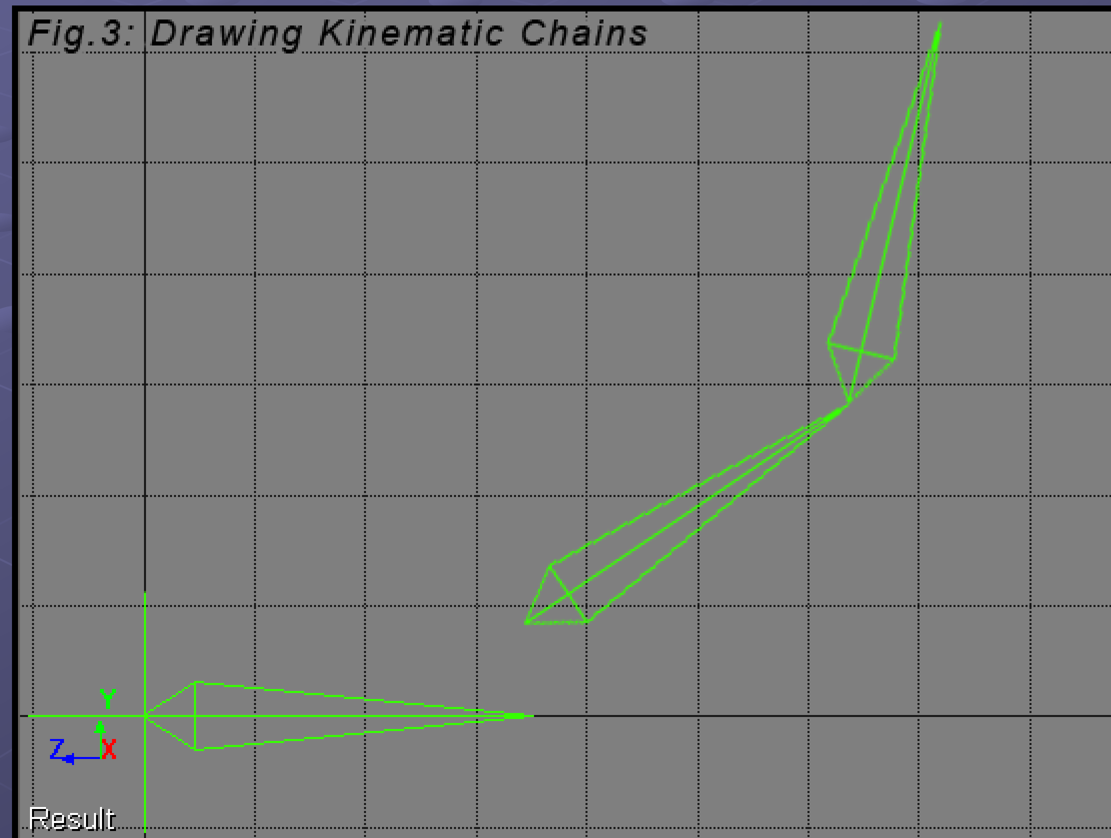
Drawing Kinematic Chains

- So on, and so forth…
  - Place the next link



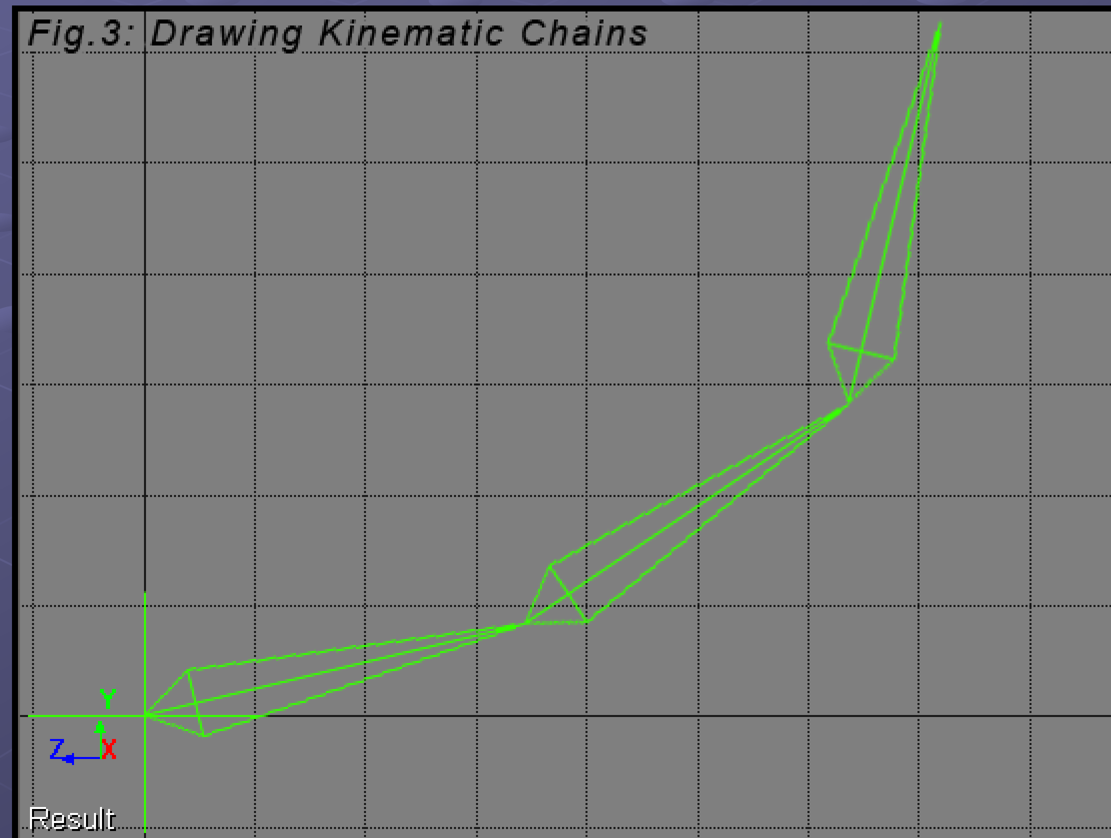
Drawing Kinematic Chains

- So on, and so forth…
  - Place the next link
  - translate



Drawing Kinematic Chains

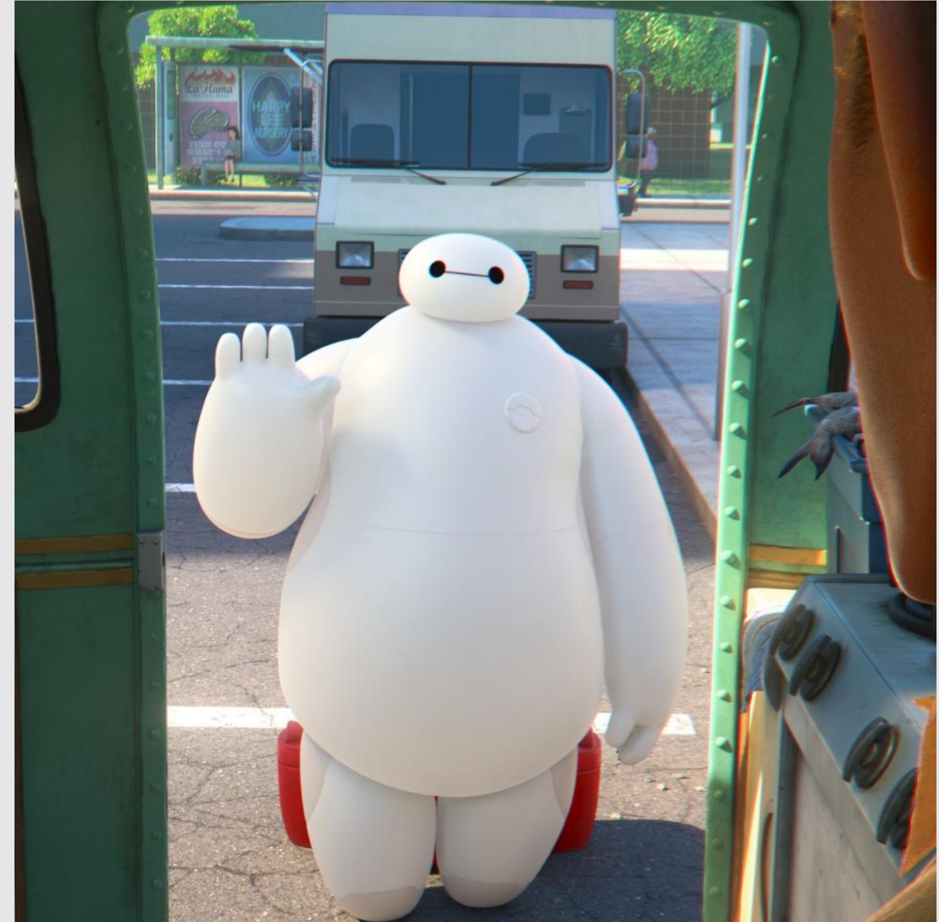
- So on, and so forth until chain is complete.
  - Place the next link
  - Translate
  - Rotate



Forward Kinematics

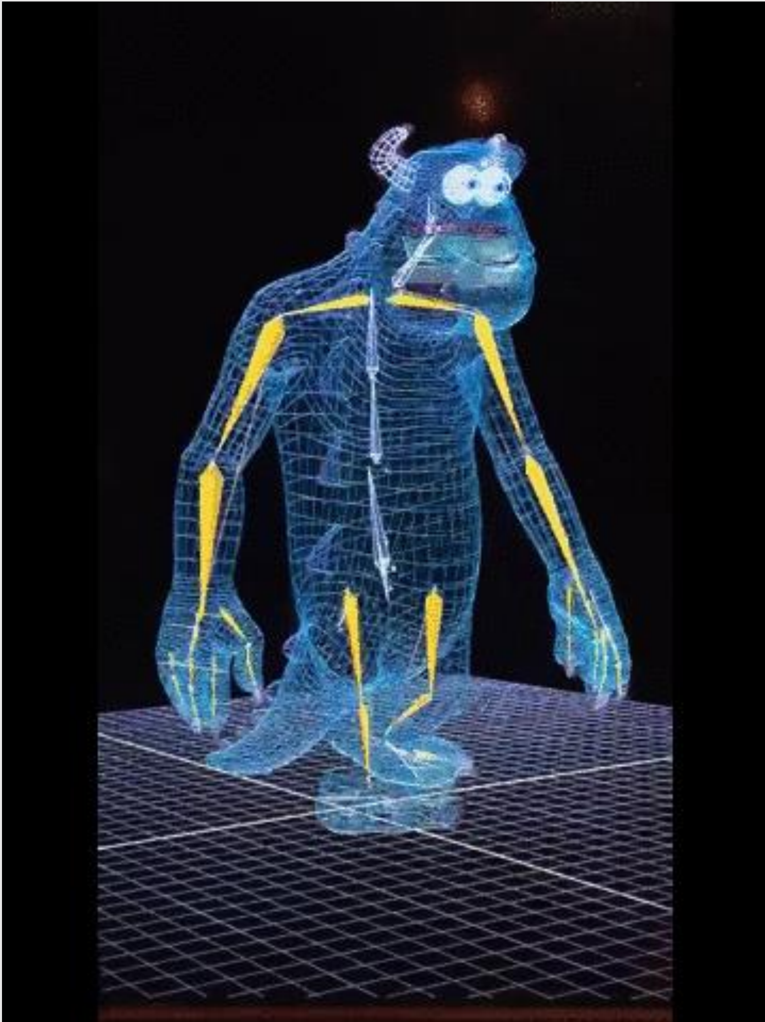
- [+] Computationally efficient
- [+] Easy interface to work with
- [+] Explicit control over every joint
- [-] Produces rigid animations
- [-] Hard to model real-world motions
- [-] Requires more keyframes

- Results often look robot-like



Big Hero 6 (2014) Disney

Linear Blend Skinning


Monster's Inc (2001) Pixar

- Vertices track with bones
  - Known as blend skinning
- For each vertex i, compute weights w_{ij} for each bone j
  - Weights are normalized for each vertex

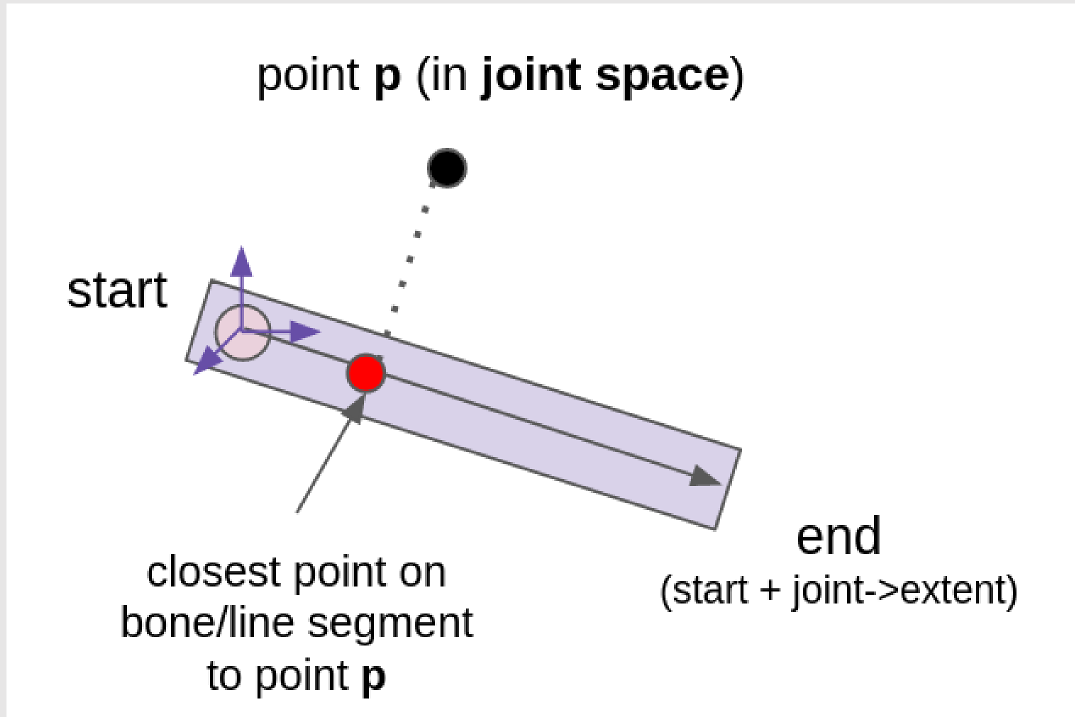
$$\sum_j w_{ij} = 1$$

  - Weights average transforms of each bone to compute posed vertex position v'_i from bind vertex v_i

$$v'_i = \sum_j (w_{ij} P_j B_j^{-1}) v_i$$

    - P_j is bone j's bone-to-pose transform
    - B_j is bone j's bone-to-bind transform
      - It should type-check :)

Computing Weights



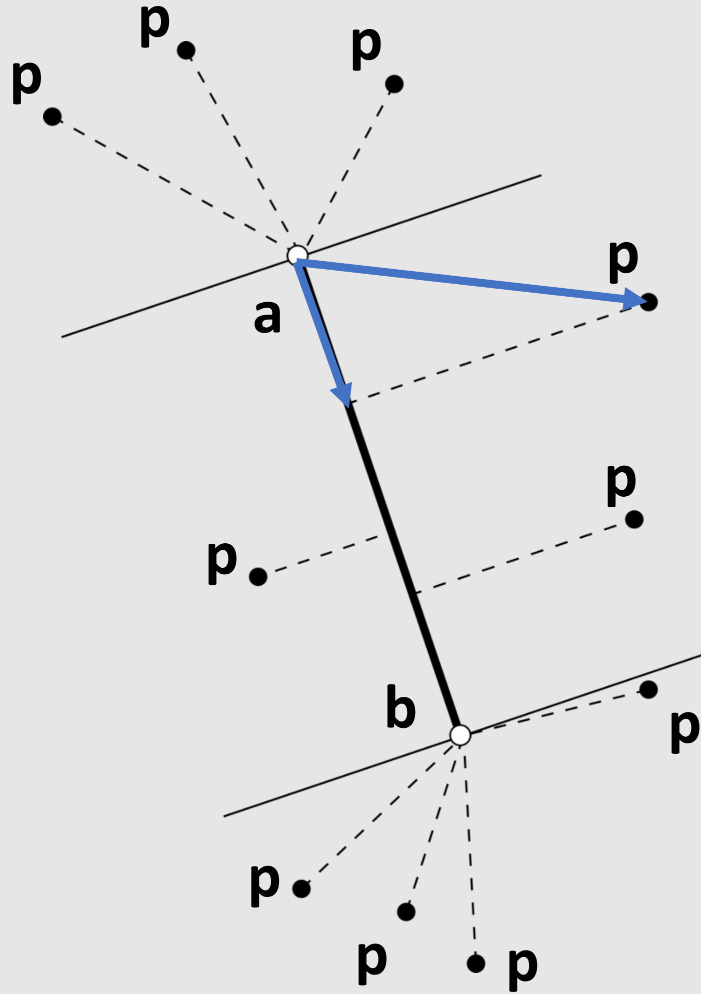
- r is the radius of the bone
- d_{ij} is the distance between v_i and its closest projection onto the bone

$$\hat{w}_{ij} = \frac{\max(0, r - d_{ij})}{r}$$

- Make sure to normalize weights

$$w_{ij} = \frac{\hat{w}_{ij}}{\sum_j \hat{w}_{ij}}$$

Review: Closest Point on a Line Segment



Compute the vector **p** from the line base **a** along the line

$$\langle \mathbf{p} - \mathbf{a}, \mathbf{b} - \mathbf{a} \rangle$$

Normalize to get a time

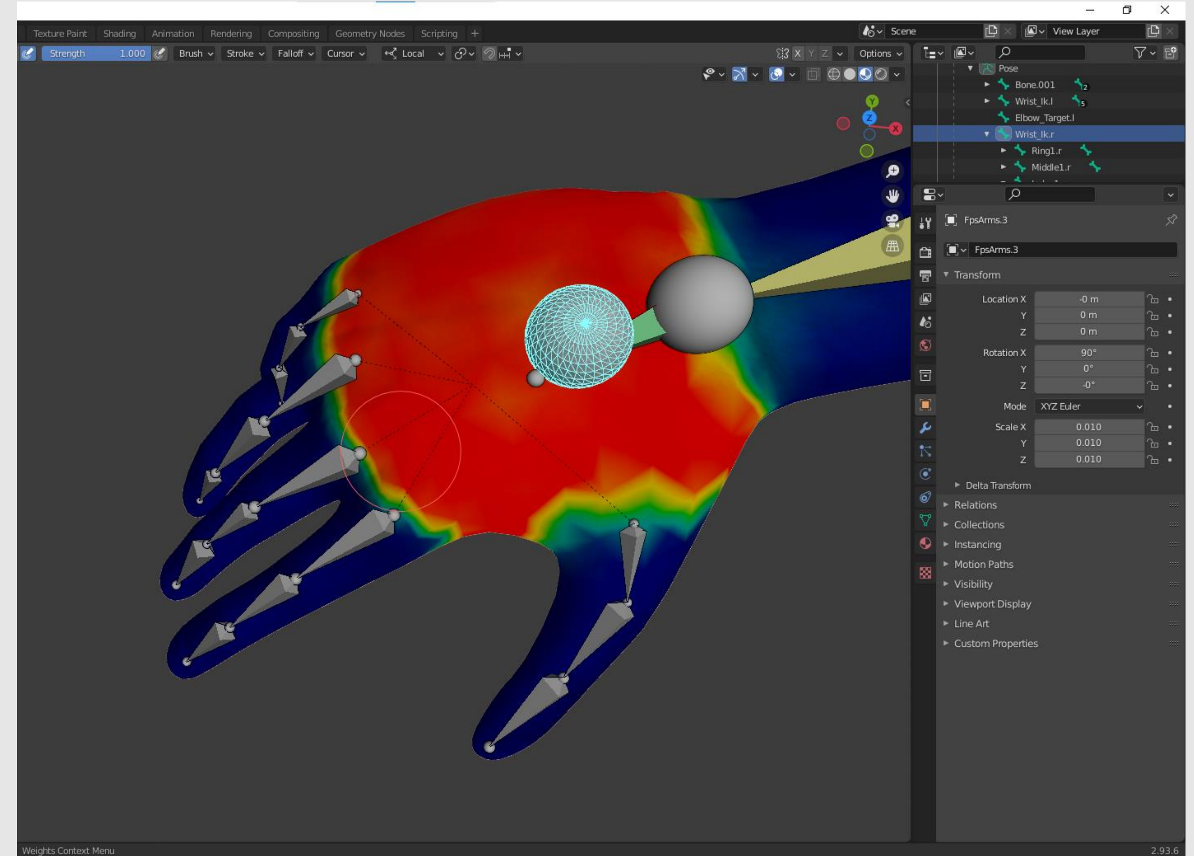
$$t = \frac{\langle \mathbf{p} - \mathbf{a}, \mathbf{b} - \mathbf{a} \rangle}{\langle \mathbf{b} - \mathbf{a}, \mathbf{b} - \mathbf{a} \rangle}$$

Clip time to range [0,1]and interpolate

$$\boldsymbol{a} + (\mathbf{b} - \mathbf{a})t$$

Weight Painting

- Computer animation applications allow you to specify weights on your own
  - Known as **weight painting**
- UI uses color to illustrate magnitude of each vertex/bone pair
- Part of the rigging pipeline

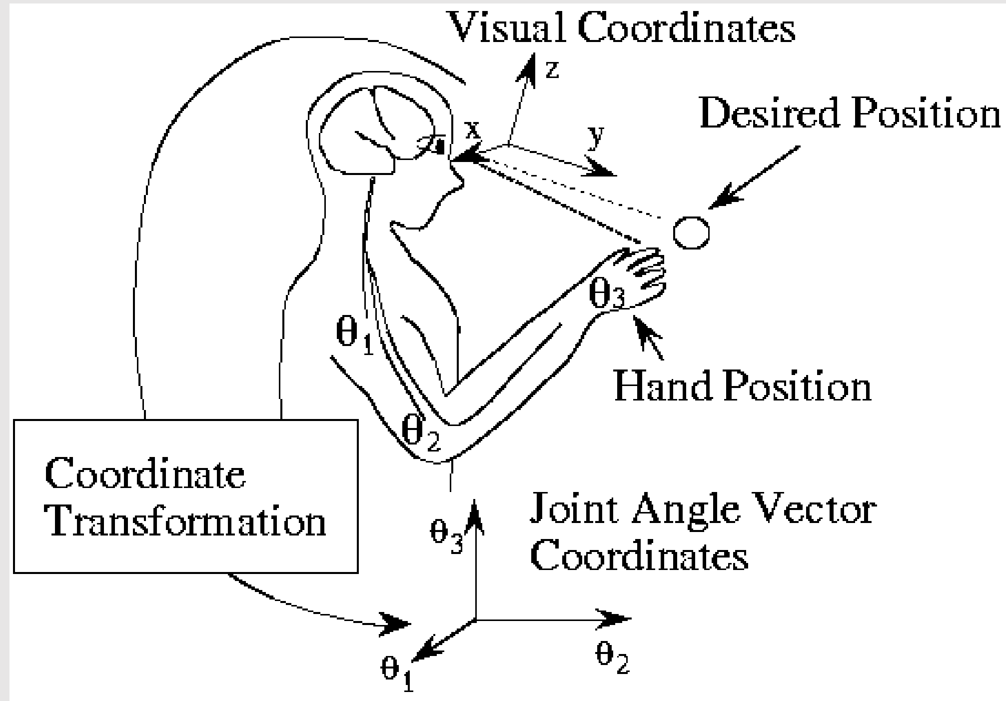


Blender (2021) Ton Roosendaal

- ~~Forward Kinematics (review)~~

- Inverse Kinematics

How Humans Move



- We don't think about the movement of each individual joint
  - Instead, we think about a part of our body, and where we want it to go
    - Our body solves for the correct movements
    - **Ex:** hand moves to reach a doorknob
- **No unique solution**
  - Many ways to catch a ball
- What if our rig behaved a similar way…

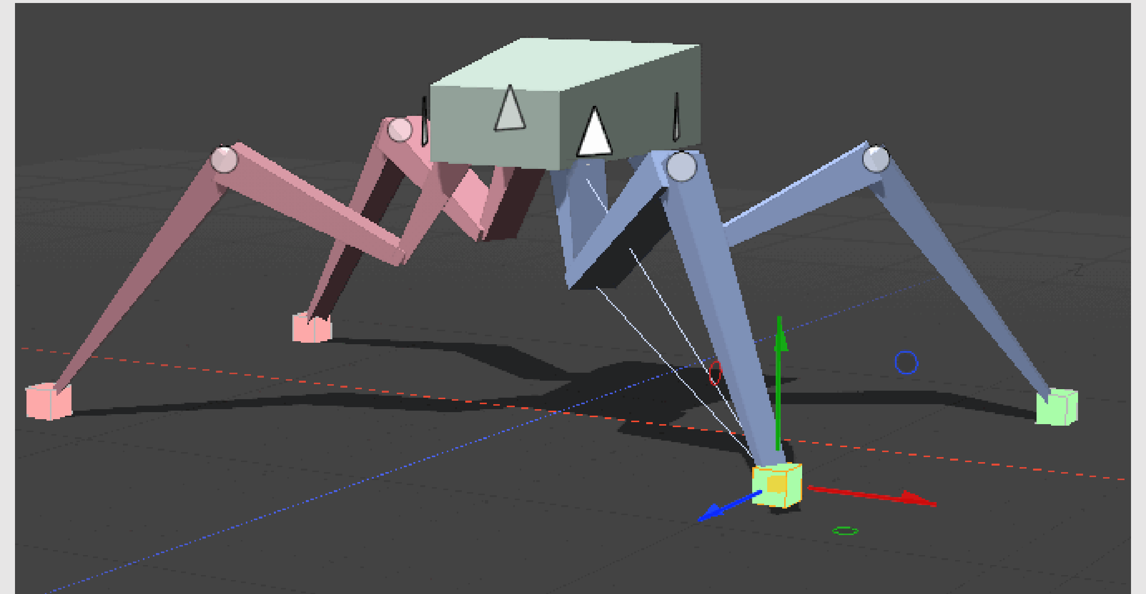
Inverse Kinematics

- Identify a bone on the rig i and a handle h that it should reach for
    - Can try to satisfy multiple targets (i, h)
- Loss function $f(q)$ for rig configuration q is:

$$f(q) = \sum_{(i,h)} \frac{1}{2} |p_i(q) - h|^2$$

    - Where $p_i(q)$ is the position of the end of bone i
- **Goal:** compute the gradient $\nabla f(q)$
    - Gradient represents how changing each joint will change the loss function
    - Apply gradient descent with some timestep τ:

$$q = q \; - \tau \, \nabla f(q)$$



Foundry (2020) Foundry Hub

Inverse Kinematic Gradient

$$\frac{df}{d\theta_k^y} = \frac{d}{d\theta_k^y}\sum_{(i,h)}\frac{1}{2}|p_i(q) - h|^2$$

Take gradient with respect to function

$$\frac{df}{d\theta_k^y} = \sum_{(i,h)}(p_i(q) - h)\frac{dp_i}{d\theta_k^y}$$

Expand p_i into transformations. Each rotation in 3D is axis-aligned

$$\frac{dp_i}{d\theta_k^y} = \frac{d}{d\theta_k^y}\left[\prod_{j=0,i-1} R(\theta_j^z)R\left(\theta_j^y\right)R(\theta_j^x)T(u_j)\right] R(\theta_i^z)R\left(\theta_i^y\right)R(\theta_i^x)u_i$$

Gradient breaks down into 3 parts:

$$\frac{dp_i}{d\theta_k^y} = R(\theta_0^z)R\left(\theta_0^y\right)R(\theta_0^x)T(u_0)\ldots R(\theta_k^z)\frac{d}{d\theta_k^y}R\left(\theta_k^y\right)R(\theta_k^x)T(u_i)\ldots R(\theta_i^z)R\left(\theta_i^y\right)R(\theta_i^x)u_i$$

[linear transformation] **[derivative]** **[transformed point]**

Inverse Kinematic Gradient

$$\frac{dp_i}{d\theta_k^y} = ???$$

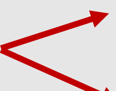
Fun fact: by transforming the axis of rotation and base point to local coordinates, Then the derivative of the rotation $R(\theta_k^y)$ by amount θ_k^y around axis y and center r of point p becomes:

$$\frac{dp_i}{d\theta_k^y} = y \times (p - r)$$

constant for a given handle

$p =$ **[linear transformation] [$R(\theta_k^y)$] [transformed point]**

$r =$ **[linear transformation'] [0,0,0,1]**



$y =$ **[linear transformation'] [$R(\theta_k^z)$] [0,1,0,0]**

specific to the current joint

[linear transformation'] = all rotations and transformations up to, but not including the kth bone

Inverse Kinematic Gradient

- Note: all joints that come before joint k can also contribute to the movement of joint k
  - **Example:** moving your shoulder moves your hand
- Need to also compute how every joint prior to joint k affects the movement of joint k
  - Gives us a gradient for each joint in range [0 - k]

$$\nabla f_k^y = (p_i(q) - h) \cdot [y_k \times (p_i(q) - r_k)]$$

$$\nabla f_{k-1}^y = (p_i(q) - h) \cdot [y_{k-1} \times (p_i(q) - r_{k-1})]$$

$$\nabla f_{k-2}^y = (p_i(q) - h) \cdot [y_{k-2} \times (p_i(q) - r_{k-2})]$$

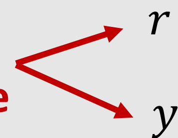
$$\dots$$

$$\nabla f_0^y = (p_i(q) - h) \cdot [y_0 \times (p_i(q) - r_0)]$$

constant for a given handle

p

r



y

specific to the current joint

Inverse Kinematic Gradient

- Each joint k will have its own vector gradient $\frac{df}{d\theta_k} = < \frac{df}{d\theta_k^x}, \frac{df}{d\theta_k^y}, \frac{df}{d\theta_k^z} >$
  - Same process for computing each component, just use x_k , y_k , or z_k
- What if we have multiple target pairs (i, h)?
  - Gradient becomes a sum!

$$\nabla f_k^y \mathrel{+}= (p_i(q) - h) \cdot [y_k \times (p_i(q) - r_k)]$$

$$\nabla f_{k-1}^y \mathrel{+}= (p_i(q) - h) \cdot [y_{k-1} \times (p_i(q) - r_{k-1})]$$

$$\nabla f_{k-2}^y \mathrel{+}= (p_i(q) - h) \cdot [y_{k-2} \times (p_i(q) - r_{k-2})]$$

…

$$\nabla f_0^y \mathrel{+}= (p_i(q) - h) \cdot [y_0 \times (p_i(q) - r_0)]$$

Inverse Kinematic Gradient

```
vec3 gradient_in_current_pose() {

  for (auto &handle : handles) {

    Vec3 h = handle.target;
    Vec3 p = // TODO: compute output point

    // walk up the kinematic chain
    for (BoneIndex b = handle.bone; b < bones.size(); b = bones[b].parent) {
      Bone const &bone = bones[b];
      Mat4 xf = // TODO: compute [linear transform']

      Vec3 r = xf * Vec3{0.0f, 0.0f, 0.0f};

      Vec3 x = // TODO: compute bone's x-axis in global (world) space
      Vec3 y = // TODO: compute bone's y-axis in global (world) space
      Vec3 z = // TODO: compute bone's z-axis in global (world) space

      gradient[b].x += dot(cross(x, p - r), p - h);
      gradient[b].y += dot(cross(y, p - r), p - h);
      gradient[b].z += dot(cross(z, p - r), p - h);
    }
  }
}
```

Inverse Kinematic Gradient

- How do we apply the gradient?
  - Iterate through each joint j and apply ∇f_j
  - Make sure to clear all gradients after each step!

$$\theta_j = \theta_j - \tau \nabla f_j$$

- Recompute the loss function

$$f(q) = \sum_{(i,h)} \frac{1}{2} |p_i(q) - h|^2$$

- If loss is lower than some threshold, terminate
  - Otherwise continue until max steps exceeded