Advanced Animation Programming

GPR-450
Daniel S. Buckstein

Hierarchies & Skeletal Animation: Inverse Kinematics
Week 7

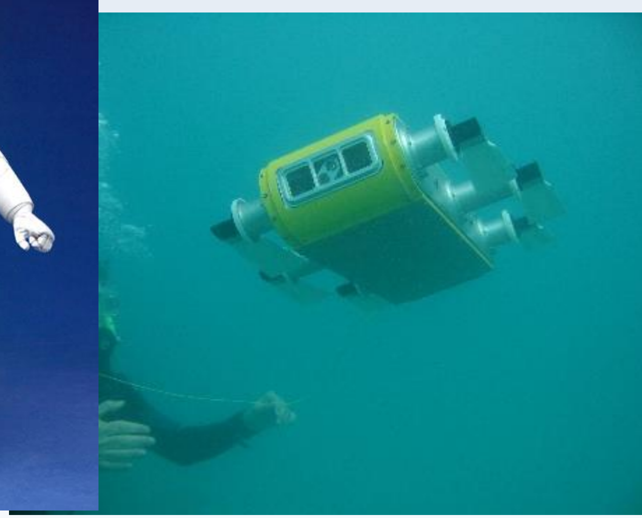
License

Skeletal Animation

- Inverse kinematics
  - Intro
  - Fundamental formulas
  - Unconstrained IK
    - Garden hose
  - Constrained IK, geometric approach
    - Magic triangles: method by Dan (haven’t seen elsewhere)

Inverse Kinematics

- Real-world problems:
- Robots: generally goal-oriented
- Critical motions need to be *perfect*



Inverse Kinematics

- FK vs. IK:

Forward kinematics (FK): we know all transformations ***between*** *root* and *goal*, therefore we know the transformation of *goal*

- Controller rotates joints directly, affecting orientation of child joints

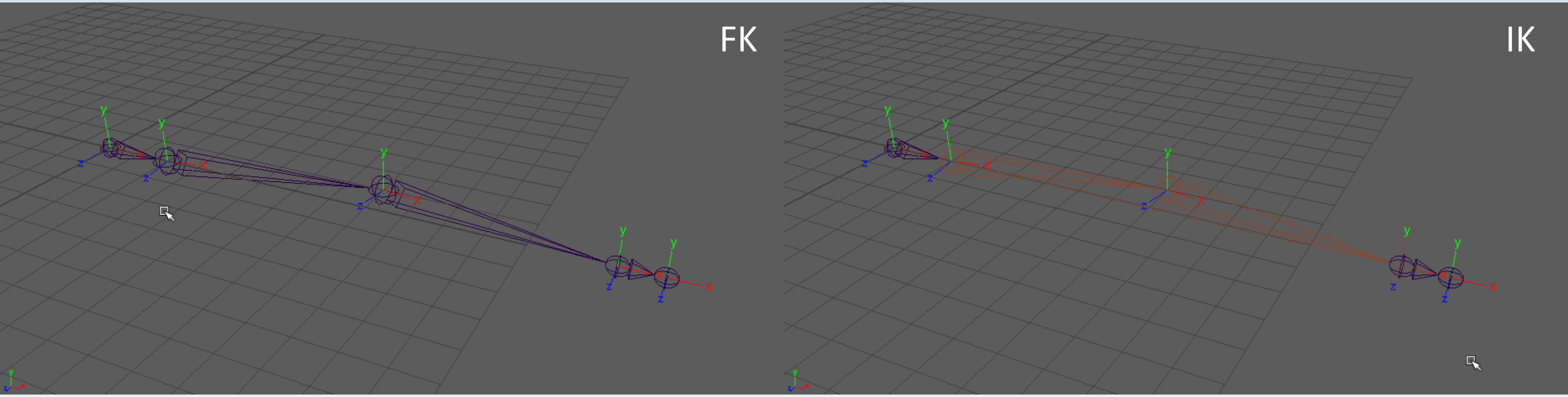
Inverse kinematics (IK): given ***only*** the *root* and the *goal*, determine the in-betweens!

- Controller changes the position of the *end effector*, influencing all of the joints between the base and the end

http://www.cs.cmu.edu/~./15464-s13/lectures/lecture6/IK.pdf

Inverse Kinematics

- FK vs. IK (Autodesk Maya):



- In terms of animating, IK is easier and faster…
- …but it's a computational nightmare… why???

Inverse Kinematics

- ***Inverse kinematics solvers***:
- Ultimately, our animation is defined by the final result (skinning, sprites)
- …and ultimately, the final result is determined using *forward kinematics* (FK)
- Therefore, *inverse kinematics* (IK) must be related to FK…

Inverse Kinematics

- ***Inverse kinematics solvers***:
- The job of an *inverse kinematics* solver:
- *Determine* ***local poses*** *such that FK will yield our end effectors'* ***world transforms***
- Corollary: IK is used to *control* FK to give us our desired output
- Here's a diagram (or two)…

Inverse Kinematics

- Our current FK-based solution:

1. Manually select or set local joint poses

↓

2. Convert local pose to local transformation

↓

3. Calculate global transformations with FK

$$\mathrm{pose}_{n,t}$$

↓

$$^{\mathrm{parent}}T_{n_t} = \mathrm{convert}\left(\mathrm{pose}_{n,t}\right)$$

↓

$$^{\mathrm{world}}T_{n_t} = {}^{\mathrm{world}}T_{\mathrm{parent}_t}\,{}^{\mathrm{parent}}T_{n_t}$$ (for all joints)

Inverse Kinematics

- ***Inverse kinematics-based solution***:

1. Set end effectors' global transforms

$^{\text{world}}T_{n_t}$ for all *effectors* → effectorList_t

2. ***Calculate local poses (this is the IK problem)***

$$\text{localPoseList}_t = \mathbf{solveIK}(\text{effectorList}_t\ , \ldots)$$

3. ***Select local pose from IK solution***, use to do FK

$$\text{pose}_{n,t}$$

4. Convert local pose to local transformation

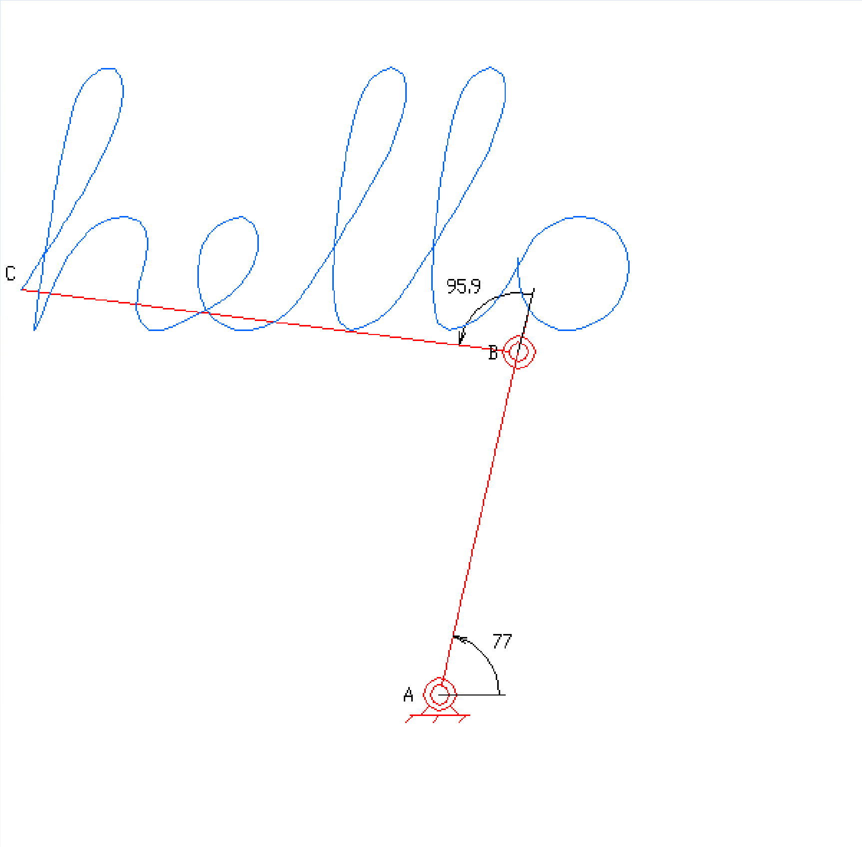
$$^{\text{parent}}T_{n_t} = \text{convert}\left(\text{pose}_{n,t}\right)$$

5. Calculate global transformations with FK

$$^{\text{world}}T_{n_t} = {}^{\text{world}}T_{\text{parent}_t}\,{}^{\text{parent}}T_{n_t}$$ (for all joints)

Inverse Kinematics

- 2D example (with solution displayed):



- End effector follows the path ("hello")
- Joint angles (about Z axis) determined to achieve the goal of the end effector following the path
- Angles are pose data for revolute joints

https://en.wikipedia.org/wiki/SCARA

Forward Kinematics

- FK math (to compute *global* transforms):

Root node:

- ${}^{\text{world}}T_n$ (already known)

Everything else:

known known

- ${}^{\text{world}}T_n = {}^{\text{world}}T_{\text{parent}} \; {}^{\text{parent}}T_n$

Unknown: solve by multiplying two things we do know!

Forward Kinematics

- **Forward kinematics transform solution:**
  - For each *node* in *hierarchy*
    - If node is root (parent index is -1)
      - Node's world transform ***is*** node's local transform
    - Else
      - Node's world transform
        = parent's world transform * node's local transform
- Assumes we know all *local transforms*
- Assumes we know parent's *global transform*

Inverse Kinematics

- IK math (to compute *local* transforms):

Root node:

- ${}^{\text{parent}}T_n$ (already known)

Everything else:

unknown known

- ${}^{\text{parent}}T_n = {}^{\text{parent}}T_{\text{world}}\ {}^{\text{world}}T_n$
- ${}^{\text{parent}}T_n = {}^{\text{world}}T_{\text{parent}}^{-1}\ {}^{\text{world}}T_n$

known

Unknown: solve by multiplying two things we do know!

Inverse Kinematics

- **Fundamentally, IK is opposite (solve *local*):**
  - For each *node* in *hierarchy*
    - If node is root (parent index is -1)
      - Node's local transform ***is*** node's world transform
    - Else
      - Node's local transform
        = parent's world transform ***inv.*** * node's world transform
- Assumes we know all *global transforms*
- Assumes we know parent's *global transform*

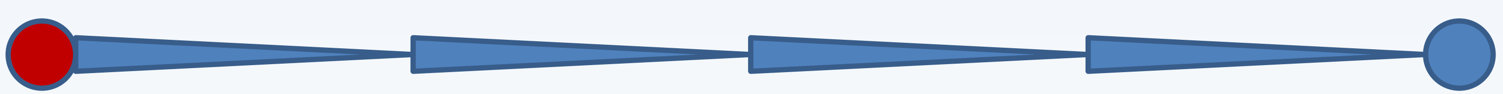
Inverse Kinematics

- The solution to an FK problem is depth-first traversal, multiplying matrices along the way
- For IK, different situations have different solutions
- An IK handle has a “solver” depending on the situation

Inverse Kinematics

- IK only requires two "***effectors***": spatial points
- The fixed effector is the root of the IK problem, a.k.a. ***base effector***
- The unfixed (user-controlled) effector is the *goal*, a.k.a. ***end effector***
- The set of joints we are solving are the ***chain***

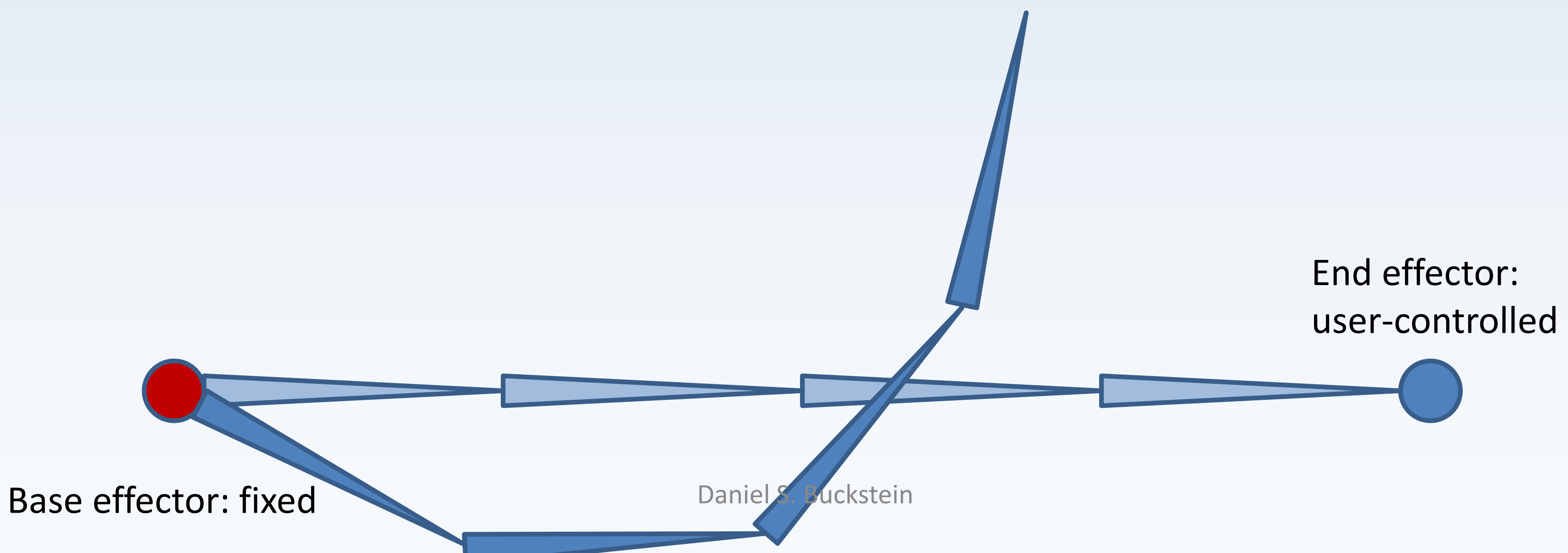
End effector:
user-controlled



Base effector: fixed*

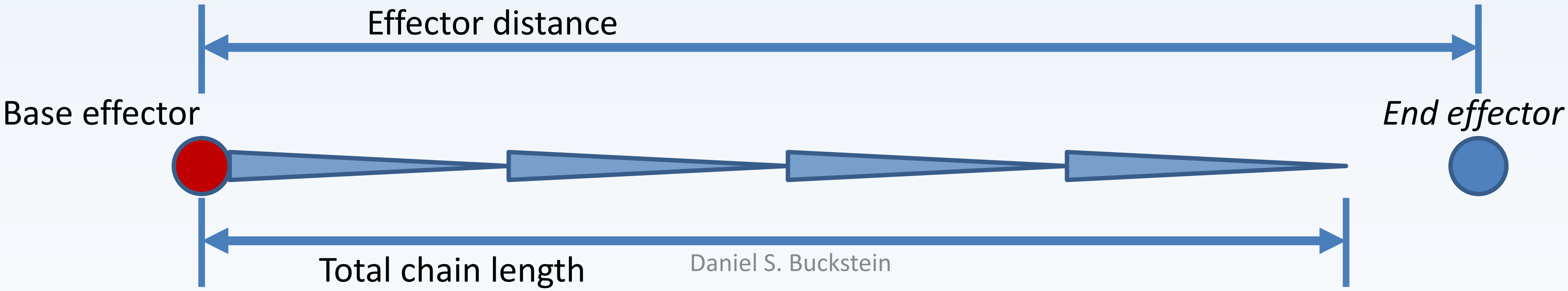
Inverse Kinematics

- We move the end effector…
- …and solve all of the in-betweens!



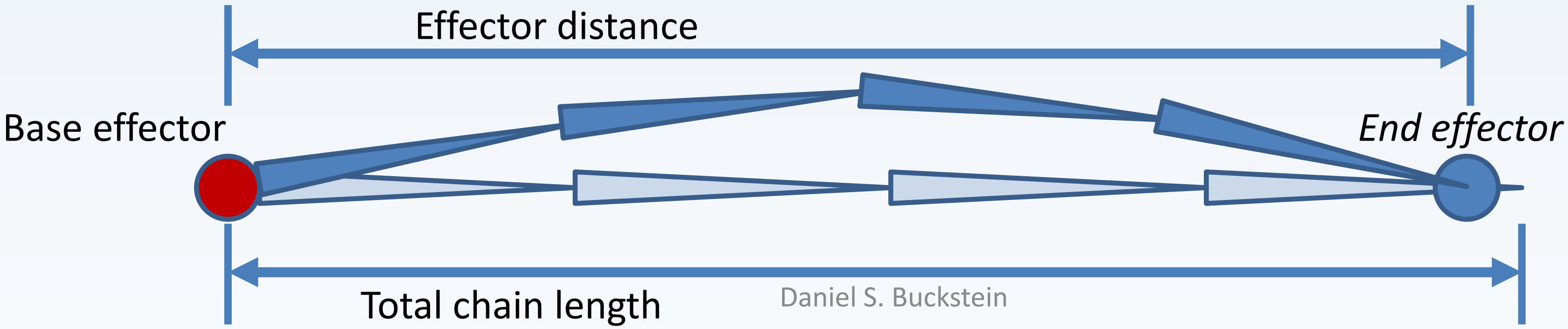
Inverse Kinematics

- Before we do *anything*… how do we know the IK system can even be solved???
- Total chain length vs. effector distance:
- If ***effector distance >= chain length***, then there is ***not*** a valid solution!
- Just solve as a straight line towards end!



Inverse Kinematics

- Before we do *anything*… how do we know the IK system can even be solved???
- Total chain length vs. effector distance:
- If ***effector distance < chain length***, then there is ***at least one*** valid solution!



Inverse Kinematics

- ***Unconstrained IK***: real-world applications?
  - https://youtu.be/wpYvejLhYPg
  - https://youtu.be/GzSVQGarHI0
  - https://youtu.be/euV1HmGm22s
- Hose: business end of hose is *end effector*, given some *locomotive behavior* such as "wander"; hydrant is *base effector* ☺
  - (somebody please do this in your game)

Inverse Kinematics: Garden Hose

- ***Unconstrained IK***: “Find the *best* solution to the chain.”
- Previous example: probably not the best solution…
- …but 4-chains are incredibly complex
- Generally we solve these problems with a “***Jacobian matrix***”:

Definitely not a topic for this course

(but feel free to look into it)

http://www.cs.cmu.edu/~./15464-s13/lectures/lecture6/IK.pdf

Inverse Kinematics : Garden Hose

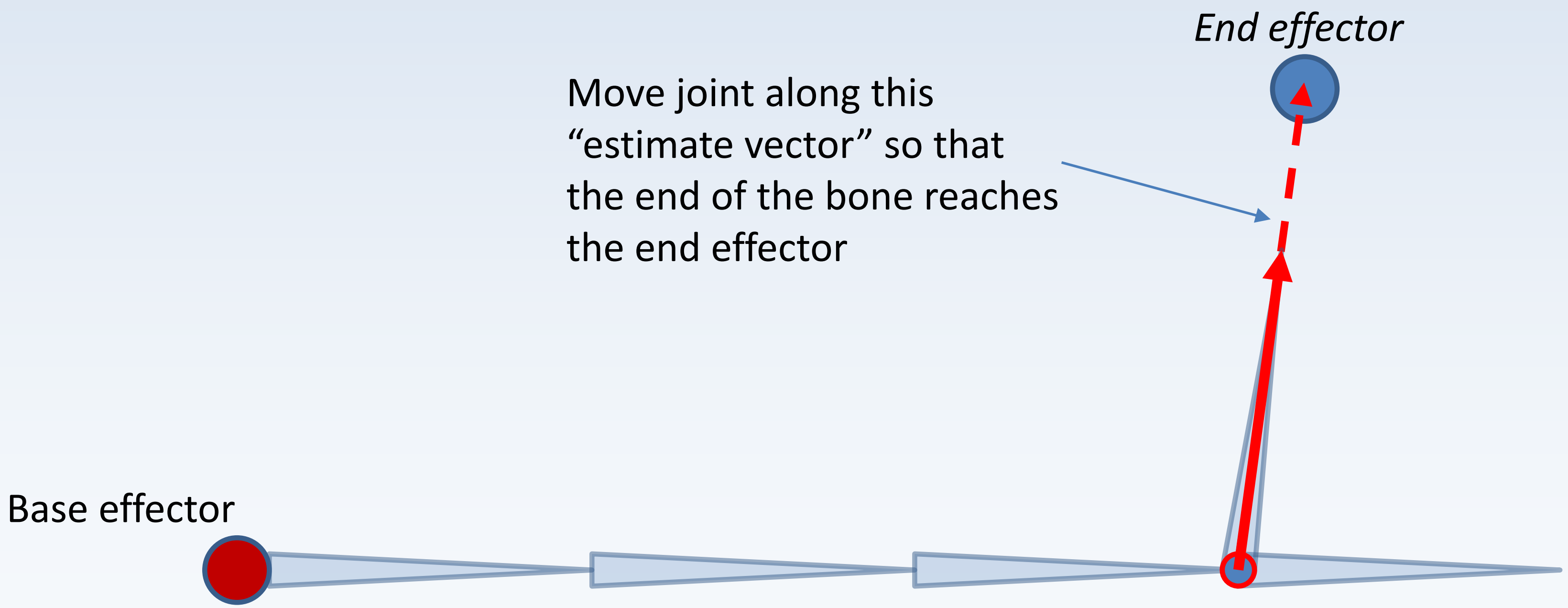
- ***Unconstrained IK***: faster approximations that use linear math instead of angles
- Basically move joints towards the effector
- Move back and forth along the chain until you find a solution that is “close enough”
- Visual example (the Jacobian method, *waaaaay* simplified)

Inverse Kinematics : Garden Hose

- ***Unconstrained IK***: Repeatedly aim joints towards the effectors and estimate if this is a good solution:

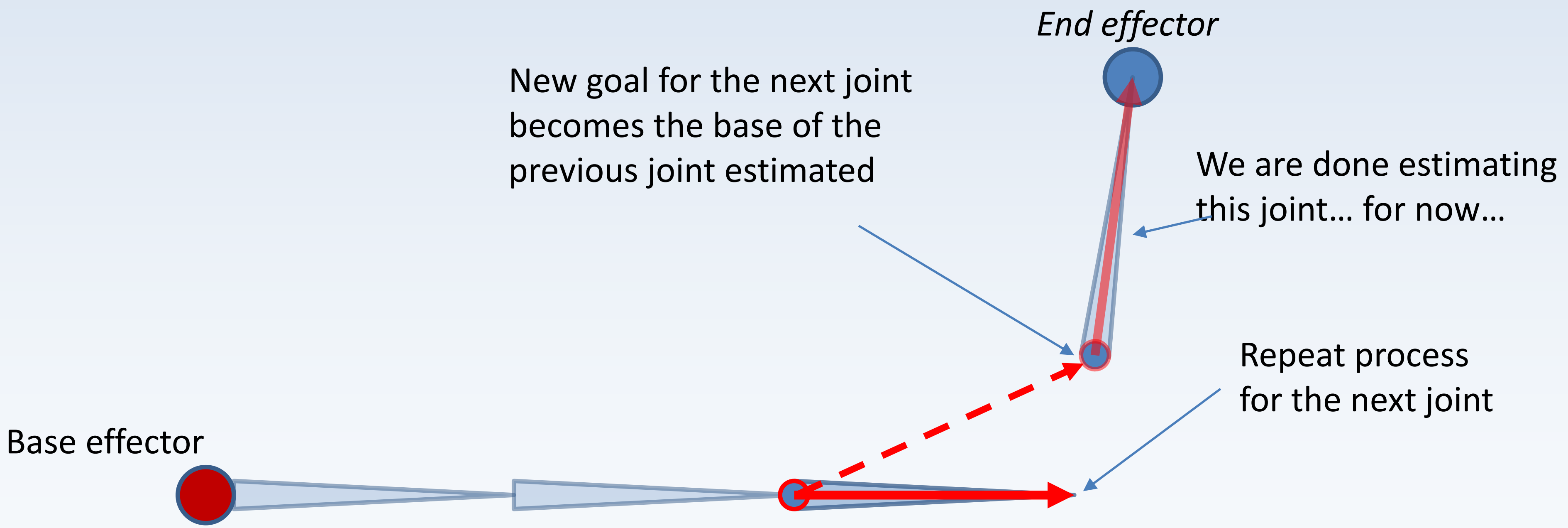
Inverse Kinematics : Garden Hose

- ***Unconstrained IK***: Repeatedly aim joints towards the effectors and estimate if this is a good solution:



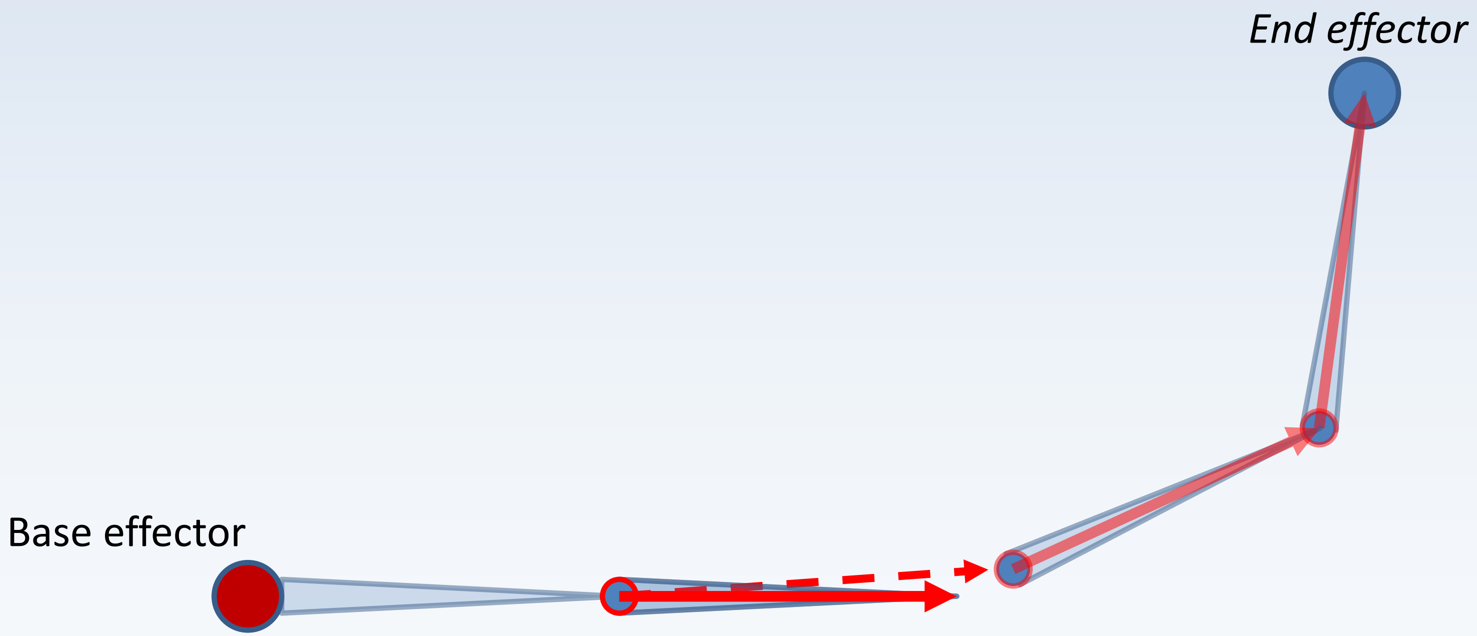
Inverse Kinematics : Garden Hose

- ***Unconstrained IK***: Repeatedly aim joints towards the effectors and estimate if this is a good solution:



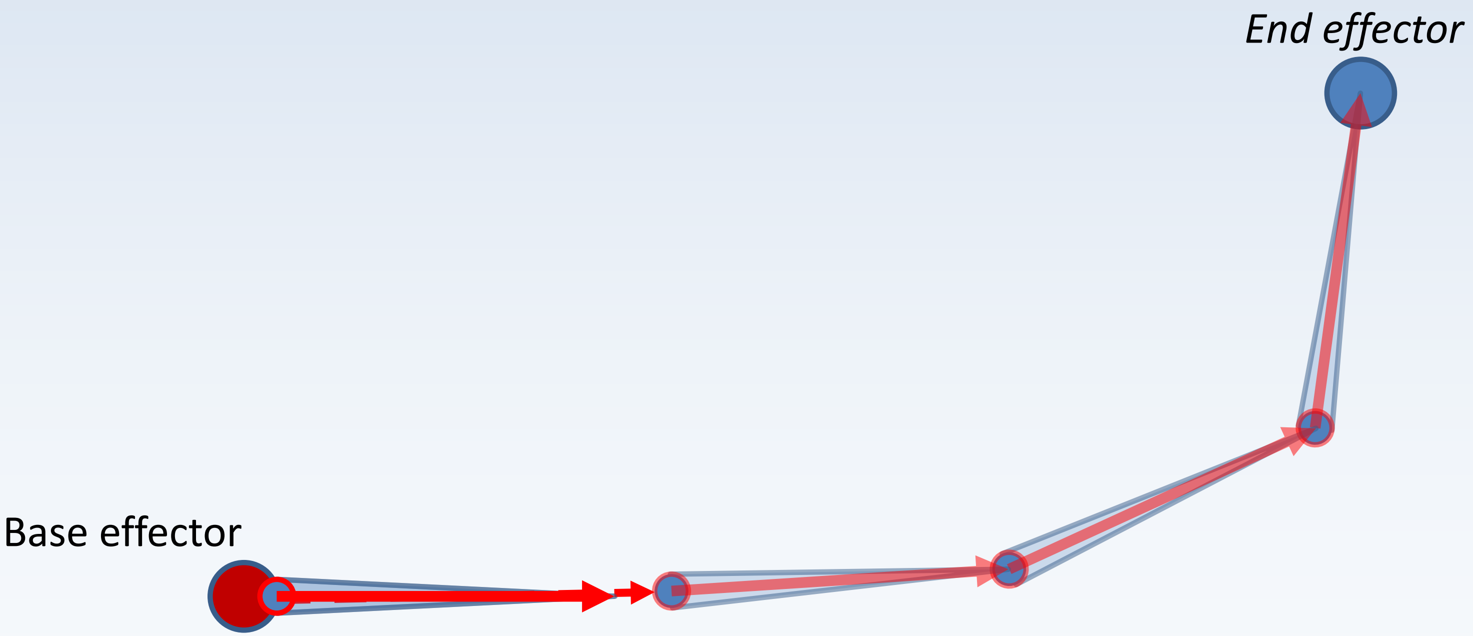
Inverse Kinematics : Garden Hose

- ***Unconstrained IK***: Repeatedly aim joints towards the effectors and estimate if this is a good solution:



Inverse Kinematics : Garden Hose

- ***Unconstrained IK***: Repeatedly aim joints towards the effectors and estimate if this is a good solution:

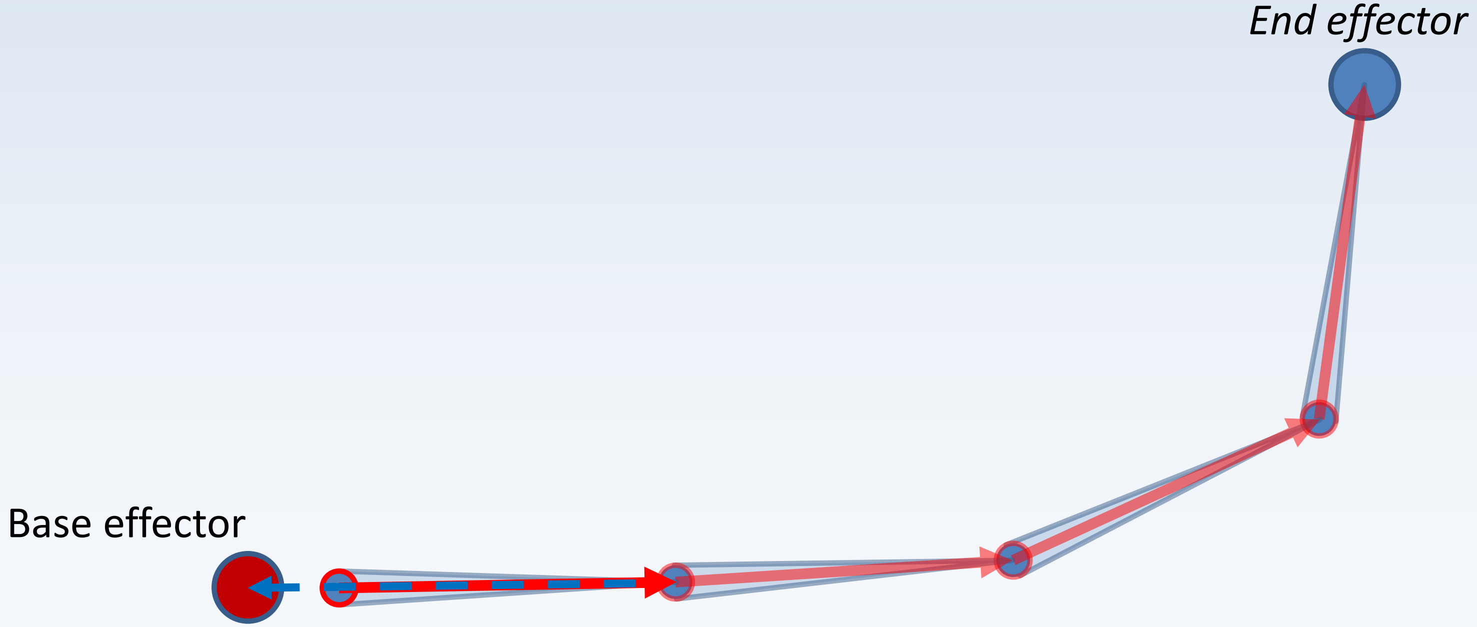


Inverse Kinematics : Garden Hose

- ***Unconstrained IK***: Repeatedly aim joints towards the effectors and estimate if this is a good solution:

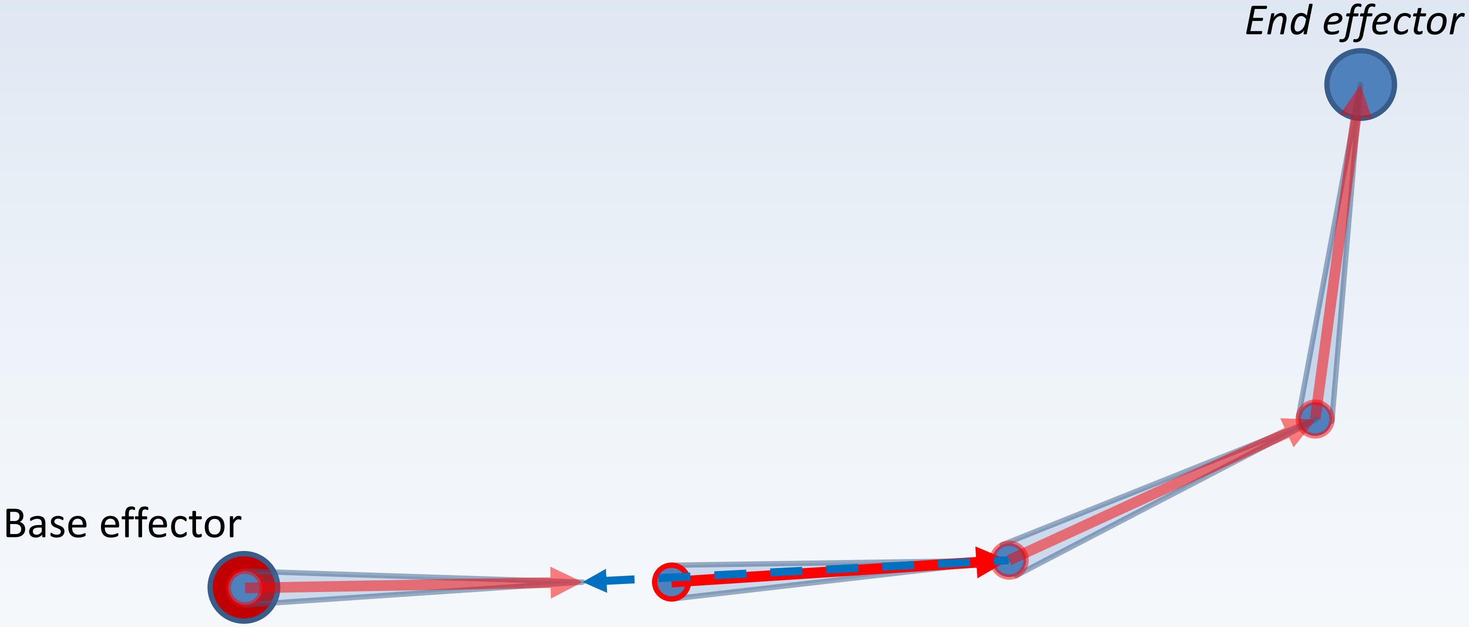
Inverse Kinematics : Garden Hose

- ***Unconstrained IK***: Once the base effector is reached, work back towards the end effector!
- Repeat until "close enough"



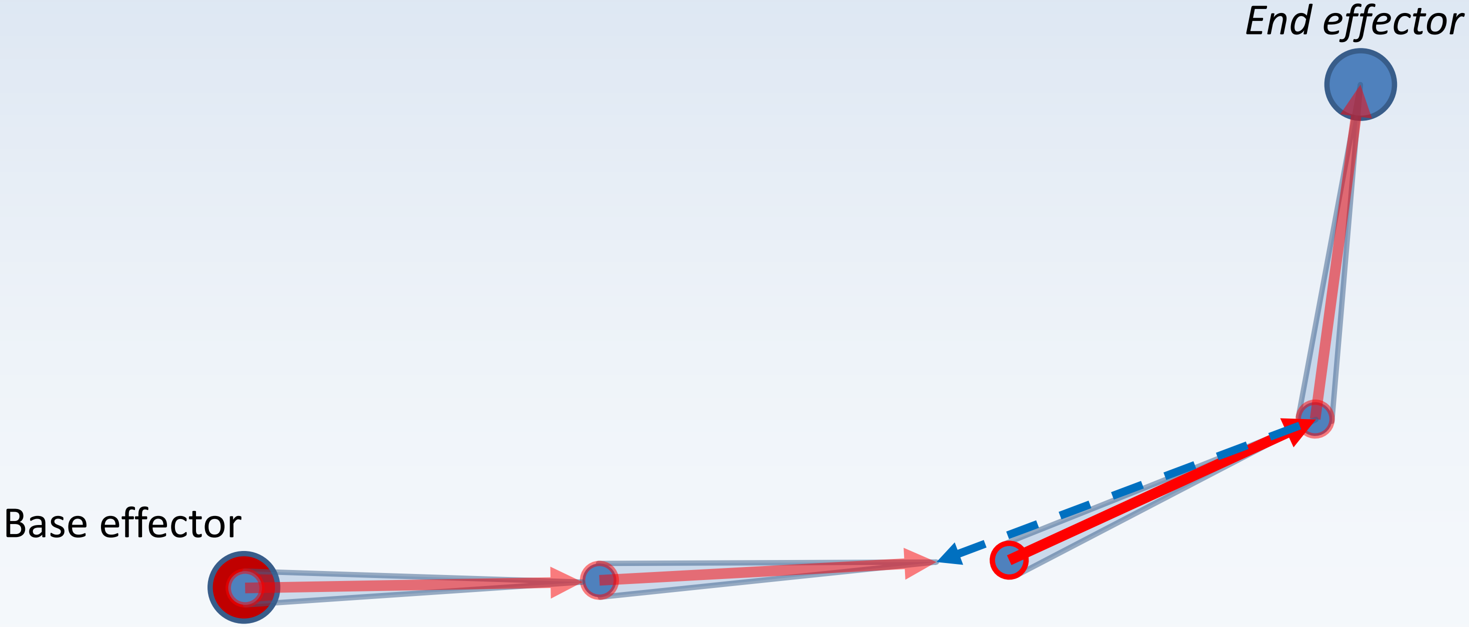
Inverse Kinematics : Garden Hose

- ***Unconstrained IK***: Once the base effector is reached, work back towards the end effector!
- Repeat until “close enough”



Inverse Kinematics : Garden Hose

- ***Unconstrained IK***: Once the base effector is reached, work back towards the end effector!
- Repeat until "close enough"



Inverse Kinematics : Garden Hose

- ***Unconstrained IK***: Once the base effector is reached, work back towards the end effector!
- Repeat until "close enough"

Inverse Kinematics : Garden Hose

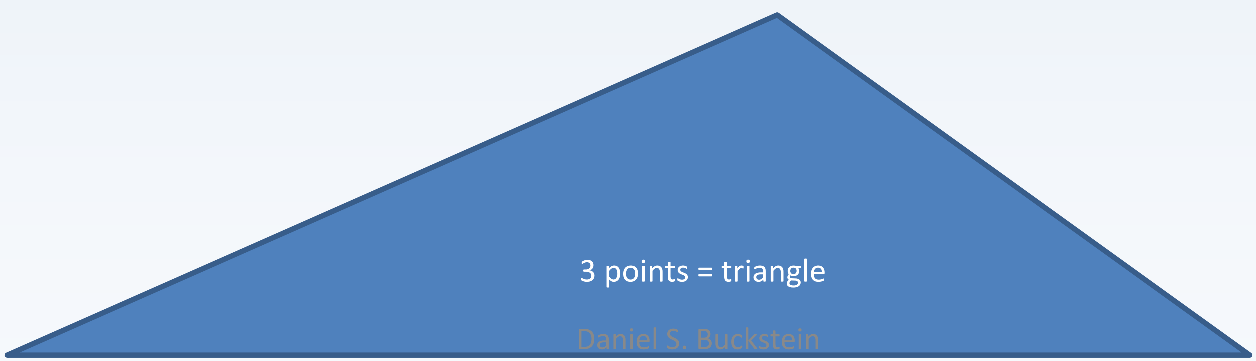
- ***Unconstrained IK***: Usually we solve until we meet a maximum number of iterations…
- …or the end of the last joint is negligibly close to the end effector!

Inverse Kinematics

- ***Constrained IK***: complex solutions:
- Jacobian inverse
  - https://en.wikipedia.org/wiki/Inverse_kinematics#The_Jacobian_inverse_technique
  - https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant
- Denavit-Hartenberg parameters
  - https://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters
- Pure geometric solutions
  - (trigonometry for the win)

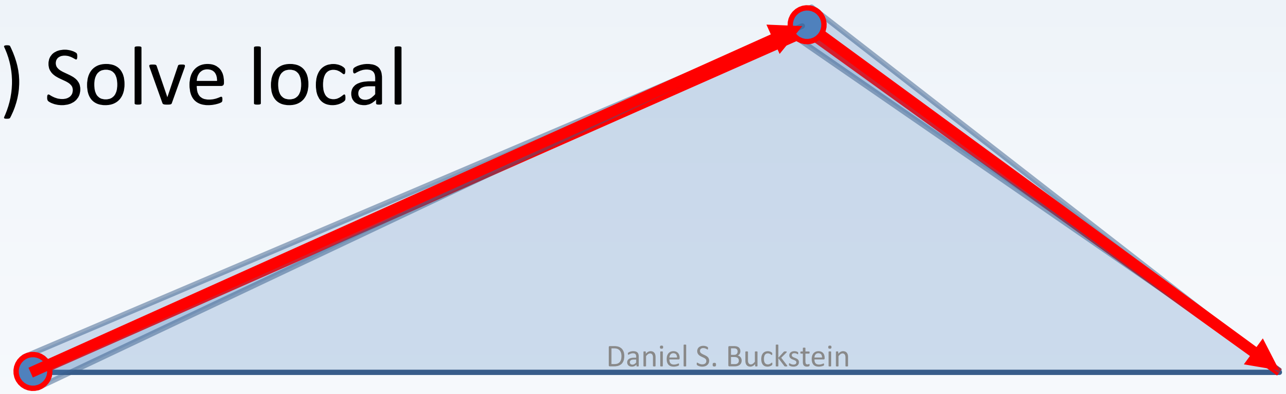
Inverse Kinematics: Triangles

- ***Triangle power***: 2-joint chains form a *triangle*
- We can solve *world transforms* using the properties of triangles
- Very useful for arm and leg IK handles
- ***Rotation plane constraint***: the IK solver must position all nodes on a known plane



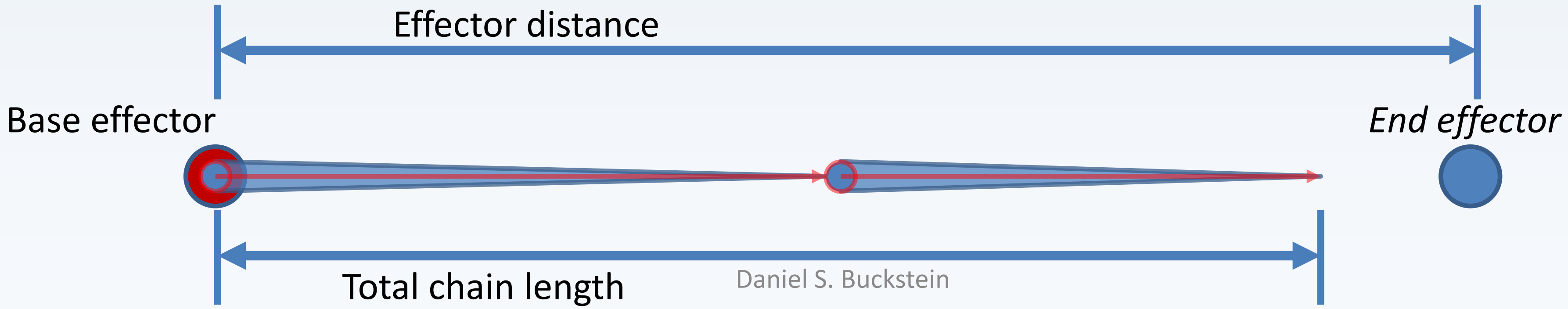
Inverse Kinematics: Triangles

- Dan figured this one out all by himself, and is happy to impart his knowledge unto thee
- Using years of experience and knowledge of linear algebra… practice makes perfect ☺
- 1) Solve positions
- 2) Solve rotations
- 3) Solve local



Inverse Kinematics: Triangles

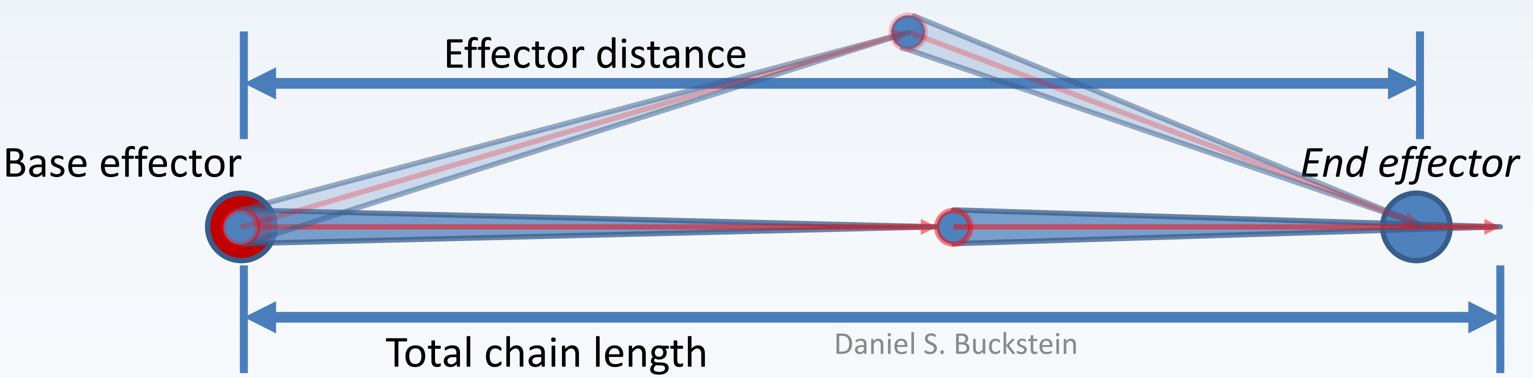
- ***1) Solve positions***
- Triangle properties
- First, determine vector from base to end effector; joint affected by end wants to snap to end effector
- Distance check: effector is too far away…



Inverse Kinematics: Triangles

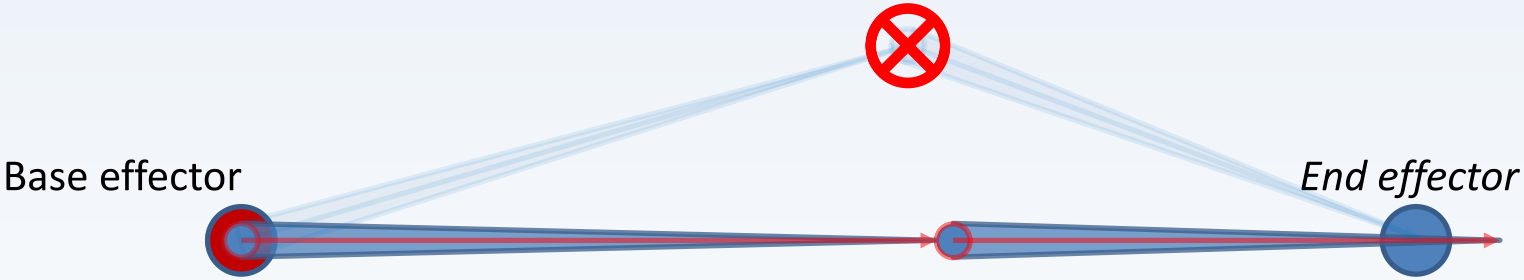
- ***1) Solve positions***
- Distance check: same as before!

- Solution exists! (shown transparent here)



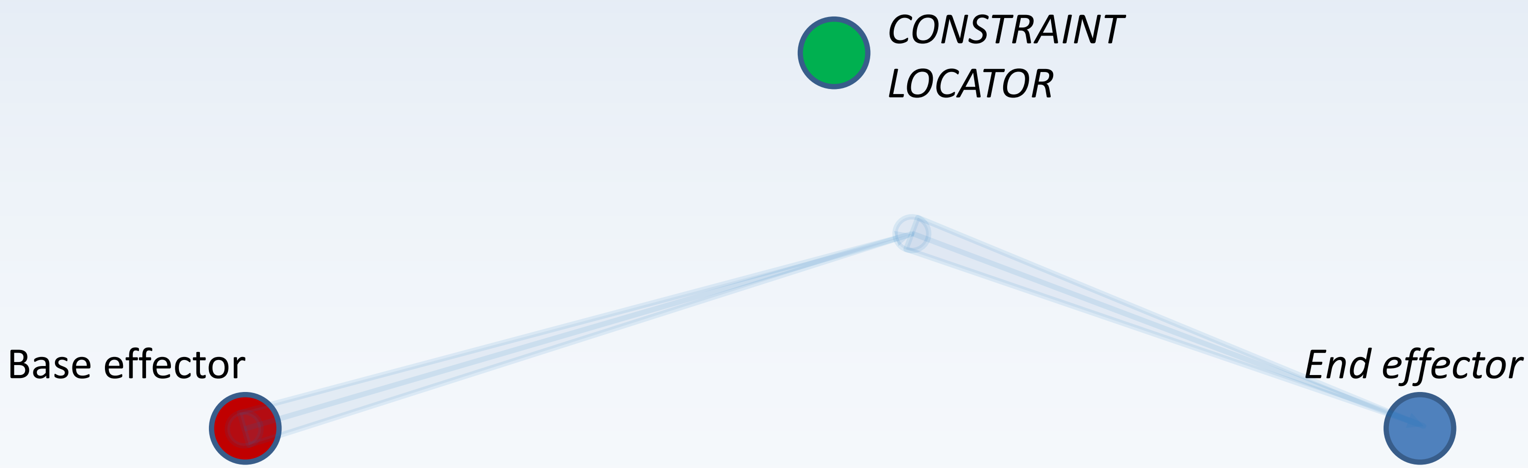
Inverse Kinematics: Triangles

- ***1) Solve positions***
- *Joint lengths must remain fixed*!!!
- Problem: a plane can be defined as long as we have three points…
- But we have not yet solved the third!!!



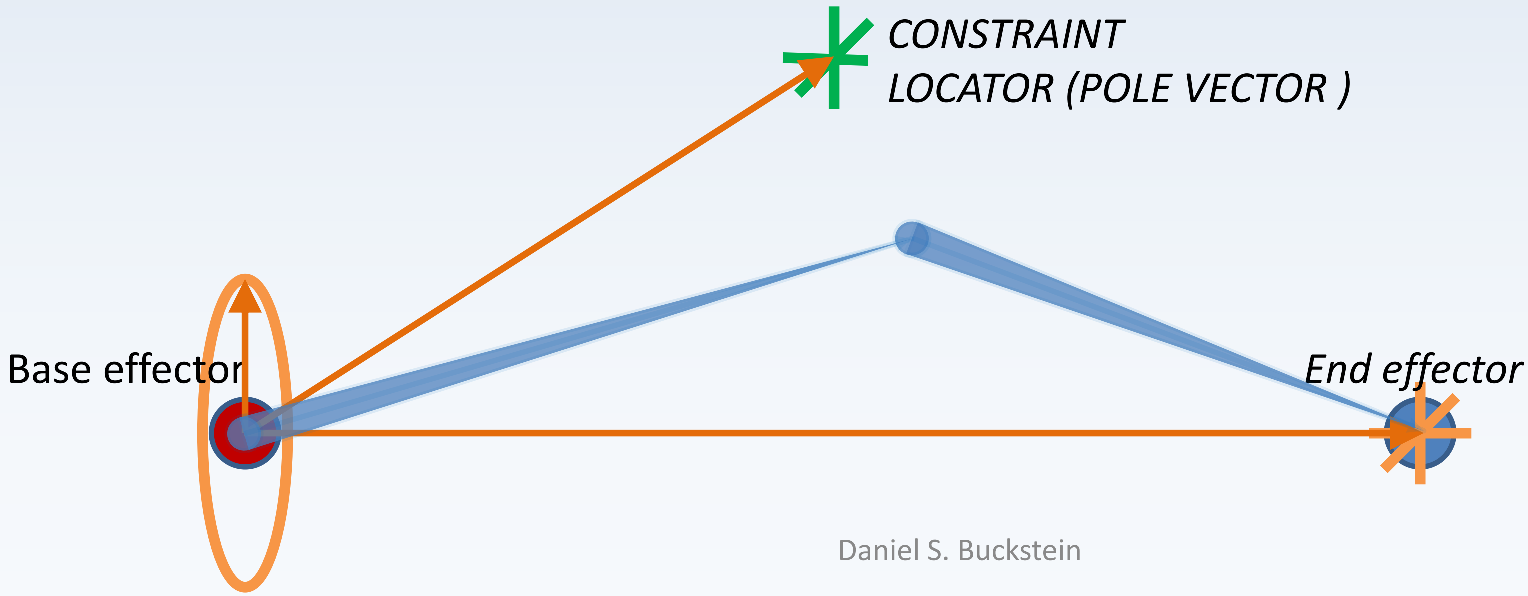
Inverse Kinematics: Triangles

- ***1) Solve positions***
- Where does the third point come from???
- We define a "constraint locator" which acts as the third point stand-in:



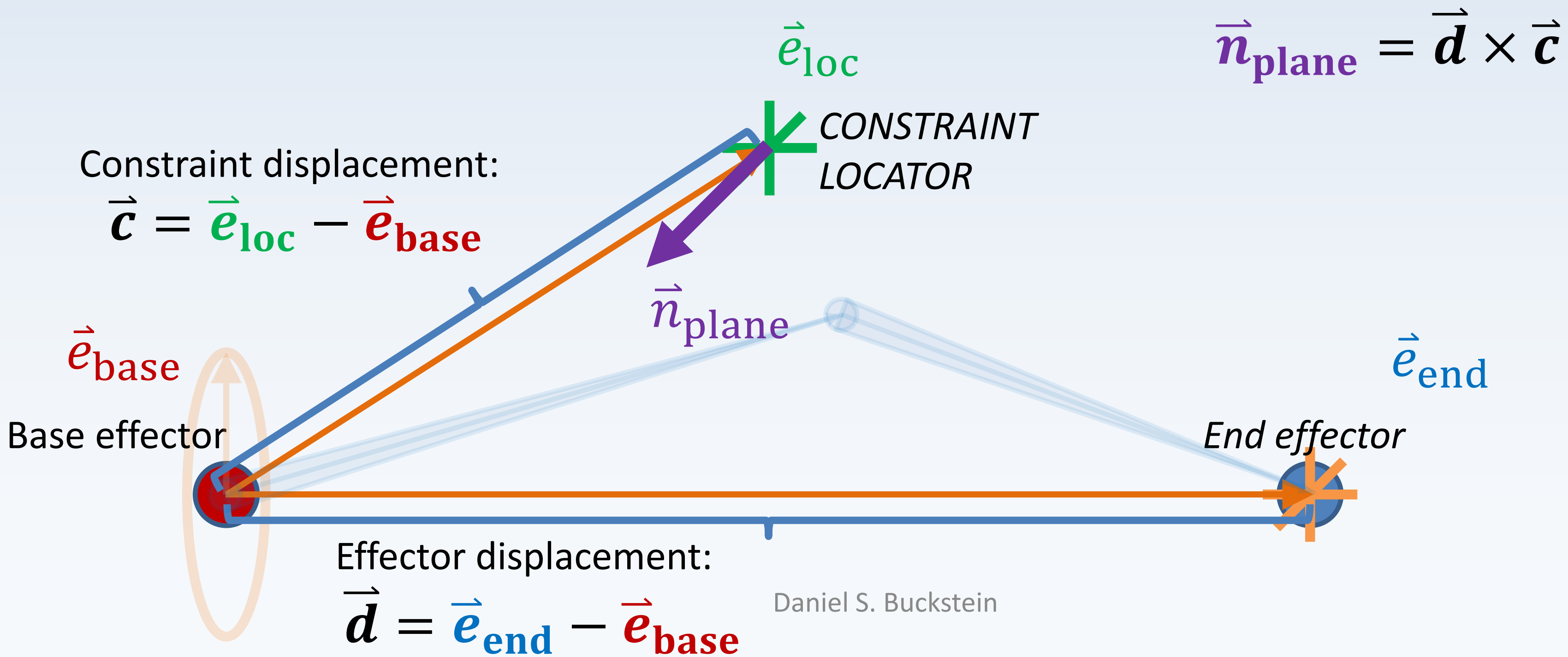
Inverse Kinematics: Triangles

- ***1) Solve positions***
- "Pole vector" constraint: plane is defined by two effector positions and third arbitrary locator



Inverse Kinematics: Triangles

- ***1) Solve positions***
- The vectors given to us here allow us to compute some very important info…

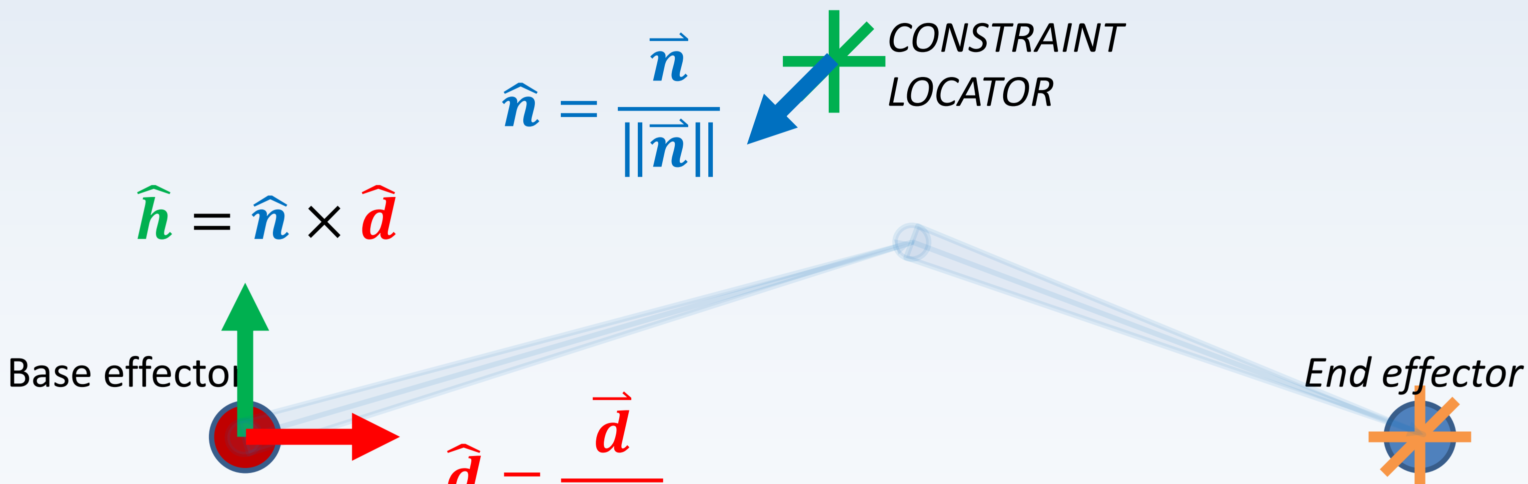


Inverse Kinematics: Triangles

- ***1) Solve positions***
- If we normalize the displacement and normal vectors…

$$\hat{n} = \frac{\vec{n}}{\|\vec{n}\|}$$

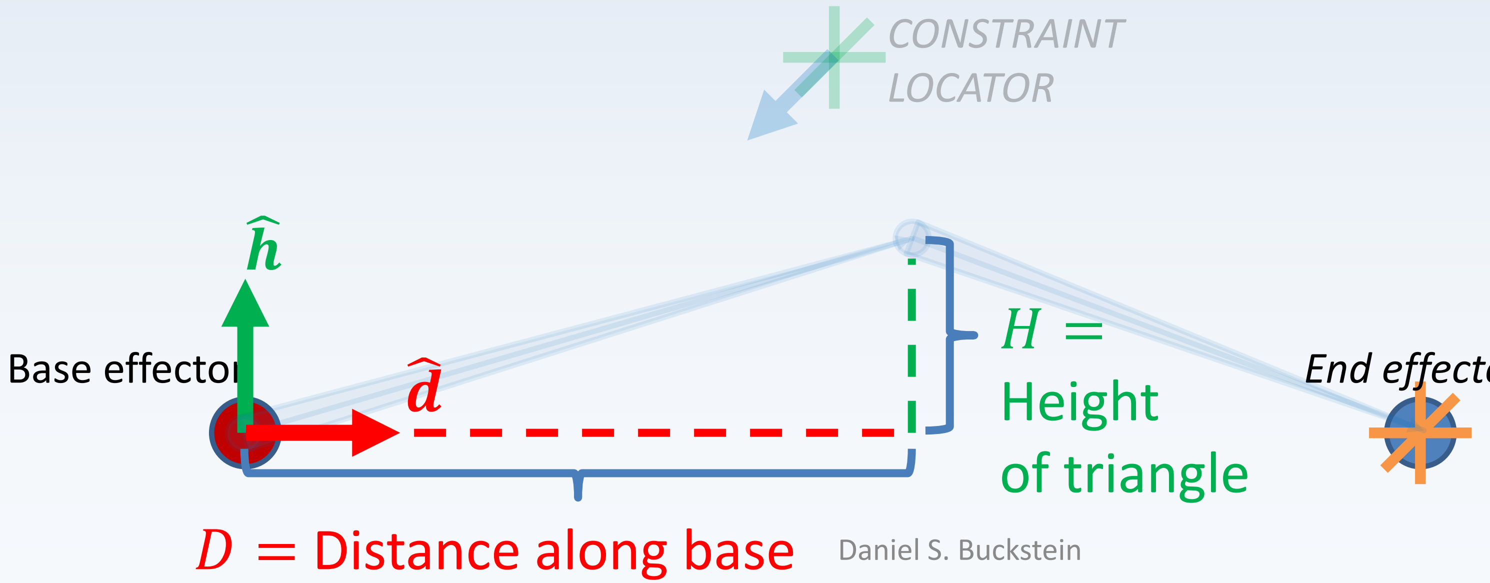
$$\hat{h} = \hat{n} \times \hat{d}$$



$$\hat{d} = \frac{\vec{d}}{\|\vec{d}\|}$$

Inverse Kinematics: Triangles

- ***1) Solve positions***
- We now have more than enough information to ***solve***… but how???



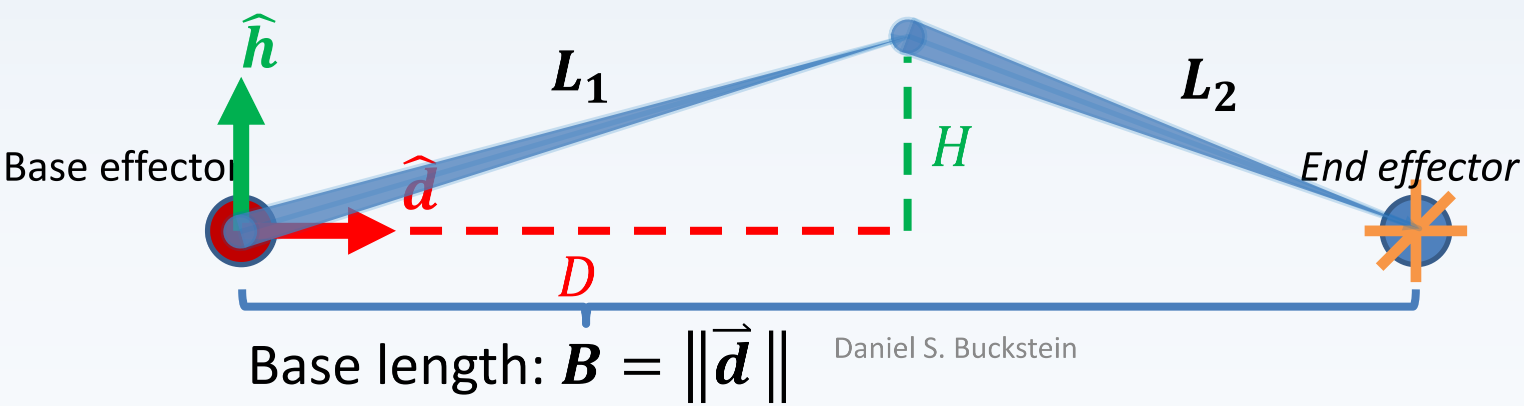
Inverse Kinematics: Triangles

- ***1) Solve positions***
- ***Geometric formula #1: <u>Heron's formula</u>***
  - Gives the area of the triangle using side lengths
  - Solve 'H' using classic triangle area formula

$$\boldsymbol{A = \sqrt{s(s-B)(s-L_1)(s-L_2)}}$$

$$s = \frac{1}{2}(B + L_1 + L_2)$$

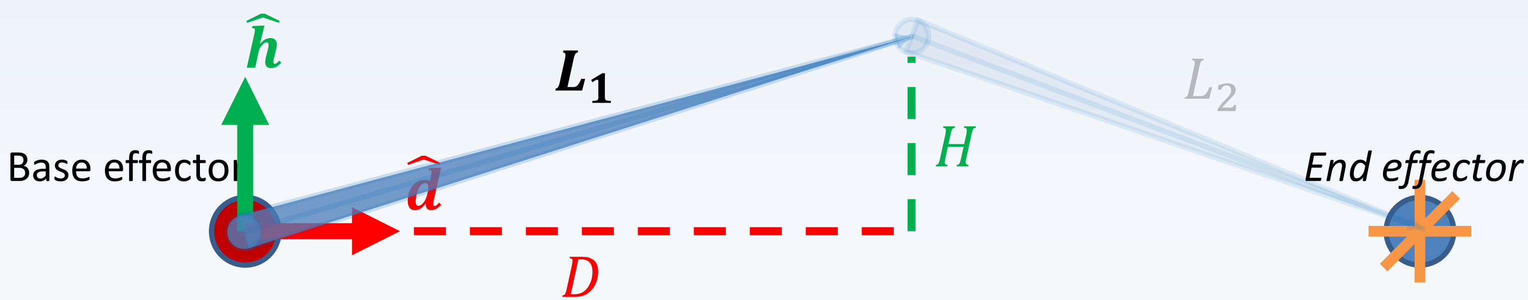
$$\boldsymbol{A = \frac{1}{2}BH \rightarrow H = \frac{2A}{B}}$$



Inverse Kinematics: Triangles

- ***1) Solve positions***
- ***Geometric formula #2: <u>Pythagorean theorem</u>***
  – Solve 'D' using first bone length and 'H'

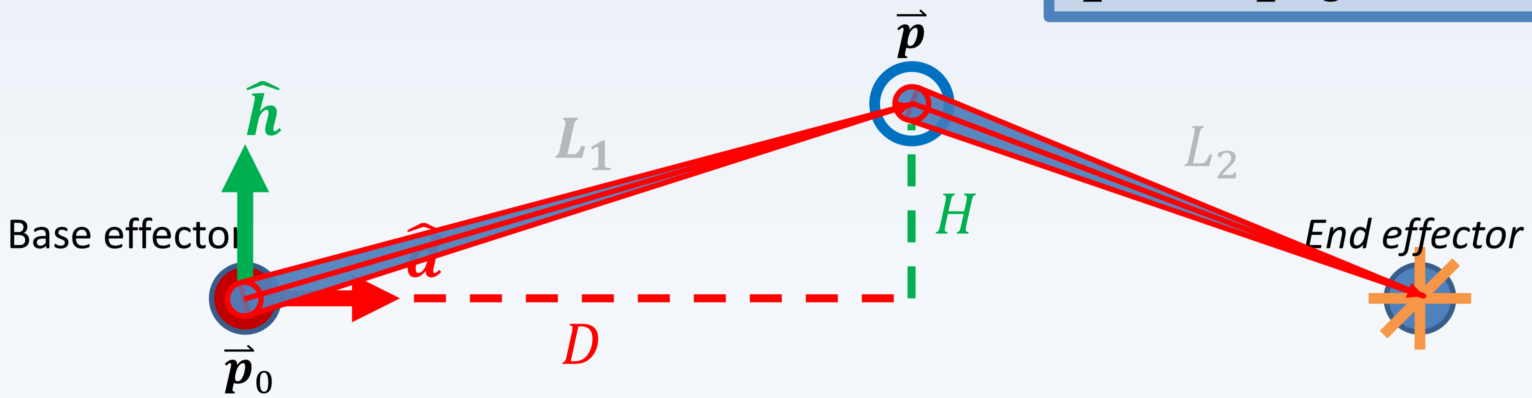
$$\boldsymbol{L_1^2 = D^2 + H^2} \longrightarrow \boldsymbol{D = \sqrt{L_1^2 - H^2}}$$



Inverse Kinematics: Triangles

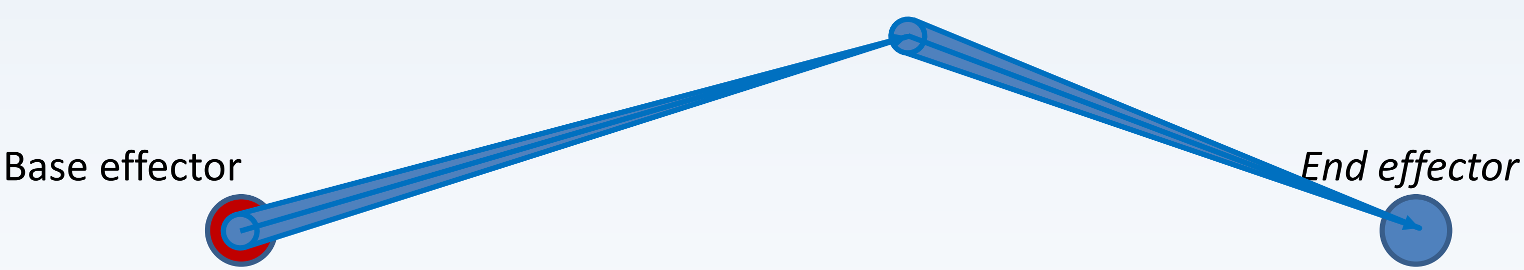
- ***1) Solve positions***
- ***SOLUTION:***
- Calculate the missing right-triangle position as an offset from the base effector's position

$$\vec{\boldsymbol{p}} = \vec{\boldsymbol{p}}_0 + D\hat{\boldsymbol{d}} + H\hat{\boldsymbol{h}}$$



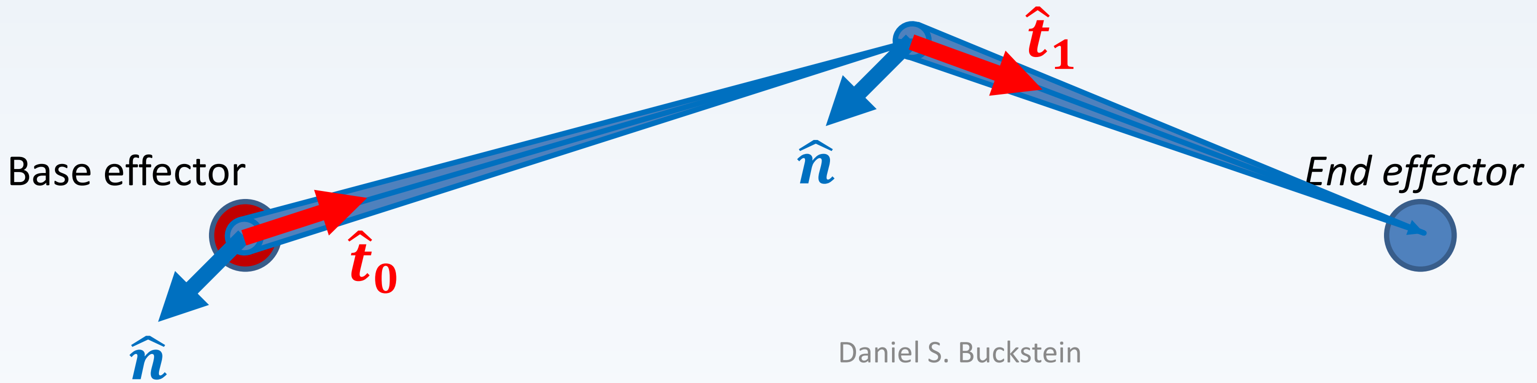
Inverse Kinematics: Triangles

- All of this happens ***in a common coordinate frame*** (e.g. world space)
- If FK is about going from local to world…
- …IK involves going from world to local… why?
  - Because *animation* takes place in local space!



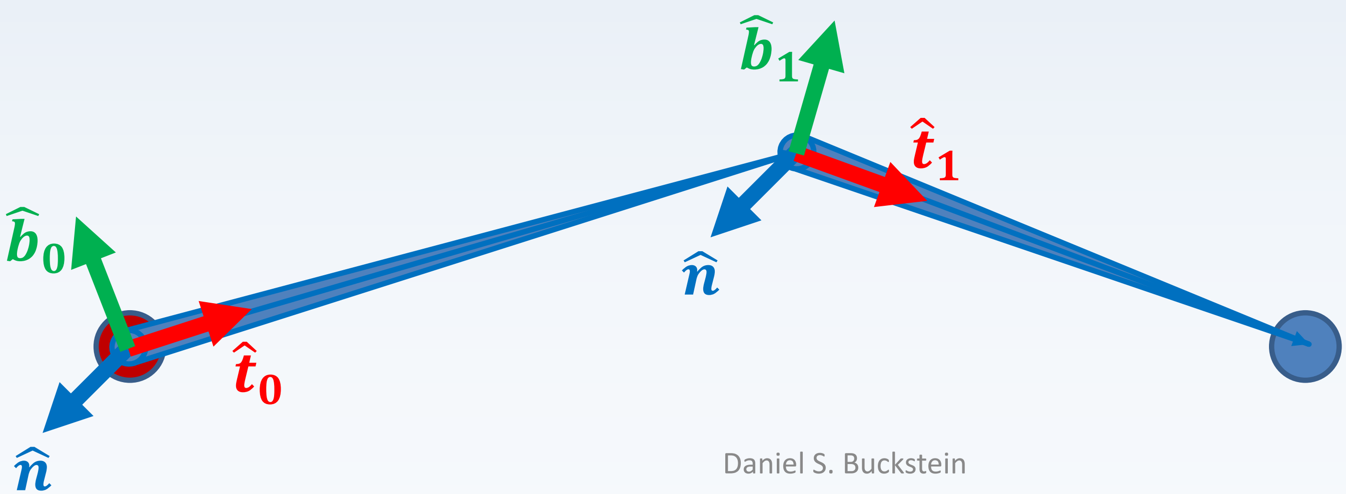
Inverse Kinematics: Triangles

- Now that we have the *position* of the middle joint, we complete the transform by solving *orientation*:
- ***2) Solve orientations***
  - We know the *normal* basis from the previous step
  - We know the *tangent* basis of each bone



Inverse Kinematics: Triangles

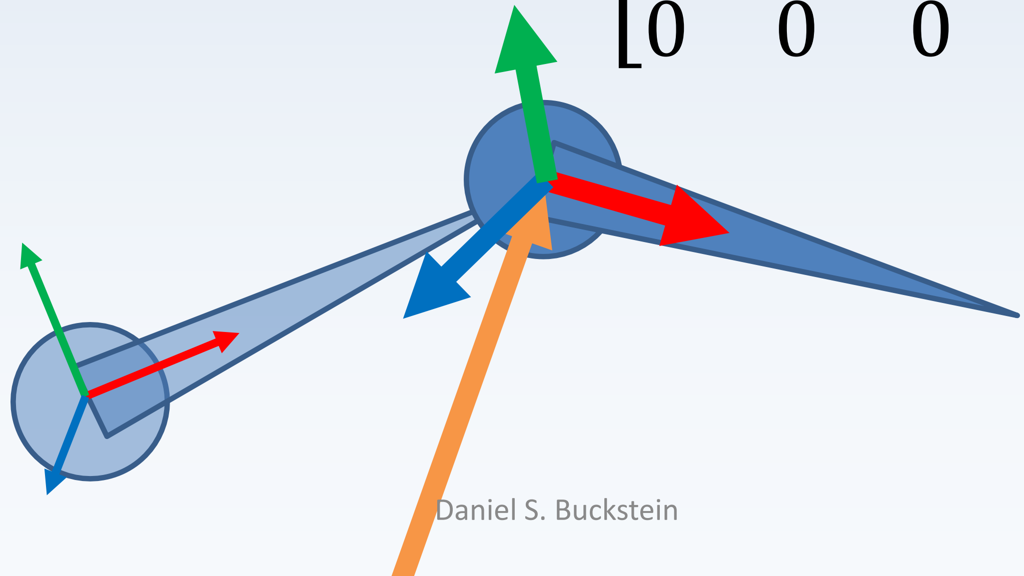
- ***2) Solve orientations***
- ***Frenet-Serret frame***: Assemble global matrix using three basis vectors calculated in common space!!!



Inverse Kinematics: Triangles

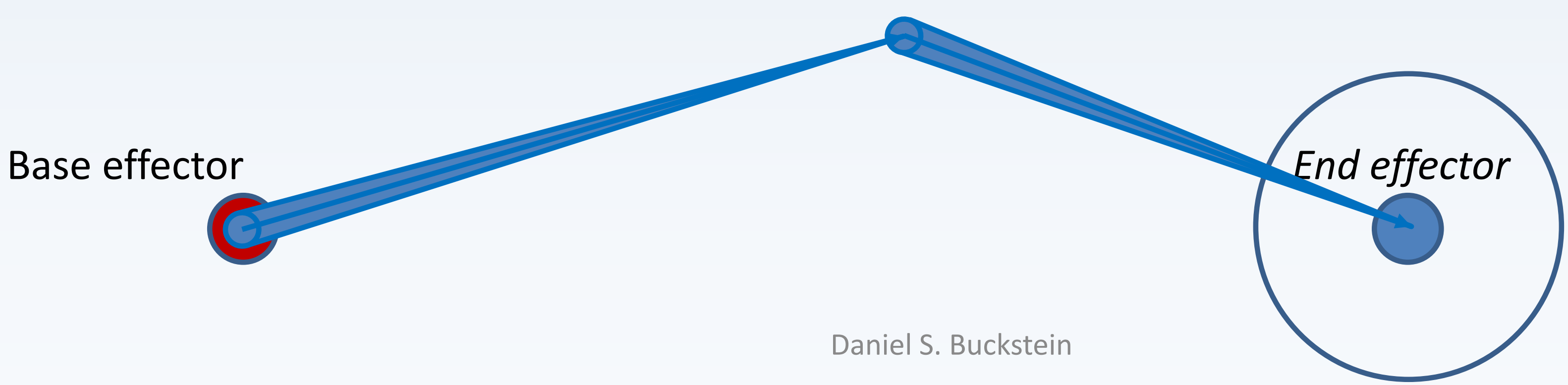
- ***2) Solve orientations***
- Global modification of something we have already seen:

$$^{\text{world}}T_{\text{joint}_{(4\times 4)}} = \begin{bmatrix} \hat{\boldsymbol{t}} & \widehat{\boldsymbol{b}} & \widehat{\boldsymbol{n}} & \overrightarrow{\boldsymbol{p}} \\ & & & \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Inverse Kinematics: Triangles

- ***2) Solve orientations***
- Finally, need to solve orientation of end effector joint
- Simple: it wants to match the end effector ☺

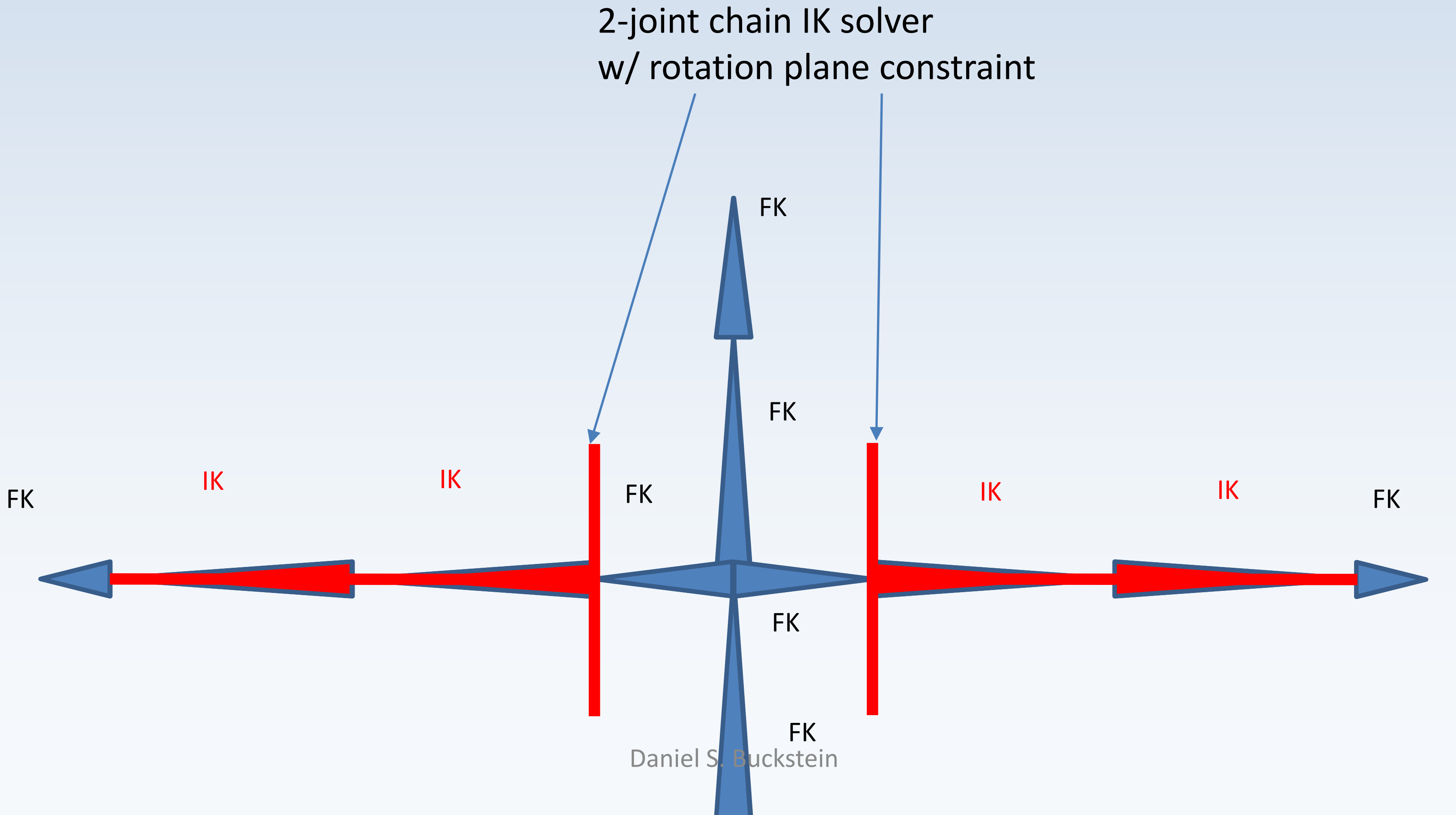


Inverse Kinematics: Triangles

- ***3) Solve local***
- Finally, apply fundamental IK formula ***for the affected joint chain only!***
- The rest of the skeleton is already solved
  - Prior FK call
- No need to overwrite local matrices if they are not part of the chain
- Now we have the local pose! :o

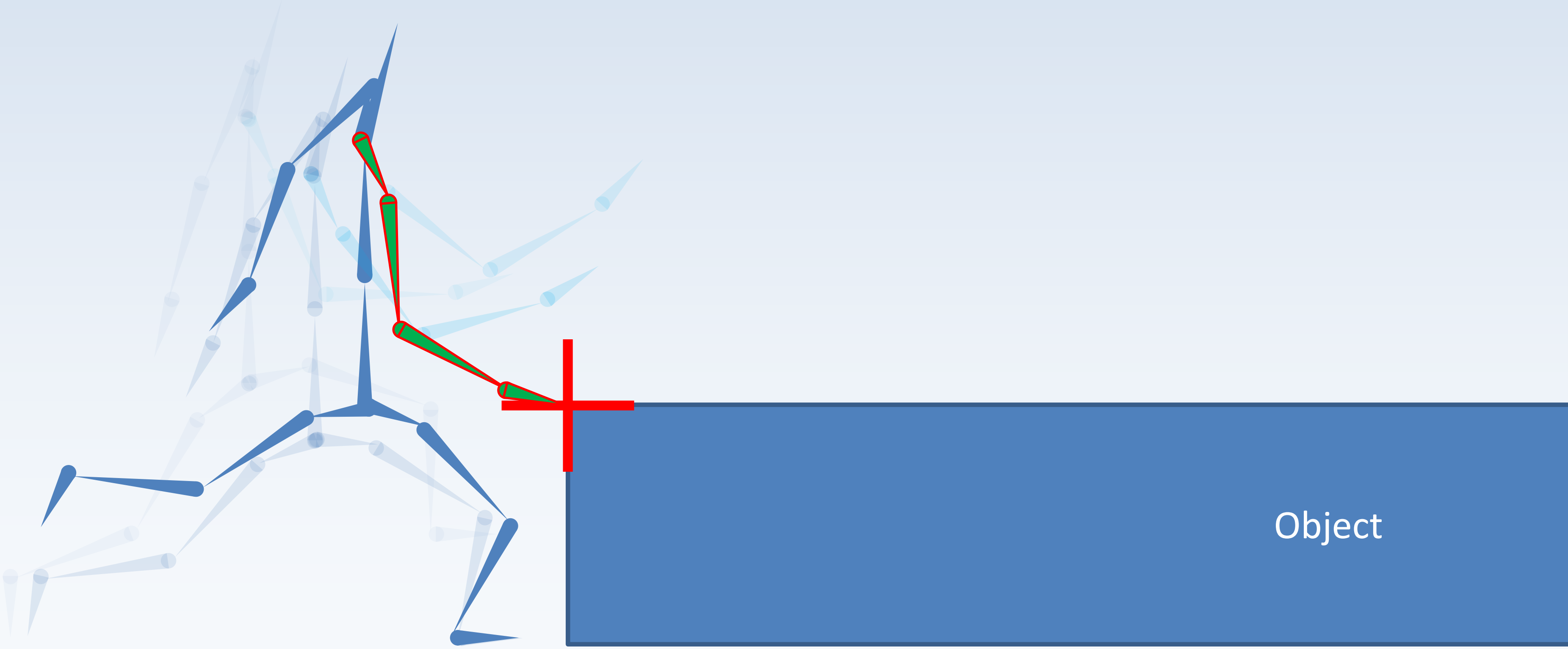
Inverse Kinematics

- FK-IK combo example: humanoid:



Inverse Kinematics

- Future applications: maybe your character wants to grab something…



Inverse Kinematics

- An interesting topic for after you get comfortable with the fundamentals:
- “Denavit-Hartenberg parameters”
- Solving a *constrained system*
- https://www.youtube.com/watch?v=rA9tm0gTln8
  - The particle systems at the end. SO GOOD.

The end.

- Questions? Comments? Concerns?