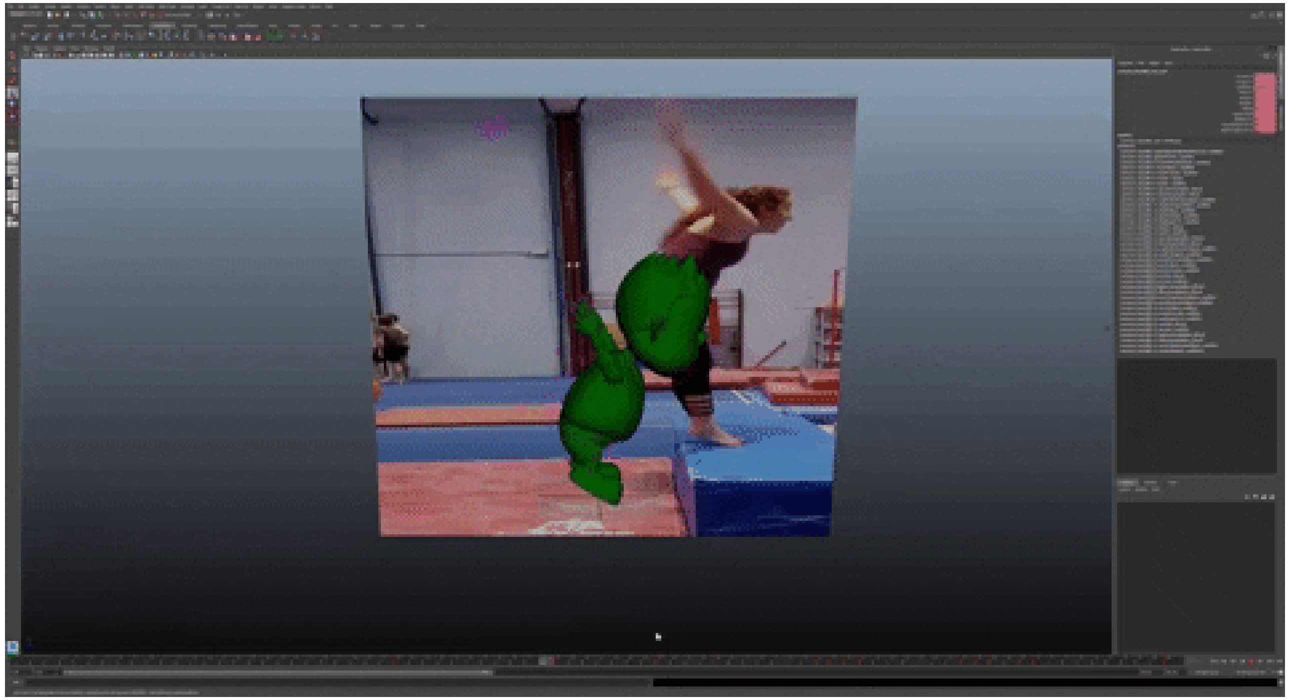
Kinematics



Some Slides/Images adapted from Marschner and Shirley and David Levin

Announcements

A6 due date moved to Sunday 26 July :)

A7 due date moved to Sunday 2 August

Office hour after class today 2-3PM

Animation and Kinematics

Monday:

Animation in Computer Graphics

Skinning for Mesh Deformation

Forward Kinematics

Keyframe Animation

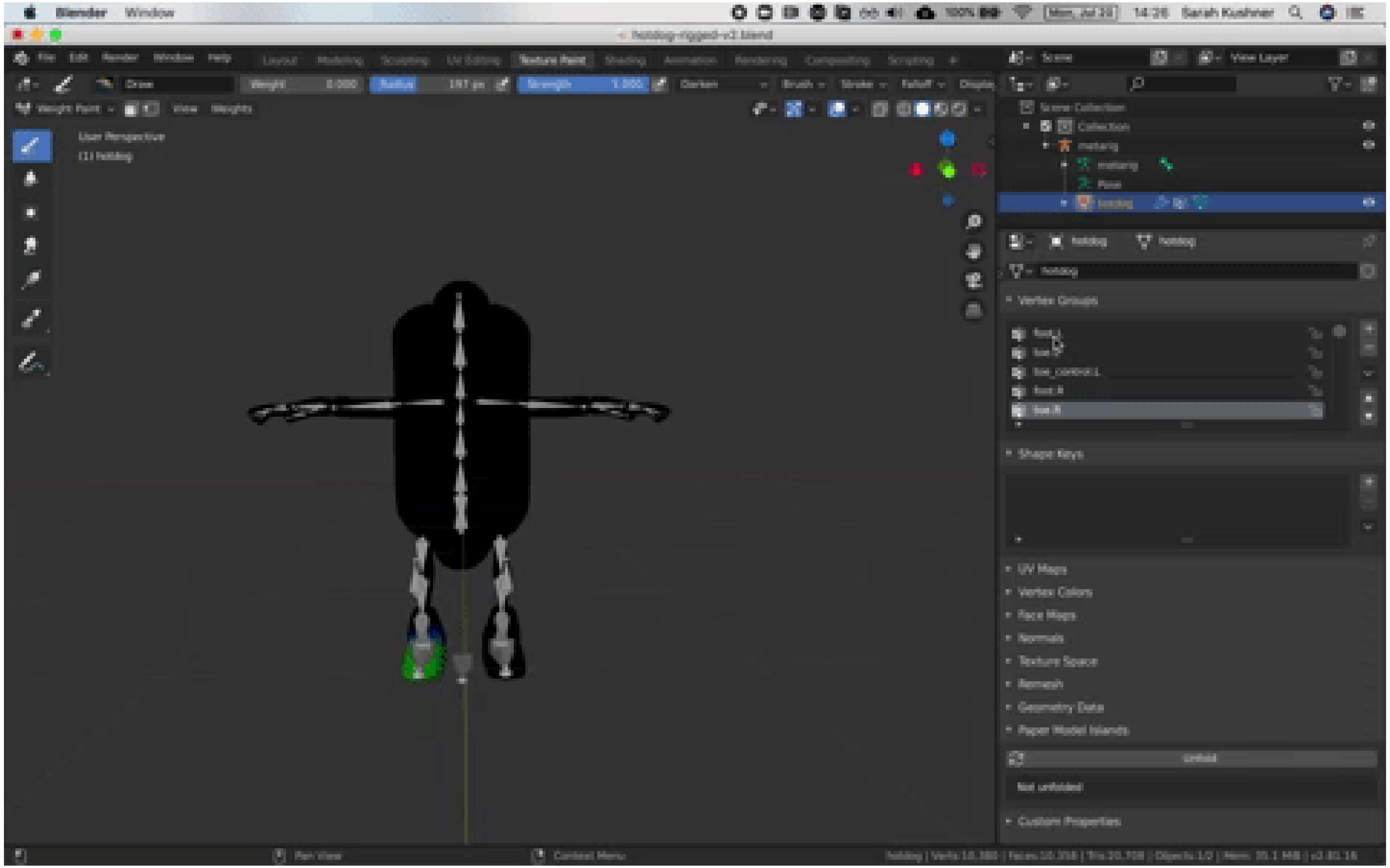
Today:

Review Skinning and Forward Kinematics

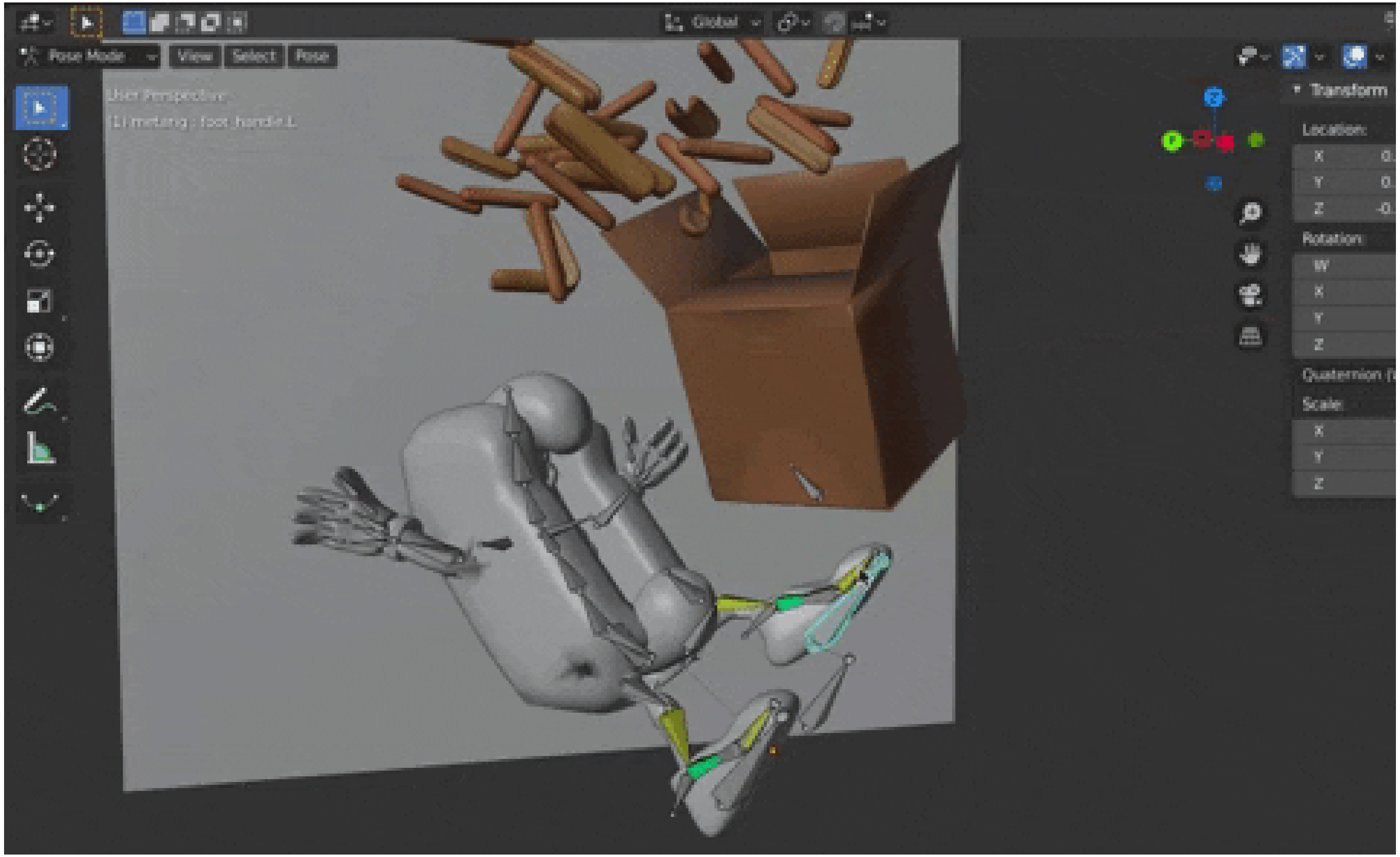
Keyframe Animation + Splines

Inverse Kinematics

Rigging and Weight Painting Example

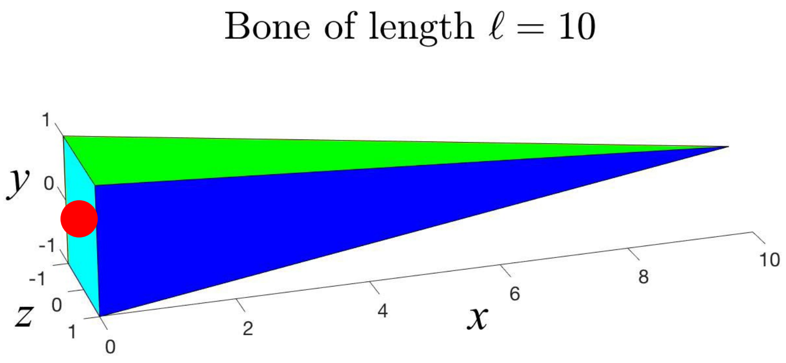


Posing Example with Inverse Kinematics (IK)



Rendered Example

Review of Skeletons: Bones

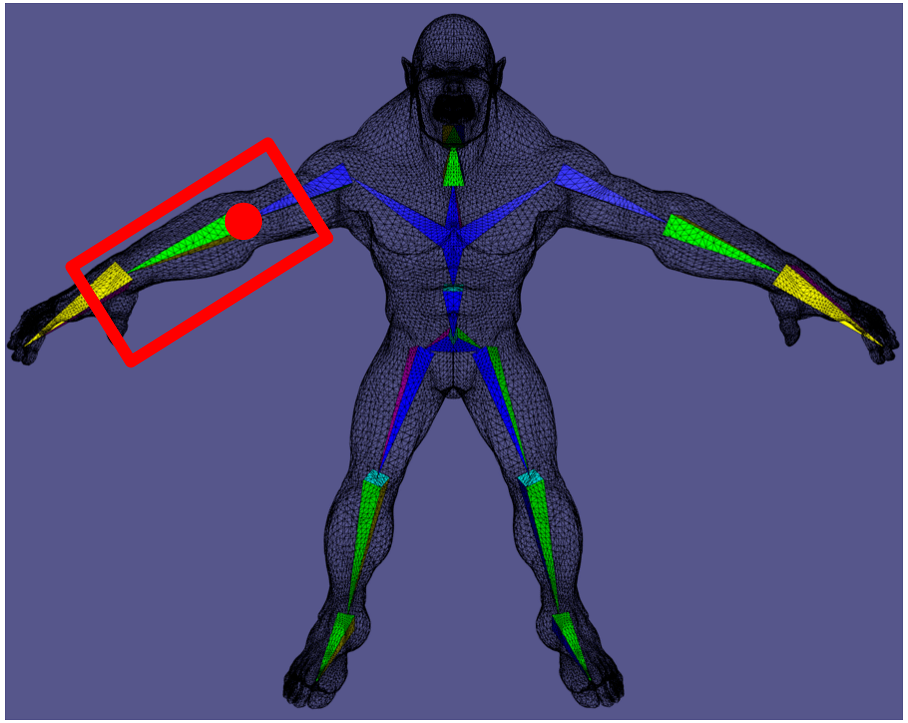


Tip at

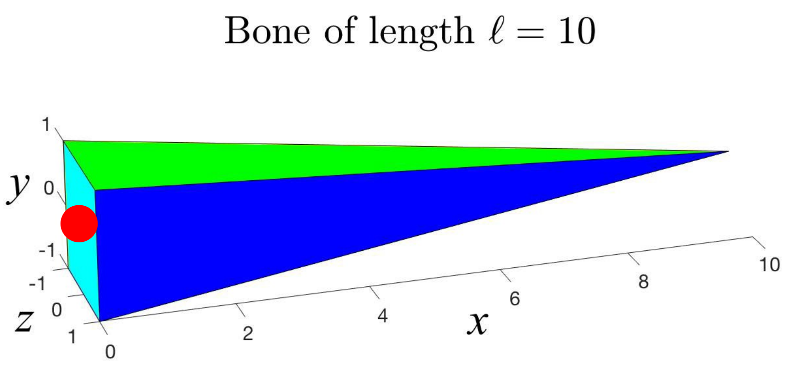
$$(\ell, 0, 0)$$

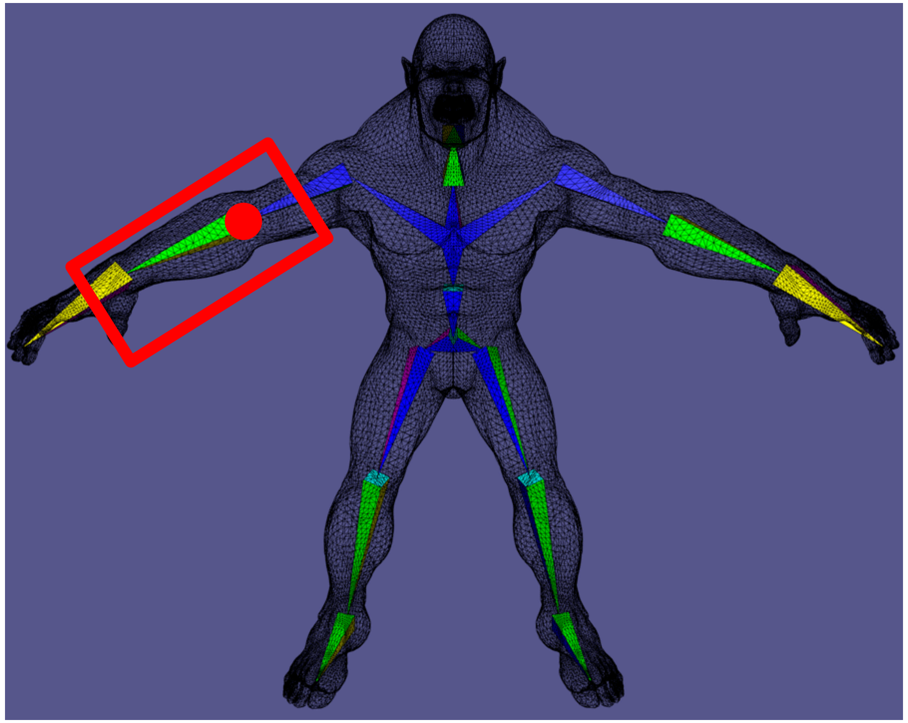
Tail at

$$(0, 0, 0)$$



Review of Skeletons: Transformations

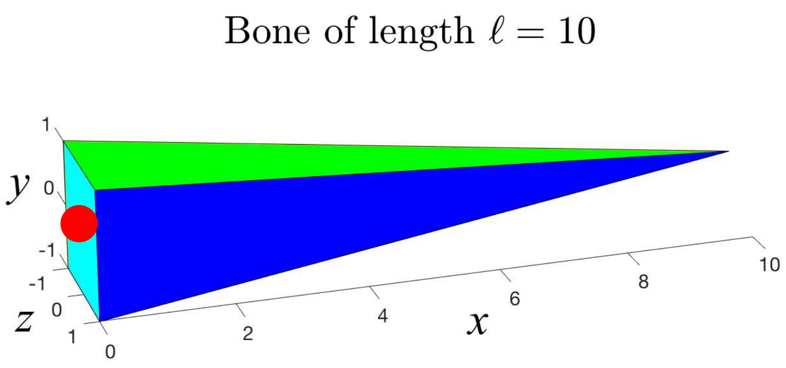


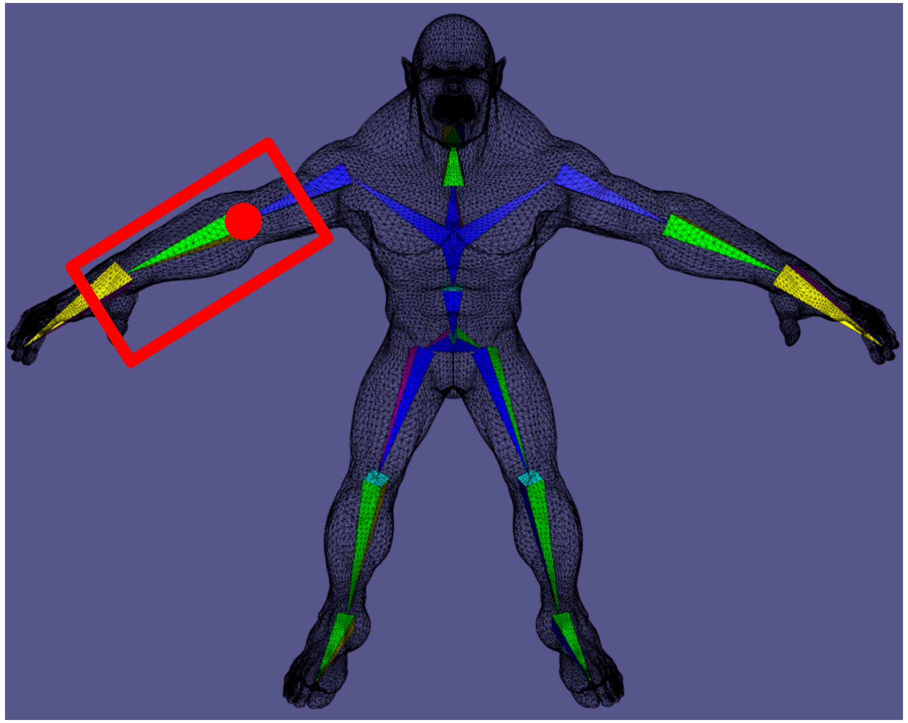


Transformations:
rotation + translation

$$T = (R \quad \hat{t}) \in \mathbb{R}^{3\times 4}$$

Review of Skeletons: Translate Canonical to Rest





Need to determine T for each bone $T_i \in \mathbb{R}^{3\times4}$

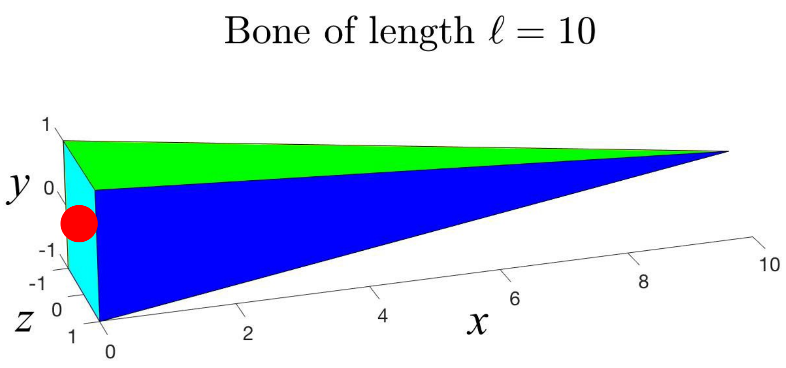
Translate from canonical pose to rest pose

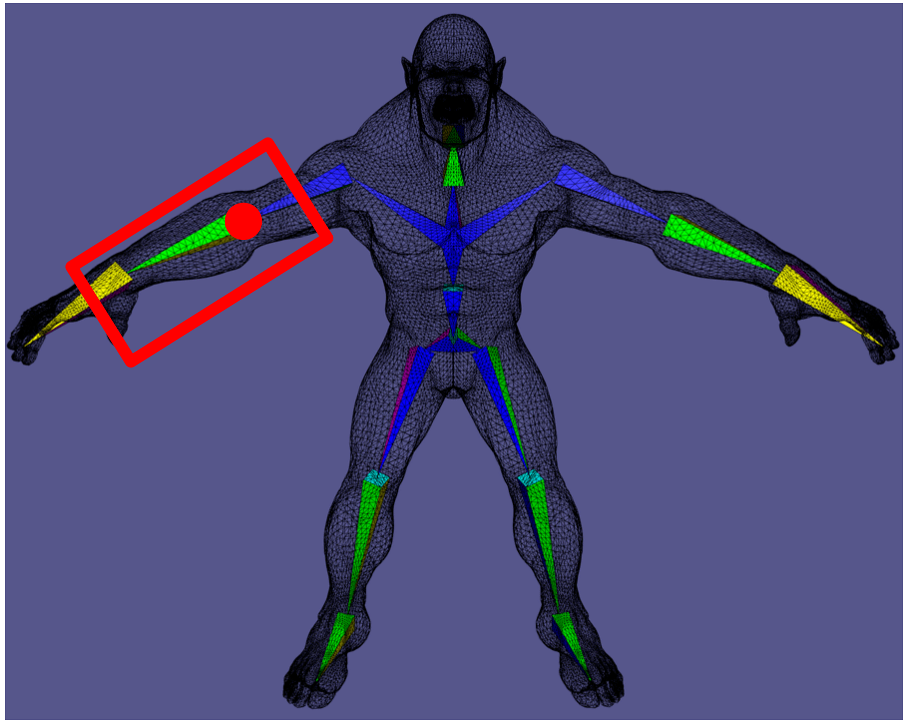
Translation part of matrix is the tail position, **s**

$$T_i = \begin{pmatrix} & & \hat{\mathbf{s}}_{ix} \\ \overline{R}_i & & \hat{\mathbf{s}}_{iy} \\ & & \hat{\mathbf{s}}_{iz} \end{pmatrix}$$

We choose the rotation so that the canonical tip maps to the rest tip, **d**

Review of Skeletons: Canonical to Rest





Need to determine
T for each bone

$$T_i \in \mathbb{R}^{3\times 4}$$

Translate from
canonical pose
to rest pose

Translation part of
matrix is the tail
position, **s**

$$T_i = \begin{pmatrix} & \hat{\mathbf{s}}_{ix} \\ \overline{R}_i & \hat{\mathbf{s}}_{iy} \\ & \hat{\mathbf{s}}_{iz} \end{pmatrix}$$

We choose the
rotation so that the
canonical tip maps
to the rest tip, **d**

Rest tail is coincident
with the rest tip of
its parent

$$\hat{\mathbf{d}}_p = \hat{\mathbf{s}}$$

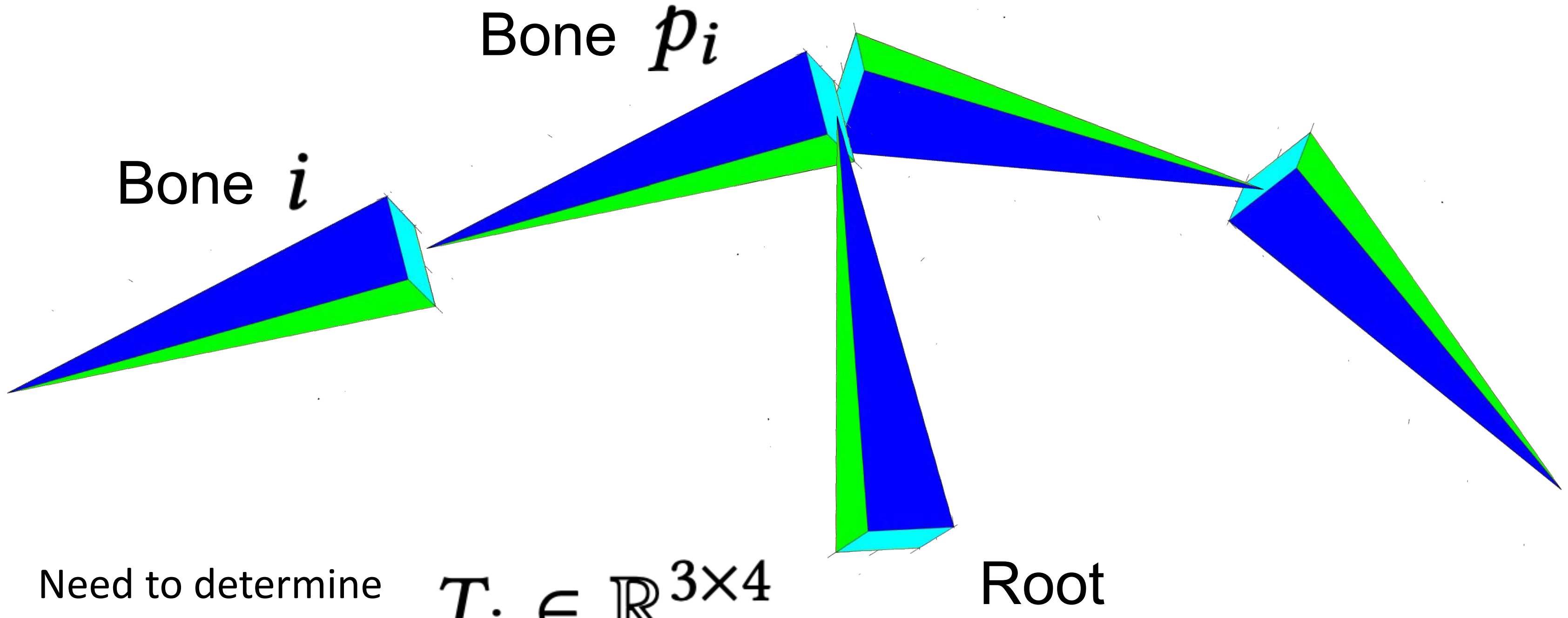
Forward Kinematics

Forward Kinematics

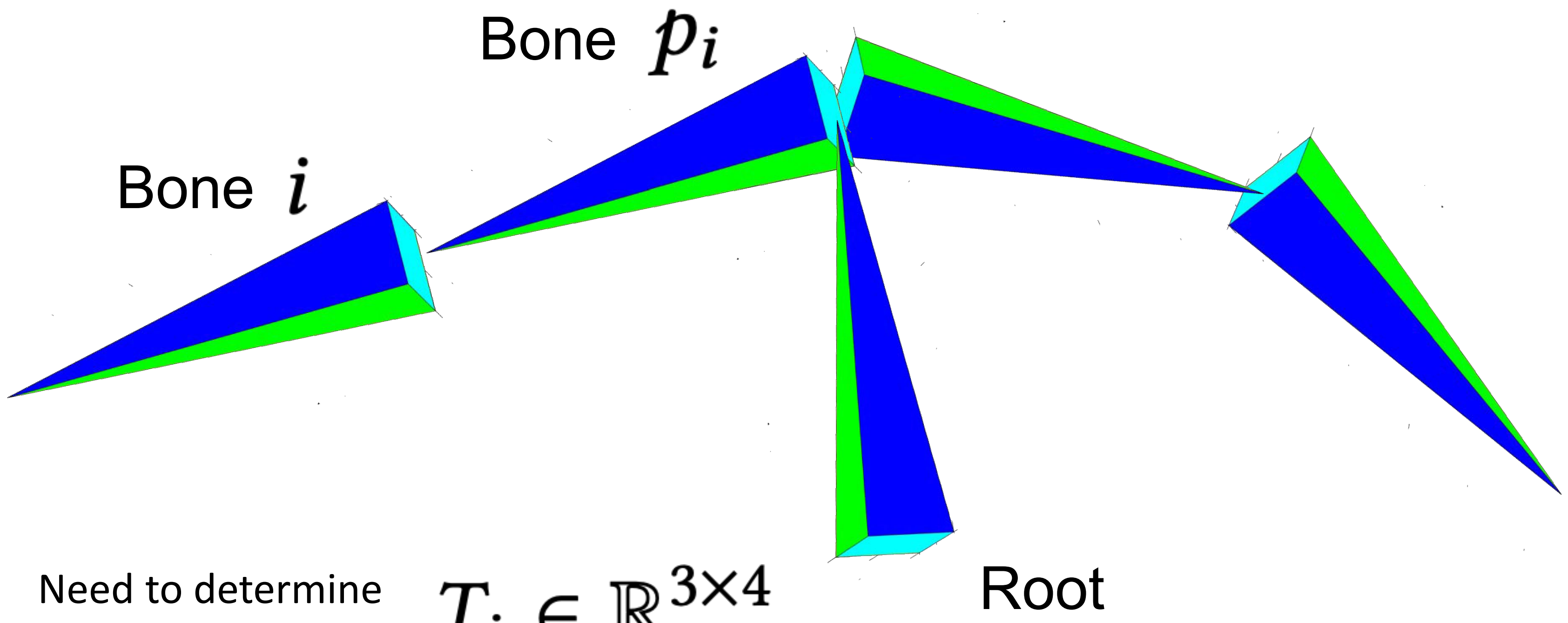
- Generate motion by setting all the bone positions by hand
- Compute the position of the end-effector from specified values for the joint parameters
  - Start from the root

https://en.wikipedia.org/wiki/Kinematics
https://en.wikipedia.org/wiki/Forward_kinematics
https://en.wikipedia.org/wiki/Robot_end_effector

Review of Skeletons: Tree



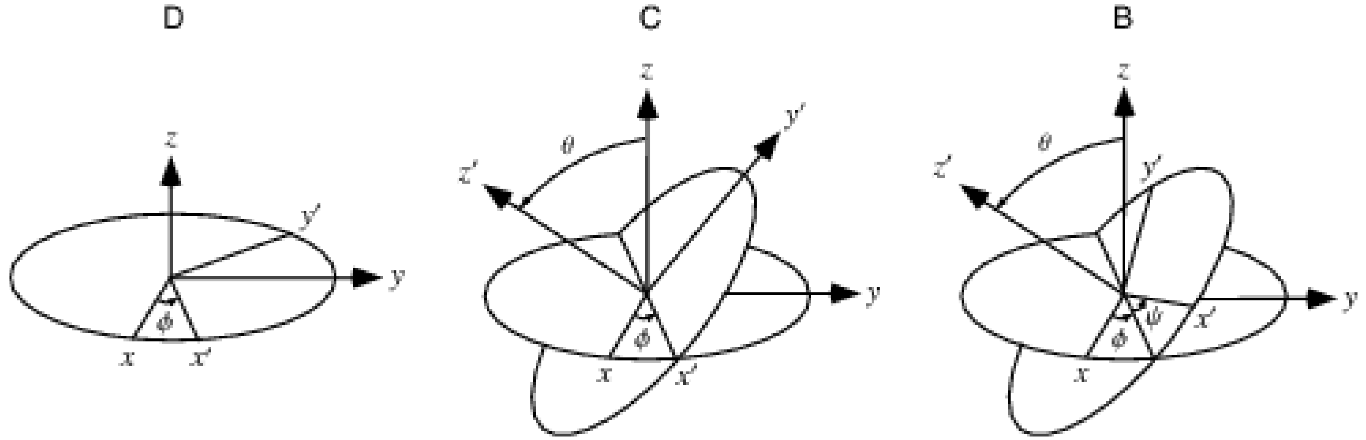
Need to determine T for each bone $T_i \in \mathbb{R}^{3\times 4}$



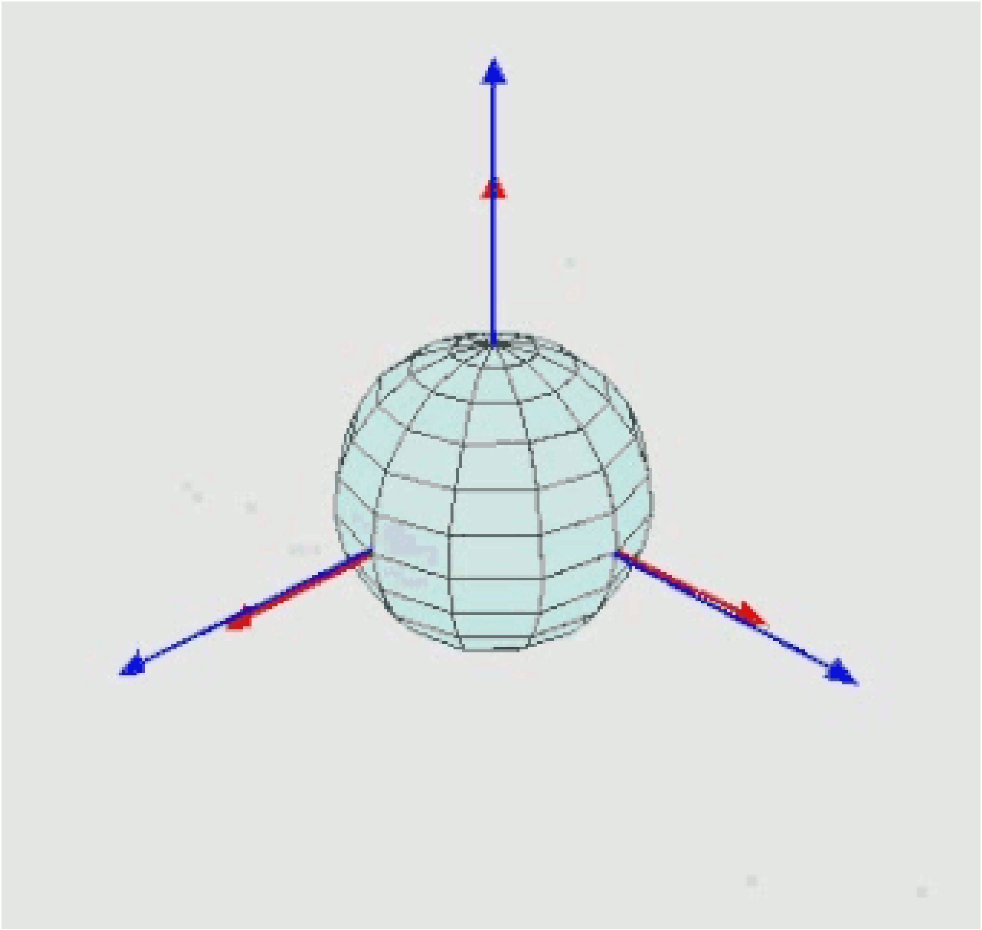
Need to determine T for each bone $T_i \in \mathbb{R}^{3\times4}$

aggregate *relative rotations* between bone ***i*** and its parent $\boldsymbol{p}_i$ $\bar{R}_i \in \mathbb{R}^{3\times3}$ ← Computed recursively!

Euler Angles



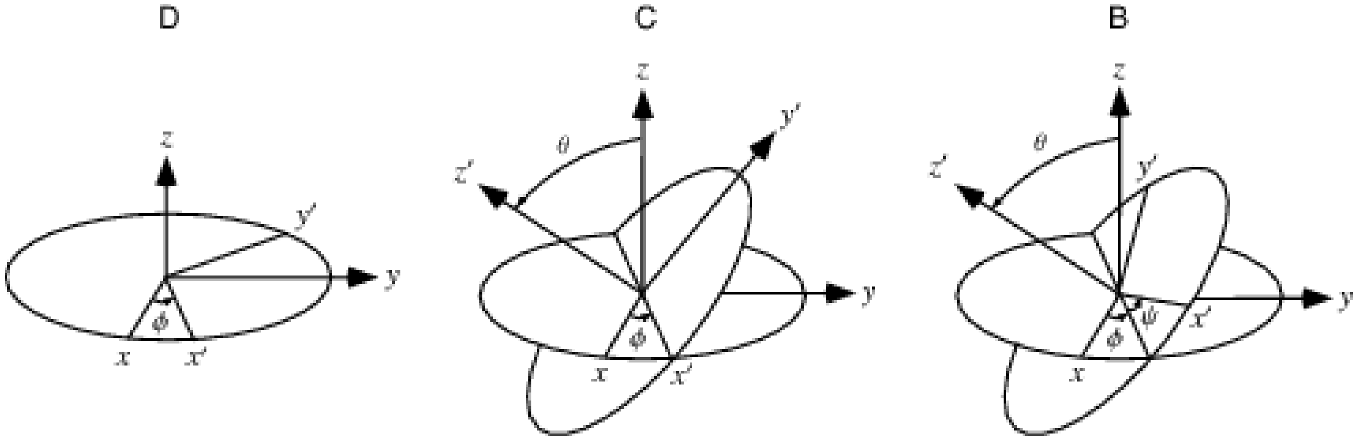
$$A = BCD$$



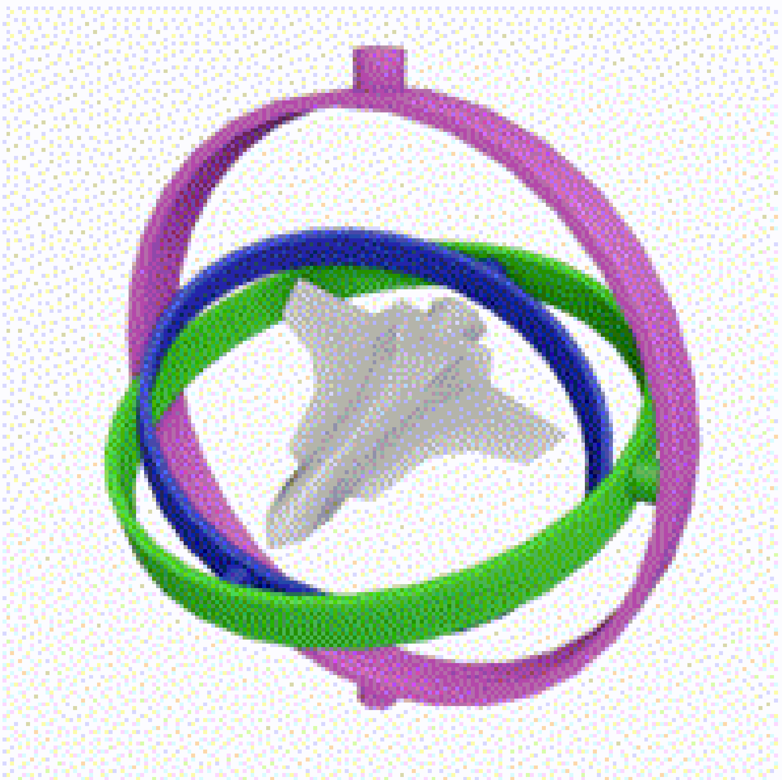
https://en.wikipedia.org/wiki/Euler_angles

https://mathworld.wolfram.com/EulerAngles.html

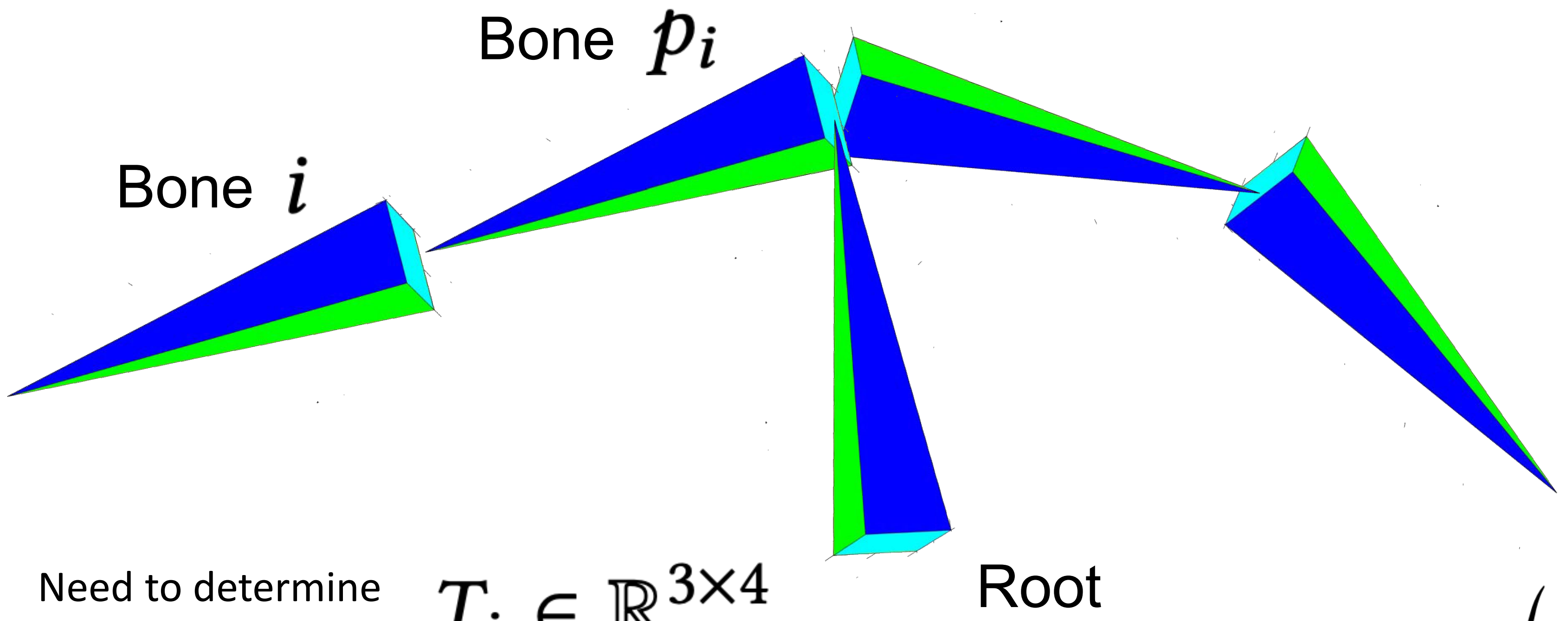
Gimbal Lock!



$$A = BCD$$



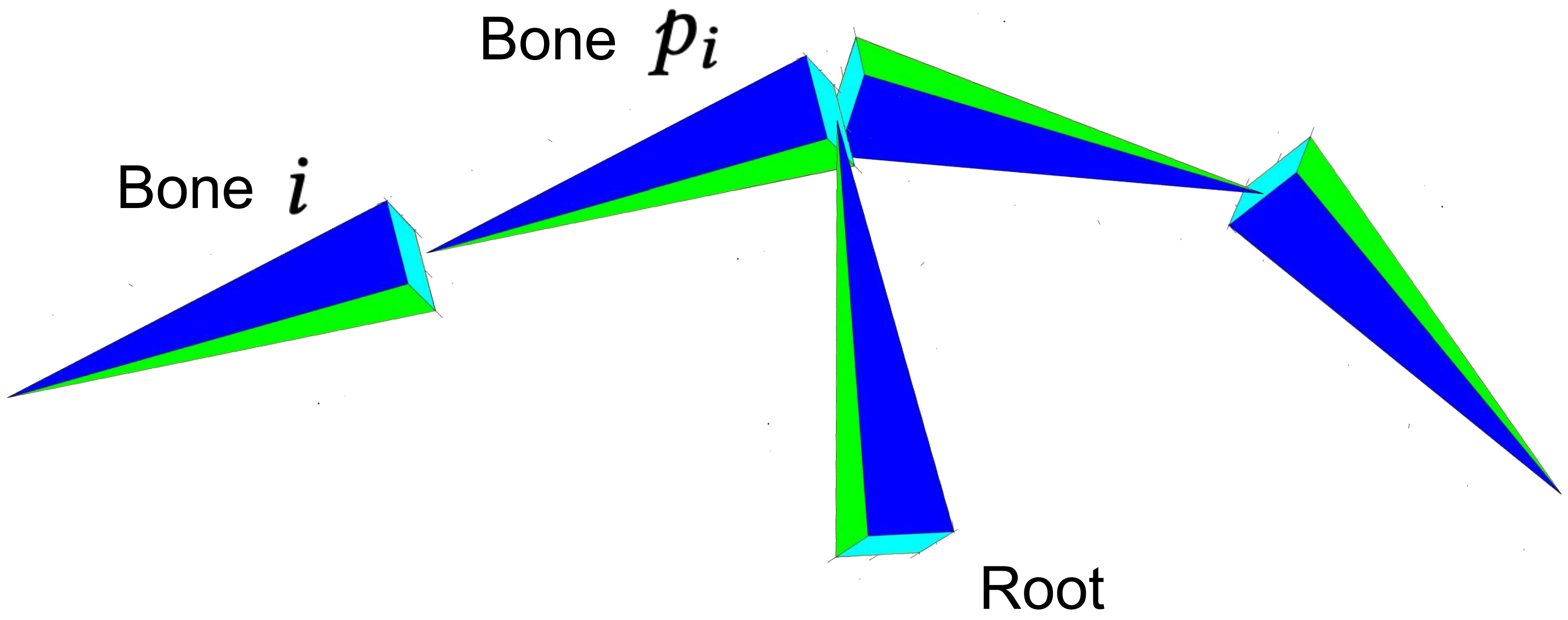
https://en.wikipedia.org/wiki/Euler_angles
https://mathworld.wolfram.com/EulerAngles.html
https://en.wikipedia.org/wiki/Gimbal
https://en.wikipedia.org/wiki/Gimbal_lock#In_three_dimensions



Need to determine
T for each bone

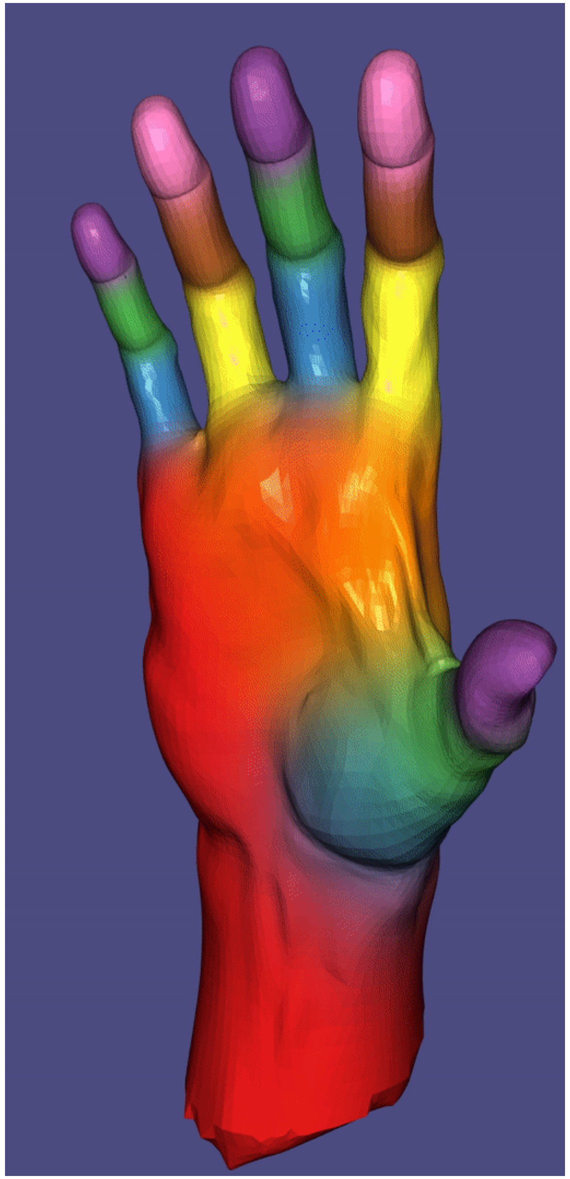
$$T_i \in \mathbb{R}^{3\times 4}$$

$$\mathbf{T}_i = \mathbf{T}_{p_i} \begin{pmatrix} & \widehat{\mathbf{T}}_i & \\ 0\ 0\ 0\ 1 \end{pmatrix} \begin{pmatrix} & & & 0 \\ & \overline{\mathbf{R}}_i & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} & \widehat{\mathbf{T}}_i & \\ 0\ 0\ 0\ 1 \end{pmatrix}^{-1}$$



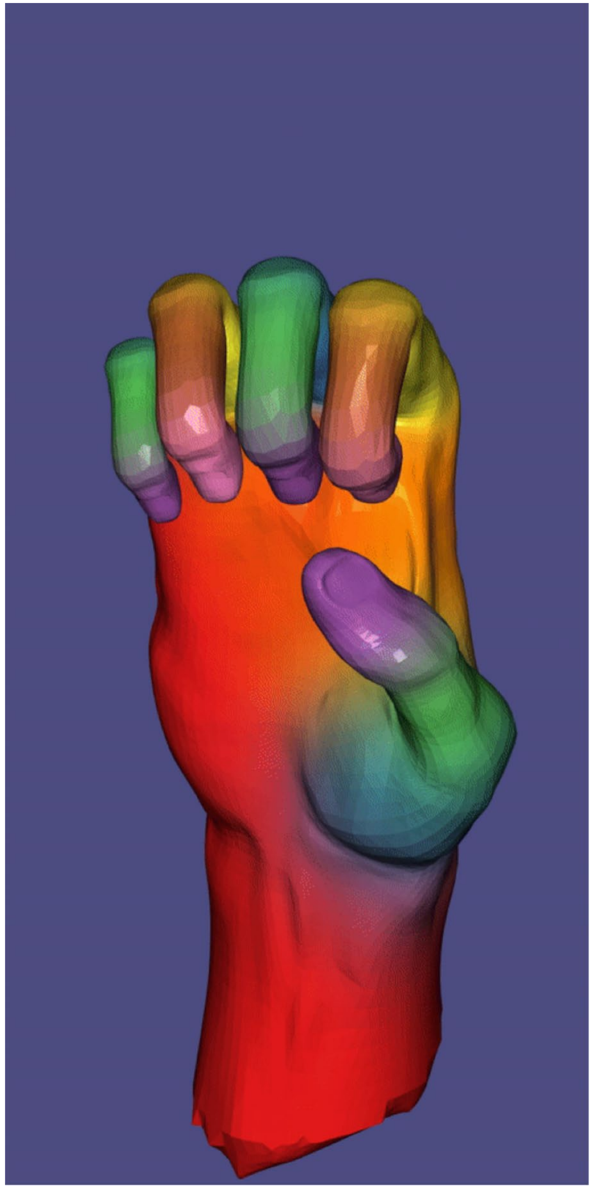
$$\mathbf{T}_i = \mathbf{T}_{p_i}\widehat{\mathbf{T}}_i \begin{pmatrix} \mathbf{R}_x(\theta_{i3}) & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ 0\ 0\ 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{R}_z(\theta_{i2}) & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ 0\ 0\ 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{R}_x(\theta_{i1}) & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ 0\ 0\ 0 & 1 \end{pmatrix} \widehat{\mathbf{T}}_i^{-1}$$

Linear Blend Skinning



$$\mathbf{v}_j = \sum_{i=1}^{\#\text{bones}} w_{ij} \mathbf{T}_i \hat{\mathbf{v}}_j$$

Specifying Keyframes



Time = 0

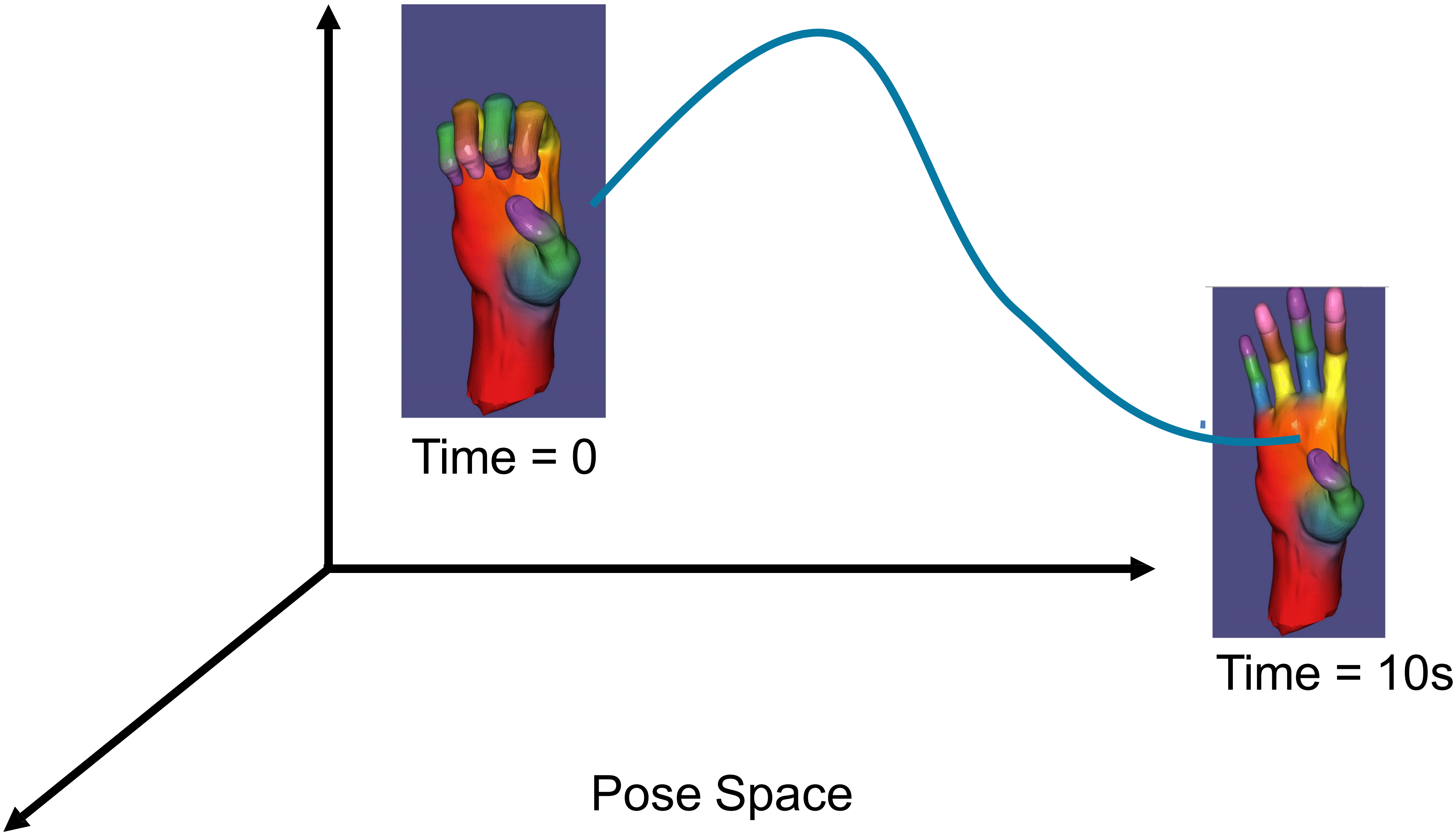
Time = 10s

Poses are generated by specifying rotations of bones

Each pose can be represented as

$$\left(t, \begin{bmatrix}\theta_{i1}\\ \theta_{i2}\\ \theta_{i3}\end{bmatrix}\right)$$

Specifying Keyframes



Smoother Animation: Interpolating Keyframes

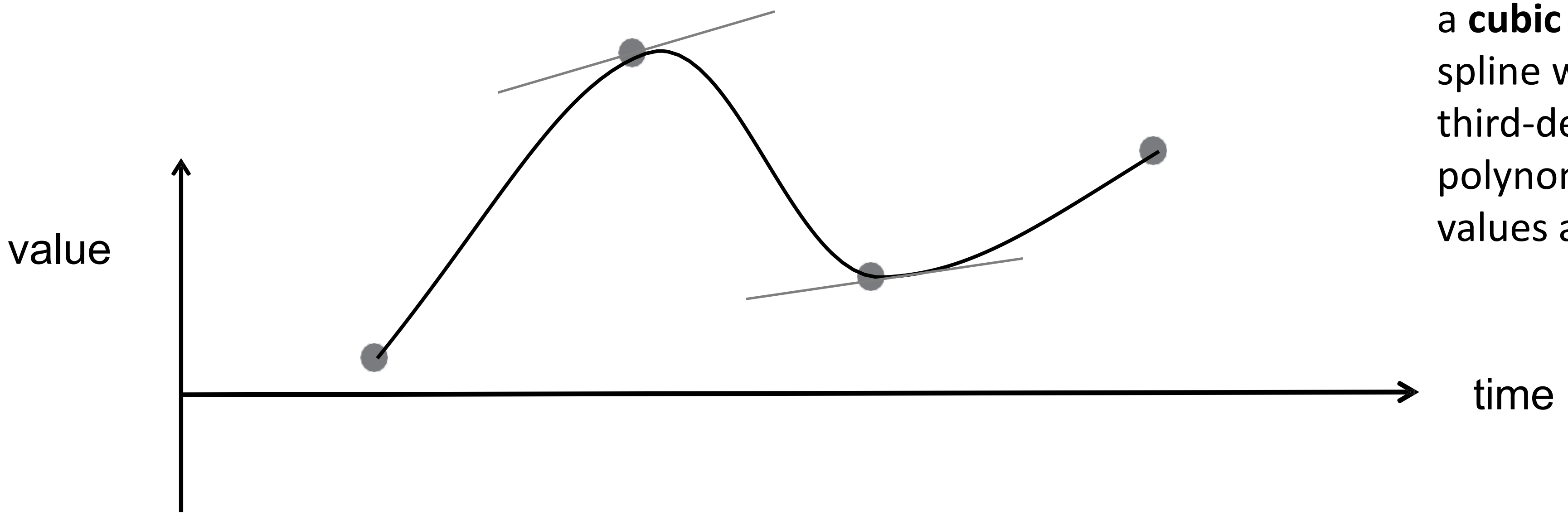
$\theta = \mathbf{c}(t)$ is a curve in the pose space

How could we construct such a curve from keyframes ?

Interpolating Keyframes

$\theta = \mathbf{c}(t)$ is a curve in the pose space

How could we construct such a curve from keyframes ?



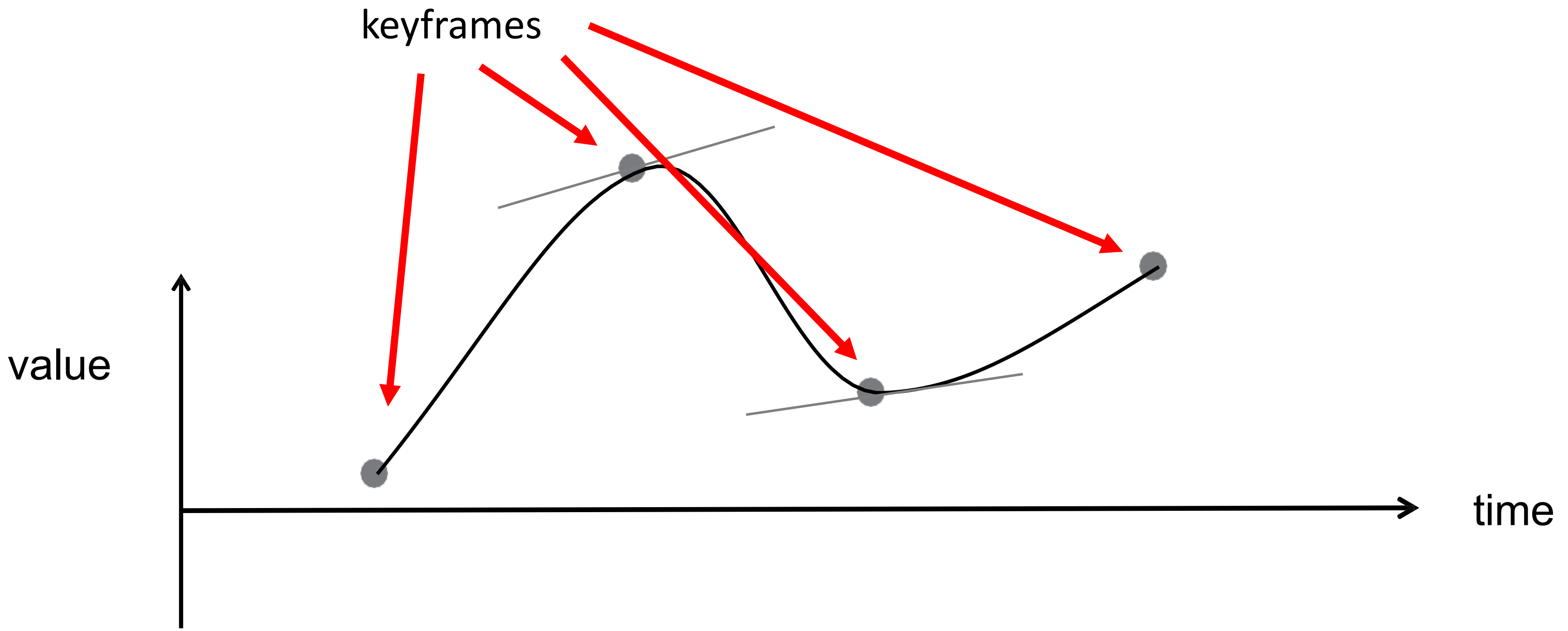
a **cubic Hermite spline** is a spline where each piece is a third-degree
polynomial specified by its values and first derivatives

https://en.wikipedia.org/wiki/Spline_(mathematics)
https://en.wikipedia.org/wiki/Cubic_Hermite_spline

Interpolating Keyframes

$\theta = \mathbf{c}(t)$ is a curve in the pose space

How could we construct such a curve from keyframes ?

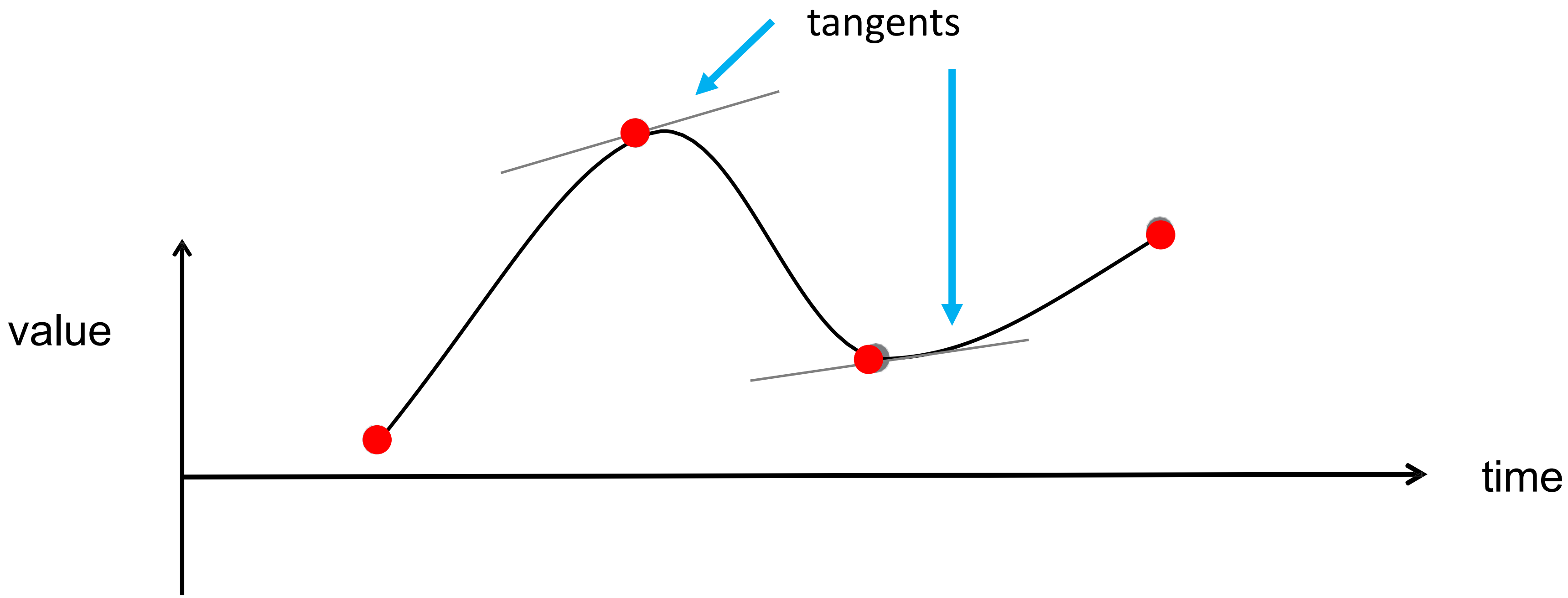


https://en.wikipedia.org/wiki/Spline_(mathematics)
https://en.wikipedia.org/wiki/Cubic_Hermite_spline

Interpolating Keyframes

$\theta = \mathbf{c}(t)$ is a curve in the pose space

How could we construct such a curve from keyframes ?

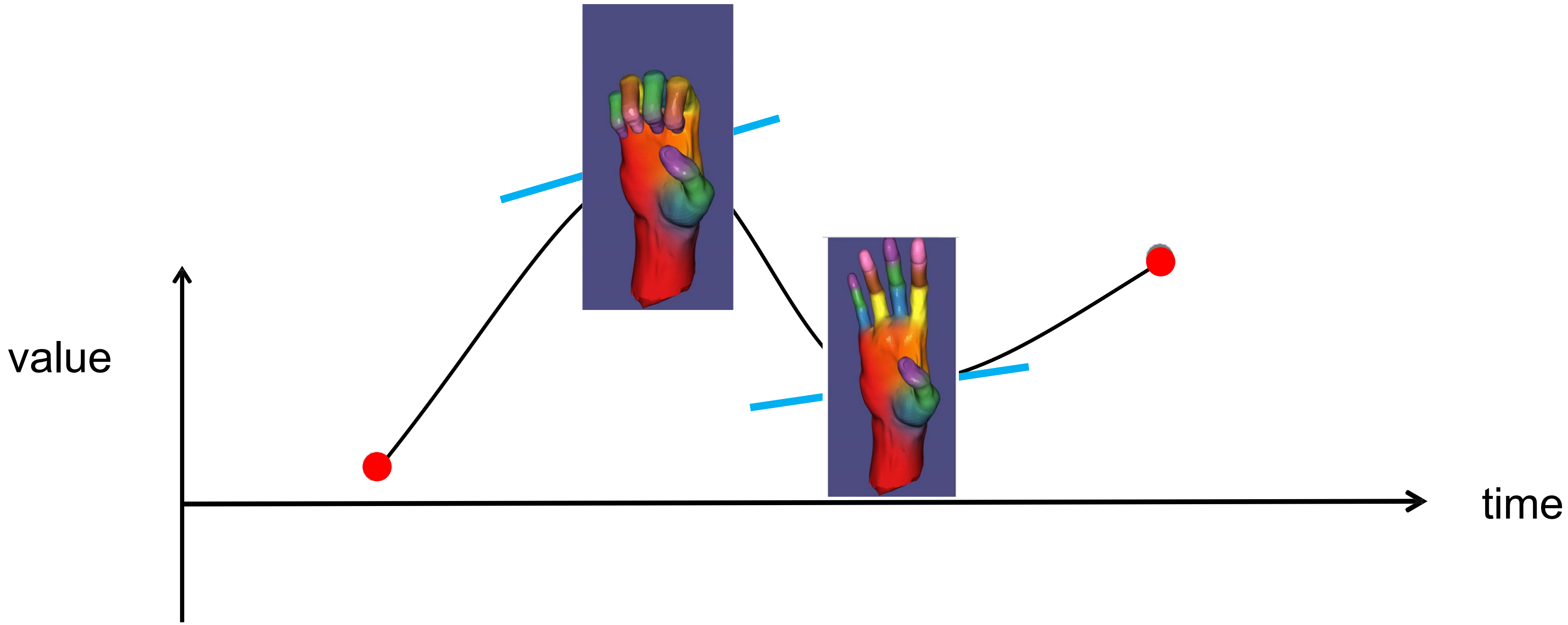


https://en.wikipedia.org/wiki/Spline_(mathematics)
https://en.wikipedia.org/wiki/Cubic_Hermite_spline

Interpolating Keyframes

$\theta = \mathbf{c}(t)$ is a curve in the pose space

How could we construct such a curve from keyframes ?



https://en.wikipedia.org/wiki/Spline_(mathematics)
https://en.wikipedia.org/wiki/Cubic_Hermite_spline

Catmull-Rom Spline

A **cubic** curve created by specifying the end points and the tangents of the curve.

$$\mathbf{c}(t) = at^3 + bt^2 + ct + d$$

Catmull-Rom Spline

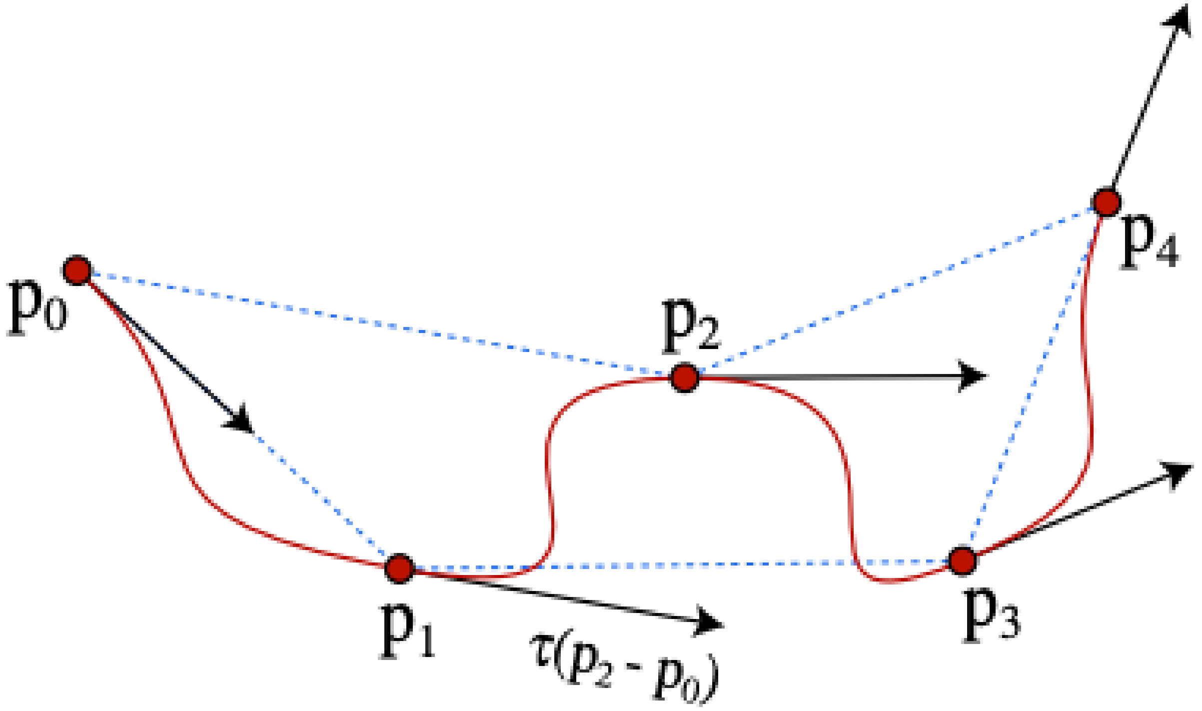
A **cubic** curve created by specifying the end points and the tangents of the curve.

$$\mathbf{c}(t) = at^3 + bt^2 + ct + d$$

$$\mathbf{c}'(t) = 3at^2 + 2bt + c$$

Catmull-Rom Spline

A **cubic** curve created by specifying the end points and the tangents of the curve.



http://graphics.cs.cmu.edu/nsp/course/15-462/Fall04/assts/catmullRom.pdf

Catmull-Rom Spline

$$\mathbf{c}(t) = at^3 + bt^2 + ct + d$$

$$\mathbf{c}'(t) = 3at^2 + 2bt + c$$

Catmull-Rom Spline

$$\mathbf{c}(t) = at^3 + bt^2 + ct + d$$

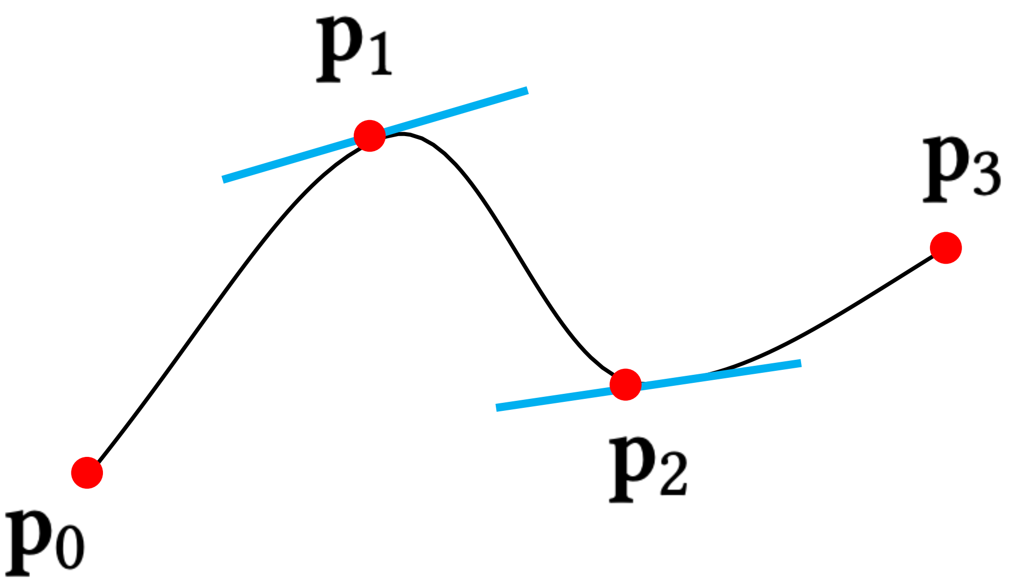
$$\mathbf{c}'(t) = 3at^2 + 2bt + c$$

$$\mathbf{c}(t) = \begin{bmatrix} d & c & b & a \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} \qquad \mathbf{c}'(t) = \begin{bmatrix} d & c & b & a \end{bmatrix} \begin{bmatrix} 3t^2 \\ 2 \\ 1 \\ 0 \end{bmatrix}$$

Catmull-Rom Spline

$$\mathbf{c}(t) = at^3 + bt^2 + ct + d$$

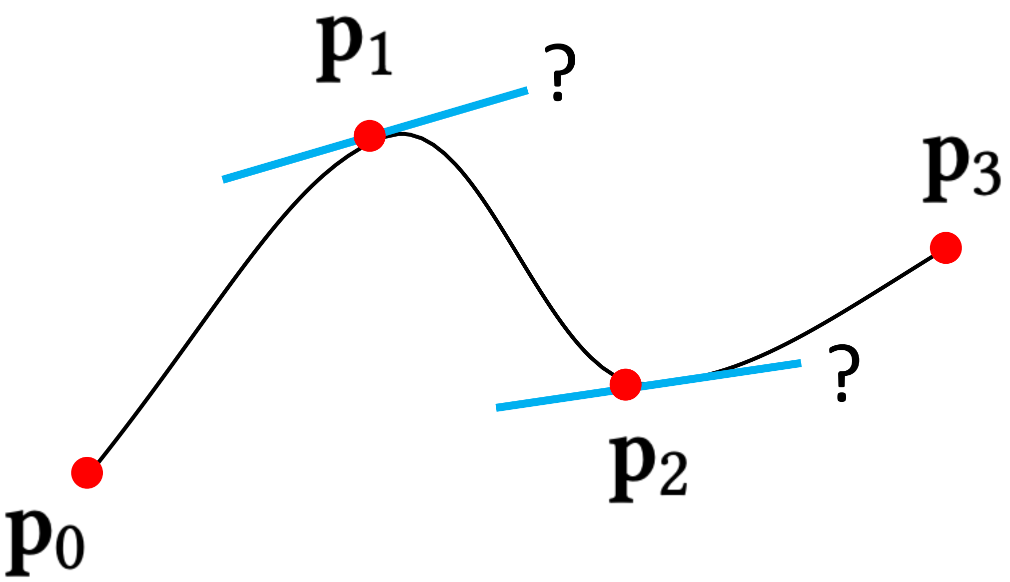
$$\mathbf{c}'(t) = 3at^2 + 2bt + c$$



Catmull-Rom Spline

$$\mathbf{c}(t) = at^3 + bt^2 + ct + d$$

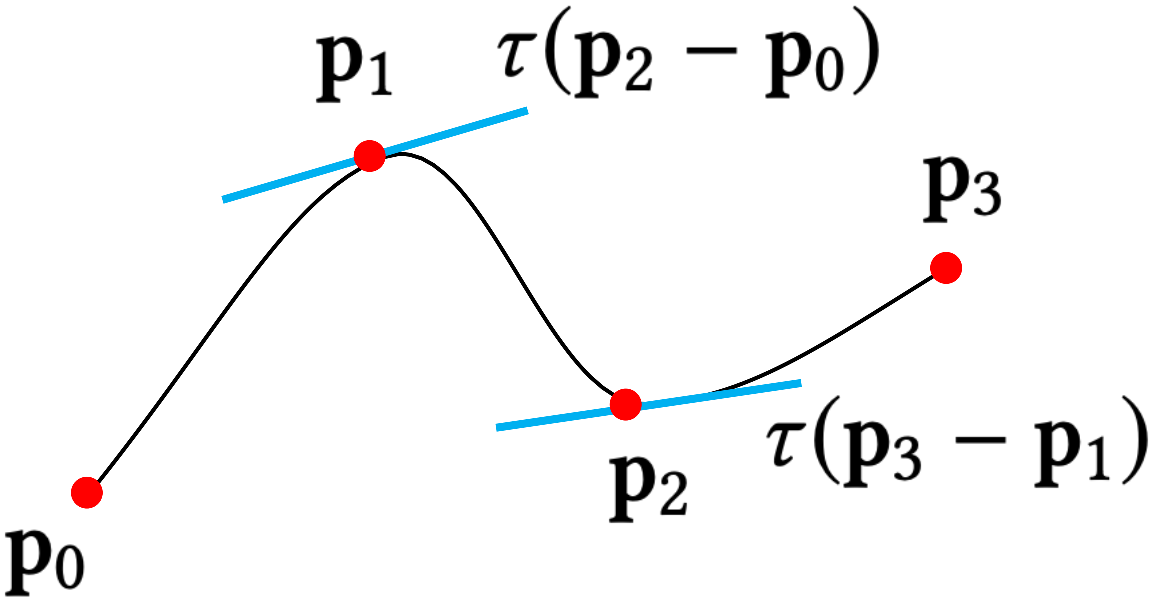
$$\mathbf{c}'(t) = 3at^2 + 2bt + c$$



Catmull-Rom Spline

$$\mathbf{c}(t) = at^3 + bt^2 + ct + d$$

$$\mathbf{c}'(t) = 3at^2 + 2bt + c$$



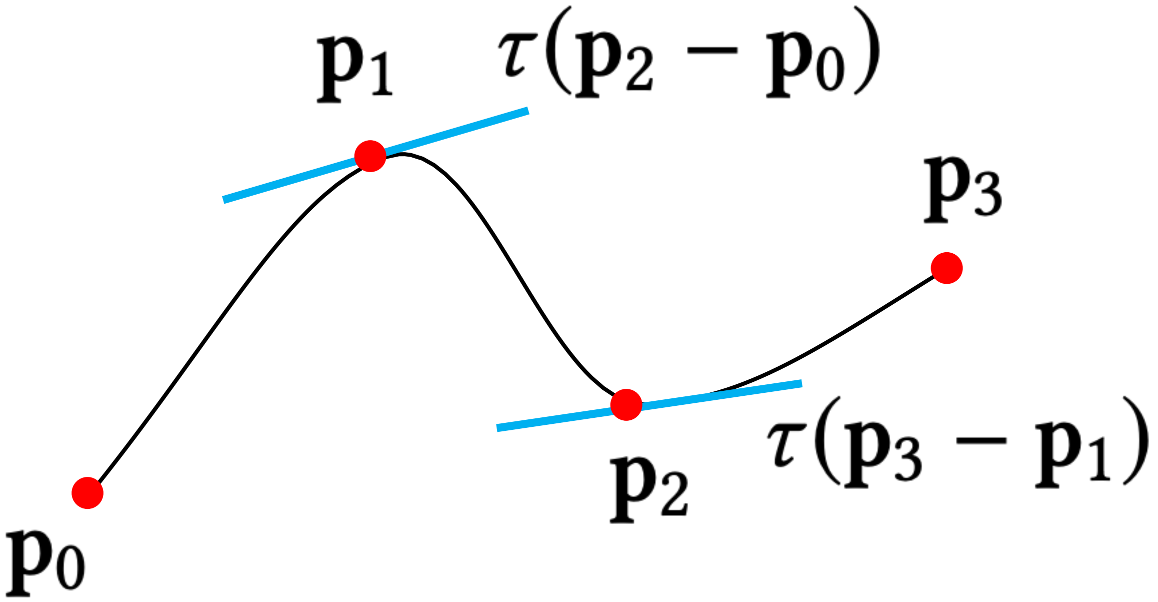
See textbook section 15.3.4 Basis Matrices for Cubics!

http://graphics.cs.cmu.edu/nsp/course/15-462/Fall04/assts/catmullRom.pdf

Catmull-Rom Spline

$$\mathbf{c}(t) = at^3 + bt^2 + ct + d$$

$$\mathbf{c}'(t) = 3at^2 + 2bt + c$$



$$\mathbf{c}(0) = d$$

$$\mathbf{c}(1) = a + b + c + d$$

$$\mathbf{c}'(0) = c$$

$$\mathbf{c}'(1) = 3a + 2b + c$$

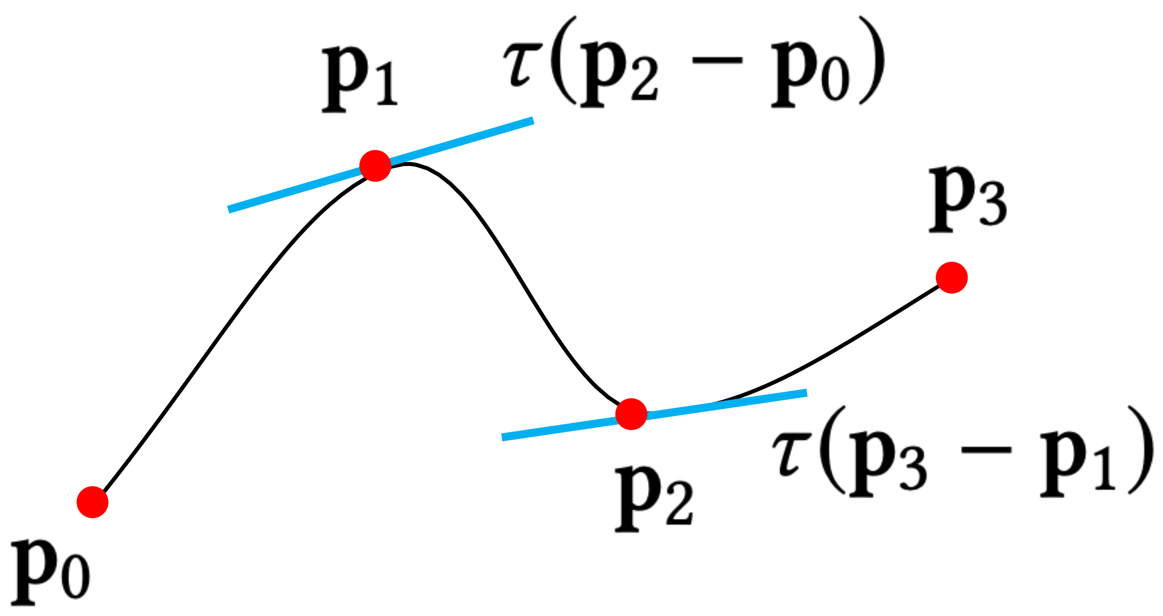
See textbook section 15.3.4 Basis Matrices for Cubics!

http://graphics.cs.cmu.edu/nsp/course/15-462/Fall04/assts/catmullRom.pdf

Catmull-Rom Spline

$$\mathbf{c}(t) = at^3 + bt^2 + ct + d$$

$$\mathbf{c}'(t) = 3at^2 + 2bt + c$$



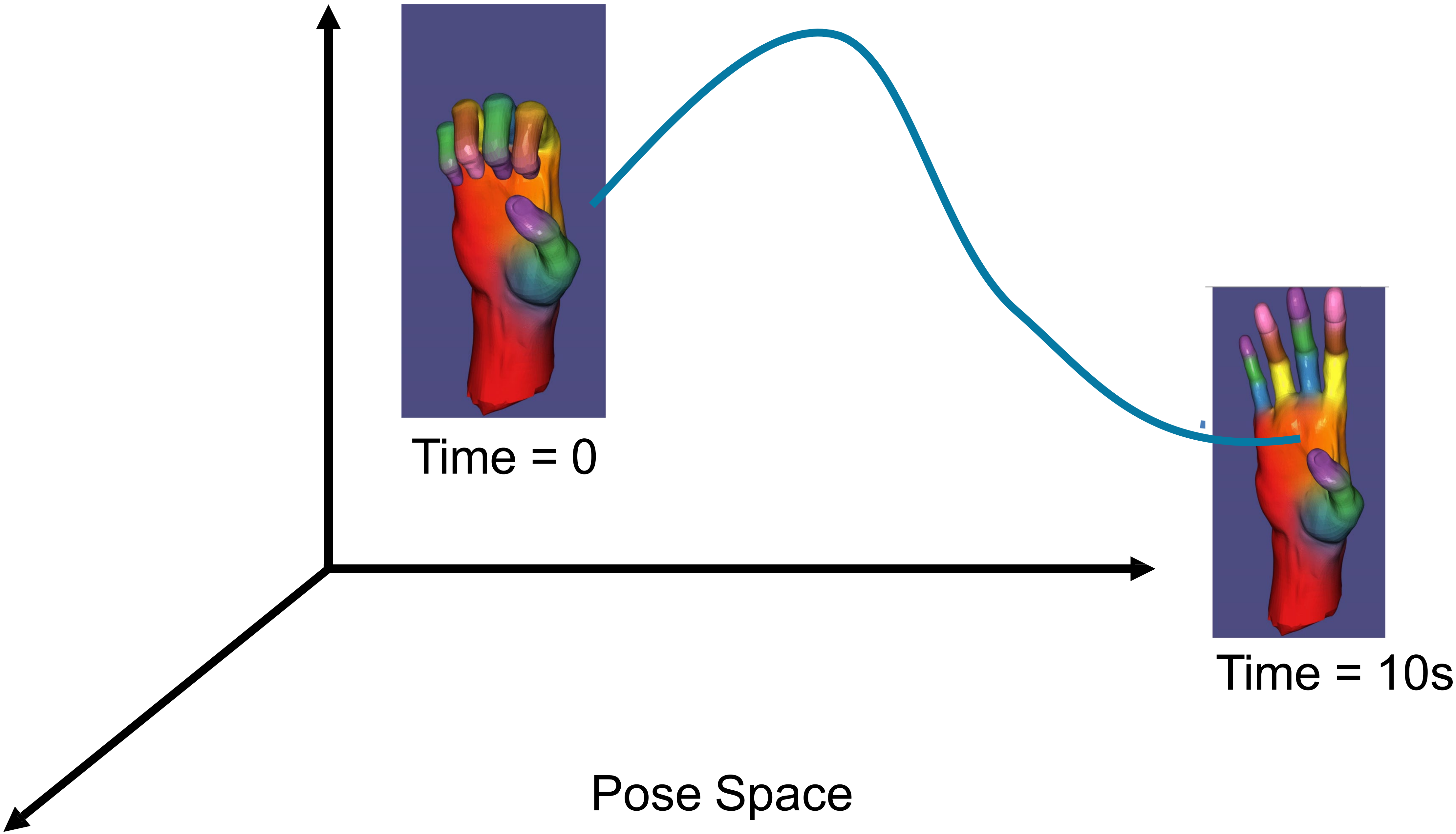
$$\mathbf{c}(0) = d$$
$$\mathbf{c}(1) = a + b + c + d$$
$$\mathbf{c}'(0) = c$$
$$\mathbf{c}'(1) = 3a + 2b + c$$

$$\mathbf{c}(0) = \mathbf{p}_1$$
$$\mathbf{c}(1) = \mathbf{p}_2$$
$$\mathbf{c}'(0) = \tau(\mathbf{p}_2 - \mathbf{p}_0)$$
$$\mathbf{c}'(1) = \tau(\mathbf{p}_3 - \mathbf{p}_1)$$

See textbook section 15.3.4 Basis Matrices for Cubics!

http://graphics.cs.cmu.edu/nsp/course/15-462/Fall04/assts/catmullRom.pdf

Specifying Keyframes



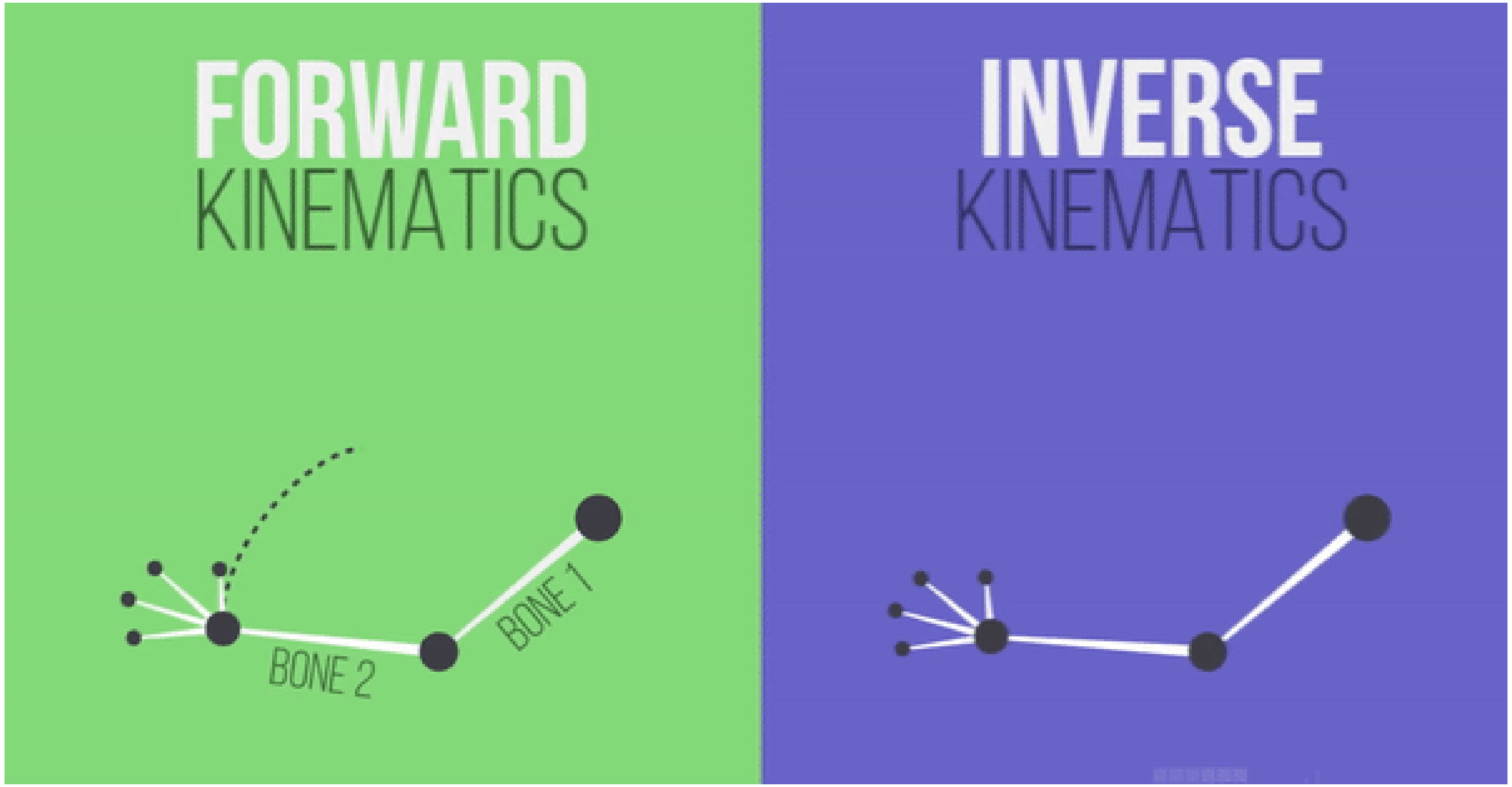
Inverse Kinematics

One last thing …

Posing all those bones can be tedious, wouldn’t it be great if you could just specify a few bones and the rest would be automatically computed?

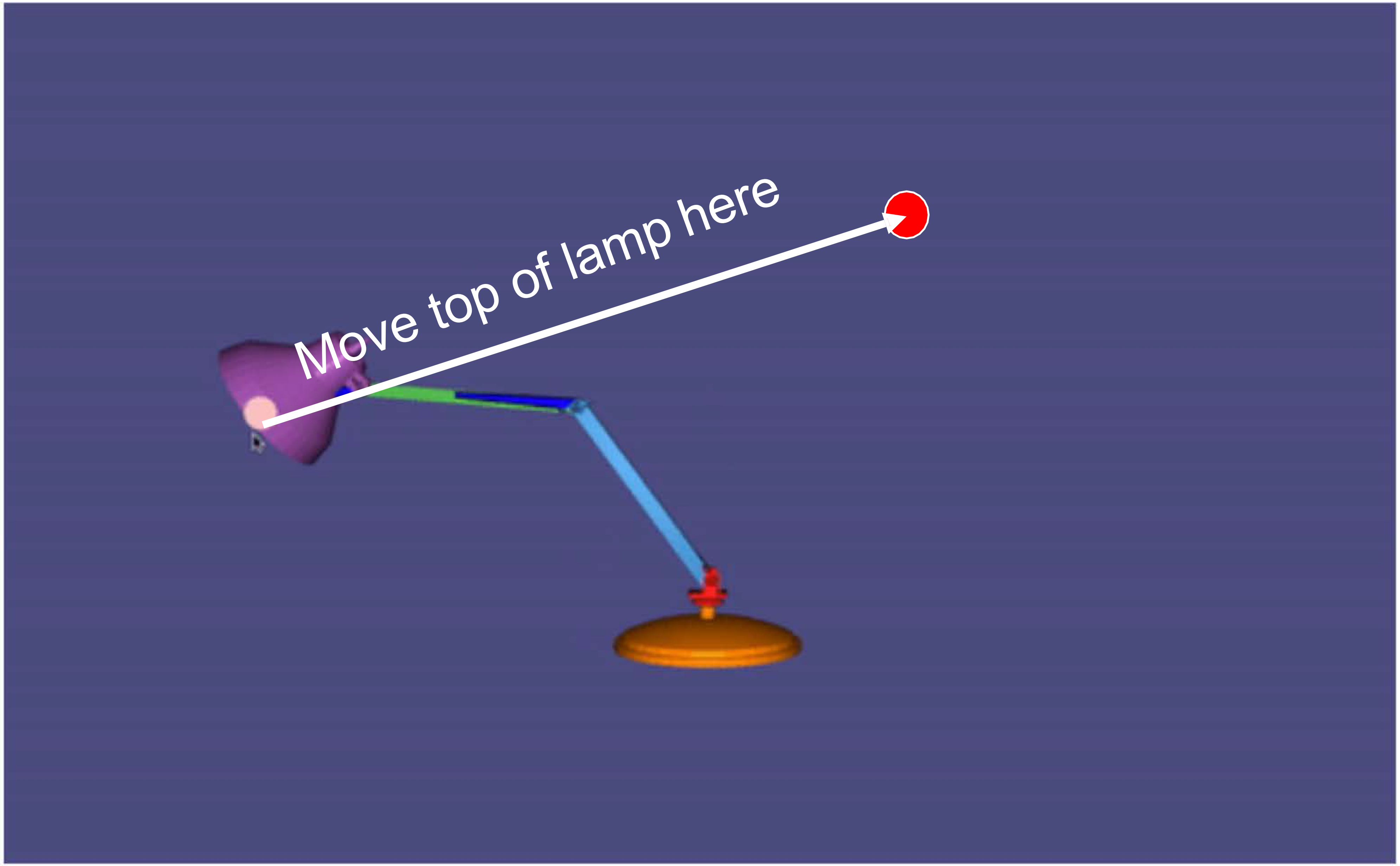
That’s what inverse kinematics does!!

Inverse Kinematics

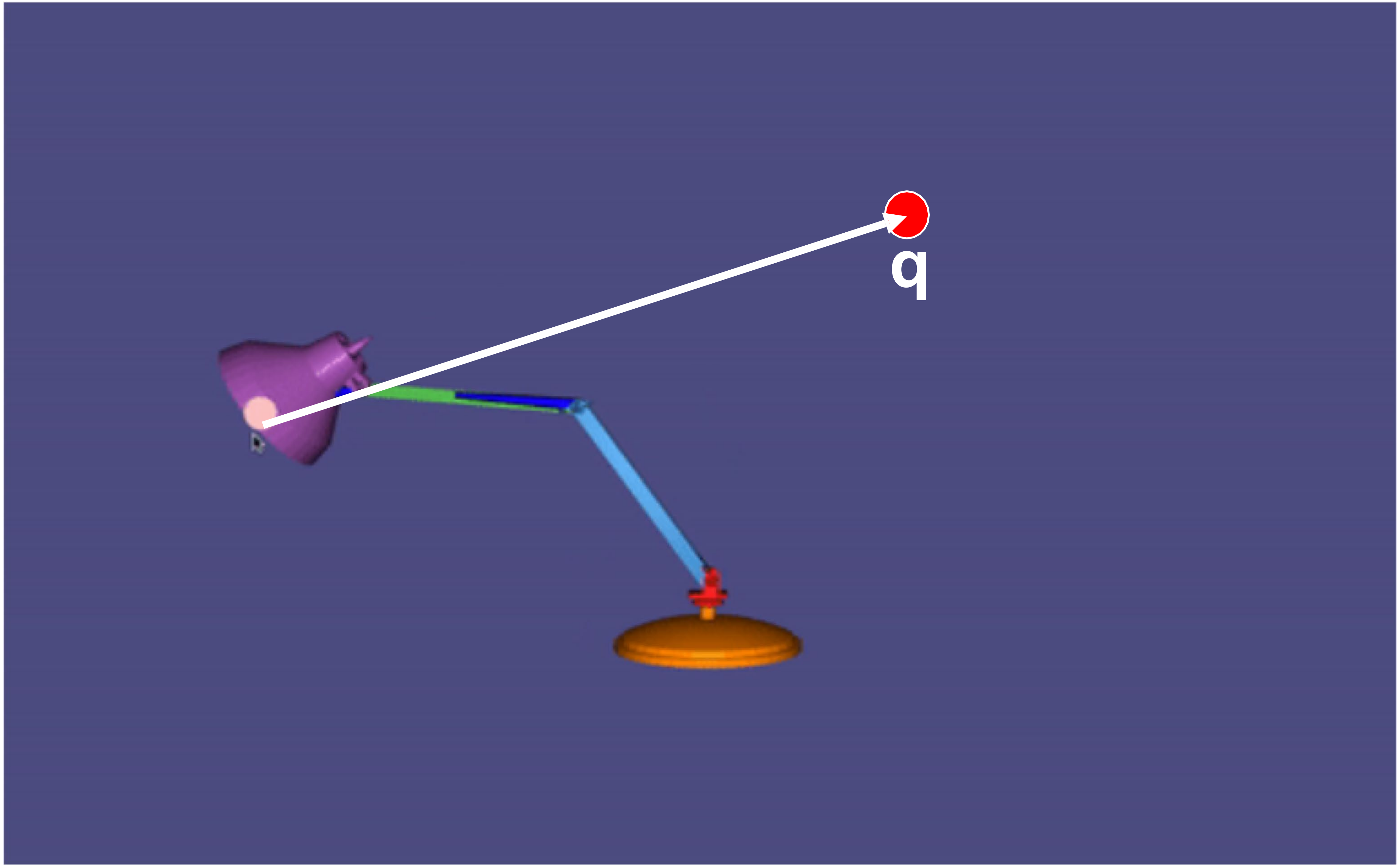


https://www.youtube.com/watch?v=0a9qIj7kwiA

Inverse Kinematics



Inverse Kinematics



Inverse Kinematics

Inverse Kinematics: Optimization Variables

$$\mathbf{a} \in \mathbb{R}^{3m}$$

Stack all the Euler angles
for *m* bones in a vector

$$\mathbf{a} = \begin{pmatrix} \theta_{11} \\ \theta_{12} \\ \theta_{13} \\ \theta_{21} \\ \theta_{22} \\ \theta_{23} \\ \vdots \\ \theta_{m1} \\ \theta_{m2} \\ \theta_{m3} \end{pmatrix}$$

Inverse Kinematics: Energy

Pick an energy:
the squared distance between the pose tip $\mathbf{x_b}$ of some bone b and a desired goal location $\mathbf{q}$

$$E(\mathbf{x}_b(\mathbf{a})) = \|\mathbf{x}_b(\mathbf{a}) - \mathbf{q}\|^2$$

Inverse Kinematics: Energy

Pick an energy:
the squared distance between the pose tip $\mathbf{x}_b$ of some bone b and a desired goal location **q**

$$E(\mathbf{x}_b(\mathbf{a})) = \|\mathbf{x}_b(\mathbf{a}) - \mathbf{q}\|^2$$

list of constrained end effectors

$$b = \{b_1, b_2, \ldots, b_k\}$$

Inverse Kinematics: Energy

Pick an energy:
the squared distance between the pose tip $\mathbf{x}_b$ of some bone b and a desired goal location $\mathbf{q}$

$$E(\mathbf{x}_b(\mathbf{a})) = \|\mathbf{x}_b(\mathbf{a}) - \mathbf{q}\|^2$$

list of constrained end effectors $\rightarrow$

$$b = \{b_1, b_2, \ldots, b_k\}$$

$$\min_{\mathbf{a}} \underbrace{\sum_{i=1}^{k} \|\mathbf{x}_{b_i}(\mathbf{a}) - \widehat{\mathbf{x}}_{b_i}\|^2}_{E(\mathbf{x}_b(\mathbf{a}))}$$

Inverse Kinematics: Energy

$$\min_{\mathbf{a}} \quad \underbrace{\sum_{i=1}^{k} \|\mathbf{x}_{b_i}(\mathbf{a}) - \widehat{\mathbf{x}}_{b_i}\|^2}_{E(\mathbf{x}_b(\mathbf{a}))}$$

Inverse Kinematics: Energy

Over all choices of Euler angles **a**, we want the angles that ensure all selected end effectors go to their prescribed locations.

Position + rotation of bone $\mathbf{x}_{b_i}$ found using forward kinematics

Constrained end effector position

$$\min_{\mathbf{a}} \underbrace{\sum_{i=1}^{k} \|\mathbf{x}_{b_i}(\mathbf{a}) - \hat{\mathbf{x}}_{b_i}\|^2}_{E(\mathbf{x}_b(\mathbf{a}))}$$

Sum over all constrained end effectors

Inverse Kinematics: Energy

Over all choices of Euler angles **a**, we want the angles that ensure all selected end effectors go to their prescribed locations.

$$\min_{\mathbf{a}} \quad \underbrace{\sum_{i=1}^{k} \|\mathbf{x}_{b_i}(\mathbf{a}) - \widehat{\mathbf{x}}_{b_i}\|^2}_{E(\mathbf{x}_b(\mathbf{a}))}$$

And we will further constrain our angles a to have minimum and maximum limits.

$$\min_{\mathbf{a}^{\min} \leq \mathbf{a} \leq \mathbf{a}^{\max}} \quad E(\mathbf{x}_b(\mathbf{a}))$$

Gradient Descent

We are *minimizing* an energy.

Make an initial guess.

iteratively improve the guess by moving in a direction that decreases!

Gradient Descent

Recall that the gradient of a function is given by

Points in direction of maximum ascent

$$\nabla f(\mathbf{x}) = (\frac{\partial f}{\partial \mathbf{x}_1}, \frac{\partial f}{\partial \mathbf{x}_2}, \ldots \frac{\partial f}{\partial \mathbf{x}_n})$$

Gradient Descent

So let's take a step in the *negative* gradient direction of the objective

$$\mathbf{a} \leftarrow \mathbf{a} - \sigma \left(\frac{dE(\mathbf{x}(\mathbf{a}))}{d\mathbf{a}} \right)^T$$

Gradient Descent

So let's take a step in the negative gradient direction of the objective

$$\mathbf{a} \leftarrow \mathbf{a} - \sigma \left(\frac{dE(\mathbf{x}(\mathbf{a}))}{d\mathbf{a}} \right)^T$$

$$\mathbf{a} \leftarrow \mathbf{a} - \sigma \left(\frac{d\mathbf{x}(\mathbf{a})}{d\mathbf{a}} \right)^T \left(\frac{dE(\mathbf{x})}{d\mathbf{x}} \right)$$

Gradient Descent

So let's take a step in the negative gradient direction of the objective

$$\mathbf{a} \leftarrow \mathbf{a} - \sigma \left(\frac{dE(\mathbf{x}(\mathbf{a}))}{d\mathbf{a}} \right)^T$$

$$\mathbf{a} \leftarrow \mathbf{a} - \sigma \boxed{\left(\frac{d\mathbf{x}(\mathbf{a})}{d\mathbf{a}} \right)^T} \left(\frac{dE(\mathbf{x})}{d\mathbf{x}} \right)$$

Gradient Descent: Kinematic Jacobian

$$\mathbf{a} \leftarrow \mathbf{a} - \sigma \boxed{\left(\frac{d\mathbf{x}(\mathbf{a})}{d\mathbf{a}}\right)^T} \left(\frac{dE(\mathbf{x})}{d\mathbf{x}}\right)$$

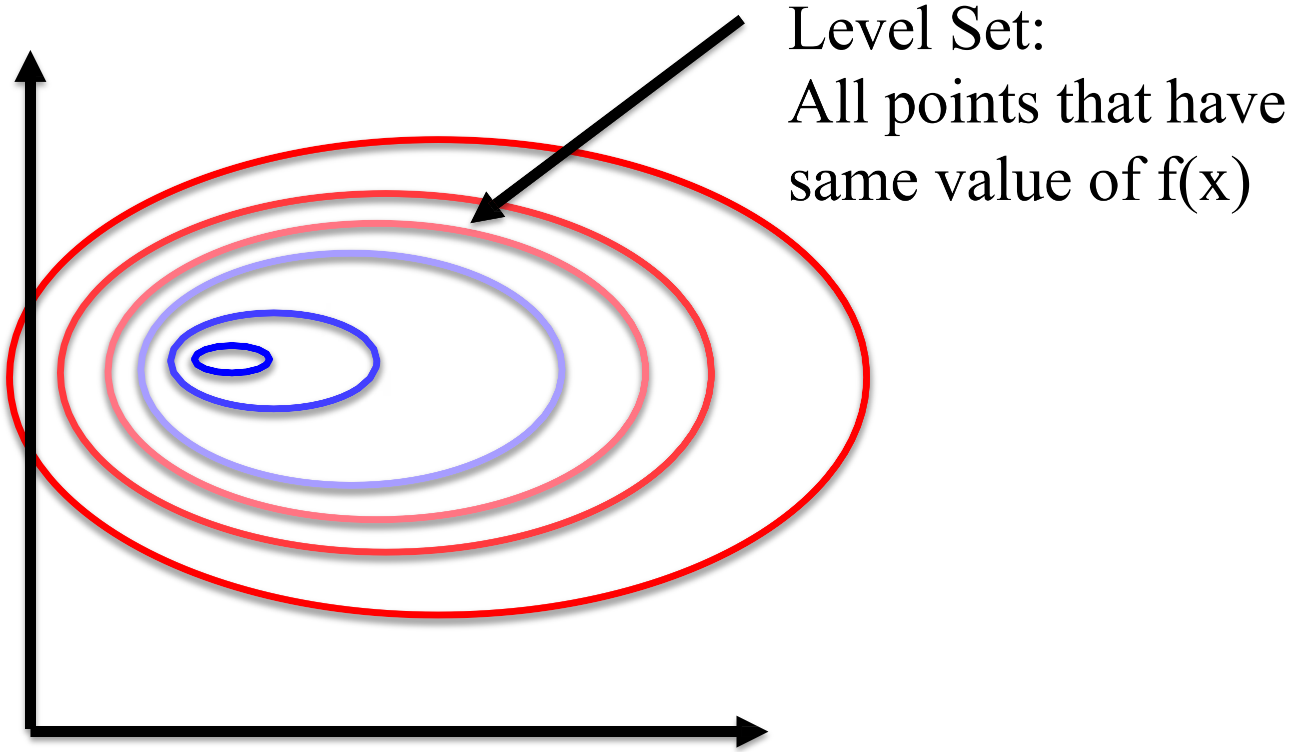
The change in tip positions **x** with respect to joint angles **a**

$$\mathbf{a} \leftarrow \mathbf{a} - \sigma \mathbf{J}^{\mathrm{T}} \left(\frac{dE(\mathbf{x})}{d\mathbf{x}}\right)$$

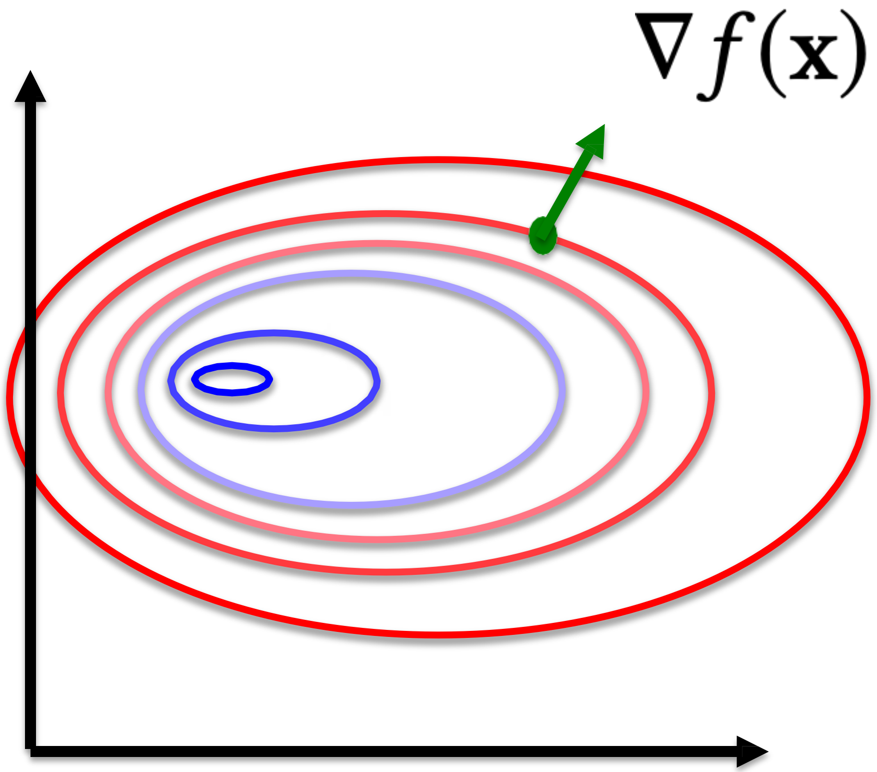
Computed using finite differences

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

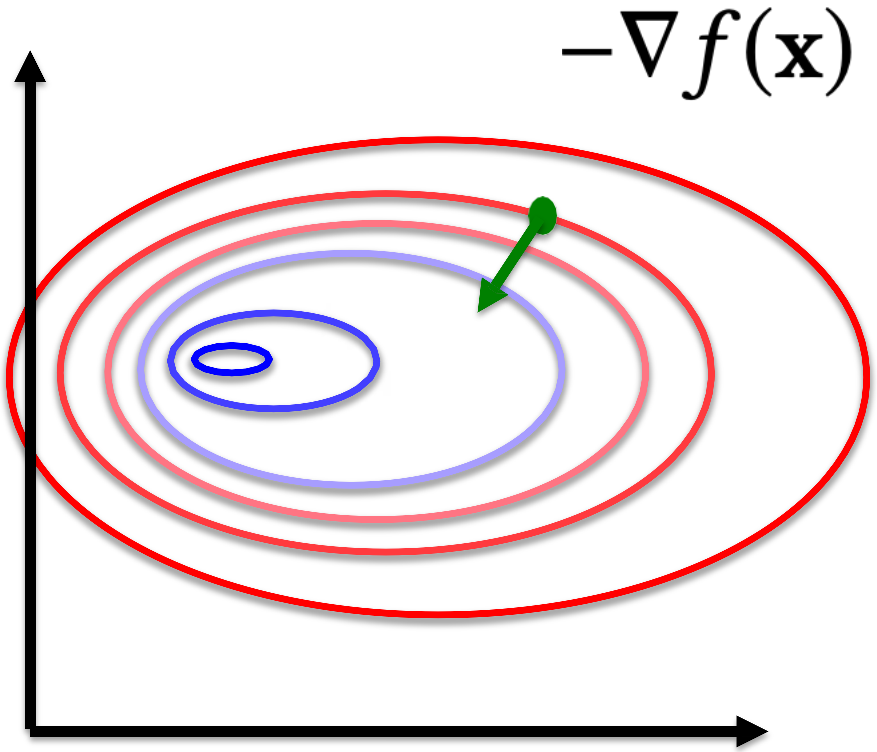
An Aside: Level Sets



Gradient Descent

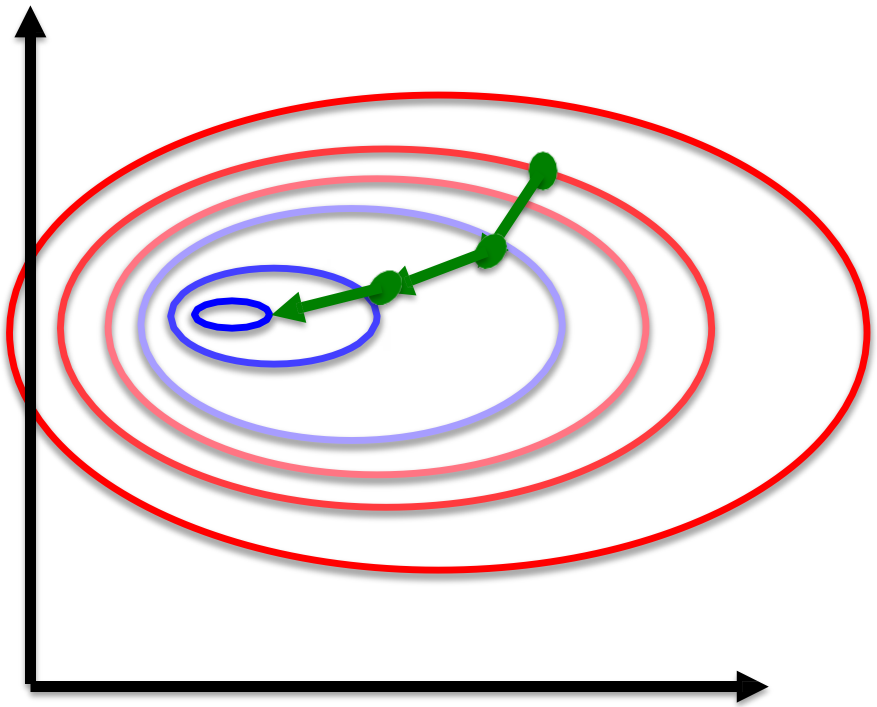


Gradient Descent



Gradient Descent

Keep moving in that direction



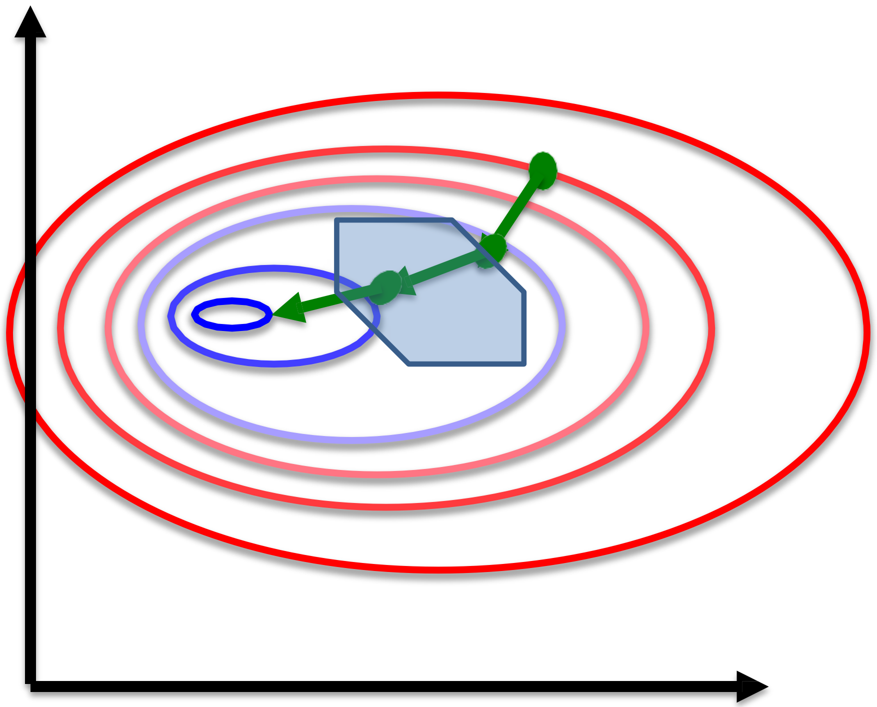
Projected Gradient Descent

After each step, project onto our feasible set of solutions by snapping **a** values to their bounds if needed

$$\mathbf{a}_i \leftarrow \max[\mathbf{a}_i^{\text{min}}, \min[\mathbf{a}_i^{\text{max}}, \mathbf{a}_i]]$$

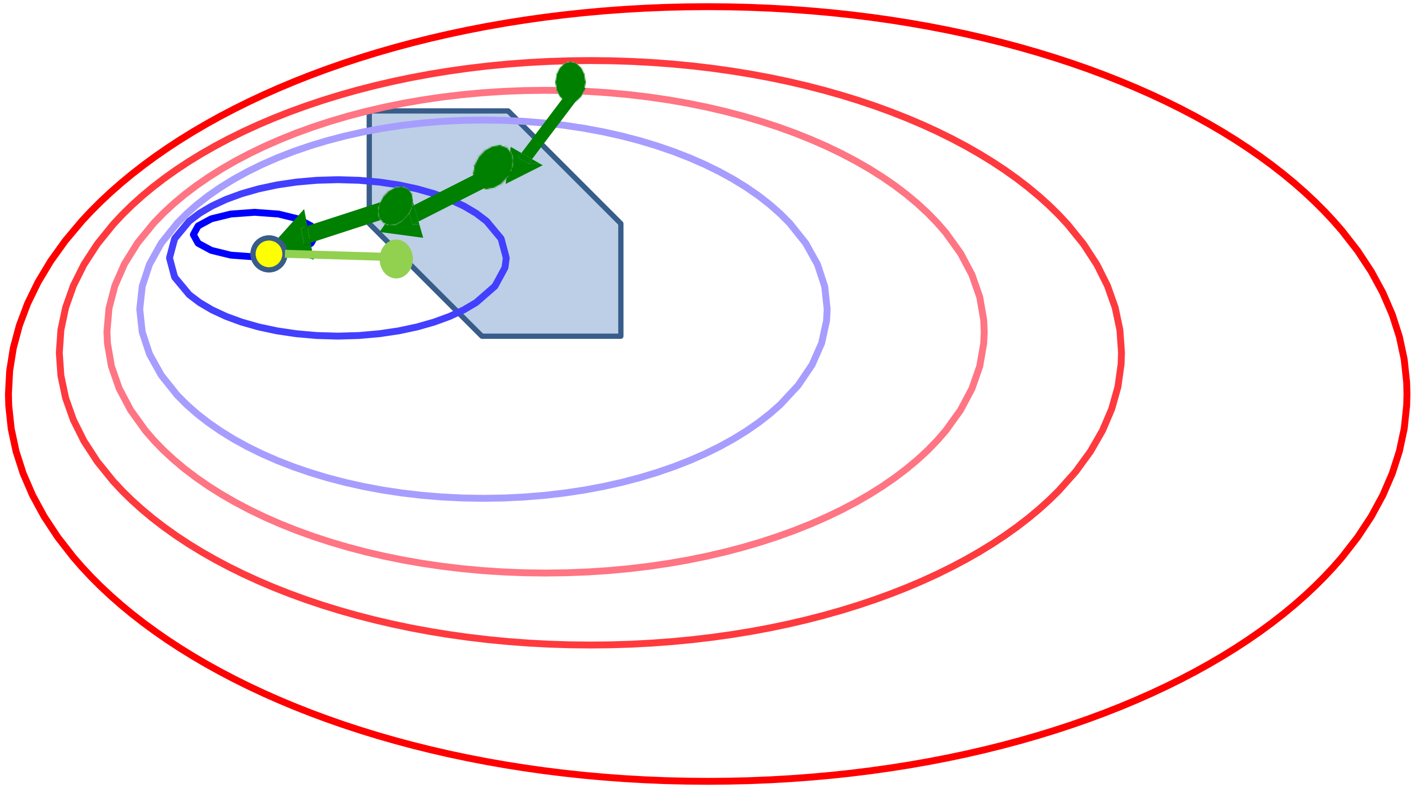
Projected Gradient Descent

Keep moving in that direction



Projected Gradient Descent

Keep moving in that direction



Line Search

$$\mathbf{a} \leftarrow \mathbf{a} - \sigma \mathbf{J}^{\top} \left(\frac{dE(\mathbf{x})}{d\mathbf{x}} \right)$$

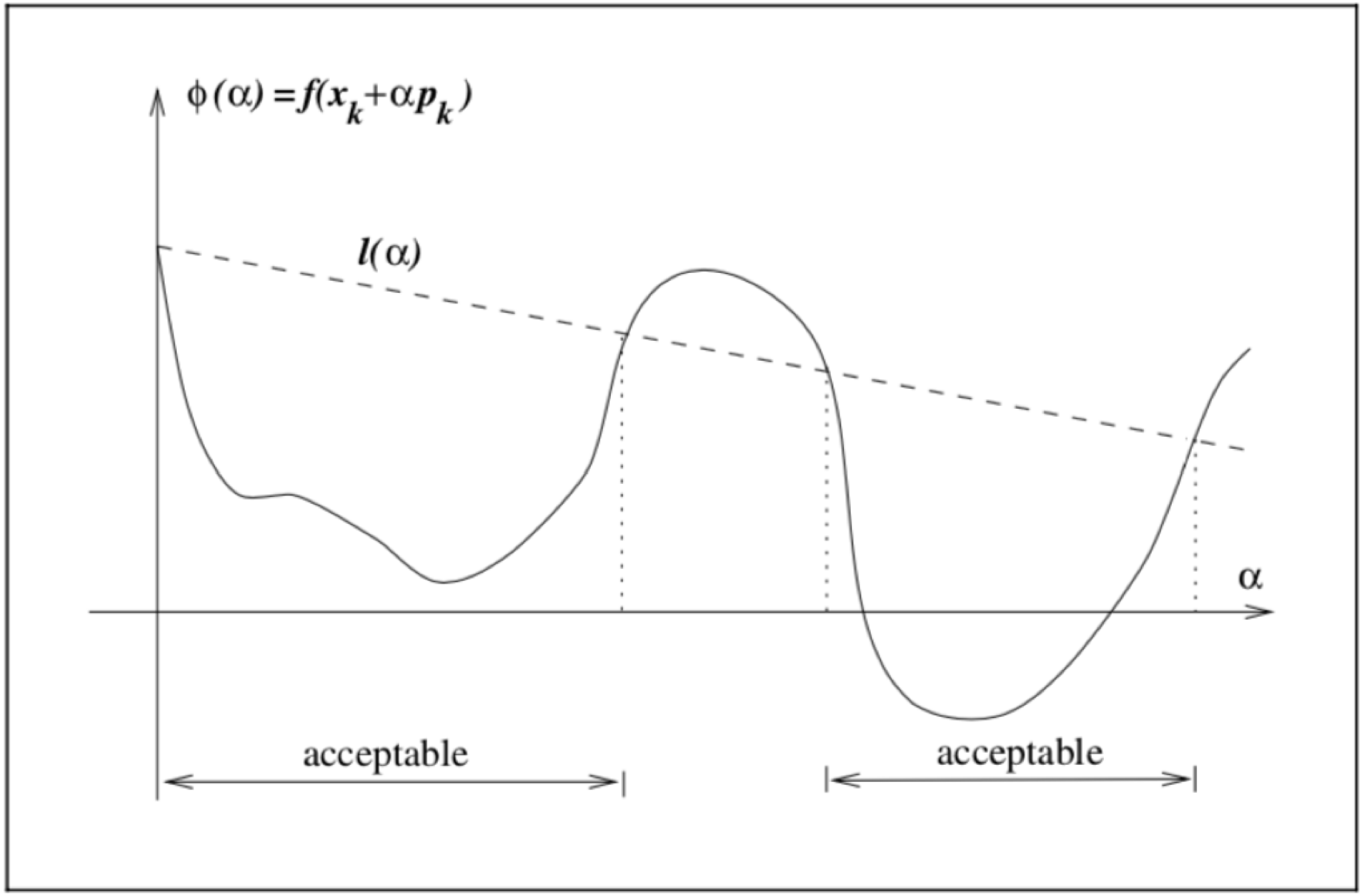
https://en.wikipedia.org/wiki/Backtracking_line_search#Algorithm

Line Search

$$\mathbf{a} \leftarrow \mathbf{a} - \boxed{\sigma}\mathbf{J}^{\mathsf{T}} \left(\frac{dE(\mathbf{x})}{d\mathbf{x}} \right)$$

https://en.wikipedia.org/wiki/Backtracking_line_search#Algorithm

When Good Optimizations Go Bad



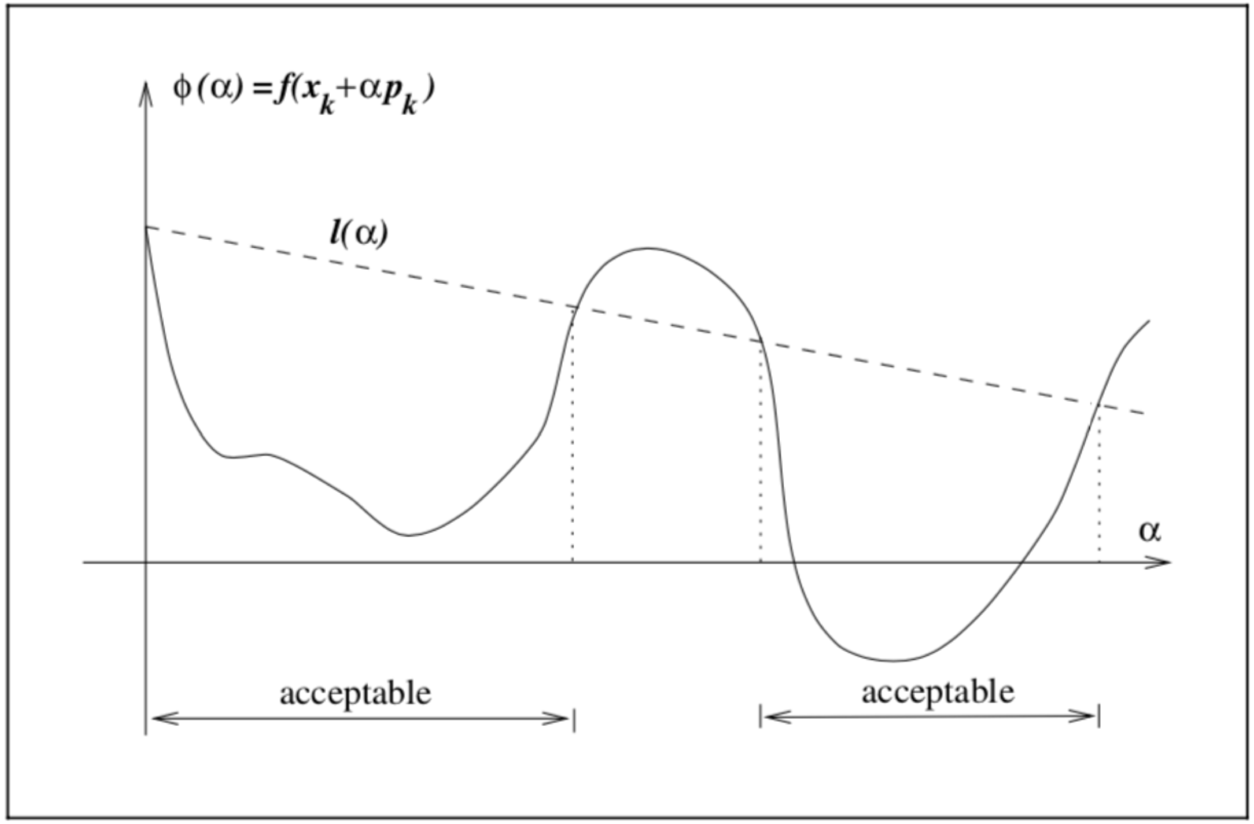
Numerical Optimization – Nocedal and Wright, pg. 33

Line Search

$$\mathbf{a} \leftarrow \mathbf{a} - \boxed{\sigma}\mathbf{J}^{\mathsf{T}}\left(\frac{dE(\mathbf{x})}{d\mathbf{x}}\right)$$

AKA we are moving in descent direction and then projecting:

$$\mathbf{a} \leftarrow \mathrm{proj}(\mathbf{a} + \Delta\mathbf{a})$$



https://en.wikipedia.org/wiki/Backtracking_line_search#Algorithm

Line Search

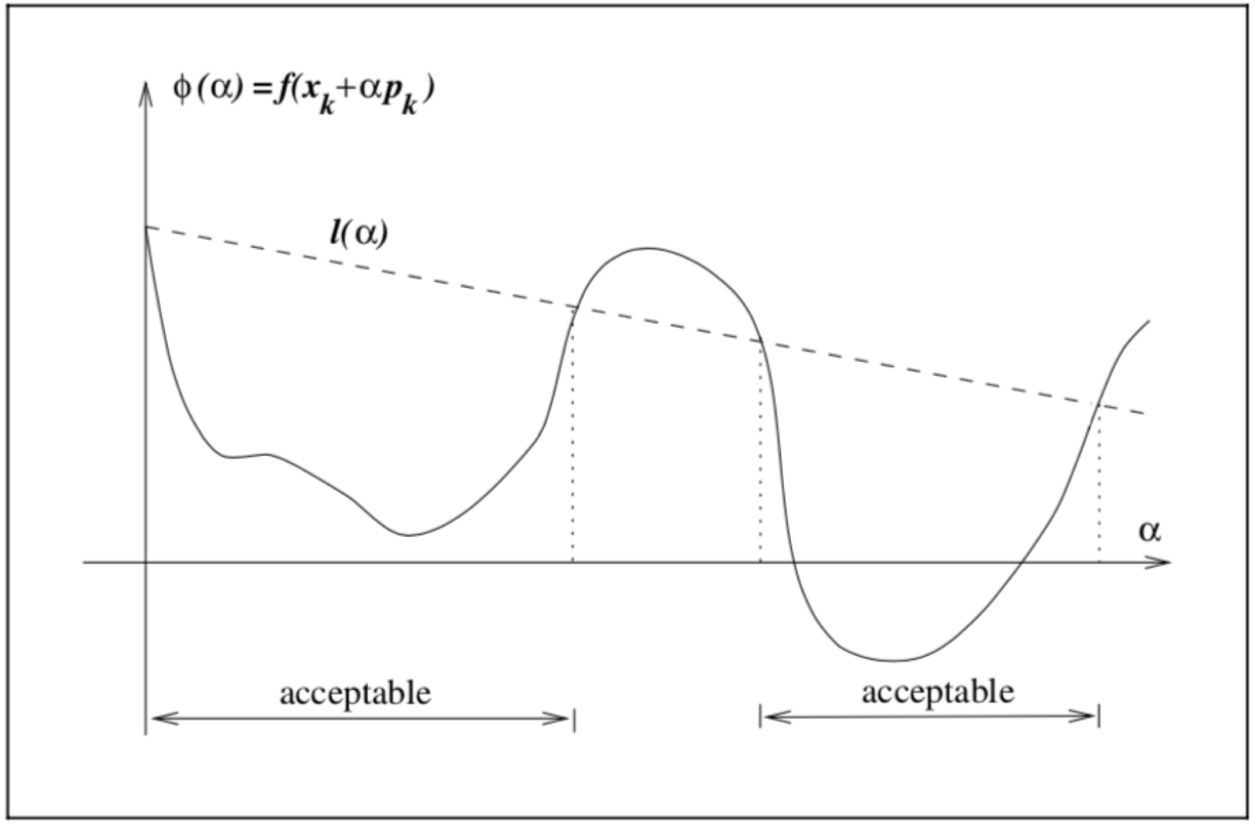
$$\mathbf{a} \leftarrow \mathbf{a} - \boxed{\sigma}\mathbf{J}^{\mathsf{T}}\left(\frac{dE(\mathbf{x})}{d\mathbf{x}}\right)$$

AKA we are moving in descent
direction and then projecting:

$$\mathbf{a} \leftarrow \mathrm{proj}(\mathbf{a} + \Delta\mathbf{a})$$

Start with large σ
and decrease by ½ until

$$E(\mathrm{proj}(\mathbf{a} + \sigma\Delta\mathbf{a})) < E(\mathbf{a})$$



https://en.wikipedia.org/wiki/Backtracking_line_search#Algorithm

Line Search

$$\mathbf{a} \leftarrow \mathbf{a} - \boxed{\sigma}\mathbf{J}^{\mathsf{T}}\left(\frac{dE(\mathbf{x})}{d\mathbf{x}}\right)$$

AKA we are moving in descent
direction and then projecting:

$$\mathbf{a} \leftarrow \mathrm{proj}(\mathbf{a} + \Delta\mathbf{a})$$

Start with large σ
and decrease by ½ until

$$E(\mathrm{proj}(\mathbf{a} + \sigma\Delta\mathbf{a})) < E(\mathbf{a})$$

https://en.wikipedia.org/wiki/Backtracking_line_search#Algorithm

Done for Today

Office hours: now