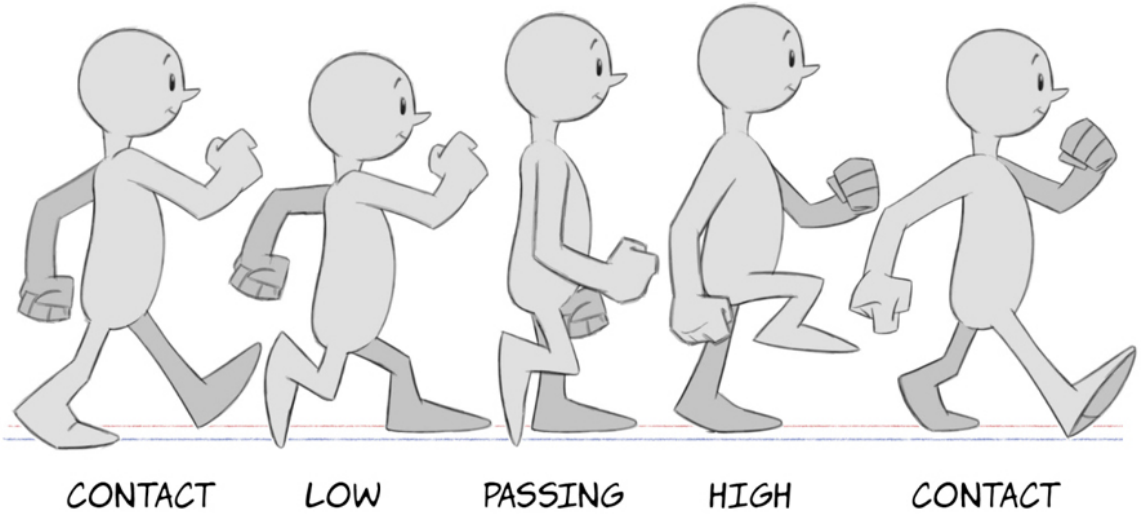
computer animation

CS4515 - 3D Computer Graphics and Animation

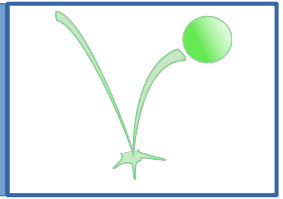


Ricardo Marroquim

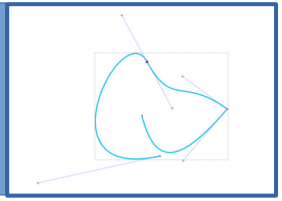
Delft University of Technology (TU Delft)

computer animation

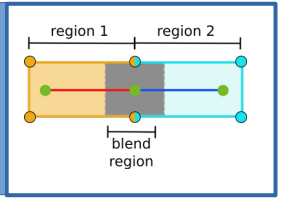
1) general ideas



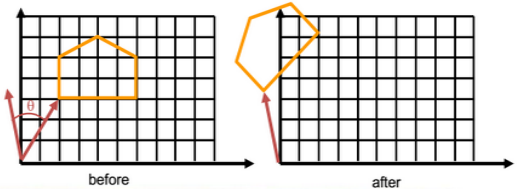
2) computational tools



3) methods



transformations



time tunnel

The Adventures of André & Wally B. (1984)

https://www.youtube.com/watch?v=a_9Tsbduk9E

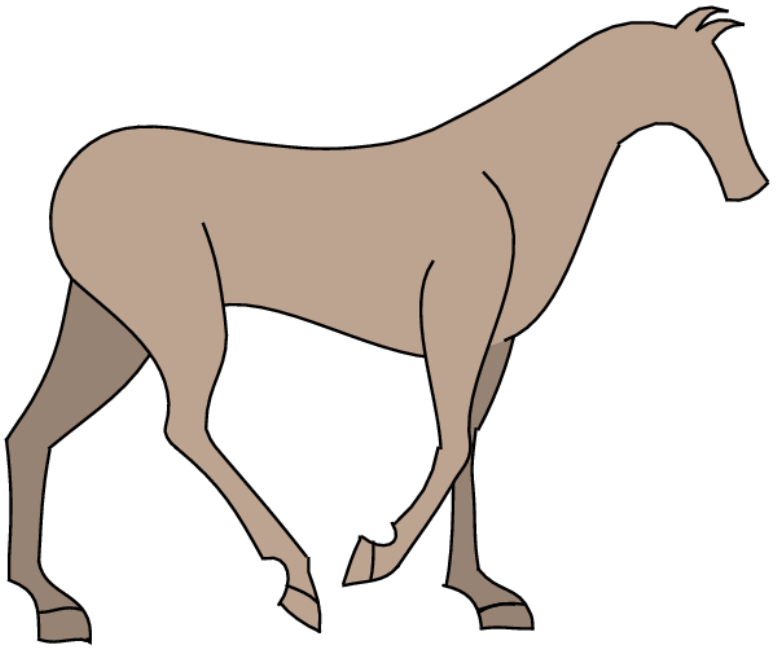
Toy Story 1 (1995)

Toy Story 4 (2019)

Raya and The Last Dragon (2021)

Elemental (2023)

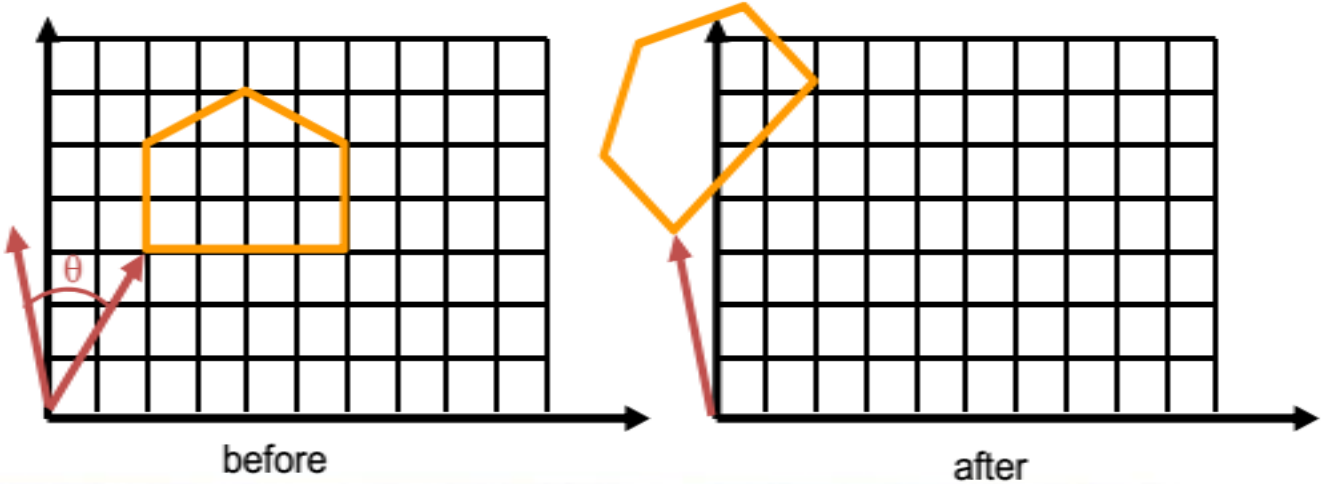
- animation is about movement
  - objects
  - light
  - texture
  - camera



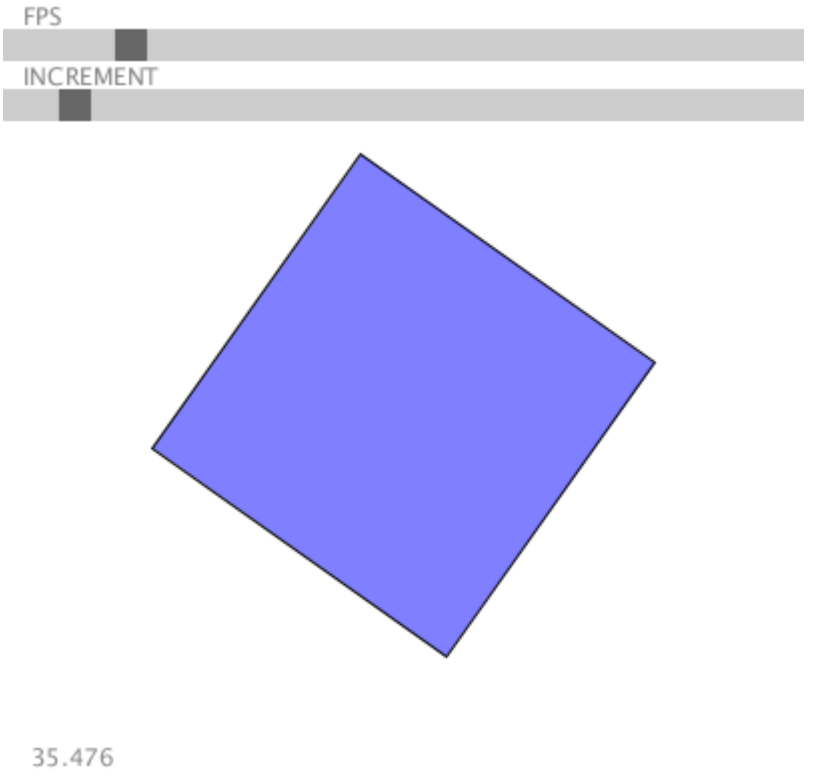
how can we animate?

- you have seen transformations

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

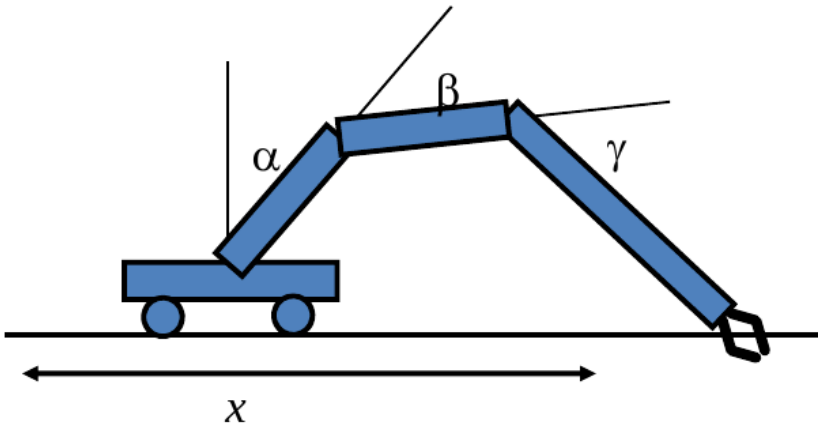


very simple animation demo

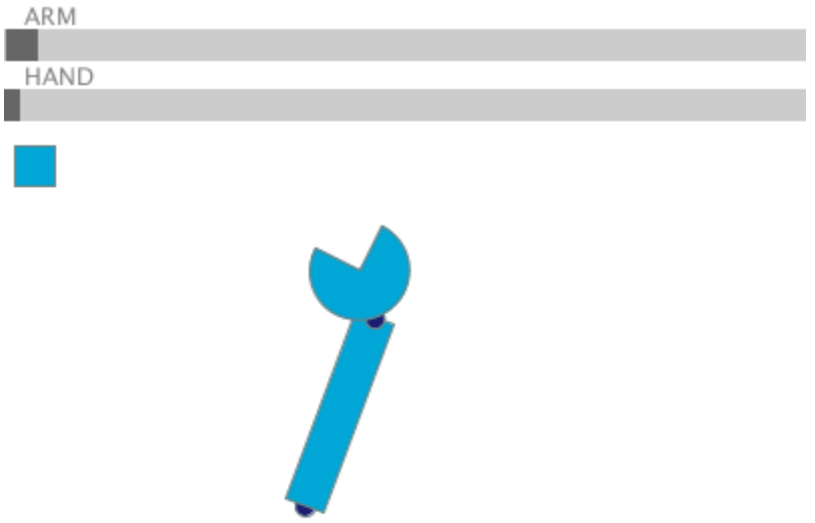


more interesting objects?

- you have seen complex objects



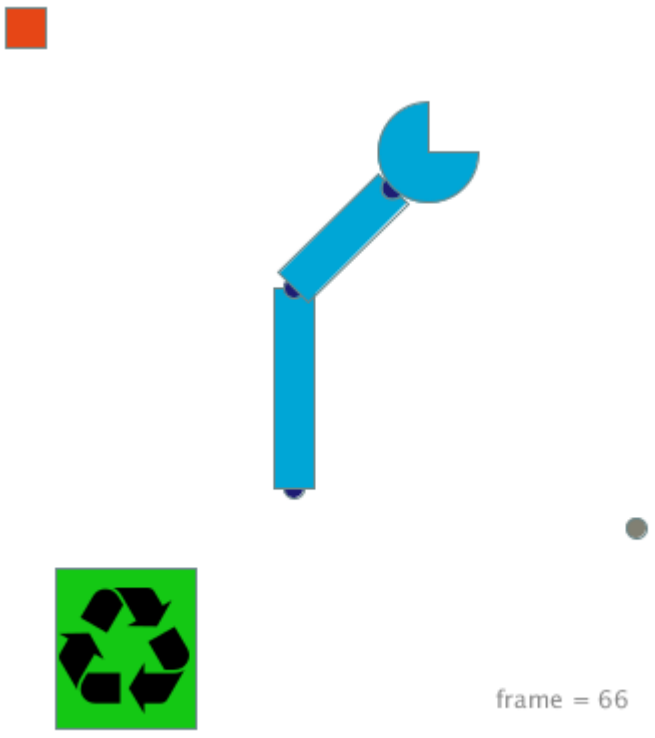
robot arm demo



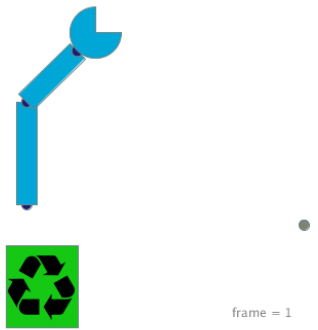
arm angle = 290.52325

hand angle = -38.43602

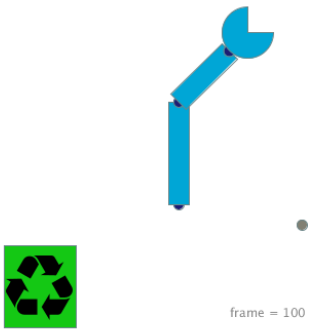
robot story demo



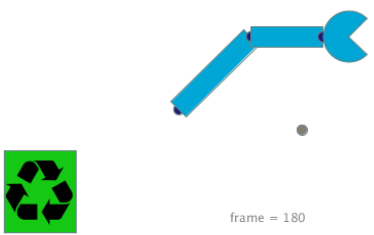
Robot story step by step



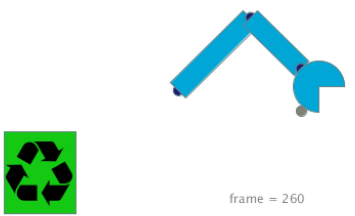
step 1



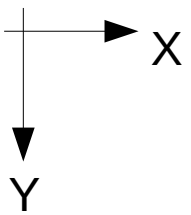
step 2:
translate arm (150,0)



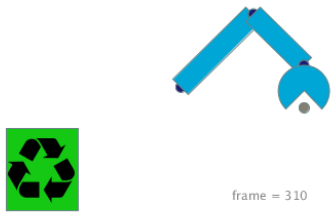
step 3:
rotate arm (45°)



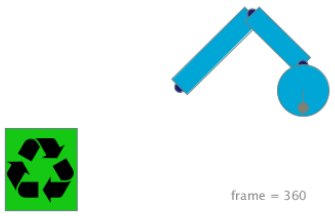
step 4:
rotate forearm (45°)



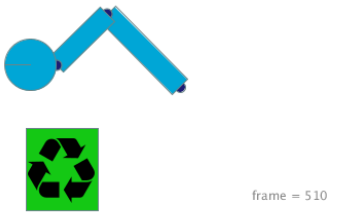
Robot story step by step



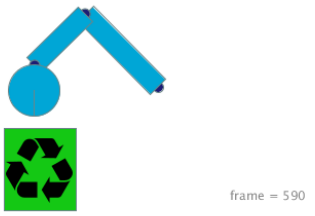
step 5:
rotate hand (45°)



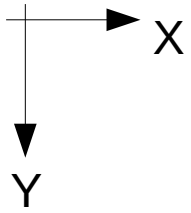
step 6:
hand aperture (-45°)
trash = invisible



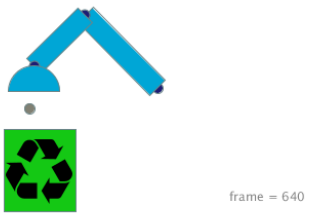
step 7:
translate arm (-20,0)
rotate arm (-90°)
rotate forearm (-180°)



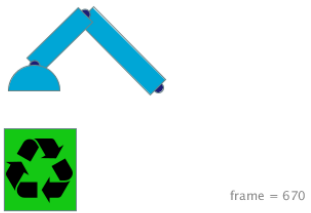
step 8:
rotate hand (-90°)
translate trash (-265,0)



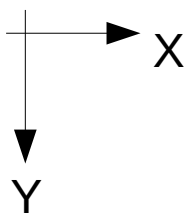
Robot story step by step



step 9:
hand aperture (180°)
trash = visible

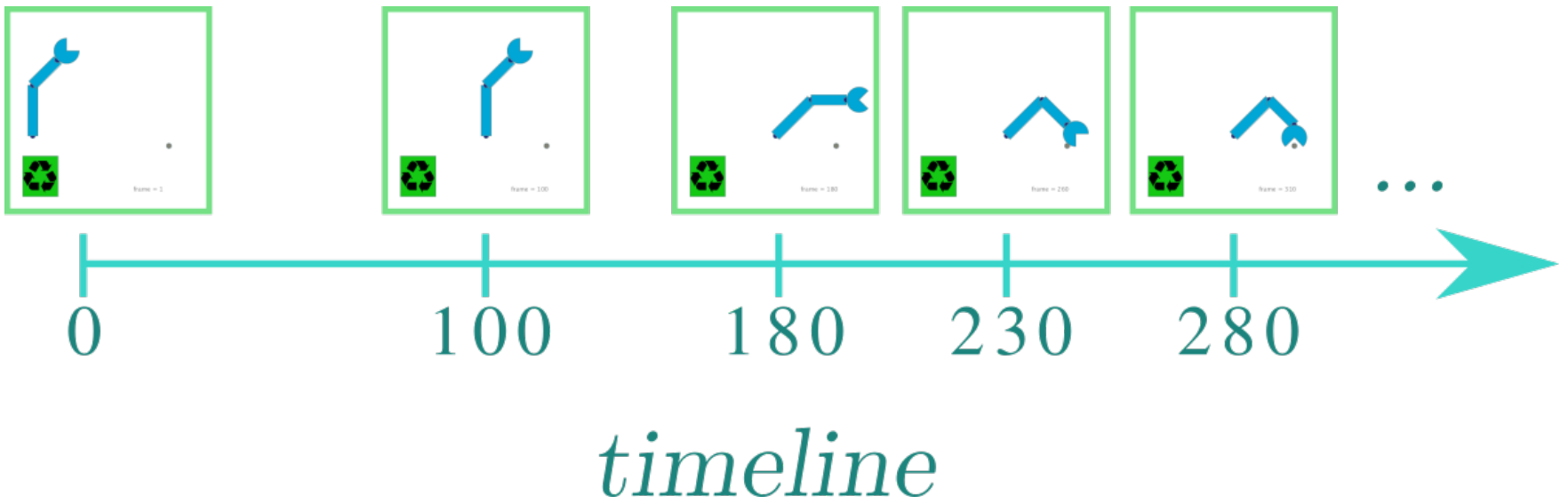


step 10:
translate trash (0, 40)



keyframing

- describe important moments in the animation



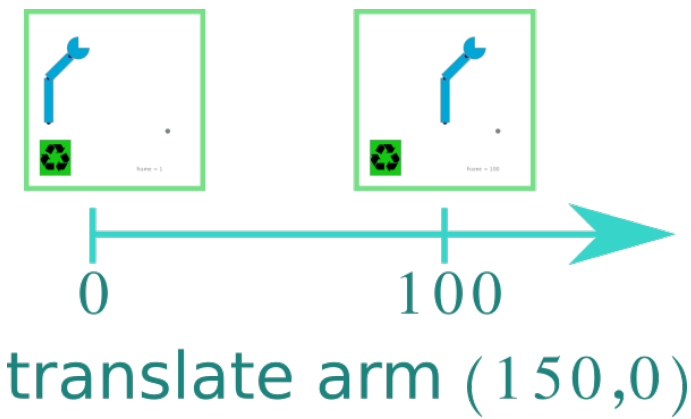
inbetweening

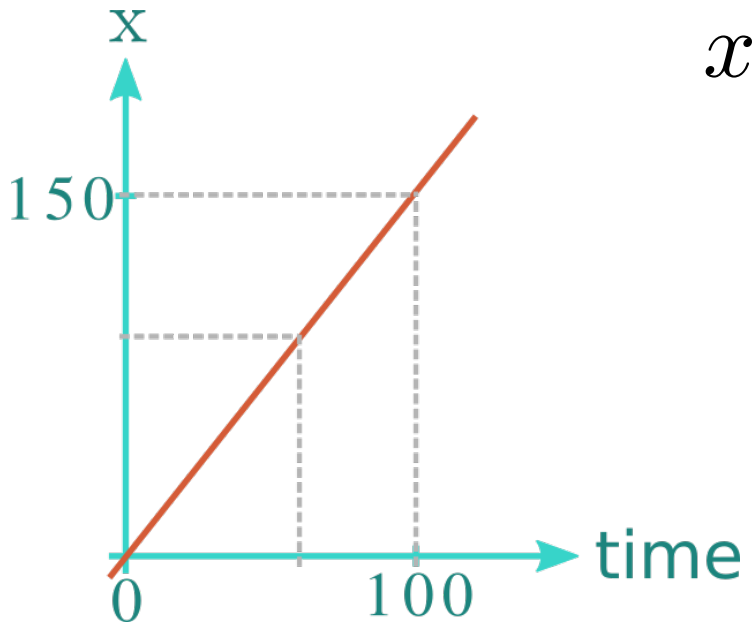
- how to move from frame to frame?
  - linear interpolation

$$\delta x = 150/100$$

$$x_{i+1} = x_i + \delta x$$

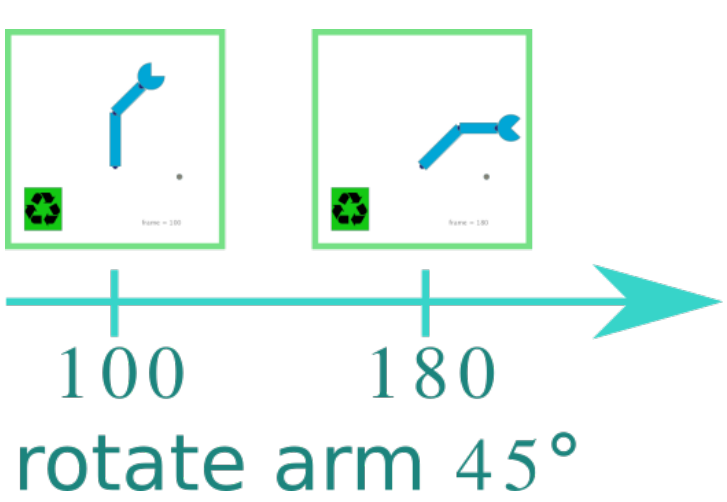
$$T = \begin{bmatrix} 1 & 0 & \delta x \\ 0 & 1 & \delta y \\ 0 & 0 & 1 \end{bmatrix}$$

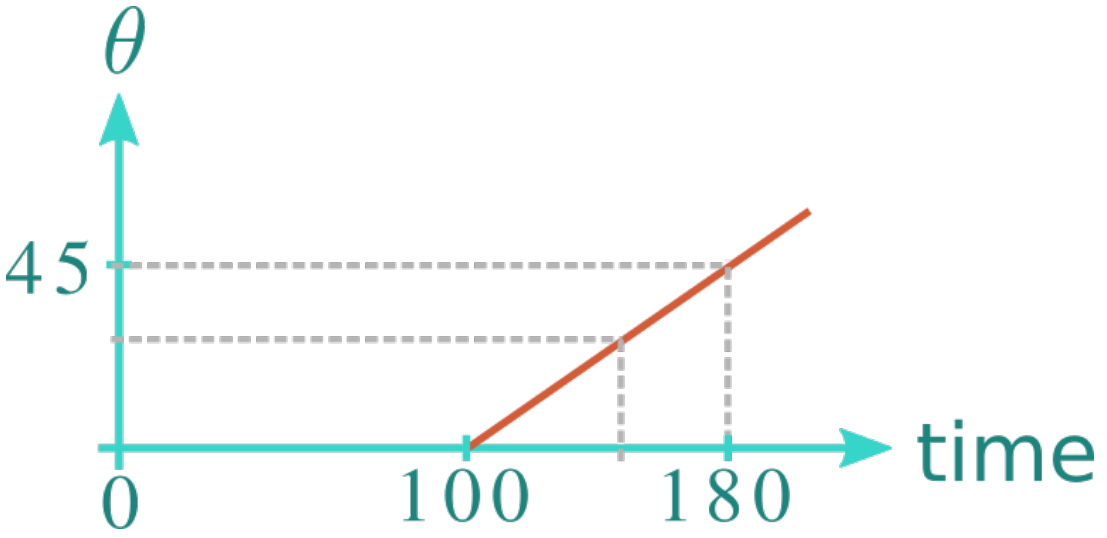




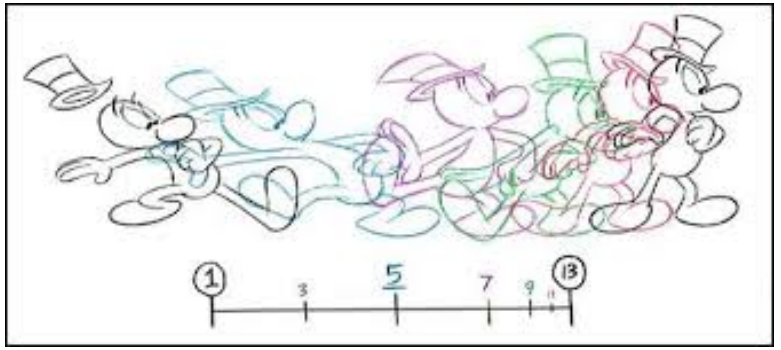
inbetweening

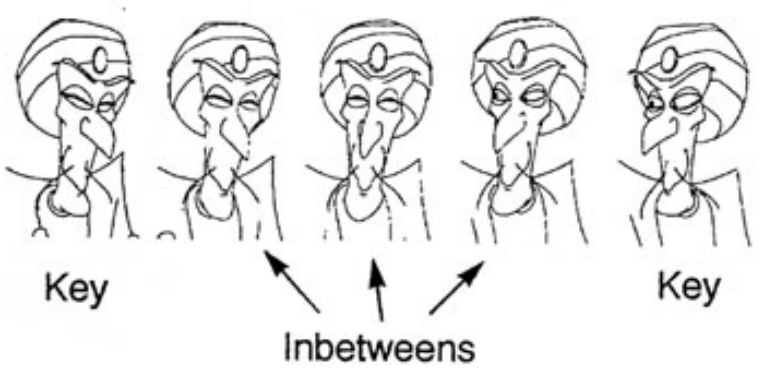
- how to move from frame to frame?
  - linear interpolation

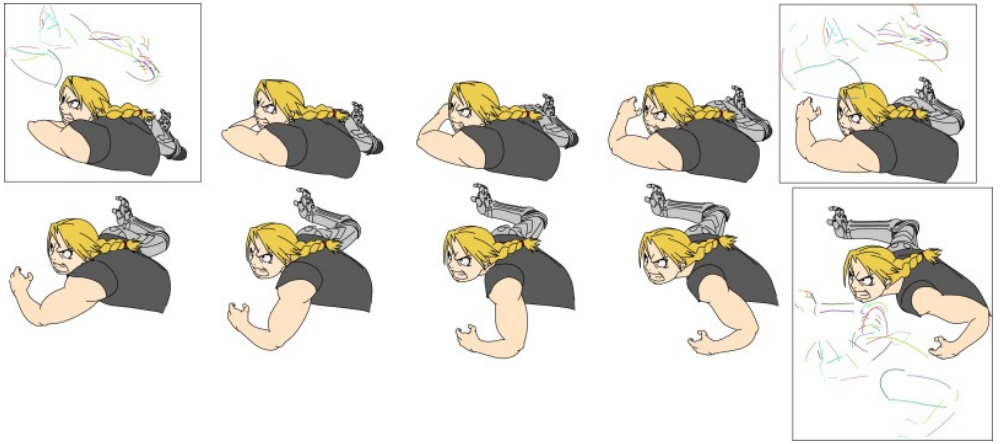




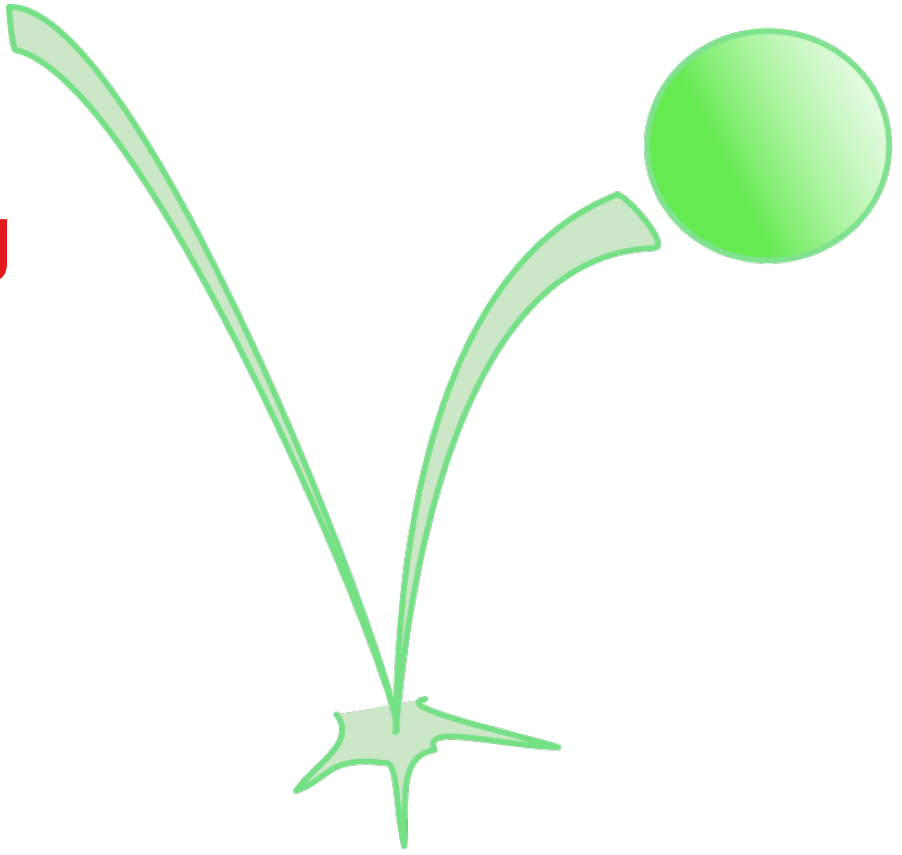
traditional inbetweening



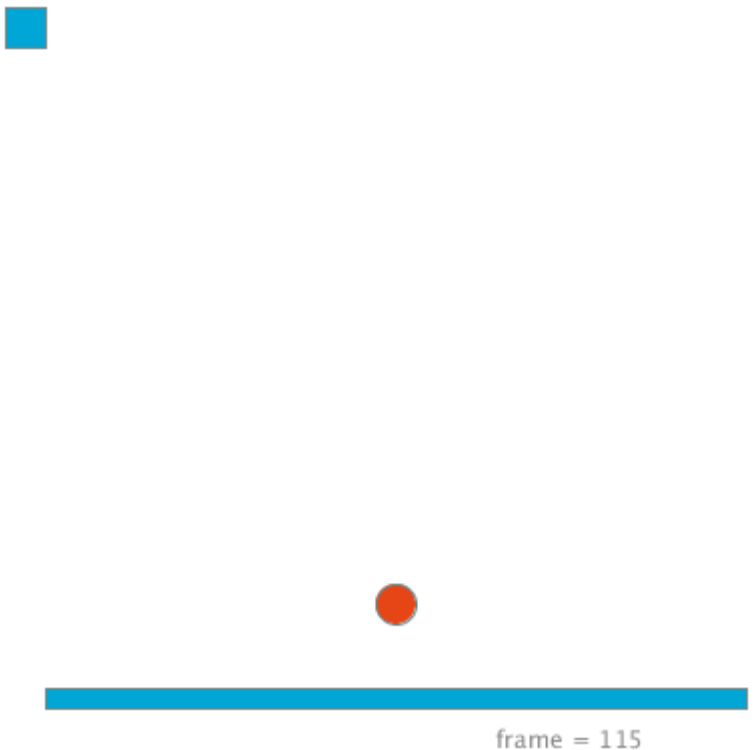




- animation is about timing
  - from X to Y in t secs …
  - expressiveness

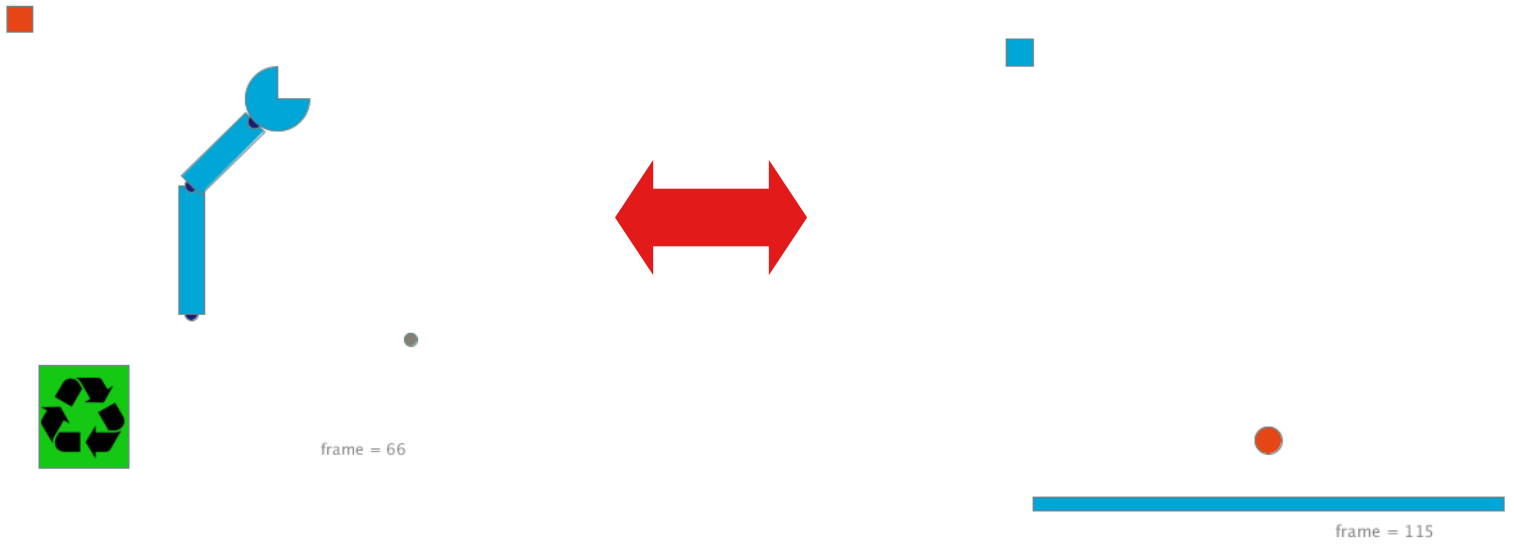


simple bouncing ball demo

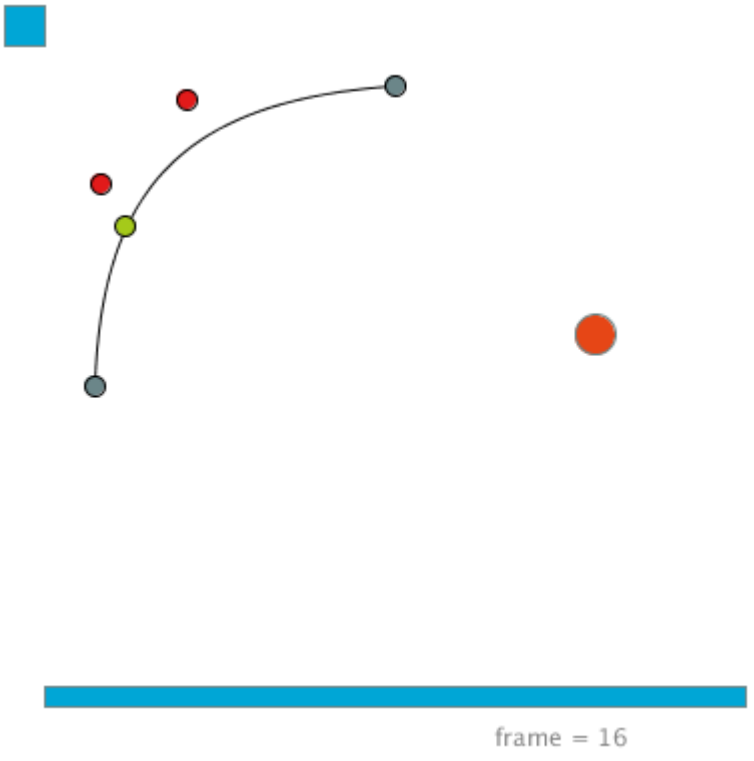


simple bouncing ball demo

- not very realistic
  - constant velocity does not work here

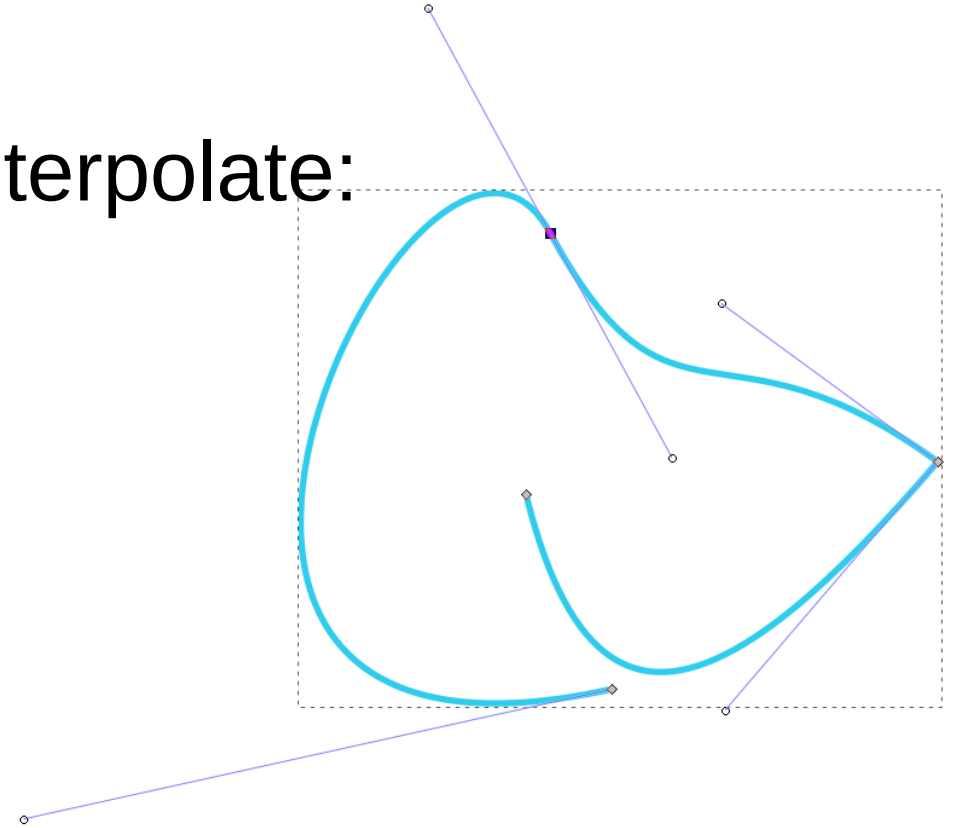


non-linear interpolation



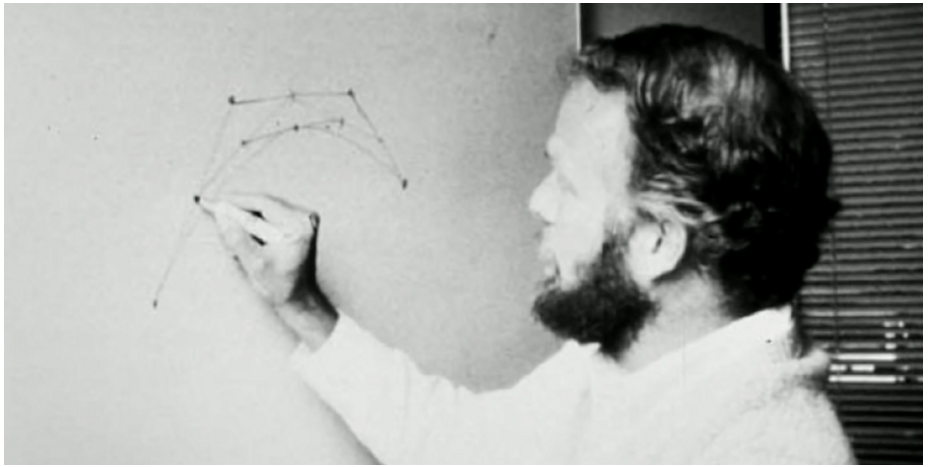
splines

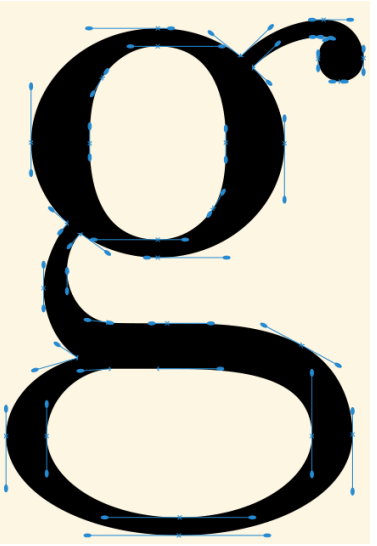
- parametric curves
- use the parameter to interpolate:
  - position
  - orientation
  - velocity
  - anything you want ...



Bézier curve

Pierre Bézier





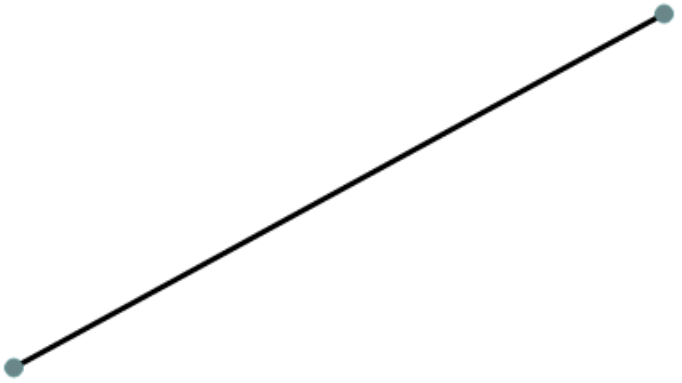
TrueType: quadratic Bézier
PostScript: cubic Bézier
OpenType: both

Paul de Casteljau

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} \text{(1-t)}^{n-i} t^i \mathbf{P}_i$$

Bézier curve

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} \text{(1-t)}^{n-i} t^i \mathbf{P}_i$$



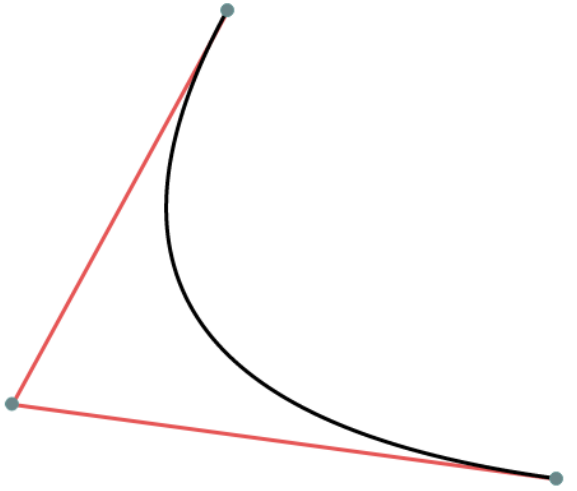
linear: 2 control points, n=1

$$B(t) = \binom{1}{0} \text{(1-t)}^{1} t^0 \mathbf{P}_0 + \binom{1}{1} \text{(1-t)}^{0} t^1 \mathbf{P}_1$$

$$B(t) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1$$

Bézier curve

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} \text{(1-t)}^{n-i} t^i \mathbf{P}_i$$



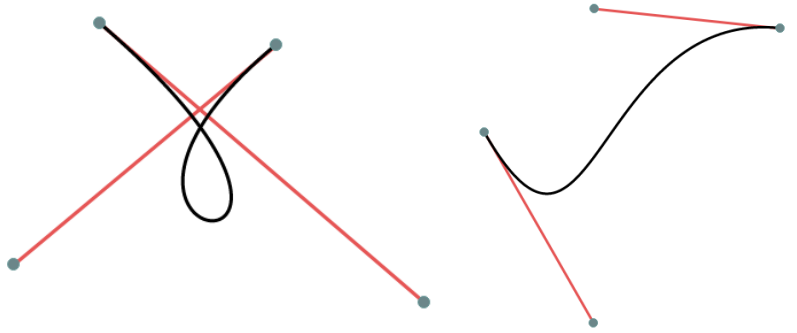
quadratic: 3 control points, n=2

$$B(t) = \binom{2}{0} \text{(1-t)}^{2} t^0 \mathbf{P}_0 + \binom{2}{1} \text{(1-t)}^{1} t^1 \mathbf{P}_1 + \binom{2}{2} \text{(1-t)}^{0} t^2 \mathbf{P}_2$$

$$B(t) = (1-t)^2 \mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2 \mathbf{P}_2$$

Bézier curve

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} \text{(1-t)}^{n-i} t^i \mathbf{P}_i$$

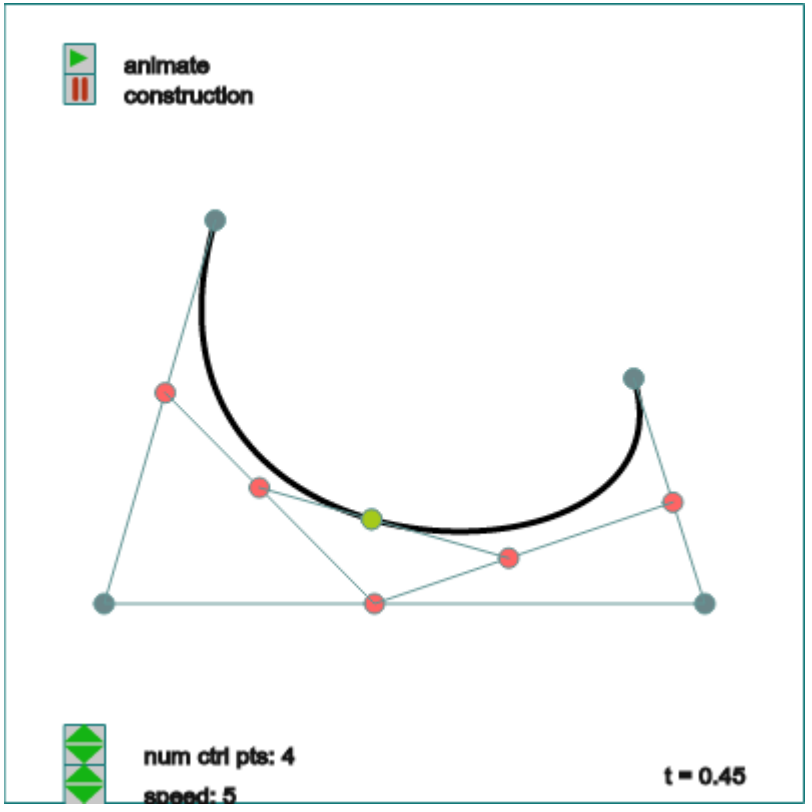


cubic: 4 control points, n=3

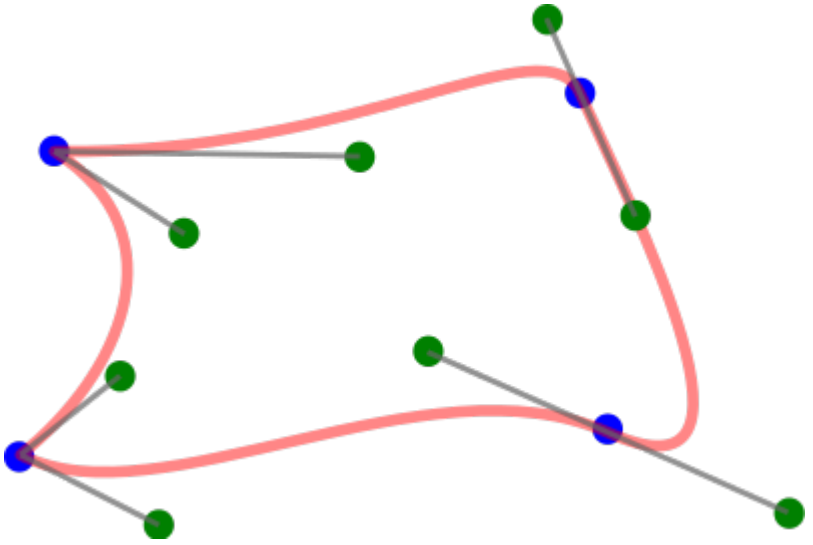
$$B(t) = \binom{3}{0} \text{(1-t)}^{3} t^0 \mathbf{P}_0 + \binom{3}{1} \text{(1-t)}^{2} t^1 \mathbf{P}_1 + \binom{3}{2} \text{(1-t)}^{1} t^2 \mathbf{P}_2 + \binom{3}{3} \text{(1-t)}^{0} t^3 \mathbf{P}_3$$

$$B(t) = (1-t)^3 \mathbf{P}_0 + 3(1-t)^2 t^1 \mathbf{P}_1 + 3(1-t)^1 t^2 \mathbf{P}_2 + t^3 \mathbf{P}_3$$

demo

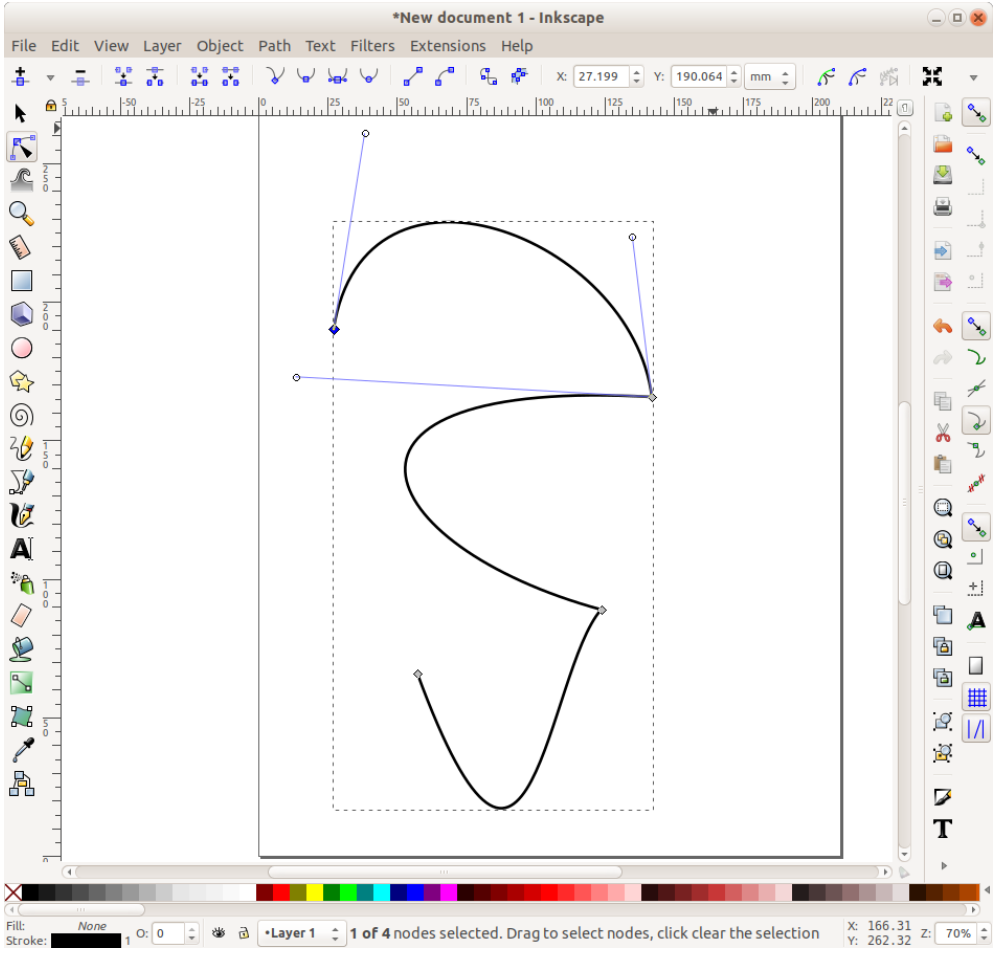


composite Bézier curve



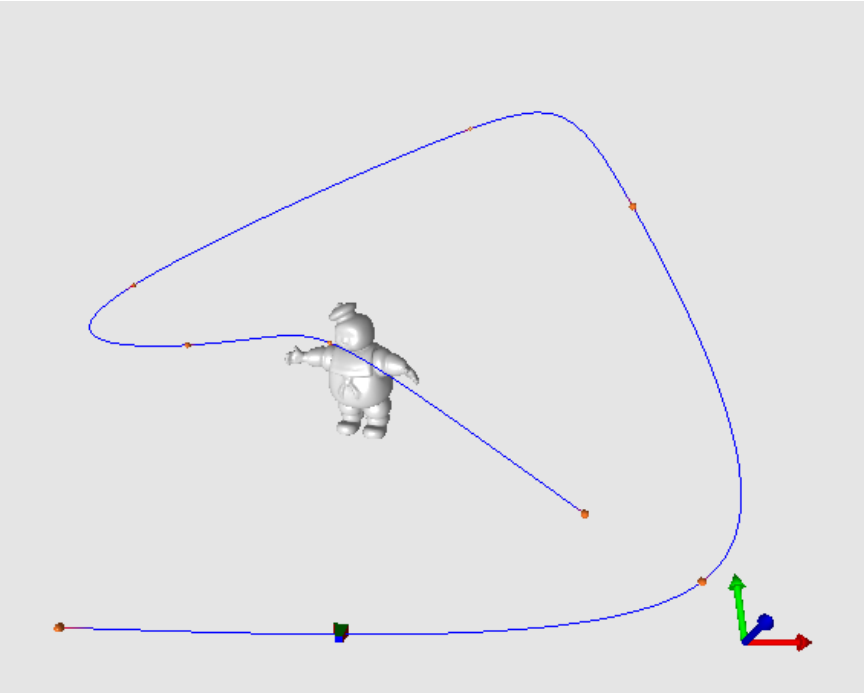
composite Bézier curve

- inkscape demo



https://inkscape.org/

camera path demo



- interpolates:
  - position
  - orientation

orientation interpolation

- you have seen rotation in 3D
  - ex. Rotation about x-axis

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

orientation interpolation

linear interpolation between two points (**LERP**)

$$\mathbf{P}(t) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1$$

linear interpolation between two rotation matrices?

$$\mathbf{R}(t) = (1-t)\mathbf{R}_0 + t\mathbf{R}_1$$

orientation interpolation

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \qquad R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_0 = R_x(\pi)$$

$$R_1 = R_z(\pi)$$

$$\mathbf{R}(t) = (1 - t)\mathbf{R}_0 + t\mathbf{R}_1$$

$$t = 0.5$$

$$R(0.5) = 0.5 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} + 0.5 \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.5 & -0.5 & 0 \\ 0.5 & 0 & -0.5 \\ 0 & 0.5 & 0.5 \end{bmatrix}$$

orientation interpolation

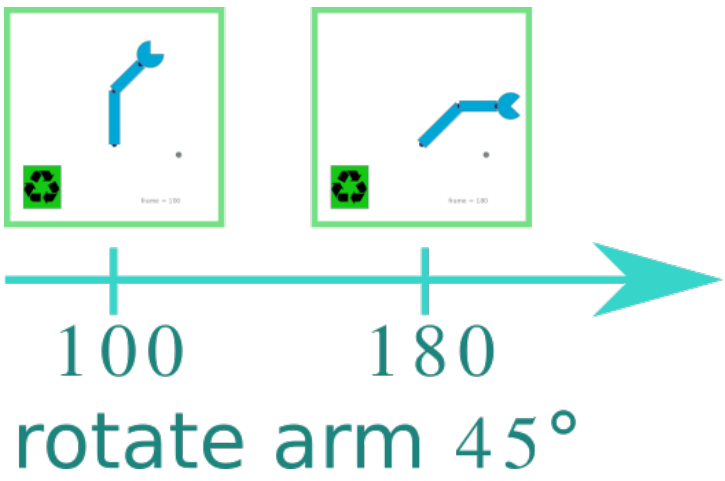
linear interpolation between two points (**LERP**)

$$\mathbf{P}(t) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1$$

linear interpolation between angles?

$$\alpha(t) = (1-t)\alpha_0 + t\alpha_1$$

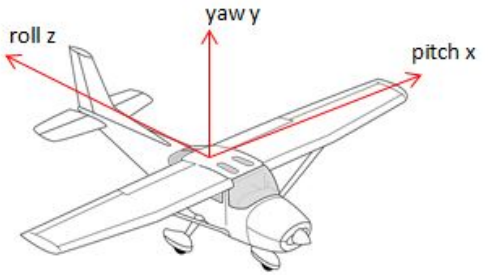
we saw this in 2D



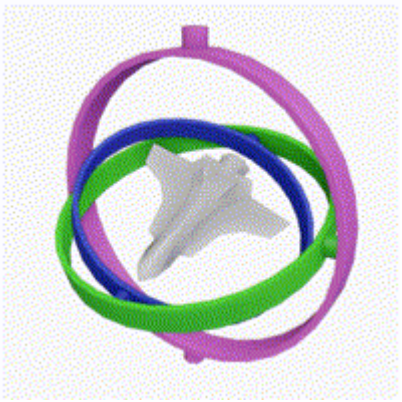
orientation interpolation

Euler angles:

$$R(\alpha, \beta, \gamma) = R_Z(\alpha) R_Y(\beta) R_X(\gamma)$$



$$\alpha(t) = (1-t)\alpha_0 + t\alpha_1$$
$$\beta(t) = (1-t)\beta_0 + t\beta_1$$
$$\gamma(t) = (1-t)\gamma_0 + t\gamma_1$$



$$R_Z R_Y R_X \neq R_X R_Y R_Z$$

ambiguity

gimbal lock!

quaternions

William Hamilton

four-dimensional complex numbers

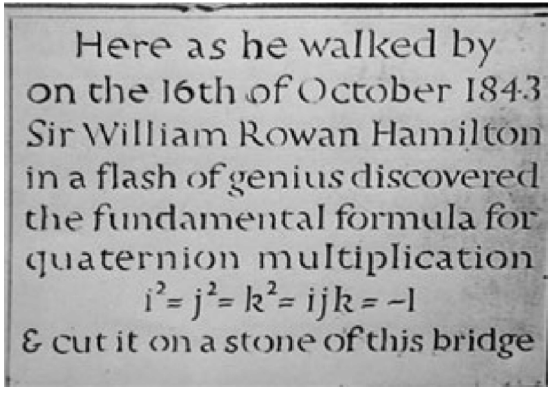
$$q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$$

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$$

scalar and vector parts

$$q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} = (s, \mathbf{v})$$

all operations are defined: addition, subtraction, multiplication ...



unit quaternions

- unit quaternion q:
  - represents rotation θ around axis **a**

$$\mathbf{q} = [q_0 \; q_1 \; q_2 \; q_3] = \langle s, \mathbf{v} \rangle$$

$$\mathbf{q} = \left\langle cos\frac{\theta}{2}, \mathbf{a} \sin\frac{\theta}{2} \right\rangle$$

  - many nice properties
    - e.g. concatenate quaternions

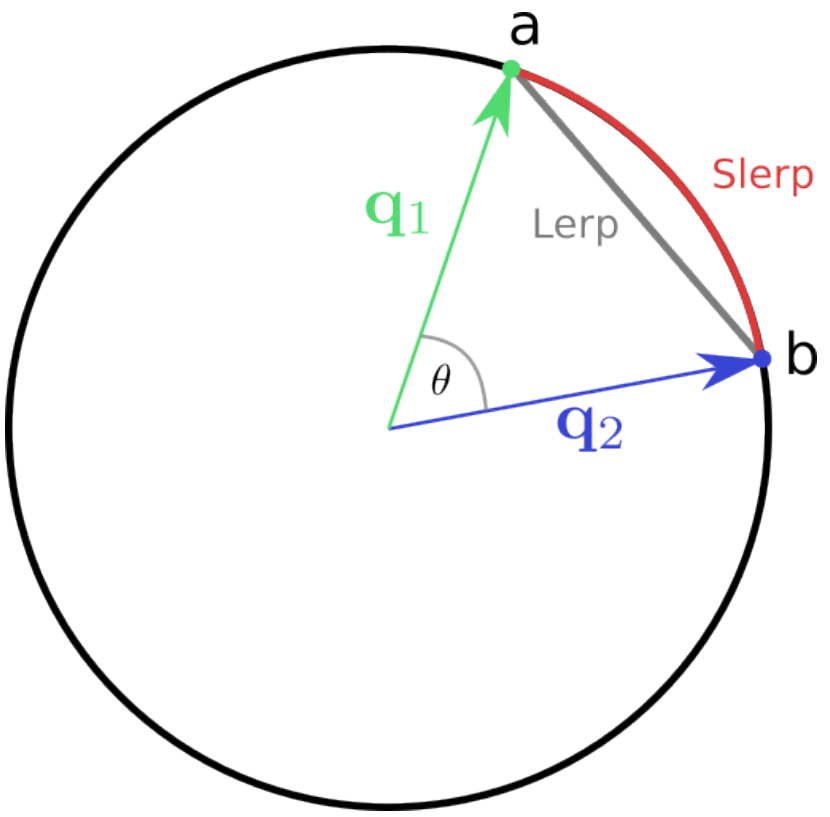
$$\mathbf{q} = \mathbf{q}_2\mathbf{q}_1$$

interpolating quaternions

- Lerp vs Slerp

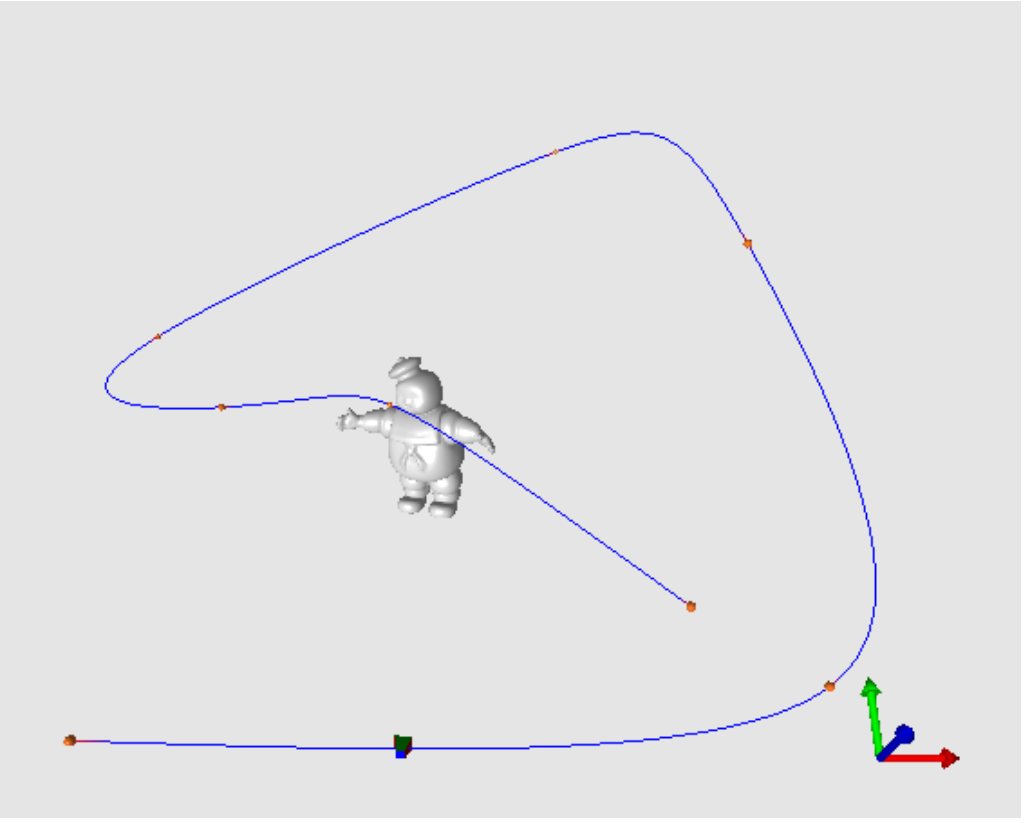
$$\mathrm{Lerp}(t, \mathbf{a}, \mathbf{b}) = (1-t)\mathbf{a} + t\mathbf{b}$$

$$\mathrm{Slerp}(t, \mathbf{q}_1, \mathbf{q}_2) = \mathbf{q}_1(\mathbf{q}_1^* \mathbf{q}_2)^t$$



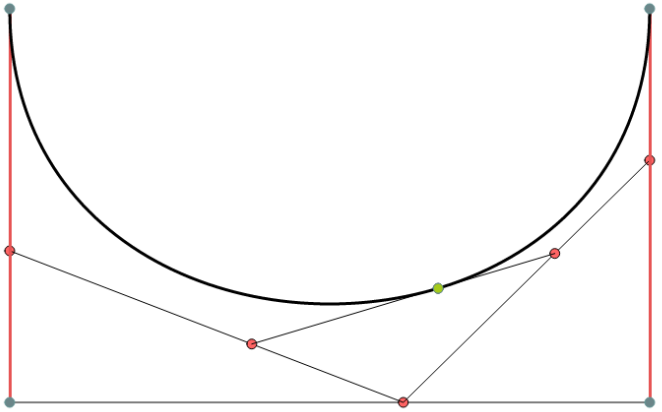
interpolating quaternions

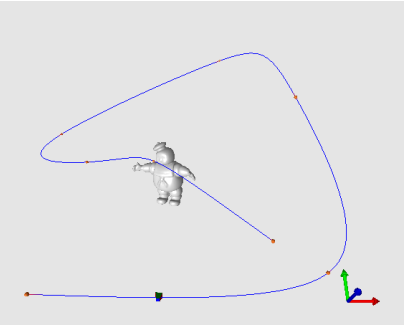
- linear interpolation on a curve (path)
  - not smooth at control points (“hiccups”)



interpolating quaternions

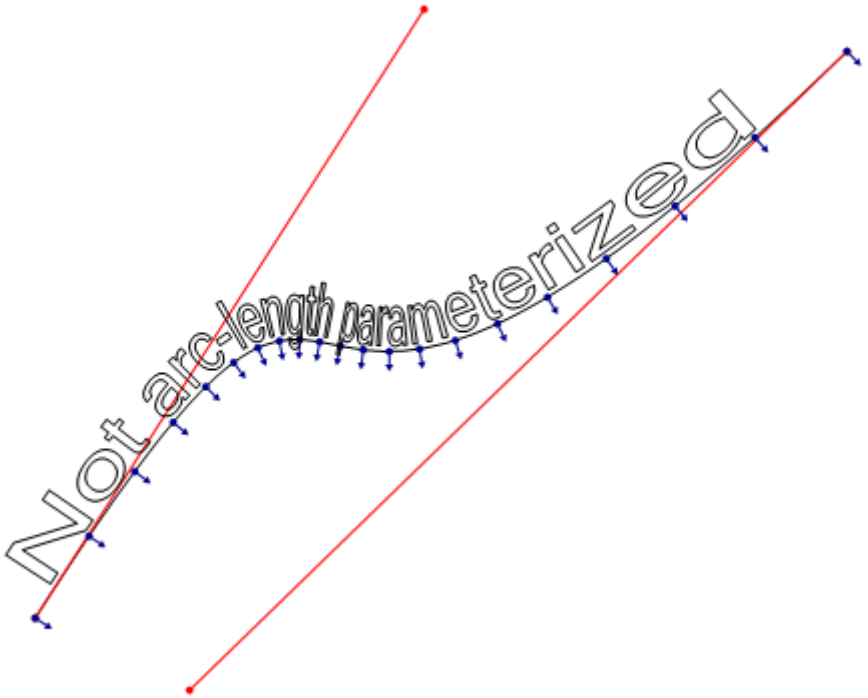
- one option: Squad
  - “Quaternions, Interpolation and Animation, Dan et al.”, http://web.mit.edu/2.998/www/QuaternionReport1.pdf
  - analogous to Bézier curve, but with Slerp instead of Lerp





uses Bézier for position
and Squad for orientation

speed along curve

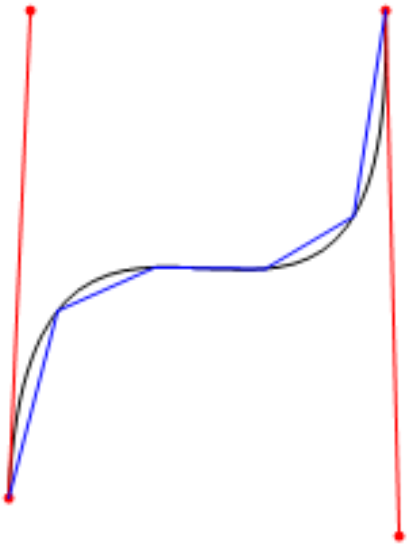


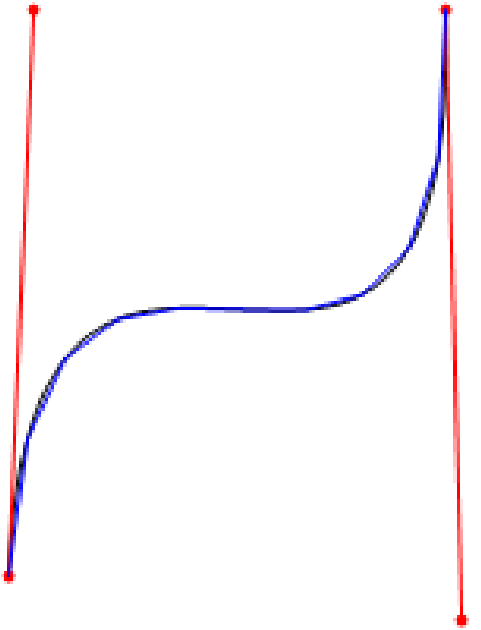
- curve parametrization
  - speed not constant
- t does not vary linearly in regards to arc-length

http://www.planetclegg.com/projects/WarpingTextToSplines.html

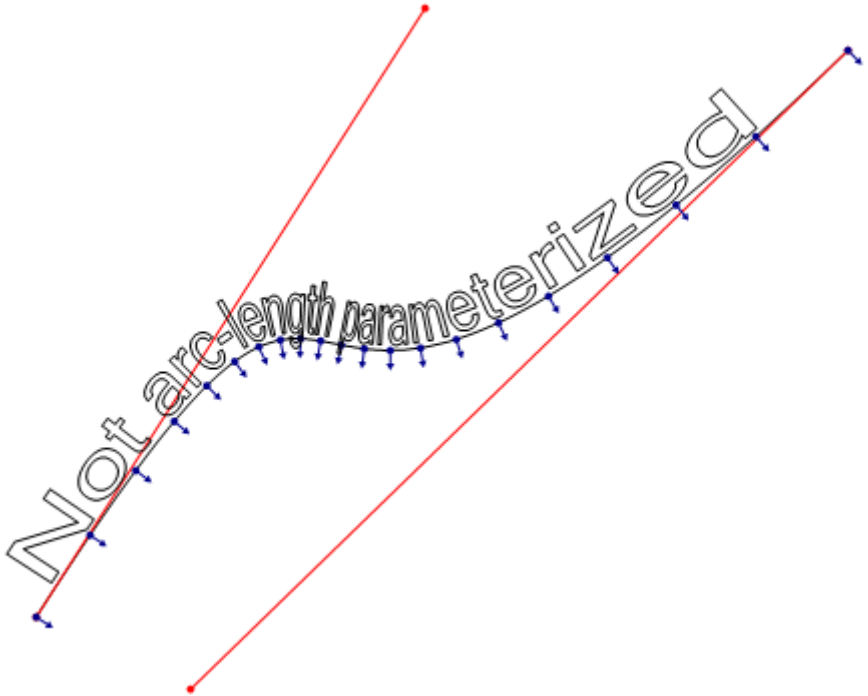
re-parameterization

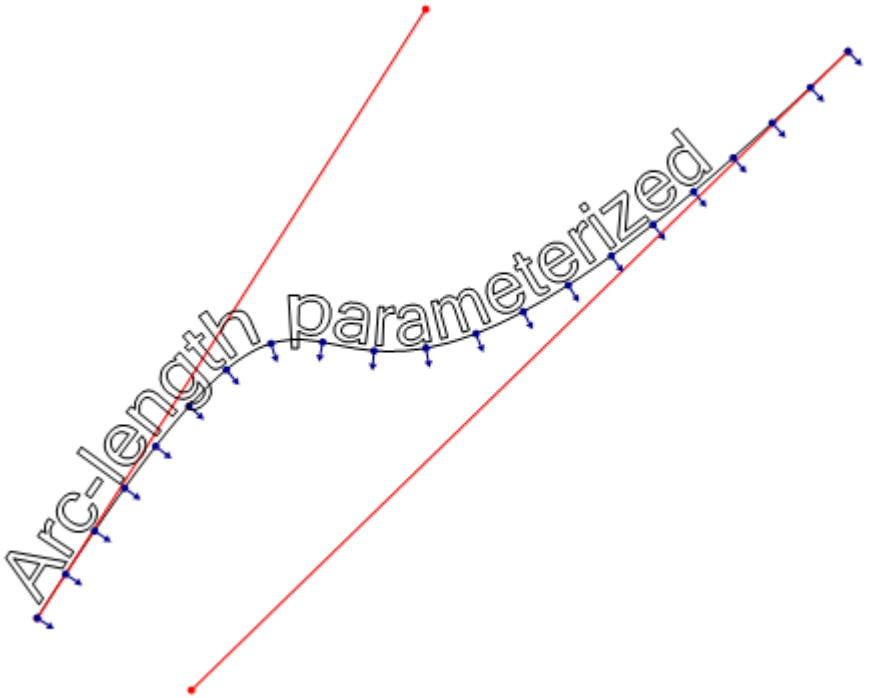
- divide the curve into equal ***t*** segments
- approximate each arc-segment by a line
- make a look-up table for arc-length
  - convert ***t*** to **arc-length**





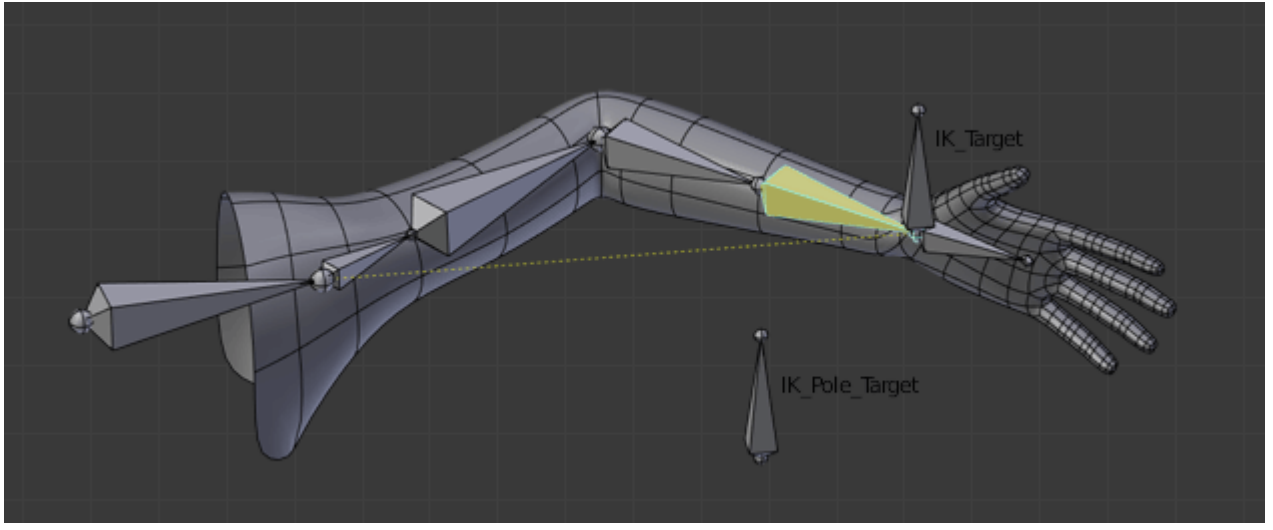
re-parameterization





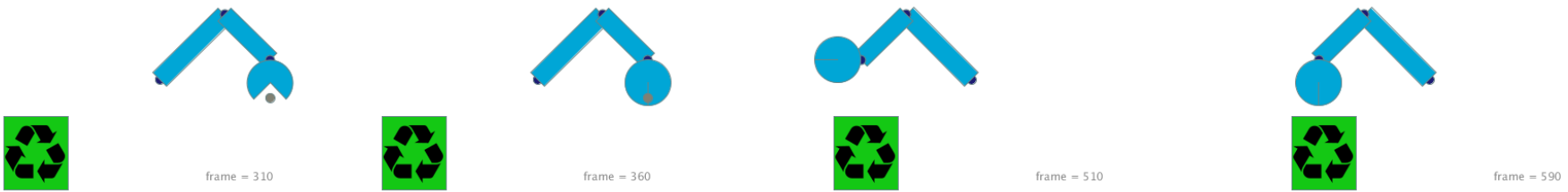
complex objects

- getting the hand at correct position is hard



forward kinematics

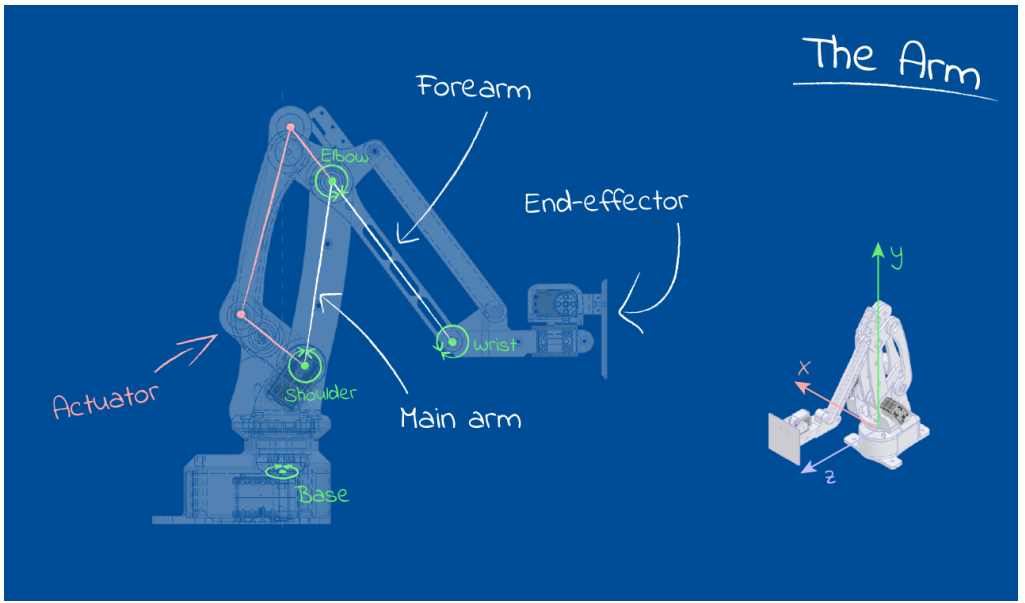
- operate on each joint individually



inverse kinematics

- animation and robotics
  - accuracy at end point is usually more important

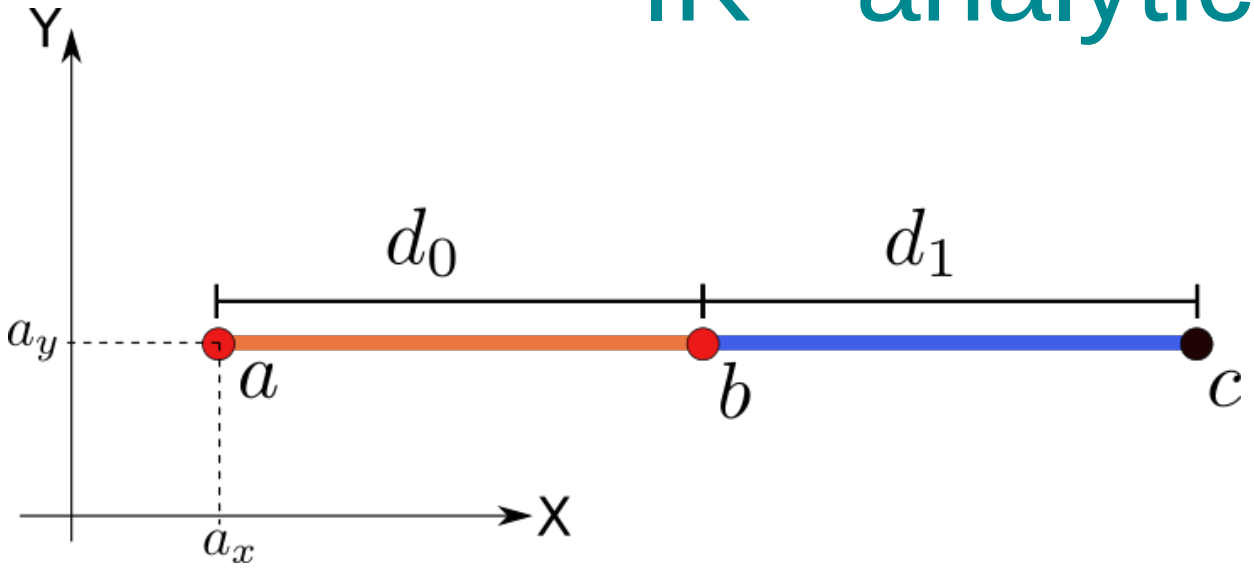
https://docs.toonboom.com/help/animate/Content/HAR/Stage/017_Cut-out_Animation/021_H1_Animating_using_Inverse_Kinematics.html



https://robohub.org/robotics-maths-python-a-fledgling-computer-scientists-guide-to-inverse-kinematics/

IK - analytical

IK - analytical



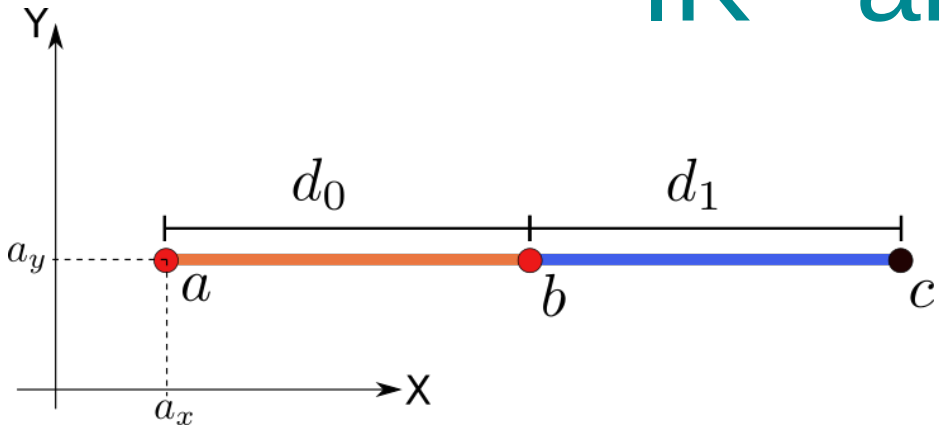
- transformation matrices for a, b and c?

$$M_a = \begin{bmatrix} 1 & 0 & a_x \\ 0 & 1 & a_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_b = M_a \begin{bmatrix} 1 & 0 & d_0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$Mc = M_b \begin{bmatrix} 1 & 0 & d_1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

IK - analytical



$$M_a = \begin{bmatrix} 1 & 0 & a_x \\ 0 & 1 & a_y \\ 0 & 0 & 1 \end{bmatrix}$$

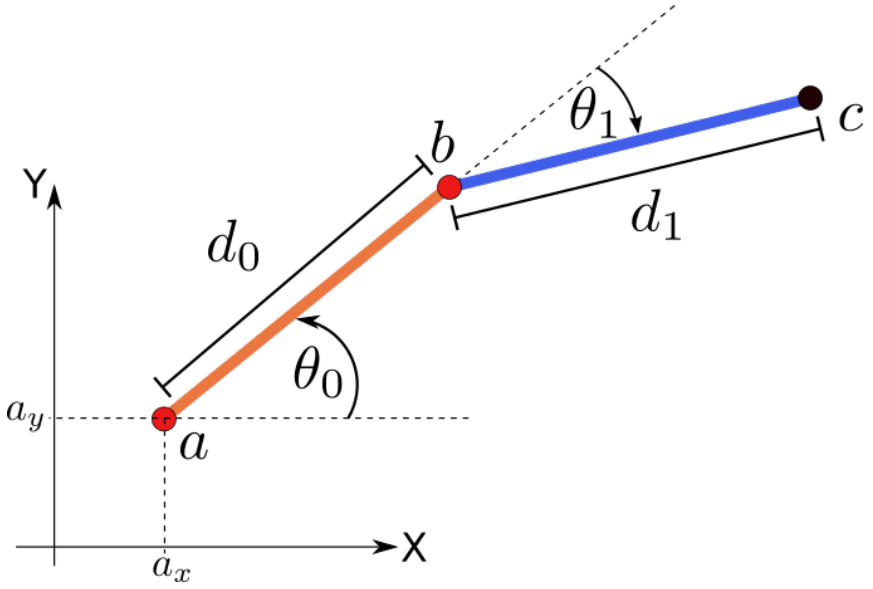
$$M_b = M_a \begin{bmatrix} 1 & 0 & d_0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$Mc = M_b \begin{bmatrix} 1 & 0 & d_1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- how do we get a, b and c?

$$a = \begin{bmatrix} 1 & 0 & a_x \\ 0 & 1 & a_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad a = M_a \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad b = M_b \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad c = M_c \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

IK - analytical

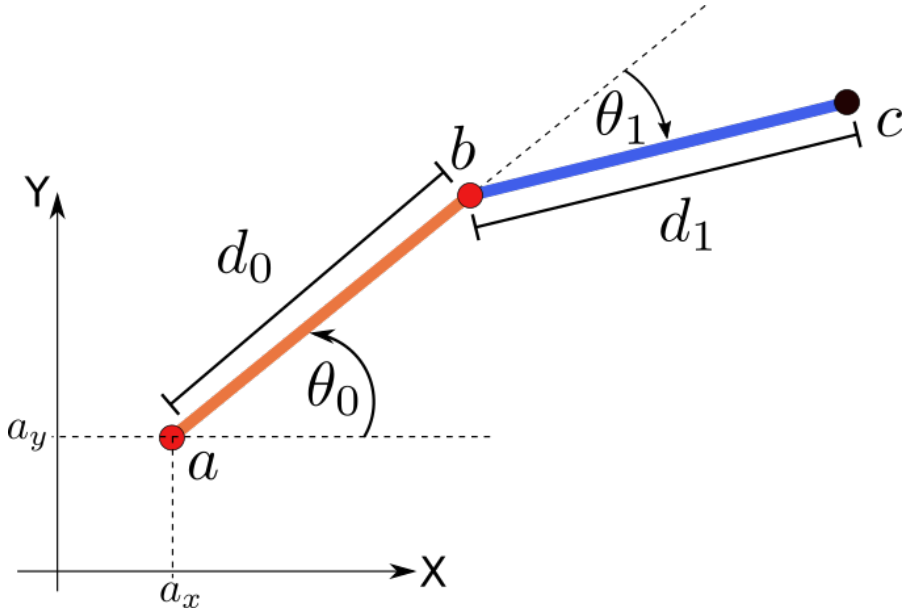


$$M_a = \begin{bmatrix} 1 & 0 & a_x \\ 0 & 1 & a_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_b = M_a \begin{bmatrix} \cos\theta_0 & -\sin\theta_0 & 0 \\ \sin\theta_0 & \cos\theta_0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & d_0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \mathrm{M}_a \begin{bmatrix} \cos\theta_0 & -\sin\theta_0 & d_0\cos\theta_0 \\ \sin\theta_0 & \cos\theta_0 & d_0\sin\theta_0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_c = M_b \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & d_1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \mathrm{M}_b \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & d_1\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & d_1\sin\theta_1 \\ 0 & 0 & 1 \end{bmatrix}$$

IK - analytical



$$M_a = \begin{bmatrix} 1 & 0 & a_x \\ 0 & 1 & a_y \\ 0 & 0 & 1 \end{bmatrix}$$

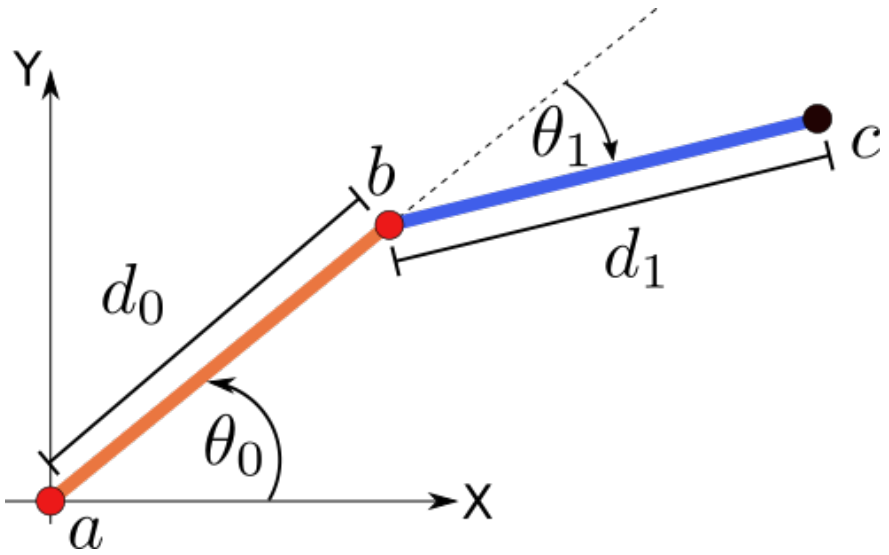
$$M_b = M_a \begin{bmatrix} \cos\theta_0 & -\sin\theta_0 & d_0\cos\theta_0 \\ \sin\theta_0 & \cos\theta_0 & d_0\sin\theta_0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_c = M_b \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & d_1\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & d_1\sin\theta_1 \\ 0 & 0 & 1 \end{bmatrix}$$

- given c, how do we get θ_0 and θ_1?
  - consider a, d_0 and d_1 fixed

- do we need to worry about b?
- do we need to worry about M_a?

IK - analytical



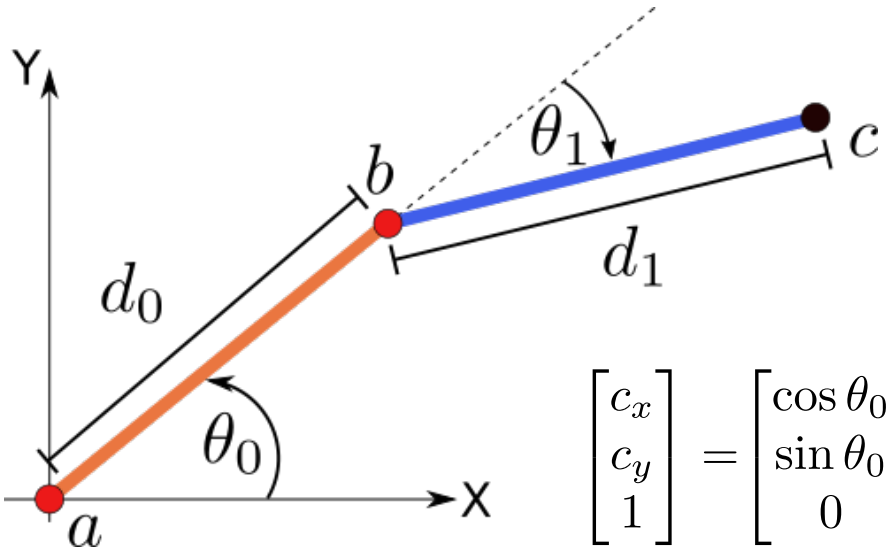
$$M_a = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_b = M_a \begin{bmatrix} \cos\theta_0 & -\sin\theta_0 & d_0\cos\theta_0 \\ \sin\theta_0 & \cos\theta_0 & d_0\sin\theta_0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_c = M_b \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & d_1\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & d_1\sin\theta_1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} c_x \\ c_y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_0 & -\sin\theta_0 & d_0\cos\theta_0 \\ \sin\theta_0 & \cos\theta_0 & d_0\sin\theta_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & d_1\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & d_1\sin\theta_1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

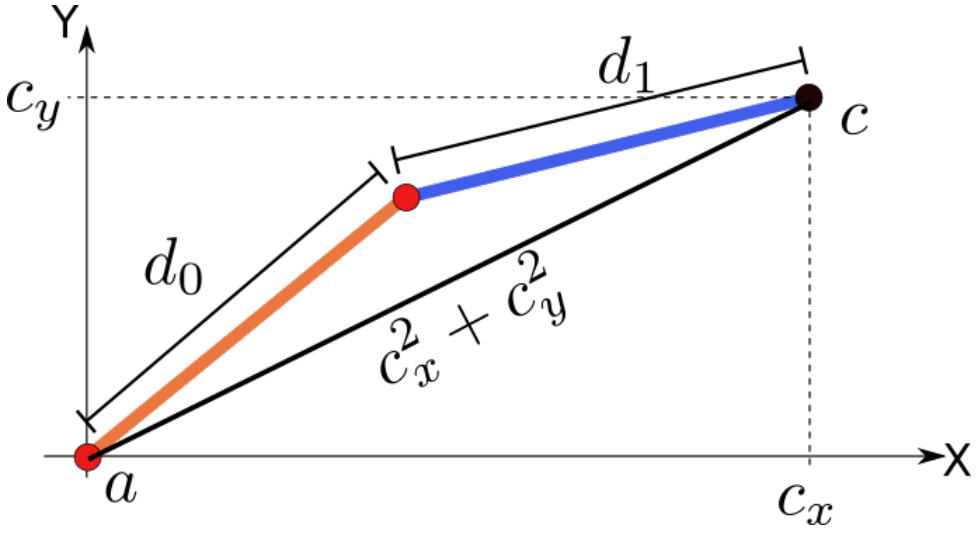
IK - analytical



$$\begin{bmatrix} c_x \\ c_y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_0 & -\sin\theta_0 & d_0\cos\theta_0 \\ \sin\theta_0 & \cos\theta_0 & d_0\sin\theta_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & d_1\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & d_1\sin\theta_1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} c_x \\ c_y \\ 1 \end{bmatrix} = \begin{bmatrix} d_0\cos\theta_0 + d_1\cos(\theta_0+\theta_1) \\ d_0\sin\theta_0 + d_1\sin(\theta_0+\theta_1) \\ 1 \end{bmatrix}$$

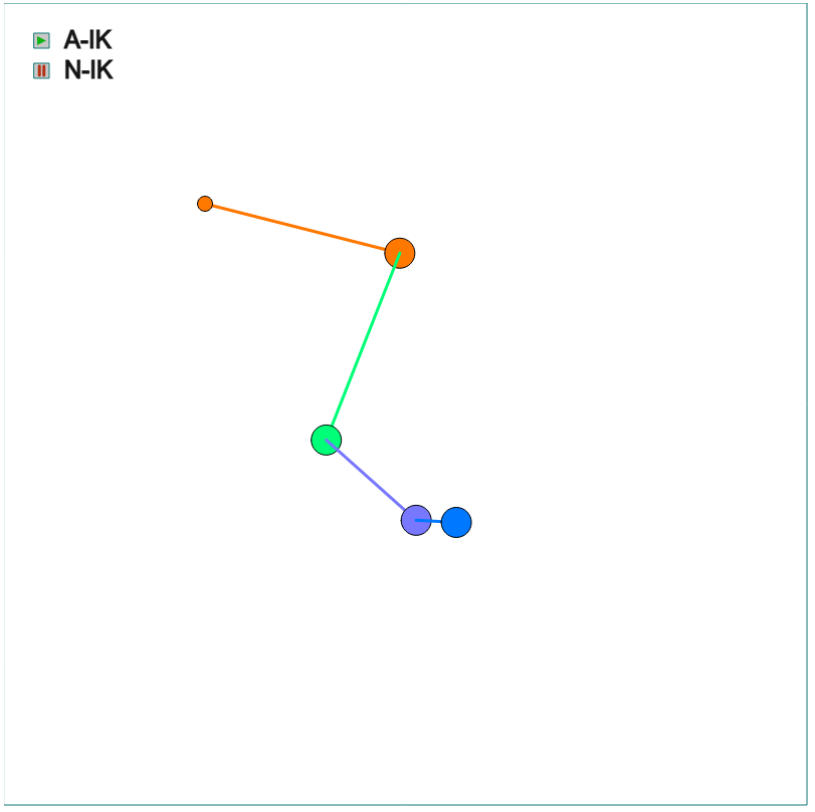
IK - analytical



$$\begin{bmatrix} c_x \\ c_y \\ 1 \end{bmatrix} = \begin{bmatrix} d_0 \cos\theta_0 + d_1 \cos(\theta_0 + \theta_1) \\ d_0 \sin\theta_0 + d_1 \sin(\theta_0 + \theta_1) \\ 1 \end{bmatrix}$$

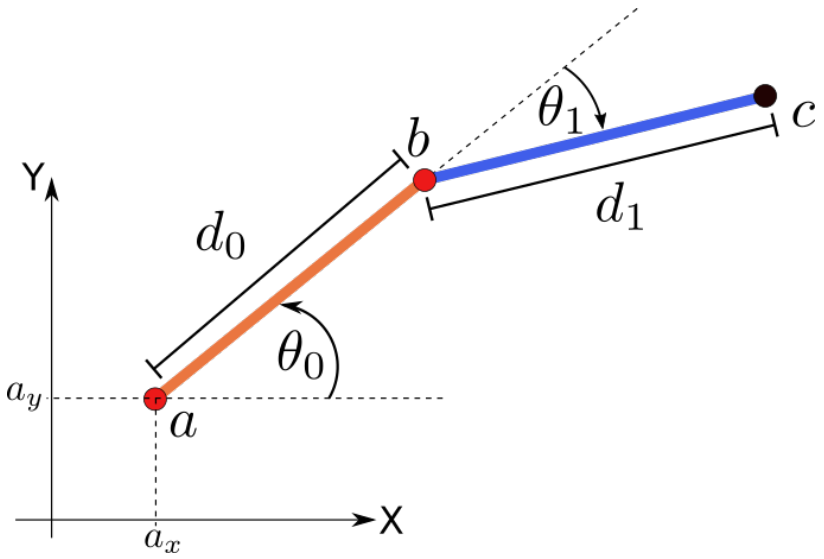
$$\theta_1 = \cos^{-1}\left(\frac{c_x^2 + c_y^2 - d_0^2 - d_1^2}{2 d_0 d_1}\right) \qquad \theta_0 = \tan^{-}1\left(\frac{y}{x}\right) - \tan^{-}1\left(\frac{d_1 \sin\theta_1}{d_0 + d_1 \cos\theta_1}\right)$$

IK - analytical



IK - analytical

- for θ_1 on other side, different (but similar) solution
- for more than 2 angles, analytical solution gets very hard

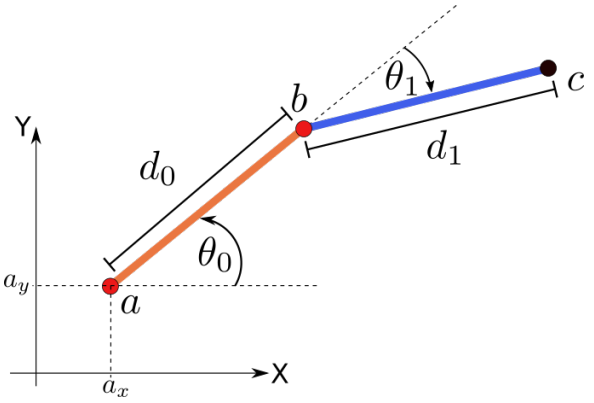


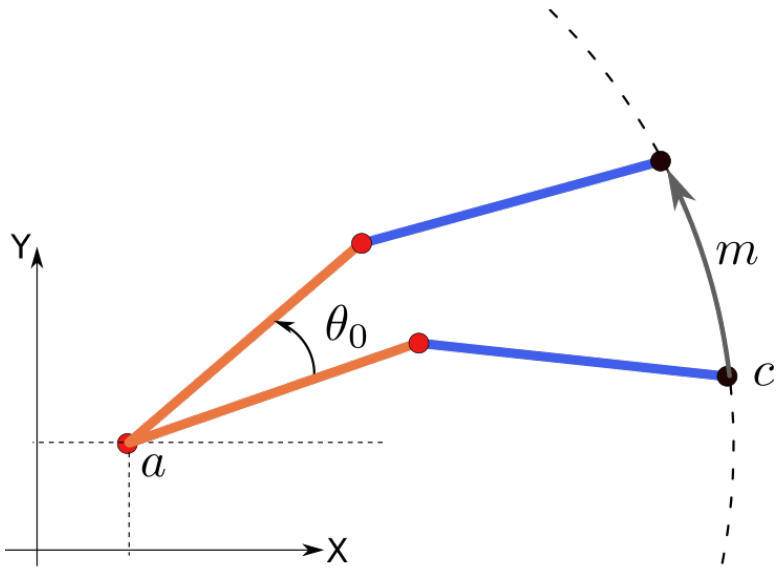
$$\theta_1 = \cos^{-1}\left(\frac{c_x^2 + c_y^2 - d_0^2 - d_1^2}{2 d_0 d_1}\right)$$

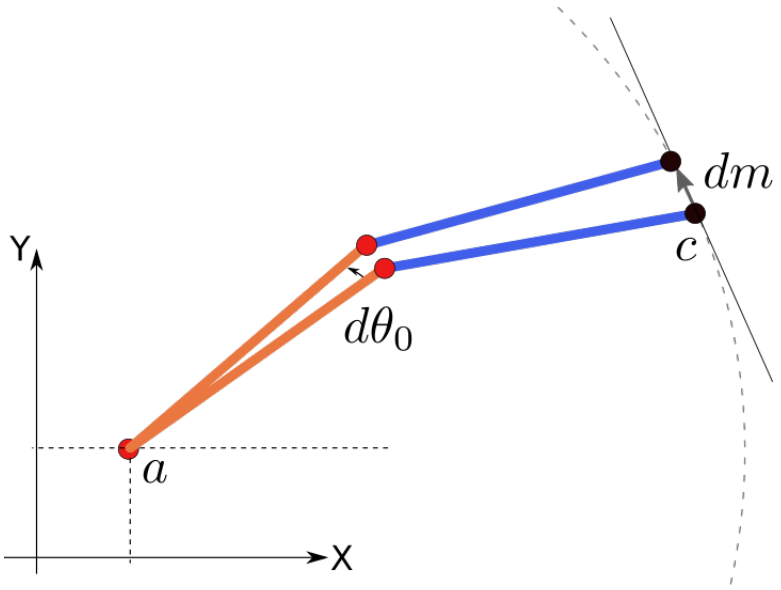
$$\theta_0 = \tan^{-}1\left(\frac{y}{x}\right) - \tan^{-}1\left(\frac{d_1 \sin\theta_1}{d_0 + d_1 \cos\theta_1}\right)$$

IK - numerical

- turn it into a linear system problem: Ax=b
- solution vector x: change of angles θ's
- small changes in c <=> small changes in θ's



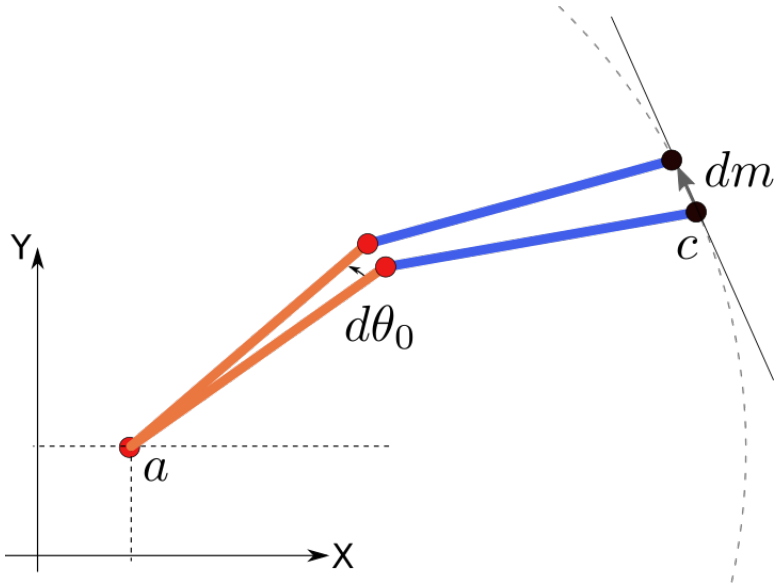




IK - numerical

- partial derivatives: how small changes in θ affects c
- Jacobian matrix with all joint angles

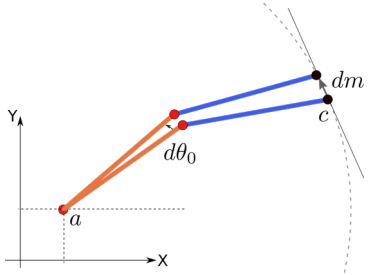
$$\frac{\delta c_x}{\delta \theta_0}, \frac{\delta c_y}{\delta \theta_0}$$

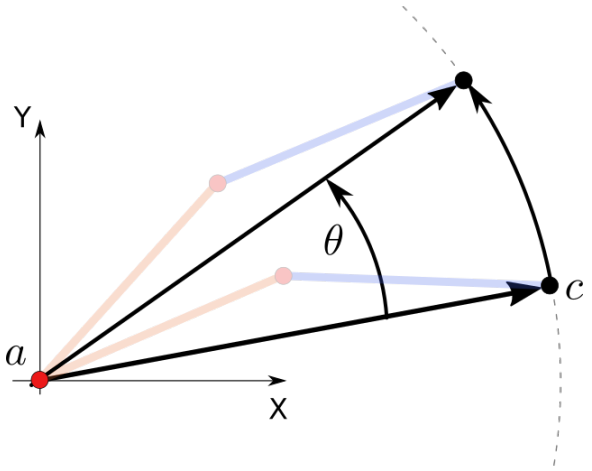


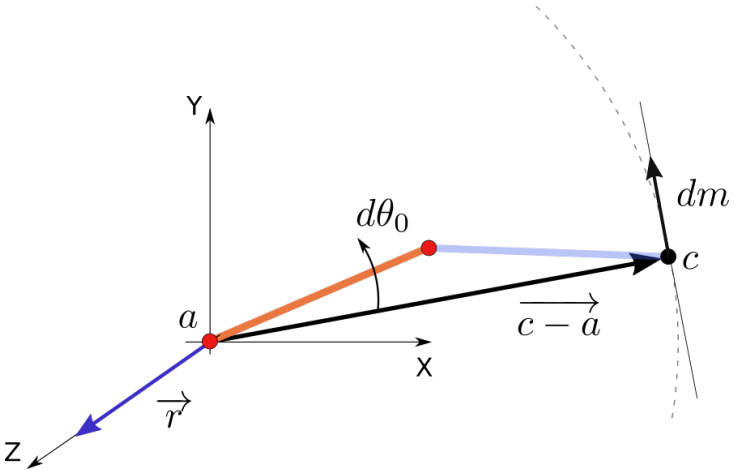
$$J = \begin{bmatrix} \frac{\delta c_x}{\delta \theta_0} & \frac{\delta c_x}{\delta \theta_1} & \frac{\delta c_x}{\delta \theta_2} & \cdots \\ \frac{\delta c_y}{\delta \theta_0} & \frac{\delta c_y}{\delta \theta_1} & \frac{\delta c_y}{\delta \theta_2} & \cdots \end{bmatrix}$$

IK - numerical

- partial derivatives?







$$\frac{\delta c_x}{\delta \theta_0}, \frac{\delta c_y}{\delta \theta_0}$$

$$dm = \left(\frac{\delta c_x}{\delta \theta_0}, \frac{\delta c_y}{\delta \theta_0}, 0 \right) = \vec{r} \times \overrightarrow{c - a}$$

IK - numerical

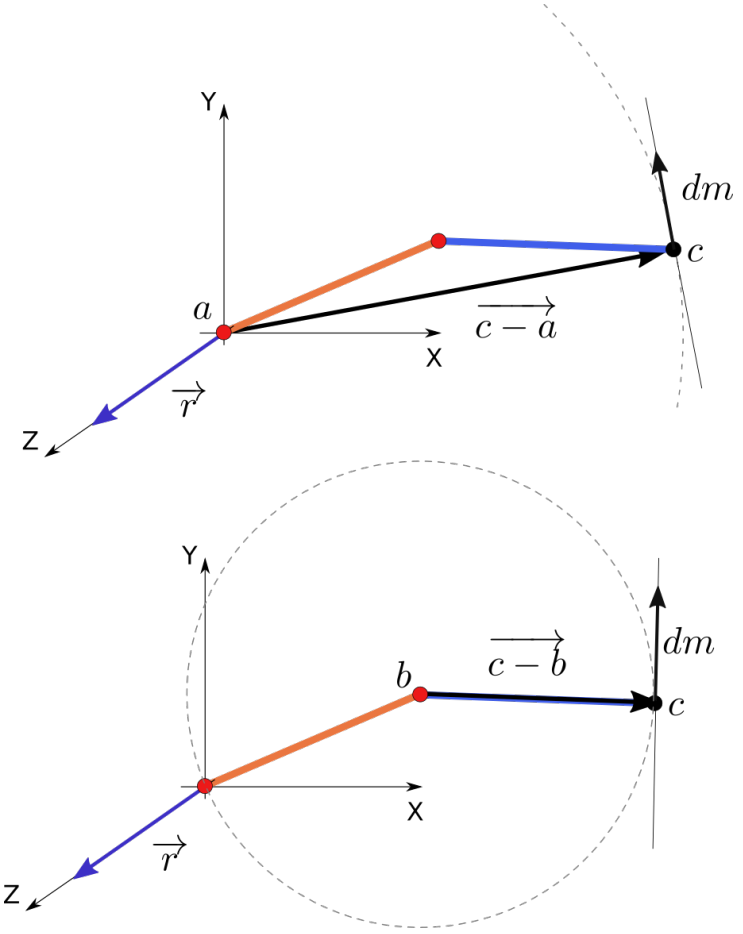
- Jacobian

$$J = \begin{bmatrix} \frac{\delta c_x}{\delta \theta_0} & \frac{\delta c_x}{\delta \theta_1} \\ \frac{\delta c_y}{\delta \theta_0} & \frac{\delta c_y}{\delta \theta_1} \end{bmatrix}$$

$$dm_i = \left(\frac{\delta c_x}{\delta \theta_i}, \frac{\delta c_y}{\delta \theta_i}, 0 \right) = \vec{r} \times \overrightarrow{c - p_i}$$

$$J = \begin{bmatrix} \left(\vec{r} \times \overrightarrow{c - a} \right)_x & \left(\vec{r} \times \overrightarrow{c - b} \right)_x \\ \left(\vec{r} \times \overrightarrow{c - a} \right)_y & \left(\vec{r} \times \overrightarrow{c - b} \right)_y \end{bmatrix}$$

$$\vec{r} = (0, 0, 1)$$



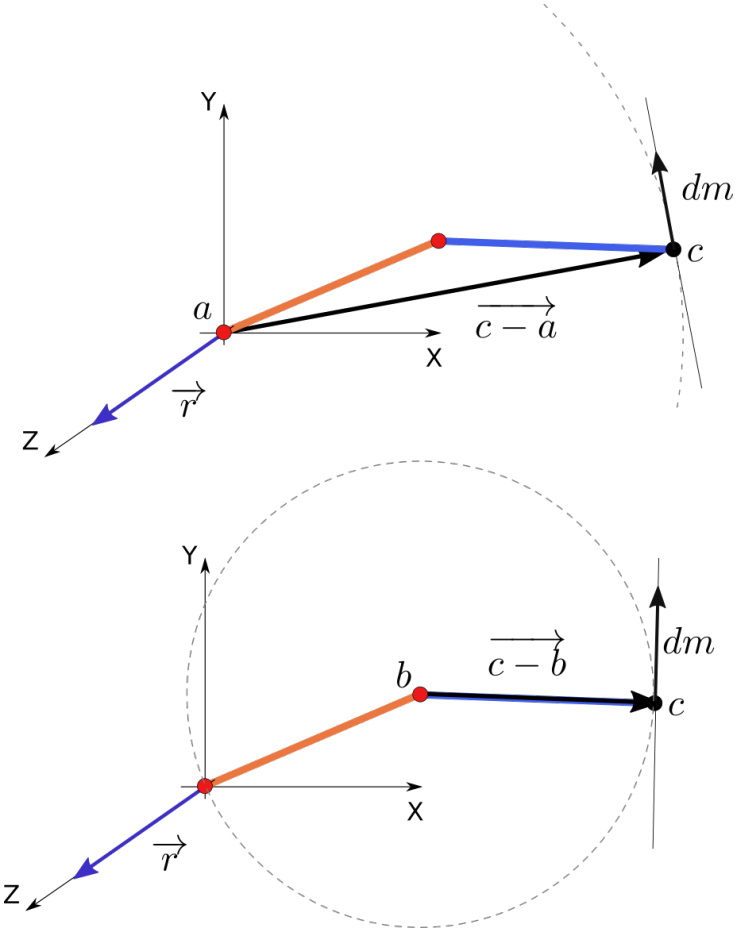
IK - numerical

- Jacobian

$$\mathbf{J}\Delta\theta = \Delta e$$

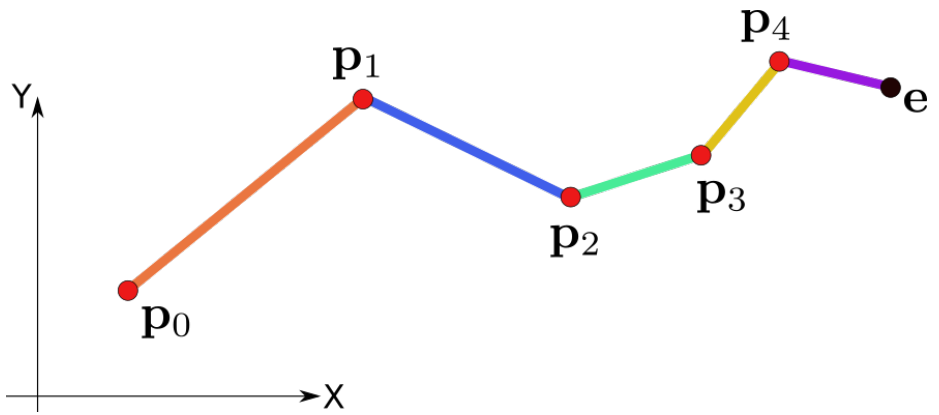
$$\begin{bmatrix} \frac{\delta c_x}{\delta \theta_0} & \frac{\delta c_x}{\delta \theta_1} \\ \frac{\delta c_y}{\delta \theta_0} & \frac{\delta c_y}{\delta \theta_1} \end{bmatrix} \begin{bmatrix} \Delta\theta_0 \\ \Delta\theta_1 \end{bmatrix} = \begin{bmatrix} \Delta c_x \\ \Delta c_y \end{bmatrix}$$

$$\Delta\theta = \mathbf{J}^{-1}\Delta e$$



IK - numerical

- n joints



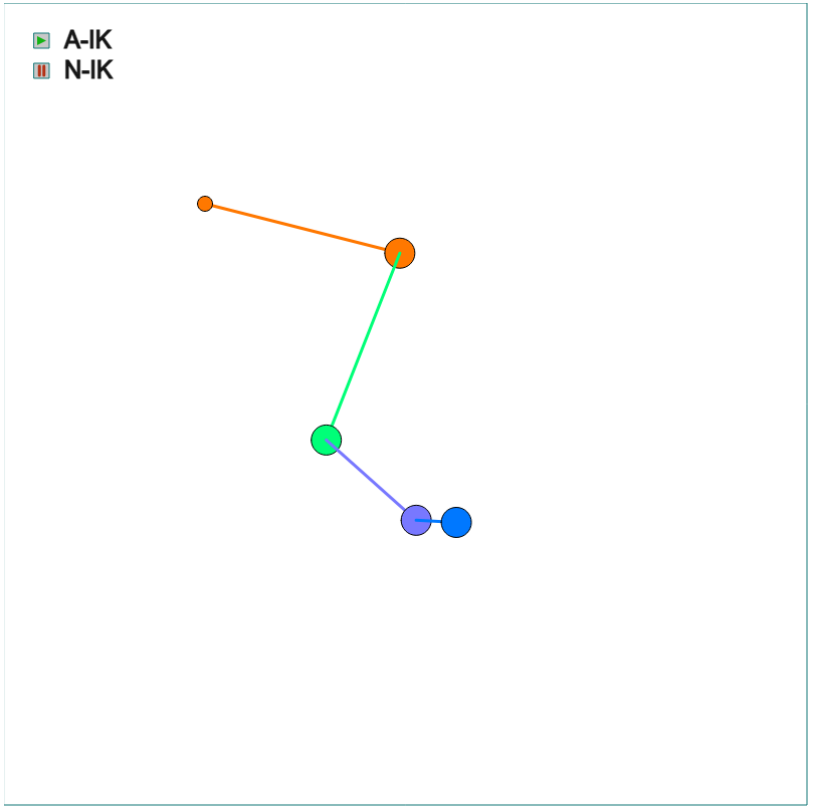
$$\Delta\theta = \mathbf{J}^{-1}\Delta e$$

$$\mathbf{J}^{-1} \approx \mathbf{J}^{T}$$

$$J = \begin{bmatrix} \frac{\delta e_x}{\delta\theta_0} & \frac{\delta e_x}{\delta\theta_1} & \frac{\delta e_x}{\delta\theta_2} & \frac{\delta e_x}{\delta\theta_3} & \frac{\delta e_x}{\delta\theta_4} & \frac{\delta e_x}{\delta\theta_5} \\ \frac{\delta e_y}{\delta\theta_0} & \frac{\delta e_y}{\delta\theta_1} & \frac{\delta e_y}{\delta\theta_2} & \frac{\delta e_y}{\delta\theta_3} & \frac{\delta e_y}{\delta\theta_4} & \frac{\delta e_y}{\delta\theta_5} \end{bmatrix}$$

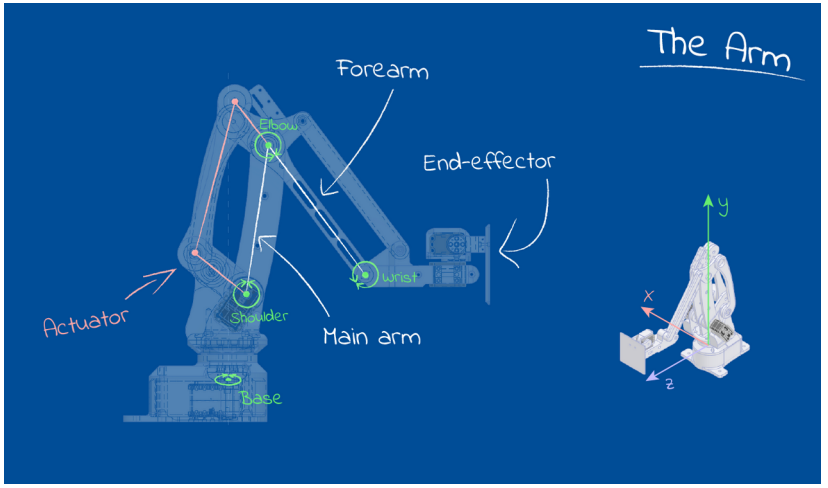
$$J = \begin{bmatrix} \left(\vec{r} \times \overrightarrow{\mathbf{e}-\mathbf{v}_0}\right)_x & \left(\vec{r} \times \overrightarrow{\mathbf{e}-\mathbf{v}_1}\right)_x & \left(\vec{r} \times \overrightarrow{\mathbf{e}-\mathbf{v}_2}\right)_x & \left(\vec{r} \times \overrightarrow{\mathbf{e}-\mathbf{v}_3}\right)_x & \left(\vec{r} \times \overrightarrow{\mathbf{e}-\mathbf{v}_4}\right)_x \\ \left(\vec{r} \times \overrightarrow{\mathbf{e}-\mathbf{v}_0}\right)_y & \left(\vec{r} \times \overrightarrow{\mathbf{e}-\mathbf{v}_1}\right)_y & \left(\vec{r} \times \overrightarrow{\mathbf{e}-\mathbf{v}_2}\right)_y & \left(\vec{r} \times \overrightarrow{\mathbf{e}-\mathbf{v}_3}\right)_y & \left(\vec{r} \times \overrightarrow{\mathbf{e}-\mathbf{v}_4}\right)_y \end{bmatrix}$$

IK - numerical



IK - numerical

- what about IK in 3D?
  - each joint has own rotation axis

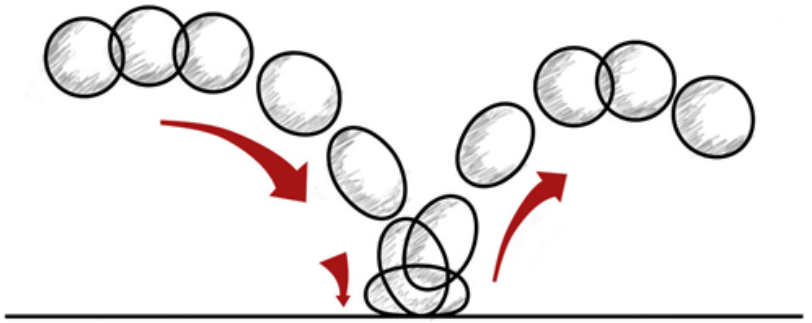


$$J = \begin{bmatrix} \frac{\delta e_x}{\delta \theta_0} & \frac{\delta e_x}{\delta \theta_1} & \frac{\delta e_x}{\delta \theta_2} & \frac{\delta e_x}{\delta \theta_3} & \frac{\delta e_x}{\delta \theta_4} & \frac{\delta e_x}{\delta \theta_5} \\ \frac{\delta e_y}{\delta \theta_0} & \frac{\delta e_y}{\delta \theta_1} & \frac{\delta e_y}{\delta \theta_2} & \frac{\delta e_y}{\delta \theta_3} & \frac{\delta e_y}{\delta \theta_4} & \frac{\delta e_y}{\delta \theta_5} \\ \frac{\delta e_z}{\delta \theta_0} & \frac{\delta e_z}{\delta \theta_1} & \frac{\delta e_z}{\delta \theta_2} & \frac{\delta e_z}{\delta \theta_3} & \frac{\delta e_z}{\delta \theta_4} & \frac{\delta e_z}{\delta \theta_5} \end{bmatrix}$$

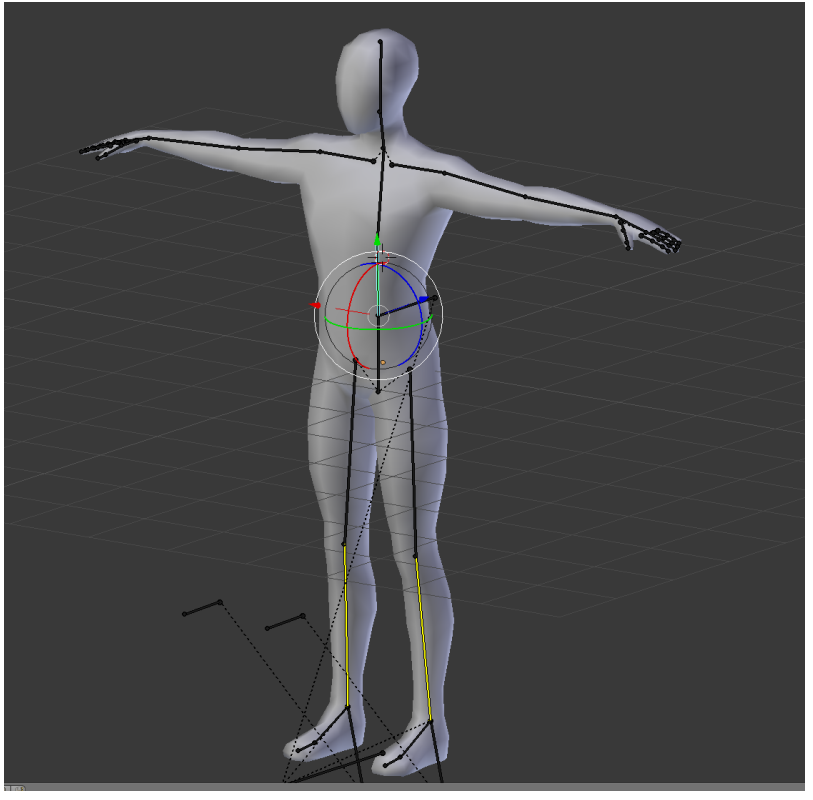
$$J = \begin{bmatrix} \left(\vec{r}_0 \times \overrightarrow{\mathbf{e} - \mathbf{v}_0}\right)_x & \left(\vec{r}_1 \times \overrightarrow{\mathbf{e} - \mathbf{v}_1}\right)_x & \left(\vec{r}_2 \times \overrightarrow{\mathbf{e} - \mathbf{v}_2}\right)_x & \left(\vec{r}_3 \times \overrightarrow{\mathbf{e} - \mathbf{v}_3}\right)_x & \left(\vec{r}_4 \times \overrightarrow{\mathbf{e} - \mathbf{v}_4}\right)_x \\ \left(\vec{r}_0 \times \overrightarrow{\mathbf{e} - \mathbf{v}_0}\right)_y & \left(\vec{r}_1 \times \overrightarrow{\mathbf{e} - \mathbf{v}_1}\right)_y & \left(\vec{r}_2 \times \overrightarrow{\mathbf{e} - \mathbf{v}_2}\right)_y & \left(\vec{r}_3 \times \overrightarrow{\mathbf{e} - \mathbf{v}_3}\right)_y & \left(\vec{r}_4 \times \overrightarrow{\mathbf{e} - \mathbf{v}_4}\right)_y \\ \left(\vec{r}_0 \times \overrightarrow{\mathbf{e} - \mathbf{v}_0}\right)_z & \left(\vec{r}_1 \times \overrightarrow{\mathbf{e} - \mathbf{v}_1}\right)_z & \left(\vec{r}_2 \times \overrightarrow{\mathbf{e} - \mathbf{v}_2}\right)_z & \left(\vec{r}_3 \times \overrightarrow{\mathbf{e} - \mathbf{v}_3}\right)_z & \left(\vec{r}_4 \times \overrightarrow{\mathbf{e} - \mathbf{v}_4}\right)_z \end{bmatrix}$$

soft world

- not everything are rigid balls and robots



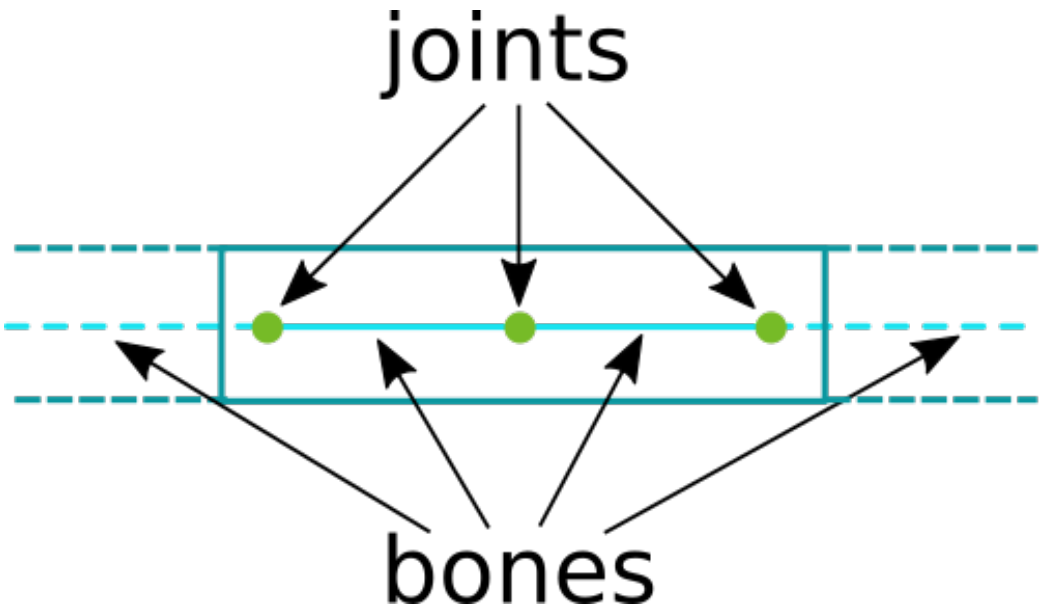
rigging and skinning



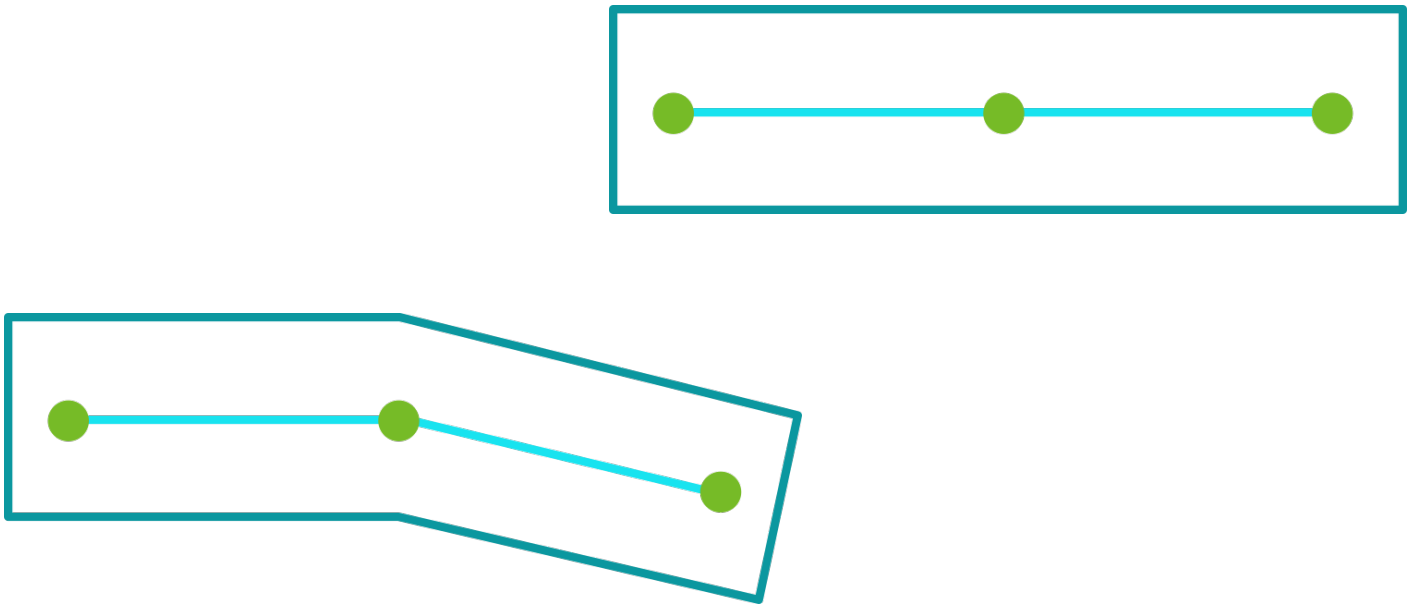
https://cgi.tutsplus.com/tutorials/building-a-basic-low-poly-character-rig-in-blender--cg-16955

rigging

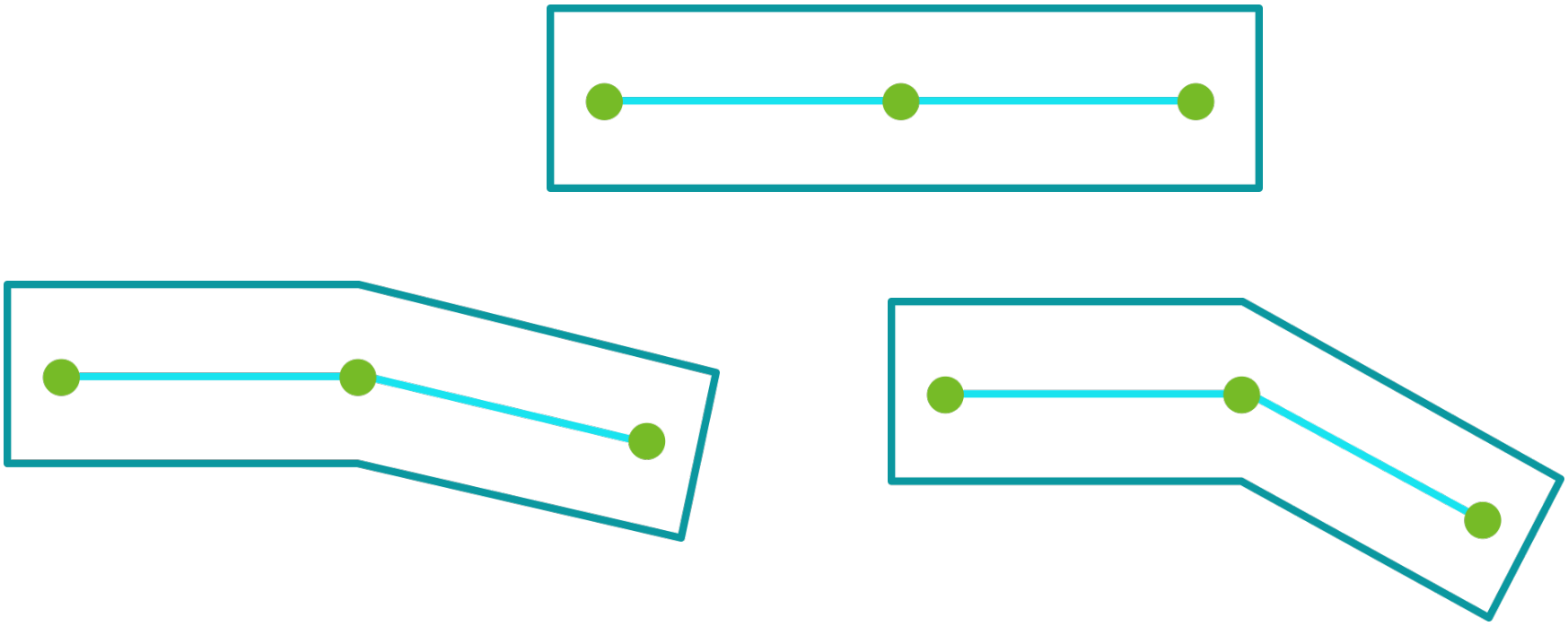
- skeleton structure to manipulate model
  - skeleton is made of joints and bones



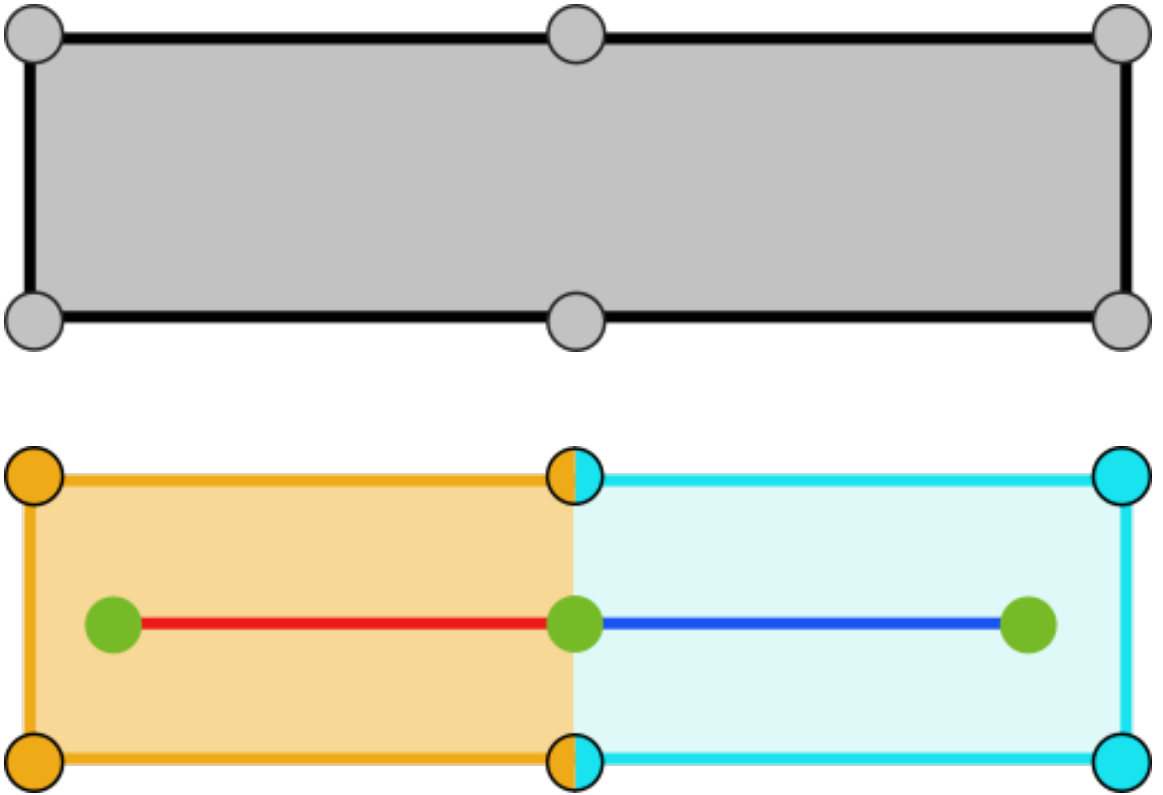
rigging



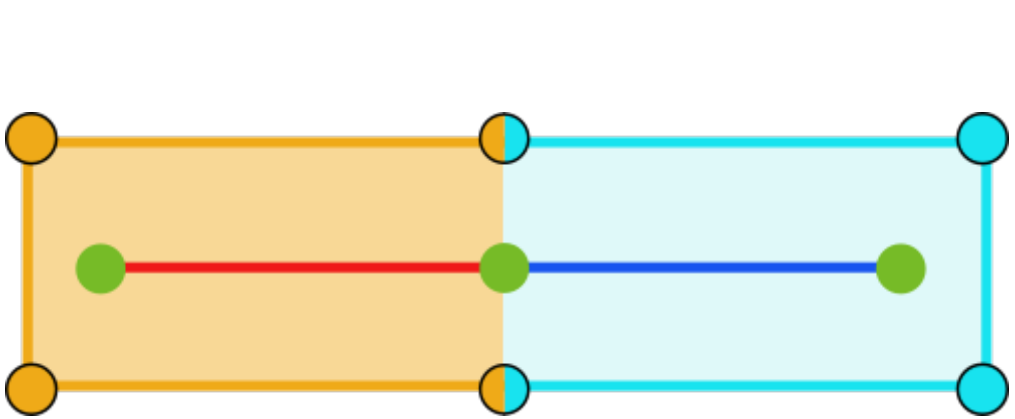
rigging

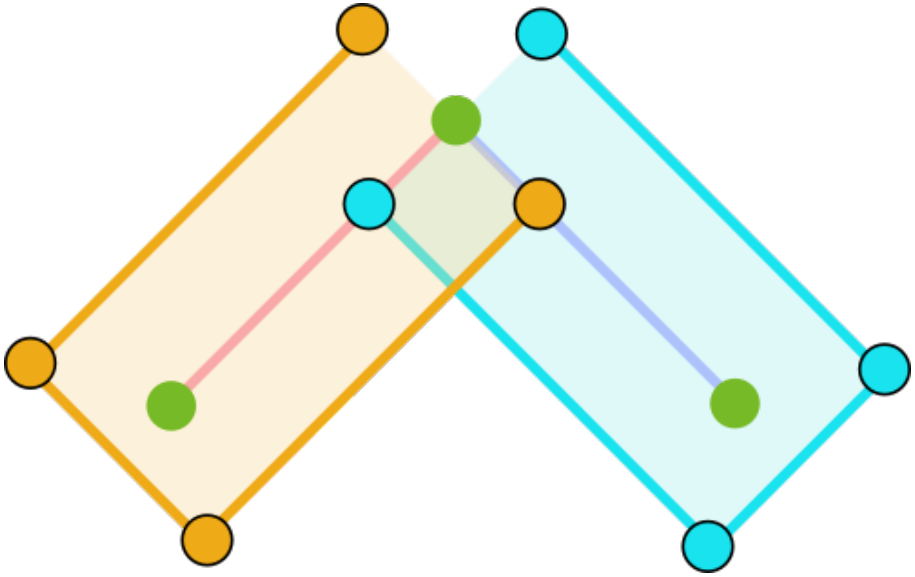


rigid transform

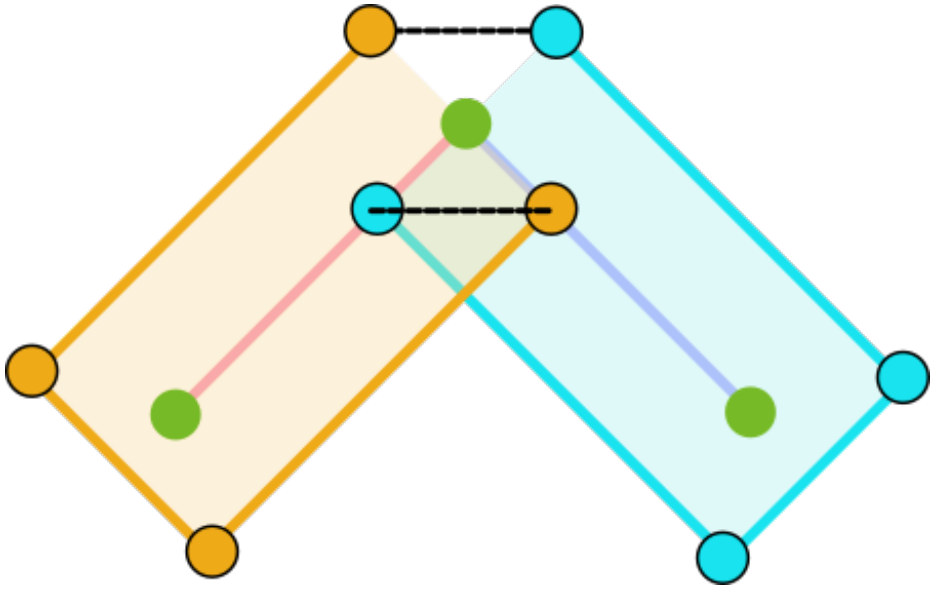


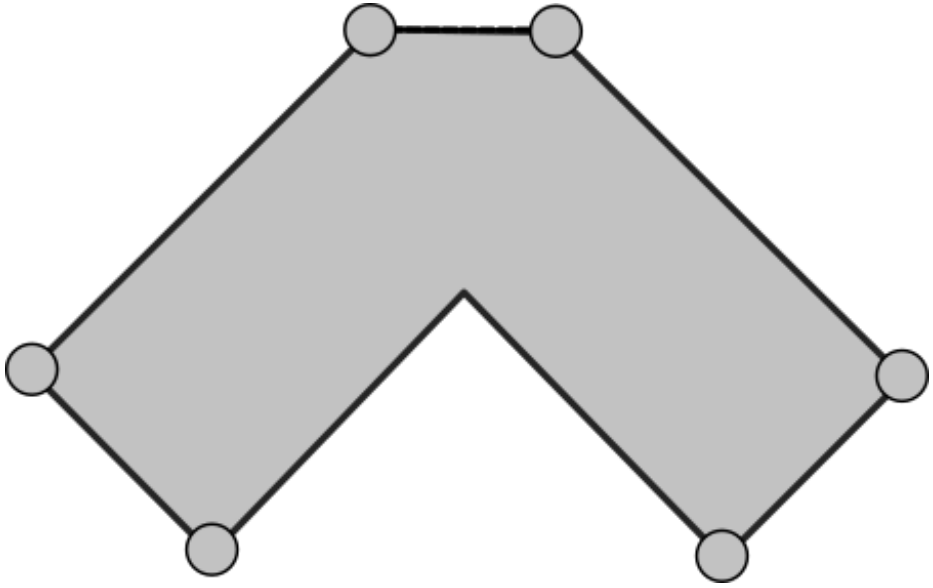
rigid transform



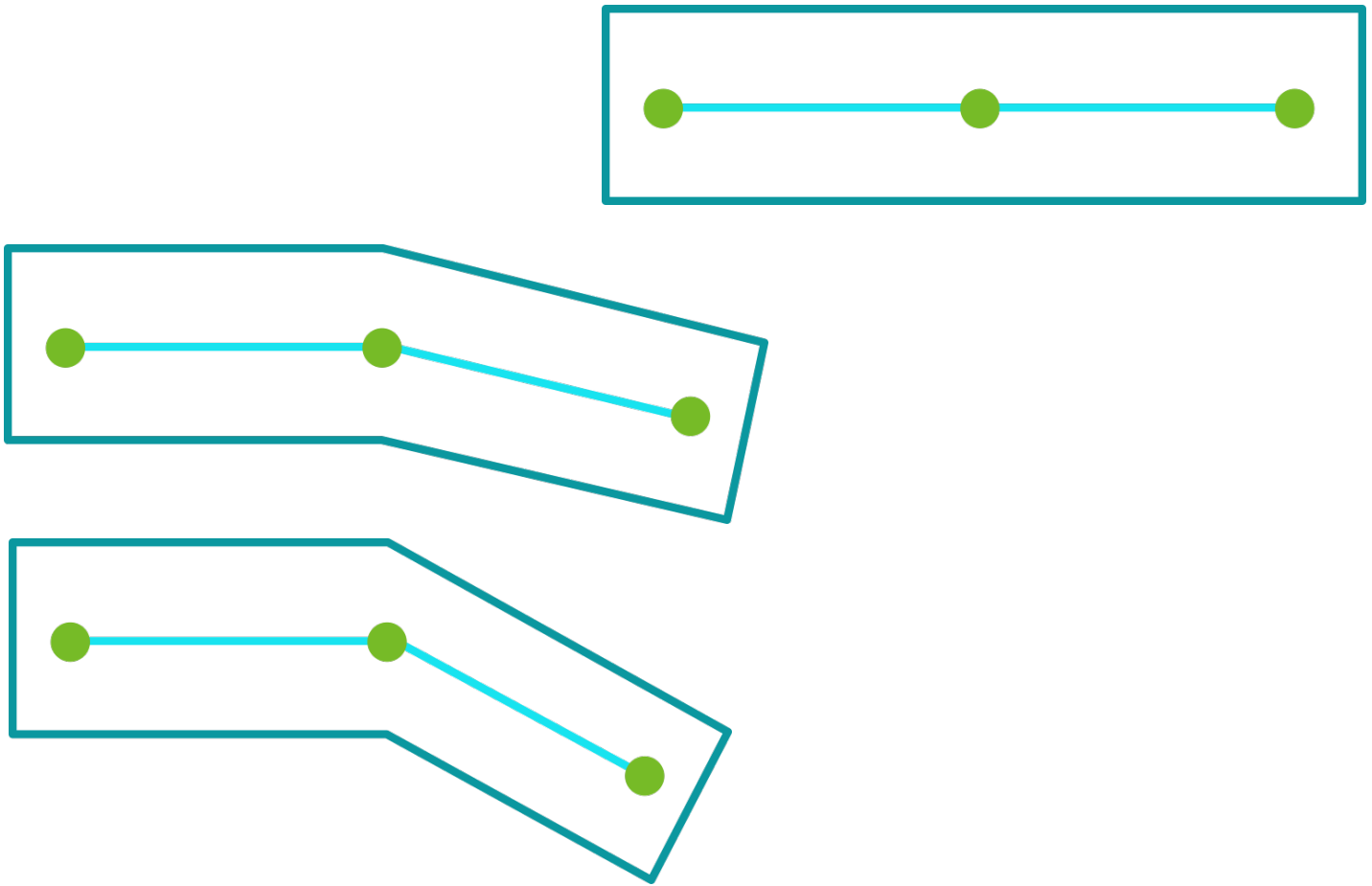


rigid transform

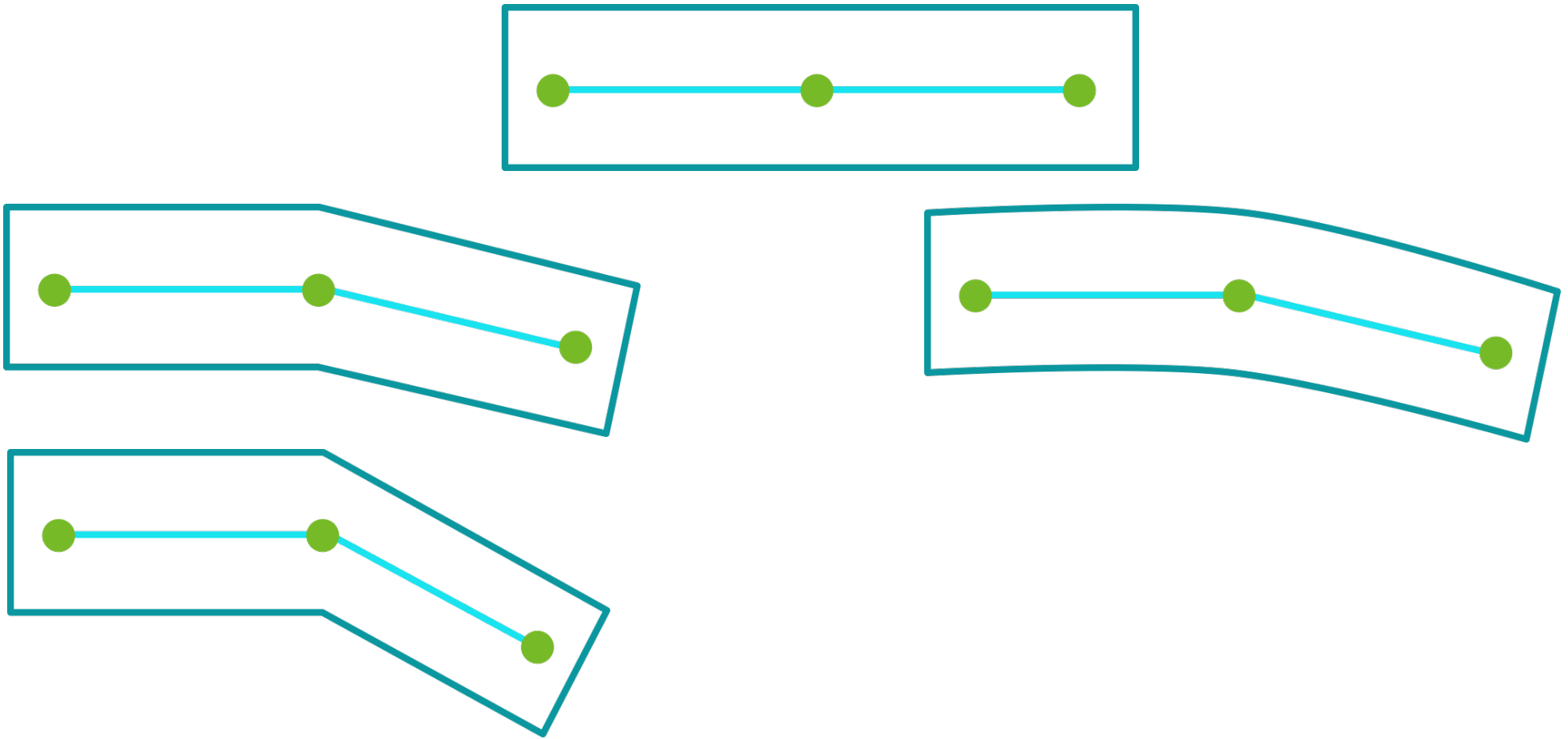




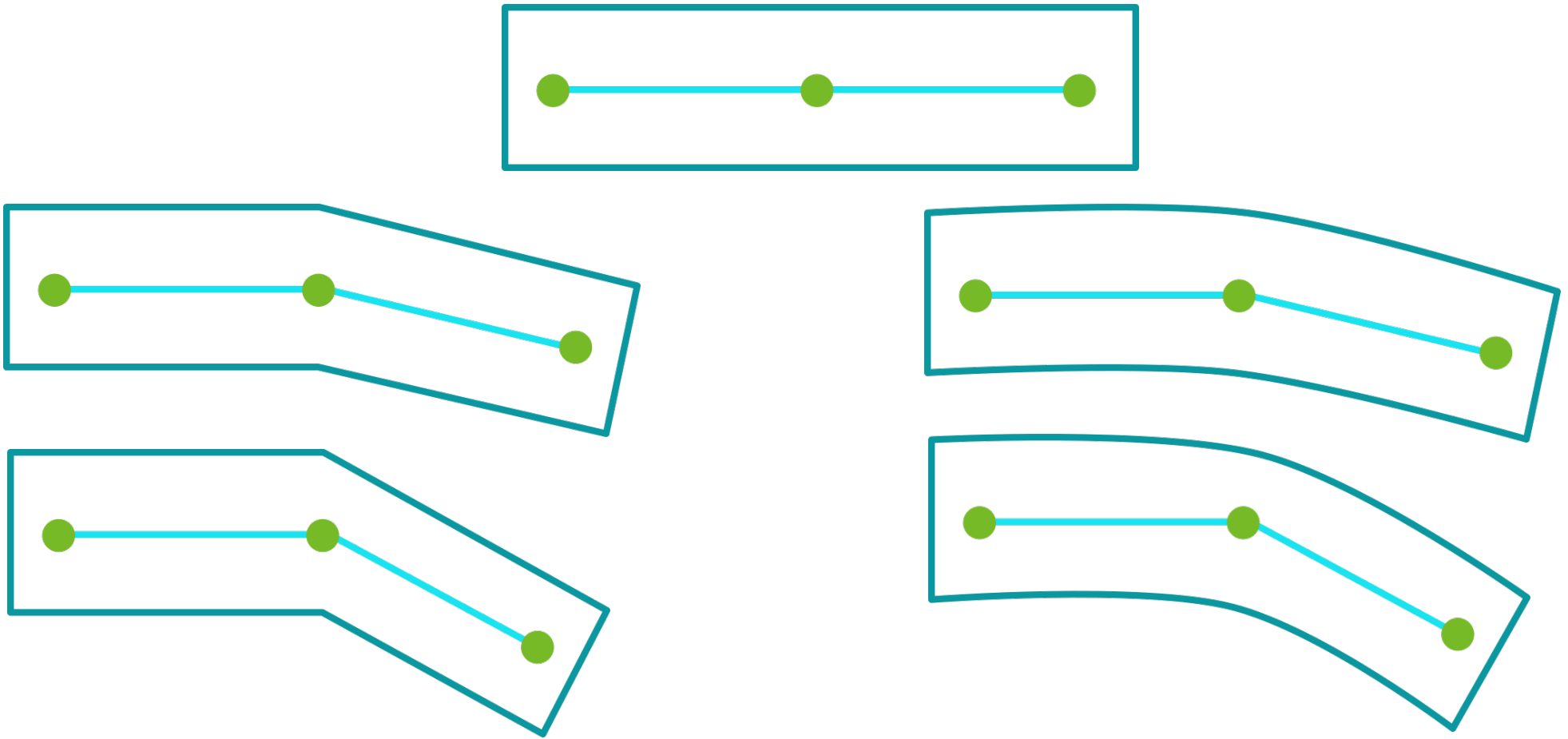
rigging



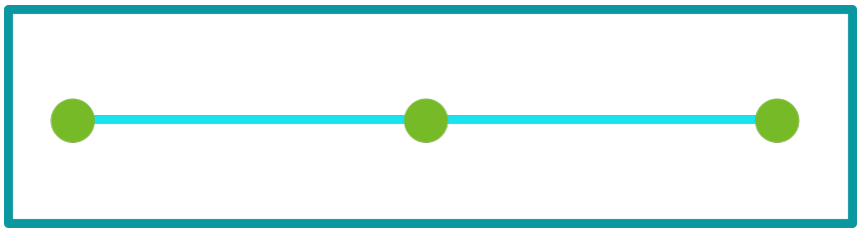
rigging

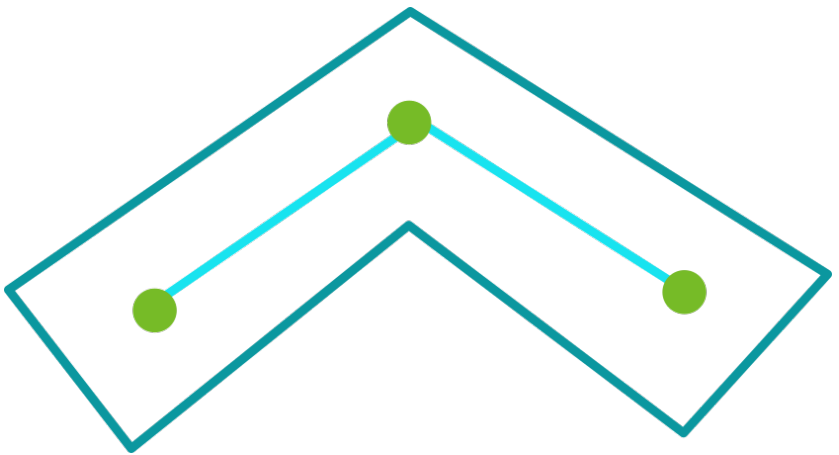


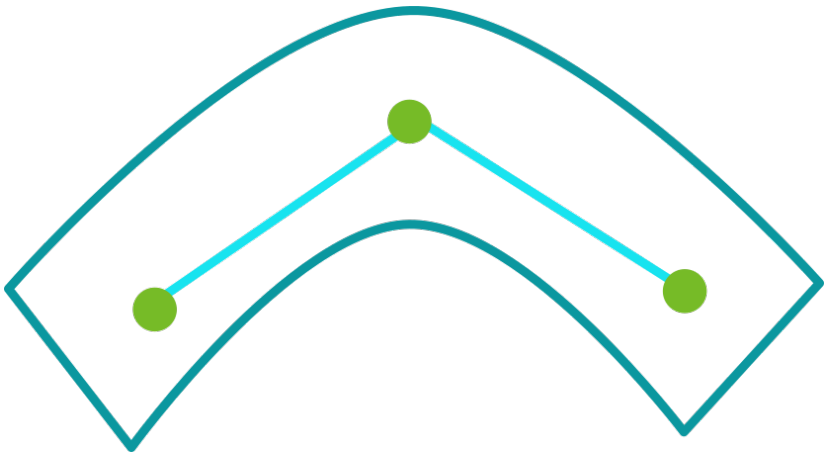
rigging



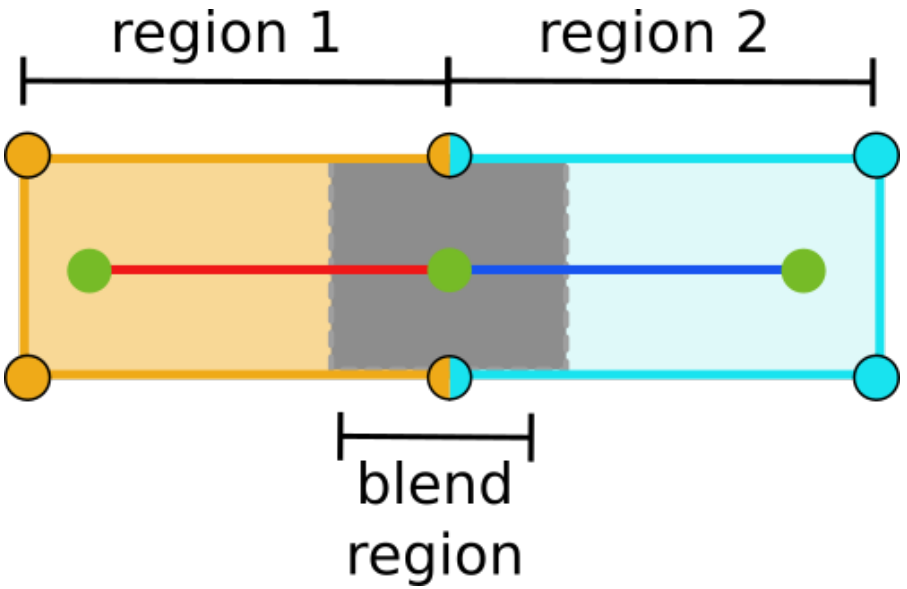
skinning



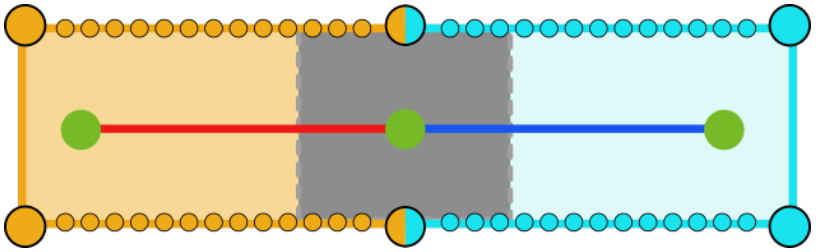




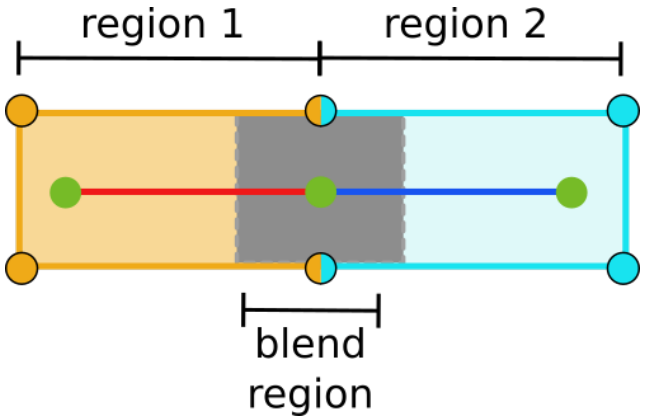
linear blend skinning



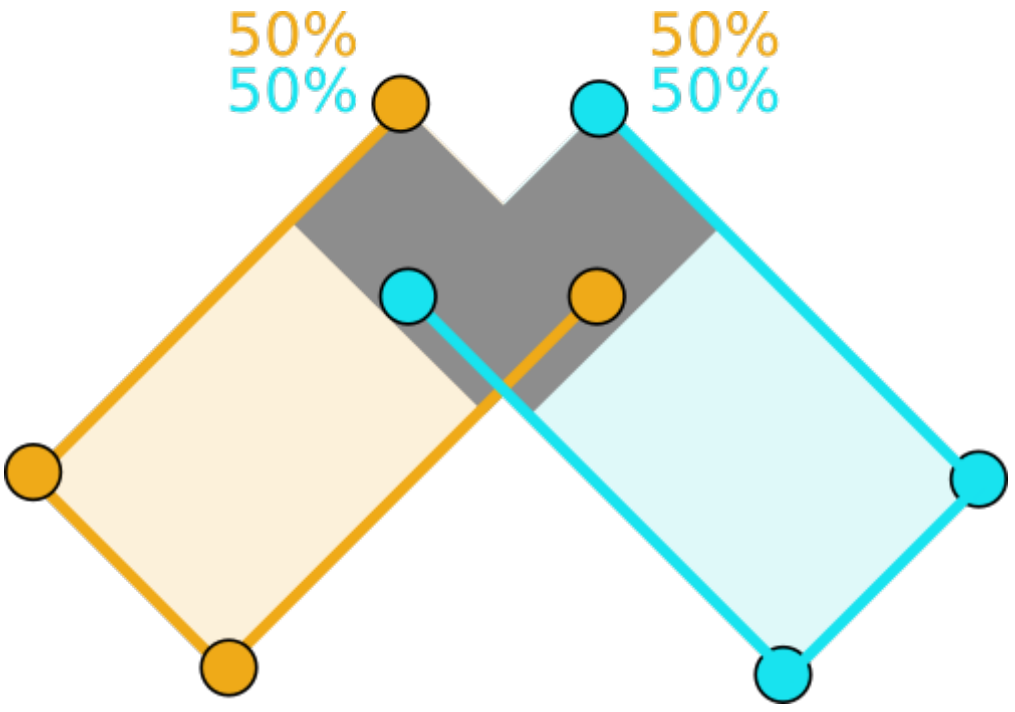
$$v'_j = \sum_i w_{ij} T_i v_j$$



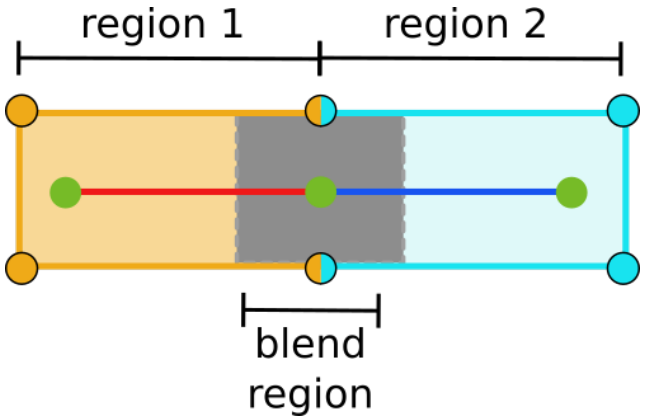
linear blend skinning



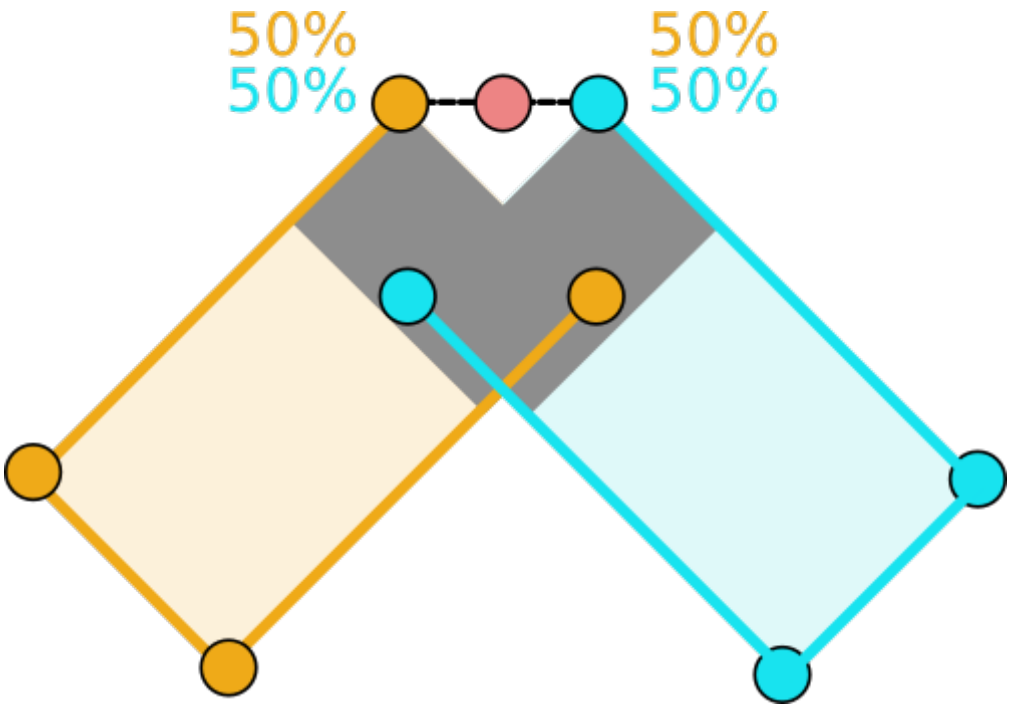
$$v'_j = \sum_i w_{ij} T_i v_j$$



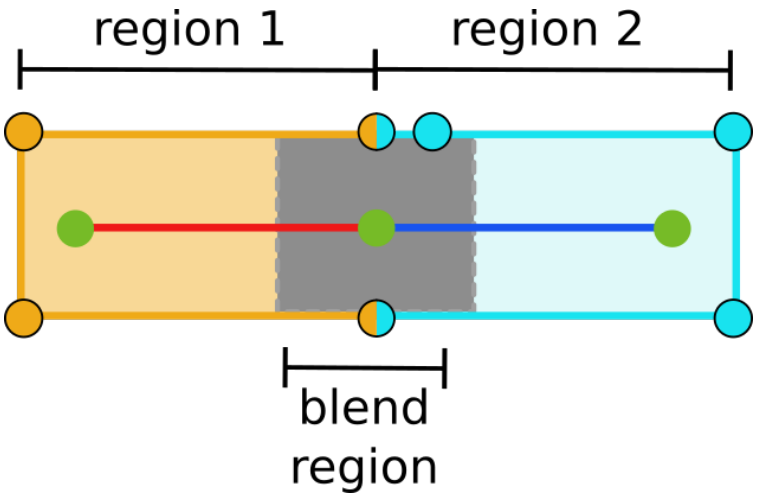
linear blend skinning



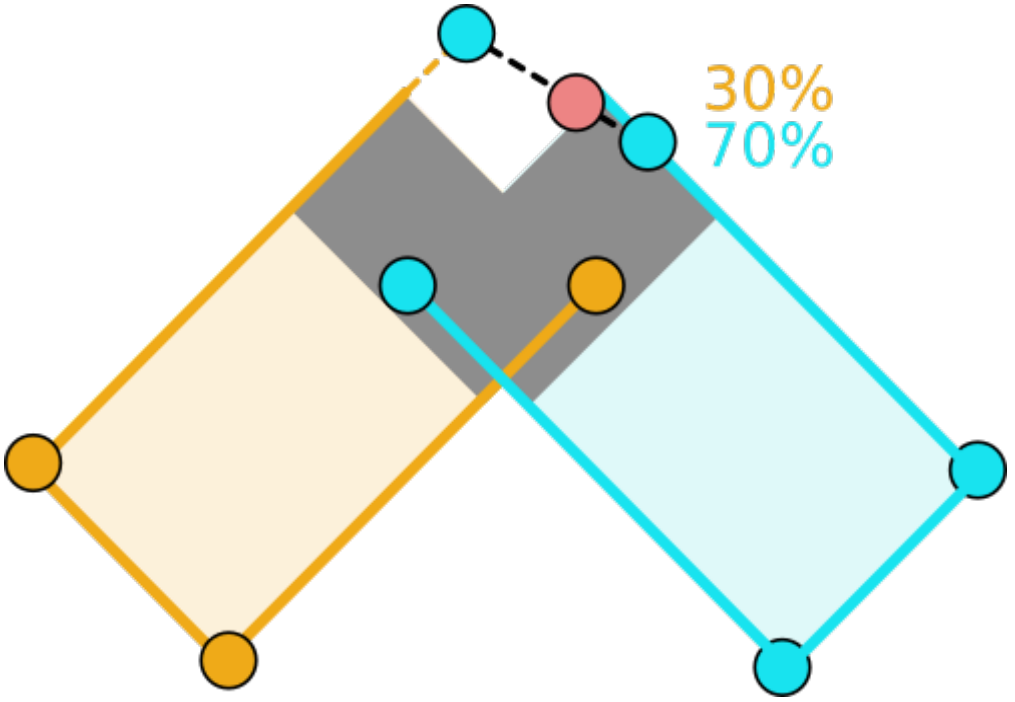
$$v'_j = \sum_i w_{ij} T_i v_j$$



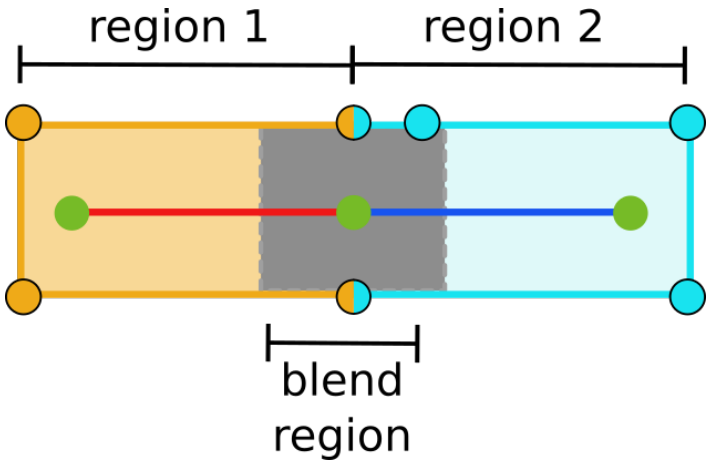
linear blend skinning



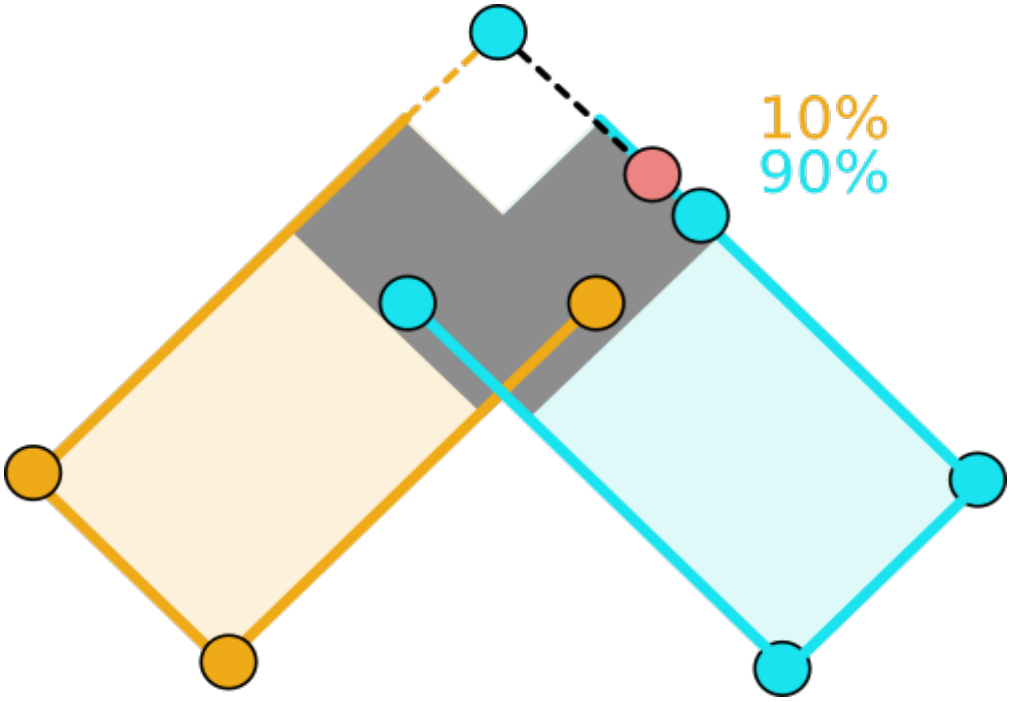
$$v'_j = \sum_i w_{ij} T_i v_j$$



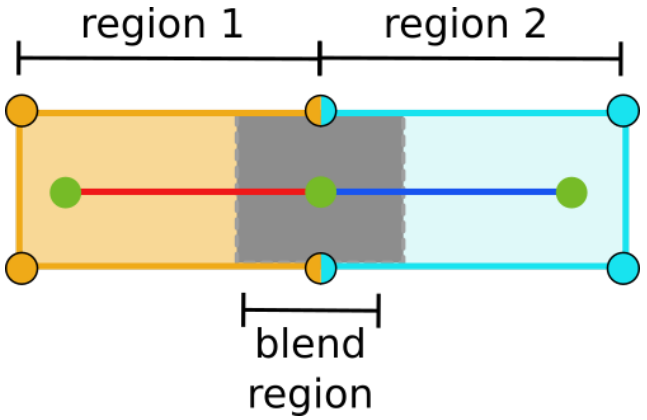
linear blend skinning



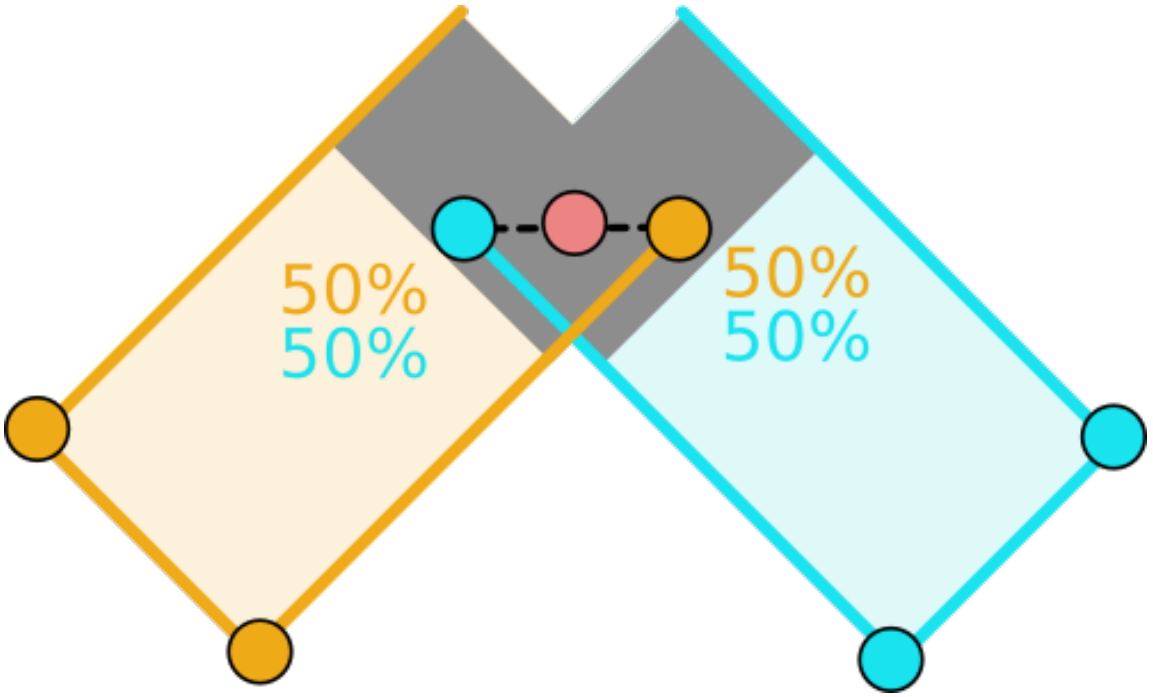
$$v'_j = \sum_i w_{ij} T_i v_j$$



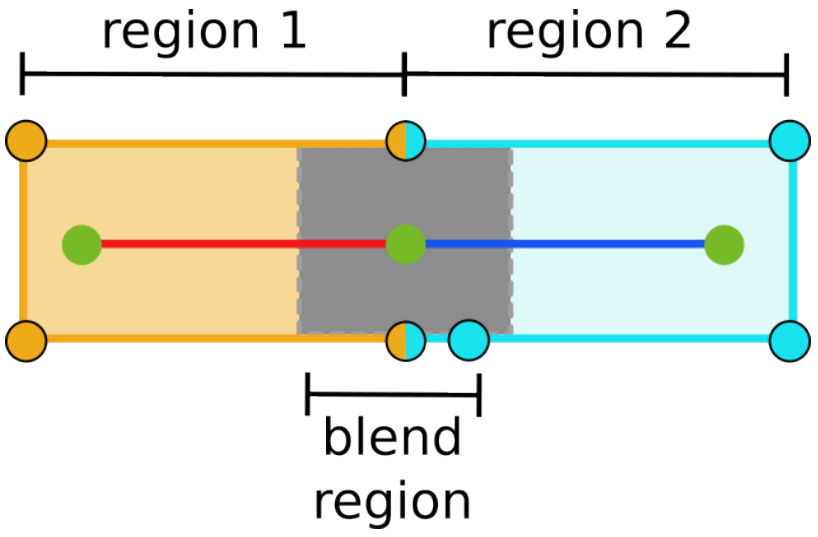
linear blend skinning

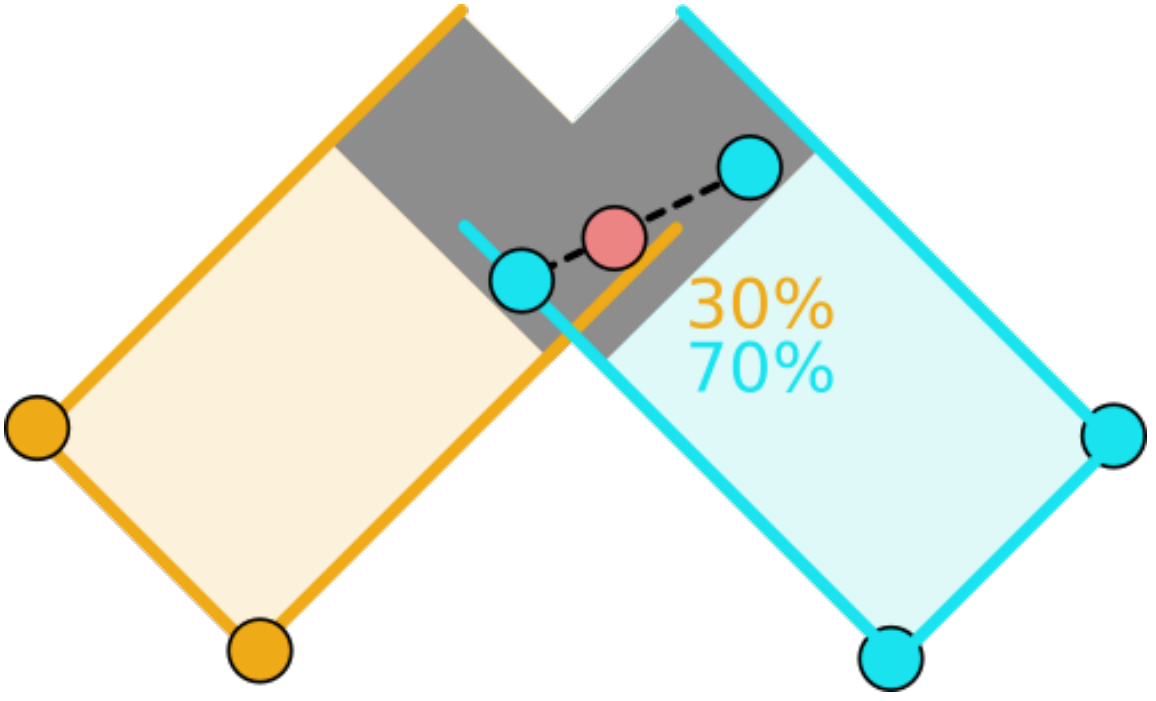


$$v'_j = \sum_i w_{ij} T_i v_j$$

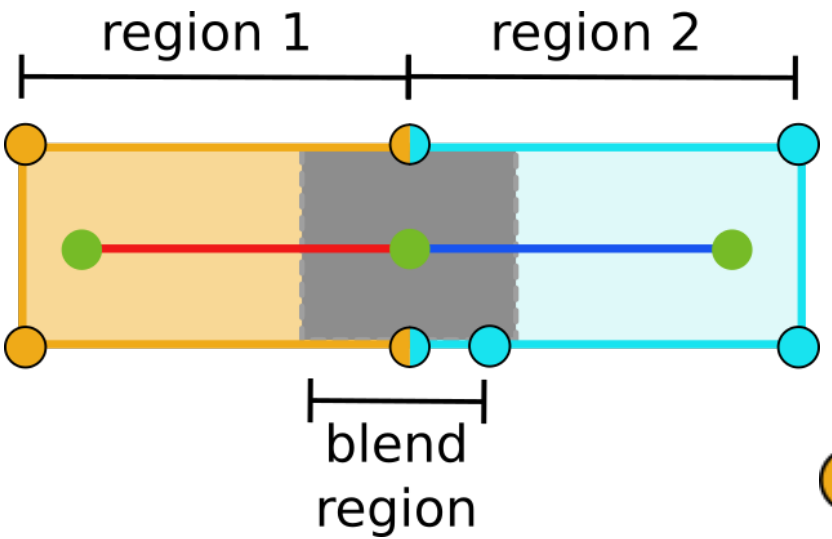


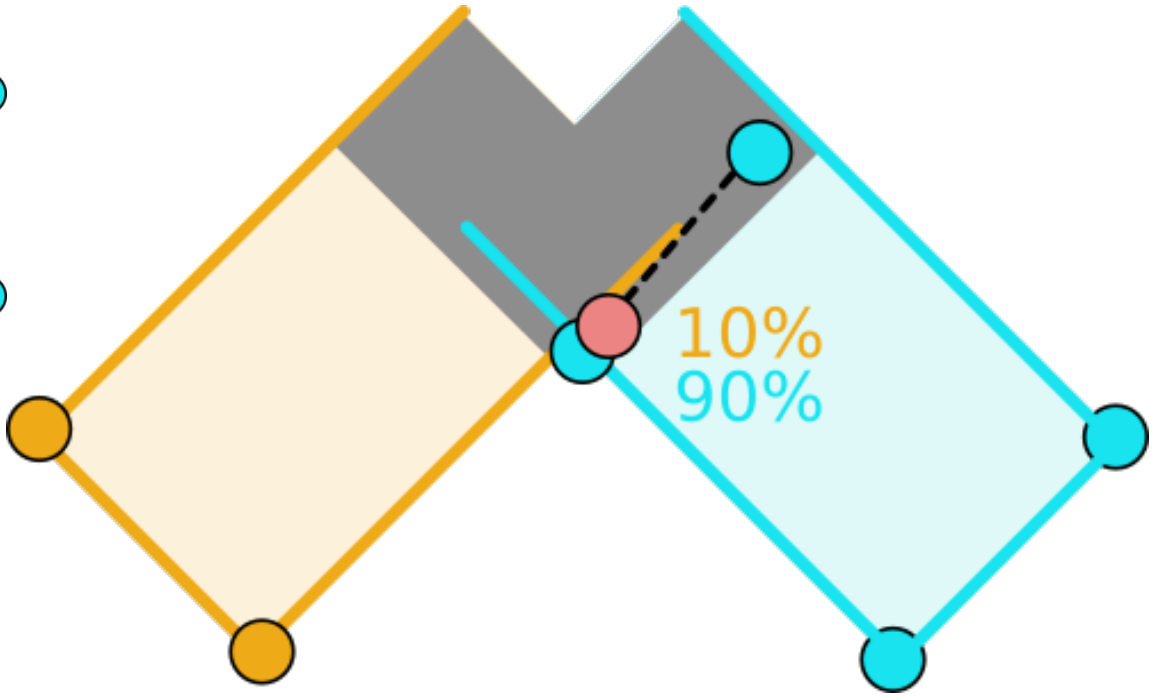
linear blend skinning



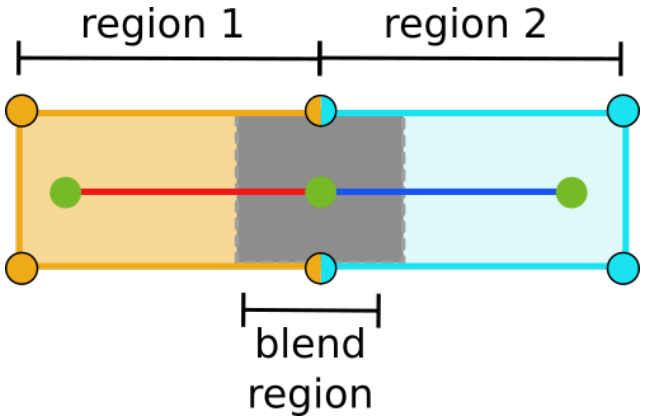


linear blend skinning

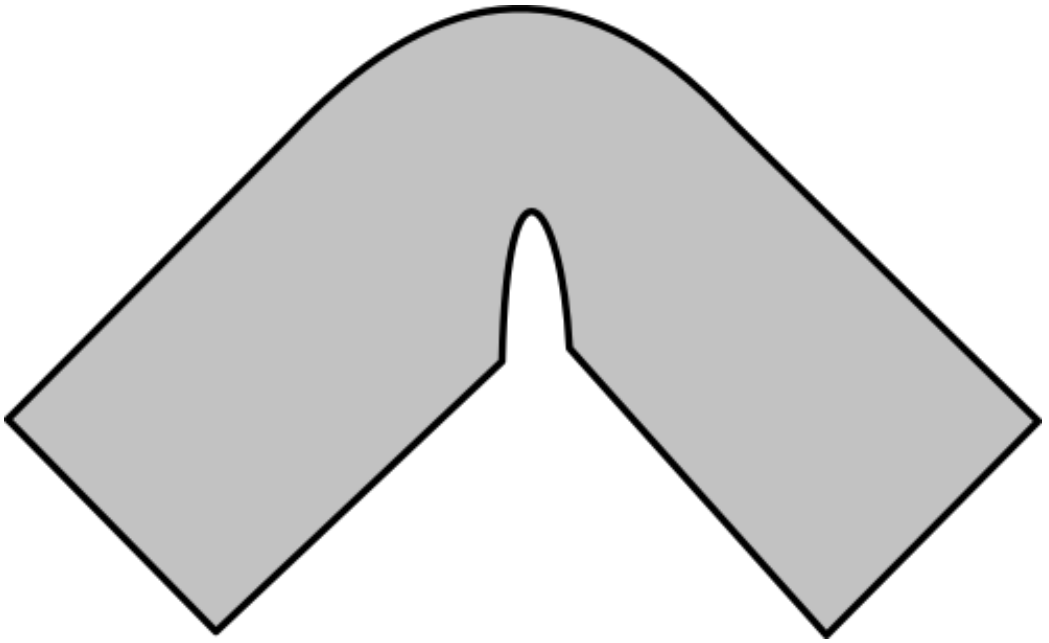




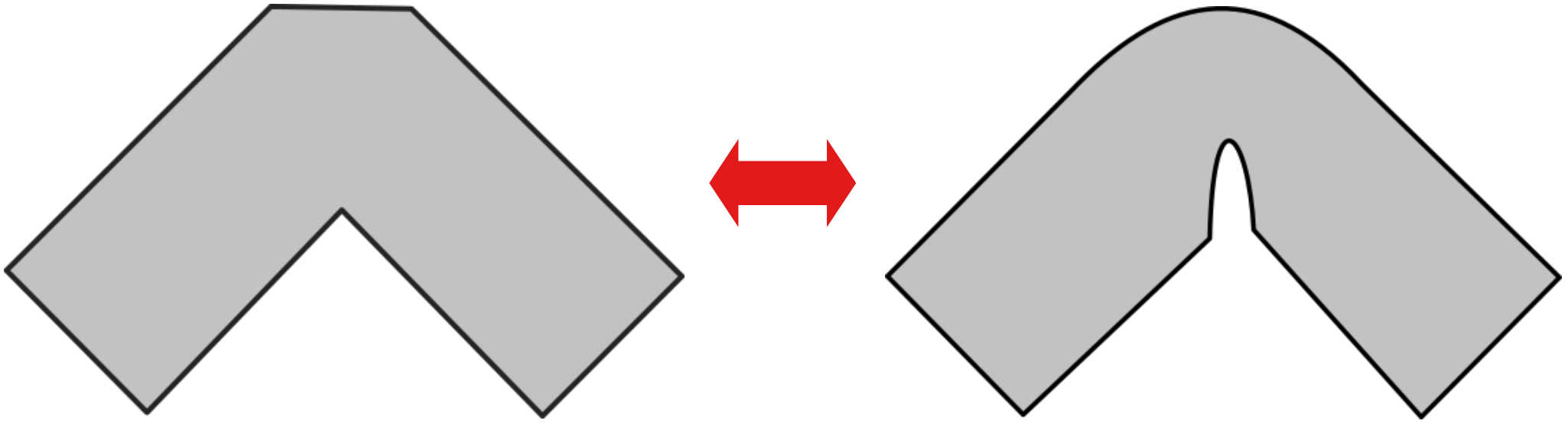
linear blend skinning



$$v'_j = \sum_i w_{ij} T_i v_j$$



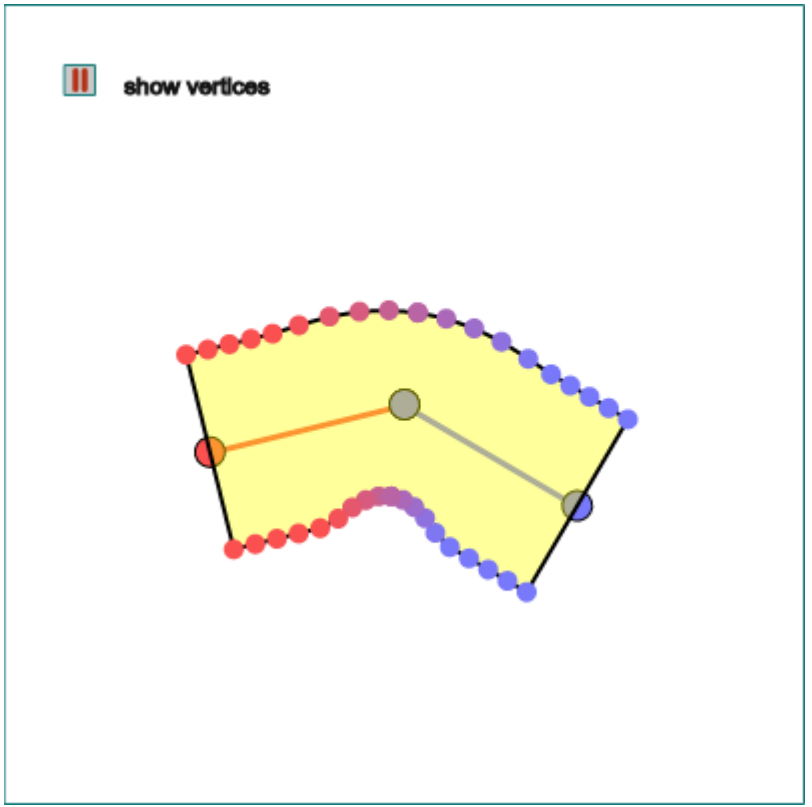
linear blend skinning



rigid

LBS

demo



linear blend skinning

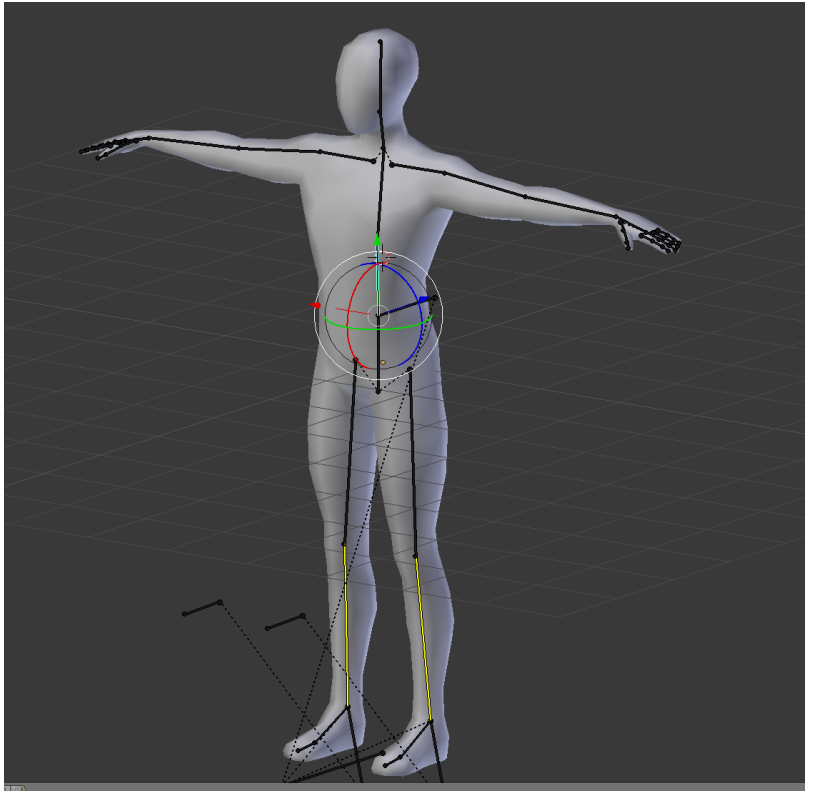
- don’t forget to interpolate the normal vectors!
- constrain weights to avoid undesirable scaling and artifacts

$$v'_j = \sum_i w_{ij} T_i v_j$$

$$\sum_i w_{ij} = 1, w_{ij} \geq 0$$

$$n'_j = \sum_i w_{ij} T_i^{-T} n_j$$

rigging + LBS



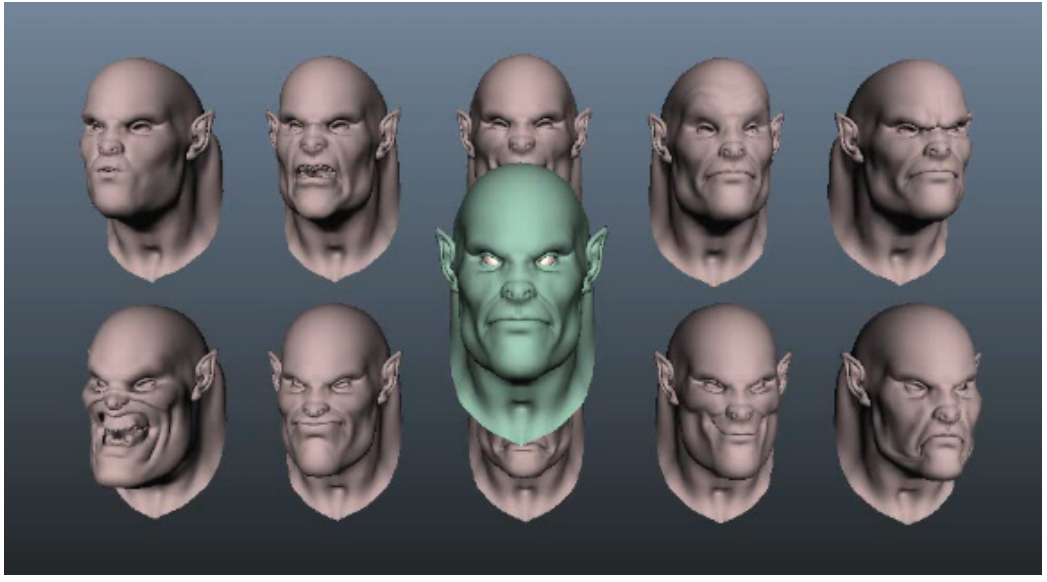
https://cgi.tutsplus.com/tutorials/building-a-basic-low-poly-character-rig-in-blender--cg-16955

realistic movements

- what about faces
  - can we use the skeleton idea??

Blend Shapes

- simple idea (and math)
  - mix many pre-configured poses to create new intermediate ones



http://cgterminal.com/2011/10/15/maya-blend-shape-tips-and-tricks-by-steven-roselle/

Blend Shapes

- use differences to neutral pose
  - each P contains differences of all vertices to neutral pose
  - all poses have the same topology

$$\Delta \mathbf{P}_i = \mathbf{P}_i - \mathbf{P}_{neutral}$$

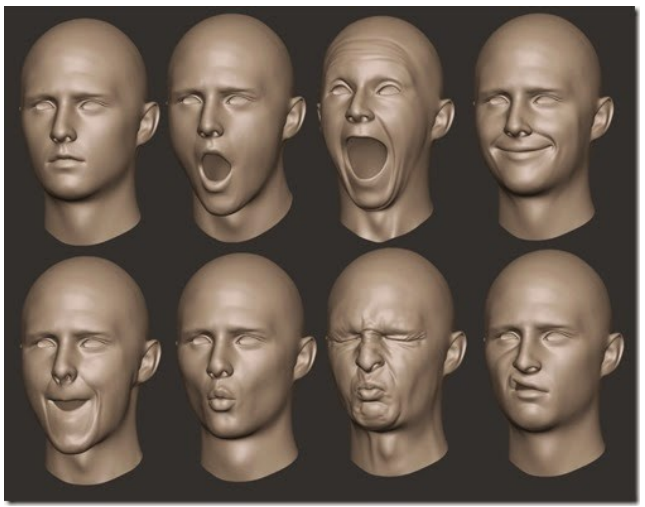
$$\mathbf{P}_{new} = \mathbf{P}_{neutral} + \sum_{k=1}^{N} w_k \Delta \mathbf{P}_k$$

$$\mathrm{P}_{new} = \mathbf{P}_{neutral} + \begin{bmatrix} \Delta \mathbf{P}_1 & \Delta \mathbf{P}_2 & \dots & \Delta \mathbf{P}_N \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix}$$

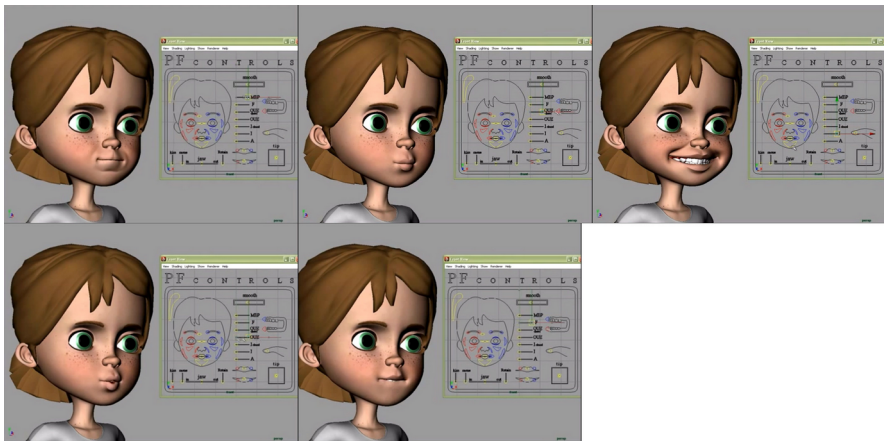
Blend Shapes

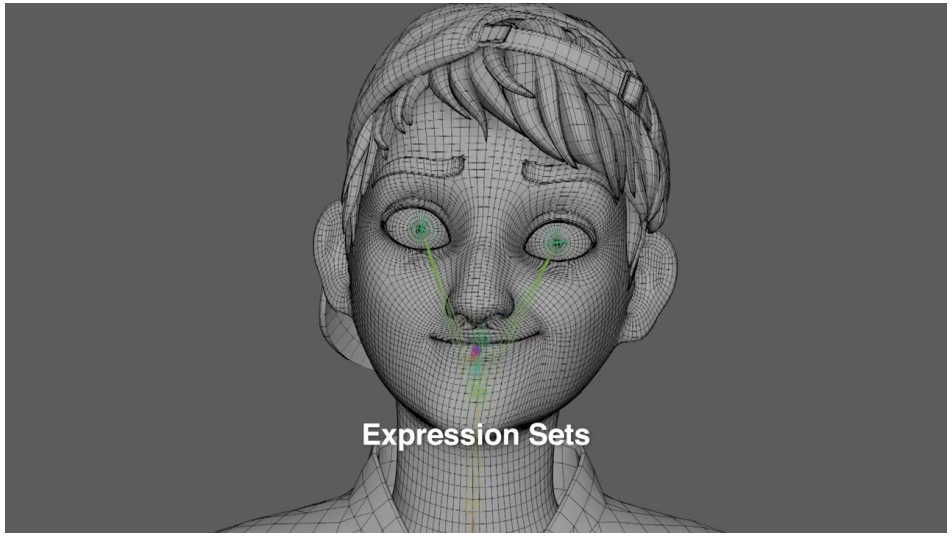
- how to set the weights?

$$\mathbf{P}_{new} = \mathbf{P}_{neutral} + \sum_{k=1}^{N} w_k \Delta \mathbf{P}_k$$



http://www.3dstudents.com/2015/02/blend-shapes.html



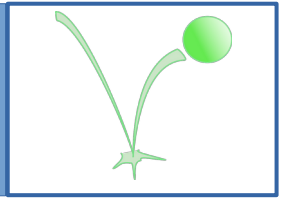


https://www.youtube.com/watch?v=Rrf97HVxvkI

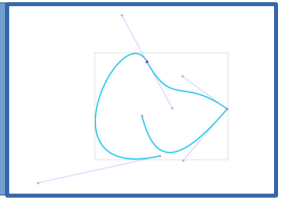
https://www.youtube.com/watch?v=dsS4FSkJf_8

computer animation

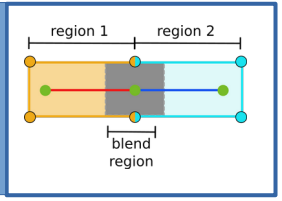
1) general ideas



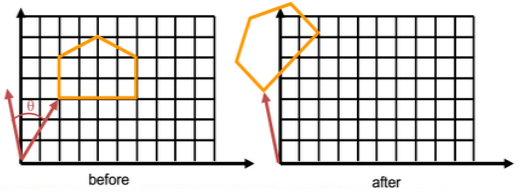
2) computational tools



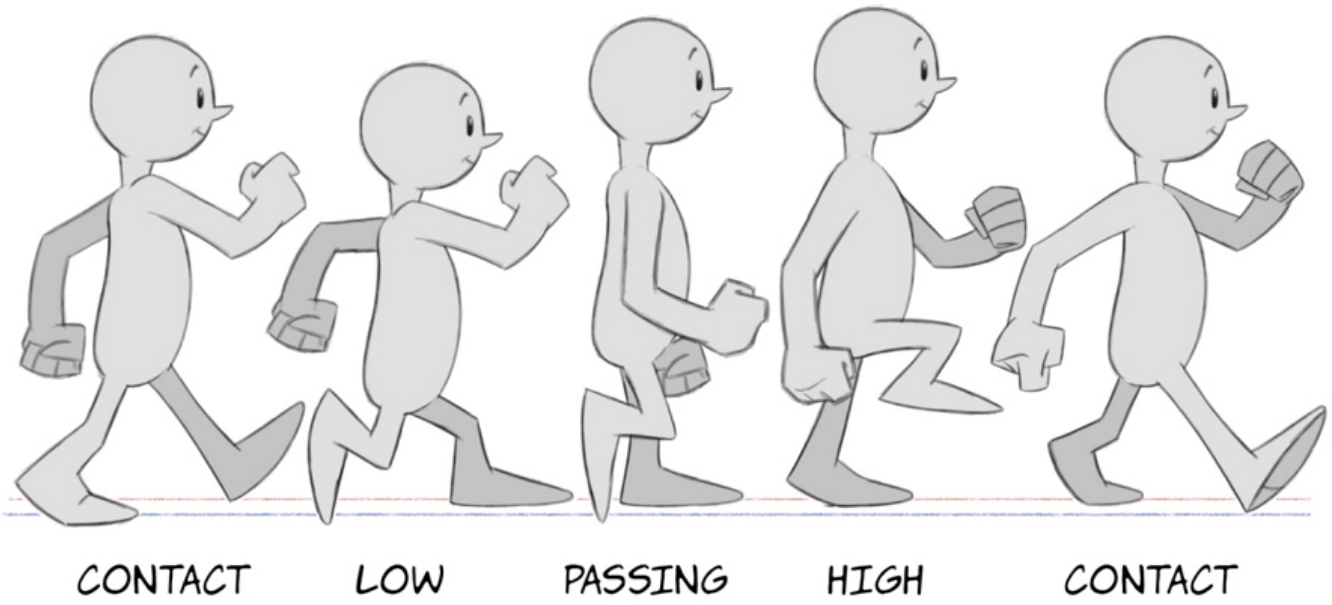
3) methods



transformations



computer animation



Ricardo Marroquim

Delft University of Technology (TU Delft)