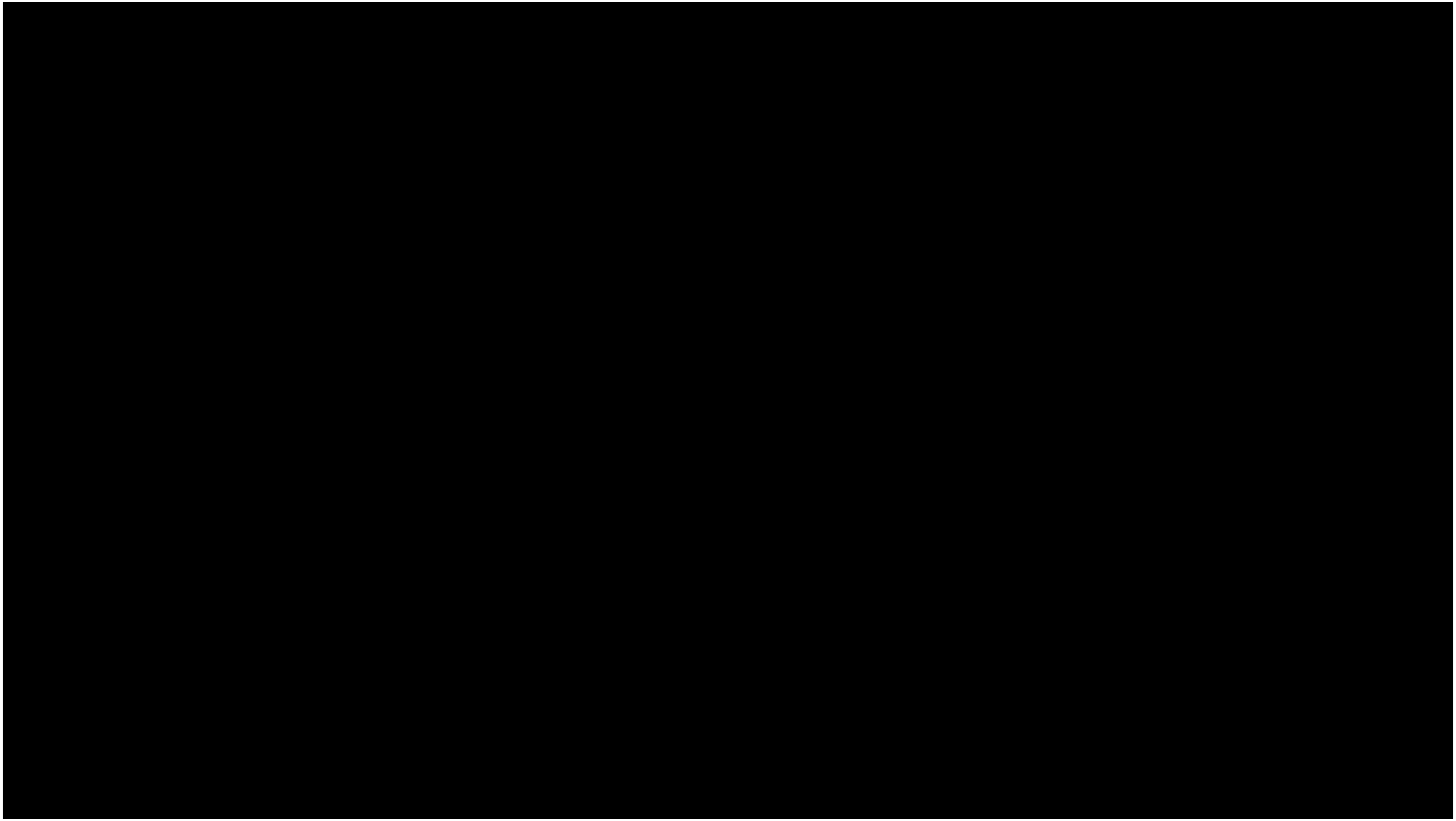
CSC418: Computer Graphics

Some slides and figures courtesy of Karan Singh
Some figures from Peter Shirley, “Fundamentals of Computer Graphics”, 3rd Ed.
Some video shots used from YouTube channel “AlanBeckerTutorials”
Other images sourced from Google images

Today

- Final bits of ray tracing
- Computer Animation

Showtime



Logistical Things

- How is Assignment 3 going ?

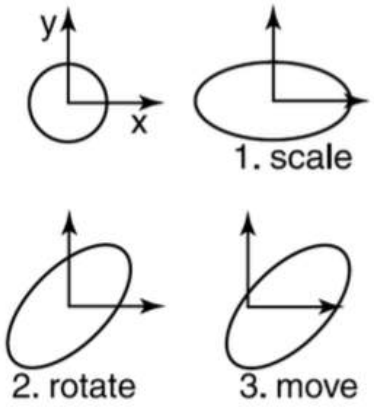
Topic 13:

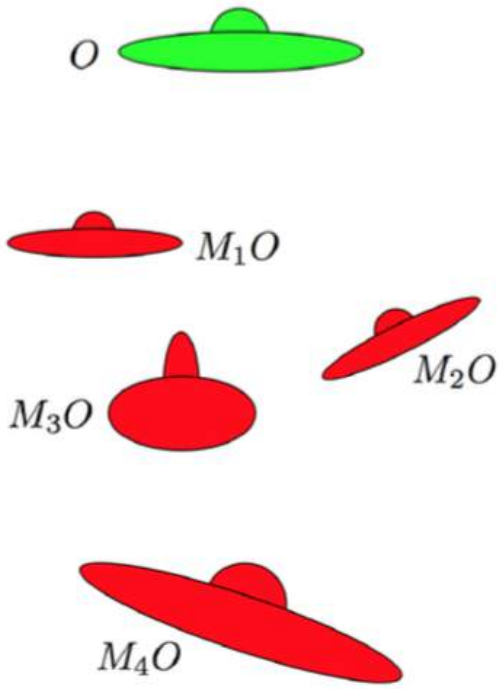
Instancing

Copying and transforming objects

Instancing is an elegant technique to place various transformed copies of an object in a scene.

Expl.: circle → elipse

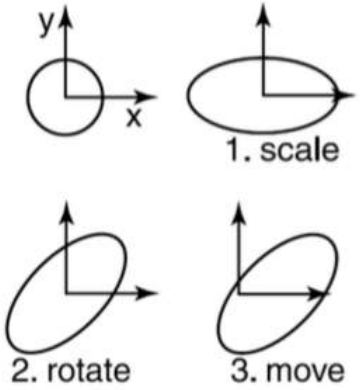


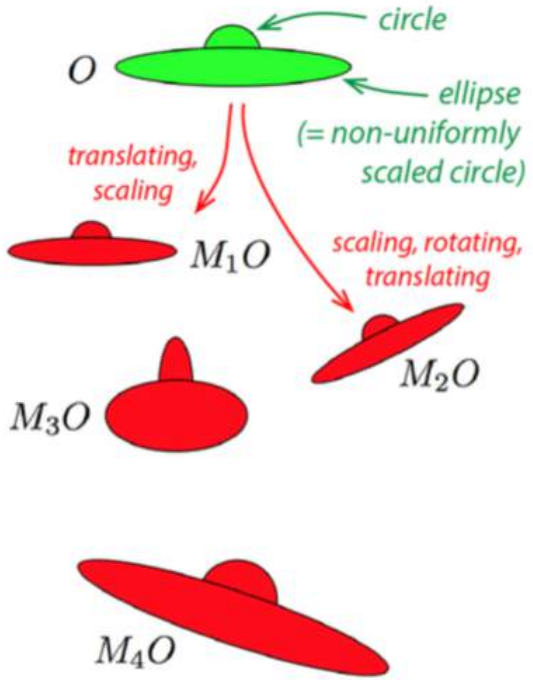


Copying and transforming objects

Instancing is an elegant technique to place various transformed copies of an object in a scene.

Expl.: circle → elipse



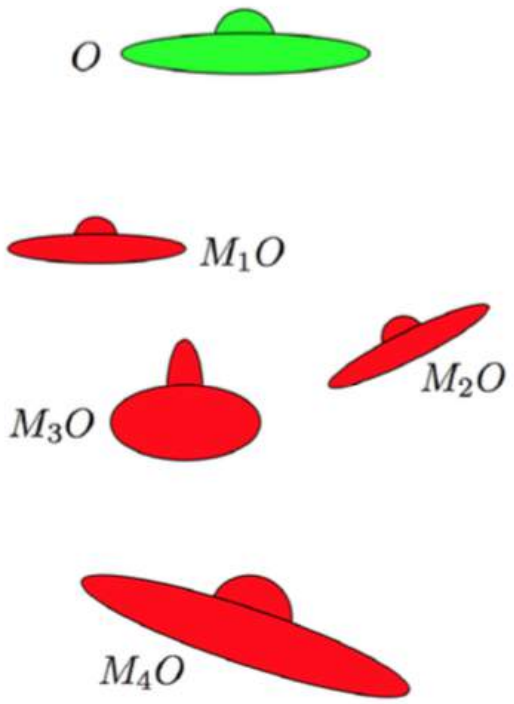


Copying and transforming objects

Instead of making actual copies, we simply store a reference to a base object, together with a transformation matrix.

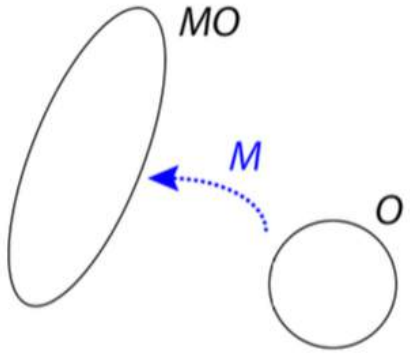
That can save us lots of storage.

Hmm, but how do we compute the intersection of a ray with a randomly rotated elipse?



Ray-instance intersection

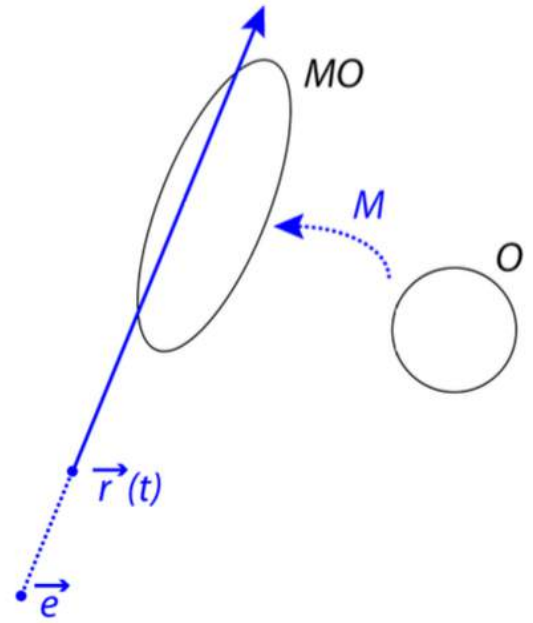
Assume an object O that is used to create an object MO via instancing.



Ray-instance intersection

Now, we want to create the intersection of MO with the ray $\vec{r}(t)$, which in turn is defined by the line

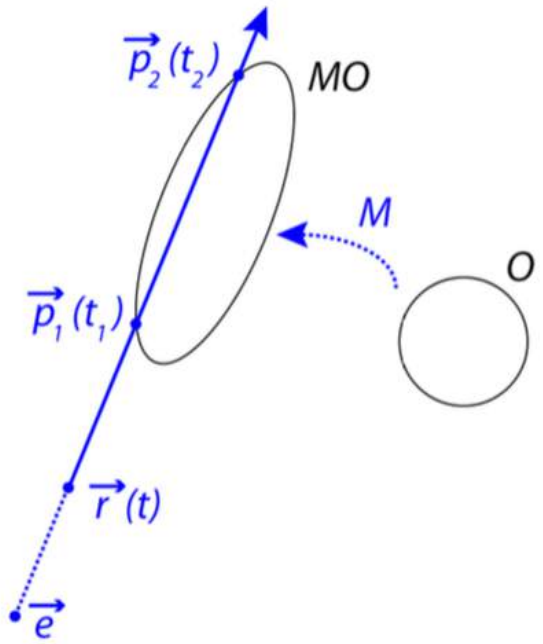
$$\vec{l}(t) = \vec{e} + t\vec{d}.$$



$\vec{l}(t) = \vec{e} + t\,\vec{d}$

Ray-instance intersection

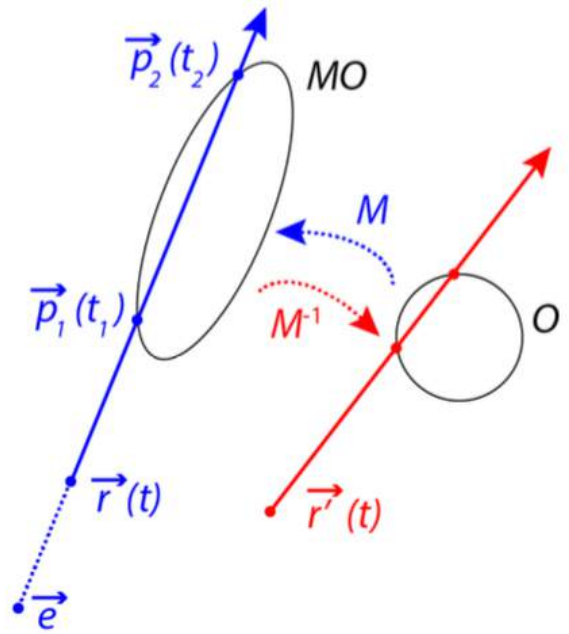
Fortunately, such
complicated intersection tests
(e.g. ray/ellipsoid) can often be
replaced by much simpler tests
(e.g. ray/sphere).



$$\vec{l}(t) = \vec{e} + t\,\vec{d}$$

Ray-instance intersection

To determine the intersections $\vec{p_i}$ of a ray $\vec{r}$ with the instance MO, we first compute the intersections $\vec{p'_i}$ of the inverse transformed ray $M^{-1}\vec{r}$ and the original object O.



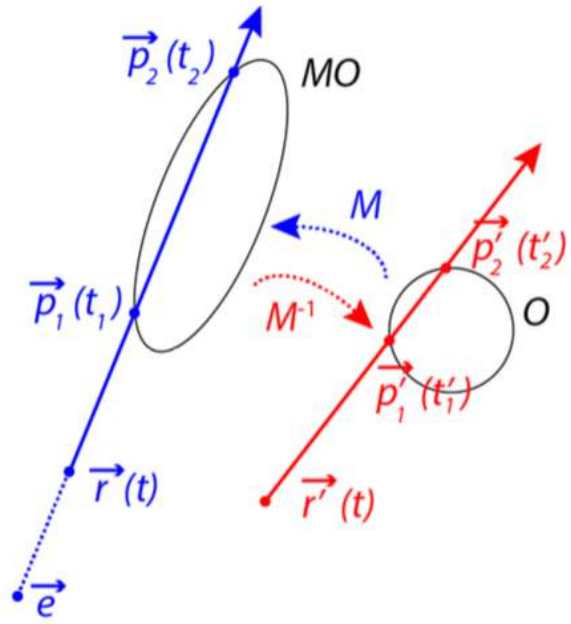
$$\vec{l}(t) = \vec{e} + t\,\vec{d}$$

Ray-instance intersection

The points $\vec{p_i}$ are then simply

$$M\vec{p_i'} \quad \text{or} \quad \vec{l}(t_i')$$

because the linear transformation preserves relative distances along the line.

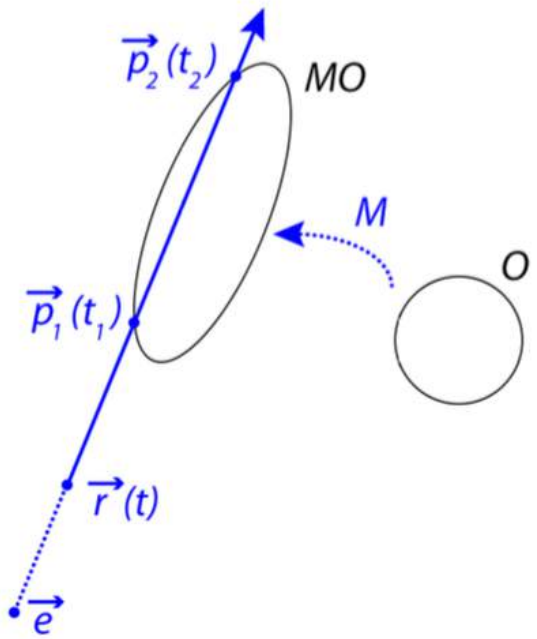


$$\vec{l}(t) = \vec{e} + t\,\vec{d}$$

Ray-instance intersection

Two pitfalls:

- The direction vector of the ray should *not* be normalized
- Surface normals transform differently!
  $\rightarrow$ use $(M^{-1})^T$ instead of M for normals



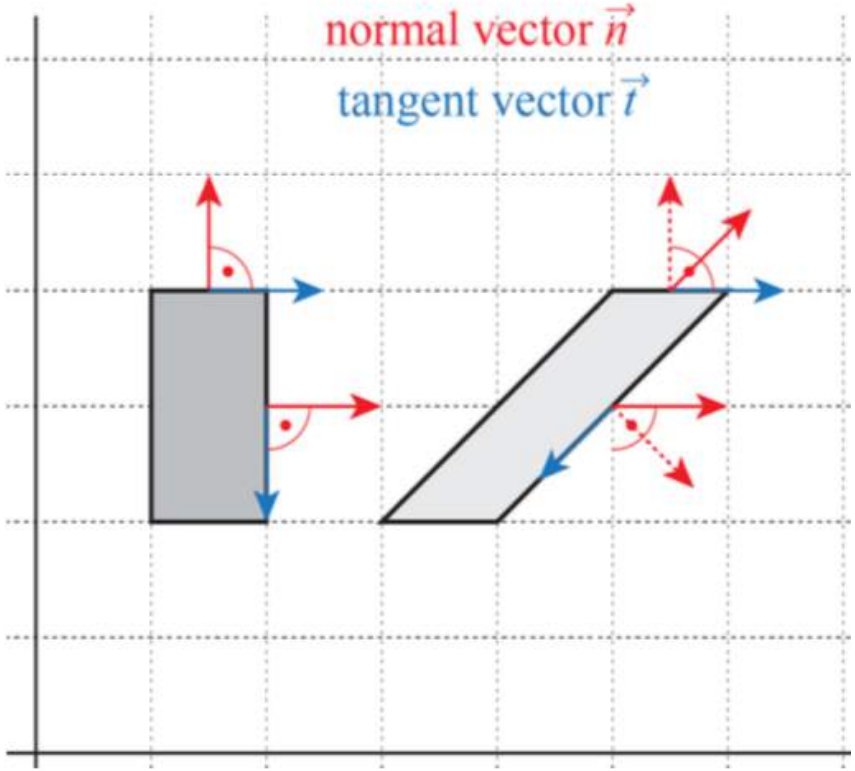
$$\vec{l}(t) = \vec{e} + t\,\vec{d}$$

Transforming normal vectors

Unfortunately, normal vectors are not always transformed properly.

E.g. look at shearing, where tangent vectors are correctly transformed but normal vectors not.

To transform a normal vector $\vec{n}$ correctly under a given linear transformation A, we have to apply the matrix $(A^{-1})^T$. Why?



Transforming normal vectors

We know that tangent vectors are tranformed correctly: $A\vec{t} = \vec{t_A}$.

But this is not necessarily true for normal vectors: $A\vec{n} \neq \vec{n_A}$.

Goal: find matrix N_A that transforms $\vec{n}$ correctly, i.e. $N_A\vec{n} = \vec{n_N}$ where $\vec{n_N}$ is the correct normal vector of the transformed surface.

Because our original normal vector $\vec{n}^T$ is perpendicular to the original tangent vector $\vec{t}$, we know that:

$$\vec{n}^T\vec{t} = 0.$$

This is the same as

$$\vec{n}^T I\vec{t} = 0$$

which is is the same as

$$\vec{n}^T A^{-1}A\vec{t} = 0$$

Transforming normal vectors

Because $A\vec{t} = \vec{t_A}$ is our correctly transformed tangent vector,
we have

$$\vec{n}^T A^{-1} \vec{t_A} = 0$$

Because their scaler product is 0, $\vec{n}^T A^{-1}$ must be orthogonal to it.
So, the vector we are looking for must be

$$\vec{n}_N^T = \vec{n}^T A^{-1}.$$

Because of how matrix multiplication is defined,
this is a transposed vector. But we can rewrite this to

$$\vec{n}_N = (\vec{n}^T A^{-1})^T.$$

And if you remember that $(AB)^T = B^T A^T$, we get

$$\vec{n}_N = (A^{-1})^T \vec{n}$$

Topic 14:

Animation

Animation Timeline

- 1908: Emile Cohl (1857-1938) France, makes his first film, FANTASMAGORIE, arguably the first animated film (running time ~1min 20sec)

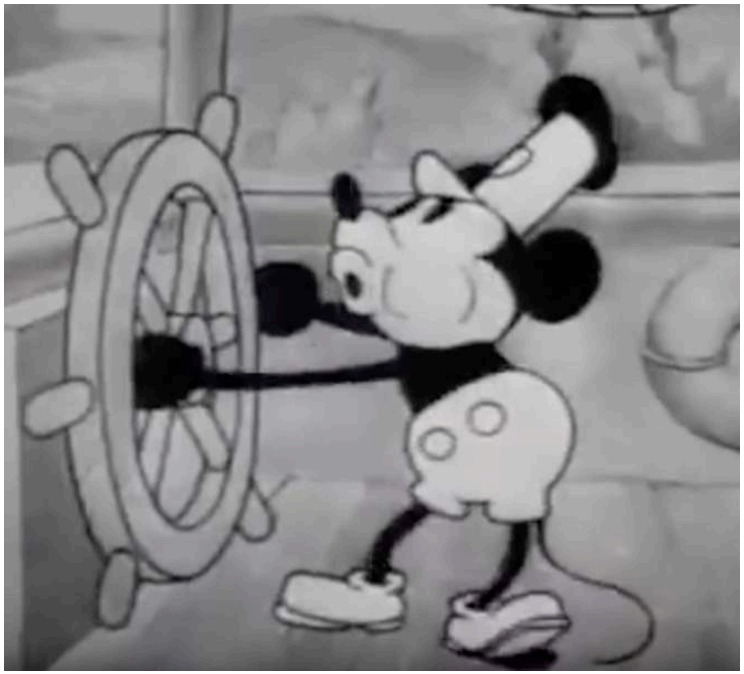
Animation Timeline

- 1911: Winsor McCay (1867-1934) makes his first film, LITTLE NEMO. McCay, already famous for comic strips, used the film in his vaudeville act. Pioneered keyframe animation, open about it and refused to patent his technique saying:

Any idiot that wants to make a couple of thousand drawings for a hundred feet of film is welcome to join the club.

Animation Timeline

- 1928: Walter Disney (1901-1966) working at the Kansas City Slide Company creates Mickey Mouse.



Animation Timeline

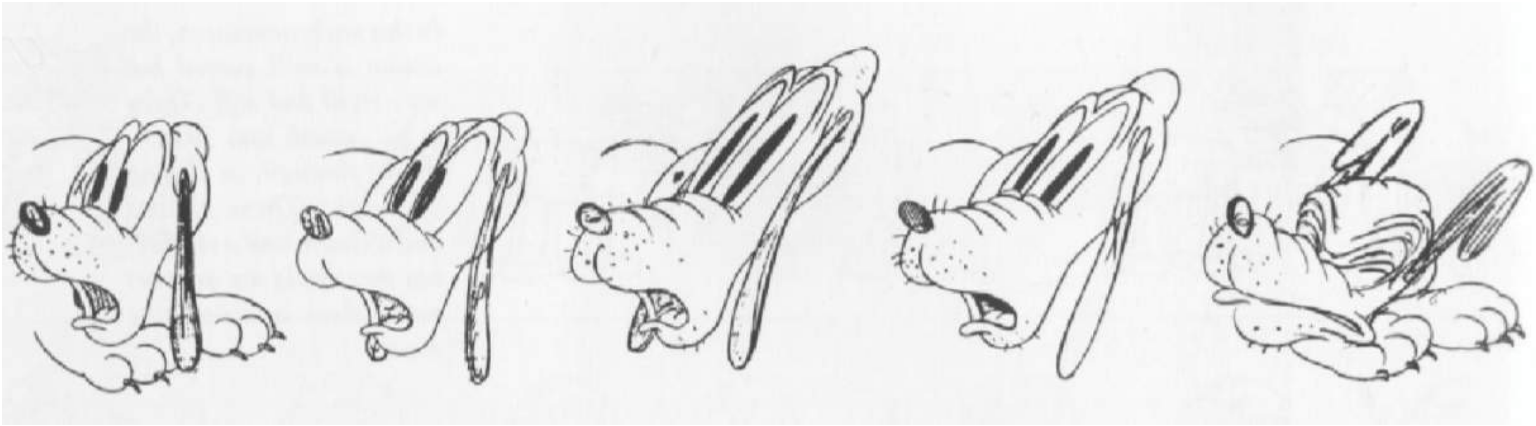
- 1974: First Computer animated film “Faim” from NFB nominated for an Oscar.

Animation Principles

- 12 basic principles of animation
- Deals with emotional timing and laws of physics
- Disney book “The Illusion of Life: Disney Animation”

Squash and Stretch

- Rigid objects look robotic: ***deformations*** make motion natural
- Accounts for physics of deformation
  - Think squishy ball...
  - Communicates to viewer object's composition, its weight,…
  - Usually large deformations conserve volume:
    Squash in one dimension, stretch in another to keep mass constant
- Also accounts for persistence of vision
  - Fast moving objects leave an elongated streak on our retinas



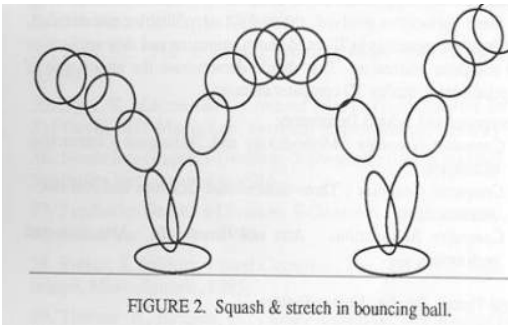
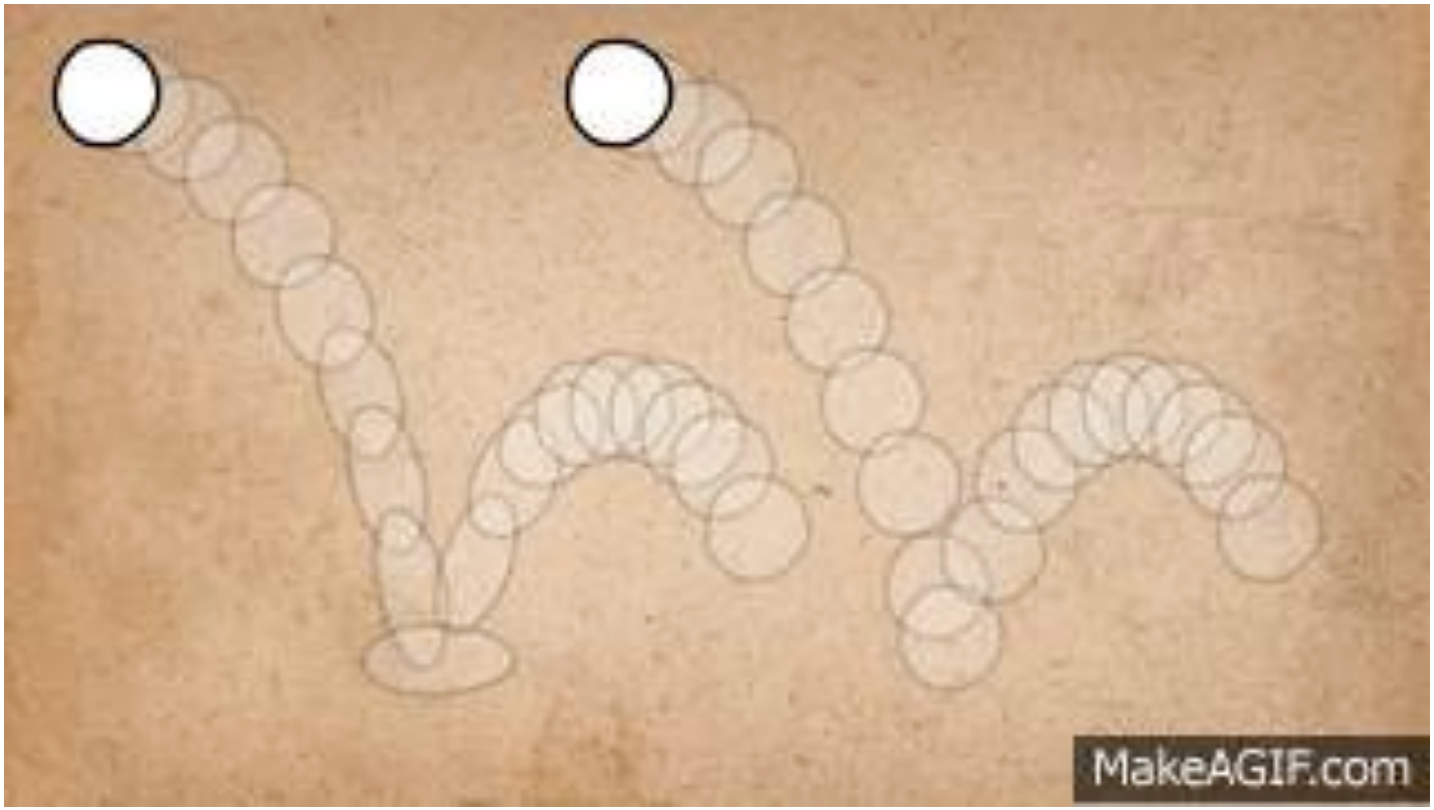
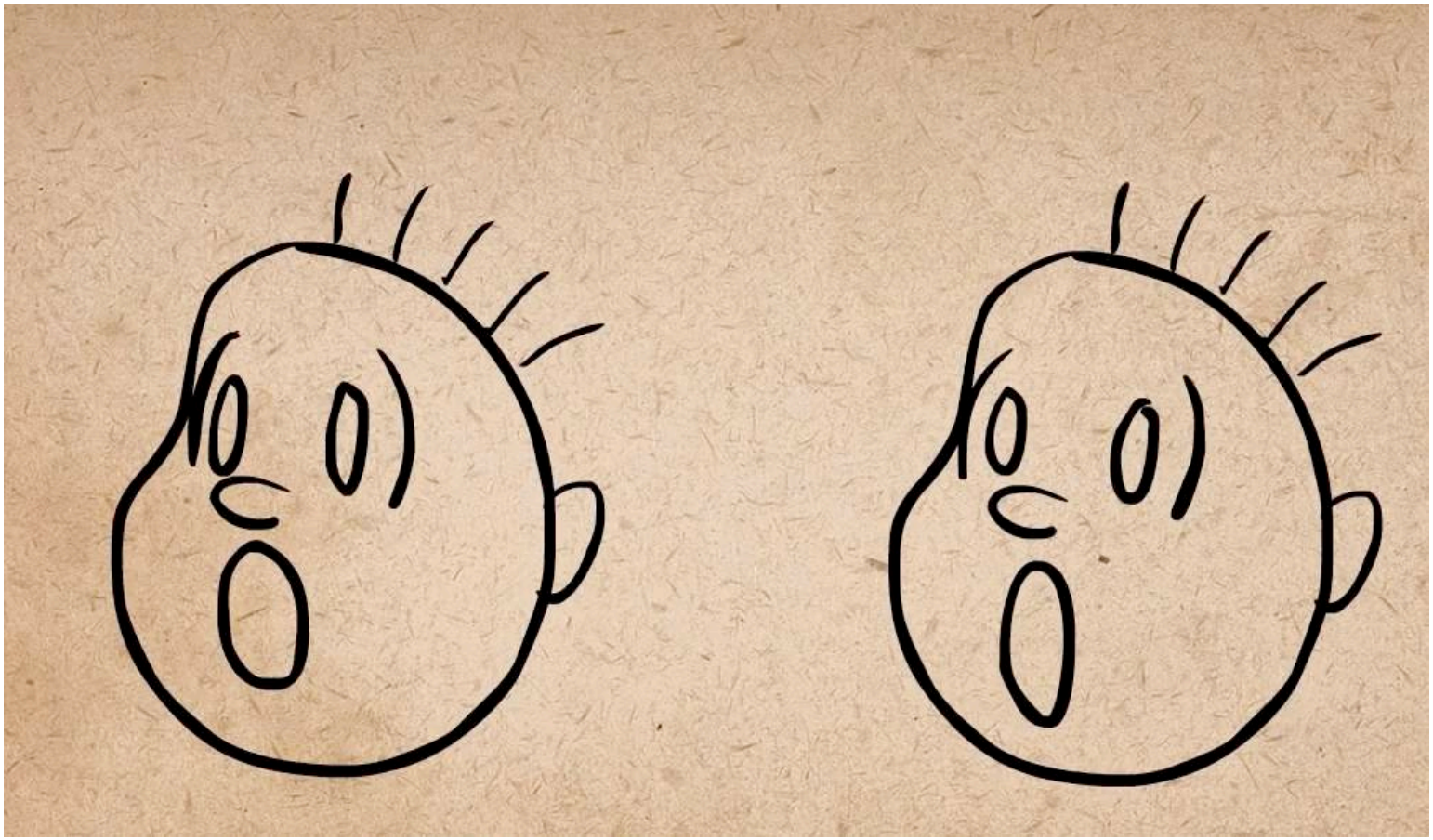


FIGURE 2. Squash & stretch in bouncing ball.

Squash and Stretch

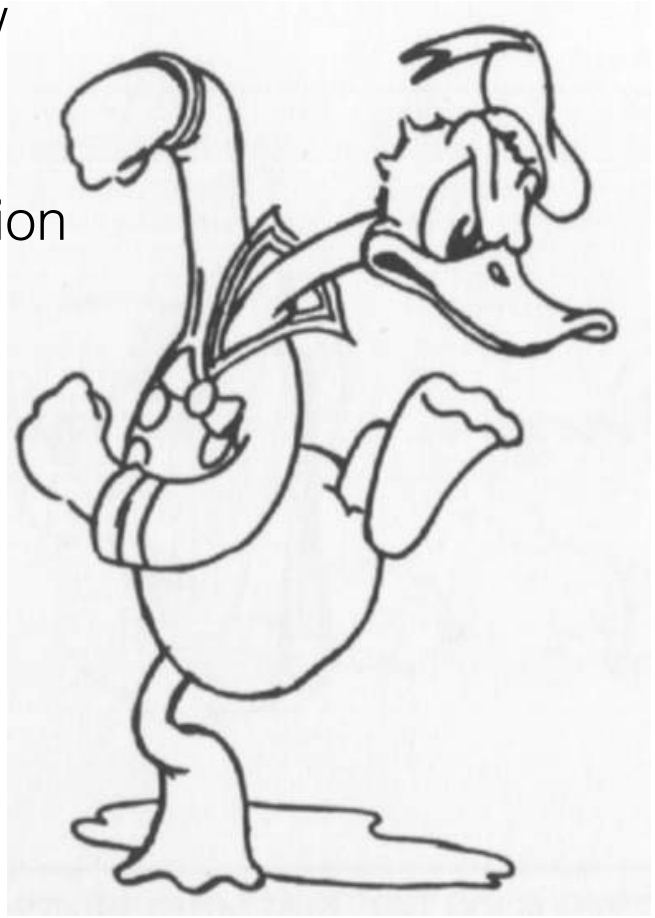


Squash and Stretch

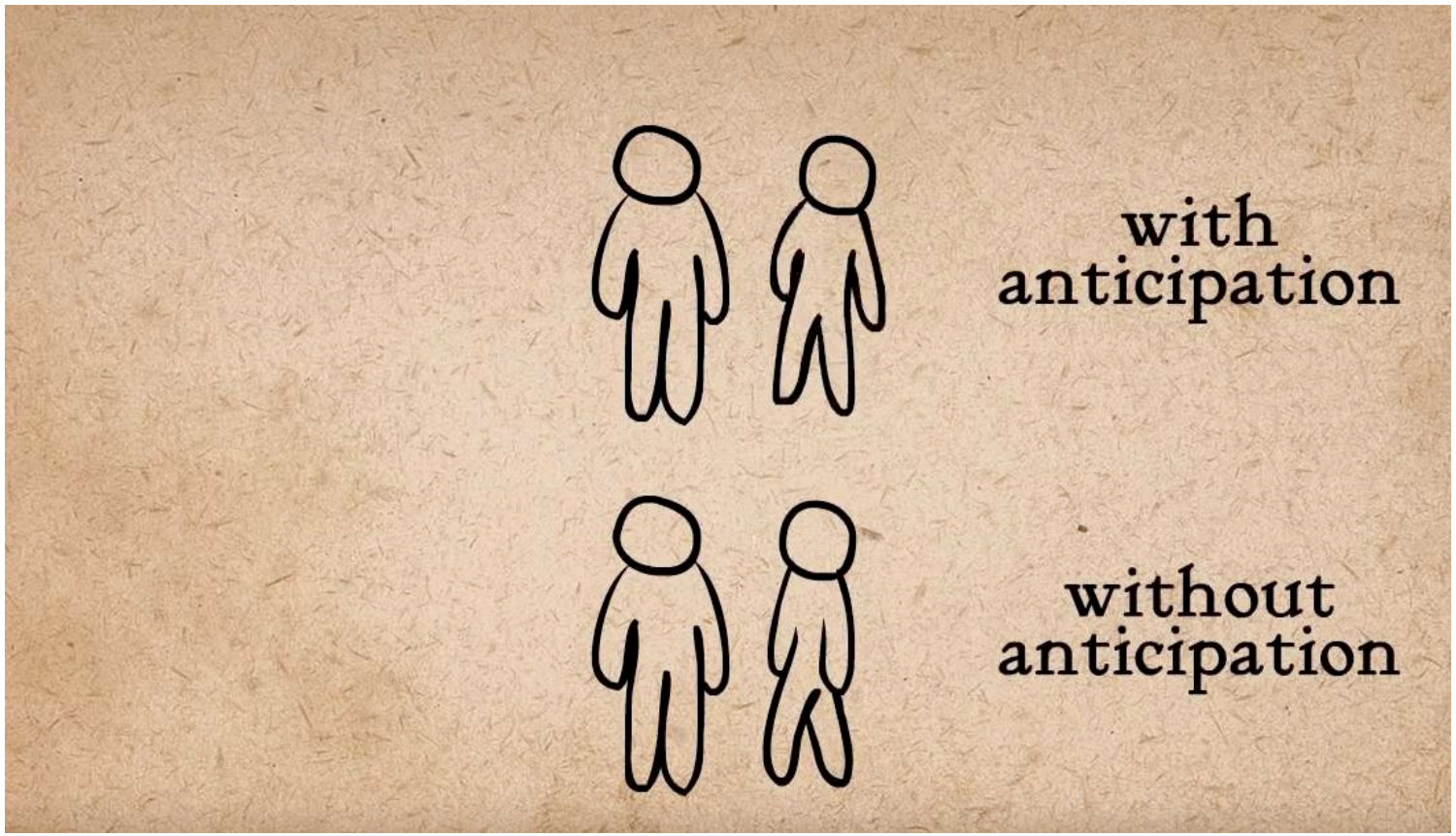


Anticipation

- The preparation before a motion
  - e.g. crouching before jumping, pitcher winding up to throw a ball
- Often physically necessary, and indicates how much effort a character is making
- Also essential for controlling the audience's attention, to make sure they don't miss the action
  - Signals something is about to happen, and where it is going to happen.



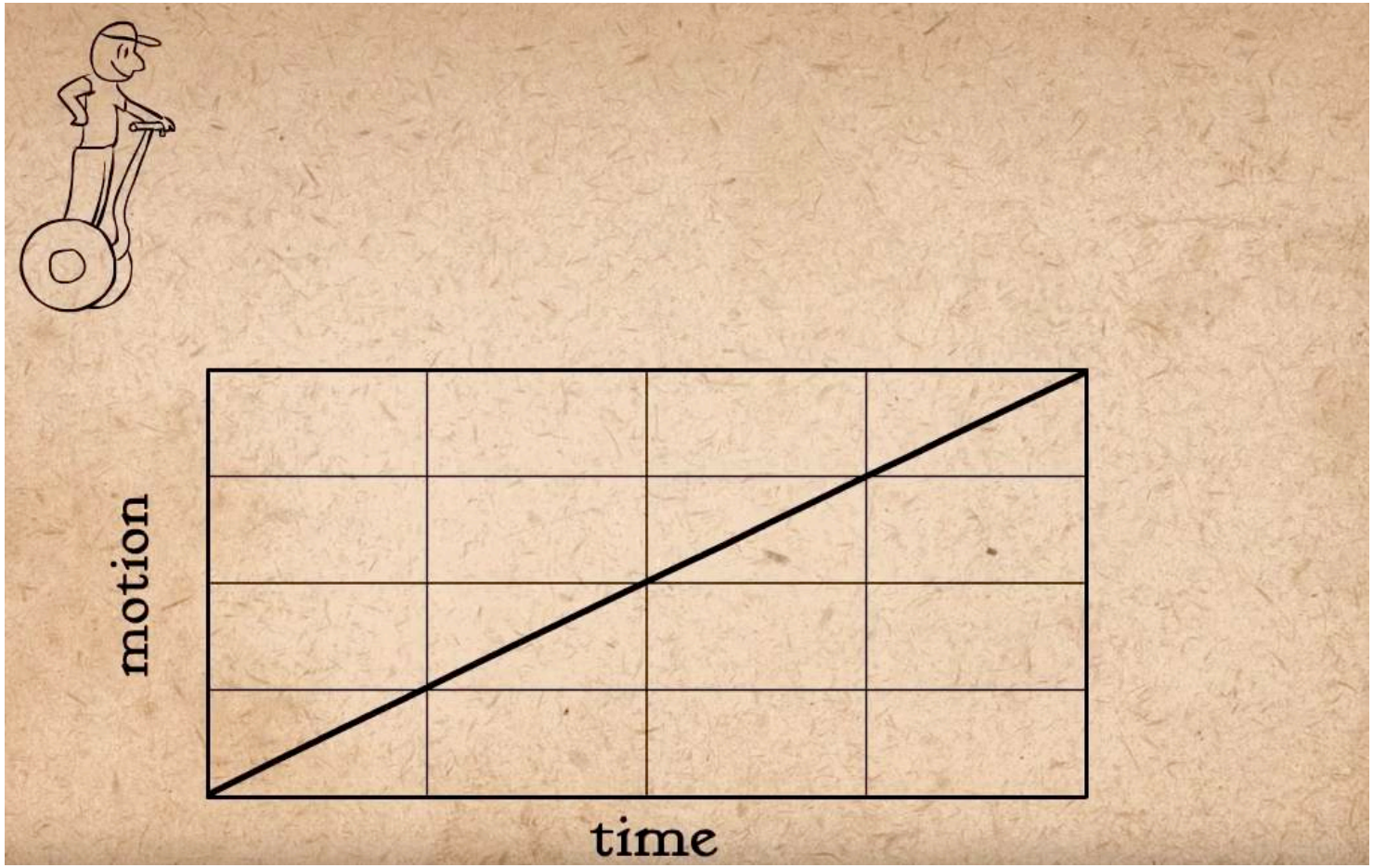
Anticipation



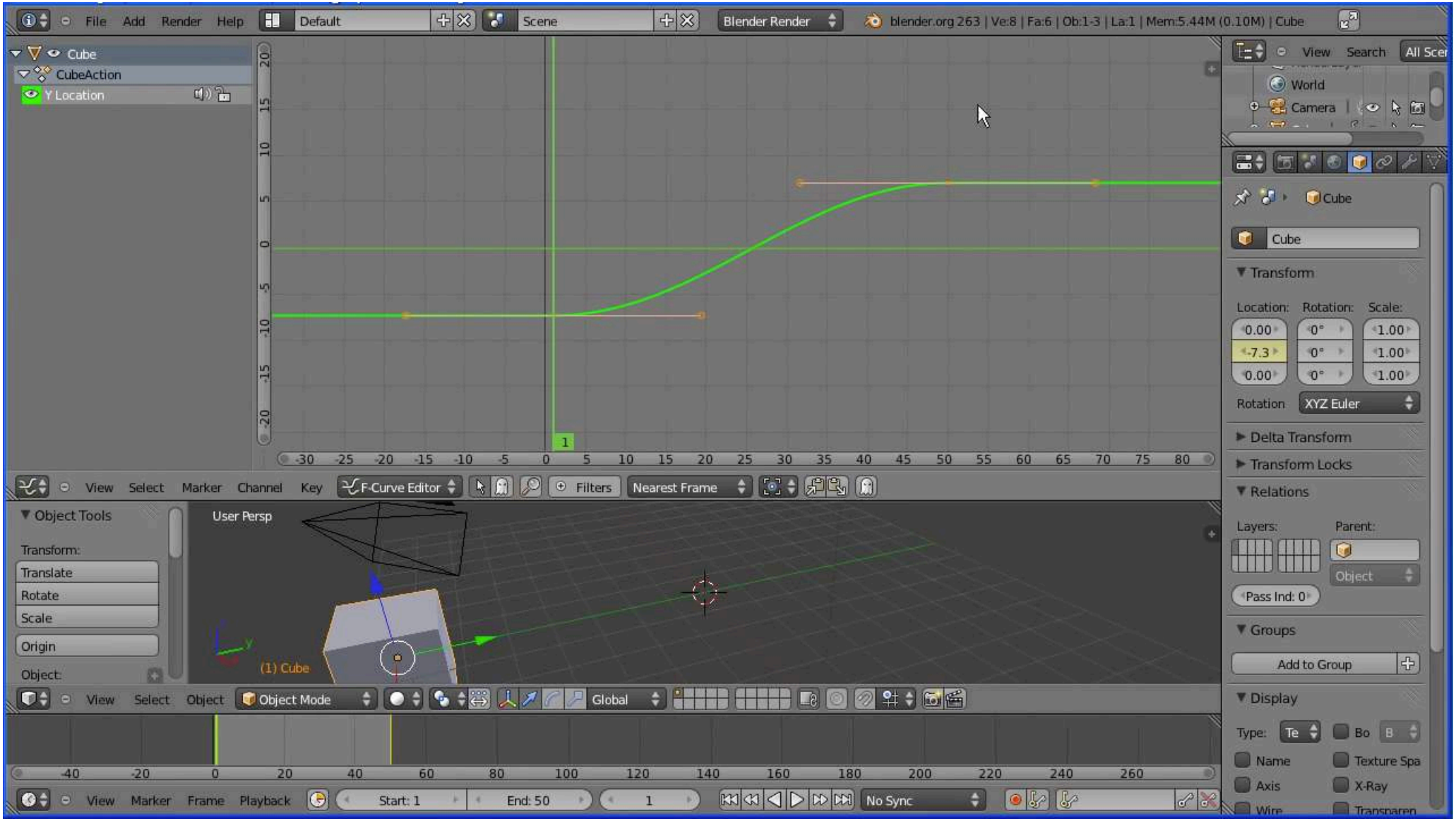
Ease-In & Ease-Out

- Objects don’t have immediate speed and don’t stop immediately
- Objects accelerate and decelerate, even if quickly
- Add more drawings near the beginning and end
- Emphasize extreme poses
- Can be controlled with spline interpolation

Ease-In & Ease-Out



Examples


File Add Render Help
Default
Scene
Blender Render
blender.org 263 | Ve:8 | Fa:6 | Ob:1-3 | La:1 | Mem:5.44M (0.10M) | Cube
Cube
CubeAction
Y Location
View Search
World
Camera
Cube
Transform
Location: Rotation: Scale:
0.00 0° 1.00
-7.3 0° 1.00
0.00 0° 1.00
Rotation XYZ Euler
Delta Transform
Transform Locks
Relations
Layers: Parent:
Object
Pass Ind: 0
Groups
Add to Group
Display
Type: Te
Name
Axis
Wire
Texture Spa
X-Ray
View Select Marker Channel Key
F-Curve Editor
Filters
Nearest Frame
Object Tools
Transform:
Translate
Rotate
Scale
Origin
Object:
User Persp
(1) Cube
View Select Object
Object Mode
Global
View Marker Frame Playback
Start: 1
End: 50
1
No Sync

Staging

- Similar concept in film and theatre
- Direct attention to most important point
- Remove ambiguity

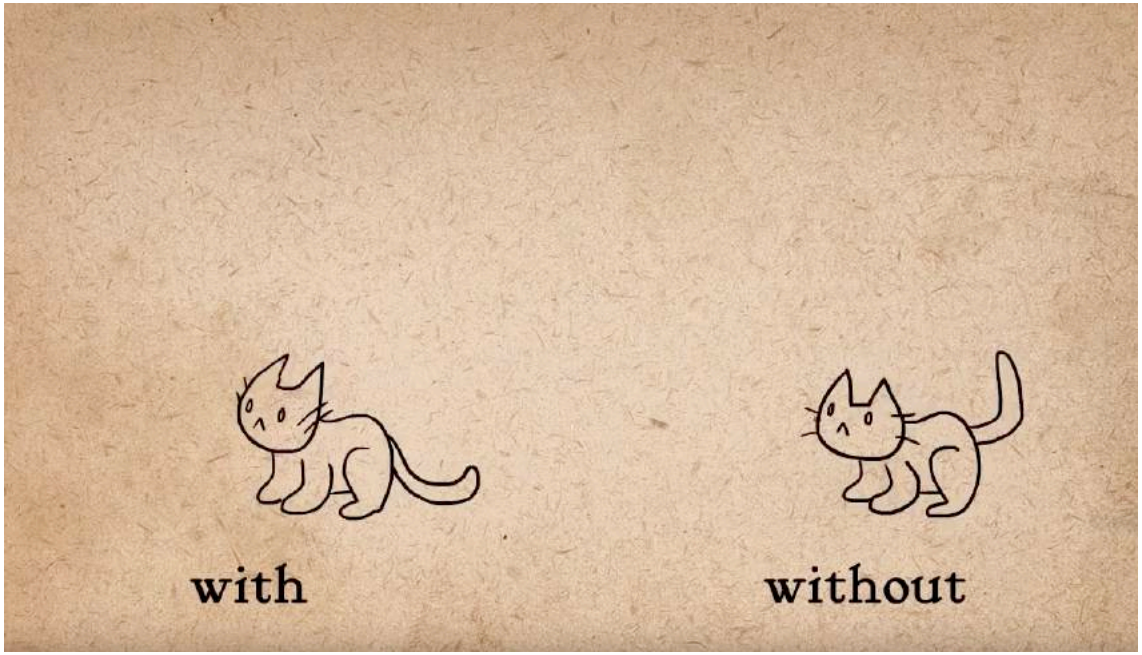
Straight Ahead and Pose to Pose

- Straight ahead is to draw the scene frame by frame from beginning to end in a linear fashion
- Pose to pose draws important key moments (key frames) and fills in between the frames (inbetweening), to give the illusion of motion.

  (more later)

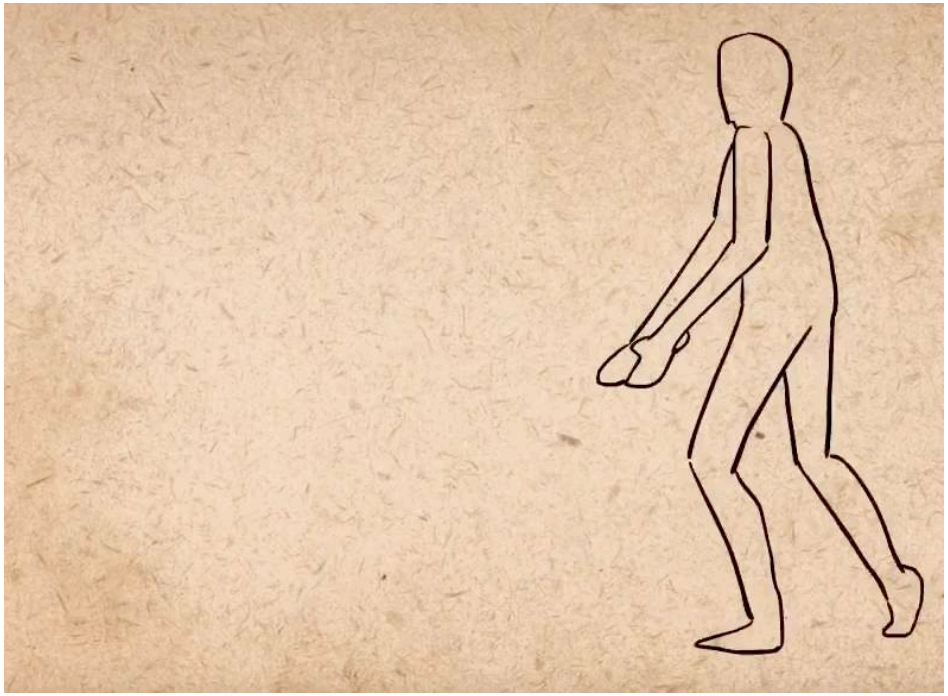
Follow-through and Overlapping Action

- Uses the principle of inertia and laws of physics
- Follow-through: adds realism to animation through continued motion of loosely held parts on the body (hair, tail, clothes, fatty tissue, etc…)
- Overlapping action: different parts of body move at different rates



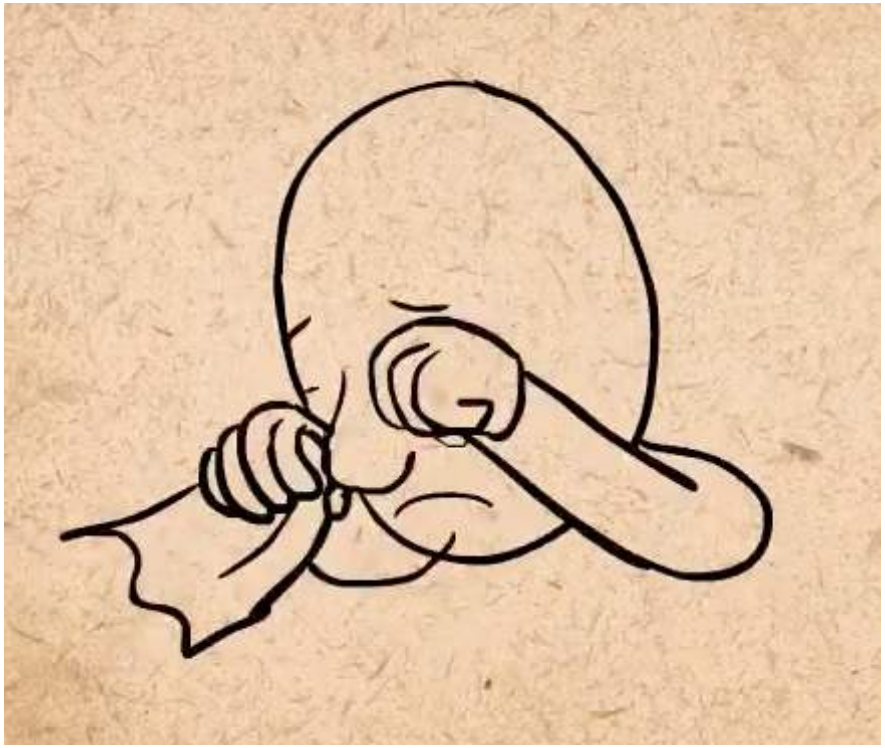
Arcs

- Things naturally move in arcs (baseball pitchers arms motion, swinging of a sword, etc…)
- Balls and objects follow a parabolic trajectory
- Follow the arc when inbetweening



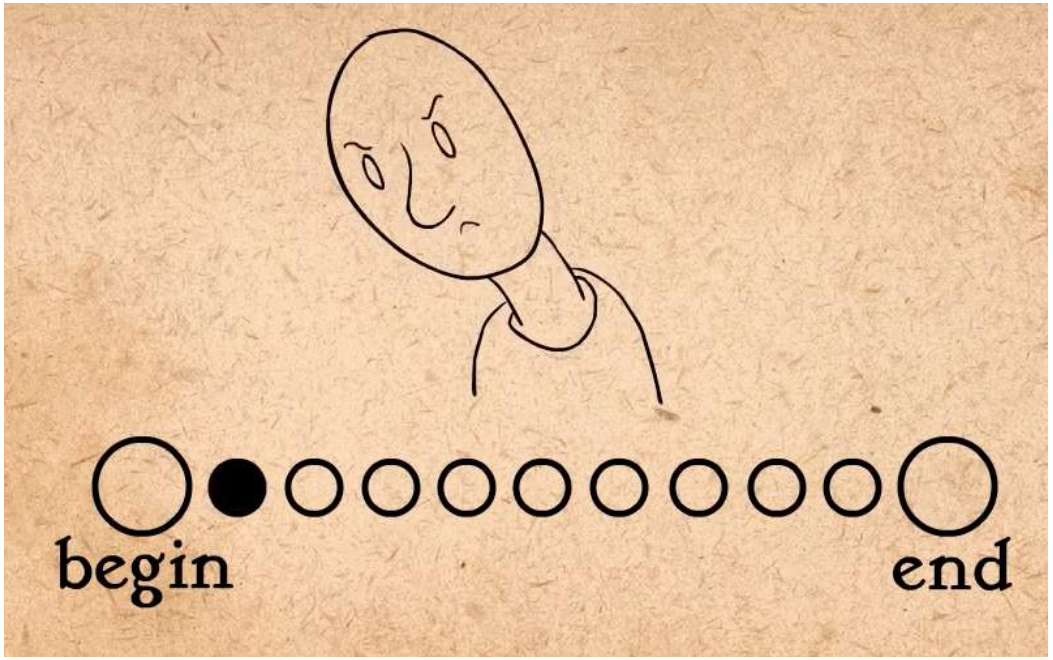
Secondary Action

- Coupling a primary action with natural secondary actions
- e.g. Person walking swinging their arms, wiping a tear while crying, facial expression while eating



Timing

- The number of drawings for a given action.
- More drawings equates to slower motion
- Less drawings equates to faster motion
- Can convey significantly different messages

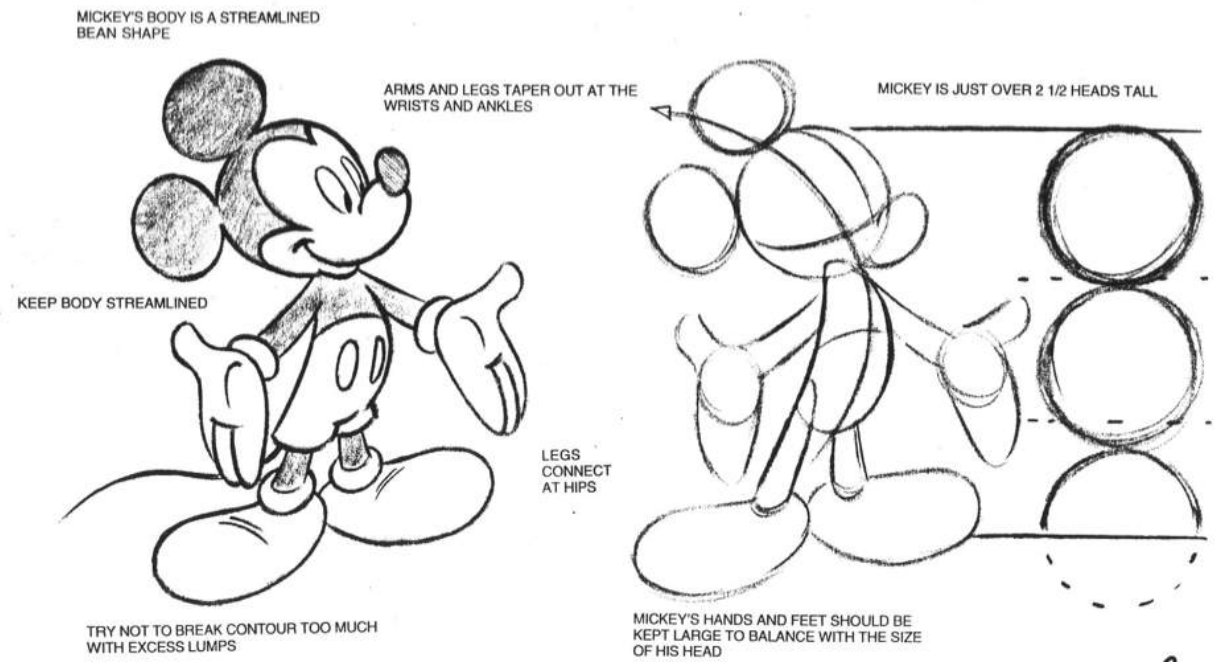


Exaggeration

- Can accentuate certain features
- Adds a degree of style
- Things that are “perfectly” real can come off as dull

Solid Drawing

- Able to draw things that are 3D
- Follow the contours of an object (e.g. sphere)
- Draw in perspective toward vanishing points
- Use solid shapes like cubes and circles to give dimension



Appeal

- Make the character real and interesting
- Pleasing to look at
- Charismatic aspect
- Interesting to look at, not necessarily “good-looking”
- Avoid symmetry

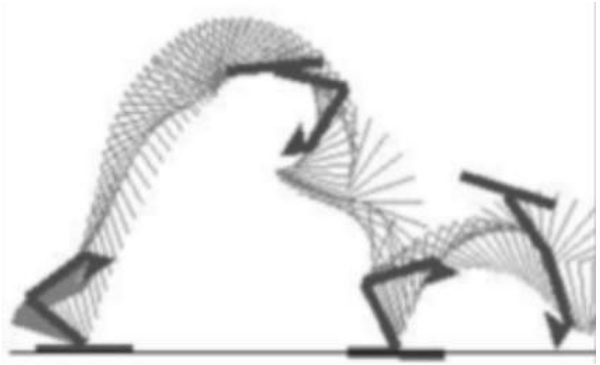
Animation Principles

- **Squash and Stretch**
- **Anticipation**
- **Ease-In & Ease-Out**
- Staging
- Straight Ahead and Pose to Pose
- Follow-through and Overlapping Action
- Arcs
- Secondary Action
- Timing
- Exaggeration
- Solid Drawing
- Appeal

Elements of CG (animation)

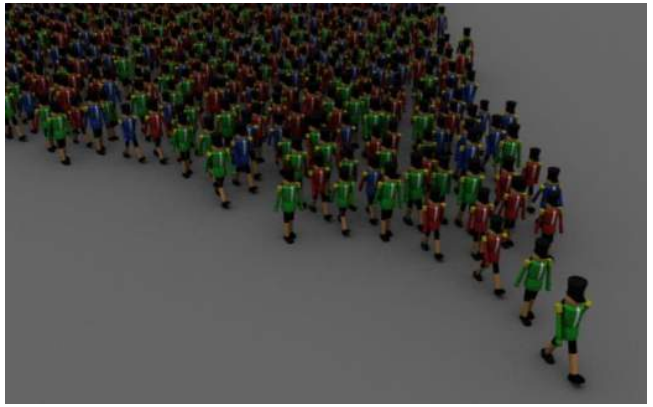
- How does one make digital models move?
  - Keyframing

  - Physical simulation



  - Motion capture

  - Behaviour rules



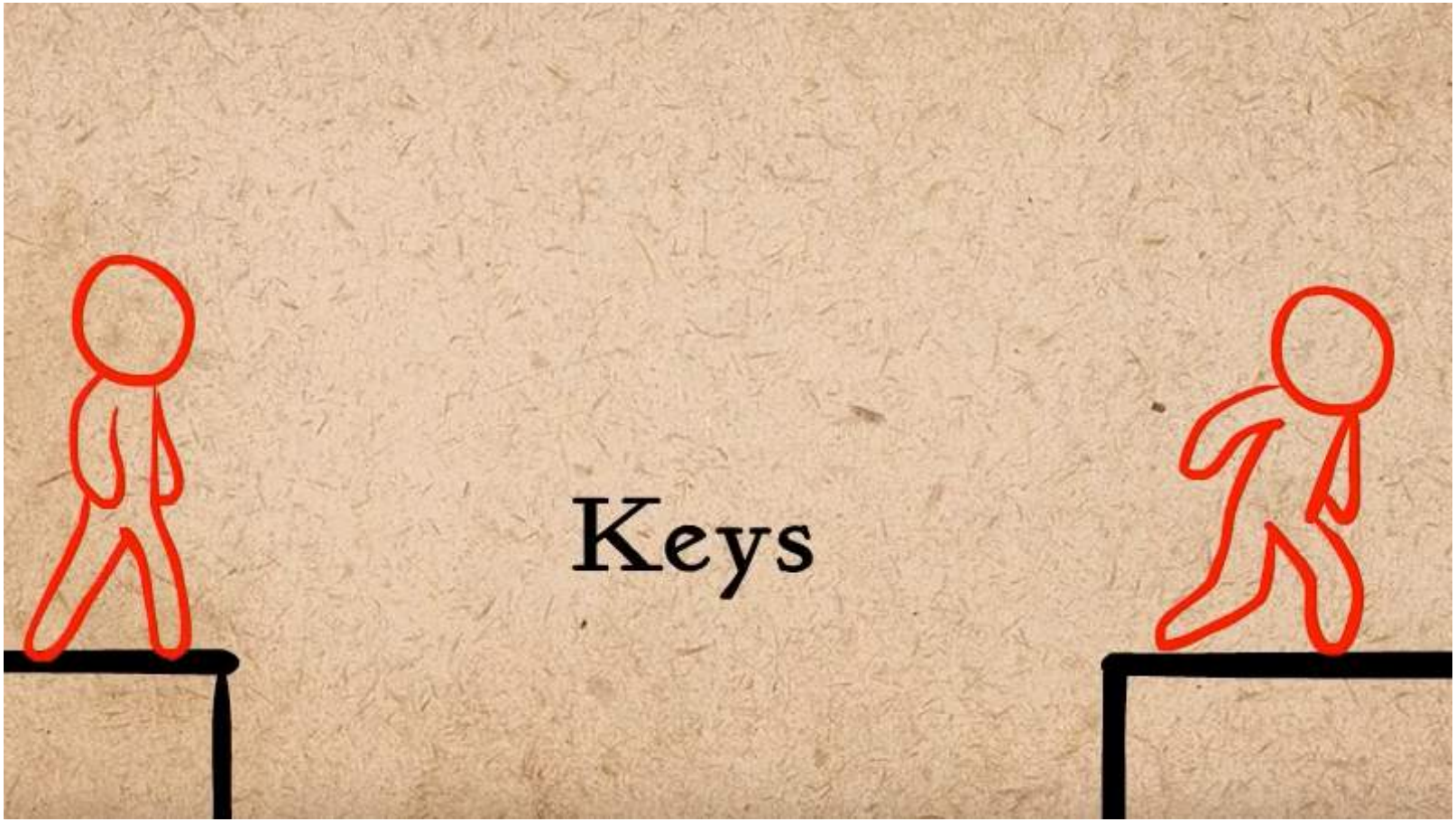
Keyframes

Keyframes, also called extremes, define important poses of a character:
Jump example:

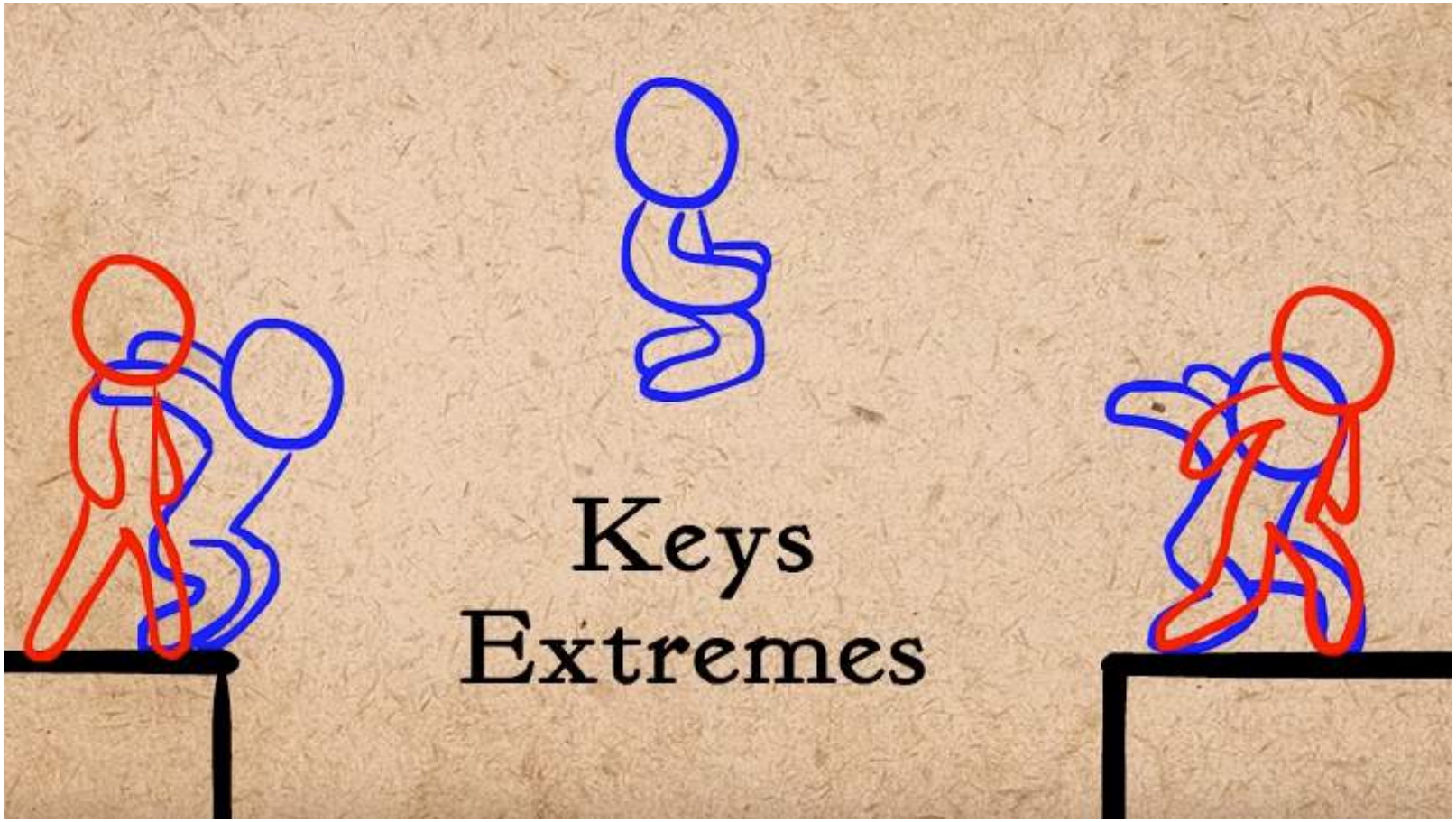
- the start
- the lowest crouch
- the lift-off
- the highest part
- the touch-down
- the lowest follow-through

- Frames in between (“inbetweens”) introduce nothing new to the motion.
- May add additional keyframes to add some interest, better control the interpolated motion.

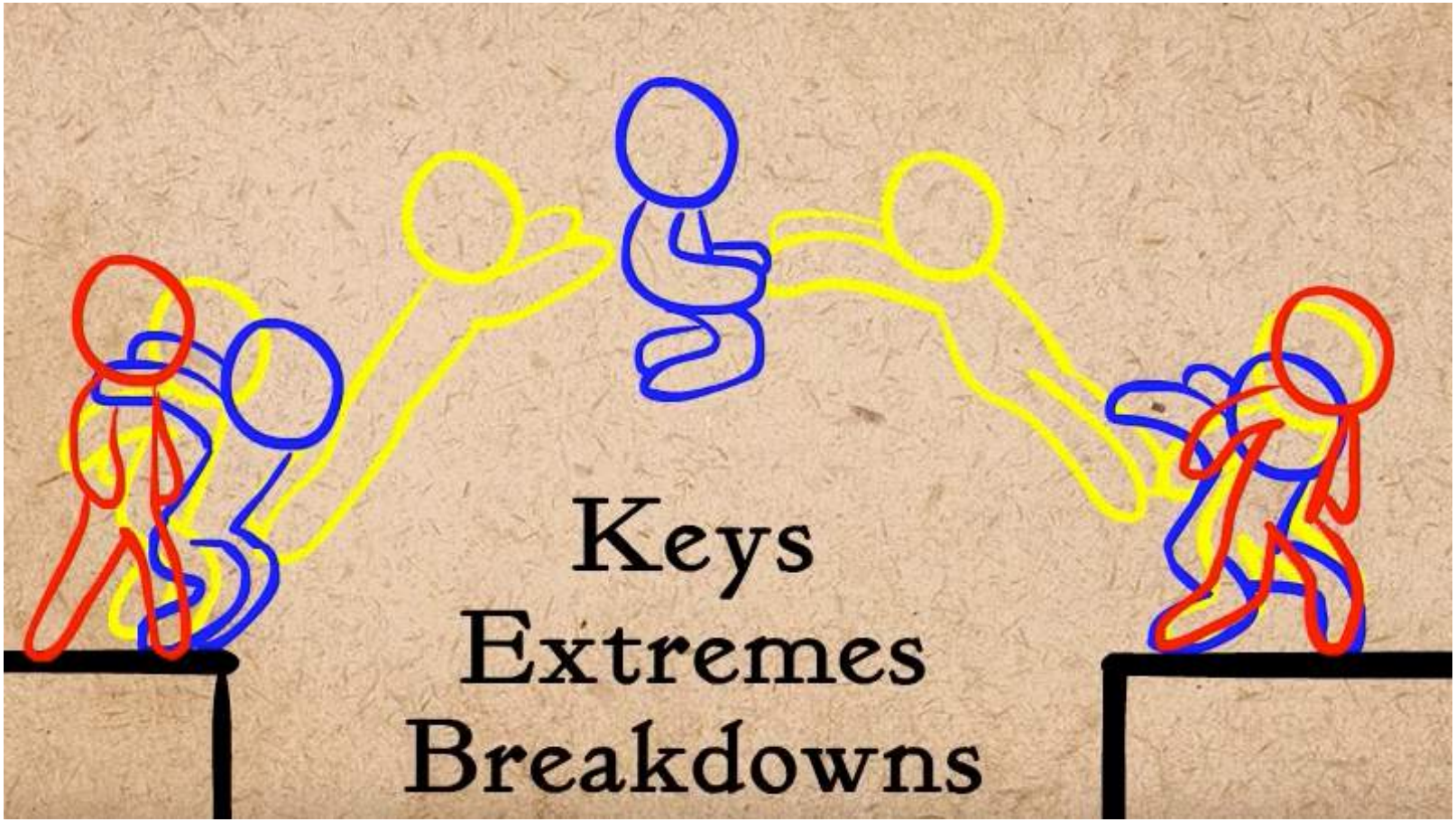
Keyframes



Keyframes



Keyframes



Keyframe Animation

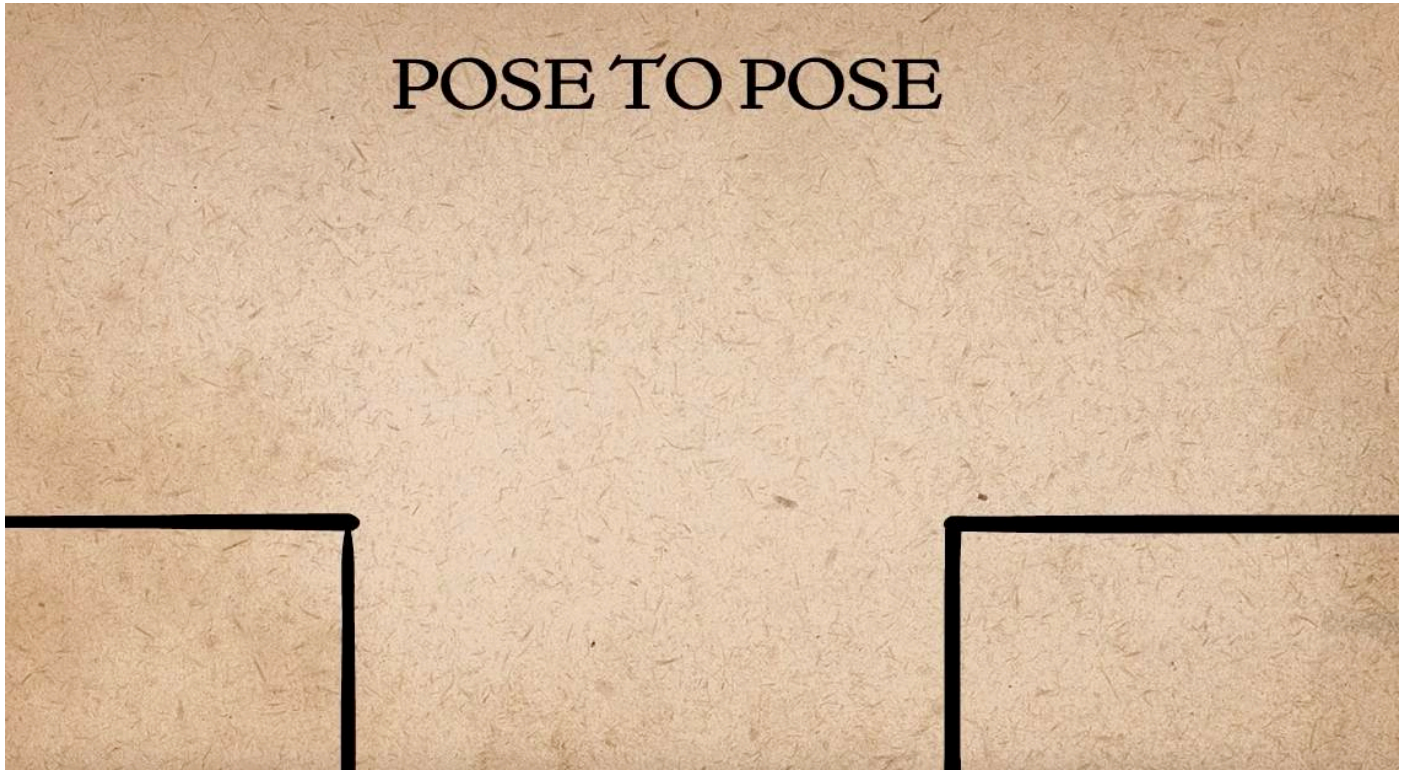
The task boils down to setting animated variables (e.g. positions, angles, sizes, ...) at each frame.

Straight-ahead: set variables in frame 0, then frame 1, frame 2, ... forward in time.

Pose-to-pose: set the variables at keyframes, let the computer smoothly interpolate values for frames in between.

Keyframe Animation

Pose to pose



Keyframe Animation

Straight ahead

Keyframe Animation

Pros:

- Very expressive
- Animator has full control of animation

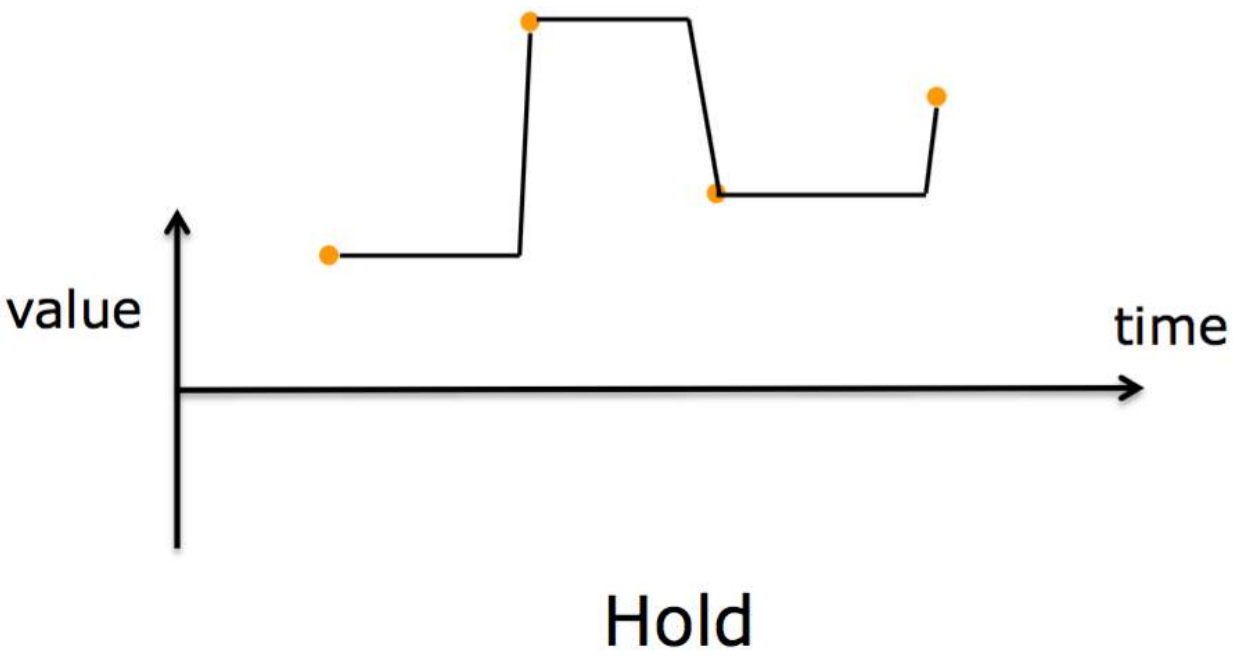
Cons

- Very labour intensive
- Difficult to create convincing physical realism (if that is a goal)

Used for practically anything except complex physical simulations (smoke, water, etc)

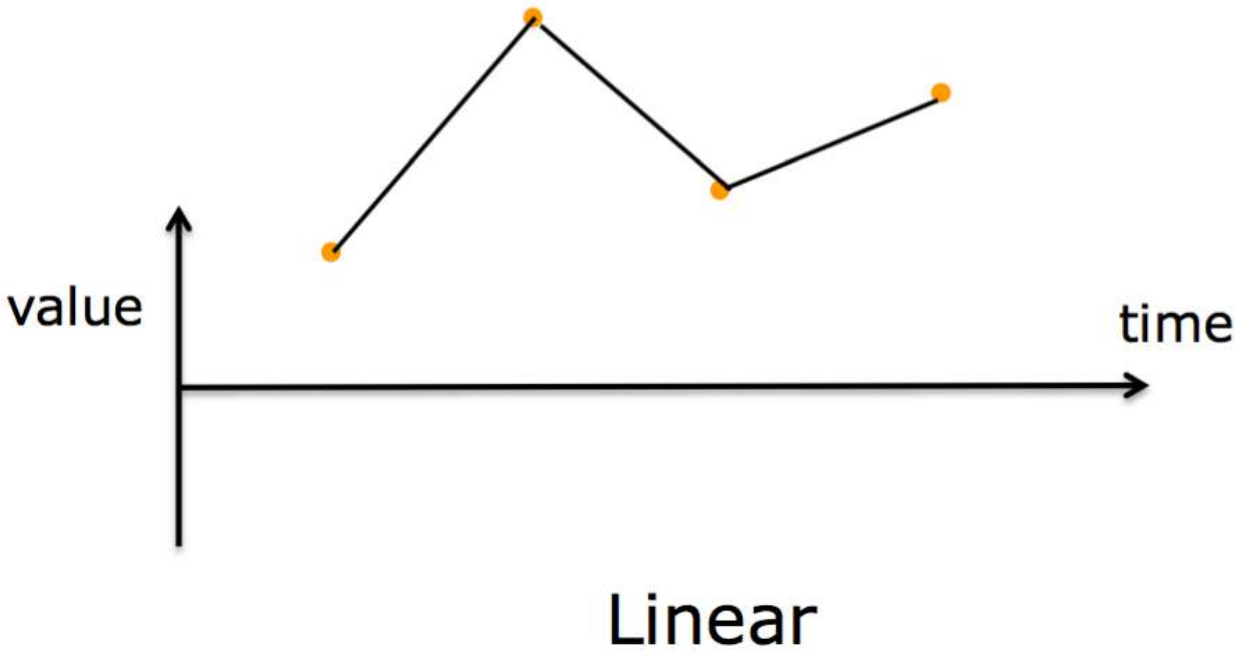
Interpolation

How do we interpolate between two values?



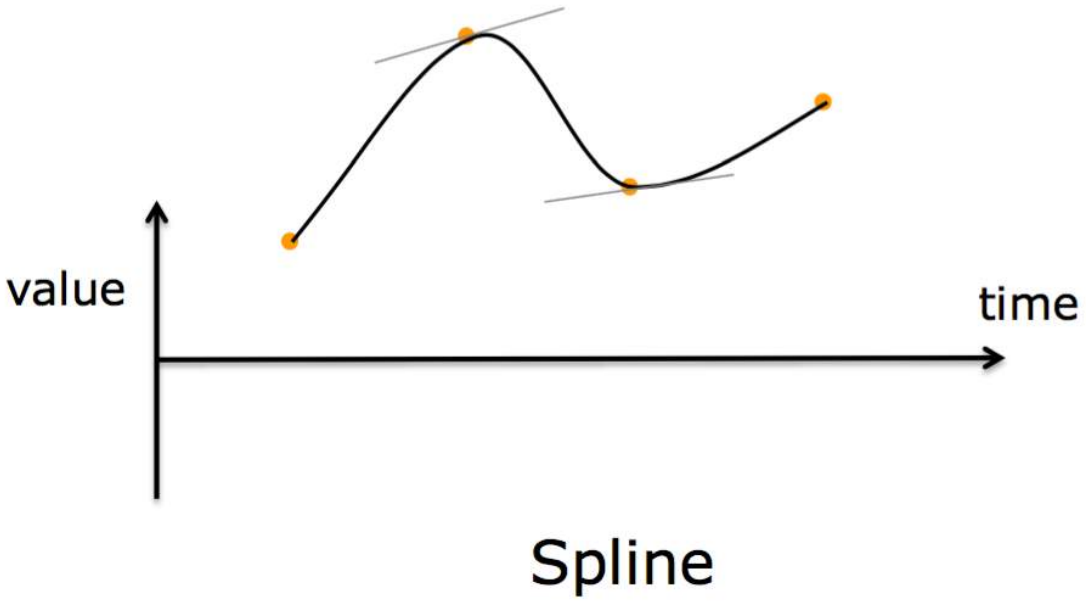
Interpolation

How do we interpolate between two values?



Interpolation

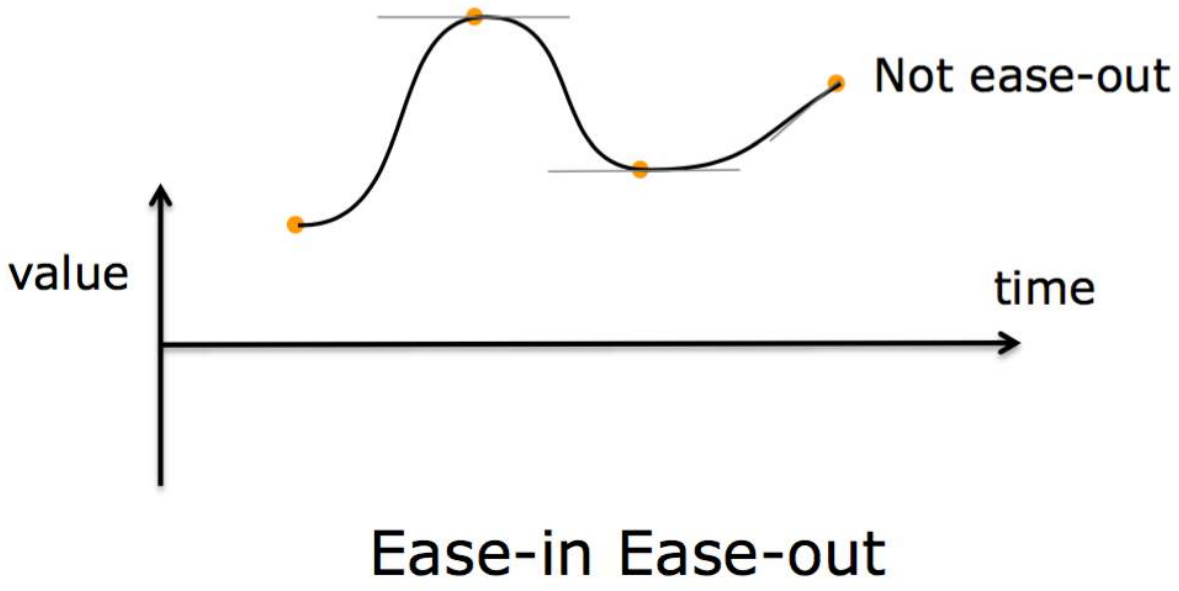
How do we interpolate between two values?



Spline

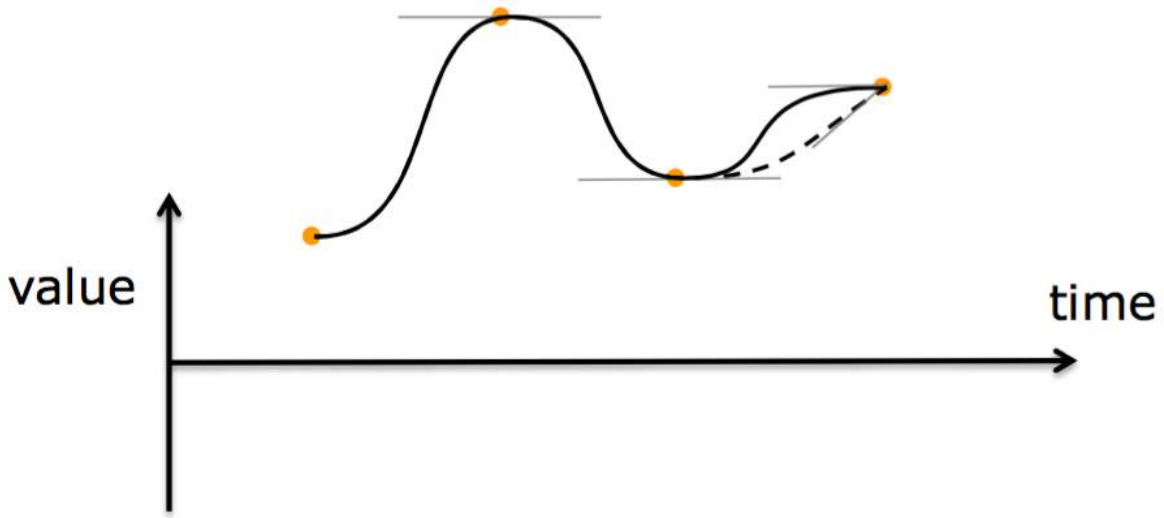
Interpolation

How do we interpolate between two values?



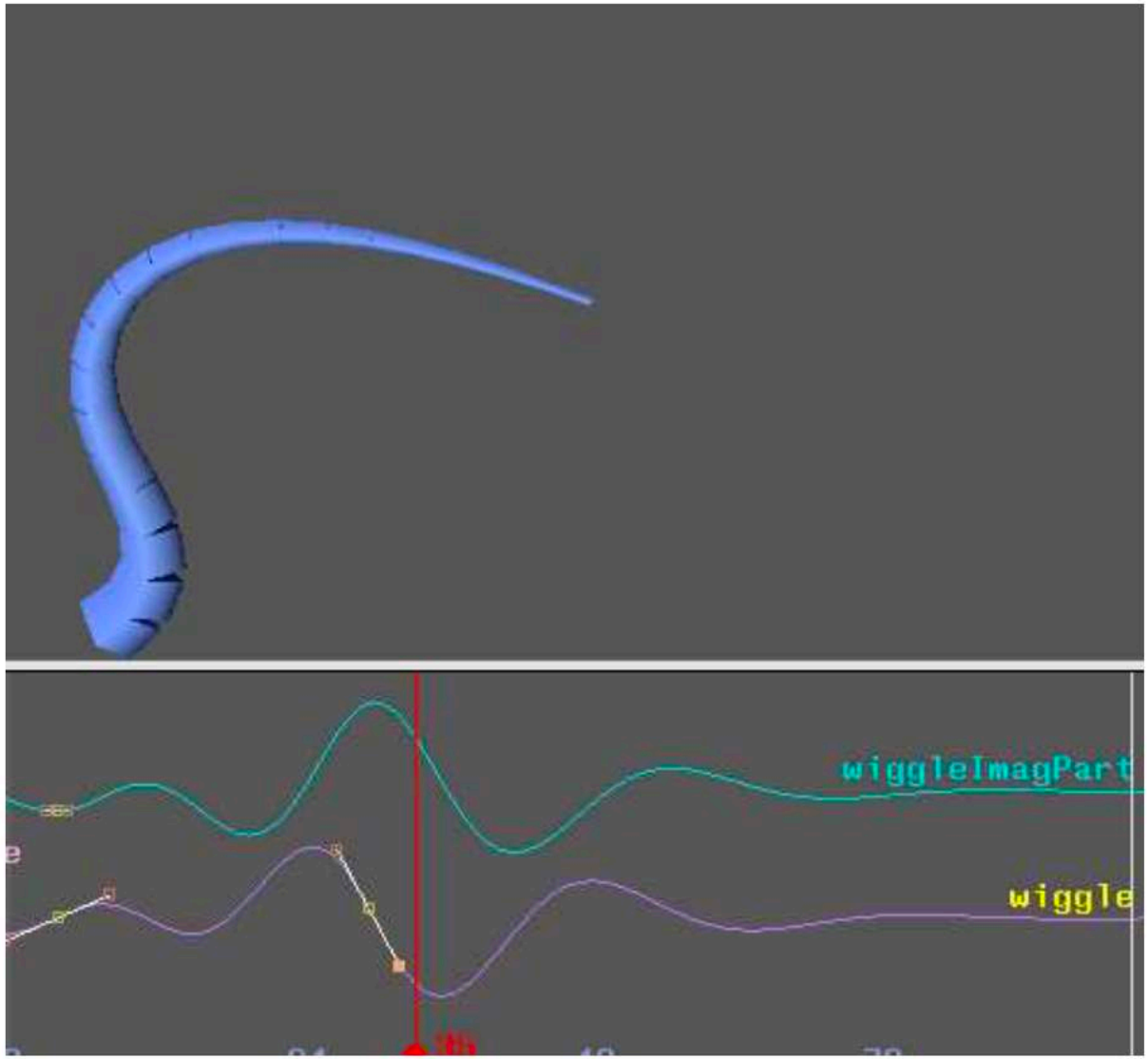
Interpolation

How do we interpolate between two values?



Ease-in Ease-out

Wiggly Splines



Kass and Anderson SIGGRAPH 2008

Forward Kinematics

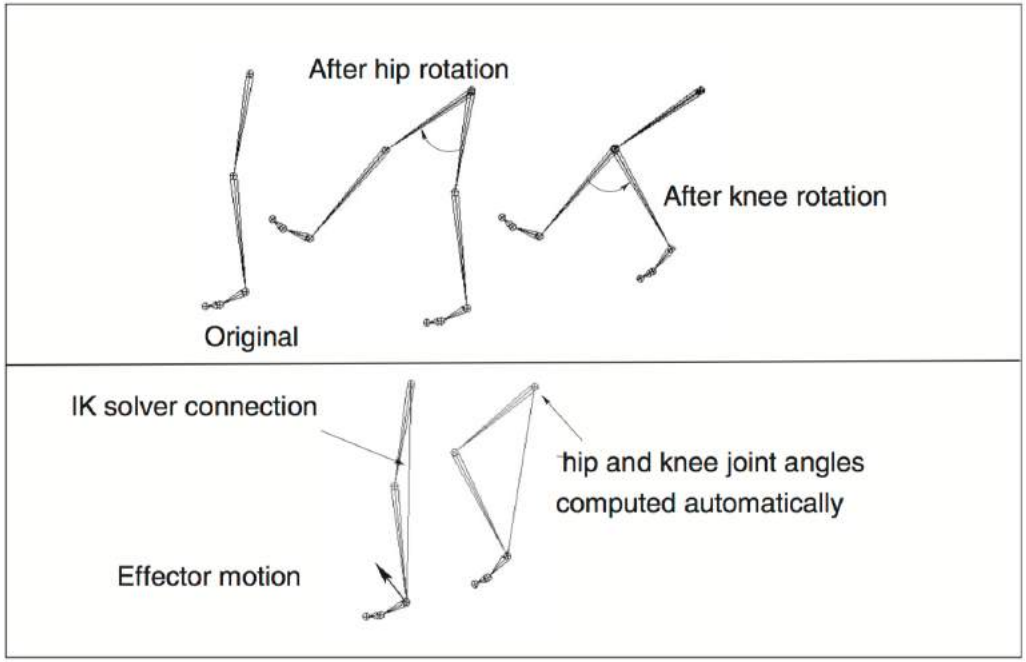
- Specify how joints should move
- Determine the space of possible motion
- Parametrize it
- Establish a mapping from joint angles to positions

Pros:

- Very easy to specify and implement

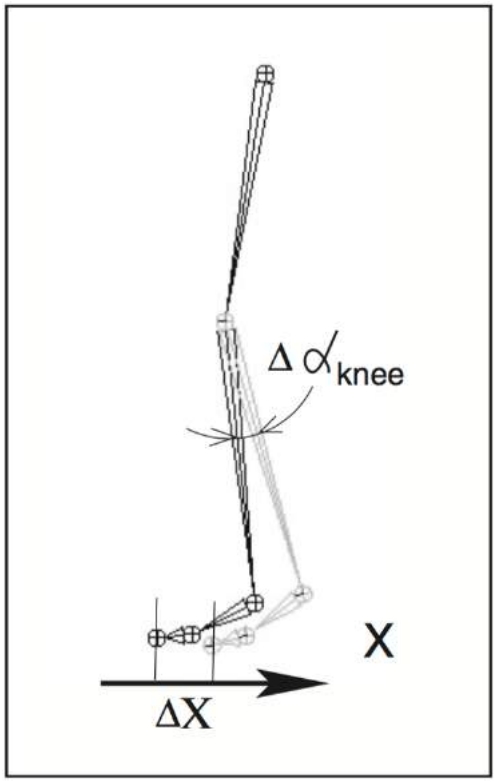
Cons:

- Often we care about <u>where</u> the character should go, not <u>how</u> to get there
- Very hard to know how to move joints of a complicated figure in order to get the desired pose (esp. in presence of obstacles)



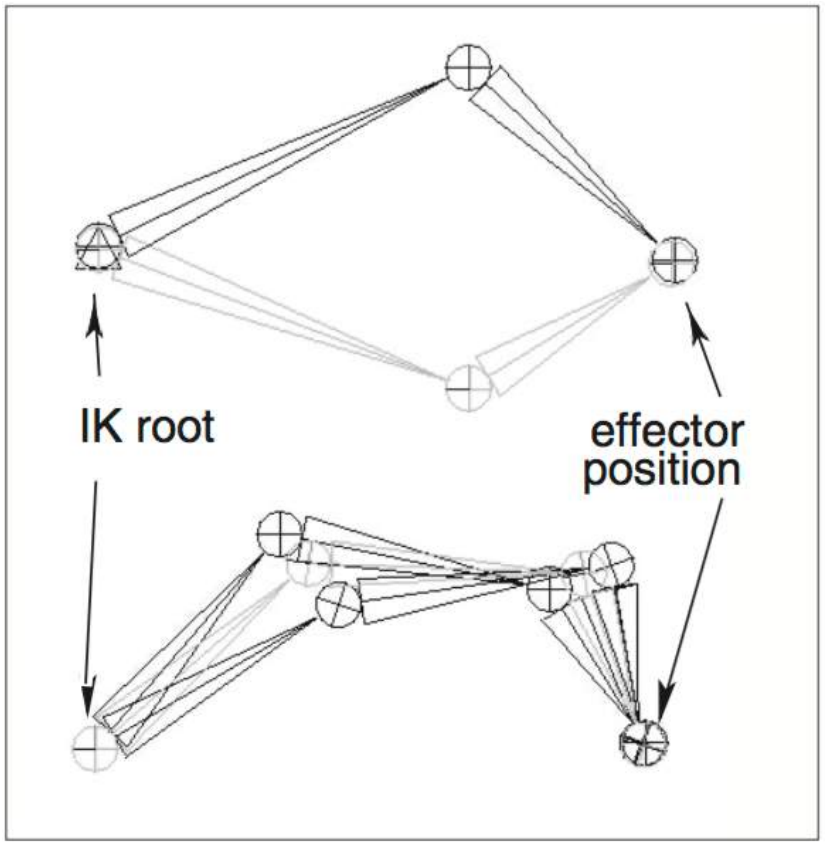
Inverse Kinematics

- Specify where character should go, then deduce join motion
- Position of the end effector is set by the animator.
- Automatically solve for joint angles that will result in that effector position.
- Solution is not usually “closed form”, iteratively solved by optimizer.

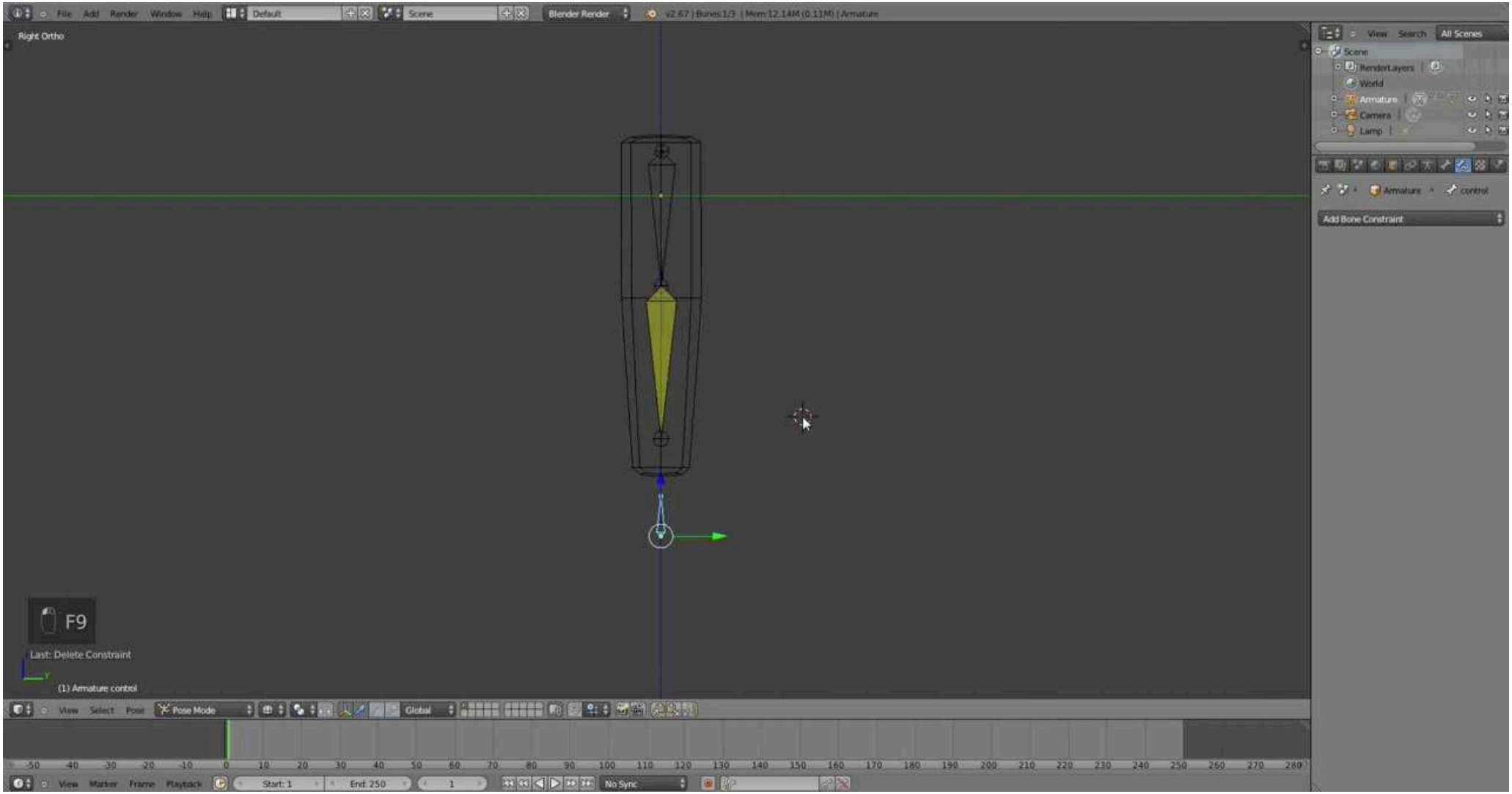


Inverse Kinematics

- Typically underconstrained: Multiple configurations of internal joints can result in the same effector position



Inverse Kinematics



Physical Simulation

- Mathematically model real-world motion
- Animate via simulation
- Smoke, fire, clouds, fluids, cloth, rigid bodies, elastic objects.

Pros:

- Once implemented, easy to specify the state of the system at a particular time.
- Model can be easily changed via parameter settings
- High degree of physical realism

Cons:

- Complex systems used to model natural phenomena
- Computationally intensive

Physical Simulation

Particles

Position	x
Velocity	$v = dx/dt$
Acceleration	$a = dv/dt = d^2x/dt^2$

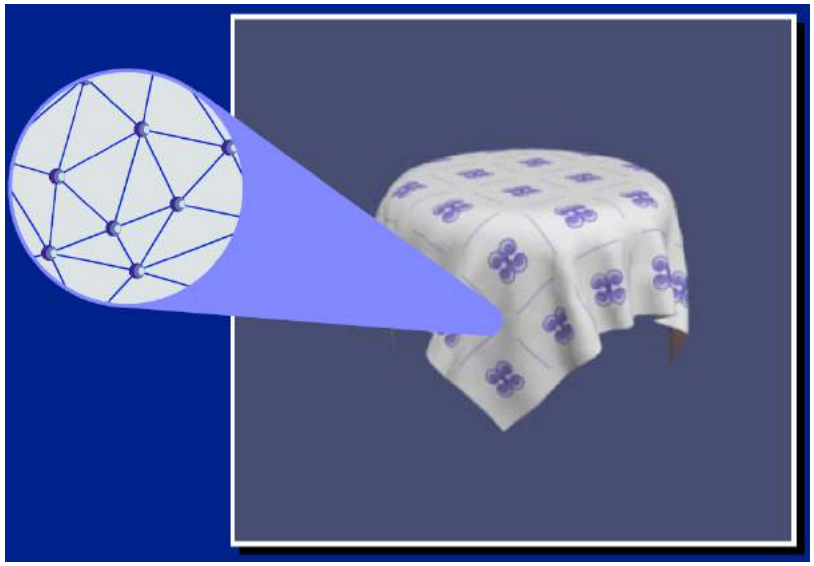
Forces

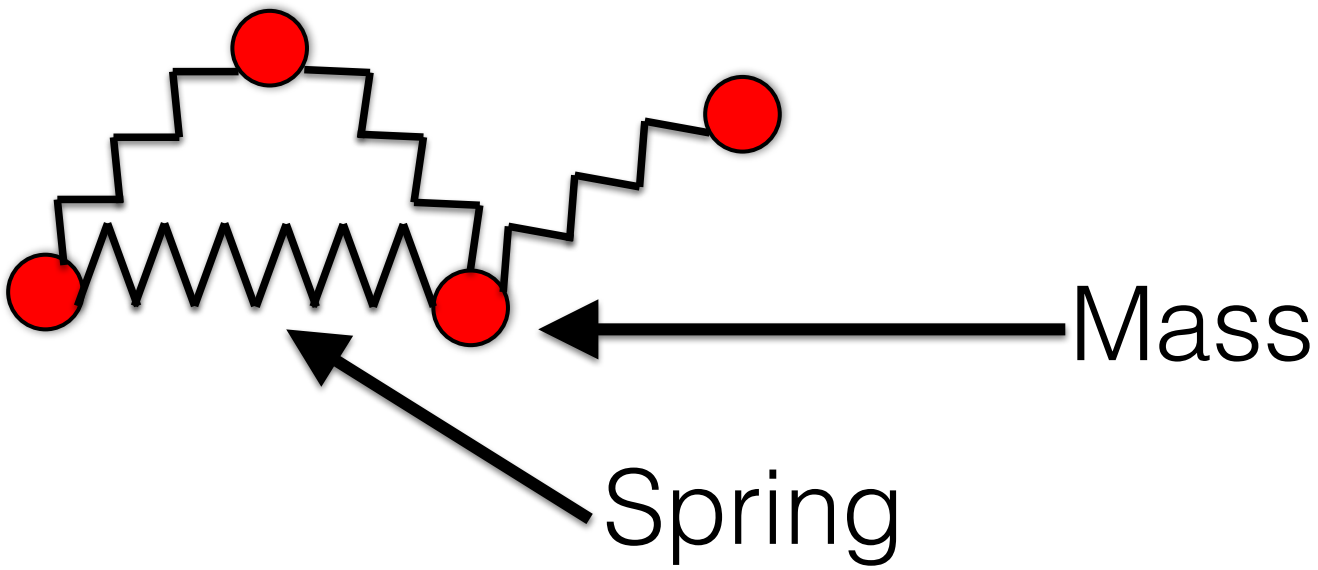
Gravity	$f=mg$
Other Stuff …	

Simulation: x,v,a used to compute forces yielding total force F, F=ma used to update a, a used to update v,x...

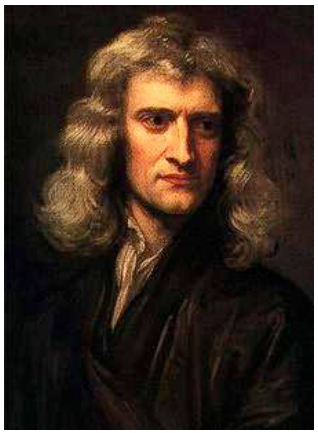
Mass Spring Systems

- One way of modeling deformable objects is as a network of masses and springs





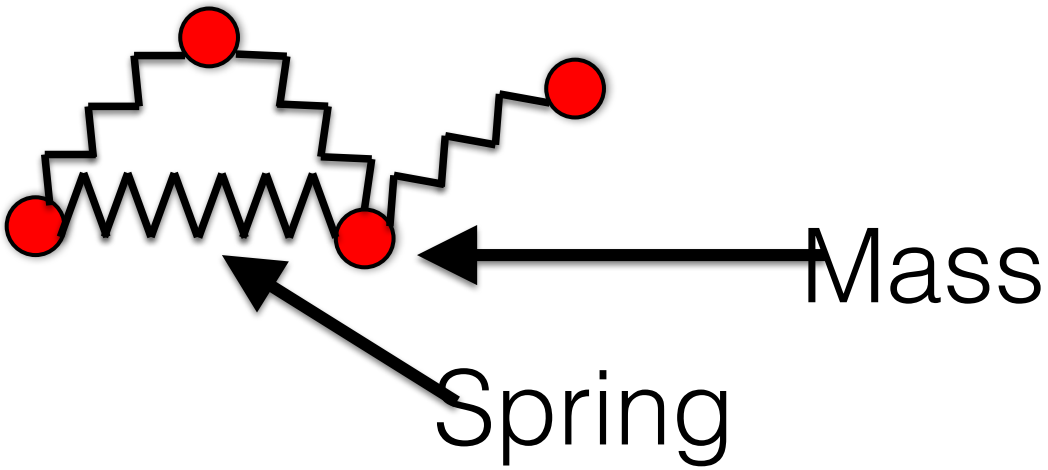
The Motion of a Mass Spring Systems



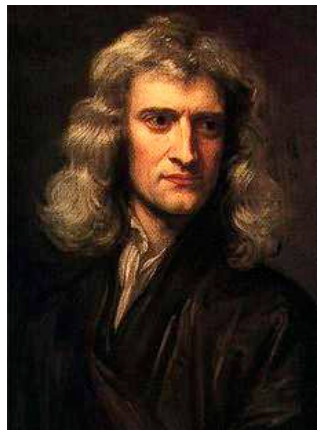
- The acceleration of a point mass is given by:

$$m\mathbf{a} = \mathbf{f}$$

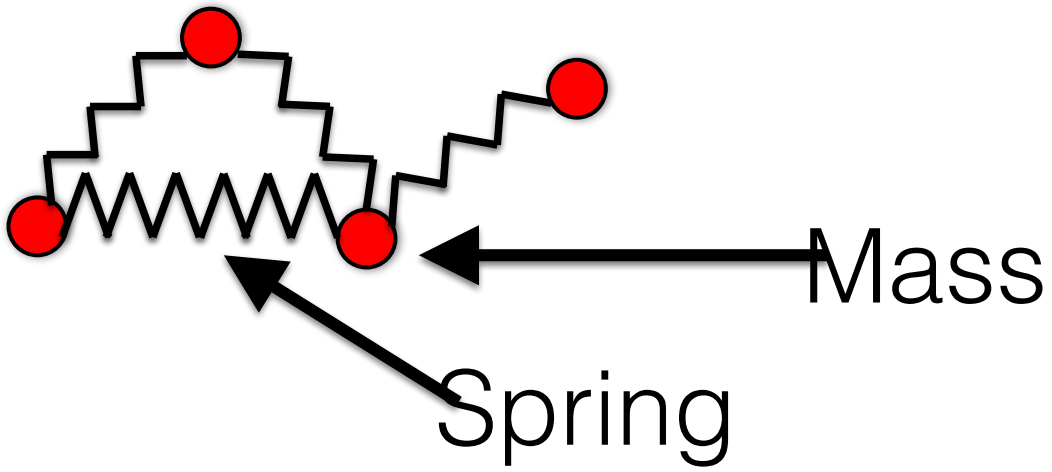
- We can use standard methods to integrate this system forward in time in order to compute the velocities and displacements of each point in the mass spring system



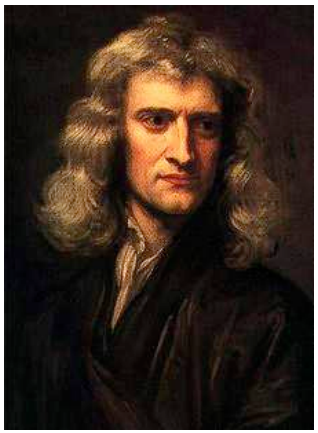
The Motion of a Mass Spring Systems



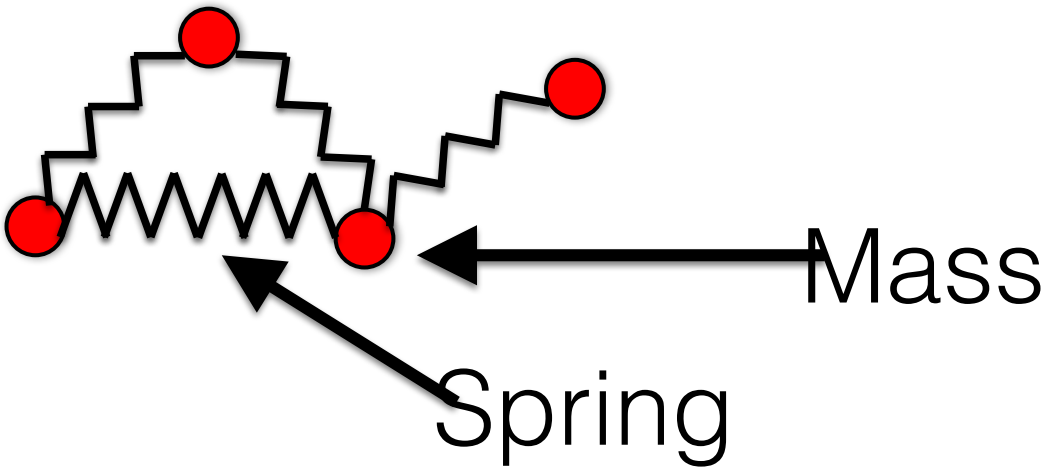
- Pseudocode:
  - a←0 //Array of accelerations
  - v←0 //Array of velocities
  - p←0 //Array of positions
  - For each particle, p
    - m←mass of the particle
    - f←sum of all spring forces acting on the particle
    - a[p]←f/m
  - End
  - v←Integrate(a)
  - p←Integrate(v)



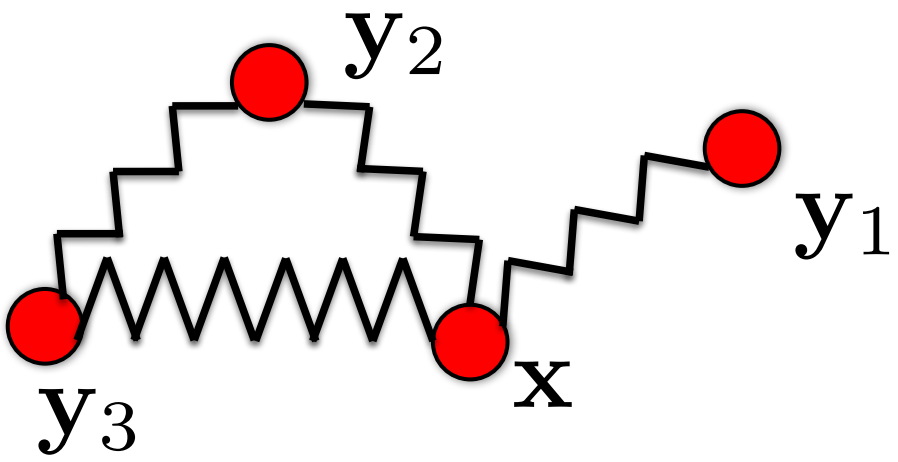
The Motion of a Mass Spring Systems



- Pseudocode:
  - a←0 //Array of accelerations
  - v←0 //Array of velocities
  - P←0 //Array of positions
  - For each particle, p
    - m←mass of the particle
    - f←sum of all spring forces acting on the particle
    - a[p]←f/m
  - End
  - v←Integrate(a)
  - p←Integrate(v)



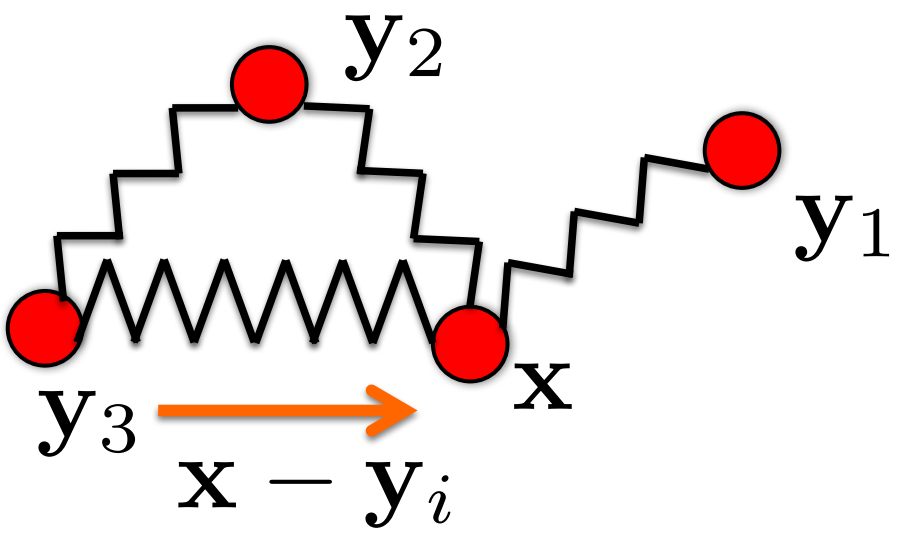
Mass Spring Systems: Forces



- The inertia on this point mass is given by:

$$m\mathbf{a} = \sum_{i=1}^{\text{springs}} -k\left(\left(\frac{l}{l_0} - 1\right)\frac{\mathbf{x} - \mathbf{y}_i}{|\mathbf{x} - \mathbf{y}_i|}\right)$$

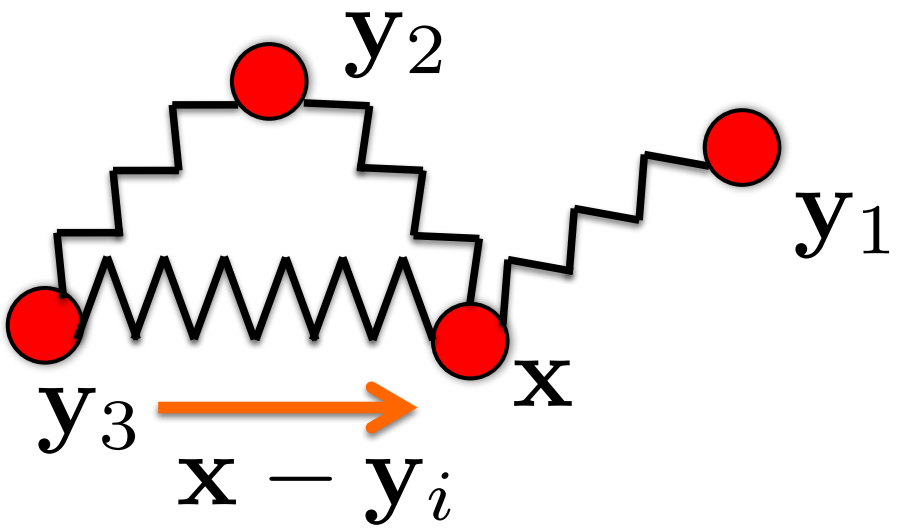
Mass Spring Systems: Refresher



- The inertia on this point mass is given by:

$$m\mathbf{a} = \sum_{i=1}^{\text{springs}} -k\left(\left(\frac{l}{l_0} - 1\right)\frac{\mathbf{x} - \mathbf{y}_i}{|\mathbf{x} - \mathbf{y}_i|}\right)$$

Mass Spring Systems: Refresher



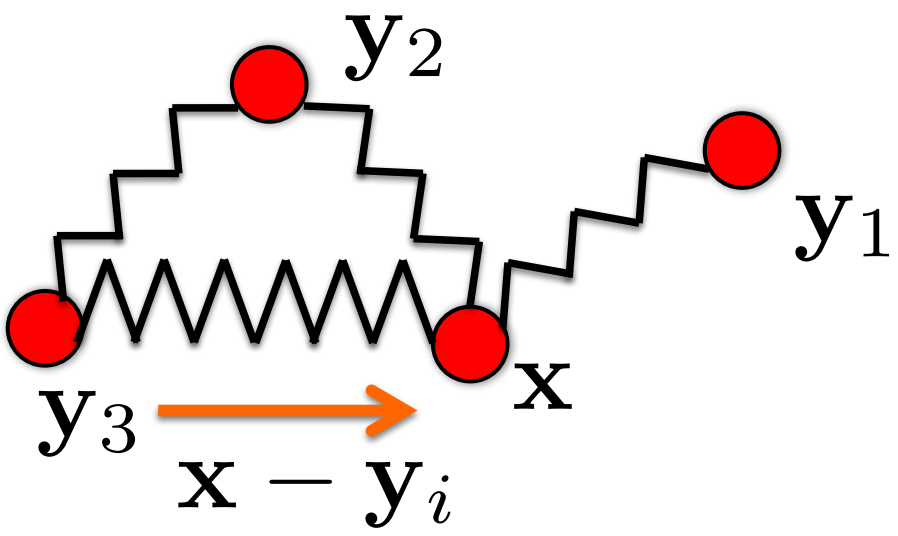
$$l = |\mathbf{x} - \mathbf{y}_i|$$

$l_0 =$ Original length

- The inertia on this point mass is given by:

$$m\mathbf{a} = \sum_{i=1}^{\text{springs}} -k \left(\left(\frac{l}{l_0} - 1 \right) \frac{\mathbf{x} - \mathbf{y}_i}{|\mathbf{x} - \mathbf{y}_i|} \right)$$

Mass Spring Systems: Refresher



$$l = |\mathbf{x} - \mathbf{y}_i|$$

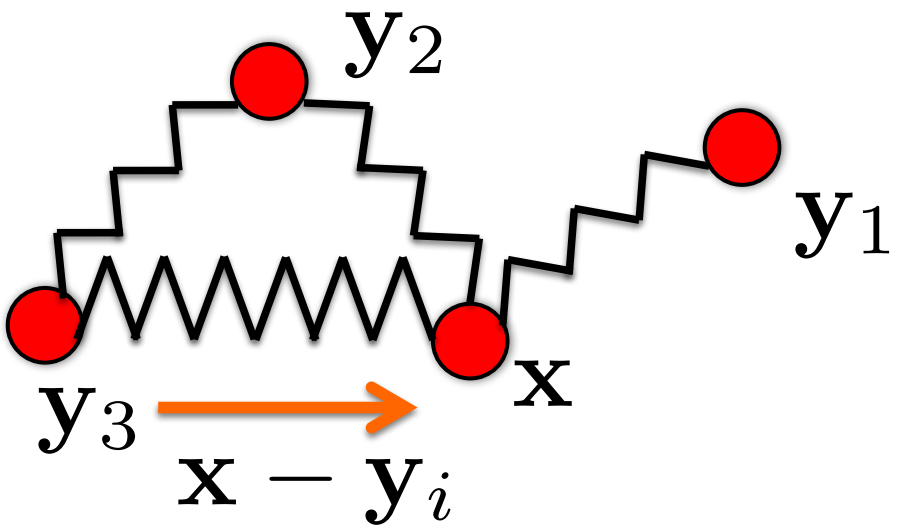
$l_0 =$ Original length

Direction Vector

$$m\mathbf{a} = \sum_{i=1}^{\text{springs}} -k\left(\left(\frac{l}{l_0} - 1\right)\frac{\mathbf{x} - \mathbf{y}_i}{|\mathbf{x} - \mathbf{y}_i|}\right)$$

- The acceleration on this point mass is given by:

Mass Spring Systems: Refresher



$$l = |\mathbf{x} - \mathbf{y}_i|$$

$l_0 =$ Original length

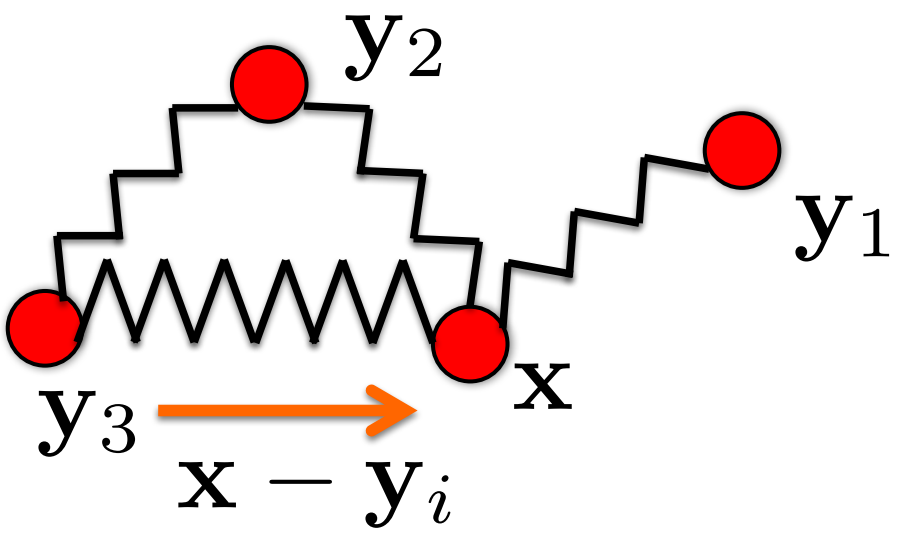
- The inertia on this point mass is given by:

Deformation

$$m\mathbf{a} = \sum_{i=1}^{\text{springs}} -k \left(\left(\frac{l}{l_0} - 1 \right) \frac{\mathbf{x} - \mathbf{y}_i}{|\mathbf{x} - \mathbf{y}_i|} \right)$$

Stiffness

Mass Spring Systems: Refresher



$$l = |\mathbf{x} - \mathbf{y}_i|$$

l_0 Original length

- The inertia on this point mass is given by:

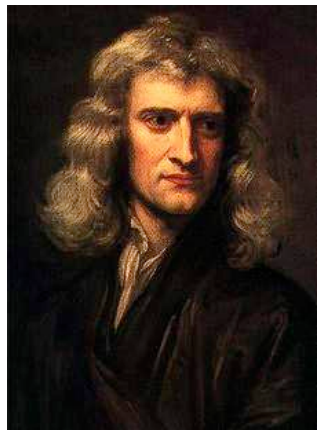
Deformation

$$m\mathbf{a} = \sum_{i=1}^{\text{springs}} -k\left(\left(\frac{l}{l_0} - 1\right)\frac{\mathbf{x} - \mathbf{y}_i}{|\mathbf{x} - \mathbf{y}_i|}\right)$$

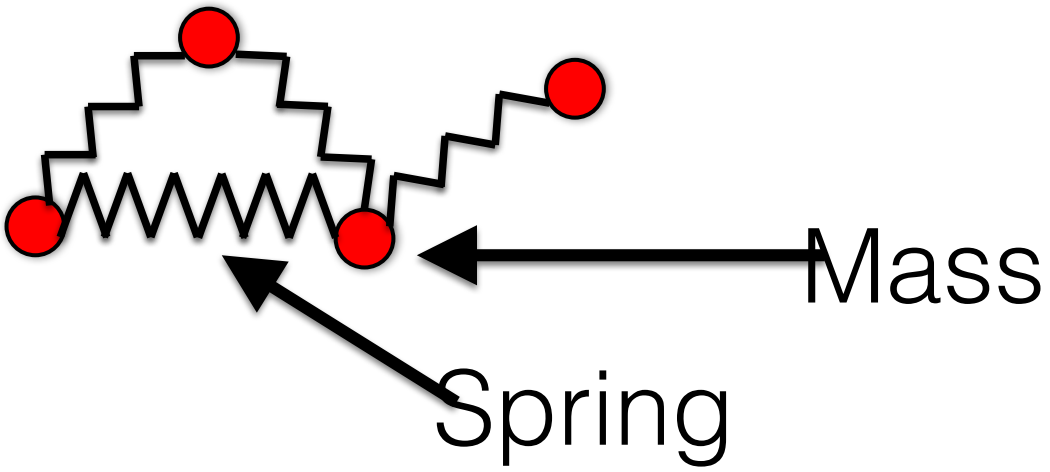
- The acceleration on this point mass is given by:

Stiffness

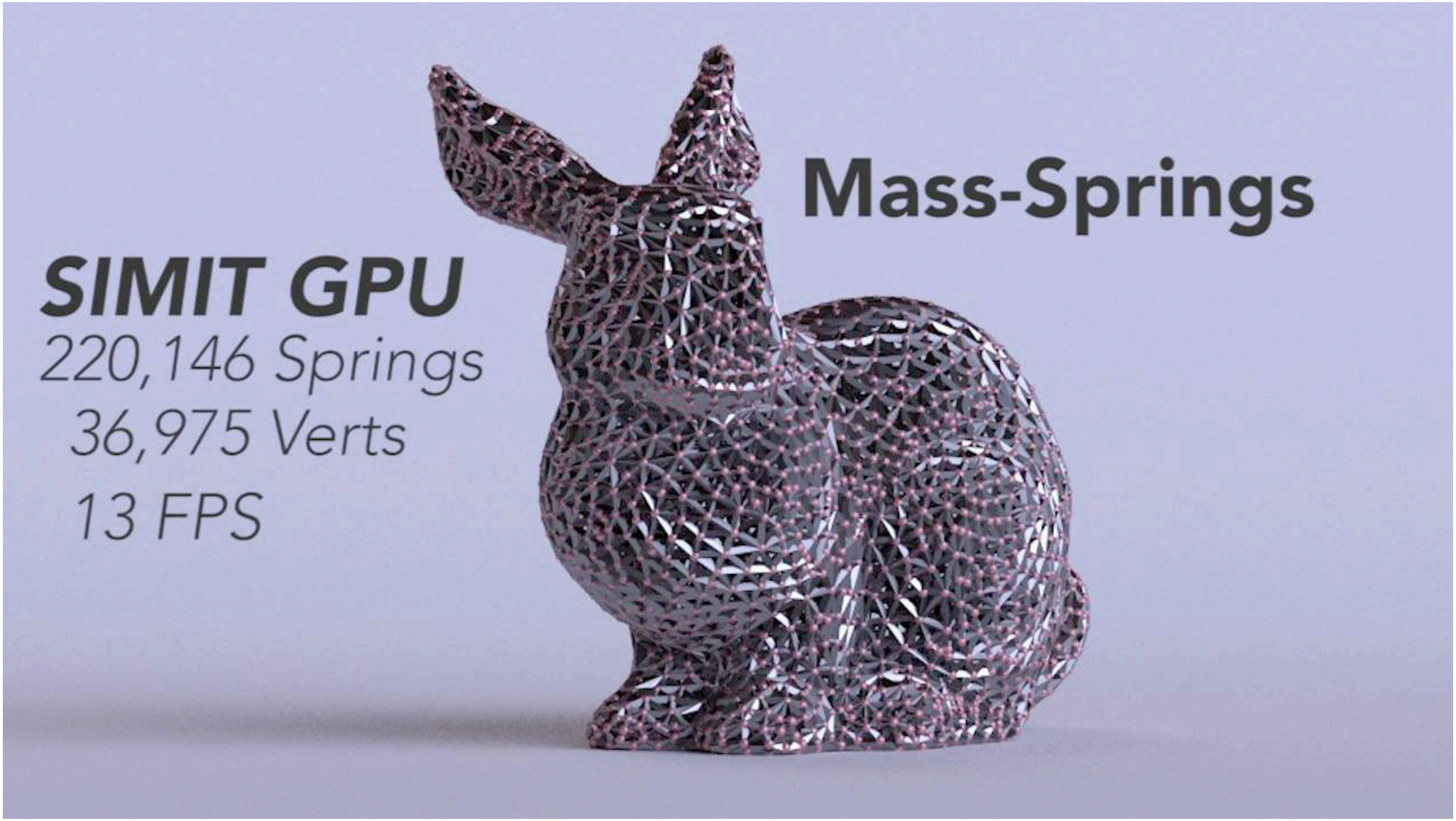
The Motion of a Mass Spring Systems



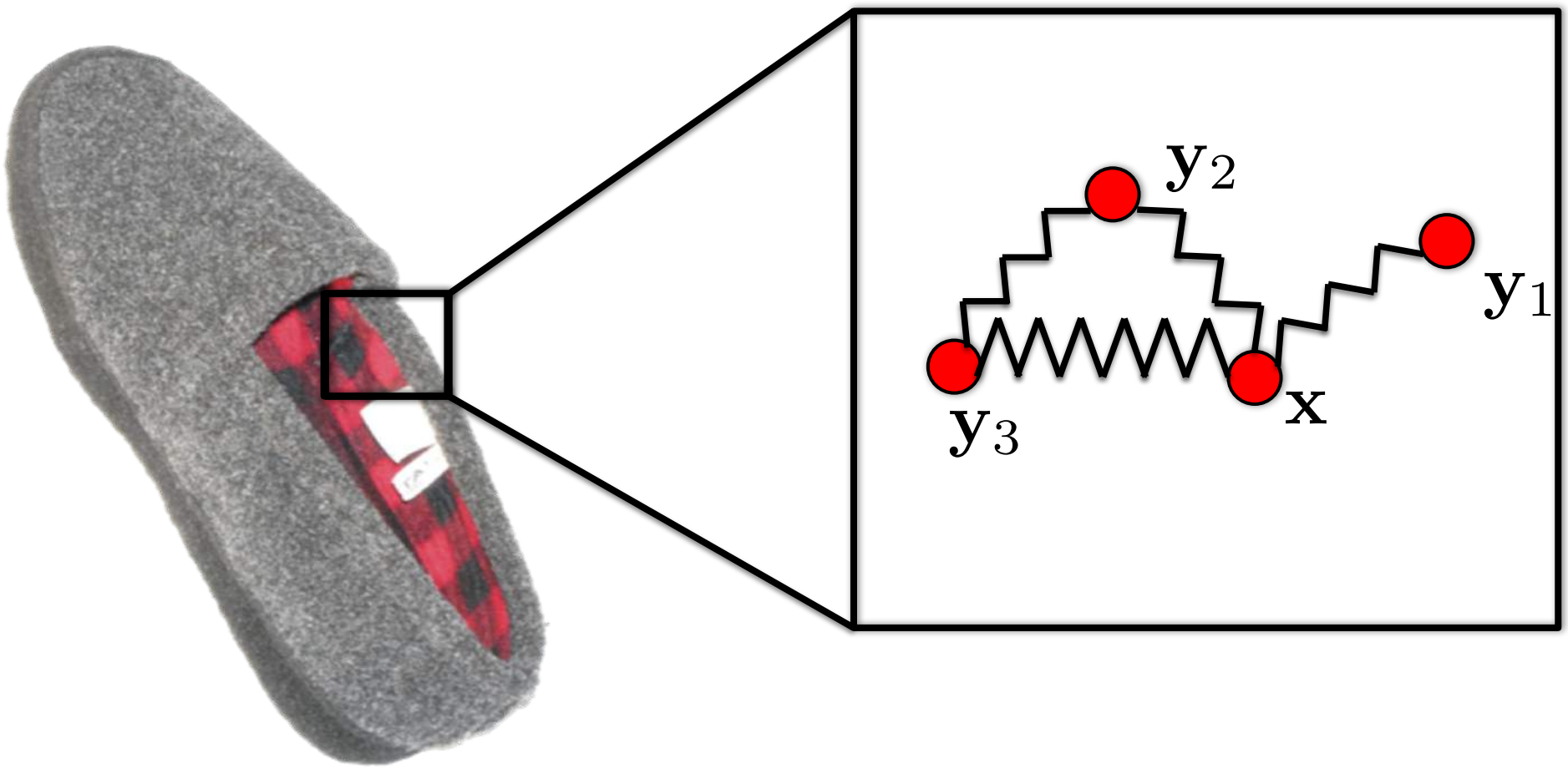
- Pseudocode:
  - a←0 //Array of accelerations
  - v←0 //Array of velocities
  - P←0 //Array of positions
  - For each particle, p
    - m←mass of the particle
    - f←sum of all spring forces acting on the particle
    - a[p]←f/m
  - End
  - v←Integrate(a)
  - p←Integrate(v)



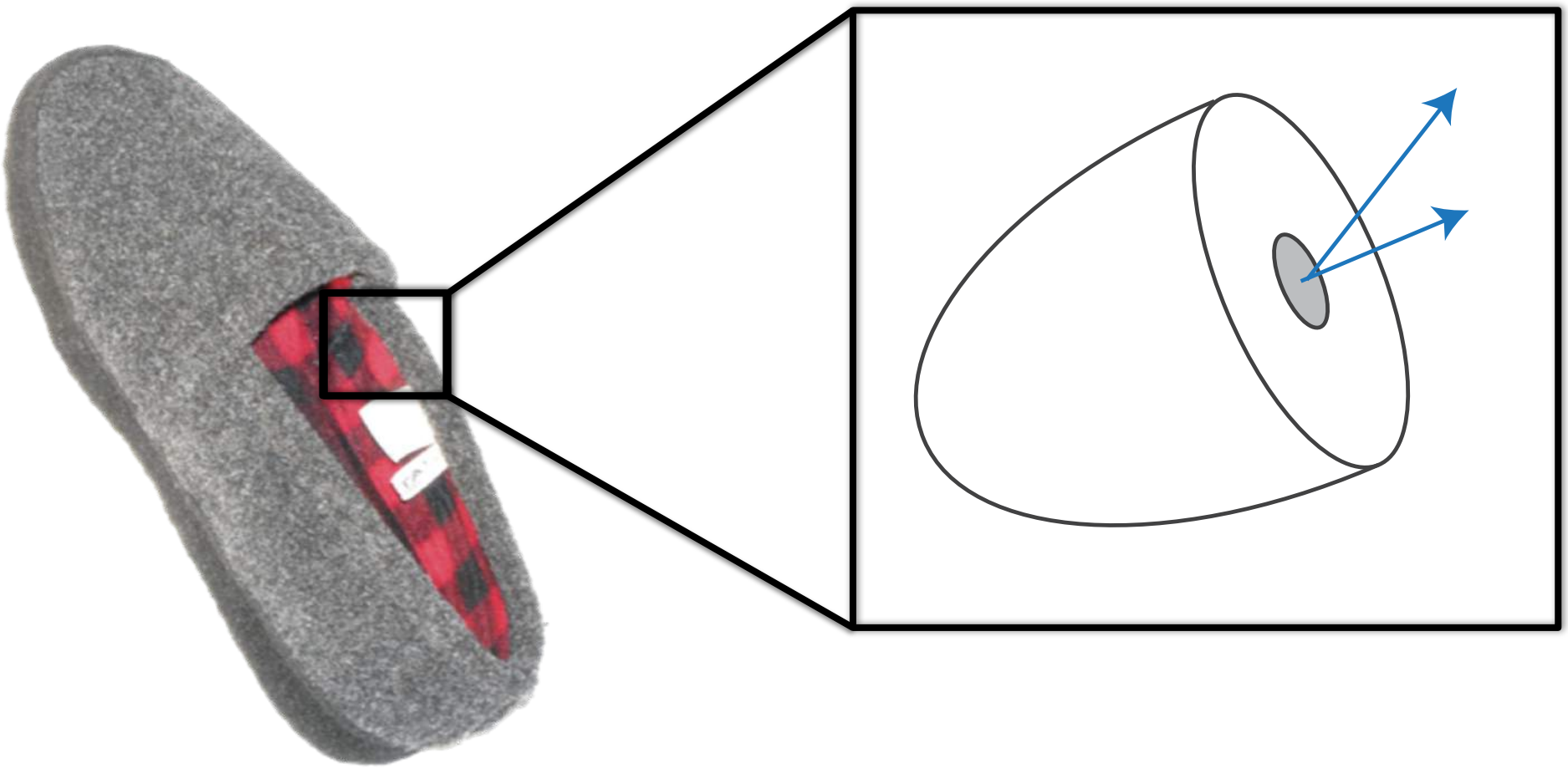
Physical Simulation: Springs



Springs

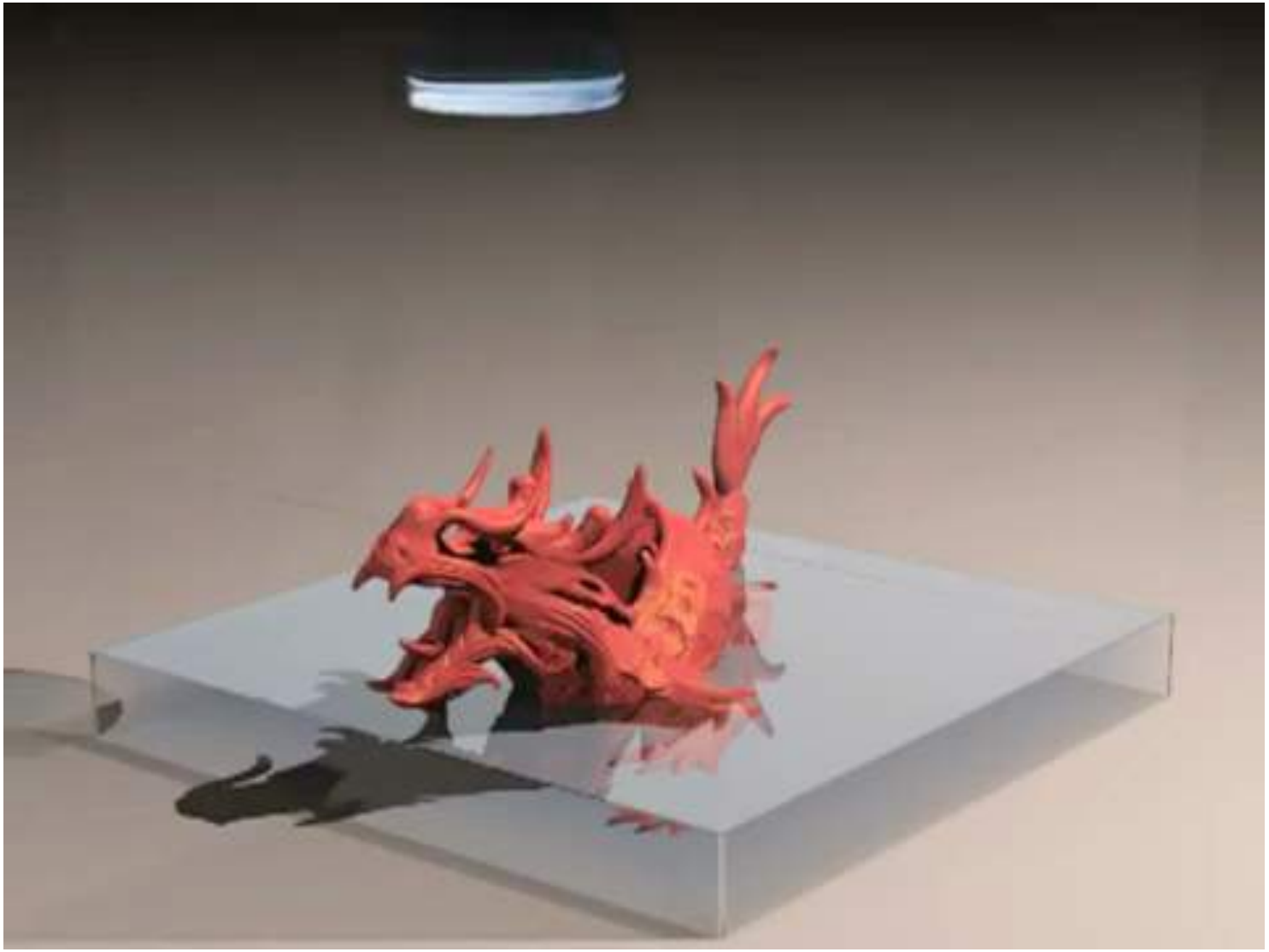


Continuum



Physical Simulation: FEM

Physical Simulation: Fluids

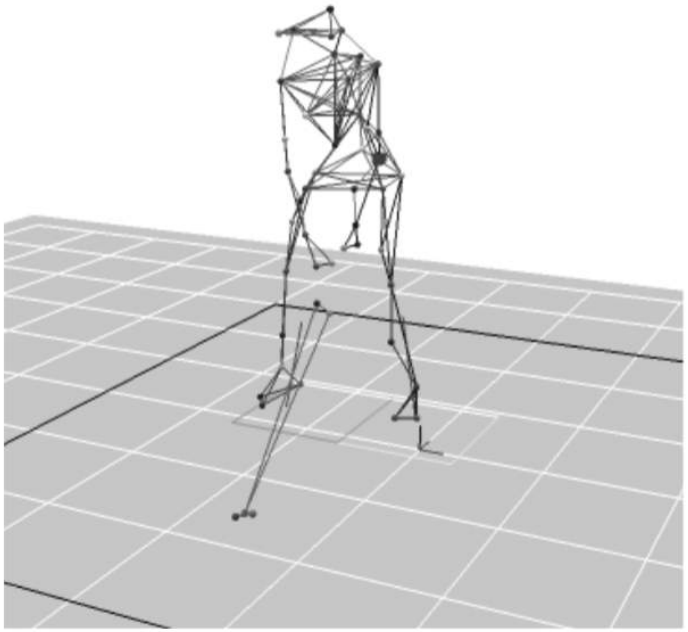


Keyframing Physics

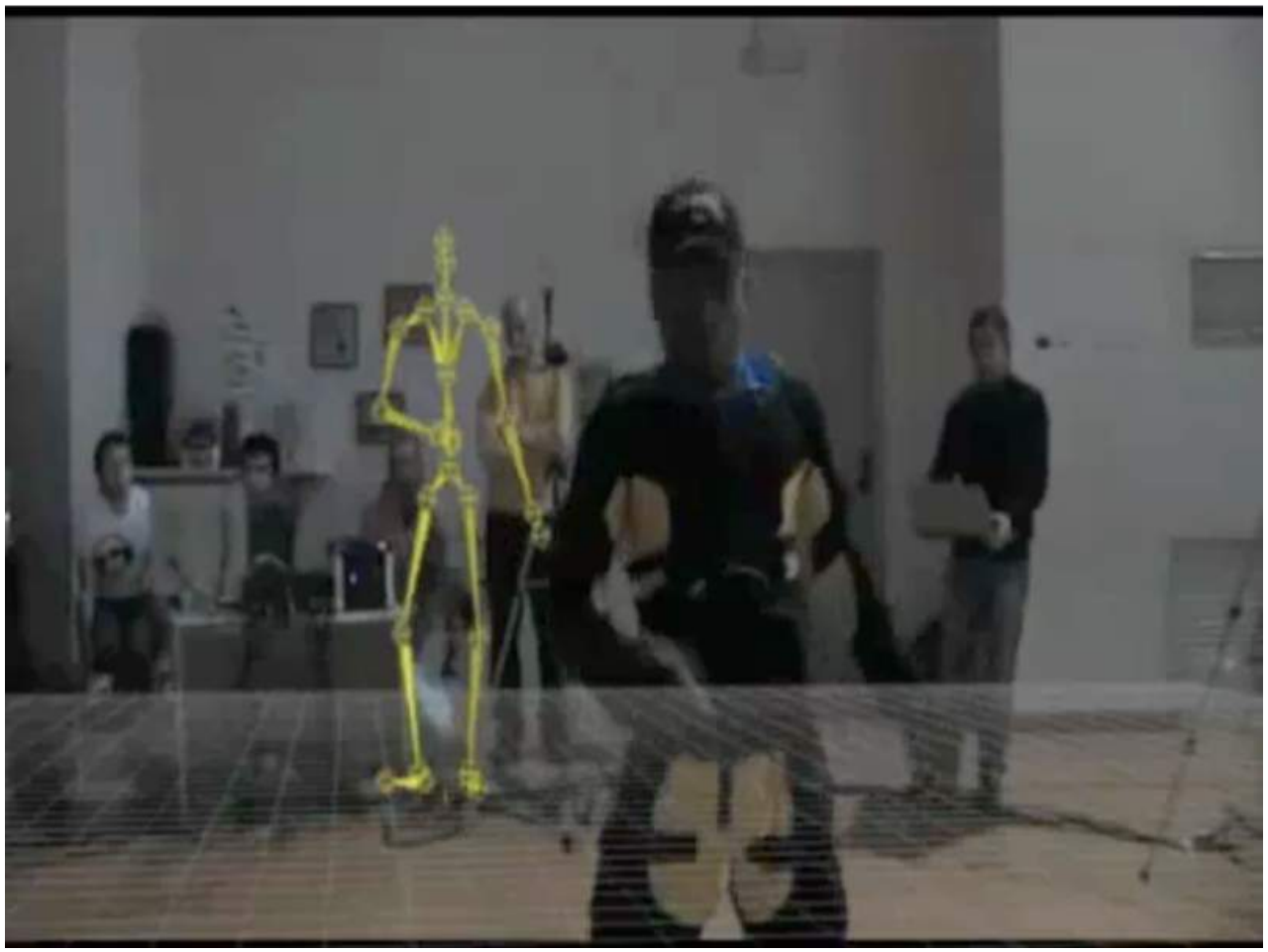
Motion Capture

- Record motions of real people/objects then transfer to digital characters.
- Markers can be occluded. Use multiple cameras and interpolate where needed.
- Noise can cause limbs to loose contact with ground or other objects. Use redundant markers to reduce noise. Correct with Inverse Kinematics.
- Retargeting: applying recorded motion to different characters.

Motion Capture



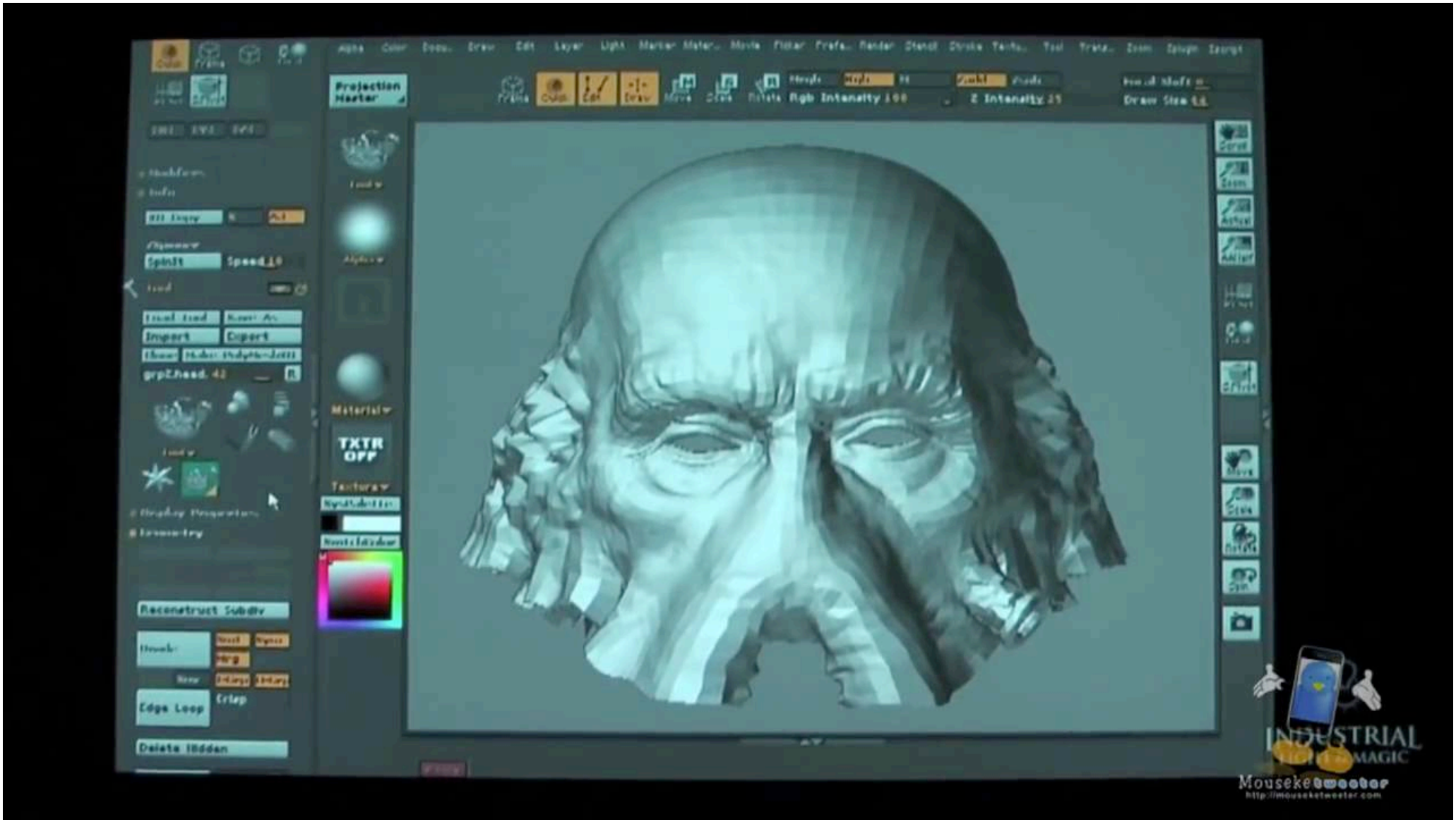
Motion Capture



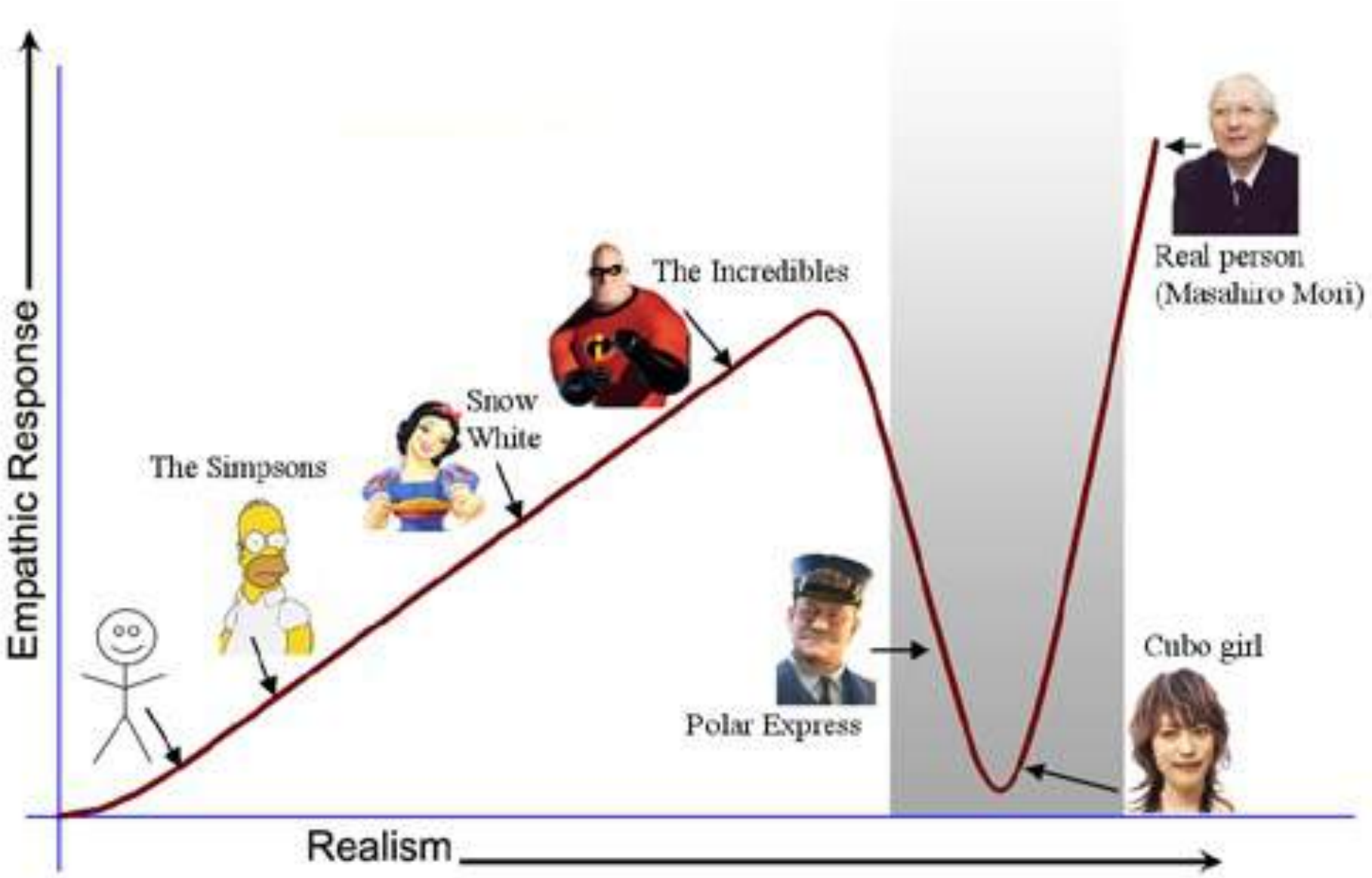
Motion Capture

Motion Capture

Motion Capture ++



Uncanny Valley



Uncanny Valley