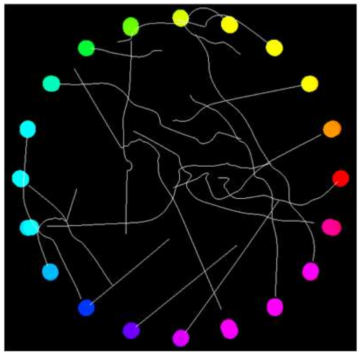
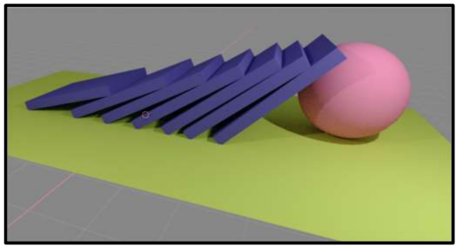
Animation

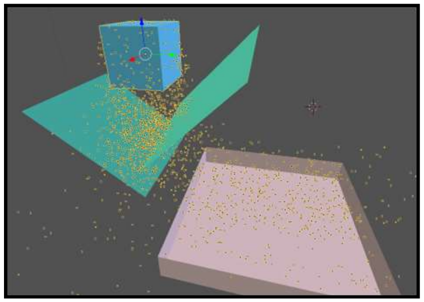


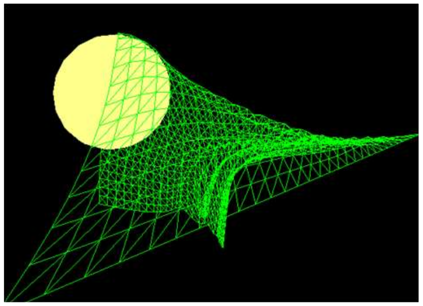


Oregon State University

Mike Bailey

mjb@cs.oregonstate.edu



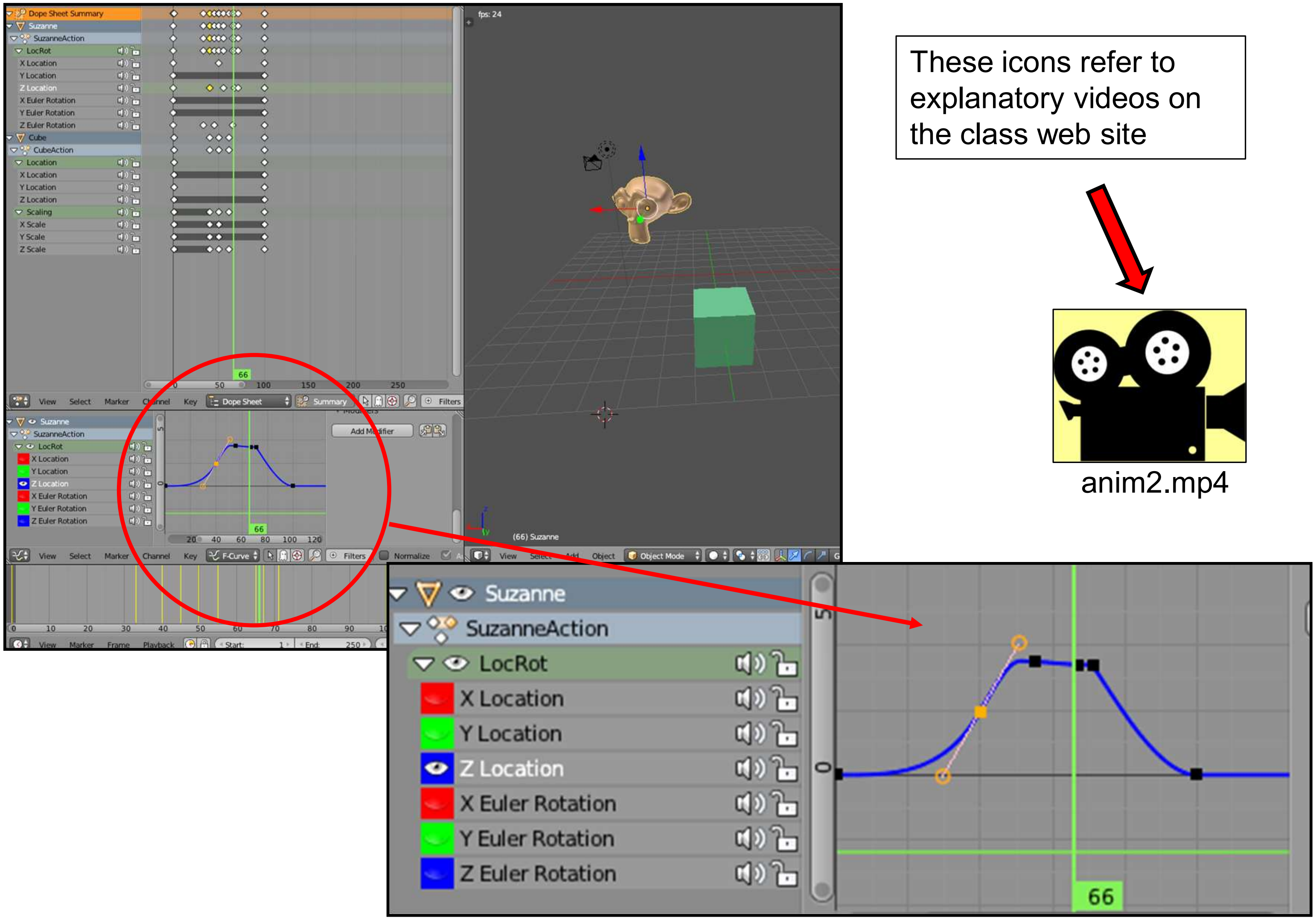


Animation

Animation is the process of giving motion to your geometric models. Before animating, there are questions you need to ask first:

- Why am I doing this?
- Do I want the animation to obey the real laws of physics? Partially? Which elements?
- Am I willing to “fake” the physics to get the objects to *want* to move in a way that I tell it?
- Do I have specific key positions I want the objects to pass through no matter what?
- Do I want to simply record the motion of a real person, animal, etc., and then play it back?

Keyframe Animation

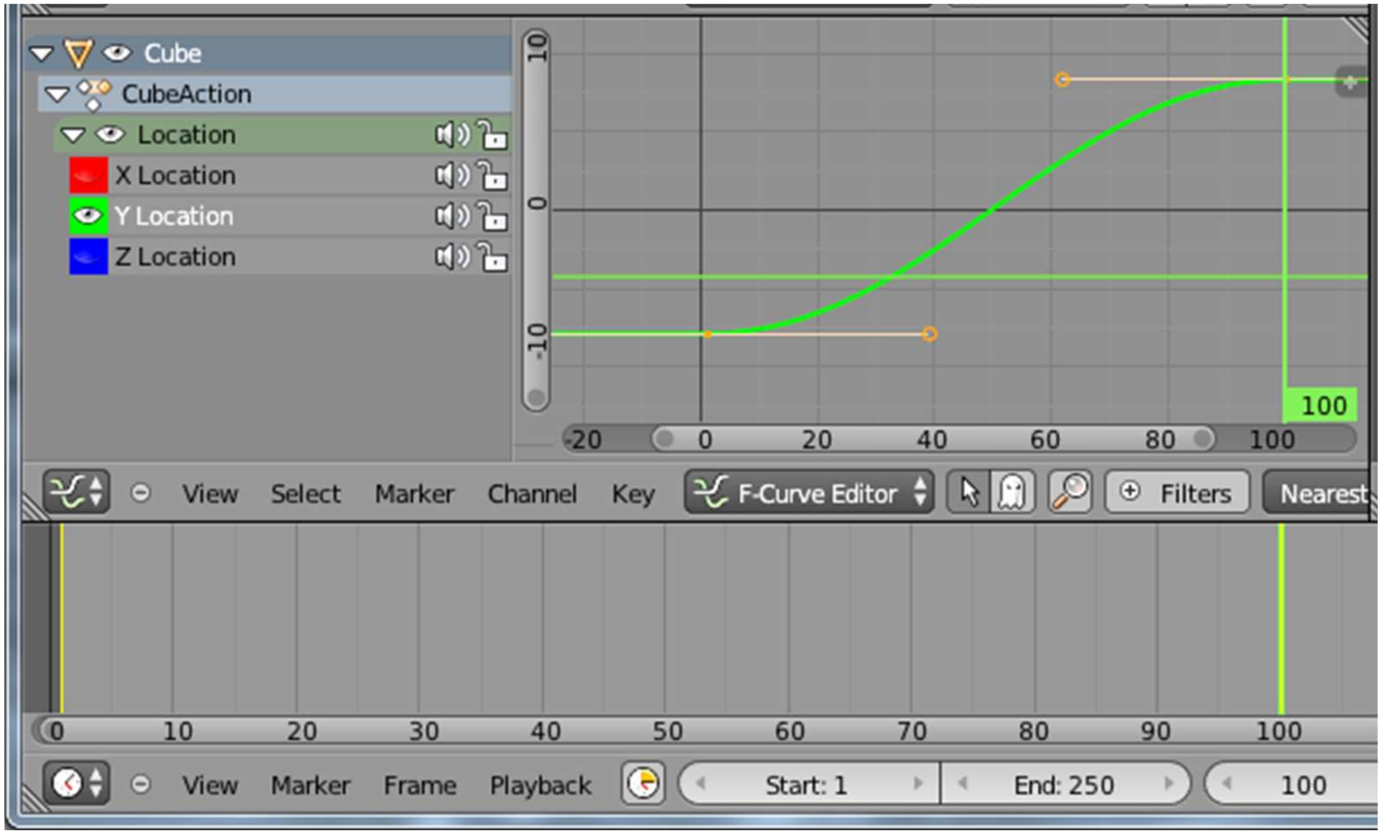


These icons refer to explanatory videos on the class web site

anim2.mp4

Keyframe Animation

Blender:



Here’s Some Code that Lets You Create DIY Keyframe Animations

For my own work, instead of *Key Frames*, I like specifying *Key Times* better.
And, so, I created a C++ class to do it for you.

```
class Keytimes:

            void    AddTimeValue( float time, float value );
            float   GetFirstTime( );
            float   GetLastTime( );
            int     GetNumKeytimes( );
            float   GetValue( float time );
            void     Init( );
            void    PrintTimeValues( );
```

Instead of Key *Frames*, I Like Specifying Key *Times* Better

```
Keytimes Xpos;
```

Declare one class per parameter you are animating. Here it wants to interpolate and animate the x-location of something.

```
int
main( int argc, char *argv[ ] )
{
        Xpos.Init( );
        Xpos.AddTimeValue(  0.0,  0.000 );
        Xpos.AddTimeValue(  2.0,  0.333 );
        Xpos.AddTimeValue(  1.0,  3.142 );
        Xpos.AddTimeValue(  0.5,  2.718 );
```

Key-Time pairs (Time and Xlocation) can be specified in any order

```
        fprintf( stderr, "%d time-value pairs:\n", Xpos.GetNumKeytimes( ) );
        Xpos.PrintTimeValues( );

        fprintf( stderr, "Time runs from %8.3f to %8.3

        for( float t = 0.; t <= 2.0; t += 0.1 )
        {
                float v = Xpos.GetValue( t );
                fprintf( stderr, "%8.3f\t%8.3f\n", t, v );
        }
}
```

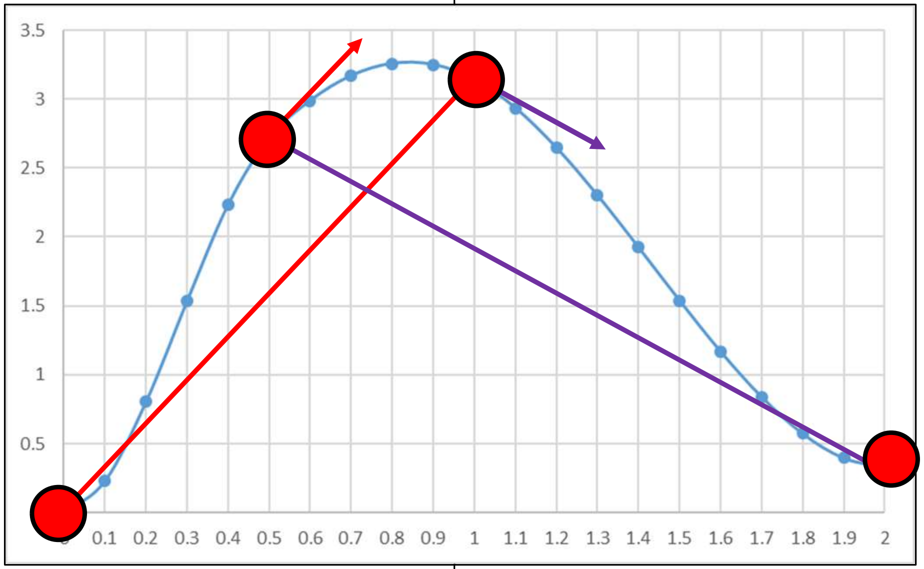
Just to demonstrate, this for-loop runs through a collection of time values and looks up the interpolate x-location at those values. Normally you would get the time from the system clock.

Oregon State
University
Computer Graphics

Instead of Key *Frames*, I Like Specifying Key *Times* Better

```
(  0.00,   0.000)
(  0.00,   0.000)  (  2.00,   0.333)
(  0.00,   0.000)  (  1.00,   3.142)  (  2.00,   0.333)
(  0.00,   0.000)  (  0.50,   2.718)  (  1.00,   3.142)  (  2.00,   0.333)
4 time-value pairs
Time runs from    0.000 to    2.000
   0.000        0.000
   0.100        0.232
   0.200        0.806
   0.300        1.535
   0.400        2.234
   0.500        2.718
   0.600        2.989
   0.700        3.170
   0.800        3.258
   0.900        3.250
   1.000        3.142
   1.100        2.935
   1.200        2.646
   1.300        2.302
   1.400        1.924
   1.500        1.539
   1.600        1.169
   1.700        0.840
   1.800        0.574
   1.900        0.397
   2.000        0.333
```

● = the originally-specified key-time pairs

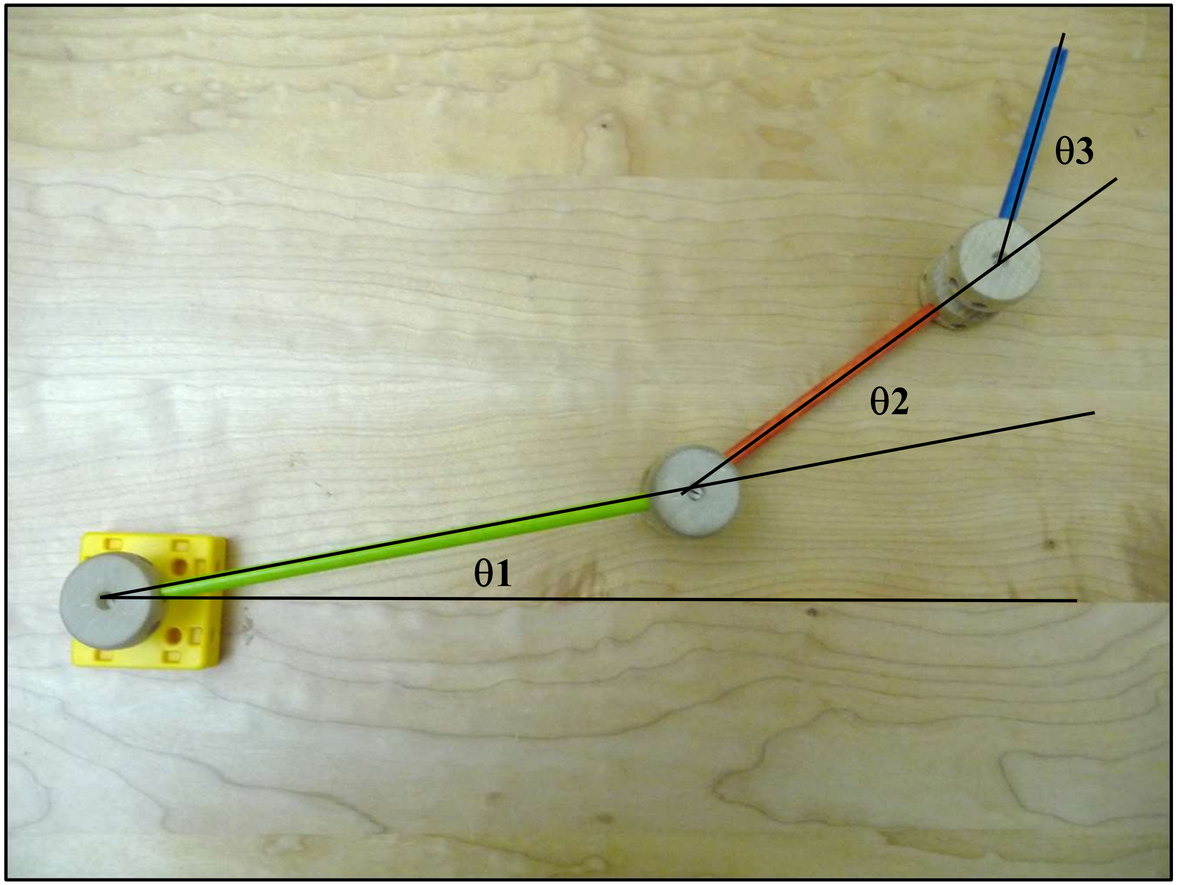


Using the System Clock in Display( ) for Timing

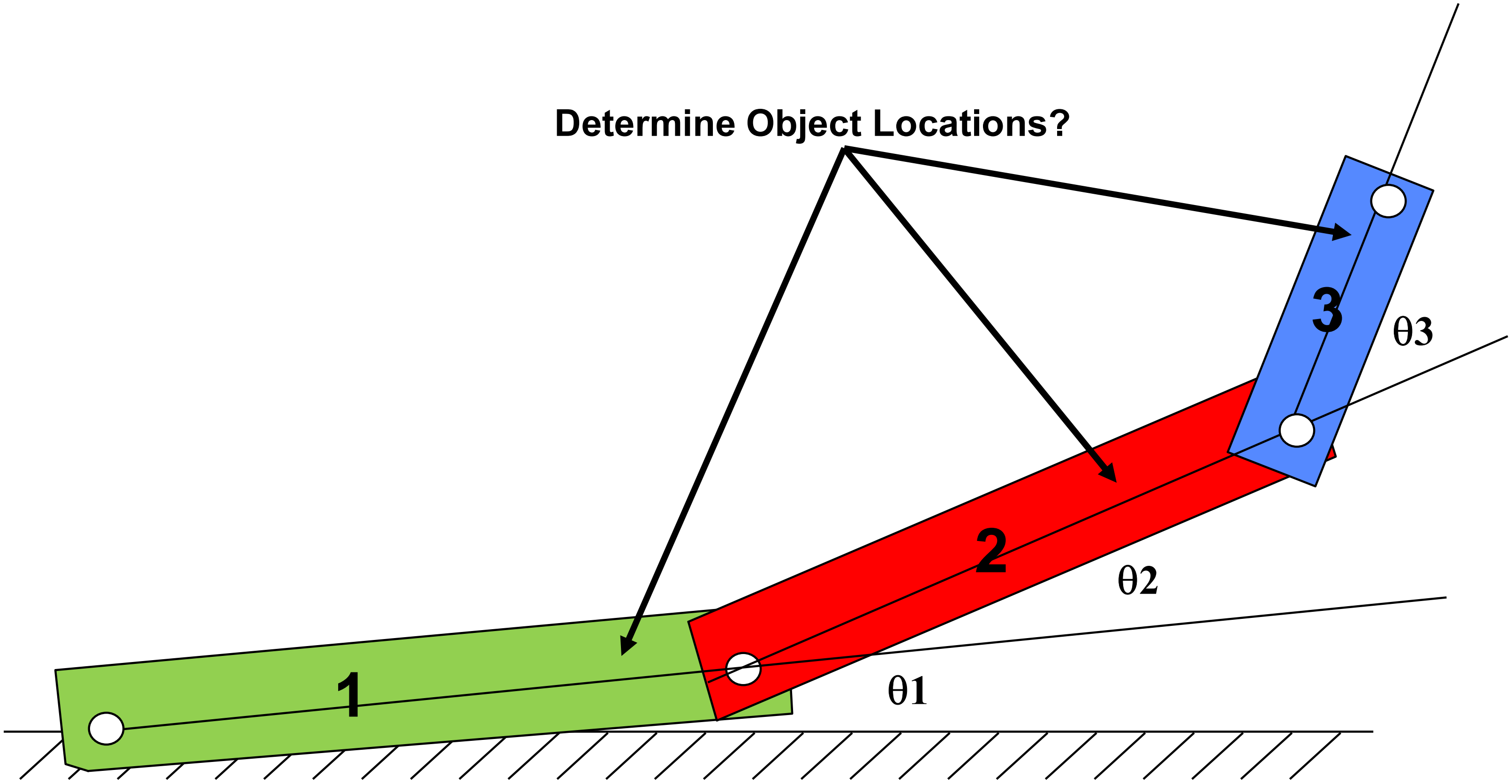
```
#define MSEC        10000                          // i.e., 10 seconds
Keytimes Xpos, Ypos, Zpos;
Keytimes ThetaX, ThetaY, ThetaZ;

. . .

if( AnimationIsOn )
{
        // # msec into the cycle ( 0 - MSEC-1 ):
        int msec = glutGet( GLUT_ELAPSED_TIME )  %  MSEC;

        // turn that into a time in seconds:
        float nowTime = (float)msec  / 1000.;
        glPushMatrix( );
                glTranslatef( Xpos.GetValue( nowTime ), Ypos.GetValue( nowTime ), Zpos.GetValue( nowTime )
);
                glRotatef( ThetaX.GetValue( nowTime ),  1., 0., 0. );
                glRotatef( ThetaY.GetValue( nowTime ),  0., 1., 0. );
                glRotatef( ThetaZ.GetValue( nowTime ),  0., 0., 1. );
                << draw the object >>
        glPopMatrix( );
}
```

Number of msec in the animation cycle

Forward Kinematics:
Change Parameters – Connected Things Move
(All children understand this)

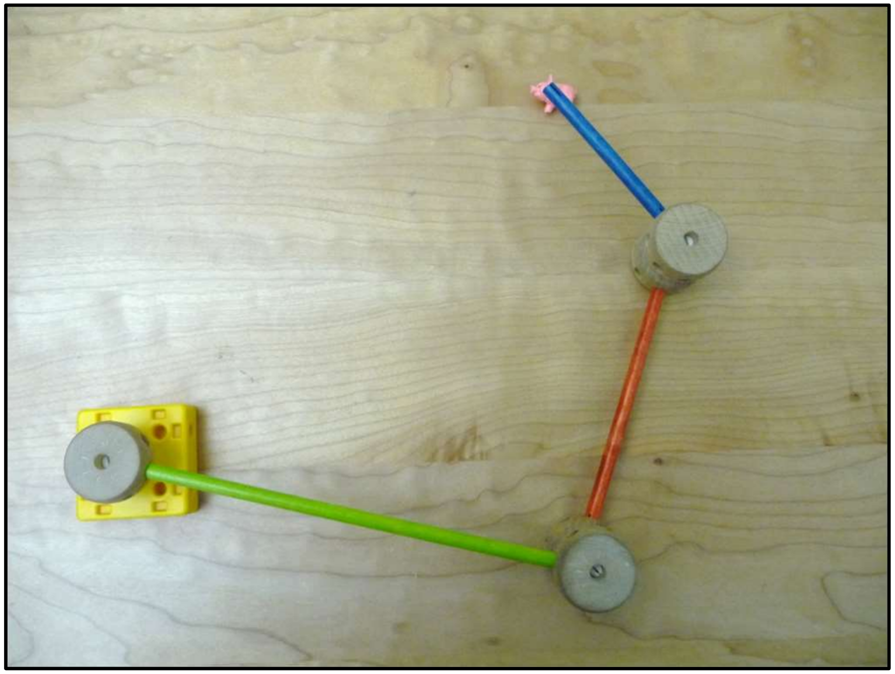


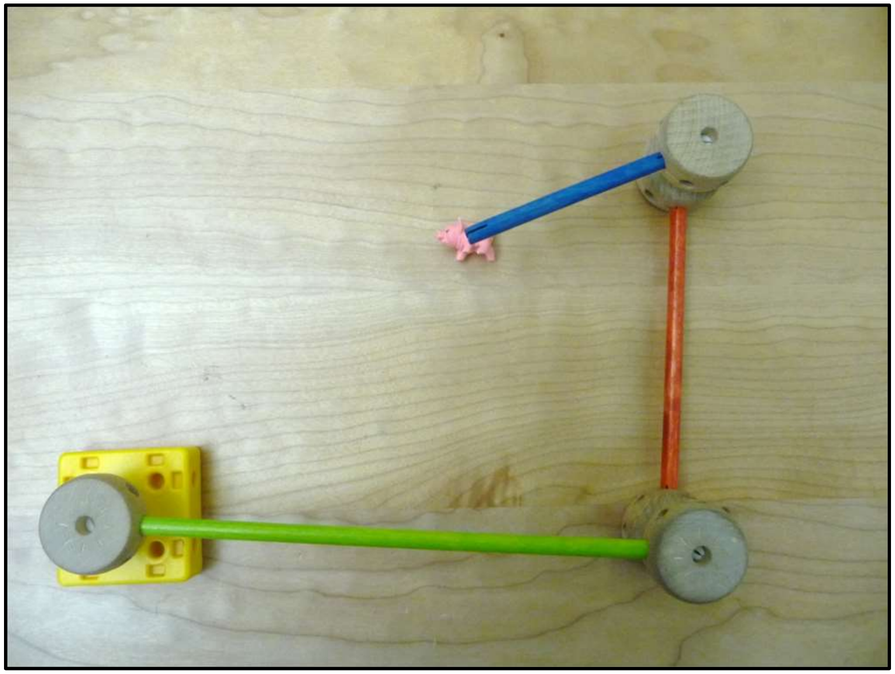
Forward Kinematics: Transformation Hierarchies



Ground

Inverse Kinematics (IK):
Things Need to Move to a Particular Location – What Parameters Will Make Them Do That?



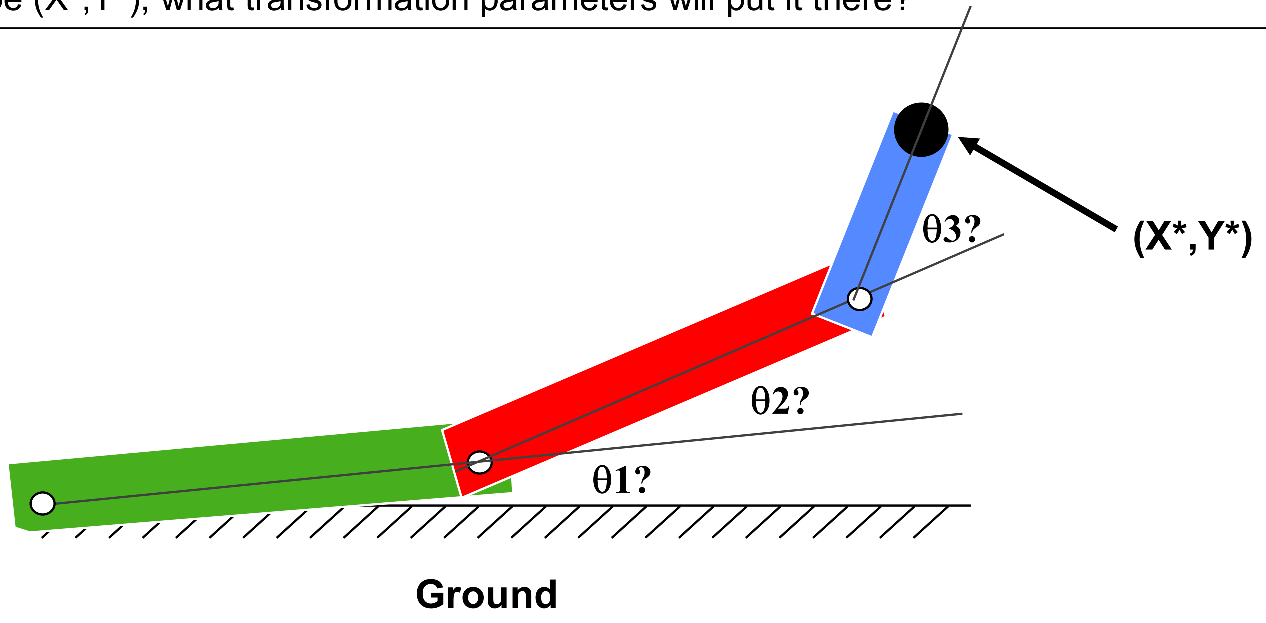


Of course, there will always be target locations that can *never* be reached.
Think about that spot in the middle of your back that you can never scratch! ☺

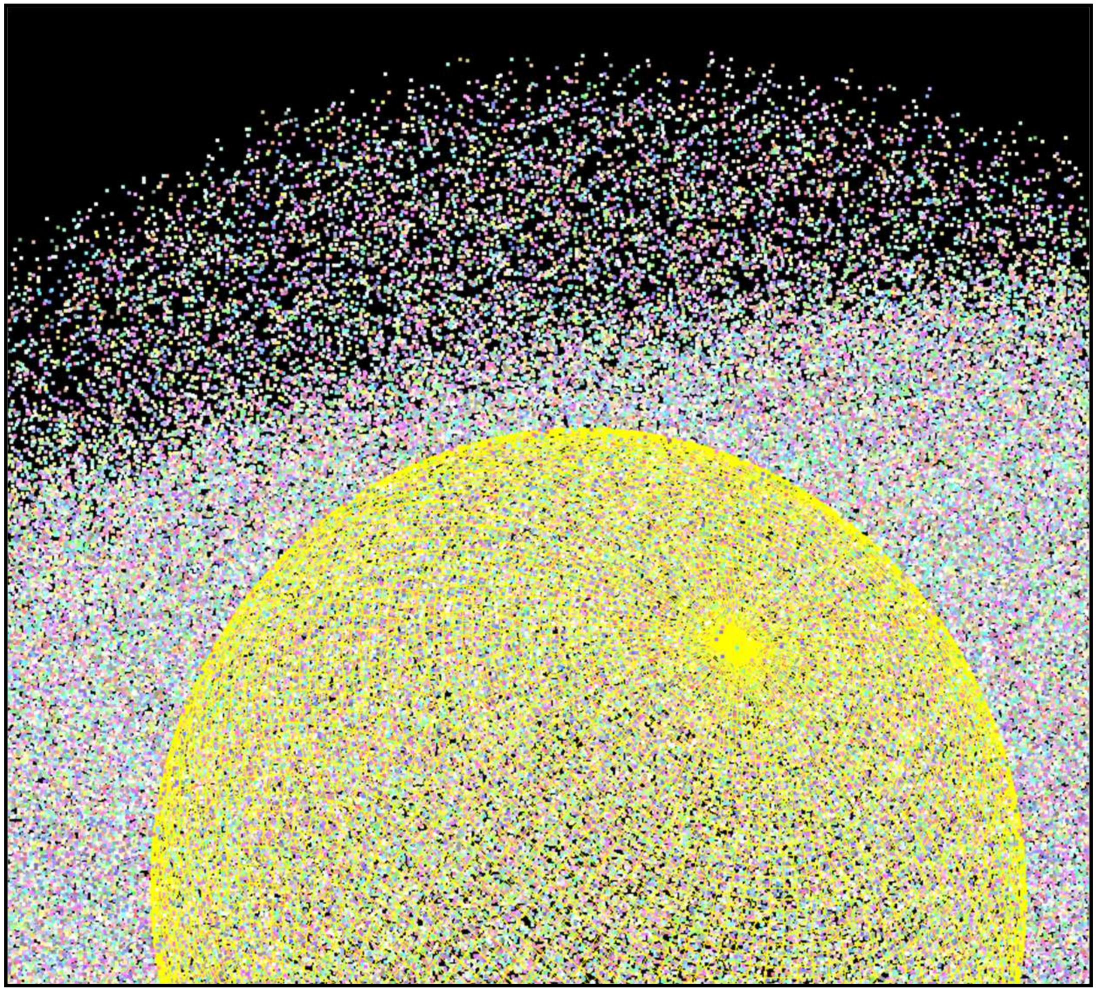
Inverse Kinematics (IK)

Forward Kinematics solves the problem “if I know the link transformation parameters, where are the links?”.

Inverse Kinematics (IK) solves the problem “If I know where I want the end of the chain to be (X*,Y*), what transformation parameters will put it there?”

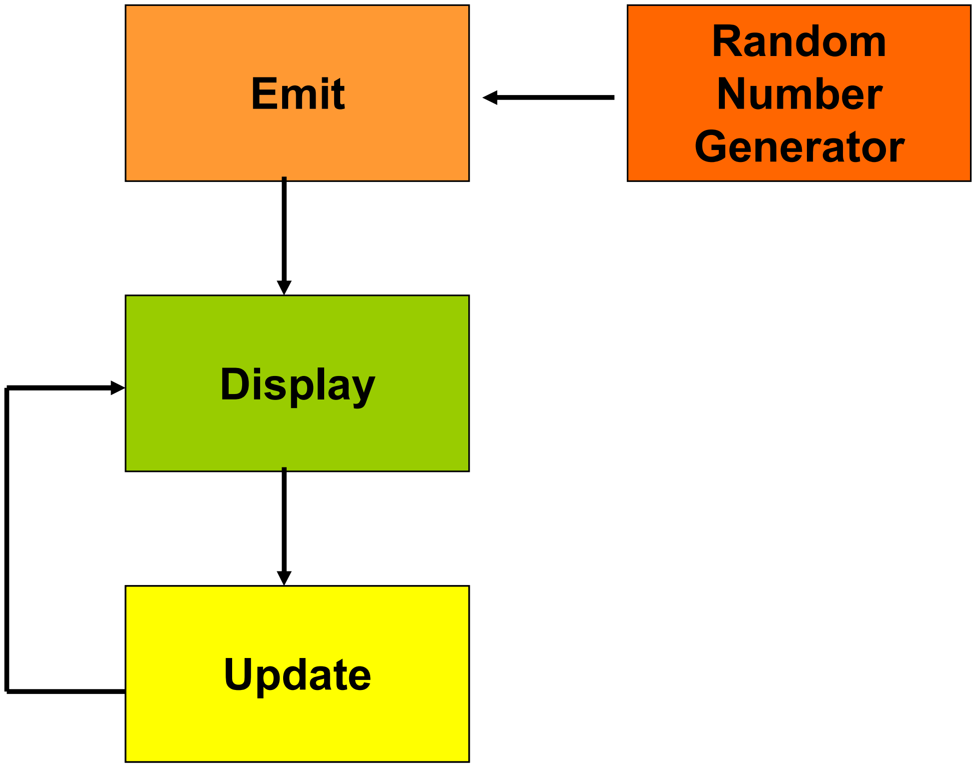


Particle Systems:
A Cross Between Modeling and Animation?

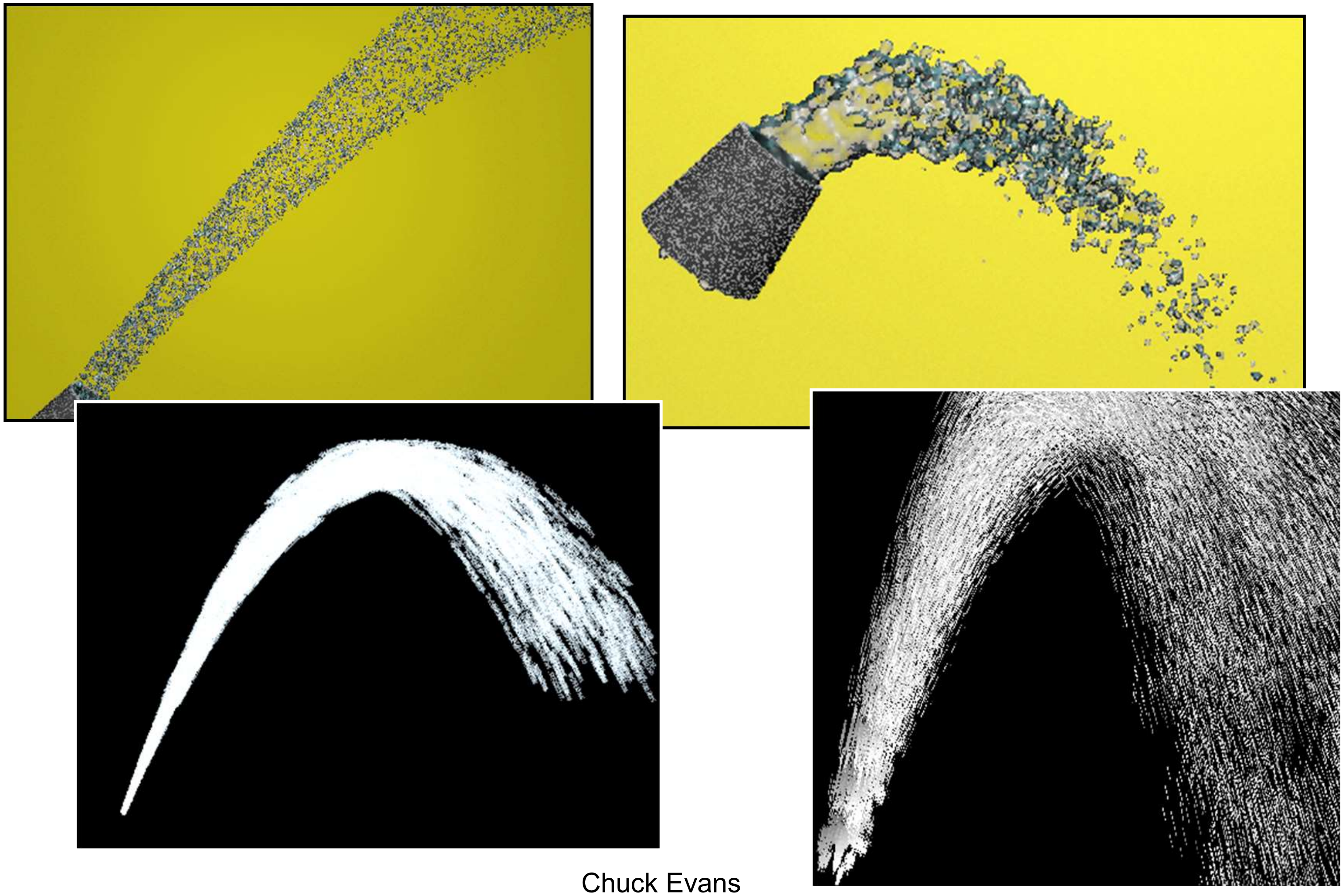


Particle Systems:
A Cross Between Modeling and Animation?

The basic process is:

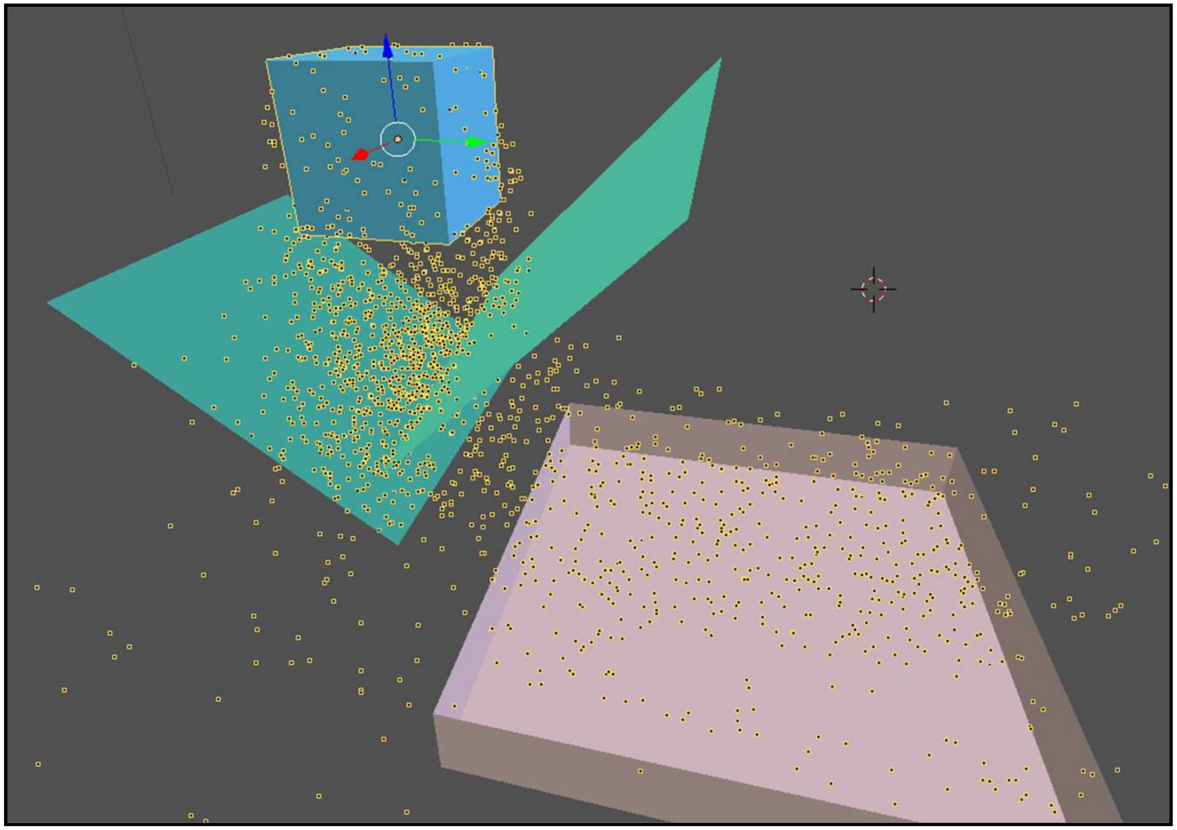


Particle Systems Examples



Chuck Evans

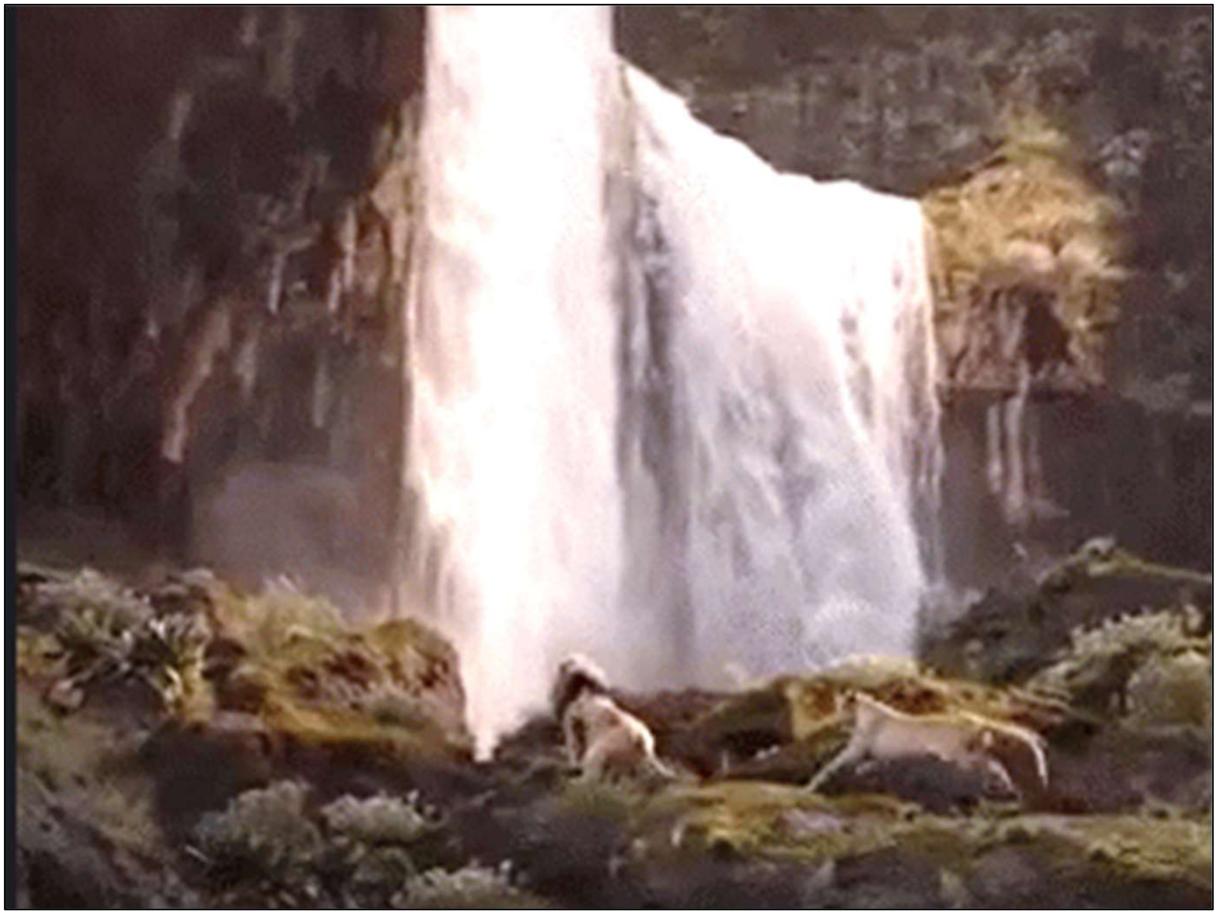
Particle Systems Examples



particles.mp4

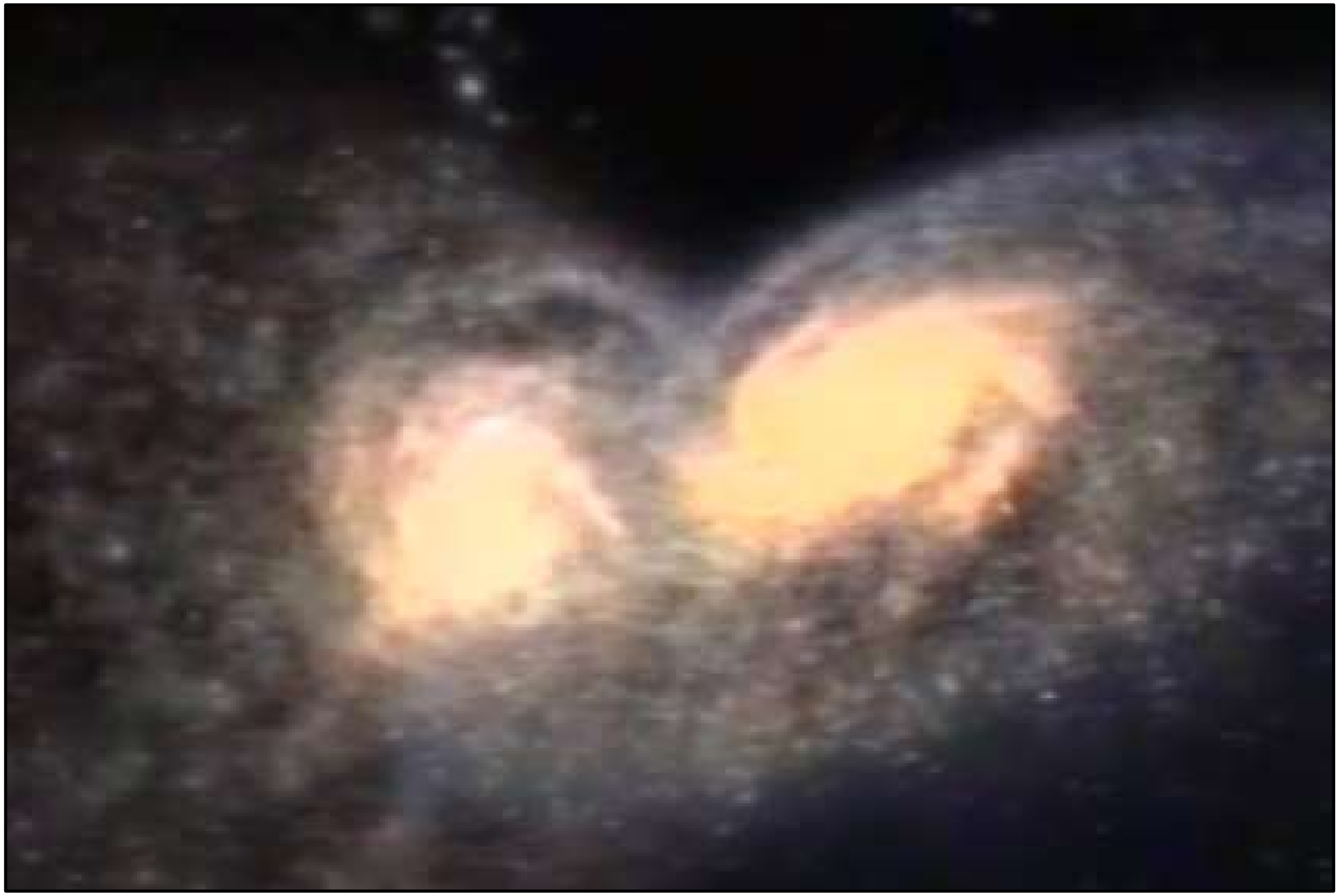
Particle Systems Examples

Particle Systems Examples



The Lion King (2019) -- Disney

A Particle System to Simulate Colliding Galaxies in *Cosmic Voyage*



Cosmic Voyage

Particles Don’t Actually Have to Be “Particles”

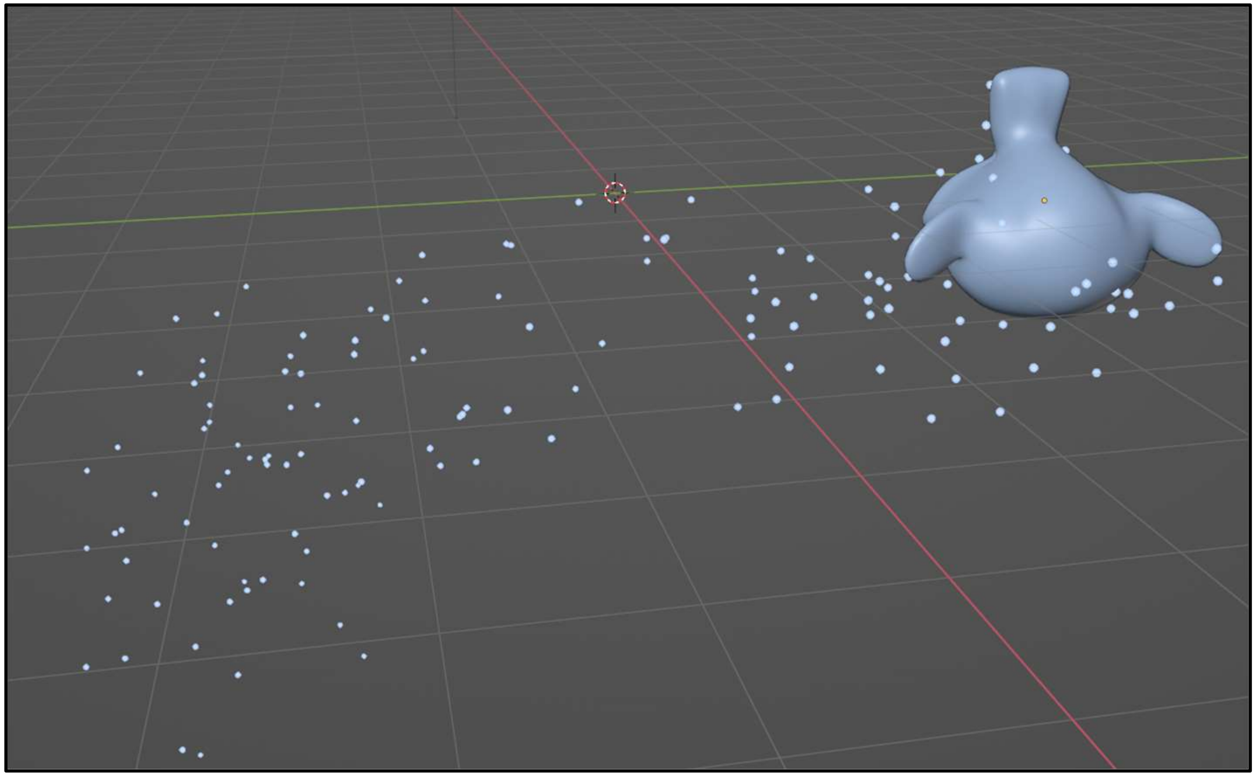
Avatar

The Lion King

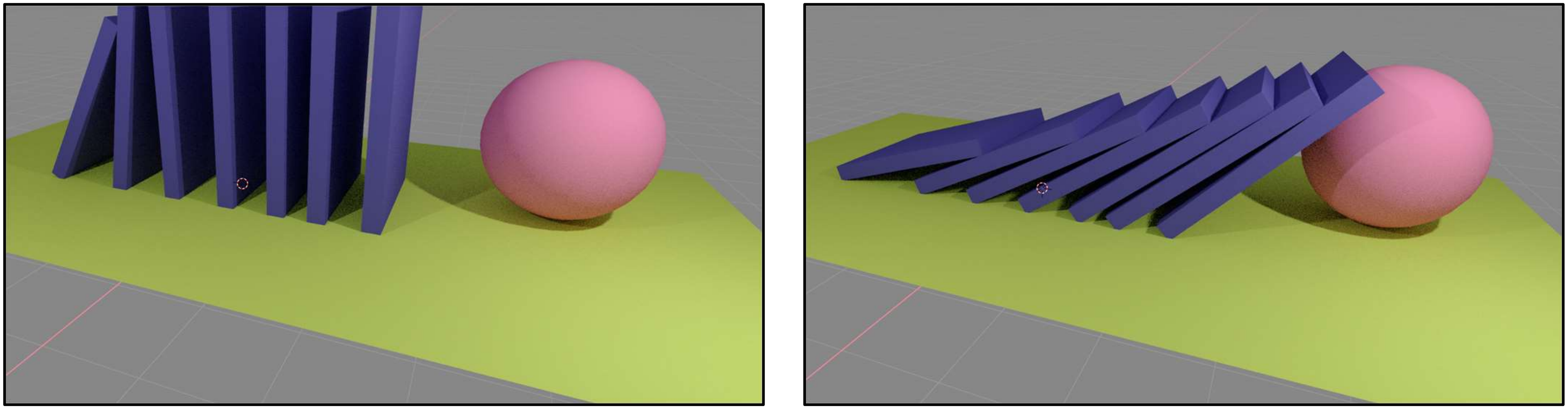
Mulan

Multiple Animation Techniques Can Be Combined

A Particle System coming from a moving keyframed object in Blender:



Animating using Rigid-body Physics



Newton’s second law:

force = mass * acceleration

or

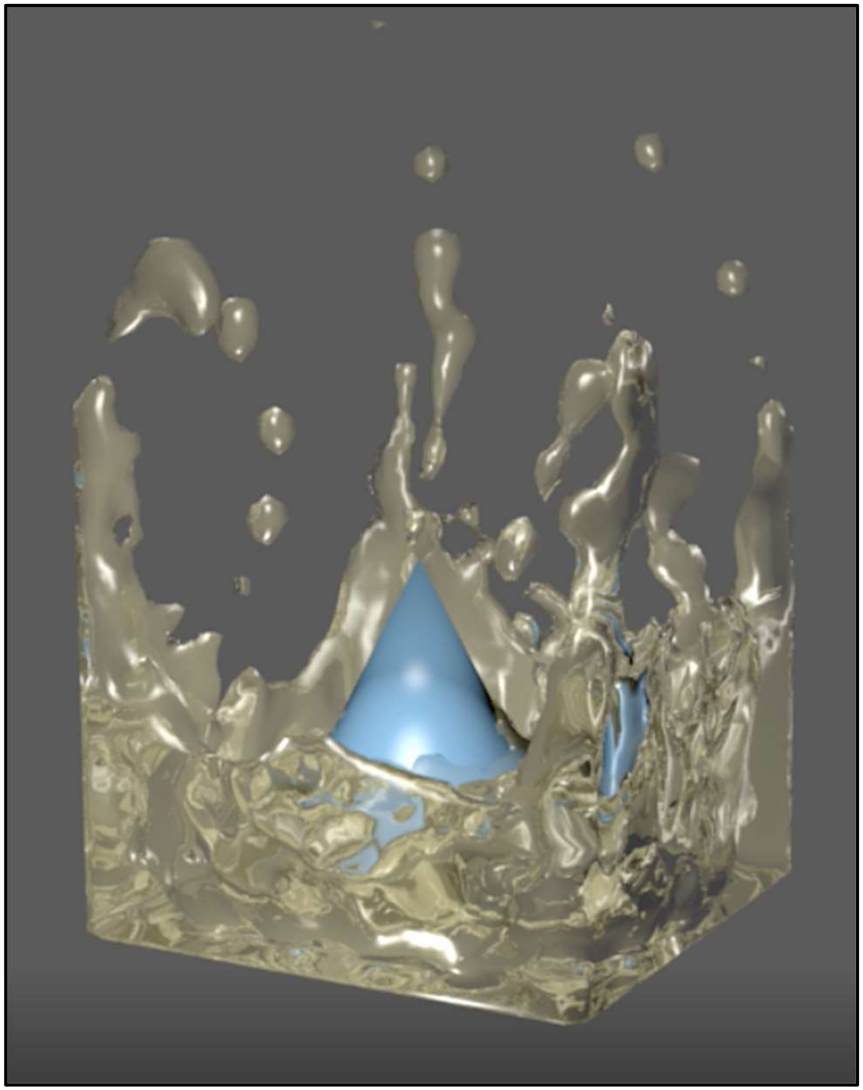
$\ddot{x}$ = acceleration = force / mass

$$x(T) = \iint_{t=0}^{t=T} \ddot{x}\,dt \approx \sum \quad \sum \ddot{x}\,\Delta t$$

dominos.mp4

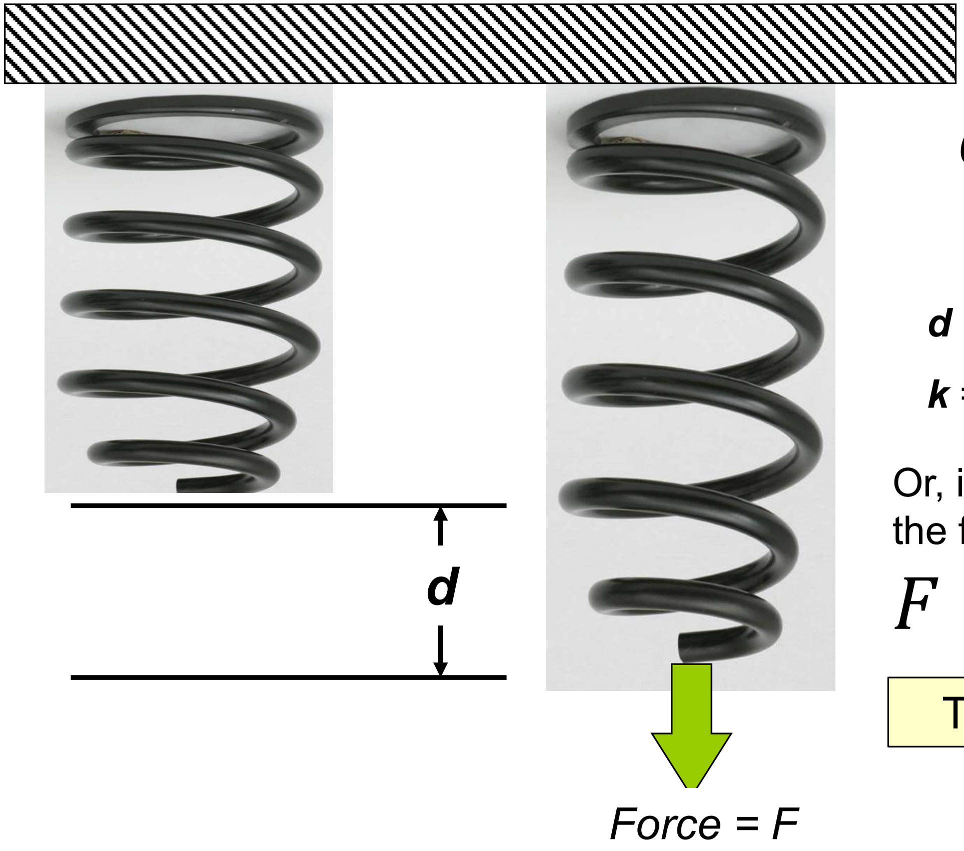
In order to make this work, you need to supply physical properties such as mass, center of mass, moment of inertia, coefficients of friction, coefficients of restitution, etc.

Animating using Fluid Physics



fluid.avi

Animating using Physics



$$d = \frac{-F}{k}$$

d = ***spring displacement*** in inches

k = ***spring stiffness*** in pounds/inch

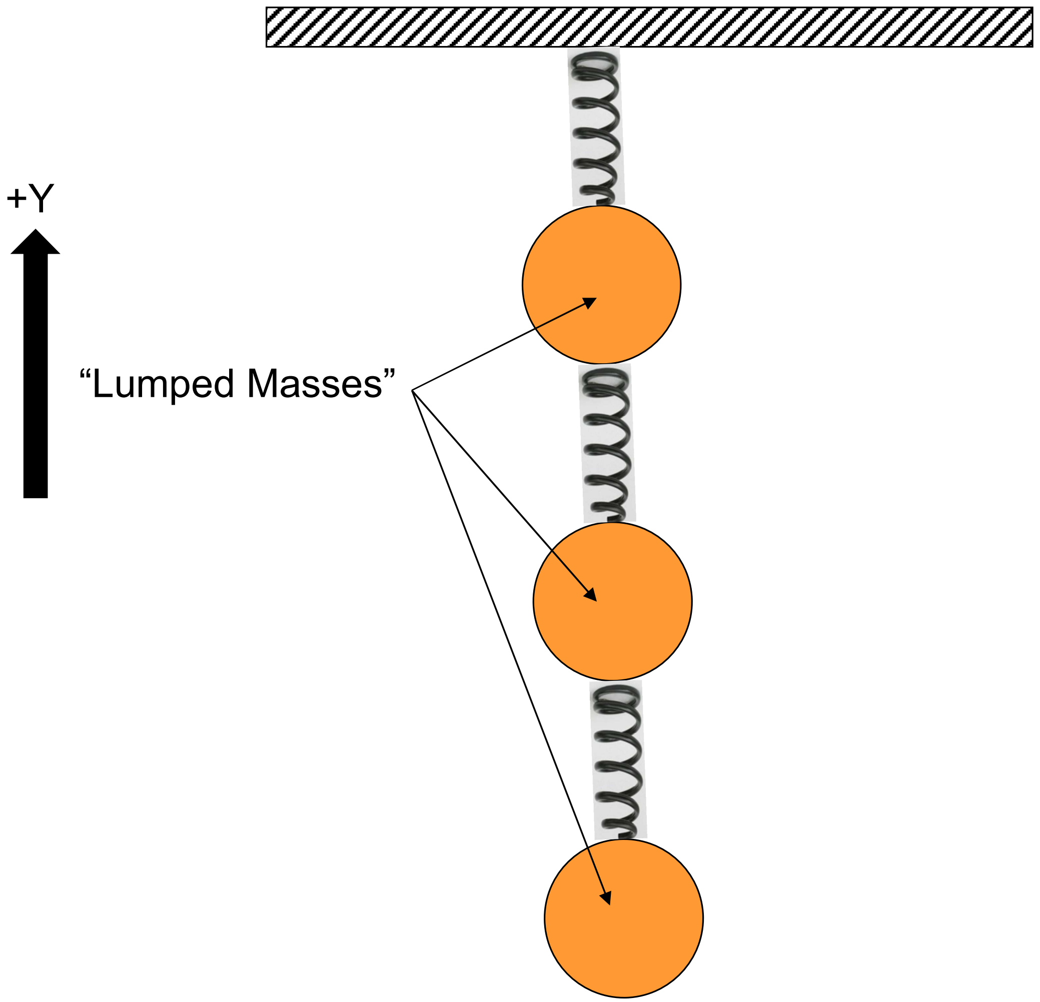
Or, if you know the displacement,
the force exerted by the spring is:

$$F = -kd$$

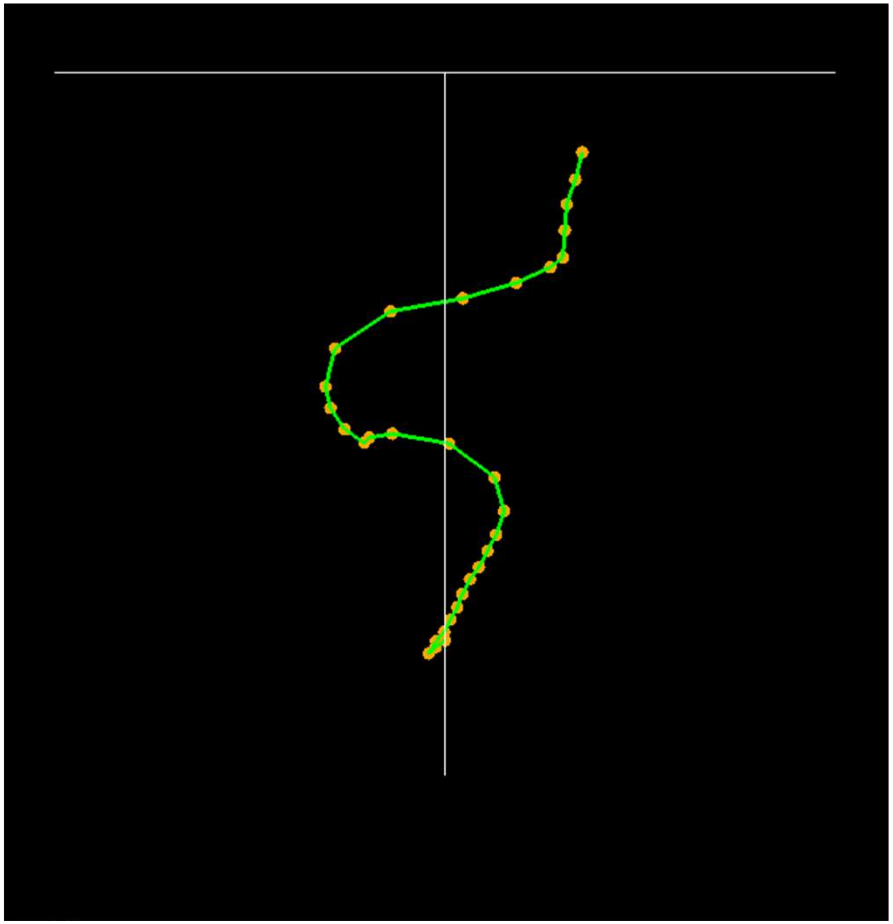
This is known as **Hooke's Law**

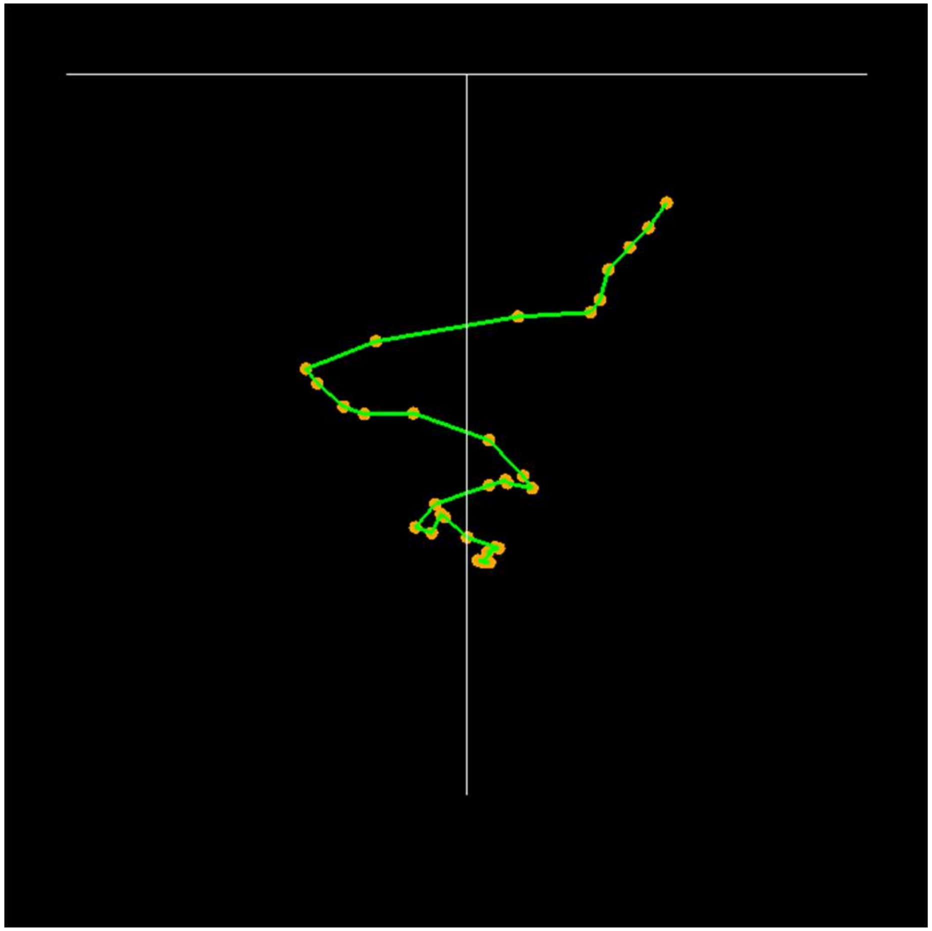
Why the minus sign? Because a spring's force acts in a direction opposite the displacement. In other words, a spring's force tries to correct its displacement.

Animating using the Physics of a Mesh of Springs

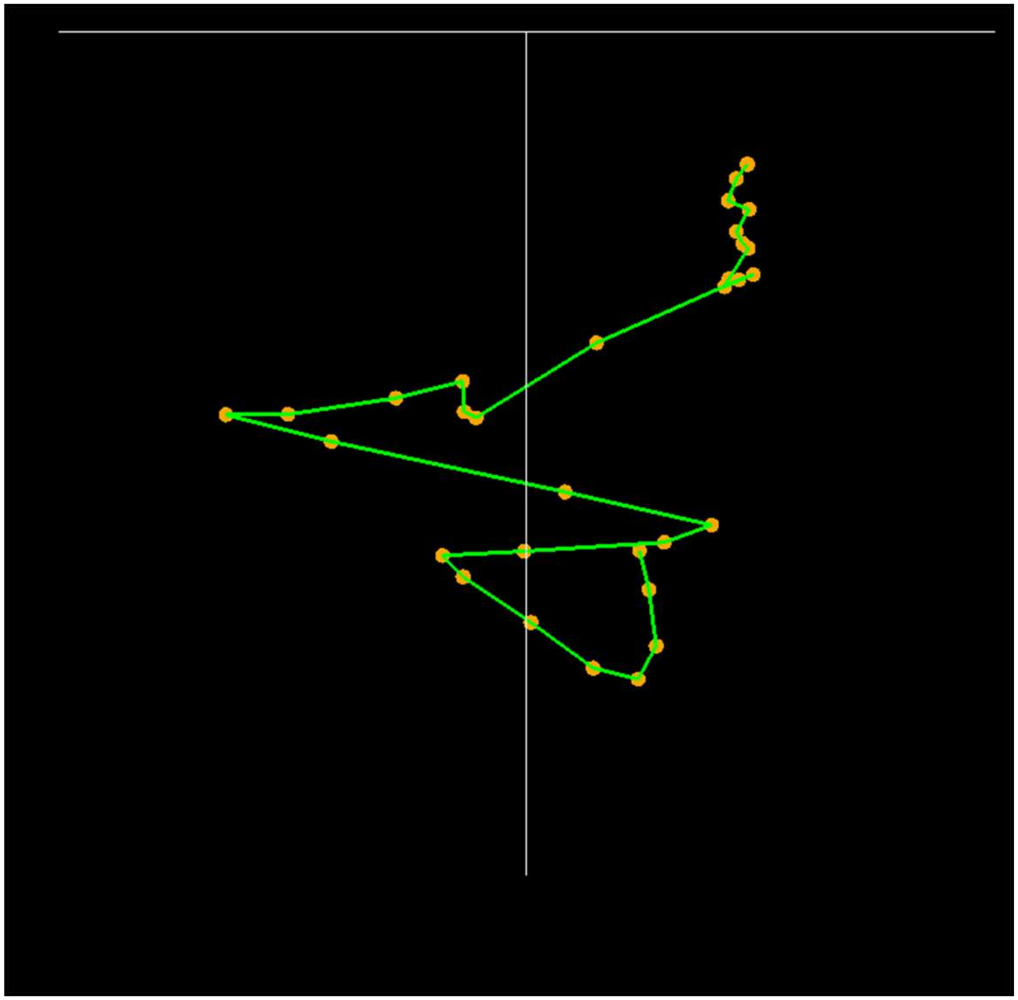


Simulating a Bouncy String or a Chain



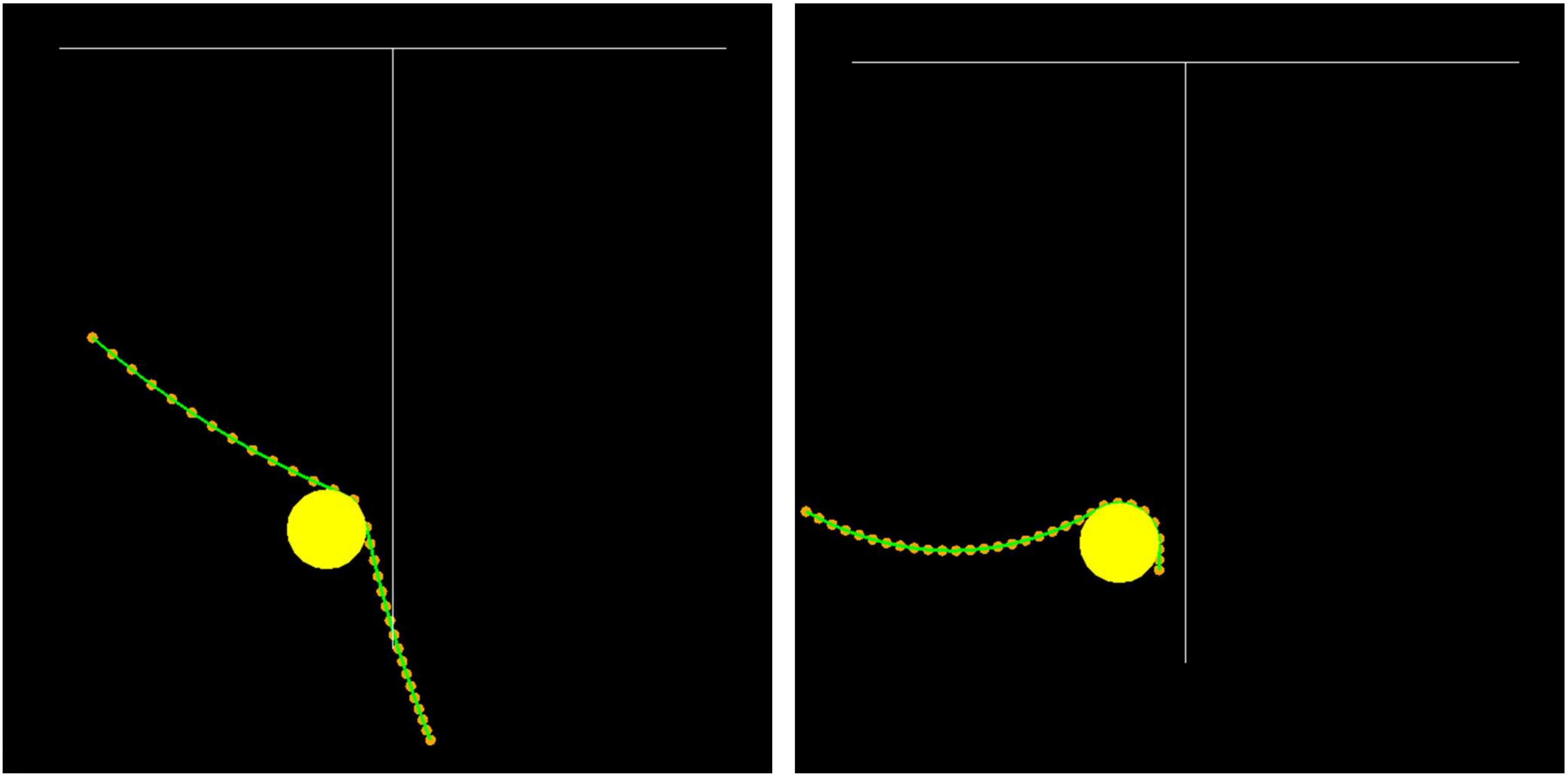


Simulating a Bouncy String or a Chain

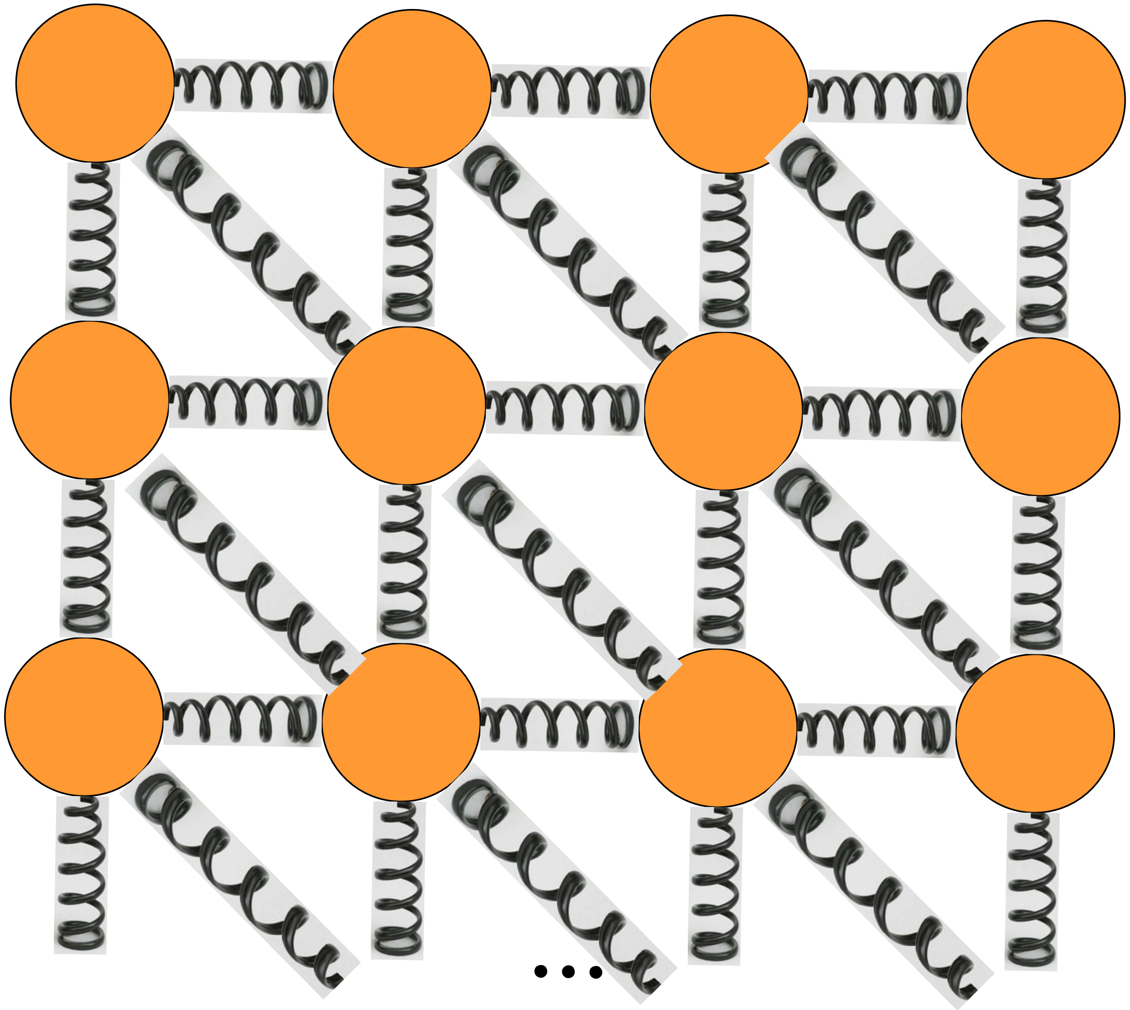


string.mp4

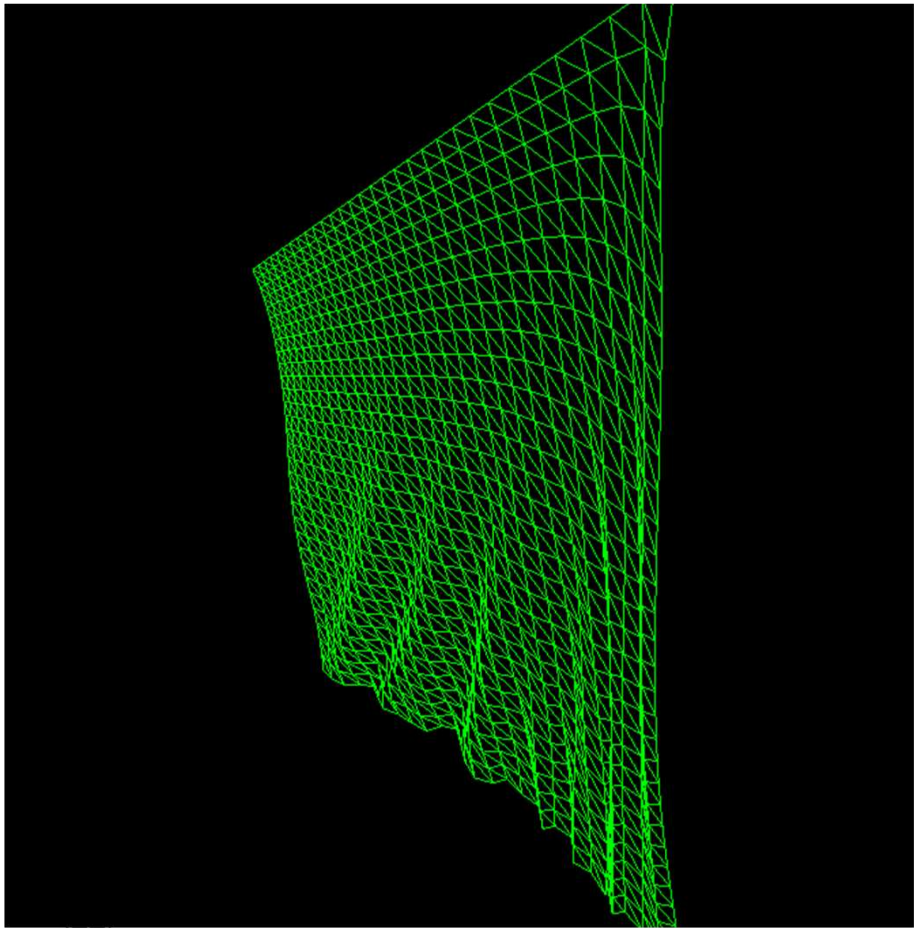
Placing a Physical Barrier in the Scene

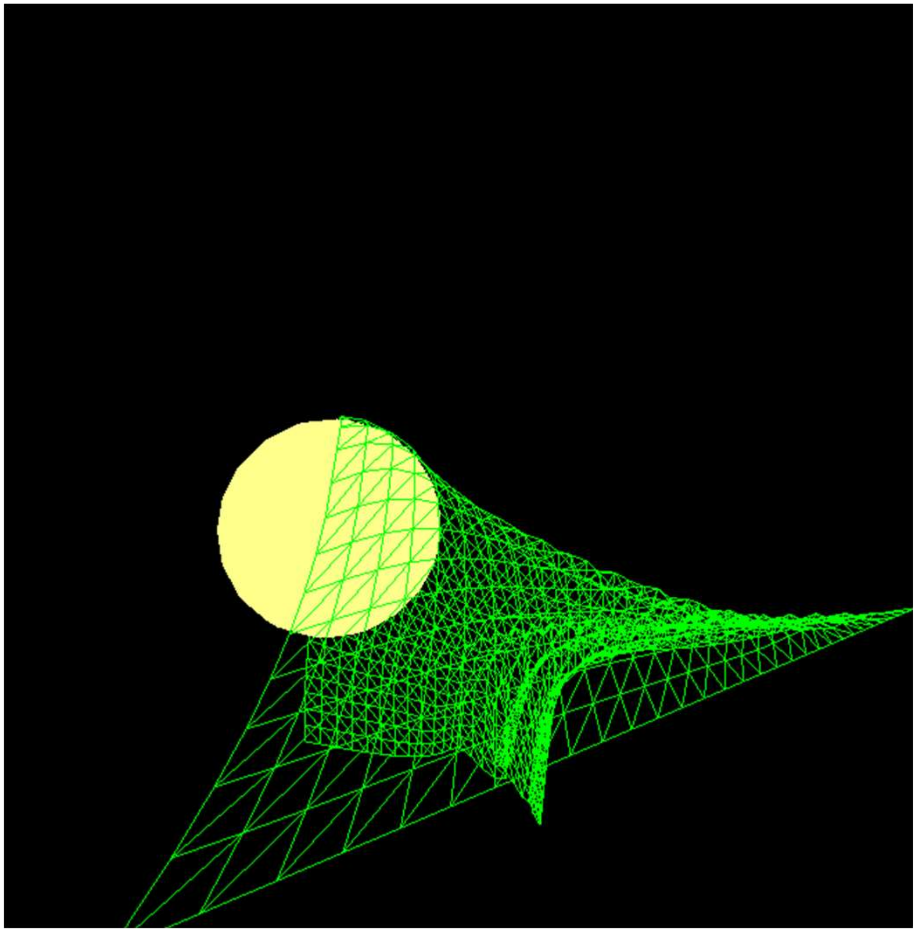


Animating Cloth

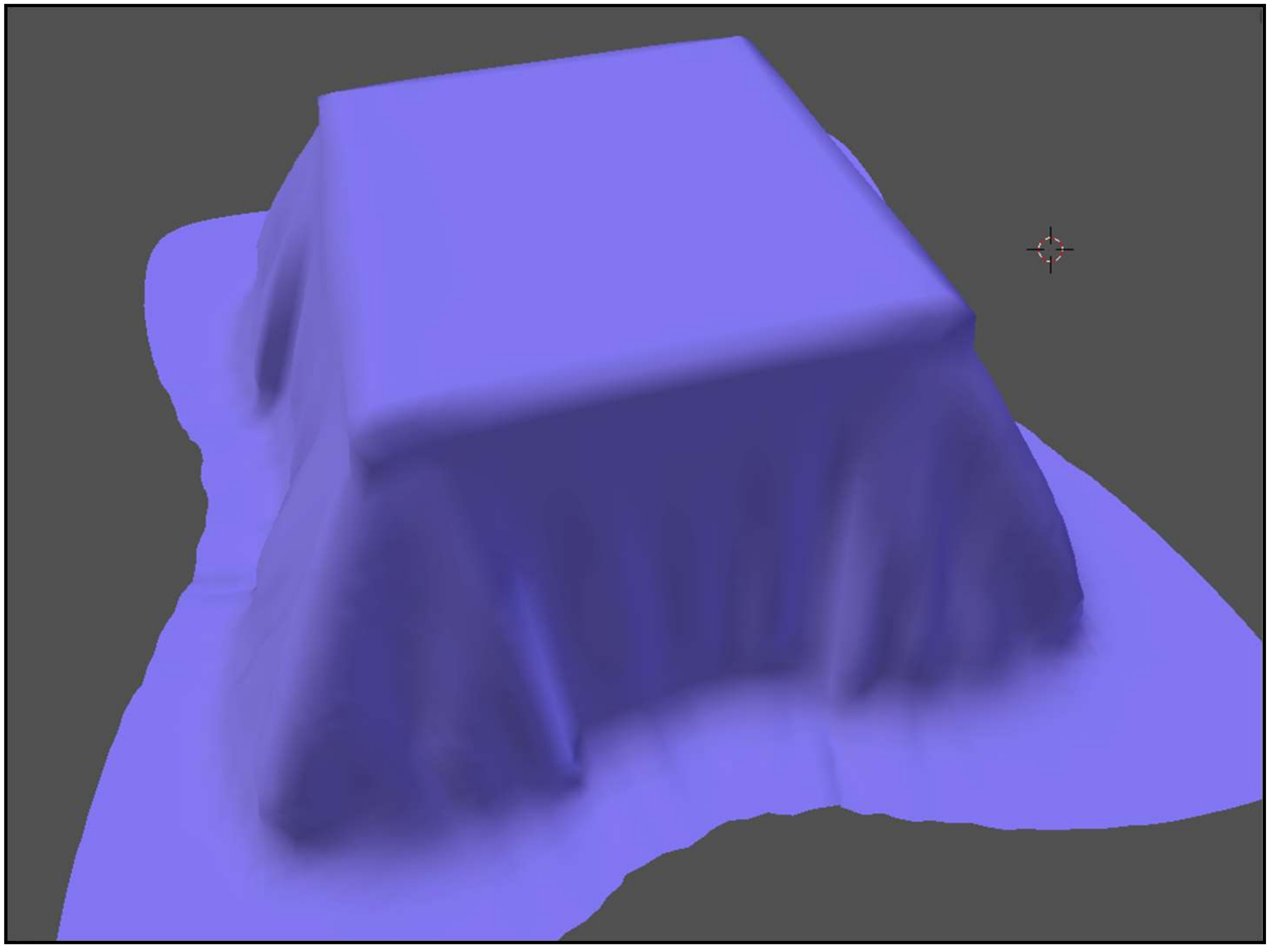


Cloth Examples





Cloth Example



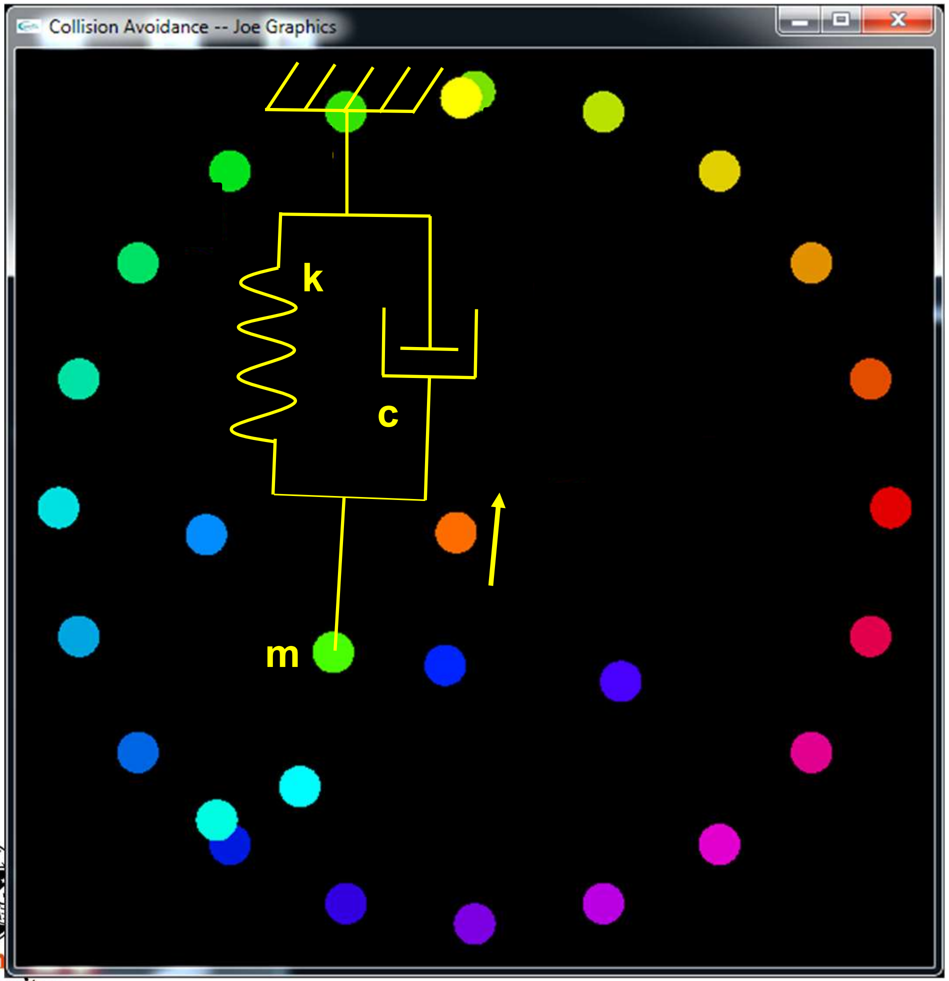
cloth.mp4

Cloth Example

PIXAR SHORTS

Geri's Game, Pixar

Functional Animation:
Setup Imaginary Physics to Make the Object *Want* to Move Towards a Goal Position



Newton's second law:

force = mass * acceleration

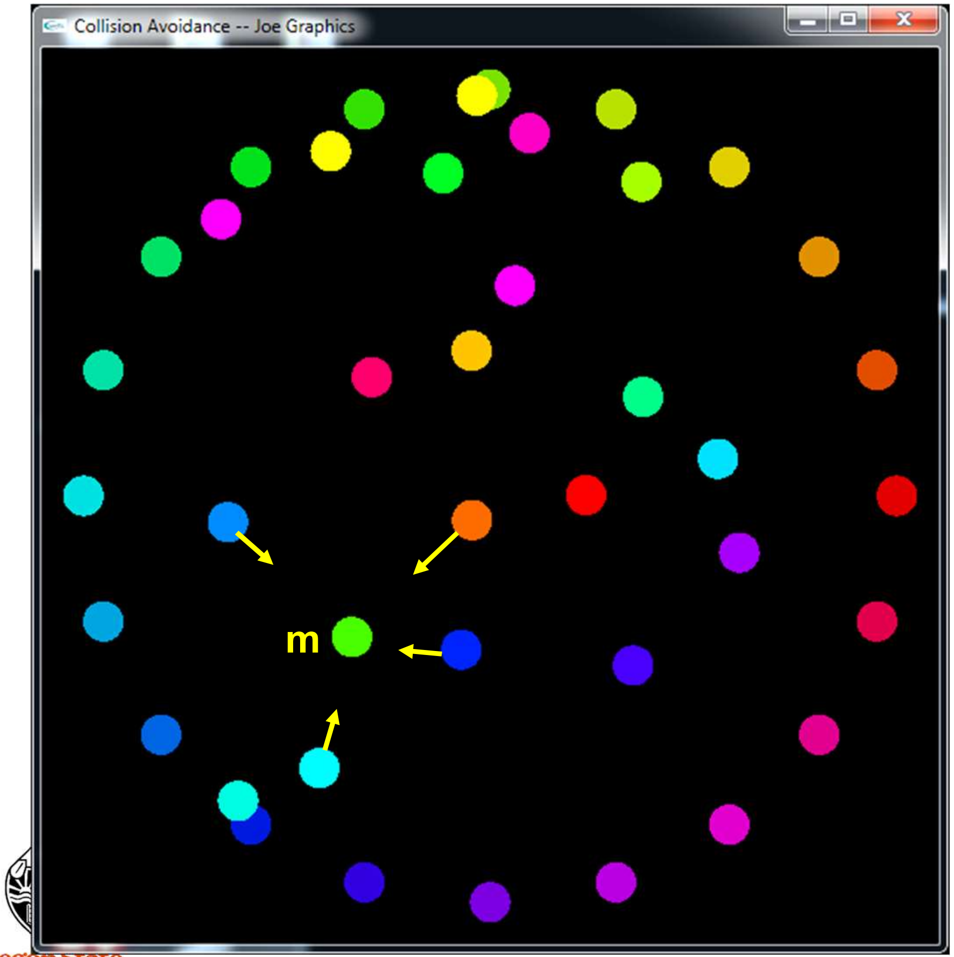
or

$\ddot{x} =$ acceleration = force / mass

$$\ddot{x} = \frac{-c\dot{x} - kx}{m}$$

- The ***k*** is an imaginary spring.
- The ***c*** is an imaginary "damper", which you can think of as a shock absorber which absorbs energy. (Your car has these to soften bouncing.)

Functional Animation:
Setup Imaginary Physics to Make the Object *Want* to Move Away from all other Objects



Newton's second law:

force = mass * acceleration

or

$\ddot{x} =$ acceleration = force / mass

$$\ddot{x} = \frac{F_{repulsive}}{m}$$

Repulsion Coefficient

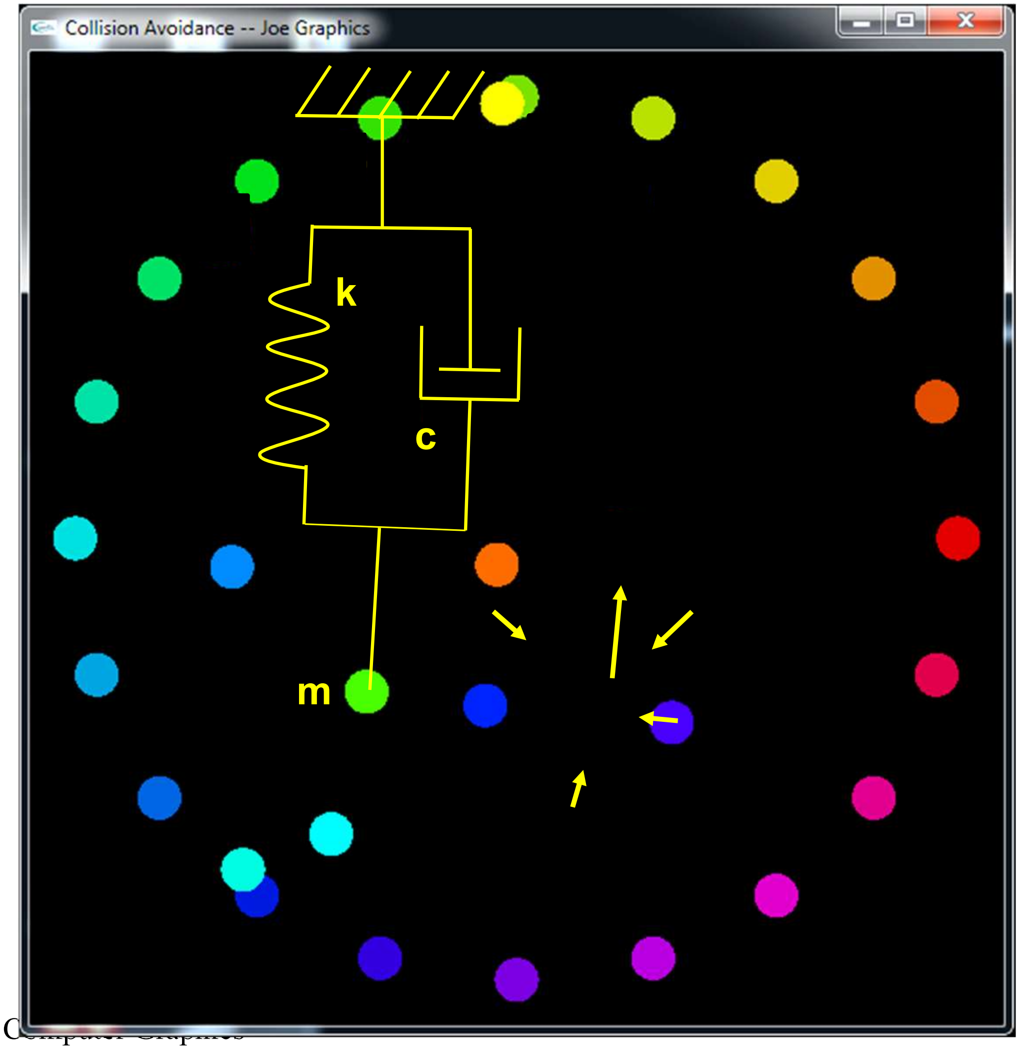
$$F_{repulsive} = \frac{C_{repulse}}{d^{Power}}$$

Distance between the boundaries of the 2 bodies

Repulsion Exponent

This isn't from a book. I just made this up. It seemed like a good idea.

Total Goal – Make the Free Body Move Towards its Final Position While Being Repelled by the Other Bodies



Newton’s second law:

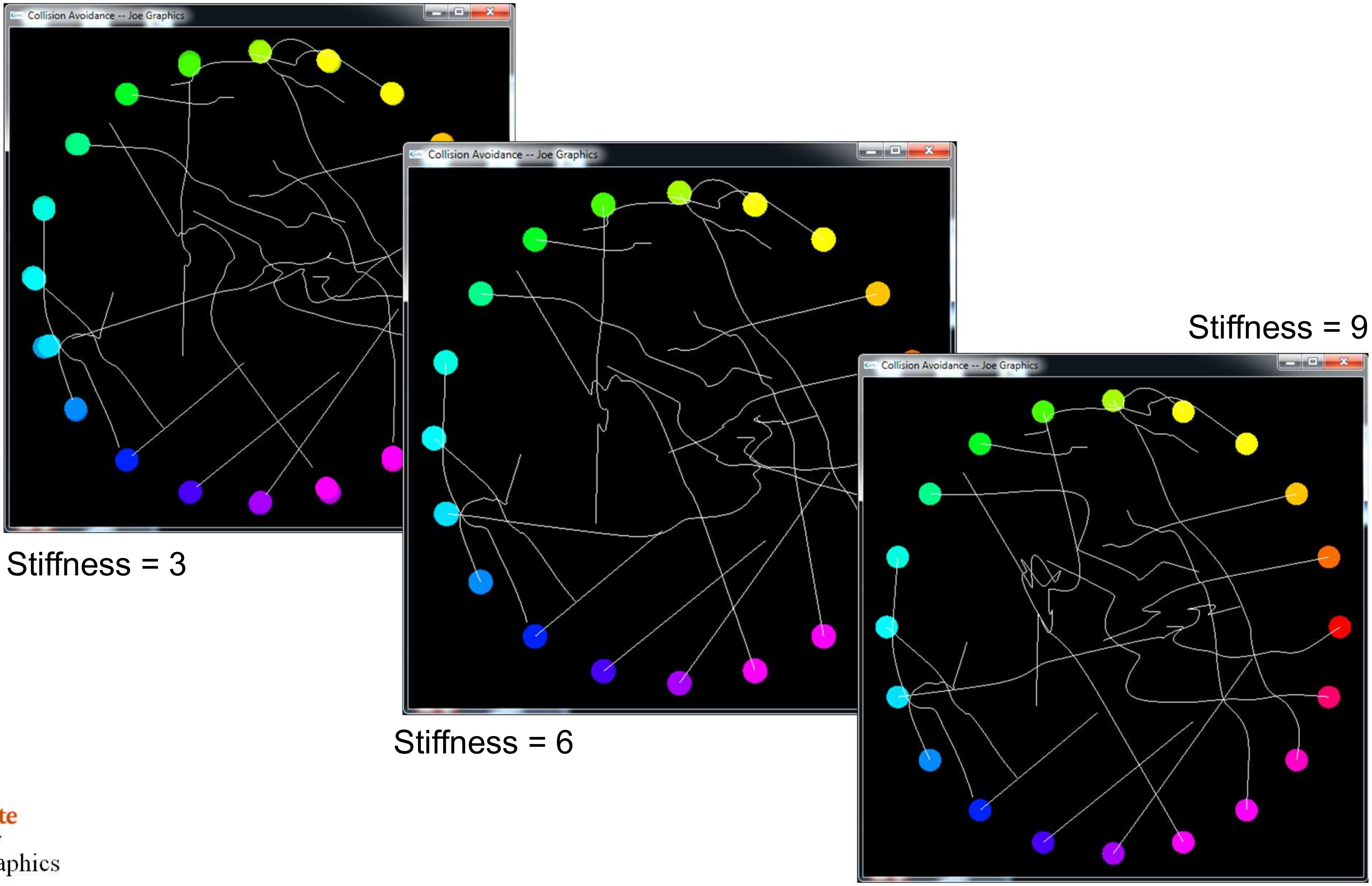
force = mass * acceleration

or

$\ddot{x}$ = acceleration = force / mass

$$\ddot{x} = \frac{-c\dot{x} - kx + \sum F_{repulsive}}{m}$$

Increasing the Stiffness

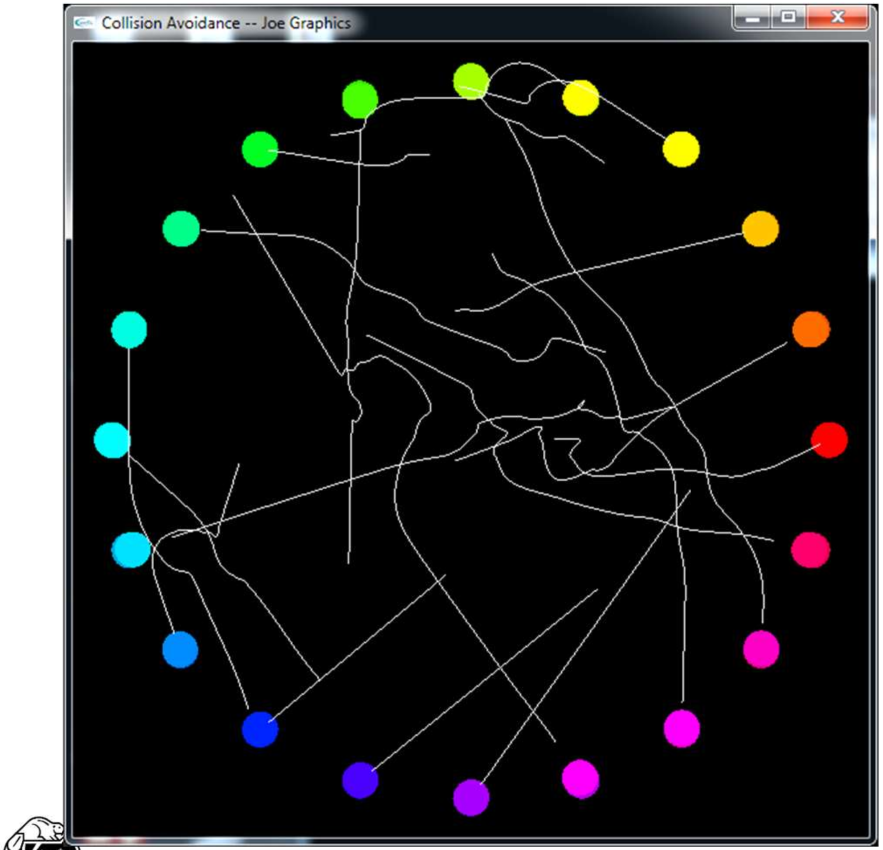

Collision Avoidance -- Joe Graphics
Collision Avoidance -- Joe Graphics
Collision Avoidance -- Joe Graphics

Stiffness = 3

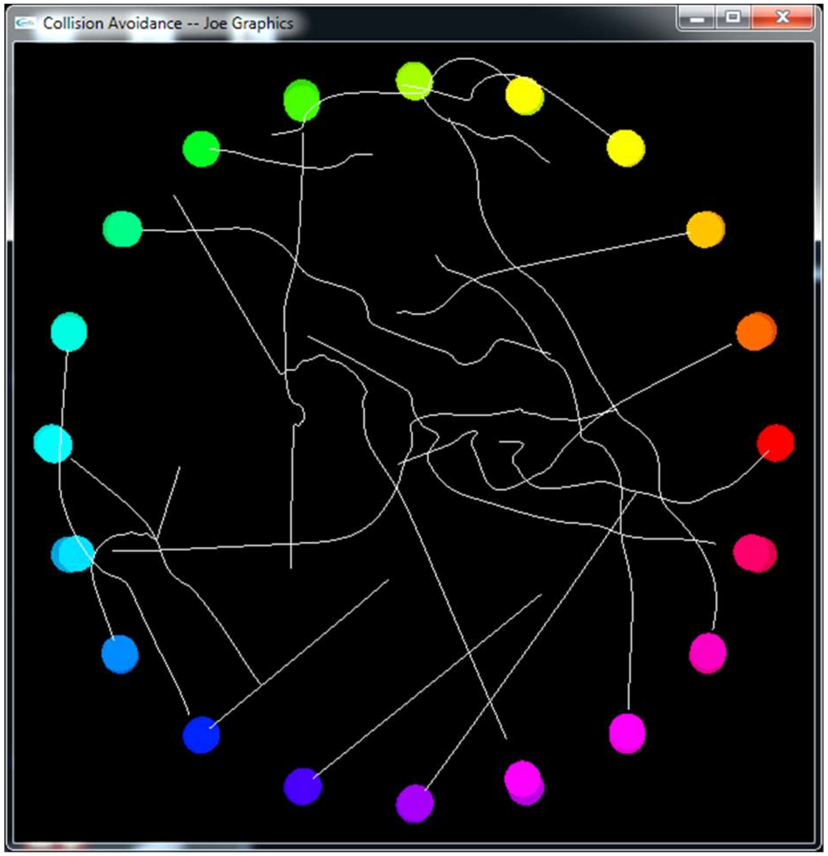
Stiffness = 6

Stiffness = 9

Increasing the Repulsion Coefficient

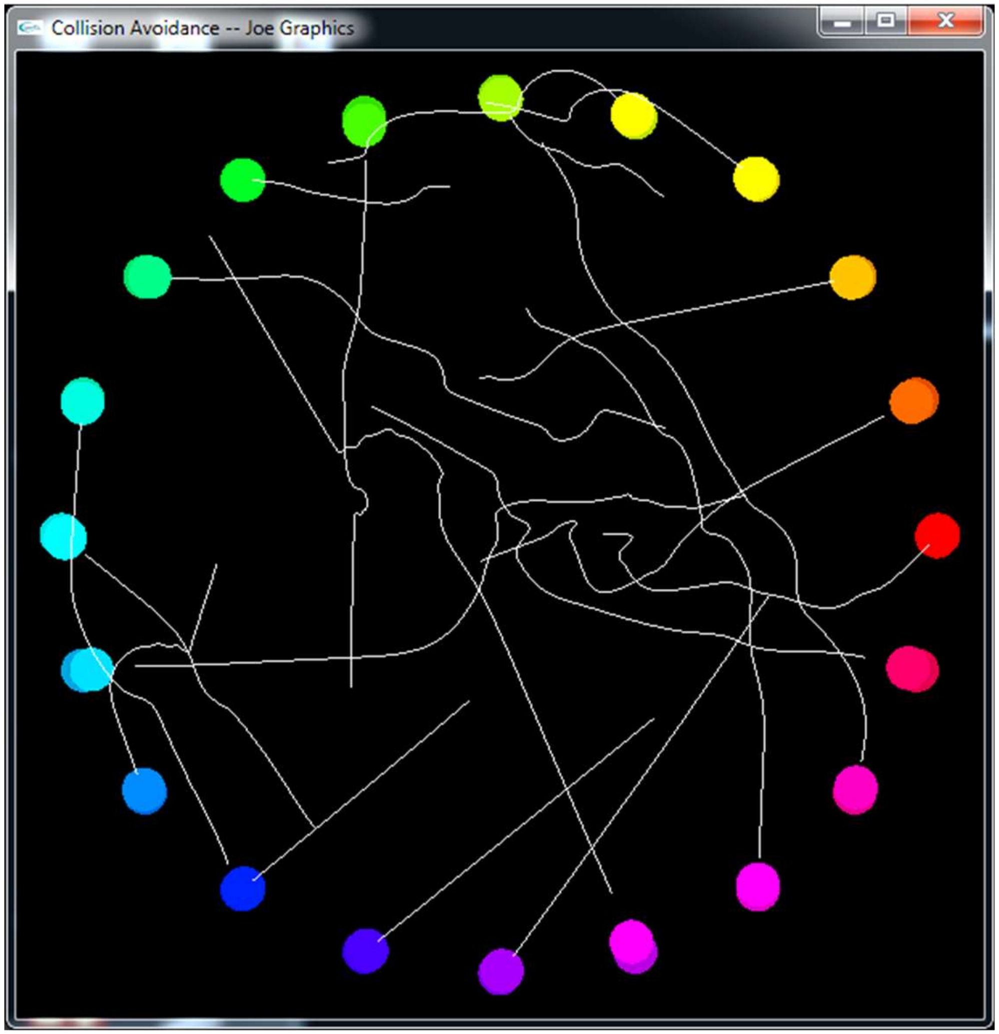


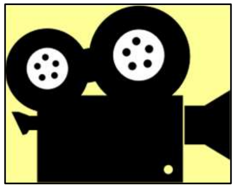
Repulse = 10



Repulse = 30

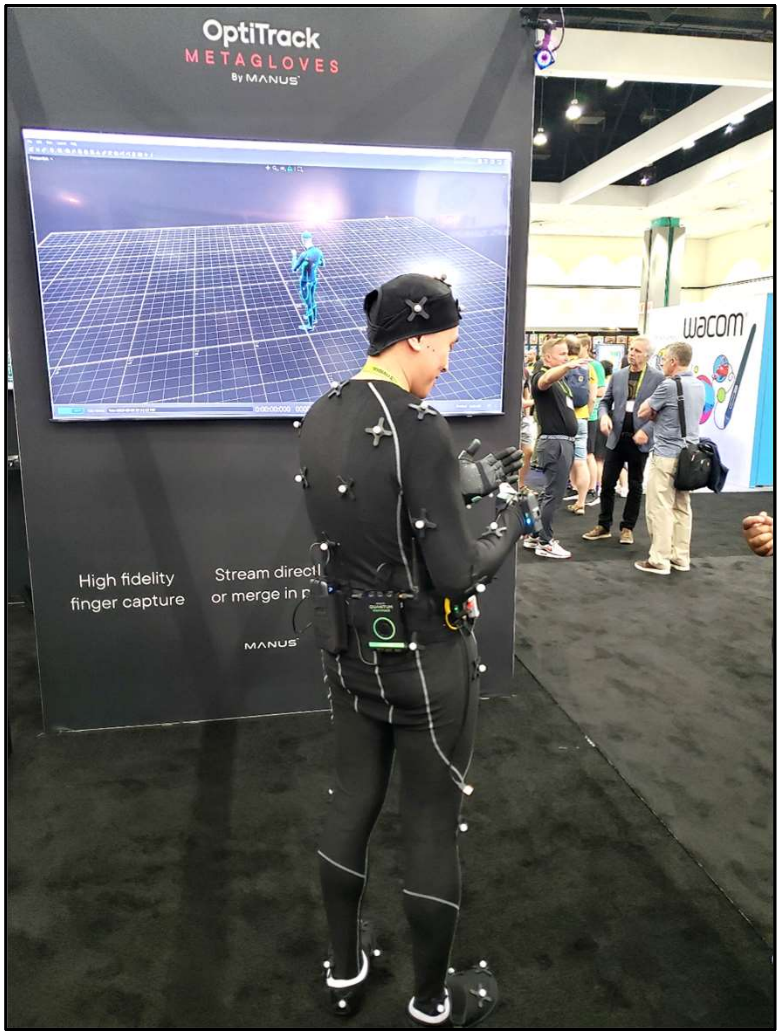
Functional Animation:
Setup Imaginary Physics to Make the Object *Want* to Move Like We Want it To

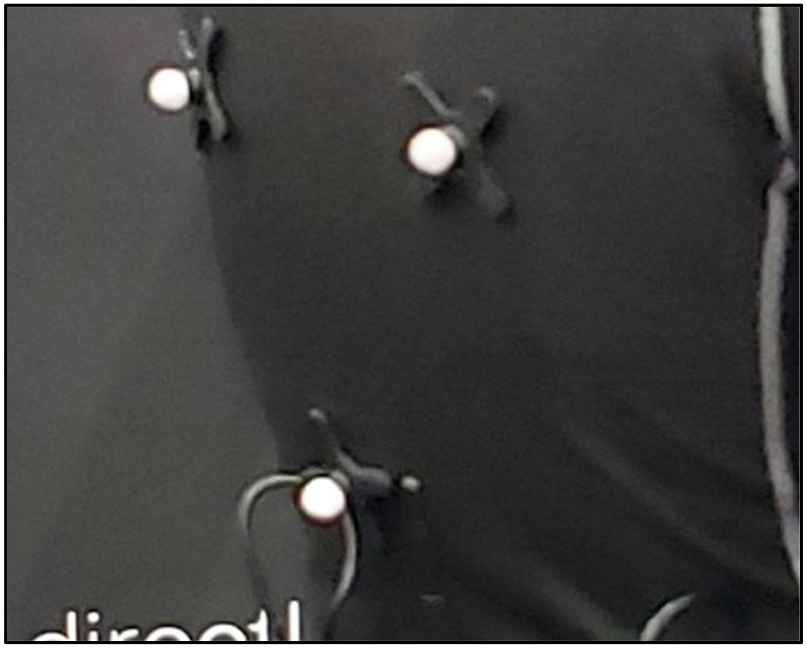




avoid.mp4

Motion Capture (“MoCap”) as an Input for Animation





Natural Point

Motion Capture is for Hands and Faces Too

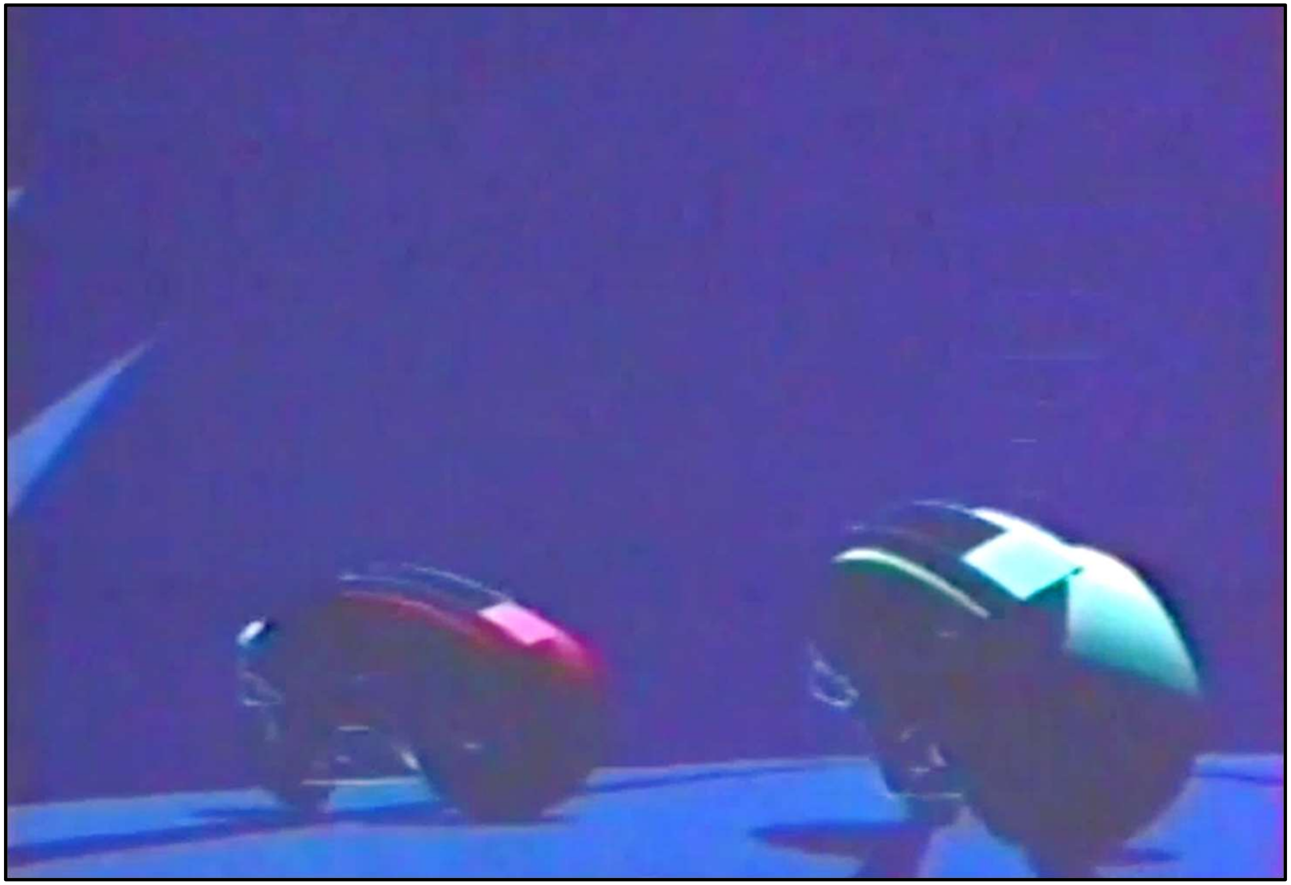
Natural Point

Even Animals can be MoCapped

My cats would never have put up with this…

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

Tron I –
They probably should have used physics, but didn't

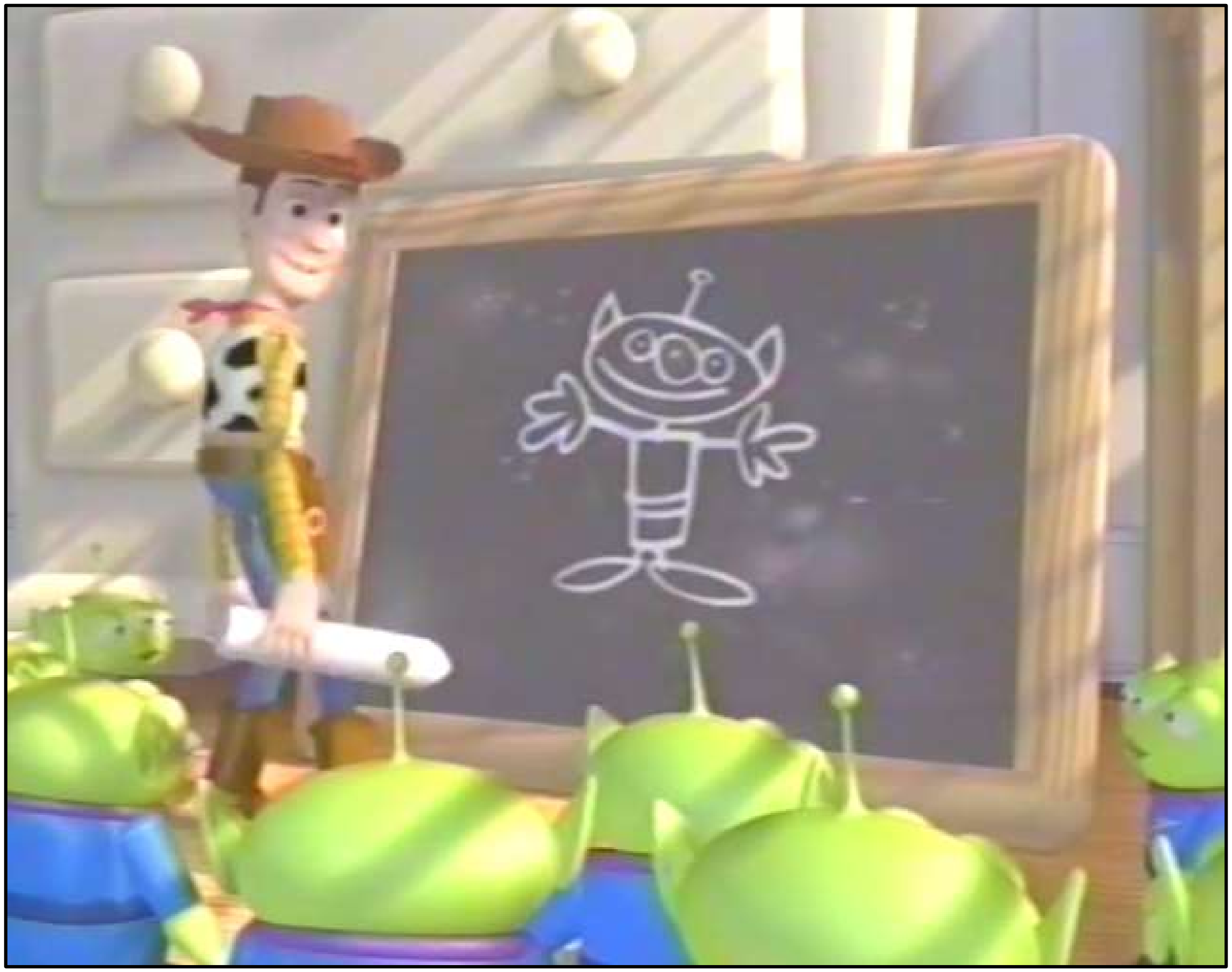


MAGI

Card Trick

Rob Russ

Pixar Animated Shorts



Pixar