15-104 Introduction to Computing for Creative Practice

Fall 2020

21 Springs (Physics of Motion)

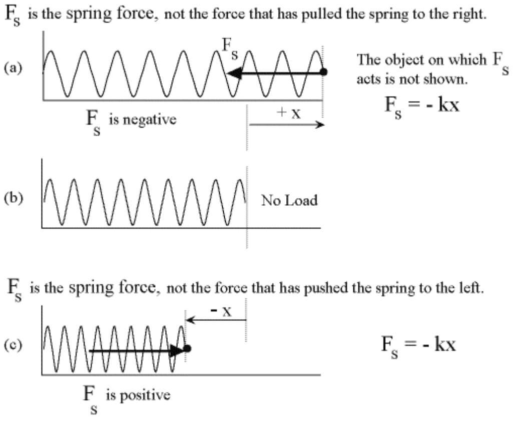
Instructor: Tom Cortina, tcortina@cs.cmu.edu, GHC 4117, 412-268-3514

The Universe Consists of Springs

- Nearly every solid material can be thought of as a series of particles which are attached to each other by elastic forces.
- From Isaac Newton (1686) we know his second law of motion, that $F = ma$. This states that in general, force is proportional to mass times acceleration.
- From Robert Hooke (1678) we know the spring law, $F = -kx$. This states that the force applied by a spring, is proportional (by some constant k) to its distention x (amount of stretch or compression). The minus sign tells us that the spring's "restorative force" happens in the opposite direction.

Example

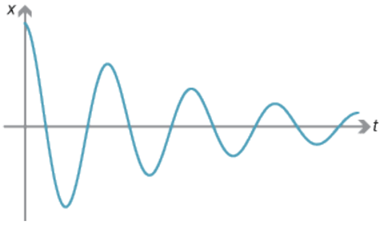
- If you stretch a spring (which is rigidly fixed at one end) one centimeter to the east, it will apply a restoring force towards the west.
- If you stretch it two centimeters eastward, it will apply a westerly force which is twice as strong.


F_s is the spring force, not the force that has pulled the spring to the right.
(a)
F_s
The object on which F_s acts is not shown.
$F_s = -kx$
F_s is negative
+ x
(b)
No Load
F_s is the spring force, not the force that has pushed the spring to the left.
- x
(c)
$F_s = -kx$
F_s is positive

from pstcc.edu

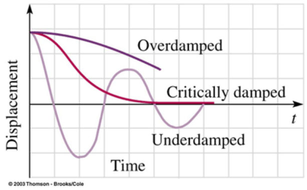
Damped Harmonic Motion

- Because $F = ma$ and $F = -kx$, we can derive that $ma = -kx$.
- Interestingly, this is actually a "second-order differential equation" — which describes a relationship between a particle's position, x, and its acceleration (or the rate of change of the rate of change of its position), a.
- The solution to such equations always take the form of an "exponentially damped cosinusoid", in what is called **damped harmonic motion**:



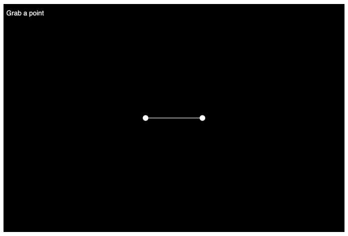
The spring constant (k)

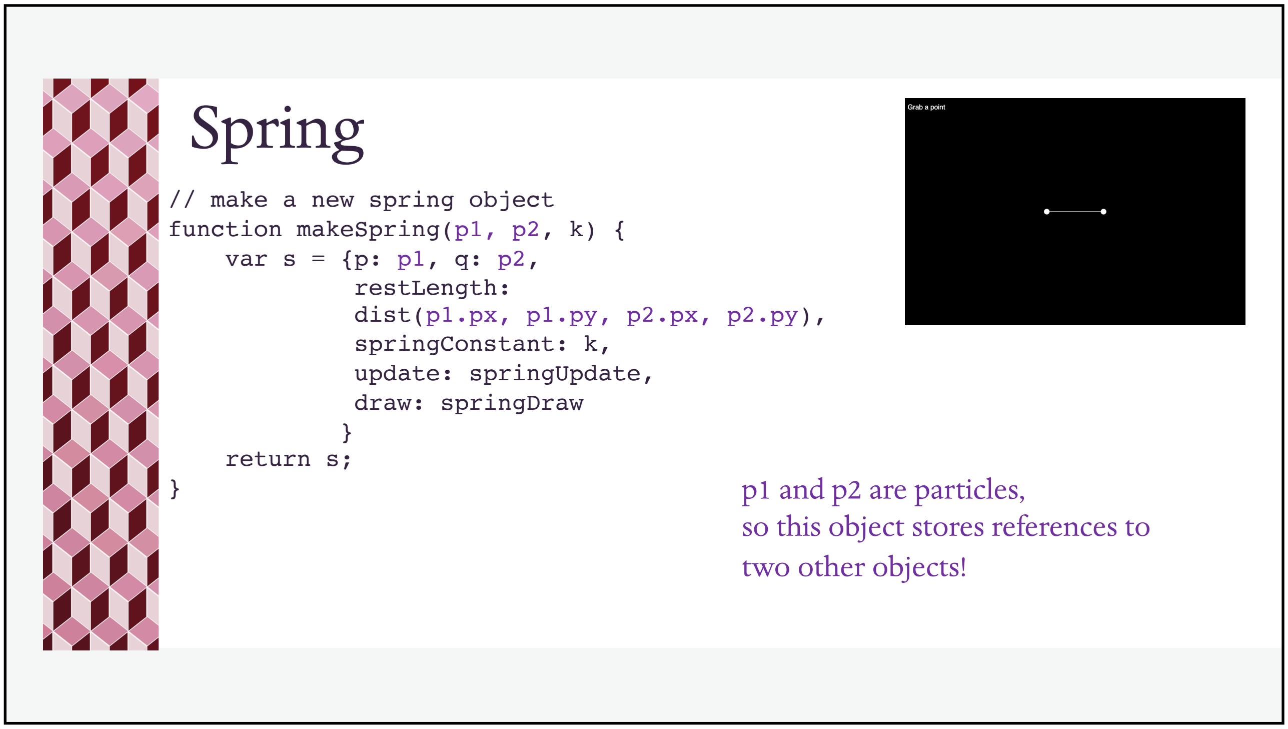
- Depending on the amount of damping, the spring may be very wiggly, or not.
- The constant ***k*** is the "spring constant" which is unique for each material.
- When ***k*** is low, the material is soft, flexible, stretchy.
- When the value of ***k*** is high, the material is stiff, snappy or brittle.
  - High values of ***k*** will expose the limits of Euler integration, our current numerical method for solving springs here; if ***k*** is too high, the simulation will "explode". (!)
- Good values for k in our solver are in the range of 0.01 to 0.5; try 0.1 initially.



Particle

```
// make a new particle (constructor)
function makeParticle(x, y, dx, dy) {
    var p = {px: x, py: y, vx: dx, vy: dy,
             mass: 1.0, damping: 0.96,
             bFixed: false,
             bLimitVelocities: false,
             bPeriodicBoundaries: false,
             bHardBoundaries: false,
             addForce: particleAddForce,
             update: particleUpdate,
             limitVelocities: particleLimitVelocities,
             handleBoundaries: particleHandleBoundaries,
             draw: particleDraw
            }
    return p;
}
```



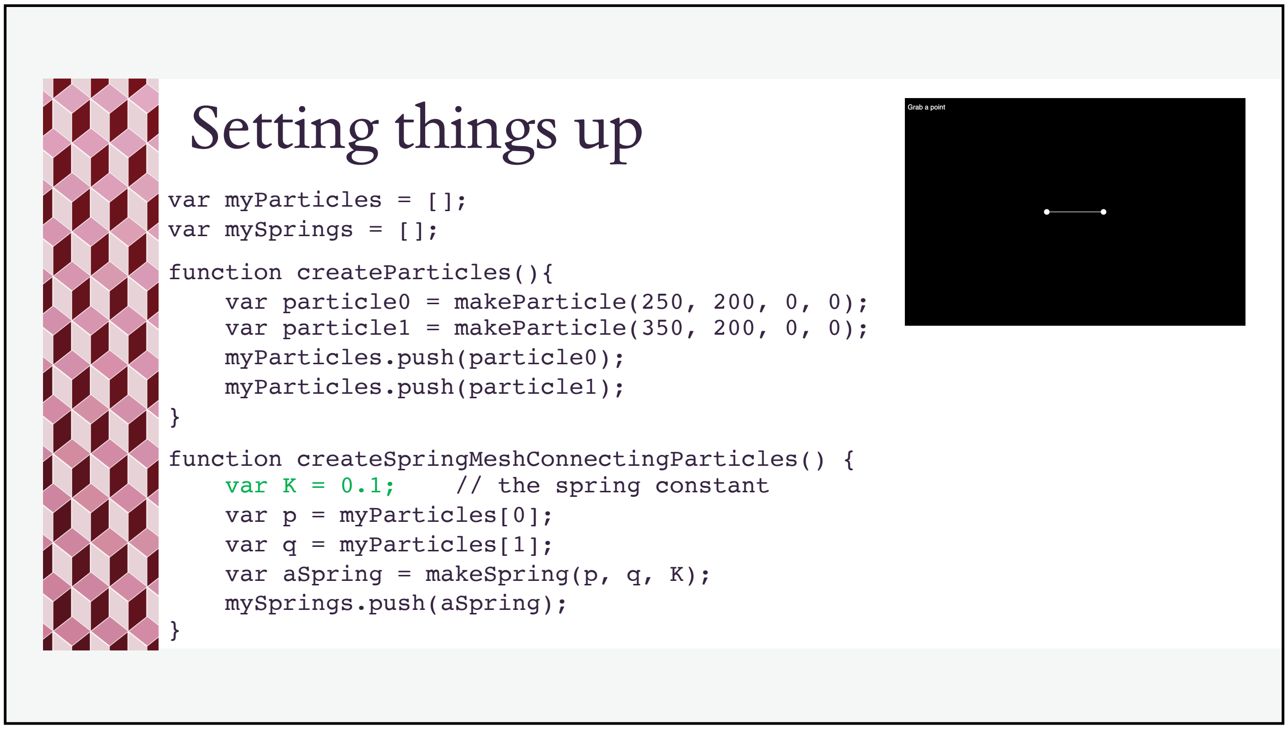


Grab a point

Spring

```
// make a new spring object
function makeSpring(p1, p2, k) {
    var s = {p: p1, q: p2,
             restLength:
             dist(p1.px, p1.py, p2.px, p2.py),
             springConstant: k,
             update: springUpdate,
             draw: springDraw
            }
    return s;
}
```

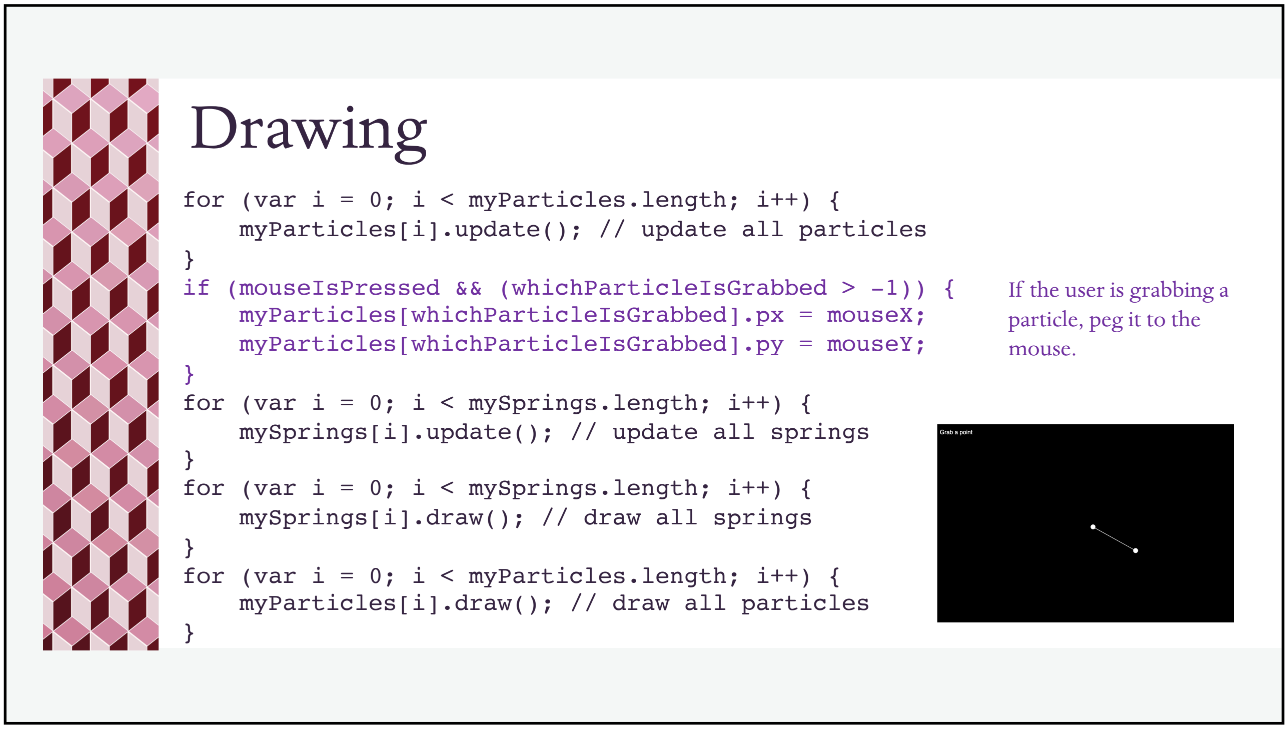
p1 and p2 are particles,
so this object stores references to
two other objects!



Grab a point

Setting things up

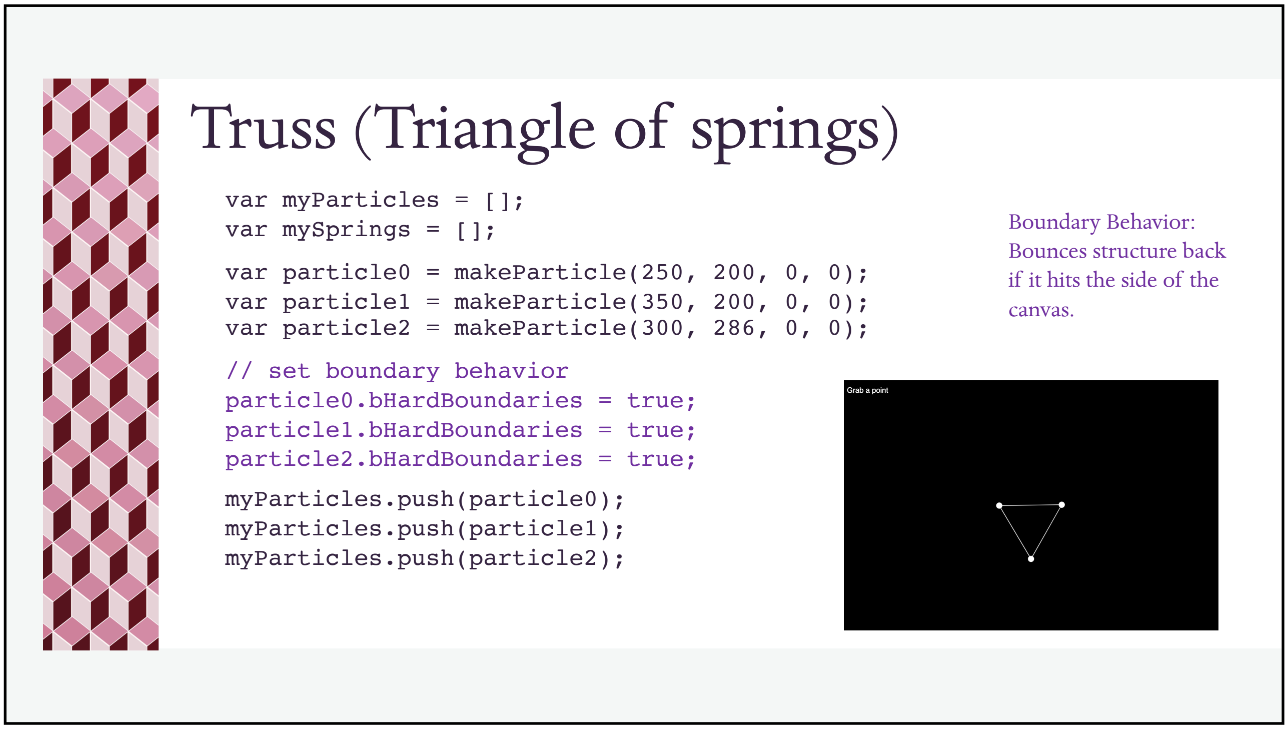
```
var myParticles = [];
var mySprings = [];

function createParticles(){
    var particle0 = makeParticle(250, 200, 0, 0);
    var particle1 = makeParticle(350, 200, 0, 0);
    myParticles.push(particle0);
    myParticles.push(particle1);
}

function createSpringMeshConnectingParticles() {
    var K = 0.1;    // the spring constant
    var p = myParticles[0];
    var q = myParticles[1];
    var aSpring = makeSpring(p, q, K);
    mySprings.push(aSpring);
}
```



Drawing

```
for (var i = 0; i < myParticles.length; i++) {
    myParticles[i].update(); // update all particles
}
if (mouseIsPressed && (whichParticleIsGrabbed > -1)) {
    myParticles[whichParticleIsGrabbed].px = mouseX;
    myParticles[whichParticleIsGrabbed].py = mouseY;
}
for (var i = 0; i < mySprings.length; i++) {
    mySprings[i].update(); // update all springs
}
for (var i = 0; i < mySprings.length; i++) {
    mySprings[i].draw(); // draw all springs
}
for (var i = 0; i < myParticles.length; i++) {
    myParticles[i].draw(); // draw all particles
}
```

If the user is grabbing a particle, peg it to the mouse.



Truss (Triangle of springs)

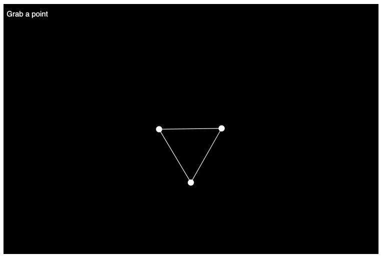
```
var myParticles = [];
var mySprings = [];

var particle0 = makeParticle(250, 200, 0, 0);
var particle1 = makeParticle(350, 200, 0, 0);
var particle2 = makeParticle(300, 286, 0, 0);

// set boundary behavior
particle0.bHardBoundaries = true;
particle1.bHardBoundaries = true;
particle2.bHardBoundaries = true;

myParticles.push(particle0);
myParticles.push(particle1);
myParticles.push(particle2);
```

Boundary Behavior: Bounces structure back if it hits the side of the canvas.

Truss (Triangle of springs)

```
var K = 0.1;

// Stitch the particles together by a spring.
var p = myParticles[0];
var q = myParticles[1];
var r = myParticles[2];

var aSpring0 = makeSpring(p, q, K);
mySprings.push(aSpring0);
var aSpring1 = makeSpring(q, r, K);
mySprings.push(aSpring1);
var aSpring2 = makeSpring(p, r, K);
mySprings.push(aSpring2);
```

Boundary Behavior: Bounces structure back if it hits the side of the canvas.

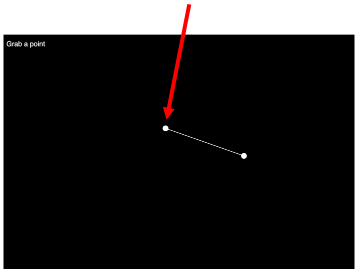


Fixed Spring

```
function createParticles(){
    var particle0 = makeParticle(250, 200, 0, 0);
    var particle1 = makeParticle(350, 200, 0, 0);
    particle1.bFixed = true;
    myParticles.push(particle0);
    myParticles.push(particle1);
}

function particleUpdate() {
    if (this.bFixed == false) {
        this.vx *= this.damping;
        this.vy *= this.damping;
        ...
    }
}
```

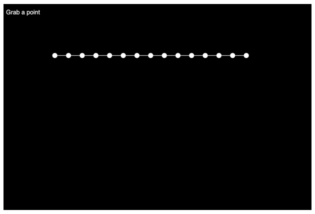
If the `bfixed` property is true, Then when this particle is updated, its `particleUpdate` function does nothing!

Fixed



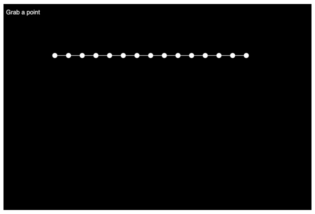
Rope (Array of Springs)

```
var myParticles = [];
var mySprings = [];
var nPoints = 15;

function createParticles() {
    for (var i = 0; i < nPoints; i++) {
        var rx = map(i, 0, nPoints, 100, 500);
        var ry = 100;
        var particle = makeParticle(rx, ry, 0, 0);
        particle.bHardBoundaries = true;
        myParticles.push(particle);
    }
}
```

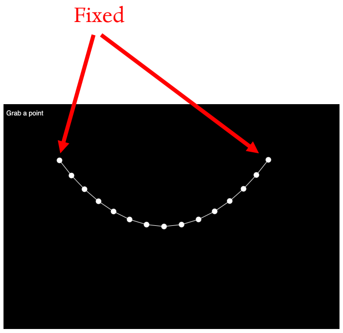


Rope (Array of Springs)

```
function createSpringMeshConnectingParticles() {
    // Stitch the particles together into a mesh by
    // connecting neighbors with a spring.

    var K = 0.1;    // the spring constant
    for (var i = 0; i < nPoints - 1; i++) {
        var p = myParticles[i];
        var q = myParticles[i + 1];
        var aSpring = makeSpring(p, q, K);
        mySprings.push(aSpring);
    }
}
```

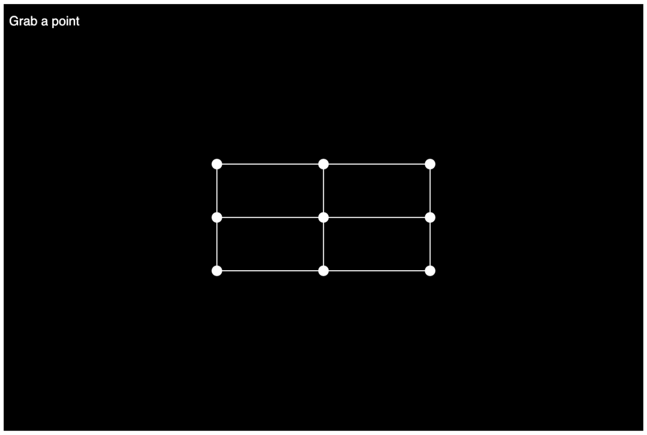


Rope with Gravity



```
function createParticles() {
    for (var i = 0; i < nPoints; i++) {
        var rx = map(i, 0, nPoints, 100, 500);
        var ry = 100;
        var particle = makeParticle(rx, ry, 0, 0);
        particle.bHardBoundaries = true;
        myParticles.push(particle);
    }
    myParticles[0].bFixed = true;
    myParticles[myParticles.length-1].bFixed = true;
}
```

In physics and geometry, a **catenary** is the **curve** that an idealized hanging chain or cable assumes under its own weight when supported only at its ends. The **catenary curve** has a U-like shape, superficially similar in appearance to a parabolic arch, but it is not a parabola. - Wikipedia

Try This