University of Houston – Clear Lake

SENG 5233

Lagrange, Euler, and JavaScript: A System Model

Joshua Pfeifer

5/8/19

Abstract

A two mass, spring, and dampener system like that on many vehicle suspensions was simulated visually using JavaScript on a webpage. The math dictating the equations of motion was derived using the Lagrangian technique, and the 2D simulation used a method of estimation for the change in the system per frame much like the Euler technique. The resulting simulation showed visually how, when placed over rough ground, the top mass experienced the least jostling acceleration when it was much heavier than the bottom mass, the spring constant was at the minimum that still avoids bottoming-out, and the dampening was high.

1 Introduction

Simulating a system can be done for many different reasons using many different methods, and making the best choice of methods should come down to the specific use case. Why are you simulating the system? What do you hope to gain from the simulation?

To explore the relationship between interconnected masses, spring, and dampener, I chose to visually represent a suspension system with a 2D web-based simulation. The purpose was not to make a highly accurate model for the development of an actual suspension system, but instead just to easily show how modifying any aspect of the system changes its performance in general. This means that both the mathematical representation of the system and its simulation over time do not have to be extremely rigorous; instead, they just must correctly show trends.

Visual representations of systems, beyond just plots generated by MATLAB, are invaluable in effectively communicating the intricacies of a system. Instead of needing to laboriously analyze graphs of velocity and position over time, actually experiencing how the simulated suspension performs when subjected to different terrains can provide clear and immediate results.

The next section elaborates on the suspension system, the third section covers mathematical modeling and the simulation itself, and following is a discussion of the results and conclusions.

2 System Description

The suspension system consists of two masses connected vertically by a spring and a dampener. Note neither masses are attached to a fixed point, so both can freefall due to gravity to the ground.

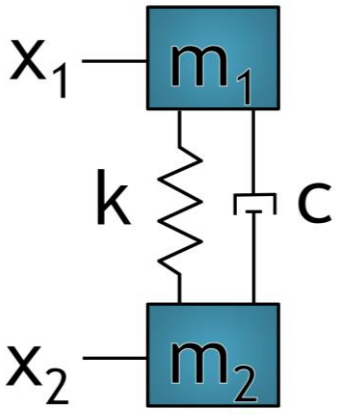

Figure 1: Suspension System

The spring has a spring constant of k, and the dampener has a dampening coefficient of c. When m_1 is at vertical location $x_1=0$ and m_2 is at $x_2=0$, the spring is fully relaxed with no potential energy stored.

Many assumptions were made about this system. The most glaring is that only vertical motion is allowed; this is like a simple interpretation of a car suspension where the motion of the wheel up and down with respect to the car and the ground is only acknowledged vertically. M_1 is always assumed to be directly over m_2, and it is assumed impossible to topple the system over. Other assumptions are that m_1 is the vehicle/rider, and m_2 is the wheel the suspension is attached to. It is also assumed m_2 has no "bounciness" with respect to hitting the ground – any downwards acceleration and velocity of m_2 simply disappear when touching the ground. K is held to be constant throughout for the massless spring, no matter any external influences such as temperature, extreme stretch or

compression conditions, etc. The same is assumed for c, including no special treatment for any extremely rapid speeds of convergence or divergence of the masses. The masses are also dealt with as point masses, where their shape and mass distribution make no difference.

3 Analysis

3.1 Mathematical Model

The mathematical method chosen to model the system was the Lagrangian technique. This method is most beneficial in more complicated systems where Newtonian mechanics would require additional constraint equations to stay in a Cartesian coordinate system [1], but it also works equally well for this system.

This technique models the system dynamics in a state function, i.e. a function (or set of functions) that depends only on the current state of the system and not its path to get to that state. The past oscillations of the suspension system should make no difference on any future oscillations – only the current state of the two degrees of freedom (x_1 and x_2, plus their derivatives with respect to time) matter for future predictions.

$$T = \frac{1}{2} m_1 \dot{x}_1^2 + \frac{1}{2} m_2 \dot{x}_2^2$$

$$V = m_1 g x_1 + m_2 g x_2 + \frac{1}{2} k (x_2 - x_1)^2$$

$$D = \frac{1}{2} c (\dot{x}_2 - \dot{x}_1)^2$$

Figure 2: Sources Energy

The first step was to identify the sources of kinetic (T), potential (V), and dampening (D) energy. The masses have kinetic energy relative to their speed (or the change in their position x), the spring has potential energy based on how much it is stretched or compressed outside of its relaxed position, the masses have potential energy due to their height, and the dampener exerts a force relative to the speed of the masses.

The second step was to combine kinetic and potential equations into the system Lagrangian equation. This is simply T – V, or kinetic minus potential energy.

$$L = \frac{1}{2} m_1 \dot{x}_1^2 + \frac{1}{2} m_2 \dot{x}_2^2 - m_1 g x_1 - m_2 g x_2 - \frac{1}{2} k (x_2 - x_1)^2$$

Figure 3: Lagrangian

Next, the generalized force related to each degree of freedom (x_1 and x_2) was derived using partial derivatives of the system Lagrangian and dampening equations. Partials with respect to x_1, following the Lagrangian formula, resulted in an equation that was rearranged to give the acceleration of x_1. The same was then done for x_2, as can be seen in the equations below.

$$Q = \left[\frac{\partial L}{\partial \dot{q}_i}\right]\frac{d}{dt} - \frac{\partial L}{\partial q_i} + \frac{\partial D}{\partial \dot{q}_i}$$

$$\frac{\partial L}{\partial \dot{x}_1} = \left[m_1 \dot{x}_1\right]\frac{d}{dt} = m_1 \ddot{x}_1 \qquad \frac{\partial L}{\partial \dot{x}_2} = \left[m_2 \dot{x}_2\right]\frac{d}{dt} = m_2 \ddot{x}_2$$

$$\frac{\partial L}{\partial x_1} = -m_1 g + k(1)(x_2 - x_1) \qquad \frac{\partial L}{\partial x_2} = -m_2 g - k(x_2 - x_1)$$

$$\frac{\partial D}{\partial \dot{x}_1} = c(\dot{x}_2 - \dot{x}_1)(-1) \qquad \frac{\partial D}{\partial \dot{x}_2} = c(\dot{x}_2 - \dot{x}_1)(+1)$$

$$Q_{x_1} = m_1 \ddot{x}_1 + m_1 g - k(x_2 - x_1) - c(\dot{x}_2 - \dot{x}_1) = 0 \qquad Q_{x_2} = m_2 \ddot{x}_2 + m_2 g + k(x_2 - x_1) + c(\dot{x}_2 - \dot{x}_1) = 0$$

$$\ddot{x}_1 = \frac{-m_1 g + k(x_2 - x_1) + c(\dot{x}_2 - \dot{x}_1)}{m_1} \qquad \ddot{x}_2 = \frac{-m_2 g - k(x_2 - x_1) - c(\dot{x}_2 - \dot{x}_1)}{m_2}$$

Figure 4: Resulting Differential Equations

3.2 Simulation

With the primary goal of providing a 2D visual representation of the system in action over time with easily adjustable input parameters, the simple graphics capability of the HTML canvas element programmed through JavaScript was chosen. Standard webpage sliders provided the user interface to alter masses plus spring and dampening constants.

Translating the continuous Lagrangian model into something displayable on the canvas element on an incremental, per-frame basis (at 30 frames-per-second) required another mathematical technique. The Euler method of estimating differential equations was chosen as it was an easy direct application of the Lagrangian math, just solved repetitively for each frame.

Given a first order differential equation with a known initial starting point, future points are predicted with the Euler method by adding the slope at that point times the step size [2]. This leads to an estimation that progressively trends farther from the actual solution to the differential equation (which is assumed unknown), but again this simulation was not intended to be extremely accurate. A graphical representation of this is provided from Wikipedia – the Euler estimate is in red, and the actual solution is in blue. Note how the slope of the blue line is followed from each red point to the next point, and then accurately re-evaluated at the new point and followed again the same set distance to the next point.

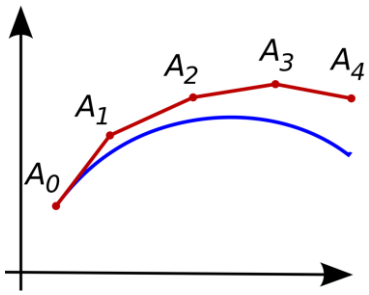

Figure 5: Euler Estimation

The application of the Euler estimate applied to this Lagrangian, second-order differential equation was done by (a) solving for the acceleration of each mass at each frame, (b) updating the velocity of each mass per frame based on its newly-calculated acceleration, and finally (c) updating the position at each frame based on new velocities. For example: if m_1 had an initial position of 10, velocity of 5, and solving for acceleration yielded an acceleration of -2, then (for this frame) velocity would be changed to 3, and then position would be updated to 13. Note that based on how the system is being displayed with the canvas element, these numbers refer to pixels (e.g. velocity of 3 moves the mass 3 pixels to the right per frame).

Mainly for aesthetic reasons, an astronaut riding a uni-motorcycle with an exaggerated spring suspension on Mars was chosen to be the visual representation of the system. The astronaut is fixed to the seat and represents m_1, and the wheel is m_2. The dampener is notably not displayed. The motion of the spring and masses are extremely obvious and require no interpretation from graphs. Numerous details are displayed at the top left of the screen, such as all input constants, max acceleration seen by the rider, and velocities of the rider and wheel. A score bar gives a general indication of what is desirable for a suspension system – low accelerations to the rider make it increase the quickest, high accelerations incur penalties, and bottoming-out the suspension resets the score to zero.

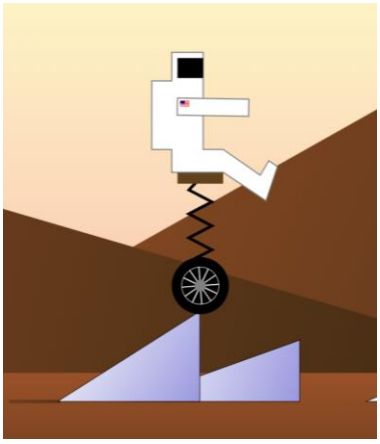

Figure 6: Simulation

The effect of the ground on the system also must be added to the equations that otherwise assume the masses are free-floating. This was done in the code as a separate per-frame calculation after acceleration was calculated and propagated down to velocity and position. If the wheel touched the ground, all negative acceleration and velocity of the wheel were cancelled out and its position was held to not fall below ground.

Randomly-generated ramps provided the necessary varying terrain to subject the suspension to a variety of tests. The horizontal motion of the system was simulated at a constant 5 pixels per frame (and thus far has played no part in any calculations), and to maintain this constant speed ramps must be climbed faster or slower dependent directly on their steepness. A vertical velocity was then imparted to the wheel anytime it

touches a ramp to ensure it climbs the hypotenuse of the ramp at a speed equivalent to reaching the top while maintaining a constant horizontal velocity. This is the equivalent of "hitting the throttle" based on the steepness of the ramp, if the throttle position translated immediately into velocity.

Each of the four sliders were then adjusted either somewhat high or low to observe the 16 different cases (2^4) of performance of the suspension system. Rider mass and wheel mass were varied, along with weakening and stiffening the spring through k and adjusting the dampening through c.

4 Results

The minimum acceleration to the rider with the minimum additional oscillations while still not bottoming out came from a relatively low spring constant (weak spring), high dampening, high rider mass, and low wheel mass. This allowed the rider to go off the edge of ramps with a very gradual and non-oscillatory transition to ground height due to the wheel shooting down extremely quickly to the new ground level and then the spring and dampener working together to slowly transition the rider afterwards.

Note that just swinging the sliders to their high or low extremes often results in glitches in the simulation due to the Euler estimate's shortcomings. These glitches also happened under initial development under almost any conditions when the acceleration, velocity, and position of each mass were calculated entirely for one mass, and then the next. This was due to the new position and velocity of the first mass, that were estimated for the next time step, were used in calculating the position and velocity of the other mass at the time step prior, which led to runaway-conditions that resulted in what appears like an explosion as the accelerations swing each frame to new extremes, ending in positive infinity and negative infinity values. This was solved for most cases by calculating both accelerations first, and only afterwards propagating down to both velocities and positions simultaneously each frame.

Similar glitches still happen when, for example, an extremely stiff spring is used in conjunction with extremely-high dampening. If the time-step estimation was small enough, these glitches should theoretically go away; however, at only 30fps they are still possible. An easy workaround is to adjust the sliders to slightly less extremes and then restart the simulation.

5 Summary and Conclusions

To simulate the effects of modifying different aspects of a suspension system, the system was modeled mathematically with the Lagrangian and animated with HTML canvas and JavaScript using Euler estimation per frame. A "Cadillac"-like ride was achieved through a weak spring, large rider-to-wheel mass ratio, and high dampening. The resulting 2D animated simulation is easy to interpret accurate general trends from, but not extremely precise as an estimate for exact movements and tolerances.

Future work to improve the simulation would include ironing out the bugs from selecting too-extreme of options that currently break the math. A first guess at a strategy for this would be to try and shrink the per-frame interval to much smaller, such as by halving the acceleration equations and doubling the frame rate. Additionally, the logic behind the interaction with the ground is not integrated well with the Lagrangian math, so further improvements are possible in that area as well.

As of the writing of this paper, the simulation can be run by anyone on nearly any connected device at joshuapfeifer.com/project.html. The code for this single webpage can be found through the website or additionally in the Appendix in Section 7. With cartoon-style animations, any complicated system can be made fun!

6 References

1. Lawrence, S. (2007, October 04). Basic Lagrangian Mechanics. Retrieved May 9, 2019, from http://www.physicsinsights.org/lagrange_1.html
2. Lakoba, T. (n.d.). Simple Euler Method and its Modifications [PDF]. University of Vermont. Retrieved May 9, 2019, from http://www.cems.uvm.edu/~tlakoba/math337/notes_1.pdf

7 Appendix

Full code of project.html below (suggest view as separate file with Sublime Text for color-coding benefit):

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.      <style>
5.          html {background-color: #a7a9ac;}
6.      </style>
7.  </head>
8.  <body>
9.      <title>Mars Ride</title>
10.     <canvas id="gameCanvas" width="800" height="600"></canvas>
11.     <div class="slideContainer">
12.         <input type="range" min="1" max="26" id="springInput" value="10">
13.         <input type="range" min="0" max="20" id="dampeningInput" value="4">
14.         <input type="range" min="1" max="20" id="massRiderInput" value="10">
15.         <input type="range" min="1" max="20" id="massWheelInput" value="10">
16.         <button onclick="start()" type="button">Start</button>
17.         <button onclick="reset()" type="button">Reset Values</button>
18.         <p><a href="SENG 5233 Project Pfeifer.pptx" download="Project.pptx">Download
    PowerPoint</a></p>
19.     </div>
20. <script>
21. //game stuff here
22. var canvas;
23. var canvasContext;
24.
25. var score = 0;
26. var scoreLeftXPos;
27.
28. var g = 1; // gravity
29. var k, c; // spring and dampening constant pulled in each time new spring object created
30. var wheel, rider; // two masses - rider top and wheel bottom
31. var spring;
32. var ground = 0;
33. var swapOrder = true;
34. var maxAccel = 0;
35. var backgroundManager; //object with functions to control background
36. var mountains = []; //scrolling background objects
37. var trail = [];
38. var ramps = [];
39. var rampVelocity = 0; //vertical velocity needed to continue up the slope of ramp while maintaining
    steady 5px/frame horizontal speed
40.
41. // contains main game loop
42. window.onload = function() {
43.     canvas = document.getElementById('gameCanvas');
44.     canvasContext = canvas.getContext('2d');
45.
46.     setCanvasSize(); // fit game canvas to screen 1st time prior to any initial setup stuff
47.
48.     // create objects
49.     start();
50.     backgroundManager = new backgroundManagerObject();//handles mountains
```

```
51.
52.         // set game loop
53.         var framesPerSecond = 30;
54.         setInterval(function() {
55.                     setCanvasSize(); // adjust as needed if window resized/phone turned
56.                     animateEverything();
57.         }, 1000/framesPerSecond);
58.  }
59.
60.  function setCanvasSize() {
61.         // get the size of the window's content area allowed for canvas
62.         var w = window.innerWidth - 20;
63.         var h = window.innerHeight - 100;
64.
65.      // If canvas size doesn't fit window, change it
66.         if(canvasContext.canvas.width !== w || canvasContext.canvas.height !== h) {
67.         canvasContext.canvas.width = w;
68.         canvasContext.canvas.height = h;
69.       }
70.  }
71.
72.  function start() {
73.         score = 0;
74.         maxAccel = 0;
75.         rampVelocity = 0;
76.         wheel = new wheelObject();
77.         rider = new riderObject();
78.         spring = new springObject();
79.         mountains = [];
80.         mountains.push(new backgroundMountainObject());//immediately create first mountain
81.         ramps = [];
82.         ramps.push(new backgroundRampObject());//immediately throw in first ramp too
83.         trail = []; // delete old dirt trail
84.         for(let i=0;i<100;i++){//set all sections of trail to "undisturbed"
85.                     trail.push(0);
86.         }
87.  }
88.
89.  function reset() {
90.         //change sliders back to original positions
91.         document.getElementById("springInput").value = "10";
92.         document.getElementById("dampeningInput").value = "4";
93.         document.getElementById("massRiderInput").value = "10";
94.         document.getElementById("massWheelInput").value = "10";
95.
96.         start(); //reset everything else
97.  }
98.
99.  function randFloat(min, max) {
100.        let randNumFloat = Math.random() * (max - min) + min;
101.        return randNumFloat;
102. }
103.
104. function wheelObject() {
105.        this.position = 0; // used for math, 0 is stable position
106.        this.xPos; // drawing position always centered
```

```
107.    this.yPos; // drawing position calculated based on this.position
108.    this.radius = 50;
109.    this.radians = 0; // used to draw rotating spokes
110.
111.    this.m2 = parseFloat(document.getElementById("massWheelInput").value);//pull in latest mass value from slider
112.    this.velocity = 30;
113.    this.acceleration;
114.
115.    this.calcAccel = function() {
116.            //do math
117.            this.acceleration = (-this.m2*g - k*(this.position-rider.position) - c*(this.velocity - rider.velocity))/this.m2;
118.    }
119.
120.    this.move = function() {
121.            //apply acceleration
122.            this.velocity+=this.acceleration;
123.
124.            //move
125.            this.position+=this.velocity;
126.            this.xPos = canvas.width/2;
127.            this.yPos = canvas.height/2 - this.position + 200;
128.
129.            //apply ground logic
130.            if(this.position < ground) {
131.                    this.position = ground;
132.                    this.velocity = Math.max(this.velocity, 0); // cancel out any downwards velocity
133.                    this.velocity = Math.max(this.velocity, rampVelocity); // ensure vertical velocity at least fast enough to climb steepness of ramp while maintaining horizontal velocity
134.                    this.acceleration = Math.max(this.acceleration, 0);     //cancel any downwards accel
135.            }
136.
137.            //apply "disturbed trail" if wheel touching at sand level "ground", not on ramp "ground" level
138.            if(this.position == ground && ground==0){
139.                    trail.unshift(1);//disturbed
140.            }else{
141.                    trail.unshift(0);//undisturbed
142.            }
143.            trail.pop();//keep overall trail same length
144.    }
145.
146.    this.draw = function() {
147.            //draw wheel trail
148.            for(let i=0;i<trail.length;i++){
149.                    if(trail[i]==1){
150.                            canvasContext.globalAlpha = 1 - (i/100);//fade out trail with distance
151.                            colorRect(this.xPos-((i+1)*5), canvas.height/2 + 200 + this.radius - 3, 5, 6, '#2b1d0e');
152.                    }
153.            }
154.            canvasContext.globalAlpha = 1; //reset back to no transparency
155.
156.            drawCircle(this.xPos, this.yPos, this.radius, 'black'); //tire
```

```
157.                drawCircleOutline(this.xPos, this.yPos, this.radius*.65, 'white'); //rim outline
158.
159.                //draw rotating spokes
160.                canvasContext.lineWidth=2;
161.                canvasContext.strokeStyle = 'white';
162.                canvasContext.beginPath();
163.                for(let i = 0; i < 12; i++) {
164.                        canvasContext.moveTo(this.xPos, this.yPos);
165.                        canvasContext.lineTo(this.xPos + this.radius * .65 * Math.cos(this.radians),
    this.yPos + this.radius *.65 * Math.sin(this.radians));
166.                        this.radians = this.radians + Math.PI / 6; // moves 30 degrees for every tick,
    making 12 ticks total
167.                }
168.                canvasContext.stroke();
169.                canvasContext.lineWidth=1; // return lines to normal thickness
170.                this.radians = this.radians+.1; // rotate a bit each frame
171.                drawCircle(this.xPos, this.yPos, this.radius * .2, 'grey');
172.        }
173.}
174.
175.function riderObject() {
176.        this.position = 0; // used for math, 0 is stable position
177.        this.xPos; // drawing position always centered
178.        this.yPos; // drawing position calculated based on this.position
179.        this.radius = 55;
180.        this.armRadians = 0; //arms will flail according to acceleration
181.
182.        this.m1 = parseFloat(document.getElementById("massRiderInput").value);//pull in latest mass
    value from slider
183.        this.velocity = 30;
184.        this.acceleration;
185.
186.        this.calcAccel = function() {
187.                //do math
188.                this.acceleration = (-this.m1*g + k*(wheel.position - this.position) + c*(wheel.velocity -
    this.velocity))/this.m1;
189.                if(Math.abs(this.acceleration)>maxAccel){
190.                        maxAccel=Math.abs(this.acceleration);
191.                }
192.                this.armRadians = this.armRadians + this.acceleration/70;//flail arm
193.        }
194.
195.        this.move = function() {
196.                //apply acceleration
197.                this.velocity+=this.acceleration;
198.
199.                this.armRadians*=.9; //slowly try to return arm to horizontal
200.
201.                //move
202.                this.position+=this.velocity;
203.                this.xPos = canvas.width/2;
204.                this.yPos = canvas.height/2 - this.position;
205.                //console.log(this.position);
206.        }
207.
208.        this.draw = function() {
```

```
209.                colorRect(this.xPos-40, this.yPos-10, 80, 20, '#654321'); //seat
210.
211.                //Rider
212.                canvasContext.save(); // save original canvas settings
213.        canvasContext.translate(this.xPos, this.yPos); // move canvas coordinate system
214.                canvasContext.lineWidth=2;
215.                canvasContext.strokeStyle = 'grey';
216.                canvasContext.fillStyle = 'white';
217.                canvasContext.beginPath();
218.                canvasContext.moveTo(40,-10); //top right corner of seat
219.                canvasContext.lineTo(115,40);
220.                canvasContext.lineTo(135,-20);
221.                canvasContext.lineTo(120,-30);
222.                canvasContext.lineTo(105,-5); //crook of shoe and leg
223.                canvasContext.lineTo(40,-50);
224.                canvasContext.lineTo(5,-50);
225.                canvasContext.lineTo(5,-220);
226.                canvasContext.lineTo(-50,-220);
227.                canvasContext.lineTo(-50,-170);
228.                canvasContext.lineTo(-85,-170);
229.                canvasContext.lineTo(-85,-50);
230.                canvasContext.lineTo(-50,-50);
231.                canvasContext.lineTo(-50,-10);
232.                canvasContext.closePath();
233.                canvasContext.fill();
234.                canvasContext.stroke();
235.
236.                colorRect(-40, -210, 45, 35, 'black'); //visor
237.
238.                //draw arm
239.        canvasContext.translate(-40, -140); // further move canvas to top left arm
240.        canvasContext.rotate(this.armRadians); //draw arm at correct angle from body
241.                colorRectandOutline(0, 0, 125, 30, 'white', 'grey') // arm rectangle
242.                // Flag on arm
243.                colorRect(5, 5, 16, 9, 'white');//white flag background
244.                canvasContext.lineWidth=1;
245.                canvasContext.strokeStyle = 'red';
246.                canvasContext.beginPath();
247.                for(let i = 5; i <= 14; i += 9 / 4) {//draw four evenly-spaced stripes
248.                        canvasContext.moveTo(5, i);
249.                        canvasContext.lineTo(21, i);
250.                }
251.                canvasContext.stroke();
252.                colorRect(5, 5, 8, 4.5, '#000080');//draw blue star section
253.                canvasContext.restore();//return to normal canvas coordinates, etc.
254.        }
255.}
256.
257.function updateScore() {
258.        if(Math.abs(rider.acceleration)<10){
259.                score = score + 1/Math.max(1,Math.pow(rider.acceleration,2));//add to score faster the
    smaller the accel
260.        }else{
261.                score = Math.max(0,score - Math.abs(rider.acceleration))//if over reasonable accel,
    deduct directly based on accel
262.        }
```

```
263.	if(rider.yPos>(wheel.yPos-wheel.radius)||maxAccel>1000) score = 0;//if bottom out or exploded, get rid of all points
264.}
265.
266.function springObject() {
267.	this.unAdjustedK = parseFloat(document.getElementById("springInput").value);//pull in latest spring value
268.	if(this.unAdjustedK<=10){
269.		k=this.unAdjustedK/10;
270.	}else{
271.		k=this.unAdjustedK-9;
272.	}
273.	c = parseFloat(document.getElementById("dampeningInput").value)/10;//pull in latest dampening value and adjust
274.	this.Color = 'black';
275.	this.numLinks = 9; // number of divisions of spring
276.
277.	// below calculated every frame
278.	this.center; //center of spring always readjusted to center of rider (which should stay center of screen)
279.	this.bottomEndPos;
280.	this.topEndPos;
281.	this.pointSpacing;
282.
283.	// instantiate spring tip points (this.numLinks - 1) with x, y coordinates that will be overridden later
284.	this.point1 = [0,0];
285.	this.point2 = [0,0];
286.	this.point3 = [0,0];
287.	this.point4 = [0,0];
288.	this.point5 = [0,0];
289.	this.point6 = [0,0];
290.	this.point7 = [0,0];
291.	this.point8 = [0,0];
292.
293.	this.draw = function() {
294.		//this.rightEndPos = cart.xPos;
295.		this.center = rider.xPos;
296.		this.bottomEndPos = wheel.yPos;
297.		this.topEndPos = rider.yPos;
298.		this.length = Math.max(1,this.bottomEndPos - this.topEndPos); //keep a min length
299.		this.pointSpacing = this.length / this.numLinks;
300.		//Set link y positions
301.		this.point1[1] = this.topEndPos + this.pointSpacing;
302.		this.point2[1] = this.topEndPos + this.pointSpacing*2;
303.		this.point3[1] = this.topEndPos + this.pointSpacing*3;
304.		this.point4[1] = this.topEndPos + this.pointSpacing*4;
305.		this.point5[1] = this.topEndPos + this.pointSpacing*5;
306.		this.point6[1] = this.topEndPos + this.pointSpacing*6;
307.		this.point7[1] = this.topEndPos + this.pointSpacing*7;
308.		this.point8[1] = this.topEndPos + this.pointSpacing*8;
309.		//set link x positions
310.		this.point1[0] = this.center-20;
311.		this.point2[0] = this.center+20;
312.		this.point3[0] = this.center-20;
313.		this.point4[0] = this.center+20;
```

```
314.                this.point5[0] = this.center-20;
315.                this.point6[0] = this.center+20;
316.                this.point7[0] = this.center-20;
317.                this.point8[0] = this.center+20;
318.
319.                canvasContext.lineWidth=5;
320.                canvasContext.strokeStyle = this.Color;
321.                canvasContext.beginPath();
322.                canvasContext.moveTo(this.center, this.topEndPos); // start at center of rider
323.                canvasContext.lineTo(this.point1[0], this.point1[1]);
324.                canvasContext.lineTo(this.point2[0], this.point2[1]);
325.                canvasContext.lineTo(this.point3[0], this.point3[1]);
326.                canvasContext.lineTo(this.point4[0], this.point4[1]);
327.                canvasContext.lineTo(this.point5[0], this.point5[1]);
328.                canvasContext.lineTo(this.point6[0], this.point6[1]);
329.                canvasContext.lineTo(this.point7[0], this.point7[1]);
330.                canvasContext.lineTo(this.point8[0], this.point8[1]);
331.                canvasContext.lineTo(this.center, this.bottomEndPos); // end spring at center of rider
332.                canvasContext.stroke();
333.                canvasContext.lineWidth=1; //return lines to normal size
334.        }
335.}
336.
337.function backgroundMountainObject() {
338.        this.yBottom; // will be recalculated each frame in case window size changes
339.        this.xLeftPos = canvas.width+10; // start just off screen
340.        this.xRightPos = this.xLeftPos + randFloat(50,canvas.width); // at least 50px wide up to width of
    screen
341.        this.xMiddlePos = randFloat(this.xLeftPos, this.xRightPos); // middle could be anywhere between
    left and right
342.        this.height = randFloat(10,canvas.height/2+200); // varies from 10px above ground to little over
    height of screen
343.        this.speed = 5;
344.
345.        this.move = function() {
346.                this.xLeftPos-=this.speed;
347.                this.xMiddlePos-=this.speed;
348.                this.xRightPos-=this.speed;
349.                this.yBottom = canvas.height/2 + 200;
350.        }
351.
352.        this.draw = function() {
353.                let grad =
    canvasContext.createLinearGradient(this.xRightPos,this.yBottom,this.xLeftPos,this.yBottom-
    this.height); //shade bottom right to top left
354.                grad.addColorStop(0,'#3f2a14');
355.                grad.addColorStop(1,'#A0522D');
356.                canvasContext.fillStyle = grad;
357.                canvasContext.beginPath();
358.                canvasContext.moveTo(this.xLeftPos,this.yBottom);
359.                canvasContext.lineTo(this.xMiddlePos,this.yBottom-this.height);
360.                canvasContext.lineTo(this.xRightPos,this.yBottom);
361.                canvasContext.closePath();
362.                canvasContext.fill();
363.        }
364.}
```

```
365.
366.function backgroundRampObject() {
367.     //randomly sized right triangle ramps
368.     this.yBottom; // will be recalculated each frame in case window size changes
369.     this.xLeftPos = canvas.width+10; // start just off screen
370.     this.xRightPos = this.xLeftPos + randFloat(50,500);
371.     this.height = randFloat(30,200);
372.     this.speed = 5; //scrolling horizontally speed
373.     this.slope = this.height / (this.xRightPos-this.xLeftPos);
374.     this.impartVelocity =  this.slope * this.speed; //per frame change in y
375.     this.centerScreenHeight;
376.
377.     this.move = function() {
378.             this.xLeftPos-=this.speed;
379.             this.xRightPos-=this.speed;
380.             this.yBottom = canvas.height/2 + 200 + wheel.radius;
381.
382.             //update height under wheel in center screen if applicable
383.             if(this.xLeftPos < canvas.width/2 && this.xRightPos > canvas.width/2){//then wheel is
    somewhere in middle of ramp
384.                     this.centerScreenHeight = (canvas.width/2 - this.xLeftPos) * this.slope; //#
    pixels above zero ground level
385.             }else{
386.                     this.centerScreenHeight = 0;
387.             }
388.
389.     }
390.
391.     this.draw = function() {
392.             let grad =
    canvasContext.createLinearGradient(this.xRightPos,this.yBottom,this.xLeftPos,this.yBottom-
    this.height); //shade bottom right to top left
393.             grad.addColorStop(0,'#9998dc');
394.             grad.addColorStop(1,'white');
395.             canvasContext.fillStyle = grad;
396.             canvasContext.strokeStyle = 'black';
397.             canvasContext.beginPath();
398.             canvasContext.moveTo(this.xLeftPos,this.yBottom);
399.             canvasContext.lineTo(this.xRightPos,this.yBottom-this.height);
400.             canvasContext.lineTo(this.xRightPos,this.yBottom);
401.             canvasContext.closePath();
402.             canvasContext.fill();
403.             canvasContext.stroke();
404.     }
405.}
406.
407.function backgroundManagerObject() {
408.     this.mountainTimeDelay = randFloat(30,400); //#of frames to wait until creating new mountain
409.     this.rampTimeDelay = randFloat(15,100); //delay for ramps
410.
411.     this.handle = function() {
412.             this.mountainTimeDelay-=1;
413.             this.rampTimeDelay-=1;
414.
415.             if(this.mountainTimeDelay<=0){//if timer runs out, make another mountain
416.                     this.createMountain();
```

```
417.                }
418.
419.                if(this.rampTimeDelay<=0){//if timer runs out, make another ramp
420.                        this.createRamp();
421.                }
422.
423.                //move and draw stuff
424.                colorRectGrad(0, 0, canvas.width, canvas.height, '#ffffcc', '#ffa4a4');//sky
425.                colorRectGrad(0, canvas.height/2+200, canvas.width, canvas.height-
    (canvas.height/2+200), '#A0522D', '#3f2a14');//ground
426.
427.                for(let i=0;i<mountains.length;i++){//move and draw each mountain
428.                        mountains[i].move();
429.                        mountains[i].draw();
430.                }
431.
432.                //ramp shennanigans
433.                ground=0;//reset each frame, and only overwrite if necessary
434.                rampVelocity=0;
435.                for(let i=0;i<ramps.length;i++){//move and draw each ramp, plus update ground and
    acceleration if needed
436.                        ramps[i].move();
437.                        ramps[i].draw();
438.                        if (ramps[i].centerScreenHeight>ground){
439.                                ground=ramps[i].centerScreenHeight;
440.                                rampVelocity=ramps[i].impartVelocity;
441.                        }
442.                }
443.        }
444.
445.        this.createMountain = function() {
446.                mountains.unshift(new backgroundMountainObject());//add new object to beginning of
    array, not end
447.                this.mountainTimeDelay = randFloat(30,400); //reset timer
448.                if(mountains.length > 20){ //don't waste memory/processing time with past mountains
449.                        mountains.pop();
450.                }
451.        }
452.
453.        this.createRamp = function() {
454.                ramps.unshift(new backgroundRampObject());//add new object to beginning of array, not
    end
455.                this.rampTimeDelay = randFloat(15,100); //reset timer
456.                if(ramps.length > 20){ //don't waste memory/processing time with past ramps
457.                        ramps.pop();
458.                }
459.        }
460.}
461.
462.function animateEverything() {
463.        //Do math
464.        wheel.calcAccel();
465.        rider.calcAccel();
466.        //apply acceleration to velocity and finally position
467.        wheel.move();
468.        rider.move();
```

```
469.
470.     updateScore();
471.
472.     //Draw in new locations
473.     backgroundManager.handle(); //draws background + moving background objects
474.     spring.draw();
475.     wheel.draw();
476.     rider.draw();
477.
478.     //score
479.     scoreLeftXPos = Math.max(200,canvas.width - (canvas.width/4000*score)-10);
480.     drawText('Score:'+Math.round(score), 20, 'black', canvas.width - 110, 30);
481.     colorRectGrad(scoreLeftXPos, 40, canvas.width-scoreLeftXPos-10, 20, '#00bfff', 'blue');
482.
483.     //text
484.     drawText('Rider Mass = '+rider.m1, 20, 'black', 10, 30);
485.     drawText('Wheel Mass = '+wheel.m2, 20, 'black', 10, 60);
486.     drawText('Spring constant = '+k, 20, 'black', 10, 90);
487.     drawText('Dampening constant = '+c, 20, 'black', 10, 120);
488.     drawText('Rider Velocity = '+Math.round(rider.velocity), 20, 'black', 10, 150);
489.     drawText('Wheel Velocity = '+Math.round(wheel.velocity), 20, 'black', 10, 180);
490.     drawText('Max Rider Acceleration = '+Math.round(maxAccel), 20, 'black', 10, 210);
491.     drawText('Adjustable inputs:', 20, 'white', 10, canvas.height-60);
492.     drawText('k', 20, 'white', 55, canvas.height-20);
493.     drawText('c', 20, 'white', 200, canvas.height-20);
494.     drawText('mRider', 20, 'white', 310, canvas.height-20);
495.     drawText('mWheel', 20, 'white', 440, canvas.height-20);
496. }
497.
498. function drawText(text, size, color, leftX, bottomY) {
499.     let fontCombo = size.toString() + 'px sans-serif';
500.     canvasContext.font = fontCombo;
501.     canvasContext.fillStyle = color;
502.     canvasContext.fillText(text, leftX, bottomY);
503. }
504.
505. function drawCircle(centerX, centerY, radius, drawColor) {
506.     canvasContext.fillStyle = drawColor;
507.     canvasContext.beginPath();
508.     //Draw including start and stop radian positions and clockwise
509.     canvasContext.arc(centerX, centerY, radius, 0, Math.PI*2, true);
510.     canvasContext.fill();
511. }
512.
513. function drawCircleOutline(centerX, centerY, radius, drawColor) {
514.     canvasContext.strokeStyle = drawColor;
515.     canvasContext.beginPath();
516.     //Draw including start and stop radian positions and clockwise
517.     canvasContext.arc(centerX, centerY, radius, 0, Math.PI*2, true);
518.     canvasContext.stroke();
519. }
520.
521. function colorRect(leftX, topY, width, height, drawColor) {
522.     canvasContext.fillStyle = drawColor;
523.     canvasContext.beginPath();
524.     canvasContext.fillRect(leftX, topY, width, height);
```

```
}

function colorRectandOutline(leftX, topY, width, height, fillColor, outlineColor) {
    canvasContext.fillStyle = fillColor;
    canvasContext.strokeStyle = outlineColor;
    canvasContext.beginPath();
    canvasContext.rect(leftX, topY, width, height);
    canvasContext.fill();
    canvasContext.stroke();
}

function colorRectGrad(leftX, topY, width, height, drawColor1, drawColor2) {
    let grad = canvasContext.createLinearGradient(leftX+width/2,topY,leftX+width/2,topY+height);
    grad.addColorStop(0,drawColor1);
    grad.addColorStop(1,drawColor2);
    canvasContext.fillStyle = grad;
    canvasContext.fillRect(leftX, topY, width, height);
}

</script>
</body>
</html>
```