Bates College

Senior Thesis

Physics Simulations Using Javascript

Author:
Peter Krieg

Advisor:
Gene Clough

Presented to The Department of Physics, Bates College
In Partial Fulfillment of the Requirements for the
Degree of Bachelor of Science

Lewiston, Maine
December 1st, 2014

Declaration of Authorship

I, Peter Krieg, declare that this thesis titled, 'Physics Simulations Using Javascript' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this College.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

BATES COLLEGE

Abstract

Faculty Name
Department of Physics

Bachelor of Sciences

Physics Simulations Using Javascript

by Peter Krieg

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too Lorem ipsum dolor sit amet, consectetur adipisicing elit. Asperiores obcaecati provident aliquam, impedit quidem iusto voluptatibus temporibus nihil error id est! Adipisci atque aut nostrum, recusandae, quae magnam repellat est.Lorem ipsum dolor sit amet, consectetur adipisicing elit. Sint distinctio autem, officia facilis molestiae veniam beatae, a deserunt laborum, optio earum laudantium atque aliquam, rerum incidunt hic repellat similique ratione?Lorem ipsum dolor sit amet, consectetur adipisicing elit. Incidunt atque amet, officiis consectetur adipisci debitis vero suscipit in doloribus, pariatur ipsa, accusantium aspernatur ab minima fugiat consequuntur distinctio, voluptates! Corrupti.Lorem ipsum dolor sit amet, consectetur adipisicing elit. Sunt deleniti eos quibusdam enim excepturi. Ea, aspernatur sed excepturi beatae, quod consequatur nisi, debitis blanditiis tempore vero vitae asperiores necessitatibus iure?

Acknowledgements

First and foremost, I would like to thank my advisor, Gene Clough. Gene, thank you for your guidance along the way: you helped formulate my ideas and encourage me along the process. I appreciate your patience and dedication to working with me–I have always enjoyed discussing problems with you. I always feel relieved walking into your office knowing that you will have some book pertaining any problem I'm having. It was only fitting that I had you for FYS advisor, and finished with thesis. I've really enjoyed talking with you these past four years.

I would also like to thank Professor Hong Lin, who met with me multiple different times to discuss my thesis. Thank you professor Lin for patiently explaining concepts with me and for letting me borrow some of your books. I appreciate your willingness to talk during an especially busy semester of teaching for you! Additionally, thank you to Professor Mark Semon, for showing me past examples of theses, and outlining the formatting requirements for Bates.

Thank you to my parents for encouraging me thorughout the whole process, and for at least trying to understand what I was doing for my thesis. Lastly, I would like to thank my friends and everyone else who was supportive of my thesis.

Contents

Abbreviations

HTML	HyperText Markup Language
JS	JavaScript
API	Application Programming Interface

Dedicated to my parents

Introduction

0.1 What is a Physics Simulation?

The purpose of this thesis is to present a series of physics simulations, each modeling a specific problem of physics as realistically as possible. These simulations differ from *animations*, which can be seen as predictable representations that always display the same visual. Animations are analagous to a movie script: no matter how many times you watch the movie, it will always end in the same way. Simulations, on the other hand, need to adapt to variable conditions, and be based partly on random processes. This brings up the topic of *dynamic* vs. *static* animation. Most of the physics simulations in this thesis will be dyamic because they present a unique viewing each time they are run, and can also involve user feedback which influences the outcome of the simulation.

Any simulation requires creating the illusion of motion. Almost every form of projected motion media uses frames to accomplish this. Researchers have shown that to make the simulation look realistic, it must be presented at a rate of around 60-100 frames per second. Anything far slower than this, and the human eye will detect the "choppiness" of the simulation. People can't detect anything much faster than 100 frames per second, so there is no need to project media faster than that, with the exception of slow-motion videography.

0.2 Methods of Producing a Simulation

The physics simulations in this thesis differ greatly from common animations. Movies and cartoons, for example, operate by displaying a series of images similar to one another, and displaying them as many frames per second to create the illusion of motion. My simulations, on the other hand, function by providing the *information* for each frame, and then providing the data for *how* the animation can be created. These instructions are passed onto the HTML5 canvas API, which creates the visual which can be seen in the web browser. Because physics simulations contain instructions instead of a series of images, the files of code take up far less space than a movie file would, for example. This is one primary advantage of coded simulations. Every simulation follows a similar set of steps, which can be simplified below:

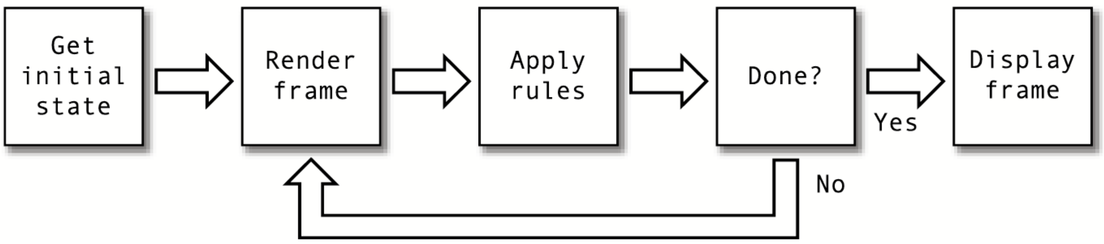

Figure 1: The frames of a general simulation

The canvas API gets the initial state of the simulation, which could the position of a ball, for example. Then, the frame is *rendered* by applying rules to the canvas element, and changing the initial state of the simulation. Once the rules have been applied, and all conditions are satisfied, the frame is rendered, and then displayed on the canvas element, to be seen in the web browser.

To produce any realistic simulation, the steps in figure 1 must be repeated multiple times per second. In fact, these steps must be repeated 60 times per second to achieve the desired 60 frames per second outlined in the previous section. Luckily, the canvas API is capable of running the instructions very quickly to make this simulation possible.

0.2.1 The Code

To program the simulations in this thesis, I chose to write the code in javascript. This scripting language is easy to view in any modern browser: therefore, all the simulations of this thesis can be viewed online. Javascript combines seamlessly with HTML5, which is why I mostly decided to use it for this thesis. The evolution of HTML (HyperText Markup Language) has progressed from simple web documents to complex web applications. For this thesis, every simulation utilizes the HTML5 <canvas>element, which has been used since around 2011. The HTML5 canvas API allows programmers to write javascript code that accesses the element and runs visual displays through a web browser The HTML needed to include a canvas can seen below:

```
1  <!doctype html>
2  <html>
3   <body>
4    <canvas id="canvas" width="500" height="500" >
5   </body>
6   <script>
7    var canvas = document.getElementById('canvas');
8    var context = canvas.getContext('2d');
9   </script>
10 </html>
```

Listing 1: The bare bones code necessary for an HTML document to include the canvas element

The above code displays the most basic HTML combined with javascript necessary to begin any simulation. Lines 7-8 are the only ones that actually contain javascript: this is the simple step necessary for the canvas API to recognize the HTML document. These two steps are necessary for any physics simulation. The step on line 7 initializes a JS variable and sets it equal to the canvas element on the web document object. The second step

All web browsers include some form of javascript interpreter: whenever the browser encounters a <script >element, it “passes” the code onto the JS interpreter. In listing 1, the HTML and JS code are written in the same document for clarity. While this is an acceptable practice, all future simulations will involve the HTML referencing to external JS documents to keep the contents separate.

While this thesis can contain code excerpts, figures, and screen-shots of various simulations, it obviously can't contain the flow of images itself. Therefore, I have put the entire thesis and its simulations on my personal website, which can be found at: http://www.peterkrieg.com/thesis. You can navigate by each chapter and view the simulations outlined in thesis.

Chapter 1

Some Basic Simulations

While the introduction outlined the computer programming necessary to produce simulations in general, this chapter will start to deal with the physics necessary to make simulations seem realistic. In this chapter, I will outline some examples of simulations with balls bouncing, and discussing the basic mechanics involved through the code.

1.1 Basic Ball Bouncing

A ball bouncing will show the basic kinematic equations, and how they are used in the javascript code. The following example displays a ball being dropped with an initial v_x, and bouncing off the walls and floor of the canvas element. The full code is shown below:

```
var canvas = document.getElementById(‘canvas’);
var context = canvas.getContext(‘2d’);

canvas.height = screen.height-200;
canvas.width = screen.width -100;

var radius = 20;
var color = ‘‘red";
var g = .1635; // acceleration due to gravity
var x = 40;  // initial horizontal position
var y = 40;  // initial vertical position
var vx = parseFloat(prompt(‘what is the initial horizontal speed of ball you
    would like?(recommended values of 1-20’));  // initial horizontal speed
var vy = 0;  // initial vertical speed

window.onload = init;

```

```
17 function init() {
18   setInterval(onEachStep, 1000/60); // 60 fps
19 };
20
21 function onEachStep() {
22   vy += g; // gravity increases the vertical speed
23   x += vx; // horizontal speed increases horizontal position
24   y += vy; // vertical speed increases vertical position
25
26   if (y > canvas.height - radius){ // if ball hits the ground
27     y = canvas.height - radius; // reposition it at the ground
28     vy *= -0.8; // then reverse and reduce its vertical speed
29   }
30   if (x > canvas.width - radius){ // if ball hits right wall
31     x = canvas.width - radius; // reposition it right at wall
32     vx *= -0.8;  // then reduce and reverse horizontal speed
33   }
34   if (x < radius){  // if ball hits left wall
35     x = radius;  // reposition it right at wall
36     vx *= -0.8  // then reverse and reduce horizontal speed
37   }
38   drawBall(); // draw the ball
39 };
40
41 function drawBall() {
42   with (context){
43     clearRect(0, 0, canvas.width, canvas.height);
44     fillStyle = color;
45     beginPath();
46     arc(x, y, radius, 0, 2*Math.PI, true);
47     closePath();
48     fill();
49   };
50 };
```

Listing 1.1: A basic ball bouncing simulation

This code functions by first setting up the canvas to be an appropriate size, on lines 1-5. Then, the simple variables of radius, color, initial positions/velocities, and acceleration are initialized. As mentioned in the introduction, the canvas HTML element defines positions in terms of pixels, with the top left corner of the canvas being the origin. Therefore, the ball is initialized to appear at (40, 40) which is near the top left corner for any computer screen. The value of g, the gravitational constant, is set to .1635 to accurately represent its value near Earth's surface, of 9.81 $\frac{m}{s^2}$. To understand why this value makes sense, it is necessary to understand the units of velocity on the HTML canvas. The position during the simulation is given in terms of pixels, which of course differs from the SI unit of meters. However, as long as g can be initialized to be 9.81 $\frac{px}{s^2}$, the simulation will still look physically accurate. This can be explained with the equation below:

$$9.81\,\frac{px}{s^2} = .1635\,\frac{\frac{px}{s}}{frame} \times \frac{60 frame}{s} \tag{1.1}$$

The value of g is calculated based on the fact that the simulation was run at 60 frames per second. Time is a central component of all physics, and for the simulations to behave realistically they must carefully take that into account.

The remainder of the code involves 3 functions that call one another to create the flow of the simulation. The first function, init ("initialize") is called when the browser window is loaded(line 15). This function simply delays the next function, onEachStep, by 16.66 ms, meaning that the function essentially runs 60 times per second, producing the desired 60 frames per second. Line 18 accomplishes this in a crude method: simulations later will involve more sophisticated techniques. The onEachStep function contains the instructions for each frame of the simulation. It involves multiple conditional if-statement loops that create the illusion that the ball bounces off of the walls and floor.

All 3 conditional loops involve a coefficient of restitution, or C_r. This is a mechanical property, representing how "bouncy" the ball is, and measures the ratio of the kinetic energy after and before the impact. This is derived below:

$$C_r = \sqrt{\frac{KE_f}{KE_i}} = \sqrt{\frac{\frac{1}{2}mv_f^2}{\frac{1}{2}mv_i^2}} = \frac{v_f}{v_i} \tag{1.2}$$

A C_r value of .8 was used for this simulation, which is comparable to that of a tennis ball[2]. The variable Cr represents this value in the code, and is simply multiplied by the velocity before impact, so the following equation results:

$$v_f = v_i * C_r \tag{1.3}$$

An essential part of any physics simulation involves *collision detection.* For the simple bouncing ball simulation, this is accomplished by conditional loops for if the ball's position exceeds the canvas constraints.

The last function of the program, drawBall, simply contains commands for the canvas API to draw. While these commands can be very complicated and intricate to create the exact visual aesthetic desired, the extent of these commands is not the purpose of this thesis. Basically, this function works by "erasing" the canvas of any previous graphics, and then creating a new visual with the arc() method.

The logic of the program can be summarized through the flow chart below:

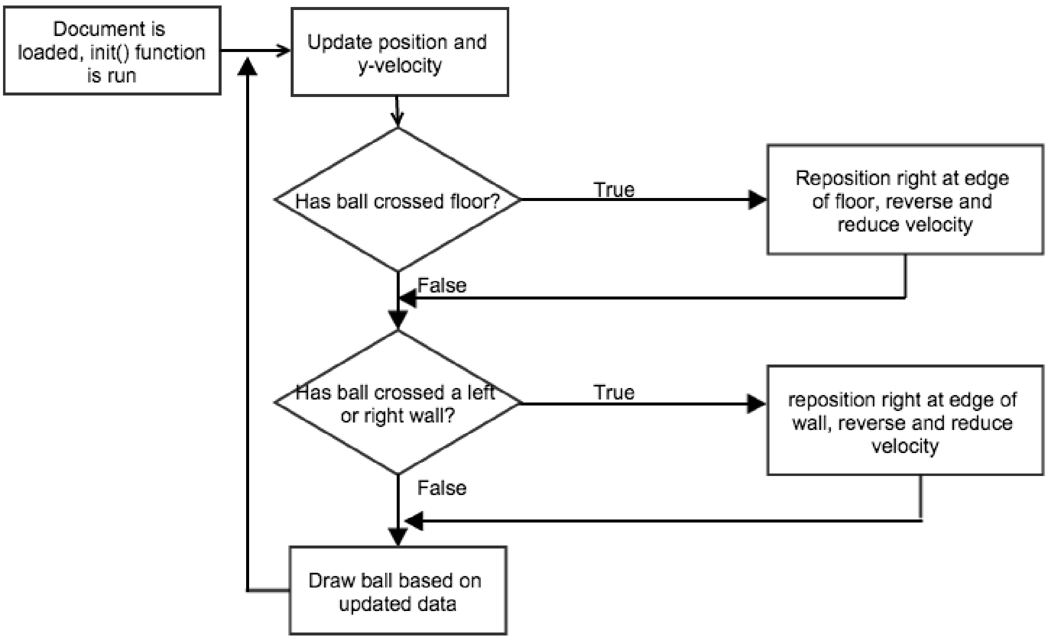

FIGURE 1.1: The logic flow chart of the basic bouncing ball simulation

1.2 More Advanced Ball Bouncing

While the previous example realistically incorporated the basic kinematic equations into account, it still fails to recognize important fundamentals of physics. The simulation in this chapter will still be a simple ball bouncing, but will take into account air resistance.

1.2.1 Background Physics

Drag is generally defined as the force on an object that resists its motion through a fluid. In the case of air resistance, the fluid is a gas, and therefore the process is called aerodynamic drag. Most of the drag force results as a response to the inertia of the fluid: the resistance it exerts to oppose being pushed aside. This can be expressed in the equation below:

$$f_{drag} = -\frac{1}{2}C_d\rho A v^2 \tag{1.4}$$

The equation involves a negative sign because the force of drag is always opposite the direction of motion. C_d is referred to as the drag coefficient, and is a dimensionless quantity that is used to model complex dependencies of shape, inclination, and flow conditions. While C_d is in general not an absolute constant for a given body shape, for the purpose of these simulations constant values were used. These values are typically determined experimentally: for example, the C_d of a sphere is approximately .47. In equation 1.4, ρ is the mass density of the fluid, in $\frac{kg}{m^3}$. Most of the simulations in this thesis occur in air, which has a density of 1.225 $\frac{kg}{m^3}$ (at sea level and 15 °C). Running the simulations in different fluids can be simulated by changing ρ to higher values (water, for example, would have ρ equal to 1000 $\frac{kg}{m^3}$). Lastly, A in equation 1.4 is the cross-sectional area of the object. A sphere, for example, would have a cross-sectional area of πr^2

The basic kinematic equations can also be used to make the simulations more physically realistic.

$$d = v_i t + \frac{1}{2}at^2 \tag{1.5}$$

$$v_f = v_i + at \tag{1.6}$$

These equations are fundamental to any physical situation and can be used to make the ball bouncing example of the previous section more realistic

1.2.2 The Code

Using these basic mechanics equations, the previous ball bouncing example can be made more physically accurate. The code below shows a second simulation which incorporates air resistance:

```
var x = 40;
var y =40;
var vy = 0;
var ay = 0;
var m = 1;
var r = 20;
var rSI = r* 0.000230909;  // radius in SI, converting px to m
var C_r = .8;  // Coefficient of restitution (tennis ball would be .8)
var rho = 1.2;    // density of air would be 1.2, water would be 1000
var dt = 60/1000;  // Time Step
var C_d = 0.47; //Coefficient of drag for sphere
var A = Math.PI * rSI * rSI;
var color = 'red';

window.onload = init();

function init(){
  setInterval(onEachStep, 1000/60);
}

function onEachStep(){
  var fy = 0;
  fy += m * 9.81;   // weight force
  if (vy>=0){
    fy -= 1* 0.5 *rho * C_d *A *vy *vy;
  }
  else {
    fy += 1*0.5 *rho *C_d *A *vy *vy;
  }

  ay = fy / m;
  vy += ay * dt;
  y += vy;

  // simple collision detection for floor only
  if (y + r > canvas.height){
    vy *= -C_r;
    y = canvas.height - r;
  }
  drawBall();
}
```

Listing 1.2: More advanced ball bouncing simulation

To eliminate redundancy, the code doesn't show previous functions used, such as drawBall(). The code also doesn't show the basic steps to initialize any simulation with the canvas and context commands. The code is very similar to the simulation in the previous section, except it incorporates air resistance. Essentially, this simulation uses more kinematic equations, by calculating the net force, acceleration, and velocity for each frame. First, the net vertical force is caculated, by combining the force of gravity $F_g = mg$ with the air drag from equation 1.4. This step involves a conditional loop for the cases of positive and negative velocity. Once the net force is calculated, the acceleration in the y-direction is found by using Newton's 2nd law of $F = ma$. From there, the velocity and vertical position of the ball are updated. Unlike the previous simulation, this example involves a variable dt, which is set to ∼16 ms for the same 60 frames per second.

This code involves interesting conversions between pixels and meters. Becuase the on screen simulation is presented eventually in terms of pixels, the physics equations must acknowledge this. The variable rSI on line 7 converts the radius of ball from pixels into meters. This is accomplished knowing the pixel density of the screen. This is commonly approximately 100 (dots per inch). The simulations were optimized for a macbook pro 15 inch model, which features 110 dpi. The calculation is shown below:

$$\frac{1m}{100cm}\frac{2.54cm}{1in}\frac{1in}{110px} \approx 0.00023091\frac{m}{px} \tag{1.7}$$

Once the conversion is made, the physics equations use the radius of the ball in terms of meters instead of pixels, which would give erroneous answers.

1.3 Multiple Balls Bouncing

So far, this chapter has dealt with a single object in motion. However, physics rarely involves just one body in motion. To demonstrate how more than one object can be displayed simultaneously, this section will show the case of multiple bouncing balls.

There is no new physics introduced in this section, but the coding concepts will be used repeatedly in later chapters of this thesis.

1.3.1 The Code

Appendix A

Appendix Title Here

THE APPENDIXGOESHERE

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Bibliography

[1] Leslie Lamport, *LaTeX: a document preparation system.* Addison Wesley, Massachusetts, 2nd edition, 1994.

[2] Roux, A., & Dickerson, J.(2007). ISB Journal of Physics *Coefficient of Restitution of a Tennis Ball.* Retrieved November 10, 2014, from http://www.isb.ac.th/HS/JoP/vol1/Papers/Tennis.