CS CAPSTONE TECHNOLOGY REVIEW

JUNE 11, 2019

INTERACTIVE 2D SIMULATIONS TO SUPPORT INQUIRY-BASED LEARNING IN MECHANICAL ENGINEERING

PREPARED FOR

OREGON STATE UNIVERSITY

TOM ESTKEDT, MILO KORETSKY

PREPARED BY

GROUP 53

LEARNING SIMULATIONS

CAMERON FRIEL

Abstract

This project involves solving the problem of some universities lacking resources to visually show physical and mechanical interactions for Mechanical Engineering concepts in the classroom. This impedes the learning process for Mechanical Engineers by forcing them to take an auditory approach to learning when the subject matter is very difficult to understand in the first place. This project is built on the research that shows that students can achieve a better understanding of difficult concepts by learning through simulated environments that they can interact with. By implementing two dimensional simulations based on these concepts, students will be able to visually interpret the concepts in the course. The solution being implemented is to create two dimensional simulations using primarily client side Java Script. The student will be able to modify certain values like the mass of an object or the angle that an object is dropped within a simulation in order to understand how interactions that occur in the real world are rationalized.

1 Introduction

Currently, some university's Mechanical Engineering classes lack the resources to visually demonstrate the physical and mechanical interactions that need to be learned in the classroom. This creates ambiguity for the students trying to understand Mechanical Engineering concepts, such as something rudimentary like how rolling cylinders of various sizes and materials down a ramp demonstrates the concepts of mass and acceleration. This project is built off of research that has shown that creating environments where students can visualize the concepts they are learning increases their understanding of the material. With the idea of cognitive conflict, when students observe ideas that challenge or contradict their prior ideas or beliefs, then they are pushed to think more about it. Especially when the concepts are non-intuitive, they may struggle. This project hopes to solve this problem by creating several two dimensional simulated environments that include these concepts. The students will be able to modify some of the variables within the simulation in order to enhance their understanding of the concepts being presented. These environments will be available online for students to see and interact with in order to attain a stronger grasp of the material. The simulated environments will go along with lessons hosted on the Concept Warehouse website, a platform to help supplement student's education.

In order to begin the development for this project, Cameron Friel has been tasked with conducting research in the areas of Software Development methodologies, Graphing Libraries for the web, and 2D Physics Engines. His role is to make sure the team sticks to deadlines and are made aware of problems that might come up. He is also responsible for making sure that any data within the simulation is able to be graphed in real time with readable data points. His third task is to make sure that the physics engine used is capable of simulating physical properties that would happen in real life into a 2D space on a web page.

2 Software Development Methodologies

The Software Development methodology chosen will be used throughout the entire project to make sure the team stays in line with the schedule set up by both the class and our own development specifications. The methodology will benefit us by setting up expected deadlines and give responsibility to each team member. It will also help make sure the client can see our progress and be able to steer us in the right direction if what we are doing is not align with what is expected.

2.1 Agile Development

Agile Development excels at being adaptive to the ever changing requirements the client can give during a project, as well as splitting development into short sprints so that errors can be caught sooner rather than later. By splitting up tasks into short sprints, the development that happens during that sprint will also be tested so errors are caught early on in the process. These short sprints also allow for constant communication with the client to show the current progress in the project to make sure what is being delivered is what the client wants. Where Agile Development falls short is that the outcome is never clear because the development is always changing and sprints are planned after the previous one ends [**?**].

2.2 Waterfall Development

The Waterfall Model is a very classic Software Development methodology because it is very structured and easy to follow. The Waterfall Model focuses on maintaining a sequential linear progression for project management [**?**]. Every project that follows the Waterfall Model starts by defining the requirements and creating the system design. After this, the project goes into implementation phase and then testing follows. Once the project has wrapped up testing the project is deployed and maintained for the rest of its life cycle. This cycle is good because it is structured and quick, but it also falters in many ways since it fails in projects that go on for a long time or when new features need to be added [**?**].

2.3 Spiral Development

The Spiral Model is used mainly in big projects that require tremendous risk management where failure cannot occur. The Spiral Model is similar to the Waterfall since it takes on the approach of a linear progression. This model has a high overhead since the project is slowly and meticulously defined by assessing risks and creating prototypes to prove that the system can be created. If the risk assessment is too much during any point in the project the project can be completely stopped [**?**].

3 Graphing Libraries

The project will require that within the simulations data be graphed in order to display a visual demonstration of what is going on. This will provide students with a secondary way to understand the material being presented that would not be possible with a physical version of it. The user will be able to visually see the graph as well as see the exact values for each point on the graph.

3.1 Highcharts

Highcharts is an interactive JavaScript graphing API that is free for non-commercial use. It is used by 72 of the top 100 corporations in the world. Additionally, Highcharts provides 20 different graphs to choose from including heat maps, scatter plots, and log graphs. The documentation is very robust and has plenty of demos for pretty much any use case. Since the library is built with pure JavaScript, there is no need for a user to have Flash or Java downloaded. It also has support for all modern browsers and is responsive to changes in screen size. When it comes to live data, Highcharts is able to dynamically graph points that come in from Ajax request or client side JavaScript [**?**].

3.2 Chart.js

Chart.js is an open source Graphing Library boasting 8 different graph types that can be used. It has responsive web capabilities and has excellent performance across all major web browsers. Chart.js provides incredible transitions when graphing and when new data comes in via animations. Chart.js is easy to use and the documentation covers its uses thoroughly. This library provides its users with many customization options including, but not limited to, a title, legend, and tool tip for its graphs. Furthermore, it has the most active Github repository when it comes to both making new content and fixing bugs when compared to other graphing libraries for the web. Lastly, Chart.js is very lightweight and is free for anyone to use [**?**].

3.3 Chartist.js

Chartist.js is another JavaScript solution for web. It supports SVG image format allowing for beautiful graphs on any resolution the end user might have. This library works on the basis of separation of concerns, meaning that customizing the graphs are dependent on CSS, while the functionality of the graph will be done with JavaScript. This library provides several animations that can be added to any of the SVG elements, though there is no support on Internet

Explorer. Overall, Chartist.js works with all major web browsers and supports responsively to the user's screen size. Its drawbacks come from its true potential being shown only to users with modern web browsers. It also is one of the least contributed repositories compared to its competitors [**?**].

4 2D Physics Engines

In order to create 2D simulations for mechanical engineering concepts, there will need to be an engine to render the elements of the simulation, as well as account for all of the physical properties that must happen within the simulation. A physics engine solves both of these issues since it includes libraries to add rigid bodies to elements, account for collision detection, add gravity to the world, and so much more. Utilizing a 2D physics engine will also negate a lot of development time reinventing the wheel with physical properties.

4.1 Matter.js

Matter.js is a free open source 2D physics game engine created for the web. Its main purpose is to create realistic 2D simulations on the web. It includes the necessary elements for a physics engine, such as rigid bodies, so each element that interacts with the world has collision detection added to it. Each element created can be tuned to the exact parameters it would have in the real world. For example, Matter.js allows one to set the objects friction and restitution so as to create a real life simulation within the browser. Matter.js has a plethora of documentation and demos for ease of use. Matter.js is written in JavaScript, making it web friendly, and does not require any third party libraries to use.

4.2 Phaser.io

Phaser.io is a HTML5 game engine which utilizes WebGL to allow for hardware acceleration. Phaser.io is open source and has an amazing set of examples and documentation for beginners. It recently integrated 2D physics into its code base in order to allow for high level physics simulations. It allows for rigid bodies, constraints, springs, restitution, and so much more. Since Phaser.io is a game engine, it is very easy to integrate assets into any simulation that we create. Additionaly, Phaser.io is built with JavaScript, so no third party libraries are needed, however it does need to have a web server up in order to test it. The game engine uses HTML5 canvas to display its games and is responsive depending on the size of the screen.

4.3 p2.js

p2.js is an open source 2D physics engine for the web. Written in JavaScript, it is easy to integrate and requires no third party libraries to be used. p2.js was show cased at the Google IO event in 2015. Its main use case is for game development, but can be used for anything requiring realistic physics properties. p2.js has all of the attributes one would need in a physics engine, such as rigid bodies, advanced constraints, collision detection, springs, and much more. A real application can be seen in one of its demos where objects of different masses are dropped into a pool of water to demonstrate buoyancy.

5 Conclusions

The best option out of the three software development methodologies would have to be Agile Development for us. Since there is very little risk management needed in our project, and we will most likely be dynamic with our design decisions, an Agile project management makes the most sense. Our client will also enjoy being able to see our additions to the project so that they can give feedback as well. All three graphing libraries chosen provide a very similar feature set, however, Chart.js is the most lightweight and provides exactly what we need. It is able to update in real time and the only graphing we will need are line graphs. Couple this with an active community, it will have the smoothest development experience. Matter.js will be our 2D physics engine of choice. It has the best documentation and has the most features that we need for our project.