**Structures with ammo.js and three.js project blog**

**Introduction**

This project I started as a part of computer science challenges it is a first-year module in which you learn and produce a project using technology you have never used before challenging you to go out of your comfort zone in learning new technologies and to solve problems you have never solved before. For this module, I was assigned the Structures project as a part of the topic choice of robotics and manufacturing topic I have an interest in.

Project objective:

**Model cheaply purchasable components e.g., 2x4 planks, nuts and bolts and create them in three.js/bullet with appropriate physics properties, including weight, friction. Objects should include costs and links for purchase.**

**Create models of physical parts with appropriate bending, twisting and breaking effects (model that they can occur) so that structures of suitable strength can be developed and limitations under forces can be incorporated into the design of machines.**

**Creating models of physical attachments e.g. nuts and bolts, nails, glue etc. that have parameters and can replicate plausible behaviours when subject to physical forces within a physics engine (including vibration)**

**Three.js and ammo.js**

To carry out this task I was assigned to use three.js and ammo.js; both JavaScript libraries; Three.js is used to create 3d graphics in a web browser and ammo.js to create a physics world these two libraries can be used in conjunction to display bodies I want to simulate and show how they are affected by physics. This task was overwhelming at first for me as I was not only new to physics and graphics engines but also, I had never used JavaScript and have no background in robotics and manufacturing, so as a prerequisite I had to learn this JavaScript. To learn this language, I followed a YouTube series created by a popular YouTube called mosh https://www.youtube.com/watch?v=W6NZfCO5SIk, This tutorial helped me pick up the basics of JavaScript like for loops, if statements, and creating objects, The tutorial also introduced me to the IDE I would be using for the duration of the project visual studio code. After getting comfortable with both JavaScript and visual studio code I had to move onto the next step learning three.js luckily three.js is a widely used library with many examples and tutorials ( https://threejs.org/ ) which I made use of. For me I have a preference for learning from YouTube tutorials, so I found a YouTube tutorial which I followed which covered the basics of three like creating a world creating your first objects, and so on. ( https://www.youtube.com/watch?v=YKzyhcyAijo&list=PLRtjMdoYXLf6mvjCmrltvsD0j12ZQDMfE ) . The next step I then moved on to learning the ammo.js library the issue I found with ammo.js is that unlike three.js there is hardly any documentation examples and tutorials to use, I struggled at first and found it overwhelming but through my searching, I found good tutorials which covered the basics well of ammo.js like constraints, objects, collision detection and filtering which got me started learning this library( https://medium.com/@bluemagnificent/intro-to-javascript-3d-physics-using-ammo-js-and-three-js-dd48df81f591 ).

**Project goals**

During this project, the main objective is to model cheaply purchasable components starting with 2\*4 planks using ammo and three. So far I have made some progress but the main idea for creating the planks for the project is to use dynamic rigid bodies connected with point 2 point constraints connecting each row of planks to the other columns and connect the columns to the rows of dynamic body, This will allow me to create bending for the object twisting and potentially breaking when certain forces are applied to the plank by creating constraints for the p2p for when certain constraints are applied to help with the twisting and bending of the plank. I also plan to add springs to the connections this will allow the object connections to remain straight if only small amounts of force are applied and if enough force is applied that the plank should bend the springs will allow the object to return to its original default position when the bending is done, the spring constraints in the future may also help with creating vibrations for the plank. To simulate the weight, I could look up planks with similar properties 2\*4 inches, as I'm starting with trying to recreate a plank of wood, I could use the weights of real-life planks of the length and height and apply the appropriate mass to each of the 1\*1\*1 body, I plan to do the same for other types of planks including metal by giving the bodies making up the plank different constraints for breaking, bending and twisting. I plan that other hard bodies like the nuts and bolts follow the same formula as the plank of course having different shapes and constraints, however for something like creating glue I think you could do this by using soft bodies instead of rigid bodies.

**Project**

Setting up the library’s and ide

Starting off creating my simulation I, of course, had to set up the workspace to do this I created a folder for my project which contained an index.html file and js folder containing the libraries ammo.js and three.js that I would be using to create my project.

Creating the three.js world

Creating the three.js world was just a matter of creating a function containing statements setting up the scene, camera, renderer, and some lighting.

Creating the ammo.js world

To create the ammo.js I had to create another function containing the declaration of parameters for the statement creating the physics world.

Creating the static world plane

Creating the plane, I just created a static rigid body a cube object with a big x and z value and 1 for the y axis representing the height and I set the mass to 0 to make it static.

Animating the rigid bodies in the world

To do this I created a recursive render frame function which instantiated a clock and called another method I called update Physics passing the clock as the parameters this method got all the positions of the ammo.js bodies in the world and updated the positions of three.js to match this, then the render loop rendered the scene and camera updating the world on the browser to match that in the physics world.

Creating the plank

When I was deciding how I should make the plank object I ran into the issue of what I should do to represent this object as there were many options and no clear solution to doing this, I ended up deciding that I should try to use several rigid bodies connected with p2p constraints and containing springs which allow the planks connections to simulate the changes in shape when physics are applied and to allow it to some degree to return to its original form like a soft body unless of course a high enough force was applied to the object that would make the plank permanently bent.

To Start I created two variables for containing the length and width of the planks in this project represented length and width are measured in inches, Then using this information I created a 2\*4 2d array the same amount of arrays as the length variable previously instantiated, and in each of these arrays the same amount of elements as the value of the width variable because 2d arrays are not a built-in function in JavaScript I used this method of creating the 2d arrays, the use of a 2d array was suggested by the module lecturer after I could not figure out how I would add a connection between each of the rigid bodies making up the planks.

After adding each of the blocks to the physics world and graphics world I added their bodies to the 2d array with their positions in the array correlating with their positions in the world relative to the other blocks making up the plank.

Next, I created the connections between the rigid bodies to do this I started with doing a nested for loop which for each column iterated through each of the rows connecting them and created another nested for loop which iterated through each row connecting each of the columns.

Adding collision constraints

During the project I had issues with the blocks falling through the static plane I realized after some code searching that my issue was my collision groups, By default, the collision group for the static plane and the blocks making up the plank were the same so they would not collide to change this I created 2 collision groups for my project and set the collision group of the static plane to colGroup1 and the blocks to colGroup2.

**Thanks for reading my blog if you want to get this project up and running with visual studio code IDE I suggest you download the project folder off GitHub and then follow this tutorial: https://docs.microsoft.com/en-us/visualstudio/ide/develop-code-in-visual-studio-without-projects-or-solutions?view=vs-2019 for how to open a project folder in vs code**