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 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. The project objective is defined below:

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 pre -requisite 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 not only got me set up on the basics of JavaScript like for loops, if statements and creating objects. I picked up the language quickly because of my experience in other programming languages like C# and java which have similarities to JavaScript and through the sample projects I was implementing to learn both the libraries for this project. however, I found out during the duration of my project the JavaScript is a little harder than showed in the tutorial due to some of the basic functions that other languages had built in which I had to implement myself like 2d arrays this tutorial also introduced me to the IDE I would be using for the duration of the project visual studio code. I had never really used this ide before, so I also had to do some tutorials learning it as well it was not that hard, and I have ended up with visual studio code being one of my favourite ides. 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 ( <https://www.youtube.com/watch?v=YKzyhcyAijo&list=PLRtjMdoYXLf6mvjCmrltvsD0j12ZQDMfE> ) this library was easy to get used to for this reasons I created some examples like creating blocks and so on. The next step I then moved onto 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 a good tutorial which covered the basic well of ammo.js constraints, objects, collision detection and filtering.

Project goals

In the short term during the first weeks of the project my goal was mainly aimed at learning and getting comfortable with three and ammo through making sample projects and learning code from examples I had found online, while also meeting my deadlines for the module like the how-to-guide on the fourth week I completed my goal with a lot of effort. The next 4 weeks I focused on learning concepts not covered well in tutorials and examples I had found and set the goal for myself of making some progress on simulating structures which I did through creating the first plank simulation object using hinges and rigid bodies which I will talk about later in the blog. I also had a goal of meeting my deadlines for this 4 week segment creating this blog showing the goals, the ideas and so on for this project I have took on and another how-guide for using ammo.js. Looking into the future I want to be able to make even more progress in simulating objects to do this I am planning to learn how to use springs and add them to my simulation to help simulate how simulated bodies I am creating react to physics applied to them. I also want to add constraints to the plank and show different materials for the plank including metal changing the constraints that cause the object to bend when certain forces are applied to them.

Ammo.js concepts

During my learning experience I covered some basic concepts of ammo.js like how to set up the physics world how to animate your world using three.js. Bu there were also concepts that were not so well covered and I had to put into practice through sample projects to get used learn and understand how to use these concepts, Each of these concepts I created a sample project and added it to a public github for your use to save you the pain of learning this stuff for yourself( <https://github.com/Ryan-McKee2001/three.js-ammo.js-blog-examples> ) you can take and use these examples to help you get started using and understanding ammo.js. The projects cover four of the main concepts of ammo.js:

* Rigid bodies
* Soft bodies
* Collision detection and filtering
* Constraints

Rigid bodies

Bodies in ammo.js are called collision objects or more commonly rigid bodies. Rigid bodies are the objects in the simulation which move, collide has mass and can have impulses applied to it.

There are three types of rigid bodies and many body shapes can be implemented in ammo.js. The three types of ammo.js bodies are:

* Static Rigid bodies – These bodies have a fixed position throughout the physics simulation and cannot be moved when defining these bodies they have a mass of 0 applied to them.
* Kinematic Rigid bodies – These bodies are not affected by the physics of the Ammo.js world but are can be animated during the physics simulation, These bodies are similar to the static rigid bodies in that they are have a mass of 0, but unlike the static rigid bodies we add the bodies to the rigid bodies array we typically define globally at the start of out program script to allow the positions of it and its three.js counterpart to be changed in the worlds as it is animated.
* Dynamic rigid bodies – These are the most intensive rigid body type in ammo.js, these bodies are fully affected by the physics of the world.

Soft bodies

Soft bodies are defined as bodies in computer graphics that visually and realistically represent the motion and properties of deformable objects while being able to some extent retain its shape, these bodies can represent bodies like balloons or cloths.

Constraints

Constraints can be thought of as joints that would be used real life to connect multiple objects together. There are multiple types of constraints in ammo.js to carry out different functions:

Point – to – point constraints.

Point to point constraints limit the translation of two pivot points of rigid bodies to match the world space. Using this constraint, you can create a chain of rigid bodies. This constraint can be useful for creating something like a robotic arm or in the case of my structures project a plank of wood using several rigid bodies in column and rows.

Hinge constraints

Hinge constraints are joints which restrict rotation around the pivot of two bodies to only one axis, therefore this axis could be useful for creating something like a door. Or even flaps on an airplane model as this constraint also allows the user to specify the limits and motor of the hinge.

Slider constraint

The slider constraint allows a body to rotate around one axis and translate along that axis. This constraint would be useful for creating something like a piston.

Cone twist constraints

This cone constraint is a special version of the point-to-point constraint that adds cone and twist axis limits. The axis for this constraint serves as a twist axis. This constraint is useful for creating something like a ragdoll.

Collision detection

Collision detection in ammo.js is a concept that allows for the collision and interaction between objects with there ammo.js. There are four main concepts within collision detection which allow you to show how objects interact in a world

* Contact Manifold check
* Contact test
* Contact pair test
* Ghost objects

Springs

**Project layout**

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 a 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 static plane like I just created 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 had to create

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 allows 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 be permantly bent.

To Start off I the plank I created two variables for containing the planks length and width in this project represented in inches, Then using this information I created a 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 width assigned in 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 connection between each of the rigid bodies making up the planks.

After adding each of the block to the physics world and graphics world I added them 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 had to add

I\*mage of 2d array\*

Adding collision constraints

During the project I had issues with the blocks falling through the static plane I realised after some code searching that my issue was my collision mask, masks in ammo.js decide whether are not two object should collide together through the use of a bitwise and operation if two objects are in close proximity in the physics world the object that is going to collide does an and operation and if the and operation does not cause the binary to be equal to 0 the two bodies collide otherwise they do not collide an example of this would be

Ball collision mask = 1

Block collision mask = 2

And operation = 0001

= 0010

0000 = 0 therefore the two objects collide

Otherwise 0001

0001

0001 = 1 therefore the two objects don’t collide. This is because the two objects are in the same collision groups.

Creating the plank hinges

To add the hinges I had to iterate through each of the columns and connect each of the objects using p2p constraints using the objects of the 2d array. I then went through each incrementally went through each row and connected each of the column items this then led to the plank being completely connected.

**Goals**

**Throughout the duration of this project the objective has slowly deviated from what I was originally assigned, Although I have so far tried to complete to project assigned, I have found that it is more important that I learn and document how to use ammo.js as it has so little documentation causing the library to be pretty inaccessible to someone who has not used a physics engine before. When I have finished this module, I hope to have made good progress in simulating some structures however I want to have made the library more accessible and easier to learn making it easier for someone to pick up from where I left my project off.**

**Plan of how it can be achieved.**