Structures with ammo.js and three.js project blog

**Introduction**

This project I started as a part of my module 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 in the physics world I create. 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.

Three.js

Getting started with ammo.js.

Before starting to add object and cool stuff you need to first define the physics world in all of the sample projects this was the main similarity which I done in all.

At first looking at examples of ammo.js projects and how they create the world the statements made no sense to me at all as I could not understand what the statements function calls for this creating the world function were doing. It took a while to get a grasp on what was happening but luckily you have me to explain these concepts and save you from the pain I dealt with trying to figure out what they did.

Ammo.js concepts

During my learning experience I covered some basic concepts of ammo.js in sequential order. These concepts that I put into practice at the start of my project through the use of some sample projects ended up becoming fundamental for the development of my structures project, I am going to go over the ammo.js concepts I learned using some sample projects I created which you can access through this github link: ( <https://github.com/Ryan-McKee2001/three.js-ammo.js-blog-examples> ) you can take and use to help you get started using and understanding ammo.js and follow along with my all my code is commented out in addition to the information in the blog so hopefully it should help you a lot with getting started using ammo.js. The projects I have created so far mainly focus on four main concepts:

* 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.
* 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.
* 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.

Now we will show examples of each of these body types in code and how they work.

Static rigid bodies

Static rigid bodies are defined similarly to how we would define a dynamic or Kinematic rigid body the difference when creating this body is that we set the mass to 0, This makes the body unmovable in the simulation and therefore static as well, Because it is static we do not need to add it to the rigid bodies array we typically define at the start of the script as we will not be moving this object throughout the physics simulation so we will not need to update the three.js representation of this static body in the animation loop. In this example we create a static body as a cube object which will be used as the plane for the world to be based.

Kinematic Rigid bodies

This body has a mass of 0 however, we add it to the rigid bodies array as we will be updating its position in the physics world thus also updating the three.js representation in the render loop function. For this example, I animated the body by just changing its z position by +0.03 for 10 frames then updating the blocks position by +0.03 on the x for another 10 frames repeating this process but with -0.03 for 10 frames on the z and x positions. There are many other things you could do like animate the block when a key is pressed by the user and so on, but I will not be covering that here.

Dynamic Rigid bodies

This is the dynamic body of the program which the other two types of rigid body are based on, In this example I set the mass of the rigid body to be 1 therefore this object will be effected by the ammo.js physics world. I also added this object to the rigid bodies array

Soft bodies

Constraints

Collision detection

Springs

**Project layout**

Setting up the library’s and ide

Creating the three.js world

Creating the ammo.js world

Creating the static world plane

Animating the rigid bodies in the world

Creating the plank

Creating the plank

When I was deciding how I should make the plnka object I ran into the issue of what I should do to represent this object, I ended up deciding using the resources in ammo.js to simulate the properties of a plank that I should use several rigid bodies connected with p2p constraints and containing springs which allows the planks connections to simulate the changes in shape of the plank and be able to return to original positions.

To Start off I the plank I created two variable 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:

I created the 2d array so when I created the cube bodies in ammo.js I could add each of them to the 2d array with a position representing their actual positions in the ammo.js world so I could later carry out other function to these blocks like adding the p2p constraints and eventually add springs to them.

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.**