**COMP410- COMPUTER GRAPHICS TERM PROJECT**

**Project Report**

**Alpay Sabuncuoglu – 40281**

Index of Content

[Introduction 2](#_Toc483790034)

[Program Design Process and Motivation 2](#_Toc483790035)

[WebGL Program Structure [2 (Parisi, 2012)] 2](#_Toc483790036)

[But I am Working With Three.JS, what is the difference with WebGL? 3](#_Toc483790037)

[Helper Libraries 3](#_Toc483790038)

[IMPLEMENTATION 3](#_Toc483790039)

[What does init() and animate() do? 4](#_Toc483790040)

[OUTPUT AND DISCUSSION 5](#_Toc483790041)

[References 6](#_Toc483790042)

# Introduction

In this project, I created a shooting game with WebGL. What I planned was; while the players continuously move in a tunnel, they shoot some objects and the gun is tracked by tracking.js and bullet motion uses real physics engine ammo.js. But some small changes happened while I was coding. First, using physics engine in a space game was not a good idea, so I changed it with laser (I also did the bullet part -which can be examined from ./assets/src/index\_bullet.js-) but that does not look nice.

The project specifications in the proposal were:

- Continuous game-play in a tuple with Three.js.

- Character creation with Maya.

- Object Tracking with tracking.js

- Real-physics simulation with ammo.js / changed with laser

# Program Design Process and Motivation

While I was planning this term project my motivation was learning maximum possible tools for everyday graphics application. WebGL is a web-content API, based on OPENGL ES 2.0 which renders on HTML canvas object and it is supported by all modern browsers [1]. It executes the program on native GPU. It is simple to create a WebGL program since it can be mixed with other HTML and JS elements and the program frame is ready because we are using it in a browser. A guide to build a WebGL application can be listed in the WebGL Program Structure content.

## WebGL Program Structure [2 (Parisi, 2012)]

1. Create a canvas element.

2. Obtain a drawing context for the canvas.

3. Initialize the viewport.

4. Create one or more buffers containing the data to be rendered (typically vertices).

5. Create one or more matrices to define the transformation from vertex buffers to screen space.

6. Create one or more shaders to implement the drawing algorithm.

7. Initialize the shaders with parameters.

8. Draw

## But I am Working with Three.JS, what is the difference with WebGL?

Three.js is a JS library that allows users to extensive usage of WebGL. So, there is no difference between WebGL and Three.JS: it is just easier to use Three libraries. To be able to display anything with three.js, programmer needs three things: A scene, a camera, and a renderer so the scene can be rendered with the camera. The program structure does not change while using Three.js but the work has done is dramatically change since extensive libraries does most of the job.

## Helper Libraries

I also used tracking.js to track movement of the gun (which is just a cyan plate) and ammo.js to use physics engine in my program. Normally, the physics engine gives a better look on a game but it was not valid for my program, while I kept it to learn how to use it. Better and impressive examples can be found under the three.js/examples/physics (<https://threejs.org/examples/>).

# IMPLEMENTATION

First, let’s traverse the program folder:

* assets
  + All library js files, common files for music and texture

🡪src

The files that I created for this project. Includes index\_bullet.js (game with bullet), index\_laser.js (game with laser), LASER.laserbeam.js and LASER.laserPros.js

* Css
  + Style file for html
* Monster
  + Monster object, which is exported form Maya

index.html

My implementation is at assets/index.js and the main program calls are implemented at index.html. Let’s look at the code at the index.html file:

<script>

if (!Detector.webgl) {

Detector.addGetWebGLMessage();

document.getElementById('container').innerHTML = "";

}

init();

animate();

var video = document.getElementById('video');

var canvas = document.getElementById('canvas');

var context = canvas.getContext('2d');

var tracker = new tracking.ColorTracker('cyan');

var target = { x: 0, y: 0};

tracker.on('track', function (event) {

if (event.data.length === 0) {

// No colors were detected in this frame.

} else {

event.data.forEach(function (rect) {

target.x = rect.x; target.y = rect.y;

console.log(target.x, target.y, rect.height, rect.width, rect.color);

});

}

});

tracking.track('#video', tracker, { camera: true });

</script>

This is the whole code in the html file. It checks if the browser supports WebGL, initiate the graphics and physics functions, animate and render them. Also I placed the tracker code here, which identifies the blue object and track it. The detailed explanation and demo program can be found from: <https://trackingjs.com/examples/color_camera.html>, which I also implemented the same design pattern in my program.

## What does init () and animate () do?

init () does what Three.js wants from us, it defines a scene, camera and renderer (this is not all init () does, but main purpose is this.) It also instantiates the graphics and physics objects. The graphical objects are:

* Monster object which is loaded with the Object/MTL loader which is defined by Three.js libraries.

*It is exported from the Maya with Three.js exporting tool. I could not create a better-looking monster, it is just a combination of some deformed cylinders and textures since Maya is totally different specialization tool.*

* Tunnel Object

tunnelMesh = new THREE.Mesh(

new THREE.CylinderGeometry(50, 50, 1024, 16, 32, true),

new THREE.MeshBasicMaterial({

color: innerColor,

ambient: innerColor,

transparent: true,

alphaMap: tunnelTexture,

shininess: 0,

side: THREE.BackSide,

})

);

tunnelMesh.rotation.x = deg2rad(90);

tunnelMesh.position.z = 128;

scene.add(tunnelMesh);

*Creating a tunnel is creating a rotating cylinder. And while rendering I changed the texture offset so it creates a wormhole effect as seen in Figure 1.*

* Laser (defined in LASER.laserbeam.js and LASER.laserPros.js)
* Starfield (turns clockwise at the background)

init() also calls initPhysics() function to initiate physics engine of the program (in the version that uses physics engine).

animate() is simple yet more complicated function which has another function in its structure:

requestAnimationFrame(animate);

render();

update();

requestAnimationFrame() is a recursive function which tells the browser that we wish to perform an animation and requests that the browser call a specified function to update an animation before the next repaint. The method takes as an argument a call-back to be invoked before the repaint.

render() function does what its name tells, it updates some variables (which could also be implemented on the update part) and renders the scene with the call:

renderer.render(scene, camera);

update() function checks the variable positions and target positions and if they intersects gives points to the player.

# OUTPUT AND DISCUSSION

So, starting from simple when we have scene, camera and renderer there is nothing but after adding tunnelMesh it looks like I did something useful. The pictures at the Figure 2 are some of the screen outputs when I first create the tunnelMesh. I used texture wrapping around the cylinder (texture is assets/electricity.jpg), and set the camera inside from the cylinder and update the offset so it looks like turning around. When the player looks from the inside of a turning cylinder, it looks like moving forward. But nothing here goes anywhere!

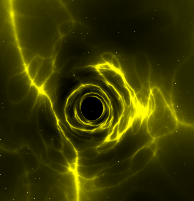
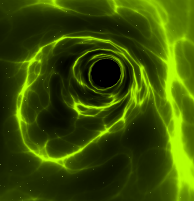
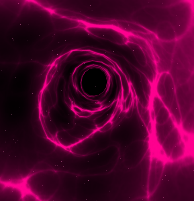


Figure 1. Screen outputs of the Tunnel Mesh

From this part, normally program uses the tracking data to move the aim, but for better debug I used mouse.x and mouse.y in report and program. Then after adding monster to the system, I should have find a solution how to recognise hitting something. I used ray-casting as a solution:

mouse.x = (event.clientX / window.innerWidth) \* 2 - 1;

mouse.y = - (event.clientY / window.innerHeight) \* 2 + 1;

var vector = new THREE.Vector3(mouse.x, mouse.y, 1).unproject(camera);

var ray = new THREE.Raycaster(camera.position, vector.sub(camera.position).normalize());

var intersects = ray.intersectObjects(targetList, true);

code part of the solution is as simple as the code above. That is it, that is why Three.js is simple handled everything for the user. Intersect array stores objects that is hit by the mouse (which I added object list to the target list so if another programmer wants to add other objects like monster they can easily add another targets).

To hit an object, we need bullet or something more space-game-like object as laser. Creating a bullet was easy, there was just a ball that goes through the direction of the mouse. But creating a laser was a bit more complicated. Which can be examined in the code file LASER.laserbeam.js and LASER.laserPros.js. But the program has a problem that when I throw the laser (or bullet): laser object and monster object cannot intersect. I intersected with the mouse which works fine but I could not intersect two objects position. This is probably the ray changes the position as laser moves toward the z direction.

## TO-DOs

* Improving the movement of characters.
* Hitting object when intersection of the bullet and monster
* Rigid body collision with loaded object and bullet (As I searched from the internet, it is a common issue that ammo.js has some problems with loaded objects, maybe the ammo library can be improved)
* Tracking.js causes frozen screen when works with WebGL. (Probably this problem also needs an improvement of the library)

# CONCLUSION

In this project, I tried to learn and master on the WebGL. Some parts of the project did not happen as expected but it was a nice and educational to make a project with a new language and new API. As promised the continuous game play and character creation works well but intersecting the bullet/laser need improvements. Also, I had some surprise when tracking.js caused frozen screen when it is embedded in WebGL content. For now, I do not know, is it because of a bottleneck effect for my system or a debug in tracking.js or three.js but this debug is added to TO-DO list.

# References

Developer, M. (tarih yok). *developer.mozilla.com*. Mozilla Developer-WebGL: https://developer.mozilla.org/en-US/docs/Web/API/WebGL\_API adresinden alındı

Parisi, T. (2012). *WebGL: Up and Running.* Sebastapol, CA: O'Reilly.