**KF6018 Computer Graphics and Animation**

**Group Report**

**Student 1: (Ella Tijssen W18022237)**

The first role for the assignment covering lighting, shadows, material, and 3D object placement, transformation, and organization. Implemented elements:

1. A ring-shaped world with textured stone mountains on the inner and outer border
2. Boulders of random placement and size on the outer border to make the mountain more uneven
3. Tree trunks placed randomly within a circular boundary along the circumference of the ring and avoiding the center so they do not spawn inside mountain or outside the ring
4. Given each tree trunk three tree leaf blocks placed randomly yet relative to the tree’s position
5. Red blocks falling randomly from the sky
6. Physics added using cannon.js provided in workshops and found online at [1]
7. Optimized leaves falling down from trees relative to the tree trunk position with randomized opacity
8. Shadows added to objects
9. Lighting function to display shadows
10. Camera movement using OrbitControls.js found online at [2]
11. Applied texture to mountain objects, requires a local server to show

The ring-shaped land has two boundaries with stone textured mountains on each. The outer side is bounded by a toroid-shaped mountain which has random boulders on the interior side and a smaller toroidal mountain marks the border of the inner side of the land. For the physics of the environment, it was preferable to have the physical plane be ring-shaped as well so objects could fall off the edge or through the center of the ring, however the difficulty in trying to apply this using cannon.js meant the idea was discarded.

The red boxes were originally applied as a test function for the physics during coding however we kept them in to demonstrate the collisions. The physics as a whole is applied using cannon.js. Objects with physical properties are the red boxes, the ground, the tree trunks, and the boulders lining the outer boundary.

The main focus of this world from my implementation was to randomize as much as possible. The trees are placed within a set of parameters that places them within the circumference of the ring and prevents them from spawning outside of it. The boulders are placed randomly along the circumference of the ring yet slightly inward to add unevenness to the outer mountain toroid. The red boxes are placed randomly at start and on reset. Each tree trunk has been assigned three child objects which are placed relative to the tree trunk using the scene graph and given a random position within this.

After the creation of the trees there were some ideas on how to add particles and we had settled on the idea of falling leaves from each tree. The first version of this effect done under Role 2 resulted in a long loading time and leaves did not reset position smoothly. While this was not my role, I felt like this could be far more optimized to reduce the loading times. I modified the code and added a second version of the effect which is the version implemented in the final version of the code. I assigned each tree trunk seven leaf blocks of random opacity and placement, each being placed randomly yet slightly above the tree trunk on the Y axis to create the illusion that they were falling from the leaf blocks, then adjusted randomly along the X and Z axis so each tree had a random placement of leaves falling. In addition, I smoothed out the animation cycle so that rather than all leaves resetting upon one going below –4 on the Y axis, each individual leaf belonging to every tree was checked whether it was below –4 using for loops. This optimized and smoothed out the leaf particle effect and reduced the loading time in the browser, and is the version used for the leaf particles in the final version of the code.

Originally, we intended for the world to not utilize textures and follow a more cartoony cel-shaded look, however due to wording in the assignment brief, I decided to add a texture and bumpmap to the mountain material to demonstrate how textures can be added. These are only visible when hosting the file in a server. The light and shadows of the world were made by creating a spotlight named LightThis and allowing all objects in the world to cast and receive shadows, excepting the road (ringRoad) and ground (ring) which would only receive shadows since the floor would not cast any shadows either way. Camera movement was added using OrbitControls and the controls were modified to use WASD keys instead of the up-down-left-right arrow keys.

Lastly, due to the students in roles 3 and 4 deciding to leave the assignment, the inclusion of a character with skeleton and joints to demonstrate animations and collision detection was no longer possible and as such is not included in the code.

**Student 2 (Matthew Laws W18010859):**

This section of the assignment focuses on graphical and special effects, moving parts and GUI. For this section, the elements implemented were:

1. A day/night system, with a function to toggle between day and night.
2. A cartoon-style outline effect for all objects in the scene, done using an open-source code found online [3].
3. Fireworks that explode in random positions only at night.
4. Leaves that fall slowly from every tree in the scene.
5. A sky-blue skybox that is visible during day.
6. For daytime, some randomly sized clouds that are animated to move slowly across the sky.
7. For nighttime, some randomly placed stars that add variety to the normally plain black background.
8. For both daytime and nighttime, a sun and moon respectively that are animated in the same way to move across the sky.
9. GUI elements – two buttons to trigger two different functions in the code. Done using an open-source library for GUI functionality [4].
10. Text feedback – added an HTML text element at the top right of the screen which counts every time a firework is set off, showing the number of fireworks that have been set off.

For the day/night system, the function toggleDay() was created to change the environment from day to night and vice versa. Doing so changes other appropriate effects and elements in the scene, such as clouds disappearing and stars appearing, the sun being replaced with the moon, the skybox being set to invisible, and lighting changing to a softer, darker ambient light.

The outline effect is enabled via a separate .js file. This file creates a THREE.ShaderMaterial material for each object in the scene. It then uses optional parameters (not used in this assignment because the default values were suitable for the environment) to set things such as the outline thickness and colour. Calling a function to render these outlines, it then renders them onto the objects in the scene, updating as appropriate as objects move and change.

The implementation of fireworks was especially complicated on the animation side, and led to the implementation of two separate functions, one to create the firework ready for animation and then one to animate/explode the firework as the scene runs. createFirework() is called when it is nighttime and when the previous firework has disappeared. explodeFirework() is then called every frame to animate the firework, until one of the particles reaches a distance limit from the original origin coordinate, where the firework disappears. Each firework is positioned in the same y and z coordinate, and at a random x coordinate along a line in front of the camera’s initial position and behind the main environment. The particles of the fireworks themselves are coloured randomly for each firework that explodes.

For the leaves that fall from each tree in the scene, an array of particles (cubes to keep with the cubic style of the trees) was created, each particle having a random transparency and position relative to the tree’s trunk. The particle array is then added using the .add() function to each tree’s trunk, which is how the position of each array can be set relative to that trunk. These particles are then all animated to fall slowly from the tree, resetting to their maximum height after they fall below the y coordinate of –4, maintaining an animation loop.

For the skybox, a simple 3000x3000x3000 CubeGeometry was created with DoubleSide materials of a sky blue colour. This creates the illusion of a sky surrounding the environment. This skybox only appears during the day of course. For nighttime, an array of stars (which are low-detail spheres placed randomly in the sky surrounding the environment) is placed in the sky to add something interesting to the default black background of three.js. These spheres have a spawn limit of 700+ distance from the environment’s origin point (0,0,0) to make sure that no stars spawn inside or close to the environment and are therefore placed in a ‘bubble’ which is set to be limited to the same size as the daytime skybox.

For day, a similar array is used for clouds except these clouds are CubeGeometries with the same height and random x and z sizes. This creates cubic clouds (again keeping with the cubic theme) in the sky during daytime that are then animated to move slowly across the sky. The sun and moon are both simple SphereGeometries that are animated to move across the sky from left to right in front of the initial camera position, resetting back to their starting position after they reach a certain x value.

For the implementation of GUI, two buttons were created and placed in the top right of the screen. The first button allows the user to toggle day and night using the toggleDay() function which is called every time the button is pressed. The second button is similar and calls another function, resetBoxes(), which resets the red boxes created by student 1 to their initial positions. The was done using the dat.gui library [4], which is a very large and popular library for implementing GUI elements into a three.js application.

The HTML text element was a simple creation of a div element in the HTML document (via JavaScript) and then setting some CSS styling to make sure the appearance is easy to read and to set the position. The variable in the innerHTML was then set to update every time the createFirework() function is called.

**References:**

1. Cannon.js for physics created by Stefan "schteppe" Hedman

[https://github.com/schteppe/cannon.js](https://github.com/schteppe/cannon.js?files=1)

1. Camera movement and controls added using OrbitControls.js

<https://threejs.org/examples/js/controls/OrbitControls.js>

1. Outline effect for three.js, *OutlineEffect.js* JavaScript file.

<https://github.com/mrdoob/three.js/blob/master/examples/js/effects/OutlineEffect.js>

1. GUI library for three.js, *dat.gui.min.js* JavaScript file.

<https://github.com/dataarts/dat.gui/blob/master/build/dat.gui.min.js>