SAPIENZA
UNIVERSITÀ DI ROMA

Interactive Graphics

FINAL PROJECT

MAKINWA, Sayo Michael
[1858908]

16.07.2019

Table of Contents

Introduction 2
Hierarchical Objects and Shading 2
Components of the game 2
The Ball 2
The Road 3
The Obstacles 3
Lights 4
Lightening and Theming the Scene 4
The Fog 5
Animations 5
User Interaction and Game Play 6
References 8

Introduction

This project work is a version of an Endless Roller Game that I have adapted [1] and made it akin to Google Chrome's T-Rex game where you race it out for maximum possible points. I have called it "Endless Roadsurf Game".

The game consists of a golden spherical ball that endlessly runs at a constant speed (determined by the difficulty level that the user is able to set) on an abandoned slopy road filled with grown trees and the debris of chopped down trees that make up the obstacles in the game. The goal of the player is to continuously avoid colliding with any of these obstacles for as long as possible. Points increase at a constant rate (also determined by the difficulty level) and the game ends when the ball runs into any of the obstacles on the path.

The game is played completely with the left, right, and up (or spacebar) arrow keys on the keyboard with no mouse interaction.

Hierarchical Objects and Shading

Hierarchical objects are built with sequences of objects, all linked together in a graph. Hierarchical objects are extremely important in 3D graphics, game characters building, robot modeling, to name just a few.

Hierarchical objects are usually represented using directed tree graphs, where each component of the structure is a node, and each except the root node has a parent (and/or a sibling). The object is built by traversing the tree from the root node all the way to the leaf node.

THREE.FlatShading is used as the material for all the components of the game to keep things simple.

Components of the game

The Ball

The ball is created with the *DodecahedronGeometry* three primitive with a radius of 0.22 and *FlatShading* for the material shading. The value of 3 is chosen for the *detail* parameter of the *DodecahedronGeometry* function to achieve a ball that appears a bit less rough, as shown below:

```
var sphereGeometry = new THREE.DodecahedronGeometry(0.22, 3);
```

```
var sphereMaterial = new THREE.MeshStandardMaterial( {color: 0xCCCC00,
        shading:THREE.FlatShading} );
ballObject = new THREE.Mesh(sphereGeometry, sphereMaterial);
```

The Road

The road is made of a giant *SphereGeometry* primitive with a radius of 26. I kept things pretty simple there as I want a really smooth road, however, since I need the road to be marked, I made the vertical segments 30, while the horizontal segments are 500 to achieve the required smoothness. The code is as shown below:

```
var sphereGeometry = new THREE.SphereGeometry( roadRad, horiSegments, verSegments);
var sphereMaterial = new THREE.MeshStandardMaterial( { color: 0xeffbeb,
    shading:THREE.FlatShading} )
roadObject = new THREE.Mesh( sphereGeometry, sphereMaterial );
```

The road is also flanked on the right and left sides by fully grown trees which was created the same way the fully grown tree obstacles are created, except with different values for the height and the positioning, as well as frequency.

The Obstacles

The obstacles are made up of grown trees and dead tree planks on the abandoned road. The trunks of the trees (for both grown trees and dead tree planks) was made using a *CylinderGeometry* while the top green parts that represent the leaves of the grown trees were made using a *ConeGeometry*.

The creation and positioning of these trees was split across different functions because the operations are repetitive. Some of the important parts of the scripts are shown below:

```
//Tree Leaves
var treeLeavesGeometry = new THREE.ConeGeometry( 0.6, 1.0, 9, 6); //value 1.0 is for obstacle
        trees. Only this part changes to a higher value for taller trees on the roadside.
var treeLeavesMaterial = new THREE.MeshStandardMaterial( { color: 0x34fe33,
        shading:THREE.FlatShading } );
var treeLeaves = new THREE.Mesh( treeLeavesGeometry, treeLeavesMaterial );

//Tree Trunks
var treeTrunkGeometry = new THREE.CylinderGeometry( 0.1, 0.125, 1.1);
```

```
var treeTrunkMaterial = new THREE.MeshStandardMaterial( { color: 0x876732,
        shading:THREE.FlatShading  } );
var treeTrunk = new THREE.Mesh( treeTrunkGeometry, treeTrunkMaterial );

//Putting the tree together
var tree = new THREE.Object3D();
tree.add(treeTrunk);
tree.add(treeLeaves);
```

Before the *var treeLeaves = new THREE.Mesh(treeLeavesGeometry, treeLeavesMaterial)* line, the actions coded in the *blowUpTreeLeaves()* and *tightenTreeLeaves()* functions in *game.js* script are applied repeatedly on the ConeGeometry to give the tree some tree-like heirarchical conic shape.

For the dead tree trunks, all the operations about the leaves were not applied at all, and for each obstacle to be created, a random decision is made whether it is to be a grown tree or a dead tree trunk.

Lights

Lightening and Theming the Scene

The game is majorly lit up with DirectionalLight, given a colour that typifies a setting or rising sun, with the position set to look like a sun rising/falling gradually from behind the scene. This light is set to cast shadow, while all the elements on the scene were set to receive shadows, and all except the road was set to also cast shadows. The positioning and colour of the sun, together with the **careful selection of the colours** of the objects in the game, all give the scene a **dark sunset theme** which is quite apealing.

```
sun = new THREE.DirectionalLight( 0xEDE792, 0.85);
sun.position.set(11, 5, -6);
sun.castShadow = true;
scene.add(sun);

//sunlight shadow values
sun.shadow.camera.near = 0.55;
sun.shadow.camera.far = 60;
sun.shadow.mapSize.width = 255;
sun.shadow.mapSize.height = 255;
```

To make the game theme to look lighter (as in daytime or sunrise), an HemisphereLight is added to the scene, and this brighten things up to make a brighter theme, an option which the user selects:

```
if (enableHemiLight) {
        var hemiLight = new THREE.HemisphereLight(0xefebeb, 0x010101, .85)
        scene.add(hemiLight);
}
```

The Fog

Thankfully, Threejs scenes have a *fog* property that is handy in simulating depth or to show a horizon, and in this game, it has been used to make the road appear endless. The colour of the fog has also been carefully selected to complement the colour of the scene and lighting, while also the renderer's *clearColor* value, as well as the *HemisphereLight* colour have been set to be close enough to the colour of the *fog*.

```
scene.fog = new THREE.FogExp2( 0xf1ffe1, 0.15);
camera = new THREE.PerspectiveCamera( 65, sceneWidth / sceneHeight, 0.50, 20000 );
renderer = new THREE.WebGLRenderer({alpha:false});
renderer.setClearColor(0xfefbfb, 1);
```

Animations

The objects that are animated in this game are the road (which is a giant sphere with most parts off the scene to create the perception of a road – the earth is spehrical ☺), and the golden ball itself.

The most primary animation is the rotation of the sphere that makes up the road to create the illusion of the ball rolling on the road. The road sphere is rotated about the x-axis at a certain rate, chosen by the game speed defined by the player at the start of the game. The ball is also rotated about the x-axis, but in the opposite direction, and at a rate that is a function of the rate of ratation of the road sphere. Both of these rotations, in addition to the fog horizon, combine to give the illusion of the ball rolling and moving on the road while the road remains static and is simply being explored by the moving ball. The ball also has some bounce effect added to it as it rolls on the road (the bounce functionality is explained in the next section).

```
var ballRollSpeed;
```

```
var rollSpeed = 0.002;
ballRollSpeed = ((rollSpeed/ballRad) * roadRad)/5;
//Inside gameLoop() function
ballObject.rotation.x -= ballRollSpeed;
roadObject.rotation.x += rollSpeed;
```

Furthermore, it appears as though we get past the trees as the game progresses; the trees are not animated, they are simply "attached to" the road sphere, so as that rotates, the trees also get moved. **This is one important advantage of heirarchical models**.

User Interaction and Game Play

The user interaction was handled by adding event listeners for the keypress of some keyboard keys, specifically the left, right, and up arrow keys, and the spacebar key.

Swerving left and right only consists of setting the presentTrack value to an absolute in the direction of the movement, and then the movement is made continuous using the *clock.getDelta()* and the *THREE.Math.lerp()* function in the *gameLoop()* function which is continuosly called by the *requestAnimationFrame()* function.

```
var presentTrack;
var moveIsValid;
…
function keyPressFunction(keyEvent){
        if (jump) return;
        var moveIsValid = true;
        if (keyEvent.keyCode === 37) {//left arrow key
                if (presentTrack == middleTrack) presentTrack = leftTrack;
                else if (presentTrack == rightTrack) presentTrack = middleTrack;
                else moveIsValid = false;
        }
        else if (keyEvent.keyCode === 39) {//right arrow key
                if (presentTrack == middleTrack) presentTrack = rightTrack;
                else if(presentTrack == leftTrack) presentTrack = middleTrack;
                else moveIsValid = false;
        }
        else {
                if (keyEvent.keyCode === 38 || keyEvent.keyCode === 32){
```

```
                    //jump with up or space key
                    bounceRate = 0.1;
                    jump = true;
                }
                moveIsValid = false;
        }

        if (moveIsValid) {
                jump = true;
                bounceRate = 0.06;
        }
}
var ballBaseY = 2;
var gravityVal = 0.005;
if (ballObject.position.y <= ballBaseY){
        jump = false;
        bounceRate = (Math.random()*0.04) + 0.005;
}
ballObject.position.y += bounceRate;
ballObject.position.x = THREE.Math.lerp(ballObject.position.x, presentTrack,
      2*clock.getDelta());
bounceRate -= gravityVal;
```

As evident from the code above, the ball is made to naturally bounce, and when player swerves right and left, the ball actually "flies" in that direction. **This makes for an interesting gameplay as a player that has not mastered this functionality can tend to lose because of it, while one that has mastered it can take advantage of this flying to avoid dead wood planks lying bare on the road without actually giving the "jump" instruction**.

After getting this user interaction down, the next important thing in the gameplay how points are computed and how the end of the game is determined.

Points are tied to the interval between addition of trees to the scene, which makes sense because it is like getting points for each obstacle in the way. However, these points are not explicitlly coded to be a reward for avoiding these obstacles, but they are added in advance, more like paying in advance for the task to be done. The decision to add them in advance is so as to have a continuosly increasing point since trees are added from a posion far ahead off the scene and they scroll to the scene, and they are also added at an

equal interval, so the points increases at the same rate until the player loses. The interval between addition of trees is set at *0.5* and this value is multiplied by a value, depending on the speed the player selects, hence, the player gets more points with increase in game speed.

```
var treesIntervals = 0.5;
var pointsMultiplier = 4;
if (clock.getElapsedTime() > treesIntervals) {
        clock.start(); //restart clock each time
        points += pointsMultiplier * treesIntervals;
        pointsText.innerHTML = userFriendlyPoints(points);
}
```

Collision with obstacles is detected with the absolute distance of the ball from any of the obstacle. For each passing tree, this distance is checked, and when it is below a certain value (*0.7*), the *gameIsOver()* function is called; this function calls the *cancelAnimationFrame()* function with the game ID to stop the *gameLoop()* call, and then displays the gameover page with the points reached and an option to go back to the landing page.

References

1. Tutorial on Creating a Simple 3D Endless Runner Game Using Three.js by by Juwal Bose on EnvatoTuts+