SAPIENZA UNIVERSITY OF ROME

Interactive Graphics

Final Exam

Fruit Collector

Presented by:
Cecilia Aponte 1822225
Huriye Kayis 1802952

Supervisor:
Prof. Marco Schaerf
23 July 2018

SAPIENZA
UNIVERSITÀ DI ROMA

1. Introduction

1.1 Project

In this project, we have created a fruit collector as seen in Figure 1. This fruit collector is a robot manipulator that has three joints, an end-effector that collects the fruits, a base, and four wheels that help the robot move to the locations when needed. This robot's task is to grab two different types of fruits, oranges and apples, that are located in different positions throughout three trees and drop them in the basket. The scene is set up in 3D with an interface to pick which fruit to collect as two buttons and the availability of movement of the angle perspective (camera) with the user's mouse. During the collecting cycle, the interface shows a description of the steps occurring while in the scene the camera is panning in different angles in order to show all the different perspectives.

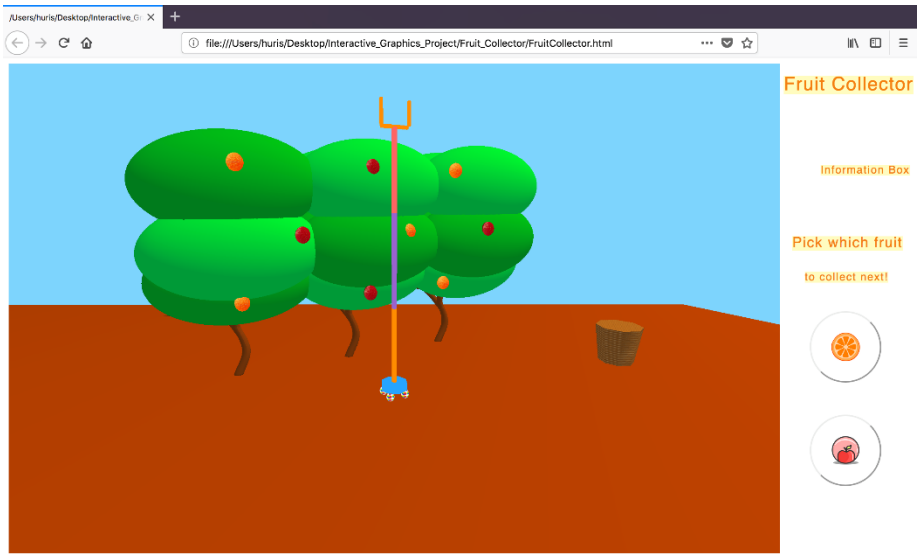

Figure 1: Scene of the Fruit Collector project

1.2 WebGL and Three.js

For the creation of this project, we used basic WebGL with the Three.js advanced library. WebGL is a Javascript API used for rendering interactive 2D and 3D graphics in a web browser. It is integrated into the standards of browsers, so the processing of the graphics is done using GPU from the browser making it faster to process graphics. As we worked through this project, we had to change browsers from Chrome to Firefox since Chrome does not allow the usage of local host files because of security issues. This means that the images used for the texture of the components were not appearing in Chrome, but since Firefox does not have such restrictions we started using this browser.

Three.js then uses the basic WebGL to provide a more advanced library which enables a faster, wider and simpler ways to create. When we started this project, we were using only basic WebGL and even though we went really far in the

creation of the fruits and the robots we started realizing how long it would be taking us to create all the components for our scene, so we looked for methods to create those faster and easier. That is when we ran into Three.js and after learning about the library and using it, we realized the improvement it would provide us. Three.js gave us the ability to create complex 3D animations without the burden of creating things from the basic level for WebGL. Some of the commands as seen in Figure 2 that were used and useful in our project were: SphereGeometry, MeshLambertMaterial, TextureLoader, Matrix4().makeTranslation, scene.add(), .position.set(), .rotate.x, among others.

```
//CONSTRUCT THE ROBOT//
//BASE
var base_geometry = new THREE.CylinderGeometry(50,50,25);
var base_material = new THREE.MeshBasicMaterial({ color: 0x31a6ff });
var robot_base = new THREE.Mesh(base_geometry,base_material);
robot_base.position.z = 0;
robot_base.position.y = earth.getWorldPosition().y + 50;

// WHEELS
var w_distance_x = [-30, 30, -30, 30];
var w_distance_z = [30, 30, -30, -30];
var wheel_geometry = new THREE.SphereGeometry(15, 15, 15);
var wheel_texture = new THREE.TextureLoader().load("strips.jpg");
var wheel_material = new THREE.MeshBasicMaterial({ map: wheel_texture });
wheel_texture.wrapS = THREE.RepeatWrapping;
wheel_texture.wrapT = THREE.RepeatWrapping;
wheel_texture.repeat.set( 4, 4);

for ( var i = 1; i < 5; i ++ )
{ // create 4 set of wheels
    var wheel = new THREE.Mesh( wheel_geometry, wheel_material );
    wheel.name = "wheel_" + i.toString();
    scene.add(wheel);
    robot_base.add(wheel);
    wheel.position.set(
      robot_base.position.x + w_distance_x[i-1],
      - robot_base.position.y * 0.5 ,
      robot_base.position.z + w_distance_z[i-1]);
```

Figure 2: Some of the functions from Three.js used throughout the project

2. Objects in the scene

2.1 Scene Background

The background of the scene seems to be basic, but in reality, it is very important since with this the creation of the rest of the components is much easier. This is because you have to possibility to have a reference point to draw things, otherwise a white canvas makes it challenging to imagine the design. The scene therefore is given a light blue color to be like the sky, as seen in Figure 3. For the ground, a giant cube is constructed so that the entire scene and components would be added on top of it. A cube geometry was constructed for this with no height (y-axis) and a width and length in x and y-axis of 5000 so that the entire scene would be visible in a variety of perspective angels. This cube was called “earth”

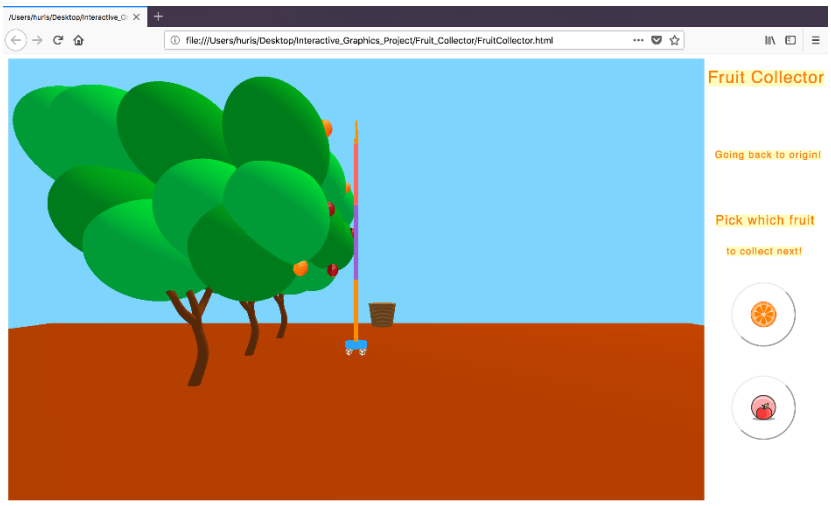

Figure 3: Scene, background, and components shown as part of the project

and was given a brown color and Mesh Lambert material so that there is no shininess, but rather a more realistic material, reflection, and color.

For the component's reflections on their materials to be appreciated, lighting is necessary. Two lights were added to the scene, one ambient and one point light. The point light is mimicking the sun, so the color of the light is given a warm orange color and an intensity of 0.5 to not distort too much the color of the components. This light is set to be, like the sun, far away and high in the sky so it is given a position of 2000, 5000, and 2000 for the x, y, and z-axis, respectively. This means that the sun would be shining above, behind and to the left of the robot location. The other light added to the scene is the ambient light which is created by the sun as the rays are dispersed in all directions equally. For this light, we gave it a white color and an intensity of 0.6 so that it would be clear that the scene is during the daytime.

2.2 Trees

To create the background in the scene, we wanted to add some trees so that the robot would be picking up the fruits from those locations. We tried multiple ways to do this, initially we started with a picture for the background, but we wanted the scene to look more in 3D. To do this, we tried to recreate the environment as a cube and place pictures on all sides of it. However, we found it was not possible to do so and also the overall experience was incomplete as the pictures had to be a perfect sequence so that the scene would look natural. Therefore, we decided to just create the background scene ourselves in 3D.

In order to do this, we searched for models that we could import and found the THREE.Tree library acquired in this link: https://github.com/mattatz/THREE.Tree. This library includes Tree.js, TreeGeometry.js, TreeHelper.js and creates tree trunks and branches. It does so by creating the tree trunk with using values inputted by us for the length and radius, as seen in Figure 4. Then we also define other values for the branches, like the number of branches to build for the hierarchy where we picked 3 so that these would not be sticking out of the tree leaves; then the radius and height values of the branches are also inputted by us. The building of the tree geometry is called and then the mesh is created with the Mesh Lambert

```
//TREEE
var tree = new THREE.Tree({
    generations : 3,        // # for branch' hierarchy
    length      : 200,      // length of root branch
    uvLength    : 600.0,     // uv.v ratio against geometry length (recommended is gene
    radius      : 20,      // radius of root branch
    radiusSegments : 16,     // # of radius segments for each branch geometry
    heightSegments : 32      // # of height segments for each branch geometry
});

var geometryt = THREE.TreeGeometry.build(tree);

var trex_1 = new THREE.Mesh( geometryt, new THREE.MeshLambertMaterial({ color: 0x8B4513
trex_1.position.set(500, earth.getWorldPosition().y, -500)

var trex_3 = new THREE.Mesh( geometryt, new THREE.MeshLambertMaterial({color: 0x8B4513}
trex_3.position.set(-500, earth.getWorldPosition().y, -500)

var trex_4 = new THREE.Mesh( geometryt, new THREE.MeshLambertMaterial({color: 0x8B4513}
trex_4.position.set(0, earth.getWorldPosition().y, -500)
```

Figure 4: Code showing the use of THREE.Tree library for the construction of trees

material where the color does not shine and is set as brown since this is what a realistic tree would look like. Three branches are created following this pattern and are next to each other in the x-axis.

The next step was to create the leaves for the trees. For this, we created spheres that were later scaled to be elongated along the x-axis. Two different colors were set so that there would be some contrast in the leaves and would look more real. Also, to make a lot more realistic, the Mesh Lambert material was used as this is for non-shiny surfaces, without specular highlights. The meshes were therefore created separated for these two material colors and were alternated its use with the leaf next to it getting the other color. Since there are many leaves needed for the trees, we cloned the meshes and added these to a group to be later added to the scene. These leaves were used for each tree, so they were added to the tree trunk hierarchy. Based on the location of the parent, the trunk, the position was set to be different in the y and z-axis for each leaf. For this, two lists were created for the location in the x and z-axis which contained the locations for each leaf. Therefore, for every leaf the location would be picked as the next index in these lists placing them above and throughout the trunk to mimic a real tree.

2.3 Fruits

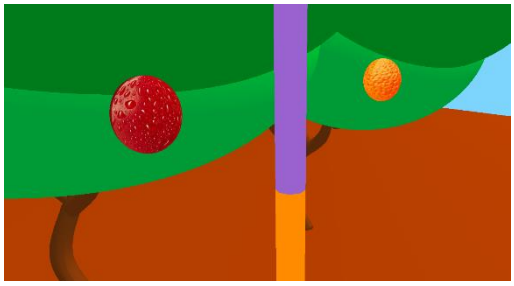

Figure 5: Fruits shown with their respective texture as apple or orange

The fruits were created using the sphere geometry in the Three library with radius, width and height segments equal to 25. The texture of a real orange was loaded using a picture to the geometry, wrapped in both vertical and horizontal directions, and repeated once. This same process was done to the apples, except the picture of an apple was used as seen in Figure 5. To create multiple fruits, we used the same method as done for the tree leaves. For each type of fruit, a name was given by adding the fruit type and its number. The mesh was created with this name and characteristics. The fruits were then added to the scene and set to the position on the tree using the getObjectByName function from the scene. The placement of the fruits was on the front side of the trees where the robot would have access to reach; it was scattered through the three tree leaves and set in positions with varying x, y, and z-axis.

2.4 Basket

For the basket, the cylinder geometry in the Three library was used with radius as the top of 100 and bottom of 75 and the length as 50. The basket was then scaled in the y direction so that it would look taller and the origin was moved to be at

the bottom of the basket (at the floor) by making a translation to the matrix by 100 downwards. The texture added to the basket was that of a real basket and was repeated 5 times throughout the basket geometry and wrapped both horizontally and vertically. The position of the basket is set to be on the floor and to the left of all the trees so as to not interfere with the scene.

2.5 Robot Manipulator

For the robot, various components were added in a hierarchical manner. Those components starting from the bottom to the top are the four wheels to move the robot, the robot base, the first, second and third joints, and ending with the grabber. The base was constructed first as this is the parent and the basis for the rest of the components. It was given a cylindrical geometry with the same top and bottom radius of 50 and a height of 25, blue color, and position zero in all axis except the y where it was raised 50 units to account for the wheels.

To construct the wheels, a similar approach was taken as when the tree leaves were constructed, as seen in Figure 6. The wheels were given a spherical geometry and radius throughout of 15. Texture is loaded to the geometry using a picture of a colorful wheel so that when these would rotate it would be clear to distinguish it. This texture is used as the material and wrapped in all directions and repeated four times. This mesh was created four times using the criteria above for each of the four wheels. These were added to the scene and to the base parent as the child. Its position was set to be on the four corners of the base by translating it in the x and z-axis with 60 units between each and half of the position downwards of the base in the y-axis.

```
//CONSTRUCT THE ROBOT//
//BASE
var base_geometry = new THREE.CylinderGeometry(50,50,25);
var base_material = new THREE.MeshBasicMaterial({ color: 0x31a6ff });
var robot_base = new THREE.Mesh(base_geometry,base_material);
robot_base.position.z = 0;
robot_base.position.y = earth.getWorldPosition().y + 50;

// WHEELS
var w_distance_x = [-30, 30, -30, 30];
var w_distance_z = [30, 30, -30, -30];
var wheel_geometry = new THREE.SphereGeometry(15, 15, 15);
var wheel_texture = new THREE.TextureLoader().load("strips.jpg");
var wheel_material = new THREE.MeshBasicMaterial({ map: wheel_texture });
wheel_texture.wrapS = THREE.RepeatWrapping;
wheel_texture.wrapT = THREE.RepeatWrapping;
wheel_texture.repeat.set( 4, 4);

for ( var i = 1; i < 5; i ++ )
{ // create 4 set of wheels
    var wheel = new THREE.Mesh( wheel_geometry, wheel_material );
    wheel.name = "wheel_" + i.toString();
    scene.add(wheel);
    robot_base.add(wheel);
    wheel.position.set(
      robot_base.position.x + w_distance_x[i-1],
      - robot_base.position.y * 0.5 ,
      robot_base.position.z + w_distance_z[i-1]);
```

Figure 6: Code showing the creation of the robot base and wheels

For each of the three joints, the same procedure was used. The joints were made from the cylinder geometry with the top and bottom radius being 10 and the length 250 except for the second joint that was given 300 as its length. For each joint, the origin was translated to the bottom (start) of the joint so that when a rotation would occur it would move about its own joint base. The material for the joints were created using the Mesh Basic material so that it would be drawn using simple shading so that the color of it would be clear and visible. We also gave a different color for each joint that way it would be clearer which joint would be moving; the first joint was given an orange color, the second purple, and the last joint pink. The joints'

mesh was created and added to the base so that they would be their children. In the Three library, the hierarchy is created by adding the first joint to the parent and then adding the previous joint to the next one. Therefore, the first joint is added to the base followed by adding the second joint to the first joint, and lastly adding the third joint to the second joint. For the location of each joint, its y-axis position was set to be at the end of the previous joint by calling the height of that component.

The last hierarchy added to the robot is that of the grabber. To create the complete grabber, a base and two arms were used. For all the grabber components, a cube geometry was used; for the base the size was set to 100, 10, and 10 for the x, y, and z-axis, respectively while for the grabber arms it was set to 10, 70 and 10, respectively. Together these components look like a “U” where the fruit will later be held. The material for these components were set like the rest of the joints with a Mesh Basic material and set to the same orange color. The grabber base is added to the hierarchy to joint 3 and the position to joint 3’s height and the arms to each corner of its base and slightly higher to that of half the grabber’s base. Similarly, the grabber arms are added to the hierarchy to its base.

Lastly, all these components are added to the scene so that it will become visible.

3. Other Implementation Requirements

3.1 User interaction

Fruit Collector has a friendly user interface, as seen in Figure 7. At the left column the actual scene is placed next to user console which is located at the right column. Right column includes basic information for user also the essential control buttons. Information box is used for displaying current action in the scene, to keep user updated with latest moves. Above the information box user can find 2 buttons which are orange and apple button. User can choose initial fruit to collect also the next ones. Once user clicks on a fruit it starts a collecting - cycle that is the process starting from one fruit picked carried into basket and robot located back to its initial position. Since the robot is designed to carry one fruit at a time user supposed to wait for one collecting-cycle to complete.

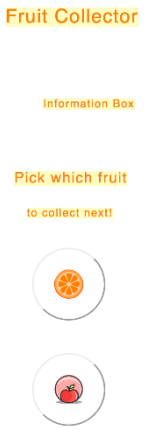

Figure 7: User console with information box and buttons

3.2 Animation

The animation for the project consists of the robot manipulator moving close to the chosen fruit, grabbing it, moving towards the basket, dropping the fruit inside

the basket, and finally returning to the initial position. The buttons for either apple or orange has to be clicked for this sequence to start.

The first step for the animation starts with incrementing the related counter for the fruit chosen by user. The fruit in the sequence is selected by the picked_fruit_generator() function. This function creates the essential bases for the fruit selected. Later by activating the step_1_check boolean animation starts. First robot base is translated according to target fruits location. Once it is aligned latest position for grabber base is stored to later use in the calculate_angle() function. This function takes three points as a parameter and used to calculate angle between 3 points for a given origin point. It returns the result either in radians or degrees. In our case degree is used since it is easier to visualize in our minds. After translation part is finished another part is activated by translation_part boolean. Here the calculated angle is used for rotating the joint_3 around its x axis. This way grabber is able to pick the fruits neatly. When rotation of joint_3 is finished next section starts which is the grabbing the fruit and getting closer to basket.

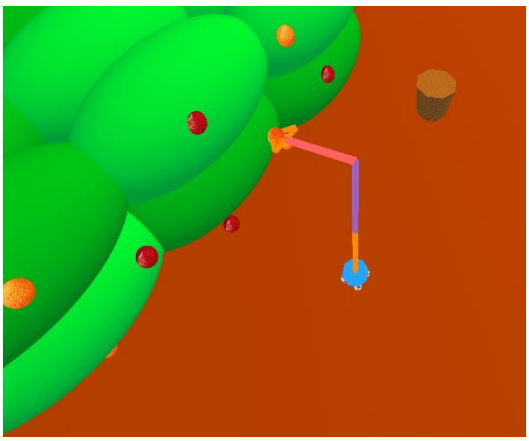

Figure 8: Robot grabbing the chosen fruit

The second step of animation follows the first step, after rotating enough around joint_3, grabber is fully aligned with the fruit in each axis. Next step is to close the arms of the grabber so that the fruit will be covered by grabber, as seen in Figure 8. This is done by rotating the arms equally in the opposite directions. By grabbing the fruit now, it becomes a child of grabber so they move together. Later robot base is translated to where basket is. Here fruit is ready to into the basket. To do that first the arms of grabber is opened, so the parent child relation between grabber and fruit is cancelled, fruit is free to move by itself. While fruit translates through the basket it also rotates around itself this way it becomes more realistic and interesting to see, as seen in Figure 9. Once the fruit translates enough to go inside basket's bottom free fall part is finished. While the fruit animates robot waits for it to complete animation.

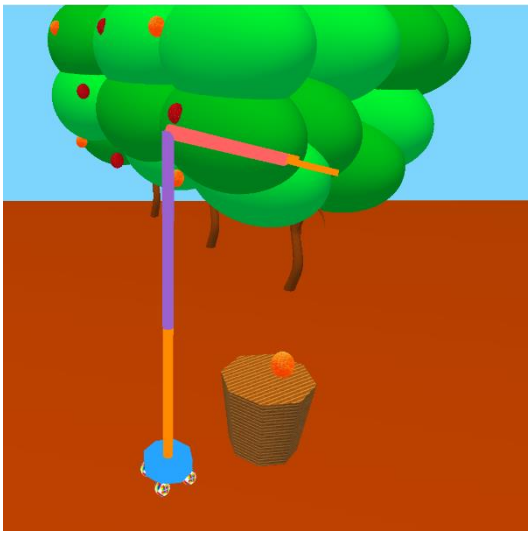

Figure 9: Robot dropping the fruit. Fruit rotates as it falls into the basket.

The third step for the animation is to reconfigure the robot back to its initial position. For this if the third joint that was previously moved is at an angle, the first

step is to rotate it back slowly by 1 degree. The information box in the user console will print this step as “Re-configuring the Robot!” and the camera will move in the z-axis to pan the area scene. At this point, it is important for the user to click another fruit so that the camera is brought back to the scene. This is a point in this current design that can be improved for the future. Then the robot is return to its initial position 0, 0, 0 by shifting the x and z-axis if the base is not in this location. It does this by rotating the wheels in the z-axis and translating the base in the x and z-axis whenever they are not in the initial position. The information box in the user console will print in this step “Going back to the origin!” as the robot moves back to the origin.

3.3 Camera Hierarchy

A perspective camera is used in order to give a clearer view of the robot through its animation process. During the its movement robot is changing its configuration many times which creates such good scenes to capture. To be able to display these scenes a perspective camera is used as a child of robots second joint. Because the second joint is at the middle of the robot and gives good opportunities to reach any part of robot. By creating a hierarchy between robot and camera the task of following the robot as it moves becomes more feasible and precise. From the initial configuration to final configuration robot is followed by camera, for example, when robot grabs the fruit from the tree or when fruit is doing the freefall through the basket it gets closer and let user to observe these moments, as seen in Figure 10.

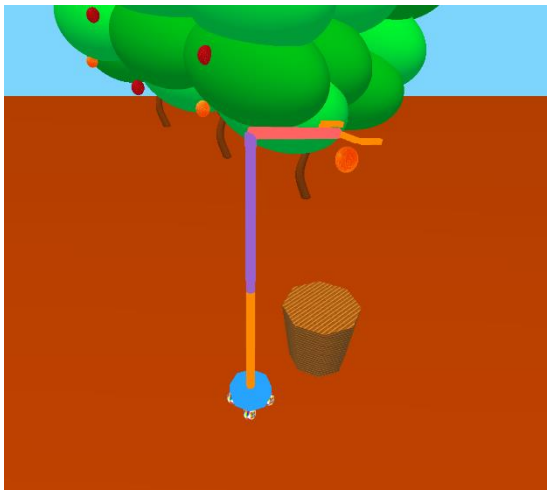

Figure 10: Shows a different perspective of the camera as compared to the positions shown previously.

4. Conclusion

In this project, we have demonstrated a robot manipulator which is designed to collect fruit from trees. To do this, the Three.js library was used to create all the components: background, trees, basket, robot, and fruits. Hierarchy and animation was used to move and rotate the components to create the desired effect. The final version of the code can be found at github: https://github.com/ccapontep/Computer_Graphics_FruitCollector.