CS606-Computer Graphics Mini Project
3D rendering with animation, camera setup, and textures

Urja Srivastava (MT2020037)
Anshuman Galav (MT2020101)
Akshay Gudiyawar (MT2020137)

Abstract

Learning Objectives: Procedural animation of a scene, Modeling a dynamic scene with a Scene Graph and relative transforms, and enabling animation through the scene graph, Generating texture mappings for objects of different shapes, Managing lights and camera, Handling collisions, Basics of a VR application

1 Problem Statement

Build a VR application with a animation storyboard. Design a scene, objects and implement animation using a hierarchical scene-graph. Design the animation in a controlled manner such that a group of objects move in a predefined path with leader deciding the path and other members following it. The path should have both static dynamic obstacles, and when the leader comes close to an obstacle, it changes its path to avoid and move past without colliding. An avatar object should be added, which gives the application VR-like feature. The user controls the movements of the avatar to simulate a VR setting. The avatar can attach itself to a moving object and can perform jump, turning operations while moving along the object. All the objects in scene should be rendered using textures. Based on user control, the textures of objects can be changed. User controls should also enable switching to cylindrical and spherical maps for some of the objects. The scene should be illuminated by multiple lights such as fixed light, fixed but tracking the moving object and a light source attached to moving objects. The scene should be observed by 3 cameras, where one of them is at fixed position observing the whole scene (default), overhead drone camera (movement controlled by user input), and another camera attached to the avatar to simulate it's view of the scene.

2 Solution Approach

We designed a animation scene with two themes, namely space and candy-land. The scene consists of a system of 2 static rotating obstacles, a space-ship fleet orbiting in a defined path and asteroids which also move in different orbits, acting as dynamic obstacles. Additionally, the scene has an avatar which is attached to the moving fleet on user choice. Textures of all the objects (except the spaceship) are changed accordingly to the theme selected. The avatar is rendered as an astronaut in space-theme and as a mario-character in candy-land theme. The application is programmed using ThreeJS graphics library, with output scene rendered on a browser. The avatar objects, space-ship are rendered by importing obj models into the application. Rest all objects are constructed with ThreeJS's object geometry construction module. We have used dat.gui library to take in user input to enable and disable properties of the scene.

3 Scene Animation Design

3.1 Animation scene graph

As mentioned, the entire scene has 2 themes - space and candy-land, in which object's and background's textures are changed accordingly. As per the requirements, we have also designed a group of objects (space-ship fleet) with leader ship leading the path and other ships following it. The fleet has 3 space-ships including leader, which orbits in a elliptical path. The ships were loaded using ThreeJS's obj-loader. Then, we used Group module to add all the space-ships in a single group-object in sequence. This will facilitate applying uniform transforms to all the children/components of the group. The orbit path was implemented by using parametric equation of ellipse:

$$x = x' + r.cos(\theta)$$

$$y = y' + r.sin(\theta)$$

To make them move through the orbit, theta value has to be dynamically changed. To solve this, we used system's clock as θ value for above equations. A satellite is also added as a child to mars object, and we added revolving motion to it to go around the planet in circle.

Both the planets earth and mars rotate around themselves in y-direction, giving the scene more realistic feel. This is done by incrementally setting y-rotation component of the objects while rendering. Similarly, we have designed three asteroids with Icosahedron and Dodecahedron Geometry, of which two of them orbit in a group in y-plane, and another in x-plane. Similar concept of parametric equations is used for these asteroids also with different parameter values. Figure [1] shows the scene in space-theme and Figure [2] shows the same scene with candy-land textures. The themes can be switched with the help of a GUI component on screen.

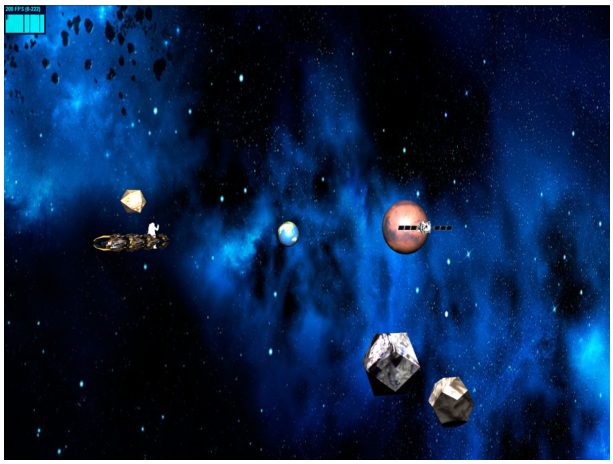

Figure 1: Scene animation with space texture

3.2 Collision detection

To implement collision detection we calculate the bounding boxes of all the obstacles (static and dynamic), fleet of spaceships and the avatar using Three.js library functions. We have constructed a list and pushed these object's bounding boxes to check for collisions. With these bounding box objects we check for intersections between any of the object's boxes, we do this by checking if the moving object box overlaps with any of the other in a loop. And similarly with avatar's bounding box in another loop. If an overlapping is detected, we move the objects (spaceship fleet and avatar) in a particular direction to avoid the collision.

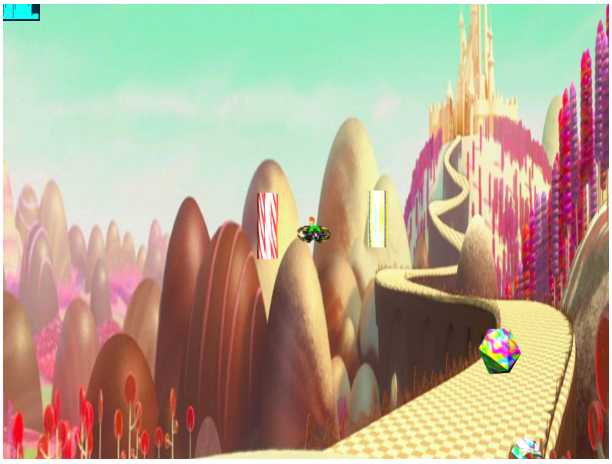

Figure 2: Scene animation with candyland textures

3.3 Avatar attaching to moving objects

An avatar object is added to the scene for giving VR-functionality to the application. With astronaut being rendered for space-theme and mario-character for candy-land theme. These models are imported by objLoader of ThreeJS. The avatar has the capability to attach to the moving space-ship fleet as per the user input. The attachment feature is implemented by adding the avatar object to space-ship group during run-time.

4 Adding Textures

To make the scene appear realistic, we have used cube-map to surround all the objects and give a feel of proper environment. All objects are implemented with textures for their surface appearance, and they can change the textures according to the theme, which is chosen by the user. For static obstacles we use planets earth and mars in space theme, we have used appropriate spherical maps of the planets to apply texture to their surfaces. The space-ship fleet is also rendered with proper normal, and specular maps. The asteroids are rendered with meteorite and asteroid pictures in space theme and candy textures in candy-land theme. The planets are switched to cylindrical texture-mapping for candy-land theme.

5 Adding Light sources

We added 3 backlights to light up the scene. We add a spotlight and set its tracking parameter to the space-ship fleet, this light is our spotlights that tracks the spaceship fleet. The positions of these lights are calculated manually to produce the de-

sired lighting effect. Two lights were added to the front and back of the spaceship fleet as headlights. All these lights can be disabled and enabled on user input. Figure [3] shows the ground-lights switched off, headlight of spaceship switched on with avatar attached to moving spaceship fleet.

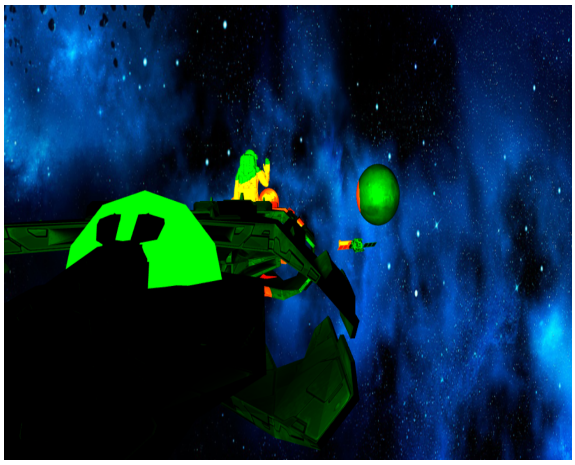

Figure 3: ground lights off, headlights switched on

6 Setting different cameras

We add a camera that looks at the scene from a fixed location, its position is calculated manually. The location is calculated manually. A second camera is added which has fly-controls enabled, this is our drone camera and can be controlled by user using arrow keys. Implementation of Fly controls is given by Three.js library. Figure [4] shows the avatar as viewed by the drone camera.

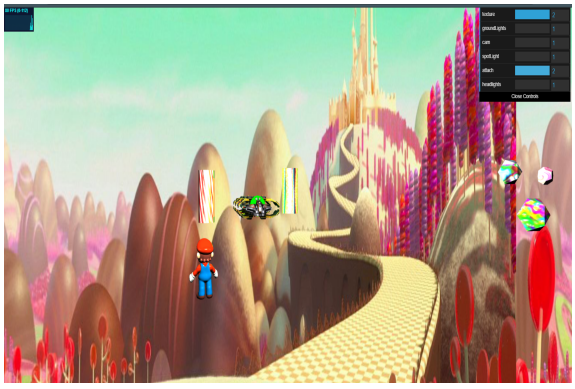

Figure 4: Avatar as seen by drone camera

First person camera (the camera attached to VR object) controls is set up by changing camera's current positions in every frame. Depending on the position of the avatar, the position of the first person camera is calculated. If the avatar is attached to the spaceship then the camera simulates the view from spaceship's perspective and the view is updated as the spaceship moves in space, otherwise the view from the avatar's position is simulated. We have also added keyboard events to control the movement of avatar's object freely in x/y/z directions. This is done for both the texture themes. And the user can switch between different camera views by GUI controls. Figure [5] shows the camera view from avatar's perspective

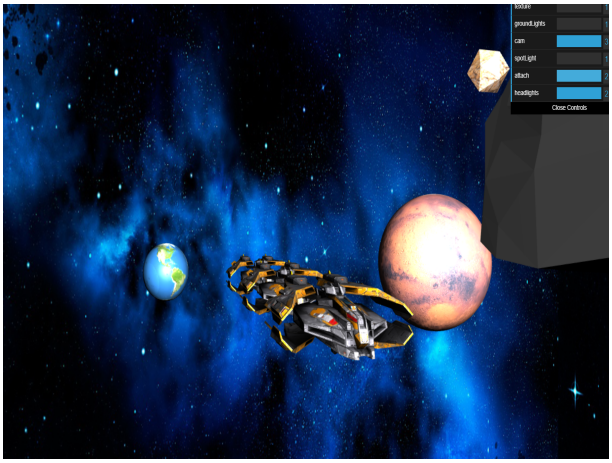

Figure 5: Space-theme from avatar's perspective

7 Conclusion

We were able to make a animation scene using scene-graph and apply motion to the objects, following a pre-defined path. We also learned how to add collision detection avoidance mechanism to moving objects. We rendered the scene with different cameras, specifically included a VR character as an object to the scene. A camera was attached to this VR object to give real-time experience. We experimented with different lights such as static, tracking and moving lights attached to objects. We learnt how to apply different textures to objects and make it appear visually aesthetic. To perform user operations, we added GUI components to control various aspects of the scene.

References

[1] Jos Dirksen, Learn Three.js, Third Edition