CS 405 Computer Graphics

3D Project Part 2

Cavit Çakır

23657

Table of Contents:

Environment Information

How to Run:

VAO Information:
- Wheels:
- Sphere:

Texture Information:
- Mars:
- Sun:
- Earth:
- Yoda(Rovers):

Features
- First Feature: Texture Mapping with Shading blending
  - Screenshots:
  - First Feature Explanation:
- Second Feature: Independent Camera Control with Mouse and Keyboard.
  - Screenshots:
  - Second Feature Explanation:
- Third Feature: One Rover Control with Mouse and Keyboard (A Rover has 4 rotating wheels)
  - Screenshots:
  - Third Feature Explanation:
- Fourth Feature: Two Rovers trying to catch and collide with the user-controlled Rover, if there is a collision the user-controlled Rover stops. Use the AABB for collision detection.
  - Screenshots:
  - Fourth Feature Explanation:

References

Environment Information

Processor: 2,6 GHz 6-Core Intel Core i7
Memory: 16 GB 2667 MHz DDR4
OS: macOS Big Sur 11.0.1

Demo:

https://drive.google.com/file/d/1ftZjtSRzbJTEcNBRIITrS8AEctT-YJ6u/view?usp=sharing

How to Run:

Make sure Assets folder and 3d_part1 in same directory then use following commands;
chmod +x 3d_part1
./3d_part1

VAO Information:

Wheels:

```
GenerateParametricShapeFrom2D(positions, normals, indices, ParametricCircle, 256, 256);
VAO donutVAO(positions, normals, indices);
```

Sphere:

```
GenerateParametricShapeFrom2D(positions, normals, indices, ParametricHalfCircle, 16, 16);
VAO sphereVAO(positions, normals, indices);
```

Texture Information:

In order to run the executable you have to put the following images into the Assets folder. Otherwise you will see error messages.
Image sources: https://www.solarsystemscope.com/textures/

Mars:

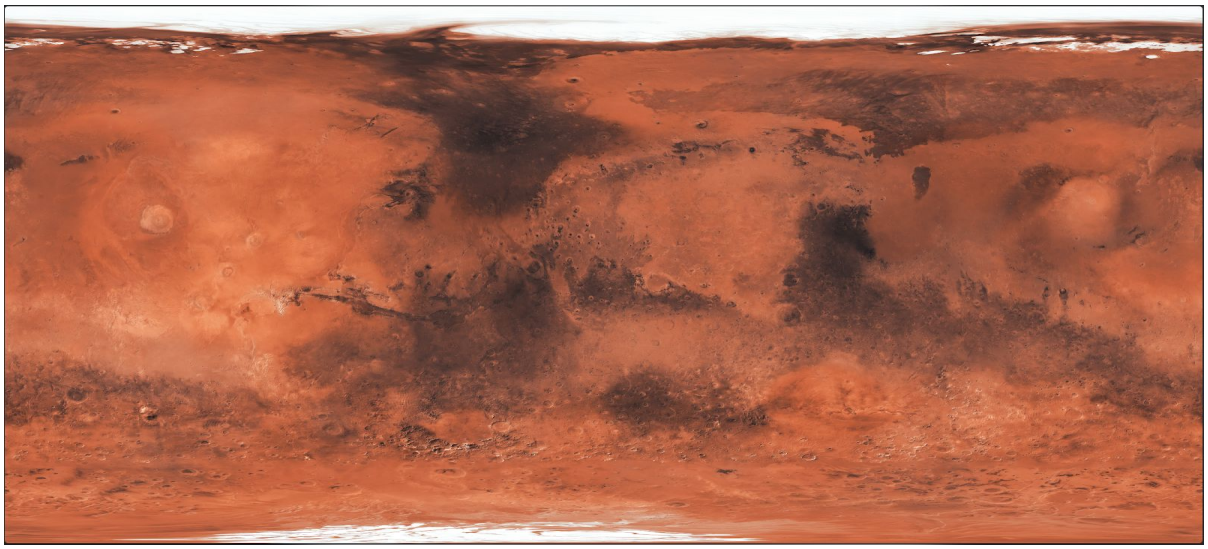


Figure 1

Sun:

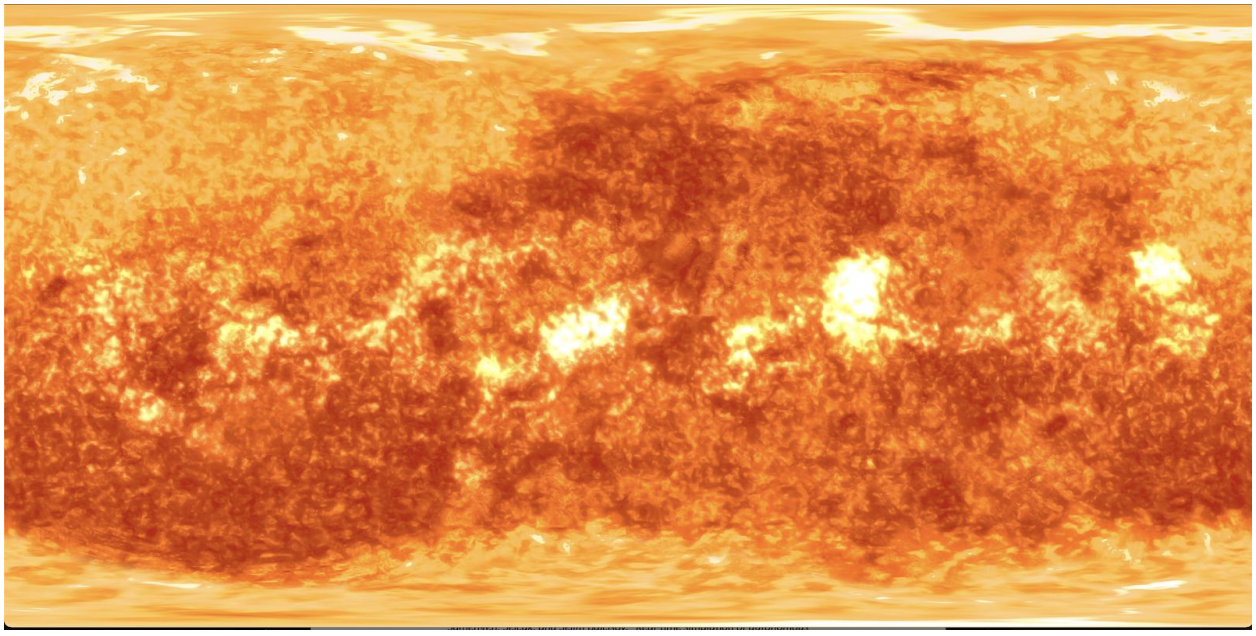


Figure 2

Earth:

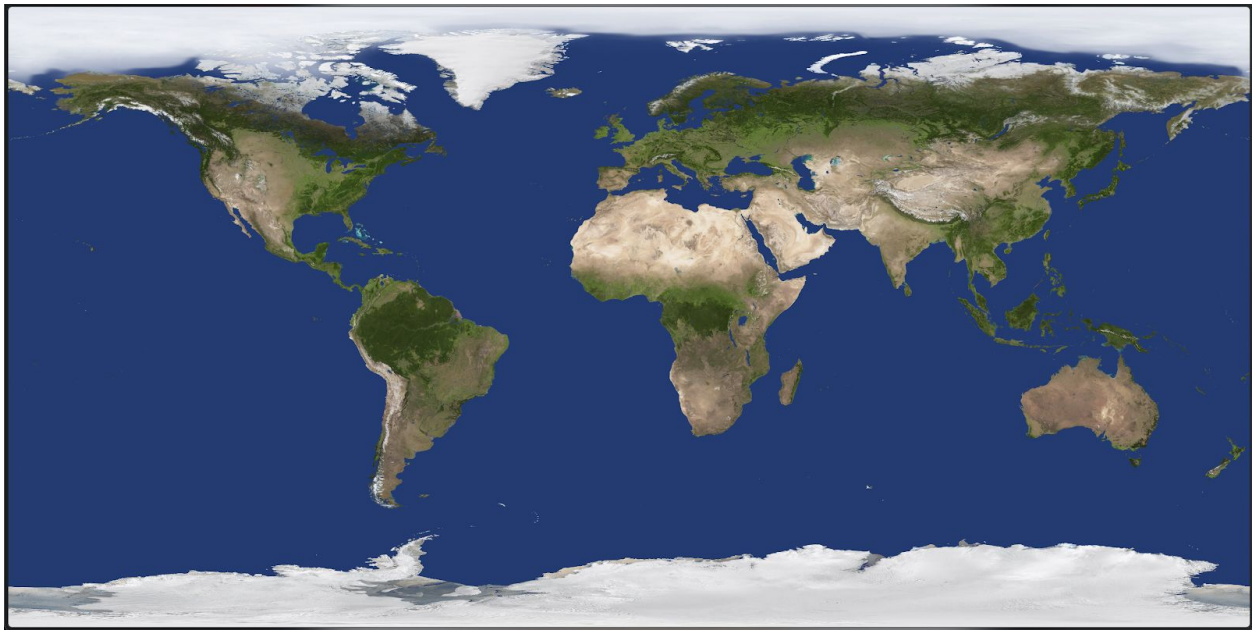


Figure 3

Yoda(Rovers):

Figure 4

Features

First Feature: Texture Mapping with Shading blending

Screenshots:

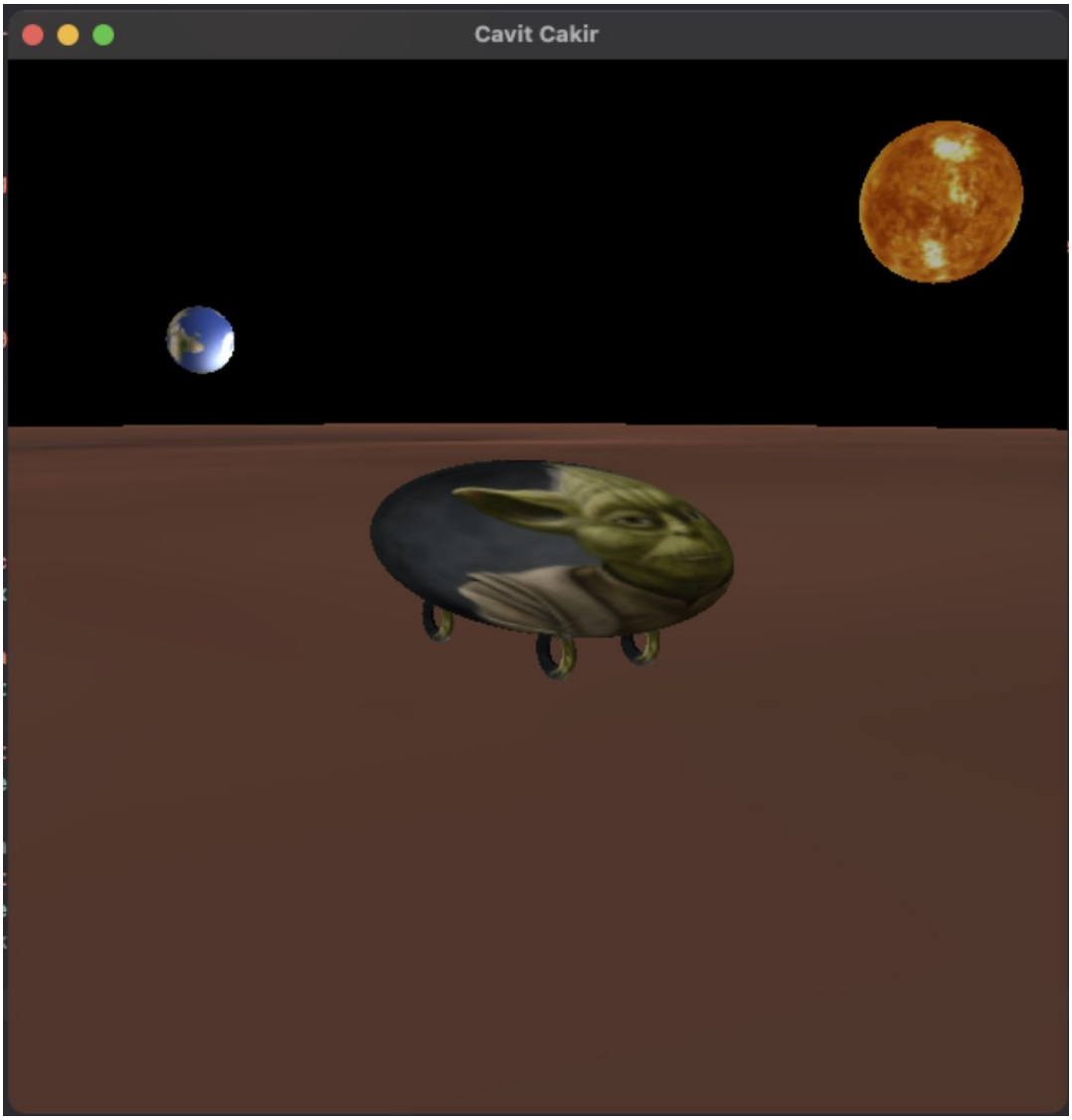


Figure 5

First Feature Explanation:

- As we can see from Figure 5, I wrapped a sphere with Mars texture and added Sun and Earth. The Sun is emitting light as we can see from the shading on the Earth and also from other screenshots. Also, Earth and the Sun are turning around themselves.

Second Feature: Independent Camera Control with Mouse and Keyboard.

Screenshots:

Figure 6

Figure 7

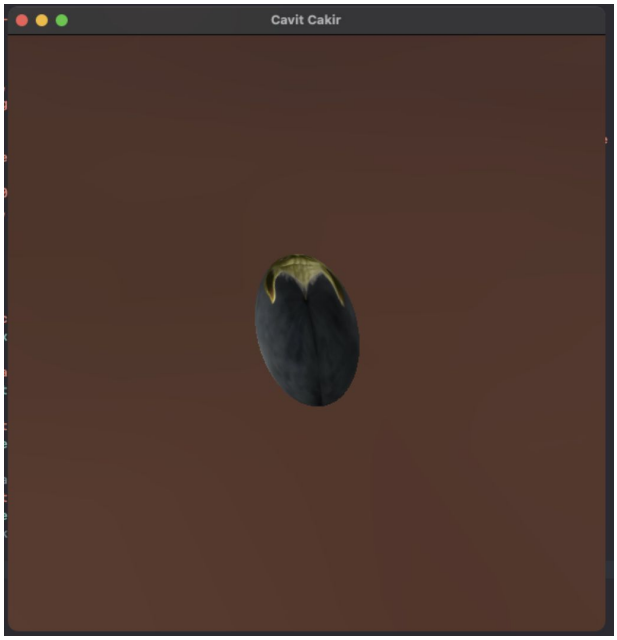


Figure 8

Figure 9

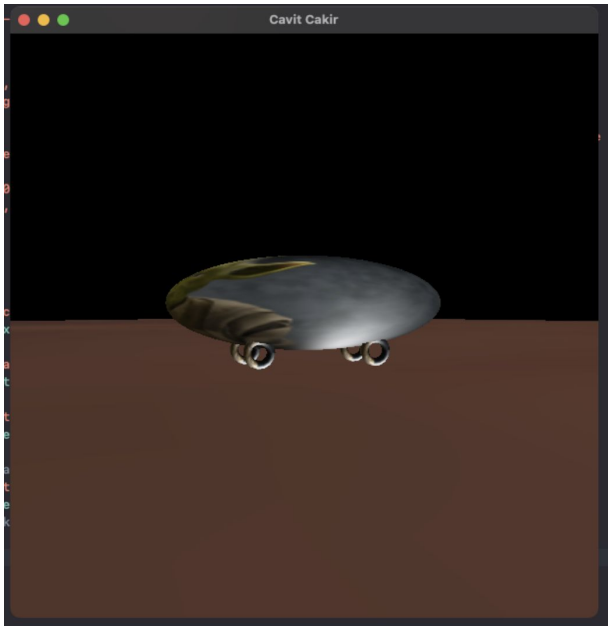


Figure 10

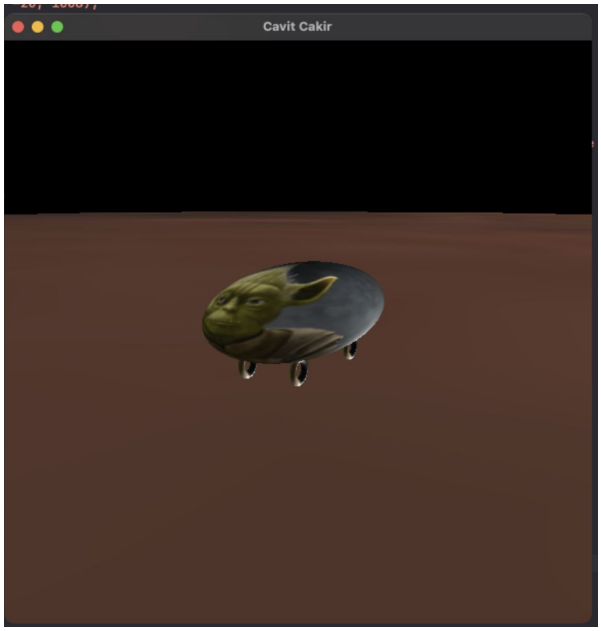


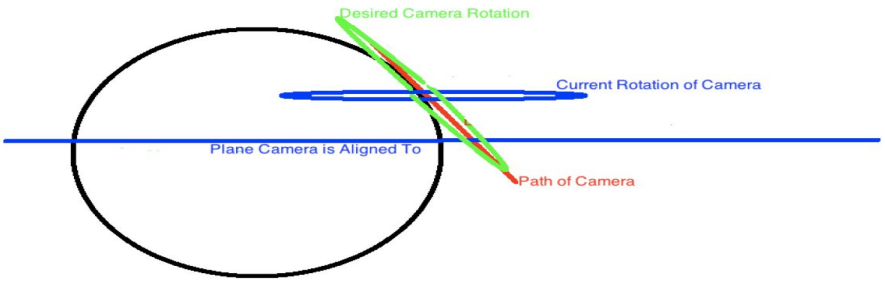
Figure 11

Second Feature Explanation:

- I restricted mouse input with center of screen, with the following if check:

```
if ((mouse_position.x * mouse_position.x) + (mouse_position.y * mouse_position.y) < 4)
```

- I implemented camera control like in a video game called GTA. So, when the mouse moves up, then the camera moves up, and it is the same with respect to left, right and down.
- Also did some restrictions in order to not move inside of Mars. Figures above show some different angles.
- Also I considered the orbital rotation on Mars and rotated the camera's look from, look at and up parameters. I encountered a problem like figure below and I solved the problem with the camera rotation[1].



Third Feature: One Rover Control with Mouse and Keyboard (A Rover has 4 rotating wheels)

Screenshots:

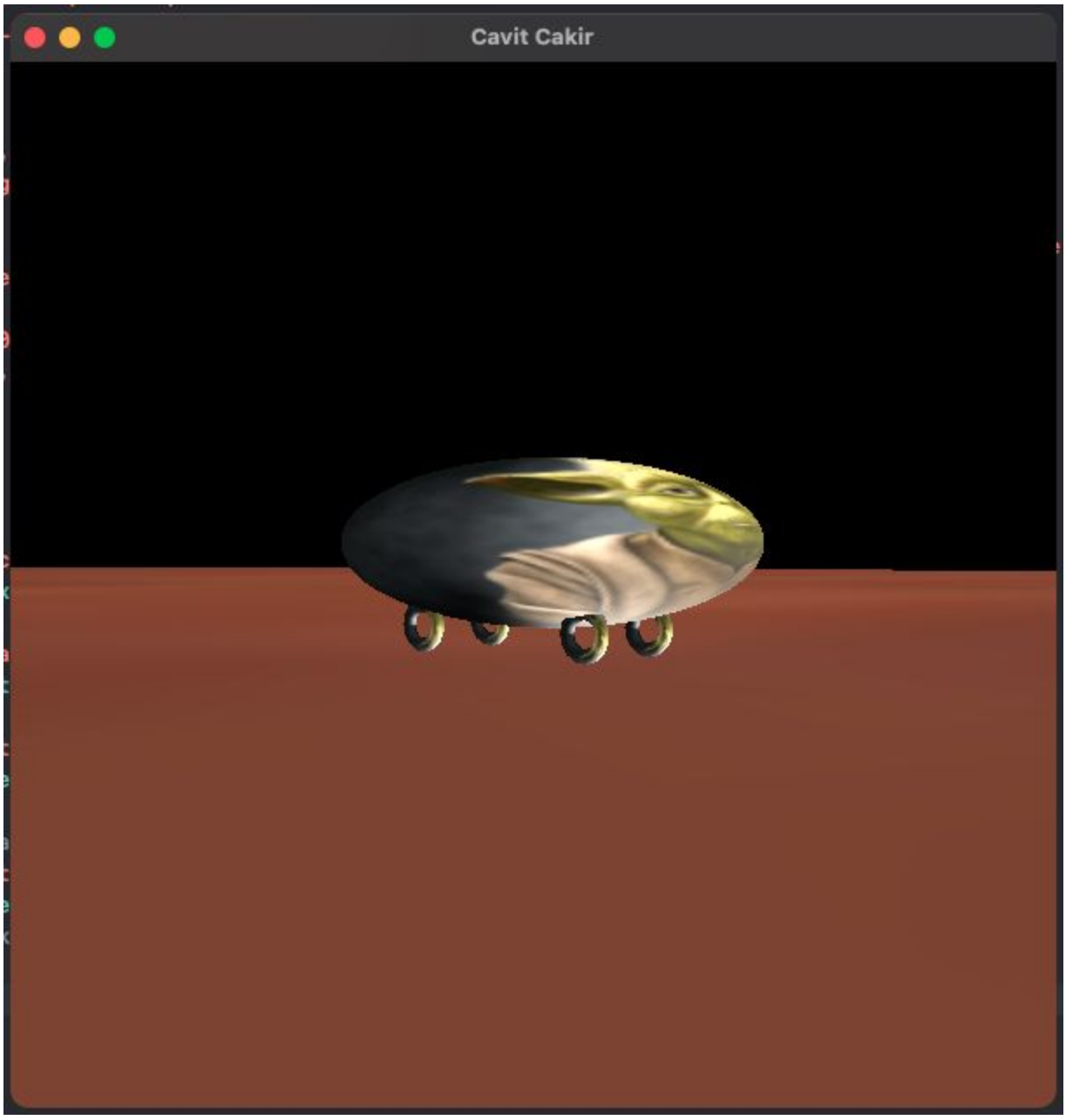


Figure 12

Third Feature Explanation:

- I calculated the rotation and translation of the rovers with an aspect to the sphere shape of Mars. I kept 2 different values for each dimension, First is the real value of that dimension(x,y,z) and the other value is delta(dx,dy,dz) which keeps how much our rover moved. So that, I calculated the rotation and transformation with regard to these two variables.
- I did the rotation with these calculations:
  - `auto roty = glm::rotate(glm::radians(float(user_rover_dx)), glm::vec3(0, -1, 0));`
  - `auto rotx = glm::rotate(glm::radians(float(user_rover_dy)), glm::vec3(1, 0, 0));`
  - `auto newrot = rotx * roty * glm::vec4(user_rover_x, user_rover_y, user_rover_z, 1);`
- I did the translation with these calculations:
  - Calculated new positions of user_rover_dimension then;
  - `transform = glm::translate(glm::vec3(user_rover_x, user_rover_y, user_rover_z + 0.1));`
- Controls are also like GTA video games, W for going forward, S for backward, A for left and D for right.

Fourth Feature: Two Rovers trying to catch and collide with the user-controlled Rover, if there is a collision the user-controlled Rover stops. Use the AABB for collision detection.

Screenshots:

Figure 13

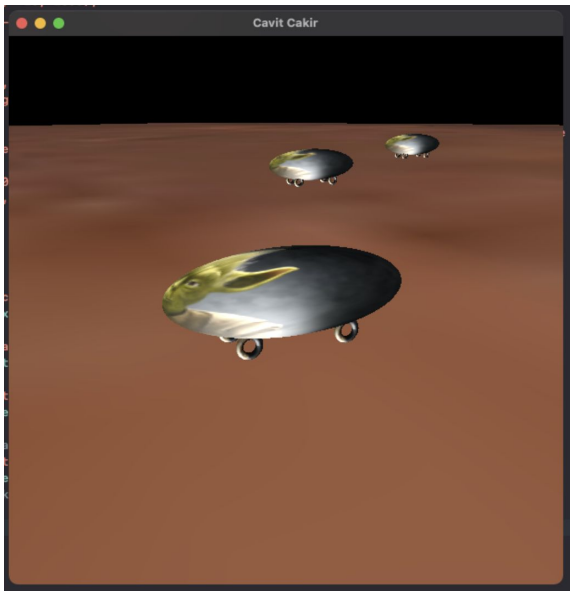


Figure 14

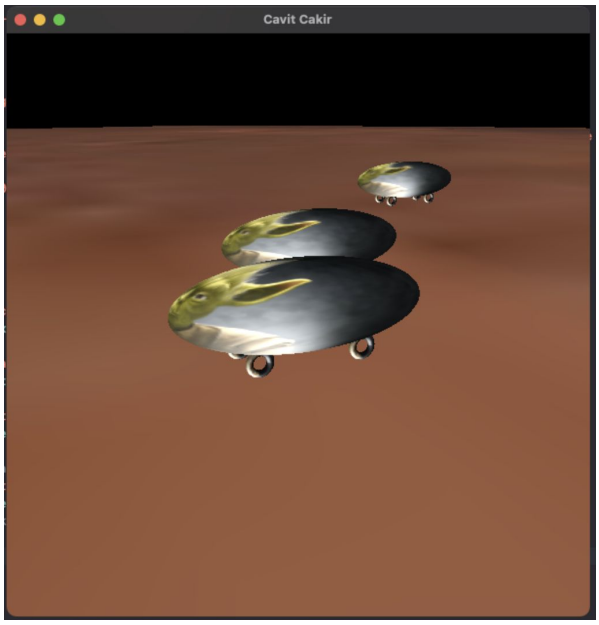


Figure 15

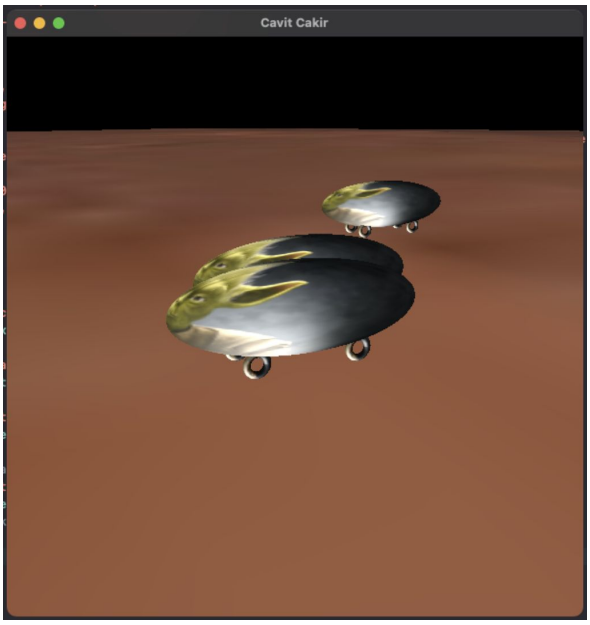


Figure 16

Fourth Feature Explanation:

- I created 2 follower rovers using the code block from 3d project part 1. One of them is faster than the other. They are also traveling in an orbital line. If any collision happens, the user rover could turn the wheels but cannot move. If the user presses K then the user rover teleports toward and continues moving.
- For the AABB check, I first checked if there are any collisions in x-axis then y-axis. If both axises collide then I can say that there is a collision.

References

[1] https://gamedev.stackexchange.com/questions/69431/camera-aligned-to-sphere-surface