Computer Graphics

Faculty of Engineering

University of Adelaide

Assignment 4

Student: Deangeli Gomes Neves ID: 1651398

Student: Deangelo Gomes Neves ID: 1651400

Date submitted: 10 June 2014

Introduction

First person shooters games (FPS games) are a type of game which is played for several people around the word [1]. Usually in this type of game, players control a character that is holding a gun in its hands while walk through of the scenario. The player's view, it is possible to see the hands and the gun as if the player was the character (see Figure 1). Thus, enhance the experience of game immersion to players. In FPS games, players can use weapons to shoot non-player characters (NPC) in the game. Furthermore, they also can jump to reach other platforms of different heights, walk through the scenery, drive vehicles and further. This report describe analyse a C++ code which implements a tiny number of FPSG's features.

Figure 1. Players' view in FPS games (HubsPages, 2009)

Source code

The code developed by the group tries to implement some common features found in FPSG.

The code creates a skybox and inserts some objects within it. There are spheres floating in the air, just to be clear the effects of lighting on skybox. There are spheres floating in the air, only to facilitate visualization of the lighting effects on the skybox. This skybox creates a snowy environment. The moonlight comes from the top of the box. The player is able to move freely in the environment. The player can use a car to move more quickly from one point to another. Each group member was assigned to work in any part of the code. The text below is discussed the contribution each member to implement this code

Deangeli Gomes Neves

MyMesh.cpp: This class are contained in all models. The purpose of this class is to provide freedom for the programmer to draw one or more objects just using a instance. To achieve this, almost variables in this class are vectors which allow the programmer store data without hindrances. The class encapsulate several commands used to store data in graphics cards; to generate the attributes, uniforms and textures id(s); and to read .obj files.

SkyBox.cpp: class is used to load and draw a skybox in the scene. To load an instance of skybox.cpp is necessary to send 5 images as input parameters where each one is mapped in different faces of the cube, except the bottom face.

MouseEventManager.cpp: The class responsible for performing the mouse events. The method "motion" calculates the angles which are used to rotate the cameras. For this calculation, it is assumed that the mouse pointer is always located on centre of the window.

KeyBoardEventManager.cpp: The class responsible for performing the keyboard events.Also this class check the character status. In order to identify when the player is holding a key, it was implemented another method to be used in glutKeyboardFunc();

Shaders: Group members decided to embed the lighting properties of objects and light sources in the "fragment shader", reducing the number of id generated. For simplicity, the group decided to set the same material properties for all objects in the scene. It was written three different vertex and fragment shaders: “sphere-phong.v.glsl” and “fPhong.glsl” which respectively the vertex and fragment shader used by the spheres model; “vlSkyBox.glsl” and “flSkyBox.glsl” which are respectively the vertex and fragment shader used by the skybox; and “vl.glsl” and “fl.glsl” which are respectively the vertex and fragment shader used by the other objects.

Main.cpp: calculation of the camera's position and object positions. Mirror effects (reflection effects). The rotation matrices in order to fix the angle of the gun, car and soldier objects.

Deangelo Gomes Neves

Camera.cpp: This class is used to generate viewer matrix. The main loop contains 3 instances of this class that produce the first-person camera, the sky-view and the third-person view (when the player is driving). There are methods two (cameraMotionCar and addEyeCoordinates) inside this class that implement the collision between the player or car and skybox’ faces.

Reader.cpp: Initially, the class "myMesh" class contained a specific method to read and store data from these types of files. However, some bugs appeared in both method and the members decided to create a separate class for reading .obj files. Basically the class contains variables to store values of vertices, normals, texture coordinates, attributes ids, uniforms ids and textures ids from an.obj file.

Gun.cpp: class used to read a gun model (. obj) and store model's information in graphics cards. It is necessary calculate the gun's model matrix in order to input it in draw method

Soldier.cpp: Class used to read a Soldier model (. obj) and store model's information in graphics cards. It is necessary calculate the Soldier's model matrix in order to input it in draw method.

Spheres.cpp: Class used to generate and draw an object with spherical shape. Applying some matrix transformations, it was possible to draw multiple spheres. Using the "Blend buffer", it was possible to achieve the effect of transparency on the spheres.

Mirror.cpp: Class used to generate and draw a plane. It is applied a "frozen water" texture in the plane, giving the impression that the material has reflective properties.

Results

The program has presented successful results. In the initial scene, the user can see a weapon positioned on the right side hand, globes orbiting the scenario, and a car (see Figure 2). There is a light source inside of the globe which is in the middle of the scene. The user can walk freely through of the scenario using keyboard (keys ‘w’, ‘s’, ‘d’, ‘a’) and mouse.

Figure 2. Initial view

In a part of the scenario, it is possible to see a light on the sky. This light represents the moon, and when the user is under it, the character and the gun are lighted (see Figure 3). The luminosity intensity changes according to the character position under the light.

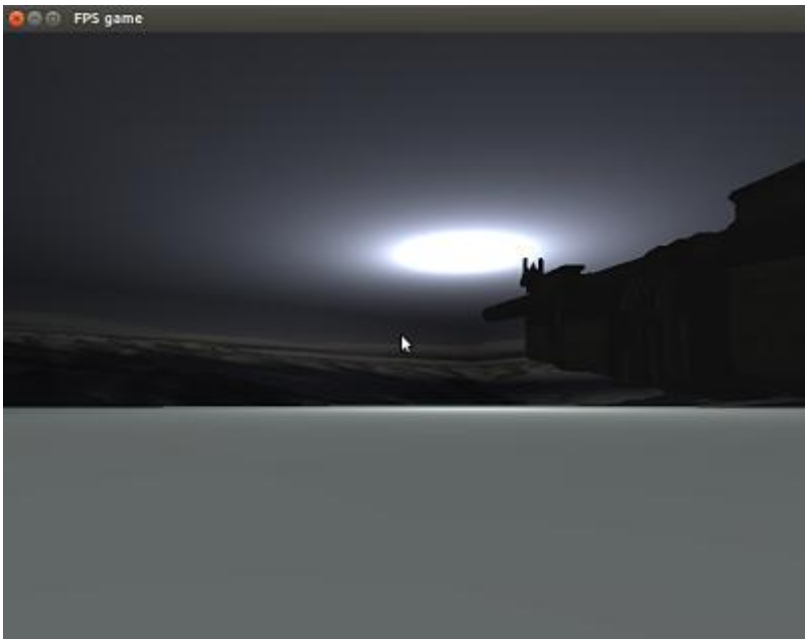

Figure 3. moon on the sky

The user can also see the top view pressing the key “u” (see Figure 4). From the top view, the user can realize that the ground is a huge mirror and it reflects everything in the scene on the floor. the ground in the scenario presents a frozen water surface which justify the reflection of the floor.

Figure 4. Top view

The car can be driven pressing the key ‘e’ near to the car. When this is done, the camera will be positioned on the back of the car (see Figure 5). While the user is controlling the car, the character disappears of the scenario (see Figure 6). There is simple obstacle avoidance in the skybox, so the character and the car cannot leave the scenario. However, there is not any obstacle avoidance for objects inside of the scenario, making able the character to cross through the car.

Figure 5. car view

Figure 6. top view when the user is inside of the car

Although the tiny number of FPSG's features has been implemented, some further features could be deployed such as the inclusion of the NPCs and the implementation of weapon shots. However, the programming level involved in these features is further complex. For example, the inclusion of interactive NPCs requires the inclusion of a simple artificial intelligence. For the future works, it is expected to correct the bug of crossing through of objects and add new FPS games features.

Reference list

J. JANSZ, and M. TANIS 2007, “Appeal of Playing Online First Person Shooter Games”, CYBERPSYCHOLOGY & BEHAVIOR, Volume 10, Number 1, pp. 133

HubsPages 2009, “How to master FPS games”, HubsPages, viewed 10 June 2014, <http://shrek619.hubpages.com/hub/How-to-master-FPS-games>