CG Project Final Report

Student Name: Xinyi OUYANG
Student ID: 20030865

Content

1. Introduction & Design

As discussed in the proposal, the basic idea of this project is creating a scene at a street corner, which should contain house, car, road and trees (showed in ***Figure1***). When actual implementing, it is found too small of this scene. So, in the middle of the project, I extended the scene from the street corner to a town environment (showed in ***Figure2***, which is a screenshot in my interim report). After the interim, other extensions are made and ***Figure3*** is the final scene of my project, which is a town with its outside environment.

Figure 1. Basic idea

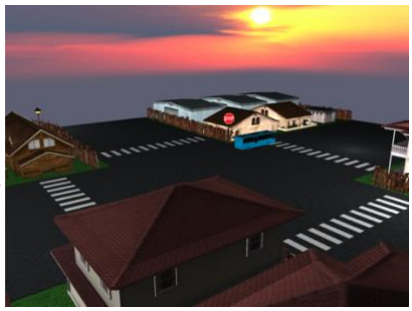
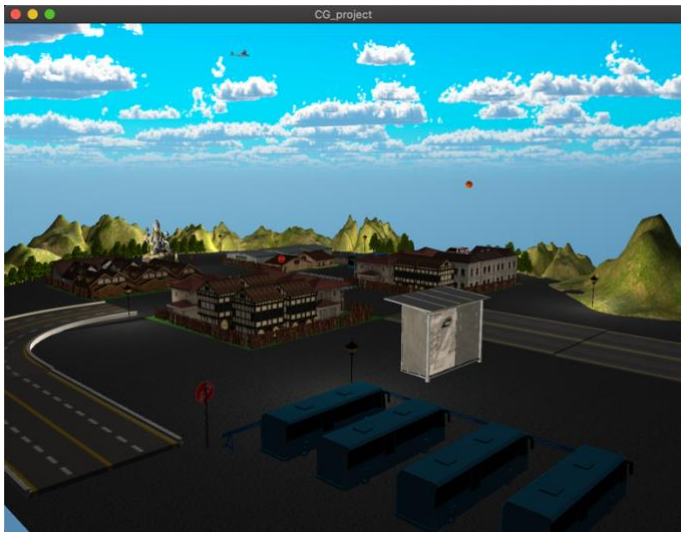
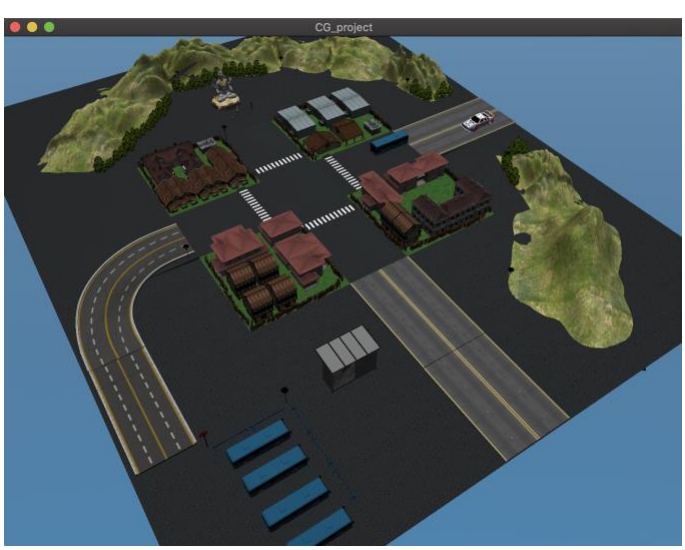


Figure 2. Screenshot in interim report

Figure 3. (the following four figures) Screenshots of the final version





The above ***Figure 3*** shows the full scene of my project at two different viewpoints at day and night (four figures total). The following ***Figure 4*** is the general structure of the scene with its components.

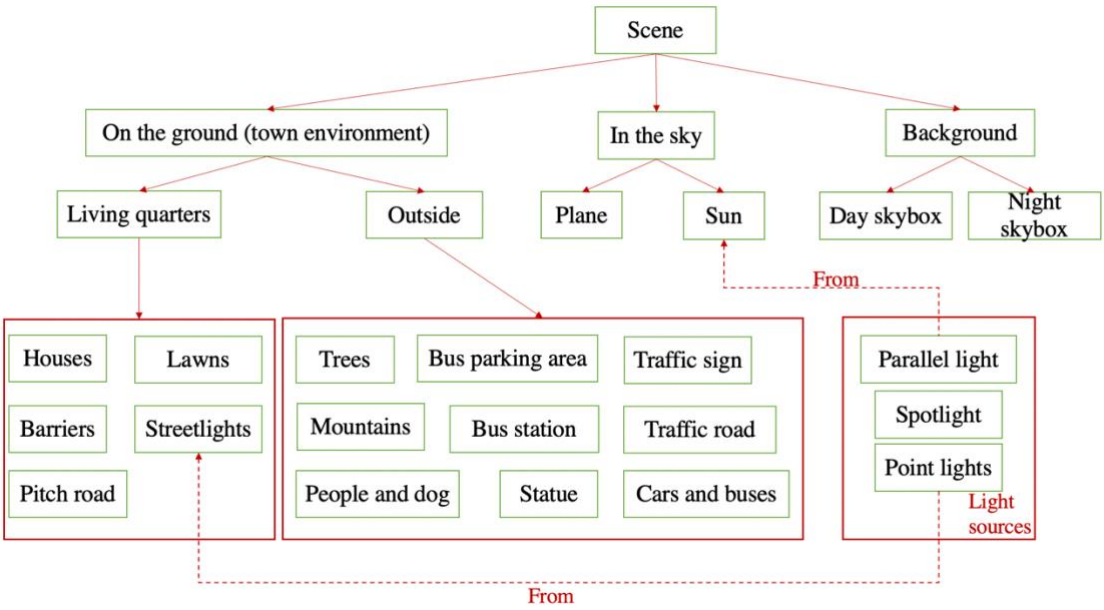


Figure 4. Design

2. Models

1) Imported models

There are many complex models downloaded from Internet such as houses, fences, car, trees, people, dog, statue, airplane and bus station.

The way to loading those models into OpenGL is reading their *.obj* file and *.mtl* file. Each model has its own *.obj* file and *.mtl* file. The *.mtl* file contains the illumination parameters and the textures of each material in the model. The *.obj* file contains its corresponding *.mtl* file, the coordinates and the normal of each vertexes of the model, the vertexes in each face, the related material for each face. The program would read the two files to tell OpenGL every parameters of the model, then a textured model can be loaded in OpenGL. The following Figure 5 is a screenshot of a part pf a *.mtl* file.

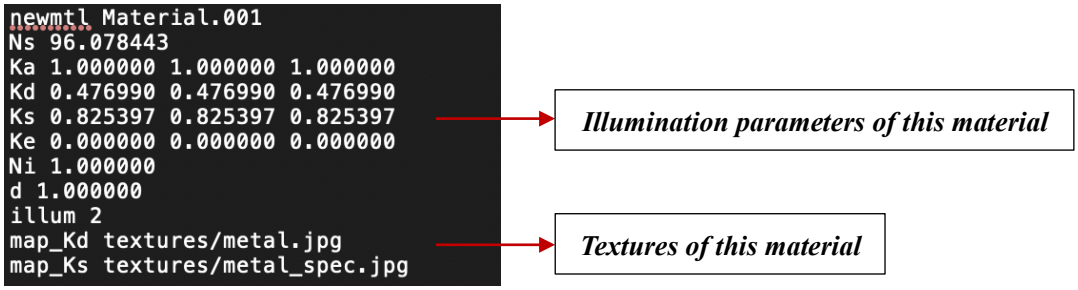


Figure 5. Example of .mtl file

2) Own-created models

There also are some own-created models in the project, such as streetlights, stop sign, the sun, mountains and the ground.

Own-created models are built by a modeling software named ***Blender***. When finishing building, I exported the models from ***Blender*** with its their corresponded files (*.obj* & *.mtl*). Then the external built model can be loaded into OpenGL by reading the two files (same as loading the downloaded models).

The following figures shows two examples of the models.

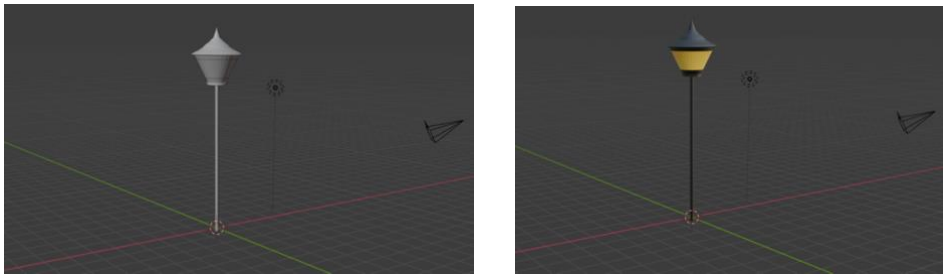


Figure 6. Streetlight model (own-created)

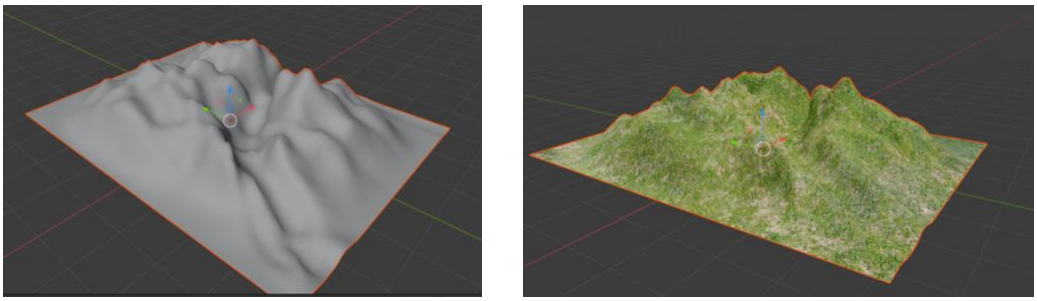


Figure 7. Mountain model (own-created)

3. Texturing

Every model in this project has its own textures and those textures make the models look more realistic. For example, in this project, the pitch texture, grass textures make the ground livelier and the wood texture make the fence more lifelike (those textures examples in my project are showed in ***Figure 8***).

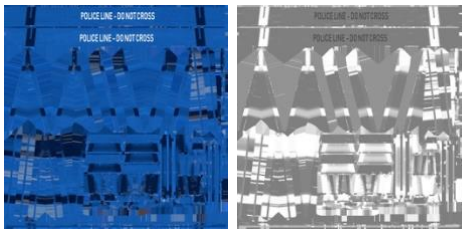
Figure 8. Texture examples

In the real world, some materials like metal can reflect from light while others cannot. To

implement it, besides using the basic diffuse textures, some specular textures should be used for one model. The following ***Figure 9&10*** shows the different texture maps for one model to make some part of the model specular while others cannot reflect from the light.

Diffuse map ***Specular map***

Figure 10. Texture maps for tree model



Diffuse map ***Specular map***

Figure 9. Texture maps for rail model

There are two ways to texture the model. The first way is using *UV editing* in ***Blender***, and the texturing information can be exported as *.mtl* file. ***Figure 11*** are the screenshots of *UV editing* in ***Blender***. The second way is just editing the *.mtl* file of the model. ***Figure 12*** shows the *.mtl* file. We can implement the texture by add lines (*map_Kd/Ks + file path*) at the end of each material information.

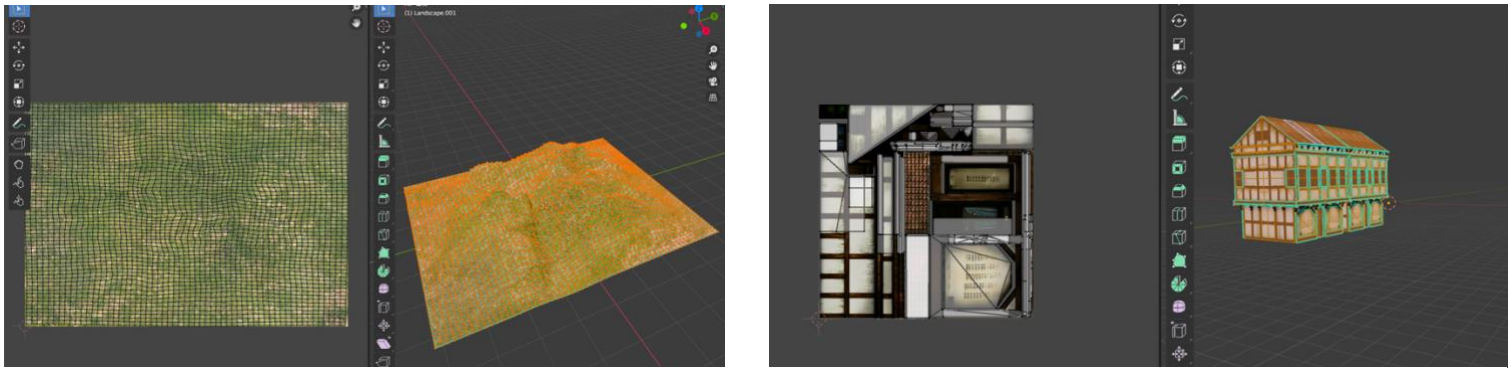


Figure 11. Texturing in Blender

```
newmtl Material.001
Ns 96.078443
Ka 1.000000 1.000000 1.000000
Kd 0.476990 0.476990 0.476990
Ks 0.825397 0.825397 0.825397
Ke 0.000000 0.000000 0.000000
Ni 1.000000
d 1.000000
illum 2
map_Kd textures/metal.jpg
map_Ks textures/metal_spec.jpg
```

Diffuse map and its file path

Specular map and its file path

Figure 12. Texturing by editing .mtl file

4. Transformation

After loading the model into OpenGL, some transformation should be made to set the model at an appropriate place with the suitable size and orientation. The following ***Figure 13*** shows the transformations for the bus in our scene.

```
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, -1.0f, 0.0f));
model = glm::translate(model, glm::vec3(-4.0f, 0.0f, 0.5f));
model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.2f));
```

Figure 13. Transformations for a bus

Since there are too many models in my project, the visualized software ***Blender*** can make the transformation easier. Therefore, besides using the code to implement some transformations, those more complex transformations of some static models are done in ***Blender*** and exported.

And other dynamic transformations are used to implement the animations, which would be discussed in the following section.

5. Animation

There are 4 main animations in this project: cars moving, airplane moving, statue rotating and the sun moving. The dash lines and arrows in following ***Figure 14*** represent the movement tracks for those models.

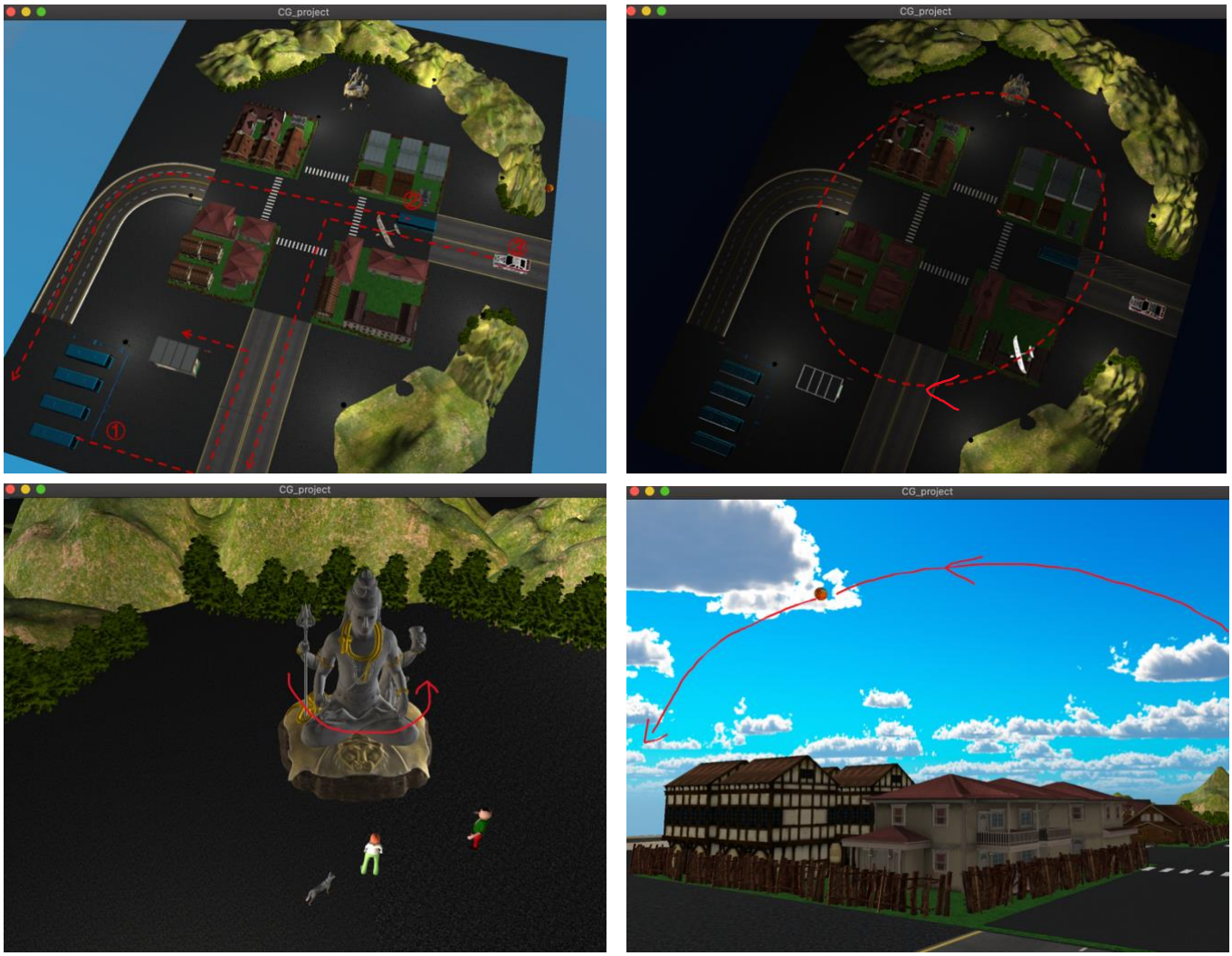


Figure 14. Animations for 6 models

1) **Cars moving**
   As showed in the first figure in ***Figure 14***, there are 3 cars can move along the dash lines. The first bus will move and finally stop at the bus parking area. The second bus will move from the bus parking area and stop at front of the bus station. The third car will move cyclically along the traffic road.
2) **Airplane moving**
   The airplane will fly around clockwise above the town.
3) **Statue rotating**
   The statue in the town will anticlockwise rotate cyclically.
4) **Sun moving**
   The sun will rise from the east and fall to the west.

The following ***Figure 15*** shows an example for the code of the first bus's animation. There are 4 states for this bus to control the bus movement.

```
65      switch(busState){
66          case 1: //go straight
67          {...}
75          case 2: //turn around
76          {...}
85          case 3: //go straight
86          {...}
99          case 4: //stop at the bus parking area
100         {...}
106         default:
107             break;
108     }
```

Figure 15. The code for bus's animation

6. Different Viewpoint

The direction of the view can be controlled by mouse movement and the camera position can be controlled by keyboard input ("W" "A" "S" "D"). The effects of controlling the different viewpoint can be seen in the previous screenshots (most of them are at different viewpoints). The related codes are showed below.

```
57  void Scene::processInput(GLFWwindow *window)
58  {
59      if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
60          glfwSetWindowShouldClose(window, true);
61      if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS)
62          camera.ProcessKeyboard(FORWARD, deltaTime);
63      if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS)
64          camera.ProcessKeyboard(BACKWARD, deltaTime);
65      if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS)
66          camera.ProcessKeyboard(LEFT, deltaTime);
67      if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)
68          camera.ProcessKeyboard(RIGHT, deltaTime);
69  }
```

```
void mouse_callback(GLFWwindow* window, double xpos, double ypos)
{
    if (firstMouse)
    {
        lastX = xpos;
        lastY = ypos;
        firstMouse = false;
    }

    float xoffset = xpos - lastX;
    float yoffset = lastY - ypos; // reversed since y-coordinates go from bottom to top

    lastX = xpos;
    lastY = ypos;

    camera.ProcessMouseMovement(xoffset, yoffset);
}
```

Figure 16. Keyboard input and mouse movement to control the viewpoint

Besides the basic changes of the viewpoint, in my project, user can move the viewpoint to be on the airplane by keyboard input, which would be discussed later in *Section 8*.

7. Lighting

There are three different kinds of lights in this project: point light, parallel light and the spotlight.

1) Point light

The point light would shine in all directions and the light would fade with distance. In this project, those streetlights are set as point lights. And there are 7 streetlights in the scene, which can be seen in ***Figure 3*** (the full scene of the project). The following ***Figure 17*** is one of the 7 streetlights.

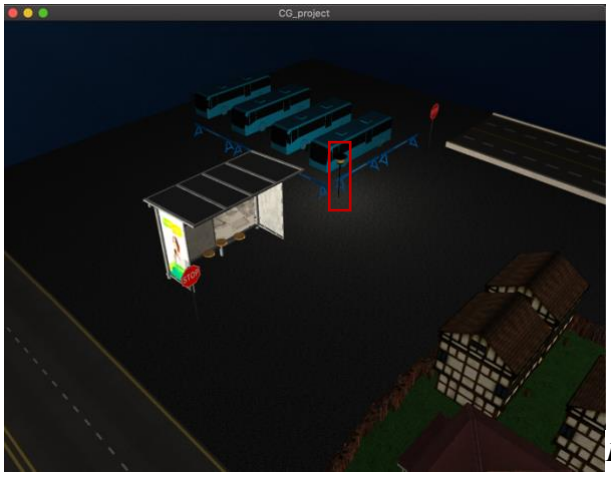


Figure 17. Example effect of streetlight

2) Parallel light

When a light is far away, each ray from the light would be approximately parallel to each other. The sunlight is an example of parallel light. As said before, there is a sun move from east to west in my project. Therefore, to make the scene more realistic, the sunlight direction will be changed by sun moving. The following ***Figure 18*** shows the effect of the different sunlight direction. The two screenshots at the same viewpoint but have different light effect because the sun is at different positions.

Figure 18. Effect of sunlight

3) Spotlight

Spotlight is a light source located at a certain point that shines in a particular direction

rather than in all directions. A flashlight is set in this project as a spotlight. The position and direction of the flashlight will be updated when at the different viewpoint. ***Figure 19*** shows the effect of the flashlight. The right one has a flashlight while the left one does not.

Figure 19. Effect of flashlight

8. Creative Ideas

1) Control the streetlights

User can use key “F” & “O” to control the streetlights.

Figure 20. Streetlights on and off

2) Flashlight

User can use key “M” & “N” to get a flashlight or turn off it. The position and direction of the flashlight will move followed the motion of the viewpoint (by mouse control and keyboard input). The effect of the flashlight is showed in previous ***Figure 19***.

3) Start moving

The airplane and the sun will start move since the program start. But the cars and statue will start moving after user press key “G”.

4) Day & Night

As illustrated previously, there is a moving sun in the project. Besides the sunlight

effect would be changed, the skybox would also be changed when the sun moves to different position. It can implement a more realistic visual effect when at day and night.

When the sun beyond the horizon line, the day skybox will be used, and the sunlight will be set. When the sun below the horizon line, the night skybox would be used and there will not be any sunlight effect. In Figure 21, the left one is when the sun beyond the horizon line and the right one is when the sun below the horizon line.

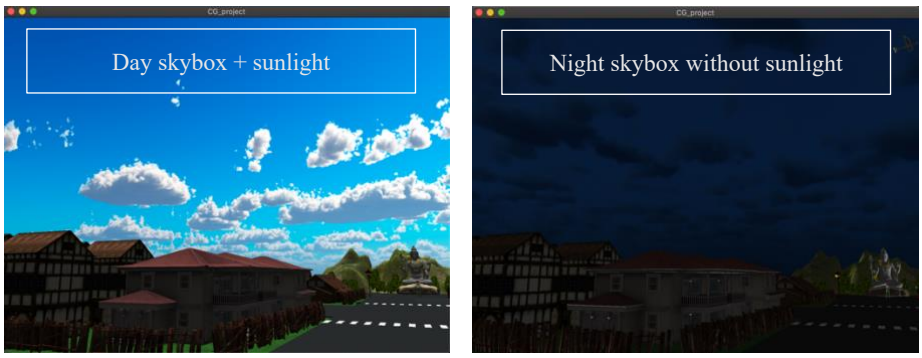


Figure 21. Day & Night

5) Get on the airplane

User can use key "P" & "Q" to get on or get off the airplane. When pressing "P", the current position will be stored, and the viewpoint will be moved onto the airplane, and changed following the airplane moving. When on the airplane, user can overlook the whole environment. At that time, user cannot use "WASD" to control the camera position (because the position will be moved following the airplane move), but user still can use mouse to control the direction of the view. The ***Figure 22*** shows the view when getting on the airplane.

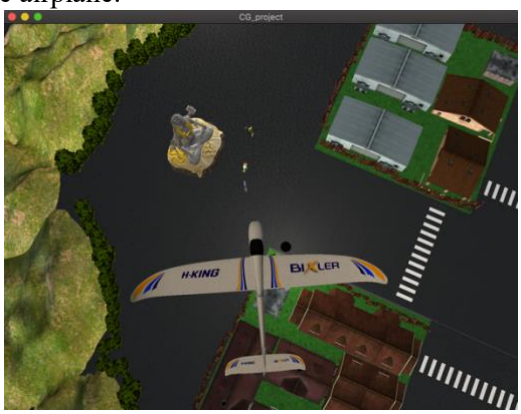


Figure 22. The view when getting on the airplane

When pressing "Q", user will get off the plane, and the viewpoint will get back to the previous position, which is stored before getting on the airplane. And now, "WASD" can be used again.

9. Code Structure

The basic structure of codes refers to the framework-demo-viewing sample on moodle. The following ***Figure 23*** shows the general structure of the project code. The full lines represent the contents of the packages and the dash lines represent the relation between the classes.

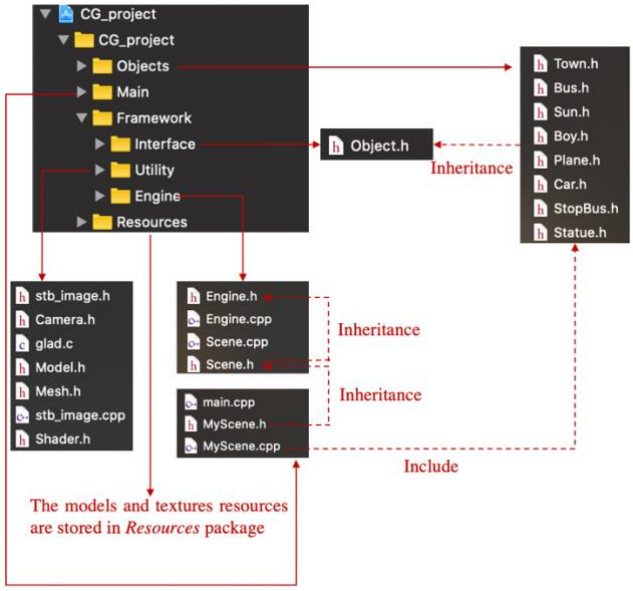


Figure 23. General structure of the code

The contents in *Utility* package are some functional head file. E.g. *Model.h* & *Mesh.h* is used to load the models and textures by reading the *.obj* & *.mtl* files. Other details of those classes can be seen in my code.

The usage of library GLFW are learned from LearnOpenGL.

10. User menu

Setting up the environment:

1) Building library *GLFW* (you can find the install steps here).
2) Setting up *GLAD*. File glad.h has been put into the package *Framework/Utility*, other details can be viewed here.
3) The library *stb_image.h* used to load the images, and the related file *stb_imagee.h* has been loaded into the package *Framework/Utility*
4) Setting up *Assimp*. Assimp is an Open-Asset-Importer-Library, the details to set up can be viewed here.

Running the program:

1) There is no command line operation, user can just run the project in *Xcode*.
2) When start the program, user can use key "W" "A" "S" "D" to move the camera position and use mouse to control the direction of the view.
3) Use key "O" to turn on the streetlights, use key "F" to turn off the streetlights.

4) Use key “N” to open the flashlight, use key “M” to close the flashlight.
5) Use key “G” to make the animations start.
6) Use key “P” to get on the airplane, use key “Q” to get off the airplane.
7) When on the airplane, the camera position is locked, “W” “A” “S” “D” cannot be used. When getting off the airplane, “W” “A” “S” “D” can be used again.
8) Press “esc” to exit the program.