COMP 3069 -- Computer Graphics

Assignment Title: Project Full Report

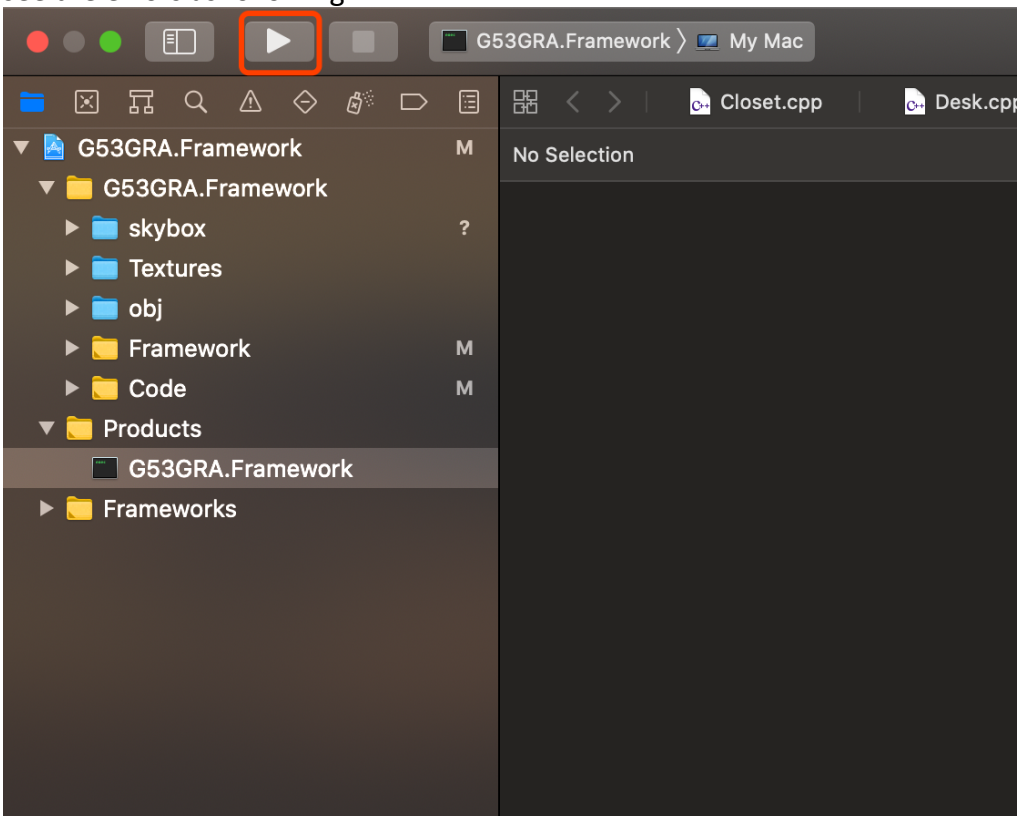
Student Name: Zewei Zhou

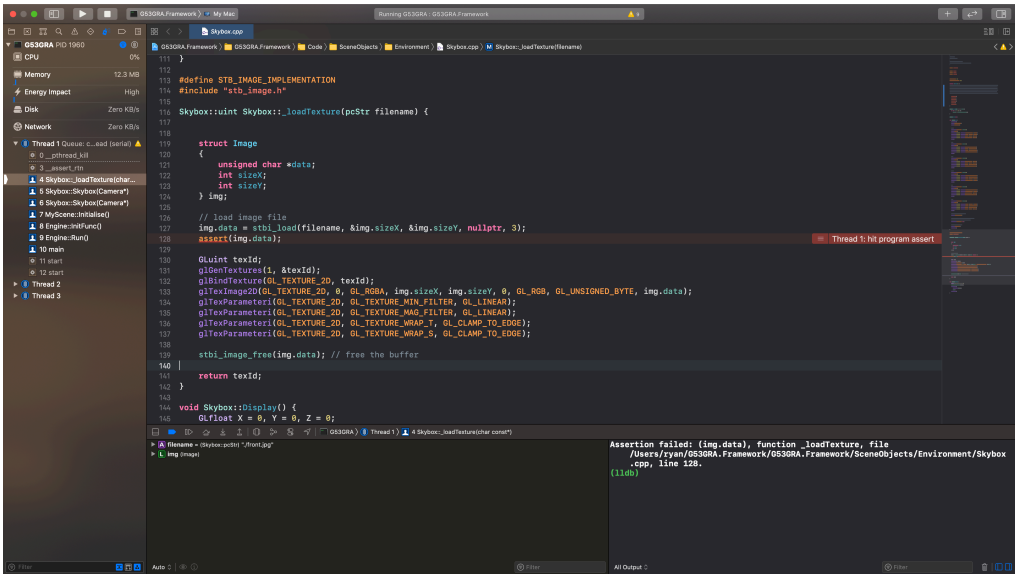
Student ID: 20029947

Due date: 4:00pm, 20 December 2020, Sunday

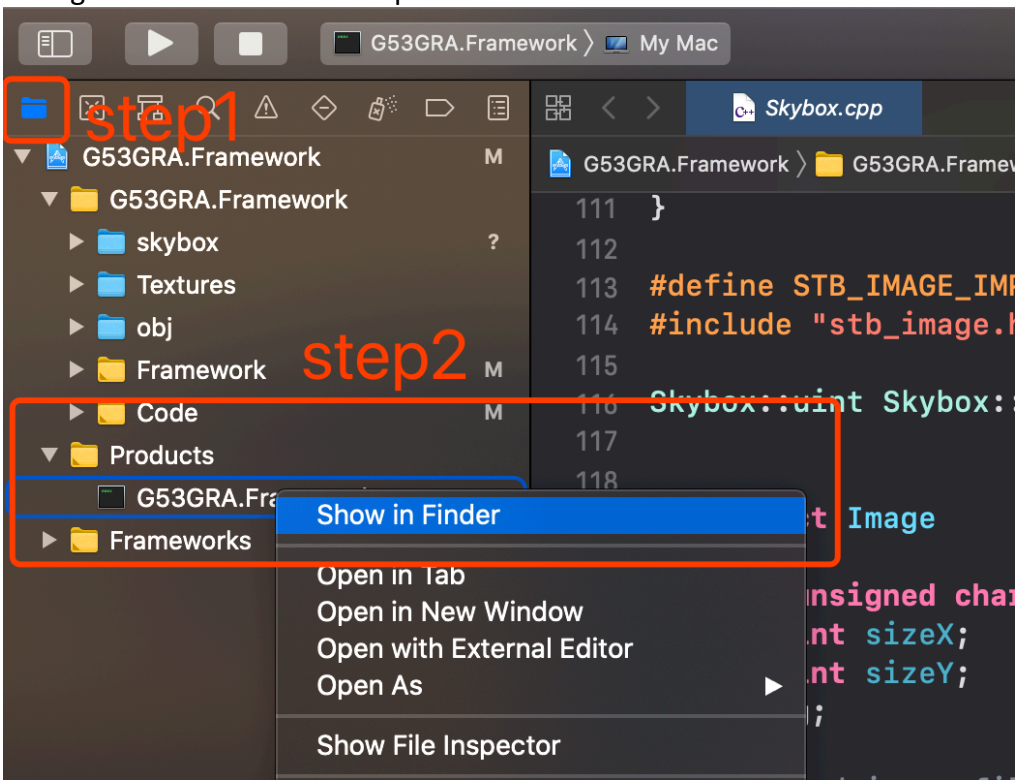
How to run the program

1. **Use Xcode**. I developed the program on a Mac machine by using Xcode. The whole project should not be opened on the computer desktop but opened in 'Downloads', otherwise it wouldn't be open. Then open 'G53GRA.Framework'. It's wired but true. **Again, don't copy the project file to the desktop, leave it in 'Downloads'. Otherwise, you won't be able to open it.**
2. **Copy sources**.
    a. In 'G53GRA.Framework', open 'G53GRA.Framework.xcodeproj' which is a XCode project file.
    b. Run the program. Please be patient since it will probably take about 10 seconds to build the project. It will build successfully but won't run correctly. You will see the errors as following.

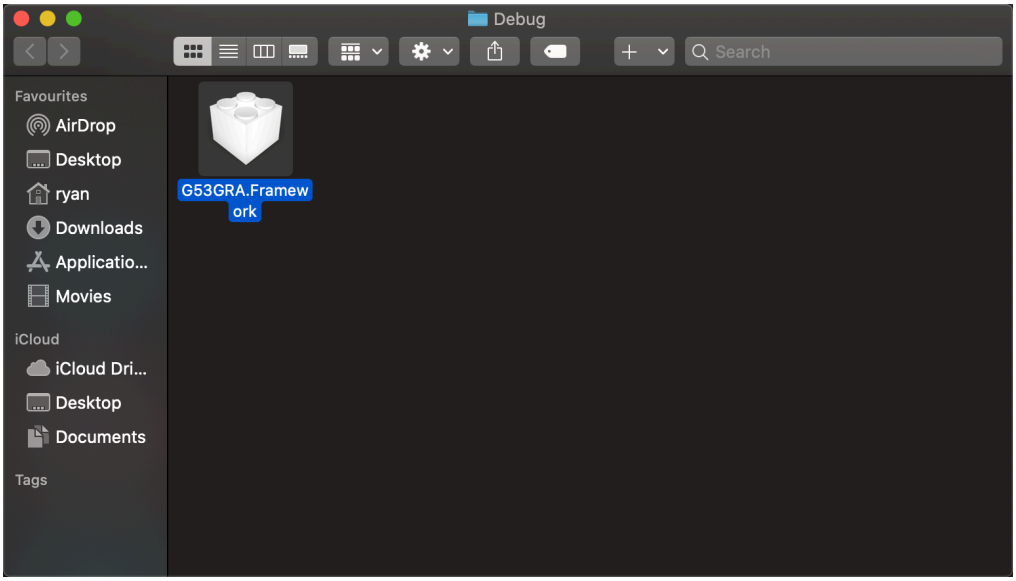


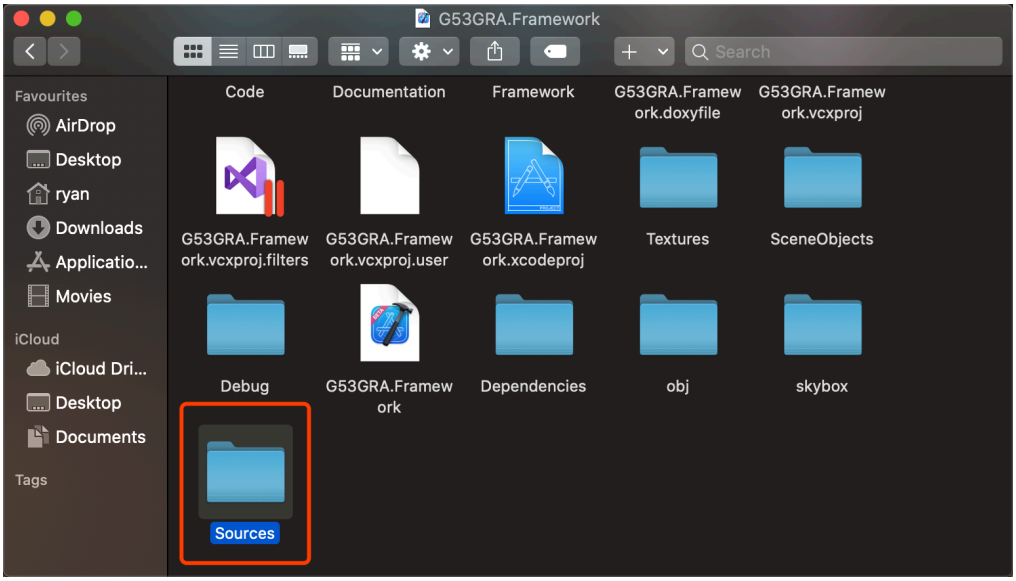


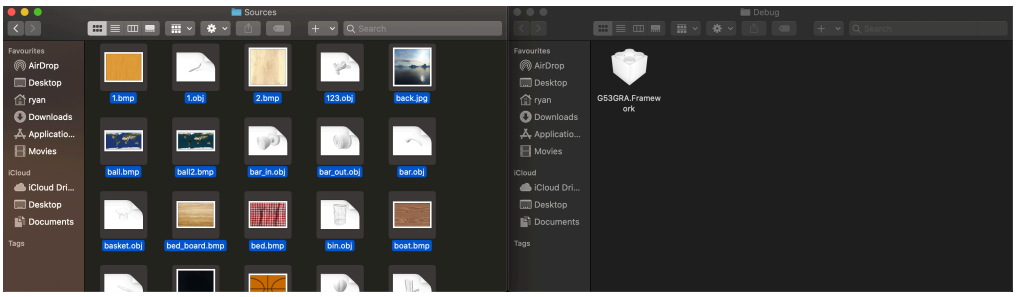
c. Now go to the ‘Product’ and open the file in folder.

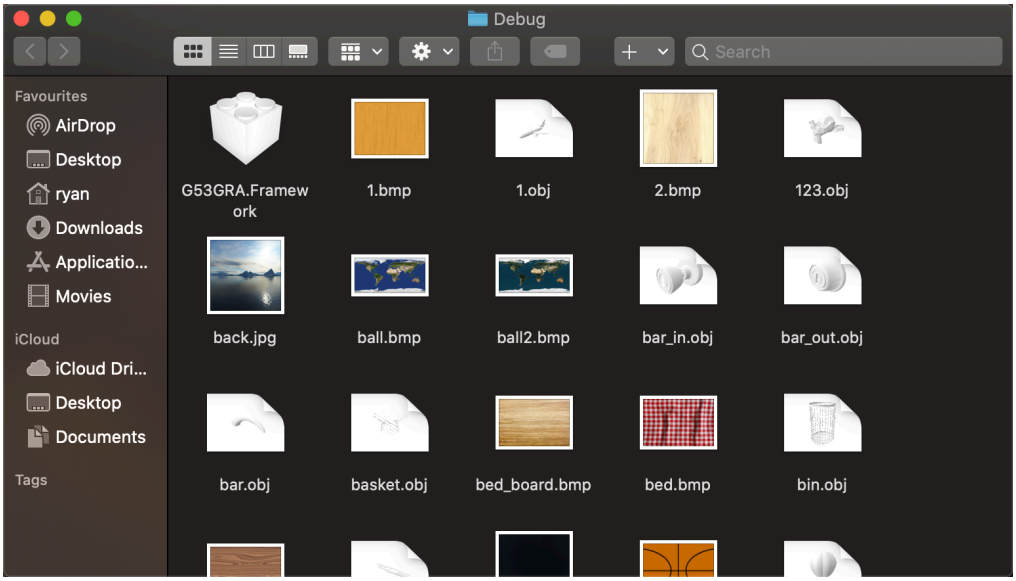


d. **Copy sources**. In the ‘Products’ folder, you will see there is only one file. Now you will need to copy all source files (.files and .bmp files) in ‘Sources’ folder to this folder.


Debug
Search
Favourites
AirDrop
Desktop
ryan
Downloads
Applicatio...
Movies
iCloud
iCloud Dri...
Desktop
Documents
Tags
G53GRA.Framew
ork

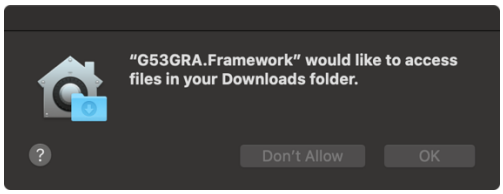

G53GRA.Framework
Search
Favourites
AirDrop
Desktop
ryan
Downloads
Applicatio...
Movies
iCloud
iCloud Dri...
Desktop
Documents
Tags
Code
Documentation
Framework
G53GRA.Framew
ork.doxyfile
G53GRA.Framew
ork.vcxproj
G53GRA.Framew
ork.vcxproj.filters
G53GRA.Framew
ork.vcxproj.user
G53GRA.Framew
ork.xcodeproj
Textures
SceneObjects
Debug
G53GRA.Framew
ork
Dependencies
obj
skybox
Sources


Sources
Debug
1.bmp
1.obj
2.bmp
123.obj
back.jpg
ball.bmp
ball2.bmp
bar_in.obj
bar_out.obj
bar.obj
basket.obj
bed_board.bmp
bed.bmp
bin.obj
boat.bmp
G53GRA.Framew
ork



Now you should be good to run the program.

3. **Run the program twice if necessary.** Since I am suggesting opening the project in 'Downloads', the Mac OS system will probably warn you that you are accessing files in 'Downloads'. Just click 'OK', stop the program and start the program again. Ignore printed warnings and errors.

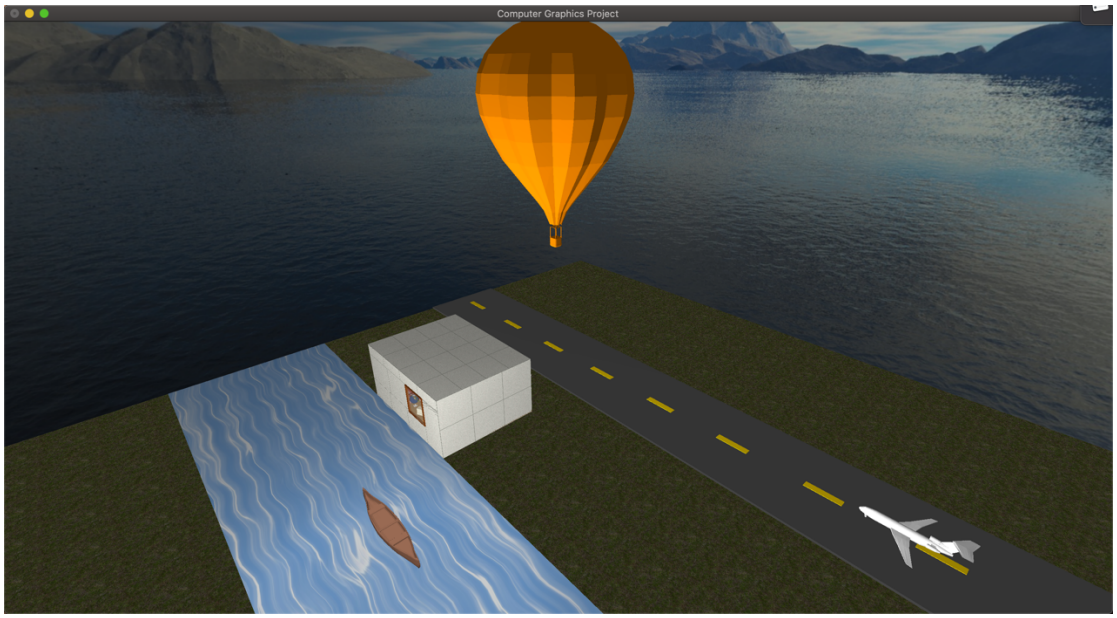


4. **Use keyboard and mouse to move the camera**. 'W': move forward; 'S': move backward; 'A': move left; 'D': move right; 'J': move upward; 'K': move downward; 'space': reset the camera.
5. **Object interactions**: '1'/'2': open/close the door; '3'/'4': open/close the closet; '5'/'6': open/close the drawers; '7': rotate the tellurion; '8'/'9': open/close the window glass; 'e'/'r': pick/throw the dart; 't': play the basketball; 'f'/'g': turn on/off the fan; 'z'/'x': take off / land the plane; 'c'/'v': get in/off the hot air balloon; ','/'.': turn on/off the room light; '['/']': turn on/off the sunlight.

Scene Screenshots:

1. Inside view

2. Outside view



Project Requirements:

1. **Objects** (Object images won't be shown in this section due to the great number of objects)
    a. **Own-created objects**
        i. **Room**. 'Room' object contains the framework of the room, the door and the window. The walls are created by drawing solid cubes and scratching them to the appropriate sizes. Then texture images are added to the surface. For convenience, the door and the window are

created within the room class. They all have the animation for opening and closing. The door can be opened or closed by rotating the door. The window can be opened by transforming the glass. Also, the window is drawn in transparency. You can see the outside through glasses.

ii. **Door**. Door. The door object is actually defined and created inside the Room object, which means there is no .cpp or .h file for Door object. Door object has a frame which is created by calling 'glutSolidCube'. The door plank is created also in such way and it has a wooden texture to improve its realistics. Besides, it also has a door bar to open the door. When the door is opening, the bar will rotate

iii. **Window**. The window is drawn with transparent glasses. You can see the outside view through the window. I drew two solid cubes to represent glasses and colored them with half transparency.

iv. **Walls**. Walls are covered by wall papers.

v. **Bed**. 'Bed' object is constructed by creating two solid cubes and scratching them. It has two parts which are covered by texture images. There is an imported object, 'Pillow', which will be introduced in 'Imported Objects' session.

vi. **Closet**. 'Closet' object contains several boards. It has three wooden boards, two doors, a cloth hanger and interlayer. Similar to door object, the whole closet was built by scaling solid cubes instead of drawing quads, which makes it look like a real closet because closet's wooden boards have thickness rather than planes with 0 thickness.

vii. **Desktop.** 'Desktop' object is constructed by combining several boards and two drawers. All surfaces are covered with texture images. You can press '5'/6' to open/close the two drawers.

viii. **Clock**. On the wall, I created a clock which shows the current time. The clock consists of the dial plate, 3 pointers (second, minute and hour). The clock will update itself in every second. The time it gets is from the computer system rather than the variable 'deltaTime', so it is more accurate. Detailed implementation can be found in 'Clock.cpp'.

ix. **Dartboard**. Similar to the construction of the clock, dartboard is created by drawing several circles.

x. **Tellurion**. This object has a sphere and a base. The sphere is covered with an earth texture image. You can rotate the tellurion by pressing '7'. The animation I implemented is smooth. The tellurion will firstly speed up and then slow down. It has an acceleration speed.

xi. **Basketball**. In the project, 'Basketball' is implemented in 'BouncingBall.cpp' and 'BouncingBall.hpp'. The bouncing ball has an interaction and animation. Users can press 'E' to bounce the ball and the ball will bounce with the effect of gravity.

b. Imported objects

i. **Chair.**

ii. **PC**. The PC is covered with a texture image which represents the wallpaper.

iii. **Dart**. You can pick up the dart by pressing ‘E’ and throw it by pressing ‘R’. When you pick up the dart, it will move with your camera view. It is like you pick up an object in real life, which the dart will move around you.

iv. **Fan.** ‘Fan’ has an animation which is rotation. You can press ‘F’/’G’ to control the fan. The animation is similar to the tellurion. It is smooth.

v. **Bulb**. ‘Bulb’ is half transparent. It can be lightened and extinguished by pressing ‘,’/’.’.

vi. **Pillow**.

vii. **Hot Air Balloon**. This object has an animation which is flowing. You can take the hot air balloon by pressing ‘C’ and get out of it by pressing ‘V’. I implemented a distance check. You can only get on the hot air balloon if you are close enough to it.

viii. **Plane**. This object has two animations which are taking off and landing. The animations are smooth. For taking off, it will firstly speed up in slow pace, and then speed up very quickly until reach its maximum speed. Landing has similar smooth animation. You can press ‘Z’ to let the plane take off and press ‘X’ to let the plane land.

ix. **Boat**. This object has one animation which is flowing in the river.

x. **Sofa**.

xi. **Table**.

xii. **Bin**.

c. Scene objects

i. **Ground**. This object is a quad with very large area. It looks like infinite, but it has fixed size.

```
        glBegin(GL_QUADS);
            // draw the left board
            glNormal3f(0, 1, 0);
            // define texture coordinates for the 4 vertices
            glTexCoord2f(0.0f, 100.0f);
            glVertex3f(-10000, 0, 10000);
            glTexCoord2f(100.0f, 100.0f);
            glVertex3f(10000, 0, 10000);
            glTexCoord2f(100.0f, 0.0f);
            glVertex3f(10000, 0, -10000);
            glTexCoord2f(0.0f, 0.0f);
            glVertex3f(-10000, 0, -10000);

        glEnd();
```

Also, a grass texture image is binded to the quad to make it look more realistic.

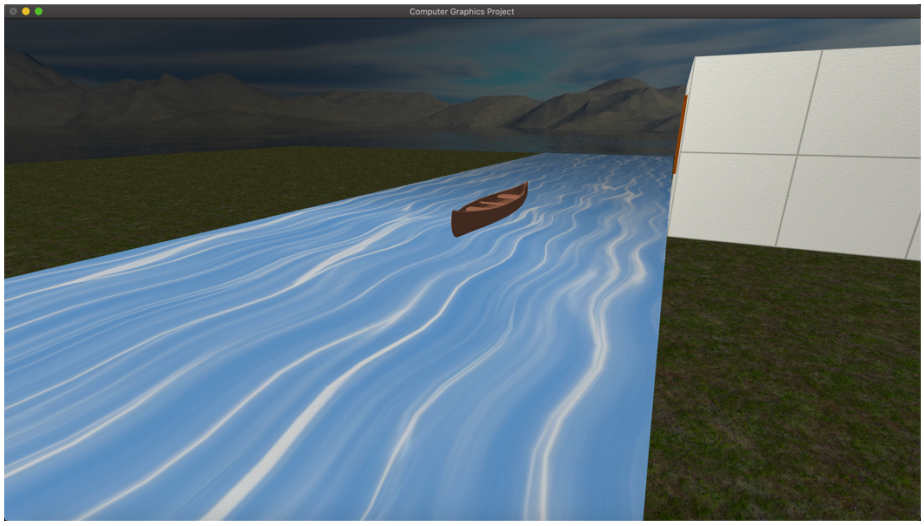
ii. **Skybox**. Skybox is implemented by drawing six quads and combine them to each other. Different texture images are covered to quads. Similar to the ground, the size of the skybox is very big so that you will not find the boundary of the scene. It makes the scene/world look infinite.

```
    glEnable(GL_COLOR_MATERIAL);
    glEnable(GL_TEXTURE_2D);

    //front
    glBindTexture(GL_TEXTURE_2D, textures[0]);
    glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0); glVertex3f(-TEX_SIZE, TEX_SIZE, -TEX_SIZE);
    glTexCoord2f(0.0, 1.0); glVertex3f(-TEX_SIZE, -TEX_SIZE, -TEX_SIZE);
    glTexCoord2f(1.0, 1.0); glVertex3f(TEX_SIZE, -TEX_SIZE, -TEX_SIZE);
    glTexCoord2f(1.0, 0.0); glVertex3f(TEX_SIZE, TEX_SIZE, -TEX_SIZE);
    glEnd();
```

iii. **Road**. This object is used for the plane to take off and land. It has no animation.

iv. **Water**. This object is constructed by repeating a water texture image. Also, it has an animation of flowing. In another word, it's not a still river. It will change itself according to the current time.

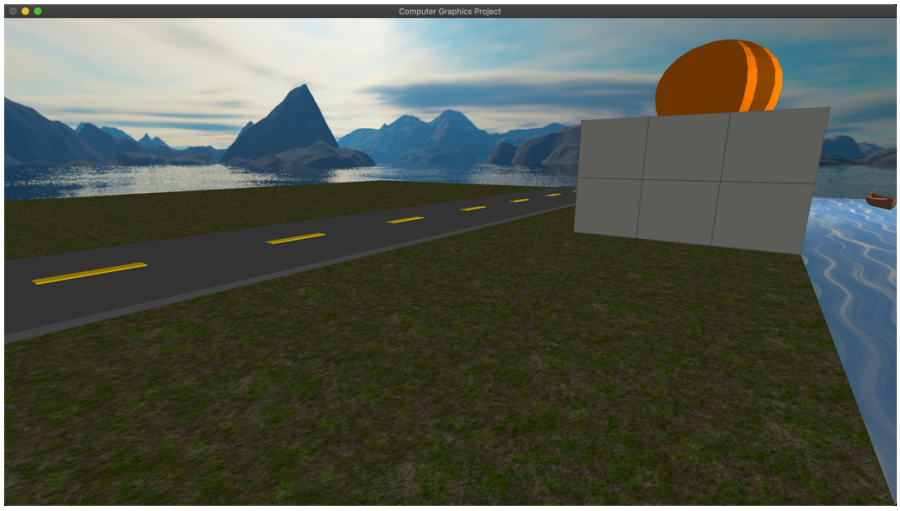


2. Transformations & Animations (Details will be shown in demo videos)

a. Door: Opening / closing
b. Closet: Opening / closing
c. Drawers: Opening / closing
d. Windows: Opening / closing
e. Clock: Ticktock
f. Dart: Pick up / Throwing / Moving with the camera view
g. Tellurion: Rotation
h. Basketball: Bouncing with the effect of gravity
i. Fan: Rotation
j. Boat: Flowing
k. Water: Flowing
l. How air balloon: Flying
m. Plane: Taking off / Landing

3. Textures:

a. **Implementation**: Texture images are read by 'Scene' using 'GetTexture' function. All texture images are kept in 'Textures' folder.
b. Many own-created objects are covered with texture images.
    i. **Ground**.



ii. **Water**.

iii. **Wall**.

iv. **Closet**.

v. **Tellurion**, **Desk, Drawers**.

vi. **Bed**.

vii. **Basketball**.

viii. **Floor**.

ix. **Door**.

x. **PC Desktop**.

4. Different Viewpoints

a. In this program, there is a camera object and you can change the camera view by pressing ‘W’, ‘S’, ‘A’ and ‘D’. Also, you can press ‘J’ and ‘K’ to move the camera upward and downward.

b. In addition to the camera, there is a hot air balloon. You take the balloon and it will take you to the air. Your camera view will change according to the height of the hot air balloon.

5. Lighting effects

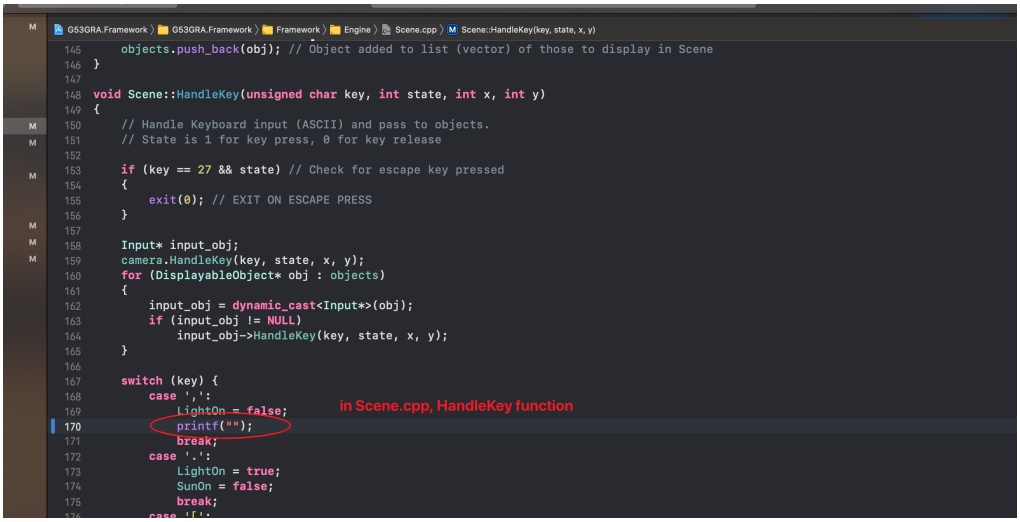
a. To create lighting effect in a fixed position, I change the way in which we normally draw the lighting. The program will firstly draw all objects to the scene, and then it will draw the lighting, otherwise, the position of the light will change according to the position of the camera.

```
48  void Scene::Draw()
49  {
50      // Clear colour and depth buffers
51      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
52      // Update based on window details (also sets initial projection properties)
53      Reshape(windowWidth, windowHeight);
54      // Reset MODELVIEW matrix
55      glMatrixMode(GL_MODELVIEW);
56      glLoadIdentity();
57      // Setup viewing properties
58      camera.SetupCamera();
59      // Display all objects in the Scene
60      for (DisplayableObject* obj : objects)
61          obj->Display();
62
63  //      light needs to be drawn after objects are drawn, so that it can be still at the same place
64      if (LightOn) {
65          glEnable(GL_LIGHTING);
66          glEnable(GL_LIGHT0);
67          GLfloat ambience[] = {0.2f, 0.2f, 0.2f, 1.0f};
68          GLfloat diffuse[] = {0.6f, 0.6f, 0.6f, 1.0f};
69          GLfloat specular[] = {1, 1, 1, 1.0f};
70          GLfloat position[] = {280, 200, 220, 1};
71          glLightfv(GL_LIGHT0, GL_AMBIENT, ambience);
72          glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
73          glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
74          glLightfv(GL_LIGHT0, GL_POSITION, position);
75      } else {
76          GLfloat ambience[] = {0.1f, 0.1f, 0.1f, 1.0f};
77          GLfloat diffuse[] = {0.15f, 0.15f, 0.15f, 1.0f};
78          GLfloat specular[] = {1.0f, 1.0f, 1.0f, 1.0f};
79          GLfloat position[] = {280, 200, 220, 1};
80          glLightfv(GL_LIGHT0, GL_AMBIENT, ambience);
81          glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
82          glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
83          glLightfv(GL_LIGHT0, GL_POSITION, position);
84      }
```

b. **Room light**. Room lighting is fixed within the room and it will not go out. But since the window is transparent, light will go out through the window. Shading is also implemented in the scene. All surfaces have normal vectors. You can turn on or off the room light by pressing ‘,’ or ‘.’.

c. **Sun light**. Similar to room light, sun light has a different position and it will create different shading effects. You can turn on or off the room light by pressing ‘[’ or ‘]’.

d. **Important notes**: After testing for many times, the commands for turning on or off the lighting will fail sometimes due to unknown reasons. I suggest you add a line of code to ‘Engine’ class. For example, add “printf(“”);” like following:




```
145     objects.push_back(obj); // Object added to list (vector) of those to display in Scene
146 }
147
148 void Scene::HandleKey(unsigned char key, int state, int x, int y)
149 {
150     // Handle Keyboard input (ASCII) and pass to objects.
151     // State is 1 for key press, 0 for key release
152
153     if (key == 27 && state) // Check for escape key pressed
154     {
155         exit(0); // EXIT ON ESCAPE PRESS
156     }
157
158     Input* input_obj;
159     camera.HandleKey(key, state, x, y);
160     for (DisplayableObject* obj : objects)
161     {
162         input_obj = dynamic_cast<Input*>(obj);
163         if (input_obj != NULL)
164             input_obj->HandleKey(key, state, x, y);
165     }
166
167     switch (key) {
168         case ',':
169             LightOn = false;
170             printf("");
171             break;
172         case '.':
173             LightOn = true;
174             SunOn = false;
175             break;
```

And then run the program. The lighting effect will work as expected.

6. **Creative ideas**.
    a. **Gravity**. In this program, gravity is implemented to make bouncing objects more real. Precisely, the basketball is the bouncing object in our program. When you press ‘T’ to play the basketball, it will firstly go up. Due to the effect of gravity, the speed will reduce to 0. When the speed is 0, it will then go down and speed up at the same time. When it touches the ground, it will reduce energy due to friction. Eventually, it will stop.
    b. **Dart movement**. When you pick up the dart, it will move around with the camera. It uses complex calculation to work out the right direction and position. Such animation can be used for other animation such as picking up a gun and shoot (FPS games).

```
16 void Dart::Display() {
17     glPushMatrix();
18         float X = 0, Y = 0, Z = 0, x = 0, y = 0, z = 0;
19
20         if (pickeup) {
21             Scene::GetCamera()->GetEyePosition(X, Y, Z);
22             Scene::GetCamera()->GetViewDirection(x, y, z);
23             position(X, Y, Z);
24         }
25
26         glTranslatef(pos[0], pos[1], pos[2]);
27
28         if (pickeup) {
29             float degree =(-x) / sqrt(x*x + z*z);
30             float theta = acos(degree) * 180;
31             theta /= 3;
32             if (z < 0) {
33                 glRotated(-theta, 0, 1, 0);
34             } else if (z > 0) {
35                 glRotated(theta, 0, 1, 0);
36             }
37             glRotated(90, 0, 0, 1);
38         } else {
39             glRotated(90, 1, 0, 0);
40         }
41
42         glTranslatef(2, 5, 0);
43         glScaled(50, 50, 50);
44         glColor3f(0.3, 0.7, 0.3);
45         DrawDart();
46         glColor3f(1, 1, 1);
47     glPopMatrix();
48 }
```

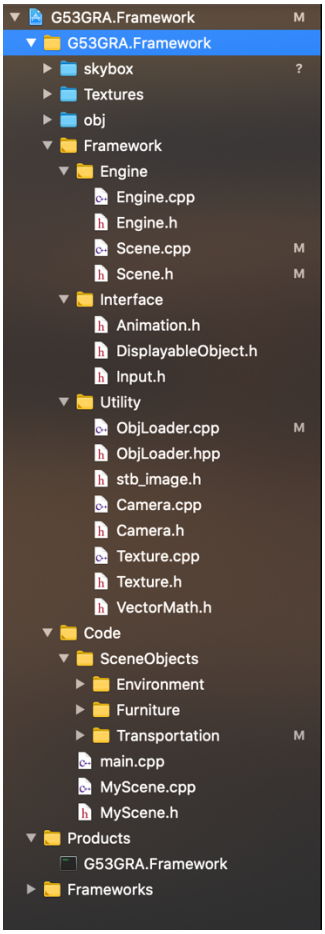
    c. **Smooth animation**. All animations in this program are smooth. For example, the basketball and fan have an acceleration speed. Plane even has an

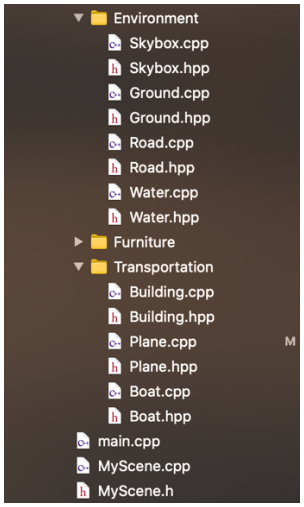
acceleration of acceleration speed for more smooth animation.

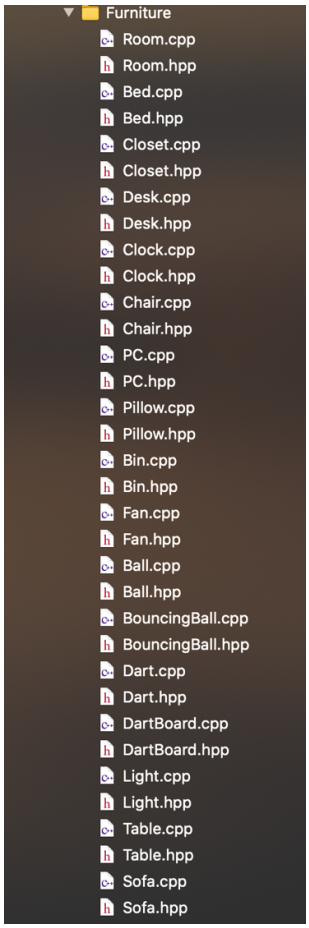
```
38  void Plane::Update(const double &deltaTime) {
39      // animation for the start step
40      if (start) {
41          pos[0] += speed;
42          if (a_speed < max_speed) {
43              speed += a_speed;       // acceleration speed
44              a_speed += a_a_speed;   // acceleration of acceleration speed
45          } else {
46              // reach the maximum speed, and is ready to rise
47              rising = true;
48          }
49
50          if (rising) {
51              // going up
52              pos[1] += y_speed;
53          }
54
55          if (rise_once && rising) {
56              // uplift the head of the plane and rise the plane body
57              rotation[2] += z_rotate;
58              if (rotation[2] >= 20) {
59                  // rotate to the maximum angle and stop to rotate
60                  rise_once = false;
61              }
62          }
63      }
```

```
65      // animation for the landing step
66      if (land) {
67          pos[0] += speed;            // has an initial speed
68          if (slow_down) {
69              speed -= a_speed;       // acceleration speed
70              a_speed += a_a_speed;   // acceleration of acceleration speed
71          }
72          if (pos[1] - land_y_speed >= 10) {
73              pos[1] -= land_y_speed;
74          } else {
75              // land on the ground
76              slow_down = true;
77          }
78          if (speed <= 0) {
79              // finish the whole process of the landing
80              land = false;
81          }
82      }
83  }
```

7. **Good code structure**
    a. Used object-oriented programming
    b. Used and extended the given framework (pure glut library)
    c. All objects are created in ‘MyScene’ class, in ‘Initialise’ function
    d. All objects inherit ‘Displayable’ class, some also inherit ‘Animation’ or ‘Input’
        i. ‘Displayable’: To be displayed in the scene, has a ‘Display’ function
        ii. ‘Animation’: To be updated in the scene, has a ‘Update’ function
        iii. ‘Input’: To handle mouse and keyboard input
    e. Project structure


G53GRA.Framework M
G53GRA.Framework
skybox ?
Textures
obj
Framework
Engine
Engine.cpp
Engine.h
Scene.cpp M
Scene.h M
Interface
Animation.h
DisplayableObject.h
Input.h
Utility
ObjLoader.cpp M
ObjLoader.hpp
stb_image.h
Camera.cpp
Camera.h
Texture.cpp
Texture.h
VectorMath.h
Code
SceneObjects
Environment
Furniture
Transportation M
main.cpp
MyScene.cpp
MyScene.h
Products
G53GRA.Framework
Frameworks


Environment
Skybox.cpp
Skybox.hpp
Ground.cpp
Ground.hpp
Road.cpp
Road.hpp
Water.cpp
Water.hpp
Furniture
Transportation
Building.cpp
Building.hpp
Plane.cpp M
Plane.hpp
Boat.cpp
Boat.hpp
main.cpp
MyScene.cpp
MyScene.h


Furniture
Room.cpp
Room.hpp
Bed.cpp
Bed.hpp
Closet.cpp
Closet.hpp
Desk.cpp
Desk.hpp
Clock.cpp
Clock.hpp
Chair.cpp
Chair.hpp
PC.cpp
PC.hpp
Pillow.cpp
Pillow.hpp
Bin.cpp
Bin.hpp
Fan.cpp
Fan.hpp
Ball.cpp
Ball.hpp
BouncingBall.cpp
BouncingBall.hpp
Dart.cpp
Dart.hpp
DartBoard.cpp
DartBoard.hpp
Light.cpp
Light.hpp
Table.cpp
Table.hpp
Sofa.cpp
Sofa.hpp