COMP 3069 - Computer Graphic

Course Work 2 Report

Li Kai WU 20027091

1. Introduction

(If want to run the code first, please scroll down to the end, there is important note!)

In my report, I have implemented a Livingroom scene. There are several static objects in this scene (see Figure 1), the walls are around this room, and when the user runs this project, it will start behind the door, which lets the user feel like going to someone's room. The closet and the table are fitting with my original room in my hometown. For example, the bookshelf, computer, and chair. Most static objects are bound with texture. Moreover, there are four dynamic objects in this scene, which are a balloon, Icosahedron, fan, and water. The light response is bound with every object, when using a keyboard or mouse to move the view of the camera, the light response will also change. The whole of objects is created by the OpenGL glut library.

Figure 1 The scene when running the code.

2. Framework of Code

This project, it divides into two parts. One is the code of the scene and each object; another is the texture of the object. It will be like Figure 2 Below.

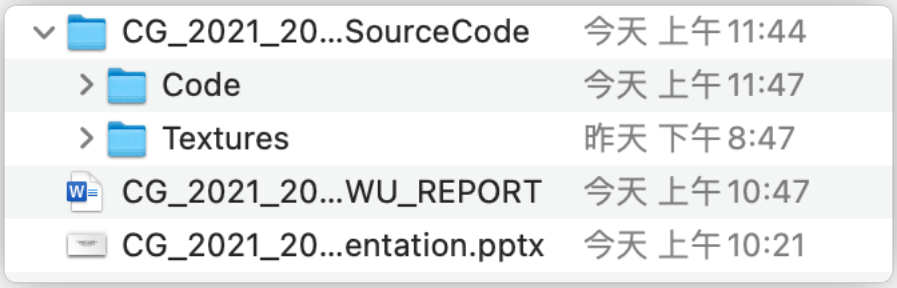


Figure 2 The code in the file

The source code has all definitions as below (see Figure 2.1) with a clear explanation.

```
#define GLUT_DISABLE_ATEXIT_HACK
#define GL_SILENCE_DEPRECATION
#pragma once

/**
* Class to be subclassed alongside DisplayableObject for all animated objects to be displayed in a Scene
* <p>
* Contains {@link #Update} method that must be overloaded. {@link #Update(float deltaTime)} is called from {@link Scene}.
*
* @author wil
*/
class Animation
{
public:
    Animation(void){}
    virtual ~Animation(void){}
    /**
    * Called each frame to update. Must be defined!
    * <p>
    * Use this to update animation sequence.
    * @param deltaTime change in time since previous call
    */
    virtual void Update(const double& deltaTime) = 0;
};

```

Figure 2.1 The Animation.h in the XCode

3. Scene

The scene must create first, then the camera will be easier to set. Therefore, the size of the window, the position of view, control setting, texture, etc. have been created first. Moreover, I create four walls around the room, and the floor and ceiling also been created in the scene (see Figure 3). The glEnable is used for each object, the glEnable function in OpenGL is used to enable various functions, which are determined by parameters. So, this function is very important in implementing various functions. Its function prototype is: void glEnable (GLenum cap); Cap is a parameter value, and each parameter value has a different function. The glEnable(GL_TEXTURE_20); is enable texture mapping.

```
// draw the left wall
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texID);
glBegin(GL_QUADS);
glMaterialfv(GL_FRONT, GL_SPECULAR, static_cast<GLfloat*>(specular));
glMaterialf(GL_FRONT, GL_SHININESS, static_cast<GLfloat>(shininess));
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, static_cast<GLfloat*>(lDiffuse));
glNormal3f(1.f, 0.f, 0.f);
glTexCoord2f(0.0f, 0.0f);
glVertex3f(scale[0] * (2.f), scale[0] * (-1.f), scale[0] * (-2.f));
glTexCoord2f(1.0f, 0.0f);
glVertex3f(scale[0] * (2.f), scale[0] * (-1.f), scale[0] * (2.f));
glTexCoord2f(1.0f, 1.0f);
glVertex3f(scale[0] * (2.f), scale[0] * (3.f), scale[0] * (2.f));
glTexCoord2f(0.0f, 1.0f);
glVertex3f(scale[0] * (2.f), scale[0] * (3.f), scale[0] * (-2.f));
glEnd();
//clear the texture attribute
glBindTexture(GL_TEXTURE_2D, NULL);
```

Figure 3: The example of the Wall

4. Camera

The Camera view is set to be like the figure below (see Figure 4). Therefore, the user can use the keyboard and mouse to control the camera to view. Users can use “W”, “A”, “S”, “D”, “E”, “Q” to move the camera by going forward, left, right, back, down, and up (see Figure 4.1). Users can also press “space” to reset the position of the camera. And use the mouse can change the angle of the camera (see Figure 4.2).

```
Camera::Camera() : wKey(0), sKey(0), aKey(0), dKey(0), uKey(0), nKey(0), currentButton(0), mouseX(0), mouseY(0)
{
    Reset();
}

void Camera::Reset(){
    // set the camera position to start at (0,0,0)
    eyePosition[0] = 0.0f;
    eyePosition[1] = 0.0f;
    eyePosition[2] = 0.4f * static_cast<float>(Scene::GetWindowHeight()) / static_cast<float>(tan(M_PI / 6.0));

    // set the view direction vector of the camera to be (0,0,-1)
    vd[0] = 0.0f;
    vd[1] = 0.0f;
    vd[2] = -1.0f;

    // set the planar forward direction vector of the camera to be (0,0,-1)
    forward[0] = 0.0f;
    forward[1] = 0.0f;
    forward[2] = -1.0f;

    // set the right vector to point along the x axis
    right[0] = 1.0f;
    right[1] = 0.0f;
    right[2] = 0.0f;

    // set the up vector of the camera to be up the y axis
    up[0] = 0.0f;
    up[1] = 1.0f;
    up[2] = 0.0f;
}
```

Figure 4: The original position of camera view

```
void Camera::HandleKey(unsigned char key, int state, int x, int y)
{

    switch (key)
    {
        case ' ':
            Reset();
            break;
        case 'A':
        case 'a':
            aKey = state;
            break;
        case 'D':
        case 'd':
            dKey = state;
            break;
        case 'W':
        case 'w':
            wKey = state;
            break;
        case 'S':
        case 's':
            sKey = state;
            break;
        case 'Q':
        case 'q':
            uKey = state;
            break;
        case 'E':
        case 'e':
            nKey = state;
            break;
        default:
            break;
    }

}
```

Figure 4.1: The function used for control camera position

```
void Camera::HandleMouseDrag(int x, int y)
{
    float rx, ry;
    float sensitivity = 0.01f; // speed of the camera moving

    // work out the difference between where the mouse was last used (mouseX, mouseY) to
    // position the view direction and the current position (x, y) the mouse is in
    rx = static_cast<float>(x - mouseX);
    ry = static_cast<float>(y - mouseY);

    // switch on which button was pressed and only do the update if the left button was pressed
    switch (currentButton)
    {
        case GLUT_LEFT_BUTTON:

            // add on the amount of change in to the left and right view direction of the camera
            if (rx > 0)
                add(vd, right, rx*sensitivity);
            else
                sub(vd, right, rx*-sensitivity);
            // add on the amount of change in to the up and down view direction of the camera
            if (ry > 0)
                sub(vd, up, ry*sensitivity);
            else
                add(vd, up, ry*-sensitivity);

            // normalise the view direction so it is length 1
            norm(vd);

            // use the view direction crossed with the up vector to obtain the corrected right vector
            cross(vd, up, right);

            // normalise the right vector
            norm(right);

            forward[0] = vd[0];
            forward[2] = vd[2];
            norm(forward);
            break;
        case GLUT_RIGHT_BUTTON:
            break;
        case GLUT_MIDDLE_BUTTON:
            break;
        default:
            break;
    }

    mouseX = x;
    mouseY = y;
}
```

Figure 4.2: The setting of the mouse control

5. Animation

In this part, I also use the animation that is from course work 1, it is the same logic to move the object as snowman and carousel. The interface class has been created in Animation.h. All of the dynamic object contain the Animation.h, through the virtual void Update(const double& deltaTime) = 0; and update the running time by change the value in double diff = static_cast<double>(t - time) / 1500.0; in the Engine.cpp.

Moreover, in this scene, there are four dynamic objects, which are Icosahedron, balloon, fan, and water. When the scene is created, four objects will start moving. The water object will be discussed in the texture part.

When the scene is created, the balloon will start moving from the original position to far away and go right and up. The balloon will incline to move in the scene. Therefore, the balloon is also bound with the cut panda texture. The object moving and code will show below:

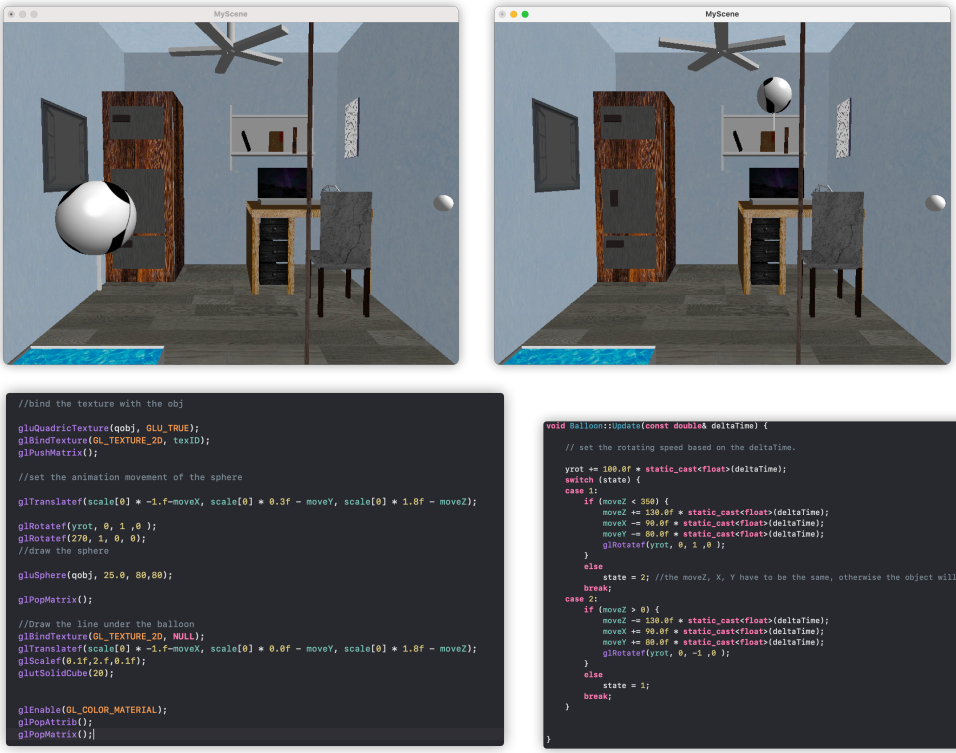


Figure 5: The Balloon object and the animation of balloon

When the scene created, the fan will start rolling, the speed will like below:

```
void Fan::Update(const double& deltaTime)
{
    count += 100.0f * static_cast<float>(deltaTime);

    yrot += 200.0f * static_cast<float>(deltaTime);

}
```

Figure 5.1: The animation of Fan

```
glPushMatrix();
// draw a object besides the computer
glTranslatef(scale[0] * (1.75f), scale[0] * (0.37f), scale[0] * (-1.5f));
glRotatef(yrot, 0, 1 ,0 );
glScalef(20, 20, 20);
glutWireIcosahedron();
glPopMatrix();
```

```
void Computer::Update(const double& deltaTime) {

    // set the rotating speed based on the deltaTime.

    yrot += 100.0f * static_cast<float>(deltaTime);

}
```

Figure 5.2: The object of Icosahedron and the animation of icosahedron

6. Object

For making the 3D object in the scene, most of them are using glVertex to create. When the place and position need to change, then the translate function will be used. When the angle of the object needs to rotate, then use the glRotate to rotate the object. When changing the size of the object, then the scaling model will be used for it.

```
glPushMatrix();
glTranslatef((scale[0]*(1.5f)), (scale[0]*1.4f), (scale[0]*(-1.9f)));
glScalef(0.05f,1.2f,0.1f);
glutSolidCube(scale[0]*1.0);
glPopMatrix();

//right
glPushMatrix();
glTranslatef((scale[0] * (0.0f)), (scale[0] * 1.4f), (scale[0] * (-1.9f)));
glScalef(0.05f, 1.2f, 0.1f);
glutSolidCube(scale[0] * 1.0);
glPopMatrix();


//mid up
glPushMatrix();
glTranslatef((scale[0] * (0.75f)), (scale[0] * 1.8f), (scale[0] * (-1.9f)));
glScalef(1.5f, 0.05f, 0.1f);
glutSolidCube(scale[0] * 1.0);
glPopMatrix();
```

Figure 6: The code in the addBook file

7. Lighting

The original light response is set in the Engine file, there are three lights are set in here. Ambient light: light emitted by a source that has been repeatedly shattered by the environment so that the direction of the incident cannot be determined, that is, it appears to come from all directions. Its characteristic is that the direction of incidence and the direction of exit are arbitrary. Diffuse light: Coming from a particular direction, it is brighter when perpendicular to an object than when tilted. Once it hits an object, it radiates evenly in all directions, with the effect that it is equally bright no matter where the viewpoint is. It is characterized by a unique direction of incidence and an arbitrary direction of the exit. Specular light: from a specific direction and reflected in another direction, a parallel laser beam on a high-quality mirror produces 100% specular reflection, which is characterized by the unique direction of incident and exit. The lighting is set as light 0 in the Engine, which light will change when the user moves the camera.

```
// Turn off 2 sided lighting
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_FALSE);

// set the ambient light model
GLfloat global_ambient[] = {0.2f, 0.2f, 0.2f, 1.0f};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, global_ambient);

glEnable(GL_LIGHTING);
GLfloat ambience[] = {0.2f, 0.2f, 0.2f, 1.0f};
GLfloat diffuse[] = {0.8f, 0.8f, 0.8f, 1.0f};
GLfloat specular[] = {1.0f, 1.0f, 1.0f, 1.0f};
GLfloat position[] = {1.0f, 1.0f, 1.0f, 0.0f};
glLightfv(GL_LIGHT0, GL_AMBIENT, ambience);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
glLightfv(GL_LIGHT0, GL_POSITION, position);
glEnable(GL_LIGHT0);

// Enable smooth shading from lighting
glShadeModel(GL_SMOOTH);

// Enable automatic normalisation of normal vectors
glEnable(GL_NORMALIZE);

current->Initialise();
time = 0;
```

Figure 7: The light setting in the scene

```
float lighting1[] = { 0.55f,0.55f,0.55f,1.f };
float specular[] = { 1.0f,1.0f,1.0f,1.0f };
float shininess = 100.0f;
//set the lighting reflection
glMaterialfv(GL_FRONT, GL_SPECULAR, static_cast<GLfloat*>(specular));
glMaterialf(GL_FRONT, GL_SHININESS, static_cast<GLfloat>(shininess));
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, static_cast<GLfloat*>(lighting1));
```

Figure 7.1: The example about how to set the light

8. Texture

The use of texture makes the scene more realistic. The objects that need to use texture will inherit the texture class. The texture must be a picture and it must be a .bmp file, and the size and angle of the texture are dependent on the object, for example, if the texture is on the wall, then the size of texture will be the same as the wall, if the texture is on the computer, then the size will be much smaller than wall's texture. When the picture is fitted with the setting, it can be used in the scene. After creating an object, need to add the path of the image in the MyScene file, and it returns the ID of the texture, that could bind with the object.

```
Wall::Wall(const std::string& filename1, const std::string& filename2) :Wall() {
    texID = Scene::GetTexture(filename1);
    texID2 = Scene::GetTexture(filename2);

}
```

Figure 8: The example of using the texture ID in the wall file

In this project, there is a dynamic texture that uses in the water object. Firstly, I decrease the value of alpha to 0.5, which sets the transparency of the texture of water, it makes more realistic. The water texture is bound to a square that does not move the position in the scene, but the position of its binding texture moves over time to simulate the animation of the water when the scene refreshes.

```
texCoords[(i + (xGridDims + 1) * j) * 2 + 0] = (float)i / (float)xGridDims;
texCoords[(i + (xGridDims + 1) * j) * 2 + 1] = (float)j / (float)zGridDims;
```

Figure 8.1: When water texture stays static

```
texCoords[(i + (xGridDims + 1) * j) * 2 + 0] = (float)sin(time + (double)j) * radius + (float)i / (float)xGridDims;
texCoords[(i + (xGridDims + 1) * j) * 2 + 1] = (float)cos(time + (double)i) * radius + (float)j / (float)zGridDims;
```

Figure 8.2: When water texture stays dynamic

According to water texture, I also decreased the value of alpha to 0.48 in window texture, which sets the transparency of the texture of the window. Therefore, the user can see the view of the board outside of the scene, it makes it more like the rear window. Moreover, when the user press "f", the dynamic texture of water will become static texture, which means the "freeze" to make water stop moving.

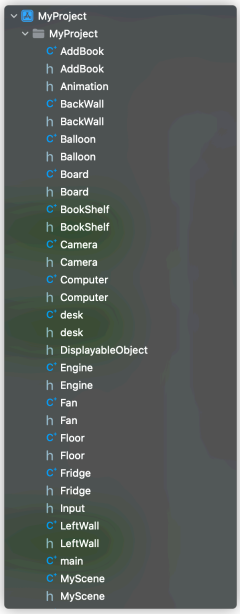
9. Conclusion

In this report, all requirements are met, object, scene, light response, animation, and some individual features in this work, I spend about three weeks finishing this project. Even though some of my creativity is not pretty good, but I think I still make some good structure of the code. I learned a lot of knowledge in this project and gained a deeper understanding of Computer Graphics.

10.Note

If you cannot run the code with texture, please change the file name into local file name, because my device for making this project is MAC, so in my texture lead part, I can only use the local path to show the texture, I would put like "./Texture/object.bmp". To make sure the user can show the texture correctly, if cannot show the texture correctly, then you can change from the file path name to local file name /Users/kelvin/ComputerGraphics/CG_2021_20027091_PROJECT/CG_2021_20027091_SourceCode (This is my local texture file)

Therefore, in my code, I separate each code into XCode, like below:



When you try to run the code, please don't forgot to put these files into the platform, like XCode, VScode. And change the path of the texture (in MyScene file). Thank you! And sorry for inconvenient!