Computer Graphics | Group 9 | Assignment 2

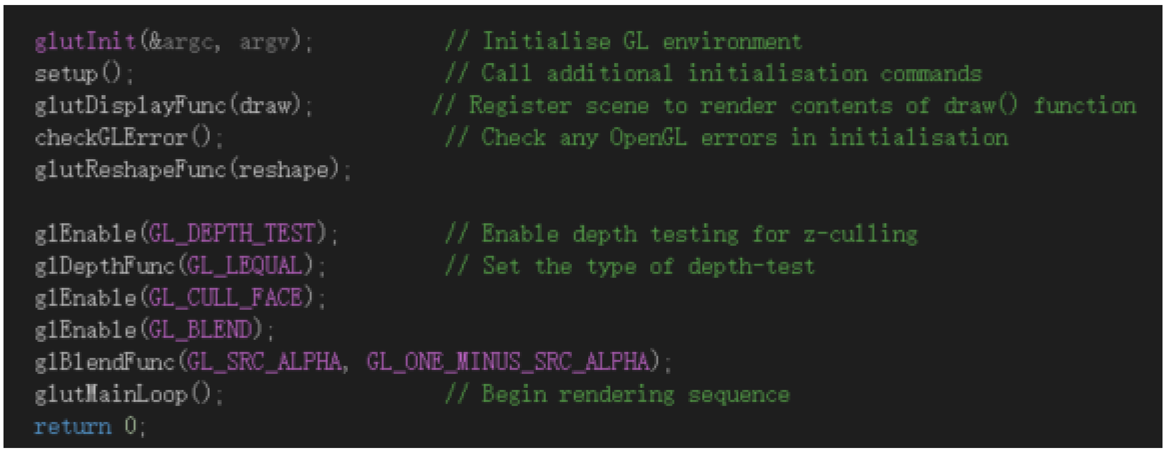
Jialu Gan, Ethan Aron Phipps, Jinhao Wang

1 – Hierarchical Modelling

Currently we did 2 different ways to accomplish task 1.

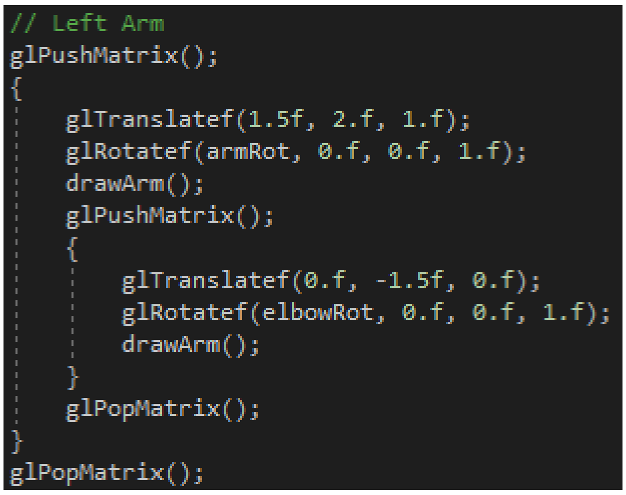
The first way does not use framework. Define different sub-methods to draw different parts of the robot, then call them directly in the drawing method (We defined it as drawRobot(), then call drawRobot() in draw() method.).

In main.cpp, keep this part uncommented in main() method to view.

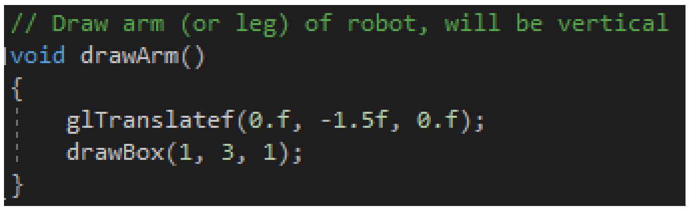


```
glutInit(&argc, argv);          // Initialise GL environment
setup();                        // Call additional initialisation commands
glutDisplayFunc(draw);         // Register scene to render contents of draw() function
checkGLError();                 // Check any OpenGL errors in initialisation
glutReshapeFunc(reshape);

glEnable(GL_DEPTH_TEST);        // Enable depth testing for z-culling
glDepthFunc(GL_LEQUAL);         // Set the type of depth-test
glEnable(GL_CULL_FACE);
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glutMainLoop();                 // Begin rendering sequence
return 0;
```

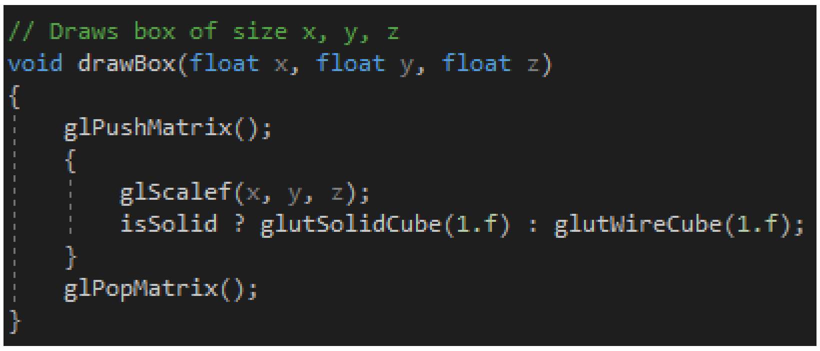
The robot is drawn from a single function, this starts by pushing onto the matrix stack, then draws the robot’s body. From the body the arms and legs are drawn individually, but by using the matrix stack they are all relative to the robot, not the previous limb. Each limb is made from their upper and lower parts, where the upper is drawn first, next to the body, and the lower is drawn after, relative to the upper limb.



```
// Left Arm
glPushMatrix();
{
    glTranslatef(1.5f, 2.f, 1.f);
    glRotatef(armRot, 0.f, 0.f, 1.f);
    drawArm();
    glPushMatrix();
    {
        glTranslatef(0.f, -1.5f, 0.f);
        glRotatef(elbowRot, 0.f, 0.f, 1.f);
        drawArm();
    }
    glPopMatrix();
}
glPopMatrix();
```

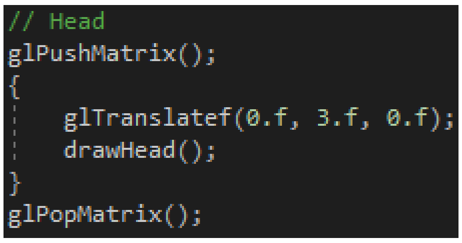


```
// Draw arm (or leg) of robot, will be vertical
void drawArm()
{
    glTranslatef(0.f, -1.5f, 0.f);
    drawBox(1, 3, 1);
}
```

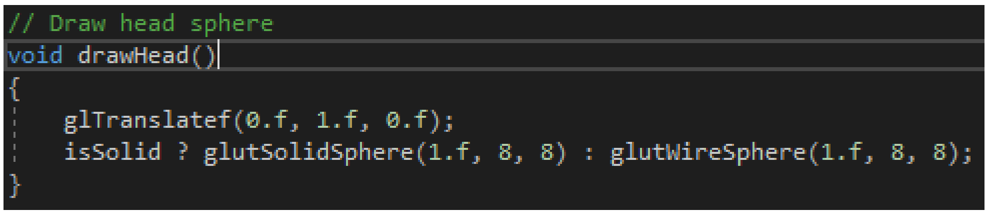


```
// Draws box of size x, y, z
void drawBox(float x, float y, float z)
{
    glPushMatrix();
    {
        glScalef(x, y, z);
        isSolid ? glutSolidCube(1.f) : glutWireCube(1.f);
    }
    glPopMatrix();
}
```

The above code shows all the code involved in the drawing of the left arm, initially the matrix is pushed as to be able to pop it later, then from the centre of the body it is translated to the shoulder and rotated, then the upper arm is drawn. It then pushes the matrix again to start drawing the lower arm, which is drawn at the end of the upper arm with a rotation. This is repeated with different rotations and translations from the centre of the body for the right arm and legs.



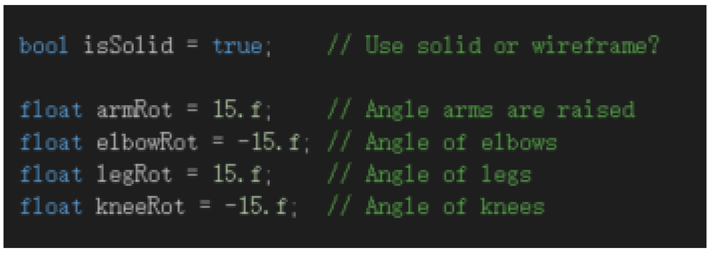
```
// Head
glPushMatrix();
{
    glTranslatef(0.f, 3.f, 0.f);
    drawHead();
}
glPopMatrix();
```



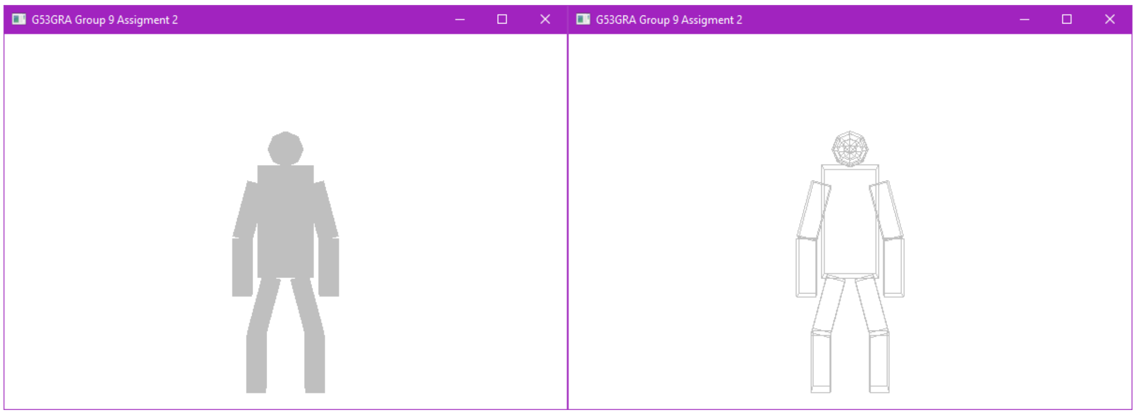
```
// Draw head sphere
void drawHead()
{
    glTranslatef(0.f, 1.f, 0.f);
    isSolid ? glutSolidSphere(1.f, 8, 8) : glutWireSphere(1.f, 8, 8);
}
```

The head is drawn with a sphere, and like the limbs it uses the matrix stack to easily get to the top of the body.

All rotations (arm, elbow, leg, knee) are modifiable in the main.cpp, alongside a Boolean that decides if the robot is drawn solid or wireframe.



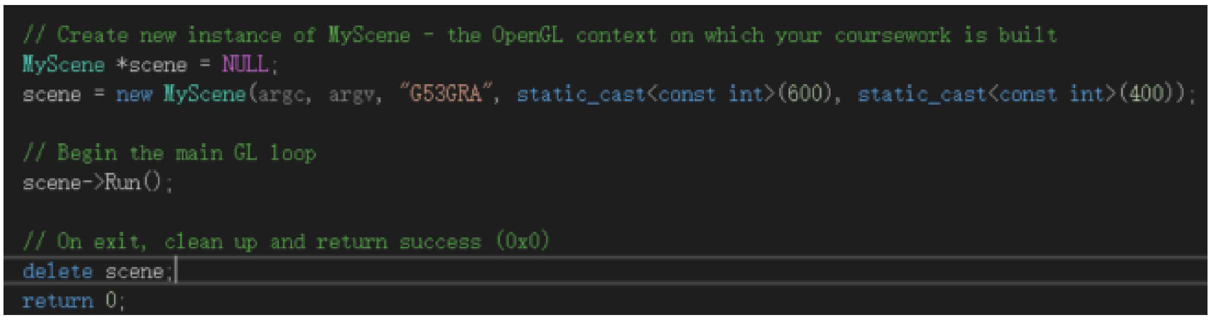
```
bool isSolid = true;    // Use solid or wireframe?

float armRot = 15.f;    // Angle arms are raised
float elbowRot = -15.f; // Angle of elbows
float legRot = 15.f;    // Angle of legs
float kneeRot = -15.f;  // Angle of knees
```



The second way uses the framework, derived from the lab Tree example, using displayable object as the base class.

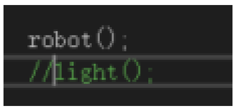
To view this, keep the following two parts uncommented in main.cpp and MyScene.cpp.

Main.cpp -- main() method:



```
// Create new instance of MyScene - the OpenGL context on which your coursework is built
MyScene *scene = NULL;
scene = new MyScene(argc, argv, "G53GRA", static_cast<const int>(600), static_cast<const int>(400));

// Begin the main GL loop
scene->Run();

// On exit, clean up and return success (0x0)
delete scene;
return 0;
```

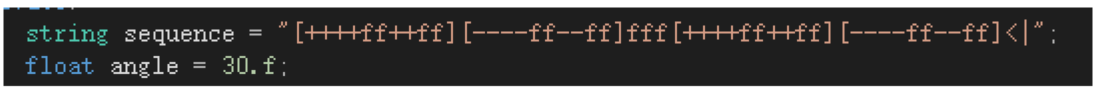
MyScene.cpp – Initialize() method:



```
robot();
//light();
```

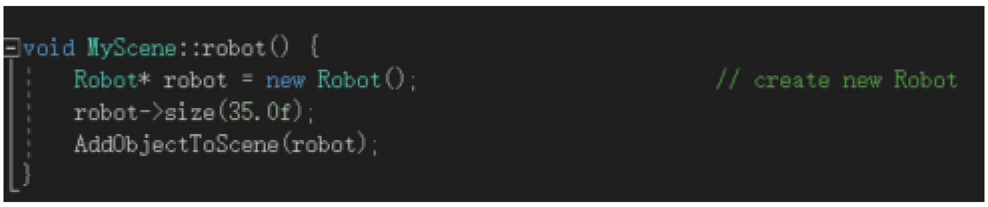
We modify some operations represented by certain characters to fit for this task.

Then we draw in order of right leg, left leg, body, right arm, left arm and head.

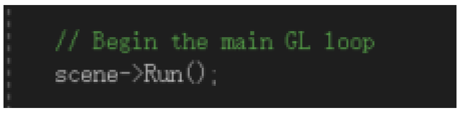


```
string sequence = "[++++ff++ff][----ff--ff]fff[++++ff++ff][----ff--ff]<|";
float angle = 30.f;
```

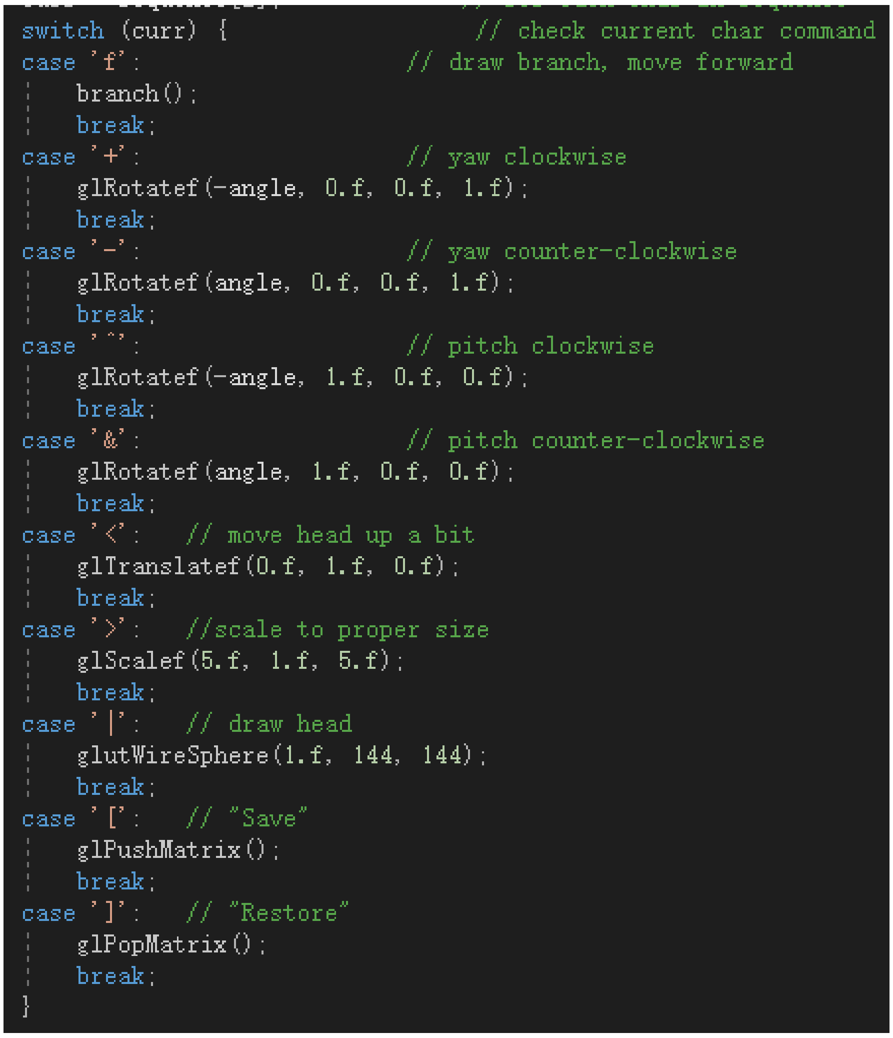
Finally, we new a Robot object in MyScene, add it to the scene and run Display() method.



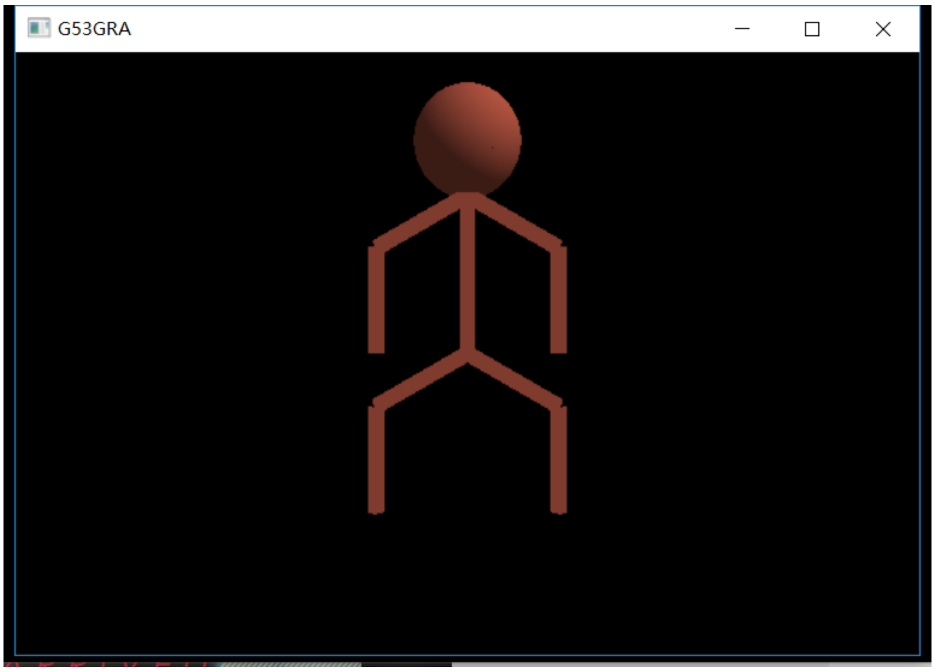
```
void MyScene::robot() {
    Robot* robot = new Robot();                    // create new Robot
    robot->size(35.0f);
    AddObjectToScene(robot);
}
```



```
// Begin the main GL loop
scene->Run();
```



```
switch (curr) {                     // check current char command
case 'f':                   // draw branch, move forward
    branch();
    break;
case '+':                   // yaw clockwise
    glRotatef(-angle, 0.f, 0.f, 1.f);
    break;
case '-':                   // yaw counter-clockwise
    glRotatef(angle, 0.f, 0.f, 1.f);
    break;
case '^':                   // pitch clockwise
    glRotatef(-angle, 1.f, 0.f, 0.f);
    break;
case '&':                   // pitch counter-clockwise
    glRotatef(angle, 1.f, 0.f, 0.f);
    break;
case '<':    // move head up a bit
    glTranslatef(0.f, 1.f, 0.f);
    break;
case '>':    //scale to proper size
    glScalef(5.f, 1.f, 5.f);
    break;
case '|':    // draw head
    glutWireSphere(1.f, 144, 144);
    break;
case '[':    // "Save"
    glPushMatrix();
    break;
case ']':    // "Restore"
    glPopMatrix();
    break;
}
```



2 – Viewing Transformation

$$\boldsymbol{u} = (0,1,0)$$
$$\boldsymbol{v} = (1,0,0)$$
$$\boldsymbol{n} = (0,0,-1)$$
$$\boldsymbol{eye} = (10,10,10)$$
$$\boldsymbol{P} = (20,20,10)$$

$$\boldsymbol{R} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{T} = \begin{bmatrix} 1 & 0 & 0 & -10 \\ 0 & 1 & 0 & -10 \\ 0 & 0 & 1 & -10 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{P'} = \boldsymbol{RTP}$$

$$\boldsymbol{P'} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -10 \\ 0 & 1 & 0 & -10 \\ 0 & 0 & 1 & -10 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 20 \\ 20 \\ 10 \\ 1 \end{bmatrix}$$

$$\boldsymbol{P'} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 10 \\ 10 \\ 0 \\ 1 \end{bmatrix}$$

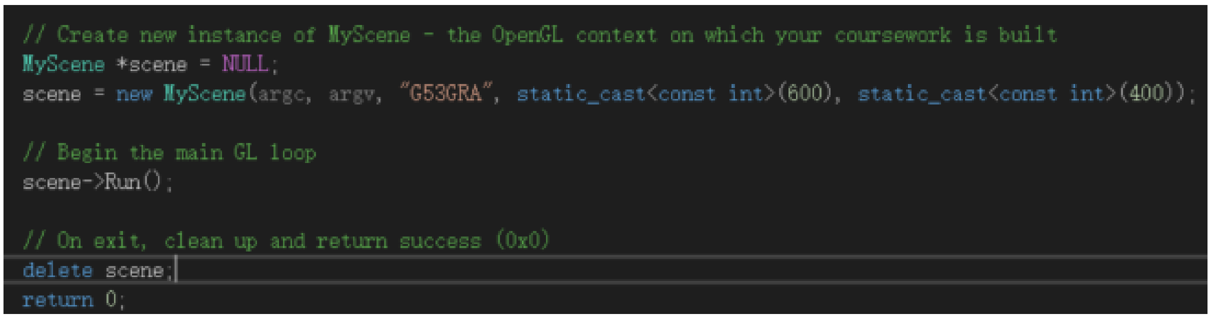
$$\boldsymbol{P'} = \begin{bmatrix} 10 \\ 10 \\ 0 \\ 1 \end{bmatrix}$$

P’s coordinates in view space are $(10, 10, 0)$.

3 – Light and Texture: Wall Light

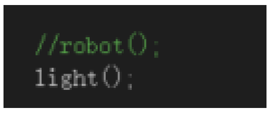
To view this task, keep the following two parts uncommented in main.cpp and MyScene.cpp.

Main.cpp -- main() method:



```
// Create new instance of MyScene - the OpenGL context on which your coursework is built
MyScene *scene = NULL;
scene = new MyScene(argc, argv, "G53GRA", static_cast<const int>(600), static_cast<const int>(400));

// Begin the main GL loop
scene->Run();

// On exit, clean up and return success (0x0)
delete scene;
return 0;
```

MyScene.cpp – Initialize() method:



```
//robot();
light();
```

This project is implemented by Phong reflection model. It consists of ambient, diffuse and specular reflection. Ambient is the basic and uniform colour of an object. Diffuse reflect the light from a surface and scatter them in many different angles. Specular reflection, known as regular reflection, reflect the incident lighting to another side of the norm with the same angle form by incident light and the surface normal.

Wall

```
//reflect all light, using white diffuse
float wDiffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };
float specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
float shininess = 128.0f;

glColor4f(0.5f, 0.5f, 0.5f, 1.0f);
glBegin(GL_QUADS);
glMaterialfv(GL_FRONT, GL_SPECULAR, static_cast<GLfloat*>(specular));
glMaterialf(GL_FRONT, GL_SHININESS, static_cast<GLfloat>(shininess));

for (int i = -10; i < 10; i++)
{
    for (int j = -10; j < 10; j++)
    {

        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, static_cast<GLfloat*>(wDiffuse));
        glNormal3f(0.0f, 0.0f, 1.0f);
        glVertex3f(scale[2] * static_cast<float>(j)+scale[2], scale[0] * static_cast<float>(i)+scale[0],
            -100.0f);
        glVertex3f( scale[2] * static_cast<float>(j), scale[0] * static_cast<float>(i)+scale[0], -100.0f);
        glVertex3f( scale[2] * static_cast<float>(j), scale[0] * static_cast<float>(i), -100.0f);
        glVertex3f( scale[2] * static_cast<float>(j)+scale[2],scale[0] * static_cast<float>(i), -100.0f);
    }
}
glEnd();
```

The for loop draws 20 * 20 small square to form a wall. Each square is grey and its material is able to reflect all light. Therefore, it is tend to dark without the lights.

Light

Light/Property	Ambient	Diffuse	Specular	Attenuation
Red	(0.f, 0.f, 0.f, 1.f)	(1.f, 0.f, 0.f, 1.f)	(0.f, 0.f, 0.f, 1.f)	0.0025f
Blue	(0.f, 0.f, 0.f, 1.f)	(0.f, 0.f, 1.f, 1.f)	(0.f, 0.f, 0.f, 1.f)	0.0025f

The red and blue positional lights stand in front of the wall and project their light to the wall.

In this project, we only add the diffuse so that the projection on the wall will be a circle. The attenuation of two lights is set as 0.0025f in case of the whole wall is cover with bright lighting. Besides, it could be observed that the two lights influence each other. Therefore, the left side is trend to pink instead of pure red, the middle area is purple and the right side is trend to dark violet. The remain edge of the screen is grey (light can be on and off by pressing number 0 ,1, 2, 3, 4).