Class work:

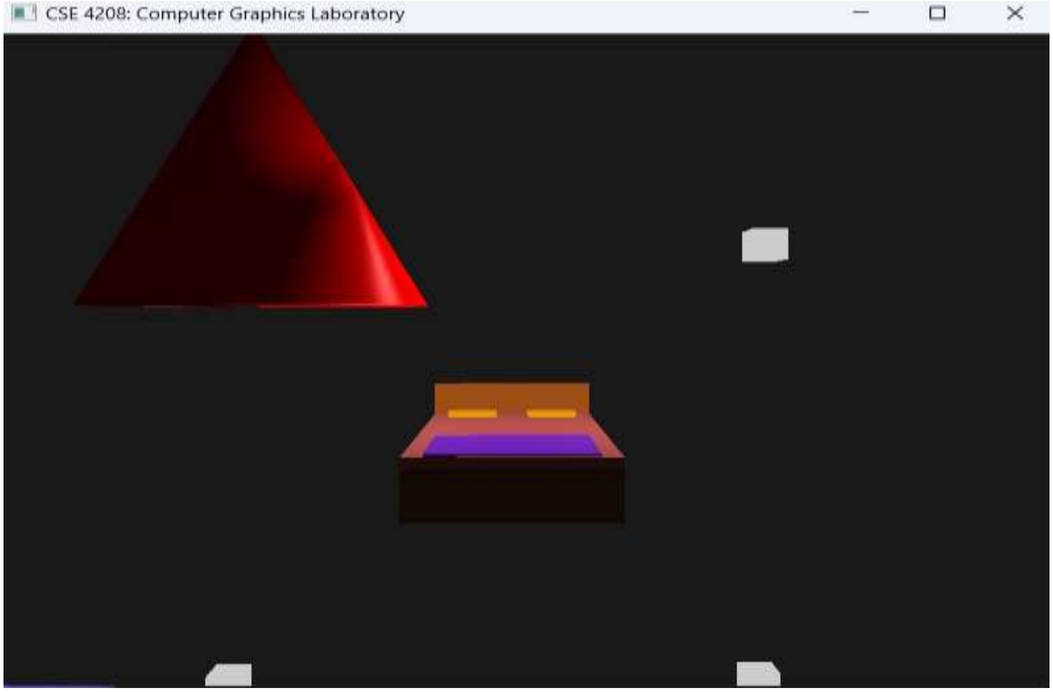

Fig 3.1: Lighting in Cone

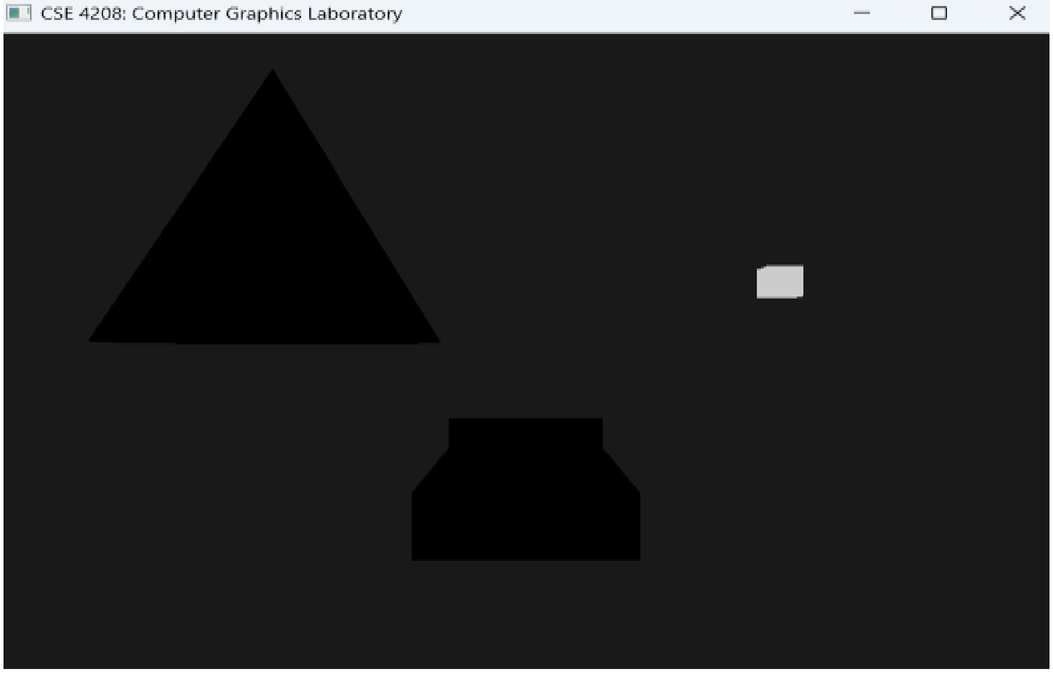

Fig 3.2 : Point light off

Assignment-3:

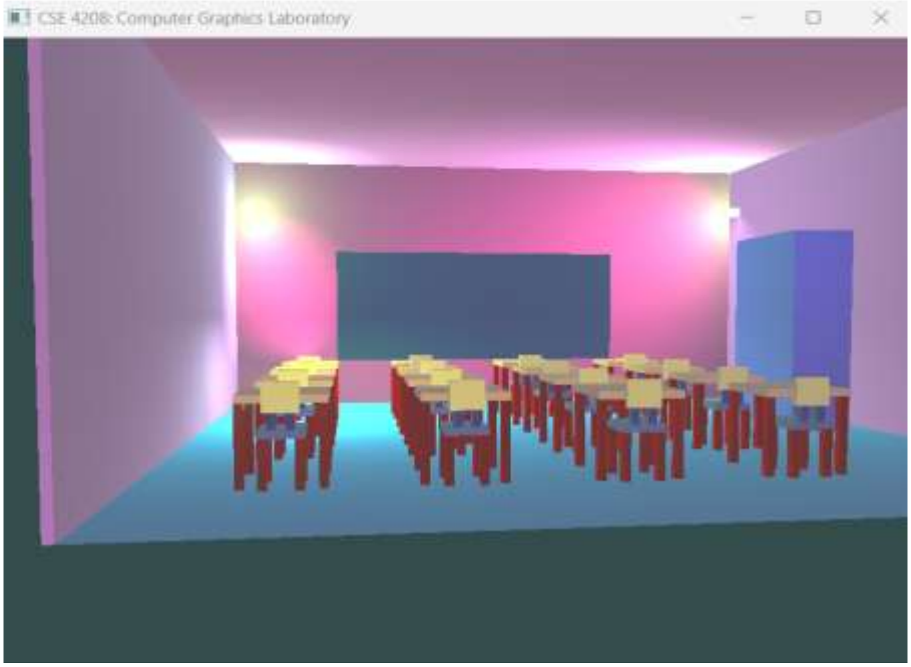

Fig 3.3 : Classroom with lighting

Directional Light:

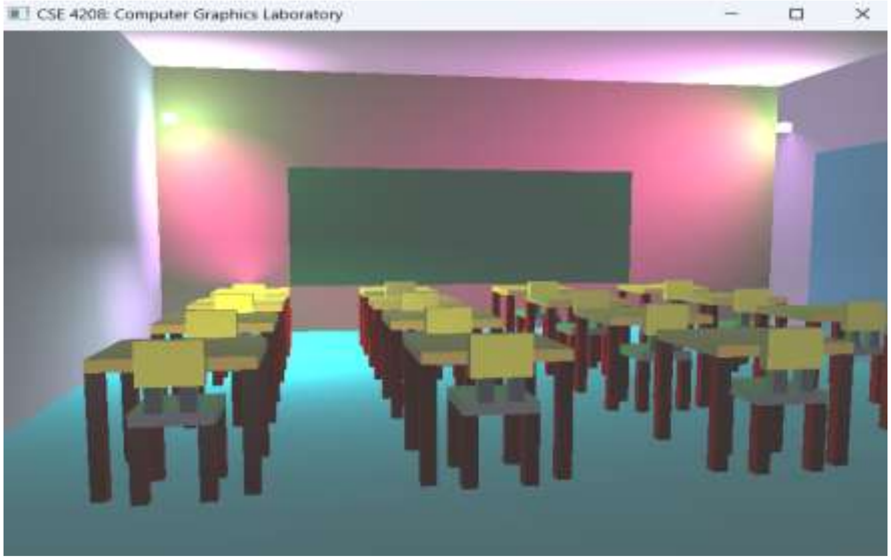

Fig 3.4 : Directional light off

Point light:

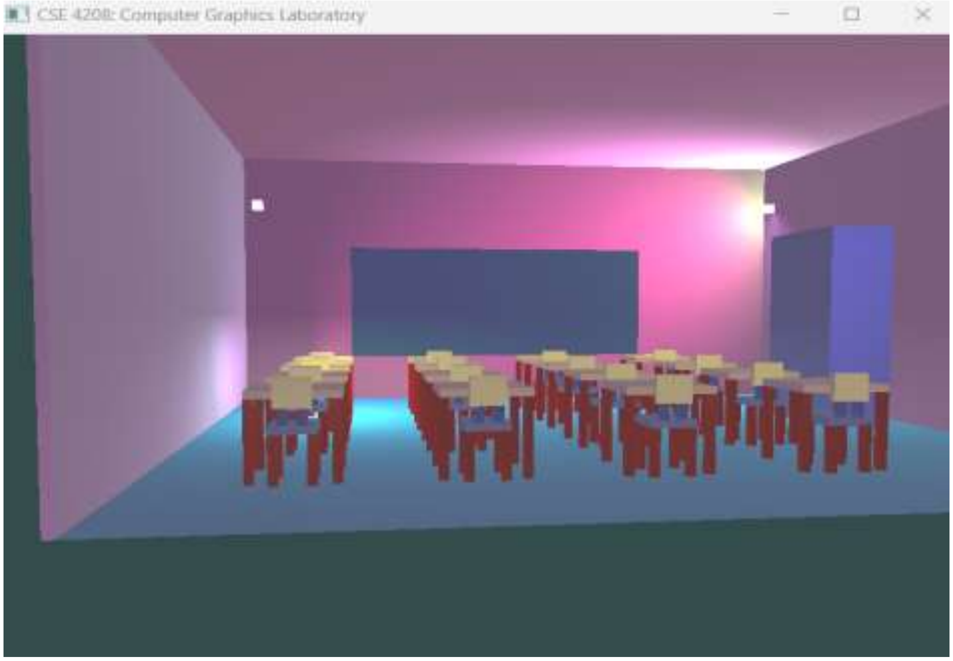

Fig 3.5 : Point light1 off

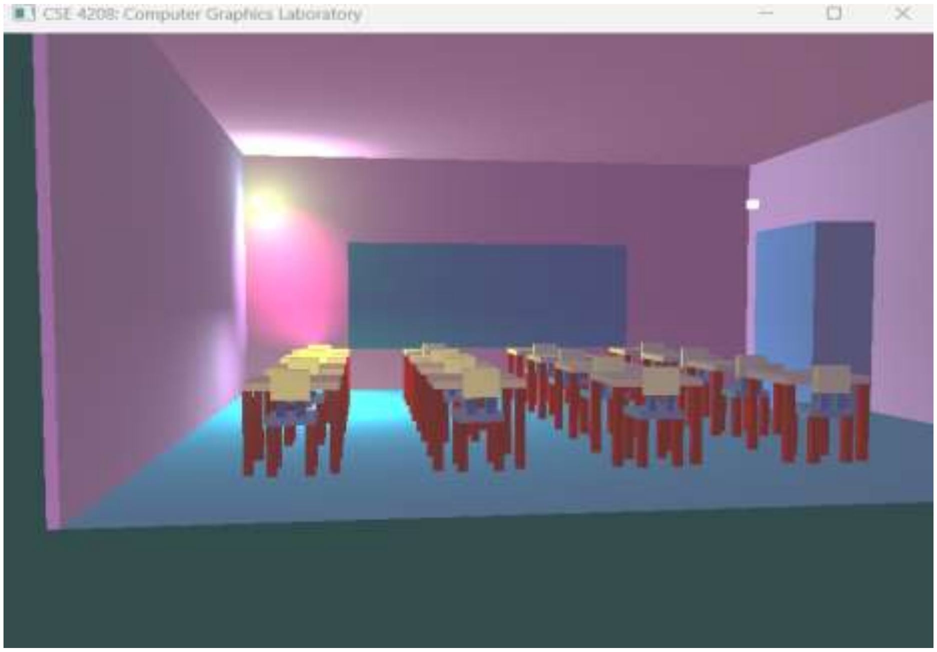

Fig 3.6 : Point light2 off

Spot light:

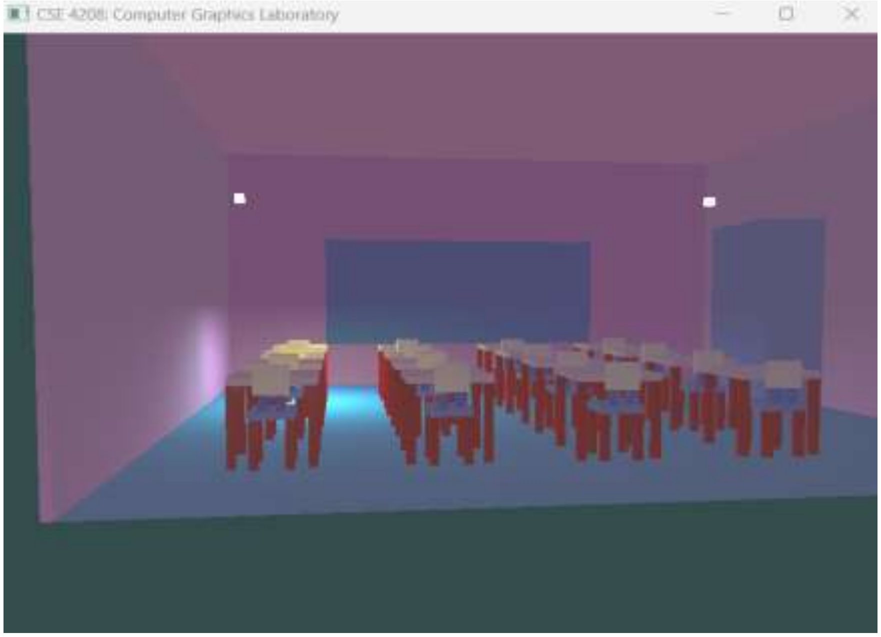

Fig 3.7 : Spot light on

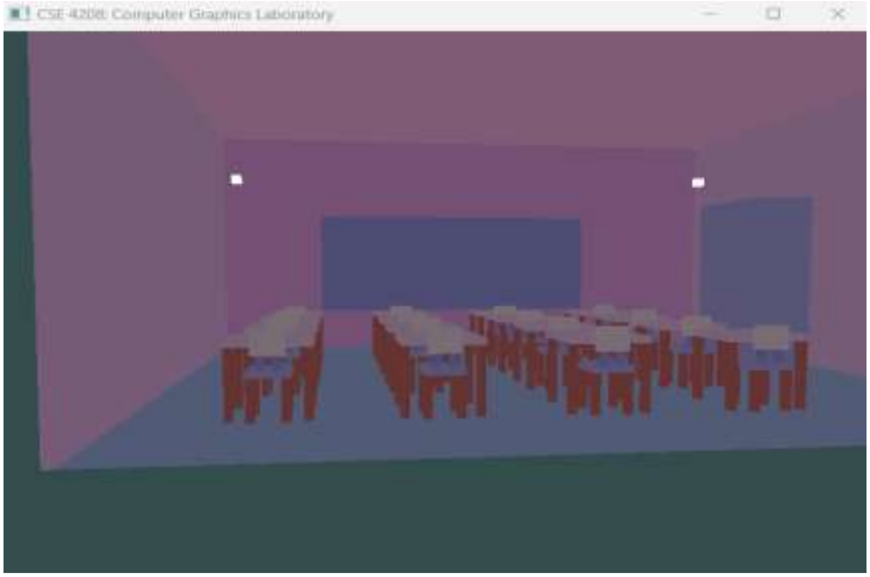

Fig 3.8 : Spot light off

Emissive light: Red maximize for understanding

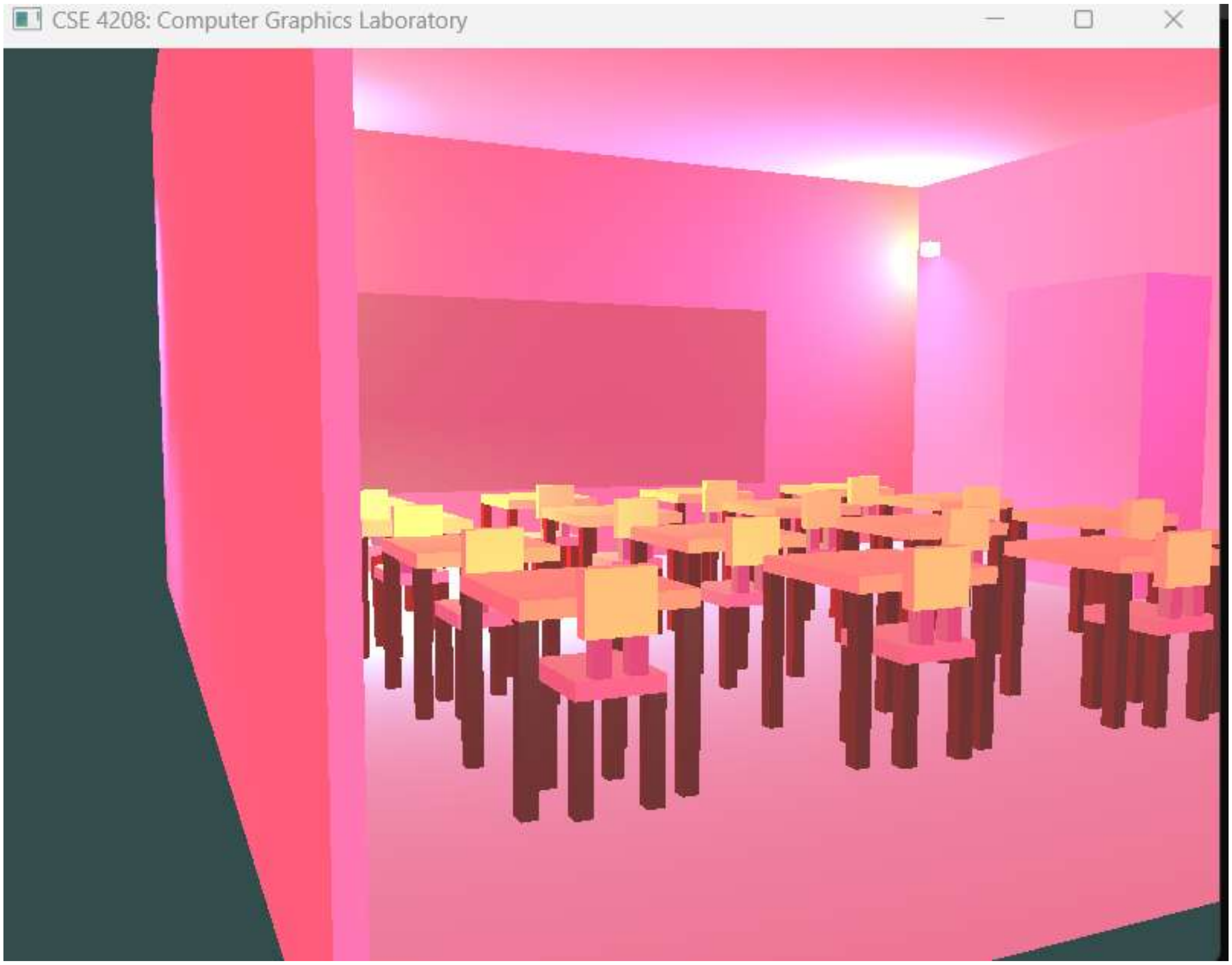

Fig 3.9 :Emissive light

Ambient light:

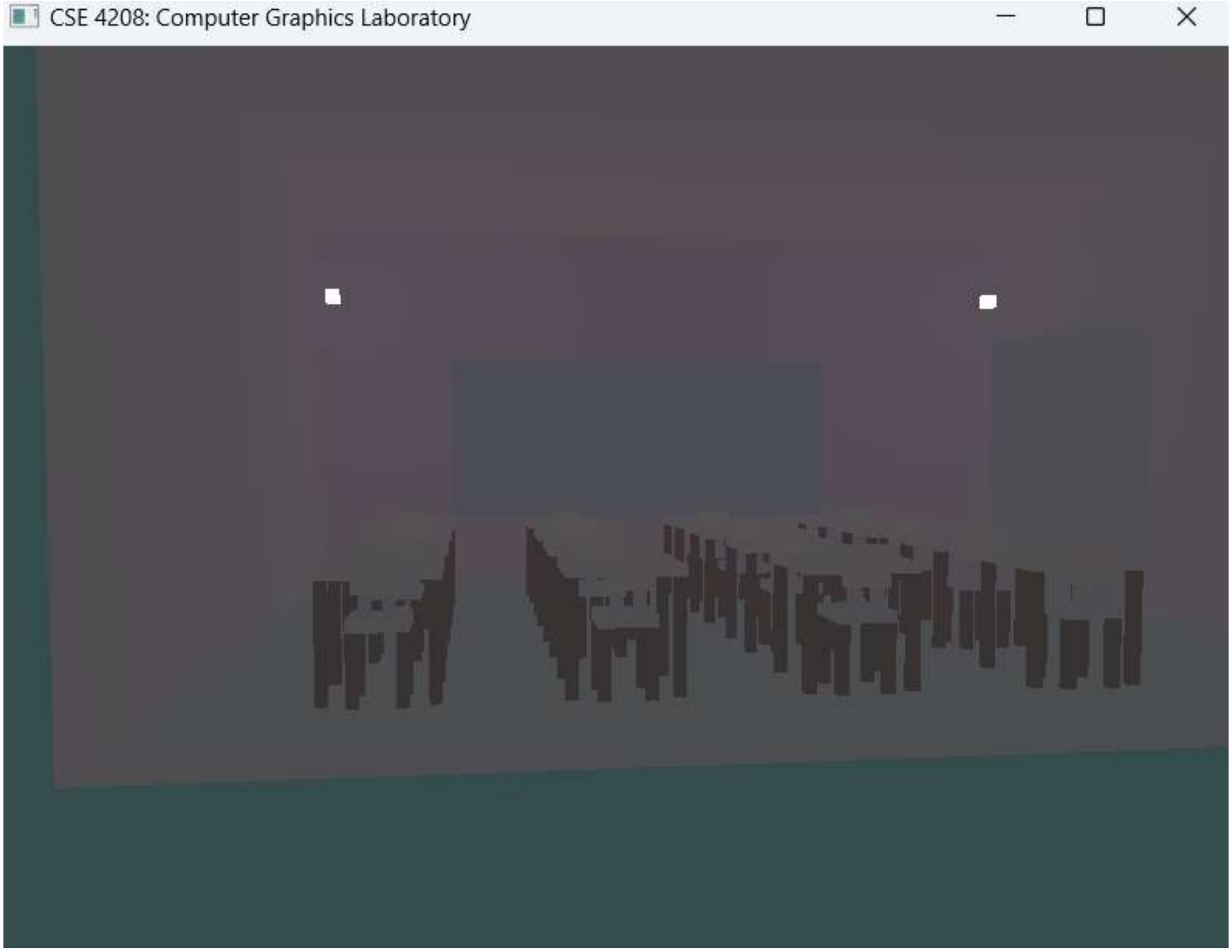

Fig 3.10 : Ambient light when all light off

Diffuse light:

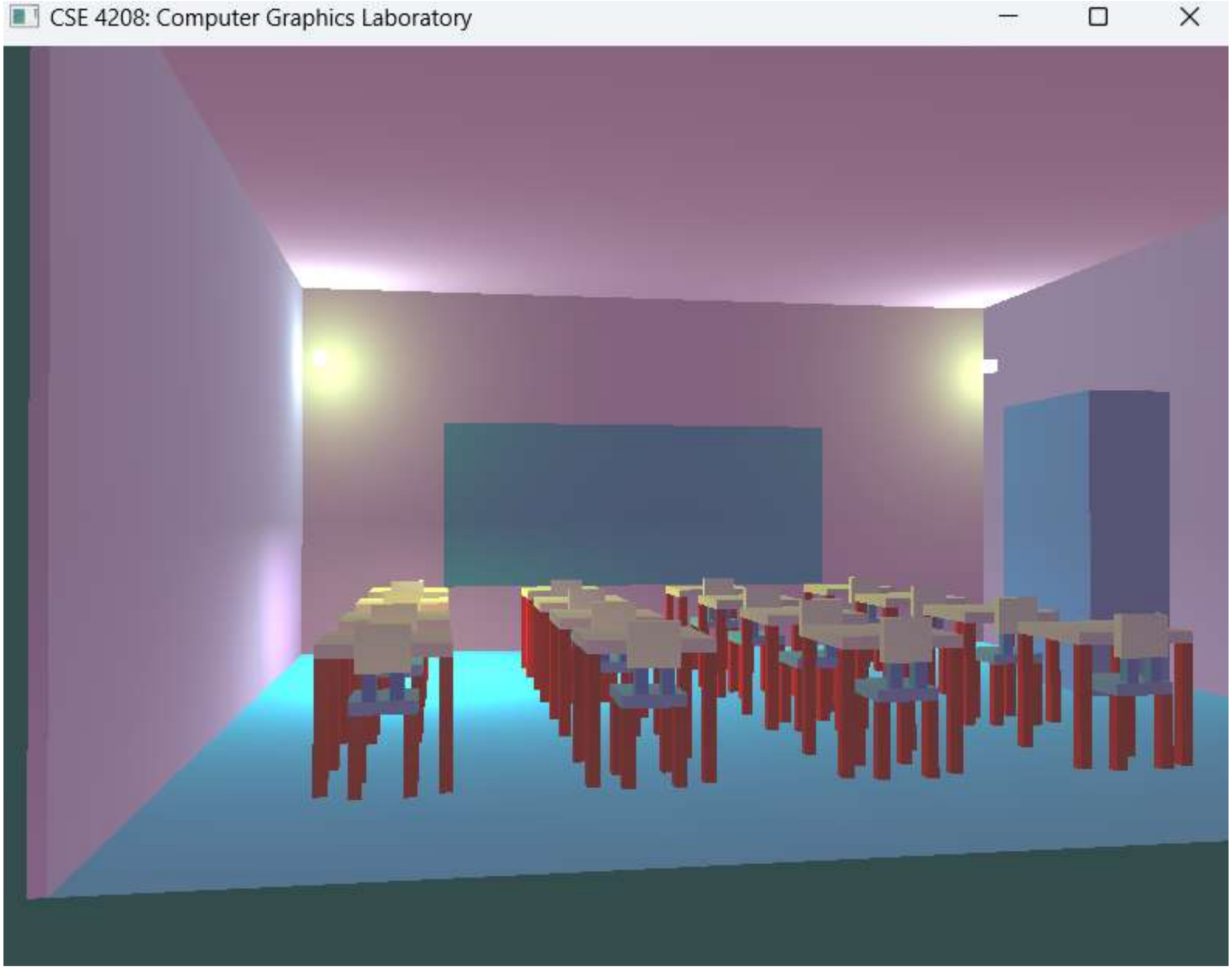

Fig 3.11 : Diffuse light

Specular light:

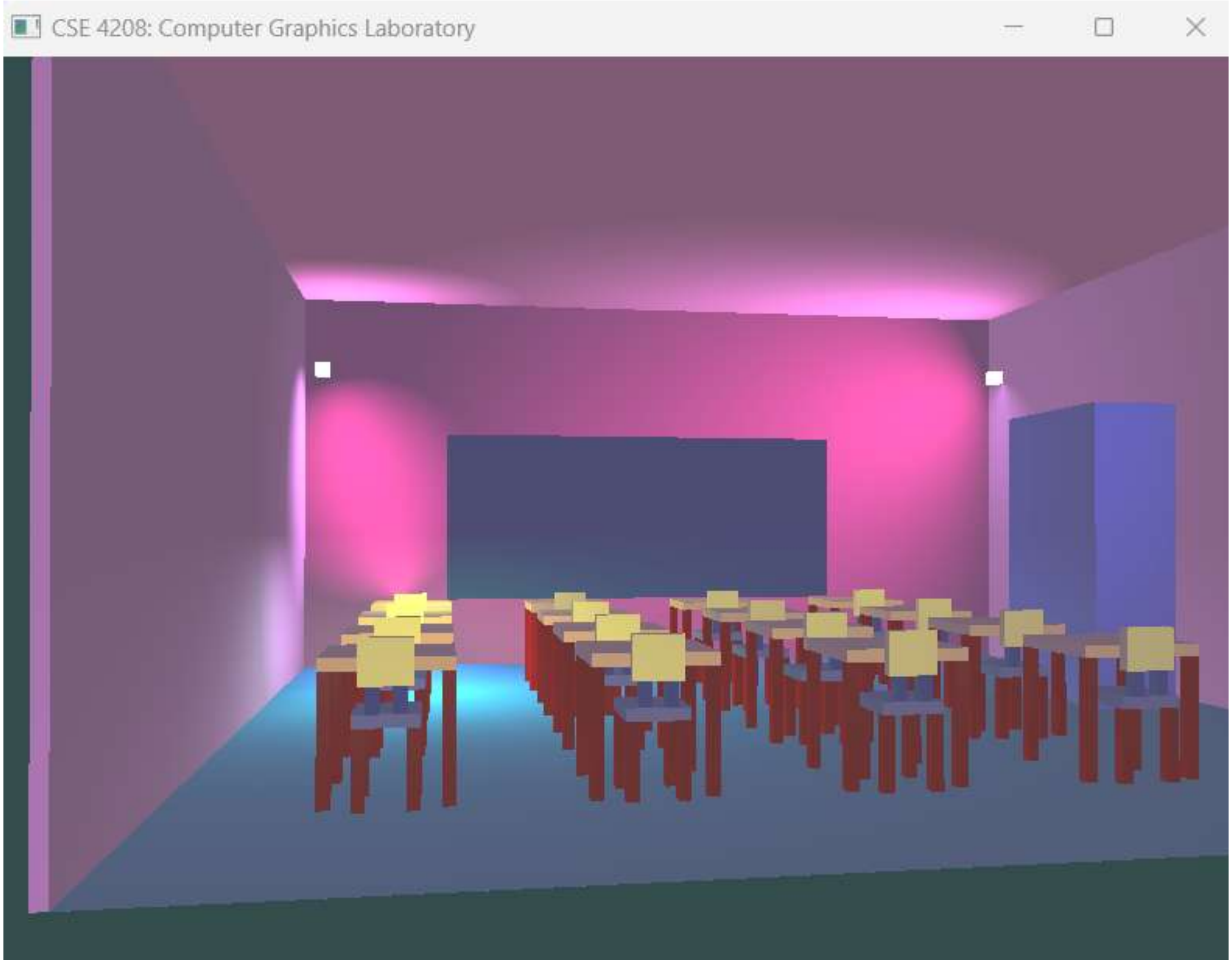

Fig 3.12 : Specular light

Pseudo code:

1.Projection function

```
// Function to calculate the perspective projection matrix

void perspectiveProjection(float fov, float aspect, float near, float far, float matrix[16]) {

    float tanHalfFov = tanf(fov / 2.0f);


    // Initialize all elements to zero

    for (int i = 0; i < 16; ++i)

        matrix[i] = 0.0f;

    matrix[0] = 1.0f / (aspect * tanHalfFov);  // (1 / (aspect * tan(fov / 2)))

    matrix[5] = 1.0f / tanHalfFov;  // (1 / tan(fov / 2))

    matrix[10] = -(far + near) / (far - near);  // (-(far + near) / (far - near))

    matrix[11] = -1.0f;  // -1

    matrix[14] = -(2 * far * near) / (far - near);  // (-(2 * far * near) / (far - near))

    matrix[15] = 0.0f;


    // Set the rest of the matrix elements to zero (by default, they are already zero)

    matrix[1] = matrix[2] = matrix[3] = 0.0f;

    matrix[4] = matrix[6] = matrix[7] = 0.0f;

    matrix[8] = matrix[9] = matrix[12] = matrix[13] = matrix[15] = 0.0f;

}

void orthogonalProjection(float left, float right, float bottom, float top, float near, float far, float matrix[16]) {

    // Compute the orthogonal projection matrix

    float r_l = right - left;

    float t_b = top - bottom;

    float n_f = far - near;
    // Set all elements of the matrix to 0
    for (int i = 0; i < 16; ++i)
        matrix[i] = 0.0f;

    // The orthogonal projection matrix (M_orth) structure:
    matrix[0] = 2.0f / r_l;  // (2 / (right - left))
    matrix[5] = 2.0f / t_b;  // (2 / (top - bottom))
    matrix[10] = 2.0f / n_f; // (2 / (near - far))
    matrix[12] = (right + left) / r_l;   // (right + left) / (right - left)
    matrix[13] = (top + bottom) / t_b;   // (top + bottom) / (top - bottom)
    matrix[14] = (far + near) / n_f;     // (far + near) / (near - far)
    matrix[15] = 1.0f;

    // Set other elements to 0 (already default, but to be explicit)
    matrix[1] = matrix[2] = matrix[3] = 0.0f;
    matrix[4] = matrix[6] = matrix[7] = 0.0f;
    matrix[8] = matrix[9] = matrix[11] = 0.0f;
}
```

2.Point light, Direction light, Specular light

```
glm::vec3 pointLightPositions[] = {
     glm::vec3(1.0f,  3.3f,  2.5f),
    glm::vec3(-9.50f,  3.4f,  2.5f),
};
PointLight pointlight1(

    pointLightPositions[0].x, pointLightPositions[0].y, pointLightPositions[0].z,  // position
    0.05f, 0.05f, 0.05f,     // ambient
    1.0f, 1.0f, 1.0f,     // diffuse
    1.0f, 1.0f, 1.0f,        // specular
    1.0f,   //k_c
    0.09f,  //k_l
    0.032f, //k_q
    1       // light number
);
PointLight pointlight2(

    pointLightPositions[1].x, pointLightPositions[1].y, pointLightPositions[1].z,  // position
    0.05f, 0.05f, 0.05f,     // ambient
    1.0f, 1.0f, 1.0f,     // diffuse
    1.0f, 1.0f, 1.0f,        // specular
    1.0f,   //k_c
    0.09f,  //k_l
    0.032f, //k_q
    2       // light number
);
    Shader lightingShader("vertexShaderForPhongShading.vs", "fragmentShaderForPhongShading.fs");
    Shader ourShader("vertexShader.vs", "fragmentShader.fs");
    float cube_vertices[] = {
        0.0f, 0.0f, 0.0f, 0.0f, 0.0f, -1.0f,
        0.5f, 0.0f, 0.0f, 0.0f, 0.0f, -1.0f,
        0.5f, 0.5f, 0.0f, 0.0f, 0.0f, -1.0f,
        0.0f, 0.5f, 0.0f, 0.0f, 0.0f, -1.0f,
…….
    };
    unsigned int cube_indices[] = {
        0, 3, 2,
        2, 1, 0,
……
    };

        // point light 1
        pointlight1.setUpPointLight(lightingShader);
        // point light 2
        pointlight2.setUpPointLight(lightingShader);
        lightingShader.setVec3("diectionalLight.directiaon", 0.0f, 0.0f, -3.0f);
        lightingShader.setVec3("diectionalLight.ambient", .2, .2, .2);
        lightingShader.setVec3("diectionalLight.diffuse", .8f, .8f, .8f);
        lightingShader.setVec3("diectionalLight.specular", 1.0f, 1.0f, 1.0f);
        if (directionToggle)
        {

            lightingShader.setBool("dlighton", true);
        }
        else if(!directionToggle)
        {
            lightingShader.setBool("dlighton", false);
        }
```

```
lightingShader.setVec3("spotlight.position", -0.5, 1, -0.5);
lightingShader.setVec3("spotlight.direction", 0, -1, 0);
lightingShader.setVec3("spotlight.ambient", .2, .2, .2);
lightingShader.setVec3("spotlight.diffuse", .8f, .8f, .8f);
lightingShader.setVec3("spotlight.specular", 1.0f, 1.0f, 1.0f);
lightingShader.setFloat("spotlight.k_c", 1.0f);
lightingShader.setFloat("spotlight.k_l", 0.09);
lightingShader.setFloat("spotlight.k_q", 0.032);
lightingShader.setFloat("cos_theta", glm::cos(glm::radians(5.5f)));
if (spotToggle)
{
   lightingShader.setBool("spotlighton", true)
}
else if (!spotToggle)
{
   lightingShader.setBool("spotlighton", false);

}
lightingShader.use();
```

3.Draw bulb and other object in lightingshader

```
glm::mat4 identityMatrix = glm::mat4(1.0f); // make sure to initialize matrix to identity matrix first
glm::mat4 translateMatrix, rotateXMatrix, rotateYMatrix, rotateZMatrix, scaleMatrix, model;
translateMatrix = glm::translate(identityMatrix, glm::vec3(translate_X, translate_Y, translate_Z));
rotateXMatrix = glm::rotate(identityMatrix, glm::radians(rotateAngle_X), glm::vec3(1.0f, 0.0f, 0.0f));
rotateYMatrix = glm::rotate(identityMatrix, glm::radians(rotateAngle_Y), glm::vec3(0.0f, 1.0f, 0.0f));
rotateZMatrix = glm::rotate(identityMatrix, glm::radians(rotateAngle_Z), glm::vec3(0.0f, 0.0f, 1.0f));
scaleMatrix = glm::scale(identityMatrix, glm::vec3(scale_X, scale_Y, scale_Z));
```

```
model = translateMatrix * rotateXMatrix * rotateYMatrix * rotateZMatrix * scaleMatrix;

lightingShader.setMat4("model", model);

board(VAO, lightingShader, model);


// also draw the lamp object(s)

ourShader.use();

//ourShader.setMat4("projection", projection);

GLint projectionLoc1 = glGetUniformLocation(ourShader.ID, "projection");


// Send the orthogonal projection matrix to the shader

glUniformMatrix4fv(projectionLoc1, 1, GL_FALSE, projectionMatrix);

ourShader.setMat4("view", view);


// we now draw as many light bulbs as we have point lights.

glBindVertexArray(lightCubeVAO);

for (unsigned int i = 0; i < 2; i++)

{

   model = glm::mat4(1.0f);

   model = glm::translate(model, pointLightPositions[i]);

   model = glm::scale(model, glm::vec3(0.4f)); // Make it a smaller cube

   ourShader.setMat4("model", model);

   ourShader.setVec4("color", glm::vec4(1.0f, 1.0f, 1.0f,1.0f));

   glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_INT, 0);

   //glDrawArrays(GL_TRIANGLES, 0, 36);

}
```

4.Keyboard implementation

```
if (glfwGetKey(window, GLFW_KEY_1) == GLFW_PRESS)

 {

   if (directionToggle)

   {

     directionToggle = !directionToggle;

   }

   else

   {    directionToggle = !directionToggle;

 }

 if (glfwGetKey(window, GLFW_KEY_2) == GLFW_PRESS)

 {

   if (point1Toggle == true)

   {

     pointlight1.turnOff();

     point1Toggle = !point1Toggle;

   }

   else

   {

     pointlight1.turnOn();

     point1Toggle = !point1Toggle;

   }

 }

 if (glfwGetKey(window, GLFW_KEY_3) == GLFW_PRESS)

 {

   if (point2Toggle == true)

   {

     pointlight2.turnOff();
```

```
        point2Toggle = !point1Toggle;
    }
    else
    {
        pointlight2.turnOn();
        point2Toggle = !point2Toggle;
    }
if (glfwGetKey(window, GLFW_KEY_4) == GLFW_PRESS)
{
    if (spotToggle)
    {
        spotToggle = !spotToggle;
    }
    else
    {
        spotToggle = !spotToggle;
    }
}
if (glfwGetKey(window, GLFW_KEY_6) == GLFW_PRESS)
{
    if (diffuseToggle)
    {
        pointlight1.turnDiffuseOff();
        pointlight2.turnDiffuseOff();
        diffuseToggle = !diffuseToggle;
    }
    else
    {

        pointlight1.turnDiffuseOn();
       pointlight2.turnDiffuseOn();

        diffuseToggle = !diffuseToggle;
    }

}
if (glfwGetKey(window, GLFW_KEY_7) == GLFW_PRESS)
{
    if (specularToggle)
    {

        pointlight1.turnSpecularOff();
        pointlight2.turnSpecularOff();
        specularToggle = !specularToggle;
    }
    else
    { pointlight1.turnSpecularOn();
        pointlight2.turnSpecularOn();

        specularToggle = !specularToggle;
    }
}
if (glfwGetKey(window, GLFW_KEY_5) == GLFW_PRESS)
{
        if (ambientToggle)

           pointlight1.turnAmbientOff();

           pointlight2.turnAmbientOff();

           ambientToggle = !ambientToggle;

        }

        else



           pointlight1.turnAmbientOn();

           pointlight2.turnAmbientOn();

           ambientToggle = !ambientToggle;

        }


}
```