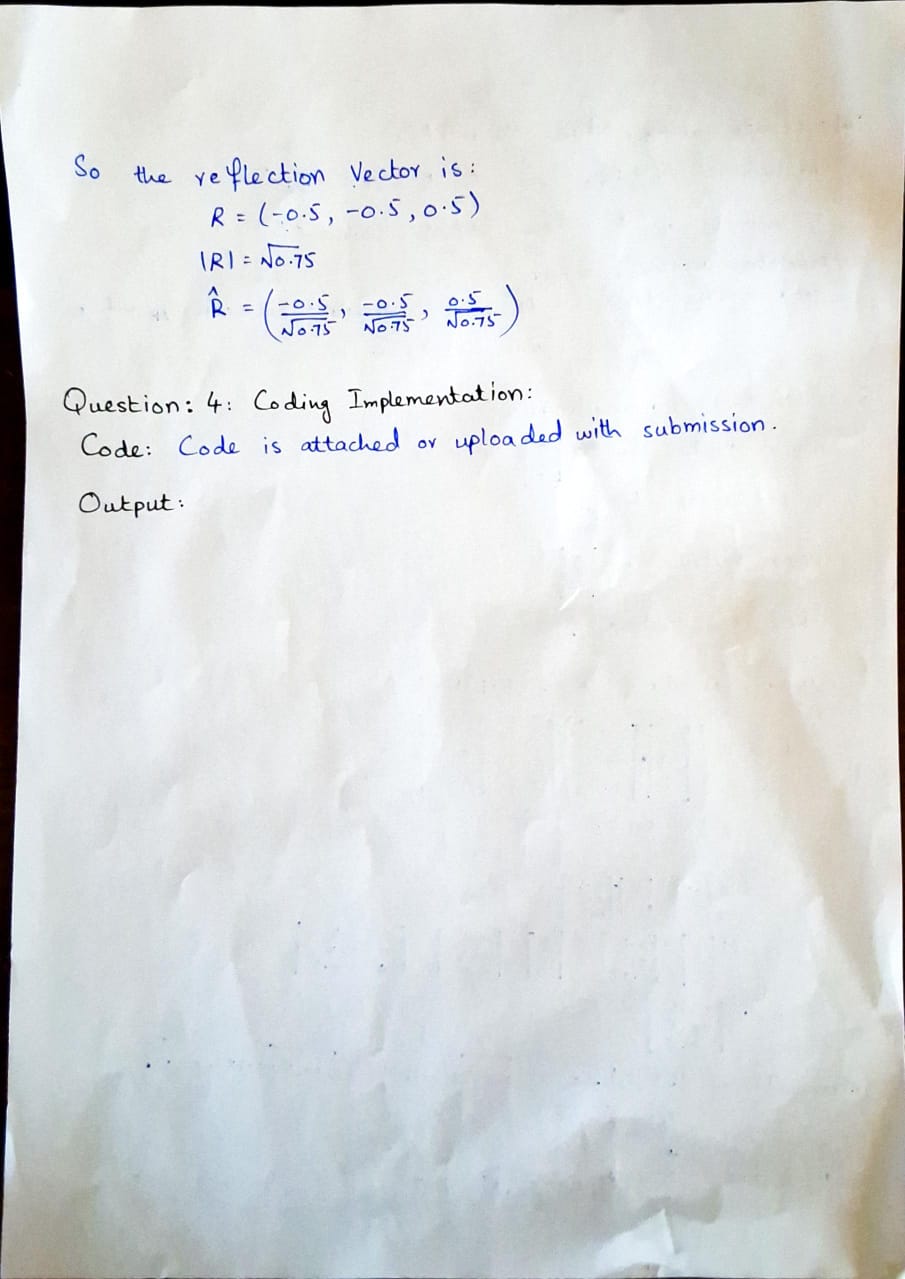
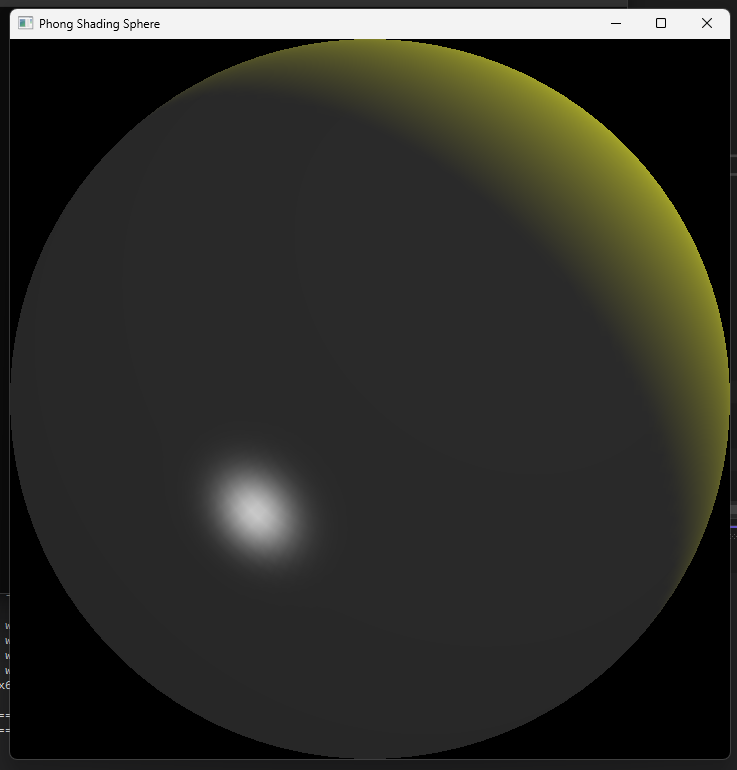
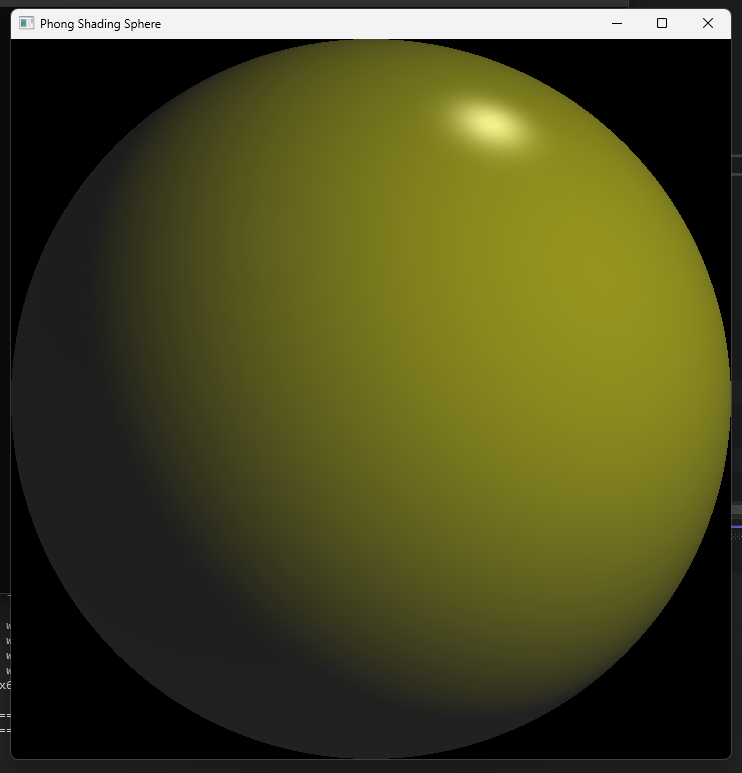


A piece of paper with writing on it

AI-generated content may be incorrect.A piece of paper with writing on it

AI-generated content may be incorrect.

Use WASD to rotate the 3D scene.

A piece of paper with blue ink

AI-generated content may be incorrect.

#include <GL/glut.h>

#include <cmath>

#include <corecrt\_math\_defines.h>

#include <iostream>

// Light position

GLfloat lightPos[] = { 10.0f, 10.0f, 10.0f, 1.0f };

// Sphere rotation angles

GLfloat angleX = 0.0f;

GLfloat angleY = 0.0f;

// Phong shading parameters

GLfloat ambient[] = { 0.2f, 0.2f, 0.2f, 1.0f };

GLfloat diffuse[] = { 0.8f, 0.8f, 0.0f, 1.0f };

GLfloat specular[] = { 0.8f, 0.8f, 0.8f, 1.0f };

GLfloat shininess = 50.0f;

// Function to normalize a vector

void normalize(float v[3]) {

float length = sqrt(v[0] \* v[0] + v[1] \* v[1] + v[2] \* v[2]);

if (length != 0.0f) {

v[0] /= length;

v[1] /= length;

v[2] /= length;

}

}

// Function to calculate the reflection vector

void reflect(float light[3], float normal[3], float reflection[3]) {

float dot = light[0] \* normal[0] + light[1] \* normal[1] + light[2] \* normal[2];

reflection[0] = 2 \* dot \* normal[0] - light[0];

reflection[1] = 2 \* dot \* normal[1] - light[1];

reflection[2] = 2 \* dot \* normal[2] - light[2];

}

// Function to calculate Phong shading

void phongShading(float normal[3], float light[3], float view[3], float ambientColor[4], float diffuseColor[4], float specularColor[4], float shininess, float outputColor[4]) {

// Ambient component

outputColor[0] = ambientColor[0];

outputColor[1] = ambientColor[1];

outputColor[2] = ambientColor[2];

outputColor[3] = ambientColor[3];

// Diffuse component

float diffuseIntensity = std::max(0.0f, normal[0] \* light[0] + normal[1] \* light[1] + normal[2] \* light[2]);

outputColor[0] += diffuseColor[0] \* diffuseIntensity;

outputColor[1] += diffuseColor[1] \* diffuseIntensity;

outputColor[2] += diffuseColor[2] \* diffuseIntensity;

// Specular component

float reflection[3];

reflect(light, normal, reflection);

float specularIntensity = std::pow(std::max(0.0f, reflection[0] \* view[0] + reflection[1] \* view[1] + reflection[2] \* view[2]), shininess);

outputColor[0] += specularColor[0] \* specularIntensity;

outputColor[1] += specularColor[1] \* specularIntensity;

outputColor[2] += specularColor[2] \* specularIntensity;

}

// Function to draw a sphere with Phong shading

void drawSphere() {

int slices = 100;

int stacks = 100;

float radius = 1.0f;

for (int i = 0; i < slices; ++i) {

float theta1 = i \* 2 \* M\_PI / slices;

float theta2 = (i + 1) \* 2 \* M\_PI / slices;

for (int j = 0; j < stacks; ++j) {

float phi1 = j \* M\_PI / stacks;

float phi2 = (j + 1) \* M\_PI / stacks;

// Vertices

float v1[3] = { radius \* sin(phi1) \* cos(theta1), radius \* sin(phi1) \* sin(theta1), radius \* cos(phi1) };

float v2[3] = { radius \* sin(phi1) \* cos(theta2), radius \* sin(phi1) \* sin(theta2), radius \* cos(phi1) };

float v3[3] = { radius \* sin(phi2) \* cos(theta2), radius \* sin(phi2) \* sin(theta2), radius \* cos(phi2) };

float v4[3] = { radius \* sin(phi2) \* cos(theta1), radius \* sin(phi2) \* sin(theta1), radius \* cos(phi2) };

// Normals

float n1[3] = { v1[0], v1[1], v1[2] };

float n2[3] = { v2[0], v2[1], v2[2] };

float n3[3] = { v3[0], v3[1], v3[2] };

float n4[3] = { v4[0], v4[1], v4[2] };

normalize(n1);

normalize(n2);

normalize(n3);

normalize(n4);

// View vector (assuming camera at (0, 0, 5))

float view[3] = { 0.0f, 0.0f, 5.0f };

normalize(view);

// Light vector

float light[3] = { lightPos[0] - v1[0], lightPos[1] - v1[1], lightPos[2] - v1[2] };

normalize(light);

// Calculate Phong shading for each vertex

float color1[4], color2[4], color3[4], color4[4];

phongShading(n1, light, view, ambient, diffuse, specular, shininess, color1);

phongShading(n2, light, view, ambient, diffuse, specular, shininess, color2);

phongShading(n3, light, view, ambient, diffuse, specular, shininess, color3);

phongShading(n4, light, view, ambient, diffuse, specular, shininess, color4);

// Draw the quad

glBegin(GL\_QUADS);

glColor4fv(color1);

glVertex3fv(v1);

glColor4fv(color2);

glVertex3fv(v2);

glColor4fv(color3);

glVertex3fv(v3);

glColor4fv(color4);

glVertex3fv(v4);

glEnd();

}

}

}

// Display function

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glLoadIdentity();

// Set light position

glLightfv(GL\_LIGHT0, GL\_POSITION, lightPos);

// Rotate the sphere

glRotatef(angleX, 1.0f, 0.0f, 0.0f);

glRotatef(angleY, 0.0f, 1.0f, 0.0f);

// Draw the sphere

drawSphere();

glutSwapBuffers();

}

// Keyboard function for rotation

void keyboard(unsigned char key, int x, int y) {

switch (key) {

case 'd':

angleX += 5.0f;

break;

case 'a':

angleX -= 5.0f;

break;

case 'w':

angleY += 5.0f;

break;

case 's':

angleY -= 5.0f;

break;

}

glutPostRedisplay();

}

// Main function

int main(int argc, char\*\* argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(720, 720);

glutCreateWindow("Phong Shading Sphere");

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glEnable(GL\_COLOR\_MATERIAL);

glutDisplayFunc(display);

glutKeyboardFunc(keyboard);

glutMainLoop();

return 0;

}