

| **TITLE**: Write an OpenGL program to implement Shadow Mapping. |
| --- |

**AIM:**

Write an OpenGL program to implement Shadow Mapping.

Create 3D object and demonstrate the shadow of same object.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

Understand the computer Input & interaction, Curves and Computer Animation

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Books/ Journals/ Websites referred:**

http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Algorithm/ Pseudocode for each process:**

1. Render the scene using the light as the camera and perform z-buffering

2. Generate a light z buffer

3. Render the scene using the regular camera, perform z-buffering

**Implementation details:**

**#include <GL/glut.h>**

**#include <GL/gl.h>**

**// Light position for shadow calculation**

**GLfloat lightPos[] = {5.0f, 8.0f, 0.0f, 1.0f};**

**// Function to initialize the environment**

**void init() {**

**glClearColor(0.0, 0.0, 0.0, 1.0); // Black background**

**glEnable(GL\_DEPTH\_TEST); // Enable depth testing**

**glShadeModel(GL\_SMOOTH); // Smooth shading**

**glEnable(GL\_LIGHTING); // Enable lighting**

**glEnable(GL\_LIGHT0); // Enable light source 0**

**glEnable(GL\_COLOR\_MATERIAL); // Enable color tracking**

**// Enable blending for transparent shadows**

**glEnable(GL\_BLEND);**

**glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);**

**}**

**// Function to calculate the shadow projection matrix**

**void shadowMatrix(GLfloat shadowMat[4][4], GLfloat groundplane[4], GLfloat lightpos[4]) {**

**GLfloat dot = groundplane[0] \* lightpos[0] +**

**groundplane[1] \* lightpos[1] +**

**groundplane[2] \* lightpos[2] +**

**groundplane[3] \* lightpos[3];**

**shadowMat[0][0] = dot - lightpos[0] \* groundplane[0];**

**shadowMat[1][0] = 0.0f - lightpos[0] \* groundplane[1];**

**shadowMat[2][0] = 0.0f - lightpos[0] \* groundplane[2];**

**shadowMat[3][0] = 0.0f - lightpos[0] \* groundplane[3];**

**shadowMat[0][1] = 0.0f - lightpos[1] \* groundplane[0];**

**shadowMat[1][1] = dot - lightpos[1] \* groundplane[1];**

**shadowMat[2][1] = 0.0f - lightpos[1] \* groundplane[2];**

**shadowMat[3][1] = 0.0f - lightpos[1] \* groundplane[3];**

**shadowMat[0][2] = 0.0f - lightpos[2] \* groundplane[0];**

**shadowMat[1][2] = 0.0f - lightpos[2] \* groundplane[1];**

**shadowMat[2][2] = dot - lightpos[2] \* groundplane[2];**

**shadowMat[3][2] = 0.0f - lightpos[2] \* groundplane[3];**

**shadowMat[0][3] = 0.0f - lightpos[3] \* groundplane[0];**

**shadowMat[1][3] = 0.0f - lightpos[3] \* groundplane[1];**

**shadowMat[2][3] = 0.0f - lightpos[3] \* groundplane[2];**

**shadowMat[3][3] = dot - lightpos[3] \* groundplane[3];**

**}**

**// Function to draw the scene (with shadows)**

**void display() {**

**glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // Clear color and depth buffers**

**// Set up lighting**

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

**// Set the view perspective**

**glMatrixMode(GL\_MODELVIEW);**

**glLoadIdentity();**

**gluLookAt(5.0, 5.0, 10.0, // Camera position**

**0.0, 0.0, 0.0, // Look at point**

**0.0, 1.0, 0.0); // Up vector**

**// Ground plane**

**glColor3f(0.5f, 0.5f, 0.5f); // Gray color for the ground**

**glBegin(GL\_QUADS);**

**glVertex3f(-5.0f, -1.0f, -5.0f);**

**glVertex3f(5.0f, -1.0f, -5.0f);**

**glVertex3f(5.0f, -1.0f, 5.0f);**

**glVertex3f(-5.0f, -1.0f, 5.0f);**

**glEnd();**

**// Define ground plane equation (y = -1)**

**GLfloat groundPlane[] = {0.0f, 1.0f, 0.0f, 1.0f}; // Plane equation Ax + By + Cz + D = 0 -> {A, B, C, D}**

**// Calculate shadow matrix**

**GLfloat shadowMat[4][4];**

**shadowMatrix(shadowMat, groundPlane, lightPos);**

**// Render the sphere (normal render)**

**glColor3f(1.0f, 0.0f, 1.0f); // Magenta color for the sphere**

**glPushMatrix();**

**glTranslatef(0.0f, 0.0f, 0.0f); // Position the sphere**

**glutSolidSphere(1.0, 50, 50); // Create a solid sphere with radius 1**

**glPopMatrix();**

**// Apply shadow matrix and render shadow (flattened onto ground)**

**glDisable(GL\_LIGHTING); // Disable lighting for the shadow (it should be dark)**

**glColor4f(1.0f, 1.0f, 1.0f, 0.75f); // White semi-transparent shadow color**

**glPushMatrix();**

**glMultMatrixf((GLfloat \*)shadowMat); // Multiply the current matrix with the shadow matrix**

**glTranslatef(0.0f, 0.0f, 0.0f); // Position the shadow where the sphere is**

**glutSolidSphere(1.0, 50, 50); // Create the shadow as a flattened sphere**

**glPopMatrix();**

**glEnable(GL\_LIGHTING); // Re-enable lighting**

**glutSwapBuffers(); // Swap buffers to display the result**

**}**

**// Reshape function to handle window resizing**

**void reshape(int width, int height) {**

**if (height == 0) height = 1;**

**float aspect = (float)width / (float)height;**

**glViewport(0, 0, width, height);**

**glMatrixMode(GL\_PROJECTION);**

**glLoadIdentity();**

**gluPerspective(45.0, aspect, 0.1, 100.0);**

**}**

**// Main function to set up the GLUT environment and run the application**

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

**glutInit(&argc, argv);**

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

**glutInitWindowSize(800, 600);**

**glutCreateWindow("Shadow Mapping");**

**init();**

**glutDisplayFunc(display);**

**glutReshapeFunc(reshape);**

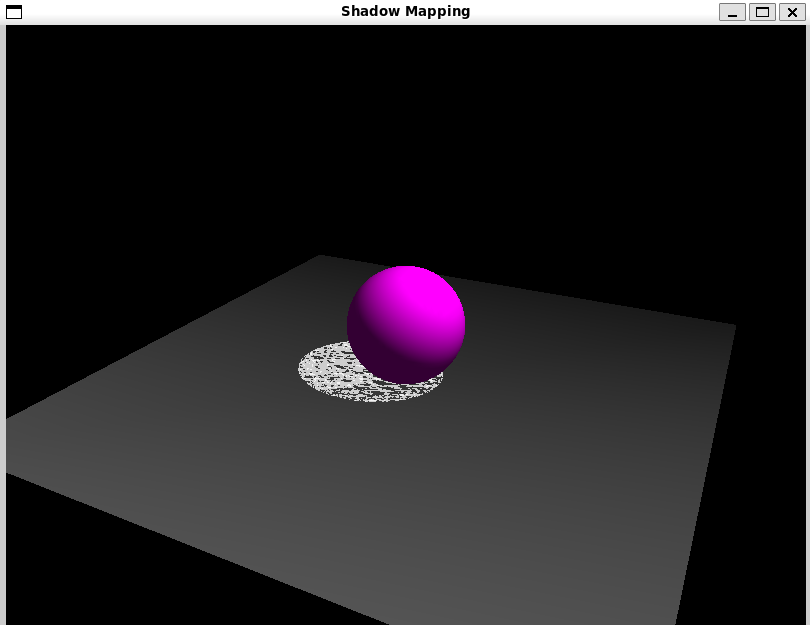
**glutMainLoop(); // Enter the event-processing loop**

**return 0;**

**}**

**Output(s) (Screen Shot):**

****



**Conclusion and discussion:**

**We have implemented Shadow Mapping.**

**Date: 29/9/24**

**Signature of faculty in-charge**

**Post lab question**

**Write a program to demonstrate shadow for two objects.**

**#include <GL/glut.h>**

**#include <GL/gl.h>**

**// Light position for shadow calculation**

**GLfloat lightPos[] = {5.0f, 8.0f, 5.0f, 1.0f};**

**// Function to initialize the environment**

**void init() {**

**glClearColor(0.0, 0.0, 0.0, 1.0); // Black background**

**glEnable(GL\_DEPTH\_TEST); // Enable depth testing**

**glShadeModel(GL\_SMOOTH); // Smooth shading**

**glEnable(GL\_LIGHTING); // Enable lighting**

**glEnable(GL\_LIGHT0); // Enable light source 0**

**glEnable(GL\_COLOR\_MATERIAL); // Enable color tracking**

**// Enable blending for transparent shadows**

**glEnable(GL\_BLEND);**

**glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);**

**}**

**// Function to calculate the shadow projection matrix**

**void shadowMatrix(GLfloat shadowMat[4][4], GLfloat groundplane[4], GLfloat lightpos[4]) {**

**GLfloat dot = groundplane[0] \* lightpos[0] +**

**groundplane[1] \* lightpos[1] +**

**groundplane[2] \* lightpos[2] +**

**groundplane[3] \* lightpos[3];**

**shadowMat[0][0] = dot - lightpos[0] \* groundplane[0];**

**shadowMat[1][0] = 0.0f - lightpos[0] \* groundplane[1];**

**shadowMat[2][0] = 0.0f - lightpos[0] \* groundplane[2];**

**shadowMat[3][0] = 0.0f - lightpos[0] \* groundplane[3];**

**shadowMat[0][1] = 0.0f - lightpos[1] \* groundplane[0];**

**shadowMat[1][1] = dot - lightpos[1] \* groundplane[1];**

**shadowMat[2][1] = 0.0f - lightpos[1] \* groundplane[2];**

**shadowMat[3][1] = 0.0f - lightpos[1] \* groundplane[3];**

**shadowMat[0][2] = 0.0f - lightpos[2] \* groundplane[0];**

**shadowMat[1][2] = 0.0f - lightpos[2] \* groundplane[1];**

**shadowMat[2][2] = dot - lightpos[2] \* groundplane[2];**

**shadowMat[3][2] = 0.0f - lightpos[2] \* groundplane[3];**

**shadowMat[0][3] = 0.0f - lightpos[3] \* groundplane[0];**

**shadowMat[1][3] = 0.0f - lightpos[3] \* groundplane[1];**

**shadowMat[2][3] = 0.0f - lightpos[3] \* groundplane[2];**

**shadowMat[3][3] = dot - lightpos[3] \* groundplane[3];**

**}**

**// Function to draw the scene (with shadows)**

**void display() {**

**glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // Clear color and depth buffers**

**// Set up lighting**

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

**// Set the view perspective**

**glMatrixMode(GL\_MODELVIEW);**

**glLoadIdentity();**

**gluLookAt(5.0, 5.0, 10.0, // Camera position**

**0.0, 0.0, 0.0, // Look at point**

**0.0, 1.0, 0.0); // Up vector**

**// Ground plane**

**glColor3f(0.5f, 0.5f, 0.5f); // Gray color for the ground**

**glBegin(GL\_QUADS);**

**glVertex3f(-5.0f, -1.0f, -5.0f);**

**glVertex3f(5.0f, -1.0f, -5.0f);**

**glVertex3f(5.0f, -1.0f, 5.0f);**

**glVertex3f(-5.0f, -1.0f, 5.0f);**

**glEnd();**

**// Define ground plane equation (y = -1)**

**GLfloat groundPlane[] = {0.0f, 1.0f, 0.0f, 1.0f}; // Plane equation Ax + By + Cz + D = 0**

**// Calculate shadow matrix**

**GLfloat shadowMat[4][4];**

**shadowMatrix(shadowMat, groundPlane, lightPos);**

**// Render the sphere (normal render)**

**glColor3f(1.0f, 0.0f, 1.0f); // Magenta color for the sphere**

**glPushMatrix();**

**glTranslatef(-2.0f, 0.0f, 0.0f); // Position the sphere**

**glutSolidSphere(1.0, 50, 50); // Create a solid sphere with radius 1**

**glPopMatrix();**

**// Apply shadow matrix and render shadow (flattened onto ground)**

**glDisable(GL\_LIGHTING); // Disable lighting for the shadow**

**glColor4f(1.0f, 1.0f, 1.0f, 0.75f); // White semi-transparent shadow color**

**glPushMatrix();**

**glMultMatrixf((GLfloat \*)shadowMat); // Multiply the current matrix with the shadow matrix**

**glTranslatef(-2.0f, 0.0f, 0.0f); // Position the shadow where the sphere is**

**glutSolidSphere(1.0, 50, 50); // Create the shadow as a flattened sphere**

**glPopMatrix();**

**glEnable(GL\_LIGHTING); // Re-enable lighting**

**// Render the cube (normal render)**

**glColor3f(0.0f, 1.0f, 0.0f); // Green color for the cube**

**glPushMatrix();**

**glTranslatef(2.0f, 0.0f, 0.0f); // Position the cube**

**glutSolidCube(1.0); // Create a solid cube with size 1**

**glPopMatrix();**

**// Apply shadow matrix and render shadow for the cube**

**glDisable(GL\_LIGHTING); // Disable lighting for the shadow**

**glColor4f(1.0f, 1.0f, 1.0f, 0.75f); // White semi-transparent shadow color**

**glPushMatrix();**

**glMultMatrixf((GLfloat \*)shadowMat); // Multiply the current matrix with the shadow matrix**

**glTranslatef(2.0f, 0.0f, 0.0f); // Position the shadow where the cube is**

**glutSolidCube(1.0); // Create the shadow as a flattened cube**

**glPopMatrix();**

**glEnable(GL\_LIGHTING); // Re-enable lighting**

**glutSwapBuffers(); // Swap buffers to display the result**

**}**

**// Reshape function to handle window resizing**

**void reshape(int width, int height) {**

**if (height == 0) height = 1;**

**float aspect = (float)width / (float)height;**

**glViewport(0, 0, width, height);**

**glMatrixMode(GL\_PROJECTION);**

**glLoadIdentity();**

**gluPerspective(45.0, aspect, 0.1, 100.0);**

**}**

**// Main function to set up the GLUT environment and run the application**

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

**glutInit(&argc, argv);**

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

**glutInitWindowSize(800, 600);**

**glutCreateWindow("Shadow Mapping 2");**

**init();**

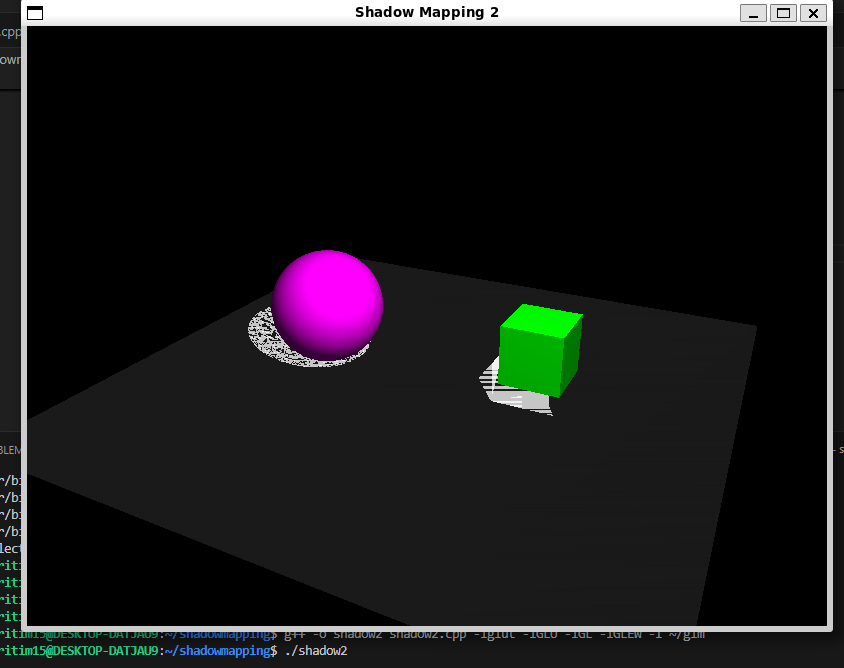
**glutDisplayFunc(display);**

**glutReshapeFunc(reshape);**

**glutMainLoop(); // Enter the event-processing loop**

**return 0;**

**}**

****

**Write to program to implement various curves (at least two - three types of curve)**

**#include <GL/glut.h>**

**#include <GL/gl.h>**

**#include <vector>**

**// Light position for shadow calculation**

**GLfloat lightPos[] = {5.0f, 8.0f, 0.0f, 1.0f};**

**// Function to initialize the environment**

**void init() {**

**glClearColor(0.0, 0.0, 0.0, 1.0); // Black background**

**glEnable(GL\_DEPTH\_TEST); // Enable depth testing**

**glShadeModel(GL\_SMOOTH); // Smooth shading**

**glEnable(GL\_LIGHTING); // Enable lighting**

**glEnable(GL\_LIGHT0); // Enable light source 0**

**glEnable(GL\_COLOR\_MATERIAL); // Enable color tracking**

**// Enable blending for transparent shadows**

**glEnable(GL\_BLEND);**

**glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);**

**}**

**// Function to calculate the shadow projection matrix**

**void shadowMatrix(GLfloat shadowMat[4][4], GLfloat groundplane[4], GLfloat lightpos[4]) {**

**GLfloat dot = groundplane[0] \* lightpos[0] +**

**groundplane[1] \* lightpos[1] +**

**groundplane[2] \* lightpos[2] +**

**groundplane[3] \* lightpos[3];**

**for (int i = 0; i < 4; ++i) {**

**shadowMat[i][0] = dot - lightpos[0] \* groundplane[i];**

**shadowMat[i][1] = 0.0f - lightpos[1] \* groundplane[i];**

**shadowMat[i][2] = 0.0f - lightpos[2] \* groundplane[i];**

**shadowMat[i][3] = 0.0f - lightpos[3] \* groundplane[i];**

**}**

**shadowMat[3][3] = dot - lightpos[3] \* groundplane[3];**

**}**

**// Function to draw Bezier curve**

**void drawBezierCurve(std::vector<GLfloat> controlPoints) {**

**glBegin(GL\_LINE\_STRIP);**

**for (GLfloat t = 0; t <= 1; t += 0.01) {**

**GLfloat x = (1 - t) \* (1 - t) \* controlPoints[0] +**

**2 \* (1 - t) \* t \* controlPoints[2] +**

**t \* t \* controlPoints[4];**

**GLfloat y = (1 - t) \* (1 - t) \* controlPoints[1] +**

**2 \* (1 - t) \* t \* controlPoints[3] +**

**t \* t \* controlPoints[5];**

**glVertex3f(x, y + 1.0f, 0.0); // Adjust y position to be above ground**

**}**

**glEnd();**

**}**

**// Function to draw Catmull-Rom spline**

**void drawCatmullRomSpline(std::vector<GLfloat> points) {**

**glBegin(GL\_LINE\_STRIP);**

**for (GLfloat t = 0; t < points.size() / 2 - 1; t += 0.01) {**

**int p0 = static\_cast<int>(t);**

**int p1 = p0 + 1;**

**int p2 = (p1 + 1) % (points.size() / 2);**

**int p3 = (p2 + 1) % (points.size() / 2);**

**GLfloat dt = t - p0;**

**GLfloat x = 0.5f \* ((2 \* points[p1 \* 2] +**

**(-points[p0 \* 2] + points[p2 \* 2]) \* dt +**

**(2 \* points[p0 \* 2] - 5 \* points[p1 \* 2] + 4 \* points[p2 \* 2] - points[p3 \* 2]) \* dt \* dt +**

**(-points[p0 \* 2] + 3 \* points[p1 \* 2] - 3 \* points[p2 \* 2] + points[p3 \* 2]) \* dt \* dt \* dt));**

**GLfloat y = 0.5f \* ((2 \* points[p1 \* 2 + 1] +**

**(-points[p0 \* 2 + 1] + points[p2 \* 2 + 1]) \* dt +**

**(2 \* points[p0 \* 2 + 1] - 5 \* points[p1 \* 2 + 1] + 4 \* points[p2 \* 2 + 1] - points[p3 \* 2 + 1]) \* dt \* dt +**

**(-points[p0 \* 2 + 1] + 3 \* points[p1 \* 2 + 1] - 3 \* points[p2 \* 2 + 1] + points[p3 \* 2 + 1]) \* dt \* dt \* dt));**

**glVertex3f(x, y + 1.0f, 0.0); // Adjust y position to be above ground**

**}**

**glEnd();**

**}**

**// Function to draw the scene (with shadows)**

**void display() {**

**glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // Clear color and depth buffers**

**// Set up lighting**

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

**// Set the view perspective**

**glMatrixMode(GL\_MODELVIEW);**

**glLoadIdentity();**

**gluLookAt(5.0, 5.0, 10.0, // Camera position**

**0.0, 0.0, 0.0, // Look at point**

**0.0, 1.0, 0.0); // Up vector**

**// Ground plane**

**glColor3f(0.5f, 0.5f, 0.5f); // Gray color for the ground**

**glBegin(GL\_QUADS);**

**glVertex3f(-5.0f, -1.0f, -5.0f);**

**glVertex3f(5.0f, -1.0f, -5.0f);**

**glVertex3f(5.0f, -1.0f, 5.0f);**

**glVertex3f(-5.0f, -1.0f, 5.0f);**

**glEnd();**

**// Define ground plane equation (y = -1)**

**GLfloat groundPlane[] = {0.0f, 1.0f, 0.0f, 1.0f}; // Plane equation**

**// Calculate shadow matrix**

**GLfloat shadowMat[4][4];**

**shadowMatrix(shadowMat, groundPlane, lightPos);**

**// Render the sphere (normal render)**

**glColor3f(1.0f, 0.0f, 1.0f); // Magenta color for the sphere**

**glPushMatrix();**

**glTranslatef(0.0f, 0.0f, 0.0f); // Position the sphere**

**glutSolidSphere(1.0, 50, 50); // Create a solid sphere with radius 1**

**glPopMatrix();**

**// Apply shadow matrix and render shadow**

**glDisable(GL\_LIGHTING); // Disable lighting for the shadow**

**glColor4f(1.0f, 1.0f, 1.0f, 0.75f); // White semi-transparent shadow color**

**glPushMatrix();**

**glMultMatrixf((GLfloat \*)shadowMat); // Multiply the current matrix with the shadow matrix**

**glutSolidSphere(1.0, 50, 50); // Create the shadow as a flattened sphere**

**glPopMatrix();**

**glEnable(GL\_LIGHTING); // Re-enable lighting**

**// Draw curves**

**std::vector<GLfloat> bezierPoints = { -2.0f, 0.0f, -1.0f, 2.0f, 1.0f, 0.0f };**

**glColor3f(0.0f, 1.0f, 0.0f); // Green for Bezier curve**

**drawBezierCurve(bezierPoints);**

**std::vector<GLfloat> splinePoints = { 2.0f, -1.0f, 1.0f, 1.0f, 0.0f, -1.0f, -1.0f, 0.5f };**

**glColor3f(0.0f, 0.0f, 1.0f); // Blue for Catmull-Rom spline**

**drawCatmullRomSpline(splinePoints);**

**glutSwapBuffers(); // Swap buffers to display the result**

**}**

**// Reshape function to handle window resizing**

**void reshape(int width, int height) {**

**if (height == 0) height = 1;**

**float aspect = (float)width / (float)height;**

**glViewport(0, 0, width, height);**

**glMatrixMode(GL\_PROJECTION);**

**glLoadIdentity();**

**gluPerspective(45.0, aspect, 0.1, 100.0);**

**}**

**// Main function to set up the GLUT environment and run the application**

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

**glutInit(&argc, argv);**

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

**glutInitWindowSize(800, 600);**

**glutCreateWindow("Shadow Mapping with Curves");**

**init();**

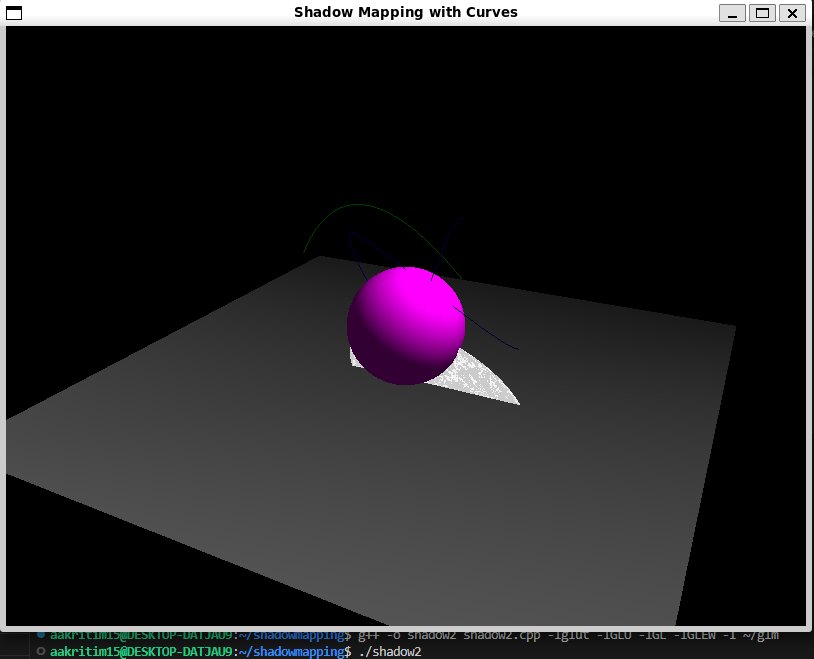
**glutDisplayFunc(display);**

**glutReshapeFunc(reshape);**

**glutMainLoop(); // Enter the event-processing loop**

**return 0;**

**}**

****