**Batch: D - 1 Roll No.: 16010122096**

**Experiment No. 08**

|  |
| --- |
| **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:**

**CO4: 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:**

**function Initialize() {**

**Create Window**

**Initialize OpenGL**

**Create FBO and depth texture**

**Set light properties**

**}**

**function RenderDepthMap() {**

**Bind FBO**

**Set viewport to depth texture size**

**Clear depth buffer**

**Set camera to light's perspective**

**Render scene objects**

**}**

**function RenderScene() {**

**Bind default framebuffer**

**Set viewport to window size**

**Clear color and depth buffers**

**Set camera to viewer's perspective**

**Render scene objects**

**For each object in scene:**

**Calculate object's position in light space**

**Compare depth with shadow map**

**If in shadow:**

**Apply shadow color**

**Else:**

**Apply light color**

**}**

**function Cleanup() {**

**Delete FBO and depth texture**

**}**

**main() {**

**Initialize()**

**while (window is open) {**

**RenderDepthMap()**

**RenderScene()**

**}**

**Cleanup()**

**}**

**Implementation details:**

*#include* <GL/glut.h>

*#include* <GL/gl.h>

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

void init() {

    glClearColor(0.53f, 0.81f, 0.98f, 1.0f);

    glEnable(GL\_DEPTH\_TEST);

    glShadeModel(GL\_SMOOTH);

    glEnable(GL\_LIGHTING);

    glEnable(GL\_LIGHT0);

    glEnable(GL\_COLOR\_MATERIAL);

    glEnable(GL\_BLEND);

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

}

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];

}

void display() {

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

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

    glMatrixMode(GL\_MODELVIEW);

    glLoadIdentity();

    gluLookAt(5.0, 5.0, 10.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

    glColor3f(0.7f, 0.9f, 0.7f);

    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();

    GLfloat groundPlane[] = {0.0f, 1.0f, 0.0f, 1.0f};

    GLfloat shadowMat[4][4];

    shadowMatrix(shadowMat, groundPlane, lightPos);

    glColor3f(1.0f, 1.0f, 0.0f);

    glPushMatrix();

    glTranslatef(0.0f, 0.0f, 0.0f);

    glutSolidSphere(1.0, 50, 50);

    glPopMatrix();

    glDisable(GL\_LIGHTING);

    glColor4f(0.0f, 0.0f, 0.0f, 0.75f);

    glPushMatrix();

    glMultMatrixf((GLfloat \*)shadowMat);

    glTranslatef(0.0f, 0.0f, 0.0f);

    glutSolidSphere(1.0, 50, 50);

    glPopMatrix();

    glEnable(GL\_LIGHTING);

    glutSwapBuffers();

}

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);

}

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

    glutInit(&argc, argv);

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

    glutInitWindowSize(800, 600);

    glutCreateWindow("Shadow Mapping");

    init();

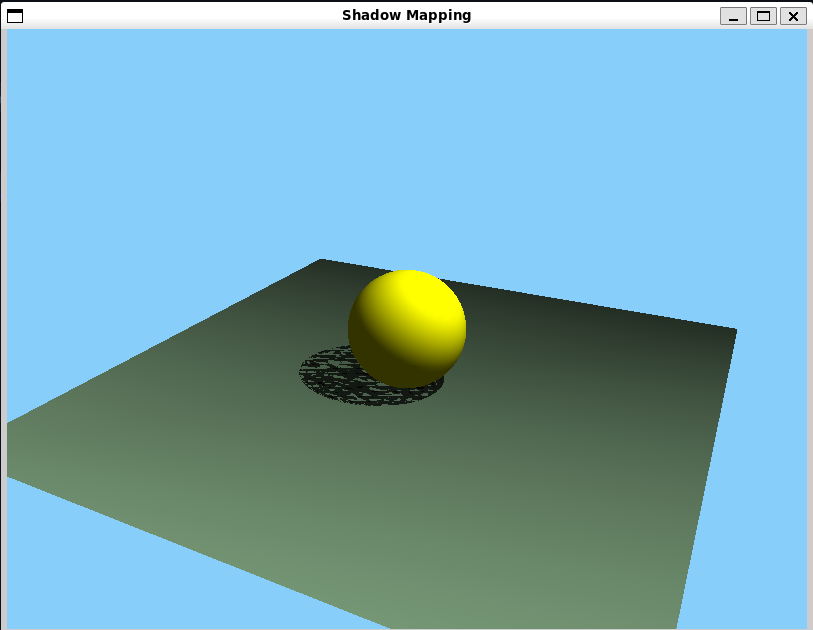
    glutDisplayFunc(display);

    glutReshapeFunc(reshape);

    glutMainLoop();

*return* 0;

}

**Output:  
**

**Screenshots from VLab(if any):**

**Conclusion and discussion:**

The implementation of various curves, including Bézier, B-spline, and Catmull-Rom, showcases the versatility of OpenGL in rendering complex shapes. This approach enhances visual representation in graphics applications, providing essential tools for designers and developers to create smooth transitions and intricate designs efficiently.

**Date: 30 / 09 / 2024 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>

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

void init() {

    glClearColor(0.0, 0.0, 0.0, 1.0);

    glEnable(GL\_DEPTH\_TEST);

    glShadeModel(GL\_SMOOTH);

    glEnable(GL\_LIGHTING);

    glEnable(GL\_LIGHT0);

    glEnable(GL\_COLOR\_MATERIAL);

    glEnable(GL\_BLEND);

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

}

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];

}

void display() {

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

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

    glMatrixMode(GL\_MODELVIEW);

    glLoadIdentity();

    gluLookAt(5.0, 5.0, 10.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

    glColor3f(0.5f, 0.5f, 0.5f);

    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();

    GLfloat groundPlane[] = {0.0f, 1.0f, 0.0f, 1.0f};

    GLfloat shadowMat[4][4];

    shadowMatrix(shadowMat, groundPlane, lightPos);

    glColor3f(0.0f, 0.0f, 1.0f); *// Blue color for the teapot*

    glPushMatrix();

    glTranslatef(-2.0f, 0.0f, 0.0f);

    glutSolidTeapot(1.0); *// Create a solid teapot with size 1*

    glPopMatrix();

    glDisable(GL\_LIGHTING);

    glColor4f(0.0f, 0.0f, 0.0f, 0.75f);

    glPushMatrix();

    glMultMatrixf((GLfloat \*)shadowMat);

    glTranslatef(-2.0f, 0.0f, 0.0f);

    glutSolidTeapot(1.0); *// Create the shadow as a flattened teapot*

    glPopMatrix();

    glEnable(GL\_LIGHTING);

    glColor3f(1.0f, 0.5f, 0.0f); *// Orange color for the cone*

    glPushMatrix();

    glTranslatef(2.0f, 0.0f, 0.0f);

    glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);

    glutSolidCone(1.0, 2.0, 50, 50); *// Create a solid cone with base radius 1 and height 2*

    glPopMatrix();

    glDisable(GL\_LIGHTING);

    glColor4f(0.0f, 0.0f, 0.0f, 0.75f);

    glPushMatrix();

    glMultMatrixf((GLfloat \*)shadowMat);

    glTranslatef(2.0f, 0.0f, 0.0f);

    glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);

    glutSolidCone(1.0, 2.0, 50, 50); *// Create the shadow as a flattened cone*

    glPopMatrix();

    glEnable(GL\_LIGHTING);

    glutSwapBuffers();

}

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);

}

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

    glutInit(&argc, argv);

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

    glutInitWindowSize(800, 600);

    glutCreateWindow("Shadow Mapping - Teapot and Cone");

    init();

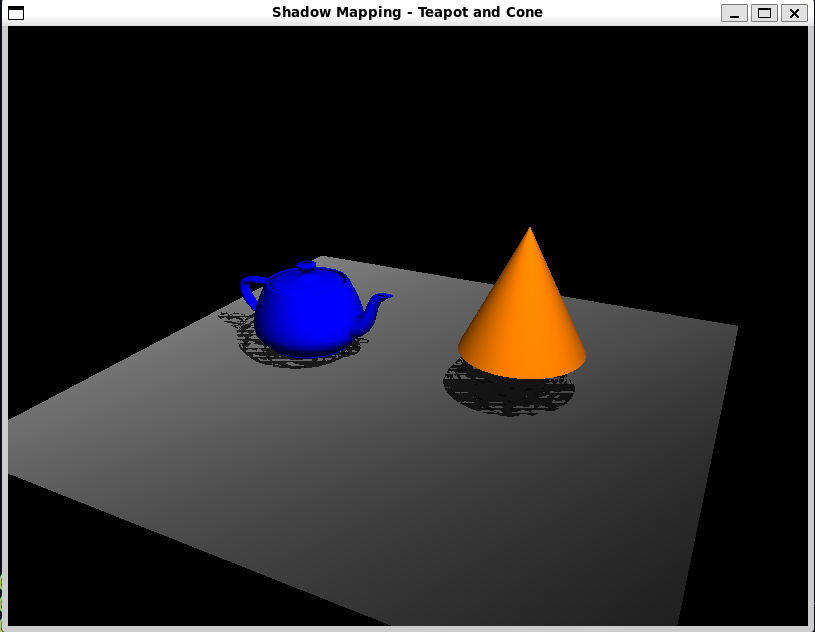
    glutDisplayFunc(display);

    glutReshapeFunc(reshape);

    glutMainLoop();

*return* 0;

}

**Output:  
**

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

*#include* <GL/glut.h>

*#include* <cmath>

void drawBezierCurve() {

    glBegin(GL\_LINE\_STRIP);

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

        float x = pow(1 - t, 3) \* -3 + 3 \* pow(1 - t, 2) \* t \* 1 + 3 \* (1 - t) \* pow(t, 2) \* 2 + pow(t, 3) \* 3;

        float y = pow(1 - t, 3) \* 3 + 3 \* pow(1 - t, 2) \* t \* 2 + 3 \* (1 - t) \* pow(t, 2) \* 1 + pow(t, 3) \* 1;

        glVertex2f(x, y);

    }

    glEnd();

}

void drawBSplineCurve() {

    glBegin(GL\_LINE\_STRIP);

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

        float x = pow(1 - t, 3) \* -1 + 3 \* pow(1 - t, 2) \* t \* 0 + 3 \* (1 - t) \* pow(t, 2) \* 1 + pow(t, 3) \* 2;

        float y = pow(1 - t, 3) \* 1 + 3 \* pow(1 - t, 2) \* t \* 1 + 3 \* (1 - t) \* pow(t, 2) \* 1 + pow(t, 3) \* 0;

        glVertex2f(x, y);

    }

    glEnd();

}

void drawCatmullRomSpline(float *p0*[2], float *p1*[2], float *p2*[2], float *p3*[2]) {

    glBegin(GL\_LINE\_STRIP);

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

        float x = 0.5 \* ((2 \* p1[0]) +

                         (-p0[0] + p2[0]) \* t +

                         (2 \* p0[0] - 5 \* p1[0] + 4 \* p2[0] - p3[0]) \* t \* t +

                         (-p0[0] + 3 \* p1[0] - 3 \* p2[0] + p3[0]) \* t \* t \* t);

        float y = 0.5 \* ((2 \* p1[1]) +

                         (-p0[1] + p2[1]) \* t +

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

                         (-p0[1] + 3 \* p1[1] - 3 \* p2[1] + p3[1]) \* t \* t \* t);

        glVertex2f(x, y);

    }

    glEnd();

}

void drawString(float *x*, float *y*, const char\* *string*, float *r*, float *g*, float *b*) {

    glColor3f(r, g, b);

    glRasterPos2f(x, y);

*while* (\*string) {

        glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, \*string);

        string++;

    }

}

void display() {

    glClear(GL\_COLOR\_BUFFER\_BIT);

    glColor3f(1.0f, 0.0f, 0.0f); *// Red for Bézier curve*

    drawBezierCurve();

    drawString(-2.5f, 2.5f, "Bézier Curve", 1.0f, 0.0f, 0.0f);

    glColor3f(0.0f, 1.0f, 0.0f); *// Green for B-spline curve*

    drawBSplineCurve();

    drawString(-2.0f, 1.5f, "B-spline Curve", 0.0f, 1.0f, 0.0f);

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

    float p0[2] = { -2.0f, 0.0f };

    float p1[2] = { -1.0f, 2.0f };

    float p2[2] = { 1.0f, 2.0f };

    float p3[2] = { 2.0f, 0.0f };

    drawCatmullRomSpline(p0, p1, p2, p3);

    drawString(-1.0f, 2.5f, "Catmull-Rom Spline", 0.0f, 0.0f, 1.0f);

    glFlush();

}

void init() {

    glClearColor(1.0, 1.0, 1.0, 1.0);

    glMatrixMode(GL\_PROJECTION);

    gluOrtho2D(-3.0, 3.0, -1.0, 3.0);

}

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

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowSize(800, 600);

    glutCreateWindow("Curves: Bézier, B-spline, Catmull-Rom");

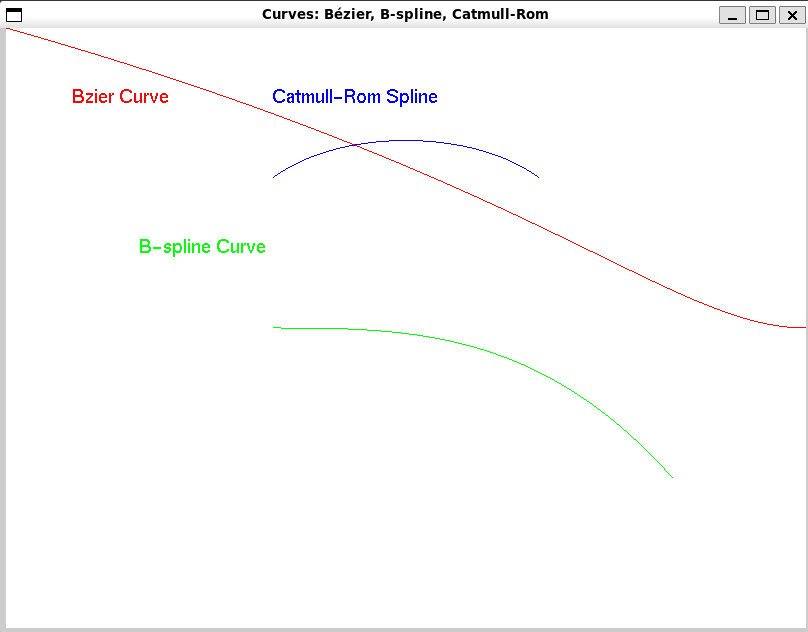
    init();

    glutDisplayFunc(display);

    glutMainLoop();

*return* 0;

}

**Output:  
**