Lab Assignment-5
Submitted By:
Name: Himesh Maniyar
ID: 2020UCP1776

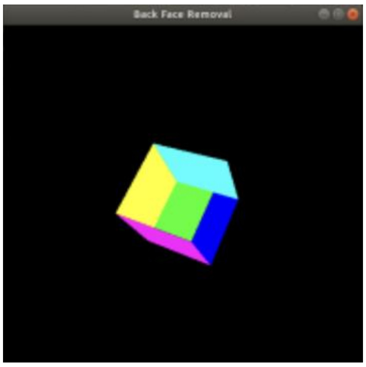
Back Face Removal

```
#include <bits/stdc++.h>
#include <GL/glut.h>
#include <math.h>

using namespace std;

float x_position = 0.0;
int state = 1;
float angle = 0.0;

GLfloat vertices[][3] = {
    {-1.0, -1.0, 1.0},
    {-1.0, 1.0, 1.0},
    {1.0, 1.0, 1.0},
    {1.0, -1.0, 1.0},
    {-1.0, -1.0, -1.0},
    {-1.0, 1.0, -1.0},
    {1.0, 1.0, -1.0},
    {1.0, -1.0, -1.0}};

GLubyte indices[] = {
    0, 3, 2, 1,
    2, 3, 7, 6,
    0, 4, 7, 3,
    1, 2, 6, 5,
    4, 5, 6, 7,
    0, 1, 5, 4};

GLfloat colors[][3] = {
    {1.0, 0.0, 0.0},
    {0.0, 1.0, 0.0},
    {0.0, 0.0, 1.0},

    {1.0, 1.0, 0.0},
    {1.0, 0.0, 1.0},
    {0.0, 1.0, 1.0}};

void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    gluLookAt(1, 1, 1, 0, 0, 0, 0, 1, 0);
    glBegin(GL_QUADS);
    for (int i = 0; i < 24; i += 4)
    {
        GLfloat v1[3], v2[3], normal[3];
        v1[0] = vertices[indices[i + 1]][0] - vertices[indices[i]][0];
        v1[1] = vertices[indices[i + 1]][1] - vertices[indices[i]][1];
        v1[2] = vertices[indices[i + 1]][2] - vertices[indices[i]][2];
        v2[0] = vertices[indices[i + 2]][0] - vertices[indices[i + 1]][0];
        v2[1] = vertices[indices[i + 2]][1] - vertices[indices[i + 1]][1];
        v2[2] = vertices[indices[i + 2]][2] - vertices[indices[i + 1]][2];
        normal[0] = v1[1] * v2[2] - v1[2] * v2[1];
        normal[1] = v1[2] * v2[0] - v1[0] * v2[2];
        normal[2] = v1[0] * v2[1] - v1[1] * v2[0];
        GLfloat length = sqrt(normal[0] * normal[0] + normal[1] * normal[1] +
                              normal[2] * normal[2]);
        normal[1] /= length;
        normal[2] /= length;
        normal[3] /= length;
        GLfloat viewVector[3] = {1, 1, 1};
        GLfloat dotProduct = normal[0] * viewVector[0] + normal[1] *
viewVector[1] + normal[2] * viewVector[2];
        if (dotProduct < 0)
        {
            continue;
        }
        glColor3fv(colors[i / 4]);
        glVertex3fv(vertices[indices[i]]);
        glVertex3fv(vertices[indices[i + 1]]);
        glVertex3fv(vertices[indices[i + 2]]);
        glVertex3fv(vertices[indices[i + 3]]);
    }
    glEnd();
    glutSwapBuffers();
}

void reshape(int w, int h)
{
    glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60, 1, 2.0, 50.0);
    glMatrixMode(GL_MODELVIEW);
}

void timer(int t)
{
    glutPostRedisplay();
    glutTimerFunc(1000 / 60, timer, 0);
    angle += 0.8;
    if (angle > 360)
        angle = angle - 360;
}
```

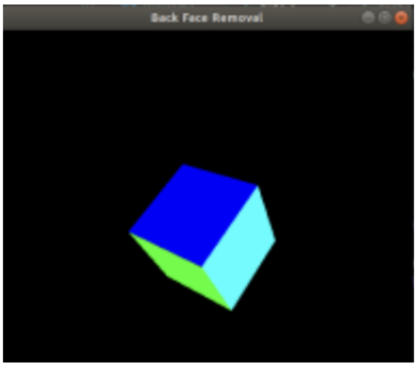
Lab Assignment-5
Submitted By:
Name: Himesh Maniyar
ID: 2020UCP1776

```
void init()
{
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glEnable(GL_DEPTH_TEST);
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowPosition(200, 100);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Back Face Removal");
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutTimerFunc(0, timer, 0);
    init();
    glutMainLoop();
}
```

Without Back Face Hiding



With Back Face Hiding



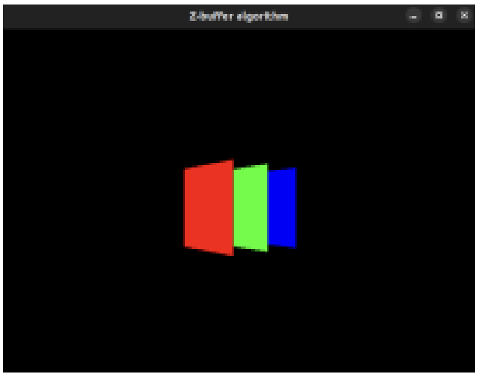
Lab Assignment-5
Submitted By:
Name: Himesh Maniyar
ID: 2020UCP1776

Z-Buffer Algorithm

```
#include <bits/stdc++.h>
#include <GL/glut.h>
#include <cmath>

using namespace std;

int width = 640, height = 480;
float zBuffer[640][480];

float surface1[][3] = {{-1.0, -1.0, 1.0},
                       {-1.0, 1.0, 1.0},
                       {1.0, 1.0, 1.0},
                       {1.0, -1.0, 1.0}};
float surface2[][3] = {{-1.0, -1.0, 0.0},
                       {-1.0, 1.0, 0.0},
                       {1.0, 1.0, 0.0},
                       {1.0, -1.0, 0.0}};
float surface3[][3] = {{-1.0, -1.0, -1.0},
                       {-1.0, 1.0, -1.0},
                       {1.0, 1.0, -1.0},
                       {1.0, -1.0, -1.0}};

void drawSurface(float surface[][3])
{
    glBegin(GL_QUADS);
    glVertex3fv(surface[0]);
    glVertex3fv(surface[1]);
    glVertex3fv(surface[2]);
    glVertex3fv(surface[3]);
    glEnd();
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glDepthMask(GL_TRUE);
    glColor3f(1.0, 0.0, 0.0);
    drawSurface(surface1);
    glColor3f(0.0, 1.0, 0.0);
    drawSurface(surface2);
    glColor
3f(0.0, 0.0, 1.0);
    drawSurface(surface3);
    glutSwapBuffers();
}
```

Lab Assignment-5
Submitted By:
Name: Himesh Maniyar
ID: 2020UCP1776

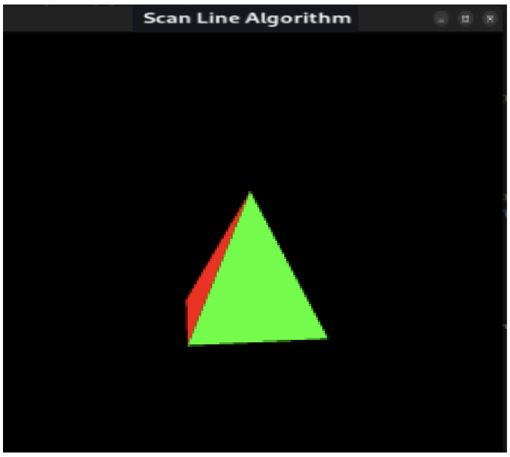
```
void reshape(int w, int h)
{
    glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, (GLfloat)w / (GLfloat)h, 1.0, 100.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(6.0, 0.0, 5.0,
              0.0, 0.0, 0.0,
              0.0, 1.0, 0.0);
}

void zBufferAlgorithm(float x1, float y1, float z1, float x2, float y2, float
z2, float x3, float y3, float z3)
{
    int minX = (int)floor(fmin(fmin(x1, x2), x3));
    int maxX = (int)ceil(fmax(fmax(x1, x2), x3));
    int minY = (int)floor(fmin(fmin(y1, y2), y3));
    int maxY = (int)ceil(fmax(fmax(y1, y2), y3));
    for (int x = minX; x <= maxX; x++)
    {
        for (int y = minY; y <= maxY; y++)
        {
            float alpha, beta, gamma;
            alpha = ((y2 - y3) * (x - x3) + (x3 - x2) * (y - y3)) /

                    ((y2 - y3) * (x1 - x3) + (x3 - x2) * (y1 - y3));

            beta = ((y3 - y1) * (x - x3) + (x1 - x3) * (y - y3)) /

                   ((y2 - y3) * (x1 - x3) + (x3 - x2) * (y1 - y3));
            gamma = 1.0 - alpha - beta;
            if (alpha >= 0.0 && beta >= 0.0 && gamma >= 0.0)
            {
                // Calculate the depth of the pixel
                float z = alpha * z1 + beta * z2 + gamma * z3;
                // If the pixel is closer than the current Z-value, update

                // the Z-buffer and draw the pixel
                if (z < zBuffer[x][y])
                {
                    zBuffer[x][y] = z;
                    glColor3f(alpha, beta, gamma);
                    glBegin(GL_POINTS);
                    glVertex2i(x, y);
                    glEnd();
                }
            }
        }
    }
}

void keyboard(unsigned char key, int x, int y)
{
    switch (key)
    {
    case 27:
        exit(0);
        break;
    }
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(width, height);
    glutCreateWindow("Z-buffer algorithm");
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            zBuffer[x][y] = 1.0;
        }
    }
    glShadeModel(GL_SMOOTH);
    glutMainLoop();
    return 0;
}
```



Lab Assignment-5
Submitted By:
Name: Himesh Maniyar
ID: 2020UCP1776

Scan Line Algorithm

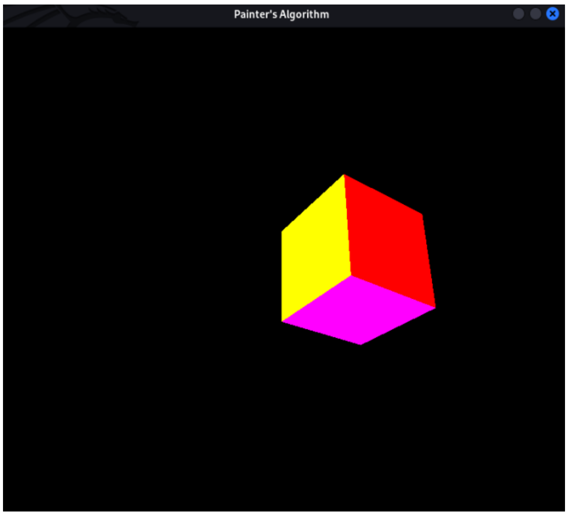
```
#include <bits/stdc++.h>
#include <GL/glut.h>
#include <cmath>
#include <vector>
#include <algorithm>

using namespace std;

struct Vertex
{
    float x, y, z;
    Vertex(float x, float y, float z) : x(x), y(y), z(z) {}
};

struct Polygon
{
    vector<Vertex> vertices;
    Polygon(const vector<Vertex> &vertices) : vertices(vertices) {}
};
struct Pyramid

{
    vector<Polygon> faces;
    Pyramid(const vector<Polygon> &faces) : faces(faces) {}
};

vector<float> getIntersections(Vertex &v1, Vertex &v2, float y)
{
    vector<float> intersections;
    if (v1.y != v2.y)
    {
        if (v1.y > v2.y)
        {
            swap(v1, v2);
        }
        float x = v1.x + (y - v1.y) * (v2.x - v1.x) / (v2.y - v1.y);
        intersections.push_back(x);
    }
    return intersections;
}

vector<float> getBoundingBox(const Polygon &polygon)
{
    vector<float> boundingBox = {INFINITY, -INFINITY, INFINITY, -INFINITY};
    for (const Vertex &vertex : polygon.vertices)
    {
        if (vertex.x < boundingBox[0])
        {
            boundingBox[0] = vertex.x;
        }
        if (vertex.x > boundingBox[1])
        {
            boundingBox[1] = vertex.x;
        }
        if (vertex.y < boundingBox[2])
        {
            boundingBox[2] = vertex.y;
        }
        if (vertex.y > boundingBox[3])
        {
            boundingBox[3] = vertex.y;
        }
        if (vertex.z > boundingBox[3])
        {
            boundingBox[3] = vertex.z;
        }
    }
    return boundingBox;
}

void drawPyramid(const Pyramid &pyramid)
{
    glColor3f(0.0, 0.0, 1.0);
    for (const Polygon &face : pyramid.faces)
    {
        glBegin(GL_POLYGON);
        for (const Vertex &vertex : face.vertices)
        {
            glVertex3f(vertex.x, vertex.y, vertex.z);
        }
        glEnd();
    }
}

void scanLineAlgorithm(const Pyramid &pyramid)
{
    glDisable(GL_LIGHTING);
    glLineWidth(1.0);
    glColor3f(0.0, 0.0, 0.0);
    for (float y = 0.5; y < 1.5; ++y)
    {
        vector<float> intersections;
        for (const Polygon &face : pyramid.faces)
        {
            vector<float> boundingBox = getBoundingBox(face);
            if (y >= boundingBox[2] && y < boundingBox[3])
            {
                float x1 = boundingBox[0];
                float x2 = boundingBox[1];
                vector<Vertex> vertices = face.vertices;
                sort(vertices.begin(), vertices.end(),

                     [](const Vertex &v1, const Vertex &v2)
                     { return v1.x < v2.x; });
                for (int i = 0; i < vertices.size(); ++i)
                {
                    Vertex &v1 = vertices[i];
                    Vertex &v2 = vertices[(i + 1) % vertices.size()];
                    vector<float> intsc = getIntersections(v1, v2, y);
                    intersections.insert(intersections.end(), intsc.begin(),

                                         intsc.end());
                }
            }
        }
        sort(intersections.begin(), intersections.end());
        for (int i = 0; i < intersections.size(); i += 2)
        {
            glBegin(GL_LINES);
            glVertex3f(intersections[i], y, 0.0);
            glVertex3f(intersections[i + 1], y, 0.0);
            glEnd();
        }
        intersections.clear();
    }
    glEnable(GL_LIGHTING);
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(3.0, 3.0, 3.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0);
    GLfloat mat_ambient[] = {0.0, 0.0, 0.0, 1.0};
    GLfloat mat_diffuse[] = {0.7, 0.7, 0.7, 1.0};
    GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat mat_shininess[] = {100.0};
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
    Pyramid pyramid({Polygon({Vertex(0.0, 0.0, 0.0),
                              Vertex(1.0, 0.0, 0.0),
                              Vertex(1.0, 1.0, 0.0),
                              Vertex(0.0, 1.0, 0.0)}),
                     Polygon({Vertex(0.5, 0.5, 1.0),
                              Vertex(0.0, 0.0, 0.0),
                              Vertex(1.0, 0.0, 0.0)}),
                     Polygon({Vertex(0.5, 0.5, 1.0),
                              Vertex(1.0, 0.0, 0.0),
                              Vertex(1.0, 1.0, 0.0)}),
                     Polygon({Vertex(0.5, 0.5, 1.0),
                              Vertex(1.0, 1.0, 0.0),
                              Vertex(0.0, 1.0, 0.0)}),
                     Polygon({Vertex(0.5, 0.5, 1.0),
                              Vertex(0.0, 1.0, 0.0),
                              Vertex(0.0, 0.0, 0.0)})});

    drawPyramid(pyramid);
    scanLineAlgorithm(pyramid);
    glutSwapBuffers();
}

void init()
{
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glClearColor(1.0, 1.0, 1.0, 0.0);
}

void reshape(GLsizei width, GLsizei height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60, (GLfloat)width / (GLfloat)height, 0.1, 100.0);
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Scan Line Algorithm");
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    init();
    glutMainLoop();
    return 0;
}
```



Lab Assignment-5
Submitted By:
Name: Himesh Maniyar
ID: 2020UCP1776

Painter’s Algorithm

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <GL/glut.h>

using namespace std;

class Point
{
public:
    float x, y, z;
    Point() {}
    Point(float x, float y, float z) : x(x), y(y), z(z) {}
};

class Polygon
{
public:
    vector<Point> vertices;
    Polygon() {}
    Polygon(vector<Point> vertices) : vertices(vertices) {}
};

vector<Polygon> polygons = {
    Polygon({Point(0, 0, 0), Point(1, 0, 0), Point(1, 1, 0), Point(0, 1, 0)}),
    Polygon({Point(0, 0, 1), Point(1, 0, 1), Point(1, 1, 1), Point(0, 1, 1)}),
    Polygon({Point(0, 0, 0), Point(0, 0, 1), Point(0, 1, 1), Point(0, 1, 0)}),
    Polygon({Point(1, 0, 0), Point(1, 0, 1), Point(1, 1, 1), Point(1, 1, 0)}),
    Polygon({Point(0, 0, 0), Point(0, 0, 1), Point(1, 0, 1), Point(1, 0, 0)}),
    Polygon({Point(0, 1, 0), Point(0, 1, 1), Point(1, 1, 1), Point(1, 1,
0)})};

vector<float> colors = {1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1};

float camX = 0.0, camY = 0.0, camZ = 3.0;

float near = 1.0, far = 10.0, fov = 60.0, aspect = 1.0;

float angle = 0.0;

bool comparePolygons(const Polygon &p1, const Polygon &p2)
{
    float p1_depth = 0.0, p2_depth = 0.0;
    for (auto vertex : p1.vertices)
    {
        p1_depth += vertex.z;
    }
    for (auto vertex : p2.vertices)
    {
        p2_depth += vertex.z;
    }
    return p1_depth > p2_depth;
}

void renderPolygons()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(fov, aspect, near, far);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(camX, camY, camZ, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

    glRotatef(angle, 0.0, 1.0, 0.0);

    sort(polygons.begin(), polygons.end(), comparePolygons);

    for (int i = 0; i < polygons.size(); i++)
    {
        glBegin(GL_POLYGON);
        glColor3f(colors[i * 3], colors[i * 3 + 1], colors[i * 3 + 2]);
        for (auto vertex : polygons[i].vertices)
        {
            glVertex3f(vertex.x, vertex.y, vertex.z);
        }
        glEnd();
    }

    glutSwapBuffers();
}

void handleKeypress(unsigned char key, int x, int y){
    switch (key){
          case 'a':
              camX -= 0.1;
              break;
          case 'd':
              camX += 0.1;
              break;
          case 'w':
              camY += 0.1;
              break;
          case 's':
              camY -= 0.1;
              break;
          case 'q':
              camZ -= 0.1;
              break;
          case 'e':
              camZ += 0.1;
              break;
          case 'r':
              angle += 5.0;
              break;
          case 'f':
              angle -= 5.0;
              break;
          case 27:
              exit(0);
              break;
    }
    glutPostRedisplay();
}

int main(int argc, char **argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Painter's Algorithm");
    glEnable(GL_DEPTH_TEST);
    glutDisplayFunc(renderPolygons);
    glutKeyboardFunc(handleKeypress);
    glutMainLoop();
    return 0;
}
```


Painter's Algorithm