COMPUTER GRAPHICS ASSIGNMENT

Submitted by,

Subi Pinsha P

2022115020

1) Draw first three letters of your name using OPENgl. Use both DDA and Bresenhams algorithm wherever needed.

```
#include <stdio.h>
#include <iostream>
#include <cmath>
#include <GL/glut.h>
#include <math.h>
#include <vector>

using namespace std;

int WINDOW_WIDTH = 720;
int WINDOW_HEIGHT = 480;

float sq_pos[] = { 0.0f, 0.0f };
float sq_rot = 0.0f;
float sq_scl = 1.0f;

const float pi = 3.14;

int X1, Y1, X2, Y2;

struct Point {
   float x, y;

   Point(float _x, float _y) : x(_x), y(_y) {}
};

vector<Point> pixels; // Vector to store pixel coordinates

void pixel(float u, float v) {
   // Add the pixel coordinates to the vector
   pixels.push_back(Point(u, v));
}
```

```
void calculateLineDDA(int x1, int y1, int x2, int y2) {

   // Calculate the slope (m)
   float m = static_cast<float>(y2 - y1) / (x2 - x1);

   // Initialize variables
   int dx = x2 - x1;
   int dy = y2 - y1;
   int steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy);

   // Calculate increments
   float xIncrement = static_cast<float>(dx) / steps;
   float yIncrement = static_cast<float>(dy) / steps;

   // Initialize current position
   float x = x1;
   float y = y1;

   // Add the starting point to the vector
   pixel(x, y);

   // Loop through the steps
   for (int i = 1; i <= steps; ++i) {
      x += xIncrement;
      y += yIncrement;

      // Add the pixel coordinates to the vector
      pixel(round(x), round(y));
   }
}

void calculateLineBresenham(int x1, int y1, int x2, int y2) {

   // Calculate the differences
   int dx = abs(x2 - x1);
   int dy = abs(y2 - y1);

   // Determine the sign of increments
   int signX = (x2 > x1) ? 1 : -1;
   int signY = (y2 > y1) ? 1 : -1;

   // Initialize decision parameters
   int decisionParameter = 2 * dy - dx;
   int x = x1, y = y1;

   // Add the starting point to the vector
   pixel(x, y);

   // Loop through the steps
   for (int i = 1; i <= dx; ++i) {
      if (decisionParameter > 0) {
```

```
            // Increment y and update decision parameter
            y += signY;
            decisionParameter += 2 * (dy - dx);
        }
        else {
            // Update decision parameter
            decisionParameter += 2 * dy;
        }

        // Increment x in each step
        x += signX;

        // Add the pixel coordinates to the vector
        pixel(x, y);
    }
}

void printPixels() {
    // Print the contents of the pixels vector
    cout << "Pixel Coordinates:\n";
    for (const auto& p : pixels) {
        cout << "(" << p.x << ", " << p.y << ")\n";
    }
}

void drawLineDDA() {
    glColor3f(1.0f, 1.0f, 1.0f);
    // Draw the pixels from the vector
    for (const auto& p : pixels) {
        glBegin(GL_QUADS);
        glVertex2f(p.x - 0.5f, p.y - 0.5f);
        glVertex2f(p.x + 0.5f, p.y - 0.5f);
        glVertex2f(p.x + 0.5f, p.y + 0.5f);
        glVertex2f(p.x - 0.5f, p.y + 0.5f);
        glEnd();
    }
}

void drawLine(int x1, int y1, int x2, int y2, float thickness) {
    // Set the line color
    glColor3f(0.0f, 1.0f, 0.0f);

    // Set the line thickness
    glLineWidth(thickness);

    // Draw the line
    glBegin(GL_LINES);
    glVertex2f(x1, y1);
    glVertex2f(x2, y2);
    glEnd();
```

```
    // Reset the line thickness to default
    glLineWidth(1.0f);
}

void square() {
    glBegin(GL_QUADS);
    glColor3f(0.0f, 1.0f, 0.0f); // green
    glVertex2f(-1.0f, -1.0f);
    glVertex2f(1.0f, -1.0f);
    glVertex2f(1.0f, 1.0f);
    glVertex2f(-1.0f, 1.0f);
    glEnd();
}

void axes() {
    // x axis
    glBegin(GL_LINES);
    glColor3f(1.0f, 0.0f, 0.0f); // red
    glVertex2f(-10.0f, 0.0f);
    glVertex2f(10.0f, 0.0f);
    glEnd();

    // y axis
    glBegin(GL_LINES);
    glColor3f(0.0f, 0.0f, 1.0f); // blue
    glVertex2f(0.0f, 10.0f);
    glVertex2f(0.0f, -10.0f);
    glEnd();
}

void cell(int u, int v) {
    glBegin(GL_LINE_LOOP);
    glVertex2f(u - 0.5f, v - 0.5f);
    glVertex2f(u + 0.5f, v - 0.5f);
    glVertex2f(u + 0.5f, v + 0.5f);
    glVertex2f(u - 0.5f, v + 0.5f);
    glEnd();
}

void grid() {
    int u = 0, v = 0, m = 0, n = 0;

    glPushMatrix();
    glColor3f(0.3f, 0.3f, 0.3f);

    for (int i = 0; i < 10; i++) {

        u = 0, v = 0;

        for (int j = 0; j < 10; j++) {
```

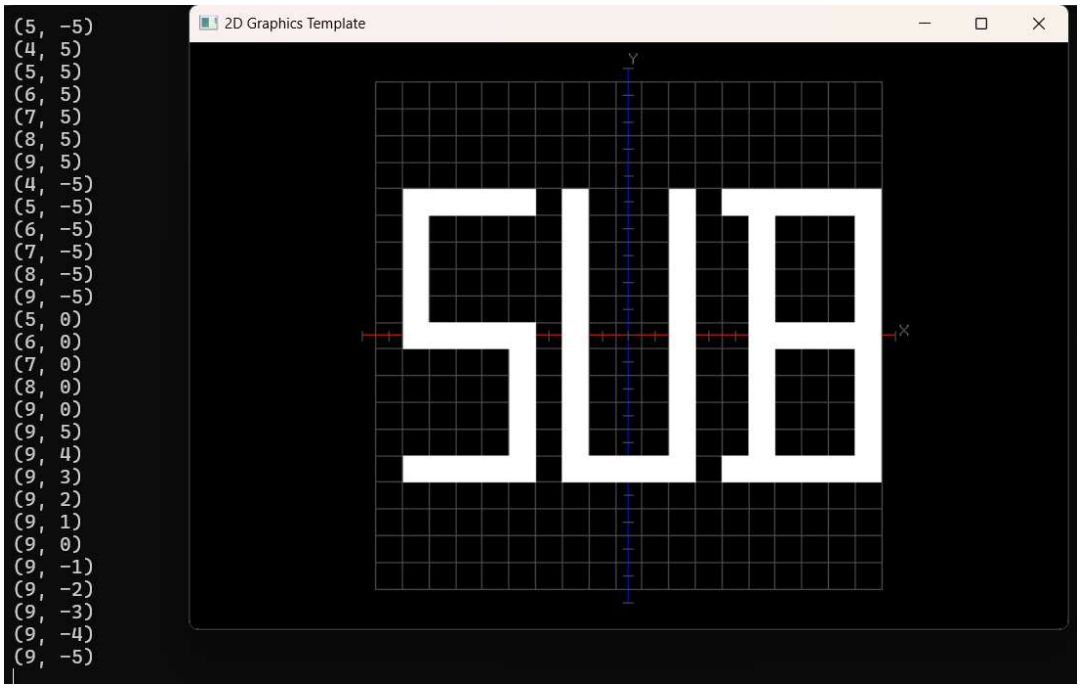
```
            cell(u, m);

            if (v != u) {
               cell(v, m);
            }

            if (m != n) {
               cell(u, n);

               if (v != u) {
                  cell(v, n);
               }
            }

            u++;
            v--;

         }

         m++;
         n--;

      }

      glPopMatrix();
   }

   void drawCartesianCoordinates() {
      // Set the line color
      glColor3f(0.3f, 0.3f, 0.3f);

      // Display tick marks on x-axis
      for (int i = -10; i <= 10; ++i) {
         glBegin(GL_LINES);
         glVertex2f(i, -0.2);
         glVertex2f(i, 0.2);
         glEnd();
      }

      // Display tick marks on y-axis
      for (int i = -10; i <= 10; ++i) {
         glBegin(GL_LINES);
         glVertex2f(-0.2, i);
         glVertex2f(0.2, i);
         glEnd();
      }

      // Display the axis labels
      glColor3f(0.5f, 0.5f, 0.5f);
      glRasterPos2f(10.2, 0.0);
      glutBitmapCharacter(GLUT_BITMAP_8_BY_13, 'X');
```

```
    glRasterPos2f(0.0, 10.2);
    glutBitmapCharacter(GLUT_BITMAP_8_BY_13, 'Y');
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    glPushMatrix();

    axes();

    // LOCAL >> T -> R -> S
    // GLOABAL >> S -> R -> T

    grid();
    drawCartesianCoordinates();

    drawLineDDA();
    drawLine(X1, Y1, X2, Y2, 4);

    glFlush();
}

void reshape(int w, int h) {
    glViewport(0, 0, w, h);

    GLfloat aspect_ratio = h == 0 ? w / 1 : (GLfloat)w / (GLfloat)h;
    // w = aspect_ratio * h
    // h = w / aspect_ratio

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    if (w <= h) {
        gluOrtho2D(-11, 11, -11 / aspect_ratio, 11 / aspect_ratio);
    }
    else {
        gluOrtho2D(-11 * aspect_ratio, 11 * aspect_ratio, -11, 11);
    }
}

void keyboardSpecial(int key, int x, int y) {
    //printf("%d\n", key);

    if (key == GLUT_KEY_UP) {
        sq_scl += 0.5f;
    }
    if (key == GLUT_KEY_DOWN) {
        sq_scl -= 0.5f;
    }
```

```
    glutPostRedisplay();
}

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

    // movement

    if (key == 'w') { // up
        sq_pos[1] += 0.5f;
    }
    if (key == 'a') { // left
        sq_pos[0] -= 0.5f;
    }
    if (key == 's') { // down
        sq_pos[1] -= 0.5f;
    }
    if (key == 'd') { // right
        sq_pos[0] += 0.5f;
    }

    glutPostRedisplay();

}

void init() {
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
}

void getUserInput() {
    cout << "Enter Coordinates To Draw Line\nUsing Bresenham Line Drawing Algorithm:\n\nNOTE:
(enter values from -10 to +10)\n\n";

    // Get user input for starting coordinates
    cout << "Enter x1: ";
    cin >> X1;
    cout << "Enter y1: ";
    cin >> Y1;

    // Get user input for ending coordinates
    cout << "Enter x2: ";
    cin >> X2;
    cout << "Enter y2: ";
    cin >> Y2;
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);

    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutInitWindowPosition(500, 300);
```

```
    glutCreateWindow("2D Graphics Template");

    //getUserInput();
       // Clear the pixel vector
    pixels.clear();

    //S
    calculateLineDDA(-8,0,-4,0);
    calculateLineDDA(-4,5,-8,5);
    calculateLineDDA(-4, -5, -8, -5);
    calculateLineDDA(-8, 0, -8, 5);
    calculateLineDDA(-4, 0, -4, -5);

    //U
    calculateLineDDA(-2, -5, -2, 5);
    calculateLineDDA(2, -5, 2, 5);
    calculateLineDDA(-2, -5, 2, -5);

    //B
    calculateLineDDA(5,5,5,-5);
    calculateLineDDA(4, 5, 9, 5);
    calculateLineDDA(4, -5, 9, -5);
    calculateLineDDA(5, 0, 9, 0);
    calculateLineDDA(9, 5, 9, -5);



    printPixels();

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);

    glutSpecialFunc(keyboardSpecial);
    glutKeyboardFunc(keyboard);


    init();
    glutMainLoop();

    return 0;
}
```

OUTPUT:

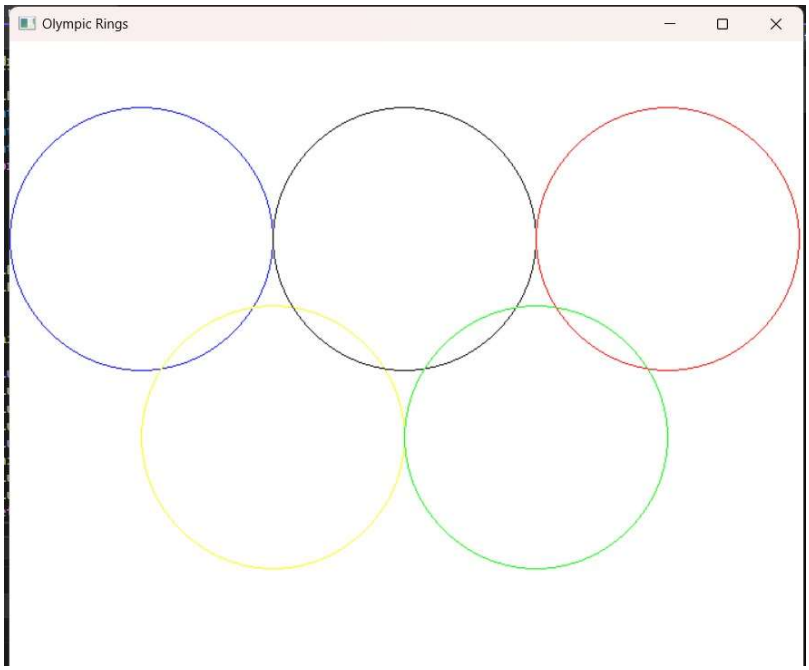


2.Draw the Olympics Logo using Mid point circle drawing algorithm and fill it with different pattern attributes.

OLYMPIC CODE:

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

constexpr int k_width = 700;
constexpr int k_height = 700;
constexpr int k_windowPosX = 100;
constexpr int k_windowPosY = 100;
constexpr double k_ringColors[5][3]{ {0.0, 0.0, 1.0}, {1.0, 1.0, 0.0}, {0.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {1.0, 0.0, 0.0} };

void drawInQuadrants(int centerX, int centerY, int x, int y)
{
    glVertex2i(centerX + x, centerY + y);
    glVertex2i(centerX - x, centerY + y);
    glVertex2i(centerX - x, centerY - y);
    glVertex2i(centerX + x, centerY - y);
}

void drawCircle(const int center[2], int radius, const double color[3])
{
    glColor3d(color[0], color[1], color[2]);
    int x = 0, y = radius, p = 1 - radius;
    while (x < y)
    {
        drawInQuadrants(center[0], center[1], x, y);
        drawInQuadrants(center[0], center[1], y, x);
        p += 2 * ++x + 1;
        if (p >= 0)
            p -= 2 * --y;
    }
}

void drawEllipse(const int center[2], int radiusX, int radiusY, const double color[3])
{
    glColor3d(color[0], color[1], color[2]);
    int rxSquared = radiusX * radiusX, rySquared = radiusY * radiusY;
    int rxSquaredTimesTwo = 2 * rxSquared, rySquaredTimesTwo = 2 * rySquared;
    int x = 0, y = radiusY;
    int p = static_cast<int>(std::round(rySquared - rxSquared * radiusY + 0.25 * rxSquared));
    while (rySquaredTimesTwo * x < rxSquaredTimesTwo * y)
    {
        drawInQuadrants(center[0], center[1], x, y);
        p += rySquaredTimesTwo * ++x + rySquared;
        if (p >= 0)
            p -= rxSquaredTimesTwo * --y;
    }
    p = static_cast<int>(std::round(rySquared * (x * x + x + 0.25) + rxSquared * (y * y - 2 * y + 1) - rxSquared * rySquared));
    while (y > 0)
    {
        drawInQuadrants(center[0], center[1], x, y);
        p += rxSquared - rxSquaredTimesTwo * --y;
        if (p <= 0)
            p += rySquaredTimesTwo * ++x;
    }
}

void init()
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, k_width, 0, k_height);
    glClear(GL_COLOR_BUFFER_BIT);
}

// Display functions:

void drawOlympicRingsAsCircles()
{
    glBegin(GL_POINTS);
    int radius = min(k_width, k_height) / 6;
    int center[2];
    for (int i = 0; i < 5; ++i)
    {
        // center[1] = 3*k_height/4 when i is even, k_height/2 otherwise.
        center[0] = (i + 1) * radius, center[1] = 3 * k_height / 4 - (i % 2) * k_height / 4;
        drawCircle(center, radius, k_ringColors[i]);
    }
    glEnd();
    glFlush();
}

void drawOlympicRingsAsEllipses()
{
    glBegin(GL_POINTS);
    int scale = min(k_width, k_height);
    int radiusX = scale / 6, radiusY = scale / 8;
    int center[2];
    for (int i = 0; i < 5; ++i)
    {
        // center[1] = 3*k_height/4 when i is even, 11*k_height/20 otherwise.
        center[0] = (i + 1) * radiusX, center[1] = 3 * k_height / 4 - (i % 2) * k_height / 5;
        drawEllipse(center, radiusX, radiusY, k_ringColors[i]);
    }
    glEnd();
    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(k_windowPosX, k_windowPosY);
    glutInitWindowSize(k_width, k_height);
    glutCreateWindow("Olympic Rings");
    init();
    glutDisplayFunc(drawOlympicRingsAsCircles);
    glutMainLoop();
    return 0;
}
```

OUTPUT:



3.Create a class room scene using the algorithms that you have learnt.

```
#include <GL/glut.h>
#include <iostream>
#define PI 3.14159265

// Camera angles
float angle = 0.0, deltaAngle = 0.0;
float x = 0.0f, z = 5.0f, lx = 0.0f, lz = -1.0f;

// Fan rotation
float fanAngle = 0.0;
// Function to draw a cube
void drawCube1(float x, float y, float z, float width, float height, float depth) {
    glPushMatrix();
    glTranslatef(x, y, z);
    glScalef(width, height, depth);
    glutSolidCube(1);
    glPopMatrix();
}

// Function to draw a cylinder (for fans and poles)
void drawCylinder1(float base, float top, float height) {
    GLUquadric* quad = gluNewQuadric();
    gluCylinder(quad, base, top, height, 20, 20);
    gluDeleteQuadric(quad);
}

void drawCircle(float x, float y, float radius) {
    glBegin(GL_LINE_LOOP);
    for (int i = 0; i < 360; i++) {
        float angle = i * PI / 180;
        glVertex2f(x + cos(angle) * radius, y + sin(angle) * radius);
    }
    glEnd();
}

void drawClock(float x, float y, float radius, float hour, float minute) {
    drawCircle(x, y, radius);

    float hourAngle = (hour / 12.0) * 360.0 * PI / 180;
    float minuteAngle = (minute / 60.0) * 360.0 * PI / 180;

    glBegin(GL_LINES);
    // Hour hand
    glVertex2f(x, y);
    glVertex2f(x + cos(PI / 2 - hourAngle) * radius * 0.5, y + sin(PI / 2 - hourAngle) * radius * 0.5);

    // Minute hand
    glVertex2f(x, y);
    glVertex2f(x + cos(PI / 2 - minuteAngle) * radius * 0.8, y + sin(PI / 2 - minuteAngle) * radius * 0.7);
    glEnd();
}

// Function to draw a cube
void drawCube(float x, float y, float z, float width, float height, float depth) {
    glPushMatrix();
    glTranslatef(x, y, z);
    glScalef(width, height, depth);
    glutSolidCube(1);
    glPopMatrix();
}

// Function to draw a cylinder (for fans and poles)
void drawCylinder(float base, float top, float height) {
    GLUquadric* quad = gluNewQuadric();
    gluCylinder(quad, base, top, height, 20, 20);
    gluDeleteQuadric(quad);
}

void drawStickman(float x, float y) {
    // Head
    drawCircle(x, y + 0.1, 0.05);

    // Body
    glBegin(GL_LINES);
    glVertex2f(x, y + 0.05);
    glVertex2f(x, y - 0.05);

    // Arms
    glVertex2f(x, y);
    glVertex2f(x - 0.05, y - 0.02);

    glVertex2f(x, y);
    glVertex2f(x + 0.05, y - 0.02);

    // Legs
    glVertex2f(x, y - 0.05);
    glVertex2f(x - 0.03, y - 0.1);

    glVertex2f(x, y - 0.05);
    glVertex2f(x + 0.03, y - 0.1);
    glEnd();
}
```

```

// Function to render the scene
void renderScene() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    gluLookAt(x, 1.5f, z, x + lx, 1.5f, z + lz, 0.0f, 1.0f, 0.0f);

    // Classroom floor
    glColor3f(0.7f, 0.7f, 0.7f);
    drawCube(0.0f, -0.5f, 0.0f, 5.0f, 0.1f, 5.0f);

    // Walls
    glColor3f(0.9f, 0.8f, 0.7f);
    drawCube(0.0f, 1.5f, -2.5f, 5.0f, 3.0f, 0.1f);
    drawCube(-2.5f, 1.5f, 0.0f, 0.1f, 3.0f, 5.0f);
    drawCube(2.5f, 1.5f, 0.0f, 0.1f, 3.0f, 5.0f);

    // Blackboard
    glColor3f(0.0f, 0.3f, 0.0f);
    drawCube(0.0f, 1.5f, -2.45f, 1.5f, 0.8f, 0.05f);

    // Teacher's desk
    glColor3f(0.6f, 0.4f, 0.2f);
    drawCube(0.0f, 0.0f, -1.5f, 1.0f, 0.5f, 0.5f);

    // Stickman Teacher

    // Stickman Students
    drawStickman(0, 0);

    // Stickman Students
    drawStickman(-1.0, -0.1);
    drawStickman(-0.8, -0.1);
    drawStickman(0.8, -0.1);
    drawStickman(1.0, -0.1);

    // Clock
    drawClock(1.0, 2.0, 0.15, 3, 15);
    // Fan (rotating)
    glColor3f(0.7f, 0.7f, 0.7f);
    glPushMatrix();
    glTranslatef(0.0f, 2.8f, 0.0f);
    drawCylinder1(0.05f, 0.05f, 0.3f);
    glRotatef(fanAngle, 0, 1, 0);
    for (int i = 0; i < 3; i++) {
        glRotatef(120, 0, 1, 0);
        drawCube1(0.3f, 0.0f, 0.0f, 0.6f, 0.05f, 0.1f);
    }
    glPopMatrix();

    glutSwapBuffers();
}

void update(int value) {
    fanAngle += 10.0f;
    if (fanAngle > 360) fanAngle -= 360;
    glutPostRedisplay();
    glutTimerFunc(50, update, 0);
}


// Initialize OpenGL
void initGL() {
    glEnable(GL_DEPTH_TEST);
    glClearColor(0.5f, 0.5f, 0.5f, 1.0f);
    glMatrixMode(GL_PROJECTION);
    gluPerspective(45.0, 1.0, 0.1, 100);
    glMatrixMode(GL_MODELVIEW);
}

// Main function
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Classroom Scene in OpenGL");

    initGL();
    glutDisplayFunc(renderScene);
    glutTimerFunc(50, update, 0);
    glutMainLoop();
    return 0;
}
```

OUTPUT: