// Name – Dahiwal Satyam Santoshkumar

// Roll Number – 207019

#include <stdio.h>

#include <iostream>

#include <math.h>

#include <GL/glut.h>

using namespace std;

double a = 0, b = 0, c = 0, d = 0, e = 0, f = 0, g = 0, h = 0;

float round\_value(float v) {

return floor(v + 0.5);

}

void lineDDA(double x1, double y1, double x2, double y2) {

double dx = (x2 - x1);

double dy = (y2 - y1);

double steps;

float xInc, yInc;

float x = x1, y = y1;

steps = (abs(dx) > abs(dy)) ? abs(dx) : abs(dy);

xInc = dx/(float)steps;

yInc = dy/(float)steps;

glBegin(GL\_POINTS);

glVertex2d(x, y);

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

x = x + xInc;

y = y + yInc;

glVertex2d(round\_value(x), round\_value(y));

}

glEnd();

glFlush();

}

void display(void) {

glClear(GL\_COLOR\_BUFFER\_BIT); //clears the OpenGL color buffer (or any other buffer or combination of buffers by using bitwise or |). OpenGL being a state machine, it is good practice to start each frame with a clean slate.

double a1, b1, c1, d1, e1, f1, g1, h1;

double a2, b2, c2, d2, e2, f2, g2, h2;

lineDDA(a, e, b, f);

lineDDA(b, f, c, g);

lineDDA(c, g, d, h);

lineDDA(d, h, a, e);

a1=(a+b)/2; e1=(e+f)/2;

b1=(b+c)/2; f1=(g+f)/2;

c1=(c+d)/2; g1=(g+h)/2;

d1=(a+d)/2; h1=(e+h)/2;

lineDDA(a1, e1, b1, f1);

lineDDA(b1, f1, c1, g1);

lineDDA(c1, g1, d1, h1);

lineDDA(d1, h1, a1, e1);

a2=(a1+b1)/2; e2=(e1+f1)/2;

b2=(b1+c1)/2; f2=(g1+f1)/2;

c2=(c1+d1)/2; g2=(g1+h1)/2;

d2=(a1+d1)/2; h2=(e1+h1)/2;

lineDDA(a2, e2, b2, f2);

lineDDA(b2, f2, c2, g2);

lineDDA(c2, g2, d2, h2);

lineDDA(d2, h2, a2, e2);

}

void Init() {

glClearColor(1.0, 1.0, 1.0, 0); /\*Background color. glClearColor specifies the red, green, blue, and alpha values used by glClear to clear the color buffers. Values specified by glClearColor are clamped to the range. \*/

glColor3f(0.0, 0.0, 0.0); /\*Vertex color. 3f : 3 for three parameters and f for floating point value of the parameter. . Here color of vertices will be black (0, 0, 0) unless new glColor3f() is used \*/

glViewport(0, 0, 640, 480); /\*Set the viewport. parameters : (xOrigin, yOrigin, width, height) \*/

glMatrixMode(GL\_PROJECTION); /\*Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted: GL\_MODELVIEW, GL\_PROJECTION, and GL\_TEXTURE. The initial value is GL\_MODELVIEW \*/

glLoadIdentity(); /\*glLoadIdentity — replace the current matrix with the identity matrix \*/

gluOrtho2D(0, 640, 0, 480); /\*gluOrtho2D sets up a two-dimensional orthographic viewing region. Parameters are : (left, right, bottom, top) \*/

// small 'd' for decimal and capital 'D' for dimension.

}

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

cout << "Please enter 4 points of Polygon in clockwise direction : ";

cout << "\nP1 :"; cin >> a; cin >> e;

cout << "P2 :"; cin >> b; cin >> f;

cout << "P3 :"; cin >> c; cin >> g;

cout << "P4 :"; cin >> d; cin >> h;

glutInit(&argc, argv); // glutInit is used to initialize the GLUT library.

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

/\*

glutInitDisplayMode sets the initial display mode.

Parameters :

GLUT\_SINGLE

Bit mask to select a single buffered window. This is the default if neither GLUT\_DOUBLE or GLUT\_SINGLE are specified.

GLUT\_RGB

Bit mask to select an RGBA mode window. This is the default if neither GLUT\_RGBA nor GLUT\_INDEX are specified. GLUT\_RGB is an alias for GLUT\_RGBA

\*/

glutInitWindowSize(640, 480);

glutInitWindowPosition(0, 0);

/\*

glutInitWindowPosition and glutInitWindowSize set the initial window position and size respectively.

void glutInitWindowSize(int width, int height);

void glutInitWindowPosition(int x, int y);

\*/

glutCreateWindow("DDA"); /\* glutCreateWindow creates a top-level window. The name will be provided to the window system as the window's name. The intent is that the window system will label the window with the name \*/

Init();

glutDisplayFunc(display); /\* glutDisplayFunc sets the display callback for the current window. \*/

glFlush(); //glFlush — force execution of GL commands in finite time

/\*

Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. glFlush empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

Because any GL program might be executed over a network, or on an accelerator that buffers commands, all programs should call glFlush whenever they count on having all of their previously issued commands completed. For example, call glFlush before waiting for user input that depends on the generated image.

\*/

glutMainLoop(); /\*glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered. \*/

}

/\* OUTPUT

satyam@ubuntu:~$ g++ dda.cpp -lglut -lGL -lGLEW -lGLU -o dda

satyam@ubuntu:~$ ./dda

Please enter 4 points of Polygon in clockwise direction :

P1 :30

30

P2 :30

300

P3 :330

300

P4 :330

30

