#include <iostream>  
#include "stdafx.h"  
#include<stdlib.h>  
#include<GL/glut.h>  
#include <math.h>  
#include<stdio.h>  
#define MIN(X,Y) ((X) < (Y) ?  (X) : (Y))  
#define MAX(X,Y) ((X) > (Y) ?  (X) : (Y))  
#define N 3  
int mat[1000][1000] ;  
int boundry[1000][1000] ;  
int sign(float x)  
{  
    if(x<0.0)  
        return -1 ;  
    else if (x>0.0)  
        return 1;  
    else  
        return 0 ;  
}  
  
  
void getCofactor(float A[N][N], float temp[N][N], int p, int q, int n)  
{  
    int i = 0, j = 0;  
   
    // Looping for each element of the matrix  
    for (int row = 0; row < n; row++)  
    {  
        for (int col = 0; col < n; col++)  
        {  
            //  Copying into temporary matrix only those element  
            //  which are not in given row and column  
            if (row != p && col != q)  
            {  
                temp[i][j++] = A[row][col];  
   
                // Row is filled, so increase row index and  
                // reset col index  
                if (j == n - 1)  
                {  
                    j = 0;  
                    i++;  
                }  
            }  
        }  
    }  
}  
   
  
int determinant(float A[N][N], int n)  
{  
    int D = 0;   
   
     
    if (n == 1)  
        return A[0][0];  
   
    float temp[N][N];   
   
    int sign = 1;    
   
      
    for (int f = 0; f < n; f++)  
    {  
          
        getCofactor(A, temp, 0, f, n);  
        D += sign \* A[0][f] \* determinant(temp, n - 1);  
  
          
        sign = -sign;  
    }  
   
    return D;  
}  
   
  
void adjoint(float A[N][N],float adj[N][N])  
{  
    if (N == 1)  
    {  
        adj[0][0] = 1;  
        return;  
    }  
   
      
    int sign = 1; float temp[N][N];  
   
    for (int i=0; i<N; i++)  
    {  
        for (int j=0; j<N; j++)  
        {  
              
            getCofactor(A, temp, i, j, N);  
  
              
            sign = ((i+j)%2==0)? 1: -1;  
   
             
            adj[j][i] = (sign)\*(determinant(temp, N-1));  
        }  
    }  
}  
  
bool inverse(float A[N][N], float inverse[N][N])  
{  
      
    int det = determinant(A, N);  
    if (det == 0)  
    {  
       
        return false;  
    }  
   
      
    float adj[N][N];  
    adjoint(A, adj);  
   
     
    for (int i=0; i<N; i++)  
        for (int j=0; j<N; j++)  
            inverse[i][j] = adj[i][j]/float(det);  
   
    return true;  
}  
   
// Generic function to display the matrix.  We use it to display  
// both adjoin and inverse. adjoin is integer matrix and inverse  
// is a float.  
template<class T>  
void display(T A[N][N])  
{  
    for (int i=0; i<N; i++)  
    {  
        for (int j=0; j<N; j++)  
            cout << A[i][j] << " ";  
        cout << endl;  
    }  
}  
   
  
  
  
  
void DDA(float x1,float y1,float x2,float y2,float size)  
{  
   
    float delx,dely ;  
    delx=abs(x2-x1) ;  
    dely=abs(y2-y1) ;  
    float x=0.0,y=0.0 ;  
    int s1=sign(x2-x1) ;  
    int s2=sign(y2-y1) ;  
    x=x1 ;  
    y=y1 ;  
    int interchange=0 ;  
    if(dely>delx)  
    {  
        float temp=delx ;  
        delx=dely ;  
        dely=temp ;  
        interchange=1 ;  
    }  
    int i=1 ;  
    float e =2\*dely-delx ;  
    for(i=1;i<=delx;i++)  
    {  
      
            glPointSize(size) ;  
            glBegin(GL\_POINTS);  
            glColor3f(0.0, 1.0, 0.0);  
      
            glVertex2d((float)(x),(float)(y));  
            boundry[(int)x][(int)y]=1 ;  
            glEnd();  
        //glFlush ();  
        while(e>0)  
        {  
            if(interchange==1)  
                x=x+s1 ;  
            else  
                y=y+s2 ;  
            e=e-2\*delx ;  
        }  
        if(interchange==1)  
            y=y+s2 ;  
        else  
            x=x+s1 ;  
        e=e+2\*dely ;  
       // i++ ;  
    }  
  
}  
  
void multiply(float mat1[][N], float mat2[][N], float res[][N])  
{  
    int i, j, k;  
    for (i = 0; i < N; i++)  
    {  
        for (j = 0; j < N; j++)  
        {  
            res[i][j] = 0;  
            for (k = 0; k < N; k++)  
                res[i][j] += mat1[i][k]\*mat2[k][j];  
        }  
    }  
}  
  
void Primitives(void)  
{  
//clear all pixels  
glClear (GL\_COLOR\_BUFFER\_BIT);  
/\*  
glBegin(GL\_LINES);  
glVertex2d(100,100);  
glVertex2d(600,100);  
glVertex2d(600,700);  
glVertex2d(300,400);  
glVertex2d(100,600);  
//glVertex2d(500,400);  
glEnd() ;  
\*/  
// draws a colorful triangle  
//DDA(0,0,70,20) ;  
//DDA(0,0,10,10);  
//DDA(6,3,4,3) ;  
//DDA(4,3,3,5) ;  
//DDA(3,5,4,7) ;  
//DDA(4,7,6,7) ;  
//DDA(6,7,7,5) ;  
//DDA(7,5,6,3) ;  
/\*  
DDA(500,400,450,313,2.0) ;  
DDA(450,313,350,313,2.0) ;  
DDA(350,313,300,400,2.0) ;  
DDA(300,400,350,487,2.0) ;  
DDA(350,487,450,487,2.0) ;  
DDA(450,487,500,400,2.0) ;  
  
DDA(520,400,460,296,4.0);  
  
DDA(460,296,340,296,4.0);  
DDA(340,296,280,400,4.0);  
DDA(280,400,340,504,4.0);  
DDA(340,504,460,504,4.0);  
DDA(460,504,520,400,4.0);  
  
DDA\_dash(550,400,475,270,4.0) ;  
DDA\_dash(475,270,325,270,4.0) ;  
DDA\_dash(325,270,250,400,4.0) ;  
DDA\_dash(250,400,325,530,4.0) ;  
DDA\_dash(325,530,475,530,4.0) ;  
DDA\_dash(475,530,550,400,4.0) ;  
  
DDA\_dot(580,400,490,244,8.0);  
DDA\_dot(490,244,310,244,8.0);  
DDA\_dot(310,244,220,400,8.0);  
DDA\_dot(220,400,310,556,8.0);  
DDA\_dot(310,556,490,556,8.0);  
DDA\_dot(490,556,580,400,8.0);  
\*/  
//circle\_fq(500,500,250);  
//elipse(400,575,25,35);  
//elipse(600,575,25,35);  
//elipse(500,350,50,40);  
DDA(100,100,200,100,4.0) ;  
DDA(200,100,150,200,4.0) ;  
DDA(150,200,100,100,4.0) ;  
  
DDA(261.6,206.69,211.6,293.29,4.0) ;  
DDA(211.6,293.29,150,200,4.0) ;  
DDA(150,200,261.6,206.69,4.0) ;  
  
  
float A[N][N] = { {100,100, 1},  
                    {200,100, 1},  
                    {150, 200, 1}};  
   
    float adj[N][N];    
   
    float inv[N][N];   
   
      
   
   // cout << "\nThe Inverse is :\n";  
    inverse(A, inv);  
      //  display(inv);  
  
     float mat2[N][N] = { {261.6,206.69, 1},  
                    {211.6,293.29, 1},  
                    {150, 200, 1}};  
  
     float res[N][N]; // To store result  
     float res2[N][N];  
    int i, j;  
    multiply(inv, mat2, res);  
    multiply(mat2,res,res2) ;  
  
    DDA(res2[0][0],res2[0][1],res2[1][0],res2[1][1],4.0) ;  
    DDA(res2[1][0],res2[1][1],res2[2][0],res2[2][1],4.0) ;  
    DDA(res2[2][0],res2[2][1],res2[0][0],res2[0][1],4.0) ;  
     
  
//seed\_fill(200,300) ;  
/\*  
poly\_color(100,100,600,100) ;  
poly\_color(600,100,600,700) ;  
poly\_color(600,700,300,400) ;  
poly\_color(300,400,100,600) ;  
poly\_color(100,600,100,100) ;  
\*/  
//circle\_fq(350,150,100);  
glFlush ();  
}  
  
  
  
void Init()  
  
{  
  
glClearColor(1.0,1.0,1.0,0);  
  
    //glColor3f(1.0,0.0,0.0); // red  
  
//glViewport(0 , 0 , 800 , 800);  
glMatrixMode(GL\_PROJECTION);  
  
glLoadIdentity();  
  
gluOrtho2D(0 , 800 , 0 , 800);  
  
}  
  
int main(int argc, char \*\*argv)  
  
{  
  
glutInit(&argc,argv);  
int i,j ;  
for(i=0;i<1000;i++)  
{  
    for(j=0;j<1000;j++)  
        mat[i][j]=boundry[i][j]=0 ;  
     
}  
  
glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  
  
glutInitWindowSize(800,800);  
  
glutInitWindowPosition(80,80);  
  
glutCreateWindow("Primitives");  
  
Init();  
  
glutDisplayFunc(Primitives);  
  
glutMainLoop();  
  
return 0;  
  
}

