#include<GL/freeglut.h>

#include<GL/gl.h>

#include<math.h>

#include<iostream>

using namespace std;

int mat1[24][3];

float trans[3][3];

int ans[24][3];

void putbresenhampixel(int x, int y){

glColor3f(0,0,0);

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

glFlush();

}

int sign(int k){

if(k==0){

return 1;

}else{

return k/abs(k);

}

}

void bresenham(int xs, int ys, int xe, int ye){

int x,y,dx,dy,s1,s2,sw,n,p;

dx=abs(xe-xs);

dy=abs(ye-ys);

x=xs;

y=ys;

s1=sign(xe-xs);

s2=sign(ye-ys);

if(dy>dx){

sw=dy;

}else{

sw=dx;

}

n=1;

p=(2\*dy)-dx;

putbresenhampixel(x,y);

while(n<sw){

if(dx>dy){

if(p<0){

x=x+s1;

p+=2\*dy;

}else{

x=x+s1;

y=y+s2;

p+=2\*(dy-dx);

}

}else{

if(p<0){

y=y+s2;

p+=2\*dx;

}else{

x=x+s1;

y=y+s2;

p+=2\*(dx-dy);

}

}

putbresenhampixel(x,y);

n++;

}

}

void getmypixel(int x, int y, float interiorcolor[3]){

glReadPixels(x,y,1,1,GL\_RGB,GL\_FLOAT, interiorcolor);

}

void putpixel(int x,int y,float fillcolor[3]){

glBegin(GL\_POINTS);

glColor3f(0,0,0);

glColor3fv(fillcolor);

glVertex2i(x,y);

glEnd();

glFlush();

}

void boundaryfill(int x, int y, float boundarycolor[3], float fillcolor[3]){

float interiorcolor[3];

getmypixel(x,y,interiorcolor);

if((interiorcolor[0]!= boundarycolor[0] || interiorcolor[1]!= boundarycolor[1] || interiorcolor[2]!= boundarycolor[2])&&(interiorcolor[0]!= fillcolor[0] || interiorcolor[1]!= fillcolor[1] || interiorcolor[2]!= fillcolor[2])){

putpixel(x,y, fillcolor);

boundaryfill(x-1, y, boundarycolor, fillcolor);

boundaryfill(x+1, y, boundarycolor, fillcolor);

boundaryfill(x, y-1, boundarycolor, fillcolor);

boundaryfill(x, y+1, boundarycolor, fillcolor);

}

}

void filling(){

float fillcolor[3] ={1.0, 0.0, 0.0};

float boundarycolor[3] ={0.0, 0.0, 0.0};

boundaryfill(175, 275, boundarycolor, fillcolor);

}

void display(){

glColor3f(0,0,0);

for(int i=0;i<24;i+=2){

bresenham(ans[i][0], ans[i][1], ans[i+1][0], ans[i+1][1]);

}

}

void initialco(){

int tempmat1[24][3]={

{100, 300, 1},

{100, 250, 1},

{100, 250, 1},

{150, 250, 1},

{150, 250, 1},

{150, 200, 1},

{150, 200, 1},

{200, 200, 1},

{200, 200, 1},

{200, 250, 1},

{200, 250, 1},

{250, 250, 1},

{250, 250, 1},

{250, 300, 1},

{250, 300, 1},

{200, 300, 1},

{200, 300, 1},

{200, 350, 1},

{200, 350, 1},

{150, 350, 1},

{150, 350, 1},

{150, 300, 1},

{150, 300, 1},

{100, 300, 1},

};

for(int i=0; i<24; i++){

for(int j=0;j<3;j++){

mat1[i][j]=tempmat1[i][j];

}

}

}

void before(){

initialco();

glColor3f(0,0,0);

glBegin(GL\_LINES);

for(int i=0;i<24;i+=2){

bresenham(mat1[i][0], mat1[i][1], mat1[i+1][0], mat1[i+1][1]);

}

glEnd();

glFlush();

}

void transformation(int theta){

float thetarad = theta \* 3.14/180;

initialco();

//translation

for(int i=0; i<3; i++){

for(int j=0; j<3; j++){

if(i==j){

trans[i][j]=1;

}else{

trans[i][j]=0;

}

}

}

trans[2][0]=-175;

trans[2][1]=-275;

for(int i=0; i<24; i++){

for(int j=0; j<3; j++){

ans[i][j]=0;

for(int k=0; k<3; k++){

ans[i][j]+=mat1[i][k]\*trans[k][j];

}

}

}

//rotation

for(int i=0; i<3; i++){

for(int j=0; j<3; j++){

if(i==j){

trans[i][j]=1;

}else{

trans[i][j]=0;

}

}

}

trans[0][0]=cos(thetarad);

trans[1][1]=cos(thetarad);

trans[0][1]=-sin(thetarad);

trans[1][0]=sin(thetarad);

for(int i=0; i<24; i++){

for(int j=0; j<3; j++){

mat1[i][j]=0;

for(int k=0; k<3; k++){

mat1[i][j]+=ans[i][k]\*trans[k][j];

}

}

}

//mat1[1][0]=mat1[2][0]=86;

//mat1[1][1]=mat1[2][1]=50;

//mat1[0][2]=mat1[1][2]=1;

//translation back

for(int i=0; i<3; i++){

for(int j=0; j<3; j++){

if(i==j){

trans[i][j]=1;

}else{

trans[i][j]=0;

}

}

}

trans[2][0]=175;

trans[2][1]=275;

for(int i=0; i<24; i++){

for(int j=0; j<3; j++){

ans[i][j]=0;

for(int k=0; k<3; k++){

ans[i][j]+=mat1[i][k]\*trans[k][j];

}

}

}

display();

}

void renderfunc(){

glClearColor(1,1,1,0);

glClear(GL\_COLOR\_BUFFER\_BIT);

gluOrtho2D(0,700,0,500);

int choice=2;

if(choice==1){ //original

before();

}else if(choice==2){ //translation

transformation(30);

//before();

filling();

}

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE);

glutInitWindowSize(700,500);

glutInitWindowPosition(100,100);

glutCreateWindow("All transformations");

glutDisplayFunc(renderfunc);

glutMainLoop();

return 1;

}