Experiment No : 11

Aim : To implement the 2D transformation on a polygon.

Theory :
Geometric Transformations :
Operations that are applied to the geometric description of an object to change its position, orientation, or size are called geometric transformations. The geometric-transformation functions that are available in all graphics packages are those for translation, rotation, and scaling. Other useful transformation routines that are sometimes included in a package are reflection and shearing operations. In computer graphics, geometric transforms are represented as matrices that the original points are multiplied by or added to.

Homogeneous Coordinates :
Homogeneous coordinates are ubiquitous in computer graphics because they allow common vector operations such as translation, rotation, scaling and perspective projection to be represented as a matrix by which the vector is multiplied. By the chain rule, any sequence of such operations can be multiplied out into a single matrix, allowing simple and efficient processing. By contrast, using Cartesian coordinates, translations and perspective projection cannot be expressed as matrix multiplications, though other operations can. Modern OpenGL and Direct3D graphics cards take advantage of homogeneous coordinates to implement a vertex shader efficiently using vector processors with 4-element registers.

Code & Output :
→ Matrix

```
#include<iostream>

using namespace std;
class matrix;
matrix matmul(matrix a, matrix b);

class matrix{
	public:
	int i,j;
	float **values;

	matrix(int I, int J){
		i=I;j=J;
		int x,y;
		values=(float**)malloc(sizeof(float*)*i);
		for(x=0;x<i;x++){
			values[x]=(float*)malloc(sizeof(float)*j);
			for(y=0;y<j;y++){
				values[x][y]=0;
			}
		}
	}

	matrix(int I, int J, float vals[]){
		i=I;
		j=J;
		int x,y;
		values=(float**)malloc(sizeof(float*)*i);
		for(x=0;x<i;x++){
			values[x]=(float*)malloc(sizeof(float)*j);
			for(y=0;y<j;y++){
```

```
                        values[x][y]=vals[x*j+y];
                    }
                }
        }
        void display(){
                int x,y;
                for(x=0;x<i;x++){
                    for(y=0;y<j;y++){
                        cout<<values[x][y]<<", ";
                    }
                    cout<<"\b\b"<<endl;
                }
        }
        float* operator[](int index){
                return values[index];
        }
};

matrix matmul(matrix a, matrix b){
        if(a.j!=b.i){
                cout<<"Matmul error: Matrices have incorrect dimensions"<<endl;
                exit(0);
        }
        matrix c(a.i,b.j);
        for (int i = 0; i < a.i; i++)
                for (int k = 0; k < a.j; k++)
                    for (int j = 0; j < b.j; j++)
                        c.values[i][j] += a.values[i][k] * b.values[k][j];
        return c;
}
```

→ **Transformations**

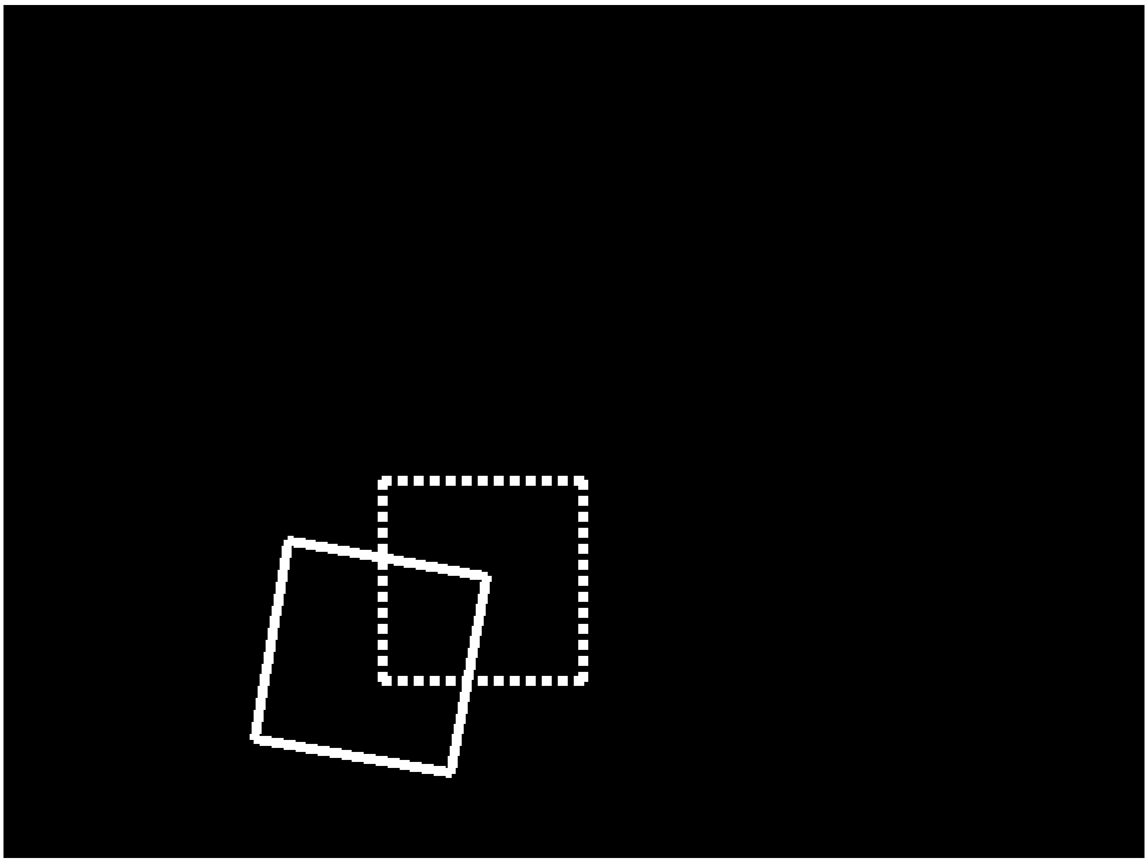
```
#include <iostream>
#include<math.h>
#include <list>
#include<graphics.h>
#include "matrix.h"

#define PI 3.14159
#define to_rad(x)  ((x/180)*PI)
using namespace std;

int thickness=4;

void eg(){
        int i=5,j=2;
        float values[]={1,2,3,4,5,6,7,8,9,10};
        matrix a=       matrix(i,j,values);
        matrix b=       matrix(j,i,values);
        /* matmul(a,b).display(); */
}
typedef struct Coordinates{
        int x,y;
}Coordinates;

typedef struct Point{
        matrix h; // homogenous matrix representation
        Point(int x,int y):h(3,1){
                h[0][0]=x; h[1][0]=y; h[2][0]=1;
        }
```

```
	Point(matrix m):h(3,1){
		if(m.i!=3 || m.j!=1){
			cout<<"Cannot convert matrix with dimensions ";
			cout<<m.i<<" "<<m.j<<"into Point(x,y)"<<endl;
			exit(1);
		}
		h[0][0]=m[0][0]; h[1][0]=m[1][0]; h[2][0]=1;
	}
	Coordinates getCoor(){
		return {(int)h[0][0],(int)h[1][0]};
	}
	void display(){
		Coordinates p=this->getCoor();
		cout<<p.x<<" "<<p.y;
	}
}Point;


typedef struct Polygon{
	int sides;
	list<Point> points;
	Polygon(){
		sides=0;
	}
	Polygon(int n, int polypoint[]){
		int i;
		for(i=0;i<n*2;i+=2){
			points.emplace_back(polypoint[i],polypoint[i+1])		;
		}
	}
	void display(){
		Coordinates c;
		for(Point p:points){
			c=p.getCoor();
			cout<<c.x<<" "<<c.y<<endl;
		}
	}
	void draw(int style=SOLID_LINE){
		Coordinates start,end;
		list<Point>::iterator it=points.begin();
		setlinestyle(style,2,thickness)	;
		start=it->getCoor();
		it++;
		while(it!=points.end()){
			end=it->getCoor();
			line(start.x,start.y,end.x,end.y);
			/* cout<<(start.x)<<" "<<start.y<<endl; */
			/* cout<<end.x<<" "<<end.y<<endl; */
			/* cout<<endl; */
			start=end;
			it++;
		}
		end=points.front().getCoor();
		line(start.x,start.y,end.x,end.y);
		/* cout<<start.x<<" "<<start.y<<endl; */
		/* cout<<end.x<<" "<<end.y<<endl<<endl; */
	}
}Polygon;

Polygon translate(Polygon poly, float tx, float ty){
        Polygon result=Polygon();
        float values[]={
                1,0,tx,
                0,1,ty,
                0,0,1
        };
        matrix transMatrix=matrix(3,3,values);

        for(Point p:poly.points){
                result.points.push_back(matmul(transMatrix,p.h));
        }
        return result;
        /* return Point(matmul(transMatrix,p.h)); */
}

Polygon rotate(Polygon poly, float theta){
        theta=to_rad(theta);
        Polygon result=Polygon();
        float values[]={
                cos(theta),-sin(theta),0,
                sin(theta),cos(theta),0,
                0,      0,      1
        };

        matrix rotMatrix=matrix(3,3,values);
        for(Point p:poly.points){
                result.points.push_back(matmul(rotMatrix,p.h));
        }
        return result;
        /* return Point(matmul(transMatrix,p.h)); */
}

Polygon scale(Polygon poly, float sx, float sy){
        Polygon result=Polygon();
        float values[]={
                sx,0,0,
                0,sy,0,
                0,0,1
        };
        matrix transMatrix=matrix(3,3,values);

        for(Point p:poly.points){
                result.points.push_back(matmul(transMatrix,p.h));
        }
        return result;
}

Polygon pivot(Polygon poly, float xr, float yr, float theta){
        return translate(rotate(translate(poly,-xr, -yr),theta),xr,yr);
}

Polygon fixedPointScale(Polygon poly, float xr, float yr, float sx, float sy){
        return translate(scale(translate(poly,-xr, -yr),sx,sy),xr,yr);
}

Polygon reflect(Polygon poly, float xr,float yr, float theta){
        Polygon result=Polygon();
        theta=to_rad(theta);
        Polygon p1=rotate(translate(poly,-xr,-yr),theta);
        float values[]={
```

```
                        1,0,0,
                        0,-1,0,
                        0,0,1
            };
            matrix refMatrix=matrix(3,3,values);
            for(Point p:p1.points){
                        result.points.push_back(matmul(refMatrix,p.h));
            }
            return translate(rotate(result,-theta),xr,yr);
}
Polygon shearx(Polygon poly,float yref,float shx){
            Polygon result=Polygon();
            float values[]={
                        1,shx,-shx*yref,
                        0,1,0,
                        0,0,1
            };
            matrix shearMatrix=matrix(3,3,values);
            for(Point p:poly.points){
                        result.points.push_back(matmul(shearMatrix,p.h));
            }
            return result;
}
Polygon sheary(Polygon poly,float xref,float shy){
            Polygon result=Polygon();
            float values[]={
                        1,0,0,
                        shy,1,-shy*xref,
                        0,0,1
            };
            matrix shearMatrix=matrix(3,3,values);
            for(Point p:poly.points){
                        result.points.push_back(matmul(shearMatrix,p.h));
            }
            return result;
}
int main(){
            int gd=DETECT,gm;
            initgraph(&gd,&gm,NULL);
            setcolor(WHITE);
            int n=4;
            int dt=3000;
            int polypoint[]={100,200, 100,300, 200,300, 200,200};
            Polygon p=Polygon(n,polypoint);
            p.draw(DASHED_LINE);
            delay(dt/2);
            p=translate(p, 100, 50);
            p.draw();
            delay(dt);
            setbkcolor(BLACK);
            p.draw(DASHED_LINE);

            delay(dt/2);
            Polygon p1=rotate(p,10);
            p1.draw();
            delay(dt);
            setbkcolor(BLACK);

            Polygon p2=scale(p1,1.2,1.2);
            p1.draw(DASHED_LINE);
```

```
        delay(dt/2);
        p2.draw();
        delay(dt);
        setbkcolor(BLACK);


        Polygon p3=reflect(p2,0,320,0);
        p2.draw(DASHED_LINE);
        delay(dt/2);
        p3.draw();
        delay(dt);
        setbkcolor(BLACK);

        p3.draw(DASHED_LINE);
        delay(dt/2);
        Polygon p4=shearx(p3,90,0.5);
        p4.draw();
        delay(dt);
        setbkcolor(BLACK);


        p3.draw(DASHED_LINE);
        Polygon p5=sheary(p3,90,0.5);
        delay(dt/2);
        p5.draw();
        delay(dt);
        setbkcolor(BLACK);

        // shearx(p,200,0.5).draw();
        // sheary(p,100,0.5).draw();
        closegraph();
}
```

Output :



Conclusion : Program to implement 2D transformation on a polygon was successfully written and executed

Deepraj Bhosale
181105016