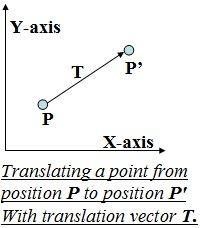
**TITLE: - Translation, Scaling and Rotation**

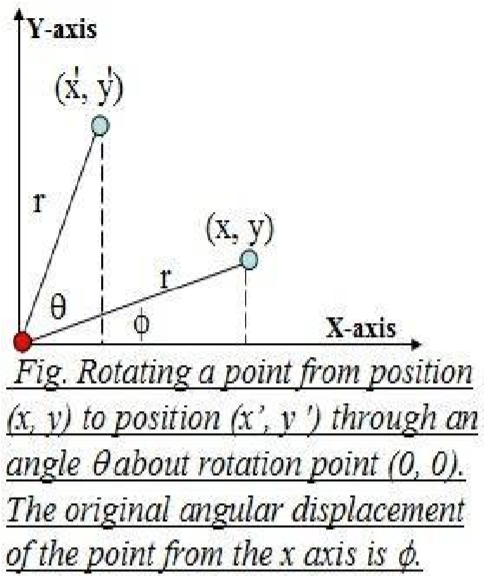
**OBJECTIVE: -**

* To implement 2d transformation.

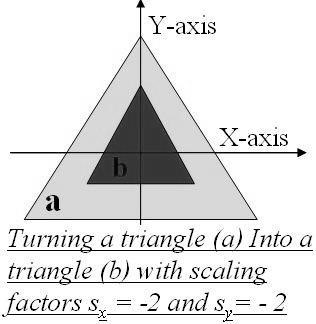
**THEORY: -**

**Translation:** In 2D transformation, translation involves moving an object in the 2D plane along the x and y axes by a specific distance. It shifts the position of the object without changing its size or orientation. If (x,y) is translated to (x’,y’) by translation factor (Tx,Ty); x’=x+Tx and y’=y+Ty.

In homogeneous form, the translation matrix can be represented as:



**Scaling:** Scaling is a 2D transformation that alters the size of an object. It involves multiplying the x and y coordinates of the object by scaling factors, resulting in either enlargement (scaling up) or reduction (scaling down) of the object's size. If (x,y) is translated to (x’,y’) by scaling factor (Sx,Sy); x’=x×Sx and y’=y×Sy.

In homogeneous form, the scaling matrix can be represented as:

**Rotation:** Rotation in 2D transformation involves rotating an object around a fixed point (usually the origin) by a certain angle. It changes the orientation of the object while preserving its shape and size. If (x,y) is rotated to (x’,y’) by angle ; x’=x+Tx and y’=y+Ty.

In homogeneous form, the rotation matrix can be represented as:

**Algorithm for Translation**

1. Read the initial 3D point (x, y, z) and translation distances (tx, ty, tz).
2. Calculate the new coordinates after translation: x' = x + tx y' = y + ty z' = z + tz
3. Display or store the translated point (x', y', z').

**Algorithm for Scaling**

1. Translate the fixed point to the origin.
2. Scale the object relative to the coordinate origin.
3. Translate the fixed point back to its original position.

The composite matrix is given by

T=T (xf, yf, zf).S(Sx, Sy, Sz).T(-xf, -yf, -zf)

**Algorithm for Rotation**

1. Translate the fixed point to the origin.
2. Perform rotation by the given angle Theta.
3. Translate the rotated point back to its original axis.

The composite matrix is given by

T=T (xf, yf, zf).R().T(-xf, -yf, -zf)

**Program to implement Translation, Scaling and Rotation:**

#include <graphics.h>

#include <conio.h>

#include<string.h>

#include<math.h>

#define scalediv 10

#define pixeldiv 20

#define len 4

#define xmax 625 // Last pixel distance from center in x-axis

#define ymax 325 // Last pixel distance from center in Y-axis

int xc = getmaxwidth() / 2, yc = getmaxheight() / 2;

int g\_in(int x\_coordinate, int y\_coordinate)

{

static char data[len];

memset(data,0,sizeof(data));

int i = 0;

while ((data[i] = getch()) != '\r')

{

if(i>=len)

{

outtextxy(x\_coordinate,y\_coordinate+20,"Char Array Limit Reached");

}

if (( isdigit(data[i])||data[i]=='-'||data[i]=='+')&&i<len)

{

outtextxy(x\_coordinate, y\_coordinate, data);

i++;

}

else if (data[i]== '\b'&&i>0)

{

i--;

data[i]='\0';

outtextxy(x\_coordinate,y\_coordinate+20," ");

strcat(data," ");

outtextxy(x\_coordinate, y\_coordinate, data);

}

else{

while (kbhit())

{

getch();

}}

}

int num=std::stoi(data);

return num;

}

int user\_interface(int x, int y)

{

int window = initwindow(getmaxwidth(), getmaxheight(), "2D TRANSFORMATIONS");

settextstyle(8,0,1);

setcolor(2);

outtextxy(5,5,"[1] Translation");

outtextxy(getmaxwidth()/2-70,5,"[2] Scaling");

outtextxy(getmaxwidth()-150,5,"[3] Rotation ");

line(0,25,getmaxwidth(),25);

line(0,getmaxheight()-30,getmaxwidth(),getmaxheight()-30);

outtextxy(10,getmaxheight()-25,"[0] CLEAR");

outtextxy(getmaxwidth()-150,getmaxheight()-25," (X) EXIT [5]");

return window;

}

void plot\_points(int ax, int ay, int bx, int by, char a[], char b[])

{

int x1=ax\*pixeldiv, y1=ay\*pixeldiv, x2=bx\*pixeldiv, y2=by\*pixeldiv;

line(xc+x1,yc-y1,xc+x2,yc-y2);

outtextxy(xc+x1,yc-y1,a);

outtextxy(xc+x2,yc-y2,b);

}

void translation(int x1, int y1, int x2, int y2, int winmain)

{

int window=initwindow(300,300,"Translation");

int Tx,Ty;

outtextxy(10,10,"Enter the Translation factor: ");

Tx=g\_in(30,25);

Ty=g\_in(50,25);

closegraph(window);

setcurrentwindow(winmain);

plot\_points(x1+Tx,y1+Ty,x2+Tx,y2+Ty,"Ta","Tb");

}

void scaling(int x1, int y1, int x2, int y2, int winmain)

{

int window=initwindow(300,300,"Translation");

int Sx,Sy;

outtextxy(10,10,"Enter the Scaling factor: ");

Sx=g\_in(30,25);

Sy=g\_in(50,25);

closegraph(window);

setcurrentwindow(winmain);

plot\_points(x1\*Sx,y1\*Sy,x2\*Sx,y2\*Sy,"Sa","Sb");

}

void rotation(int x1, int y1, int x2, int y2, int winmain)

{

int window=initwindow(300,300,"Translation");

float angle;

outtextxy(10,10,"Enter the angle of rotation: ");

angle=g\_in(25,25);

angle = (angle \* 3.14) / 180;

int a, b, c, d;

a = xc + x1 \* cos(angle) - y1 \* sin(angle);

b = yc + x1 \* sin(angle) + y1 \* cos(angle);

c = xc + x2 \* cos(angle) - y2 \* sin(angle);

d = yc + x2 \* sin(angle) + y2 \* cos(angle);

line(xc + x1, yc + y1, xc + x2, yc + y2);

x1 = a - xc;

y1 = b - yc;

x2 = c - xc;

y2 = d - yc;

closegraph(window);

setcurrentwindow(winmain);

plot\_points(x1,y1,x2,y2,"Ra","Rb");

}

int main()

{

int main\_window=user\_interface(xc,yc);

int p\_xpoint=xc+pixeldiv, n\_xpoint=xc-pixeldiv; //X-axis: y constant

int p\_ypoint=yc-pixeldiv, n\_ypoint=yc+pixeldiv; //y-axis: x constant

for(int i=0; (p\_ypoint<=ymax+yc)&&(p\_xpoint<=xmax+xc); i++)

{

int color = RGB(40,40,40);

setcolor(color);

setlinestyle(DOTTED\_LINE,0,0);

if(n\_ypoint<=ymax+yc)

{

line(xc-xmax,p\_ypoint,xc+xmax,p\_ypoint); //Dotted line in positive Y axis parallel to x axis

line(xc-xmax,n\_ypoint,xc+xmax,n\_ypoint); //Dotted line in negative Y axis parallel to x axis

p\_ypoint-=pixeldiv;

n\_ypoint+=pixeldiv;

}

if(p\_xpoint<=xmax+xc)

{

line(p\_xpoint,yc-ymax,p\_xpoint,yc+ymax); //Dotted line in positive X axis parallel to Y axis

line(n\_xpoint,yc-ymax,n\_xpoint,yc+ymax); //Dotted line in negative X axis parallel to Y axis

p\_xpoint+=pixeldiv;

n\_xpoint-=pixeldiv;

}

}

int x1=0,x2=10,y1=0,y2=5;

int color = RGB(150, 150, 150);

setcolor(color);

setlinestyle(0,0,2);

line(xc,yc-ymax,xc,yc+ymax); //axis line

line(xc-xmax,yc,xc+xmax,yc); //axis line

plot\_points(x1,y1,x2,y2,"A","B"); //reference line to observe transformation

loop:

//Choice and switch case for transformations

char a=getch();

int choice= a-'0';

switch(choice)

{

case 1:

{

setcolor(RED);

translation(x1,y1,x2,y2,main\_window);

goto loop;

}

case 2:

{

setcolor(BLUE);

scaling(x1,y1,x2,y2,main\_window);

goto loop;

}

case 3:

{

setcolor(YELLOW);

rotation(x1,y1,x2,y2,main\_window);

goto loop;

}

case 4:

{

exit(0);

}

case 0:

{

closegraph();

main();

}

default:

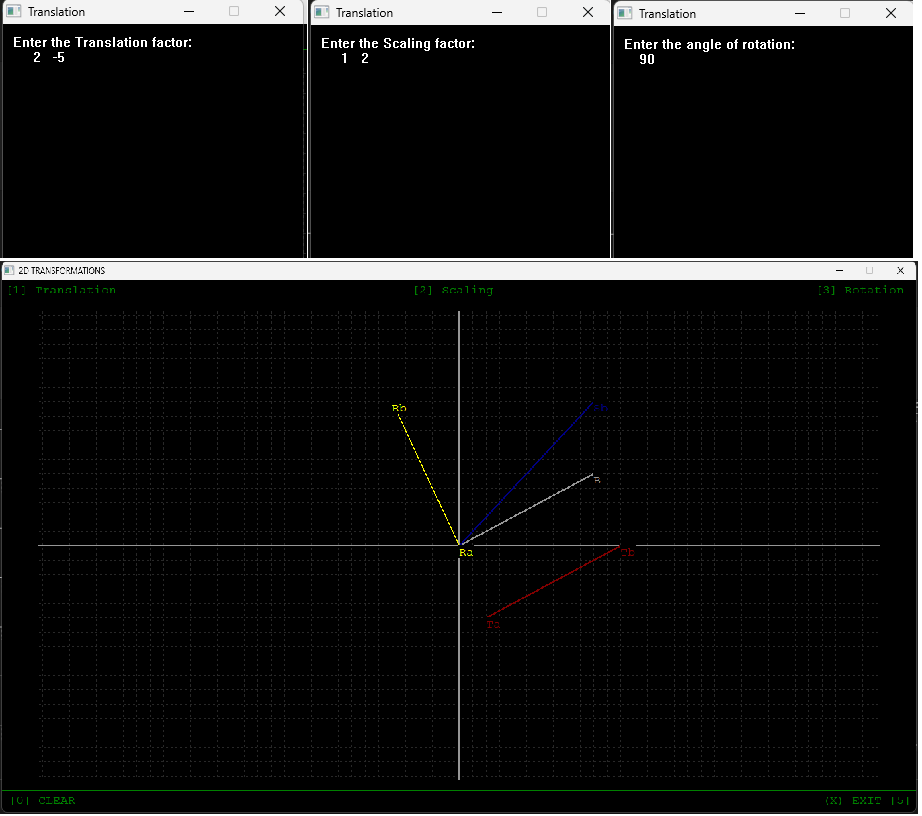
{

goto loop;

}

}

}



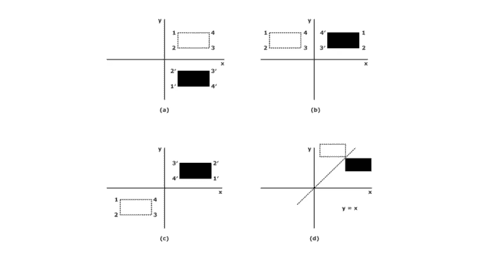
**CONCLUSION**:

In this way, we implement 2D transformations through writing code in C++ programming language and analysed its precision and way of calculation.

**TITLE: - Reflection and Shearing**

**OBJECTIVE: -**

* To implement shearing and reflection in a triangle.

**THEORY: -**

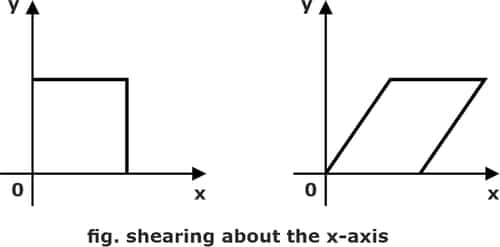
**Reflection:** Reflection is the mirror image of the original object. In other words, we will say that it is the rotation operation with 180­0. In reflection transformation, the object’s size does not change. The following figures reflect the X and Y axes and the origin respectively.

In homogeneous form, the translation matrix can be represented as:

[About X-axis]

[About Y-axis]

[About line x=y]

**Shearing: -** Shearing is the transformation used to change the shape of an existing object in the 2D plane. The size of the object changes along the X-direction as well as the Y-direction. Shearing along the X-axis is as follows:

**Shearing (x-direction reference) / X-axis parallel Shearing:**

**Shearing (y-direction reference) / Y-axis parallel Shearing:**

**Program to implement Reflection:**

#include <graphics.h>

#include <conio.h>

#include<string.h>

#include<math.h>

#define scalediv 10

#define pixeldiv 20

#define len 4

#define xmax 625 // Last pixel distance from center in x-axis

#define ymax 325 // Last pixel distance from center in Y-axis

int xc = getmaxwidth() / 2, yc = getmaxheight() / 2;

static int x1,x2,x3,Y1,y2,y3;

int shape;

int a=x1, c=x2, e=x3, b= Y1, d=y2, f=y3;

int g\_in(int x\_coordinate, int y\_coordinate)

{

static char data[len];

memset(data,0,sizeof(data));

int i = 0;

while ((data[i] = getch()) != '\r'){

if(i>=len)

{

outtextxy(x\_coordinate,y\_coordinate+20,"Char Array Limit Reached");

}

if (( isdigit(data[i])||data[i]=='-'||data[i]=='+')&&i<len)

{

outtextxy(x\_coordinate, y\_coordinate, data);

i++;

}

else if (data[i]== '\b'&&i>0)

{

i--;

data[i]='\0';

outtextxy(x\_coordinate,y\_coordinate+20," ");

strcat(data," ");

outtextxy(x\_coordinate, y\_coordinate, data);

}

else

{

while (kbhit())

{

getch();

}

}

}

if(i>=1){

int num=std::stoi(data);

return num;}

else

g\_in(xc,yc);

}

void user\_interface(int x, int y){

settextstyle(8,0,1);

setcolor(2);

outtextxy(5,5,"[1] REFLECTION ABOUT X AXIS");

outtextxy(getmaxwidth()/2-120,5,"[2] REFLECTION ABOUT X=Y");

outtextxy(getmaxwidth()-320,5,"[3] REFLECTION ABOUT Y AXIS ");

line(0,25,getmaxwidth(),25);

line(0,getmaxheight()-30,getmaxwidth(),getmaxheight()-30);

outtextxy(getmaxwidth()/2-70,getmaxheight()-25,"[4] SET OBJECT");

outtextxy(10,getmaxheight()-25,"[0] CLEAR");

outtextxy(getmaxwidth()-150,getmaxheight()-25," (X) EXIT [5]");

}

void plot\_points(int ax, int ay, int bx, int by, char a[], char b[]){

int x1=ax\*pixeldiv, Y1=ay\*pixeldiv, x2=bx\*pixeldiv, y2=by\*pixeldiv;

line(xc+x1,yc-Y1,xc+x2,yc-y2);

outtextxy(xc+x1,yc-Y1,a);

outtextxy(xc+x2,yc-y2,b);

}

void reflection\_xaxis(){

a=x1;

b=-Y1;

c=x2;

d=-y2;

e=x3;

f=-y3;}

void reflection\_yaxis()

{

a=-x1;

b=Y1;

c=-x2;

d=y2;

e=-x3;

f=y3;}

void reflection\_xyaxis()

{

a=-x1;

b=-Y1;

c=-x2;

d=-y2;

e=-x3;

f=-y3;

}

void plot\_transformation()

{

if(shape==1)

{

plot\_points(a,b,c,d,"A","B");

}

if(shape==2)

{

plot\_points(a,b,c,d,"A","B");

plot\_points(a,b,e,f,"A","C");

plot\_points(c,d,e,f,"B","C");

}

if(shape==3)

{

plot\_points(a,b,a,d,"A","C");

plot\_points(a,d,c,d,"C","D");

plot\_points(c,d,c,b,"D","B");

plot\_points(c,b,a,b,"B","A");

}

}

void set\_object(int winmain)

{

int window=initwindow(500,300,"SET OBJECT");

loop:

outtextxy(10,10,"[1] Line");

outtextxy(10,30,"[2] Triangle");

outtextxy(10,50,"[3] Rectangle");

outtextxy(10,70,"[4] Leave It !!");

char a=getch();

cleardevice();

int choice= a-'0';

switch (choice)

{

case 1:

{

outtextxy(10,10,"Enter two points (x1,Y1) and (x2,y2)");

x1=g\_in(20,30);

Y1=g\_in (50,30);

x2=g\_in(20,50);

y2=g\_in(50,50);

shape=1;

closegraph(window);

setcurrentwindow(winmain);

plot\_points(x1,Y1,x2,y2,"A","B");

break;

}

case 2:

{

outtextxy(10,10,"Enter two diagonal points (x1,Y1) and (x2,y2) and (x3,y3)");

x1=g\_in(20,30);

Y1=g\_in (50,30);

x2=g\_in(20,50);

y2=g\_in(50,50);

x3=g\_in(20,70);

y3=g\_in(50,70);

shape=2;

closegraph(window);

setcurrentwindow(winmain);

plot\_points(x1,Y1,x2,y2,"A","B");

plot\_points(x1,Y1,x3,y3,"A","C");

plot\_points(x2,y2,x3,y3,"B","C");

break;

}

case 3:

{

outtextxy(10,10,"Enter two diagonal points (x1,Y1) and (x2,y2)");

x1=g\_in(20,30);

Y1=g\_in (50,30);

x2=g\_in(20,50);

y2=g\_in(50,50);

shape=3;

closegraph(window);

setcurrentwindow(winmain);

plot\_points(x1,Y1,x1,y2,"A","C");

plot\_points(x1,y2,x2,y2,"C","D");

plot\_points(x2,y2,x2,Y1,"D","B");

plot\_points(x2,Y1,x1,Y1,"B","A");

break;

}

case 4:

{

closegraph(window);

setcurrentwindow(winmain);

break;

}

default:

{

goto loop;

}

}

}

int main()

{

int main\_window = initwindow(getmaxwidth(), getmaxheight(), "2D TRANSFORMATIONS");

TOP:

user\_interface(xc,yc);

int p\_xpoint=xc+pixeldiv, n\_xpoint=xc-pixeldiv; //X-axis: y constant

int p\_ypoint=yc-pixeldiv, n\_ypoint=yc+pixeldiv; //y-axis: x constant

for(int i=0; (p\_ypoint<=ymax+yc)&&(p\_xpoint<=xmax+xc); i++)

{

int color = RGB(40,40,40);

setcolor(color);

setlinestyle(DOTTED\_LINE,0,0);

if(n\_ypoint<=ymax+yc)

{

line(xc-xmax,p\_ypoint,xc+xmax,p\_ypoint); //Dotted line in positive Y axis parallel to x axis

line(xc-xmax,n\_ypoint,xc+xmax,n\_ypoint); //Dotted line in negative Y axis parallel to x axis

p\_ypoint-=pixeldiv;

n\_ypoint+=pixeldiv;

}

if(p\_xpoint<=xmax+xc)

{

line(p\_xpoint,yc-ymax,p\_xpoint,yc+ymax); //Dotted line in positive X axis parallel to Y axis

line(n\_xpoint,yc-ymax,n\_xpoint,yc+ymax); //Dotted line in negative X axis parallel to Y axis

p\_xpoint+=pixeldiv;

n\_xpoint-=pixeldiv;

}

}

int color = RGB(150, 150, 150);

setcolor(color);

setlinestyle(0,0,2);

line(xc,yc-ymax,xc,yc+ymax); //axis line

line(xc-xmax,yc,xc+xmax,yc); //axis line

loop:

//Choice and switch case for transformations

char a=getch();

int choice= a-'0';

switch(choice)

{

case 1:{

setcolor(RED);

reflection\_xaxis();

plot\_transformation();

goto loop;

}

case 2:{

setcolor(BLUE);

reflection\_xyaxis();

plot\_transformation();

goto loop;

}

case 3:{

setcolor(YELLOW);

reflection\_yaxis();

plot\_transformation();

goto loop;

}

case 4:{

setcolor(WHITE);

set\_object(main\_window);

goto loop;

}

case 5:{

exit(0);

}

case 0:{

x1=0,x2=0,x3=0,Y1=0,y2=0,y3=0,shape=0;

cleardevice();

goto TOP;

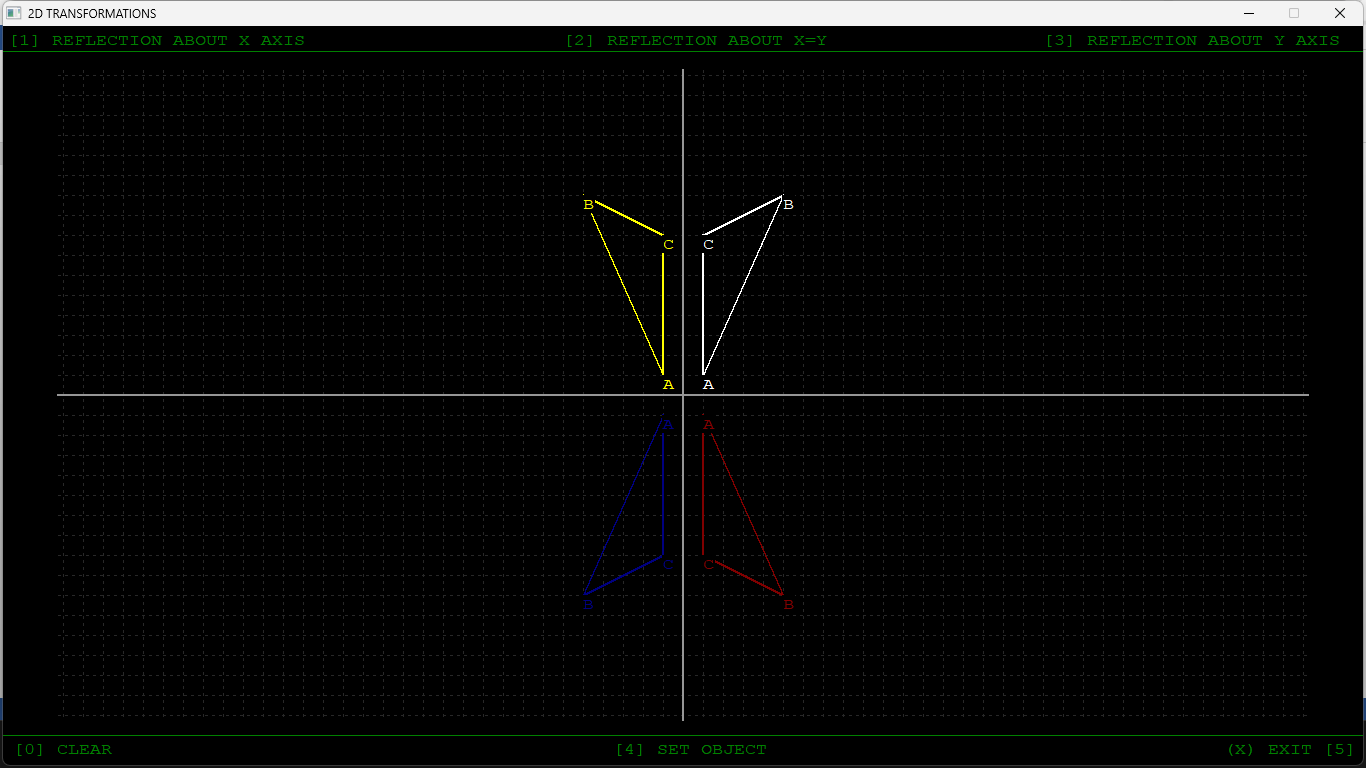
}

default:

goto loop;

}}

**Output:**

****

**CONCLUSION**:

In this way, we implement 2D transformations through writing code in C++ programming language and analysed its precision and way of calculation.