

**Date:**

---

**Aim:**

To write a C Program for Line Drawing using Digital Differential Analyzer (DDA) Method.

**Algorithm:**

1. Start
2. Read the two end points of a line (x1, y1) & (x2, y2).
3.  $\Delta x = x_2 - x_1$  &  $\Delta y = y_2 - y_1$
4. Initialize (x, y) with (x1, y1)
5.  $x = x_1$
6.  $y = y_1$
7. if  $\text{abs}(\Delta x) > \text{abs}(\Delta y)$  then
8.  $\text{steps} = \Delta x$
9. else
10.  $\text{steps} = \Delta y$
11.  $x \text{ increment} = \Delta x / \text{steps}$ .
12.  $y \text{ increment} = \Delta y / \text{steps}$
13. Plot the rounded coordinate (x, y)
14. Initialize counter k=1
15. Start the loop
  - $x = x + x \text{ increment}$
  - $y = y + y \text{ increment}$
  - Plot the rounded coordinate(x, y)
16. Continue the loop till the counter k= steps
17. Stop.

## Program:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void main()
{
    int gd = DETECT, gm;
    int x1, y1, x2, y2,dx,dy,steps,k;
    float xincrement,yincrement,x,y;
    initgraph(&gd, &gm, "..\\bgi");
    printf("Enter the Starting Point of x axis : ");
    scanf("%d", &x1);
    printf("Enter the Starting of y axis      : ");
    scanf("%d", &y1);
    printf("Enter the End Point of x axis    : ");
    scanf("%d", &x2);
    printf("Enter the End Point of y axis    : ");
    scanf("%d", &y2);
    clrscr();
    dx = x2 - x1;
    dy = y2 - y1;
    x=x1;
    y=y1;
    if(abs(dx) > abs(dy))
        steps=abs(dx);
    else
        steps=abs(dy);
    xincrement=dx/(float)steps;
    yincrement=dy/(float)steps;
    putpixel(ceil(x), ceil(y), RED);
    for(k=1;k<=steps;k++)
    {
        x=x+xincrement;
        y=y+yincrement;
        putpixel(ceil(x),ceil(y), RED);
    }
    getch();
    closegraph();
}
```

### Output:

```
Enter the Starting Point of x axis : 135  
Enter the Starting of y axis      : 135  
Enter the End Point of x axis    :267  
Enter the End Point of y axis    :267
```



### Result:

Thus, the C program to implement the DDA algorithm for line drawing was written, executed and the output was verified successfully.

**Ex.No: 1B**

## **Line Drawing Using Bresenham's Algorithm**

**DATE:**

---

**Aim:**

To write a C program to draw a line by using Bresenham's algorithm.

**Algorithm:**

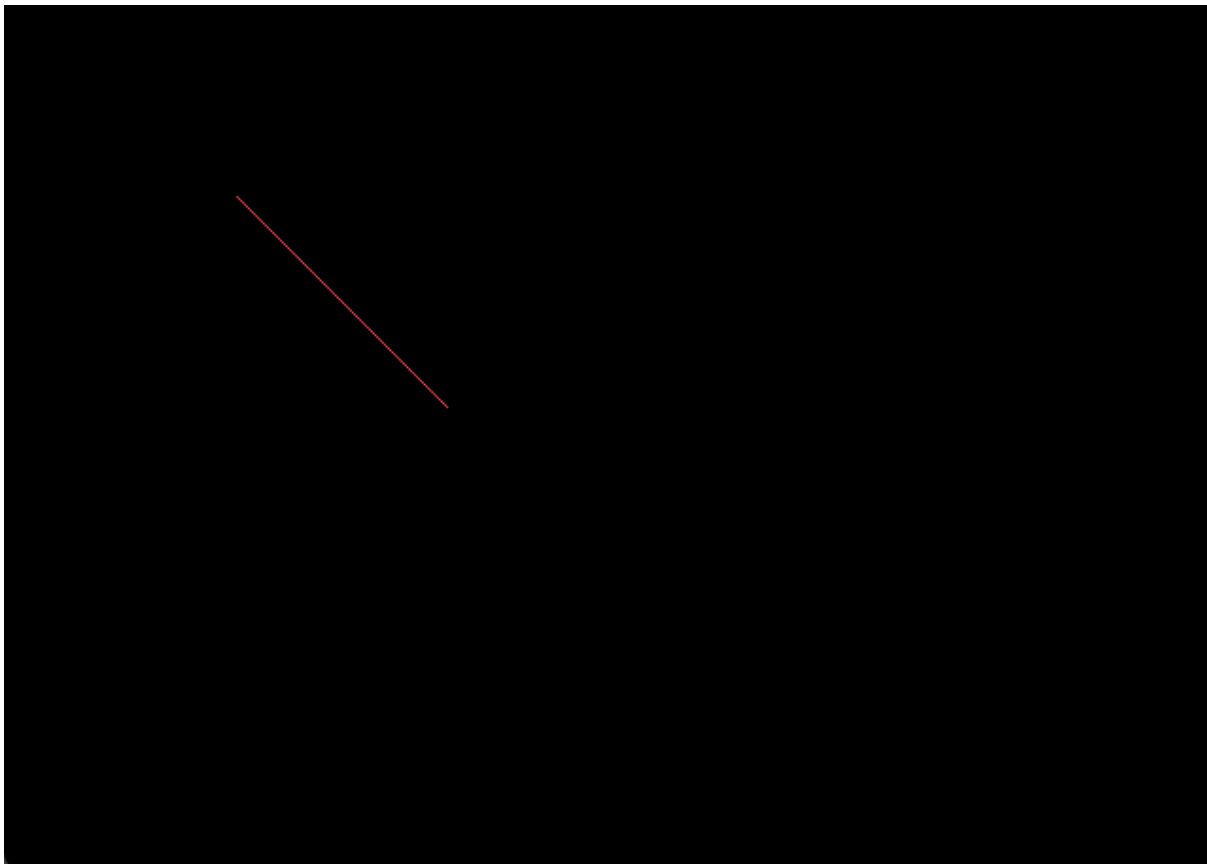
1. Start the program.
2. Input the starting and end points and find the dx and dy values.
3. Set the p value as  $p=2*(dy-dx)$ .
4. Check  $x_a > x_b$  then store  $(x_b, y_b)$  in  $(x, y)$  otherwise store  $(x_a, y_a)$  in  $(x, y)$ .
5. Put a pixel and check p value to set the value by the formula  $p=p+2*dy-dx$ .
6. Repeat the step 5 until  $x < x_{end}$ .
7. Display the line.
8. Stop the program.

**Program:**

```
#include<stdio.h>
#include<graphics.h>
void main()
{
    int dx,dy,x,y,xend,p,gd=DETECT,gm,xa,xb,ya,yb;
    printf("Line drawing using Bresenham's Algorithm:\n");
    printf("Enter the starting point and ending point:");
    scanf("%d%d%d%d",&xa,&ya,&xb,&yb);
    initgraph(&gd,&gm,"d:/tc/bgi");
    dx=abs(xa-xb);
    dy=abs(ya-yb);
    p=2*(dy-dx);
    if(xa>xb)
    {
        x=xb;
        y=yb;
        xend=xa;
    }
    else
    {
        x=xa;
        y=ya;
        xend=xb;
    }
    putpixel(x,y,12);
    do
    {
        if(p<0)
            p+=2*dy;
        else
        {
            y++;
            x++;
            p+=2*dy;
        }
        putpixel(x,y,12);
    }
    while(x<xend);
    getch();
    closegraph();
}
```

### Output:

```
C:\TURBOC3\BIN>TC C:\TURBOC3\Projects\expt1b.cpp
Line drawing using Bresenham's Algorithm:
Enter the starting point and ending point:123 123
234 234_
```



### Result:

Thus the C program to implement the Bresenham's algorithm for line drawing was written, executed and the output was verified successfully.

**DATE:**

---

**Aim:** To write a C Program for Circle Drawing using Midpoint Circle Method.

**Algorithm:**

1. Start
2. Get the center point (xcenter, ycenter) and radius(r) of a circle.
3. Initialize the variables
  - i.  $x=0$ ;  $y=r$ ;  $p=1-r$ ;
4. If ( $p < 0$ ) then
  - i.  $p=p+(2*x)+1$else
  - i.  $y--$ ;
  - ii.  $p=p+2*(x-y)+1$
5. Repeat step 4 until  $x < y$  and Increment x by 1 each time.
6. Plot the pixel to display the circle.
7. Display the circle
8. Stop.

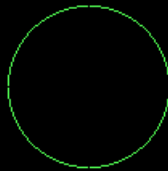
## Program:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
int x,y,r,p,xcenter, ycenter;
void circleplot(int,int,int,int);
void main()
{
int gd = DETECT, gm;
initgraph(&gd,&gm,"..\\BGI:");
printf("\nEnter the x-coordinate of the centre point :");
scanf("%d",&xcenter);
printf("\nEnter the y-coordinate of the centre point :");
scanf("%d",&ycenter);
printf("\nEnter the radius of a circle:");
scanf("%d",&r);
x=0;
y=r;
p=1-r;
while (x<y)
{
x++;
if (p<0)
p=p+2*x+1;
else
{
y--;
p=p+2*(x-y)+1;
}
circleplot(xcenter,ycenter,x,y);
}
getch();
closegraph();
}
void circleplot(int xcenter, int ycenter,int x, int y)
{
putpixel(xcenter+x,ycenter+y,10);
putpixel(xcenter-x,ycenter+y,10);
putpixel(xcenter+x,ycenter-y,10);
putpixel(xcenter-x,ycenter-y,10);
putpixel(xcenter+y,ycenter+x,10);
putpixel(xcenter-y,ycenter+x,10);
putpixel(xcenter+y,ycenter-x,10);
putpixel(xcenter-y,ycenter-x,10);
}
```



### Output:

```
Enter the x-coordinate of the centre point :250  
Enter the y-coordinate of the centre point :340  
Enter the radius of a circle:50
```



### Result:

Thus the C program to implement the Mid point circle drawing was written, executed and the output was verified successfully

**Ex.No: 2A**

**2D Transformations – Translation.**

**DATE:**

---

**Aim:**

To write a C program for implementing the two dimensional transformation such as Translation.

**Algorithm:**

1. Start the program.
2. Input the no of vertices and their corresponding co-ordinates.
3. Display the input figure by using the draw() function.
4. Input the Transformation factor for both x and y , then also display the figure before transformation.
5. By using these co-ordinates , we can change the position of the figure.
6. Clear the screen and then display the figure after the transformation.
7. Terminate the program.

**Program:**

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
int n,a[25],a1[25],i;
void draw(int *,int);
void main()
{
    int gm,gd=DETECT;
    int tpx,tpy;
    initgraph(&gd,&gm,"c:/tc/bgi");
    cleardevice();
    printf("Enter the no of vertices\n");

    scanf("%d",&n);
    printf("Enter the co_ordinates\n");
    for(i=0;i<n*2;i++)
        scanf("%d",&a[i]);
    outtextxy(50,50,"The input figure is");
    draw(a,4);
    clrscr();
    cleardevice();
    printf("Enter the transformation factor x and y\n");
    scanf("%d%d",&tpx,&tpy);
    for(i=0;i<n*2;i+=2)
        a1[i]=a[i]+tpx;
    for(i=1;i<n*2;i+=2)
        a1[i]=a[i]+tpy;
    cleardevice();
    outtextxy(50,50,"Before transformation");
    draw(a,2);
    outtextxy(50,50,"After transformation");
    draw(a1,4);
    getch();
}
void draw(int *a,int col)
{
    setfillstyle(BKSLASH_FILL,col);
    fillpoly(n,a);
    getch();
    cleardevice();}
```

## Output:

```
Enter the no of vertices
4
Enter the co_ordinates
150 150 the input figure is
250 150
150 250
250 250
```



```
Enter the transformation factor x and y
150 200
```

Before transformation



**Result:**

Thus, the C program to implement the two dimensional translation was written, executed and the output was verified successfully.

**DATE:**

---

**Aim:**

To write a C program for implementing the two dimensional transformation such as Rotation.

**Algorithm:**

1. Start the program.
2. Declare the variables and initiate the initgraph() method.
3. Get the no of vertices and their corresponding co-ordinates.
4. Using them, display the input figure then get the rotation angle and fixed point of rotation.
5. Depending upon the fixed point, rotate the figure by changing the co-ordinates of the vertex.
6. Display the given figure, before rotation and also after rotation.
7. Terminate the program.

## Program:

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
int a[25],a1[25],i,n;
void draw(int *,int);
void main()
{
    double angle;
    float pi;
    int px,py,t,j;
    int gm,gd=DETECT;
    initgraph(&gd,&gm,"c:/tc/bgi");
    printf("\t\n**Rotation***");
    printf("Enter the no of vertices:");
    scanf("%d",&n);
    printf("Enter the coordinates:");
    for(i=0;i<n*2;i++)
        scanf("%d",&a[i]);
    outtextxy(50,50,"Input figure");
    draw(a,4);
    pi=3.14/180;
    printf("Enter the rotation angle:");
    scanf("%d",&t);
    printf("Enter the rotation point:");
    scanf("%d%d",&px,&py);
    j=0;
    angle=(double)(t*pi);
    for(i=0;i<n*2;i+=2)
    {
        a1[j]=px+((a[i]-px)*cos(angle))-((a[i+1]-py)*sin(angle));
        a1[++j]=py+((a[i]-px)*sin(angle))-((a[i+1]-py)*cos(angle));
        j++;
    }
    cleardevice();
    outtextxy(50,50,"Before Rotation:");
    circle(px,py,2);
    line(px,py,px,0);
    line(px,py,getmaxx(),py);
    line(px,py,px,getmaxy());
    outtextxy(px,py,"axis");
    draw(a,2);
    outtextxy(300,50,"After Rotation");
    circle(px,py,2);
    line(px,py,0,py);
    line(px,py,getmaxx(),py);
    line(px,py,px,getmaxy());
    outtextxy(px,py,"axis");
    draw(a1,4);
}
void draw(int *a,int col)
```

```
{  
  setfillstyle(BKSLASH_FILL,col);  
  fillpoly(n,a);  
  getch();  
  cleardevice();  
}
```



## Output:

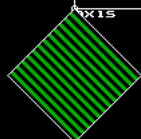
```
**Rotation**Enter the no of vertices:4
Enter the coordinates:
Input figure
150 150
200 200
150 250
100 200
```

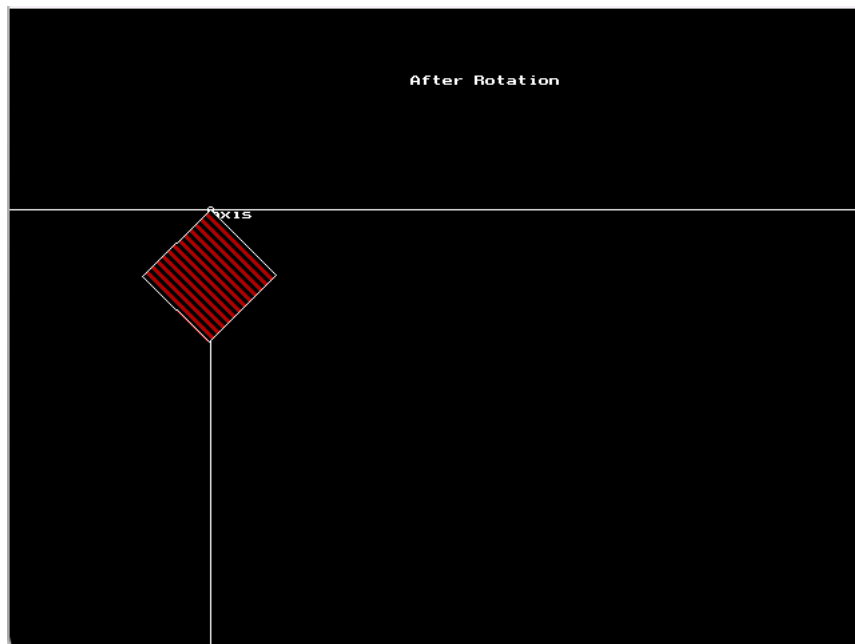


```
Enter the rotation angle:180
Enter the rotation point:150 150
```

Before Rotation:

axis





**Result:**

Thus the C program to implement the two dimensional rotation was written, executed and the output was verified successfully.

**Ex.No.: 2C**

## **2D Transformations – Scaling**

**DATE:**

---

**Aim:**

To write a C program for implementing the two dimensional transformation such as Scaling.

**Algorithm:**

1. Start the program.
2. Get the number of vertices and their corresponding co-ordinates of the vertices.
3. Display the input figure by using the co-ordinates.
4. Cut the Scaling factor and multiply this with co-ordinates.
5. Display the Zooming figure.
6. Terminate the program.

**Program:**

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
int n,a[25],a1[25],i,k;
void draw(int *,int);
void main()
{
    int gm,gd=DETECT;
    int tpx,tpy;
    initgraph(&gd,&gm,"c:/tc/bgi");
    cleardevice();
    printf("***SCALLING**8\n");
    printf("Enter the no of vertices:\n");
    scanf("%d",&n);
    printf("Enter the coordinates:\n");
    for(i=0;i<n*2;i++)
        scanf("%d",&a[i]);
    clrscr();
    cleardevice();
    outtextxy(50,50,"The input figure is");
    draw(a,4);
    clrscr();
    cleardevice();
    printf("Enter the value to zoom:\n");
    scanf("%d",&k);
    for(i=0;i<n*2;i+=2)
        a1[i]=a[i]*k;
    for(i=1;i<n*2;i+=2)
        a1[i]=a[i]*k;
    outtextxy(50,50,"after zooming");
    gotoxy(10,10);
    draw(a1,4);
}
void draw(int *a,
int col)
{
    setfillstyle(SLASH_FILL,col);
    fillpoly(n,a);
    getch();
    cleardevice();
}
```

## Output:

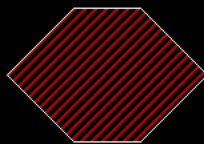
```
***SCALING**  
Enter the no of vertices:  
6  
Enter the coordinates:  
150 150 175 150 200 175 175 200 150 200 125 175
```

The input figure is



```
Enter the value to zoom:  
2
```

after zooming



## Result:

Thus, the C program to implement the two-dimensional scaling was written, executed and the output was verified successfully.

**Ex.No: 2D**

## **2D Transformations – Shearing & Reflection**

**DATE:**

---

**Aim:**

To write a C program for implementing the two dimensional transformation such as Shearing & Reflection.

**Algorithm:**

1. Start the program.
2. Initialize the graphics drive and mode.
3. Read the co-ordinates of 2D object, which is transformed.
4. For shearing read the shearing factor and apply the shearing factor formula with respect to x & y.
  - i)  $x' = x + shx * y$   
 $y' = y$
  - ii)  $y' = y + shy * x$   
 $x' = x$
5. Stop the program.

**Program:**

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<string.h>
#include<stdlib.h>
void main()
{
    int gd=DETECT,gm,midx,midy,midw,n;
    int xs1,xs2,xs3,ys1,ys2,ys3,ch,x1,y1,x2,y2,x3,y3,abs,c,b,w,z,sh;
    initgraph(&gd,&gm,"c:/tc/bgi");
    printf("\nEnter the I co-ordinate***\n");
    scanf("%d%d",&x1,&y1);
    printf("\nEnter the II coordinate***\n");
    scanf("%d%d",&x2,&y2);
    midx=abs((x2-x1)/2);
    midy=abs((y2-y1)/2);
    cleardevice();
    do
    {
        printf("1.Shearing\n2.Reflection\n3.Quit\n");
        scanf("%d",&n);
        switch(n)
        {
            case 1:
                cleardevice();
                line(x1,y1,x2,y2);
                moveto(x2,y2);
                lineto(x1+midx,y1+midy);
                lineto(x1,y1);
                printf("Enter the shearing factor\n");
                scanf("%d",&sh);
                printf("Enter the choice 1 with respect to x,2 with respect to y\n");
                scanf("%d",&ch);
                printf("%d",ch);
                if(ch==1)
                {
                    xs1=x1+sh*y1;
                    xs2=x2+sh*y2;
                    line(x1,y1,xs2,y2);
                    moveto(xs2,y2);
                    lineto(xs1+abs((xs2-xs1)/2),y1+abs((y2-y1)/2));
                    lineto(x1,y1);
                }
                else
                {
                    ys1=y1+sh*x1;
                    ys2=y2+sh*x2;
                    line(x1,y1,x2,ys2);
                    moveto(x2,ys2);
                    lineto(x1+abs((x2-x1)/2),ys1+abs((ys2-ys1)/2));
                }
            }
        }
    }
```

```
    lineto(x1,y1);
}
getch();
break;
case 2:
    cleardevice();
    line(x1,y1,x2,y2);
    moveto(x2,y2);
    lineto(x1+midx,y1+midy);
    lineto(x1,y1);
    if(y1>y2)
        z=y1;
    else
        z=y2;
    if(x1<x2)
        w=x2;
    else
        w=x1;
    line(0,z+10,w+50,z+10);
    line(x1,z+20+(z-y1),x2,z+20+(z-y2));
    moveto(x2,z+20+(z-y2));
    lineto(x1+midx,z+20+(z-(y1+midy)));
    lineto(x1,z+20+(z-y1));
    getch();
    break;
case 3:
    exit(0);
    break;
}}
while(n<=3);
getch();
}
```



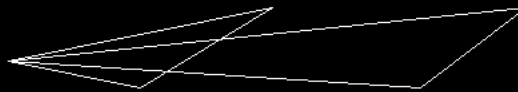
## Output:

```
Enter the I co-ordinate***
170 230

Enter the II coordinate***
368 190
```

```
1.Shearing
2.Reflection
3.Quit
1
```

```
Enter the shearing factor
```



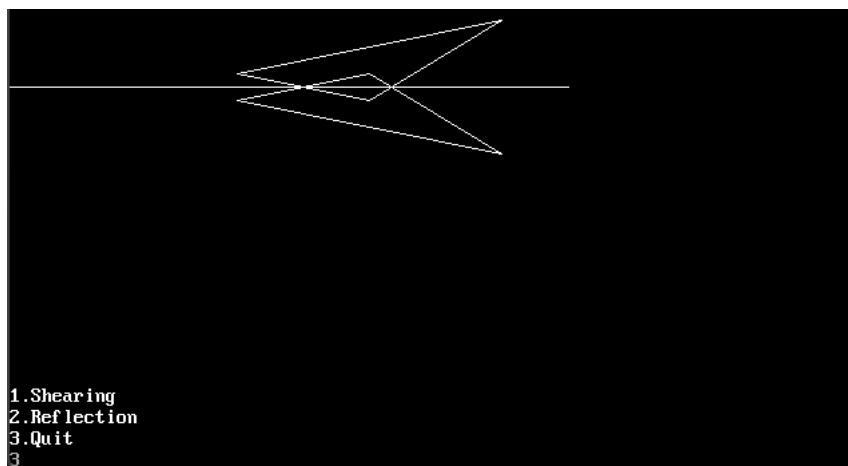
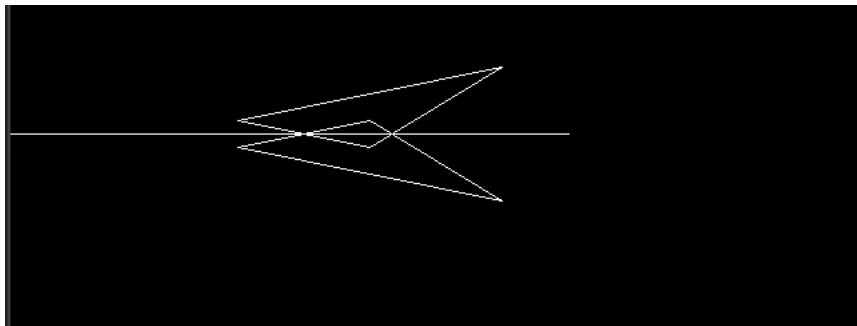
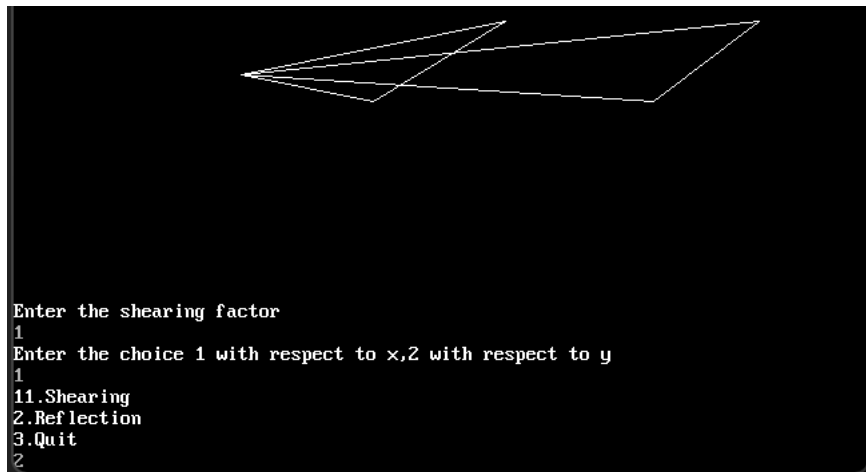
```
1
Enter the choice 1 with respect to x,2 with respect to y
1
11.Shearing
2.Reflection
3.Quit
1
```



```
Enter the shearing factor
1
Enter the choice 1 with respect to x,2 with respect to y
1
```



```
Enter the shearing factor
1
Enter the choice 1 with respect to x,2 with respect to y
1
1
```



### Result:

Thus the C program to implement the 2D reflection and shearing was written, executed and the output was verified successfully.

**DATE:**

---

**Aim:**

To write a C program for implementing Window to viewport Translation.

**Steps:**

1. Input the minimum and maximum coordinates of a window.
2. Input the minimum and maximum coordinates of a view port.
3. Get the coordinates of a point.
4. Display a point using set pixel function.

**Program:**

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void main()
{
int gm,gr,xwmin,ywmin,xwmax,
ywmax,xvmin,yvmin,xvmax,yvmax,xw,yw,xv,yv,sx,sy;
clrscr();
detectgraph(&gm,&gr);
initgraph(&gm,&gr,"d:\\tc\\BGI");
printf("Enter the window min coordinate(xwmin,ywmin):\n");
scanf("%d%d",&xwmin,&ywmin);
printf("Enter the window max coordinate(xwmax,ywmax):\n");
scanf("%d%d",&xwmax,&ywmax);
rectangle(xwmin,ywmax,xwmax,ywmin);
printf("Enter the viewport min coordinate(xvmin,yvmin):\n");
scanf("%d%d",&xvmin,&yvmin);
printf("Enter the viewport max coordinate(xvmax,yvmax):\n");
scanf("%d%d",&xvmax,&yvmax);
sx=(xvmax-xvmin)/(xwmax-xwmin);
sy=(yvmax-yvmin)/(ywmax-ywmin);
rectangle(xvmin,yvmax,xvmax,yvmin);
printf("Enter the point coordinate(xw,yw) in the window:\n");
scanf("%d%d",&xw,&yw);
putpixel(xw,yw,15);
xv=(sx*(xw-xwmin))+xvmin;
yv=(sy*(yw-ywmin))+yvmin;
putpixel(xv,yv,15);
getch();
}
```

### Output:

```
Enter the window min coordinate(xwmin,ywmin):  
150 150  
Enter the window max coordinate(xwmax,ywmax):  
200 200  
Enter the viewport min coordinate(xvmin,yvmin):  
320 320  
Enter the viewport max coordinate(xvmax,yvmax):  
330 330  
Enter the point coordinate(xw,yw) in the window:  
390 390
```



### Result:

Thus the c program to implement window to view port mapping was coded and executed successfully.

**Ex.No: 3**

## **Composite 2D Transformations**

**DATE:**

---

**Aim:**

To write a C program to implement 2D transformations.

**Algorithm:**

1. Enter the choice for transformation.
2. Perform the translation, rotation, scaling, reflection and shearing of 2D object.
3. Get the needed parameters for the transformation from the user.
4. Incase of rotation, object can be rotated about x or y axis.
5. Display the transmitted object in the screen.

**Program:**

```
#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<dos.h>

#include<math.h>

#include<stdlib.h>

void input();

void output();

void translation();

void rotation();

void scaling();

int a[10][2],i,x,temp,y,n,fx,fy,opt,ch,j=1;

int sx,sy;

int tx,ty;

float k;

void input()

{

printf("\t\t\tTranslation");

printf("\n\nEnter the number of lines: ");

scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("\n\nEnter the coordinates for the Line: ");

scanf("%d%d%d%d",&a[i][0],&a[i][1],&a[i+1][0],&a[i+1][1]);

}

}

void output()
```

```

{
for(i=0;i<n;i++)

{
line(a[i][0],a[i][1],a[i+1][0],a[i+1][1]); }

}

void translation()

{
printf("enter the tranformation vertex tx,ty:\n");
scanf("%d%d",&tx,&ty);

cleardevice();

output();

for(i=0;i<=n;i++)

{
a[i][0]=a[i][0]+tx;
a[i][1]=a[i][1]+ty;
}

setlinestyle(2,1,1);

output();

setlinestyle(0,1,1);

line(75,460,125,460);

outtextxy(130,460,"Before translation");

setlinestyle(2,1,1);

line(300,460,350,460);

outtextxy(355,460,"After translation");

outtextxy(520,460,"S.S.RENJINI");

delay(100);

}

Void rotation()

```

```

{
printf("\nEnter the rotating angle: ");
scanf("%d",&y);
printf("\nEnter the pivot point:");
scanf("%d%d",&fx,&fy);

cleardevice();

output();

k=(y*3.14)/180;

for(i=0;i<=n;i++)
{
tx=fx+(a[i][0]-fx)*cos(k)-(a[i][1]-fy)*sin(k);
ty=fy+(a[i][0]-fx)*sin(k)+(a[i][1]-fy)*cos(k);
a[i][0]=tx;
a[i][1]=ty;
}

setlinestyle(2,1,1);

output();

setlinestyle(0,1,1);

line(75,460,125,460);

outtextxy(130,460,"Before rotation");

setlinestyle(2,1,1);

line(300,460,350,460);

outtextxy(355,460,"After rotation");

outtextxy(520,460,"SUNDAR");

delay(100);

}

void scaling()
{

```



```
printf("\nEnter the scaling factor: ");

scanf("%d%d",&sx,&sy);

cleardevice();

output();

for(i=0;i<=n;i++)

{

a[i][0]=a[i][0]*sx;

a[i][1]=a[i][1]*sy;

}

setlinestyle(2,1,1);

output();

setlinestyle(0,1,1);

line(75,460,125,460);

outtextxy(130,460,"Before scaling");

setlinestyle(2,1,1);

line(300,460,350,460);

outtextxy(355,460,"After scaling");

outtextxy(520,460,"S.S.RENJNI");

delay(100);

}

void main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"D:\\TC\\bgi");

input();

output();

do

{
```

```

printf("\nWhat Do you want to do with current object\n");

printf("\n1:Translation\n2:Rotation\n3:Scaling\n");

printf("\nEnter your Choice: ");

scanf("%d",&ch);

switch(ch)

{

case 1:translation();

break;

case 2:rotation();

break;

case 3:

scaling();

break;

}

delay(6000);

closegraph();

initgraph(&gd,&gm,"D:\\TC\\bgi");

setlinestyle(0,1,1);

printf("To continue press 1 & to EXIT press 0:");

scanf("%d",&j);

if(j==1)

{

printf("\nThe current object has the line coordinates:");

for(i=0;i<n;i++)

{

printf("\n");


printf("For line %d:",i+1);

printf("%d %d %d %d",a[i][0],a[i][1],a[i+1][0],a[i+1][1]);

```

```
}  
  
}  
  
}while(j==1);  
  
getch();  
  
}
```

### Output:

```
Translation  
Enter the number of lines: 3  
  
Enter the coordinates for the Line: 100  
100  
200  
100  
      
  
Enter the coordinates for the Line: 200  
100  
150  
75  
  
Enter the coordinates for the Line: 150  
75  
100  
100  
  
What Do you want to do with current object  
  
1:Translation  
2:Rotation  
3:Scaling  
  
Enter your Choice: 1  
enter the tranformation vertex tx,ty:  
150  
75
```



To continue press 1 & to EXIT press 0:1

The current object has the line coordinates:

For line 1:250 175 350 175

For line 2:350 175 300 150

For line 3:300 150 250 175

What Do you want to do with current object

1:Translation

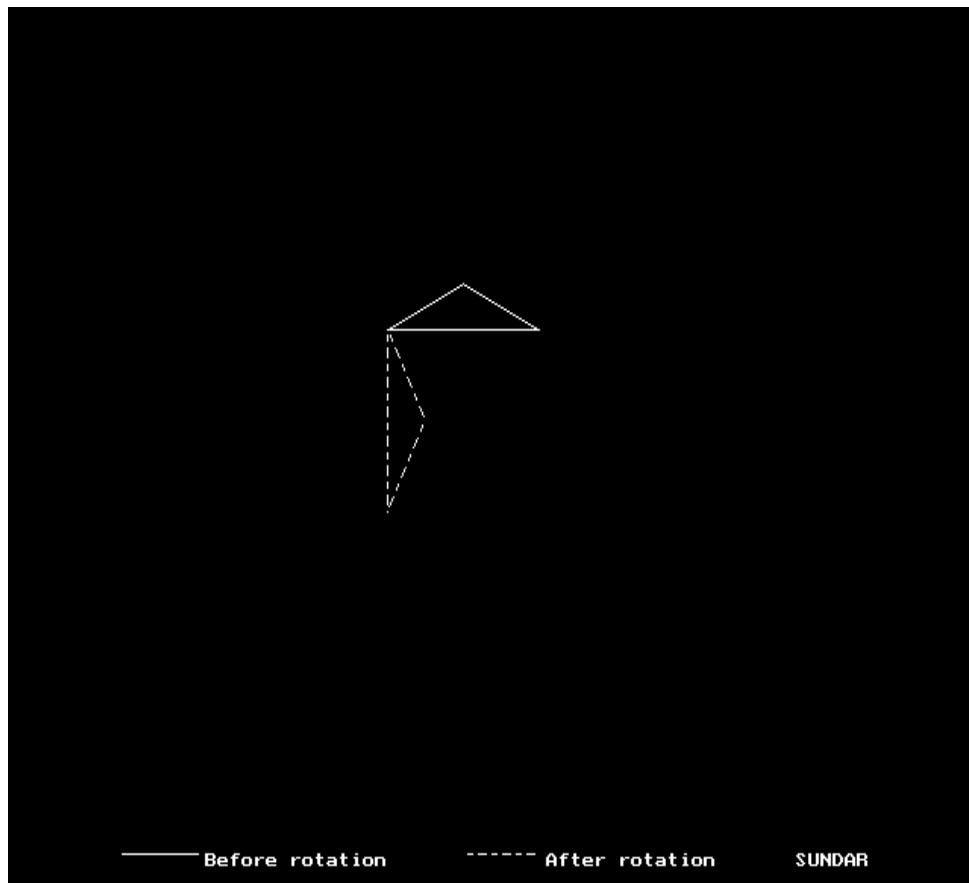
2:Rotation

3:Scaling

Enter your Choice: 2

Enter the rotating angle: 90

Enter the pivot point:250  
175



To continue press 1 & to EXIT press 0:1

The current object has the line coordinates:

For line 1:250 175 250 274

For line 2:250 274 275 224

For line 3:275 224 250 175

What Do you want to do with current object

1:Translation

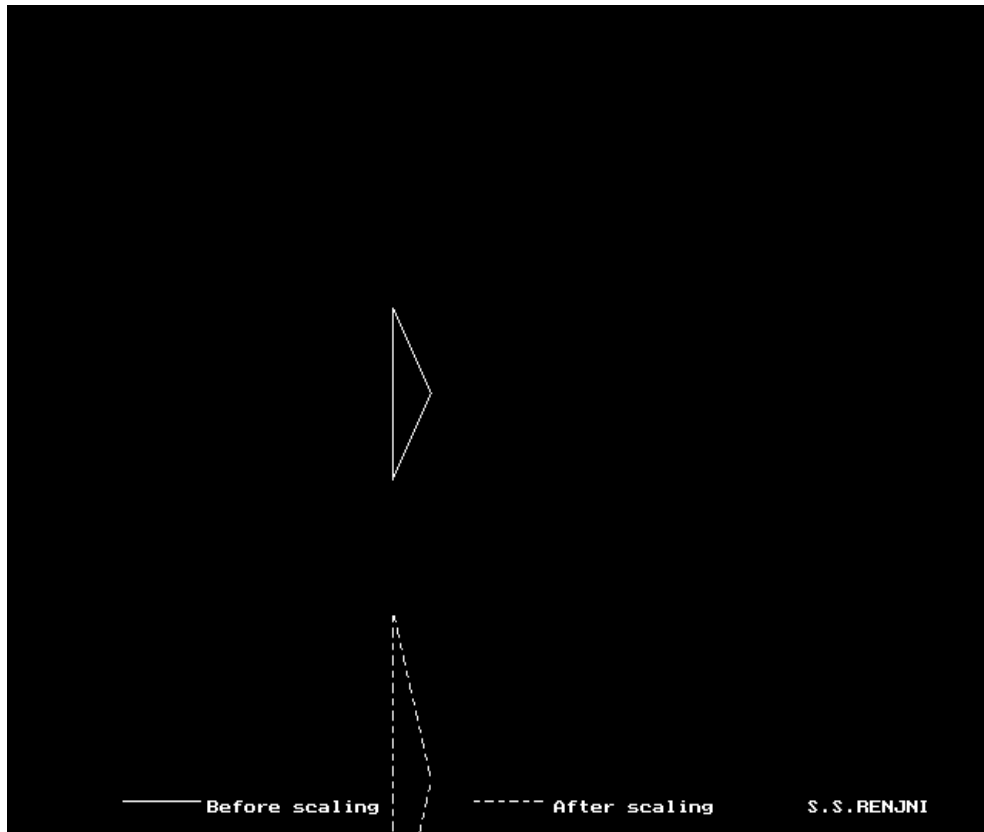
2:Rotation

3:Scaling

Enter your Choice: 3

Enter the scaling factor: 1

2



To continue press 1 & to EXIT press 0:0

### Result:

Thus the C program to implement the Composite 2D-Transformations was written, executed and the output was verified successfully.sow

**Ex.No: 4**

**Cohen Sutherland 2D line clipping and Windowing**

**DATE:**

---

**Aim:**

To write a C program to implement the Cohen-Sutherland 2D line clipping.

**Algorithm:**

1. Start the program.
2. Read two end points of the line say  $p1(x1,y1)$  and  $p2(x2,y2)$
3. Read two corners (left-top and right-bottom) of the window, say  $(Wx1,Wy1)$  and  $(Wx2,Wy2)$
4. Assign the region codes for two endpoints  $p1$  and  $p2$  using following steps:

Initialize code with bits 0000

Set Bit 1-if( $x < Wx1$ )

Set Bit 2-if( $x > Wx2$ )

Set Bit 3-if( $y < Wy2$ )

Set Bit 4-if( $y > Wy1$ )

5. Check for visibility of line  $p1p2$

a) If region codes for both endpoints  $p1$  and  $p2$  are zero then the line is completely visible. Hence draw the line and go to 10.

b) If region codes for endpoints are not zero and the logical ANDing of them is also nonzero then the line is completely invisible, so reject the line and go to 10.

c) If region codes for two endpoints do not satisfy the conditions in 4a & 4b the line is partially visible.

6. Determine the intersecting edge of the clipping window by inspecting the region codes of two endpoints.

a) If region codes for both the endpoints are non-zero, find intersection points  $p1'$  and  $p2'$  with boundary edges of clipping window with respect to point  $p1$  and point  $p2$ , respectively.

b) If region codes for any one endpoint is non-zero, find intersection points  $p1'$  and  $p2'$  with boundary edge of clipping window with respect to it.

7. Divide the line segments considering intersection points.

8. Reject the line segments if any one end point of it appears outside the clipping window.

9. Draw the remaining line segments.

10. Stop the program.

**Program:**

```
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<dos.h>

#include<math.h>

#include<graphics.h>

typedef struct coordinate

{

int x,y;

char code[4];

}pt;

void drawwindow();

void drawline(pt p1,pt p2,int c1);

pt setcode(pt p);

int visibility(pt p1,pt p2);

pt resetendpt(pt p1,pt p2);

main()

{

int gd=DETECT,gm,v;

pt p1,p2,ptemp;

initgraph(&gd,&gm,"D:/tc/bgi");

cleardevice();

printf("\n\n Enter the End-point1(x,y):");

scanf("%d%d",&p1.x,&p1.y);

printf("\n\n Enter the End-point2(x,y):");

scanf("%d%d",&p2.x,&p2.y);

cleardevice();
```



```
drawwindow();

getch();

drawline(p1,p2,15);

getch();

p1=setcode(p1);

p2=setcode(p2);

v=visibility(p1,p2);

switch(v)

{

case 0:

cleardevice();

drawwindow();

drawline(p1,p2,15);

break;

case 1:

cleardevice();

drawwindow();

break;

case 2:

cleardevice();

p1=resetendpt(p1,p2);

p2=resetendpt(p2,p1);

drawwindow();

drawline(p1,p2,15);

break;

}

getch();

closegraph();
```

```
return(0);

}

void drawwindow()

{

setcolor(RED);

line(150,100,450,100);

line(450,100,450,350);

line(450,350,150,350);

line(150,350,150,100);

}

void drawline(pt p1,pt p2,int c1)

{

setcolor(c1);

line(p1.x,p1.y,p2.x,p2.y);

}

pt setcode(pt p)

{

pt ptemp;

if(p.y<100)

ptemp.code[0]='1';

else

ptemp.code[0]='0';

if(p.y>350)

ptemp.code[1]='1';

else

ptemp.code[1]='0';

if(p.x>450)

ptemp.code[2]='1';
```

```
else

ptemp.code[2]='0';

if(p.x<150)

ptemp.code[3]='1';

else

ptemp.code[3]='0';

ptemp.x=p.x;

ptemp.y=p.y;

return(ptemp);

}

int visibility(pt p1,pt p2)

{

int i,flag=0;

for(i=0;i<4;i++)

{

if((p1.code[i]!='\0')||(p2.code[i]!='0'))

flag=1;

}

if(flag==0)

return 1;

return 2;

}

pt resetendpt(pt p1,pt p2)

{

pt temp;

int x,y,i;

float m,k;

if(p1.code[3]=='1');
```

```

x=150;

if(p1.code[2]=='1')

x=450;

if((p1.code[3]=='1')||(p1.code[2]=='1'))

{

m=(float)(p2.y-p1.y)/(p2.x-p1.x);

k=(p1.y+(m*(x-p1.x)));

temp.y=k;

temp.x=x;

for(i=0;i<4;i++)

temp.code[i]=p1.code[i];

if(temp.y<=350 && temp.y>=100)

return(temp);

}

if(p1.code[0]=='1')

y=100;

if(p1.code[1]=='1')

y=350;

if((p1.code[0]=='1')||(p1.code[1]=='1'))

{

m=(float)(p2.y-p1.y)/(p2.x-p1.x);

k=(float)p1.x+(float)(y-p1.y)/m;

temp.x=k;

temp.y=y;

for(i=0;i<4;i++)

temp.code[i]=p1.code[i];

return(temp);

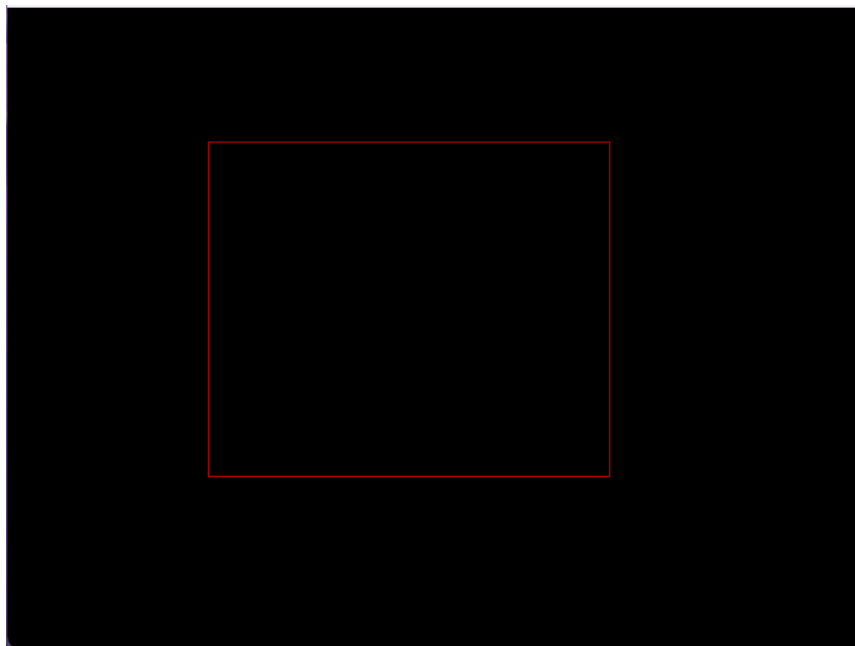
}

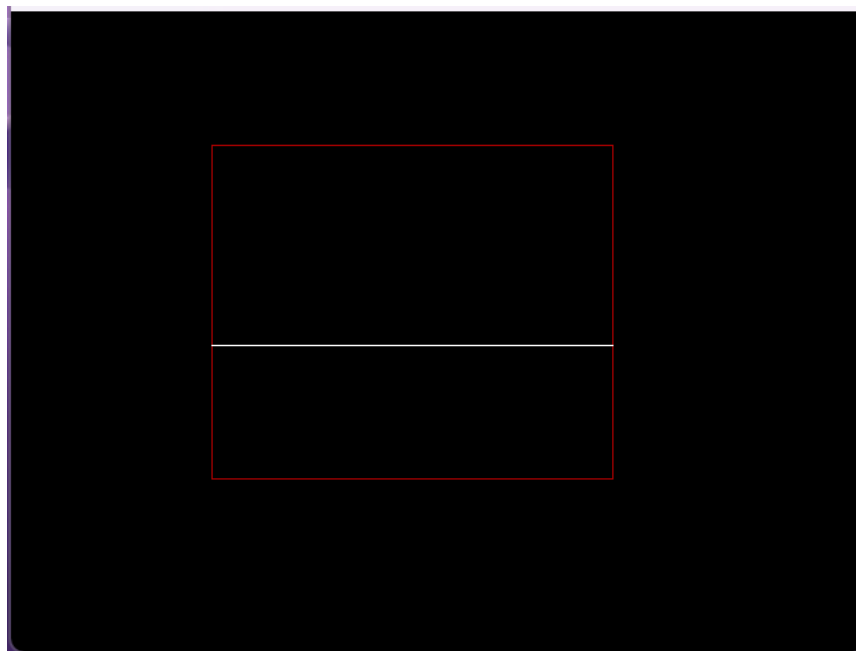
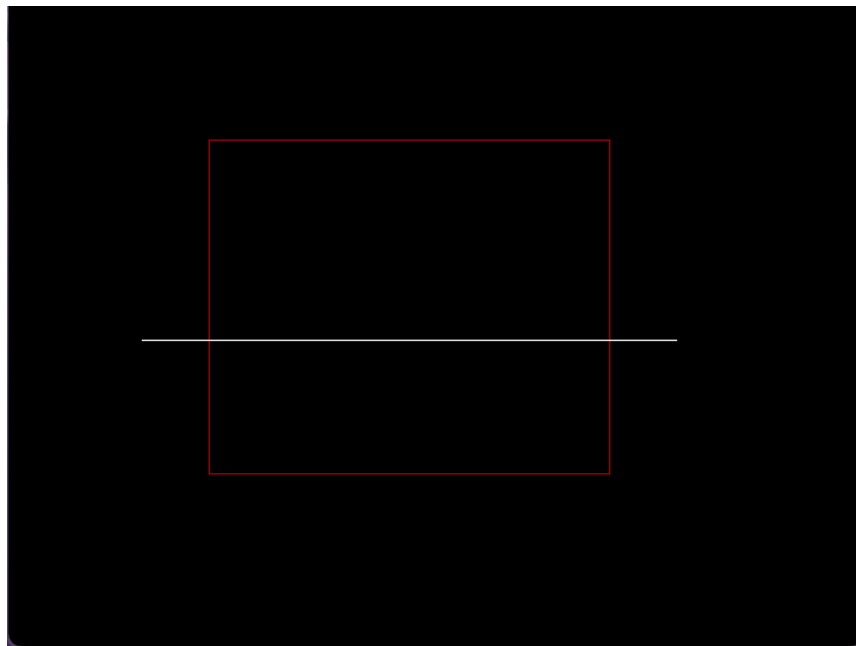
```

```
else return(p1);  
}
```

**Output:**

```
Enter the End-point1(x,y):100 250  
Enter the End-point2(x,y):500 250
```





**Result:**

Thus the C program to implement the line clipping was written, executed and the output was verified successfully.

**Ex.No: 5**

**3D transformations - Translation, Rotation, Scaling**

**DATE:**

---

**Aim:**

To write a C program to implement three dimensional transformations such as Translation, Rotation and Scaling.

**Algorithm:**

1. Start the program.
2. Draw the 3D object and also display the choices.
3. Get the choice and perform the corresponding actions.
4. The choices is '1' then get the translation values and move the 3D-object.
5. The choices is '2' then get the scaling vectors and change the size of 3D-object.
6. The choices is '3' then get the rotation angle to rotate the 3D-object.
7. The choices is '4' then exit from the output window.
8. Terminate the program.

**Program:**

```
#include<stdio.h>

#include<graphics.h>

#include<dos.h>

#include<conio.h>

#include<math.h>

#include<process.h>

int points[50];

int points1[50];

int x=25;

int n=5;

void construct(int[]);

void construct1(int[],int[]);
```

```
void main()

{
int gd=DETECT;
int gm=DETECT;
char z;

initgraph(&gm,&gd,"D:/tc/bgi");

do
{
int points[50]={ 100,100,100,150,150,150,150,100,100,100};
int a;
cleardevice();

printf("\n\n\t\t\t1.Translation");
printf("\n\n\t\t\t2.Scaling");
printf("\n\n\t\t\t3.Rotation");
printf("\n\n\t\t\t4.Exit");
printf("\n\n\tEnter the choice:");

scanf("%d",&a);

switch(a)
{
case 1:
{
int tx,ty,i,point[50],point1[50];

construct(points);

printf("Enter the Translation Values:\n");

scanf("%d%d",&tx,&ty);

for(i=0;i<=2*n;i=i+2)
{
point[i]=points[i]+tx;
```



```
point1[i]=points1[i]+tx;
}
for(i=1;i<2*n;i=i+2)
{
point[i]=points[i]+ty;
point1[i]=points1[i]+ty;
}
construct1(point,point1);
getch();
break;
}
case 2:
{
int sx,sy,sz,point[50],point1[50],i;
construct(points);
printf("Enter the scaling values:\n");
scanf("%d%d%d",&sx,&sy,&sz);
point[0]=points[0];
point[1]=points[1];
point[2]=points[2];
point[3]=points[1]+sy*(points[3]-points[1]);
point[4]=points[2]+sx*(points[4]-points[2]);
point[5]=points[7]+sy*(points[5]-points[7]);
point[6]=points[0]+sx*(points[6]-points[0]);
point[7]=points[7];
point[8]=points[8];
point[9]=points[9];
for(i=0;i<2*n;i++)
```

```

{
point1[i]=point[i]+(x*sz);
}

construct1(point,point1);

getch();

break;
}

case 3:

{
int point[50],point1[50],i;

float an;

construct(points);

printf("Enter the angle:\n");

scanf("%f",&an);

an=(float)an*3.14/180;

for(i=0;i<=8;i+=2)

{

point[i]=points[0]+(points[i]-points[0])*cos(an)-(points[i+1]-points[1])*sin(an);

}

for(i=1;i<=9;i+=2)

{

point[i]=points[1]+(points[i]-points[1])*cos(an)-(points[i-1]-points[0])*sin(an);

}

for(i=0;i<=9;i+=2)

{

point1[i]=points1[0]+(points1[i]-points1[0])*cos(an)-(points1[i+1]-points1[1])*sin(an);

}

for(i=1;i<=9;i+=2)

```

```

{
point1[i]=points1[1]+(points1[i]-points1[1])*cos(an)-(points1[i-1]-points1[0])*sin(an);
}
construct1(point,point1);
getch();
break;
}
case 4:
{
exit(0);
break;
}
}
printf("Do you want to continue say(Y/N)?:"");
scanf("%s",&z);
}while(z=='Y');
getch();
closegraph();
}
void construct(int points[50])
{
int x=25,n=5,i;
cleardevice();
for(i=0;i<2*n;i++)
{
points1[i]=points[i]+x;
}
construct1(points,points1);

```

```

}

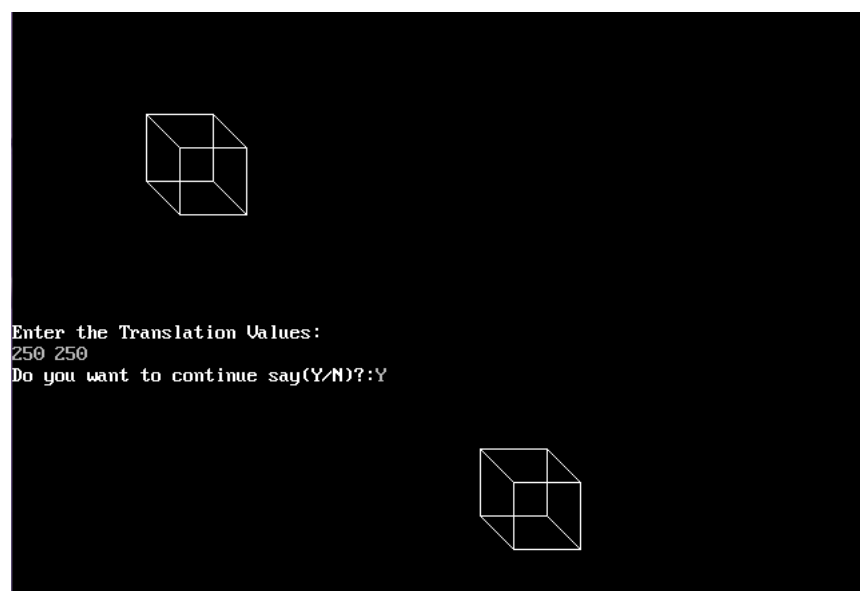
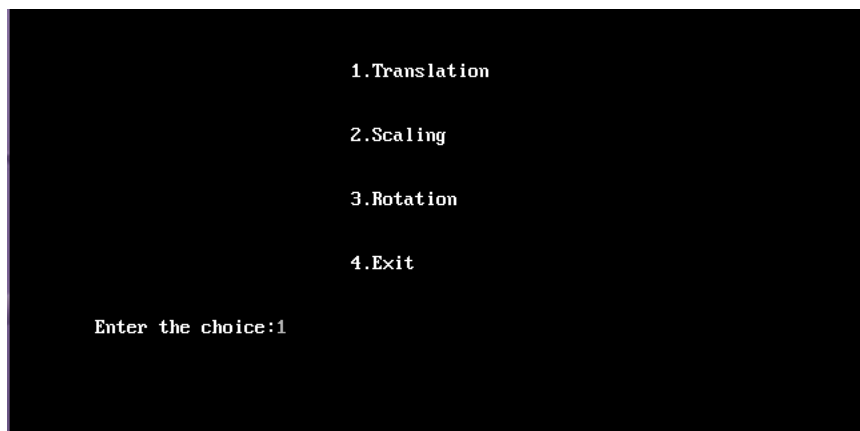
void construct1(int x[50],int y[50])

{
    drawpoly(n,x);

    drawpoly(n,y);
    line(x[0],x[1],y[0],y[1]);
    line(x[2],x[3],y[2],y[3]);
    line(x[4],x[5],y[4],y[5]);
    line(x[6],x[7],y[6],y[7]);
}

```

### Output:



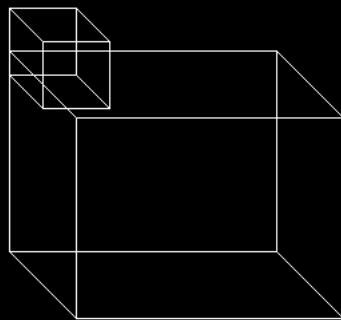
1.Translation

2.Scaling

3.Rotation

4.Exit

Enter the choice:2



Enter the scaling values:

4 3 2

Do you want to continue say(Y/N)?:Y

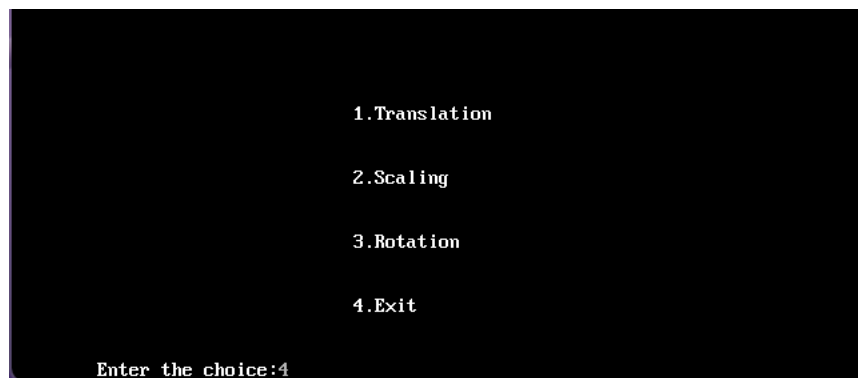
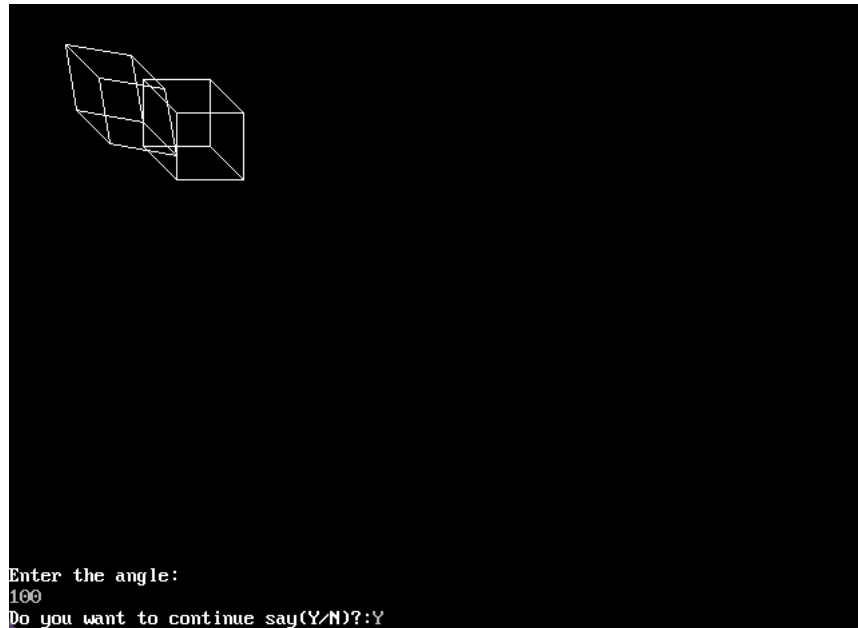
1.Translation

2.Scaling

3.Rotation

4.Exit

Enter the choice:3



### Result:

Thus the C program to implement the 3D-Transformations was written, executed and the output was verified successfully.

**Ex.No: 6**

## **3D Projections – Parallel, Perspective**

**DATE:**

---

**Aim:**

To write a C Program for Parallel, Perspective Projections

**Algorithm:**

1. Start the program.
2. Draw the 3D object and also display the choices.
3. Get the choice and perform the corresponding actions.
4. The choices is '1' then get the perspective values and move the 3D-object.
5. The choices is '2' then get the scaling vectors and change the size of 3D-object.
6. The choices is '4' then exit from the output window.
7. Terminate the program.

```
# include <iostream.h>
# include <graphics.h>
# include <conio.h>
# include <math.h>

# define n1 -1.0
# define n2 1.0
# define n3 1.0

# define x0 1.0
# define y0 1.0
# define z0 1.0

# define a 0.2
# define b 0.2
# define c 0.2

void show_screen( );
void draw_cube(int [8][3]);
void draw_pyramid(int [5][3]);
void get_projected_point(int&,int&,int&);
void multiply_matrices(constfloat[4],constfloat[4][4],float[4]);
void Line(constint,constint,constint,constint);

int main( )
{
    int driver=VGA;
    int mode=VGAHI;

    initgraph(&driver,&mode,"..\\Bgi");
```

```

show_screen( );

int cube[8][3]={
    {370,200,50},    // front left top
    {470,200,50},    // front right top
    {470,300,50},    // front right bottom
    {370,300,50},    // front left bottom
    {370,200,-50},   // back left top
    {470,200,-50},   // back right top
    {470,300,-50},   // back right bottom
    {370,300,-50}    // back left bottom
};

setcolor(15);
draw_cube(cube);

settextstyle(0,0,1);
outtextxy(400,320,"Cube");

int pyramid[5][3]={
    {120,300,50},    // base front left
    {220,300,50},    // base front right
    {220,300,-50},   // base back right
    {120,300,-50},   // base back left
    {170,150,0}      // top
};

setcolor(15);
draw_pyramid(pyramid);

settextstyle(0,0,1);
outtextxy(135,320,"Pyramid");

getch( );
closegraph( );
return 0;
}

```

```

/*****//-----
---- draw_cube( ) -----
//*****/
void draw_cube(int edge_points[8][3])
{
    for(int i=0;i<8;i++)
        get_projected_point(edge_points[i][0],
                             edge_points[i][1],edge_points[i][2]);

    Line(edge_points[0][0],edge_points[0][1],
          edge_points[1][0],edge_points[1][1]);
    Line(edge_points[1][0],edge_points[1][1],
          edge_points[2][0],edge_points[2][1]);
    Line(edge_points[2][0],edge_points[2][1],
          edge_points[3][0],edge_points[3][1]);
    Line(edge_points[3][0],edge_points[3][1],
          edge_points[0][0],edge_points[0][1]);

    Line(edge_points[4][0],edge_points[4][1],
          edge_points[5][0],edge_points[5][1]);
    Line(edge_points[5][0],edge_points[5][1],
          edge_points[6][0],edge_points[6][1]);
    Line(edge_points[6][0],edge_points[6][1],

```



```

        edge_points[7][0],edge_points[7][1]);
Line(edge_points[7][0],edge_points[7][1],
      edge_points[4][0],edge_points[4][1]);

Line(edge_points[0][0],edge_points[0][1],
      edge_points[4][0],edge_points[4][1]);
Line(edge_points[1][0],edge_points[1][1],
      edge_points[5][0],edge_points[5][1]);
Line(edge_points[2][0],edge_points[2][1],
      edge_points[6][0],edge_points[6][1]);
Line(edge_points[3][0],edge_points[3][1],
      edge_points[7][0],edge_points[7][1]);
}

/*****//-----
--- draw_pyramid( ) -----
//*****/void
draw_pyramid(int edge_points[5][3])
{
    for(int i=0;i<5;i++)
        get_projected_point(edge_points[i][0],
                             edge_points[i][1],edge_points[i][2]);

    Line(edge_points[0][0],edge_points[0][1],
          edge_points[1][0],edge_points[1][1]);
    Line(edge_points[1][0],edge_points[1][1],
          edge_points[2][0],edge_points[2][1]);
    Line(edge_points[2][0],edge_points[2][1],
          edge_points[3][0],edge_points[3][1]);
    Line(edge_points[3][0],edge_points[3][1],
          edge_points[0][0],edge_points[0][1]);

    Line(edge_points[0][0],edge_points[0][1],
          edge_points[4][0],edge_points[4][1]);
    Line(edge_points[1][0],edge_points[1][1],
          edge_points[4][0],edge_points[4][1]);
    Line(edge_points[2][0],edge_points[2][1],
          edge_points[4][0],edge_points[4][1]);
    Line(edge_points[3][0],edge_points[3][1],
          edge_points[4][0],edge_points[4][1]);
}

/*****//-----
get_projected_point( ) -----
//*****/
void get_projected_point(int& x,int& y,int& z)
{
    float d0=((n1*x0)+(n2*y0)+(n3*z0));
    float d1=((n1*a)+(n2*b)+(n3*c));
    float d=(d0-d1);

    float Per_NRC[4][4]={
        {(d-(a*n1)),(b*n1),(c*n1),n1 },
        {(a*n2),(d+n2*b),(c*n2),n2 },
        {(a*n3),(b*n3),(d+(c*n3)),n3 },
        {(-a*d0),(-b*d0),(-c*d0),-d1 }
    };

    float xy[4]={x,y,z,1};
    float new_xy[4]={0};

```

```

multiply_matrices(xy,Per_NRC,new_xy);

x=(int)(new_xy[0]+0.5);
y=(int)(new_xy[1]+0.5);
z=(int)(new_xy[2]+0.5);
}

/*****//-----
multiply_matrices( ) -----
//*****/void
multiply_matrices(constfloat matrix_1[4],
                  constfloat matrix_2[4][4],float matrix_3[4])
{
    for(int count_1=0;count_1<4;count_1++)
    {
        for(int count_2=0;count_2<4;count_2++)
            matrix_3[count_1]+=
                (matrix_1[count_2]*matrix_2[count_2][count_1]);
    }
}

/*****//-----
---- Line( ) -----
//*****/
void Line(constint x_1,constint y_1,constint x_2,constint y_2)
{
    int color=getcolor( );

    int x1=x_1;
    int y1=y_1;

    int x2=x_2;
    int y2=y_2;

    if(x_1>x_2)
    {
        x1=x_2;
        y1=y_2;

        x2=x_1;
        y2=y_1;
    }

    int dx=abs(x2-x1);
    int dy=abs(y2-y1);
    int inc_dec=((y2>=y1)?1:-1);

    if(dx>dy)
    {
        int two_dy=(2*dy);
        int two_dy_dx=(2*(dy-dx));
        int p=((2*dy)-dx);

        int x=x1;
        int y=y1;

        putpixel(x,y,color);

        while(x<x2)
        {
            x++;

```

```

        if(p<0)
            p+=two_dy;

        else
        {
            y+=inc_dec;
            p+=two_dy_dx;
        }

        putpixel(x,y,color);
    }
}

else
{
    int two_dx=(2*dx);
    int two_dx_dy=(2*(dx-dy));
    int p=((2*dx)-dy);

    int x=x1;
    int y=y1;

    putpixel(x,y,color);

    while(y!=y2)
    {
        y+=inc_dec;

        if(p<0)
            p+=two_dx;

        else
        {
            x++;
            p+=two_dx_dy;
        }

        putpixel(x,y,color);
    }
}

/*****//-----
---- show_screen( ) -----
//*****/void
show_screen( )
{
    setfillstyle(1,1);
    bar(182,26,438,38);

    settextstyle(0,0,1);
    setcolor(15);

    outtextxy(5,5,"*****");
};

    outtextxy(5,17,"*-
*****_");
    outtextxy(5,29,"*-----*");
    outtextxy(5,41,"*-
*****_");
    outtextxy(5,53,"*-
*****_");

```

```

setcolor(11);
outtextxy(189,29,"General Perspective Projection");

setcolor(15);

for(int count=0;count<=30;count++)
    outtextxy(5,(65+(count*12)), "*_*");

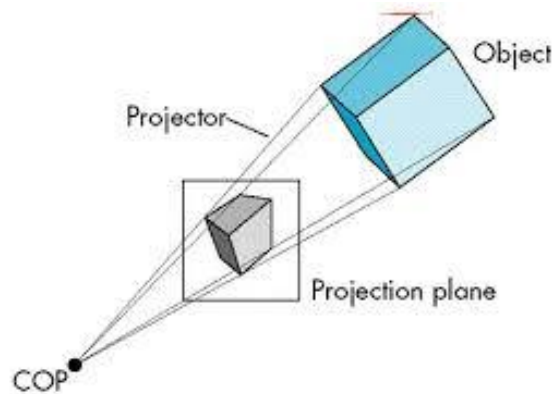
    outtextxy(5,438,"*-
*****_*");
    outtextxy(5,450,"*-----*");

outtextxy(5,462,"*****
**");

setcolor(12);
outtextxy(227,450," Press any key to exit ");

```

### Output:



### Result:

Thus the C program to implement the parallel and perspective was written, executed and the output was verified successfully

DATE:

**Aim:**

To Study about OPENGL – features, Primitives and Programming basics.

**Introduction:**

OpenGL is a powerful software interface used to produce high-quality computer-generated images and interactive graphics applications by rendering 2D and 3D geometric objects, bitmaps, and color images.

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you're using. Similarly, OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. With OpenGL, you must build up your desired model from a small set of *geometric primitives* - points, lines, and polygons.

A sophisticated library that provides these features could certainly be built on top of OpenGL. The OpenGL Utility Library (GLU) provides many of the modeling features, such as quadric surfaces and NURBS curves and surfaces. GLU is a standard part of every OpenGL implementation. Also, there is a higher-level, object-oriented toolkit, Open Inventor, which is built atop OpenGL, and is available separately for many implementations of OpenGL.

**Table 1-1 : Command Suffixes and Argument Data Types**

Suffix	Data Type	Typical C-Language Type	OpenGL Type
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	int or long	GLint, GLsizei
f	32-bit floating-point	float	GLfloat, GLclampf
d	64-bit floating-point	double	GLdouble, GLclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean

us	16-bit unsigned integer	unsigned short	GLushort
ui	32-bit unsigned integer	unsigned int or unsigned long	GLuint, GLenum, GLbitfield

```
glVertex2i(1, 3);
glVertex2f(1.0, 3.0);
```

are equivalent, except that the first specifies the vertex's coordinates as 32-bit integers

The following lines show how you might use a vector and a nonvector version of the command that sets the current color:

```
glColor3f(1.0, 0.0, 0.0);
GLfloat color_array[] = { 1.0, 0.0, 0.0 };
glColor3fv(color_array);
```

### GLUT, the OpenGL Utility Toolkit

GLUT to simplify opening windows, detecting input, and so on. If you have an implementation of OpenGL and GLUT on your system, the examples in this book should run without change when linked with them.

Five routines perform tasks necessary to initialize a window.

- **glutInit**(int *\*argc*, char *\*\*argv*) initializes GLUT and processes any command line arguments (for X, this would be options like -display and -geometry). **glutInit()** should be called before any other GLUT routine.
- **glutInitDisplayMode**(unsigned int *mode*) specifies whether to use an *RGBA* or color-index color model. You can also specify whether you want a single- or double-buffered window. (If you're working in color-index mode, you'll want to load certain colors into the color map; use **glutSetColor()** to do this.) Finally, you can use this routine to indicate that you want the window to have an associated depth, stencil, and/or accumulation buffer. For example, if you want a window with double buffering, the *RGBA* color model, and a depth buffer, you might call **glutInitDisplayMode**(*GLUT\_DOUBLE* | *GLUT\_RGB* | *GLUT\_DEPTH*).
- **glutInitWindowPosition**(int *x*, int *y*) specifies the screen location for the upper-left corner of your window.
- **glutInitWindowSize**(int *width*, int *size*) specifies the size, in pixels, of your window.
- int **glutCreateWindow**(char *\*string*) creates a window with an OpenGL context. It returns a unique identifier for the new window. Be warned: Until **glutMainLoop()** is called (see next section), the window is not yet displayed.

```
#include <GL/gl.h>
#include <GL/glut.h>
```

```
void display(void)
```

```

{
/* clear all pixels */
glClear (GL_COLOR_BUFFER_BIT);

/* draw white polygon (rectangle) with corners at
 * (0.25, 0.25, 0.0) and (0.75, 0.75, 0.0)
 */
glColor3f (1.0, 1.0, 1.0);
glBegin(GL_POLYGON);
    glVertex3f (0.25, 0.25, 0.0);
    glVertex3f (0.75, 0.25, 0.0);
    glVertex3f (0.75, 0.75, 0.0);
    glVertex3f (0.25, 0.75, 0.0);
glEnd();

/* don't wait!
 * start processing buffered OpenGL routines
 */
glFlush ();
}

void init (void)
{
/* select clearing (background) color */
glClearColor (0.0, 0.0, 0.0, 0.0);

/* initialize viewing values */
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

/*
 * Declare initial window size, position, and display mode * (single buffer and RGBA). Open
window with "hello" in its title bar. Call initialization routines.
 * Register callback function to display graphics.
 * Enter main loop and process events.
 */
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (250, 250);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("hello");
    init ();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0; /* ISO C requires main to return int. */
}

```

## Handling Input Events

You can use these routines to register callback commands that are invoked when specified events occur.

- **glutReshapeFunc**(void (\*func)(int *w*, int *h*)) indicates what action should be taken when the window is resized.
- **glutKeyboardFunc**(void (\*func)(unsigned char *key*, int *x*, int *y*)) and **glutMouseFunc**(void (\*func)(int *button*, int *state*, int *x*, int *y*)) allow you to link a keyboard key or a mouse button with a routine that's invoked when the key or mouse button is pressed or released.
- **glutMotionFunc**(void (\*func)(int *x*, int *y*)) registers a routine to call back when the mouse is moved while a mouse button is also pressed.

## Drawing Three-Dimensional Objects

GLUT includes several routines for drawing these three-dimensional objects:

cone	icosahedron	teapot
cube	octahedron	tetrahedron
dodecahedron	sphere	torus

You can draw these objects as wireframes or as solid shaded objects with surface normals defined. For example, the routines for a cube and a sphere are as follows:

```
void glutWireCube(GLdouble size);
```

```
void glutSolidCube(GLdouble size);
```

```
void glutWireSphere(GLdouble radius, GLint slices, GLint stacks);
```

```
void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);
```

## Animation

Most OpenGL implementations provide double-buffering - hardware or software that supplies two complete color buffers. One is displayed while the other is being drawn. When the drawing of a frame is complete, the two buffers are swapped, so the one that was being viewed is now used for drawing, and vice versa.



#### Procedure

```
open_window_in_double_buffer_mode();  
for (i = 0; i < 1000000; i++) {  
    clear_the_window();  
    draw_frame(i);  
    swap_the_buffers();  
}
```

#### Result:

The basics of OPENGL Programming was thus studied.

**Ex.No: 8**

## **Creating 3D Objects and Scenes**

**DATE:**

---

**Aim:**

To Write a OpenGL Program to display 3 D Objects and 3D Scenes.

**Description:**

The following primitives are used to display 3D objects.

void **glutWireCube**(GLdouble *size*);

void **glutSolidCube**(GLdouble *size*);

void **glutWireSphere**(GLdouble *radius*, GLint *slices*, GLint *stacks*);

void **glutSolidSphere**(GLdouble *radius*, GLint *slices*, GLint *stacks*);

**glPushMatrix()** copies the matrix on the top of the stack. This would be like a Save function

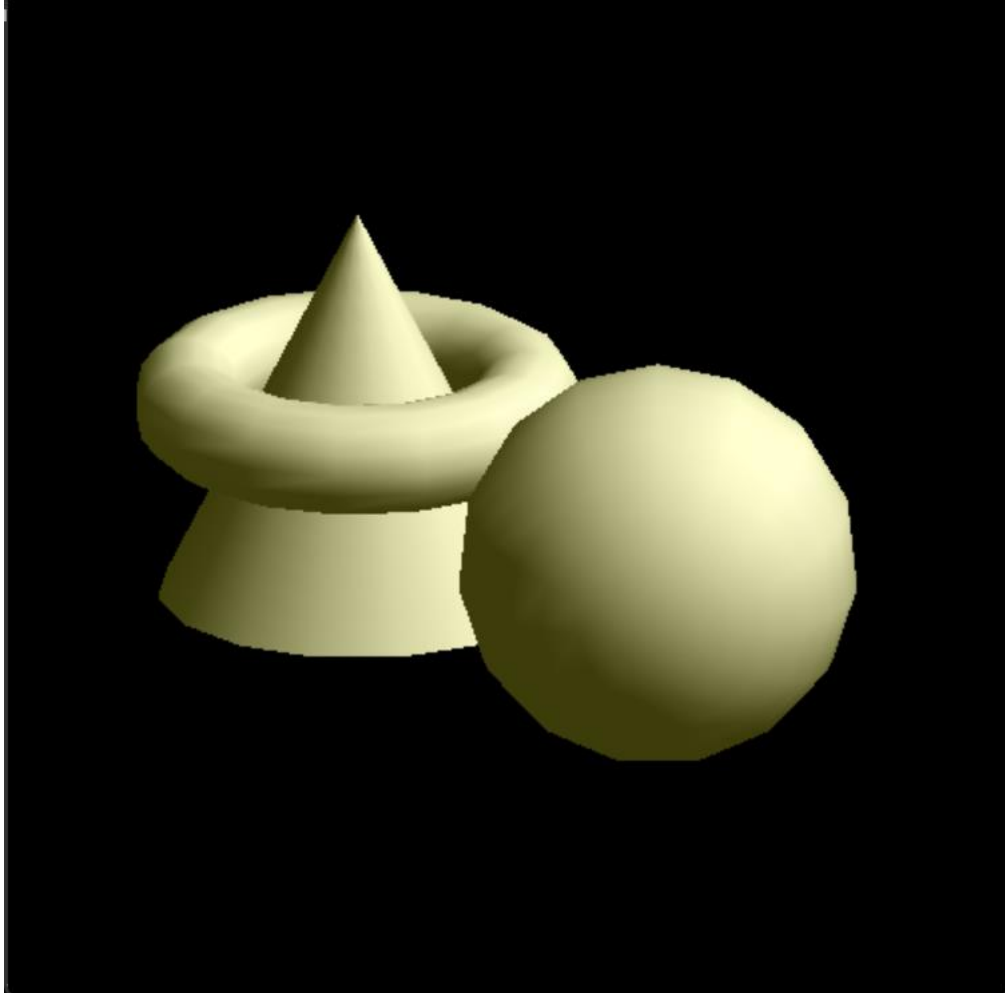
**glPopMatrix()** would be the equivalent of load. This gets rid of the top matrix in the stack, and sets the matrix underneath it as the new current matrix. So, a **glPopMatrix()** could undo immediately the result of several or even hundreds of transformations

**Program:**

```
#include <gl/glut.h>
void init (void)
{
    GLfloat light_ambient[] = { 1.0, 1.0, 0.0, 1.0 };
    GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
    glLightfv (GL_LIGHT0, GL_AMBIENT, light_ambient);
    glLightfv (GL_LIGHT0, GL_DIFFUSE, light_diffuse);
    glLightfv (GL_LIGHT0, GL_SPECULAR, light_specular);
    glLightfv (GL_LIGHT0, GL_POSITION, light_position);
    glEnable (GL_LIGHTING);
    glEnable (GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);
}
void display (void)
{
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix ();
    glRotatef (20.0, 1.0, 0.0, 0.0);
    glPushMatrix ();
    glTranslatef (-0.75, 0.5, 0.0);
    glRotatef (90.0, 1.0, 0.0, 0.0);
    glutSolidTorus (0.275, 0.85, 15, 15);
    glPopMatrix ();
    glPushMatrix ();
    glTranslatef (-0.75, -0.5, 0.0);
    glRotatef (270.0, 1.0, 0.0, 0.0);
    glutSolidCone (1.0, 2.0, 15, 15);
    glPopMatrix ();
    glPushMatrix ();
    glTranslatef (0.75, 0.0, 1.0);
    glutSolidSphere (1.0, 15, 15);
    glPopMatrix (); glPopMatrix ();
    glFlush ();
}
void reshape(int w, int h)
{
    glViewport (0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    if (w <= h)
```

```
glOrtho (-2.5, 2.5, -2.5*(GLfloat)h/(GLfloat)w,  
2.5*(GLfloat)h/(GLfloat)w, -10.0, 10.0);  
else  
glOrtho (-2.5*(GLfloat)w/(GLfloat)h,  
2.5*(GLfloat)w/(GLfloat)h, -2.5, 2.5, -10.0, 10.0);  
glMatrixMode (GL_MODELVIEW);  
glLoadIdentity ();  
}  
int main(int argc, char** argv)  
{  
    glutInit(&argc, argv);  
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);  
    glutInitWindowSize (500, 500);  
    glutCreateWindow (argv[0]);  
    init ();  
    glutReshapeFunc (reshape);  
    glutDisplayFunc(display);  
    glutMainLoop();  
    return 0;  
}
```

**Output:**



**Result:**

Thus the 3D Objects and scenes were drawn using Opengl.

**Ex.No: 10      2D Animation – To create Interactive animation using any authoring tool.**

**DATE:**

---

**Aim:**

To create Interactive animation using the tool.

#### **10 A)Pencil –Animation tool**

step 1. Draw backgrounds, or import a background from an image file

step2. Create multiple layers for characters or other animated objects

step 3.Draw each frame manually, or copy and paste the previous images (no ‘tween’ option)

step 4 Export your animation in .FLV format, or as individual images for each frame.

Step 5Add sound to your animation

#### **10 b)Synfig Studio**

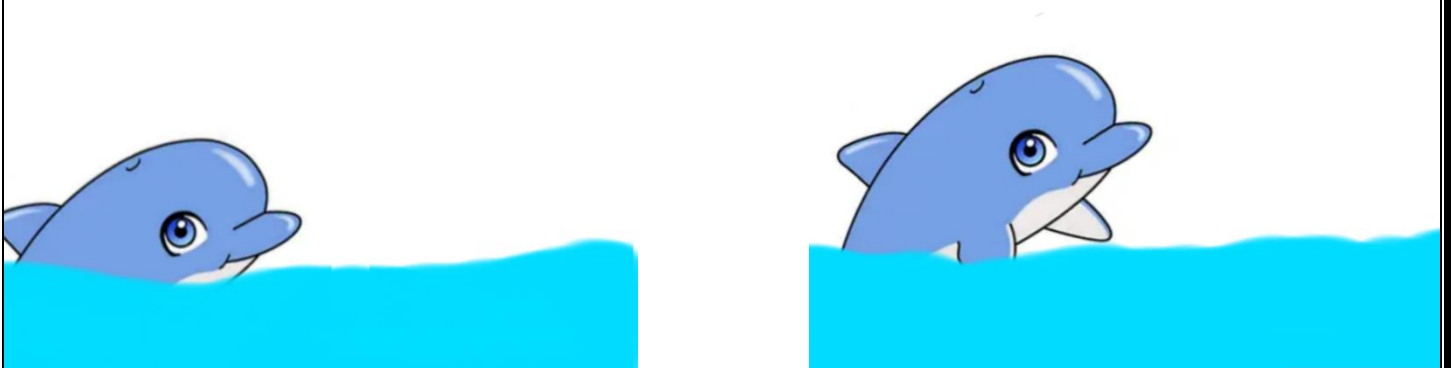
##### **Software Decsription**

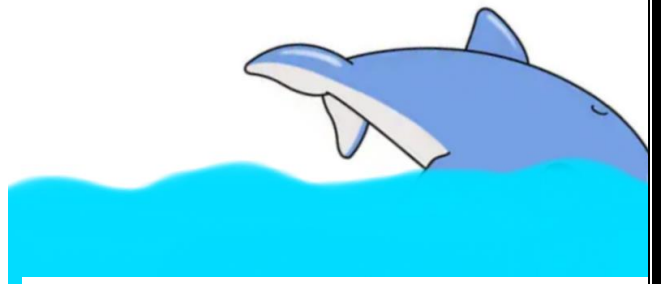
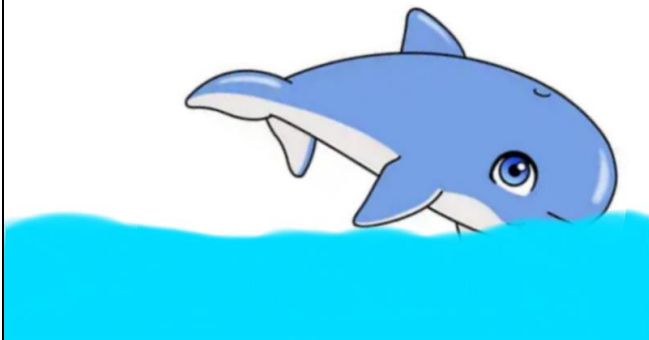
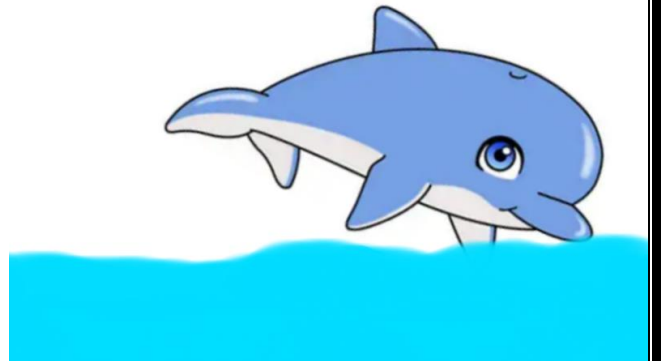
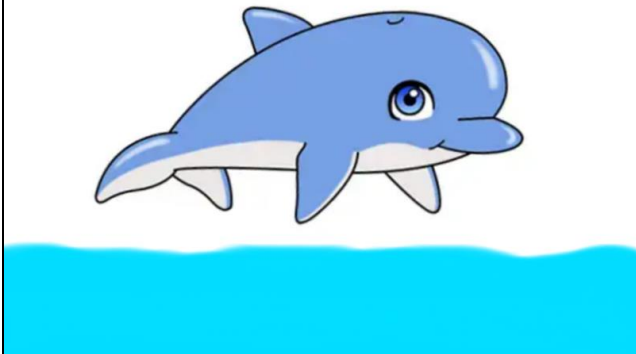
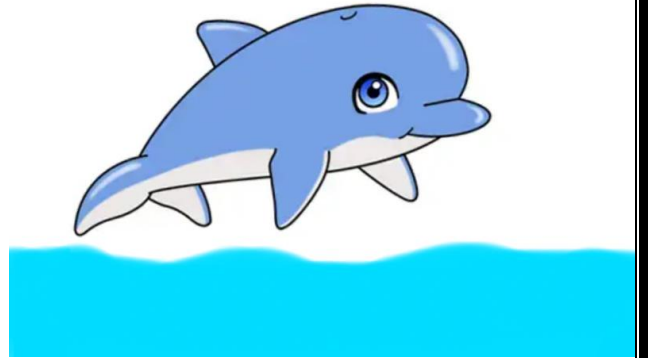
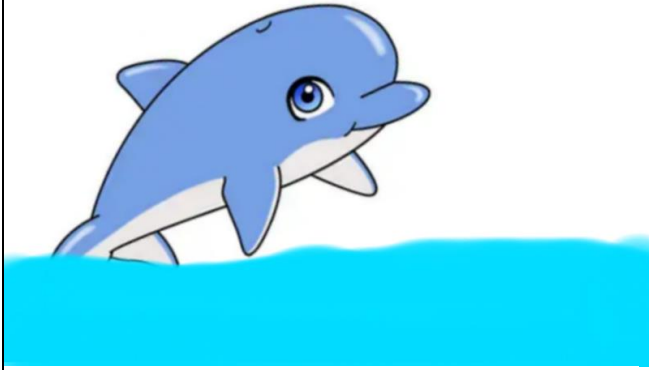
- Synfig Studio is a free and open-source 2D animation software
- Designed as powerful industrial-strength solution for creating film-quality animation using a vector and bitmap artwork.
- It eliminates the need to create animation frame-by frame, allowing you to produce 2D animation of a higher quality with fewer people and resources.

##### **Steps:**

1. Export animations in a variety of common video formats
2. ‘Tween’ objects from one area to another, without have to manually draw each frame
3. Works well with vector images

**Output:**





**Result:**

Thus 2D– Interactive animation is created using the software pencil and synfig studio.