Bharathiar University, Coimbatore
Department of Information Technology.

Graphics and Multimedia Systems Lab- 22IT23P

CONTENT

1- BRESENHAM’S LINE DRAWING ALGORITHM

AIM:

To draw a line using Bresenham’s algorithm

ALGORITHM:

1. Read the starting and ending co-ordinates
2. Calculate the distance between end points
3. Find the starting and ending locations
4. While starting location is less than the ending location ,do the following
    a. Calculate x & y co- ordinates
    b. Plot the pixel
5. Display the line.

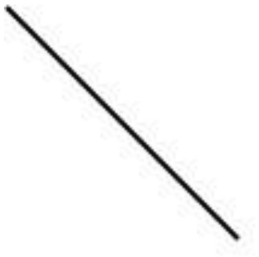
SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h> void
main()
{
int x1,x2,y1,y2,gd=DETECT,gm,dx,dy,step;
int x,y,const1,const2,k,p,x_end; initgraph(&gd,&gm,"");
printf("\nEnter the end point co-ordinates of the line.......\n");
scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
dx=abs(x2-x1);
dy=abs(y2-y1);
p=2*dy-dx;
const1=2*dy;
const2=2*(dy-dx);
if(x1>x2) { x=x2;
y=y2; x_end=x1;
} else {
x=x1;
y=y1;
x_end=x2;
}
putpixel(x,y,15);
while(x<x_end)
{ x++;
if(p<0)
p+=const1;
else
  {   y++;
p+=const2;
  }
 putpixel(x,y,15);
 }  getch();
closegraph();
}
```

INPUT:

```
Enter the end point co-ordinates of the line.......
100   200   300   400
```

OUTPUT:



RESULT:

Thus the program to draw a line using Bresenham’s algorithm was executed.

2- BRESENHAM’S CIRCLE ALGORITHMS

AIM:

To draw a circle using Bresenham’s circle algorithm

ALGORITHM:

```
1. Read the center and radius of the circle.
2. Shift the centre to origin ,consider the points  on y-axis at a distance of radius
3. Plot the pixel at the centre of circle
4. While x – coordinates  is less than the radius  ,do the following
    a. Increment x by one and calculate y- co ordinate
    b. Shift the coordinate to original position
    c. Plot the pixel
5. Display the circle.
```

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h> void
main()
{
void plotpoint(int,int,int,int); int
xc,yc,r,x,y,p;
int gd=DETECT,gm;
clrscr();
initgraph(&gd,&gm," ");
printf("Enter the center co-ordinates of the circle:\n");
scanf("%d%d",&xc,&yc);
printf("Enter the radius of the circle:\n");
scanf("%d",&r);
x=0; y=r;
plotpoint(xc,yc,x,y);
p=1-r; while(x<y) {
if(p<0)
x=x+1; else
{
x=x+1;
y=y-1; }
if(p<0)
p=p+2*
x+1;
else p=p+2*(x-y)+1;
plotpoint(xc,yc,x,y);
} getch();
}
```

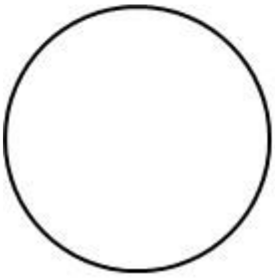
```
void plotpoint(int xc,int yc,int x,int y)
```

```
{ putpixel(xc+x,yc+y,1);
putpixel(xc-x,yc+y,1);
putpixel(xc+x,yc-y,1);
putpixel(xc-x,yc-y,1);
putpixel(xc+y,yc+x,1);
putpixel(xc-y,yc+x,1);
putpixel(xc+y,yc-x,1);
putpixel(xc-y,yc-x,1);
 }
```

INPUT:

```
Enter the center co-ordinates of the circle:
200
200
Enter the radius of the circle:
50
```

OUTPUT:



RESULT :

Thus the program to draw a circle using Bresenham’s algorithm was executed.

3- BRESENHAM’S ELLIPSE ALGORITHMS

AIM:

To draw an ellipse using Bresenham’s algorithm

ALGORITHM:

1. Read the center and radius of x & y axis
2. Obtain the first position on an ellipse centered on the origin
3. Calculate the initial value of decision parameter
4. At each position calculate the next point along ellipse
5. Determine the symmetric points in other 3 quadrants and plot them
6. Display the ellipse.

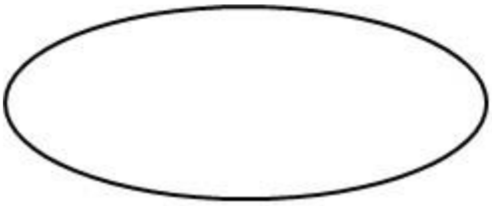
SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#define ROUND(a) ((int)(a+0.5))
void plotpoint(long int,long int,long int,long int); void
main()
{
long int xc,yc,rx,ry,rx2,ry2,x,y,px,p,tworx2,twory2,py;
int gd=DETECT,gm; initgraph(&gd,&gm,"
");
printf("Enter the center co-ordinates of the ellipse:\n");
scanf("%ld %ld ",&xc,&yc); printf("Enter the radius
along x-axis and y-axis:\n");
scanf("%ld %ld",&rx,&ry);
x=0; y=ry; px=0;
rx2=rx*rx; ry2=ry*ry;
tworx2=2*rx2;
twory2=2*ry2;
py=tworx2*y;
plotpoint(xc,yc,x,y);

p=ROUND(ry2-(rx2*ry)+(.25*rx2));
while(px<py)
{ x++;
px+=twory2;
if(p<0)
{ p+=ry2+px;
} else
{
y--;
py-=tworx2;
p+=ry2+px-py;
}
plotpoint(xc,yc,x,y);
}
p=ROUND(ry2*(x+0.5)*(x+0.5)+rx2*(y-1)*(y-1)-rx2*ry2);

while(y>0) {
y--;
py-=tworx2;

if(p>0)
p+=rx2-py;
else { x++;
px+=twory2;
p+=rx2-py+px;
}
plotpoint(xc,yc,x,y);
} getch();
}

void plotpoint(long int xc,long int yc,long int x,long int y)
{
putpixel(xc+x,yc+y,156); putpixel(xc-
x,yc+y,156); putpixel(xc+x,yc-y,156);
putpixel(xc-x,yc-y,156);
}
```

INPUT:

```
Enter the center co-ordinates of the ellipse:
200
200
Enter the radius along x-axis and y-axis:
80
40
```

OUTPUT:



RESULT:

Thus the program to draw an ellipse using Bresenham’s algorithm was executed.

4. TWO DIMENSIONAL TRANSFORMATION

AIM :

To implement the various 2D transformations like translation, scaling, rotation, shearing and reflection.

ALGORITHM:

To draw polygon

1. Read the number of vertices.
2. Read the x & y coordinates of the vertices and store it in an array.
3. Draw the polygon using drawpoly().

Translation

It is applied to an object by repositioning it along a straight line path from one coordinate to another.

1. Read the translation distance tx & ty
2. Add tx & ty to the coordinates to move to the new position
3. Display the polygon

Scaling

It alters the size of an object, by multiplying the coordinate values of each vertex by scaling factors.

1. Read the scaling factor sx
2. Multiply the scaling sx with each coordinate vertex to alter the size.
3. Display the polygon.

Rotation

It is applied to an object by repositioning it along a circular path in the xy plane

1. Read the rotation factor or pivot point (a) and distance (xr, yr) from origin.
2. Polygon is rotated by displacing each vertex through the specified rotation angle (a)
3. Display the polygon.

Reflection

A reflection is a transformation that produces a mirror image of an object.

1. Reflection of an object is produced by displacing the object along x axis & y axis being the same.
2. Display the polygon.

Shearing

A transformation that distorts the shape of an object

1. Read the shearing factor (sh).
2. Shearing an object relative to x axis produced by an expression
    a. $x^l = x + sh * y$
    b. yl = y
3. Display the polygon.

SOURCE CODE :

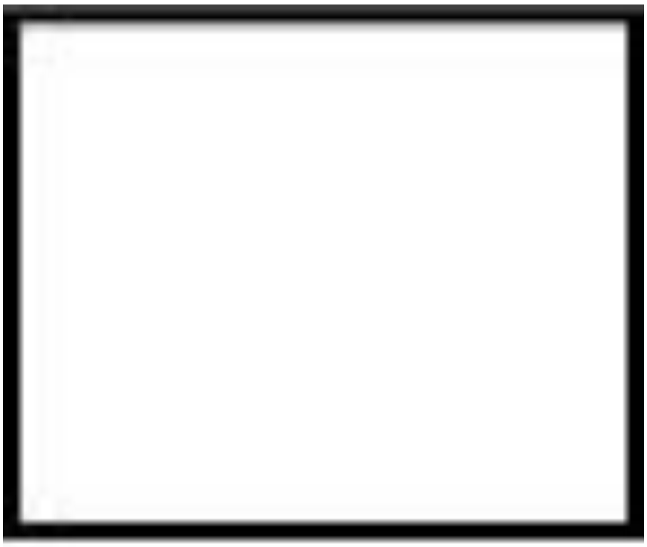
```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void main() {
int i,poly4[10],poly1[10],poly2[10],poly3[10],tx,ty,sx,n,xr,yr,a;
int poly[10],poly5[10],sy; int gd=DETECT,gm;
initgraph(&gd,&gm," ");
printf("\nEnter the number of vertices of the polygon:\n"); scanf("%d",&n);
printf("Enter the(x,y) co-ordinates of the vertices:\n");
for(i=0;i<2*n;i++)
{
scanf("%d",&poly[i]);
} poly[2*n]=poly[0];
poly[2*n+1]=poly[1];
outtextxy(70,70,"Original
image"); drawpoly(n+1,poly);

//TRANSLATION
getch(); cleardevice();
outtextxy(30,30,"OriginalImage");
drawpoly(n+1,poly);
outtextxy(10,10,"Enter the Translation Factor");
gotoxy(30,3); scanf("%d %d",&tx,&ty);
for(i=0;i<2*n;i+=2)
{  poly1[i]=poly[i]+tx;
poly1[i+1]=poly[i+1]+ty;
} poly1[2*n]=poly1[0];
poly1[2*n+1]=poly1[1];
drawpoly(n+1,poly1);
getch();
cleardevice();
//SCALING
outtextxy(30,30,"Original Image");
drawpoly(n+1,poly);
outtextxy(10,10,"Enter the Scaling Factor");
gotoxy(30,3); scanf("%d",&sx);
for(i=0;i<2*n;i+=2)
{ poly2[i]=poly[i]*sx;
poly2[i+1]=poly[i+1]*sx;
} poly2[2*n]=poly2[0];
poly2[2*n+1]=poly2[1];
drawpoly(n+1,poly2);
getch(); cleardevice();
//ROTATION
outtextxy(30,30,"Original Image"); drawpoly(n+1,poly);
outtextxy(10,10,"Enter the Rotation Factor");
gotoxy(30,3); scanf("%d%d%d",&xr,&yr,&a);
for(i=0;i<2*n;i+=2)
{   poly3[i]=xr+((poly[i]-xr)*cos(a))-((poly[i+1]-yr)*sin(a));
poly3[i+1]=yr+((poly[i+1]-yr)*cos(a))+((poly[i]-
xr)*sin(a));
} poly3[2*n]=poly3[0];
poly3[2*n+1]=poly3[1];
drawpoly(n+1,poly3);
getch(); cleardevice();

//REFLECTION
outtextxy(30,30,"Original Image");
drawpoly(n+1,poly); outtextxy(10,10,"Reflected
Image"); for(i=0;i<2*n;i+=2)
{ poly4[i]=640-poly[i];
poly4[i+1]=poly[i+1];
} poly4[2*n]=poly4[0];
poly4[2*n+1]=poly4[1];
drawpoly(n+1,poly4);
getch();
cleardevice();

//SHEARING
outtextxy(30,30,"Original Image");
drawpoly(n+1,poly);
outtextxy(10,10,"Enter the shear factor");
gotoxy(30,3); scanf("%d",&sh);
for(i=0;i<2*n;i+=2)
{ poly5[i]=poly[i]+sh*poly[i+1];
poly5[i+1]=poly[i+1];
}
poly5[2*n]=poly5[0]; poly5[2*n+1]=poly5[1];
drawpoly(n+1,poly5);
getch();
cleardevice();

}
```

Input:

```
Enter the number of vertices of the polygon:
4
Enter the(x,y) co-ordinates of the vertices:
100 100 150 100 150 150 100 150
```

Output

Original image



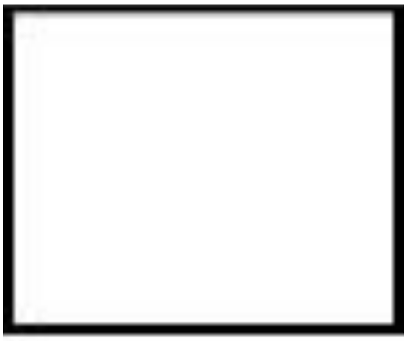
Translation

Input:

Enter the Translation Factor 200 200

Output

Original image



Scaling

Input:

Enter the scaling Factor 2

Output

Original image

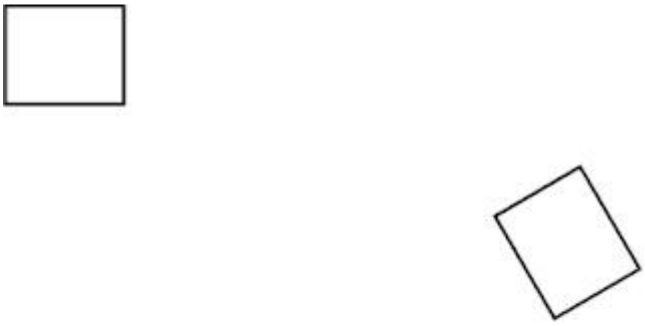
Rotation

Input:

Enter the Reflection Factor: 250 250 60

Output

Original image



Reflection

Output

Original image

Shearing

Input:

```
Enter the Shearing Factor:      1
```

Output

Original image

RESULT:

Thus the program to implement various 2D transformations like translation, scaling, rotation, reflection & shearing was executed.

5- LINE CLIPPING

AIM:

To write a program to clip the line using Cohen Sutherland algorithm.

ALGORITHM:

1. Read the starting & ending coordinates of the line to be clipped.
2. Draw the clipping rectangle
3. Display the line before clipping.
4. Check the line end points against clipping boundaries in the order left, right, bottom & top.
5. Find the intersection point of the line with the boundary of clipping rectangle.
6. Discard the line which is outside the window after saving the intersection points.
7. Display the line inside the clipping window.

SOURCE CODE

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
#include<graphics.h>
/* Defining structure for end point of line */ 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,"");
cleardevice();
printf("\n\n\tEnter End point 1 (x,y): ");
scanf("%d %d",&p1.x,&p1.y); printf("\n\n\tEnter
End point 2 (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);
}

/* function to draw window*/
void drawwindow()
{  setcolor(RED);
line(150,100,450,100);
line(450,100,450,350);
line(450,350,150,350);
line(150,350,150,100);
 }

 /*function to draw line between two points*/
void drawline(pt p1,pt p2,int c1)
 {  setcolor(c1);
line(p1.x,p1.y,p2.x,p2.y);
 }

 /*function to set code of the coordinate*/
pt setcode(pt p)
{
 pt ptemp;

 if(p.y<100)
ptemp.code[0]='1'; //top
else   ptemp.code[0]='0';
 if(p.y>350)

 ptemp.code[1]='1'; //bottom
else   ptemp.code[1]='0';

  if(p.x>450)
ptemp.code[2]='1'; //right
else   ptemp.code[2]='0';

  if(p.x<150)
ptemp.code[3]='1'; //left
else   ptemp.code[3]='0';
```

```
  ptemp.x = p.x;
ptemp.y=p.y;
  return(ptemp);
  }

  /*function to determine visibillity of line*/
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(0);
  for(i=0;i<4;i++)
  {
  if((p1.code[i]==p2.code[i])&&(p1.code[i]=='1'))
  flag=0;
  }

  if(flag==0)
return(1);    return(2);
  }
  /*function to find new end points*/
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')  /*cutting top edge*/
y=100;
```

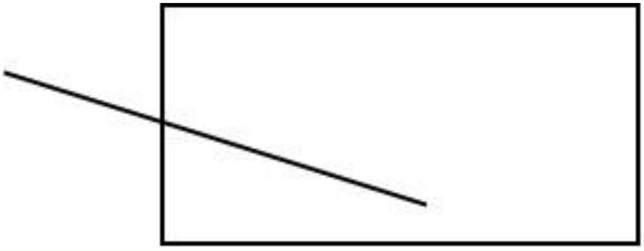
```
   if(p1.code[1]=='1')  /*cutting bottom edge*/
 y=350;
    if((p1.code[0]=='1')||(p1.code[1]=='1'))
   {
     m=(float)(p2.y-p1.y)/(p2.x-p1.x);
 k=(float)p1.x+(float)(y-p1.y)/m;
 temp.x=k;      temp.y=y;
 for(i=0;i<4;i++)
 temp.code[i]=p1.code[i];
     return(temp);
     }
 else
     return(p1);
     }
```

Input:

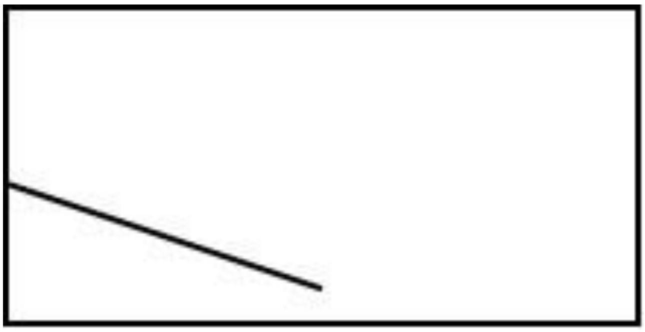
Enter the staring co ordinate:
130   90
Enter the ending co ordinate:
200   125

OUTPUT

Line before clipping



Line after clipping



RESULT: Thus the program to clip the line using Cohen Sutherland algorithm was executed.

6- POLYGON CLIPPING

AIM:

To write a program to clip the polygon using Cohen Sutherland algorithm.

ALGORITHM:

1. Read the coordinates of clipping window.
2. Draw the clipping window.
3. Read the number of polygon vertices & draw the polygon
4. Check the polygon vertices against clipping boundaries in the order left, right, bottom & top.
5. Find the intersection point of the polygon with the boundary of clipping rectangle.
6. Discard the polygon vertices which are outside the window after saving the intersection points.
7. Display the polygon inside the clipping window.

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h> #include<stdlib.h>
enum area{left,right,top,bottom}; typedef
struct
{  double x,y;  }
points; points
outvertex[10]; points
vertex[10];
int max=0;
int n; enum
area id;
void sutherclip(int,int,int,int); points
intersect(int,points,points); int
inside(int,points);

int inside(int clipbound,points s)
{   int
pos=0;
switch(id)
            {
case left:
if(s.x>clipbound)
                                 pos=1;
break;                case right:
if(s.x<clipbound)
```

```
pos=1;
break;                case top:
if(s.y>clipbound)
pos=1;
break;                case bottom:
if(s.y<clipbound)
pos=1;
break;
               }
    return(pos);
    }
points intersect(int clipbound,points s ,points p)
{ points
temp;
double calc;


switch(id)
{  case
left:  case
right:
      temp.x=clipbound;
      temp.y=s.y+(p.y-s.y)*(clipbound-s.x)/(p.x-s.x);
      break;  case
bottom:  case top:
temp.y=clipbound;
      temp.x=s.x+(p.x-s.x)*(clipbound-s.y)/(p.y-s.y);
      break;
      }
 return temp;
 }


 void clip(int xmin,enum area id1)
 {
  int i;
  points temp;
points s,p;   int
pt1,pt2;   id=id1;
for(i=0;i<n;i++)
  {
s=vertex[i];
if(i==n-1)
p=vertex[0];
else
p=vertex[i+1];
```

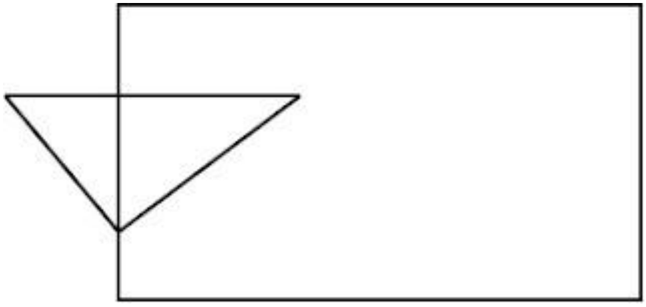
```
pt1=inside(xmi
n,s);
   pt2=inside(xmin,p);

   if(pt1==1 && pt2==1)
outvertex[max++]=p;
   if(pt1==0 && pt2==1)
    {
     temp=intersect(xmin,s,p);
outvertex[max++]=temp;
     outvertex[max++]=p;
     }
    if(pt1==1 && pt2==0)
    {
     temp=intersect(xmin,s,p);
     outvertex[max++]=temp;
      }
  }
  n=max;

  for(i=0;i<max;i++)
vertex[i]=outvertex[i];
  max=0;
  }

  void sutherclip(int xmin,int xmax,int ymin,int ymax)
  {
clip(xmin,left);
clip(xmax,right);
clip(ymin,top);
  clip(ymax,bottom);
  }

  void main()    scanf("%lf %lf",&vertex[i].x ,&vertex[i].y);
    }
    initgraph(&gd,&gm,"");
printf("\n\nBefore Clipping");
rectangle(xmin,ymin,xmax,ymax);
    for(i=0;i<n-1;i++)
    {
    line(vertex[i].x,vertex[i].y,vertex[i+1].x,vertex[i+1].y);
    line(vertex[n-1].x,vertex[n-1].y,vertex[0].x,vertex[0].y);
    }
    getch();
    sutherclip(xmin,xmax,ymin,ymax);
    cleardevice();      printf("After
clipping");
rectangle(xmin,ymin,xmax,ymax);
for(i=0;i<n-1;i++)
    {
    line(outvertex[i].x,outvertex[i].y,outvertex[i+1].x,outvertex[i+1].y);      line(outvertex[n-
1].x,outvertex[n-1].y,outvertex[0].x,outvertex[0].y);
    }
getch();
    closegraph();
```

INPUT:

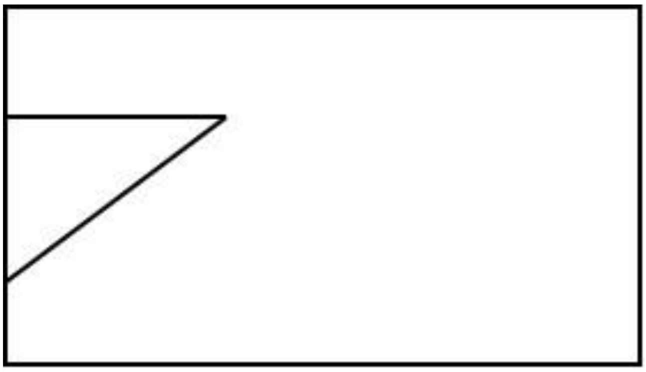
```
       Enter the co ordinates of clipping window
100 100  300  300
       Enter the no of vertices of the polygon…..
       3
       Enter the x & y co ordinates of vertex 1
        75    150
       Enter the x & y co ordinates of vertex 2
       150    150
      Enter the x & y co ordinates of vertex 3
      100   200
```

Output:

Before Clipping



After Clipping



RESULT:

Thus the program to clip the polygon using Cohen Sutherland algorithm was executed.

7- WINDOW-VIEWPORT MAPPING

AIM:

To write a program to perform window – view port mapping.

ALGORITHM:

1. Read the window & view port co-ordinates.
2. Read the number of polygon vertices 3.
   Draw the filled polygon inside the window.
4. Draw the view port
5. Map the polygon which is in the window to the view port.
6. Display the polygon inside the view port.

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h> #include<math.h>
int x[10],y[10],pol[25],ny,xymin,xymax,ywmax,ywmin,yymax,yymin,xwmax,xwmin; void
viewport()
{ float
sx,sy; int
i,j,k;
sx=(float)((float)(xymax-xymin)/(float)(xwmax-xwmin));
sy=(float)((float)(yymax-yymin)/(float)(ywmax-ywmin)); for(i=0;i<3;i++)
{
x[i]=sx*(x[i]-xwmin)+xymin; y[i]=sy*(y[i]-ywmin)+yymin;
} k=0;
for(i=0;i<ny;i++)
{ pol[k]=x[i];
pol[k+1]=y[i];
k+=2;
}
}

void main()
```

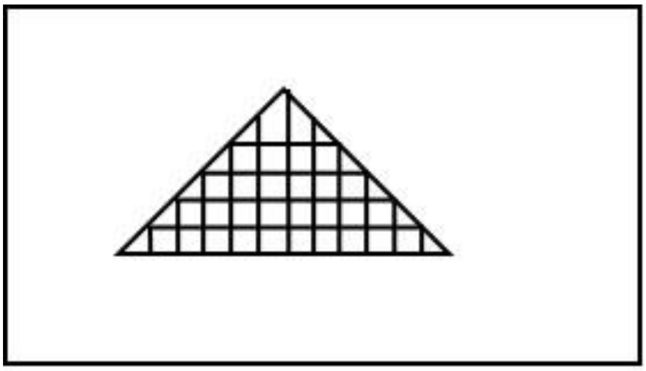
```
{
int i,j,d=DETECT,m=DETECT;
clrscr();
printf("Enter the window co ordinates:");
scanf("%d%d%d%d",&xwmin,&ywmin,&xwmax,&ywmax);
printf("Enter the view port co ordinates:");
scanf("%d%d%d%d",&xymin,&yymin,&xymax,&yymax);
printf("Enter the number of vertices");
scanf("%d",&ny); printf("Enter the
%d values",ny*2); for(i=0;i<ny;i++)
scanf("%d%d",&x[i],&y[i]);
j=0; for(i=0;i<ny;i++)
{ pol[j]=x[i];
pol[j+1]=y[i]; j+=2;
}
initgraph(&d,&m,"");
settextstyle(3,0,3); outtextxy(xwmin,ywmin-35,"Window");
rectangle(xwmin,ywmin,xwmax,ywmax);
setfillstyle(8,getmaxcolor()); fillpoly(ny,pol);
moveto(x[0],y[0]);
lineto(x[0],y[0]);
getch(); viewport();
outtextxy(xymin,yymin-35,"Viewport");
rectangle(xymin,yymin,xymax,yymax);
setfillstyle(8,getmaxcolor());
fillpoly(ny,pol); moveto(x[0],y[0]);
lineto(x[0],y[0]); getch(); closegraph();
}
```

<u>Input:</u>

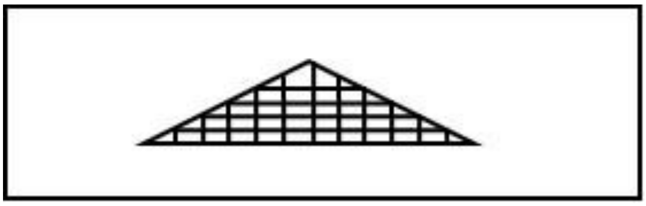
```
Enter the window co ordinates:
  120 50 350 140
Enter the view port co ordinates:
  400 150 500 170
Enter the number of vertices
  3
Enter the 6 values: 250 80 200 100 300 100
```

<u>Output:</u>

WINDOW



VIEWPORT



RESULT:

Thus the program to perform window

8. THREE DIMENSIONAL TRANSFORMATION

AIM:

To implement the various 3D transformations like translation, scaling and rotation.

.

ALGORITHM:

To draw polygon

1. Read the number of vertices.
2. Read the x & y coordinates of the vertices and store it in an array.
3. Draw the polygon using drawpoly().

Translation

It is applied to an object by repositioning it along a straight line path from one coordinate to another.

1. Read the translation distance tx & ty
2. Add tx & ty to the coordinates to move to the new position
3. Display the polygon

Scaling

It alters the size of an object, by multiplying the coordinate values of each vertex by scaling factors.

1. Read the scaling factor sx
2. Multiply the scaling sx with each coordinate vertex to alter the size.
3. Display the polygon.

Rotation

It is applied to an object by repositioning it along a circular path in the xy plane.

1. Read the rotation factor or pivot point (a) and distance (xr, yr) from origin.
2. Polygon is rotated by displacing each vertex through the specified rotation angle (a).
3. Display the polygon.

SOURCE CODE:

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

void main()
{
int n,poly[20],poly1[20],poly2[10],sx,poly3[10];
int tx,ty,xr,yr,a,sy,poly4[10],poly5[10],poly6[10],poly7[10]; int
i;
int gd=DETECT,gm;
initgraph(&gd,&gm," ");

//STEP TO DRAW A POLYGAN
printf("\nEnter the number of vertices of the polygan.....\n"); scanf("%d",&n);
printf("\nEnter the X,Y co-ordinates of each vertex....\n"); for(i=0;i<2*n;i++)
{ scanf("%d",&poly[i]);
poly[i]=(poly[i])+40;
poly[i+1]=(poly[i+1])+40;
poly1[i]=(poly[i]+50)+40;
poly1[i+1]=(poly[i+1]+50)+40;
} poly[2*n]=poly[0];
poly[2*n+1]=poly[1];
poly1[2*n]=poly1[0];
poly1[2*n+1]=poly1[1];
outtextxy(70,70,"ORIGINAL IMAGE");
drawpoly(n+1,poly);
drawpoly(n+1,poly1); for(i=0;i<2*n;i+=2)
line(poly[i],poly[i+1],poly1[i],poly1[i+1]);
getch(); cleardevice();
```

//TRANSLATION

```
outtextxy(30,30,"ORIGINAL IMAGE");
drawpoly(n+1,poly);
drawpoly(n+1,poly1); for(i=0;i<2*n;i+=2)
line(poly[i],poly[i+1],poly1[i],poly1[i+1]);
getch();
outtextxy(10,10,"Enter the Translation factor:"); gotoxy(30,3);
scanf("%d%d",&tx,&ty);

for(i=0;i<2*n;i+=2)
{ poly2[i]=poly[i]+tx;
poly2[i+1]=poly[i+1]+ty;
poly3[i]=poly1[i]+tx;
poly3[i+1]=poly1[i+1]+ty
;
}
poly2[2*n]=poly2[0];
poly2[2*n+1]=poly2[1];
poly3[2*n]=poly3[0];
poly3[2*n+1]=poly3[1];
drawpoly(n+1,poly2); drawpoly(n+1,poly3);
for(i=0;i<2*n;i+=2)
```

```
line(poly2[i],poly2[i+1],poly3[i],poly3[i+1]);
getch(); cleardevice();

//SCALING

printf("\n\n\t original image!");
drawpoly(n+1,poly);
drawpoly(n+1,poly1); gotoxy(300,100);
for(i=0;i<2*n;i+=2)
line(poly[i],poly[i+1],poly1[i],poly1[i+1]);
getch();
outtextxy(10,10,"Enter the scaling factor:");
gotoxy(30,3); scanf("%d",&sx);
for(i=0;i<2*n;i+=2) { poly4[i]=poly[i]*sx;
poly4[i+1]=poly[i+1]*sx;
poly5[i]=poly1[i]*sx;
poly5[i+1]=poly1[i+1]*sx;
} poly4[2*n]=poly4[0];
poly4[2*n+1]=poly4[1];
poly5[2*n]=poly5[0];
poly5[2*n+1]=poly5[1];
drawpoly(n+1,poly4); drawpoly(n+1,poly5);
for(i=0;i<2*n;i+=2)
line(poly4[i],poly4[i+1],poly5[i],poly5[i+1]);
getch(); cleardevice();

//ROTATION
printf("\n\n\t original image!");
drawpoly(n+1,poly);
drawpoly(n+1,poly1); for(i=0;i<2*n;i+=2)
line(poly[i],poly[i+1],poly1[i],poly1[i+1]);
getch();
outtextxy(10,10,"Enter the rotation factor"); gotoxy(30,3);
scanf("%d%d%d",&xr,&yr,&a);
for(i=0;i<2*n;i+=2)
{ poly6[i]=xr+((poly[i]-xr)*cos(a))-((poly[i+1]-yr)*sin(a))+70;
poly6[i+1]=yr+((poly[i+1]-yr)*cos(a))+((poly[i]-xr)*sin(a))+70;
poly7[i]=xr+((poly[i]-xr)*cos(a))-((poly1[i+1]-yr)*sin(a))+70;
poly7[i+1]=yr+((poly[i+1]-yr)*cos(a))+((poly1[i]-xr)*sin(a))+70;
}
poly6[2*n]=poly6[0];
poly6[2*n+1]=poly6[1];
poly7[2*n]=poly7[0];
poly7[2*n+1]=poly7[1];
drawpoly(n+1,poly6);
drawpoly(n+1,poly7); for(i=0;i<2*n;i+=2)
```

```
line(poly6[i],poly6[i+1],poly7[i],poly7[i+1]);
getch(); cleardevice(); }
```

Input:

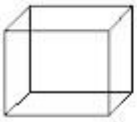
Enter the number of vertices of the polygon:
4

Enter the(x,y) co-ordinates of the vertices:

100 100 150 100 150 150 100 150

Output

Original image



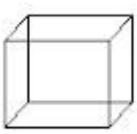
Translation

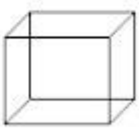
Input:

Enter the Translation Factor 200 200

Output

Original image





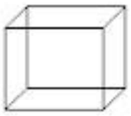
Scaling

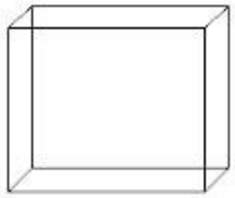
Input:

Enter the scaling Factor 2

Output

Original image





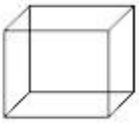
Rotation

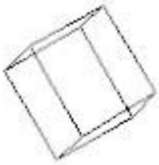
Input:

Enter the Reflection Factor: 250 250 60

Output

Original image





RESULT:

Thus the program to implement various 3D transformations like translation, scaling & rotation was executed.

9. VISUALIZE THE PROJECTION OF 3D IMAGES.

AIM:

To write a program to visualize the projection of 3D images.

ALGORITHM:

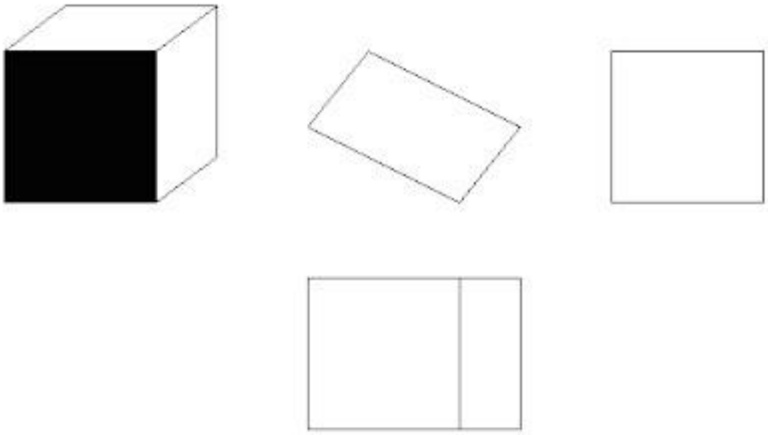
1. Draw the 3D bar using the inbuilt function bar3d.
2. Draw the front, side & top elevation of the 3D image by projecting the 3D image using the parallel line projection.
3. Display the 3D image along with the front, side & top view.

SOURCE CODE:

```
#include<graphics.h>
void top()
{
moveto(100,200); outtext("TOP
VIEW");
line(400,200,440,150);
line(400,200,300,150);
line(300,150,340,100);
line(340,100,440,150);
}
void side()
{
  moveto(100,200);
outtext("SIDE VIEW");
rectangle(500,100,600,200);
}

void front(int wid)
{
 moveto(100,200);
outtext("FRONT VIEW");
rectangle(400,250,400+wid,350);
 rectangle(300,250,400,350);
 }

 void main()
 {
 int x =DETECT,y,i,mx,my,wid=40;
 initgraph(&x,&y,"");
 mx=100;
my=100;
```

```
 setfillstyle(USER_FILL,3);
moveto(100,250);  outtext("3D
IMAGE");
bar3d(mx,my,mx+100,my+10
0,wid,1);
 getch();
cleardevice();
top();  getch();
cleardevice();
side();
getch();
cleardevice();
front(wid);
getch();
 cleardevice();
 bar3d(mx,my,mx+100,my+100,wid,1);
 line(400,200,440,150);
line(400,200,300,150);  line(300,150,340,100);
line(340,100,440,150);
rectangle(500,100,600,200);
rectangle(400,250,400+wid,350);
rectangle(300,250,400,350);
 getch();
closegraph();
  }
```

Output:



RESULT: Thus the program to visualize the projection of 3D images was executed.

10. CONVERSIONS BETWEEN COLOR MODELS

AIM:

To write the program to convert HSV color to RGB color model and vice versa.

ALGORITHM:

HSV to RGB

1. Read the H, S, V values in the range 0 to 1.
2. If the value of s is 0 then it is gray scale and the R, G, B becomes the value of V.
3. If the value of h is 1.0 then assign h=0 else h=h*6.0 and perform the following
   i= floor(h);   f=h-i;
   aa=v*(1-s);   bb=v*(1-s*f);   cc=v*(1-s*(1f)));
4. Based on the i value assign v,aa,bb,cc to RGB and display the RGB values.
   RGB to HSV 1.   Read the R, G, B values.

2. Find the min,max value among the RGB values
3. Assign the maximum value to V.
4. S value is calculated by (max-min)/max.
5. H value is calculated by comparing R, G, and B values to max value.
6. Display the H, S and V values.

SOURCE CODE:

Program 1:// To convert HSV TO RGB

```
#include<stdio.h>
#include<math.h> #include<conio.h>
void hsvtorgb(float h,float s,float v,float *r,float *g,float *b)
{
int i; float aa,bb,cc,f; if(s==0)
*r=*g=*b=v; else { if(h==1.0)
h=0; h*=6.0; i=floor(h); f=h-i;
aa=v*(1-s); bb=v*(1-(s*f));
cc=v*(1-(s*(1-f))); switch(i) {
case 0: *r=v; *g=cc; *b=aa; break;
case 1: *r=bb; *g=v; *b=aa; break;
case 2: *r=aa; *g=v; *b=cc; break;
case 3: *r=aa; *g=bb; *b=v; break;
case 4: *r=cc; *g=aa; *b=v; break;
case 5: *r=v; *g=aa; *b=bb; break;
}
} } void
main() {
float h,s,v,r=0,g=0,b=0;
clrscr();
printf("Enter the H,S,V values:\n");
scanf("%f%f%f",&h,&s,&v);
hsvtorgb(h,s,v,&r,&g,&b); printf("The R,G,B
values:\n%f %f %f\n",r,g,b);
getch(); }
```

Input:

Enter the H, S, V values: 0.1 0.2 0.3

Output:

```
 The R, G, B values:

  0.300000    0.276000     0.240000
```

Program 2: //To Convert RGB to HSV

```
#include<math.h>
#include<stdio.h>
#include<conio.h>
#define MAX(a,b)(a>b?a:b) #define MIN(a,b)(a<b?a:b)
void rgbtohsv(float r,float g,float b,float *h,float *s,float *v)
{
float max = MAX(r,MAX(g,b));
float min = MIN(r,MIN(g,b));
float delta=max-min; *v=max;
if(max!=0.0) {
*s=delta/max;
if(r==max) *h=(g-
b)/delta; else
if(g==max)
*h=2+(b-r)/delta;
else if(b==max)
*h=4+(b-r)/delta;
*h*=60.0; if(*h<0)
*h+=360.0;
*h/=360.0;
}
}

void main()
{
float r,g,b,h=0,s=0,v=0;
clrscr();
printf("Enter the value of R,G,B\n:");
scanf("%f%f%f",&r,&g,&b);
rgbtohsv(r,g,b,&h,&s,&v); printf("H=%f
, S=%f ,V=%f",h,s,v);
getch(); }
```

Input:

```
 Enter the R, G, B values:     0.3    0.276      0.24
```

Output:

```
The H, S, V values:

 0.100000    0.200000      0.300000
```

RESULT:

Thus the program to convert HSV color to RGB color models and vice versa was executed. **7(A)-**

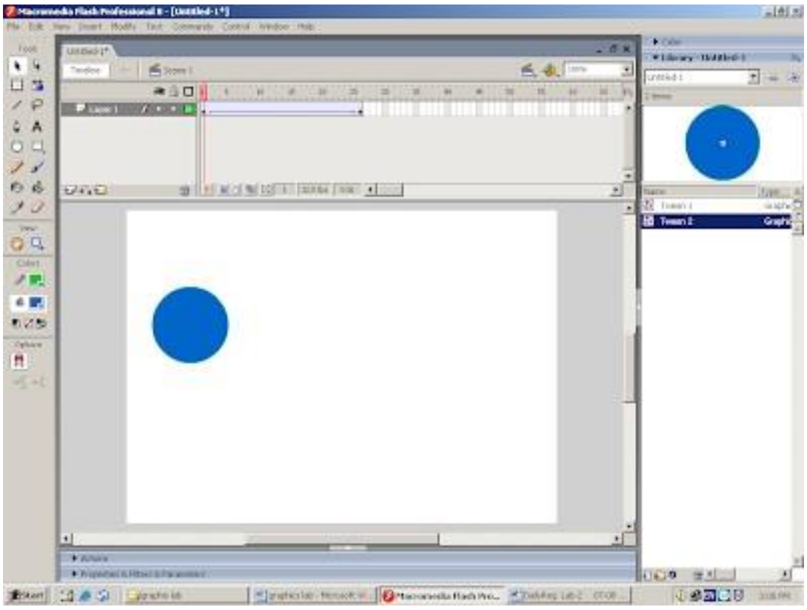
11. MOTION TWEENING

AIM:

To create motion tweening of an object.

ALGORITHM:

1. Select the layer and place the ball by drawing with the help of tools.
2. Select the frames by pressing F6 or right click and select insert frame.
3. Click the layer and right click & create motion tween.
4. Move the ball in the screen to the required destination point.
5. Press ctrl + Enter
6. Enter to show it in full screen


Macromedia Flash Professional 8 - [Untitled-1*]
File Edit View Insert Modify Text Commands Control Window Help
Untitled-1*
Timeline
Scene 1
Layer 1
Library - Untitled-1
Tween 1
Tween 2
Graphic

RESULT:

Thus the motion tweening of an object has been implemented and the output was verified.

12-SHAPE TWEENING – OBJECT AND TEXT

AIM:

To create shape tweening of an object and text**.**

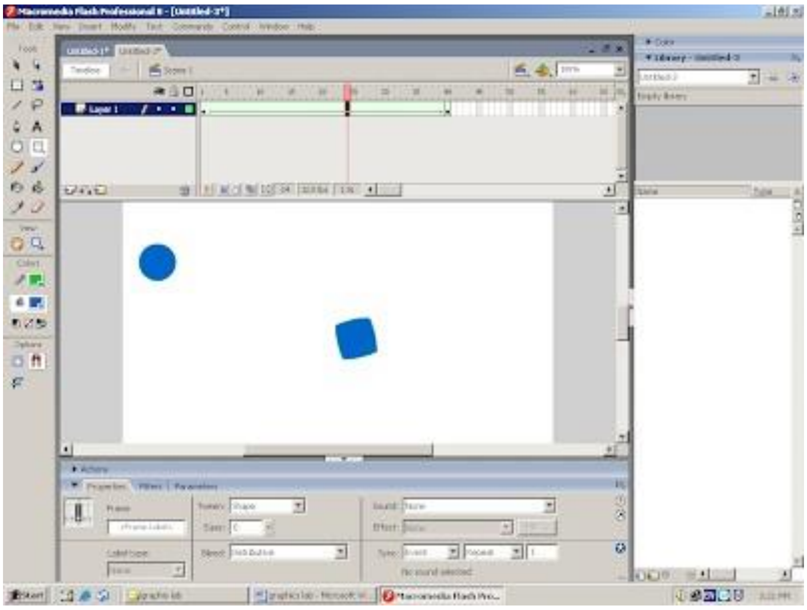
ALGORITHM:

Object:

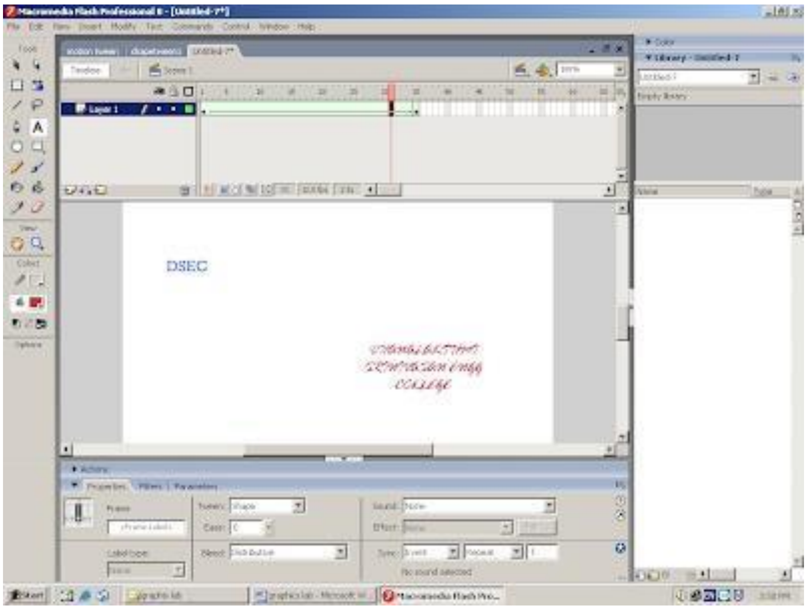
1. Select the layer and place a ball.
2. Select the frame by pressing F6 (or) right click the mouse and select insert frame.
3. In the same layer, create another object say rectangle.
4. click the layer and in properties change the tween to shape.
5. Press ctrl+Enter to show it in full screen.

Text:

1. Select the text from the tool box and place it
2. Press ctrl twice a time.
3. Select the frame by pressing F6 or right click the mouse and select insert frame.
4. Select the layer and in properties change the tween as shape.
5. Press ctrl+enter to show it in full screen.



Shape tweening of text :



Tween: Shape

Ease: 0

Blend: Distributive

Sound: None

Sync: Event

Repeat: 1

No sound selected

RESULT:

Thus the shape tweening-text and object has been implemented and the output was verified.

13-GUIDE LAYER

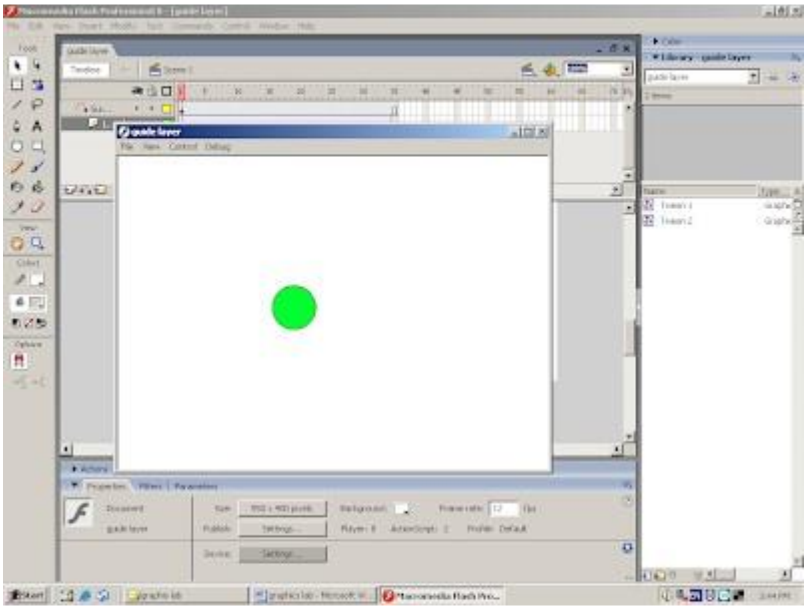
AIM:

To create an animation using guide Layer.

ALGORITHM:

1. Select the layer and draw an object.
2. Select the frame by pressing F6.
3. Select the guide layer in the layer options.
4. With the help of pencil tool, draw the path.
5. Move the object over the path.
6. Select the first layer and select create motion tween by right clicking the mouse.
7. Press Ctrl+Enter to show it in full screen.

OUTPUT:



RESULT:

Thus the guide layer has been implemented and the output was verified.

14- MASKING

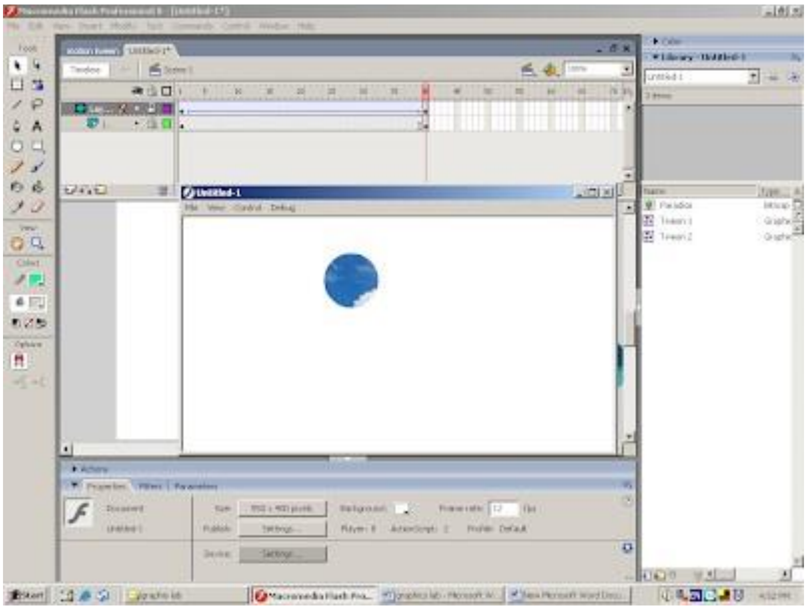
AIM:

To implement masking concept

ALGORITHM:

1. Select the text.
2. Select the frame by pressing F6.
3. Insert another new layer and select create motion tween.
4. Right click the frame and select create motion tween.
5. Move the ball over select the mark 6. Press Ctrl+Enter to show it in full screen.

OUTPUT:



RESULT:

Thus the masking has been implemented and the output was verified.