DEV BHOOMI INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering

Year: 3rd **Semester:6th**

Computer Graphics- PCS-551

LAB MANUAL

Prepared By: HOD(CSE)

DEV BHOOMI INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering

INDEX

S. No	Practical's Name	S/w used	Remark
1	Implementation of line generation using slope's method	Turbo C/C++	
2	Implementation of Circle generation using Mid Point method and Bresenham's method.	Turbo C/C++	
3	Implementation of ellipse generation using Mid Point Method.	Turbo C/C++	
4	Implementation of Polygon filling using flood fill, boundary fill, and scan line algorithms.	Turbo C/C++	
5	Implementation of 2D transformation: Translation, Scaling, Rotation, Mirror reflection and Shearing	Turbo C/C++	
6	Implementation of Line Clipping using Cohen- Sutherland Algorithm and Bisection method.	Turbo C/C++	
7	Implementation of Polygon Clipping using Sutherland- Hodgeman Algorithm.	Turbo C/C++	

DEV BHOOMI INSTITUTE OF TECHNOLOGY

LAB MANUAL

	Course Name: Computer Graphics.	Experiment No. 1	
	Course Code : PCS-551 **Faculty :** Mr. Himanshu dewedi	**Branch:** CSE	**Semester:** V

AIM: Implementation of line generation using slope's method, DDA, and Bresenham's Algorithm.

Algorithm:

1. Read the end point coordinates (x1, y1) and (x2, y2) for a line.
2. *dx*= x2- x1

   dy= y2-y1
3. If abs (*dx*) > abs (*dy*) then

   Step= abs (*dx*)

Otherwise

   Step= abs (*dy*)

4. xinc= *dx* /step

   yinc= *dy* / step

   x= x1

   y= y1
5. putpixel (x, y, color)
6. x= x+ xinc

   y= y+ yinc

   putpixel (x, y, color)
7. Repeat step 6 until x= x2 and y= y2.

ii) **Bresenham's Algorithm for Line Drawing:** The end points for this algorithm are required as integers. This algorithm entirely implemented with integer arithmetic. As integer arithmetic is much faster than floating- point arithmetic so this algorithm is faster than DDA algorithm. Furthermore this algorithm doesn't require any multiplication or

division. This algorithm can be divided into two parts one for slope $|m| < 1$ and another $|m| > 1$.

Algorithm:

Case 1: For $|m| < 1$.

Step 1: Read (x1, y1) and (x2, y2) as the endpoint coordinates.

Step 2: dx = $|x2 - x1|$

dy = $|y2 - y1|$

p= 2* dy- dx

Step 3: If p < 0 then

If (x1 < x2)

x1= x1+ 1

else

x1= x1- 1

p= p+ 2* dy

otherwise

if (y1 < y2)

y1= y1 + 1

else

y1 = y1 – 1

if (x1 < x2)

x1= x1 + 1.

else

x1 = x1 – 1

p= p + 2* (dy – dx)

Step 4: putpixel (x, y, color)

Step 5: Repeat steps 3 and 4 until x1 = x2.

Algorithm:

Case 2: For $|m| > 1$.

Step 1: Read (x1, y1) and (x2, y2) as the endpoint coordinates.

Step 2: dx = |x2- x1|

dy = |y2 – y1|

p= 2* dy- dx

Step 3: p= 2* dx – dy

Step 4: if p < 0 then

If (y1 < y2)

y1= y1 + 1

else

y1= y1 – 1

p = p + 2 * dx

otherwise

if (y1 < y2)

y1 = y1 + 1

else

y1= y1 – 1

if (x1 < x2)

x1 = x1 + 1

else

x1 = x1 – 1

p = p + 2 * (dy – dx)

Step 5: putpixel (x, y, color)

Step 6: Repeat steps 4 and 5 until y1 = y2.

Programs:

```
//THE SIMPLE DDA PROGRAM:
#include <graphics.h>
#include <dos.h>
#include <stdio.h>
#include <conio.h>
```

```
  void main()
  {
/* request auto detection */
int gdriver = DETECT, gmode, errorcode,i;
float x1,x2,y1,y2,l,dx,dy;
float x,y,xinc,yinc;

/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "F:\\TC\\BGI");

/* read result of initialization */
//errorcode = graphresult();
/* an error occurred */
  printf("\n Enter the coordinates of the end points of the line");
  scanf("%f%f%f%f",&x1,&y1,&x2,&y2);
  dy=y2-y1;
  dx=x2-x1;
  if(abs(dy)>=abs(dx))
  l=abs(dy);
  else
  l=abs(dx);
  xinc=dx/l;
  yinc=dy/l;
  x=x1;
  y=y1;
  for(i=1;i<l;i++)
  {
  x=x+xinc;
  y=y+yinc;
  putpixel(ceil(x),ceil(y),10);
  }
```

```
getch();
}
```

Outcome:

To understand the implementation of line generation algorithms.

DEV BHOOMI INSTITUTE OF TECHNOLOGY

LAB MANUAL

	Course Name: Computer Graphics.	Experiment No. 2	
	Course Code : PCS-551 **Faculty :** Mr. Himanshu dewedi	**Branch:** CSE	**Semester:** V

AIM: Implementation of Circle generation using Mid Point method and Bresenham's method.

Algorithm:

1. Read the radius r of the circle.
2. Initialize starting positions as

   X= 0

   Y= r
3. Calculate the initial values of decision parameter as

   p= 1.25 - r
4. do

   {

   plot (x, y)

   if (d<0)

   {

   x= x+1

   y=y

   d=d + 2*x +1

   }

   else

   {

   x= x+1

   y= y- 1

   d=d + 2*x + 2*y +1

   }

   while(x<y)

5. Determine symmetry points.
6. stop.

The Bresenham's Circle Drawing Algorithm:

1. Read radius r.
2. x=0
   y=r
   p= 3-2*r
3. If p<0 Then
   x= x+ 1
   p= p+ 4*x +6
   otherwise
   y= y-1
   x=x+1
   p=p+4*(x-y) +10
4. putpixel(x, y, color)
5. Repeat steps 3 and 4 until x<= y.

Programs:

BRESENHAM'S CIRCLE ALGORITHEM:-

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<complex.h>
#include<math.h>
#include<stdlib.h>
void main()
{
  int x1,y1,x,y,r,p,gdriver=DETECT,gmode,errorcode;
  initgraph(&gdriver,&gmode,"c:\\tc\\bgi");
  errorcode=graphresult();
```

```
if(errorcode!=grOk)
{
  cout << "graphics error is: %s" << grapherrormsg(errorcode);
  cout << "hit a key to halt" ;
  getch();
  exit(1);
}
  cleardevice();
cout << "enter the value of x1,y1,r" << endl;
cin >>x1>>y1>>r;
x=0;
y=r;
p=3 - 2 * r;
while(x<=y)
{
  putpixel(x1+x,y1+y,2);
  putpixel(x1-x,y1+y,2);
  putpixel(x1+x,y1-y,2);
  putpixel(x1-x,y1-y,2);
  putpixel(x1+y,y1+x,2);
  putpixel(x1-y,y1+x,2);
  putpixel(x1+y,y1-x,2);
  putpixel(x1-y,y1-x,2);
  if(p<0)
  p+=4* x++ +6;
  else
  p+=4* (x++-y--) +10;
}
getch();  }
```

Outcome: To understand the implementation of Circle generation algorithms.

DEV BHOOMI INSTITUTE OF TECHNOLOGY

LAB MANUAL

	Course Name: Computer Graphics.	Experiment No. 3	
	Course Code : PCS-551 **Faculty :** Mr. Himanshu dewedi	**Branch:** CSE	**Semester:** V

AIM: Implementation of ellipse generation using Mid Point Method.

Algorithm:

1. Read redii r_x and r_y.
2. Initialize starting point as

   x= 0

   y= r_y
3. Calculate the initial value of decision parameter in region1 as $d_1= r^2_y- r^2_x r_y +0.25 * r^2_x$
4. Initialize dx and dy as

   dx= $2*r^2_y x$

   dy= $2*r^2_x y$
5. do

   {

   plot (x, y)

   if ($d_1<0$)

   {

   x=x+1

   y=y

   dx=dx+ $2r^2_y$

   $d_1=d_1+ dx+ r^2_y$

   }

   else

   {

   x=x+1

y=y- 1

dx=dx+ $2r^2_y$

dy=dy- $2r^2_x$

$d_1=d_1$+ dx- dy+ r^2_y

}While(dx< dy)

6. Calculate the initial value of decision parameter in region2 as $d_2= r^2_y(x+0.5)2+ r^2_x$ (y-1)2 - $r^2_x r^2_y$
7. do

{

plot (x, y)

if (d_2>0)

{

x=x

y=y- 1

dy=dy- $2r^2_x$

$d_2=d_2$- dy+ r^2_x

}

else

{

x=x+1

y=y- 1

dx=dx+ $2r^2_y$

dy=dy- $2r^2_x$

$d_2=d_2$+ dx- dy+ r^2_x

}While(y>0)

8. Determine symmetrical points in other three quadrants.
9. Stop.

Program:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
 void ellipsemidpoint(int xcenter, int ycenter, int rx, int ry);
 void  plotpoint(int x,int xcenter, int ycenter,int y);
int maxx, maxy,xcenter,ycenter,rx,ry;
int main(void)
{
  int gdriver=DETECT, gmode, errorcode;
  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
  errorcode = graphresult();
    if (errorcode != grOk)
  {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);
  }
  maxx = getmaxx();
  maxy = getmaxy();
  xcenter= 250;
  ycenter= 250;
  rx= 50;
```

```
  ry= 50;
  ellipsemidpoint(xcenter,ycenter,rx,ry);
  getch();
closegraph();
 return(0);
}
 void ellipsemidpoint( int xcenter, int ycenter,int rx, int ry)
 {
  int p,px,py,x,y,ry2,rx2,tworx2,twory2;
   ry2= ry * ry;
   rx2= rx * rx;
   twory2= 2 * ry2;
   tworx2= 2 * rx2;
   x= 0;
   y= ry;
   plotpoint(x, xcenter,  ycenter, y);

  p= abs(ry2 - ry2 * ry + ( 0.25 * rx2));
  px= 0;
  py= tworx2 * y;
   while(px<py)
            {
              x= x+1;
              px= px + twory2;
               if(p>=0)
                {
                  y= y-1;
                  py= py - tworx2;
                 }
              if(p<0)
               {
```

```
            p= p + ry2 + px;
            }
     else
          {
           p= p + ry2 + px;
          }
    plotpoint( x, xcenter,  ycenter, y);
   }
 p= abs(ry2 * (x + 6.5) * (x + 6.5) + rx2 * (y - 1) * (y - 1) - rx2 * ry2);
while (y>0)
          {
           y= y-1;
           py= py - tworx2;
          if(p<0)
           {
            x= x +1;
            px= px + twory2;
           }
    if(p>0)
          {
           p= p + rx2 - py;
          }
  else
    {
          p= p + rx2 -py +px;
    plotpoint(x, xcenter,  ycenter, y);

    }
  }
}
 void plotpoint(int x,int xcenter, int ycenter,int y)
```

```
{

        putpixel (xcenter + x, ycenter + y, 2);
        putpixel (xcenter - x, ycenter + y, 2);
        putpixel (xcenter + x, ycenter - y, 2);
        putpixel (xcenter - x, ycenter - y, 2);
}
```

Outcome:

To understand the implementation of ellipse generation.

DEV BHOOMI INSTITUTE OF TECHNOLOGY

LAB MANUAL

	Course Name: Computer Graphics.	Experiment No. 4	
	Course Code : PCS-551 **Faculty :** Mr. Himanshu dewedi	**Branch:** CSE	**Semester:** V

AIM: Implementation of Polygon filling using flood fill, boundary fill, and scan line algorithms.

Algorithm:

1) Boundary Fill / Edge Fill Algorithm:

```
Boundary_fill (x, y, f_color, b_color)
{
if(getpixel(x, y) != b_color && getpixel(x, y) != f_color)
{   putpixel(x, y, f_color)
boundary_fill (x+1, y, f_color, b_color)
boundary_fill (x, y+1, f_color, b_color)
boundary_fill (x-1, y, f_color, b_color)
boundary_fill (x, y-1, f_color, b_color)
  }
}
```

2) Flood Fill Algorithm:

Sometimes it is required to fill in an area that is not defined within a single colour boundary. In such cases we can fill areas by replacing a specified interior colour instead for searching for a new boundary colour. This approach is called a flood fill algorithm.

```
flood_fill (x, y, old_color, new_color)
{
if( getpixel(x, y) = old_color)
{
 putpixel(x, y, new_color)
flood_fill  (x+1, y, old_color, new_color)
flood_fill  (x-1, y, old_color, new_color)
```

```
flood_fill  (x, y+1, old_color, new_color)
flood_fill  (x, y-1, old_color, new_color)
flood_fill  (x+1, y+1, old_color, new_color)
flood_fill  (x-1, y+1, old_color, new_color)
flood_fill  (x-1, y-1, old_color, new_color)
flood_fill  (x+1, y-1, old_color, new_color)
        }
}
```

3) Scan-line Algorithm for Polygon Filling:

1. Read number of vertices of polygon, i.e. n.
2. Read the x and y coordinates of all vertices as x[n] and y[n].
3. Find minimum and maximum values of y and store as ymin and ymax.
4. Store the initial x value (x1), y values y1 and y2 for two end points x increment by 1/m from scan- line to the line for each edge in the array edges [n][4], while doing this check that y1>y2, if not, interchange y1 and y2 and corresponding x1 and x2 so that for each edge, y1 represents its maximum y coordinate and y2 represents its minimum y coordinate.
5. Sort the rows of array edges [n][4] in descending order of y1.
6. Set y= ymax.
7. Find the active edges and update active edge list:
   If(y>=y1 and y1<y2)
   [edge is active]
   else
   [Edge is not active]
8. Compute the x-intersects for all active edges for current y values as $x_{i+1}= x_i+ 1/m$
9. If x-intersect is vertex, then apply vertex tests to check whether to consider one intersects or two intersects. Store all x-intersects in x-intersect [ ] array.
10. Sort x- intersect[ ] array in the ascending order.
11. Extract pairs of intersects from the sorted x- intersect[ ] array.
12. Pass pairs of values to line drawing routine to draw corresponding line segments.
13. Set y= y-1

14. Repeat steps 7 through 13 until y>= ymin.
15. Stop.
16.

Program:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
  /* request auto detection */
  int gdriver = DETECT, gmode, errorcode;
  int x1, y1,x2,y2;

  /* initialize graphics, local variables
*/
  initgraph(&gdriver, &gmode, "F:\\TC\\BGI");
  printf("Enter the coordinates of the rectangle");
  scanf("%d%d%d%d",&x1,&y1,&x2,&y2);

  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)
  /* an error occurred */
  {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);
    /* terminate with an error code */
```

```
  }

 // maxx = getmaxx();
 // maxy = getmaxy();
  /* select drawing color */
  setcolor(RED);
  /* select fill color */
  setfillstyle(SOLID_FILL, GREEN);
  /* draw a border around the screen */
  rectangle(x1,y1,x2,y2);
  /* draw some circles */
  circle(x1 / 3, y1 /2, 50);
  circle(x2 / 2, 20, 100);
  circle(x1-20, y1-50, 75);
  circle(20, y1-20, 25);
  /* wait for a key */
  getch();
  /* fill in bounded region */
  floodfill(2, 2, BLUE);
  /* clean up */
  getch();
  closegraph();
  return 0;
}
```

Outcome: To understand the implementation of Polygon filling algorithms.

DEV BHOOMI INSTITUTE OF TECHNOLOGY

LAB MANUAL

	Course Name: Computer Graphics.	Experiment No. 5	
	Course Code : PCS-551 **Faculty :** Mr. Himanshu dewedi	**Branch:** CSE	**Semester:** V

AIM: Implementation of 2D transformation: Translation, Scaling, Rotation, Mirror reflection and Shearing (write a menu driven program)

Program:

```
//Translation
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
int x1,y1,x2,y2,x,y,x3,y3,x4,y4,ch;
int main(void)
{
  int gdriver = DETECT, gmode, errorcode;
  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
  errorcode = graphresult();
  if (errorcode != grOk)
  {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);
  }
  do
  {
```

```
getch();
cleardevice();
cout<<"    ##############MAIN-MENU################\n";
cout<<"               TRANSLATION\n";
cout<<"                1.LINE\n";
cout<<"                2.RECTANGLE\n";
cout<<"                3.TRIANGLE\n";
cout<<"enter your choice:0 for exit:\n";
cin>>ch;
switch(ch)
{
case 1:              cout<<"enter the values of line coordinates:";
                        cin>>x1>>y1>>x2>>y2;
                        cout<<"enter the translation coordinates:";cin>>x>>y;
                        cleardevice();
                        line(x1,y1,x2,y2);
                        cout<<"now hit a key to see translation:";
                        getch();
                        line(x1+x,y1+y,x2+x,y2+y);
                        break;

case 2:              cout<<"enter the top left coordinates:";
                        cin>>x1>>y1;
                        cout<<"enter the bottom right coordinates:";
                        cin>>x2>>y2;
                        cout<<"enter the values of translation coordinates:\n";
                        cin>>x>>y;
                        cleardevice();
                        rectangle(x1,y1,x2,y2);
                        cout<<"now hit a key to see translation:";
                        getch();
```

```
                rectangle(x1+x,y1+y,x2+x,y2+y);
                break;
case 3:         cout<<"enter coordinates of line1:\n";
                cin>>x1>>y1>>x2>>y2;
                cout<<"enter coordinates for relative line:\n";
                cin>>x3>>y3;
                cout<<"enter translation coordinates:\n";cin>>x>>y;
                cleardevice();
                line(x1,y1,x2,y2);
                moveto(x2,y2);
                lineto(x3,y3);
                moveto(x3,y3);
                lineto(x1,y1);
                cout<<"now hit a key to see translation:";
                getch();
                moveto(x1+x,y1+y);
                lineto(x2+x,y2+y);
                moveto(x2+x,y2+y);
                lineto(x3+x,y3+y);
                moveto(x3+x,y3+y);
                lineto(x1+x,y1+y);
                break;
        case 0: break;
        default:cout<<"invalid choice";break;
        }}while(ch!=0);
        getch();
        closegraph();
        }

    //rotation
    #include <graphics.h>
```

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include <math.h>
float x1,y1,x2,y2,x,y,x3,y3,x4,y4,a;
int ch;
int main(void)
{
  int gdriver = DETECT, gmode, errorcode;
  clrscr();
  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
  errorcode = graphresult();
  if (errorcode != grOk)
  {
       printf("Graphics error: %s\n", grapherrormsg(errorcode));
       printf("Press any key to halt:");
       getch();
       exit(1);
  }
  do
  {
  cout<<"    ##############MAIN-MENU################\n";
  cout<<"                ROTATION\n";
  cout<<"                1.LINE\n";
  cout<<"                2.RECTANGLE\n";
  cout<<"                3.TRIANGLE\n";
  cout<<"enter your choice:0 for exit:\n";
  cin>>ch;
  switch(ch)
```

```
{
case 1:          cout<<"enter the values of line coordinates:";
                    cin>>x1>>y1>>x2>>y2;
                 cout<<"enter the value for angle of rotation:";
                 cin>>a;
                 cleardevice();
                 line(x1,y1,x2,y2);
                 a=a*(3.14/180);
                 x1=(x1*cos(a))-(y1*sin(a));
                 y1=(x1*sin(a))+(y1*cos(a));
                 x2=(x2*cos(a))-(y2*sin(a));
                 y2=(x2*sin(a))+(y2*cos(a));
                 cout<<"now hit a key to see rotation:";
                 getch();
                 line(x1,y1,x2,y2);
                 break;

case 2:          cout<<"enter the top left coordinates:";
                 cin>>x1>>y1;
                 cout<<"enter the bottom right coordinates:";
                 cin>>x2>>y2;
                 cout<<"enter the value for angle of rotation:";
                 cin>>a;
                 cleardevice();
                 rectangle(x1,y1,x2,y2);
                 a=a*(3.14/180);
                 x1=(x1*cos(a))-(y1*sin(a));
                 y1=(x1*sin(a))+(y1*cos(a));
                 x2=(x2*cos(a))-(y2*sin(a));
                 y2=(x2*sin(a))+(y2*cos(a));
                 cout<<"now hit a key to see rotation:";
```

```
                getch();
                rectangle(x1,y1,x2,y2);
                break;
case 3:         cout<<"enter coordinates of line1:\n";
                cin>>x1>>y1>>x2>>y2;
                cout<<"enter coordinates for relative line:\n";
                cin>>x3>>y3;
                cout<<"enter the angle of rotation:\n";cin>>a;
                cleardevice();
                line(x1,y1,x2,y2);
                moveto(x2,y2);
                lineto(x3,y3);
                moveto(x3,y3);
                lineto(x1,y1);
                a=a*(3.14/180);
                x1=(x1*cos(a))-(y1*sin(a));
                y1=(x1*sin(a))+(y1*cos(a));
                x2=(x2*cos(a))-(y2*sin(a));
                y2=(x2*sin(a))+(y2*cos(a));
                x3=(x3*cos(a))-(y3*sin(a));
                y3=(x3*sin(a))+(y3*cos(a));
                cout<<"now hit a key to see rotation:";
                getch();
                moveto(x1,y1);
                lineto(x2,y2);
                moveto(x2,y2);
                lineto(x3,y3);
                moveto(x3,y3);
                lineto(x1,y1);
                break;
              case 0: break;
```

```
            default:cout<<"invalid choice";break;
            }}while(ch!=0);
            getch();
            closegraph();
            }


//shearing

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
int x1,y1,x2,y2,x,y,x3,y3,x4,y4,ch;
int main(void)
{
  int gdriver = DETECT, gmode, errorcode;
  clrscr();
  initgraph(&gdriver, &gmode, "F:\\tc\\bgi");
  errorcode = graphresult();
  if (errorcode != grOk)
  {
       printf("Graphics error: %s\n", grapherrormsg(errorcode));
       printf("Press any key to halt:");
       getch();
       exit(1);
  }
  do
  {
  cout<<"    ##############MAIN-MENU################\n";
```

```
cout<<"                    SHEARING\n";
cout<<"                     1.LINE\n";
cout<<"                     2.RECTANGLE\n";
cout<<"                     3.TRIANGLE\n";
cout<<"enter your choice:0 for exit:\n";
cin>>ch;
switch(ch)
{
case 1:          cout<<"enter the values of line coordinates:";
                 cin>>x1>>y1>>x2>>y2;
                 cout<<"enter the value of shearing for xaxis:";
                 cin>>x;
                 cout<<"enter the value of shearing for y-axis:";
                 cin>>y;
                 cleardevice();
                 line(x1,y1,x2,y2);
                 cout<<"now hit a key to see shear in x_axis:";
                 getch();
                 line(x1,y1,x2*x,y2);
                 cout<<"\nnow hit a key to see shear in y_axis:";
                 getch();
                 line(x1,y1,x2,y2*y);
                 break;

case 2:          cout<<"enter the top left coordinates:";
                 cin>>x1>>y1;
                 cout<<"enter the bottom right coordinates:";
                 cin>>x2>>y2;
                 cout<<"enter the values of shearing coordinate for x_shear:\n";
                 cin>>x;
                 cout<<"enter the values of shearing coordinate for y_shear:\n";
```

```
                cin>>y;

                cleardevice();
                rectangle(x1,y1,x2,y2);
                cout<<"now hit a key to see shear in x_axis:";
                getch();
                rectangle(x1,y1,x2*x,y2);
                cout<<"\nnow hit a key to see shear in y_axis:";
                getch();
                rectangle(x1,y1,x2,y2*y);
                break;
case 3:         cout<<"enter coordinates of line1:\n";
                cin>>x1>>y1>>x2>>y2;
                cout<<"enter coordinates for relative line:\n";
                cin>>x3>>y3;
                cout<<"enter shear coordinate for x_shear:\n";cin>>x;
                cout<<"enter shear coordinate for y_shear:\n";cin>>x;

                cleardevice();
                line(x1,y1,x2,y2);
                moveto(x2,y2);
                lineto(x3,y3);
                moveto(x3,y3);
                lineto(x1,y1);
                cout<<"\nnow hit a key to see shear in x_axis:";
                getch();
                moveto(x1,y1);
                lineto(x2*x,y2);
                moveto(x2*x,y2);
                lineto(x3*x,y3);
                moveto(x3*x,y3);
```

```
                    lineto(x1,y1);
                    cout<<"\nnow hit a key to see shear in y_axis:";
                    getch();
                    moveto(x1,y1);
                    lineto(x2,y2*y);
                    moveto(x2,y2*y);
                    lineto(x3,y3*y);
                    moveto(x3,y3*y);
                    lineto(x1,y1);
                    break;
  case 0: break;
  default:cout<<"invalid choice";break;
  }}while(ch!=0);
  getch();
  closegraph();
  }
```

Outcome:

To understand the implementation of 2D transformation: Translation, Scaling, Rotation, Mirror reflection.

DEV BHOOMI INSTITUTE OF TECHNOLOGY

LAB MANUAL

	Course Name: Computer Graphics.	Experiment No. 6	
	Course Code : PCS-551 **Faculty :** Mr. Himanshu dewedi	**Branch:** CSE	**Semester:** V

AIM: Implementation of Line Clipping using Cohen- Sutherland Algorithm and Bisection method.

Algorithm:

1. Establish region code for all line end points.
    - High order bit is set to 1 if point is above the window.
    - Next to high order bit is set to 1 if the point is below the window.
    - Previous to the lower order bit is set to 1 if the point is right to the window.
    - And lower order bit is set to 1 if the point is left to the window.

2. Determine which lines are completely inside the window and which are completely outside the window. We can do this using following tests:
    a) If both the end points of the line having region codes "0000" line is completely inside the window.
    b) If logical AND of the region codes of the two endpoints is <> "0000" line is completely outside. In the other way we can say that at same bit position of the two end points having "1" line is completely outside.
3. If both the tests in step 2 failed then the line is not completely inside nor completely outside. So we find out the intersections with the boundary of the window.

    m= (Y2 – Y1)/ (X2- X1)

    a) If bit1 is "1" line intersects with the left boundary

    Yi=Y1+m*(X – X1)

    Where X= Xwmin

    b) If bit2 is "1" line intersects with the right boundary

Yi=Y1+m*(X – X1)

Where X= Xwmax

c) If bit3 is “1” line intersects with the bottom boundary

Xi= X1+ (Y-Y1)/m

Where Y= Ywmin

d) If bit4 is “1” line intersects with the bottom boundary

Xi= X1+ (Y-Y1)/m

Where Y= Ywmax

Where Xi and Yi are the X and Y intercepts for that line.

4. Repeat step1 to step3 until line is completely accepted or rejected.

Program:

```
/*- Line clipping using cohen sutherland algo -*/
#include<stdio.h>
#include<graphics.h>
#include<conio.h>

typedef unsigned int outcode;
enum {          TOP=0x1, BOTTOM=0x2, RIGHT=0x4, LEFT=0x8 };
int calcode(float,float,float,float,float,float);
void lineclip(float x0, float y0, float x1, float y1, float xwmin, float ywmin,
float xwmax, float ywmax )
float x0,y0,x1,y1,xwmin,ywmin,xwmax,ywmax;
{
  int gd,gm;
  outcode code0,code1,codeout;
  int accept = 0, done=0;
  code0 = calcode(x0,y0,xwmin,ywmin,xwmax,ywmax);
  code1 = calcode(x1,y1,xwmin,ywmin,xwmax,ywmax);

  do{
      if(!(code0 | code1))
```

```
{
accept =1 ; done =1; }
else
 if(code0 & code1)     done = 1;
 else
 {
   float x,y;
   codeout = code0 ? code0 : code1;
   if(codeout & TOP)
   {
         x = x0 + (x1-x0)*(ywmax-y0)/(y1-y0);
         y = ywmax;
   }
   else
    if( codeout & BOTTOM)
    {
          x = x0 + (x1-x0)*(ywmin-y0)/(y1-y0);
          y = ywmin;
    }
    else
         if ( codeout & RIGHT)
         {
           y = y0+(y1-y0)*(xwmax-x0)/(x1-x0);
           x = xwmax;
         }
         else
         {
            y = y0 + (y1-y0)*(xwmin-x0)/(x1-x0);
            x = xwmin;
          }
 if( codeout == code0)
```

```
                {
                   x0 = x;  y0 = y;
                   code0=calcode(x0,y0,xwmin,ywmin,xwmax,ywmax);
                }
                else
                {
                  x1 = x;   y1 = y;
                  code1 = calcode(x1,y1,xwmin,ywmin,xwmax,ywmax);
                }
             }
          } while( done == 0);
          if(accept)  line(x0,y0,x1,y1);
          rectangle(xwmin,ywmin,xwmax,ywmax);
          getch();
}
/****************** COHEN-SUTHERLAND LINE CLIPPING ***************/
 int  calcode (float x,float y,float xwmin,float ywmin,float xwmax,float ywmax)
 float  x,y,xwmin,ywmin,xwmax,ywmax;
 {
   int  code =0;
   if(y> ywmax)
         code |=TOP;
   else if( y<ywmin)
         code |= BOTTOM;
   else if(x > xwmax)
         code |= RIGHT;
   else if ( x< xwmin)
         code |= LEFT;
   return(code);
}
main()
```

```
{
        float x2,y2,x1,y1,xwmin,ywmin,xwmax,ywmax;
        int gd,gm;
        detectgraph(&gd,&gm);
        initgraph(&gd,&gm,"F:\\TC\\BGI");
        printf("\n\n\tEnter the co-ordinates of Line :");
        printf("\n\n\tX1  Y1  :  ");
        scanf("%f %f",&x1,&y1);
        printf("\n\n\tX2  Y2  :  ");
        scanf("%f %f",&x2,&y2);
        printf("\n\tEnter the co_ordinates of window  :\n ");
        printf("\n\txwmin , ywmin  : ");
        scanf("%f %f",&xwmin,&ywmin);
        printf("\n\txwmax , ywmax  : ");
        scanf("%f %f",&xwmax,&ywmax);
        line(x1,y1,x2,y2);
        rectangle(xwmin,ywmin,xwmax,ywmax);
        getch();
        cleardevice();
        lineclip(x1,y1,x2,y2,xwmin,ywmin,xwmax,ywmax );
        getch();
        closegraph();
        }
```

Assignment Problem

1> WAP for implementing Cohen Hodgeman clipping algorithm.

Outcome:

To understand the implementation of Line Clipping.

DEV BHOOMI INSTITUTE OF TECHNOLOGY

LAB MANUAL

	Course Name: Computer Graphics.	Experiment No. 7	
	Course Code : PCS-551 **Faculty :** Mr. Himanshu dewedi	**Branch:** CSE	**Semester:** V

AIM: Implementation of Polygon Clipping using Sutherland- Hodgeman Algorithm.

Algorithm:

1. Read the coordinates of the vertices in X[i] and Y[i] array. i varies from 1 to n. Where n is the number of vertices.

2. Check each vertex against each boundary of the window for which side it lies. It can be done as below:

Lower left corner of the Window (Xwmin, Ywmin)

Upper right corner of the Window (Xwmax, Ywmax).

a) If X[i] < Xwmin then the vertex lies on the left side of the window. And X= Xwmin, go to Step3.

b) If X[i] > Xwmax then the vertex lies on the right side of the window. And X= Xwmax go to step3.

c) If Y[i] > Ywmin then the vertex lies on the bottom of the window. And Y= Ywmin go to step3.

d) If Y[i] > Ywmax then the vertex lies on the top of the window. And Y= Ywmax go to step3.

3. Calculate the intersection with the respective boundary using m and m1 (slope) as:

X[0]= X[n]

X[n+1]= X[1]

Y[0]= Y[n]

Y[n+1]= Y[1]

M =(Y[i] – Y[I-1])/ (X[i]- X[i-1])

M1= (Y[i] – Y[I+1])/ (X[i]- X[i +1])

a) Intersection with the respective boundaries (left and right) are

Y1[i]= Y[i] + m(X- X[i])

Y2[i]= Y[i] + m1(X- X[i])

b) Intersection with the bottom and top boundaries are:

X1[i]= X[i] + (Y- Y[i])/m

X2[I]= X[I] + (Y- Y[i])/m1

4. Draw the lines using these intersections.

Program:

```
/*Program to implement Ploygon Clipping */
#include<graphics.h>
#include<stdio.h>
#include<conio.h>
#include<dos.h>
union REGS i,o;
struct pt
{
\       int x,y;
\};
float xl,xr,yt,yb,m,slope[20];
\int bc=0,xc,yc,n=0,k,dy,dx,x,y,temp,a[20][2],xi[20];
struct point
\{
\       float x,y;
};
\enum bound {left,right,bottom,top};
int inside(struct point p, enum bound b)
{       int c=1;
switch(b)       {
        case left  :  if(p.x<xl)
                                c=0;
```

```
                        break;
  case right  : if(p.x>xr)
                        c=0;
                        break;
  case bottom : if(p.y>yb)
                        c=0;
                        break;
  case top    : if(p.y<yt)
                        c=0;
                        break;
  }
  return(c);
}
struct point intersect (struct point p1,struct point p2,enum bound b)
{
  struct point t;
  float m=0;
  if(p2.x!=p1.x) m=(p2.y - p1.y)/(p2.x-p1.x);
  switch(b)
  {
        case left :  t.x =xl;
                              t.y = p2.y+(xl-p2.x)*m;
                              break;
        case right:  t.x=xr;
                              t.y = p2.y + (xr-p2.x)*m;
                              break;
        case bottom: t.y = yb;
                              if(p1.x==p2.x) t.x=p2.x;
                              else t.x = p2.x +(yb-p2.y)/m;
                              break;
        case top   : t.y = yt;
```

```
                                if(p1.x==p2.x) t.x=p2.x;
                                else t.x = p2.x +(yt-p2.y)/m;
                                break;
}
return t;
}
initmouse()
{
        i.x.ax=0;
        int86(0x33,&i,&o);
        return(o.x.ax);
}
showmouseptr()
{
        i.x.ax=1;
        int86(0x33,&i,&o);
        return(0);
}
hidemouseptr()
{
        i.x.ax=2;
        int86(0x33,&i,&o);
        return(0);
}
getmousepos(int *button,int *x,int *y)
{
        i.x.ax=3;
        int86(0x33,&i,&o);
        *button=o.x.bx;
        *x=o.x.cx;
        *y=o.x.dx;
```

```
}
void main()
{
enum bound b;
int cou,i,gd=DETECT,gm,flag;
struct point p[30],pout[30],z;
initgraph(&gd,&gm,"c:\\tc\\bgi");
cleardevice();
showmouseptr();
while(bc!=2)    //poly
{
        getmousepos(&bc,&xc,&yc);
        if(bc==1)
        {
                p[n].x=xc;
                p[n].y=yc;
                n++;
                hidemouseptr();
                if(n>1)
                line(p[n-2].x,p[n-2].y,xc,yc);
                showmouseptr();
                delay(200);
        }
}
p[n]=p[0];
hidemouseptr();
line(p[n-1].x,p[n-1].y,p[n].x,p[n].y);
showmouseptr();
getmousepos(&bc,&xc,&yc);
flag=1;
bc=0;
```

```
while(bc!=2)      //window
{
        if((bc==1) && (flag==1))
        {
                xl=xc;
                yt=yc;
                flag=2;
                delay(200);
        }
        else
        {
                xr=xc;
                yb=yc;
                delay(200);
        }
        getmousepos(&bc,&xc,&yc);
}
hidemouseptr();
rectangle(xl,yt,xr,yb);
getch();
for(b=left;b<=top;b++)
{
        cou =-1;
        for(i=0;i<n;i++)
        if((inside(p[i],b)==0) && (inside(p[i+1],b)==1))
        {
                z=intersect(p[i],p[i+1],b);
                pout[++cou] =z;
                pout[++cou]=p[i+1];
        }
        else
```

```
        if((inside(p[i],b)==1)&&(inside(p[i+1],b)==1)) pout[++cou]=p[i+1];
        else
        if((inside(p[i],b)==1)&&(inside(p[i+1],b)==0))
        {
                z = intersect(p[i],p[i+1],b);
                pout[++cou]=z;
        }
        pout[++cou]=pout[0];
        n=cou;
        for(i=0;i<=n;i++) p[i]=pout[i];
}
getch();
cleardevice();
rectangle(xl,yt,xr,yb);
for(i=0;i<n;i++) line(p[i].x,p[i].y,p[i+1].x,p[i+1].y);
getch();
for(i=0;i<=n;i++)
{
        a[i][0]=p[i].x;
        a[i][1]=p[i].y;
}
getch();
}
```

Outcome:

To understand the implementation of Polygon Clipping using Sutherland- Hodgeman Algorithm.