COMPUTER GRAPHICS

LAB PRACTICALS RECORD

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Dr. B R AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY
JALANDHAR – 144011, PUNJAB (INDIA)

Submitted To:

Ms. Jagriti
Asst. Professor
Department of CSE

Submitted By:

Nikhil Bansal
13103011
6th Semester

PROGRAM 1

DDA LINE ALGORITHM

Description:

In computer graphics, a digital differential analyzer (DDA) is hardware or software used for linear interpolation of variables over an interval between start and end point. DDAs are used for rasterization of lines, triangles and polygons. In its simplest implementation, the DDA algorithm interpolates values in interval by computing for each x_i the equations

$$x_i = x_{i-1} + 1/m$$
$$y_i = y_{i-1} + m$$
$$\Delta x = x_{end} - x_{start} \text{ and } \Delta y = y_{end} - y_{start} \text{ and } m = \Delta y/\Delta x.$$

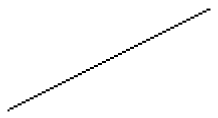
Program:

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

int main(int argc,char *argv[])
{
  if(argc<5){
    printf("Enter coordinates of end points of line on commandine\n");
    return -1;
  }

  //coordinates output file
  FILE *coordinates=fopen("coordinates", "w");

  //commandline input
  int i,j,x1,x2,y1,y2;
  float currx,curry;
  x1=atoi(argv[1]);
  y1=atoi(argv[2]);
  x2=atoi(argv[3]);
  y2=atoi(argv[4]);

  //if coordinates are not in increasing order of x then make them
  if(x1>x2){
    int temp=x1;
    x1=x2;
    x2=temp;
    temp=y1;
    y1=y2;
    y2=temp;
```

```
}

//graphics initialise
int gd = DETECT,gm;
initgraph(&gd,&gm,NULL);
setbkcolor(WHITE);
setcolor(BLACK);

//find the slope
float m=((float)y2-y1)/(x2-x1);

if(m<=1&&m>=-1){
  putpixel(x1,y1,getcolor());
  currx=x1;
  curry=y1;
  while(currx!=x2){
    currx+=1;
    curry=curry+m;
    fprintf(coordinates, "%d %d\n",(int)currx,(int)curry);
    putpixel((int)currx,(int)curry,getcolor());
  }
}

if(m>1||m<-1){
  if(m>1){
    putpixel(x1,y1,getcolor());
    currx=x1;
    curry=y1;
    while(curry!=y2){
      curry+=1;
      currx=currx+1/m;
      fprintf(coordinates,"%d %d\n",(int)currx,(int)curry);
      putpixel((int)currx,(int)curry,getcolor());
    }
  }
  else{
    putpixel(x2,y2,getcolor());
    currx=x2;
    curry=y2;
    while(curry!=y1){
      curry+=1;
      currx=currx+1/m;
      fprintf(coordinates, "%d %d\n",(int)currx,(int)curry);
      putpixel((int)currx,(int)curry,getcolor());
    }
  }
```

```
   }

   delay(5000);
   closegraph();
   return 0;
}
```

Output:



PROGRAM 2

BRESNHAM'S LINE ALGORITHM

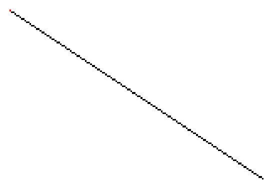
Description:

The Bresenham's line algorithm is an algorithm that determines the points of an *n*-dimensional raster that should be selected in order to form a close approximation to a straight line between two points. It is commonly used to draw lines on a computer screen, as it uses only integer addition, subtraction and bit shifting, all of which are very cheap operations in standard computer architectures. It is one of the earliest algorithms developed in the field of computer graphics. An extension to the original algorithm may be used for drawing circles.

Program:

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

// absolute i.e mod of x
int abs(int x){
  if(x<0)
    return -x;
  else
    return x;
}

// bresnham line algo used to draw line
// works only for |m|<1
int bresnhamLine(int x1, int y1, int x2, int y2){
  int p_curr,currx,curry;

  // coordinates output file
  FILE *coordinates=fopen("coordinates", "w");

  //slope
  float m=((float)y2-y1)/(x2-x1);
  int dx=abs(x2-x1);
  int dy=abs(y2-y1);
  putpixel(x1, y1, getcolor());

  // algorithm
  if(m<=1 && m>=-1){
    currx=x1;
    curry=y1;
    p_curr=2*dy-dx;
    putpixel(x1,y1,RED);
    if(m>=0){
      for(currx=x1+1;currx<=x2;currx++){
        if(p_curr>=0){
          curry++;
          p_curr=p_curr+2*dy-2*dx;
        }
        else{
          p_curr=p_curr+2*dy;
        }
        fprintf(coordinates,"%d %d\n",currx,curry);
        putpixel(currx, curry, getcolor());
      }
    }
    else{
      for(currx=x1+1;currx<=x2;currx++){
        if(p_curr>=0){
          curry--;
          p_curr=p_curr+2*dy-2*dx;
        }
        else{
          p_curr=p_curr+2*dy;
        }
        fprintf(coordinates,"%d %d\n",currx,curry);
        putpixel(currx,curry,RED);
        putpixel(currx+5, curry+5, getcolor());
      }
    }
  }

  fclose(coordinates);
  return 0;
}

int main(int argc,char *argv[]){
  // command line arguments check
  if(argc<5){
    printf("Enter coordinates of end points of line on commandine\n");
    return -1;
  }

  // commandline input
  int x1,x2,y1,y2;
  x1=atoi(argv[1]);
  y1=atoi(argv[2]);
  x2=atoi(argv[3]);
  y2=atoi(argv[4]);

  //if coordinates are not in increasing order of x then make them
  if(x1>x2){
    int temp=x1;
    x1=x2;
    x2=temp;
    temp=y1;
    y1=y2;
    y2=temp;
  }

  // Initialise graphics
  int gd = DETECT,gm;
  initgraph(&gd,&gm,NULL);
  setbkcolor(WHITE);
  setcolor(BLACK);

  // draw line
  bresnhamLine(x1,y1,x2,y2);

  // delay to able to view graphics
        delay(5000);
  closegraph();
        return 0;
}
```

Output:



PROGRAM 3

TRIGNOMETRIC CIRCLE

Description:

It is the basic algorithm used to draw circle. In this algo, we basically find the coordinates by using the trigonometry formulas. We find x and y coordinates by :

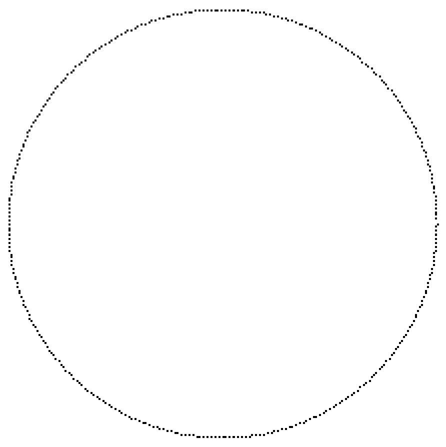
$$x = r * \cos(angle)$$
$$y = r * \sin(angle)$$

Program:

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

// draw the circle with given integer center and radius
int trignometricCircle(int x,int y,int radius){
   float curr_x,curr_y;
   int angle;
   FILE *coordinates=fopen("coordinates", "w");

   // algo
   for(angle=0;angle<360;angle++){
      curr_x=x+cos((float)angle/180*3.14)*radius;
      curr_y=y+sin((float)angle/180*3.14)*radius;
      putpixel((int)curr_x,(int)curr_y,getcolor());
      fprintf(coordinates, "%d %d\n", (int)curr_x, (int)curr_y);
   }
   return 0;
}

int main(int argc,char *argv[]){

   //command-line parameters check
   if(argc<3){
      printf("Enter 3 arguments on commandine\n");
      return 0;
   }

   //graphics initialisation
   int gd = DETECT,gm;
   initgraph(&gd,&gm,NULL);
   setbkcolor(WHITE);
   setcolor(BLACK);

   //get the center and radius
   int x,y,radius;
   x=atoi(argv[1]);
   y=atoi(argv[2]);
```

```
  radius=atoi(argv[3]);

  //Draw the circle using Trignometric algo
  trignometricCircle(x,y,radius);

  //delay so as to view the screen
  delay(5000);
          return 0;
}
```

Output:



PROGRAM 4

MID POINT CIRCLE

Description:

In computer graphics, the midpoint circle algorithm is an algorithm used to determine the points needed for drawing a circle. Bresenham's circle algorithm is derived from the midpoint circle algorithm. The algorithm can be generalized to conic sections. It is more efficient than trigonometric circle algorithm as it doesn’t use any trigonometric functions.
The algorithm is related to work by Pitteway and Van Aken.

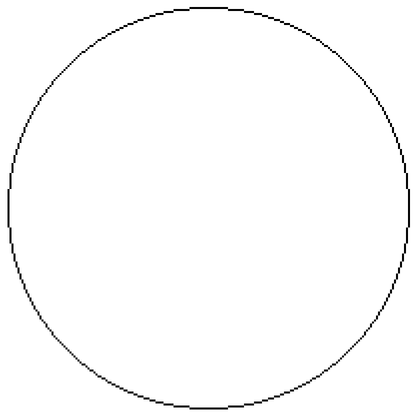
Program:

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

int midPointCircle(float x,float y,float radius){
        //coordinates output file
        FILE *coordinates=fopen("coordinates", "w");

        float pinit,pcurr;
        int curr_x,curr_y;

        // calculate the initial decision parameter
        if(floor(radius)-radius==0)
                pinit=1-radius;
        else
                pinit=5.00/4-radius;

        // initialisations
        curr_x=0;
        curr_y=floor(radius);
        pcurr=pinit;

        // operate while loop until x<y
        while(curr_x<=curr_y){
                // output points
                putpixel((int)(curr_x+x),(int)(curr_y+y),getcolor());
                putpixel((int)(-curr_x+x),(int)(curr_y+y),getcolor());
                putpixel((int)(curr_x+x),(int)(-curr_y+y),getcolor());
                putpixel((int)(-curr_x+x),(int)(-curr_y+y),getcolor());
                putpixel((int)(curr_y+y),(int)(curr_x+x),getcolor());
                putpixel((int)(-curr_y+y),(int)(curr_x+x),getcolor());
                putpixel((int)(curr_y+y),(int)(-curr_x+x),getcolor());
```

```
			putpixel((int)(-curr_y+y),(int)(-curr_x+x),getcolor());
			fprintf(coordinates,"%d %d\n",(int)(curr_x+x),(int)(curr_y+y));
			fprintf(coordinates,"%d %d\n",(int)(-curr_x+x),(int)(curr_y+y));
			fprintf(coordinates,"%d %d\n",(int)(curr_x+x),(int)(-curr_y+y));
			fprintf(coordinates,"%d %d\n",(int)(-curr_x+x),(int)(-curr_y+y));
			fprintf(coordinates,"%d %d\n",(int)(curr_y+y),(int)(curr_x+x));
			fprintf(coordinates,"%d %d\n",(int)(-curr_y+y),(int)(curr_x+x));
			fprintf(coordinates,"%d %d\n",(int)(curr_y+y),(int)(-curr_x+x));
			fprintf(coordinates,"%d %d\n",(int)(-curr_y+y),(int)(-curr_x+x));

			// algo
			if(pcurr<0){
				curr_x+=1;
				pcurr=pcurr+2*curr_x+1;
			}
			else{
				curr_x+=1;
				curr_y-=1;
				pcurr=pcurr+2*curr_x+1-2*curr_y;
			}
		}

		// close the output file
		fclose(coordinates);
		return 0;
}

int main(int argc,char *argv[]){

   //command-line parameters check
   if(argc<3){
      printf("Enter 3 arguments on commandine\n");
      return 0;
   }

		//get the center and radius
   float x,y,radius;
   x=atoi(argv[1]);
   y=atoi(argv[2]);
   radius=atof(argv[3]);

   // check if x and y are greater than radius else pixel out of range will be there
   if(x<radius||y<radius){
			printf("Circle cannot be displayed\nAs x and y are less than radius so there will be pixel out of
range.\n");
			return 0;
```

```
    }

    //graphics initialisation
    int gd = DETECT,gm;
    initgraph(&gd,&gm,NULL);
    setbkcolor(WHITE);
            setcolor(BLACK);

    //Draw the circle using Trignometric algo
    midPointCircle(x,y,radius);

    //delay so as to view the screen
    delay(5000);
            return 0;
}
```

Output:



PROGRAM 5

TRIGNOMETRIC ELLIPSE

Description:

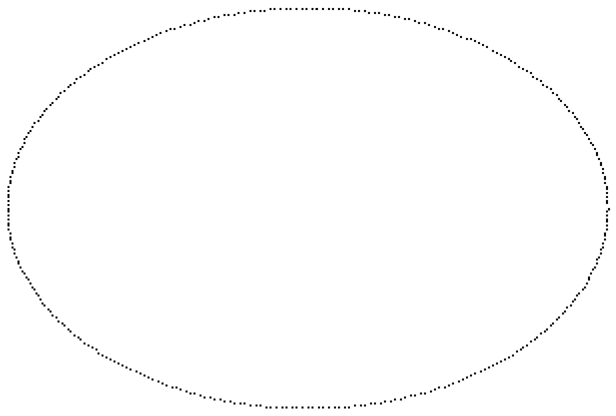
It is the basic algorithm used to draw circle. In this algo, we basically find the coordinates by using the trigonometry formulas. We find x and y coordinates by :

x = a * cos(angle)
y = b * sin(angle)

Program:

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

// draw the circle with given integer center and axes
int trignometricEllipse(int x,int y,int a,int b){
  float curr_x,curr_y;
  int angle;
  FILE *coordinates=fopen("coordinates", "w");

  // algo
  for(angle=0;angle<360;angle++){
    curr_x=x+a*cos((float)angle/180*3.14);
    curr_y=y+b*sin((float)angle/180*3.14);
    putpixel((int)curr_x,(int)curr_y,getcolor());
    fprintf(coordinates, "%d %d\n", (int)curr_x, (int)curr_y);
  }

  fclose(coordinates);
  return 0;
}

int main(int argc,char *argv[]){

  //command-line parameters check
  if(argc<4){
    printf("Enter 4 arguments on commandine\n");
    return 0;
  }

  //get the center and radius
  int x,y,a,b;
  x=atoi(argv[1]);
  y=atoi(argv[2]);
  a=atoi(argv[3]);
  b=atoi(argv[4]);

  // check for pixel out of range
  if(x<a||y<b){
    printf("Enter center of ellipse such that center points are less than a and b.\nElse therer will be pixel out of
range.\n");
    return 0;
  }

  //graphics initialisation
  int gd = DETECT,gm;
  initgraph(&gd,&gm,NULL);
  setbkcolor(WHITE);
  setcolor(BLACK);

  //Draw the ellipse using Trignometric algo
  trignometricEllipse(x,y,a,b);

  //delay so as to view the screen
  delay(5000);
  closegraph();
        return 0;
}
```

Output:



PROGRAM 6

MID POINT ELLIPSE

Description:

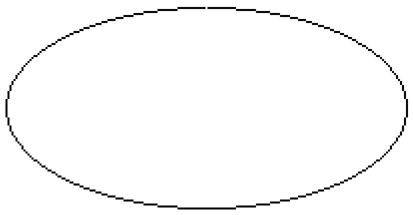
RSA In computer graphics, the midpoint ellipse algorithm is an algorithm used to determine the points needed for drawing a ellipse. The algorithm can be generalized to conic sections. It is more efficient than trigonometric ellipse algorithm as it doesn’t use any trigonometric functions. It considers the distance of the midpoint to the point on the ellipse and takes the nearest point.
The algorithm is related to work by Pitteway and Van Aken.

Program:

```
#include <bits/stdc++.h>
using namespace std;
#include <graphics.h>

// draw the circle with given integer center and axes
int midPointEllipse(int x,int y,int a,int b){
        int curr_x,curr_y,pcurr;
  FILE *coordinates=fopen("coordinates", "w");

  // initialisations
  curr_x=0;
  curr_y=b;
  pcurr=b*b-a*a*b+a*a/4;
  putpixel(x+curr_x, y+curr_y, getcolor());

  // region 1
  while(2*b*b*curr_x<2*a*a*curr_y){
    if(pcurr<0){
      curr_x++;
      pcurr=pcurr+2*b*b*curr_x+b*b;
    }
    else{
      curr_x++;
      curr_y--;
      pcurr=pcurr+2*b*b*curr_x-2*a*a*curr_y+b*b;
    }
    putpixel(x+curr_x, y+curr_y, getcolor());
    putpixel(x-curr_x, y-curr_y, getcolor());
    putpixel(x+curr_x, y-curr_y, getcolor());
    putpixel(x-curr_x, y+curr_y, getcolor());
    fprintf(coordinates, "%d %d\n", curr_x, curr_y);
  }

  // region 2
  pcurr=b*b*(curr_x+0.5)*(curr_x+0.5)+a*a*(curr_y-1)*(curr_y-1)-a*a*b*b;
  while(curr_y!=0){
    if(pcurr>0){
      curr_y--;
      pcurr=pcurr-2*a*a*curr_y+a*a;
    }
    else{
      curr_x++;
      curr_y--;
      pcurr=pcurr+2*b*b*curr_x-2*a*a*curr_y+a*a;
    }
    putpixel(x+curr_x, y+curr_y, getcolor());
    putpixel(x-curr_x, y-curr_y, getcolor());
    putpixel(x-curr_x, y+curr_y, getcolor());
    putpixel(x+curr_x, y-curr_y, getcolor());
    fprintf(coordinates, "%d %d\n", curr_x, curr_y);
  }

  fclose(coordinates);
        return 0;
}

int main(int argc,char *argv[]){

  //command-line parameters check
  if(argc<4){
    printf("Enter 4 arguments on commandine\n");
    return 0;
  }

  //get the center and radius
  int x,y,a,b;
  x=atoi(argv[1]);
  y=atoi(argv[2]);
  a=atoi(argv[3]);
  b=atoi(argv[4]);

  // check for pixel out of range
  if(x<a||y<b){
    printf("Enter center of ellipse such that center points are less than a and b.\nElse therer will be pixel out of
range.\n");
    return 0;
  }

  //graphics initialisation
  int gd = DETECT,gm;
  initgraph(&gd,&gm,NULL);
  setbkcolor(WHITE);
  setcolor(BLACK);

  //Draw the ellipse using Trignometric algo
  midPointEllipse(x,y,a,b);

  //delay so as to view the screen
  delay(5000);
```

```
        return 0;
}
```

Output:



PROGRAM 7

2D TRANSFORMATIONS

Description:

The Transformation means changing some graphics into something else by applying rules. We can have various types of transformations such as translation, scaling up or down, rotation, shearing, etc. When a transformation takes place on a 2D plane, it is called 2D transformation.
Transformations play an important role in computer graphics to reposition the graphics on the screen and change their size or orientation.

Program:

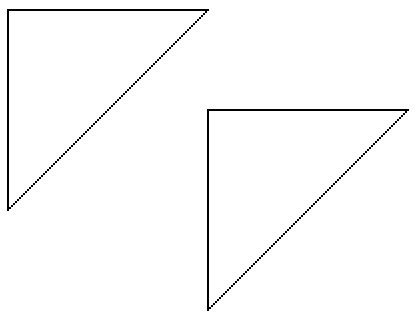
```
#include <bits/stdc++.h>
#include <graphics.h>
using namespace std;

class point{
public:
	int x,y;
	point(){
		x=0;
		y=0;
	}
	point(int x,int y){
		this->x=x;
		this->y=y;
	}
};

class translation{
public:
	int x,y;
	translation(){
		x=0;
		y=0;
	}
	translation(int x,int y){
		this->x=x;
		this->y=y;
	}
};

class rotation{
public:
	float angle;
	rotation(){
		angle=0;
	}
	rotation(int angle){
		this->angle=angle;
	}
};

class scale{
public:
	int x,y;
	scale(){
		x=0;
		y=0;
	}
	scale(int x, int y){
		this->x=x;
		this->y=y;
	}
};

point translatePoint(point p, translation t){
	p.x+=t.x;
	p.y+=t.y;
	return p;
}

point rotatePoint(point p, rotation r){
	point result;
	result.x=abs((p.x)*cos(r.angle)-(p.y)*sin(r.angle));
	result.y=abs((p.x)*sin(r.angle)+(p.y)*cos(r.angle));
	return result;
}

point scalePoint(point p, scale s){
	p.x=(s.x)*(p.x);
	p.y=(s.y)*(p.y);
	return p;
}

int plotPoint(point p){
	putpixel(p.x, p.y, getcolor());
	return 0;
}
```

```
int plotTriangle(point a, point b, point c){
        line(a.x, a.y, b.x, b.y);
        line(b.x, b.y, c.x, c.y);
        line(c.x, c.y, a.x, a.y);
        return 0;
}

int main(int argc,char *argv[]){
        FILE *input=fopen("input", "r");

        // initialisations
        point a,b,c,aFinal,bFinal,cFinal;
        translation t;
        rotation r;
        scale s;
        fscanf(input,"%d%d",&a.x,&a.y);
        fscanf(input,"%d%d",&b.x,&b.y);
        fscanf(input,"%d%d",&c.x,&c.y);
        fscanf(input,"%d%d",&t.x,&t.y);
        fscanf(input,"%f",&r.angle);
        fscanf(input,"%d%d",&s.x,&s.y);

        //graphics initialisation
        int gd = DETECT,gm;
        initgraph(&gd,&gm,NULL);
        setbkcolor(WHITE);
        setcolor(BLACK);

        // translation
        plotTriangle(a,b,c);
        aFinal=translatePoint(a,t);
        bFinal=translatePoint(b,t);
        cFinal=translatePoint(c,t);
        sleep(3);
        plotTriangle(aFinal,bFinal,cFinal);

        // rotation
        sleep(3);
        cleardevice();
        plotTriangle(a,b,c);
        aFinal=rotatePoint(a,r);
        bFinal=rotatePoint(b,r);
        cFinal=rotatePoint(c,r);
        sleep(3);
        plotTriangle(aFinal,bFinal,cFinal);
        // scaling
        sleep(3);
        cleardevice();
        plotTriangle(a,b,c);
        aFinal=scalePoint(a,s);
        bFinal=scalePoint(b,s);
        cFinal=scalePoint(c,s);
        sleep(3);
        plotTriangle(aFinal,bFinal,cFinal);

        //delay so as to view the screen and close the graph
        delay(5000);
        closegraph();
        fclose(input);
        return 0;
}
```

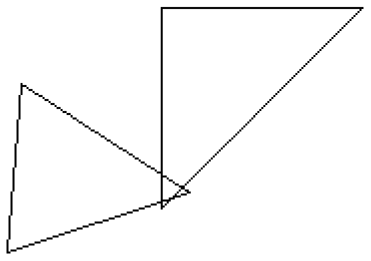
Output:

Points of Triangle: (100, 100), (100, 200), (200, 100).

Translation:
X: 100   Y: 50



Rotation:
Angle: 1 radian



Scaling:

X: 1 Y: 2