

A Laboratory Manual for

Computer Graphics Lab (210255)

Semester – IV

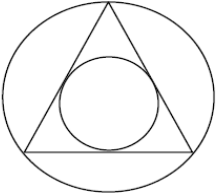
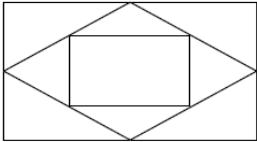
(Computer Engineering) Bachelor Degree in Engineering

Author: Prof. D.S.Shingate



SANDIP INSITUTE OF ENGINEERING & MANAGEMENT, NASHIK
DEPARTMENT OF COMPUTER ENGINEERING

List of Experiment

Group	Ass. No.	Problem Statement	
A	1	Write C++ program to draw line using DDA and Bresenham's algorithm.	
	2	Write C++ program to draw circle using Bresenham's algorithm.	
	3	Write C++ program to draw 2-D object and perform following basic transformations, a) Scaling b) Translation c) Rotation	
	4	Write C++ program to draw inscribed and Circumscribed circles in the triangle as shown as an example below. (Use any Circle drawing and Line drawing algorithms)	
	5	Write C++ program to draw the following pattern using DDA Line drawing algorithms	
B	6	Write C++ program for line drawing using Bresenham's algorithm with patterns such as solid, dotted, dashed, dash dot and thick.	
	7	Write C++ program to draw a convex polygon and fill it with desired color using Seed fill algorithm.	
	8	Write C++ program to implement reflection of 2-D object about X axis, Y axis and about X=Y axis. Also rotate object about arbitrary point given by user.	
	9	Generate Bouncing ball animation using Blender 3D	
	10	Write C++ program to generate Hilbert curve using concept of fractals.	
C	11	Write C++ program to simulate following scene- Clock with pendulum	

Assignment -1

Att	Perm	Oral	Total	Sign
(2)	(5)	(3)	(10)	

Title of Assignment: Draw a line.

Problem Definition: Write C++ program to draw line using DDA and Bresenham's algorithm.

1.1 Prerequisite: Basic primitives of computer graphics and concepts of OOP.

1.2 Learning Objective:

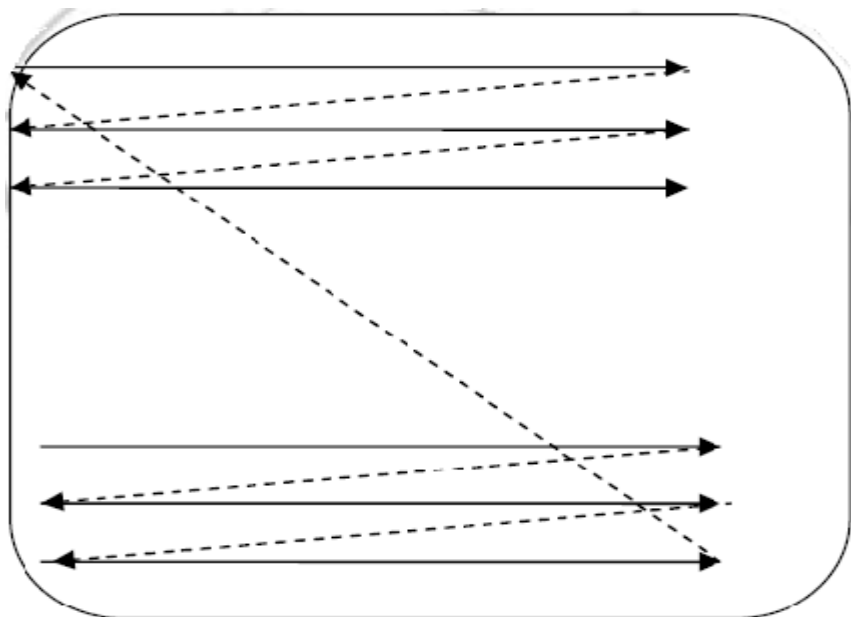
Understand the functions available in Graphics library.

1.3 Relevant Theory / Literature Survey:**Raster Scan Display:**

When the beam reaches the bottom of the screen, it is made OFF and rapidly retracted back to top left to start again. A display produced in this way is called **Raster Scan Display**.

Display.

The refresh process is independent of the complexity of the image.

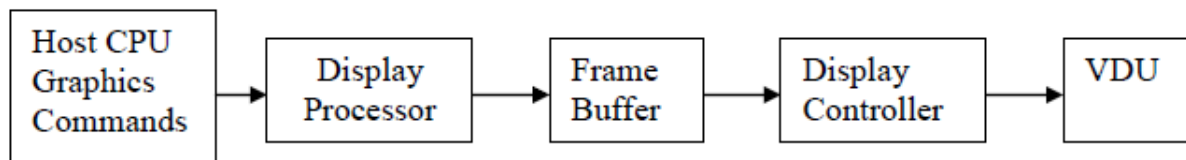


Sequences of operations performed in raster scan are performed through following components.

- a) Graphics Commands.
- b) Display Conversion (Scan Conversion)
- c) Frame buffer.
- d) Display Controller
- e) Visual Display Unit (VDU).

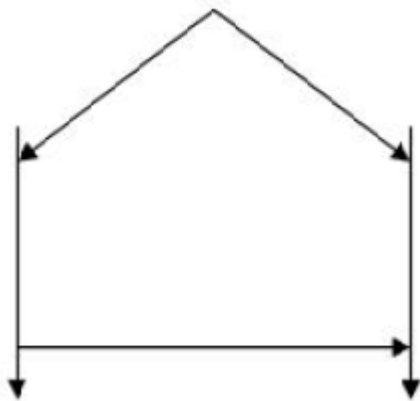
Characteristics:

- i. It is very cost effective, even inexpensive.
- ii. It has availability of large memory and has high refresh rate.
- iii. It has ability to display areas filled with solid colors and patterns.

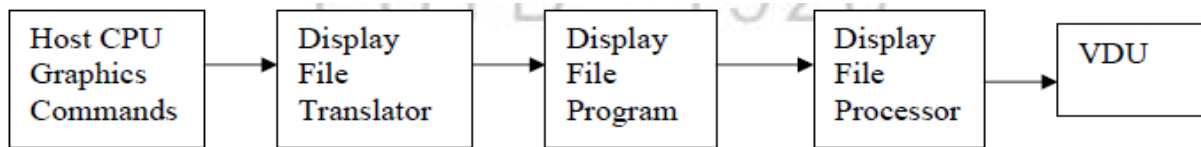
Block Diagram:**Vector Scan Display:**

In vector scan display, the beam is moved between the endpoints of graphics primitives.

It flickers when the number of primitives in the buffer becomes too large.

**Characteristics:**

- i. This is also called as **Random Scan Display**.
- ii. It draws a continuous and smooth line.
- iii. It only draws lines and characters and is more costly.

Block Diagram:**Line:**

It is the path between two end points.

Any point (x, y) on the line must follow the line equation:

Point is the fundamental element of the picture representation. It is nothing but the position in a plane defined as either pairs or triplets of numbers depending whether the data are two or three dimensional. Thus (x1, y1) or (x1, y1, z1) would represent a point two or three dimensional space.

Two points used to specify line by below equation

$y = m * x + b$, where

- **m is the slope of the line.**
- **b is a constant that represent the intercept on y-axis.**

If we have two end points, (x0, y0) and (x1, y1), then the **slope** of the line (**m**) between those points can be calculated as:

$$m = \Delta y / \Delta x = (y1 - y0) / (x1 - x0)$$

Draw a line with the help of line equation is very time consuming because it's required lots of calculation. A cathode ray tube (CRT) raster display is considered a matrix of discrete finite area cells (pixels), it is not possible to draw a straight line directly from one point to another. The process of determining which pixels provide the best approximation to the desired line is called rasterization.

To draw a line, we can use two algorithms:

- i) DDA (Digital Differential Analyzer)/Vector Generation Algorithm.**
- ii) Bresenham's Line Algorithm.**

1. DDA Line Drawing Algorithm:

Digital Differential Analyzer (DDA) algorithm is the simple line generation algorithm. It is the simplest algorithm and it does not require special skills for implementation. It is a faster method for calculating pixel positions than the direct use of equation $y=mx + b$. It eliminates

the multiplication in the equation by making use of raster characteristics, so that appropriate increments are applied in the x or y direction to find the pixel positions along the line path.

Disadvantages of DDA Algorithm

1. Floating point arithmetic in DDA algorithm is still time-consuming.
2. The algorithm is orientation dependent. Hence end point accuracy is poor.

Algorithm for DDA Line:

1. Start
2. Declare the variables and gDriver = DETECT and gMode
3. Initialize the graphics mode using `initgraph()`
4. Read the line end points (x_1, y_1) and (x_2, y_2) such that they are not equal.
5. Calculate the difference between two end points.

$$dx = x_1 - x_2$$

$$dy = y_1 - y_2$$

6. If $dx \geq dy$ and $x_1 \leq x_2$ then //gentle slope lines
Increment x by 1
Increment y by m
- Else if $dx \geq dy$ and $x_1 > x_2$ then //gentle slope lines
Increment x by -1
Increment y by -m
- Else if $dx < dy$ and $y_1 \leq y_2$ then //steep/sharp slope lines
Increment x by $1/m$
Increment y by 1
- Else if $dx < dy$ and $y_1 > y_2$ then //steep/sharp slope lines
Increment x by $-1/m$
Increment y by -1

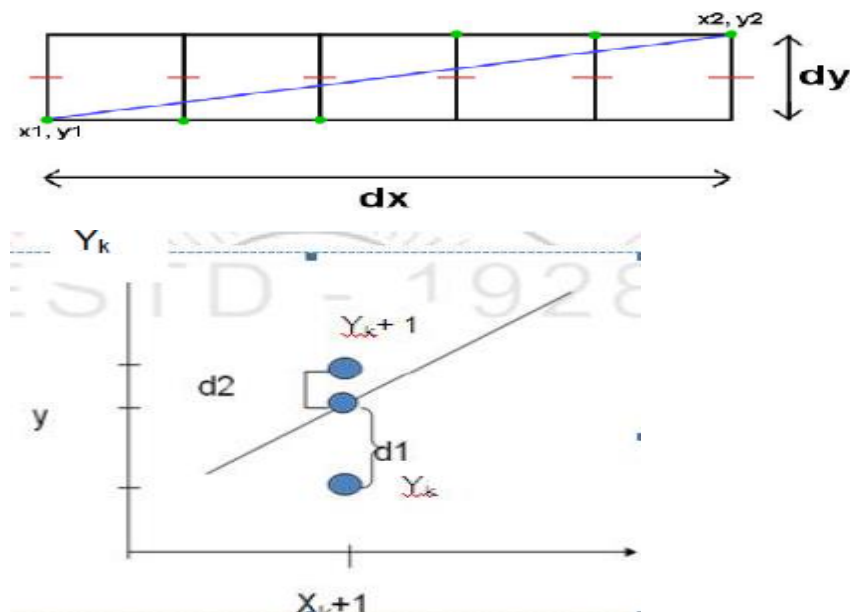
7. plot `xincrement` and `yincrement`
8. repeat step 6 until other end point is reach
9. Closegraph
10. Stop.

2. Bresenham's Line Drawing Algorithm:

Bresenham's is another incremental scan-conversion algorithm. It is named After Jack Bresenham who worked at IBM for 27 years before entering academics.

He had developed this algorithm in early 1960 at IBM..

The biggest advantage of this algorithm is that it uses only integer calculations. When we have a point (x_k, y_k) , then we must decide whether to draw the point (x_{k+1}, y_k) or (x_{k+1}, y_{k+1}) . Note that, at all cases, we move to x_{k+1} , still we must decide to move to y_k or y_{k+1} . The increment in the other variable is determined by examining the distance between the actual line location and the nearest pixel. The distance is called **decision variable** or the **error**.



As shown in figure the line does not pass through all raster points. It passes through raster point $(0,0)$. It is seen that the intercept of line X_{k+1} is closer to the line Y_{k+1} than to the line Y_k .

Algorithm for Bresenham's Line:

1. Start
2. Declare the variables and $gDriver = DETECT$ and $gMode$
3. Initialize the graphics mode using $initgraph()$
4. Read the line end points (x_1, y_1) and (x_2, y_2) such that they are not equal.
5. Calculate $dx = x_2 - x_1$ and $dy = y_2 - y_1$
6. $x = x_1$ and $y = y_1$ [Initialize starting point]
7. $e = 2 * dy - dx$ [Initialize value of decision variable]
8. $i = 1$

```

9. plot(x,y)
10. while(e>=0)
{
y=y+1
e=e - 2*dx
}
x=x+1
e=e + 2*dy
11. i=i+1
12. if(i<=dx) then go to step 9
13. Closegraph
14. Stop.

```

Comparison between Vector generation and Bresenham's Line Algorithm

Sr.No.	Vector Generation Algorithm/ DDA	Bresenham's
1	It uses Floating point arithmetic	It uses only integer addition, subtraction and multiplication by 2.
2	Due to floating point arithmetic and rounding function it takes more time.	It is quicker than vector generation algorithm
3	Less efficient	More efficient
4	Where speed is important this algorithm need to be implemented in hardware	Hardware implementation is not required.

1.4 Questions:-

- Q 1: Which algorithm is faster in DDA and Bresenham's?
- Q 2: State advantages of Bresenham's Line drawing algorithm?
- Q 3: Difference between DDA Line and Bresenham's line algorithm?
- Q 4: Which Algorithm is efficient for line drawing?
- Q 5: What is long form of DDA?
- Q 6: Which line is clearer DDA or Line generated by Equation?

1.5 Conclusion:-

In This way we have studied that how to draw a line using DDA and Bresenham's Algorithms.

Assignment - 2

Att	Perm	Oral	Total	Sign
(2)	(5)	(3)	(10)	

Title of Assignment: Draw a Circle.

Problem Definition: Write C++ program to draw circle using Bresenham's algorithm.

2.1 Prerequisite: Line Generation algorithms.

2.2 Learning Objective:

Understand the concept of circle generation algorithms.

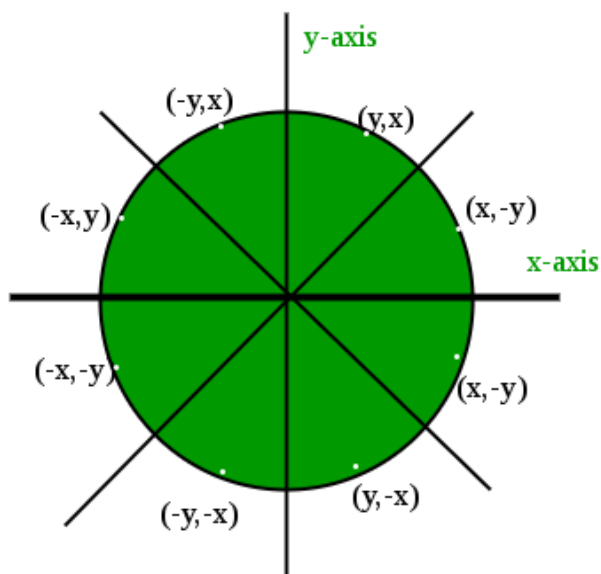
2.3 Relevant Theory / Literature Survey:

Drawing a circle on the screen is a little complex than drawing a line. There are two popular algorithms for generating a circle – **Bresenham's Algorithm** and **Midpoint Circle Algorithm**. These algorithms are based on the idea of determining the subsequent points required to draw the circle. Let us discuss the algorithms in detail –

The equation of circle is $X^2+Y^2=r^2$,

where r is radius.

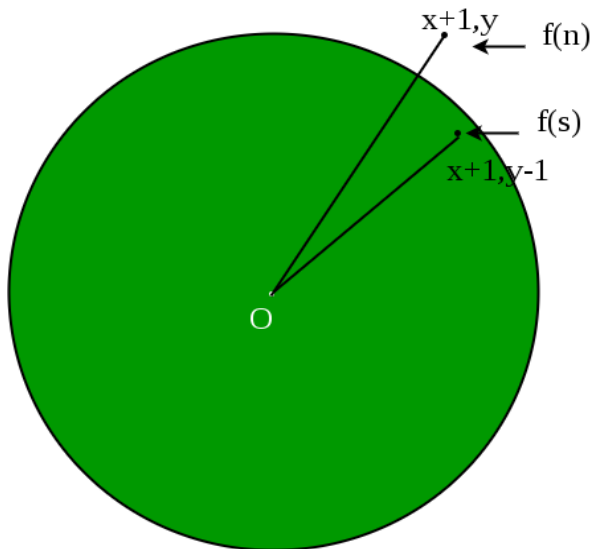
For every pixel (x, y) it calculates, we draw a pixel in each of the 8 octants of the circle as shown below,



For a pixel (x,y) all possible pixels in 8 octants.

Bresenham's Algorithm

Now, we will see how to calculate the next pixel location from a previously known pixel location (x, y) . In Bresenham's algorithm at any point (x, y) we have two options either to choose the next pixel in the east i.e. $(x+1, y)$ or in the south east i.e. $(x+1, y-1)$.



And this can be decided by using the decision parameter d as:

- If $d > 0$, then $(x+1, y-1)$ is to be chosen as the next pixel as it will be closer to the arc.
- else $(x+1, y)$ is to be chosen as next pixel.

Now to draw the circle for a given radius ' r ' and centre (x_c, y_c) We will start from $(0, r)$ and move in first quadrant till $x=y$ (i.e. 45 degree). We should start from listed initial condition:

$$d = 3 - (2 * r)$$

$$x = 0$$

$$y = r$$

Now for each pixel, we will do the following operations:

1. Set initial values of (x_c, y_c) and (x, y)
2. Set decision parameter d to $d = 3 - (2 * r)$.
3. Repeat steps 4 to 8 until $x \leq y$
4. call `drawCircle(int xc, int yc, int x, int y)` function.
5. Increment value of x .
6. If $d < 0$, set $d = d + (4 * x) + 6$
7. Else, set $d = d + 4 * (x - y) + 10$ and decrement y by 1.
8. call `drawCircle(int xc, int yc, int x, int y)` function

drawCircle() function:

```
// function to draw all other 7 pixels
// present at symmetric position
drawCircle(int xc, int yc, int x, int y)
{
    putpixel(xc+x, yc+y, RED);
    putpixel(xc-x, yc+y, RED);
    putpixel(xc+x, yc-y, RED);
    putpixel(xc-x, yc-y, RED);
    putpixel(xc+y, yc+x, RED);
    putpixel(xc-y, yc+x, RED);
    putpixel(xc+y, yc-x, RED);
    putpixel(xc-y, yc-x, RED);
}
```

2.4 Questions:-

Q 1: Explain the Bresenham's circle drawing algorithm?

Q 2: Explain the circle equation.

Q 3: Difference between Bresenham's and Mid point Circle drawing algorithm?

2.5 Conclusion:-

In This way we have studied that how to draw a circle on the screen.

Assignment- 3

Att	Perm	Oral	Total	Sign
(2)	(5)	(3)	(10)	

Title of Assignment: 2 D Transformations.

Problem Definition: Write C++ program to draw 2-D object and perform following basic transformations, a) Scaling b) Translation c) Rotation

3.1 Prerequisite: Basic primitives of computer graphics and concept of OOP.

3.2 Learning Objective:

How to transform given polygon in 2D.

3.3 Relevant Theory / Literature Survey:

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.

Translation

Translation of a data point is accomplished through the addition of factors to the x and y coordinates.

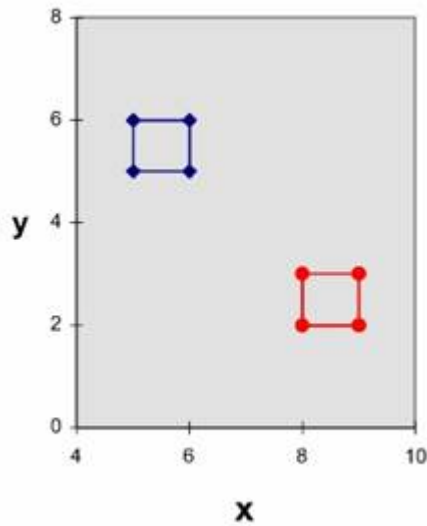
$P(x,y)$ becomes $P' (x',y')$

$$x' = x + T_x$$

$$y' = y + T_y$$

T_x and T_y are the shift in x and y coordinates.

Example: translating a square (Blue) by adding $T_x = 3$ to each x coordinate, and $T_y = -3$ to each y coordinate (Red).



Scaling

Scaling proceeds by multiplying the coordinate values by S_x and S_y , scaling factors in the x and y axis directions

$$x' = x S_x$$

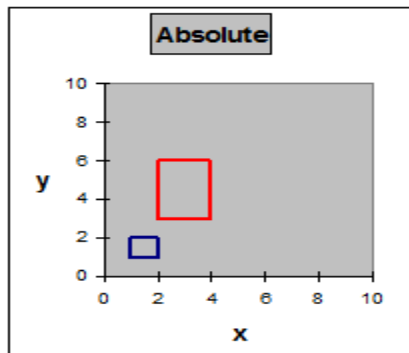
$$y' = y S_y$$

Example: scaling a square by $S_x = 2$ and $S_y = 3$.

Sx	Sy
2	3

Original Points	
x	y
1	1
2	1
2	2
1	2
1	1

Absolute Scaling	
x'	y'
2	3
4	3
4	6
2	6
2	3



Rotation

The matrix for rotating a point about an origin in a 2D plane is defined as:

$$R_{\beta} = \begin{bmatrix} \cos\beta & -\sin\beta \\ \sin\beta & \cos\beta \end{bmatrix}$$

Thus the rotation of a 2D vector in a plane is done as follows:

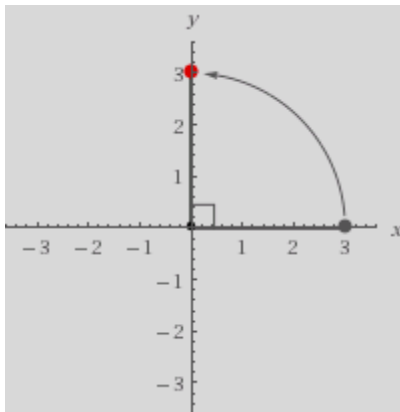
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

For example: To rotate a vector 90 degrees anticlockwise is done as follows:

$$\vec{u} = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \end{bmatrix}$$



Transformations as matrices

Scale:

$$X_{new} = s_x X_{old}$$

$$Y_{new} = s_y Y_{old}$$

$$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x \cdot x \\ s_y \cdot y \end{bmatrix}$$

Rotation:

$$x_2 = x_1 \cos \theta - y_1 \sin \theta$$

$$y_2 = x_1 \sin \theta + y_1 \cos \theta$$

$$\begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \cos \theta - y \sin \theta \\ x \sin \theta + y \cos \theta \end{bmatrix}$$

Translation:

$$X_{new} = X_{old} + t_x$$

$$Y_{new} = Y_{old} + t_y$$

$$\begin{bmatrix} t_x \\ t_y \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \end{bmatrix}$$

3.4 Questions:-

Q 1: State and explain different 2D transformation?

Q 2: Explain the concept of Homogeneous coordinates?

Q 3: Explain 2D Translation & Scaling?

Q.4 Explain the clockwise and anticlockwise 2D rotation?

3.5 Conclusion:-

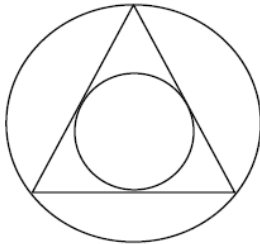
In This way we have studied that how to transform a given polygon in 2D.

Assignment- 4

Att	Perm	Oral	Total	Sign
(2)	(5)	(3)	(10)	

Title of Assignment: Drawing given using Circle drawing and Line drawing algorithms.

Problem Definition: Write C++ program to draw inscribed and Circumscribed circles in the triangle as shown as an example below



4.1 Prerequisite: Basic primitives of line generation & circle generation.

4.2 Learning Objective:

Understand how to draw different pattern using line generation algorithms and circle generation.

4.3 Relevant Theory / Literature Survey:**Line:**

It is the path between two end points.

Any point (x, y) on the line must follow the line equation:

Point is the fundamental element of the picture representation. It is nothing but the position in a plane defined as either pairs or triplets of numbers depending whether the data are two or three dimensional. Thus (x1, y1) or (x1, y1, z1) would represent a point two or three dimensional space.

Two points used to specify line by below equation

$$y = m * x + b, \text{ where}$$

- **m is the slope of the line.**
- **b is a constant that represent the intercept on y-axis.**

If we have two end points, (x0, y0) and (x1, y1), then the **slope** of the line (**m**) between those points can be calculated as:

$$m = \Delta y / \Delta x = (y_1 - y_0) / (x_1 - x_0)$$

Draw a line with the help of line equation is very time consuming because it's required lots of calculation. A cathode ray tube (CRT) raster display is considered a matrix of discrete finite area cells (pixels), it is not possible to draw a straight line directly from one point to another. The process of determining which pixels provide the best approximation to the desired line is called rasterization.

To draw a line, we can use two algorithms:

- i) DDA (Digital Differential Analyzer)/Vector Generation Algorithm.**
- ii) Bresenham's Line Algorithm.**

1. DDA Line Drawing Algorithm:

Algorithm for DDA Line:

1. Start
2. Declare the variables and gDriver = DETECT and gMode
3. Initialize the graphics mode using initgraph()
4. Read the line end points (x1,y1) and (x2,y2) such that they are not equal.
5. Calculate the difference between two end points.

$$dx = x_1 - x_2$$

$$dy = y_1 - y_2$$

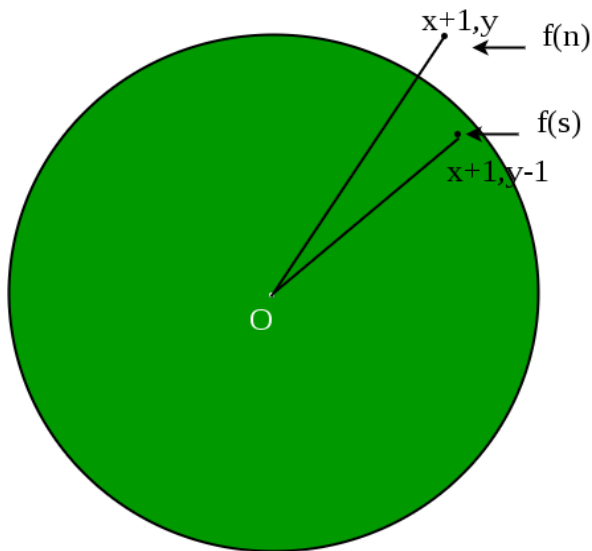
6. If $dx \geq dy$ and $x_1 \leq x_2$ then //gentle slope lines
 Increment x by 1
 Increment y by m
- Else if $dx \geq dy$ and $x_1 > x_2$ then //gentle slope lines
 Increment x by -1
 Increment y by -m
- Else if $dx < dy$ and $y_1 \leq y_2$ then //steep/sharp slope lines
 Increment x by 1/m
 Increment y by 1
- Else if $dx < dy$ and $y_1 > y_2$ then //steep/sharp slope lines
 Increment x by -1/m
 Increment y by -1

7. plot xincrement and yincrement

8. repeat step 6 until other end point is reach
9. Closegraph
10. Stop.

Bresenham's circle drawing algorithm

Now, we will see how to calculate the next pixel location from a previously known pixel location (x, y) . In Bresenham's algorithm at any point (x, y) we have two option either to choose the next pixel in the east i.e. $(x+1, y)$ or in the south east i.e. $(x+1, y-1)$.



And this can be decided by using the decision parameter d as:

- If $d > 0$, then $(x+1, y-1)$ is to be chosen as the next pixel as it will be closer to the arc.
- else $(x+1, y)$ is to be chosen as next pixel.

Now to draw the circle for a given radius ' r ' and centre (x_c, y_c) We will start from $(0, r)$ and move in first quadrant till $x=y$ (i.e. 45 degree). We should start from listed initial condition:

$$d = 3 - (2 * r)$$

$$x = 0$$

$$y = r$$

Now for each pixel, we will do the following operations:

9. Set initial values of (x_c, y_c) and (x, y)
10. Set decision parameter d to $d = 3 - (2 * r)$.
11. Repeat steps 4 to 8 until $x \leq y$

12. call drawCircle(int xc, int yc, int x, int y) function.
13. Increment value of x.
14. If $d < 0$, set $d = d + (4*x) + 6$
15. Else, set $d = d + 4 * (x - y) + 10$ and decrement y by 1.
16. call drawCircle(int xc, int yc, int x, int y) function

drawCircle() function:

```
// function to draw all other 7 pixels
// present at symmetric position
drawCircle(int xc, int yc, int x, int y)
{
    putpixel(xc+x, yc+y, RED);
    putpixel(xc-x, yc+y, RED);
    putpixel(xc+x, yc-y, RED);
    putpixel(xc-x, yc-y, RED);
    putpixel(xc+y, yc+x, RED);
    putpixel(xc-y, yc+x, RED);
    putpixel(xc+y, yc-x, RED);
    putpixel(xc-y, yc-x, RED);
}
```

4.4 Questions:-

- Q 1: Which algorithm is faster in DDA and Bresenham's?
- Q 2: State advantages of Bresenham's Line drawing algorithm?
- Q 3: Difference between DDA Line and Bresenham's line algorithm?
- Q 4: Which Algorithm is efficient for line drawing?
- Q 5: Explain the Bresenham's circle drawing algorithm?
- Q 6: Explain the circle equation.
- Q 7: Difference between Bresenham's and Mid point Circle drawing algorithm?

4.5 Conclusion:-

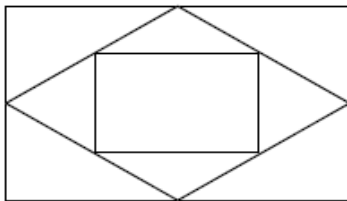
In This way we have studied that how to draw a given pattern of line using DDA Algorithm and circle using Bresenham's circle generation algorithm.

Assignment- 5

Att	Perm	Oral	Total	Sign
(2)	(5)	(3)	(10)	

Title of Assignment: Drawing given using DDA line generation algorithm.

Problem Definition: Write C++ program to draw the following pattern using DDA Line drawing algorithms



5.1 Prerequisite: Basic primitives of line generation.

5.2 Learning Objective:

Understand how to draw different pattern using line generation algorithms.

5.3 Relevant Theory / Literature Survey:

Line:

It is the path between two end points.

Any point (x, y) on the line must follow the line equation:

Point is the fundamental element of the picture representation. It is nothing but the position in a plane defined as either pairs or triplets of numbers depending whether the data are two or three dimensional. Thus (x1, y1) or (x1, y1, z1) would represent a point two or three dimensional space.

Two points used to specify line by below equation

$y = m * x + b$, where

– **m is the slope of the line.**

– **b is a constant that represent the intercept on y-axis.**

If we have two end points, (x0, y0) and (x1, y1), then the **slope** of the line (**m**) between those points can be calculated as:

$$\mathbf{m = \Delta y / \Delta x = (y1 - y0) / (x1 - x0)}$$

Draw a line with the help of line equation is very time consuming because it's required lots of calculation. A cathode ray tube (CRT) raster display is considered a matrix of discrete finite area cells (pixels), it is not possible to draw a straight line directly from one point to another. The process of determining which pixels provide the best approximation to the desired line is called rasterization.

To draw a line, we can use two algorithms:

i) DDA (Digital Differential Analyzer)/Vector Generation Algorithm.

ii) Bresenham's Line Algorithm.

1. DDA Line Drawing Algorithm:

Algorithm for DDA Line:

1. Start

2. Declare the variables and gDriver = DETECT and gMode

3. Initialize the graphics mode using initgraph()

4. Read the line end points (x1,y1) and (x2,y2) such that they are not equal.

5. Calculate the difference between two end points.

$$dx = x1 - x2$$

$$dy = y1 - y2$$

6. If $dx \geq dy$ and $x1 \leq x2$ then //gentle slope lines

Increment x by 1

Increment y by m

Else if $dx \geq dy$ and $x1 > x2$ then //gentle slope lines

Increment x by -1

Increment y by -m

Else if $dx < dy$ and $y1 \leq y2$ then //steep/sharp slope lines

Increment x by $1/m$

Increment y by 1

Else if $dx < dy$ and $y1 > y2$ then //steep/sharp slope lines

Increment x by $-1/m$

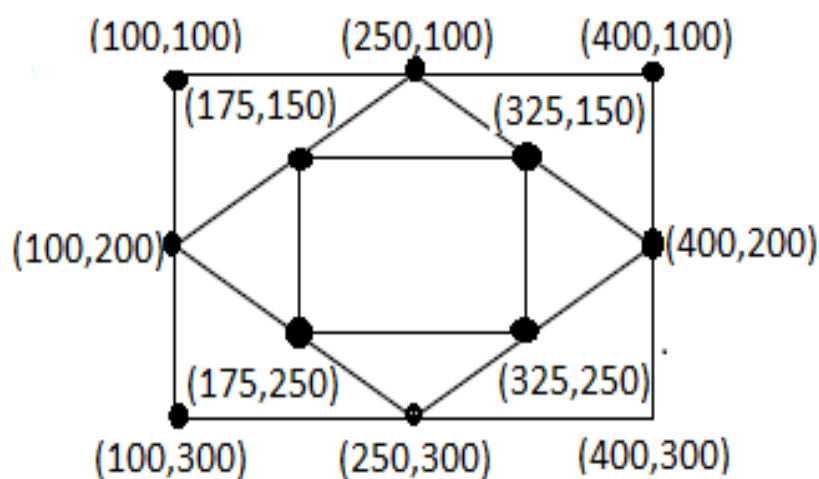
Increment y by -1

7. plot xincrement and yincrement

8. repeat step 6 until other end point is reach

9. Closegraph

10. Stop.



5.4 Questions:-

- Q 1: Which algorithm is faster in DDA and Bresenham's?
- Q 2: State advantages of Bresenham's Line drawing algorithm?
- Q 3: Difference between DDA Line and Bresenham's line algorithm?
- Q 4: Which Algorithm is efficient for line drawing?
- Q 5: What is long form of DDA?
- Q 6: Which line is clearer DDA or Line generated by Equation?

5.5 Conclusion:-

In This way we have studied that how to draw a given pattern of line using DDA Algorithm.

Assignment - 6

Att	Perm	Oral	Total	Sign
(2)	(5)	(3)	(10)	

Title of Assignment: Draw a line with styles.

Problem Definition: Write C++ program for line drawing using DDA or Bresenham's algorithm with patterns such as solid, dotted, dashed, dash dot and thick.

6.1 Prerequisite: Basic primitives of computer graphics and concepts of OOP.

6.2 Learning Objective: Understand the functions available in Graphics library.

6.3 Relevant Theory / Literature Survey:

LINE: It is the path between two end points.

We can draw line by two methods

1] Using inbuilt function

2] Using pixel filling (by logic)

1] Using inbuilt function

Purpose: - draws a line between two specified points

Syntax: -void far line(int x1, int y1, int x2, int y2);

Comments: -Line() draws a line in the current color using current line style and thickness.

Linestyle : This is a function used to sets the linestyle and width or pattern . Sets the style for all lines drawn by line, lineto, rectangle,drawpoly, etc.

Syntax : void far setlinestyle(int style, unsigned pattern, int thickness)

Total 5 styles are available in C Compiler

Name	Value
SOLID_LINE	0
CENTER_LINE	1
DASHED_LINE	2
DOTTED_LINE	3
USERBIT_LINE	4

upattern is a 16-bit pattern that applies only if linestyle is USERBIT_LINE (4). In that case, whenever a bit in the pattern word is 1, the corresponding pixel in the line is drawn in the current drawing color.

For example, a solid line corresponds to a upattern of 0xFFFF (all pixels drawn), while a dashed line can correspond to a upattern of 0x3333(short dashes) or 0x0F0F(long dashes) or 0x3F3F (longer dashes).

Thickness: Sets the width of the line. Two available widths are given in the table.

Name	Value	Meaning
NORM_WIDTH	1	1 pixel
THICK_WIDTH	3	3 pixel

2] Using pixel filling (by logic)

- **Solid Line**

We can use general line drawing algorithm to display solid line. Such as DDA, Bresenham's etc.

- **Dotted Line**

We can easily modify the general line drawing algorithm to display dotted line.

By plotting the alternate pixels in line we can display the dotted line

- **Dashed Line**

We can easily modify the general line drawing algorithm to display dashed line. We have to plot alternate group of pixels along the line to get dashed line.

- **Thick Line**

To produce a thick line we have to run two line drawing algorithms in parallel to find the pixel along the line edges.

Properties of good line drawing algorithm

1. Straight lines should appear as straight lines.
2. They should start and end accurately.
3. Displayed lines should have constant brightness along their length, independent of the line length and orientation
4. Line should be drawn rapidly

Setting font/text Style and Size

There are various functions in graphics library for text operations.

settextstyle() : Sets the current text characteristics i:e text font style, the text direction and its font size

void far settextstyle(int Style,int Horizontal/Vertical,int Size);

Available font styles are :

Font Style	Value	Meaning
DEFAULT_FONT	0	8x8 bit-mapped font
TRIPLEX_FONT	1	Stroked triplex font
SMALL_FONT	2	Stroked small font
SANS_SERIF_FONT	3	Stroked sans-serif font
GOTHIC_FONT	4	Stroked gothic font

Two available directions are:

Name	Value	Direction
HORIZ_DIR	0	Left to Right
VERT_DIR	1	Bottom to Top

settextjustify() : It is used for text alignment i:e left, right and center

void far settextjustify(int hor, int vert)

Three available alignments are

Name	Value	Direction
LEFT_TEXT	0	Left
CENTER_TEXT	1	Center
RIGHT_TEXT	2	Right

6.4 Questions:-

Q 1: How to draw line with different styles without using inbuilt function?

Q 2: What are the properties of line drawing algorithm?

Q 3: Which arguments should pass for line style?

Q 4: how to draw thick and thin line?

6.5 Conclusion:-

In This way we have studied that how to draw a line with line style.

Assignment - 7

Att	Perm	Oral	Total	Sign
(2)	(5)	(3)	(10)	

Title of Assignment: Polygon filling using seed fill algorithm.

Problem Definition: Write C++ program to draw a convex polygon and fill it with desired color using Seed fill algorithm.

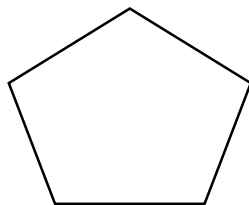
7.1 Prerequisite: Basic of polygon types and polygon filling algorithms.

7.2 Learning Objective: Use inbuilt function of graphics to draw convex polygon (Square, Rectangle, and Triangle) and seed fill algorithm.

7.3 Relevant Theory / Literature Survey:

Polygon

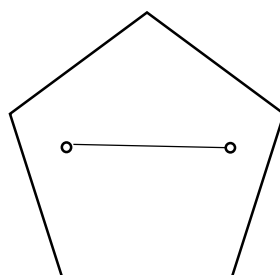
When starting and ending point of any polyline is same i.e. when polyline is closed then it is called polygon

**There are two types of polygon**

The classification of polygons is based on where the line segment joining any two points within the polygon is going to lie.

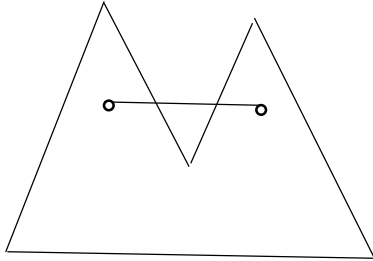
1) Convex polygon

It is a polygon in which any two points taken are surely inside the polygon lies complete inside the polygon, then that polygon is called as convex polygon.



2) Concave polygon

It is a polygon in which the line segment joining any two points within the polygon may not lie completely inside the polygon.



Representation of Polygon

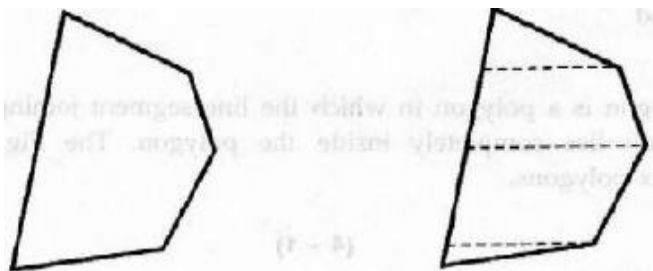
To display polygon on graphic system, it must first be decided, how to represent it. There are 3 approaches to represent polygons accordingly.

1) Polygon drawing primitive approach

Some graphics device supports polygon drawing primitive approach. Directly drawn polygon in graphics system is stored & that polygon acts as a whole unit.

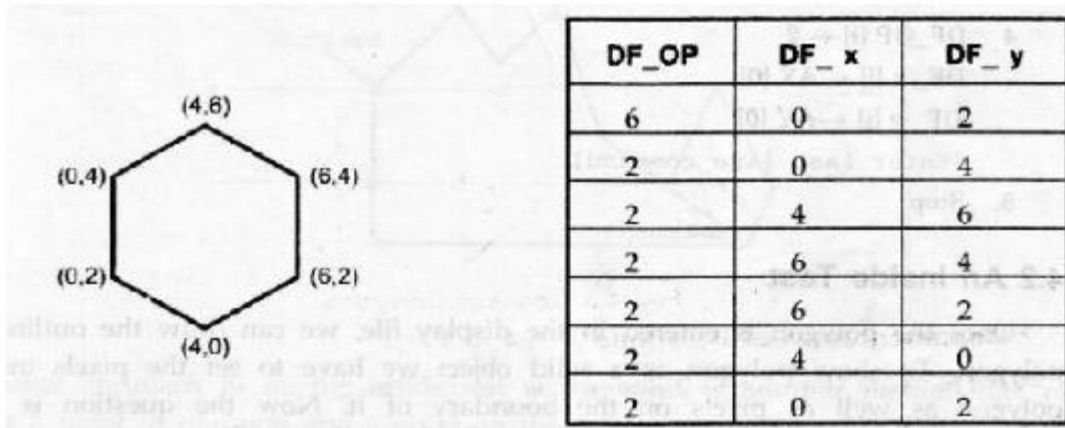
2) Trapezoid primitive Approach

Some graphics devices support trapezoid primitive. Trapezoids are formed from two scan line and two line segments.



3) Line & Point Approach

Most of the graphics devices do not provide any polygon support at all. In such cases, Polygon can be broken into lines & points and stored the full polygon in the display file. The display polygon is then done using line commands.



Convex polygon

A convex polygon is defined as a polygon with all its interior angles less than 180° . This means that all the vertices of the polygon will point outwards, away from the interior of the shape. Think of it as a 'bulging' polygon. Note that a triangle (3-gon) is always convex.

Polygon filling

It is the process of coloring in a fixed area or region. It also means highlighting all the pixels which lie inside the polygon with any color then other background color. There are two basic approaches used to fill the polygon. One way to fill polygon is to start from a given "seed" "point known to be inside the polygon & highlight outward from the point in neighboring pixels. Until we encounter the boundary pixel. The approach is called "seed fill." Another approach to fill polygon is "scan line" in which it is checked whether pixel is inside the polygon or outside the polygon.

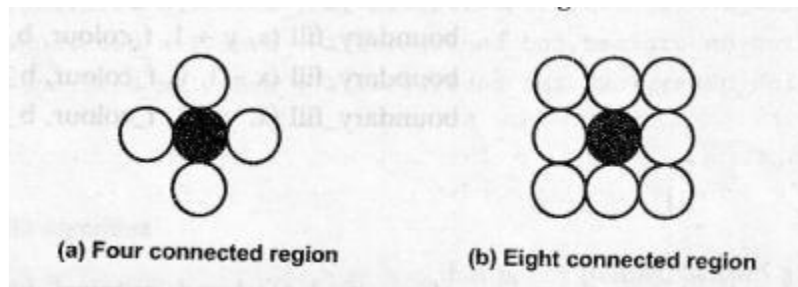
Seed Fill

The seed fill algorithm is further classified as flood algorithm and boundary fill algorithm. Algorithms that fill interior defined regions are called flood-fill algorithms; those that fill boundary defined regions are called boundary fill algorithms or edge fill algorithms.

Boundary fill algorithm

In this algorithm, one point which is surely inside the polygon is taken. This point is called seed point which is nothing but a point from which filling process starts. Boundary defines region may be either 4-connected or 8-connected.

Check the color of pixel If boundary color i.e pixel are not reached, pixels are highlighted (by fill colored) & the process is continued until boundary pixels are reached.



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,+1f_color,b_color)
Boundary_fill(x-1,y,f_color,b_color)
Boundary_fill(x,y-1,f_color,b_color)
}
}
```

Flood Fill Algorithm

In this algorithm, areas are filled by replacing a specified interior color instead of searching a specified interior boundary color. Here, instead of checking boundary color, whether pixels are having polygons original color is checked. Using either 4-connected or 8-connected approach step through pixel position until all interior point has been filled.

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)
}
}
```

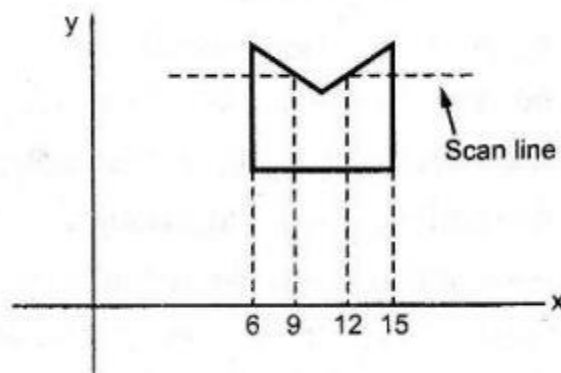
Hence, `getpixel` function gives color of specified pixel & `putpixel` function draws pixel specified color.

Scan-line algorithm

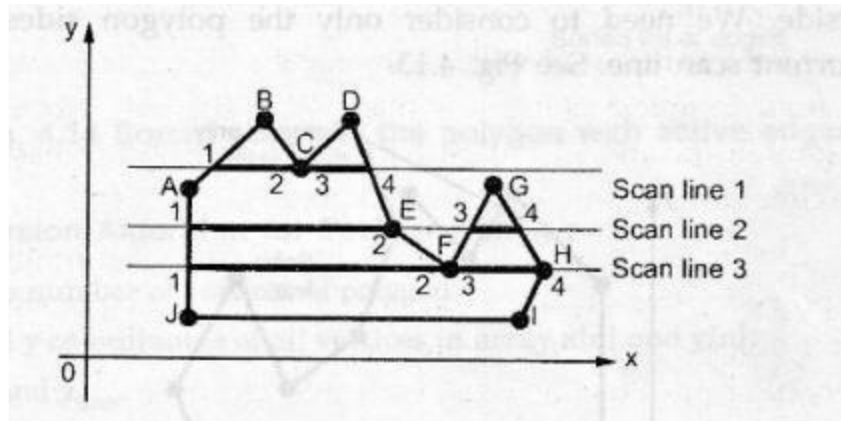
The following illustrates the scan line algorithm for filling of polygon. Identify rightmost and leftmost pixels of the seed pixel and then drawing a horizontal line between these boundary pixels. This procedure is repeated until complete polygon is filled. We have to stack only beginning position of each horizontal line.

For each scan line crossing a polygon, this algorithm locates the intersection points of scan line with polygon edges. These intersection points are then sorted from left to right & the corresponding positions between each intersection pair are set to the specified fill color. The scan line algorithm first finds the largest & smallest y values of the polygon.

Then it starts with the largest y value & works it may down. Scanning is done from left to right in the manner of a raster display.



In some cases intersection points is a vertex so special handling is required to find the exact intersection point. So we must look at the other end points of the two line segments of the polygon which meet at this vertex. If these points lie on the same side of the scan line then the point in question count as a two intersection /an even number of intersection. If these points lie on the opposite side of the scan line then the point in question count as a single intersection.



Features of scan line algorithm

1. Scan line algorithm is used for filling of polygons.
2. This algorithm solves the problem of hidden surfaces while generating display scan line.
3. It is used in orthogonal projection.
4. It is non recursive algorithm.
5. In scan line algorithm we have to stack only a beginning position for horizontal pixel scan, instead of stacking all unprocessed neighboring positions around the current position. Therefore it is sufficient algorithm.

7.4 Questions:-

- Q 1: Define polygon?
- Q 2: What are the types of polygon?
- Q 3: Explain convex polygon?
- Q 4: Define Concave polygon.
- Q 5: How many ways to represent polygon?
- Q 6: How many methods of polygon filing?
- Q 7: Enlist any three polygon filling algorithm.
- Q 8: Explain with example and compare seed fill and edge fill algorithm.
- Q 9: What is scan line polygon filling algorithm? Explain with an example.

7.5 Conclusion:-

In This way we have studied that how to draw a convex polygon and how to fill a polygon with seed fill algorithm.

Assignment - 8

Att	Perm	Oral	Total	Sign
(2)	(5)	(3)	(10)	

Title of Assignment: 2D Reflection.

Problem Definition: Write C++ program to implement reflection of 2-D object about X axis, Y axis and about X=Y axis. Also rotate object about arbitrary point given by user.

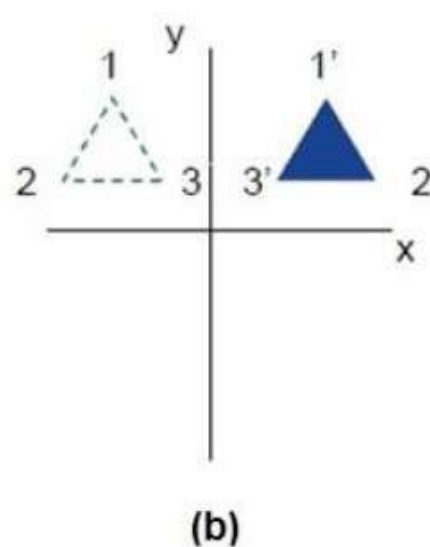
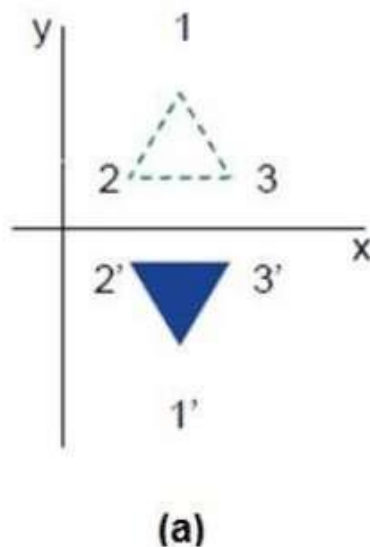
8.1 Prerequisite: Basic of 2D transformations.

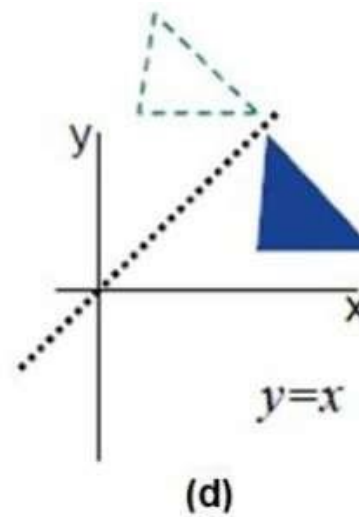
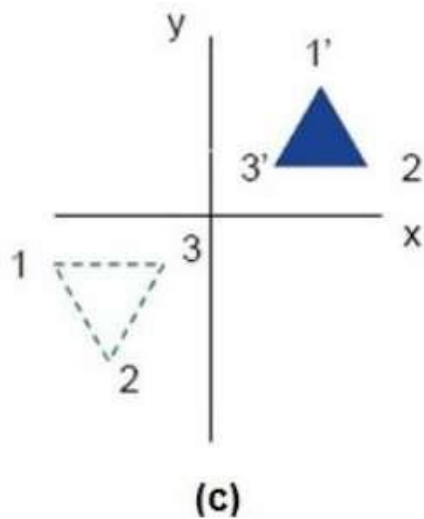
8.2 Learning Objective: 2D reflections about different axis and rotation with an arbitrary point.

8.3 Relevant Theory / Literature Survey:

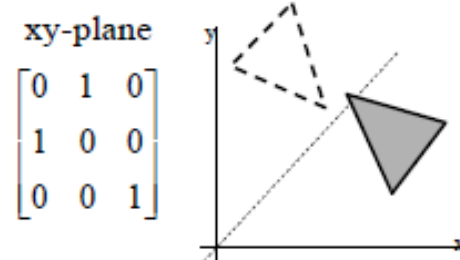
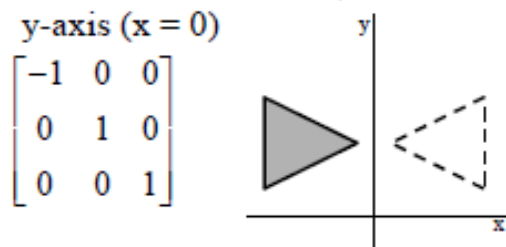
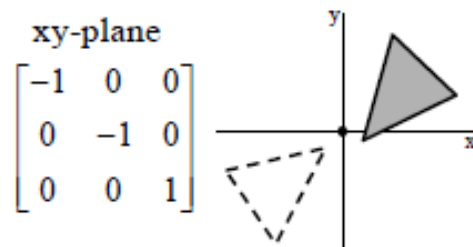
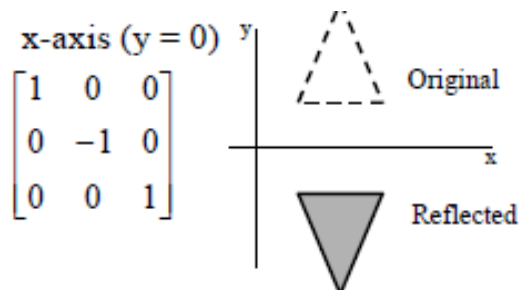
Reflection is the mirror image of original object. In other words, we can say that it is a rotation operation with 180° . In reflection transformation, the size of the object does not change.

The following figures show reflections with respect to X and Y axes, and about the origin respectively.





Reflection matrices:



8.4 Questions:-

Q 1: What is 2D Reflection, explain in brief?

Q 2: Explain 2D reflections with X, Y & X=Y axis with reflection matrices.

Q 3: Give practical scenario where you need reflections.

8.5 Conclusion:-

In This way we have studied different 2D reflections.

Assignment - 9

Att	Perm	Oral	Total	Sign
(2)	(5)	(3)	(10)	

Title of Assignment: Hilbert curve.

Problem Definition: Write C++ program to generate Hilbert curve using concept of fractals.

9.1 Prerequisite: Circle generation.

9.2 Learning Objective: Generating curve.

9.3 Relevant Theory / Literature Survey:

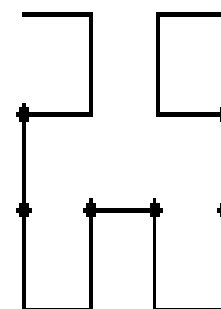
The Hilbert curve is a space filling curve that visits every point in a square grid with a size of 2×2 , 4×4 , 8×8 , 16×16 , or any other power of 2. Applications of the Hilbert curve are in image processing: especially image compression and dithering. It has advantages in those operations where the coherence between neighbouring pixels is important.

Cups and joins

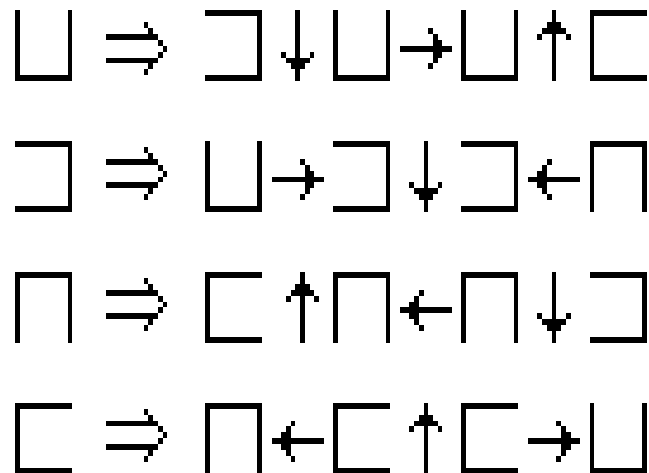
The basic elements of the Hilbert curves are what I call "cups" (a square with one open side) and "joins" (a vector that joins two cups). The "open" side of a cup can be top, bottom, left or right. In addition, every cup has two end-points, and each of these can be the "entry" point or the "exit" point. So, there are eight possible varieties of cups. In practice, a Hilbert curve uses only four types of cups. In a similar vein, a join has a direction: up, down, left or right.



A first order Hilbert curve is just a single cup (see the figure on the left). It fills a 2×2 space. The second order Hilbert curve replaces that cup by four (smaller) cups, which are linked together by three joins (see the figure on the right; the link between a cup and a join has been marked with a fat dot in the figure). Every next order repeats the process or replacing each cup by four smaller cups and three joins.



Cup subdivision rules



9.4 Questions:-

Q 1: What is Fractals?

Q 2: What is Benzier curve?

Q 3: What is B-spline curve?

Q 3: What is Hibert curve?

Q 3: Explain is types of curve?

9.5 Conclusion:-

In This way we have studied generating Hibert curve.

Assignment - 10

Att	Perm	Oral	Total	Sign
(2)	(5)	(3)	(10)	

Title of Assignment: Bouncing ball animation in Blender.

Problem Definition: Write a program using Blender to draw a Bouncing ball animation.

10.1 Prerequisite: Blender animation tool, basic concept of animation.

10.2 Learning Objective: To study about authoring tool like Blender.

10.3 Relevant Theory / Literature Survey:

Critical to learning any software application is some initial understanding of the basic concepts: how that software's world works and the fundamental skills you need to work in that world. If you have never used a three dimensional (3D) software application before, you may initially find Maya different compared to 2D applications. If you are wondering "where do I begin?", this chapter is the best place to start. We recommend that you complete the lessons in this chapter so the essential concepts and skills presented become familiar to you.

Blender is a professional free and open-source 3D computer graphics software product used for creating animated films, visual effects, art, 3D printed models, interactive 3D applications and video games. Blender's features include 3D modeling, UV unwrapping, texturing, raster graphics editing, rigging and skinning, fluid and smoke simulation, particle simulation, soft body simulation, sculpting, animating, match moving, camera tracking, rendering, video editing and compositing. It further features an integrated game engine.

Official releases of Blender for Microsoft Windows, MacOS and Linux, as well as a port for FreeBSD, are available in both 32-bit and 64-bit versions. Though it is often distributed without extensive example scenes found in some other programs, the software contains features that are characteristic of high-end 3D software. Among its capabilities are:

- Support for a variety of geometric primitives, including polygon meshes, fast subdivision surface modeling, Bezier curves, NURBS surfaces, metaballs, icospheres, multi-res digital sculpting (including dynamic topology, maps baking, remeshing,

resymetrize, decimation), outline font, and a new n-gon modeling system called B-mesh.

- Internal render engine with scanline rendering, indirect lighting, and ambient occlusion that can export in a wide variety of formats.
- A pathtracer render engine called Cycles, which can take advantage of the GPU for rendering. Cycles supports the Open Shading Language since Blender 2.65.
- Integration with a number of external render engines through plugins.
- Keyframed animation tools including inverse kinematics, armature (skeletal), hook, curve and lattice-based deformations, shape animations, non-linear animation, constraints, and vertex weighting.
- Simulation tools for soft body dynamics including mesh collision detection, LBM fluid dynamics, smoke simulation, Bullet rigid body dynamics, ocean generator with waves.
- A particle system that includes support for particle-based hair.
- Modifiers to apply non-destructive effects.
- Python scripting for tool creation and prototyping, game logic, importing/exporting from other formats, task automation and custom tools.
- Basic non-linear video/audio editing.
- The Blender Game Engine, a sub-project, offers interactivity features such as collision detection, dynamics engine, and programmable logic. It also allows the creation of stand-alone, real-time applications ranging from architectural visualization to video games.
- A fully integrated node-based compositor within the rendering pipeline accelerated with OpenCL.
- Procedural and node-based textures, as well as texture painting, projective painting, vertex painting, weight painting and dynamic painting.
- Real-time control during physics simulation and rendering.
- Camera and object tracking.

User interface



Blender's user interface underwent a significant update during the 2.5x series

Blender's user interface incorporates the following concepts:

Editing modes

The two primary modes of work are *Object Mode* and *Edit Mode*, which are toggled with the Tab key. Object mode is used to manipulate individual objects as a unit, while Edit mode is used to manipulate the actual object data. For example, Object Mode can be used to move, scale, and rotate entire polygon meshes, and Edit Mode can be used to manipulate the individual vertices of a single mesh. There are also several other modes, such as Vertex Paint, Weight Paint, and Sculpt Mode.

Hotkey usage

Most of the commands are accessible via hotkeys. There are also comprehensive GUI menus.

Numeric input

Numeric buttons can be "dragged" to change their value directly without the need to aim at a particular widget, as well as being set using the keyboard. Both sliders and number buttons can be constrained to various step sizes with modifiers like the Ctrl and Shift keys. Python expressions can also be typed directly into number entry fields, allowing mathematical expressions to specify values.

Workspace management

The Blender GUI builds its own tiled windowing system on top of one or multiple windows provided by the underlying platform. One platform window (often sized to fill the screen) is divided into sections and subsections that can be of any type of Blender's views or window-types. The user can define multiple layouts of such Blender windows, called *screens*, and switch quickly between them by selecting from a menu or with keyboard shortcuts. Each window-type's own GUI elements can be controlled

with the same tools that manipulate 3D view. For example, one can zoom in and out of GUI-buttons using similar controls one zooms in and out in the 3D viewport. The GUI viewport and screen layout is fully user-customizable. It is possible to set up the interface for specific tasks such as video editing or UV mapping or texturing by hiding features not used for the task.

Hardware requirements

Blender hardware requirements

Hardware	Minimum	Recommended	Production-standard
Processor	32-bit dual core 2 GHz CPU with SSE2 support	64-bit quad core CPU	64-bit eight core CPU
Memory	2 GB RAM	8 GB RAM	16 GB RAM
Graphics card	OpenGL compatible card with 2 GB video RAM	2.1 OpenGL 3.2 compatible card with 2 GB video RAM (CUDA or OpenCL for GPU rendering)	Dual OpenGL 3.2 compatible cards with 4 GB video RAM
Display	1280×768 pixels, 24-bit color	1920×1080 pixels, 24-bit color	Dual 1920×1080 pixels, 24-bit color
Input	Mouse or trackpad	Three-button mouse	Three-button mouse and graphics tablet

Supported platforms

Blender is available for Windows Vista and above, Mac OSX 10.6 and above, and Linux.

Blender 2.76b is the last supported release for Windows XP

10.4 Questions:-

Q 1: What is Blender?

Q 2: How many tools are there for creating animation?

Q 3: What is animation?

Q 4: With the help of Blender how we will create animation?

Q 5: What do you keyframe?

10.5 Conclusion:-

Blender complete software is a comprehensive 3D solution for producing professional quality graphics animation projects. It integrates all of the best tools for 3D modeling, animation, and rendering.

Assignment - 11

Att	Perm	Oral	Total	Sign
(2)	(5)	(3)	(10)	

Title of Assignment: Clock with pendulum.

Problem Definition: Write C++ program to simulate following scene-

Clock with pendulum

11.1 Prerequisite: Line & Circle Generation algorithms.

11.2 Learning Objective: To study about different animation creation using Line & Circle Generation algorithms with moving the line in non-linear path.

11.3 Relevant Theory / Literature Survey:

A **pendulum** is a weight suspended from a pivot so that it can swing freely. When a pendulum is displaced sideways from its resting, equilibrium position, it is subject to a restoring force due to gravity that will accelerate it back toward the equilibrium position. When released, the restoring force combined with the pendulum's mass causes it to oscillate about the equilibrium position, swinging back and forth. The time for one complete cycle, a left swing and a right swing, is called the period. The period depends on the length of the pendulum and also to a slight degree on the amplitude, the width of the pendulum's swing.

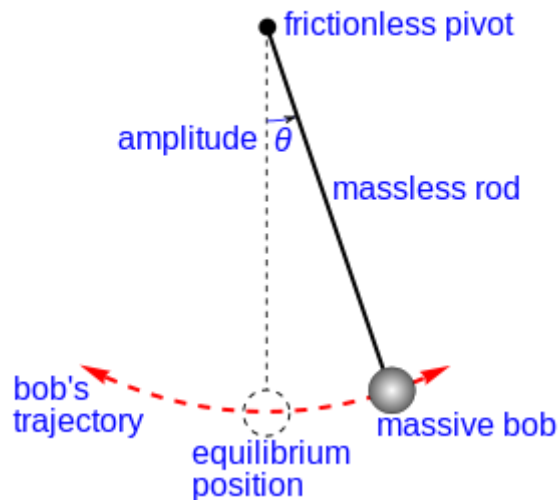


Figure: Simple gravity pendulum" model assumes no friction or air resistance.

11.4 Questions:-

Q 1: Explain different Line Generation algorithms?

Q 2: Explain different Circle Generation algorithms?

Q 3: Discuss the advantages and disadvantages of different Line Generation algorithms?

Q 4: Discuss the advantages and disadvantages of different Circle Generation algorithms?

Q.5: Explain the technique used for moving the pendulum?

11.5 Conclusion:-

Thus we had created a simple clock with moving pendulum using C++.