



SECOND YEAR
Information Technology (2015
Course)

LABORATORY MANUAL
For

COMPUTER GRAPHICS LABORATORY

SEMESTER - II

[Subject code: 214456]

[Prepared By]

Ms. Kavita A. Sultanpure
Dr. Shweta C. Dharmadhikari
Mr. Abhinay G. Dhamankar

**PUNE INSTITUTE OF COMPUTER
TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY**

LABORATORY MANUAL

AY 2018-19

**214456: COMPUTER GRAPHICS LABORATORY
SECOND YEAR - INFORMATION TECHNOLOGY
SEMESTER - II**

<u>TEACHING SCHEME</u> <u>SCHEME</u>	<u>EXAMINATION</u>
Lectures: 3 Hrs/Week 50 Marks	Theory:
Practical: 4Hrs/Week 50 Marks	On-Line:

:::|| Prepared By ||:::

Ms. Kavita A. Sultanpure

Dr. S. C.

Dharmadhikari

Mr. Abhinay g. dhamankar

214456: COMPUTER GRAPHICS LABORATORY**Prerequisites:**

1. Basic Geometry, Trigonometry, Vectors and Matrices
2. Basics of Data Structures and Algorithms

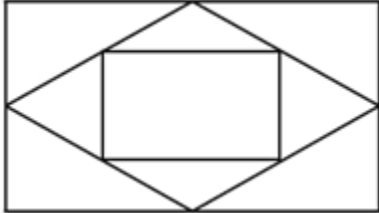

Course Objectives:

1. To acquaint the learners with the basic concepts of Computer Graphics
2. To learn the various algorithms for generating and rendering graphical figures
3. To get familiar with mathematics behind the graphical transformations
4. To understand and apply various methods and techniques regarding projections, animation, shading, illumination and lighting

Course Outcomes: On completion of the course, learner will be able to –

1. Apply mathematics and logic to develop Computer programs for elementary graphic
2. operations
3. Develop scientific and strategic approach to solve complex problems in the domain of
4. Computer Graphics
5. Develop the competency to understand the concepts related to Computer Vision and Virtual reality
6. Apply the logic to develop animation and gaming programs

LIST OF ASSIGNMNETS

LAB EXPT.N O	PROBLEM STATEMENT
PART - A	
1.	<p>Draw the following pattern using any Line drawing algorithms.</p> 
2.	<p>Draw inscribed and Circumscribed circles in the triangle as shown as an example below (Use any Circle drawing and Line drawing algorithms)</p> 
3.	<p>Draw the polygons by using the mouse. Choose colors by clicking on the designed color pane. Use window port to draw. (Use DDA algorithm for line drawing)</p>
4.	<p>Draw a 4X4 chessboard rotated 45° with the horizontal axis. Use Bresenham algorithm to draw all the lines. Use seed fill algorithm to fill black squares of the rotated chessboard</p>
PART - B	
1.	<p>Implement Cohen Sutherland algorithm to clip any given Line/polygon. Provide the vertices of the Line/polygon to be clipped and pattern of clipping interactively.</p>
2.	<p>Implement translation, sheer, rotation and scaling transformations on equilateral triangle and rhombus.</p>
3.	<p>Implement Cube rotation about vertical axis passing through its centroid.</p>
4.	<p>Generate fractal patterns by using Koch curves.</p>
5.	<p>Animation : Implement any one of the following animation assignments, i) Clock with pendulum</p>

	<ul style="list-style-type: none">ii) National Flag hoistingiii) Vehicle/boat locomotioniv) Falling Water drop into the water and generated waves after impactv) Kaleidoscope views generation (at least 3 colorful patterns)
--	--



Subject Coordinator
Department (I.T)
Dr. S. C. Dharmadhikari

Head of

Dr. B. A. Sonkamble

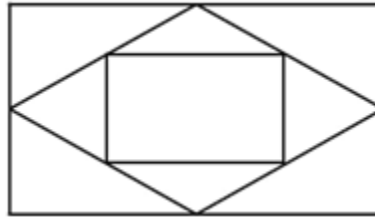
Assignment No 1

AIM:

To implement line drawing algorithms.

PROBLEM STATEMENT:

Draw the following pattern using any Line drawing algorithms.



THEORY:

Line Drawing:

The Cartesian slope intercept equation for a straight line is:

$$y = mx + b \quad \text{.....1}$$

Where m is the slope of a line and b is the y intercept of the line. If two endpoints of a line segment are specified at positions (x_1, y_1) , & (x_2, y_2)

We can determine the values of m & b as:

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x} \quad \text{.....2}$$

$$b = y_1 - mx_1 \quad \text{.....3}$$

All line generation algorithms are based on equation 1 and calculations given in equations 2 and 3. For any given x interval Δx along a line we can calculate y interval Δy from equation 2 as:

$$\Delta y = m \Delta x \quad \text{.....4}$$

Similarly we can obtain the x interval Δx to a specified Δy as:

$$\Delta x = \frac{\Delta y}{m} \quad \text{.....5}$$

For drawing a line we need to turn ON the pixels which are on the line segment.

DDA's line generation algorithm:

It is Digital Differential Analyzer (DDA). The digital differential analyzer is a scan conversion algorithm base on calculating either Δy or Δx using

equation 4 or equation 5. We sample the line at unit intervals in one coordinate and determine corresponding integer values nearest line path for other coordinate. So initially we are plotting the first endpoint of the line and then depending on the slope we are sampling at unit intervals in either x direction i.e. $\Delta x = 1$ and determine y interval Δy to plot each successive y value or we are sampling at unit intervals in y direction i.e. $\Delta y = 1$ and determine x interval Δx to plot each successive x value.

Consider a line with positive slope. If the slope is less than or equal to 1, we sample it at unit x intervals ($\Delta x = 1$) and compute each successive y value as

$$y_{k+1} = y_k + m \quad \text{.....6}$$

Here k is the Subscript which takes integer values starting from 1 for the first point and increases by 1 until the final end point is reached. The value of m can be any real number between 0 and 1. The calculated y value must be rounded to the nearest integer. For lines with positive slope greater than 1, we reverse the roles of x and y. i. e. we sample at unit y Intervals ($\Delta y = 1$) and calculate each succeeding x value as

$$x_{k+1} = x_k + 1/m \quad \text{.....7}$$

Equations 6 & 7 are based on the assumption that the lines are to be processed from the left end point to right end point. If this processing is reversed, so the starting end point is at right then either we will have $\Delta x = -1$ and

$$y_{k+1} = y_k - m \quad \text{.....8}$$

Or

$\Delta y = -1$ and

$$x_{k+1} = x_k - 1/m \quad \text{.....9}$$

Equations 6 through 9 can also be used to calculate pixel positions for lines having negative slope. If the absolute value of the slope is less than 1 and the start endpoint is at the left, we start at $\Delta x = 1$ and calculate y values with Eq. 6. When the start endpoint is at the right (for the same slope), we set $\Delta x = -1$ and obtain y positions from Eq. 8. Similarly, when the absolute value of a negative slope is greater than 1, we use $\Delta y = -1$ and Equation 9 or we use $\Delta y = 1$ and equation 7.

Midpoint of Line Segment

The point halfway between the endpoints of a line segment is called the midpoint. A midpoint divides a line segment into two equal segments. If the line segments are vertical or horizontal, you may find the midpoint by

simply dividing the length of the segment by 2. If (x_1, y_1) and (x_2, y_2) are two end points of line segment, then midpoint (x_m, y_m) is calculated by

$$(x_m, y_m) = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} \right)$$

Length of line segment:

When we need to find the length (distance) of a segment such as AB we simply count the distance from point A to point B. Let (x_1, y_1) and (x_2, y_2) are two end points of line segment, construct a right triangle as shown in fig, by Pythagoras theorem we can calculate the length of AB. Length of AE is calculated by $(x_2 - x_1)$, similarly the length of EB is $(y_2 - y_1)$.

$$\text{Length of AB} = [(x_2 - x_1)^2 + (y_2 - y_1)^2]^{1/2}$$

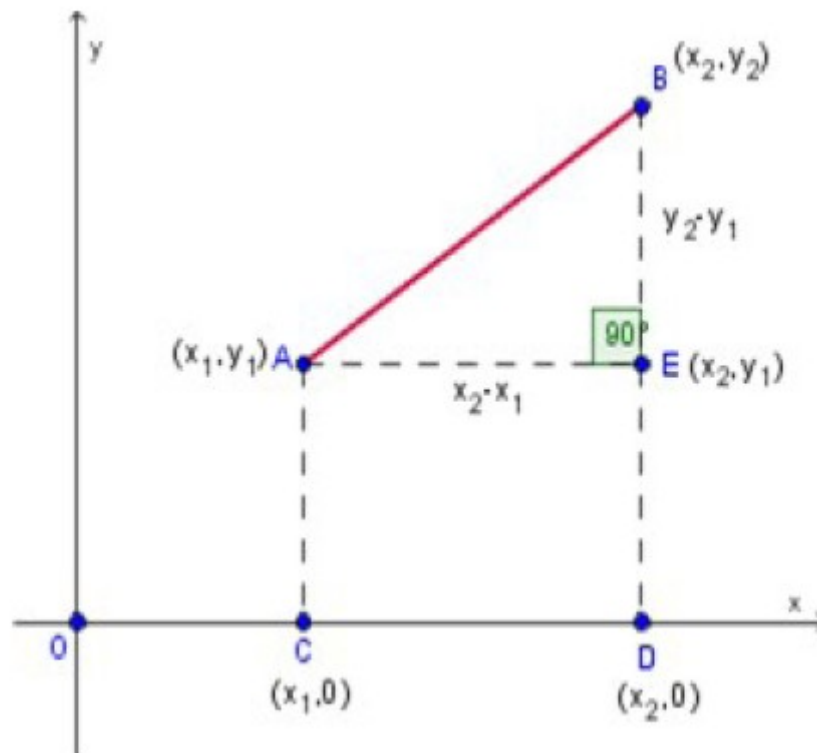


Fig: length of line segment

ALGORITHM:

Step 1: Accept the bottom left vertex coordinates of the pattern to be drawn and the length and the breadth of the outer rectangle from user.

Step 2: Find out the vertices of outer rectangle by using length & breadth. Draw the lines between vertices of outer rectangle by using following steps of DDA algorithm:

Step 2.1: Accept line endpoints (x1, y1) & (x2, y2) from user.

Step 2.2: initialize the variables dx and dy as:

dx=abs(x2-x1)

dy=abs(y2-y1)

Step 2.3: Calculate steps i.e. the total number of successive pixels to be plotted as:

if(dx>=dy)then

steps=dx

else

steps=dy

step 2.4: calculate x interval Δx and y interval Δy to be applied at each sampling step as:

$\Delta x = (x2 - x1) / \text{steps}$

$\Delta y = (y2 - y1) / \text{steps}$

step 2.5: Plot the initial pixel

x=x1

y=y1

glBegin(GL_LINES);

glVertex2d(x, y);

glEnd();

Step 2.6: calculate the remaining pixels and plot them.

Main loop

for i=1 to steps-1

x=x+ Δx

y=y+ Δy

glBegin(GL_LINES);

glVertex2d(roundoff(x),roundoff(y));

glEnd();

/* roundoff is a function which takes floating value and returns the nearest integer of input value */

next i

end

/*Round off function definition*/

int roundoff(float p)

{

int q;

q=p;

```
        if((p-q)<0.5)
            return q;
        else
            return q++;
    }
```

Step 2.7: Stop.

Step 3: Find out the midpoints of the edges of the outer rectangle by using above midpoint formula.

Step 4: Draw the inner rhombus by drawing lines using above DDA algorithm between calculated midpoints of outer rectangle.

Step 5: Find out the midpoints of the edges of the rhombus by using above midpoint formula.

Step 6: Draw the outer rectangle by drawing lines using above DDA algorithm between calculated midpoints of rhombus.

Step 7: Stop.

INPUT:

The bottom left vertex coordinates of the pattern to be drawn and the length and the breadth of the outer rectangle.

OUTPUT:

The above Pattern

Assignment No 2

AIM:

To implement Line and Circle drawing algorithms.

PROBLEM STATEMENT:

Draw inscribed and Circumscribed circles in the triangle as shown as an example below (Use any Circle drawing and Line drawing algorithms)

**THEORY:****Bresenham's algorithm for circle drawing:**

In addition to drawing a straight line the Bresenham's incremental approach can be applied to a circle. To begin with circle drawing, note that only one octant of the circle need to be generated. The other parts can be obtained by using circle's symmetry property. The circle is symmetric in quadrants as well as octants. So if we are able to determine circle boundary in one octant we can map it to remaining octant (See figure 1).

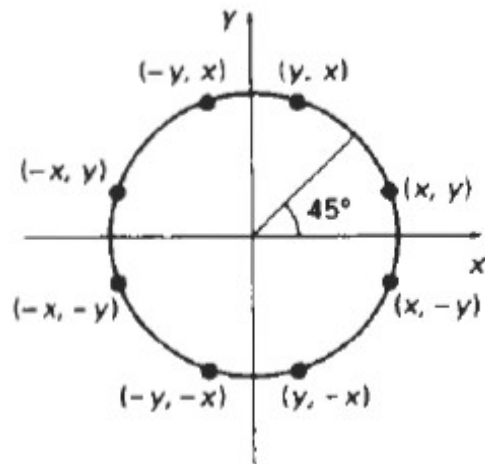


Figure 1: Symmetry of circle in octants.

To derive Bresenham's circle generation algorithm, consider the first octant of the circle centered at origin. Notice that, if the algorithm begins at $x=0$ and $y=R$ (R is the radius), then for clockwise generation of the circle the circle's arc is towards down (see figure 2) and in first quadrant the absolute value of slope of the arc is less than one. So we sample at unit x direction and calculate the y values. Note that calculated y values will be monotonically decreasing for successive x values.

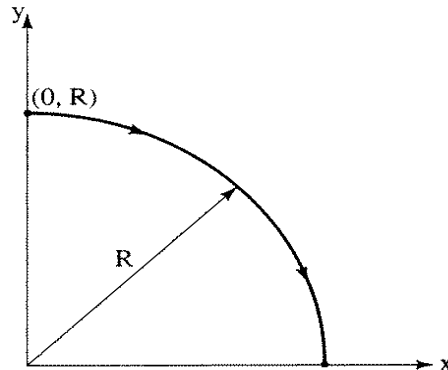


Figure 2: First quadrant of the circle

For clockwise generation of the circle assuming pixel at (x_k, y_k) is already plotted we next need to determine next pixel on circle boundary. There are only three possible selections for the next pixel which best represent the circle:

- 1-Horizontal to the right
- 2-Diagonally downward to the right
- 3-Vertically downward

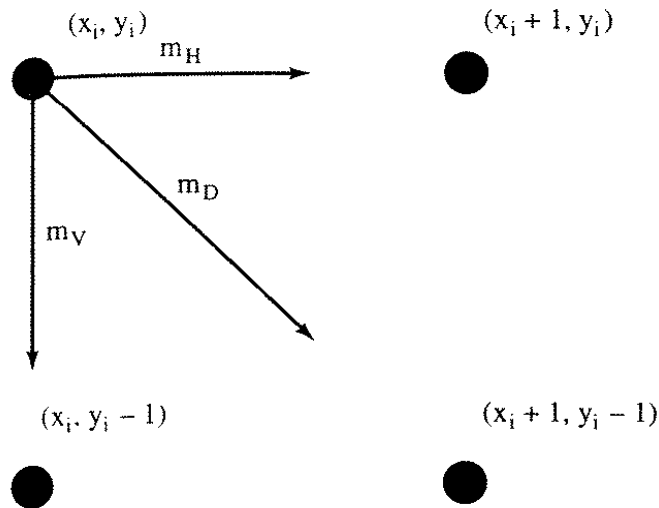


Figure 3: First octant pixel selections

The distance between true circle and these pixels are labeled MH, MD, and MV in figure 3.

The algorithm chooses the pixel (the movement) which minimize the square of the distance between one of these pixels and the true circle

The distance in the three cases are measured by :

$$MH = | (x_k + 1)^2 + (y_k)^2 - R^2 |$$

$$MD = | (x_k + 1)^2 + (y_k - 1)^2 - R^2 |$$

$$MV = | (x_k)^2 + (y_k - 1)^2 - R^2 |$$

The above calculations are simplified by considering that there will be only five possible intersections of the pixel grid with vicinity of the point (x_k, y_k) . As shown in figure 4.

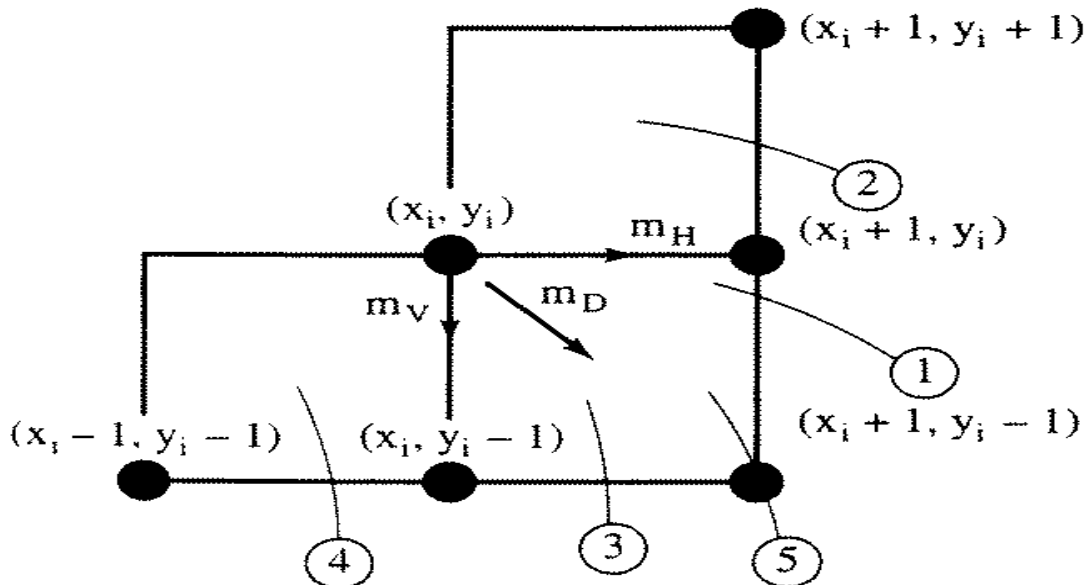


Figure 4: Intersection of a circle and the raster grid

Now to apply Bresenham's method we define function of circle

$$f_{\text{circle}}(x, y) = x^2 + y^2 - R^2 \dots\dots\dots 1$$

Any point (x, y) on the boundary of circle satisfies the function $f_{\text{circle}}(x, y) = 0$. If the point is in the interior of the circle then the circle function is negative. And if the point is outside of circle then the circle function is positive. So the relative position of point (x, y) can be determined by checking the sign of circle function.

$$\begin{aligned} f_{\text{circle}}(x, y) &\leq 0 \text{ if } (x, y) \text{ is outside the circle boundary} \\ &= 0 \text{ if } (x, y) \text{ is on the circle boundary} \\ &> 0 \text{ if } (x, y) \text{ is inside the circle boundary} \end{aligned}$$

.....

....2

Now we will apply circle function at diagonal point $(x_k + 1, y_k - 1)$. And the circle function tests in equation 2 are performed for $(x_k + 1, y_k - 1)$ at each sampling step. So as per the Bresenham's line drawing algorithm it is desirable to use the sign of the decision parameter d_k to choose proper pixel at next position. Thus the circle function at $(x_k + 1, y_k - 1)$ is the decision parameter (error term) for Bresenham's algorithm.

$$f_{\text{circle}}(x_k + 1, y_k - 1) = d_k = (x_k + 1)^2 + (y_k - 1)^2 - R^2 \dots\dots\dots 3$$

If $d_k < 0$ then the diagonal point $(x_k + 1, y_k - 1)$ is inside the actual circle i.e. we use case 1 or case 2 see figure 4. It is clear that either the pixel at $(x_k + 1, y_k) = \text{MH}$ or that the pixel at $(x_k + 1, y_k - 1) = \text{MD}$ must be chosen. So to determine whether to choose MH or MD, consider case 1 by applying the

circle function at $(x_k + 1, y_k)$ and at $(x_k + 1, y_k - 1)$ and checking the difference between the circle functions:

$$\delta = [(x_k + 1)^2 + (y_k)^2 - R^2] - [(x_k + 1)^2 + (y_k - 1)^2 - R^2] \dots\dots\dots 4$$

if $\delta \leq 0$ then the distance from the actual circle to the diagonal pixel (MD) is greater than that to the horizontal pixel (MH). Conversely if $\delta > 0$ then the distance to the horizontal pixel, MH is greater so,

If $\delta \leq 0$ choose MH $(x_k + 1, y_k)$

If $\delta > 0$ choose MD $(x_k + 1, y_k - 1)$

Now in order to minimize the calculations involved in evaluating δ consider case 1.

Now circle function at $(x_k + 1, y_k)$ will be

$$(x_k + 1)^2 + (y_k)^2 - R^2 \geq 0 \dots\dots\dots 5$$

Because horizontal pixel $(x_k + 1, y_k)$ will always be outside the circle boundary and circle function at $(x_k + 1, y_k - 1)$ will be

$$(x_k + 1)^2 + (y_k - 1)^2 - R^2 < 0 \dots\dots\dots 6$$

Because diagonal pixel $(x_k + 1, y_k - 1)$ will always be inside the circle boundary

Now δ can be calculated by putting values from equation 5 and 6 in equation 4 as:

$$\begin{aligned} \delta &= (x_k + 1)^2 + (y_k)^2 - R^2 - (-(x_k + 1)^2 + (y_k - 1)^2 - R^2) \\ &= (x_k + 1)^2 + (y_k)^2 - R^2 + (x_k + 1)^2 + (y_k - 1)^2 - R^2 \\ &= (x_k + 1)^2 + (y_k)^2 - 2(y_k) + 1 - R^2 + (x_k + 1)^2 + (y_k - 1)^2 \\ &\quad + 2(y_k) - 1 - R^2 \\ &= 2[(x_k + 1)^2 + (y_k - 1)^2 - R^2] + 2(y_k) - 1 \dots\dots\dots 7 \end{aligned}$$

Substituting the value of d_k in equation 7 we get

$$\begin{aligned} &= 2(d_k) + 2y_k - 1 \\ \delta &= 2(d_k + y_k) - 1 \dots\dots\dots 8 \end{aligned}$$

If $d_k > 0$ then the diagonal point $(x_k + 1, y_k - 1)$ is outside the actual circle i.e. we use case 3 or case 4 see figure 4. It is clear that either the pixel at $(x_k + 1, y_k - 1) = MD$ or that the pixel at $(x_k, y_k - 1) = MV$ must be chosen. So to determine whether to choose MH or MD, consider case 3 by applying the circle function at $(x_k + 1, y_k - 1)$ and at $(x_k, y_k - 1)$ and checking the difference between the circle functions:

$$\delta' = [(x_k + 1)^2 + (y_k - 1)^2 - R^2] - [(x_k)^2 + (y_k - 1)^2 - R^2] \dots\dots\dots 9$$

if $\delta' \leq 0$ then the distance from the actual circle to the vertical pixel (MV) is greater than that to the diagonal pixel (MD) so diagonal pixel is chosen .

Conversely if $\delta' > 0$ then the distance to the diagonal pixel, MD is greater so vertical pixel is chosen. Thus

If $\delta' \leq 0$ choose MD $(x_k + 1, y_k - 1)$

If $\delta' > 0$ choose MV $(x_k, y_k - 1)$

Again in order to minimize the calculations involved in evaluating δ' consider case 3.

Now circle function at $(x_k + 1, y_k - 1)$ will be

$$(x_k + 1)^2 + (y_k - 1)^2 - R^2 \geq 0 \dots\dots\dots 10$$

Because diagonal pixel $(x_k + 1, y_k - 1)$ will always be outside the circle boundary and

circle function at $(x_k, y_k - 1)$ will be

$$(x_k)^2 + (y_k - 1)^2 - R^2 < 0 \dots\dots\dots 11$$

Because vertical pixel $(x_k, y_k - 1)$ will always be inside the circle boundary

Now δ' can be calculated by putting values from equation 10 and 11 in equation 9 as:

$$\begin{aligned} \delta' &= (x_k + 1)^2 + (y_k - 1)^2 - R^2 - (-(x_k)^2 + (y_k - 1)^2 - R^2) \\ &= (x_k + 1)^2 + (y_k - 1)^2 - R^2 + (x_k)^2 + (y_k - 1)^2 - R^2 \\ &= (x_k + 1)^2 + (x_k)^2 + 2(x_k) + 1 - R^2 + (y_k - 1)^2 + (y_k - 1)^2 - \end{aligned}$$

$$R^2 - 2(x_k) - 1$$

$$= 2[(x_k + 1)^2 + (y_k - 1)^2 - R^2] - 2(x_k) - 1 \dots\dots\dots 12$$

Substituting the value of d_k in equation 12 we get

$$= 2(d_k) - 2x_k - 1$$

$$\delta = 2(d_k - x_k) - 1 \dots\dots\dots 13$$

If $d_k = 0$ then pixel $(x_k + 1, y_k - 1)$ is on the actual circle. We choose case 5 see figure 4 we choose the pixel at $(x_k + 1, y_k - 1)$ i.e. MD

The successive decision parameters can be calculated by using incremental integer calculations. In case of horizontal move

$$x_{k+1} = x_k + 1$$

$$y_{k+1} = y_k$$

So decision parameter at $k + 1$ can be calculated as:

$$\begin{aligned} d_{k+1} &= (x_{k+1} + 1)^2 + (y_{k+1} - 1)^2 - R^2 \\ &= (x_{k+1})^2 + 2(x_{k+1}) + 1 + (y_{k+1} - 1)^2 - R^2 \\ &= (x_k + 1)^2 + 2(x_{k+1}) + 1 + (y_k - 1)^2 - R^2 \\ &= (x_k + 1)^2 + (y_k - 1)^2 - R^2 + 2(x_{k+1}) + 1 \\ d_{k+1} &= d_k + 2(x_{k+1}) + 1 \dots\dots\dots 14 \end{aligned}$$

similarly, in case of vertical move

$$x_{k+1} = x_k$$

$$y_{k+1} = y_k - 1$$

So decision parameter at $k + 1$ can be calculated as:

$$\begin{aligned}d_{k+1} &= (x_{k+1} + 1)^2 + (y_{k+1} - 1)^2 - R^2 \\&= (x_k + 1)^2 + (y_k - 1 - 1)^2 - R^2 \\d_{k+1} &= d_k - 2(y_{k+1}) + 1 \dots\dots\dots 15\end{aligned}$$

similarly, in case of diagonal move

$$\begin{aligned}x_{k+1} &= x_k + 1 \\y_{k+1} &= y_k - 1\end{aligned}$$

So decision parameter at $k + 1$ can be calculated as:

$$\begin{aligned}d_{k+1} &= (x_{k+1} + 1)^2 + (y_{k+1} - 1)^2 - R^2 \\&= (x_k + 1 + 1)^2 + (y_k - 1 - 1)^2 - R^2 \\d_{k+1} &= d_k + 2(x_{k+1}) - 2(y_{k+1}) + 2 \dots\dots\dots 16\end{aligned}$$

Bresenham's Circle Drawing Algorithm (for first octant clockwise display of pixels):

1. Accept the centre (X_c, Y_c) and radius (r) of circle from user.
2. Initialize
 - $x = 0$ and $y = r$
3. Calculate initial value of error term
 - $d_k = 2(1 - r)$
4. Execute following statements (a to d) while y is greater or equal to x
 - While($y \geq x$)
 - a) Display pixel (x, y)
 - putpixel($X_c + x, Y_c + y$)
 - b) Check if error term is less than zero then set delta and select either horizontal pixel or diagonal pixel.
 - if($d_k < 0$) then
 - delta = $2d_k + 2y - 1$
 - If($\text{delta} \leq 0$) then
 - //select horizontal pixel
 - $x = x + 1$
 - $d_k = d_k + 2x + 1$
 - else
 - // select diagonal pixel
 - $x = x + 1$
 - $y = y - 1$
 - $d_k = d_k + 2x - 2y + 2$
 - c) Else If error term is greater than zero then set delta and select either diagonal or vertical pixel
 - else if($d_k > 0$)
 - delta_dash = $2d_k - 2x - 1$
 - If($\text{delta_dash} \leq 0$) then
 - // select diagonal pixel
 - $x = x + 1$
 - $y = y - 1$

```

    dk = dk + 2x - 2y + 2
else
    // select vertical pixel
    y = y-1
    dk = dk - 2y + 1

```

d) Else if error term is zero then select diagonal term directly

```

else if(dk == 0)
    // select diagonal pixel
    x = x+1
    y = y-1
    dk = dk + 2x - 2y + 2

```

Midpoint circle Algorithm:

In this algorithm the user has to give the radius r and screen center position (x_c, y_c) . First we have to calculate pixel positions around a circle path centered at the coordinate origin $(0, 0)$. Then each calculated position (x, y) is moved to its proper screen position by adding x_c to x and y_c to y . Then calculate the value of the decision parameter and determine the symmetry points in the other seven octants. To apply the midpoint method we define a circle function:

$$f_{\text{circle}}(x, y) = x^2 + y^2 - r^2 \dots\dots\dots 1$$

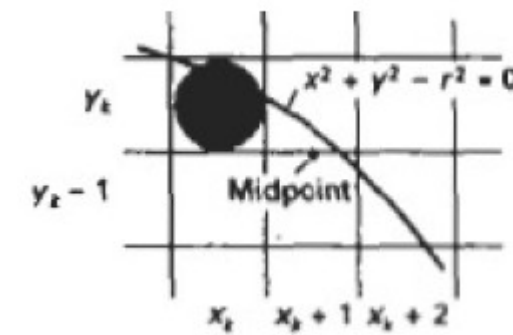


Figure 5: Midpoint of the possible pixels at sampling position $x_k + 1$.

Any point (x, y) on the boundary of the circle with radius r satisfies the equation $f_{\text{circle}}(x, y) = 0$. If the point is in the interior of the circle, the circle function is negative. And if the point is outside of the circle, the circle function is positive. Then for any sampling position there are always two possible pixels near the circular path so the mid positions between pixels near the circle path at each sampling step are obtained and the circle function is applied at that midpoint. If circle equation evaluates positive means midpoint is outside so we select inner pixel otherwise we select outer pixel. Thus, the circle function is the decision parameter in the midpoint algorithm and we can setup incremental calculations for this

function. Our decision parameter is the circle function $f_{\text{circle}}(x, y) = x^2 + y^2 - r^2$ evaluated at the midpoint between the two pixels:

$$\begin{aligned} p_k &= f_{\text{circle}}(x_k+1, y_k-1/2) \\ &= (x_k+1)^2 + (y_k-1/2)^2 - r^2 \end{aligned}$$

If $p_k < 0$, this midpoint is inside the circle and the pixel on scan line y_k is closer to the circle boundary. Otherwise, the mid position is outside or on the circle boundary and we select the pixel on scan line y_{k+1} . Successive decision parameters are obtained using incremental calculations. Here the next sampling position is $x_{k+1}+1 = x_k+2$

$$\begin{aligned} p_k &= f_{\text{circle}}(x_{k+1}+1, y_{k+1}-1/2) \\ &= [(x_k+1)+1]^2 + (y_{k+1}-1/2)^2 - r^2 \end{aligned}$$

OR

$$p_{k+1} = p_k + 2(x_k+1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$$

Where y_{k+1} is either y_k or y_k-1 depending upon the sign of p_k .

Evaluation of the terms $2x_{k+1}$ and $2y_{k+1}$ can also be done incrementally as $2x_{k+1} = 2x_k + 2$ and $2y_{k+1} = 2y_k - 2$. The initial decision parameter is obtained by evaluating the circle function at the start position $(x_0, y_0) = (0, r)$

$$p_0 = f_{\text{circle}}(1, r-1/2) = 1 + (r-1/2)^2 - r^2 = 5/4 - r$$

The midpoint method calculates pixel positions along the circumference of a circle using integer additions and subtractions assuming that the circle parameters are specified in integer screen coordinate.

Mid Point circle Drawing Algorithm:

1. Input radius r and circle center (x_c, y_c) , and obtain the first point on the circumference of the circle centered on the origin as $(x_0, y_0) = (0, r)$
2. Calculate the initial value of the decision parameter as $p_0 = 5/4 - r$
3. At each x_k position, starting at $k = 0$, perform the following test:
If $p_k < 0$, the next point along the circle centered on $(0, 0)$ is $(x_k + 1, y_k)$ and

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

$$\text{Where } 2x_{k+1} = 2x_k + 2$$

2. Otherwise, the next point along the circle is $(x_k + 1, y_k - 1)$ and

$$p_{k+1} = p_k + 2x_{k+1} - 2y_{k+1} + 1$$

$$\text{Where } 2x_{k+1} = 2x_k + 2 \text{ and } 2y_{k+1} = 2y_k - 2.$$

4. Determine symmetry points in the other seven octants.
5. Move each calculated pixel position (x, y) onto the circular path centered on (x_c, y_c) and plot the coordinate values :

$$x = x + x_c$$

$$y = y + y_c$$

5. Repeat steps 3 through 5 until $x \geq y$.

Equilateral triangle: In geometry, an equilateral triangle is a triangle in which all three sides are equal. In the familiar Euclidean geometry, equilateral triangles are also equiangular; that is, all three internal angles are also congruent to each other and are each 60°

Principal properties: Denoting the common length of the sides of the equilateral triangle as a , we can determine using the Pythagorean theorem that:

- The radius of the circumscribed circle is : $R = \frac{\sqrt{3}}{3}a$
- The radius of the inscribed circle is r or
- The geometric center of the triangle is the center of the circumscribed and inscribed circles
- And the altitude (height) from any side is $\frac{\sqrt{3}}{2}a$.

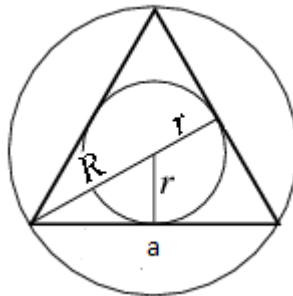


Figure 6: Inscribed and circumscribed circles of a equilateral triangle.

ALGORITHM:

- Step 1: Accept the bottom left vertex coordinates and the length of the equilateral triangle.
- Step 2: Find out the remaining vertices of the equilateral triangle and draw it using DDA algorithm.
- Step 3: Find out the center and radii of inscribed and circumscribed circles using the theory explained above
- Step 4: Draw the inscribed and circumscribed circles using the Bresenham's/Midpoint circle drawing algorithm.
- Step 5: Stop.

INPUT:

Equilateral triangle vertices , inner circle's centre point and its radius.

OUTPUT:

The output will be the equilateral triangle with inscribed and circumscribed circle.

Assignment No 3

AIM:

To implement mouse handling in OpenGL .

PROBLEM STATEMENT:

Draw the polygons by using the mouse. Choose colors by clicking on the designed color pane. Use window port to draw. (Use DDA algorithm for line drawing)

THEORY:**Event-driven Programs**

- Respond to events, such as mouse click or move, key press, or window reshape or resize.
- Programmer provides “call-back” functions to handle each event.
- Call-back functions must be registered with OpenGL to let it know which function handles which event.

```
// include OpenGL libraries
```

```
void main()
```

```
{
```

```
    glutDisplayFunc(myDisplay); // register the redraw function
```

```
    glutReshapeFunc(myReshape); // register the reshape function
```

```
    glutMouseFunc(myMouse); // register the mouse action function
```

```
    glutMotionFunc(myMotionFunc); // register the mouse motion function
```

```
        glutKeyboardFunc(myKeyboard); // register the keyboard action  
function
```

```
    .....
```

```
    glutMainLoop();           // enter the unending main loop
```

```
}
```

glutMouseFunc(void (*func) (int button, int state, int x, int y));

- Detects mouse click.
- Func() handles the mouse event. It has three arguments button (GLUT_LEFT_BUTTON, GLUT_RIGHT_BUTTON, GLUT_MIDDLE_BUTTON), state (GLUT_UP, GLUT_DOWN), x and y are the coordinates where mouse is moved.

Two types of functions for mouse motion detection.

1. Active (when mouse click is there) glutMotionFunc(void (*func) (int x, int y))
2. Passive (motion without click) glutPassiveMotionFunc(*func) (int x, int y))

glutPassiveMotionFunc(myPassiveMotionFunc)

- Handles case where mouse enters the window with *no* buttons pressed.

glutKeyboardFunc(myKeyboardFunc);

- Handles key presses and releases. Knows which key was pressed and mouse location.

glutMainLoop()

- Runs forever waiting for an event. When one occurs, it is handled by the appropriate callback function.

ALGORITHM:

1. Implement the designated color pane.
2. Select the color from the pane using mouse.
3. Set the selected color.
4. Draw the polygon by using mouse and line drawing algorithm.
5. Fill the polygon by using the color and fill line algorithm

INPUT:-

OUTPUT:

Filled polygon with color selected from color pane.

Assignment No 4

AIM:

To implement rotation and seed fill algorithm.

PROBLEM STATEMENT:

Draw a 4X4 chessboard rotated 45° with the horizontal axis. Use Bresenham algorithm to draw all the lines. Use seed fill algorithm to fill black squares of the rotated chessboard

THEORY:

Bresenham's line drawing:

This is the most correct and efficient raster line drawing algorithm. It scans line using only incremental integer calculations.

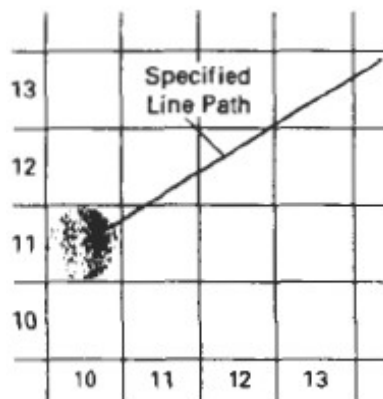


Figure 1: Section of a display screen where a straight line segment is to be plotted.

Figure 1 and 2 illustrates the section of a display screen where straight line segments are to be generated. The vertical axes shows scan line positions and the horizontal axes identify pixel columns. Sampling at unit x intervals in these examples we need to decide which of the two possible pixel positions is closer to the line path at each sample step starting from left

endpoint shown in figure 1 we need to determine at next sample position whether to plot pixel at position at (11, 11) or the one at (11,12). Similarly in figure 2 shows line path for a line having negative slope starting from the left endpoint at (34, 34) in this case whether to select next pixel position as (51, 50) or (51, 49)?

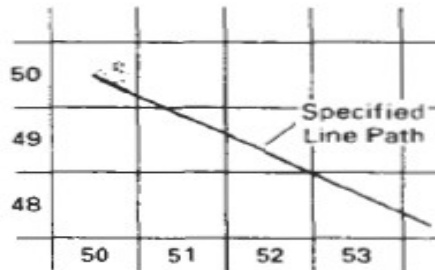


Figure 2: Section of a display screen where a negative slope line segment is to be plotted.

The solution for above question can be given by Bresenham's line drawing algorithm. Bresenham's algorithm uses an integer parameter and tests the sign of integer parameter whose value is proportional to the difference between separations of the two pixel positions from the actual line path.

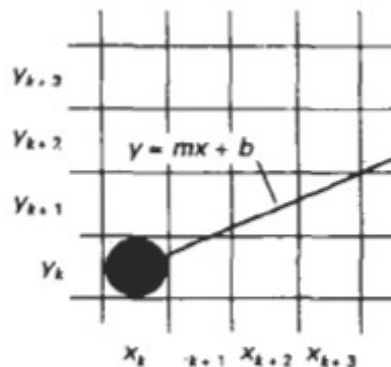


Figure 3: Section of the screen grid showing a pixel in column x_k on scan line y_k that is to be plotted along the path of a line segment with slope $0 < |m| < 1$.

Consider the scan-conversion process for lines with absolute slope less than or equal to 1. Pixel positions along a line path are then determined by sampling at unit x intervals. Starting from the left endpoint (x_0, y_0) of a given line, we step to each successive column (x position) and plot the pixel whose column y value is closest to the line path. Figure 3 shows the k^{th} step of this process. Assuming that we have already displayed the pixel at

(x_k, y_k) , we next need to decide which pixel to plot in column position x_k+1 . Our choices are the pixels at positions (x_k+1, y_k) and (x_k+1, y_k+1)

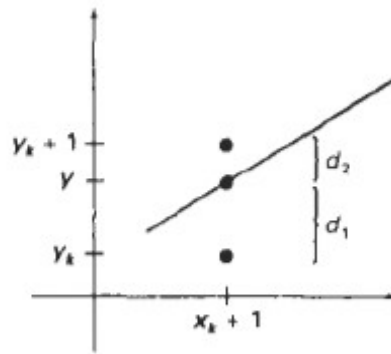


Figure 4: Distances between pixel positions & line y coordinate at sampling position $x_k + 1$.

At sampling position $x_k + 1$, we label vertical pixel separations from the mathematical line path as p_1 , and p_2 see figure 4.

The y coordinate on the mathematical line at pixel scan-line position $y_k + 1$ is calculated as

$$y = m(x_k + 1) + c \dots\dots\dots(1) \text{ then}$$

$$p_1 = y - y_k$$

$$= m(x_k + 1) + c - y_k$$

$$p_2 = (y_k + 1) - y$$

$$= y_k + 1 - m(x_k + 1) - c$$

The difference between these two separations is

$$p_1 - p_2 = 2m(x_k + 1) + 2c - 2y_k - 1 \dots\dots\dots(2)$$

now $m = dy/dx$ putting in equation 1 & multiplying it by dx on both sides in order to make it in integer calculations we get

$$d_k = dx(p_1 - p_2)$$

$$d_k = 2dyx_k - 2dxy_k + [2dy + 2cdx - dx] \dots\dots\dots(3)$$

The term $[2dy + 2cdx - dx]$ is constant for all x_k, y_k

So let $C = [2dy + 2cdx - dx]$

$$d_k = 2dyx_k - 2dxy_k + C \dots\dots\dots(4)$$

Now At step $k+1$ the decision parameter will be

$$d_{k+1} = 2dy(x_{k+1}) - 2dxy_{k+1} + C \dots\dots\dots(5)$$

but $x_{k+1} = x_k + 1$ as we are sampling at unit x intervals

$$d_{k+1} = 2dyx_k + 2dy - 2dxy_{k+1} + C \dots\dots\dots(6)$$

equation 6-4 gives us

$$d_{k+1} = d_k + 2dy - 2dx(y_{k+1} - y_k)$$

where the term $y_{k+1}-y_k$ is either 0 or 1 depending on the sign of decision parameter d_k

now if d_k is -ve i.e. p_1 is less

so (x_k+1, y_k) is selected and

$$d_{k+1} = d_k + 2dy$$

and if d_k is +ve i.e. p_2 is less

so (x_k+1, y_k+1) is selected and

$$d_{k+1} = d_k + 2dy - 2dx$$

The initial decision parameter can be calculated by putting $(x_k, y_k) = (0, 0)$, $c = 0$ in equation (3)

$$\text{so } d_0 = 2dy - dx$$

Bresenham's line drawing algorithm for $|m| \leq 1$:

1. Input the two line endpoints and store the left endpoint in (x_0, y_0)
2. Load (x_0, y_0) into the frame buffer; that is, plot the first point.
3. Calculate constants dx , dy , $2dy$, and $2dy - 2dx$, and obtain the starting value for the decision parameter as

$$d_0 = 2dy - dx$$

4. At each x_k along the line, starting at $k = 0$, perform the following test:

If $d_k < 0$, the next point to plot is (x_k+1, y_k) and

$$d_{k+1} = d_k + 2dy$$

Otherwise, the next point to plot is (x_k+1, y_k+1)

$$d_{k+1} = d_k + 2dy - 2dx$$

6. Repeat step 4 dx times.

Rotation about Pivot point:

By performing following successive transformations we can find out rotation of an object about any pivot point (fixed point).

1. Translate the pivot point along with the object so that the pivot-point can be brought to origin.
2. Rotate (anticlockwise) the object about origin.
3. Translate the pivot point along with the object so that the pivot-point can be brought to its original position.

This transformation sequence is shown in the Fig 5

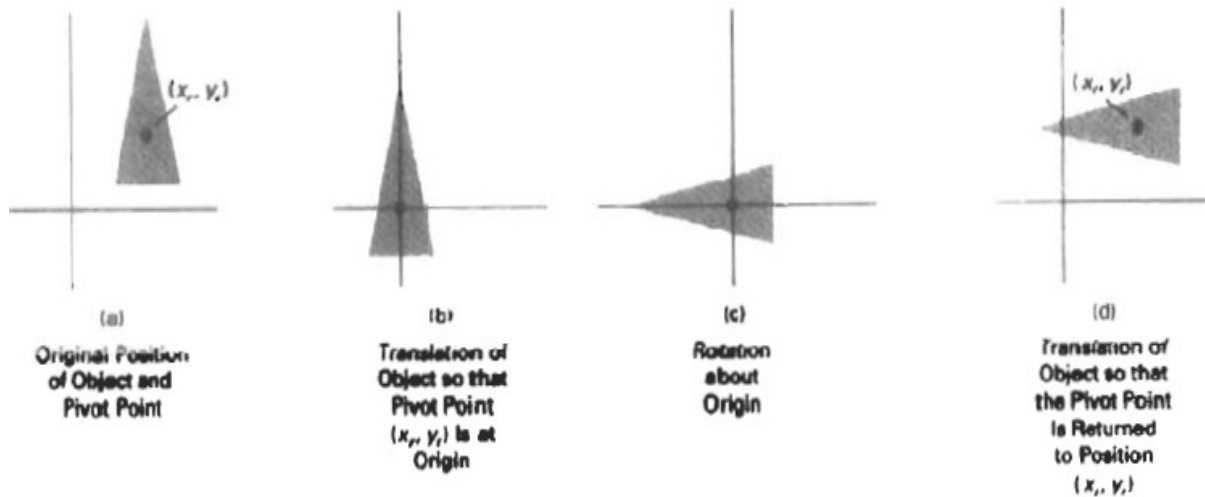


Figure 5: steps involved in 2D rotation about pivot point.

The composite transformation for rotation about fixed point can be obtained as:

$$\begin{aligned}
 & \begin{bmatrix} 1 & 0 & r_x \\ 0 & 1 & r_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos & -\sin & 0 \\ \sin & \cos & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -r_x \\ 0 & 1 & -r_y \\ 0 & 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} \cos & -\sin & r_x(1-\cos)+r_y\sin \\ \sin & \cos & r_y(1-\cos)-r_x\sin \\ 0 & 0 & 1 \end{bmatrix} \quad \text{.....7}
 \end{aligned}$$

It can also be written in short form as:

$$\mathbf{R}(r_x, r_y, \theta) = \mathbf{T}(r_x, r_y) \cdot \mathbf{R}(\theta) \cdot \mathbf{T}(-r_x, -r_y)$$

.....8

Polygon Filling - Seed Fill:

Basically polygon or area filling is the process of coloring inside area of the said object. If the polygon or object is defined at pixel level then the seed fill algorithms are used to fill such objects. The seed fill algorithms are of two types viz. Boundary - fill algorithm and flood - fill algorithm.

1. Boundary - fill algorithm

In boundary fill algorithm we start the filling procedure with a pixel known as seed pixel, inside the area of the object to be filled, and color the inside region outward towards the boundary. The object can be filled with the color same as the object boundary.

A boundary - fill algorithm inputs the coordinates of the seed point (x, y), boundary color and fill color. It tests the color of the seed point; if the seed point is not having color same as boundary then it fills that point with fill color. The procedure is carried out recursively by considering its neighbors as new seed.

There are two methods for identifying the neighbors of the current processing pixel. Figure 6 a) shows four neighbors surrounded by current processing point. The four positions are at right, left, above, and below the current pixel. Objects filled by this method are called as 4 - connected see figure 6 a). Another method uses the neighbors of previous method including diagonal pixels see figure 6 b). Total 8 pixels are tested for filling. This method is known as 8 - connected and used to fill areas with complex regions.

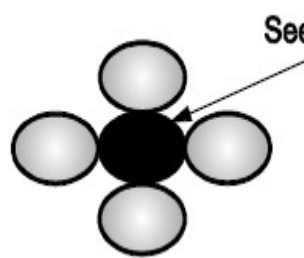


Figure 6 a) 4 connected method

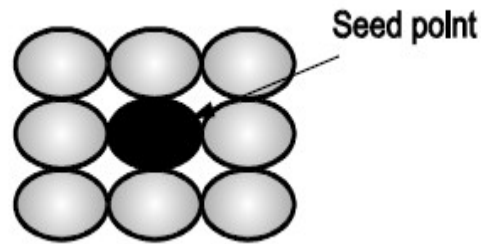


Figure 6 b) 8 connected method

The following procedure explains the recursive method for boundary - fill

```
void boundary_fill(int x ,int y, pixel boundaryColor, pixel fillColor)
{
    glReadPixels(x,y,1,1,GL_RGB,GL_UNSIGNED_BYTE, &c);
    if(c.red!= boundaryColor.red && c.green!=
boundaryColor.green && c.blue!=
boundaryColor.blue && c.red!= fillColor.red && c.green!=
fillColor.green &&
c.blue!= fillColor.blue)
    {
        glColor3ub(fillColor.red, fillColor.green, fillColor.blue);
        glBegin(GL_POINTS);
            glVertex2i(x, y);
        glEnd();
        glFlush();
        boundary_fill(x+1,y, boundaryColor, fillColor);
        boundary_fill(x-1,y, boundaryColor, fillColor);
        boundary_fill(x,y+1, boundaryColor, fillColor);
        boundary_fill(x,y-1, boundaryColor, fillColor);
    }
}
```

2) Flood fill algorithm

If the object to be filled is having more than one boundary color then flood fill algorithm can be used to fill such objects. In flood fill algorithm we start the filling procedure with a pixel known as seed

pixel, inside the area of the object to be filled. We can fill such objects by replacing the given interior color with filling color

A flood – fill algorithm inputs the coordinates of the seed point (x, y), old color and fill color. It tests the color of the seed point; if the seed point is having color same as old color then it fills that point with fill color. The procedure is carried out recursively by considering its neighbors as new seed.

The following procedure explains the recursive method for boundary – fill

```
void flood_fill(int x ,int y, pixel oldColor, pixel fillColor)
{
    glReadPixels(x,y,1,1,GL_RGB,GL_UNSIGNED_BYTE, &c);
    if(c.red== oldColor.red  &&  c.green== oldColor.green  &&
c.blue==
        oldColor.blue)
    {
        glColor3ub(fillColor.red, fillColor.green, fillColor.blue);
        glBegin(GL_POINTS);
            glVertex2i(x, y);
        glEnd();
        glFlush();
        flood_fill(x+1,y, oldColor, fillColor);
        flood_fill(x-1,y, oldColor, fillColor);
        flood_fill(x,y+1, oldColor, fillColor);
        flood_fill(x,y-1, oldColor, fillColor);
    }
}
```

ALGORITHM:

- Step 1: Accept the bottom left vertex coordinates of the 4x4 chessboard and the length of chessboard.
- Step 2: Divide the length into 4 segments and find out the length of each box of the chessboard.
- Step 3: By using length of chessboard and length of each box identify the endpoints of the lines to be drawn to draw the chessboard. Note that total 10 lines should be drawn to draw the final chessboard so identify 10 pair of endpoints and store them in a column vector form (Column matrix form) line by line.
- Step 4: Draw the chessboard by drawing lines between each pair of endpoints of each line using column matrix representation of endpoints.
- Step 5: Apply above mentioned anticlockwise rotation to rotate the chessboard by 45° about bottom left vertex of chessboard.

Step 6: Identify the alternate boxes of the chessboard and apply any seed fill method to fill them individually with black color.

Step 7: Stop.

INPUT:

The bottom left vertex coordinates and length of the chessboard to be drawn.

OUTPUT:

Rotated chessboard about bottom left corner by 45° .

Assignment No 5

AIM:

To Implement Cohen Sutherland algorithm.

PROBLEM STATEMENT:

Implement Cohen Sutherland algorithm to clip any given Line/polygon. Provide the vertices of the Line/polygon to be clipped and pattern of clipping interactively.

THEORY:

Cohen-Sutherland line clipping:-

Clipping means omitting the part of picture which lies outside the window. It is simplest to clip a straight line. Here we have to examine each and every line which we are going to display. We have to determine whether or not a line is completely outside the window. If the line is completely inside then display it fully. If it is completely outside discard it.

Sutherland -Hodgement polygon clipping:-

Basically polygon is a set of lines only. But it is closed figure. If we use clipping algorithm, then each line of polygon is separately considered and clipped. After clipping we will get a set of lines i.e a polygon. This is called as polygon clipping.

ALGORITHM:

Cohen-sutherland Algorithm:-

1. Accept the window co-ordinate from user and store in x_L, x_H, y_L, y_H .
2. Accept the end points A and B of line with component A_x, A_y, B_x, B_y ;
3. Initialize A and B code as array of size four.
4. Calculate Acode and Bcode by comparing,
If $A_x < x_L$ then Acode (4)=1 else 0
If $A_x > x_H$ then Acode (3)=1 else 0
If $A_y < y_L$ then Acode (2)=1 else 0
If $A_y > y_H$ then Acode (1)=1 else 0
5. Check if Acode and Bcode are 0000 if yes, display line AB. If not then proceed.

6. Take logical AND of Acode and Bcode, check the result, if it is nonzero, discard the line and go to step 8 else proceed.
7. Divide the line by finding intersection of line with window boundary and give new outcode to intersection point and go to step 5
8. END.

INPUT:

INPUT for Cohen-sutherland Algorithm:-

Input: Rectangular area of interest (Defined by below four values which are coordinates of bottom left and top right)

$x_{\min} = 4, y_{\min} = 4, x_{\max} = 10, y_{\max} = 8$

A set of lines (Defined by two corner coordinates)

Line1: $x_1 = 5, y_1 = 5, x_2 = 7, y_2 = 7$

Line 2: $x_1 = 7, y_1 = 9, x_2 = 11, y_2 = 9$

Line 3: $x_1 = 1, y_1 = 5, x_2 = 4, y_2 = 1$

OUTPUT:

OUTPUT for Cohen-sutherland Algorithm

Line 1: Accepted from (5, 5) to (7, 7)

Line 2: Accepted from (7.8, 8) to (10, 5.25)

Line 3: Rejected

Assignment No 6

AIM:

To implement 2D transformations.

PROBLEM STATEMENT:

Implement translation, sheer, rotation and scaling transformations on equilateral triangle and rhombus.

THEORY:

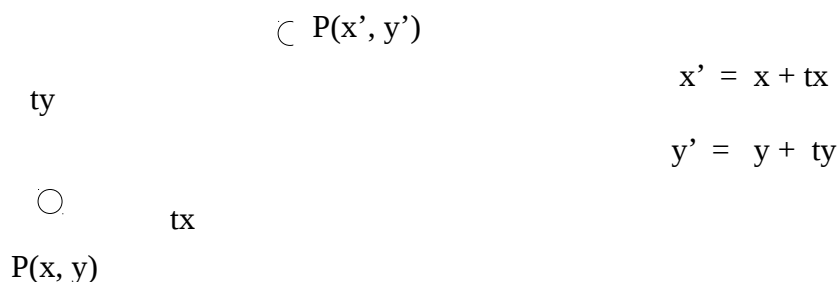
2D transformation:

Transformation allows us to uniformly alter the entire picture instead of drawing a new picture.

Transformation means translation, scaling, rotation, shearing or combination of all.

Translation:

Translation is applied to an object by repositioning it along a straight line path from one coordinate location to another. This can be easily done by adding to each point the amount by which we want to shift the point.

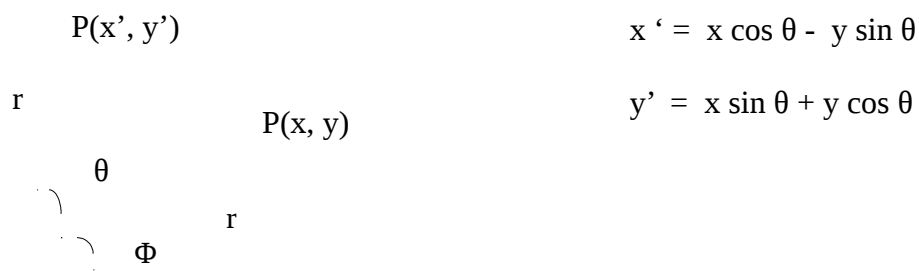


Translation Matrix :

$$T = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation:

The 2D rotation is applied to object by repositioning it along circular path in xy. To generate rotation we specify a rotation angle and position (xr , yr) of rotation point .



$$x' = x \cos \theta - y \sin \theta$$

$$y' = x \sin \theta + y \cos \theta$$

Rotation Matrix :

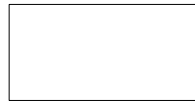
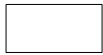
$$R = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Scaling:

Scaling transformation alters the size of an object. This operation can be carried out for polygon by multiplying coordinates valued (x, y) of each vertex by scaling factors S_x and S_y to produce the transformed coordinates.

Before Scaling

After Scaling



$$x' = x \cdot S_x$$

$$y' = y \cdot S_y$$

Scaling Matrix :

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Shearing:

A transformation that slants the shape of an object is called the shear transformation.

X Shear



Original Object

Object after x shear

$$X_{sh} = \begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Y Shear:



Original Object

Object after y shear

$$Y_{sh} = \begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

ALGORITHM:

1. Accept Coordinates of object and represent in 2-D Vector
2. Display the original shape of object using 2-D Vector
3. Accept the transformation factors as per the choice provided by user
 - If choice = Translation then Accept tx , ty ; Go To Step 4
 - If choice = Rotation then Accept Orientation & Angle of rotation ; Go To Step 4
 - If choice = Scaling then Accept Sx , Sy ; Go To Step 4
 - If choice = X- Shear then Accept Shx ; Go To Step 4
 - If choice = Y- Shear then Accept Shy ; Go To Step 4
4. Represent transformation factors using 2-D Vector
5. Apply the chosen transformation on original shape using corresponding equation as explained in theory [2-D Vector form]
6. Display the transformed shape of object using 2-D Vector

INPUT:

Coordinates of object [equilateral triangle and rhombus].

OUTPUT:

Transformed object co-ordinates after applying above mention transformations

Assignment No 7

AIM:

To implement Cube rotation.

PROBLEM STATEMENT:

Implement Cube rotation about vertical axis passing through its centroid.

THEORY:

Methods for geometric transformations and object modeling in three dimensions are extended from two-dimensional methods by including considerations for the z coordinate.

However, the extension for three-dimensional rotation is less straightforward. In three-dimensional space, one can select any spatial orientation for the rotation axis. Most graphics packages handle three-dimensional rotation as a composite of three rotations, one for each of the three Cartesian axes. Alternatively, a user can easily set up a general rotation matrix, given the orientation of the axis and the required rotation angle. In 2D, rotation is rotation about a point, which is usually taken to be the origin. In 3D, rotation is rotation about a line, which is called the axis of rotation. Any line can be an axis of rotation, but we generally use an axis that passes through the origin. The most common choices for axis of rotation are the coordinate's axes, that is, the x-axis, the y-axis, or the z-axis. Sometimes, however, it's convenient to be able to use a different line as the axis. Table 8.1 highlights the rotation axes and their corresponding angle of rotations along with equivalent matrices.

Table 8.1 Basic 3- D Rotations

Axis of Rotation	Rotation Equation	Rotation Matrix
Rotation about Z -axis (xy plane)	$x' = x \cos(q) - y \sin(q)$ $y' = x \sin(q) + y \cos(q)$ $z' = z$	$\begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Rotation about X-axis (yz plane)	$y' = y \cos(q) - z \sin(q)$ $z' = y \sin(q) + z \cos(q)$ $x' = x$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Rotation about Y-axis (xz plane)	$z' = z \cos(q) - x \sin(q)$ $x' = z \sin(q) + x \cos(q)$ $y' = y$	$\begin{bmatrix} \cos \theta & 0 & -\sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

A rotation matrix for any axis that does not coincide with a coordinate axis can be set up as a composite transformation involving combinations of translations and the coordinate-axes rotations. The required composite matrix can be obtained by first setting up the transformation sequence that moves the selected rotation axis onto one of the coordinate axes. Then set up the rotation matrix about that coordinate axis for the specified rotation angle. The last step is to obtain the inverse transformation sequence that returns the rotation axis to its original position. In the special case where an object is to be rotated about an axis that is parallel to one of the coordinate axes, then desired rotation can be obtained with the following transformation sequence.

1. Translate the object so that the rotation axis coincides with the parallel coordinate axis.
2. Perform the specified rotation about that axis.
3. Translate the object so that the rotation axis is moved back to its original position.

ALGORITHM:

1. Accept Coordinates of cube wrt x,y and z axes and represent in 3-D Vector
2. Display the original shape of cube using 3-D Vector
3. Accept the Rotation Orientation, axes of rotation & Angle of rotation
4. Represent rotation factors using 3-D Vector

5. Apply the rotation on original cube wrt corresponding equation as per chosen axes using 3-D Vector form (Refer Table 8.1)
6. Display the rotated cube using 3-D Vector

INPUT:

Coordinates of cube w.r.t. x, y and z axes.

OUTPUT:

Transformed object co-ordinates after applying 3D rotation

Assignment No 8

AIM:

To implement Koch curve.

PROBLEM STATEMENT:

Generate fractal patterns by using Koch curves.

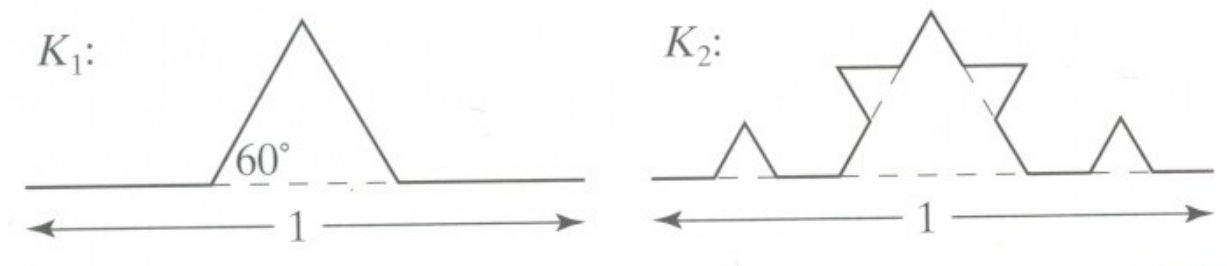
THEORY:**Introduction:**

Curves are one of the most essential objects to create high resolution graphics. While using many small poly lines allows creating graphics that appear smooth at fixed resolutions, they do not preserve smoothness when scaled and also require a tremendous amount of storage for any high resolution image. Curves can be stored much easier, can be scaled to any resolution without losing smoothness, and most importantly provide a much easier way to specify real world objects.

The Koch snowflake (also known as the Koch curve, Koch star, or Koch island) is a mathematical curve and one of the earliest fractal curves to have been described. It is based on the Koch curve, which appeared in a 1904 paper titled "On a continuous curve without tangents, constructible from elementary geometry" by the Swedish mathematician Helge von Koch. Writing a function that recursively calls itself is one technique for generating a fractal pattern on screen.

The progression for the area of the snowflake converges to $\frac{8}{5}$ times the area of the original triangle, while the progression for the snowflake's

perimeter diverges to infinity. Consequently, the snowflake has a finite area bounded by an infinitely long line.



ALGORITHM:

Pseudo code, Start with straight line of length 1

Recursively Divide line into 3 equal parts

Replace middle section with triangular bump with sides of length 1/3

New length = 4/3

Can form Koch snowflake by joining three Koch curves

Perimeter of snowflake grows as

$$P_i = 3 \left(\frac{4}{3} \right)^i$$

Where P_i is the perimeter of the ith snowflake iteration

To draw K_n:

If (n equals 0) draw straight line

Else{

 Draw K_{n-1}

 Turn left 60°

 Draw K_{n-1}

 Turn right 120°

 Draw K_{n-1}

 Turn left 60°

}

INPUT:

Draw a Von Koch Curve

Object is assumed as triangle.

Iteration for a pattern is taken as an Input.

OUTPUT:

Display the Koch curve with pattern.

Assignment No 9

AIM:

To implement animation based assignments.

PROBLEM STATEMENT:

Implement any one of the following animation assignments,

- i) Clock with pendulum
- ii) National Flag hoisting
- iii) Vehicle/boat locomotion
- iv) Falling Water drop into the water and generated waves after impact

Kaleidoscope views generation (at least 3 colorful patterns)

THEORY:**Computer Animation**

In its simplest form, computer animation simply means:

Using a standard renderer to produce consecutive frames where in the animation consists of relative movement between rigid bodies and possibly movement of the view point or virtual camera. This is completely analogous to model animation where scale models are photographed by special cameras

ALGORITHM:

Displaying animation sequences

- To achieve smooth animation, a sequence of images (frames) has to be presented on a screen with the speed of at least 30 per second
- Animations frames can be
 - pre-computed in advance and pre-loaded in memory
 - computed in real time (e.g. movement of the cursor)

INPUT:

i) Clock with pendulum

Co-ordinate of object (for e.g. Center Co-Ordinate of Circle:-xCenter, yCenter, Radius of Circle is assume for clock and pendulum)

OUTPUT:

Continuously displaying the object with specific time period (for e.g. 30 second)