Assignment No: 1

- Title: Install explore the OpenGL
- Theory:

OpenGL (open Graphics Library) is a cross-platform, Hardware-accelerated, language-independent, industrial Standard API for producing 3D (or 2D including) graphics. Modern computers have dedicated GPU (Graphics Processing Unit) with its own memory to speed up graphics rendering. OpenGL is the software interface to graphics hardware. In other words, OpenGL graphic rendering commands issued by your applications could be directed to the graphic hardware and accelerated.

We use 3 sets of libraries in our OpenGL programs:

1) Core OpenGL (GL): Consist of hundreds of commands, which begin with a prefix "gl" (eg., glColor, glVertex, glTranslate, glRotate). The Core OpenGL models an object via a set of geometric primitives such as point, line and polygon.

2) OpenGL Utility Library (GLU): built on-top of the core OpenGL to provide important utilities (such as setting camera view and projection) and more buildings models (such as quadratic surfaces and polygon tessellation). GLU commands start with a prefix "glu" (eg. gluLookAt, gluPerspective).

3) OpenGL Utilities Toolkit (GLUT): OpenGL is designed to be independent of the windowing

System or operating system. GLUT is needed to interact with the operating system (such as creating a windows, handling key and mouse inputs) it also provides more (e.g., glutCreateViewWindow, glutMouseFunc) GLUT is perform independent, which is perform built on the top of platform-specific OpenGL extension such as GLX for X windows System, WGL for Microsoft Windows, and AGL, CGL or Cocoa for Mac OS.
Quoting from the Opengl.org: "GLUT is designed for constructing small-to medium sized OpenGL programs. While GLUT is well-suited to learning OpenGL and developing simple OpenGL appli, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native-native windows system toolkits. GLUT is simple, easy, and small."
Alternative of GLUT includes SDL, ...

4) OpenGL Extension Wrangler Library (GLEW):
"GLEW is a cross-platform open-source C/C++ extension loading library. GLEW provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target standalone utility called 'glewinfo.exe" (under the "bin" directory) can be used to produce the list of OpenGL functions supported by user graphics systems.

OpenGL as State Machine:
OpenGL operates as a state machine, and

maintain a set of state variables (such as the foreground color, background color, and many more) in a state machine. Once the value of a state variable is set, the value persists until a new value is given.
For example, we set the "clearing" (background) color to black once in initGL(). We use this setting to clear the window in the display() repeatedly (display() is called back whenever there is a window repaint request) - the clearing color is not changed in the entire program.

Naming Convention for OpenGL functions:

All OpenGL functions:
• begins with lowercase gl (for core OpenGL), glu (for OpenGL Utility) or glut (for OpenGL Utility toolkit).

followed by the purpose of a function, in camel case (initial-capitalized), eg., glColor to specify the drawing color, glVertex to define the position of a vertex.

• followed by specifications for the parameters, eg., glColor3f takes three float parameters. glVertex2i takes two int parameters.
(This is needed as C language does not support function overloading. Different versions of the function need to be written for different parameter lists).

The Conversion can be expressed as follows:

returnType glFunction[234][sifd](GLtype value,...);

returnType glFunction[234][sifd]v(GLtype * value);

Install OpenGL on Ubuntu
For installing OpenGL on Ubuntu, just execute the following command (like installing any other thing) in terminal:

```
sudo apt-get install freeglut3-dev
```

For working on Ubuntu operating System:

```
gcc filename.c -lGL -lGLU -lglut
```

where filename filename.c is the name of the file as with which this program is saved.

Install OpenGL on windows in Code::Blocks
1) Download code Block and install it.
2) Go to the link and to download zip file from the download link that appears after freeglut MinGW package with having link name as Download freeglut 3.0.0 for MinGW and extract it.
3) open notepad with run as administrator and open file from:
   1) This PC > C: (C-drive) > Programm files (x86) > CodeBlocks > share > CodeBlocks > templates > (then click to Show All files)
   2) Next, open glut.dop and search all glut32

and replace with freeglut.
3) Then, open from This PC > C: (C-drive) > Program Files (x86) > CodeBlocks > share > cb codeBlocks > templates > wizard > glut (then click to show All files)
4) open Wizard.script and here, also replace all glut32. with freeglut

4. Then go to freeglut folder (where it was downloaded) and
1) Include > GL and copy all these file from ~~there~~ there 2) Go to This PC > C: (C-drive) > Program Files (x86) > CodeBlocks > MinGW > include > GL and paste it.
3) Then, from download folder freeglut > lib, copy these files and go to This PC > C: (C-drive) > program files (x86) > CodeBlocks > MinGW > lib and paste it.
4) Again Go to downloaded folder freeglut > bin and to copy one file (freeglut.dll) from here and go to This PC > C: (C-drive) > windows > SysWow64 and paste this file.

5. Now Open Code::Blocks
1) Select file > New > Project > GLUT project > Next 2) Give project title anything and then choose Next. 3) For selecting GLUT's location: This PC > C: (C-drive) > Program files (x86) > CodeBlocks > MinGW. 4) Press Ok > Next > Finish

Conclusion:
We learned steps to install OpenGL

Assignment No: 2

- Title: Implement DDA and Bresenham line drawing algorithm to draw.
  i) Simple line
  ii) Dotted line
  iii) Dashed line
  iv) Solid line.

- Theory:

DDA Line Drawing Algorithm:

Digital differential analyzer (DDA) is a hardware or software used for linear interpolation of variables over an interval between start and end point. DDAs are used for rasterization of lines, triangles and polygons. DDA algorithm is an incremental scan conversion method. Here we perform calculations at each step using the results from the preceding step. The characteristic of the DDA algorithm is to take unit steps along one coordinate and compute the corresponding values along the other coordinate.

Algorithm:

Step 1: Start Algorithm

Step 2: Declare x1, y1, x2, y2, dx, dy, x, y as integer variables.

Step 3: Enter value of x1, y1, x2, y2

Step 4: Calculate dx = x2 - x1

Step 5: Calculate dy = y2 - y1

Step 6: If ABS(dx) > ABS(dy)
Then step = abs(dx)
Else

Step 7: xinc = dx/step
yinc = dy/step
assign x = x1
assign y = y1

Step 8: Set pixel(x, y)

Step 9: x = x + xinc
y = y + yinc
Set pixels (Round(x), Round(y))

Step 10: Repeate step 9 until x = x2

Step 11: End Algorithm.

Advantages of DDA Algorithm
1. It is the simplest algorithm and it does not require special skills for implementation.
2. It is 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:
1. Floating point arithmetic in DDA algorithm is still time-consuming.
2. The algorithm is orientation dependent.

Bresenham's line algorithm:

Bresenham's line algorithm that determines which points in an n-dimentional raster should be plotted in order to form a close approximation to a straight line between two given points. It is commonly used to draw lines on a computer screen, as it uses only integer addition, subtraction and bit shifting, all of which are very cheap operations in standard computer architectures. It is one of the earliest algorithm developed in the field of computer graphics. A minor extension to the original algorithm also deals with drawing ~~circui~~ circles.

Algorithm:

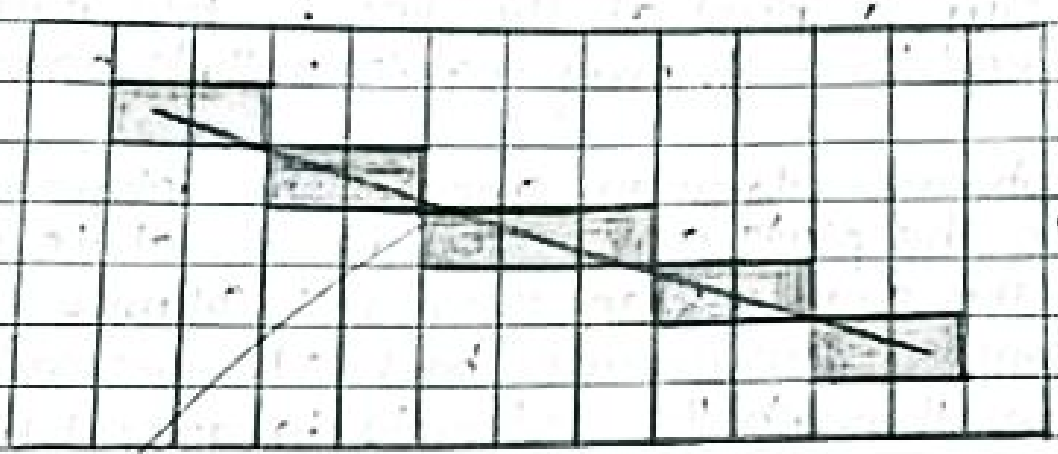


Illustration of the result of Bresenham's line algorithm. (0,0) is at the top left corner of the grid, (1,1) is at the top left end of the line and (11,5) is at the bottom right end of the line.

The common conventions will be used:

- ~~te~~ the top left is (0,0) such as that pixel

coordinates increase in the right and down directions and (eg that the pixel at (7,4) is directly above the pixel at (7,5)) and
- that the pixel centers have integer coordinates.

-The endpoints of the line are the pixels at (x_0, y_0) and (x_1, y_1), where the first coordinate of the pair is the column and the second is the row.

The algorithm will be initially presented only for the octant in which the segment goes down and to the right ($x_0 \le x_1$, $y_0 < y_1$), and its horizontal projection $x_1 - x_0$ is longer than the vertical projection $y_1 - y_0$ (the line has a negative slope whose absolute value is less than 1). In this octant, for each column x between x_0 and x_1, there is exactly one row y (computed by the algorithm) containing a pixel of the line, while each row between y_0 and y_1 may contain multiple rasterized pixels.

Bresenham's algorithm chooses the integer y corresponding to the pixel's center that is closest to ideal (fraction) y for the same x; on successive columns y can remain the same or increase by 1. The general equation of the line through the endpoints is given by:

$$\frac{y - y_0}{y_1 - y_0} = \frac{x - x_0}{x_1 - x_0}$$

Since we know the column, x, the pixel row, y, is the given by rounding the quantity to the nearest integer

$$y = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) + y_0$$

The slope $(y_1-y_0)/(x_0-x_0)$ depends on the endpoint coordinates only and can be precomputed and the ideal y for successive integer values of x can be computed starting from y_0 and repeatedly adding the slope.

Step 1 : Start Algorithm
Step 2 : Declare variable $x_1, x_2, y_1, y_2, d, i_1, i_2, dx, dy$.
Step 3 : Enter value of x_1, y_1, x_2, y_2
where x_1, y_1 are coordinates of starting point
And x_2, y_2 are coordinates of ending point.
Step 4 : Calculate $dx = x_2 - x_1$
Calculate $dy = y_2 - y_1$
Calculate $i_1 = 2 * dy$
Calculate $i_2 = 2 * (dy - dx)$
Calculate $d = i_1 - dx$
Step 5 : Consider (x, y) as starting point and x_{end} as minimum possible value of x
If $dx < 0$
Then $x = x_2$
$y = y_2$
$x_{end} = x_1$
If $dx > 0$
Then $x = x_1$
$y = y_1$
$x_{end} = x_2$
Step 6 : Generate point at (x, y) coordinates.
Step 7 : check if whole line is generated
If $x >= x_{end}$
Stop
Step 8 : Calculate co-ordinates of the next pixels
If $d < 0$

Difference between DDA & Bresenhams:

	DDA Algorithm	Bresenham's Algorithm
Arithmetic	DDA algorithm uses floating points i.e Real Arithmetic	Bresenhams algorithm uses fixed points i.e Integers Arithmetic
Operations	DDA Algorithm uses multiplication and division in its operations	Bresenhams algorithm uses only subtraction & addition in its operations
Speed	DDA Algorithm is rather slow than Bresenhams algorithm in line drawing because it uses real arithmetic (floating-point operations)	Bresenhams is faster than DDA algorithm in line drawing beaus it performs only addition and subtraction in its calculation and uses only integer arithmetic so it runs significantly faster.
Accuracy & Efficiency	DDA Algorithm is not as accurate and efficient as Bresenham's algorithm.	Bresenhams algorithm is more efficient and draws circles with much more accuracy than DDA algorithm
Drawing	DDA algorithm can draw circle circles and curves but that are not as accurate as Bresenham's algorithm.	Bresenhams algorithm can draw circuits and curves with much more accuracy than DDA algorithm.
Expensive	DDA Algorithm uses an enormous numbers of floating-point multiplications so it is expensive.	Bresenhams algorithm is less expensive than DDA algorithm as it uses only addition and subtraction.

Then d = d + i1

IF d ≥ 0

Then d = d + i2

Increment y = y + 1

Step 9: Increment x = x + 1

Step 10: Draw a point of lastest (x, y) Coordinates

Step 11: Go to Step 7

Step 12: End of Algorithm

- Conclusion:

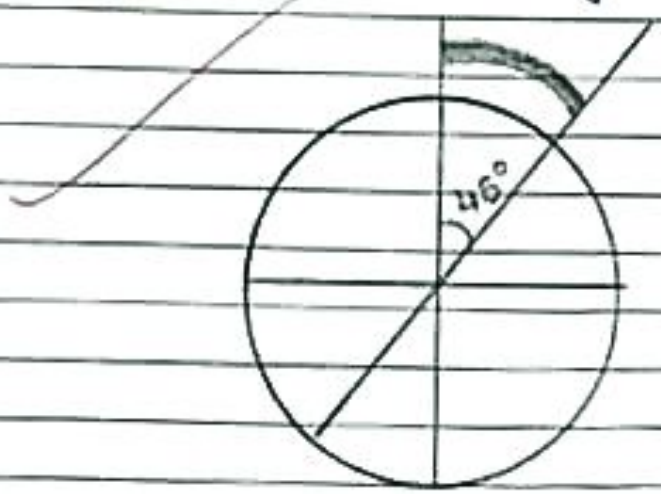
DDA and Bresenham's Line Drawing Algorithm are understand and executed successful

Assignment No. 3

- Title: Implement Bresenham's drawing Algorithm to draw any object. The object should be displayed in all the quadrants with respect to center and radius.

- Theory:

Circles have the property of being highly symmetrical, which is handy when it comes to drawing them on a display screen.

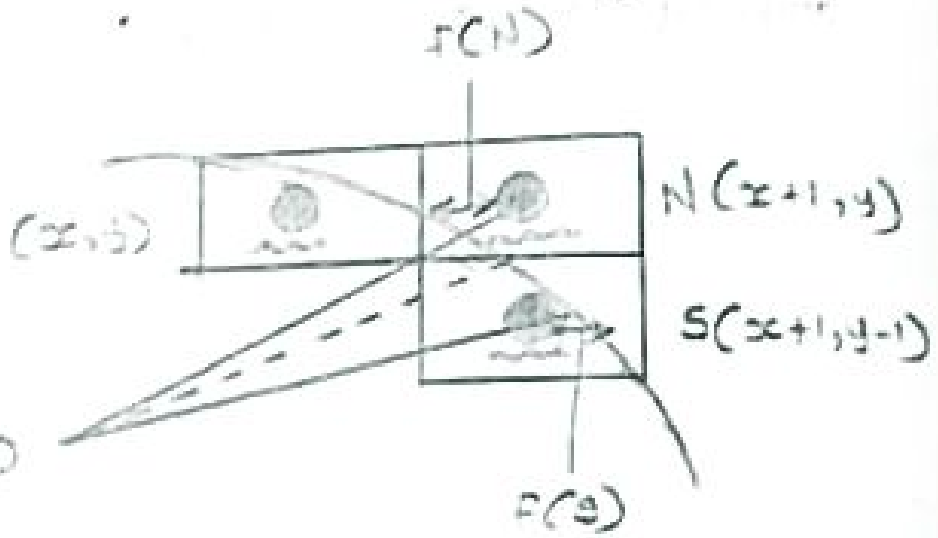


We know that there are 360 degrees in a circle. First we'll see that a circle is symmetrical about the x-axis, so only the first 180 degrees need to calculated. Next, we see that it's also symmetrical about the y axis, so now we only need to be calculated the first 90 degrees. Finally, we see that the circle is also symmetrical about the 45 degree diagonal axis; so we only need to calculated the first 45 degrees.

```
putpixel (centerx + x, centery + y)
putpixel (centerx + x, centery - y)
```

```
putpixel(center x+x, center y+y)
putpixel(center x-x, center y-y)
putpixel(center x+y, center y+x)
putpixel(center x+y, center y-x)
putpixel(center x-y, center y+x)
putpixel(center x-y, center y-x)
```

Now consider a very small continuous arc of the circle interpolated below, passing by the discrete pixels as shown:



As can be easily intercepted, the continuous arc of the circle cannot be plotted on a raster display device, but has to be approximated by nearest choosing the pixels to be highlighted. At any point (x,y), we have two choices - to choose the pixel on east of it, i.e. ~~N(x+1)~~ N(x+1,y) or the south east pixels S(x+1,y-1). To choose the pixel, we determine the errors involved with both N & S which are F(N) and F(S) respectively and whichever gives the lesser errors, we choose that pixel.

Let $d_i = F(N) + F(S)$, where d can be called as "decision" parameter, so that If $d_i <= 0$, then

N (x+1, y) is to be chosen as next pixel i.e. $x_{i+1} = x_i + 1$ and $y_{i+1} = y_i$,
and
If $d_i > 0$, then, S (x+1, y-1) is to be chosen as next pixel i.e $x_{i+1} = x_i + 1$ and $y_{i+1} = y_i - 1$

Algorithm:

Step 1: Start algorithm

Step 2: Declare p, q, x, y, r, d variables p, q are coordinates of the center of the circle r is the radius of the circle.

Step 3: Enter the value of r

Step 4: Calculate $d = 3 - 2r$

Step 5: Initialize $x = 0$, $y = r$

Step 6: check if the whole circle is scan Converted
If $x >= y$
Stop.

Step 7: plot eight points by using concepts of eight-way Symmetry. The center is at (p, q) current active pixel is (x, y)
putpixel (x+p, y+q)
putpixel (y+p, x+pq)

putpixel (-y+p, x+q)
putpixel (-x+p, y+q)
putpixel (-x+p, -y+q)
putpixel (-y+p, -x+q)
putpixel (y+p, -x+q)
putpixel (x+p, -y-q)

Step 8: Find location of next pixels to be scanned

If $d < 0$
then $d = d + 4x + 6$
increment $x = x + 1$
If $d \geq 0$
then $d = d + 4(x-y) + 10$
increment $x = x + 1$
decrement $y = y - 1$

Step 9: Go to Step 6

Step 10: Stop Algorithm

- Conclusion:

Bresenhams circle drawing algorithm understood and implemented successfully.

Q.

Assignment No. 4

- Implement the following pl polygon filling methods

- Theory:

A convex polygon is simple polygon (not self-intersecting) in which no line segment between two points on the boundry ever goes outside the polygon. Equivalently, it is a simple polygon whose interior is a convex set. In a convex polygon, all interior angles are less than or equal to 180 degrees, while in a strictly convex polygon all interior angles are strictly less than 180 degrees.

A simple polygon which is not convex is called concave.

Seed Filling:

In this method a particular seed points is picked and we start filling upwards and downwards pixels until boundry is reached. Seed fill method is of two types: Boundry fill and flood fill.

Boundry Fill Algorithm:

In Boundry fill Algorithm a seed point is fixed, and then neighbouring pixels are checked to match with the boundry color. Then color filling is done

until boundry is reached. A region may be 4 connected and 8 connected

Procedure for filling four connected region: color is pre specified by parameter fill color (f-color) and boundry color is specified by boun Color (b-color). getpixel() function gives the color of specified pixel and putpixel() fills the pixel with parallular color.

```
boundry_Fill(x,y,f_color,b_color)
{
If(getpixel(x,y) != b_color && getpixel(x,y) != f_co
putpixel(x,y,f_color)
boundry_Fill(x+1,y,f_color,b_color);
boundry_Fill(x,y+1,f_color,b_color);
boundry_Fill(x-1,y,f_color,b_color);
boundry_Fill(x,y-1,f_color,b_color);
  }
}
```

Flood fill algorithm:

There are some cases where the boundry color is different then the fill color by examining the colors of neighboring pixels. So instead of matching it within a boundry color a specified color is matched.

procedure for filling a 8-connected region:

```
flood_Fill (x, y, old_color, new_color)
{
putpixels (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+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)
}}
```

Disadvantages of Seed Fill Algorithm:

1) If an inside pixel is the same other color then the fill terminates and the polygon remains unfilled.
2) Seed fill method does not work for large polygons.

- Conclusion:

Successfully executed Seed fill algorithm and Boundry fill algorithm to draw a convex polygon and fill with desired.

Assignment No. 5

- Title : Implement cohen sutherland polygon clipping method to clip the polygon with respect the viewport and window. Use mouse click, keyboard interface.

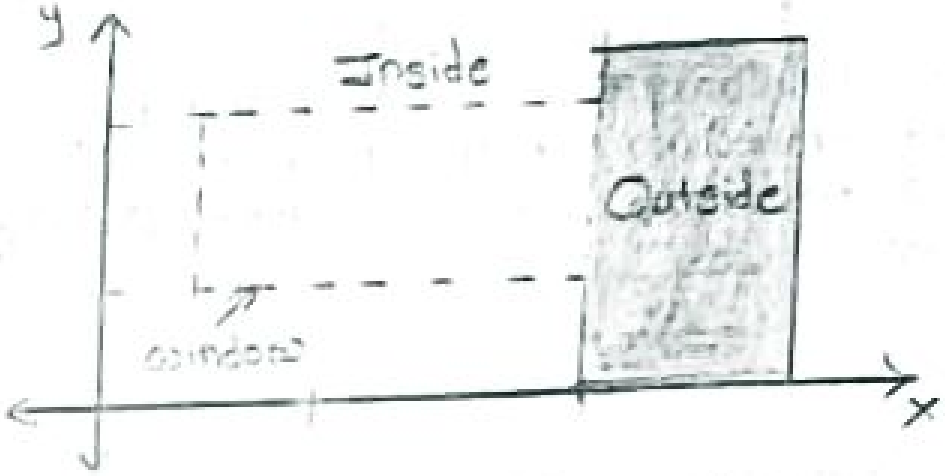
- Theory :

Cohen-Sutherland Line clipping

The cohen-sutherland line clipping to algorithm quickly detects and dispenses with two common and trivial cases. To clip a line, we need to consider only its endpoints. If both endpoints of line lie inside the window, the entire line lies inside the window It is trivially accepted and needs no clipping. On the other hand, if both endpoints of a line lie entirely to one side of the window, the line must lie entirely outside of the window. It is trivially rejected and needs to be neither clipped nor displayed.

Inside-outside Window codes.

To determine whether endpoints are inside or outside a window, the algorithm sets up a half space code for each endpoint. Each edge of the window defines an infinite the line that divides the whole space into two half spaces, the inside half-space and the outside half space, as shown below:



As you proceed around the window, extending each edge and defining an inside half-space and an outside half-space, nine regions are created - the eight "outside" regions and the one "inside" region. Each of the nine regions associated with the window is assigned a 4 bit code to identify the region. Each of the nine regions associated with the window is assigned a 4-bit code to either a 1 (true) or a 0 (false). If the region is to the left of the window, the first bit of the code is set to 1. If to the right, the third bit is set, and if to the bottom, the fourth bit is set. The 4 bit in the code then identify each of the nine regions as shown below.

1001	0001	0101
1000	0000	0100
	window	
1010	0010	0110

For any endpoint (x, y) of a line, the code can be determined that identifies which region the endpoints lies. The code's bits are set according to the following conditions:

- First bit set 1 : point lies to left of window
  $x < x_{min}$
- Second ~~point~~ bit set 1 : point lies to right to window
  $x > x_{max}$
- Third bit set 1 : Point lies below (bottom) window
  $y < y_{min}$
- Fourth setbit set 1 : Point lies above (top) window
  $y > y_{max}$

The sequence for reading the codes bits is LRBT (Left, Right, Bottom, Top)

Once the codes for each point of a line are determined, the logical AND operation of the codes determines if the line is completely outside of the window. If the logical AND of end points is not zero, the line can be trivally rejected. For example, if if an endpoint had a code of 1001 while the other endpoint had a code of 1010, the logical AND would be 1000 had codes of 1001 and 0110, the logical AND would be 0000, and the line could not be trivally rejected.

The logical OR of the endpoints codes determines if the line is completely inside the window. If the logical OR is zero, the line can be trivally accepted. For example, if the endpoints codes are 0000 and 0000, the logical OR is 0000 - the line can be trivally accepted. If the endpoints codes are 0000 and 0110, the logical OR is 0110 and the line can not be trivally accepted.

Algorithm:

The cohen-Sutherland algorithm uses a divide and conquer strategy. The line segment's endpoints are tested to see if the line can be trivally accepted or rejected, ~~If the line cannot~~ an intersection of the line with a window edge is determined and the trivial reject/accept test is repeated. This process is ~~determine~~ continue until the line is accepted.

To perform the trivial acceptance and rejection tests, we extend the edges of the window to divide the plane of the window into the nine regions. Each end point of the line segment is then assigned the code of the region in which it lies.

1) Given a line segment with endpoint $P1 = (x1, y1)$ and $P2 = (x2, y2)$

2) Compute the 4 bit codes for each endpoint

If both codes are 0000, (bitwise OR of the codes yeilds 0000) line lies completely inside the window: pass the endpoints to the drawLine.

If both codes have a 1 in the same bit position (bitwise AND of the codes is not 0000), the line lies outside the window. It can be trivially rejected.

3. If a line cannot be trivially accepted or rejected at least one of the two endpoints must lie outside the window and the line segment crosses a window edge. This line must be clipped at a window edge before being passed to the drawing routine.

4. Examine one of the endpoints, say P1 = (x1, y1). Read P1's 4 bit code in order : Left to Right, Bottom to Top.

5. When a set bit (1) is found, compute the intersection I of the corresponding window edge with the line from P1 to P2. Replace P1 with I and repeat the algorithm.

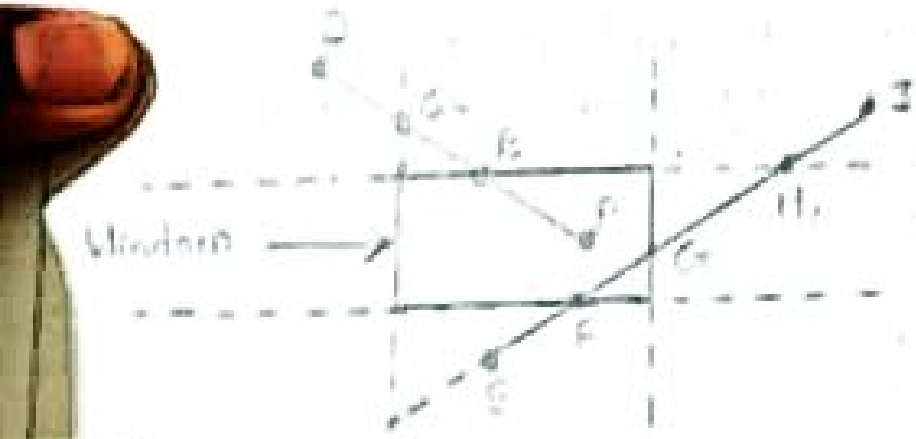
Before clipping

1. Consider the line segment AD. Point A has an outcode of 0000 and point D has an outcode of 1001. The logical AND of these outcodes is zero therefore, the line cannot trivally rejected. Also, the logical OR of the outcodes is not 'zero' therefore, the line cannot be trivally accepted. The algorithm then chooses D as the outside point (its outcode contains 1's). By our testing order, we first use the top edge to clip AD to B. The algorithm then recomputes B's

outcode as 0000. Then the next iteration of algorithm PG is tested and it is trivially accepted and display.

2. Consider the line segment PI

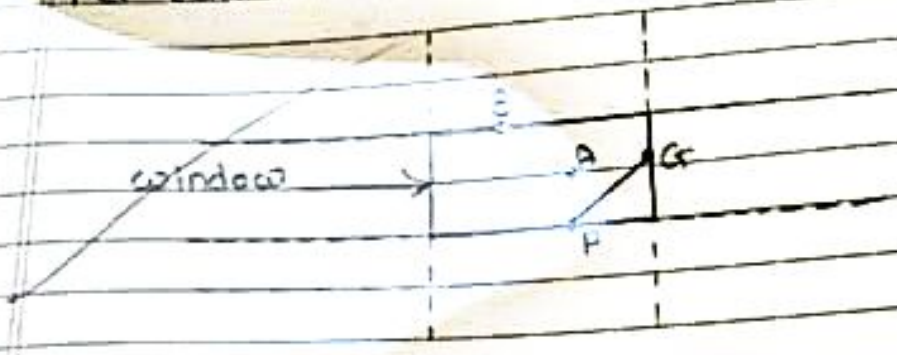
Point I has an outcode of 0100, while point P is outcode is 1010. The results of the logical tests show that the line can neither be trivally rejected or accepted. Point I is determined to be an outside point, so the algorithm clips the line against the bottom edge of the window. Now line PI has been clipped to be line PI. Line PI is tested and cannot be trivially accepted or rejected. Point I has an outcode of 0100, so the algorithm chooses pain I as an outside point. Since its outside is 1010. The line PI is clipped against the



Window's top edge, yielding a new line PH. Line PH cannot be trivally accepted or rejected. Since H's outcode is 0010, the next iteration of the algorithm clips against the window's right edge, yeilding line FG. The next iteration of the algorithm tests FG, and it is ~~being~~ trivially accepted and display.

After clipping

After clipping the segments AD and EJ, the result is that only the line segments AB and FG can be seen in the window.



- Conclusion:

Cohen Sutherland algorithm is easy to implement algorithm to clip the given line segment against rectangle window.

Assignment No. 6

- Title : Implement following 2D Transformation on the object with respect to
  a. Scaling
  b. Rotation about arbitary point
  c. Reflection

- Theory :

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.

Transformation play an important role in computer graphics to reposition the graphics on the screen and change their size or orientation.

Homogeneous Coordinates :

To perform a sequence of transformation such as translation followed by rotation and scaling, we need to follow a sequential process :

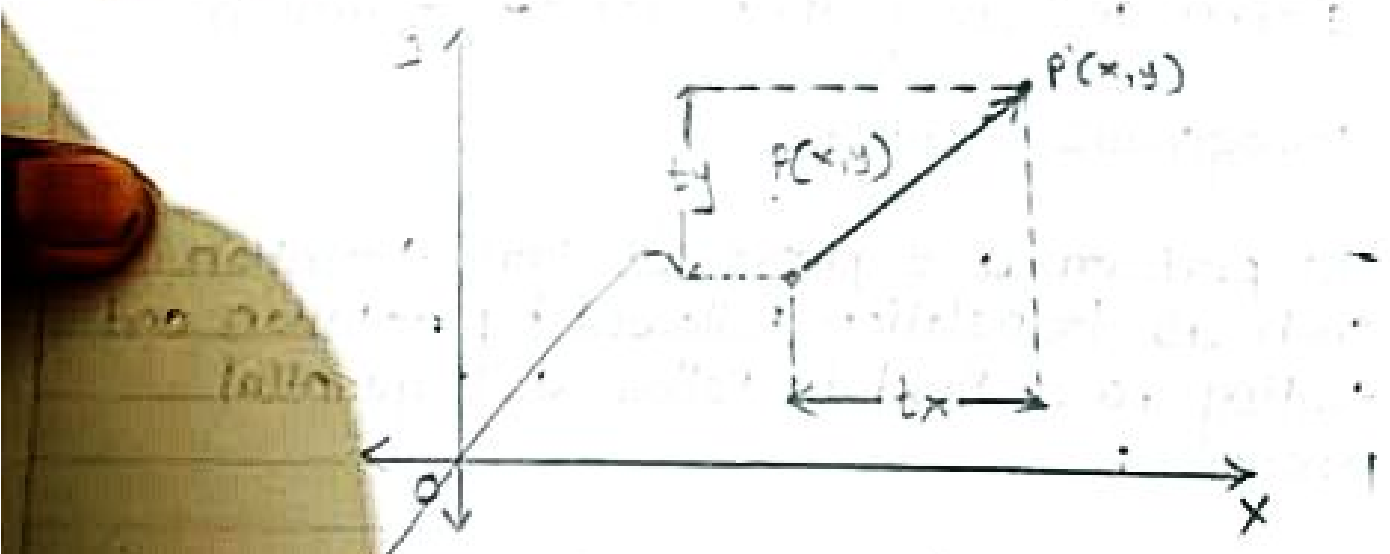
- Translate the coordinates
- Rotate the translated coordinates, then
- Scale the rotated coordinates to complete the composite transformation.

To shorten this process, we have to use 3×3 translation matrix instead of 2×2 transformation matrix. To convert a 2×2 matrix to 3×3 matrix, we have to add an extra dummy coordinate W.

In this way, we can represent the point by 3 numbers instead of 2 numbers, which is called Homogeneous Coordinate System. In this system, we can represent all transformation equations in matrix multiplication. Any cartesian P(x,y) can be converted to homogeneous coordinates by P'(xh, yh, h).

Translation:

A translation moves an object to a different position on the screen. You can translate a point in 2D by adding translation coordinate (tx, ty) to the original coordinate (x, y) to get the new coordinate (x', y').



From the above figure, you can write but:

$$x' = x + t_x$$
$$y' = y + t_y$$

The pair (t_x, t_y) is called the translation vector or shift vector. The above equations can also be represented using the column vectors.

$$P = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

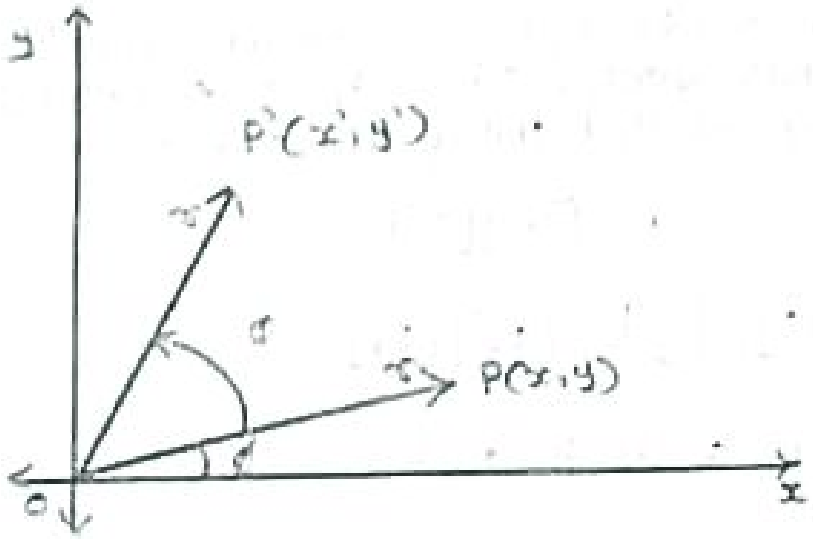
We can write it as:

$$P' = P + T$$

Rotation:

In rotation, we rotate the object at particular angle θ (theta) from its origin. From the following figure, we can see that the point $P(x, y)$ is located at angle ϕ from the horizontal X coordinate with distance r from the origin.

Let us suppose you want to rotate it at the angle θ. After rotating it to a new location, you will get a new point $P'(x', y')$



Using Standard trigonometric the original coordinate of point (x,y) can be represented as:

$$x = r\cos\phi \quad \text{---} (1)$$

$$y = r\sin\phi \quad \text{---} (2)$$

Same way we can represent the point $P'(x', y')$

$$x' = r\cos(\phi+\theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta \quad \text{---} (3)$$

$$y' = r\sin(\phi+\theta) = r\cos\phi\sin\theta + r\sin\phi\cos\theta \quad \text{---} (4)$$

Substituting equation (1) and (2) in (3) and (4):

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

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

Representing the above equation in matrix form

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

$$P' = P.R$$

where R is the rotation matrix

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

The rotation angle can be positive and negative.

For positive rotation angle, we can use the above rotation matrix. However, for negative angle rotation, the matrix will change as shown below:

$$R = \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \quad (\because \cos(-\theta) = \cos\theta \text{ and } \sin(-\theta) = -\sin\theta)$$

Scaling :

To change the size of an object, scaling transformation is used. In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result.

Let us assume that the original coordinates are (x, y), the scaling factors are (S_x, S_y) and the produced coordinates are (x', y'). This can be mathematically represented as shown below:

$$x' = x \cdot S_x \text{ and } y' = y \cdot S_y$$

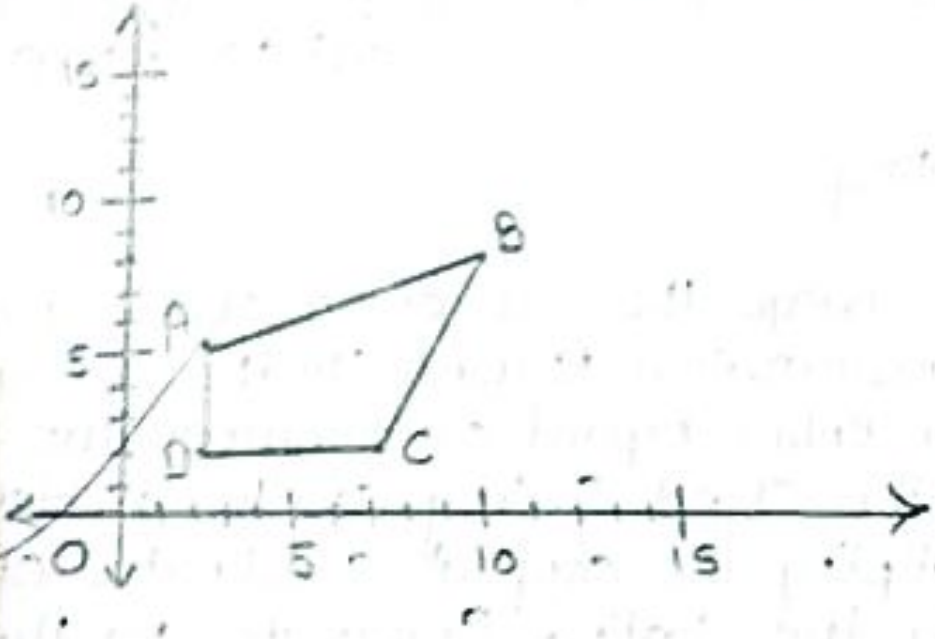
The Scaling Factor S_x, S_y scales the object in x and y direction respectively. The above equations can also be represented in matrix form as below:

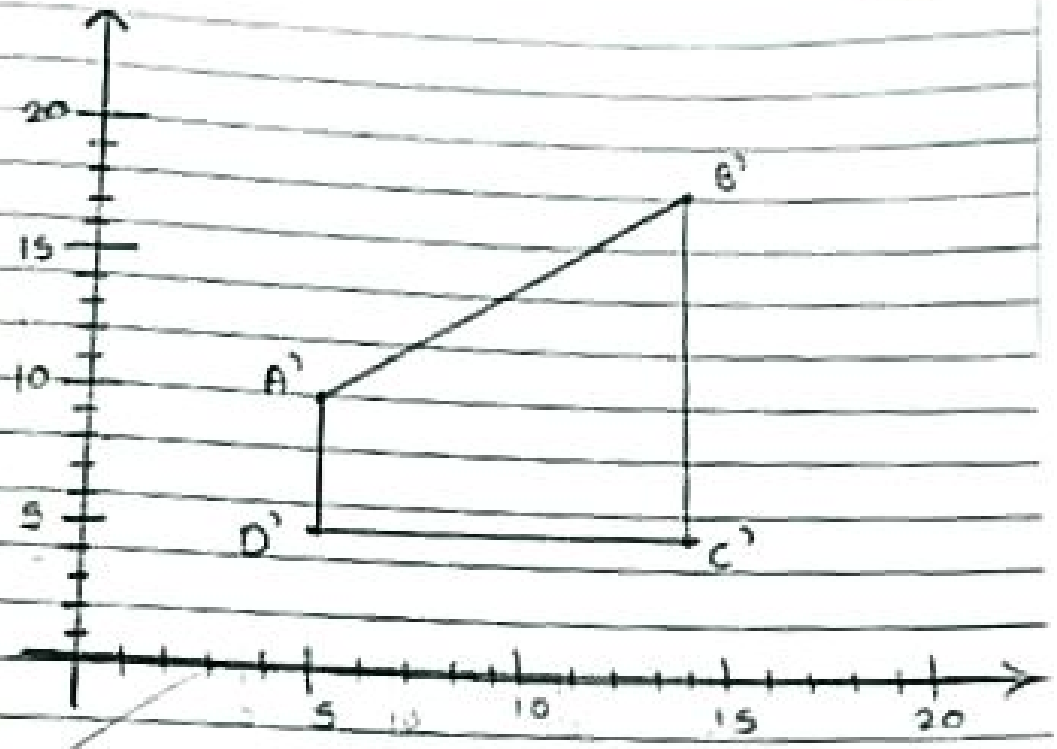
$$(x'y') = (xy)(S_x OO S_y)$$

or

$$P' = P \cdot S$$

where S is the scaling Matrix. The scaling process is shown in following figure.





Psaudo code:

Algorithm:

1. Start
2. Initialize the graphics mode
3. Construct a 2D object (use Drawpoly())
   eg (x,y)
4. A) Translation
   a. Get the translation value tx,ty
   b. Move the 2d object with tx,ty
   (x' = x + tx, y' = y + ty)
   c. Plot (x',y')
5. B) Scaling
   a. Get the scaling value Sx, Sy
   b. Resize the object with Sx, Sy (x' = x * Sx, y' = y * Sy)
   c. Plot (x',y')

6. C) Rotation

a. Get the Rotation angle

b. Rotate the object by the angle ϕ

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

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

c. Plot (x', y')

- Conduction :

We have learned the 2D transformation on different objects.

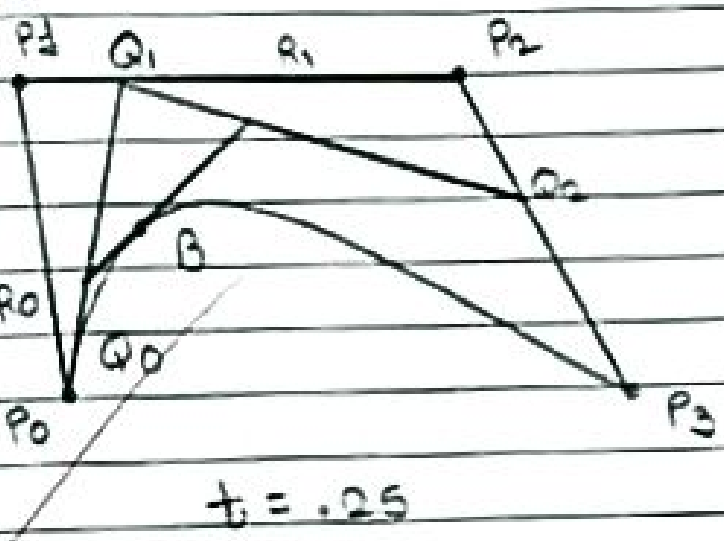
Assignment No. 1

- Title: Generate fractal patterns using
  a. Bezier Curve
  b. Koch Curve

- Theory:

Bezier Curve:

Bezier curve is a ~~math~~ mathematically defined curve used in two-dimensional graphic applications like Adobe Illustrator, Inkscape etc. The curve is defined by four points: the initial position and the terminating position i.e P0 and P3 respectively (which are called "anchors") and two separate middle points i.e P1 and P2 (which are called "handles") in our example. Bezier curves are frequently used in computer graphics, animation, modelling, etc.



Properties of bezier curves:

They always pass through the first and

last control points.

2. They are contained in the convex hull of their defining control points.

3. The degree of the polynominal defining the curve segment is one less than the number of defining polygon point. Therefore, for 4 control points, the degree of the polynominal is 3, i.e. cubic polynominal.

4. A Bezier Bezier curve generally follows the shape of the defining polygon.

5. The direction of the tanget vector at the endpoints is the same as that of the vector determined by the first and last segments.

6. Bezier curves exhibit global control means moving a control points alters the shape of the whole curve.

koch Curve:

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.

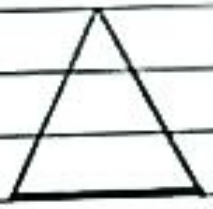
The progression for the area of the snow-flake converges to 8/5 times the area of the original triangle, while the progression for the snowflake's perimeter diverges to ~~iete~~ infinity. Consequently, the snowflake has a finite are bounded by an infinity long line.

Construction

Step 1:

Draw an equilateral triangle. You can draw it with a compass or protractor, or just eyeball it if you don't want to spend too much time drawing the snowflake

It's best if the length of the sides are divisible by 3, because of the nature of this fractal. This will become clear in the next few steps



Step 2:

Divide each side in three equal parts. This why it is handy to have the sides divisible by three

Step 3:

Draw an equilateral triangle on each middle part. Measure the length of the middle middle third to know the length of the sides of these new triangles

Step 4:

Divide each outer side into thirds. You can see the 2nd generation of triangles covers a bit of the first. These three line segments shouldn't be parted in three.

Step 5:

Draw an equilateral triangle on each middle part

- Conclusion:

We learned the Bezier and koch curve.

Assignment NO. 8.

- Title: Implement animation principles for any object

- Theory:

Definition of Animation:

Animation is the process of creating an illusion of motion and shape change by means of rapid display of various type of pictures that were made to create a single scene

Principles of Animation:

There are 12 basic principles of animation, it gives the sense of weight and volume to draw an object.

1. Squash and Sketch:
This is the most important principle of animation, it gives the sence of weight and volume to draw an object.

2. Anticipation:
In this principle animator will create a starting sence like that it shows that something will happen, almost nothing happens suddenly.

3. Staging:
Animator creates such type of seescene

like that it shows that something will happen almost nothing happen suddenly

4. Straight Ahead:
In this principle, all frames are drawn from beginning to the end and then fill all the internal or sence.

5. Flow through and overlapping action:
Two objects action have different speed in any scene can easily describe this principle.

6. Slow in slow out:
When an object have maximum acceleration in between and resist to the beginning and end will show ~~and~~ this principles working.

7. Arc:
Arcs are present in almost all animation as no object will follow straight line and follows some arc in its action.

8. Secondary action:
As with one character's action second character move shows the multiple dimension of an animation.

9. Timming
For playing a given action a perfect timing is very important.

10. Exaggeration

This principle creates extra reality in the scene by developing a proper animation style.

11. Solid drawing

In this principle, any object will created into 3D form to get realistic visualization of scene.

12. Appeal:

Any character need not be as same as any real character but it somewhat seems to be like what which create a proper thinking in the audience's mind.

Function	Description.
initgraph	It initialization the graphics system by loading the passed graphics driver then changing the system into graphic mode.
getmaxx	It returns the maximum X coordinate in current graphics mode and driver.
getmaxy	It returns the maximum X coordinate in current graphics mode and driver.
setcolor	It changes the current drawing colour. Default color is white. Each colour is assigned a number, like Black is 0 and RED is 4. Here we

	are using colour constants define inside graphics.h header file.
setfillstyle	It sets the physical current fill pattern and fill color.
circle	It draws a circle with radius r and center at (x,y)
line	It draws straight line between two points on screen.
arc	It draws a circular arc from start angle till end angle
floodfill	It is used to fill a closed area with current fill pattern and fill color. It takes any point inside a closed area and color of the boundry is input.
cleardevice	It clears the screen, and sets current position to (0,0).
delay	It is used to suspend execution of a program for n milliseconds
closegraph	It unloads the graphics drivers and sets the screen back to text mode.

- conclusion : Learned the animation or objects using different transformations.