Roll No…… Date of Performance:..... Date of Completion:.....

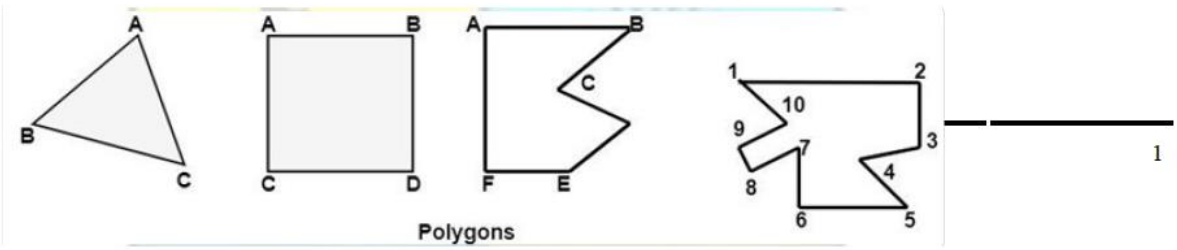
Part 2 Group A
Assignment No. 1

Title	A concave polygon filling using scan fill algorithm
Aim/Problem Statement	**Write C++ program to draw a concave polygon and fill it with desired color using scan fill algorithm. Apply the concept of inheritance.**
CO Mapped	CO3
Pre-requisite	1. Basic programming skills of C++ 2. 64-bit Open source Linux 3. Open Source C++ Programming tool like G++/GCC
Learning Objective	To understand and implement scanline polygon fill algorithm.

Theory:

1.1 **Polygon**

A polygon is a closed planar path composed of a finite number of sequential line segments. A polygon is a two-dimensional shape formed with more than three straight lines. When starting point and terminal point is same then it is called polygon.



Types of Polygons

1. Concave
2. Convex
3. Complex

● **A convex polygon** is a simple polygon whose interior is a convex set. In a convex polygon, all interior angles are less than 180 degrees.

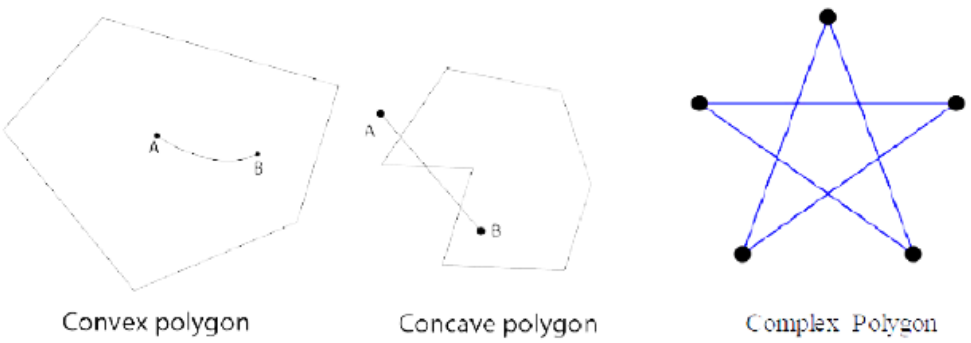
● The following properties of a simple polygon are all equivalent to convexity:

- Every internal angle is less than or equal to 180 degrees.
- Every line segment between two vertices remains inside or on the boundary of the polygon.

● **Convex Polygons**: In a convex polygon, any line segment joining any two inside points lies inside the polygon. A straight line drawn through a convex polygon crosses at most two sides.

● A concave polygon will always have an interior angle greater than 180 degrees. It is possible to cut a concave polygon into a set of convex polygons. You can draw at least one straight line through a concave polygon that crosses more than two sides.

● Complex polygon is a polygon whose sides cross over each other one or more times.

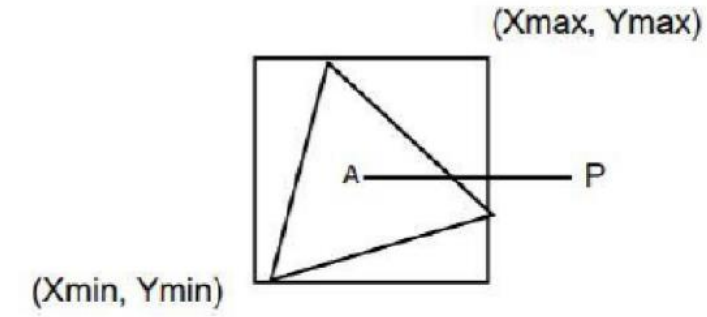


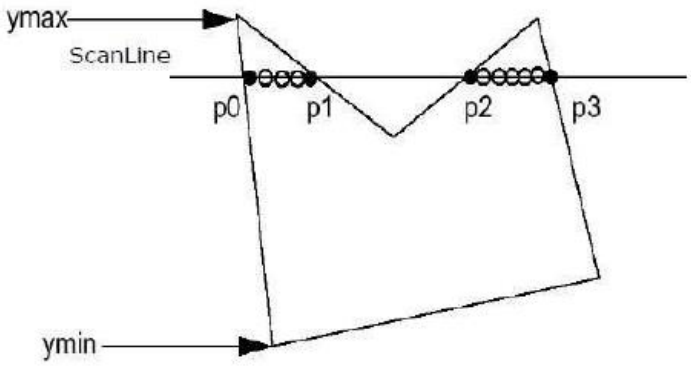
Convex polygon Concave polygon Complex Polygon

1.2 **Inside outside test (Even- Odd Test):**

● We assume that the vertex list for the polygon is already stored and proceed as follows.

● Draw any point outside the range Xmin and Xmax and Ymin and Ymax.
Draw a scan line through P up to a point A under study





● If this scan line does not pass through any of the vertices then its contribution is equal to the number of times it intersects the edges of the polygon.

- Say C if C is odd then A lies inside the polygon.
- C is even then it lies outside the polygon.
-

● If it passes through any of the vertices then the contribution of this intersection say V is,

- Taken as 2 or even. If the other points of the two edges lie on one side of the scan line.
- Taken as 1 if the other end points of the 2 edges lie on the opposite sides of the scan- line.
- Here will be total contribution is C + V.

1.3 **Polygon Filling:**
For filling polygons with particular colors, you need to determine the pixels falling on the border of the polygon and those which fall inside the polygon.

Scan fill algorithm:
A scan-line fill of a region is performed by first determining the intersection positions of the boundaries of the fill region with the screen scan lines. Then the fill colors are applied to each section of a scan line that lies within the interior of the fill region.The scanline fill algorithm identifies the same interior regions as the odd-even rule.

It is an image space algorithm. It processes one line at a time rather than one pixel at a time. It uses the concept area of coherence. This algorithm records edge list, active edge list. So accurate bookkeeping is necessary. The edge list or edge table contains the

coordinate of two endpoints. Active Edge List (AEL) contain edges a given scan line intersects during its sweep. The active edge list (AEL) should be sorted in increasing order of x. The AEL is dynamic, growing and shrinking.

Algorithm

Step1: Start algorithm

Step2: Initialize the desired data structure

1. Create a polygon table having color, edge pointers, coefficients
2. Establish edge table contains information regarding, the endpoint of edges, pointer to polygon, inverse slope.
3. Create Active edge list. This will be sorted in increasing order of x.
4. Create a flag F. It will have two values either on or off.

Step3: Perform the following steps for all scan lines

1. Enter values in Active edge list (AEL) in sorted order using y as value
2. Scan until the flag, i.e. F is on using a background color
3. When one polygon flag is on, and this is for surface S1enter color intensity as I1into refresh buffer
4. When two or image surface flag are on, sort the surfaces according to depth and use intensity value Sn for the nth surface. This surface will have least z depth value
5. Use the concept of coherence for remaining planes.

Step4: Stop Algorithm

Roll No…… Date of Performance:..... Date of Completion:.....

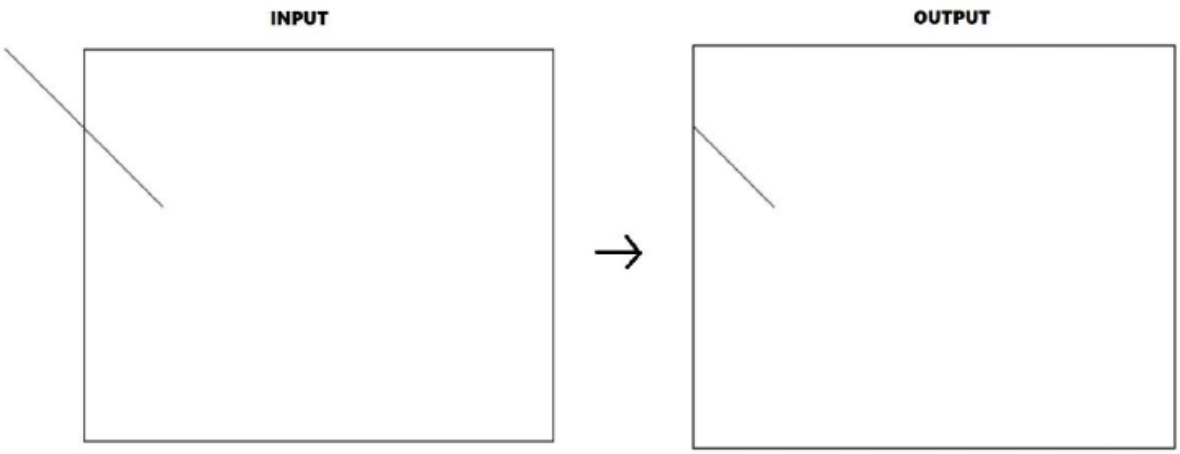
Part 2 Group A
Assignment No. 2

Title	Polygon clipping using Cohen Southerland line clipping algorithm
Aim/Problem Statement	Write C++ program to implement Cohen Southerland line clipping algorithm.
CO Mapped	CO 4
Pre-requisite	1. Basic programming skills of C++ 2. 64-bit Open source Linux 3. Open Source C++ Programming tool like G++/GCC
Learning Objective	To learn Cohen Southerland line clipping algorithm.

Theory:

Cohen Sutherland Algorithm is a **line clipping algorithm** that cuts lines to portions which are within a rectangular area. It eliminates the lines from a given set of lines and rectangle area of interest (view port) which belongs outside the area of interest and clips those lines which are partially inside the area of interest.

Example:

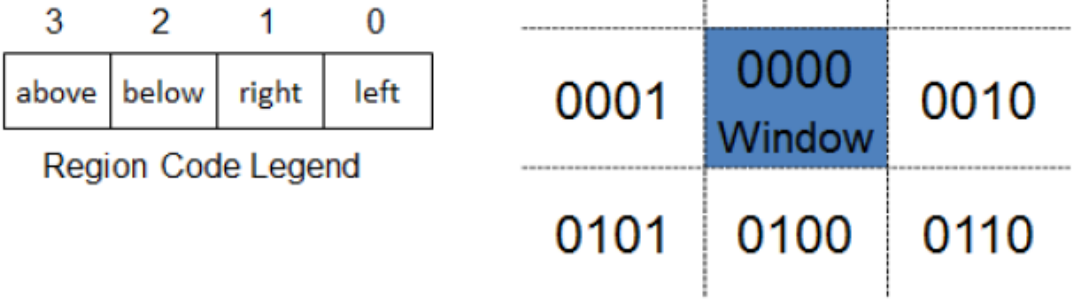


Cohen Sutherland Line Clipping Algorithm

Algorithm

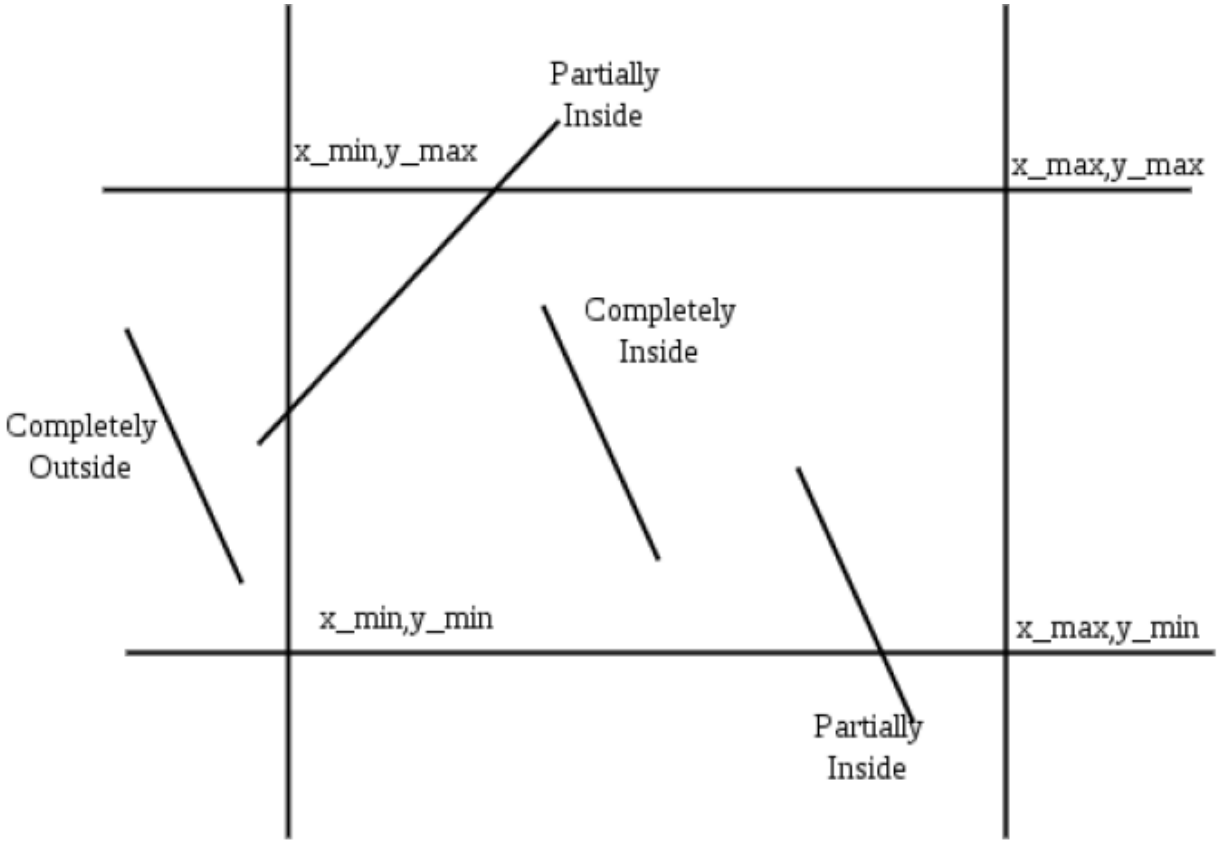
The algorithm divides **a two-dimensional space** into **9 regions** (eight outside regions and one inside region) and then efficiently determines the lines and portions of lines that are visible in the central region of interest (the viewport).

Following image illustrates the 9 regions:



As you seen each region is denoted by a 4 bit code like 0101 for the bottom right region
Four Bit code is calculated by comparing extreme end point of given line (x,y) by four co-ordinates x_min, x_max, y_max, y_min which are the coordinates of the area of interest (0000)
Calculate the four bit code as follows:

- Set First Bit if 1 Points lies to left of window (**x < x_min**)
- Set Second Bit if 1 Points lies to right of window (**x > x_max**)
- Set Third Bit if 1 Points lies to left of window (**y < y_min**)
- Set Forth Bit if 1 Points lies to right of window (y > **y_max**)



The more efficient Cohen-Sutherland Algorithm performs initial tests on a line to determine whether intersection calculations can be avoided.

Pseudocode

- **Step 1** : Assign a region code for two endpoints of given line
- **Step 2** : If both endpoints have a region code 0000 then given line is completely inside and we will keep this line.
- **Step 3:** If step 2 fails, perform the logical AND operation for both region codes.
- **Step 3.1:** If the result is not 0000, then given line is completely outside.
- **Step 3.2 :** Else line is partially inside.
- **Step 3.2.a :** Choose an endpoint of the line that is outside the given rectangle.
- **Step 3.2.b :** Find the intersection point of the rectangular boundary(based on region code)
- **Step 3.2.c :** Replace endpoint with the intersection point and upgrade the region code.
- **Step 3.2.d :** Repeat step 2 until we find a clipped line either trivially accepted or rejected.
- **Step 4:** Repeat step 1 for all lines.

Conclusion:

Roll No…… Date of Performance:..... Date of Completion:.....

Part 2 Group A
Assignment No. 3

Title	Pattern drawing using line and circle.
Aim/Problem Statement	Write C++ program to draw a given pattern. Use DDA line and Bresenham's circle drawing algorithm. Apply the concept of encapsulation.
CO Mapped	CO 3
Pre-requisite	1. Basic programming skills of C++ 2. 64-bit Open source Linux 3. Open Source C++ Programming tool like G++/GCC
Learning Objective	To learn and apply DDA line and Bresenham's circle drawing algorithm.

Theory:

DDA Line Drawing Algorithm:

Line is a basic element in graphics. To draw a line, you need two end points between which you can draw a line. Digital Differential Analyzer (DDA) line drawing algorithm is the simplest line drawing algorithm in computer graphics. It works on incremental method. It plots the points from starting point of line to end point of line by incrementing in X and Y direction in each iteration.

DDA line drawing algorithm works as follows:

Step 1: Get coordinates of both the end points (X_1, Y_1) and (X_2, Y_2) from user.
Step 2: Calculate the difference between two end points in X and Y direction.

```
dx = X2 –X1;
dy = Y2 –Y1;
```

Step 3: Based on the calculated difference in step-2, you need to identify the number of steps to put pixel. If dx > dy, then you need more steps in x coordinate; otherwise in y coordinate.

```
if (absolute(dx) > absolute(dy))
Steps = absolute(dx);
else
Steps = absolute(dy);
```

Step 4: Calculate the increment in x coordinate and y coordinate.

```
Xincrement = dx / (float) steps;
Yincrement = dy / (float) steps;
```

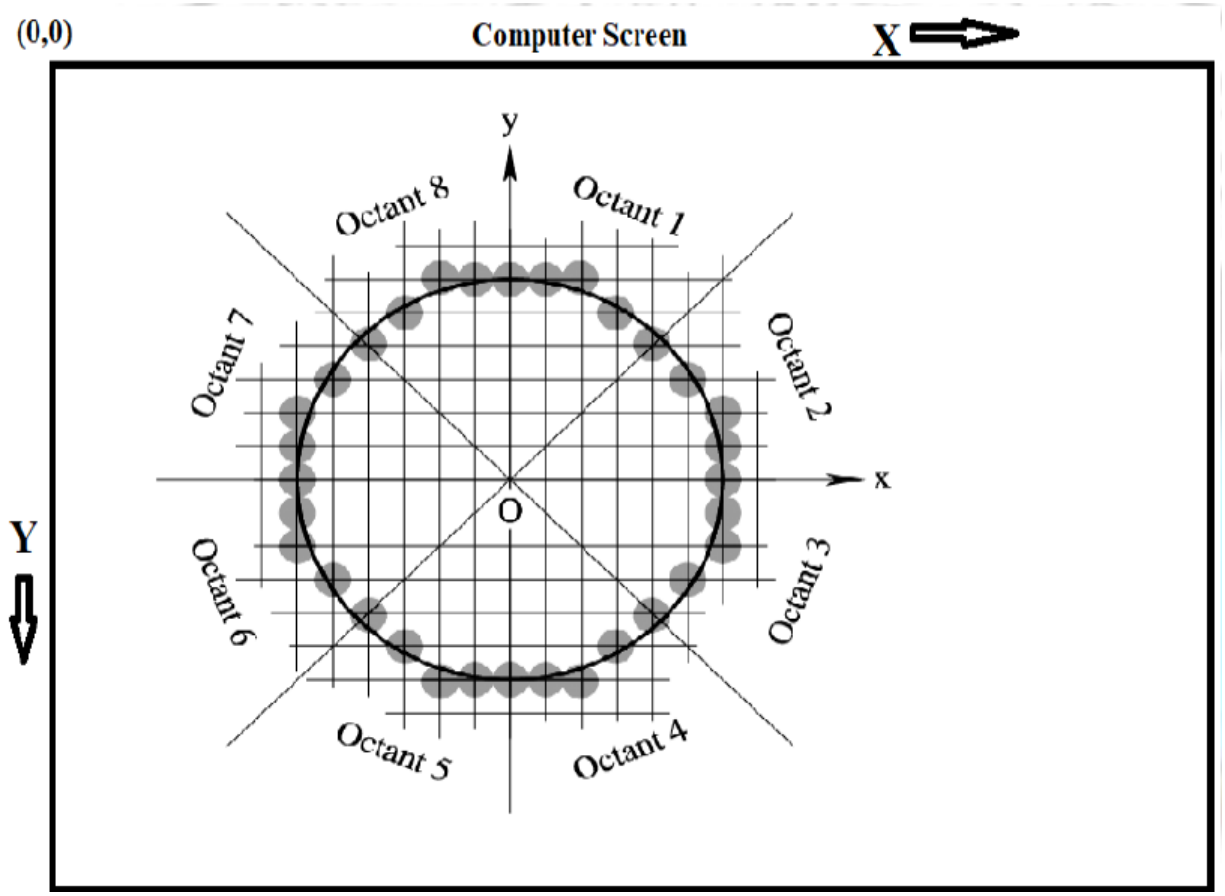
Step 5: Plot the pixels by successfully incrementing x and y coordinates accordingly and complete the drawing of the line.

```
for(int i=0; i < Steps; i++)
{
X1 = X1 + Xincrement;
Y1 = Y1 + Yincrement;
putpixel(Round(X1), Round(Y1), ColorName);
}
```

Bresenham's Circle Drawing Algorithm:

Circle is an eight-way symmetric figure. The shape of circle is the same in all quadrants. In each quadrant, there are two octants. If the calculation of the point of one octant is done, then the other seven points can be calculated easily by using the concept of eight-way symmetry.

Bresenham's Circle Drawing Algorithm is a circle drawing algorithm that selects the nearest pixel position to complete the arc. The unique part of this algorithm is that is uses only integer arithmetic which makes it significantly faster than other algorithms using floating point arithmetic.



Step 1: Read the x and y coordinates of center: (centx, centy)
Step 2: Read the radius of circle: (r)
Step 3: Initialize, x = 0; y = r;
Step 4: Initialize decision parameter: p = 3 – (2 * r)

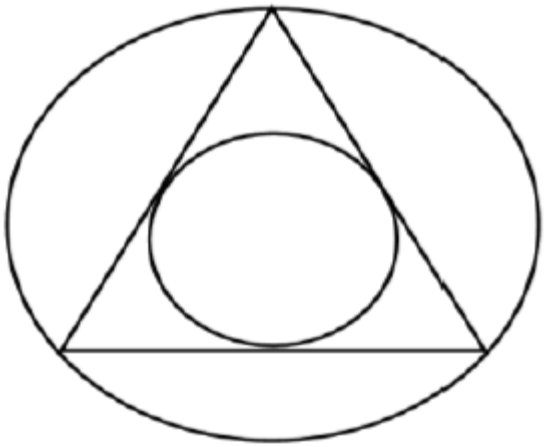
Step 5:

```
do {
putpixel(centx+x, centy-y, ColorName);
if (p<0)
p = p+(4*x)+6;
else
{
p=p+[4*(x-y)]+10;
y=y-1;
}
x=x+1;
} while(x<y)
```

Here in step 5, putpixel() function is used which will print Octant-1 of the circle with radius r after the completion of all the iterations of do-while loop. Because of the eight-way symmetry property of circle, we can draw other octants of the circle using following putpixel() functions:

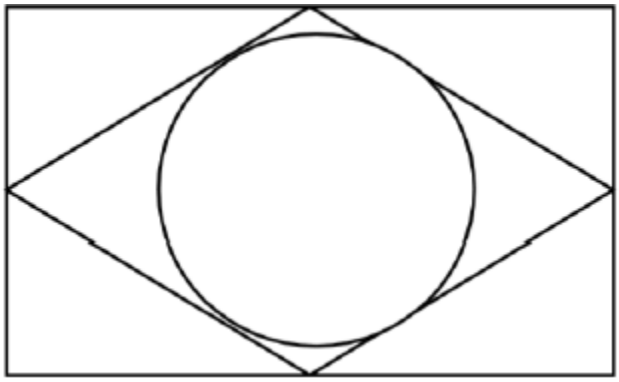
Octant 1: putpixel(centx+x, centy-y, ColorName);
Octant 2: putpixel(centx+y, centy-x, ColorName);
Octant 3: putpixel(centx+y, centy+x, ColorName);
Octant 4: putpixel(centx+x, centy+y, ColorName);
Octant 5: putpixel(centx-x, centy+y, ColorName);
Octant 6: putpixel(centx-y, centy+x, ColorName);
Octant 7: putpixel(centx-y, centy-x, ColorName);
Octant 8: putpixel(centx-x, centy-y, ColorName);

Drawing Pattern using lines and circles:



This pattern is made up of one equilateral triangle and two concentric circles. To draw the triangle, we require coordinates of 3 vertices forming an equilateral triangle. To draw two concentric circles, we require coordinates of common center and radius of both the circles.
We will take coordinates of circle and radius of one of the circle from user. Then using the properties of an equilateral triangle, we can find all 3 vertices of equilateral triangle and radius of other circle. Once we get all these parameters, we can call DDA line drawing and Bresenham's circle drawing algorithms by passing appropriate parameters to get required pattern.

OR



To draw this pattern, we require two rectangles and one circle. We can use suitable geometry to get coordinates to draw rectangles and circle. Then we can call DDA line drawing and Bresenham's circle drawing algorithms by passing appropriate parameters to get required pattern.

Conclusion:

Roll No…… Date of Performance:..... Date of Completion:.....

Part 2 Group B
Assignment No. 4

Title	Basic 2-D Transformations.
Aim/Problem Statement	a) Write C++ program to draw 2-D object and perform following basic transformations: Scaling, Translation, Rotation. Apply the concept of operator overloading. **OR** b) Write C++ program to implement translation, rotation and scaling transformations on equilateral triangle and rhombus. Apply the concept of operator overloading.
CO Mapped	CO 4
Pre-requisite	1. Basic programming skills of C++ 2. 64-bit Open source Linux 3. Open Source C++ Programming tool like G++/GCC
Learning Objective	To learn and apply basic transformations on 2-D objects.

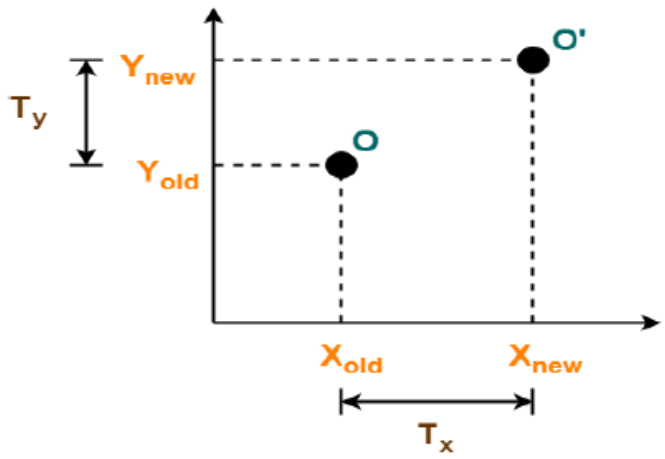
Theory:

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, reflection etc. When a transformation takes place on a 2D plane, it is called 2D transformation. Transformations play an important role in computer graphics to reposition the graphics on the screen and change their size or orientation. Translation, Scaling and Rotation are basic transformations.

1) Translation:

A translation moves an object to a different position on the screen. You can translate a point in 2D by adding translation coordinate or translation vector (T_x, T_y) to the original coordinates. Consider

- Initial coordinates of the object O = (X_{old}, Y_{old})
- New coordinates of the object O after translation = (X_{new}, Y_{new})
- Translation vector or Shift vector = (T_x, T_y)



This translation is achieved by adding the translation coordinates to the old coordinates of the object as-

$X_{new} = X_{old} + T_x$ (This denotes translation towards X axis)

$Y_{new} = Y_{old} + T_y$ (This denotes translation towards Y axis)

In Matrix form, the above translation equations may be represented as-

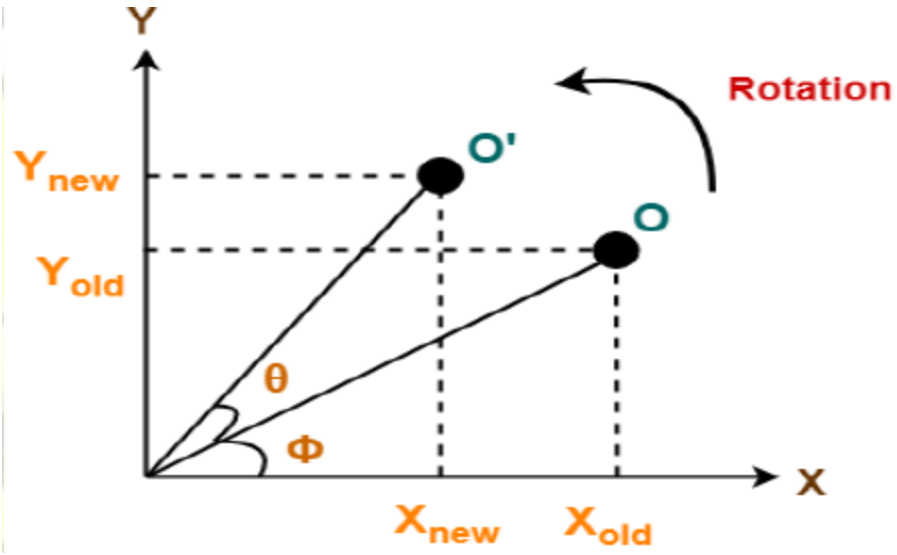
$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

Translation Matrix

2) Rotation:

In rotation, we rotate the object at particular angle θ (theta) from its original position. Consider

- Initial coordinates of the object O = (X_{old}, Y_{old})
- Initial angle of the object O with respect to origin = Φ
- Rotation angle = θ
- New coordinates of the object O after rotation = (X_{new}, Y_{new})



This anti-clockwise rotation is achieved by using the following rotation equations-

$X_{new} = X_{old} \times \cos\theta - Y_{old} \times \sin\theta$

$Y_{new} = X_{old} \times \sin\theta + Y_{old} \times \cos\theta$

In Matrix form, the above rotation equations may be represented as-

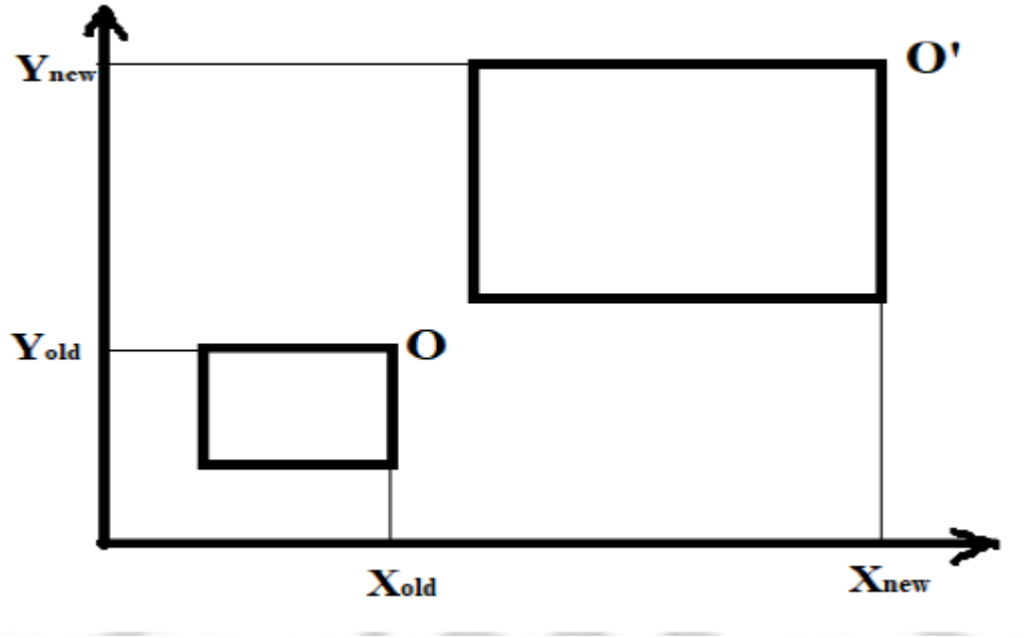
$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Rotation Matrix

3) Scaling:

Scaling transformation is used to change the size of an object. 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 (S_x, S_y). If scaling factor > 1, then the object size is increased. If scaling factor < 1, then the object size is reduced. Consider

- Initial coordinates of the object O = (X_{old}, Y_{old})
- Scaling factor for X-axis = S_x
- Scaling factor for Y-axis = S_y
- New coordinates of the object O after scaling = (X_{new}, Y_{new})



This scaling is achieved by using the following scaling equations-

$X_{new} = X_{old} \times S_x$

$Y_{new} = Y_{old} \times S_y$

In Matrix form, the above scaling equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Scaling Matrix

Homogeneous Coordinates:

Matrix multiplication is easier to implement in hardware and software as compared to matrix addition. Hence we want to replace matrix addition by multiplication while performing transformation operations. So the solution is **homogeneous coordinates**, which allows us to express all transformations (including translation) as matrix multiplications.

To obtain homogeneous coordinates we have to represent transformation matrices in 3x3 matrices instead of 2x2. For this we add dummy coordinate. Each 2 dimensional position (x,y) can be represented by homogeneous coordinate as (x,y,1).

Translation Matrix (Homogeneous Coordinates representation)

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Rotation Matrix (Homogeneous Coordinates representation)

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Scaling Matrix (Homogeneous Coordinates representation)

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} x \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Applying transformations on equilateral triangle:

Consider that coordinates of vertices of equilateral triangle are (X1,Y1), (X2,Y2) and (X3,Y3). After applying basic transformations, we will get corresponding coordinates as (X1',Y1'), (X2',Y2') and (X3',Y3') respectively. Following multiplication will give the translation on this equilateral triangle:

$$\begin{bmatrix} X1' & X2' & X3' \\ Y1' & Y2' & Y3' \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X1 & X2 & X3 \\ Y1 & Y2 & Y3 \\ 1 & 1 & 1 \end{bmatrix}$$

Similarly we can apply rotation and scaling on equilateral triangle.

Applying transformations on rhombus:

Consider that coordinates of vertices of rhombus are (X1,Y1), (X2,Y2), (X3,Y3) and (X4,Y4) applying basic transformations, we will get corresponding coordinates as (X1',Y1'), (X2',Y2'), (X3',Y3') and (X4',Y4') respectively. Following multiplication will give the translation on this rhombus:

$$\begin{bmatrix} X1' & X2' & X3' & X4' \\ Y1' & Y2' & Y3' & Y4' \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X1 & X2 & X3 & X4 \\ Y1 & Y2 & Y3 & Y4 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Similarly we can apply rotation and scaling on rhombus.

Conclusion:

Roll No…… Date of Performance:..... Date of Completion:.....

Part 2 Group B
Assignment No. 5

Title	Curves and fractals
Aim/Problem Statement	a) Write C++ program to generate snowflake using concept of fractals. **OR** b) Write C++ program to generate Hilbert curve using concept of fractals. **OR** c) Write C++ program to generate fractal patterns by using Koch curves.
CO Mapped	CO 5
Pre-requisite	1. Basic programming skills of C++ 2. 64-bit Open source Linux 3. Open Source C++ Programming tool like G++/GCC
Learning Objective	To study curves and fractals

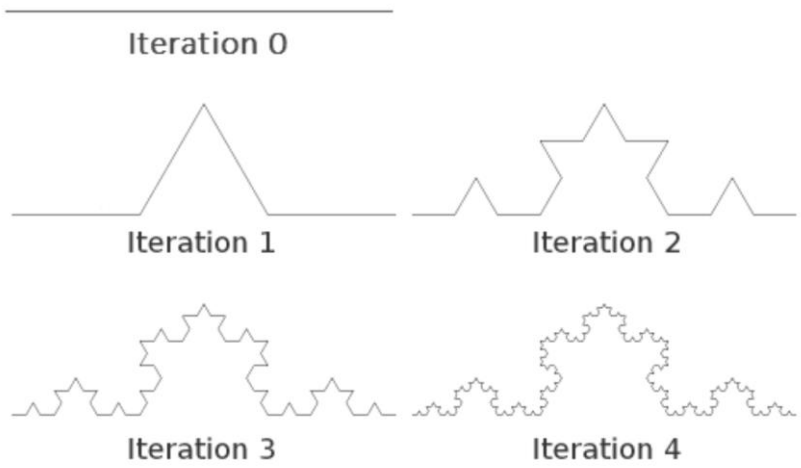
Theory:

Koch Curve:

The Koch curve fractal was first introduced in 1904 by Helge von Koch. It was one of the first fractal objects to be described. To create a Koch curve

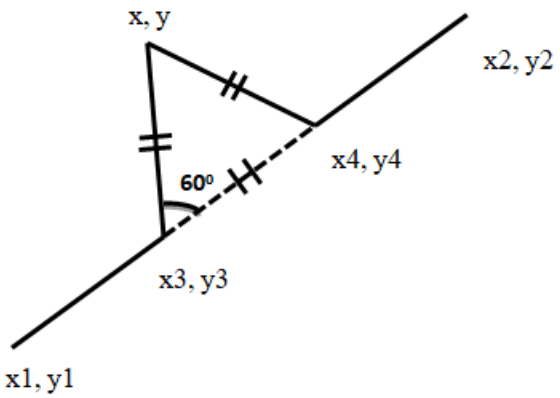
1. Create a line and divide it into three parts.
2. The second part is now rotated by 60°.
3. Add another part which goes from the end of part 2 to the beginning of part 3
4. Repeat step 1 to step 3 with each part of the line.

We will get following Koch curve as number of iteration goes on increasing



Step 1: In Iteration 0, we have a horizontal line.

Step 2: In Iteration 1, line is divided into 3 equal parts. Middle part of a line is rotated in 60_0, because it forms a perfect an equilateral triangle as shown below:



Here, (x1,y1) and (x2, y2) is accepted from user.
Now, we can see line is divided into 3 equal segments: segment((x1,y1),(x3,y3)), segment((x3,y3),(x4,y4)),segment((x4,y4),(x2,y2)) in above figure.
Coordinates of middle two points will be calculated as follows:

```
x3= (2*x1+x2)/3;
y3= (2*y1+y2)/3;
x4= (x1+2*x2)/3;
y4= (y1+2*y2)/3;
```

In our curve, middle segment((x3,y3),(x4,y4)) will not be drawn. Now, in order to find out coordinates of the top vertex (x,y) of equilateral triangle, we have rotate point (x4,y4) with respect to arbitrary point (x3,y3) by angle of 60 degree in anticlockwise direction. After performing this rotation, we will get rotated coordinates (x, y) as:

$x = x_3 + (x_4 - x_3) * \cos\theta + (y_4 - y_3) * \sin\theta \; y = y_3 - (x_4 - x_3) * \sin\theta + (y_4 - y_3) * \cos\theta$

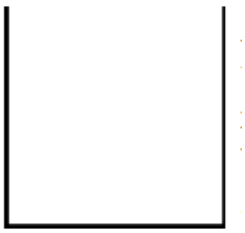
Step 3: In iteration 2, you will repeat step 2 for every segment obtained in iteration1.
In this way, you can generate Koch curve for any number of iterations.

The Hilbert curve

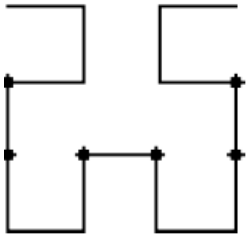
The Hilbert curve is a space filling curve that visits every point in a square grid with a size of 2×2, 4×4, 8×8, 16×16, or any other power of 2. It was first described by David Hilbert in 1892. Applications of the Hilbert curve are in image processing: especially image compression and dithering. It has advantages in those operations where the coherence between neighbouring pixels is important. The Hilbert curve is also a special version of a quadtree; any image processing function that benefits from the use of quadtrees may also use a Hilbert curve.

Cups and joins

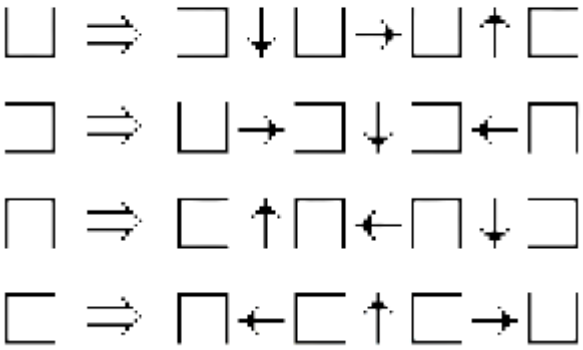
The basic elements of the Hilbert curves are what I call "cups" (a square with one open side) and "joins" (a vector that joins two cups). The "open" side of a cup can be top, bottom, left or right. In addition, every cup has two end-points, and each of these can be the "entry" point or the "exit" point.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



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



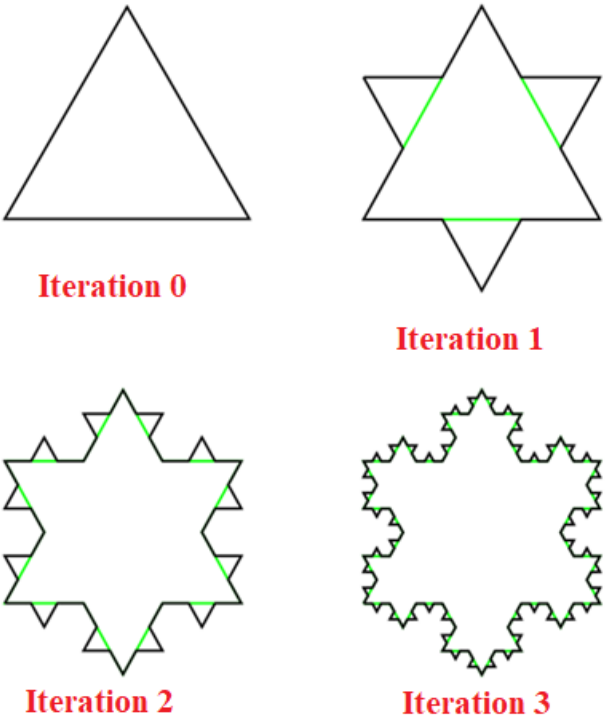
Cup subdivision rules



The function presented below (in the "C" language) computes the Hilbert curve. Note that the curve is symmetrical around the vertical axis. It would therefore be sufficient to draw half of the Hilbert curve.

Snowflake curve:

Snowflake curve is drawn using koch curve iterations. In koch curve, we just have a single line in the starting iteration and in snowflake curve, we have an equilateral triangle. Draw an equilateral triangle and repeat the steps of Koch curve generation for all three segments of an equilateral triangle.



Conclusion:

Roll No…… Date of Performance:..... Date of Completion:.....

Part 2 Group C
Assignment No. 6

Implementation of OpenGL functions	
Aim/Problem Statement	a) Design and simulate any data structure like stack or queue visualization using graphics. Simulation should include all operations performed on designed data structure. Implement the same using OpenGL. **OR** b) Write C++ program to draw 3-D cube and perform following transformations on it using OpenGL i) Scaling ii) Translation iii) Rotation about an axis (X/Y/Z). **OR** c) Write OpenGL program to draw Sun Rise and Sunset.
CO Mapped	CO 4, Co 5
Pre-requisite	1. Basic programming skills of C++ and OpenGL 2. 64-bit Open source Linux 3. Open Source C++ Programming tool like G++/GCC, OpenGL
Learning Objective	To implement OpenGL functions.

Theory:

OpenGL Basics:

Open Graphics Library (OpenGL) is a cross-language (language independent), cross-platform (platform independent) API for rendering 2D and 3D Vector Graphics (use of polygons to represent image). OpenGL is a low-level, widely supported modeling and rendering software package, available across all platforms. It can be used in a range of graphics applications, such as games, CAD design, or modeling. OpenGL API is designed mostly in hardware.

 Design: This API is defined as a set of functions which may be called by the client program. Although functions are similar to those of C language but it is language independent.

 Development: It is an evolving API and Khronos Group regularly releases its new version having some extended feature compare to previous one. GPU vendors may also provide some additional functionality in the form of extension.

 Associated Libraries: The earliest version is released with a companion library called OpenGL utility library. But since OpenGL is quite a complex process. So in order to make it easier other library such as OpenGL Utility Toolkit is added which is later superseded by freeglut. Later included library were GLEE, GLEW and glbinding.

 Implementation: Mesa 3D is an open source implementation of OpenGL. It can do pure software rendering and it may also use hardware acceleration on BSD, Linux, and other platforms by taking advantage of Direct Rendering Infrastructure.

Installation of OpenGL on Ubuntu

We need the following sets of libraries in programming OpenGL:

1. **Core OpenGL (GL):** consists of hundreds of functions, which begin with a prefix "gl" (e.g., 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 and more building models (such as qradric surfaces). GLU functions start with a prefix "glu" (e.g., gluLookAt, gluPerspective)

OpenGL Utilities Toolkit (GLUT): provides support to interact with the Operating System (such as creating a window, handling key and mouse inputs); and more building models (such as sphere and torus). GLUT functions start with a prefix of "glut" (e.g., glutCreatewindow, glutMouseFunc). GLUT is designed for constructing small to medium sized OpenGL programs. While GLUT is well-suited to learning OpenGL and developing simple OpenGL applications, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native window 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 platform.

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

Prerequisites for OpenGL
Since OpenGL is a graphics API and not a platform of its own, it requires a language to operate in and the language of choice is C++.

Getting started with OpenGL
Overview of an OpenGL program
 Main
- Open window and configure frame buffer (using GLUT for example)
- Initialize GL states and display (Double buffer, color mode, etc.)

 Loop
- Check for events

if window event (resize, unhide, maximize etc.)
modify the viewport
and Redraw
else if input event (keyboard and mouse etc.)
handle the event (such as move the camera or change the state)
and usually draw the scene

- Redraw
    - Clear the screen (and buffers e.g., z-buffer)
    - Change states (if desired)
    - Render
    - Swap buffers (if double buffer)

OpenGL order of operations

- Construct shapes (geometric descriptions of objects – vertices, edges, polygons etc.)
- Use OpenGL to
    - Arrange shape in 3D (using transformations)
    - Select your vantage point (and perhaps lights)
    - Calculate color and texture properties of each object
    - Convert shapes into pixels on screen

OpenGL Syntax

- All functions have the form: gl*
    - *glVertex3f()* – 3 means that this function take three arguments, and f means that the type of those arguments is float.
    - *glVertex2i()* – 2 means that this function take two arguments, and i means that the type of those arguments is integer
- All variable types have the form: GL*
    - In OpenGL program it is better to use OpenGL variable types (portability)
        - *Glfloat* instead of *float*
        - *Glint* instead of *int*

OpenGL primitives

Drawing two lines

```
glBegin(GL_LINES);
glVertex3f(_,_,_); // start point of line 1
glVertex3f(_,_,_); // end point of line 1
glVertex3f(_,_,_); // start point of line 2
glVertex3f(_,_,_); // end point of line 2
glEnd();
```

We can replace GL_*LINES* with *GL_POINTS, GL_LINELOOP, GL_POLYGON* etc.

OpenGL states

- On/off (e.g., depth buffer test)
    - `glEnable( GLenum )`
    - `glDisable( GLenum )`
    -
- Examples:
    - `glEnable(GL_DEPTH_TEST);`
    - `glDisable(GL_LIGHTING);`

- Mode States
  - Once the mode is set the effect stays until reset
  - Examples:
    - *glShadeModel(GL_FLAT)* or *glShadeModel(GL_SMOOTH)*
    - *glLightModel(…)* etc.

Drawing in 3D

- Depth buffer (or z-buffer) allows scene to remove hidden surfaces. Use *glEnable(GL_DEPTH_TEST)* to enable it.
  - `glPolygonMode( Face, Mode )`

- Face: `GL_FRONT, GL_BACK, GL_FRONT_AND_BACK`Mode: `GL_LINE, GL_POINT, GL_FILL`
  - `glCullFace( Mode )`

- Mode: `GL_FRONT, GL_BACK, GL_FRONT_AND_BACK`
  - `glFrontFace( Vertex_Ordering )`
  - Vertex Ordering: `GL_CW or GL_CCW`

Viewing transformation

- glMatrixMode (Mode)
  - Mode: `GL_MODELVIEW, GL_PROJECTION,` or `GL_TEXTURE`
- `glLoadIdentity()`
- `glTranslate3f(x,y,z)`
- `glRotate3f(angle,x,y,z)`
- `glScale3f(x,y,z)`

OpenGL provides a consistent interface to the underlying graphics hardware. This abstraction allows a single program to run a different graphics hardware easily. A program written with OpenGL can even be run in software (slowly) on machines with no graphics acceleration. OpenGL function names always begin with *gl*, such as *glClear()*,and they may end with characters that indicate the types of the parameters, for example *glColor3f(GLfloat red, GLfloat green, GLfloat blue)* takes three floating-point color parameters and *glColor4dv(const GLdouble *v)*takes a pointer to an array that contains 4 double-precision floating-point values. OpenGL constants begin with *GL*, such as *GL DEPTH*. OpenGL also uses special names for types that are passed to its functions, such as *GLfloat*or *GLint*, the corresponding C types are compatible, that is *float*and *int*respectively.
GLU is the OpenGL utility library. It contains useful functions at a higher level than those provided by OpenGL, for example, to draw complex shapes or set up cameras. All GLU functions are written on top of OpenGL. Like OpenGL, GLU function names begin with glu,and constants begin with GLU.
GLUT, the OpenGL Utility Toolkit, provides a system for setting up callbacks for interacting with the user and functions for dealing with the windowing system. This abstraction allows a program to run on different operating systems with only a recompile. Glut follows the convention of prepending function names with glut and constants with GLUT.

Writing an OpenGL Program with GLUT

An OpenGL program using the three libraries listed above must include the appropriate headers. This requires the following three lines:
#include<GL/gl.h>
#include <GL/glu.h>

#include <GL/glut.h>

Before OpenGL rendering calls can be made, some initialization has to be done. With GLUT, this consists of initializing the GLUT library, initializingthe display mode, creating the window, and setting up callback functions. The following lines initialize a full color, double buffered display:

glutInit(&argc, argv);

glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);

Double buffering means that there are two buffers, a front buffer and a back buffer. The front buffer is displayed to the user, while the back buffer is used for rendering operations. This prevents flickering that would occur if we rendered directly to the front buffer.

Next, a window is created withGLUT that will contain the viewport which displays the OpenGL front buffer with the following three lines:

glutInitWindowPosition(px, py);

glutInitWindowSize(sx, sy);

glutCreateWindow(name);

To register callback functions, we simply pass the name of the function that handles the event to the appropriate GLUT function.

glutReshapeFunc(reshape);

glutDisplayFunc(display);

Here, the functions should have the following prototypes:
void reshape(int width, int height);
void display();
In this example, when the user resizes the window, reshape is called by GLUT, and when the display needs to be refreshed, the display function is called. For animation, an idle event handler that takes no arguments can be created to call the display function to constantly redraw the scene with *glutIdleFunc*. Once all the callbacks have been set up, a call to *glutMainLoop*allows the program to run.
In the display function, typically the image buffer is cleared, primitives are rendered to it, and the results are presented to the user. The following line clears the image buffer, setting each pixel color to the clear color, which can be configured to be any color:
glClear(GL_COLOR_BUFFER_BIT);
The next line sets the current rendering color to blue. OpenGL behaves like a state machine, so certain state such as the rendering color is saved by OpenGL and used automatically later as it is needed.
glColor3f(0.0f, 0.0f, 1.0f);
To render a primitive, such as a point, line, or polygon, OpenGL requires that a call to *glBegin*is made to specify the type of primitive being rendered.
glBegin(GL_LINES);
Only a subset of OpenGL commands isavailable after a call to *glBegin*. The main command that is used is *glVertex*, which specifies a vertex position. In GL LINES mode, each pair of vertices define endpoints of a line segment. In this case, a line would be drawn from the point at (x0, y0) to (x1, y1).
glVertex2f(x0, y0); glVertex2f(x1, y1);
A call to glEnd completes rendering of the current primitive. *glEnd()*; Finally, the back buffer needsto be swapped to the front buffer that the user will see, which GLUT can handle for us:
glutSwapBuffers();
Developer-Driven Advantages
 Industry standard

An independent consortium, the OpenGL Architecture Review Board, guides the OpenGL specification. With broad industry support, OpenGL is the only truly open, vendor-neutral, multiplatform graphics standard.
 Stable OpenGL implementations have been available for more than seven years on a wide variety of platforms. Additions to the specification are well controlled, and proposed updates are announced in time for developers to adopt changes. Backward compatibility requirements ensure that existing applications do not become obsolete.

 Reliable and portable

All OpenGL applications produce consistent visual display results on any OpenGL API-compliant hardware, regardless of operating system or windowing system.
 Evolving Because of its thorough and forward-looking design, OpenGL allows new hardware innovations to be accessible through the API via the OpenGL extension mechanism. In this way, innovations appear in the API in a timely fashion, letting application developers and hardware vendors incorporate new features into their normal product release cycles.

 Scalable OpenGL API-based applications can run on systems ranging from consumer electronics to PCs, workstations, and supercomputers. As a result, applications can scale to any class of machine that the developer chooses to target.

 Easy to use

OpenGL is well structured with an intuitive design and logical commands. Efficient OpenGL routines typically result in applications with fewer lines of code than those that make up programs generated using other graphics libraries or packages. In addition, OpenGL drivers encapsulate information about the underlying hardware, freeing the application developer from having to design for specific hardware features.
 Well-documented:

Numerous books have been published about OpenGL, and a great deal of sample code is readily available, making information about OpenGL inexpensive and easy to obtain.

Conclusion:

Roll No…… Date of Performance:..... Date of Completion:.....

Part 2 Group C
Assignment No. 7

Title	Animation using C++
Aim/Problem Statement	a) Write a C++ program to control a ball using arrow keys. Apply the concept of polymorphism. **OR** Write a C++ program to implement bouncing ball using sine wave form. Apply the concept of polymorphism. **OR** Write C++ program to draw man walking in the rain with an umbrella. Apply the concept of polymorphism. **OR** Write a C++ program to implement the game of 8 puzzle. Apply the concept of polymorphism. **OR** Write a C++ program to implement the game Tic Tac Toe. Apply the concept of polymorphism.
CO Mapped	CO 5
Pre-requisite	1. Basic programming skills of C++ 2. 64-bit Open source Linux 3. Open Source C++ Programming tool like G++/GCC
Learning Objective	To learn scanline polygon fill algorithm.

Theory:

What is animation?

Animation is the process of designing, drawing, making layouts and preparation of photographic sequences which are integrated in the multimedia and gaming products. Animation involves the exploitation and management of still images to generate the illusion of movement.

How to move an element to left, right, up and down using arrow keys?

To detect which arrow key is pressed, you can use ncurses.h header file. Arrow key's code is defined as: KEY_UP, KEY_DOWN, KEY_LEFT, KEY_RIGHT.

```
#include<ncurses.h>

 int main()

{ int ch;

/* Curses Initialisations */

 initscr();

raw();

keypad(stdscr, TRUE);
```

```
noecho();


printw("Welcome - Press # to Exit\n");
while((ch = getch()) != '#')
{
switch(ch)
{
case KEY_UP: printw("\nUp Arrow");
break;
case KEY_DOWN: printw("\nDown Arrow");
break;
case KEY_LEFT: printw("\nLeft Arrow");
 break;
case KEY_RIGHT: printw("\nRight Arrow");
break;
 default:
{
printw("\nThe pressed key is ");
attron(A_BOLD);
printw("%c", ch);
attroff(A_BOLD);
}
}
 }
 printw("\n\nBye Now!\n");
refresh();
getch();
endwin();
return 0;
}
```

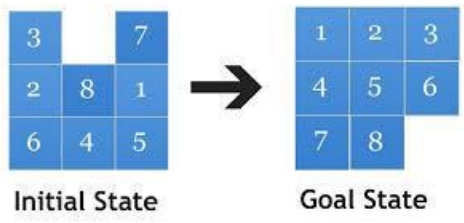
How to draw a sine wave using c++?

```
#include <math.h>
#include <graphics.h>
#include <iostream>
int main() {
int gd = DETECT, gm;
int angle = 0;
double x, y;
initgraph(&gd, &gm, NULL);
line(0, getmaxy() / 2, getmaxx(), getmaxy() / 2);
/* generate a sine wave */
for(x = 0; x < getmaxx(); x+=3) {
/* calculate y value given x */
y = 50*sin(angle*3.141/180);
y = getmaxy()/2 - y;
/* color a pixel at the given position */
putpixel(x, y, 15);
delay(100);
/* increment angle */
angle+=5;
}

/* deallocate memory allocated for graphics screen */
closegraph();
return 0;
}
```

A game of 8 puzzle:

An 8 puzzle is a simple game consisting of a 3 x 3 grid (containing 9 squares). One of the squares is empty. The object is to move to squares around into different positions and having the numbers displayed in the "goal state". The image to the left can be thought of as an unsolved initial state of the "3 x 3" 8 puzzle. Eight digits will in random order. To solve a puzzle, you have to move blocks by performing translation of blocks.



Implementation of Tic-Tac-Toe game

Rules of the Game

- The game is to be played between two people (in this program between HUMAN and COMPUTER).
- One of the player chooses ‘O’ and the other ‘X’ to mark their respective cells.
- The game starts with one of the players and the game ends when one of the players has one whole row/ column/ diagonal filled with his/her respective character (‘O’ or ‘X’).
- If no one wins, then the game is said to be a draw.

O	X	O
O	X	X
X	O	X

Note: In all above programs, you have to perform translation of an image to show animation effect.

Conclusion: