Group A

Assignment No A1.

Write C++/Java program to draw line using DDA and Bresenham’s algorithm. Inherit pixel class and Use function overloading.

Aim:
To draw line using DDA and Bresenham’s algorithm

Software Requirements:
64-bit Open source Linux or its derivative
Open Source C++ Programming tool like G++/GCC

Objectives: Identify the Line Drawing algorithms of computer graphics

Outcomes: Implement computer graphics programs in C++ using the line drawing algorithms.

Theory Concepts/ Logic/ Algorithm:

Digital Differential Analyzer (DDA)
The slope of a straight line is given as
M = (y2-y1)/(x2-x1) i.e ∆y / ∆x
This equation can be used to obtain a rasterized straight line. For any given x interval ∆x along a line, we can compute the corresponding y interval ∆y as,

\[ \Delta y = \left\lfloor \frac{y_2 - y_1}{x_2 - x_1} \right\rfloor \Delta x \]

Similarly, we can obtain the x interval ∆x corresponding to a specified ∆y as,

\[ \Delta x = \left\lfloor \frac{(x_2 - x_1)}{(y_2 - y_1)} \right\rfloor \Delta y \]

Once the intervals are known the values for next x and next y on the straight line can be obtained as follows

\[ x_{i+1} = x_i + \Delta x \]
\[ x_i + \left\lfloor \frac{(x_2 - x_1)}{(y_2 - y_1)} \right\rfloor \Delta y \]

And

\[ y_{i+1} = y_i + \Delta y \]
\[ y_i + \left\lfloor \frac{(y_2 - y_1)}{(x_2 - x_1)} \right\rfloor \Delta x \]
these equations represents a recursion relation for successive values of x and y along the required line. Such a way of rasterizing a line is called a digital differential analyzer (DDA).

For simple DDA either $\Delta x$ or $\Delta y$, whichever is larger, is chosen as one raster unit, i.e.
If $|\Delta x| \geq |\Delta y|$ 
Then $\Delta x = 1$
Else $\Delta y = 1$

**Algorithm:**

1. Read the line end points $(x_1, y_1)$ and $(x_2, y_2)$ such that they are not equal.
   (If equal then plot that point and exit)
2. $\Delta x = |x_2 - x_1|$ and $\Delta y = |y_2 - y_1|$
3. if $(\Delta x \geq \Delta y)$ then
   length=$\Delta x$
else
   length=$\Delta y$
end if
4. $\Delta x = (x_2 - x_1) / length$
   $\Delta y = (y_2 - y_1) / length$
5. $x = x_1 + 0.5 \times \text{Sign} (\Delta x)$
   $y = y_1 + 0.5 \times \text{Sign} (\Delta y)$

(Sign function makes the algorithm work in all quadrants. It returns $-1, 0, 1$ depending on whether its argument is $<0, =0, >0$ respectively. The factor $0.5$ makes it possible to round the values in the integer function rather than truncating them.)
6. i=1 (Begins the loop, in this loop points are plotted)
   While (i<=$length$)
   {
   Plot (Integer (x), Integer (y))
   x=x+$\Delta x$
   y=y+$\Delta y$
   i=i+1
   }
7. Stop.
Bresenham’s Line

The basic principle of Bresenham’s line algorithm is to select the optimum raster locations to represent a straight line. To get this the algorithm always increments either x or y by one unit depending on the slope of line. The increment in the other variable is determined by examining the distance between the actual line location and the nearest pixel. This distance is called decision variable or the error.

In mathematical terms decision variable or the error is defined as

\[ E = D_B - D_A \]

If \( e > 0 \), then it implies that \( D_B > D_A \), i.e. the pixel above the line is closer to the true line. If \( D_B < D_A \), then we can say that the pixel below the line is closer to the true line.

The error term is initially set as

\[ e = 2\Delta y - \Delta x \]

Where \( \Delta y = y_2 - y_1 \),

\[ \Delta x = x_2 - x_1 \]

Then according to value of \( e \) following actions are taken.

While (\( e \geq 0 \))

\{
  y = y + 1
  e = e - 2\Delta x
\}

\[ x = x + 1 \]

\[ e = e + 2\Delta y \]

Algorithm:

1. Read the line end points \((x_1, y_1)\) and \((x_2, y_2)\) such that they are not equal.
   (If equal then plot that point and exit)
2. \( \Delta x = |x_2 - x_1| \) and \( \Delta y = |y_2 - y_1| \)
3. [Initialize starting point]
   \[ x = x_1 \]
   \[ y = y_1 \]
4. \( e = 2\Delta y - \Delta x \)
   [Initialize value of decision variable or error to compensate for nonzero intercepts]
4. \( i = 1 \) [Initialize counter]
5. Plot \((x, y)\)
6. While (\( e \geq 0 \))
{ 
    y=y+1
    e=e-2*Δx
}

x=x+1
e=e+2*Δy
7. I=I+1
8. If (I<=Δx) then go to step 6.
9. Stop

Flowchart for Bresenham’s Line Drawing Algorithm:

Flowchart for DDA Line Drawing Algorithm:
Assignment No A2.

Write C++/Java program to draw circle using Bresenham‘s algorithm. Inherit pixel class

Aim:
To draw Circle using Bresenham’s algorithm

Software Requirements:
64-bit Open source Linux or its derivative
Open Source C++ Programming tool like G++/GCC

Objectives: Identify the Circle Drawing algorithms of computer graphics
Outcomes: Implement computer graphics programs in C++ using the Circle drawing algorithms.

Theory Concepts/ Logic/ Algorithm:
Bresenham’s Circle Drawing Algorithm

This algorithm considers the eight-way symmetry of the circle to generate it. It plots 1/8th part of the circle, i.e. from 90° to 45°. As the circle is drawn from 90° to 45°, the x moves in positive direction and y moves in the negative direction. To achieve the best approximation to the true circle we have to select those pixels in the raster that fall the least distance from the true circle.

Here the equation for \( d_i \) at starting point, i.e. at \( x=0 \) and \( y=r \) can be simplified as,

\[
d_i = 3 - 2r \quad \text{(where r is radius)}
\]

For \( d_i < 0 \), \( x_{i+1} = x_i + 1 \) and
For \( d_i \geq 0 \), \( x_{i+1} = x_i + 1 \) and \( y_{i+1} = y_i - 1 \)

Similarly, the equations for \( d_i+1 \) for both the cases are given as
For \( d_i < 0 \), \( d_{i+1} = d_i + 4x_i+6 \) and
For \( d_i \geq 0 \), \( d_{i+1} = d_i + 4(x_i-y_i) +10. \)

Reflecting about x & y-axis can draw the remaining part of the circle.

Algorithm:

1. Read the radius(r) of the circle.
2. \( d=3-2r \) (Initialize the decision variable)
3. \( x=0, y=r \)  (Initialize starting point)
4. do
   
   \{
      plot(x,y)
      plot(y,x)
      plot(y,-x)
      plot(x,-y)
      plot(-x,-y)
      plot(-y,-x)
      plot(-y,x)
      plot(-x,y)
   
   if(d<0) then
   \{
      d=d+4x+6
   \}
   else
   \{
      d=d+4(x-y)+10
      y=y-1
   \}
   x=x+1
\}while(x<y)
5. Stop.
Flowchart for Bresenham’s Circle Drawing Algorithm:

Start

- X=0, Y=r

- D=3-2r

- Call Draw Circle(Xc,Yc,X,Y)

- X=X+1

- If X<Y

  - D<0

    - Y=Y-1

    - D=D+4(X-Y)+10

    - D=D+4X+6

    - Call Draw Circle(Xc,Yc,X,Y)

    - X=X+1

  - Else

    - Stop

Assignment No A3.
Write C++/Java program to draw 2-D object and perform following basic transformations,
   a) Scaling
   b) Translation
   c) Rotation
Use operator overloading.

Aim:
To draw 2-D object and perform basic transformations

Software Requirements:
64-bit Open source Linux or its derivative
Open Source C++ Programming tool like G++/GCC

Objectives: Identify the 2-D object transformations of computer graphics
Outcomes: Implement computer graphics programs in C++ using the basic 2-D object transformations.

Theory Concepts/ Logic/ Algorithm:

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

Translation

A translation moves an object to a different position on the screen. You can translate a point in 2D by adding translation coordinate \((t_x, t_y)\) to the original coordinate \((X, Y)\) to get the new coordinate \((X', Y')\).

From the above figure, you can write that –

\[ X' = X + t_x \]
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.

\[ P = [X]/[Y] \]
\[ p' = [X']/[Y'] \]
\[ T = [tx]/[ty] \]

We can write it as –

\[ P' = P + T \]

**Rotation**

In rotation, we rotate the object at particular angle \(\theta\) (theta) from its origin. From the following figure, we can see that the point \(P(X, Y)\) is located at angle \(\phi\) from the horizontal X coordinate with distance \(r\) from the origin. Let us suppose you want to rotate it at the angle \(\theta\). After rotating it to a new location, you will get a new point \(P' (X', Y')\).

Using standard trigonometric the original coordinate of point \(P(X, Y)\) can be represented as –

\[ X = r \cos \phi \].....(1)
\[ Y = r \sin \phi \].....(2)

Same way we can represent the point \(P' (X', Y')\) as –

\[ x' = r \cos(\phi + \theta) = r \cos \phi \cos \theta - r \sin \phi \sin \theta \].....(3)
\[ y' = r \sin(\phi + \theta) = r \cos \phi \sin \theta + r \sin \phi \cos \theta \].....(4)

Substituting equation (1) & (2) in (3) & (4) respectively, we will get

\[ x' = x \cos \theta - y \sin \theta \]
\[ y' = x \sin \theta + y \cos \theta \]
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 −

\[
\begin{align*}
X' &= X \cdot S_X \\
Y' &= Y \cdot S_Y
\end{align*}
\]

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 −

\[
\begin{pmatrix}
X' \\
Y'
\end{pmatrix} = \begin{pmatrix}
X \\
Y
\end{pmatrix} \begin{bmatrix}
S_X & 0 \\
0 & S_Y
\end{bmatrix}
\]

OR

\[
P' = P \cdot S
\]

Where \(S\) is the scaling matrix. The scaling process is shown in the following figure.
If we provide values less than 1 to the scaling factor $S$, then we can reduce the size of the object. If we provide values greater than 1, then we can increase the size of the object.

**Algorithm:**

1. Start
2. Initialize the graphics mode.
3. Construct a 2D object (use Drawpoly()) e.g. $(x,y)$
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 $\phi$
      \[ x' = x \cos \phi - y \sin \phi \]
      \[ y' = x \sin \phi + y \cos \phi \]
   c. Plot $(x', y')$
Flow chart for 2D Transformations

Start

- True
  - Translation
    - Get tx, ty
    - \( (x' = x + tx, y' = y + ty) \)
    - Plot(x, y)
- False
  - Rotat
  - False
    - Scale
      - Get Sx, Sy
      - \( x' = x \cdot Sx, y' = y \cdot Sy \)
  - True
    - Get Angle
      - \( x' = x \cdot \cos \phi - y \cdot \sin \phi \)
      - \( y' = x \cdot \sin \phi - y \cdot \cos \phi \)
  - True
    - Stop
Assignment No A4.

Write C++/Java program to fill polygon using scan line algorithm. Use mouse interfacing to draw polygon.

Aim:
To fill polygon using scan line algorithm

Software Requirements:
64-bit Open source Linux or its derivative
Open Source C++ Programming tool like G++/GCC

Objectives: Identify the scan line algorithm of computer graphics
Outcomes: Implement computer graphics programs in C++ using scan line algorithm.

Theory Concepts/ Logic/ Algorithm:
Scan-line area filling uses the property of coherence to color the polygon. Scanline coherence means if a point lies inside a polygon then other points lying on the same scanline near to it are also likely to be inside the polygon.

It works as follows:
Step 1 – Find out the Ymin and Ymax from the given polygon.

Step 2 – ScanLine intersects with each edge of the polygon from Ymin to Ymax. Name each intersection point of the polygon. As per the figure shown above, they are named as p0, p1, p2, p3.

Step 3 – Sort the intersection point in the increasing order of X coordinate i.e. (p0, p1), (p1, p2), and (p2, p3).

Step 4 – Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.

Algorithm
1. The scan conversion algorithm works as follows
   a. Intersect each scanline with all edges
   b. Sort intersections in x
c. Calculate parity of intersections to determine in/out

d. Fill the “in” pixels

2. Special cases to be handled:
   a. Horizontal edges should be excluded
   b. Vertices lying on scanlines handled by shortening of edges,

3. Coherence between scanlines tells us that
   a. Edges that intersect scanline $y$ are likely to intersect $y + 1$
   b. $X$ changes predictably from scanline $y$ to $y + 1$ (Incremental Calculation Possible)

4. The slope of the edge is constant from one scan line to the next:
   a. Let $m$ denote the slope of the edge.
   b.

   \[
   y_{k+1} - y_k = 1
   \]

   \[
   x_{k+1} = x_k + \frac{1}{m}
   \]

5. Each successive $x$ is computed by adding the inverse of the slope and rounding to the nearest integer
Flow chart for Scan Fill Algorithm

1. Start
2. Get Ymin and Ymax
3. Intersect scan line at points of all edges
4. Sort the intersection Points
5. Fill the “in” points
6. Stop
Assignment No A5.

Write C++/Java program to draw the following pattern using any Line drawing algorithms.

Aim:
To draw the pattern using any line drawing algorithm

Software Requirements:
64-bit Open source Linux or its derivative
Open Source C++ Programming tool like G++/GCC

Objectives: Identify the line drawing and circle drawing algorithms of computer graphics
Outcomes: Implement computer graphics programs in C++ using the line drawing algorithms.

Theory Concepts/ Logic/ Algorithm:

Digital Differential Analyzer (DDA) Algorithm:
1. Read the line end points \((x_1, y_1)\) and \((x_2, y_2)\) such that they are not equal.
   (If equal then plot that point and exit)
2. \(\Delta x = |x_2-x_1|\) and \(\Delta y = |y_2-y_1|\)
3. if \((\Delta x >= \Delta y)\) then
   length=\(\Delta x\)
else
   length=\(\Delta y\)
end if
4. \(\Delta x= (x_2-x_1) / length\)
   \(\Delta y= (y_2-y_1) / length\)
5. \(x=x_1+0.5 * \) Sign \((\Delta x)\)
   \(y=y_1+0.5 * \) Sign \((\Delta y)\)

(Sign function makes the algorithm work in all quadrants. It returns \(-1, 0, 1\) depending on whether its argument is \(<0, =0, >0\) respectively. The factor 0.5 makes it possible to round the values in the integer function rather than
truncating them.)

6. i=1 (Begins the loop, in this loop points are plotted)
   
   While (i<=length)
   {
     Plot (Integer (x), Integer (y))
     
     x=x+Δx
     y=y+Δy
     i=i+1
   }

7. Stop.

**Bresenham's Line Algorithm:**

1. Read the line end points (x1, y1) and (x2, y2) such that they are not equal.
   (If equal then plot that point and exit)
2. Δx = |x2-x1| and Δy = |y2-y1|
3. [Initialize starting point]
   x=x1
   y=y1
   e=2*Δy-Δx
   [Initialize value of decision variable or error to compensate for nonzero intercepts]
4. I=1 [Initialize counter]
5. Plot (x, y)
6. While (e>=0)
   {
     y=y+1
     e=e-2*Δx
   }
7. I=I+1
8. If (I<=Δx) then go to step 6.
9. Stop.
Algorithm:

1. Read the coordinate points using mouse interface.

2. Draw line using any line drawing algorithm connecting the coordinate points.

**Flow chart**

```
Start
↓
Accept coordinates
↓
Draw Line through points
↓
Stop
```
Write C++/Java program for line drawing using DDA or Bresenham's algorithm with patterns such as solid, dotted, dashed, dash dot and thick.

Aim:
To draw the different line styles using any line drawing algorithm

Software Requirements:
64-bit Open source Linux or its derivative
Open Source C++ Programming tool like G++/GCC

Objectives: Identify the line drawing algorithms of computer graphics
Outcomes: Implement computer graphics programs in C++ using the line drawing algorithms.

Theory Concepts/ Logic/ Algorithm:

Digital Differential Analyzer (DDA) Algorithm:
1. Read the line end points \((x_1, y_1)\) and \((x_2, y_2)\) such that they are not equal.
   (If equal then plot that point and exit)
2. \(\Delta x = |x_2-x_1|\) and \(\Delta y = |y_2-y_1|\)
3. if \((\Delta x >= \Delta y)\) then
   length = \(\Delta x\)
else
   length = \(\Delta y\)
end if
4. \(\Delta x = (x_2-x_1) / \text{length}\)
   \(\Delta y = (y_2-y_1) / \text{length}\)
5. \(x = x_1 + 0.5 * \text{Sign}(\Delta x)\)
   \(y = y_1 + 0.5 * \text{Sign}(\Delta y)\)

(Sign function makes the algorithm work in all quadrants. It returns \(-1, 0, 1\) depending on whether its argument is \(<0, =0, >0\) respectively. The factor 0.5 makes it possible to round the values in the integer function rather than truncating them.)
6. \(i=1\) (Begins the loop, in this loop points are plotted)
   While \((i<\text{length})\)
{ 
    Plot (Integer (x), Integer (y))
    x=x+Δx
    y=y+Δy
    i=i+1
}
7. Stop.

**Bresenham’s Line Algorithm:**
1. Read the line end points (x1, y1) and (x2, y2) such that they are not equal. (If equal then plot that point and exit)
2. Δx = |x₂-x₁| and Δy = |y₂-y₁|
3. [Initialize starting point]
    x=x₁
    y=y₁
4. e=2*Δy-Δx
   [Initialize value of decision variable or error to compensate for nonzero intercepts]
4. I=1 [Initialize counter]
5. Plot (x, y)
6. While (e>=0)
   { y=y+1
     e=e-2*Δx
   }
7. x=x+1
8. e=e+2*Δy
7. I=I+1
8. If ( I<=Δx) then go to step 6.
9. Stop.

**Line Style:**

Basic attributes of a straight line segment are its type, its width, and its color. Possible selections for the line-type attribute include solid lines, dashed lines, and dotted lines. We modify a line drawing algorithm to generate such lines by setting the length and spacing of displayed solid sections along the line path. A dashed line could be displayed by generating an inter dash spacing that is equal to the length of the solid sections. Both the length of the dashes and the inter dash spacing are often specified as user options. A dotted line can be displayed by generating very short dashes with the spacing equal to or greater than the dash size.

**Line Width:**

Implementation of line-width options depends on the capabilities of the output device. A heavy line on a monitor could be displayed as adjacent parallel lines. To draw a line between (xa ,ya ) and (xb,yb) with thickness w, we
would have a top boundary between the points (xa,ya+wy) and (xb,yb+wy) and a lower boundary between (xa,ya+wy) and (xb,yb-wy) where wy is given by

\[ w_y = \frac{(w-1) \left( \left| x_b-x_a \right| + \left( y_b-y_a \right) \right)^{1/2}}{2| x_b-x_a |} \]

**Algorithm:**

1. Accept coordinates from user.
2. Accept choice for the line pattern.
3. For dashed line draw line by generating an inter dash spacing that is equal to the length of the solid sections.
4. For dotted line by generating very short dashes with the spacing equal to or greater than the dash size.
5. For thick line between (xa,ya) and (xb,yb) with thickness w we would have a top boundary between the points (xa,ya+wy) and (xb,yb+wy) and a lower boundary between (xa,ya+wy) and (xb,yb-wy) where wy is given by

\[ w_y = \frac{(w-1) \left( \left| x_b-x_a \right| + \left( y_b-y_a \right) \right)^{1/2}}{2| x_b-x_a |} \]

6. Draw line using any line drawing algorithm.
Write C++/Java program to draw a convex polygon and fill it with desired color using Seed fill algorithm. Use mouse interfacing to draw polygon.

Aim:
To draw a convex polygon and fill it with desired color using seed fill algorithm

Software Requirements:
64-bit Open source Linux or its derivative
Open Source C++ Programming tool like G++/GCC

Objectives: Identify the polygon drawing algorithms and polygon filling algorithms of computer graphics

Outcomes: Implement computer graphics programs in C++ using the polygon drawing algorithm and fill the polygon using poly fill algorithm.

Theory Concepts/ Logic/ Algorithm:
Convex Polygon
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. The polygon is entirely contained in a closed half-plane defined by each of its edges. For each edge, the vertices not contained in the edge are on the same side of the line that the edge defines.
The angle at each vertex contains all other vertices in its interior (except the three vertices defining the angle).
Convex Polygons: In a convex polygon, any line segment joining any two inside points lies inside the polygon.

Seed Fill Algorithm
- Used when an area defined with multiple color boundaries
- Start at a point inside a region
- Replace a specified interior color (old color) with fill color
- Fill the 4-connected or 8-connected region until all interior points being replaced

Algorithm:
```cpp
void SeedFill ( int x, int y, color newcolor, color old)
{
    if ( ReadPixel ( x, y ) == old)
        SeedFill ( x+1, y, newcolor, old );
        SeedFill ( x-1, y, newcolor, old );
        SeedFill ( x, y+1, newcolor, old );
        SeedFill ( x, y-1, newcolor, old );
}
```
Assignment No B3.

Write C++/Java program to draw any object such as flower, waves using any curve generation techniques

Aim:
To draw a objects using any curve generation technique.

Software Requirements:
- 64-bit Open source Linux or its derivative
- Open Source C++ Programming tool like G++/GCC

Objectives: Identify different curve generation algorithms of computer graphics

Outcomes: Implement computer graphics programs in C++ using curve generation techniques.

Theory Concepts/ Logic/ Algorithm:

**Bezier Curve**

A Bezier curve is determined by a defining polygon

![Bezier Curve and its defining polygon](image)

Bezier curves have the following properties –

- They generally follow the shape of the control polygon, which consists of the segments joining the control points.
- They always pass through the first and last control points.
- They are contained in the convex hull of their defining control points.
- The degree of the polynomial defining the curve segment is one less that the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is 3, i.e. cubic polynomial.
- A Bezier curve generally follows the shape of the defining polygon.
- The direction of the tangent vector at the end points is same as that of the vector determined by first and last segments.
- The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.
- No straight line intersects a Bezier curve more times than it intersects its control polygon.
- They are invariant under an affine transformation.
- Bezier curves exhibit global control means moving a control point alters the shape of the whole curve.
A given Bezier curve can be subdivided at a point \( t=t_0 \) into two Bezier segments which join together at the point corresponding to the parameter value \( t=t_0 \).

These curves can be generated under the control of other points. Approximate tangents by using control points are used to generate the curve. The Bezier curve can be represented mathematically as

\[
\sum_{k=0}^{n} P_i B_i^n(t)
\]

Where \( n \) is the polynomial degree, \( i \) is the index, and \( t \) is the variable.

Algorithm:

The objective here is to find points in the middle of two nearby points and iterate this until we have no more iteration. The new values of points will give us the curve. The famous Bezier equation is the exact formulation of this idea. Here is the algorithm:

**Step 1**: Select a value \( t \in [0,1] \). This value remains constant for the rest of the steps.

**Step 2**: Set \( P_i[0](t) = P_i \) for \( i = 0,...,n \).

**Step 3**: For \( j=0,...,n \), set \( \frac{P_i^{[j]}(t)}{i!} = (1-t) \frac{P_i^{[j-1]}(t)}{(j-1)!} + t \frac{P_i^{[j-1]}(t)}{(j-1)!} \) for \( i = j,...,n \).

**Step 4**: \( g(t) = P_{[n]}(n) \) for

A whole algorithm could be summarized into a formula and a straightforward implementation would yield correct results. Here, \( n \) denotes the number of points and \( P \) denotes the points themselves. The factorial coefficients of the points are simply called the Bernstein basis functions

\[
\gamma(i) = \sum_{i=0}^{n} \frac{n!}{i!(n-i)!} (1-t)^{n-i} t^i
\]
Assignment No B4.

Write program to implement Cohen Sutherland Hodgman algorithm to clip any polygon. Provide the vertices of the polygon to be clipped and pattern of clipping interactively

**Aim:**
To clip a polygon using Cohen Sutherland Hodgman algorithm.

**Software Requirements:**
64-bit Open source Linux or its derivative
Open Source C++ Programming tool like G++/GCC

**Objectives:** Identify different clipping algorithms of computer graphics
**Outcomes:** Implement computer graphics programs in C++ using polygon clipping techniques.

**Theory Concepts/ Logic/ Algorithm:**
The Sutherland-Hodgman algorithm performs a clipping of a polygon against each window edge in turn. It accepts an ordered sequence of vertices v1, v2, v3, ..., vn and puts out a set of vertices defining the clipped polygon.

This figure represents a polygon (the large, solid, upward pointing arrow) before clipping has occurred.

The following figures show how this algorithm works at each edge, clipping the polygon.
a. Clipping against the left side of the clip window.
b. Clipping against the top side of the clip window.
c. Clipping against the right side of the clip window.
d. Clipping against the bottom side of the clip window.

As the algorithm goes around the edges of the window, clipping the polygon, it encounters four types of edges. All four edge types are illustrated by the polygon in the following figure. For each edge type, zero, one, or two vertices are added to the output list of vertices that define the clipped polygon.

![Polygon Clip Window Diagram]

The four types of edges are:

1. Edges that are totally inside the clip window. - add the second inside vertex point
2. Edges that are leaving the clip window. - add the intersection point as a vertex
3. Edges that are entirely outside the clip window. - add nothing to the vertex output list
4. Edges that are entering the clip window. - save the intersection and inside points as vertices

Assume that we're clipping a polygon's edge with vertices at \((x_1, y_1)\) and \((x_2, y_2)\) against a clip window with vertices at \((x_{\text{min}}, y_{\text{min}})\) and \((x_{\text{max}}, y_{\text{max}})\).

The location \((IX, IY)\) of the intersection of the edge with the left side of the window is:

i. \(IX = x_{\text{min}}\)
ii. \(IY = \text{slope} \times (x_{\text{min}} - x_1) + y_1\), where the slope = \((y_2 - y_1)/(x_2 - x_1)\)

The location of the intersection of the edge with the right side of the window is:

i. \(IX = x_{\text{max}}\)
ii. \(IY = \text{slope} \times (x_{\text{max}} - x_1) + y_1\), where the slope = \((y_2 - y_1)/(x_2 - x_1)\)

The intersection of the polygon's edge with the top side of the window is:

i. \(IX = x_1 + (y_{\text{max}} - y_1) / \text{slope}\)
ii. \(IY = y_{\text{max}}\)

Finally, the intersection of the edge with the bottom side of the window is:

i. \(IX = x_1 + (y_{\text{min}} - y_1) / \text{slope}\)
ii. \(IY = y_{\text{min}}\)
Algorithm:

1. Input Coordinates of all vertices of the polygon
2. Input coordinates of the clipping window
3. Consider the left edge of the window
4. Compare the vertices of each edge of the polygon, individually with the clipping plane
5. Save the resulting intersections and vertices in the new list of vertices according to four possible relationships between the edge and the clipping boundary discussed earlier
6. Repeat the steps 4 and 5 for remaining edges of the clipping window. Each time the resultant list of vertices is successively passed to process the next edge of the clipping window
7. Stop
Assignment No B5.

Write C++/Java program to implement translation, sheer, rotation and scaling transformations on triangle

Aim:
To implement translation, sheer, rotation, and scaling transformations

Software Requirements:
64-bit Open source Linux or its derivative
Open Source C++ Programming tool like G++/GCC

Objectives: Identify different object transformations of computer graphics
Outcomes: Implement computer graphics programs in C++ using object transformations.

Theory Concepts/ Logic/ Algorithm:

1. Translation: Translation is defined as moving the object from one position to another position along straight line path.

We can move the objects based on translation distances along x and y axis. tx denotes translation distance along x-axis and ty denotes translation distance along y axis.

Translation Distance: It is nothing but by how much units we should shift the object from one location to another along x, y-axis.

Consider (x,y) are old coordinates of a point. Then the new coordinates of that same point (x',y') can be obtained as follows:

\[ x' = x + tx \]
\[ y' = y + ty \]

We denote translation transformation as P. we express above equations in matrix form as:

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix}
= \begin{bmatrix}
  x \\
  y
\end{bmatrix} + \begin{bmatrix}
  tx \\
  ty
\end{bmatrix}
\]

\[ x,y \text{--- old coordinates} \]
\[ x',y' \text{--- new coordinates after translation} \]
\[ tx,ty \text{--- translation distances, T is} \]
2. **Scaling:** Scaling refers to changing the size of the object either by increasing or decreasing. We will increase or decrease the size of the object based on scaling factors along x and y-axis.

![Before and After Scaling Diagram](image)

If \((x, y)\) are old coordinates of the object, then new coordinates of the object after applying scaling transformation are obtained as:

\[
\begin{align*}
x' &= x \times sx \\
y' &= y \times sy.
\end{align*}
\]

\(sx\) and \(sy\) are scaling factors along x-axis and y-axis. We express the above equations in matrix form as:

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} =
\begin{bmatrix}
s_x & 0 \\
0 & s_y
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
\]

**Scaling Matrix**

3. **Rotation:** A rotation repositions all points in an object along a circular path in the plane centered at the pivot point. We rotate an object by an angle \(\theta\).

New coordinates after rotation depend on both \(x\) and \(y\):

- \(x' = x \cos \theta - y \sin \theta\)
- \(y' = x \sin \theta + y \cos \theta\)

or in matrix form:

\[
P' = R \cdot P,
\]

\(R\)-rotation matrix.

![Before and After Rotation Diagram](image)
4. **Reflection**: Reflection is nothing but producing mirror image of an object. Reflection can be done just by rotating the object about given axis of reflection with an angle of 180 degrees.

![X-axis reflection and Y-axis reflection images](image)

5. **Shear**:

1. Shear is the translation along an axis by an amount that increases linearly with another axis (Y). It produces shape distortions as if objects were composed of layers that are caused to slide over each other.
2. Shear transformations are very useful in creating italic letters and slanted letters from regular letters.
3. Shear transformation changes the shape of the object to a slant position.

   ![Original, X-shear, and Y-shear images](image)

4. Shear transformation is of 2 types:
   a. **X-shear**: changing x-coordinate value and keeping y constant
      
      \[ x' = x + shx \cdot y \]
      
      \[ y' = y \]
   b. **Y-shear**: changing y coordinates value and keeping x constant
      
      \[ x' = x \]
      
      \[ y' = y + shy \cdot x \]

   shx and shy are shear factors along x and y-axis.
Write C++/Java program to draw 3-D cube and perform following transformations on it using OpenGL. a) Scaling b) Translation c) Rotation about one axis

Aim:
To perform 3 D transformations using OpenGL.

Software Requirements:
64-bit Open source Linux or its derivative
Open Source C++ Programming tool like G++/GCC

Objectives: Identify different 3D transformations of computer graphics
Outcomes: Implement computer graphics programs with 3 D transformations using OpenGL

Theory Concepts/ Logic/ Algorithm:

OpenGL:
OpenGL is a software interface that allows the programmer to create 2D and 3D graphics images. OpenGL is both a standard API and the implementation of that API. You can call the functions that comprise OpenGL from a program you write and expect to see the same results no matter where your program is running. OpenGL is independent of the hardware, operating, and windowing systems in use. The fact that it is windowing-system independent, makes it portable. OpenGL program must interface with the windowing system of the platform where the graphics are to be displayed. Therefore, a number of windowing toolkits have been developed for use with OpenGL. OpenGL functions in a client/server environment. That is, the application program producing the graphics may run on a machine other than the one on which the graphics are displayed. The server part of OpenGL, which runs on the workstation where the graphics are displayed, can access whatever physical graphics device or frame buffer is available on that machine.

In OpenGL translation, rotation, and scaling are performed using commands such as:

\texttt{glTranslate\{fd\}(X,Y,Z)} – \texttt{glTranslatef(1.0, 2.5, 3.0)}
\texttt{glRotate\{df\}(Angle, X, Y, Z)} - \texttt{glRotatef(60.0, 0.0, 0.0, 1.0)}
\texttt{glScale\{df\}(X, Y, Z)} - \texttt{glScalef(1.0, 1.5, 2.0)}
3D Transformations:

A point in 3-Dimensional space \([x\ y\ z]\) is represented by a four dimensional position vector.

\[
[x'\ y'\ z'\ h] = [x\ y\ z\ 1][T]
\]

The generalized 4*4 transformation matrix is,

\[
[T] = \begin{bmatrix}
a & b & c & p \\
d & e & f & q \\
g & i & j & r \\
l & m & n & s
\end{bmatrix}
\]

3D Scaling:- Diagonal terms produce local and overall scaling.

\[
[x\ y\ z\ 1] = \begin{bmatrix}
a & 0 & 0 & 0 \\
0 & e & 0 & 0 \\
0 & 0 & j & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
S = (ax, ex, jz) = [ax\ ey\ jz\ 1] = [x'\ y'\ z'\ 1]
\]
Overall / Uniform Scaling

3 Dimensional shearing: [off diagonal terms produce shearing in 3D]

\[
\begin{bmatrix}
1 & b & c & 0 \\
0 & d & e & f \\
g & 0 & i & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[= [x + yd + gz, bx + y + iz, cx + fy + z, 1] \]

3 Dimensional rotation:

For rotation about the x-axis, the x coordinates of the position vectors do not change the rotation occurs in planes perpendicular to the x-axis.

Transformation matrix for rotation about x-axis by an angle is

\[
[T] = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos \theta & \sin \theta & 0 \\
0 & -\sin \theta & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Rotation about z-axis:

\[ R_z = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

Rotation about y-axis:

\[ R_y = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]
3D Reflection

Reflection through the XY plane is

\[
[T] = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & -1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

Reflection through YZ plane

\[
[T] = \begin{bmatrix}
-1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

Reflection through the XZ plane

\[
[T] = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & -1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

3D Translation:

\[
[T] = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
h & m & n & 1 \\
\end{bmatrix}
\]

Translated Homogenous are

\[
[x' \ y' \ z' \ h] = [x \ y \ z \ 1] 
= \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
h & m & n & 1 \\
\end{bmatrix}
\]

\[
[x' \ y' \ z' \ h]=(x + h) \ (y + m) \ (z + m) \ 1
\]
Write C++/Java program to simulate Vehicle/boat locomotion or similar scene.

**Aim:**
To simulate Vehicle locomotion.

**Software Requirements:**
64-bit Open source Linux or its derivative
Open Source C++ Programming tool like G++/GCC

**Objectives:** Identify transformation of objects to simulate objects in computer graphics

**Outcomes:** Implement simple animations using object transformations.

**Theory Concepts/ Logic/ Algorithm:**
For moving any object, we incrementally calculate the object coordinates and redraw the picture to give a feel of animation by using for loop. Suppose if we want to move a circle from left to right means, we have to shift the position of circle along x-direction continuously in regular intervals.

The program illustrates the movement of objects by using for loop and also using transformations like rotation, translation etc.