1. Aim: To write and execute a Python program to draw a straight line using the **slope-intercept equation**.

Description: The equation of a straight line is:

y=mx+by = mx + b

where:

- m = slope of the line
- b = y-intercept
- x = x-coordinate
- y = corresponding y-coordinate

In this program, the slope m is taken as 2 and the y-intercept b is taken as 1. For a range of x-values, the corresponding y-values are calculated using the equation y = mx + b.

The calculated points are then plotted using the matplotlib library.

Algorithm:

Step 1: Start

Step 2: Import the matplotlib.pyplot library.

Step 3: Define the slope m and y-intercept b.

Step 4: Generate a range of x-values from -10 to 10.

Step 5: Calculate the corresponding y-values using: y=mx+by = mx + b

Step 6: Plot the calculated x and y values.

Step 7: Add X-axis and Y-axis labels.

Step 8: Add a title to the graph.

Step 9: Enable the grid.

Step 10: Display the graph.

Step 11: End

Result: Thus, the program was successfully executed and a straight line was drawn using the **slope-intercept equation**.

2. Aim: To write and execute a Python program to draw a line using the **Digital Differential Analyzer (DDA) line drawing algorithm**.

Description: DDA (Digital Differential Analyzer) is a line drawing algorithm used in computer graphics. It draws a straight line between two given points. It calculates the intermediate points between the starting and ending points. These points are plotted to form a straight line.

Algorithm:

Step 1: Import the matplotlib.pyplot library.

Step 2: Define the starting point (x1, y1) and ending point (x2, y2).

Step 3: Calculate dx and dy.

Step 4: Calculate the number of steps using:

steps=max(|dx|,|dy|)steps=max(|dx|,|dy|)

Step 5: Calculate x and y increments.

Step 6: Initialize the starting coordinates.

Step 7: Repeat the following steps for the calculated number of steps:

- Store the rounded x-coordinate.
- Store the rounded y-coordinate.
- Increment x.
- Increment y.

Step 8: Plot all generated points.

Step 9: Add labels, title, and grid.

Step 10: Display the graph.

Result: Thus, the program was successfully executed and a line was drawn between the given points using the **DDA line drawing algorithm**.

3. Aim: To write and execute a Python program to draw a straight line using **Bresenham's line drawing algorithm**.

Description: Bresenham's line algorithm is a **line drawing algorithm** used in computer graphics. It draws a line by selecting the **best pixels** between two given points. It uses **integer calculations**, so it is fast and efficient. It uses a **decision parameter** to decide the next pixel.

Algorithm:

Step 1: Define the starting point (x1, y1) and ending point (x2, y2).

Step 2: Calculate:

dx=x2−x1dx=x_2-x_1

dy=y2−y1dy=y_2-y_1

Step 3: Calculate the initial decision parameter:

p=2dy−dxp=2dy-dx

Step 4: Store the starting point.

Step 5: Increase x by one.

Step 6: Check the decision parameter.

Step 7: If p < 0, keep the same y-coordinate and update:

p=p+2dyp=p+2dy

Step 8: Otherwise, increase y by one and update:

p=p+2dy−2dxp=p+2dy-2dx

Step 9: Repeat until the ending point is reached.

Step 10: Plot all generated points.

Result: Thus, the program was successfully executed and a line was drawn using **Bresenham's line drawing algorithm**.

4. Aim: To write and execute a Python program to draw a circle using **polar coordinates**.

Description: The **Polar Circle Drawing Method** is used to draw a circle in computer graphics. It calculates the points of a circle using the **center, radius, and angle**. The angle is changed from **0° to 360°** to get points on the circle. These points are plotted to form the complete circle.

Algorithm:

Step 1: Import matplotlib and math.

Step 2: Define the center of the circle.

Step 3: Define the radius.

Step 4: Create empty lists for x and y coordinates.

Step 5: Generate angles from 0° to 360°.

Step 6: Convert each angle from degrees to radians.

Step 7: Calculate x using: x=xc+rcos⁡θx=x_c+r\cos\theta

Step 8: Calculate y using: y=yc+rsin⁡θy=y_c+r\sin\theta

Step 9: Store the generated coordinates.

Step 10: Plot the points.

Step 11: Set equal scaling for both axes.

Step 12: Display the circle.

Result: Thus, the program was successfully executed and a circle was drawn using **polar coordinate scan conversion**.

5. Aim: To write and execute a Python program to draw a circle using **Bresenham's circle drawing algorithm**.

Description: Bresenham's Circle Algorithm is an **efficient circle drawing algorithm** used in computer graphics. It uses a **decision parameter** to select the next pixel. It uses the **8-way symmetry** of a circle to draw all the points. The initial decision parameter is **p = 3 − 2r**, and the process continues until **x > y**.

Algorithm:

Step 1: Define the center (xc, yc) and radius r.

Step 2: Initialize: x=0x=0 y=ry=r

Step 3: Set the initial decision parameter: p=3−2rp=3-2r

Step 4: Calculate the eight symmetric points.

Step 5: Store these points.

Step 6: If p < 0, update: p=p+4x+6p=p+4x+6

Step 7: Otherwise, update: p=p+4(x−y)+10p=p+4(x-y)+10 and decrease y.

Step 8: Increase x.

Step 9: Repeat until x > y.

Step 10: Plot all points.

Result: Thus, the program was successfully executed and a circle was drawn using **Bresenham's circle drawing algorithm**.

6. Aim: To write and execute a Python program to draw a circle using the **Midpoint Circle Drawing Algorithm**.

Description: The **Midpoint Circle Algorithm** is used to draw a circle in computer graphics. It uses a **decision parameter** to select the next pixel. It uses the **8-way symmetry** of a circle to generate the points. The initial values are **x = 0, y = r, and p = 1 − r**.

Algorithm:

Step 1: Define the center and radius.

Step 2: Initialize x = 0.

Step 3: Initialize y = r.

Step 4: Calculate the initial decision parameter: p=1−rp=1-r

Step 5: Generate the eight symmetric points.

Step 6: If p < 0, update: p=p+2x+3p=p+2x+3

Step 7: Otherwise update: p=p+2(x−y)+5p=p+2(x-y)+5 and decrease y.

Step 8: Increase x.

Step 9: Repeat until x > y.

Step 10: Plot all generated points.

Result: Thus, the program was successfully executed and a circle was drawn using the **Midpoint Circle Drawing Algorithm**.

7. Aim: To write and execute a Python program to perform **2D transformations**, namely:

- Translation
- Rotation
- Scaling
- Reflection
- Shearing

on a two-dimensional object.

Description: 2D Transformations are operations used in computer graphics to **change the position, size, direction, or shape of an object**. The main types of 2D transformations are **Translation, Rotation, Scaling, Reflection, and Shearing**. Translation moves an object from one position to another, Rotation turns an object through a specified angle, Scaling changes the size of an object, Reflection produces a mirror image, and Shearing slants or changes the shape of an object.

Algorithm:

Step 1: Import matplotlib and numpy.

Step 2: Define the coordinates of the original triangle.

Step 3: Perform translation by adding tx and ty.

Step 4: Perform rotation using the rotation matrix.

Step 5: Perform scaling using sx and sy.

Step 6: Perform reflection about the X-axis.

Step 7: Perform shearing in the X-direction.

Step 8: Create subplots for all transformations.

Step 9: Plot the original and transformed objects.

Step 10: Display all transformation results.

Result: Thus, the program was successfully executed and the **Translation, Rotation, Scaling, Reflection, and Shearing** transformations were performed on a 2D triangle.

8. Aim: To write and execute a Python program to implement the **Cohen-Sutherland line clipping algorithm** and perform **window-to-viewport mapping**.

Description: The **Cohen-Sutherland Algorithm** is used in computer graphics to **clip a line inside a rectangular window**. Each endpoint of the line is given a **4-bit region code** based on its position: left, right, top, or bottom. The code **0000** represents a point inside the window. Using these codes, the algorithm decides whether the line is **completely inside, completely outside, or needs clipping**. After clipping, the line can be displayed in the required viewport.

Algorithm:

Step 1: Define the clipping window.

Step 2: Define the original line endpoints.

Step 3: Define region codes for LEFT, RIGHT, BOTTOM, and TOP.

Step 4: Calculate the region code of each endpoint.

Step 5: If both codes are zero, accept the line.

Step 6: If the bitwise AND of the two codes is not zero, reject the line.

Step 7: Otherwise select the endpoint outside the clipping window.

Step 8: Calculate its intersection with the appropriate boundary.

Step 9: Replace the outside endpoint with the intersection point.

Step 10: Repeat the clipping process.

Step 11: If the line is accepted, calculate the viewport coordinates.

Step 12: Display the original line, clipping window, clipped line, and viewport.

Result: Thus, the program was successfully executed and **Cohen-Sutherland line clipping** and **window-to-viewport mapping** were performed successfully.

9. Aim: To write and execute a Python program to implement the **Liang-Barsky line clipping algorithm**.

Description: The **Liang-Barsky Algorithm** is an efficient line clipping algorithm used in computer graphics. It uses the **parametric form of a line** to find the portion of the line inside a clipping window. The value of **u ranges from 0 to 1**, representing the points between the starting and ending points of the line. The algorithm checks the **four boundaries** of the clipping window and selects the part of the line that lies inside the window.

Algorithm:

Step 1: Define the clipping window.

Step 2: Define the starting and ending points of the line.

Step 3: Calculate:

dx=x2−x1dx=x_2-x_1

dy=y2−y1dy=y_2-y_1

Step 4: Calculate the arrays p and q.

Step 5: Initialize:

u1=0u_1=0

u2=1u_2=1

Step 6: Process all four boundaries.

Step 7: If p = 0 and q < 0, reject the line.

Step 8: If p < 0, update u1.

Step 9: If p > 0, update u2.

Step 10: If u1 > u2, reject the line.

Step 11: Otherwise calculate the clipped endpoints.

Step 12: Display the original line, clipped line, and clipping window.

Result: Thus, the program was successfully executed and the given line was clipped using the **Liang-Barsky line clipping algorithm**.

10. Aim: To write and execute a Python program to perform **3D transformations**, namely Translation, Rotation, and Scaling on a three-dimensional object.

Description: 3D Transformations are used in computer graphics to change the position, size, or direction of an object in 3D space. The main transformations are Translation, Rotation, and Scaling. Translation moves the object, Rotation turns the object, and Scaling changes its size. A cube is used as the original 3D object.

Algorithm:

Step 1: Import matplotlib and numpy.

Step 2: Define the eight vertices of the cube.

Step 3: Define the twelve edges of the cube.

Step 4: Create a function to draw the cube.

Step 5: Perform translation by adding translation values to all vertices.

Step 6: Create the rotation matrix for rotation around the Z-axis.

Step 7: Apply the rotation matrix to the vertices.

Step 8: Perform scaling by multiplying the coordinates by scaling factors.

Step 9: Create separate 3D subplots.

Step 10: Display the original, translated, rotated, and scaled cubes.

Result: Thus, the program was successfully executed and **Translation, Rotation, and Scaling** transformations were performed on a 3D cube.

11. Aim: To write and execute a Python program to draw a **house (HUT)** using basic geometrical shapes and line segments.

Description: A geometrical figure can be constructed by combining basic lines, rectangles, and polygons.

In this program, a house is created using:

- Rectangular body
- Triangular roof
- Rectangular door
- Two windows
- Window cross-lines

The matplotlib library is used to draw the geometrical figure.

Algorithm:

Step 1: Import matplotlib.pyplot.

Step 2: Create a figure window.

Step 3: Define the coordinates of the house body.

Step 4: Plot the house body.

Step 5: Define and plot the roof.

Step 6: Define and plot the door.

Step 7: Define and plot the left window.

Step 8: Define and plot the right window.

Step 9: Draw horizontal and vertical lines inside the windows.

Step 10: Add title and axis labels.

Step 11: Enable the grid.

Step 12: Set equal scaling.

Step 13: Display the house.

Result: Thus, the program was successfully executed and a **house/HUT geometrical figure** was drawn using Python.

12. Aim: To write and execute a Python program to draw a **regular decagon**.

Description: A **decagon** is a polygon with **10 sides and 10 vertices**. In a regular decagon, the vertices are equally spaced around a circle. The angle between two consecutive vertices is **36°**. The coordinates of the vertices are calculated using the **radius and angle** of the circle.

Algorithm:

Step 1: Import matplotlib and math.

Step 2: Set the number of sides to 10.

Step 3: Define the radius.

Step 4: Create empty lists for x and y coordinates.

Step 5: Calculate the angle for each vertex.

Step 6: Calculate x-coordinate using:

x=rcos⁡θx=r\cos\theta

Step 7: Calculate y-coordinate using:

y=rsin⁡θy=r\sin\theta

Step 8: Store all coordinates.

Step 9: Plot the points and connect them.

Step 10: Add title and axis labels.

Step 11: Enable grid and equal scaling.

Step 12: Display the decagon.

Result: Thus, the program was successfully executed and a **regular decagon with 10 sides** was drawn.

13. Aim: To write and execute a Python program to demonstrate **polygon filling** using Python's matplotlib library.

Description: A polygon is a closed 2D shape made up of straight lines. Polygon filling is the process of filling the inside of a polygon with a color or pattern. In this program, plt.fill() fills the polygon, and plt.plot() draws its boundary.

Algorithm:

Step 1: Import matplotlib.pyplot.

Step 2: Define the x-coordinates of the polygon vertices.

Step 3: Define the y-coordinates of the polygon vertices.

Step 4: Use the fill() function to fill the polygon.

Step 5: Add the first vertex at the end of the coordinate lists to close the boundary.

Step 6: Draw the boundary of the polygon.

Step 7: Add the title and axis labels.

Step 8: Enable the grid.

Step 9: Set equal scaling for both axes.

Step 10: Display the filled polygon.

Result: Thus, the program was successfully executed and the given polygon was **filled successfully** using Python.