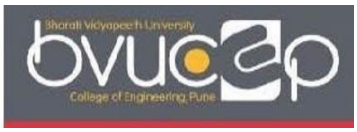
BHARATI VIDYAPEETH DEEMED UNIVERSITY
COLLEGE OF ENGINEERING, PUNE

Lab Manual

Computer Graphics and Visualisation



DEPARTMENT OF COMPUTER ENGINEERING

VISION OF THE INSTITUTE

To be World Class Institute for Social Transformation Through Dynamic Education.

MISSION OF THE INSTITUTE

A. To provide quality technical education with advanced equipment, qualified faculty members, infrastructure to meet needs of profession and society.

B. To provide an environment conducive to innovation, creativity, research, and entrepreneurial leadership.

C. To practice and promote professional ethics, transparency and accountability for social community, economic and environmental conditions.

VISION OF THE DEPARTMENT

To pursue and excel in the Endeavour for creating globally recognized Computer Engineers through Quality education.

MISSION OF THE DEPARTMENT

- To impart engineering knowledge and skills confirming to a dynamic curriculum.
- To develop professional, entrepreneurial & research competencies encompassing continuous intellectual growth.
- To produce qualified graduates exhibiting societal and ethical responsibilities in working environment.

PROGRAM EDUCATIONAL OBJECTIVES

1. Demonstrate technical and professional competencies by applying engineering fundamentals, computing principles and technologies.
2. Learn, Practice, and grow as skilled professionals/ entrepreneur/researchers adapting to the evolving computing landscape.
3. Demonstrate professional attitude, ethics, understanding of social context and interpersonal skills leading to a successful career.

PROGRAM SPECIFIC OUTCOMES

1. To apply fundamental knowledge and technical skills towards solving Engineering problems.
2. To employ expertise and ethical practise through continuing intellectual growth and adapting to the working environment.

PROGRAM OUTCOMES

1. To apply knowledge of computing and mathematics appropriate to the domain.
2. To logically define, analyze and solve real world problems.

3. To apply design principles in developing hardware/software systems of varying complexity that meet the specified needs.
4. To interpret and analyze data for providing solutions to complex engineering problems.
5. To use and practice engineering and IT tools for professional growth.
6. To understand and respond to legal and ethical issues involving the use of technology for societal benefits.
7. To develop societal relevant projects using available resources.
8. To exhibit professional and ethical responsibilities.
9. To work effectively as an individual and a team member within the professional environment.
10. To prepare and present technical documents using effective communication skills.
11. To demonstrate effective leadership skills throughout the project management life cycle.
12. To understand the significance of lifelong learning for professional development.

GENRAL INSTUCTIONS:

- Equipment in the lab is meant for the use of students. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care.
- Students are required to carry their reference materials, files and records with completed assignment while entering the lab.
- Students are supposed to occupy the systems allotted to them and are not supposed to talk or make noise in the lab.
- All the students should perform the given assignment individually.
- Lab can be used in free time/lunch hours by the students who need to use the systems should take prior permission from the lab in-charge.
- All the Students are instructed to carry their identity cards when entering the lab.
- Lab files need to be submitted on or before date of submission.
- Students are not supposed to use pen drives, compact drives or any other storage devices in the lab.
- For Laboratory related updates and assignments students should refer to the notice board in the Lab.

EXAMINATION SCHEME

Practical Exam: 25 Marks
Term Work: 25 Marks
Total: 50 Marks
Minimum Marks required: 20 Marks

PROCEDURE OF EVALUATION

Each practical/assignment shall be assessed continuously on the scale of 25 marks. The distribution of marks as follows.

Sr. No	Evaluation Criteria	Marks for each Criteria	Rubrics
1	**Timely Submission**	**07**	➢ Punctuality reflects the work ethics. Students should reflect that work ethics by completing the lab assignments and reports in a timely manner without being reminded or warned.
2	**Presentation**	**06**	➢ Student are expected to write the technical document (lab report) in their own words. The presentation of the contents in the lab report should be complete, unambiguous, clear, understandable. The report should document approach/algorithm/design and code with proper explanation.
3	**Understanding**	**12**	➢ Correctness and Robustness of the code is expected. The Learners should have an in-depth knowledge of the practical assignment performed. The learner should be able to explain methodology used for designing and developing the program/solution. He/she should clearly understand the purpose of the assignment and its outcome.

LABORATORY USAGE

Students use computers for executing the lab experiments, document the results and to prepare the technical documents for making the lab reports.

OBJECTIVE

The objective of this lab is to make students aware and practice implementation of various advance data structures by designing algorithms and implementing programs in C.

PRACTICAL PRE-REQUISITE

- C and C++ Programming Language Skills
- Data structure and algorithm.

SOFTWARE REQUIREMENTS

- C compiler.
- OpenGL

COURSE OUTCOMES

1. Apply fundamental concepts and practical skills in computer graphics.
2. Implement and use classic and modern algorithms and data structures in computer graphic to 3-D geometry, 3D modelling and 3D object Representation.
3. Acquire practical skills on additional advanced concepts, e.g. hidden surfaces & lines, curves & fractals.
4. Demonstrate graphics programming skills for different animation techniques & virtual reality.
5. Improve different solid modelling skills.
6. Apply basics of rendering & physical based modelling to image.

CONTRIBUTION TO PROGRAM OUTCOMES

Program Outcomes												PSOs	
PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2
3	3	3		1	3	3	2	3	3		3	3	2
3	3	3	3	2	1		1	1		3	3	3	2
3	3	3	3	3	1	1		1	2	3	3	3	2
3	3	3	3	3	3	3	3	3			3	3	2
	3	3	1	1			1	1		1	1	2	
2		3		3	3	3				3	3	2	2

DESIGN EXPERIENCE GAINED

The students gain moderate design experience by several graphics related algorithms and convert that its execution using various tools.

LEARNING EXPERIENCE GAINED

The students learn both soft skills and technical skills while they are undergoing the practical sessions. The soft skills gained in terms of communication, presentation and behavior. While technical skills they gained in terms of programming and efficient algorithm design using data structures.

Assignment list

Class: B. Tech (Computer Engineering) Div.-I & II **Semester**: - IV

Subject: Computer Graphics & Visualisation (CG&V)

Sr. No.	Name of Experiment/Assignment
01	Basics of computer graphics.
02	Study of Line Generation Algorithms.
03	Study of Circle Drawing Algorithms.
04	Study of Polygon Filling Algorithms.
05	Study of 2-D Clipping.
06	Study of 2-D Transformation.
07	Study of 3-D Transformation.
08	Study of Hidden Surfaces and Lines.
09	Study of Animation.
10	Study of Curve Generation.

Assignment No.- 01

Basics of computer graphics

- **Aim:** To understand fundamentals of computer graphics and draw BVDUCOE logo using inbuilt Fuctions.
- **Theory:**

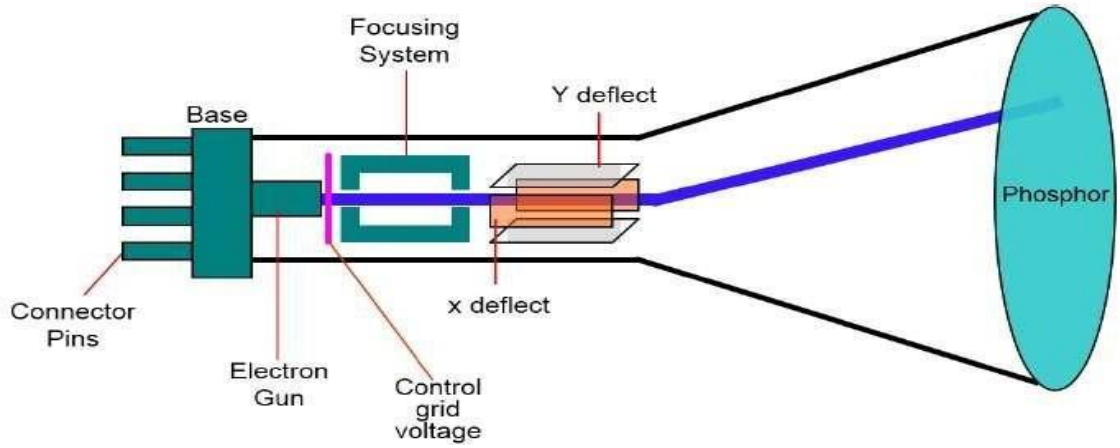
Computer graphics is an art of drawing pictures on computer screens with the help of programming. It involves computations, creation, and manipulation of data. In other words, we can say that computer graphics is a rendering tool for the generation and manipulation of images.

Cathode Ray Tube

The primary output device in a graphical system is the video monitor. The main element of a video monitor is the Cathode Ray Tube (CRT), shown in the following illustration.

The operation of CRT is very simple −

- The electron gun emits a beam of electrons (cathode rays).
- The electron beam passes through focusing and deflection systems that direct it towards specified positions on the phosphor-coated screen.
- When the beam hits the screen, the phosphor emits a small spot of light at each position contacted by the electron beam.
- It redraws the picture by directing the electron beam back over the same screen points quickly.



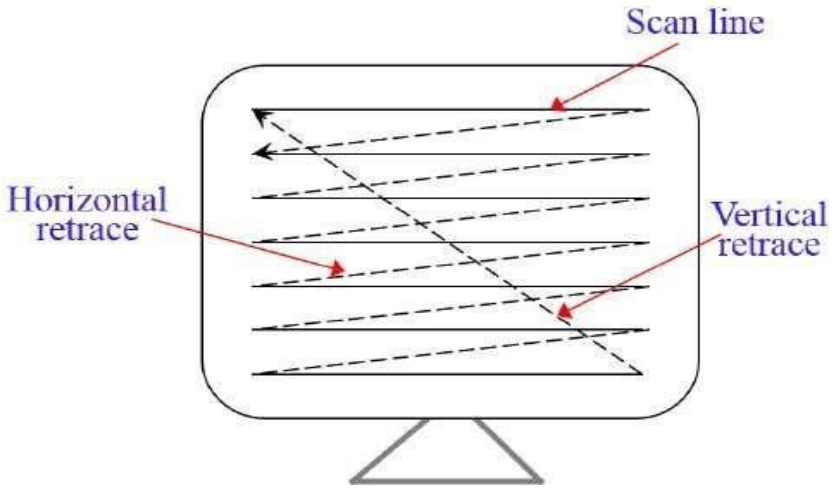
There are two ways (Random scan and Raster scan) by which we can display an object on the screen.

Raster Scan

In a raster scan system, the electron beam is swept across the screen, one row at a time from top to bottom. As the electron beam moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots.

Picture definition is stored in memory area called the Refresh Buffer or Frame Buffer. This memory area holds the set of intensity values for all the screen points. Stored intensity values are then retrieved from the refresh buffer and "painted" on the screen one row (scan line) at a time as shown in the following illustration.

Each screen point is referred to as a pixel (picture element) or pel. At the end of each scan line, the electron beam returns to the left side of the screen to begin displaying the next scan line.

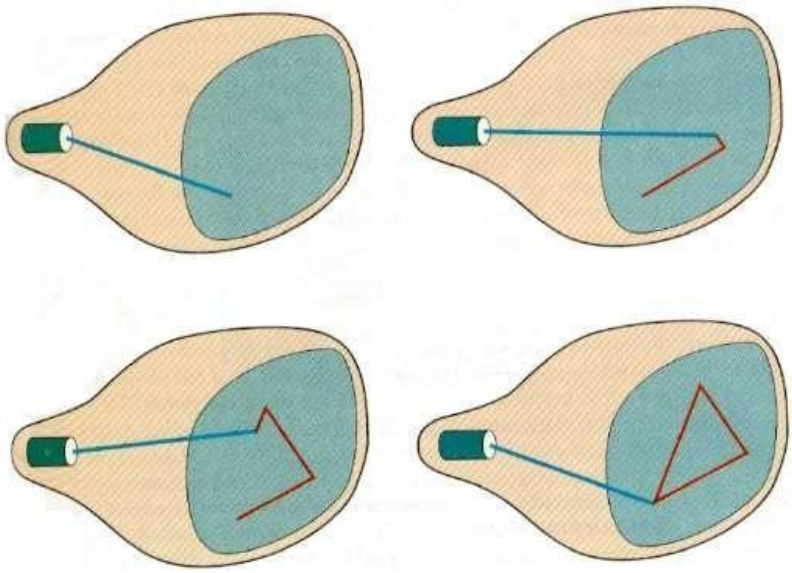


Random Scan (Vector Scan)

In this technique, the electron beam is directed only to the part of the screen where the picture is to be drawn rather than scanning from left to right and top to bottom as in raster scan. It is also called vector display, stroke-writing display, or calligraphic display.

Picture definition is stored as a set of line-drawing commands in an area of memory referred to as the refresh display file. To display a specified picture, the system cycles through the set of commands in the display file, drawing each component line in turn. After all the line-drawing commands are processed, the system cycles back to the first line command in the list.

Random-scan displays are designed to draw all the component lines of a picture 30 to 60 times each second.



Application of Computer Graphics

Computer Graphics has numerous applications, some of which are listed below −

- Computer graphics user interfaces (GUIs) − A graphic, mouse-oriented paradigm which allows the user to interact with a computer.
- Business presentation graphics − "A picture is worth a thousand words".
- Cartography − Drawing maps.
- Weather Maps − Real-time mapping, symbolic representations.
- Satellite Imaging − Geodesic images.
- Photo Enhancement − Sharpening blurred photos.
- Medical imaging − MRIs, CAT scans, etc. - Non-invasive internal examination.

- Engineering drawings − mechanical, electrical, civil, etc. - Replacing the blueprints of the past.
- Typography − The use of character images in publishing - replacing the hard type of the past.
- Architecture − Construction plans, exterior sketches - replacing the blueprints and hand drawings of the past.
- Art − Computers provide a new medium for artists.
- Training − Flight simulators, computer aided instruction, etc.
- Entertainment − Movies and games.
- Simulation and modeling − Replacing physical modeling and enactments

Functions of Graphics.h

C graphics using graphics.h functions can be used to draw different shapes, display text in different fonts, change colors and many more. Using functions of graphics.h in Turbo C compiler you can make graphics programs, animations, projects and games. You can draw circles, lines, rectangles, bars and many other geometrical figures. You can change their colors using the available functions and fill them. Following is a list of functions of graphics.h header file.

`initGraphics()` `initGraphics(width, height)`	Creates the graphics window on the screen.
`drawArc(bounds, start, sweep)` `drawArc(x, y, width, height, start, sweep)`	Draws an elliptical arc inscribed in a rectangle.
`fillArc(bounds, start, sweep)` `fillArc(x, y, width, height, start, sweep)`	Fills a wedge-shaped area of an elliptical arc.

`drawImage(filename, pt)`
`drawImage(filename, x, y)`
`drawImage(filename, bounds)`
`drawImage(filename, x, y, width, height)`
	Draws the image from the specified file with its upper left corner at the specified point.

`getImageBounds(filename)`	Returns the bounds of the image contained in the specified file.
`drawLine(p0, p1)` `drawLine(x0, y0, x1, y1)`	Draws a line connecting the specified points.

`drawPolarLine(p0, r, theta)` `drawPolarLine(x0, y0, r, theta)`	Draws a line of length `r` in the direction `theta` from the initial point.
`drawOval(bounds)` `drawOval(x, y, width, height)`	Draws the frame of a oval with the specified bounds.
`fillOval(bounds)` `fillOval(x, y, width, height)`	Fills the frame of a oval with the specified bounds.
`drawRect(bounds)` `drawRect(x, y, width, height)`	Draws the frame of a rectangle with the specified bounds.
`fillRect(bounds)` `fillRect(x, y, width, height)`	Fills the frame of a rectangle with the specified bounds.

`drawPolygon(polygon)`
`drawPolygon(polygon, pt)`
`drawPolygon(polygon, x, y)`
	Draws the outline of the specified polygon.

`fillPolygon(polygon)`
`fillPolygon(polygon, pt)`
`fillPolygon(polygon, x, y)`
	Fills the frame of the specified polygon.

`drawString(str, pt)` `drawString(str, x, y)`	Draws the string `str` so that its baseline origin appears at the specified point.
`getStringWidth(str)`	Returns the width of the string `str` when displayed in the current font.
`setFont(font)`	Sets a new font.
`getFont()`	Returns the current font.
`setColor(color)`	Sets the color used for drawing.
`getColor()`	Returns the current color as a string in the form `"#rrggbb"`.
`saveGraphicsState()`	Saves the state of the graphics context.
`restoreGraphicsState()`	Restores the graphics state from the most recent call to `saveGraphicsState()`.
`getWindowWidth()`	Returns the width of the graphics window in pixels.
`getWindowHeight()`	Returns the height of the graphics window in pixels.
`repaint()`	Issues a request to update the graphics window.
`pause(milliseconds)`	Pauses for the indicated number of milliseconds.
`waitForClick()`	Waits for a mouse click to occur anywhere in the window.

`setWindowTitle(title)`	Sets the title of the primary graphics window.
`getWindowTitle()`	Returns the title of the primary graphics window.
`exitGraphics()`	Closes the graphics window and exits from the application without waiting for any additional user interaction.

Assignment No.- 02

Study of Line Generation Algorithms

- **Aim:** To understand and implement line generation algorithms
- **Theory:**

Digital Differential Analyser (DDA)

Algorithm:

Step-1: Enter Starting Points (x1 and y1) and Ending Points (x2 and y2).

Step-2: Find m=(y2-y1)/(x2-x1)

Step-3: If(m<=1)

then

x1=x1+1;

y1=y1+m;

putpixel (x1, y1, WHITE);

Step-4: If(m>1)

y1=y1+1;

x1=x1+1/m;

putpixel (x1, y1, WHITE);

Step-5: Repeat Step-4 till x1 <=x2 or y1<=y2. (Till starting point to line goes up to ending point of the line).

- Advantages of DDA Algorithm

1. It is the simplest algorithm and it does not require special skills for implementation.
2. It is a faster method for calculating pixel positions than the direct use of equation y = mx + b. It eliminates the multiplication in the equation by

making use of raster

characteristics, so that appropriate increments are applied in the x or v direction to find the pixel positions along the line path.

❖ **Disadvantages of DDA Algorithm**

1. Floating point arithmetic in DDA algorithm is still time-consuming.
2. The algorithm is orientation dependent. Hence end point accuracy is poor.

Bresenham's Line Algorithm

Algorithm:

Step-1: Enter Starting Points (x1 and y1) and Ending Points (x2 and y2).

Step-2: Find dy=y2-y1; dx=x2-x1; and d=2*dy-dx;

Step-3: If(d>=0) then

x1=x1+1

;

y1=y1+

1;

d=d+2 (dy-dx);

Step-4: putpixel (x1, y1,

WHITE) If(d<0)

x1=x1+1;

d=d+2dy;

putpixel (x1, y1, WHITE)

Step-5: Repeat Step-4 till x1 <=x2 or y1<=y2. (Till starting point to line goes up to ending point of the line).

❖ **Advantages of DDA Algorithm**

1. No involvement of m (slope of line).
2. The line produced by this method is the best and smoothest.
3. The line produced by this method are better than all the other methods.
4. This method involves no approximations as it involves only addition of 1.

Assignment No. - 03

Study of Circle Drawing Algorithms

- **Aim:** To understand and implement line generation algorithms

- **Theory:**

DDA Algorithm

Step-1: Read r (radius of circle) and calculate ε (2 pow(n-1) ≤ r <2 pow (n) and ε= 2 pow(-n))

Step-2: start_x =0

start_y =r

Step-3: x1= start_x

y1= start_y

Step-4: do

{

x2 = x1 + εy1

y2= y1- εx2

Plot ((int x2), (int

y2)) x1= x2;

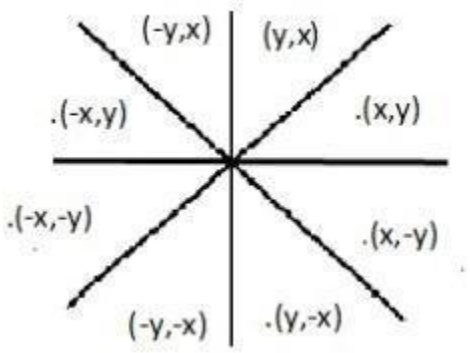
y1= y2;

} while (y1-start_y) <ε or (start_x-x1) >ε

Step-5: Stop.

The 8-way symmetry

The 8-way symmetry is the most important step in drawing the complete circle. It involves reflection with respect to x and y axis to generate 8 points from a given point. The method is explained as follows.



The above diagram explains the 8-way symmetry.

Point-1: Take a point (x, y) in first quadrant.

Point-2: Mirror reflect point (x, y) with respect to y axis we will get another point called (-x, y) Point-3: Mirror reflect point (x, y) with respect to x axis we will get another point called (x,-y) Point-4: Mirror reflect point (-x, y) with respect to x axis we will get another point called (-x,-y) Point-5: Take a point (y, x) in first quadrant.

Point-6: Mirror reflect point (y, x) with respect to y axis we will get another point called (-y, x) Point-7: Mirror reflect point (y, x) with respect to x axis we will get another point called (y,-x) Point-8: Mirror reflect point (-y, x) with respect to x axis we will get another point called (-y,-x)

Bresenham's Algorithm

Algorithm:

Step-1: Enter the center (h, k) and radius (r) of the circle.

Step-2: Find d=3-2*r and take x = 0, y = r;

Step-3: If (d>=0) then x = x +1 and y = y -1; d = d + 4*(x-y) + 10;

Step-4: if (d <0) then x =x +1 and d = d +4*x + 6;

Step-5: Implement 8-Way Symmetry as

follows Putpixel (x+h, y+k)
Putpixel (-x+h,
y+k) Putpixel
(x+h, -y+k)
Putpixel (-x+h, -
y+k) Putpixel
(y+h,x+k) Putpixel
(-y+h,x+k)
Putpixel (y+h,-
x+k) Putpixel (-
y+h,-x+k)

Step-6: Go to step 3 and repeat till x = r /sqrt (2);

Assignment No. - 04

Study of Polygon Filling Algorithms

➢ **Aim:** To understand and implement polygon filling algorithms

➢ **Theory :**

Scan-Line Polygon Fill Algorithm

For each scan line crossing a polygon, this algorithm locates the intersection points of the scan line with the polygon edges. These intersection points are shorted from left to right. Then, we fill the pixels between each intersection pair.

Boundary-Fill Algorithm

- This algorithm starts at a point inside a region and paint the interior outward towards the boundary.
- This is a simple method but not efficient: 1. It is recursive method which may occupy a large stack size in the main memory.

```
void BoundaryFill(int x, int y, COLOR fill, COLOR boundary)
{ COLOR current;
current=GetPixel(x,
y);
if (current<>boundary) and (current<>fill)
then { SetPixel(x,y,fill);
BoundaryFill(x+1,y,fill,boundary);
BoundaryFill(x-1,y,fill,boundary);
BoundaryFill(x,y+1,fill,boundary);
BoundaryFill(x,y-1,fill,boundary);
}
}
```

Flood-Fill Algorithm

- Flood-Fill is similar to Boundary-Fill. The difference is that Flood-Fill is to fill an area which I not defined by a single boundary color.

```
void FlooFill(int x, int y, COLOR fill, COLOR old_color)
{ if (GetPixel(x,y)==
old_color) { SetPixel(x,y,fill);
FloodFill(x+1,y,fill,boundary)
; FloodFill(x-
1,y,fill,boundary);
FloodFill(x,y+1,fill,boundary)
; FloodFill(x,y-
1,fill,boundary);
}
```

Seed-Fill Algorithm

One way to fill the polygon is to start from a given "seed" point known to be inside the polygon and highlight outward from this point i.e. neighbouring pixels until we encounter the boundary pixels. This approach is call seed fill because color floes from the seed pixels until reaching the polygon boundary like water flooding on the surface of the container.

Boundary-Fill Algorithm

- In this method, edges of polygon are drawn.
- Than starting from seed, any point inside the polygon we examine the neighbouring pixels to check whether the boundary pixels is reached.
- If boundary pixels are not reached, pixels are highlighted, and the process is continued until boundary pixels are reached.

```
void BoundaryFill(int x, int y, f_color, b_color)
{if (getpixel (x,y)!= b_color &&
getpixel(x,y)!=f_color) { Putpixel(x,y,f_color);
BoundaryFill(x+1,y,f_color,b_color);
BoundaryFill(x-1,y, f_color,b_color);
```

```
BoundaryFill(x,y+1,
f_color,b_color);
BoundaryFill(x,y-1,
f_color,b_color);
}
}
```

Flood-Fill Algorithm

-In this algorithm we fill in an area which is not defined within a single-color boundary.

-In such cases we can fill areas by replacing a specified interior color instead for searching a boundary color.

-WE examine the seed and examine the neighbouring

```
pixels. void FlooFill(int x, int y, old_color, new_color)
{if( getpixel (x,y) = new_color){
Putpixel(x,y,f_color);
FloodFill(x+1,y,old_color,new_col
or); FloodFill(x-1,y,
old_color,new_color);
FloodFill(x,y+1,
old_color,new_color);
FloodFill(x,y-1,
old_color,new_color);
}
}
```

Assignment No. - 05

Study of 2-D Clipping

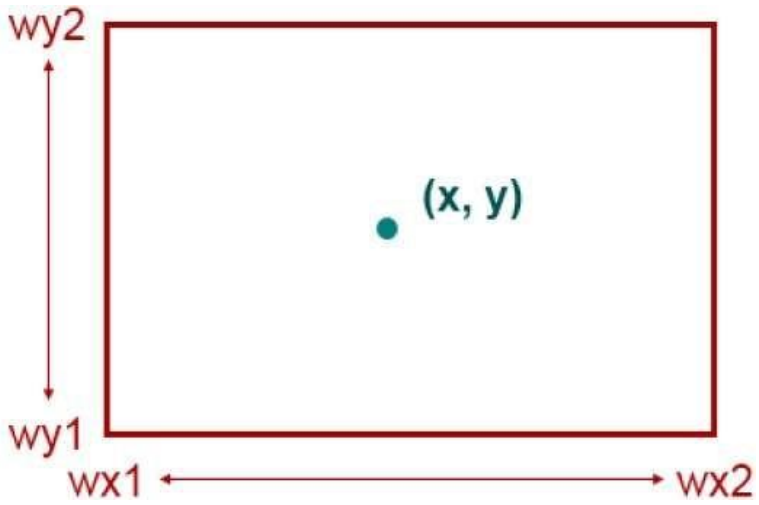
➢ **Aim:** To understand and implement 2D clipping.

➢ **Theory:**

The primary use of clipping in computer graphics is to remove objects, lines, or line segments that are outside the viewing pane. The viewing transformation is insensitive to the position of points relative to the viewing volume −especially those points behind the viewer −and it is necessary to remove these points before generating the view.

Point Clipping

Clipping a point from a given window is very easy. Consider the following figure, where the rectangle indicates the window. Point clipping tells us whether the given point (X, Y) is within the given window or not; and decides whether we will use the minimum and maximum coordinates of the window.
The X-coordinate of the given point is inside the window, if X lies in between Wx1 ≤ X ≤ Wx2. Same way, Y coordinate of the given point is inside the window, if Y lies in between Wy1 ≤ Y ≤ Wy2.



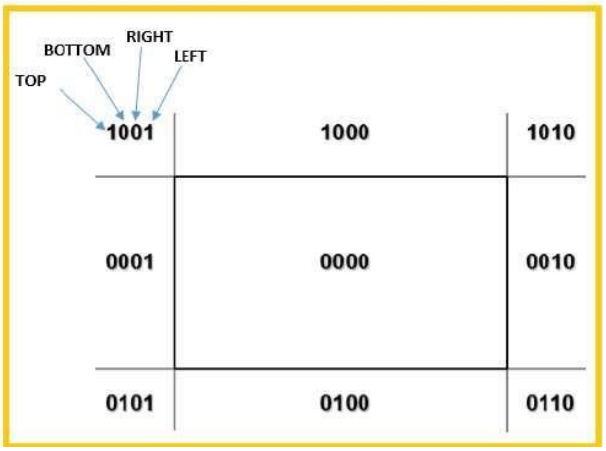
Line Clipping

The concept of line clipping is same as point clipping. In line clipping, we will cut the portion of line which is outside of window and keep only the portion that is inside the window.

Cohen-Sutherland Line Clippings

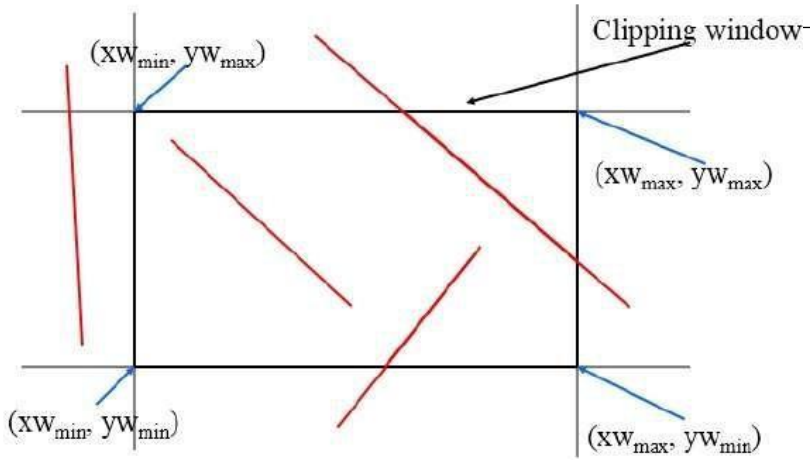
This algorithm uses the clipping window as shown in the following figure. The minimum coordinate for the clipping region is (XW_{min},YW_{min}) and the maximum coordinate for the clipping region is(XW_{max},YW_{max}).

We will use 4-bits to divide the entire region. These 4 bits represent the Top, Bottom, Right, and Left of the region as shown in the following figure. Here, the **TOP** and **LEFT** bit is set to 1 because it is the **TOP-LEFT** corner.



There are 3 possibilities for the line −

- Line can be completely inside the window (This line should be accepted).
- Line can be completely outside of the window (This line will be completely removed from the region).
- Line can be partially inside the window (We will find intersection point and draw only that portion of line that is inside region).



Algorithm

Step 1 − Assign a region code for each endpoints.

Step 2 − If both endpoints have a region code **0000** then accept this line. **Step 3** − Else, perform the logical **AND** operation for both region codes. **Step 3.1** − If the result is not **0000,** then reject the line.

Step 3.2 − Else you need clipping.

Step 3.2.1 − Choose an endpoint of the line that is outside the window.

Step 3.2.2 − Find the intersection point at the window boundary (base on region code).

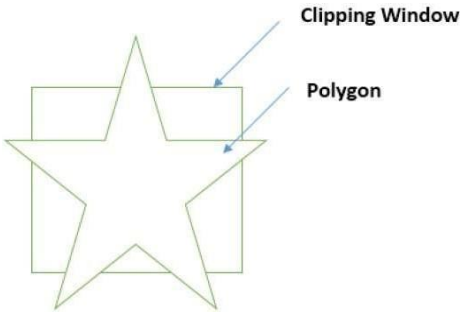
Step 3.2.3 − Replace endpoint with the intersection point and update the region code. **Step 3.2.4** − Repeat step 2 until we find a clipped line either trivially accepted or trivially rejected.

Step 4 − Repeat step 1 for other lines.

Polygon Clipping (Sutherland Hodgeman Algorithm)

A polygon can also be clipped by specifying the clipping window. Sutherland Hodgeman polygon clipping algorithm is used for polygon clipping. In this algorithm, all the vertices of the polygon are clipped against each edge of the clipping window.

First the polygon is clipped against the left edge of the polygon window to get new vertices of the polygon. These new vertices are used to clip the polygon against right edge, top edge, bottom edge, of the clipping window as shown in the following figure.



While processing an edge of a polygon with clipping window, an intersection point is found if edge is not completely inside clipping window and the a partial edge from the intersection point to the outside edge is clipped. The following figures show left, right, top and bottom edge clippings −

Computer Graphics

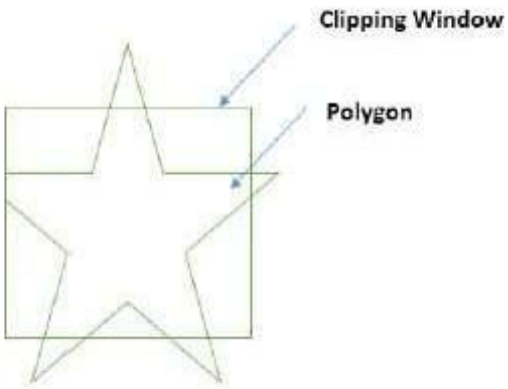


Figure: Clipping Left Edge

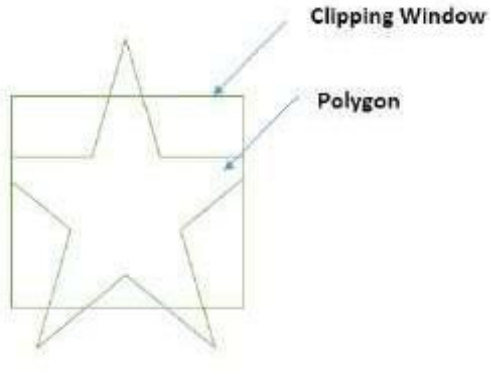


Figure: Clipping Right Edge

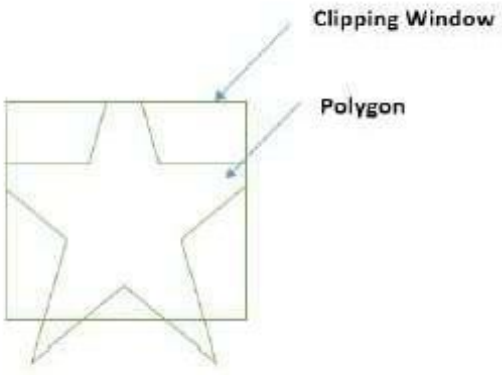


Figure: Clipping Top Edge

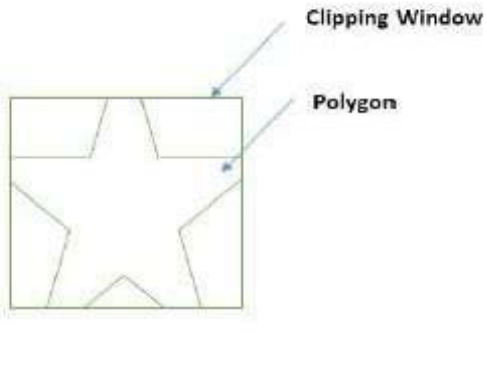


Figure: Clipping Bottom Edge

Algorithm

1) Read coordinates of all vertices of polygon.
2) Read coordinates of clipping window.
3) Consider left edge of window.
4) Compare vertices of each edge of polygon individually with clipping plane.
5) Save resulting intersection and vertices in new list of vertices according to 4 possible relationships between edge and clipping boundary:
   a) If 1st vertex of edge is outside window boundary and second vertex of edge is inside, then, intersection point of polygon edge with window boundary and second vertex are added to output vertex list.
   b) If both vertices of edge are inside window boundary, only second vertex is added to output vertex list.
   c) If 1st vertex of edge is inside window boundary and second vertex of edge is outside, then, only edge intersection with window boundary is added to output vertex list.
   d) If both vertices of edge are outside window boundary, nothing is added to output vertex list.
6) Repeat Steps 4 to 5 for remaining edges of clipping window. Each time successively pass the resultant list of vertices to process next edge of clipping window.
7) Stop.

Assignment No. - 06

Study of 2-D Transformation

➢ **Aim:** To understand and implement 2-D transformation.

1. Scaling
2. Rotation
3. Reflection

➢ **Theory :**

Scaling

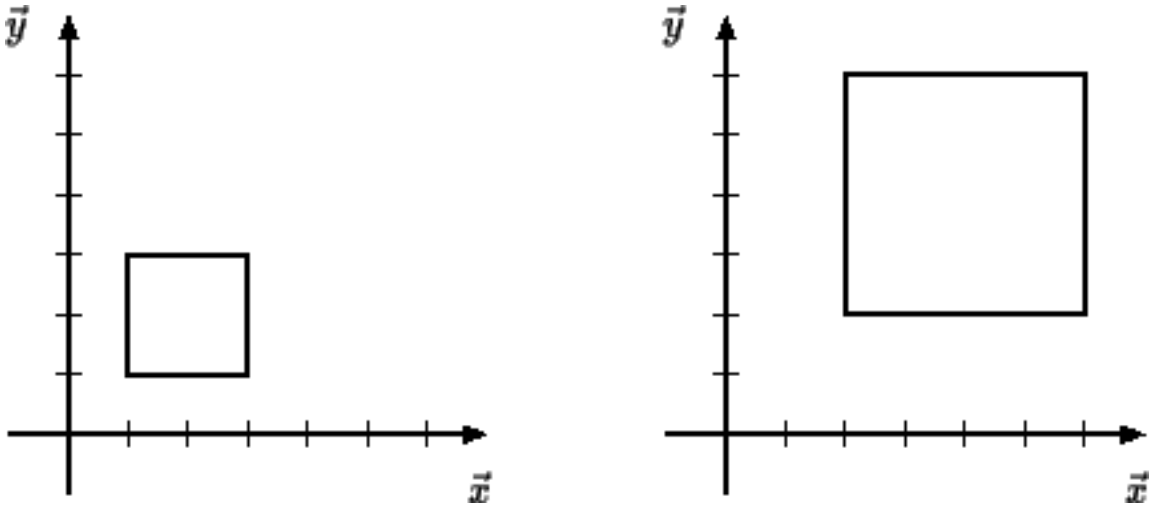
A scaling transformation alters size of an object. In the scaling process, we either compress or expand the dimension of the object.
Scaling operation can be achieved by multiplying each vertex coordinate (x, y) of the polygon by scaling factor sx and sy to produce the transformed coordinates as (x', y').
So,
x' = x * sx and
y' = y * sy.

The scaling factor sx, sy scales the object in X and Y direction respectively. So, the above equation can be represented in matrix form:

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix}$$

Or P' = S . P

Scaling process:



Algorithm:

1. Make 2 X 2 scaling matrix as :

Sx 0

0 Sy

2. For each point of the polygon.

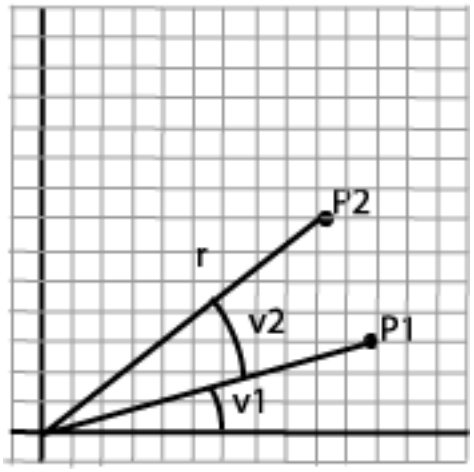
(i) Make 1X2 matrix P, where P[0][0] equals to x coordinate of the post and P[1][0] equals to y coordinate of the point.

(ii) Multiply scaling matrix S with point matrix P to get the new

coordinate. (iii)Draw the polygon using new coordinates.

Rotation

A two-dimensional rotation is applied to and object by repositioning it along a circular path in the xy plane. To generate a rotation, we specify a angular rotation theta and the position of the rotation point about which the object is to be rotated.



Rotation is more complicated to express and we have to use trigonometry to formulate it. $P_1=(x_1,y_1)=(r\cdot\cos(v_1),r\cdot\sin(v_1))$
$P_2=(x_2,y_2)=(r\cdot\cos(v_1+v_2),r\cdot\sin(v_1+v_2))$

We'll introduce the trigonometric formulas for the sum of two angles: $\sin(a+b) = \cos(a)\cdot\sin(b)+\sin(a)\cdot\cos(b)$
$\cos(a+b) = \cos(a)\cdot\cos(b)-\sin(a)\cdot\sin(b)$

and get:
$P_1=((r\cdot\cos(v_1), r\cdot\sin(v_1))$
$P_2=(r\cdot\cos(v_1)\cdot\cos(v_2)-r\cdot\sin(v_1)\cdot\sin(v_2), r\cdot\cos(v_1)\cdot\sin(v_2)+r\cdot\sin(v_1)\cdot\cos(v_2))$

We insert:
$x_1=r\cdot\cos(v_1)$
$y_1=r\cdot\sin(v_1)$
in P2's coordinates:

$P_2=(x_2, y_2)=(x1\cdot\cos(v_2)-y1\cdot\sin(v_2), x1\cdot\sin(v_2)+y1\cdot\cos(v_2))$
and we have expressed P2's coordinates with P1's coordinates and the rotation angle v. $x_2 = x_1\cdot\cos(v) - y_1\cdot\sin(v)$
$y_2 = x_1\cdot\sin(v) + y_1\cdot\cos(v)$

Notice: Rotation is expressed relative to origin. This also means that the sides in figures that are rotated create new angles with the axes after a rotation. We assume that the positive rotation angle is counterclockwise.

Algorithm:

1. Start
2. Initialize the graphics mode.
3. Construct a 2D object (use Drawpoly()) e.g. (x, y)
4. a. Get the Rotation angle

b. Rotate the object by the angle ϕ

x'=x cos ϕ - y sin

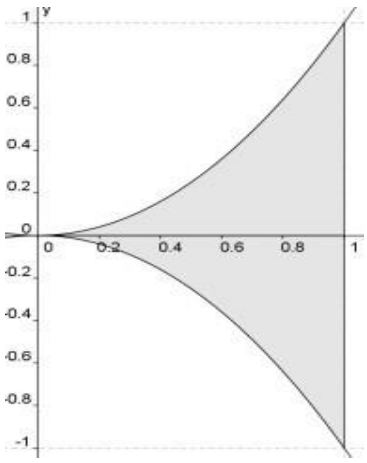
ϕ y'=x sin ϕ - y

cosϕ

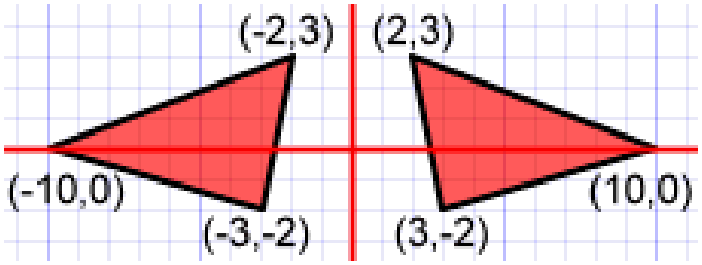
c. Plot (x', y')

Reflection

The mirror image of any image for 2D reflection is generated with respect to the "Axis of Reflection". For that we need to rotate main object 180 Degrees about the reflection axis. Following figures are shown for better understanding of Computer Graphics Reflection. In the above figure, it shows the half of the above image is reflected with respected to the X-axis. So there we may choose axis of reflection in the XY plane. One must remember that, when the reflection axis is a line in the XY plane, say X-axis, the rotation path about this X-axis is in a plane which is perpendicular to the XY plane. So Image gets reflected with respect to any axis say X- axis or Y-axis. Generally, the image is original plane is represented with the points say 1,2,3, whereas the reflected image is denoted with the same numbers with dashes say 1',2',3' and likewise.



As this image is reflecting with respect to the X-axis, the reflection transformation obviously keeps X-values same. But one must notice that, the image "Flips" 180 degrees and the values of Y of coordinate positions. And similarly when the image gets reflected with respect to Y-axis is shown figure. As mentioned in the above X-axis, case the point gets reflected with respect to Y- axis and obviously point gets flipped.



Types of Computer Graphics Reflection:

Transformation in Computer Graphics Reflection is broadly classified in to Two Categories. They are,

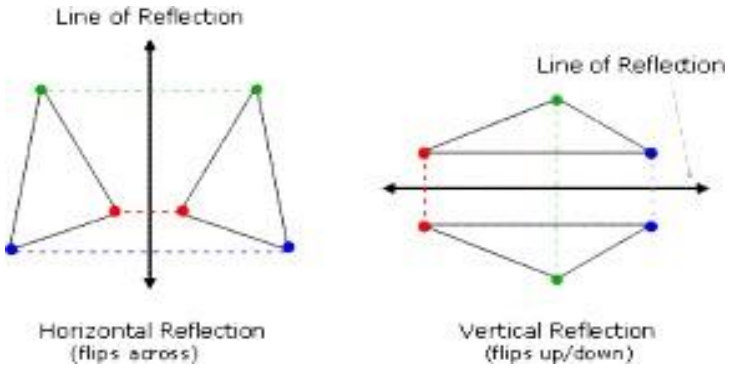
I. Horizontal Reflection
II. Vertical Reflection

I. Horizontal Reflection:

When Image gets flipped across, then the Image reflection is known as Horizontal Reflection. And here image gets reflected with respect to the Y-axis.

III. Vertical Reflection:

When Image gets flipped up and down, the reflection is referred as Vertical Reflection. For easy understanding, we are providing detailed image analysis, which show both Horizontal and Vertical Reflections.



Generally Reflection about any line in Computer Graphics is represented by any line,

y = mx + b

The line y = mx + b, can be achieved with a combination of translate-rotate-reflect transformation. In this several transformations Translation, Rotation and Reflection took place. For Instance, consider a line, so what do we do first? Yes first we translate the image, so that it generally moves towards one of its side from origin point. So then we rotate the same object to one of the coordinate axes and reflect it about that axis. Now we restore the object to its actual position with the inverse rotation and translation.

Assignment No. - 07

Study of 3-D Transformation

➢ **Aim:** To understand and implement 2-D transformation.

1. Rotation
2. Scaling
3. Shear

➢ **Theory:**

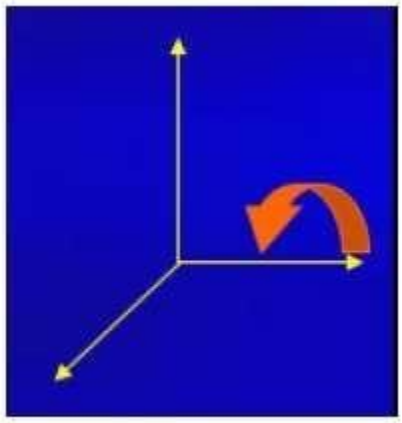
Rotation

3D rotation is not same as 2D rotation. In 3D rotation, we have to specify the angle of rotation along with the axis of rotation. We can perform 3D rotation about X, Y, and Z axes. They are represented in the matrix form as below −

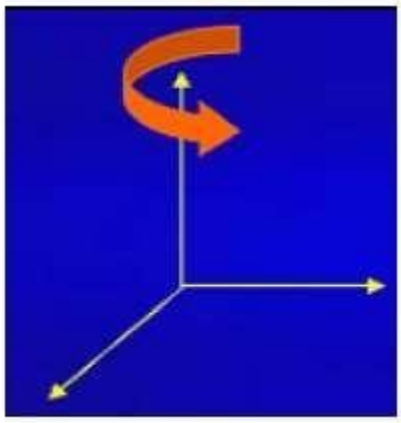
$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} R_y(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} R_z(\theta)$$

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

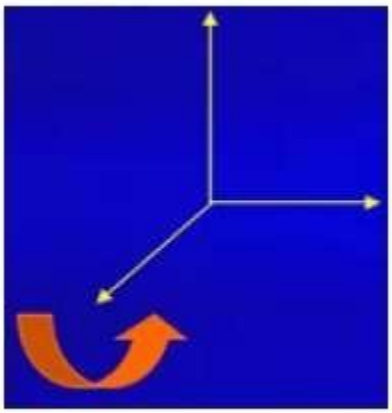
The following figure explains the rotation about various axes −



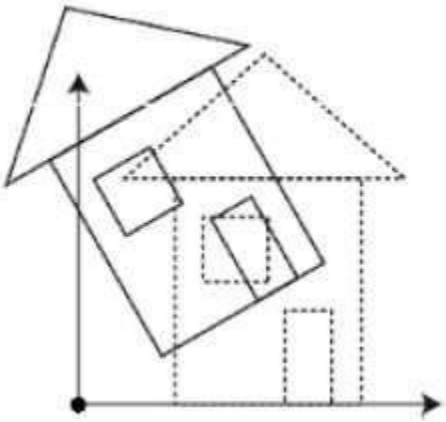
Rotation about x-axis



Rotation about y-axis

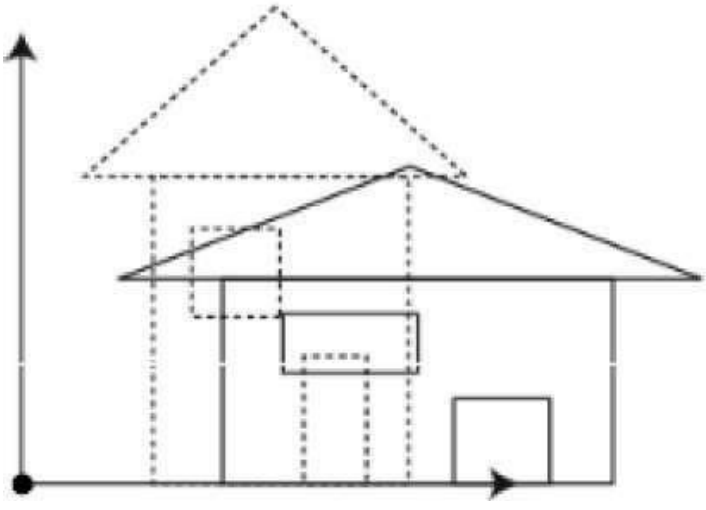


Rotation about z-axis



Scaling

You can change the size of an object using scaling transformation. 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. The following figure shows the effect of 3D scaling −



In 3D scaling operation, three coordinates are used. Let us assume that the original coordinates are (X, Y, Z), scaling factors are (SX,SY,Sz)(SX,SY,Sz) respectively, and the produced coordinates are (X', Y', Z'). This can be mathematically represented as shown below –

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

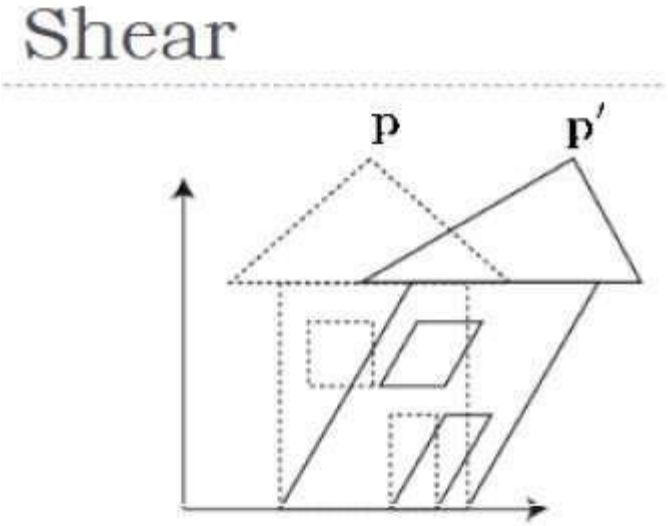
$$P' = P \cdot S$$

$$[X' \;\; Y' \;\; Z' \;\; 1] = [X \;\; Y \;\; Z \;\; 1] \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= [X.S_x \;\; Y.S_y \;\; Z.S_z \;\; 1]$$

Shear

A transformation that slants the shape of an object is called the **shear transformation**. Like in 2D shear, we can shear an object along the X-axis, Y-axis, or Z-axis in 3D.



As shown in the above figure, there is a coordinate P. You can shear it to get a new coordinate P', which can be represented in 3D matrix form as below −

$$Sh = \begin{bmatrix} 1 & sh_x^y & sh_x^z & 0 \\ sh_y^x & 1 & sh_y^z & 0 \\ sh_z^x & sh_z^y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P' = P \cdot Sh$$

$$X' = X + Sh_x^y Y + Sh_x^z Z$$

$$Y' = Sh_y^x X + Y + sh_y^z Z$$

$$Z' = Sh_z^x X + Sh_z^y Y + Z$$

Transformation Matrices

Transformation matrix is a basic tool for transformation. A matrix with n x m dimensions is multiplied with the coordinate of objects. Usually 3 x 3 or 4 x 4 matrices are used for transformation. For example, consider the following matrix for various operation.

$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$	$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$Sh = \begin{bmatrix} 1 & sh_x^y & sh_x^z & 0 \\ sh_y^x & 1 & sh_y^z & 0 \\ sh_z^x & sh_z^y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Translation Matrix	**Scaling Matrix**	**Shear Matrix**
$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$R_y(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$R_z(\theta) = \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Rotation Matrix		

Assignment No. - 08

Study of Hidden Surfaces and

➢ **Aim:** To understand and implement Painter's algorithm.

➢ **Theory:**

When we view a picture containing non-transparent objects and surfaces, then we cannot see those objects from view which are behind from objects closer to eye. We must remove these hidden surfaces to get a realistic screen image. The identification and removal of these surfaces is called Hidden-surface problem.

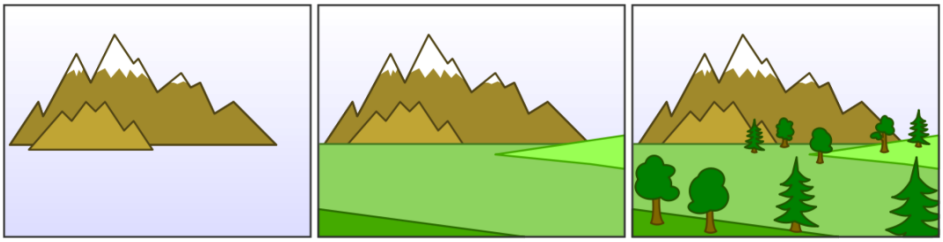
There are two approaches for removing hidden surface problems −Object-Space method and Image-space method. The Object-space method is implemented in physical coordinate system and image-space method is implemented in screen coordinate system.

When we want to display a 3D object on a 2D screen, we need to identify those parts of a screen that are visible from a chosen viewing position.

Painter's Algorithm

The painter's algorithm, also known as a priority fill, is one of the simplest solutions to the visibility problem in 3D computer graphics. When projecting a 3D scene onto a 2D plane, it is necessary at some point to decide which polygons are visible, and which are hidden.

The name "painter's algorithm" refers to the technique employed by many painters for painting distant parts of a scene before parts which are nearer thereby covering some areas of distant parts. The painter's algorithm sorts all the polygons in a scene by their depth and then paints them in this order, farthest to closest. It will paint over the parts that are normally not visible — thus solving the visibility problem — at the cost of having painted invisible areas of distant objects. The ordering used by the algorithm is called a '*depth order*', and does not have to respect the numerical distances to the parts of the scene: the essential property of this ordering is, rather, that if one object obscures part of another then the first object is painted after the object that it obscures. Thus, a valid ordering can be described as a topological ordering of a directed acyclic graph representing occlusions between objects.



The distant mountains are painted first, followed by the closer meadows; finally, the trees, are painted. Although some trees are more distant from the viewpoint than some parts of the meadows, the ordering (mountains, meadows, trees) forms a valid depth order, because no object in the ordering obscures any part of a later object.

Algorithm

1) Sort given polygons in order of decreasing depth.
2) Determine polygons in polygon list whose Z extents overlap that of A.
3) Perform Tests 2 to 6 for each B.
4) If B passes test, scan convert A.
5) If tests fail for B, swap A and B in list and make a note that B is swapped. If B has already been swapped use plane congaing A to divide B in to two polygons B1 and B2. Replace B with B1 and B2. Repeat Step 3.

Assignment No. - 09

Study of Animation

➢ **Aim:** To understand the concept of Animation.

➢ **Theory:**

Animation means giving life to any object in computer graphics. It has the power of injecting energy and emotions into the most seemingly inanimate objects. Computer-assisted animation and computer-generated animation are two categories of computer animation. It can be presented via film or video.

The basic idea behind animation is to play back the recorded images at the rates fast enough to fool the human eye into interpreting them as continuous motion. Animation can make a series of dead images come alive. Animation can be used in many areas like entertainment, computer aided-design, scientific visualization, training, education, e-commerce, and computer art.

Animators have invented and used a variety of different animation techniques. Basically, there are six animation technique which we would discuss one by one in this section.

Traditional Animation (frame by frame)

Traditionally most of the animation was done by hand. All the frames in an animation had to be drawn by hand. Since each second of animation requires 24 frames (film), the amount of efforts required to create even the shortest of movies can be tremendous.

Keyframing

In this technique, a storyboard is laid out and then the artists draw the major frames of the animation. Major frames are the ones in which prominent changes take place. They are the key points of animation. Keyframing requires that the animator specifies critical or key positions for the objects. The computer then automatically fills in the missing frames by smoothly interpolating between those positions.

Procedural

In a procedural animation, the objects are animated by a procedure −a set of rules −not by keyframing. The animator specifies rules and initial conditions and runs simulation. Rules are often based on physical rules of the real world expressed by mathematical equations.

Behavioral

In behavioral animation, an autonomous character determines its own actions, at least to a certain extent. This gives the character some ability to improvise, and frees the animator from the need to specify each detail of every character's motion.

Performance Based (Motion Capture)

Another technique is Motion Capture, in which magnetic or vision-based sensors record the actions of a human or animal object in three dimensions. A computer then uses these data to animate the object.

This technology has enabled a number of famous athletes to supply the actions for characters in sports video games. Motion capture is pretty popular with the animators mainly because some of the commonplace human actions can be captured with relative ease. However, there can be serious discrepancies between the shapes or dimensions of the subject and the graphical character and this may lead to problems of exact execution.

Physically Based (Dynamics)

Unlike key framing and motion picture, simulation uses the laws of physics to generate motion of pictures and other objects. Simulations can be easily used to produce slightly different sequences while maintaining physical realism. Secondly, real-time simulations allow a higher degree of interactivity where the real person can manoeuvre the actions of the simulated character.

In contrast the applications based on key-framing and motion select and modify motions form a pre-computed library of motions. One drawback that simulation suffers from is the expertise and time required to handcraft the appropriate controls systems.

Assignment No. - 10

Study of Curve

➢ **Aim:** To understand the concept of curve generation.

➢ **Theory :**

2-D Curve Generation

Curves are one of the most essential objects to create high-resolution graphics. While using many small polylines allows creating graphics that appear smooth at fixed resolutions, they do not preserve smoothness when scaled and also require a tremendous amount of storage for any high-resolution image. Curves can be stored much easier, can be scaled to any resolution without losing smoothness, and most importantly provide a much easier way to specify real- world objects.
All of the popular curves used in graphics are specified by parametric equations. Instead of specifying a function of the form y = f(x), the equations y = fy(u) and x = fx(u) are used. Using parametric equations allows curves that can double back and cross themselves, which are impossible to specify in a single equation in the y = f(x) case. Parametric equations are also easier to evaluate: changing u results in moving a fixed distance along the curve, while in the traditional equation form much work is needed to determine whether to step through x or y, and determining how large a step to take based on the slope.

B-SPLINE CURVES

B-splines correct the deficiencies of LeGrange interpolated curves by defining blending functions that vary each control point's control from 0 far away from the point to its maximum near the point. This can be done by making it so the curve does not pass through each control point, but instead just passes near them. The sum of the B-Spline blending functions also always sum to 1, and the slopes between curve segments are continuous. Finally, the final curve always lies within the convex hull of the control points.
While B-splines can be of higher order, usually cubic B-splines are used. Cubic functions allow for enough curve flexibility for most tasks, and are easier to work with then higher order functions since they behave more predictably (higher order functions may get unexpected wiggles and kinks). The cubic B-Spline blending functions work over four control points. The blending functions appear below. Note that special blending functions are needed for the first and last two sections of the B-Spline so that it passes through the first and last points.

First section:

B[1](u) = (1 - u)^3
B[2](u) = 21 u^3 / 12 - 9 u^2 / 2 + 3 u
B[3](u) = -11 u^3 / 12 + 3 u^2 / 2
B[4](u) = u^3 /

6 Second

section:

B[1](u) = (1 - u)^3 / 4
B[2](u) = 7 u^3 / 12 - 5 u^2 / 4 + u / 4 + 7
/ 12 B[3](u) = - u^3 / 2 + u^2 / 2 + u / 2 +
1 / 6 B[4](u) = u^3 / 6

Middle sections:

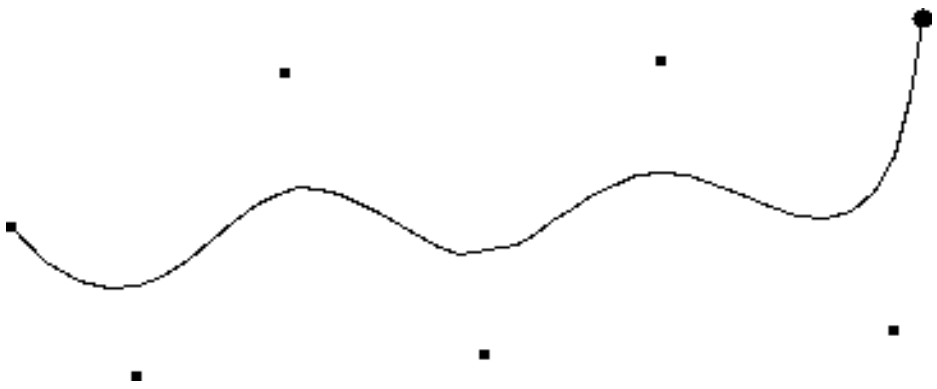
B[1](u) = (1 - u)^3 / 6
B[2](u) = u^3 / 2 - u^2 + 2 / 3
B[3](u) = - u^3 / 2 + u^2 / 2 + u / 2 + 1
/ 6 B[4](u) = u^3 / 6

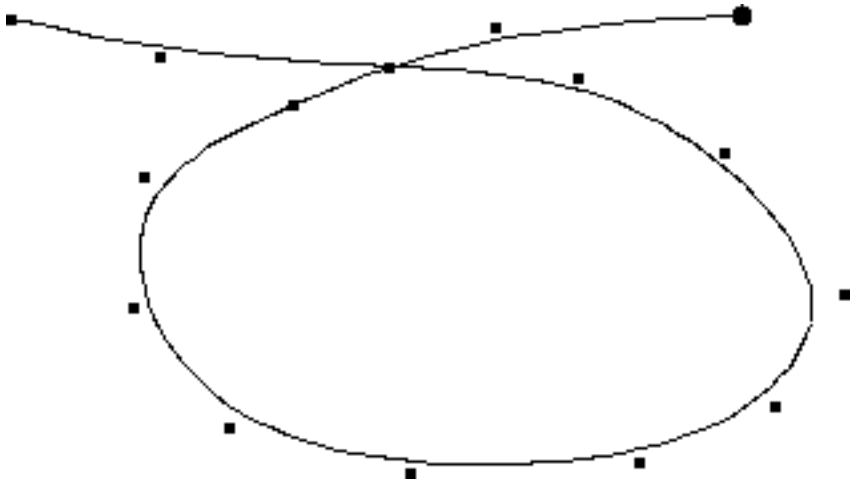
Second to last section:

B[i](u) = F[5 - i](1 - u) where F[i](u) is the ith blending function for the second

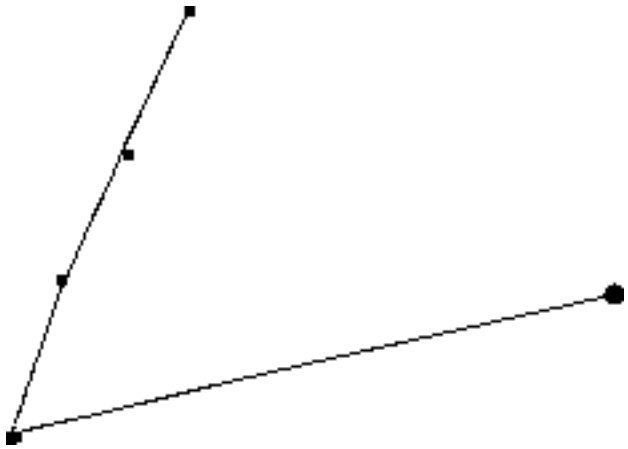
section. Last section:

B[i](u) = F[5 - i](1 - u) where F[i](u) is the ith blending function for the first section.





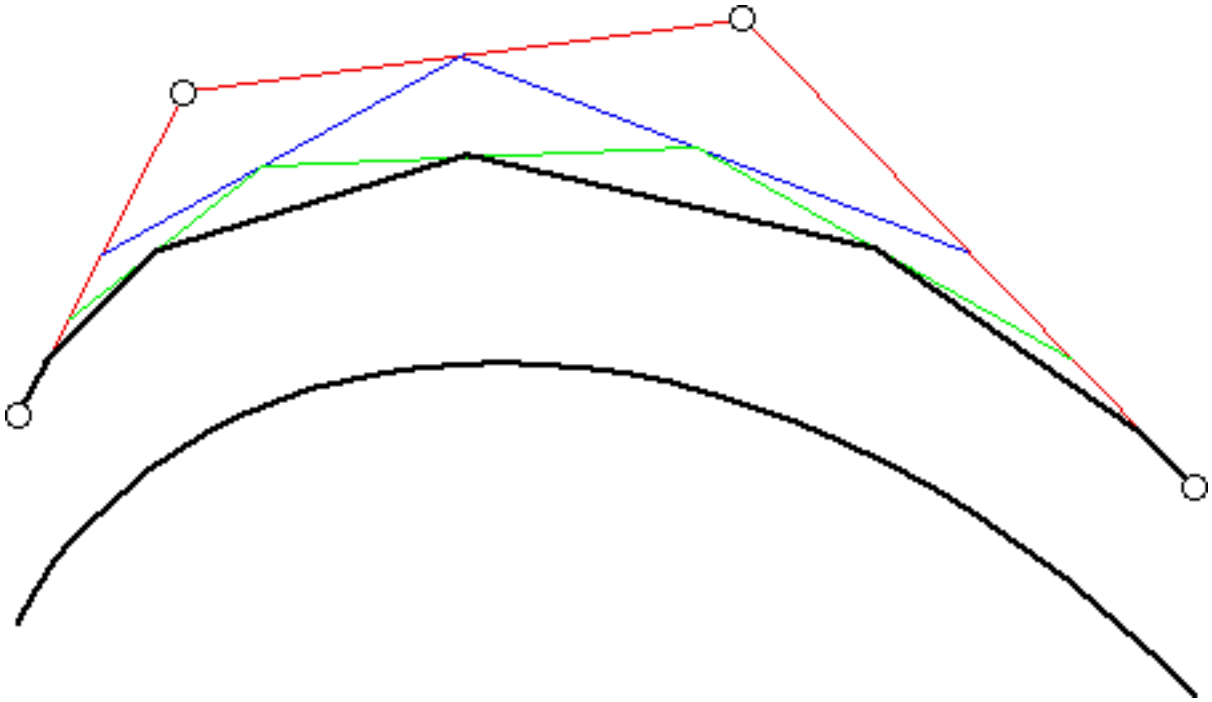
Note that B-Spines are designed to eliminate sharp corners, as seen above. A sharp corner can be obtained, however, by using duplicate sequential control points. Using three identical control points causes the curve to pass through the point, as illustrated below.



BEZIER CURVES

Bezier curves are another very popular curve type. Bezier curves, unlike B-splines, must have precisely n control points, where n + 1 is the degree of the Bezier polynomial. To create a longer curve, it is necessary to connect multiple Bezier curves. This is usually done by making the last control point of one curve the same as the first control point of the next curve. Bezier curves pass through the first and last control points of each curve segment, however, which makes them quite easy to work with and popular for use in interactive design programs. Bezier curves, like B-Spline curves, always lie within the convex hull of the control points, and always have the sum of the basis functions add to 1.

The idea behind Bezier curves is quite simple. To create a smooth curve, one intuitive mechanism is to first connect the four control points with lines. Then draw new lines connecting the midpoints of those lines, and more new lines to connect the midpoints of the new lines, and so forth until the resulting curve is sufficiently smooth. This is illustrated below.



While it is possible to define a curve type based on connecting the last third of one segment to the first third of the next, or the last fourth of one segment to the first fourth of the next, and so on, using midpoints allows for a very efficient subdivision algorithm to be derived.

The basis functions for cubic Bezier curves, which have the same results as the above subdivision process, appear below. These are based on Bernstein polynomials.

B[1](u) = (1 - u)^3
B[2](u) = 3 u (1 - u)^2
B[3](u) = 3 u^2 (1 -
u) B[4](u) = u^3

The DeCasteljau representation of Bezier curves is usually used, however. This is a recursive definition of the above polynomials that is based on the subdivision illustrated earlier.

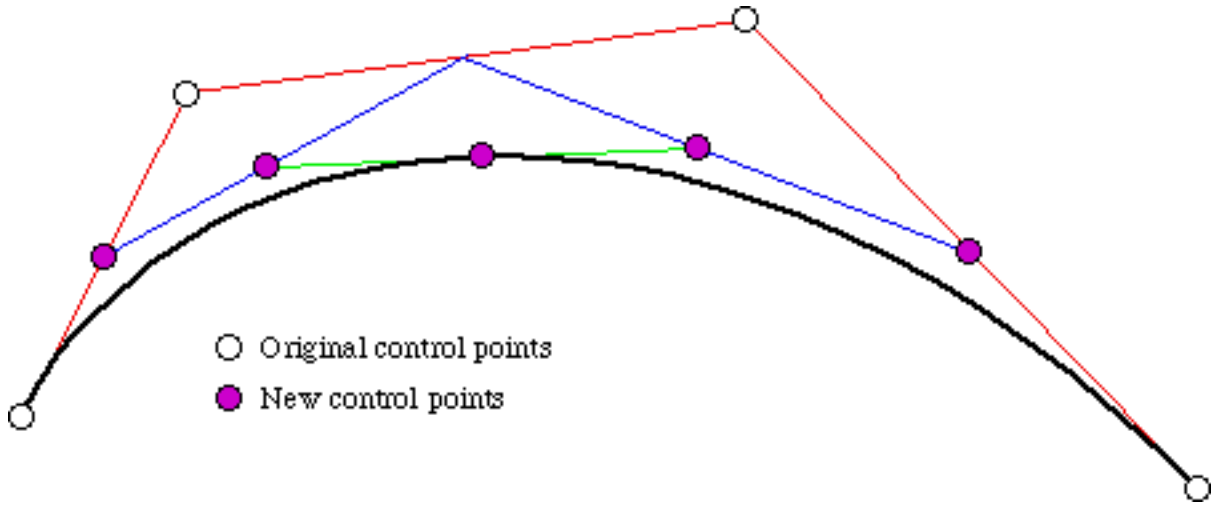
$$p_i^r(u) = (1 - u)\, p_i^{r-1}(u) + u\, p_{i+1}^{r-1}(u)$$

Where

p[1], p[2], . . ., p[n] are the control points

r = 1, 2, . . ., n
i = 0, 1, . . ., n - r
p[i](u) = p[i]

A point on the curve is given by p[0]^n (u).
Using this derivation, it is possible to see a simple recursive subdivision process for generating cubic Bezier curves. This works by taking the original Bezier curve and breaking it into two smaller Bezier curves, as illustrated below. These Bezier curves can then be broken into smaller Bezier curves, and so on, until the control points of the Bezier curve are very close to a straight line. When this happens, a straight line is drawn between the first and last control points, and the recursion pops back up.



At each stage of subdivision, the control points of the sub-curves are

calculated by: q[0] = p[0]
q[1] = (p[0] + p[1]) / 2
q[2] = (q[1] / 2) + (p[1] + p[2]) /
4 q[3] = (q[2] + r[1]) / 2

r[0] = q[3]
r[1] = (p[1] + p[2]) / 4 + (r[2] /
2) r[2] = (p[2] + p[3]) / 2
r[3] = p[3]
where p[1..4] are the control points of the original Bezier, and q[1..4] and r[1..4] are the control points of the sub-curves. This derivation was first done by Lane.
The linearity test for the above method is quite simple: all that needs to be done is calculate the sum of the distance from points p[1] and p[2] to the line connecting p[0] and p[4], and see if this is greater then some tolerance.

CURVES

All of the above can be applied to 3-D curves by including a third equation to specify z: fz(u) = sum from i = 1 to n of z[i] B[i](u)and using the same blending functions. The optimized version of the Bezier curve formulation, however, would need to be changed.

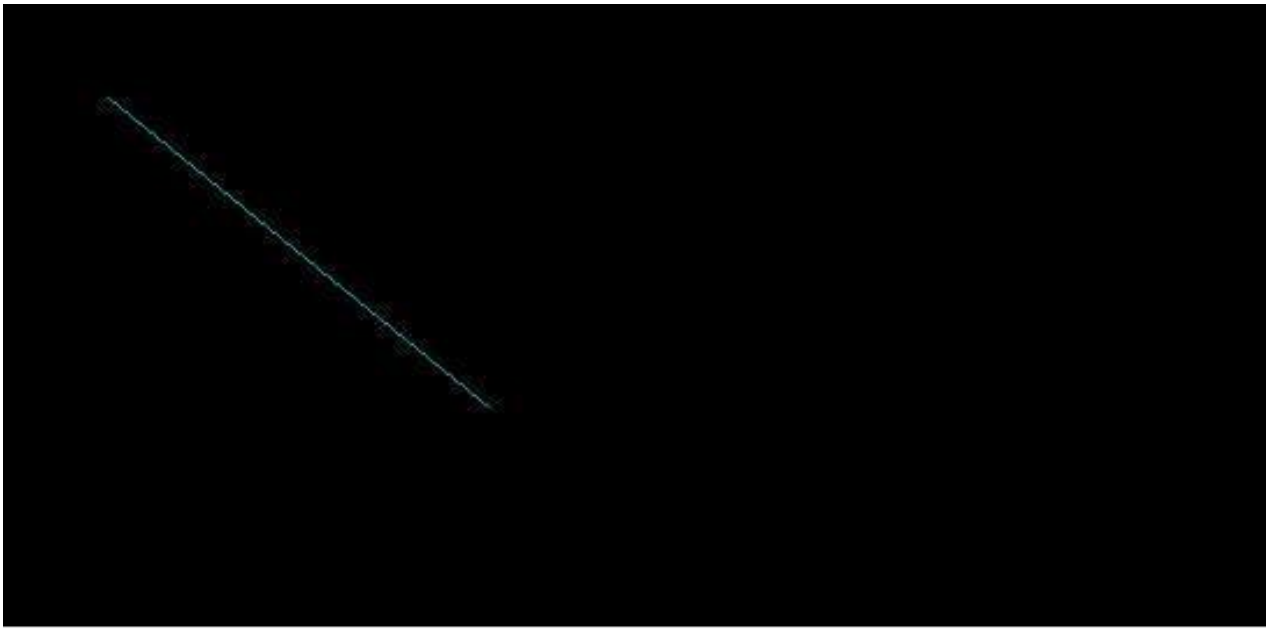
Sample Code Section

Program to Implement Bresenham alogrithm for line.

```
#include<conio.h>
#include<stdio.h>
#include<graphics.h>
#include<stdlib.h>
#include<math.h>

void main()
{
/*STEP 1:INITIALISATION*/
/*request for auto detection*/
int gdriver= DETECT,gmode,errorcode;

int x1,y1,x2,y2,i,s1,s2, x,y,e;

/*STEP 2: INPUT (x1,y1),(x2,y2)*/
printf("\nENTER VALUES FOR x1= ");
scanf("%d",&x1);

printf("\nENTER VALUES FOR x2= ");
scanf("%d",&x2);

printf("\nENTER VALUES FOR y1= ");
scanf("%d",&y1);

printf("\nENTER VALUES FOR y2= ");
scanf("%d",&y2);
/*initialize graphics local variable*/
initgraph(&gdriver,&gmode,"c:\\tcpp\\bgi");

/*read graphresult*/
errorcode=graphresult();

if(errorcode!=grOk)
 {
   printf("Graphics error: %s\n",grapherrormsg(errorcode));
   printf("Press any key to continue:");
   getch();
   exit(1);
 }
 /*STEP 3: LENGTH EVUALATION*/

  s1=abs(x2-x1);
```

```
  s2=abs(y2-y1);

/*STEP 4: */
  e=2*s2-s1;


/*STEP 6:LOOP TO PUTPIXEL*/

for(i=1;i<=s1;i++)
   {
    putpixel(x,y,3);
    while(e>=0)
    {
     y=y+1;
     e=e-2*s1;
    }
     x=x+1;
     e=e+2*s2;
    }
/*STEP: 7 STOP*/
 getch();
  closegraph();
 }
```

Expected Output :

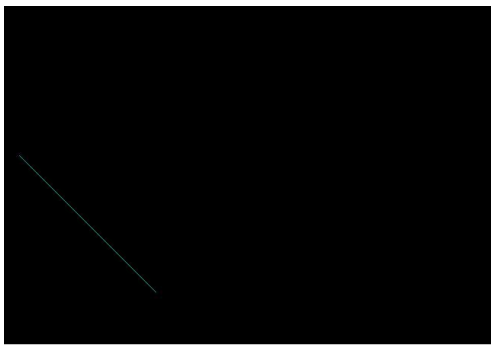


Program to Implement DDA alogrithm for line.

```
#include<conio.h>
#include<stdio.h>
#include<graphics.h>
#include<stdlib.h>
#include<math.h>

void main()
{
/*STEP 1:INITIALISATION*/
/*request for auto detection*/
int gdriver= DETECT,gmode,errorcode;

int x1,y1,x2,y2,length,i,s1,s2,filterx,filtery;
float x,y,dx,dy;

/*STEP 2: INPUT (x1,y1),(x2,y2)*/
printf("\nENTER VALUES FOR x1= ");
scanf("%d",&x1);

printf("\nENTER VALUES FOR x2= ");
scanf("%d",&x2);

printf("\nENTER VALUES FOR y1= ");
scanf("%d",&y1);

printf("\nENTER VALUES FOR y2= ");
scanf("%d",&y2);
/*initialize graphics local variable*/
initgraph(&gdriver,&gmode,"c:\\tcpp\\bgi");

/*read graphresult*/
errorcode=graphresult();

if(errorcode!=grOk)
 {
   printf("Graphics error: %s\n",grapherrormsg(errorcode));
   printf("Press any key to continue:");
   getch();
   exit(1);
 }
 /*STEP 3: LENGTH EVUALATION*/

  s1=abs(x2-x1);
  s2=abs(y2-y1);
```

```
 if(s1>=s2)
 length=s1;
 else
   length=s2;

/*STEP 4: */
  dx=1.0*(x2-x1)/length;
  dy=1.0*(y2-y1)/length;

 /*STEP 5:INCREMENTATION*/
 s1=x2-x1;

 s2=y2-y1;

 if(s1<0)
   x=x1-0.5;
 else
   x=x1+0.5;

 if(s2<0)
   y=y1-0.5;
 else
   y=y1+0.5;
/*STEP 6:LOOP TO PUTPIXEL*/

for(i=1;i<=length;i++)
   {
    filterx=x;
    filtery=y;
    putpixel(filterx,filtery,3);
    x=x+dx;
    y=y+dy;
   }
/*STEP: 7 STOP*/
 getch();
  closegraph();
 }
```

Expected Output:

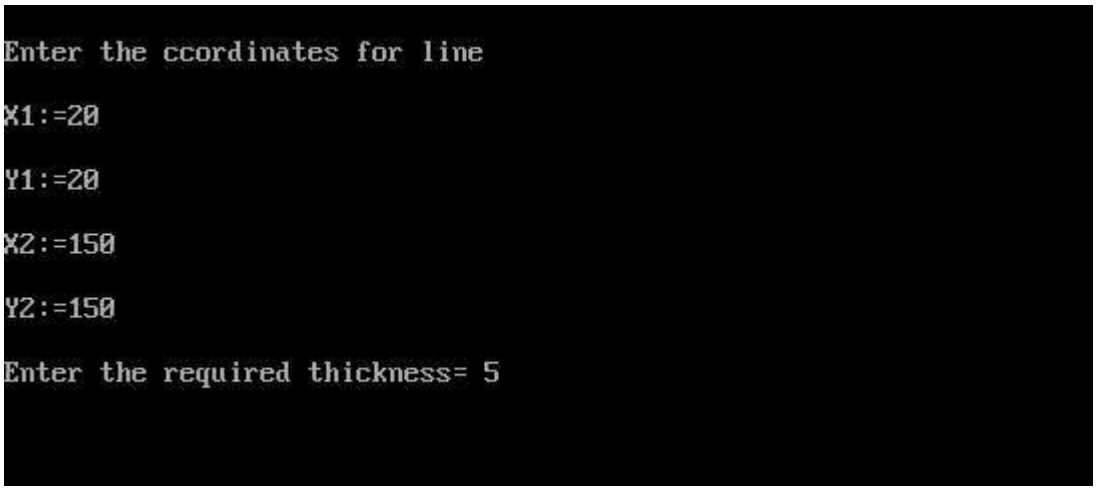


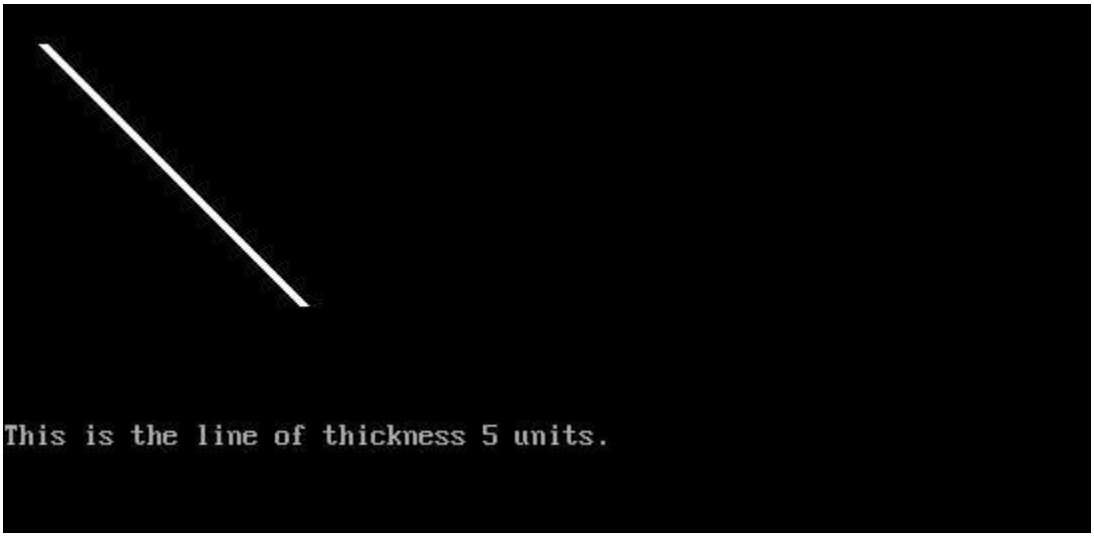
Program to Implement Thickness Of The Line.

```
#include<conio.h>
#include<stdio.h>
#include<graphics.h>
#include<math.h>

void main()
{
int gd,i,gm,thickness;
float wx,wy,x1,y1,x2,y2;
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"d:\\tcpp\\bgi");
printf("\nEnter the ccordinates for line\n");
printf("\nX1:=");
scanf("%f",&x1);
printf("\nY1:=");
scanf("%f",&y1);
printf("\nX2:=");
scanf("%f",&x2);
printf("\nY2:=");
scanf("%f",&y2);

/*ENTER THE THICKNESS OF THE LINE ----*/
printf("\nEnter the required thickness= ");
scanf("%d",&thickness);
cleardevice();
line(x1,y1,x2,y2);
if((y2-y1)/(x2-x1)<1)
 {
 wy=(thickness-1)*sqrt(pow((x2-x1),2)+pow((y2-y1),2))/(2*fabs(x2-x1));
 for(i=0;i<wy;i++)
   {
   line(x1,y1-i,x2,y2-i);
   line(x1,y1+i,x2,y2+i);
   }
 }
 else
 {
 wx=(thickness-1)*sqrt(pow((x2-x1),2)+pow((y2-y1),2))/(2*fabs(y2-y1));
 for(i=0;i<wx;i++)
 {
  line(x1-i,y1,x2-i,y2);
  line(x1+i,y1,x2+i,y2);
 }
}
printf("\nThis is the line of thickness %d units.\n",thickness);
getch();
```

```
closegraph();
}
```

Expected Output:



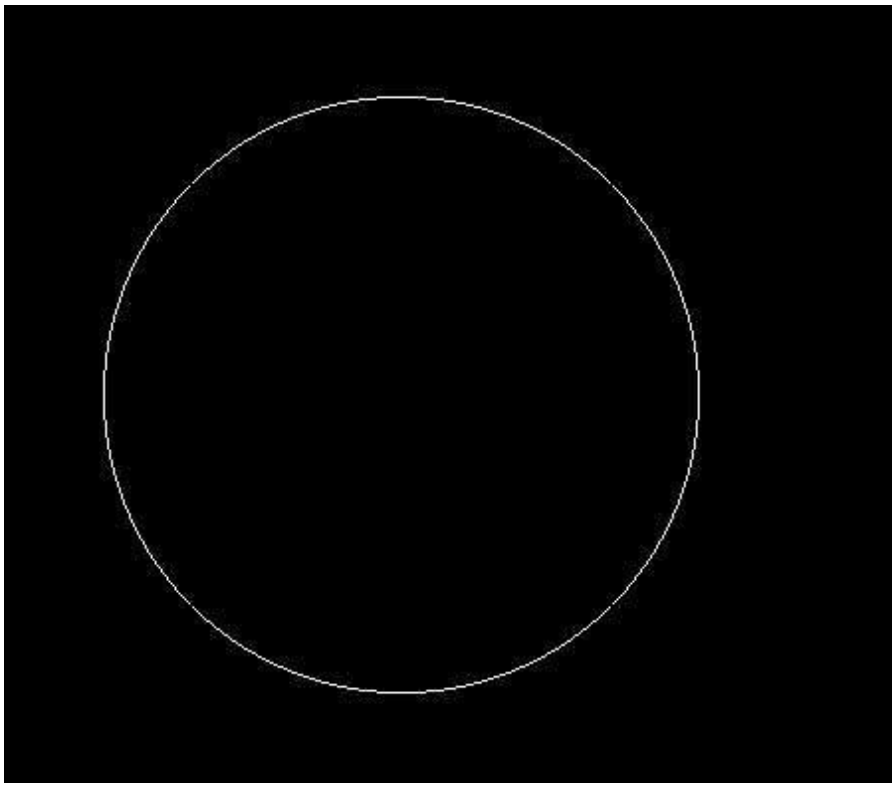


Program to Draw a Circle.

```
#include<conio.h>
#include<stdio.h>
#include<graphics.h>
#include<dos.h>
#include<math.h>

void main()
{
float d;
int gd,gm,x,y;
int r;
clrscr();

/*Read the radius of circle*/
printf("\nEnter the radius of circle= ");
scanf("%d",&r);

/*Initilize graphics mode*/
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"d:\\tcpp\\bgi");

/*Initialise starting point*/
x=0;
y=r;

/*initialise the decision variable*/
d=3-2*r;

do
{
  putpixel(200+x,200+y,15);
  putpixel(200+y,200+x,15);
  putpixel(200+y,200-x,15);
  putpixel(200+x,200-y,15);
  putpixel(200-x,200-y,15);
  putpixel(200-y,200-x,15);
  putpixel(200-y,200+x,15);
  putpixel(200-x,200+y,15);
  if(d<0)
   {
      d=d+4*x+6;
   }
    else
    {
      d=d+4*(x-y)+10;
      y=y-1;
```

```
    }
      x=x+1;
      delay(100);
    }
    while(x<y);
    getch();
    closegraph();
}
```

Expected Output:

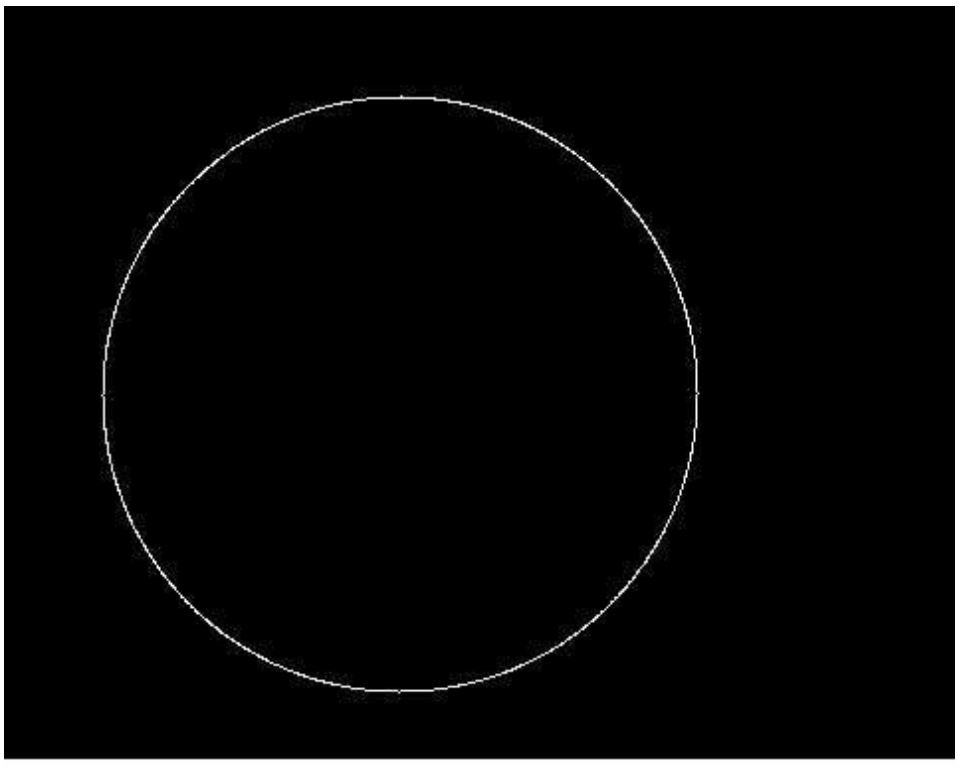


Circle Drawing Using DDA Algorithm.

```
#include<conio.h>
#include<stdio.h>
#include<graphics.h>
#include<math.h>
#include<stdlib.h>
#include<dos.h>
void main()
{

float x1,y1,x2,y2,startx,starty,epsilon;
int i,val,r;
int gdriver=DETECT,gmode,errorcode;

clrscr();

/*read two end points of line*/
printf("\nEnter the radius ofa circle :");
scanf("%d",&r);

/*INITIALIZE GRAPHICS MODE*/
initgraph(&gdriver,&gmode,"d:\\tcpp\\bgi");
errorcode=graphresult();

if(errorcode!=grOk)
 {
  printf("\ngraphics error:",grapherrormsg(errorcode));
  printf("\npress any key to continue:");
  getch();
  exit(1);
 }
/*INITALIZE STARTING POINT*/
x1=r*cos(0);
y1=r*sin(0);
startx=x1;
starty=y1;

/*Calculation for epsilon*/
i=0;
do
{
  val=pow(2,i);
  i++;
}
while(val<r);
epsilon=1/pow(2,i-1);
do
```

```
{
 x2=x1+y1*epsilon;
 y2=y1-epsilon*x2;
 putpixel(200+x2,200+y2,15);


/*Reinitilize the current point*/
x1=x2;
y1=y2;
delay(100);
}
while((y1-starty)<epsilon||(startx-x1)>epsilon);
getch();
closegraph();
}
```

Expected Output:



/* Algorithms for Polygon Filling
1. Flood Fill Algorithm 2. Scan Line Algorithm.*/

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<graphics.h>
# include <dos.h>
struct edge
{
 int x1,y1,x2,y2;
 int flag;
};
struct edge ed[10],temped;
float dx,dy,m[10],x_int[10],inter_x[10];
int x[10],y[10],ymax=0,ymin=480,yy,temp;
int ch,n,i,j,k;

void flood(int x,int y)
{
 rectangle(50,50,100,100);
 if (getpixel(x,y)==BLACK)
 {
   putpixel(x,y,BLUE);
   flood(x+1,y);
   flood(x-1,y);
   flood(x,y+1);
   flood(x,y-1);
   flood(x+1,y+1);
   flood(x-1,y-1);
   flood(x+1,y-1);
   flood(x-1,y+1);
 }
 getch();
}

void scan_line()
{
 clrscr();

 printf("\n Enter the limit for Vertices :: ");
 scanf("%d",&n);

 printf("\n Enter the vertices :: ");
 for (i=0;i<n;i++)
 {
  printf(" x[%d] :: ",i);
  scanf("%d",&x[i]);
```

```
 printf(" y[%d] :: ",i);
 scanf("%d",&y[i]);
 if (y[i]>ymax)
 ymax=y[i];
 if (y[i]<ymin)
 ymin=y[i];
 ed[i].x1=x[i];
 ed[i].y1=y[i];
}
clrscr();
for (i=0;i<n-1;i++)
{
 ed[i].x2=ed[i+1].x1;
 ed[i].y2=ed[i+1].y1;
 ed[i].flag=0;
}
 ed[i].x2=ed[0].x1;
 ed[i].y2=ed[0].y1;
 ed[i].flag=0;

for (i=0;i<n;i++)
{
 if (ed[i].y1<ed[i].y2)
 {
  temp=ed[i].x1;
  ed[i].x1=ed[i].x2;
  ed[i].x2=temp;
  temp=ed[i].y1;
  ed[i].y1=ed[i].y2;
  ed[i].y2=temp;
 }
}

for (i=0;i<n;i++)
{
 line(ed[i].x1,ed[i].y1,ed[i].x2,ed[i].y2);
}

for (i=0;i<n-1;i++)
{
 for (j=0;j<n-1;j++)
 {
  if (ed[j].y1<ed[j+1].y1)
  {
   temped=ed[j];
   ed[j]=ed[j+1];
   ed[j+1]=temped;
  }
```

```
 if (ed[j].y1==ed[j+1].y1)
 {
  if (ed[j].y2<ed[j+1].y2)
  {
   temped=ed[j];
   ed[j]=ed[j+1];
   ed[j+1]=temped;
  }
  if (ed[j].y2==ed[j+1].y2)
  {
   if (ed[j].x1<ed[j+1].x1)
   {
   temped=ed[j];
   ed[j]=ed[j+1];
   ed[j+1]=temped;
   }
  }
 }
}
}

for (i=0;i<n;i++)
{
 dx=ed[i].x2-ed[i].x1;
 dy=ed[i].y2-ed[i].y1;
 if (dy==0)
  m[i]=0;
 else
  m[i]=dx/dy;
 inter_x[i]=ed[i].x1;
}
yy=ymax;
while(yy>ymin)
{
 for (i=0;i<n;i++)
 {
  if (yy>ed[i].y2 && yy<=ed[i].y1)
   ed[i].flag=1;
  else
   ed[i].flag=0;
 }

j=0;
for (i=0;i<n;i++)
{
 if (ed[i].flag==1)
 {
  if (yy==ed[i].y1)
```

```
  {
   x_int[j]=ed[i].x1;
   j++;
   if(ed[i-1].y1==yy && ed[i-1].y1<yy)
   {
    x_int[j]=ed[i].x1;
    j++;
   }
   if (ed[i+1].y1==yy && ed[i+1].y1<yy)
   {
    x_int[j]=ed[i].x1;
    j++;
   }
  }
  else
  {
   x_int[j]=inter_x[i]+(-m[i]);
   inter_x[i]=x_int[j];
   j++;
  }
 }
}
for (i=0;i<j;i++)
{
 for (k=0;k<j-1;k++)
 {
  if (x_int[k]>x_int[k+1])
  {
   temp=x_int[k];
   x_int[k]=x_int[k+1];
   x_int[k+1]=temp;
  }
 }
}

for (i=0;i<j;i+=2)
{
 line (x_int[i],yy,x_int[i+1],yy);
}
yy--;
delay(50);
}
getch();

}
int main()
{
 int gdriver = DETECT, gmode;
```

```
initgraph(&gdriver, &gmode, "e:\\tc\\bgi");
do
{
 printf("\n\n *****MAIN MENU***** ");
 printf("\n\n 1. FLOOD FILL ALGORITHM ");
 printf("\n\n 2. SCAN LINE ALGORITHM ");
 printf("\n\n 3. EXIT ");
 printf("\n\n Enter your choice of operation :: ");
 scanf("%d",&ch);

 switch(ch)
 {
  case 1: flood(55,55);
                                        break;
  case 2: scan_line();
                                        break;
  case 3: break;
 }
}while(ch!=3);
getch();
closegraph();
return 0;
}
```

Expected Output:

##PROGRAM TO IMPLEMENT SUTHERLAND

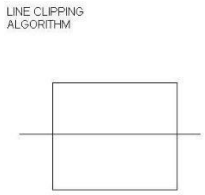
```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<dos.h>
#include<math.h>
#include<graphics.h>

/*Defining structure for end point of line*/
typedef struct coordinate
{
int x,y;
char code[4];
}PT;
void drawwindow();
void drawline(PT p1,PT p2,int cl);
coordinate setcode(PT p);
int visibility(PT p1,PT p2);
PT resetendpt(PT p1,PT p2);
void main()
{
 int gd=DETECT,gm,v;
 PT p1,p2,ptemp;
 initgraph(&gd,&gm,"e:\\tc\\bgi");
 cleardevice();
 printf("\n\n\t\tEnter end-point 1(x,y): ");
 scanf("%d",&p1.x);
 scanf("%d",&p1.y);
 printf("\n\n\t\tEnter end-point 2(x,y): ");
 scanf("%d",&p2.x);
 scanf("%d",&p2.y);
 cleardevice();
 drawwindow();
 getch();
 drawline(p1,p2,15);
 getch();
 p1=setcode(p1);
 p2=setcode(p2);
 v=visibility(p1,p2);
 switch(v)
 {
case 0:            /*line completely visible*/
     cleardevice();
     drawwindow();
     drawline(p1,p2,15);
     break;
case 1:            /*line completely invisible*/
```

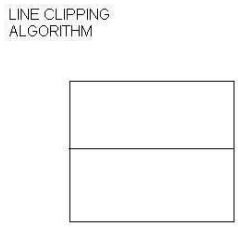
```
        cleardevice();
        drawwindow();
        break;
case 2:             /*line partly visible*/
        cleardevice();
        p1=resetendpt(p1,p2);
        p2=resetendpt(p2,p1);
        drawwindow();
        drawline(p1,p2,15);
        break;
}
getch();
closegraph();
}
/*function to draw window*/

void drawwindow()
{
setcolor(15);
line(150,100,450,100);
line(450,100,450,350);
line(450,350,150,350);
line(150,350,150,100);
}
/*function to draw line between two points---*/
void drawline(PT p1,PT p2,int cl)
{
setcolor(cl);
line(p1.x,p1.y,p2.x,p2.y);
}
/*function to set code of the coordinate --- */
PT setcode(PT p)
{
PT ptemp;
if(p.y<100)
ptemp.code[0]='1';/*top*/
else
ptemp.code[0]='0';
if(p.y>350)
ptemp.code[1]='1';/*bottom*/
else
ptemp.code[1]='0';
if(p.x>450)
ptemp.code[2]='1';/*right*/
else
ptemp.code[2]='0';
if(p.x<150)
ptemp.code[3]='1';/*left*/
```

```
else
ptemp.code[3]='0';
ptemp.x=p.x;
ptemp.y=p.y;
return(ptemp);
}
/*function to determine visibility of line---*/
int visibility(PT p1,PT p2)
{
int i,flag=0;
for(i=0;i<4;i++)
{
if((p1.code[i]!='0')||(p2.code[i]!='0'))
  flag=1;
}
if(flag==0)
return (0);
for(i=0;i<4;i++)
{
if((p1.code[i]==p2.code[i])&&(p1.code[i]=='1'))
flag=0;
}
if(flag==0)
return(1);
return(2);
}
/*function to find new end points---*/
PT resetendpt(PT p1,PT p2)
{
PT temp;
int x,y,i;
float m,k;
if(p1.code[3]=='1')/*cutting left edge*/
x=150;
if(p1.code[2]=='1')/*cutting right edge*/
x=450;
if((p1.code[3]=='1')||(p1.code[2]=='1'))
{
m=(float)(p2.y-p1.y) /(p2.x-p1.x);
k=(p1.y+(m*(x-p1.x)));
temp.y=k;
temp.x=x;
for(i=0;i<4;i++)
temp.code[i]=p1.code[i];
if(temp.y<=350&&temp.y>=100)
return(temp);
}
if(p1.code[0]=='1')/*cutting top edge*/
```

```
y=100;
if(p1.code[1]=='1')/*cutting bottom edge*/
y=350;
if((p1.code[0]=='1')||(p1.code[1]=='1'))
{
m=(float)(p2.y-p1.y)/(p2.x-p1.x);
k=(float)p1.x+(float)(y-p1.y)/m;
temp.x=k;
temp.y=y;
for(i=0;i<4;i++)
temp.code[i]=p1.code[i];
return(temp);
}
else
return(p1);
}
```

Expected Output:





```
#include<conio.h>
#include<graphics.h>
#include<math.h>
#include<iostream.h>
typedef struct
{
 float x;
 float y;
}point;
int n;
int polyy(int,int);
void main()
{
 void polyy(point*,int);
 void leftclip(point,point*,point*);
 void rightclip(point,point*,point*);
 void topclip(point,point*,point*);
 void bottomclip(point,point*,point*);

 int i,j,k=0;
 point d,p1,p2,p[20],pi1,pi2,pp[20];
 initgraph(&k,&k,"c:\\tcpp\\bgi");
 cout<<"\nENTER CO-ORDINATES (left,top) :: ";
 cin>>p1.x>>p1.y;

 cout<<"\nENTER CO-ORDINATES (right,bottom) :: ";
 cin>>p2.x>>p2.y;

 cout<<"\nENTER NO. OF VERTEX :: ";
 cin>>n;

 cout<<"\nENTER THE CO-ORDINATES OF POLYGON :: ";
 for(i=0;i<n;i++)
 {
  cout<<"\nEnter co-ordinates of vertex "<<i+1<<" :: ";
  cin>>p[i].x>>p[i].y;
 }
 p[i].x=p[0].x;
 p[i].y=p[0].y;
 cleardevice();

 polyy(p,n);
 getch();
 rectangle(p1.x,p1.y,p2.x,p2.y);
 leftclip(p1,p,pp);
 rightclip(p2,p,pp);
 topclip(p1,p,pp);
 bottomclip(p2,p,pp);
```

```
 getch();
 clearviewport();
 rectangle(p1.x,p1.y,p2.x,p2.y);
 polyy(p,n);
 getch();
}
void leftclip(point p1,point* p,point* pp)
{
 int i,j=0;
 for(i=0;i<n;i++)
 {
  if(p[i].x<p1.x && p[i+1].x >=p1.x)
  {
   if(p[i+1].x-p[i].x!=0)
   {
    pp[j].y=(p[i+1].y-p[i].y)/(p[i+1].x-p[i].x)*(p1.x-p[i].x)+p[i].y;
   }
   else
    pp[j].y=p[i].y;

   pp[j].x=p1.x;
   j++;
   pp[j].x=p[i+1].x;
   pp[j].y=p[i+1].y;
   j++;
  }
  if(p[i].x>p1.x && p[i+1].x >=p1.x)
  {
   pp[j].y=p[i+1].y;
   pp[j].x=p[i+1].x;
   j++;
  }
  if(p[i].x>p1.x && p[i+1].x <=p1.x)
  {
   if(p[i+1].x-p[i].x!=0)
    pp[j].y=(p[i+1].y-p[i].y)/(p[i+1].x-p[i].x)*(p1.x-p[i].x)+p[i].y;
   else
   {
    pp[j].y=p[i].y;}
    pp[j].x=p1.x;
    j++;
   }
  }
 for(i=0;i<j;i++)
 {
  p[i].x=pp[i].x;
  p[i].y=pp[i].y;
 }
```

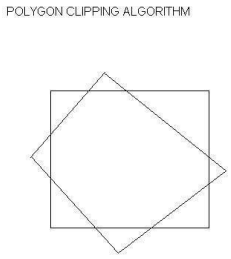
```
 p[i].x=pp[0].x;
 p[i].y=pp[0].y;
 n=j;
}


void rightclip(point p2,point* p,point* pp)
{
 int i,j=0;
 for(i=0;i<n;i++)
 {
  if(p[i].x>p2.x && p[i+1].x <=p2.x)
  {
   if(p[i+1].x-p[i].x!=0)
    pp[j].y=(p[i+1].y-p[i].y)/(p[i+1].x-p[i].x)*(p2.x-p[i].x)+p[i].y;
   else
     pp[j].y=p[i].y;

   pp[j].x=p2.x;
   j++;
   pp[j].x=p[i+1].x;
   pp[j].y=p[i+1].y;
   j++;
  }
  if(p[i].x<p2.x && p[i+1].x <=p2.x)
  {
   pp[j].y=p[i+1].y;
   pp[j].x=p[i+1].x;
   j++;
  }
  if(p[i].x<p2.x && p[i+1].x >=p2.x)
  {
   if(p[i+1].x-p[i].x!=0)
    pp[j].y=(p[i+1].y-p[i].y)/(p[i+1].x-p[i].x)*(p2.x-p[i].x)+p[i].y;
   else
    pp[j].y=p[i].y;

   pp[j].x=p2.x;
   j++;
  }
 }
 for(i=0;i<j;i++)
 {
  p[i].x=pp[i].x;
  p[i].y=pp[i].y;
 }
 p[i].x=pp[0].x;
 p[i].y=pp[0].y;
```

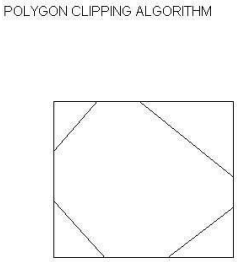
```
 n=j;
}

void topclip(point p1,point* p,point* pp)
{
 int i,j=0;
 for(i=0;i<n;i++)
 {
 if(p[i].y<p1.y && p[i+1].y >=p1.y)
 {
  if(p[i+1].y-p[i].y!=0)
   pp[j].x=(p[i+1].x-p[i].x)/(p[i+1].y-p[i].y)*(p1.y-p[i].y)+p[i].x;
  else
     pp[j].x=p[i].x;

  pp[j].y=p1.y;
  j++;
  pp[j].x=p[i+1].x;
  pp[j].y=p[i+1].y;
  j++;
 }
 if(p[i].y>p1.y && p[i+1].y >=p1.y)
 {
  pp[j].y=p[i+1].y;
  pp[j].x=p[i+1].x;
  j++;
 }
 if(p[i].y>p1.y && p[i+1].y <=p1.y)
 {
  if(p[i+1].y-p[i].y!=0)
   pp[j].x=(p[i+1].x-p[i].x)/(p[i+1].y-p[i].y)*(p1.y-p[i].y)+p[i].x;
  else
   pp[j].x=p[i].x;

  pp[j].y=p1.y;
  j++;
 }
}
for(i=0;i<j;i++)
{
 p[i].x=pp[i].x;
 p[i].y=pp[i].y;
}
p[i].x=pp[0].x;
p[i].y=pp[0].y;
n=j;
}
```

```
void bottomclip(point p2,point* p,point* pp)
{
 int i,j=0;
 for(i=0;i<n;i++)
 {
  if(p[i].y>p2.y && p[i+1].y <=p2.y)
  {
   if(p[i+1].y-p[i].y!=0)
   { pp[j].x=(p[i+1].x-p[i].x)/(p[i+1].y-p[i].y)*(p2.y-p[i].y)+p[i].x;
   }
   else
     pp[j].x=p[i].x;

   pp[j].y=p2.y;
   j++;
   pp[j].x=p[i+1].x;
   pp[j].y=p[i+1].y;
   j++;
  }
  if(p[i].y<p2.y && p[i+1].y <=p2.y)
  {
   pp[j].y=p[i+1].y;
   pp[j].x=p[i+1].x;
   j++;
  }
  if(p[i].y<p2.y && p[i+1].y >=p2.y)
  {
   if(p[i+1].y-p[i].y!=0)
   { pp[j].x=(p[i+1].x-p[i].x)/(p[i+1].y-p[i].y)*(p2.y-p[i].y)+p[i].x;
   }
   else
   {pp[j].x=p[i].x;}

   pp[j].y=p2.y;
   j++;
  }
 }
 for(i=0;i<j;i++)
 {
  p[i].x=pp[i].x;
  p[i].y=pp[i].y;
 }
 p[i].x=pp[0].x;
 p[i].y=pp[0].y;
 n=j;
}
void polyy(point x[20],int n)
{
```

```
 int i;
 for(i=0;i<n-1;i++)
 {
  line(x[i].x,x[i].y,x[i+1].x,x[i+1].y);
 }
 line(x[i].x,x[i].y,x[0].x,x[0].y);
}
```

Expected Output:





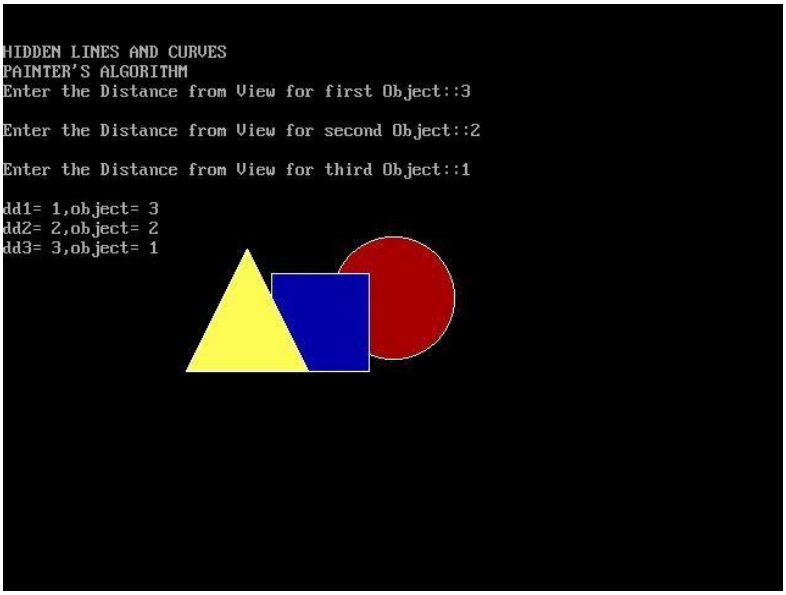
/* Program to implement PAINTER's algorithm
***/**

```
#include<conio.h>
#include<stdio.h>
#include<GRAPHICS.H>
#include<IOSTREAM.H>

int d1,d2,d3,i,j,k,dd2[2],dd1[2],dd3[2];

void circ()
{
circle(320,240,50);
setfillstyle(SOLID_FILL,RED);
fillellipse(320,240,50,50);
}

void triangle()
{
int poly[6];
line(200,200,150,300);
line(200,200,250,300);
line(150,300,250,300);
setfillstyle(SOLID_FILL,YELLOW);
poly[0]=200;poly[1]=200;poly[2]=150;poly[3]=300;
poly[4]=250;poly[5]=300;
fillpoly(3,poly);
}

void square()
{
int poly[8];
line(220,220,300,220);
line(220,220,220,300);
line(300,220,300,300);
line(300,300,220,300);
setfillstyle(SOLID_FILL,BLUE);
poly[0]=220;poly[1]=220;poly[2]=300;poly[3]=220;
poly[4]=300;poly[5]=300;poly[6]=220;poly[7]=300;
fillpoly(4,poly);
}

int sort(int d1,int d2,int d3)
{
if(d1>d2 && d1>d3)
  {
  dd3[2]=d1;
  dd3[1]=1;
```

```
 if(d2>d3)
 {
  dd2[2]=d2;dd2[1]=2;dd1[2]=d3;dd1[1]=3;
 }
 else
 {
 dd2[2]=d3;dd2[1]=3;dd1[2]=d2;dd1[1]=2;
 }
 }
 else if(d2>d1 && d2>d3)
                                          {
                                           dd3[2]=d2;dd3[1]=2;
                                           if(d1>d3)
                                            {
                                             dd2[2]=d1;dd2[1]=1;dd1[2]=d3;dd1[1]=3;
                                            }
                                            else
                                            {
                                            dd2[2]=d3;dd2[1]=3;dd1[2]=d1;dd1[1]=1;
                                            }
                                          }
                                          else if(d3>d2 && d3>d1)
                                          {
                                           dd3[2]=d3;dd3[1]=3;
                                           if(d2>d1)
                                            {
                                            dd2[2]=d2;dd2[1]=2;dd1[2]=d1;dd1[1]=1;
                                            }
                                            else
                                            {
                                            dd2[2]=d1;dd2[1]=1;dd1[2]=d2;dd1[1]=2;
                                            }
                                          }
                                          return 0;
}

int main()
{
int gd=DETECT,gm;
initgraph(&gd,&gm,"e:\\tcpp\\bgi");
cout<<"\n\nHIDDEN LINES AND CURVES";
cout<<"\nPAINTER'S ALGORITHM";
cout<<"\nEnter the Distance from View for first Object::";
cin>>d1;
cout<<"\nEnter the Distance from View for second Object::";
cin>>d2;
cout<<"\nEnter the Distance from View for third Object::";
cin>>d3;
```

```
sort(d1,d2,d3);
cout<<"\ndd1= "<<dd1[2]<<",object= "<<dd1[1];
cout<<"\ndd2= "<<dd2[2]<<",object= "<<dd2[1];
cout<<"\ndd3= "<<dd3[2]<<",object= "<<dd3[1];

if(dd3[1]==1)
  circ();
else if(dd3[1]==2)
                                        square();
    else
                                        triangle();

if(dd2[1]==1)
  circ();
else if(dd2[1]==2)
                                        square();
    else
                                        triangle();
if(dd1[1]==1)
  circ();
else if(dd1[1]==2)
                                        square();
    else
                                        triangle();

getch();
closegraph();
return 0;
}
```

Expected Output:

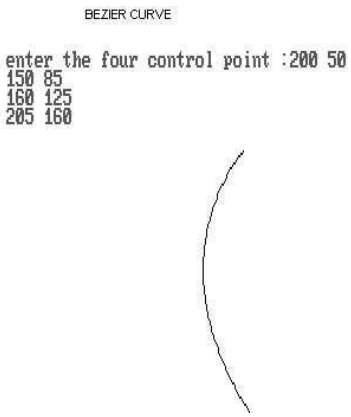


/*PROGRAM TO IMPLEMENT BEZIER CURVE

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>

void bizer(float x1,float y1,float x2,float y2,float x3,float y3,float x4,float y4)
{ float xm1,xm2,xm3,xm4,xm5,xm6;
  float ym1,ym2,ym3,ym4,ym5,ym6;

  if(abs(x1-x2)>1||abs(y1-y2)>1||abs(x2-x3)>1||abs(y2-y3)>1||abs(x3-x4)>1||abs(y3-y4)>1)
                                          { xm1=(x1+x2)/2;
                                            xm2=(x2+x3)/2;
                                            xm3=(x3+x4)/2;
                                            xm4=(xm1+xm2)/2;
                                            xm5=(xm2+xm3)/2;
                                            xm6=(xm4+xm5)/2;
                                            ym1=(y1+y2)/2;
                                            ym2=(y2+y3)/2;
                                            ym3=(y3+y4)/2;
                                            ym4=(ym1+ym2)/2;
                                            ym5=(ym2+ym3)/2;
                                            ym6=(ym4+ym5)/2;
                                           bizer(x1,y1,xm1,ym1,xm4,ym4,xm6,ym6);
                                           bizer(x4,y4,xm3,ym3,xm5,ym5,xm6,ym6);
                                          }
  else
                                          { line(x1,y1,x2,y2);
                                            line(x2,y2,x3,y3);
                                            line(x3,y3,x4,y4);
                                          }
}

void main()
{ int gd=DETECT,gm,i,bclr;
  float x[4],y[4];

  initgraph(&gd,&gm,"e:\\tc\\bgi");
  bclr=getpixel(getmaxx(),getmaxy());
  line(50,50,50,250);
    line(50,50,100,50);
    line(100,50,100,100);
    line(100,100,50,100);
    setcolor(bclr);
    line(50,50,100,50);
    line(100,50,100,100);
```

```
  line(100,100,50,100);
 for(i=0;i<10;i++)
 { setcolor(6);
  if(i%2==0)
  {
  bizer(50,50,60,55,70,55,75,50);
  bizer(75,50,80,45,90,45,100,50);
  }
  else
  {
  bizer(50,50,60,45,70,45,75,50);
  bizer(75,50,80,55,90,55,100,50);
  }
  delay(1000);
  setcolor(bclr);
  if(i%2==0)
  {
  bizer(50,50,60,55,70,55,75,50);
  bizer(75,50,80,45,90,45,100,50);
  }
  else
  {
  bizer(50,50,60,45,70,45,75,50);
  bizer(75,50,80,55,90,55,100,50);
  }
 }


  getch();
 closegraph();
}
```

Expected Output:

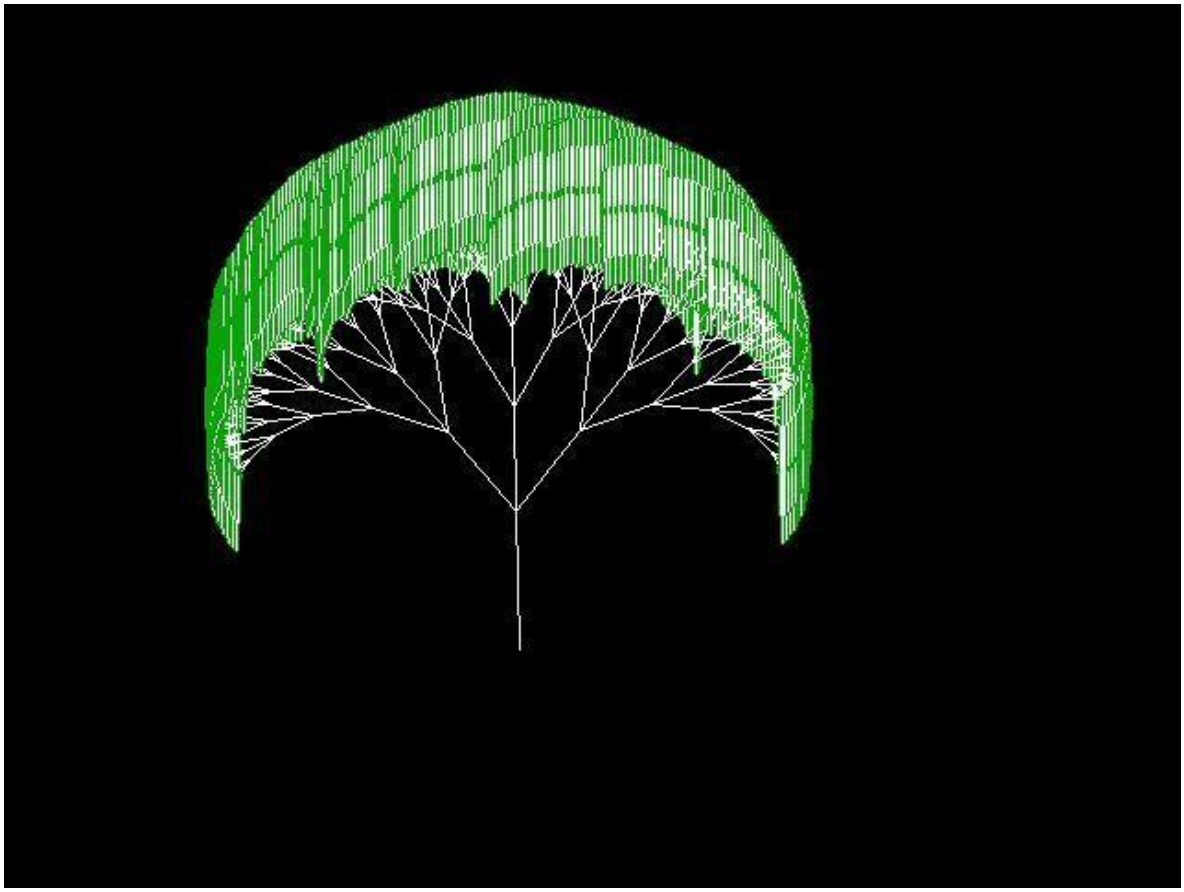


/*PROGRAM TO IMPLEMENT FRACTAL */

```
#include<stdio.h>
#include<dos.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
#define radian 0.0174
int x=280,y=350,numbranch=3;
float spreadratio=0.8,lengthratio=0.75;
 void  main()
 {
 int gd=DETECT,gm;
 int drawtree(int x1,int y1,float a,float l,float f,int n);
 initgraph(&gd,&gm,"e:\\tc\\bgi");
 drawtree(x,y,270,75,80,7);
 getch();
 restorecrtmode();
 }
 int drawtree(int x1,int y1,float a,float l,float f,int n)
 {
 int i,num,x2,y2;
 float delang,ang;
 if(n>0)
{
x2=x1+l*cos(radian*a);
y2=y1+l*sin(radian*a);
setcolor(WHITE);
getch();
line(x1,y1,x2,y2);
num=numbranch;

if(num>1)
 delang=f/(num-1.0);
 else
 delang=0.0;
 ang=a-f/2.0-delang;
 for(i=1;i<=num;i++)
 {
 ang+=delang;
// delay(10);
 drawtree(x2,y2,ang,l*lengthratio,f*spreadratio,n-1);
 }
 }
 else
 {
 getch();
setcolor(17+1);
```

```
 ellipse(x2,y2,0,190,3,4);
// delay(10);
fillellipse(x2,y2,2,40);
}
 }
```

Expected Output:



2D Transformation

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>

float midx,midy,n,obj[10][3]={0},mat[10][3]={0},tra[3][3]={0};

void window()
{
  setcolor(2);
 line(0,midy,midx*2,midy);
 line(midx,0,midx,midy*2);
}

void draw()
{
 setcolor(3);
 for(int i=0;i<n-1;i++)
 {
  line(obj[i][0]+midx,midy-obj[i][1],obj[i+1][0]+midx,midy-obj[i+1][1]);
 }
  line(obj[i][0]+midx,midy-obj[i][1],obj[0][0]+midx,midy-obj[0][1]);
}

 void final()
{
 setcolor(4);
 for(int i=0;i<n-1;i++)
 {
  line(mat[i][0]+midx,midy-mat[i][1],mat[i+1][0]+midx,midy-mat[i+1][1]);
 }
  line(mat[i][0]+midx,midy-mat[i][1],mat[0][0]+midx,midy-mat[0][1]);
}



 void trans()
 {
  int i,j,k;
  float tx,ty;
  cout<<"\n\nInput the translation vector(tx,ty):";
  cin>>tx>>ty;
  for(i=0;i<3;i++)
                                                for(j=0;j<3;j++)
                                                  {
```

```
                                        if(i==j)
                                                tra[i][j]=1;
                                        else
                                                tra[i][j]=0;
                                        }
 tra[2][0]=tx;
 tra[2][1]=ty;

 for(i=0;i<n;i++)
  for(j=0;j<3;j++)
                                       {
                                        mat[i][j]=0;
                                        for(k=0;k<3;k++)
                                         {
                                          mat[i][j]=mat[i][j]+obj[i][k]*tra[k][j];
                                         }
                                       }
 cleardevice();
 window();
 draw();
 final();

}

 void scale()
 {
  int i,j,k;
  float sx,sy;
  cout<<"\n\nInput the scaling vector(Sx,Sy):";
  cin>>sx>>sy;
  for(i=0;i<3;i++)
                                       for(j=0;j<3;j++)
                                        tra[i][j]=0;

   tra[0][0]=sx;
   tra[1][1]=sy;
   tra[2][2]=1;

  for(i=0;i<n;i++)
   for(j=0;j<3;j++)
                                       {
                                        mat[i][j]=0;
                                        for(k=0;k<3;k++)
                                         {
                                          mat[i][j]=mat[i][j]+obj[i][k]*tra[k][j];
                                         }
                                       }
  cleardevice();
```

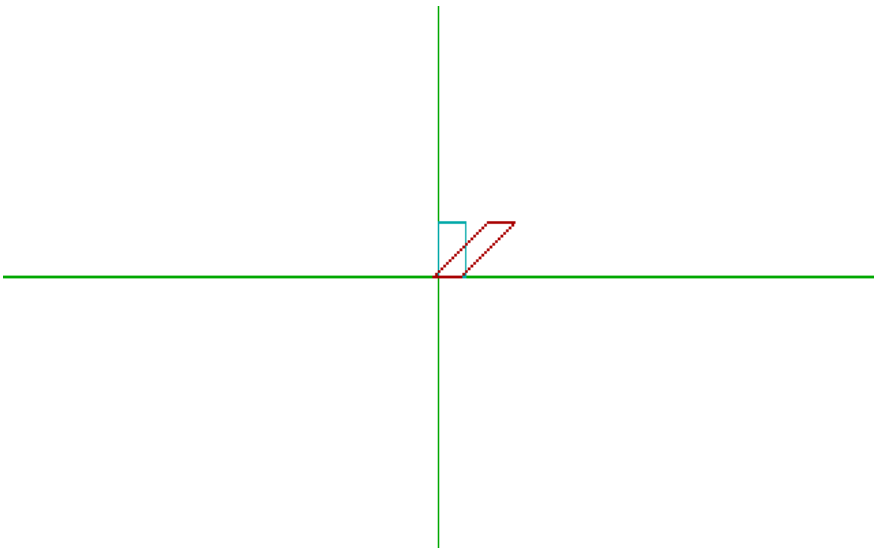
```
 window();
 draw();
 final();

}

 void rot()
 {
 int i,j,k;
 float theta;
 cout<<"\n\nInput the angle of rotation :";
 cin>>theta;
 theta=(3.14/180)*theta;
 for(i=0;i<3;i++)
                                        for(j=0;j<3;j++)
                                         tra[i][j]=0;

  tra[1][1]=tra[0][0]=cos(theta);
  tra[0][1]=sin(theta);
  tra[1][0]=-sin(theta);
  tra[2][0]=-midx*cos(theta)+midy*sin(theta)+midx;
  tra[2][1]=-midx*sin(theta)-midy*cos(theta)+midy;
  tra[2][2]=1;
 for(i=0;i<n;i++)
  for(j=0;j<3;j++)
                                        {
                                         mat[i][j]=0;
                                         for(k=0;k<3;k++)
                                          {
                                           mat[i][j]=mat[i][j]+obj[i][k]*tra[k][j];
                                          }
                                        }
 cleardevice();
 window();
 draw();
 final();

}

 void xshear()
 {
 int i,j,k;
 float shx,yref;
 cout<<"\n\nInput the X-Shear and Yref:";
 cin>>shx>>yref;
 for(i=0;i<3;i++)
                                        for(j=0;j<3;j++)
                                         {
```

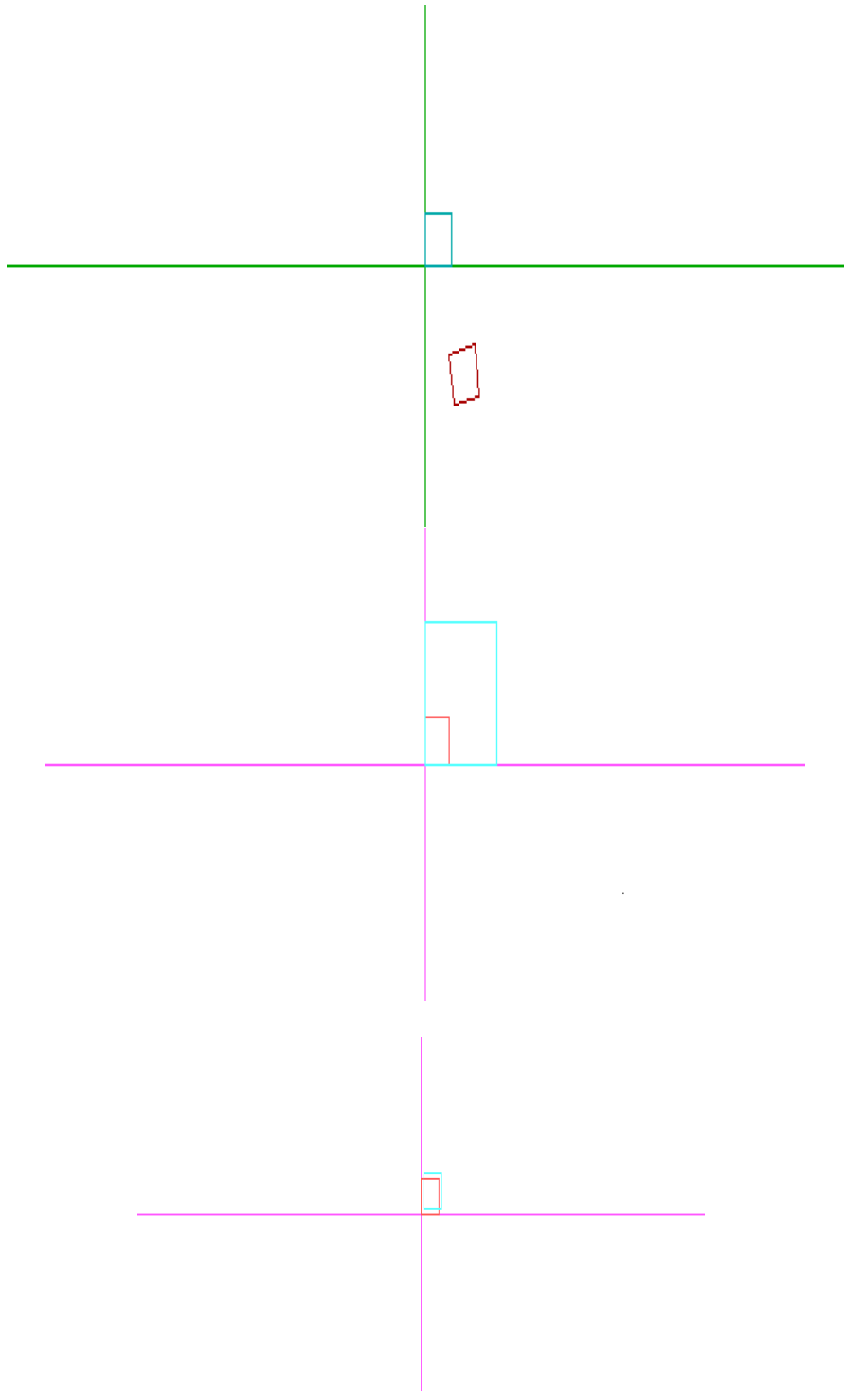
```
                                        if (i==j)
                                                tra[i][j]=1;
                                        else
                                                tra[i][j]=0;
                                        }

  tra[1][0]=shx;
  tra[2][0]=-1*shx*yref;

 for(i=0;i<n;i++)
  for(j=0;j<3;j++)
                                        {
                                         mat[i][j]=0;
                                         for(k=0;k<3;k++)
                                          {
                                           mat[i][j]=mat[i][j]+obj[i][k]*tra[k][j];
                                          }
                                        }
 cleardevice();
 window();
 draw();
 final();

}

 void yshear()
 {
 int i,j,k;
 float shy,xref;
 cout<<"\n\nInput the Y-Shear and Xref:";
 cin>>shy>>xref;
 for(i=0;i<3;i++)
                                        for(j=0;j<3;j++)
                                         {
                                         if (i==j)
                                                tra[i][j]=1;
                                         else
                                                tra[i][j]=0;
                                         }

  tra[0][1]=shy;
  tra[2][1]=-1*shy*xref;

 for(i=0;i<n;i++)
  for(j=0;j<3;j++)
                                        {
                                         mat[i][j]=0;
                                         for(k=0;k<3;k++)
```

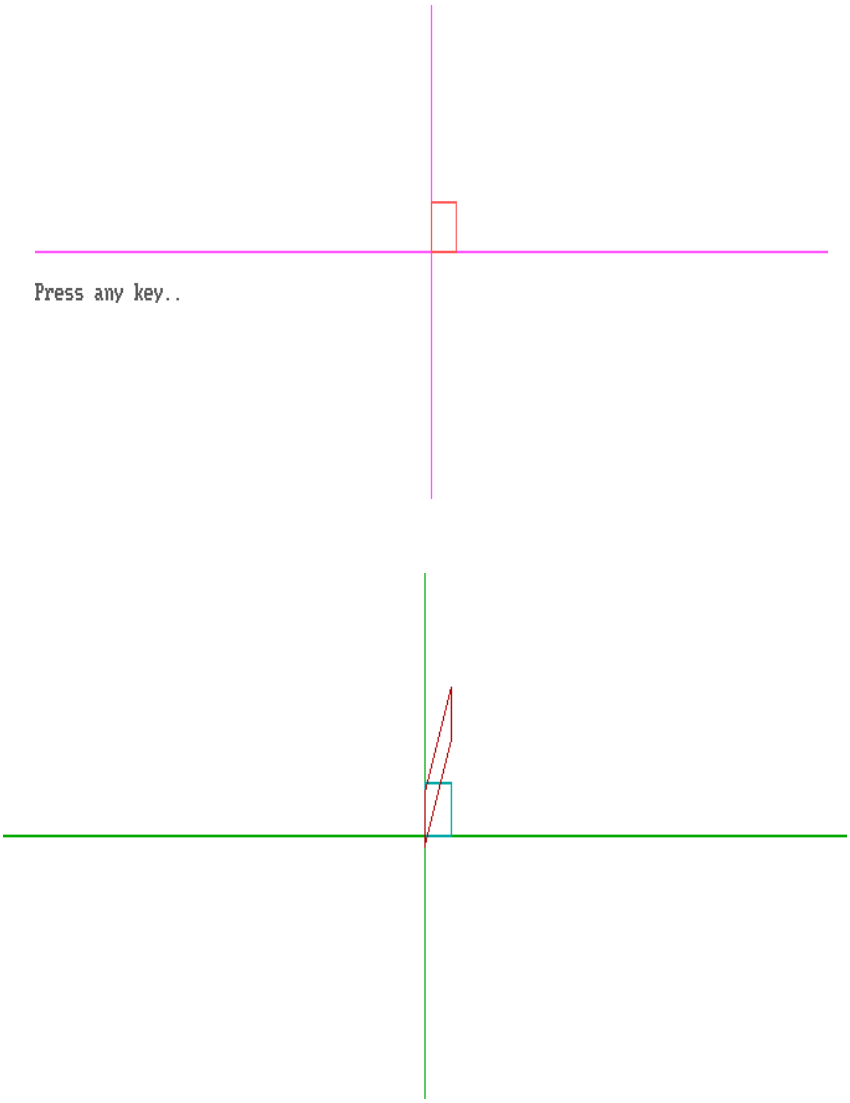
```
                                        {
                                         mat[i][j]=mat[i][j]+obj[i][k]*tra[k][j];
                                        }
                                       }
 cleardevice();
 window();
 draw();
 final();

}

void main()
{
 int gd=DETECT,gm,ch;
 initgraph(&gd,&gm,"");
 midx=getmaxx()/2;midy=getmaxy()/2;
 cout<<"\nInput the no. of vertices";
 cin>>n;
 cout<<"\nInput the vertices..\n";
 for(int i=0;i<n;i++)
 { cout<<"\nInput the "<<i+1<<" co-ordinate (x,y):";
   cin>>obj[i][0]>>obj[i][1];
   obj[i][2]=1;
 }
 window();
 draw();
 cout<<"\n\nPress any key..";
do
{
 getch();
 cleardevice();
 cout<<"\n\n\t\t\t\tMenu";
 cout<<"\n\n\t\t1.Translation";
 cout<<"\n\t\t2.Scaling";
 cout<<"\n\t\t3.Rotation";
 cout<<"\n\t\t4.X-Shear";
 cout<<"\n\t\t5.Y-Shear";
 cout<<"\n\t\t6.Reflection";
 cout<<"\n\t\t7.Exit";
 cout<<"\n\n\tInput your choice(1-7):";
 cin>>ch;
 switch(ch)
  {
   case 1:trans();
                                                break;
   case 2:scale();
                                                break;
   case 3:rot();
```

```
                                                    break;
  case 4:xshear();
                                                    break;
  case 5:yshear();
                                                    break;
/* case 6:ref();
                                                    break;    */
  case 7:break;
  default:cout<<"\n\n\tWrong choice ... Input again ..";
                                                    getch();
  }

 }while(ch!=7);

 getch();
 closegraph();
}
```

Expected Output:






Press any key..