

Vidarbha Youth Welfare Society's
Prof. Ram Meghe Institute of Technology & Research
Badnera, Amravati (M.S.) 444701



Lab Manual

Semester VII

Subject: - (7KS06) Computer Graphics Lab

Department of Computer Science & Engineering

Vidarbha Youth Welfare Society's
Prof. Ram Meghe Institute of Technology & Research
Badnera, Amravati (M.S.) 444701



CERTIFICATE

This is to certify that Mr/Miss _____Enrolment No.
_____ Roll No. _____ Section_____of B.E. Final year
Semester VII department of Computer Science & Engineering has satisfactorily
completed the term work of the subject **COMPUTER GRAPHICS LAB** prescribed
by Sant Gadge Baba Amravati University, Amravati during the academic term 2022-
23.

Signature of the faculty

Head of Department

Date:

1. Vision and Mission of the Institute and Programme

- **Vision& Mission statement of the Institute**

VISION

To become a pace-setting centre of excellence believing in three universal values namely Synergy, Trust and Passion, with zeal to serve the Nation in the global scenario.

MISSION

To dedicate ourselves to the highest standard of technical education & research in core & emerging engineering disciplines and strive for the overall personality development of students so as to nurture not only quintessential technocrats but also responsible citizens.

- **Vision& Mission statement of the Department of Computer Science & Engineering**

VISION

To ensure that the world saves time and other depletable resources and free it from complexity by providing efficient computing services.

MISSION

To dedicate ourselves to the highest standard by providing knowledge, skills and wisdom to the incumbent by imparting value based education to enable them to solve complex system by simple algorithms and to make them innovative, research oriented to serve the global society, while imbibing highest ethical values.

2. Program educational objective (PEO's), program outcomes (PO's) and Program Specific Outcomes (PSO's)

- **Program educational objective (PEO's)**

PEO1. Preparation: To prepare students for successful careers in software industry that meet the needs of Indian and multinational companies or to excel in Higher studies.

PEO2. Core competence: To develop the ability among students to synthesize data and technical concepts for software design and development.

PEO3. Breadth: To inculcate in students professional and ethical attitude, effective communication skills, teamwork skills, multidisciplinary approach and an ability to relate engineering issues to broader social context.

PEO4. Professionalism: To provide students with a sound foundation in the mathematical, scientific and computer engineering fundamentals required to solve engineering problems and also pursue higher studies.

PEO5. Learning Environment: To promote student with an academic environment aware of excellence, leadership, written ethical codes and guidelines and the life-long learning needed for a successful professional career.

- **Program Outcomes (PO's)**

Engineering Graduate will be able to:

PO1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2. Problem analysis: Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write

effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

- **Program Specific Outcomes (PSO's)**

PSO1: Foundation of Computer System Development: Ability to use knowledge of computer systems and design principles in building the hardware and software components / products in the domain of embedded system, artificial intelligence, databases, networking, web technology and mobile computing.

PSO2: Problem Solving Ability: Ability to apply knowledge in various problem domains and implement innovative / suitable solutions to cater to needs of industry, business and e-governance by imbibing highest ethical and economical values.

3. Syllabus (Practical) Course outcomes, Mapping with PO's

Course Number and Title	: (7KS06) Computer Graphics Lab
Faculty	: Prof.S.V.Deshmukh (<i>svdeshmukh@mitra.ac.in</i>) Dr.A.R.Mune (<i>armune@mitra.ac.in</i>) Prof.S.V.Kalbande (<i>svkalbande@mitra.ac.in</i>)
Course Type	: Theory/Laboratory
Compulsory / Elective	: Compulsory
Teaching Methods	: Lecture: 04 Hr / week. Laboratory: 02 Hrs / week (04 Batches)
Subject Credits	: 04 Credits.
Course Assessment	: Lab Performance + Viva-Voce Semester examination by SGBAU
Grading Policy	: 50 % Internal + 50 % External 25 Internal Marks + 25 External Marks (Viva-Voce Semester exam by SGBAU)

Unit wise Course Contents

Unit I: Introduction and Overview of Graphics System

Definition and Representative uses of computer graphics, Overview of coordinate system, Definition of scan conversion, rasterization and rendering. Raster scan & random scan displays, Architecture of raster graphics system with display processor, Architecture of random scan systems.

Unit II: Output Primitives

Scan conversions of point, line, circle and ellipse: DDA algorithm and Bresenham algorithm for line drawing, midpoint algorithm for circle, midpoint algorithm for ellipse drawing (Mathematical derivation for above algorithms is expected); Aliasing, Antialiasing techniques like Pre and post filtering, super sampling, and pixel phasing).; Filled Area Primitive: Scan line Polygon Fill algorithm, inside outside tests, Boundary Fill and Flood fill algorithm

Unit III: Two Dimensional Geometric Transformations

Basic transformations: Translation, Scaling, Rotation, Matrix representation and Homogeneous Coordinates Composite transformation Other transformations: Reflection and Shear

Unit IV: Two-Dimensional Viewing and Clipping

Viewing transformation pipeline and Window to Viewport coordinate transformation, Clipping operations: Point clipping, Line clipping algorithms: Cohen-Sutherland, Liang Barsky, Polygon Clipping Algorithms: Sutherland-Hodgeman, Weiler-Atherton.

Unit V: Three-Dimensional Geometric Transformations, Curves and Fractal Generation

3D Transformations: Translation, Rotation, Scaling and Reflection, Composite transformations: Rotation about an arbitrary axis, Projections – Parallel, Perspective. (Matrix Representation), Bezier Curve, B-Spline Curve, Fractal- Geometry: Fractal Dimension, Koch Curve.

Unit VI: Visible Surface Detection and Animation

Visible Surface Detection: Classification of Visible Surface Detection algorithm, Back Surface detection method, Depth Buffer method, Area Subdivision method Animation: Introduction, Design of animation sequences, Animation languages, Keyframe, Morphing, Motion specification.

Text Book: Hearn, Baker, “Computer Graphics (C version)” – Pearson Education.

Reference Books:

1. J. Foley, V. Dam, S. Feiner, J. Hughes, —Computer Graphics Principles and Practice, 2nd Edition, Pearson Education, 2003, ISBN 81 – 7808 – 038 – 9.
2. D. Rogers, J. Adams, —Mathematical Elements for Computer Graphics, 2nd Edition, TataMcGrawHill Publication, 2002, ISBN 0 – 07 – 048677 – 8.
3. Mario Zechner, Robert Green, —Beginning Android 4 Games Development, Apress, ISBN: 978-81- 322-0575-3.

• Course Learning Objectives:

Throughout the course, students will be expected to demonstrate their understanding of Computer Graphics by being able to do each of the following:

- To acquaint the learner with the basic concepts of Computer Graphics.
- To learn the various algorithms for generating and rendering graphical figures.
- To get familiar with the mathematics behind the graphical transformations.
- To understand various methods and techniques regarding projections, animation, shading, illumination, and lighting

• Course Outcomes (CO's)

By the end of the course, students will be able to:

Sr. No.	Course Outcome
K706.1	Describe the basic concepts of Computer Graphics.
K706.2	Demonstrate various algorithms for basic graphics primitives.
K706.3	Apply 2-D geometric transformations on graphical objects.

K706.4	Use various Clipping algorithms on graphical objects
K706.5	Explore 3-D geometric transformations, curve representation techniques and projections methods
K706.6	Explain visible surface detection techniques and Animation.

• **Mapping of Course Outcomes with Program Outcomes:**

Course	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12
K704.1	2	2	2	2	2	1	-	-	1	1	1	1
K704.2	2	3	2	2	2	-	-	-	1	1	1	1
K704.3	2	3	3	3	2	-	-	-	1	1	1	1
K704.4	2	3	3	2	2	2	-	-	2	1	1	1
K704.5	3	3	3	2	3	-	-	-	2	1	1	1
K704.6	3	3	3	3	3	2	-	-	2	1	2	1
K708.7	3	3	3	3	3	2	2	-	3	1	2	1

Correlation levels: 1. Low 2. Medium 3.High

• **Mapping of Course Outcomes with Program Specific Outcomes**

Course	PSO 1	PSO 2
K704.1	3	2
K704.2	3	2
K704.3	3	2
K704.4	3	2
K704.5	3	2
K704.6	3	3
K708.7	3	3

Correlation levels: 1. Low 2. Medium 3.High

• **Lab-Course Learning Objectives:**

1. To acquaint the learner with the basic concepts of Computer Graphics.
2. To learn the various algorithms for generating and rendering graphical figures.
3. To get familiar with mathematics behind the graphical transformations.
4. To understand and apply various methods and techniques regarding projections, animation, shading, illumination and lighting

5. To prepare the student for advance areas like Image Processing or Computer Vision or Virtual Reality and professional avenues in the field of Computer Graphics.

4. Practical Evaluation Guidelines: (ACPV Guidelines)

Guidelines for Awarding Internal Marks for Practical:

At the end of the semester, internal assessment marks for practical shall be the average of marks scored in all the experiments.

- a. **Attendance (05 Marks):** These 05 marks will be given for the regularity of a student. If the student is present on the scheduled day, he / she will be awarded 05 marks.
- b. **Competency (05 Marks):** Here the basic aim is to check competency of the student. The skill to interpret the aim of the practical and provide solution for the same. Here expectation from the student is to improvise the existing solution, and the given modification. The marks will be given according to the level of improvement made in the practical.
- c. **Performance (05 Marks):** Here the basic aim is to check whether the student has developed the skill to write down the code on his own, and compile / debug. If a student correctly executes the practical on the scheduled day within the allocated time; he / she will be awarded full marks. Otherwise, marks will be given according to the level of completion / execution of practical.
- d. **Viva-Voce (05 Marks):** These 05 marks will be totally based on the performance in viva-voce. There shall be viva-voce on the completion of each practical in the same practical session. The student shall get the record checked before next practical.

List of Practical

Index

Sr. No.	List of Practical	Page No.	Date	Remark
1	Write a Program to draw a line using DDA algorithm.			
2	Write a Program to draw a line using Bresenham's algorithm.			
3	Write a Program to draw a circle using Bresenham's algorithm.			
4	Write a program for 2-D transformations, a) Scaling b) Translation c) Rotation			
5	Write a program for 3-D transformations, a) Scaling b) Translation e) Rotation			
6	Write program to fill polygon using scan line algorithm			
7	Write a program to fill rectangle using floodfill algorithm.			
8	Write a program to clip line using following algorithm: Cohen-Sutherland algorithm.			
9	Write a program to draw following type of curve-Hilbert's Curve			

PRACTICAL NO 1

Aim: Write a Program to draw a line using DDA algorithm.

Software Required: Turbo C++

Theory:

Digital Differential Analyzer algorithm is a simple line generation algorithm. A line connects to end points. It is a basic element in computer graphics. To draw a line, you need 2 points between which you can draw a line. DDA is used for interpolation of variables over an interval between start and end point, DDS are used for rasterization of lines, triangles, and polygons. They can be extended to nonlinear function, such as perspective mapping, quadratic curves, and travels in pixels.

In its simplest implementation for linear cases such as lines, the DDA algorithm interpolates values in interval by computing for each x_i the equations $x_i = x_{i-1} + 1$, $y_i = y_{i-1} + 1$.

Procedure:

Step 1: Read the input of the 2 end points of the line as (x1, y1) & (x2, y2) such that $x1 \neq x2$ and $y1 \neq y2$

Step 2: Calculate $dx = x2 - x1$ and $dy = y2 - y1$

Step 3:

```
if(dx>=dy)
    step=dx
```

```
else
    step=dy
```

Step 4: $x_{in} = dx / step$ & $y_{in} = dy / step$

Step 5: $x = x1 + 0.5$ & $y = y1 + 0.5$

Step 6:

```
for(k = 0; k < step; k++)
{
    x = x + xin
    y = y + yin
    putpixel(x, y)
}
```

Sample Output of the source example:



Program:

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

void main( )
{
    float x,y,x1,y1,x2,y2,dx,dy,step;
    int i,gd = DETECT,gm;

    initgraph(&gd,&gm,"c:\\turbo3\\bgi");

    printf("Enter the value of x1 and y1 : ");
    scanf("%f%f",&x1,&y1);
    printf("Enter the value of x2 and y2 : ");
    scanf("%f%f",&x2,&y2);

    dx = abs(x2-x1);
    dy = abs(y2-y1);

    if(dx>=dy)
        step=dx;
    else
        step=dy;

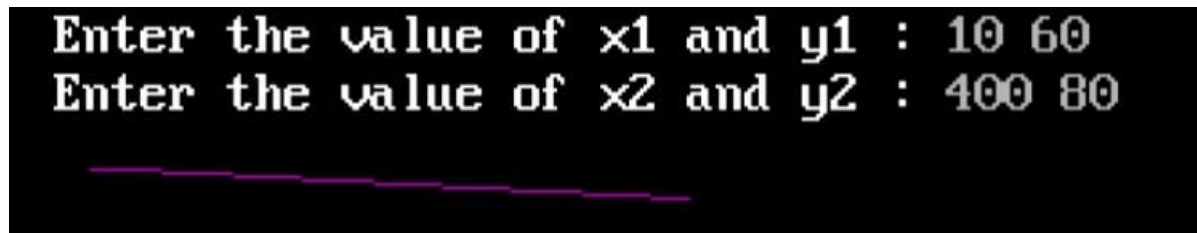
    dx = dx/step;
    dy = dy/step;

    x = x1;
    y = y1;

    i = 1;
    while(i<=step)
    {
        putpixel(x,y,5);
        x=x+dx;
        y=y+dy;
        i=i+1;
        delay(100);
    }

    closegraph();
}
```

Output:-



A screenshot of a terminal window with a black background and white text. The first line shows the prompt 'Enter the value of x1 and y1 : ' followed by the input '10 60'. The second line shows the prompt 'Enter the value of x2 and y2 : ' followed by the input '400 80'. Below the second line, there is a horizontal line of small, light blue dashes.

```
Enter the value of x1 and y1 : 10 60
Enter the value of x2 and y2 : 400 80
_____
```

Conclusion:-

Viva Questions:

- 1) Define the term Rasterization.
- 2) Write slope intercept form of line.
- 3) Write advantages and disadvantages of DDA Algorithm.
- 4) Define aspect ration.

(Space for answers)

PRACTICAL NO 2

Aim: Write a Program to draw a line using Bresenham's algorithm.

Software Required: Turbo C++

Theory:

Algorithm determines the point of n-dimensional raster that is selected to form a close approximation to a straight line between two points to stop it is used to draw line primitives in a bitmap image as it uses only integer addition subtraction and bit shifting. It is an incremental error method. It is one of the earliest algorithms developed in the field of computer graphics.

Consider a line with initial point (x_1, y_1) and terminal point (x_2, y_2) in device space. If $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$, we define the driving axis (DA) to be the x-axis if $|\Delta x| \geq |\Delta y|$, and the y-axis if $|\Delta y| > |\Delta x|$. The DA is used as the "axis of control" for the algorithm and is the axis of maximum movement. Within the main loop of the algorithm, the coordinate corresponding to the DA is incremented by one unit. The coordinate corresponding to the other axis (usually denoted the passive axis or PA) is only incremented as needed.

Procedure:

1. Input the two line end-points, storing the left end-point in (x_0, y_0)
2. Plot the point (x_0, y_0)
3. Calculate the constants Δx , Δy , $2\Delta y$, and $(2\Delta y - 2\Delta x)$ and get the first value for the decision parameter as: $P_0 = 2\Delta y - 2\Delta x$
4. At each x_k along the line, starting at $k=0$, perform the following test:
 If $P_k < 0$, the next point to plot is (x_k+1, y_k) and $P_{k+1} = P_k + 2\Delta y$
 Otherwise, the next point to plot is (x_k+1, y_k+1) and $P_{k+1} = P_k + 2\Delta y - 2\Delta x$
5. Repeat step 4 (Δx) times.

Sample Output of the source example:



Program:

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

void drawline(int x0, int y0, int x1, int y1)
{
    int dx, dy, p, x, y;
    dx=x1-x0;
    dy=y1-y0;
    x=x0;
```



```

y=y0;

p=2*dy-dx;

while(x<x1)
{
    if(p>=0)
    {
        putpixel(x,y,7);
        y=y+1;
        p=p+2*dy-2*dx;
    }
    else
    {
        putpixel(x,y,7);
        p=p+2*dy;
    }
    x=x+1;
    delay(100);
}

int main()
{
    int gdriver=DETECT, gmode, error, x0, y0, x1, y1;
    initgraph(&gdriver, &gmode, "c:\\turbo3\\bgi");

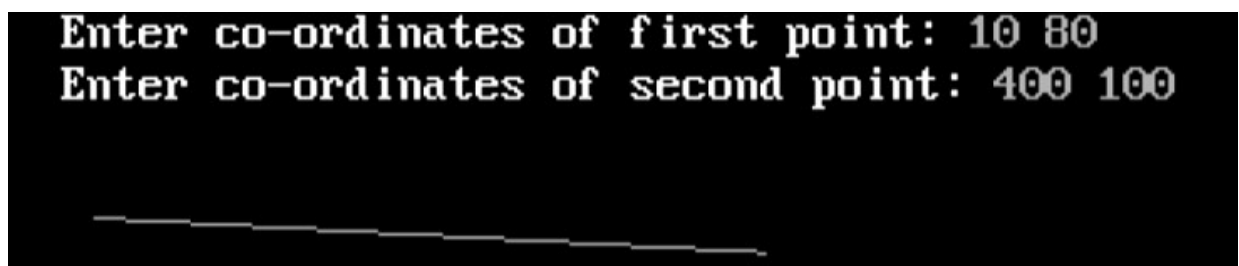
    printf("Enter co-ordinates of first point: ");
    scanf("%d%d", &x0, &y0);

    printf("Enter co-ordinates of second point: ");
    scanf("%d%d", &x1, &y1);
    drawline(x0, y0, x1, y1);

    return 0;
}

```

Output:



Conclusion:-

Viva Questions:

- 1) Define the term decision parameter.
- 2) Write slope intercept form of line.
- 3) Write advantages of Bresenham's over DDA Algorithm.
- 4) State one difference between raster graphic and random graphic.

(Space for answers)

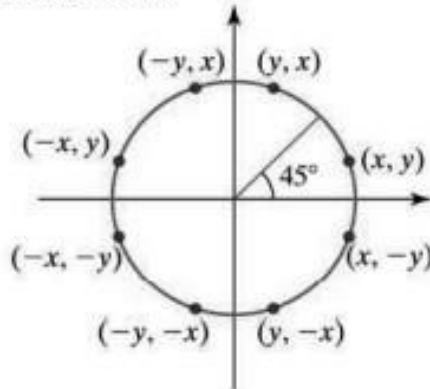
PRACTICAL NO 3

Aim: Write a Program to draw a circle using Bresenham's algorithm.

Software Required: Turbo C++

Theory:

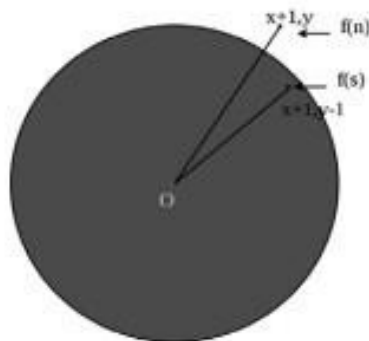
The equation of circle is $x^2 + y^2 = r^2$, where (x, y) are coordinates of a center and r is radius. To draw circle in computer graphics we use Bresenham's circle drawing algorithm and Mid Point circle drawing algorithm.



It's not easy to display a continuous arc on the raster display. Instead, we have to choose the nearest pixel position to complete the arc.

Bresenham's Circle algorithm:

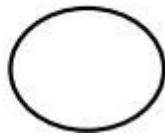
Bresenham's algorithm is based on the idea of determining the subsequent pixels required to draw the circle. $x+1, y$ is to find the next pixel to draw the circle.



Procedure:

1. Set initial values of center of the circle (xc, yc) and (x, y)
2. Set decision parameter d to $d = 3 - (2 * r)$.
3. Repeat steps 4 to 8 until $x \leq y$
4. Call drawCircle(int xc, int yc, int x, int y) function.
5. Increment value of x .
6. If $d < 0$, set $d = d + (4 * x) + 6$
7. Else, set $d = d + 4 * (x - y) + 10$ and decrement y by 1.
8. Call drawCircle(int xc, int yc, int x, int y) function.

Sample Output of the source example:



Program:

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

void drawCircle(int xc, int yc, int x, int y)
{
    putpixel(xc+x, yc+y, RED);
    putpixel(xc-x, yc+y, RED);
    putpixel(xc+x, yc-y, RED);
    putpixel(xc-x, yc-y, RED);
    putpixel(xc+y, yc+x, RED);
    putpixel(xc-y, yc+x, RED);
    putpixel(xc+y, yc-x, RED);
    putpixel(xc-y, yc-x, RED);
}

void circleBres(int xc, int yc, int r)
{
    int x = 0, y = r;
    int d = 3 - 2 * r;
    drawCircle(xc, yc, x, y);
    while (y >= x)
    {
        // for each pixel we will
        // draw all eight pixels

        x++;

        // check for decision parameter
        // and correspondingly
        // update d, x, y
        if (d > 0)
        {
            y--;
            d = d + 4 * (x - y) + 10;
        }
        else
            d = d + 4 * x + 6;
        drawCircle(xc, yc, x, y);
        delay(50);
    }
}
```

```
int main()
{
    int xc = 50, yc = 50, r = 30;
    int gd = DETECT, gm;
    initgraph(&gd, &gm, ""); // initialize graph
    circleBres(xc, yc, r); // function call
    return 0;
}
```

Output:



Conclusion:

Viva Questions:

- 1) Write equation of a circle
- 2) Write the algorithm to draw 8-way symmetry of a circle.
- 3) How the value of decision parameter (d) is calculated?

(Space for answers)

PRACTICAL NO 4

Aim: Write a program for 2-D transformations a) Scaling b) Translation c) Rotation.

Software Required: Turbo C++

Theory:

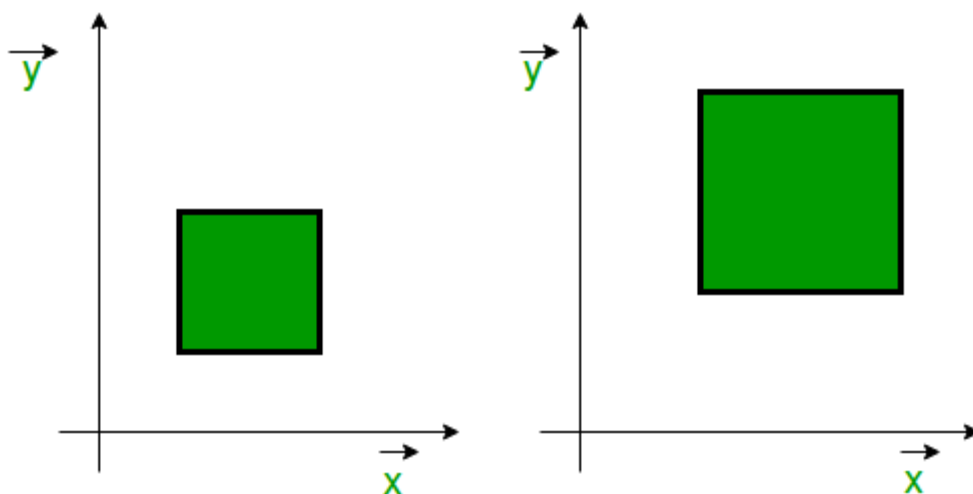
We can use a 2×2 matrix to change, or transform, a 2D vector. This kind of operation, which takes in a 2-vector and produces another 2-vector by a simple matrix multiplication, is a linear transformation.

By this simple formula we can achieve a variety of useful transformations, depending on what we put in the entries of the matrix. For our purposes, consider moving along the x-axis a horizontal move and along the y-axis, a vertical move.

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 factors, s_x and s_y , to produce the transformed coordinates as (x', y'). So, $x' = x * s_x$, and $y' = y * s_y$. The scaling factors, s_x and s_y , scale the object in X and Y direction respectively. So, the above equation can be represented in matrix form.

Procedure:

1. Make a 2x2 scaling matrix S as:
(s_x 0)
(0 s_y)
2. For each point of the polygon.
 - (i) Make a 2x1 matrix P, where P[0][0] equals to x coordinate of the point 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.
3. Draw the polygon using new coordinates.



Program:

```
// Scaling
#include<stdio.h>
#include<graphics.h>
void findNewCoordinate(int s[][2], int p[][1])
{
    int temp[2][1] = { 0 };
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 1; j++)
            for (int k = 0; k < 2; k++)
                temp[i][j] += (s[i][k] * p[k][j]);
    p[0][0] = temp[0][0];
    p[1][0] = temp[1][0];
}
void scale(int x[], int y[], int sx, int sy)
{
    line(x[0], y[0], x[1], y[1]);
    line(x[1], y[1], x[2], y[2]);
    line(x[2], y[2], x[0], y[0]);
    int s[2][2] = { sx, 0, 0, sy };
    int p[2][1];
    for (int i = 0; i < 3; i++)
    {
        p[0][0] = x[i];
        p[1][0] = y[i];
        findNewCoordinate(s, p);
        x[i] = p[0][0];
        y[i] = p[1][0];
    }
    line(x[0], y[0], x[1], y[1]);
    line(x[1], y[1], x[2], y[2]);
    line(x[2], y[2], x[0], y[0]);
}
int main()
{
    int x[] = { 100, 200, 300 };
    int y[] = { 200, 100, 200 };
    int sx = 2, sy = 2;
    int gd, gm;
    detectgraph(&gd, &gm);
    initgraph(&gd, &gm, " ");
    scale(x, y, sx, sy);

    getch();
    return 0;
}
```

// Translation

```
#include<bits/stdc++.h>
#include<graphics.h>
using namespace std;

// function to translate rectangle
void translateRectangle ( int P[][2], int T[])
{
    int gd = DETECT, gm, errorcode;
    initgraph (&gd, &gm, "c:\\tc\\bgi");
    setcolor (2);

    rectangle (P[0][0], P[0][1], P[1][0], P[1][1]);

    P[0][0] = P[0][0] + T[0];
    P[0][1] = P[0][1] + T[1];
    P[1][0] = P[1][0] + T[0];
    P[1][1] = P[1][1] + T[1];

    rectangle (P[0][0], P[0][1], P[1][0], P[1][1]);
}

// driver program
int main()
{
    int P[2][2] = {5, 8, 12, 18};
    int T[] = {2, 1}; // translation factor
    translateRectangle (P, T);
    return 0;
}
```

// Rotation

```
#include <math.h>
#include <stdio.h>
#define SIN(x) sin(x * 3.141592653589 / 180)
#define COS(x) cos(x * 3.141592653589 / 180)

void rotate(float a[][2], int n, int x_pivot, int y_pivot,
            int angle)
{
    int i = 0;
    while (i < n) {
        // Shifting the pivot point to the origin
        // and the given points accordingly
        int x_shifted = a[i][0] - x_pivot;
        int y_shifted = a[i][1] - y_pivot;

        // Calculating the rotated point co-ordinates
        // and shifting it back
        a[i][0] = x_pivot
```

```

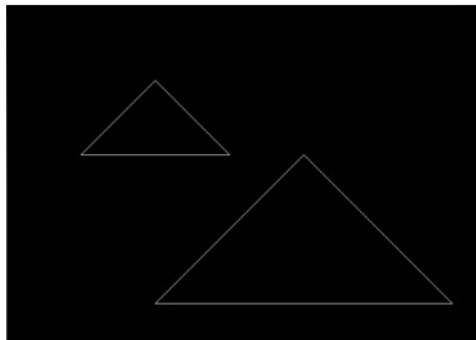
        + (x_shifted * COS(angle)
          - y_shifted * SIN(angle));
a[i][1] = y_pivot
        + (x_shifted * SIN(angle)
          + y_shifted * COS(angle));
printf("%f, %f) ", a[i][0], a[i][1]);
i++;
    }
}

int main()
{
    int size1 = 4; // No. of vertices

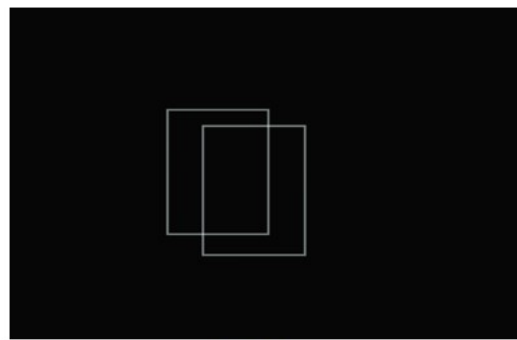
    float points_list1[][2] = { { 100, 100 }, { 150, 200 }, { 200, 200 }, { 200, 150 } };
    rotate(points_list1, size1, 0, 0, 90);
    return 0;
}

```

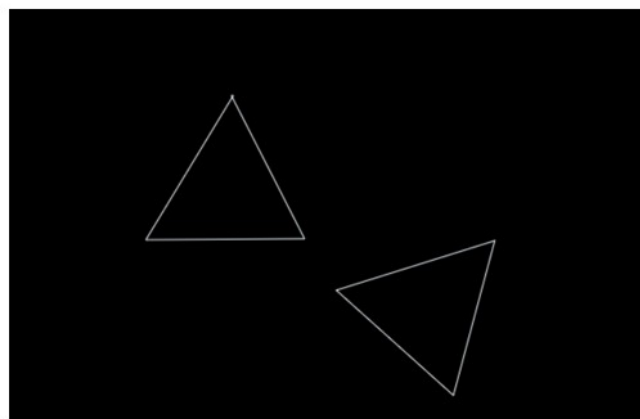
Output:



Scaling



Translation



Rotation

Conclusion:

Viva Questions:

1. Which are the basic geometric transformations?
2. Define the terms scaling, translation, and rotation.
3. What is viewing transformation?
4. What is 2D transformation?

(Space for answers)

PRACTICAL NO 5

Aim: Write a program for 3-D transformations a) Scaling b) Translation c) Rotation.

Software Required: Turbo C++

Theory:

3-D Transformation: In very general terms a 3D model is a mathematical representation of a physical entity that occupies space. In more practical terms, a 3D model is made of a description of its shape and a description of its color appearance. 3-D Transformation is the process of manipulating the view of a three-D object with respect to its original position by modifying its physical attributes through various methods of transformation like Translation, Scaling, Rotation, Shear, etc.

Properties of 3-D Transformation:

- Lines are preserved,
- Parallelism is preserved,
- Proportional distances are preserved

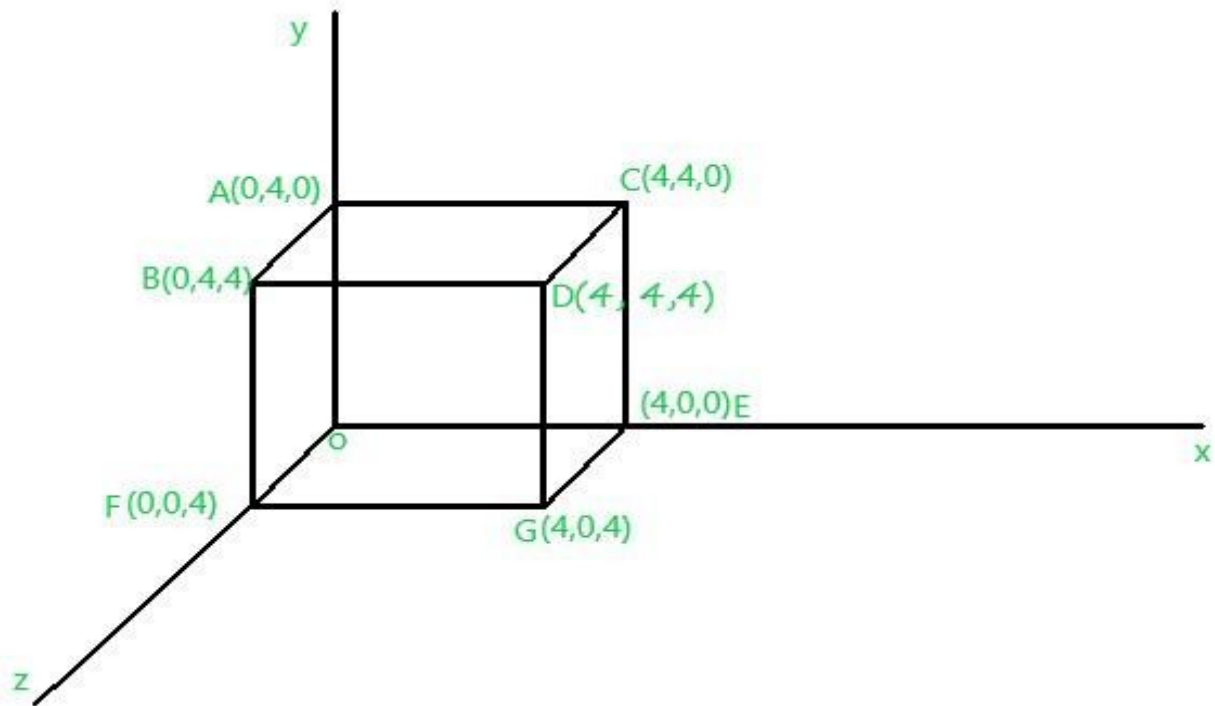
Procedure:

It is the process of changing the relative location of a 3-D object with respect to the original position by changing its coordinates. Translation transformation matrix in the 3-D image is shown as –

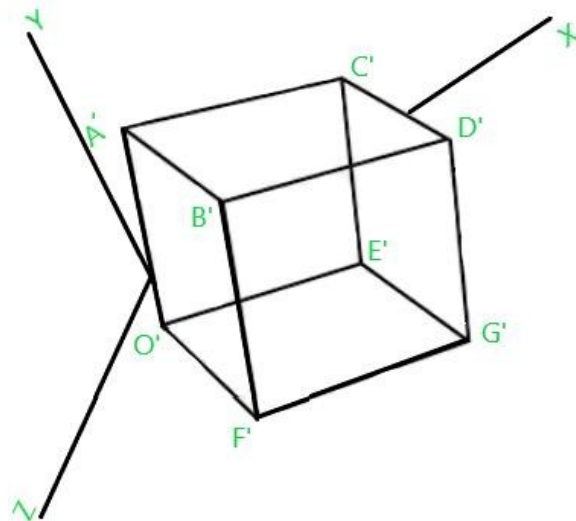
Where D_x , D_y , D_z are the Translation distances, let a point in 3D space is $P(x, y, z)$ over which we want to apply Translation Transformation operation and we are given with translation distance $[D_x, D_y, D_z]$ So, new position of the point after applying translation operation would be –

Perform translation transformation on the following figure where the given translation distances are $D_x = 2$, $D_y = 4$, $D_z = 6$.

On applying Translation Transformation we get corresponding points –



After performing **translation transformation** over the **Fig.1**, it will look like as below –



Program:

```
#include <iostream>
#include <cmath>
using namespace std;
typedef struct {
    float x;
    float y;
    float z;
}Point;
Point points;

float temp = 0;

void showPoint(){
    cout<<"("<<points.x<<","<<points.y<<","<<points.z<<")"<<endl;
}

void translate(float tx, float ty, float tz){
    points.x += tx;
    points.y += ty;
    points.z += tz;
    cout<<"After Translation, new point is :";
    showPoint();
}

void rotatex(float angle){
    angle = angle * M_PI / 180.0;
    temp = points.y;
    points.y = points.y * cos(angle) - points.z * sin(angle);
    points.z = temp * sin(angle) + points.z * cos(angle);
    cout<<"After rotation about x, new point is: ";
    showPoint();
}

void rotatey(float angle){
    angle = (angle * M_PI) / 180.0;
    temp = points.z;
    points.z = points.z * cos(angle) - points.x * sin(angle);
    points.x = temp * sin(angle) + points.x * cos(angle);
    cout<<"After rotation about y, new point is: ";
    showPoint();
}

void rotatez(float angle){
    angle = angle * M_PI / 180.0;
    temp = points.x;
    points.x = points.x * cos(angle) - points.y * sin(angle);
    points.y = temp * sin(angle) + points.y * cos(angle);
    cout<<"After rotation about z, new point is: ";
    showPoint();
}

void scale(float sf, float xf, float yf, float zf){
```



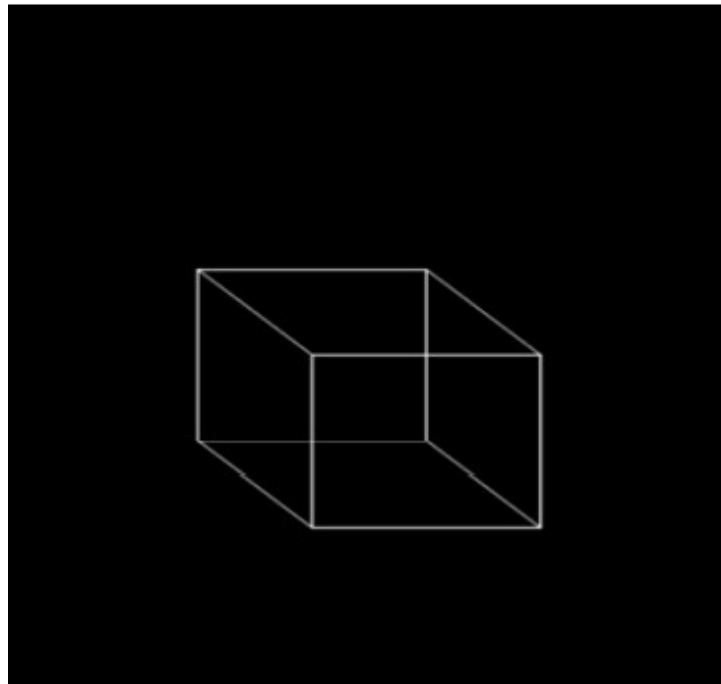
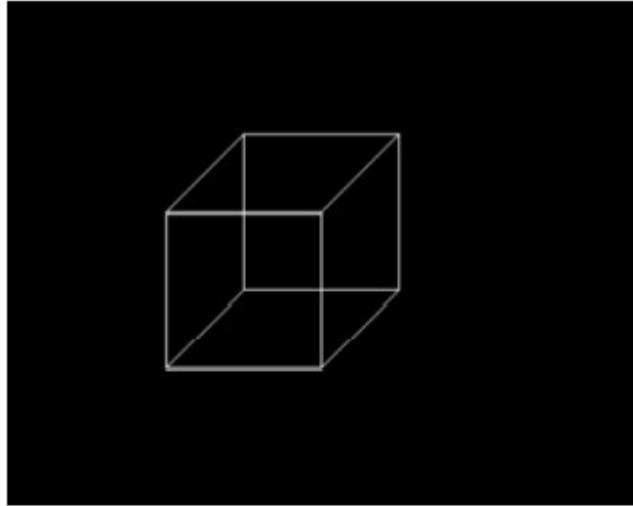
```

    points.x = points.x * sf + (1 - sf) * xf;
    points.y = points.y * sf + (1 - sf) * yf;
    points.z = points.z * sf + (1 - sf) * zf;
    cout<<"After scaling, new point is: ";
    showPoint();
}

int main()
{
    float tx = 0, ty = 0, tz = 0;
    float sf = 0, xf = 0, yf = 0, zf = 0;
    int choose;
    float angle;
    cout<<"Enter the initial point you want to transform:";
    cin>>points.x>>points.y>>points.z;
    cout<<"Choose the following: "<<endl;
    cout<<"1. Translate"<<endl;
    cout<<"2. Rotate about X axis"<<endl;
    cout<<"3. Rotate about Y axis"<<endl;
    cout<<"4. Rotate about Z axis"<<endl;
    cout<<"5. Scale"<<endl;
    cin>>choose;
    switch(choose){
        case 1:
            cout<<"Enter the value of tx, ty and tz: ";
            cin>>tx>>ty>>tz;
            translate(tx, ty, tz);
            break;
        case 2:
            cout<<"Enter the angle: ";
            cin>>angle;
            rotatex(angle);
            break;
        case 3:
            cout<<"Enter the angle: ";
            cin>>angle;
            rotatey(angle);
            break;
        case 4:
            cout<<"Enter the angle: ";
            cin>>angle;
            rotatez(angle);
            break;
        case 5:
            cout<<"Enter the value of sf, xf, yf and zf: ";
            cin>>sf>>xf>>yf>>zf;
            scale(sf, xf, yf, zf);
            break;
        default:
            break;
    }
    return 0;
}

```

Output:



Conclusion:

Viva Questions:

- 1) Define curve.
- 2) Define the terms scaling, translation, and rotation.
- 3) What is 3D transformation?
- 4) Write properties of 3D transformation?

(Space for answers)

PRACTICAL NO 6

Aim: Write a program to fill polygon using scan line algorithm.

Software Required: Turbo C++

Theory:

Scanline filling is basically filling up of polygons using horizontal lines or scanlines. The purpose of the SLPF algorithm is to fill (color) the interior pixels of a polygon given only the vertices of the figure. To understand Scanline, think of the image being drawn by a single pen starting from bottom left, continuing to the right, plotting only points where there is a point present in the image, and when the line is complete, start from the next line and continue. This algorithm works by intersecting scanline with polygon edges and fills the polygon between pairs of intersections.

Special cases of polygon vertices:

1. If both lines intersecting at the vertex are on the same side of the scanline, consider it as two points.
2. If lines intersecting at the vertex are at opposite sides of the scanline, consider it as only one point.

Components of Polygon fill:

1. Edge Buckets: It contains an edge's information. The entries of edge bucket vary according to data structure you have used. In the example we are taking below, there are three edge buckets namely: ymax, xofymin, slopeinverse.
2. Edge Table: It consists of several edge lists -> holds all of the edges that compose the figure. When creating edges, the vertices of the edge need to be ordered from left to right and the edges are maintained in increasing yMin order. Filling is complete once all of the edges are removed from the ET
3. Active List: IT maintains the current edges being used to fill in the polygon. Edges are pushed into the AL from the Edge Table when an edge's yMin is equal to the current scan line being processed.
4. The Active List will be re-sorted after every pass.

Procedure:

1. We will process the polygon edge after edge, and store in the edge Table.
2. Storing is done by storing the edge in the same scanline edge tuple as the lowermost point's y-coordinate value of the edge.
3. After addition of any edge in an edge tuple, the tuple is sorted using insertion sort, according to its xofymin value.
4. After the whole polygon is added to the edge table, the figure is now filled.

5. Filling is started from the first scanline at the bottom, and continued till the top.
6. Now the active edge table is taken and the following things are repeated for each scanline:
 - a. Copy all edge buckets of the designated scanline to the active edge tuple
 - b. Perform an insertion sort according to the xofymin values
 - i. Remove all edge buckets whose ymax is equal or greater than the scanline
 - c. Fill-up pairs of edges in active tuple, if any vertex is got, follow these instructions:
 - If both lines intersecting at the vertex are on the same side of the scanline, consider it as two points.
 - If lines intersecting at the vertex are at opposite sides of the scanline, consider it as only one point.
 - d. Update the xofymin by adding slope inverse for each bucket.

Program:

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

void main()
{
    int n,i,j,k,gd,gm,dy,dx;
    int x,y,temp;
    int a[20][2],xi[20];
    float slope[20];
    clrscr();
    printf("\nEnter no.of edges of polygon");
    scanf("%d",&n);
    printf("\nEnter co-ordinates of polygon");

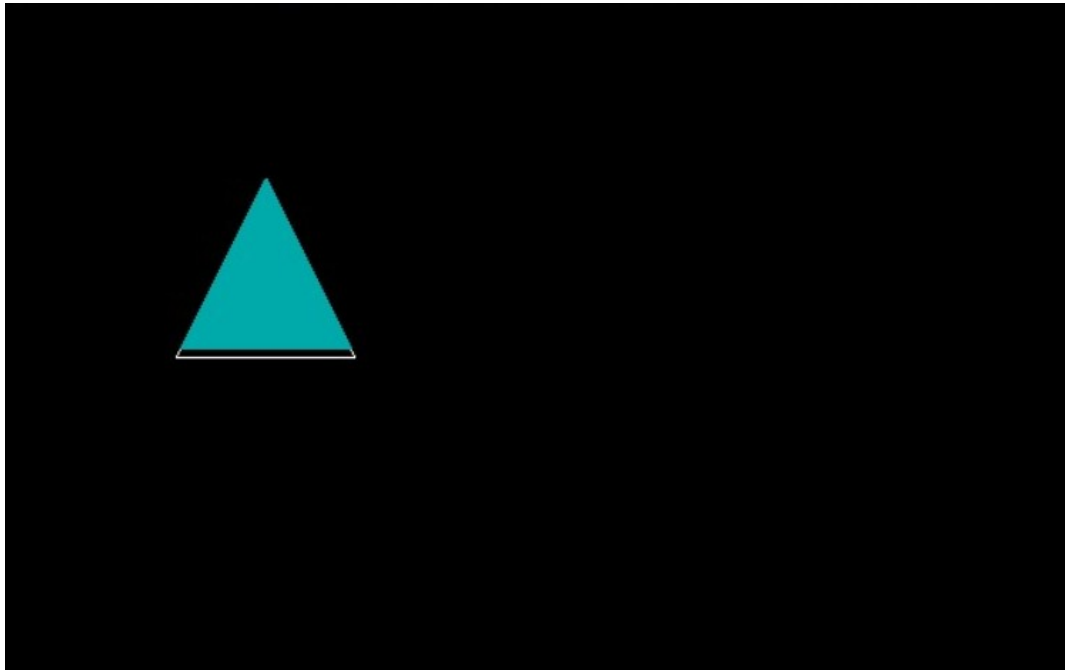
    for(i=0;i<n;i++) {
        printf("\tX%dY%d:",i,i);
        scanf("%d%d",&a[i][0],&a[i][1]);
    }
    a[n][0]=a[0][0];
    a[n][1]=a[0][1];
    detectgraph(&gd,&gm);
    initgraph(&gd,&gm,"c:\\tc\\bgi");
    for(i=0;i<n;i++) {
        line(a[i][0],a[i][1],a[i+1][0],a[i+1][1]);
    }
    getch();
    for(i=0;i<n;i++) {
        dy=a[i+1][1]-a[i][1];
        dx=a[i+1][0]-a[i][0];
        if(dy==0)
            slope[i]=1.0;
        if(dx==0)
            slope[i]=0.0;
        if((dy!=0)&&(dx!=0)) {
            slope[i]=(float)dx/dy;
        }
    }
    for(y=0;y<480;y++)
    {
        k=0;for(i=0;i<n;i++)
        {
            if((a[i][1]<=y)&&(a[i+1][1]>y)){(((a[i][1]>y)&&(a[i+1][1]<=y))xi[k]=(int)(a[i][0]+slope[i]*(y-a[i][1])));
                k++;
            }
        }
    }
}
```

```

for(j=0;j<k-1;j++)
    for(i=0;i<k;i++){
        if(xi[i]>xi[i+1]){
            temp=xi[i];
            xi[i]=xi[i+1];
            xi[i+1]=temp;
        }
    }
setcolor(35);
for(i=0;i<k;i+=2)
{
    line(xi[i],y,xi[i+1]+1,y);
    getch();
}
}

```

Output:



Conclusion:

Viva Questions:

- 1) Define scan line algorithm.
- 2) Define flood fill algorithm.
- 3) Define boundary fill algorithm.
- 4) State one advantage of each algorithm mentioned above.

(Space for answers)

PRACTICAL NO 7

Aim: Write a program to fill color in rectangle using floodfill.

Software Required: Turbo C++

Theory:

In this method, a point or seed which is inside region is selected. This point is called a seed point. Then four connected approaches or eight connected approaches is used to fill with specified color.

The flood fill algorithm has many characters similar to boundary fill. But this method is more suitable for filling multiple colors boundary. When boundary is of many colors and interior is to be filled with one color we use this algorithm.

Procedure:

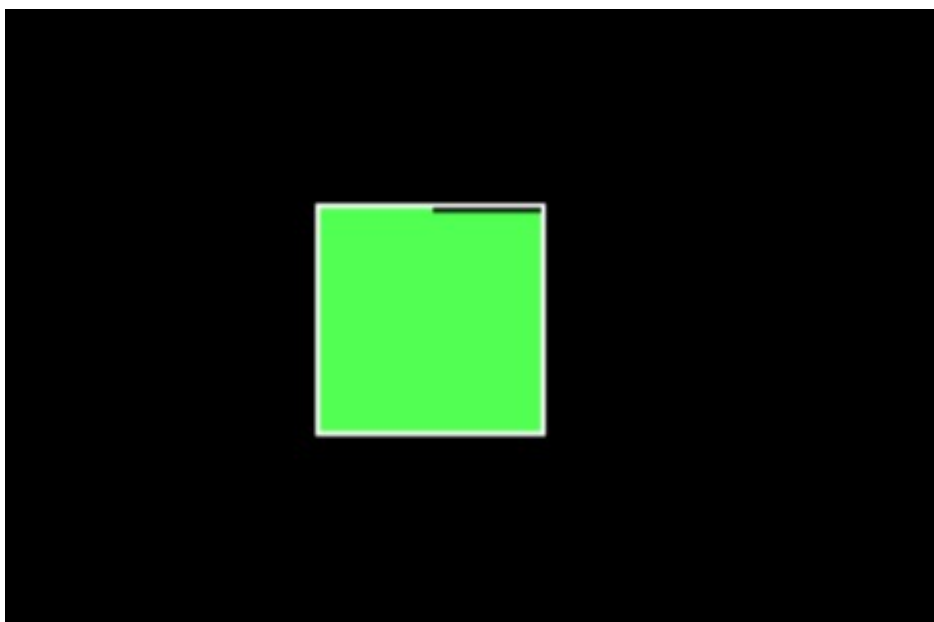
1. Take the position of the starting point.
2. Decide whether you want to go in 4 directions (N, S, W, E) or 8 directions (N, S, W, E, NW, NE, SW, SE).
3. Choose a replacement color and a target color.
4. Travel in those directions.
5. If the tile you land on is a target, replace it with the chosen color.
6. Repeat 4 and 5 until you've been everywhere within the boundaries.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
void flood(int,int,int,int);
void main()
{
    int gd=DETECT,gm;
    initgraph(&gd,&gm,"C:/TURBOC3/bgi");

    rectangle(50,50,120,120);
    flood(55,55,10,0);
    getch();
}
void flood(intx,inty,intfillColor, intdefaultColor)
{
    if(getpixel(x,y)==defaultColor)
    {
        delay(1);
        putpixel(x,y,fillColor);
        flood(x+1,y,fillColor,defaultColor);
        flood(x-1,y,fillColor,defaultColor);
        flood(x,y+1,fillColor,defaultColor);
        flood(x,y-1,fillColor,defaultColor);
    }
}
```

Output:



Conclusion:

Viva Questions:

1. Explain difference between floodfill and boundary fill.
2. Define DDA.
3. What is animation?

(Space for answers)

PRACTICAL NO 8

Aim: Write a program to clip line using Cohen Sutherland algorithm.

Software Required: Turbo C++

Theory:

This is one of the oldest and most popular line clipping algorithm. To speed up the process this algorithm performs initial tests that reduce number of intersections than must be calculated. It does so by using a 4-bit code called as region code or out codes. These codes identify location of the end point of line. Each bit position indicates a direction, starting from the rightmost position of each bit indicates left, right bottom, top respectively. Once we establish region codes for both the endpoints of a line we determine whether the endpoint is visible, partially visible or invisible with the help of ANDing of the region codes.

Procedure

1. Read 2 end point of line as $p1(x1, y1)$ and $p2(x2, y2)$
2. Read 2 corner points of the clipping window (left-top and right-bottom) as $(wx1, wy1)$ and $(wx2, wy2)$
3. Assign the region codes for 2 endpoints $p1$ and $p2$ using following steps :-
 - Initialize code with 0000
 - Set bit 1 if $x < wx1$
 - Set bit 2 if $x > wx2$
 - Set bit 3 if $y < wy1$
 - Set bit 4 if $y > wy2$
4. Check for visibility of line
 - a. If region codes for both endpoints are zero, then line is completely visible. Draw the line go to step 9
 - b. If region codes for endpoints are not zero and logical ANDing of them is also nonzero then line is invisible. Discard the line and move to step 9.
 - c. If it does not satisfy 4.a and 4.b then line is partially visible.
5. Determine the intersecting edge of clipping window as follows: -
 - a. If region codes for both endpoints are nonzero find intersection points $p1$ and $p2$ with boundary edges.
 - b. If region codes for any one end point is nonzero then find intersection

point p1 and p2

6. Divide the line segments considering intersection points.
7. Reject line segment if any end point of line appears outside of any boundary.
8. Draw the clipped line segment.
9. Stop.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<graphics.h>
#include<dos.h>
typedef struct coordinate
{ int x,y;
char code[4];
} PT;

void drawwindow();
void drawline(PT p1,PT p2);
PT setcode(PT p);
int visibility(PT p1,PT p2);
PT resetendpt(PT p1,PT p2);

void main()
{
int gd=DETECT,v,gm;
PT p1,p2,p3,p4,ptemp;
printf("\nEnter x1 and y1\n");
scanf("%d %d",&p1.x,&p1.y);
printf("\nEnter x2 and y2\n");
scanf("%d %d",&p2.x,&p2.y);
initgraph(&gd,&gm,"c:\\turbo3\\bgi");
drawwindow();
delay(500);
drawline(p1,p2);
delay(500);
cleardevice();
delay(500);
p1=setcode(p1);
p2=setcode(p2);
v=visibility(p1,p2);
delay(500);
switch(v)
{
case 0: drawwindow();
```

```

delay(500);
drawline(p1,p2);
break;
case 1: drawwindow();
delay(500);
break;
case 2: p3=resetendpt(p1,p2);
p4=resetendpt(p2,p1);
drawwindow();
delay(500);
drawline(p3,p4);
break;
}
delay(5000);
closegraph();
}

void drawwindow()
{
line(150,100,450,100);
line(450,100,450,350);
line(450,350,150,350);
line(150,350,150,100);
}

void drawline(PT p1,PT p2)
{
line(p1.x,p1.y,p2.x,p2.y);
}

PT setcode(PT p) //for setting the 4 bit code
{
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);
}

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);
}

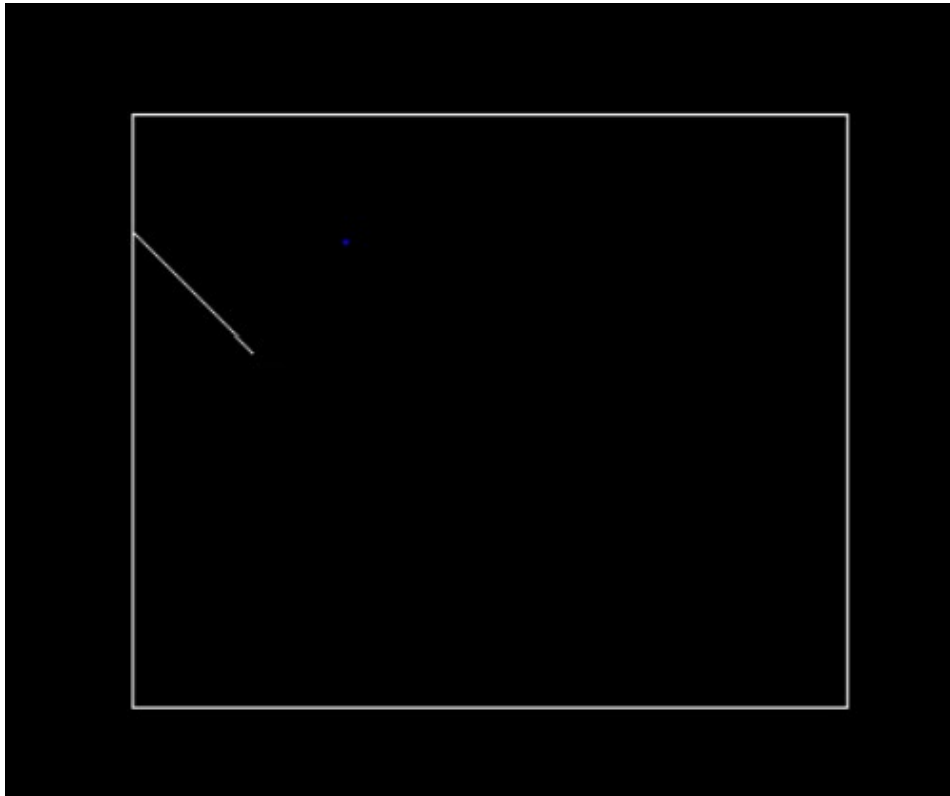
PT resetendpt(PT p1,PT p2)
{
PT temp;
int x,y,i;
float m,k;
if(p1.code[3]=='1')
x=150;
if(p1.code[2]=='1')
x=450;
if((p1.code[3]=='1') || (p1.code[2]=='1'))
{
m=(float)(p2.y-p1.y)/(p2.x-p1.x);
k=(p1.y+(m*(x-p1.x)));
temp.y=k;
temp.x=x;
for(i=0;i<4;i++)
temp.code[i]=p1.code[i];
if(temp.y<=350 && temp.y>=100)
return (temp);
}
if(p1.code[0]=='1')
y=100;
if(p1.code[1]=='1')
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);
}
```

Output:



Conclusion:-

Viva Questions:

- 1) What is the difference between setfillstyle() and floodfill()
 - 2) Define clipping and clip window.
 - 3) State pros and cons of Cohen-Sutherland line clipping algorithm
- (Space for answers)

Practical No 9

Aim: Write a program to draw following type of curve – Hilbert's curve..

Software Required: Turbo C++

Theory:

The Hilbert curve is a space filling curve that visits every point in a square grid with a size of 2x2, 4x4, 8x8, 16x16 or any other power of 2. Hilbert curve in image processing is used for image compression and dithering. It has advantages in those operations where the coherence between neighboring pixels is important. The Hilbert curve is also a special version of a quad tree; any image processing function that benefits from the use of quad trees may also use a Hilbert curve. The Curve can be built by following Successive approximation.

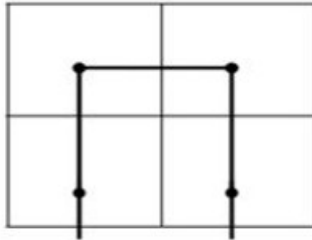


Figure 1: First Approximation to Hilbert Curve

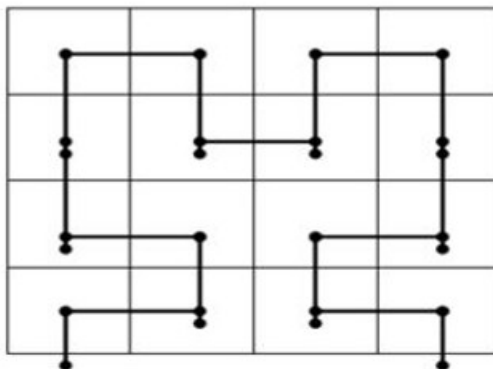


Figure 2: Second Approximation to Hilbert Curve

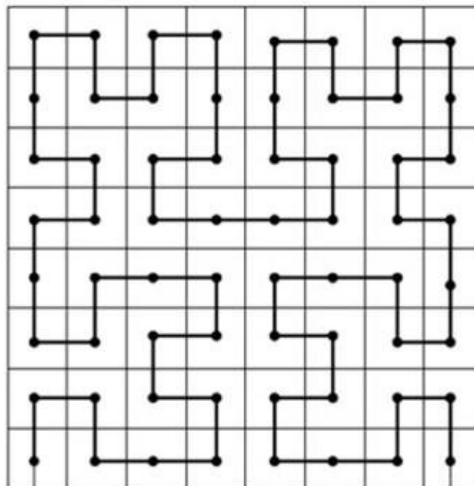


Figure 3: Third Approximation to Hilbert Curve

- The first approximation will divide square into 4 quadrant and draw curve which connect the connect point (Figure 1)
- In second again divide each quadrant and again connects the center point (Figure 2).
- In the third approximation again subdivide the quadrant. If again connects the centers of the finest level before stepping to the next level (Figure 3)

Procedure:

1. Hilbert subroutine draws the Hilbert curve.
2. It takes as parameters the depth of recursion, and dx and dy values that give the direction in which it should draw.
3. It recursively draws four smaller Hilbert curves and connects them with lines.

Program:

```
#include <stdio.h>
#define N 32
#define K 3
#define MAX N * K

typedef struct { int x; int y; } point;
void rot(int n, point *p, int rx, int ry) {
    int t;
    if (!ry) {
        if (rx == 1) {
            p->x = n - 1 - p->x;
            p->y = n - 1 - p->y;
        }
        t = p->x;
        p->x = p->y;
        p->y = t;
    }
}

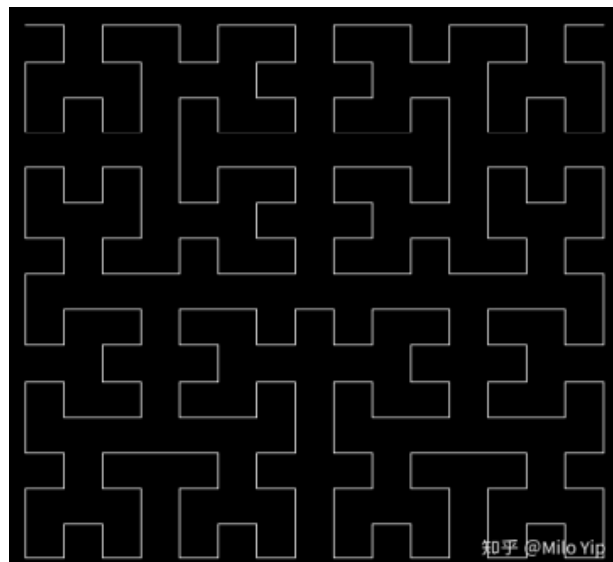
void d2pt(int n, int d, point *p) {
    int s = 1, t = d, rx, ry;
    p->x = 0;
    p->y = 0;
    while (s < n) {
        rx = 1 & (t / 2);
        ry = 1 & (t ^ rx);
        rot(s, p, rx, ry);
        p->x += s * rx;
        p->y += s * ry;
        t /= 4;
        s *= 2;
    }
}
```

```

int main() {
    int d, x, y, cx, cy, px, py;
    char pts[MAX][MAX];
    point curr, prev;
    for (x = 0; x < MAX; ++x)
        for (y = 0; y < MAX; ++y) pts[x][y] = ' ';
    prev.x = prev.y = 0;
    pts[0][0] = '.';
    for (d = 1; d < N * N; ++d) {
        d2pt(N, d, &curr);
        cx = curr.x * K;
        cy = curr.y * K;
        px = prev.x * K;
        py = prev.y * K;
        pts[cx][cy] = '.';
        if (cx == px) {
            if (py < cy)
                for (y = py + 1; y < cy; ++y) pts[cx][y] = '|';
            else
                for (y = cy + 1; y < py; ++y) pts[cx][y] = '|';
        }
        else {
            if (px < cx)
                for (x = px + 1; x < cx; ++x) pts[x][cy] = '_';
            else
                for (x = cx + 1; x < px; ++x) pts[x][cy] = '_';
        }
        prev = curr;
    }
    for (x = 0; x < MAX; ++x) {
        for (y = 0; y < MAX; ++y) printf("%c", pts[y][x]);
        printf("\n");
    }
    return 0;
}

```

Output:



Conclusion:

Viva Questions:

- 1) Define curve.
- 2) Write topological and fractal dimension of Hilbert's curve.
- 3) Define Hilbert curve and Peano curve.