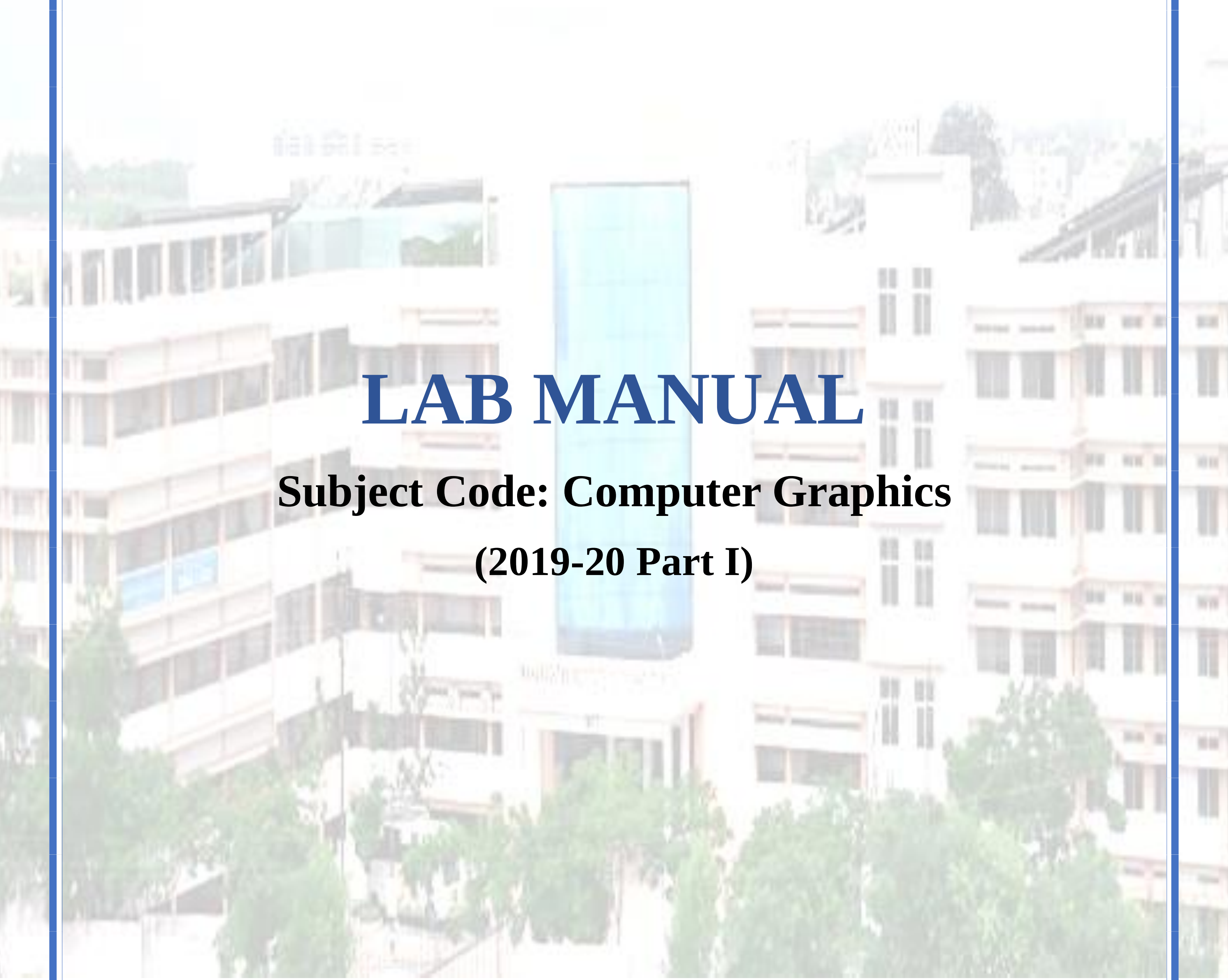
G.S. Mandal's

Maharashtra Institute of Technology, Aurangabad

Department of Computer Science and Engineering



LAB MANUAL

Subject Code: Computer Graphics

(2019-20 Part I)

Maharashtra Institute of Technology, Aurangabad
NH-211, MIT Campus, Satara Village Road, Aurangabad- 431 010 (M.S.); India.
Phone: (0240) 2375222; Fax: (0240) 2376618, E-mail: principalmitt@mit.asia
Website: www.btech.mit.asia

Department of Computer Science and Engineering

Vision

To develop the department as a center of excellence in the field of computer science and engineering by imparting knowledge & training to the students for meeting growing needs of the industry & society.

Mission

Providing quality education through a well-designed curriculum in tune with the challenging needs of software industry by providing state of the art facilities and to impart knowledge in the thrust areas of computer science and engineering.

Department of Computer Science and Engineering

Program Educational Objectives

PEO1: To prepare the students to achieve success in Computing Domain to create individual careers, innovations or to work as a key contributor to the private or Government sector and society.

PEO2: To develop the ability among the students to understand Computing and mathematical fundamentals and apply the principles of Computer Science for analyzing, designing and testing software for solving problems.

PEO3: To empower the students with ability to quickly reflect the changes in the new technologies in the area of computer software, hardware, networking and database management.

PEO4: To promote the students with awareness for lifelong learning, introduce them to professional practice, ethics and code of professionalism to remain continuous in their profession and leaders in technological society.

Program Specific Objectives

PSO1: Identify appropriate data structures and algorithms for a given contextual problem and develop programs to design and implement web applications.

PSO3: Design and manage the large databases and develop their own databases to solve real world problems and to design, build, manage networks and apply wireless techniques in mobile based applications.

PSO3: Design a variety of computer-based components and systems using computer hardware, system software, systems integration process and use standard testing tools for assuring the software quality.

Department of Computer Science and Engineering

Program Outcomes

PO1: Apply knowledge of mathematics, science, and engineering fundamentals to solve problems in Computer science and Engineering.

PO2: Identify, formulate and analyze complex problems.

PO3: Design system components or processes to meet the desired needs within realistic constraints for the public health and safety, cultural, societal and environmental considerations.

PO4: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data for valid conclusions.

PO5: Select and apply modern engineering tools to solve the complex engineering problem.

PO6: Apply knowledge to assess contemporary issues.

PO7: Understand the impact of engineering solutions in a global, economic, environmental, and societal context.

PO8: Apply ethical principles and commit to professional ethics and responsibilities.

PO9: Work effectively as an individual, and as a member or leader in diverse teams and in multidisciplinary settings.

PO10: Communicate effectively in both verbal and written form.

PO11: Demonstrate knowledge and apply engineering and management principles to manage projects and in multi-disciplinary environment.

PO12: To engage in life-long learning to adopt to the technological changes.

Department of Computer Science and Engineering

Course: **CSE204/222 Course Name: Computer Graphics**

Course Outcomes:

After Completing the course students will be able to

CO1: Describe basic concepts in computer graphics

CO2: Distinguish between line drawing, clipping and area filling algorithms

CO3: Write simple OPEN-GL program using basic primitives .

CO4: Apply 2D and 3D transformation on given object

CO5: Compare various visible surface detection algorithms

CO6: Analyze and select proper graphic technique suitable application areas

Mapping

Experiment No.	Blooms Level	Mapping To CO
1	3	CO1
2	3	CO2
3	3	CO2
4	3	CO2
5	3	CO3
6	3	CO4
7	3	CO5
8	3	CO6
9	3	CO6
10	3	CO6
11	4	CO6
12	3	CO4

Index

Contents **Page No.**

Experiment No. 01

Aim: Design and Develop simple graphics program using basic graphics functions defined in graphics.h

Theory:

In the graphic design of printed material are frequently produced on computers, as are the still and moving images seen in comic strips and animations. The realistic images viewed and manipulated in electronic games and computer simulations could not be created or supported without the enhanced capabilities of modern computer graphics. Computer graphics also are essential to scientific visualization, a discipline that uses images and colours to model complex phenomena such as air currents and electric fields, and to computer-aided engineering and design, in which objects are drawn and analyzed in computer programs. Even the windows-based graphical user interface, now a common means of interacting with innumerable computer programs, is a product of computer graphics.

What are initgraph, gd and gm?

- gd = graphdriver;
- gm = graphmode;

Syntax for initgraph:

```
void initgraph (int *graphdriver, int *graphmode, char *pathtodriver) ;
```

Description for initgraph:

initgraph

initgraph is used to initialize the graphics system by loading a graphics driver from disk and thereby putting the system into graphics mode.

To start the graphics system, we first call the initgraph function. initgraph may use a particular graphics driver and mode, or it may auto-detect and pick the corresponding driver at runtime, according to our needs.

If we tell initgraph to autodetect, it calls detectgraph to select a graphics driver and mode. It also resets all graphics settings to their defaults values like current position, color, viewport and so on and also resets graphresult to 0.

Normally, memory is allocated by initgraph to load a particular graphics driver through _graphgetmem, then it loads the appropriate BGI file from disk.

pathtodriver

pathtodriver denotes the directory path where initgraph must look for graphic drivers. initgraphfirst goes through the directed path to look for the files and if they are not found there, it goes to the current directory. The graphic driver must files must be present in the current directory if the pathtodriver is null.

graphdriver

*graphdriver is the integer that specifies which graphics driver is to be used. We can give it a value using a constant of the graphics_drivers enum type.

graphmode

*graphmode is also an integer that specifies the initial graphics mode. The table for the values of *graphmode are given in the tlink below and its values are assigned in the same way as for *graphdriver.

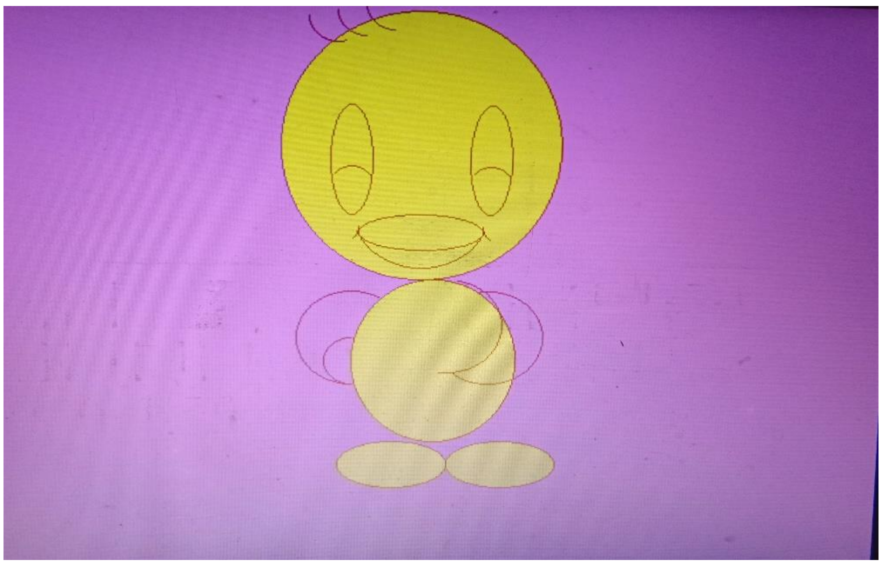
graphdriver and graphmode must be given proper values from the tables or we will get absurd and unexpected results. The exception here is when graphdriver = DETECT. In this case, initgraph sets *graphmode to the highest resolution available for the detected driver.

What is BGI?

Borland Graphics Interface (BGI) is a graphics library that is bundled with several Borland compilers for the DOS operating systems since 1987. The library loads graphic drivers (*.BGI) and vector fonts (*.CHR) from disk so to provide device independent graphics support to the programmers.

BGI is accessible in C/C++ with graphics.lib/graphics.h.

Sample Output:



Conclusion:

Thus we have created an object using basic functions in computer graphics and the output was verified

Experiment No. 2

Aim: Implement DDA & Brsenham's Line Drawing algorithm in C

Theory:

DDA Line Drawing Algorithm:

1. Calculate dx = xb – xa, dy = yb – ya and assign x = xa, y = ya.
2. a) If abs (dx)>abs (dy) then set steps in abs (dx) times.
   b) Otherwise set steps in abs (dy) times.
3. Calculate the xIncrement = dx / steps and yIncrement = dy / steps.
4. Plot the coordinate values (x, y).
5. Initially k = 0.
6. Plot the coordinate values (x + xIncrement, y + yIncrement) until k ≥ steps.

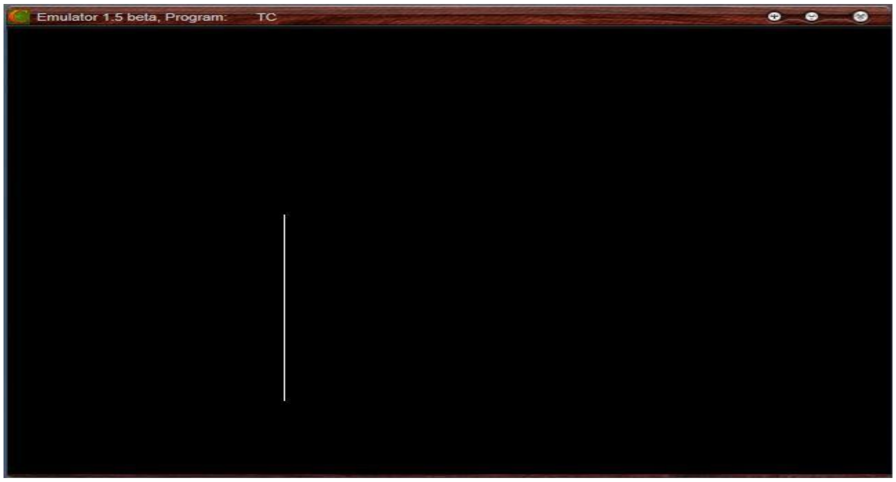
Bresenham's Line Drawing Algorithm:

1. Calculate dx = abs (xa – xb), dy = abs (ya – yb), p = 2 * dy – dx, twoDy = 2 * dy, twoDyDx = 2 * (dy - dx).
2. a) If xa > xb then set x = xb, y = yb and xEnd = xa.
   b) Otherwise set x = xa, y = ya, xEnd = xb.
3. Plot the coordinate values (x, y).
4. a) If p < 0, then plot the coordinate values (x+1, y) and set p = p + twoDy.
   b) Otherwise plot the coordinate values (x+1, y+1) and set p = p + twoDyDx.
5. Repeat the step 4 until x ≥ xEnd.

Input/Output:

Enter the start point of line (xa,ya):200,200

Enter the start point of line (xb,yb):200,400



Conclusion:

Thus the algorithms for drawing a line using DDA algorithm and Bresenham algorithm has been implemented and the output was verified

EXPERIMNET NO.03

Aim: Implement Cohen Sutherland line clipping algorithm in C/C++.

Algorithm:

Cohen Sutherland Line Clipping Algorithm:

In the algorithm, first of all, it is detected whether line lies inside the screen or it is outside the screen. All lines come under any one of the following categories:

1. Visible
2. Not Visible
3. Clipping Case

1. Visible: If a line lies within the window, i.e., both endpoints of the line lies within the window. A line is visible and will be displayed as it is.

2. Not Visible: If a line lies outside the window it will be invisible and rejected. Such lines will not display. If any one of the following inequalities is satisfied, then the line is considered invisible. Let A (x_1,y_2) and B (x_2,y_2) are endpoints of line.

x_{min},x_{max} are coordinates of the window.

y_{min},y_{max} are also coordinates of the window.

$x_1>x_{max}$

$x_2>x_{max}$

$y_1>y_{max}$

$y_2>y_{max}$

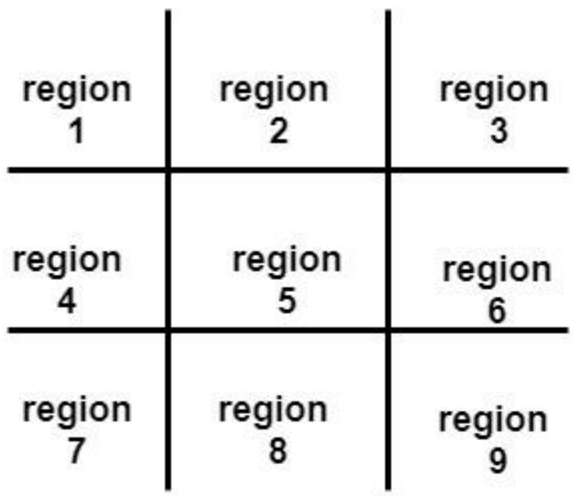
$x_1<x_{min}$

$x_2<x_{min}$

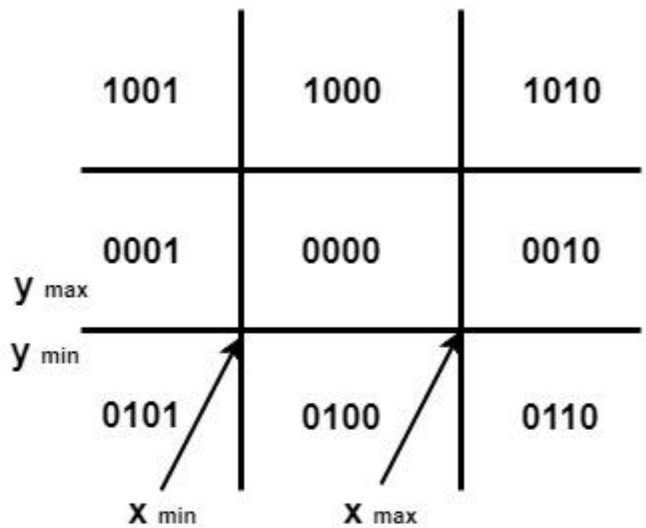
$y_1<y_{min}$

$y_2<y_{min}$

3. Clipping Case: If the line is neither visible case nor invisible case. It is considered to be clipped case. First of all, the category of a line is found based on nine regions given below. All nine regions are assigned codes. Each code is of 4 bits. If both endpoints of the line have end bits zero, then the line is considered to be visible.



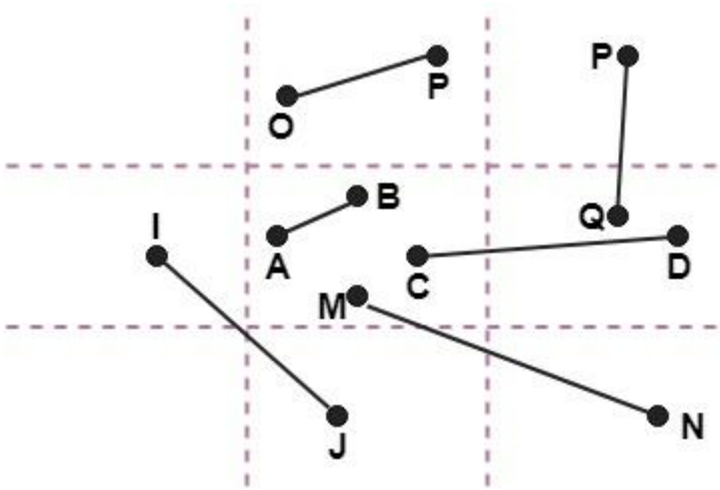
9 region



bits assigned to 9 regions

The center area is having the code, 0000, i.e., region 5 is considered a rectangle window.

Following figure show lines of various types



Line AB is the visible case

Line OP is an invisible case

Line PQ is an invisible line
Line IJ are clipping candidates
Line MN are clipping candidate
Line CD are clipping candidate

Advantage of Cohen Sutherland Line Clipping:

1. It calculates end-points very quickly and rejects and accepts lines quickly.
2. It can clip pictures much large than screen size.

Algorithm of Cohen Sutherland Line Clipping:

Step1: Calculate positions of both endpoints of the line

Step2: Perform OR operation on both of these end-points

Step3: If the OR operation gives 0000
Then
line is considered to be visible
else
Perform AND operation on both endpoints
If And ≠ 0000
then the line is invisible
else
And=0000
Line is considered the clipped case.

Step4: If a line is clipped case, find an intersection with boundaries of the window
$m=(y_2-y_1)(x_2-x_1)$

(a) If bit 1 is "1" line intersects with left boundary of rectangle window
$y_3=y_1+m(x-X_1)$
where $X = X_{wmin}$
where $X_{wmin}$is the minimum value of X co-ordinate of window

(b) If bit 2 is "1" line intersect with right boundary

$y_3=y_1+m(X-X_1)$

where $X = X_{wmax}$

where X more is maximum value of X co-ordinate of the window

(c) If bit 3 is "1" line intersects with bottom boundary

$X_3=X_1+(y-y_1)/m$

where $y = y_{wmin}$

y_{wmin} is the minimum value of Y co-ordinate of the window

(d) If bit 4 is "1" line intersects with the top boundary

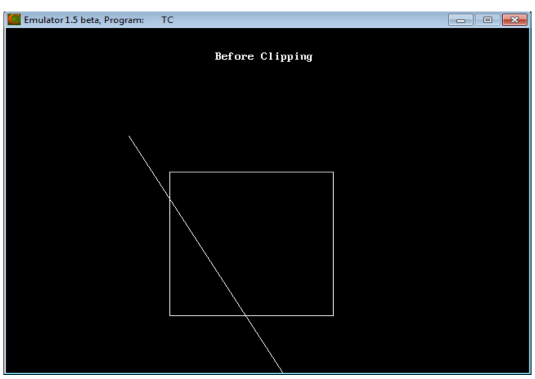
$X_{3=x}1+(y-y_1)/m$

where $y = y_{wmax}$

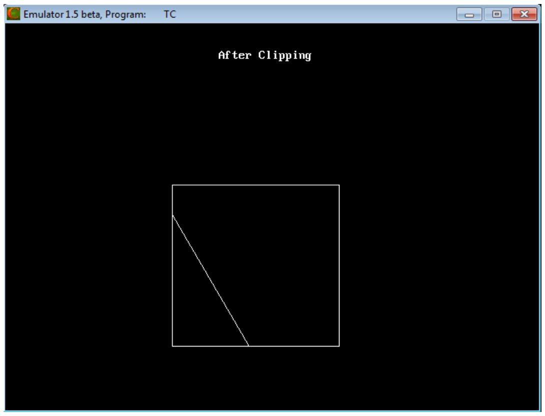
y_{wmax} is the maximum value of Y co-ordinate of the windowEnter the clip window coordinates:200 200 400 400

Enter the line coordinates: 150 150 350 500

Expected Output:-



w with clipped line



Conclusion:

Thus the Cohensan's Sutherland line clipping algorithm was implemented and the output was verified

EXPERIMENT NO 4

Aim : Implement polygon filling algorithms in C/C++.

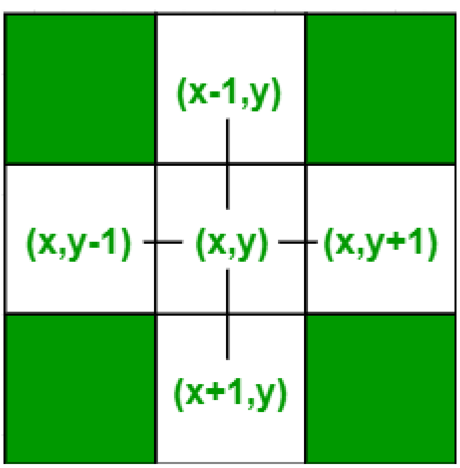
Theory

The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.

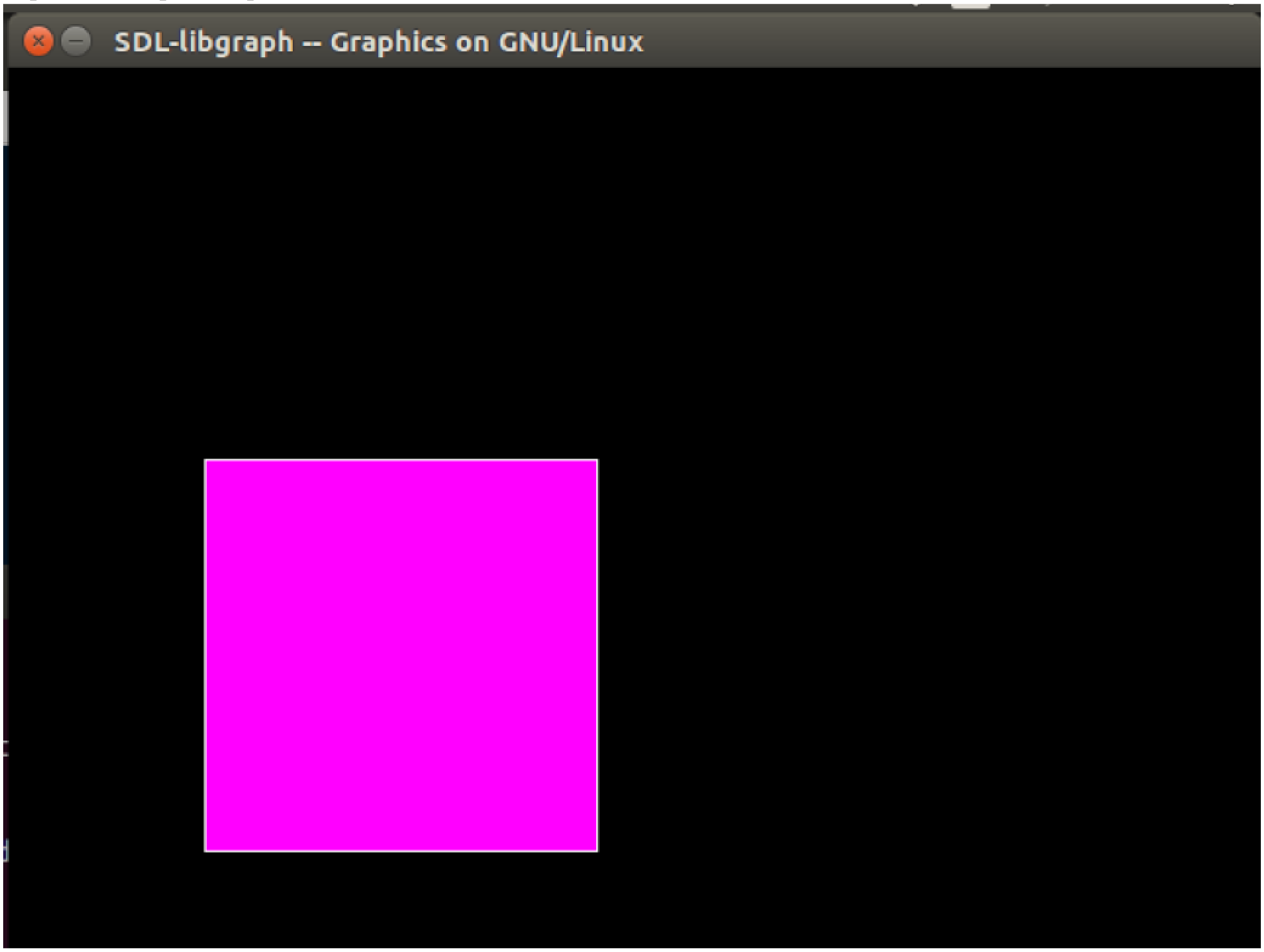
4-connected pixels : After painting a pixel, the function is called for four neighboring points. These are the pixel positions that are right, left, above and below the current pixel. Areas filled by this method are called 4-connected. Below given is the algorithm :

Algorithm :

```
Void boundaryFill4(int  x, int  y, int fill_color, int boundary_color)
{
If(getpixel(x, y) != boundary_color &&
    getpixel(x, y) != fill_color)
{
   putpixel(x,y, fill_color);
   boundaryFill4(x+1, y, fill_color, boundary_color);
   boundaryFill4(x, y+1, fill_color, boundary_color);
   boundaryFill4(x-1, y, fill_color, boundary_color);
boundaryFill4(x, y-1, fill_color, boundary_color);
}
```



Expected sample Output:



Conclusion:

Thus the implementation of polygon filling algorithm was done and the output was verified

EXPERIMENT NO.5

AIM:

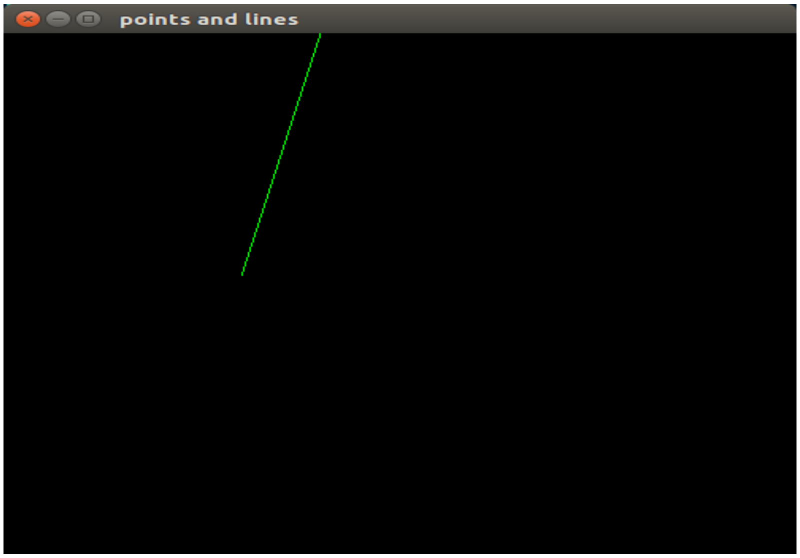
Design and develop OpenGL programs to implement basic graphics primitives

Theory:

Program:

```
#include <GL/glut.h>
void init2D(float r, float g, float b)
{
glClearColor(r,g,b,0.0);
glMatrixMode (GL_PROJECTION);
gluOrtho2D (0.0, 200.0, 0.0, 150.0);
}
void display(void)
{
int i;
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0, 0.0, 0.0);
//draw a line
glBegin(GL_LINES);
glVertex2i(10,10);
glVertex2i(100,100);
glEnd();
glFlush();
}
void main(int argc,char *argv[])
{
glutInit(&argc,argv);
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize (500, 500);
glutInitWindowPosition (100, 100);
glutCreateWindow ("points and lines");
init2D(0.0,0.0,0.0);
glutDisplayFunc(display);
glutMainLoop();
}
```

Output:



Conclusion:

Thus we have implemented the basic priitives in Open Gl to draw a line and the output was verified

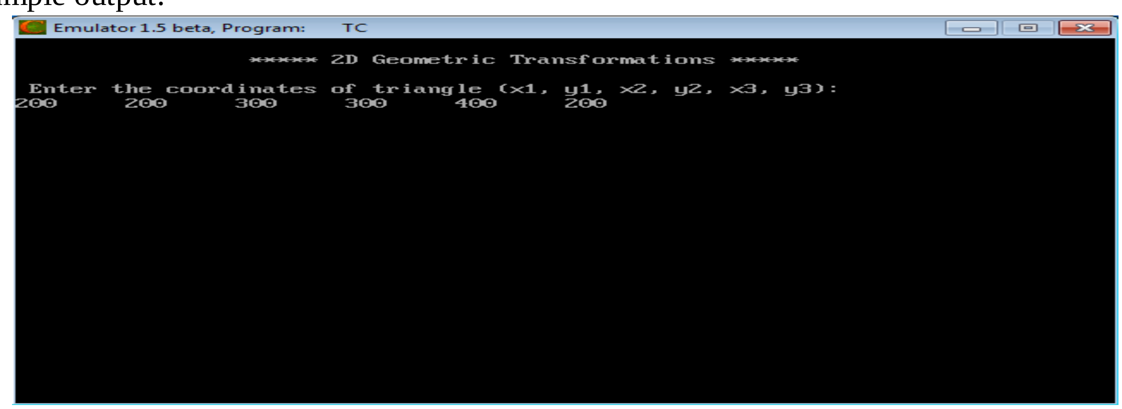
EXPERIMENT NO 6:

Aim:Write C/C++ program to draw 2D object and perform translation, rotation and scaling transformations

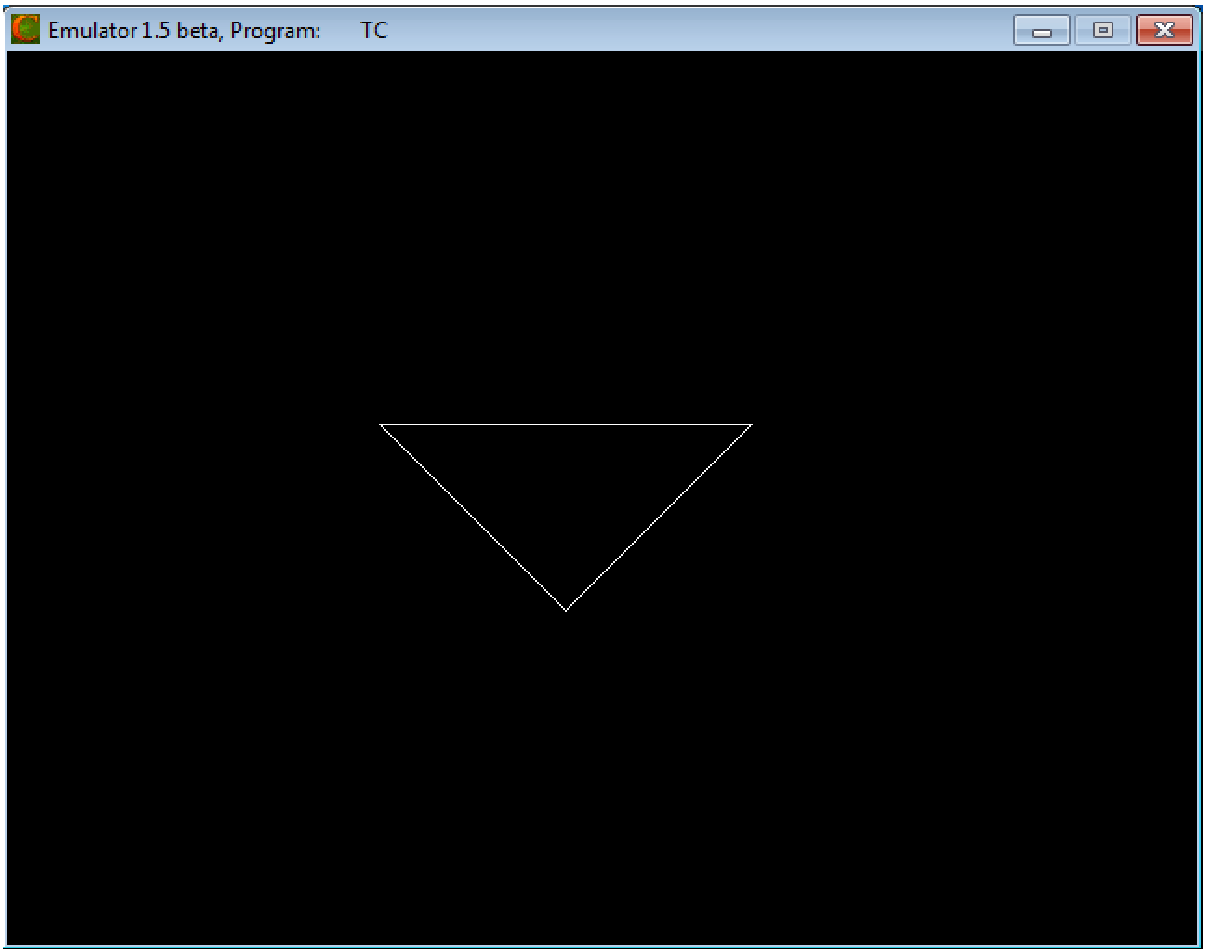
THEORY:
Algorithm:

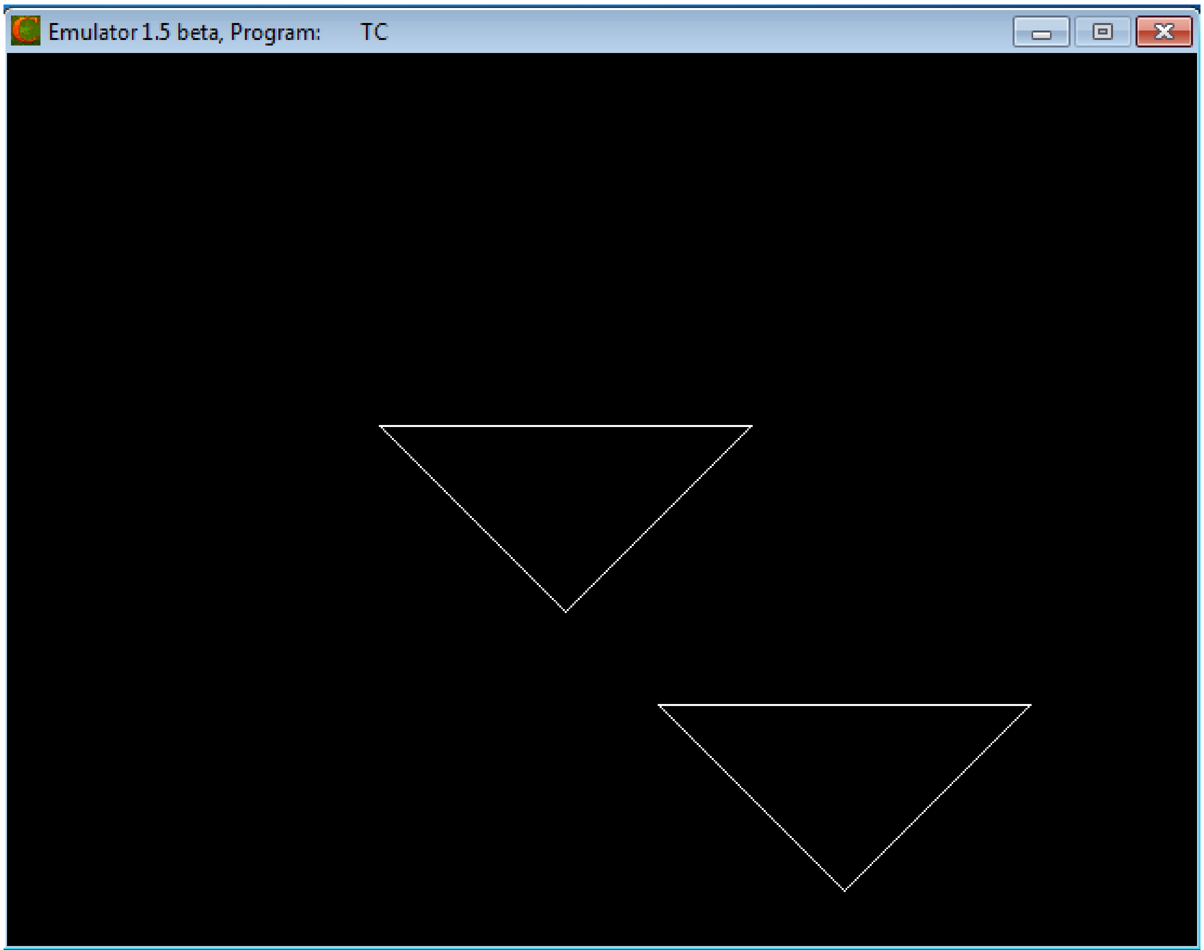
1. Get the coordinates of triangle (x1, y1, x2, y2, x3, y3).
2. Draw the original triangle.
3. Print the menu for choosing 2D Geometric Transformation.
   a. If user choose translation then get the translation factors(x, y) and draw the translated triangle in the following coordinates (x1+x, y1+y, x2+x, y2+y, x3+x, y3+y).
   b. (i) If user choose rotation then get the rotation angle (t) and reference point of the rotation (rx, ry).
      (ii) Change the t value to t = t * (3.14 / 180) and calculate the rotated coordinates by the following formulae
      rx1 = rx + (x1 - rx) * cos (t) - (y1 - ry) * sin (t); ry1 = ry + (x1 - rx) * sin (t) + (y1 - ry) * cos (t);
      (iii) Similarly calculate the coordinates rx2, ry2, rx3, ry3 and draw the rotated triangle in the following coordinates (rx1, ry1, rx2, ry2, rx3, ry3).
   c. If user choose scaling then get the scaling factors(x, y) and draw the scaleded triangle in the following coordinates (x1*x, y1*y, x2*x, y2*y, x3*x, y3*y).

Expected sample output:



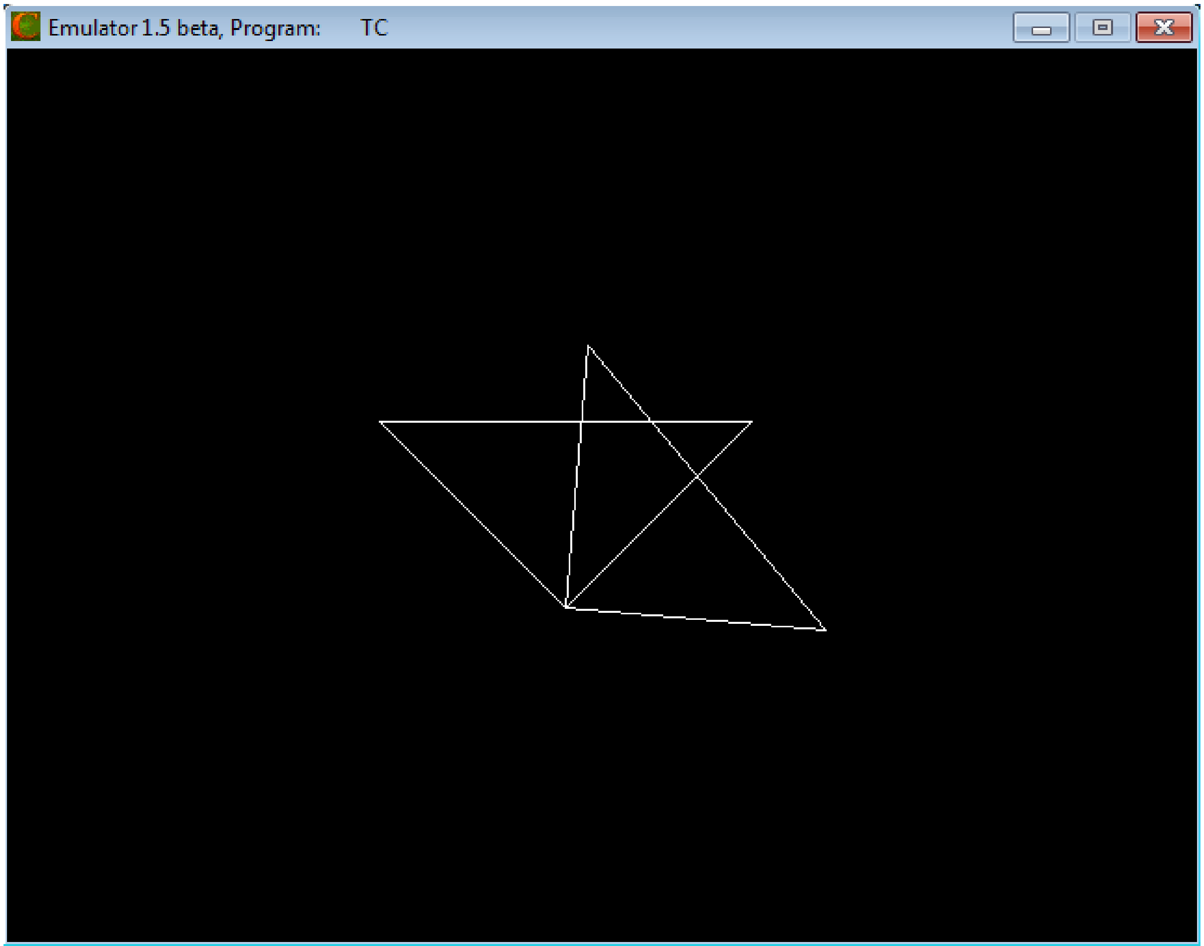
translation



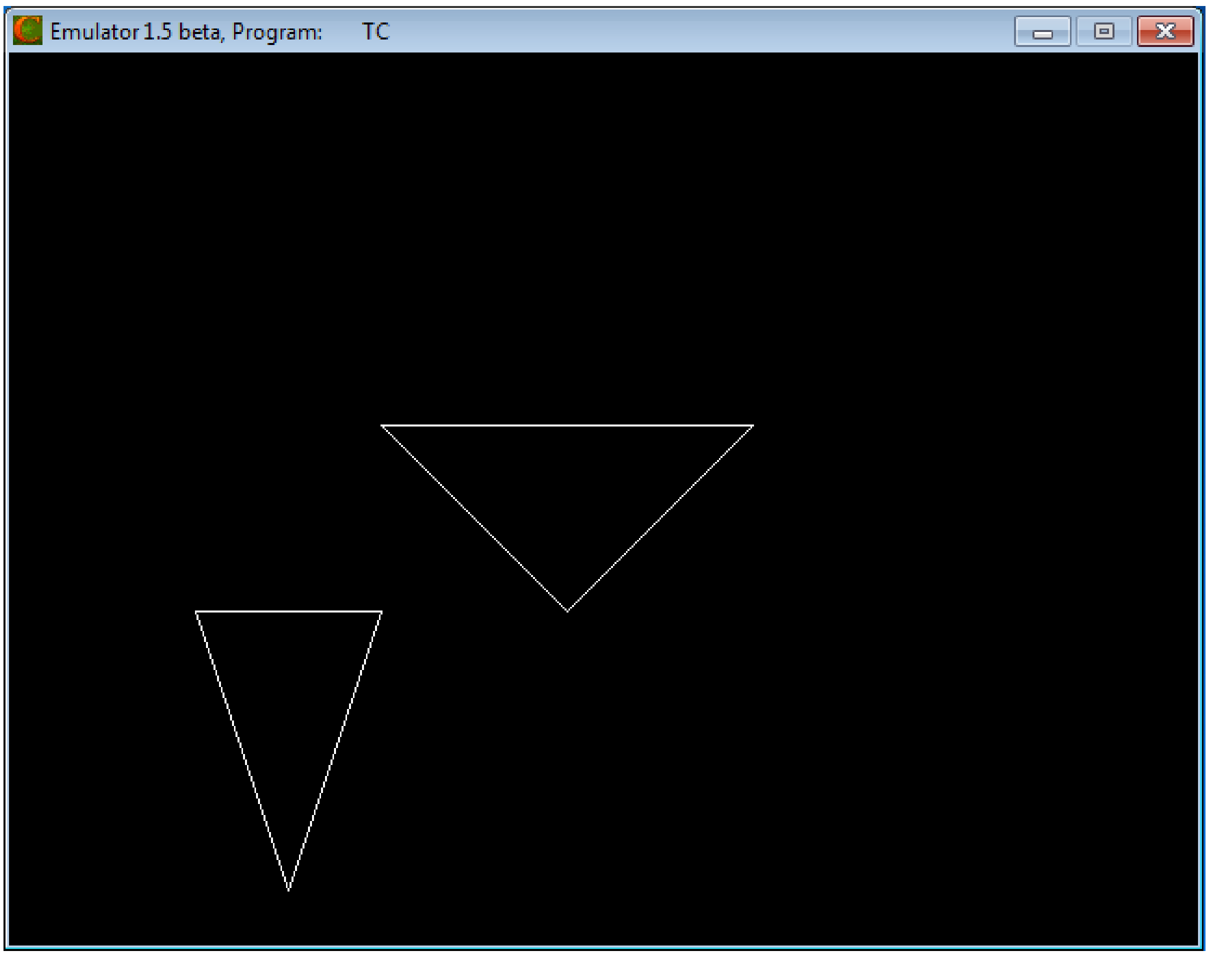


After rotation of 50 degree

With reference point 300 300



After scaling



Conclusion :Thus we have implemented program for 2D transformation on basic objects.

EXPERIMENT NO:7

AIM:- Write C/C++ program to implement any one hidden surface removal 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.

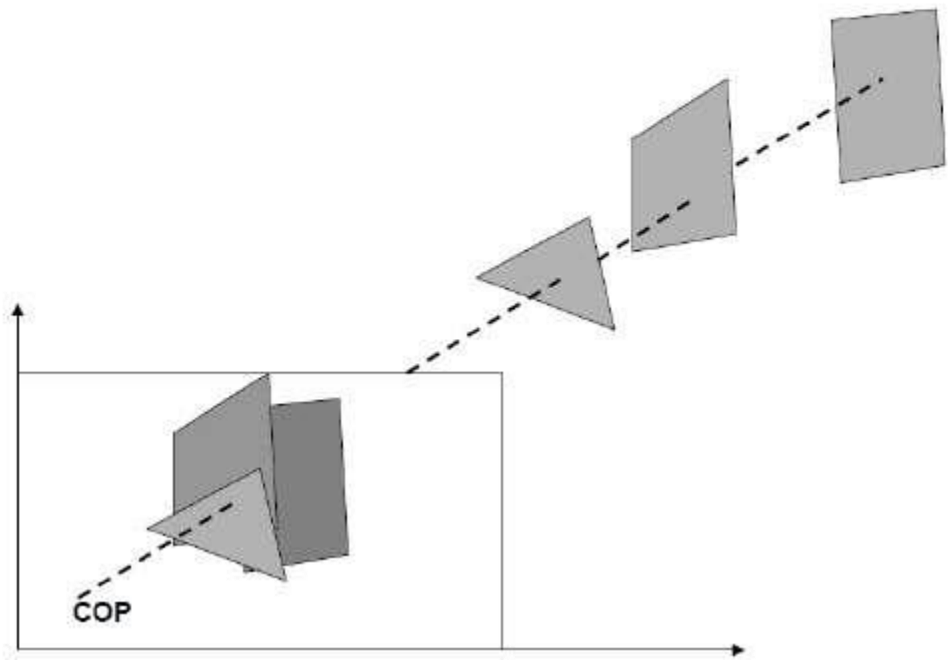
Depth Buffer *Z−Buffer* Method

This method is developed by Cutmull. It is an image-space approach. The basic idea is to test the Z-depth of each surface to determine the closest *visible* surface.
In this method each surface is processed separately one pixel position at a time across the surface. The depth values for a pixel are compared and the closest *smallestz* surface determines the color to be displayed in the frame buffer.

It is applied very efficiently on surfaces of polygon. Surfaces can be processed in any order. To override the closer polygons from the far ones, two buffers named **frame buffer** and **depth buffer,** are used.

Depth buffer is used to store depth values for *x,y* position, as surfaces are processed $0 \leq depth \leq 1$.
The **frame buffer** is used to store the intensity value of color value at each position *x,y*.

The z-coordinates are usually normalized to the range [0, 1]. The 0 value for z-coordinate indicates back clipping pane and 1 value for z-coordinates indicates front clipping pane.



Algorithm

Step-1 − Set the buffer values −

Depthbuffer *x,y* = 0
Framebuffer *x,y* = background color
Step-2 − Process each polygon *Oneatatime*
For each projected *x,y* pixel position of a polygon, calculate depth z.
If Z > depthbuffer *x,y*

Compute surface color,

set depthbuffer *x,y* = z,
framebuffer *x,y* = surfacecolor *x,y*

Advantages

- It is easy to implement.
- It reduces the speed problem if implemented in hardware.
- It processes one object at a time.

Disadvantages

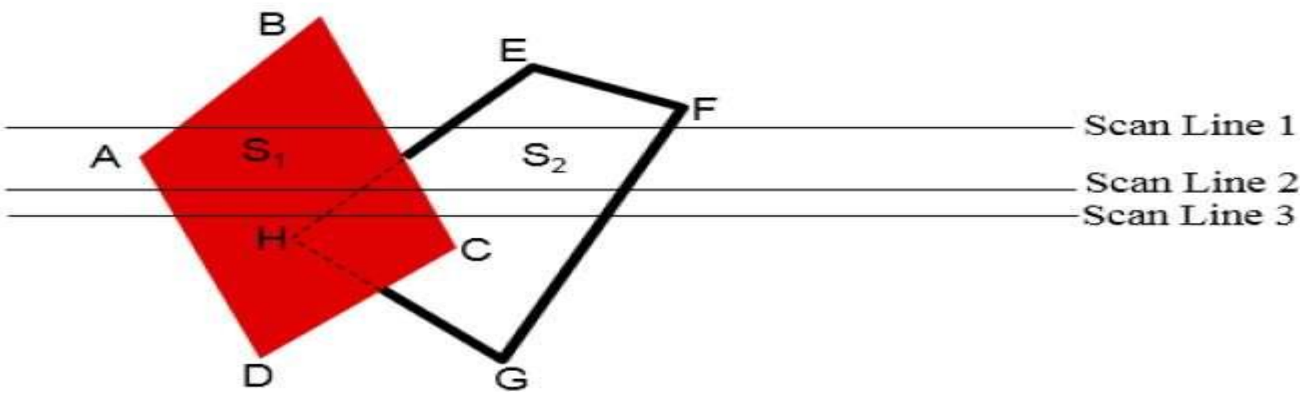
- It requires large memory.
- It is time consuming process.

Scan-Line Method

It is an image-space method to identify visible surface. This method has a depth information for only single scan-line. In order to require one scan-line of depth values, we must group and process all polygons intersecting a given scan-line at the same time before processing the next scan-line. Two important tables, **edge table** and **polygon table,** are maintained for this.

The Edge Table − It contains coordinate endpoints of each line in the scene, the inverse slope of each line, and pointers into the polygon table to connect edges to surfaces.

The Polygon Table − It contains the plane coefficients, surface material properties, other surface data, and may be pointers to the edge table.



To facilitate the search for surfaces crossing a given scan-line, an active list of edges is formed. The active list stores only those edges that cross the scan-line in order of increasing x. Also a flag is set for each surface to indicate whether a position along a scan-line is either inside or outside the surface.

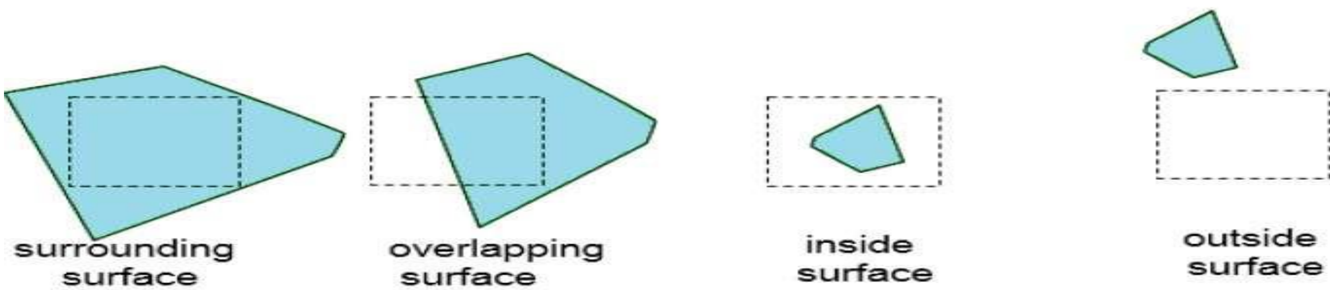
Pixel positions across each scan-line are processed from left to right. At the left intersection with a surface, the surface flag is turned on and at the right, the flag is turned off. You only need to perform depth calculations when multiple surfaces have their flags turned on at a certain scan-line position.

Area-Subdivision Method

The area-subdivision method takes advantage by locating those view areas that represent part of a single surface. Divide the total viewing area into smaller and smaller rectangles until each small area is the projection of part of a single visible surface or no surface at all.

Continue this process until the subdivisions are easily analyzed as belonging to a single surface or until they are reduced to the size of a single pixel. An easy way to do this is to successively divide the area into four equal parts at each step. There are four possible relationships that a surface can have with a specified area boundary.

- **Surrounding surface** − One that completely encloses the area.
- **Overlapping surface** − One that is partly inside and partly outside the area.
- **Inside surface** − One that is completely inside the area.
- **Outside surface** − One that is completely outside the area.



The tests for determining surface visibility within an area can be stated in terms of these four classifications. No further subdivisions of a specified area are needed if one of the following conditions is true −

- All surfaces are outside surfaces with respect to the area.
- Only one inside, overlapping or surrounding surface is in the area.
- A surrounding surface obscures all other surfaces within the area boundaries.

Conclusion:

Thus we have summarized different visible surface detection algorithms.

	G.S. MANDAL'S MAHARASHTRA INSTITUTE OF TECHNOLOGY, AURABGABAD	
	LAB WORK INSTRUCTION SHEET	
	DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING	
CLASS: S.Y. B. TECH PART: I (2019-20)	**LAB: 513(A)**	**SUBJECT: CSE-204,222Computer Graphics**

EXPERIMENT NO :08

AIM: Write C/C++ program to draw any object using any curve generation technique

Bezier Curves

Bezier curve is discovered by the French engineer **Pierre Bézier**. These curves can be generated under the control of other points. Approximate tangents by using control points are used to generate curve. The Bezier curve can be represented mathematically as −

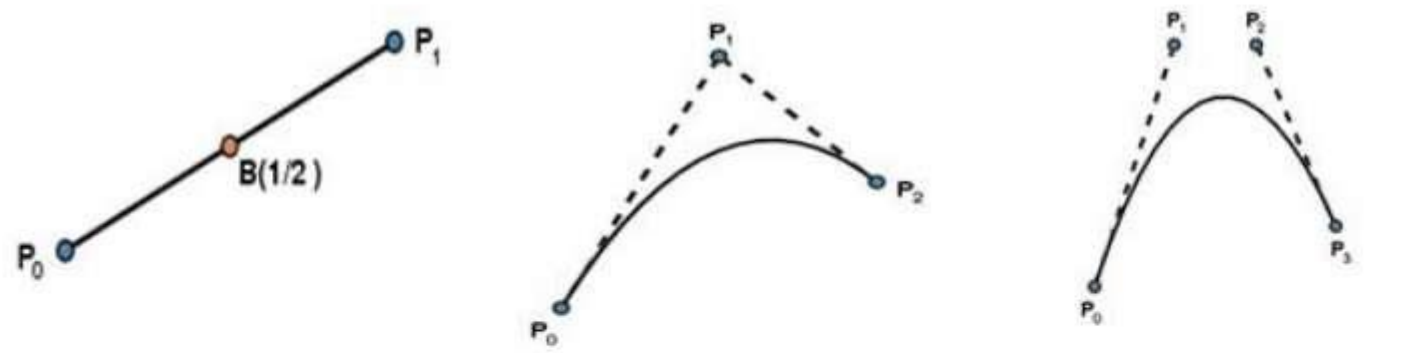
∑k=0nPiBni(t)∑k=0nPiBin(t)

Where pipi is the set of points and Bni(t)Bin(t) represents the Bernstein polynomials which are given by −

Bni(t)=(ni)(1−t)n−itiBin(t)=(ni)(1−t)n−iti

Where **n** is the polynomial degree, **i** is the index, and **t** is the variable.

The simplest Bézier curve is the straight line from the point P0P0 to P1P1. A quadratic Bezier curve is determined by three control points. A cubic Bezier curve is determined by four control points.



Simple Bezier Curve   Quadratic Bazier Curve   Cubic Bazier Curve

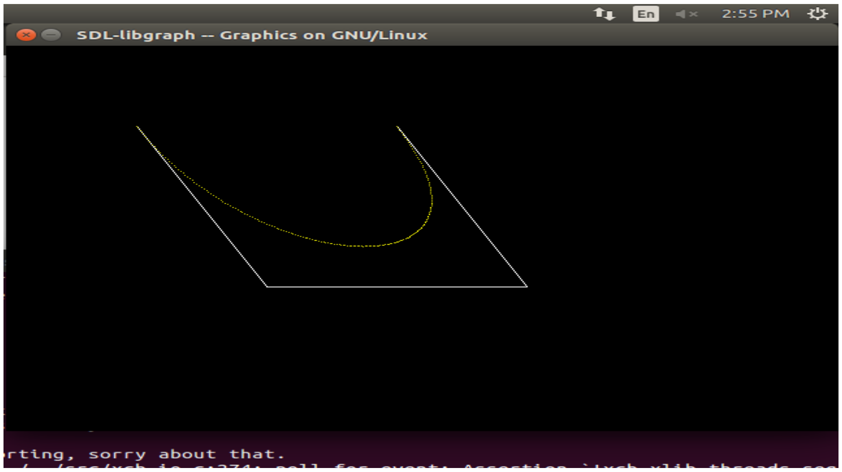
Properties of Bezier Curves

Bezier curves have the following properties −

- They generally follow the shape of the control polygon, which consists of the segments joining the control points.
- They always pass through the first and last control points.
- They are contained in the convex hull of their defining control points.

- The degree of the polynomial defining the curve segment is one less that the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is 3, i.e. cubic polynomial.
- A Bezier curve generally follows the shape of the defining polygon.
- The direction of the tangent vector at the end points is same as that of the vector determined by first and last segments.
- The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.
- No straight line intersects a Bezier curve more times than it intersects its control polygon.
- They are invariant under an affine transformation.
- Bezier curves exhibit global control means moving a control point alters the shape of the whole curve.
- A given Bezier curve can be subdivided at a point t=t0 into two Bezier segments which join together at the point corresponding to the parameter value t=t0.

Output:



Conclusion:

Thus We have implemented the program for Bezier curve generation.

EXPERIMENT NO.9

Aim: Write C/C++ program to generate snowflake using concept of fractals.

Theory: **koch Curve or Koch Snowflake**

What is Koch Curve?

The Koch snowflake (also known as the Koch curve, Koch star, or Koch island) is a mathematical curve and one of the earliest fractal curves to have been described. It is based on the Koch curve, which appeared in a 1904 paper titled "On a continuous curve without tangents, constructible from elementary geometry" by the Swedish mathematician Helge von Koch.

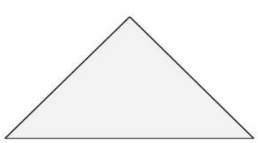
The progression for the area of the snowflake converges to 8/5 times the area of the original triangle, while the progression for the snowflake's perimeter diverges to infinity. Consequently, the snowflake has a finite area bounded by an infinitely long line.

Construction

Step1:

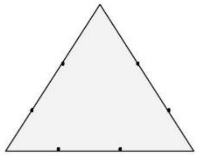
Draw an equilateral triangle. You can draw it with a compass or protractor, or just eyeball it if you don't want to spend too much time drawing the snowflake.

- It's best if the length of the sides are divisible by 3, because of the nature of this fractal. This will become clear in the next few steps.

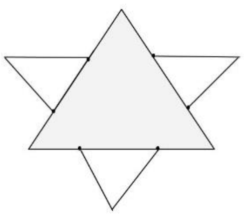


Step2:

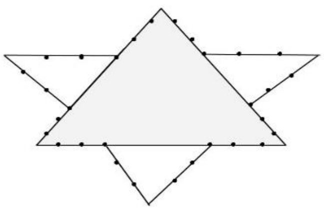
Divide each side in three equal parts. This is why it is handy to have the sides divisible by three.



Step3:
Draw an equilateral triangle on each middle part. Measure the length of the middle third to know the length of the sides of these new triangles.



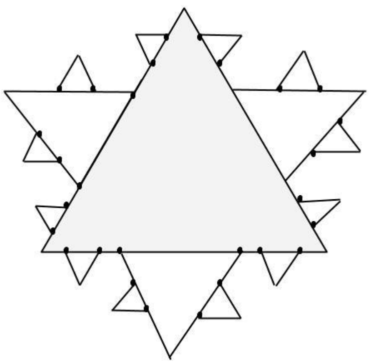
Step4:
Divide each outer side into thirds. You can see the 2nd generation of triangles covers a bit of the first. These three line segments shouldn't be parted in three.

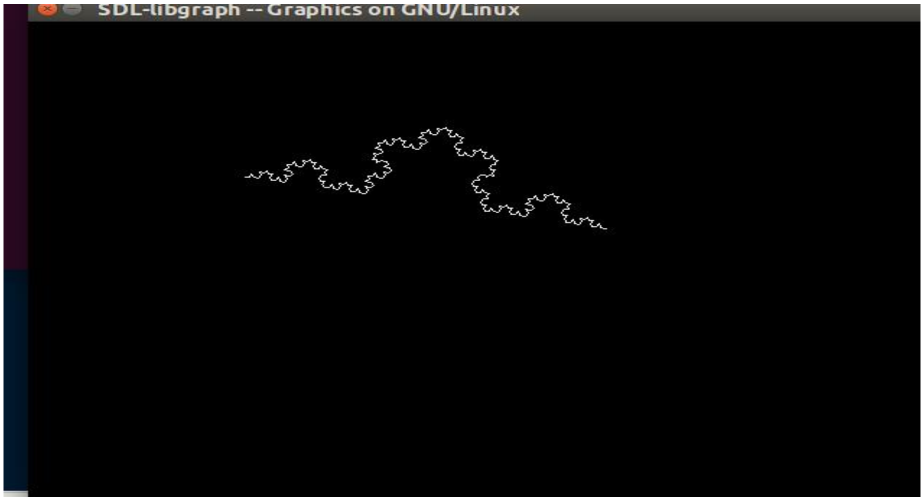


Step5:

Draw an equilateral triangle on each middle part.

- Note how you draw each next generation of parts that are one 3rd of the mast one.





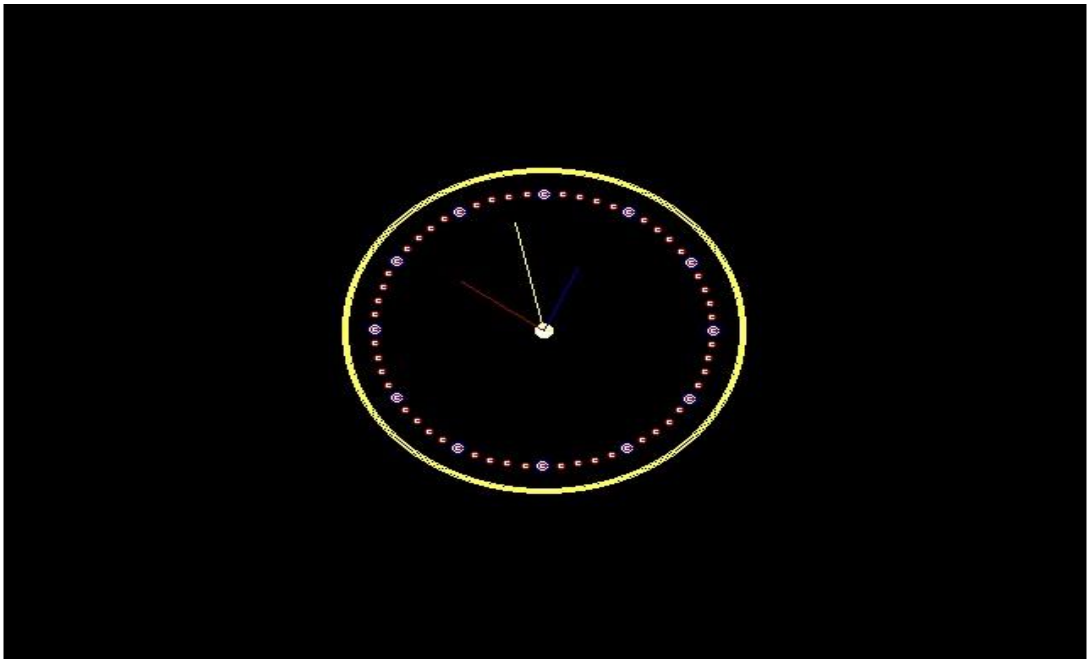
Conclusion :Thus we have implemented the program for snowflake generation.

EXPERIMENT NO 10

Aim:Write a C program TO simulate any one of or similar object

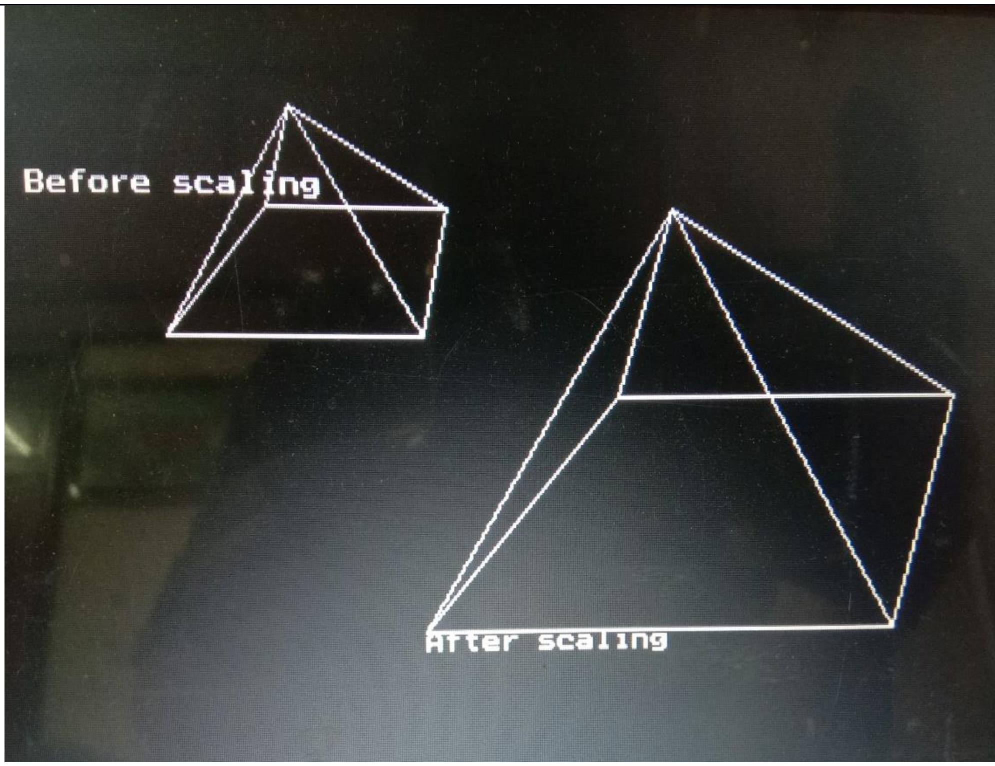
- **Chess / Ludo Board**
- **Mickey Mouse**
- **Moving 3d Object in free space**
- **Analog clock**
- **Tower of Hanoi**

Expected Sample output:



EXPERIMENT NO.12 : WRITE A PROGRAM FOR 3 D TRANSFORMAION

Output:

: 

Conclusion:

Thus we have implemented the program to apply 3d transformation on given object.