



**SINHGAD TECHNICAL EDUCATION SOCIETY'S**  
**Sinhgad College of Engineering**

**DEPARTMENT OF COMPUTER ENGINEERING**

**LAB MANUAL**

**Computer Graphics Lab**

**(2015 Course)**

**S.E.COMPUTER**

**Teaching Scheme**

**Practical: 2Hrs/ Week**

**Examination Scheme**

**Term Work: 25 Marks**

**Practical : 50 Marks**

**Credit :01**

**Version: 1.0**

**Year: 2016-17**

**SINHGAD COLLEGE OF ENGINEERING**  
**DEPARTMENT OF COMPUTER ENGINEERING**

## SUBJECT: Computer Graphics Lab

**Teaching Scheme:**

**Practical: 2 hrs. / week**

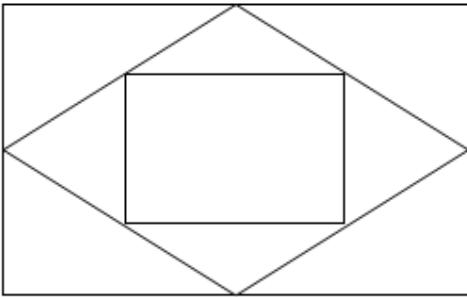
**Examination Scheme:**

**Term work: 50 marks**

**Oral: 50 marks**

**Credit : 01**

### LIST OF ASSIGNMENTS

Assignment No.	Title
<b>Group A</b>	
1.	Write C++/Java program to draw line using DDA and Bresenham's algorithm. Inherit pixel class and Use function overloading.
2.	Write C++/Java program to draw circle using Bresenham's algorithm. Inherit pixel class.
3.	Write C++/Java program to draw 2-D object and perform following basic transformations, a) Scaling b) Translation c) Rotation Use operator overloading.
4.	Write C++/Java program to fill polygon using scan line algorithm. Use mouse interfacing to draw polygon.
5.	Write C++/Java program to draw the following pattern using any Line drawing algorithms. 
<b>Group B</b>	
1.	Write C++/Java program for line drawing using DDA or Bresenham's algorithm with patterns such as solid, dotted, dashed, dash dot and thick.
2.	Write C++/Java program to draw a concave polygon and fill it with desired pattern using scan line algorithm. Use mouse interfacing to draw polygon.
3.	Write C++/Java program to implement reflection of 2-D object about X axis, Y axis and about X=Y axis. Also rotate object about arbitrary point given by

	user.
4.	Write C++/Java program to implement translation, sheer, rotation and scaling transformations on equilateral triangle and rhombus.
5.	Write C++/Java program to draw any object such as flower, waves using any curve generation techniques
<b>Group C</b>	
1	Write C++/Java program to draw 3-D cube and perform following transformations on it using OpenGL. a) Scaling b) Translation c) Rotation about one axis
2	Write C++/Java program to simulate any one of or similar scene- <ul style="list-style-type: none"> <li>• Clock with pendulum</li> <li>• National Flag hoisting</li> <li>• Vehicle/boat locomotion</li> <li>• Water drop falling into the water and generated waves after impact</li> <li>• Kaleidoscope views generation (at least 3 colorful patterns)</li> </ul>

Prof. S.P. Bholane

Prof. D. D. Gatade

Prof. D. R. Pawar

Subject In-Charges

Prof. D. D. Gatade

Subject Coordinator

Prof. M. P. Wankhade

Head of Department

## Computer Graphics Lab

- **Course Objectives:**

- 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 mathematics behind the graphical transformations
- To understand and apply various methods and techniques regarding projections, animation, shading, illumination and lighting

- **Course Outcomes:**

On completion of the course, student will be able to–

- Apply mathematics and logic to develop Computer programs for elementary graphic operations
- Develop scientific and strategic approach to solve complex problems in the domain of Computer Graphics
- Develop the competency to understand the concepts related to Computer Vision and Virtual reality
- Apply the logic to develop animation and gaming programs

## Assignment No: 1 A

### Title: Line Drawing

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

#### Theory:

A line in Computer Graphics typically refers to a line segment, which is portion of straight line that extends indefinitely in opposite directions.

Equation of line is :  $y = mx + b$ ,

where  $m$  = slope of line

$b$  = y intercept of line.

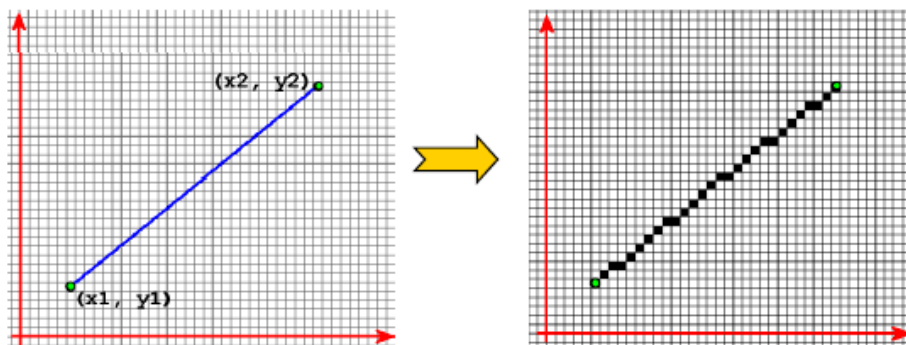


Fig : scan converting line

Line Drawing is accomplished by calculating intermediate point co-ordinates along the line path between two given end points.

Line Drawing Algorithms:

1. DDA ( digital Differential Algorithm )
2. Bresenham's Line Drawing

#### A. DDA :

1. **Accept**  $i/p = (x_s, y_s) \& (x_e, y_e)$
2. If  $\text{abs}(x_e - x_s) \geq \text{abs}(y_e - y_s)$   
Then  $\text{step} = \text{abs}(x_e - x_s)$   
Else  
 $\text{step} = \text{abs}(y_e - y_s)$
3. Find increment in x and y

$$\Delta x = (x_e - x_s)/\text{step}$$

$$\Delta y = (y_e - y_s)/\text{step}$$

4. Initialize start point

$$x = x_s$$

$$y = x_s$$

- 5.

Plot(round(x),round(y),1)

$$X = x + \Delta x$$

$$Y = y + \Delta y$$

6. Repeat step 5 till steps.

### **B. Bresenham's Algorithm :**

**Main Idea:** Move across the x-axis by one unit intervals and at each step choose between two different 'y' coordinates.

#### Algorithm

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  $\Delta x$ ,  $\Delta y$ ,  $2\Delta y$ , and  $(2\Delta y - 2\Delta x)$  and get the first value for the decision parameter as:  $P_0 = 2\Delta y - \Delta x$
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  $(\Delta x)$  times

## Assignment No: 2 A

### Title: Bresenham's Circle

**Problem Statement:** Write a C++ class for a circle drawing inheriting line class.

### Theory:

A circle is symmetrical figure. Any circle generating algorithm can take advantage of circle's symmetry to plot 8 points for each value that the algorithm calculates.

**8-way symmetry** is used by reflecting each calculated point around each 45 degree axis.

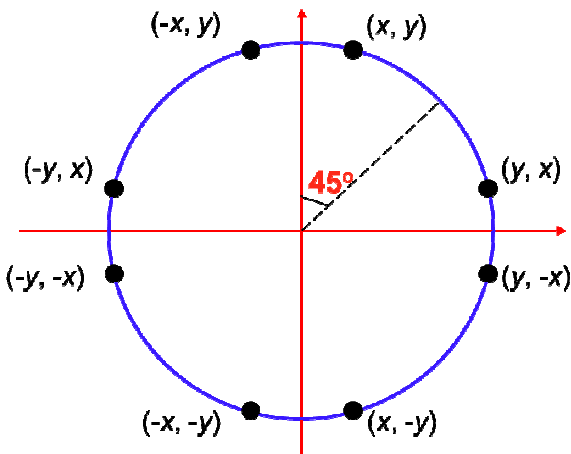


Fig: 8 way Symmetry of Circle

**Defining Circle:** A circle is defined as the set of points that are all at a given distance  $r$  from a center position  $(x_c, y_c)$ .

### Circle representation :

Cartesian form :  $x^2 + y^2 = r^2$

Polar form :  $x = r \cos \theta$

$$y = r \sin \theta$$

### Bresenham's Circle Drawing Algorithm:

**Idea :** Increment  $x$  by one unit and find corresponding  $y$  close to true circle path.

A better algorithm for generating circle has been developed by Bresenham's . Coordinates on the periphery of a circle which is centered at origin are computed for the 1/8 th part of the circle and remaining pixels are computed by using 8 way symmetry property of the circle.

**Algorithm:**

1. read radius  $r$
2.  $d = 3 - 2r$
3.  $x = 0$  and  $y = r$   
    *do {*  
         $Plot(x, y)$   
         $Plot(y, x)$   
         $Plot(y, -x)$   
         $Plot(x, -y)$   
         $Plot(-x, -y)$   
         $Plot(-y, -x)$   
         $Plot(-y, x)$   
         $Plot(-x, y)$   
    *if  $d < 0$  then*  
         $d = d + 4x + 6$   
    *else if  $\{ d \geq 0 \text{ then } d = d + 4(x - y) + 10 \text{ } y = y - 1 \}$*   
     $x++;$   
    *} while  $(x \leq y)$*
4. stop.



### Assignment No:04 A

**Problem Statement:** Write a Java program to fill polygon using Scan line polygon filling algorithm.

**Theory:**

**Polygon filling Algorithms:**

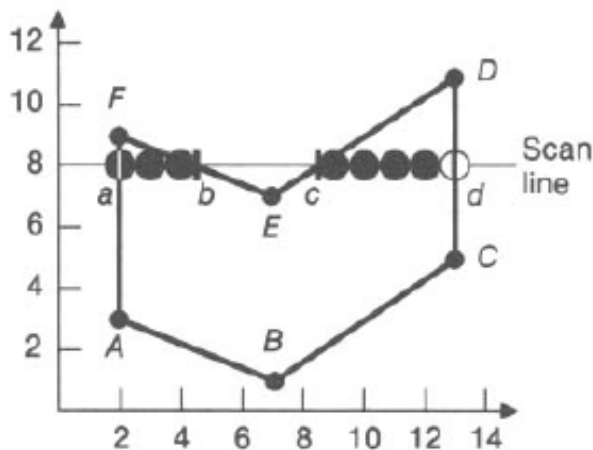
1. Flood fill
2. Boundary fill
3. Scan line polygon filling

**Scan line polygon filling algorithm :**

**Advantages:**

1. Faster
2. Less memory needed

**Algorithm :**



For each scan line:

1. Find the intersections of the scan line with all edges of the polygon.
2. Sort the intersections by increasing x-coordinate.
3. Fill in all pixels between pairs of intersections by line command .

**Mathematical Model of Polygon filling :**

n->no. of vertices of the polygon

$$\text{Filled polygon} = \left\{ \begin{array}{l} \{ \text{line} \langle p_i, p_{i+1} \rangle \mid p_i = \langle x_i, y_i \rangle \text{ and } p_{i+1} = \langle x_{i+1}, y_{i+1} \rangle \} \text{ where } 1 \leq i < n \\ \{ \text{line} \langle p_n, p_1 \rangle \mid p_n = \langle x_n, y_n \rangle \text{ and } p_1 = \langle x_1, y_1 \rangle \} \\ \text{And } P(x,y) = 1 \text{ if } P \text{ lies inside polygon} \\ \phantom{\text{And } } = 0 \text{ otherwise} \end{array} \right.$$

## Assignment No: 1B

**Title:** Line Styles in Qt Creator.

**Problem Statement:** Write a program in C/C++ to draw a line with line style (Thick, Thin, Dotted)

### Theory:

A line in Computer Graphics typically refers to a line segment, which is portion of straight line that extends indefinitely in opposite directions.

Equation of line is,

$$y = mx + b,$$

where  $m$  = slope of line

$b$  = y intercept of line.

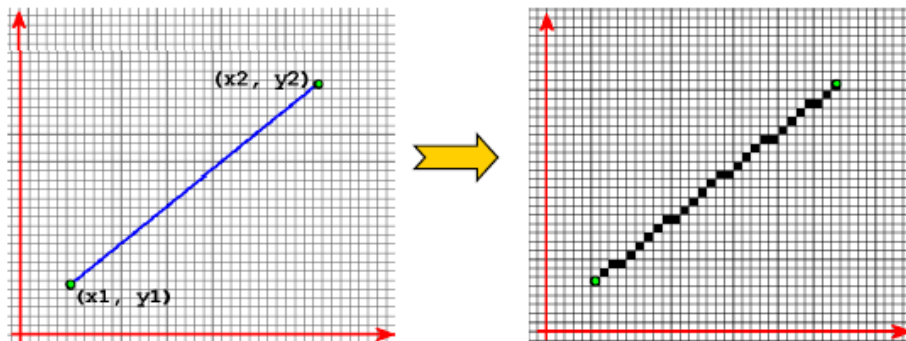


Fig : scan converting line

### Line Drawing with styles

1. Thick
2. Dotted
3. Dashed

### Dotted line :

General line drawing algorithm can be modified to display dotted line. Alternate pixels can be plotted to display dotted line.

### **Dashed line :**

In general line drawing algorithm alternate group of pixels are plotted to display dashed line.

### **Thick Line :**

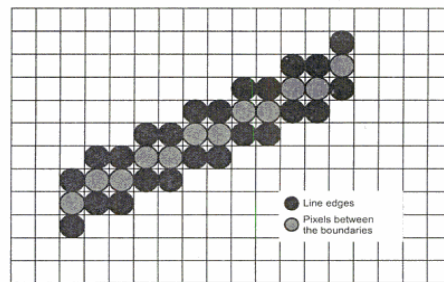


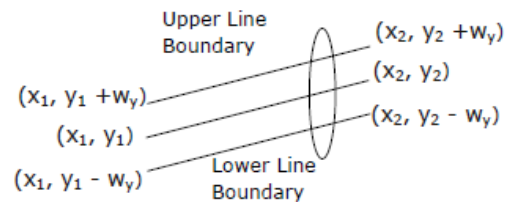
Fig. 2.19 Thick line

Let us assume a line with co-or  $(x_1, y_1)$  and  $(x_2, y_2)$  and width  $w$ .

Upper and lower line boundaries are :

$[(x_1, y_1 + w_y), (x_2, y_2 + w_y)]$  and  $[(x_1, y_1 - w_y), (x_2, y_2 - w_y)]$  respectively.

$$w_y = \frac{w-1}{2} \cdot \frac{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}{|x_2 - x_1|}$$



### **Algorithm :**

1. Accept line co-ordinates and thickness  $w$
2. If the slope of the line is  $< 1$  then increment  $x$  by 1 always and find  $y$
3. Find  $w_y$
4. Draw parallel line segments from center of the thick line till you reach the bottom and top boundaries.
5. Stop

## Assignment No: 2B

**Title:** Polygon drawing in Java

**Problem Statement:** Write a Java program to draw a simple polygons (Square, Rectangle, Triangle)

**Theory:**

**Polygon :** It is a closed polyline.

**Types of Polygon :**

1. **Convex Polygon :** A convex polygon is a polygon such that for any two points inside the polygon, all points on the line segment connecting them are also inside the Polygon.



2. **Concave Polygon :** A concave polygon is a polygon such that for any two points inside the polygon, if some points on the line segment connecting them are not inside the Polygon.



**Graphics in java :** To do custom graphics in a JAVA application, write a new class that extends the [JPanel](#) class. In that class, override the definition of the `paintComponent()` method.

**A Custom Graphics Template:**

```
import java.awt.*;
```

```
import java.awt.event.*;
```

```
import javax.swing.*;
```

```
import java.swing.event.*;

public class ClassName extends JPanel {

    public void paintComponent(Graphics g) {          super.paintComponent(g);

    }

}
```

### Java Swing:

It is java framework. Java Swing is a lightweight Java graphical user interface (GUI) widget toolkit that includes a rich set of widgets. It includes several packages for developing rich desktop applications in Java. Swing includes built-in controls such as trees, image buttons, tabbed panes, sliders, toolbars, color choosers, tables, and text areas to display HTTP or rich text format (RTF).

**Scanner class** simplifies console input. The Scanner class is a class in java.util, which allows the user to read values of various type.

1. **import java.awt.\*** : AWT stands for **Abstract Window Toolkit**. The Abstract Window Toolkit provides many classes for programmers to use. It is the connection between the application and the GUI. It contains classes that programmers can use to make graphical components, e.g., buttons, labels, frames
2. **import.java.util.\*** : The java.util package contains classes that deal with collections, events, date and time, internationalization and various helpful utilities.
3. **Import.java.swing.\*** : Swing is built on top of AWT, and provides a new set of more sophisticated graphical interface components.
4. **Import.java.swing.event.\*** : This package defines classes and interfaces used for event handling in the AWT.

Here **Jpanel** is the **canvason** which drawing is done.

**paintComponent()** method is pre-defined. If we want to draw something on the panel, then you need to override it.

A **Graphics object**g controls the visual appearance of a Swing component.

Graphics object is obtained as a parameter to the paintComponent() method. Once you have the Graphics object, you can send it messages to change the color or font that it uses, or to draw a variety of geometric figures.

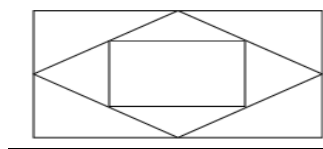
**super.paintComponent(g)** invokes the paintComponent method from the superclass of JPanel (the JComponent class) to erase whatever is currently drawn on the panel. **This is useful for animation.**

### **JFrame:**

A Frame is a top-level window with a title and a border. A frame, implemented as an instance of the [JFrame](#) class, is a window that has decorations such as a border, a title, and supports button components that close or magnify the window. Applications with a GUI usually include at least one frame.

### **Assignment No. : 4**

Draw the following pattern using any Line drawing algorithms.



#### **Aim**

Draw the following pattern using any Line drawing algorithms.

#### **Objective(s)**

<b>1</b>	To Learn DDA line drawing algorithm
<b>2</b>	To learn Bresenham line drawing algorithm

#### **Theory**

A line connects two points. It is a basic element in graphics. To draw a line, you need two points between which you can draw a line. In the following three algorithms, we refer the one point of line as  $X_0, Y_0$  and the second point of line as  $X_1, Y_1$ .

#### **DDA Algorithm**



Digital Differential Analyzer (DDA) algorithm is the simple line generation algorithm which is explained step by step here.

**Step 1** – Get the input of two end points  $(X_0, Y_0)$  and  $(X_1, Y_1)$ .

**Step 2** – Calculate the difference between two end points.

$$dx = X_1 - X_0$$

$$dy = Y_1 - Y_0$$

**Step 3** – Based on the calculated difference in step-2, you need to identify the number of steps to put pixel. If  $dx > dy$ , then you need more steps in x coordinate; otherwise in y coordinate.

```
if (absolute(dx) > absolute(dy))
```

```
    Steps = absolute(dx);
```

```
else
```

```
    Steps = absolute(dy);
```

**Step 4** – Calculate the increment in x coordinate and y coordinate.

```
Xincrement = dx / (float) steps;
```

```
Yincrement = dy / (float) steps;
```

**Step 5** – Put the pixel by successfully incrementing x and y coordinates accordingly and complete the drawing of the line.

```
for(int v=0; v < Steps; v++)
```

```
{
```

```
    x = x + Xincrement;
```

```
    y = y + Yincrement;
```

```
    putpixel(Round(x), Round(y));
```

```
}
```

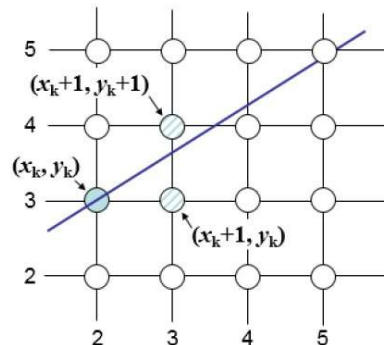
### Difference Between DDA Line Drawing Algorithm and Bresenham's Line Drawing Algorithm

	Digital Differential Analyzer	Bresenham's Line Drawing Algorithm
<b>Arithmetic</b>	DDA algorithm uses <b>floating points</b> i.e. <b>Real Arithmetic</b> .	Bresenham's algorithm uses <b>fixed points</b> i.e. <b>Integer Arithmetic</b> .
<b>Operations</b>	DDA algorithm uses <b>multiplication</b> and <b>division</b> in its operations.	Bresenham's algorithm uses only <b>subtraction</b> and <b>addition</b> in its operations.
<b>Speed</b>	DDA algorithm is rather <b>slowly</b> than Bresenham's algorithm in line drawing because it uses real arithmetic (floating-point operations).	Bresenham's algorithm is faster than DDA algorithm in line drawing because it performs only addition and subtraction in its calculation and uses only integer arithmetic so it runs significantly <b>faster</b> .
<b>Accuracy &amp; Efficiency</b>	DDA algorithm is not as accurate and efficient as Bresenham's algorithm.	Bresenham's algorithm is more efficient and much accurate than DDA algorithm.
<b>Drawing</b>	DDA algorithm can draw circles and curves but that are not as accurate as Bresenham's algorithm.	Bresenham's algorithm can draw circles and curves with much more accuracy than DDA algorithm.
<b>Round Off</b>	DDA algorithm round off the coordinates to integer that is nearest to the line.	Bresenham's algorithm does not <b>round off</b> but takes the incremental value in its operation.
<b>Expensive</b>	DDA algorithm uses an enormous number of floating-point multiplications so it is expensive.	Bresenham's algorithm is less expensive than DDA algorithm as it uses only addition and subtraction.

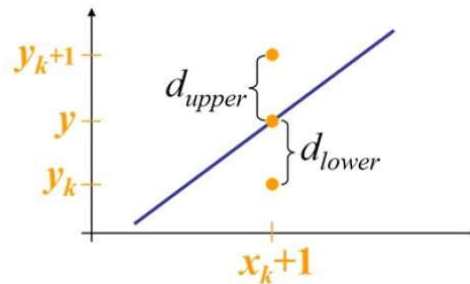
### Bresenham's Line Generation

The Bresenham algorithm is another incremental scan conversion algorithm. The big advantage of this algorithm is that, it uses only integer calculations. Moving across the x axis in unit intervals and at each step choose between two different y coordinates.

For example, as shown in the following illustration, from position (2, 3) you need to choose between (3, 3) and (3, 4). You would like the point that is closer to the original line.



At sample position  $x_{k+1}$ , the vertical separations from the mathematical line are labelled as  $d_{upper}$  and  $d_{lower}$ .



From the above illustration, the y coordinate on the mathematical line at  $x_{k+1}$  is –

$$Y = m(X_{k+1}) + b$$

So,  $d_{upper}$  and  $d_{lower}$  are given as follows –

$$\begin{aligned} d_{lower} &= y - y_k \\ &= m(X_{k+1}) + b - Y_k = m(X_{k+1}) + b - Y_k \end{aligned}$$

and

$$\begin{aligned} d_{upper} &= (y_{k+1}) - y \\ &= Y_{k+1} - m(X_{k+1}) - b = Y_{k+1} - m(X_{k+1}) - b \end{aligned}$$

You can use these to make a simple decision about which pixel is closer to the mathematical line. This simple decision is based on the difference between the two pixel positions.

$$d_{lower} - d_{upper} = 2m(x_{k+1}) - 2y_k + 2b - 1 \quad d_{lower} - d_{upper} = 2m(x_{k+1}) - 2y_k + 2b - 1$$

Let us substitute  $m$  with  $dy/dx$  where  $dx$  and  $dy$  are the differences between the end-points.

$$\begin{aligned} dx(d_{lower}-d_{upper}) &= dx(2dyx(x_k+1)-2y_k+2b-1) \\ dx(d_{lower}-d_{upper}) &= dx(2dyx(x_k+1)-2y_k+2b-1) \\ &= 2dy.x_k - 2dx.y_k + 2dy + 2dx(2b-1) = 2dy.x_k - 2dx.y_k + 2dy + 2dx(2b-1) \\ &= 2dy.x_k - 2dx.y_k + C = 2dy.x_k - 2dx.y_k + C \end{aligned}$$

So, a decision parameter  $P_k$  for the  $k$ th step along a line is given by –

$$\begin{aligned} P_k &= dx(d_{lower}-d_{upper}) \\ P_k &= dx(d_{lower}-d_{upper}) \\ &= 2dy.x_k - 2dx.y_k + C = 2dy.x_k - 2dx.y_k + C \end{aligned}$$

The sign of the decision parameter  $P_k$  is the same as that of  $d_{lower}-d_{upper}$ .

If  $P_k$  is negative, then choose the lower pixel, otherwise choose the upper pixel.

Remember, the coordinate changes occur along the  $x$  axis in unit steps, so you can do everything with integer calculations. At step  $k+1$ , the decision parameter is given as –

$$P_{k+1} = 2dy.x_{k+1} - 2dx.y_{k+1} + C = 2dy.x_{k+1} - 2dx.y_{k+1} + C$$

Subtracting  $P_k$  from this we get –

$$P_{k+1} - P_k = 2dy(x_{k+1} - x_k) - 2dx(y_{k+1} - y_k)$$

But,  $x_{k+1}$  is the same as  $x_k + 1$ . So –

$$P_{k+1} = P_k + 2dy - 2dx(y_{k+1} - y_k)$$

Where,  $y_{k+1} - y_k$  is either 0 or 1 depending on the sign of  $P_k$ .

The first decision parameter  $P_0$  is evaluated at  $(x_0, y_0)$  is given as –

$$P_0 = 2dy - dx$$

Now, keeping in mind all the above points and calculations, here is the Bresenham algorithm for slope  $m < 1$  –

**Step 1** – Input the two end-points of line, storing the left end-point in  $(x_0, y_0)$ .

**Step 2** – Plot the point  $(x_0, y_0)$ .

**Step 3** – Calculate the constants  $dx$ ,  $dy$ ,  $2dy$ , and  $(2dy - 2dx)$  and get the first value for the decision parameter as –

$$p_0 = 2dy - dx \quad p_0 = 2dy - dx$$

**Step 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

$$p_{k+1} = p_k + 2dy \quad p_{k+1} = p_k + 2dy$$

Otherwise,

$$p_{k+1} = p_k + 2dy - 2dx \quad p_{k+1} = p_k + 2dy - 2dx$$

**Step 5** – Repeat step 4  $(dx - 1)$  times.

For  $m > 1$ , find out whether you need to increment  $x$  while incrementing  $y$  each time.

After solving, the equation for decision parameter  $P_k$  will be very similar, just the  $x$  and  $y$  in the equation gets interchanged.

Input
Starting and ending point of line

Output
Given pattern in the problem statement

Lab. Based FAQ
Explain DDA
Explain Bresenham
Difference Between DDA and Bresenham

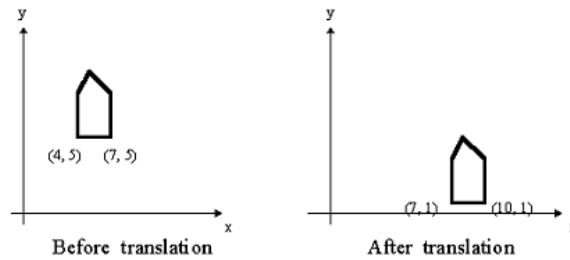
### **Assignment No. : 4B**

Implement translation, sheer, rotation and scaling transformations on equilateral triangle and rhombus.

Aim	
Implement translation, sheer, rotation and scaling transformations on equilateral triangle and rhombus.	

Objective(s)	
1	2 D Homogeneous coordinate system
2	Understand and Implement 2D transformations in Laboratory.

Theory	
<p>1. <b>Translation:</b> Translation is defined as moving the object from one position to another <i>position along straight line path.</i></p>	



We can move the objects based on translation distances along x and y axis.  $t_x$  denotes translation distance along x-axis and  $t_y$  denotes translation distance along y axis.

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

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

$$X' = x + t_x$$

$$Y' = y + t_y$$

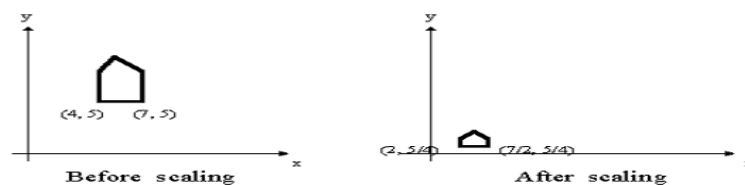
**2. Scaling:** scaling refers to changing the size of the object either by increasing or decreasing. We will increase or decrease the size of the object based on scaling factors along x and y -axis.

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

$$x' = x * s_x$$

$$y' = y * s_y.$$

$s_x$  and  $s_y$  are scaling factors along x-axis and y-axis. we express the above equations in matrix form as:



**3. Rotation :** A rotation repositions all points in an object along a circular path in the plane centered at the

pivot point. We rotate an object by an angle theta

New coordinates after rotation depend on both x and y

$$x' = x \cos \theta - y \sin \theta$$

$$y' = x \sin \theta + y \cos \theta$$

or in matrix form:

$$P' = R \bullet P,$$

R-rotation matrix.

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

#### **4. Shear:**

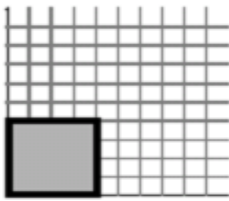
1. Shear is the translation along an axis by an amount that increases linearly with another axis (Y). It produces shape distortions as if objects were composed of layers that are caused to slide over each other.

2. Shear transformations are very useful in creating italic letters and slanted letters from regular letters.

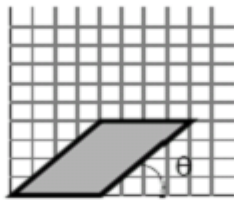
3. Shear transformation changes the shape of the object to a slant position.



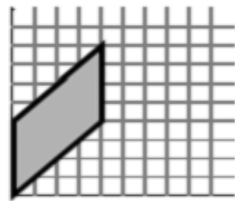
original



x - shear



y - shear



4. Shear transformation is of 2 types:

a. X-shear: changing x-coordinate value and keeping y constant

$$x' = x + sh_x * y$$
$$y' = y$$

b. Y-shear: changing y coordinates value and keeping x constant

$$x' = x$$
$$y' = y + sh_y * x$$

sh<sub>x</sub> and sh<sub>y</sub> are shear factors along x and y-axis

### Input

Enter an object of any shape for transformation

### Output

Output object should undergo above mention transformations and display the transformed Object

### Lab. Based FAQ

1. What is homogeneous co-ordinate system

### Practice Assignment

- |  |
|--|
| 1. Draw a house and perform all the above transformations. |
|--|

**Assignment No. : 2C**

Animation : Implement any one of the following animation assignments,

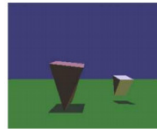
- i) Clock with pendulum
- ii) National Flag hoisting
- iii) Vehicle/boat locomotion
- iv) Falling Water drop into the water and generated waves after impact
- v) Kaleidoscope views generation (at least 3 colorful patterns)

<b>Aim</b>
Animation : Implement any one of the following animation assignments

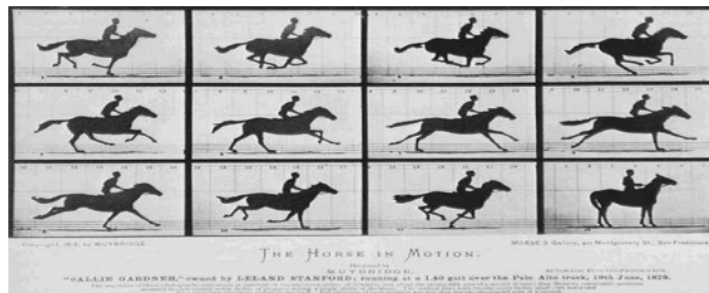
Objective(s)	
1	To learn different types of animation
2	To learn OpenGL function which support for animation

## Theory

Motion can bring the simplest of characters to life. Even simple polygonal shapes can convey a number of human qualities when animated: identity, character, gender, mood, intention, emotion, and so on. Very simple



A movie is a sequence of frames of still images. For video, the frame rate is typically 24 frames per second. For film, this is 30 frames per second.



In general, animation may be achieved by specifying a model with  $n$  parameters that identify degrees of freedom that an animator may be interested in such as

- polygon vertices,
- spline control,
- joint angles,
- muscle contraction,
- camera parameters, or color.

With  $n$  parameters, this results in a vector  $q$  in  $n$ -dimensional state space. Parameters may be varied to generate animation. A model's motion is a trajectory through its state space or a set of motion curves for each parameter over time, i.e.  $\tilde{q}(t)$ , where  $t$  is the time of the current frame.

Every animation technique reduces to specifying the state space trajectory.

The basic animation algorithm is then: for  $t=t_1$  to  $t_{end}$ :  $render(\tilde{q}(t))$ .

Modeling and animation are loosely coupled. Modeling describes control values and their actions.

Animation describes how to vary the control values. There are a number of animation techniques,

including the following:

- User driven animation
  - Keyframing
  - Motion capture
- Procedural animation
  - Physical simulation
  - Particle systems
  - Crowd behaviors
- Data-driven animation

### **Keyframing**

Keyframing is an animation technique where motion curves are interpolated through states at times,  $(\tilde{q}_1, \dots, \tilde{q}_T)$ , called keyframes, specified by a user

### **Kinematics**

Kinematics describe the properties of shape and motion independent of physical forces that cause motion. Kinematic techniques are used often in keyframing, with an animator either setting joint parameters explicitly with forward kinematics or specifying a few key joint orientations and having the rest computed automatically with inverse kinematics.

### **Forward Kinematics**

With forward kinematics, a point  $p$  is positioned by  $p = f(\_)$  where  $\_$  is a state vector  $(\theta_1, \theta_2, \dots, \theta_n)$  specifying the position, orientation, and rotation of all joints.

For the above example,  $p$

$$p = (l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2), l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2)).$$

### **Inverse Kinematics**

With inverse kinematics, a user specifies the position of the end effector,  $p$ , and the algorithm has to evaluate the required  $\theta$  to give  $p$ . That is  $\theta = f^{-1}(p)$ .

Usually, numerical methods are used to solve this problem, as it is often nonlinear and either underdetermined or over determined. A system is underdetermined when there is not a unique solution, such as when there are more equations than unknowns. A system is over determined when it is inconsistent and has no solutions. Extra constraints are necessary to obtain unique and stable solutions. For example, constraints may be placed on the range of joint motion and the solution may be required to minimize the kinetic energy of the system.

### **Motion Capture**

In motion capture, an actor has a number of small, round markers attached to his or her body

that reflect light in frequency ranges that motion capture cameras are specifically designed to pick up



