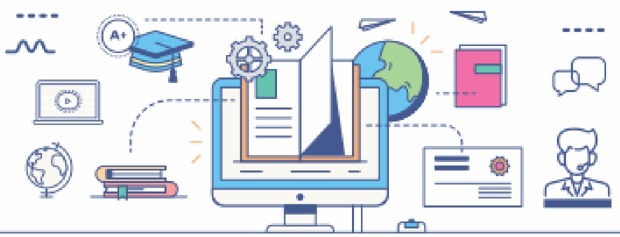
05201330 / 05202182 – Computer Graphics

Dr. Ghanshyam Rathod, Assistant Professor
Parul Institute of Computer Application



Attributes of Output Primitives

- In computer graphics, output primitives are the basic geometric elements used to create images or graphical representations on the screen.
- Attributes related to these primitives, as well as various algorithms to fill areas or handle special effects, are crucial in defining the visual appearance of graphics.
- Output primitives refer to the basic objects used in graphics rendering, such as points, lines, and polygons. Each of these can have associated attributes that define their appearance on the screen.

Attributes of Output Primitives cont.

Line Style: Defines how lines are drawn. It can include various types:

- Solid lines
- Dashed lines
- Dotted lines
- Dash-dot lines

These line styles can be used to convey different types of information or simply for aesthetic purposes.

Color and Intensity:

- **Color** refers to the color used to render primitives, typically represented using RGB (Red, Green, Blue) values or a similar color model.
- **Intensity** refers to the brightness or lightness of a color. For example, a pure red color can have high intensity, while a dimmer version of the same red can have lower intensity.

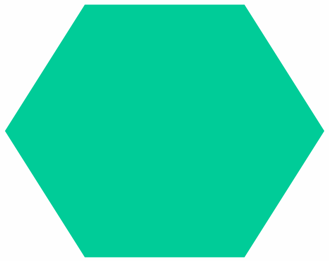
Polygon Filling Algorithm

1. **Scan line Fill Algorithm**
2. **Seed Fill Algorithm**
    - **Boundary Fill Algorithm**
    - **Flood Fill Algorithm**

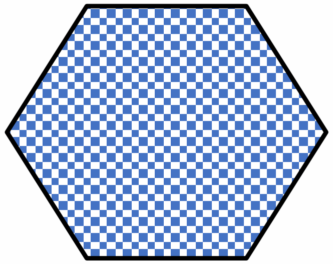
Polygon Filling Algorithm

Types of filling

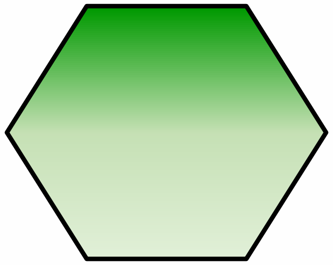
1. **Solid-fill:** All the pixels inside the polygon's boundary are illuminated.



2. **Pattern-fill:** The polygon is filled with an arbitrary predefined pattern.



3. **Gradient Fill:** It is a range of colors that change in value as their position changes. It specifies a range of position-dependent colors, usually used to fill a region.



Polygon Representation (Definition)

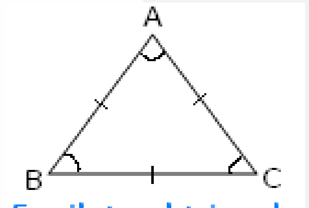
The polygon can be represented by listing its n vertices in an ordered list.

$$P = \{(x_1, y_1), (x_2, y_2), \ldots\ldots, (x_n, y_n)\}.$$

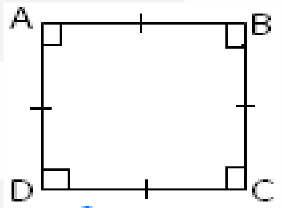
The polygon can be displayed by drawing a line between **(x_1, y_1)**, and **(x_2, y_2), then a line between (x_2, y_2)**, and **(x_3, y_3)**, and so on until the end vertex. In order to close up the polygon, a line between **(x_n, y_n)**, and **(x_1, y_1)** must be drawn.

Types of Polygons: Regular and Irregular

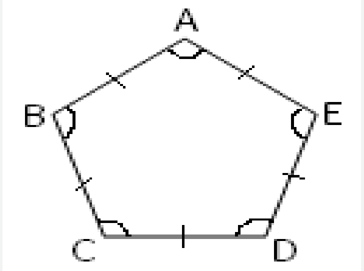
Regular Polygon: A polygon which has all its sides of equal length and all its angles of equal measures is called a **regular polygon**.



Equilateral triangle

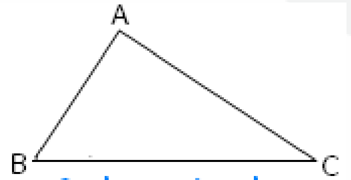


Square



Regular pentagon

Irregular Polygon: A polygon which has all its sides of unequal length and all its angles of unequal measures is called an irregular polygon.

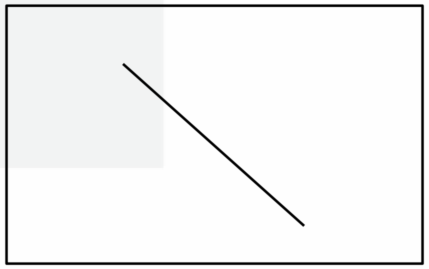


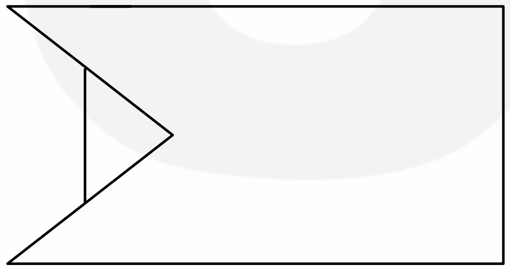
Scalene triangle

Types of Polygons: Regular and Irregular

Convex Polygon - For any two points P_1, P_2 inside the polygon, all points on the line segment which connects P_1 and P_2 are inside the polygon.

Concave Polygon - Aline segment connecting any two points may or may not lie inside the polygon.ie A polygon which is not convex.


Convex Polygon

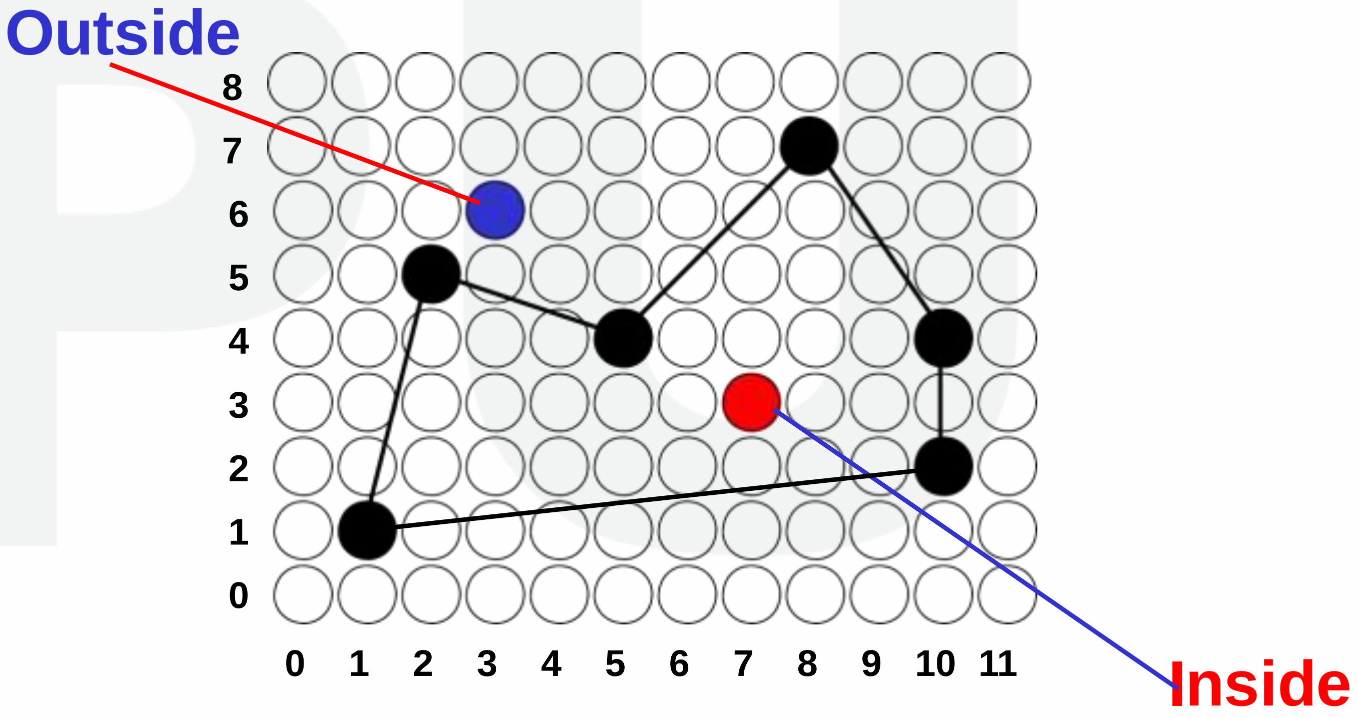

Concave Polygon

Inside-Outside Tests

- When filling polygons we should decide whether a particular point is interior or exterior to a polygon.
- A rule called the **odd-parity** (or the **odd-even rule**) is applied to test whether a point is interior or not.
- To apply this rule, we conceptually draw a line starting from the particular point and extending to a distance point outside the coordinate extends of the object in any direction such that **no polygon vertex intersects** with **the line**.

Inside-Outside Tests

The point is considered to be **interior** if the number of intersections between the line and the polygon edges is **odd**. Otherwise, The point is exterior point.

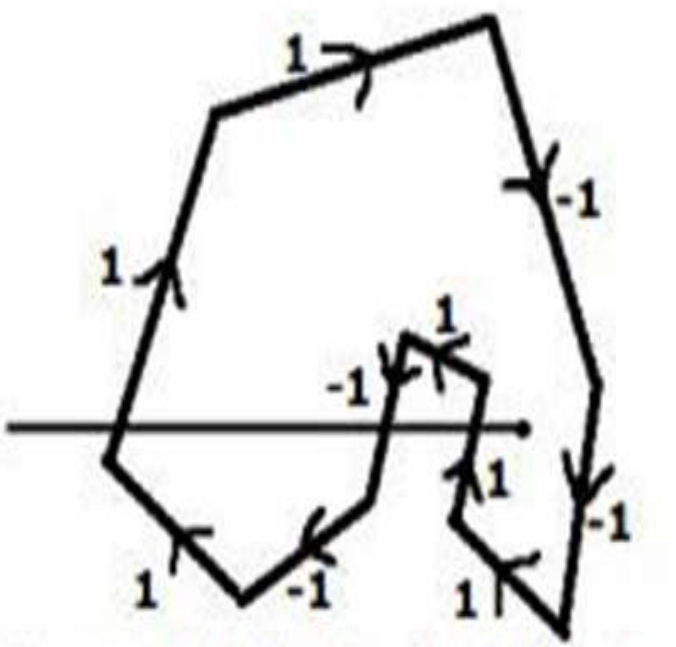


WINDING NUMBER METHOD FOR INSIDE OUT SIDE TEST

If the winding number is **nonzero** then *P* is defined to be an **interior** point **Else** *P* is taken to be an **exterior** point**.**

1. **Sum of edge is nonzero means Inside the polygon**

2. **Sum of edge is zero means outside the polygon**

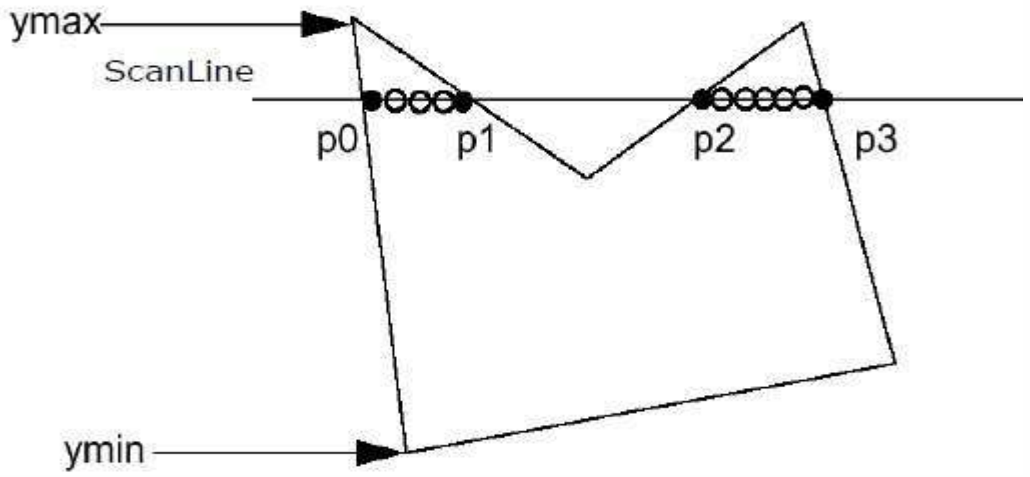


Winding Number Method

The Scan-Line Polygon Fill Algorithm

This algorithm works by intersecting scanline with polygon edges and fills the polygon between pairs of intersections. The following steps depict how this algorithm works.

Step 1 – Find out the Ymin and Ymax from the given polygon.



Scan Line Drawing Algorithm

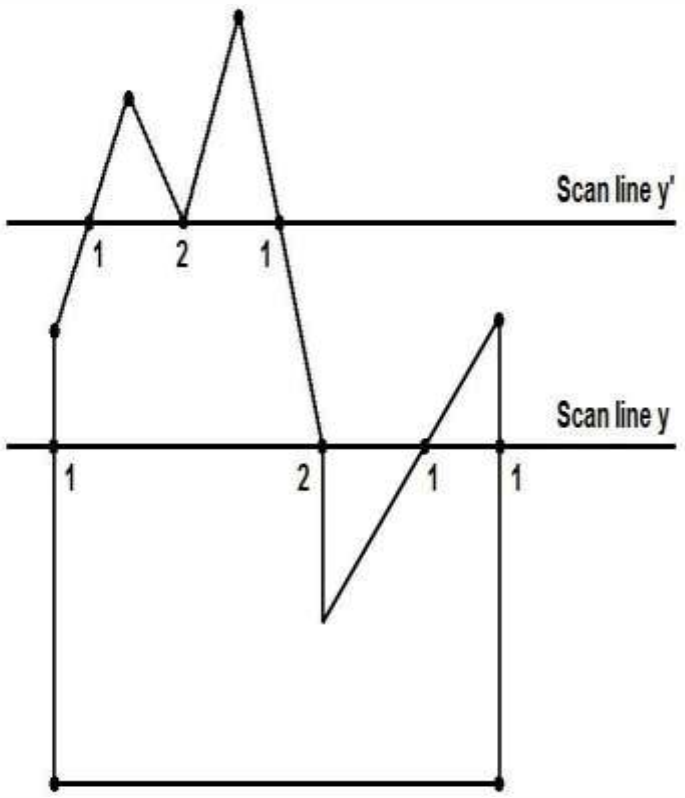
- **Step 2** – Scan Line intersects with each edge of the polygon from Y_{Min} to Y_{Max}. Name each intersection point of the polygon. As per the figure shown above, they are named as p0, p1, p2, p3.

- **Step 3** – Sort the intersection point in the increasing order of X coordinate i.e. (p0, p1), (p1, p2), and (p2, p3).

- **Step 4** – Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.

The Scan-Line Polygon Fill Algorithm

Dealing with Vertices



If the edges having a common intersection point lies on the same side of the scan line, then the intersection point is considered two.

But if the edges having a common intersection point lies on the different side of the scan line, then the intersection point is considered one.

Here please count number of intersection 1.

Problem:
Calculating the Intersection Points is a slow process.

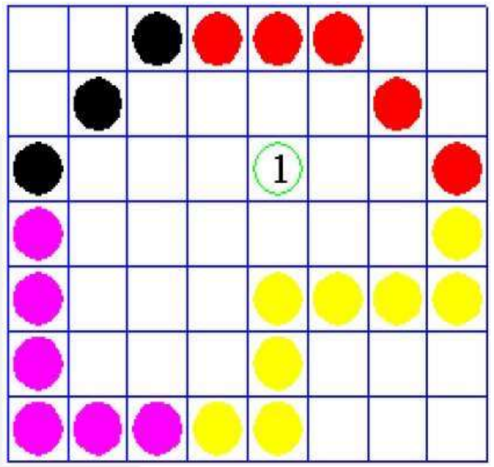
The Scan-Line Polygon Fill Algorithm

Step-04:

Keep repeating Step-03 until the end point is reached or number of iterations equals to (ΔX-1) times.

Flood Fill Algorithm

- Sometimes we come across an object where we want to fill the area and its boundary with different colors.
- In other words, it replaces the interior color of the object with the fill color. When no more pixels of the original interior color exist, the algorithm is completed.
- This algorithm relies on the Four-connect or Eight-connect method of filling in the pixels.
- But instead of looking for the boundary color, it is looking for all adjacent pixels that are a part of the interior.



Parul® University

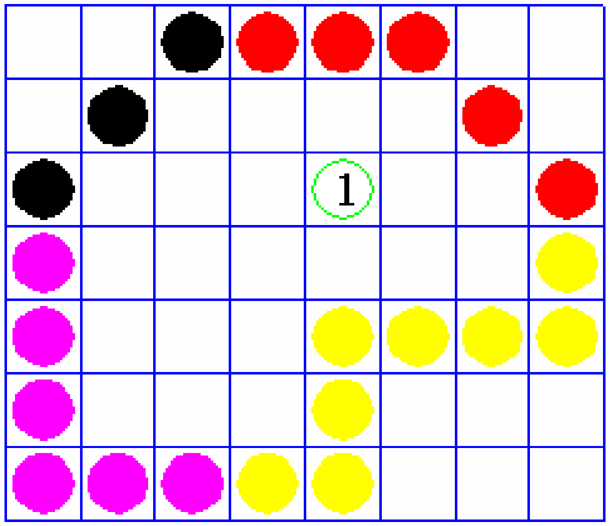
DIGITAL LEARNING CONTENT

Flood Fill Algorithm

Sometimes we want to fill in (recolour) an area that is not defined within a single colour boundary.

We paint such areas by replacing a specified interior colour instead of searching for a boundary colour value.

This approach is called a **flood-fill algorithm**.



Flood Fill Algorithm

We start from a specified interior pixel (x, y) and reassign all pixel values that are currently set to a given interior color with the desired fill color.

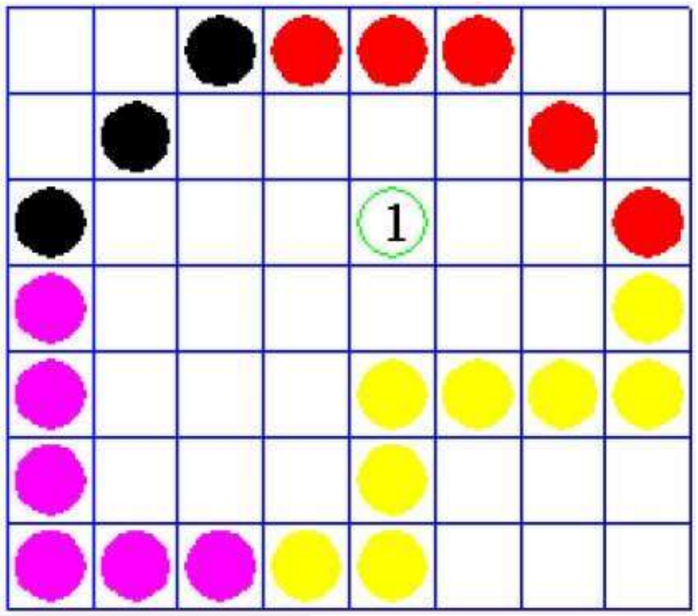
If the area has **more than one** interior color, we can first **reassign pixel values** so that all interior pixels have the same color.

Using either **4-connected** or **8-connected** approach, we then step through pixel positions until all interior pixels have been repainted.

In this algorithm we match color of a pixel with background color of the object. We stop color fill until we does not find a pixel where color is not same as background color.

Flood Fill Algorithm

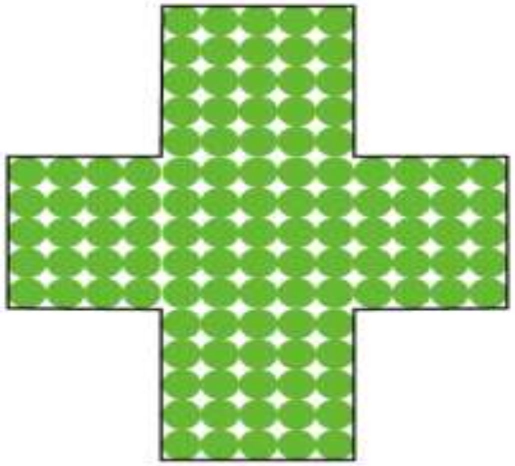
```
Procedure floodfill (x, y,fill_ color, old_color: integer)
{
   If (getpixel (x, y)=old_color)
  {
   setpixel (x, y, fill_color);
   fill (x+1, y, fill_color, old_color);
    fill (x-1, y, fill_color, old_color);
   fill (x, y+1, fill_color, old_color);
   fill (x, y-1, fill_color, old_color);
    }
}
```



Flood Fill Algorithm

Advantages of Flood-fill algorithm

- It provides an easy way to fill color in graphics.
- The Flood-fill algorithm colors the whole area through interconnected pixels by a single color.
- The algorithm fills the same color inside the boundary.



Disadvantages of Flood-fill Algorithm

- It is a more time-consuming algorithm.
- Sometimes it does not work on large polygons.

Boundary Fill Algorithm

- Another approach to area filling is to start at a point inside a region and paint the interior outward toward the boundary.
- If the boundary is specified in a single color, the fill algorithm processed outward pixel by pixel until the boundary color is encountered.
- A boundary-fill procedure accepts as input the coordinate of the interior **point (x, y)**, a **fill color**, and a **boundary color**.

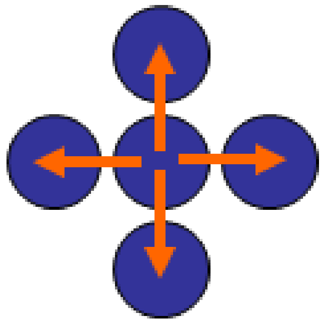
Boundary Fill Algorithm

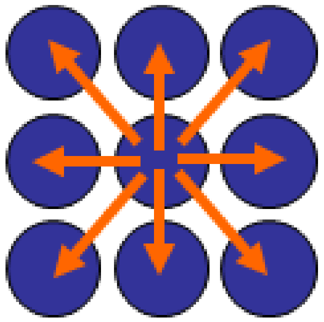
The following steps illustrate the **recursive** boundary-fill algorithm:

1. Start from an interior point.
2. If the current pixel is **not already** filled and if it is not an edge point, then set the pixel with the fill color, and store its neighbouring pixels (**4** or **8 - connected**) in the stack for processing. Store only neighbouring pixel that is **not already** filled and is not an edge point.
3. Select the next pixel from the stack, and continue with step **2**.

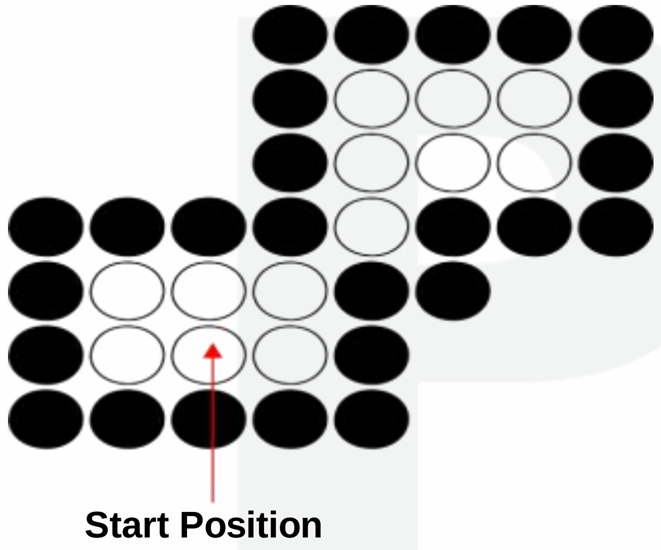
Boundary Fill Algorithm

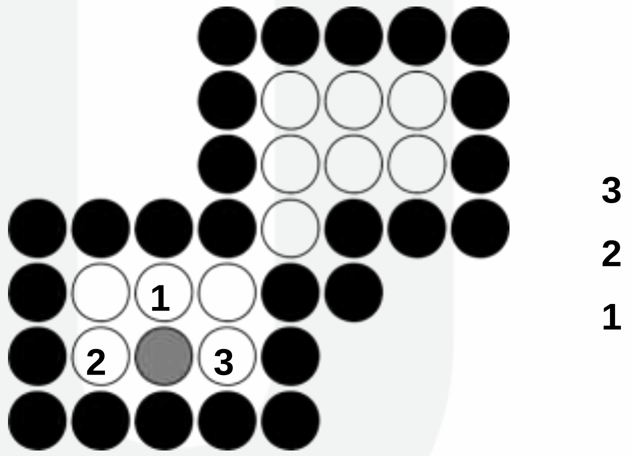
The order of pixels that should be added to stack using 4-connected is above, below, left, and right. For 8-connected is above, below, left, right, above- left, above-right, below-left, and below-right.



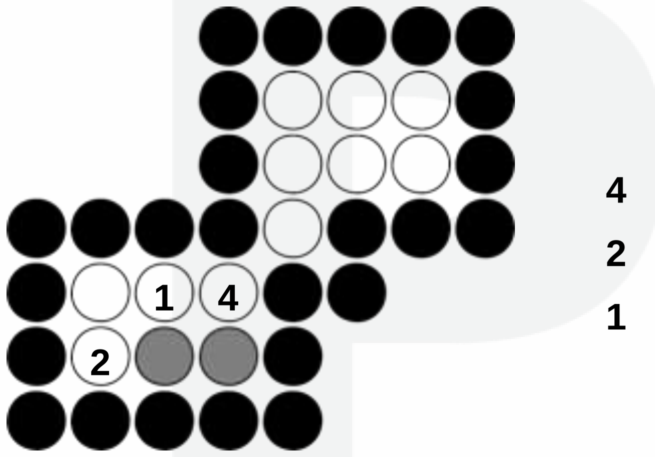


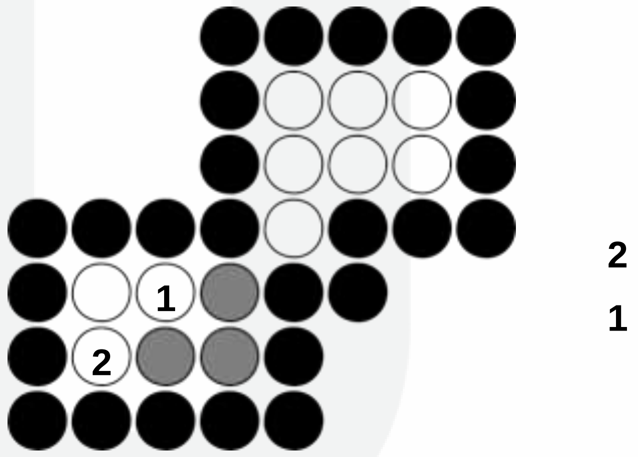
Boundary Fill Algorithm : 4-connected (Example)





Boundary Fill Algorithm : 4-connected (Example)

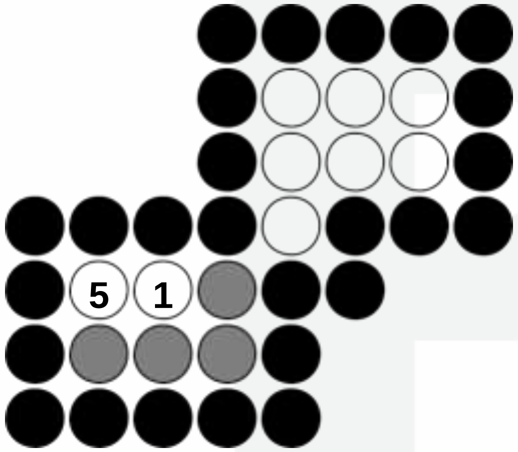




Parul® University

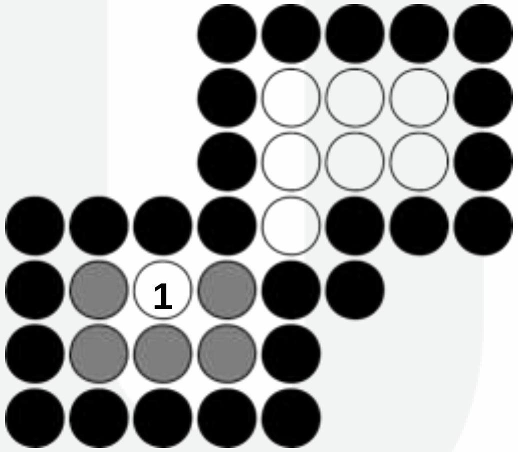
DIGITAL LEARNING CONTENT

Boundary Fill Algorithm : 4-connected (Example)



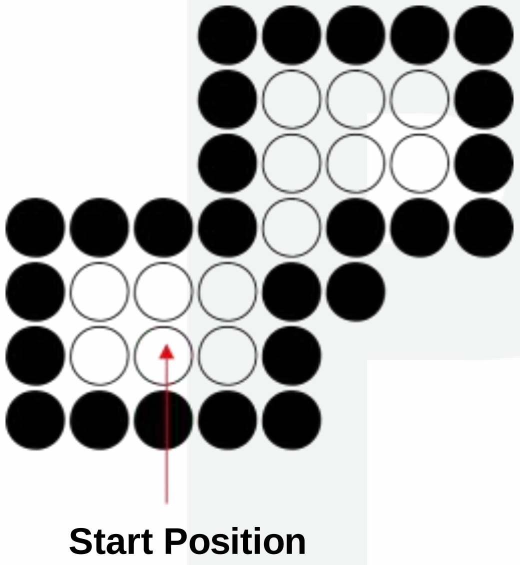
5

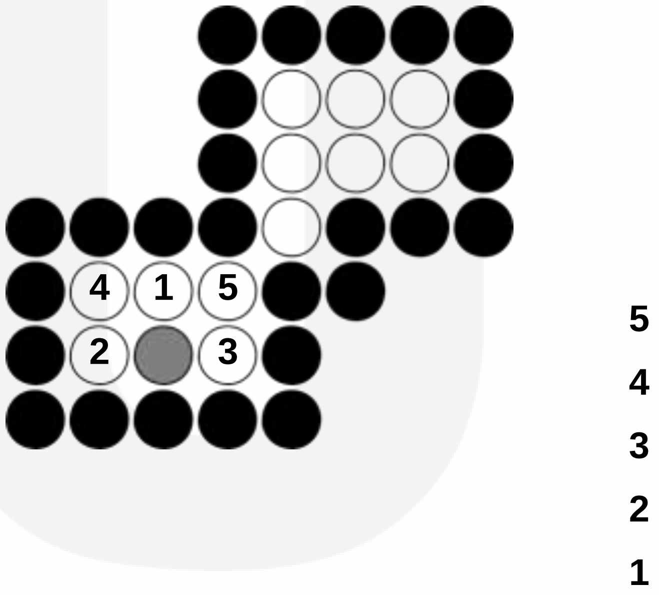
1



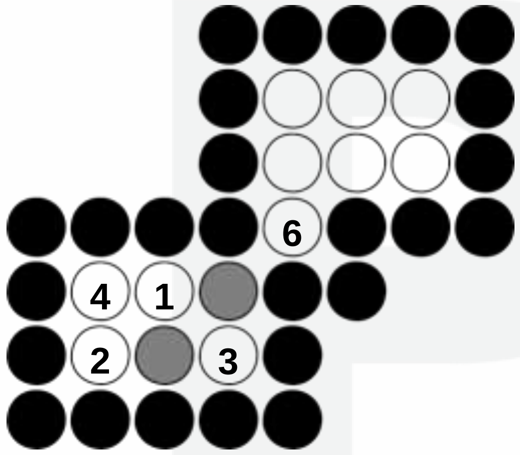
1

Boundary Fill Algorithm(8-connected Example)

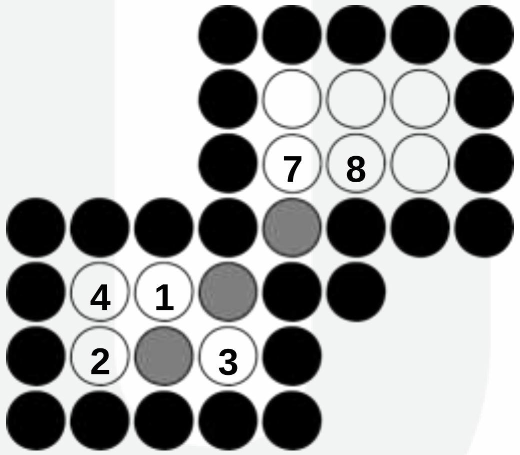




Boundary Fill Algorithm(8-connected Example)

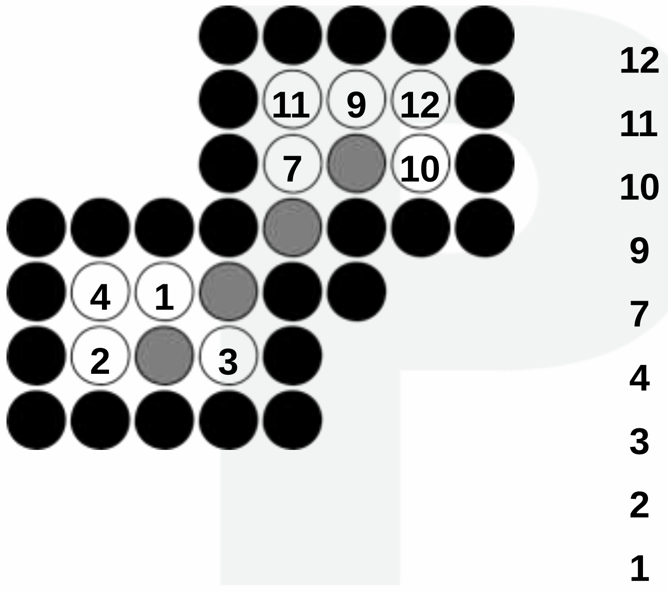


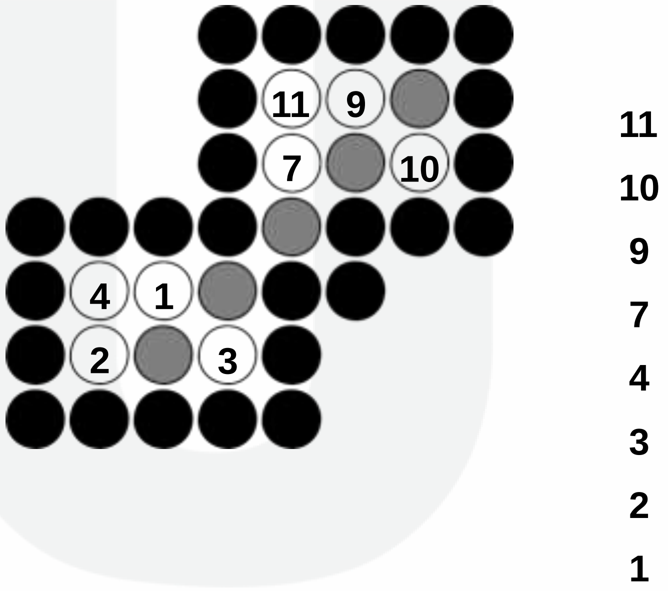
6
4
3
2
1



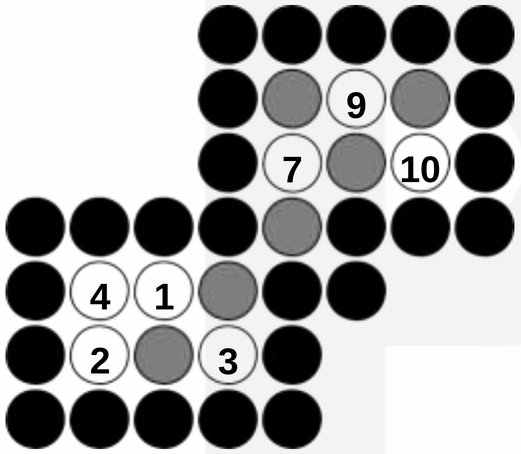
8
7
4
3
2
1

Boundary Fill Algorithm(8-connected Example)





Boundary Fill Algorithm(8-connected Example)



10

9

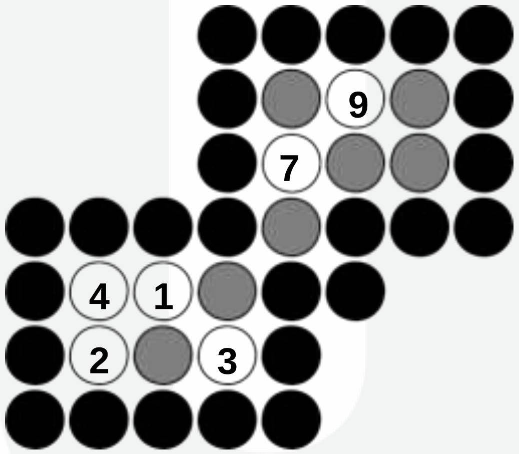
7

4

3

2

1



9

7

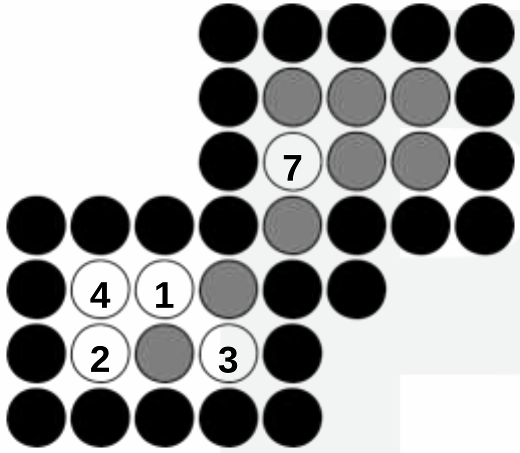
4

3

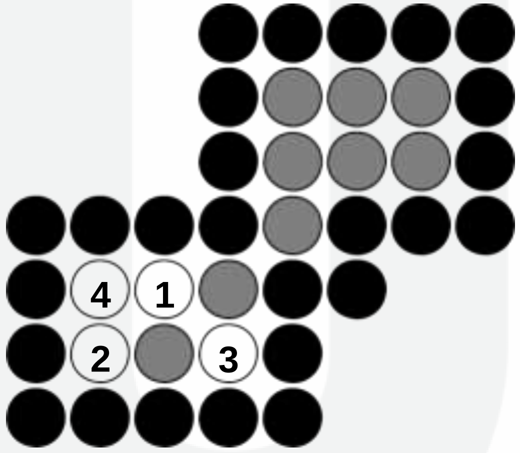
2

1

Boundary Fill Algorithm(8-connected Example)

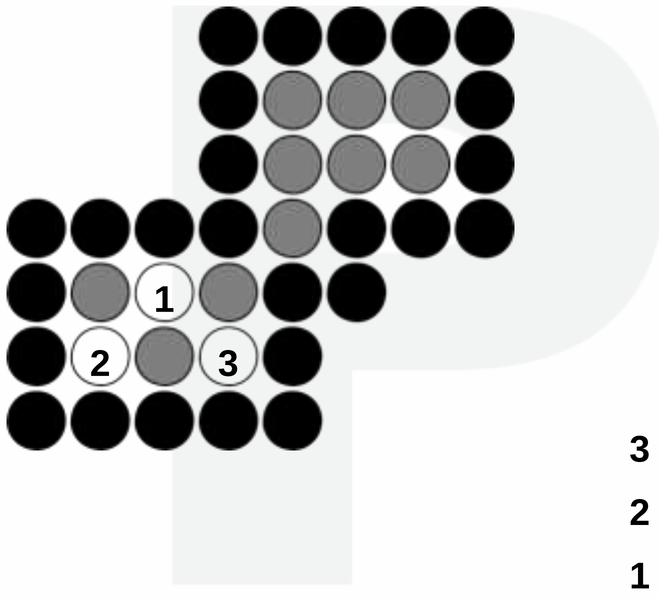


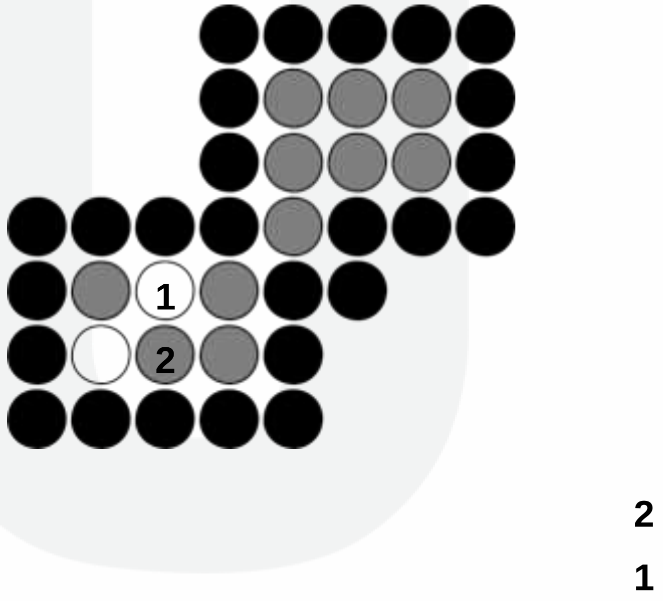
7
4
3
2
1



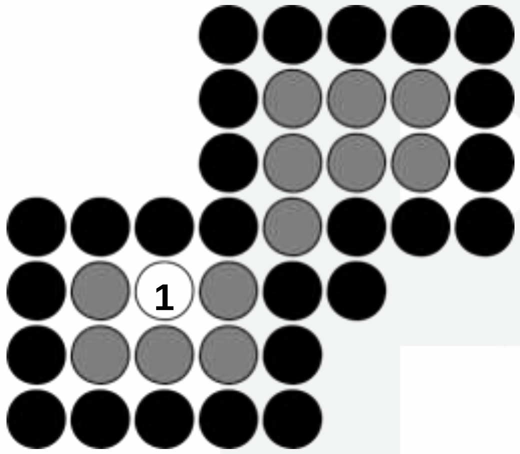
4
3
2
1

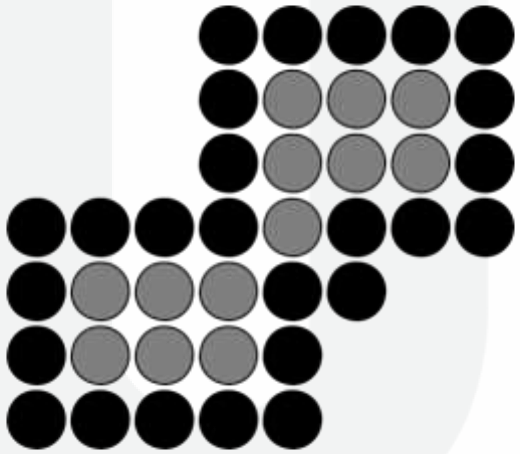
Boundary Fill Algorithm(8-connected Example)





Boundary Fill Algorithm(8-connected Example)





1

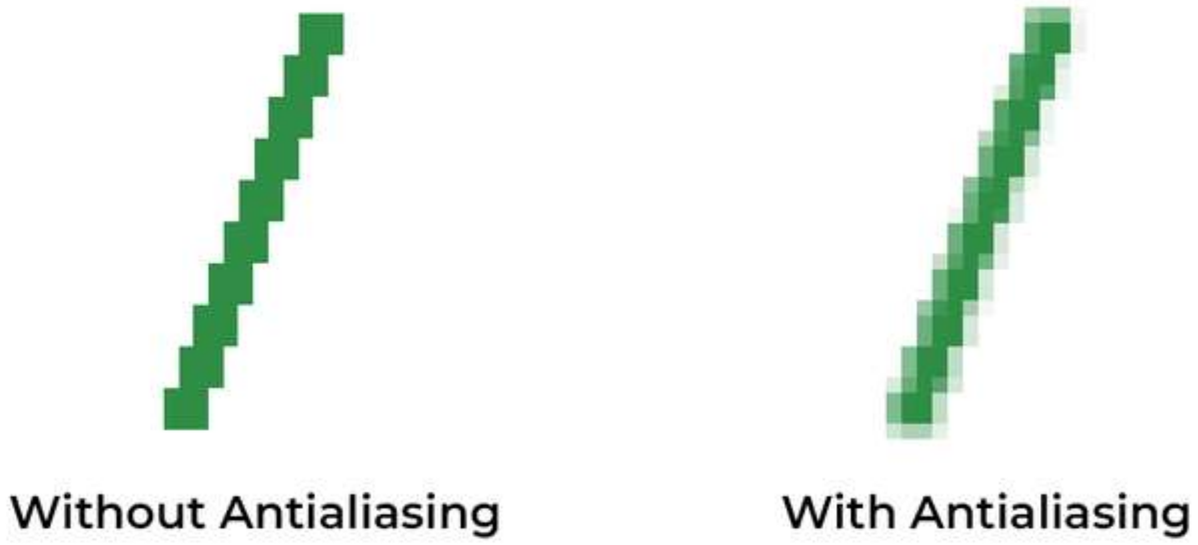
Anti-Aliasing

- The output primitive generated with the raster algorithms contains the jagged or stairstep appearance, because of the conversion of the pixel value into integers. This type of distortion (change in shape) is called aliasing. There are methods
- To remove such problem, which are known as antialiasing
- A simple method for antialiasing is that increase the sampling rate (no. of pixels) assuming that the screen consists of more pixels.
- This technique of creating output primitive at high resolution, but displaying it at low resolution, is called **supersampling** or **postfiltering**.

Anti-Aliasing

Methods of Anti-Aliasing

- High-Resolution Display
- Post-Filtering
- Pre-Filtering
- Pixel Phasing



Anti-Aliasing

1. **Using High-Resolution Display:** One way to reduce the aliasing effect and increase the sampling rate is to simply display objects at a higher resolution. Using high resolution, the jaggies become so small that they become indistinguishable from the human eye. Hence, jagged edges get blurred out and edges appear smooth.

   For **example**, retina displays in Apple devices, and OLED displays have high pixel density due to which jaggies formed are so small that they blurred and are in-distinguishable by our eyes.

Anti-Aliasing

2. **Post Filtering (Supersampling):** A simple method for antialiasing is that increase the sampling rate (no. of pixels) assuming that the screen consists of more pixels. This technique of creating output primitive at high resolution, but displaying it at low resolution to the desired screen size.

   For **example**, In gaming, **SSAA** (Supersample Antialiasing) or **FSAA** (Full-Scene Antialiasing) is used to create the best image quality. It is often called pure AA and hence is very slow and has a very high computational cost. This technique was widely used in the early days. A better style of Anti-Aliasing is **MSAA** (Multisampling Antialiasing) which is a faster and more approximate style of supersampling AA.

Anti-Aliasing

3. **Pre-Filtering (Area Sampling):** In area sampling, pixel intensities are calculated proportionally to areas of overlap of each pixel with objects to be displayed. Here pixel color is computed based on the overlap of the scene's objects with a pixel area.

   Example:

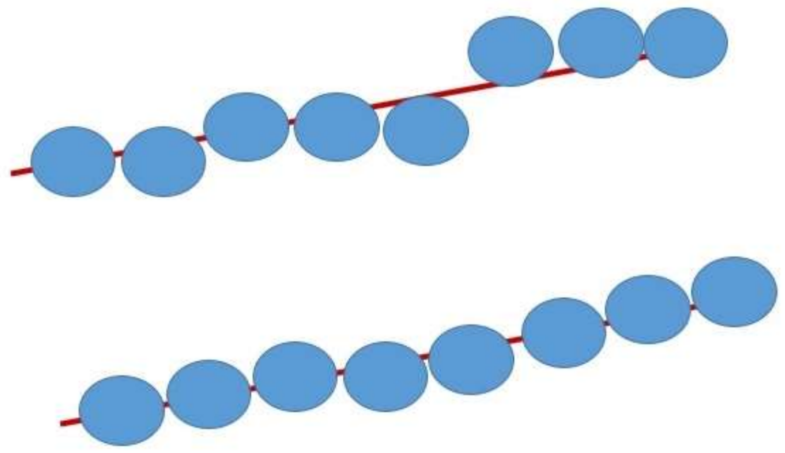
   Before Area Sampling: Imagine you have an image with many pixels. When you want to reduce its size, each pixel might represent a small section of the image.

   After Area Sampling: When the image is resized, area sampling takes into account the color and details of neighboring pixels to decide how to combine or average them into a smaller set of pixels. This helps maintain the key details and reduce the blocky appearance that might otherwise result from shrinking the image.

Anti-Aliasing

3. **Pixel Phasing:** Pixel phasing is a method of anti-aliasing that involves shifting the position of the pixels slightly to reduce the appearance of jagged edges.

   This method is often used in combination with other methods of anti-aliasing to produce the best results.



Transformation

- Transformation means changing some graphics into something else by applying rules.
- It is used to reposition the graphics on the screen and change their size or orientation.
- The basic types of transformations are …

  1) Translation,

  2) Rotation,

  3) Scaling,

  4) Shearing,

  5) Reflection

  When a transformation takes place on a 2D plane, it is called 2D transformation.

Homogenous Coordinate System

A 3×3 transformation matrix can be used instead of 2×2 transformation matrix. To convert a 2×2 matrix to 3×3 matrix, we have to add an extra dummy coordinate W.

Any point can be represented by using 3 numbers instead of 2 numbers, which is called **Homogenous Coordinate** system.

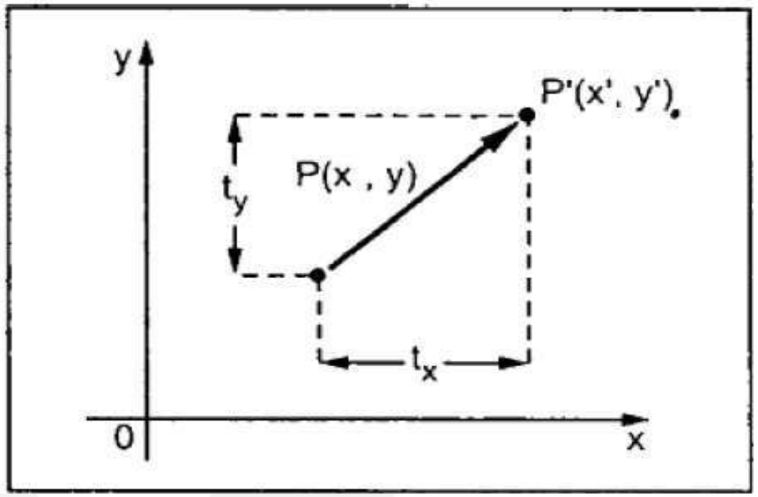
All the transformation equations can be represented in matrix multiplication. Any Cartesian point P(X, Y) can be converted to homogenous coordinates by P' (X_h, Y_h, h).

Translation

Translation refers to the shifting of a point to some other place, **whose distance with regard to the present point is known.**

A translation moves an object to a different position on the screen.

You can translate a point in 2D by adding translation coordinate (t_x, t_y) to the original coordinate (X, Y) to get the new coordinate (X', Y').



Translation

From the above figure, we can write

$$X' = X + t_x$$

$$Y' = Y + t_y$$

Where (t_x, t_y) is called the **translation vector or shift vector**.

The above equations can also be represented using the column vectors.

P = [X][Y]

P’= [X’][Y’]

T = [tx][ty]

Which can be written as

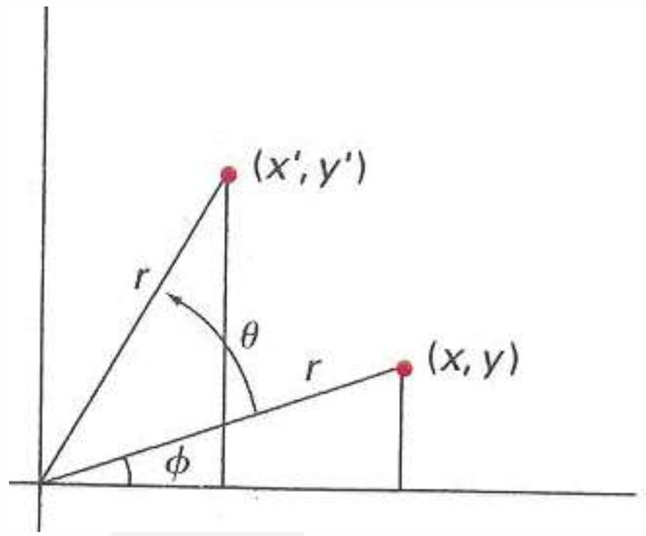
$$P' = P + T$$

Rotation

Rotation means to rotate a point about an axis.

In rotation, an object is rotated at a particular angle θ (theta) from its origin.

From the following figure, we can see that the point PX, Y is located at angle ϕ from the horizontal X coordinate with distance r from the origin



Rotation

Let us suppose you want to rotate it at the angle θ. After rotating it to a new location, you will get a new point P' (X' , Y').

Using standard trigonometric the original coordinate of point P(X, Y) can be represented as –

$$x = r\cos\varphi ......(1)$$

$$y = r\sin\varphi ......(2)$$

Same way we can represent the point P' (X', Y') as –

$$x' = r\cos(\varphi+\theta) = r\cos\varphi\cos\theta - r\sin\varphi\sin\theta ....(3)$$

$$y' = r\sin(\varphi+\theta) = r\cos\varphi\sin\theta + r\sin\varphi\cos\theta ....(4)$$

Rotation

Substituting equation (1) & (2) in (3) & (4) respectively, we will get

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

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

Representing the above equation in matrix form,

$$[X'Y'] = [XY]\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

P' = P . R

Where R is the rotation matrix

Rotation

The rotation angle can be positive or negative.

For positive rotation angle, we can use the below rotation matrix.

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

However, for negative angle rotation, the matrix will change as shown below –

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

Rotation

Example

Given a triangle with corner coordinates (0, 0), (1, 0) and (1, 1). Rotate the triangle by 90 degree anticlockwise direction and find out the new coordinates.

We rotate a polygon by rotating each vertex of it with the same rotation angle.

Given-

- Old corner coordinates of the triangle = A (0, 0), B(1, 0), C(1, 1)
- Rotation angle = $\theta = 90^\circ$

Rotation

For Coordinates A (0, 0)

Let the new coordinates of corner A after rotation = (x',y').

Applying the rotation equations, we have-

X'= x*cosθ – Y* sinθ

= 0 * cos90º – 0 * sin90º

= 0

Y'= x* sinθ +y* cosθ

= 0*sin90º + 0 * cos90º

= 0

Thus, New coordinates of corner A after rotation = (0, 0).

Rotation

For Coordinates B(1, 0)

Let the new coordinates of corner B after rotation = (x',y').

X'= x*cosθ – y* sinθ = 1 x cos90º – 0 x sin90º = 0

Y'= x*sinθ +y*cosθ = 1 x sin90º + 0 x cos90º = 1 + 0 = 1

Thus, New coordinates of corner B after rotation = (0, 1).

For Coordinates C(1, 1)

Let the new coordinates of corner C after rotation = (Xnew, Ynew).

X'= x* cosθ –y*sinθ = 1 x cos90º – 1 x sin90º= 0 – 1= -1

Y'= x*sinθ + y*cosθ= 1 x sin90º + 1 x cos90º= 1 + 0= 1

Thus, New coordinates of corner C after rotation = (-1, 1).

Thus, New coordinates of Triangle after Rotation = A(0,0), B(0,1), C(-1,1)

Scaling

- To change the size of an object, scaling transformation is used.
- **Scaling** is the concept of increasing (or decreasing) the size of a picture in any direction. (When it is done in both directions, the increase or decrease in both directions need not be same)
- Scaling can be achieved by **multiplying the original coordinates of the object** with the **scaling factor** to get the desired result.

$X' = X \cdot S_X$ and $Y' = Y \cdot S_Y$ Where, original coordinates are (X, Y)

Scaling factors are (S_X, S_Y), and the

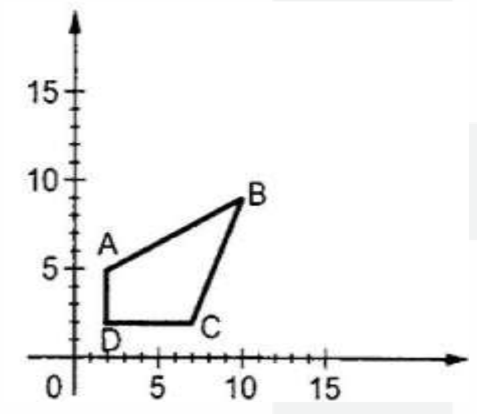
Produced coordinates are (X’, Y’).

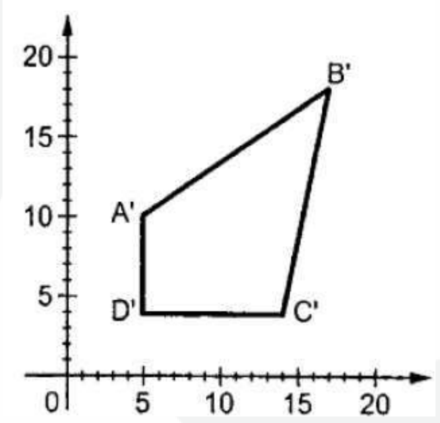
Scaling

The above equations is represented in matrix form

OR $P' = P \cdot S$

Where S is the scaling matrix. The scaling process is shown in the following figure.





If S value is less than 1, then size of object reduces

If S greater than 1, then size of object increase

Scaling: Example

Given a square object with coordinate points A(0, 3), B(3, 3), C(3, 0), D(0, 0). Apply the scaling parameter 2 towards X axis and 3 towards Y axis and obtain the new coordinates of the object.

Old corner coordinates of the square = A (0, 3), B(3, 3), C(3, 0), D(0, 0)

- Scaling factor along X axis = Sx=2
- Scaling factor along Y axis = Sy=3

For Coordinates A(0, 3):x=0 and y=3

Let the new coordinates of corner A after scaling = (X_{new}, Y_{new}).

Applying the scaling equations, we have-

- $X' = X*S_x = 0 \times 2 = 0$
- $Y' = Y* S_y = 3 \times 3 = 9$

Thus, New coordinates of corner A after scaling = A'(0, 9).

Scaling: Example

For Coordinates B(3, 3) : B’(6,9)

Let the new coordinates of corner B after scaling = (X_{new}, Y_{new}).

Applying the scaling equations, we have-

- $X_{new} = X_{old} \times S_x = 3 \times 2 = 6$
- $Y_{new} = Y_{old} \times S_y = 3 \times 3 = 9$

Thus, New coordinates of corner B after scaling = (6, 9).

For Coordinates C(3, 0): C’(6,0)

- $X_{new} = X_{old} \times S_x = 3 \times 2 = 6$
- $Y_{new} = Y_{old} \times S_y = 0 \times 3 = 0$

Thus, New coordinates of corner C after scaling = (6, 0).

Scaling: Example

For Coordinates D(0, 0)

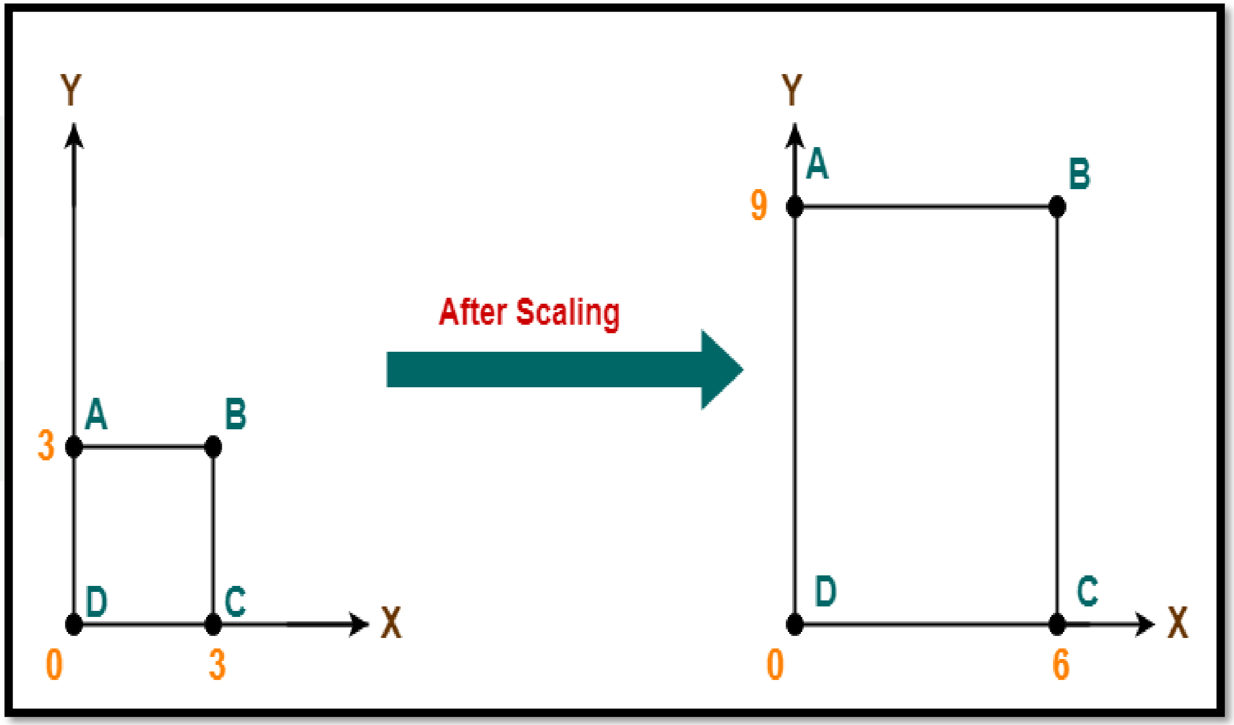
:D’(0,0)

Let the new coordinates of corner D after scaling = (X_{new}, Y_{new}).

Applying the scaling equations, we have-

- $X_{new} = X_{old} \times S_x = 0 \times 2 = 0$
- $Y_{new} = Y_{old} \times S_y = 0 \times 3 = 0$

Thus, New coordinates of corner D after scaling = (0, 0).

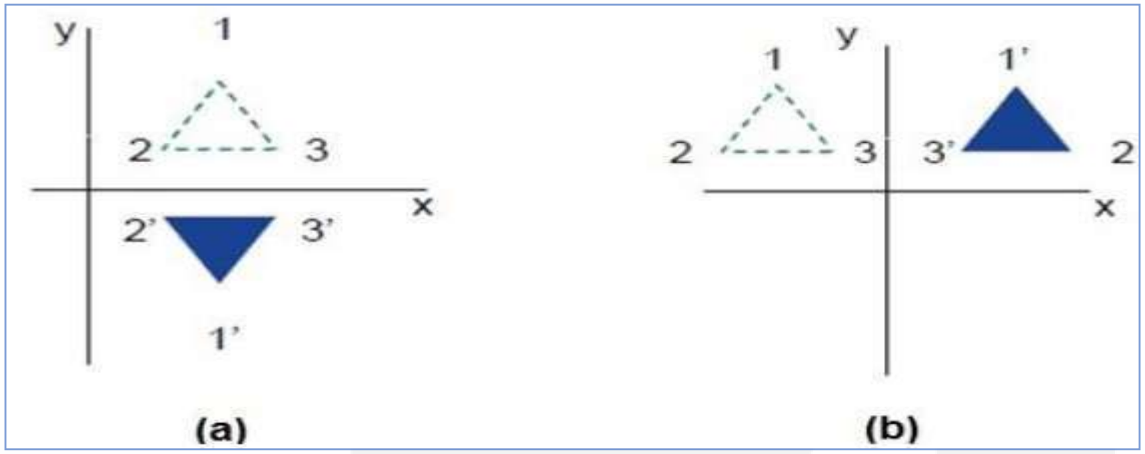


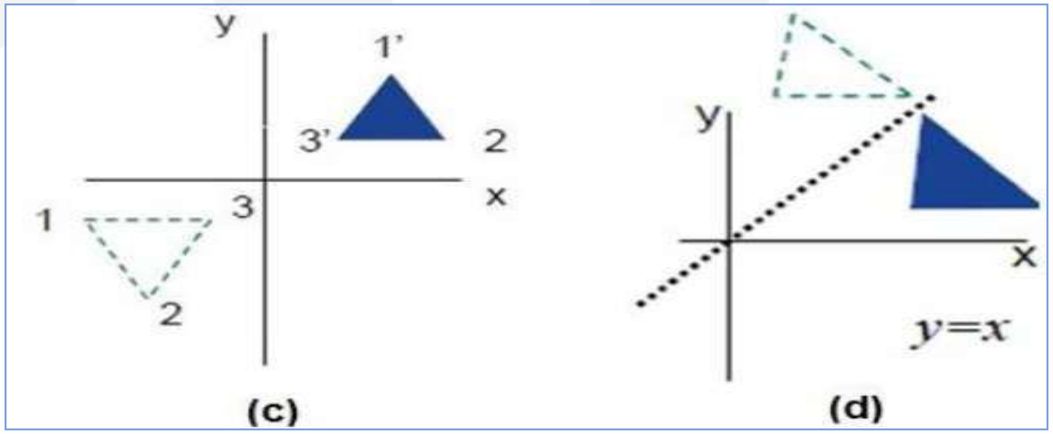
Reflection

- Reflection is the mirror image of original object. In other words, we can say that it is a rotation operation with 180°. In reflection transformation, the size of the object does not change.

- The following figures show reflections with respect to X and Y axes, and about the origin respectively.

Reflection





Reflection

Reflection On X-Axis:

This reflection is achieved by using the following reflection equations- (X’ and Y’ are the new coordinates)

X’ = X

Y’ = -Y

In Matrix form, the above reflection equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Reflection Matrix

(Reflection Along X Axis)

Reflection

Reflection On Y-Axis:

This reflection is achieved by using the following reflection equations- (X’ and Y’ are the new coordinates)

X’ = -X

Y’ = Y

In Matrix form, the above reflection equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Reflection Matrix

(Reflection Along Y Axis)

Reflection: Example

Given a triangle with coordinate points A(3, 4), B(6, 4), C(5, 6). Apply the reflection on the X axis and obtain the new coordinates of the object.

Given-

- Old corner coordinates of the triangle = A (3, 4), B(6, 4), C(5, 6)
- Reflection has to be taken on the X axis

Reflection: Example

<u>For Coordinates A(3, 4)</u>

Let the new coordinates of corner A after reflection = (X_{new}, Y_{new}).

Applying the reflection equations, we have-

- $X_{new} = X_{old} = 3$
- $Y_{new} = -Y_{old} = -4$

Thus, New coordinates of corner A after reflection = (3, -4).

Reflection: Example

<u>For Coordinates B(6, 4)</u>

Let the new coordinates of corner B after reflection = (X_{new}, Y_{new}).

Applying the reflection equations, we have-

- $X_{new} = X_{old} = 6$
- $Y_{new} = -Y_{old} = -4$

Thus, New coordinates of corner B after reflection = (6, -4).

Reflection: Example

For Coordinates C(5, 6)

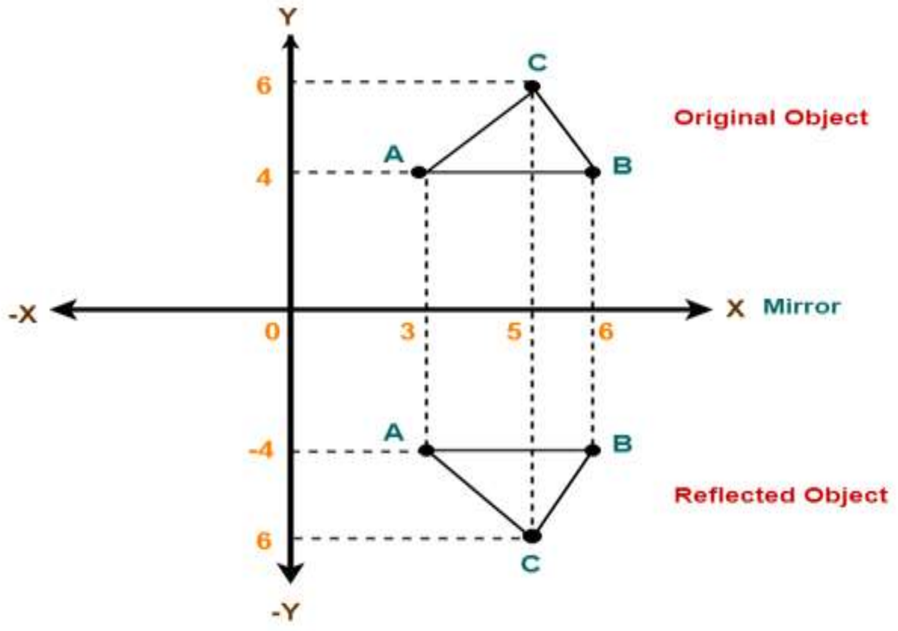
Let the new coordinates of corner C after reflection = (X_{new}, Y_{new}).

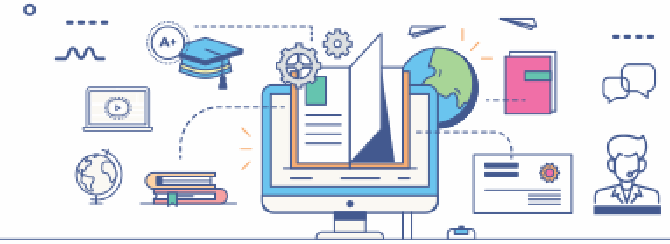
Applying the reflection equations, we have-

- $X_{new} = X_{old} = 5$
- $Y_{new} = -Y_{old} = -6$

Thus, New coordinates of corner C after reflection = (5, -6).

Thus, New coordinates of the triangle after reflection = A (3, -4), B(6, -4), C(5, -6).





Shear

A transformation that slants the shape of an object is called the shear transformation.

Types of shear transformations :

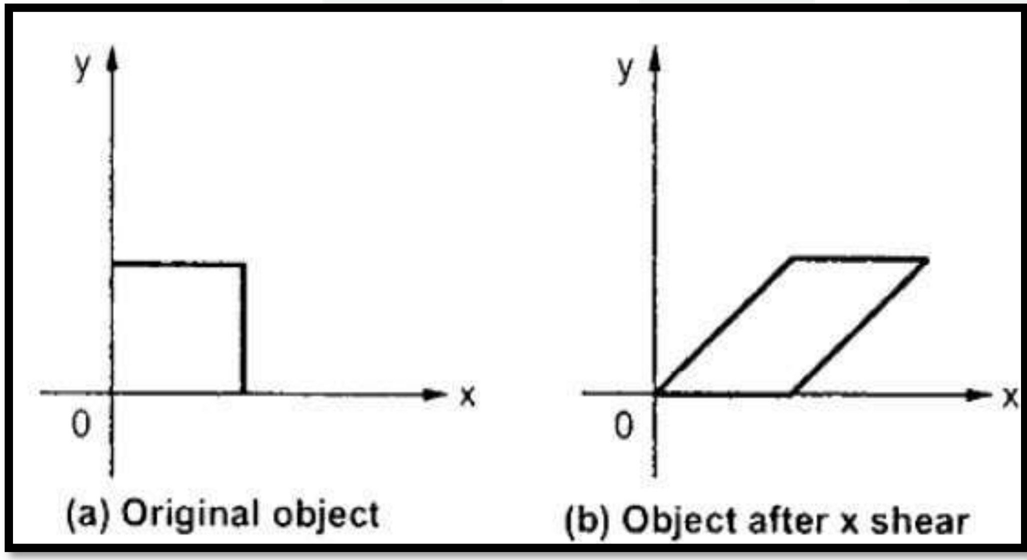
X-Shear and **Y-Shear**.

One shifts X coordinates values and other shifts Y coordinate values. However; in both the cases only one coordinate changes its coordinates and other preserves its values. Shearing is also termed as **Skewing**.

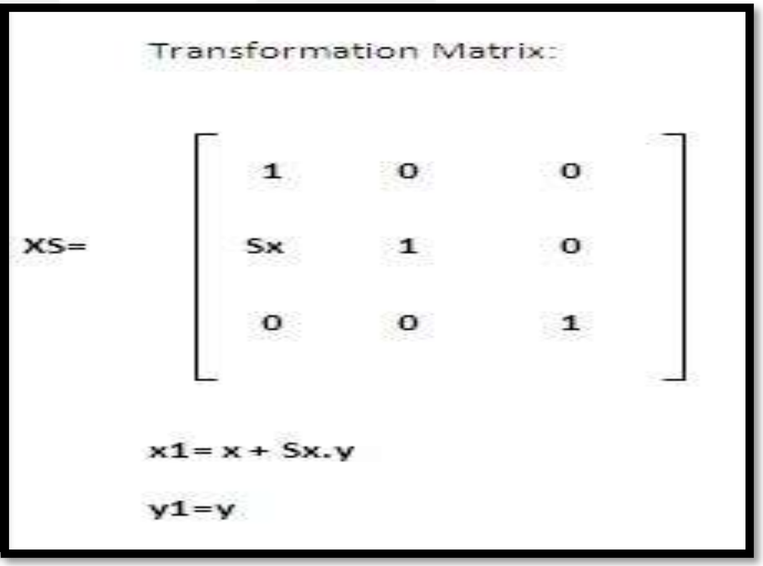
X-Shear

The X-Shear preserves the Y coordinate and changes are made to X coordinates, which causes the vertical lines to tilt right or left as shown in below figure.

The transformation matrix for X-Shear can be represented as –



(a) Original object
(b) Object after x shear


Transformation Matrix:

$$XS = \begin{bmatrix} 1 & 0 & 0 \\ Sx & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$x1 = x + Sx.y$

$y1 = y$

X-Shear

Shearing in X Axis-

$X_{new} = X_{old} + Sh_x \times Y_{old}$

$Y_{new} = Y_{old}$

The transformation matrix for X-Shear can be represented as –

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & Sh_x \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Shearing Matrix

(In X axis)

Y-Shear

The Y-Shear preserves the X coordinates and changes the Y coordinates which causes the horizontal lines to transform into lines which slopes up or down as shown in the following figure.

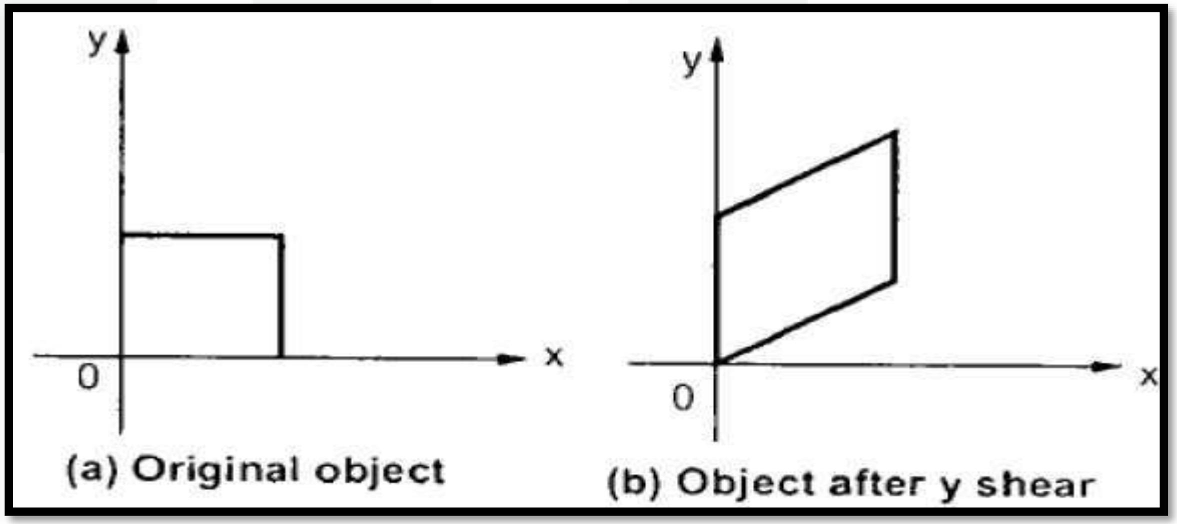
The Y-Shear can be represented in matrix from as:

Transformation Matrix:

$$XY = \begin{bmatrix} 1 & Sy & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$x1 = x$

$y1 = y + Sx$



(a) Original object

(b) Object after y shear

Parul® University

DIGITAL LEARNING CONTENT

Y-Shear

Shearing in Y axis is achieved by using the following shearing equations-

- $X_{new} = X_{old}$
- $Y_{new} = Y_{old} + Sh_y \times X_{old}$

The Y-Shear can be represented in matrix from as –

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ Sh_y & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Shearing Matrix

(In Y axis)

Shear - Example

Given a triangle with points (1, 1), (0, 0) and (1, 0). Apply shear parameter 2 on X axis and 2 on Y axis and find out the new coordinates of the object.

Given-

- Old corner coordinates of the triangle = A (1, 1), B(0, 0), C(1, 0)
- Shearing parameter towards X direction (Sh_x) = 2
- Shearing parameter towards Y direction (Sh_y) = 2

Shear - Example

Shearing in X Axis- For Coordinates A(1, 1)

Let the new coordinates of corner A after shearing = (X_{new}, Y_{new}).

Applying the shearing equations, we have-

- $X_{new} = X_{old} + Sh_x \times Y_{old} = 1 + 2 \times 1 = 3$
- $Y_{new} = Y_{old} = 1$

Thus, New coordinates of corner A after shearing = (3, 1).

Shear - Example

For Coordinates B(0, 0)

Let the new coordinates of corner B after shearing = (X_{new}, Y_{new}).

Applying the shearing equations, we have-

- $X_{new} = X_{old} + Sh_x \times Y_{old} = 0 + 2 \times 0 = 0$
- $Y_{new} = Y_{old} = 0$

Thus, New coordinates of corner B after shearing = (0, 0).

For Coordinates C(1, 0)

Thus, New coordinates of corner C after shearing = (1, 0).

New coordinates of the triangle after shearing in X axis = A (3, 1), B(0, 0), C(1, 0).

Shear - Example

<u>Shearing in Y Axis-</u>

<u>For Coordinates A(1, 1)</u>

Let the new coordinates of corner A after shearing = (X_{new}, Y_{new}).

Applying the shearing equations, we have-

- $X_{new} = X_{old} = 1$
- $Y_{new} = Y_{old} + Sh_y \times X_{old} = 1 + 2 \times 1 = 3$

Thus, New coordinates of corner A after shearing = (1, 3).

<u>For Coordinates B(0, 0),</u> New coordinates of corner B after shearing = (0, 0).

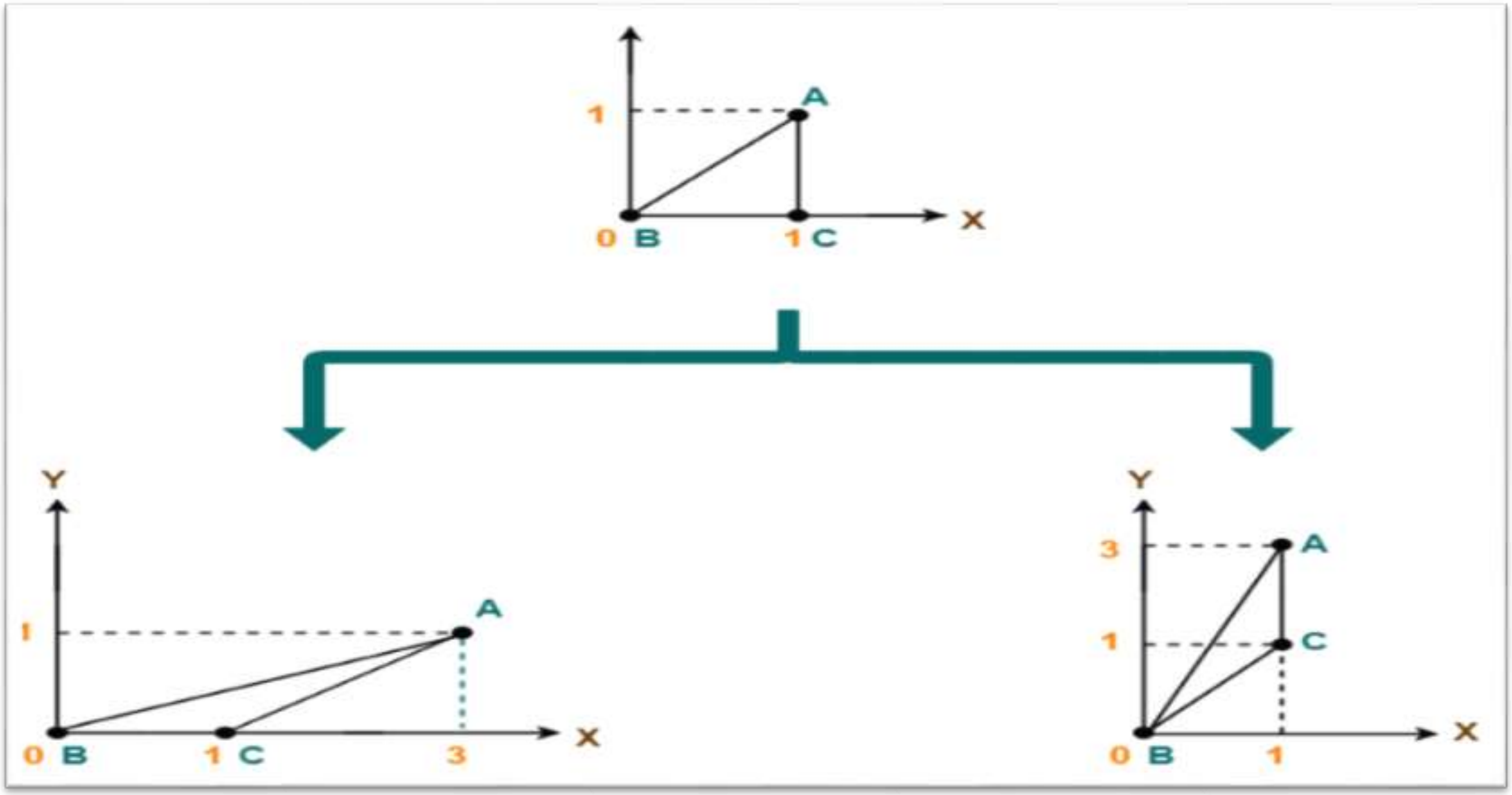
<u>For Coordinates C(1, 0),</u> New coordinates of corner B after shearing = (1, 2).

New coordinates of the triangle after shearing in Y axis = A (1, 3), B(0, 0), C(1, 2).

Parul® University

DIGITAL LEARNING CONTENT

Shear - Example



Composite Transformation :

- A number of transformations or sequence of transformations can be combined into single one called as composition. The resulting matrix is called as composite matrix. The process of combining is called as concatenation.

- Suppose we want to perform rotation about an arbitrary point, then we can perform it by the sequence of three transformations

✓ Translation
✓ Rotation
✓ Reverse Translation

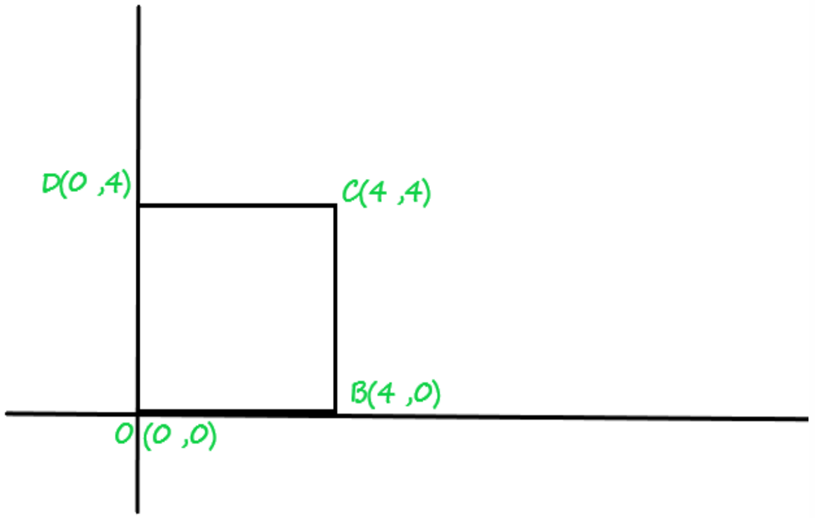
- The ordering sequence of these numbers of transformations must not be changed. If a matrix is represented in column form, then the composite transformation is performed by multiplying matrix in order from right to left side. The output obtained from the previous matrix is multiplied with the new coming matrix.

Example

- Consider we have a 2-D object on which we first apply transformation
- **T_1 (2-D matrix condition)** and then
- we apply transformation **T_2 (2-D matrix condition)** then we get the object which is transformed.
- So, we can obtain equivalent transformation by multiplying the **T_1 & T_2 (2-D matrix conditions)** with each other and then applying the **T_{12} (resultant of T_1 X T_2)** which get the transformed final image.

Example

- Consider we have a square **O(0, 0), B(4, 0), C(4, 4), D(0, 4)**
- on which we first apply **T1 (scaling transformation) given scaling factor is**
- **Sx = Sy = 0.5** and then we apply
- **T2(rotation transformation in clockwise direction)** it by 90*(angle),
- in last we perform **T3(reflection transformation about origin).**
- The square O, B, C, D looks like:



Example

- **First, we perform scaling transformation over a 2-D object:**

  Representation of scaling condition:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} X \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{pmatrix} 2 & 1 \\ 3 & 2 \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2\cdot 1 + 1\cdot 2 \\ 3\cdot 1 + 2\cdot 2 \end{pmatrix} = \begin{pmatrix} 4 \\ 7 \end{pmatrix}_{2\times 1}$$

For coordinate O(0, 0) :

$$O\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} * \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$O\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

For coordinate B(4, 0) :

$$B\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} * \begin{bmatrix} 4 \\ 0 \end{bmatrix}$$

$$B\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

Example

For coordinate C(4, 4) :

$$C\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} * \begin{bmatrix} 4 \\ 4 \end{bmatrix}$$

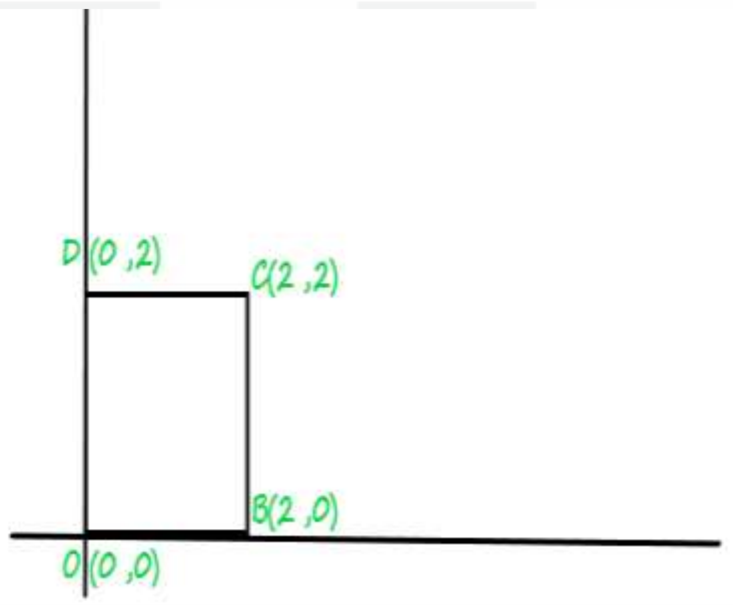
$$C\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

For coordinate D(0, 4) :

$$D\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} * \begin{bmatrix} 0 \\ 4 \end{bmatrix}$$

$$D\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

2-D object after scaling :



Example

*Now, we'll perform rotation transformation in clockwise-direction on Fig.2 by 90$^{\theta}$:

The condition of rotation transformation of 2-D object about origin is :

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}$$

$$Cos90 = 0$$

$$sin90 = 1$$

Example

For coordinate O(0, 0) :

$$O\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}0&1\\-1&0\end{bmatrix}*\begin{bmatrix}0\\0\end{bmatrix}$$

$$O\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}0\\0\end{bmatrix}$$

For coordinate B(2, 0) :

$$B\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}0&1\\-1&0\end{bmatrix}*\begin{bmatrix}2\\0\end{bmatrix}$$

$$B\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}0\\-2\end{bmatrix}$$

For coordinate C(2, 2) :

$$C\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}0&1\\-1&0\end{bmatrix}*\begin{bmatrix}2\\2\end{bmatrix}$$

$$C\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}2\\-2\end{bmatrix}$$

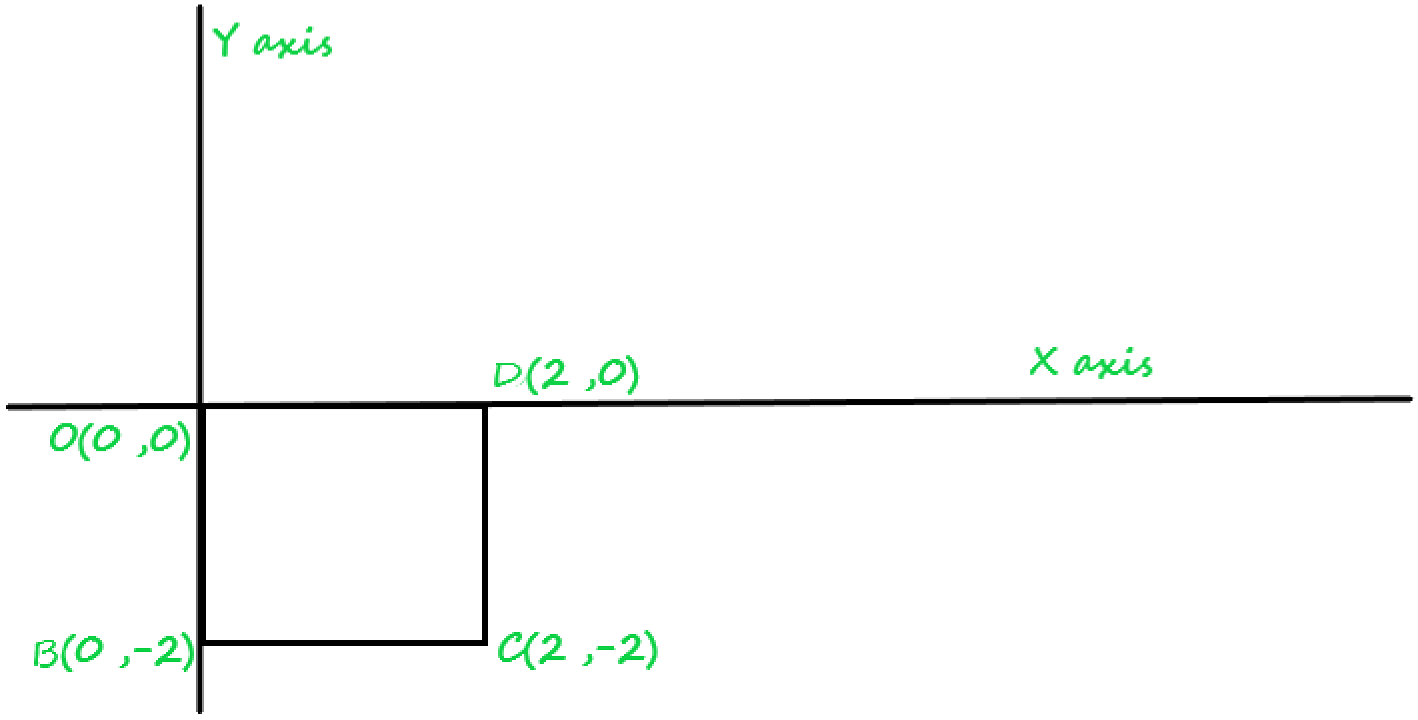
For coordinate D(0, 2) :

$$D\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}0&1\\-1&0\end{bmatrix}*\begin{bmatrix}0\\2\end{bmatrix}$$

$$D\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}2\\0\end{bmatrix}$$

2-D object after rotating about origin by 90^* angle :

Example



Example

*Now, we'll perform third last operation on Fig.3, by reflecting it about origin :

The condition of reflecting an object about origin is

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}$$

For coordinate O(0, 0) :

$$O \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} * \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$O \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Example

For coordinate B'(0, 0) :

$$B' \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} * \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

$$B' \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$$

For coordinate C'(0, 0) :

$$C' \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} * \begin{bmatrix} 2 \\ -2 \end{bmatrix}$$

$$C' \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \end{bmatrix}$$

For coordinate D'(0, 0) :

$$D' \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} * \begin{bmatrix} 0 \\ -2 \end{bmatrix}$$

$$D' \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

The final 2-D object after reflecting about origin, we get :

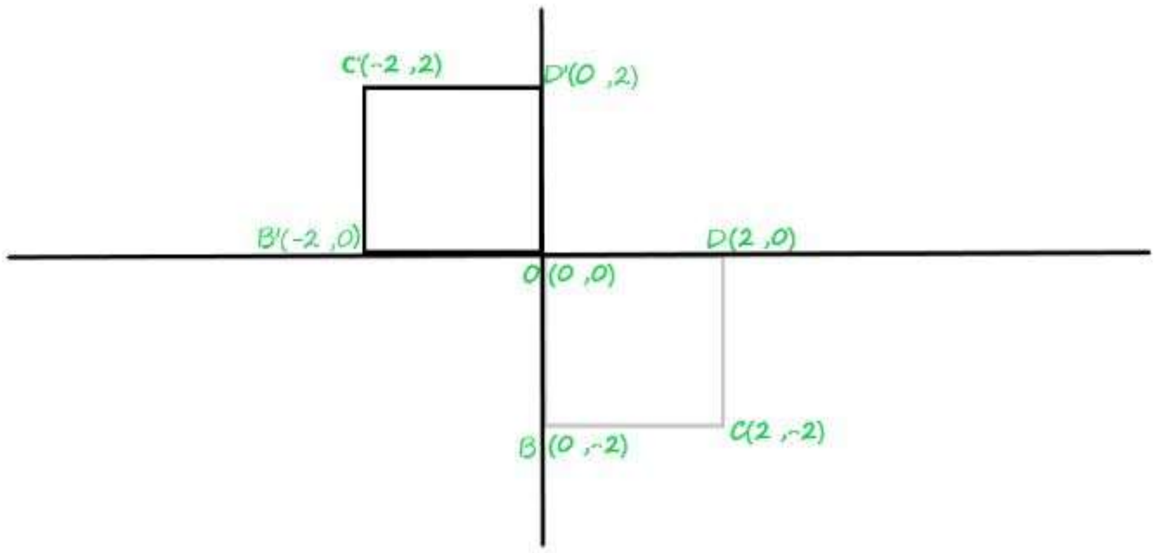


Fig.4

Example

Note : The above finale result of **Fig.4**, that we get after applying all transformation one after one in a serial manner. We could also get the same result by combining all the transformation 2-D matrix conditions together and multiplying each other and get a resultant of multiplication(R). Then, applying that 2D-resultant matrix(R) at each coordinate of the given square(above). So, you will get the same result as you have in **Fig.4.**

Solution using Composite transformation :

*First, we multiplied 2-D matrix conditions of **Scaling transformation** with **Rotation transformation :**

Example

$$[R_1] = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

$$\left[R_1\right] = \begin{bmatrix} 0 & 0.5 \\ -0.5 & 0 \end{bmatrix}$$

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

*Now, we multiplied Resultant 2-D matrix(R_1) with the third last given Reflecting condition of transformation(R_2) to get Resultant(R) :

$$[R] = \begin{bmatrix} 0 & 0.5 \\ -0.5 & 0 \end{bmatrix} * \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$\left[R\right] = \begin{bmatrix} 0 & -0.5 \\ 0.5 & 0 \end{bmatrix}$$

$$A \times B = \begin{bmatrix} A_{11}{}^{*}B_{11}+A_{12}{}^{*}B_{21} & A_{11}{}^{*}B_{12}+A_{12}{}^{*}B_{22} \\ A_{21}{}^{*}B_{11}+A_{22}{}^{*}B_{21} & A_{21}{}^{*}B_{12}+A_{22}{}^{*}B_{22} \end{bmatrix}$$

Now, we'll applied the Resultant(R) of 2d-matrix at each coordinate of the given object (square) to get the final transformed or modified object.

Parul® University

DIGITAL LEARNING CONTENT

Example

First transformed coordinate O(0, 0) is :

$$O\begin{bmatrix}x'\\y'\end{bmatrix} = \begin{bmatrix}0 & -0.5\\0.5 & 0\end{bmatrix} * \begin{bmatrix}0\\0\end{bmatrix}$$

$$O\begin{bmatrix}x'\\y'\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$$

Second, transformed coordinate B'(4, 0) is :

$$B'\begin{bmatrix}x'\\y'\end{bmatrix} = \begin{bmatrix}0 & -0.5\\0.5 & 0\end{bmatrix} * \begin{bmatrix}4\\0\end{bmatrix}$$

$$B'\begin{bmatrix}x'\\y'\end{bmatrix} = \begin{bmatrix}0\\2\end{bmatrix}$$

Third transformed coordinate C'(4, 4) is :

$$C'\begin{bmatrix}x'\\y'\end{bmatrix} = \begin{bmatrix}0 & -0.5\\0.5 & 0\end{bmatrix} * \begin{bmatrix}4\\4\end{bmatrix}$$

$$C'\begin{bmatrix}x'\\y'\end{bmatrix} = \begin{bmatrix}-2\\2\end{bmatrix}$$

Fourth transformed coordinate D'(0, 4) is :

$$D'\begin{bmatrix}x'\\y'\end{bmatrix} = \begin{bmatrix}0 & -0.5\\0.5 & 0\end{bmatrix} * \begin{bmatrix}0\\4\end{bmatrix}$$

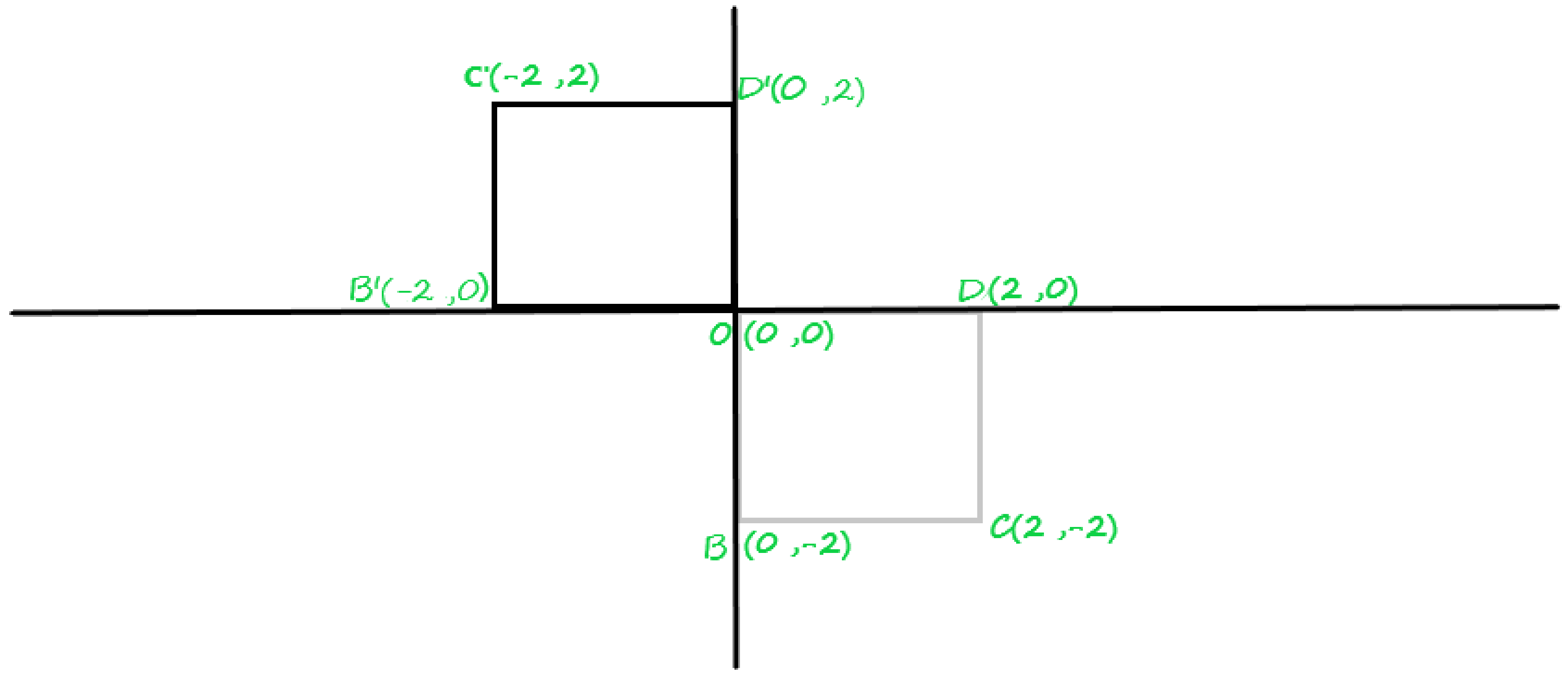
$$D'\begin{bmatrix}x'\\y'\end{bmatrix} = \begin{bmatrix}-2\\0\end{bmatrix}$$

The final result of the transformed object that you get would be same as above :

Parul® University

DIGITAL LEARNING CONTENT

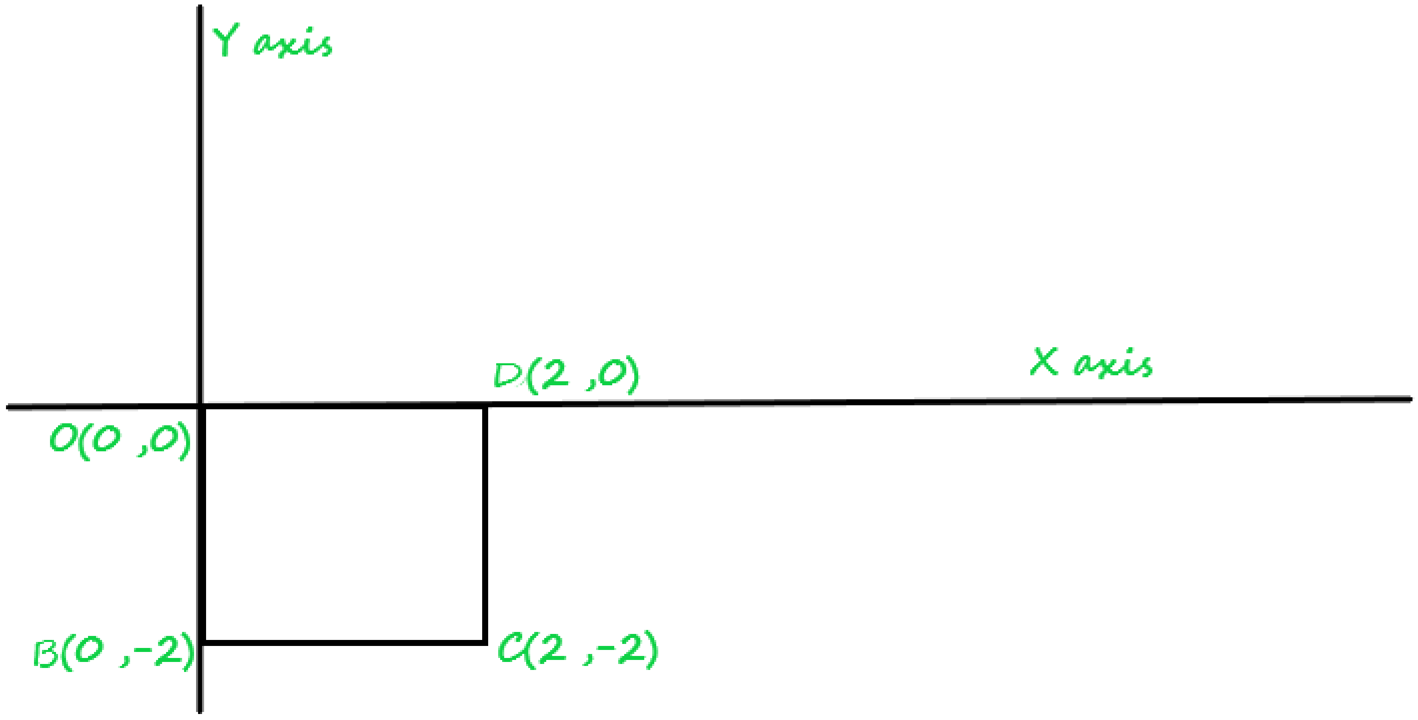
Example



Parul® University

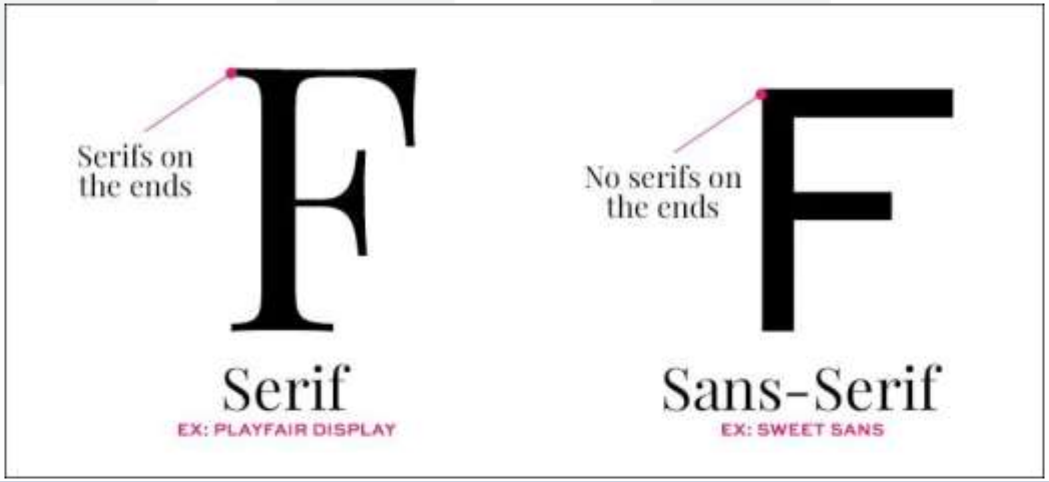
DIGITAL LEARNING CONTENT

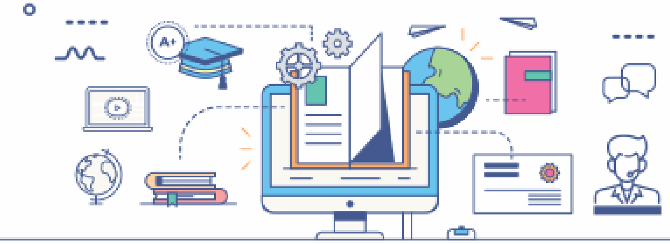
Example



Character generation

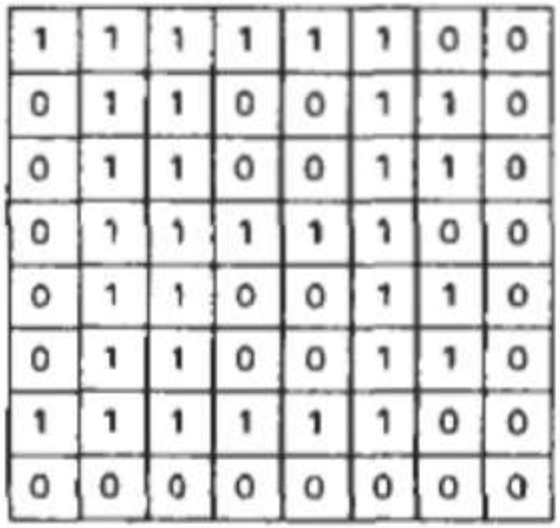
- There is variety of types of characters that we can display.
- The overall design of set of characters is called typeface. There are hundreds of typefaces available today. They are also referred as fonts.
- It can be classified mainly into two types: **Serif** and **Sans serif**.
- Serif font consists of the accents (small lines) for each character, while sans serif does not.





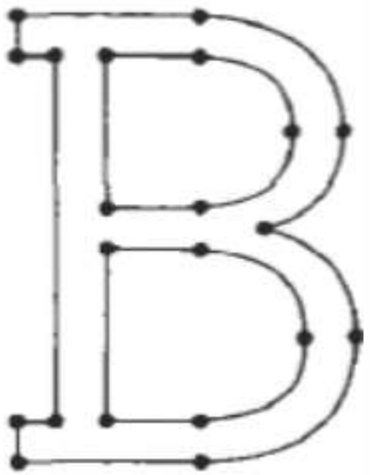
Bitmap Font / Bitmapped Font

- A simple method for representing t he character shapes in a particular typeface is to use rectangular grid patterns.
- Figure below shows pattern for particular letter



1	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0

(a)



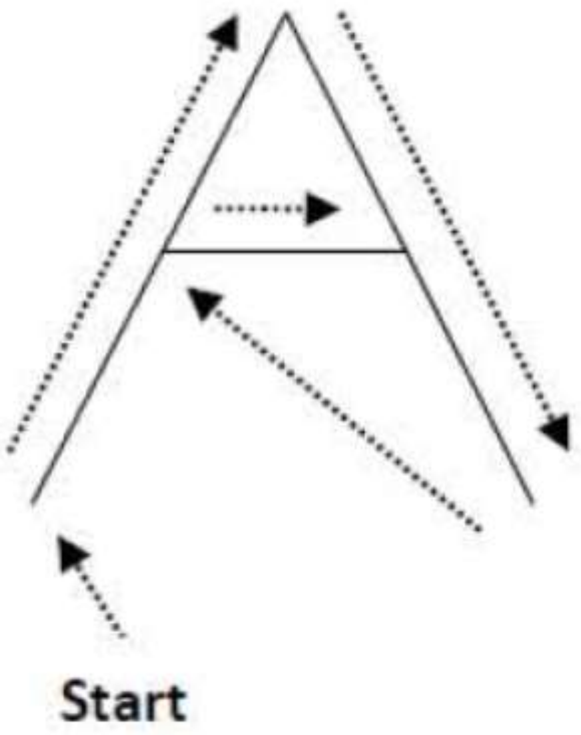
(b)

Bitmap Font / Bitmapped Font

- This method uses 2D array, where, series of 1 represents a line, and the combination of these 1's will create particular character or number.
- Bitmap fonts are the simplest to define and display as character grid only need to be mapped to a frame buffer position.
- Bitmap fonts require more space because each variation (size and format l must be stored in a font cache.

Stroke Method

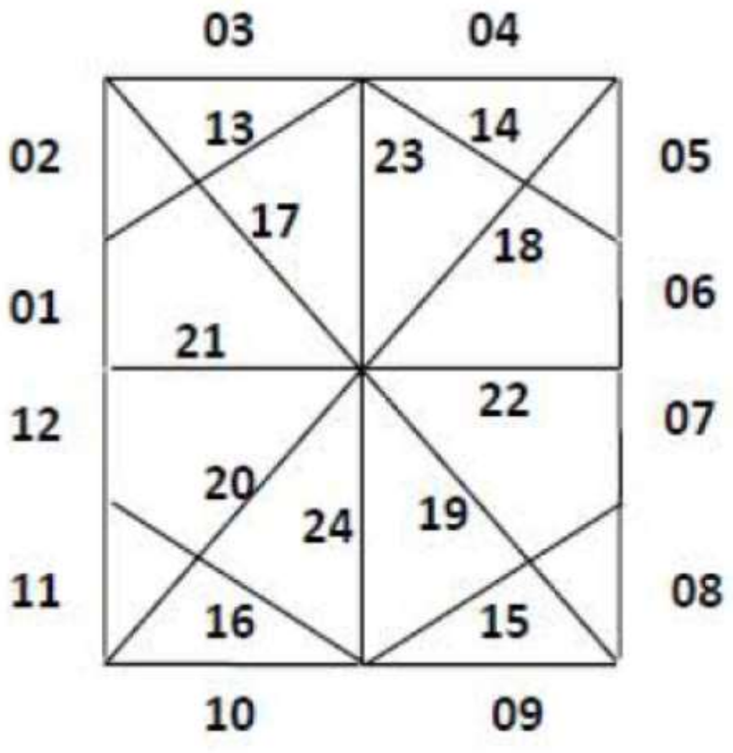
- It uses small line segments t o generate a character.
- The small series of line segments are drawn like a stroke of a pen to form a character as shown in figure .
- We can generate our own stroke method by calling line drawing algorithm.
- Here it is necessary t o decide which line segments are needs for each character and then draw that line to display character.
- It support scaling by changing length of line segment.



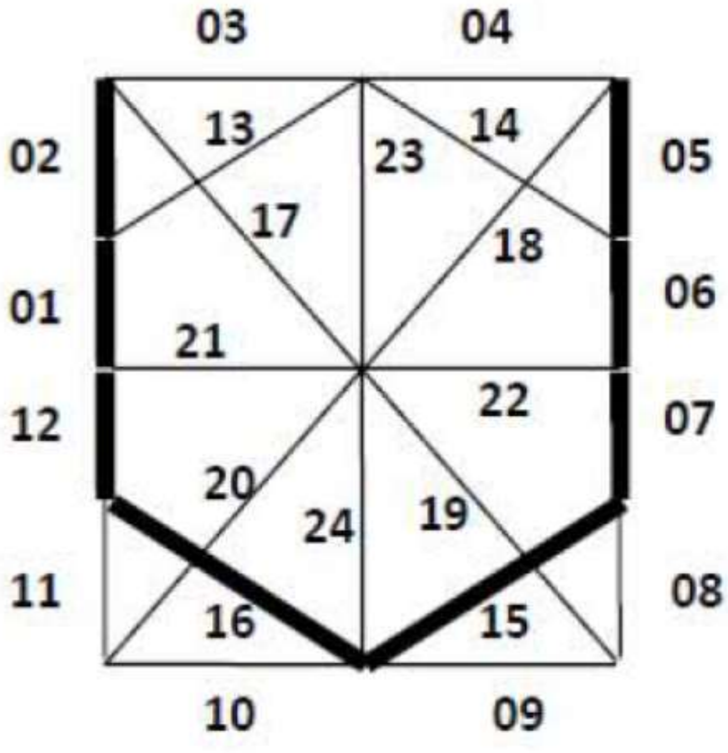
Parul® University

DIGITAL LEARNING CONTENT

Starburst Method



(a)



(b)

Starburst Method

- In this method a fix pattern of line segments are used to generate characters.
- As shown in figure there are 24-line segments are there.
- We highlight those lines which are necessary to draw a particular character.
- Pattern of particular character is stored in the form of 24-bit code. In which each bit represents corresponding line having that number.

Starburst Method

- That code contains 0 or 1 based on line segment need to highlight. We put bit value 1 for highlighted line and 0 for other line.
- Code for letter V is:
- 1 1 0 0 1 1 1 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0.
- This technique is not used nowadays because:
  1. It requires more memory to store 24-bit code for single character.
  2. It requires conversion from code to character.
  3. It doesn't provide curve shapes,

DIGITAL LEARNING CONTENT