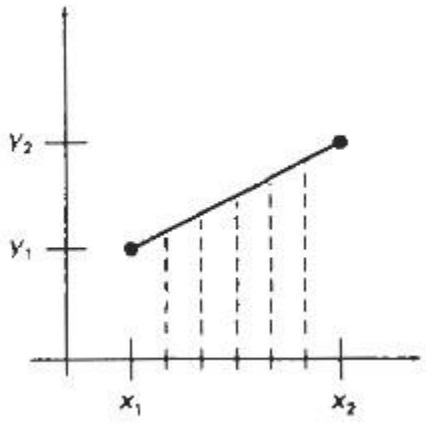
1. DDA line drawing Algorithm:

- DDA - Digital Differential Analyzer line drawing algorithm takes dx and dy into consideration for calculating the increment values for each x and y coordinates.
- The increment values found from the slope of the line according to the dx and dy.
- If the dx is greater, the increment value for the x coordinates will be constant 1 i.e. dx/dx and increment value for y coordinates will be the slope value of the line i.e. dx/dy.
- And if the dy is greater, the increment value for the y coordinates will be constant 1 i.e. dy/dy and increment value for x coordinates will be the slope value of the line i.e. dy/dx.
- The increment values are the intervals between each x and y coordinates.



- The increment values are added to every coordinate of x and y respectively.

Algorithm:

START

INPUT Point1(x1, y1) and Point2(x2, y2)

LET dx = x2 - x1, dy = y2 - y1

IF dx > dy

then,

```
        steps = abs(dx)
    ELSE
        steps = abs(dy)
    END IF

    LET xinc = dx / steps
    LET yinc = dy / steps

    LET i = 0
    WHILE i < steps
    do,
        PutPixel(x, y)
        x = x + xinc;
        y = y + yinc;
    END WHILE
END
```

2. Bresenham's line drawing algorithm:

- An efficient line algorithm relatively to the DDA line drawing algorithm
- We need to determine the next coordinate, the closest coordinate to the actual line will be the next coordinate.
- This algorithm only works for the positively slope line where x is always incremented by 1 and y is either incremented by 1 or not.
- To decide the next coordinate i.e. (x_{k+1}, y_{k+1}) either ($x_k + 1, y_k$) or ($x_k + 1, y_k + 1$).
- The difference between the new y i.e. $y = mx + b$, y and $y_k + 1$ and y and y_k
  i.e. $d1 = y - y_k$ and $d2 = y - y_k + 1$
- $d1 = (m(x_k + 1) + c) - y_k$, $d2 = (y_k + 1) - m(x_k + 1) - c$
- If $d1 - d2 < 0$, which means d2 is greater.
- To calculate d1 - d2

  $d1 - d2 = [m(x_k + 1) + b - y_k] - [y_k + 1 - m(x_k + 1) - c]$
  $d1 - d2 = m(x_k + 1) + b - y_k - y_k - 1 + m(x_k + 1) + c$

$d1 - d2 = 2m(x_k + 1) - 2y_k + 2c - 1$
$dx(d1 - d2) = 2dy\ (x_k + 1) - 2y_k dx + 2cdx - dx$
$dx(d1 - d2) = 2dyx_k - 2dxy_k + 2dy + 2dxc - dx$

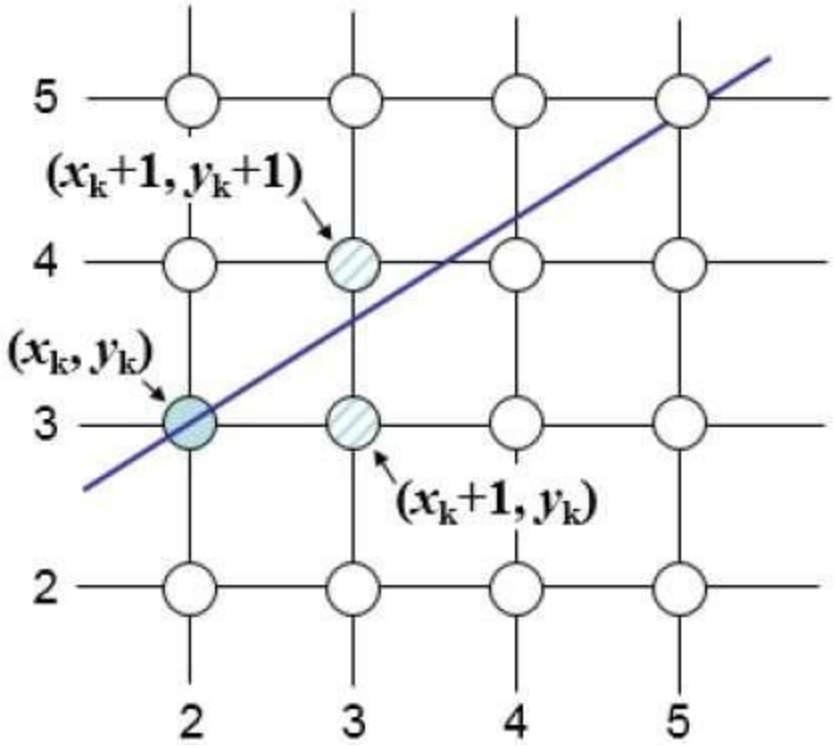
LET $C = 2dy + 2dxc - dx$ be the constant

$dx(d1 - d2) = 2dyx_k - 2dxy_k + C$
$p_k = 2dyx_k - 2dxy_k + C$

- Now the next difference d1 - d2 is. $p_{k+1} = 2dyx_x - 2dxy_k + C$
- Calculate $p_{k+1} - p_k$
  $p_{k+1} - p_k = 2dyx_{k+1} - 2dxy_{k+1} + C - 2dyx_k + 2dxy_k - C$
  $p_{k+1} - p_k = 2dyx_{k+1} - 2dxy_{k+1} - 2dyx_k + 2dxy_k$
  $p_{k+1} - p_k = 2dy(x_{x+1} - x_k) - 2dx(y_{k+1} - y_k)$

  $p_{k+1} = p_k + 2dy(x_{x+1} - x_k) - 2dx(y_{k+1} - y_k)$

- If the $p_{k+1} - p_k < 0$
  $p_{k+1} = p_k + 2dy$
  and if not
  $p_{k+1} = p_k + 2dy - 2dx$
- To first take the decision at initial step we need to find the initial decision parameter
  $p_1 = 2dyx_1 - 2dxy_1 + 2dy + 2dxb - dx$
  $p_1 = 2dyx_1 - 2dxy_1 + 2dy + 2dx(y1 - dy/dx(x1)) - dx$
  $p_1 = 2dy - dx$



Algorithm:

START

INPUT Point1(x1, y1) and Point2(x2, y2)

LET x = x1, y = y1
LET dx = x2 - x1, dy = y2 - y1

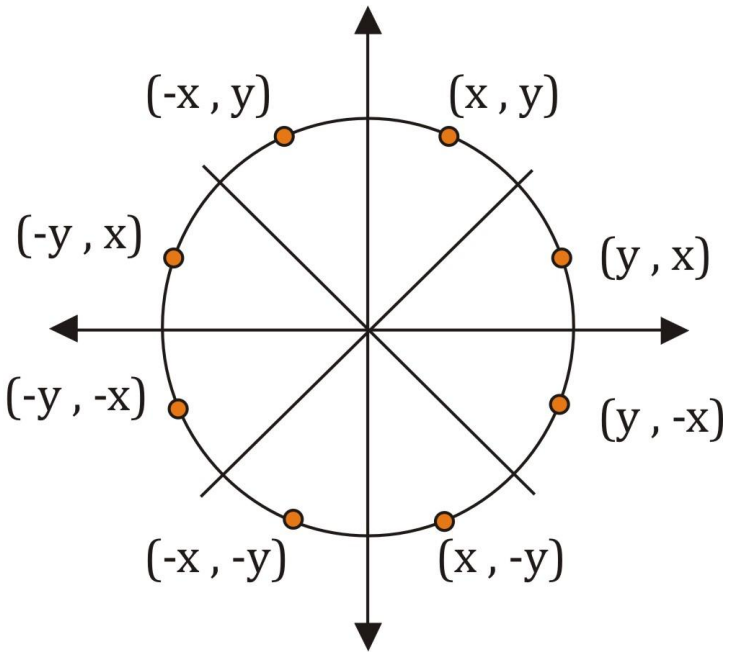
LET p = 2dy - dx

WHILE x <= x2
do,
SetPixel(x, y)
IF p < 0
then,
p = p + 2dy
ELSE
p = p + 2dy - 2dx

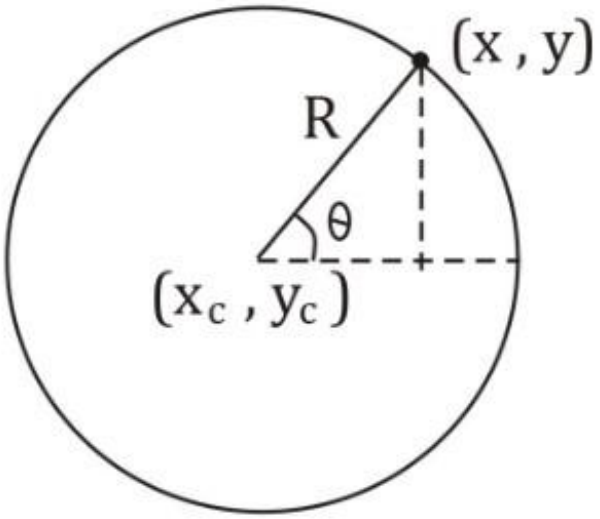
```
            y = y + 1
        END IF

        x = x + 1
    END WHILE
END
```

3. Circle drawing algorithm:

- Symmetry of the circle:
- A circle can be divided into 8 segments.
- Divide a circle into 4 quadrants by passing two perpendicular lines at the center of the circle.
- Dividing each quadrant with 45°, providing us with 8 different octants as shown in the image below:



- Circle Equation Algorithm:
  Circle equation i.e. $x^2 + y^2 = r^2$
  where, x = x coordinate
  y = y coordinate
  r = circle radius



- Each of the x coordinates in the segments 90^o to 45^o is found by stepping x from 0 to r / $\sqrt{2}$ and each of the y coordinates is found by evaluating $\sqrt{r^2 - x^2}$ for each step of x.

  Algorithm:

  START

  INPUT r
  (h, k) = coordinates of circle center
  x = 0
  i = step size
  xend = r / $\sqrt{2}$

  WHILE x > xend
  do,
  $y = \sqrt{r^2 - x^2}$

  symmetry(x + h, y + k)
  symmetry(-x + h, -y + k)
  symmetry(y + h, x + k)
  symmetry(-y + h, -x + k)
  symmetry(-y + h, x + k)
  symmetry(y + h, -x + k)
  symmetry(-x + h, y + k)

```
            symmetry(x + h, -y + k)

            x = x + i
        END WHILE
END
```

- Bresenham's circle drawing algorithm:
- To select the closest pixel point to the circle arc, after incrementing a point we need to calculate the next point according to it.

- xc and xy are the center points of the circle and r is radius
- The x = 0 is initialized and y is radius.
- First we need to find the initial decision parameter d = 3 - 2r is found by it we can decide whether we need to decrement the y or not.
- If d is less than 0 i.e. negative we need to decrement the y and also calculate d = d + 4x + 6.
- If it is more than 0 i.e. positive we do not need to decrement the y, but need to calculate d = d + 4(x - y) + 10.
- Pixels are put as per the circle symmetry as explained above, If you put a pixel for one coordinate it must have symmetry for another 7 different points.
- In symmetry the center of the circle is added to make the circle from the center of the circle.

```
START
        INPUT xc, xy, r
        LET x = 0, y = r
        LET d = 3 - 2r
```

```
        WHILE x < y
        Do,
                Symmetry(xc + x, yc + y)
                Symmetry(xc + x, yc - y)
                Symmetry(xc - x, yc + y)
                Symmetry(xc - x, yc - y)
                Symmetry(xc + y, yc + x)
                Symmetry(xc + y, yc - x)
                Symmetry(xc - y, yc + x)
                Symmetry(xc - y, yc - x)

                IF d < 0
                then,
                        d = d + 4x + 6
                ELSE
                        d = d + 4(x - y) + 10
                        y = y - 1
        END WHILE
END
```

4. Inside Test:

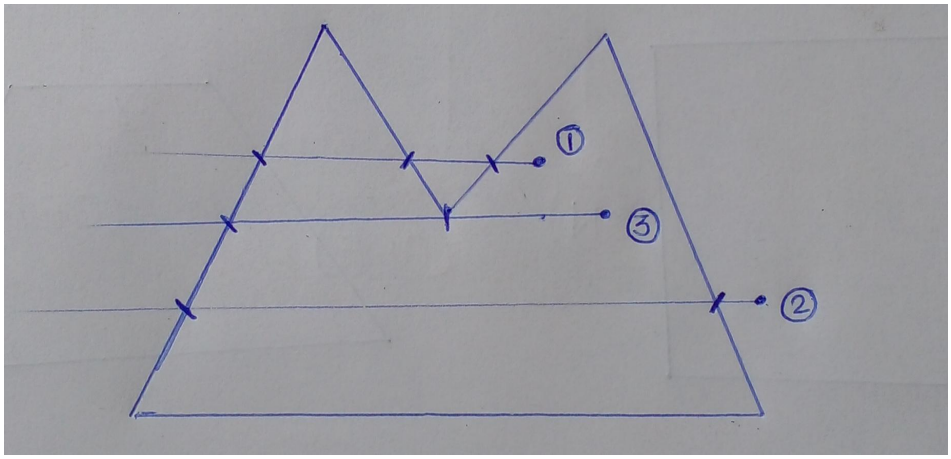
- To Determine whether a point is inside or outside of the polygon Inside test is used.

  There are two method for inside test:

  a. Odd - Even Parity Method
  b. Winding Number Method

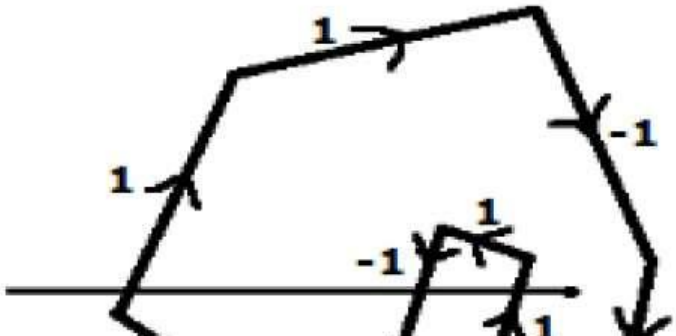
a. Odd - Even Parity Method:

- The point which is to be inside tested, a ray is drawn from that point, if the ray intersects with the sides of the polygon an odd number of times, the point is inside of the polygon or if the ray intersects with the sides of the polygon an even number of times, the point is outside of the polygon.

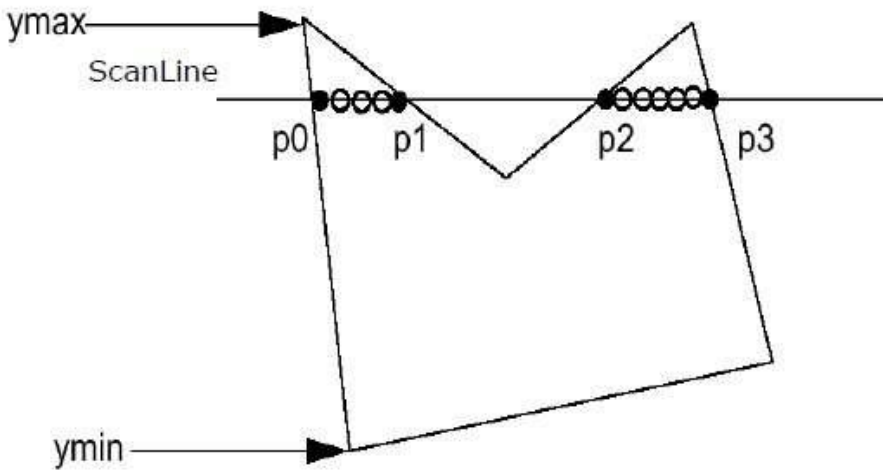


b. Winding Number Method:

- To count the number of times polygon edges wind around the point is called the winding number method.
- The point which is to be inside tested, a ray is drawn from that point, the ray intersects the sides of the polygon.
- At Initial winding number W is 0, If the side of the polygon is going from right to left we add a 1 to winding number W and If the side of the polygon is going from left to right we subtract a 1 from that winding number W.
- At the end if the winding number W is Non-zero then the point is considered to be inside of the polygon. Otherwise, outside of the polygon.



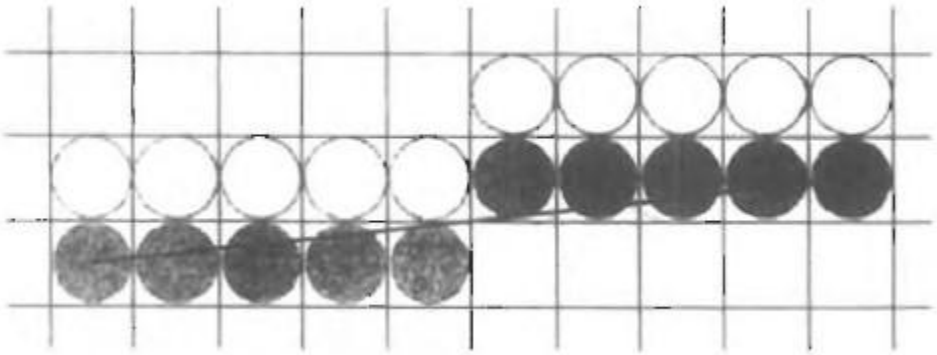
5. Polygon Filling method:
    a. Scan - line fill Method:
    - Scan lines are passed through the polygon from minimum y to maximum y and minimum x to maximum x.
    - The Points that intersect the polygon border are used to make a line of the desirable fill color.
    - Collectively these all scan lines form the Polygon.



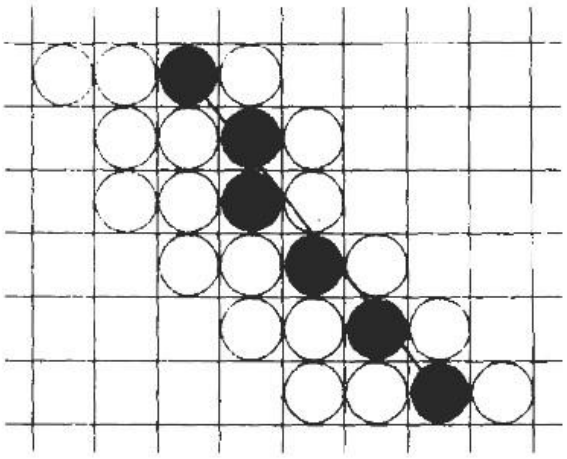
    b. Flood fill Method:
    - In flood fill an arbitrary point is taken from inside of the polygon.
    - The current point (x, y) is filled with a new color, and moved to points at top(x, y-1), bottom(x, y+1), left(x-1, y) and right(x+1, y).
    - If the next point is already filled, they are ignored and moved to the point from top(x, y-1), bottom(x, y+1), left(x-1, y) and right(x+1, y).
    - If the next point is not already filled, it is then filled and moved to the next point, any from top(x, y-1), bottom(x, y+1), left(x-1, y) and right(x+1, y).
    - After it is done the next point is considered as the current point and the process is repeated till all the points are not replaced with the fill color.

6. Line style:
    - Thick line:

- Thick line is generated by plotting the same line some number of pixels i.e. 1 above the original line. Thus, giving us a thick line double the original width.
- The number of pixels, depends on the width of the line.



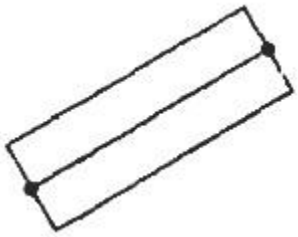
- If the magnitude of the line is greater than 1, then the plotting must be done vertically on the original line.



- Thick line caps:
    1. Butt caps
    2. Round caps
    3. Projecting square caps

1. Butt caps:

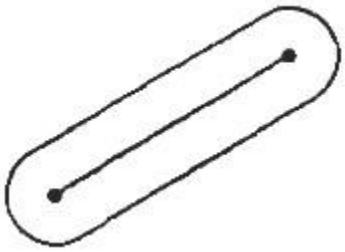
- Butt caps obtained by adjusting the end positions of the component parallel lines so that the thick line is displayed with square ends that are perpendicular to the line path.



Butt Cap

2. Round caps:

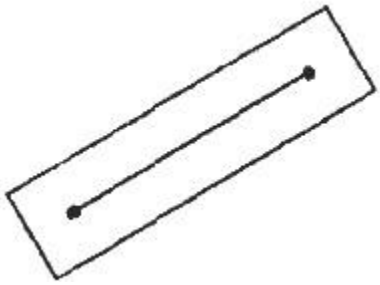
- Round cap is obtained by adding a filled semicircle to each butt cap. The circular arcs are centered on the line endpoints and have a diameter equal to the line width.



Round Cap

3. Projecting square caps:

- Projecting square cap is obtained by extending the line and adding butt caps that are positioned one-half of the line width beyond the specified endpoints.



6. Font:

- Font also known as typeface, which is a set of characters with a particular design style such as New York , Courier, Helvetica, London, Times Roman , and various special symbol groups.
- The Characters in a selected font can also be displayed with assorted underlying styles (solid, dotted, double), **boldface**, *in italics*, and in outline or shadow styles.

7. Scaling:

- To change the scale of an object to scale up or to scale down is called Scaling.
- To scaling factors i.e. s_x and s_y for x and y coordinates respectively are multiplied with x and y coordinates to get the new x_{new} , y_{new} coordinates.

  Scaling Matrix:

$$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

  Finding new Coordinates i.e. x' and y' after Scaling:

$$[\,x'\ \ y'\,] = [\,x\ \ y\,] * \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

Original Object

- Uniform Scaling:
  The scaling in which scaling factors i.e. s_x and s_y are the same is called Uniform scaling.

Uniform Scaling

- Differential Scaling:
  The scaling in which scaling factors i.e. s_x and s_y are different is called Differential Scaling.

Differential Scaling

8. Translation - Including homogeneous coordinates:

To change the position of an object from one location to another is called Translation.

For a normal point

i.e. $x' = t_x + x, y' = t_y + y$

Translation matrix 1d: $[t_x \; t_y]$

Homogeneous coordinates:

If we want to multiply a 2 X 2 matrix with a 3 X 3 matrix we need to expand the 2 X 2 matrix, providing we also expand the coordinate positions. With the help of cartesian coordinate position (x, y) with the homogeneous coordinate triple (x_h , y_h , h).

The translation matrix cannot be combined with another 2d homogeneous matrix, so to enable it to be combined with another matrix, homogeneous coordinates are included in the original matrix to make it a 2d homogeneous matrix, so we could combine it with another matrix.

Translation homogeneous matrix 2d:

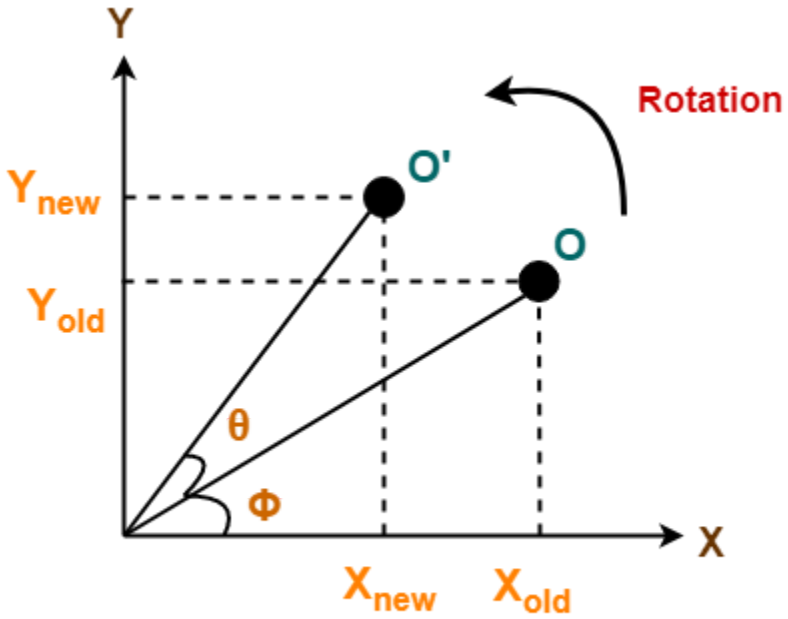
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

Translation matrix with homogeneous coordinates:

$$[x' \; y' \; 1] = [x \; y \; 1] * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

10 : Basic Rotation

- Rotation is a motion of a certain space that preserves at least one point. There are two types of rotations: **Clockwise** and **Counter-clockwise**.
- To rotate an object, we have to specify the angle of rotation which is called **Pivot Point**.



2D Rotation in Computer Graphics

- The above image represents a counterclockwise rotation. The following derivations are for counterclockwise rotations.
- In this figure, the rotation r is the constant distance of the point from the origin, angle ϕ is the original angular position of the point from the horizontal, and **θ** us the rotation angle. Using standard trigonometric identities, we can express the transformed coordinates in terms of angles **θ** and ϕ as

$$X' = r\cos(\phi + \theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta$$
$$Y' = r\sin(\phi + \theta) = r\cos\phi\sin\theta - r\sin\phi\cos\theta \quad (1)$$

- The original coordinates of the point in polar coordinates are

$$X = r\cos\phi, \qquad Y = r\sin\phi \quad (2)$$

- Substituting (1) into (2), we get a transformation equations for rotating a point(x, y) through an angle **θ** about the origin:

$$X' = x\cos\theta - y\sin\theta$$
$$Y' = x\sin\theta + y\cos\theta \quad (3)$$

- The above equation can be represented in matrix like this:

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

- Just like translations, rotations are transformations that move objects without deformation. Every point on an object is rotated through the same angle.

11: Fixed Point Scaling

- When scaling is done by choosing a particular position (called **Fixed Point**) , which will remain unchanged after the scaling transformation is done, this type of scaling is called **fixed point scaling**.
- Commonly, a vertex of an object or the center point is used as a fixed point for such scaling operations. Remaining vertices are scaled relative to the fixed point by scaling the distance from each vertex to the fixed point.
- For a vertex having coordinates (x, y), the scaled coordinates (x', y') are calculated as

$$X' = x_f + (x - x_f)S_x, \qquad Y' = y_f + (y - y_f)s_y \quad (4)$$

- Where x_f and y_f are fixed points.
- These equations can be written as follows:

$$X' = x \cdot s_x + x_f(1 - s_x)$$
$$Y' = y \cdot s_y + y_f(1 - s_y) \qquad (5)$$

12. Fixed point rotation

- Fixed point rotation is performed in 3 steps
  1. Translating the object to origin
  2. Performing rotation
  3. Translating the object back to its original position
- These steps are represented in matrix form below

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -t_x & -t_y & 1 \end{bmatrix} \begin{bmatrix} \cos\Theta & \sin\Theta & 0 \\ -\sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -t_x & -t_y & 1 \end{bmatrix} \begin{bmatrix} \cos\Theta & \sin\Theta & 0 \\ -\sin\Theta & \cos\Theta & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

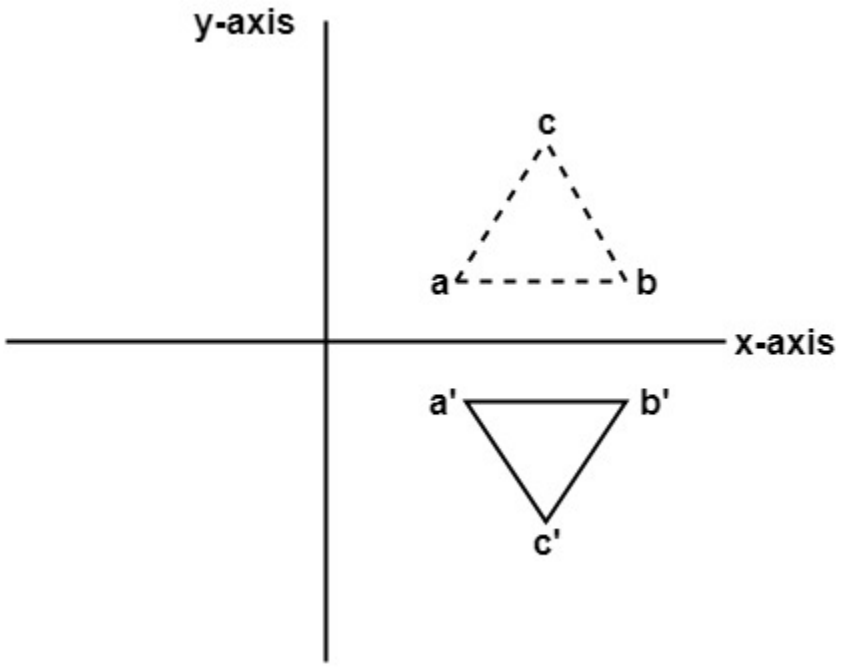
$$= \begin{bmatrix} \cos\Theta & \sin\Theta & 0 \\ -\sin\Theta & \cos\Theta & 0 \\ -t_x\cos\Theta + t_y\sin\Theta + t_x & -t_x\sin\Theta - t_y\cos\Theta + t_y & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\Theta & \sin\Theta & 0 \\ -\sin\Theta & \cos\Theta & 0 \\ t_x(1-\cos\Theta) + t_y\sin\Theta & t_y(1-\cos\Theta) - t_x\sin\Theta & 1 \end{bmatrix} \qquad (a)$$

- The equation (a) is called a composite transformation matrix for fixed point rotation.

13. Reflection

- A reflection is a transformation that produces a **mirror image** of an object. This is performed relative to an axis of reflection by rotating the object 180° about the reflection axis.



- The above image shows a reflection on line y = 0.
- Reflection about the line y=0, the x-axis, is accomplished with the following transformation matrix.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (6)$$

- This transformation changes the y values of coordinates. The final rotation will look like the above image where abc are the original position and a'b'c' are the reflected object.
- A reflection about the line x=0, the y-axis is accomplished with the following transformation matrix.

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (7)$$

- To flip both the x and y coordinates is accomplished with the following transformation matrix.

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (8)$$

- If we choose to perform a rotation on y = x, the following transformation matrix is used.

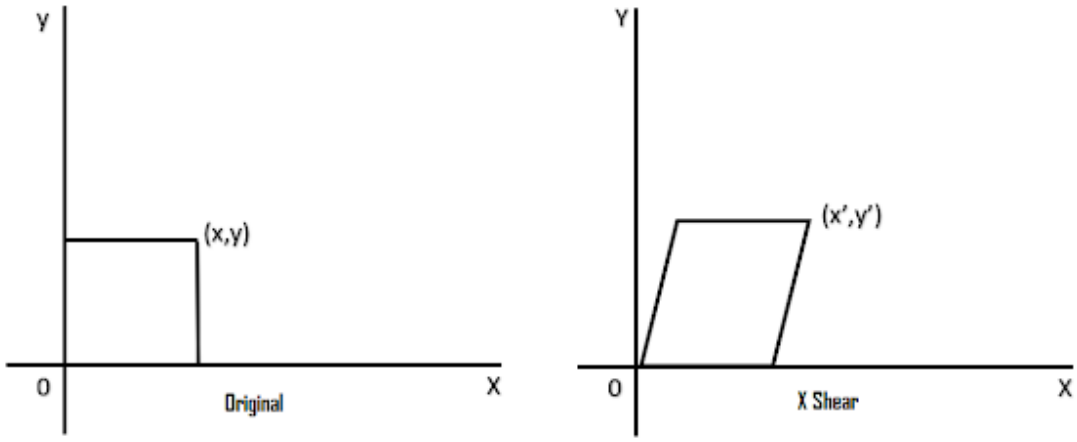
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (9)$$

- For rotation at y = -x, we use the following transformation matrix.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (10)$$

14. Shear

- A transformation that distorts the shape of an object such that the transformed shape appears as if the objects were composed of internal layers that had been caused to slide over each other is called a shear.



- The commonly used shearing transformations are done on x-axis (called x shear) and y-axis (called y shear)
- The x shear transformation matrix is given below

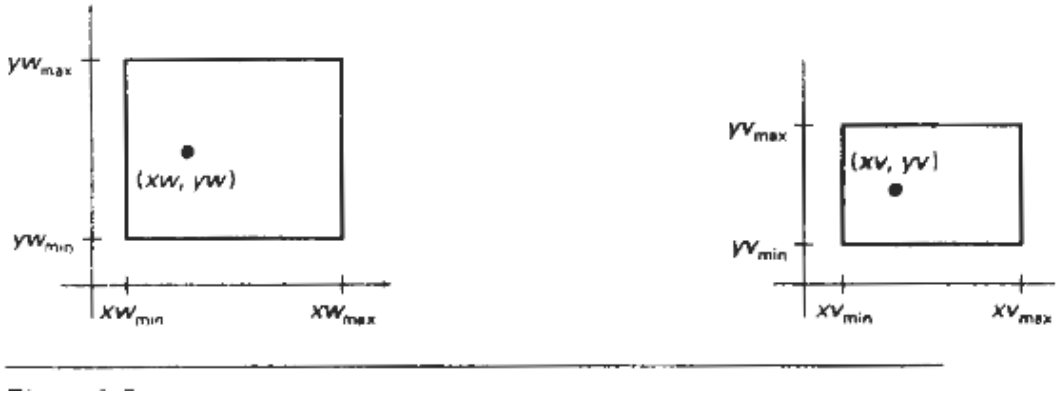
$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (11)$$

- The y shear transformation matrix is given below

$$\begin{bmatrix} 1 & 0 & 0 \\ Sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (12)$$

15. Window to viewport transformation

- When the object is in the viewing reference frame , we have to choose the window extents in viewing coordinates and select the viewport limits in normalized coordinates. Then the object is transferred to normalized device coordinates. We do this using a transformation that maintains the same relative placement of objects in normalized space as they had in viewing coordinates.



(b) Window to viewport coordinate transformation

- Figure (b) shows window-to-viewport mapping. A point at position (xw, yw) in the windows is **Mapped** into position (xv, yv) in the associated viewport. To maintain the same relative placement in the viewport as in the windows, we require that

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}} \qquad (13)$$

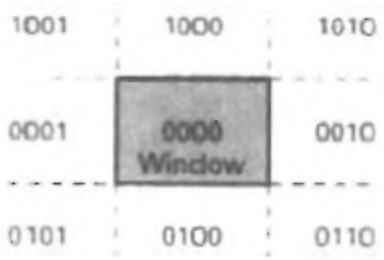
- Solving these expressions for the viewport position (xw, yv) we have

$$S_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$$

$$S_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}} \qquad (14)$$

16. Cohen - Sutherland

- This is a line clipping algorithm that speeds up the processing of line segments by performing initial tests that reduce the number of intersections that must be calculated. Every line endpoint in a picture is assigned a four-digit binary code, called a **region code**. This identifies the location of the point relative to the boundaries of the clipping rectangle.



- The above figure is an example of the binary codes where bit 1 represents left, 2 for right, 3 for below and 4 for above the clipping window.
- Bit values in the region code are determined by comparing endpoint coordinate values (x, y) to the clip boundaries.
- Once we set region code for all the endpoints, we can determine which lines are completely inside the clip windows and which are clearly outside.
- Lines that are inside the window having a region code 0000 are accepted. Lines having a 1 in the same bit position in the region codes for each endpoint are completely outside the clipping rectangle and we reject these lines.
- TO test lines for total clipping is to perform the logical *and* operation with both region codes. If the result is not 0000, the line is completely outside the clipping region.
- If the line endpoints are passing through multiple regions, we find the intersection point until we get the points that are completely inside the clipping window.

16.2 Liang- Barsky line clipping

- Liang-Barsky line clipping method uses the parametric equation of line to figure out the intersection points at the clipping window.

$$X = x_1 + u\Delta x$$
$$Y = y_1 = u\Delta y \quad (15)$$

- Where $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$.
- The point clipping conditions in the parametric form are represented as follows:

$$Xw_{min} <= x_1 + u\Delta x <= xw_{max}$$
$$Yw_{min} <= y_1 + u\Delta y <= yw_{max} \quad (16)$$

- The equation can be expressed as

$$Up_k <= q_k, \qquad k=1,2,3,4 \quad (17)$$

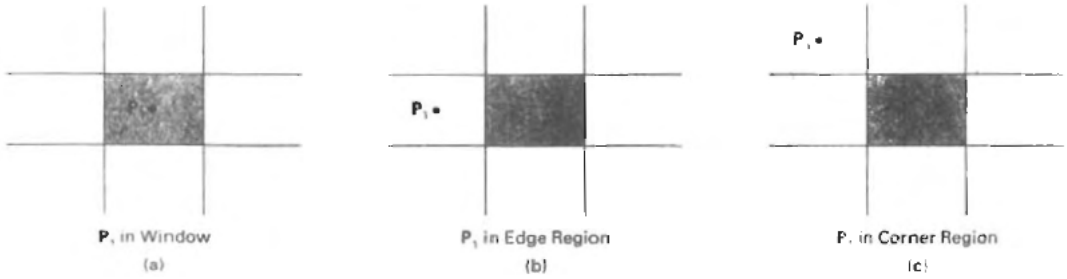
- Where $p_1 = -\Delta x, \quad q_1 = x_1 - xw_{min}$
  $p_2 = \Delta x, \quad q_2 = xw_{max} - x_1$
  $p_3 = -\Delta x, \quad q_3 = y_1 - yw_{min}$
  $p_4 = -\Delta x, \quad q_4 = yw_{max} - y1 \quad (18)$
- Lines that are parallel to the clipping boundaries have $p_k = 0$, k = 1,2,3,4 correspond to the left, right, bottom and top boundaries, respectively.
- For a non zero value of p_k we calculate the value of u that is the intersection point.

$$U = q_k / p_k \quad (19)$$

- We take the maximum of u for the pk<0 , and minimum for pk>0.
- If u1>u2, the line is completely outside the clipping window and it can be rejected. Otherwise, the endpoints of the clipped line are calculated from the two values of parameter u.

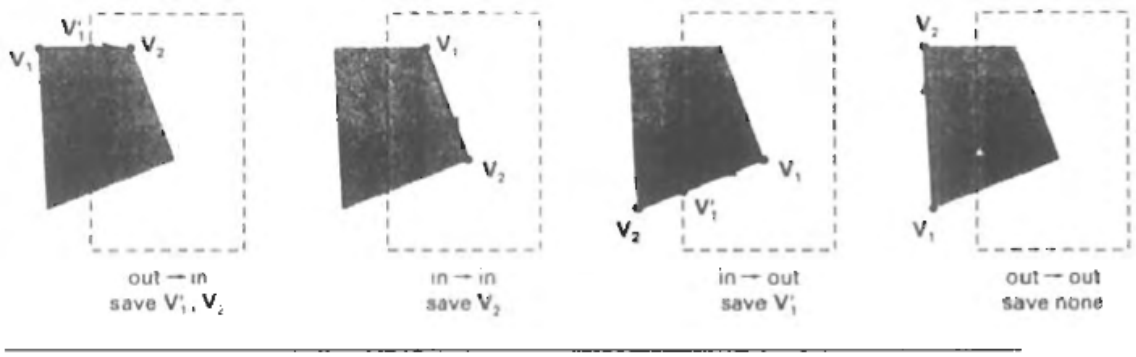
16.3 NLN/Midpoint Subdivision

- The Nicoll-Lee-Nicoll algorithm avoids multiple clipping of an individual line segment.
- The algorithm can only be applied to the two-dimensional clipping whereas Liang-Barsky and Cohen-Sutherland methods can also be used for three-dimentional clipping.



- For a line with endpoints p_1 and p_2 we find out the position of p_1 for the possible regions relative to the clipping rectangle.
- Only the three regions shown in the diagram are considered and if the point is in a different region, we move it to any of the three regions using reflection or rotation method.
- After that, we determine p_2 relative to p_1.
- Then the intersection point between the regions are found out that contains p_2. If both p_1 and p_2 are inside the clipping rectangle we save the entire line.
- If p_1 is in the region to the left of the windows, we set up the four regions L,LT,LR and LB,
- If p_2 is in the region L, we clip the line at the left boundary and save the line segment from this intersection point to p_2.
- If $p_{2,}$ is in the region LT, we save the line segment from this intersection point to p_2.
- If p_2 is in the region LT, we save the line segment from the left window boundary to the top boundary.
- If it is not in any of the four regions, the entire line is clipped.

17. Sutherland-hodgmen polygon clipping



- There are four possible cases when processing vertices in sequence around the perimeter of a polygon,
- Each pair of polygon vertices is passed to a window boundary clipper and the following test are made:
    - If the vertex is outside the windows boundary and the second vertex is inside, both the intersection point of the polygon edge with the window boundary and the second vertex are added to the output vertex list.
    - If both vertices are inside the window boundary, only the second vertex is selected.
    - If the first vertex is inside the boundary, the edge intersection with the window boundary is selected.
    - If both the vertices are outside the windows, nothing is added to the output list.

- After the vertices are processed for one clip window boundary, the output vertices are displayed.

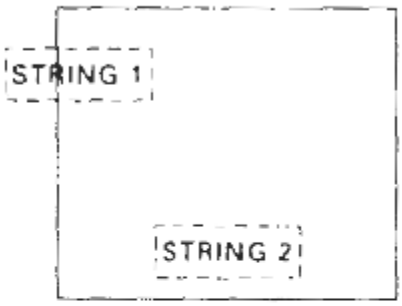
18. Exterior clipping

- When we clip a picture to the exterior of a specified region, the picture parts to be saved are those that are outside the region. This is known as exterior clipping.

- One of the examples of exterior clipping is the multiple window systems. When higher-priority windows overlap another window, the objects are clipped to the exterior of the overlapping windows for better user experience.
- This technique can also be used for combining graphs, maps, or schematics.

19. Text clipping

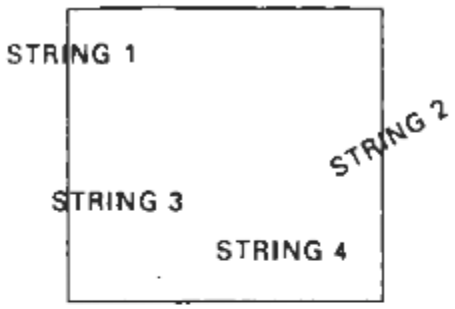
There are multiple methods to perform text clipping.
The simplest method is all-or-none string-clipping in which we discard the string unless all of the string is inside a clip window as shown in the figure below.



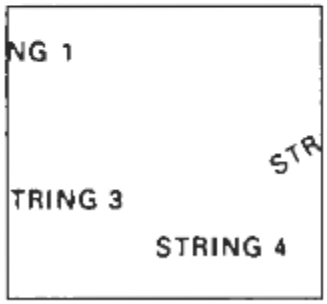
Before Clipping

After Clipping

- The other method is known as all-or-none character-clipping where we discard only the characters that are not completely inside the window as shown in the figure below.



Before Clipping



After Clipping

- The final method for handling text clipping is to clip the components of individual characters. Here, the characters are treated as lines and if an individual character overlaps a clip window boundary, we clip off the parts of the characters that are outside the window.

20. Animation

- The term computer animation generally refers to any time sequence of visual changes in a scene.
- Animation is a combination of all the transformations happening at a very rapid rate that gives an illusion as if the object is moving.

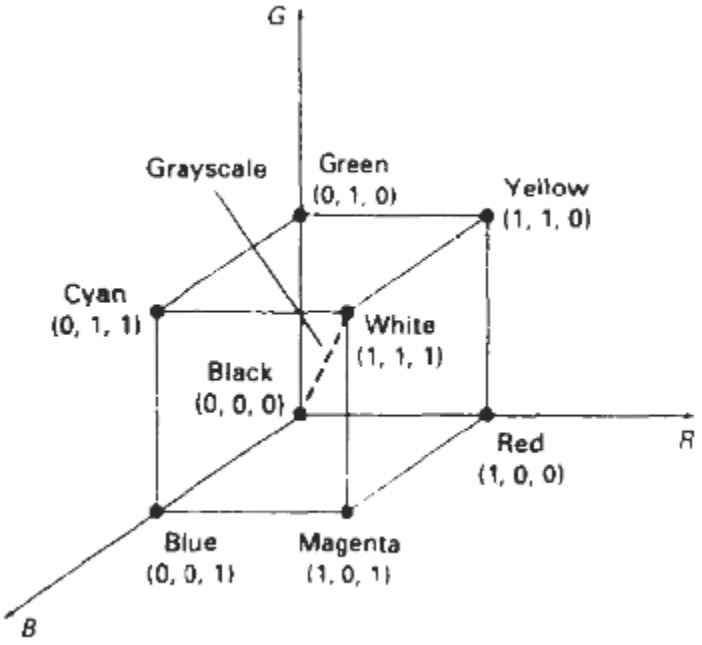
- These sequences are designed with the following steps:
    - Storyboard layout
    - Object definitions
    - Key-frame specifications
    - Generation of in-between frames
- The storyboard defines the motion sequence as a set of basic events. They consist of a set of rough sketches or a list of basic ideas for the motion.
- The objects are defined in terms of shapes and associated movements are specified along with them.
- The key frame is a detailed drawing of the scene at a certain time in the animation sequence where each object is positioned according to the time for that frame.
- The in-betweens are the immediate frames between the key frames. A movie uses 24 frames per second and the terminals are refreshed at 30 to 60 frames per second.
- Some steps in the development of an animation sequence include object manipulations and rendering, camera motions, and the generation of in-betweens,
- The well known animation packages are : OpenGL, Blender, Maya
- On raster systems, we can generate real-time animation in limited applications using rater operations.
- Animation functions include a graphics editor, a key-frame generator, an in-between generator, and standard graphics routines.
- A general-purpose language, such as C, Python, C# and C++ are often used to program the animation functions.
- There are two types of systems that is used to develop animation
    1. Key-frame systems
    2. Parameterized systems
- Keyframe systems are specialized animation languages designed simply to generate the in-betweens from the user-specified key frames.

- Parameterized systems allow object-motion characteristics to be specified as part of the object definitions. The adjustable parameters control such object characteristics as degrees of freedom, motion limitations, and allowable shape changes.
- Another type of animation is morphing. Transformation of object shapes from one form to another is called **Morphing** which is a shortened form of metamorphosis. They are applied to any motion or transition involving a change in shape.

21. Color Model

21.1 RGB Color Model

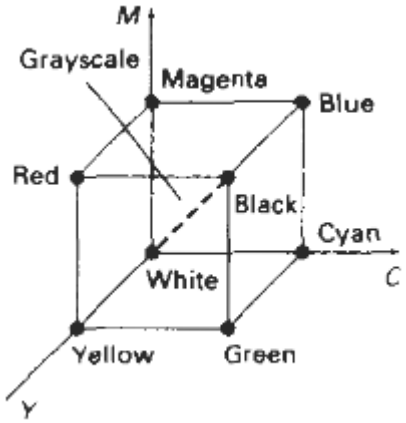
- RGB color model is based on how our eyes perceive color through the stimulation of three visual pigments in the cones of the retina.
- By comparing intensities in a light source, we perceive the color of the light.
- This model has three color primaries, red, green, and blue,



- We can represent this model with the unit cube defined on R, G and B axes as shown in above figure. The origin represents black, and the vertex with coordinates (1,1,1) is white.
- This color model is an additive model because the intensities of the primary colors are **added** to produce various colors.
- A color c is expressed in RGB components as
    - C = R**R** + G**G** + B**B**
- The magenta vertex is obtained by adding red and blue to produce to triple (1, 0, 1) and white at (1, 1, 1) is the sum of the red, green, and blue vertices.
- Shades of grey are represented along the main diagonal of the cube from the origin (black) to the white vertex.
- Each point along this diagonal has an equal contribution from each primary color, so that a gray shade halfway between black and white is represented as (0.5,0.5,0.5).

21.2 CMY Color Model

- A color model defined with the primary colors cyan, magenta and yellow is useful for describing color output to hard-copy devices.
- These devices produce a color picture by coating a paper with color pigments. We see the colors by reflected light, a subtractive process.



- In the CMY model, point (1,1,1) represents black, because all components of the incident light are subtracted.

- The origin represents white light. Equal amounts of each of the primary colors produce grays, along the main diagonal of the cube.
- A combination of cyan and magenta ink produces blue light, because the red and green components of the incident light are absorbed.
- Other colors are obtained by a similar subtractive process.
  We can express the conversion from RGB representation to a CMY with the matrix transformation

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

- Where the white is represented in the RGB system as the unit column vector.
- We convert from a CMY color represent to an RGB by subtracting it with the unit column vector where black is represented in the CMY system as the unit column vector.

201906100110112 Shreyanshu Vyas
201906100110052 Kushal Gaywala