Computer Graphics

18 June 2023 06:50

Line Drawing Algorithms:

DDA stands for Digital Differential Analyzer. It is an incremental method of scan conversion of line. In this method calculation is performed at each step but by using results of previous steps.

Advantage:

1. It is a faster method than method of using direct use of line equation.
2. This method does not use multiplication theorem.
3. It allows us to detect the change in the value of x and y ,so plotting of same point twice is not possible.
4. This method gives overflow indication when a point is repositioned.
5. It is an easy method because each step involves just two additions.

Disadvantage:

1. It involves floating point additions rounding off is done. Accumulations of round off error cause accumulation of error.
2. Rounding off operations and floating point operations consumes a lot of time.
3. It is more suitable for generating line using the software. But it is less suited for hardware implementation.

DDA Algorithm:

Step1: Start Algorithm
Step2: Declare x_1,y_1,x_2,y_2,dx,dy,x,y as integer variables.
Step3: Enter value of x_1,y_1,x_2,y_2.
Step4: Calculate dx = x_2-x_1
Step5: Calculate dy = y_2-y_1
Step6: If ABS (dx) > ABS (dy)
Then step = abs (dx)
Else
Step7: x_{inc}=dx/step
y_{inc}=dy/step
assign x = x_1
assign y = y_1
Step8: Set pixel (x, y)
Step9: x = x + x_{inc}
y = y + y_{inc}
Set pixels (Round (x), Round (y))
Step10: Repeat step 9 until x = x_2
Step11: End Algorithm
>

① find slope $\Rightarrow \frac{\Delta y}{\Delta x} = m$

② if $|\Delta x| \geq |\Delta y|$
assign $|\Delta x| = 1$ (max inc.)
$x_{i+1} = x_i + 1$
$y_{i+1} = y_i + m\Delta x$
$= y_i + m$

if $|\Delta y| > |\Delta x|$
assign $|\Delta y| = 1$ (max inc.)
$y_{i+1} = y_i + 1$
$x_{i+1} = x_i + \Delta y/m$
$= x_i + 1/m$

Bresenham's Line Algorithm

This algorithm is used for scan converting a line. It was developed by Bresenham. It is an efficient method because it involves only integer addition, subtractions, and multiplication
operations. These operations can be performed very rapidly so lines can be generated quickly.
In this method, next pixel selected is that one who has the least distance from true line.

The method works as follows:

Assume a pixel $P_1'(x_1',y_1')$,then select subsequent pixels as we work our may to the night, one pixel position at a time in the horizontal direction toward $P_2'(x_2',y_2')$.

Once a pixel in choose at any step
The next pixel is

1. Either the one to its right (lower-bound for the line)
2. One top its right and up (upper-bound for the line)

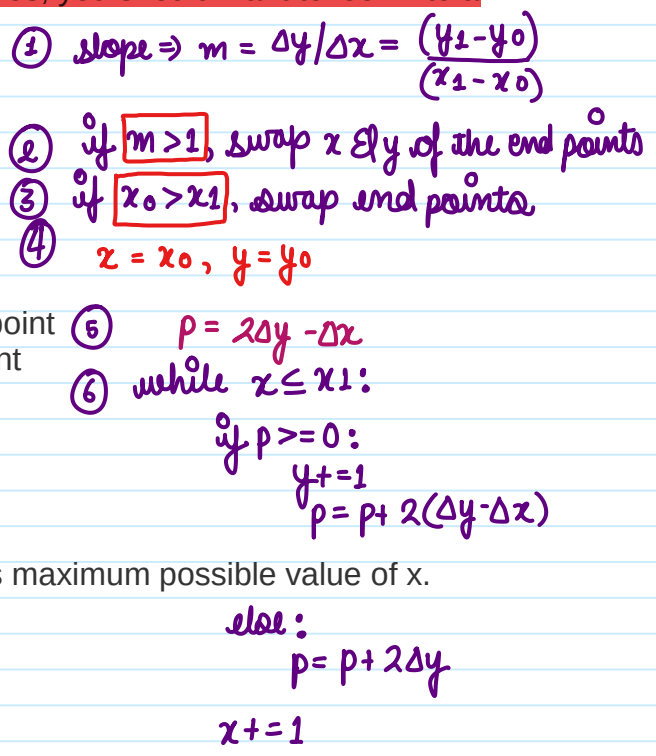
Advantage:

1. It involves only integer arithmetic, so it is simple.
2. It avoids the generation of duplicate points.
3. It can be implemented using hardware because it does not use multiplication and division.
4. It is faster as compared to DDA (Digital Differential Analyzer) because it does not involve floating point calculations like DDA Algorithm.

Disadvantage:

1. This algorithm is meant for basic line drawing only Initializing is not a part of Bresenham's line algorithm. So to draw smooth lines, you should want to look into a different algorithm.

Bresenham's Line Algorithm:



Step1: Start Algorithm
Step2: Declare variable $x_1,x_2,y_1,y_2,d,i_1,i_2,dx,dy$
Step3: Enter value of x_1,y_1,x_2,y_2
Where $x_1,y_1$are coordinates of starting point
And x_2,y_2 are coordinates of Ending point
Step4: Calculate $dx = x_2-x_1$
Calculate $dy = y_2-y_1$
Calculate $i_1=2*dy$
Calculate $i_2=2*(dy-dx)$
Calculate $d=i_1-dx$
Step5: Consider (x, y) as starting point and x_{end}as maximum possible value of x.
If $dx < 0$
Then $x = x_2$
$y = y_2$
$x_{end}=x_1$
If $dx > 0$
Then $x = x_1$
$y = y_1$
$x_{end}=x_2$
Step6: Generate point at (x,y)coordinates.
Step7: Check if whole line is generated.
If $x > = x_{end}$
Stop.

Step8: Calculate co-ordinates of the next pixel

If $d < 0$

Then $d = d + i_1$

If $d \geq 0$

Then $d = d + i_2$

Increment $y = y + 1$

Step9: Increment $x = x + 1$

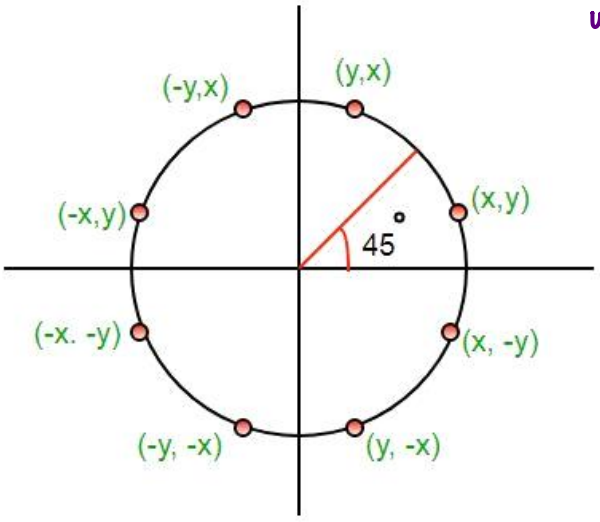
Step10: Draw a point of latest (x, y) coordinates

Step11: Go to step 7

Step12: End of Algorithm

Mid-Point Circle Drawing Algorithm

- The **mid-point** circle drawing algorithm is an algorithm used to determine the points needed for rasterizing a circle.
- We use the **mid-point** algorithm to calculate all the perimeter points of the circle in the **first octant** and then print them along with their mirror points in the other octants. This will work because a circle is symmetric about its centre.




```
x=0, y=r
p=1-r
while x≤y:
    #8dimensional geometry
    x+=1
    if p≤0:
        p=p+2x+3
    else:
        y=y-1
        p=p+2(x-y)+5
```

The algorithm is very similar to the Mid-Point Line Generation Algorithm. Here, only the boundary condition is different.

For any given pixel (x, y), the next pixel to be plotted is either **(x, y+1)** or **(x-1, y+1)**. This can be decided by following the steps below.

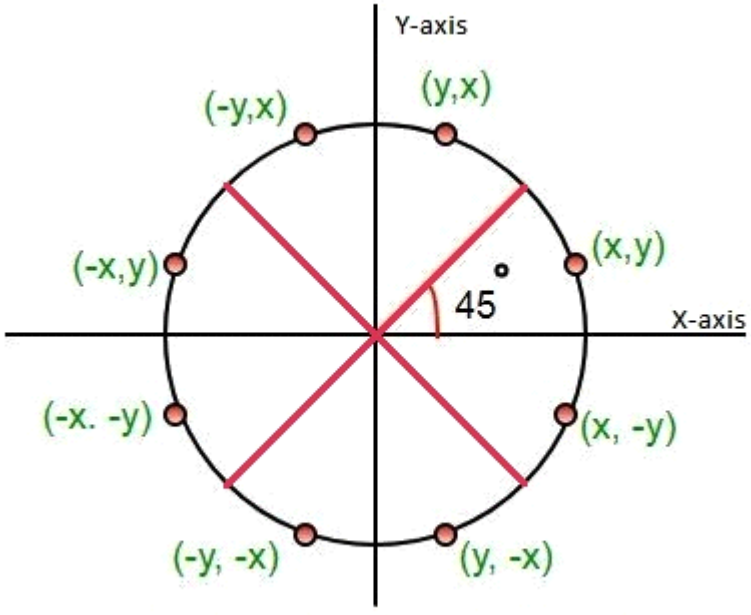
1. Find the mid-point **p** of the two possible pixels
2. If **p** lies inside or on the circle perimeter, we plot the pixel (x, y+1), otherwise if it's outside we plot the pixel (x-1, y+1)

Algorithm:

Step1: Put x =0, y =r in equation 2
We have p=1-r
Step2: Repeat steps while x ≤ y
Plot (x, y)
If (p<0)
Then set p = p + 2x + 3
Else
p = p + 2(x-y)+5
y =y - 1 (end if)
x =x+1 (end loop)
Step3: End

Bresenham's Circle Algorithm:

As per Eight way symmetry property of circle, circle can be divided into **8 octants** each of **45-degrees**.
The Algorithm calculate the location of pixels in the first octant of 45 degrees and extends it to the other 7 octants. For every pixel (x, y), the algorithm draw a pixel in each of the 8 octants of the circle as shown below :
Assumption : Center of Cirle is Origin.
Following image illustrates the 8 octants with corresponding pixel:

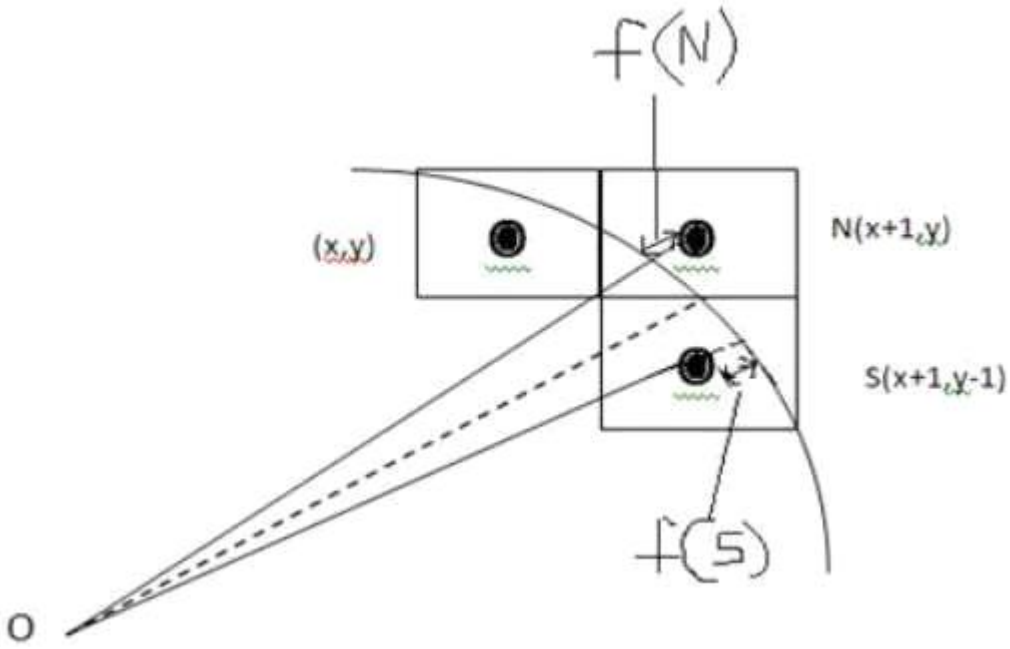


For each pixel (x,y) all possible pixels in 8 octants

$x_0 = 0, y_0 = r$
$p = 3 - 2r$
while $x \le y$:
$x += 1$
if $p \le 0$:
$p = p + 4x + 6$
else :
$y = y - 1$
$p = p + 4(x - y) + 10$

Point (x, y) is in the first octant and is on the circle.
To calculate the next pixel location, can be either:

- N (x+1, y)
- S (x+1, y-1)



It is decided by using the decision parameter d as:

- If **d <= 0**, then **N(x+1, y)** is to be chosen as next pixel.
- If **d > 0**, then **S(x+1, y-1)** is to be chosen as the next pixel.

Algorithm:

Step1: Start Algorithm
Step2: Declare p, q, x, y, r, d variables
p, q are coordinates of the center of the circle
r is the radius of the circle
Step3: Enter the value of r
Step4: Calculate d = 3 - 2r
Step5: Initialize x=0
&nbsy= r
Step6: Check if the whole circle is scan converted
If x > = y
Stop
Step7: Plot eight points by using concepts of eight-way symmetry. The center is at (p, q). Current active pixel is (x, y).
putpixel (x+p, y+q)
putpixel (y+p, x+q)
putpixel (-y+p, x+q)
putpixel (-x+p, y+q)
putpixel (-x+p, -y+q)
putpixel (-y+p, -x+q)
putpixel (y+p, -x+q)
putpixel (x+p, -y-q)
Step8: Find location of next pixels to be scanned
If d < 0
then d = d + 4x + 6
increment x = x + 1
If d ≥ 0
then d = d + 4 (x - y) + 10
increment x = x + 1
decrement y = y - 1
Step9: Go to step 6

Step10: Stop Algorithm

Clipping

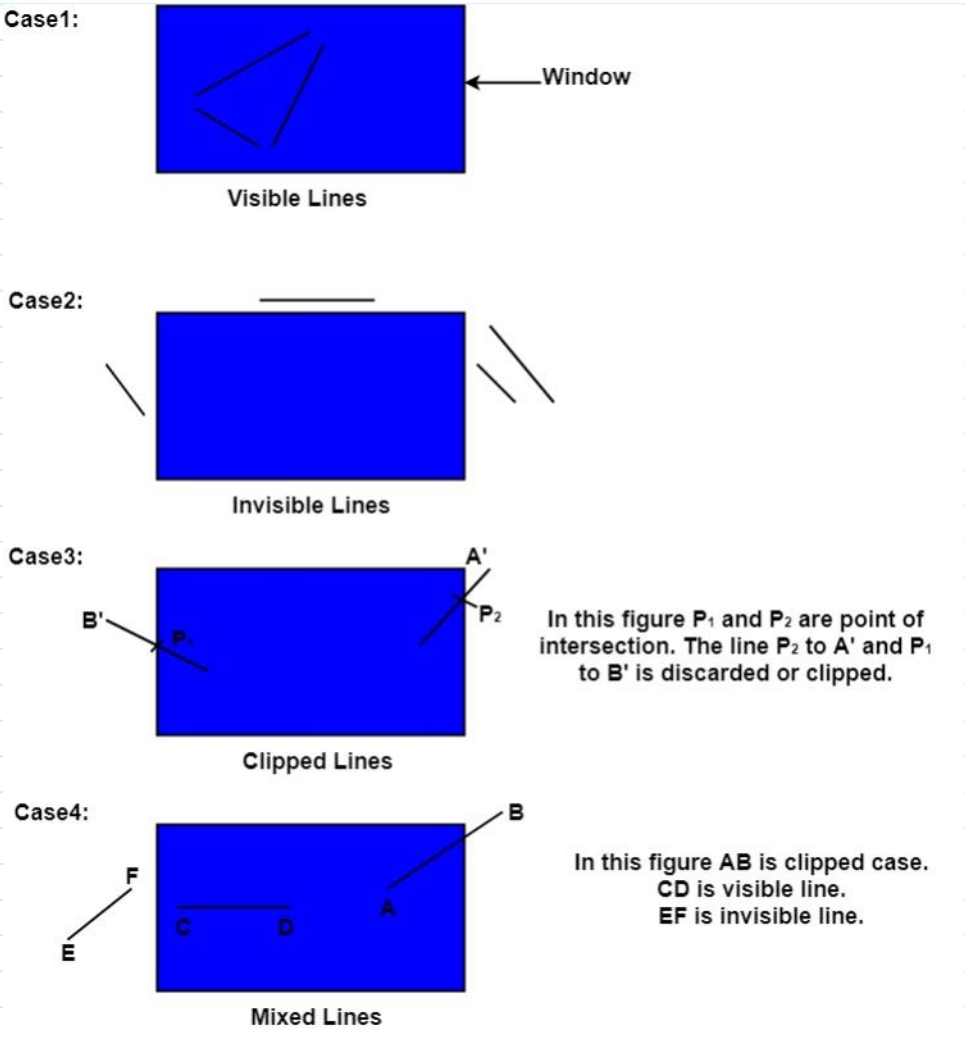
1. **Purpose:** Clipping is performed to determine which parts of a graphical object should be rendered or displayed on the screen or within a specific viewing region.
2. **Types of Clipping:** There are different types of clipping techniques based on the graphical entity being clipped, such as point/line clipping and polygon clipping.
3. **Point/Line Clipping:** It deals with determining whether points or lines extend beyond the viewport or viewing window and discarding the portions that are outside. Cohen-Sutherland and Liang-Barsky algorithms are commonly used for this purpose.
4. **Polygon Clipping:** It involves determining which parts of a polygon lie within the view frustum or viewing region. The Sutherland-Hodgman and Weiler-Atherton algorithms are commonly used for polygon clipping.
5. **Clipping in 3D:** In three-dimensional graphics, clipping becomes more complex. Objects outside the view frustum, which is a pyramid-like volume, need to be clipped. Calculations in three dimensions are performed to determine visibility and clip objects accordingly.
6. **Efficiency and Performance:** Clipping improves rendering efficiency by discarding portions of objects that are outside the view frustum, reducing unnecessary calculations and rendering time.
7. **Visual Fidelity:** Clipping ensures that only the visible portions of objects are rendered, resulting in improved visual fidelity by avoiding rendering of invisible or occluded parts.
8. **Integration in Rendering Pipeline:** Clipping is an essential step in the rendering pipeline of computer graphics, typically performed after geometry transformation and before rasterization.
9. **Hardware Acceleration:** Graphics hardware often includes specialized units, such as the clipping unit or fixed-function hardware, to accelerate the clipping process for improved performance.
10. **Importance in Virtual Reality and Augmented Reality:** Clipping plays a crucial role in rendering virtual objects in augmented reality and virtual reality applications, where objects outside the user's field of view need to be clipped to maintain immersion and optimize rendering resources.

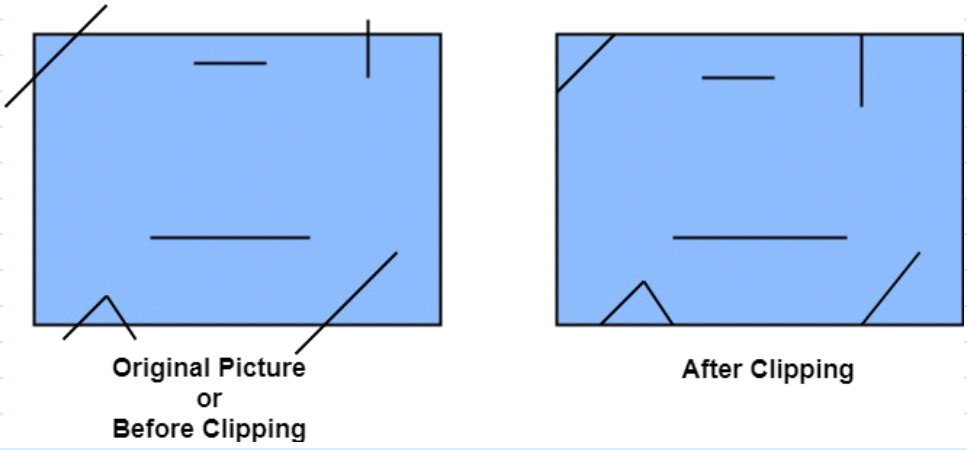
Types of Lines:

Lines are of three types:

1. **Visible:** A line or lines entirely inside the window is considered visible

2. **Invisible:** A line entirely outside the window is considered invisible
3. **Clipped:** A line partially inside the window and partially outside is clipped. For clipping point of intersection of a line with the window is determined.





Applications of clipping:

1. It will extract part we desire.
2. For identifying the visible and invisible area in the 3D object.
3. For creating objects using solid modeling.
4. For drawing operations.
5. Operations related to the pointing of an object.
6. For deleting, copying, moving part of an object.

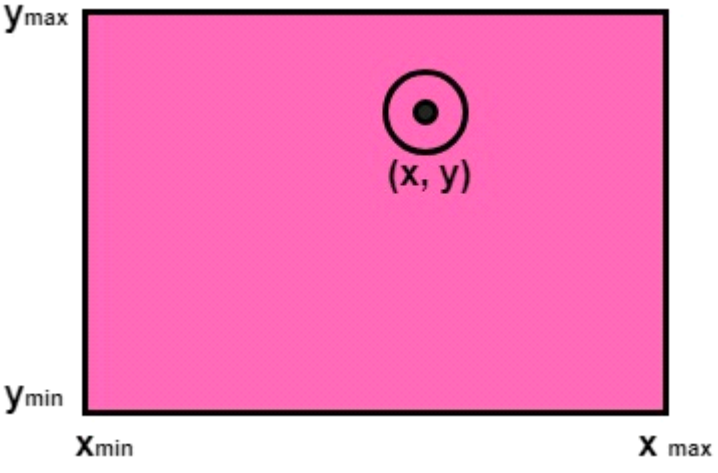
Types of Clipping:

1. Point Clipping
2. Line Clipping
3. Area Clipping (Polygon)
4. Curve Clipping
5. Text Clipping
6. Exterior Clipping

Point Clipping:

Point Clipping is used to determining, whether the point is inside the window or not. For this following conditions are checked.

1. $x \leq x_{max}$
2. $x \geq x_{min}$
3. $y \leq y_{max}$
4. $y \geq y_{min}$



The (x, y) is coordinate of the point. If anyone from the above inequalities is false, then

the point will fall outside the window and will not be considered to be visible.

Line Clipping:

- Line clipping in computer graphics determines which parts of a line segment should be displayed within a specified viewing region.
- It involves selectively discarding the portions of the line segment that lie outside the clipping region.
- Line clipping is essential for maintaining visual fidelity, especially when rendering lines that extend beyond the viewport.
- By clipping the line to the visible portion, unnecessary computations and rendering operations can be avoided, resulting in improved efficiency.
- Two commonly used line clipping algorithms are the **Cohen-Sutherland algorithm** and the **Liang-Barsky algorithm.**

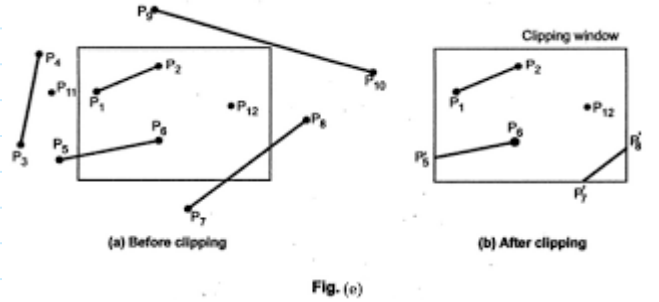


Fig. (e)

Cohen Sutherland Line Clipping Algorithm:

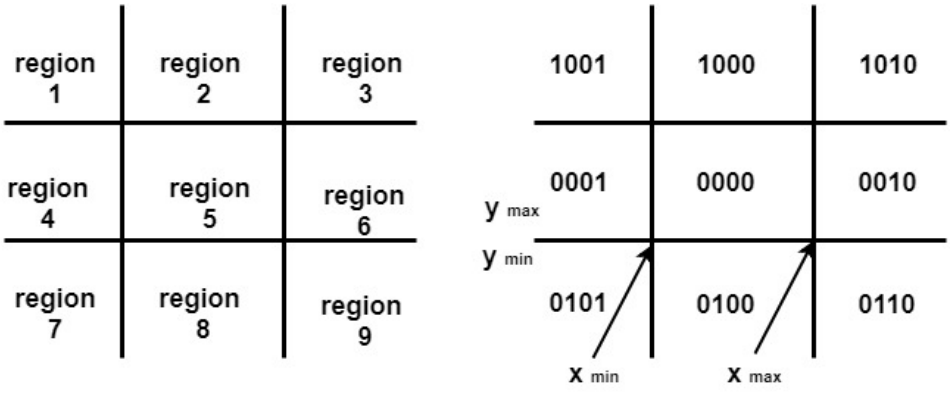
In the algorithm, first of all, it is detected whether line lies inside the screen or it is outside the screen. All lines come under any one of the following categories:

1. Visible
2. Not Visible
3. Clipping Case

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

2. Not Visible: If a line lies outside the window it will be invisible and rejected. Such lines will not display. If any one of the following inequalities is satisfied, then the line is considered invisible

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

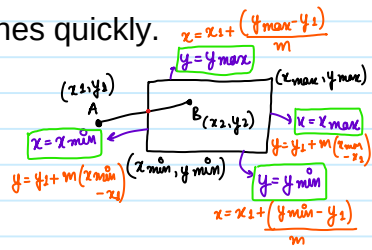


9 region bits assigned to 9 regions

heck binary digits:- TBRL which can be defined as top, bottom, right, and left accordingly.

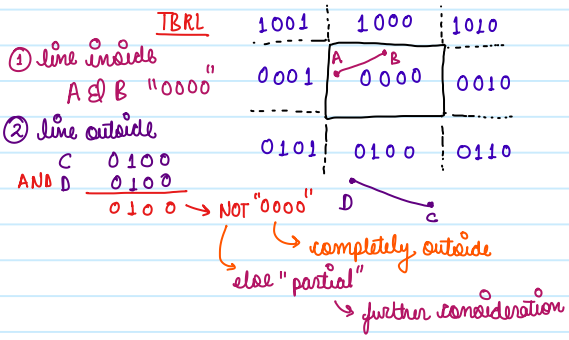
Advantage of Cohen Sutherland Line Clipping:

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



The region code for point (x, y) is set according to the scheme

Bit 1 = $(y-y_{max})$
Bit 2 = $(y_{min}-y)$
Bit 3 = $(x-x_{max})$
Bit 4 = $(x_{min}-x)$



Algorithm

1) Assign the region codes to both endpoints.

2) Perform OR operation on both of these endpoints.

3) if OR = 0000,

then it is completely visible (inside the window).

else

Perform AND operation on both these endpoints.

i) if AND ≠ 0000,

then the line is invisible and not inside the window. Also, it **can't be** considered for clipping.

ii) else

AND = 0000, the line is partially inside the window and **considered for clipping.**

4) After confirming that the **line is partially inside** the window, then we find the intersection with the boundary of the window. By using the following formula:-

Slope:- m= (y2-y1)/(x2-x1)

a) If the line passes through **top** or the line intersects with the top boundary of the window.

x = x + (ymax – y)/m

y = ymax

b) If the line passes through the **bottom** or the line intersects with the bottom boundary of the window.

x = x + (ymin – y)/m

y = ymin

c) If the line passes through the **left** region or the line intersects with the left boundary of the window.

y = y+ (xmin – x)*m

x = xmin

d) If the line passes through the **right** region or the line intersects with the right boundary of the window.

y = y + (xmax -x)*m

x = xmax

5) Now, overwrite the endpoints with a new one and update it.

6) Repeat the 4th step till your line doesn't get completely clipped

Liang-Barsky Line Clipping Algorithm:

- Liang and Barsky have established an algorithm that uses floating-point arithmetic but finds the appropriate endpoints with at most four computations.
- This algorithm uses the parametric equations for a line and solves four inequalities to find the range of the parameter for which the line is in the viewport.

Liang-Barsky's Line Clipping Algorithm

$$X = x1 + \Delta x \cdot u$$

$$Y = y1 + \Delta y \cdot u$$

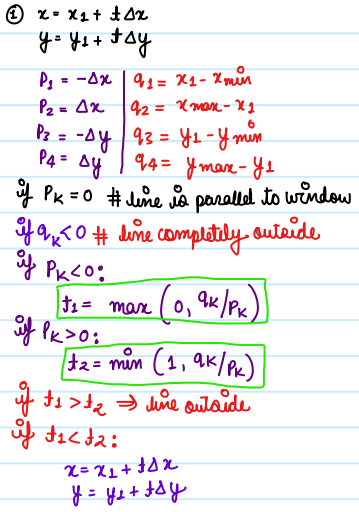
The parametric eqn of line between (x1,y1) and (x2,y2) is:

$$Xmin <= x1 + \Delta x \cdot u <= Xmax$$

$$Ymin <= y1 + \Delta y \cdot u <= Ymax$$

----------(1)

Where, $\Delta x = x2 - x1$, $\Delta y = y2 - y1$, $0 <= u <= 1$ For a point (x,y) inside the clipping window



$P_k u <= q_k$, k=1,2,3,4

----------(2)

Rewriting the four inequalities of eqn (2)

Where ,

p1= -Δx, q1=x1-Xmin (left)
p2= Δx, q2=Xmax- X1 (right)
p3= -Δy, q3=y1-Ymin (bottom)
p4= Δy, q4=Ymax- Y1 (Top)

Case 1: If Pk =0 for all I => both ends points are same => it is a point.

Case 2: If **Pk =0** for any two i=> Line is parallel to the window boundary
And if **qk <0** => the line is completely outside the window boundary

Else if **qk>=0**=> line is inside the window boundaries

Case 3: Pk <0 => Line is potentially Entering (PE)

Case4: Pk>0 => Line is potentially Leaving (PL)

When Pk != 0, the intersection points can be calculated using the values of u that can further be calculated using the formula rk=qk/pk

Algorithm

Step1:

If Pk=0 and qk <0 for some i (invisible, ignore)

Step2:

If Pk=0 and qk >0 (Completely visible, save)

Step3:

∀ k s.t. Pk <0, rk=qk/pk U1=max(0, rk)

Step4:

∀ k s.t. Pk >0, rk=qk/pk U2=min(1, rk)

Step 5:

If u1 >= u2 => invisible else use u1 and u2 and eqn 1 for calculating the end points.

Q1. Use Liang-Barsky's line Clipping algorithm to Clip the line P1P2 given by the coordinates P1 (0, 2) and P2 (5, 7) w.r.t the clipping window whose Principal diagonal coordinates are (1, 1) and (4, 4) respectively.

Q2. Use Liang-Barsky's line Clipping algorithm to Clip the line P1P2 given by the coordinates P1 (0, 5)

and P2 (15, 5) against the clipping window whose principal diagonal coordinates are (2, 3) and (10, 9) respectively.

Q2. Use Liang-Barsky's line Clipping algorithm to Clip the following lines against the clipping window whose principal diagonal coordinates are (2, 2) and (6, 6) respectively.

i. **P1 (3, 5) and P2 (3, 9)**

ii. **P1 (1, 8) and P2 (1, 10)**

iii. **P1(1,2) and p2(7,7)**

Polygon:

Polygon is a representation of the surface. It is primitive which is closed in nature. It is formed using a collection of lines. It is also called as many-sided figure. The lines combined to form polygon are called sides or edges. The lines are obtained by combining two vertices.

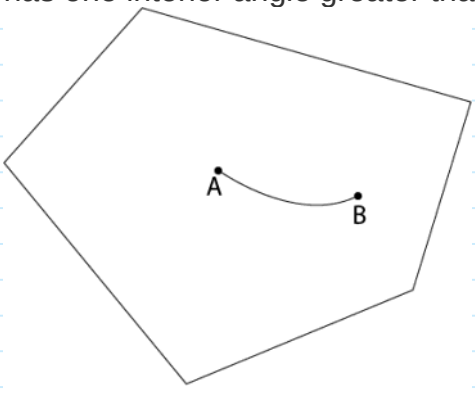
Example of Polygon:

1. Triangle
2. Rectangle
3. Hexagon
4. Pentagon

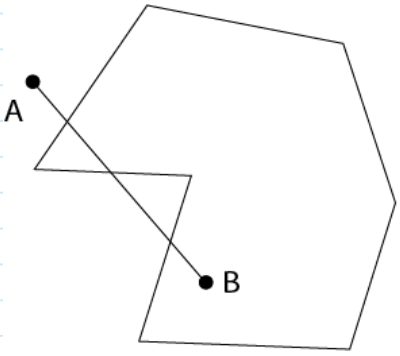
Types of Polygons

1. Concave
2. Convex

A polygon is called convex of line joining any two interior points of the polygon lies inside the polygon. A non-convex polygon is said to be concave. A concave polygon has one interior angle greater than 180°. So that it can be clipped into similar polygons.

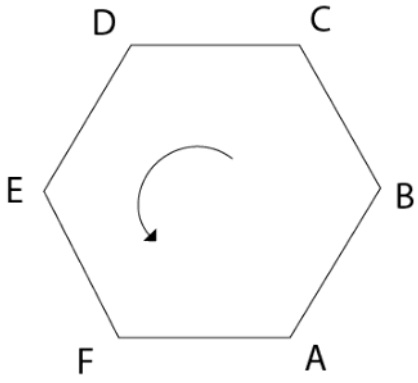


Convex polygon

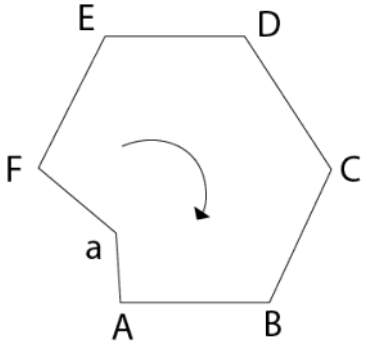


Concave polygon

A polygon can be positive or negative oriented. If we visit vertices and vertices visit produces counterclockwise circuit, then orientation is said to be positive.



Polygon with positive orientation



Polygon with negative orientation

Sutherland-Hodgeman Polygon Clipping:

- It is performed by processing the boundary of polygon against each window corner or edge.
- First of all entire polygon is clipped against one edge, then resulting polygon is considered
- then the polygon is considered against the second edge, so on for all four edges.

Four possible situations while processing

1. If the first vertex is an outside the window, the second vertex is inside the window.

Then second vertex is added to the output list. The point of intersection of window boundary and polygon side (edge) is also added to the output line.

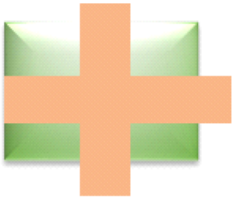
2. If both vertexes are inside window boundary. Then only second vertex is added to the output list.
3. If the first vertex is inside the window and second is an outside window. The edge which intersects with window is added to output list.
4. If both vertices are the outside window, then nothing is added to output list.

Following figures shows original polygon and clipping of polygon against four windows.

Original polygon

Clipping against left edge of window

Clipping against bottom corner of window



Clipping against Top of window

Clipping against right edge of window

Final polygon after Clipping

Disadvantage of Hodgmen Algorithm:

- This method requires a considerable amount of memory.
- The first of all polygons are stored in original form.
- Then clipping against left edge done and output is stored
- . Then clipping against right edge done, then top edge.
- Finally, the bottom edge is clipped. Results of all these operations are stored in memory.
- So wastage of memory for storing intermediate polygons.

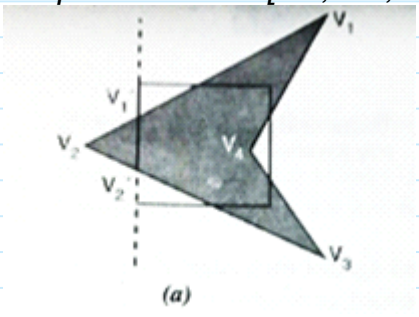
specific rules to consider for each of the four sides of the window. Here are the rules for clipping against each side:

1. **Left Side of the Window:**
   - Check each edge of the polygon against the left side of the window.
   - If an edge crosses the left side from outside to inside, compute the intersection point between the edge and the left side.
   - Add the intersection point to the list of clipped vertices.

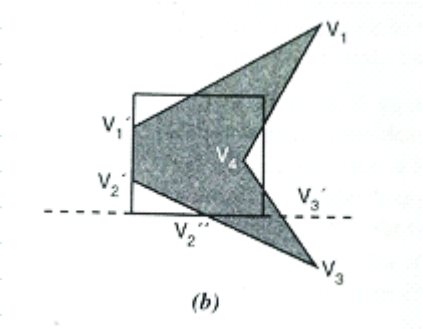
- If an edge crosses the left side from inside to outside, add the second endpoint of the edge to the list of clipped vertices.

2. **Right Side of the Window:**
   - Check each edge of the polygon against the right side of the window.
   - If an edge crosses the right side from outside to inside, compute the intersection point between the edge and the right side.
   - Add the intersection point to the list of clipped vertices.
   - If an edge crosses the right side from inside to outside, add the second endpoint of the edge to the list of clipped vertices.
3. **Bottom Side of the Window:**
   - Check each edge of the polygon against the bottom side of the window.
   - If an edge crosses the bottom side from outside to inside, compute the intersection point between the edge and the bottom side.
   - Add the intersection point to the list of clipped vertices.
   - If an edge crosses the bottom side from inside to outside, add the second endpoint of the edge to the list of clipped vertices.
4. **Top Side of the Window:**
   - Check each edge of the polygon against the top side of the window.
   - If an edge crosses the top side from outside to inside, compute the intersection point between the edge and the top side.
   - Add the intersection point to the list of clipped vertices.
   - If an edge crosses the top side from inside to outside, add the second endpoint of the edge to the list of clipped vertices.

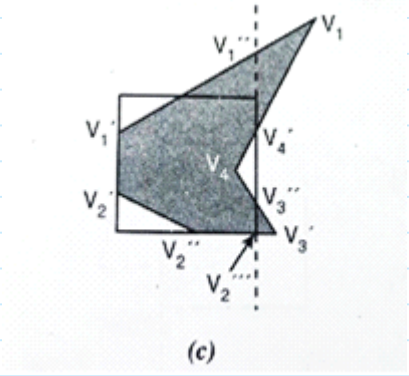
*Step1 – **Clip against Left edge***
Input vertex list [V_1, V_2, V_3, V_4]
Edge V_1-> V_2 : output V_1‘
Edge V_2-> V_3 : output V_2‘, V_3
Edge V_3-> V_4 : output V_4
Edge V_4-> V_1 : output V_1
Output vertex list [V_1‘, V_2‘, V_3, V_4, V_1]



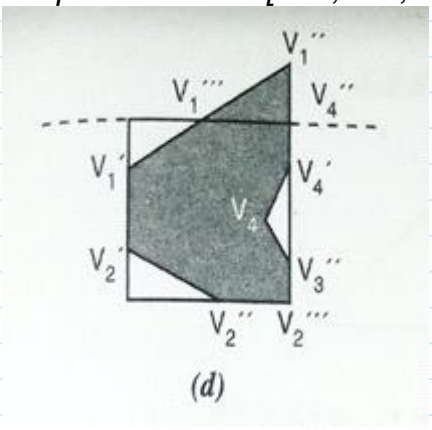
*Step2 – **Clip against Bottom edge***
Input vertex list [V_1‘, V_2‘, V_3, V_4, V_1]
Edge V_1‘-> V_2‘ : output V_2‘
Edge V_2‘-> V_3 : output V_2”
Edge V_3-> V_4 : output V_3‘, V_4
Edge V_4-> V_1 : output V_1
Edge V_1-> V_1‘ : output V_1‘
Output vertex list [V_2‘, V_2‘, V_2”, V_3‘, V_4, V_1, V_1‘]



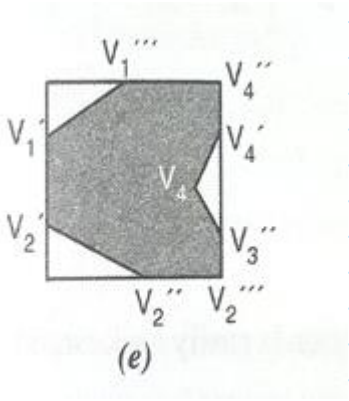
Step3 – ***Clip against Right edge***
Input vertex list [V2‘, V2”, V3‘, V4, V1, V1]
Edge V2‘-> V2” : output V2”
Edge V2”-> V3‘ : output V2”’
Edge V3‘-> V4 : output V3”, V4
Edge V4-> V1 : output V4‘
Edge V1-> V1‘ : output V1”, V1‘
Edge V1‘-> V2‘ : output V2‘
Output vertex list [V2”, V2”’, V3”, V4, V4‘, V1”, V1‘, V2‘]



Step4 – ***Clip against Top edge***
Input vertex list [V2”, V2”’, V3”, V4‘, V1”, V1‘, V2‘]
Edge V2‘-> V2”’ : output V2”’
Edge V2”’-> V3” : output V3”
Edge V3”-> V4 : output V4
Edge V4‘-> V4‘ : output V4‘
Edge V4‘-> V1” : output V4”
Edge V1”-> V1‘ : output V1”’, V1‘
Edge V1‘-> V2‘ : output V2‘
Edge V2‘-> V2” : output V2”
Output vertex list [V2”’, V3”, V4, V4‘, V4”, V1”’, V1‘, V2‘, V2”]

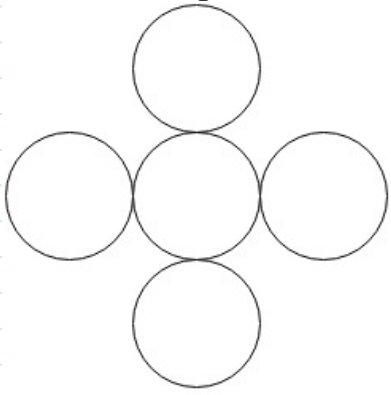


Step5 – ***The Final Clipped Polygon***
Output vertex list [V2”’, V3”, V4, V4‘, V1”’, V1‘, V2‘, V2”]

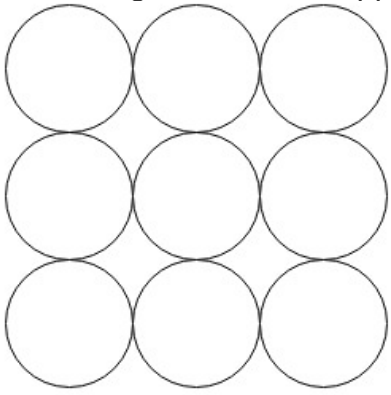


Boundary Filled Algorithm:

This algorithm uses the recursive method. First of all, a starting pixel called as the seed is considered. The algorithm checks boundary pixel or adjacent pixels are colored or not. If the adjacent pixel is already filled or colored then leave it, otherwise fill it. The filling is done using four connected or eight connected approaches.



Four Connected



Eight Connected

Four connected approaches is more suitable than the eight connected approaches.

1. Four connected approaches: In this approach, left, right, above, below pixels are tested.

2. Eight connected approaches: In this approach, left, right, above, below and four diagonals are selected.

Boundary can be checked by seeing pixels from left and right first. Then pixels are checked by seeing pixels from top to bottom. The algorithm takes time and memory because some recursive calls are needed.

Problem with recursive boundary fill algorithm:

It may not fill regions sometimes correctly when some interior pixel is already filled with color. The algorithm will check this boundary pixel for filling and will found already filled so recursive process will terminate. This may vary because of another interior pixel unfilled.

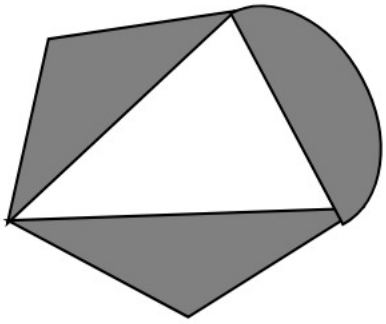
Algorithm:

```
Procedure fill (x, y, color, color1: integer)
int c;
c=getpixel (x, y);
if (c!=color) (c!=color1)
```

```
{
    setpixel (x, y, color)
    fill (x+1, y, color, color 1);
     fill (x-1, y, color, color 1);
    fill (x, y+1, color, color 1);
    fill (x, y-1, color, color 1);
}
```

Flood Fill Algorithm:

- In this method, a point or seed which is inside region is selected.
- This point is called a seed point.
- Then four connected approaches or eight connected approaches is used to fill with specified color.
- The flood fill algorithm has many characters similar to boundary fill.
- this method is more suitable for filling multiple colors boundary.
- When boundary is of many colors and interior is to be filled with one color we use this algorithm.



we start from a specified interior point (x, y) an reassign all pixel values are currently set to a given interior color with the desired color. Using either a 4-connected or 8-connected approaches, we then step through pixel positions until all interior points have been repainted.

Disadvantage:

1. Very slow algorithm
2. May be fail for large polygons
3. Initial pixel required more knowledge about surrounding pixels.

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);
```

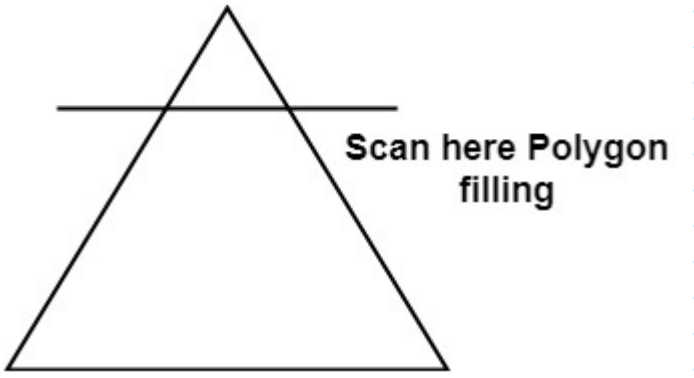
```
    }
}
```

Scan Line Polygon Fill Algorithm:

- This algorithm lines interior points of a polygon on the scan line and these points are done on or off according to requirement.
- The polygon is filled with various colors by coloring various pixels.

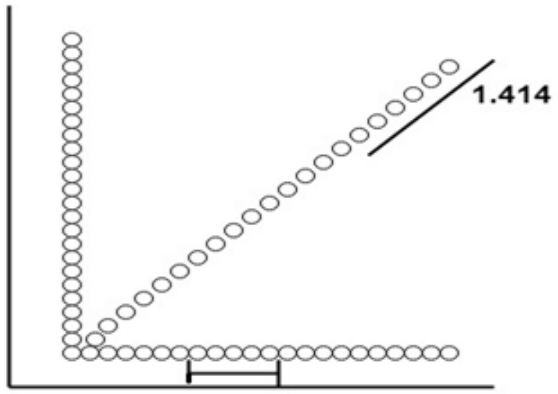
The steps for polygon clipping using the scanline algorithm:

1. **Perform scanning:** Start scanning the screen using the raster scanning concept, moving from the top left corner to the bottom right corner.
2. **Find intersection points:** While scanning, identify the points where the scanline intersects with the edges of the polygon. Determine the x-coordinate of the intersection points using linear interpolation.
3. **Store intersection points:** Store the intersection points in the frame buffer and set their intensities to a high value.
4. **Maintain coherence property:** Utilize the coherence property, which states that if a pixel is inside the polygon, the next pixel on the scanline will also be inside the polygon. This property allows for efficient processing by reducing the number of intersection calculations.



Side effects of Scan Conversion:

1. Staircase or Jagged: Staircase like appearance is seen while the scan was converting line or circle.
2. Unequal Intensity: It deals with unequal appearance of the brightness of different lines. An inclined line appears less bright as compared to the horizontal and vertical line.



Pixels along with horizontal line are 1 unit apart and vertical.
Pixels along diagonal line are 1.414 units.

Two Dimensional Transformations

refers to the process of altering the position, size, orientation, or shape of objects in a graphical scene. It involves manipulating the coordinates and attributes of graphical elements to achieve desired visual effects. Transformations are fundamental operations used in computer graphics to create, manipulate, and animate objects

There are two complementary points of view for describing object transformation.

1. Geometric Transformation: The object itself is transformed relative to the coordinate system or background. The mathematical statement of this viewpoint is defined by geometric transformations applied to each point of the object.
2. Coordinate Transformation: The object is held stationary while the coordinate system is transformed relative to the object. This effect is attained through the application of coordinate transformations.

Types of Transformations:

1. Translation
2. Scaling
3. Rotating
4. Reflection
5. Shearing

Translation

It is the straight line movement of an object from one position to another is called Translation. Here the object is positioned from one coordinate location to another.

$x_1=x+T_x$

$y_1=y+T_y$

Matrix for Translation:

$$\begin{vmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{vmatrix} \text{ Or } \begin{vmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{vmatrix}$$

Scaling:

It is used to alter or change the size of objects. The change is done using scaling factors. There are two scaling factors, i.e. S_x in x direction S_y in y-direction. If the original position is x and y. Scaling factors are S_x and S_y then the value of coordinates after scaling will be x^1 and y_1.

Matrix for Scaling:

$$S = \begin{vmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Rotation:

It is a process of changing the angle of the object. Rotation can be clockwise or anticlockwise. For rotation, we have to specify the angle of rotation and rotation point. Rotation point is also called a pivot point. It is print about which object is rotated.

Types of Rotation:

1. Anticlockwise
2. Counterclockwise

The positive value of the pivot point (rotation angle) rotates an object in a counter-clockwise (anti-clockwise) direction.
The negative value of the pivot point (rotation angle) rotates an object in a clockwise direction.
When the object is rotated, then every point of the object is rotated by the same angle.

Matrix for homogeneous co-ordinate rotation (clockwise)

$$R = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Matrix for homogeneous co-ordinate rotation (anticlockwise)

$$R = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Reflection:

It is a transformation which produces a mirror image of an object. The mirror image can be either about x-axis or y-axis. The object is rotated by180°.

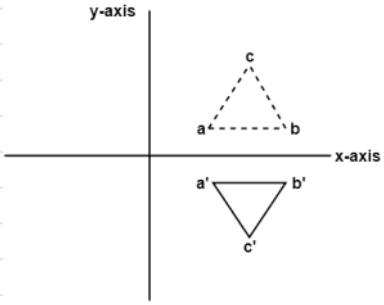
Types of Reflection:

1. Reflection about the x-axis
2. Reflection about the y-axis
3. Reflection about an axis perpendicular to xy plane and passing through the origin
4. Reflection about line y=x

1. Reflection about x-axis: The object can be reflected about x-axis with the help of

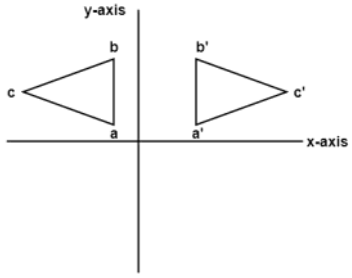
the following matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



2. Reflection about y-axis: The object can be reflected about y-axis with the help of following transformation matrix

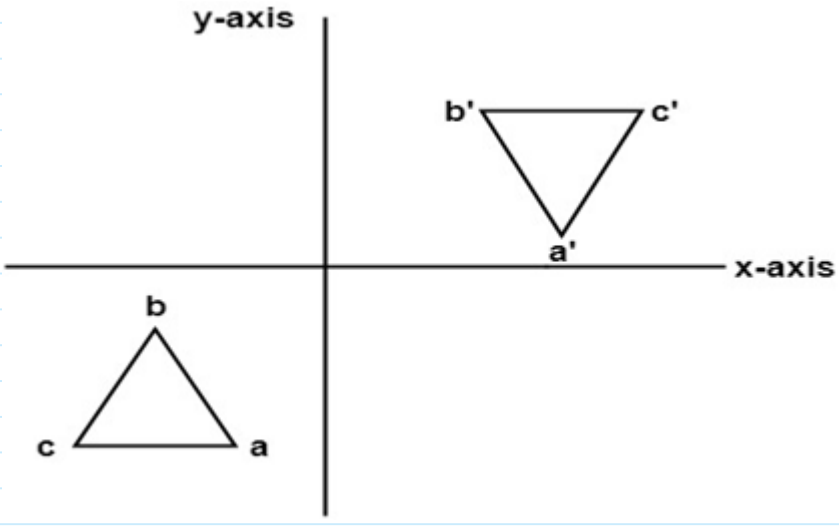
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



3. Reflection about an axis perpendicular to xy plane and passing through origin:

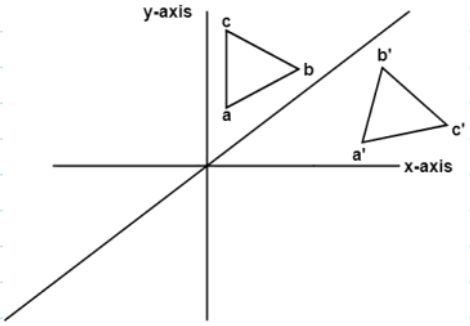
In the matrix of this transformation is given below

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



4. Reflection about line y=x: The object may be reflected about line y = x with the help of following transformation matrix

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



Shearing:

It is transformation which changes the shape of object. The sliding of layers of object occur. The shear can be in one direction or in two directions.
Shearing in the X-direction: In this horizontal shearing sliding of layers occur. The homogeneous matrix for shearing in the x-direction is shown below:

$$\begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Shearing:

It is transformation which changes the shape of object. The sliding of layers of object occur. The shear can be in one direction or in two directions.
Shearing in the X-direction: In this horizontal shearing sliding of layers occur. The homogeneous matrix for shearing in the x-direction is shown below:

$$\begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

1. Translation
2. Rotation
3. Reverse Translation

Advantage of composition or concatenation of matrix:

1. It transformations become compact.
2. The number of operations will be reduced.
3. Rules used for defining transformation in form of equations are complex as compared to matrix.

Three Dimensional Transformations

The geometric transformations play a vital role in generating images of three Dimensional objects with the help of these transformations. The location of objects relative to others can be easily expressed. Sometimes viewpoint changes rapidly, or sometimes objects move in relation to each other. For this number of transformation can be carried out repeatedly.

Translation

It is the movement of an object from one position to another position. Translation is done using translation vectors. There are three vectors in 3D instead of two. These vectors are in x, y, and z directions. Translation in the x-direction is represented using T_x. The translation is y-direction is represented using T_y. The translation in the z-direction is represented using T_z.

$$\begin{pmatrix} x^1 \\ y^1 \\ z^1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Scaling

Scaling is used to change the size of an object. The size can be increased or decreased. The scaling three factors are required S_x S_y and S_z.
S_x=Scaling factor in x- direction
S_y=Scaling factor in y-direction
S_z=Scaling factor in z-direction

Matrix for Scaling

$$\begin{Bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

Scaling of the object relative to a fixed point

Following are steps performed when scaling of objects with fixed point (a, b, c). It can be represented as below:

1. Translate fixed point to the origin
2. Scale the object relative to the origin
3. Translate object back to its original position.

Rotation about Arbitrary Axis

When the object is rotated about an axis that is not parallel to any one of co-ordinate axis, i.e., x, y, z. Then additional transformations are required. First of all, alignment is needed, and then the object is being back to the original position. Following steps are required

1. Translate the object to the origin
2. Rotate object so that axis of object coincide with any of coordinate axis.
3. Perform rotation about co-ordinate axis with whom coinciding is done.
4. Apply inverse rotation to bring rotation back to the original position.

Matrix for representing three-dimensional rotations about the Z axis

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Matrix for representing three-dimensional rotations about the X axis

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

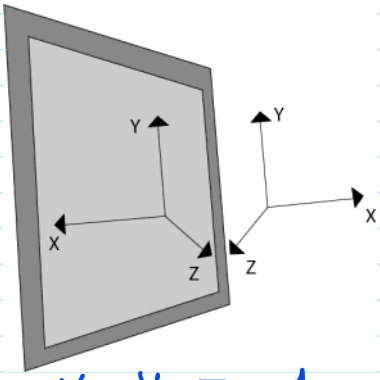
Matrix for representing three-dimensional rotations about the Y axis

$$\begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Reflection

It is also called a mirror image of an object. For this reflection axis and reflection of plane is selected. Three-dimensional reflections are similar to two dimensions. Reflection is 180° about the given axis. For reflection, plane is selected (xy,xz or yz). Following matrices show reflection respect to all these three planes.

Reflection relative to XY plane



XY plane

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

YZ plane

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ZX plane

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

	X	Y	Z	1
X	1	0	0	0
Y	0	1	0	0
Z	0	0	-1	0
1	0	0	0	1

Reflection relative to YZ plane

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Reflection relative to ZX plane

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Shearing

It is change in the shape of the object. It is also called as deformation. Change can be in the x -direction or y -direction or both directions in case of 2D. If shear occurs in both directions, the object will be distorted. But in 3D shear can occur in three directions.

Matrix for shear

$$\begin{pmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Hidden Surface Removal

remove the hidden parts using geometric sorting

- Hidden Surface Removal is the process of removing the hidden parts of solid objects in computer graphics.
- In real life, opaque materials block light rays from hidden parts, but in computer graphics, all parts of objects are displayed.
- Hidden surface algorithms distinguish visible parts from hidden parts using geometric sorting.
- Geometric sorting locates objects near the observer to determine visibility.
- Hidden line and hidden surface algorithms use different forms of coherence to reduce computation.
- Coherence types include scan line coherence, frame coherence, and object coherence.
- Hidden surface algorithms exploit coherence properties to increase efficiency.
- Hidden surface removal is similar to two-dimensional scan conversions.

Types of hidden surface detection algorithms

1. Object space methods
2. Image space methods

Object Space Methods:

- Compares various parts of objects to determine visible, invisible, or hardly visible surfaces.
- Primarily used for wireframe models to determine visible lines.
- Line-based algorithms that determine obstructed parts of an object and draw them in the same color.

Image Space Methods:

- Determines the positions of pixels to locate visible surfaces.
- Used to determine visible surfaces instead of visible lines.
- Each pixel is evaluated for visibility, and if visible, the pixel is drawn in the appropriate color.
- Also known as Visible Surface Determination.

General Information:

- Both methods are used for visible surface determination in computer graphics.
- Object space methods compare parts of objects, while image space methods analyze the visibility of individual pixels.
- Implementation of these methods requires significant processing time and computational power.
- Image space methods involve more computations as each object's visibility and surface determination are considered.

Object Space	Image Space
1. Image space is object based. It concentrates on geometrical relation among objects in the scene.	1. It is a pixel-based method. It is concerned with the final image, what is visible within each raster pixel.
2. Here surface visibility is determined.	2. Here line visibility or point visibility is determined.
3. It is performed at the precision with	3. It is performed using the

which each object is defined, No resolution is considered.	resolution of the display device.
4. Calculations are not based on the resolution of the display so change of object can be easily adjusted.	4. Calculations are resolution base, so the change is difficult to adjust.
5. These were developed for vector graphics system.	5. These are developed for raster devices.
6. Object-based algorithms operate on continuous object data.	6. These operate on object data.
7. Vector display used for object method has large address space.	7. Raster systems used for image space methods have limited address space.
8. Object precision is used for application where speed is required.	8. There are suitable for application where accuracy is required.
9. It requires a lot of calculations if the image is to enlarge.	9. Image can be enlarged without losing accuracy.
10. If the number of objects in the scene increases, computation time also increases.	10. In this method complexity increase with the complexity of visible parts.

Considerations for selecting or designing hidden surface algorithms:

Following three considerations are taken:

1. Sorting
2. Coherence
3. Machine

Sorting: All surfaces are sorted in two classes, i.e., visible and invisible. Pixels are colored accordingly. Several sorting algorithms are available i.e.

1. Bubble sort
2. Shell sort
3. Quick sort
4. Tree sort
5. Radix sort

Different sorting algorithms are applied to different hidden surface algorithms. Sorting of objects is done using x and y, z co-ordinates. Mostly z coordinate is used for sorting. The efficiency of sorting algorithm affects the hidden surface removal algorithm. For sorting complex scenes or hundreds of polygons complex sorts are used, i.e., quick sort, tree sort, radix sort.

Coherence

It is used to take advantage of the constant value of the surface of the scene. It is based on how much regularity exists in the scene. When we moved from one polygon of one object to another polygon of same object color and shearing will remain unchanged.

Types of Coherence

1. Edge coherence
2. Object coherence
3. Face coherence
4. Area coherence
5. Depth coherence
6. Scan line coherence
7. Frame coherence
8. Implied edge coherence

Z-Buffer Algorithm

The Z-Buffer algorithm, also known as the depth buffer algorithm, is a widely used method for hidden surface removal in computer graphics. It operates on the principle of comparing the depths (Z-values) of objects or polygons at each pixel location to determine visibility. Here are the key points about the Z-Buffer algorithm:

1. **Depth Buffer:** The Z-Buffer algorithm uses a dedicated buffer called the depth buffer or Z-buffer, which is an array with the same dimensions as the frame buffer. Each element in the depth buffer stores the depth value (Z-value) of the closest visible object at that pixel location.

2. **Initialization:** Before rendering begins, the depth buffer is initialized with a maximum depth value (e.g., positive infinity) indicating that no objects have been rendered yet.

3. **Depth Test:** As each polygon or fragment is processed during rendering, the algorithm performs a depth test. The depth test compares the Z-value of the current polygon or fragment with the Z-value stored in the corresponding depth buffer location. If the current Z-value is closer to the viewer, it replaces the value in the depth buffer and gets rendered.

4. **Updating the Frame Buffer:** Simultaneously with the depth test, the algorithm updates the frame buffer with the color of the closest visible object or fragment.

5. **Visibility Determination:** The Z-Buffer algorithm automatically handles complex scenes with overlapping objects and determines the visibility of each pixel based on the Z-values. Objects that are closer to the viewer will have their Z-values updated in the depth buffer, ensuring that they appear correctly.

6. **Efficient and Automatic:** The Z-Buffer algorithm is efficient because it only requires a single pass over the scene for depth testing and frame buffer updating. It also handles visibility automatically without the need for explicit sorting or explicit visibility determination.

Limitations of Depth Buffer

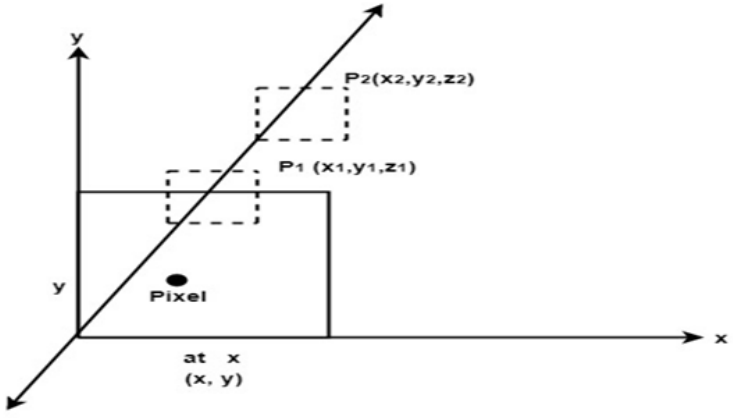
1. The depth buffer Algorithm is not always practical because of the enormous size of depth and intensity arrays.
2. Generating an image with a raster of 500 x 500 pixels requires 2, 50,000 storage locations for each array.
3. Even though the frame buffer may provide memory for intensity array, the depth array remains large.
4. To reduce the amount of storage required, the image can be divided into many smaller images, and the depth buffer algorithm is applied to each in turn.
5. For example, the original 500 x 500 faster can be divided into 100 rasters each 50 x 50 pixels.
6. Processing each small raster requires array of only 2500 elements, but execution time grows because each polygon is processed many times.
7. Subdivision of the screen does not always increase execution time instead it can help reduce the work required to generate the image. This reduction arises because of coherence between small regions of the screen.

Algorithm

1. Initialize depth and intensity values for all pixels to background values.
2. For each polygon:

- Find pixels within the polygon boundaries on the screen.
- Calculate the depth of the polygon at each pixel.
- If the polygon is closer to the observer (z < depth), update depth and intensity values.

3. After processing all polygons, the intensity array contains the solution.
4. The algorithm requires a representation of opaque surfaces in the scene, typically as polygons.



Animation → movement on screen

- Animation refers to the movement on the screen of the display device created by displaying a sequence of still images.

- Animation is the technique of designing, drawing, making layouts and preparation of photographic series which are integrated into the multimedia and gaming products.

- Animation connects the exploitation and management of still images to generate the illusion of movement.
- A person who creates animations is called animator. He/she use various computer technologies to capture the pictures and then to animate these in the desired sequence.

key principles of animation

Here are the key principles of animation in computer graphics, summarized in points:

1. **Squash and Stretch:** This principle adds exaggeration and flexibility to objects, giving them a more dynamic and organic feel. It is used to convey weight, impact, and elasticity.

2. **Timing and Spacing:** The timing and spacing of movements play a crucial role in creating believable and appealing animations. It involves controlling the speed and rhythm of movements to convey the desired effect.

3. **Anticipation:** Anticipation is the preparation for a major action or movement. It helps to make actions feel more realistic by providing a visual cue to the audience about what is going to happen next.

4. **Staging:** Staging refers to the presentation and arrangement of objects and characters within a scene. It involves framing the animation in a way that directs the viewer's attention to the most important elements.

5. **Follow-through and Overlapping Action:** Follow-through involves the continuation of movement after the main action is completed. Overlapping action refers to different parts of an object or character moving at different rates, adding to the overall fluidity and naturalness of the animation.

6. **Slow-in and Slow-out:** Objects in the real world rarely start or stop abruptly. The principle of slow-in and slow-out adds more realism by gradually accelerating or decelerating the movement at the beginning and end.

7. **Arcs:** Most natural movements follow an arc or a curved path rather than a straight line. Incorporating arcs in animation helps to make movements look more organic and visually pleasing.

8. **Exaggeration:** Exaggeration is used to emphasize certain aspects of a movement or action, making it more expressive and engaging. It allows animators to go beyond strict realism to create visually appealing and entertaining animations.

9. **Secondary Action:** Secondary actions are additional movements that complement the main action, adding depth and richness to the animation. They can enhance the storytelling and add more character to the scene.

10. **Appeal:** The principle of appeal involves designing characters, objects, and movements in a way that captivates and engages the audience. It focuses on

creating visually interesting and appealing elements that resonate with viewers.

By incorporating these principles of animation, animators can bring life, emotion, and believability to their computer-generated animations, creating captivating and immersive experiences for the audience.

types of animation in computer graphics

1. **Traditional Animation:** Also known as cel animation, traditional animation involves creating each frame of the animation by hand-drawing or painting on transparent celluloid sheets (cels) and photographing them in sequence.

2. **2D Computer Animation:** 2D computer animation refers to creating animations using digital tools and software. It involves manipulating 2D images, vector graphics, or sprites to create motion and visual effects.

3. **3D Computer Animation:** 3D computer animation involves creating animations in a three-dimensional virtual environment. It uses computer-generated models, textures, and lighting to bring objects, characters, and scenes to life.

4. **Stop Motion Animation:** Stop motion animation involves capturing individual frames by physically manipulating real-world objects or models. Each frame is photographed, and when played in sequence, it creates the illusion of movement.

5. **Motion Graphics:** Motion graphics combine elements of graphic design and animation to create visually engaging and dynamic content. It is often used for title sequences, commercials, and visual effects in film and television.

6. **CGI Animation:** Computer-generated imagery (CGI) animation refers to the creation of realistic or stylized visuals entirely using computer graphics. It is extensively used in movies, video games, and simulations.

7. **Claymation:** Claymation, also known as clay animation, is a form of stop motion animation that uses clay or plasticine models. The models are sculpted and manipulated to create the desired movements.

8. **Cut-out Animation:** Cut-out animation involves using pre-existing images or shapes, usually cut from paper or cardstock, and animating them by moving and repositioning them frame by frame.

9. **Character Animation:** Character animation focuses on bringing characters to life through movement, expressions, and actions. It can be done using various techniques such as keyframe animation, motion capture, or puppetry.

10. **Interactive Animation:** Interactive animation involves animations that respond to user input or interactions in real-time. It is commonly used in video games, interactive websites, and user interfaces.

These are just some of the many types of animation in computer graphics. Each type offers unique approaches, techniques, and applications, allowing for diverse and engaging animated content.

Types of Animation Systems

1. Scripting Systems → predetermined scripts

- Scripting systems in animation use specialized languages or tools.
- They allow animators to control the behavior of characters and objects.
- Timeline control enables precise sequencing of actions.
- Event-driven programming triggers actions based on specific events.
- Expressive control allows manipulation of animation parameters.
- Integration with animation software enhances workflow.
- Extensibility and customization options are available.
- Scripting systems automate tasks and streamline workflows.
- Collaboration and reusability are facilitated.
- They provide fine-grained control for complex and dynamic animations.

2. Procedural Animation

- Procedural animation uses algorithms to generate animation automatically.
- It allows for real-time adaptability and responsiveness.
- Animation is created based on mathematical formulas, simulations, or procedural techniques.
- Procedural animations can exhibit natural variations and realistic behaviors.
- They are often used for simulating physical phenomena, character movement, and procedural effects.
- Procedural animation reduces the need for manual keyframing and scripting.
- It enables efficient generation of complex and dynamic animations.
- Procedural animations can be highly customizable and reusable.
- They offer flexibility for exploring different animation styles and variations.

3. Representational Animation

- Representational animation visually represents real-world objects, characters, or scenes.
- It focuses on realism and accuracy in the depiction of these elements.
- The goal is to create animations that closely resemble their real-life counterparts.
- It involves meticulous attention to detail in modeling, texturing, and rendering.
- Representational animation is commonly used in fields such as film, gaming, architecture, and scientific visualization.
- It requires advanced techniques in 3D modeling, character animation, and visual effects.
- The emphasis is on creating lifelike movements, textures, lighting, and shading.
- Representational animation aims to evoke a sense of believability and immersion for the viewer.
- It often requires extensive resources, rendering time, and computational power.
- The focus is on achieving high-quality visuals that accurately depict real-world elements.

4. Stochastic Animation

- Stochastic animation incorporates randomness and unpredictability into the animation.
- It aims to create non-repetitive and unique animations with natural variations.
- Simulations and procedural generation techniques are used to introduce randomness.
- Stochastic animation captures complexity, chaos, and emergent behaviors.
- It allows for creative exploration and unexpected results.
- Stochastic animation is utilized in experimental, artistic, and computational contexts.
- It enhances realism, engagement, and artistic expression in animations

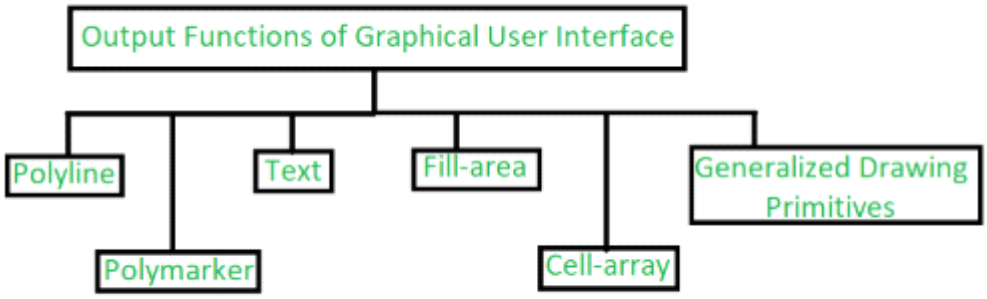
5. Behavioural Animation

- Behavioral animation focuses on creating realistic and lifelike behaviors for characters or objects.
- It simulates natural movements, interactions, and reactions.
- Character personalities and traits influence their behaviors.
- Motion capture or keyframing techniques are used to animate behaviors.
- AI techniques enable intelligent and autonomous character behaviors.

- Decision-making algorithms and planning mechanisms drive behavioral animation.

Graphical Kernel System

Graphical Kernel System is software which used for two-dimensional graphics. It was adopted as first graphics software standard by International Standard Organization (ISO). It has features for drawing in 2-dimensional vector graphics which is suitable for charting and similar purpose.
The 2-dimensional computer graphics which is closely related to six output functions of Graphical Kernel System (GKS).



1. **Polyline –**
   As from the name 'poly' means 'many'. Polyline is function which has ability to draw **one or more straight lines** through coordinates` which user has given to them.

2. **Polymarker –**
   This function is used to **draw a symbol** at coordinate which user has provided. There are 5 types of symbols which is used by this software namely **: x + * 0**.

3. **Text –**
   This function is used to **add text** at given coordinates by user.

4. **Fill-area –**
   In this feature, it allows a polygon to be draw and it can be filled with coordinates which are given. There is variety of fill-area which includes **hollow, solid** and there is also **variety of hatching and patterns**.

5. **Cell-array –**
   In this firstly pattern is defined by user and it **outputs in rectangle** according to given coordinates by user.

6. **Generalized Drawing Primitives –**
   It provides user various kinds of facilities. Mostly all of systems has various kinds of software for arcs of circle or ellipse and also drawing of a smooth curve with set of given points.

GKS Workstation:

1. GKS is a standardized graphics programming system that provides a common interface for creating and manipulating graphics in computer applications.

2. A GKS Workstation refers to the physical output device or display on which the graphics output is rendered. It can be a monitor, printer, plotter, or any other graphics output device.

3. GKS Workstation provides a standardized set of commands and functions for creating and rendering graphics primitives, such as lines, polygons, text, and images, on the output device.

4. Workstation attributes, such as color, line style, and text font, can be set and controlled to customize the appearance of the graphics output.

5. GKS Workstation allows for device-independent programming, meaning that the same set of graphics commands can be used across different output devices without modification.

Metafiles:

1. A metafile is a file format that stores a sequence of graphics commands and data representing a graphical image or object.

2. Metafiles are used to store graphics output in a device-independent format, allowing for portability and playback on different output devices.

3. Metafiles contain a record of graphics commands and attributes used to create the image, enabling the recreation of the graphics output on different systems.

4. Metafiles can be used for archiving, printing, and sharing graphics output, as they capture the drawing instructions and associated attributes.

5. Metafiles can be created using GKS or other graphics systems and can be stored in various formats, such as vector-based or raster-based metafile formats.

6. Metafiles can be replayed or rendered on different output devices or systems, ensuring consistent graphics output across platforms.

7. Metafiles provide a means of preserving and transferring graphical information in a compact and platform-independent manner.

In summary, GKS Workstation refers to the output device used for rendering graphics in a standardized graphics programming system, while metafiles are file formats used to store graphics commands and data for device-independent representation and playback.