29-oct

Computer Graphics Test

21118

Page No.
Date

1)

→ DDA line drawing algorithm:

i) DDA stands for Digital Differential Analyzer
ii) The DDA is one of the fastest line drawing algorithm
iii) It is easiest method of line drawing

Steps:

i) Consider the two endpoints of line as (x_0, y_0) (x_1, y_1)
ii) Calculate $dy = y_1 - y_0$ & $dx = x_1 - x_0$
iii) Calculate number of steps.
number of steps = max (abs(dx), abs(dy))
iv) Calculate $x_{in} = \frac{dx}{\text{no. of steps}}$ &
$y_{in} = \frac{dy}{\text{no. of steps}}$
v) increase x_0 by x_{in} till we get x_1
& increase y_0 by y_{in} till we get y_1
In the process draw each pixel.

Disadvantages of DDA line drawing algorithm:

i) DDA line drawing algorithm involves floating point operations. Due to which round-off error may occur.
ii) Also processing time of such operations is higher.
iii) The sharp line is not the output (in most cases) in this algorithm.

Example:

i) Here A(-2,-1) & B(6,3)
ii) $dx = 6-(-2) = 8$
& $dy = 3-(-1) = 4$

iii) no. of steps = max (dx, dy)
= max (8, 4)
= 8

iv) $x_{increment} = \frac{dx}{steps} = \frac{8}{8} = 1$

$y_{increment} = \frac{dy}{steps} = \frac{4}{8} = 0.5$

v) each next pixel (x_n, y_n) can be given as
$(x_{n+1} + x_{in}, y_{n+1} + y_{in})$
∴ pixels are (-2,-1) (-1, -0.5) (0,0) (1,0.5)
(2,1) (3, 1.5) (4, 2) (5,2.5)
(6,3)

~~display representation~~:
pixels are always integer values.
So. rounding off the above values
we get
(-2,-1) (-1,0) (0,0) (1,2) (2,1)
(3,2) (4,2) (5,3) (6,3)

pixels after rounding off
(-2,-1)
(-1,)
(0,0)
(1,0)
(2,1)
(3,1)
(4,2)
(5,2)
(6,3)

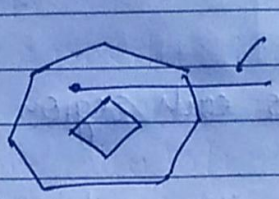
2)
→ During polygon filling only those pixels should be filled which are inside the polygon.
- Accurate decision of pixel, wheather it is inside (a) outside polygon is important & there are two methods to determine that
i) Even-odd method
ii) Winding Number method

Even-odd method:
i) This method constructs a line from a given point outside the polygon & checks the intersect

of a line with polygon boundry.

ii) Draw a line from any point inside the polygon to any point outside the polygon, and count the number of intersections of line with polygon.

iii) If the no. of intersections are odd, then the point is inside the polygon. Otherwise, it is outside the polygon.

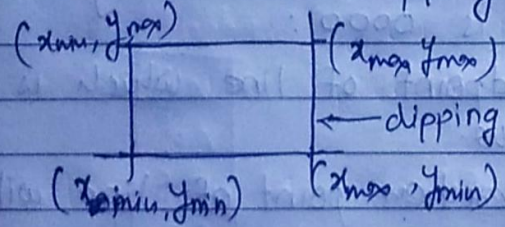


9)

→ Cohen-Sutherland line clipping Algorithm:

i) In line clipping, we will remove the portion of line which is outside of window.

ii) In Cohen-Sutherland line clipping we consider co-ordinates of clipping window, let's say.



iii) Also we use 4-bits to divide entire region.
The 4-bits are in the order Top, Bottom, Right, Left.
They are as shown in fig.

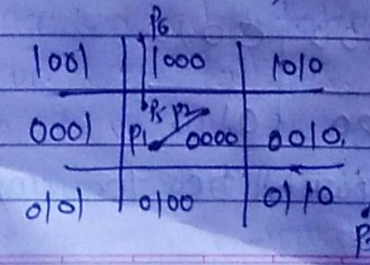
1001	1000	1010
0001	0000	0010
0101	0100	0110

iv) There are 3 possibilities for any line:

a) Line can be completely inside the window.
→ This line should be accepted completely
b) Line can be completely outside of the window
→ This line should be rejected completely
c) Line can be partially inside the window
→ We have to find intersection point & take the pixels inside the window.

Algorithm:
Step1: Assign region code for each region in the window.
Step2: If both endpoints have the code 0000 then the line lies inside the window. Take it completely.
Step3: Else perform logical AND operation with both endpoints.
If the result is not 0000, then line lies completely outside the window. Reject it.
Step4: else if result is 0000:
i) choose endpoint of line which is outside of window.
ii) find intersection point of line with window boundry.
iii) Replace endpoint with intersection point & update region code.
iv) Repeat step 2 until we find a clipped line either trivially accepted (or) trivially rejected

Example:



consider line P_1P_2 : both endpoints have code 0000 hence it is trivially accepted.

consider line P_3P_4 : ~~both endpoints have~~ Performing logical AND (0010 AND 0110) the result is 0010 which is not 0000. Hence this line is trivially rejected.

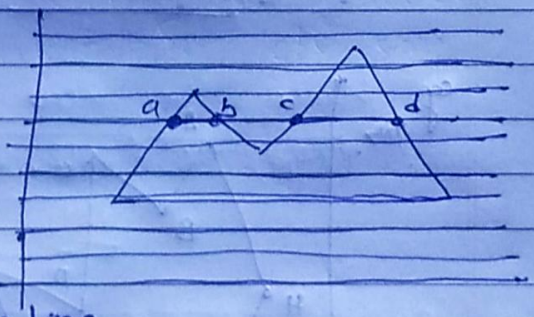
consider line P_5P_6: This line is partially accepted as logical AND operation of endpoints is 0000. In this case we have to clip the line by finding intersection point.

4)

→ Polygon fill with scan-line Algorithm:

~~polygon filling~~: 1) scanline fill algorithm computes the visible span of each scan line with polygon boundries. It works with all types of polygons & holes.

2) The scan line intersects polygon boundry at a, b, c, d,



3) Scan fill algorithm works as follows:
- find intersection of scan-line with polygon edges.
- sort all intersection on their x-co-ordinate
- fill the span using a pair of sorted intersection points.

4) when scanline intersects the polygon boundry, parity is odd & when it intersects again, parity is even. fill the span b/w odd to even parity. Initially parity would be even.

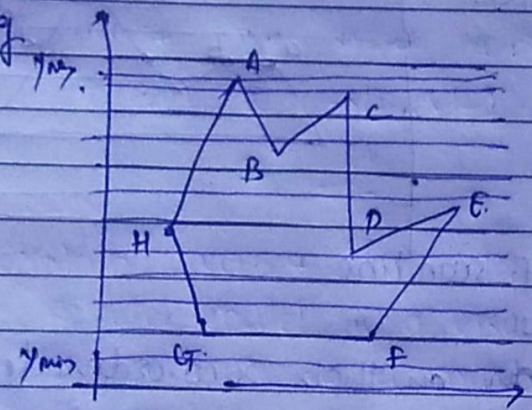
5) When scan-line intersects the polygon edge at point a, parity becomes odd, when it intersects edge at point b, parity becomes even. Hence, fill the span b/w a & b Similarly parity would be odd for c & and even for d. So span of c to d will be filled, whereas span from b to c will not be filled, because parity is same.

6) in scanline algorithm $y_{k+1} = y_k + 1$
& $x_{k+1} = x_k + 1/m$.

7) Algorithm uses three data structures called Edge table (ET) & Active Edge Table (AET) & Global Edge Table. Edge table maintains the following information for each edge in a polygon: except horizontal edges

y_{max}	x_{min}	$\frac{1}{m}$	next

eg



Edge list

AH
AB
BC
CD
DE
EF
HG
GF

After setting up edge-table algorithm works as follows:

i) Set Yscan to smallest y for which there is an entry in Edge Table (ET)

ii) Active edge table is initially empty.

Page No.
Date

ii) Repeat the following steps until both AET & ET are empty:
- a) Move edges from ET to AET whose $y_{min} = y_{scan}$.
- b) fill the visible span
- c) Remove edges with $y_{max} = y_{scan}$
- d) Increment y by 1
- e) for each edges in AET, update x-co-ordinate
- f) Sort all edges on X.

5) Write transformation matrix for:

i) 2D reflection wrt. Y-axis

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

ii) 3D Rotation about X-axis

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

6) Consider square P(0,0), Q(0,10), R(10,10), S(10,0). Rotate the square anticlockwise about fixed point R(10,10) by an $\angle 45°$.

→ here $\theta = 45°$

$$R_H = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos 45° & -\sin 45° & 0 \\ \sin 45° & \cos 45° & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

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

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{x}{\sqrt{2}} & \frac{y}{\sqrt{2}} \\ \frac{x}{\sqrt{2}} + \frac{y}{\sqrt{2}} & \\ 1 & \end{bmatrix}$$

In our case

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 10 & 10 \\ 0 & 10 & 10 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\therefore \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 10 & 10 \\ 0 & 10 & 10 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -10/\sqrt{2} & 0 & 10/\sqrt{2} \\ 0 & 10/\sqrt{2} & 10\sqrt{2} & 10/\sqrt{2} \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Hence the new coordinates of square are $P(0,0)$, $Q'(-5\sqrt{2}, 5\sqrt{2})$, $R'(0, 10\sqrt{2})$, $S'(5\sqrt{2}, 5\sqrt{2})$

Ans

7) Explain RGB & HSV color model.

→ A color model is a visualization that depicts the color spectrum as a multidimensional model. Most modern color models have 3 dimensions.

RGB:

1) It is a color model with three dimensions - red, green & blue. These three colors are mixed in specific proportion to produce specific color.
2) The RGB color model is often depicted as a cube

by mapping the red, green & blue dimensions onto x, y & z-axis in 3D space.

3) The RGB color model is not an especially intuitive model for creating colors in code.

HSV:

1) It is a cylindrical color model that remaps the RGB primary colors into dimensions that are easier for humans to understand.
Like the Munsell Color System, these dimensions are hue, saturation & value.
   i) hue: specifies angle of color on RGB color circle.
   ii) Saturation controls the amount of color used.
   iii) Value controls brightness of color.

2) Three dimensions of HSV color model are interdependent
If value of ~~three~~ dimension of a color is 0%, the amount of hue & saturation doesn't matter as color will be black.
If saturation of color is set to 0%, the hue doesn't matter as no color used.