Scan Conversion / Rasterization:

Process of converting 2D or 3D graphic primitives (eg: point, circle, filled polygons) from their geometric definitions into set of pixels in the image space.

→ handled by computer program / graphics

* mathematical property → actual pixels

Scan converting a point steps:

1) Input (x, y) point

2) As the pixel grid is discrete, round the coordinates to the nearest integer values

$$x' = \text{floor}(x + 0.5)$$
$$y' = \text{floor}(y + 0.5)$$

Ex: (2.3, 4.6)

→ (floor (2.3 + 0.5), floor (4.6 + 0.5))

→ (floor (2.8), floor (5.1))

→ (2, 5)

3) Plot pixels

Graphic primitives: basic graphic elements, fundamental shapes

→ has position, size, shape

CamScanner

Scan converting a Line

1) Direct Line Equation method

→ (2, 2) (7, 5) $\boxed{y = mx + b}$

$$m = \frac{5-2}{7-2} = \frac{3}{5} \rightarrow b = y - mx$$

$$\therefore b = 5 - \frac{3}{5} \cdot 7$$

$$= \frac{25-21}{5} = \frac{4}{5}$$

$\therefore x = 3, \quad y = 2\frac{3}{5} \approx 3$

$x = 4, \quad y = 3\frac{1}{5} \approx 3$

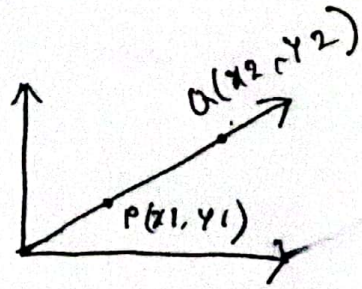
$x = 5, \quad y = 3\frac{4}{5} \approx 4$

$x = 6, \quad y = 4\frac{2}{5} \approx 4$

Cons : too slow, Pros: Easy

2 cases

a) $|m| \leq 1$



→ To draw line PQ.

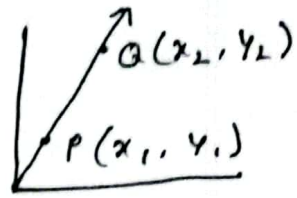
$x = x++$ from P

$y = mx + b$

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

y = round (y), do until $x_1 < x_2$

b) $|m| > 1$



To draw line PQ.

$y = y++$ from P

$x = (y - b)/m$

x = round (x), do until $y_1 < y_2$

2) Digital Differential Analyzer (DDA) Algo

speeding up ~~sca~~ direct line eqn method

2 cases:

a) $|m| \leq 1$,

$\rightarrow x(i+1) = x(i) + 1$

$y(i+1) = y(i) + m$

```
while (x0 <= xf) {
    x0++;
    y0 += m;
    y0 = floor (y0 + 0.5)
}
```

b) $|m| > 1$

$\rightarrow y(i+1) = y(i) + 1$

$x(i+1) = x(i) + \frac{1}{m}$

```
while (y0 <= yf) {
    y0++;
    x0 += 1/m ;
    x0 = round (x0)    // floor(x0+0.5)
}
```

Pros: 1) fasten than direct line eqn. method

2) no need of multiplication

Cons: 1) as we round the value, errors add in each step.

2) make line slightly off from the real

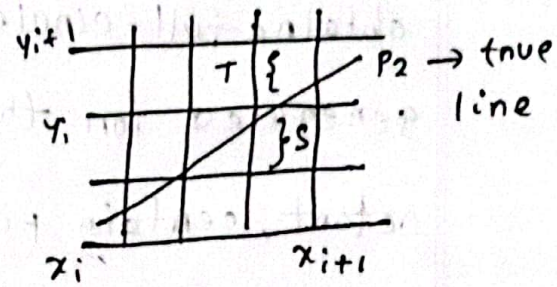
3) rounding + floating point arithmetic → slow.

Direct Line	DDA
– Best for precision	– best for computation
– slow	– fast
– low precision loss	– high precision loss
– more accurate end point	– less accurate end point (may never reach)

3) * Bresenham's Line Drawing Algo:

$P(x_1, y_1)$; $Q(x_6, y_6)$

Base case: $\boxed{d_1 = 2\Delta y - \Delta x}$



Then:

if $(d_i \geq 0)$

$d(i+1) = d_i + 2\Delta y - 2\Delta x \rightarrow (x_{i+1}, y_{i+1})$

if $(d_i < 0)$: $d(i+1) = d_i + 2\Delta y \rightarrow (x_{i+1}, y_i)$

$\Delta y = y_6 - y_1$ $\quad \Delta x = x_6 - x_1$

Pros: 1) No more floating point calculations

2) No need to round up values

3) Fast and better precision

Cons: 1) Can't draw smooth line

Final Ex: $d_i = -3$, $x_i = 3$, $y_i = 2$; $x_{i+1} = ?$ $y_{i+1} = ?$

use Bresenham.

→ $d_i = -3 < 0$

$\therefore x_{i+1} = x_i + 1 = 3 + 1 = \boxed{4}$

$y_{i+1} = y_i = \boxed{2}$

7

Overstrike: When 8-way symmetry is used to obtain a full-circle from pixel coordinates generated for the 0° to 45° on the 90° to 45° octant, certain pixels are set or plotted twice. This phenomenon is known as overstrike.

Final

* How to prevent it?

→ We need to ensure that each pixel is plotted only once.

1) Selective quadrant skipping: skip pixels that are equidistant from circle's center and fall on $x=0$ or $y=0$.

2) Update decision variable so that each pixel is considered only once.

3) Maintain symmetry across all octants of the circle to ensure consistent plotting.

CamScanner

Final

Bresenham	vs	Line Drawing
– use simple integer arithmetic		→ use floating-point arithmetic
– very efficient (speed).		– less efficient

Final: Plot a dashed circle using Bresenham's circle drawing algo:

→ Define a dashed pattern that specifies which pixels to plot and which to skip.

Create an array of boolean values where

true → pixel to plot
false → " " skip

Modified algo:

```
def draw_dashed_circle(xc, yc, r, dashed_pat):
    x = 0
    y = 0
    d = 3 - 2*r

    while x <= y:
        for i in range(len(dashed_pat)):
            if dashed_pat[i]:
                plot_pixel(xc+x, yc+y)   // 1st q.
                plot_pixel(xc+y, yc+x)   // 2nd
```

modify ←

```
plot_pixel (xc - y, yc + x)      // 3rd
plot_pixel (xc - x, yc + y)      // 4th
plot_pixel (xc - x, yc - y)      // 5th
plot_pixel (xc - y, yc - x)      // 6
plot_pixel (xc + y, yc - x)      // 7
plot_pixel (xc + x, yc - y)      // 8
x ++
if d <= 0  d += 4 * x + 6
else :
    y --
    d += 4 * (x - y) + 10
```

Final

* Apply Bresenham to draw a line from (2,3) to (8,12). Show ~~sets~~ steps and resulting pixel coordinates.

→ $\Delta x = 8 - 2 = 6$

$\Delta y = 12 - 3 = 9$

Initial decision variable, $d = 2\Delta y - \Delta x$

$= 2 * 9 - 6 = 12$

Starting pixel $(x, y) = (2, 3)$

Steps: $d_{i+1} = d_i + 2\Delta y - 2\Delta x$

1) $d = 12 \geq 0$ $\quad d_{i+1} = 12 + 2 \cdot 9 - 2 \cdot 6 = 18$

$\therefore (x, y) = (3, 4)$

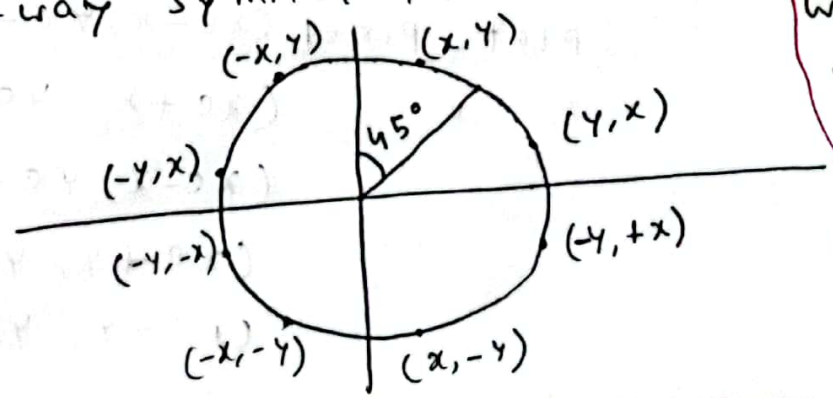
2) $d = 18 \geq 0$ $\quad d_{i+1} = 18 + 18 - 12 = 24$

$\therefore (x, y) = (4, 5)$

Hence, → $\boxed{(5,6), (6,7), (7,8), (8,9), (9,10), (10,11), (11,12), (12,13)}$

17 What's 8 way symmetry for a circle?

→ Circle is an eight-way symmetric figure. The shape of circle is the same in all quadrants. In each quadrant, there are 2 octants. (1/8) If the calculation of the point of one octant is done, then the other 7 points can be calculated easily by using the concept of 8-way symmetry.



We'll only calculate 45°. from this → whole circle

(tt) * In Bresenham, if $d_i = 0$, guananteed that S and T are equally far from true circle?

3.17 → $\boxed{\text{Yes}}$ There's chance that T and S are not always equally far.

$d_i = 0$; midpoint between S and T is very close to true circle

because, algorithm's approximation can lead to minor deviations from the true circle shape

(tt) * Modify "Midpoint circle drawing" algo to draw a dashed circle.

→ length of each dash

```
→ func draw (xc, yc, r, dash_len):
        x = r , y = 0 , d = 1 - r , dash_cnt = 0
    while x >= y ;
        if dash_cnt = 0:
            plot_pixel (xc+x, yc+y)
            plot_pixel (xc-x, yc+y)
                       (xc+x, yc-y)
                       (xc-x, yc-y)
                       (xc+y, yc+x)
                       (xc-y, yc+x)
```

plot_pixel(xc + y, yc - x)
" (xc - y, yc - x)

dash_cnt = (dash_cnt + 1) % (2* dash_len)

if d < 0: d = d + 2* y + 3, y = y + 1

else: d = d + 2* (y - x) + 5

y++, x--

→ dash_cnt to track whether the algo should plot or skip pixels to create a dashed pattern.

plot for → 1 dash length

skip for → next "

Final 16

* DDA vs Bresenham

DDA	Bresenham
1) only multiplication and division are used	1) addition and subtraction
2) ~~not~~ less efficient	2) more efficient
3) not fast	3) fast
4) less accuracy and precision	4) more
5) less optimized	5) more optimized
6) high cost	6) less cost

CamScanner

Final

* The endpoints of a line (0,0) and (6,18). Compute each value of y as x steps from 0 to 6 and plot the results.

$\rightarrow$ $y = mx + b$ $\quad \therefore m = \frac{18-0}{6-0} = 3$

$\therefore b = y - mx$

$= 0 - 3 \cdot 0 = 0$

$\therefore y = 3x$

$\therefore x = 0, y = 0 \rightarrow (0,0), (1,3) \ldots (6,18)$

Final

* what are $\boxed{4}$ connected and $\boxed{8}$ connected pixels. example and dignam.

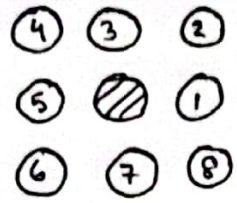
$\rightarrow$ pixels are considered connected to each other to form a continuous boundary:

2 methods for determining pixel connectivity:

4-connected and 8-connected.

4: A pixel may have up to 4 neighbors: (above, below, left, right)

8 : 8 neighbors (above, below, right, left and diagonally adjacent)



* Bresenham's Circle Algo (increment)

$x^2 + y^2 = r^2$ Given x, y, r

1st decision parameter, $d = 3 - (2 * r)$

if $(d_i < 0)$ $d_{i+1} = d_i + 4x + 6$, $x++$

if $(d_i \geq 0)$ $\boxed{d_{i+1} = d_i + 4(x-y) + 10 \quad x++, y--}$

* Midpoint circle-drawing Algo :

Base $(0, r)$, $d = (5/4) - r$

if $(d_i < 0)$ $d_{i+1} = d_i + 2(x_i + 1) + 1$

if $(d_i \geq 0)$ $\begin{cases} d_{i+1} = d_i + 2(x_i + 1) + 1 - 2(y_i - 1) \\ y-- \end{cases}$

final * $\boxed{\text{Circle symmetry property}}$: circle looks the same when you rotate/reflect it.

→ same look from diff. angle/direction

→ because it has certain points / line that

$\boxed{3.17}$ decision variable, $d_i = D(T) + D(S)$ to help choose between pixels S and T.

D → not true measure of distance from center of a pixel to the true circle.

→ d_S → actual distance from S to true circle

d_T → " " " T " " "

substitute. $x \to x + 1$

$y \to y_i$

$\therefore d_i = 2x^2 + y^2 + (y-1)^2 - 2r^2 = 0$

Since, $(r + d_T)^2 = x^2 + y^2$ —— (1)

$(r - d_S)^2 = x^2 + (y-1)^2$ —— (2)

$\therefore$ (1) → $2r d_T + d_T^2 = \boxed{x^2 + y^2 - r^2}$ —— (3) $= 0$

(2) → $-2r d_S + d_S^2 = \boxed{x^2 + (y-1)^2 - r^2}$ —— (4) $= 0$

$\therefore 2r d_T + d_T^2 - 2r d_S + d_S^2 = 0$ (3) + (4)

→ $d_T (2r + d_T) = d_S (2r - d_S)$

→ $\frac{d_T}{d_S} = \frac{2r - d_S}{2r + d_T} < 1$

$\therefore \boxed{d_T < d_S}$ $\therefore d_i = 0$ → T is closen to true circle than S

* What do you require for 3D viewing an object? mathematically how do you specify the view plane?

→ Requirements for 3D Viewing:

1) 3D object: A model defined by vertices, edges and surfaces.

2) Camera setup:

a) Position: where the camera is located.

b) Orientation: The direction the camera is pointing.

c) Field of View (FOV): The extent of the scene visible to the camera.

d) Clipping Planes: Near and far distances where objects are visible.

Specifying the view plane: The view plane is the surface where the 3D scene is projected to form a 2D image. Here's how to specify it:

1) Camera position (Eye): $E = (E_x, E_y, E_z)$

2) Look-at Point $L = (L_x, L_y, L_z)$

3) Up vecton : $U = (U_x, U_y, U_z)$

4) Compute the camena Basis vectons:

* View Dinection D : $D = L - E$

Nonmalize D to get the unit vecton d

$$d = \frac{D}{|D|} = \left(\frac{D_x}{|D|}, \frac{D_y}{|D|}, \frac{D_z}{|D|}\right)$$

* Right vecton R :

$$R = U \times d$$

Nonmalize R to get the unit vecton n

$$n = \frac{R}{|R|}$$

* True up vecton U': $U' = d \times n$

unit vecton $u' = \frac{U'}{|U'|}$

* View plane center c :

$$c = E + \text{distance} \cdot d$$

distance from camena to the view plane

* view plane dimension:

① Height: $h = 2 \cdot d \cdot \tan\left(\frac{FOV_y}{2}\right)$

② width : $w = h \cdot$ aspect ratio

Transforming to the view plane :

To convert world coordinates to camera coordinates, use the view transformation matrix :

$$M_{view} = \begin{bmatrix} r_x & r_y & r_z & -r \cdot E \\ u_x' & u_y' & u_z' & -u' \cdot E \\ d_x & d_y & d_z & -d \cdot E \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$r, u', d \rightarrow$ camera's basis vector

$E \rightarrow$ camera position

18 Final

What's "Histogram Equalization" in image processing ?

→ a technique that adjusts the contrast of an image by redistributing the pixel values in its histogram.

<u>Pros:</u> simplicity and effectiveness in improving the contrast of images.

<u>Cons:</u> Artificial noise into the image. tendency to amplify noise.

18 Final

Q What's region code? What does each bit represent?

→ A region code is a 4-bit binary code used in the Cohen-Sutherland line clipping algorithm to indicate a point's position relative to a rectangular clipping window.

Each bit represents a direction.

TBPL

bit 3 → top	bit 1 → right
bit 2 → bottom	bit 0 → left

This code helps quickly determine whether a line segment is inside, outside or needs clipping.

18 Final

Q What's analog image?

→ a continuous visual signal, not divided into pixels. Typically captured by analog devices like film cameras or CRT televisions, where light is captured smoothly on film, resulting in detailed and seamless images.

CamScanner

18 Final

Find the transformation that scales (with respect to origin) by 'a' units in the x direction for point P(x, y)

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

$\therefore \quad P'(x' y') = (a.x, y)$

Conditions for the clipping candidate in the Cohen - Sutherland algorithm?

→ ① Both endpoints are inside the clipping window → accepted.

② Both outside → rejected

③ partially inside the window → needs further clipping.

How do you model an object in CG?

→ In CG, objects are typically modeled using geometric primitives like points, lines and polygons. The most common method is using meshes made up of triangles or other polygons that define the object's surface.

What are the geometric forms used for modeling of objects?

① Polygons (Triangles, Quads): Used for mesh models.

② Curves and Surfaces: for smooth surfaces.

③ Solid Geometry: basic shapes like cubes, spheres and cylinders.

Find the normalization transformation that maps a window whose lower left corner is at (1, 1) and upper right corner is at (3, 5) onto a viewport that's the entire normalized device screen. [(0, 0), (1, 1)] [(0, 0), (1/2, 1/2)]

→ $W_{x\,min} = 1$ $W_{x\,max} = 3$

$W_{y\,min} = 1$ $W_{y\,max} = 5$

$V_{x\,min} = 0$ $V_{x\,max} = 1$

$V_{y\,min} = 0$ $V_{y\,max} = 1$

$$\therefore S_x = \frac{1-0}{3-1} = \frac{1}{2} \qquad \frac{1/2 - 0}{3-1} = \frac{1}{4}$$

$$S_y = \frac{1-0}{5-1} = \frac{1}{4} \qquad \frac{1/2 - 0}{5-1} = \frac{1}{8}$$

CamScanner

$T_x = V_{x\,min} - W_{x\,min} \cdot S_x = 0 - 1 \cdot \frac{1}{2} = -\frac{1}{2} \,/\, -\frac{1}{4}$

$T_y = V_{y\,min} - W_{y\,min} \cdot S_y = 0 - 1 \cdot \frac{1}{4} = -\frac{1}{4} \,/\, -\frac{1}{8}$

∴ Nonmalization Transformation matrix:

$$\begin{bmatrix} S_x & 0 & T_x \\ 0 & S_y & T_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1/2 & 0 & -1/2 \\ 0 & 1/4 & -1/4 \\ 0 & 0 & 1 \end{bmatrix}$$

(2)

Types of Image Compression Techniques:

1) **Lossy Compression**

Ex: JPEG, MP3

- reduces file size by removing some data, resulting in loss of quality.
- suitable for photos, videos and audio where perfect accuracy is not necessary
- web image, streaming videos, audio files

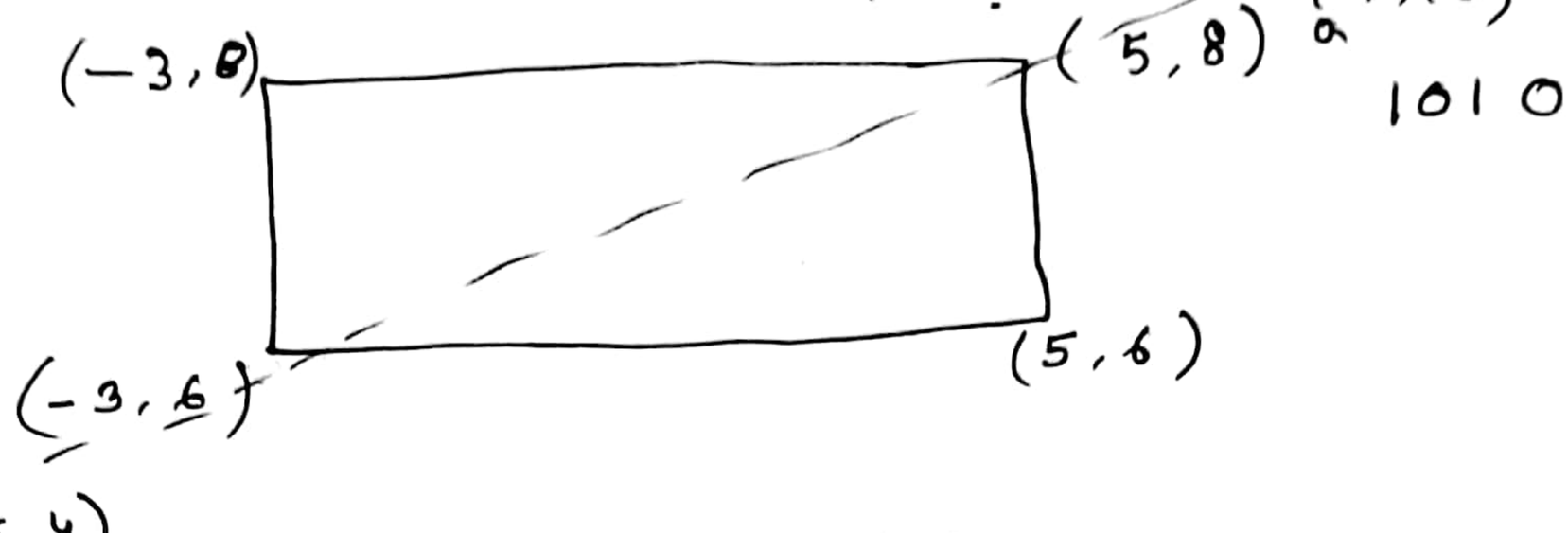
2) **Lossless**

Ex: PNG, GIF, FLAC

- reduce file size without losing data maintaining original quality.
- suitable for text and data file, where accuracy is crucial.
- text documents, sw, medical imaging

Apply Cohen - Suthenland line clipping to clip a line segment with endpoints (-5, 4) and (7, 10) against a window with conners (-3, 8) and (5, 6).

$x_{min} = -3$, $x_{max} = 5$, $y_{min} = 6$, $y_{max} = 8$



0101

P = 0101
Q = 1010
AND = 0000 → clipping

Clipping against left boundary $x = -3$:

for $(-5, 4)$: $m = \frac{10-4}{7+5} = \frac{6}{12} = \frac{1}{2}$

x_1 y_1

$y = y_1 + m(x - x_1)$

$= 4 + \frac{1}{2}(-3+5) = 5$

∴ ~~new~~ intensection point $(-3, 5)$

Clipping against right boundary $x = 5$:

for $(7, 10)$: $m = \frac{10-4}{7+5} = \frac{1}{2}$

$y = 10 + \frac{1}{2}(5-7) = 9$

∴ point $(5, 9)$

$(-3,5)$ bottom : $y = 6$

$x = -3 + 2(6-5)$

$= -1$

∴ $(-1, 6)$

top : $y = 8$ $(5, 9)$

$x = 5 + 2(8-9)$

$= 3$

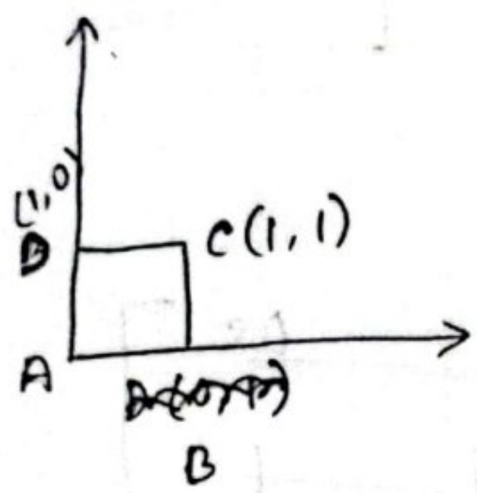
∴ $(3, 8)$

(4d2) The matrix $\begin{pmatrix} 1 & a \\ b & 1 \end{pmatrix}$ defines a transformation called a simultaneous sheaning or sheaning. The special case when $b = 0$ is

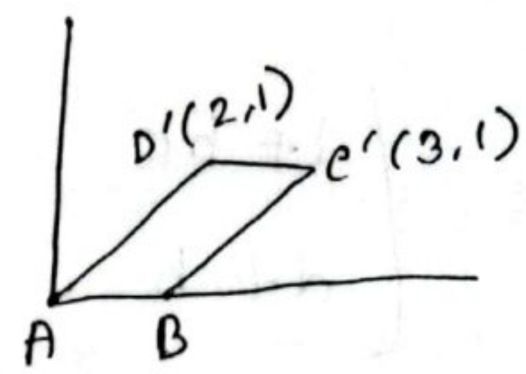
p-93

called sheaning in the x direction. When a = 0 we've sheaning in the y direction. Illustrate the effect of these sheaning transformations on the square A (0, 0), B (1, 0), C(1, 1) and D (0, 1) when a = 2, b = 3

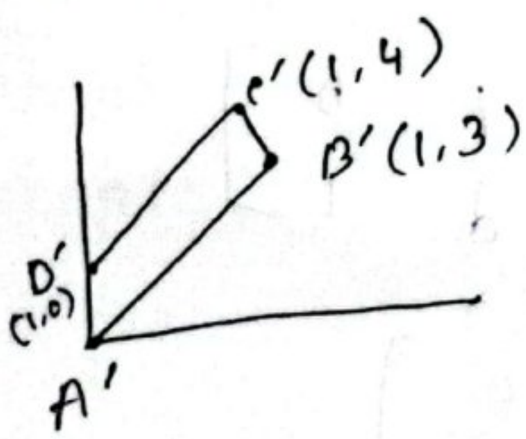
* original square:



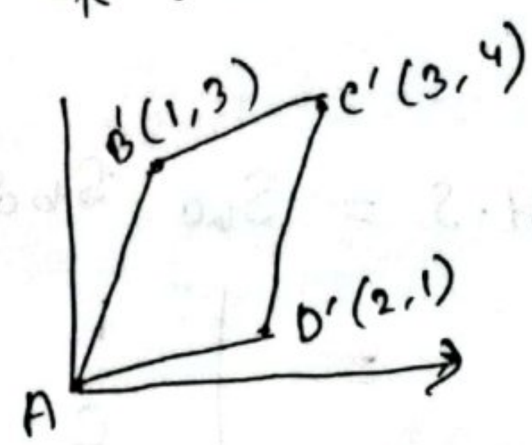
* sheaning in x direction



* sheaning in y:



* sheaning in both.



shearing in x axis: x' = x+ay
(0,0) -> (0,0)
(1,0) -> (1+2.0,0) -> (1,0)
(1,1) -> (1+2.1,1) -> (3,1)
(0,1) -> (0+2.1,1) -> (2,1)

shearing in y axis: y' = y + bx
(0,0) -> (0,0)
(1,0) -> (1,0+3.1) -> (1,3)
(1,1) -> (1,1+3.1) -> (1,4)
(0,1) -> (0,1+3.0) -> (0,1)

shearing in x and y axis: x' = x + ay, y' = y + bx

(0,0) -> (0,0)
(1,0) -> (1+2.0,0+3.1) -> (1,3)
(1,1) -> (1+2.1,1+3.1) -> (3,4)
(0,1) -> (0+2.1,1+3.0) -> (2,1)

6.2) Tilting is defined as a rotation about the
p-128 x-axis followed by y-axis.

① find the tilting matrix.

② Does the order of performing rotation matter?

→ ⓐ $T = \cancel{R_x(\theta)} \cdot R_y(\phi) \cdot R_x\theta$

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

$= \begin{bmatrix} \cos\phi & \sin\theta\sin\phi & \sin\phi\cos\theta \\ 0 & \cos\theta & -\sin\theta \\ -\sin\phi & \cos\theta\cos\phi & \cos\phi\cos\theta \end{bmatrix}$

ⓑ To determine, If order matters, we need to consider the opposite order.

$T' = R_x\theta \cdot R_y\phi$

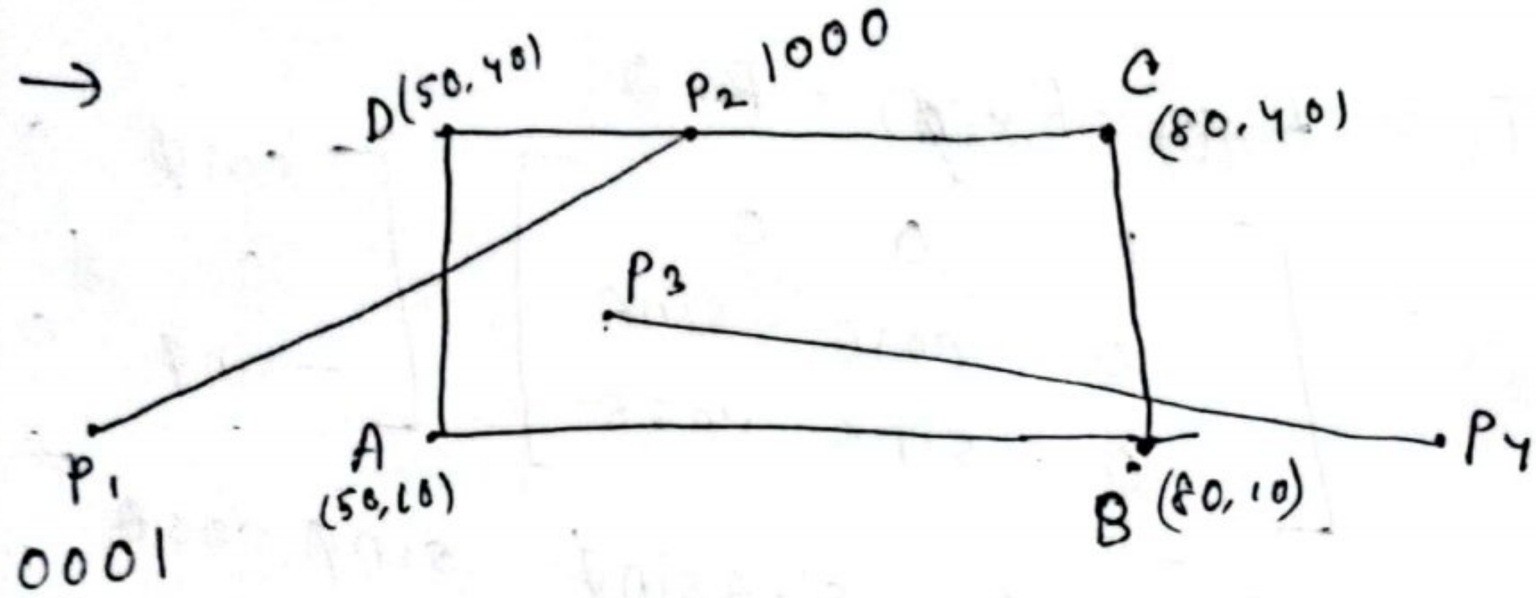
$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix}$

$= \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ -\sin\theta\sin\phi & \cos\theta & -\sin\theta\cos\phi \\ -\sin\phi\cos\theta & \sin\theta & \cos\theta\cos\phi \end{bmatrix}$

$\boxed{\therefore T' \neq T \text{ Yes does matter}}$

Use the Cohen - Suthenland algo to clip two lines P1 (35, 10), P2 (65, 40) and P3 (65, 20) P4(95, 10) against against a window A(50, 10), B(80, 10) C (80, 40), D (50, 40). Also, find the clipping position.

→



Line 1: $P_1 = 0001$

$P_2 = 1000$

And $= 0000$ → Line is pantially visible.

$$m = \frac{40-10}{65-35} = \frac{30}{30} = 1$$

$$y_1 = m(x_2 - x_1) + y_1$$

$$= m(50 - 35) + 10$$

$$= 1 \cdot 25 + 10 = \cancel{40}\ 35$$

$$x_1 = \frac{1}{m}(y_T - y_1) + x_1$$

$$= 1(40 - 10) + 35 = 65$$

$y_2 = m(x_R - x) + y = 1(80 - 65) + 40 = 55$

$x_2 = \frac{1}{m}(y_B - y) + x = \frac{1}{1}(10 - 40) + 65 = 35$

Line 2 :

$P3 = 0000$

$P4 = 0010$

$AND = 0000 \rightarrow$ pantially visible

$m = \frac{10-20}{95-65} = \frac{-10}{30} = -\frac{1}{3}$

$y_1 = m(x_L - x) + y = -\frac{1}{3}(50 - 65) + 20$

$= -\frac{1}{3}(-15) + 10 = 25$

$x_1 = \frac{1}{m}(y_T - y) + x = -3(40 - 20) + 65$

$= -3 \cdot 20 + 65 = \cancel{-5}\ 5$

$y_2 = m(x_R - x) + y \cancel{= -\frac{1}{3}(80 - 80) +}$

$= -\frac{1}{3}(80 - 95) + 10 = 15$

$x_2 = \frac{1}{m}(y_B - y) + x$

$= -3(10 - 10) + 95 = 95$

$\therefore$ clipping : $P_1'(65, 35)$, $P_2'(35, 55)$, $P_3'(5, 25)$, $P_4'(95, 15)$

Name 2 algonithms that can be used to find out the visible pontion of given lines in a centain viewpont:

① Cohen - Suthenland Line Clipping algo: uses a divide - and - conquen strategy to necunsively divide the line and check if each segment lies inside the clipping window on needs funthen clipping.

② Liang - Bansky Line Clipping Algo: uses parametric line equations and inequalities to pen fonm clipping mone effeciently by reducing the numben of intensection calculations required.

Resolution: no. of pixels / unit length
↓
clanity of pic

window to viewpont mapping is the process of transfonming a specific pant of a langen scene (the window) to fit within a display area (the viewpont). fon example, if you've a

map showing coordinates from (50, 50) to (150, 150) and you want to display it on a screen area from (0, 0) to (400, 400). you scale and translate the map coordinates to fit this screen area.

Sol. $S_x = \dfrac{V_{xmax} - V_{xmin}}{W_{xmax} - W_{xmin}}$, $S_y = \dfrac{V_{ymax} - V_{ymin}}{W_{ymax} - W_{ymin}}$

Express window-to-viewport mapping in the form of a composite transformation matrix.

① Translate the window to origin:

$$T_1 = \begin{bmatrix} 1 & 0 & -W_{xmin} \\ 0 & 1 & -W_{ymin} \\ 0 & 0 & 1 \end{bmatrix}$$

② scale the window to fit the viewport.

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

③ Translate the scaled window to viewport.

$$T_2 = \begin{bmatrix} 1 & 0 & V_{xmin} \\ 0 & 1 & V_{ymin} \\ 0 & 0 & 1 \end{bmatrix}$$

$$\therefore \boxed{T = T_2 \cdot S \cdot T_1}$$

CamScanner

Composite Trans-formation matrix

$$= \begin{bmatrix} S_x & 0 & Vx_{min} - S_x \cdot Wx_{min} \\ 0 & S_y & Vy_{min} - S_y \cdot Wy_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

This matrix can be used to map any point in the window to its connesponding point in the viewpont.

Polygon vs polylines

Polygon	polyline
① Always a closed shape	① can be either open on closed
② Ex: Triangle, squane	② Ex: zigzag line, a walking path.
③ used to nepnesent bounded aneas, formed by a finite no. of line segments connected end-to-end.	③ used to nepneset paths, noutes that don't necessanily fonm a closed shape.

Compute size of a 640 x 480 image at 240 pixels pen inch.

$$\rightarrow \frac{640}{240} \times \frac{480}{240} = 2.67 \times 2 \text{ inch.}$$

Composite Transformation:

→ refers to the combination of two or more basic transformations (such as translation, rotation, scaling) into a single operation. This is achieved by multiplying the corresponding transformation matrices together.

→ $T_v . S . T_v^{-1}$ → translation, scaled, retranslation

Ideally view volume is infinite but in reality we prefer to use a finite volume. Why

→ It's computationally manageable, focusing resources on what's visible. The clipping process is efficient and straightforward.

How do you specify the finite view volume?

→ you set boundaries that describe how far and wide the camera can see. This include deciding how close and far objects appear, and the shape of what's visible, whether like a box or a pyramid.