Date

2. Digital Differential Analyzer (DDA) ~~System~~ line Algorithm:

a. It is an incremental algorithm (Next step uses previous step; won't use line eq to determine next point)

$x_k\ y_k$ next point $x_{k+1}\ y_{k+1}$

such that $m = \frac{\Delta y}{\Delta x} = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$ (All in continuous system).

$x_{k+1} = \Delta x + x_k = \frac{\Delta y}{m} + x_k$ (m is used only once).

$\boxed{x_{k+1} = \frac{\Delta y}{m} + x_k}$ $\boxed{y_{k+1} = m\Delta x + y_k}$.

b. This algorithm is based on calculating Δy & Δx.

c. We sample the line at unit intervals in 1 direction and determine corresponding integer value nearest to the line path. (Try to make Δx or Δy 1).

i) If slope <= 1 we sample at unit x interval ($\Delta x = 1$)
and $\boxed{y_{k+1} = m + y_k}$ k will be an integer y_{k+1} is
$\therefore$ m is real, $\therefore y_{k+1}$ is real, round y_{k+1} to nearest integer.

ii) $m > 1$ $\Delta y = 1$
$\boxed{x_{k+1} = \frac{1}{m} + x_k}$

} lines will be drawn from left to right. (lines with +ve slope)

iii) Lines from Right to left.
Slope <= 1
$\Delta x = -1$ $\boxed{y_{k+1} = y_k - m}$

iv) R to L
slope > 1
$\Delta y = -1$ $\boxed{x_{k+1} = x_k - \frac{1}{m}}$

Date

d. dx and dy are differences in each direction from start to end point. Difference with greater magnitude

e. determines 'steps' (how many times algorithm will run).

e. Offset needed at each state to generate next pixel along the line → x_{incr}, y_{incr} → for accuracy

eg. Line from L to R $dx > dy$ $x_{incr} = 1$ $dy = m$ $\left(\because \frac{dy}{dx} = m, dx = 1\right)$
R to L $dx > dy$ $x_{incr} = -1$ $dy = -m$

Algorithm:

```
#define Round(a) ((int)(a+0.5))
void line DDA (int xa, int xb, int ya, int yb)
{ int dx = xb - xa,
      dy = yb - ya,
      steps, k;
  float xincr, yincr, x = xa, y = ya;

  if abs(dx) > abs(dy)
     steps = abs(dx);
  else
     steps = abs(dy);
  xincr = dx/steps;          // one of them will become 1 or -1
  yincr = dy/steps;

  setpixel(round(x), round(y));     // to put in raster
  for (k=0; k<steps; k++)
   { x = x + xincr;
     y = y + yincr;
     setpixel(round(x), round(y));
   }
}
```

Disadvantages: Has round operation & costly

↳ Incremental algo (Makes use of previous calculations)

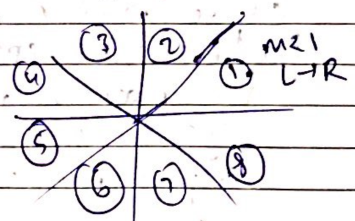
Bresenham's Algo. (Has no round function)

1. Input the two line endpoints and store the leftmost endpoint (x_0, y_0)
2. Load x_0, y_0 into frame buffer, i.e., plot the very 1st point.
3. Calculate the constants Δx, Δy, $2\Delta y$, $2\Delta y - \Delta x$ and obtain the starting value for decision parameter as $p_0 = 2\Delta y - \Delta x$
4. At each x_k along the line starting at $k=0$, perform the following test:
   If $p_k < 0$, the next point to plot is (x_k+1, y_k) and $p_{k+1} = p_k + 2\Delta y$

   Otherwise the next point to plot is (x_k+1, y_k+1) & $p_{k+1} = p_k + 2\Delta y - 2\Delta x$
5. Repeat step 4 Δx times

Δx → diff in x co-ordinates (Final - initial)
Δy → " " y " (" " ")

This algo works in the 1st octant (+ve slope <1) when we're moving L to R.



We are at step no. k in the algo.

At sampling position x_k+1, we label pixel separations from mathematical line d_1 and d_2.

Assumption → First point is integer
End " " "

y coordinate on the mathematical line at pixel position x_k+1 is

$$y = m(x_k+1) + b$$

$$d_1 = y - y_k \qquad d_2 = y_{k+1} - y = y_{k+1} - m(x_k+1) - b$$

$$d_1 - d_2 = 2m(x_k+1) - 2y_k + 2b - 1$$

$$m = \frac{\Delta y}{\Delta x}$$

$$d_1 - d_2 = \frac{2\Delta y}{\Delta x}(x_k+1) - 2y_k + 2b - 1$$

$$= \frac{2\Delta y(x_k+1) - 2\Delta x y_k + 2\Delta x b - \Delta x}{\Delta x}$$

$$\boxed{\Delta x(d_1 - d_2)} = 2\Delta y(x_k+1) - 2\Delta x y_k + 2\Delta x b - \Delta x$$

p_k — sign of p_k = sign of $d_1 - d_2$ — $\Delta x \to +ve$.

$$p_k = 2\Delta y x_k + 2\Delta y - 2\Delta x y_k + 2\Delta x b - \Delta x$$

$$= 2\Delta y x_k - 2\Delta x y_k + \underbrace{2\Delta y + \Delta x(2b-1)}_{\text{constant}}$$

= c (independent of pixel position)

$$\boxed{p_k = 2\Delta y x_k - 2\Delta x y_k + c} \quad (1)$$

$$p_{k+1} = 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + c$$

$x_k + 1$

$$p_{k+1} - p_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(\underbrace{y_{k+1} - y_k}_{\text{can take value 0 or 1}})$$

(y_k / y_k+1)

$$= 2\Delta y - 2\Delta x(y_{k+1} - y_k)$$

$$\boxed{p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)} \quad (2)$$

Calculation of p_0 (initial decision parameter):

Use (1)

$$p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + \Delta x(2b-1)$$

$y_0 = mx_0 + b$

$mx_0 + b - y_0 = 0$

Page No. ____ Date

$= 2\Delta y x_0 - 2\Delta x y_0 + \Delta x 2b + 2\Delta y - \Delta x$

Dividing and multiplying by Δx.

$\cancel{P_0 = \frac{2\Delta y}{\Delta x} x_0 - 2y_0 + 2b}$ *

$P_0 = 2mx_0\Delta x - 2\Delta x y_0 + \Delta x 2b + 2\Delta y - \Delta x$

$= 2\Delta x(\underbrace{mx_0 - y_0 + b}_{0}) + 2\Delta y - \Delta x$

$P_0 = 2\Delta y - \Delta x$

Mid-point Circle Drawing Algorithm :: → Draws 1 dot in each octant

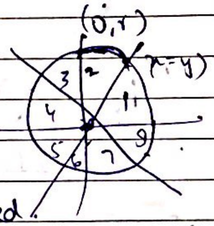
Assumptions: We generally take integer radius.

i) The circle we are drawing is at origin.

ii) We are drawing in the 2nd octant.

iii) Sampling in unit x direction

$f(x,y) = x^2 + y^2 - r^2 = 0$ (Eqt. of circle)



Assume we have just plotted (x_k, y_k)

Next point should be (x_{k+1}, y_{k+1}) to be plotted.

$P_k = f_{circle}(x_k + 1, y_k - 1/2)$.

$\Rightarrow P_k = (x_k+1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2$ — (1)

if $P_k < 0 \Rightarrow r$ is greater / mid-point is inside (East)

$\Rightarrow$ East pixel is chosen i.e. (x_k+1, y_k)

$P_k \geq 0 \Rightarrow r$ is smaller / mid-point is outside (South East)

$\Rightarrow (x_k+1, y_k-1)$

$P_{k+1} = (x_{k+1}+1)^2 + (y_{k+1} - 1/2)^2 - r^2$

Page No. ____ Date

* $\cancel{y_k^2} + 1 - 2y_k - \cancel{y_k} - \cancel{y_k} + 1 + \cancel{y_k}$

$-2(y_k - 1) = -2y_{k+1}$

$\cancel{= (x_k+2)^2 + (y_{k+1} - 1/2)^2 - r^2}$

$\cancel{= (x_k)^2 + 4x_k + 4 + (y_{k+1})^2 - y_{k+1} + \frac{1}{4} - r^2}$

~~Replacing r^2 from (1)~~

$\cancel{P_{k+1} =}$

$= (x_k+1)^2 + 1 + 2(x_k+1) + (y_{k+1})^2 + \frac{1}{4} - y_{k+1} - r^2$.

Replacing r^2 from (1)

$P_{k+1} = \cancel{(x_k+1)^2} + 1 + 2(x_k+1) + (y_{k+1})^2 + \frac{1}{4} - y_{k+1} + P_k - \cancel{(x_k+1)^2}$

$- \left(y_k - \frac{1}{2}\right)^2$.

$P_{k+1} = P_k + 2(x_k+1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$

If $y_{k+1} = y_k$

$P_{k+1} = P_k + 2(x_k+1) + 1 \quad P_k < 0$

If $y_{k+1} = y_k - 1$

$P_{k+1} = P_k + 2(x_k+1) + 1 - 2y_{k+1}$ (After solving) $\quad y_{k+1} \to y_k - 1$

Initial decision parameter:

We start from $(0, r)$.

From equation of P_k,

$P_0 = 1 + \left(r - \frac{1}{2}\right)^2 - r^2 = 1 - r + \frac{1}{4} = \boxed{\frac{5 - r}{4}} \approx 1 - r$ (To avoid floating point calculations)

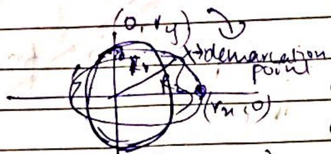
Stop when $x = y$

→ Incremental algorithm → 4 way symmetrical

Page No.
Date

Mid-point ellipse drawing algorithm:

Assumptions. Centre of ellipse is at origin.
Value of r_x (semi-major axis), r_y (semi-minor axis), centre of ellipse are needed.



Starting point : $(0, r_y)$ Final : $(r_x, 0)$

At point X, slope $= -1$ $dy/dx = -1$

Before X slope < -1 After X slope > -1

⇒ We cannot sample in one direction. $(r_x < r_y)$ Integer values Assumption

X → Last point of R1

Steps:

1. Start drawing the ellipse from $(0, r_y)$
2. Check everytime whether slope $= -1$ or not
3. In region R1 sampling is done in unit x direction.
4. " " R2 " " " " -ve " y

We'll be applying algo. in 1st quadrant.
1st quadrant divided into 2 parts < R1, R2 since slope varies in these regions.

Demarcation point when slope $= -1$

$f_{ellipse}(x, y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2$ $\quad \frac{x^2}{r_x^2} + \frac{y^2}{r_y^2} = 1$

$f(x, y) = 0$ ⇒ point is on the curve

$f(x, y) < 0$ ⇒ point is inside the curve
$f(x, y) > 0$ ⇒ " " outside " "

Differentiating $f(x, y)$

$\frac{dy}{dx} = -\frac{2r_y^2 x}{2r_x^2 y}$ At the boundary of R_1 and R_2 slope $= -1$ (At X)

$= -1$ magnitude

In R1, slope > -1 with sign <1, >1
R2, slope < -1

$\Rightarrow 2r_y^2 x = 2r_x^2 y$

When $2r_y^2 x \geq 2r_x^2 y$ ⇒ Moved to R_2 → Includes X.
$<$ ⇒ R1

Page No.
Date

At kth ~~pos~~ step, decision parameter of R1, $p1_k = f_{ellipse}\left(x_k + 1, y_k - \frac{1}{2}\right)$

Putting in eqn of ellipse; $P1_k$
$r_y^2(x_k + 1)^2 + r_x^2\left(y_k - \frac{1}{2}\right)^2 - r_x^2 r_y^2$

If $p1_k < 0$ ⇒ y_k.

$p_k 1 > 0$ ⇒ $y_k - 1$

$p1_{k+1} = f_{ellipse}\left(x_{k+1} + 1, y_{k+1} - 1/2\right)$

$= r_y^2 [(x_k + 1) + 1] + r_x^2\left(y_{k+1} - 1/2\right)^2 - \underbrace{r_x^2 r_y^2}_{\text{Replace from } p_k}$

$= p1_k + 2r_y^2(x_k + 1) + r_y^2 + r_x^2\left[\left(y_{k+1} - \frac{1}{2}\right)^2 - \left(y_k - \frac{1}{2}\right)^2\right]$

$p1_{k+1} = p1_k + 2r_y^2(x_k + 1) + r_y^2$ $\quad p1_k < 0$
$p1_{k+1} = p1_k + 2r_y^2(x_k + 1) + r_y^2 - 2r_x^2 y_{k+1}$ $\quad p1_k \geq 0$

$p1_0 = \cancel{\frac{r_x^2}{4} - r_x^2 r_y^2 = r_x^2\left(\frac{1}{4} - r_y^2\right)}$

$= r_y^2 + r_x^2\left(r_y - \frac{1}{2}\right)^2 - r_x^2 r_y^2$

$= r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$

Region R2: sampling in -ve unit y direction

$p^2_k = \left(x_k + \frac{1}{2}, y_k - 1\right)$

$= r_y^2\left(x_k + \frac{1}{2}\right)^2 + r_x^2(y_k - 1)^2 - r_x^2 r_y^2$

If $p^2_k > 0$ → mid point outside → x_k
$p^2_k < 0$ → " inside → $x_k + 1$

$p2_{k+1} = f_{ellipse}\left(x_{k+1} + 1/2,\ y_{k+1} - 1\right)$

$= r_y^2 \left(x_{k+1} + 1/2\right)^2 + r_x^2 \left((y_k - 1) - 1\right)^2 - r_x^2 r_y^2$

$p2_{k+1} = p2_k - 2r_x^2 (y_k - 1) + r_x^2 + r_y^2 \left[(x_{k+1} + 1/2)^2 - (x_k + 1/2)^2\right]$

If $x_{k+1} = x_k$

$p2_{k+1} = p2_k - 2r_x^2 (y_k - 1) + r_x^2$

$x_{k+1} = x_k + 1$

$p2_{k+1} = p2_k - 2r_x^2 (y_k - 1) + r_x^2 \ldots\ldots$

Calculation of $p2_0$:

On entering R2 initial position is taken as x_0, y_0 and that is the last position selected in R1.

$p2_0 = f_{ellipse}\left(x_0 + \frac{1}{2},\ y_0 - 1\right)$

$= r_y^2 (x_0 + 1/2)^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$

$r_x^2, r_y^2, 2r_x^2, 2r_y^2$ → calculated from before

~~Ant~~ 2-D Transformations: Types:

1. Translation
2. Rotation
3. Scaling
4. Reflection (Mirroring)
5. Shear &

Changes in orientation, shape & size are accomplished with geometric transformations that alter the coordinate descriptions of the objects.

Translation: It is applied to an object by repositioning the object along a straight line path from one co-ordinate to other.

$(x, y) \to (x', y')$

$x' = x + t_x \qquad y' = y + t_y$

(t_x, t_y) → translation vector / shift vector

$P = \begin{bmatrix} x \\ y \end{bmatrix} \qquad P' = \begin{bmatrix} x' \\ y' \end{bmatrix}$

$P' = P + T$

$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$ Matrix addition operation

1. No deformation of the object takes place during translation. Hence, it is called rigid transformation.

eg. Translating a line ← translate each & every point
translate the endpoints and re-draw the line between the 2 new endpoints.

Polygon ← Translate the vertices, redraw polygon.

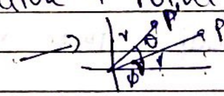
Curve − Translate the controlling points, redraw the curve
→ for circle centre & radius
ellipse → major minor axes, ~~is~~ centre

Rotation : We use polar co-ordinates.

A 2-D rotation is applied to an object by repositioning it along a circular path in ~~xy~~ plane.

Pivot point / Point of rotation → Point about which rotation is done.

2 types ← w.r.t origin
w.r.t pivot point ≠ origin



θ − angle of rotation.

$x = r\cos\phi$

$y = r\sin\phi$

$x' = r\cos(\theta + \phi)$
$y' = r\sin(\theta + \phi)$ } open

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

$y' = x\sin\theta + y\cos\theta$ moving counter-clockwise.

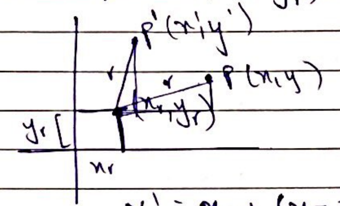
Matrix Form: $\curvearrowright R_\theta$

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

$P' = R_\theta P$

Rotation about point other than origin:

Pivot point : (x_r, y_r)



If we ~~put~~ move origin to pivot point, we need to subtract x_r, y_r from all points.

$x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$

$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$

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

$P' = R_\theta(P) + \text{translation}$.

Rotation comprises of multiplication of matrix
Translation " " addition " "
Rotation about pivot point = Rotation + Translation

Scaling: Transformation that alters the size of the object.

$x' = x s_x$ $\quad s_x \rightarrow$ scaling factor in x direction
$y' = y s_y$ $\quad s_y$ " " " y "

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$P' = S.P$

If $s_x < 1$, $s_y < 1$ reduction
$s_x > 1$, $s_y > 1$ enlargement
$s_x = 1$, $s_y = 1$ no change
Whenever $s_x = s_y \rightarrow$ uniform scaling
$s_x \neq s_y \rightarrow$ differential "

Objects are scaled as well as repositioned in scaling operation. $\rightarrow$ not necessary (if we keep one ~~fixed~~ point/control point fixed no repositioning.)

If we scale w.r.t a fixed point $x_f, y_f \neq$ origin:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x - x_f \\ y - y_f \end{bmatrix} + \begin{bmatrix} x_f \\ y_f \end{bmatrix}$$

$x' = x s_x + x_f(1 - s_x)$

$y' = y s_y + y_f(1 - s_y)$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_x \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_f(1 - s_x) \\ y_f(1 - s_y) \end{bmatrix}$$

$P' = SP + \text{translation}$

Homogenous Co-ordinates:

Translation

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

$x' = x + t_x$
$y' = y + t_y$
$1 = 1$

For inverse replace t_x by $-t_x$, t_y by $-t_y$

Rotation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Inverse: θ by $-\theta$

Scaling:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Inverse: S_x by $\frac{1}{S_x}$,

S_y by $\frac{1}{S_y}$

Composite Transformations: Sequence of transformations

Order of applying transformation matters!

Column major.

Multiplication / Application of transformations will be from right to left.

eg. 2 transformations T_1, T_2

$P' = T_2 . T_1 . P$ $\xleftarrow{\text{order}}$

$T' = T_2 . T_1$ → final transformation matrix.

successive translation operations are additive in nature.

" rotation " " " "

$P' = R_{\theta_2} R_{\theta_1} P = R_{(\theta_2+\theta_1)} P$

Scaling → not additive.

2 successive rotations are commutative.

2 successive translations → commutative

successive scaling → commutative.

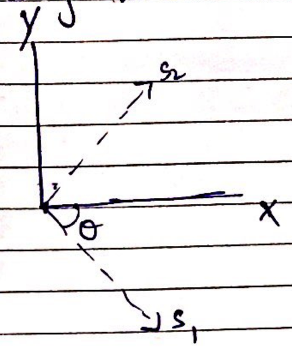
$$\underset{S_2}{\begin{bmatrix} S_{x_2} & 0 & 0 \\ 0 & S_{y_2} & 0 \\ 0 & 0 & 1 \end{bmatrix}} \underset{S_1}{\begin{bmatrix} S_{x_1} & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}}$$

$$S_2 . S_1 = \begin{bmatrix} S_{x_2} S_{x_1} & 0 & 0 \\ 0 & S_{y_2} S_{y_1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation & uniform scaling → commutative

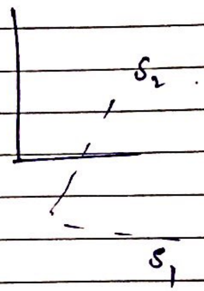
Pivot point in rotation is similar to fixed point in scaling.

Scaling w.r.t $S_1 S_2$ axes (Not = x or y axis).



Angle b/w S_1 & S_2 = 90°.

$P' = R_\theta S_2 R_\theta S_1$



Translate origin

Rotate axis

Operation

Inverse Rotate

Inverse Translate