

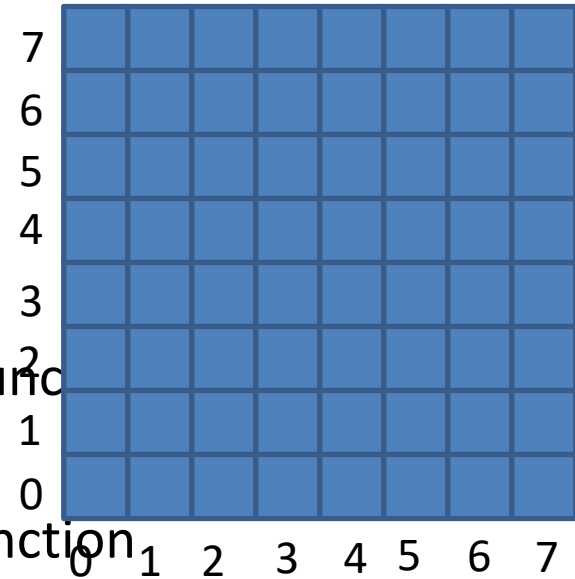
GRAPHICS PRIMITIVES

Outline

- Points
- Line drawing algorithms.
- Circle drawing algorithm.
- Ellipse drawing algorithm.
- Scan-Line polygon filling algorithm.
- Inside-Outside test.
- Boundary fill algorithm.
- Flood fill algorithm.
- Character generation.
- Line attributes.
- Color and grayscale levels
- Area fill attributes.
- Character attributes.

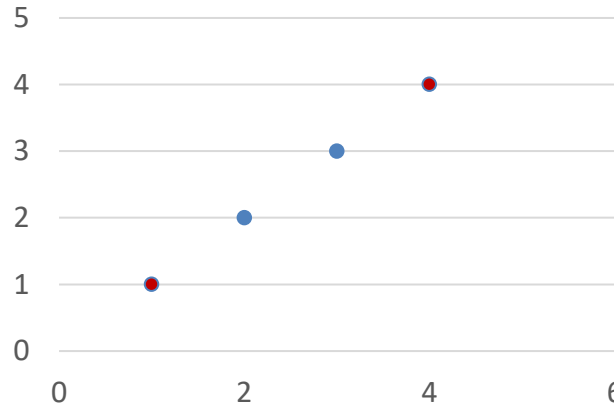
Point

- Point plotting is done by converting a single coordinate position furnished by an application program into appropriate operations for the output device in use.
- Example: Plot point $P(3, 2)$
 - Line $x = 3$
 - Line $y = 2$
 - Point $P(3, 2)$
- To draw the point on the screen we use function
 - ✓ $setpixel(x, y)$
- To draw the pixel in C language we use function
 - ✓ $putpixel(x, y, color)$
- Similarly for retrieving color of pixel we have function
 - ✓ $getpixel(x, y)$



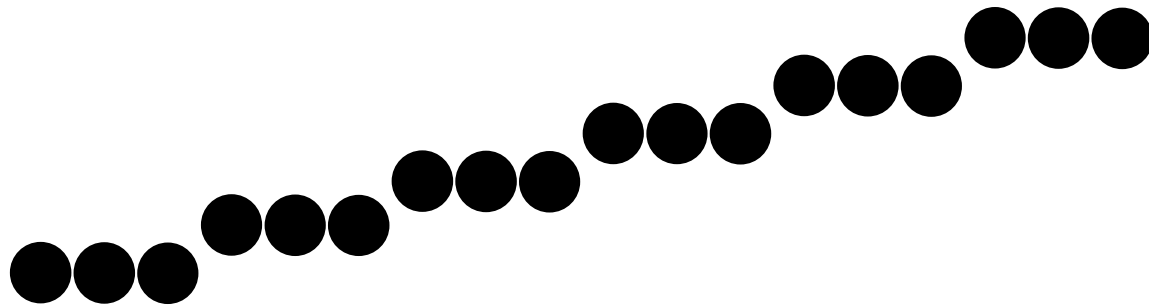
Line

- Line drawing is done by calculating intermediate positions along the line path between two specified endpoint positions.
- The output device is then directed to fill in those positions between the end points with some color.
- For some device straight line can be drawn by random scan display, a line end point to other.
- Digital devices display straight line by plotting discrete points between the two end points.
- Discrete coordinate positions along the line path are calculated from the equation of the line.



Contd.

- Screen locations are referenced with integer values.
- So plotted positions may only approximate actual line positions between two specified endpoints. For example line position of $(12.36, 23.87)$ would be converted to pixel position $(12, 24)$.
- This rounding of coordinate values to integers causes lines to be displayed with a stair step appearance (“the **Jaggies**”).



Line Drawing Algorithms

- The Cartesian slop-intercept equation for a straight line is
 - $y = mx + b$
 - with ' m ' representing slop and ' b ' as the intercept.
- It is possible to draw line using this equation but for efficiency purpose we use different line drawing algorithm.
 - DDA Algorithm
 - Bresenham's Line Algorithm
- We can also use this algorithm in parallel if we have more number of processors.

Introduction to DDA Algorithm

- Full form of DDA is Digital Differential Analyzer
- DDA is scan conversion line drawing algorithm based on calculating either Δy or Δx using line equation.
- We sample the line at unit intervals in one coordinate and find corresponding integer values nearest the line path for the other coordinate.
- Selecting unit interval in either x or y direction based on way we process line.

Unit Step Direction in DDA Algorithm

- Processing from left to right.

- ✓ Slope is "+ve", & Magnitude is Less than 1

- ✓ Slope is "-ve", & Magnitude is Less than 1

$$\Delta X = 1$$

- ✓ Slope is "+ve", & Magnitude is greater than 1

$$\Delta Y = 1$$

- ✓ Slope is "-ve", & Magnitude is greater than 1

$$\Delta Y = -1$$

- Processing from right to left.

- ✓ Slope is "+ve", & Magnitude is Less than 1

- ✓ Slope is "-ve", & Magnitude is Less than 1

$$\Delta X = -1$$

- ✓ Slope is "+ve", & Magnitude is greater than 1

$$\Delta Y = -1$$

- ✓ Slope is "-ve", & Magnitude is greater than 1

$$\Delta Y = 1$$

Derivation of DDA Algorithm

- We sample at unit x interval ($\Delta x = 1$) and calculate each successive y value as follow:
- $y = mx + b$
- $y_1 = m(x + 1) + b$ [For first intermediate point]
- $y_k = m(x + k) + b$ [For k^{th} intermediate point]
- $y_{k+1} = m(x + k + 1) + b$ [For $k + 1^{\text{th}}$ intermediate point]
- Subtract y_k from y_{k+1}
- $y_{k+1} - y_k = m(x + k + 1) + b - m(x + k) - b$
- $y_{k+1} = y_k + m$
- Using this equation computation becomes faster than normal line equation.
- As m is any real value calculated y value must be rounded to nearest integer.

Contd.

- We sample at unit y interval ($\Delta y = 1$) and calculate each successive x value as follow:
- $x = (y - b)/m$
- $x_1 = ((y+1) - b)/m$ [For first intermediate point]
- $x_k = ((y+k) - b)/m$ [For k^{th} intermediate point]
- $x_{k+1} = ((y+k+1) - b)/m$ [For $k + 1^{\text{th}}$ intermediate point]

Subtract x_k from x_{k+1}

- $x_{k+1} - x_k = \{((y+k+1) - b)/m\} - \{((y+k) - b)/m\}$
- $x_{k+1} = x_k + 1/m$

Similarly

- for $\Delta x = -1$: we obtain $y_{k+1} = y_k - m$
- for $\Delta y = -1$: we obtain $x_{k+1} = x_k - 1/m$

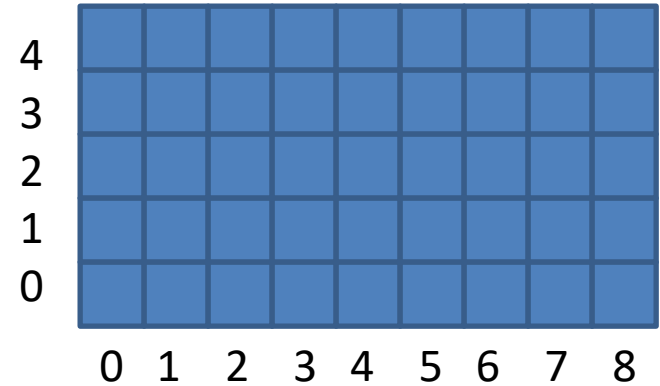
Procedure for DDA line algorithm.

```
Void lineDDA (int xa, int ya, int xb, int yb)
{
    int dx = xb - xa, dy = yb - ya, steps, k;
    float xincrement, yincrement, x = xa, y = ya;
    if (abs(dx)>abs(dy))
    {
        Steps = abs (dx);
    }
    else
    {
        Steps = abs (dy);
    }
    xincrement = dx/(float) steps;
    yincrement = dy/(float) steps;

    setpixel (ROUND (x), ROUND (y));
    for(k=0;k<steps;k++)
    {
        x += xincrement;
        y += yincrement;
        setpixel (ROUND (x), ROUND (y));
    }
}
```

Example DDA Algorithm

- Example: Draw line AB with coordinates $A(2,2)$, $B(6,4)$.
- $m = \frac{4-2}{6-2} = \frac{2}{4} = \frac{1}{2}$
- m is “+ve” and less than 1 so $\Delta x = 1$.
- $x_0 = 2, y_0 = 2$
[Plot the initial point as given]
- $x_1 = 3, y_1 = y_0 + m = 2 + 0.5 = 2.5$
[Plot the first intermediate point by rounding it to $(3, 3)$]
- $x_2 = 4, y_2 = y_1 + m = 2.5 + 0.5 = 3$
[Plot the second intermediate point by rounding it to $(4, 3)$]
- $x_3 = 5, y_3 = y_2 + m = 3 + 0.5 = 3.5$
[Plot the third intermediate point by rounding it to $(5, 4)$]
- $x_4 = 6, y_4 = 4$
[Plot End point given]



DDA Algorithm

▪ **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.

Introduction to Bresenham's Line Algorithm

- An accurate and efficient raster line-generating algorithm, developed by Bresenham.
- It scan converts line using only incremental integer calculations.
- That can be modified to display circles and other curves.
- Based on slope we take unit step in one direction and decide pixel of other direction from two candidate pixel.
- If $|\Delta x| > |\Delta y|$ we sample at unit x interval and vice versa.

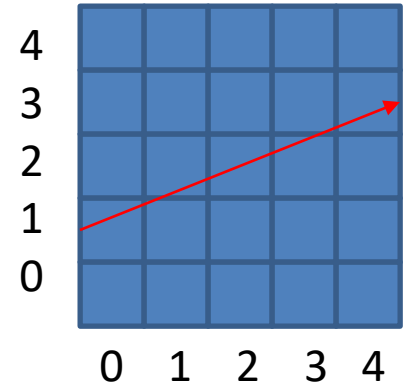
Line Path & Candidate pixel

- Example $|\Delta X| > |\Delta Y|$, and ΔY is “+ve”.

- Initial point (X_k, Y_k)

- Line path

- Candidate pixels $\{(X_k+1, Y_k), (X_k+1, Y_k+1)\}$



- Now we need to decide which candidate pixel is more closer to actual line.
- For that we use decision parameter (P_k) equation.
- Decision parameter can be derived by calculating distance of actual line from two candidate pixel.

Derivation Bresenham's Line Algorithm

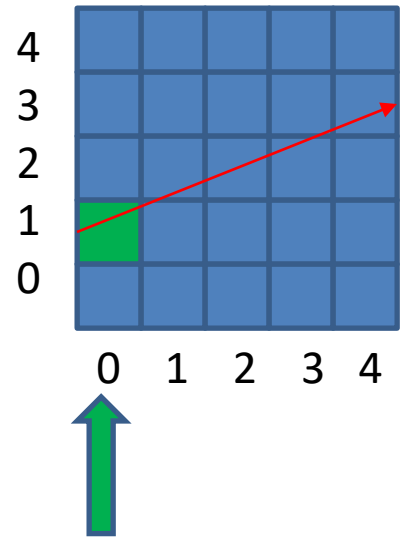
- $y = mx + b$ [Line Equation]
- $y = m(x_k) + b$ [Actual Y value at X_k position]
- $y = m(x_k + 1) + b$ [Actual Y value at $X_k + 1$ position]

Distance between actual line position and lower candidate pixel

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

Distance between actual line position and upper candidate pixel

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



Contd.

Calculate $d_1 - d_2$

- $d_1 - d_2 = \{m(x_k + 1) + b - y_k\} - \{(y_k + 1) - m(x_k + 1) - b\}$
- $d_1 - d_2 = \{mx_k + m + b - y_k\} - \{y_k + 1 - mx_k - m - b\}$
- $d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2b - 1$
- $d_1 - d_2 = 2(\Delta y/\Delta x)(x_k + 1) - 2y_k + 2b - 1$ [Put $m = \Delta y/\Delta x$]

Decision parameter

- $p_k = \Delta x(d_1 - d_2)$
- $p_k = \Delta x \{2(\Delta y/\Delta x)(x_k + 1) - 2y_k + 2b - 1\}$
- $p_k = 2\Delta yx_k - 2\Delta xy_k + 2\Delta y + 2\Delta xb - \Delta x$
- $p_k = 2\Delta yx_k - 2\Delta xy_k + C$ [Replacing single constant C for simplicity]

Similarly

- $p_{k+1} = 2\Delta yx_{k+1} - 2\Delta xy_{k+1} + C$

Contd.

Subtract p_k from p_{k+1}

- $p_{k+1} - p_k = 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + C - 2\Delta y x_k + 2\Delta x y_k - C$
- $p_{k+1} - p_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$
[where $(x_{k+1} - x_k) = 1$]
- $p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$
[where $(y_{k+1} - y_k) = 0$ or 1 depending on selection of previous pixel]

Initial Decision parameter

- The first decision parameter p_0 is calculated using equation of p_k .
- $p_k = 2\Delta y x_k - 2\Delta x y_k + 2\Delta y + 2\Delta x b - \Delta x$
- $p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2\Delta x b - \Delta x$ [Put $k = 0$]
- $p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2\Delta x (y_0 - mx_0) - \Delta x$
[Substitute $b = y_0 - mx_0$]
- $p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2\Delta x (y_0 - (\Delta y / \Delta x) x_0) - \Delta x$
[Substitute $m = \Delta y / \Delta x$]
- $p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2\Delta x y_0 - 2\Delta y x_0 - \Delta x$
- $p_0 = 2\Delta y - \Delta x$
[Initial decision parameter with all terms are constant]

Bresenham's Line Algorithm

1. Input the two line endpoints and store the left endpoint in (x_0, y_0) .
2. Load (x_0, y_0) into the frame buffer; that is, plot the first point.
3. Calculate constants Δx , Δy , $2\Delta y$, and $2\Delta y - 2\Delta x$, and obtain the starting value for the 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 $(x_k + 1, y_k + 1)$ and

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

5. Repeat step-4 Δx times.

Description of Bresenham's Line

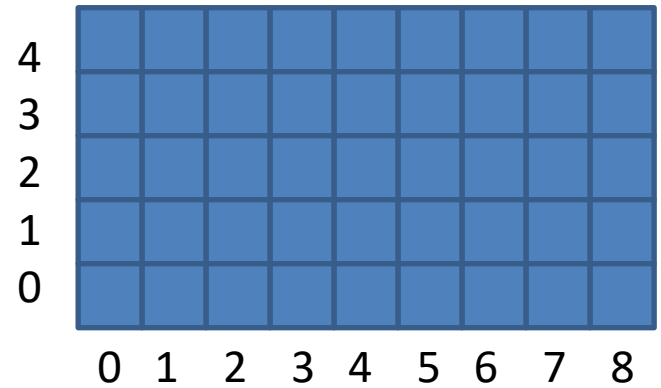
Algorithm

- Bresenham's algorithm is generalized to lines with arbitrary slope.
- For lines with positive slope greater than 1 we interchange the roles of the x and y directions.
- Also we can revise algorithm to draw line from right endpoint to left endpoint, both x and y decrease as we step from right to left.
- When $p_k = 0$ we can choose either lower or upper pixel but same for whole line.
- For the negative slope the procedure are similar except that now one coordinate decreases as the other increases.
- The special case handle separately by loading directly into the frame buffer without processing.
 - ✓ Horizontal line ($\Delta y = 0$),
 - ✓ Vertical line ($\Delta x = 0$)
 - ✓ Diagonal line with $|\Delta x| = |\Delta y|$

Example Bresenham's Line

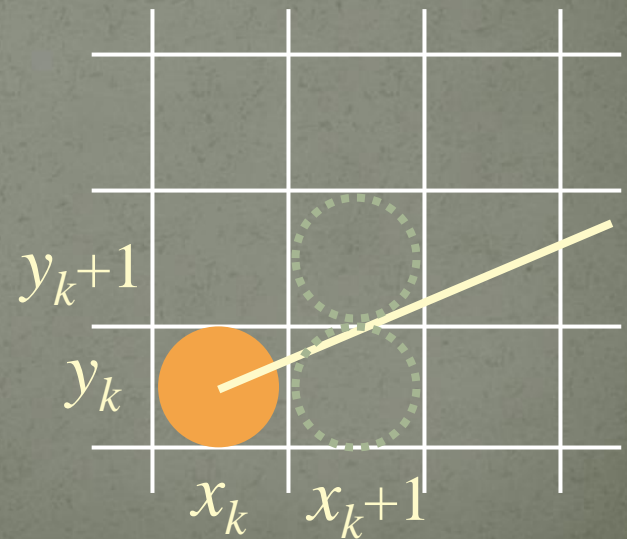
Algorithm

- Example: Draw line AB with coordinates $A(2,2)$, $B(6,4)$.
- Plot left end point $A(2, 2)$.
- Calculate:
 - $\Delta x = 4$,
 - $\Delta y = 2$,
 - $2\Delta y = 4$,
 - $2\Delta y - 2\Delta x = -4$,
- $p_0 = 2\Delta y - \Delta x = 0$
- Now $p_0 \neq 0$ so we select upper pixel $(3, 3)$.
- $p_1 = p_0 + 2\Delta y - 2\Delta x = 0 + (-4) = -4$
- Now $p_1 < 0$ so we select lower pixel $(4, 3)$.
- $p_2 = p_1 + 2\Delta y = -4 + (4) = 0$
- Now $p_2 \neq 0$ so we select upper pixel $(5, 4)$.
- $p_3 = p_2 + 2\Delta y - 2\Delta x = 0 + (-4) = -4$
- Now $p_3 < 0$ so we select lower pixel $(6, 4)$.



Bresenham's Line Algorithm

- Accurate and Efficient
 - Use only incremental integer calculations
 - Test the sign of an integer parameter
- Case) Positive Slope Less Than 1
 - After the pixel (x_k, y_k) is displayed, next which pixel is decided to plot in column x_{k+1} ?
 - (x_{k+1}, y_k) or (x_{k+1}, y_{k+1})



Bresenham's Algorithm(cont.)

- Case) Positive Slope Less Than 1

- y at sampling position x_k

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

- Difference

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

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

$$d_1 - d_2 < 0 \rightarrow (x_k + 1, y_k)$$

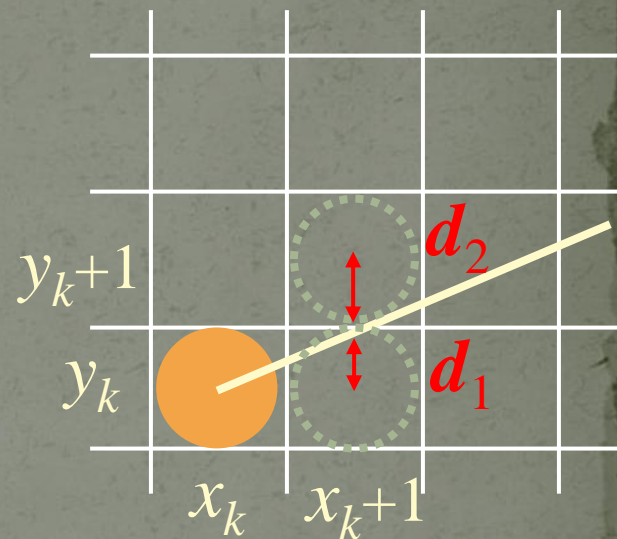
$$d_1 - d_2 > 0 \rightarrow (x_k + 1, y_k + 1)$$

- Decision parameter

$$p_k = \Delta x(d_1 - d_2)$$

$$= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + 2\Delta y + \Delta x(2b - 1)$$

$$= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$$



Bresenham's Algorithm(cont.)

- Case) Positive Slope Less Than 1

- Decision parameter

$$\begin{aligned} p_{k+1} - p_k &= (2\Delta y \cdot x_{k+1} - 2\Delta x \cdot y_{k+1} + c) - (2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c) \\ &= 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k) \end{aligned}$$

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

- Decision parameter of a starting pixel (x_0, y_0)

$$\begin{aligned} p_0 &= 2\Delta y \cdot x_0 - 2\Delta x \cdot y_0 + 2\Delta y + \Delta x(2b - 1) \\ &= 2\Delta y \cdot x_0 - 2\Delta x \cdot (mx_0 + b) + 2\Delta y + \Delta x(2b - 1) \\ &= 2\Delta y \cdot x_0 - 2\Delta y \cdot x_0 - 2b\Delta x + 2\Delta y + 2b\Delta x - \Delta x \end{aligned}$$

$$\therefore p_0 = 2\Delta y - \Delta x$$

Bresenham's Algorithm(cont.) for $m < 1$

- Algorithm for $0 < m < 1$
 - Input the two line endpoints and store the left end point in (x_0, y_0)
 - Load (x_0, y_0) into the frame buffer; that is, plot the first point
 - Calculate constants Δx , Δy , $2\Delta y$, and $2\Delta y - 2\Delta x$, and obtain the starting value for the decision parameter as

$$p_0 = 2\Delta y - \Delta x$$

- At each x_k along the line, start 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}) and

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

- Repeat step 4 Δx times

Generalized Bresenham Algorithm

```
void bres_general(int x1, int y1, int x2, int y2)  
{  
  int dx, dy, x, y, d, s1, s2, swap=0, temp;  
  dx = abs(x2 - x1);  
  dy = abs(y2 - y1);  
  s1 = sign(x2-x1);  
  s2 = sign(y2-y1);  
  /* Check if dx or dy has a greater range */  
  /* if dy has a greater range than dx swap dx and dy */  
  if(dy > dx){temp = dx; dx = dy; dy = temp; swap = 1;}
```

```
/* Set the initial decision  
parameter and the initial  
point */
```

```
d = 2 * dy - dx;
```

```
x = x1;
```

```
y = y1;
```

```
int i;
```

```
for(i = 1; i <= dx; i++)
```

```
{
```

```
    putpixel(x,y,WHITE);
```

```
while(d >= 0)
```

```
{
```

```
    if(swap) x = x + s1;
```

```
    else
```

```
        y = y + s2;
```

```
    d = d - 2 * dx;
```

```
}
```

```
if(swap) y = y + s2;
```

```
else x = x + s1;
```

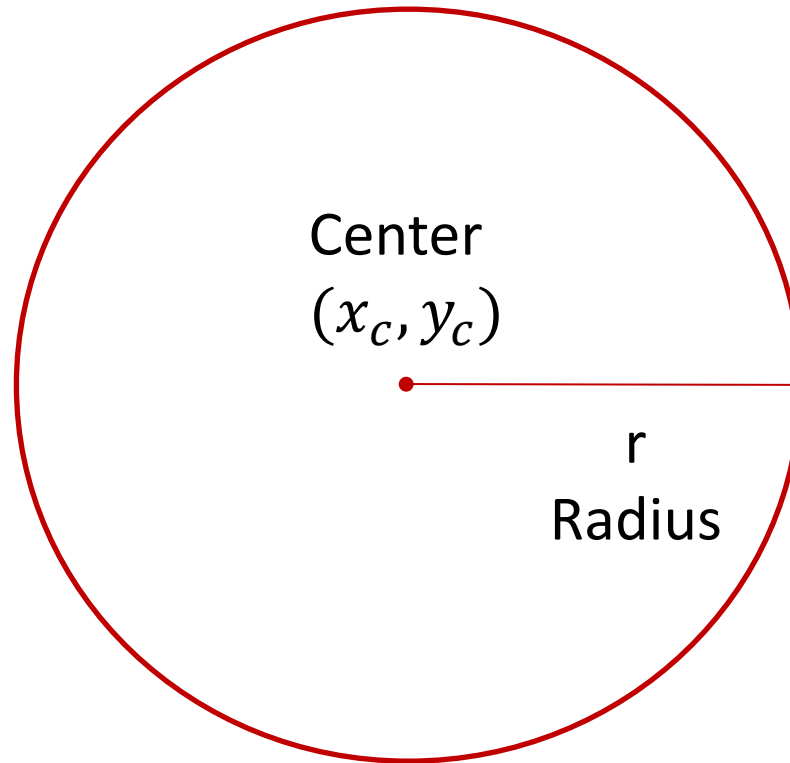
```
d = d + 2 * dy;
```

```
}
```

```
}
```


Circle

- A circle is defined as the set of points that are all at a given distance r from a center position say (x_c, y_c) .

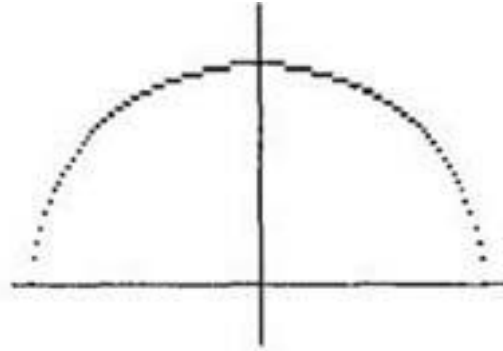


Properties of Circle- Cartesian Coordinate

- Cartesian coordinates equation :
- $(x - x_c)^2 + (y - y_c)^2 = r^2$
- We could use this equation to calculate circular boundary points.
- We increment 1 in x direction in every steps from $x_c - r$ to $x_c + r$ and calculate corresponding y values at each position as:
 - $(x - x_c)^2 + (y - y_c)^2 = r^2$
 - $(y - y_c)^2 = r^2 - (x - x_c)^2$
 - $(y - y_c) = \pm \sqrt{r^2 - (x_c - x)^2}$
 - $y = y_c \pm \sqrt{r^2 - (x_c - x)^2}$

Contd.

- But this is not best method as it requires more number of calculations which take more time to execute.
- And also spacing between the plotted pixel positions is not uniform.



- We can adjust spacing by stepping through y values and calculating x values whenever the absolute value of the slope of the circle is greater than 1.
- But it will increase computation time.

Properties of Circle- Polar Coordinate

- Another way to eliminate the non-uniform spacing is to draw circle using polar coordinates 'r' and 'θ'.
- Calculating circle boundary using polar equation is given by pair of equations which is as follows.

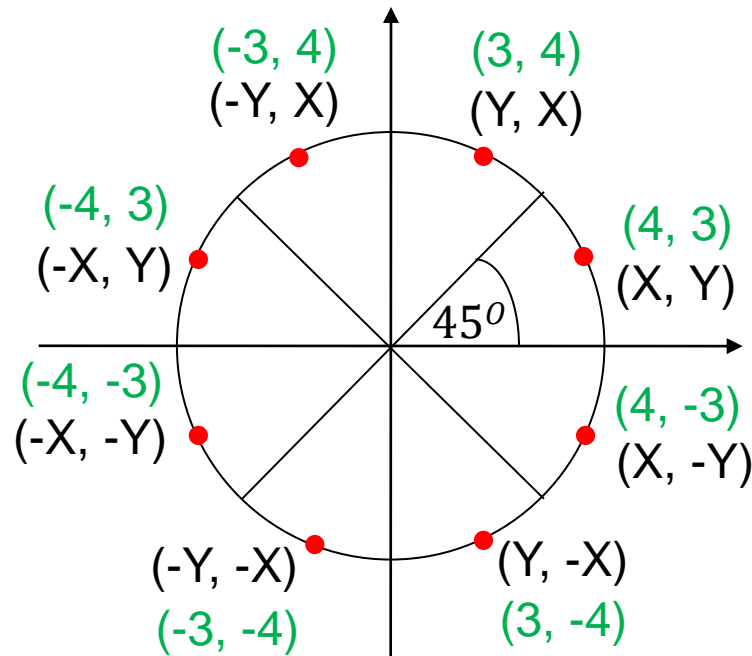
$$x = x_c + r \cos \theta$$

$$y = y_c + r \sin \theta$$

- When display is produce using these equations using fixed angular step size circle is plotted with uniform spacing.
- The step size 'θ' is chosen according to application and display device.
- For a more continuous boundary on a raster display we can set the step size at $\frac{1}{r}$.

Properties of Circle- Symmetry

- Computation can be reduced by considering symmetry city property of circles.
- The shape of circle is similar in each octant.



Circle Algorithm

- Taking advantage of this symmetry property of circle we can generate all pixel position on boundary of circle by calculating only one sector from $x = 0$ to $x = y$.
- Determining pixel position along circumference of circle using any of two equations shown above still required large computation.
- More efficient circle algorithm are based on incremental calculation of decision parameters, as in the Bresenham line algorithm.
- Bresenham's line algorithm can be adapted to circle generation by setting decision parameter for finding closest pixel to the circumference at each sampling step.

Contd.

- A method for direct distance comparison to test the midpoint between two pixels to determine if this midpoint is inside or outside the circle boundary.
- This method is easily applied to other conics also.
- Midpoint approach generates same pixel position as generated by bresenham's circle algorithm.
- The error involve in locating pixel positions along any conic section using midpoint test is limited to one-half the pixel separation.

Introduction to Midpoint Circle

Algorithm

- In this we sample at unit interval and determine the closest pixel position to the specified circle path at each step.
- Given radius r and center (x_c, y_c)
- We first setup our algorithm to calculate circular path coordinates for center $(0, 0)$.
- And then we will transfer calculated pixel position to center (x_c, y_c) by adding x_c to x and y_c to y .
- Along the circle section from $x = 0$ to $x = y$ in the first quadrant, the slope of the curve varies from 0 to -1 .
- So we can step unit step in positive x direction over this octant and use a decision parameter to determine which of the two possible y position is closer to the circular path.

Decision Parameter Midpoint Circle Algorithm

- Position in the other seven octants are then obtain by symmetry.
- For the decision parameter we use the circle function which is:

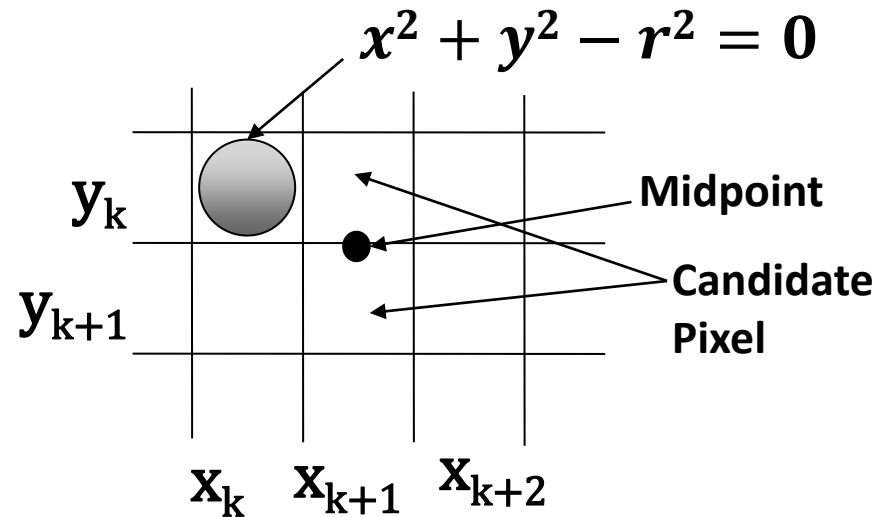
$$f_{circle}(x, y) = x^2 + y^2 - r^2$$

$$f_{circle}(x, y) \begin{cases} < 0 & \text{if } (x, y) \text{ is inside the circle boundary} \\ = 0 & \text{if } (x, y) \text{ is on the circle boundary} \\ > 0 & \text{if } (x, y) \text{ is outside the circle boundary} \end{cases}$$

- Above equation we calculate for the mid positions between pixels near the circular path at each sampling step.
- And we setup incremental calculation for this function as we did in the line algorithm.

Midpoint between Candidate pixel

- Figure shows the midpoint between the two candidate pixels at sampling position $x_k + 1$.



- Assuming we have just plotted the pixel at (x_k, y_k) .
- Next we determine whether the pixel at position $(x_k + 1, y_k)$ or the one at position $(x_k + 1, y_k - 1)$ is closer to circle boundary.

Derivation Midpoint Circle

Algorithm

- So for finding which pixel is more closer using decision parameter evaluated at the midpoint between two candidate pixels as below:
- $p_k = f_{circle}(x_k + 1, y_k - \frac{1}{2})$
- $p_k = (x_k + 1)^2 + (y_k - \frac{1}{2})^2 - r^2$
- If $p_k < 0$, midpoint is inside the circle and the pixel on the scan line y_k is closer to circle boundary.
- Otherwise midpoint is outside or on the boundary and we select the scan line $y_k - 1$.

Contd.

- Successive decision parameters are obtained using incremental calculations as follows:

- $p_{k+1} = f_{circle}(x_{k+1} + 1, y_{k+1} - \frac{1}{2})$

- $p_{k+1} = [(x_k + 1) + 1]^2 + (y_{k+1} - \frac{1}{2})^2 - r^2$

- Now we can obtain recursive calculation using equation of p_{k+1} and p_k as follows

- $p_{k+1} - p_k = \left([(x_k + 1) + 1]^2 + (y_{k+1} - \frac{1}{2})^2 - r^2 \right) - \left((x_k + 1)^2 + (y_k - \frac{1}{2})^2 - r^2 \right)$

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

Contd.

- $p_{k+1} - p_k = (x_k + 1)^2 + 2(x_k + 1) + 1 + y_{k+1}^2 - y_{k+1} + \frac{1}{4} - r^2 - (x_k + 1)^2 - y_k^2 + y_k - \frac{1}{4} + r^2$
- $p_{k+1} - p_k = 2(x_k + 1) + 1 + y_{k+1}^2 - y_{k+1} - y_k^2 + y_k$
- $p_{k+1} - p_k = 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$
- $p_{k+1} = p_k + \mathbf{2(x_k + 1)} + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$
- **Now we can put $2x_{k+1} = 2(x_k + 1)$**
- $p_{k+1} = p_k + \mathbf{2x_{k+1}} + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + \mathbf{1}$

Contd.

- $p_{k+1} = p_k + 2x_{k+1} + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$
- In above equation y_{k+1} is either y_k or $y_k - 1$ depending on the sign of the p_k .
- If we select $y_{k+1} = y_k$.
- $p_{k+1} = p_k + 2x_{k+1} + 1$ OR
- $p_{k+1} = p_k + 2x_k + 3$ where $2x_{k+1} = 2x_k + 2$
- If we select $y_{k+1} = y_k - 1$.
- $p_{k+1} = p_k + 2x_{k+1} + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$
- $p_{k+1} = p_k + 2x_{k+1} + (y_{k+1} + y_k)(y_{k+1} - y_k) - (y_{k+1} - y_k) + 1$
- $p_{k+1} = p_k + 2x_{k+1} + (y_k - 1 + y_k)(y_k - 1 - y_k) - (y_k - 1 - y_k) + 1$
- $p_{k+1} = p_k + 2x_{k+1} + (2y_k - 1)(-1) - (-1) + 1$

Contd.

- $p_{k+1} = p_k + 2x_{k+1} - 2y_k + 1 + 1 + 1$
- $p_{k+1} = p_k + 2x_{k+1} - (2y_k - 2) + 1$
- Now put $2y_{k+1} = 2y_k - 2$.
- $p_{k+1} = p_k + 2x_{k+1} - 2y_{k+1} + 1$
- OR
- $p_{k+1} = p_k + 2x_k + 2 - 2y_k + 2 + 1$
- $p_{k+1} = p_k + 2x_k - 2y_k + 5$
where $2x_{k+1} = 2x_k + 2$, $2y_{k+1} = 2y_k - 2$

IDP Midpoint Circle Algorithm

- The initial decision parameter(IDP) is obtained by evaluating the circle function at the start position $(x_0, y_0) = (0, r)$ as follows.
- $p_0 = f_{circle}(0 + 1, r - \frac{1}{2})$
- $p_0 = 1^2 + (r - \frac{1}{2})^2 - r^2$
- $p_0 = 1 + r^2 - r + \frac{1}{4} - r^2$
- $p_0 = \frac{5}{4} - r$
- $p_0 \approx 1 - r$

Algorithm for Midpoint Circle Generation

1. Input radius r and circle center (x_c, y_c) , and obtain the first point on the circumference of a circle centered on the origin as

$$(x_0, y_0) = (0, r)$$

2. calculate the initial value of the decision parameter as

$$p_0 = \frac{5}{4} - r$$

3. At each x_k position, starting at $k = 0$, perform the following test:

If $p_k < 0$, the next point along the circle centered on $(0, 0)$ is $(x_k + 1, y_k)$ &

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

Otherwise, the next point along the circle is $(x_k + 1, y_k - 1)$ &

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$$

$$\text{Where } 2x_{k+1} = 2x_k + 2, \text{ \& } 2y_{k+1} = 2y_k - 2.$$

4. Determine symmetry points in the other seven octants.
5. Move each calculated pixel position (x, y) onto the circular path centered on (x_c, y_c) and plot the coordinate values:

$$x = x + x_c, y = y + y_c$$

6. Repeat steps 3 through 5 until $x \geq y$.

Contd.

- $p_5 = p_4 + 2x_5 + 1$
- $p_5 = -3 + 10 + 1 = 8$
- Now $p_5 \not\leq 0$ we select (6, 8)
- $p_6 = p_5 + 2x_6 + 1 - 2y_6$
- $p_6 = 8 + 12 + 1 - 16 = 5$
- Now $p_6 \not\leq 0$ we select (7, 7)
- Now Loop exit as $x \geq y$, as
- in our case $7 \geq 7$

k	p_k	(x_{k+1}, y_{k+1})
0	-9	(1, 10)
1	-6	(2, 10)
2	-1	(3, 10)
3	6	(4, 9)
4	-3	(5, 9)

Contd.

- Than we calculate pixel position for given center (1,1) using equations:

- $x = x + x_c = x + 1$

- $y = y + y_c = y + 1$

Center (0, 0)	Center (1, 1)
(1, 10)	(2, 11)
(2, 10)	(3, 11)
(3, 10)	(4, 11)
(4, 9)	(5, 10)
(5, 9)	(6, 10)
(6, 8)	(7, 9)
(7, 7)	(8, 8)

Contd.

- Plot the pixel.
- First plot initial point.

(1, 11)

Center (1, 1)

(2, 11)

(3, 11)

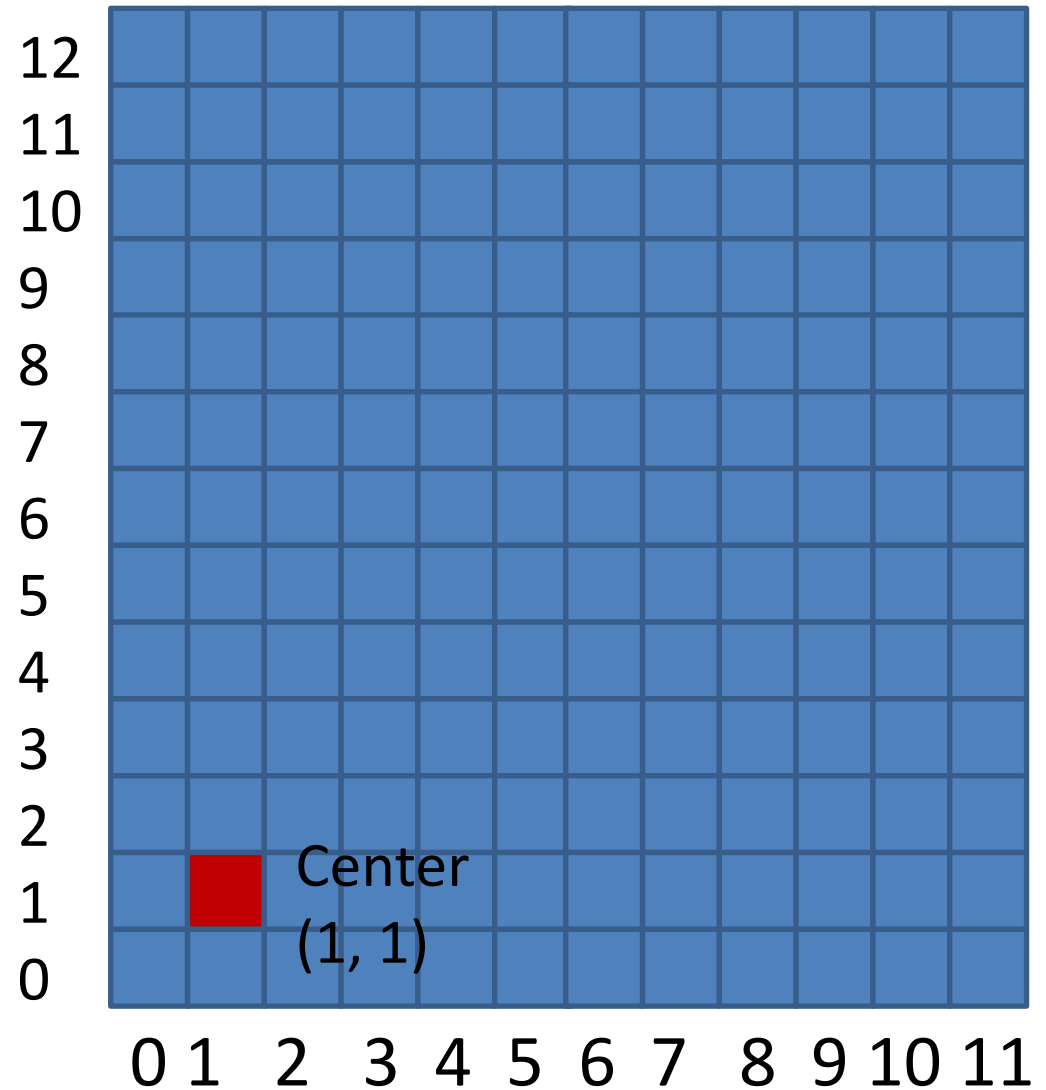
(4, 11)

(5, 10)

(6, 10)

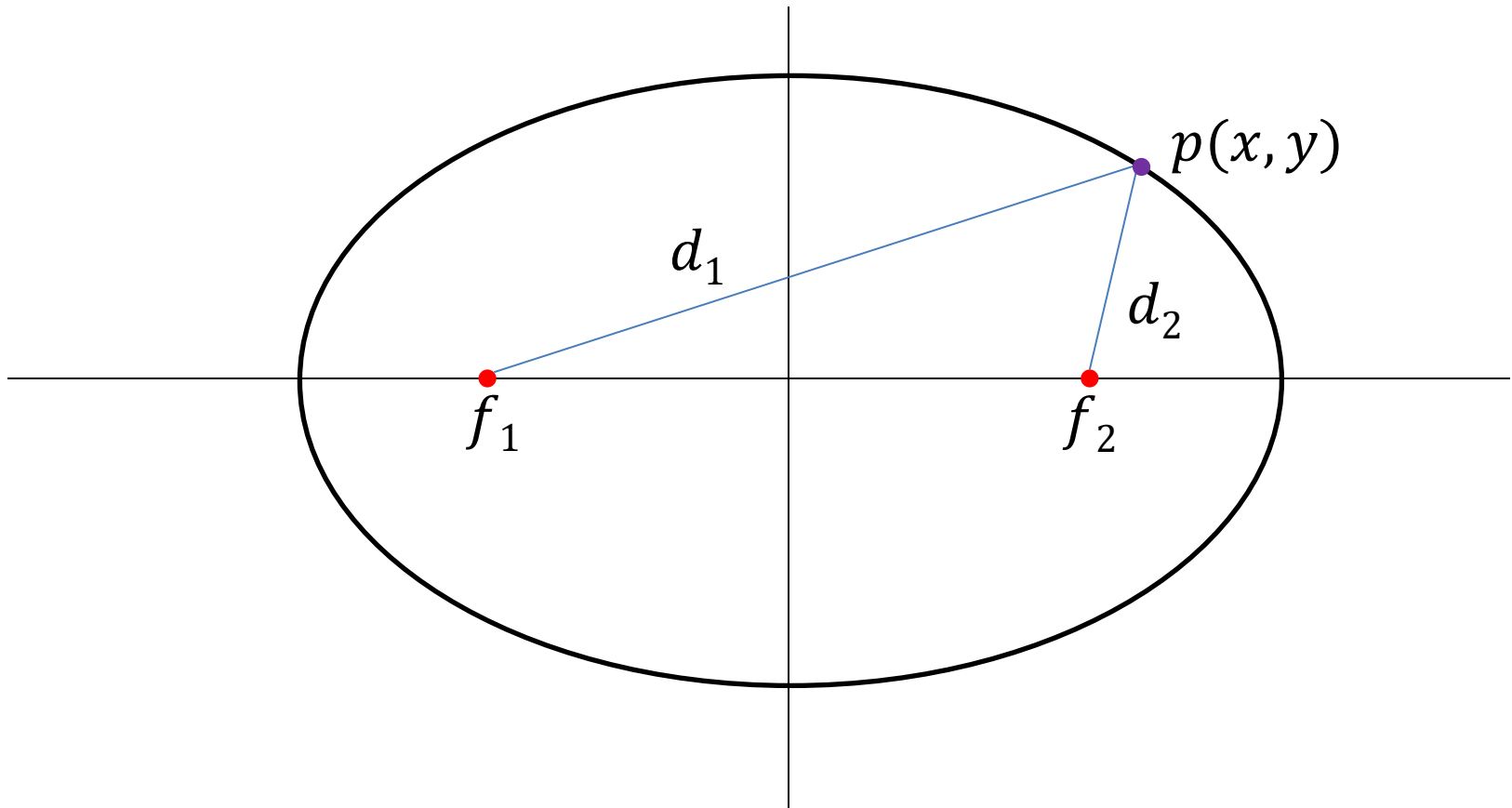
(7, 9)

(8, 8)



Ellipse

- AN ellipse is defined as the set of points such that the sum of the distances from two fixed positions (**foci**) is same for all points.



Contd.

- If we labeled distance from two foci to any point on ellipse boundary as d_1 and d_2 then the general equation of an ellipse can be written as:

$$d_1 + d_2 = \textit{Constant}$$

- Expressing distance in terms of focal coordinates $f_1 = (x_1, y_1)$ and $f_2 = (x_2, y_2)$ we have
- $\sqrt{(x - x_1)^2 + (y - y_1)^2} + \sqrt{(x - x_2)^2 + (y - y_2)^2} = \textit{Constant}$
[Using Distance formula]

Properties of Ellipse-Specifying Equations

- An interactive method for specifying an ellipse in an arbitrary orientation is to input
 - ✓ two foci and
 - ✓ a point on the ellipse boundary.

- With this three coordinates we can evaluate constant in equation:

$$\sqrt{(x - x_1)^2 + (y - y_1)^2} + \sqrt{(x - x_2)^2 + (y - y_2)^2} = \textit{Constant}$$

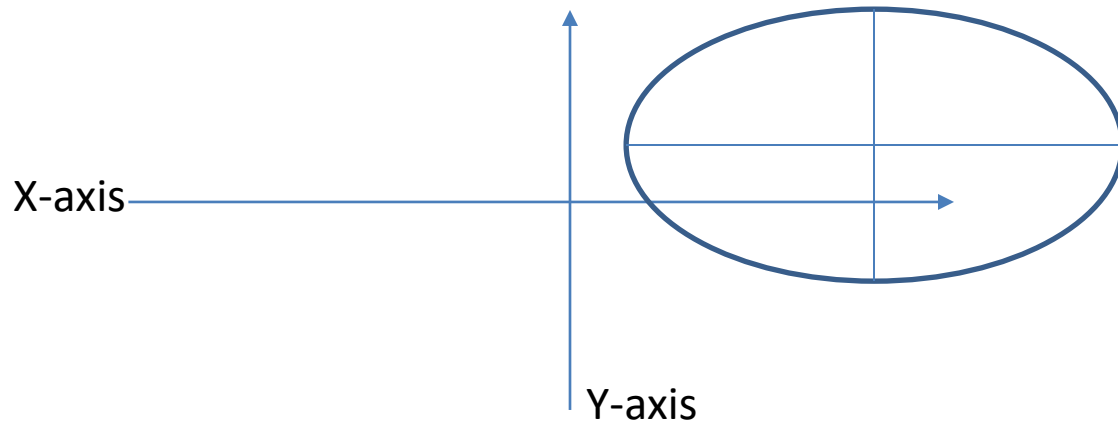
- We can also write this equation in the form

$$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$$

- Where the coefficients $A, B, C, D, E,$ and F are evaluated in terms of the focal coordinates and the dimensions of the major and minor axes of the ellipse.

Contd.

- Then coefficient in $Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$ can be evaluated and used to generate pixels along the elliptical path.
- We can say ellipse is in standard position if their major and minor axes are parallel to x-axis and y-axis.
- Ellipse equation are greatly simplified if we align major and minor axis with coordinate axes i.e. x-axis and y-axis.



Contd.

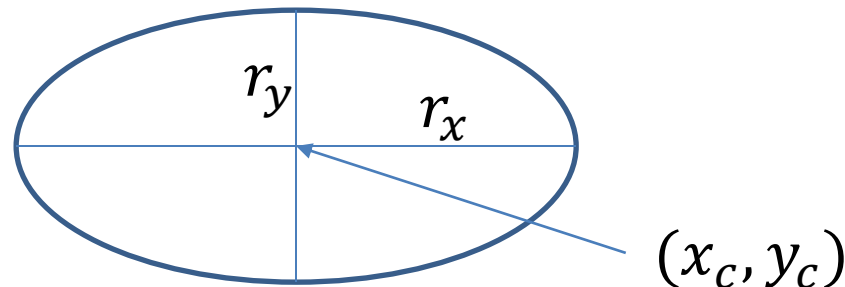
- Equation of ellipse can be written in terms of the ellipse center coordinates (x_c, y_c) and parameters r_x and r_y as.

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

- Using the polar coordinates r and θ , we can also describe the ellipse in standard position with the parametric equations:

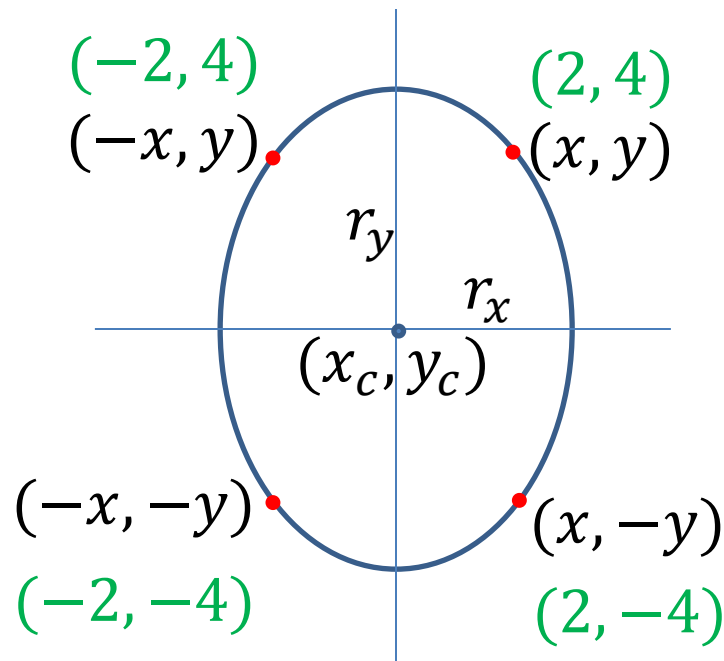
$$x = x_c + r_x \cos \theta$$

$$y = y_c + r_y \sin \theta$$



Properties of Ellipse-Symmetry

- Symmetry property further reduced computations.
- An ellipse in standard position is symmetric between quadrant.



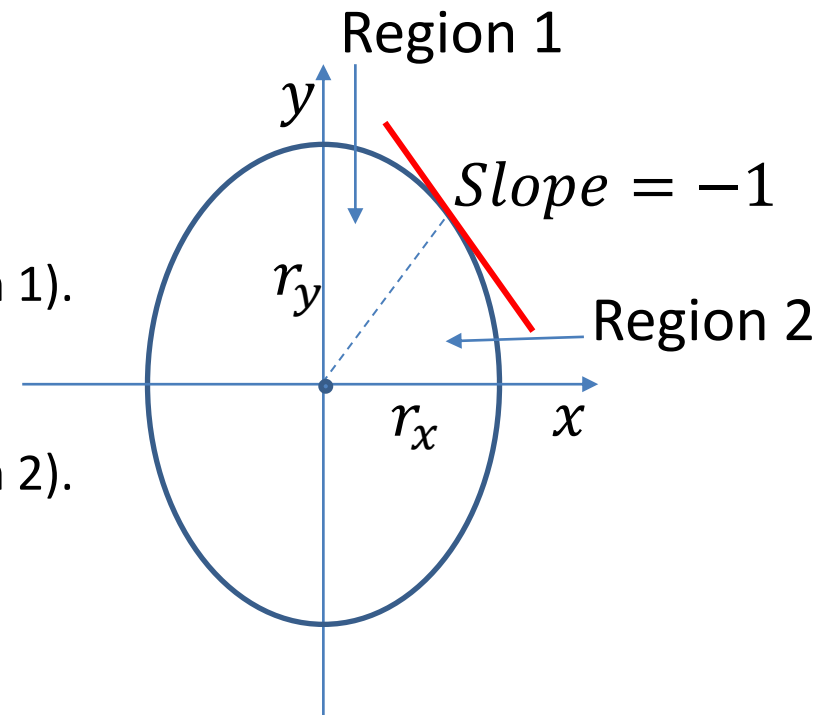
Introduction to Midpoint Ellipse

Algorithm

- Given parameters r_x, r_y , & (x_c, y_c) .
- We determine points (x, y) for an ellipse in standard position centered on the origin.
- Then we shift the points so the ellipse is centered at (x_c, y_c) .
- If we want to display the ellipse in nonstandard position then we rotate the ellipse about its center to align with required direction.
- For the present we consider only the standard position.
- We draw ellipse in first quadrant and than use symmetry property for other three quadrant.

Regions in Midpoint Ellipse Algorithm

- In this method we divide first quadrant into two parts according to the slope of an ellipse
 - Boundary divides region at
 - slope = -1.
- We take unit step in X direction
 - If magnitude of ellipse slope < 1 (Region 1).
- We take unit step in Y direction
 - If magnitude of ellipse slope > 1 (Region 2).



Ways of Processing Midpoint Ellipse Algorithm

- We can start from $(0, r_y)$ and step clockwise along the elliptical path in the first quadrant
- Alternatively, we could start at $(r_x, 0)$ and select points in a counterclockwise order.
- With parallel processors, we could calculate pixel positions in the two regions simultaneously.
- Here we consider sequential implementation of midpoint algorithm.
- We take the start position at $(0, r_y)$ and steps along the elliptical path in clockwise order through the first quadrant.

Decision Parameter Midpoint Ellipse Algorithm

- We define ellipse function for center of ellipse at (0, 0) as follows.

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

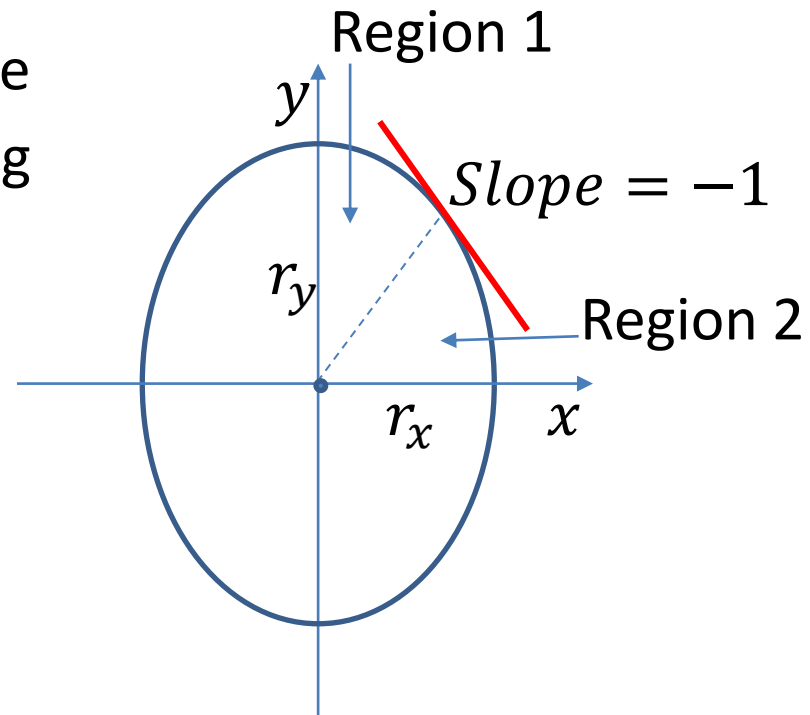
- Which has the following properties:

$$f_{ellipse}(x, y) \begin{cases} < 0 & \text{if } (x, y) \text{ is inside the ellipse boundary} \\ = 0 & \text{if } (x, y) \text{ is on the ellipse boundary} \\ > 0 & \text{if } (x, y) \text{ is outside the ellipse boundary} \end{cases}$$

- Thus the ellipse function serves as the decision parameter in the midpoint ellipse algorithm.
- At each sampling position we select the next pixel from two candidate pixel.

Processing Steps of Midpoint Ellipse Algorithm

- Starting at $(0, r_y)$ we take unit step in x direction until we reach the boundary between region-1 and region-2.
- Then we switch to unit steps in y direction in remaining portion on ellipse in first quadrant.
- At each step we need to test the value of the slope of the curve for deciding the end point of the region-1.

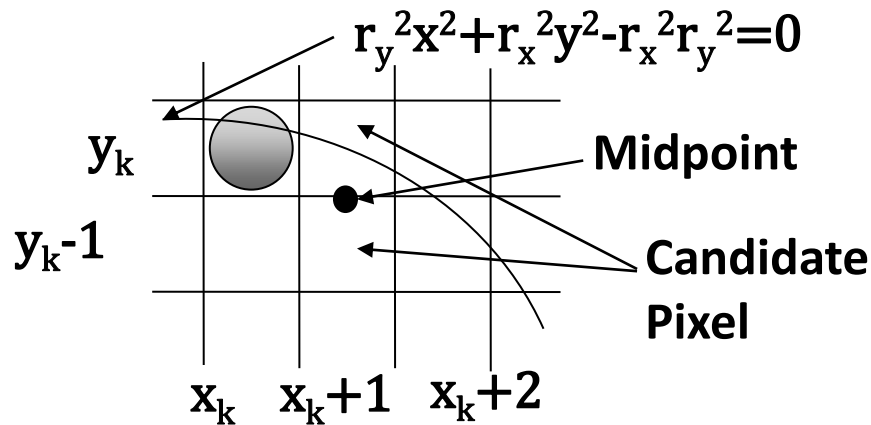


Decide Boundary between Region 1 and 2

- The ellipse slope is calculated using following equation.
- $$\frac{dy}{dx} = -\frac{2r_y^2 x}{2r_x^2 y}$$
- At boundary between region 1 and 2 slope= -1 and equation become.
- $$2r_y^2 x = 2r_x^2 y$$
- Therefore we move out of region 1 whenever following equation is false:
- $$2r_y^2 x \leq 2r_x^2 y$$

Midpoint between Candidate pixel in Region 1

- Figure shows the midpoint between the two candidate pixels at sampling position $x_k + 1$ in the first region.



- Assume we are at (x_k, y_k) position and we determine the next position along the ellipse path, by evaluating decision parameter at midpoint between two candidate pixels.
- $p1_k = f_{ellipse} \left(x_k + 1, y_k - \frac{1}{2} \right)$

Derivation for Region 1

- $p1_k = f_{ellipse} \left(x_k + 1, y_k - \frac{1}{2} \right)$
- $p1_k = r_y^2 (x_k + 1)^2 + r_x^2 \left(y_k - \frac{1}{2} \right)^2 - r_x^2 r_y^2$
- If $p1_k < 0$, the midpoint is inside the ellipse and the pixel on scan line y_k is closer to ellipse boundary
- Otherwise the midpoint is outside or on the ellipse boundary and we select the pixel $y_k - 1$.

Contd.

- At the next sampling position decision parameter for region 1 is evaluated as.

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

- $p1_{k+1} = r_y^2 [(x_k + 1) + 1]^2 + r_x^2 \left(y_{k+1} - \frac{1}{2} \right)^2 - r_x^2 r_y^2$

- Now subtract $p1_k$ from $p1_{k+1}$

- $p1_{k+1} - p1_k = r_y^2 [(x_k + 1) + 1]^2 + r_x^2 \left(y_{k+1} - \frac{1}{2} \right)^2 - r_x^2 r_y^2 - r_y^2 (x_k + 1)^2 - r_x^2 \left(y_k - \frac{1}{2} \right)^2 + r_x^2 r_y^2$

Contd.

- $p1_{k+1} - p1_k = r_y^2 [(x_k + 1) + 1]^2 + r_x^2 \left(y_{k+1} - \frac{1}{2}\right)^2 - r_y^2 (x_k + 1)^2 - r_x^2 \left(y_k - \frac{1}{2}\right)^2$
- $p1_{k+1} - p1_k = r_y^2 (x_k + 1)^2 + 2r_y^2 (x_k + 1) + r_y^2 + r_x^2 \left(y_{k+1} - \frac{1}{2}\right)^2 - r_y^2 (x_k + 1)^2 - r_x^2 \left(y_k - \frac{1}{2}\right)^2$
- $p1_{k+1} - p1_k = 2r_y^2 (x_k + 1) + r_y^2 + r_x^2 \left[\left(y_{k+1} - \frac{1}{2}\right)^2 - \left(y_k - \frac{1}{2}\right)^2 \right]$
- Now making $p1_{k+1}$ as subject.
- $p1_{k+1} = p1_k + 2r_y^2 (x_k + 1) + r_y^2 + r_x^2 \left[\left(y_{k+1} - \frac{1}{2}\right)^2 - \left(y_k - \frac{1}{2}\right)^2 \right]$

Contd.

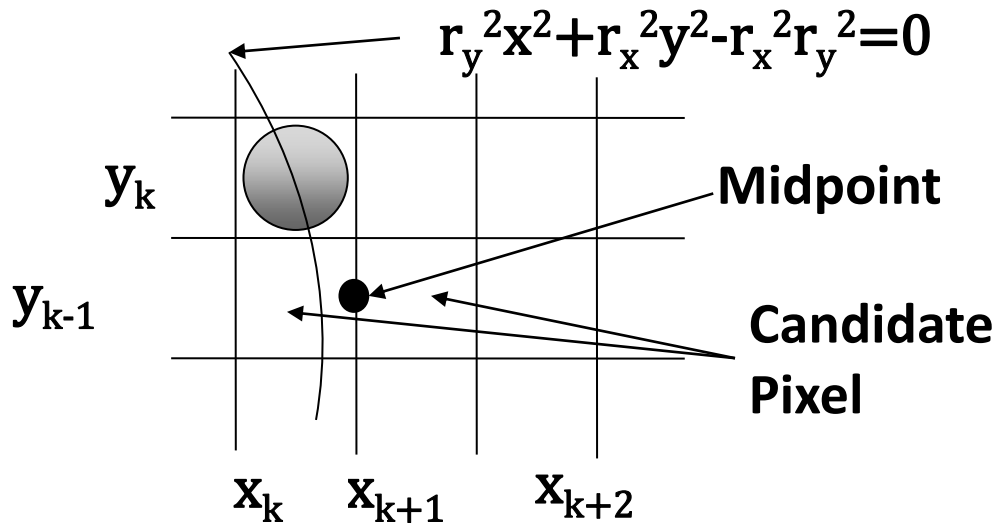
- $p1_{k+1} = p1_k + 2r_y^2(x_k + 1) + r_y^2 + r_x^2 \left[\left(y_{k+1} - \frac{1}{2} \right)^2 - \left(y_k - \frac{1}{2} \right)^2 \right]$
- y_{k+1} is either y_k or $y_k - 1$, depends on the sign of $p1_k$

IDP for Region 1

- Now we calculate the initial decision parameter $p1_0$ by putting $(x_0, y_0) = (0, r_y)$ as follow.
- $p1_0 = f_{ellipse} \left(0 + 1, r_y - \frac{1}{2} \right)$
- $p1_0 = r_y^2(1)^2 + r_x^2 \left(r_y - \frac{1}{2} \right)^2 - r_x^2 r_y^2$
- $p1_0 = r_y^2 + r_x^2 \left(r_y - \frac{1}{2} \right)^2 - r_x^2 r_y^2$
- $p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$

Midpoint between Candidate pixel in Region 2

- Now we similarly calculate over region-2.
- Unit stepping in negative y direction and the midpoint is now taken between horizontal pixels at each step.



- For this region, the decision parameter is evaluated as follows.
- $$p2_k = f_{ellipse} \left(x_k + \frac{1}{2}, y_k - 1 \right)$$

Derivation for Region 2

- $p2_k = f_{ellipse} \left(x_k + \frac{1}{2}, y_k - 1 \right)$
- $p2_k = r_y^2 \left(x_k + \frac{1}{2} \right)^2 + r_x^2 (y_k - 1)^2 - r_x^2 r_y^2$
- If $p2_k > 0$ the midpoint is outside the ellipse boundary, and we select the pixel at x_k .
- If $p2_k \leq 0$ the midpoint is inside or on the ellipse boundary and we select $x_k + 1$.

Contd.

- At the next sampling position decision parameter for region 2 is evaluated as.

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

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

- Now subtract $p2_k$ from $p2_{k+1}$

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

Contd.

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

Contd.

- $p2_{k+1} - p2_k = -2r_x^2(y_k - 1) + r_x^2 + r_y^2 \left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2 \right]$
- Now making $p2_{k+1}$ as subject.
- $p2_{k+1} = p2_k - 2r_x^2(y_k - 1) + r_x^2 + r_y^2 \left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2 \right]$
- Here x_{k+1} is either x_k or $x_k + 1$, depends on the sign of $p2_k$.

IDP for Region 2

- In region-2 initial position is selected which is last position of region one and the initial decision parameter is calculated as follows.
- $p2_0 = f_{ellipse} \left(x_0 + \frac{1}{2}, y_0 - 1 \right)$
- $p2_0 = r_y^2 \left(x_0 + \frac{1}{2} \right)^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$
- For simplify calculation of $p2_0$ we could also select pixel position in counterclockwise order starting at $(r_x, 0)$.

Algorithm for Midpoint Ellipse Generation

1. Input r_x , r_y and ellipse center (x_c, y_c) , and obtain the first point on an ellipse centered on the origin as

$$(x_0, y_0) = (0, r_y)$$

2. Calculate the initial value of the decision parameter in region 1 as

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

3. At each x_k position in region 1, starting at $k = 0$, perform the following test:

If $p1_k < 0$, then the next point along the ellipse is (x_{k+1}, y_k) and

$$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} + r_y^2$$

Otherwise, the next point along the ellipse is $(x_{k+1}, y_k - 1)$ and

$$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} + r_y^2 - 2r_x^2 y_{k+1}$$

With

$$2r_y^2 x_{k+1} = 2r_y^2 x_k + 2r_y^2, \quad 2r_x^2 y_{k+1} = 2r_x^2 y_k - 2r_x^2$$

And continue until $2r_y^2 x \leq 2r_x^2 y$

Contd.

4. Calculate the initial value of the decision parameter in region 2 using the last point (x_0, y_0) calculated in region 1 as

$$p2_0 = r_y^2 \left(x_0 + \frac{1}{2}\right)^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

5. At each y_k position in region-2, starting at $k = 0$, perform the following test:
If $p2_k > 0$, the next point along the ellipse is $(x_k, y_k - 1)$ and

$$p2_{k+1} = p2_k - 2r_x^2 y_{k+1} + r_x^2$$

Otherwise, the next point along the ellipse is $(x_k + 1, y_k - 1)$ and

$$p2_{k+1} = p2_k - 2r_x^2 y_{k+1} + r_x^2 + 2r_y^2 x_{k+1}$$

Using the same incremental calculations for x and y as in region 1.

6. Determine symmetry points in the other three quadrants.
7. Move each calculated pixel position (x, y) onto the elliptical path centered on (x_c, y_c) and plot the coordinate values:

$$x = x + x_c, y = y + y_c$$

8. Repeat the steps for region 2 until $y_k \geq 0$.

Example Midpoint Ellipse Algorithm

- Example: Calculate intermediate pixel position (For first quadrant) for ellipse with $r_x = 8, r_y = 6$ and ellipse center is at origin
- Initial point $(0, r_y) = (0, 6)$
- $p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$
- $p1_0 = 6^2 - 8^2 * 6 + \frac{1}{4} 8^2$
- $p1_0 = -332$

Contd.

- $p1_0 = -332$
- $p1_{k+1} = p1_k + 2r_y^2(x_k + 1) + r_y^2 + r_x^2 \left[\left(y_{k+1} - \frac{1}{2} \right)^2 - \left(y_k - \frac{1}{2} \right)^2 \right]$
- $p1_1 = -332 + 2 * 6^2(0 + 1) + 6^2 + 8^2 \left[\left(6 - \frac{1}{2} \right)^2 - \left(6 - \frac{1}{2} \right)^2 \right] = -224$

K	$p1_k$	(x_{k+1}, y_{k+1})

Calculation stop when $2r_y^2x > 2r_x^2y$

*If $p1_k < 0$ so we select $y_{k+1} = y_k$
Else we select $y_{k+1} = y_k - 1$*

Contd.

- $p1_{k+1} = p1_k + 2r_y^2(x_k + 1) + r_y^2 + r_x^2 \left[\left(y_{k+1} - \frac{1}{2} \right)^2 - \left(y_k - \frac{1}{2} \right)^2 \right]$
- $p1_2 = -224 + 2 * 6^2(1 + 1) + 6^2 + 8^2 \left[\left(6 - \frac{1}{2} \right)^2 - \left(6 - \frac{1}{2} \right)^2 \right] = -44$

K	p1 _k	(x _{k+1} , y _{k+1})
0	-332	(1, 6)
1	-224	(2, 6)

Calculation stop when $2r_y^2x > 2r_x^2y$

*If $p1_k < 0$ so we select $y_{k+1} = y_k$
Else we select $y_{k+1} = y_k - 1$*

Contd.

- $p1_{k+1} = p1_k + 2r_y^2(x_k + 1) + r_y^2 + r_x^2 \left[\left(y_{k+1} - \frac{1}{2} \right)^2 - \left(y_k - \frac{1}{2} \right)^2 \right]$
- $p1_3 = -44 + 2 * 6^2(2 + 1) + 6^2 + 8^2 \left[\left(6 - \frac{1}{2} \right)^2 - \left(6 - \frac{1}{2} \right)^2 \right] = 208$

K	p1 _k	(x _{k+1} , y _{k+1})
0	-332	(1, 6)
1	-224	(2, 6)
2	-44	(3, 6)

Calculation stop when $2r_y^2x > 2r_x^2y$

*If $p1_k < 0$ so we select $y_{k+1} = y_k$
Else we select $y_{k+1} = y_k - 1$*

Contd.

- $p1_{k+1} = p1_k + 2r_y^2(x_k + 1) + r_y^2 + r_x^2 \left[\left(y_{k+1} - \frac{1}{2} \right)^2 - \left(y_k - \frac{1}{2} \right)^2 \right]$
- $p1_4 = 208 + 2 * 6^2(3 + 1) + 6^2 + 8^2 \left[\left(5 - \frac{1}{2} \right)^2 - \left(6 - \frac{1}{2} \right)^2 \right] = -108$

K	p1 _k	(x _{k+1} , y _{k+1})
0	-332	(1, 6)
1	-224	(2, 6)
2	-44	(3, 6)
3	208	(4, 5)

Calculation stop when $2r_y^2x > 2r_x^2y$

*If $p1_k < 0$ so we select $y_{k+1} = y_k$
Else we select $y_{k+1} = y_k - 1$*

Contd.

- $p1_{k+1} = p1_k + 2r_y^2(x_k + 1) + r_y^2 + r_x^2 \left[\left(y_{k+1} - \frac{1}{2} \right)^2 - \left(y_k - \frac{1}{2} \right)^2 \right]$
- $p1_5 = -108 + 2 * 6^2(4 + 1) + 6^2 + 8^2 \left[\left(5 - \frac{1}{2} \right)^2 - \left(5 - \frac{1}{2} \right)^2 \right] = 288$

K	p1 _k	(x _{k+1} , y _{k+1})
0	-332	(1, 6)
1	-224	(2, 6)
2	-44	(3, 6)
3	208	(4, 5)
4	-108	(5, 5)

Calculation stop when $2r_y^2x > 2r_x^2y$

*If $p1_k < 0$ so we select $y_{k+1} = y_k$
Else we select $y_{k+1} = y_k - 1$*

Contd.

- $p1_{k+1} = p1_k + 2r_y^2(x_k + 1) + r_y^2 + r_x^2 \left[\left(y_{k+1} - \frac{1}{2} \right)^2 - \left(y_k - \frac{1}{2} \right)^2 \right]$
- $p1_6 = 288 + 2 * 6^2(5 + 1) + 6^2 + 8^2 \left[\left(4 - \frac{1}{2} \right)^2 - \left(5 - \frac{1}{2} \right)^2 \right] = 244$

K	p1 _k	(x _{k+1} , y _{k+1})
0	-332	(1, 6)
1	-224	(2, 6)
2	-44	(3, 6)
3	208	(4, 5)
4	-108	(5, 5)
5	288	(6, 4)

Calculation stop when $2r_y^2x > 2r_x^2y$

*If $p1_k < 0$ so we select $y_{k+1} = y_k$
Else we select $y_{k+1} = y_k - 1$*

Contd.

- $p2_0 = r_y^2 \left(x_0 + \frac{1}{2}\right)^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$

- $p2_0 = 6^2 \left(7 + \frac{1}{2}\right)^2 + 8^2 (3 - 1)^2 - 8^2 6^2 = -23$

K	p2 _k	(x _{k+1} , y _{k+1})

Calculation stop when $y_k \leq 0$

*If $p2_k > 0$ so we select $x_{k+1} = x_k$
Else we select $x_{k+1} = x_k + 1$*

Contd.

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

- $p2_1 = -23 - 2 * 8^2 (3 - 1) + 8^2 + 6^2 \left[\left(8 + \frac{1}{2}\right)^2 - \left(7 + \frac{1}{2}\right)^2 \right] = 361$

K	p2 _k	(x _{k+1} , y _{k+1})
0	-23	(8, 2)

Calculation stop when $y_k \leq 0$

*If $p2_k > 0$ so we select $x_{k+1} = x_k$
Else we select $x_{k+1} = x_k + 1$*

Contd.

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

- $$p2_2 = 361 - 2 * 8^2 (2 - 1) + 8^2 + 6^2 \left[\left(8 + \frac{1}{2} \right)^2 - \left(8 + \frac{1}{2} \right)^2 \right] = 297$$

K	p2 _k	(x _{k+1} , y _{k+1})
0	-23	(8, 2)
1	361	(8, 1)

*Calculation stop when
 $y_k \leq 0$*

*If $p2_k > 0$ so we select $x_{k+1} = x_k$
Else we select $x_{k+1} = x_k + 1$*

Contd.

- Plot the pixel.
- Plot initial point(0, 6)

Center (0, 0)

(1, 6)

(2, 6)

(3, 6)

(4, 5)

(5, 5)

(6, 4)

(7, 3)

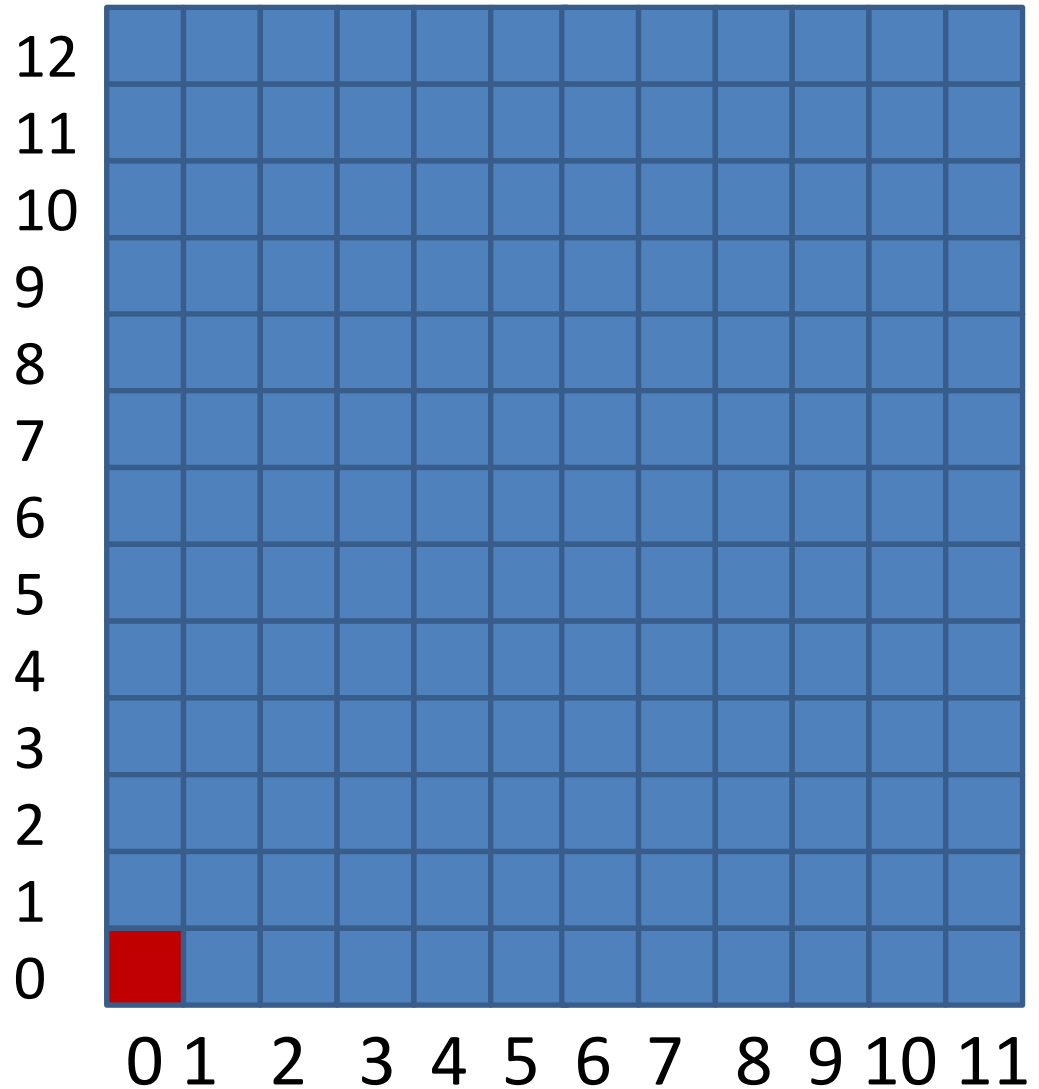
Center (0, 0)

(8, 2)

(8, 1)

(8, 0)

Center
(0, 0)



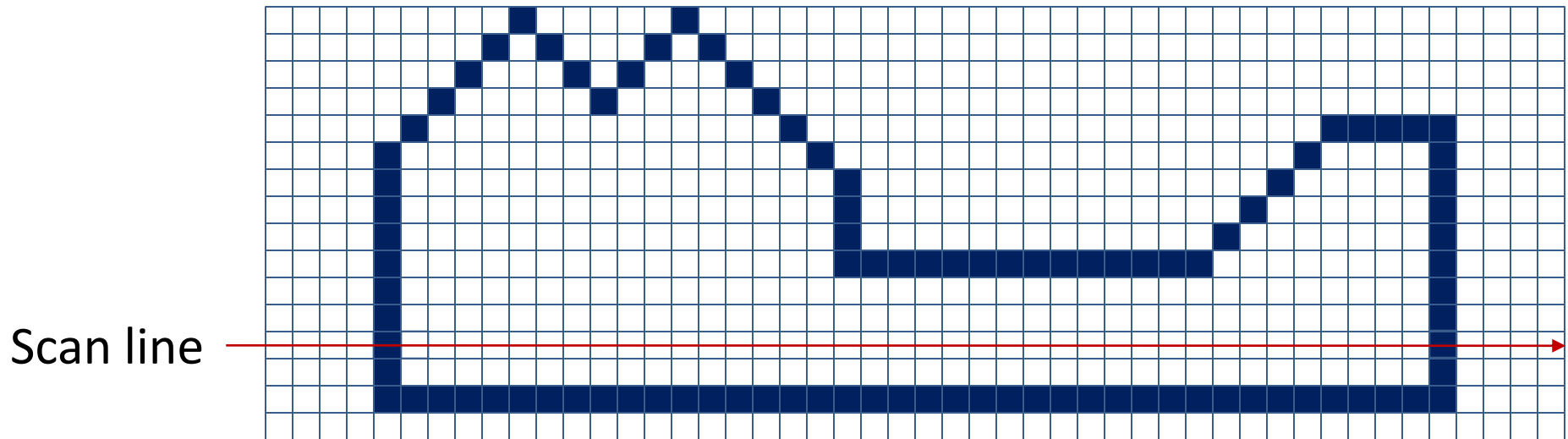
Filled-Area Primitives

- In practical we often use polygon which are filled with some colour or pattern inside it.
- There are two basic approaches to area filling on raster systems.
 - One way to fill an area is to determine the overlap intervals for scan line that cross the area.
 - Another method is to start from a given interior position and paint outward from this point until we encounter boundary.



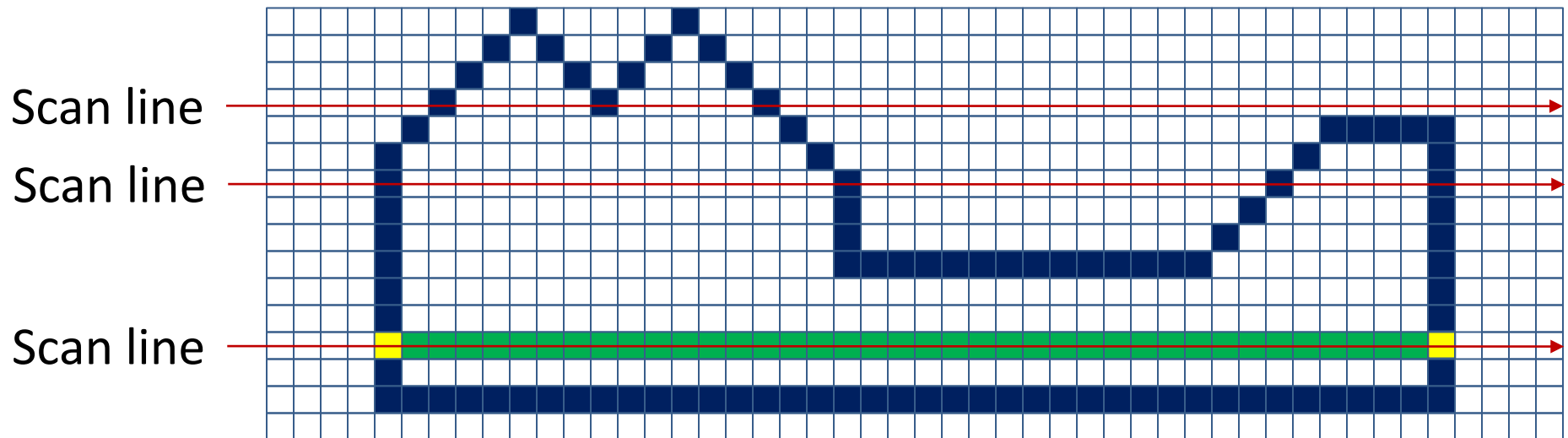
Scan-Line Polygon Fill Algorithm

- For each scan-line crossing a polygon, the algorithm locates the intersection points of scan line with the polygon edges.
- These intersection points are stored from left to right.
- Frame buffer positions between each pair of intersection points are set to specified fill color.



Contd.

- Scan line intersects at vertex are required special handling.
- For vertex we must look at the other endpoints of the two line segments which meet at this vertex.
 - If these points lie on the same (up or down) side of the scan line, then that point is counts as two intersection points.
 - If they lie on opposite sides of the scan line, then the point is counted as single intersection.



Edge Intersection Calculation with Scan-Line

- Coherence methods often involve incremental calculations applied along a single scan line or between successive scan lines.
- In determining edge intersections, we can set up incremental coordinate calculations along any edge by exploiting the fact that the slope of the edge is constant from one scan line to the next.
- For above figure we can write slope equation for polygon boundary as follows.
- $$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$
- Since change in y coordinates between the two scan lines is simply
- $$y_{k+1} - y_k = 1$$

Contd.

- So slope equation can be modified as follows

- $$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

- $$m = \frac{1}{x_{k+1} - x_k}$$

- $$x_{k+1} - x_k = \frac{1}{m}$$

- $$x_{k+1} = x_k + \frac{1}{m}$$

- Each successive x intercept can thus be calculated by adding the inverse of the slope and rounding to the nearest integer.

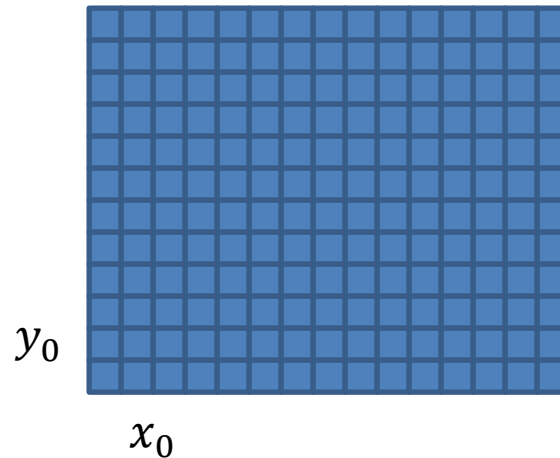
Edge Intersection Calculation with Scan-Line for parallel execution

- For parallel execution of this algorithm we assign each scan line to separate processor in that case instead of using previous x values for calculation we use initial x values by using equation as.
- $$x_k = x_0 + \frac{k}{m}$$
- Now if we put $m = \frac{\Delta y}{\Delta x}$ in incremental calculation equation $x_{k+1} = x_k + \frac{1}{m}$ then we obtain equation as.
- $$x_{k+1} = x_k + \frac{\Delta x}{\Delta y}$$
- Using this equation we can perform integer evaluation of x intercept.

Simplified Method for Edge Intersection Calculation with Scan-Line

1. Suppose $m = 7/3$
2. Initially, set counter to 0, and increment to 3 (which is Δx).
3. When move to next scan line, increment counter by adding Δx
4. When counter is equal to or greater than 7 (which is Δy), increment the $x - intercept$ (in other words, the $x - intercept$ for this scan line is one more than the previous scan line), and decrement counter by 7 (which is Δy). $\Delta x = 3, \Delta y = 7$

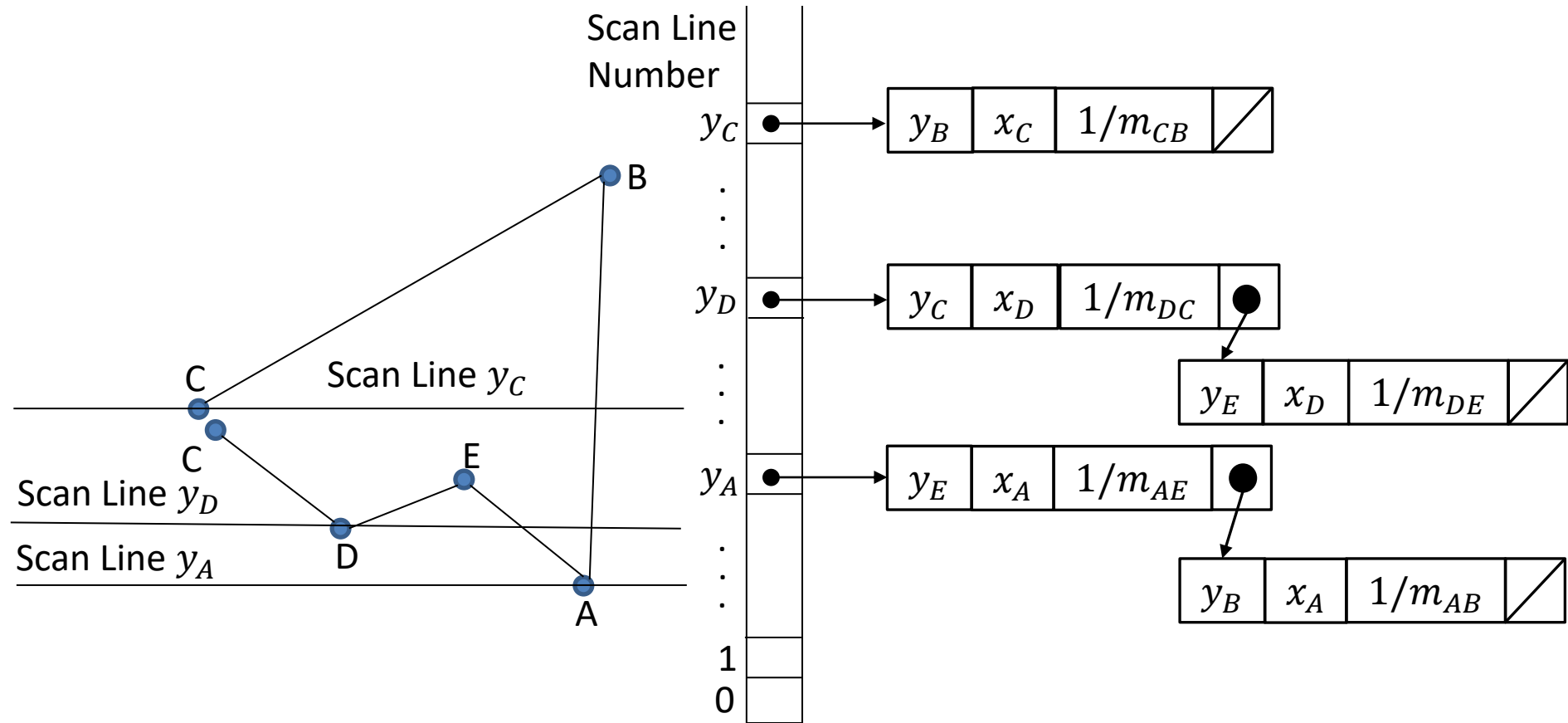
Counter=0



Use of Sorted Edge table in Scan-Line Polygon Fill Algorithm

- To efficiently perform a polygon fill, we can first store the polygon boundary in a sorted edge table.
- It contains all the information necessary to process the scan lines efficiently.
- We use bucket sort to store the edge sorted on the smallest y value of each edge in the correct scan line positions.
- Only the non-horizontal edges are entered into the sorted edge table.

Contd.

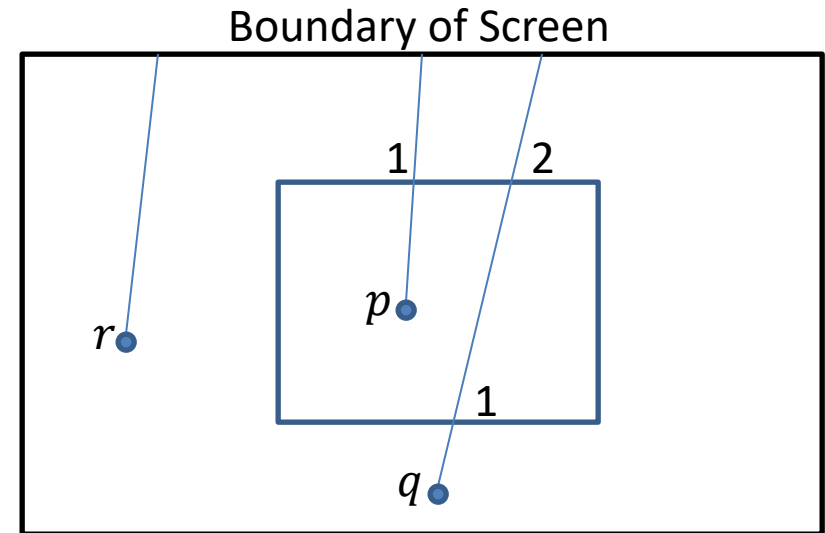


Inside-Outside Tests

- In area filling and other graphics operation often required to find particular point is inside or outside the polygon.
- For finding which region is inside or which region is outside most graphics package use either
 1. Odd even rule OR
 2. Nonzero winding number rule

Odd Even/ Odd Parity/ Even Odd Rule

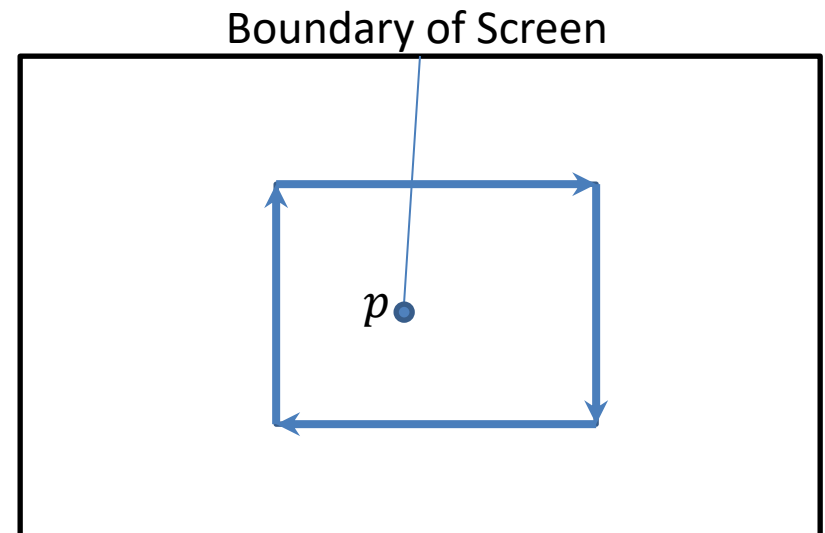
- By conceptually drawing a line from any position p to a distant point outside the coordinate extents of the object.
- Than counting the number of edges crossing by this line.
 1. If Edge count is odd, than p is an interior point.
 2. Otherwise p is exterior point.
- To obtain accurate edge count we must sure that line selected is does not pass from any vertices.



Nonzero Winding Number Rule

- This method counts the number of times the polygon edges wind around a particular point in the counterclockwise direction.
- This count is called the winding number.
- We apply this rule by initializing winding number with 0.
- Then draw a line for any point p to distant point beyond the coordinate extents of the object.

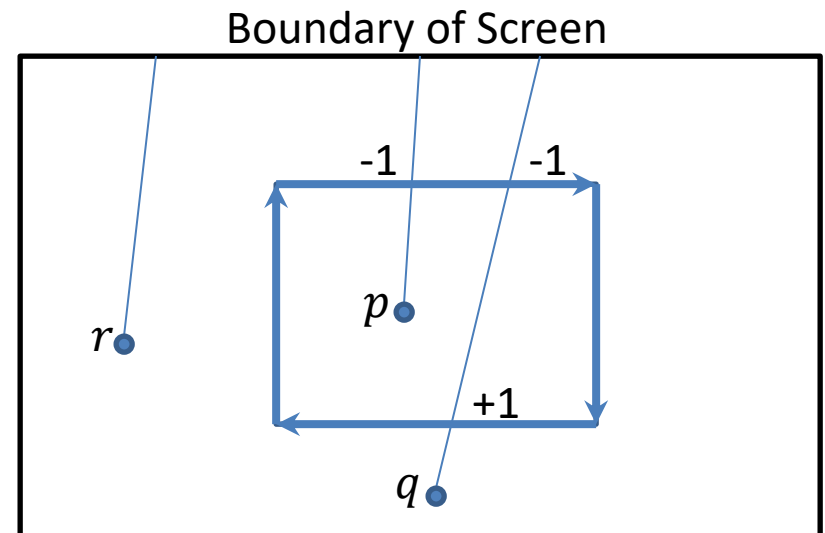
Winding number=0



Contd.

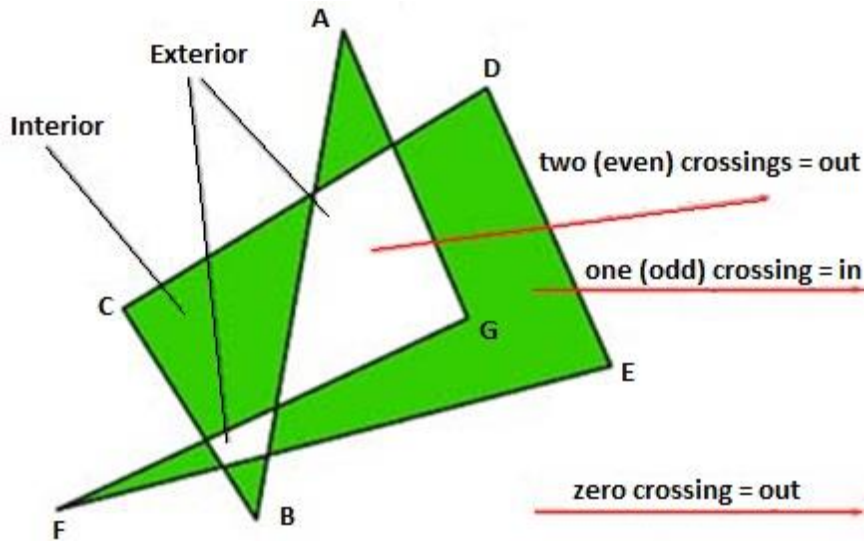
- The line we choose must not pass through vertices.
- Then we move along that line we find number of intersecting edges.
 1. If edge cross our line from right to left We add 1 to winding number
 2. Otherwise subtract 1 from winding number
- IF the final value of winding number is nonzero then the point is interior otherwise point is exterior.

Winding number = $0 - 1 = -1$

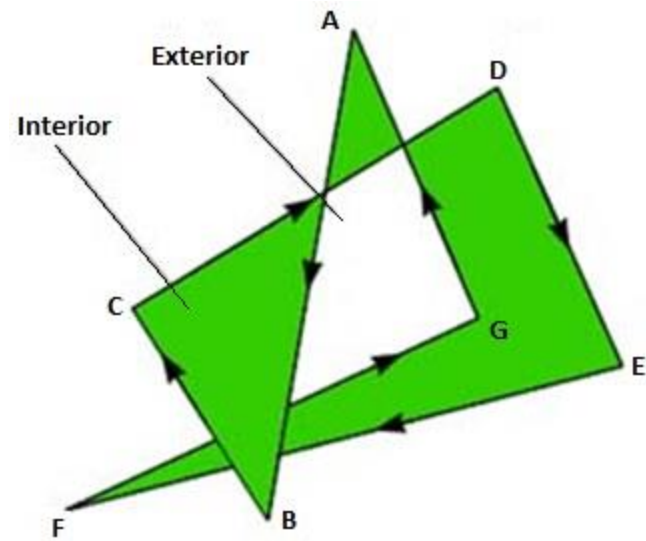


Comparison between Odd Even Rule and Nonzero Winding Rule

- For standard polygons and simple object both rule gives same result but for more complicated shape both rule gives different result.



Odd Even Rule



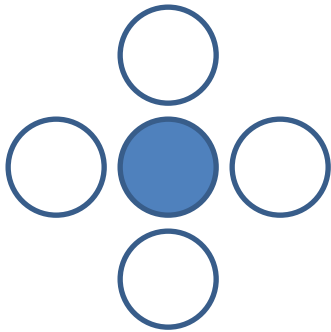
Nonzero Winding Rule

Scan-Line Fill of Curved Boundary Areas

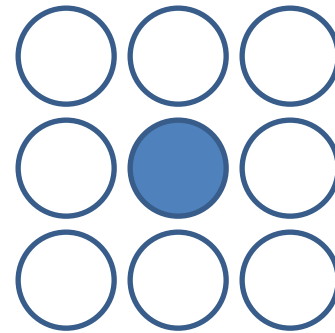
- Scan-line fill of region with curved boundary is more time consuming as intersection calculation now involves nonlinear boundaries.
- For simple curve such as circle or ellipse scan line fill process is straight forward process.
- We calculate the two scan line intersection on opposite side of the curve.
- This is same as generating pixel position along the curve boundary using standard equation of curve.
- Then we fill the color between two boundary intersections.
- Symmetry property is used to reduce the calculation.

Introduction to Boundary / Edge Fill Algorithm

- In this method, edges of the polygons are drawn.
- Then starting with some seed (any point inside the polygon) we examine the neighbouring pixels to check whether the boundary pixel is reached.
- If boundary pixels are not reached, pixels are highlighted and the process is continued until boundary pixels are reached.
- Selection of neighbour pixel is either 4-connected or 8-connected.



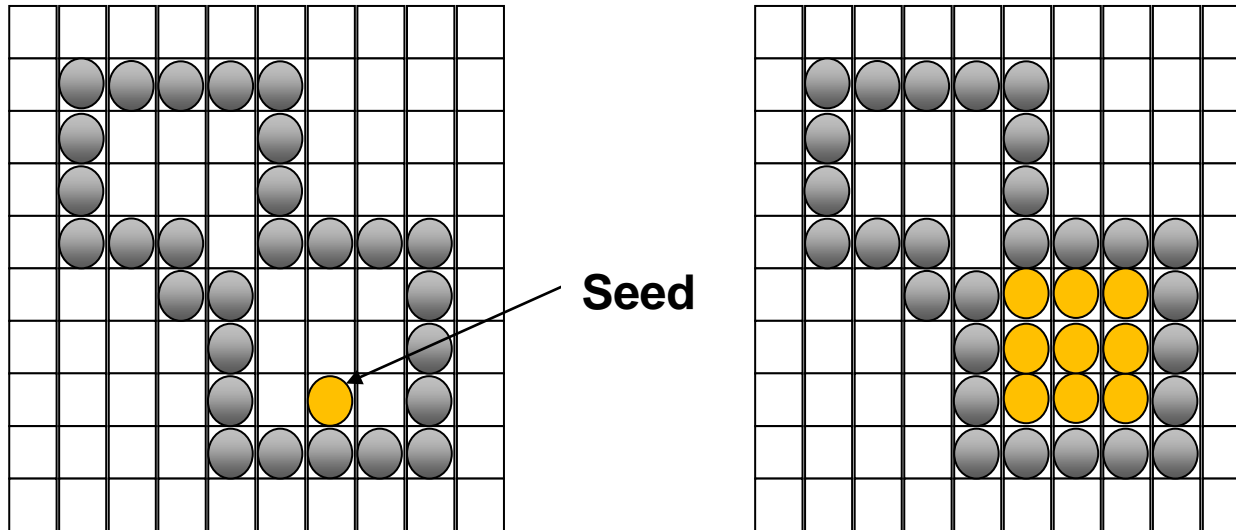
4-Connected Region



8-Connected Region

Contd.

- In some cases, an 8-connected algorithm is more accurate than the 4-connected algorithm.
- Some times 4-connected algorithm produces the partial fill.



Boundary / Edge Fill Algorithm

Procedure:

```
boundary-fill4(x, y, f-colour, b-colour)
```

```
{
```

```
    if(getpixel (x,y) != b-colour && gepixel (x, y) != f-colour)
```

```
    {
```

```
        putpixel (x, y, f-colour)
```

```
        boundary-fill4(x + 1, y, f-colour, b-colour);
```

```
        boundary-fill4(x, y + 1, f-colour, b-colour);
```

```
        boundary-fill4(x - 1, y, f-colour, b-colour);
```

```
        boundary-fill4(x, y - 1, f-colour, b-colour);
```

```
    }
```

```
}
```

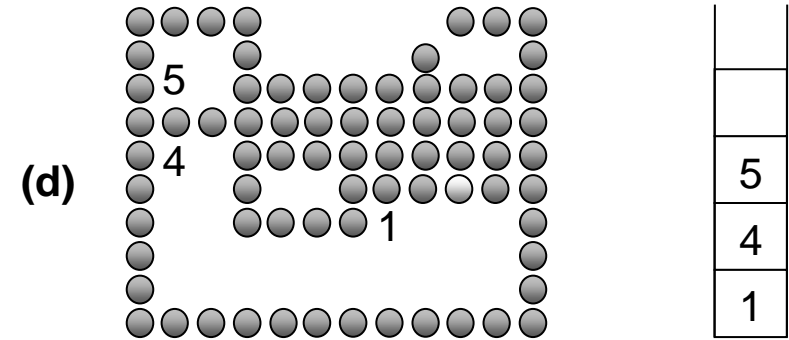
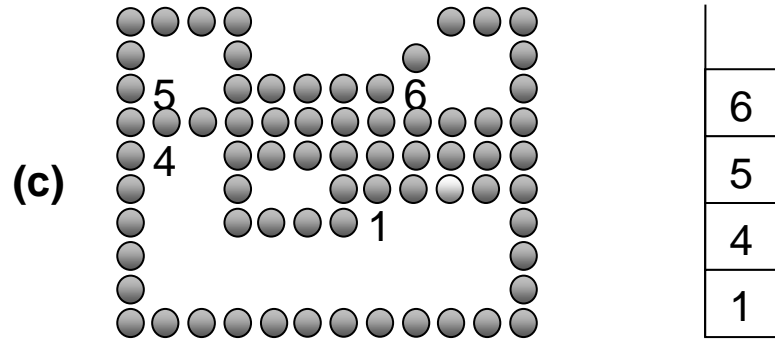
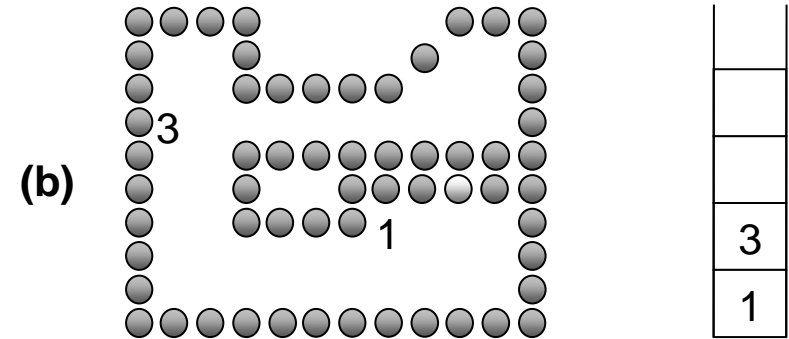
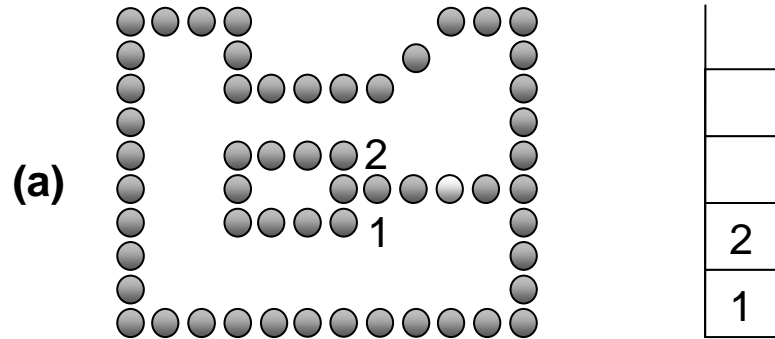
Problem of Staking and Efficient Method

- Same procedure can be modified according to 8 connected region algorithm by including four additional statements to test diagonal positions.
- This procedure requires considerable stacking of neighbouring points more, efficient methods are generally employed.
- Efficient method fill horizontal pixel spans across scan lines, instead of proceeding to 4 connected or 8 connected neighbouring points.
- Then we need only stack a beginning position for each horizontal pixel span, instead of stacking all unprocessed neighbouring positions around the current position.
- Starting from the initial interior point with this method, we first fill in the contiguous span of pixels on this starting scan line.

Contd.

- Then we locate and stack starting positions for spans on the adjacent scan lines.
- Spans are defined as the contiguous horizontal string of positions bounded by pixels displayed in the area border colour.
- At each subsequent step, we unstack the next start position and repeat the process.
- For e.g.

Contd.



Introduction to Flood-Fill

Algorithm

- Sometimes it is required to fill in an area that is not defined within a single colour boundary.
- In such cases we can fill areas by replacing a specified interior colour instead of searching for a boundary colour.
- This approach is called a flood-fill algorithm. Like boundary fill algorithm, here we start with some seed and examine the neighbouring pixels.
- However, here pixels are checked for a specified interior colour instead of boundary colour and they are replaced by new colour.
- Using either a 4-connected or 8-connected approach, we can step through pixel positions until all interior point have been filled.

Flood-Fill Algorithm

Procedure :

```
flood-fill4(x, y, new-colour, old-colour)
{
    if(getpixel (x,y) == old-colour)
    {
        putpixel (x, y, new-colour)
        flood-fill4 (x + 1, y, new-colour, old -colour);
        flood-fill4 (x, y + 1, new -colour, old -colour);
        flood-fill4 (x - 1, y, new -colour, old -colour);
        flood-fill4 (x, y - 1, new -colour, old-colour);
    }
}
```

Character Generation

- We can display letters and numbers in variety of size and style.
- The overall design style for the set of character is called typeface.
- Today large numbers of typefaces are available for computer application for example Helvetica, Arial etc.
- Originally, the term font referred to a set of cast metal character forms in a particular size and format.
- Example: 10-point Courier Italic or 12- point Palatino Bold.

Contd.

- Now, the terms font and typeface are often used interchangeably, since printing is no longer done with cast metal forms.
- Methods of character generation are:
 - Bitmap Font/ Bitmapped Font
 - Outline Font
 - Stroke Method
 - Starbust Method

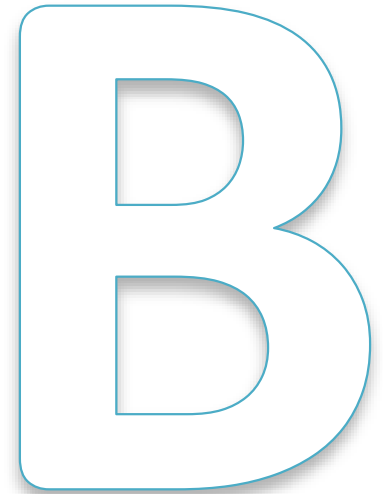
Bitmap Font/ Bitmapped Font

- A simple method for representing the character shapes in a particular typeface is to use rectangular grid patterns.
- In frame buffer, the 1 bits designate which pixel positions are to be displayed on the monitor.
- Bitmap fonts are the simplest to define and display.
- Bitmap fonts require more space.
- It is possible to generate different size and other variation from one set but this usually does not produce good result.

0	0	1	1	1	1	0	0
0	1	1	1	1	1	1	0
1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1

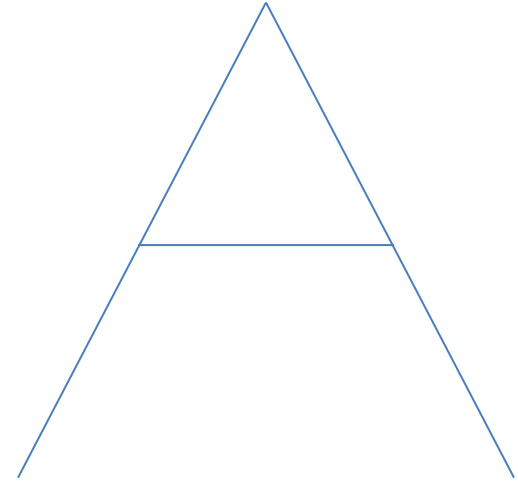
Outline Font

- In this method character is generated using curve section and straight line as combine assembly.
- To display the character we need to fill interior region of the character.
- This method requires less storage since each variation does not required a distinct font cache.
- We can produce boldface, italic, or different sizes by manipulating the curve definitions for the character outlines.
- But this will take more time to process the outline fonts, because they must be scan converted into the frame buffer.



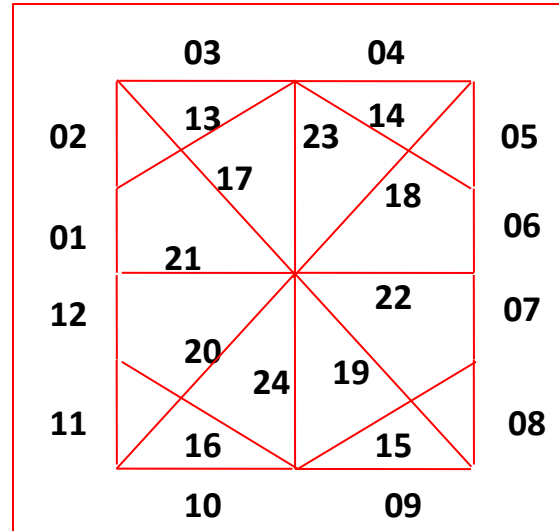
Stroke Method

- It uses small line segments to generate a character.
- The small series of line segments are drawn like a stroke of a pen to form a character.
- We can generate our own stroke method by calling line drawing algorithm.
- Here it is necessary to decide which line segments are needed for each character and then draw that line to display character.
- It support scaling by changing length of line segment.



Starburst Method

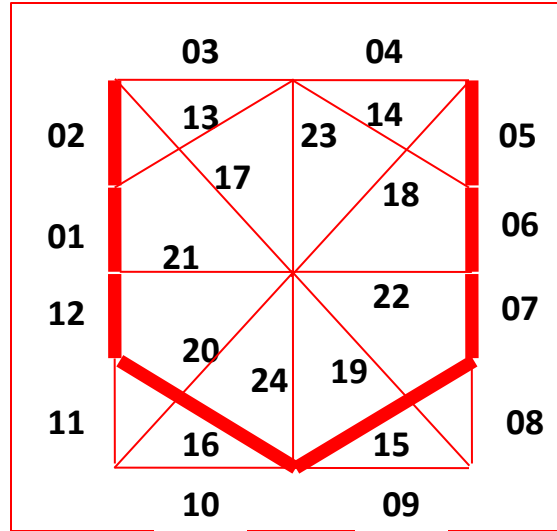
- In this method a fix pattern of lines (24 line) segments are used to generate characters.



- We highlight those lines which are necessary to draw a particular character.
- Pattern for particular character is stored in the form of 24 bit code.
- In which each bit represents corresponding line having that number.
- We put bit value 1 for highlighted line and 0 for other line.

Contd.

- Example letter V




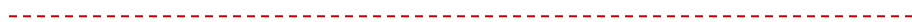


- Code for letter V = **1 1 0 0 1 1 1 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0**
- This technique is not used now a days because:
 - It requires more memory to store 24 bit code for single character.
 - It requires conversion from code to character.
 - It doesn't provide curve shapes.

Line Attributes

- Basic attributes of a straight line segment are:
 - Type
 - Dimension
 - color
 - pen or brush option.

Line Type

- Possible selection for the line-type attribute includes solid lines, dashed lines, and dotted lines etc.

1		Solid
2		Dashed
3		Dotted
4		Dotdash

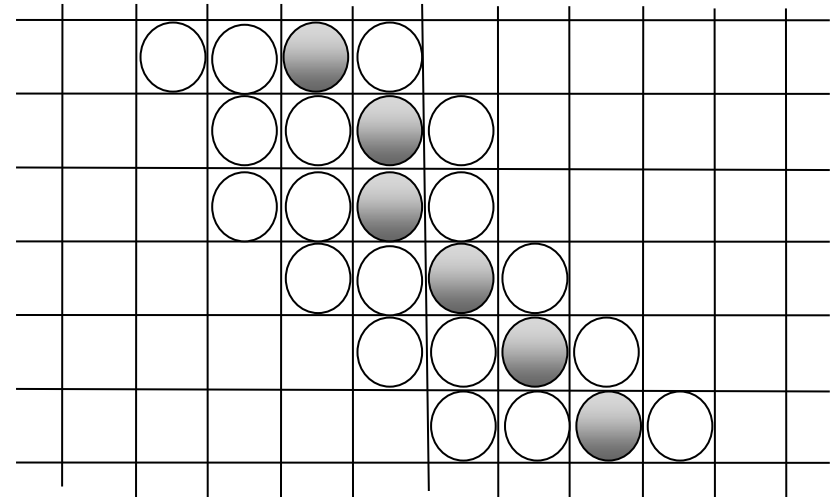
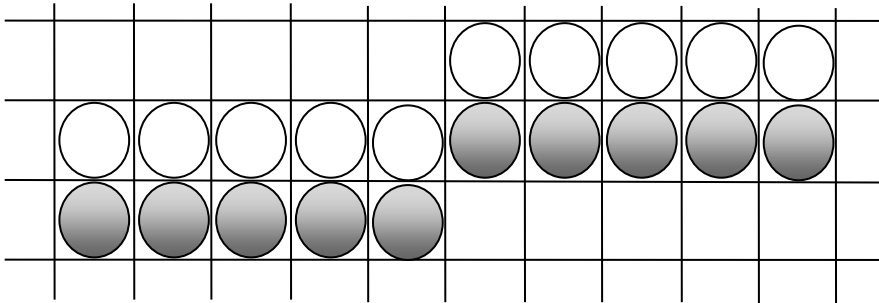
- We modify a line –drawing algorithm to generate such lines by setting the length and spacing of displayed solid sections along the line path.
- To set line type attributes in a PHIGS application program, a user invokes the function: **setLinetype(lt)**
- Where parameter **lt** is assigned a positive integer value of 1, 2, 3, 4... etc. to generate lines that are, respectively solid, dashed, dotted, or dotdash etc.

Line Width

- Implementation of line-width options depends on the capabilities of the output device.
- A heavy line on a video monitor could be displayed as adjacent parallel lines, while a pen plotter might require pen changes.
- To set line width attributes in a PHIGS application program, a user invokes the function: **setLinewidthScalFactor (lw)**
- Line-width parameter **lw** is assigned a positive number to indicate the relative width of the line to be displayed.
- Values greater than 1 produce lines thicker than the standard line width and values less than the 1 produce line thinner than the standard line width.

Contd.

- In raster graphics we generate thick line by plotting
 - above and below pixel of line path when slope $|m| < 1$. &
 - left and right pixel of line path when slope $|m| > 1$.



Line Width at Endpoints and Join

- As we change width of the line we can also change line end and join of two lines which are shown below



Butt caps



Projecting square caps



Round caps



Miter join



Round join



Bevel join

Line color

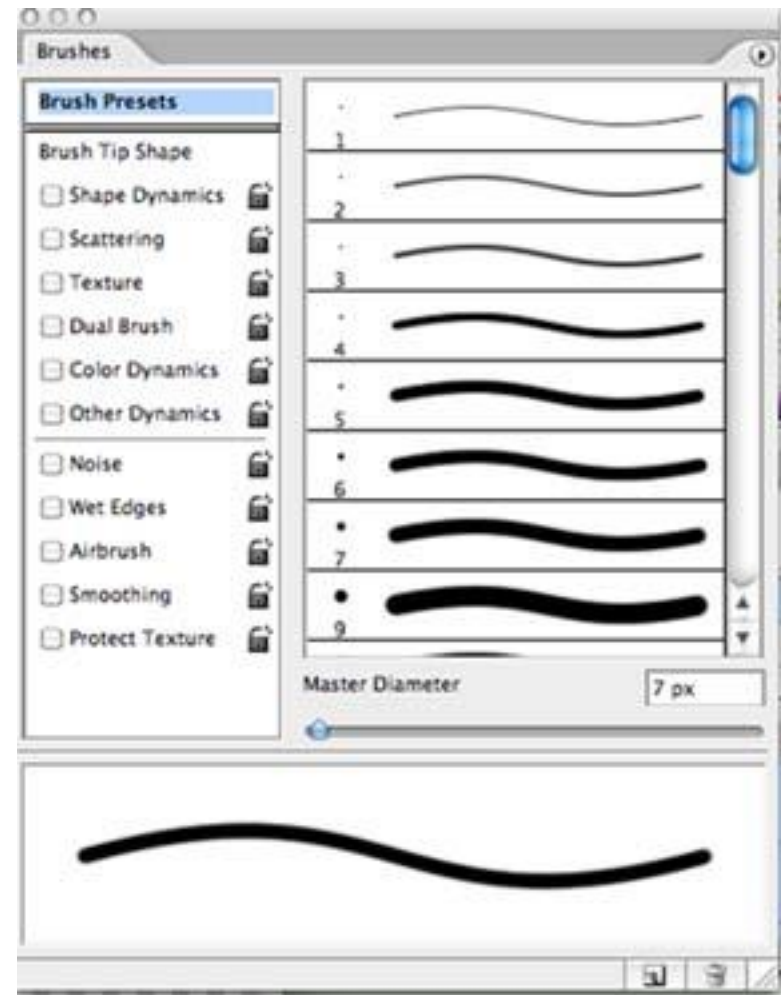
- The name itself suggests that it is defining color of line displayed on the screen.



- By default system produce line with current color but we can change this color by following function in PHIGS package as follows: **setPolylinecolorIndex (lc)**
- In this **lc** is constant specifying particular color to be set.

Pen and Brush Options

- In some graphics packages line is displayed with pen and brush selections.
- Options in this category include shape, size, and pattern.
- These shapes can be stored in a pixel mask that identifies the array of pixel positions that are to be set along the line path.
- Also, lines can be displayed with selected patterns by superimposing the pattern values onto the pen or brush mask.



Color and Grayscale Levels

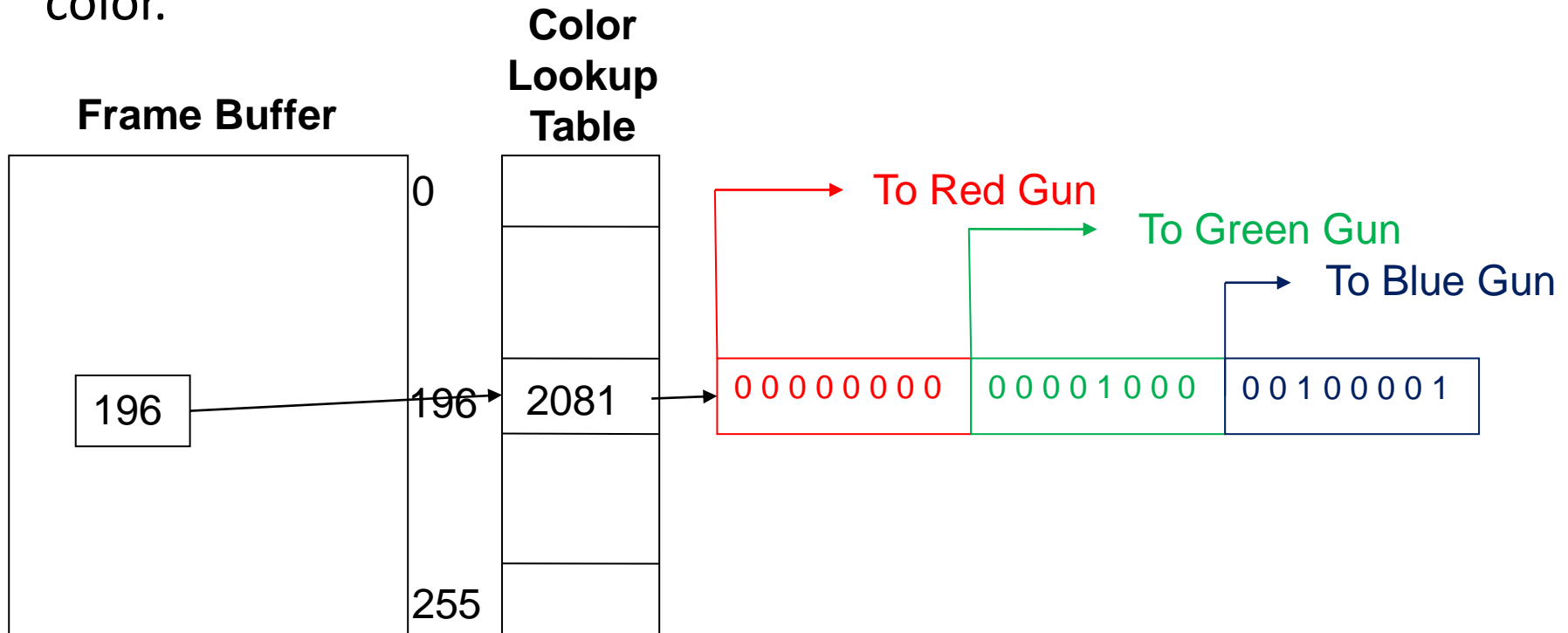
- Various colors and intensity-level options can be made available to a user, depending on the capabilities and design objectives of a particular system.
- General purpose raster-scan systems, for example, usually provide a wide range of colors, while random-scan monitors typically offer only a few color choices, if any.
- In a color raster system, the number of color choices available depends on the amount of storage provided per pixel in the frame buffer

Contd.

- Also, color-information can be stored in the frame buffer in two ways:
 - We can store color codes directly in the frame buffer OR
 - We can put the color codes in a separate table and use pixel values as an index into this table
- With direct storage scheme we required large memory for frame buffer when we display more color.
- While in case of table it is reduced and we call it **color table** or **color lookup table**.

Color Lookup Table

- Color values of 24 bit is stored in lookup table and in frame buffer we store only 8 bit index of required color.
- So that size of frame buffer is reduced and we can display more color.



Greyscale

- With monitors that have no color capability, color function can be used in an application program to set the shades of grey (greyscale) for display primitives.
- Numeric values between 0-to-1 can be used to specify greyscale levels.
- This numeric values is converted in binary code for store in raster system. **Example: frame buffer with 2 bits per pixel.**

Intensity Code	Stored Intensity Values In The		Displayed Greyscale
	Frame Buffer	Binary Code	
0.0	0	00	Black
0.33	1	01	Dark grey
0.67	2	10	Light grey
1.0	3	11	White

Area-Fill Attributes

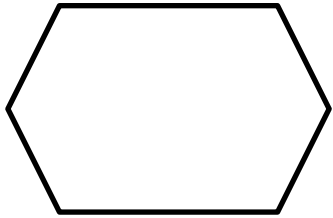
- For filling any area we have choice between solid colors or pattern to fill all these are include in area fill attributes. Which are:
 - Fill Styles
 - Pattern Fill
 - Soft Fill

Fill Styles

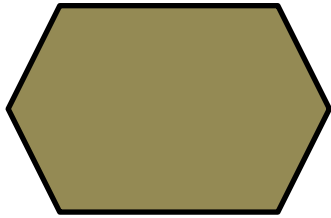
- Area are generally displayed with three basic style.
 1. hollow with color border
 2. filled with solid color
 3. filled with some design
- In PHIGS package fill style is selected by following function:
setInteriorStyle (fs)
- Value of fs include hollow ,solid, pattern etc.

Contd.

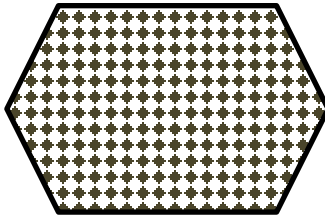
- Another values for fill style is hatch, which is patterns of line like parallel line or crossed line.



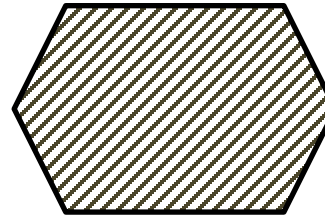
Hollow



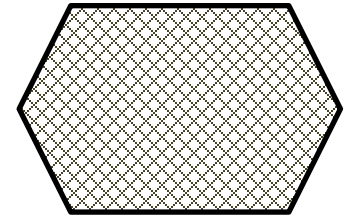
Solid



Pattern



Diagonal
Hatch Fill



Diagonal Cross-
Hatch Fill

- For setting interior color in PHIGS package we use:
setInteriorColorIndex (fc)
- Where **fc** specify the fill color.

Pattern Fill

- We select the pattern with **setInteriorStyleIndex (pi)**
- Where pattern index parameter **pi** specifies position in pattern table entry.
- For example:
 - **SetInteriorStyle(pattern);**
 - **setInteriorStyleIndex (2);**
 - **fillArea (n, points);**

Pattern Table

Index(pi)	Pattern(cp)
1	$\begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}$
2	$\begin{bmatrix} 2 & 1 & 2 \\ 1 & 2 & 1 \\ 2 & 1 & 2 \end{bmatrix}$

Contd.

- We can also maintain separate table for hatch pattern.
- We can also generate our own table with required pattern.
- Other function used for setting other style as follows:
setpatternsizes (dx, dy)
- **setPatternReferencePoint (position)**
- We can create our own pattern by setting and resetting group of pixel and then map it into the color matrix.

Soft Fill

- Soft fill is filling layer of color on back ground color so that we can obtain the combination of both color.
- It is used to recolor or repaint so that we can obtain layer of multiple color and get new color combination.
- One use of this algorithm is soften the fill at boundary so that blurred effect will reduce the aliasing effect.
- For example if we fill t amount of foreground color then pixel color is obtain as:
 - $p = tF + (1 - t)B$
 - Where F is foreground color and B is background color

Contd.

- If we see this color in RGB component then:
- $p(\text{pixel}) = (p_r, p_g, p_b)$
- $f(\text{foreground}) = (f_r, f_g, f_b)$
- $b(\text{background}) = (b_r, b_g, b_b)$
- Then we can calculate t as follows:
- $$t = \frac{P_k - B_k}{F_k - B_k}$$
- If we use more than two colors say three at that time the equation becomes as follows:
- $$p = t_0 F + t_1 B_1 + (1 - t_0 - t_1) B_2$$
- Where the sum of coefficients t_0 , t_1 , and $(1 - t_0 - t_1)$ is 1.

Character Attributes

- The appearance of displayed characters is controlled by attributes such as:
 - Font
 - Size
 - Color
 - Orientation.
- Attributes can be set for entire string or may be individually.

Text Attributes

- In text we are having so many style and design like italic fonts, bold fonts etc.
- For setting the font style in PHIGS package we have function:
setTextFont (tf)
- Where **tf** is used to specify text font.
- For setting color of character in PHIGS we have function:
setTextColorIndex (tc)
- Where text color parameter **tc** specifies an allowable color code.
- For setting the size of the text we use function:
setCharacterheight (ch)
- Where **ch** is used to specify character height.

Contd.

- For scaling the character we use function:
setCharacterExpansionFactor (cw)
- Where character width parameter **cw** is set to a positive real number that scale the character body width.
- Spacing between character is controlled by function:
- **setCharacterSpacing (cs)**
- Where character spacing parameter **cs** can be assigned any real value.

Contd.

- The orientation for a displayed character string is set according to the direction of the character up vector:

setCharacterUpVector (upvect)

- Parameter `upvect` in this function is assigned two values that specify the x and y vector components.
- Text is then displayed so that the orientation of characters from baseline to cap line is in the direction of the up vector.
- For setting the path of the character we use function:

setTextPath (tp)

- Where the text path parameter `tp` can be assigned the value: right, left, up, or down.

Contd.

- For setting the alignment we use function:

setTextAlignment (h, v)

- Where parameter **h** and **v** control horizontal and vertical alignment respectively.

- For specifying precision for text display is given with function:

setTextPrecision (tpr)

- Where text precision parameter **tpr** is assigned one of the values: string, char, or stroke.
- The highest-quality text is produced when the parameter is set to the value stroke.

Marker Attributes

- A marker symbol display single character in different color and in different sizes.
- For marker attributes implementation by procedure that load the chosen character into the raster at defined position with the specified color and size.
- We select marker type using function: **setMarkerType** ([mt](#))
- Where marker type parameter [mt](#) is set to an integer code.

Contd.

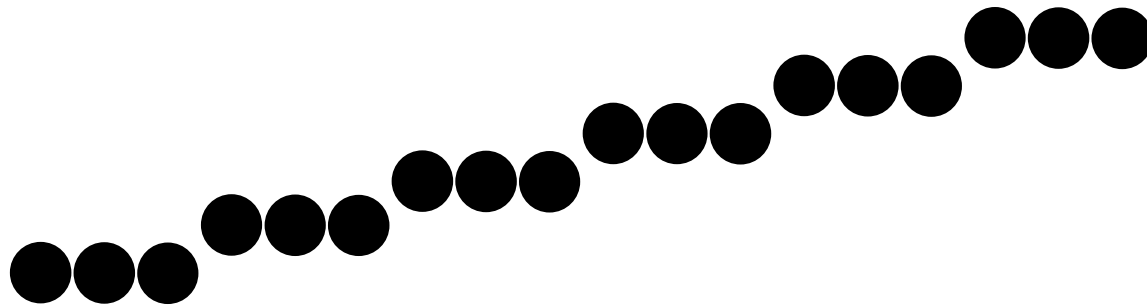
- Typical codes for marker type are the integers 1 through 5, specifying, respectively:
 1. a dot (.)
 2. a vertical cross (+)
 3. an asterisk (*)
 4. a circle (o)
 5. a diagonal cross (x).
- Displayed marker types are centred on the marker coordinates.

Contd.

- We set the marker size with function:
SetMarkerSizeScaleFactor ([ms](#))
- Where parameter marker size [ms](#) assigned a positive number according to need for scaling.
- For setting marker color we use function:
setPolymarkerColorIndex ([mc](#))
- Where parameter [mc](#) specify the color of the marker symbol.

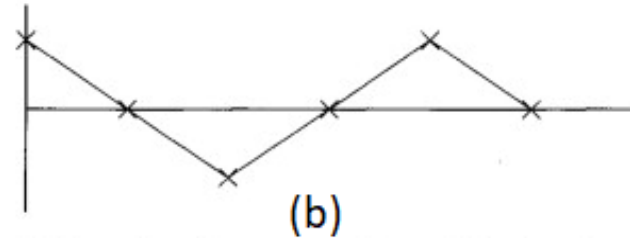
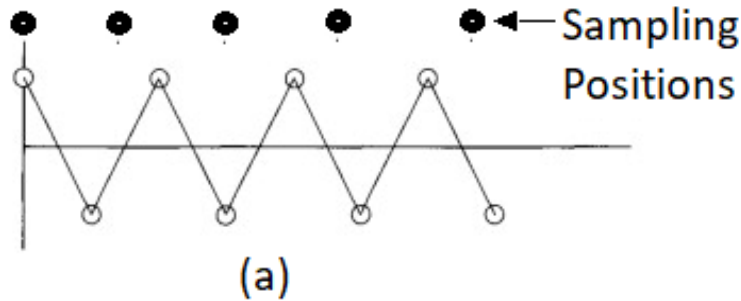
Aliasing

- Primitives generated in raster graphics by various algorithms have stair step shape or jagged appearance.
- Jagged appearance is due to integer calculation by rounding actual values.
- This distortion of actual information due to low frequency sampling is called **aliasing**.



Antialiasing

- Minimise effect of aliasing by some way is known as **antialiasing**.
- In periodic shape distortion may be occurs due to under sampling.



- To avoid losing information from such periodic objects, we need to set the sampling frequency to at least twice that of the highest frequency occurring in the object.
- This is referred to as the **Nyquist sampling frequency** (or Nyquist sampling rate):

$$f_s = 2f_{max}$$

Contd.

- In other words sampling interval should be no larger than one-half the cycle. Which is called **nyquist sampling interval**.

$$\Delta x_s = \frac{\Delta x_{cycle}}{2}$$

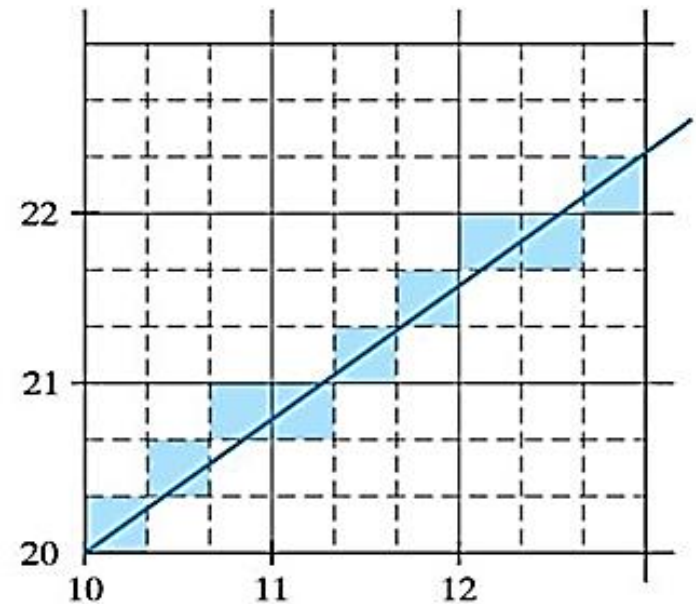
- One way to solve this problem is to display image on higher resolution.
- But it has also limit that how much large frame buffer we can maintain along with maintaining refresh rate 30 to 60 frame per second.
- And also on higher resolution aliasing will remains up to some extents.
- With raster systems that are capable of displaying more than two intensity levels (color or grayscale), we can apply antialiasing methods to modify pixel intensities.
- By appropriately varying the intensities of pixels along the boundaries of primitives, we can smooth the edges to lessen the aliasing effect.

Antialiasing Methods

1. Supersampling Straight Line Segments
2. Pixel-Weighting Masks
3. Area Sampling Straight Line Segments
4. Filtering Techniques
5. Pixel Phasing
6. Compensating For Line Intensity Differences
7. Antialiasing Area Boundaries

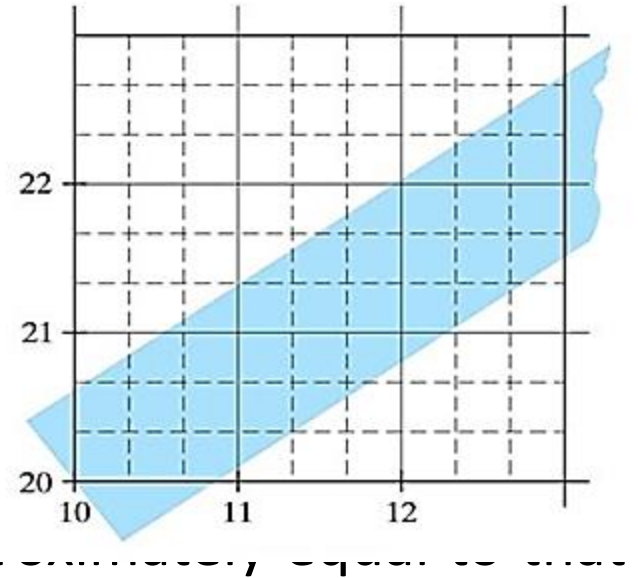
Supersampling Straight Line Segments

- For the greyscale display of a straight-line segment, we can divide each pixel into a number of sub pixels and determine the number of sub pixel along the line path.
- Then we set intensity level of each pixel proportional to number of sub pixel along the line path.
- E.g. in figure area of each pixel is divided into nine sub pixel and then we determine how many number of sub pixel are along the line (it can be 3 or 2 or 1 as we divide into 9 sub pixel).
- Based on number 3 or 2 or 1 we assign intensity value to that pixel.



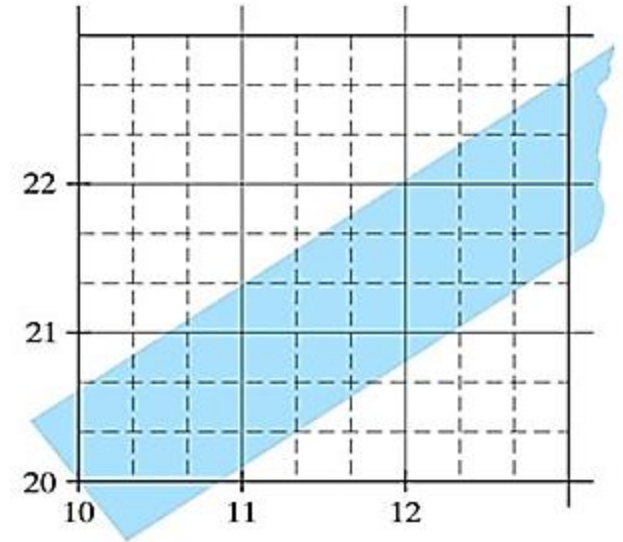
Contd.

- We can achieve four intensity levels by dividing pixels into four sub-pixels and five intensity levels by dividing each sub-pixel into five sub-sub-pixels.
- Lower intensity gives blurred effect and is called antialiasing.
- Other way is we considered pixel area as a whole and treated the line as a mathematical entity $v = y - x$.
- Actually, displayed lines have a width approx. equal to that of a pixel.
- If we take finite width of the line into account, we can perform supersampling by setting each pixel intensity proportional to the number of sub pixels inside the polygon representing the line area.



Contd.

- A sub pixel can be considered to be inside the line if its lower left corner is inside the polygon boundaries.
- Advantage of this is that it having number of intensity equals to number of sub pixel.
- Another advantage of this is that it will distribute total intensity over more pixels.
- E.g. in figure pixel below and left to (10, 20) is also assigned some intensity levels so that aliasing will reduce.
- For color display we can modify levels of color by mixing background color and line color.



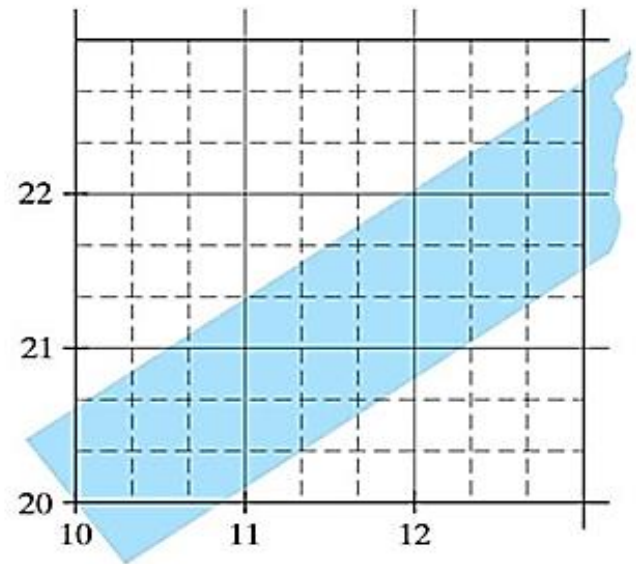
Pixel-Weighting Masks

- Supersampling methods are often implemented by giving more weight to sub pixels near the center of pixel area.
- As we expect centre sub pixel to be more important in determining the overall intensity of a pixel.
- For the 3 by 3 pixel subdivisions we have considered so far, a weighting scheme as in fig. could be used.
- The center sub pixel here is weighted four times that of the corner sub pixels and twice that of the remaining sub pixels.
- By averaging the weight of sub pixels which are along the line and assign intensity proportional to average weight will reduce aliasing effect.

1	2	1
2	4	2
1	2	1

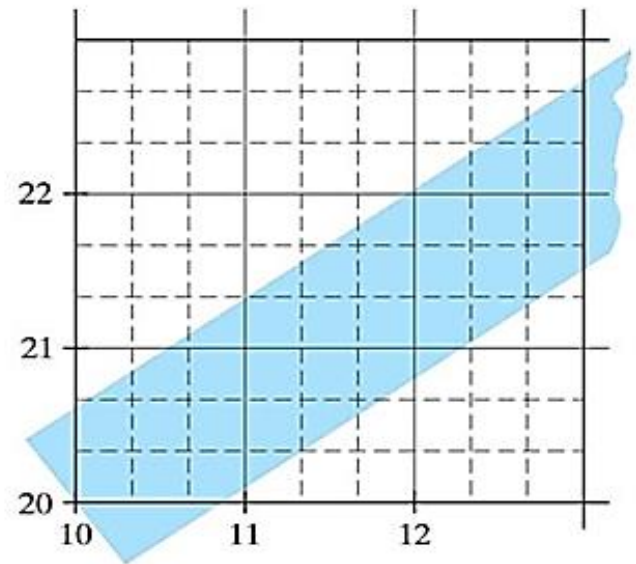
Area Sampling Straight Line Segments

- In this scheme we treat line as finite width rectangle, and the section of the line area between two adjacent vertical or two adjacent horizontal screen grid lines is then a trapezoids.
- Overlap areas for pixels are calculated by determining how much of the trapezoid overlaps each pixel in that vertical column (or horizontal row).
- E.g. pixel with screen grid coordinates (10, 20) is about 90 percent covered by the line area, so its intensity would be set to 90 percent of the maximum intensity.



Contd.

- Similarly, the pixel at (10 21) would be set to an intensity of about 15-percent of maximum.
- With color displays, the areas of pixel overlap with different color regions is calculated and the final pixel color is taken as the average color of the various overlap areas.



Filtering Techniques

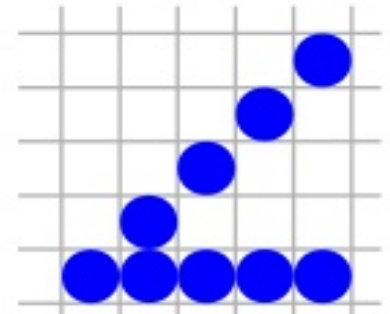
- It is more accurate method for antialiasing.
- Common example of filter is rectangular, conical and Gaussian filters.
- Methods for applying the filter function are similar to applying a weighting mask, but now we integrate over the pixel surface to obtain the weighted average intensity.
- For reduce computation we often use table look up.

Pixel Phasing

- On raster system if we pass the electron beam from the closer sub pixel so that overall pixel is shifted by factor $\frac{1}{4}$, $\frac{1}{2}$, or $\frac{3}{4}$ to pixel diameter.
- Where beam strike at that part of pixel get more intensity then other parts of the pixel and gives antialiasing effect.
- Some systems also allow the size of individual pixels to be adjusted as an additional means for distributing intensities.

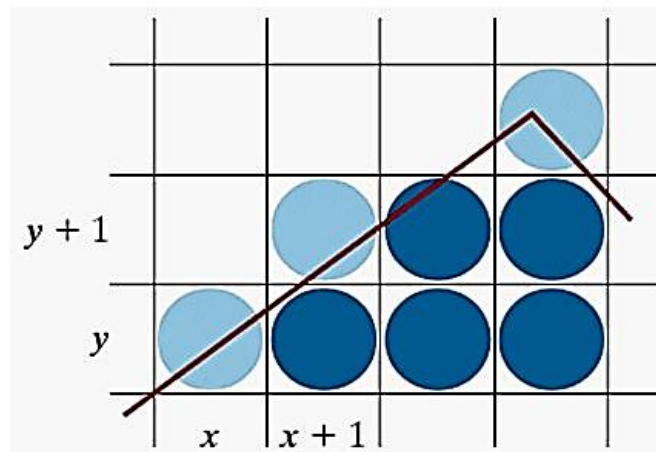
Compensating For Line Intensity Differences

- Antialiasing a line to soften the aliasing effect also compensates for another raster effect, illustrated fig.
- As both lines are display with same number of pixel and then also length of diagonal line is greater than horizontal line by factor $\sqrt{2}$.
- So that diagonal line is display with less intensity then horizontal line.
- For compensating this we display diagonal line with high intensity and horizontal line with low intensity so that this effect is minimize.
- In general we set intensity according to slope of the line.



Antialiasing Area Boundaries

- Methods we discuss for antialiasing line can also be applied for area boundary.
- If system capabilities permit the repositioning of pixels, area boundaries can be smoothen by adjusting boundary pixel positions.
- Other method is to adjust each pixel intensity at boundary position according to percent of area inside the boundary.



Contd.

- In fig. pixel at (x, y) is assigned half the intensity as its $\frac{1}{2}$ area is inside the area boundary.
- Similar adjustments based on percent of area of pixel inside are applied to other pixel.

