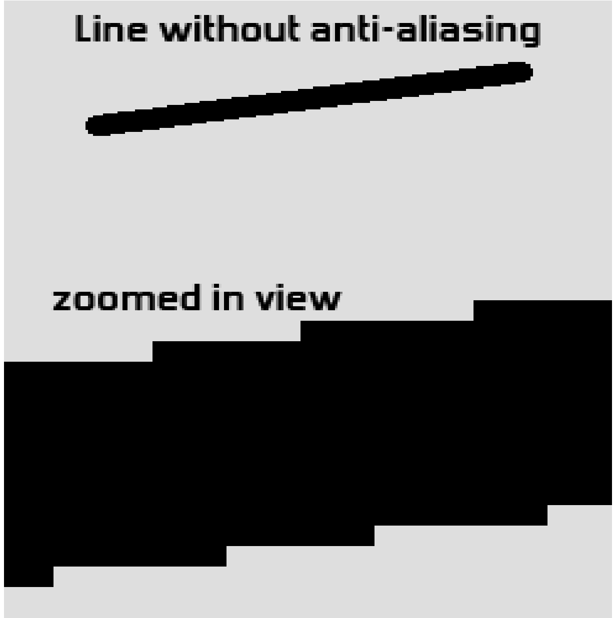
Antialiasing

Foley – 3.17 (Chapter 3)

What is Antialiasing?

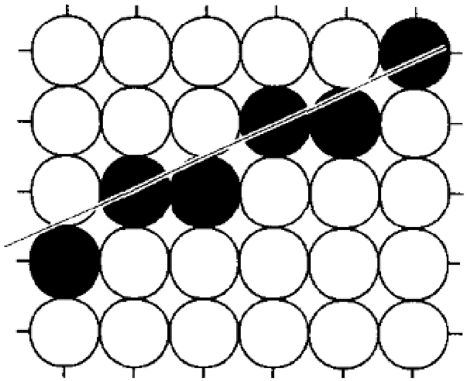
- Aliasing
  - Approximating a continuous entity with discrete samples
  - Jagging / staircasing effect
  - Result of an all-or-nothing approach to scan conversion
    - Each pixel is either colored or left unchanged



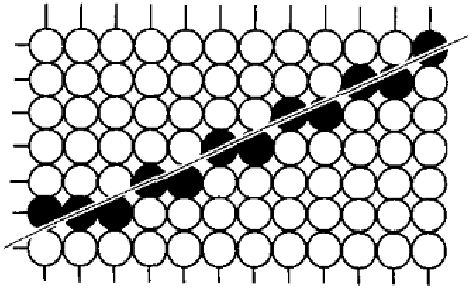
- Antialiasing
  - The application of techniques that reduce or eliminate aliasing

Antialiasing Techniques

- Increase Screen Resolution
  - Costly
    - Only diminishes, does not solve
    - Higher memory
    - More time for scan-conversion



(a) res. W X H



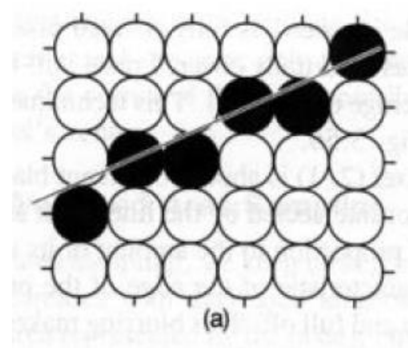
(b) res. 2W X 2H

- Area Sampling
  - Unweighted Area Sampling
  - Weighted Area Sampling

Unweighted Area Sampling

Basic Idea:

- Horizontal or vertical line passes through one pixel per row or column
- Lines at other angles go through multiple pixels per row or column



- A line is considered as a rectangle of a desired thickness covering a portion of the grid

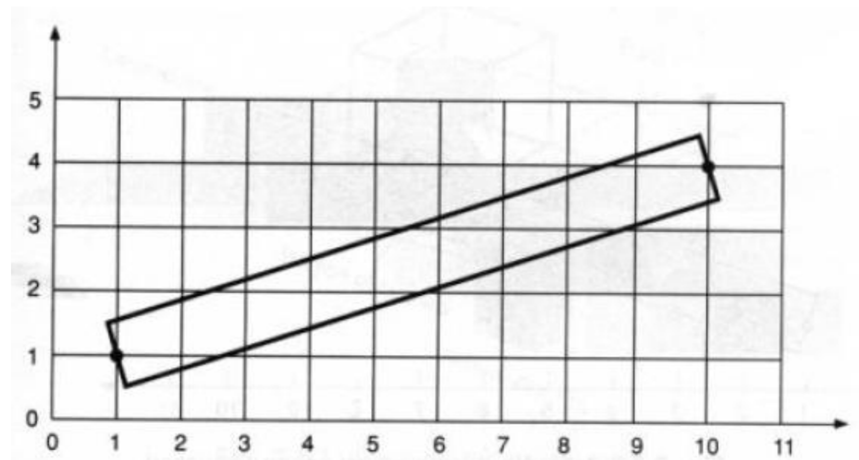


Figure 3.55 Line of nonzero width from point (1,1) to point (10,4).

Unweighted Area Sampling

Basic Idea:

- Pixels on the grid overlapped by the rectangle are colored to an appropriate intensity.
  General rule : A line contributes to each pixel's intensity an *amount proportional to the percentage* of the pixel's tile it covers.

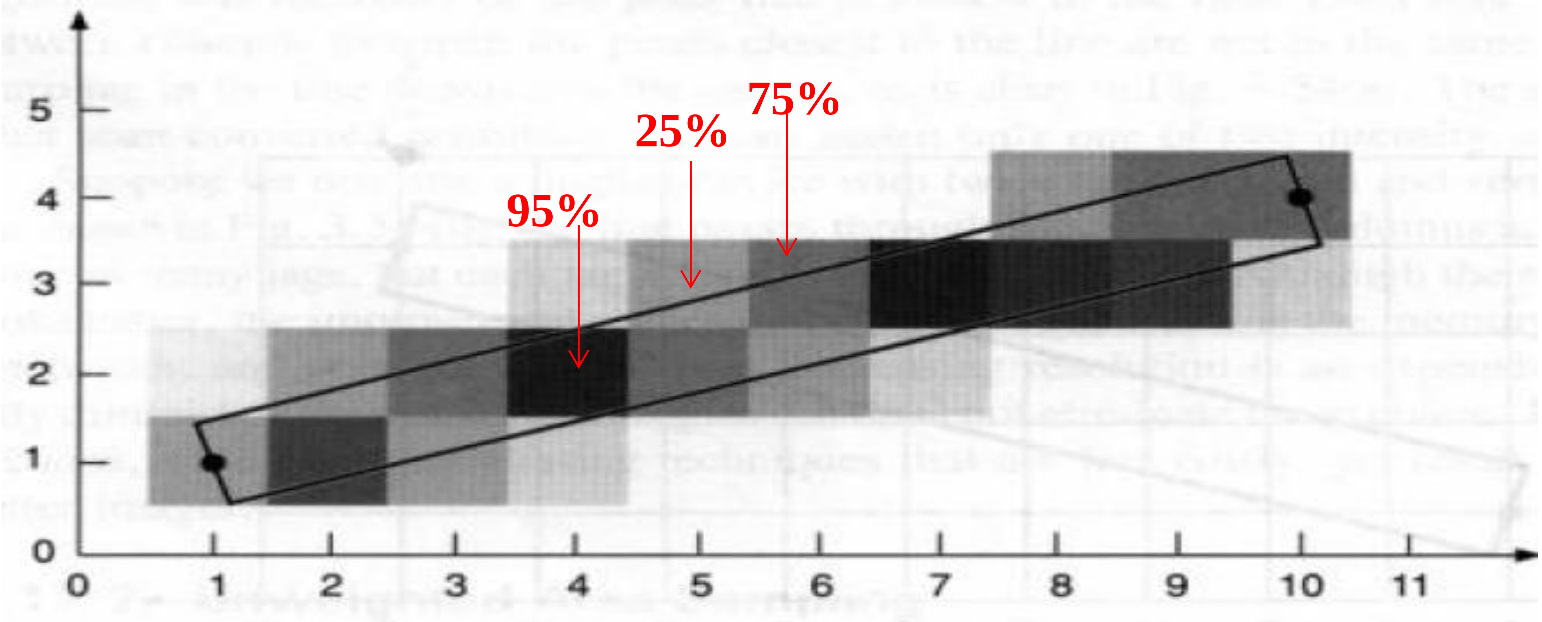


Fig. 3.56 Intensity proportional to area covered.

Unweighted Area Sampling

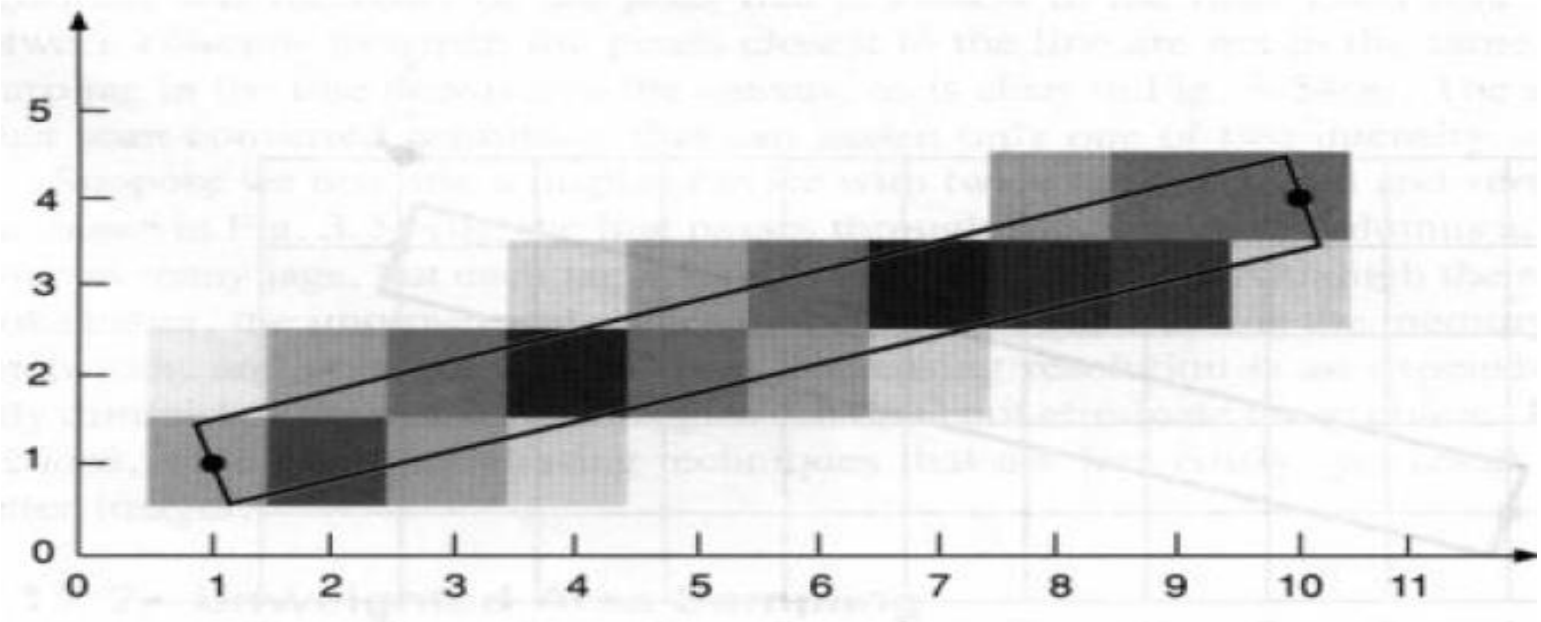


Fig. 3.56 Intensity proportional to area covered.

Intensity of pixel centered at (x,y) :

$I_{x,y} = I_{max} . dA . Weight$

dA = area overlap for pixel at (x,y)

Weight = 1 for unweighted area sampling

Unweighted Area Sampling

Properties :

1. The intensity decreases with decreased area overlap.
2. Primitive cannot influence the intensity at a pixel if the primitive does not intersect it.
3. Equal areas contribute equal intensity, regardless of **distance from pixel center to area**.

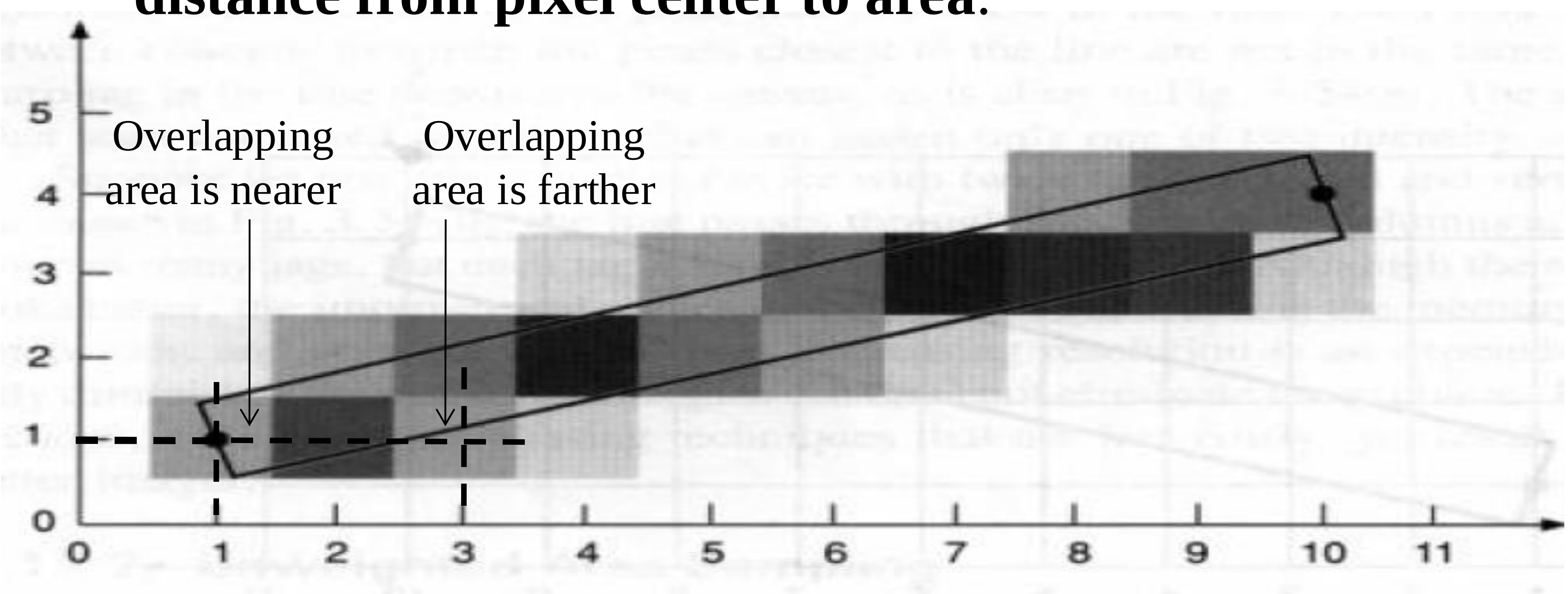


Fig. 3.56 Intensity proportional to area covered.

Unweighted Area Sampling

- Weighting Function
  - Uniform weight for equal overlapping area

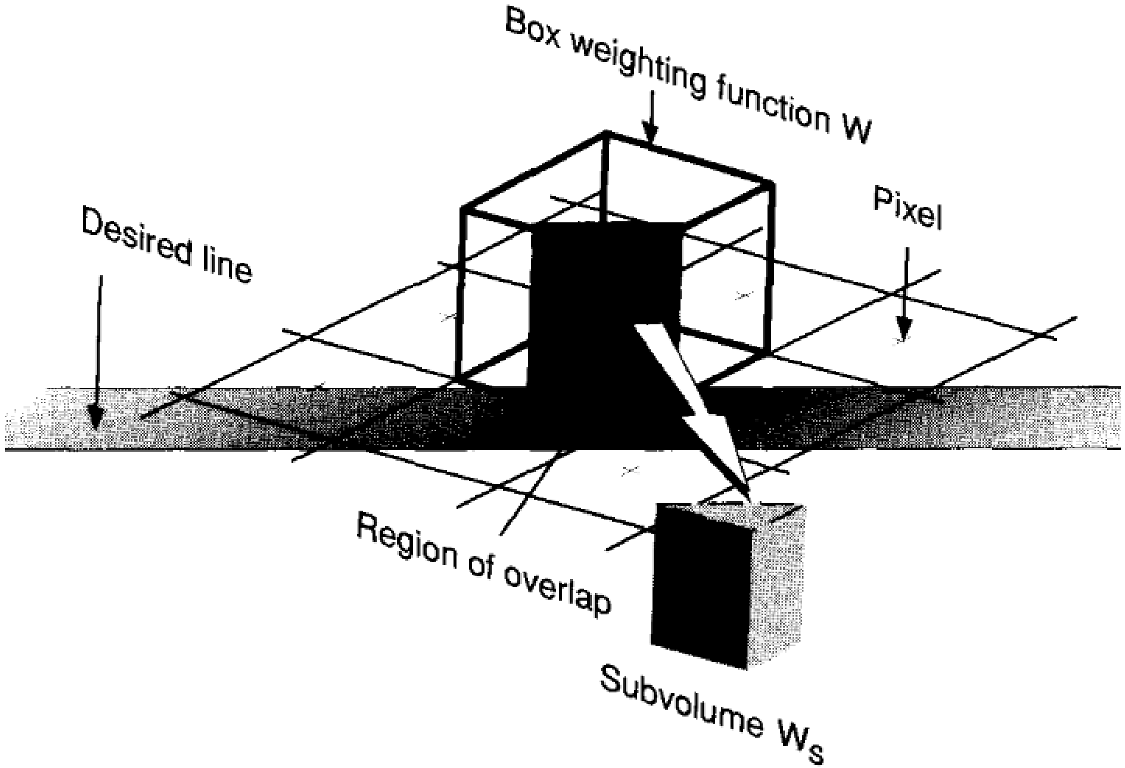


Fig. 3.57 Box filter for square pixel.

Weighted Area Sampling

Properties :

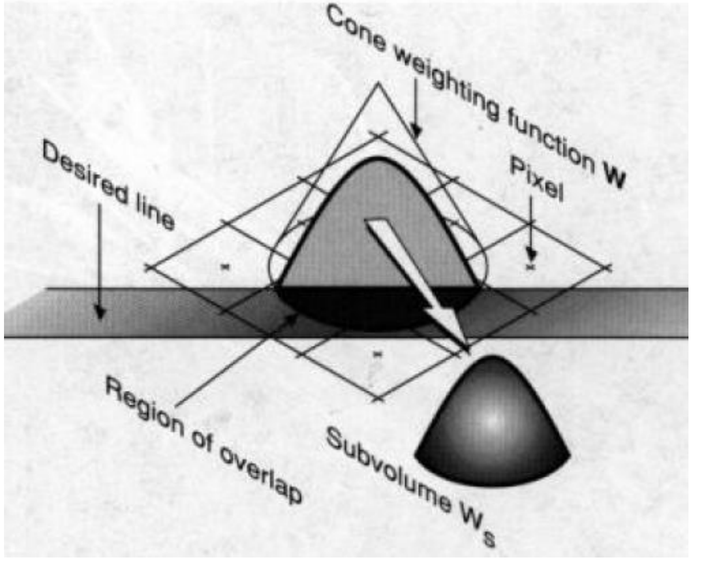
- First and second properties are unchanged
- Property 3 has been modified as:
  - *Equal areas contribute unequally* : A small area closer to the pixel center has greater influence than does one at a greater distance.

Weighted Area Sampling

- Remember, the intesity for unweighted area sampling

  Intensity of pixel centered at (x,y)

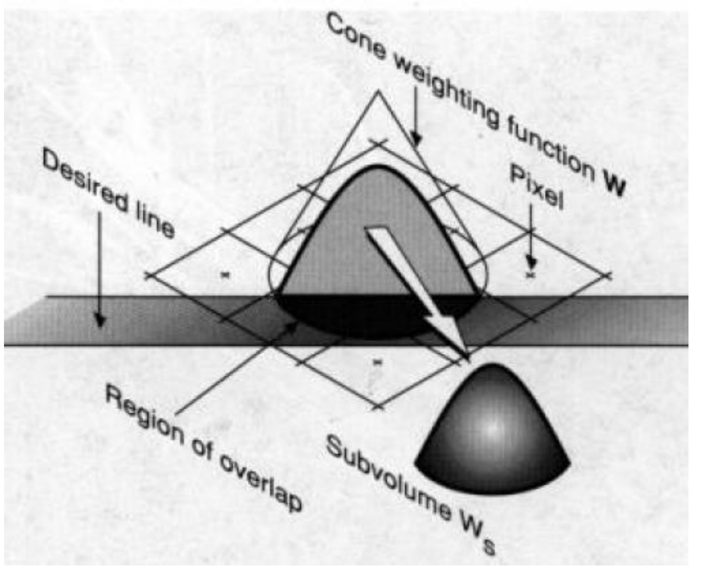
  $I_{x,y} = I_{max} . dA .$ **Weight**



- Now, we need to construct a weighting function that gives less weight to area further away from pixel center than it does to equal but closer one.
- **So weighting must be a decreasing function of distance.**
- A **circular cone** is an example of such a function.

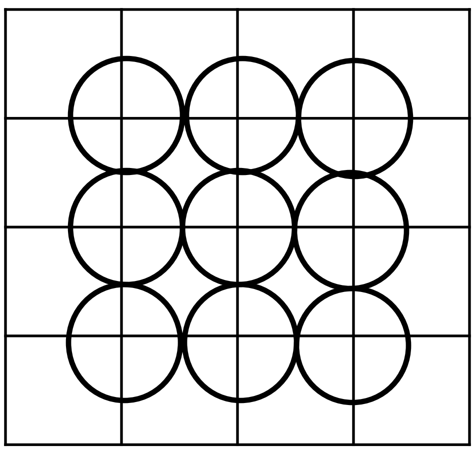
Weighted Area Sampling

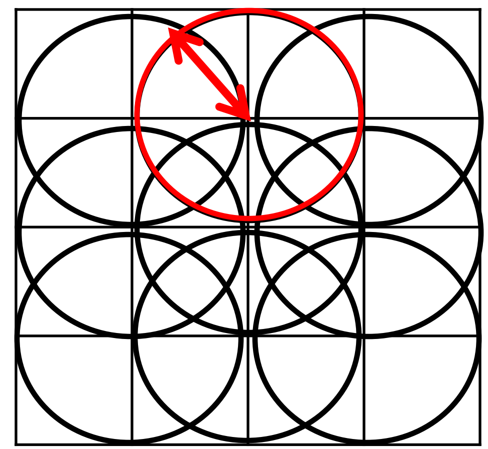
- To obtain such weighting, **pixels are considered as circles**.
- Each pixel is considered to have a conical volume.
- Intensity at pixel (x,y)
  - $I_{x,y} = I_{max} . W_s$
  - where W_s is the sub volume of cone



Weighted Area Sampling

- Each circular pixel has radius of **1 unit** so the pixels will be overlapping
  - Ensures that no area in the grid is left uncovered by the pixels





Weighted Area Sampling

- Each line intersects more than one pixel
  - Usually three pixels

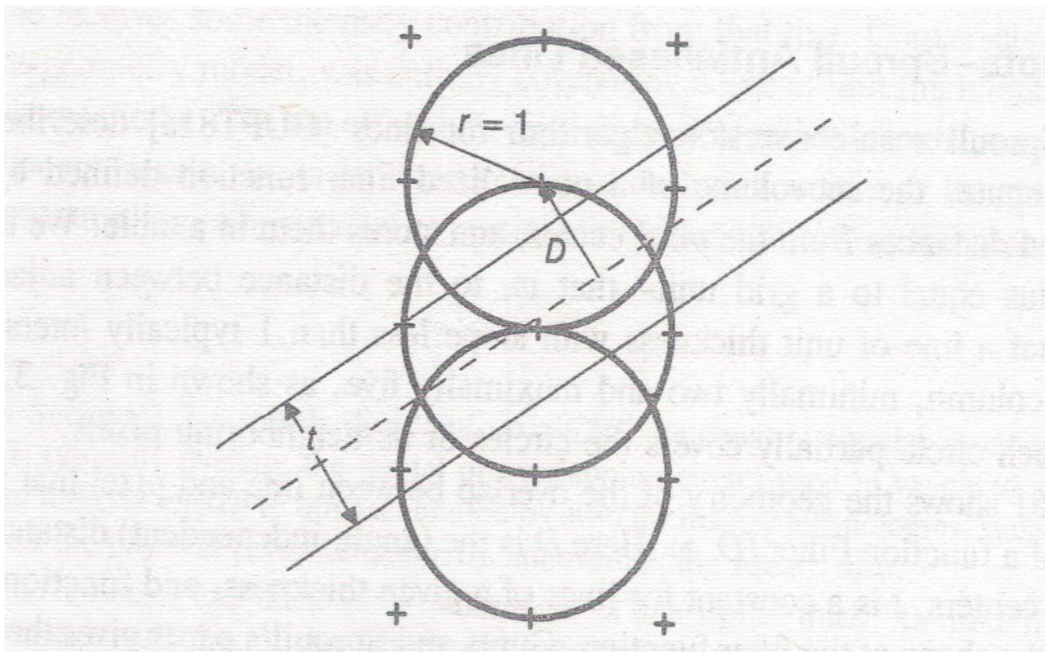


Fig. 3.60 One-unit-thick line intersects 3-pixel supports.

Gupta-Sproull Weighting Function

- Uses **Gupta-Sproull** weighting function, Filter(*D*, *t*)
  - D = Perpendicular distance between pixel center and the line center
  - t = Constant for a line of given thickness
  - A table takes distance D as index and provides corresponding Filter(*D*, *t*) value i.e. Intensity.

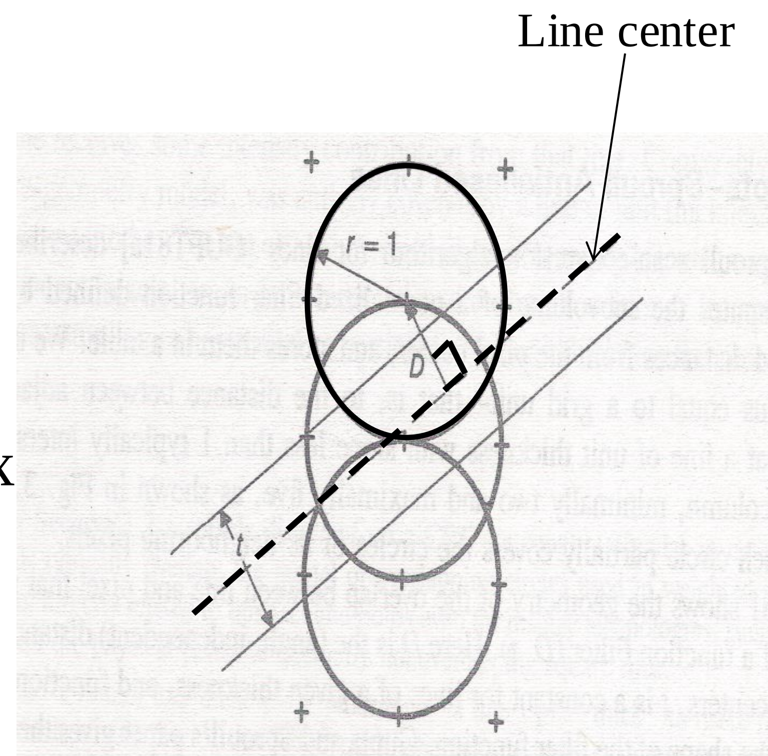
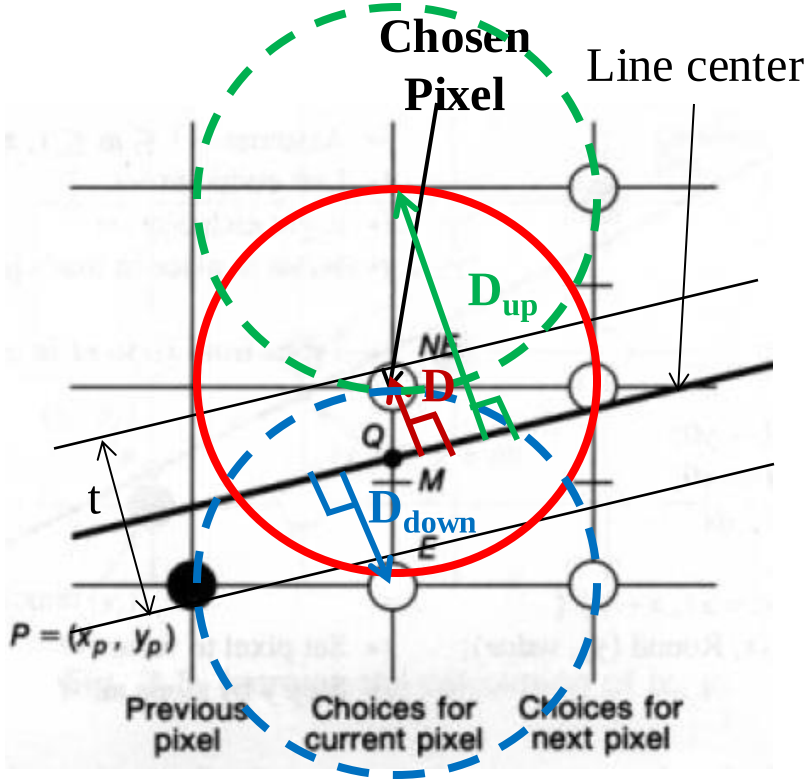


Fig. 3.60 One-unit-thick line intersects 3-pixel supports.

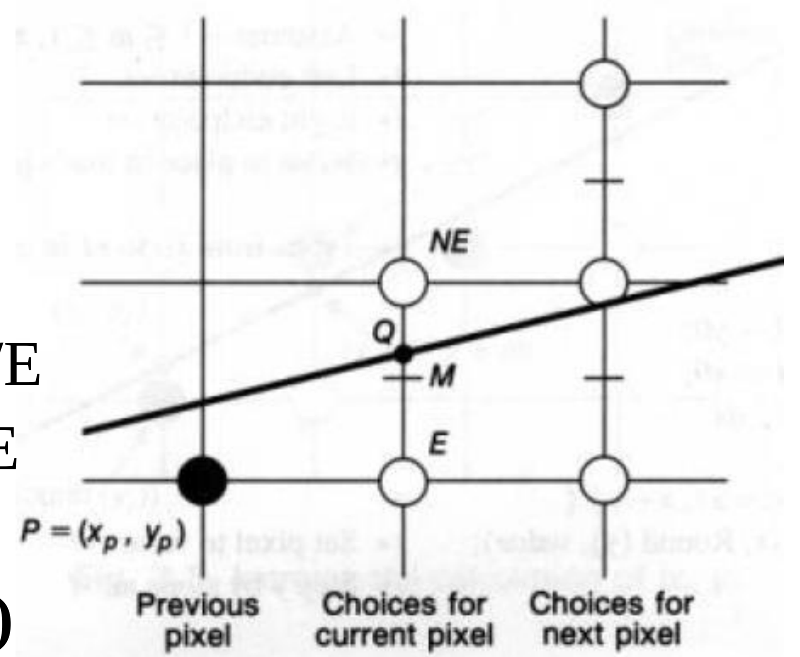
Gupta-Sproull Antialiased Lines

- Modifies the midpoint line algorithm for scan converting lines
- The decision variable d is used as before to choose between E and NE
- Then Distance D between the chosen pixel to the line is computed
- Using D, distances between its two vertical neighbors and the line: D_{up}, and D_{down} are computed
- If the slope > 1, distances between its two vertical neighbors and the line: D_{right}, and D_{left} are computed
- Then intensity of chosen pixel and its two vertical neighbors are set, on the basis of the distances from these pixels to the line, using Filter(D, t)



Midpoint Line Algorithm - Recap

- Initially we are at $P(x_p,y_p)$
- Choose between **NE** and **E**
- Midpoint **M** is $\mathbf{(x_p+1, y_p+1/2)}$
- **M lies on which side of the line?**
  - M is on the line → any pixel NE/E
  - M is below the line → choose NE
  - M is above the line → choose E
- **Find the equation of the line f (x,y)**
  - f (x,y) = ax + by + c
    a = dy; b = -dx; c = dx.B (B is the y-intercept)
  - Let, d = f (M) = f $(x_p+1, y_p+1/2)$
  - d = 0 → M is on the line
    d > 0 → M is below the line
    d < 0 → M is above the line



Midpoint Line Algorithm - Recap

- Initial value of decision variable $\mathbf{d_{start}}$
- Start point (x_o, y_o)
- $d_{start} = f(x_o+1,\ y_o+1/2)$

  $= f(x_o, y_o) + dy - dx/2$

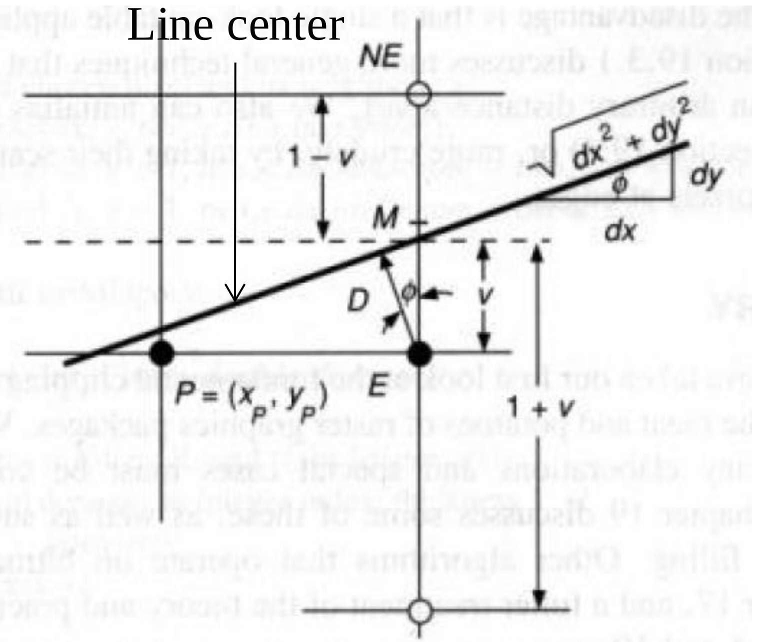
  $= dy - dx/2$

Here the fraction in **dx/2** can be avoided by

considering $\mathbf{f(x,y) = 2(ax+by+c)}$

Gupta-Sproull: Calculation of D and v

- The decision variable d is used as before to choose between E and NE
  - D = perpendicular distance between chosen pixel (E) and line center
  - v = Vertical distance between chosen pixel (E) and line center

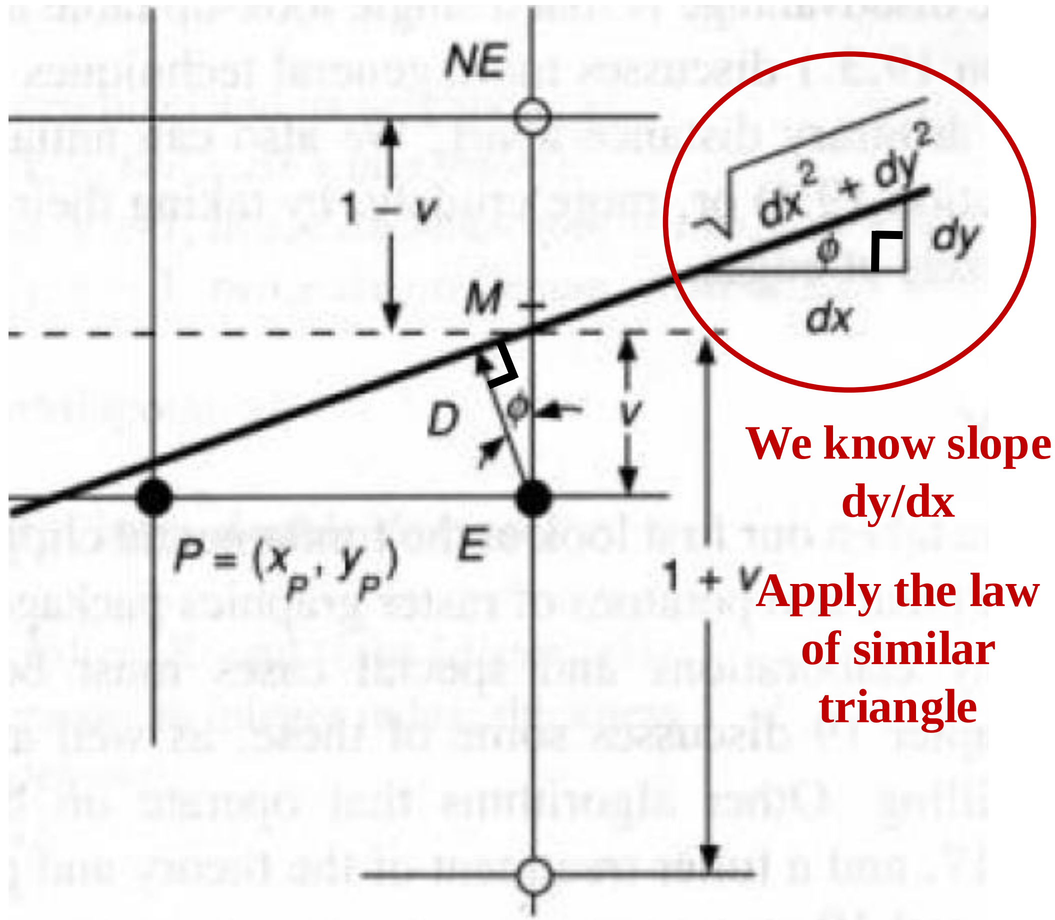


Calculation of D

- For E,
  - D = perpendicular distance between chosen pixel (E) and the line center

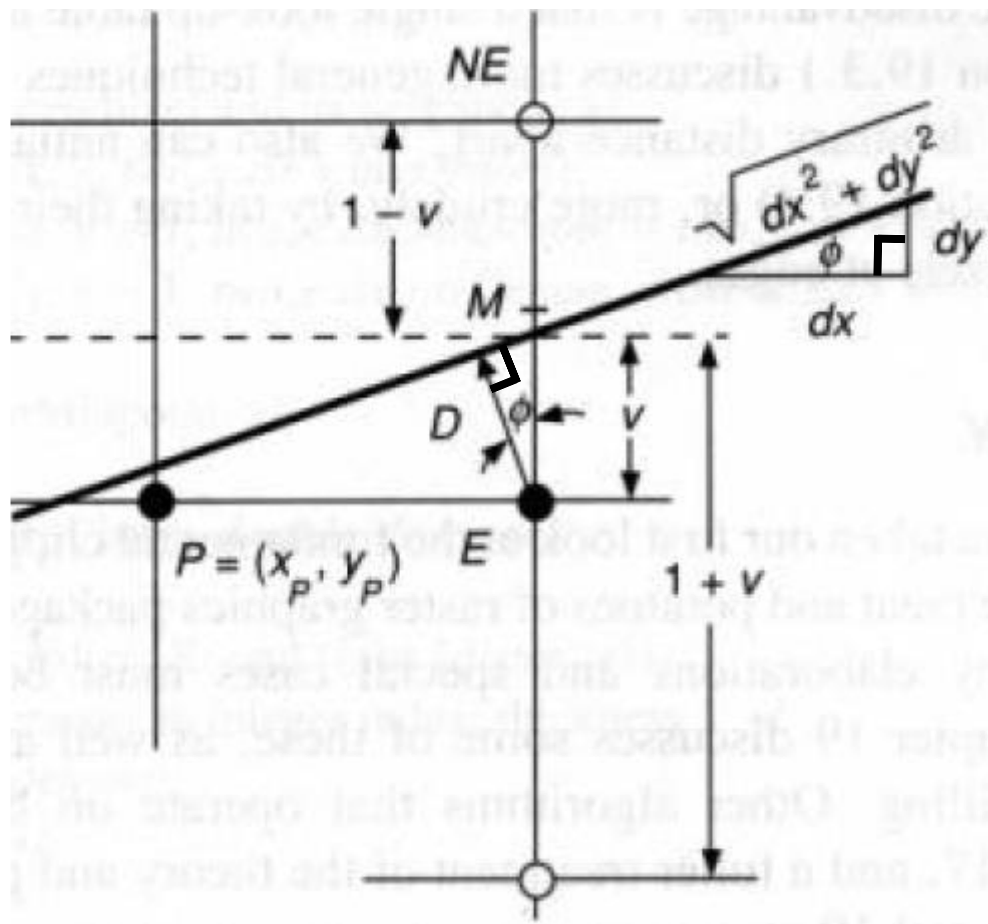
$$D = v\cos\phi$$

$$= \frac{vdx}{\sqrt{dx^2 + dy^2}}$$



Calculation of v

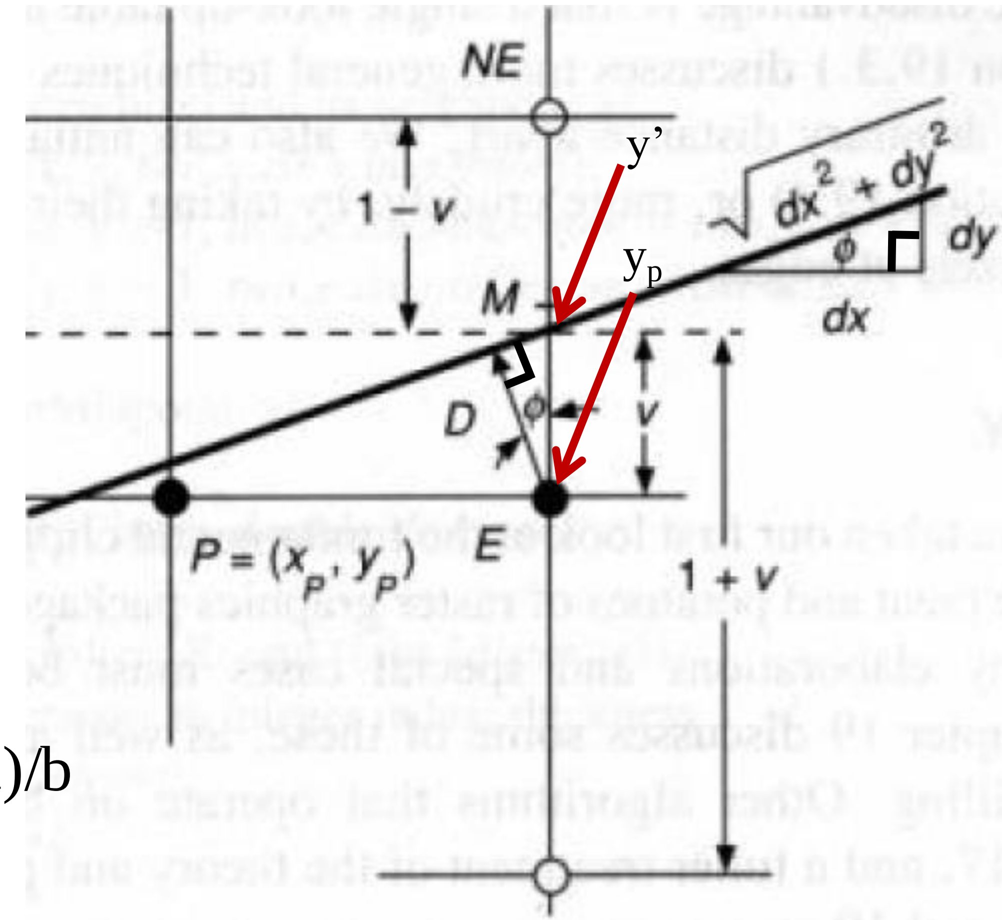
- For E,
  - v = vertical distance between chosen pixel (E) and line center
    - +ve if line center passes above the chosen pixel
    - -ve if line center passes below the chosen pixel
    - consider the absolute value



Calculation of v

- v = vertical distance between chosen pixel (E) and line center
  - vertical distance = subtraction of y of E from y of the intersecting point
- Go from $P = (x_p, y_p)$ to $E = (x_p+1, y_p)$
- v = y of line at x_p+1 - y of E at x_p+1 $= y' - y_p$
- How to get y' ?
  - Equation of Line, $ax+by+c=0$

    $\Rightarrow y = -(ax+c)/b$
  - So, $y' = -(a(x_p+1)+c)/b$



Calculation of v

- From $v = y' - y_p$ we can derive, $vdx = a(x_p+1) + by_p + c$

For avoiding fraction,

$2vdx = 2(a(x_p+1) + by_p + c)$

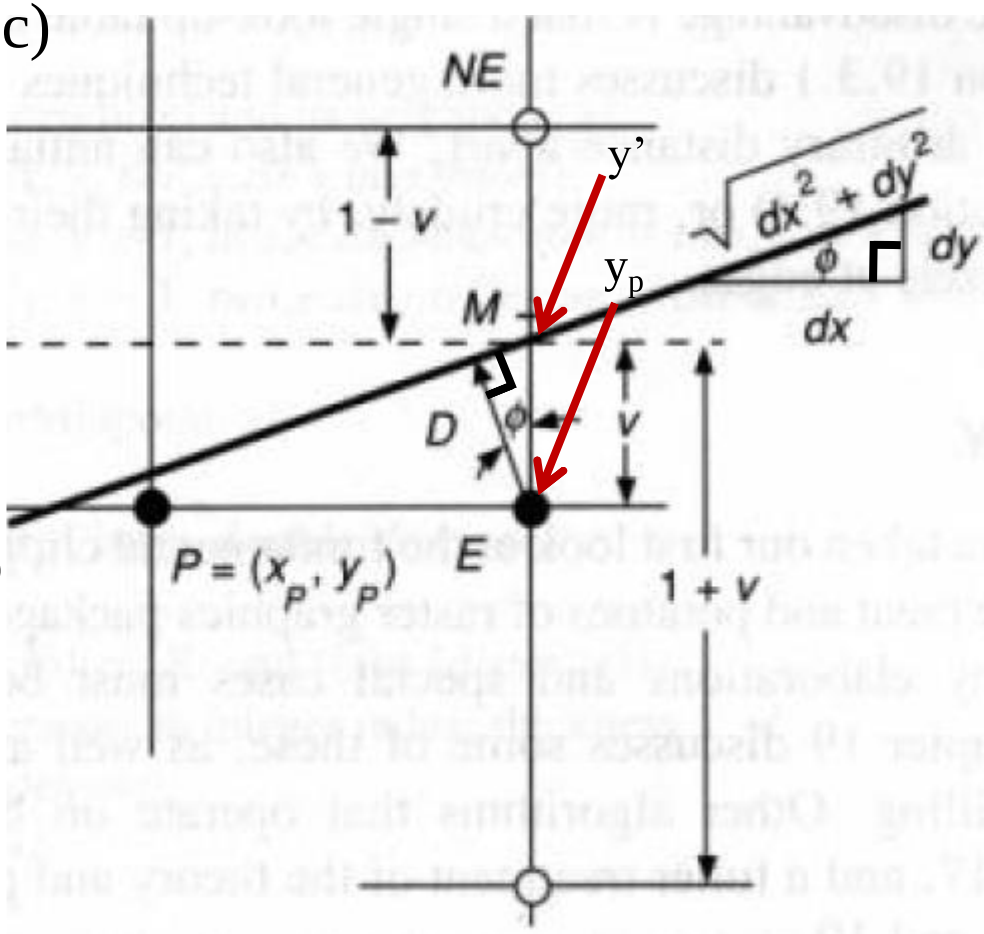
$2vdx = f(x_p+1, y_p)$

Present In terms of
$M = d_{old} = f(x_p+1, y_p+1/2)$

$2vdx = \mathbf{2(a(x_p+1) + b(y_p+½) + c)} - b$

$2vdx = f(x_p+1, y_p+1/2) - b$
$= d - b = d + dx$

$$vdx = \frac{d + dx}{2}$$



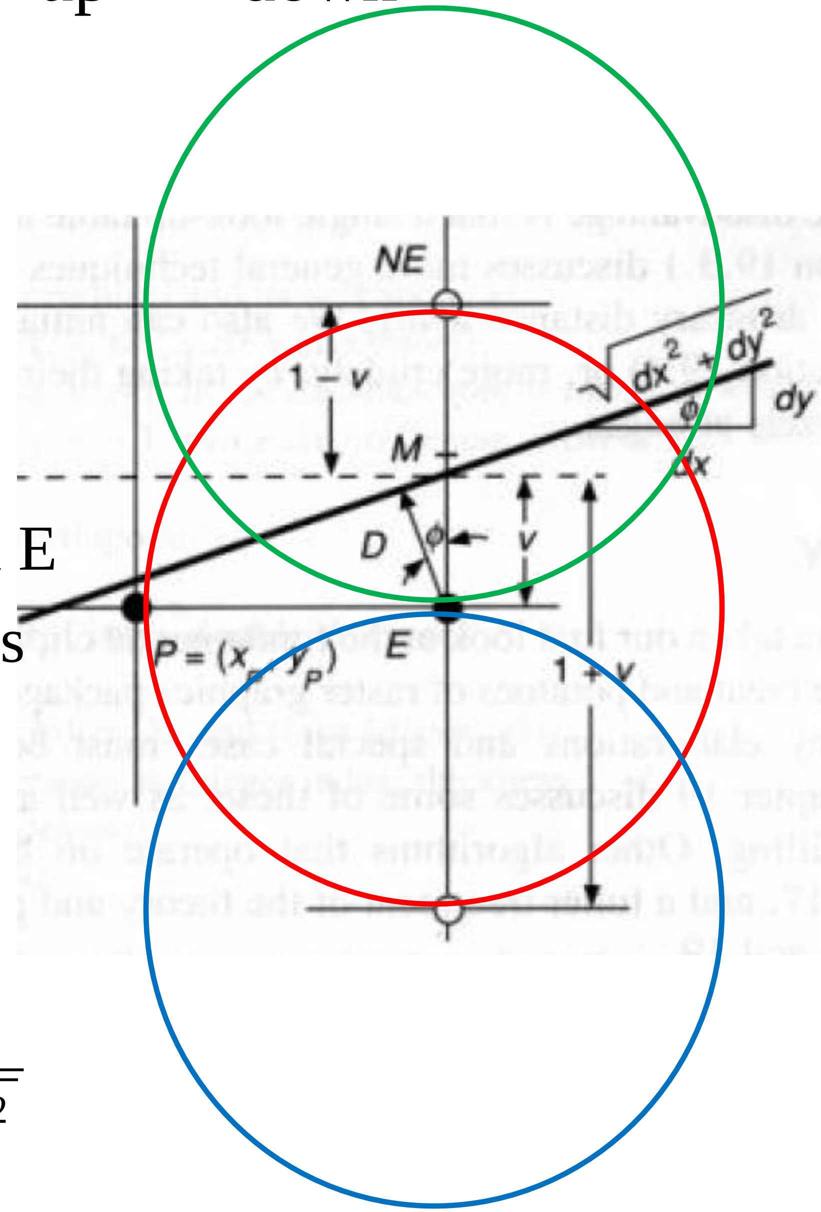
Calculation of D_{up}, D_{down}

$$vdx = \frac{d + dx}{2}$$

$$D = \frac{vdx}{\sqrt{dx^2 + dy^2}} = \frac{dx + d}{2\sqrt{dx^2 + dy^2}}$$

So we have D, and v for selected pixel E

Now, find D for upper and lower pixels



$$D_{up} = \frac{2(1 - v)dx}{2\sqrt{dx^2 + dy^2}} = \frac{2dx - 2vdx}{2\sqrt{dx^2 + dy^2}}$$

$$D_{down} = \frac{2(1 + v)dx}{2\sqrt{dx^2 + dy^2}} = \frac{2dx + 2vdx}{2\sqrt{dx^2 + dy^2}}$$

Calculation of D and v

- For NE,

$2vdx = f(x_p+1, y_p+1)$

$= \mathbf{2(a(x_p+1) + b(y_p+1) + c)}$

$vdx = a(x_p+1) + b(y_p+1) + c$

$= (a(x_p+1) + b(y_p+1/2) + c\ ) + ½\ b$

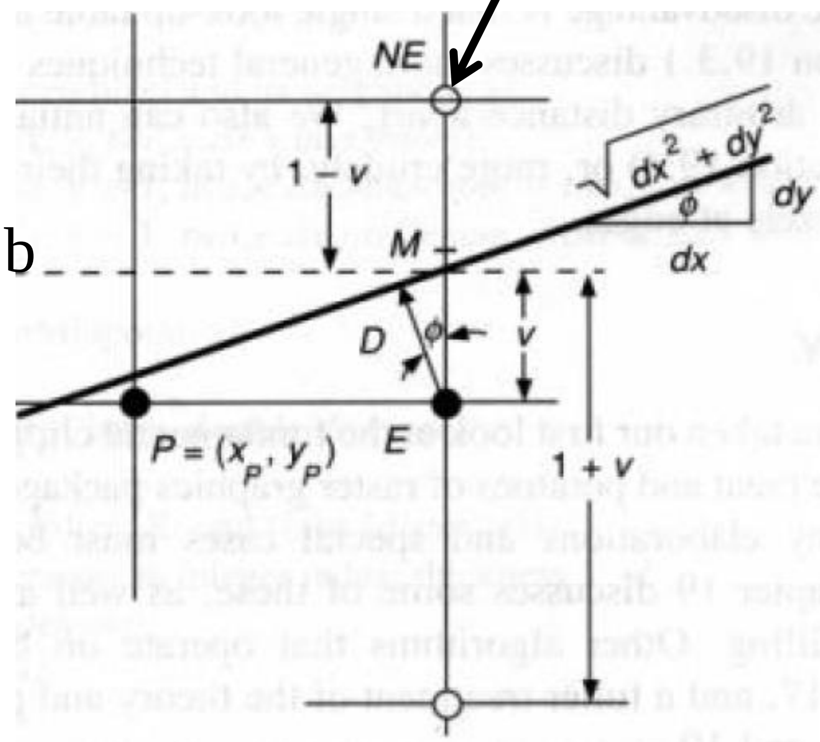
$= f(x_p+1, y_p+1/2)/2 + ½\ b$

$= d/2 + b/2 = (d+b)/2$

$$vdx = \frac{d - dx}{2}$$

$$D = \frac{vdx}{\sqrt{dx^2 + dy^2}} = \frac{d - dx}{2\sqrt{dx^2 + dy^2}}$$

Chosen Pixel



Also find the values of D for The pixel above NE and The pixel below NE

Gupta-Sproull Algorithm

- Modification of Midpoint line algorithm
- Pseudocode: Figure 3.62 of textbook

```
if d < 0 then
  begin                                          {Choose E}
    two_v_dx := d + dx;
    d := d + incrE;
    x := x + 1
  end
else
  begin                                          {Choose NE}
    two_v_dx := d - dx;
    d := d + incrNE;
    x := x + 1;
    y := y + 1
  end;
...
IntensifyPixel (x, y, two_v_dx * invDenom);
IntensifyPixel (x, y + 1, two_dx_invDenom - two_v_dx * invDenom);
IntensifyPixel (x, y - 1, two_dx_invDenom + two_v_dx * invDenom)
```

Thank you ☺