Rasterization or Scan Conversion

- Drawing lines on a raster grid implicitly involves approximation.
- The general process : *rasterization* or *scan-conversion*.
- What is the best way to draw a line from the pixel (x1,y1) to (x2,y2)?
- Such a line should ideally have the following properties.
  - straight
  - pass through endpoints
  - smooth
  - independent of endpoint order
  - uniform brightness
  - brightness independent of slope
  - efficient

Line Drawing - Algorithm 1

A Straightforward Implementation

```
Drawline(x1,y1,x2,y2)
int x1,y1,x2,y2;
{
  float y;
  int x;

  for (x=x1; x<=x2; x++) {
    y = y1 +  (x-x1)*(y2-y1)/(x2-x1)
    SetPixel(x, Round(y) );
  }
}
```


Line Drawing - Algorithm 2

A Better Implementation

```
DrawLine(x1,y1,x2,y2)
int x1,y1,x2,y2;
{
  float m,y;
  int dx,dy,x;
  dx = x2 - x1;
  dy = y2 - y1;
  m = dy/dx;
  y = y1 + 0.5;
  for (x=x1; x<=x2; x++) {
    SetPixel(x, Floor(y) );
    y = y + m;
  }
}
```


Line Drawing Algorithm Comparison

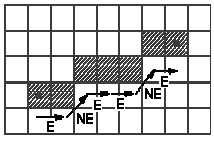
- Advantages over Algorithm 1
  - eliminates multiplication
  - improves speed
- Disadvantages
  - round-off error builds up
  - get pixel drift
  - rounding and floating point arithmetic still time consuming
  - works well only for |m| < 1
  - need to loop in y for |m| > 1
  - need to handle special cases

Point and line drawing routines

- Implicit: f(x,y) = 0
  - F(x,y) = (x-x0)dy - (y-y0)dx
  - if F(x,y) = 0 then (x,y) is on line
  - F(x,y) > 0 then (x,y) is below line
  - F(x,y) < 0 then (x,y) is above line

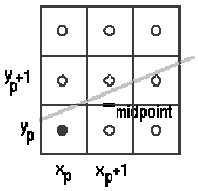
Line Drawing - Midpoint Algorithm

- The Midpoint or Bresenham's Algorithm for |m|<=1
  - The midpoint algorithm is better than the previous algorithms in that it uses only integer calculations. It treats line drawing as a sequence of decisions. For each pixel that is drawn the next pixel will be either N or NE, as shown below.



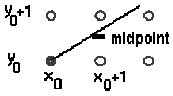
Midpoint Algorithm

- The midpoint algorithm makes use of the the implicit definition of the line, F(x,y) = 0. The N/NE decisions are made as follows.
- $d = F(x_p + 1, y_p + 0.5)$
  - if $d < 0$ line below midpoint choose E
  - if $d > 0$ line above midpoint choose NE
- if E is chosen
  - $d_{new} = F(x_p + 2, y_p + 0.5)$
  - $d_{new} - d_{old} = F(x_p + 2, y_p + 0.5) - F(x_p + 1, y_p + 0.5)$
  - $\Delta d = d_{new} - d_{old} = dy$



Midpoint Algorithm

- If NE is chosen
  - $d_{new} = F(x_p + 2, y_p + 1.5)$
  - $\Delta d = dy - dx$
- Initialization
  - $d_{start} = F(x_0 + 1, y_0 + 0.5) = (x_0 + 1 - x_0)\, dy - (y_0 + 0.5 - y_0)dx$
    $= dy - dx/2$
- Integer only algorithm
  - $F'(x,y) = 2F(x,y)$; $d' = 2d$
  - $d'_{start} = 2dy - dx$
  - $\Delta d' = 2\Delta d$



Midpoint Algorithm for x1 < x2 and slope <= 1

```
drawline(x1, y1, x2, y2, colour)
int x1, y1, x2, y2, colour;
{
  int dx, dy, d, incE, incNE, x, y;

  dx = x2 - x1;
  dy = y2 - y1;
  d = 2*dy - dx;
  incE = 2*dy;
  incNE = 2*(dy - dx);
  y = y1;
  for (x=x1; x<=x2; x++) {
    setpixel(x, y, color);
    if (d>0) {
      d = d + incNE;
      y = y + 1;
    } else {
      d = d + incE;
    } } }
```


General Bresenham's Algorithm

- To generalize to lines with arbitrary slope
  - consider symmetry between various octants and quadrants
  - for m > 1, interchange roles of x and y, that is step in y direction, and decide whether x values is above or below line
  - if m > 1, and right endpoint is the first point, both x and y decrease. To ensure uniqueness, independent of direction, always choose upper (or lower) point if the line goes through the mid-point
  - handle special cases without invoking algorithm: horizontal, vertical and diagonal lines

Additional Issues

- End-point order
  - cannot just interchange end-points
  - does not work when we use line styles since we need the pattern to go the same way on all segments of a polygon
- later we must deal with issue of clipping
- varying the intensity of a line with the slope
  - consider horizontal line and diagonal line
  - both have same number of pixels
  - diagonal $\sqrt{2}$ times horizontal line in length
  - intensity per unit length less for diagonal