Scan Conversion:
Drawing Lines on Raster Display
bresenham
Raster Display

- Discrete grid of elements (frame buffers, pixels)
- Shapes drawn by setting the “right” elements
- Frame buffer is scanned, one line at a time, to refresh the image (as opposed to vector display)

Properties
- Difficult to draw smooth lines
- Displays only a discrete approximation of any shape
- Refresh of entire frame buffer
Terminology

- **Pixel**: Picture element
  - Smallest accessible element in picture
  - Usually rectangular or circular

- **Aspect Ratio**: Ratio between physical dimensions of pixel
  - (not necessarily 1 !!)

- **Dynamic Range**: Ratio between minimal (not zero!) and maximal light intensity emitted by displayed pixel
Terminology (cont’d)

- **Resolution**: number of distinguishable rows and columns on device. Measured in:
  - Absolute values (1K x 1K)
  - Relative values (300 dots per inch)

- **Screen Space**: discrete 2D Cartesian coordinate system of screen pixels

- **Object Space**: 3D Cartesian coordinate system of the universe where the objects (to be displayed) are embedded
Basic Line Drawing

Assume $x_1 < x_2$ and line slope absolute value is $\leq 1$

Line $(x_1, y_1, x_2, y_2)$
begin
  float $dx, dy, x, y, slope$;
  $dx \leftarrow x_2 - x_1$;
  $dy \leftarrow y_2 - y_1$;
  $slope \leftarrow \frac{dy}{dx}$;
  $y \leftarrow y_1$
  for $x$ from $x_1$ to $x_2$ do
  begin
    PlotPixel ($x$, Round ($y$)) ;
    $y \leftarrow y + slope$;
  end ;
end ;

Questions:

Can this algorithm use integer arithmetic?

Does it accumulate error?

Is the error significant?
Recursive Line Drawing

Simple, recursive, integer, line drawing:

```plaintext
Line ( x_1, y_1, x_2, y_2 )
begin
  int x, y;
  x ← (x_1 + x_2) / 2;
  y ← (y_1 + y_2) / 2;
  if ((x = x_1 and y = y_1) or
      (x = x_2 and y = y_2))
    return;
  else begin
    PlotPixel (x, y);
    Line (x_1, y_1, x, y);
    Line (x_1, y_1, x, y);
    Line (x, y, x_2, y_2);
  end;
end;
```

Questions:

How does the algorithm work?

Does the algorithm accumulate error?

Is it significant?
Recursive Line Drawing (cont’d)

More Potential Problems:

- Line is not drawn sequentially
- Function call for each pixel drawn

*We want a faster and accurate algorithm!*
Midpoint (Bresenham) Algorithm

- **Assumptions:**
  \[ x_2 > x_1, \ y_2 > y_1 \quad \text{and} \quad \frac{dy}{dx} = \frac{y_2 - y_1}{x_2 - x_1} < 1 \]

- **Idea:**
  - Define error function
  - Proceed along the line incrementally
  - Select direction that minimizes accumulated error

**Definitions**
\[
\tau = \left\{(x, y) \mid ax + by + c = xdy - ydx + c = 0\right\}
\]
\[
d(x, y) = 2(xdy - ydx + c)
\]
Midpoint (Bresenham) Algorithm (cont’d)

\[
d(x, y) = 2(xdy - ydx + c)
\]

- Given point \( P=(x,y) \), \( d(x,y) \) is the signed distance of \( P \) to \( \tau \) (up to normalization factor)

- \( d \) is zero for \( P \in \tau \)
  \( \Rightarrow d \) may serve as error function to be minimized

- Starting point satisfies \( d(x_1,y_1) = 0 \)

- Each step moves right (east) or upper right (northeast)
Midpoint Line Drawing (cont’d)

- Sign of $d(x_1+1, y_1+\frac{1}{2})$ indicates whether to move east or northeast

- At $(x_1, y_1)$
  
  $d_{\text{start}} = d(x_1+1, y_1+\frac{1}{2}) = 2dy - dx$

- Increment in $d$ (after each step)
  
  **Move east**
  
  $\Delta_e = d(x+2, y+\frac{1}{2}) - d(x+1, y+\frac{1}{2}) = 2((x+2)dy - (y+\frac{1}{2})dx + c) - 2((x+1)dy - (y+\frac{1}{2})dx + c) = 2dy$

  **Move northeast**

  $\Delta_{ne} = d(x+2, y+3/2) - d(x+1, y+\frac{1}{2}) = 2((x+2)dy - (y+3/2)dx + c) - 2((x+1)dy - (y+\frac{1}{2})dx + c) = 2(dy - dx)$
# Midpoint Line Algorithm

**Line** \((x_1, y_1, x_2, y_2)\)

```plaintext
begin
    \textbf{int} \ x, y, dx, dy, d, \Delta_e, \Delta_{ne};
    x \leftarrow x_1; \quad y \leftarrow y_1;
    \text{dx} \leftarrow x_2 - x_1; \quad \text{dy} \leftarrow y_2 - y_1;
    d \leftarrow 2 * \text{dy} - \text{dx};
    \Delta_e \leftarrow 2 * \text{dy}; \quad \Delta_{ne} \leftarrow 2 * (\text{dy} - \text{dx});

    \textbf{PlotPixel} (x, y);

    \textbf{while} (x < x_2) \textbf{do}
    \begin{align*}
        \textbf{if} & \ (d < 0) \ \textbf{then} \ \begin{align*}
            \textbf{begin} & \ d \leftarrow d + \Delta_e; \\
                      & \ x \leftarrow x + 1;
        \textbf{end}; & \\
        \textbf{else} & \begin{align*}
            \textbf{begin} & \ d \leftarrow d + \Delta_{ne}; \\
                      & \ x \leftarrow x + 1; \\
                      & \ y \leftarrow y + 1;
        \textbf{end}; & \\
    \textbf{PlotPixel} & (x, y);
    \end{align*}
    \end{align*}
\end{align*}
end;
end;
```

- **bresenham**
- **fineline**
Midpoint Examples

Question: Is there a problem with the MidPoint algorithm (hint: horizontal vs. diagonal lines)?

Comment: Algorithm extends to higher order curves – e.g. circles
Error Function Intuition

- Error function $d$ can be viewed as *explicit surface*:
  
  $$d(x,y)=2(xdy-ydx+c)$$

- Implicit equation of origin-centered circle $C$ is
  
  $$x^2+y^2-R^2 = 0$$

- Error functional
  
  $$d(x,y) = x^2+y^2-R^2$$

don $C$

- Can also be seen as an *explicit surface*
Circle Midpoint

- For first quadrant, start from \((x_1, y_1) = (0, R)\)
- Move *east* or *south-east*
- \(d(x, y)\) will be a threshold criteria at the midpoint

- Because of symmetry - suffices to deal with octant of first quadrant for which \(y > x\)
Circle Midpoint (cont’d)

- At the beginning: \( d_{\text{start}} = d(x_1 + 1, y_1 - \frac{1}{2}) = d(1, R - \frac{1}{2}) = \frac{5}{4} - R \)

- Moving east:
  \[
  \Delta_e = d(x + 2, y - \frac{1}{2}) - d(x + 1, y - \frac{1}{2})
  = (x + 2)^2 + (y - \frac{1}{2})^2 - R^2 - (x + 1)^2 - (y - \frac{1}{2})^2 + R^2
  = 2x + 3
  \]

- Moving south-east:
  \[
  \Delta_{se} = d(x + 2, y - \frac{3}{2}) - d(x + 1, y - \frac{1}{2})
  = (x + 2)^2 + (y - \frac{3}{2})^2 - R^2 - (x + 1)^2 - (y - \frac{1}{2})^2 + R^2
  = 2(x - y) + 5
  \]

- \( \Delta_e \) and \( \Delta_{se} \) not constant anymore

- Since \( d \) is incremented by integer values, one can use \( d_{\text{start}} = 1 - R \), yielding an **integer algorithm**

- This last change has no effect on threshold criteria

\[
d(x, y) = x^2 + y^2 - R^2
\]
Midpoint Circle Algorithm

**CircleOctant2(R)**

begin
  int x, y, d;
  x ← 0;
  y ← R;
  d ← 1 − R;
  PlotPixel (x, y);
  while (y > x) do
    if (d < 0) then
      begin
        d ← d + 2x + 3;
        x ← x + 1;
      end;
    else begin
      d ← d + 2(x - y) + 5;
      x ← x + 1;
      y ← y - 1;
    end;
  end;

Generate other 7 octants by mirroring (*)

**Question:** What will happen if we change “while (y > x)” to “while (y > 0)”?
Circle Midpoint Examples

Full Circles

One Quadrant