Scan Conversion:
Drawing Lines on Raster Display
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 $\leq 1$

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

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 (x₁, y₁, x₂, y₂)
begin
    int x, y;
    x ← (x₁ + x₂) / 2;
    y ← (y₁ + y₂) / 2;
    if ((x = x₁ and y = y₁) or
        (x = x₂ and y = y₂))
        return;
    else begin
        PlotPixel (x, y);
        Line (x₁, y₁, x, y);
        Line (x₁, y₁, x, y);
        Line (x, y, x₂, y₂);
        Line (x, y, x₂, y₂);
    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:
  \[ x_2 > x_1, \quad y_2 > y_1 \quad \text{and} \quad \frac{dy}{dx} = \frac{y_2 - y_1}{x_2 - x_1} < 1 \]

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

Definitions
\[
\tau = \{(x, y) \mid ax + by + c = xdy - ydx + c = 0\} \\
d(x, y) = 2(xdy - ydx + c)
\]
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 \( P \in \tau \)
  \[ \Rightarrow d \text{ may serve as error function to be minimized} \]

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

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

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

```plaintext
Line (x_1, y_1, x_2, y_2)
begin
    int x, y, dx, dy, d, \Delta_e, \Delta_{ne};
x \leftarrow x_1;
y \leftarrow y_1;
dx \leftarrow x_2 - x_1;
dy \leftarrow y_2 - y_1;
d \leftarrow 2 * dy - dx;
\Delta_e \leftarrow 2 * dy;
\Delta_{ne} \leftarrow 2 * (dy - dx);
PlotPixel (x, y);
while (x < x_2) do
    if (d < 0) then
        begin
            d \leftarrow d + \Delta_e;
x \leftarrow x + 1;
        end;
    else begin
        d \leftarrow d + \Delta_{ne};
x \leftarrow x + 1;
y \leftarrow y + 1;
    end;
    \textbf{PlotPixel} (x, y);
end;
end;
```

Scan Conversion - Center for Graphics and Geometric Computing, Technion
Midpoint Examples

- Midpoint (squares) vs. Recursive (squares) line drawing
- Midpoint line drawing

**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:
  
  $$d(x,y) = 2(x\,dy - y\,dx + c)$$

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

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

  vanishes on $C$

- 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: 
  \[ d_{start} = d(x_1+1,y_1-1/2) = d(1,R-1/2) = 5/4 - R \]
- Moving east:
  \[ \Delta_e = d(x+2,y-1/2) - d(x+1,y-1/2) \]
  \[ = (x+2)^2 + (y-1/2)^2 - R^2 - (x+1)^2 - (y-1/2)^2 + R^2 \]
  \[ = 2x + 3 \]
- Moving south-east:
  \[ \Delta_{se} = d(x+2,y-3/2) - d(x+1,y-1/2) \]
  \[ = (x+2)^2 + (y-3/2)^2 - R^2 - (x+1)^2 - (y-1/2)^2 + R^2 \]
  \[ = 2(x-y) + 5 \]
- \( \Delta_e \) and \( \Delta_{se} \) not constant anymore
- Since \( d \) is incremented by integer values, one can use \( d_{start} = 1 - R \), yielding an integer algorithm
- This last change has no effect on threshold criteria
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;
    (* PlotPixel(x, y);)
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