

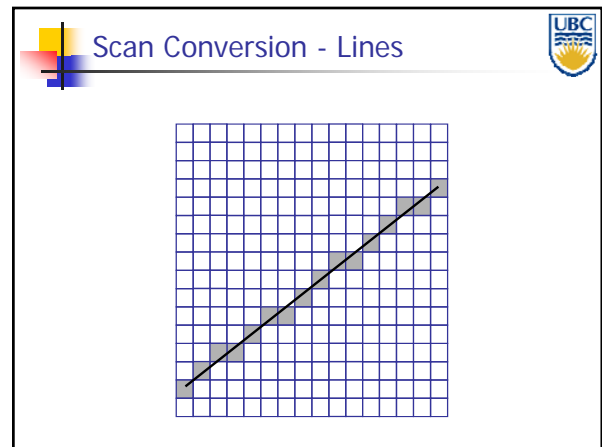
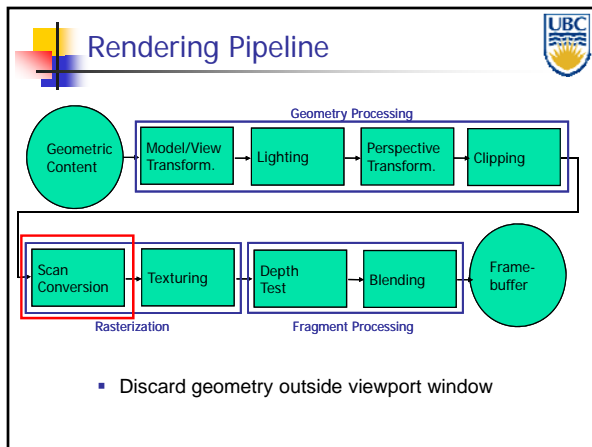
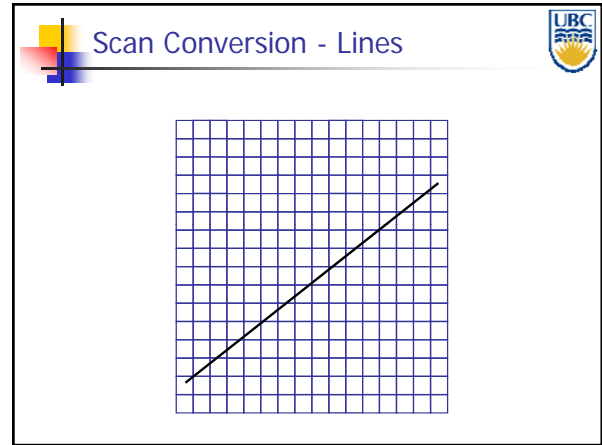
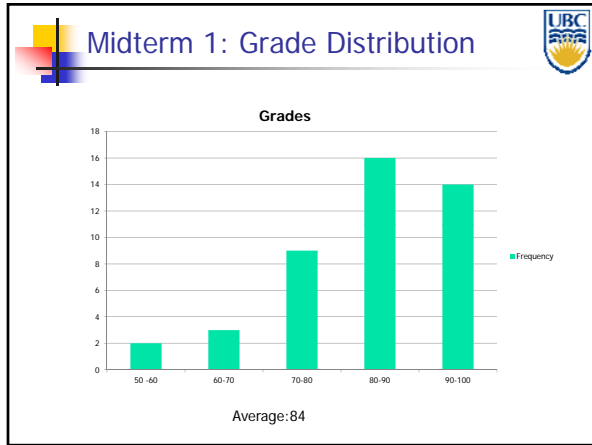
Chapter 8

Scan Conversion – Drawing on Raster Display (part 1 – Lines)

Scan Conversion - Rasterization

Convert continuous rendering primitives into discrete fragments/pixels

- Lines
 - Bresenham (Midpoint)
- Triangles
 - Implicit formulation
 - Scanline
- Interpolation



Idea: Use Explicit Line Formula

Explicit - one coordinate as function of the others

$$y = f(x)$$

$$z = f(x, y)$$

line

$$y = mx + b$$

$$y = \frac{(y_2 - y_1)}{(x_2 - x_1)}(x - x_1) + y_1$$

Typically separate into 4 (or 8) cases (why?)

Midpoint (Bresenham) Algorithm

- Key Observation 2:**
 - Can decide based on whether midpoint is above/below line
 - How?
 - Evaluate implicit line equation at $(x+1, y+1/2)$

Basic Line Drawing

Assume $x_1 < x_2$ & line slope absolute value is ≤ 1

```

Line (x1, y1, x2, y2)
begin
  float dx, dy, x, y, slope;
  dx ← x2 - x1;
  dy ← y2 - y1;
  slope ← dy/dx;
  y ← y1;
  for x from x1 to x2 do
    begin
      PlotPixel ( x, Round ( y ) );
      y ← y + slope;
    end;
  end;
  
```

Questions:
Can this algorithm use integer arithmetic?

Bresenham Algorithm

Implicit formulation = distance (up to scale)

$$\tau = \{(x, y) | ax + by + c = xdy - ydx + c = 0\}$$

$$d(x, y) = 2(xdy - ydx + c)$$

- Given point $P = (x, y)$, $d(x, y)$ is signed distance of P to τ (up to scale)
- d is zero for $P \in \tau$

Midpoint (Bresenham) Algorithm

- Key Observation 1:**
 - At each step have ONLY 2 choices
 - East/North-East

Bresenham (Midpoint) Algorithm

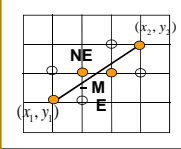
- Starting point satisfies $d(x_1, y_1) = 0$
- Each step moves right (east) or upper right (northeast)
- Sign of $d(x + 1; y + \frac{1}{2})$ indicates if to move east or northeast

Bresenham (Midpoint) Algorithm

```

Line (x1, y1, x2, y2)
begin
  int x, y, dx, dy, d;
  x ← x1; y ← y1;
  dx ← x2 - x1; dy ← y2 - y1;
  PlotPixel (x, y);
  while (x < x2) do
    d = (2x + 2)dy - (2y + 1)dx + 2c; // 2((x+1)dy - (y+.5)dx + c)
    if (d < 0) then
      begin
        x ← x + 1;
      end;
    else begin
      x ← x + 1;
      y ← y + 1;
    end;
    PlotPixel (x, y);
  end;
end;

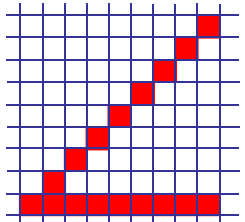
```



bresenham

Bresenham Examples

- Intensity depends on angle



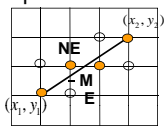
- Comment: extends to higher order curves – e.g. circles

fineline

Bresenham (Midpoint) Algorithm

- Insanely efficient version (less computations inside the loop)
 - compute d incrementally
- At (x_1, y_1)

$$d_{start} = d(x_1 + 1, y_1 + \frac{1}{2}) = 2dy - dx$$
- Increment in d (after each step)
 - If move east $\Delta_e = d(x + 2, y + \frac{1}{2}) - d(x + 1, y + \frac{1}{2}) = 2((x + 2)dy - (y + \frac{1}{2})dx + c) - 2((x + 1)dy - (y + \frac{1}{2})dx + c) = 2dy$
 - If move northeast $\Delta_{ne} = d(x_1 + 2, y_1 + \frac{3}{2}) - d(x_1 + 1, y_1 + \frac{1}{2}) = 2((x + 2)dy - (y + \frac{3}{2})dx + c) - 2((x + 1)dy - (y + \frac{1}{2})dx + c) = 2(dy - dx)$



Comparison: float/integer

Assume $x_1 < x_2$ & line slope is ≤ 1

```

Line (x1, y1, x2, y2)
begin
  int x, y, dx, dy, d, A_e, A_ne;
  x ← x1; y ← y1;
  dx ← x2 - x1; dy ← y2 - y1;
  d ← 2 * dy - dx;
  A_e ← 2 * dy; A_ne ← 2 * (dy - dx);
  PlotPixel (x, y);
  while (x < x2) do
    if (d < 0) then
      begin
        d ← d + A_e;
      end;
    else begin
      d ← d + A_ne;
      y ← y + 1;
    end;
    x ← x + 1;
    PlotPixel (x, y);
  end;
end;

```

```

Line (x1, y1, x2, y2)
begin
  float dx, dy, x, y, slope;
  dx ← x2 - x1;
  dy ← y2 - y1;
  slope ← dy / dx;
  y ← y1;
  for x from x1 to x2 do
    begin
      PlotPixel (x, Round (y));
      y ← y + slope;
    end;
  end;
end;

```

Bresenham (Midpoint) Algorithm

```

Line (x1, y1, x2, y2)
begin
  int x, y, dx, dy, d, A_e, A_ne;
  x ← x1; y ← y1;
  dx ← x2 - x1; dy ← y2 - y1;
  d ← 2 * dy - dx;
  A_e ← 2 * dy; A_ne ← 2 * (dy - dx);
  PlotPixel (x, y);
  while (x < x2) do
    if (d < 0) then
      begin
        d ← d + A_e;
        x ← x + 1;
      end;
    else begin
      d ← d + A_ne;
      x ← x + 1;
      y ← y + 1;
    end;
    PlotPixel (x, y);
  end;
end;

```

bresenham


Implicit test

- Instead of clipping line in continuous space
 - For each integer value of (x, y) test if inside window just before drawing
- Inefficient on CPU
- On a parallel (GPU) processor can be surprisingly fast


```

Line (x1, y1, x2, y2)
begin
  float dx, dy, x, y, slope;
  dx ← x2 - x1;
  dy ← y2 - y1;
  slope ← dy / dx;
  y ← y1;
  for x from x1 to x2 do
    begin
      y_int = Round (y);
      if inside (x, y_int) PlotPixel (x, y_int);
      y ← y + slope;
    end;
  end;
end;

```



Scan Conversion of Lines



Discussion

- Integer: Bresenham
 - Good for hardware implementations (integer!)
- Floating Point
 - May be faster for software (depends on system)!
 - Easier to parallelize