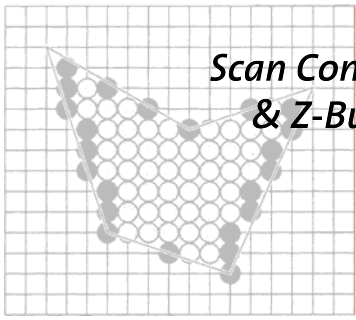
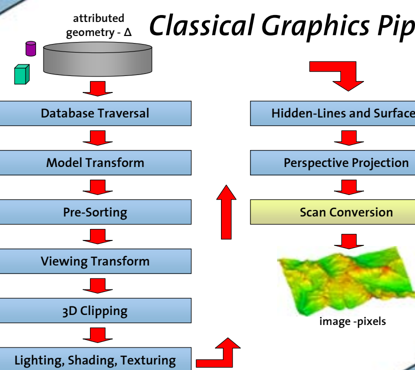
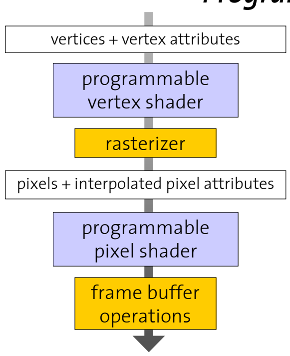


Scan Conversion & Z-Buffering

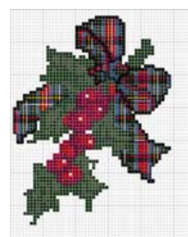
Classical Graphics Pipeline



Programmable GPU's



An Old Problem



Stitchery (2D)

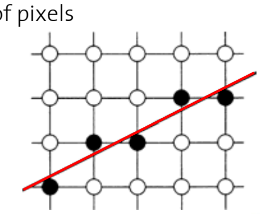
Lego (3D)

Today

- Scan conversion of
  - Lines
  - Circles
  - Polygons
- Z-Buffering

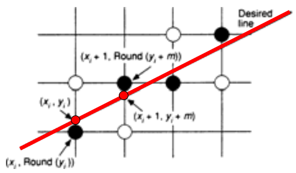
Scan Conversion of Lines

- Raster graphic displays
  ⇒ Generation of discrete pixel values
- Problem: Approximation of lines by a finite number of pixels



Digital Differential Analyzer

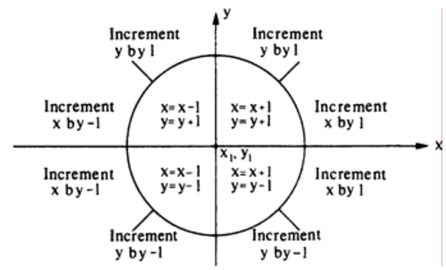
- Gradient ***m*** given by $\frac{\Delta y}{\Delta x}$



$y = mx + B$

- Take pixel with nearest distance to the line

Increments in Different Octants

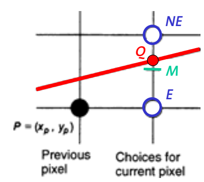


Disadvantages

- Costly rounding operations
- Unnecessary floating point arithmetic
- Error accumulation

Bresenham Line – Concept

- Goal: Fast decision which pixel has to be drawn next
- Criterion: position of the midpoint ***M*** with respect to the intersection point ***Q***



Bresenham Line – Implicit Equation

- Use implicit form of the straight line

$$F(x,y) = ax + by + c = 0$$

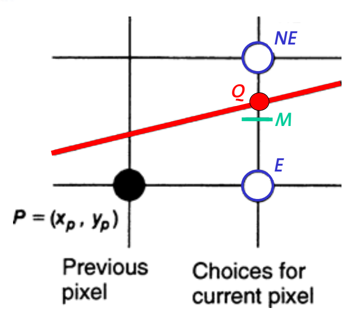
with $a = \Delta y,\ b = -\Delta x \text{ and } c = \Delta x \cdot B$

since $y = \frac{\Delta y}{\Delta x} x + B \Rightarrow \Delta y \cdot x - \Delta x \cdot y + \Delta x \cdot B = 0$

- Evaluation at the midpoint ***M***

$$d = F(M) = F(x_p + 1, y_p + \tfrac{1}{2})$$

Bresenham Line – Pixel Decision



$d > 0 \Rightarrow$ *select pixel* ***NE***

$d < 0 \Rightarrow$ *select pixel* ***E***

Bresenham Line – Update Criterion *E*

- Fast incremental update of the decision variable

$$d_{old} = F(x_p + 1, y_p + \tfrac{1}{2}) = a(x_p + 1) + b(y_p + \tfrac{1}{2}) + c$$

- Choosing pixel ***E***, the new midpoint criterion is

$$d_{new} = F(x_p + 2, y_p + \tfrac{1}{2}) = a(x_p + 2) + b(y_p + \tfrac{1}{2}) + c$$

- The difference provides the desired increment

$$d_{new} - d_{old} = a = \Delta y$$

Bresenham Line – Update Criterion *NE*

- Fast incremental update of the decision variable

$$d_{old} = F(x_p + 1, y_p + \tfrac{1}{2}) = a(x_p + 1) + b(y_p + \tfrac{1}{2}) + c$$

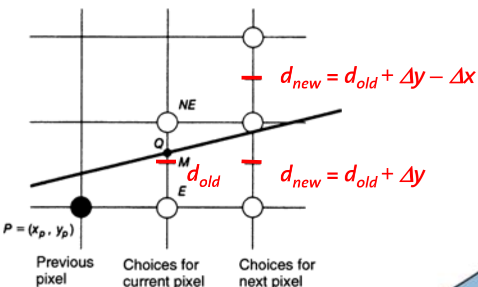
- Choosing pixel ***NE***, the new midpoint criterion is

$$d_{new} = F(x_p + 2, y_p + \tfrac{3}{2}) = a(x_p + 2) + b(y_p + \tfrac{3}{2}) + c$$

- The difference provides the desired increment

$$d_{new} - d_{old} = a + b = \Delta y - \Delta x$$

Bresenham Line – Update Criterion



Bresenham Line – Elimination of Floating Point Arithmetic

- Initialization of the decision criterion…

$$F(x_0 + 1, y_0 + \tfrac{1}{2}) = a(x_0 + 1) + b(y_0 + \tfrac{1}{2}) + c$$
$$= ax_0 + by_0 + c + a + b/2$$
$$= F(x_0, y_0) + a + b/2$$
$$d_{start} = a + b/2 = \Delta y - \Delta x/2$$

- … may result in a floating point number
- Multiplication with factor 2

$$F(x, y) = 2(ax + by + c)$$

C-Code

```
void BresenhamLine(int x0, int y0, int x1, int y1) {

        int dx, dy, incE, incNE, d, x, y;

        dx = x1 - x0; dy = y1 - y0;
        d = 2*dy - dx;
        incE = 2*dy;
        incNE = 2*(dy - dx);
        x = x0; y = y0;
        WritePixel(x, y);                          /* write start pixel */
        while (x < x1) {
                if (d <= 0)                                /* choose E */
                        d += incE;
                else {
                        d += incNE;                        /* choose NE */
                        y++;
                }
                x++;
                WritePixel(x, y);
        }
}
```


Z-Buffer – Concept

- Occlusion problem when rendering several polygons (hidden surfaces)
- Z-Buffer
  - ⇒ *Additional buffer for depth values*
  - ⇒ *Stores during scan conversion for each pixel the distance to the viewer*
  - ⇒ *Storage: Additional 16 to 32 bits per pixel*

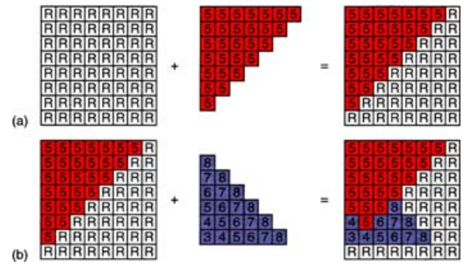
Z-Buffer – Algorithm

1.) Initialize all z-values to ∞

2.) Scan conversion of all polygons:
if z-value of a polygon pixel is smaller than the current z-buffer value
⇒ replace z-buffer value by z-value of polygon

- Complexity of algorithm:
  O(N) with N = number of polygons
- Most important hidden surface algorithm
- Mostly implemented in hardware

Z-Buffer – Example

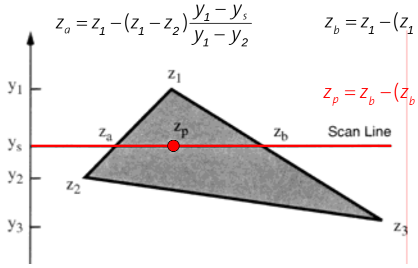


Z-Buffer – Scan Conversion of Polygons

$$z_a = z_1 - (z_1 - z_2)\frac{y_1 - y_s}{y_1 - y_2}$$

$$z_b = z_1 - (z_1 - z_3)\frac{y_1 - y_s}{y_1 - y_3}$$

$$z_p = z_b - (z_b - z_a)\frac{x_b - x_p}{y_b - y_a}$$



Z-Buffer – Remarks

- Clipping at front and back-plane possible with Z-Buffer as well
- The limited depth resolution may produce aliasing effects