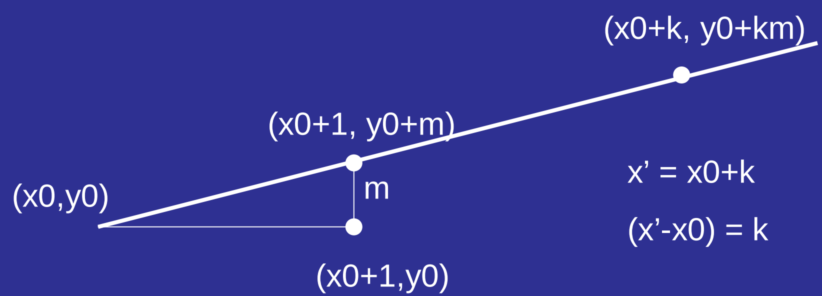
Discretization: Geometric Primitives

- Line Segment
- Triangle – *These are key primitives*
- General polygon.

Line Segments

- I want to try to discuss this as a simple example of linear interpolation (more later).
- y = mx + b
- Given (x0,y0) to (x1,y1)
  - m = (y1-y0)/(x1-x0)
  - b = y0 – mx0
- Set of points: (x’, y0 + m(x’-x0))



So we can think of a line as what we get when y is a function of x, and we linearly interpolate y between a starting value, y0, at x0, and an ending value of y1, and x1.

Another way to think of this is that we compute a y’ to go with an x’ by taking a weighted average of x0 and x1 to get x’, and then taking the same weighted average of y0 and y1 to get y’.

$x' = ax1 + (1-a)x0$. $a = (x'-x0)/(x1-x0)$

Then find y’ by taking:

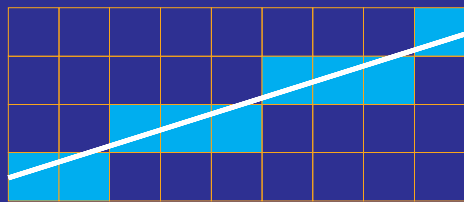
$y' = ay1 + (1-a)y0$.

Note: $y' = (y1-y0)(x'-x0)/(x1-x0) + y0$

$= m\,(x'-x0) + y0$

This is what we got before. This way of looking at it, though, can be generalized to interpolating between three points in the plane.

Line with slope 0<= m <= 1



For each x value, find y and round off.

y(x0) = y0.

y(x0+1) = y0 + m

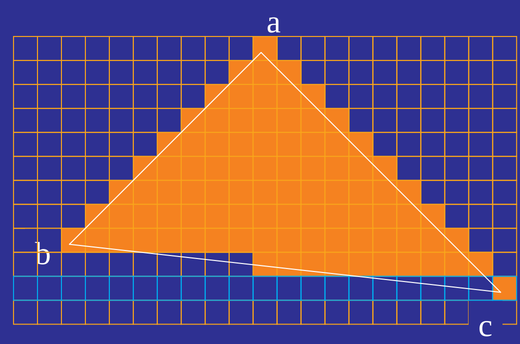
y(x0+k) = y(x0+k-1) + m

Fill in (xi, round(y(xi)))

Other Slopes

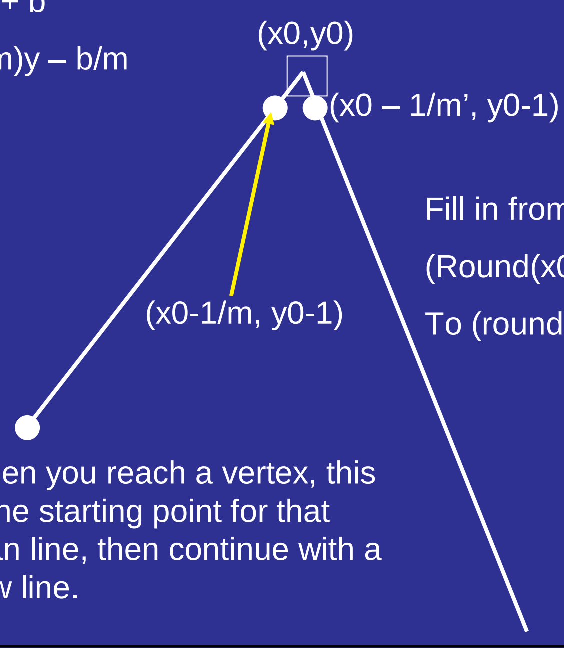
- For 1 <= m just reverse role of x and y.
  - y = mx + b => x = (1/m)y – b/m
- For -1 <= m <= 0 we can do the same thing as 0 <= m <= 1
- m <= -1 same as m >= 1, except we reduce y.
- Other cases are similar.

Triangles



$y = mx + b$

$x = (1/m)y – b/m$



Fill in from

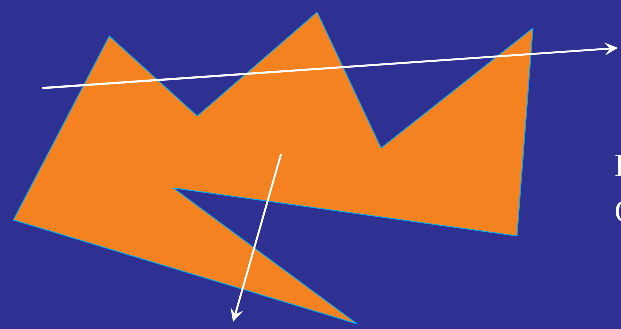
(Round(x0-1/m), y0-1)

To (round(x0-1/m’), y0-1)

When you reach a vertex, this is the starting point for that scan line, then continue with a new line.

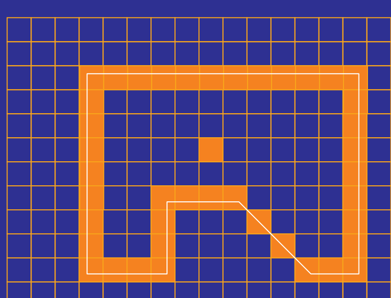
General Polygon

- Break up into triangles
- Test each pixel – crossing number test



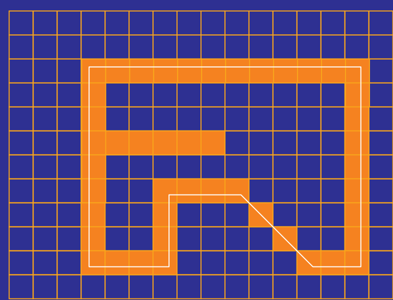
Even: Outside
Odd: Inside

Flood Fill / Seed Fill



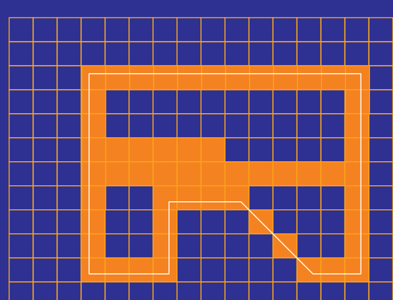
```
flood_fill (x, y)
{ if (read_pixel (x, y) != ORANGE)
   { write_pixel (x, y) = ORANGE;
      flood_fill (x - 1, y);
      flood_fill (x +1, y);
      flood_fill (x, y - 1);
      flood_fill (x, y +1);
   }
}
```

Flood Fill / Seed Fill



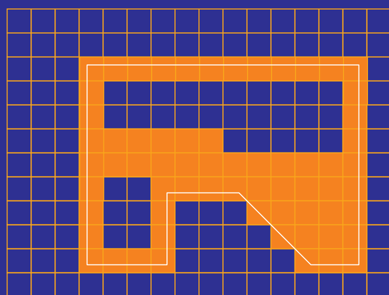
```
flood_fill (x, y)
{ if (read_pixel (x, y) != ORANGE)
   { write_pixel (x, y) = ORANGE;
      flood_fill (x - 1, y);
      flood_fill (x+1, y);
      flood_fill (x, y - 1);
      flood_fill (x, y +1);
   }
}
```

Flood Fill / Seed Fill



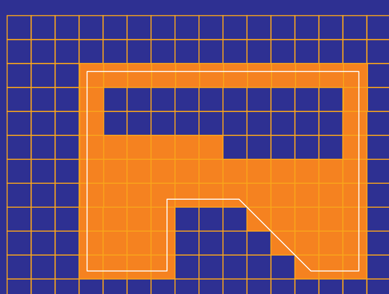
```
flood_fill (x, y)
{ if (read_pixel (x, y) != ORANGE)
   { write_pixel (x, y) = ORANGE;
      flood_fill (x - 1, y);
      flood_fill (x+1, y);
      flood_fill (x, y - 1);
      flood_fill (x, y +1);
   }
}
```

Flood Fill / Seed Fill



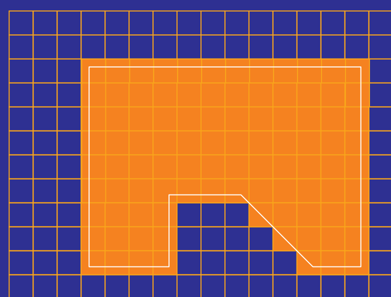
```
flood_fill (x, y)
{ if (read_pixel (x, y) != ORANGE)
   { write_pixel (x, y) = ORANGE;
      flood_fill (x - 1, y);
      flood_fill (x+1, y);
      flood_fill (x, y - 1);
      flood_fill (x, y +1);
   }
}
```

Flood Fill / Seed Fill




```
flood_fill (x, y)
{ if (read_pixel (x, y) != ORANGE)
   { write_pixel (x, y) = ORANGE;
      flood_fill (x - 1, y);
      flood_fill (x+1, y);
      flood_fill (x, y - 1);
      flood_fill (x, y +1);
   }
}
```

Flood Fill / Seed Fill



```
flood_fill (x, y)
{ if (read_pixel (x, y) != ORANGE)
   { write_pixel (x, y) = ORANGE;
      flood_fill (x - 1, y);
      flood_fill (x+1, y);
      flood_fill (x, y - 1);
      flood_fill (x, y +1);
   }
}
```

Z-Buffer Algorithm

- Image precision, object order
- Scan-convert each object
- Maintain the depth (in Z-buffer) and color (in color buffer) of the closest object at each pixel
- Display the final color buffer
- Simple; easy to implement in hardware

Z-Buffer Algorithm

```
for( each pixel(i, j) )          // clear Z-buffer and frame buffer
{
    z_buffer[i][j] = far_plane_z;
    color_buffer[i][j] = background_color;
}

for( each face A)
    for( each pixel(i, j) in the projection of A)
    {
        Compute depth z and color c of A at (i,j);
        if( z > z_buffer[i][j] )
        {
            z_buffer[i][j] = z;
            color_buffer[i][j] = c;
        }
    }
```

Efficient Z-Buffer

- Just like line discretization in one more dim.
- Polygon satisfies plane equation

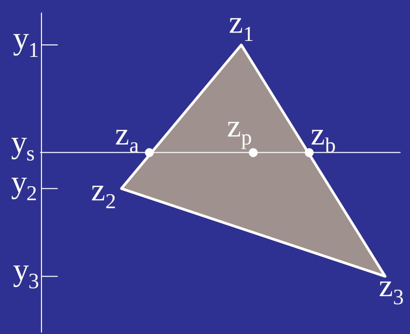
$$Ax + By + Cz + D = 0$$

- Z can be solved as

$$z = \frac{-D - Ax - By}{C}$$

- Take advantage of coherence
  - within scan line: $\Delta z = -\frac{A}{C}\Delta x$
  - next scan line: $\Delta z = -\frac{B}{C}\Delta y$

Z Value Interpolation



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

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

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

Z-Buffer: Analysis

- Advantages
  - Simple
  - Easy hardware implementation
  - Objects can be non-polygons
- Disadvantages
  - Separate buffer for depth
  - No transparency
  - No antialiasing: one item visible per pixel