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 \((x_0,y_0)\) to \((x_1,y_1)\)
  - \(m = \frac{(y_1-y_0)}{(x_1-x_0)}\)
  - \(b = y_0 - mx_0\)
- Set of points: \((x', y_0 + m(x'-x_0))\)
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, $y_0$, at $x_0$, and an ending value of $y_1$, and $x_1$.

Another way to think of this is that we compute a $y'$ to go with an $x'$ by taking a weighted average of $x_0$ and $x_1$ to get $x'$, and then taking the same weighted average of $y_0$ and $y_1$ to get $y'$.

$$x' = a x_1 + (1-a)x_0. \quad a = \frac{(x'-x_0)}{(x_1-x_0)}$$

Then find $y'$ by taking:

$$y' = ay_1 + (1-a)y_0.$$

Note: $y' = (y_1-y_0)(x'-x_0)/(x_1-x_0) + y_0$

$$= m (x'-x_0) + y_0$$

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 $0 \leq m \leq 1$

For each $x$ value, find $y$ and round off.

$y(x_0) = y_0.$

$y(x_0+1) = y_0 + m$

$y(x_0+k) = y(x_0+k-1) + m$

Fill in $(x_i, \text{round}(y(x_i)))$

**Other Slopes**

- For $1 \leq m$ just reverse role of $x$ and $y$.
  - $y = mx + b \Rightarrow x = (1/m)y - b/m$
- For $-1 \leq m \leq 0$ we can do the same thing as $0 \leq m \leq 1$
- $m \leq -1$ same as $m \geq 1$, except we reduce $y$.
- Other cases are similar.
Triangles

\[ y = mx + b \]

\[ x = \frac{1}{m}y - \frac{b}{m} \]

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

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

```c
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

```c
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

```c
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

```c
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

```c
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

\[ z_u = z_i - (z_i - z_j) \frac{y_i - y_u}{y_i - y_j} \]

\[ z_b = z_i - (z_i - z_j) \frac{y_i - y_b}{y_i - y_j} \]

\[ z_p = z_b - (z_b - z_u) \frac{x_u - x_p}{x_u - x_b} \]

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