Rasterization

CPSC 453
Transform to 2D
Transform to 2D

\[
\begin{bmatrix}
  x_{\text{pixel}} \\
  y_{\text{pixel}} \\
  z_{\text{canonical}} \\
  1
\end{bmatrix} = (M_{\text{vp}}M_{\text{orth}}) \begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
\]
We want:

- Fast
- Efficient
- Simple
Rasterization *aka* Scanline renders

- Finding all pixels in an image occupied by a geometric primitive
Rasterization

- Finding all pixels in an image occupied by a geometric primitive
Rasterization

• Finding all pixels in an image occupied by a geometric primitive
Implicit Line equation

\[ f(x, y) = 0 \]

\[ f(x, y) = y - mx - b \]

\[ m = -b/a \]

Hard to represent line \( x=0 \)
Implicit Line equation

\[ f(x, y) = -2 \]

\[ f(x, y) = -1 \]

\[ f(x, y) = 0 \]

\[ f(x, y) = 1 \]

\[ f(x, y) = 2 \]

\[ f(x, y) = Ax + By + C \]
Implicit Line equation

\[ f(x, y) = Ax + By + C \]

\[ f(x, y) \equiv (y_0 - y_1)x + (x_1 - x_0)y + x_0 y_1 - x_1 y_0 = 0 \]
Midpoint algorithm \( m \in (0, 1] \)

\[
\begin{align*}
y &= y_0 \\
\text{for } x = x_0 \text{ to } x_1 \text{ do} \\
&\quad \text{draw}(x, y) \\
&\quad \text{if (some condition) then} \\
&\quad \quad y = y + 1
\end{align*}
\]

- “thinnest line” (1 pixel)
- no gaps
Line drawing: *midpoint algorithm*

- Finding all pixels in an image occupied by a geometric primitive

```
y = y_0
for x = x_0 to x_1 do
    draw(x, y)
    if f(x + 1, y + 0.5) < 0 then
        y = y + 1
```
Finding all pixels in an image occupied by a geometric primitive

Line drawing: *midpoint algorithm*

```plaintext
y = y_0
for x = x_0 to x_1 do
  draw(x, y)
  if f(x + 1, y + 0.5) < 0 then
    y = y + 1
```
Finding all pixels in an image occupied by a geometric primitive

Line drawing: *midpoint algorithm*

- Finding all pixels in an image occupied by a geometric primitive

```plaintext
y = y_0

for x = x_0 to x_1 do
    draw(x, y)
    if f(x + 1, y + 0.5) < 0 then
        y = y + 1
```
Line drawing: *midpoint algorithm*

- Finding all pixels in an image occupied by a geometric primitive

\[
y = y_0 \\
\text{for } x = x_0 \text{ to } x_1 \text{ do} \\
\text{draw}(x, y) \\
\text{if } f(x + 1, y + 0.5) < 0 \text{ then} \\
\quad y = y + 1
\]
Finding all pixels in an image occupied by a geometric primitive

Line drawing: *midpoint algorithm*

\[
y = y_0 \\
\text{for } x = x_0 \text{ to } x_1 \text{ do} \\
\quad \text{draw}(x, y) \\
\quad \text{if } f(x + 1, y + 0.5) < 0 \text{ then} \\
\quad \quad y = y + 1
\]
Finding all pixels in an image occupied by a geometric primitive

**Line drawing: midpoint algorithm**

- for \( x = x_0 \) to \( x_1 \) do
  - \( \text{draw}(x, y) \)
  - if \( f(x + 1, y + 0.5) < 0 \) then
    - \( y = y + 1 \)
Finding all pixels in an image occupied by a geometric primitive

Line drawing: *midpoint algorithm*

```
y = y_0
for x = x_0 to x_1 do
    draw(x, y)
    if f(x + 1, y + 0.5) < 0 then
        y = y + 1
```
Line drawing: *midpoint algorithm*

- Finding all pixels in an image occupied by a geometric primitive

```
y = y_0
for x = x_0 to x_1 do
  draw(x, y)
  if f(x + 1, y + 0.5) < 0 then
    y = y + 1
```
Line drawing: *midpoint algorithm* (incremental)

\[
f(x, y) \equiv (y_0 - y_1)x + (x_1 - x_0)y + x_0y_1 - x_1y_0 = 0
\]

\[
f(x + 1, y) = f(x, y) + (y_0 - y_1)
\]

\[
f(x + 1, y + 1) = f(x, y) + (y_0 - y_1) + (x_1 - x_0).
\]
Line drawing: *midpoint algorithm (incremental)*

\[ f(x, y) \equiv (y_0 - y_1)x + (x_1 - x_0)y + x_0y_1 - x_1y_0 = 0 \]

\[
\begin{align*}
f(x + 1, y) &= f(x, y) + (y_0 - y_1) \\
f(x + 1, y + 1) &= f(x, y) + (y_0 - y_1) + (x_1 - x_0).
\end{align*}
\]

Potential *numerical* issues?

```
y = y_0 \\
d = f(x_0 + 1, y_0 + 0.5) \\
for x = x_0 to x_1 do \\
    draw(x, y) \\
    if d < 0 then \\
        y = y + 1 \\
        d = d + (x_1 - x_0) + (y_0 - y_1) \\
    else \\
        d = d + (y_0 - y_1)
```
Similar arguments for $m \notin (0,1]$
Interpolating values

- Finding all pixels in an image occupied by a geometric primitive
Interpolating values

- Finding all pixels in an image occupied by a geometric primitive

\[ 7 \text{ steps} - \Delta = \frac{1}{7} \]
Interpolating values

- Finding all pixels in an image occupied by a geometric primitive

\[
\begin{array}{cccccc}
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
\frac{1}{7} & \frac{2}{7} & \frac{3}{7} & \frac{4}{7} & \frac{5}{7} & \frac{6}{7} \\
1.0 & & & & & \\
\end{array}
\]

7 steps \[\Delta = \frac{1}{7}\]
Interpolating values

https://observablehq.com/@infowantstobeseen/drawing-lines
Triangle Rasterization: Raster each line?
Triangle Rasterization

Inside-outside test

Interpolation

\[ c = \alpha c_0 + \beta c_1 + \gamma c_2. \]
Triangle Rasterization: barycentric coordinates

\[ p = a + \beta (b - a) + \gamma (c - a). \]
Non-orthogonal coordinates
Triangle Rasterization: barycentric coordinates

\[ p = a + \beta(b - a) + \gamma(c - a). \]
Triangle Rasterization: barycentric coordinates

\[ p = a + \beta(b - a) + \gamma(c - a). \]

\[ p = (1 - \beta - \gamma)a + \beta b + \gamma c. \]

\( \beta = 2, \gamma = 0.5 \)
Triangle Rasterization: barycentric coordinates

\[ p = a + \beta(b - a) + \gamma(c - a). \]

\[ p = (1 - \beta - \gamma)a + \beta b + \gamma c. \]

\[ \alpha \equiv 1 - \beta - \gamma, \]

\[ \beta = 2, \gamma = 0.5 \]
Triangle Rasterization: barycentric coordinates

\[ p = a + \beta(b - a) + \gamma(c - a). \]

\[ p = (1 - \beta - \gamma)a + \beta b + \gamma c. \]

\[ \alpha \equiv 1 - \beta - \gamma, \]

\[ \alpha + \beta + \gamma = 1. \]
Triangle Rasterization: barycentric coordinates

\[ p = a + \beta(b - a) + \gamma(c - a). \]

\[ p = (1 - \beta - \gamma)a + \beta b + \gamma c. \]

\[ \alpha \equiv 1 - \beta - \gamma, \]

\[ \alpha + \beta + \gamma = 1. \]

\[ p(\alpha, \beta, \gamma) = \alpha a + \beta b + \gamma c, \]

\[ \beta = 2, \gamma = 0.5 \]
Triangle Rasterization: barycentric coordinates

\[ p = a + \beta (b - a) + \gamma (c - a). \]

\[ p = (1 - \beta - \gamma) a + \beta b + \gamma c. \]

\[ \alpha \equiv 1 - \beta - \gamma, \]

\[ \alpha + \beta + \gamma = 1. \]

\[ p(\alpha, \beta, \gamma) = \alpha a + \beta b + \gamma c, \]

\[ \beta = 2, \gamma = 0.5 \]
barycentric coordinates

All points *inside* the triangle have:

\[
0 < \alpha < 1, \\
0 < \beta < 1, \\
0 < \gamma < 1. 
\]

Barycentric Coords. for $\triangle abc$: $\alpha = 0.44, \beta = 0.21, \gamma = 0.35$

observablehq.com/@infowantstobeseen/barycentric-coordinates
barycentric coordinates

All points inside the triangle have:

\[ 0 < \alpha < 1, \]
\[ 0 < \beta < 1, \]
\[ 0 < \gamma < 1. \]

Barycentric Coords. for \( \Delta abc \): \( \alpha = 0.65, \beta = -0.14, \gamma = 0.49 \)
Calculate barycentric coordinates

\[
\begin{bmatrix}
  x_b - x_a & x_c - x_a \\
  y_b - y_a & y_c - y_a \\
\end{bmatrix}
\begin{bmatrix}
  \beta \\
  \gamma \\
\end{bmatrix}
=
\begin{bmatrix}
  x_p - x_a \\
  y_p - y_a \\
\end{bmatrix}
\]
Calculate barycentric coordinates

\[
\begin{bmatrix}
  x_b - x_a & x_c - x_a \\
  y_b - y_a & y_c - y_a
\end{bmatrix}
\begin{bmatrix}
  \beta \\n  \phi - x_a
\end{bmatrix}
\]

~Boring~

Use geometric reasoning...
Triangle Rasterization

\[ x_{\text{min}} = \text{floor} \, (x_i) \]
\[ x_{\text{max}} = \text{ceiling} \, (x_i) \]
\[ y_{\text{min}} = \text{floor} \, (y_i) \]
\[ y_{\text{max}} = \text{ceiling} \, (y_i) \]

\textbf{for} \ y = y_{\text{min}} \ \textbf{to} \ y_{\text{max}} \ \textbf{do}

\textbf{for} \ x = x_{\text{min}} \ \textbf{to} \ x_{\text{max}} \ \textbf{do}

\[ \alpha = \frac{f_{12}(x, y)}{f_{12}(x_0, y_0)} \]
\[ \beta = \frac{f_{20}(x, y)}{f_{20}(x_1, y_1)} \]
\[ \gamma = \frac{f_{01}(x, y)}{f_{01}(x_2, y_2)} \]

\textbf{if} \ (\alpha > 0 \ \text{and} \ \beta > 0 \ \text{and} \ \gamma > 0) \ \textbf{then}

\[ \mathbf{c} = \alpha \mathbf{c}_0 + \beta \mathbf{c}_1 + \gamma \mathbf{c}_2 \]

\text{drawpixel} \ (x, y) \ \text{with color} \ \mathbf{c}
Option 2) Barycentric coordinates via areas

\[ \alpha = \frac{A_a}{A} \]
\[ \beta = \frac{A_b}{A} \]
\[ \gamma = \frac{A_c}{A} \]
Quad Rasterization?

- **Bilinear interpolation**
Quad Rasterization?

- **Bilinear interpolation... but is not unique (e.g. mean value)**
- **Hardware is specifically optimized for triangles**
- **Graphics drivers typically split input geometry into triangles**
Shared Edges

https://observablehq.com/@infowantstobeseen/drawing-triangles
Shared Edges
Clipping
Most pernicious: near plane clipping
Most pernicious: near plane clipping
Choose the space to clip within
Choose the space to clip within

for each of six planes do
    if (triangle entirely outside of plane) then
        break (triangle is not visible)
    else if triangle spans plane then
        clip triangle
    if (quadrilateral is left) then
        break into two triangles

Use geometric reasoning...
Minimal 3D Pipeline

Sort 3D rendering by object depth

Wikipedia.org
Occlusion cycle: *painter’s algorithm breaks down*...
Sort 3D rendering by depth
### z-Buffer

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>
z-Buffer

Triangle A

Triangle A depth values

1

1

1

1

1

1

1

1

∞

∞

∞

∞

3

∞

∞

3

3

∞

5

7

Triangle A
z-Buffer

Triangle B

Triangle B depth values
z-Buffer

Triangle A & Triangle B

Combined z-buffer
A simple three-dimensional scene

Z-buffer representation
z-Buffer: “z-fighting”

1

1 3 3 1
1 3 3 1
1 3 3 1
Culling primitives

- View volume culling
- Backface culling
- Occlusion culling
Culling primitives
Culling primitives
Culling primitives