Rasterization

UCSD CSE 167
Tzu-Mao Li
Goal: given 3D shapes, project them to an image
Goal: given 3D shapes, project them to an image
We will focus on triangles

we will talk about other shapes next time
Why triangles?

- after perspective projection, a triangle is still a triangle
- so they are easier to render
Why triangles?

- after perspective projection, a triangle is still a triangle
- so they are easier to render
- triangles can represent very complex shapes!
We use “triangle meshes” to store triangles usually more efficient than storing individual triangles (why?)

vertices:
0: (0, 1, 0)
1: (1, 0, 0)
2: (0, 0, 1)
3: (-1, 0, 0)

faces:
0: (0, 1, 2)
1: (0, 1, 3)
2: (0, 2, 3)
3: (1, 2, 3)
Let’s start with a single triangle

plan: project the 3 vertices of the triangle to the screen, render the 2D triangle using Homework 1
Projecting one vertex to the screen

we covered it last time!

we will generalize to arbitrary camera poses this Friday
Rendering a single triangle

1. Take the three vertices, divide their coordinates by \(-z\)
2. Render the 2D triangle using Homework 1
Rendering a single triangle

1. take the three vertices, divide their coordinates by \(-z\)
2. render the 2D triangle using Homework 1

are we done?
What if one or more vertices are behind the camera?

the perspective projection would be wrong in this case!
We need to “clip” the triangle near clipping plane (e.g., z = -0.00001)

reading: Sutherland-Hodgman algorithm
https://en.wikipedia.org/wiki/Sutherland%E2%80%93Hodgman_algorithm
We need to “clip” the triangle

near clipping plane (e.g., $z = -0.00001$)  near clipping plane

reading: Sutherland-Hodgman algorithm
https://en.wikipedia.org/wiki/Sutherland%E2%80%93Hodgman_algorithm
Rendering a single triangle

1. clip the triangle against near clipping plane
2. take the three (or four) vertices, divide their coordinates by -z
3. render the 2D triangle(s) using Homework 1
What about multiple triangles?

```python
2d_triangles = []
for each triangle:
    clip the triangle
    take the 3-4 vertices and divide them by -z
    2d_triangles.push(triangles)
render 2d_triangles using Homework 1
```

are we done?
What about multiple triangles?

2d_triangles = []
for each triangle:
    clip the triangle
    take the 3-4 vertices and divide them by -z
2d_triangles.push(triangles)
render 2d_triangles using Homework 1

triangles can block each other!
Attempt 1: painter’s algorithm

2d_triangles = []
for each triangle:
    clip the triangle
    take the 3-4 vertices and divide them by -z
2d_triangles.push(triangles)
sort 2d_triangles by their mean z coordinates
render 2d_triangles using Homework 1

will this work?
Painter’s algorithm can often fail

cyclic order

interpenetration
Instead, we need to figure out the depth correspond to the pixel sample.
We need barycentric coordinates

each point $p$ inside the triangle can be written as

$$p = b_0 p_0 + b_1 p_1 + b_2 p_2$$

$b_0, b_1, b_2 \geq 0$

$b_0 + b_1 + b_2 = 1$

if we have the barycentric coordinates, we get the depth value
We need barycentric coordinates

each point \( \mathbf{p} \) inside the triangle can be written as

\[
\mathbf{p} = b_0 \mathbf{p}_0 + b_1 \mathbf{p}_1 + b_2 \mathbf{p}_2
\]

\( b_0, b_1, b_2 \geq 0 \)

\( b_0 + b_1 + b_2 = 1 \)

\[
b_0 = \frac{\text{area}(\mathbf{p}, \mathbf{p}_1, \mathbf{p}_2)}{\text{area}(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2)}
\]
We need barycentric coordinates

each point \( p \) inside the triangle can be written as

\[
p = b_0 p_0 + b_1 p_1 + b_2 p_2
\]

\[
b_0, b_1, b_2 \geq 0
\]

\[
b_0 + b_1 + b_2 = 1
\]

\[
b_0 = \frac{\text{area}(p, p_1, p_2)}{\text{area}(p_0, p_1, p_2)} \quad b_1 = \frac{\text{area}(p_0, p, p_2)}{\text{area}(p_0, p_1, p_2)}
\]
We need barycentric coordinates

each point \( \mathbf{p} \) inside the triangle can be written as

\[
\mathbf{p} = b_0 \mathbf{p}_0 + b_1 \mathbf{p}_1 + b_2 \mathbf{p}_2
\]

\[b_0, b_1, b_2 \geq 0\]

\[b_0 + b_1 + b_2 = 1\]

\[
b_0 = \frac{\text{area}(\mathbf{p}, \mathbf{p}_1, \mathbf{p}_2)}{\text{area}(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2)}
\]

\[
b_1 = \frac{\text{area}(\mathbf{p}_0, \mathbf{p}, \mathbf{p}_2)}{\text{area}(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2)}
\]

\[
b_2 = \frac{\text{area}(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p})}{\text{area}(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2)}
\]
We need barycentric coordinates

proof for 2D: given $p, p_0, p_1, p_2$, we can solve for $b_0, b_1, b_2$

$$p = b_0 p_0 + b_1 p_1 + b_2 p_2$$
We need barycentric coordinates

proof for 2D: given $p, p_0, p_1, p_2$, we can solve for $b_0, b_1, b_2$

$$p = b_0 p_0 + b_1 p_1 + b_2 p_2$$

$$\begin{bmatrix} p_0.x & p_1.x & p_2.x \\ p_0.y & p_1.y & p_2.y \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} p.x \\ p.y \\ 1 \end{bmatrix}$$

(pretend they are at z=1)
We need barycentric coordinates

proof for 2D: given \( \mathbf{p}, \mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2 \), we can solve for \( b_0, b_1, b_2 \)

\[
\begin{pmatrix}
\mathbf{p}_0 \cdot x & \mathbf{p}_1 \cdot x & \mathbf{p}_2 \cdot x \\
\mathbf{p}_0 \cdot y & \mathbf{p}_1 \cdot y & \mathbf{p}_2 \cdot y \\
1 & 1 & 1
\end{pmatrix}
\begin{pmatrix}
b_0 \\
b_1 \\
b_2
\end{pmatrix} =
\begin{pmatrix}
\mathbf{p} \cdot x \\
\mathbf{p} \cdot y \\
1
\end{pmatrix}
\]

Cramer’s rule

\[
b_0 = \frac{\det \begin{pmatrix}
\mathbf{p} \cdot x & \mathbf{p}_1 \cdot x & \mathbf{p}_2 \cdot x \\
\mathbf{p} \cdot y & \mathbf{p}_1 \cdot y & \mathbf{p}_2 \cdot y \\
1 & 1 & 1
\end{pmatrix}}{\det \begin{pmatrix}
\mathbf{p}_0 \cdot x & \mathbf{p}_1 \cdot x & \mathbf{p}_2 \cdot x \\
\mathbf{p}_0 \cdot y & \mathbf{p}_1 \cdot y & \mathbf{p}_2 \cdot y \\
1 & 1 & 1
\end{pmatrix}}
\]

\[
b_1 = \frac{\det \begin{pmatrix}
\mathbf{p}_0 \cdot x & \mathbf{p} \cdot x & \mathbf{p}_2 \cdot x \\
\mathbf{p}_0 \cdot y & \mathbf{p} \cdot y & \mathbf{p}_2 \cdot y \\
1 & 1 & 1
\end{pmatrix}}{\det \begin{pmatrix}
\mathbf{p}_0 \cdot x & \mathbf{p}_1 \cdot x & \mathbf{p}_2 \cdot x \\
\mathbf{p}_0 \cdot y & \mathbf{p}_1 \cdot y & \mathbf{p}_2 \cdot y \\
1 & 1 & 1
\end{pmatrix}}
\]

\[
b_2 = \frac{\det \begin{pmatrix}
\mathbf{p}_0 \cdot x & \mathbf{p}_1 \cdot x & \mathbf{p} \cdot x \\
\mathbf{p}_0 \cdot y & \mathbf{p}_1 \cdot y & \mathbf{p} \cdot y \\
1 & 1 & 1
\end{pmatrix}}{\det \begin{pmatrix}
\mathbf{p}_0 \cdot x & \mathbf{p}_1 \cdot x & \mathbf{p}_2 \cdot x \\
\mathbf{p}_0 \cdot y & \mathbf{p}_1 \cdot y & \mathbf{p}_2 \cdot y \\
1 & 1 & 1
\end{pmatrix}}
\]

check out the 3Blue1Brown video for Cramer’s rule
https://www.youtube.com/watch?v=jBsC34PxzoM
We need barycentric coordinates

proof for 2D: given $p$, $p_0$, $p_1$, $p_2$, we can solve for $b_0$, $b_1$, $b_2$

$$\begin{vmatrix}
    p_0 \cdot x & p_1 \cdot x & p_2 \cdot x \\
p_0 \cdot y & p_1 \cdot y & p_2 \cdot y \\
1 & 1 & 1
\end{vmatrix} = \begin{vmatrix}
    p_1 \cdot x & p_2 \cdot x \\
p_1 \cdot y & p_2 \cdot y \\
1 & 1
\end{vmatrix} - \begin{vmatrix}
    p_0 \cdot x & p_2 \cdot x \\
p_0 \cdot y & p_2 \cdot y \\
1 & 1
\end{vmatrix} + \begin{vmatrix}
    p_0 \cdot x & p_1 \cdot x \\
p_0 \cdot y & p_1 \cdot y \\
1 & 1
\end{vmatrix}$$

2D determinant = area of the parallelogram

check out the 3Blue1Brown video for determinant
https://www.youtube.com/watch?v=Ip3X9LOh2dk
Given a pixel sample and a screen triangle, we can figure out its barycentric coordinates:

\[
\begin{align*}
    b_0' &= \frac{\text{area}(p', p_1', p_2')}{\text{area}(p_0', p_1', p_2')} \\
    b_1' &= \frac{\text{area}(p_0', p', p_2')}{\text{area}(p_0', p_1', p_2')} \\
    b_2' &= \frac{\text{area}(p_0', p_1', p')}{\text{area}(p_0', p_1', p_2')}
\end{align*}
\]
Our goal is to figure out the barycentric coordinates of the 3D triangle

\[ b_0 = \frac{\text{area}(p, p_1, p_2)}{\text{area}(p_0, p_1, p_2)} \]
\[ b_1 = \frac{\text{area}(p_0, p, p_2)}{\text{area}(p_0, p_1, p_2)} \]
\[ b_2 = \frac{\text{area}(p_0, p_1, p)}{\text{area}(p_0, p_1, p_2)} \]

are they the same?

\[ b'_0 = \frac{\text{area}(p', p'_1, p'_2)}{\text{area}(p'_0, p'_1, p'_2)} \]
\[ b'_1 = \frac{\text{area}(p'_0, p'_1, p'_2)}{\text{area}(p'_0, p'_1, p'_2)} \]
\[ b'_2 = \frac{\text{area}(p'_0, p'_1, p'_2)}{\text{area}(p'_0, p'_1, p'_2)} \]
We solve for the 3D barycentric coordinates using the projection relation

\[ p' = -\frac{p}{p \cdot z} \]

\[ p_0' = -\frac{p_0}{p_0 \cdot z} \quad p_1' = -\frac{p_1}{p_1 \cdot z} \quad p_2' = -\frac{p_2}{p_2 \cdot z} \]
We solve for the 3D barycentric coordinates using the projection relation

\[
p' = -\frac{b_0 p_0 + b_1 p_1 + b_2 p_2}{b_0 p_0 \cdot z + b_1 p_1 \cdot z + b_2 p_2 \cdot z}
\]

\[
p_0' = -\frac{p_0}{p_0 \cdot z} \quad p_1' = -\frac{p_1}{p_1 \cdot z} \quad p_2' = -\frac{p_2}{p_2 \cdot z}
\]
We solve for the 3D barycentric coordinates using the projection relation

\[ p' = \frac{b_0 (p_0 \cdot z) p'_0 + b_1 (p_1 \cdot z) p'_1 + b_2 (p_2 \cdot z) p'_2}{b_0 p_0 \cdot z + b_1 p_1 \cdot z + b_2 p_2 \cdot z} \]

\[ p' = b_0 p'_0 + b_1 p'_1 + b_2 p'_2 \]
We solve for the 3D barycentric coordinates using the projection relation

\[ b'_0 = \frac{b_0 \mathbf{p}_0 \cdot z}{b_0 \mathbf{p}_0 \cdot z + b_1 \mathbf{p}_1 \cdot z + b_2 \mathbf{p}_2 \cdot z} \]

\[ b'_1 = \frac{b_1 \mathbf{p}_1 \cdot z}{b_0 \mathbf{p}_0 \cdot z + b_1 \mathbf{p}_1 \cdot z + b_2 \mathbf{p}_2 \cdot z} \]

\[ b'_2 = \frac{b_2 \mathbf{p}_2 \cdot z}{b_0 \mathbf{p}_0 \cdot z + b_1 \mathbf{p}_1 \cdot z + b_2 \mathbf{p}_2 \cdot z} \]
We solve for the 3D barycentric coordinates using the projection relation

\[ b'_0 = \frac{p'_0 \cdot z}{Z}, \]

\[ b'_1 = \frac{p'_1 \cdot z}{Z}, \]

\[ b'_2 = \frac{p'_2 \cdot z}{Z}, \]

\[ Z = p'_0 \cdot z + p'_1 \cdot z + p'_2 \cdot z. \]
We solve for the 3D barycentric coordinates using the projection relation

\[ b_0 = \frac{Zb'_0}{p_0 \cdot z} \]

\[ b_1 = \frac{Zb'_1}{p_1 \cdot z} \]

\[ b_2 = \frac{Zb'_2}{p_2 \cdot z} \]

\[ Z = b_0 p_0 \cdot z + b_1 p_1 \cdot z + b_2 p_2 \cdot z \]
We solve for the 3D barycentric coordinates using the projection relation

\[
\begin{align*}
Z &= b_0 \mathbf{p}_0 \cdot \mathbf{z} + b_1 \mathbf{p}_1 \cdot \mathbf{z} + b_2 \mathbf{p}_2 \cdot \mathbf{z} \\

b_0 &= \frac{Zb'_0}{\mathbf{p}_0 \cdot \mathbf{z}} \\
b_1 &= \frac{Zb'_1}{\mathbf{p}_1 \cdot \mathbf{z}} \\
b_2 &= \frac{Zb'_2}{\mathbf{p}_2 \cdot \mathbf{z}} \\

b_0 + b_1 + b_2 &= 1
\end{align*}
\]
We solve for the 3D barycentric coordinates using the projection relation

\[\mathbf{b}_0' \mathbf{p}_0 + \mathbf{b}_1' \mathbf{p}_1 + \mathbf{b}_2' \mathbf{p}_2 = \mathbf{p} \]

\[\mathbf{b}_0 = \frac{\mathbf{b}_0'}{\mathbf{p}_0 \cdot z} + \frac{\mathbf{b}_1'}{\mathbf{p}_1 \cdot z} + \frac{\mathbf{b}_2'}{\mathbf{p}_2 \cdot z}\]

\[\mathbf{b}_1 = \frac{\mathbf{b}_0'}{\mathbf{p}_0 \cdot z} + \frac{\mathbf{b}_1'}{\mathbf{p}_1 \cdot z} + \frac{\mathbf{b}_2'}{\mathbf{p}_2 \cdot z}\]

\[\mathbf{b}_2 = \frac{\mathbf{b}_0'}{\mathbf{p}_0 \cdot z} + \frac{\mathbf{b}_1'}{\mathbf{p}_1 \cdot z} + \frac{\mathbf{b}_2'}{\mathbf{p}_2 \cdot z}\]
The 3D barycentric coordinates are the 2D ones weighted by inverse depth.

\[ b_0 = \frac{b_0'}{p_0 \cdot z} + \frac{b_1'}{p_1 \cdot z} + \frac{b_2'}{p_2 \cdot z} \]

\[ b_1 = \frac{b_1'}{p_1 \cdot z} + \frac{b_0'}{p_0 \cdot z} + \frac{b_2'}{p_2 \cdot z} \]

\[ b_2 = \frac{b_2'}{p_2 \cdot z} + \frac{b_0'}{p_0 \cdot z} + \frac{b_1'}{p_1 \cdot z} \]
We can now interpolate any values on the 3D triangle.

This is called the “perspective-corrected interpolation”.

\[ Z = \frac{1}{\frac{b_0'}{p_0 \cdot z} + \frac{b_1'}{p_1 \cdot z} + \frac{b_2'}{p_2 \cdot z}} \]

\[ v = b_0 v_0 + b_1 v_1 + b_2 v_2 \]
The rasterization algorithm

\[ z_{\text{buffer}} = \text{inf}(w, h) \]
\[ \text{img} = \text{zeros}(w, h) \]

for each clipped 3D triangle:
  get the 2D triangle by dividing \(-z\)
  compute image space bounding box
  for each pixel \((x, y)\) in the bounding box:
    if the pixel center hits the triangle
      interpolate \(Z\)
    if \((Z < z_{\text{buffer}}[x, y])\):
      \[ z_{\text{buffer}}[x, y] = Z \]
      \[ \text{img}[x, y] = \ldots \]
This is basically the rasterization algorithm used in Unreal!

**Micropoly** software rasterizer

- 128 triangle clusters => threadgroup size 128
- 1 thread per vertex
  - Transform position
  - Store in groupshared
  - If more than 128 verts loop (max 2)
- 1 thread per triangle
  - Fetch indexes
  - Fetch transformed positions
  - Calculate edge equations and depth gradient
  - Calculate screen bounding rect
  - For all pixels in rect
    - If inside all edges then write pixel

Speeding up rasterization

```python
z_buffer = zeros(w, h)
img = zeros(w, h)
for each clipped 3D triangle:
    get the 2D triangle by dividing -z
    compute image space bounding box
    for each pixel (x, y) in the bounding box:
        if the pixel center hits the triangle
            interpolate Z
            if (Z < z_buffer[x, y]):
                z_buffer[x, y] = Z
                img[x, y] = ...
```

we can often skip triangles by
1. frustum culling, and 2. occlusion culling
Frustum culling

triangles outside of the “view frustum” can be discarded
Frustum culling

triangles outside of the “view frustum” can be discarded

many rasterizers also have a “far clipping plane” to discard triangles that are too far away
Occlusion culling

don’t need to draw a triangle if it’s fully blocked by others
Occlusion culling

- for each triangle, test with the current Z buffer
- find the minimum Z among the three vertices (e.g., min Z = 6)
Occlusion culling

- for each triangle, test with the current Z buffer
- find the minimum Z among the three vertices (e.g., min Z = 6)
- next, form the screen space bounding box

Z buffer

```
<table>
<thead>
<tr>
<th>7</th>
<th>3</th>
<th>2</th>
<th>0</th>
<th>1</th>
<th>4</th>
<th>5</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>
```
Occlusion culling

- for each triangle, test with the current Z buffer
- find the minimum Z among the three vertices (e.g., \( \text{min } Z = 6 \))
- next, form the screen space bounding box
- find the maximum of the Z value within the box (e.g., \( \text{max } Z = 5 \))

\[
\begin{array}{cccccccc}
7 & 3 & 2 & 0 & 1 & 4 & 5 & 3 \\
7 & 0 & 3 & 0 & 2 & 3 & 9 & 1 \\
1 & 1 & 3 & 4 & 2 & 1 & 7 & 0 \\
0 & 5 & 4 & 3 & 4 & 2 & 9 & \\
8 & 0 & 1 & 2 & 3 & 4 & 5 & 4 \\
1 & 2 & 4 & 0 & 0 & 4 & 5 & 6 \\
2 & 3 & 4 & 5 & 4 & 3 & 4 & 3 \\
5 & 6 & 5 & 4 & 1 & 0 & 8 & 8 \\
\end{array}
\]
Occlusion culling

- for each triangle, test with the current Z buffer
- find the minimum Z among the three vertices (e.g., min Z = 6)
- next, form the screen space bounding box
- find the maximum of the Z value within the box (e.g., max Z = 5)
- skip the triangle if min Z > max Z

Z buffer

<table>
<thead>
<tr>
<th>7</th>
<th>3</th>
<th>2</th>
<th>0</th>
<th>1</th>
<th>4</th>
<th>5</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>
Occlusion culling

- for each triangle, test with the current Z buffer

- find the minimum Z among the three vertices (e.g., min Z = 6)

- next, form the screen space bounding box

  slow if done naively!

- find the maximum of the Z value within the box (e.g., max Z = 5)

- skip the triangle if min Z > max Z

<table>
<thead>
<tr>
<th>Z buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 3 2 0 1 4 5 3</td>
</tr>
<tr>
<td>7 0 3 0 2 3 9 1</td>
</tr>
<tr>
<td>1 1 3 4 2 1 7 0</td>
</tr>
<tr>
<td>0 5 5 4 3 4 2 9</td>
</tr>
<tr>
<td>8 0 1 2 3 4 5 4</td>
</tr>
<tr>
<td>1 2 4 0 0 4 5 6</td>
</tr>
<tr>
<td>2 3 4 5 4 3 4 3</td>
</tr>
<tr>
<td>5 6 5 4 1 0 8 8</td>
</tr>
</tbody>
</table>
Hierarchical Z-buffer for occlusion culling
[Greene et al. 1993]

build a “pyramid” from the Z buffer
Hierarchical Z-buffer for occlusion culling
[Greene et al. 1993]

min Z = 6

• test the top level of the pyramid
Hierarchical Z-buffer for occlusion culling
[Greene et al. 1993]

- \( \min Z = 6 \)
- test the top level of the pyramid
- if not skipped, test the second level
Hierarchical Z-buffer for occlusion culling
[Greene et al. 1993]

\[ \text{min } Z = 6 \]

- test the top level of the pyramid
- if not skipped, test the second level
- recurse into each cell

```
   7  | 3  |
----+----+
   5  | 5  |
----+----+
   8  | 8  |
```
Hierarchical Z-buffer for occlusion culling
[Greene et al. 1993]

min $Z = 6$

- test the top level of the pyramid
- if not skipped, test the second level
- recurse into each cell
- stop recursing when $\max Z < \min Z$ (or no overlap with the bounding box)
Hierarchical Z-buffer for occlusion culling [Greene et al. 1993]

min $Z = 6$

- test the top level of the pyramid
- if not skipped, test the second level
- recurse into each cell
- stop recursing when $\max Z < \min Z$ (or no overlap with the bounding box)
Hierarchical Z-buffer for occlusion culling
[Greene et al. 1993]

- \( \text{min } Z = 6 \)

- test the top level of the pyramid

- if not skipped, test the second level

- recurse into each cell

- stop recursing when \( \text{max } Z < \text{min } Z \) (or no overlap with the bounding box)
Hierarchical Z-buffer for occlusion culling

[Greene et al. 1993]

min $Z = 6$

- test the top level of the pyramid
- if not skipped, test the second level
- recurse into each cell
- stop recursing when $\max Z < \min Z$ (or no overlap with the bounding box)
In practice: usually cull many triangles together
Hierarchical Z-Buffer is still used today!

- Occlusion cull against Hierarchical Z-Buffer (HZB)
- Calculate screen rect from bounds
- Test against lowest mip where screen rect <= 4x4 pixels

Next: ray tracing vs rasterization