Visibility (hidden surface removal)

- A correct rendering requires correct visibility calculations.
- Correct visibility – when multiple opaque polygons cover the same screen space, only the front most one is visible (remove the hidden surfaces).

![Wrong visibility](image1.png)  ![Correct visibility](image2.png)
Visibility (hidden surface removal)

- Goal: determine which objects are visible to the eye
  - Determine what colors to use to paint the pixels
- Active research subject - lots of algorithms have been proposed in the past (and is still a hot topic)
Where is visibility performed in the graphics pipeline?
OpenGL - Image Space Approach

- Determine which of the n objects is visible to each pixel on the image plane

for (each pixel in the image) {
    determine the object closest to the pixel
    draw the pixel using the object’s color
}
Image Space Approach – Z-buffer

- Method used in most of graphics hardware (and thus OpenGL): **Z-buffer algorithm**

- Basic idea:
  - rasterize every input polygon
  - For every pixel in the polygon interior, calculate its corresponding Z value (by interpolation)
  - Choose the color of the polygon whose z value is the closest to the eye to paint the pixel.
Z (depth) buffer algorithm

How to choose the polygon that has the closet Z for a given pixel?

1. Initialize (clear) every pixel in the z buffer to a very large negative value (remember object in front of eye always negative Z values)
2. Then run the following loop:
Z (depth) Buffer Algorithm

For each polygon {
    for each pixel \((x,y)\) inside the polygon projection area {
        if \((Z\_polygon\_pixel(x,y) > depth\_buffer(x,y))\) {
            depth\_buffer(x,y) = Z\_polygon\_pixel(x,y);
            color\_buffer(x,y) = polygon color at \((x,y)\)
        }
    }
}
Z buffer example

Final image

Top View

eye

Z = -.5

Z = -.3
Z buffer example (2)

Step 1: Initialize the depth buffer

$$\begin{bmatrix}
-999 & -999 & -999 & -999 \\
-999 & -999 & -999 & -999 \\
-999 & -999 & -999 & -999 \\
-999 & -999 & -999 & -999 \\
-999 & -999 & -999 & -999 \\
\end{bmatrix}$$
Step 2: Draw the blue polygon (assuming the OpenGL program draws blue polygon first – the order does not affect the final result any way).

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-999</td>
<td>-999</td>
<td>-999</td>
<td>-999</td>
</tr>
<tr>
<td>-999</td>
<td>-999</td>
<td>-999</td>
<td>-999</td>
</tr>
<tr>
<td>-.5</td>
<td>-.5</td>
<td>-999</td>
<td>-999</td>
</tr>
<tr>
<td>-.5</td>
<td>-.5</td>
<td>-999</td>
<td>-999</td>
</tr>
</tbody>
</table>

Z = -.3

Z = -.5
Step 3: Draw the yellow polygon

<table>
<thead>
<tr>
<th>-999</th>
<th>-999</th>
<th>-999</th>
<th>-999</th>
</tr>
</thead>
<tbody>
<tr>
<td>-999</td>
<td>.3</td>
<td>.3</td>
<td>-999</td>
</tr>
<tr>
<td>-.5</td>
<td>.3</td>
<td>.3</td>
<td>-999</td>
</tr>
<tr>
<td>-.5</td>
<td>.5</td>
<td>-999</td>
<td>-999</td>
</tr>
</tbody>
</table>
OpenGL functions

- glEnable(GL_DEPTH_TEST)
- glDepthFunc(GLenum fun):
  - GL_NEVER; GL_ALWAYS; GL_LESS; GL_LEQUAL;
    GL_EQUAL; GL_GEQUAL; GL_GREATER; GL_NOTEQUAL;
  - Default: GL_LESS
- Need to remember to initialize the depth buffer in GLUT
  - glutInitDisplayMode(GLUT_RGB|GLUT_DOUBLE|GLUT_DEPTH);
- glDepthMask(GLboolean flag)
  - Whether to write to the depth buffer or not
  - GL_TRUE or GL_FALSE
  - Default is GL_TRUE once gl depth test is enabled
Other Approaches

- Not implemented by most of the graphics hardware
  - Back face culling (supported by OpenGL)
  - View frustum culling
  - Ray tracing
  - Painter’s algorithm
  - And many more …
Back Face Culling

- Back face culling – remove the back faces of a closed opaque object

- How to detect back faces?
View-Frustum Culling

- Remove objects that are outside the viewing frustum

Done by application
Ray Tracing

- Ray tracing is another example of an image space method.
- Ray tracing: Cast a ray from eye through each pixel to the world. Calculate ray-object intersection.
Painter’s Algorithm

- A depth sorting method
- Surfaces are sorted in the order of decreasing depth
- Surfaces are drawn in the sorted order, and overwrite the pixels in the frame buffer
- Two problems:
  - It can be nontrivial to sort the surfaces
  - It can be no solution for the sorting order