Hidden surface Removal

- Drawing polygon faces on screen consumes CPU cycles
- We cannot see every surface in scene
- To save time, draw only surfaces we see
- Surfaces we cannot see and their elimination methods:
  - Occluded surfaces: hidden surface removal (visibility)
  - Back faces: back face culling
  - Faces outside view volume: viewing frustrum culling

Definitions:
- Object space: before vertices are mapped to pixels
- Image space: after vertices have been rasterized

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

Correct visibility

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)
**Visibility (hidden surface removal)**

Where is visibility performed in the graphics pipeline?

1. **Modeling and viewing**
2. **Per vertex lighting**
3. **Projection**
4. **Viewport mapping**
5. **Interpolate vertex colors**
6. **Display**

Note: Map \((x,y)\) values to screen (draw) and use \(z\) value for depth testing.

**OpenGL – Image Space Approach**

- Determine which of the \(n\) objects is visible to each pixel on the image plane.

```plaintext```
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
- Requires lots of memory
- Basic idea:
  - Rasterize every input polygon
  - Recall that we have \(z\) at polygon vertices
  - 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.

**Image Space Approach – Z-buffer**

- Recall: after projection transformation
- In viewport transformation
  - \(x,y\) used to draw screen image
  - \(z\) component is mapped to pseudo-depth with range \([0,1]\)
- However, objects/polygons are made up of vertices
- Hence \(z\) is known at vertices
- Point in object seen through pixel may be between vertices
- Need to interpolate to find \(z\)
Z (depth) buffer algorithm

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

  - Assumption for example: eye at z = 0, farther objects have increasingly negative values
    1. Initialize (clear) every pixel in the z buffer to a very large negative value
    2. Track polygon Z's.
    3. As we rasterize polygons, check to see if polygon's Z through this pixel is less than current minimum Z through this pixel
    4. 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)
        }
    }
}
```

Note: we have depths at vertices. Interpolate for interior depths

Z buffer example

Eye

Z = -.3

Final image

Top view

Z buffer example

Step 1: Initialize the depth buffer

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td>99</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>99</td>
<td>99</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>99</td>
<td>99</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>99</td>
<td>99</td>
<td>99</td>
<td>99</td>
</tr>
</tbody>
</table>
Z buffer example

Step 2: Draw the blue polygon (assuming the OpenGL program draws blue polygon first – the order does not affect the final result any way).

Step 3: Draw the yellow polygon

Z buffer drawback: wastes resources by rendering a face and then drawing over it

Combined z-buffer and Gouraud Shading (fig 8.31)

For(int y = ybott; y <= ytop; y++) // for each scan line
{
    find xleft and xright
    find dleft and dright, and dinc
    find colorleft and colorright, and colorinc
    for(int x = xleft, c = colorleft, d = dleft; x <= xright;
    x++, c+= colorinc, d+= dinc)
    {
        if(d < d(x)[y])
        {
            put c into the pixel at (x, y)
            d[x][y] = d; // update the closest depth
        }
    }
}

OpenGL HSR Commands

- Primarily three commands to do HSR
- glutInitDisplayMode(GLUT_DEPTH | GLUT_RGB) instructs OpenGL to create depth buffer
- glEnable(GL_DEPTH_TEST) enables depth testing
- glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT) initializes the depth buffer every time we draw a new picture
**Back Face Culling**

- Back faces: faces of opaque object which are "pointing away" from viewer
- Back face culling – remove back faces (supported by OpenGL)

![Back face diagram]

- How to detect back faces?

**Back Face Culling**

- If we find backface, do not draw, save rendering resources
- There must be other forward face(s) closer to eye
- F is face of object we want to test if backface
- P is a point on F
- Form view vector, V as (eye – P)
- N is normal to face F

Backface test: F is backface if N.V < 0  why??

**Back Face Culling: Draw mesh front faces**

```cpp
void Mesh::drawFrontFaces() {
    for(int f = 0; f < numFaces; f++) {
        if(isBackFace(f, ...)) continue;
        glBegin(GL_POLYGON);
        int in = face[f].vert[v].normIndex;
        int iv = face[v].vert[v].vertIndex;
        glNormal3f(norm[in].x, norm[in].y, norm[in].z);
        glVertex3f(pt[iv].x, pt[iv].y, pt[iv].z);
        glEnd();
    }
}
```

Ref: case study 7.5, pg 406, Hill

**View-Frustum Culling**

- Remove objects that are outside the viewing frustum
- Done by 3D clipping algorithm (e.g. Liang-Barsky)
Ray Tracing

- Ray tracing is another example of image space method
- Ray tracing: Cast a ray from eye through each pixel to the world.
- Question: what does eye see in direction looking through a given pixel?

References
- Hill, section 8.5, chapter 13