Visibility
Visibility

- What is visible?
  - Which objects are visible?
  - Which pixels (fragments) to render?

- Why check for visibility?
  - Efficiency
  - Correctness?
  - Disambiguation

The Double Eagle Tanker:
4GB of data, 82 M Triangles
From: http://www.cs.unc.edu/~geom/hardware/#Vis
Simple question

• The *art gallery* problem:

  Given a planar art gallery, what is the minimum number of guards that need to be placed at the corners (but inside the gallery) so that every part of the gallery is visible to at least one guard.
Simple question

- NP – Complete!

- Upper bound: floor(N/3) for a simple polygon with N vertices.

- Determining visibility is not always easy.
Visibility

• The **Image Space** problem formulation for (each *pixel* in the rendered image)

  
  \[ \{ \]
  
  - determine the object closest to the viewer that is intercepted by the projector (ray) through the pixel;
  - draw the pixel in the appropriate color;

  \[ \} \]

• Worst case complexity: \( np \)

  \( n : \) number of objects, \( p : \) number of pixels
Visibility

• The **Object Space** problem formulation
  for (every **object** in the world)
  
  {  
  - determine those parts of the object whose view is unobstructed by other parts of itself or any other object;
  - draw those parts in the appropriate color;
  }

• **Worst case complexity:** $n^2$

  $n$ : number of objects
Visibility

• Types of visibility computation we have seen:
  – Clipping – 2D and 3D
  – View-frustum clipping/culling
  – Backface culling

http://geometricalgebra.org
Visibility

• Backface Culling

\[(p - e) \cdot \vec{n} > 0\] Cull

\[(p - e) \cdot \vec{n} \leq 0\] Do Not Cull (may be visible)

Simple Idea:

Discard surface patches that face away from the camera.
Visibility

• Backface Culling

If \( p_1, p_2, p_3, p_4 \) are the patch vertices in CCW order seen from outside then the outward facing normal is given by:
\[
\vec{n} = (p_2 - p_1) \times (p_3 - p_1)
\]

Compute the outward normals and do Backface culling in the WCS.
Visibility

• Backface Culling is not enough

Backface Culling does not remove all occluded patches (it is a conservative algorithm as are many visibility algorithms) – the example shown here is a case of self occlusion.
Visibility

- Z-Buffer and Scan Conversion
  - Initialize the z-buffer to the max Z value.
  - `glClear`, `glDepthRange`

```
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
```
Visibility

- Z-Buffer and Scan Conversion

```
<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```
Visibility

• Z-Buffer and Scan Conversion

(0.0,0.6,0.5)  
(0.1,0.1,0.2) (0.5,0.1,0.7)
Visibility

• Z-Buffer and Scan Conversion

Note that almost everywhere the result is independent of the order of drawing these polygons. Except at the pixels where the depth may be the same (this is very unlikely).
Visibility

• Z-Buffer Algorithm

Initialize

\[
\text{zbuf}[i, j] = \text{MAX}_\text{DEPTH} \\
\text{cbuf}[i, j] = \text{BACKGROUND}_\text{COLOR}
\]

for (each scan converted polygon)

{

\quad \text{Find pseudodepth, } z, \text{ of polygon at pixel (x, y) with color c}

\quad \text{if } (z < \text{zbuf}[i, j]) \{ \text{zbuf}[i, j] = z; \text{cbuf}[i, j] = c; \}

}
Visibility

• Z-Buffer Algorithm
  • Advantages
    – Simple, Accurate (modulo non-linear z-mapping).
    – Independent of order of drawing polygons.
  • Disadvantages
    – Memory (not an issue these days).
    – Wasted computation when over-writing distant points
  • Complexity
    – Time: $O(nm \cdot k)$ – nxm pixels, k polygons
    – Space: $O(nm \cdot b)$ – nxm pixels, b bytes precision per pixel
Visibility

- Z-Buffer Algorithm and Scan Conversion

  - Construct the active edge list AEL for every scanline.
  - Interpolate the pseudodepth for each active edge.

```plaintext
for each edge [(xᵢ, yᵢ, dᵢ) and (xⱼ, yⱼ, dⱼ)] with yᵢ < yⱼ {
    x = xᵢ, d = dᵢ, Δx = (xⱼ - xᵢ) / (yⱼ - yᵢ) and Δd = (dⱼ - dᵢ) / (yⱼ - yᵢ)
    for (y = yᵢ, y < yⱼ; y++) {
        insert (x, d) into the AEL of scanline y such that it is sorted on the x values
        x = x + Δx, d = d + Δd
    }
}
```
Visibility

- **Z-Buffer Algorithm and Scan Conversion**
  - Compute the active edge list.
  - Interpolate the pseudodepth for each active edge.
  - Now, for rendering a $\Delta ABC$:
    
    ```
    \[
    \begin{align*}
    \text{cbuf}[i,j] &= \text{BACKGROUND\_COLOR} \\
    \text{zbuf}[i,j] &= \text{MAX\_DEPTH}, \forall 0 \leq i < N, 0 \leq j < M \\
    y_{\min} &= \min(y_A, y_B, y_C) \quad y_{\max} = \max(y_A, y_B, y_C) \\
    \text{for } (y = y_{\min}; y \leq y_{\max}; y++) 
    \{ 
    \quad \text{get } (x_p, d_p) \text{ and } (x_q, d_q) \text{ from the AEL with } x_p < x_q \\
    \quad \Delta d = \frac{(d_q - d_p)}{(x_q - x_p)}; \\
    \quad \text{for } (x = x_p, d = d_p; x \leq x_q; x++, d = d + \Delta d) 
    \{ 
    \quad \text{if } (d < \text{zbuf}[x, y]) \{ \text{zbuf}[x, y] = d, \text{cbuf}[x, y] = c \} 
    \} 
    \}
    \end{align*}
    ```
    ```
    \[
    \begin{align*}
    d_1' &\quad x_7 \\
    d_2' &\quad x_6 \\
    d_3' &\quad x_5 \\
    d_4' &\quad x_4 \\
    d_5 &\quad x_2 \\
    d_6 &\quad x_2 \\
    \end{align*}
    \]
    \]
    \[
    \begin{align*}
    y_1 \\
    y_2 \\
    y_3 \\
    y_4 \\
    y_5 \\
    y_6 \\
    y_7 \\
    y_8 \\
    \end{align*}
    \]
    \[
    \begin{align*}
    x_1 &\quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \quad x_8 \\
    \end{align*}
    \]
    ```
    - Note: The color $c$ at a pixel is also interpolated along the scanline like the pseudodepth is...
Visibility

• Z-Buffer Algorithm and Scan Conversion

• Compute the active edge list.
• Interpolate the pseudodepth for each active edge.
Visibility

- **Painter's Algorithm**
  - Sort polygon's in increasing order of depth.
  - Draw the sorted list of polygons from back to front, i.e., from greatest depth to lesser depths.
  - What happens when a polygon has vertices at different depths?
Visibility

- Painter's Algorithm
  - Sort polygon's in increasing order of depth.
  - Draw the sorted list of polygons from back to front, i.e., from greatest depth to lesser depths.
  - What happens when a polygon has vertices at different depths?
  - Sort according to depth of farthest vertex.
Visibility

- Painter's Algorithm
  - Sort polygon's in increasing order of depth.
  - Draw the sorted list of polygons from back to front, i.e., from greatest depth to lesser depths.
  - What happens when a polygon has vertices at different depths?
  - Sort according to depth of farthest vertex.
  - Does it always work?
  - How often do we sort?
Visibility

- Painter's Algorithm
Visibility

- Painter's Algorithm
Visibility

- Binary Space Partitioning (BSP) Trees

- Observe the correct order of drawing polygons as the eye moves
Visibility

- Binary Space Partitioning (BSP) Trees

- If \( e \) and \( T_2 \) are on the same side of \( T_1 \)
  - Draw \( T_1 \) and then draw \( T_2 \)

- If \( e \) and \( T_2 \) are on different sides of \( T_1 \)
  - Draw \( T_2 \) and then draw \( T_1 \)
Visibility

• Binary Space Partitioning (BSP) Trees

• If the implicit equation of the plane containing \( T_1 \)
  is given by: \( f(r) = (r - p) \cdot n \)
• If \( f(q) \cdot f(e) > 0 \) then draw \( T_1 \) and then draw \( T_2 \)
• If \( f(q) \cdot f(e) < 0 \) then draw \( T_2 \) and then draw \( T_1 \)
Visibility

- BSP Tree is an efficient data structure for quickly determining the inside/outside relation between polygons and the camera position.

- Two Phases
  - Preprocessing: BSP Tree construction (done once for a given scene)
  - Rendering: BSP Tree traversal (done whenever the eye position changes)
Visibility

- BSP Tree construction
Visibility

• BSP Tree construction
Visibility

- BSP Tree construction
Visibility

- BSP Tree construction
Visibility

- BSP Tree construction
Visibility

- BSP Tree traversal

- If e is outside (or in front of) a face i
  - Draw everything behind i, Draw i, Draw everything in front of i

- If e is inside a (or behind) face i
  - Draw everything in front of i, Draw i, Draw everything behind i
Visibility

- BSP Tree traversal

- If e is outside (or in front of) a face i
  - Draw everything behind i, Draw i, Draw everything in front of i

- If e is inside a (or behind) face i
  - Draw everything in front of i, Draw i, Draw everything behind i

What is the traversal order now?