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

- 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

Visibility

- Backface Culling

  \[(p - e) \cdot n > 0 \quad \text{Do not Cull (may be visible)}\]

  \[(p - e) \cdot n < 0 \quad \text{Cull}\]

  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:

  \[ \mathbf{n} = (p_2 - p_1) \times (p_3 - p_1) \]

  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 (so are many visibility algorithms) – the example shown here is a case of self-occlusion.

Visibility

- Floating Horizon Algorithm

Given a surface described by \( f(x, y, z) = 0 \),

We can sample it at many 2D cutting planes, yielding a set of curves of the form \( y = f(x, z) \).

Visibility

- Z-Buffer and Scan Conversion

  - Initialize the z-buffer to the max Z value.
  - \text{glClear, glDepthRange}

Visibility

- Z-Buffer Algorithm

```c
// Initialize
zbuf[i, j] = MAX_DEPTH
sbuf[i, j] = BACKGROUND_COLOR
for (each scan converted polygon)
{
    find pseudodepth, \( z \), of polygon at pixel \((x, y)\) with color \( c \)
    if \( z < zbuf[i, j] \)
        \{ zbuf[i, j] = z; cbuf[i, j] = c; \}
}
```

Visibility

- Advantages

  - Simple, accurate (modulo non-linear z-mapping).
  - Independence of order of drawing polygons.

  - Disadvantages

    - Memory (not an issue these days).
    - Wasted computation when over-writing distant points.

    - Complexity

      - Time: \( O(nm) \) – \( n \times m \) polygons.
      - Space: \( O(nm) \) – \( n \times m \) pixels, \( b \) bytes precision per pixel.

Visibility

- Floating Horizon Algorithm

  - For each slicing plane \( i \), with \( z = z_i \),
    - Compute \( y \) for any \( x \) on the curve.
    - The point \((x, y, z_i)\) is visible if \( y > y_j \) for all \( j < i \) and \( x = x_j \).
Visibility

- Painter's Algorithm
  - Sort polygons 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 polygons 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 polygons 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 polygons 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

- 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 \( T_1 \)

Inside (Front)
Outside (Back)

- BSP Tree construction

  - If the implicit equation of the plane containing \( T_1 \) is given by:
    - 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 construction

  - 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 traversal

  - If \( e \) is outside (or in front of) \( T \):
    - Draw everything behind \( T \), Draw \( T \), Draw everything in front of \( T \)
  - If \( e \) is inside \( T \) (or behind) \( T \):
    - Draw everything in front of \( T \), Draw \( T \), Draw everything behind \( T \)
Visibility

- BSP Tree traversal

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

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

What is the traversal order now?