Visibility Culling

- Back face culling
- View-frustrum culling
- Detail culling
- Occlusion culling

View-Frustum Culling

- Remove objects that are outside the viewing frustum

1. Construct bounding volumes (BV's)
2. Create hierarchy
3. BV/V-F intersection tests

Mostly done in "Application Stage"

View-Frustum Culling (2)

- Culling against bounding volumes to save time
- Bounding volumes - AABB, OBB, Spheres, etc. - easy to compute, as tight as possible

AABB Sphere OBB

View-Frustum Culling (3)

- Often done hierarchically to save time

In-order, top-down traversal and test

View-Frustum Culling (4)

- Two popular hierarchical data structures - BSP Tree and Octree

Axis-Aligned BSP, Polygon-Aligned BSP

Intersecting?
**View-Frustum Culling (5)**
- **Octree**
  - A parent has 8 childrens
  - Subdivide the space until the number of primitives within each leaf node is less than a threshold
  - In-order, top-down traversal

**Detail Culling**
- A technique that sacrifices quality for speed
- Base on the size of projected BV – if it is too small, discard it.
- Also often done hierarchically
- Always helps to create a hierarchical structure, or scene graph.

**Occulusion Culling**
- Discard objects that are occluded
- Z-buffer is not the smartest algorithm in the world (particularly for high depth-complexity scenes)
- We want to avoid the processing of invisible objects

**Occlusion Culling (2)**
- **OcclusionCulling (G)**
  - Or = empty
  - For each object g in G
    - if (isOccluded(g, Or))
      - skip g
    - else
      - render (g)
      - update (Or)
  - end
- End

- **G**: input graphics data
- **Or**: occlusion representation

The problem:
1. algorithms for isOccluded()
2. Fast update Or

**Hierarchical Visibility**
- **Object-space octree**
  - Primitives in a octree node are hidden if the octree node (cube) is hidden
  - A octree cube is hidden if its 6 faces are hidden polygons
  - Hierarchical visibility test:

**Hierarchical Visibility (obj-sp.)**
- From the root of octree:
  - View-frustum culling
  - Scan conversion each of the 6 faces and perform z-buffering
  - If all 6 faces are hidden, discard the entire node and sub-branches
  - Otherwise, render the primitives here and traverse the front-to-back children recursively
**Hierarchical Visibility (obj-sp.)**
- Scan conversion the octree faces can be expensive – cover a large number of pixels (overhead)
- How can we reduce the overhead?
- Goal: quickly conclude that a large polygon is hidden
- Method: use hierarchical z-buffer!

**Hierarchical Z-buffer**
An image-space approach
- Create a Z-pyramid
  - 1 value
  - ¼ resolution
  - Original Z-buffer

**Hierarchical Z-buffer (2)**

**Hierarchical Z-buffer (3)**

Isoccluded \(g, Z_p\)

\[
\text{near } z = \text{nearZ(BV}(g))
\]

if \(\text{near } Z \text{ behind } Z_p\text{.root}\text{.z}\)

return true

else

return ( Isoccluded \(g, Z_p\text{.c}[0]\) &&

Isoccluded \(g, Z_p\text{.c}[1]\) &&

Isoccluded \(g, Z_p\text{.c}[2]\) &&

Isoccluded \(g, Z_p\text{.c}[3]\) )

end

**Hierarchical Z-buffer (4)**
Visibility \((\text{OctreeNode } N)\)

if IsOccluded \((N, Z_p)\) then return;

for each primitive \(p\) in \(N\)

render and update \(Z_p\)

def

for each child node \(C\) of \(N\) in front-to-back order

Visibility \((C)\)

def

**Some Practical Issues**
- A fast software algorithm
- Lack of hardware support
- Scan conversion
- Efficient query of if a polygon is visible (without render it)
- Z feedback
Combining with hardware

- Utilizing frame-to-frame coherence
  - First frame – regular HZ algorithm (software)
    - Remember the visible octree nodes
  - Second frame (view changes slightly)
    - Render the previous visible nodes using OpenGL
    - Read back the Z-buffer and construct Z-pyramid
    - Perform regular HZ (software)
  - What about the third frame?
  - Utilizing hardware to perform rendering and Z-buffering – considerably faster