Image Space Visibility

- Object-Space visibility does not readily support the occlusion of one big object by many small objects
  - Visibility changes faster with many small occluders, so cell based approaches work poorly
  - Identifying smaller occluders to merge is difficult
- Image-Space algorithms merge the *projections* of occluders onto the image plane
  - Merging is easier because the projections lie in a plane
  - Two issues: Which parts of the scene are occluded, and how far back are the occluders
  - Combine with an object-space hierarchy to avoid touching all objects

Hierarchical Z-buffer

(Greene, Kass, Miller, 1993)

- Conventional z-buffers store the occluding pixel depth for every pixel, and test every polygon separately
- Subdividing the world and projecting octree nodes does not help, because few nodes are occluded by single pixels
- The *hierarchical z-buffer* builds a hierarchy on the depth-buffer, and tests world space octree cells against the hierarchy
  - z-buffer hierarchy is a quadtree
  - Each node stores the maximum depth of its sub-trees (pixels)
  - Color buffer still stores color information

Hierarchical Z-buffer Run-Time

- Initialize all levels of quadtree to infinite depth
- Starting at the octree root, test world-space octree cells against the appropriate level of the depth quadtree
  - If behind max-depth, then stop
  - If not behind max-depth, then subdivide octree cell
- If a leaf node of the world octree doesn't pass the depth test:
  - Render its contents into the depth buffer
  - Push any changes in pixel depth back up the depth-buffer quadtree

More Hierarchical Z-Buffer

- Later extensions add antialiasing and other features
- Existing implementations are fully software - hardware depth buffer reads are too slow
- Performance is poor if polygons are rendered back to front
  - Temporal coherence: Render cells visible in previous frame first
- HP fx graphics chips have a feature that is related
  - A hardware query operation to tell you if a bounding volume is visible, without touching the various buffers
  - Cost is approx that of rendering 125 triangles, so you can loose if the test fails often (for example, back to front rendering)
- Can also use the "picking" feature of OpenGL

Hierarchical Occlusion Maps

(Zhang, Manocha, Hudson, Hoff, 1997)

- Similar to hierarchical z-buffer, but avoids many reads of the depth-buffer for hardware implementation
- Choose a set of good occluders as a preprocessing step
  - Same metric as Coorg/Teller: essentially projected solid angle
- In first pass, render the chosen occluders in pure white, no shading, optional depth-buffering
- Take the color-buffer and build a quadtree on it - pixel values are *average* of sub-tree values
  - Can use texture mip-mapping hardware
- Result is a multi-resolution map that tells you how well regions of the screen are occluded

Hierarchical Occlusion Maps (cont)

- When rendering, project world octree cells onto the map
- If the relevant parts of the map are gray or black, the box cell cannot be culled
- If the map is white, depth must be tested
  - Option 1: test against a plane placed behind all occluders in the map
  - Option 2: build a low-resolution depth map and test against sub-regions (requires a depth buffer read)
- Approximate visibility is supported
  - Define a visibility threshold - if the map is above the threshold, consider it white (objects are occluded)
  - Avoids, for instance, rendering objects through small holes or cracks

Directional Discretized Occluders

(Bernardini, Klosowski, El-Sana, 2000)

- Pre-process an octree subdivision of the world to locate faces of octree cells that could be used as occluders
  - Faces are marked as occluders for some subset of views
  - A marked face indicates that, for the given range of views, it will never be seen (and so nothing behind it will be seen)
- To render, project cells in top down, front to back order
  - Render occluding cell faces into a quadtree in image space
  - Use the quadtree to cull later cells
- No need for depth information, because the octree imposes an ordering - things rendered later must be behind mask
- The pre-process to determine occluding faces is very expensive

Volume-Based Visibility

- Cells and portals without the portals
  - Portals guided the search for potentially visible objects
  - Without portals, naïve pre-processing is extremely expensive
- Occluder Fusion (Schaufler, Dorsey, Decoret, Sillion, 2000)
  - Occlusion by volumetric elements (voxels), not surfaces
  - Observe that if an occluding voxel bounds the "shadow" of another occluding voxel, the occlusion region can be extended
  - Massive culling achieved in city environments
- Extended Projections (Durand, Drettakis, Thollot, Puech, 2000)
  - Project occluding objects onto a plane at fixed depth
  - Careful analysis to detect more occlusions, and algorithms to exploit hardware

Managing Dynamic Scenes

- To determine if a dynamic object is visible
  - Bound the object with a *temporal bounding volume* (TBV)
    - If the TBV is invisible, so is the object
  - Test for visibility of bound
  - Little work on the relative payoff between:
    - Smaller, less visible bounds evaluated more frequently, or,
    - Bigger, more visible bounds evaluated less frequently
  - Little work on efficiently computing and managing temporal bounding volumes
- Little, if any, work on using dynamic objects as occluders
- The primary source of open problems in visibility
  - Also related to dynamic radiosity