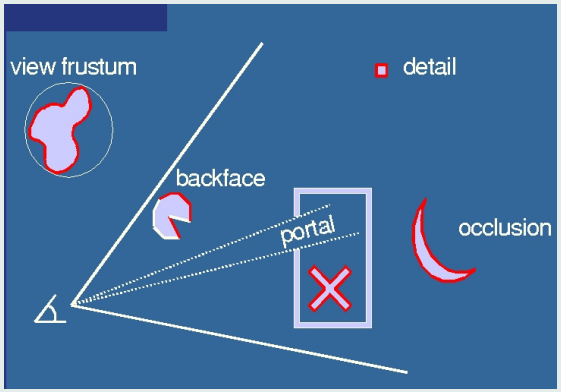
Culling Techniques

- Cull: **"Remove from a flock"**
- In graphics, culling removes parts of a scene that do not contributed to the final image
- Fastest polygon to render - the one that is never sent to the renderer!
- Focus on culling as relates to rendering
- Culling
- Classify techniques as backface, view frustum and occlusion culling.
- Can take place at the application, geometry or rasterizer (hardware assisted)

Culling Techniques



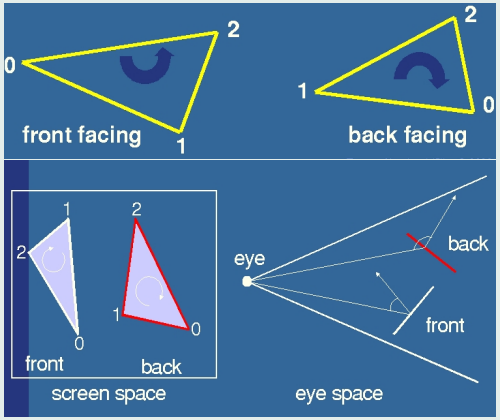
Culling Techniques: Terminology

- Exact Visible Set(EVS): the set of polygons that is part of the visible surface; extremely expensive to compute from any point of view.
- Potentially Visible Set(PVS): Prediction of the EVS
- By approximating the EVS, in general, PVS can be computed faster.
- Ultimate visibility is determined by the Z-buffer.

Backface Culling

- Discards polygons facing away from the viewer
- Application: Closed surfaces (example: sphere), room walls
- Can be done in screen space or eye space
- Computation: $\mathbf{n} = (\mathbf{v}_1 - \mathbf{v}_0) \times (\mathbf{v}_2 - \mathbf{v}_0)$
- Screen Space: $\mathbf{n} = (0, 0, a)$ or $(0, 0, -a)$
- Eye Space: $\mathbf{n} \cdot \mathbf{V} > 0$
- Benefits: Less workload on the rasterizer or geometry stage

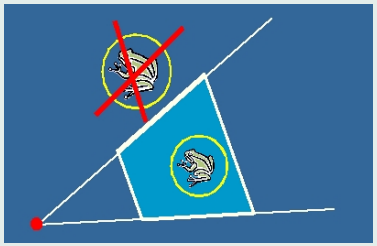
Backface Culling



Clustered Backface Culling

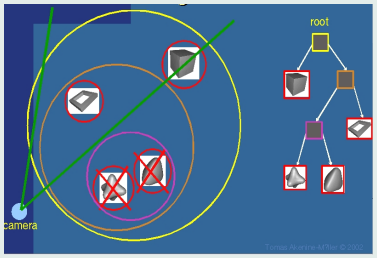
- Group normals of objects into clusters and enclose by solid cones (front-facing, backfacing cones).
- Tests with cones can eliminate collections of polygons from the rendering stage
- Alternately, discretize directions into frustums that emanate from the center of a cube.
- Refer text, for details.

View Frustum Culling



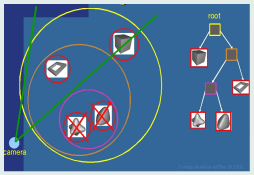
- Bound every natural group of primitives by a simple volume (e.g., sphere, box)
- If a Bounding Volume (BV) is outside the view frustum, then the entire contents of BV is not visible
- Can we further accelerate view frustum culling? Use a hierarchy

Hierarchical View Frustum Culling



- Can use Bounding Volume Hierarchy (BVH) - using preorder traversal
- If a node's BV is outside the frustum, then discard all geometry under it, else must test its children (recursively)
- If leaf nodes are intersected, geometry inside them go through the pipeline, clipping resolves the visibility.

Hierarchical View Frustum Culling



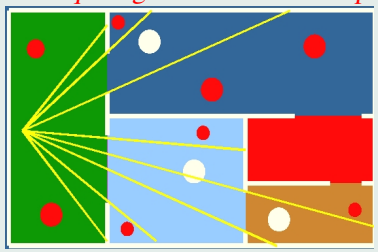
- Typically view frustum culling operates in the application stage (CPU), benefits geometry, rasterizer stages
- Can also use BSP trees, octrees, but they are not flexible for dynamic scenes
- Polygon-aligned BSP trees are simple and useful - tests against the splitting plane determine the pruning, traversal order
- Can exploit frame-to-frame coherence by caching and updating planes intersected with the spatial structure

Portal Culling

- Culling specific to architectural type models.
- Rationale: Exploit the fact that walls act as large occluders in indoor scenes.
- Thus, perform (diminished) view frustum culling through each portal
- Must preprocess the scene into a set of cells - usually rooms, hallways; doors, windows that connect rooms are the portals
- Preprocessed scene maintained in an adjacency graph

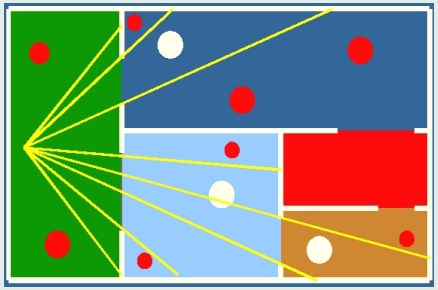
Portal Culling (Luebke 1995)

Simple Method requiring a small amount of preprocessing



- Locate cell V containing viewer
- Initialize 2D bounding box P to screen rectangle
- Render geometry of V starting from viewer, and through P.
- Recurse on portals neighboring V; project portal onto P, using axis-aligned BB (BB_{aa} and perform $P \bigcap BB_{aa}$.

Portal Culling Example



- **Non-Empty Intersection:** Perform view frustum culling using intersection rectangle to form the view frustum.
- **Non-Empty Intersection:** Recurse on neighbor of projected cell.
- **Termination Criteria**: (1) Current AABB is empty, (2) out of time to render cell (that is ”far away”)

Portal Culling Example

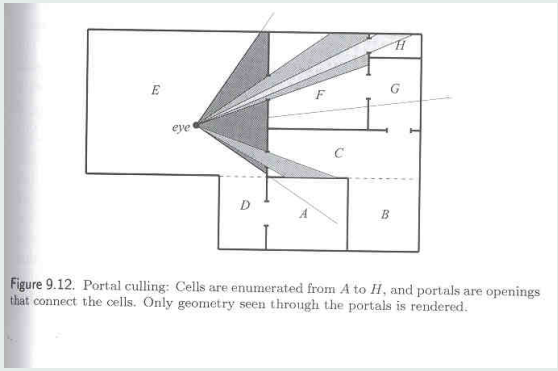
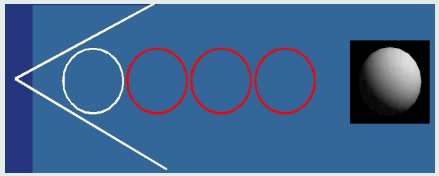


Figure 9.12. Portal culling: Cells are enumerated from A to H, and portals are openings that connect the cells. Only geometry seen through the portals is rendered.

Detail Culling

- Technique that sacrifices quality for speed
- Small details contribute little or nothing, especially during motion (highly dynamic environments).
- Method: Project BV of object onto screen, estimate area in pixels and remove object if below a threshold
- Also known as screen size culling, this can also be thought of as a simplified LOD technique

Occlusion Culling



- Visibility using the Z-Buffer can be very inefficient.
- Example: Consider rendering a sequence of spheres
- All spheres are rendered, but only the closest is visible!
- Depth Complexity: the number of times a pixel is overwritten
- **Real-life examples:** rain forest, city models, interior of skyscraper, engine block
- Occlusion Culling: Methods to avoid these inefficiencies

Occlusion Culling(OC)

- Algorithmic approaches to avoid drawing related inefficiencies.
- Optimally, only visible objects are drawn.
- Occlusion culling algorithms perform simple tests early on to avoid sending objects down the graphics pipeline.
- Types of occlusion culling: point-based, cell-based
  - **Point-Based OC:** visibility from a single viewing location.
  - **Cell-Based OC:** visibility from a region of space
  - OC based on image(2D), object(3D) or ray space.
- Cell based OC is more expensive to compute, but valid for a few frames

Occlusion Culling Algorithm

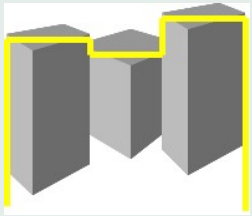
- Given Occlusion Representation (O_R), the set of objects G,

for each object g
  $O_R = \varnothing; P = \varnothing$
  for each object $g \in G$
    if (IsOccluded(g, O_R))
      Skip (g)
    else
      Render (g)
      Add (g, P)
      if (LargeEnough (P))
        Update (O_R, P)
        $P = \varnothing$
      end
    end
  end

Issues in Occlusion Culling Algorithms

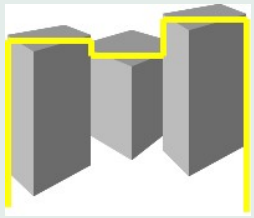
- Occluded objects are not processed.
- Potentially visible objects: Has to be rendered, added to P, might undergo merging (fusion), based on occluding power of the set.
- Want to fuse objects that can form the best occluders for incoming objects.
- Some algorithms cannot merge occluders, some update O_R only once per frame
- Object ordering matters, rough front to back sorting performed by some algorithms.
- Object distance matters; matchbox can occlude Golden Gate Bridge!

Occlusion Horizons (Wonka,Schmalstieg 1999, Downs 2001



- Targeted at rendering urban environments.
- Objects composed of buildings and ground (2.5D) treated as a height field
- Occlusion horizon is constructed on the fly and used to cull incoming objects (roughly front to back order)
- Point Based occlusion culling

Occlusion Horizons Method

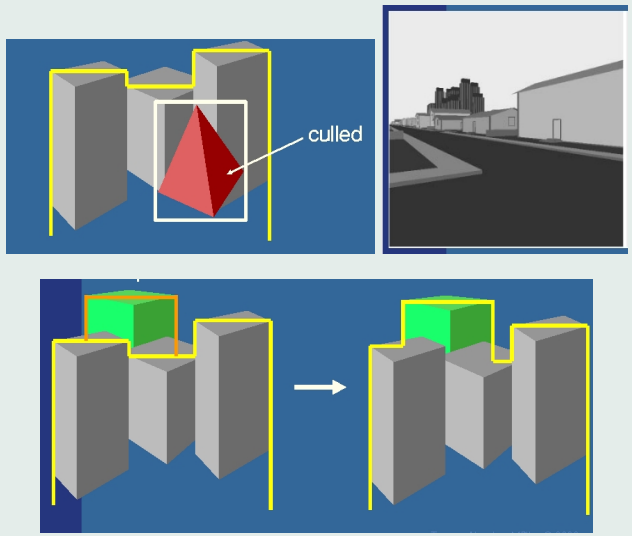


- Quadtree of the XY plane of the scene is built
- Sweep a plan parallel to near-plane from the viewer, front to back.
- Objects to be culled are overestimated with a bounding volume, occluders are underestimated with Convex Vertical Prisms (CVP)
- Union of CVPs and ground mesh forms a height field, and a conservative estimate of occluding power.
- Horizon representation: piecewise constant function of x.

Occlusion Horizons: Algorithm

- Initially, horizon is a horizontal line, with $y = 0$.
- **Horizon Rep:** 1D binary tree, with nodes maintaining range of x values (x_{min}, x_{max}); leaf nodes store y (constant).
- Internal nodes store y_{min} and, y_{max})
- **Processing**:
  - Priority queue, Q is used to process objects in front to back order, sortez by smallest Z.
  - Pop object, project BB to screen, compute 2D AABB and test with the horizon (binary tree).
  - Reject if object below horizon, else will be added to update the horizon
  - Fusing the Occluder: Done based on farthest Z
- Additional Optimizations: Refer Text.

Occlusion Horizons: Example



Shaft Occlusion Culling (Schauffler, 2000)

- Cell based, operates in object space
- One of the first cell-based schemes that could handle 3D scenes
- Applications: Visibility preprocessing for walkthroughs, ray tracing with area lights.
- Represent view cells using AABBs
- Quadtree describes the occluding power of scene
- Leaf node is either opaque, boundary or empty

Shaft Occlusion Culling:Method

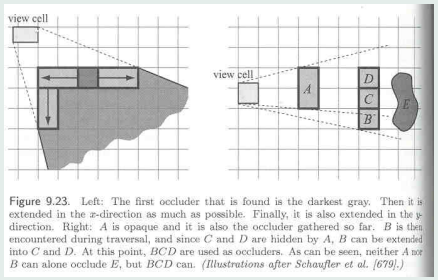


Figure 9.23. Left: The first occluder that is found is the darkest gray. Then it is extended in the x-direction as much as possible. Finally, it is also extended in the y-direction. Right: A is opaque and it is also the occluder gathered so far. B is then encountered during traversal, and since C and D are hidden by A, B can be extended into C and D. At this point, BCD are used as occluders. As can be seen, neither A nor B can alone occlude E, but BCD can. *(Illustrations after Schaufler et al. [679].)*

- Starting from view cell, traverse quadtree for opaque node
- Traversal is performed outwards from the view cell to maximize occlusion
- Extend occluder along the coordinate axis that gives the largest angle from view cell to occluder (can grow in 2 directions)
- Hidden Leaf nodes can also be considered to be part of the occluder
- In 3D, can extend in 3 directions and an octree is used.

Hardware Occlusion Queries

- HP Visualize FX Graphics:
  - Supports occlusion queries, compares a set of polygons to current Z-buffer contents.
  - Typically bounding volume polygons used.
  - Performance: 25 to 100 percent, long latency.
- NVIDIA:
  - NV query, returns the number of pixels that pass the depth test
  - Cull object if $n = 0$, or $n > thresh$ (speed/quality tradeoff) or use to determine LOD.
  - Can test if the result is back, else return control to application, and allows multiple queries in parallel.
- Bounding volumes and hierarchies can be used to further optimally use the hardware queries (see text).

Hierarchical Z-Buffering (Greene 93, 95)

- Signficance influence on occlusion culling research
- Potential for hardware implementation, real-time applications.
- Scene model in an octree, frame's Z-buffer is a pyramid
- Octree enables object-space culling, while Z-buffer pyramid enables hierarchical Z-buffering of primitves, bounding volumes
- Z-Pyramid is the occlusion representation.

Hierarchical Z-Buffering (HZB) Algorithm

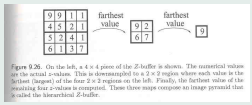


Figure 9.26. On the left, a 4×4 piece of the Z-buffer is shown. The numerical values are the actual z-values. This is downsampled to a 2×2 region where each value is the farthest (largest) of the four 2×2 regions on the left. Finally, the farthest value of the remaining four z-values is computed. These three maps compose an image pyramid that is called the hierarchical Z-buffer.

- Z-Pyramid Construction: Finest level is the standard Z-buffer, all other levels - Z value is the farthest of the 2×2 window of adjacent finer level.
- If a Z value is overwritten, the coarser levels are updated in sequence.
- Hierarchical Culling:
  - BB of octree (screen projected) is tested against the coarsest level
  - If face's nearest depth (z_{near}) is farther, the face is culled, else continue down the Z-pyramid
  - If bottom level of Z-pyramid reached, and octree box is visible, testing continues down the octree.

Hierarchical Z-Buffering (HZB) Algorithm

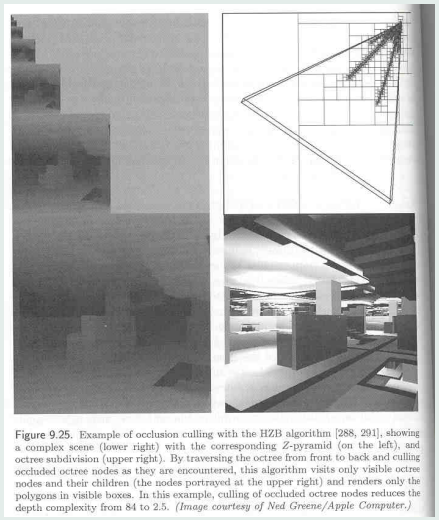


Figure 9.25. Example of occlusion culling with the HZB algorithm [288, 291], showing a complex scene (lower right) with the corresponding Z-pyramid (on the left), and octree subdivision (upper right). By traversing the octree from front to back and culling occluded octree nodes as they are encountered, this algorithm visits only visible octree nodes and their children (the nodes portrayed at the upper right) and renders only the polygons in visible boxes. In this example, culling of occluded octree nodes reduces the depth complexity from 84 to 2.5. *(Image courtesy of Ned Greene/Apple Computer.)*

HZB Algorithm: Real-time Applications

- Modify hardware Z-buffer pipeline to support HZB, requiring a Z-pyramid
- This will enable rendering considerably more complex scenes
- Permit the tip of the pyramid to be read out from H/W to the application, permitting culling at the beginning of the pipeline.
- Additional Optimizations: BSP trees with coverage masks, frame-to-frame coherence (visibility changes are slow, so visible octree nodes are cached), etc.

Level Of Detail(LOD)

- Basic Idea: Use simpler versions of an object as it contributes lesser and lesser to an image
- No need to draw 10,000 polygons of a Ferrari, if it projects to a 10×5 pixels on the screen!
- Using simpler models leads to significant rendering speedup
- Can be combined with fog rendering
- LOD algorithms have three major components, generation, selection, switching
- Generation: A topic in of itself, significant amount of work in polygon simplification algorithms (Chapter 11)

LOD Example

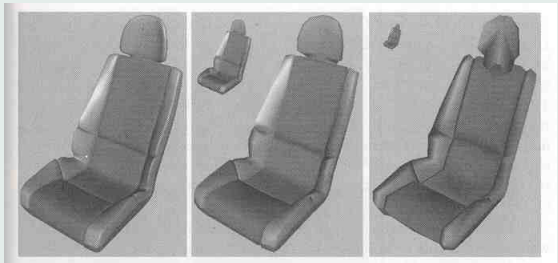


Figure 9.29. Here we show three different levels of detail for a car chair. The two pictures in the upper left corner of the middle and right images show the two chairs farther away from the viewer; still, these simplified models seem to approximate the chair fairly well from these distances. Note also how the shading degenerates as fewer triangles are used. *(Model reused courtesy of Nya Perspektiv Design AB.)*

LOD Switching

Abrupt LOD model switching during rendering is distracting and leads to popping

Discrete Geometry LODs

- Represent each LOD of the object with differing numbers of primitives, usually as triangle strips, and pulled in from DMA memory
- Popping is the worst for this method, as LODs can switch on any frame

LOD Switching:Blend LODs

- To void popping, do a linear blend from one LOD to another over a few frames
- Smoother switch, but we are rendering two LODs, which can be expensive.
- However, only over a few frames and not all objects are switching simultaneously.
- Method: Draw the current LOD (say LOD1) opaquely, then fade in LOD2 (alpha from 0 towards 1), switch LOD to LOD2, then fade out LOD2 (fade in/out with Z-test on, Z-writes disabled).

LOD Switching:Alpha LODs

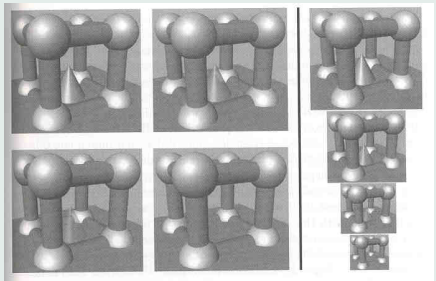


Figure 9.30. The cone in the middle is rendered using an alpha LOD. The transparency of the cone is increased when the distance to it increases, and it finally disappears. The images on the left are shown from the same distance for viewing purposes, while the images to the right of the line are shown at different sizes.

- Use transparency to fade out objects, based on LOD criteria
- Example, when objects move away from viewer, they become gradually more and more transparent and finally disappear.

LOD Switching:CLODs and Geomorph LODs

- Generating LOD models using edge collapse (an example of simplification algorithms) can also provide smoother transitions
- Reducing the number of primitives in models across LODs can create a Continuous Level of Detail (CLOD).
- Removing an edge, edge collapse or adding an edge, vertex split in a mesh is the basis for generating CLODs.
- Disadvantage: Triangle strips are more difficult to use with CLODs.
- Geomorphs: Vertex connectivity between LODs are maintained and linear interpolation is performed to make smooth transitions.

LOD Selection

When and on what basis do we switch LODs?

- Range-Based:
  - Each LOD of an object is associated with a range of distances from the viewer, eg. $0 < r_1 < r_2 < \cdots$.
  - LOD1 corrsponds to $(0, r_1)$, LOD2 corresponds to (r_1, r_2), etc.
- Projected Area-Based:
  - Size of area projected by object decreases with viewer distance.
  - Can use BV of object to simplify calculations, eg., spheres, boxes (details,formulas in section 9.8.2).

LOD Selection

- Hysteresis:
  - If the LOD metric varies too frequently (around a boundary), popping will occur.
  - Introduce hysteresis, so that the LOD switch is asymmetric when view distance is increasing vs. when its decreasing
- Other Methods
  - Based on importance of objects, motion, focus
  - Constraints based on a polygon budget, textures.