Visibility and Occlusion Culling

CS334 Fall 2015

Daniel G. Aliaga
Department of Computer Science
Purdue University

[some slides based on those of Benjamin Mora]
Why?

- To avoid processing geometry that does not need to be processed.
- Useful when having millions (or billions) of triangles.
- Can be done at the triangle or pixel level.
Topics

• Older Concerns
• View Frustum Culling
• Back Face Culling
• Portal Culling
• Occlusion Culling
  – Hierarchical Z Buffer
  – Hierarchical Occlusion Maps
  – Some others...
Topics

• Older Concerns
• View Frustum Culling
• Back Face Culling
• Portal Culling
• Occlusion Culling
  – Hierarchical Z Buffer
  – Hierarchical Occlusion Maps
  – Some others...
Hidden Surfaces Removal

• Needed before raster-graphics and high memory availability
Painter’s Algorithm

- Primitive can be projected from the farthest one to the closest (Back-to-Front) without any need to find the closest intersection.
- All the primitives being on the same side of the viewpoint (A) will be intersected first, so a front-to-back analysis is more efficient.
Priority List/BSP Trees (Fuchs)
Priority List/BSP Trees (Fuch)

• Proposed by Henry Fuchs (1980)
• By preprocessing the scene and creating a binary space subdivision, primitives can then be projected in a visibility order.
• Handles Transparency correctly.
• In practice, constructing a Binary Space Partitioning tree is difficult!
Area Subdivision Algorithms

- Warnock Algorithm (1969)
  - Not really used anymore
  - Hierarchical method
  - Quadtree structure
  - Assumes no overlapping
  - Visibility is computed on a per-block basis
Area Subdivision Algorithms

- **Weiler-Atherton algorithm**
  - Works on 3D primitives instead of using screen space subdivisions.
  - Maintains a list of clipped (visible) polygons.
    - Every time a new polygon is processed, clipping with all the (visible) polygons is performed and a new list of polygons is generated.
    - Once all the polygons processed, the clipped parts can be used for the final image.
Area Subdivision Algorithms

• Weiler Atherton algorithm.
  – Provides a general clipping algorithm for concave polygons.

– Other algorithm for clipping: Sutherland-Hodgman algorithm

![Diagram of clipping polygons]
Possible Issues with these algorithms

• Could be too complex (e.g., 2D clipping).
  – Lead to several cases and bugs in implementations.

• Not fast enough.
  – E.g. clipping.

• Hardly parallelizable.
  – Not suitable to hardware acceleration.
Topics

• Older Concerns
• **View Frustum Culling**
• Back Face Culling
• Portal Culling
• Occlusion Culling
  – Hierarchical Z Buffer
  – Hierarchical Occlusion Maps
  – Some others...
Culling

• **View Frustum Culling.**
  – Primitives outside of the view frustum discarded.
  – Often implemented on graphics cards
    • Note: not the same as clipping!

[Diagram of viewing pyramid and culling]

- Viewing pyramid
- Viewpoint
- Visible Triangle
- Culled triangles
Culling

- Can be accelerated (software) by grouping triangle and computing the frustum intersection only for the bounding box.

Don’t need to process these triangles.
Hierarchical View Frustum Culling
Topics

• Older Concerns
• View Frustum Culling
• Back Face Culling
• Portal Culling
• Occlusion Culling
  – Hierarchical Z Buffer
  – Hierarchical Occlusion Maps
  – Some others...
Back Face Culling

- If the viewpoint is fronting the backface, then the triangle is not processed.
- Hardware accelerated
- Can be enabled/disabled.
- Orientation given by the order of projected vertices.
Topics

• Older Concerns
• View Frustum Culling
• Back Face Culling
• **Portal Culling**
• Occlusion Culling
  – Hierarchical Z Buffer
  – Hierarchical Occlusion Maps
  – Some others...
Cells and Portals Culling
Cells and Portals Culling

• A visibility graph is constructed and used for macro-level scene culling
Image-based C & P Culling

• Increase interactive rendering performance of architectural walkthroughs
  – Cull away non visible “areas”
  – Replace geometry behind portals (doorways and windows) with images warped to the current eye position
  – Reduce the rendering complexity with a small number of image samples
Portal Images
Adding Portal Image Warping

McMillan & Bishop Warping Equation:

\[ x_2 = \delta(x_1) \ P_2^{-1} (c_1 - c_2) + P_2^{-1} P_1 x_1 \]

- Move pixels based on distance to eye
- ~Texture mapping

• Per-pixel distance values are used to warp pixels to their correct location for the current eye position
Portal Image Warping

- Projecting the eye onto the reference image partitions it into 1, 2 or 4 sheets
- A raster scan of each sheet produces a back-to-front ordering of warped pixels
Creating Portal Images

Ideal portal image would be one sampled from the current eye position
Creating Portal Images

A large number of static reference images (~120) needed to eliminate *popping*
Creating Portal Images

With warping we use a much smaller number of reference images.
Creating Portal Images

Choose closest reference image and warp to current eye position
Problem: Exposure Events

We observe tears in regions with unknown visibility data.
Reducing Exposure Events

To address this we:

1. Render multiple reference images
2. Allow warped pixels to persist and fill-in minor tears
Parallelization: Sheets

Eye view direction similar to reference COP view direction

Since the projection plane is perpendicular to the view direction, we usually have 4 sheets

We warp each sheet on a different processor
Topics

• Older Concerns
• View Frustum Culling
• Back Face Culling
• Portal Culling
• Occlusion Culling
  – Hierarchical Z Buffer
  – Hierarchical Occlusion Maps
  – Some others...
Z-Buffer test

• Example:

Final image

Final z-buffer
Z-Buffer test

• Used by current hardware technology
• Primitives can be sent to the graphics hardware in any order, thanks to the z-buffer test that will keep the closest fragments.
• A z value is stored for every pixel (z-buffer)
• Algorithm for a given pixel:
  – If the rasterized z-value is less than the current z-value
  – Then replace the previous color and z-value by the new ones
Hierarchical Z-buffer

- Proposed by Greene et al.
- A hierarchical z-max pyramid is constructed above the z-buffer
- Every time a z value is updated, the hierarchy is updated
The Z-Pyramid

• The content of the Z-buffer is the finest level in the pyramid

• Coarser levels are created by grouping together four neighbouring pixels and keeping the largest z-value

• The coarsest level is just one value corresponding to overall max z
The Z-Pyramid

- □ = furthest
- □ = closer
- □ = closest

Depth taken from the z-buffer

Objects are rendered

Construct pyramid by taking max of each 4
Using The Z-Pyramid

☐ = furthest
■ = closer
■■ = closest
Maintaining the Z-Pyramid

• Ideally every time an object is rendered causing a change in the Z-buffer, this change is propagated through the pyramid

• However this is not a practical approach
More Realistic Implementation

• Make use of frame to frame coherence
  – at start of each frame render the nodes that were visible in previous frame
  – read the z-buffer and construct the z-pyramid
  – now traverse the octree using the z-pyramid for occlusion but without updating it
HZB: Discussion

- It provides good acceleration in very dense scenes
- Getting the necessary information from the Z-buffer is costly
- Some hardware support exists
Hierarchical Occlusion Maps

(all images from Hansong Zhang’s dissertation)
Hierarchical Occlusion Maps

• Motivation
  – Occlusion is cumulative
  – Occluders can be “fused”
Hierarchical Occlusion Maps

• Triangles are initially grouped into separate regions of space such as bounding boxes, grids or trees.
• This data structure is traversed in a front to back order.
  – The visibility of the region is given by the HOM.
  – If region is visible, then its content (i.e., triangles) must be projected.
  – During projection, the content of the different maps is updated every time a pixel becomes opaque.
Hierarchical Occlusion Map (HOM)

• The occlusion images can be readily generated by hardware; how?
  – Just render the scene with white=visible and black=background

• Note that this process is at some selected resolution
  – So let's change the resolution!
Hierarchical Occlusion Map (HOM)

- And we get this:
Hierarchical Occlusion Map (HOM)

• And we get this:
Hierarchical Occlusion Map (HOM)

- Creation algorithm for HOMs:

  1. Read framebuffer for occlusion map.
  2. Copy framebuffer to the texture memory.
  3. Render square half of the current size with proper texture coordinates.
  4. Framebuffer (Originally the rendered image of the occluder scene).
Hierarchical Occlusion Map (HOM)

• How do you use an HOM?
  – First cull using the coarsest HOM
  – Then the next coarsest...
  – Until the finest level of the HOM
  – If something is visible in all levels, then it is visible
  – So what is the gain?
Hierarchical Occlusion Map (HOM)

• Observation
  – Even at the finest level, it is not known if the object is “truly” visible
  – But at the resolution you care about, it is...
Hierarchical Occlusion Map (HOM)

• Conclusion
  – The hierarchy is conservative
  – Generate from the coarest HOM to the finest level you care about
  – Only use those and select at run-time depending on performance
Example Results
Some questions?

• For a 2x2 grid of pixels in the occlusion map, what occlusion value do you choose?
  – Min?
  – Max?
  – Average?

• See hierarchical z-buffers!
Occlusion Queries

• Current graphics hardware allows counting the number of fragments that pass the z-test.

• 3 steps:
  – Lock the z-buffer and frame buffer (impossible to modify the content of these buffers).
  – Render a bounding box.
    • If no pixel has passed the z-test, then the region inside the BB is not visible.
  – Unlock the Z-buffer and Frame-Buffer.
Occlusion Queries

• Can take advantage of fast $z$ tests.

• Efficient only if the BB contains many primitives.

• Must wait for the answer.
  
  – The program should do something before testing the answer.
Depth Peeling

From

Interactive Order-Independent Transparency
Cass Everitt
NVIDIA OpenGL Applications Engineering
Depth Peeling

Layer 0

Layer 1

Layer 2

Layer 3

Figure 3. These images illustrate simple depth peeling. Layer 0 shows the nearest depths, layer 1 shows the second depths, and so on. Two-sided lighting with vivid coloring is used to help distinguish the surfaces.

From
Interactive Order-Independent Transparency
Cass Everitt
NVIDIA OpenGL Applications Engineering
Depth Peeling

• Z-buffer based visibility only allows finding the nearest elements of the scene.
  – Transparency cannot correctly be handled.

• Solution: use a multipass algorithm to find the second nearest surface, third nearest surface, and etc... for every pixel
  – At the end, combine the different images in a front-to-back-order.