Occlusion Culling Algorithms

Developers are always going to want better performance in real-time rendering, and so speed-up tec
acceleration schemes will always be needed. In this excerpt from Chapter 7, "Speed-Up Techniques," o
Rendering, the authors discuss the class of acceleration schemes known as the occlusion culling tech

by Tomas Möller

Excerpted from *Real-Time Rendering*
(AK Peters, 1999)

One of the great myths concerning computers is that one day we will have enough proce
Even in a relatively simple application like word processing, we find that additional power
applied to all sorts of things, such as on-the-fly spell and grammar checking, more elabo
presentation, antialiased text display, automated voice recognition and dictation, etc.

In real-time rendering we have at least three performance goals: more frames per secon
resolution, and more (and more realistic) objects in the scene. A speed of 60-72 frames p
generally considered enough, and perhaps 1600x1200 is enough resolution for a while, but
real upper limit on scene complexity. The rendering of a Boeing-777 would include 132,500
parts and over 3,000,000 fasteners, which would yield a polygonal model with over 500,00
polygons [Cripe98]. Our conclusion: speed-up techniques and acceleration schemes will
needed.

In this article, we will talk about a certain class of acceleration scheme called *occlusion c*
techniques. Most of this article is an excerpt from chapter 7, "Speed-Up Techniques" from
Real-Time Rendering (www.realtimerendering.com or www.acm.org/tog/resources/RTR/)
the occlusion culling section is preceded by sections on backface and clustered culling, h
view-frustum culling, portal culling, and detail culling. Sections on impostor algorithms, le
techniques, triangle fan, strip and polygon mesh techniques follow after.

To *cull* can mean to "select from a flock," and in the context of computer graphics this is e
culling techniques do. The flock is the whole scene that we want to render, and the selecti
to those portions of the scene that are not considered to contribute to the final image. The

smart mechanism in all respects. For example, it has the following implications. Imagine t
viewer is looking along a line where 10 spheres are placed. This is illustrated in Figure 1.

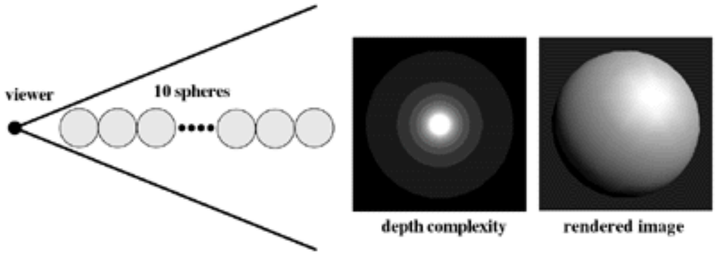

Figure 1. An illustration of how occlusion culling can be useful. Ten spheres are placed in a line, and looking along this line (left). The depth complexity image in the middle shows that some pixels are several times, even though the final image (on the right) only shows one sphere.

An image rendered from this viewpoint will show but one sphere, even though all 10 spher scan-converted and compared to the Z-buffer and then potentially written to the color bu buffer. The simple conclusion in this case is that nine spheres will be drawn unnecessarily uninteresting scene is not that likely to be found in reality, but it describes (from its viewpo densely populated model. These sorts of configurations are found in such real scenes as an engine, a city, and the inside of a skyscraper.

Thus it seems plausible that an algorithmic approach to avoid this kind of inefficiency ma terms of speed. Such approaches go under the name of *occlusion culling algorithms*, sinc cull away (avoid drawing) objects that are occluded, that is, inside the view frustum but n the final image. The optimal occlusion culling algorithm would select only the objects tha In a sense, the Z-buffer selects and renders only those objects which are visible, but not b objects are sent through the pipeline. The idea behind efficient occlusion culling algorithm perform some simple tests early on and so avoid sending data through much of the pipe

Pseudocode for a general occlusion culling algorithm is shown in Figure 2, where the func isOccluded, often called the *visibility test*, checks whether an object is occluded. *G* is the s geometrical objects to be rendered; *OR* is the occlusion representation.

```
OcclusionCullingAlgorithm (G)
OR=empty
for each object g in G
if(isOccluded(g,OR))
Skip(g)
else
Render(g)
Update(OR,g)
end
end
```

updated with that object.

For some algorithms, it is expensive to update the occlusion representation, so this is only
(before the actual rendering starts) with the objects that are believed to be good occlude
then updated from frame to frame.

A number of occlusion algorithms will be scrutinized in this section.

Hierarchical Z-Buffering and the Hierarchical Visibility Algorithm

One approach to occlusion culling is the *hierarchical visibility* (HV) algorithm [Greene93].
algorithm maintains the scene model in an octree, and a frame's Z-buffer as an image py
which we call a Z-pyramid. The octree enables hierarchical culling of occluded regions of
and the Z pyramid enables hierarchical Z-buffering of individual primitives and bounding
Z-pyramid is thus the occlusion representation of this algorithm. Examples of these data s
shown in Figure 3.

Any method can be employed for organizing scene primitives in an octree, although Gree
[Greene93] recommend a specific algorithm that avoids assigning small primitives to lar
nodes. In general, an octree is constructed by enclosing the entire scene in a minimal axis
box. The rest of the procedure is recursive in nature, and starts by checking whether the b
fewer than a threshold number of primitives. If it does, the algorithm binds the primitives t
and then terminates the recursion. Otherwise, it subdivides the box along its main axes us
planes, thereby forming eight boxes (hence the name octree). Each new box is tested and
subdivided again into 2x2x2 smaller boxes. This process continues until each box contains
the threshold number of primitives, or until the recursion has reached a specified deepest
[Samet89a,Samet89b]. This is illustrated in two dimensions, where the data structure is ca
quadtree, in Figure 4.

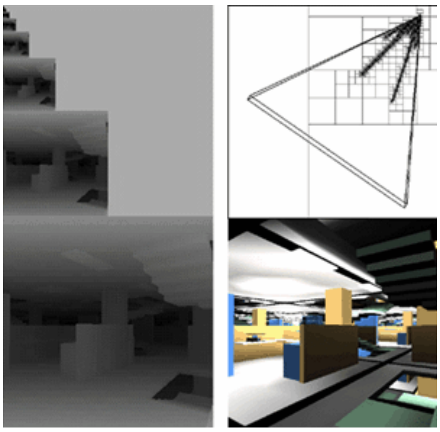

**Figure 3. Example of occlusion culling with the hierarchical visibility algorithm [Greene95], showing
scene (lower right) with the corresponding Z-pyramid (on the left), and octree subdivision (upper r
traversing the octree from front to back and culling occluded octree nodes as they are encountered
algorithm only visits visible octree nodes and their children (the nodes portrayed at the upper right
renders the polygons in visible boxes. In this example, culling of occluded octree nodes reduces the**

to process the contents of that box further, since its contents do not contribute to the fina
Otherwise, we render the primitives associated with the node into the Z-pyramid (*tileInto*
pseudocode) and then process each of the node's children (if it has any) in front-to-bac
this same recursive procedure. When recursion finishes, all visible primitives have been *til*
pyramid, and a standard Z-buffer image of the scene has been created.

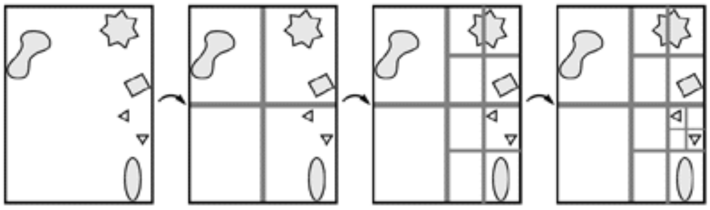

Figure 4. The construction of a quadtree (which is the two-dimensional version of an octree). The c starts from the left by enclosing all objects in a bounding box. Then the boxes are recursively divide equal-sized boxes until each box (in this case) is empty or contains one object.

The HV algorithm performs occlusion culling very efficiently because it only traverses visi
nodes and their children, and it only renders the primitives in visible nodes. This can save
work in scenes that are densely occluded. For example, in the scene pictured in Figure 3, r
99% of on-screen polygons are inside occluded octree nodes, which are therefore culled
pyramid [Greene95].

```
ProcessOctreeNode(OctreeNode N)
if(isOccluded(NBV, ZP)) then return;
for each primitive p in N
tileInto(p, ZP)
```

SU

```
ProcessOctreeNode(C)
end
```

Figure 5. Pseudocode for the hierarchical visibility algorithm. To render a frame this procedure is ca root node of the octree. *NBV* is the bounding volume of the octree node *N*, and *ZP* is the Z-pyramid th occlusion representation of this algorithm. The operation tileInto renders a primitive p into the Z-py this also updates the entire Z-pyramid.

Now we will describe how the Z-pyramid is maintained and how it is used to accelerate c
finest (highest-resolution) level of the Z-pyramid is simply a standard Z-buffer. At all othe
z-value is the farthest z in the corresponding 2x2 window of the adjacent finer level. There
value represents the farthest z for a square region of the screen. To maintain a Z-pyramid
z-value is overwritten in the Z-buffer it is propagated through the coarser levels of the Z-p

Figure 6. On the left, a 4x4 piece of the Z-buffer is shown. The numerical values are the actual z-valu downsampled to a 2x2 region where each value is the farthest (largest) of the four 2x2 regions on th the farthest value of the remaining four z-values is computed. These three maps compose an imag which is called the hierarchical Z-buffer.

Next, we describe how hierarchical culling of octree nodes is done. To determine whether visible, the front faces of its bounding box are tested against the Z-pyramid. The node is o of its front faces are occluded by the Z-pyramid. To establish whether an individual face i we begin at the coarsest Z-pyramid cell that encloses the face's screen projection. The fa depth within the cell
(znear) is then compared to the Z-pyramid value, and if znear is farther, the face is know occluded. For densely occluded scenes, this procedure often culls an occluded face with depth comparison. When this initial test fails to cull a face, its visibility can be definitively by recursively traversing from the initial Z-pyramid cell down to finer levels in the Z-pyram Additional depth comparisons at the encountered child cells must be performed. If this su procedure does not ultimately find a visible sample on the face, the face is occluded. In s the bounding boxes of octree nodes overlap deeply on the screen, this hierarchical proce establish visibility much more efficiently than can conventional Z-buffering. For example, pictured in Figure 3, hierarchical culling of octree nodes with the Z-pyramid generates rou hundred times fewer depth comparisons than visibility testing with conventional Z-buffer

As we have seen, the original HV algorithm [Greene93] used a Z-pyramid only for occlusi employed traditional Z-buffer scan conversion to render the polygons in visible octree no Subsequently, a more efficient method, called *hierarchical polygon tiling* [Greene96], wa This algorithm adapts the basic screen subdivision procedure described above to polygo is, finding the visible samples on a polygon. When tiling a polygon into the Z-pyramid, it m determined, at each level of subdivision where the polygon is compared to a Z-pyramid c the polygon overlaps the cell and if so, whether the polygon is occluded by the cell. Overl accelerated with coverage masks that indicate which subcells within a Z-pyramid cell ar the polygon, and occlusion tests are performed by comparing the polygon's nearest z-va cell to the Z-pyramid value. This procedure for hierarchical Z-buffering is very efficient, be traverses regions of the screen where a polygon is visible or nearly visible.

Hierarchical polygon tiling can be performed even more efficiently if the polygons in a sc organized into a BSP tree or an "octree of BSP trees" [Greene96]. The reason is that this en traversing polygons in strict front-to-back order, which eliminates the need to maintain d information. Rather, the only occlusion information required is whether or not each image been written, and this information can be maintained in an image pyramid of coverage r a *coverage pyramid*. This is the data structure maintained by hierarchical polygon tiling v coverage masks [Greene96], which is similar to hierarchical Z-buffering except that occlu are performed with coverage-mask operations instead of depth comparisons, which acc considerably.

The HV algorithm implemented with hierarchical tiling may well be the most efficient met for software rendering of complex scenes composed of polygons, but it is not fast enough time rendering of complex scenes on today's microprocessors. To enable real-time rende et al. [Greene93] suggest modifying hardware Z-buffer pipelines to support HV, which rec pass, the standard HV algorithm is run in software, traversing the octree from front to bac skipping nodes which have already been rendered. This second pass fills in any missing p scene. The final step in processing a frame is to update the visible node list. Typically, this the HV algorithm runs considerably faster than the all-software version, because nearly a polygons are rendered with Z-buffer hardware.

Greene and Kass [Greene94b] have developed an extension to hierarchical Z-buffering w antialiased scenes with error bounds. Another interesting algorithm for occlusion culling i skeleton developed by Durand et al. [Durand97,Durand97b].

The HOM Algorithm

The *hierarchical occlusion map* (HOM) algorithm [Zhang97] is another way of enabling hi image-space culling (such as the hierarchical Z-buffering algorithm). However, the HOM can be used on systems that have graphics hardware but not a hardware Z-pyramid, and handle dynamic scenes. The HOM algorithm is described in detail in Zhang's Ph.D. thesis [

We start by describing how the function isOccluded works. This function, used in the pseu Figure 2, is a key part of the algorithm. This occlusion test takes place after the view trans viewer is located at the origin looking down the negative z-axis, with the x-axis going to th the y-axis going upwards. The test is then divided into two parts: a one-dimensional *dept* z-direction and a two-dimensional *overlap test* in the xy plane, i.e., whereby the image ge The overlap test supports approximate visibility culling, where objects that "shine through in the occluders can be culled away using an opacity threshold parameter.

For both tests, a set of potentially good occluders is identified before the scene is rendere occlusion representation is built from these. This step is followed by the rendering of the s the occluders are rendered without an occlusion test. Then the rest of the scene is proces having each object tested against the occlusion representation. If the object occluded by representation, it is not rendered.

For the two-dimensional overlap test, the occluders are first rendered into the color buffe color on a black background. Therefore, texturing, lighting, and Z-buffering can be turned advantage of this operation is that a number of small occluders can be combined into a occluder. The
rendered image, which is called an *occlusion map*, is read back into the main memory of computer. For simplicity, we assume that this image has the resolution of *2^n* x *2^n* pixels the base for the occlusion representation. Then a *hierarchy of occlusion maps* (HOM), i.e., pyramid of occlusion maps, is created by averaging over *2^n-1* x *2^n-1* pixel blocks to form of *2^n-1* x *2^n-1* pixels. This is done recursively until a minimum size is reached (for examp The highest-resolution level of the HOM is numbered 0, with increasing numbers having d resolution. The gray-scale values in the HOM are said to be the *opacity* of the pixels. A hig value (near white) for a pixel at a level above 0 means that most of the pixels it represent covered by the HOM.

The creation of the HOM can be implemented either on the CPU or by texture mapping, wi interpolation used as a minification filter. For large image sizes, the texture filtering appro found to be faster, and for small image sizes, the CPU was faster. Of course, this varies wit graphics hardware. For a 1024x1024 image, Zhang et al. [Zhang97] used a 256x256 image for the HOM. An example of a HOM is shown in Figure 7.

black background in 256x256 pixels, an image which is called occlusion map 0. This image is subsa
128x128 pixels by averaging over 2x2 pixels. This is done recursively down to 8 x 8 pixels. (Model is re
of Nya Perspektiv Design AB.)**

The overlap test against the HOM starts by projecting the bounding volume of the object t onto the screen (Zhang et al. [Zhang97] used oriented bounding boxes). This projection is bounded by a rectangle, which then covers more pixels than the object enclosed in the b volume. So this test is a *conservative test*, meaning that even if the test results show that not occluded, it may still be so. This rectangle is then compared against the HOM for over overlap test starts at the level in which the size of the pixel in the HOM is approximately th rectangle. If all pixels in the rectangle are opaque (which means fully white for non-appro culling), then the rectangle is occluded in the xy plane and the object is said to pass the t other hand, if a pixel is not opaque, then the test for that pixel continues recursively to the the HOM which are covered by the rectangle, meaning that the resolution of the occlusion increases with each test.

For approximate visibility culling, the pixels in the HOM are not compared to full opacity, i.e rather against an opacity threshold value, a gray-scale value. The lower the threshold val approximate the culling. The advantage here is that if a pixel is not fully opaque (white) b than the threshold, then the overlap test can terminate earlier. The penalty is that some o omitted from rendering even though it is (partially) visible. The opacity values are not con one level to another in the HOM, as shown in the following example.

Example: *Computation of opacity threshold values*

Assume the rendered image is 1024x1024 pixels and that the lowest level in the HOM (i.e., t the largest resolution) has a resolution of 128x128 pixels. A pixel in this level-zero occlusion corresponds to an 8x8-pixel region in the rendered image. Also assume that a 2x2 region pixels in an 8x8 region can pass as a negligible hole. This would give an opacity value *O*=1 2^2/8^8=0.9375. The next level in the HOM would then have a 64x64 resolution, and a pixe would correspond to 16x16 pixels in the rendered image. So the opacity threshold at this le *O*=1-2^2/16^2 which is approximately 0.984.

We will now derive a recursive formula for computing the opacity values of the different le HOM. The opacity of the level with the highest resolution in the HOM is O0=1-n/m, where n the number of black pixels that can be considered a negligible hole, and m is the number the rendered image represented by one pixel in this occlusion map (m=8x8 in the examp The next level in the HOM has a threshold of *O*1=1-n/(4m)=1-(1-*O*0)/4=(3+*O*0)/4. This reas generalized to the formula in below for the kth level in the HOM.

Ok+1=(3+*Ok*)/4

For more details on this topic, consult Zhang's Ph.D. thesis [Zhang98].

For the one-dimensional z-depth test, we must be able to determine whether an object is selected occluders. Zhang [Zhang98] describes a number of methods, and we choose to *depth estimation buffer*, which provides reasonable estimation and does not require a Z- implemented as a software Z-buffer that divides the screen into a number of rectangular are rather large in relation to the pixel size. The selected occluders are inserted into this b each region the farthest z-value is stored. This is in contrast to a normal Z-buffer, which st

Figure 8. Illustration of the depth estimation buffer. Illustration after Zhang [Zhang98].

The depth estimation buffer is built for each frame. During rendering, to test whether an o the depth test (i.e., whether it is behind the occluders) the z-value of the nearest vertex of box is computed. This value is compared against the z-values of all regions in the depth e buffer that the bounding box rectangle covers in screen space. If the near value of the bo larger than the stored z-depth in all regions, then the object passes the depth test, and is occluded in the depth direction. A resolution of 64x64 regions in the depth estimation buf by Zhang et al. [Zhang97].

For an object to be occluded, it must thus first pass the overlap test; i.e., the rectangle of t bounding volume of the object must pass the HOM test. Then it must pass the depth test, be behind the occluders. If an object passes both tests, the object is occluded and is not r

Before the algorithm starts, an occluder database is built, where a few criteria are used to certain objects [Zhang98]. First, small objects do not work well in the occluder database, s usually cover a small portion of the image unless the viewer is very close to them. Second a high polygon count are also avoided, as these may negatively affect the performance o rendering of the occlusion map. Third, objects with large or ill-shaped bounding boxes (e. bounding box for a skinny polygon) should be avoided, as they may cause the depth esti to be too conservative. Finally, redundant objects are avoided: for example, a clock on a w contribute as much to occlusion as the wall itself.

At runtime, occluders are selected from the database. To avoid allowing the creation of th become a bottleneck, there is a limit to the number of occluders that can be selected. Th selected with respect to their distance from the viewer and to their size. Only objects insid frustum are selected. A case when this does not work well is shown in Figure 9.

Get daily news, dev blogs, and stories from Game Developer straight to your inbox

Subscribe

LATEST JOBS

[TECHNICAL LIGHTING ARTIST – TURN 10 STUDIOS]

[SR CHARACTER ARTIST – GEARS OF WAR – THE COALITION]

[SOCIAL MEDIA / COMMUNITY MANAGER]

MORE JOBS

CONNECT WITH US

Explore the

Game Developer Job Board

Browse open positions across the game industry or recruit new talent for your studio

BROWSE

Subscribe to

Game Developer Newsletter

Get daily Game Developer top stories every morning straight into your inbox

SUBSCRIBE

Follo

@ga

Follo
to-d
info

Discover More From Informa Tech

Game Developers Conference

Game Developer Jobs

GDC Vault

Game Career Guide

Independent Games Festival

Game Developers Choice Awards

Omdia

Working With Us

Contact Us

About Us

Advertise

Fo

registered office is 5 Howick Place, London, SW1P 1WG.

Get daily news, dev blogs, and stories from Game Developer straight to your inbox

Subscribe

LATEST JOBS

Microsoft

REDMOND, WA, USA

1.04.23

Microsoft

HYBRID (VANCOUVER, BC, CANADA)

1.04.23

Fast Travel Games

HYBRID (STOCKHOLM, SWEDEN)

1.09.23

Nig

LOS A

1.09.2

TECHNICAL LIGHTING ARTIST – TURN 10 STUDIOS

SR CHARACTER ARTIST – GEARS OF WAR – THE COALITION

SOCIAL MEDIA / COMMUNITY MANAGER

LEVE SCR STU

MORE JOBS

CONNECT WITH US

Explore the

Subscribe to

Follow us

Discover More From Informa Tech

Game Developers Conference

Game Developer Jobs

GDC Vault

Game Career Guide

Independent Games Festival

Game Developers Choice Awards

Omdia

Working With Us

Contact Us

About Us

Advertise

Follow Game D

Home Cool

Copyright © 2023 Informa PLC Informa UK Limited is a company registered in England and Wal
1072954 whose registered office is 5 Howick