Real-Time Occlusion Culling with a Lazy Occlusion Grid

Heinrich Hey[1] Robert F. Tobler[2] Werner Purgathofer[1]
[1]Vienna University of Technology [2]VRVis
{hey,rft,wp}@cg.tuwien.ac.at

Abstract. We present a new conservative image-space occlusion culling method to increase the rendering speed of very large general scenes on today's available hardware without time-expensive preprocessing. The method is based on a low-resolution grid upon a conventional z-buffer. The occlusion information in the grid is updated in a lazy manner. In comparison to related methods this significantly reduces the number of pixels that have to be read from the z-buffer. The grid allows fast decisions if an object is occluded or potentially visible. It is used together with a bounding volume hierarchy that is traversed in a front-to-back order and which allows to cull large parts of the scene at once. A special front-to-back traversal is used if no pixel-level query for the furthest z-value of an image area is available. We show that the method works efficiently on today's available hardware and we compare it with related methods.

1 Introduction

Complex scenes may consist of millions of polygons, much more than available graphics hardware can render at interactive frame-rates if only hierarchical view frustum culling and back face culling [11] is used. Occlusion culling methods try to determine which parts of the scene are occluded so that they do not have to be drawn.

In this paper we present a new conservative image-space occlusion culling method for general scenes. It does its occlusion calculations on the fly during rendering. It does not require time-expensive visibility-preprocessing which makes it suited for applications that need to display the scene instantly after the user has modified it, eg. for interactive changes in animations or virtual environments, or for scenes where dynamic objects are important occluders.

The image is subdivided into a low-resolution grid of tiles. Each tile stores occlusion information that shows if the tile is occluded by already drawn objects, and a flag that shows if this occlusion information is outdated. A tile's outdated-flag is set when an object is drawn and the projection of the object's bounding volume (BVol) onto the image plane intersects the tile's area. The tile's occlusion information is not immediately updated after the object has been drawn.

Occlusion of an object is determined by testing if its BVol is occluded in all tiles that intersect the BVol's projection on the image plane. The BVol's occlusion test returns that it is potentially visible if the first tile is found where the BVol is not occluded. First those up-to-date tiles are tested where the occlusion can be determined fast by means of the tiles' occlusion information. This avoids that the z-values of all the pixels of these tiles have to be tested. Only if the BVol is occluded in all these tiles then the occlusion information of outdated tiles has to be updated so that the BVol can be tested against them. The update of a tile's occlusion information is done with a pixel-level query that tests the z-values of the tile's pixels.

This image-space occlusion test is used on a bounding volume hierarchy that is traversed in a front-to-back order. If a bouding volume is rated as occluded then it is culled without having to process its sub-objects and sub-BVols. This way a large part

of the scene can be culled at once. The occlusion test can be done in two different versions:

- The *occlusion state-version* is for systems that provide a pixel-level query which returns if all pixels in a given image area are occluded (their z-values are less than z_{max}, z_{max} corresponds to an unoccluded pixel). This query is already available on some of today's hardware [8,12]. The cost of this query compared to the cost of drawing primitives varies between different hardware [13]. On other systems the query can be implemented in software if the hardware provides fast reading access to the z-buffer. Each tile's occlusion information consists of an occlusion state that can be *free* (completely unoccluded), *partially occluded*, or *full* (completely occluded), and a flag that shows if the state is outdated. A special front-to-back traversal of the BVols is used so that objects are occlusion-tested and drawn in an order that guarantees correct occlusion. If an arbitrary front-to-back traversal would be used then BVols could be falsely occluded by already drawn objects behind them. Note that no primitive-wise front-to-back sorting is needed because the exact visibility of the primitives is solved by drawing them with the z-buffer hardware.
- The *z_{far}-version* is for systems that provide a pixel-level query which returns the furthest z-value of all pixels in the z-buffer in a given image area [3]. On systems where this query is not available in hardware (unfortunately this is commonly the case today) it can be implemented in software if the hardware provides fast reading access to the z-buffer. Each tile's occlusion information consists of the furthest z-value (z_{far}) of its pixels rendered so far, and a flag that shows if z_{far} is outdated. Any appoximative front-to-back traversal [5] can be used because the z_{far}-version allows to test the occlusion of a BVol after objects behind it have been drawn.

We have chosen a flat grid instead of a pyramid [5,6,14] because of the low average number of tiles that have to be tested per BVol, as can be seen in our results. The optimal number of pixels per tile that gives the best overall-performance is system-dependent and can be determined by testing typical scenes of the desired application with different numbers of pixels per tile.

A major feature that distinguishes our method from related methods like the hierarchical z-buffer [5] is that we update the occlusion information of a tile only when it is queried and if it is currently marked as outdated (*lazy update*) instead of updating it every time after an object has been drawn in its image area. Therefore a significantly lower number of pixels must be read from the z-buffer because:

- an object is potentially visible if the first unoccluded tile is found in its image area. Therefore up-to-date tiles are queried first which reduces the chance that outdated tiles have to be queried and updated.
- often several objects draw into a tile's area before the tile is queried and updated.

In section 2 we review existing occlusion culling techniques with special emphasis on image-space methods. Section 3 and 4 describe the occlusion state-version and the z_{far}-version in more detail. In section 5 we describe our implementation and present our results and a comparison with related methods. Section 6 finally presents our conclusions.

2 Previous work

For a static scene occlusion information can be precomputed by subdividing the scene into cells and calculating the potentially visible set (PVS) of each cell [2] which usually requires between several minutes and several hours depending on the scene complexity. The advantage of precomputed PVSs is that the display-phase is usually very fast because the objects in the PVS of the viewpoint's cell can be rendered without any further occlusion culling-overhead. Therefore these methods are often used eg. in games [1] where the frame-rate is the major criterion and the time-expensive precomputation is secondary.

Methods that, like our new method, do their occlusion calculations on the fly during rendering [10] have the advantage that they do not need a time-expensive precomputation, but of course the occlusion calculation during rendering produces some overhead. The hierarchical z-buffer (HZB) [5] is an image-space method that does occlusion culling on the fly with a pyramid of z-values. An octree subdivision is used for hierarchical culling of the scene. An improved version of the HZB [7] reduces the required memory traffic, but currently there is no hardware implementation of the HZB available. Hierarchical coverage masks [6] use a pyramid that contains an occlusion state instead of a z-value. Table-lookups and bit-operations are used instead of traditional scanline-rasterization. Geometry is traversed in exact front-to-back order. Currently available graphics-hardware can only be used for texturing and shading. Hierarchical occlusion maps (HOM) [14] work with a pyramid of occlusion-values which is initially build for a few heuristically chosen occluders. This assumes that they occlude large parts of the scene. HOM support non-conservative culling which speeds up the computation but which does not guarantee that all visible objects are drawn. A simple occlusion test can be done with an available hardware accelerated occlusion query [8,12] that rasterizes a BVol without modifying any buffer and that returns whether any fragment passed the z-test. This query is used eg. in the conservative prioritized-layered projection algorithm [9] to cull occluded cells from the front. Extended hardware accelerated occlusion queries have been suggested that return additional occlusion information [4] and that also work in parallel [3].

3 Occlusion state-version

In the occlusion test of a BVol we classify the tiles that intersect the BVol into two types, as shown in fig. 1:

- *Internal tiles* are those which are completely covered by the BVol.
- *Border tiles* are those which only partially intersect the BVol.

We do this to be able to do a pixel-level query to determine if the part of the BVol that intersects a border tile is occluded if the tile is only partially occluded.

Initially (before any object is drawn or any BVol is tested) the z-buffer is cleared and all tiles are set to free-state. The occlusion test of a BVol works as follows:

- Test if one of the BVol's up-to-date internal tiles is not in full-state. If this is true then the BVol is potentially visible.
- After that, but only if we not already have potential visibility, test if one of the BVol's up-to-date border tiles is in free-state. If this is true then the BVol is potentially visible.

- Next, but only if we not already have potential visibility, test each outdated internal tile of the BVol with a pixel-level query in the tile's whole area whether the tile is full or partially occluded, and set the tile's state. If one of these tiles is partially occluded then the BVol is potentially visible.
- At last, but only if we not already have potential visibility, test each border tile of the BVol which is outdated or in partially occluded-state with a pixel-level query in the intersection area of the tile and the BVol. If the pixel-level query returns that the intersection area is not occluded then the BVol is potentially visible.

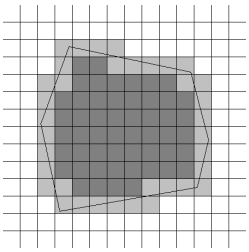

Fig. 1. Border tiles (light grey) and internal tiles (dark grey) of the tested BVol.

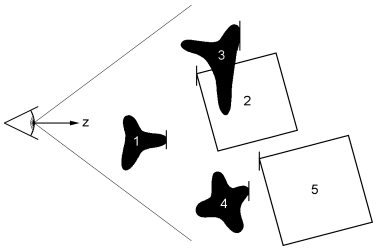

Fig. 2. Front-to-back traversal for the occlusion state-version: z_{near}-sorted BVols (white) are occlusion-tested before z_{far}-sorted potentially visible objects (black) that are not completely in front of them are drawn. z_{near}/z_{far} is marked at each BVol/object.

The approximate front-to-back traversal of the occlusion state-version is illustrated in fig. 2. It uses two lists:

- BVols that are not occlusion-tested yet are sorted by their respective nearest z-value (z_{near}). Initially this *test-list* contains the root BVol of the scene.
- BVols that are already rated as potentially visible and that have objects as direct children are sorted by their respective furthest z-value (z_{far}). Initially this *draw-list* is empty. In fig. 2 the objects are shown instead of the BVols of the draw-list.

If the z_{far} of the frontmost BVol of the draw-list is smaller than the z_{near} of the frontmost BVol of the test-list then the frontmost BVol of the draw-list is removed from the list and its object is drawn. Otherwise the frontmost BVol of the test-list is removed from the list and is occlusion-tested. If it is occluded then it is culled, otherwise its sub-BVols are inserted into the test-list and the BVol is inserted into the draw-list if it has an object as direct child.

4 Z_{far}-version

The z_{far}-version works similar to the occlusion state-version. Initially (before any object is drawn or any BVol is tested) the z-buffer is cleared and all tiles' z_{far} are set to z_{max}. In contrast to the occlusion state-variant the z_{far}-version compares if the nearest z-value (z_{near}) of the BVol is greater than the tile's z_{far} to determine if the BVol is occluded or potentially visible in a tile's area. An outdated tile's z_{far} is updated with a pixel-level query that returns the furthest z-value of all pixels in the tile's area.

5 Implementation and Results

We have implemented and tested occlusion culling with the lazy occlusion grid on a PC with a 900 MHz Thunderbird CPU and a GeForce2 GTS graphics card under

OpenGL. We had no access to graphics hardware that supports pixel-level occlusion queries, therefore we implemented the pixel-level queries in software by reading the hardware z-buffer, which is done with the glReadPixels function. The size of the grid's tiles is 32x32 pixels per tile and has been determined heuristically as described in section 1. Of course on other systems the optimal tile-size may be different. We have measured that our hardware does 34,783 glReadPixels of 32x32 z-values (35,617,792 pixels) per second without flushing the rendering pipeline. For our scenes (fig. 3-5, see appendix) we have used a hierarchy of axis-aligned bounding boxes (BBox), but any other kind of BVol could also be used. The image area of a BBox is approximated by its bounding rectangle in the image. The traversal of the BBoxes hierarchy incorporates hierarchical view frustum culling [11] which is implemented by clipping the BBoxes' polygons in software. The BBoxes hierarchy is initially built for the given set of objects of the scene. In the forest scene each tree is an object with an own bounding box. In the city scene each triangle is a primitive object and the scene is hierarchically subdivided into bounding boxes until each bounding box contains no more than 1500 triangles. This generation of the bounding boxes hierarchy takes less than one second for our scenes. The forest scene contains 1,694,426 triangles and the city scene contains 34,034,176 triangles. We tested each of these scenes with a walkthrough that was rendered with occlusion culling

- with the occlusion state-version of the lazy occlusion grid (LOG).
- with the occlusion state-version of the occlusion grid, but each tile is immediately updated after an object has been drawn into its image area if the tile has not already been marked as full (busy occlusion grid (BOG)).
- with occlusion culling with the HZB [5]. After an object has been drawn the conventional z-buffer in the image area of the BBox is read to update the HZB.
- with occlusion culling solely with a pixel-level query per BBox (PQ) that tests all pixels in the image area of the BBox.

and finally without occlusion culling (no OC), but still with hierarchical view frustum culling. The scenes were rendered at 640x480 as well as 1280x960 pixels to show to what extent image resolution affects the rendering time and the number of pixels that are read from the z-buffer. The average rendering time per frame, the average number of drawn triangles per frame (this means that they are sent to OpenGL, backface culling is done by OpenGL), and the average number of pixels that are read from the z-buffer per frame are shown in table 1. We have measured that with our hardware 38-52% of the total rendering time is spent for the glReadPixels calls when we use the LOG and 54-69% when we use the BOG. The average number of tiles that are tested per BBox with the LOG (including those tiles where no pixel-level query is done) is 14.6 in the forest scene and 17.7 in the city scene at 640x480. The average frame-rate with the LOG is 2.1 to 6.9 times faster than with the BOG, which shows the importance of the lazy update. In the moderately large forest scene rendering without occlusion culling is even faster than most of the occlusion culling methods.

6 Conclusion

We have shown that the lazy occlusion grid considerably reduces the number of pixels that have to be read on today's available z-buffer hardware to determine occlusion, and that this significantly increases performance. Future work includes utilization of temporal coherence and support of parallel pixel-level occlusion queries.

Table 1. Average time, no. drawn triangles and no. read pixels per frame of walkthrough.

forest 640x480	LOG	BOG	HZB	PQ	no OC
time [s]	0.018	0.073	0.162	0.152	0.045
no. drawn triangles	7,009	7,009	7,009	7,009	147,969
no. read pixels	198,621	2,288,217	2,682,035	5,880,310	-

forest 1280x960	LOG	BOG	HZB	PQ	no OC
time [s]	0.031	0.216	0.632	0.611	0.052
no. drawn triangles	7,510	7,510	7,510	7,510	147,969
no. read pixels	424,713	6,844,437	10,729,627	23,495,623	-

city 640x480	LOG	BOG	HZB	PQ	no OC
time [s]	0.021	0.045	0.087	0.158	0.741
no. drawn triangles	11,981	11,981	11,981	11,981	1,964,918
no. read pixels	283,306	1,272,965	1,361,715	6,032,486	-

city 1280x960	LOG	BOG	HZB	PQ	no OC
time [s]	0.033	0.138	0.328	0.630	0.742
no. drawn triangles	11,775	11,775	11,775	11,775	1,964,918
no. read pixels	312,053	4,230,168	5,417,068	24,042,029	-

Acknowledgments

This work has been supported by the Austrian Science Fund (FWF) project P13600-INF. Thanks to M. Wimmer and P. Wonka for the original version of the city model.

References

1. M. Abrash. Inside Quake: Visible Surface Determination. *Dr. Dobb's Sourcebook* January/February 1996 pp. 41-45
2. J. Airey, J. Rohlf, F. Brooks Jr. Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments. *Symposium on Interactive 3D Graphics 90* p. 41
3. D. Bartz, M. Meißner, T. Hüttner. Extending Graphics Hardware For Occlusion Queries In OpenGL. *EUROGRAPHICS/SIGGRAPH workshop on graphics hardware 98* pp. 97-103
4. D. Bartz, M. Meißner, T. Hüttner. OpenGL-assisted Occlusion Culling for Large Polygonal Models. *Computers & Graphics* 23 (1999) pp. 667-679
5. N. Greene, M. Kass, G. Miller. Hierarchical Z-Buffer Visibility. *SIGGRAPH 93* p. 231
6. N. Greene. Hierarchical Polygon Tiling with Coverage Masks. *SIGGRAPH 96* pp. 65-74
7. N. Greene. Occlusion Culling with Optimized Hierarchical Buffering. *SIGGRAPH 99 Sketches & Applications* p. 261
8. Hewlett-Packard. OpenGL Implementation Guide. www.hp.com/workstations/support/documentation/manuals/user_guides/graphics/opengl/ImpGuide/01_Overview.html#OcclusionExtension, 2000
9. J. T. Klosowski, C. T. Silva. Efficient Conservative Visibility Culling Using The Prioritized-Layered Projection Algorithm. *SIGGRAPH 2000* Course Notes 4
10. D. Luebke, C. Georges. Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets. *Symposium on Interactive 3D Graphics 95* pp. 105-106
11. T. Möller, E. Haines. *Real-Time Rendering* pp. 192-200, 1999
12. N. Scott, D. Olsen, E. Gannett. An Overview of the VISUALIZE fx Graphics Accelerator Hardware. *Hewlett-Packard Journal* May 1998 pp. 28-34
13. K. Severson. *VISUALIZE Workstation Graphics for Windows NT*. Hewlett-Packard product literature, 1999
14. H. Zhang, D. Manocha, T. Hudson, K. Hoff III. Visibility Culling using Hierarchical Occlusion Maps. *SIGGRAPH 97* pp. 77-88

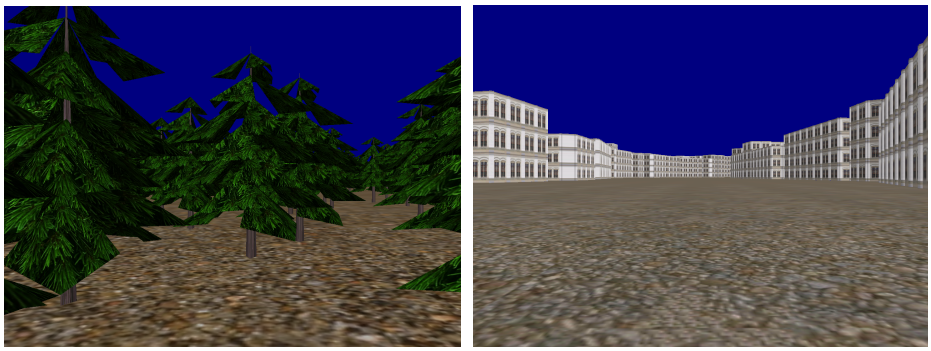

Fig. 3. Actually rendered image.

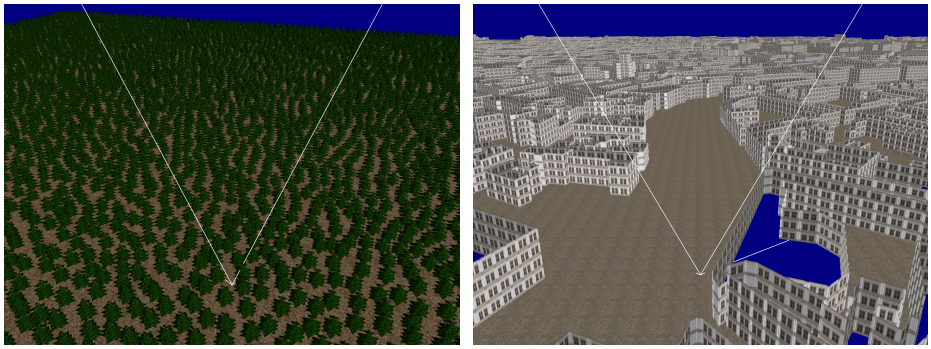

Fig. 4. View from above. The view frustum of figure 3 is visualized as wireframe.

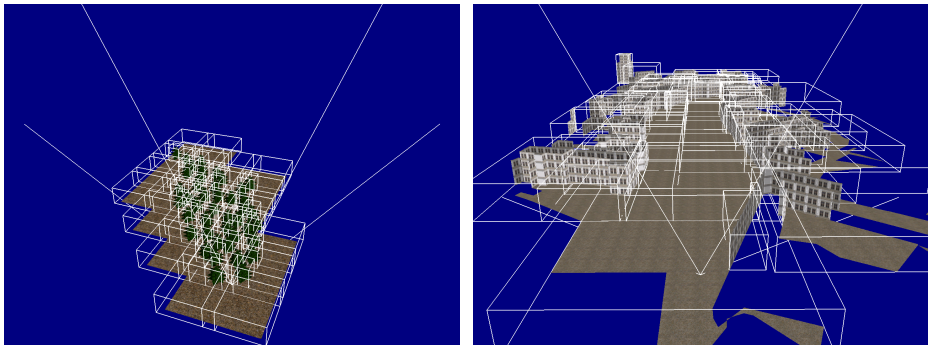

Fig. 5. Only the objects that pass the occlusion test and that are therefore drawn (sent to OpenGL) in figure 3, their leaf-bounding boxes and the view frustum.