

Improving Occlusion Query Efficiency with Occupancy Maps

Dirk Staneker

Dirk Bartz

Michael Meißner*

WSI/GRIS

University of Tübingen, Germany

ABSTRACT

Image space occlusion culling is an useful approach to reduce the rendering load of large polygonal models. Like most large model techniques, it trades overhead costs with the rendering costs of the possibly occluded geometry. Meanwhile, modern graphics hardware supports occlusion culling, whereas they associate a significant query overhead, which hurts in particular, if the occlusion culling query itself was unsuccessful.

In this paper, we propose the Occupancy Map - a compact, cache-optimized representation of coverage information - to reduce the number of costly but unsuccessful occlusion culling queries and to arrange multiple occlusion queries. The information of the Occupancy Map is used to skip an occlusion query, if the respective map area is not yet set - the respective area has not yet received rendered pixels -, hence an occlusion query would always return not occluded.

The remaining occlusion information is efficiently determined by asynchronous multiple occlusion queries with hardware-supported query functionality. To avoid redundant results, we arrange these multiple occlusion queries according to the information of several Occupancy Maps. Our presented technique is conservative and benefits from a partial depth order of the geometry.

1 INTRODUCTION

The datasets for visualization are growing faster in size than the rendering speed of modern graphics subsystems. Several techniques exist to solve this problem. Most of them reduce the number of polygons, others use sampling techniques like ray tracing [15] or point sampling [6]. To reduce the number of polygons, level-of-detail [7] or impostor techniques are also used. Another approach is occlusion culling, which is in the focus of this paper, where hidden parts of a scene are detected and excluded from the rendering process.

A serious drawback of occlusion culling is that it does not provide good performance improvements for all polygonal models. In particular models with low occlusion expose the overhead of occlusion queries which can result in a slow-down, if that overhead exceeds the benefits of not rendered geometry. Another problem are the setup costs for the occlusion queries, because of many state changes like disabling frame and depth buffer writes.

In this paper, we introduce the *Occupancy Map* which significantly reduces occlusion overhead, thus making the performance of occlusion culling approaches less sensitive to the varying depth complexities of the various models and reduces the number of state changes with multiple occlusion queries. In early experiments [3] we ascertained that it is difficult to benefit from these queries because of false-positive results. With the *Occupancy Map*, we address these difficulties and save state changes with the efficient use of multiple occlusion queries.

1.1 Related work

Cohen-Or et al. give a recent overview on the various occlusion culling techniques [4]. While they can be classified in object space [5], and image space techniques [8, 17, 1], we are focusing on image space techniques.

In particular image space techniques frequently use lower resolution framebuffer representation to trace contributions, like a z-pyramid [8], occlusion maps [17], or a virtual occlusion buffer [1]. Meißner et al. [11] proposed a visibility mask within the rasterization stage of the graphics hardware to save internal rasterization bandwidth. In contrast to those hardware approaches, our software technique aims at the reduction of unnecessary occlusion queries and helps to arrange multiple queries to avoid redundant queries and to save state changes.

Obviously, one of the fastest ways to utilize an occlusion query for geometry culling for general scenes is by hardware support [2]. Several solutions are available, some of them use core OpenGL functionality [1, 16], others are using the histogram extension [10] to perform occlusion queries. Specific occlusion culling support is available as well, e.g. with the Hewlett-Packard VISUALIZE fx [13] and the nVIDIA Geforce3 and Geforce4 Ti [12]. Both series of graphics subsystems support the OpenGL extension from Hewlett-Packard, `GL_HP_occlusion_test` and special extensions for multiple queries and measuring the amount of visible pixels. In the future, all DirectX 9.0 or OpenGL 2.0 capable graphics hardware should support these features.

This paper is organized as follows; after describing our occlusion culling approach in Section 2, we introduce the *Occupancy Map* in Section 3. Section 4 describes the implementation and is followed by our result section (Section 5.) Finally, the paper concludes in Section 6.

2 OPENGL AND OCCLUSION CULLING

Our culling approach is based on view-frustum and occlusion culling of the nodes of the scene graph, which contains the hierarchically organized polygonal scene. While the inner nodes of the scene only contain the bounding volume (we use here axis-aligned bounding boxes, AABBs) of their associated sub-tree, the leaf nodes contain the actual geometry and their corresponding AABBs.

The scene traversal performs an interleaved culling and rendering step. Starting with the root node, it takes the bounding volume of the front-most node and performs a view-frustum culling test. If the node (actually its bounding volume) is determined inside the view-frustum, its child nodes are added to the front-to-back sorted list of current nodes. Otherwise, the whole sub-tree is skipped. Occlusion culling is performed by the AABBs of the geometry nodes in the leaf nodes. For details on the implementation, please refer to Section 4.

Obviously, most of the front-most geometry will always be visible (if located in the view-frustum). Hence, the respective occlusion queries will (almost) always return a negative (not occluded) result.

* {staneker,bartz,meissner}@gris.uni-tuebingen.de

Unfortunately, the occlusion query itself is not for free; software and hardware approaches associate significant latency and setup costs with it. Estimates from various applications show that 5 to 10% of the geometry are always not occluded, since they are in front of almost every other geometry [1]. If the scene has only low occlusion but high polygonal complexity (see Fig. 3), the costs of the unsuccessful queries can have a significant penalty. In our novel, software-based *Occupancy Map* approach, we address this problem. It helps to reduce the dependency of the efficiency of occlusion culling from specific datasets; datasets with high occlusion will perform well as with "standard" occlusion culling, while the latency and setup costs of unsuccessful queries in datasets with low occlusion will be saved by the *Occupancy Map*. Furthermore we are addressing the setup costs by multiple *Occupancy Maps*. They are used to manage multiple occlusion queries.

For occlusion culling, we use the above mentioned hardware supported technique which is originally based on the HP Occlusion Flag. HP extended this technique for multiple queries, which was adopted by nVIDIA in a similar way with the nVIDIA Occlusion Query. It is driven through an OpenGL extension, which provides a special occlusion mode, similar to the selection mode of OpenGL. For an occlusion test, the test geometry (in our case an AABB) is rendered in the occlusion mode of the extension with disabled color- and z-buffer writes to avoid actual modifications to the framebuffers. If the test geometry is not occluded (at least one pixel of it triggered a z-buffer write), the actual model geometry associated with the bounding volume is rendered.

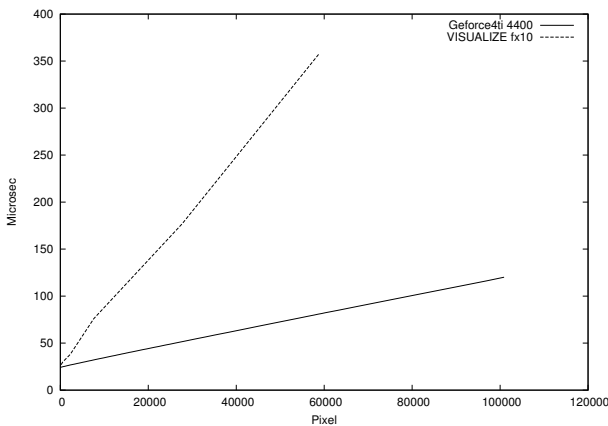


Figure 1: Latency for the occlusion query on an Intel P4@2400 MHz with a nVIDIA Geforce4 Ti 4400 and on an Intel Pentium III@750 MHz with a HP VISUALIZE fx10 for different sizes of a test volume.

The Occlusion Flag-based tests associates two different costs; first it has to wait for the completion of the pipeline flush (because of the disabling of depth and color buffer writes and other state changes) and second it rasterizes the test geometry. Severson [14] associates the equivalent of approximately 190 triangles of 25 pixels each with one query with the HP Flag. However, we found a strong correlation with the z-buffer bandwidth (number of rasterized pixels of the test geometry) and the latency required for an occlusion query (see Fig. 1). With enabled backface culling the query is almost twice as fast as without, since backface culling only requires rasterization of the front faces.

Note that different graphic subsystems show similar characteristics; we performed the same benchmark on an Intel PIII@750 MHz with a HP VISUALIZE fx10 with the HP Flag and on an Intel P4@2400MHz with a nVIDIA Geforce4 Ti4400 (see Fig. 1).

To reduce the latency of setup costs for an occlusion query, the

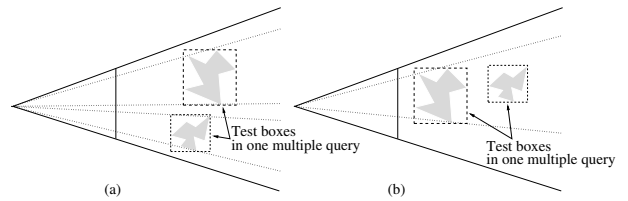


Figure 2: Multiple query without (a) and with (b) possibly redundant results.

HP visibility extension and the nVIDIA extension support multiple tests in one query. The visibility of more than one bounding volume can be determined at the same time by such a multiple query. For each tested bounding volume the visibility is returned. One major problem of multiple queries is the selection of the bounding volumes, because there can be false positive results if two bounding volumes test the same screen space region (see Figure 2.) The second volume, behind the first one, is tested against maybe not up-to-date depth values, because the geometry of the first bounding volume is not yet rendered. The *Occupancy Map* can be used to reduce such redundant tests and helps to arrange the bounding volumes for multiple queries.

3 OCCUPANCY MAP

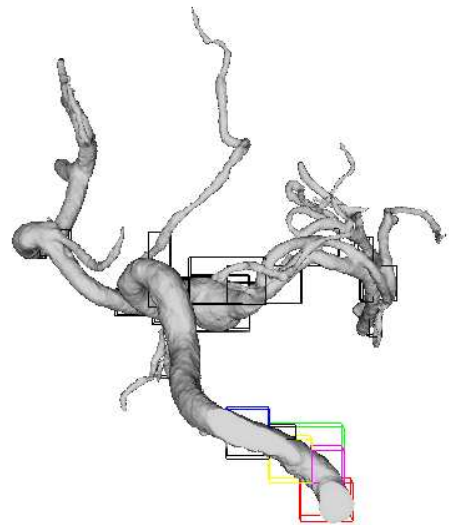


Figure 3: Example scene with low occlusion. Tested AABBs with *Occupancy Map*.

As noted in the previous section, the latency of an occlusion query depends on the number of rasterized pixels of the bounding volume. In particular large, partially visible bounding volumes will spend significant time in the rasterization stage of the graphics accelerator. In order to reduce the associated costs, we try to avoid occlusion queries or at least reduce their latency. For multiple occlusion queries an arrangement is necessary to avoid redundant queries and to be independent of heuristics for special scenes.

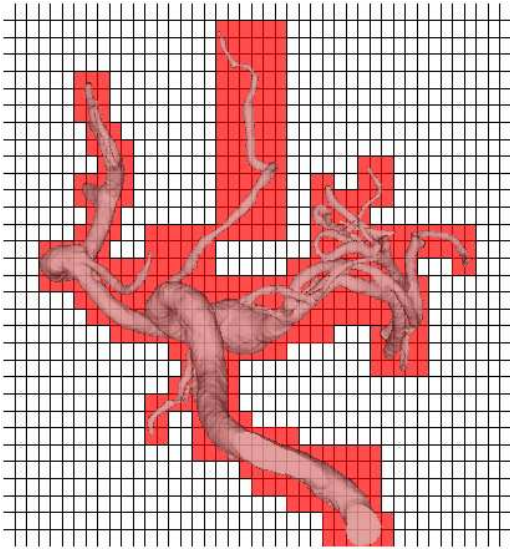


Figure 4: *Occupancy Map* for example scene.

3.1 Visible AABBs in the Front

AABBs in screen space areas, which are not yet covered by geometry are always visible, because there is no occluding geometry. To avoid queries of the respective AABBs, we are using an *Occupancy Map* (OM), which enables a fast reject of occlusion queries in screen areas which are not yet covered by scene pixels. As soon as the *Occupancy Map* detects that the target screen area is “empty”, it cancels the occlusion query and initiates the rendering of the respective scene geometry. Note that the *Occupancy Map* is conservative, since it is essentially storing the conservative lower resolution coverage information of the framebuffer. However, it is not exact and will occasionally initiate the rendering of geometry which would have been determined occluded by the actual query. This is also due to the approximation of the scene entity AABB by a screen space AABB. Nevertheless, we found that with the used *Occupancy Map* size (see below), this was not significant at all.

Technically, an *Occupancy Map* is a cache optimized bit-field realized as an array of 32 Bit integers with 512 entries. Every bit represents a tile in the screen space. If a bit is set, the respective tile is occupied by already rendered geometry. Otherwise, no geometry has yet been rendered into the associated screen region; it is empty (see Fig. 4). The effectiveness of the tile size is a trade-off between precision (resolution) and overhead (memory and update). Meißner et al. [11] provided interesting measurements on the effective resolution in the context of a hardware implementation. In our context, we found that our *Occupancy Map* size of 2048 Bytes is quite effective. The representing tile size can be adapted to the window size; for an example window of $1024 * 768$ pixels, every tile represents $4 * 48$ pixels. If the whole scene is inside the view-frustum, the screen space size of the scene’s AABB can be used to scale the *Occupancy Map* to provide a better resolution. Due to the small size, the *Occupancy Map* can be easily accommodated in the first level cache to permit an extremely low latency (orders of magnitude lower than a read back from the graphics accelerator to acquire the occlusion query result).

For a lookup in the *Occupancy Map*, we transform the screen space AABB of the 3D AABB of the queried scene entity, and check if it overlaps with empty (unset) regions of the *Occupancy Map*. If that is the case (dotted box in Fig. 5), the corresponding AABB is assumed visible and the occlusion query for this node is canceled. Note that for good performance, the rendered scene en-

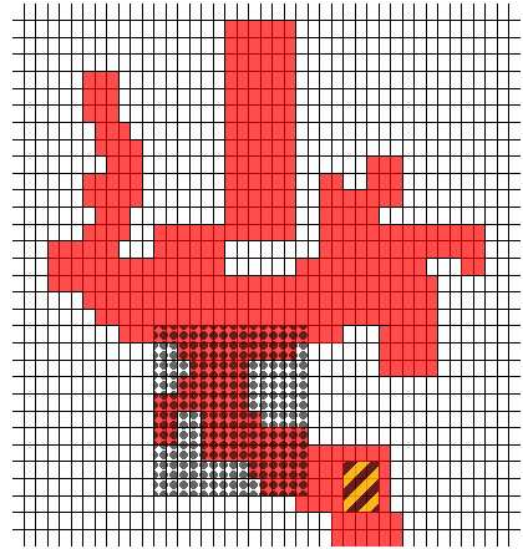


Figure 5: Request to the *Occupancy Map*; the dotted box is detected as visible, the hatched one as possibly occluded. For the latter one, an occlusion query will follow.

tities should be partially organized front-to-back, although it is not really mandatory.

3.2 Organization of Multiple Occlusion Queries

To further reduce the latency of an occlusion query, we are using multiple occlusion queries. This method reduces setup costs, because state changes are solely necessary before and after the multiple query. However, redundant queries (see Figure 2) have to be avoided in order not to get false-positive results of geometry that is indeed occluded. To reduce this problem, we are using multiple *Occupancy Maps*. Each *Occupancy Map* arranges one multiple query. Figure 6 shows the architecture of our approach.

The first *Occupancy Map* in our architecture works like described in the previous section. If the AABB completely overlaps with full (set) regions, the 3D AABB is potentially occluded and has to be tested with an occlusion query (OM test 0).

The *Occupancy Maps* organizing multiple queries have a slightly different meaning compared with the *Occupancy Map* from the previous section. A set bit in such an *Occupancy Map* means that this region is covered by an AABB from the corresponding test list. An AABB is added to a multiple occlusion query, if at least one bit in the corresponding *Occupancy Map* is not set in its respective screen region of the AABB. This means that the AABB will test a region in screen-space, which is not yet covered by another AABB from the respective test list. AABBs can overlap in screen-space, which could result in redundant queries, but they never occlude each other. We have found that this approximation is adequate to save redundant queries and to have many different AABBs in one multiple occlusion query. If all *Occupancy Map* bits covered by the AABB are already set, the AABB is tested with the subsequent *Occupancy Map*. The AABB is always rendered into the tested *Occupancy Map* to mark the corresponding screen space region as used. Multiple occlusion queries are performed after processing all AABBs and their corresponding geometry nodes.

We tested the method on different models and we have found that five multiple occlusion queries are sufficient. All AABBs, which passed the fourth *Occupancy Map* are added automatically to the

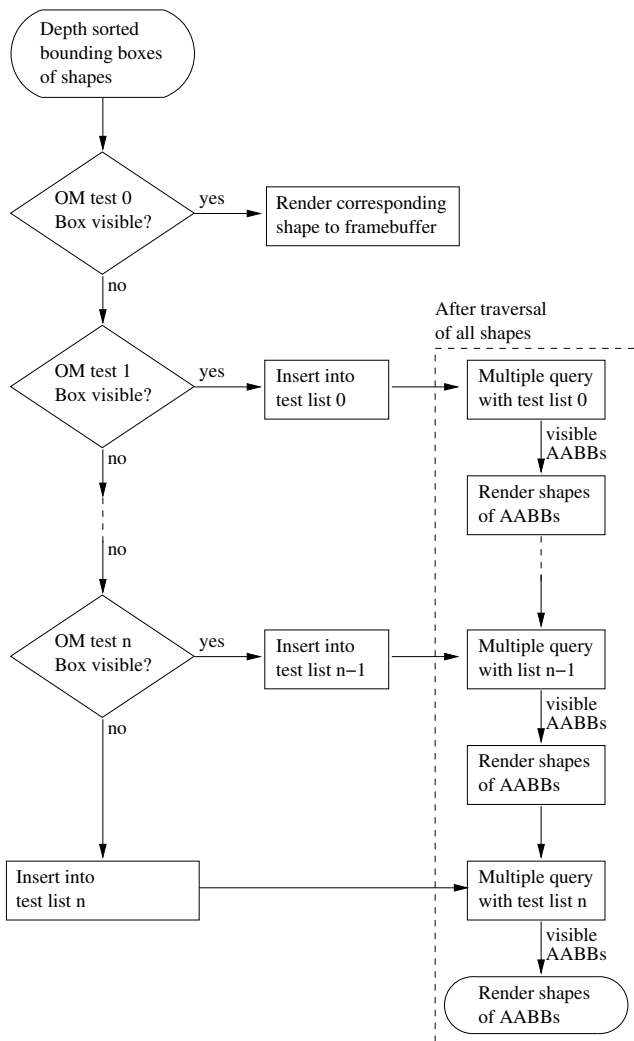


Figure 6: Architecture.

last occlusion query.

4 IMPLEMENTATION

All our measurements are performed in *Jupiter*, a scene graph-based toolkit for the interactive visualization of large polygonal models [9]. *Jupiter* is based originally on a software initiative of Hewlett-Packard Company (HP) and Engineering Animation Inc. (EAI), and on the OpenSource project *Kelvin* in collaboration with the University of Tübingen. In the following, we briefly describe how occlusion culling is integrated in the *Jupiter* toolkit. For more details please refer to Bartz et al. [3].

4.1 *Jupiter* Scene Graph

In *Jupiter*, models are represented as scene graphs which describe a hierarchical organization of the objects of the model. The scene graph consists of a variety of nodes, describing the partition of the model into objects and groups of objects. Important for rendering are the *JtShape* nodes, containing geometry as leafs in the scene graph. Important for occlusion culling are the AABBs of the shape nodes.

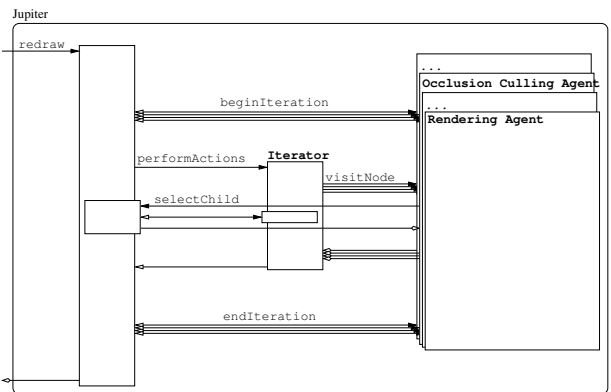


Figure 7: A strategy calls a set of pipelined agents which process the scene graph.

The scene graph traversal in *Jupiter* is managed by a set of pipelined *agents*. These evaluate the importance of the scene graph nodes and manipulate the traversal order accordingly. The agents are called in a pipelined fashion, where the following agent processes the resulting node from the previous agent, if it is not culled (Fig. 7). One agent computes a priority, which decides the further traversal.

4.2 The Occlusion Culling Agent

Rendering and occlusion culling of the scene graph work in an interleaved fashion. In the first step, all shape nodes are evaluated and arranged by their corresponding AABB with the *Occupancy Maps*. If the first *Occupancy Map* results in a visible shape, the shape is rendered by the rendering agent. The remaining shapes are assigned to the associated occlusion query list, managed by the occlusion culling agent. After processing of all the shapes in the scene graph, the visibility of the shapes in the occlusion query lists are tested with multiple occlusion queries. All visible shapes are rendered immediately after the test of their corresponding occlusion culling list.

In our implementation, we used five *Occupancy Maps*. The first one keeps information about the already rendered shapes. The remaining four *Occupancy Maps* are responsible for the assignment to one of the five occlusion culling lists. The fifth occlusion culling list gets all the shape nodes whose are not assigned to one of the first four lists. To avoid redundant occlusion queries, a front-to-back sorted traversal of the scene graph is used. The nearest corner of the node's AABB to a given viewpoint is used for this sorting. For all nodes, we applied view-frustum culling.

5 RESULTS

5.1 *Occupancy Maps* Reduce Unnecessary Visible Tests

For our evaluation of the *Occupancy Map* (OM), we used different polygonal models with viewpoints on camera paths around the model. The frames were rendered at a resolution of 876×584 on an Intel Pentium III@750 MHz running Linux with a HP VISUALIZE fx10. In the first frames, the model is located in the view-frustum and rotates around its axes. Thereafter, the camera zooms to and back from a closer view, while some parts are removed by view-frustum culling. *Angio* is an extracted isosurface of a blood vessel tree from a 3D medical scanner (Fig. 12b). It consists of 419639

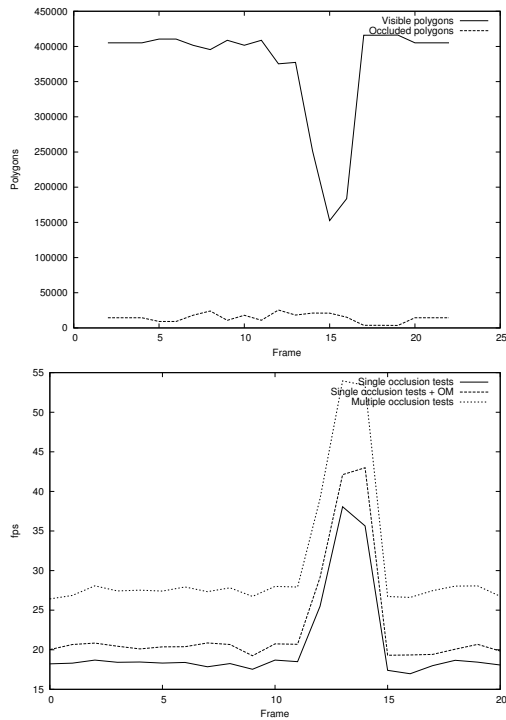


Figure 8: Angio frame rates and polygons.

	Framerates [fps]	
	without OM	with OM
Angio	7.0	7.8
BoomBox	10.8	11.1
FormulaOne	6.3	6.5

Table 1: Resulting average framerates for the test models.

polygons, organized in 177 leaf nodes. It is the smallest dataset of the three models with the lowest occlusion, thus the Angio does not benefit from occlusion culling. Table 1 shows the framerates for occlusion culling with and without an *Occupancy Map*. The Angio dataset is in particular suited to measure the efficiency of the *Occupancy Map* for views with low occlusion; an average of 13% of the rasterization bandwidth for the regular occlusion query is only required for occlusion queries with *Occupancy Map* (see Table 2). Due to these savings, it gained 0.8 fps.

	Occupancy Map [Pixels]		
	without OM	with OM	
Angio	2 953 200	382 450	13%
BoomBox	3 469 702	1 365 960	39%
FormulaOne	1 217 297	544 213	45%

Table 2: Rasterized pixels for not occluded AABBs.

5.2 Multiple Queries Arranged by *Occupancy Maps*

To evaluate the performance of the multiple occlusion queries, organized by the *Occupancy Maps*, we used an Intel P4@2400 MHz

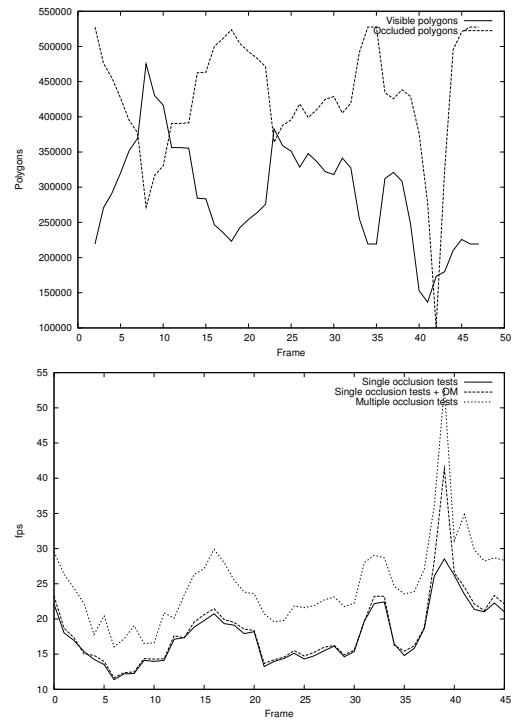


Figure 9: Formula One frame rates and polygons.

	Shapes	Num	Average
		polygons	polygons/shape
Angio	177	419 639	2 370
Boom Box	530	644 268	1 215
Formula One	306	746 827	2 441
City	1900	4 059 195	2 134

Table 3: Geometry of the used models.

with a nVIDIA Geforce4 Ti 4400 and the models listed in Table 3. We rendered a camera path for each model at a resolution of 1000×900 and 32 Bit colors. The occlusion culling was done with the nVIDIA occlusion query extension.

The Boom Box and the Formula One car (see Figure 12a/c) are the MCAD models from the previous test. The City model (see Figure 12d) is an artificial city with some Formula One cars in the streets and has the highest complexity of the four models with high depth complexity.

We rendered different camera paths for each model. In some frames we zoomed into the scene and some parts of the scene were view-frustum culled. Our measurements are performed with single occlusion query for each leaf node in the scene graph (without using an *Occupancy Map*), then we repeated the test but with one *Occupancy Map* to reduce tests of visible AABBs in the front. In the last test, we rendered the camera path with multiple occlusion queries organized by five *Occupancy Maps*.

With occlusion culling we achieved average framerates between 17.2 fps (City) and 28.0 fps (Boom Box) for the different models (see Table 4.) With the *Occupancy Map* we improved these results to 18.5 fps (City) and 28.4 fps (Boom Box). The best results were achieved with multiple occlusion queries, between 24.9 fps (Formula One) and 37.9 fps (City).

Table 5 shows the speed-ups of the *Occupancy Map* over the occlusion culling without an *Occupancy Map* and the speed-ups over

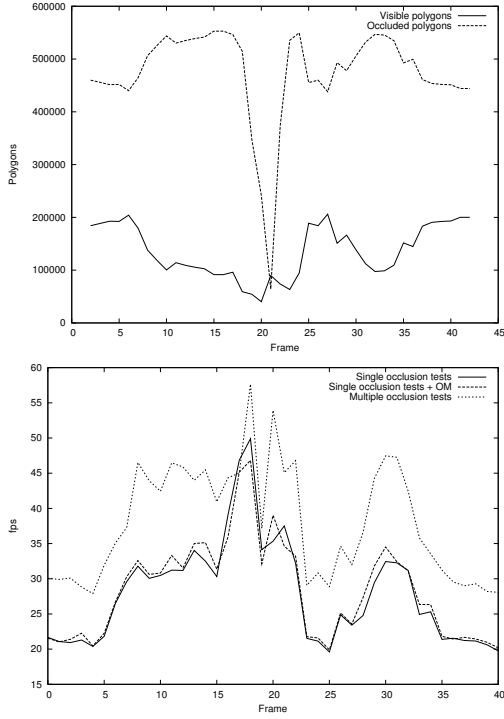


Figure 10: Boom Box frame rates and polygons.

	Framerates [fps]		
	single ocq	single ocq OM	multiple ocq
Angio	20.3	22.8	30.5
BoomBox	28.0	28.4	37.9
FormulaOne	17.7	18.4	24.9
City	17.2	18.5	30.7

Table 4: Resulting average framerates for the test models.

the multiple occlusion queries and the single occlusion query with *Occupancy Map*. In particular scenes with low occlusion (Angio) or large occluders in the front (City) benefit from an *Occupancy Map*. A further speed-up can be achieved with the multiple occlusion queries. In the Figures 8, 9, 10, and 11 an overview of the different frame rates and the amount of visible or occluded polygons for each frame in our camera paths is given.

6 CONCLUSIONS

In this paper, we presented the *Occupancy Map*, an approach to reduce the occlusion culling overhead. It features two qualities; first, unnecessary visible queries are prevented. This is very useful for scenes with low occlusion (see Figure 3). The *Occupancy Map* lookups are very fast to avoid additional overhead. Models with large occluders in the frontal part of the model are almost immediately tested through the regular occlusion culling query. The *Occupancy Map* required only little computation and memory. No complex data structures are needed and it can be easily integrated with other occlusion culling approaches.

Second, the *Occupancy Map* can be used to arrange AABBs for multiple occlusion queries with less computation. Multiple occlusion queries help to reduce state changes and thus to reduce the

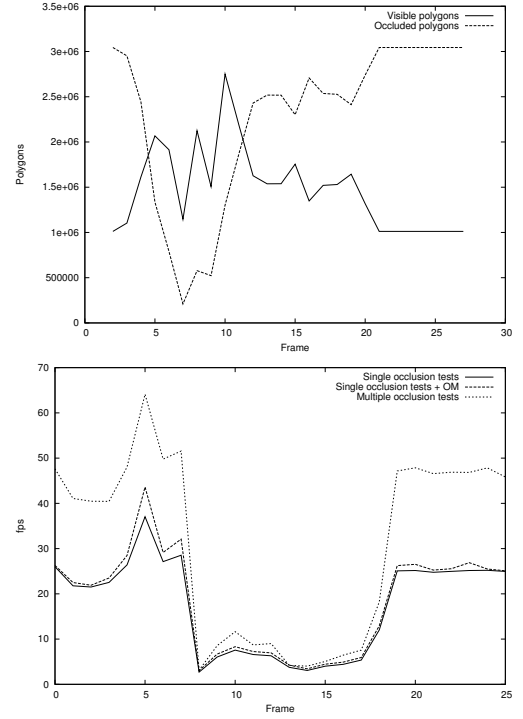


Figure 11: City frame rates and polygons.

	Speed-Up	
	single ocq OM	multiple ocq
Angio	12%	34%
BoomBox	1%	33%
FormulaOne	4%	35%
City	8%	66%

Table 5: Resulting average speed-ups for the test models over the occlusion culling without an *Occupancy Map* and the speed-ups over the multiple occlusion queries and the single occlusion query with *Occupancy Map*.

latency of a query. We achieved a speed-up between 35% and 78% over unsophisticated occlusion culling.

6.1 Future work

The arrangement of the multiple occlusion queries can be easily parallelized. During traversal of the scene graph and the rendering of the shapes in the front (detected by the first *Occupancy Map*), a second thread can perform and arrange the multiple occlusion queries by the other *Occupancy Maps*.

During the experiments, we observed that not occluded AABBs of the models could be represented by their eight corners in many cases. Therefore, we can improve the accuracy of the *Occupancy Map* lookup and the performance of the occlusion query by first testing for the corners of the AABBs (or other bounding volumes), before we proceed with the full AABBs. However, in the case of occluded corners, but not occluded AABBs, the setup time of the overall query has doubled.

Besides the use for occlusion culling, screen space AABB of the *Occupancy Map* lookup gives information of the screen size for

the bounding box. This can be used for level-of-detail selection or screen size culling.

ACKNOWLEDGEMENTS

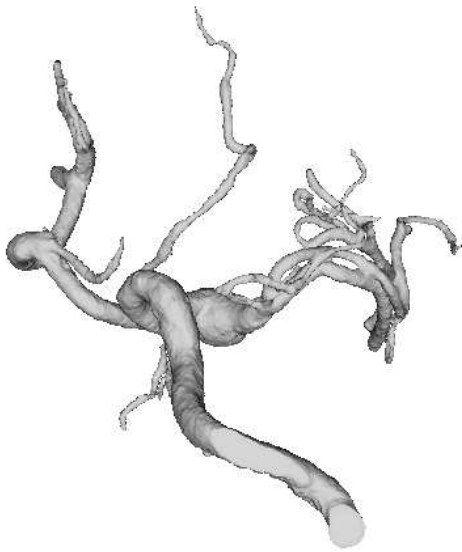
This work is supported by OpenSG PLUS project of the federal ministry of education and research (bmb+f). The MCAD datasets are courtesy of Engineering Animation Inc. and Hewlett-Packard Company. We would like to thank Alexander Ehlert and Anxo del Rió of the University of Tübingen for proof reading.

REFERENCES

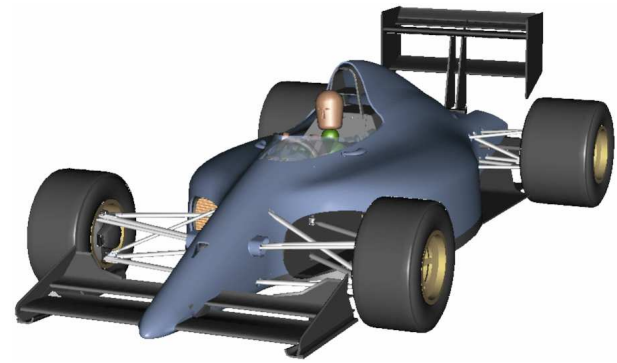
- [1] BARTZ, D., MEISSNER, M., AND HÜTTNER, T. OpenGL-assisted Occlusion Culling of Large Polygonal Models. *Computers & Graphics* 23, 5 (1999), 667–679.
- [2] BARTZ, D., AND SKALEJ, M. VIVENDI - A Virtual Ventricle Endoscopy System for Virtual Medicine. In *Proc. of Symposium on Visualization* (1999), pp. 155–166,324.
- [3] BARTZ, D., STANEKER, D., STRASSER, W., CRIPE, B., GASKINS, T., ORTON, K., CARTER, M., JOHANNSEN, A., AND TROM, J. Jupiter: A Toolkit for Interactive Large Model Visualization. In *Proc. of Symposium on Parallel and Large Data Visualization and Graphics* (2001), pp. 129–134.
- [4] COHEN-OR, D., CHRYSANTHOU, Y., DURAND, F., AND SILVA, C. Visibility: Problems, Techniques, and Application. In *ACM SIGGRAPH Course 4* (2000).
- [5] COORG, S., AND TELLER, S. Temporally Coherent Conservative Visibility. In *Proc. of ACM Symposium on Computational Geometry* (1996), pp. 78–87.
- [6] DEBEVEC, P., BREGLER, C., COHEN, M., SZELISKI, R., MCMILLAN, L., AND SILLION, F. Image-Based Modeling, Rendering, and Lighting. In *ACM SIGGRAPH Course 35* (2000).
- [7] GARLAND, M. Multiresolution Modeling: Survey and Future Opportunities. In *Eurographics STAR report 2* (1999).
- [8] GREENE, N., KASS, M., AND MILLER, G. Hierarchical Z-Buffer Visibility. In *Proc. of ACM SIGGRAPH* (1993), pp. 231–238.
- [9] HEWLETT-PACKARD. Jupiter 1.0 Specification. Tech. rep., Hewlett Packard Company, Corvallis, OR, 1998.
- [10] KLOSOWSKI, J., AND SILVA, C. Efficient Conservative Visibility Culling Using the Prioritized-Layered Projection Algorithm. *IEEE Transactions on Visualization and Computer Graphics* 7, 4 (2001).
- [11] MEISSNER, M., BARTZ, D., GÜNTHER, R., AND STRASSER, W. Visibility Driven Rasterization. *Computer Graphics Forum* 20, 4 (2001), 283–294.
- [12] NVIDIA. NVIDIA OpenGL Extension Specifications, 2002.
- [13] SCOTT, N., OLSEN, D., AND GANNETT, E. An Overview of the VISUALIZE fx Graphics Accelerator Hardware. *The Hewlett-Packard Journal*, May (1998), 28–34.
- [14] SEVERSON, K. VISUALIZE fx Graphics Accelerator Hardware. Tech. rep., Hewlett Packard Company, available from <http://www.hp.com/workstations/support/documentation/whitepapers.html>, 1999.
- [15] SLUSALLEK, P., PARKER, S., REINHARD, E., PFISTER, H., AND PURCELL, T. Interactive Ray-Tracing. In *ACM SIGGRAPH Course 13* (2001).
- [16] STANEKER, D. An Occlusion Culling Toolkit for OpenSG PLUS. <http://www.opensg.org/OpenSGPLUS/symposium/>, 2003.
- [17] ZHANG, H., MANOCHA, D., HUDSON, T., AND HOFF, K. E. Visibility Culling Using Hierarchical Occlusion Maps. In *Proc. of ACM SIGGRAPH* (1997), pp. 77–88.



(a)



(b)



(c)



(d)

Figure 12: Rendered datasets: (a) Boom Box; (b) Angio; (c) FormulaOne; (d) City