Efficient Multiple Occlusion Queries for Scene Graph Systems

Dirk Staneker, Dirk Bartz, Wolfgang Straßer

WSI-2004-6
August 2004

Graphisch-Interaktive Systeme
Wilhelm-Schickard-Institut
Universität Tübingen
D-72076 Tübingen, Germany
e-mail: {staneker,bartz,strasser}@gris.uni-tuebingen.de
WWW: http://www.gris.uni-tuebingen.de

ISSN 0946-3852

Efficient Multiple Occlusion Queries for Scene Graph Systems

Dirk Staneker   Dirk Bartz   Wolfgang Straßer

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. Unfortunately these hardware extensions consume fillrate and latency costs.

In this paper, we propose a new technique for scene graph traversal optimized for efficient use of occlusion queries. Our approach uses several Occupancy Maps to organize the scene graph traversal. During traversal hierarchical occlusion culling, view frustrum culling and rendering is performed.

The 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.

Contents

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 [SPR$^+$01] or point sampling [WFP$^+$01]. To reduce the number of polygons, level-of-detail [Gar99] 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 [SBM03], we presented an approach to reduce the number of unnecessary queries and how multiple queries can be implemented efficently for general scene graph systems. A drawback of this approach is the lack of hierarchical occlusion culling of subtrees in the scene graph.

In this paper, we introduce a new traversal technique, which significantly reduces occlusion overhead, thus making the performance of occlusion culling approaches less sensitive to the varying depth complexities of the various models, reduces the number of state changes with multiple occlusion queries and performs hierarchical occlusion culling of occluded subtrees of the scene graph. Note that the goal of our approach is to achieve optimal performance for a varying set of models and viewing situations, even if little or no occlusion or depth complexity is present, thus regular occlusion culling would produce a slow down. Also we are only using the bounding box hierarchy of the scene graph without precomputing special hierarchies or data structures. Temporal coherence is almost not used to better support dynamic scenes, only the bounding box hierarchy of the scene graph has to be up to date in each frame.

1.1 Related work

Cohen-Or et al. give a recent overview on the various occlusion culling techniques [COCSD03]. While they can be classified in object space [CT96], and image space techniques [GKM93, ZMHH97, BMH99], 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 [GKM93], occlusion maps [ZMHH97], or a virtual occlusion buffer [BMH99]. Meißner et al. [MBGS01] 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, helps to arrange multiple queries to avoid redundant queries, and furthermore helps 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 [BS99]. Several solutions are available, some of them use core OpenGL functionality [BMH99, Sta03], others are using the histogram extension [KS01] to perform occlusion queries. Specific occlusion culling support is available as well, e.g. with the Hewlett-Packard VISUALIZE fx [SOG98] and the nVIDIA Geforce3 and Geforce4 Ti [NVI02]. 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. Multiple occlusion queries are available as ARB extension in newer OpenGL versions, these are used for the presented techniques in this paper.

2 OpenGL and Occlusion Culling

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 a BB) is rendered in the test 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.

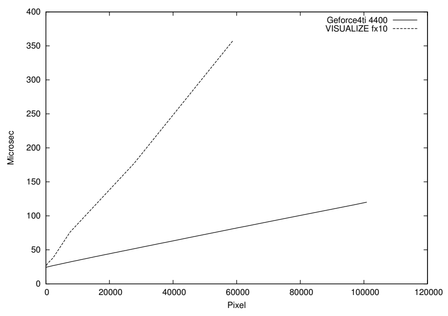


Figure 1: Latency for the occlusion query on an Intel P4@2400 MHz with a nVIDIA Geforce4 Ti 4400 and on an Intel Pentium III@750 MHz with a HP VISUALIZE fx10 for different sizes of a test volume [SBM03].

The Occlusion Query-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 [Sev99] 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 from [SBM03]).

To reduce the latency of setup costs for an occlusion query, the HP visibility extension and the nVIDIA extension support multiple tests in one query. The visibility of

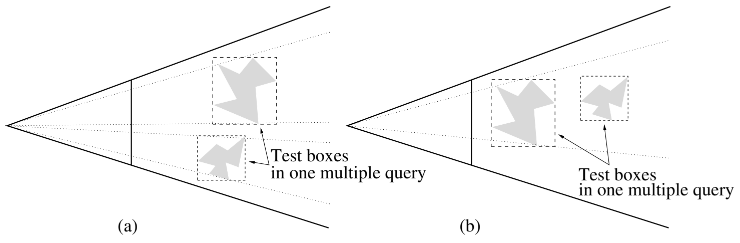


Figure 2: Multiple query without (a) and with (b) possibly redundant results.

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

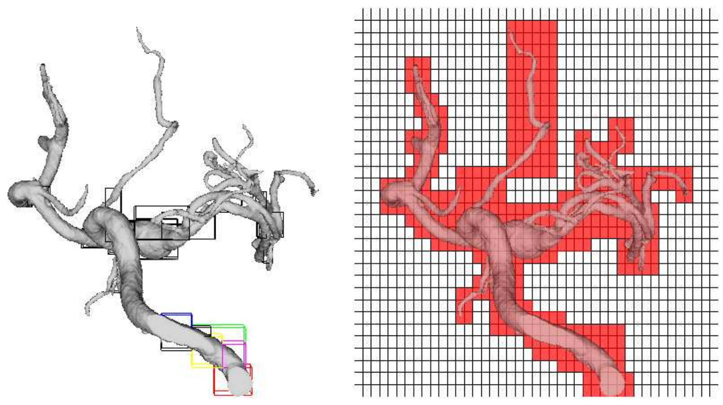


Figure 3: *Left:* Scene with low occlusion. *Right: Occupancy Map* for scene [SBM03].

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. 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 [BMH99]. 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 software-based *Occupancy Map* approach [SBM03], 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 using multiple occlusion queries managed by an *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 at large using the *Occupancy Map* (Section 3.1). If we cannot avoid the occlusion query, we attempt to reduce their latency by using multiple occlusion queries. As noted before, they are difficult to use efficiently, due to redundant false positive queries [BSS$^+$01]. In Section 3.2, we avoid redundant queries by employing multiple *Occupancy Maps*. This will also render the occlusion culling approach mostly independent of special heuristics for particular scenes.

3.1 Visible Bounding Boxes in the Front

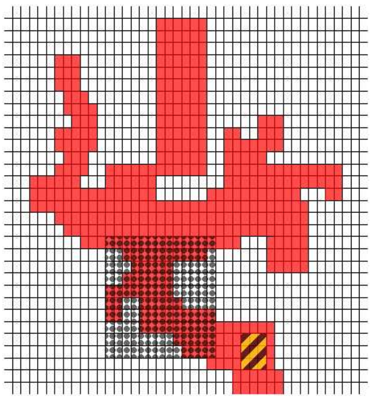


Figure 4: Request to the *Occupancy Map*; the dotted box is detected as visible, the hatched one as possibly occluded. For the latter one, an occlusion query will follow [SBM03].

BBs[1] 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 BBs, 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 BB by a screen space AABB. Nevertheless, we found that with the used *Occupancy Map* size (see below), this was not significant at all. See [SBM03] for a more detailed description and results of the *Occupancy Map* within the *Jupiter* scene graph.

[1]Note that other bounding volumes [BKS01] work as well as standard BBs with the *Occupancy Map*.

For a lookup in the *Occupancy Map*, we transform the screen space BB of the 3D BB 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. 4), the corresponding BB is assumed visible and the occlusion query for this node is canceled. Note that for good performance, the rendered scene entities 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 an *Occupancy Map* for each multiple query. In contrast to [SBM03] we present an approach for *hierarchical* occlusion culling with multiple queries in this paper.

The first *Occupancy Map* in our architecture works like described in [SBM03]. If the screen space AABB completely overlaps with full (set) regions, the 3D BB 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 a BB from the corresponding test list. A BB 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 BBs in one multiple occlusion query. If all *Occupancy Map* bits covered by the AABB are already set, the AABB is tested in a subsequent *Occupancy Map*. The AABB is always rendered into the tested *Occupancy Map* to mark the corresponding screen space region as used.

4 Scene Graph Traversal

Our culling approach is based on view-frustum and occlusion culling of the nodes in 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 bounding boxes, BBs) of their associated sub-tree, the leaf nodes contain the actual geometry and their corresponding BBs.

All our measurements are performed in *OpenSG*, a scene graph-based toolkit for the interactive visualization of large polygonal models [OF00]. OpenSG PLUS has occlusion culling functionality [Sta02, Sta03, SBS04]. In this paper, we describe an enhanced technique for scene graph traversal to benefit efficiently from hardware occlusion queries. In [SBM03] in contrast, we used *Jupiter* [HP98, BSS$^+$01] for our tests and measurements.

4.1 OpenSG Scene Graph

In *OpenSG*, 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 geometry nodes, containing geometry as leafs in the scene graph. Occlusion culling is done with the bounding boxes of the nodes, which are used in the hardware occlusion queries. We implemented a new OpenSG *RenderAction* with our new traversal and culling techniques.

4.2 Traversal Scheme

The scene traversal performs an interleaved culling and rendering step. Starting with the root node, it takes the bounding volume of the 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. To avoid redundant occlusion queries, a front-to-back sorted traversal of the scene graph is used. The closest corner of the node's BB to a given viewpoint is used for this sorting.

In the first stage, we use the *Occupancy Map* to find the visible nodes in front of the scene [SBM02]:

```
add root node to priority queue;

while(priority queue not empty){
  node = get first node from priority queue;

  if(node is outside view frustum){
    cull node;
  }else{

    if(node is visible in occupancy map){
      if(node is geometry){
        render node and assign node to occupancy map;
      }else{
        add children front-to-back to priority queue;
      }
    }else
      add node to pending list;
  }
}
```

Nodes, with a BB that is visible in the *Occupancy Map*, are assumed visible in the scene, since there is no rendered geometry up to now which could cover the region, see [SBM03]. If such a node is a geometry node, it is rendered and the *Occupancy Map* is updated by its screen space BB. Nodes, whose BB is covered in the *Occupancy Map* are added to the pending list. These nodes are probably occluded and therefore tested with occlusion queries in the next stage of the traversal scheme.

In the second stage, multiple occlusion queries are performed on the nodes of the scene graph. To avoid false positive results, an *Occupancy Map* is used to distribute the occlusion tests in different screen space regions:

```
n = 0;
while(pending list not empty && n<max_tests){
  add nodes from pending list to priority queue;
  clear occupancy map;

  while(priority queue not empty){
    node = get first node from priority queue;
    if(node is outside view frustum){
      cull node;
    }else{
      if(node is visible in occupancy map){
        add node to test list;
        assign node to occupancy map;
      }else
        add node to pending list;
    }
  }

  perform multiple occlusion query with test list;

  for(each visible node){
    if(node is geometry)
      render node;
    else
      add children to pending list;
  }

  n++;
}
```

In contrast to [SBM03], we use also the BBs of the inner nodes of the scene graph and not only the BBs of the geometry nodes of the leafs. This leads to a hierarchical approach, since complete subgraphs can be culled, if the BB of the subgraph is occluded. The main problem is that we need a more enhanced organisation of the traversal and culling approach. Children of visible inner nodes have to be traversed further and tested for occlusion. Two priority queues (test list and pending list) are used to organize the traversal and multiple occlusion queries. The priority queues are working in a *double-buffered* manner. In [SBM03], a fixed number of *Occupancy Maps* is used, while we are using only one *Occupancy Map* in this approach, which is cleared after each multiple occlusion query. To reduce the maximum number of occlusion queries given by the *Occupancy Map*, we only use n times the *Occupancy Map*. After n we perform multiple occlusion queries for all remaining nodes. n is calculated from frame to frame, so there is some limited use of temporal coherence: $n_{frame+1} = 7/10 \cdot m_{frame}$, where m is the number of all (organized by *Occupancy Map*+ remaining queries) multiple occlusion queries. Note that in case of drastic movement, which invalids temporal coherence, the method is still conservative, due to the occlusion queries of the last stage, only the efficiency might be reduced.

To avoid too many occlusion queries given by the *Occupancy Map* in the backstage of the scene, we perform brute force multiple occlusion queries on the remaining nodes, since BBs in the back are often occluded:

```
while(pending list not empty){
  perform multiple occlusion query with pending list;
  clear pending list;

  for(each visible node){
    if(node is geometry)
      render node;
    else
      add children to pending list;
  }

  n++;
}

max_tests = 7/10 * n;
```

5 Results

	Number of polygons
BoomBox	644 268
Formula One	746 827
Cotton Picker	10 610 166
City Model	64 898 464

Table 1: Test models.

	Framerates [fps]				
	a) only vfc	b) occlusion culling		c) multiple queries	
BoomBox	22.1	31.1	(+41%)	42.9	(+94%, +38%)
Formula One	27.7	33.7	(+22%)	41.9	(+51%, +24%)
Cotton Picker	8.8	17.9	(+103%)	20.6	(+135%, +15%)
City Model	0.5	19.9	(+3880%)	20.7	(+4040%, +4%)

Table 2: Resulting average framerates and corresponding speed-ups for the test models.

To evaluate the performance of the multiple occlusion queries, organized by the *Occupancy Maps*, we used an Intel P4@2400 MHz with a nVIDIA Geforce FX5600XT and the models listed in Table 1. We rendered a camera path for each model at a resolution of 800 $\times$ 600 and 32 Bit color depth. The occlusion culling was done with the nVIDIA occlusion query extension.

The Boom Box, the Formula One car and the Cotton Picker (see Figure 9a/c) are MCAD models. The City model (see Figure 9d) is an artificial city with some Formula One cars in the streets and has the highest complexity of the four models with very high depth complexity. We did not optimize the scene graph for occlusion culling or traversal.

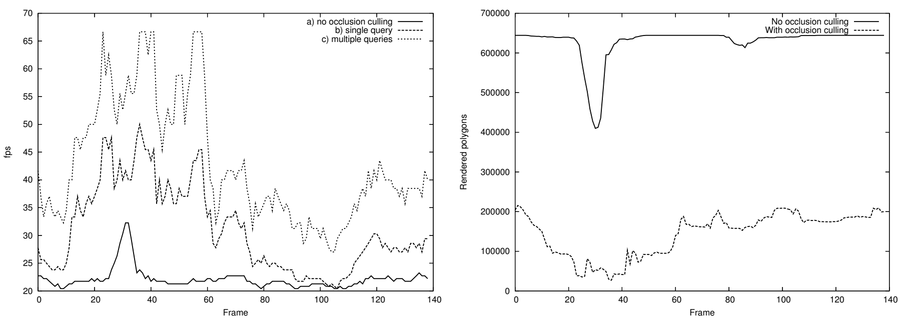


Figure 5: Boom Box frame rates and rendered polygons.

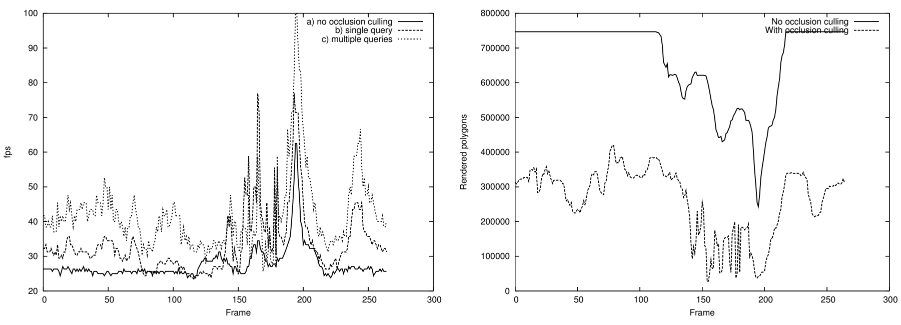


Figure 6: Formula One frame rates and rendered polygons.

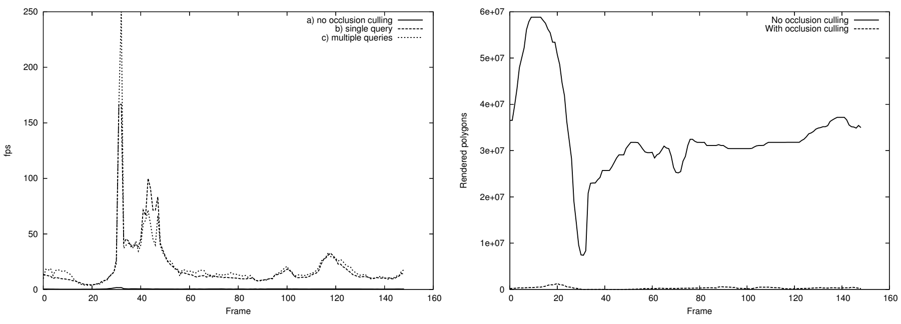


Figure 7: City frame rates and rendered polygons.

We rendered different camera paths for each model. In some frames, we zoomed into the scene and parts of the scene outside of the view frustum were culled. Our measurements are performed a) only with view frustum culling, b) with a single occlusion query for each node in the scene graph (without using an *Occupancy Map*), c) with multiple occlusion queries organized by *Occupancy Maps*.

With occlusion culling, we achieved average framerates between 17.9 fps (Cotton Picker) and 33.7 fps (Formula One) for the different models (see Table 2.) With the new traversal scheme with *Occupancy Maps* and multiple occlusion queries we improved these results to 20.6 fps (Cotton Picker) and 42.9 fps (Boom Box). The best speed-ups were achieved with multiple occlusion queries in scenes with lower depth complex-

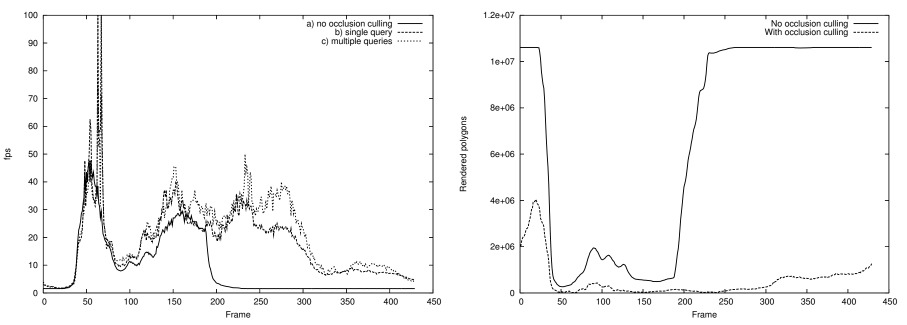


Figure 8: Cotton Picker frame rates and polygons.

ity, because of the trade-off between rendering and occlusion query state changes. In scenes with very high depth complexity, the occlusion queries dominate and with very low depth complexity the rendering dominates.

In Figures 5−−8 the frame- and renderrates (how many polygons are rednered) are given. In scenes with higher depth complexity, the speed-up is not high, because most of the costs are the fillrate for the occlusion tests and the scene graph traversal in the (occluded) backstage of the scene, which is similar with or without multiple queries.

6 Conclusions

In this paper we presented an algorithm for hierarchical occlusion culling with multiple occlusion queries in hardware. The algorithm uses a screen space approximation of the bounding boxes in the scene graph to arrange bounding volumes for occlusion queries during traversal of the scene graph. Only a few queries are necessary to get the visibility information of the scene graph nodes. We achieved a rendering speed-up of up to 38% compared to the traditional occlusion test in each node. The algorithm requires no special scene organization and no preprocessing, also it does not utilize temporal coherence and can be implemented with almost every hierarchical scene representation.

6.1 Future work

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.

No temporal coherence is used in the presented approach, but could be used to further speed up the construction and organization of the traversal and culling.

Another obvious improvment is to render the geometry nodes in a state sorted fashion to further reduce the number of state changes during rendering. This can be easy implemented in the presented traversal scheme.

Acknowledgements

This work was supported by the OpenSG PLUS project of the german federal ministry of education and research (bmb+f). The MCAD datasets are courtesy of Engineering Animation Inc. and Hewlett-Packard Company.

References

[BKS01] D. Bartz, J. Klosowski, and D. Staneker. Tighter Bounding Volumes for Better Occlusion Performance. In *Visual Proc. of ACM SIGGRAPH*, page 213, 2001.

[BMH99] D. Bartz, M. Meißner, and T. Hüttner. OpenGL-assisted Occlusion Culling of Large Polygonal Models. *Computers & Graphics*, 23(5):667–679, 1999.

[BS99] D. Bartz and M. Skalej. VIVENDI - A Virtual Ventricle Endoscopy System for Virtual Medicine. In *Proc. of Symposium on Visualization*, pages 155–166,324, 1999.

[BSS$^+$01] D. Bartz, D. Staneker, W. Straßer, B. Cripe, T. Gaskins, K. Orton, M. Carter, A. Johannsen, and J. Trom. Jupiter: A Toolkit for Interactive Large Model Visualization. In *Proc. of Symposium on Parallel and Large Data Visualization and Graphics*, pages 129–134, 2001.

[COCSD03] D. Cohen-Or, Y. L. Chrysanthou, C. T. Silva, and F. Durand. A Survey of Visibility for Walkthrough Applications. *IEEE Transactions on Visualization and Computer Graphics*, 9(3), 2003.

[CT96] S. Coorg and S. Teller. Temporally Coherent Conservative Visibility. In *Proc. of ACM Symposium on Computational Geometry*, pages 78–87, 1996.

[Gar99] M. Garland. Multiresolution Modeling: Survey and Future Opportunities. In *Eurographics STAR report 2*, 1999.

[GKM93] N. Greene, M. Kass, and G. Miller. Hierarchical Z-Buffer Visibility. In *Proc. of ACM SIGGRAPH*, pages 231–238, 1993.

[HP98] Hewlett-Packard. Jupiter 1.0 Specification. Technical report, Hewlett Packard Company, Corvallis, OR, 1998.

[KS01] J. Klosowski and C. Silva. Efficient Conservative Visibility Culling Using the Prioritized-Layered Projection Algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 7(4), 2001.

[MBGS01] M. Meißner, D. Bartz, R. Günther, and W. Straßer. Visibility Driven Rasterization. *Computer Graphics Forum*, 20(4):283–294, 2001.

[NVI02] NVIDIA. NVIDIA OpenGL Extension Specifications, 2002.

[OF00] OpenSG-Forum. OpenSG - Open Source Scenegraph. http://www.opensg.org, 2000.

[SBM02] D. Staneker, D. Bartz, and M. Meißner. Using Occupancy Maps for Better Occlusion Query Efficiency. In *EG Workshop on Rendering*, 06 2002. Poster.

[SBM03] D. Staneker, D. Bartz, and M. Meißner. Improving Occlusion Query Efficiency with Occupancy Maps. In *Proc. of Symposium on Parallel and Large Data Visualization and Graphics*, pages 111–118, 2003.

[SBS04] D. Staneker, D. Bartz, and W. Straßer. Occlusion Culling in OpenSG PLUS. *Computers & Graphics*, 28(1):87–92, 2004.

[Sev99] K. Severson. VISUALIZE fx Graphics Accelerator Hardware. Technical report, Hewlett Packard Company, available from http://www.hp.com/workstations/support/documentation/ whitepapers.html, 1999.

[SOG98] N. Scott, D. Olsen, and E. Gannett. An Overview of the VISUALIZE fx Graphics Accelerator Hardware. *The Hewlett-Packard Journal*, (May):28–34, 1998.

[SPR$^+$01] P. Slusallek, S. Parker, E. Reinhard, H. Pfister, and T. Purcell. Interactive Ray-Tracing. In *ACM SIGGRAPH Course 13*, 2001.

[Sta02] D. Staneker. A First Step towards Occlusion Culling in OpenSG PLUS, 2002.

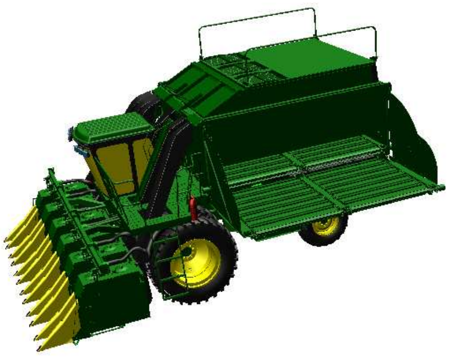
[Sta03] D. Staneker. An Occlusion Culling Toolkit for OpenSG PLUS. http://www.opensg.org/ OpenSGPLUS/symposium/, 2003.

[WFP$^+$01] M. Wand, M. Fischer, I. Peter, F. Meyer auf der Heide, and W. Straßer. The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. In *SIGGRAPH 2001*, 2001.

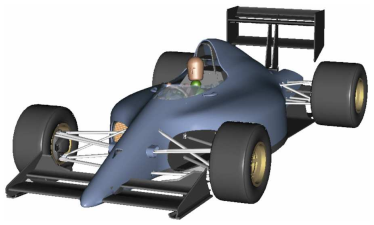
[ZMHH97] H. Zhang, D. Manocha, T. Hudson, and Kenneth E. Hoff. Visibility Culling Using Hierarchical Occlusion Maps. In *Proc. of ACM SIGGRAPH*, pages 77–88, 1997.

7 Datasets

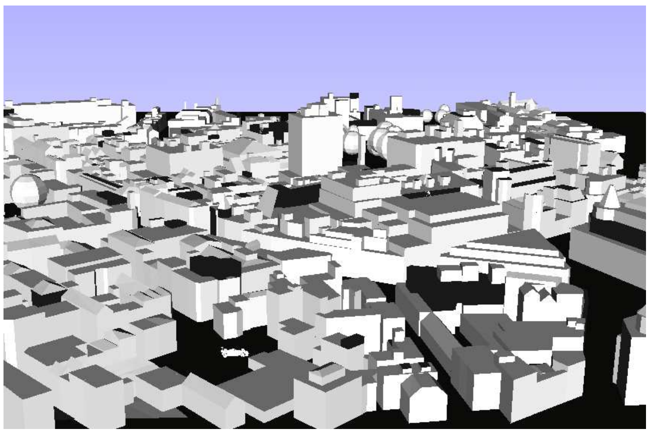
(a)



(b)



(c)



(d)

Figure 9: Used datasets: (a) Boom Box; (b) Cotton Picker; (c) Formula One; (d) City