Generation of Subdivision Hierarchies for Efficient Occlusion Culling of Large Polygonal Models

Michael Meißner, Dirk Bartz, Tobias Hüttner,
Gordon Müller, and Jens Einighammer

WSI-99-13

The same technical report is also published at
Institute of Computer Graphics, Technical University Braunschweig, TUBSCG-1999-6-1

ISSN 0946-3852
WSI/GRIS
University of Tübingen
Auf der Morgenstelle 10/C9 D-72076 Tübingen, Germany

Institute of Computer Graphics
Technical University of Braunschweig,
Rebenring 18, D-38106 Braunschweig, Germany

Email: {meissner, bartz, thuettne, jeinigh}@gris.uni-tuebingen.de,
gordon.mueller@tu-bs.de

WWW: http://www.gris.uni-tuebingen.de
http://www.cg.cs.tu-bs.de

ABSTRACT

Efficient handling of large polygonal scenes has always been a challenging task. In recent years, view-frustum and occlusion culling have drawn a lot of attention for reducing the complexity of those scenes. The problem of how to efficiently organize such scenes for fast image synthesis is widely neglected, although the answer heavily affects the overall performance.

In this paper, we present three novel algorithms for efficient scene subdivision and compare those with another already available algorithm for subdividing general polygonal models into a hierarchy of sub-models for an occlusion culling application. While the latter is available as a commercial product, the three other approaches introduce new algorithms for scene subdivision which achieve significant better results.

CR Categories:

E.1 [Data Structures]: Trees
I.3.5 [Computer Graphics]: Visibility Culling, Occlusion Culling
I.3.7 [Three-Dimensional Graphics and Realism]:
Hidden Line/Surface Removal

Keywords:

Large Polygonal Models, Hierarchical Scene Organization, Occlusion Culling.

1 Introduction

Hierarchical methods are crucial for the efficient handling of large computer graphics scenes. Several approaches address the problem of generating scene hierarchies, such as subdivision surfaces[5, 20, 19], multi-resolution frameworks[13, 18], scene databases[29], or regular subdivision schemes such as BSP-trees[7] or octrees[24]. An important application field of scene organization is view-frustum and occlusion culling, which specifically requires appropriate subdivision schemes. However, most available approaches lack either flexibility or efficient handling.

In this paper, we present three novel approaches to organize polygonal scenes hierarchically. Starting from general polygonal models, spatial coherence and a set of heuristics is exploited for this purpose. No special assumptions on the topology or the alignment are made in order to provide a robust subdivision scheme.
The goal of these model hierarchies is to ensure good occlusion culling performance, and hence better render performance. Consequently, we compare the generated hierarchies with an already available method, the spatialization technique provided in SGI's OpenGL Optimizer[16], and a regular subdivision, such as BSP-trees or Octrees. We use the generated hierarchies in an occlusion culling algorithm based on the Hewlett-Packard Occlusion Culling Flag[25]. Although our focus is on hierarchical and efficient rendering using occlusion culling, these algorithms can be applied to some hierarchical global illumination approaches as well[6].

This paper is organized as follows: In the next Section, we briefly outline previous approaches for the hierarchical organization of polygonal scenes. Section 2 introduces basic terminology and the occlusion culling algorithm used to evaluate the different subdivision approaches discussed in Section 3. In Section 4, we compare the results of the discussed approaches with respect to their render performance using occlusion culling. Finally, we present a conclusion and briefly describe future work.

1.1 Related Work

Several previous papers on visibility and occlusion culling touch the topic of scene organization. Generally, we observed that subdivision schemes for arbitrary polygonal models are difficult to derive. Hong et al.[12] used a technique which subdivides a CT-based colon volume dataset along its skeleton. The size of the different subdivision entities depends on how many voxels belong to this entity. While this scheme produced good results for a tube-like colon model, it is not efficient for general models.

Snyder and Lengyel[26] proposed that the designer of the scene needs to provide the subdivision. Similarly, Zhang et al. used a predefined scene database[29, 14]. Models built in large Computer-Aided-Design (CAD) systems might already include appropriate subdivision information, due to the design process which uses hierarchical notions like grouping and replication. Besides approaches which use a building floor plan for this purpose[1, 2, 28, 21], no other methods for deriving subdivision hierarchies from CAD models are known to us.

A more general approach is to organize a polygonal model into regular spatial subdivision schemes, such as BSP-trees[7, 23, 10] or Octrees[9, 11, 3, 4, 27]. While these subdivision schemes produce good results on polygonal models extracted by the Marching Cubes algorithm from uniform grid volume datasets — which provide a "natural" subdivision on Marching Cubes cell base —, these schemes run into numerous problems on general models. If a polygon of the model lies across a subdivision boundary, it must be either split into several parts, in order to produce a disjunct representation of the bounding entities, or handled in another special way. Splitting polygons however, can increase the number of small and narrow polygons tremendously.

In previous approaches for occlusion culling[15], we used the spatialization functionality of SGI's Optimizer package[16] — now part of the Fahrenheit Large Model Visualization API — which generates scene hierarchies automatically. However, these subdivision hierarchies needed to be tuned manually in order to get sufficient performance and motivated the work described in this paper.

2 Model Subdivision and Occlusion Culling

Although we focus in this paper on efficient hierarchical rendering using occlusion culling, there are several other applications where spatial coherence can be exploited to speed-up the respective processing. Specifically, a hierarchical scene organization can be used for global illumination approaches, collision detection, and many more.

2.1 Model Subdivision

Generally, a polygonal scene can be subdivided into smaller parts, where this subdivision can be either hierarchical or non-hierarchical. We call each part of this subdivision a *subdivision entity*. If information at different multi-resolution levels are required, usually a hierarchical organization is chosen, where different subdivision entities are combined into one parent entity which contains the whole information of the associated subdivision entities, or only information with less detail (a lower level-of-detail). This subdivision can be represented as a tree which we will call a *subdivision tree* or *subdivision graph*. We use an *OpenInventor*-based icon graph representation which consists of different kinds of nodes; *geometry nodes*, *subdivision nodes*, and *leaf nodes* (see Fig. 1). A geometry node only contains the geometry of the actual model with respect to the used subdivision method. In contrast, a subdivision node does not contain any geometry of the actual dataset; it only contains the spatial boundaries of the associated geometry nodes, thus the subdivision node is the implementation of the abstract subdivision entity. Finally, the leaf node is build of a subdivision node with exactly one geometry node as child.

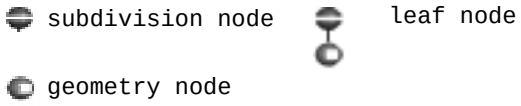


Figure 1: Possible nodes in a subdivision tree.

We test the suitability of the subdivisions for occlusion culling on three different polygonal datasets, which are described in Table 1. Due to grid limitations of the Octree-based Regular Space Decomposition approach (ORSD) (see Section 3.3), we test this subdivision approach only on a Marching Cubes-generated model (ventricle dataset).

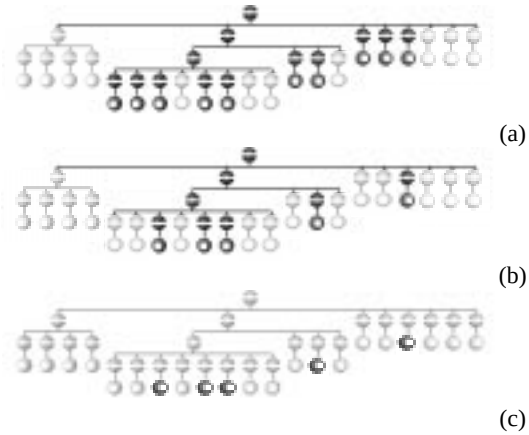
2.2 Occlusion Culling

We evaluate the subdivision quality of these four approaches with a standard hierarchical occlusion culling algorithm based on the HP-Occlusion-Culling-Flag[25] and a basic view-frustum culling algorithm[8].

The subdivision tree hierarchy is processed top-down, left-right. First, each subdivision node of the tree is tested for intersection with the view-frustum (view-frustum culling). All the nodes which have a non-empty intersection, are located within the view-frustum and

Dataset	Grid Type/ Source	#Triangles	FR
Ventricle System	Uniform/ MRI	270,882	4.6
Cathedral	Unstructured/ CAD	416,763	3.8
City	Unstructured/ Modeler	1,408,152	0.9

Table 1: Models; as gold standard for the speed-up due to occlusion culling, we show the frame rate for the datasets without any culling mechanism.



(a) (b) (c)

Figure 2: General Algorithm — not occluded nodes are shown with dark contrast, culled nodes are shown with bright contrast: Subdivision tree (a) after View-Frustum Culling and (b) after Occlusion Culling. The not occluded, and therefore rendered geometry nodes, are shown in (c).

may contain visible sub-structures. Consequently, we proceed with the view-frustum test, until we reach the leaf levels of the subdivision tree (see Fig.2a).
In the second step, all the remaining subdivision nodes are depth-sorted according to their associated bounding boxes and tested for occlusion using the HP flag. The actual geometry of the leaf node closest to the eye is rendered into the empty framebuffer. All further rendering is performed in an interleaved fashion. First, we render the bounding box of the next-closest subdivision node in a special occlusion mode which does not contribute to the framebuffer. If this bounding box would have a visible contribution to the framebuffer, the HP-Occlusion-Culling-Flag is set TRUE by the graphics hardware, and we proceed with the child nodes of the node; in case that it is a leaf node, we render the content of the geometry node in the standard render mode (see Fig. 2c). If the bounding box does not have any contribution (flag is set FALSE), this box and all the associated geometry are not visible and therefore, they are culled (see Fig. 2b).

3 Generating Subdivision Hierarchies

Generating hierarchical subdivisions of very large models can be done in numerous ways. In this Section, we will present three research algorithms and one commercial tool which generate a subdivision hierarchy starting from given models: *D-BVS*, *p-HBVO*, *ORSD*, and *SGI*.
The last is part of SGI's OpenGL Optimizer toolkit (opoptimize), while the other three novel algorithms have been developed and implemented by the authors. All approaches subdivide general polygonal models, whereas the octree-based ORSD only subdivides uniform grid datasets.

3.1 Dimension-oriented Bounding Volume Subdivision (D-BVS)

The goal of the volume oriented D-BVS subdivision algorithm is to generate evenly-sized, cube-shaped bounding boxes, hence minimizing the area of the screen projection of these bounding boxes. This goal is approached by splitting the bounding boxes multiple times in the largest dimension. The size of the bounding boxes and the associated sub-models is controlled by user-specified parameters, such as the minimal number of polygons.

Starting with the root subdivision entity — which contains the whole model — the associated bounding volume is split n_{split} times along its largest dimension such that each fraction is approximately of the same size as the second largest dimension.

$$n_{split} = \frac{\text{largest bounding volume dimension}}{\text{second largest bounding volume dimension}} \qquad (1)$$

This process continues recursively until the termination criteria are met. These criteria give lower bounds for the number of polygons of the subdivision entities or the size of the dimensions of the associated bounding boxes, in order to avoid undersized subdivision entities which increase the occlusion culling overhead without improving the cull rate sufficiently. Occasionally, the split-operation

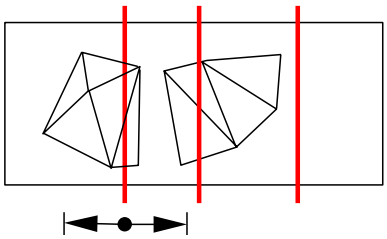


Figure 3: Moving subdivision planes to reduce polygon splits. The other planes are already calculated.

of the subdivision process splits a polygon which lies across the subdivision boundary into two new polygons (this is usually not the case for uniform grid datasets). This can tremendously increase the number of polygons and frequently, these polygons are small and narrow, thus resulting in numerical problems. To compensate this, we apply two techniques. First, the subdivision plane is moved along the subdivision dimension so as to reduce the number of additional polygons (Fig. 3). The direction and value of the movement is controlled by user-specified parameters. Nevertheless, very large polygons can not be handled by moving the subdivision planes, because they cover several high-level subdivision entities (i.e., a single polygon factory hall floor). Therefore, we apply our second technique, where those polygons are *pulled up* into a leaf node close to the tree root. The associated geometry is no longer affected by split operations in lower tree levels and is rendered, while the renderer traverses that part of the subdivision tree.

In general, this algorithm can handle all types of polygonal scenes without producing significantly more polygons. It optimizes shape and size of the subdivision entities, hence their bounding boxes. However, the polygon load is not evenly distributed to the

subdivision entities, possibly resulting in a less balanced subdivision tree.

3.2 Polygon-based Hierarchical Bounding Volume Optimization (p-HBVO)

The polygon-oriented Hierarchical-Bounding-Volume-Optimization (*p-HBVO*) method subdivides recursively a set of polygons into two subdivision entities. Instead of arbitrarily selecting possible subdivision planes, these planes are given by the barycenter of each polygon (triangle). By evaluating a cost-function, an optimal subdivision plane is established. At each subdivision level, the individual polygons are assigned to exactly one subdivision entity of that level. Consequently, no polygons are split, hence no new polygons are generated by this method.

Starting from the root node, at each subdivision step, we sort the polygons along all coordinate axes, where the barycenter of each polygon serves as sorting key. Based on these three ordered lists, we evaluate the potential subdivision planes along each axis for each entry in the respective list by splitting the sorted list of polygons into a *left* and *right* part. In contrast to pre-defined subdivision planes of the median cut scheme[17], we evaluate for each possible subdivision plane — defined by the entries in the lists — a cost function which approximates the costs of rendering the polygons of one of the two subdivision entities, generated by the respective subdivision plane. By minimizing this cost-function over all possible subdivision planes, we obtain an optimal subdivision plane which generates two new subdivision entities; one contains all *left*−polygons, the other one contains all *right*−polygons. The subdivision process terminates when either the number of polygons, or the subdivision depth exceeds one of the two predefined parameters: *Max_Triangles_Per_Subdivision_Entity* or *Max_Subdivision_Depth*. These parameters are specified by the user and supplied at the start of the subdivision process.
In most cases, our cost function is identical to one which has already been successfully applied in ray tracing environments[22]. Adopting this cost function is possible since the objective is the same; both algorithms traverse the scene graph in a similar way to determine visibility. The costs of a subdivision entity H, with children H_{left} and H_{right}, is given by:

$$C_H(axis) \quad = \quad \frac{S(H_{left})}{S(H)} \cdot |H_{left}| + \frac{S(H_{right})}{S(H)} \cdot |H_{right}|$$

where

- $|H|$ is the number of polygons within hierarchy H,
- $S(H)$ the surface area of the bounding box associated to subscene H, and
- $axis \in \{X, Y, Z\}$.

Overall, this algorithm generates well balanced subdivision tree with respect to their polygon load. Furthermore, polygons of individual objects are detected and clustered together (see Section 4, city dataset). Optimal performance was achieved with finer subdivisions, which usually leads to higher cull costs, but also enables lower render rates.

3.3 Octree-based Regular Space Decomposition (ORSD)

As mentioned earlier, regular subdivisions are well suited for uniform grid datasets, such as generated by MRI or CT scanners (i.e., ventricle dataset). Uniform grid datasets consist of a set of sample values (voxels), arranged on a uniform grid. A cube of eight neighboring voxels is called a cell, where cells with a non-empty intersection with the selected isosurface are called relevant cells.

We discuss such a subdivision scheme, based on an Octree-based Regular Space Decomposition method (ORSD). In contrast to the other approaches, ORSD uses a cell-based (voxel-based) evaluation criterion, where the number of relevant cells (relevant cell load or RCL) controls the subdivision process. This criterion is only a rough approximation of the actual number of extracted polygons, considering that each relevant cell represents between one and five triangles. In our experiences however, RCL turned out to be precise enough. Figure 9 visualizes a subset of the generated subdivisions. Note that the shown bounding boxes are bounding the actual geometry, not the respective octant volume. Consequently, they are of different sizes. After the construction of the entire octree, the

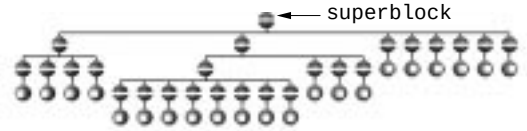


Figure 4: Octree-based subdivision tree.

RCL of each octant is already calculated. Subsequently, the octree is traversed recursively, starting with the superblock. If the RCL is above a user-specified threshold, the block is classified as a subdivision node, thus being further subdivided into its child blocks. In the other case, the block is considered as a leaf node. The associated relevant cells — at the bottom level of the octree — contain the polygons of the isosurface, which are assigned to that leaf node. This subdivision process results in a hierarchy of blocks, where the leaf nodes contain the actual geometry (see Fig. 4 and Section 2.1).

Overall, ORSD is a simple but efficient subdivision scheme. As shown in the results (see Section 4), the indirect evaluation method (RCL instead of number of polygons) does not affect the occlusion culling performance adversely. Furthermore, bounding box size and polygon load balance were adequate. However, ORSD is limited to uniform grid datasets.

3.4 SGI's opoptimize (SGI)

SGI's OpenGL Optimizer is a C++ toolkit for CAD applications that provides scene graph functionality for handling and visualization of large polygonal scenes. It includes mechanisms for subdivision of databases as well as for tessellation, simplification, and others.

Opoptimize (depicted in the following sections as "SGI"), which is part of the toolkit, provides functionality for the subdivision of model databases. The subdivision method realized in SGI is similar to the construction of an octree; each subdivision entity is split into eight equally sized subdivision entities. This process is repeated recursively, until a certain threshold criteria for the iterated subdivisions is reached.
Octree-based spatial subdivision is a simple and efficient subdivision scheme. However, the SGI subdivision mechanism subdivides space not by simply bisecting edges of a cube, as in an octree, but by choosing subdivision planes so that the rendering loads of the resulting parts are similar. As a result, the amount of geometry in each subdivision on each side of the cutting plane is approximately the same. Polygons which are split due to the subdivision are distributed to the respective subdivision entities.
The main parameters that can be used to control the subdivision are hints for the lowest and highest amount of triangles (tri_{min}, tri_{max})

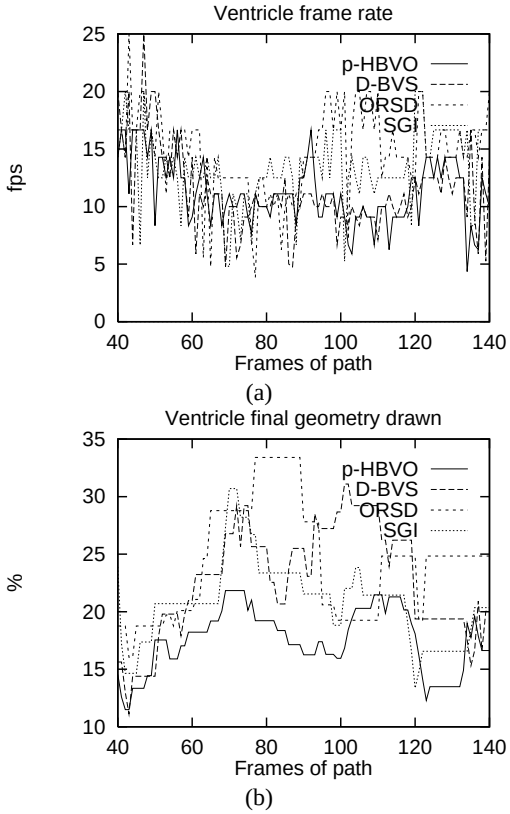


Figure 5: Ventricle dataset; window of frames 40 to 140 of the total path of 150 frames: (a) frame rate, (b) render rate.

in each subdivision entity at the leaf-level of the subdivision hierarchy. However, the subdivision algorithm only tries to meet these criteria but is not bound to it. Note, that this tool usually produces triangle strips to achieve better render performance.

In general, SGI generates subdivision hierarchies with a well-balanced polygon load. However, the bounding boxes of the subdivision entities are less suited for occlusion culling applications, because the cost function determining the subdivision entities is obviously not optimized with respect to the volume of the bounding boxes. We observed that the right-most branch of the subdivision tree frequently contained large subsets (bounding box volume size) of the model, even in the lower tree levels.

4 Results

In this Section, we discuss the efficiency of our three novel algorithms and opoptimize of SGI's OpenGL Optimizer (SGI) with respect to their occlusion culling-based render performance. All measurements are performed on a HP B180/fx4 graphics workstation. The three different polygonal datasets (see Table 1) represent typical scenarios of different application areas. Their subdivision trees are constructed in the same way, where the geometry is built of individual polygons (triangles) without any triangle strips. SGI usually generates triangle strips. In order to obtain a comparable model, we converted these strips into individual triangles. Furthermore, we removed redundant tree nodes. During our evaluation, we measured the time spent for view-frustum culling (vfc), occlusion culling (occ), and rendering of the not occluded geometry (ren) (see Tables 2, 3, and 4) of different subdivision granularities. For the evaluation, we took the subdivision with the best culling-based render performance. Furthermore, we show frame rate of walk-throughs of the datasets and the respective render rate, which mirrors the percentage of the geometry of the scene which was rendered (this is reciprocal to the cull rate, which represents the percentage which was not rendered due to culling). The success of the occlusion culling can be seen as well in the tables; the frame rate is shown for walk-throughs with vfc and occ, and vfc only.

Ventricle Dataset

The first dataset is a polygonal model of the ventricular system of the human brain extracted from a MRI scan. We explore the dataset by moving through the lower part (Cisterna Magna) of the polygonal model. Most of the model structures through-out the walk-through are located within the view-frustum, while the structures with the largest number of polygons (located in the upper part or lateral ventricles) were not visible due to occlusion. All polygons of this model are aligned on the uniform cell grid and are of approximately the same size. All three novel algorithms were able to detect this "natural" subdivision boundaries; only SGI generated approximately 15% additional polygons due to splitting operation in-between the grid points.

Figure 5 shows frame rate and render rate of the four evaluated algorithms; Table 2 shows the time measurements. The most interesting detail is the low amount of time consumed by view-frustum and occlusion culling by ORSD, due to its coarse subdivision. The render rate of p-HBVO and D-BVS was approximately 25% better than the render rate of the ORSD approach. However, the finer subdivision (see Fig. 9) introduced additional culling costs twice as much as for ORSD, resulting in a lower frame rate.

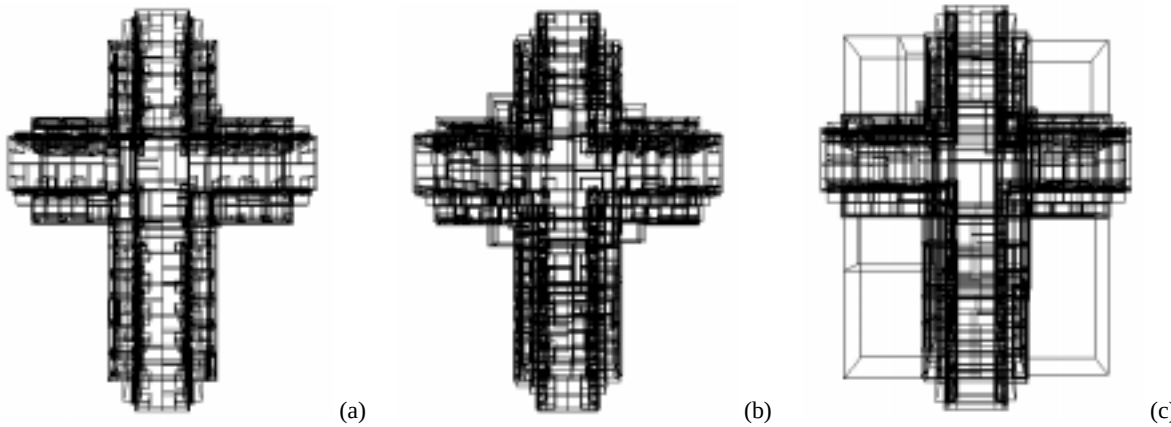
Cathedral Dataset

This dataset represents the interior of a gothic cathedral, designed with a CAD system (see Table 1). Occlusion is limited to small parts of the model, because a large share of the polygons are visible from most view points within the model. Figure 7 shows a very fine granular subdivision of the cathedral model. Especially the p-HBVO approach (a) adapts very nicely to the structures of the model, such as pillars and arcs. In contrast, the subdivision generated by SGI (c) introduces very large bounding boxes, which do not adapt properly to the actual geometry.

The p-HBVO approach performed best on this dataset (see Fig. 6). This is due to the low culling costs, compared to SGI and D-BVS (see Table 3). The bounding boxes of D-BVS and SGI are not really suited for occlusion culling; especially their occlusion culling time is significantly higher compared to p-HBVO, thus reducing the frame rate severely. Consequently, view-frustum culling only is faster than occlusion culling for those approaches.

City Dataset

The city dataset is an artificial model of 400 basic building models with some interior, which contains most of the polygonal complexity. Consequently, most of the polygons of this model are occluded. However, only the p-HBVO approach was able to subdivide all the interior into individual subdivision entities, hence resulting in very low render rates (see Table 4 and Fig. 8). In contrast, the D-BVS approach did not detect the interior objects. The optimal computed subdivision was of coarser granularity (10% of the subdivision entities of p-HBVO), resulting in less time spent for culling (vfc and occ), but a higher render rate of 41 times larger than the render rate of p-HBVO. Similar, the SGI approach did not detect the interior objects as well. It used a coarse granular subdivision, which con-



(a) (b) (c)

Figure 7: Cathedral dataset — subdivided by (a) p-HBVO, (b) D-BVS, and (c) SGI; the arts and pillars of the cathedral are well detected by p-HBVO and D-BVS. SGI only used a regular spatial subdivision.

sequently increased the culling time (vfc and occ) in comparison to p-HBVO.

Summary

Overall, two of our three novel model subdivision approaches were able to generate subdivisions with faster rendering due to higher cull performance. This was achieved by reducing culling costs or by reducing the render rate of the dataset. On uniform grid datasets, the basic ORSD approach produced a model subdivision which performed best, mostly due to the low time spent to establish occlusion or non-occlusion.

Generally, we observed that models with high occlusion do not require very fine subdivision (ventricle dataset, D-BVS vs. ORSD). On the other hand, a fine subdivision pays off if interior (thus completely occluded) objects are clustered in a subdivision entity (city dataset, p-HBVO vs. SGI). In contrast, models with low occlusion (cathedral dataset) can benefit from finer subdivisions if the culling costs loss is only a fraction of the rendering costs gain (see city dataset, p-HBVO). However, this was not true of the cathedral dataset (D-BVS).

Note that the p-HBVO approach builds a binary subdivision tree. This usually results in deeper trees, hence more intermediate subdivision entities. This increases the time spent for occlusion culling significantly. Once this binary tree was re-build into a quad tree representation, we achieved a frame rate increase of approximately 20%.

To summarize, we always achieved a speed-up due to occlusion culling-based rendering. Especially with the city dataset, we achieved a speed-up of 15.6 after culling of 99.9% of the model geometry with the p-HBVO algorithm. The other approaches obtained a similar speed-up, while culling less polygons.
On the ventricle dataset, the ORSD approach accomplished the best results; 77.6% of the geometry were culled, due to view-frustum and occlusion culling. This culling performance resulted in a frame rate speed-up of 3.3.

5 Conclusion and Future Work

In this paper, we discussed four different approaches for hierarchical subdivisions of large polygonal models. One of the candidates of the discussion is a commercially available product (SGI's OpenGL Optimizer), while the other approaches are still research projects.

Overall, the former approaches achieved similar (D-BVS) or better performance — evaluated with an occlusion culling application — than the tool of SGI's OpenGL Optimizer. It turned out that a simple octree-based scheme (ORSD) suits very well to uniform grid datasets; bounding box size as much as polygonal load balance was handled well. However, this scheme does not work on unstructured grid datasets because of the missing "natural" subdivision on cell-base.
Especially the p-HBVO approach performed well on all datasets. Whereas the D-BVS approach optimized the size and shape of the bounding boxes, the p-HBVO approach was able to generate good bounding boxes, while balancing the polygon load as well.

In the future, we will look into optimal subdivision tree traversal strategies. Usually, geometry of a model which is closest to the eye is not occluded. Consequently, if we assign a culling budget only to the rear geometry, the culling costs can reduced significantly.
So far, only spatial coherence information is exploited in order to combine raw triangles into subdivision entities. If neighborhood connectivity information —, i.e., semantic information which describes what kind of object should be combined (like the roof of a house in the city dataset) — is used, we expect subdivisions which perform better than the current ones. This will be a major future research focus.
Finally, hierarchical subdivision schemes can be used for numerous other fields in computer graphics. In the future, we will look into some of these fields, such as collision detection.

6 Acknowledgements

This work has been supported by German Research Foundation (DFG) Project SFB 382, the MedWis program of the German Federal Ministry for Education, Science, Research and Technology, the State of Baden-Württemberg, and by the DFG grant Fe 431/4-1 and Fe 431/4-2.

We would like to thank Tom Malzbender at HP Labs, Palo Alto, CA and Alan Ward at HP Workstation Systems Lab, Ft. Collins, CO for providing us with a HP B180 system with fx4 graphics. Furthermore, we thanks Michael Doggett for proof-reading.

References

[1] J. Airey, J. Rohlf, and F. Brooks. Towards Image Realism with Interactive Update Rates in Complex Virtual Building

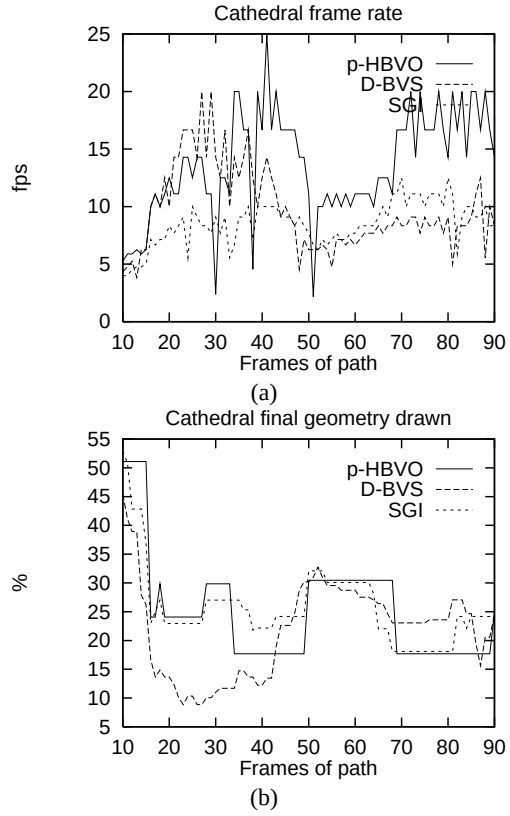


Figure 6: Cathedral dataset; all frames of the path are shown: (a) frame rate, (b) render rate.

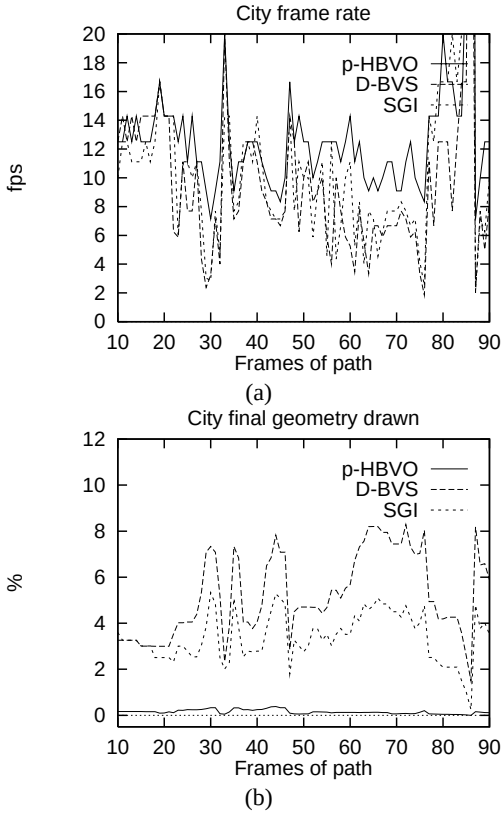


Figure 8: City dataset; window of frames 10 to 90 of the total path of 200 frames: (a) frame rate, (b) render rate.

Environments. In *Proc. of ACM Symposium on Interactive 3D Graphics*, pages 41–50, 1990.

[2] J.M. Airey. *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations*. PhD thesis, Department of Computer Science, University of North Carolina, Chapel-Hill, 1990.

[3] S. Coorg and S. Teller. Temporally Coherent Conservative Visibility. In *Proc. of ACM Symposium on Computational Geometry*, 1996.

[4] S. Coorg and S. Teller. Real-Time Occlusion Culling for Models with Large Occluders. In *Proc. of ACM Symposium on Interactive 3D Graphics*, pages 83–90,189, 1997.

[5] T. DeRose, M. Kass, and T. Truong. Subdivision Surfaces in Character Animation. In *Proc. of ACM SIGGRAPH*, pages 85–94, 1998.

[6] Andrew Glassner (Ed.). *An Introduction to Ray Tracing*. Academic Press, London, 4th edition, 1993.

[7] H. Fuchs, Z. Kedem, and B. Naylor. On Visible Surface Generation by a Priori Tree Structures. In *Proc. of ACM SIGGRAPH*, pages 124–133, 1980.

[8] B. Garlick, D. Baum, and J. Winget. Interactive Viewing of Large Geometric Databases Using Multiprocessor Graphics Workstations. In *SIGGRAPH'90 course notes: Parallel Algorithms and Architectures for 3D Image Generation*, 1990.

[9] N. Greene. *Hierarchical Rendering of Complex Environments*. PhD thesis, Computer and Information Science, University of California, Santa Cruz, 1995.

[10] N. Greene. Hierarchical Polygon Tiling with Coverage Masks. In *Proc. of ACM SIGGRAPH*, pages 65–74, 1996.

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

[12] L. Hong, S. Muraki, A. Kaufman, D. Bartz, and T. He. Virtual Voyage: Interactive Navigation in the Human Colon. In *Proc. of ACM SIGGRAPH*, pages 27–34, 1997.

[13] H. Hoppe. Progressive Meshes. In *Proc. of ACM SIGGRAPH*, pages 99–108, 1996.

[14] T Hudson, D. Manocha, J. Cohen, M. Lin, Kenneth E. Hoff, and H. Zhang. Accelerated Occlusion Culling Using Shadow Frusta. In *Proc. of ACM Symposium on Computational Geometry*, 1997.

[15] T. Hüttner, M. Meißner, and D. Bartz. OpenGL-assisted Visibility Queries of Large Polygonal Models. Technical Report WSI-98-6, ISSN 0946-3852, Dept. of Computer Science (WSI), University of Tübingen, 1998.

[16] Silicon Graphics Inc. Optimizer Manual. Technical report, 1997.

[17] T. L. Kay and J. T. Kajiya. Ray Tracing Complex Scenes. In *Proc. of ACM SIGGRAPH*, pages 269–78, 1986.

[18] R. Klein. Multiresolution Representation for Surface Meshes Based on the Vertex Decimation Method. *Computer & Graphics*, 22(1), 1998.

[19] L. Kobbelt, S. Campagna, J. Vorsatz, and H. Seidel. Interactive Multi-Resolution Modeling on Arbitary Meshes. In *Proc. of ACM SIGGRAPH*, pages 105–114, 1998.

[20] A. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. MAPS: Multiresolution Adaptive Parametrization of Surfaces. In *Proc. of ACM SIGGRAPH*, pages 95–104, 1998.

[21] D. Luebke and C. Georges. Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets. In *Proc. of ACM Symposium on Interactive 3D Graphics Conference*, 1995.

[22] G. Müller and D. Fellner. Hybrid Scene Structuring with Application to Ray Tracing. In *Proc. of ICVC'99 (to appear)*, 1999.

[23] B. Naylor. Partitioning Tree Image Representation and Generation From 3D Geometric Models. In *Proc. of Graphics Interface'92*, pages 201–212, 1992.

[24] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, 1994.

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

[26] J. Snyder and J. Lengyel. Visibility Sorting and Compositing without Splitting for Image Layer Decompositions. In *Proc. of ACM SIGGRAPH*, pages 219–230, 1998.

[27] O. Sudarsky and C. Gotsman. Output-Sensitive Visibility Algorithms for Dynamic Scenes with Applications to Virtual Reality. In *Proc. of Eurographics'96 conference*, pages 249–258, 1996.

[28] S. Teller and C.H. Sequin. Visibility Pre-processing for Interactive Walkthroughs. In *Proc. of ACM SIGGRAPH*, pages 61–69, 1991.

[29] 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.

Appendix A: Walk-through Measurements

Approach:	p-HBVO	D-BVS	ORSD	SGI
#Subdiv nodes	80	41	6	29
#Leaf nodes	81	41	28	36
Time for vfc [s]	0.0075	0.0065	0.0029	0.0041
Time for occ [s]	0.03	0.0232	0.0111	0.0163
Time for ren [s]	0.0506	0.0564	0.06	0.06
Render rate [%]	16.0	19.7	22.4	19.7
Frame rate [fps]	12.3	13.4	15.3	13.6
Render rate of vfc only [%]	70.6	78.7	83.0	78.9
Frame rate of vfc only [fps]	5.5	5.3	5.3	5.3

Table 2: Ventricle dataset: subdivision granularity, average render rates, frame rates, and time consumed by occlusion culling based rendering.

Approach:	p-HBVO	D-BVS	SGI
#Subdiv. nodes	9	59	51
#Leaf nodes	10	67	52
Time for vfc [s]	0.0021	0.0077	0.0089
Time for occ [s]	0.0041	0.0322	0.027
Time for ren [s]	0.1361	0.1239	0.1426
Render rate [%]	30.0	25.9	30.1
Frame rate [fps]	12.4	8.8	7.8
Render rate of vfc only [%]	33.1	25.9	32.2
Frame rate of vfc only[fps]	12.4	10.8	9.5

Table 3: Cathedral dataset: subdivision granularity, average render rates, frame rates, and time consumed by occlusion culling based rendering.

Approach:	p-HBVO	D-BVS	SGI
#Subdiv nodes	2722	266	420
#Leaf nodes	2723	495	420
Time for vfc [s]	0.0189	0.0111	0.0124
Time for occ [s]	0.0541	0.0318	0.0332
Time for ren [s]	0.0073	0.0831	0.0787
Render rate [%]	0.1	4.1	3.6
Frame rate [fps]	14.0	10.9	11.8
Render rate of vfc only [%]	47.0	50.7	50.4
Frame rate of vfc only [fps]	1.0	1.4	1.4

Table 4: City dataset: Subdivision granularity, average render rates, frame rates, and time consumed by occlusion culling based rendering.

Sample Hierarchies

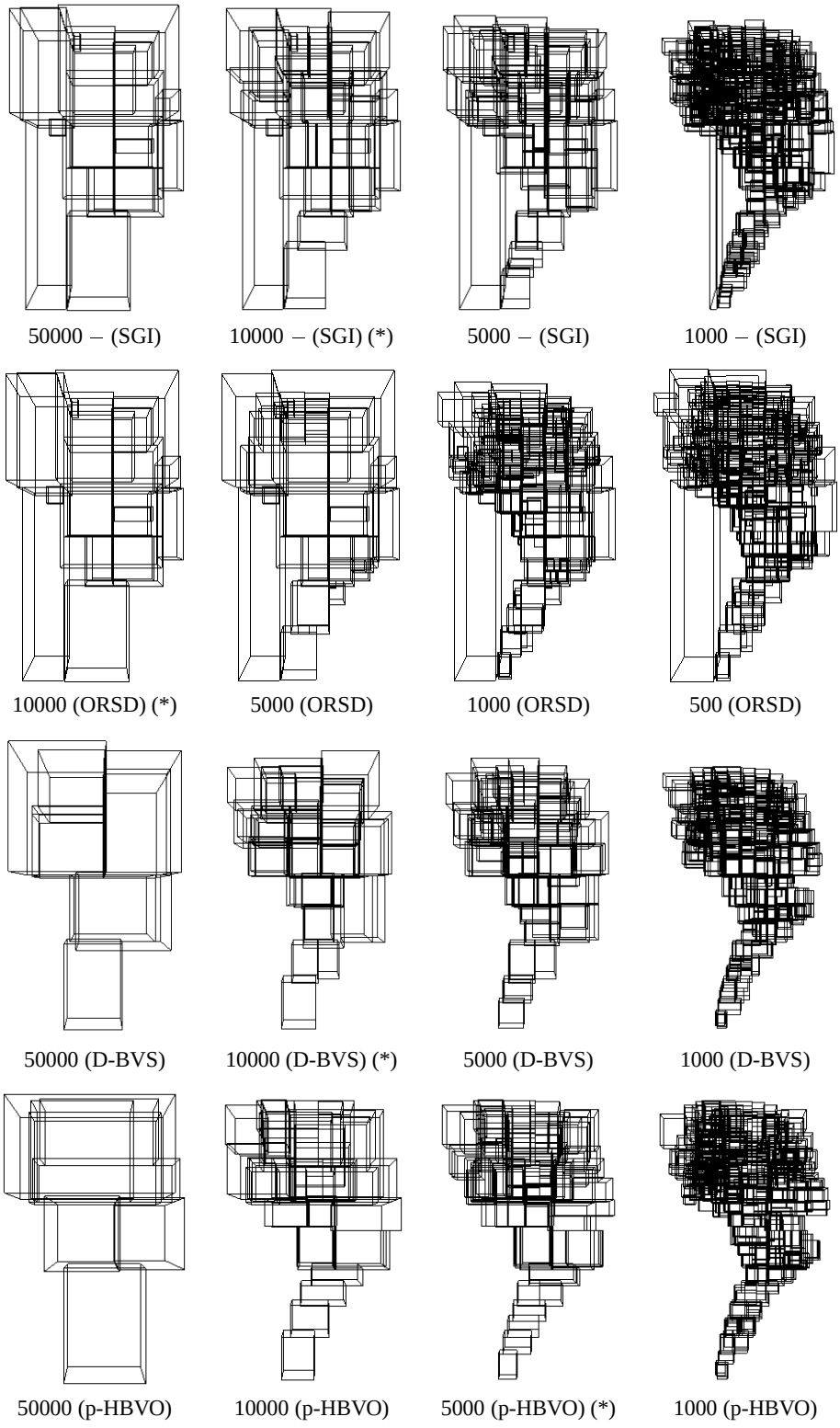


Figure 9: Different bounding box levels of the discussed subdivision methods. The asterisk (*) marks the used subdivision hierarchy, which provided the highest frame rate of this occlusion culling approach.

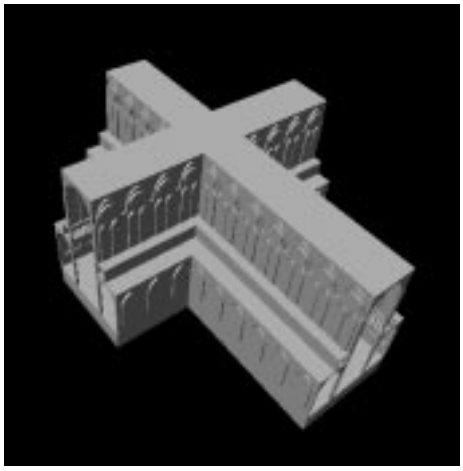
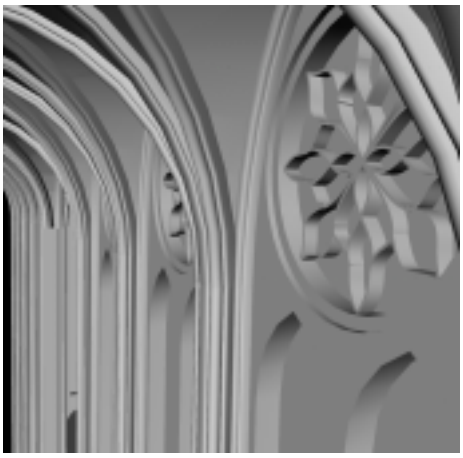

(a)

(b)

Figure 10: Cathedral dataset: (a) Overview, (b) Inside view.

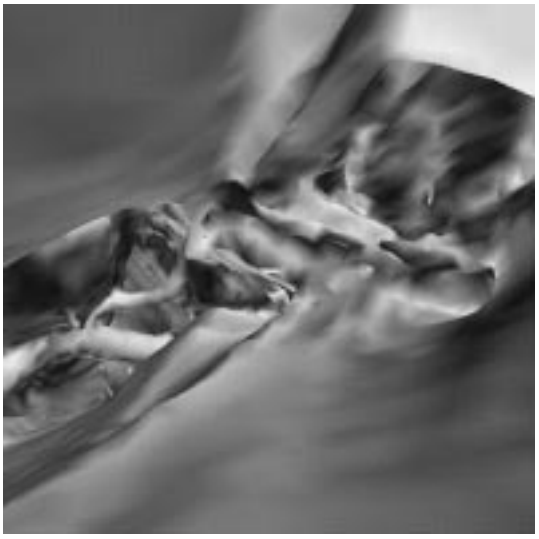

(a)

(b)

Figure 11: Ventricle dataset: (a) Overview, (b) Inside view.

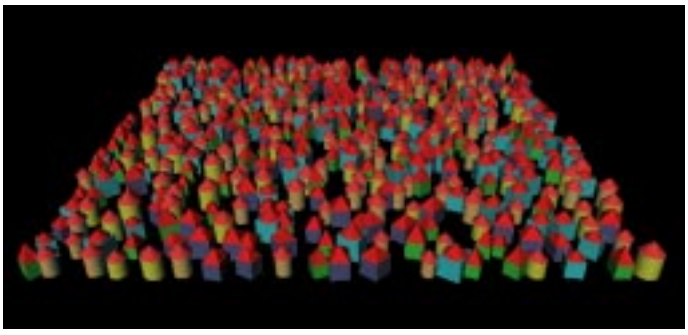
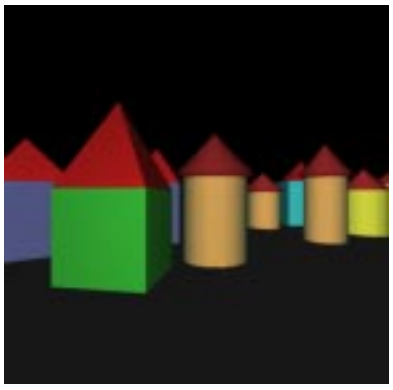

(a)

(b)

Figure 12: City dataset: (a) Overview, (b) Inside view.