Tighter Bounding Volumes for Better Occlusion Culling Performance

Dirk Bartz[1], James T. Klosowski[2], Dirk Staneker[1]
[1]Visual Computing for Medicine, University of Tübingen, Germany
[2]IBM, T.J. Watson Research Center, Hawthorne, NY, USA

WSI-2005-13
July 2005
(based on work done 2002 - 2004)

Visual Computing for Medicine
Graphisch-Interaktive Systeme
Wilhelm-Schickard-Institut
Universität Tübingen
D-72076 Tübingen, Germany
e-mail: bartz@gris.uni-tuebingen.de
WWW: http://www.gris.uni-tuebingen.de/areas/vcm

© WSI 2005
ISSN 0946-3852

Abstract

Bounding volumes are used in computer graphics to approximate the actual geometric shape of an object in a scene. The main intention is to reduce the costs associated with visibility or interference tests. The bounding volumes most commonly used have been axis-aligned bounding boxes and bounding spheres. In this paper, we propose the use of discrete orientation polytopes (*k*-dops) as bounding volumes for the specific use of visibility culling. Occlusion tests are computed more accurately using *k*-dops, but most importantly, they are also computed more efficiently. We illustrate this point through a series of experiments using a wide range of data models under varying viewing conditions. Although no bounding volume works the best in every situation, *k*-dops are often the best, and also work very well in those cases where they are not the best, therefore they provide good results without having to analyze applications and different bounding volumes.

CR Categories:

I.3.3 Picture/Image Generation: Viewing Algorithms, Occlusion Culling;
I.3.5 Computational Geometry and Object Modeling: Object Hierarchies;
I.3.5 Three-Dimensional Graphics and Realism: Hidden Line/Surface Removal;

Keyword: Visibility and occlusion culling, bounding volumes, large-scale data visualization

1 INTRODUCTION

Bounding volume hierarchies are a powerful technique for handling complex models. Such hierarchies are used to coherently organize the model as well as to provide multiple representations of the model, i.e. levels-of-detail. Furthermore, many visibility and interference tests are based on these hierarchies to reduce the associated costs. These include intersection tests for ray tracing [40, 22, 12, 28], interference tests for collision detection [13, 2, 23, 42, 24, 30, 25], and visibility tests for occlusion culling [15, 43, 3] or radiosity [16, 11].

In most applications, axis-aligned bounding boxes (AABBs) [12, 15, 39, 3] or bounding spheres [40, 32, 31, 20, 6] are used to approximate the geometric shape of an object. However, in many cases this approximation fills a much larger volume in object space, and a much larger projected area in screen space, than the actual geometry, resulting in additional (unnecessary) tests. These bounding volumes have been popular over the years because of their simplicity and effectiveness under the right conditions. A careful analysis of the situations in which AABBs perform well were conducted by several researchers [38, 44, 1], while others have extensively studied how to best create hierarchies of spheres [20, 6].

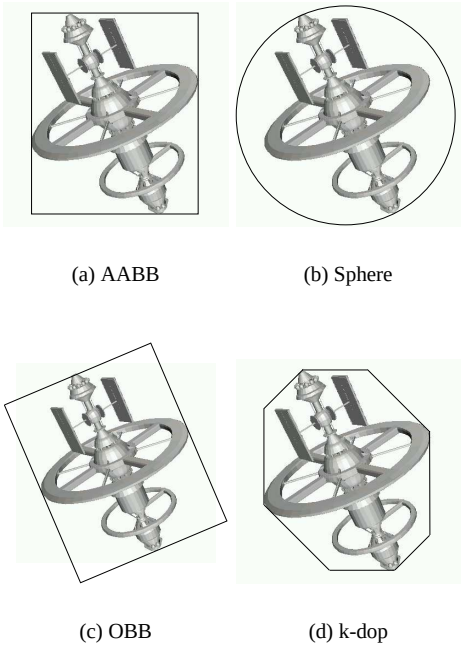
Alternatively, oriented bounding boxes (OBB) were proposed [33, 13, 2, 21], where the spanning axes of the bounding box are oriented according to the shape of the object, thus generating a tighter approximation of the original shape than AABBs. While OBBs perform better for collision detection than AABBs, the benefits for occlusion culling are significantly smaller. This is mainly due to the fact that the rasterized screen area of an OBB is almost the same as for an AABB (see our comparison of surface areas in Table 5).

Klosowski et al.[18, 23] proposed a collision detection scheme using discrete orientation polytopes (*k*-dops) which enabled faster collision tests than OBBs in many cases. Essentially, *k*-dops are an approximation of an object by computing bounding planes of that object along k/2 directions [22]. An AABB is one example of a 6-dop, whose bounding planes correspond to the coordinate axes. Another common *k*-dop is the 26-dop, which is an AABB with the twelve edges and eight corners cut to the object's surface (6 + 12 + 8 = 26 bounding planes). The tightest (convex) bounding volume that one could use is the convex hull. While the convex hull can minimize the number of interference tests, it is much more expensive to compute and store than most other bounding volumes.

The major contribution of this paper is the introduction of more complex bounding volumes for occlusion culling. We further present empirical evidence for the suitability of these bounding volumes. We specifically investigate the tradeoffs between bounding volumes including query times, construction times, and complexity (rendering and storage). Compared to AABBs and bounding spheres, *k*-dops enable a closer approximation of the geometric shape of an object which in turn generates a smaller number of necessary occlusion tests. The closer approximation comes at the expense of a slightly more complex bounding volume to compute and to store, which could result in more expensive occlusion tests. To determine the practicality of *k*-dops for occlusion culling algorithms, we conduct a thorough test of several bounding volumes on a variety of data models. While in general we cannot guarantee that any one bounding volume will be the best for all models, we show that *k*-dops are an excellent choice that work well in the vast majority of situations, thus eliminating the question of which bounding volume to use for a given situation.

2 BOUNDING VOLUMES

Many bounding volumes have been used for approximating complex models and accelerating interference and visibility queries.



(a) AABB (b) Sphere

(c) OBB (d) k-dop

Figure 1: Approximations of an object using bounding volumes: (a) an axis-aligned bounding box (AABB), (b) a sphere, (c) an oriented bounding box (OBB), and (d) a *k*-dop (where $k = 8$).

For example, spheres, AABBs, and OBBs have been popular within collision detection, ray tracing, and visibility algorithms. Figure 1 illustrates these bounding volumes approximating a space station model (in 2D). The quality and efficiency of these bounding volumes are measured based upon conflicting objectives including: (a) approximation quality, (b) rendering complexity, (c) computational expense, and (d) interference complexity.

Bounding volumes should tightly fit the underlying geometric shape of the object to provide good approximations. Tight fitting bounding volumes can reduce the number of visibility queries by eliminating *false positives*, i.e., queries which indicate, based upon the bounding volume information, that an object is visible when, in fact, it is not. Bounding volumes are also used for visualization purposes within occlusion culling and level-of-detail algorithms. Thus, the rendering complexity of a bounding volume can be an important factor when selecting which volumes to use within a visibility algorithm.

A third objective when selecting bounding volumes is the time required to compute the volumes. When a data set is modified often, either during the design phase, a rigid-motion animation sequence, or as the result of using vertex shaders, recomputing a bounding volume hierarchy, either from scratch or by using the previous hierarchy, must be an efficient operation. If the data is more stable, this becomes less of an issue. In other applications, the complexity of determining whether two bounding volumes interfere is important. This is true for collision detection algorithms that determine if models are in contact with one another. However, for image-based visibility algorithms, this objective is less important, which is why spheres do not have a specific advantage due to their fast interference tests. We therefore refer the interested reader to [13, 23] for more details on this objective.

As indicated above, the most relevant objectives when choosing a bounding volume depend upon the application domain. Within the current context of occlusion culling, objectives (a), (b), and (c)

are the most relevant and are discussed with respect to *k*-dops in further detail below.

3 K-DOPS

Klosowski et al. [18, 23] proposed the use of discrete orientation polytopes, or *k*-dops, to approximate complex models and accelerate collision detection queries. A *k*-dop is a convex polytope whose facets are determined by halfplanes whose outward normals come from a small fixed set of *k* orientations. An axis-aligned bounding box is one example of a 6-dop, since the six facets of the AABB are determined by the halfplanes whose normals correspond to the positive and negative coordinate axes. Figure 1d illustrates a 2D example of a *k*-dop, where $k = 8$. *k*-dops have been designed to use pairs of parallel planes (defined by pairs of collinear, but oppositely oriented, vectors), analogous to the AABB. Consequently, a *k*-dop can be completely defined by $k/2$ intervals which describe the extents of the *k*-dop along the $k/2$ fixed directions. One benefit of this is that storing *k*-dops is very memory efficient, requiring only *k* values each, since the orientations are known.

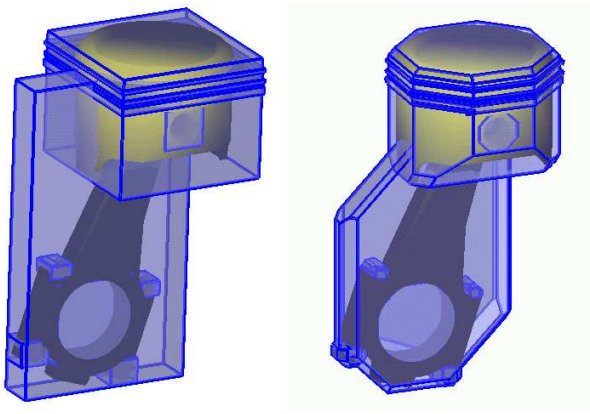
Although the term *k*-dops was coined in 1996 [18], Kay and Kajiya [22] first used this type of bounding volume in their ray tracing work in 1986. They described the pairs of parallel planes used to approximate an object as bounding slabs, and provided an efficient technique for intersecting a ray and a bounding volume consisting of multiple bounding slabs. In addition to ray tracing [22] and collision detection [23, 42, 17], several researchers have also used *k*-dops for rendering purposes [36, 9]. To date however, *k*-dops have never been used for occlusion culling.

3.1 Approximation Quality

k-dops attempt to strike a compromise between the poor approximations of spheres and AABBs and the increased complexity of OBBs and convex hulls. For larger values of *k*, *k*-dops allow approximations to more closely resemble the convex hull of the object, while still maintaining a simple structure. Even for small values of *k* (e.g. 14, 18, 26), *k*-dops can provide a well-distributed sampling of bounding planes and achieve most of the benefits of using completely arbitrary planes for each bounding volume. Although the bounding planes of each OBB can be selected to tightly approximate the geometry, there are still only six planes being used. The *k*-dop on the other hand, uses additional planes to eliminate the empty regions of space typically found in the corners of bounding boxes (see Figures 2 and 3). Thus, *k*-dops do not rely on the geometry being oriented in any particular way. The number and distribution of the bounding planes in our *k*-dops completely determine the effectiveness of their approximations. Many combinations of fixed orientations have been experimented with in different areas, including 14-dops [22, 18, 23] (consisting of the six AABB planes and the eight corner planes), 18-dops [9, 23] (consisting of the six AABB planes and the 12 edge planes), and 26-dops [18, 36, 23] (the combination of all of these planes).

3.2 Bounding Volume Complexity

In some applications, bounding volume hierarchies are used for rendering as well as visibility queries. In both instances, the rendering complexity of the bounding volume influences the efficiency of the operations. We typically measure the rendering complexity in terms of the number of triangles to be rendered or by the bounding volumes size, e.g. surface area or volume, which serves as a good approximation of the number of pixels that its triangles occupy on the screen after rasterization. *k*-dops are again a good compromise between all of the bounding volumes. They do not require that many more triangles than AABBs or OBBs, but they usually occupy a smaller screen area (see Section 5). Although convex hulls occupy the least screen area of all convex bounding volumes, they are the most complex to render in terms of the number of triangles, which limits their efficiency.



(a) AABBs (b) 26-dops

Figure 2: Piston components of an engine model and their (transparent) bounding volumes. The 26-dops reduce the empty space in the AABBs and are a significantly tighter approximation.

In addition to increasing the rendering complexity, bounding volumes represented by many triangles have a larger storage requirement than the less complex bounding volumes. For extremely large data models, this additional memory usage may become a limiting factor with respect to the models that can be handled.

3.3 Computational Expense

By design, *k*-dops are almost as efficient as spheres and AABBs to compute. To compute an AABB for an object, one needs only calculate the minimum and maximum *x*, *y*, and *z* values for all of the vertices in the object. This is equivalent to taking the dot product of each of the vertices in the object with the three vectors (1,0,0), (0,1,0), (0,0,1), and then finding the minimum and maximum values for each of the vectors. The same approach applies to computing *k*-dops. For the 26-dop mentioned earlier, the 13 vectors would be (1,0,0), (0,1,0), (0,0,1), (1,1,0), (1,0,1), (0,1,1), (1,-1,0), (1,0,-1), (0,1,-1),(1,1,1), (1,-1,1), (1,1,-1), (-1,1,1). It is common practice when computing AABBs to avoid the dot product (in this case three multiplications and two additions) and simply compare the current minimum and maximum *x*, *y*, and *z* values to each of the vertices in the object. For the 26-dop described above (and all of the other *k*-dops that we use), this practice also applies and results in more efficient computations.

Our discussion on the computation of *k*-dops so far has been limited to finding the $k/2$ intervals which bound the geometry. For the occlusion culling queries that we perform (see below), we need to determine the boundaries (vertices and triangles) of the *k*-dops. We compute this information using geometric duality [29, 10] and a convex hull algorithm. The basic idea behind duality is that points, lines, and planes can be interpreted in different ways. For example, a line $y = mx + b$ (in 2D) can also be specified by two points (m, b). Thus, each point in the 2D plane could be interpreted as a point, or as a line whose coordinates correspond to its slope *m* and intercept *b*. One of the purposes of using alternative interpretations is that the structure and relationships between these objects may be-

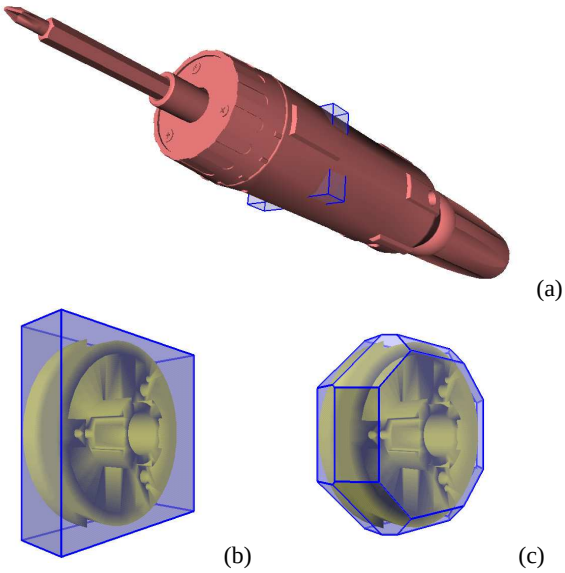


Figure 3: Screw driver: (a) Full dataset with (blue) AABB of a motor part; the 26-dop is not visible. (b) Motor part with transparent AABB, and (c) with transparent 26-dop.

come clearer under a different interpretation. In the present context, to find the intersection of k planes in 3D, we map each plane (respectively edge/vertex) to a vertex (edge/plane) in dual space and then compute the convex hull of these k vertices in $O(k \log k)$ time. Thanks to the properties of duality [29], we can then map the convex hull in dual space back to find the intersections of the k bounding planes. Again, each vertex (edge/plane) in dual space maps back to a plane (edge/vertex) from which we now have our boundary representation of the k-dop. Since k is typically a relatively small constant, this preprocessing step is quite fast.

The complexity of the entire k-dop construction is $O(nk) + O(k \log k)$ (computing the k bounding planes plus the boundary representation of the intersection of the bounding planes). Since k is a fixed constant (typically much less than n, the number of vertices in the geometry), this reduces to $O(n)$, versus $O(n \log n)$ for computing convex hulls. Tight fitting OBBs can also be constructed in $O(n \log n)$ time [13], or in $O(n)$ time at the cost of looser fitting approximations. As asymptotic complexity can hide large constants which influence implementations of these construction algorithms, we have included in Figure 5 the actual computational cost for a single bounding volume as the amount of geometry increases.

In the experiments described later, the underlying geometry of the models is not being modified between frames, only the camera is changing. However, if this were the case (see future work discussion), k-dops can be updated efficiently if the geometry they approximate undergoes rigid motion. By transforming the bounding vertices of the original k-dop, a new bounding volume can be recomputed quickly by finding the minimum and maximum values of these transformed vertices along the k-dops $k/2$ directions. This recomputation is considerably faster than the original calculation as the k-dop will (typically) have significantly fewer vertices than the geometry being approximated. Please refer to Section 5 for the maximum number of triangles (and hence vertices) in the k-dops that we discuss.

For the occlusion culling algorithm that we describe below, we would not need to even recompute the $k/2$ bounding planes. Once the initial k-dops have been computed (bounding planes, boundary triangles, and vertices), the boundary triangles will always be valid as the model (and its boundary) undergoes rigid motion. We could simply update the vertices of the original k-dop, and then render the same triangles (i.e. triples of vertices indices) for the occlusion culling queries.

4 CULLING ALGORITHM

For our experiments, we have focused on a straight-forward occlusion culling algorithm that allows us to easily interchange several bounding volumes and generate interesting statistics regarding their effectiveness, including overall culling percentage, frame rate, and triangle complexity of the bounding volumes. We refer the interested reader to an excellent survey on visibility culling [7] for a more general discussion of culling algorithms and their various strengths and weaknesses.

In the algorithm used here, we perform two stages of culling, as described by Bartz, et al. [4]. First, we perform a hierarchical view frustum culling operation to eventually determine which of the leaf nodes of the model hierarchy are not visible because they are outside of the user's field of view. (We use AABBs for this first stage because the additional complexity of the other bounding volumes would increase the view-frustum culling costs significantly – in contrast to the occlusion culling costs.) Based upon the implicitly calculated near and far depth values of the leaf nodes, we arrange the potentially visible nodes in a front-to-back list L. In the second culling stage, we process L in an interleaved fashion. After disabling the writing of the depth and color buffers, we test if rendering the bounding volume of the front-most leaf node in L would have modified the depth buffer, if it had been enabled. If no change would have resulted, the bounding volume is determined to be occluded, and its associated actual geometry does not need to be rendered. Otherwise, we enable the depth and color buffers for writing and render the associated actual geometry of the leaf node and proceed with the next node in the list.

For the detection of the potential change in the depth buffer, we use the Hewlett-Packard occlusion culling flag (HP flag) [34, 4], however, any other occlusion culling technique could be used (e.g. the virtual occlusion buffer [3]). Support for such occlusion queries has recently become the norm on almost all graphics hardware, including the adapters from NVIDIA, ATI, and HP. This functionality has been very well received because of the greater performance of the operations in hardware as opposed to software. Most recently many researchers have taken advantage of these features including (but not limited to) [4, 5, 19, 26, 8, 37, 14, 41].

The front-most n leaf nodes of L are rarely occluded; we render the geometry in these nodes without any occlusion test. The parameter n is very application dependent. For endoscopic applications, approximately 10% of the front-most nodes are never occluded. For mechanical CAD (MCAD) models, n is smaller (roughly 5%), since case elements are frequently occluding the interior geometry. For the architectural models, n is typically around 10%, with the exception of the Manhattan data where n is 15%, due to the viewer's perspective being a bird's eye view of the model and therefore more geometry is typically visible.

Hierarchical Culling Queries An alternative culling algorithm that we have experimented with performs hierarchical occlusion tests in addition to the hierarchical view frustum culling. In our original algorithm, we are using sequential occlusion tests on the leaf nodes in L. The alternative algorithm, starting at the root node, performs occlusion queries for the interior nodes as well in the hopes that we can quickly eliminate a large portion of the hierarchy (by finding an occluded interior node). While the idea is quite reasonable, this algorithm may not necessarily produce faster frame rates than our original algorithm because of the additional culling tests that are performed. As an example, consider the case

model	*# frames*	*# triangles*	*# objects*
Screw driver	180	156,424	83
Boom box	61	644,268	530
Formula One	41	746,827	306
London	208	864,716	718
City	200	1,403,096	228
Manhattan	132	2,938,836	698
Angiography	1138	1,817,731	278

Table 1: Number of viewpoints (frames), triangles, and objects for each model used in our experiments.

when a leaf node is visible from the current viewpoint. The bounding volume of this node, as well as every interior node on the path from the leaf to the root, will be found to be visible and we will have gained nothing by conducting these occlusion tests. If many of the interior node tests return false (meaning the node is not occluded), then we may inadvertently increase our overall rendering times. This was, in fact, the result presented in the paper describing the Jupiter toolkit [5]. Our results using this algorithm have been mixed and are presented in the following section.

5 EXPERIMENTAL RESULTS

We describe several experiments to demonstrate the benefits of using *k*-dops as bounding volumes in comparison to AABBs, which have consistently been the standard approximation for culling. For completeness, we have also tested other bounding volumes including spheres, oriented bounding boxes (OBBs), and convex hulls. The current tests were computed on a Linux PC with a 3 GHz Pentium 4 CPU and a NVIDIA FX 5600 graphics adapter.

Table 1 shows an overview of the models used for our experiments from MCAD, walkthrough systems, and from a medical scanner. Each model is stored in a hierarchical tree representation, where the leaf nodes of these trees contain the actual geometry of the models, while the interior nodes combine several leaves to produce higher hierarchy elements. For most of the data, the model hierarchies were specified by the original component hierarchies specified during the data creation process. In the cases where the model hierarchies were not provided with the data (London, City), we used a triangle clustering algorithm, whose evaluation method is the distance of the barycenters of the triangles, and generate a spatial organization of the scene into compact objects [27]. In all of these cases, the hierarchies do not specifically favor any one bounding volume over any of the others.

For each model, we perform the occlusion culling tests as described in Section 4 for arbitrary viewpoints, which represent typical view situations of the data sets, such as rotations, close-ups, and walkthroughs of the geometry. Rendering using OpenGL was not optimized for frame rates; we used display lists of lit triangle meshes, but not triangle strips or vertex arrays. The models we have used are described below and illustrated in Figures 3-10. With the exception of the City and Manhattan models, all of our models are from real-world data. The models we chose to use provided a good spectrum of models to test various aspects of the complete rendering system.

Screw driver An MCAD model consisting of a deep tree hierarchy where the screw driver's case is composed of two parts which occlude most of the interior geometry (Figure 3a).

Boom box An MCAD model consisting of a deep tree hierarchy. Most of the geometry is organized in the central unit of the boom box, which is occluded by the cover components (Figure 9).

Formula One The racing car is a complex MCAD model organized in a flat hierarchy with tightly fitting outer components (Figure 4).

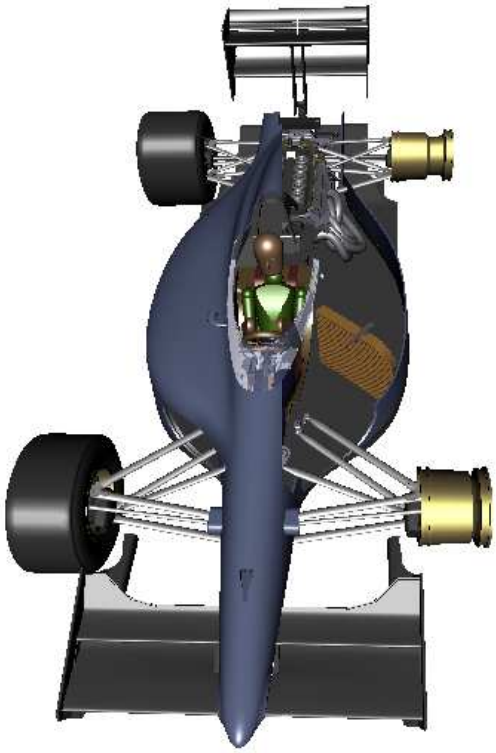


Figure 4: Formula One: the cover parts have been removed on the right side of a clipping plane.

London A model of the Westend of London where some buildings contain added interior geometry. The hierarchy is a quadtree-like structure which is generated using a triangle clustering algorithm from ray tracing [28]. The walkthrough of this model is done at the pedestrian (street) level (Figure 10).

City An artificial scene generated by a modeling system where all buildings contain interior geometry. Its quadtree-based hierarchical organization is generated using the same triangle clustering algorithm as above. This walkthrough is also from a pedestrian's viewpoint (Figure 8).

Manhattan A generated city model of skyscrapers and smaller buildings. The model itself is laid out on a checkerboard-like grid and every building is enriched with additional geometry, i.e. furniture, which is completely enclosed in the building. Here, the walkthrough is really a fly-over of the city, i.e. from a bird's perspective (Figure 6).

Angiography An extracted isosurface of an arterial blood vessel from a rotational angiography scan of a human head. The hierarchy is based on an octree decomposition of the original volume dataset (Figure 7).

5.1 Culling Performance

Table 2 presents the occlusion culling results for all of our data sets. For all but one of these models, we achieved additional culling using *k*-dops in comparison to bounding volumes traditionally used for occlusion culling (AABBs). As the major reason for the better performance, we note the tighter approximation of the *k*-dop-based bounding volumes. More important than the approximation quality, the increased culling performance also resulted in an increased rendering performance as measured by frame rate. Compared to AABBs, 26-dops provided increases in frame rates of 30% (screw

Model		AABB	OBB	Sphere	14-dop	18-dop	26-dop	CH
Screw driver	culling %	56.0	56.1	51.6	63.0	69.8	70.1	74.4
	fps	59.5	59.1	51.0	65.4	76.2	77.5	**82.0**
	# triangles	12	12	320	43(44)	54(60)	77(92)	301(770)
Boom box	culling %	76.5	74.5	69.7	76.8	77.0	77.1	77.4
	fps	**20.8**	19.5	15.5	20.2	20.1	20.4	18.0
	# triangles	12	12	320	44(44)	55(60)	80(92)	578(3264)
Formula One	culling %	66.0	66.7	51.9	69.3	69.9	69.8	73.9
	fps	19.9	19.9	14.1	21.4	21.7	21.4	**22.8**
	# triangles	12	12	320	44(44)	57(60)	85(92)	433(4390)
London	culling %	97.3	97.1	96.6	97.5	97.7	97.8	98.0
	fps	55.2	50.4	41.5	55.5	57.7	**58.8**	57.9
	# triangles	12	12	320	35(44)	45(60)	66(92)	191(1340)
City	culling %	96.3	96.3	95.7	97.1	97.3	97.4	97.6
	fps	58.6	56.1	46.9	64.7	66.8	67.0	**68.8**
	# triangles	12	12	320	44(44)	56(60)	83(92)	75(1340)
Manhattan	culling %	92.8	92.4	91.6	93.4	93.4	93.5	93.9
	fps	17.6	16.2	14.3	18.0	18.5	18.7	**19.1**
	# triangles	12	12	320	41(44)	50(60)	73(92)	28(684)
Angiography	culling %	97.2	97.0	96.9	97.3	97.3	97.4	97.4
	fps	60.0	54.7	51.4	60.6	60.9	**61.1**	60.6
	# triangles	12	12	320	43(44)	53(60)	79(92)	209(734)

Table 2: Occlusion culling statistics: For each model, the overall performance of each bounding volume is listed as the average percentage of geometry culled (culling %) and the frames per second (fps) achieved over a series of viewpoints. The best frame rate is highlighted in bold for each model. The bounding volumes are the axis-aligned bounding box (AABB), the oriented bounding box (OBB), the bounding sphere, the 14-dop, the 18-dop, the 26-dop, and the convex hull (CH). We also report the average (maximum) number of triangles for each bounding volume. For the AABB, sphere, and OBB, the maximum value is always identical to the average. Note that the maximum number of triangles for the convex hulls is often quite large.

driver), 8% (formula one), 7% (London), 14% (city), 6% (Manhattan), and 2% (angiography).

Although our frame rates increased for all but one model when using *k*-dops, the MCAD models produced slightly mixed results. The screw driver and formula one models were receptive to using *k*-dops, while the boom box model was not (26-dops saw a decrease of 2% with respect to frame rate). We noticed that the overall culling percentage did not increase signifi cantly for the boom box (for any bounding volume), as a result of the many axis-aligned components of the model. In general, however, the MCAD models that we have experimented with consist of many interior parts which are often occluded by outer cover components. The loosely fi tting bounding volumes (e.g. AABBs) used on the interior geometry often protrude through the outer case of the models, while the tighter *k*-dops do not (see Figure 3).

In the case of the boom box, since the interior components are well aligned with the component axes, even the AABBs do a good job approximating them. However, a benefi t of using *k*-dops is that you need not rely on the geometry of the model being aligned in such a manner for the bounding volumes to work well. To illustrate this point, we took the boom box model and rotated it 45 degrees around the vector ¡1,1,1¿ so that its components would no longer be aligned with the coordinate axes. As expected, the effi ciency of AABBs decreases, as can be seen in Table 3. In contrast to the previous test, bounding volumes less dependent upon the orientation of the model perform better than the fi xed AABBs; even the bounding sphere had a higher average culling percentage than the AABB. The convex hull continues to have the highest culling rate, followed by the 26-dop and (in this test) the OBB. Interesting, the OBB was able to cull nearly the same amount of geometry as the 26-dops and actually achieved a slightly higher frame rate. The point still remains that bounding volumes that are largely independent upon the orientation of the underlying geometry that they are approximating need not assume or rely on such an orientation in order to provide good approximations and fast frame rates.

BV	Culling %	fps
AABB	71.3	16.4
OBB	75.8	18.6
Sphere	72.4	15.7
14-dop	74.9	18.1
18-dop	75.1	18.1
26-dop	75.9	18.4
CH	78.7	17.4

Table 3: Occlusion culling statistics for the boom box after undergoing a 45 degree rotation about ¡1,1,1¿

The three architectural models all benefi ted from tighter bounding volumes with the increase in frame rate ranging from 6% (Manhattan) to 14% (City). In the City model, an artifi cial scene generated by a modeling system, each of the buildings was modeled as an individual object including the interior geometry. The other models had no object distinction, so interior geometry was present in only a few of the buildings, and the hierarchies created were not as tight fi tting as in the City model. Another signifi cant factor in the City model benefi ting to a larger extent is that the roofs of the building are not flat (as in the other models), so the tighter bounding volumes could approximate them better than the AABBs.

The angiography model, which was generated using Marching Cubes, was largely occluded (due to the endoscopic walkthrough) and each of the bounding volumes did well to detect this. The slight improvement in culling performance when using the tighter bounding *k*-dops resulted in a slight improvement in frame rate of 2%, compared to AABBs.

The slight culling and frame rate improvements that were

Model		AABB	14-dop	18-dop	26-dop
Screw driver	culling %	53.7	61.2	68.3	68.5
	fps	59.2	59.7	72.4	**84.6**
Boom box	culling %	75.3	75.6	75.8	75.9
	fps	21.0	19.3	19.2	**21.2**
Formula One	culling %	65.6	68.8	69.3	69.2
	fps	18.0	18.5	19.3	18.8
London	culling %	97.2	97.4	97.5	97.6
	fps	52.4	54.7	56.4	57.1
City	culling %	96.0	96.8	96.9	97.0
	fps	50.7	55.9	58.6	59.3
Manhattan	culling %	92.6	93.1	93.1	93.2
	fps	15.9	16.5	16.7	16.7
Angiography	culling %	97.0	97.1	97.1	97.2
	fps	57.3	60.0	60.2	60.5

Table 4: Occlusion culling statistics for the hierarchical culling test. The bold numbers indicate where this culling algorithm produced faster frame rates than the sequential culling algorithm.

achieved on the angiography model illustrate that the additional rendering complexity of the *k*-dops is negligible. This is the result of each occlusion culling query using the HP flag being equivalent to rendering roughly 190 triangles [35], which is much more than needed for a 26-dop. There is clearly a limit however, in how complex the bounding volumes can be. For example, each of the convex hulls for the boom box averaged 578 triangles and resulted in the convex hull actually being significantly slower than the AABBs. A similar result occured between OBBs and 26-dops when the boom box was rotated.

In general, the oriented bounding boxes provided little improvement in occlusion culling performance over AABBs. This is because their screen space coverage is almost identical (see Table 5). One of the most interesting things we learned was that the convex hulls were the best bounding volume in four examples for the occlusion culling algorithm (strictly in terms of frame rates). In the two MCAD models (screw driver and formula one), there were significant increases in culling performance as the bounding volumes became more complex. Since the convex hulls are the tightest bounding volume, their culling rates were the best. In the two architectural models in which the convex hull produced the best frame rates, there were only slight increases in culling percentages, but enough so that the overall frame rate was faster. In these cases, the average number of triangles of the convex hulls used is quite low (75 for City and 28 for Manhattan) so the improved culling percentage is achieved without any real overhead in terms of occlusion culling time.

Hierarchical Culling Queries As discussed in Section 4, we experimented with two culling algorithms: one in which we examined in depth order (sequentially) the leaf nodes of the bounding volume hierarchy, and another algorithm in which we performed culling queries hierarchically. As can be seen in Table 4, the hierarchical culling algorithm produced results that were close to the sequential tests, however, in only two of the cases did it actually produce faster frame rates (highlighted in bold text). The rationale for using hierarchical queries is that we may be able to prune a large portion of the tree away with a single culling query, however, as can be seen in these results, as well as those in [5], the hierarchical tests can result in significantly more culling queries being performed which in turn increases the overall running time of the algorithm. Overall, hierarchical culling queries realized no improvements compared to the sequential approach.

5.2 Approximation Quality

To better illustrate the improved approximation quality, we computed the total volume and surface area for all of the bounding volumes of the leaf nodes of our model hierarchies, and presented them in Table 5 (note the reordered columns). The bounding spheres are clearly the worst approximation, and the *k*-dops are better than everything else, with the natural exception being the convex hulls, the tightest of all (convex) bounding volumes. It is interesting to note that the total volume and surface area for a particular model mimic each other very well as we change bounding volumes: they are both monotonically increasing as we go from convex hulls, to 26-dops, to 18-dops, to 14-dops, to AABBs, to OBBs, and finally to spheres. The only exceptions to this occur between AABBs and OBBs and are highlighted in bold text. In comparing AABBs and OBBs, we see that these two bounding volumes are very comparable with respect to volume and surface area (and therefore rasterized screen space), which resulted in similar culling and frame rates (see above).

It remains an open issue which of these two metrics is better in the current scenario. While others have favored using the surface area [21], we have used the volume because there are examples of tighter (non-convex) bounding volumes that have larger surface areas than looser bounding volumes. This is not the case when using the volume criterion.

5.3 Bounding Volume Complexity

The complexity of the bounding volumes effects both the rendering and storage costs. As the rendering costs of the bounding volume are really encapsulated in the previous discussion on culling performance, we will focus here on the storage costs. The average and maximum bounding volume complexity are provided in Table 2. The clear winners here are the AABB and OBB, as they each require only 12 triangles. The bounding sphere must be tessellated (during preprocessing only) for the rendering-based, image-space culling algorithm: the more complex the tessellation, the more accurate it is, but the more expensive it is to store. For the experiments that we report here, we used a triangulated sphere with 320 triangles. The *k*-dops are quite simple to store. The 14-, 18-, and 26-dops each require at most 44, 60, and 92 triangles respectively. The convex hulls on the other hand, often required several hundred triangles each on average, and up to several thousand triangles each in the worst case.

Model		CH	26-dop	18-dop	14-dop	AABB	OBB	Sphere
Screw driver	volume	830K	1,033K	1,065K	1,204K	1,307K	1,364K	23M
	surface	190K	218K	223K	242K	**269K**	**268K**	866K
Boom box	volume	45M	52M	54M	57M	63M	71M	291M
	surface	2,581K	2,743K	2,893K	3,051	3,363K	3,568K	6,797K
Formula One	volume	204M	302M	311M	359M	443M	452M	4,043M
	surface	9M	11M	11M	12M	14M	14M	35M
London	volume	19M	21M	22M	23M	**27M**	**26M**	69M
	surface	1,401K	1,534K	1,590K	1,666K	1,930K	2,543K	3,713K
City	volume	231K	279K	306K	354K	**457K**	**408K**	995K
	surface	1,561K	1,782K	1,833K	2,028K	**2,618K**	**2,421K**	3,471K
Manhattan	volume	413M	525M	549M	572	676M	1,016M	3,426M
	surface	30M	33M	35M	35M	40M	51M	83M
Angiography	volume	8M	13M	14M	16M	26M	33M	74M
	surface	1,360K	1,723K	1,866K	1,952K	2,829K	3,377K	4,473K

Table 5: Total volume and surface area for all of the bounding volumes of the hierarchy leaf nodes for our data models. For each model, the volume and surface area are monotonically increasing across bounding volumes: convex hulls, 26-dops, 18-dops, 14-dops, AABBs, OBBs, spheres. The only exceptions to this are between the AABBs and OBBs and are highlighted in bold.

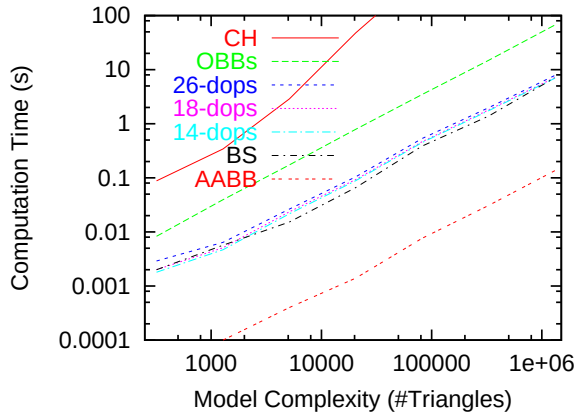


Figure 5: Computation times: For each bounding volume, we recorded how long it took to compute as the number of vertices increased (note the log-log plot).

5.4 Computational Expense

Figure 5 illustrates the computational expense of computing the various bounding volumes for increasingly complex geometry (note the log-log scale). We varied the geometry to consist of only a few vertices up to one million vertices to see how the construction algorithms scaled. Not surprisingly, the AABBs were the easiest to compute, followed by the bounding spheres. As the value of k increases, the cost of computing our k-dops also increases, but the overall cost of building k-dops is clearly linear in the number of vertices. This is also true for OBBs, although they are more expensive to compute than any of our k-dops. Convex hulls are the most complex to compute, as shown by the plot.

For our real-world data, we found that in practice the same relationship held for computing the various BVs: AABBs and spheres were the fastest to compute, followed by our k-dops, OBBs, and finally convex hulls (which were 5-7 times slower to compute than the k-dops (for our models).

6 CONCLUSIONS

We have introduced the use of complex bounding volumes such as k-dops for the purpose of occlusion culling. In comparison to the volumes that have typically been used for such purposes (AABBs), k-dops are more effective at culling occluded geometry and result in faster interactive rendering. These improvements come at the cost of a slightly more complex storage requirement for each bounding volume. In comparison to bounding spheres and OBBs, our k-dops are again more efficient at culling and rendering and slightly more complex to store. In a few examples, using convex hulls for bounding volumes resulted in the fastest rendering rates. Convex hulls do carry an additional overhead as opposed to k-dops: they are more costly to compute, they are more complicated to compute due to degeneracies, and they require significantly more storage than do k-dops.

For our occlusion culling algorithm, the best choice of k is 26. The additional complexity of the 26-dop provides better approximations than the other k-dops, but the additional triangles that need to be rendered do not slow down our frame rates. There is a limit in the number of triangles that can be used for the tight approximations; in some cases, the convex hulls are actually slower than AABBs.

We conclude that in using 26-dops, one can avoid the complications of trying to determine which bounding volume will work the best for their particular data model. We have shown that our k-dops provide a well-distributed sampling of bounding planes and therefore do not rely on the geometry being oriented in any particular way. In our experiments, the 26-dop produced the best rendering frame rates in many of our tests. In those instances where it did not provide the best frame rate, its results were very close to the best.

Recent trends in graphics hardware allow the dynamic modification of object geometry (e.g. vertex shaders, articulated motion). While our current approach requires a re-computation of the bounding volume, only minor incremental updates of the bounding volume geometry might be necessary. Consequently, as future work, we would like to investigate how these incremental improvements of the bounding volumes for dynamic objects can be achieved.

ACKNOWLEDGEMENTS

This work is supported by the Large Model Visualization project of the Workstation Systems Lab, Ft. Collins, CO of the Hewlett-

Figure 6: A bird's eye view of the Manhattan dataset.

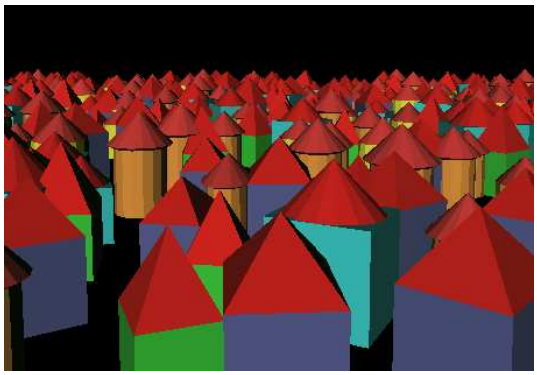


Figure 8: City: An artificial scene generated by a modeling system where all buildings contain interior geometry.

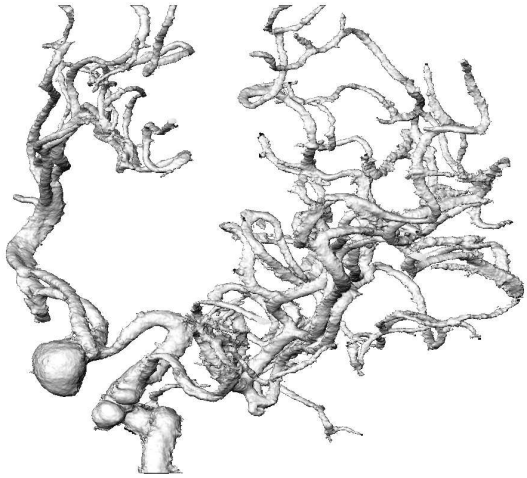


Figure 7: Angiography: An extracted isosurface of an arterial blood vessel from a rotational angiography scan of a human head.

Packard Company and by DFG project CatTrain. The MCAD datasets are courtesy of IBM, Engineering Animation Inc. and Hewlett-Packard Company. We would also like to thanks Tommer Leyvand and Daniel Cohen-Or of the Tel Aviv University for providing us with their fancy city-generator software.

REFERENCES

[1] P. K. Agarwal, M. de Berg, M. Hammar, H. J. Haverkort, and J. Gudmundsson. Box-Trees and R-Trees with Near-Optimal Query Time. *Discrete and Computational Geometry*, 28:291–312, 2002.

[2] G. Barequet, B. Chazelle, L. Guibas, J. Mitchell, and A. Tal. BOXTREE: A Hierarchical Representation for Surfaces in 3D. In *Proc. of Eurographics*, pages 387–396, 1996.

[3] D. Bartz, M. Meißner, and T. H¨uttner. OpenGL-assisted Occlusion Culling of Large Polygonal Models. *Computers and Graphics - Special Issue on Visibility - Techniques and Applications*, 23(5):667–679, 1999.

[4] 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.

[5] 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 Parallel and Large-Data Visualization and Graphics (PVG 01)*, pages 129–134, 2001.

[6] G. Bradshaw and C. O'Sullivan. Sphere-tree Construction Using Dynamic Medial Axis Approximation. In *Proc. of ACM SIGGRAPH Symposium on Computer Animation*, pages 33–40, 2002.

[7] D. Cohen-Or, Y. Chrysanthou, C. T. Silva, and F. Durand. A Survey of Visibility for Walkthrough Applications. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):412–431, July 2003.

[8] W. T. Correa, J. T. Klosowski, and C. T. Silva. Visibility-based prefetching for interactive out-of-core rendering. In *Proc. of IEEE Symposium on Parallel and Large-Data Visualization and Graphics (PVG 03)*, pages 1–8, 2003.

[9] A. Crosnier and J.R. Rossignac. Tribox Bounds for Three-Dimensional Objects. *Computers & Graphics*, 23(3):429–437, 1999.

[10] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 1997. ISBN 3-540-61270-X.

[11] F. Durand, G. Drettakis, and C. Puech. The Visibility Skeleton: A Powerful And Effi cient Multi-Purpose Global Visibility Tool. In *Proc. of ACM SIGGRAPH*, pages 89–100, 1997.

[12] J. Goldsmith and J. Salmon. Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics and Applications*, 7:14–20, 1987.

[13] S. Gottschalk, M. Lin, and D. Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. In *Proc. of ACM SIGGRAPH*, pages 171–180, 1996.

[14] N. Govindaraju, A. Sud, S.-E. Yoon, and D. Manocha. Interactive Visibility Culling in Complex Environments Using Occlusion-Switches. In *Proceedings of the 2003 Symposium on Interactive 3D graphics*, pages 103–112. ACM Press, 2003.

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

[16] E. A. Haines and J. R. Wallace. Shaft Culling for Effi cient Ray-Traced Radiosity. In *Proc. of the Second Eurographics Workshop on Rendering*, 1994.

[17] T. He. Fast Collision Detection Using QuOSPO Trees. In *Proc. of ACM Symposium on Interactive 3D Graphics*, pages 55–62, 1999.

[18] M. Held, J. Klosowski, and J. Mitchell. Real-time Collision Detection for Motion Simulation Within Complex Environment. In *Visual Proc. of ACM SIGGRAPH*, page 151, 1996.

[19] K. Hillesland, B. Salomon, A. Lastra, and D. Manocha. Fast and Simple Occlusion Culling Using Hardware-based Depth Queries. Technical Report TR02-039, Technical Report TR02-039, Department of Computer Science, University of North Carolina, 2002.

[20] P. Hubbard. Approximating Polyhedra with Spheres for Time-Critical Collision Detection. *ACM Transactions on Graphics*, 15(3):179–210, 1996.

[21] A. Iones, S. Zhukov, and A. Krupkin. On Optimality of OBBs for Visibility Tests for Frustum Culling, Ray Shooting and Collision Detection. In *Proc. of Computer Graphics International (CGI 98)*, pages 256–263, 1998.

[22] T. Kay and J. Kajiya. Ray Tracing Complex Scenes. In *Proc. of ACM SIGGRAPH*, pages 269–278, 1986.

[23] J. Klosowski, M.Held, J. Mitchell, H. Sowizral, and K. Zikan. Effi - cient Collision Detection Using Bounding Volume Hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[24] S. Krishnan, M. Gopi, M. Lin, D. Manocha, and A. Pattekar. Rapid and Accurate Contact Determination between Spline Models using ShellTrees. *Computer Graphics Forum (Proc. of Eurographics)*, 17(3):315–326, 1998.

[25] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. Fast Distance Queries using Rectangular Swept Sphere Volumes. In *Proc. IEEE International Conference on Robotics and Automation*, 2000.

[26] T. Leyvand, O. Sorkine, and D. Cohen-Or. Ray Space Factorization for From-Region Visibility. *ACM Transactions on Graphics (Proc. of ACM SIGGRAPH)*, 22(3):595–604, 2003.

[27] M. Meißner, D. Bartz, T. H¨uttner, G. M¨uller, and J. Einighammer. Generation of Decomposition Hierarchies for Effi cient Occlusion Culling of Large Polygonal Models. In *Proc. of Vision, Modeling, and Visualization*, pages 225–232, 2001.

[28] G. M¨uller and D. Fellner. Hybrid Scene Structuring with Application to Ray Tracing. In *Proc. of ICVC*, pages 19–26, 1999.

[29] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1993. ISBN 0-521-44034-3.

[30] C. O'Sullivan and J. Dingliana. Realtime Collision Detection and Response using Spheretrees. In *Proc. of Spring Conference on Computer Graphics*, pages 83–92, 1999.

[31] I. J. Palmer and R. L. Grimsdale. Collision Detection for Animation Using Sphere-Trees. *Computer Graphics Forum*, 14(2):105–116, June 1995.

[32] S. Quinlan. Effcient Distance Computation between Non-Convex Objects. In *Proc. International Conference on Robotics and Automation*, pages 3324–3329, 1994.

[33] S. Rubin and T. Whitted. A 3-Dimensional Representation for Fast Rendering of Complex Scenes. In *Proc. of ACM SIGGRAPH*, pages 11–116, 1980.

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

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

[36] L. Sobierajski-Avila and W. Schroeder. Interactive Visualization of Aircraft and Power Generation Engines. In *Proc. of IEEE Visualization*, pages 483–486, 1997.

[37] D. Staneker, D. Bartz, and M. Meißner. Improving Occlusion Query Effi ciency with Occupancy Maps. In *Proc. of IEEE Symposium on Parallel and Large-Data Visualization and Graphics (PVG 03)*, pages 111–118, 2003.

[38] S. Suri, P. M. Hubbard, and John F. Hughes. Analyzing Bounding Boxes for Object Intersection. *ACM Transactions on Graphics*, 18(3):257–277, July 1999.

[39] G. van den Bergen. Effi cient Collision Detection of Complex Deformable Models using AABB Trees. *Journal of Graphics Tools*, 3(5):1–13, 1997.

[40] H. Weghorst, G. Hooper, and D. Greenberg. Improved Computational Methods for Ray Tracing. *ACM Transactions on Graphics*, 3(1):52–69, 1984.

[41] M. Wimmer, J. Bittner, H. Piringer, and W. Purgathofer. Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful. *Computer Graphics Forum (Proc. of Eurographics)*, 23(3), 2004.

[42] G. Zachmann. Rapid Collision Detection by Dynamically Aligned DOP-Trees. In *Proc. of IEEE Symposium on Virtual Reality (VRAIS)*, 1998.

[43] 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.

[44] Y. Zhou and S. Suri. Analysis of a Bounding Box Heuristic for Object Intersection. *Journal of the ACM*, 46(6):833–857, November 1999.

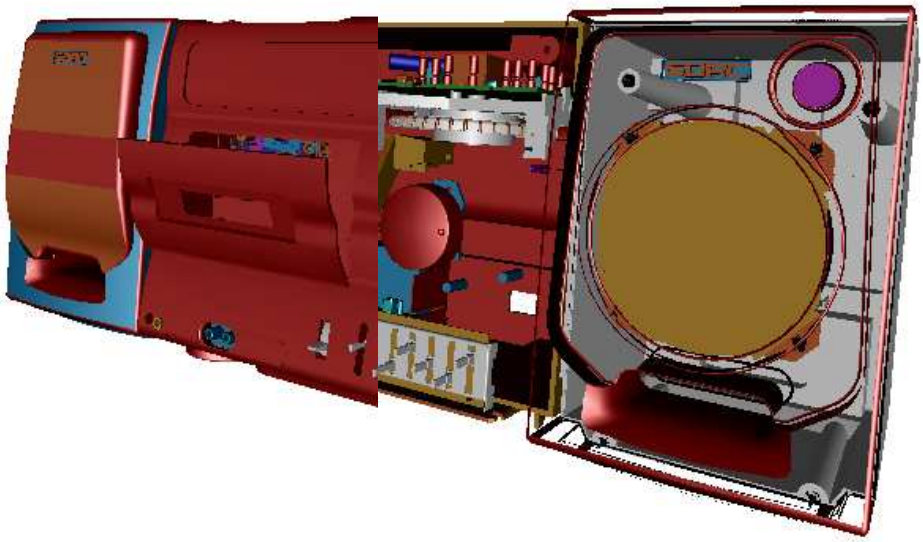


Figure 9: Boom Box: Cover parts are removed on the right side of the clipping plane.

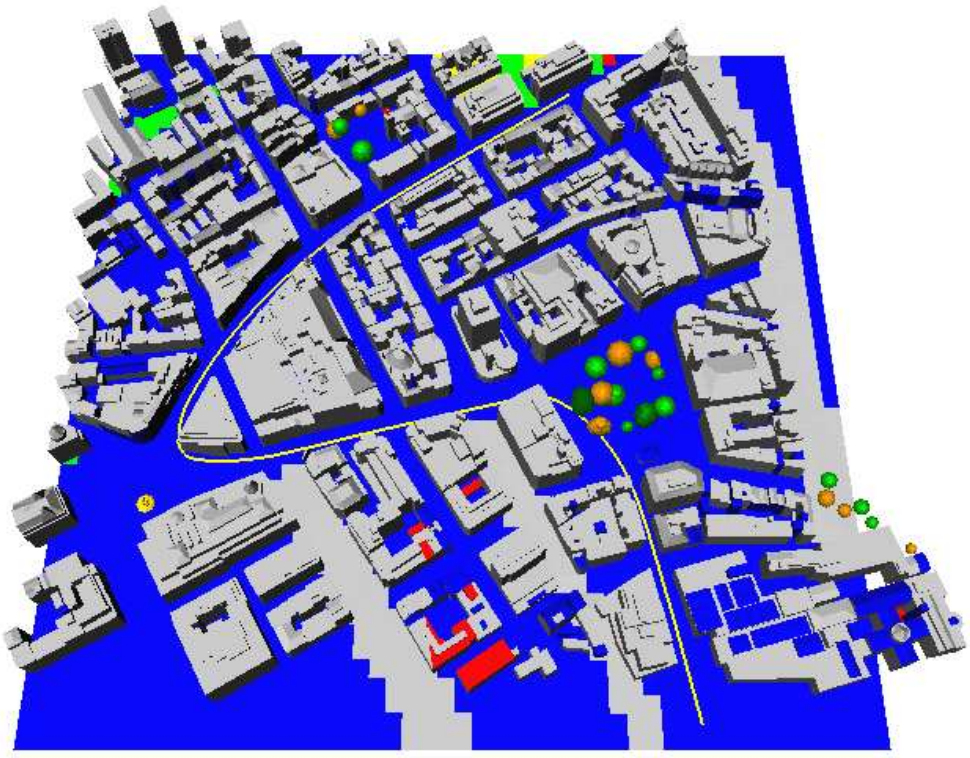


Figure 10: London: A model of the Westend of London. The yellow line shows the path (from the lower right corner to the upper right corner) used for the measurements in the paper.