Optimized View Frustum Culling Algorithms for Bounding Boxes

Ulf Assarsson and Tomas Möller
Department of Computer Engineering
Chalmers University of Technology, Sweden
Accepted for publication in *journals of graphics tools*

March 1999, revised February 2000

Abstract

This paper presents optimizations for faster view frustum culling (VFC) for axis aligned bounding box (AABB) and oriented bounding box (OBB) hierarchies. We exploit frame-to-frame coherency by caching and by comparing against previous distances and rotation angles. By using an octant test, we potentially halve the number of plane tests needed, and we also evaluate masking, which is a well-known technique. The optimizations can be used for arbitrary bounding volumes, but we only present results for AABBs and OBBs. In particular, we provide solutions which is $2-11$ times faster than other VFC algorithms for AABBs and OBBs, depending on the circumstances.

1 Introduction

Bounding volume hierarchies are commonly used to speed up the rendering of a scene by using a view frustum culling (VFC) algorithm on the hierarchy [Clark76]. Each node in the hierarchy has a bounding volume (BV) that encloses a part of the scene. The hierarchy is traversed from the root, and if a BV is found to be outside the frustum during the traversal, then the contents of that BV need not be processed further, and performance is gained. Reducing the time for view frustum culling will increase performance of a single processor system and free processor time to other tasks. With view frustum culling taking 6 ms before speedup and 1.2 ms after speedup[1] and with 33 frames per second (30 ms/frame, no synchronization to monitor frequency), the view frustum culling goes from taking 20% of total execution time to only 4%, thus saving 16%.

We present several optimizations for culling axis-aligned bounding boxes (AABBs) and oriented bounding boxes (OBBs) against a frustum used for perspective viewing. Frame-to-frame coherency is exploited by caching and by comparing against previous distances and rotation angles. An octant test is introduced, which potentially halves the number of plane tests needed, and we also evaluate masking [Bishop98]. All these optimizations can be used for arbitrary bounding volumes which we show in our technical report [Assarsson99], where we also investigate bounding spheres, with speedups[2] from 1.2 up to 1.4 times. That report also provides more details on the results for AABBs.

2 Related Work

When reviewing existing view frustum culling algorithms [Bishop98, DirectModel, Hoff96a, Hoff96b, Hoff97, Green95, Greene94], we found that there are two common ways to approach the view frustum culling problem.

[1] These are our figures for path 1, model 3 with the plane-coherency and octant test optimization (see section 4).
[2] In this paper we define speedup as *time1/time2*, which means that a speedup of 1.0 is no speedup at all.

One approach is to perspective transform the bounding volume of a node to be tested and the view frustum, to the perspective coordinate system and perform the testing there. This is popular when the bounding volumes are axis aligned bounding boxes, since this results in testing two AABBs (if the perspective transformed AABB is bounded by a minimal AABB, see figure 1) against each other, which can be done with only six comparisons after the transformation has been done. The disadvantage is that we must transform the bounding volume to the perspective coordinate system. This means that all eight vertices of the AABB must be multiplied with the view- and projection (perspective) matrix, which includes at least 72 multiplications. The view frustum culler in DirectModel is based on this method [DirectModel].

The other approach is to test the bounding volume against the six planes defining the view frustum [Hoff96a, Hoff96b, Hoff97, Green95, Greene94]. This is also the approach that we choose. This has the advantage that trivial rejection or acceptance tests can be made [Greene94, Green95]. Should these fast tests fail, traditionally the more expensive intersection tests necessary to find exact intersection between a box and a frustum are computed. Instead of that, we recursively continue testing the planes of the sub-boxes, in order to get higher performance.

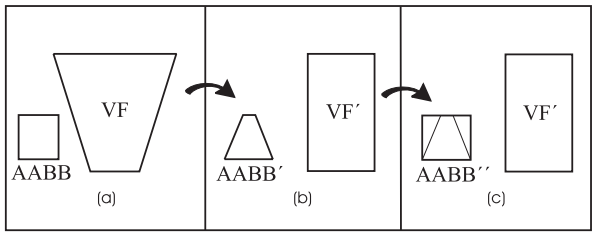


Figure 1: (a) View frustum and an AABB. (b) The same view frustum and AABB perspective transformed. (c) The perspective transformed AABB is bounded by a minimal AABB in the perspective coordinate system, which is tested for intersection with the view frustum.

VFCs can be based on BSP-trees to gain speed [Chin95] with the drawback that BSP-trees only represent static environments.

Slater et al. [Slater97] present a VFC based on a probabilistic caching scheme using ellipsoids, which according to their results provides comparable speedups to our methods[3]. It may, however, erroneously cull objects that should be visible.

3 Frustum-AABB/OBB Intersection

Our algorithms for view frustum culling of hierarchies with AABBs or OBBs are all based on a basic intersection test. Four different optimizations can be added on top of this, so we have partitioned our main algorithm into five steps:

- *the basic intersection test*, sped up to test just two box corners (section 3.1)
- *the plane-coherency test*, taking advantage of frame-to-frame coherency by using previous test results (section 3.2)
- *the octant test*, allowing half of the plane/bounding-volume tests to be avoided while using symmetric frustums (section 3.3)
- *masking*, in which bounding-box/plane test results are passed on and reused by children bounding-boxes (section 3.4)

[3]The most fair way is probably to compare their reported speedup of about 1.7 with the speedups provided by the combinations of our optimizations compared to our basic intersection test. See section 4 or our technical report for more details [Assarsson99].

- *TR*[4] *coherency test*, which allows reuse of previous frame test results when the view changes in limited ways (section 3.5)

The optimizations can be utilized in a VFC independently of each other. That is, we can choose and combine the steps anyway we like. A view frustum (VF) is defined by six planes:

$$\pi_i : \mathbf{n}_i \cdot \mathbf{x} + d_i = 0 \tag{1}$$

$i = 0 \ldots 5$, where $\mathbf{n}_i$ is the normal and d_i is the offset of plane π_i, and $\mathbf{x}$ is an arbitrary point on the plane. We say that a point $\mathbf{x}$ is outside a plane π_i if $\mathbf{n}_i \cdot \mathbf{x} + d_i > 0$. If the point is inside all planes then the point is inside the view frustum.

3.1 Basic Intersection Test

An exact intersection test between a box and a frustum may be expensive to compute. Therefore we use the following strategy: for each frustum plane, test if the box is outside, inside or intersecting the plane. If outside, terminate and report `outside`. If the box is inside all planes, return `inside`, else return `intersecting`. Note that this is an approximation; sometimes when we report `intersecting` the box may be outside (see figure 2). For intersecting boxes we continue the hierarchy traversal, so the end result is still correct. If a node is a leaf

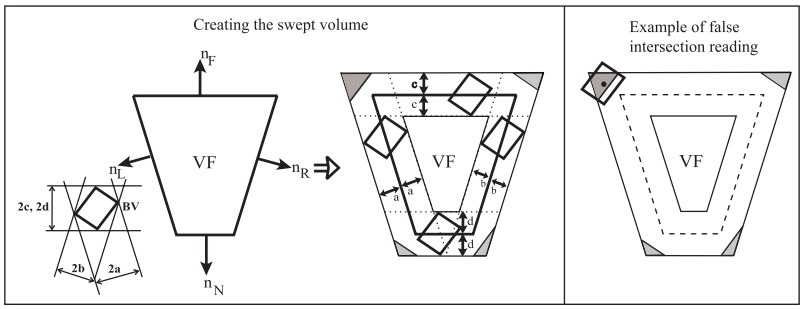


Figure 2: If the box center is within the grayed sections, our approximation reports `intersection` even though the box is completely outside the view frustum (VF). This is explained by the following; an intersection test can be conducted by testing the box center against the volume obtained by sweeping the box along the planes of the VF, keeping the box center on the planes. Our approximation is the same as testing the box center against the resulting inner and outer planes parallel with, and at different offsets from the VF planes. In three dimensions the corresponding gray sections are located at the corners and the edges. If the center is outside the planes, the box is outside. If the center is between the planes, we have intersection, and if it is inside, the box is inside. See our technical report for details and a generalization to any bounding volumes [Assarsson99].

then we can choose to do more accurate tests. However, since we found no observable penalty in the rendering performance, we skipped those tests.

We first present how to determine if a box intersect a plane. A naive method is to test all eight points against the plane. However, only two points need to be tested [Haines94, Greene94, Green95], namely those that forms the diagonal that is most closely aligned with the plane's normal, and that passes through the box center. These points are called the *n- and p-vertices*, where the p-vertex has a greater signed distance from the plane than the n-vertex.

First, the n-vertex is inserted in the plane equation. If the n-vertex is outside then the box is outside (both the plane and the frustum) and the test terminates. Then the p-vertex is tested, and if the p-vertex is inside then the the box is inside the plane. Otherwise the box intersects the plane. This is illustrated in figure 3. Finding the two *n- and p-vertices* can be done in 9 multiplications and 3 comparisons by projecting the normal of the view frustum plane

[4]TR stands for Translation and Rotation

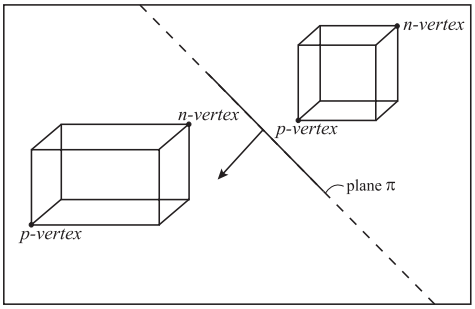


Figure 3: The negative far point (n-vertex) and positive far point (p-vertex) of a bounding box corresponding to plane π and its normal.

```
bool intersect = false
for i in [all view frustum planes π_i] do
     v_n ← negative far point (n-vertex) in world
           coordinates of box relative to π_i
     a ← v_n · n_i + d_i
     if a > 0 then return Outside
     v_p ← positive far point (p-vertex) in world
           coordinates of box relative to π_i
     b ← v_p · n_i + d_i
     if b > 0 then intersect = true
end loop
if intersect then return Intersecting
else return Inside
```

Figure 4: Pseudo code of general algorithm for culling AABBs or OBBs

on to the box's axes and test the signs of the x-, y- and z-components of the projection. Since all AABBs are given in the world coordinate system (aligned to the world x-, y- and z-axes) and we transform all view frustum plane equations (i.e. plane normals and offsets) to world coordinates at the beginning of each frame, we have the AABBs and the normal of the planes in the same coordinate system, which makes a projection unnecessary. We can use the signs of the x-, y- and z-components of the plane normal immediately, leaving us with only three comparisons. If we create a bitfield of the signs, letting for instance a negative sign be represented by a '0' and a positive sign be represented by a '1', we can use this bitfield to get the p-vertex from a Look Up Table (LUT). In this way we avoid the conditional branches caused by if-statements, which can lead to expensive processor pipeline prediction misses. This idea is used by Donovan et al. to accelerate clipping [Donovan94]. If we order the LUT properly, we can invert the bitfield to get the n-vertex.

If we are going to test multiple AABBs against the view frustum (which generally is the case) and since all AABBs have the same orientation, it is a good idea to precompute the bitfields (indices to the n- and p-vertices) for each view frustum plane once each frame [Haines94].

A listing of the algorithm for testing a box against a frustum is given in figure 4.

3.2 The Plane-Coherency Test

The goal of this test is to exploit temporal coherence. Assume that a BV of a node was outside one of the view frustum planes last time it was tested for intersection (previous frame). For small movements of the view frustum there is a high probability that the node is outside the same plane this time, which means that we should start testing

against that plane hoping for fast rejection of the BV. If the BV was outside a plane last frame, then an index to this plane is cached in the BV structure. For each intersection test, we start testing against that plane and test the others afterwards if necessary.

We test the planes in the order: $left$, $right$, $near$, far, up, and $down$. No experiments in finding an "optimal" order has been conducted.

3.3 The Octant Test

Assume that we split the view frustum in half along each axes, resulting in eight parts, like the first subdivision of an octree. We call each part an *octant* of the view frustum.

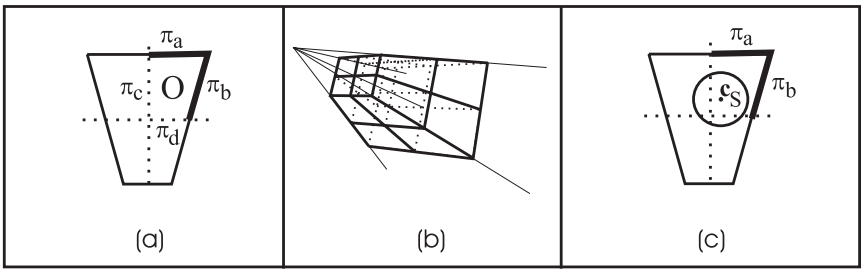


Figure 5: (a) 2D-view of a symmetric view frustum divided in half along each axis. π_a and π_b are the outer planes of octant O. π_c and π_d are the inner planes. (b) View frustum divided in octants. (c) For a symmetrical VF it is sufficent to test for intersection against the outer planes of the octant in which the center ($\mathbf{c}_S$) of the bounding sphere (of the box) lies.

If we have a symmetrical view frustum, which is the most common case (a CAVE [CrusNeira93] is one exception), and a bounding sphere, it is sufficient to test for culling against the outer three planes of the octant in which the center of the bounding sphere lies (see figure 5). This means that if the bounding sphere is inside the three nearest (outer) planes, it must also be inside all planes of the view frustum. If it is outside any of the planes, we know it is totally outside the view frustum, and otherwise it is intersecting.

This can be extended to general bounding volumes; see our report [Assarsson99]. To be able to use the octant test for boxes, the distance from the box center to a box corner must be smaller than the smallest distance from the view frustum center[5] to the frustum planes (see figure 6). This is true, since an arbitrary BV cannot intersect the planes of another octant without intersecting the planes of the selected octant, if the above holds. The distance between the center of the view frustum and its nearest plane can be precomputed once for each frame.

The cost of locating the octant was found to be approximately equal to one plane/box test.

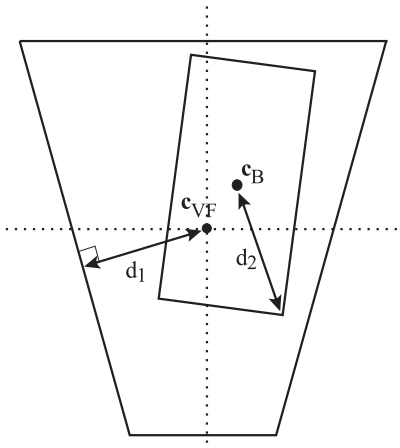
3.4 Masking

Assume that a node's BV is completely inside one of the planes of the view frustum. Then, as pointed out by Bishop et al. [Bishop98], we know that the BVs of the node's children also lie completely inside that plane, and that plane can be eliminated (masked off) from further testing in the subtree of the node.

When traversing the scene graph, a mask (implemented as a bitfield) is sent from the parent to the children. This mask is used to store a bit for each frustum plane, which indicates whether the parent is inside that plane. Before each plane test, we check if that plane is masked off or not. In this way, plane tests can be avoided if the parent of a node is inside one or more planes.

If we can eliminate the low-level polygon clipping against the window border corresponding to a view frustum plane, for all nodes that are totally inside that plane, then maybe masking could pay off a lot [Bishop98]. Low

[5]The center of the frustum is the sum of the eight frustum corners divided by eight.



$\mathbf{c}_B$ = center of the box
$\mathbf{c}_{VF}$ = center of the view frustum
d_1 = distance between the view frustum center and its closest plane(s)
d_2 = distance from the box center to a box corner

Figure 6: If $d_2 <= d_1$ we can use the octant test for bounding boxes as well.

level clipping of polygons is usually done against each view frustum plane for each polygon sent for rendering. For nodes that are totally inside the view frustum, all clipping could be disabled and then potentially provide speedups.

3.5 The TR Coherency Test

TR coherency stands for translation and rotation coherency. In this optimization step, we exploit the fact that when navigating in a three dimensional world, you sometimes only rotate around one axis or translate, or you might even be standing still. For objects that have not moved since the last frame the following applies:

1. If, for instance, a BV was outside the left plane of the view frustum last frame, and the view frustum only has rotated to the right since then, we know that the BV still is outside the left plane (assuming that the rotation is smaller than $180^\circ - angle\ between\ left\ and\ right\ plane$). In general this means that if only view frustum rotations have been done around either the x-axis, y-axis or the z-axis of the view frustum since last culling invocation, we can return `outside` for BVs if they were outside the plane last frame and if the distance to the plane must have increased (see figure 7a).

2. If the view frustum only has done a pure translation since last frame, the distances from all BVs to the same view frustum plane have increased or decreased by the same fixed amount Δd (see figure 7b). This Δd is possible to precompute once for all intersection tests against the corresponding plane. If only a translation (in any direction) has been done since last view frustum culling invocation, we precompute Δd_i for each view frustum plane π_i by projecting the translation on the normal of the planes. For each BV and view frustum plane to be tested, we compare the corresponding Δd_i with the distance between the BV and the plane last frame.

For (1) we precompute the plane that can use this optimization (if any), and for each BV which was outside this plane last frame, we return `outside`. Let us assume the view frustum axes are arranged according to figure 7c. If the view frustum, since last frame, has done a pure rotation around the y-axis in the positive direction, we can do quick rejection against the right plane. If instead the rotation was negative, we can do quick rejection against the

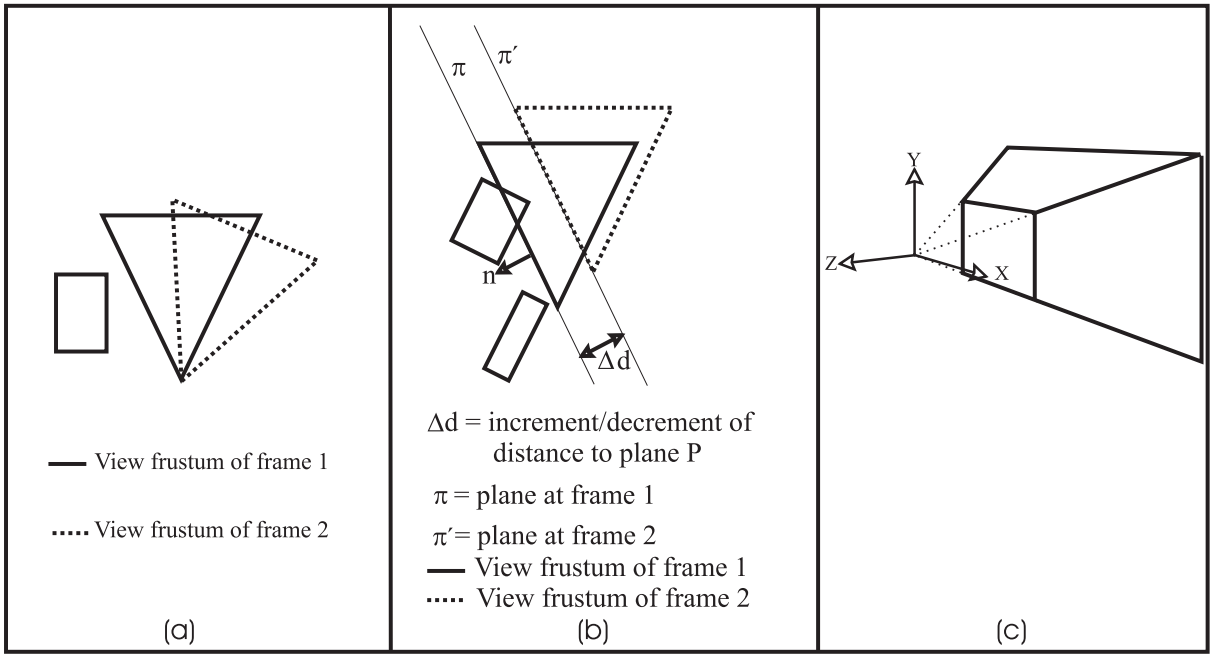


Figure 7: (a) Rotations of the view frustum. If the BV of a non-moving object was outside the view frustum at frame 1, we know that, because of the direction of rotation, it is also outside in frame 2. (b) Translations of the view frustum. (c) A view frustum and its frustum coordinate axes.

left plane. We have to keep track of the accumulated rotations to be able to invalidate any quick rejections when the total rotation around the axis exceeds $180^o - angle\ between\ left\ and\ right\ plane$. The x-axis and the up- and down planes are treated similarly. If rotations only occured around the z-axis, objects outside the near- and far plane will remain outside.

For (2) we have to add members to the BV structure holding the distances from the BV to the different planes of the view frustum, and we must also add a member indicating whether or not the BV was explicitly tested last time (otherwise the interesting distances is not calculated, and we must perform our test with another method).

4 Results

Each optimization and combinations of optimizations have been thoroughly tested to determine whether its possibly introduced overhead has paid off in shorter average execution times. The presented figures are speedups of the VFC-algorithms - not of total rendering time - and are compared against a view frustum culler testing AABBs in the perspective coordinate system (see section 2).

The implementation was done on a double PentiumII 200 MHz with 128 Mb RAM and were compared with the VFC in DirectModel on the same machine. All algorithms were tested on three virtual environments:

- Model 1: a car factory shop floor (184,000 polygons, 3800 graph nodes)
- Model 2: a factory cell (167,000 polygons, 188 graph nodes)
- Model 3: a factory shop floor (52,000 polygons, 1274 graph nodes)

We used four paths in our tests. For each path, about a million box/intersection tests were computed.

path	1	1	1	2	2	2	3	3	3	4	4	4
model	*1*	*2*	*3*	*1*	*2*	*3*	*1*	*2*	*3*	*1*	*2*	*3*
only Basic intersection test	2.8	1.9	3.9	2.2	2.0	3.1	3.9	2.5	4.3	3.1	2.2	3.7
Plane-coherency + octant test	**4.0**	**2.4**	**5.1**	**2.8**	**2.6**	**3.9**	4.8	**3.5**	**5.6**	3.3	3.0	5.1
Plane-coherency + TR coherency	3.8	2.0	4.0	2.5	2.2	3.0	5.0	2.8	4.4	**8.3**	3.1	**11.0**
Plane-coherency + octant test + TR coherency	3.7	2.2	4.5	2.6	2.4	3.6	**5.1**	3.0	4.8	8.0	**3.3**	9.0

Table 1: Speedup for the most promising combinations of optimizations, except for the 'only Basic intersection test'-row which is there for comparison issues. The figures are speedup compared to a view frustum culler testing AABBs in the perspective coordinate system (see section 2). The speedup figure of the best algorithm in each test case is marked with bold text. Each figure corresponds to a path, model and an algorithm.

- Path 1: sampled from a user navigating through our scenes
- Path 2: constructed with both a translation and a rotation each frame
- Path 3: pure rotations only
- Path 4: pure translations only

Table 1 presents the results of the most promising optimizations. For statistical data about other combinations, see our report [Assarsson99].

The speedup numbers were obtained using AABBs. For OBBs we did not create any separate OBB-tree. Instead we treated the AABBs in the AABB-hierarchy as OBBs, i.e added the necessary 9 multiplications and 6 additions in each intersection test and continued comparing against the AABB-algorithm of DirectModel. The penalty for OBBs showed to be about 10% more computation time for all test cases. Since OBBs provide better fits, they might give better overall performance.

If we for AABBs precompute the indices to the n- and p-vertices once each frame, instead of calculating them for each box (see section 3.1), an additional speedup of $5 - 10\%$ will be achieved. This optimization cannot be used for OBBs.

The plane-coherency + octant test is best in all test cases except for pure translations, where the plane-coherency + TR coherency test is superior. For symmetrical frustums (which is most common except for CAVEs [CrusNeira93]) we recommend the plane-coherency + octant test, and if we expect many pure translations also the TR coherency test to get the best of both worlds.

For asymmetric frustums, we recommend that the plane-coherency test and the TR coherency test are combined and used[6].

Masking was not found to be competitive with the algorithms above.

Our recommended combinations of optimizations boost the basic intersection test up to 3.0 times with an average of 1.4 times.

For individual intersection tests between a BV and the view frustum, the AABB implementation of Direct-Model was sometimes faster than our implementations. This occurred in average for $\approx 0.2\%$ of all intersection tests, which means that for $\approx 99.8\%$ of all cases, our algorithms were faster. This figure is based on timing the intersection tests with the CPU clock, which means that anomalies due to cache misses and page faults are included.

[6]For asymmetric frustums the octant test cannot be used.

5 Discussion

All our optimizations, including the basic intersection test, can be used on any kind of bounding volume, but we have only gathered statistics for AABBs, OBBs and bounding spheres. Since the sphere/plane test is so short, there was little gain in using our optimizations. For details, see our report [Assarsson99].

We have not made use of the fact that in general, the near clip plane and far clip plane are parallel, in any other cases than for the *octant test* and the *TR coherency test.* We could add this to our *basic intersection test* as well. We should then treat the near- and far clip planes as a pair of parallel planes instead of two individual, saving at least 6 multiplications and 2 evaluations of the *n- and p-vertices.* The reason why we did not include this is that we did not find an easy way to insert this into the algorithm, without slowing down other parts or make the code ugly.

If we find that the bounding volume is neither completely outside any plane, nor completely inside all planes, we might want to store one of the intersecting planes so that we can start checking against that plane in the next round, hoping that the bounding volume would have moved outside the plane and the view frustum.

It would also be interesting to modify our algorithm to handle k-DOPs[7] [Klosowski97] as bounding volumes. For DOPs we should probably take advantage of that they consist of pairs of parallel planes. Finding the *n- and p-vertices* or maximum extension in a specific direction from the center of a DOP is not trivial. Look up tables could perhaps be used in some cases, or we might have to approach the problem in a totally different way.

Since the difference in cost of using AABBs and OBBs is small, it would be interesting to investigate whether OBB hierarchies are faster than AABB hierarchies.

6 Acknowledgements

We would like to thank professor Per Stenström for his extensive guidance in improving the quality of this paper. Thanks to Lars Östlund, manager of research and development at Digital Plant Technologies AB, ABB, for the financial support. We also thank Eric Haines for pointing us at his and Wallace's paper [Haines94].

References

[Assarsson99] Ulf Assarsson and Tomas Möller, "Optimized View Frustum Culling Algorithms", Technical Report 99-3, Department of Computer Engineering, Chalmers University of Technology, *http://www.ce.chalmers.se/staff/uffe/* March 1999.

[Bishop98] Lars Bishop, Dave Eberly, Turner Whitted, Mark Finch, Michael Shantz, "Designing a PC Game Engine", *Computer Graphics in Entertainment*, pp. 46–53, January/february 1998.

[Chin95] Norman Chin, "A Walk through BSP Trees", *Graphics Gems V, Heckbert*, pp. 121–138, 1995.

[Clark76] James H. Clark, "Hierarchical Geometric Models for Visible Surface Algorithm", *Communications of the ACM*, vol. 19, no. 10, pp. 547–554, October 1976.

[CrusNeira93] Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, "Surround-screen Projection-based Virtual Reality: The Design and Implementation of the CAVE", *Computer Graphics (SIGGRAPH '93 Proceedings)*, pp 135-142, volume 27, aug, 1993.

[Donovan94] Walt Donovan, Tim van Hook, "Direct Outcode Calculation for Faster Clip Testing", *Graphics Gems IV, Heckbert*, pp. 125–131, 1994.

[DirectModel] *DirectModel 1.0 Specification*, Hewlett Packard Company, Corvalis, 1998

[7] A k-DOP (discrete oriented polytope) is made up of k pairs of parallel planes, the intersection of which forms a bounding volume. A bounding box can be thought of as a k-DOP where k is three and all planes are orthogonal.

[Greene94] Ned Greene, “Detecting Intersection of a Rectangular Solid and a Convex Polyhedron”, *Graphics Gems IV, Heckbert*, pp. 74–82, 1994.

[Green95] Daniel Green, Don Hatch, “Fast Polygon-Cube Intersection Testing”, *Graphics Gems V, Heckbert*, pp. 375–379, 1995.

[Haines94] “Shaft Culling for Efficient Ray-Traced Radiosity”, Eric A. Haines and John R. Wallace, *Photorealistic Rendering in Computer Graphics (Proceedings of the Second Eurographics Workshop on Rendering)*, Springer-Verlag, New York, pp.122–138, 1994, also in *SIGGRAPH ’91 Frontiers in Rendering course notes*.

[Hoff96a] K. Hoff, “A Fast Method for Culling of Oriented-Bounding Boxes (OBBs) Against a Perspective Viewing Frustum in Large ”Walktrough” Models”, *http://www.cs.unc.edu/ hoff/research/index.html*, 1996.

[Hoff96b] K. Hoff, “A Faster Overlap Test for a Plane and a Bounding Box”, *http://www.cs.unc.edu/ hoff/research/index.html*, 07/08/96, 1996.

[Hoff97] K. Hoff, “Fast AABB/View-Frustum Overlap Test”, *http://www.cs.unc.edu/ hoff/research/index.html*, 1997.

[Klosowski97] J.T. Klosowski, M. Held, J.S.B. Mitchell,H. Sowizral, K. Zikan “Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs” , *http://www.ams.sunysb.edu/∼jklosow/projects/coll_det/collision.html*, 1997.

[Slater97] Mel Slater, Yiorgos Chrysanthou, Department of Computer Science, University College London, “View Volume Culling Using a Probabilistic Caching Scheme” *ACM VRST ’97 Lausanne Switzerland*, 1997.