Ray Tracing using Hierarchies of Slab Cut Balls

Linus Källberg and Thomas Larsson

Mälardalen University, Sweden

Abstract

In this paper, bounding volume trees of slab cut balls are evaluated and compared with other types of trees for ray tracing. A novel tree construction algorithm is proposed, which utilizes a relative orientation heuristic between parent and child nodes. Also, a fast intersection test between a ray and a slab cut ball is presented. Experimental comparisons to other commonly used enclosing shapes reveal that the slab cut ball is attractive. In particular, the slab cut ball outperforms the sphere in all tested scenes with speed-up factors between 1 and 4.

Categories and Subject Descriptors (according to ACM CCS): [I.3.6]: Methodology and Techniques—Graphics data structures and data types, [I.3.7]: Three-Dimensional Graphics and Realism—Raytracing

1. Introduction

One of the most widely used data structures for ray tracing is the bounding volume hierarchy (BVH) [KK86, WBS07, DHK07]. This paper presents an initial study on ray tracing using an enclosing shape called the Slab Cut Ball (SCB) in the BVH. An SCB is the intersection volume of the space between two parallel planes, also known as a slab, and a sphere (or ball). The sphere is represented by a center point **c** and radius r, and the slab by a normal **n**, defining the orientation, and signed distances e and f to the slab planes measured from **c**; see Figure 1. This type of bounding volume has been utilized recently in collision detection with promising results [LAM09].

The main advantages of the SCB is its ability to provide fast intersections tests, low transformation costs, and tight-fitting approximations of geometric models at a low memory cost. In particular, for rays traveling along smooth surfaces of low curvature, the worst case behavior of looser-fitting BVs in the tree traversals can be avoided. This is illustrated in Figure 2. Handling this situation well is important to lower the number of required intersections tests in ray shooting.

The main contributions are (1) a top-down construction algorithm for creating tight-fitting SCB trees using a child orientation heuristic; (2) an efficient ray/SCB intersection test; and (3) an experimental comparison of using BVHs of SCBs, balls, AABBs, and OBBs for ray tracing. More details about the presented techniques are given in a companion report [Käl09].

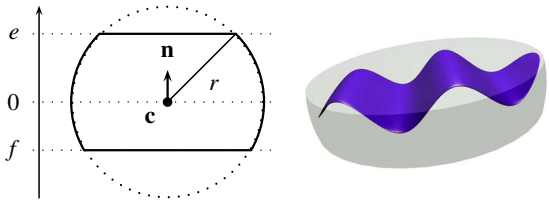

Figure 1: *The parameters used to represent the SCB (left) and an example SCB enclosing a 3D model (right).*

2. Tree construction

The SCB tree is constructed using a recursive top-down strategy. In each step, an SCB is computed to tightly enclose the geometry, and then the primitives are partitioned into two groups on which the algorithm recurses. A leaf is created when there is only one primitive left in a group.

To compute a tight-fitting SCB, first a nearly optimal ball is computed using the EPOS-26 algorithm [Lar08]. A slab is then found by computing a tight-fitting OBB around the geometry and choosing one of its three slabs. Most straightforward would be to choose the slab giving the smallest surface area of the SCB, but as is discussed in the following,

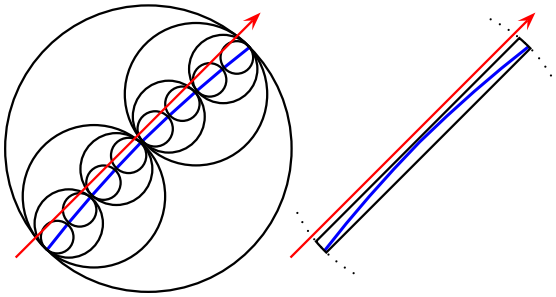

Figure 2: *Cross-sections showing three levels of a sphere tree (left) and one level of a SCB tree (right). Both trees are enclosing the same tessellation of a blue smooth surface. Whereas the red ray intersects all 15 BVs in the sphere tree, not a single BV is hit in the SCB tree.*

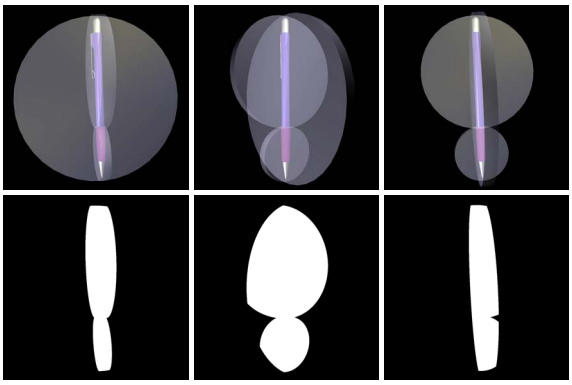

Figure 3: *The child orientation heuristic can help to increase the pruning power in cases where the SCBs are quite poor-fitting by themselves. The top row shows three views of a pencil model together with the first two levels of its SCB tree. In the bottom row, pixels for which the eye rays hit both the parent and child SCBs are highlighted. Clearly, the approach causes many rays to be rejected from further processing on the second level of the BVH.*

better results are possible by considering the other two slabs as well.

The situations when the SCB with the smallest surface area might not be the best choice are when the geometry being enclosed is long and thin. This kind of geometry cannot be tightly enclosed by SCBs, but by using a slab with a large angle to the slab of the parent, much of the empty space in the parent is cut away by the child. Therefore, a larger SCB might be a better choice than a smaller one if it is oriented more orthogonal to the parent.

A child orientation heuristic is used to weigh in this property in the choice of slab by computing the resulting SCB surface areas A_i, $0 \leq i \leq 2$, when using the different slabs, and then choosing the slab giving the smallest *reduced* surface area A'_i:

$$A'_i = (1 - p \cdot \text{ortho}(\mathbf{n}_i, \mathbf{n}_P))A_i, \tag{1}$$

where $\mathbf{n}_i$ is the normal of the ith slab, $\mathbf{n}_P$ is the normal of the parent's slab, and ortho measures the degree of orthogonality between them, giving a value ranging from 0, meaning completely parallel, to 1, meaning completely orthogonal. The parameter $p \in [0,1]$ controls how much this relative orientation between the child and the parent should be weighed in at the expense of the surface area; a value of $p = 0$ corresponds to the simple approach of choosing the smallest SCB. Tests of different values of this parameter have shown that $p \approx 0.2$ is a good choice. The benefits of using the proposed child orientation heuristic when dealing with long thin geometric models is illustrated in Figure 3.

After the SCB has been computed, the primitives are partitioned into two groups using a strategy similar to Wald et al. [WBS07]. A set of candidate partitions are generated, and then the best one is identified using a surface area heuristic (SAH):

$$\text{cost}(n_L, n_R) = \text{SA}(n_L)|n_L| + \text{SA}(n_R)|n_R|, \tag{2}$$

where n_L and n_R are the resulting left and right groups of primitives, respectively, and SA gives the surface area of an enclosing OBB with the same orientation as the OBB used in the SCB fitting step. The reason for using OBBs for the surface area factors instead of SCBs is that OBBs better approximate the shape of the underlying geometry, so this generally leads to better decisions. Also, reusing the orientation of the previous OBB speeds up the computation of these OBBs, which then simply amounts to computing their extents.

The candidate partitions are generated using a set of partitioning planes, where each plane gives a partition by categorizing the primitives based on which side of the plane their centroids lie. The planes are taken to be parallel with the sides of the OBB used in the SCB computation, which gives three different plane orientations. For each orientation, a plane is first positioned so that it separates one primitive from the rest. An OBB is then fitted to this primitive so that the left term in the cost function can be calculated and stored into the first element of an array. This array will later hold the costs for all the partitions generated with the current plane orientation. The plane is then repeatedly repositioned so that one primitive moves over to the other side at a time, until all but one primitive have changed side. This can be done efficiently by first sorting the primitives on the projections of their centroids onto the plane normal. At each step, also the OBB is enlarged to enclose one more primitive, and a cost term is calculated and stored into the cost array. A similar procedure is then carried out where a plane is repeatedly repositioned in the opposite direction, and an OBB is repeatedly enlarged so that the right term of all the partition costs can be calculated and added to the cost array. After doing this

with all three plane orientations, the best partition is found by scanning the cost arrays.

INTERSECTIONFINITERAYSCB(R,S)
input: $R(t) = \mathbf{o} + t\,\mathbf{d}$, $t \in [0,t_h]$, $S = \{\mathbf{c}, r, \mathbf{n}, e, f\}$
output: The intersection status (**true** or **false**)

1. $\mathbf{o}' \leftarrow \mathbf{o} - \mathbf{c}$
2. $k \leftarrow \mathbf{o}' \cdot \mathbf{n}$
3. **if** $\|\mathbf{o}'\|^2 \leq r^2$ **and** $f \leq k \leq e$ **then return true**
4. $b \leftarrow \mathbf{o}' \cdot \mathbf{d}$
5. **if** $\|\mathbf{o}'\|^2 - b^2 > r^2$ **then return false**
6. $l \leftarrow \mathbf{d} \cdot \mathbf{n}$
7. $l' \leftarrow 1/l$
8. $t_0 \leftarrow l'(e-k)$
9. $t_1 \leftarrow l'(f-k)$
10. **if** $t_0 > t_1$ **then** SWAP(t_0,t_1)
11. **if** $t_h < t_0$ **or** $t_1 < 0$ **then return false**
12. $t_n \leftarrow -b$
13. $t_n \leftarrow \text{clamp}(t_n, [0,t_h])$
14. $t_n \leftarrow \text{clamp}(t_n, [t_0,t_1])$
15. **return** $\|\mathbf{o}'\|^2 + (-b-t_n)^2 - b^2 \leq r^2$

Figure 4: *Intersection test between a ray R and a slab S.*

3. Intersection tests

In Figure 4, pseudocode for the developed ray/SCB intersection test is given. The upper bound t_h on the ray length is updated when primitives are intersected, to support efficient rendering of scenes with high depth complexity. First, it is tested whether the ray starts inside the SCB or not. If it does, an intersection has been trivially found (Lines 1–3). Similarly, if the ray misses the ball entirely, intersection has been trivially rejected (Lines 4–5). Then, the parametric values where the ray enters and leaves the slab are calculated and sorted (Lines 6–10), and if it turns out that the finite ray does not reach the slab, intersection is rejected (Line 11).

After this, the intersection status can be determined by considering the point on the ray located inside the slab and also closest to the center $\mathbf{c}$ of the SCB. First, the parametric value $t = t_n$ for this point is initialized to $-b$ (Line 12), which gives the point closest to $\mathbf{c}$ on the infinite line coincident with the ray. Then t_n is adjusted to its correct value by two clamping operations, first to the parameter interval of the finite ray and then to the parameter interval where the ray passes through the slab (Lines 13–14). Now, if this point $R(t_n)$ is located inside the ball, the finite ray hits the SCB, otherwise the ray and the SCB are disjoint (Line 15).

Finally, the cases where the ray is parallel to the slab planes deserve some consideration, since then $l = 0$. Fortunately, by relying on the IEEE-754 interpretation of division by zero, the problem can be solved without introducing any special cases. When the parallel ray is inside the slab, $t_0 = -\infty$ and $t_1 = \infty$. If the parallel ray is outside the slab, t_0 and t_1 will be equal and either $-\infty$ or ∞. None of these cases presents any problem for the algorithm as it is presented. The potential problem of calculating $0/0$, resulting in the value NaN, has been avoided by factoring out the division by l (Lines 7–9).

4. Results

For evaluation, ray tracing performance has been measured for the scenes shown in the top row of Table 1. The *Rings* and *Tree* scenes are due to Eric Haines, and the *Echinodermania II* model is due to George W. Hart. The experiments have been repeated with SCB trees built with $p = 0$ and $p = 0.2$, and also with ball trees, AABB trees, and OBB trees. All trees are built using variants of the SAH, and they are all traversed in an identical manner. The ray/ball intersection test comes from the book by Ericson [Eri05], but has been extended with a simple test to reject hits lying beyond the nearest hit with a primitive. The ray/OBB intersection test comes from the Wild Magic library version 4.10 ("3D segment/box"), but some optimizations inspired by a similar ray/AABB intersection test in Ericson's book have been applied. The ray/AABB intersection test used is the publicly available code by Eisemann et al. [EGMM07]. The code has, however, been slightly reformulated to also reject hits lying beyond $R(t_h)$. All runs have been executed single-threaded on a laptop PC with a 2.53GHz Intel Core 2 Duo T9400 CPU and 3.48 GB of RAM, running Windows XP SP3. The program has been compiled with Microsoft Visual C++ 2005 using the standard release configuration.

The measured data is shown in Table 1. In four of the scenes, the SCB trees have better performance than all the other trees. It is clear that extending the ball with a slab gives substantial improvements in intersection test counts and performance. The SCB trees are outperformed by other trees in *Bars*, *Conference*, and slightly in *Bunny*. The first two of these scenes contain a lot of long and thin triangles, which makes them a challenge for the SCBs. Furthermore, these triangles are axis-aligned, which makes them easy for AABBs to enclose tightly. In these two scenes, also the largest effect of changing the parameter p from 0 to 0.2 is observed—a 14 % performance improvement in *Conference* and 30.5 % in *Bars*—and this supports the rationale of the child orientation heuristic. However, in *Rings* and *Tree*, the SCB trees have the best performance of all trees, which is surprising because these scenes also have a lot of long and thin triangles, so they are expected to be a challenge for the SCB trees as well. On these scenes, the OBB trees are expected to excel, because the geometry is not axis-aligned, and OBBs have a superior ability to adjust to such geometry, but only on *Rings* the OBB tree performs slightly better than the AABB tree.

The ball trees have reasonable performance on some of the scenes, but on others, in particular those that present difficulties for the SCB trees, their performance breaks down

Scene	Tri. count	SCB tree ($p = 0$)			SCB tree ($p = 0.2$)			Ball tree			OBB tree			AABB tree		
		t	N_b	N_t	t	N_b	N_t	t	N_b	N_t	t	N_b	N_t	t	N_b	N_t
Echino.	124.3	2.043	21.24	1.24	2.045	20.97	1.26	2.190	31.46	4.51	4.668	29.89	5.48	3.418	38.45	9.64
Knots	44.8	1.463	14.67	0.68	1.437	14.17	0.65	1.980	25.53	5.16	2.384	17.63	1.30	1.837	24.75	4.10
Bars	0.6	5.896	28.31	19.71	4.100	21.77	12.04	16.644	85.38	80.02	1.357	7.70	1.24	0.809	8.18	1.25
Confer.	282.8	4.555	35.43	7.86	3.917	32.94	5.91	14.172	118.17	62.16	3.558	26.59	2.61	2.044	28.28	4.87
Rings	554.4	19.945	161.42	30.34	18.500	159.86	23.63	41.355	324.06	179.39	19.680	156.43	7.88	22.601	187.24	83.33
Bunny	69.5	1.302	14.60	0.63	1.291	14.44	0.63	1.494	24.43	2.49	1.989	14.97	1.07	1.139	17.30	1.76
Tree	540.5	1.751	16.60	1.87	1.694	16.62	1.47	3.339	31.70	11.08	2.910	16.52	3.07	2.434	20.32	6.26

Table 1: *Benchmark scenes and results. The table shows triangle counts in thousands, rendering times t in seconds, number of ray/BV tests N_b and ray/triangle tests N_t in millions. The abbreviated names refer to "Echinodermania II" and "Conference".*

completely; on *Bars*, the rendering takes 20 times longer with the ball tree than with the AABB tree, and 4 times longer than with the SCB tree. They also have poor performance on *Rings* and *Tree*.

The OBB trees consistently have low intersection test counts. However, due to the costly ray/OBB intersection test, they almost never perform better than the AABBs, except on the *Rings* scene, where the oriented geometry cannot be tightly enclosed by AABBs. They also perform well on *Bars* and *Conference*, where their ability to enclose the long and thin triangles gives them an advantage over SCBs and balls.

Drawing from the results on *Bars* and *Conference*, AABB trees clearly are the most appropriate choice in scenes with much axis-alignment. On scenes with little axis-alignment, in contrast, they become quite loose-fitting and incur more intersection tests than shapes with an ability to adjust their orientation. Despite this, however, on *Bunny*, the AABB tree has better performance than the SCB and OBB trees, which indicates that they have a faster intersection test.

5. Conclusions

The results indicate that the SCB in many situations is able to lower the number of intersection tests compared to other bounding volume shapes, and due to the quick ray/SCB intersection test, this also leads to competitive rendering times.

In the future, it would be interesting to repeat this study using a highly optimized ray tracer striving for real-time rendering by utilizing the parallelization capabilities of recent hardware. Possible extensions to deal with deformable scenes would also be important to explore [WMG*09]. Furthermore, in addition to the bounding shapes used for comparison in this paper, other shapes exist that may be competitive, such as cylinders, k-DOPs, and sphere-swept volumes [KK86, Eri05].

Besides ray tracing, SCB trees may also be advantageous for accelerating more sophisticated global illumination algorithms. For example, when simulating various scattering effects and ambient occlusion, the number of rays that travel in parallel with the surface geometry may raise significantly in certain cases. The proposed tree construction algorithm is also likely to be beneficial in collision detection. In particular, using the child orientation heuristic may be crucial for certain relative configurations between shapes with elongated primitives [LAM09].

References

[DHK07] DAMMERTZ H., HANIKA J., KELLER A.: Shallow bounding volume hierarchies for fast SIMD ray tracing of incoherent rays. *Computer Graphics Forum 27*, 4 (2007), 1225–1233. 1

[EGMM07] EISEMANN M., GROSCH T., MAGNOR M., MUELLER S.: Fast ray/axis-aligned bounding box overlap tests using ray slopes. *journal of graphic tools 12*, 4 (2007). 3

[Eri05] ERICSON C.: *Real-Time Collision Detection*. Morgan Kaufmann, 2005. 3, 4

[Käl09] KÄLLBERG L.: *Accelerating Ray Tracing using Bounding Volume Hierarchies of Slab Cut Balls*. Master's thesis, Mälardalen University, Sweden, October 2009. 1

[KK86] KAY T. L., KAJIYA J. T.: Ray tracing complex scenes. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (1986), pp. 269–278. 1, 4

[LAM09] LARSSON T., AKENINE-MÖLLER T.: Bounding volume hierarchies of slab cut balls. *Computer Graphics Forum 28*, 8 (2009), 2379–2395. 1, 4

[Lar08] LARSSON T.: Fast and tight fitting bounding spheres. In *Proceedings of The Annual SIGRAD Conference* (November 2008), Linköping University Electronic Press, pp. 27–30. 1

[WBS07] WALD I., BOULOS S., SHIRLEY P.: Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Transactions on Graphics 26*, 1 (2007). 1, 2

[WMG*09] WALD I., MARK W. R., GÜNTHER J., BOULOS S., IZE T., HUNT W., PARKER S. G., SHIRLEY P.: State of the art in ray tracing animated scenes. *Computer Graphics Forum 28*, 6 (2009), 1691–1722. 4