Bounding Volume Hierarchies of Slab Cut Balls

Thomas Larsson
Mälardalen University
Sweden

Tomas Akenine-Möller
Lund University
Sweden

June 26, 2008

Abstract

We introduce a bounding volume hierarchy based on the *Slab Cut Ball*. This novel type of enclosing volume provides an attractive balance between tightness of fit, cost of overlap testing, and memory requirement. The hierarchy construction algorithm includes a new method for the construction of tight bounding volumes in worst case $O(n)$ time, which means our tree data structure is constructed in $O(n \log n)$ time using traditional top-down building methods. A fast overlap test method between two slab cut balls is also proposed, requiring as few as 28–99 arithmetic operations, including the transformation cost. Practical collision detection experiments confirm that our tree data structure is amenable for high performance collision queries. For example, in all the tested benchmarks, our bounding volume hierarchy consistently gives performance improvements over both the sphere tree and the OBB tree data structure. In particular, our method is asymptotically faster than the sphere tree, and it also outperforms the OBB tree, in close proximity situations.

Keywords: Bounding volumes; Hierarchical data structures; Overlap testing; Collision detection; Rigid bodies; Simulation; Animation

1 Introduction

We introduce the bounding volume hierarchy (BVH) of slab cut balls. The name *Slab Cut Ball* (SCB) refers to a new type of bounding volume (BV) defined as the intersection of a sphere and the space between two parallel planes, also known as a *slab*.[1] The idea behind the SCB is to combine the best features of OBBs, spheres, and k-DOPs. From the sphere, extremely efficient coarse tests are inherited. The ability to represent arbitrarily oriented flat areas tightly is derived from the OBB. The usage of the slab concept also stems from the k-DOP, however; we allow the slab to be oriented arbitrarily to improve tightness of fit. Note that the extents of the slab planes are limited by the sphere, which means only two circular cross-sections of the sphere are located on the actual boundary of the SCB volume. An example comparing the shape of the SCB to the other classical bounding volumes in 2D is shown in Figure 1.

[1]More generally, a slab is the region between two parallel hyperplanes in $\mathbb{R}^n$.

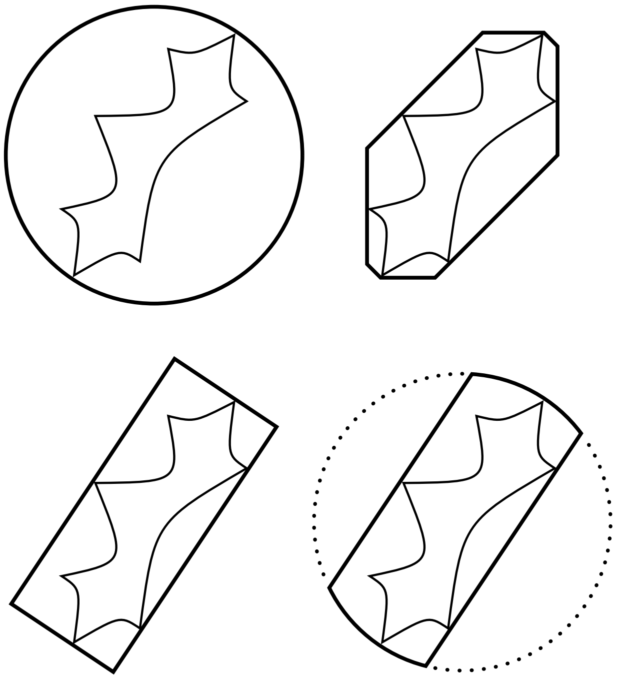


Figure 1: The slab cut ball as a new bounding volume (bottom right) in relation to its three classical ancestors, i.e., the OBB (bottom left), sphere (top left), and k-DOP (top right). The combination of a sphere and an arbitrarily oriented slab provides the means for tightness of fit, efficient overlap tests, and low memory requirements.

An SCB is represented by a center point and a radius for the sphere. The slab is represented by a normal, defining the orientation, and two scalar values defining the width. This means that the SCB improves on both the memory cost and transformation cost of OBBs. Additionally, the transformation problem of AABBs and k-DOPs for rigid body motion is avoided, since the slab is oriented arbitrarily. Other advantages are that the SCB of a point set can be constructed efficiently, and that overlap tests can be made very fast.

It is evident that bounding volumes are very important in computer graphics applications. They have found extensive use in, for example, collision detection (CD) [16, 27, 32, 11, 33, 36, 14], ray tracing [49, 15, 47] and view frustum culling (VFC) [2, 3]. The most popular types of BVs seem to be the AABB [48, 46, 31, 33], OBB [16, 4, 43], sphere [42, 41, 23, 26, 36, 14], and k-DOP [27, 52, 37]. Other choices include the zonotope [18], pie slice [4], cylinder [5], ellipsoid [35], VADOP [7], and convex hull [10, 40].

The idea of combining two or more simple shapes to form a new type of bounding volume has also been considered previously. Examples include the QuOSPO [19], sphere–AABB intersection volume [25], spherical shell [28], capsule (line swept sphere) [29, 9], and lozenge (rectangular swept sphere) [30, 9].

When compared to all these above mentioned bounding volume types, we have found the SCB to be an excellent container for rigid bodies (or rigid body parts), since it successfully combines the advantages of OBBs and spheres. The main feature of the OBB hierarchy is its ability to adapt tightly to the underlying geometry [16]. By using only a single slab as a part of our volume definition, the SCB hierarchy also gains a similar ability to fit tightly around the objects, and moreover at a lower memory cost than for OBB trees.

Furthermore, the sphere is highly attractive because various overlap tests and distance queries become extremely efficient. Therefore, sphere trees have been used extensively to accelerate rigid body collision detection [42, 41]. For example, the time-critical collision detection approach [22, 23, 39], which fits well together with approximate collision response mechanisms [8, 14], are based on spheres. Evidently, these approaches can also be used together with our SCB hierarchies, since a bounding sphere is directly available in each tree node as a part of the SCB definition. For the SCB volume, the rigid body transformation cost involves only the transformation of a point (the center of the sphere) and a normal vector (the orientation of the slab).

As several authors have pointed out, the cost, T, of performing geometric queries using BVHs can be captured by the following cost function [16, 27, 49]:

$$T = N_v \times C_v + N_p \times C_p + N_u \times C_u \tag{1}$$

In the context of rigid body collision detection, N_v and N_p are the number of BV-BV and primitive-primitive overlap tests with C_v and C_p costs per test, respectively. N_u is the number of transformed BVs due to rigid motion of the bodies, and C_u is the cost of transforming a single BV. As can be seen here, tight fitting BVs are attractive to lower N_v, which in turn leads to lower N_p and N_u. However, simpler BVs give lower C_v and C_u, but generally yields a looser fit, thereby increasing the number of tests. This is the well-known trade-off between tightness of fit and overlap test cost, and we argue that the SCB volume provides a highly attractive balance between these two seemingly opposing features.

In the rest of this report, we present the details of our novel type of bounding volume hierarchy. Our main contributions are as follows: (1) We introduce a new bounding volume type to improve the performance of BVHs. (2) We propose an efficient algorithm for computing a tight-fitting SCB enclosing a point set (or a general polygonal model) in worst-case O(n) time. A fast $O(n \log n)$ top-down BVH construction method is also given. (3) An inexpensive SCB–SCB intersection test using only 99 arithmetic operations in the worst case is proposed. (4) Empirical evidence is given to illustrate the efficiency of our BVH, as compared to other common BVH types, for collision detection or interference detection between pairs of rigid bodies.

1.1 SCB Representation and Memory Cost

We represent the SCB with a ball, B, and a slab, S. The ball is given by a center point, $\mathbf{c}$, and a radius, r, and the slab by the normal, $\mathbf{n}$, of the slab planes and the signed distances e and f from $\mathbf{c}$ to the planes along the normal direction. This is illustrated in Figure 2. The area, A, and volume, V, of an SCB are found by applying the area and volume formulas for sphere segments, which gives the following equations:

$$A = 2\pi r(e-f) + a^2\pi + b^2\pi, \tag{2}$$

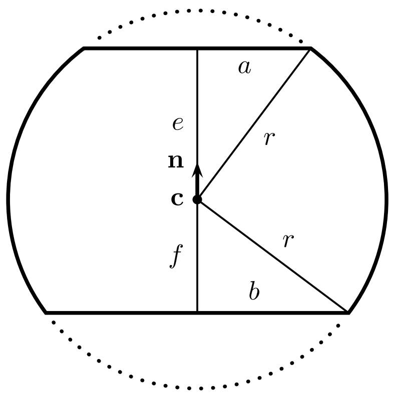


Figure 2: The SCB is defined by the sphere center **c** and radius r, the signed distances e and f to the slab planes measured from **c** and the normal vector **n** of the slab planes.

$$V = \frac{\pi}{6}e(3r^2+3a^2+e^2)+\frac{\pi}{6}|f|(3r^2+3b^2+f^2). \tag{3}$$

We can simplify these equations, since the distances a and b are given by the Pythagorean Theorem as $a=\sqrt{r^2-e^2}$ and $b=\sqrt{r^2-f^2}$ (see Figure 2). Thus,

$$A = \pi\left(2r(e-f)+2r^2-e^2-f^2\right), \tag{4}$$

$$V = \frac{\pi}{3}\left(e(3r^2-e^2)+|f|(3r^2-f^2)\right). \tag{5}$$

Regarding the memory cost of the SCB volume, 9 scalar values need to be stored. This can be compared to other volumes, e.g. the sphere, AABB, OBB, and 26-DOP, requiring 4, 6, 15, 26 scalar values of storage, respectively. We note that the BV storage cost may be important not only for saving memory, but also for improving the cache hit rate during hierarchy traversals. Therefore, the low memory cost of the SCB is an attractive feature.

If all the values are 4 bytes wide, the storage requirement of an SCB amounts to $9 \times 4 = 36$ bytes. However, we note that this can be reduced easily. For example, instead of storing e and f directly, we can store them as a fraction of the sphere radius using, for example, two bytes per value. Similarly, the normal of the slab can be quantized in the BV computation algorithm to make the components of the normalized normal, n_x, n_y, and n_z, representable as two bytes per component. In this way, the total memory requirement for an SCB can be adapted to, for example, 26 bytes. If we want to pack BVH nodes efficiently with respect to common cache line sizes (32, 64, or 128 bytes), this leaves 6 bytes of extra storage for other necessary node data apart from the actual SCB.

2 Fast SCB Computation

An SCB covering a polygonal model can be computed by using existing algorithms for computing the enclosing ball and the minimum width of a point set, respectively. Optimal bounding spheres can, for example, be computed

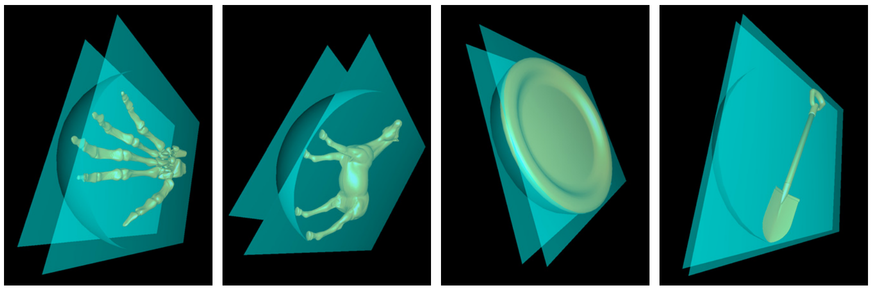
```
ComputeBoundingSCB($P, B, S$)
    input: A point set $P = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n\}$ with $n$ points
    output: A ball $B = \{\mathbf{c}, r)\}$ and slab $S = \{\mathbf{n}, e, f\}$
1.      Compute26DopAndIts26ExtremalPoints($P, D, E$)
2.      if($n > 26$) then
3.          $B' \leftarrow$ MinBall($E$)
4.          $B \leftarrow$ UpdateBallToEncloseAllPoints($P, B'$)
5.      else
6.          $B \leftarrow$ MinBall($P$)
7.      $S \leftarrow$ InitSlabFromMinimumSlabOf26Dop($\mathbf{c}, D$)
8.      for each point pair $(\mathbf{h}_i, \mathbf{l}_i) \in E$
9.          $\mathbf{p} \leftarrow$ SelectFurthestPointFromLine($\mathbf{h}_i, \mathbf{l}_i, E$)
10.         $\mathbf{n}' \leftarrow (\mathbf{l}_i - \mathbf{h}_i) \times (\mathbf{p} - \mathbf{h}_i) / \parallel (\mathbf{l}_i - \mathbf{h}_i) \times (\mathbf{p} - \mathbf{h}_i) \parallel$
11.         ComputeSlabWidth($P, \mathbf{c}, \mathbf{n}', e', f'$)
12.         if ($e' - f' < e - f$) then
13.             $S \leftarrow \{\mathbf{n}', e', f'\}$
```

Figure 3: Algorithm for computing an SCB enclosing a point set in worst case $O(n)$ time.

in expected $O(n)$ time using the randomized algorithm by Welzl [50, 12]. A reasonable tight sphere can also be computed very efficiently using Ritter's method [44]. However, Ritter's method fails to compute tight-fitting spheres for some models, in particular models where the optimal enclosing sphere is uniquely defined by three or four supporting vertices.

The problem of finding the optimal, i.e. the narrowest, slab is often referred to as computing the minimal width of a point set. There are theoretical solutions with time complexity $O(n^{3/2+\varepsilon})$ [1]. The earlier Houle-Toussant algorithm has a worst-case time complexity of $O(n^2)$ [21]. However, it seems to be quite usable in practice as demonstrated by several implementations [45, 13]. Additionally, Chan proposes $(1 + \varepsilon)$-factor approximation algorithms for the minimal width problem of a point set [6].

Since our goal is to compute a tight-fitting SCB covering the point set defining a polygonal model in worst case $O(n)$ time, we propose the BV construction algorithm shown in pseudocode in Figure 3. The first part of the algorithm computes a 26-DOP represented as a set of intervals, $D = \{[h_i, l_i]\}$ with $i \in [0, 12]$, while also storing the extremal or supporting vertex pairs $E = \{[\mathbf{h}_i, \mathbf{l}_i]\}$ of the 26-DOP, i.e., the vertex pairs in E are the points that span the slab planes of the 26-DOP (Line 1). Note that a fixed set of 13 normal vectors with integer coordinates selected from the set $\{0, \pm 1\}$ are used for faster computation of the 26-DOP (cf. [27]). Thus, at most 26 unique extremal vertices of the model are selected. In Lines 3–4, we first compute an optimal enclosing sphere covering E using Gärtner's minball implementation [12] of Welzl's algorithm [50]. Then, the sphere is grown to cover all points P of the model, if needed. However, when $n \leq 26$, we instead compute an optimal enclosing ball directly of the polygonal model itself (Line 6). Note that this approach improves the tightness of fit of the spheres significantly compared to, for example, the popular Ritter's



$n = 327323$ $t = 65$ ms | $n = 48485$ $t = 9.5$ ms | $n = 7282$ $t = 1.5$ ms | $n = 1600$ $t = 0.4$ ms

Figure 4: Some models with their computed SCBs. The number of vertices of the models, n, and the SCB computation times, t, are also given.

algorithm [44] in the general case.

The remaining part of the algorithm focuses on finding a tight-fitting slab. First, we select the slab in the 26-DOP with minimum width as an initial slab (Line 7), which we then try to improve by considering the normal vectors of 13 large triangles formed by an extremal point pair in E and an additional point $\mathbf{p} \in E$. Here $\mathbf{p}$ is always chosen as the point furthest away from the line through the current point pair $(\mathbf{h}_i, \mathbf{l}_i)$ (Line 9). Then the normalized normal of the constructed triangle is computed (Line 10). After this, the resulting slab width using this normal is computed in Line 11, and finally, the slab is saved, if it is the narrowest slab found so far (Lines 12–13). Optionally, as a last step in the algorithm, a user-defined error margin ε may be used to extend the slab width slightly to avoid underlying primitives to touch the slab planes.

Note that this approach gives close to optimal spheres and a good approximation of the minimum width slab (in particular for rather flat objects). The results from using our method to compute the SCBs for some initial test models we used are shown in Figure 4. The execution times were measured using a single-threaded C++ implementation of the algorithm running on a computer with an Intel CPU T2600 2.16 GHz. As can be seen, the time grows linearly with the number of vertices, as expected. Finally, note that these test models were randomly rotated before the computation to avoid conveniently aligned models, e.g. with respect to the fixed normal vectors used in the 26-DOP computation.

3 Hierarchy Construction

The tree building strategy we use is a simple recursive $O(n \log n)$ top-down construction method. It is also a generic construction method in the sense that it utilizes no knowledge of the actual BV type used in the hierarchy nodes in the partitioning of the geometric primitives. Furthermore, no restrictive assumptions about the polygonal input model is made. For instance, a model represented by a polygon soup without topological information works fine.

The construction proceeds as follows. Starting from the root node enclosing a complete polygonal model, we recursively split the geometric primitives in two subsets forming a left and right tree branch. The recursion proceeds until there is only a single primitive left, in which case a leaf node is created. The split

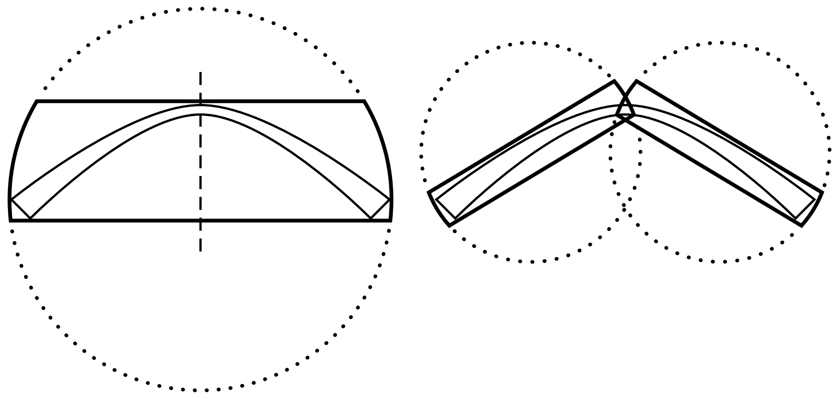


Figure 5: The result of a geometry split operation during hierarchy construction. The split axis is shown with a dashed line. Note how the SCB volumes are able to align themselves with the underlying surface, as opposed to the spheres shown with dotted lines.

heuristic we use is based on the in-place partitioning technique used in quicksort [20]. In our case, the pivot element is the split plane dividing the longest side of the axis-aligned box of the remaining primitives through the centroid of the box. The primitives are represented using an array of integers, with each integer being a polygon index. The polygon indices are then rearranged in-place into a left and right subset using the centroids of the primitives for comparisons with the pivot element. When the two subsets have been found, a BV of a specified type is computed for each subset and assigned to the child nodes. Such a split operation using the SCB as the BV type is exemplified in Figure 5. Note also that just as quicksort usually sorts remaining short sublists of constant length using, for example, insertion sort, we too switch to insertion sort in our partition strategy when there are less than 8 primitives left to partition.

To ensure robustness as well as reasonably balanced hierarchies, however, we detect whenever the relative proportions of the primitives in their respective subset are worse than 10% and 90% . In these cases, we redo the split using split planes through the remaining sides of the box in turn, starting with the second longest side. Should these split efforts also fail to be reasonably balanced, which seems to occur rarely in practice, our final choice is the split plane dividing the longest side of the box through the median of 7 randomly selected triangle centroids.

3.1 SCB Convergence Rate

As argued by Gottschalk et al. [16], the diameter of the BV is expected to halve as we descend one level of the hierarchy given that a binary tree is used. If at the same time the width, or thickness, of the BV is quartered, the convergence rate is quadratic. This is a very attractive feature of a BV, since it leads to asymptotically fewer BV–BV tests in collision queries in certain complex geometric situations, which is referred to as Parallel Close Proximity (PCP), meaning that every point of a surface is close to some point on the other surface.

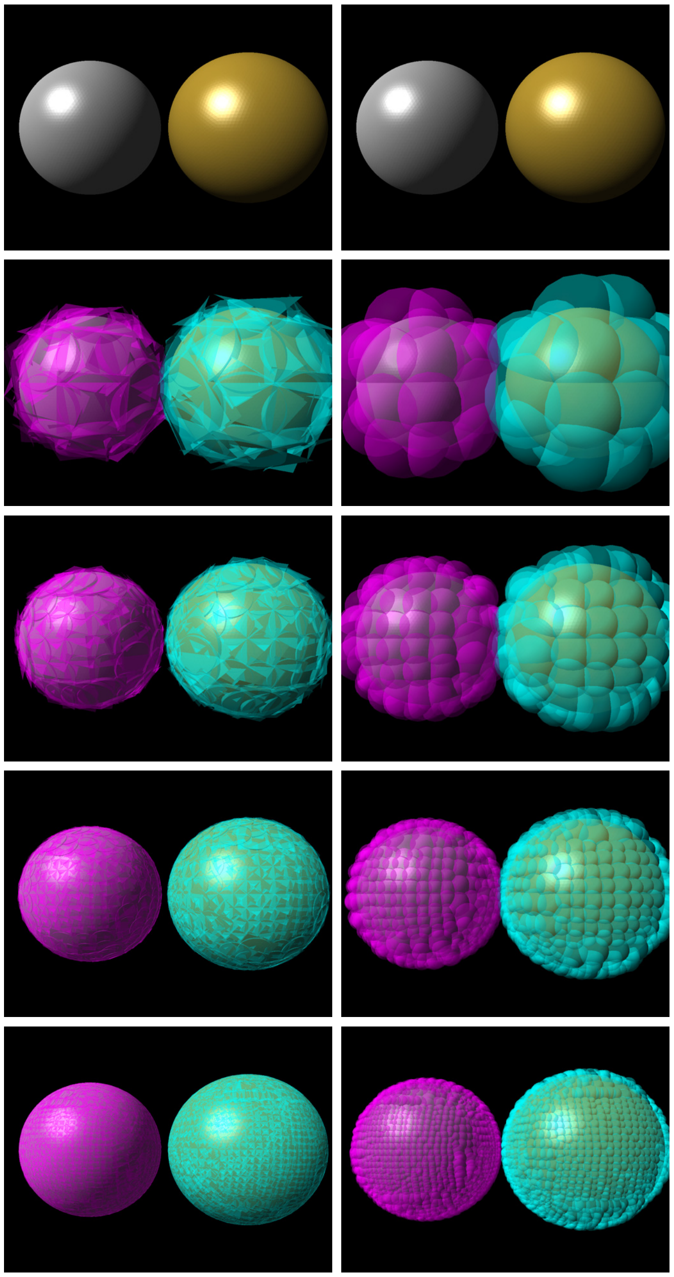


Figure 6: The pruning power of SCBs versus spheres. Two tessellated sphere models are shown together with their BVH of SCBs (left) and BVH of spheres (right). The levels shown are 5, 7, 9, and 11, respectively. As can be seen, the SCB hierarchy converge much faster to a tighter approximation of the mesh surface. For example, notice how the size of the gap between the two sphere models become large rapidly in the SCB case as we go deeper down in the hierarchies.

Such situations occur, for example, in virtual prototyping when fitting machine parts together.

Although the theoretical analysis by Gottschalk et al. [16] suggests a quadratic convergence rate for OBBs, it can also be applied for SCBs with diameter $d = 2r$ and thickness $\tau = e - f$. However, as pointed out, several simplifications are made in the analysis, such as assuming only nearly flat surfaces (with a smooth curvature) and ignoring BV packing inefficiencies. The true geometry convergence of a BVH, however, is model-dependent and may vary drastically across different parts of a model. In general, for closed meshes, the convergence for OBBs, or SCBs, may be linear in the upper levels of the hierarchy because the volumes cover mesh surfaces of high curvature, quadratic in subtrees covering nearly planar surfaces with a smooth curvature, and in the leaf nodes τ can be zero, or close to zero, for example, where the BV bounds a single polygon, giving an infinitely large BV aspect ratio improvement.

Therefore, the performance of collision detection algorithms using BVHs are highly model- and scenario-specific and experimental evaluation is necessary. However, as is shown by our experimental results in Section 5, the SCB hierarchies are able to efficiently handle a full range of common situations. These include models in parallel close proximity, models at deep interpenetration, models barely touching each other, and models well separated.

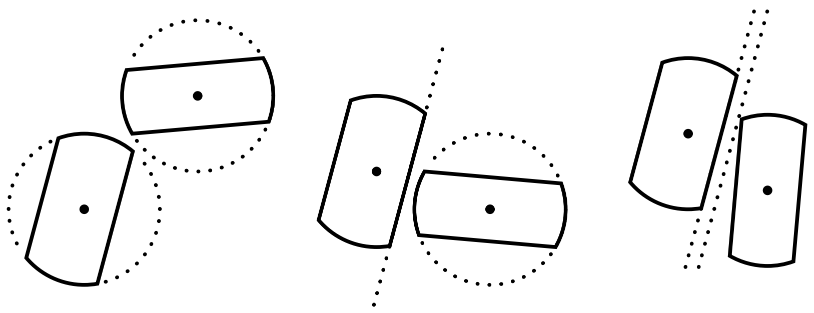


Figure 7: The three main types of distance measures used for simple overlap rejection in the overlap test: (A) Sphere–Sphere, (B) Plane–Sphere, and (C) Plane–Plane.

4 A Fast SCB–SCB Overlap Test

The main idea behind the overlap test is to utilize three types of quick rejections, which boils down to simple separation conditions based on sphere–sphere, plane–sphere, and plane–plane relative locations, as shown in Figure 7. Denote the involved SCBs as $A_0 = \{B_0, S_0\} = \{\mathbf{c}_0, r_0, \mathbf{n}_0, e_0, f_0\}$ and $A_1 = \{B_1, S_1\} = \{\mathbf{c}_1, r_1, \mathbf{n}_1, e_1, f_1\}$. Each SCB is also associated with a rigid body transformation matrix, which means A_0 is transformed by $\mathbf{M}_0$, and A_1 by $\mathbf{M}_1$. We choose to execute the overlap test by transforming A_1 into the local space of A_0 using the transformation

$$\mathbf{M}_{1\to 0} \quad = \mathbf{M}_0^{-1}\mathbf{M}_1 \tag{6}$$

Note that the calculation of $\mathbf{M}_{1\to 0}$ is an initial setup cost, which in the case of hierarchical CD, only needs to be computed once for an entire CD traversal of

```
OVERLAP($A_0, A_1$)
    input: $A_0 = \{\mathbf{c}_0, r_0, \mathbf{n}_0, e_0, f_0\}$, $A_1 = \{\mathbf{c}_1, r_1, \mathbf{n}_1, e_1, f_1\}$, and $\mathbf{M}_{1\to 0}$
    output: The overlap status (true or false) of $A_0$ and $A_1$
1.      $\mathbf{c}'_1 \leftarrow \mathbf{M}_{1\to 0}\ \mathbf{c}_1 - \mathbf{c}_0$
2.      if $\| \mathbf{c}'_1 \|^2 > (r_0 + r_1)^2$ return false
3.      $d_0 \leftarrow \mathbf{n}_0 \cdot \mathbf{c}'_1$
4.      if $d_0 - e_0 > r_1$ or $d_0 - f_0 < -r_1$ return false
5.      $\mathbf{n}'_1 \leftarrow \mathbf{M}_{1\to 0}\ \mathbf{n}_1$
6.      $\mathbf{c}'_0 \leftarrow -\mathbf{c}'_1$
7.      $d_1 \leftarrow \mathbf{n}'_1 \cdot \mathbf{c}'_0$
8.      if $d_1 - e_1 > r_0$ or $d_1 - f_1 < -r_1$ return false
9.      $c_\alpha \leftarrow \mathbf{n}_0 \cdot \mathbf{n}'_1$
10.     if $c_\alpha < -a$
11.         if $d_0 > 0$
12.             if CYLINFRONTOFSLAB($\mathbf{c}'_1, \mathbf{n}'_1, e_1, r_1, \mathbf{n}_0, e_0, c_\alpha$) return false
13.             if CYLINFRONTOFSLAB($\mathbf{c}'_0, \mathbf{n}_0, e_0, r_0, \mathbf{n}'_1, e_1, c_\alpha$) return false
14.         else
15.             if CYLBEHINDSLAB($\mathbf{c}'_1, \mathbf{n}'_1, f_1, r_1, \mathbf{n}_0, f_0, c_\alpha$) return false
16.             if CYLBEHINDSLAB($\mathbf{c}'_0, \mathbf{n}_0, f_0, r_0, \mathbf{n}'_1, f_1, c_\alpha$) return false
17.     else if $c_\alpha > a$
18.         if $d_0 > 0$
19.             if CYLINFRONTOFSLAB($\mathbf{c}'_1, \mathbf{n}'_1, f_1, r_1, \mathbf{n}_0, e_0, c_\alpha$) return false
20.             if CYLINFRONTOFSLAB($\mathbf{c}'_0, \mathbf{n}_0, f_0, r_0, \mathbf{n}'_1, e_1, c_\alpha$) return false
21.         else
22.             if CYLBEHINDSLAB($\mathbf{c}'_1, \mathbf{n}'_1, e_1, r_1, \mathbf{n}_0, f_0, c_\alpha$) return false
23.             if CYLBEHINDSLAB($\mathbf{c}'_0, \mathbf{n}_0, e_0, r_0, \mathbf{n}'_1, f_1, c_\alpha$) return false
24.     return true
```

Figure 8: A fast conservative SCB–SCB overlap test algorithm.

two hierarchies.

The pseudocode given in Figure 8 shows the details of the overlap test. First the overlap status of the two balls B_0 and B_1 are tested (Lines 1–2). Then follow two tests of the type sphere–slab, i.e., the overlap status of B_0 and S_1 (Lines 3–4) as well as B_1 and S_0 (Lines 5–8) are examined based on computing simple sphere–plane distances. Finally, given that the orientations of the slabs are reasonably aligned, which is tested on Lines 9–10 and 17[2], a separation test for the two slabs S_0 and S_1 is executed (Lines 10–23).

To be able to do this last type of rejection test, the extent of at least one of the slabs has to be bounded. To bound S_1, we use a cylinder C_1 with its main axis aligned with the normal $\mathbf{n}_1$, and $\mathbf{c}_1$ as the bottom base point. The radius of the cylinder is simply r_1, and the height of the cylinder is either e_1 or $|f_1|$, depending on which side is facing towards the slab S_0. A quick rejection can now be done given that the circular top of the cylinder does not reach the slab S_0. Then follows the corresponding test between the cylinder C_0 and slab S_1. Note that there are four different cases for the overlap test between the cylinder

[2]The constant a used in Lines 10 and 17 was set to 0.7 in our implementation.

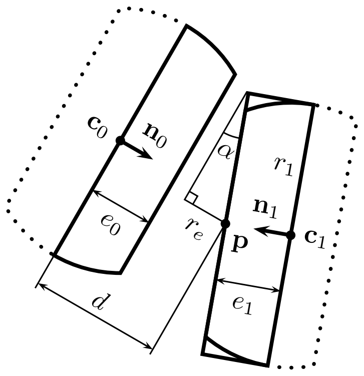


Figure 9: Rejection test for aligned slab case. A cylinder is derived which is then tested for separation with the slab of the other volume by considering the effective radius, r_e of the cylinder.

task	#op	#cmp	exits
transform point	18	-	-
sphere/sphere	10	1	(28, 1)
slab/sphere	7	2	(35, 3)
transform normal	15	-	-
sphere/slab	10	2	(60, 5)
slab/cylinder	22	4	(82, 9)
cylinder/slab	17	4	(99, 13)

Table 1: SCB-SCB overlap test cost. The number of simple arithmetic operations (#op) and comparisons (#cmp) per listed task are given. The last column shows the possible exits or returns with the corresponding accumulated operation cost.

and the slab depending on which sides of the SCBs are facing towards each other. Which case to apply is found by the conditional tests on Lines 10, 11, 17, and 18. The calculations needed for the actual cylinder–slab rejection test are very simple. Let us consider one of the four possible cases. From Figure 9, we see that

$$\mathbf{p} = \mathbf{c}_1 + e_1\mathbf{n}_1 \tag{7}$$

$$d = \mathbf{p} \cdot \mathbf{n}_0 \tag{8}$$

$$r_e = r_1 \sin\alpha \tag{9}$$

$$\cos\alpha = \mathbf{n}_0 \cdot \mathbf{n}_1 \tag{10}$$

Therefore,

$$r_e = r_1\sqrt{1-\cos^2\alpha}, \tag{11}$$

$$r_e^2 = r_1^2(1-(\mathbf{n}_0 \cdot \mathbf{n}_1)^2). \tag{12}$$

Now, given that $d > e_0$ and $(d - e_0)^2 > r_e^2$, we can conclude that there is no overlap.

Table 1 gives an overview of the operation cost of the overlap test, which includes the cost of transforming the second SCB into the local coordinate sys-

	Model data	
Model	n_v	n_p
Sphere-L1	10242	20480
Sphere-L2	40962	81920
KnotA-L1	1764	3528
KnotB-L1	1836	3672
KnotA-L2	3430	6860
KnotB-L2	3570	7140
KnotA-L3	5488	10976
KnotB-L3	5712	11424
Horse	48485	96966
Hand	327323	654666

Table 2: Data for the triangular meshes used in the benchmarks, where n_v and n_p are the number of vertices and triangles respectively.

tem of the first SCB. Only simple arithmetic operations (add, sub, mul) and comparisons are used. Note that early outs are possible after either 28, 35, 60, and 82 operations. The worst case is 99 operations and 13 comparisons.

This can be compared to the OBB–OBB overlap test by Gottschalk et al., which requires 252 simple arithmetic operations and 15 comparisons in the worst case, including the transformation cost [16]. In the best case, our test requires 28 operations and 1 comparison, compared to 49 operations and 1 comparison for the OBB test, assuming only the midpoint and one axis is transformed before the first separating axis test is executed. Note that without the transform cost, our overlap test only requires 66 arithmetic operations in the worst case, and 10 operations in the best case. For OBBs, the corresponding operation cost is 189 operations in the worst case, and 26 in the best case. See Gottschalk's PhD thesis for the details of the OBB–OBB overlap test [17].

A potential drawback of our overlap test is that it is conservative, meaning that false positives are possible leading to deeper hierarchy traversals before the overlap status can be answered. Using conservative overlap tests, however, is a common technique in hierarchical collision detection [46, 27, 32, 34], since it often speeds up the overlap test significantly, thereby giving an overall performance gain. Our experiments confirm that this is indeed the case using our conservative overlap test, even under really hard circumstances, such as parallel close proximity situations.

As an alternative of using an enclosing cylinder as a safe approximation when calculating the effective radius, r_e, for the last type of rejection in the overlap test, the effective radius of the intersection volume itself could be considered. We note that this would reduce the number of false positives somewhat, but it would also make the overlap test more computationally expensive.

5 Evaluation

The algorithms presented have been implemented in C++, and compiled using Microsoft Visual Studio 2005 with the standard release setting. The performance measurements were run single-threaded on a laptop machine with an

	BallTree			SCBTree			OBBTree		
Scene	t	N_v	N_p	t	N_v	N_p	t	N_v	N_p
Spheres-L1	8.0	99733	10585	3.3	21412	2112	4.9	13350	1135
Spheres-L2	16.4	200235	20776	6.5	42295	4419	9.2	24835	2240
Knots-L1	9.7	68543	15503	1.8	13005	63	3.0	8731	0
Knots-L2	12.5	121316	31703	1.7	14189	53	2.4	9047	0
Knots-L3	18.4	175479	49840	1.8	14367	33	2.5	9365	0
Horses	8.6	93193	12709	5.8	39469	4773	8.4	23200	1748
Hands	101.4	1028875	161923	62.1	387257	53083	83.2	239346	17522

Table 3: Comparison of the average CD execution times, t, in ms, the average number of BV–BV tests, N_v, and the average number of triangle–triangle tests, N_p, for the different types of BVHs and test scenes.

Intel T2600 2.16 GHz CPU and 1 GBytes of RAM.

The collision detection query takes two rigid bodies as input, where each body is represented by a triangle mesh (or a triangle soup) and a rigid body transformation matrix. The search for intersecting triangle pairs is performed by a recursive dual traversal of the BVHs associated with the models. The pseudocode for such a traversal is given by Gottschalk [17]. In each recursive step, we descend in the hierarchy where the radius of current bounding volume is the largest. We have tried other descent rules, based on maximum volume or area, or using only parts of the area or volume computation. However, descending in the subtree with the largest radius gave the best overall execution times in all our experiments. See Ericsson's book for a further discussion of descent rules [11].

We have tested our methods on four different benchmark scenarios, which we refer to as *Spheres*, *Knots*, *Horses*, and *Hands*. Some sample images from these scenarios are shown in Figure 10. The first two test scenes are also repeated using two and three different levels-of-detail (LODs) of the triangular meshes, respectively. For each scenario, executions times are measured for three different types of bounding volumes in the hierarchies. The used types are the SCB, ball (or sphere), and OBB. We call the resulting hierarchy types *SCBTree*, *BallTree*, and *OBBTree*, respectively. The first two of these types are implemented in our own system. For the comparisons with the OBBTree, the publicly available software package RAPID, version 2.01, by Gottschalk is used. Data for the models used in the experiments are given in Table 2. The results from all the CD benchmark tests are summarized in Table 3.

The experimental setup for our first scenario called Spheres is described below. Two finely tessellated spheres with radii of 0.95 and 1.0 are moving straight through each other. Initially, their center points are located at $(-1, 0, 0)$ and $(1, 0, 0)$. Then during 500 frames of simulation, the sphere models are translated in equally sized steps along the x-axis towards each other while also rotating around their own local center point. In frame 250, their center points are coincident and there is PCP over the entire mesh surfaces. Note that a very similar test scenario was used by Gottschalk with the purpose of demonstrating the superiority of the OBB over simpler BV types in grazing or near grazing situations [17]. The sphere meshes are initially generated by subdividing an icosahedron. The scenario is run using two different LODs of the meshes, with

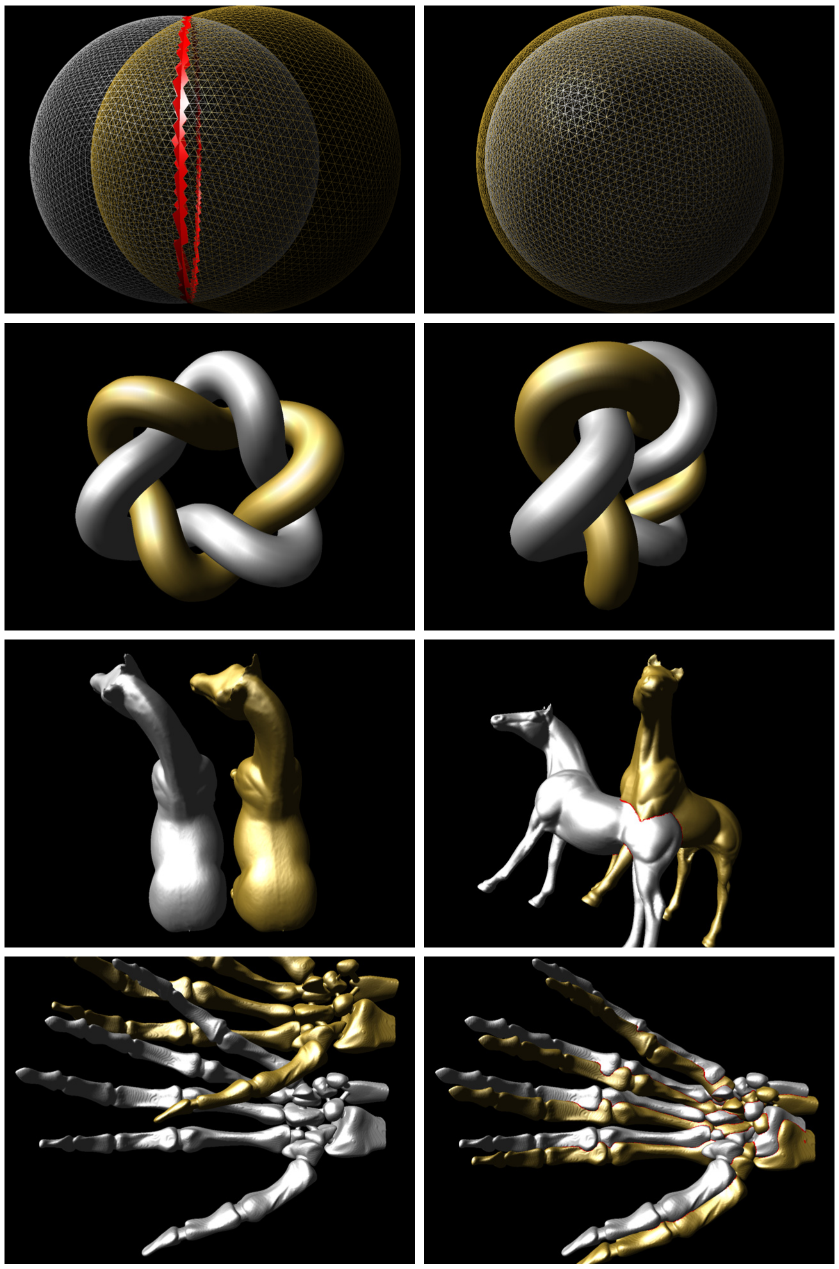


Figure 10: Selected images from the four benchmark scenarios Spheres, Knots, Horses, and Hands. The simplest LOD is shown for the Spheres scene, and the most complex LOD for the Knots scene.

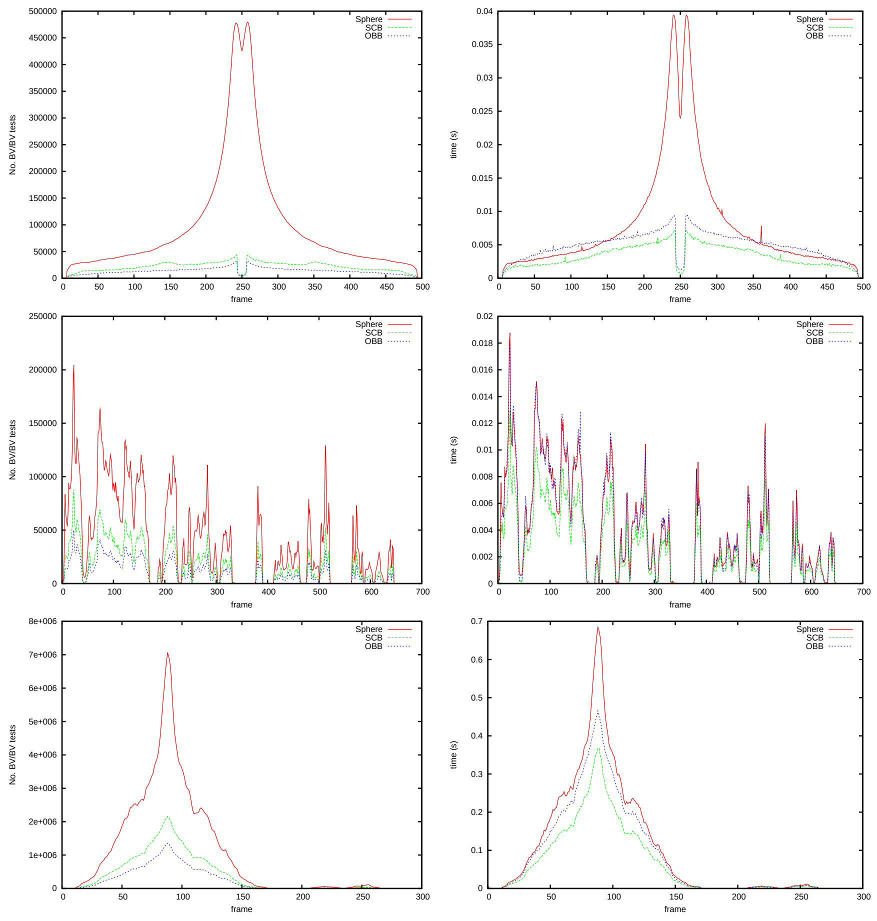


Figure 11: The number of BV/BV tests, N_v (left), and CD time per frame (right) for the benchmarks Spheres-L1 (top), Horses (middle), and Hands (bottom).

either 20480 or 81920 triangles per sphere model. The average numbers of BV-BV tests and CD times over all frames are given in Table 3.

Several interesting performance aspects are revealed by plotting N_v and the CD time for each frame of the simulation. The plots for the simplest LOD are given in Figure 11. As expected, N_v is large for the BallTree and the performance breaks down completely while near or in the PCP situation. Using SCBs requires approximately twice as many BV/BV tests as OBBs, except while in PCP (frame 244-256), when all of a sudden SCBs require slightly fewer tests than OBBs. Over the entire scenario, the SCBTree is faster than the other BV types. Although the performance profiles of the SCBTree and OBBTree are quite similar, we note that the OBBTree is actually slower than the BallTree during frames 20–140 and 360–480. On average, the speed-up for the SCB-Tree is 2.4 compared to the BallTree, and 1.5 compared to the OBBTree. In the PCP situation (frame 244-256), the corresponding speed-ups are 29.7 and 1.7. Although the plots are not included here, very similar advantages for the

SCBTree can also be seen for the second LOD. For example, in this case the SCBTree gives a speed-up of 38.6 compared to the BallTree and 1.7 compared to the OBBTree during the PCP situation (frame 244-256).

In the second scenario, two intertwined knot models are rotated together during 500 frames of animation in such a way that their relative distance to each other remains the same. Since the two knots are close to each other along a three-dimensional curve, we refer to the geometric situation for Curve Close Proximity (CCP). We repeat the experiment using three different LODs, and as expected both the SCBTree and the OBBTree are asymptotically faster than the BallTree. The speed-ups over the BallTree when using the SCBTree are 5.4, 7.4, and 10.2 over the three LODs. For the OBBTree the corresponding speed-ups are 3.2, 5.2, and 7.4. As can be seen, the SCBTree is faster than the OBBTree over all LODs.

The third scenario includes two rotating horse models that are slowly moving apart from each other. The rotation is varied around all three principal coordinate axes by incrementing Euler rotation angles so that a variety of different relative poses between the models are tested. The scenario involves no close proximity situation (PCP or CCP), which would give an apparent advantage to the tighter bounding volume types. The average collision detection time per frame is 8.6 ms using the BallTree and 8.4 ms using the OBBTree. Again, the SCBTree is faster than the other methods. It spends 5.8 ms per frame on average, giving speed-ups of 1.5 and 1.4, respectively. See Figure 11 for a plot of N_v and the CD time per frame.

In the fourth benchmark, a skeleton hand falls down through another skeleton hand. A substantial number of intersections occur, because of the high polygon count of the hand model. There is no case of close proximity situation (PCP or CCP) occurring in the scenario. The average CD time per frame is 101 ms using the BallTree and 83 ms using the OBBTree. Again, the SCBTree, which only required 62 ms CD time per frame, is faster than the other methods. Plots with per frame data are given in Figure 11.

Finally, note that the sphere volumes used in the BallTree nodes in our experiments are as tight as the spheres produced by our SCB construction algorithm, since they are constructed using the first part of the algorithm (line 1–6) in Figure 3. When we re-run the experiments after using Ritter's sphere construction method instead when building the BallTree, the execution time increased by 18, 2, 12, and 10 percent in the benchmarks Spheres, Knots, Horses, and Hands, respectively.

6 Discussion

While SCBs are able to tightly approximate many simple shapes, there are exceptions, such as long thin objects like a pencil or the spade shown in Figure 4. On the other hand, OBBs also show poor fitting in relation to certain simple shapes where SCBs excel, such as rather flat circular objects like plates or frisbees.

Still, objects with long thin triangles, which arise in for example sparsely tessellated cones or cylinders, seem to be the main problem for SCBs. It would be easy to design a scenario where the advantage of OBBs for meshes with such long thin triangles would shine through. In this study, we have assumed that the input meshes are finely tessellated, which means that the SCBs can adapt to the surface of the mesh tightly over surface areas with a smooth curvature. However, we see several possible solutions to the problems with long thin triangles.

One approach would be to allow several SCBs to tile themselves over a single polygon which would avoid too loose-fitting volumes. This solution has been suggested previously for sphere trees [36]. Unfortunately, this approach would somewhat defeat our goal of low memory consumption. Another way to handle the problem would be to use a hybrid hierarchy type in which both SCBs and OBBs are used to make it possible to approximate the long thin triangles with OBBs. However, this would make the implementation more complex and the memory requirement per node would differ depending of the type of volume used.

A third solution would be to extend the SCB to use a sphere cut by 0–k optional slabs (0–2 slabs seem best). Clearly two slabs would be enough to give our volume the same kind of tightness of fit as for OBBs even for long thin objects. The value of k would be chosen individually per node during hierarchy construction to reach a specified quality metric. However, this would increase the storage cost, transformation cost, and overlap test cost (whenever $k > 1$). Furthermore, the memory size of the SCB would vary from volume to volume in the tree nodes. This would make cache efficient layout of the hierarchies in arrays more complex.

A variation of this solution would be to select k number of slabs from a set of fixed orientations in model local space. Then during rigid body motion the slabs could still be rotated freely with the model. However, this would still increase the storage cost, but not as much as when arbitrary slab orientations are used. Unfortunately, in this case the volume would lose its ability to closely approximate even flat parts of the underlying meshes with severe performance implications in PCP situations.

Finally, rather than including a second arbitrarily oriented slab in the SCB, a somewhat similar effect can perhaps be achieved by orienting the slabs in parent/child nodes carefully in relation to each other when dealing with polygons with a problematic size and shape. In fact, it might always be a good idea to consider the orientation of the slab in the parent node when defining the slab orientations in the child nodes. This means that the minimal width, or narrowest, slab is not always the best choice. Controlling the relative orientation of the slabs between parent and child nodes for long thin objects in an intelligent way is left here as future work.

7 Conclusions and Future Work

With its low memory requirements, efficient overlap test, and the ability to rapidly converge into tight-fitting bounding volumes over smooth surface areas, we have good reasons to believe that the SCB is an attractive type of volume for BVHs. This has been confirmed by the results from our collision detection experiments between complex polygonal models which showed that the SCB-Tree gave a clear performance advantage over a wide range of scenarios. In particular, these results also indicate that the SCBTree is asymptotically faster than the BallTree, as well as being faster than the OBBTree, in PCP situations. Furthermore, since our approach can be regarded as an extension of ball trees, which are widely used in areas such as computer graphics, computational geometry, robotics, and visualization, there is a large number of applications that could potentially benefit from our approach.

In the future, we would like to examine the suitability of the SCB hierarchy for accelerating other types of geometric queries such as distance queries, ray tracing, view frustum culling, and occlusion culling. Also, generalization of the SCBTree to support efficient queries for some types of deformable bodies seems interesting.

Improving the hierarchy construction method would also be worthwhile. So far, we have only used a simple generic top-down tree construction method to build the BVHs. By designing specialized tree building heuristics taking the actual size and orientation of the SCB volumes into consideration during the tree creation process, for example, when the split axis is selected in a top-down tree building approach, or when grouping nodes together in a bottom-up tree building approach, we expect that hierarchies of higher quality can be created (cf. [38]). Improved construction also involves efficient handling of large polygons, or long thin polygons, which are for example common in tesselations of some simple shapes such as cones or cylinders. By improving the quality of the constructed hierarchies, N_v and N_p in Equation 1 will be lowered even further.

We also see several other opportunities to lower the memory consumption of our hierarchies than those outlined in Section 1.1. The simplest approaches include storing m polygons per leaf node instead of one, and/or switching to multiway trees with for example tertiary, quaternary, or even octonary tree nodes. For example, if a complete binary tree data structure is replaced by a complete octonary tree data structure, the number of internal nodes is reduced by a factor of seven. More aggressive memory saving can be achieved by storing the parameters defining the child volumes in relation to the parameters of their parents. Additionally, note that optimizing for speed on specific hardware involves fitting as many BVs as possible in a cache line/block. On a higher algorithmic level, it also involves designing cache aware or cache oblivious data structures and algorithms (see e.g. [51, 24]). Finally, multi-core CPUs and parallel programming methods can be utilized to give our methods a further performance boost.

Acknowledgments: Special thanks to Björn Lisper for constructive comments on a draft version of this report.

References

[1] Pankaj K. Agarwal and Micha Sharir. Efficient randomized algorithms for some geometric optimization problems. *Discrete & Computational Geometry*, 16(4):317–337, 1996.

[2] Ulf Assarsson and Tomas Möller. Optimized view frustum culling algorithms. Technical report, No. 99-3, Department of Computer Engineering, Chalmers University of Technology, March 1999.

[3] Ulf Assarsson and Tomas Möller. Optimized view frustum culling algorithms for bounding boxes. *journal of graphics tools*, 5(1):9–22, 2000.

[4] Gill Barequet, Bernard Chazelle, Leonidas J. Guibas, Joseph S. B. Mitchell, and Ayellet Tal. BOXTREE: A hierarchical representation for surfaces in 3D. *Computer Graphics Forum*, 15(3):387–396, 1996.

[5] Sergey Bereg. Cylindrical hierarchy for deforming necklaces. *International Journal of Computational Geometry & Applications*, 14(1-2):3–17, 2004.

[6] Timothy M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. In *SCG '00: Proceedings of the 16th Annual Symposium on Computational Geometry*, pages 300–309, New York, NY, USA, 2000. ACM Press.

[7] Daniel S. Coming and Oliver G. Staadt. Velocity-aligned discrete oriented polytopes for dynamic collision detection. *IEEE Transactions on Visualization and Computer Graphics*, 14(1):1–12, 2008.

[8] John Dingliana and Carol O'Sullivan. Graceful degradation of collision handling in physically based animation. *Computer Graphics Forum*, 19(3):239–248, 2000.

[9] David H. Eberly. *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics*. Morgan Kaufmann, 2001.

[10] Stephan A. Ehmann and Ming C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum,*, 20(3):500–510, September 2001.

[11] Christer Ericsson. *Real-Time Collision Detection*. Morgan Kaufmann, 2005.

[12] Bernd Gärtner. Fast and robust smallest enclosing balls. In *ESA '99: Proceedings of the 7th Annual European Symposium on Algorithms*, pages 325–338, London, UK, 1999. Springer-Verlag.

[13] Bernd Gärtner and Thomas Herrmann. Computing the width of a point set in 3-space. In *Thirteenth Canadian Conference on Computational Geometry*, pages 101–103, August 2001.

[14] Thanh Giang and Carol O'Sullivan. Approximate collision response using closest feature maps. *Computer & Graphics*, 30(2):423–431, 2006.

[15] A. Glassner. *An Introduction to Ray Tracing*. Academic Press, 1989.

[16] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. In *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 171–180, New York, NY, USA, 1996. ACM Press.

[17] Stefan Gottschalk. *Collision Queries using Oriented Bounding Boxes*. PhD thesis, 2000.

[18] Leonidas J. Guibas, An Nguyen, and Li Zhang. Zonotopes as bounding volumes. In *SODA '03: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 803–812, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[19] Taosong He. Fast collision detection using QuOSPO trees. In *SI3D '99: Proceedings of the 1999 Symposium on Interactive 3D Graphics*, pages 55–62, New York, NY, USA, 1999. ACM Press.

[20] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–15, 1962.

[21] M. E. Houle and G. T. Toussaint. Computing the width of a set. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):761–765, 1988.

[22] Philip M. Hubbard. Interactive collision detection. In *IEEE Symposium on Research Frontiers in Virtual Reality*, pages 24–31, 1993.

[23] Philip M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics (TOG)*, 15(3):179–210, 1996.

[24] Dave Kasik. Efficient data reduction and cache-coherent techniques toward real-time performance. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, New York, NY, USA, 2007. ACM.

[25] Norio Katayama and Shin'ichi Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 369–380, New York, NY, USA, 1997. ACM Press.

[26] Ladislav Kavan and Jiri Zara. Fast collision detection for skeletally deformable models. *Computer Graphics Forum*, 24(3):363–372, 2005.

[27] J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[28] Shankar Krishnan, Amol Pattekar, Ming C. Lin, and Dinesh Manocha. Spherical shell: A higher order bounding bolume for fast proximity queries. In *WAFR '98: Proceedings of the 3rd Workshop on the Algorithmic Foundations of Robotics*, pages 177–190, Natick, MA, USA, 1998. A. K. Peters, Ltd.

[29] Eric Larsen, Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha. Fast proximity queries with swept sphere volumes. Technical report, Department of Computer Science, University of North Carolina at Chapel Hill, 1999.

[30] Eric Larsen, Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha. Fast distance queries with rectangular swept sphere volumes. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3719–3726, 2000.

[31] Thomas Larsson and Tomas Akenine-Möller. Collision detection for continuously deforming bodies. In *Eurographics Conference*, pages 325–333, September 2001.

[32] Thomas Larsson and Tomas Akenine-Möller. Efficient collision detection for models deformed by morphing. *The Visual Computer*, 19(2-3):164–174, 2003.

[33] Thomas Larsson and Tomas Akenine-Möller. A dynamic bounding volume hierarchy for generalized collision detection. *Computer & Graphics*, 30(2):451–460, 2006.

[34] Thomas Larsson, Tomas Akenine-Möller, and Eric Lenguel. On faster sphere–box overlap testing. *journal of graphics tools*, 12(1):3–8, 2007.

[35] Shengjun Liu, Charlie C. L. Wang, Kin-Chuen Hui, Xiaogang Jin, and Hanli Zhao. Ellipsoid-tree construction for solid objects. In *SPM '07: Proceedings of the 2007 ACM Symposium on Solid and Physical Modeling*, pages 303–308, New York, NY, USA, 2007. ACM.

[36] Cesar Mendoza and Carol O'Sullivan. Interruptible collision detection for deformable objects. *Computer & Graphics*, 30(2):432–438, 2006.

[37] J. Mezger, S. Kimmerle, and O. Etzmuss. Hierarchical techniques in collision detection for cloth animation. *Journal of WSCG*, 11:322–329, 2003.

[38] Stephen M. Omohundro. Five balltree construction algorithms. Technical Report 89-063, International Computer Science Institute, Berkeley, California, November 1989.

[39] Carol O'Sullivan and John Dingliana. Real-time collision detection and response using sphere-trees. In *Spring Conference on Computer Graphics (SCCG'99)*, pages 83–92, 1999.

[40] Miguel A. Otaduy and Ming C. Lin. CLODs: Dual hierarchies for multiresolution collision detection. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 94–101, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[41] I. J. Palmer and R. L. Grimsdale. Collision detection for animation using sphere-trees. *Computer Graphics Forum*, 14(2):105–116, 1995.

[42] S. Quinlan. Efficient distance computation between non-convex objects. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3324–3329, 1994.

[43] S. Redon, A. Kheddary, and S. Coquillart. Fast continuous collision detection between rigid bodies. *Computer Graphics Forum*, 21(3):279–288, September 2002.

[44] J. Ritter. An efficient bounding sphere. In A. Glassner, editor, *Graphics Gems*, pages 301–303. Academic Press, 1990.

[45] J. Schwerdt, M. Smid, J. Majhi, and R. Janardan. Computing the width of a three-dimensional point set: An experimental study. *Journal of Experimental Algorithmics*, 4, 1999.

[46] Gino van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *journal of graphics tools*, 2(4):1–14, 1997.

[47] Ingo Wald, Solomon Boulos, and Peter Shirley. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Transactions on Graphics*, 26(1), 2007.

[48] Robert Webb and Mike Gigante. Using dynamic bounding volume hierarchies to improve efficiency of rigid body simulations. In *Proceedings of the 10th International Conference of the Computer Graphics Society on Visual Computing : Integrating Computer Graphics with Computer Vision*, pages 825–842. Springer-Verlag New York, Inc., 1992.

[49] Hank Weghorst, Gary Hooper, and Donald P. Greenberg. Improved computational methods for ray tracing. *ACM Transaction on Graphics*, 3(1):52–69, 1984.

[50] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In H. Maurer, editor, *New Results and Trends in Computer Science, Lecture Notes in Computer Science 555*, pages 359–370. Springer, 1991.

[51] Sung-Eui Yoon and Dinesh Manocha. Cache-Efficient Layouts of Bounding Volume Hierarchies. *Computer Graphics Forum*, 25(3):507–516, 2006.

[52] G. Zachmann. Rapid collision detection by dynamically aligned DOP-trees. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 90–97, March 1998.