CAB: Fast Update of OBB Trees for Collision Detection between Articulated Bodies

Harald Schmidl Nolan Walker Ming C. Lin
{schmidl|walkern|lin}@cs.unc.edu
Department of Computer Science
Sitterson Hall, Campus Box 3175
Chapel Hill, NC 27599-3175

Abstract

We present a new, fast approach for updating oriented bounding box hierarchies for articulated humanoid models, using a bottom up approach. The algorithm approximates existing techniques by assuming a major body axis. Existence of a major axis allows merging of bounding boxes in a hierarchy approximately but with sufficient accuracy. For scenarios with close proximity a speedup by a factor 2 on average is achieved compared to existing techniques.

1 Introduction

Collision detection algorithms often have in common the breakdown of the larger problem into subproblems: a broad phase and a narrow phase. The exact and often more expensive collision detection computation of the narrow phase is only run if a cheaper filtering effect in the broad phase has determined that this step is necessary. Bounding boxes are very helpful in the broad phase [1, 3, 5, 7]. Only if simpler boxes that enclose the original geometry are overlapping will exact collision detection be called. In this paper, we propose an efficient algorithm on updating oriented bounding box (OBB) hierarchies of articulated humanoid models for fast broad-phase culling.

2 Background

Efficient algorithms for collision detection use data structures based on hierarchical representations. Our **c**ollision handling algorithm for **a**rticulated **b**odies, CAB, utilizes results from previous work on hierarchies of axis aligned bounding boxes (AABB) [1] and hierarchies of object oriented bounding boxes (OBB) [3]. OBBs generally have a tighter fit than AABBs. OBBs for rigid bodies can be precomputed and are only transformed to body positions at runtime. It is common to find OBBs by taking the covariance matrix of the underlying geometry and use the eigenvectors of this matrix as the box axes [3]. Using the same approach to build a hierarchy at run-time, *i.e.* to merge boxes into internal parent nodes, can be fairly expensive.

For deformable or articulated bodies, the bounding volume hierarchy (BVH) must be updated as the underlying body geometry changes. Recent work [1] suggests that updating AABBs is faster. Larsson and Akenine-Möller [6] presented a way to further speed up the update of BVHs through a hybrid approach that only updates a hierarchy partially to a certain depth, going deeper on demand. However, the speedup only applies for hierarchies with many levels and when most boxes are not overlapping. Scenarios as the braided chains in figure 2(a), when handled with AABBs, will have all boxes engaged in overlap although the enclosed primitives are mostly free of collisions. Hence, all hierarchy levels have to be built on demand and such approaches no longer offer any advantage.

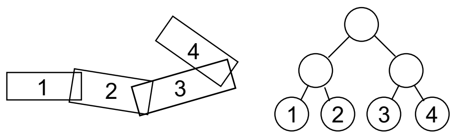
This paper presents CAB, a new bounding box fitting algorithm for OBBTrees [3]. For articulated, human-like figures, this approach achieves faster hierarchy update than hierarchies of AABBs. It can also handle close proximity scenarios where almost all internal nodes of two object hierarchies are overlapping with superior runtime performance.

3 Algorithm

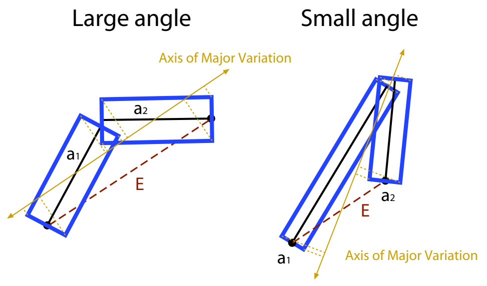
This section describes our new, efficient OBB fitting algorithm for articulated bodies. We first present our assumptions, then the mathematical background, and finally the algorithm in detail.

3.1 Assumptions

We build the bounding volume hierarchy (BVH) based on the acyclic articulation hierarchy with the rigid unit bodies in the leaf nodes. Figure 1(a) gives an example of a binary hierarchy.



(a) CAB uses a bottom up binary hierarchy.



(b) Finding parent box axes with principle component analysis and CAB's approximation.

Figure 1: Some detail on CAB's hierarchy shape and generation.

CAB assumes that the geometry comes from a humanoid model. The humanoid geometry can be visualized as a collection of ellipsoids connected with joints. The ellipsoids are elongated, *i.e.* they have a longest, or *major* axis. We assume that the ellipsoids are jointed together such that their major axes are perpendicular to the joint axis. We define an OBB as a box center $\vec{c}$, three unit vector axes $\vec{A}_i$, and three scalar extents e_i.

3.2 Algorithm Description

In general, to test if two BVHs intersect, we perform a simultaneous top-down traversal of both trees as described in [1, 3]. When a link in an articulated body moves, the entire hierarchy may need to be updated. Rigid body geometry at the leaf nodes is encapsulated with a well-fitting OBB that can be pre-computed offline. When the rigid body moves or rotates, the OBB that surrounds it is updated accordingly. CAB uses a bottom up update to refit internal nodes. We base our approach intuitively on finding the axis of major variation for a given set of data points. The CAB algorithm is designed to find a good approximation to this axis.

Consider two rigid bodies from the articulation hierarchy connected by a single joint. A very good approximation to the axis of major variation will be either one of the major axes of the child boxes in the case of small angles, or the "endpoint vector", *i.e.* the vector connecting the endpoints of the child boxes, in the case of large angles. Figure 1(b) illustrates the difference between CAB and principle component analysis. CAB picks $\vec{E}$ as the major axis for the large angle case and $\vec{a}_1$ in the small angle case.

3.2.1 Parent Box Computation

Let B_1 and B_2 be two child boxes in an articulated hierarchy connected by some joint. Let B_2 have some arbitrary rotation about its articulation point at this joint. To find the new axes of the parent bounding box we find first the longest axis of each child. Let $\vec{a}_1$ and $\vec{a}_2$ be the longest axes of B_1 and

B_2 respectively, pointing away from the joint connecting B_1 and B_2. Call the unknown parent box axes $\vec{A}_1$, $\vec{A}_2$, and $\vec{A}_3$. We first calculate the endpoint vector $\vec{E} = \vec{a}_1 - \vec{a}_2$. Next, we find the axis of major variance ($\vec{M}$) by finding the longest vector among $\vec{a}_1$, $\vec{a}_2$, and the endpoint vector $\vec{E}$. We normalize and yield

$$\vec{A}_1 = |\vec{M}|^{-1}\vec{M}. \tag{1}$$

Note: alternatively we have also tried the following[1]. If $\vec{E}$ is shortest, find $\vec{A}_1$ by taking the normalized sum of $\vec{a}_1$ and $\vec{a}_2$. However, we found this will result in a slightly larger parent box. Hence, filtering of collisions will not be as good and runtime performance suffers slightly.

The axis of minor variance immediately follows. It is the normal of the plane containing the child box axes,

$$\vec{A}_2 = |\vec{a}_1 \times \vec{a}_2|^{-1}(\vec{a}_1 \times \vec{a}_2). \tag{2}$$

The third unknown axis will then be uniquely determined by the cross product of these two perpendicular vectors,

$$\vec{A}_3 = \vec{A}_1 \times \vec{A}_2. \tag{3}$$

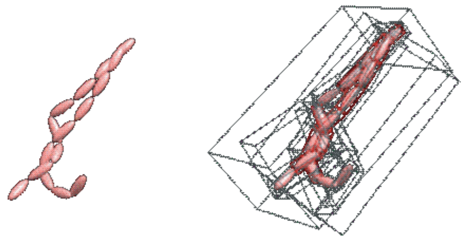
Hence we have found an orthonormal system that represents the orientation of the parent box.

There is a special case when $\vec{a}_1$ and $\vec{a}_2$ are nearly parallel, which causes $\vec{A}_2$ to not be well-defined in our original formulation. Thus, before we do any of the calculations above, we first test the dot product of $\vec{a}_1$ and $\vec{a}_2$. If this dot product is above a tolerance $(1 - 10^{-5})$, we consider the vectors parallel. When this happens, we identify the child box with the largest axis, not considering the major axes $\vec{a}_1$ and $\vec{a}_2$. The parent box uses all of this child's axes as an orthonormal basis.

Once we have found the axes, we use the child bounding boxes to find the extents of the parent box. For each parent bounding box axis $\vec{A}_i$, we project the child bounding boxes onto that axis. We examine the minimal (min_i) and maximal (max_i) values of this projection. The extent e_i of this axis and the box center $\vec{c}$ will be given as

$$e_i = \frac{1}{2}(max_i - min_i) \qquad \text{and} \qquad \vec{c} = \sum_{i=1}^{3} \frac{1}{2}(min_i + max_i)\vec{A}_i. \tag{4}$$

4 Analysis


(a) Two colliding chains with and without the boxes.

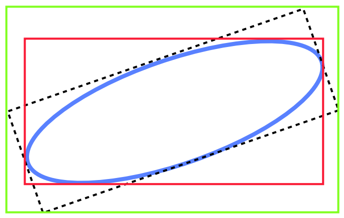

(b) Simplified AABB.

Figure 2: Some detail on CAB's hierarchy shape and generation.

This section presents experimental results and analysis. In the first set of experiments we employed an implementation of optimization-based animation [8] to generate frames of motion data for chains of varying length with bodies of different size and shape. Refer to figure 2(a) for an example of two colliding chains with and without an OBB hierarchy drawn. The final experiment uses data gathered from a virtual reality application. In all cases, the motion data is then used to test different BVH approaches. Exact collision queries are done in SWIFT++ [2] extended by DEEP [4] to detect intersection, collision normals, and the penetration depth if applicable. The reader may access MPEG videos and other related materials through the web site listed at the end of this paper.

[1]Suggested by a JGT reviewer

The timings for the following tables were taken with an implementation of our algorithms in C++. Tables 1-4 were taken on a Pentium 4 1.8GHz CPU running Windows 2000, and table 5 was taken on a 866MHz Pentium 3 laptop running Windows XP. We recorded the total number of calls to the exact query, the number of false alarms where a query returned *false*, *i.e.* no overlap, the time spent for all queries (t[query]), the time for the box hierarchy update (t[upd]), the time for testing the hierarchies for overlapping pairs of boxes (t[test]), and finally the total (t[total]). All numbers are cumulative for the number of frames that were simulated. **Note**: times are rounded to two decimal places and do not always add up exactly therefore.

	#coll	#false	t[query]	t[upd]	t[test]	t[total]
brute	1200000	1199786	19.34	0	0	19.34
$AABB^+$	71534	71320	1.86	135.86	4.94	142.67
$AABB^*$	84148	83934	1.99	8.08	5.42	15.94
OBB	5740	5526	0.16	14.63	1.75	16.54
CAB	5740	5526	0.16	6.34	2.06	8.56

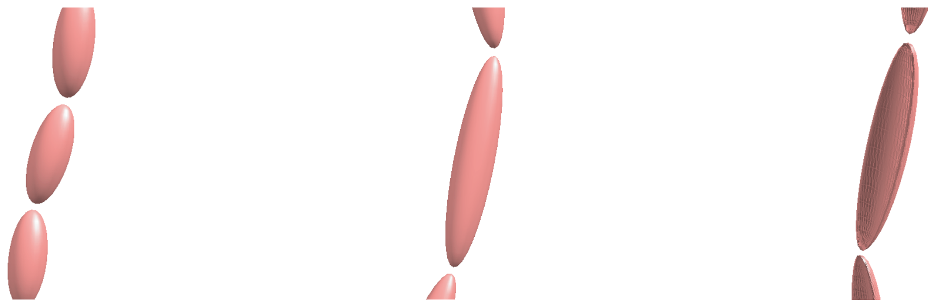
Table 1: 16 convex links, ratio 3:1, 10000 frames.

	#coll	#false	t[query]	t[upd]	t[test]	t[total]
brute	1200000	1199646	20.52	0	0	20.52
$AABB^+$	130112	129758	2.97	136.34	7.04	146.34
$AABB^*$	149466	149112	3.44	8.04	7.45	18.94
OBB	7194	6840	0.21	14.62	1.60	16.43
CAB	7194	6840	0.21	6.33	2.26	8.80

Table 2: 16 convex links, ratio 6:1, 10000 frames.

Brute-force testing and $AABB^+$ are included for completeness only. $AABB^+$ always fits boxes to all body vertices and runs highly inefficient. $AABB^*$ uses the eight vertices of a rectangular hull around the original geometry for fitting a bounding box instead of all original vertices. This case is illustrated for two dimensions in figure 2(b). Although the green box fits less tightly there is a tremendous advantage in update speed. *OBB* indicates the timing on finding the hierarchy by covariance analysis [3]. Despite of this approach resulting in better fitting boxes, *CAB* excels due the advantage in the update time. The numbers of overall queries and false alarms are the same for both *OBB* and *CAB* because the leaf boxes are exactly the same in both cases, only the internal parent nodes differ.

Tables 1 and 2 show timings for chains of bodies with different length to thickness ratios as shown in figures 3(a) and 3(b) respectively. The ratio $6:1$ in table 2 represents approximately the geometry of



(a) Convex links with length to thickness ratio 3:1.

(b) Convex links with length to thickness ratio 6:1.

(c) Non-convex links with length to thickness ratio 6:1.

Figure 3: Closeups of the different links' geometry.

	#coll	#false	t[query]	t[upd]	t[test]	t[total]
brute	1200000	1199927	22.24	0	0	22.24
$AABB^+$	130112	130039	3.71	135.79	7.13	146.63
$AABB^*$	148784	148711	3.97	8.06	7.46	19.50
OBB	3405	3332	0.34	14.70	0.85	15.88
CAB	3405	3332	0.34	6.36	1.38	8.08

Table 3: 16 non-convex links, ratio 6:1, 10000 frames.

	#coll	#false	t[query]	t[upd]	t[test]	t[total]
brute	1200000	1190472	21.37	0	0	21.37
$AABB^+$	65936	56408	3.88	136.34	4.67	144.89
$AABB^*$	93502	83974	4.55	8.11	5.64	18.29
OBB	24731	15203	2.83	14.90	3.01	20.74
CAB	24731	15203	2.80	6.35	4.01	13.17

Table 4: 16 convex links, "braided", ratio 6:1, 10000 frames.

a human lower or upper arm. It is clear that OBB approximate the body geometry better than AABBs and less dependent on length to thickness ratio.

Table 3 is for non-convex bodies. Figure 3(c) shows the dented links reminiscent of a boat hull. The ratio of length to *maximum* thickness is still 6 : 1. The AABBs have therefore almost the same size as in table 2 and the times for box testing are comparable for the two tables. However, the OBBs shrink and fit more closely. Non-convex distance queries are more expensive in SWIFT and DEEP than convex queries. Thus, the effect of better OBB filtering is more evident for non-convex bodies and is shown in the larger performance gap between $AABB^*$ and *CAB*.

The advantage of *CAB* is less noticeable in table 4. This experiment uses again the links shown in figure 3(b) but with radically different chain motion. The chains end up "braided" together in very close contact. In this table, $AABB^*$ is actually slightly faster than *OBB*. This is because of the underlying motion which generates relatively more contacts between bodies. The close contact between bodies moves the cost of *OBB* and *CAB* testing closer to that of $AABB^*$ but the advantage of faster update for *CAB* persists.

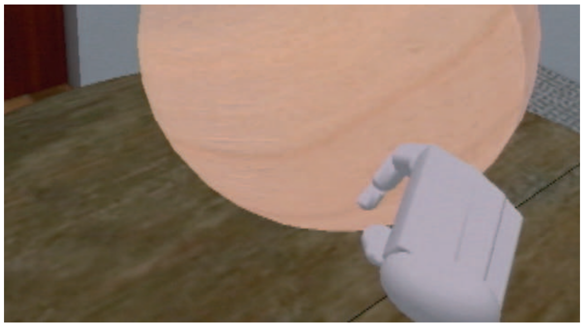


Figure 4: The hand flicks a ball over a table.

Table 5 shows timings for an articulated hand interacting with a virtual environment. See figure 4 for an articulated hand that plays with a ball. We used a Cyberglove for tracking the hand. The overall advantage of CAB persists even in this more complicated scenario with multiple branching in the links. The number of false alarms is relatively higher because the hand is not always in contact with the environment but is always close to it, *i.e.* boxes overlap without the hand touching anything.

Many applications, such as virtual reality or general animation use a frame rate of 30 frames per

	#coll	#false	t[query]	t[upd]	t[test]	t[total]
brute	288072	287308	16.08	0	0	16.08
$AABB^+$	25893	25129	7.92	68.42	1.00	77.35
$AABB^*$	27337	26573	8.79	4.48	0.88	14.16
OBB	11031	10267	5.62	6.65	0.52	12.79
CAB	11031	10267	5.65	3.01	0.56	9.22

Table 5: An articulated hand interacting with a ball, 4000 frames.

second. The total time spent for *CAB* hierarchy update, testing, and exact collision queries in our examples ranges from approximately 0.9 to 1.3 milliseconds per frame. This corresponds to 3%-4% of the frame time assuming each step takes 1/30 second. This performance shows that *CAB* is suitable for many real-time applications.

Web Materials

Mpeg videos and other supplementary materials are available online at
http://www.acm.org/jgt/papers/SchmidlWalkerLin04.

References

[1] G. Van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–13, 1997.

[2] S. Ehmann and M. C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum (Proc. of Eurographics'2001)*, 20(3):500–510, 2001.

[3] S. Gottschalk, M. Lin, and D. Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. *Proc. of ACM Siggraph'96*, pages 171–180, 1996.

[4] Y. Kim, M. Lin, and D. Manocha. Deep: An incremental algorithm for penetration depth computation between convex polytopes. *Proc. of IEEE Conference on Robotics and Automation*, pages 921–926, 2002.

[5] James T. Klosowski, Martin Held, Joseph S.B. Mitchell, Henry Sowizral, and Karel Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[6] T. Larsson and T. Akenine-Möller. Collision detection for continuously deforming bodies. *Eurographics Conference 2001*, pages 325–333, 2001. short presentation.

[7] M. Lin and S. Gottschalk. Collision detection between geometric models: A survey. *Proc. of IMA Conference on Mathematics of Surfaces*, pages 37–56, 1998.

[8] Victor J. Milenkovic and Harald Schmidl. Optimization-based animation. *SIGGRAPH 01 Conference Proceedings*, pages 37–46, 2001.