International Journal of Shape Modeling
© World Scientific Publishing Company

TEMPORAL COHERENCE IN BOUNDING VOLUME HIERARCHIES FOR COLLISION DETECTION

OREN TROPP

Technion, Department of Electrical Engineering
Haifa, 32000, Israel
troppo@tx.technion.ac.il

AYELLET TAL

Technion, Department of Electrical Engineering
Haifa, 32000, Israel
ayellet@ee.technion.ac.il
http://www.ee.technion.ac.il/∼ayellet

ILAN SHIMSHONI

Haifa University, Department of Management Information Systems
Haifa, 31905, Israel
ishimshoni@mis.haifa.ac.il
http://www.mis.haifa.ac.il/∼ishimshoni

DAVID P. DOBKIN

Princeton University, Department of Computer Science
Princeton NJ, 08544, USA
dpd@cs.princeton.edu
http://www.cs.princeton.edu/∼dpd

Collision detection is a fundamental problem in computer graphics. In this paper, temporal coherence is studied and an algorithm exploiting it for bounding volume hierarchies, is presented. We show that maintaining some of the intersection tests computed in the previous frame, along with certain information, is able to speedup the intersection tests considerably. The algorithm is able to accelerate the collision detection for small motions and works as fast as the regular algorithm for large motions, where temporal coherence does not exist. The algorithm framework can be implemented for any type of bounding volume hierarchy. To demonstrate this, it was implemented for the OBB and the AABB data structures and tested on several benchmark scenarios.

Keywords: Collision detection, bounding volume hierarchies, OBB, AABB

1. Introduction

Collision detection is a fundamental problem in computer graphics, as well as in computational geometry, solid modeling, robotics and molecular modeling [1,2]. The goal is to find efficient techniques for determining whether objects intersect.

Of the many algorithms and data structures used in collision detection [3,4,5,6,7], we focus on the *Bounding volume hierarchy* representation, which is a simple and popular data structure. The attractiveness of this data structure stems from the fact that it gives rise to a fast "rejection test". We explore this representation in a dynamic environment.

There are several possible shapes used in bounding volume hierarchies, such as spheres [8,9], cones [10,11], prisms [12], k-DOPs [13], axis aligned boxes (AABB) [14] and oriented bounding boxes (OBB) [15,16].

Our basic premise is that in dynamic environments, objects comprising the scene do not move much between consecutive frames [17,18,19,13,20]. This *temporal coherence* implies that collision queries are typically dependent on queries made in previous frames.

Temporal coherence for collision detection was introduced in [17], where Voronoi diagram based data structures were used. In [13], a temporal coherence data structure for bounding volume hierarchies, the *front*, was propose for k-DOPS. This data structure was further investigated in [20] for the spherical bounding volume data structure and later in [19] for a convex hull based data structure. The current work succeeds these methods, extending the technique, generalizing it for any bounding volume hierarchy and implementing it for OBB and AABB.

The underlying assumption is that most of the bounding volume intersection tests performed in the current frame were performed in the preceding frame as well. Moreover, the result of this test, *intersection* or *disjointness*, is the same most of the time. Therefore, rather than processing each collision query from scratch, by comparing the roots of the bounding volumes and expanding nodes downward, the algorithm "remembers" which collision tests determined the result before, and starts there. Thus, less bounding volume intersection tests are performed. To support the scheme, we use a *Bounding Volume Test Tree*, proposed in [19,20], which represents the collision tests performed thus far and the *front* data structure [19,13,20], which maintains the set of intersection tests that determine the collision between the objects.

There are various possible ways to update the front [20,19]. The question remains whether certain strategies are better than others. The current paper attempts to answer this question. Rather then developing a strategy depending on frames, our strategy depends on a bound on the motion of the objects. The paper provides some empirical evidence to support the proposed strategy.

To accelerate the intersection tests performed on the front nodes in the next frame, a novel *coherence intersection test* is introduced. This test utilizes information saved from the previous frame to speedup the current intersection test.

In our experiments, we use meshes composed of up to hundreds of thousands of polygons. The models move and can come in close proximity of each other. We show that our algorithm can run up to twice as fast as the equivalent non-coherence algorithm for OBB. Moreover, we show that collision detection is accelerated as the number of frames increases (for the same motion). Thus, the smoother the animation, the greater the benefit of our algorithm.

The paper makes the following contributions:

(1) A coherence intersection test is introduced. Rather than just maintaining the front,

some additional data, generated in the previous intersection test, is stored and used to speed up the test in the next frame.

(2) A statistical analysis of temporal coherence for bounding volumes and its relation to the size of the objects and the motion between frames, is provided .
(3) Front updating policies are devised empirically from runs on several scenes. Unlike previous work, which utilize frame-based policies, the proposed policies depend on the motion performed between frames. Thus, the policies can deal with small and large motions.
(4) Based on the statistical observations, it is proposed that when large motion is detected, the algorithm should revert automatically to the basic (non-coherence) algorithm until it detects smaller motion again. Thus, the algorithm always performs at least as well as the non-coherence algorithm.
(5) The coherence scheme is implemented for the OBB and AABB data structures.
(6) The insight gained from the statistical observations suggests that experiments on coherence algorithms should be run as a function of motion size.

The rest of the paper is structured as follows. Section 2 describes the data structure and the operations supported for it. Section 3 presents the coherence-based collision detection algorithm. Section 4 describes the coherence intersection test. Section 5 elaborates on determining the algorithm's control parameters that yield the best performance. The experimental results are described in Section 6. We conclude in Section 7.

2. Data Structure

This section describes the major data structures used in the algorithm. It starts with a general review of bounding volume hierarchies, then describes the bounding volume test tree and in particular, the front. Finally, it describes the operations for updating the front of the bounding volume test tree.

Bounding Volume (BV) Hierarchies:

A BV hierarchy is one of the most prevalent data structures for collision detection, which gives rise to fast "rejection". In this representation, each node in the hierarchy is a bounding volume that bounds a sub-object and in particular, a sub-mesh. Each internal node bounds all the mesh faces bounded by its two children, the root bounds the whole mesh, and a leaf bounds at least one triangle. Sibling nodes might overlap, yet each face belongs to only one node at each hierarchical level.

Given two BV hierarchies, a collision test is performed top-down. Initially, the roots are tested for disjointness. In the affirmative case, the process stops. Otherwise (intersection), the descendants of one node are tested against the BV of the other, and the process recurses. When the BVs of the leaves intersect, a primitive-primitive (i.e., triangle-triangle) collision test is performed.

The *Axis Aligned Bounding Box (AABB)* [14] and the *Oriented Bounding Box (OBB)* [15,16] are popular bounding shapes. An AABB bounds the underlying geometry with the minimal bounding box aligned with the major axes of the local (object's) coordinate system. Obviously, constructing an AABB hierarchy is simple and fast. However, the AABB fails to tightly bound the underlying geometry. An OBB hierarchy is more flexible in choosing the orientation of the bounding box. Though this representation is more complex to compute and check for box-box intersection, the geometry is tightly bounded and thus less intersection tests are performed overall.

Bounding Volume Test Tree (BVTT):

The intersection tests that should be performed for determining a collision between BV hierarchies, can be described as a binary tree, the *Bounding Volume Test Tree (BVTT)* [18]. In this tree, every node represents a single BV-BV intersection test. For instance, the root of the tree represents the intersection test between the roots of the BV hierarchies of the given objects.

In a BVTT, every internal node represents a BV-BV intersection test that results in an intersection. A leaf represents either a BV-BV intersection test between disjoint boxes or a true intersection between triangles. We denote each BVTT node as *intersecting* or *disjoint*, according to the result of the intersection test represented by this node.

The Front:

A front of the BVTT is a subset of the tree nodes which satisfies the condition that every path from a leaf to the root contains a single node from this subset. For instance, the simplest front is the root of the BVTT. The largest front contains all the leaves of the BVTT and only them. Figure 1 illustrates four possible fronts of a single BVTT.

The importance of the front stems from the fact that the temporal coherence algorithm suggests to start the collision query from the nodes in the previous front rather than from scratch, as customary. For example, if the objects have not moved since the last frame, collision can be determined only by repeating the leaf tests of the BVTT, which can be further accelerated by using the coherence intersection test (discussed below). Skipping the internal nodes of the BVTT has an added benefit, since these nodes represent tests that result in intersections between BVs, which are typically more computationally expensive than tests resulting in disjointness.

For instance, suppose that a collision detection procedure requires the tests represented by the BVTT in Figure 2. If the search starts from the front in Figure 1(a), 13 BV-BV intersection tests will be required (as done in the non-coherence algorithm). Starting from the front in Figure 1(b) requires 12 tests, from 1(c) – 9 tests, and from 1(d) – 10 tests.

Two changes might occur in the front between consecutive frames. First, a disjoint front node might become intersecting, thus requiring the expansion of its descendants (the down arrows in Figure 1, with respect to Figure 2). In this case, this node should *sprout* (or drop [19]) new nodes, and the algorithm should recurse on these nodes. Second, an ancestor of a front node in the BVTT might become disjoint and thus can be added to the front, replacing its descendants, as illustrated in Figure 1(d), where the nodes with the up arrows

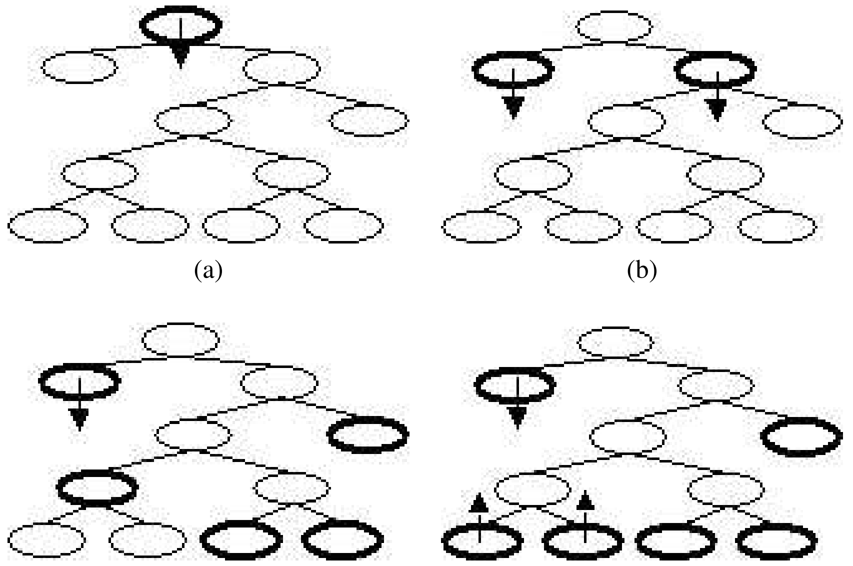


Fig. 1. Four possible fronts of a BVTT (the black nodes). The down (up) arrows mark nodes that should sprout (be pruned) in the next frame, shown in Figure 2

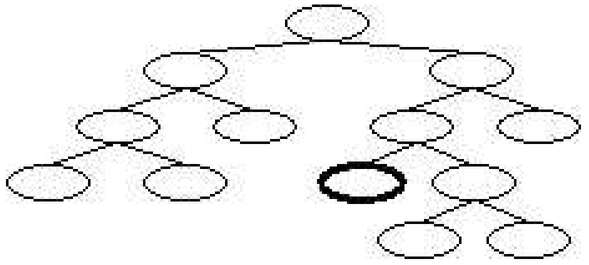


Fig. 2. The *BVTT* of the next frame

can be replaced by their parent (highlighted in Figure 2). In this case we say that the front is *pruned* (or raised).

The *optimal front* contains the lowest collection of nodes that exist both in the current BVTT and in the next BVTT. Maintaining a good front plays a crucial role in the efficiency of our algorithm.

Two issues should be addressed in this regard. First, if a previously-disjoint front node becomes intersecting, should the front be updated (by sprouting)? Second, if two sibling nodes in the front become disjoint, should the algorithm attempt to prune these nodes? Though intuitively the answers to these questions are affirmative, too many updates to the front add computations. This trade-off and the conditions for pruning and sprouting are discussed below.

Both [20] and [19] suggest update strategies for the front. Both suggest to sprout the front up to the level at which the BV's are disjoint or that they are leaves. As for pruning, in [20] the front is not pruned. If in two or three consecutive frames the front does not increase in length, the front is rebuilt from scratch. In [19], on the other hand, it is suggested to try to prune the tree one level at a time if two sibling nodes do not intersect and attempt to replace them with their parent. Both [20,19] do not provide empirical evidence as to the quality of these strategies.

Rather then developing a strategy that depends on frames, our strategy depends on a bound on the motion of the objects. Empirical evidence demonstrates that the magnitude of the motion is directly related to temporal coherence. The motion magnitude is also used to reason about the intersection between BV's.

Sprouting:

Given that a disjoint front node becomes intersecting, should the front be updated? Intuitively, if the node's descendants belong to the BVTT of the next frame, it is beneficial to sprout the front now, because it will save intersection tests. Unfortunately, the algorithm has no knowledge regarding the next frame. Therefore, the probability that an intersecting node remains intersecting should be evaluated.

Let C_A be the cost of keeping node A in the front (and processing its subtree), c_A be the cost of testing node A for intersection, and C_D be the cost of putting the node's descendants in the front and processing them. Let P_A be the probability that node A should be in the front in the next frame. Then,

$$C_A = P_A \cdot (c_A + C_D) + (1 - P_A) \cdot c_A = c_A + P_A C_D. \tag{1}$$

A should be kept in the front iff the expected cost
$C_A < C_D \Leftrightarrow P_A < 1 - c_A/C_D$.

Estimating P_A (using Minkowski sums) leads to unavailing expressions. Moreover, C_D is unknown. Therefore, a strategy based on this expression is impractical.

Instead, the ratio between the size of the bounding volumes and a bound on the magnitude of the motion from the previous frame is studied. The intuition is that an intersecting node is likely to repeat only if its two boxes are "large" relative to the movement size. In addition, this node should be close in the hierarchy to the determining intersection test nodes. The details are described in the next section.

Pruning:

Pruning is vital for avoiding a situation in which the front can grow but can never shrink. A front is not as efficient as it should be if it contains two siblings whose parent is disjoint. Pruning the two siblings produces a smaller front for the next frame and saves a test. When should the algorithm perform pruning?

Attempting to prune the front nodes is expensive since it requires a *non-coherence intersection test* of the parent node. If the parent turns out to be intersecting, the test was performed in vain.

The goal is to balance the high cost of attempting to prune the front and the cost of maintaining a good front. We tried several strategies. The approach that yields the best results is to attempt to prune the front when the objects have moved substantially relative to the movement size, since the last pruning attempt. The intuition is that pruning is likely to be successful when the BVTT has changed considerably. The exact condition is described in the next section.

3. Coherence-Based Collision Detection Algorithm

The key observation is that due to temporal coherence, the front of the previous frame resembles the front of the current frame. Thus, rather than performing all the intersection tests described in the BVTT tree (i.e., executing the non-coherence algorithm), the algorithm starts from the front of the previous frame. Obviously, this front does not always match the new BVTT tree and thus needs to be modified by pruning or sprouting.

The magnitude of movement between consecutive frames dominates the changes between their BVTT trees. Intuitively, if the amount of movement is large relative to the size of the BVs tested for intersection, the test result can change from disjointness to intersection and vice versa. Moreover, when large motions occur, the BVs belonging to BVTT nodes may also change, yielding the coherence information useless. Thus, the magnitude of the global motion has to be monitored to help decide when to use the coherence information and when to disregard it.

Each disjoint node in the front contains additional data that can accelerate the disjointness test in the next frame. This data includes a lower bound, *margin*, on the distance between the BVs tested in this node and additional information regrading the separation between the nodes (e.g., their separating plane).

The algorithm proposed is fast not only because less intersection tests are performed, but also because the stored data enables it to perform faster BV-BV intersection tests.

We now turn to the general coherence algorithm whose course is described in Algorithm 1. We will then elaborate on each step.

Let e_o be the “size” of the smaller object’s bounding volume and e_1, e_2 be the “size” of the two bounding volumes in a given BVTT node. In the case of boxes, e_o, e_1 and e_2 are the longest edges of the corresponding bounding boxes. Let $\left|\Delta x_{frame}\right|$ be an upper bound on the magnitude of the movement of a point on the object since the last frame.

Stage 1 (lines 1-5): When the motion (Δx_{frame}) is considered large with respect to the size of the smaller object (e_o) or in the first frame, temporal coherence cannot be used and the original non-coherence algorithm needs to be applied. Let $P_{coherence}$ be an experimentally determined threshold parameter (Section 5). A motion is considered large relative to the object’s size if

$$\left|\Delta x_{frame}\right| / e_o > P_{coherence}. \tag{2}$$

When the non-coherence algorithm is used, BVTT nodes are inserted to the front and their coherence data is updated. As explained above, this is done by studying the ratio between the size of the bounding volumes $e_i, i = 1,2$ and a bound on the magnitude of the

Algorithm 1 Algorithm overview (for each frame)

```
1:  if the motion is large then
2:      Call the no-coherence algorithm
3:      Sprout nodes in the front and update coherence data
4:      Exit
5:  end if
6:  for each node in the front do
7:      if the node was disjoint then
8:          Check whether it is still disjoint using coherence (4.2)
9:      else if the node was intersecting then
10:         Transform the coordinate systems
11:         Perform full non-coherence BV-BV test for the node (4.1)
12:     end if
13:     if the node is disjoint in the new frame then
14:         Keep it in the front
15:     else if the node is intersecting in the new frame then
16:         Expand the node with the non-coherence algorithm
17:         Update the front according to the sprouting policy
18:     end if
19: end for
20: if the movement since last pruning is substantial then
21:     Attempt to prune the tree
22: end if
```

motion from the previous frame $|\Delta x_{frame}|$. Since we would like to descend the BVTT as much as possible, the criterion is to keep in the front the lowest BVTT node such that

$$\left|\Delta x_{frame}\right|/e_i > P_{sprout}, \quad \text{for } i = 1,2, \tag{3}$$

where P_{sprout} is a threshold. The values of P_{sprout} for different types of BVs are determined experimentally in Section 5. In the case that the front building process reaches a BVTT leaf node, it is always inserted into the front. The empirical results given in Section 5 show that a BVTT-leaf only front ($P_{sprout} \to \infty$) is optimal.

Stage 2 (lines 7-8): If the node was disjoint in the last frame, a *coherence BV-BV intersection test* is performed. Specifically, if $margin > \left|\Delta x_{frame}\right|$, then the BVs are guaranteed to still be disjoint. If this test fails, the next attempt is to separate the BVs using the same separating axis found in the last frame. This test is BV type specific. Only if both of these tests fail, an expensive BV-BV intersection test is performed. Details of the implementation of this stage for the OBB and AABB data structures are described in Section 4.

Stage 3 (lines 9-11): A node is considered intersecting in two cases: either it was intersecting in the last frame or the coherence intersection test (Stage 2) resulted in intersection. In this case, the algorithm performs a full non-coherence BV-BV test for the node (Section 4.1). This test should be preceded by transformation of the coordinate system.

Stage 4 (lines 13-14): If the node is disjoint in the new frame, it is maintained in the front.
Stage 5 (lines 15-18): If the node is intersecting in the new frame, it is expanded by the non-coherence algorithm. The algorithm needs to determine whether to insert the node into the front according to Equation 3.
Stage 6 (lines 20-22): After all the nodes of the front have been processed, the algorithm determines whether to attempt to prune the front. Let Δx_{prune} be an upper bound on the movement since the previous pruning. A pruning attempt is performed on the whole front when:

$$\left|\Delta x_{prune}\right| / e_o > P_{prune}. \tag{4}$$

P_{prune} is determined experimentally in Section 5.

4. Coherence Intersection Test

The framework described above is general for bounding volume hierarchies. For each specific BV type, a different coherence intersection test (used in Stage 2 of the algorithm) has to be developed and the data that is stored in the front nodes changes accordingly. The challenge is to design a test that achieves a considerable speedup with respect to the non-coherence BV-BV intersection test.

To demonstrate the general applicability of our approach, we present in this section the coherence intersection tests for the OBB and AABB data structures.

The section starts by describing the standard non-coherence intersection test. Then, a margin-coherence test that uses a bound on the movement from the previous frame to check for disjointness, is described. The section concludes with the axis-coherence test, which is specific for each data structure and assumes that the same separating axis which separated in frame f is still separating in frame $f+1$. Only if these two coherence tests fail, the expensive non-coherence test is performed.

4.1. *Non-coherence box-box intersection test*

The *separating axis theorem (SAT)* states that if two convex polytopes in 3D are disjoint, then there exists a separating plane between them that is either parallel to a face of either polytope, or parallel to an edge from each. The separating axis is the normal of this plane. When the vertices of the polytopes are projected to this axis, the intervals they define do not overlap. In the case of boxes, there are 15 potential separating planes.

In [15], this test is efficiently performed as illustrated in Figure 3. Axis L is separating if the distance between the projections of the boxes on it are disjoint. We briefly repeat the description of the test as presented in [15].

Given two boxes A and B. Let A^i (B^i), $i = 1,2,3$, be the unit vector directions of the edges of box A (B). Let a_i (b_i) be half the length of edge i. Let R be the rotation matrix and T the translation vector that define the transformation from the coordinate system of A to that of B.

The algorithm works in the coordinate system whose origin is the center of box A and the three axes are the edge directions of A. In this coordinate system, matrix $[A^1, A^2, A^3]$ is

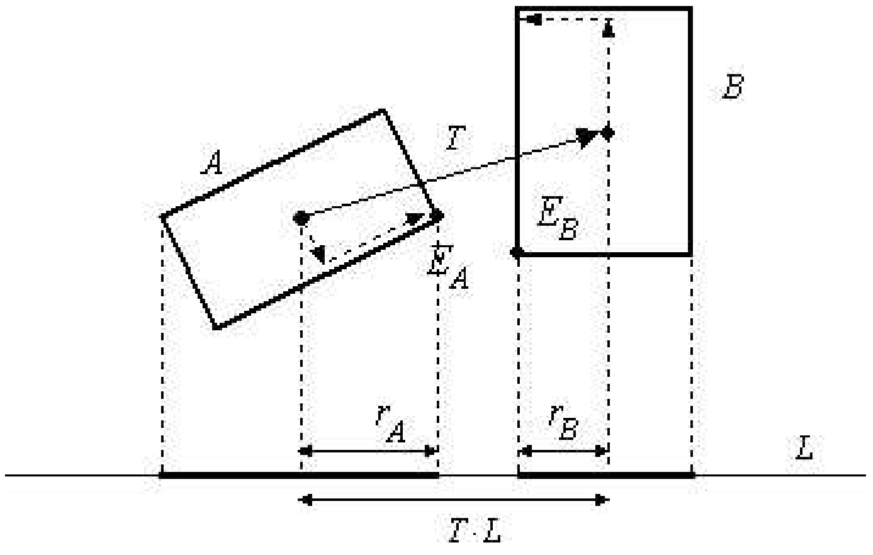


Fig. 3. L is separating for boxes A and B because their projection intervals on L are disjoint.

the unit matrix, $R = [B^1, B^2, B^3]$ and T is the vector connecting the centers of the boxes.

We define the *radius of the projection* r_A (r_B) to be:

$$r_A = \sum_{i=1}^{3} \left| a_i A^i \cdot L \right|, \quad r_B = \sum_{i=1}^{3} \left| b_i B^i \cdot L \right|. \tag{5}$$

The distance between the projections of the centers of the boxes is $|T \cdot L|$. Thus, Axis L is a separating axis **iff**

$$|T \cdot L| \geq \sum_{i=1}^{3} \left| a_i A^i \cdot L \right| + \sum_{i=1}^{3} \left| b_i B^i \cdot L \right| = r_A + r_B. \tag{6}$$

This expression can be simplified when L is a direction of an edge or a cross product between edge directions. For example, if $L = A^1 \times B^2$, then

$$\left| a_2 A^2 \cdot (A^1 \times B^2) \right| = \left| a_2 B^2 \cdot (A^2 \times A^1) \right| = a_2 \left| R_{32} \right|.$$

The last equality stems from the fact that in the coordinate system of A, the columns of the rotation matrix are the edge directions of box B. Using similar simplifications, the expression in Equation 6 is reduced to:

$$\begin{aligned} &|T_3 R_{22} - T_2 R_{32}| > \\ &\qquad a_2 |R_{32}| + a_3 |R_{22}| + b_1 |R_{13}| + b_3 |R_{11}|. \end{aligned}$$

4.2. *Margin-coherence box-box intersection test*

For each node in the front, a variable *margin*, defined as the difference of the inequality of Equation 6, is maintained:

$$margin = |T \cdot L| - r_A - r_B. \tag{7}$$

margin is the size of the separating interval on the separating axis L, whose end points are projections of a vertex of A and a vertex of B (marked E_A, E_B in Figure 3). Thus,

$$margin = |(E_B - E_A) \cdot L| \tag{8}$$

Being a projection, *margin* does not provide the exact distance between E_A and E_B. However, since L is a unit vector, the real distance is at least the margin.

Assume that between frames f and $f+1$, object B rotates by $\Delta\alpha$ (about some axis passing through the box's center) and translates by Δt. Then, the movement of vertex E_B, ΔE_B, is bounded by

$$|\Delta E_B| \leq \left|\Delta x_{frame}\right| = |\Delta t| + \Delta\alpha \cdot b, \tag{9}$$

where b is the maximal distance from the center of the bounding box of B to one of its vertices and Δx_{frame} is the maximal possible movement between the frames for any point on object B. Since object A is stationary and no point on box B moves more than Δx_{frame}, *margin* cannot change by more than Δx_{frame}. Formally:

$$\begin{aligned} margin^{f+1} = \left|(E_B^{f+1} - E_A^{f+1}) \cdot L\right| = \\ \left|(E_B^{f} + \Delta E_B - E_A^{f}) \cdot L\right| \geq \\ \left|(E_B^{f} - E_A^{f}) \cdot L\right| - |\Delta E_B \cdot L| \geq \\ \left|(E_B^{f} - E_A^{f}) \cdot L\right| - \left|\Delta x_{frame} \cdot L\right| \geq \\ \left|(E_B^{f} - E_A^{f}) \cdot L\right| - \left|\Delta x_{frame}\right| = \\ margin^{f} - \left|\Delta x_{frame}\right|. \end{aligned} \tag{10}$$

Therefore, a lower bound on $margin^{f+1}$ can be computed from the approximated margin of the previous frame, $\widehat{margin^f}$, by:

$$\widehat{margin}^{f+1} = \widehat{margin^f} - \left|\Delta x_{frame}\right|.$$

If $\widehat{margin}^{f+1}$ is positive, then the boxes are still disjoint, along the same separating axis as in the previous frame. However, if $\widehat{margin}^{f+1}$ is negative, a more accurate test must be performed in order to confirm disjointness. It is quite plausible that the same axis which separated the two boxes, still separates them. Thus, this axis is tested first. If this fails, the full intersection test is performed on the two boxes.

Note that $\left|\Delta x_{frame}\right|$ is the same for all the boxes of object B for a given frame. Thus, this margin test is extremely fast and does not require to manipulate and transform the points from one coordinate system to the other.

4.3. *Axis-coherence test*

It is hereby shown how to efficiently perform an intersection test between boxes A and B in frame $f+1$, for OBB and AABB, assuming an intersection test was performed in frame f and resulted in disjointness. For convenience, assume that A remains stationary between the frames and only B moved. (It is always possible to transform the coordinate systems such that the relative movement between A and B becomes a movement of B alone.)

The coherence hypothesis is that the same separating axis found in frame f, is also separating in frame $f+1$. This hypothesis is checked by recomputing all the elements in Equation 6 (r_A, r_B and $|T \cdot L|$), for the new frame, in a way that relies on the saved data.

OBB:

In this case, for each front node, the previous separating axis, L^f, is maintained as a 3×1 unit vector. We keep L^f in the coordinate system of object B. Thus, when B rotates, so does L^f. After a rotation between the frames, the rotated L^f is not necessarily one of the 15 separating axes of the SAT theorem, nevertheless it can still separate disjoint objects.

Observe that r_B does not change from frame to frame because all the terms in Equation 5 rotate together as part of object B. Therefore, r_B can be saved and need not be computed again.

In order to compute r_A, L is transformed from the coordinate system of B to that of object A, and then once again from the coordinate system of A to that of the specific box in A. This requires two matrix-vector multiplications.

Let T_A and T_B be the box centers of A and B, respectively. $T \cdot L$ is computed by:

$$T \cdot L = (T_A - T_B) \cdot L = T_A \cdot L - T_B \cdot L.$$

This holds for any coordinate system because dot products are invariant to coordinate system transformations. $T_A \cdot L$ is calculated in the coordinate system of A, since both T_A and L are already known there. As for $T_B \cdot L$, in the coordinate system of A:

$$\begin{aligned} T_B &= T_{Bint} + T_{BA} \\ L \cdot T_B &= L \cdot (T_{Bint} + T_{BA}) = L \cdot T_{Bint} + L \cdot T_{BA}, \end{aligned}$$

where *Bint* is the position of a box in B's hierarchy relative to the origin of object B and T_{BA} is the vector connecting the origins of the two objects.

Both dot products can now be calculated in the coordinate system of our choice. T_{Bint} and L are already known in the coordinate system of B. T_{BA} is the same for all the boxes in a given frame. Thus, it is preferable to compute the dot products in the coordinate system of B as well.

Transformation between coordinate systems:

Since typically each object is defined in its local coordinate system, the two objects need to be transformed into a common coordinate system before the intersection test (Stage 3).

While often overlooked, this transformation is computationally expensive and has to be taken into account when evaluating the cost of an intersection test.

It is impossible to maintain the placement of each box relative to its parent, as customary, because a box is often accessed without previously encountering its parent. Instead, the data of each node is maintained in the coordinate system of the object. This results in an extra computational cost when a test is performed without coherence. However, this cost is more than compensated for by the savings achieved by the coherence test.

Special case - no rotation:

In several applications, it is common that the change between consecutive frames is translation alone. In this case, the computation is further simplified. Since r_A and r_B (Equation 5) do not change and in Equation 6, $T^{f+1} = T^f + \Delta t$, where Δt is the translation between frames, the new margin can be computed by:

$$\begin{gathered} margin^{f+1} = L \cdot T^{f+1} - r_A - r_B = \\ L \cdot (T^f + \Delta t) - r_A - r_B = margin^f + L \cdot \Delta t. \end{gathered} \tag{13}$$

This efficiently computes the margin for the separating axis (L^f) in frame $f+1$ and can therefore accurately predict if L^f is still separating. It can be used to replace the longer axis-coherence procedure described above.

AABB:

To make the best of coherence, a variant of *AABB* that uses *SAT lite* [14] is used. In this variant, only six out of the fifteen candidate separating planes are tested for separation. These are the face normals of the boxes. Although the intersection test might err and report on false intersections, 95% of the disjoint pairs are detected [14]. Moreover, this is a conservative test, since disjointness will be detected while descending the BV hierarchy. Therefore, while the number of box-box tests is not significantly increased, the computational complexity of each intersection test is considerably reduced.

In the non-coherence case, $T \cdot L$ should be tested for separation (Equation 6). Because the boxes move and rotate, T has to be recalculated at each frame, requiring an expensive matrix-vector product.

Since the separating axis checked in *SAT lite* is always an edge of a box, we observe that in the coordinate system of this box, L has only one non-zero element, i. Thus, $T \cdot L$ is reduced to T_i. As a result, rather than transforming the vector T using matrix operations, only T_i needs to be computed. Since this can be done only in the coordinate system in which L is an edge direction and L might come from either box, the box whose coordinate system is used should be the box that generated the separating axis. Suppose this box is A.

Once $T \cdot L$ has been computed, r_A is simply equal to a_i and r_B is computed using Equation 5. If L is still separating, only one axis direction is tested, instead of 4.2 axes on average [14].

5. Empirical Analysis of Optimal Parameters

The algorithm described in Section 3 has three numerical parameters that determine the use of coherence and the optimization of the front: $P_{coherence}$, P_{sprout} and P_{prune}. We experimentally tested these parameters by calling the algorithm with different values on various scenarios. The values achieving the highest speedups, are used in our experiments, avoiding any user tuning.

In order to be able to present the results of the different scenarios in the same graph, the running times are presented as the ratio between the coherence algorithm and the original (non-coherence) algorithm. That is to say, minima in the graphs achieve the highest speedups.

Coherence threshold:

If the movement from the last frame is large relative to the size of the moving object, the algorithm disregards the coherence data and starts from scratch using the non-coherence algorithm (Stage 1). Figure 4 shows that the optimum lies around $P_{coherence} = 0.14$ for the OBB and 0.2 for the AABB. That is to say, when the movement is larger than 0.14 of the object's size, coherence should not be used for the OBB.

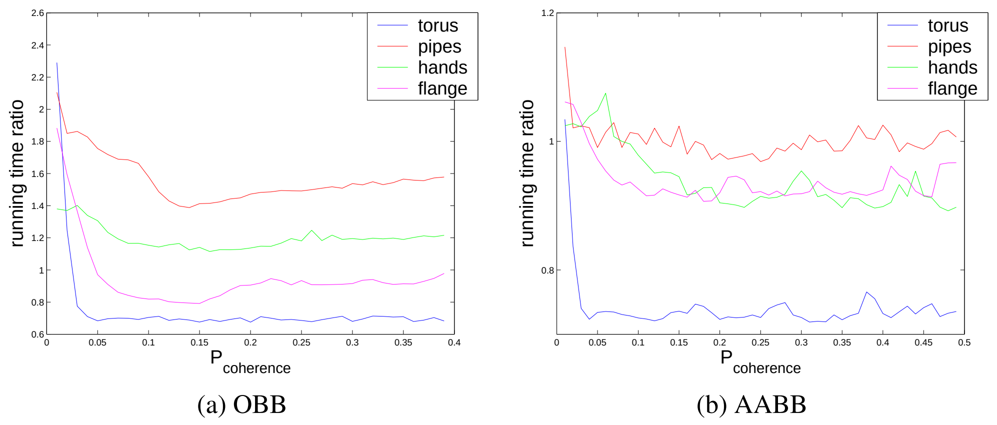


Fig. 4. When to use coherence?

Similar results are observed in Figures 8-9, where the coherence algorithm is shown to be faster than the original algorithm for movements smaller than $P_{coherence} \cdot e_o$.

Sprouting:

The parameter P_{sprout} helps to determine how far down the BVTT the front should be. For OBB, the results which show the running time as the function of the value of P_{sprout} are shown in Figure 5. The running time decreases as a function of the value of P_{sprout}. Therefore, a leaves-only front strategy, which is also simpler to implement, is used.

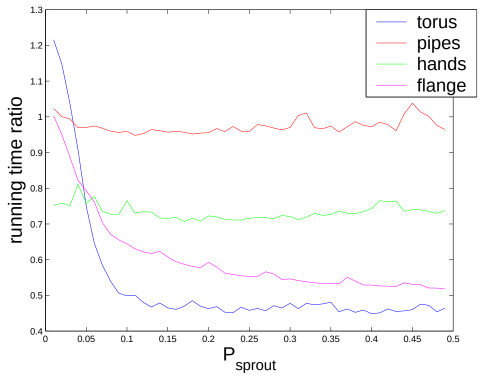


Fig. 5. Performance of OBB as a function of P_{sprout}

For AABB, a bounding box of a single triangle is a 3D box (in contrary to the OBB, where it is a 2D box). Therefore, a box is an extremely poor bounding volume for a triangle, resulting in many erroneous *intersecting* results. As a result, tree expansion often continues until the triangle-triangle test, where no efficient coherence test exists. The sprouting policy is thus to keep in the front the lowest-level boxes (rather than triangles), because boxes have coherence data and a fast disjointness test.

Pruning:

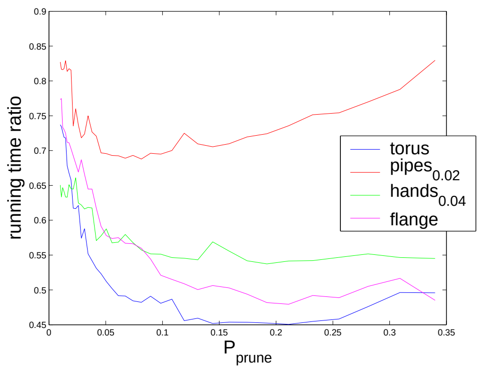
The parameter P_{prune} determines when the algorithm should attempt to prune the front. This is done when the relative motion since the previous pruning is larger than P_{prune} (Stage 6). Figure 6 shows the performance as a function of P_{prune}. The optimum is achieved around $P_{prune} = 0.17$ for the OBB data structure and $P_{prune} = 0.15$ for the AABB data structure. These values are stable across the different scenarios and are therefore used by us in our experiments.

6. Experimental Results

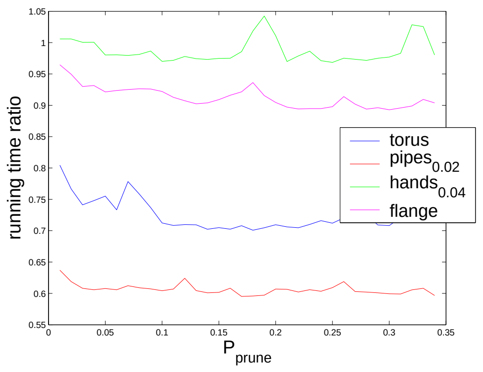
This section describes the results obtained by our algorithm for the OBB and AABB data structures. In the implementation, the RAPID software package is used [21]. For AABB, the *SAT lite* algorithm is used with the necessary code modifications [14]. The tests are executed on a Pentium 4 1.8Ghz processor. The code is compiled with Microsoft visual studio 6.0 under the standard optimizations for speed.

It is important to note that in the experiments, our coherence based algorithms are compared to their non-coherence variants. We do not attempt to choose the best bounding volume (for which there are mixed results in the literature) but rather, to show the advantages of temporal coherence.

As before, the running times are presented as the ratio between the coherence algorithm and the original (non-coherence) algorithm. Movement sizes are given in units of

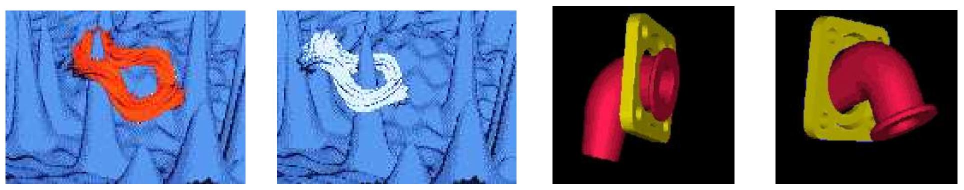


(a) OBB

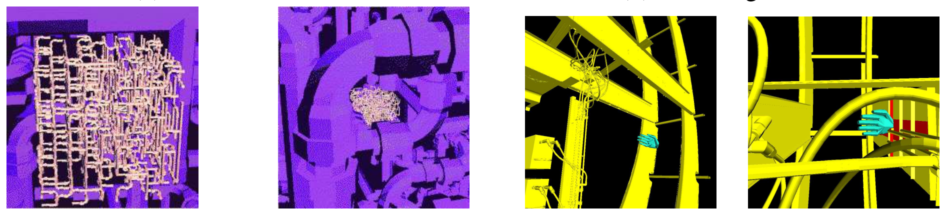


(b) AABB

Fig. 6. Performance as a function of P_{prune}



(a) The torus model (b) The flange model



(c) The pipes model (d) The hand model

Fig. 7. Benchmarks

$\Delta x_{frame}/e_o$ (i.e., the motion size relative to the size of the moving object).

In our experiments, several standard benchmarks are used, each consisting of a moving object within a stationary environment (Figure 7). Table 1 describes some characteristics of the scenarios. For each scenario, the number of triangles of the stationary environment and the moving object are given. For each path, the number of frames, the median movement size and the percentage of movements which are larger than 10% of the moving object's largest edge of its bounding box (e_o), are displayed.

In the original paths, the motion size varies considerably from frame to frame. As coherence is highly dependent on the size of the motion, the algorithm should be tested as a function of a typical movement size. Therefore, for each benchmark scenario, similar paths, each having a different motion size, are produced by interpolating the original path.

Model	Pipes	Torus	Hand	Flange
Triangles Env	143690	98114	169944	990
Triangles Obj	143690	20000	404	5306
Number of frames	3954	3001	2528	1018
AABB:				
Median motion size	0.133	0.018	0.148	0.16
% of large motions	61.52	0	60.15	71.48
OBB:				
Median motion size	0.09	0.018	0.147	0.137
% of large motions	43.21	0	60.15	62.9

Table 1. Characteristics of the scenarios

Each test is performed 70 times and the results are averaged.

Figure 8 compared our OBB algorithm to the original algorithm for some typical movements. As expected, the smaller the movement, the more beneficial the coherence algorithm is. For instance, for a movement of 0.05 of the object's size, our algorithm is twice as fast as the original non-coherence algorithm for the hand scenario. This experimental result is important, since it predicts the efficiency of our algorithm for new scenarios with a known movement distribution. Figure 9 shows our results for AABB.

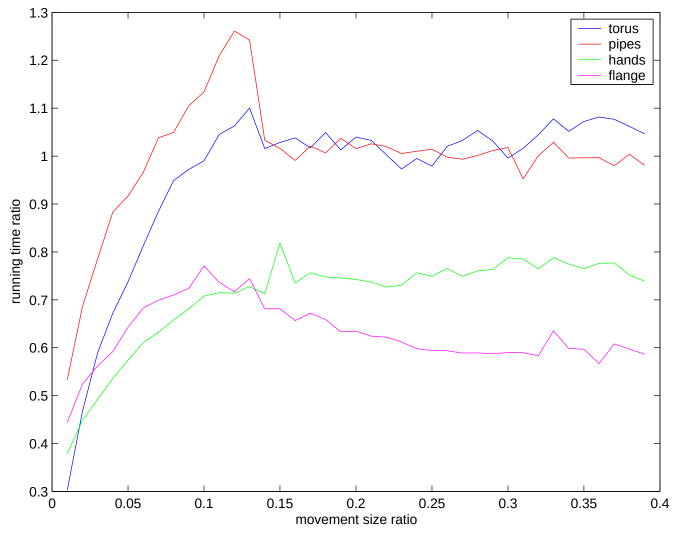


Fig. 8. Performance for different movement sizes using the OBB data structure.

Finally, Table 2 compares the coherence and non-coherence algorithms on the original benchmark scenarios and for interpolated scenarios for motion size of 0.03. Some of the original scenarios contain very large movements between frames (often much larger the the

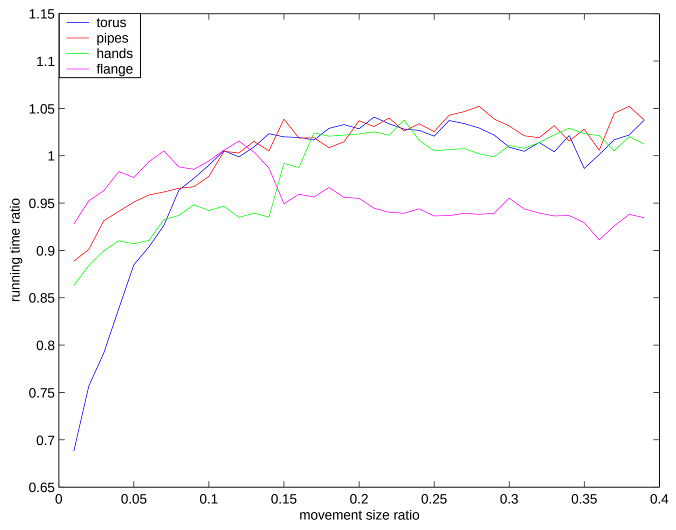


Fig. 9. Performance for different movement sizes using the AABB data structure.

object itself) and are not ideal for our algorithm. Even so, the results clearly demonstrate that the coherence algorithm is always faster. The advantage of the coherence approach on OBB varies from a factor of 2 for the Torus and the Flange and 1.1 for the Pipes model (which includes many very large movements). For the interpolated scenarios, the results are naturally better, yielding speedups factors of 1.27–2 for OBB and speedups factors of 1.04–1.26 for AABB. The difference between OBB and AABB is caused by the fact that AABB bounds the underlying models poorly, resulting in many triangle-triangle tests, where coherence cannot help.

Algorithm	Original				Interpolated 0.03			
	Torus	Pipes	Hand	Flange	Torus	Pipes	Hand	Flange
OBB non-coherence	1550	1150	1300	3355	1143	4389	6110	6894
OBB coherence	748	1050	951	1783	674	3449	3006	3878
AABB non-coherence	2834	850	1950	2684	1925	4897	7808	7123
AABBlite non-coherence	2200	682	1512	2003	1545	3735	6263	5486
AABB coherence	1562	672	1381	1812	1231	3480	5634	5283

Table 2. Comparison between the coherence and non-coherence algorithms for several scenarios

In conclusion, while temporal coherence is highly dependent on the size of the movement between frames, it can still be used on any scenario and increase performance. This is because the loss of performance due to a large movement is usually marginal, while the gain when the movement is small is substantial.

7. Conclusion

This paper has presented a general framework for exploiting temporal coherence in bounding volumes hierarchies for collision detection, using the *front* data structure. Within this framework, a novel coherence intersection test is introduced. This test speeds up the computation by using data, the margin and the separating axis, saved from the previous frame in the front.

The paper further explores the front data structure, proposing new front updating policies, which depend on the motion performed between frames.

In addition, the paper presents a statistical analysis of temporal coherence for bounding volumes and its relation to the size of the objects and the motion between frames. Based on these statistical observations, it is proposed when to use coherence and when to revert to the non-coherence tests, thus resulting in an algorithm that always performs at least as well as the non-coherence equivalent.

To demonstrate the benefit of the general framework, the algorithm was implemented for two bounding volume data structures, the AABB and the OBB. It is shown that the speedup is dominated by the velocity of the object within the scene. The slower the velocity with respect to the size of the object, the higher the speedup.

It is also shown that the speedup for the OBB is more significant than that achieved for the AABB. This is because the savings are gained by avoiding to perform intersection tests, which are more costly for OBB. Moreover, since OBBs have tighter bounding volumes, disjointness is discovered higher in the hierarchy, where coherence lasts longer, and less triangle-triangle tests are performed.

It is important to point out that the paper does not attempt to choose the best bounding volume but rather, to demonstrate the advantages of temporal coherence.

In the future, it is possible to test the general scheme for other bounding volume schemes, such as k-DOPs. Other applications, such as path planning, can also benefit from utilizing this technique.

Acknowledgements

This work was partially supported by European FP6 NoE grant 506766 (AIM@SHAPE), by the Israeli Ministry of Science, grant 01-01-01509 and by the S. Langberg Research Fund. The pipes and the torus models are courtesy of the Gamma group at UNC; the hand model is courtesy of the Boeing corporation; the flange model is courtesy of the Parasol Laboratory, Texas A&M university.

References

1. M. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In *Proc. of IMA Conference on Mathematics of Surfaces*, pages 37–56, 1998.
2. P Jimenez, F Thomas, and C Torras. 3D collision detection: a survey. *Computers and Graphics*, 25:269–285, 2001.
3. D.P. Dobkin and D.G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoret. Comput. Sci.*, 27:241–253, 1983.

4. J. Cohen, M. Lin, D. Manocha, and K. Ponamgi. I-COLLIDE: an interactive and exact collision detection system for large-scaled environments. In *Proc. ACM Int. 3D Graphics Conf.*, pages 189–196, 1995.
5. M. Held, J.T. Klosowski, and J.S.B. Mitchell. Evaluation of collision detection methods for virtual reality fly-throughs. *the 7th Canad. Conf. Computat. Geometry*, 14:36–43(2):205–210, 1995.
6. S. Ar, G. Montag, and A. Tal. Deferred, self-organizing BSP trees. In *Eurographics*, pages 269–278, 2002.
7. S. Ar, B. Chazelle, and A. Tal. Self-customized BSP trees for collision detection. *Computational Geometry: Theory and Applications*, 15(1-3):91–102, 2000.
8. P.M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Trans. on Graph.*, 15(3):179–210, 1996.
9. I.J. Palmer and R.L. Grimsdale. Collision detection for animation using sphere trees. *ACM Trans. on Graph.*, 14(4):105–116, 1995.
10. Y.C. Chen. An introduction to hierarchical probe model. Technical report, Dept. of Mathematical Sciences, Purdue Univ, 1985.
11. H. Samet. *Spatial Data Structures: Quadtrees, Octrees, and Other Hierarchical Methods.* Addison-Wesley, Redding, Mass., 1989.
12. J. Ponce and O. Faugeras. An object centered hierarchical representation for 3d objects: The prism tree. *Computer Vision, Graphics, and Image Processing*, 38:1–28, 1987.
13. 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.
14. Gino van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *J. Graph. Tools*, 2(4):1–13, 1997.
15. S. Gottschalk, M.C. Lin, and D. Manocha. OBBTree: a hierarchical structure for rapid interference detection. In . *ACM SIGGRAPH*, pages 171–180, 1996.
16. G. Barequet, B. Chazelle, L.J. Guibas, J. Mitchell, and A. Tal. BOXTREE: a hierarchical representation for surfaces in 3D. In *Proc. Eurographics*, pages 387–396, 1996.
17. M. Lin and J. Canny. A fast algorithm for incremental distance calculation. In *IEEE Int. Conf. on Robotics and Automation*, pages 1008–1014, 1991.
18. M.A. Otaduy and M.C. Lin. CLODs: dual hierarchies for multiresolution collision detection. In *Proc. of Eurographics Symposium on Geometry Processing*, pages 94–101, 2003.
19. S.A. Ehmann and M.C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum*, 20(3):500–511, 2001.
20. T.-Y. Li and J.-S. Chen. Incremental 3D collision detection with hierarchical data structures. In *VRST '98*, pages 139–144, November 1998.
21. http://www.cs.unc.edu/∼geom/obb/obbt.html.