Temporal Coherence in Bounding Volume Hierarchies for Collision Detection

Oren Tropp[1] Ayellet Tal[1] Ilan Shimshoni[1] David P. Dobkin[2]
[1]Technion [2]Princeton University
E-mail: [1]{ayellet@ee,ilans@ie}.technion.ac.il [2]dpd@cs.princeton.edu

Abstract

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 [10, 6]. Of the many algorithms and data structures used in collision detection [3, 2, 5, 1], we focus on the *Bounding volume (BV) hierarchy* representation. We explore this representation in a dynamic environment.

Our basic premise is that in dynamic environments, objects comprising the scene do not move much between consecutive frames [9, 11, 4, 7, 8]. *Temporal coherence* implies that collision queries and their results are typically dependent on queries made in previous frames.

Temporal coherence for collision detection was introduced in [9] for Voronoi diagram based data structures. In [7], a temporal coherence data structure for BV hierarchies, the *front*, was propose for k-DOPS. This data structure was further investigated in [8] for the spherical bounding volume data structure and later in [4] 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 [4, 8], which represents the collision tests performed thus far and the *front* data structure [4, 7, 8], which maintains the set of intersection tests that determine the collision between the objects.

There are various possible ways to update the front [8, 4]. The question remains whether certain strategies are better than others. Rather than developing a strategy that depends on frames, our strategy depends on a bound on the motion of the objects. The full 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.

An additional contribution of this paper is devising front updating policies empirically. 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.

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 a smaller motion again. Thus, the algorithm always performs at least as well as the non-coherence algorithm.

The coherence scheme is implemented for the OBB and AABB data structures. 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.

2. Algorithm and Data Structure

The intersection tests that should be performed for determining a collision between Bounding Volume (BV) hierarchies, can be described as a binary tree, the *Bounding Volume*

Test Tree (BVTT) [11]. 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.

Two operations on the front are supported: *sprouting* (replacing a node by its descendants) and *pruning* (replacing a set of nodes by their common ancestor) . Instead of developing a strategy that depends on frames [8, 4] , 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. See the full paper for our definitions of sprouting and pruning.

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.

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. This is done by a novel box-box intersection test, which is based on coherence. For each node in the front data structure, information from previous frames is maintained and used in the current frame. This information includes the *margin* between the boxes. If the motion is smaller than the margin, the boxes are guaranteed not to intersect. Updating the margin is a major concern of the algorithm. When the margin test fails, we show how to efficiently perform an intersection test between the boxes for OBB and AABB based on previous intersection test results. The coherence hypothesis is that the same separating axis found in the previous frame, is also separating in the current frame.

The general coherence algorithm is described in Algorithm 1. We refer the reader to the full paper.

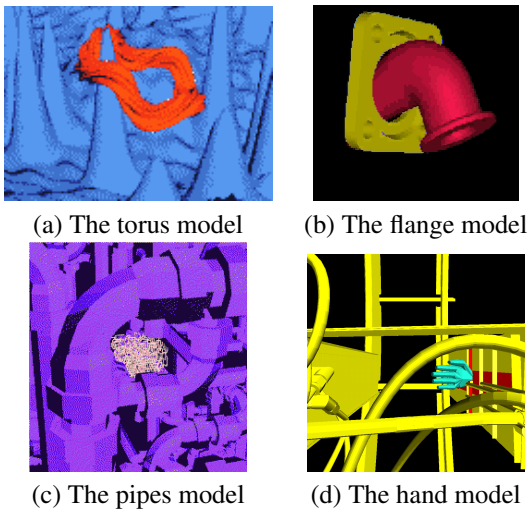
Algorithm 1 Algorithm overview (for each frame)

```
if the motion is large then
    Call the no-coherence algorithm
    Sprout nodes in the front and update coherence data
    Exit
end if
for each node in the front do
    if the node was disjoint then
        Check whether it is still disjoint using coherence
    else if the node was intersecting then
        Transform the coordinate systems
        Perform full non-coherence BV-BV test for the node
    end if
    if the node is disjoint in the new frame then
        Keep it in the front
    else if the node is intersecting in the new frame then
        Expand the node with the non-coherence algorithm
        Update the front according to the sprouting policy
    end if
end for
if the movement since last pruning is substantial then
    Attempt to prune the tree
end if
```

3. 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 [12]. For AABB, the *SAT lite* algorithm is used with the necessary code modifications [13]. The tests are executed on a Pentium 4 1.8Ghz processor. 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 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 1). For each benchmark scenario, similar paths, each having a different motion size, are produced by interpolating the original path. Each test is performed 70 times and the results are averaged.



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

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

Figure 1. Benchmarks

Figure 2 compared our OBB algorithm to the original algorithm for some typical movements. As expected, the

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

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. Similar results are obtained for AABB.

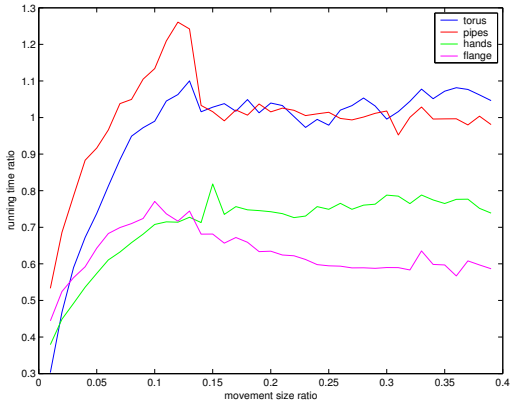


Figure 2. Performance for different movement sizes using the OBB data structure.

Finally, Table 1 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 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.

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.

Acknowledgments: This work was partially supported by European FP6 NoE grant 506766 (AIM@SHAPE), by the Israeli Ministry of Science, grant 01-01-01509. The models are courtesy of UNC, Texas A&M university and the Boeing corporation.

References

[1] S. Ar, G. Montag, and A. Tal. Deferred, self-organizing BSP trees. In *Eurographics*, pages 269–278, 2002.

[2] 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.

[3] D. Dobkin and D. Kirkpatrick. Fast detection of polyhedral intersection. *Theoret. Comput. Sci.*, 27:241–253, 1983.

[4] S. Ehmann and M. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum*, 20(3):500–511, 2001.

[5] M. Held, J. Klosowski, and J. 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] P. Jimenez, F. Thomas, and C. Torras. 3D collision detection: a survey. *Computers and Graphics*, 25:269–285, 2001.

[7] J. Klosowski, M. Held, J. 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.

[8] T.-Y. Li and J.-S. Chen. Incremental 3D collision detection with hierarchical data structures. In *VRST '98*, pages 139–144, Nov. 1998.

[9] M. Lin and J. Canny. A fast algorithm for incremental distance calculation. In *IEEE Int. Conf. on Robotics and Automation*, pages 1008–1014, 1991.

[10] 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.

[11] M. Otaduy and M. Lin. CLODs: dual hierarchies for multiresolution collision detection. In *Proc. of Eurographics Symposium on Geometry Processing*, pages 94–101, 2003.

[12] http://www.cs.unc.edu/∼geom/obb/obbt.html.

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