Memory-optimized bounding-volume hierarchies

Pierre Terdiman

March 2001

Abstract

Memory-optimized bounding-volume hierarchies are presented, in the context of collision detection. An efficient implementation is presented as well, using only 20 bytes per triangle for a complete binary tree. We show why this approach ultimately yields faster collision detection queries than the standard one.

1) Introduction

Bounding-volume hierarchies are the most common collision detection (CD) structures when dealing with non-convex meshes or polygon soups [1]. They have been used in numerous freely-available collision detection packages (RAPID [2], SOLID [4], QuickCD [3], PQP [10], etc) to investigate the performance of various bounding volumes (BV). While using BV-trees is a very efficient method when it comes to detecting collisions between arbitrary geometries, it still has one major drawback : memory requirements.

A lot of papers have focused on the efficiency of collision queries : various ways of building the trees have been explored [4], experiments have been made with hybrid hierarchies [14], and almost the whole BV-spectrum (1) has been implemented in at least one available CD library.

However, memory requirements have not received that much attention. BV-trees often require a non-negligeable amount of bytes per triangle, which can ultimately lead to major problems [8].

In this paper, we present two easy ways to dramatically reduce the memory footprint of BV-trees.

2) Problem statement

A complete BV-tree is made of 2*N-1 nodes, where N is the number of primitives (usually triangles) in the input model. In complete trees each leaf node contains a single primitive, which means there are N leaf nodes in them, and N-1 internal nodes. [4]

(1) Sphere, AABB, OBB, k-DOP, convex hull

The size of a single node mainly depends on the bounding volume it contains. Table 1 gives a summary of the number of bytes required by standard BVs, assuming 32-bits floats (2). Table 2 gives the number of bytes consumed by the bounding volumes for various numbers of N. This does not take into account the extra pointers needed to walk the hierarchy or the actual triangle data (supposed to take a fixed amount of bytes regardless of the BV in use).

Bounding volume	**Size in bytes**	**Used in**
Sphere, a.k.a. PSS	16	PQP
AABB	24	SOLID
LSS	32	PQP
OBB (using quaternions)	40	Personal library
RSS	48	PQP
OBB (using matrices)	60	RAPID
k-DOP (using floats)	4*k	Quick-CD

Table 1 : size of common bounding volumes

Bounding volume	**1000 triangles**	**10000 triangles**	**100000 triangles**
Sphere, a.k.a. PSS	31 Ko	312 Ko	3.05 Mo
AABB	46 Ko	468 Ko	4.57 Mo
LSS	62 Ko	624 Ko	6.10 Mo
OBB (using quaternions)	78 Ko	781 Ko	7.62 Mo
RSS	93 Ko	937 Ko	9.15 Mo
OBB (using matrices)	117 Ko	1.14 Mo	11.44 Mo
18-DOP (3)	140 Ko	1.37 Mo	13.73 Mo

Table 2 : size of BVs in a BV-tree for various combinations of BV / model

The problem appears clearly : BV-trees quickly require a huge amount of bytes. In particular cases such as games running on consoles with limited memory, this can be a major problem. Next-generation machines such as the X-Box are expected to deal with meshes of 100K triangles without difficulties. But the available memory remains limited, which makes the usage of plain BV-trees quite delicate. Even for rapid prototyping, the increased amount of available memory in the system often comes with an increased number of triangles in the models (up to several millions), which leads to the same conclusion : memory is the bottleneck. [8].

(2) Multiply all values by 2 for double precision. In all our tests, no significant differences have been found between single and double precision, so we just use floats.
(3) 18-DOPs are the most efficient k-DOPs according to [3]

3) Reducing memory requirements

3-1) Cost equations

The cost equation for collision detection is, in its basic form [2] :

$$T = Nv * Cv + Np * Cp$$

with

T = total cost function for interference detection
Nv = number of bounding volume pair overlap tests
Cv = cost of testing a pair of BVs for overlap
Np = number of primitive pairs tested for interference
Cp = cost of testing a pair of primitives for interference

Let's introduce a similar cost equation for memory usage :

$$M = Nbi * Sbi + Nbl * Sbl$$

with

M = total memory requirements in bytes
Nbi = number of internal nodes
Sbi = size of a single internal node in bytes
Nbl = number of leaf nodes
Sbl = size of a single leaf node in bytes

3-2) Early optimizations

There are two standard ways to reduce M.

The first way is to have more than one primitive in each leaf node. The resulting BV-tree is not as deep as a complete one, hence wasting less bytes since Nbi as well as Nbl decrease. However, this means testing two leaf nodes L1 and L2 for overlap yields N1*N2 primitive-primitive tests, where N1 (resp. N2) is the number of primitives contained in L1 (resp. L2). In other words, M is reduced but T is increased. Despite this, many game developers have taken this route [9], since they just couldn't afford the memory footprint coming with complete BV-trees.

The second way is to reduce Nbi and Nbl. Let's define indeed a canonical internal node, containing :

- a bounding volume
- a pointer/index to the positive/left child
- a pointer/index to the negative/right child

Let's also define a canonical leaf node, containing :

- a bounding volume
- a pointer/index to the surrounded primitive

A simple observation can be made in order to save some bytes : we can organize nodes as we like in memory, hence for example ensuring positive and negative children are always stored in contiguous locations. So we can get rid of the negative pointer/index, by making it implicit and always ensuring negative index = positive index + 1 (or vice versa).

It is worth noting standard collision detection packages are usually *not* memory-friendly to begin with. RAPID for example keeps three pointers in each node, even if one (resp. two) of them is (resp. are) useless in case of internal (resp. leaf) nodes. One can instead efficiently pack all three pointers in a single one, so that any BV-node only needs a bounding volume and a single pointer, regardless of its nature. This approach has been used in most recent packages such as PQP, but not in older ones (RAPID, V-COLLIDE, SOLID…)

In some cases, the size of the bounding volume itself can be reduced. For example rotation matrices in RAPID can be replaced with a unit quaternion, as reported in the original OBB tree paper. [2] Nonetheless, only minor gains can be expected with those approachs, and we need something else to dramatically reduce memory requirements.

3-3) Aggressive optimizations

We present here two easy ways to greatly optimize the size of BV-trees. Both of them, individually, allow one to reduce memory requirements by about 50%. Combining them provides an immediate memory gain of about 75%.

The first optimization reduces the number of nodes in the tree, while the second one reduces the size of each node.

3-3-1) Wiping the leaf nodes out

In a complete tree, a leaf node contains a single surrounded primitive (say a unique triangle), as well as the primitive’s bounding volume. During the recursive collision queries, when it comes to checking two leaf nodes for overlap, the pair of bounding volumes is used first and a BV-BV test is performed. Then, only in case the bounding volumes collide, the underlying primitives are tested for exact intersection. Our basic idea is simply to skip the first bounding volume test, and directly perform the primitive-primitive test instead. This approach was motivated by the following observation using OBB trees : state-of-the-art triangle-triangle overlap test [5] and OBB-OBB intersection tests [2][11] are about as fast (Table 3). So, skiping the OBB-OBB test for leaf nodes should be efficient because :

- If the triangles collide, we have to perform the triangle overlap test anyway. In such a case, performing the OBB-OBB test prior to the triangle-triangle one is just a waste of time.
- If they don’t we detect it as well, roughly as fast as before.

Overlap test	**Intersection occurs (w/wo cache misses)**	**Quickest early exit (w/wo cache misses)**
OBB-OBB	~3500 / 520 cycles	~1500 / 174 cycles
Triangle-Triangle	~2900 / 350 cycles	~1500 / 123 cycles
Triangle-AABB	~4000 / 560 cycles	~900 / 44 cycles

Table 3 : speed of overlap tests, mesured on a Celeron 500 Mhz (VC++ 6.0, Release mode)

Now, does this mean the bounding volume is not needed anymore in leaf nodes ? No, because we still may have to collide a leaf node against an internal one. That is, if we want to get rid of bounding volumes in leaf nodes, we also need a primitive-BV intersection test. Assuming we have this routine, there's only a primitive-pointer remaining in leaf nodes. We can get rid of them as well, by storing them in parent nodes, replacing previous pointers to the leaf nodes we just discarded.

On one hand this has one drawback : we can't mix this with a scheme where the negative box pointer is made implicit. Indeed, both children can now be internal nodes or primitive pointers, whereas children of a standard BV-node share a common nature. Hence we really need to keep track of two pointers in each node.

On the other hand, we can now totally wipe the leaf nodes out. This is a huge gain as far as memory is concerned. Recall a complete tree contains N leaf nodes out of 2*N-1. We just dropped half of them.

Moreover, replacing a coarse BV-BV test with a more accurate primitive-BV test actually leads to less tests than the original design. For example, a BV-BV test can report overlap whereas the new primitive-BV test does not – and the recursive query stops earlier.

As a bonus, the tree construction code can be easier. In RAPID for example, a dedicated routine computes the OBB for leaf nodes. If we discard them, we don't need this dedicated code anymore.

Finally, the required triangle-box test has been derived [6], and turns out to be roughly as fast as the standard OBB-OBB test (Table 3). In other words this new approach needs half less bytes as before and ends up running faster overall.

The new layout for collision queries is, in pseudo-code :

```
A and B are two bounding volumes
A0 and A1 are both children of A
B0 and B1 are both children of B

test(A, B)
{
        if(A overlaps B)
        {
                // A0B0
                if(A0 is leaf)
                        if(B0 is leaf)      test(leaf, leaf)    => Primitive-Primitive test
                        else                test(leaf, box)     => Primitive-BV test
                else
                        if(B0 is leaf)      test(box, leaf)     => Primitive-BV test
                        else                test(box, box)      => BV-BV test

                // Repeat for A0B1
                        …
                // Repeat for A1B0
                        …
                // Repeat for A1B1
                        …
         }
}
```

Code complexity is clearly greater than what it is in the counterpart collision queries of usual CD libraries, but we believe it’s worth the effort.

Finally the cost equation becomes :

$$T = Nv * Cv + Np * Cp + Npv * Cpv$$

with

T = total cost function for interference detection
Nv = number of bounding volume pair overlap tests
Cv = cost of testing a pair of BVs for overlap
Np = number of primitive pairs tested for interference
Cp = cost of testing a pair of primitives for interference
Npv = number of primitive-BV pair overlap tests
Cpv = cost of testing a primitive against a BV for overlap

3-3-2) Quantized trees

Our second approach was to use a fast and easy compression scheme, to reduce the size of each bounding volume. We chose a simple quantization method, not to introduce a huge speed penalty in the system. Each floating-point value is quantized to a 16-bits integer, and dequantized at runtime, during the recursive collision queries. The process is simple (e.g. 6 int-to-float conversions and 6 fmuls for an AABB) and similar to many geometry compression strategies. [12]

Using naively-quantized bounding volumes for collision detection can produce incorrect results, since the dequantized volume can end up a bit smaller than the original one. In other words, the quantization is not conservative.

Hence, we use an extra piece of code while building the quantized trees, to fix problematic nodes. This guarantees the dequantized volumes are as large or larger than the original ones. Of course, using slightly larger volumes leads to more tests during collision queries, and there is an obvious price to pay here – the usual tradeoff between speed and memory.

Nevertheless, mixing this approach and the previous one – where we removed leaf nodes - is far less costly. Indeed, by discarding leaf nodes we have exchanged a lot of BV-BV tests for primitive-BV or primitive-primitive tests. The number of bounding volumes to dequantize is then reduced (saving the dequantization time), and the impact of quantization is not as big as with a standard tree (not as many BVs are involved in overlap tests). Moreover we can further help this behaviour and reduce the influence of bounding-volumes by using coarser BV-BV tests, which means recursive collision queries quickly start using primitive-BV or primitive-primitive tests. Those coarser tests have already been used in [4] (where they are named *SAT-lite* tests) and in [7] (where the discarded separating axes are labelled as *Class III axes*).

On top of that, mixing the two strategies saves a lot of memory, which leads to far less cache misses. Fetching data from memory is one of the most costly steps in a processing pipeline [7], and we believe our new mixed BV-tree is a lot more efficient than standard ones regarding this issue.

4) Implementation and results

We have implemented the aforementioned methods on PC, in a collision detection library named *OPCODE* (OPtimized COllision DEtection). Our implementation uses AABBs as the default bounding volume. Since our primary goal was to reduce the memory requirements, it has been selected as the best compromise in the whole BV-spectrum.

Our default AABB-tree implementation is efficient and comparable to previously published versions [4] – yet using less bytes, with only one pointer per node. SAT-lite tests using only 6 separating axes can be dynamically selected instead of the canonical tests (15 separating axes). The triangle-triangle test used [5] is also faster than the one used in RAPID [2][4].

We then modified the code in order to discard all leaf nodes, using a new efficient triangle-box test [6]. The original code by Möller has further partially been rewritten in order to take advantage of particular assembly instructions – namely FCOMI and FCMOV. We also successfully applied the SAT-lite strategy to the triangle-box test as well.

Finally, we quantized the remaining bounding-volumes to end up with a BV-node as small as 20 bytes (12 bytes for a quantized AABB, 8 bytes for two pointers). Not only this is smaller than even a single standard AABB, but also we need half the nodes of a standard tree. A comparison of our implementation against common libraries is summarized in Table 4. It is worth noting it could further be reduced to 16 bytes per triangle with a bit of extra work, by replacing pointers with delta-encoded 16 bits indices. [14]

Collision detection library	**Bytes per triangle and comments**
Quick-CD – using 18-DOPS	168, assuming floats instead of doubles
RAPID 2.01	144, assuming floats instead of doubles
SOLID 2.0	62, counting 1 leaf and 1 internal node/tri
OPCODE (non-compressed tree)	32, size of a cache line
OPCODE (quantized tree)	20

Table 4 : memory requirements of various CD packages

Table 5 shows the number of overlap tests performed in four scenes where objects overlap. As we expected, the total number of tests (BV-BV + Prim-Prim + BV-Prim) is smaller for trees whose leaves have been removed. This is due to the BV-Prim test beeing more accurate than the BV-BV one, and hence stoping the recursive descent earlier.

Models	**BV-BV tests**	**Prim-Prim tests**	**BV-Prim tests**	**Total**
Knot (normal)	19375	3212	0	22587 (100%)
Knot (no leaf)	8483	5003	5445	18931 (83%)
Venus (normal)	36709	5074	0	41783 (100%)
Venus (no leaf)	16988	7531	7796	32315 (77%)
Chevy (normal)	62101	10104	0	72205 (100%)
Chevy (no leaf)	24779	13272	29760	67811 (93%)
Teapot (normal)	18405	2367	0	20772 (100%)
Teapot (no leaf)	8705	3964	4454	17123 (82%)

Table 5 : total number of intersection tests with or without leaf nodes

As far as speed is concerned, it is a lot more difficult to provide accurate comparisons and fair figures [13]. Often, simply changing the relative orientation of two models makes one library faster or slower than another. In many cases we have found our implementation to outperform RAPID (up to 5 times faster), mainly when objects were deeply overlapping. This is very encouraging since intersection testing using AABB trees usually takes 50% *longer* than using OBB trees in cases where there is a lot of overlap among the models [4]. In some other cases RAPID was faster, mainly in close-proximity scenarii where OBBs are really more appropriate than AABBs (and indeed RAPID sometimes uses a single OBB-OBB test in those situations, whereas we need a lot more). Nonetheless it's interesting to note our version was often not significantly slower than RAPID in those cases. The library and more information about its performance against RAPID can be found here : www.codercorner.com/Opcode.htm

Table 6 shows the influence of quantization and triangle-box tests, for various models of various complexity, compared to our standard version. A value of 1.0 represents the speed of the original code. A value such as 2.0 means the collision query takes 2 times longer than the normal code. So values lesser than 1.0 are actually speedups.

Models	**No leaves (i.e. with triangle-box tests)**	**Quantized**	**No leaves + quantized**	**No leaves + quantized + SAT lite**
Knot / Knot	1.09	1.36	1.21	1.06
Venus / Room	0.97	2.07	1.22	1.00
Torus / Torus	1.10	1.86	1.23	1.07
Teapot / Teapot	0.89	1.34	1.01	0.83
Sphere/ Cylinder	0.88	1.52	0.97	0.86
Knot / Knot 2	0.75	1.31	0.82	0.68
Venus / Venus	0.80	1.36	0.92	0.76
Venus / Torus	0.98	2.89	1.19	0.99
Explora	1.00	2.73	1.21	0.99
Gears	1.18	1.41	1.35	1.06
Shad	0.95	1.20	1.04	0.79
Shad 2	0.86	1.21	0.93	0.73
Chevy / Chevy	0.82	1.08	0.93	0.77
Char / Char	0.88	1.37	1.09	1.05

Table 6 : influence of memory-optimizations on speed

On one hand, it confirms discarding leaf nodes and introducing triangle-box overlap tests usually yields faster queries. On the other hand using quantized volumes always results in a slowdown. Nonetheless, quantizing no-leaf trees is not as costly - as we expected. Sometimes it even still yields faster queries, despite memory requirements have been divided by 4.

Finally, the last column gives results for quantized no-leaf trees using SAT-lite tests in both BV-BV and primitive-BV overlap tests. Most values are below 1.0 or only slightly greater, which means the new tree uses less ram and on top of that provides faster queries most of the time.

5) Conclusion and acknowledgement

We presented two ways to greatly reduce the memory requirements of bounding volume hierarchies : using primitive-BV tests to discard leaf nodes, and using quantized bounding volumes. Mixing the two methods divides memory requirements by 4, and also usually yields faster collision queries.

In our implementation, named OPCODE, a complete tree is 7.2 times smaller than a tree in RAPID and yet collision queries often run as fast or faster.

Thanks to Tomas Möller for the fast triangle-box code.

[1] Tomas Möller, Eric Haines, Real-Time Rendering (p.232) ISBN 1-56881-101-2

[2] Gottschalk, S., M.C. Lin, and D. Manocha, OBBTree : A Hierarchical Structure for Rapid Interference Detection, *Computer Graphics (SIGGRAPH'96 Proceedings)*, pp. 171-180, August, 1996.

[3] Klosowski, James Thomas, Efficient Collision Detection for Interactive 3D Graphics and Virtual Environments, PhD dissertation, May 1998

[4] van den Bergen, G. Efficient Collision Detection of Complex Deformable Models Using AABB Trees, *Journal of Graphics Tools*, vol. 2, no. 4, 1997

[5] Möller, Tomas, A Fast Triangle-Triangle Intersection Test, *Journal of Graphics Tools*, vol. 2, no. 2, pp. 25-30, 1997

[6] Möller, Tomas, Fast Triangle-Box Overlap Testing, to appear.

[7] Intel, AP-812, Applying Streaming SIMD Extensions to Collision Detection

[8] Larsen, Eric, Static N-Body Collision Detection for Massive Models, http://www.cs.unc.edu/~larsen/290/project.html

[9] GD-Algorithms list archives

[10] Larsen, E., Gottschalk S., M.C.Lin, Manocha D., Fast Proximity Queries with Swept Sphere Volumes, technical report TR99-018, Department of Computer Science, UNC Chapel Hill

[11] Gomez M., Simple Intersection Tests for Games, October 18, 1999 http://www.gamasutra.com/features/19991018/Gomez_1.htm

[12] Deering, Michael, Geometry Compression, *Computer Graphics (SIGGRAPH'95 Proceedings)*, pp. 13-20, August 1995.

[13] Zachmann, Gabriel, Benchmarking Collision Detection Algorithms, http://www.igd.fhg.de/~zach/coldet/

[14] Gottschalk, Stefan, Personal communication, 2000