Bounding box collision detection

Tomasz Koziara, Nenad Bićanić

Department of Civil Engineering
University of Glasgow, Glasgow G12 8LT, Scotland, UK
t.koziara@civil.gla.ac.uk

keywords: bounding box intersection, collision detection, spatial hashing, segment tree, sweep-plane, priority search tree

1. Introduction

In this paper we address the problem of intersection detection between pairs of bounding boxes in three-dimensional space. Our motivation for investigating this problem comes from the need for an efficient collision detection framework applicable in the context of large scale contact analysis. Not considering underlying physics of the mechanical contact problem we focus on purely geometrical setting. Moreover we simplify the geometrical entities further by assuming that the objects we are dealing with are axis aligned bounding boxes undergoing arbitrary combination of a simultaneous rigid move, shrinking and expanding along specific directions. This simplification, as it could seem restrictive, allows us not to relay on object representation details and at the same time gain an insight into combinatorial nature of the problem. This in turn helps to realise the place of the collision detection pipeline among other computational tasks in contact mechanics. Assuming we are dealing with n boxes, with k intersecting pairs among them, we try to address the question how close can we get to an ideal algorithm having $O(n+k)$ running time and $O(n)$ space demands.

2. Related work

Collision detection problem emerges in a wide range of applications: physically based simulations, animation, virtual reality, robotics - to mention only few of them. Here we focus only on the *box intersection* problem, which is usually a sub-problem in a more general contact detection setting. Still we can try to address it using one of few basic paradigms. We can namely use *space partitioning* approach ([3, 2]), *volume hierarchy* approach ([5, 6]), or *spatial reduction* technique ([1, 7]). *Space partitioning* methods subdivide the space into a union of simple volumes and then filter objects by immersing them in a predefined spatial structure. *Volume hierarchy* methods build tree-like structures on hierarchies of bounding volumes and enable efficient intersection test on these structures. *Space reduction* methods address the problem by solving and combining simpler, lower-dimensional sub-problems. These methods are especially well suited to work with *axis aligned bounding boxes* (AABB).
An important theoretical result obtained recently by Suri et al. [4], reassures usefulness of the bounding volume heuristics. It states that as long as we are using convex volumes (e.g. AABB) and meet some reasonable assumptions about their relation to the shape of objects they bound, we can expect that the number of intersections reported will remain proportional to the number of actual object intersections.

3. Algorithms

Our objective is to search for an optimal algorithm for three-dimensional box intersection problem. By optimal we mean an algorithm with $O(n \log n + k)$ running time bound and $O(n)$ space requirements. Such a hypothetical algorithm should ideally also take advantage of the *temporal coherence* and in that case reduce its expected running time to $O(n+k)$ steps. To our knowledge, no such algorithm exists. In the following subsections we will present briefly basic techniques that can be applied in our setting. None of them is optimal in a sense

defined above. However they are a good starting point for further investigations. In the sequel we compare both complexity, and implementation aspects of considered approaches, as this is where constant factors in the O-notation hide.

3.1. **Multi-level Segment Trees.** One of tested algorithms is HYBRID, presented in [1]. It exploits a data structure called *segment tree* (ST), conventionally used to solve *stabbing query* problem in the one-dimensional space. The problem can be defined as follows: Having a set of n intervals denoted S and a query point p, report all intervals that contain the point. To apply this structure in our setting, firstly we notice that two boxes intersect if and only if their projections on coordinate axes overlap, and secondly that two intervals overlap if either of them contains a low endpoint of the other. ST structure is well suited for solving one dimensional interval overlap test (Figure 1 presents an example of ST, for precise description see [8]). To address three-dimensional problem we need to build a *multi-level* version of ST. This way we end up with an algorithm having $O(n \log^3 n + k)$ running time and, rather unfeasible, $O(n \log^3 n)$ space requirements. Hybridisation technique proposed in [1] decreases space usage to $O(n)$ with the idea of building *multi-level ST* recursively, rather than storing the whole structure in the memory. Another important speedup factor introduced in the mentioned work is using *cutoffs*, which basically means abandoning the idea of building the complete structure and applying simple one-dimensional *scanning* test when a number of intervals to be processed in a sub-tree drops down to an empirically selected value.

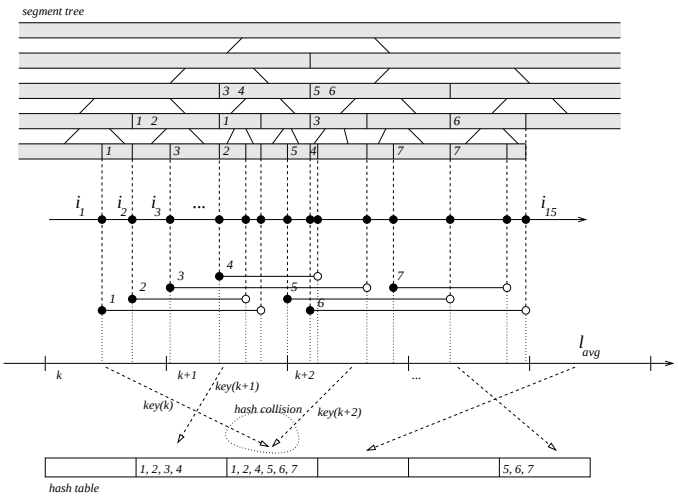

FIGURE 1. Segment tree and hash table structures built on one-dimensional set of intervals.

3.2. **Spatial Hashing.** The basic idea here is to subdivide a 3-space into a set of orthogonal *voxels*[1] V, and define an surjective mapping $f_{hash} : V \ni v \longrightarrow z \in \{1, 2, ..., h\}$ [2]. This way, with each voxel we associate a list L_z storing boxes that overlap it (Figure 1 presents one-dimensional analogue). We then process non empty lists L_z detecting exact box intersections. This methodology has several drawbacks. First we need to choose a good *hashing function* f_{hash} (the more injective, the better). Then we need to select a proper voxel size l_V. In the end we need to deal with repeated reports of pairs of overlapping boxes. It can be shown that if $l_V = O(l_{avg})$, then $\sum |L_z| = O(n)$, where l_{avg} is an average box size and n is the number of boxes. From that it results, that creating lists L_z takes $O(n)$ steps, and sorting them $O(n \log n)$. Sorting is necessary if we want to apply one-dimensional *scanning* for precise contact detection. The efficiency of this approach decreases with growing average length of L_z. Under the assumption of uniform and reasonably sparse packing of space with boxes it

[1]The word *voxel* comes from the world of image processing, and describes "the smallest distinguishable box-shaped part of a three-dimensional image" (http://www.webopedia.com). It will be used throughout this paper in a broader sense, namely: a *voxel* is the smallest distinguishable box-shaped subset of n-dimensional space.

will perform well, but its asymptotic behaviour is rather poor. The algorithm resulting from this approach will be further called HASH2.

3.3. **Sweep-plane approach.** Sweeping is one of the classical techniques employed in computational geometry. Using a *sweep-plane* we reduce our setting to *two-dimensional dynamic rectangle intersection* problem. Assuming that we have a dynamic structure for rectangle intersection problem DR, and a sweep-plane is orthogonal to z-direction, whenever a box is about to enter it, we *query* DR with its xy-rectangle r, and report all the overlaps with rectangles stored in DR. Then we *insert* r into DR. When a box is about to leave the sweep-plane, we *delete* its xy-rectangle from DR. As we can sort z-coordinates of box endpoints in $O(n \log n)$ steps, the actual efficiency depends on the complexity of *query*, *insertion* and *deletion* operations. Optimal structure should enable logarithmic update and query times, but as it has not been invented yet, we only try to imitate it. This way we obtain four versions of the sweep-plane algorithm: SWEEP1 (DR is imitated by two-dimensional hashing and single linked lists for storing rectangles), SWEEP2 (DR is imitated by two-dimensional hashing and a *priority search tree* (PST) along the x-direction), SWEEP3 (DR is imitated by PST along x-direction, for each reported x-intersection we need to check the y-one), SWEEP4 (DR is imitated by one-dimensional hashing along x-direction and PST along y-direction). *Priority search tree*, mentioned above is a classical structure used in computational geometry to solve the *dynamic interval intersection* problem [8].

3.4. **Temporal coherence.** We would like to exploit *temporal coherence*, i.e. the fact that adjacent time frames of dynamic simulation are likely to differ only "slightly". Namely we would like to take an advantage of the fact, that *insertion sort* allows us to reorder almost sorted data in $O(n)$ steps [7]. It is not possible to use this technique in the HYBRID algorithm, as the multi-level tree structure is built in top-down manner and stored only partially on the program stack (recursion). In the case of HASH2 algorithm there is also not much to do, as boxes can migrate from one list L_z to another, thereby destroying coherence. The good news is that we can easily speed up SWEEP algorithms. So assuming that our DR structure is optimal, we could reach $O(n \log m + k)$ running time, where $m \leq n$. Optimistically m could be a small number, if only a small number of boxes would intersect the sweep-plane at one time.

4. Experiments

4.1. **Testing sets.** Our testing suite comprises three different classes of sets. To test the asymptotic behaviour of the algorithms we use "over-packed" random distribution of boxes as well as over-packed spherical distribution. The term "over-packed" means here that the number of intersections is $O(n^2)$. To test the influence of *uniformity* in box distribution we use adjacent cubic sets. We also test the algorithms with spherical distribution of boxes resulting in $O(n)$ intersections. We do not test skew or elongated configurations, neither we test configurations with considerably varying box sizes or aspect ratios, as we are interested only in a general comparison.

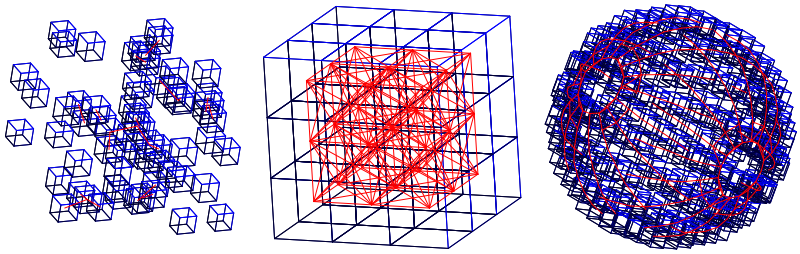

Figure 2. Examples of three classes of testing sets: random, adjacent, and spherical surface distributions of bounding boxes.

4.2. **Performance comparison.** All the test were run on IBM T42p laptop with 1.7GHz CPU and 1GB of RAM. The algorithms were implemented in ANSI C, and compiled without any optimisation. Some of the results could be probably improved by putting more care into coding.

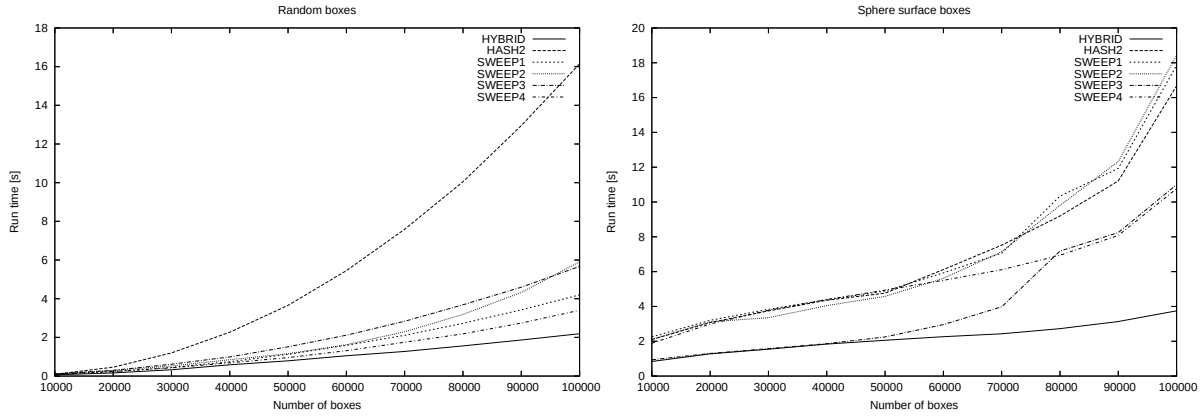

FIGURE 3. Random set of boxes and boxes packed on spherical surface with number of intersections $k = O(n^2)$.

For the random set, the space becomes much more "over-packed" then it does for the spherical distribution (Figure 3). This is why HASH2 algorithm performs that poorly. The best performing algorithm is HYBRID, while all the SWEEP algorithms perform on average the same. However SWEEP4 performs almost the same as HYBRID until the level of 50K of boxes on spherical surface. This is because the distribution of boxes on one half of the sphere is aligned in a way that intersection of intervals in x-direction is almost equivalent with intersection of boxes (so PST performs optimally), while this is not the case on the other side. Also SWEEP4 seems to perform reasonably well.

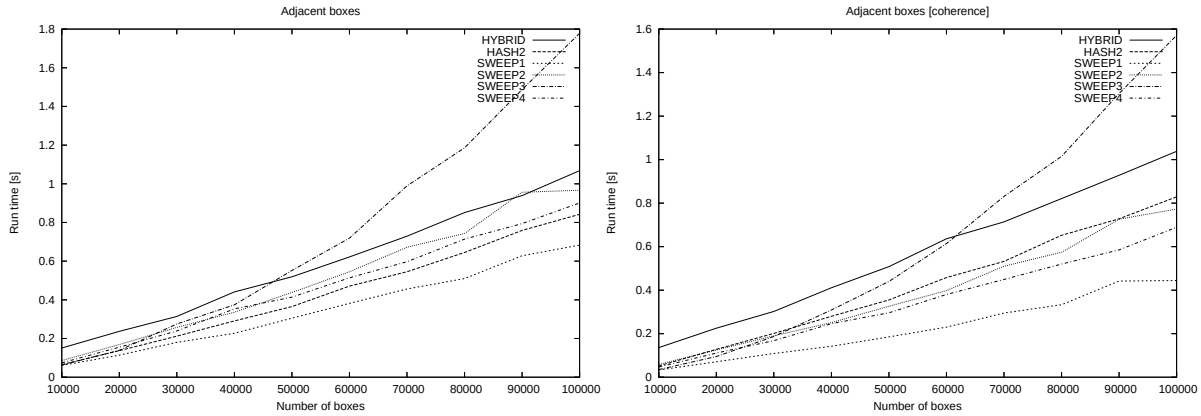

FIGURE 4. Sets of adjacent boxes with and without temporal coherence assumption. Number of reported intersections $k = O(n)$.

Figure 4 represents a typical setting where we would expect a good performance from the HASH2 algorithm. It is therefore no surprising that it outperforms most of the remaining algorithms. Only SWEEP1, due to the fact that it also uses hashing (in 2-space) and simple list for storing rectangles occupying the same voxel, outperforms HASH2. As the distribution of boxes in the the directions parallel to the sweep-plane is uniform, all the operations on lists L_z have average constant time complexity, while for all the rest of SWEEP algorithms this constant is bigger. It is worth noting that the assumption of time coherence reduces only about $\frac{1}{3}$ of the run time - the cost of sorting coordinates along the sweep direction. Again SWEEP4 places itself among the best performing algorithms. HYBRID and SWEEP3 are being outperform, as they do not benefit from the uniformity of input sets.

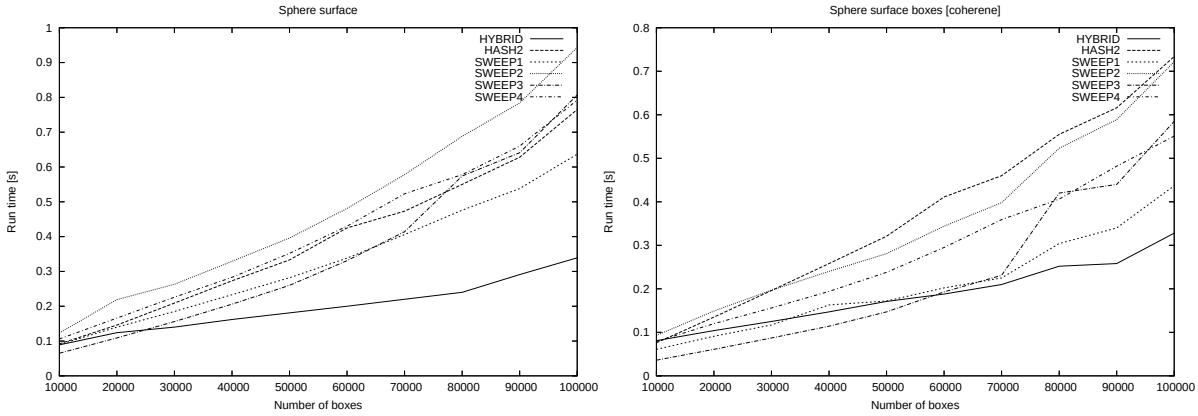

FIGURE 5. Spherical distributions of boxes with $k = O(n)$, with and without coherence assumption.

In Figure 5, again HYBRID is the best performer and even coherence does not allow the fastest SWEEP algorithm to outperform it. As the space is not densely packed SWEEP1 also performs well. On average, all the SWEEP algorithms perform similarly, which shows that all our attempts of imitating an optimal DR structure are more or less equally (un)successful.

5. Conclusions

We have reviewed a number of approaches for dealing with *axis aligned bounding box collision* problem in 3D. It seems that there is no efficient approach to date, that would exploit the advantage of *temporal coherence* in this setting. Paradoxically the best known to us static algorithm (HYBRID, [1]), usually outperforms all the remaining approaches, even when they are applied with the assumption of temporal coherence. This shows that there is still much to be done in the field of bounding box collision detection. The first apparent improvement would be to find an optimal dynamic algorithm for the rectangle intersection problem. That would result in the theoretically optimal algorithm for box intersection problem, but presumably, temporal coherence speedup would not be much greater than obtained here. This suggests, that one should rather look for an algorithm, not optimal in general, but performing better in a time-coherent, dynamic setting.

References

[1] A. Zomorodian, H. Edelsbrunner, "Fast software for box intersections", 2002, *International Journal of Computational Geometry and Applications*, Vol. 12, Numbers 1-2, pp. 143-172.

[2] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets, M. Gross, "Optimized Spatial Hashing for Collision Detection of Deformable Objects", 2003, *Proc. Vision, Modelling, Visualization VMV'03*, pp. 47-54.

[3] F. Ganovelli, J. Dingliana, C. O'Sulivan, "BucketTree: Improving Collision Detection Between Deformable Objects", 2000, *Proceedings of SCCG2000*, pp. 156-163.

[4] S. Suri, P. M. Hubbard, J. F. Hughes, "Analyzing Bounding Boxes for Object Intersection", 1999, *ACM Trans. Graph 18(3)*, pp. 257-277.

[5] D. L. James, D. K. Pai, "DB-Tree: Output-Sensitive Collision Detection for Reduced Deformable Models", 2004, *ACM Translactions on Graphics (TOG), v. 23, n. 3.*

[6] S. Gottschalk, M. C. Lin, D. Manocha, "OOBTree: A Hierarchical Structure for Rapid Interface Detection", 1996, *Proceedings of SIGGRAPH*, pp. 171-180.

[7] J. D. Cohen, M. C. Lin, D. Manocha, M. K. Ponamgi, "I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments", 1995, *Proceedings 1995 Symposium on Interactive 3D Graphics*, pp. 189-196.

[8] H. Samet, "The Design and Analysis of Spatial Data Structures", *Addison-Wesley, Reading, MA, 1990. ISBN 0-201-50255-0.*