**[BSP Trees and Collision Detection](https://0fps.net/2009/02/12/bsp-trees-and-collision-detection/)**

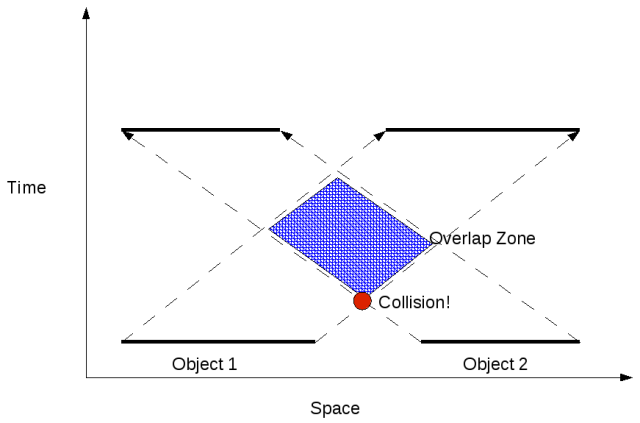
Posted on [February 12, 2009](https://0fps.net/2009/02/12/bsp-trees-and-collision-detection/)by [mikolalysenko](https://0fps.net/author/mikolalysenko/" \o "View all posts by mikolalysenko)

Recently I published a paper in CAD on [BSP tree merging via linear programming](http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6TYR-4TYYT8X-1&_user=443835&_coverDate=12%2F31%2F2008&_rdoc=6&_fmt=high&_orig=browse&_srch=doc-info(%23toc%235625%232008%23999599987%23857235%23FLA%23display%23Volume)&_cdi=5625&_sort=d&_docanchor=&_ct=9&_acct=C000020958&_version=1&_urlVersion=0&_userid=443835&md5=b80e614950d09342effb91bc1f9c1489).  At a very high level, the idea behind this paper was to replace the tree partitioning step with a linear programming feasibility test.  Doing this leads to a bunch of nice things, such as the removal of the partition function and all 9 of its special cases.  It also has the benefit that that for some suitable linear programming algorithms, the cost of evaluating feasibility can be amortized over the tree traversal giving an optimal linear-time output sensitive algorithm.

Of course as it always turns out with these things, I recently discovered that I was not the first to think of using linear programming for solving this problem.  Just last week [Prof. Alberto Paoluzzi](http://paoluzzi.dia.uniroma3.it/) visited our lab, and I was lucky enough to get a chance to talk to him about BSP trees.  As it just so happens, he proposed using linear programming for tree partitioning in his book [Geometric Programming for Computer-Aided Design](http://www.google.com/books?id=w6q6Bh4HcfEC&dq=geometric+programming+for+computer+aided+design&printsec=frontcover&source=bn#PPR7,M1) (which is actually a very good read.)  Of course, this shouldn’t diminish the contribution of our paper, as we do present a much simpler merge algorithm along with an analysis of both time complexity and robustness.  In any future works, I will definitely add a reference to his book.

Anyway, I wanted to talk about one idea which didn’t make it into that paper, but I still happen to think is rather interesting.  As an accidental side effect of our merging algorithm, we discovered a straightforward method for solving exact continuous collision detection between arbitrary moving non-convex objects.  The key idea to this process was initially suggested to me by my colleague Nicholas Smolinske (who sadly is no longer in graduate school.)  Here’s how it goes:

Suppose you have two convex shapes moving at a constant velocity.  For each shape, the object Minkowski summed with its trajectory traces out a 4D convex set space and time.  If you take these two sets for both shapes and intersect them, then the result is also a convex shape in 4D.  Moreover, the lowest point in time within the overlap region must be the point at which the two objects initially made contact.  This idea is illustrated in the following diagram:



Because all of the sets we are dealing with are convex, it is possible to exactly solve for the point of intersection using linear programming with an objective function c(x,t) = -t.  Now, the next question is how to do this with BSP trees?  Well, it turns out that one side effect from the algorithm proposed in our paper is that it ends up evaluating linear programming on each convex cell contained in the final BSP tree.  We can make use of this by simply picking the same objective function as above, and storing the minimal point over all convex cells as our result.  Alternatively, if we only want to get the point of intersection, then we don’t even need to compute the result of the merge so we may instead use it as a kind of driver for solving linear programming over the cell-complex hierarchically.

The idea of BSP trees representing shapes in space/time is not new, and to the best of my knowledge was first explored by [Agarwal et al.](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.2692) (they give some novel tree construction strategies, but don’t really deal with merging or related issues).  Another interesting way to look at this idea is that one could understand the BSP tree as generalizing some of the ideas of polytope hierarchies (such as Dobkin-Kirkpatrick) to non-convex sets.  Of course this still doesn’t answer the big question which is how to go about building decent BSP trees (or hierarchies for that matter) and also how to pick which order to merge them.  Someday I might go back and try refining these ideas to the point where they could make for a nice short conference paper, but for now it is just something fun to think about.