Computing the Width of a Point Set in 3-Space

Bernd Gärtner
Institute for Theoretical Computer Science
ETH Zürich
CH-8092 Zürich
Switzerland
gaertner@inf.ethz.ch

Thomas Herrmann*
Institute for Operations Research
ETH Zürich
CH-8092 Zürich
Switzerland
herrmann@ifor.math.ethz.ch

Abstract

We present a new algorithm to solve the 3-dimensional width problem, i.e. to determine two parallel planes of smallest distance such that the region between the two planes contains a given point set. Like the algorithm of Houle and Toussaint, our method has quadratic worst-case complexity but is much faster in practice. In contrast to Houle and Toussaint, we do not use plane sweep or point location techniques; instead, we apply simple methods from linear optimization.

The resulting implementation seems to be faster than an existing implementation of Houle and Toussaint's method using the LEDA library[1], and it is now part of the Computational Geometry Algorithms Library CGAL [2].

1 The Width Problem

Given a point set $\mathcal{P} \subseteq \mathbb{R}^d$, the width $\omega(\mathcal{P})$ of $\mathcal{P}$ is the smallest number ω such that there exists a unit vector u (*width direction*) and numbers $\overline{\delta}, \underline{\delta}$ such that $\omega = \overline{\delta} - \underline{\delta}$ and

$$\underline{\delta} \leq u^T p \leq \overline{\delta}, \quad \text{for all } p \in \mathcal{P}.$$

The hyperplanes $\{x \mid u^T x = \overline{\delta}\}$ and $\{x \mid u^T x = \underline{\delta}\}$ are perpendicular to u and define the *width planes* belonging to an optimal direction u.

It is clear that $\omega(\mathcal{P}) = \omega(conv(\mathcal{P}))$, which means that the computation of the convex hull is a natural preprocessing step. Moreover (Lemma 2.1), the width planes are supporting planes of the polytope $conv(\mathcal{P})$.

In the plane, the width of a set of n points can be computed in $O(n \log n)$ time, where only $O(n)$ time is needed once the convex hull has been computed [2]. We generalize this idea to 3-space, where the best theoretical solution is due to Agarwal and Sharir who presented an $\mathcal{O}(n^{3/2+\varepsilon})$ algorithm [1]. An earlier algorithm due to Houle and Toussaint has complexity $O(n^2)$ [2].

2 The Houle-Toussaint Algorithm

In the following we describe the basic outline of the Houle-Toussaint method in the version which has been implemented by Schwerdt et al. [3]. It starts with an observation that also underlies our new method.

Let F and F' be a pair of faces of $conv(\mathcal{P})$. F, F' are called *antipodal pairs*, if there exist two parallel supporting hyperplanes $h \neq h'$ such that $F \subseteq h$, $F' \subseteq h'$.

Lemma 2.1 (Houle and Toussaint). *There exists a pair of antipodal faces F, F' of $conv(\mathcal{P})$ such that $\omega(\mathcal{P}) = \omega(F \cup F')$, where F is a vertex and F' a facet, or both F and F' are nonparallel edges.*

Houle and Toussaint's algorithm enumerates all antipodal vertex-facet and edge-edge pairs and determines an optimal pair. The width ω, the width planes and the width direction u of a single pair can easily be computed in $O(1)$ time.

The enumeration is done by constructing an instance of a planar graph overlay problem. To get this instance, one splits $conv(\mathcal{P})$ into an upper and a lower hull and builds an upper and a lower graph, both embedded in the upper hemisphere $\mathbb{S}^2_+$ of the unit sphere $\mathbb{S}^2$.

The vertices of the upper graph G_u are the outer normals of the facets of the upper hull, where two vertices are connected by a great arc segment if the corresponding facets are adjacent. For the lower hull, one proceeds similarly, starting from the inner facet normals. This yields to the graph G_ℓ.

The crucial property is that there is a one-to-one correspondence between antipodal edge-edge pairs and intersections between edges of G_u and G_ℓ. Moreover, a vertex-facet pair translates to a vertex of one graph lying in a face of the other graph.

*European Graduate Program Combinatorics, Geometry, and Computation (CGC), Berlin - Zürich, `http://www.cgc.ethz.ch`
[1] `http://www.algorithmic-solutions.com`
[2] `http://www.cgal.org`

Thus, if there are I antipodal pairs, they can be reported in time $O(n \log n + I)$ by standard sweep and point location techniques (where a slight twist is that a hemisphere has to be swept).

The implementation of Schwerdt et al. uses the LEDA library of efficient data types and algorithms, and handles all degenerate cases. The computations do not incur any roundoff errors because exact multi-precision arithmetic is used.

This is the algorithm we are going to compare with; we also use exact arithmetic and deal with degeneracies. However, we argue that the sweep technique - which gets quite involved under degeneracies - is unnecessary. Instead, simple optimization techniques and the combinatorial properties of the convex hull can be used. While the resulting algorithm is fast in practice, its worst-case complexity can be $\Theta(n^2)$ even if I is small.

3 Rotating Plane Algorithm

Our new approach is to enumerate the antipodal vertex-facet and edge-edge pairs directly. Therefore we formulate the width-problem as an optimization problem having linear constraints but a non-convex objective function.

We are looking for the two parallel width-planes $h_1 : ax + by + cz + d = 0$ and $h_2 : ax + by + cz + 1 = 0$ that enclose $\mathcal{P}$ and have minimum distance. The problem is now to find a, b, c, d minimizing the (squared) distance $\frac{(1-d)^2}{a^2+b^2+c^2}$, such that the constraints

$$ap_x + bp_y + cp_z + d \leq 0$$

and

$$ap_x + bp_y + cp_z + 1 \geq 0$$

are satisfied for all points $p \in \mathcal{P}$. These inequalities encode the conditions that the points lie between the width planes.

The width itself is then the square root of the objective function value, the width direction is simply the unit vector according to (a, b, c) and the width planes are h_1 and h_2.

To enumerate all edge-edge and vertex-facet pairs directly we start with an arbitrary facet f and determine its antipodal vertices $V := \{v_1, \ldots, v_k\}$. This preprocessing step can be done in $O(n)$ time.

Once such an initial pair is known we rotate the parallel planes h_1, h_2, supporting V and f respectively, about an incident edge e of f until h_2 supports the other facet f' incident to e. During this rotation procedure we preserve parallelism and the supporting property of the two planes, and we report all edge-edge pairs belonging to e as well as the antipodal vertices of f'.

The critical step is as follows: Suppose an antipodal vertex-edge pair w, e and parallel planes h_1 and h_2 are given, supporting w and e respectively. We rotate h_2 about e. Two events might happen during this rotation procedure:

(i) Either h_2 supports a facet f' incident to e, or

(ii) h_1 supports an additional vertex v.

In the first case we found an antipodal vertex-facet pair, in the latter case an edge-edge pair occurred.

To detect which case occurs first, the adjacent vertices $v_1, \ldots, v_k$ of w are determined. These are the candidates to become tight, if the rotation angle increases. Then for each v_i, the rotation angle necessary to make v_i tight is computed. The vertex v with minimal angle α_v will be supported next during the rotation procedure unless f' is supported first. If both events happen simultaneously then we detected a degenerate case of an edge-facet or facet-facet pair. After reporting the new pair, we repeat this rotating procedure with e and the new vertex v until h_2 supports f'.

Degeneracy handling is very simple: The only case that can occur is that there is not a unique vertex v having minimal angle. To find all vertices which become tight next, we perform a graph search, starting in w, and exploring the neighborhoods of all tight vertices. We stop as soon as every vertex adjacent to a tight vertex is non-tight. This is much easier than the degeneracy handling necessary in sweep techniques.

To have linear worst case time for processing a single edge e (a "round"), one has to mark the vertices to prevent from looking at an edge twice. After reporting (wv, e) as an edge-edge pair the 'new' vertex v becomes active and the 'old' vertex w becomes marked as passive.[3] This means that we never process an edge incident to this vertex in the same round anymore. After a round, all vertices become unmarked again.

We do this rotating procedure for every edge e. We begin with plane h_2 supporting a facet $f \ni e$ and a parallel plane h_1 supporting the antipodal vertices of f.

The marking prevents from looking at a vertex twice during a round, and thus every edge is processed at most once. Hence all edge-edge pairs belonging to a certain edge e are enumerated in $O(n)$ worst case time. Since there are $O(n)$ edges the algorithm has worst case running time $O(n^2)$.

Potentially bad inputs for our algorithm are polytopes with high-degree vertices. It might happen that the neighborhoods of these vertices are checked in many rounds. This means, we might have $\Theta(n^2)$ runtime even if there are only very few edge-edge pairs.

[3] in the degenerate case, these might be sets of vertices

4 Results

A preliminary variant of the algorithm described here is embedded in CGAL as the `Width_3` package. We tested this code on a Sun Sparc Ultra-1 with random point sets differing in size and type.

The points were taken uniformly at random from a cube, from a ball or on a sphere. We compared the results to the existing implementation by Schwert et al. that uses point location and sweeping techniques (`Cwidth`).

Since both programs use a convex hull computation first, the time (in seconds) in Table 1 is measured without the initial convex hull step.

	Size	Cwidth	Width3
In Cube	1.000	9.13 s	2.94 s
	10.000	16.45 s	5.56 s
	100.000	23.08 s	10.75 s
	1.000.000	17.29 s	21.77 s
In Ball	100	3.72 s	1.15 s
	1.000	17.61 s	5.39 s
	10.000	64.94 s	19.37 s
	100.000	219.93 s	66.72 s
On Sphere	10	0.96 s	0.38 s
	100	14.78 s	5.95 s
	1.000	191.39 s	72.01 s
	10.000	2303.02 s	832.33 s

Table 1: Comparison between algorithms for different types of point sets.

Remark: Currently, we cannot explain why Cwidth outperforms our method for 1.000.000 points in a cube, while it is slower for less points.

5 Conclusion

Our algorithm has the big advantage that we can directly determine all edge-edge and vertex-facet pairs by simple calculations and some combinatorial properties. Instead of doing some preprocessing steps – like graph overlay – and sweeping with quite difficult degeneracy handling, we only have to look at neighborhoods of vertices. The price we pay for this is that our algorithm is not output-sensitive, meaning that the runtime is not dominated by the number of edge-edge pairs in general. Still, we have shown that our method can be faster in practice.

References

[1] Pankaj K. Agarwal and Micha Sharir. Efficient Randomized Algorithms for some Geometric Optimization Problems. *Discrete & Computational Geometry*, 16:317–337, 1996.

[2] Michael E. Houle and Godfried T. Toussaint. Computing the Width of a Set. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):761–765, 1988.

[3] Jörg Schwerdt, Michiel Smid, Jayanth Majhi, and Ravi Janardan. Computing the Width of a Three-dimensional Point Set: An Experimantal Study. In Kurt Mehlhorn, editor, *Proceedings WAE'98 Saarbrücken, Germany*, pages 62–73, August 1998.

Acknowledgements

We thank Michael Hoffmann and Sven Schönherr for their help in porting our code to CGAL, and Jörg Schwerdt for sharing his implementation of the Houle-Toussaint algorithm.