Discrete Comput Geom (2016) 56:860–865
DOI 10.1007/s00454-016-9785-3

A Simpler Linear-Time Algorithm for Intersecting Two Convex Polyhedra in Three Dimensions

Timothy M. Chan[1]

Received: 22 July 2015 / Revised: 23 March 2016 / Accepted: 13 April 2016 /
Published online: 25 April 2016

Abstract Chazelle [SIAM J Comput 21(4):671–696, 1992] gave a linear-time algorithm to compute the intersection of two convex polyhedra in three dimensions. We present a simpler algorithm to do the same.

Keywords Convex polyhedra · Intersection · Dobkin–Kirkpatrick hierarchy

1 Introduction

This note concerns the following problem: given two convex polyhedra of size $O(n)$ in 3-D, compute their intersection. Equivalently, the dual problem is to compute the convex hull of the two convex polyhedra, i.e., *merge* two convex hulls.

This is one of the most basic computational problems about convex polyhedra. Algorithms for the problem have been used as subroutines to solve many other problems in computational geometry (see [2] for just one example).

In the 1970s, Preparata and Hong [13] observed that two linearly separated convex hulls in 3-D can be merged in linear time. (Earlier Shamos and Hoey [14] observed the same for the special case of two linearly separated Delaunay triangulations in 2-D, and later Kirkpatrick [9] showed how to merge two arbitrary Delaunay triangulations

Editor in Charge: János Pach

A preliminary version of this work appeared in the *Proceedings of the* 31*st International Symposium on Computational Geometry*, 2015. Part of this work was done during the author's visit to the Hong Kong University of Science and Technology.

Timothy M. Chan
tmchan@uwaterloo.ca

[1] Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada

in 2-D in linear time.) The general problem was eventually settled a decade later when Chazelle [4] announced a linear-time algorithm for intersecting/merging two arbitrary convex polyhedra in 3-D.

Chazelle's algorithm, like many of his other works, is a tour de force. It started with a standard construction of the *Dobkin–Kirkpatrick* (*DK*) *hierarchies* [6,7] of the input polyhedra in both primal and dual space, but incorporated pages of intricate ideas and details. To give a flavor of the overall plan, we only mention that the running time satisfies a recurrence of the form $T(n) = 4\,T(\delta n) + O(n)$, which indeed solves to $T(n) = O(n)$ for a sufficiently small positive constant $\delta < 1/4$.

A thesis by Martin [11] described a simplification of Chazelle's algorithm that avoided switching back and forth with duality, but needed to navigate simultaneously in the DK hierarchies of the insides and outsides of the polyhedra. The details were still lengthy, and the recurrence was "improved" to $T(n) = 2\,T(\delta n) + O(n)$.

Chazelle's work dated back to a time when the unifying techniques of randomized geometric divide-and-conquer [5,12] were just starting to flourish. This prompts the question of whether more modern concepts like sampling, conflict lists, etc. might give a simpler explanation for why the problem can be solved in linear time. After all, at a gut level, this textbook problem shouldn't be that hard to solve (although one could say the same for the problem of triangulating a simple polygon [1,3]!).

In this note, we show that there is indeed a simpler linear-time algorithm for intersecting two convex polyhedra. Our solution ends up not requiring random sampling, but falls back to the DK hierarchy. We only need to navigate in the hierarchies of the outsides, and we don't need to switch between primal and dual space. Furthermore, we get a more usual recurrence $T(n) = T(\delta n) + O(n)$—in other words, a more conventional way of using DK hierarchies turns out to work after all! There are concrete advantages to having the better recurrence when considering other computational models; for example, the algorithm is more efficiently parallelizable. However, we believe the simplicity of the solution is what is the most valuable aspect of the work.

2 Preliminaries

We begin by computing a point o in the intersection of the two convex polyhedra; this can be done in linear time by 3-D linear programming [8] (known randomized algorithms are particularly simple), or in polylogarithmic time using DK hierarchies [6,7]. By translation, we may make o the origin. It suffices to compute the part of the intersection in $\{z > 0\}$. By a projective transformation $(x, y, z) \mapsto (x/z, y/z, -1/z)$, we can move o to $(0, 0, -\infty)$ and thus assume that both input polyhedra are unbounded from below, i.e., they are (the regions underneath) lower envelopes of planes. We assume that the planes are in general position, by standard perturbation techniques.

Given a set H of planes, let $\mathcal{P}(H)$ denote the region underneath its lower envelope. We say that H is *nonredundant* if all planes of H participate in the boundary of $\mathcal{P}(H)$. Given $P = \mathcal{P}(H)$, let $\mathcal{T}(P)$ denote a triangulation of P. More precisely, we triangulate each face of P, and for each triangle $v_1v_2v_3$ we take the region underneath

$v_1v_2v_3$ (a prism unbounded from below) as a cell of $\mathcal{T}(P)$. For any region Δ, the *conflict list* $H_{|\Delta}$ is the subset of all planes of H intersecting Δ.

A standard approach to computing the lower envelope of H is to pick a random sample H' of H, construct the lower envelope of the conflict list $H_{|\Delta}$ inside Δ for each cell $\Delta \in \mathcal{T}(\mathcal{P}(H'))$, and then glue the results together. Although we will not use randomization, we will adapt similar ideas.

Given $\mathcal{P}(H_1)$ and $\mathcal{P}(H_2)$ for two nonredundant sets H_1 and H_2 of planes, our problem is to compute $P = \mathcal{P}(H_1) \cap \mathcal{P}(H_2)$ (i.e., $P = \mathcal{P}(H_1 \cup H_2)$). In order to allow for a recursive algorithm, we need to strengthen the output requirement and require further information to be reported for each vertex v of P. Our key idea is this. Since v is in the intersection, we know that v is on or below $\mathcal{P}(H_j)$ for each $j \in \{1, 2\}$. Thus, there exist three vertices w_1, w_2, w_3 of $\mathcal{P}(H_j)$ that "witness" this fact, i.e., that have v below[1] $\triangle w_1w_2w_3$. We will require the algorithm to output one such triple for each v and j. It is important that we do not insist $w_1w_2w_3$ be a face of (a triangulated) $\mathcal{P}(H_j)$. Otherwise, one can show that finding such witnesses may require $\Omega(n \log n)$ comparisons in the worst case! Witnesses will make the generation of conflict lists easy; on the other hand, extra work will be required to find witnesses.

To summarize, we will solve the following stronger problem:

Problem: Given $\mathcal{P}(H_1)$ and $\mathcal{P}(H_2)$ for two nonredundant sets H_1 and H_2 of n planes, compute $P = \mathcal{P}(H_1) \cap \mathcal{P}(H_2)$, and for each vertex v of P and each $j \in \{1, 2\}$, report some vertices w_1, w_2, w_3 of $\mathcal{P}(H_j)$, called the *j-witnesses* of v, such that v is below $\triangle w_1w_2w_3$.

3 The Algorithm

We are now ready to give the algorithm outline to solve the problem:

Intersect$(\mathcal{P}(H_1), \mathcal{P}(H_2))$:
0. if H_1 and H_2 have size below a constant then return answer directly
1. for $j \in \{1, 2\}$:
2. choose an independent set of faces of $\mathcal{P}(H_j)$
3. let I_j be the planes defining these faces, and let $H'_j = H_j \setminus I_j$
4. obtain $\mathcal{P}(H'_j)$ from $\mathcal{P}(H_j)$
5. $P' =$ **Intersect**$(\mathcal{P}(H'_1), \mathcal{P}(H'_2))$

// now compute the intersection P of $\mathcal{P}(H_1)$ and $\mathcal{P}(H_2)$

6. for each $\Delta \in \mathcal{T}(P')$:
7. for $j \in \{1, 2\}$:
8. find the conflict list $H_{j|\Delta}$ by searching in the candidate list
$C_{j,\Delta} := \{\, h \in H_j : h \text{ lies below a } j\text{-witness of a vertex of } \Delta \,\}$
9. compute the intersection of $\mathcal{P}(H_{1|\Delta})$ and $\mathcal{P}(H_{2|\Delta})$ inside Δ
10. glue all the polyhedra from line 9 to get P

// now compute new witnesses for P

11. for each $\Delta \in \mathcal{T}(P')$:

[1] Throughout the paper, "below" means "below or is incident to" unless preceded by "strictly".

12. for $j \in \{1, 2\}$:
13. for each vertex v of P inside Δ:
14. find j-witnesses of v by searching in the candidate witness list
$W_{j,\Delta} := \{$ vertices w of $\mathcal{P}(H_j)$:
w is a j-witness of a vertex of Δ or
w is on a plane in $I_j \cap C_{j,\Delta}$ $\}$
15. return P with all its witnesses

We explain the algorithm in more detail. In line 2, independence means that the chosen faces do not share any edges. By applying a standard linear-time greedy algorithm to a planar graph in the dual, we can always choose an independent set of at least αn faces each with at most c vertices, for some constants α and c; for example, see Kirkpatrick's original paper [10], which has $\alpha = 1/24$ and $c = 11$.

Line 4 takes linear time: The difference of two polyhedra $\mathcal{P}(H_j)$ and $\mathcal{P}(H'_j)$ consists of disjoint constant-size regions (*caps*), since we are removing an independent set of constant-size faces (and the input planes are assumed to be nonredundant). Each cap is delimited by a face f in the independent set I_j, and the lower envelope of the at most c planes defining the faces adjacent to f, constructible in constant time. We can set up pointers from each vertex of $\mathcal{P}(H'_j)$ to the cap it belongs to. (The hierarchy of polyhedra produced from the recursion is called the *Dobkin–Kirkpatrick hierarchy* [6,7].)

Line 5 contains the main recursive call, where the number of planes in either input set drops to at most $(1 - \alpha)n$.

In line 8, we use witnesses for P' to help generate conflict lists. Any plane h in the conflict list $H_{j\,|\Delta}$ must lie below one u of the three vertices of Δ. Since u (a vertex of P') lies below $\triangle w'_1 w'_2 w'_3$ for its j-witnesses w'_1, w'_2, w'_3, it follows that h lies below some w'_i and must indeed be in the candidate list $C_{j,\Delta}$.

There are at most nine j-witnesses for the three vertices of Δ. Each j-witness w'_i (a vertex of $\mathcal{P}(H'_j)$) has at most $O(1)$ planes of H_j below it: namely, its three defining planes, and at most one plane from I_j strictly below it (which we can identify from the cap it belongs to). Thus, the candidate list $C_{j,\Delta}$ has constant size, and so each conflict list $H_{j\,|\Delta}$ can be generated in constant time.

Line 9 takes constant time even by a brute-force algorithm. Line 10 then takes linear total time (using a graph search to generate the edge skeleton).

We show, with a slightly subtle proof, that in line 14, we can indeed always find j-witnesses from the candidate witness list $W_{j,\Delta}$:

Lemma 1 *For a vertex v of P inside Δ, there exist $w_1, w_2, w_3 \in W_{j,\Delta}$ such that v lies below $\triangle w_1 w_2 w_3$.*

Proof Let W' be the at most nine j-witnesses of the three vertices of Δ. Then v lies below the upper hull of W' and is thus below the upper hull of some three points $w'_1, w'_2, w'_3 \in W'$ (which are vertices of $\mathcal{P}(H'_j)$). Let Γ be the region underneath $\triangle w'_1 w'_2 w'_3$.

Since v is in $\mathcal{P}(H_j) \cap \Gamma$, there exist three vertices u_1, u_2, u_3 of $\mathcal{P}(H_j) \cap \Gamma$ such that v is below $\triangle u_1 u_2 u_3$. (Note that each u_i may not necessarily be a vertex of $\mathcal{P}(H_j)$ as it could lie on the boundary of Γ.) For each $i \in \{1, 2, 3\}$, we claim that u_i is below the upper hull of $W_{j,\Delta}$:

Case 1: u_i is a vertex of Γ, i.e., $u_i \in \{w'_1, w'_2, w'_3\}$. Then u_i is a vertex of $\mathcal{P}(H'_j)$. Since u_i is in $\mathcal{P}(H_j)$, it follows that u_i is a vertex of $\mathcal{P}(H_j)$. Thus, u_i is in $W_{j,\Delta}$ by definition of $W_{j,\Delta}$, and the claim is trivially true.

Case 2: u_i is not a vertex of Γ. Then u_i must be defined by at least one plane $h \in H_j$ that intersects the interior of Γ. This plane h is strictly below at least one of w'_1, w'_2, w'_3 and so must be a member of I_j and also a member of $C_{j,\Delta}$. Now, u_i lies in the face of $\mathcal{P}(H_j)$ defined by h; all the vertices of this face are in $W_{j,\Delta}$ by definition of $W_{j,\Delta}$, and the claim is again true.

Since v is below $\triangle u_1u_2u_3$, it follows that v is below the upper hull of $W_{j,\Delta}$ and is thus below $\triangle w_1w_2w_3$ for some three vertices $w_1, w_2, w_3 \in W_{j,\Delta}$. □

The candidate witness list $W_{j,\Delta}$ has constant size, since there are at most nine j-witnesses for the three vertices of Δ, each plane in I_j contains at most c vertices, and there are $O(1)$ planes in $C_{j,\Delta}$. So, line 13 can be done in constant time by brute force. The entire loop in lines 11–14 then takes linear total time.

The overall running time of the algorithm satisfies the recurrence $T(n) = T((1-\alpha)n) + O(n)$, which solves to $T(n) = O(n)$.

4 Remarks

An alternative, slightly slower algorithm. There is a more "standard" algorithm based on sampling, without using witnesses, that gives almost linear $n2^{O(\log^* n)}$ expected time. For the readers who are familiar with randomization techniques [5,12] and enjoy comparisons of different approaches, we briefly sketch the alternative:

First consider a multiset version of H_j where the multiplicity $w_j(h)$ (the *weight*) of each plane $h \in H_j$ is the size of the face of $\mathcal{P}(H_j)$ defined by h. The multiset still has $O(n)$ size. We draw a random sample H'_j of the multiset of size $r = O(n/\log n)$. We construct $\mathcal{P}(H'_1)$, $\mathcal{P}(H'_2)$, and their intersection P' by an $O(r \log r)$-time algorithm, which takes $O(n)$ time.

For each vertex v of $\mathcal{P}(H'_j)$, we can construct its conflict list $H_{j|v}$ (the list of all planes of H_j below v) as follows: first find an initial plane of H_j below v by a point location query in the xy-projection of $\mathcal{P}(H_j)$; then find all planes of H_j below v by a graph search over the faces of $\mathcal{P}(H_j)$. This works because the planes below v correspond to the faces visible to v, which are connected in the boundary of $\mathcal{P}(H_j)$ (assuming that H_j is nonredundant). We can preprocess in linear time for point location in $O(\log n)$ time [10], so the $O(r)$ point location queries cost $O(n)$ total time. The graph search takes time proportional to the weight of $H_{j|v}$. The total time over all v is $O(r \cdot n/r) = O(n)$ in expectation, by Clarkson and Shor's analysis [5].

Next, for each vertex v of P', we can compute its conflict list $H_{j|v}$ as follows: first find a cell $\Delta \in \mathcal{T}(\mathcal{P}(H'_j))$ containing v by a point location query in the xy-projection of $\mathcal{T}(\mathcal{P}(H'_j))$; then search in the conflict lists of the three vertices of Δ (which are vertices of $\mathcal{P}(H'_j)$) found in the preceding paragraph. The $O(r)$ point location queries again cost $O(n)$ total time. So, this step again takes $O(n)$ expected total time.

For each cell $\Delta \in \mathcal{T}(P')$, we can now generate the conflict list $H_{j|\Delta}$ from the conflict lists of the three vertices of Δ (which are vertices of P') found in the preceding

paragraph. We then recursively compute the intersection of $\mathcal{P}(H_{1|\Delta})$ and $\mathcal{P}(H_{2|\Delta})$ inside Δ, and glue the polyhedra together.

By Clarkson and Shor's analysis [5], the total expected running time satisfies the recurrence $T(n) = \sum_i T(n_i) + O(n)$ where $\max_i n_i = O((n/r)\log n) = O(\log^2 n)$ with high probability and $\sum_i n_i$ has expected value $O(r \cdot n/r) = O(n)$. With $O(\log^* n)$ iterations, this yields an expected time bound of $n2^{O(\log^* n)}$.

An open problem. An interesting question is whether we can similarly merge lower envelopes of *pseudo-planes* in 3-D in linear time, under an appropriate definition of pseudo-planes where three pseudo-planes may intersect in at most one point. This would have applications to merging two additively weighted Voronoi diagrams in 2-D, for instance. Our concept of witnesses is not immediately generalizable, although the alternative $n2^{O(\log^* n)}$-time randomized algorithm could still work, at least for the case of 2-D additively weighted Voronoi diagrams. Another similar open question is whether two intersections of unit balls in 3-D can be merged in linear time.

Acknowledgments The author thanks Stefan Langerman for discussion on these problems.

References

1. Amato, N.M., Goodrich, M.T., Ramos, E.A.: A randomized algorithm for triangulating a simple polygon in linear time. Discrete Comput. Geom. **26**(2), 245–265 (2001)
2. Chan, T.M.: Deterministic algorithms for 2-D convex programming and 3-D online linear programming. J. Algorithms **27**(1), 147–166 (1998)
3. Chazelle, B.: Triangulating a simple polygon in linear time. Discrete Comput. Geom. **6**, 485–524 (1991)
4. Chazelle, B.: An optimal algorithm for intersecting three-dimensional convex polyhedra. SIAM J. Comput. **21**(4), 671–696 (1992)
5. Clarkson, K.L., Shor, P.W.: Application of random sampling in computational geometry. II. Discrete Comput. Geom. **4**, 387–421 (1989)
6. Dobkin, D.P., Kirkpatrick, D.G.: A linear algorithm for determining the separation of convex polyhedra. J. Algorithms **6**(3), 381–392 (1985)
7. Dobkin, D.P., Kirkpatrick, D.G.: Determining the separation of preprocessed polyhedra—A unified approach. In: Proceedings of the 17th International Colloquium on Automata, Languages and Programming, pp. 400–413 (1990)
8. Dyer, M., Megiddo, N., Welzl, E.: Linear programming. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, Chapter 45, 2nd edn. CRC Press, New York (2004)
9. Kirkpatrick, D.G.: Efficient computation of continuous skeletons. In: Proceedings of the 20th Annual Symposium on Foundations of Computer Science, pp. 18–27 (1979)
10. Kirkpatrick, D.G.: Optimal search in planar subdivisions. SIAM J. Comput. **12**(1), 28–35 (1983)
11. Martin, A.K.: A simple primal algorithm for intersecting 3-polyhedra in linear time. Master's thesis, Department of Computer Science, University of British Columbia. https://circle.ubc.ca/handle/2429/30114 or http://www.cs.ubc.ca/cgi-bin/tr/1991/TR-91-16 (1991)
12. Mulmuley, K.: Computational Geometry: An Introduction Through Randomized Algorithms. Prentice Hall, Englewood Cliffs (1993)
13. Preparata, F.P., Hong, S.J.: Convex hulls of finite sets of points in two and three dimensions. Commun. ACM **20**(2), 87–93 (1977)
14. Shamos M.I., Hoey D.: Closest-point problems. In: Proceedings of the 16th Annual Symposium on Foundations of Computer Science, pp. 151–162 (1975)