Optimizing the computation of sequences of determinantal predicates

Ioannis Z. Emiris* Vissarion Fisikopoulos* Luis Peñaranda*

Abstract

Orientation is the core predicate in many important geometric algorithms, such as convex hull and triangulation computations. This operation reduces to compute the sign of a determinant. We propose a method that improves the amortized complexity of the determinants involved in a convex hull computation. Moreover, we study how can we use the computation done in a convex hull construction to improve the construction of subsequent convex hulls. Our two main tools are the dynamic determinant computation and the reuse of determinantal minors. We finally validate our approach by implementing and testing one of our methods.

Keywords Orientation, determinant, convex hull, triangulation, CGAL implementation.

1 Introduction

In general dimension d, the orientation of $d+1$ points is computed as the determinant of a matrix containing the coordinates of the points as columns, plus one last row full of ones. As the dimension grows, a higher percentage of the computation time is consumed by these predicates. In this paper we study the computation of sequences of determinantal predicates in a single, or in a sequence of, *exact* convex hull computations. A typical example of the latter is the computation of many regular triangulations of the same pointset, which arise in algorithms that compute secondary [16] and resultant polytopes [8]. Our study can be extended to other determinantal predicates such as inCircle/inSphere and Volume.

Our contribution is twofold. First, we propose a method that improves the amortized complexity of the determinants involved in a convex hull computation (Sect. 2). This method uses the dynamic determinant evaluation procedure introduced in [18] to solve problems in graphs. Moreover, we study sequences of convex hull computations and propose a method that uses the computation done in a convex hull construction to improve the construction of subsequent convex hulls (Sect. 3). Second, we implement a simple method that optimizes the computation of subsequent determinantal predicates in both single and sequence of convex hull computations. The experiments show in the first scenario a speedup up to dimension 5 and in the second a speedup of 100 times when the dimension is 6 (Sect. 4).

Let us review previous work. There is a variety of algorithms and implementations for computing the determinant of a $d \times d$ matrix. Denoting their complexity by $O(d^\omega)$, the best known ω is 2.697263 [13]. However good asymptotic complexity does not imply good behavior in practice for small and medium dimensions. For instance, LinBox [6] implements algorithms with state-of-the-art asymptotic complexity, but introduces a significant overhead in medium dimensions, and seems most suitable in very high dimensions (typically > 100). There exists also a variety of algorithms for determinant sign computation [1, 4] with good asymptotic complexity. Eigen [11] seems to be suitable for medium to high dimensions, whereas CGAL [5] determinants proved to be efficient in low to medium dimensions. Decomposition methods have complexity $O(d^3)$, but they require the construction of intermediate objects, which adds a constant cost in computations and makes them slow in practice. There exist other simple methods such as [17] with complexity $O(d^4)$ and [2] with complexity $O(dM(d))$ where $M(d)$ is the complexity of matrix multiplication. Albeit simpler to implement, they also construct intermediate objects and thus suffer the same practical problem as decomposition methods. Laplace expansion, the simplest determinant method, does not need intermediate matrices and has complexity $O(d!)$ (note however that $d! < d^3$ for $d < 6$).

Concerning of sequences of determinants, the reference software for computing triangulations of a set of points, TOPCOM [16], computes all Orientation determinants that will be needed and stores their signs.

2 Convex Hull Computation

This section discusses the problem of dynamic computation of determinants produced in geometric computations such as convex hull and triangulation algorithms. These algorithms rely on the signs of determinantal Orientation predicates. The orientation of $d+1$ d-dimensional points is the sign of the determinant of a $(d+1) \times (d+1)$ matrix, where each column contains the d coordinates of each point, plus a 1 on

*National and Kapodistrian University of Athens, Department of Informatics and Telecommunications, Athens, Greece. {emiris,vissarion,lpenaranda}@di.uoa.gr. Partial support from project "Computational Geometric Learning", which acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET-Open grant number: 255827.

This is an extended abstract of a presentation given at EuroCG 2012. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear in a conference with formal proceedings and/or in a journal.

the last place.

In the *dynamic determinant problem*, a $d \times d$ matrix A is given. Then, allowing some preprocessing, we should be able to handle updates of elements of A and queries for the current value of the determinant.

Lemma 1 [18, Thm.2] *The dynamic determinant problem can be solved in $O(d^\omega)$ time for preprocessing, $O(d^2)$ time for one column updates and $O(1)$ time for queries.*

We want to apply this result to incremental convex hull algorithms. Let $\mathcal{A} \subset \mathbb{R}^d$ be a pointset with $\mathrm{CH}(\mathcal{A})$ of dimension d, where $\mathrm{CH}(\cdot)$ denote the convex hull. In particular, we focus on the Beneath-and-Beyond (BB) algorithm [7, Sect. 8.4], where the construction of $\mathrm{CH}(\mathcal{A})$ is essentially the construction of a placing triangulation of $\mathrm{CH}(\mathcal{A})$ [15, Sect. 4.3]. Given $\mathcal{A}$, the algorithm at the first step computes a d-simplex and at every step it adds points by keeping a triangulated convex hull of the inserted points. At each step, a new point p is connected with all its visible facets. In order to determine if a facet f is visible from p we have to compute an Orientation predicate that involves p and the points of f. For each visible facet f a new full dimensional face (cell) σ is created, by connecting p to the points of f. Every cell σ has a vertex v that is not a vertex of f. At every step if we know the value of the Orientation determinant involves the vertices of cell σ we only need to compute the new value if we change v with p in this determinant. We can use Lem. 1 to compute this determinant in $O(d^2)$.

The idea is to store the value of Orientation in its corresponding cell together with an inverse $d \times d$ matrix that is used for updates [18, Sect. 4]. Denote t the number of cells of the resulting placing triangulation of $\mathrm{CH}(\mathcal{A})$ that stores the determinantal values and the inverse matrices. Note that every cell constructed by the algorithm corresponds to an Orientation predicate involving its points. The total number of predicates computed is thus t. The first predicate, corresponding to the initial d-simplex, is computed in $O(d^\omega)$. Then, the following holds.

Theorem 2 *The Orientation predicates of the BB algorithm can be computed in $O(d^2)$ amortized time and $O(d^2 t)$ space, where d is the dimension of the points and t is the number of cells of the resulting placing triangulation of the convex hull.*

This result improves the amortized computational complexity of the determinants involved in convex hull computation using the BB from $O(d^\omega)$ to $O(d^2)$.

The method presented in this section can be adapted to other algorithms which compute determinants of matrix updates. For instance, walk algorithms for point location in triangulations compute orientations involving vertices of neighboring cells.

3 Sequences of Convex Hull Computations

In this section we study the problem of computing a sequence of regular triangulations of a d-dimensional pointset $\mathcal{A}$ for given lifting vectors w. This can be done by computing the convex hull of the lifted $\mathcal{A}$ according to w and project its upper hull. This idea has already appeared in [9] where, given a system of $n+1$ polynomials in n variables, the proposed algorithm computes the Newton polytope (or a projection of it) of the resultant of this system. Given the input polynomials, it first defines a pointset $\mathcal{A}$ in dimension $2n$ (*i.e.*, $d = 2n$) and it needs to compute the regular triangulations of $\mathcal{A}$ given some lifting vectors. Similar problems often appear in combinatorial geometry, as in computations of secondary polytopes [16] or in the computation of volumes (*i.e.*, the Volume predicate) of the cells of a triangulation. Again motivated by [9], we want to compute the volume of the cells of regular triangulations of $\mathcal{A}$ given some lifting vectors.

The basic observation is that in every convex hull computation the input points differ only in their last coordinate, which comes from different lifting vectors. Thus, we expect many Orientation predicates that appear in this computation to be similar. Consider now the $d \times |\mathcal{A}|$ matrix with the points of $\mathcal{A}$ as columns. Define P as the extension of this matrix by adding a lifting vector $\vec{w}$ as the last row. We use the Laplace expansion along the last row for computing the determinant of the square submatrix formed by any $d+1$ columns of P; wlog these are the first $d+1$ columns $a_1, \ldots, a_{d+1}$. Let $\langle a_1, \ldots, a_{d+1} \rangle \setminus i$ be the vector $\langle a_1, \ldots, a_{d+1} \rangle$ without its i-th element; $P_{\langle a_1, \ldots, a_{d+1} \rangle \setminus i}$ is the $d \times d$ matrix obtained from the d first elements of the columns whose indices are in $\langle a_1, \ldots, a_{d+1} \rangle \setminus i$.

Orientation is the sign of the determinant of $P^{hom}_{\langle a_1, \ldots, a_{d+2} \rangle}$, constructed by columns $a_1, \ldots, a_{d+2}$ when we add $\vec{1} \in \mathbb{R}^{d+2}$ as a last row. Computing a regular triangulation is a long sequence of such predicates with different a_i's. We expand along the last two rows and compute the determinants $|P_{\langle a_1, \ldots, a_{d+2} \rangle \setminus \{i,j\}}|$ for $\binom{d+2}{2}$ combinations of $i, j \in \{1, \ldots, d+2\}$. The Volume predicate equals the determinant of $P^{hom}_{\langle a_1, \ldots, a_{d+1} \rangle}$, constructed by columns $a_1, \ldots, a_{d+1}$ when we replace its last row by $\vec{1} \in \mathbb{R}^{d+1}$.

Example 1 *Consider the following matrix, corresponding to a poinset $\mathcal{A}$, given the lifting vector $\vec{w} = \{w_1, \ldots, w_9\}$.*

$$P = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & w_7 & w_8 & w_9 \end{pmatrix}$$

Consider the computation of Orientation as a compu-

tation of $\binom{6}{4} = 15$ (4×4) *minors using the Laplace expansion. If we have computed* $|P^{hom}_{\langle 1,2,3,4,5,6\rangle}|$ *then the computation of* $|P^{hom}_{\langle 1,2,3,4,5,7\rangle}|$ *needs only* $\binom{6}{4} - \binom{5}{4} = 10$ *new minors. More interestingly, by giving a new lifting vector* $\vec{w'}$ *we can compute* $|P'^{\,hom}_{\langle 1,2,3,4,5,6\rangle}|$ *without computing any new minors.*

Our contribution consists in maintaining a data structure with the computed minors which are independent of $\vec{w}$. Once computed, they can be reused at subsequent steps of the algorithm. The main advantage of our scheme is that, for a new $\vec{w}$, the only change in P is its last row, hence computing the new determinants is done by reusing stored minors.

Lemma 3 *The complexity of Orientation and Volume predicates is* $O(d)$ *when all minors have been already computed.*

Proof. In Volume predicate, we expand along the row w, performing $O(d)$ arithmetic operations. For Orientation, we reduce the computation of the expansion of the last two rows to $O(d)$ by hashing the minors of the matrix with the row $\vec{1}$. □

4 Experiments on Hashing Determinants

We implemented in C++ the *hashing determinants* scheme, which consists in computing determinants using the Laplace expansion as in Sect. 3, storing all the computed minors in a hash table.

To avoid constructing a new matrix each time we compute a determinant, we keep a table T with all the points. The evaluation of the determinant using the Laplace expansion is performed recursively by using only the indices on T.

For the hash table implementation, we looked for a hashing function that takes as input a vector of integers (*i.e.*, the indices in T of the points that form the matrix) and returns a hash value that minimizes collisions. We considered many different hash functions, including some variations of the well-known FNV hash [10]. We obtained the best results with the implementation of Boost Hash [12], which has constant lookup cost in practice. For convex hull computation in general dimension we rely on CGAL's package `triangulation`, under development [3].

All the experiments that follow ran on an Intel Core i5-2400 3.1GHz, with 6MB L2 cache and 8GB RAM, under 64-bit Debian GNU/Linux. We performed experimental tests of our implementation in the scenario analysed in Sect. 3: we computed a sequence of convex hulls in the context of computing resultant polytopes [9]. Fig. 1 (left) shows a speedup of 100 times for 6 dimensional convex hulls. For 4-dimensional convex hulls, we gain a speedup of 18 times (we refer to [9] for graphs not shown here). An interesting point is that, with our technique, we computed in $<$ 2min an instance where $|\mathcal{A}| = 37$, that would take $>$ 1hr to compute otherwise. Thus, when the dimension and $|\mathcal{A}|$ becomes larger, this method allows us to compute instances that would be intractable otherwise.

The computation of one convex hull can also benefit from the hashing determinants scheme when using Laplace expansion. Even if a determinant is never computed twice, minors of size $< d$ can be reused. Based on this observation, we performed tests on the computation of one convex hull (Fig. 1, right). We generated random rational points uniformly distributed in the cube $[-100, 100]^d$, where the dimension d ranges between 2 and 6. Experiments show that our method performs better up to dimension 5, as expected. In dimension 6, hashing of minors does not help because Laplace expansion becomes very slow. We expect much better results for higher dimensions by implementing the algorithm of Sect. 2.

The drawback of our approach is the storage complexity, which is in $O(n^d)$. The hash table can be cleared at any moment to limit memory consumption, at the cost of dropping all previously computed minors. In our experiments, we obtained a good tradeoff between efficiency and memory consumption by clearing the hash table when it reaches 10^6 minors.

A more sophisticated approach, inspired from the paging problem in computer systems, consists in replacing a determinant in the data structure with the newly computed one when the data structure has reached a threshold size. A *replacement strategy* specifies the choice of which determinant to evict (for instance, the less used determinant or the less recently used determinant). However, the hash table data structure is not the most suitable for this. A good candidate to replace it are *tries* [14], which permit to index the stored values using a tree of depth d (the size of the matrices), thus yielding a $O(d)$ lookup and insertion time (since d is small, this is comparable to the cost of our hashing function). Tries are more suitable to implement replacement strategies. For instance, a *trie* permits us to store, on each node, the number of lookups on its successors. This information would permit to prune the less used subtrees at a given depth.

5 Future Work

We expect to quantify the gain of the BB algorithm when using dynamic determinants as suggested in Sect. 2 by performing a complexity analysis in which the dimension is not treated as a constant (to our knowledge, this analysis was not performed before).

For the computation of a sequence of convex hulls, we should consider implementing other determinant algorithms, to be more competitive dimensions > 5, and *tries*, to improve memory handling. For the computation of one single convex hull, we may implement

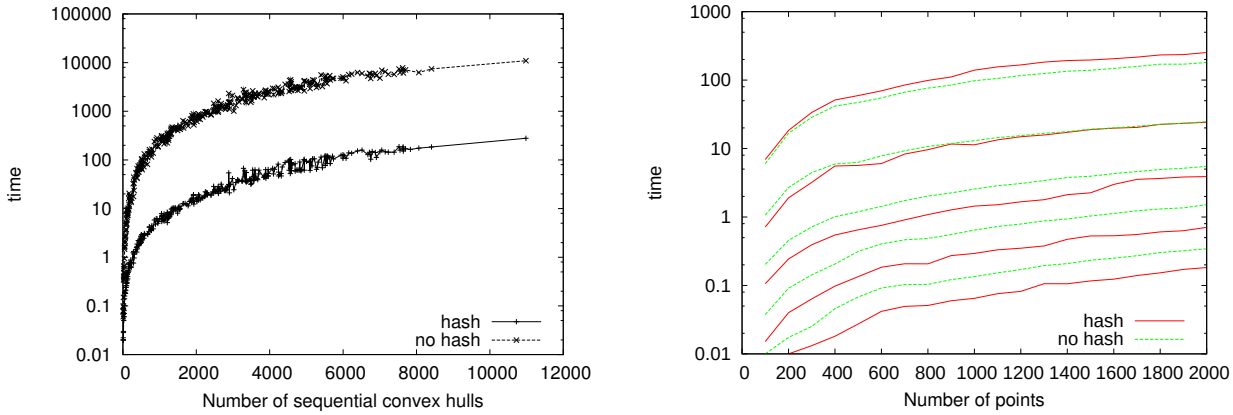

Figure 1: To the left, time (in seconds) for computing many 6-dimensional convex hulls of a set of points changing their lifting, hashing and non-hashing minors; the number of input points range from 10 to 45. To the right, time (in seconds) for computing one convex hull of a point set $\mathcal{A}$ as function of the number of input points. Graphs correspond to different ambient dimensions of the input points, *i.e.*, from bottom to top $\dim(\mathcal{A}) = 2, \ldots, 6$.

the algorithm presented in Sect. 2. Finally, it would be interesting to develop an interface that permits using the hashed determinants scheme transparently (*i.e.*, without knowledge of points' indices), to permit its use in any algorithm that uses determinants. This would also allow us to submit all our implementations to CGAL.

Finally, we should consider the extension of our approach to the 1-skeleton representation of a triangulation [3] which reduces the memory consumption with a penalty of slower running times.

Acknowledgments We thank O. Devillers and S. Hornus for discussions on `triangulation`, G. Rote for his suggestion on [2, 17], as well as the anonymous reviewers for their useful comments.

References

[1] J. Abbott, M. Bronstein, and T. Mulders. Fast deterministic computation of determinants of dense matrices. In *ISSAC*, pp. 197–203, 1999.

[2] R.S. Bird. A simple division-free algorithm for computing determinants. *Inf. Process. Lett.*, 111:1072–1074, Nov. 2011.

[3] J.-D. Boissonnat, O. Devillers, and S. Hornus. Incremental construction of the Delaunay triangulation and the Delaunay graph in medium dimension. In *SoCG*, pp. 208–216, 2009.

[4] H. Brönnimann, I.Z. Emiris, V. Pan, and S. Pion. Sign determination in Residue Number Systems. *Theor. Comp. Science, Spec. Issue on Real Numbers & Computers*, 210(1):173–197, 1999.

[5] CGAL: Computational geometry algorithms library. http://www.cgal.org.

[6] J.-G. Dumas, T. Gautier, M. Giesbrecht, P. Giorgi, B. Hovinen, E. Kaltofen, B.D. Saunders, W.J. Turner, and G. Villard. Linbox: A generic library for exact linear algebra. In *ICMS*, pp. 40–50, 2002.

[7] H. Edelsbrunner. *Algorithms in combinatorial geometry*. Springer-Verlag New York, Inc., New York, NY, USA, 1987.

[8] I.Z. Emiris, V. Fisikopoulos, and C. Konaxis. Exact and approximate algorithms for resultant polytopes. In *Proc. Europ. Workshop Computat. Geometry*, Assisi, Italy, 2012.

[9] I.Z. Emiris, V. Fisikopoulos, C. Konaxis, and L. Peñaranda. An output-sensitive algorithm for computing projections of resultant polytopes. arXiv:1108.5985v2 [cs.SC], 2011.

[10] G. Fowler, L.C. Noll, and P. Vo. FNV hash. www.isthe.com/chongo/tech/comp/fnv/, 1991.

[11] G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[12] D. James. Boost functional library. www.boost.org/libs/functional/hash, 2008.

[13] E. Kaltofen and G. Villard. On the complexity of computing determinants. *Computational Complexity*, 13:91–130, 2005.

[14] D.E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.

[15] J.A. De Loera, J. Rambau, and F. Santos. *Triangulations: Structures for Algorithms and Applications*, volume 25 of *Algorithms and Computation in Mathematics*. Springer, 2010.

[16] J. Rambau. TOPCOM: Triangulations of point configurations and oriented matroids. In A.M. Cohen, X-S. Gao, and N. Takayama, eds., *Math. Software: ICMS*, pp. 330–340. World Scientific, 2002.

[17] G. Rote. Division-free algorithms for the determinant and the Pfaffian: algebraic and combinatorial approaches. In *Comp. Disc. Math.*, pp. 119–135, 2001.

[18] P. Sankowski and M. Mucha. Fast dynamic transitive closure with lookahead. *Algorithmica*, 56:180–197, 2010. 10.1007/s00453-008-9166-2.