Optimizing the computation of sequences of determinantal predicates

Ioannis Z. Emiris* Vissarion Fisikopoulos* Luis Peñaranda*

Abstract

The orientation predicate is the core procedure in many important geometric algorithms, such as convex hull and triangulation computations. As the dimension of the computation space 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 signle or a sequence of convex hull computations.

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 reusage of determinantal minors. Finally, 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 single convex hull scenario a speedup up to dimension 5 and in sequences of convex hulls a speedup of 100 times when the dimension is 6.

Keywords: orientation predicate, determinant, convex hull, triangulation, CGAL implementation.

1 Introduction

The orientation predicate is the core procedure in many important geometric algorithms, such as convex hull and triangulation computations. 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 signle or a sequence of convex hull computations. A typical example of the second is the computation of many regular triangulations of the same pointset arise in algorithms that compute secondary [14] and resultant polytopes [7]. This 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 [16] to solve problems in graphs. Moreover, we study sequences of convex hull computations and propose a method that use 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. By denoting $O(d^\omega)$ their complexity, the best current ω is 2.697263 [11]. However good asymptotic complexity does not imply respectively good behavior in practice for small and medium dimensions. For instance, LinBox [5] which implements algorithms with state-of-the-art asymptotic complexity, introduces a significant overhead in medium dimensions, and seems most suitable in very high dimensions (typically > 100). Eigen [9] seems to be suitable for medium to high dimensions, whereas CGAL [4] determinants proved to be efficient in low to medium dimensions. On the other hand, decomposition methods have complexity $O(n^3)$, but they require the construction of intermediate objects, which adds a constant cost in computations and makes them slow in practice. There exist other methods that avoid divisions, such as [15] with complexity $O(n^4)$ and [2] with complexity $O(nM(n))$ where $M(n)$ is the complexity of matrix multiplication. Albeit simpler to implement, these algorithms also construct intermediate objects and thus suffer the same practical problem as decomposition methods. This intermediate object construction problem proved to be, in practice, the bottleneck of determinant computation in small dimensions, and is the reason why we opted for an implementation of the Laplace expansion algorithm.

*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.

There exists also a variety of algorithms for determinant sign computation [3, 1]. On the side of sequences of determinants, the reference software for computing triangulations of a set of points, TOPCOM [14], computes all orientation determinants that will be needed and stores their signs.

2 Convex hull computations

This section discusses the problem of dymanic 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 predicate 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 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 [16, Thm.2] *The dynamic determinant problem can be solved in $O(d^\omega)$ for preprocessing, $O(d^2)$ for one column updates and $O(1)$ 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 [6, Sect. 8.4], where the construction of $\mathrm{CH}(\mathcal{A})$ is essentially the construction of a placing triangulation of $\mathrm{CH}(\mathcal{A})$ [13, 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 the orientation predicate in its corresponding cell together with an inverse $d\times d$ matrix that is used for updates [16, 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. Thus, the total number of predicates computed is t. We only compute the first predicate, that corresponds to the initial d-simplex, from scratch in $O(d^\omega)$. Then, the following holds.

Theorem 2 *The orientation predicates of BB algorithm can be computed in $O(d^2)$ amortized time and $O(d^2t)$ space.*

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

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 [7] 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 $2n$, ie. $d=2n$, 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 [14] or in the computation of volumes (ie. the volume predicate) of the cells of a triangulation. Again motivated by [7], 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 the different lifting vectors. Thus, we expect that many orientation predicates appear in this computation will 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 lifting values w as the last row. We use the Laplace (or cofactor) 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 $\langle a_1,\ldots,a_{d+1}\rangle\setminus i$.

The orientation predicate 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 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 $\vec{1} \in \mathbb{R}^{d+1}$. We expand along the last two rows and compute the minor determinants, as in the previous case.

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

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

Consider the computation of an orientation predicate as a computation 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 $\widehat{w'}$ 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 w and once computed they can be reused at subsequent steps of the algorithm. The main advantage of our scheme is that, for a new w, the only change in P is its last row, hence computing the new determinants can be done by reusing stored minors.

Lemma 3 *The complexity of the orientation and volume predicates is $O(n^2)$ and $O(n)$, respectively, if all minors have already been computed.*

4 Hashing determinants: implementation and experiments

We propose and implement in C++ the *hashing determinants* scheme which consists in using the Laplace expansion to compute determinants and store the computed minors in a hash table. We perform experimental tests of our implementation in the two scenarios analysed in Sect. 2, 3. In the first we compute the convex hull of an input pointset and in the second we compute a sequence of convex hulls in the context of computing resultant polytopes [7]. All timings are on an Intel Core i5-2400 3.1GHz, with 6MB L2 cache and 8GB RAM, running 64-bit Debian GNU/Linux.

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

For the hash table implementation, we looked for a hashing function that takes as input a vector of `size_t` and returns a `size_t` that minimizes collisions. We considered many different hash functions, including some variations of the well-known FNV hash [8]. We obtained the best results with the implementation of Boost Hash [10], which performs a *left fold* on the input vector, using start value 0 and `f(x,y)=x^(y+0x9e3779b9+(x<<6)+(x>>2))`, and has a constant lookup cost in practice. For convex hull computations we rely on CGAL, using mainly a preliminary version of package `triangulation` under development, working in general dimension.

In the case of a single convex hull, we show in Fig. 1 (left) the timings of computing the convex hull of points with rational coefficients, 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. We expect much better results for higher dimensions by implementing the algorithm of Sect. 2 which suggests to store an inverse matrix instead of storing the minors of the Laplace expansion.

In the case of sequential determinants, we gain a speedup of 18 times for 4 dimensional convex hulls. In dimension 6 we gain a speedup of 100 times (Fig. 1 (right)). We computed in $<$ 2min an instance where $|\mathcal{A}| = 37$ and 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.

An advantage of this scheme is that it doesn't need to construct a matrix each time it computes a predicate. Many determinant algorithms modify the input matrix; this makes necessary to create a new matrix and introduces a constant overhead on each minor computation.

The drawback is the amount of storage, which is in $O(n!)$. 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, is the following. When a new determinant is computed and the size of the data structure has reached a threshold size, evict a determinant from the data structure and add the new one. A *replacement strategy* specifies the choice of which determinant to evict. However, the hash table data structure is not the most suitable for this. On the other hand, *tries* [12] 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 suitalbe to implement replacement strategies. For instance, a trie permits us to store, on each node, the number of lookups on

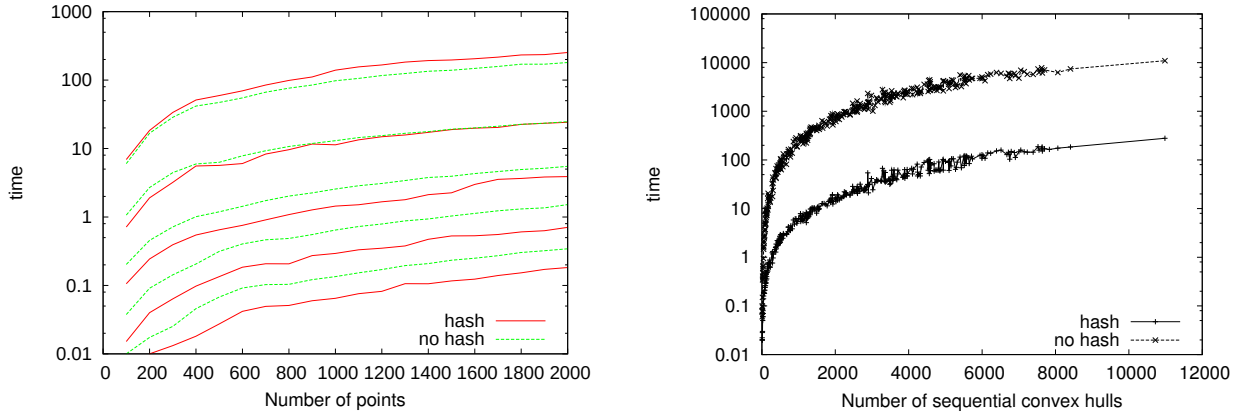

Figure 1: Time for computing a convex hull of a point set $\mathcal{A}$ as function of the number of input points. Graphs correspond to different ambient dimensions of the input points i.e. from bottom to top $dim(\mathcal{A}) = 2, \ldots, 6$ (left). Performance comparisons for hashing versus non-hashing for sequences of 6-dimensional convex hulls; the number of input points range from 10 to 45 (right).

all its sucessors. This information permits to easily prune the less used subtrees at a given depth.

5 Future work

In the theoretical side we expect to reduce the complexity of BB algorithm by using dynamic determinant computations as suggested in Sect. 2.

Our implementation can be used in a general manner, not only inside the CGAL package. The next step is, thus, to develop an interface that permits using the hashed determinants technique transparently (i.e., without knowledge of points' indices). We should also consider implementing other determinant algorithms (such as [15] and [2]) to be competitive in medium dimensions, as well as the algorithm presented in Sect. 2 and tries with replacement strategies as presented in Sect. 4. Finally, we plan to submit our functions as a package to CGAL.

Acknowledgment We thank O. Devillers and S. Hornus for discussions on `triangulation`.

References

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

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

[3] 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.

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

[5] 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*, pages 40–50, 2002.

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

[7] 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.

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

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

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

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

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

[13] 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.

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

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

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