Project number IST-25582

CGL

Computational Geometric Learning

High Dimensional Predicates: Update on Algorithms and Software

STREP

Information Society Technologies

Period covered:	November 1, 2012 – October 31, 2013
Date of preparation:	November 4, 2013
Date of revision:	November 4, 2013
Start date of project:	November 1, 2010
Duration:	3 years
Project coordinator name:	Joachim Giesen (FSU)
Project coordinator organisation:	Friedrich-Schiller-Universität Jena Jena, Germany

High Dimensional Predicates: Update on Algorithms and Software

Ioannis Z. Emiris* Vissarion Fisikopoulos* Luis Peñaranda**

November 4, 2013

Abstract

Determinant predicates are the core procedures in many important geometric algorithms, such as convex hull computations and point location. As the dimension of the computation space grows, a higher percentage of the computation time is consumed by these predicates. We study the sequences of determinants that appear in geometric algorithms. We use dynamic determinant algorithms to speed-up the computation of each predicate by using information from previously computed predicates.

In a previous technical report [EFP12] we have proposed, implemented and experimentally evaluated algorithms for the determinants involved in incremental convex hull and point location algorithms. We have presented an extended version of the hashing determinants method for a sequence of convex hull operations and a CGAL submission of the implementation of this method.

In this report we present an update on this work by proposing a dynamic determinant method for the gift wrapping convex hull algorithm, present experiments on convex hulls on a real practical scenario and an update on our CGAL submission.

Keywords: computational geometry, determinant algorithms, Orientation predicate, convex hull, point location, experimental analysis

1 Introduction

Determinantal predicates are in the core of many important geometric algorithms. Convex hull and regular triangulation algorithms use *orientation* predicates, the Delaunay triangulation algorithms also involve the *in-sphere* predicate. Furthermore, algorithms for exact volume computation of a convex polytope rely on determinantal volume formulas. In general dimension d, the orientation predicate of $d+1$ points is the sign of the determinant of a matrix containing the homogeneous coordinates of the points as columns. In a similar way, the volume determinant formula and in-sphere predicate of $d+1$ and $d+2$ points respectively can be defined. In practice, as the dimension grows, a higher percentage of the computation time is consumed by these core procedures. To this end, we study algorithms and implementations for the computation of the determinants that appear in geometric computations.

Our main observation is that, in a sequence of computations of determinants that appear in geometric algorithms, the computation of one predicate can be accelerated by using information from the computation of previously computed predicates. In this paper, we study orientation

*National and Kapodistrian University of Athens, Department of Informatics and Telecommunications, Athens, Greece. emiris, vfisikop@di.uoa.gr

**Current affiliation: IMPA – Instituto Nacional de Matemática Pura e Aplicada, Rio de Janeiro, Brazil. luisp@impa.br

predicates that appear in convex hull computations. The convex hull problem is probably the most fundamental problem in discrete and computational geometry. In fact, the problems of regular, Delaunay triangulations and Voronoi diagrams reduce to it by computing a convex hull in on dimension higher.

In a previous technical report [EFP12] we have proposed, implemented and experimentally evaluated algorithms with quadratic complexity for the determinants involved in incremental convex hull algorithms and linear complexity for those involved in point location algorithms. Additionally, we had discussed a variant of this algorithm that can perform computations over the integers. This has also been presented in [FP12]. Lastly, we have presented an extended version of the hashing determinants method for a sequence of convex hull operations and a CGAL submission of the implementation of this method.

In this report we present an update on this work by proposing a dynamic determinant method for the gift wrapping convex hull algorithm, present experiments on convex hulls on a real practical scenario and an update on our CGAL submission.

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 [KV05]. However, good asymptotic complexity does not imply good behavior in practice for small and medium dimensions. For instance, LinBox [DGG$^+$02], which implements algorithms with state-of-the-art asymptotic complexity, introduces a significant overhead in medium dimensions, and seems most suitable in very high dimensions (typically > 100). Eigen [GJ$^+$10] and CGAL [CGA] implement decomposition methods (*e.g.*, LU decomposition) of complexity $O(n^3)$ and seem to be suitable for low to medium dimensions.

It is worth mentioning a family of determinant algorithms that use combinatorial approaches. They were introduced by Mahajan and Vinay [MV97], and are based on *clow* (closed ordered walk) sequences. The main interest of these methods is that they avoid divisions. Rote presented a method with complexity $O(n^4)$ [Rot01]. Urbańska conceived a method that uses fast matrix multiplication [CW87] to obtain a complexity $O(n^{3.03})$ [Urb10]. Bird introduced an algorithm with complexity $O(nM(n))$, where $M(n)$ is the complexity of matrix multiplication [Bir11]. The latter algorithm operates with some rows of upper-triangular matrices, what makes the constant hidden in the complexity bound very small.

In addition, there exists a variety of algorithms for determinant sign computation [BEPP99, ABM99]. The problem of computation of several determinants has also been studied. TOPCOM [Ram02], the reference software for computing triangulations of a set of points, efficiently pre-computes all orientation determinants that will be needed in the computation and stores their signs. In [EFKP13], a similar problem is studied in the context of computational algebraic geometry. The computation of orientation predicates is accelerated by maintaining a hash table of computed minors of the determinants. These minors appear many times in the computation. Although, applying that method to the convex hull computation does not lead to a more efficient algorithm.

Our main tools are the Sherman-Morrison formulas [SM50, Bar51]. They relate the inverse of a matrix after a small-rank perturbation to the inverse of the original matrix. Other applications of these formulas include solving the dynamic transitive closure problem in graphs [San04] and studying the effect of new links on Google Page Rank [AL06].

1.1 CGAL submission update

We implemented our method based on CGAL. Our package, called *Lifting_Kernel_d*, introduces a wrapper that can be parametrized by any CGAL d-dimensional kernel (actually, a *model* of

the *Kernel_d* concept).

Our wrapper inherits from the base kernel (the kernel used as parameter of our class) all the geometric objects and some functors. Namely, it inherits all the functors from the base kernel, except `Orientation_d`, which is redefined in order to use our hashing scheme. Moreover, our wrapper adds three new functors:

- `Lifted_orientation_d`, which computes the orientation of $d+2$ points in dimension $d+1$; points are given as d-dimensional kernel points plus one vector of lifting values (this is the lifted analogous to the redefined `Orientation_d` predicate);
- `Scaled_volume_d`, which computes the scaled volume of a polytope defined by $d+1$ points in dimension d (the actual volume multiplied by $d!$);
- `Lifted_scaled_volume_d`, which computes the scaled volume of $d+2$ points in dimension d, taking as parameter the $d+2$ points plus a vector of lifting values (this functor is the lifted analogous of `Scaled_volume_d`).

Technically, the hash table was implemented using the Boost libraries [boo], namely the unordered container library [MSJ08]. We looked for a hashing function, that takes as input a vector of integers and returns an integer, that minimizes collisions. We considered many different hash functions, including some variations of the well-known FNV hash [FNV91]. We obtained the best results with the implementation of Boost Hash [Jam08], which shows fewer collisions than the other tested functions. The function `boost::hash<std::vector<size_t> >` performs a *left fold* [Bir98] on the input vector, using start value 0 and fold function `f(x,y) = x^(y+0x9e3779b9+(x<<6)+(x>>2))`, and has a constant lookup cost in practice.

We clear the hash table when it contains 10^6 minors. This gives a good tradeoff between efficiency and memory consumption.

We submitted our implementation to the CGAL Editorial Board. The first round of reviews gave us very valuable ideas to improve our design and to better adapt to the needs of the Computational Geometry community. The main comments of the reviewers were the following.

- Consider filtering predicates, even if the d-dimensioanl CGAL kernel does not provide filtering, or explain how the wrapper behaves in case of parameterizing with a filtered kernel.
- Polish the interface: the interface described above is the result of the discussion with the reviewers.
- Present the implementation as a wrapper and not as a new kernel.
- Be clear about memory consumption and explain how the table is emptied.

Considering these comments and many useful minor remarks, we plan to resubmit our implementation soon. It should be stressed that the description of our kernel in this section already took into account these issues.

2 Dynamic Determinants and Geometric Algorithms

In the *dynamic determinant problem*, a $d \times d$ matrix A is given. Allowing some preprocessing, we should be able to handle updates of elements of A and return the current value of the determinant. We consider here only non-singular updates, that is, updates that do not make A singular. Let $(A)_i$ denote the i-th column of A, and e_i the vector with 1 in its i-th place and 0 everywhere else.

Algorithm 1: Gift Wrapping Convex Hull $(\mathcal{A})$

Input : pointset $\mathcal{A} \subset \mathbb{R}^d$
Output: convex hull of $\mathcal{A}$

Compute the first facet F;
$\mathcal{F} \leftarrow \{F\}$;
Initialize $\mathcal{R}$ with the ridges $\big(F \setminus \{v\}, v\big)$ for each $v \in F$;

while $\mathcal{R} \neq \emptyset$ **do**
  Let (R, c) be an element of $\mathcal{R}$;
  $F \leftarrow$ New_Face$(\mathcal{A}, R, c)$;
  $\mathcal{F} \leftarrow \mathcal{F} \cup \{F\}$;
  foreach $x \in F$ **do**
    if $\big(F \setminus \{x\}, y\big) \in \mathcal{R}$ *for some* $y \in \mathcal{A}$ **then**
      $\mathcal{R} \leftarrow \mathcal{R} \setminus \big(F \setminus \{x\}, y\big)$
    else
      $\mathcal{R} \leftarrow \mathcal{R} \cup \big(F \setminus \{x\}, x\big)$

return $\mathcal{R}$;

Consider the matrix A', resulting from replacing the i-th column of A by a vector u. The Sherman-Morrison formula [SM50, Bar51] states that $\big(A + wv^T\big)^{-1} = A^{-1} - \frac{(A^{-1}w)(v^T A^{-1})}{1+v^T A^{-1} w}$. An i-th column update of A is performed by substituting $v = e_i$ and $w = u - (A)_i$ in the above formula. Then, we can write A'^{-1} as follows.

$$A'^{-1} = \big(A + (u - (A)_i)e_i^T\big)^{-1} = A^{-1} - \frac{\big(A^{-1}(u - (A)_i)\big)\ (e_i^T A^{-1})}{1 + e_i^T A^{-1}(u - (A)_i)} \tag{1}$$

If A^{-1} is computed, we compute A'^{-1} using Equation 1 in $3d^2+2d+O(1)$ arithmetic operations. Similarly, the matrix determinant lemma [Har97] gives Equation 2 below to compute $\det(A')$ in $2d + O(1)$ arithmetic operations, if $\det(A)$ is computed.

$$\det(A') = \det\big(A + (u - (A)_i)e_i^T\big) = \big(1 + e_i^T A^{-1}(u - (A)_i)\big)\det(A) \tag{2}$$

Equations 1 and 2 lead to the following result.

Proposition 1. [SM50] *The dynamic determinant problem can be solved using $O(d^\omega)$ arithmetic operations for preprocessing and $O(d^2)$ for non-singular one column updates.*

Indeed, this computation can also be performed over a ring. To this end, we use the adjoint of A, denoted by A^{adj}, rather than the inverse. It holds that $A^{\text{adj}} = \det(A)A^{-1}$, thus we obtain the following two equations.

$$A'^{\text{adj}} = \frac{1}{\det(A)}\Big(A^{\text{adj}}\det(A') - \Big(A^{\text{adj}}(u - (A)_i)\Big)\ \Big(e_i^T A^{\text{adj}}\Big)\Big) \tag{3}$$

$$\det(A') = \det(A) + e_i^T A^{\text{adj}}(u - (A)_i) \tag{4}$$

The only division, in Equation 3, is known to be exact, *i.e.*, its remainder is zero. The above computations can be performed in $5d^2 + d + O(1)$ arithmetic operations for Equation 3 and in $2d+O(1)$ for Equation 4. In the sequel, we will call *dyn_inv* the dynamic determinant algorithm which uses Equations 1 and 2, and *dyn_adj* the one which uses Equations 3 and 4.

Algorithm 2: New_Face $(\mathcal{A}, R, c)$

Input : pointset $\mathcal{A} \in \mathbb{R}^d$, ridge R and vertex $c \in \mathcal{A}$ s.t. $R \cup \{c\}$ is a facet
Output: facet which is not $R \cup \{c\}$

Let $u \in \mathcal{A}$ be a point not in $R \cup \{c\}$;
init_det $\leftarrow \det(R, u, c)$;

foreach $t \in \mathcal{A} \setminus \big(R \cup \{u, c\}\big)$ **do**
    `// In the first loop update` c `by` t `in` $\det(R, u, c)$
    new_det $\leftarrow \det(R, u, t)$ updated by new t using Eqs. 1, 2 or Eqs. 3, 4;
    if *new_det* $*$ *init_det* < 0 **then**
        $u \leftarrow t$;
        init_det $\leftarrow$ *new_det*;

return $R \cup \{u\}$;

We discuss here the application of dynamic determinants to the *gift wrapping* convex hull algorithm [CK70]. The gift wrapping algorithm (Algorithm 1) computes first a facet of the convex hull of a pointset $\mathcal{A} \in \mathbb{R}^d$. This can be done by solving a linear program with d unknowns and n constraints. It continues by computing the new facet for every ridge whose other facet has already been computed. Note that every ridge is adjacent to exactly two facets and this is the high-dimensional analog of edges adjacent to two facets in $\mathbb{R}^3$.

The computation of the new facet (Algorithm 2) performs $O(n)$ orientation tests. First, it computes $\det(R, u, c)$ and $A^{\text{adj}}_{R,u,c}$ where (R, c) is the existing facet and u a new point not belonging to that facet. Then, it computes $\det(R, u, t)$ for a new point $t \notin R \cup \{c, u\}$ which is an update of c by t in $A_{R,u,c}$. Thus it can be computed in $O(d^2)$ arithmetic operations using Equations 1,2 or Equations 3,4. Similarly, in the next step a new point t' will be tested. The orientation test can be computed as updates of either u or t in $A_{R,u,t}$ by t'. If t, c were in different half-spaces with respect to (R, u) in the previous step then u should be updated by t', otherwise t should be updated.

The complexity of solving the linear program that computes the first facet is $O(d^{3.5}L^2 \log L \log\log L)$ by using Karmarkar interior-point method [Kar84]. Here, $L = \log(1 + D_{\max}) + \log(1 + \alpha)$, $D_{\max} = \max\{|\det(M)| : M \text{ is a square submatrix of the matrix constructed by the } n \text{ points}\}$ and α is the largest point coordinate. Algorithm 2 is called h times by Algorithm 1, where h is the total number of vertices in $\text{conv}(\mathcal{A})$. It is clear that Algorithm 2 runs in $O(d^3 + nd^2)$ and $O(d^2)$ space. The size of the data structure for ridges $\mathcal{R}$ is $O(n^{\lfloor d/2 \rfloor})$ [Zie95, §8.4] but we can perform searching and inserting in a time logarithmic to this size, *i.e.*, $O(d \log n)$. We conclude that the complexity of Algorithm 1 is $O(d^{3.5}L^2 \log L \log\log L + hd^3 + hnd^2)$, which reduces to $O(hnd^2)$ when $d^{1.5}L^2 \in o(n)$.

Lemma 2. *Given a d-dimensional pointset all, except h, orientation predicates of the gift wrapping convex hull algorithm can be computed in $O(d^2)$ time and $O(d^2)$ space, where h is the number of the vertices of the convex hull.*

The following result improves the complexity of gift wrapping algorithm, which is $O(hnd^3)$. Note also that the space complexity of gift wrapping algorithm is bounded by $O(n^{\lfloor d/2 \rfloor})$ [Zie95, §8.4], which shows that Algorithm 1 does not have polynomial complexity.

Corollary 3. *Given n d-dimensional points, the complexity of gift wrapping algorithm is $O(hnd^2)$, where h is the number of the vertices of the convex hull and we assume that $d^{1.5}L^2 \in o(n)$.*

3 Implementation and Experimental Analysis

We propose the *hashed dynamic determinants* scheme and implement it in C++. The scheme consists of efficient implementations of algorithms *dyn_inv* and *dyn_adj* and a hash table, which stores intermediate results (matrices and determinants). The design of our implementation is modular, that is, it can be used on top of either algebraic software providing dynamic determinant algorithm implementations or geometric software providing geometric predicates (such as orientation). We desing and perform experiments with both algebraic and geometric software to quantify the efficiency of our method.

The hash table has been implemented using the Boost libraries [boo]. To reduce memory consumption and speed-up look-up time, we sort the lists of indices that form the hash keys. We also use the *GNU Multiple Precision arithmetic library* (GMP), the current standard for multiple-precision arithmetic, which provides integer and rational types `mpz_t` and `mpq_t`, respectively.

For the experimental analysis of the behavior of dynamic determinants used in convex hull algorithms, we experiment with the incremental convex hull algorithm `triangulation` [BDH09] which implements [CMS93]. We propose and implement a variant of `triangulation`, which we will call `hdch`, implementing the hashed dynamic determinants scheme for dimensions ≥ 6. In the case of integer coefficients, we test `hdch` using `mpq_t` (`hdch_q`) or `mpz_t` (`hdch_z`).

The code is publicly available from

`http://hdch.sourceforge.net`.

The data used in the experiments are also available in the above web-page and thus all the experimental results can be reproduced.

All experiments ran on an Intel Core i5-2400 3.1GHz, with 6MB L2 cache and 8GB RAM, running 64-bit Debian GNU/Linux.

In this report we carry out experiments using as input the vertices of resultant polytopes which have integral coefficients (Section 1). The results in Table 1 emphasize the utilization of the hashed dynamic determinants scheme when working with real data.

$\lvert\mathcal{A}\rvert$	d	time(sec)			volume
		`hdch_q`	`hdch_z`	`triang`	
80	6	0.54	0.27	0.66	368986.7
100	6	0.69	0.33	0.87	108096.3
110	6	1.20	0.52	1.40	1456226058.5
125	6	1.28	0.61	1.66	66137.3
376	7	17.07	7.80	24.41	1713149926.2
414	7	23.02	10.91	32.54	82132445.9
500	7	29.40	13.05	41.22	2593047991.6
528	7	38.22	17.96	54.91	33727790.7

Table 1: Comparison of `hdch_q`, `hdch_z` and `triangulation` computing resultant polytopes.

4 Future Work

A future improvement in the memory consumption of our method could be the exploitation of hybrid memory management techniques as discussed in Section 3. Additionally, the utilization of dynamic determinants in gift-wrapping convex hull algorithms (Algorithm 1) could be helpful

towards this direction. Finally, studying the behavior of our scheme using filtered computations, could lead to even more efficient implementations.

References

[ABM99] John Abbott, Manuel Bronstein, and Thom Mulders. Fast deterministic computation of determinants of dense matrices. In *ISSAC*, pages 197–203, 1999.

[AL06] Konstantin Avrachenkov and Nelly Litvak. The effect of new links on Google PageRank. *Stoch. Models*, 22:2006, 2006.

[Bar51] Maurice S. Bartlett. An inverse matrix adjustment arising in discriminant analysis. *The Annals of Mathematical Statistics*, 22(1):107–111, 1951.

[BDH09] J.-D. Boissonnat, O. Devillers, and S. Hornus. Incremental construction of the Delaunay triangulation and the Delaunay graph in medium dimension. In *SoCG*, pages 208–216, 2009.

[BEPP99] H. Brönnimann, I.Z. Emiris, V. Pan, and S. Pion. Sign determination in Residue Number Systems. *Theor. Comp. Science*, 210(1):173–197, 1999.

[Bir98] Richard S. Bird. *Introduction to Functional Programming Using Haskell*. Prentice-Hall, 1998.

[Bir11] Richard S. Bird. A simple division-free algorithm for computing determinants. *Inf. Process. Lett.*, 111:1072–1074, November 2011.

[boo] Boost: peer reviewed C++ libraries.

[CGA] CGAL: Computational geometry algorithms library.

[CK70] Donald R. Chand and Sham S. Kapur. An algorithm for convex polytopes. *J. ACM*, 17(1):78–86, January 1970.

[CMS93] K.L. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. *Comput. Geom.: Theory & Appl.*, 3:185–121, 1993.

[CW87] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 1–6, New York, NY, USA, 1987. ACM.

[DGG+02] 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.

[EFKP13] Ioannis Z. Emiris, Vissarion Fisikopoulos, Christos Konaxis, and Luis Peñaranda. An oracle-based, output sensitive algorithm for projections of resultant polytopes. *International Journal of Computational Geometry and Applications*, 2013. (to appear).

[EFP12] I.Z. Emiris, V. Fisikopoulos, and L. Peñaranda. High dimensional predicates: Algorithms and software. Technical Report CGL-TR-27, NKUA, 2012.

[FNV91] G. Fowler, L.C. Noll, and P. Vo. FNV hash. www.isthe.com/chongo/tech/comp/fnv/, 1991.

[FP12] Vissarion Fisikopoulos and Luis Pearanda. Faster geometric algorithms via dynamic determinant computation. In Leah Epstein and Paolo Ferragina, editors, *Algorithms ESA 2012*, volume 7501 of *Lecture Notes in Computer Science*, pages 443–454. Springer Berlin Heidelberg, 2012.

[GJ+10] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3, 2010.

[Har97] David A. Harville. *Matrix algebra from a statistician's perspective*. Springer-Verlag, New York, 1997.

[Jam08] D. James. Boost functional library. www.boost.org/ libs/functional/hash, 2008.

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, December 1984.

[KV05] Erich Kaltofen and Gilles Villard. On the complexity of computing determinants. *Computational Complexity*, 13:91–130, 2005.

[MSJ08] J.B. Maitin Shepard and D. James. Boost unordered library. www.boost.org/libs/unordered, 2008.

[MV97] Meena Mahajan and V. Vinay. A combinatorial algorithm for the determinant. In *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, SODA '97, pages 730–738, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.

[Ram02] 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.

[Rot01] Günter Rote. Division-free algorithms for the determinant and the Pfaffian: algebraic and combinatorial approaches. In *Comp. Disc. Math.*, pages 119–135, 2001.

[San04] Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse. In *Proc. IEEE Symp. on Found. Comp. Sci.*, pages 509–517, 2004.

[SM50] Jack Sherman and Winifred J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, 1950.

[Urb10] Anna Urbańska. Faster combinatorial algorithms for determinant and Pfaffian. *Algorithmica*, 56:35–50, 2010.

[Zie95] G.M. Ziegler. *Lectures on Polytopes*. Springer, 1995.