

# Formalizing Convex Hulls Algorithms

David Pichardie

ENS Cachan-Bretagne

`David.Pichardie@eleves.bretagne.ens-cachan.fr`

Yves Bertot

INRIA Sophia Antipolis

`Yves.Bertot@inria.fr`

No Institute Given

## 1 Introduction

**Abstract.** We study the development of formally proved algorithms for computational geometry. The result of this work is a formal description of the basic principles that make convex hull algorithms work and two programs that implement convex hull computation and have been automatically obtained from formally verified mathematical proofs. A special attention has been given to handling degenerated cases that are often overlooked by conventional algorithm presentations.

Algorithms to compute the convex hull of a collection of points in two or three dimensions abound. The complexity of this problem is known to be approximately the same as for a sorting algorithm in the worst case, but the average complexity depends very much on the distribution of data: some algorithms actually have a complexity of the form  $n \times h$ , where  $n$  is the total number of points and  $h$  is the number of points on the convex hull. These algorithms will be more efficient than  $n \log(n)$  when few points are on the convex hull. This makes it useful to have several algorithms around.

We have studied two algorithms. The first is an incremental algorithm, where new points are progressively added to the input data. If the new point is outside the convex hull, then it is necessary to add two new edges to the convex hull and remove some others. The second algorithm is known as the package-wrapping algorithm: it follows the intuition of tying a band around the collection of points. At the  $n^{\text{th}}$  step, the band is already in contact with a few points and the algorithm proceeds by computing what the next point is going to be: if with the band turning clockwise around the points, the next point in the hull is going to be the leftmost remaining point.

All these algorithms rely on an orientation predicate that expresses whether the points of a triangle are enumerated clockwise or counter-clockwise. This orientation predicate is easily computed using the coordinates of the points, but it is meaningless when the points are aligned. Usual presentation of convex hulls algorithms assume that three points in the input data are never aligned.

The structure of convex hull algorithms rely on the fact that the orientation predicate satisfies some properties: for instance the triangles built with four points have to be oriented in a consistent manner. Knuth [8] describes the minimal properties of the orientation predicates and calls them *axioms*.

Knuth’s approach of axioms for convex hulls has the nice property of separating concerns about the properties of arithmetic expressions containing point coordinates and the control structure of algorithms. It has proved a good organization principle for our description and proof development.

Thus, our work contains distinct parts. The first part is about the axioms, and showing that they hold for an implementation predicate. This part involves some numerical computation. The second part describes the main structure of the algorithms based on the axioms, with numeric computation completely avoided. In the last part, we revisit the algorithms to make them robust with respect to degenerated data.

All our proofs have been mechanically checked using the Coq system [2].

### 1.1 Related work

Automatic theorem proving and geometry have been in contact for some time. The work in this domain that we are better aware of is that of Chou [5]. In this development, theorems about basic geometric figures like straight lines, triangles and circles are proved automatically, but there is no result about computational geometry *per se*, since algorithms are not the object of the study.

Puitg and Dufourd [9], used the Coq proof system to describe notions of planarity. Work on finding minimal axioms to describe geometric concepts has also been done in constructive proof theory by Jan von Plato [11] and formalized in Coq by Gilles Kahn [7].

Last, we should refer to all the work done on computational geometry, but this domain is far too large to be cited entirely, we can only refer to the books we have used as reference, [6] and [3], and [10].

## 2 Knuth’s “axioms” for convex hulls

In what follows, we assume that points are taken from a set  $P$  and we describe the orientation predicate as a predicate over three points  $p, q, r$  in  $P$ , which we denote  $\widehat{pqr}$ .

Knuth’s axioms describe the various ways in which triangles can be looked at and arranged on the plane. The first axiom expresses that when enumerating the points of a triangle, one may start with any of them:

**Axiom 1**  $\forall p, q, r. \widehat{pqr} \Rightarrow \widehat{qrp}$ .

As a corollary of this axiom one also has  $\widehat{pqr} \Rightarrow \widehat{rqp}$ . This axiom is “cyclic” it can be repeated indefinitely on the same data. For this reason, it impedes automatic proof search procedures.

The second axiom expresses that if a triangle is oriented counter-clockwise, then the same triangle with two points transposed is in fact oriented clockwise.

**Axiom 2**  $\forall p, q, r. \widehat{pqr} \Rightarrow \neg \widehat{prq}$ .

An immediate consequence of the first two axioms is that three points are oriented only if they are pairwise distinct:

$$\forall p, q, r. \widehat{pqr} \Rightarrow p \neq q \wedge q \neq r \wedge r \neq p$$

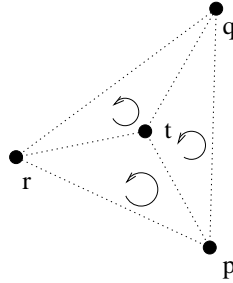
Many of the lemmas appearing in our development therefore rely on the assumptions that the points being considered are pairwise distinct. We also have a predicate on lists of points expressing that they contain pairwise distinct points.

The third axiom expresses that a triangle is either oriented clockwise or counter-clockwise.

**Axiom 3**  $\forall p, q, r. p \neq q \Rightarrow q \neq r \Rightarrow p \neq r \Rightarrow \widehat{pqr} \vee \widehat{prq}$ .

This affirmation voluntarily overlooks the fact that points may be aligned. This is a tradition of computational geometry that algorithms can be studied without taking care of what are called *degenerate* cases. We will have to study this more carefully later.

The fourth axiom expresses that the four triangles obtained with four arbitrary points may not be oriented in an arbitrary manner: there has to be some sort of consistency, easily understood by observing figure 1.

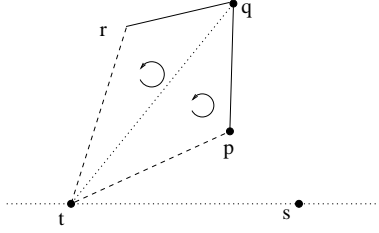


**Fig. 1.** Axiom 4: Consistent orientation for four triangles.

**Axiom 4**  $\forall p, q, r, t. \widehat{tqr} \wedge \widehat{ptr} \wedge \widehat{pqt} \Rightarrow \widehat{pqr}$ .

The fifth axiom expresses that the orientation predicate may be used to sort points in some way, with a notion that is similar to transitivity. Still one has to be careful to avoid going in circles, this is the reason why this axiom uses 5 points, see figure 2. Clever algorithms use this property of like-transitivity to avoid looking at all points.

**Axiom 5**  $\forall p, q, r, s, t. \widehat{tsp} \wedge \widehat{tsq} \wedge \widehat{tsr} \wedge \widehat{tpq} \wedge \widehat{tqr} \Rightarrow \widehat{tpr}$



**Fig. 2.** Axiom 5: sorting points in a half plane.

The first three orientation properties express that the three points  $p$ ,  $q$ , and  $r$  are in the same half plane. The other three orientation predicates express transitivity: from the point of view of  $t$ , if  $q$  is left of  $p$  and  $r$  is left of  $q$ , then  $r$  is left of  $p$ .

There is a variant to this axiom, that can actually be proved using axioms 1, 2, 3, and 5, and which we will refer to by the name *axiom 5'*.

**Axiom 5'**  $\forall p, q, r, s, t. \widehat{stp} \wedge \widehat{stq} \wedge \widehat{str} \wedge \widehat{tpq} \wedge \widehat{tqr} \Rightarrow \widehat{tpr}$

Knuth shows that these axioms are independent. For the case of general positions, these axioms are enough, but if we want our algorithms to be robust for possibly aligned points in the input, there are two possibilities. The first one is to introduce notions that are meaningful for triplets of points that are aligned. In this case, we also use segments and the property that a point belongs to a segment given by its two extremities. This adds up to 9 more axioms, which we do not describe here for the moment. The second solution is to find a way to extend orientation to aligned points in a way that still satisfies the axioms. Perturbation methods as described in [1] provide a satisfactory approach to this.

## 2.1 Proving the axioms

When points are given as elements of  $\mathbb{R}^2$ , the orientation predicate is given by observing the sign of a determinant:

$$\widehat{pqr} \equiv \begin{vmatrix} x_p & y_p & 1 \\ x_q & y_q & 1 \\ x_r & y_r & 1 \end{vmatrix} > 0.$$

In the following we will write  $|pqr|$  for this determinant. Knuth's axioms are not proper axioms anymore, they become simple consequences of the properties of addition and multiplication of real numbers.

The proof of Axioms 1 and 2 is given by two equations:

$$|pqr| = |qrp|, \quad |pqr| = |prq|.$$

From a mechanical proof point of view, the proofs of these equalities is done painlessly by the usual confluent rewriting system for ring structures, as implemented by the `Ring` tactic in Coq. Axiom 3 simply does not hold in general.

When points are aligned, the determinant is null. A common solution to this problem is to consider only data that are in *general position*, that is, assume that no three points given in the problem are ever aligned. We will first follow this practice to get a better idea of the algorithms. In a later section, we will see different solutions to the problem of considering degenerated cases.

In man-made proofs, Axiom 4 relies on an equality obtained from a 4-by-4 determinant that is obviously null, since it has 2 identical columns.

$$\begin{vmatrix} x_p & y_p & 1 & 1 \\ x_q & y_q & 1 & 1 \\ x_r & y_r & 1 & 1 \\ x_t & y_t & 1 & 1 \end{vmatrix} = |pqr| - |tqr| - |ptr| - |pqt| = 0.$$

Proving the right-hand side equality is also a simple matter of equality in a ring structure. Now, if  $|tqr|$ ,  $|ptr|$ ,  $|pqt|$  are all strictly positive, then  $|pqr|$  is.

Verifying Axiom 5 also relies on an equation, known as Cramer's equation, which has the following form:

$$|stq||tpr| = |tqr||stp| + |tpq||str|$$

Here again, automatically proving this equation is a simple matter of ring based computations.

The conclusion of this section on the orientation predicate is twofold. First numeric computation can be done in a ring rather than in a complete field. This is important as one of the reasons for problems in geometric algorithms is the loss of precision in floating point arithmetic. For convex hulls, floating points are not required. The points may be given by integer coordinates in a small grid. If floating points are used, one may notice that, since the determinant basically is a polynomial of degree two in the coordinates of the three points being considered, it is only necessary to compute the orientation predicate with a precision that is double the precision of the input data.

## 2.2 The specification

What is the convex hull of a set of points  $S$ ? One definition is to describe it as a minimal subset  $S' \subset S$ , such that all points in  $S$  can be described as positive linear combinations of points in  $S'$ .

An alternative description of the convex hull uses oriented edges. A possibly open line  $S'$  *encompasses* a set of points  $S$  if for any point  $p$  in  $S$  and any two consecutive points  $t_i t_{i+1}$  in  $S'$ ,  $\widehat{t_i t_{i+1} p}$  or  $p \in \{t_i t_{i+1}\}$  holds. We consider that a list  $l$  is a convex hull of  $S$  if  $l$  is included in  $S$ ,  $l$  is non-empty if  $S$  is not,  $l$  contains no duplicates, and if  $t$  is the last element of  $l$ , then  $t \cdot l$ , formed by adding  $t$  at the head of  $l$ , encompasses  $S$ . In other terms, the convex hull is defined as a minimal intersection of half-planes.

When a data set is the empty set, a singleton, or a pair, then the convex hull is simply the list enumerating the set's elements.

Knuth chooses the second definition in his monograph. We will also, for several reasons. First, lists are a natural data-structure in the functional programming language we use in our proofs. Second, this definition also relies on the orientation predicate, thus making it possible to forget everything about numerical computation in the main structure of algorithm. Third, most algorithms to compute convex hulls naturally rely on the fact that the intermediary data they construct already is a list of points included in  $S$ .

The three parts of the specification are given using inductive definitions on lists (actually we also consider that the input data is given as a list of points).

Still, we have made an effort to establish a bridge between the two definitions. We have mechanically verified a proof that if  $S'$  is a convex hull of  $S$  and  $p$  is an arbitrary point in  $S$ , then there exists three points  $t_i, t_j$ , and  $t_k \in S'$  such that  $p$  is inside the triangle formed by  $t_i, t_j$  and  $t_k$ . This is expressed by the following three orientation predicates:

$$\widehat{t_i t_j p} \quad \widehat{t_j t_k p} \quad \widehat{t_k t_i p}$$

The proof by induction on the length of a convex polygon  $S'$  encompassing  $p$  works as follows. First,  $S'$  cannot be empty, since it encompasses  $p$ . If  $S'$  is a singleton or a pair, the degenerate  $p, p, p$  will do. If the convex polygon contains at least three points, first check whether  $p$  is equal to  $t_1, t_2$ , or  $t_3$ , then compute  $\widehat{t_1 t_3 p}$ . If any of these conditions holds, then take  $t_1 t_2 t_3$ . If none of these conditions holds, then  $p$  is actually inside the convex polygon defined by  $t_1 t_3 \dots$ . You can use an induction hypothesis on this smaller convex polygon to reach the result.

Once you have the three points encompassing  $p$  it is possible to show that the coordinates of  $p$  are a positive linear combination of the coordinates of the three points. This is simply because the coordinates of  $p$  verify the following equation:

$$p \times |t_i t_j t_k| = t_i \times |t_j t_k p| + t_j \times |t_k t_i p| + t_k \times |t_i t_j p|$$

This equation is easily verified using the ring decision procedure.

If  $\widehat{t_i t_j p}$ ,  $\widehat{t_j t_k p}$ , and  $\widehat{t_k t_i p}$  hold, then the three determinants on the right-hand side are positive and by Axiom 4 the determinant on the left-hand side also is. Dividing by this determinant is legitimate and  $p$  really is obtained by a positive linear combination of the  $t$ 's. The cases where the convex hull contains less than three points are also easy to prove. Notice that this proof requires that we work in a field.

### 2.3 Finding initial data

Some algorithms require that one should be able to find one or two points that are already present in the convex hull to start with. The package-wrapping algorithm is among them. The usual technique is to look for the minimal point using a lexical ordering on the coordinates.

If  $t$  is the minimal point for the lexical ordering on coordinates, and if  $p, q$ , and  $r$  are points of a triangle encompassing  $t$ , where the first coordinates of  $p, q$ , and  $r$  are such  $p_1 \geq r_1$  then it is necessary that  $r_1 \geq q_1$ . Using this result again,

we have  $p_1 \geq r_1 \Rightarrow r_1 \geq p_1$ . Thus all three first coordinates must be equal. But if they are, the points are aligned and cannot constitute a proper triangle.

Once we have the point  $t_2$  whose coordinates are minimal for the lexicographic order, we can find its left neighbour by choosing the only point that has the same second coordinate as  $t_2$ . If no such point exists, take the point that is maximal for the order  $\prec$  defined by:

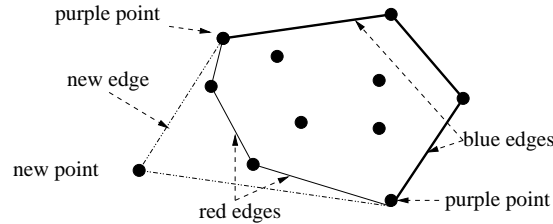
$$p \prec q \Leftrightarrow \widehat{tpq}$$

That  $\prec$  is transitive is ensured by Axiom 5', using the point  $(t_x, t_y + 1)$  as point  $s$ .

### 3 Proving algorithms

In this section we review the descriptions and proofs for the abstract setting: positions are general (no three points are aligned) and all numerical computation is hidden behind the orientation predicate and the function producing initial points.

#### 3.1 The incremental algorithm



**Fig. 3.** Red and blue edges in the incremental algorithm

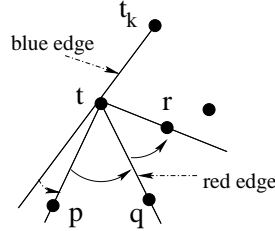
The incremental algorithm works by taking the points of the input data one by one and constructing the convex hull of all the points seen so far. At each step, either the new point already lies inside the current convex hull and one can directly proceed to the next, or some edges need to be removed and two edges need to be added. All tests rely on the orientation predicate.

The edges from the previous convex hull are sorted into two categories, called *blue* and *red* edges (see figure 3). Blue edges represent edges, for which the new point lies on good side. In our description we also say that an edge is blue if the new point is one of the extremities. Red edges are the other. When all edges are blue, the point is inside. All edges cannot be red at the same time and red edges are contiguous.

When looking for red edges there may be four cases:

1. no red edges (including no edges at all),
2. the edge between the last point of the list and the first point of the list (the closing edge) is red, but the last edge of the list is not.
3. the closing edge is red and the last edge too,
4. the closing edge is blue.

In the first case, nothing needs to be done. In the second case, the list of edges can be viewed as the concatenation of two lists,  $l_r$  and  $l_b$ , such that  $l_r$  contains only red edges  $l_b$  contains only blue edges, and the edge between the last element in  $l_r$  and the first element in  $l_b$  is red. This relies on the property that the list describing the convex hull has no duplicates. In this case, the result is the list  $p \cdot l_b$ . To prove this, we need to show that  $p$  is not already in the list (easy), that the new point  $p$  is in the input data (easy), that  $p$  is left of all the edges in the convex hull (easy, since the edges that are kept are all blue, and the two new edges have  $p$  at the extremity), and that all the data considered so far is left of the edges  $tp$  and  $pt'$ , where  $t$  and  $t'$  are the purple points, the last and first elements of  $l_b$ , respectively. The point  $t$  is different from  $t'$  if there is a red edge. Let us detail part of the proof for the other cases (see figure 4).



**Fig. 4.** Using axiom 5' to justify adding an edge.

Since  $t$  and  $t'$  are distinct, there exists a  $t_k$  distinct from  $t$  that is last-but-one in  $l_b$ . Let  $q$  be the other extremity of the first red edge, and let  $r$  be an arbitrary point among those that have already been processed. By construction, we have  $\widehat{t_k t r}$ ,  $\widehat{t_k t q}$ , and  $\widehat{t q r}$ . Because  $tq$  is red we don't have  $\widehat{t q p}$ . By Axiom 3, this gives  $\widehat{t p q}$ . Because  $t_k t$  is blue, we have  $\widehat{t_k t p}$ . All hypotheses are present to use Axiom 5' and conclude that  $\widehat{t p r}$  holds. The proof for  $\widehat{p t' r}$  has the same structure, but uses Axiom 5.

The two other cases are solved by rotating the convex hull to return to the second case. Rotating a convex hull is simply decomposing this convex hull as the concatenation of two lists. In the following, we will denote concatenation of two lists  $l_1$  and  $l_2$  as  $l_1 \bullet l_2$ . Since concatenation is a simple structural recursive function (recursive over the first argument), it is a simple matter of structural recursion over  $l_1$  to show that rotation preserves inclusion in another list, that it preserves the absence of duplicate elements, and that it preserves the data encompassed. Thus, if  $l_2 \bullet l_1$  is a convex hull of  $l$ ,  $l_1 \bullet l_2$  also is.

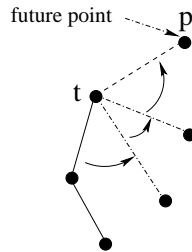
For the third case, one traverses down the list representing the convex hull until one finds the first point  $t_j$  such that  $t_{j-1}t_j$  is blue and  $t_jt_{j+1}$  is red. In that case the rotation is given by  $l_2 = t_j \cdots t$  and  $l_1 = t_1 \cdots t_j$ . If  $t_i$  is the first point in  $l_1$  such that the edge  $t_{i-1}t_i$  is red and the edge  $t_it_{i+1}$  is blue, then one has:  $l_r = t_j \cdots t \bullet (t_1 \cdots t_{i-1})$  and  $l_b = t_{i+1}t_j$ .

The fourth case is symmetric to the third one. One also needs to find the two *purple* points that are boundaries between blue and red edges but they are in reverse order.

The main algorithm repeats this computation for each point. It is programmed as a simple structural recursion over the list of inputs.

### 3.2 The package-wrapping algorithm

The package-wrapping algorithm relies on the possibility to find an initial edge that is known to be part of the convex hull, then it proceeds by constructing an open encompassing list  $l$  of pairwise distinct points, included in the data, finding the right neighbour of the first element in the list at each step. At each step one compares the right neighbour with the last element of the list. When the two points are equal, the algorithm terminates without adding the current list. When the two points are different, the right neighbour is added to the list and recursion restarts (see figure 5). During the recursion the list always has at least



**Fig. 5.** Intermediary iteration for the package-wrapping algorithm.

two elements, let  $t$  be the head and  $s$  be the second element. Finding  $t$ 's right neighbour is done by finding the greatest element for the relation  $\prec$  defined by

$$p \prec q \Leftrightarrow \widehat{tpq}.$$

This relation is not reflexive, but its reflexive closure is a total order, thanks to Axioms 2, 3, and 5. Finding the least element of a list for an arbitrary order is an easy piece of functional programming, and it is also easily proved correct. If  $p$  is this greatest element,  $\widehat{tqp}$  holds for every point in the input data, and by Axiom 1  $\widehat{ptq}$  also holds. Thus the input data is left of the edge  $pt$  and the list  $p \cdot l$  also encompasses the input data.

A key ingredient to proving the termination of the algorithm is to ensure that the list being constructed never contains duplicate elements and is included in the input data. The hardest part is to show that checking that  $p$  is distinct from the last element of  $l$  is enough to ensure that  $p$  does not belong to  $l$ . In fact, if  $t$  is the head of the list,  $t'$  is the last element and  $q$  is another element of the list, one needs to show that  $\widehat{tqt'}$  holds. This is done by recursion over the length of list  $l$  but there are many intricacies with base cases, because the proof relies on Axiom 5, which needs 5 distinct points. Let us have a look at this proof.

First assume  $q$  is the second element of the list,  $\widehat{tqt'}$  comes easily from the fact that the list encompasses the input data and  $t'$  is in the input data. Then if  $q$  is not the second element of the list, let  $t_1$  be this second element. We do the proof by structural recursion over the list  $q \cdots t'$ . If this list is reduced to 0 or 1 element, this is contradictory with our hypotheses, if this list has two elements, then  $qt'$  is an edge from the list, and  $\widehat{qt't}$  holds because  $t$  is in the input data. The result then comes from Axiom 1. If the list  $q \cdots t'$  has more than two elements, let  $q'$  be the second element, one has  $\widehat{tq't'}$  by induction hypothesis,  $\widehat{tqq'}$  because  $qq'$  encompasses the input data and thanks to Axiom 1. Moreover, the properties  $\widehat{tt_1q}$ ,  $\widehat{tt_1q'}$ , and  $\widehat{tt_1t'}$  hold, because  $q$ ,  $q'$ , and  $t'$  are in the input data. This is enough to use axiom 5 and conclude.

Now, the package-wrapping algorithm is not structural recursive. The termination argument is that the size of the list  $l$  is bounded by the size of the input data, because all its elements are pairwise distinct, and that this size increases at each step. For this reason, the function is defined using Coq's solution for defining well-founded recursive functions. This means that we have to exhibit a set containing the principal argument of the recursive function and a well-founded order on this set and show that recursive calls are done only on smaller terms.

The set  $L_S$  we exhibit is the set of lists that are included in the input data  $S$  and contain no duplicates. In Coq's type theory, this set is described as a type using dependently typed inductive types. We show that the length of a list in this set is always smaller than the size  $N$  of the input data  $S$ . We define the order  $<_S$  to be :

$$l <_S l' \Leftrightarrow N - \text{length}(l) < N - \text{length}(l').$$

Proving that this order is well-founded is a simple application of theorems about composing well-founded orders with functions provided in Coq's libraries. In other theorem provers where the termination of recursive functions is ensured by exhibiting measure functions, the measure function to exhibit is the following one:

$$f : l \mapsto N - \text{length}(l).$$

At each recursive step, the main argument is  $l$  and the argument for the next recursive call is  $p \cdot l$  such that  $p \cdot l \in L_S$ , therefore one has  $\text{length}(p) < \text{length}(p \cdot l) \leq N$  and  $f(p \cdot l) < f(l)$ .

## 4 Degenerated cases

If we want to allow for aligned points in the input data, we need to review our technique. One solution is to change Axiom 3, since it is the one that assumes that a triple of points can always be viewed as a triangle. If we do that, any decision taken with respect to orientation in the algorithms must be reviewed and replaced by a new decision taking extra cases into account. The structure of the algorithm remains, but all the details of its control-flow are deformed by this constraint.

A second solution is to change the orientation predicate so that it satisfies all five axioms even in the presence of aligned triples. This solution somehow imposes that one changes the definition of convex hulls, since our definition relies on the orientation predicate. We shall see that both solutions actually require that we relax our definition of a convex hull.

### 4.1 Segments

When three distinct points are aligned, one of them lies between the two other. We introduce a new predicate, written  $\lceil pqr \rceil$  to express that  $q$  occurs in side the segment  $pr$ . Axiom 3 is changed to obtain the following formulation:

**Axiom 3**  $\forall p, q, r. \quad p = q \vee q = r \vee p = r \vee \widehat{pqr} \vee \widehat{prq} \vee \lceil pqr \rceil \vee \lceil qrp \rceil \vee \lceil rpq \rceil.$

In Knuth's work, Axioms 1-5 express a form of internal consistency for the orientation predicate. We need to have the same kind of internal consistency for the segment predicate and between segments and oriented triangles. The first two axioms describe internal consistency for three aligned points.

**Axiom 6**  $\forall p, q, r. \quad \lceil pqr \rceil \Rightarrow \lceil rqp \rceil.$

**Axiom 7**  $\forall p, q, r. \quad \lceil pqr \rceil \Rightarrow \neg \lceil qpr \rceil.$

The next five axioms describe consistency between segments and triangles.

**Axiom 8**  $\forall p, q, r. \quad \lceil pqr \rceil \Rightarrow \widehat{pqr}.$

**Axiom 9**  $\forall p, q, r, t. \quad \widehat{pqr} \wedge \lceil ptq \rceil \Rightarrow \widehat{ptr}.$

**Axiom 10**  $\forall p, q, r, t. \quad \widehat{pqr} \wedge \lceil pqt \rceil \Rightarrow \widehat{ptr}.$

**Axiom 11**  $\forall p, q, r, t. \quad \widehat{pqr} \wedge \lceil ptq \rceil \Rightarrow \widehat{tqr}.$

**Axiom 12**  $\forall p, q, r, t. \quad \widehat{pqr} \wedge \lceil tpq \rceil \Rightarrow \widehat{tqr}.$

The last two axioms describe internal consistency of segments for four aligned points.

**Axiom 13**  $\forall p, q, r, t. \quad \lceil qrt \rceil \wedge \lceil pqt \rceil \Rightarrow \lceil prt \rceil.$

**Axiom 14**  $\forall p, q, r, t. \quad \lceil qrt \rceil \wedge \lceil pqr \rceil \Rightarrow \lceil prt \rceil.$

With these new notions, we change the specification of a convex hull for a set  $S$  to be a list of points  $t_1 \dots t_k$  such that for every  $p$  in  $S$  and every  $t_i, t_j$  such that  $t_j = t_{i+1}$  or  $t_j = t_1$  if  $t_i = t_k$  one has  $\widehat{t_i t_j p} \vee \lceil t_i t_j p \rceil.$

## 4.2 The working horse predicate

In the new setting we use a new predicate that combines orientation and segment membership. For the rest of the paper, we will denote this predicate  $\overline{pqr}$ , with the following meaning:

$$\overline{pqr} \Leftrightarrow \widehat{pqr} \vee r \in \{p, q\} \vee ]prq[.$$

Generalizing our algorithms to the degenerated cases works by replacing the orientation predicate by the new predicate wherever it occurs and adding a few equality tests to take care of degeneracies. What matters really is that equivalents of Axiom 5 still hold:

$$\forall p, q, r, s, t. \quad \overline{tsp} \wedge \overline{tsq} \wedge \overline{tsr} \wedge \neg \overline{qtp} \wedge \overline{qtr} \Rightarrow \overline{ptr}$$

$$\forall p, q, r, s, t. \quad \overline{stp} \wedge \overline{stq} \wedge \overline{str} \wedge \neg \overline{tqp} \wedge \overline{tqr} \Rightarrow \overline{tpr}.$$

Thus, if  $t$  is known to be on the convex hull, we keep the property that the order  $\prec'$  can be used to find minimal and maximal elements, when this order is defined as follows:

$$p \prec' q \Leftrightarrow \overline{tpq}.$$

## 4.3 Perturbation

An alternative approach is to change the orientation predicate so that Axiom 3 always holds. This option seems awkward: if we change the orientation predicate how can we be sure that we actually compute the same convex hull? The answer is to make sure that we change the orientation predicate in a manner that remains consistent with the general orientation predicate, so that the convex hull computed in non-degenerated cases is unchanged.

The solution we are going to present is based on the idea that if points are aligned, then moving one of the points only slightly will remove the degeneracy. This makes it possible to compute an approximation of the hull. However, points are not actually moved, and all points that should appear on the convex hull will be guaranteed to appear in the approximation we compute. Imprecisions will occur only for points that are on the segment between two legitimate vertices of the convex hull. For these points, this method may or may not include them in the resulting convex hull.

The solution is to consider that all points are moving in a continuous manner and that the configuration that is being studied is the snapshot at date  $t = 0$ . The determinants used to compute the orientation predicate also are continuous functions in  $t$ , that may have the value 0 when  $t = 0$ . The orientation predicate is continuous for all unaligned triples. However, if the perturbation function can be viewed as a polynomial function in  $t$  it is possible to ensure that  $\lim_{t \rightarrow 0^+}$  is either positive or negative. If the points are not aligned, the limit is the same as the value in 0, if there are aligned points, taking a value of  $t$  that is small enough will always return the same sign, so that we do not even have to predict the value of  $t$ : the sign can be predicted from the signs of the polynomial coefficient.

We use a function taken from [1] to indicate how all points move. The coordinates of the point  $p$  actually are given by the formula  $x_p + t \times y_p, y_p + t \times (x_p^2 + y_p^2)$ . The orientation predicate for points  $x, y, z$  then becomes the following term:

$$\begin{vmatrix} x_p + t \times y_p & y_p + t \times (x_p^2 + y_p^2) & 1 \\ x_q + t \times y_q & y_q + t \times (x_q^2 + y_q^2) & 1 \\ x_r + t \times y_r & y_r + t \times (x_r^2 + y_r^2) & 1 \end{vmatrix} = D_1 + D_2 \times t + D_3 \times t^2,$$

where  $D_1, D_2$ , and  $D_3$  are the determinants defined as follows:

$$D_1 = \begin{vmatrix} x_p & y_p & 1 \\ x_q & y_q & 1 \\ x_r & y_r & 1 \end{vmatrix} \quad D_2 = \begin{vmatrix} y_p & x_p^2 + y_p^2 & 1 \\ y_q & x_q^2 + y_q^2 & 1 \\ y_r & x_r^2 + y_r^2 & 1 \end{vmatrix} \quad D_3 = \begin{vmatrix} x_p & x_p^2 + y_p^2 & 1 \\ x_q & x_q^2 + y_q^2 & 1 \\ x_r & x_r^2 + y_r^2 & 1 \end{vmatrix}.$$

The proof that all three determinants cannot be zero is formalized using the sketch provided in [1] and relies very much on deciding equalities in a ring, plus a lot of painful reasoning on equalities. However, we work on polynomials and all our proofs are done in an ordered archimedean field: the field of rational numbers suffices so that our model is adequate for arithmetic numbers as they are implemented in computers.

If the first determinant is 0, then the three points are aligned. There exists three numbers  $m_1, m_2$ , and  $m_3$  such that  $m_1 \times y = m_2 \times x + m_3$  is the equation of the line containing the three points. As these points are distinct,  $m_1$  and  $m_2$  cannot both be null. There are two cases, depending on whether  $m_1 = 0$ .

If  $m_1 = 0$ , then  $x_p = x_q = x_r$ , the second determinant is equal to the following determinant:

$$\begin{aligned} \begin{vmatrix} y_p & x_p^2 + y_p^2 & 1 \\ y_q & x_q^2 + y_q^2 & 1 \\ y_r & x_r^2 + y_r^2 & 1 \end{vmatrix} &= \begin{vmatrix} y_p & y_p^2 & 1 \\ y_q & y_q^2 & 1 \\ y_r & y_r^2 & 1 \end{vmatrix} \\ &= (y_p - y_q)(y_q - y_r)(y_r - y_p). \end{aligned} \quad (1)$$

This value can only be zero if one of the factors is 0, in which case two of the points have both coordinates equal: they are the same.

If  $m_1 = 0, D_2 = 0$  if and only if  $m_1 \times D_2 = 0$ .

$$\begin{aligned} m_1^2 \times \begin{vmatrix} y_p & x_p^2 + y_p^2 & 1 \\ y_q & x_q^2 + y_q^2 & 1 \\ y_r & x_r^2 + y_r^2 & 1 \end{vmatrix} &= (m_1^2 + m_2^2) \begin{vmatrix} y_p & y_p^2 & 1 \\ y_q & y_q^2 & 1 \\ y_r & y_r^2 & 1 \end{vmatrix} \\ &= (m_1^2 + m_2^2)(y_p - y_q)(y_q - y_r)(y_r - y_p) \end{aligned} \quad (2)$$

Here again, at least two of the points must have the same vertical coordinate for the determinant to be 0.

The same reasoning applies to the third determinant, which is symmetric with the second one, replacing  $x$  coordinates with  $y$  coordinates.

Now for every triple of points, the values  $D_1, D_2$ , and  $D_3$  cannot be 0 at the same time. We can then show that for every triple  $p, q, r$  in the input data,

there exists a number  $\epsilon_{pqr}$  such that the perturbed determinant never becomes 0 in the interval  $]0, \epsilon_{pqr}[$ . Let  $abs(x)$  be the absolute value of  $x$ . If  $D_1$  is non null, then we can choose  $\epsilon_{pqr}$  to be the following value:

$$\epsilon_{pqr} = \min(1, \text{abs}(\frac{D_1}{2 \times D_2}), \text{abs}(\frac{D_1}{2 \times D_3}))$$

If  $D_2 = 0$  or  $D_3 = 0$ , just ignore the corresponding term in the minimal computation.

If  $D_1 = 0$  and  $D_3 \neq 0$ , then we can choose  $\epsilon_{pqr}$  to be the following value:

$$\epsilon_{pqr} = \min(1, \text{abs}(\frac{D_2}{D_3})).$$

If  $D_1 = 0$  and  $D_3 = 0$  we can take any positive value for  $\epsilon_{pqr}$ .

This concludes the proof of existence of  $\epsilon_{pqr}$ . This proof was also made only using axioms that are fulfilled by the field of rational numbers.

We could now replace the orientation predicate used in the previous section by a new orientation predicate, noted  $\overline{pqr}$  and defined in the following manner:

$$\begin{aligned} \overline{pqr} &= \begin{vmatrix} x_p & y_p & 1 \\ x_q & y_q & 1 \\ x_r & y_r & 1 \end{vmatrix} > 0 \\ \vee \begin{vmatrix} x_p & y_p & 1 \\ x_q & y_q & 1 \\ x_r & y_r & 1 \end{vmatrix} = 0 \wedge \begin{vmatrix} y_p x_p^2 + y_p^2 & 1 \\ y_q x_q^2 + y_q^2 & 1 \\ y_r x_r^2 + y_r^2 & 1 \end{vmatrix} > 0 \\ \vee \begin{vmatrix} x_p & y_p & 1 \\ x_q & y_q & 1 \\ x_r & y_r & 1 \end{vmatrix} = \begin{vmatrix} y_p x_p^2 + y_p^2 & 1 \\ y_q x_q^2 + y_q^2 & 1 \\ y_r x_r^2 + y_r^2 & 1 \end{vmatrix} = 0 \wedge \begin{vmatrix} x_p x_p^2 + y_p^2 & 1 \\ x_q x_q^2 + y_q^2 & 1 \\ x_r x_r^2 + y_r^2 & 1 \end{vmatrix} = 0 \end{aligned}$$

Note that the  $\epsilon_{pqr}$  are never computed, we only show their existence to show that the computation of these two determinants is enough to predict the orientations for a general position that is arbitrarily close to the input data. It is not necessary to return a convex hull composed of perturbed points: the points from the input data are correct, since they can be shown to be arbitrarily closed to a legitimate convex hull (we have not done this proof yet).

It is also interesting that the convex hull computation done in this manner returns exactly the same result as the computation described in previous section if the input data is not degenerated. If the input data is degenerated and the convex hulls are required to contain no aligned points, a trivial traversal of the list of points may be needed to remove the useless points. The cost of this traversal should be proportional to the total cost.

It is remarkable that testing the sign of the  $D_2$  can be done by simply combining the signs of the terms  $(y_p - y_q)$ ,  $(y_q - y_r)$ ,  $(y_r - y_p)$ , according to equations (1) and (2). In fact,  $D_2$  is strictly positive if and only if the line carrying the three points is not horizontal, the point  $r$  is outside the segment  $pq$ , and the points  $p$  and  $q$  are ordered in their second coordinate, similarly for  $D_3$ . As a result,

testing the sign of  $D_2$  and  $D_3$  is equivalent to testing whether  $r$  is outside the segment  $pq$  and whether  $p$  and  $q$  are ordered lexicographically (using the second coordinate first). Thus, we have a program that computes  $\overline{(pqr)}$  without computing the larger determinants above. This algebraic reasoning brings an elegant justification to the use of segment predicates and lexical ordering in combination with traditional orientation.

## 5 Conclusion

The development of certified algorithms is well suited to domains where it is easy to have a mathematical formulation of the data and the specifications. Computational geometry is one of these domains. Euclidean geometry makes it easy to connect the various notions encountered in computational geometry to real analysis and algebra. In our experiment, the presence of a decision procedure for ring equalities has been instrumental, although it has also been quite annoying that this decision procedure was not complete, at least for the version we used. This decision procedure is the `Ring tactic`, a tactic based on a reflective approach [4].

The problem of numeric computation also deserves some thoughts. We have performed our proofs with an “axiomatized” set of numbers for the coordinates. The axioms we have taken were carefully chosen to make them accurate as a representation of computer numbers used in exact computation. These numbers live in an archimedean field. The model for the field that is needed to express the correction of the algorithm is the field of rational numbers. However, the algorithm does not actually use numbers in the whole field. It only computes a polynomial expression of degree two, so that if the numbers describing input data are bounded, the set of numbers needed for the algorithm to work is much smaller. It could be useful to express the difference between the “concrete” arithmetic used in the algorithm, and the “abstract” arithmetic used in the proof. This is actually the spirit of the work of Laurent théry and his colleagues in a project to study computer arithmetics.

Perturbation methods are used in other parts of computational geometry, for instance for Delaunay triangulation, where degenerate positions occur when four points are cocyclic. They seem very well adapted as a method to make algorithms robust to degeneracy without changing their structure. Still, it seems a rare and lucky occurrence that perturbations are used in this paper to justify an algorithm that actually does not use any perturbation, since it only boils down to an extension of the orientation predicate with a segment predicate and lexical ordering.

We actually heard about the perturbation method after having done most of the proofs for the algorithm based on the extended axioms combining the traditional orientation and segment predicates. Still both developments are useful in their own right, since the first development can be tuned to include or exclude colinear points from the convex hull at will. For the second method, we can predict that colinear points on edges of the convex hull that grow lexicographically will

be included in the convex hull, while colinear points on edges that decrease lexicographically will not be included. This information that we have not formally proved at the time of writing this article, may be used when writing a second pass that removes colinear points, but it won't be easy to recover colinear points that have already been discarded by the convex hull computation.

We would like to acknowledge the help of several researchers from the computational geometry group at INRIA Sophia Antipolis. In particular, Mariette Yvinec pointed us to Knuth's work on axioms and Olivier Devillers described us the perturbation method.

## References

1. Pierre Alliez, Olivier Devillers, and Jack Snoeyink. Removing degeneracies by perturbing the problem or the world. *Reliable Computing*, 6:61–79, 2000. Special Issue on Computational Geometry.
2. Bruno Barras, Samuel Boutin, Cristina Cornes, Judicael Courant, Yann Coscoy, David Delahaye, Jean-Christophe Filliatre Daniel de Rauglaudre, Eduardo Gimenez, Hugo Herbelin, Gerard Huet, Cesar Munoz, Chetan Murthy, Catherine Parent, Christine Paulin-Mohring, Amokrane Saibi, and Benjamin Werner. *The Coq Proof Assistant User's Guide. Version 6.3.1*. INRIA, December 1999.
3. Jean-Daniel Boissonnat and Mariette Yvinec. *Algorithmic geometry*. Cambridge University Press, 1995.
4. Samuel Boutin. Using reflection to build efficient and certified decision procedures. In *Theoretical Aspects of Computer Science*, number 1281 in Lecture Notes in Computer Science. Springer-Verlag, 1997.
5. S.C. Chou, X.S. Gao, and J.Z. Zhang. *Machine Proofs in Geometry*. World Scientific, Singapore, 1994.
6. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms*. MIT-Press, 1990.
7. Gilles Kahn. Elements of constructive geometry group theory and domain theory, 1995. available as a Coq contribution at <http://coq.inria.fr/contribs-eng.html>.
8. Donald Knuth. *Axioms and Hulls*. Number 606 in Lecture Notes in Computer Science. Springer-Verlag, 1991.
9. François Puitg and Jean-François Dufourd. Formal specification and theorem proving breakthroughs in geometric modeling. In *Theorem Proving in Higher-Order Logics*, volume 1479 of *Lecture Notes in Computer Science*, pages 410–422. Springer-Verlag, 1998.
10. Robert Sedgewick. *Algorithms*. Addison-Wesley, 1988.
11. Jan von Plato. A constructive theory of ordered affine geometry. *Indagationes Mathematicae*, 9:549–562, 1998.