Lecture 3
GEOMETRIC COMPUTATION, I

Among numerical software, those that might be described as "geometric" are notoriously non-robust. Understanding geometric computation is key to solving the non-robustness problem. In this chapter, we introduce the reader to some simple geometric problems related to the convex hull. Various forms of non-robustness can be illustrated using such examples.

§1. Convex Hull of Planar Points

The prototype geometric objects are points and lines in the plane. We begin with one of the simplest problems in geometric computing: computing the convex hull of a set of points in the plane. The one-dimensional analogue of convex hull is the problem of computing the maxima and minima of a set of numbers. Like sorting (which also has a geometric analogue in higher dimensions), our convex hull problem has been extensively studied and there are many known algorithms known for the problem. This chapter will introduce several such algorithms, to investigate their non-robustness properties. We first develop a convex hull algorithm in a leisurely fashion.

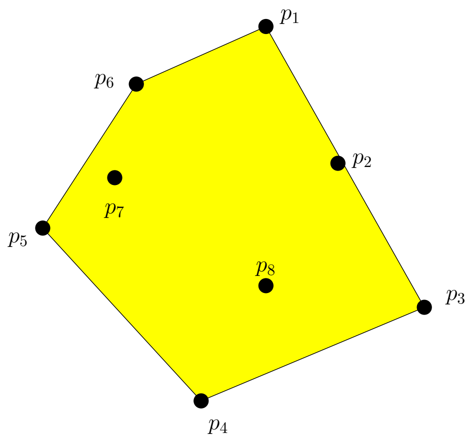

Figure 1: Convex hull of a set of points

The computational problem is this: given a set S of points in the plane, compute its convex hull. Figure 1 displays the convex hull of the set $S = \{p_1, p_2, \ldots, p_8\}$ of points. The output is essentially a convex polygon determined by the five points p_1, p_6, p_5, p_4, p_3. These points are said to be **extreme** in S (see the Appendix of this Chapter for some the basic definitions).

The output needs to be specified with some care: we not only want to determine the extreme points in S, but we also want to list them in some specific circular order, either clockwise or counter clockwise order. For specificity, choose the clockwise order. Thus, for the input set in Figure 1, we want to list the output set $\{p_1, p_6, p_5, p_4, p_3\}$ in a specific order $(p_1, p_3, p_4, p_5, p_6)$ or any cyclic equivalent, such as $(p_4, p_5, p_6, p_1, p_3)$.

A possible ambiguity arises. Are points on the relative interior of a convex hull edge considered "extreme"? In Figure 1, the point p_2 just inside the convex hull. But suppose p_2 lies on the edge $[p_1, p_3]$: should it be included in our convex hull output? According to our definition in the Appendix, points in the relative interior of the edge $[p_1, p_3]$ are not considered extreme. It is also possible to define our convex hull problem so that such points are required to appear in our circular list.

¶1. Checking Extremity of a Point. One approach to computing a convex hull of S is to reduce it to checking if a point $p \in S$ is extremal in S. We eventually show that if we retain some extra information in extremity checking, we can link them into a circular list corresponding to a convex hull.

So how do we check if a point p is extreme in S? First we make a simple observation: it is easy to find at least one extremal points: if $p_{\min} \in S$ is the lexicographically smallest point in S, then p is extremal. If S has at least two points, then the lexicographically largest point $p_{\max} \in S$ is another extremal point.

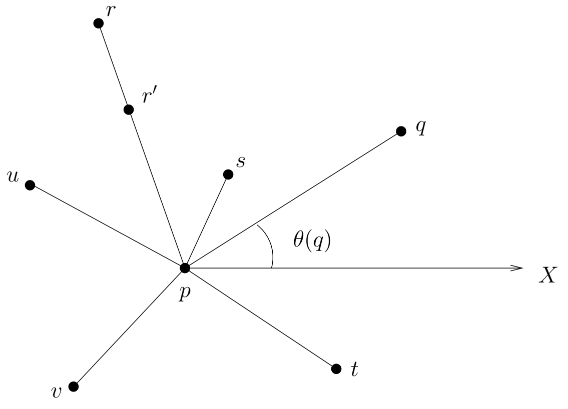

Figure 2: Circular sorting S about p.

In general, we can easily reduce the question of extremity of p in S to another problem: circular sorting of $S \setminus \{p\}$ about p. This is the problem where, for each point $q \in S \setminus \{p\}$, define $\theta(q) = \angle(Xpq)$ to be the angle swept as the ray $\overrightarrow{pX}$ turn counter-clockwise to reach the ray $\overrightarrow{pq}$. Here, X is any point to the right of p (see Figure 2). The problem of **circular sorting** of S about the point p amounts to sorting the set

$$\{\theta(q) : q \in S \setminus \{p\}\}.$$

For instance, sorting the set S in Figure 2 gives the output (q, s, r, r', u, v, t). In case of ties for two points (e.g., r, r'), their relative order is not important for our purposes (e.g., we could also output (q, s, r', r, u, v, t)).

Let $(p_1, \ldots, p_{n-1})$ be output of circular sorting $S \setminus \{p\}$ about p. Then p is extreme iff there is some $1 \le i \le n$ such that

$$\theta(p_i) - \theta(p_{i-1}) > \pi. \tag{1}$$

When $i = 1$, we must interpret $\theta(p_0)$ in (1) as $\theta(p_n) - 2\pi < 0$.

Circular sorting has interest in its own right (see Graham scan below). But for the purpose of deciding whether p is extreme, circular sorting is an overkill. Let us develop a simpler method.

Here is one approach, illustrated in Figure 3. Let $p = (p_x, p_y)$ and $S' = S \setminus \{p\}$. We split S' into two sets $S' = S_a \uplus S_b$ where S_a comprise those points $q \in S'$ that lies *above* the X-axis: either $q_y > p_y$ or $(q_y = p_y$ and $q_x > p_x)$. Similarly, S_b comprise those points $q \in S'$ lying *below* the X-axis. Thus, a point v on the positive X-axis would belong to S_a while a point u on the negative X-axis would belong to S_b (see Figure 3). Define

$$\theta_a^- := \min\{\theta(q) : q \in S_a\}, \quad \theta_a^+ := \max\{\theta(q) : q \in S_a\}$$

When S_a is empty, $\theta_a^- = +\infty$ and $\theta_b^+ = -\infty$. Similarly let $\theta_b^- := \min\{\theta(q) : q \in S_b\}$ and $\theta_b^+ := \max\{\theta(q) : q \in S_b\}$. Note that $0 \le \theta_a^- \le \theta_b^+ < \pi$ if S_a is non-empty, and $\pi \le \theta_b^- \le \theta_b^+ < 2\pi$ if S_b is non-empty. The following is easy to see:

Lemma 1. *Point p is extreme in S iff one of the following three conditions hold:*
1. S_a or S_b is empty.
2. $\theta_b^+ - \theta_a^- < \pi$.
3. $\theta_b^- - \theta_a^+ > \pi$.

Based on this lemma, it is now easy to give an $O(n)$ time algorithm to check for extremity:

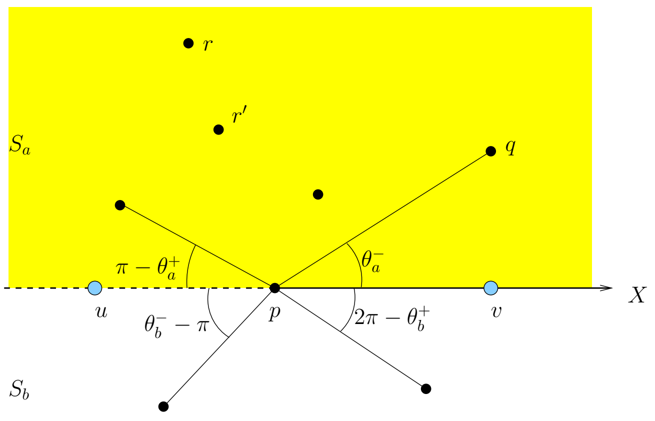

Figure 3: Checking extremity of point p.

```
Extremity Test (Conceptual)
Input: A set of points $S$ and $p \in S$.
Output: True if $p$ is extreme in $S$ and false otherwise.
1.  $\theta_a^- \leftarrow \theta_b^- \leftarrow +\infty$,  $\theta_a^+ \leftarrow \theta_b^+ \leftarrow -\infty$.    $\triangleleft$ Initialization
2.  for each $q \in S \setminus \{p\}$    $\triangleleft$ Main Loop
3.      if $(q.Y > p.Y)$ or $(q.Y = p.Y$ and $q.X > p.X)$
4.          then $\theta_a^- \leftarrow \min\{\theta_a^-, \theta(q)\}$ and $\theta_a^+ \leftarrow \max\{\theta_a^+, \theta(q)\}$.
5.          else $\theta_b^- \leftarrow \min\{\theta_b^-, \theta(q)\}$ and $\theta_b^+ \leftarrow \max\{\theta_b^+, \theta(q)\}$.
6.  if $[(\theta_a^-$ or $\theta_b^-$ is infinite) or $(\theta_b^+ - \theta_a^- < \pi)$ or $(\theta_b^- - \theta_a^+ > \pi)]$
            return True.
7.  else
            return False.
```

We leave the correctness proof to an Exercise. Note that this algorithm assumes the ability to compute the angles $\theta(q)$ for $q \in S$. As discussed next, this is not a practical assumption.

Exercises

Exercise 1.1:
(i) Prove the correctness of Lemma 1.
(ii) Prove the correctness of the (Conceptual) Extremity Test. ◊

Exercise 1.2: Develop another conceptual Extremity Test in which we assume that the two extreme points $p_{\min}$ and $p_{\max}$ have already been found, and we want to test another point $p \in S \setminus \{p_{\min}, p_{\max}\}$ for extremity. HINT: The idea is that having the extra information about $p_{\min}$ and $p_{\max}$ should simplify our task. To further simplify the problem, you may assume that p lies below the line through $p_{\min}$ and $p_{\max}$. ◊

Exercise 1.3: Generalize Lemma 1 to three dimensions. Give a corresponding algorithm to check extremity of a point in $S \subseteq \mathbb{R}^3$, in analogy to the previous exercise. ◊

End Exercises

§2. Some Geometric Predicates

We consider the above Extremity Test based on Lemma 1 to be a "conceptual" algorithm, not a practical one. This is because the algorithm requires the computation of angles $\theta(q)$, which are transcendental[1] quantities. Thus, the computed angles are necessarily approximate, and it is non-trivial to turn this into an exact algorithm. More generally, we must be wary of the Real RAM model often assumed in theoretical algorithms. Real computational difficulties, including undecidable questions, might lurk hidden its abstraction. To avoid the computation of angles, we now introduce an important low-level tool in Computational Geometry: the signed area of an oriented triangle.

¶2. Signed Area. Let A, B, C be three points in the plane. The **signed area** of ABC is half of the following determinant

$$\Delta(A,B,C) := \det\begin{bmatrix} a & a' & 1 \\ b & b' & 1 \\ c & c' & 1 \end{bmatrix} \tag{2}$$

$$= \det\begin{bmatrix} a & a' & 1 \\ b-a & b'-a & 0 \\ c-a & c'-a & 0 \end{bmatrix} = \det\begin{bmatrix} b-a & b'-a' \\ c-a & c'-a' \end{bmatrix} \tag{3}$$

where $A = \mathsf{Point}(a, a'), B = \mathsf{Point}(b, b')$, etc.

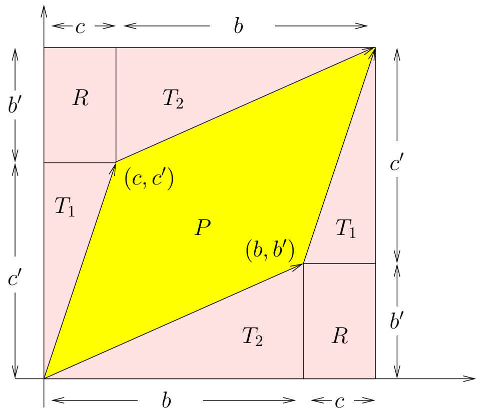

Figure 4: Area of parallelogram determined by vectors $(b, b'), (c, c')$.

To justify of this formula, we assume without loss of generality that $a = a' = 0$. First assume that both vectors (b, b') and (c, c') are in the first quadrant, with vector (b, b') making a larger angle with the positive x-axis than vector (c, c'). Referring to Figure 4, the area of the parallelogram P is equal to the area of the enclosing rectangle $(b + b')(c + c')$ minus the area of the two smaller rectangles $2R = 2bc'$, minus the area of the 4 triangles $2(T_1 + T_2) = bb' + cc'$. This yields

$$\Delta(A,B,C) = (b + b')(c + c') - (2bc' + bb' + cc') = b'c - bc'.$$

The above figure assumes that the angle $\theta(B) \leq \theta(C)$. In case $\theta(B) > \theta(C)$, we obtain the same area with an negative sign. We leave it as an exercise to verify this formula even when the 2 vectors are not both in the first quadrant.

This formula generalizes to volume in 3 and higher dimensions. For instance, the signed volume of the tetrahedron defined by four points $A, B, C, D \in \mathbb{R}^3$, given in this order, is

$$\frac{1}{6}\det\begin{bmatrix} A & 1 \\ B & 1 \\ C & 1 \\ D & 1 \end{bmatrix} \tag{4}$$

[1]A transcendental computation is one that involves non-algebraic numbers such as π. Assuming the input numbers are algebraic, then $\theta(q)$ is generally non-algebraic. Algebraic and transcendental numbers will be treated in detail later in the book.

where the row indicated by $(A, 1)$ is filled with the coordinates of A followed by an extra "1", etc. The factor of $1/6$ will be $1/d!$ in d-dimensions.

¶3. The Orientation Predicate. Using the signed area, we define several useful predicates. The orientation predicate is defined as

$$\mathsf{Orientation}(A, B, C) = \mathtt{sign}(\Delta(A, B, C)).$$

Thus the orientation predicate is a three-valued function, returning $-1, 0$ or $+1$. In general, a **predicate** is any function with a (small) finite number of possible outcomes. In case there are only two (resp., three) possible outcomes, we call it a **truth predicate** (resp., **geometric predicate**). Truth predicates are well-known and widely used in mathematical logic and computer science. Geometric predicates are perhaps less well-known, but they arise naturally as in the following examples:

- Relative to a triangle, a point can be **inside**, **outside**, or **on the boundary**.
- Relative to an oriented hyperplane, a point can be **left**, **right**, or **on the hyperplane**.
- Two lines in the plane can be **intersect** in a single point, be **coincident**, or **non-intersecting**.
- Two triangular areas can be **disjoint**, have a **containment** relationship, or their boundaries may **intersect**.

Geometric relations with more than 3 values are clearly also possible. E.g., the relation between two lines in 3-D space is 4-valued because they are either co-planar (in which case we can classify them into the 3 cases as above), or they can be skew. We also note that 3-valued predicates have a well-known non-geometric application in which the 3 values are interpreted as **true**, **false** and **unknown**. It is then viewed as a generalization of 2-valued logic.

What is the significance of the orientation predicate? One answer is immediate: when the output is 0, it means that A, B, C are collinear since the area of the triangle is zero exactly in this case. For instance if two of the points are coincident, then the output is 0. What about the non-zero outputs? Let us do a simple experiment: let $A = (0, 0), B = (1, 0)$ and $C = (0, 1)$. Then we see that

$$\Delta(A, B, C) = \det \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = 1.$$

Thus, if you look from the point $(0, 0)$ towards the point $(1, 0)$, then $(0, 1)$ is to the left. In general:

CLAIM: *If A, B, C are not collinear, and you look from point A to point B, then point C lies to the left if and only if* $\mathsf{Orientation}(A, B, C) = +1$.

We argue as follows: fix $A = (0, 0)$ and $B = (1, 0)$ and let C vary continuously, starting from the position $(0, 1)$. As long as C does not enter the X-axis, the area $\Delta(A, B, C)$ remains non-zero. Since the area is a continuous function, we conclude that area will remain positive, i.e., $\mathsf{Orientation}(A, B, C) = +1$. By the same token, as long as C lies strictly below the X-axis, the area remains negative and so $\mathsf{Orientation}(A, B, C) = -1$. Moreover, the area vanishes iff C lies on the X-axis. This proves our CLAIM when A, B are the special points $(0, 0)$ and $(1, 0)$.

In general, if A, B, C is an arbitrary non-collinear triple, there is a unique a rigid transformation T that transforms (A, B) to $(T(A), T(B)) = ((0, 0), (1, 0))$: first translate A to $(0, 0)$, then rotate B to the positive X axis, and finally scale B to $(1, 0)$. A translation, rotation and scaling transformation is represented by a 3×3 matrix with positive determinant (Exercise). Thus T has positive determinant. Moreover, we see that

$$\Delta(T(A), T(B), T(C)) = \det(T)\Delta(A, B, C). \tag{5}$$

Since $\det(T) > 0$, the transformation is orientation preserving. Finally, we note that (A, B, C) is a left-turn (in the sense of the CLAIM) iff $T(C)$ is in the upper half plane. Hence our CLAIM holds generally.

¶4. Some Truth Predicates. The orientation predicate is a geometric (3-valued) predicate. We now define several useful truth (2-valued) predicates from this single geometric predicate.

In the CLAIM above, we have already implicitly defined the left-turn predicate:

$$\mathsf{LeftTurn}(A,B,C) \quad\equiv\quad \text{“}\mathsf{Orientation}(A,B,C) > 0\text{”}.$$

Similarly, the $\mathsf{RightTurn}(A,B,C)$ predicate gives the truth value of “$\mathsf{Orientation}(A,B,C) < 0$”.

Our goal is to reduce the Extremity Test to truth predicates such as $\mathsf{LeftTurn}$. Unfortunately, $\mathsf{LeftTurn}$ by itself is not enough in the presence of “degenerate” inputs. For our extremity problem, a set of points is degenerate if three input points are collinear. We need new predicates to take into account the relative disposition of three collinear points.

For instance, in Figure 2, when we discover that p, r, r' are collinear, we want to discard r' as non-extreme. So we define a helper predicate, called $Between(A,B,C)$, which evaluates to true iff A, B, C are collinear with B strictly between A and C. We define

$$Between(A,B,C) \quad\equiv\quad \text{“}\mathsf{Orientation}(A,B,C) = 0 \text{ and } \langle A-B, C-B\rangle < 0\text{”}$$

where $\langle u, v\rangle$ is just scalar product of two vectors. To see this, from elementary geometry, we see that $\langle A-B, C-B\rangle = \|A-B\|\cdot\|C-B\|\cos\theta$ provided $A \neq B$ and $B \neq C$ and θ is angle between the vectors $A-B, C-B$. Since $\langle A-B, C-B\rangle < 0$, we know that $A \neq B$, $B \neq C$ and $\cos\theta < 0$. Since A, B, C are collinear, $\theta = 0$ or $\theta = \pi$. We conclude that θ must in fact be π; this proves that B lies strictly between A and C.

Define the **modified Left Turn Predicate** as:

$$LeftTurn^*(A,B,C) \quad\equiv\quad \text{“}\mathsf{Orientation}(A,B,C) > 0 \text{ or } Between(C,A,B)\text{”}.$$

The modified Right Turn Predicate is similar,

$$RightTurn^*(A,B,C) \quad\equiv\quad \text{“}\mathsf{Orientation}(A,B,C) < 0 \text{ or } Between(C,A,B)\text{”}.$$

Note that in both calls to the $Between$ predicate, the arguments are (C, A, B) and not (A, B, C).

¶5. Enhanced Extremity Test. We now return to our conceptual Extremity Test. Our plan is to modify it into a practical algorithm by using the various predicates which we just developed.

Before installing the modified turn predicates, we make another adjustment: instead of computing the angle θ_a^-, we only remember the point in S_a that determines this angle. This point is denoted p_a^-. Similarly, we replace the other 3 angles by the points p_a^+, p_b^- and p_b^+. Initially, these points are initialized to the NULL value. Moreover, the $LeftTurn^*(p,q,r)$ predicate always return true if any of the arguments is NULL.

We also enhance our output as follows: if the point p is extreme in S, we output a triple (r,p,q) such that p, q are also extreme in S and $\mathsf{LeftTurn}(r,p,q) = \text{TRUE}$. if the point p is non-extreme in S, we output 3 points $u, v, w \in S \setminus \{p\}$ such that p in contained in the triangle (u, v, w) and $\mathsf{LeftTurn}(u,v,w)$. This is captured by another truth predicate defined as follows:

$$Inside(p,u,v,w) \quad\equiv\quad LeftTurn^*(p,u,v) \wedge LeftTurn^*(p,v,w) \wedge LeftTurn^*(p,w,u). \tag{6}$$

The above considerations finally lead to the following algorithm for the extremity test. It is best to understand this algorithm as a refinement of the original Conceptual Extremity Test.

```
EXTREMITY TEST
Input: A set of points $S$ and $p \in S$. Assume $|S| \geq 2$.
Output: If $p$ is extreme in $S$, output $(r, p, q)$ such that
        $r, q$ are extreme and $\mathsf{LeftTurn}(r, p, q) = \text{TRUE}$.
        Else, output the points $u, v, w \in S \setminus \{p\}$ such that
        $p$ lies inside triangle $(u, v, w)$ and $\mathsf{LeftTurn}(r, p, q) = \text{TRUE}$.
1.  ▷ INITIALIZATION:
        $p_a^- \leftarrow p_b^- \leftarrow p_a^+ \leftarrow p_b^+ \leftarrow \text{NULL}$.
2.  ▷ MAIN LOOP:
        for each $q \in S \setminus \{p\}$
3.          if $(q.Y > p.Y)$ or $(q.Y = p.Y$ and $q.X > p.X)$
4.              then
                    if $LeftTurn^*(p_a^-, p, q)$ then $p_a^- \leftarrow q$
                    if $RightTurn^*(p_a^+, p, q)$ then $p_a^+ \leftarrow q$
5.              else
                    if $LeftTurn^*(p_b^-, p, q)$ then $p_b^- \leftarrow q$
                    if $RightTurn^*(p_b^+, p, q)$ then $p_b^+ \leftarrow q$
6.  ▷ OUTPUT FOR EXTREME CASES:
        if $(p_a^- = \text{NULL})$ then return $(p_b^+, p, p_b^-)$.
        if $(p_b^- = \text{NULL})$ then return $(p_a^+, p, p_a^-)$.
        if $LeftTurn^*(p_a^+, p, p_b^-)$ then return $(p_a^+, p, p_b^-)$
        if $LeftTurn^*(p_b^+, p, p_a^-)$ then return $(p_b^+, p, p_a^-)$
7.  ▷ OUTPUT FOR NON-EXTREME CASES:
        if $Inside(p, p_a^-, p_a^+, p_b^-)$ then return $(p_a^-, p_a^+, p_b^-)$
        else return $(p_a^-, p_b^-, p_b^+)$
```

Note that above algorithm uses only two kinds of predicates: the modified left- and right-turns.

Exercises

Exercise 2.1:
(i) Give the general formula for the 3×3 matrix T that transforms any pair (A, B) of distinct points to $((0,0),(1,0))$.
(ii) What is T when $(A, B) = ((-2,3),(3,2))$? ♢

Exercise 2.2: A fundamental property of the left turn predicate is that

$$\mathsf{LeftTurn}(A, B, C) \equiv \mathsf{LeftTurn}(B, C, A) \equiv \mathsf{LeftTurn}(C, A, B).$$

Kettner and Welzl [3] reported that for a randomly generated set of 1000 points in the unit box $[0,1) \times [0,1)$, about one third of the $\binom{1000}{3}$ triples (A, B, C) of points fails this identity when evaluated in IEEE single precision coordinates. No failures were observed in IEEE double precision coordinates. Please repeat their experiments. ♢

Exercise 2.3:
(i) Implement the Extremity Test algorithm.
(ii) Run your algorithm on the following input points taken from [2].

$$\begin{aligned}
p_1 &= (\ 7.30000\,00000\,00019\,4,\ 7.30000\,00000\,00016\,7),\\
p_2 &= (24.00000\,00000\,00068,\ \ 24.00000\,00000\,00061\ ),\\
p_3 &= (24.00000\,00000\,0005,\ \ 24.00000\,00000\,00053\ ),\\
p_4 &= (\ 0.50000\,00000\,00016\,21,\ 0.50000\,00000\,00012\,43),\\
p_5 &= (8,4),\\
p_6 &= (4,9),\\
p_7 &= (15,27),
\end{aligned}$$

$$\begin{aligned} p_8 &= (26,25), \\ p_9 &= (19,11). \end{aligned}$$

◊

Exercise 2.4: Develop an Extremity Test for 3 dimensional points. Fix a set $S \subseteq \mathbb{R}^3$ and a point $p \in S$. To see if p is extremal, you want to find two other points q, r such that relative to the plane $H(p,q,r)$ spanned by $\{p,q,r\}$, all the remaining points of S lies strictly to one side of this plane. ◊

End Exercises

§3. Simplified Graham Scan

There are probably a dozen known convex hull algorithms. One of the earliest algorithms is called the "Graham Scan", from Ron Graham (1975). The idea of Graham scan was to first perform an angular sorting of the input set S about an arbitrary point $p^* \notin S$ (e.g., p^* may be origin). If $(p_1, \ldots, p_n)$ is this sorted list, we then incrementally compute the convex hull of the prefix $(p_1, \ldots, p_i)$ for $i = 2, \ldots, n$. There is a very simple stack-based algorithm to achieve this incremental sorting.

We now describe variant that is described by Kettner and Welzl [3] (it is a variant of what is known as "Andrew Sort"). The basic idea is choose let $(p_1, \ldots, p_n)$ be the lexicographical sorting of S. This can be interpreted as choosing p^* sufficiently far to the west and south of S for the Graham scan; see Exercise. Besides its simplicity, we will also use it to illustrate an algorithmic idea called "conservative predicates".

Reducibility Among Some Geometry Problems: Convex hull in 1-D is computing the max-min of a set of points. Convex hull in 2-D is rather like sorting since a 2-D convex hull is almost like a linearly sorted list. Indeed, this section shows how convex hull in 2-D can be reduced to sorting in 1-D. Similarly, convex hull in 3-D can be reduced to sorting in 2-D (there is a notion of "sorting points" in k-D). Another such reduction is that Voronoi diagrams (which will be introduced later) in k-dimensions and be reduced to convex hull in $(k+1)$-dimensions.

It is a generally useful device to split the 2-D convex hull problem into two subproblems: computing an upper hull and a lower hull. Recall that

$$p_{\min}, p_{\max} \in S$$

are two extremal points of the convex hull of S (assuming S has at least two points). Hence, we may relabel the points on the convex hull so that

$$(p_{\min}, p_1, \ldots, p_{k-1}, p_k, p_{\max}, q_\ell, q_{\ell-1}, \ldots, q_1)$$

is the counter-clockwise listing of the convex hull of S. Here, $k \geq 0$ and $\ell \geq 0$. Then we may call the two subsequences

$$(p_{\min}, p_1, \ldots, p_{k-1}, p_k, p_{\max}), \qquad (p_{\min}, q_1, \ldots, q_{\ell-1}, q_\ell p_{\max})$$

the **lower** and **upper hulls** of S. See Figure 5 for an illustration.

By symmetry, it is sufficient to show how to compute the lower hull of S. Suppose we begin by computing the lexicographic ordering of all the points in S, discarding any duplicates we find. Let this lexicographic ordering be

$$L = (p_{\min}, p_1, p_2, \ldots, p_m, p_{\max}) \tag{7}$$

where $p_{i-1} <_{\texttt{LEX}} p_i$ for all $i = 1, \ldots, m+1$, with $p_0 = p_{\min}, p_{m+1} = p_{\max}$.

Clearly, the lower hull of S is a subsequence of L. In general, any lexicographically sorted sequence of the form (7) that contains the lower hull as a subsequence, with the first and last elements $p_{\min}$ and $p_{\max}$, is called a **liberal hull**. Temporarily, we call a point "good" if it belongs to the lower hull, and "bad" otherwise.

The idea of Kettner and Welzl is to begin with a liberal hull, and successively eliminate bad points. A simple observation is this:

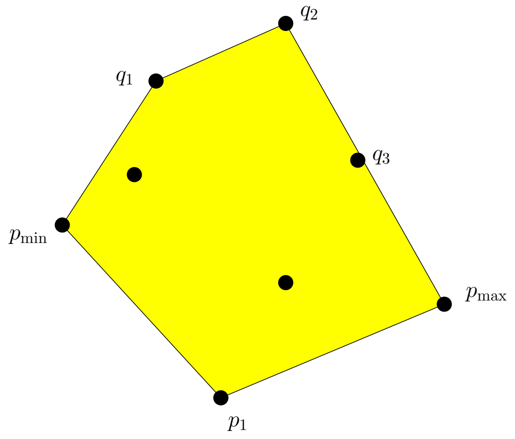

Figure 5: Lower $(p_{\min}, p_1, p_{\max})$ and upper $(p_{\min}, q_1, q_2, q_3, p_{\max})$ hulls

Observation 1. *Let (p, q, r) be 3 points in a liberal hull with $p <_{\texttt{LEX}} q <_{\texttt{LEX}} r$.*
(a) If the predicate

$$\mathsf{LeftTurn}(p, r, q)$$

holds, then q is bad.
(b) If q is bad, then there exists some p, r such that (a) holds.

So we can use this predicate to test the badness of any point q as long as $q \notin \{p_{\min}, p_{\max}\}$. How can we systematically apply this observation to eliminate bad points? It turns out that to test $q = p_i$, it is enough to choose $(p, q, r) = (p_{i-1}, p_i, p_{i+1})$, provided we do this systematically.

Assume that the points are distinct and have been lexicographically sorted in an array $P[0..n-1]$, and further $p_0 = p_{\min}$ and $p_{n-1} = p_{\max}$. Then one way to test for bad points is to go from left to right in the array. Suppose that the subarray $P[0..i-1]$ has been tested, and all the good points have been moved (left-shifted) into $P[0..k]$ for some $k \le i-1 < m$. Therefore $P[0..k]$ is actually a lower hull for $P[0..i-1]$. Since

$$P[0..k]; P[i..n-1] = P[0..k; i..n-1]$$

is a liberal hull, we can use the above observation to test if $P[k]$ is bad: if

$$\mathsf{LeftTurn}(P[k], P[k-1], P[i])$$

holds, then we can discard $P[k]$ (this amounts to setting $k \leftarrow k-1$). If $P[k]$ is not bad, we extend our induction hypothesis by incrementing k, and then moving $P[i]$ to the kth array location:

$$k := k+1; \quad P[k] \leftarrow P[i].$$

This can summarized be captured in the following algorithm:

```
LowerHullScan($S$)
Input: $S$ is a set $n > 1$ points.
Output: $k$ and array $P[0..k]$ representing the lower hull of $S$.
    ▷ Initialization
    Compute the lexicographic sort of $S$.
        Let the sorted sequence be $P[0..m]$ with $P[i-1] <_{\mathsf{LEX}} P[i]$ $(1 \le i \le m < n)$
        If $m \le 1$, return$(m, P)$.
        $k \leftarrow 1$
    ▷ Main Loop
    for $(i = 2; i \le m; i \leftarrow i + 1)$
        ▷ Invariant: $P[0..k]$ is a lower hull of $P[0..i-1]$.
        while $(k > 1$ and LeftTurn$(P[k], P[k-1], P[i]))$
            $k \leftarrow k - 1$
        $k \leftarrow k + 1$; $P[k] \leftarrow P[i]$
    $P[k+1] \leftarrow P[m]$; return$(k+1, P)$
```

¶6. How to Use Conservative Predicates. Let us now add a new twist from Kettner and Welzl: suppose we implement an imperfect predicate, $\mathsf{LeftTurn}'(A, B, C)$ with the property that for all points A, B, C, we have

$$\mathsf{LeftTurn}'(A, B, C) \Rightarrow \mathsf{LeftTurn}(A, B, C).$$

Then $\mathsf{LeftTurn}'$ is called a **conservative version** of LeftTurn. In this setting, we may describe the original predicate as **exact**.

In terms of discarding bad points, if we use $\mathsf{LeftTurn}'$, we may sometimes fail to detect bad points. But that is OK, because we are interested in liberal lower hulls. Of course, if we define $\mathsf{LeftTurn}'(A, B, C)$ to be always false, this is not very useful. The point is that we can implement conservative predicates $\mathsf{LeftTurn}'(A, B, C)$ which are (1) very efficient, and (2) agrees with the exact predicate most of the time. If property (2) holds, we say the predicate is **efficacious**.

If we use a conservative version of LeftTurn in our LowerHullScan Algorithm above, we would have computed a liberal lower hull. To compute the true lower hull, we proceed in three phases as follows:

- Sorting Phase: sort the points lexicographically.
- Filter Phase: run the while loop of the LowerHullScan Algorithm using a conservative $\mathsf{LeftTurn}'$ predicate.
- Cleanup Phase: run the while loop again using the exact LeftTurn predicate.

As borne out by experiments [3], this can speedup the overall algorithm. This speedup may sound somewhat surprising: logically speaking, the filter phase is optional. Nevertheless, filter phase is relatively fast when we use an efficient conservative predicate. If the conservative predicate is not efficacious, then the filter phase is wasted, and the algorithm is faster without it. But for a large class of inputs (especially randomly generated ones) many bad points will be eliminated by the filter phase. This will greatly speed up the Cleanup phase, and hence the overall algorithm.

This filter idea is a very general paradigm in exact geometric computation. There is another further idea: sometimes, it is possible to "verify" if something is good more efficiently than calling the exact LeftTurn predicate. In this case, the Clean Up phase call a "verify left turn predicate" first. Only when verification fails, we will call exact predicate. We will encounter these ideas of filtering and verification again.

¶7. How to Construct Conservative Predicates Let us address the issue of constructing conservative predicates which are efficient and efficacious. The general idea is to exploit fast floating point hardware operations. There is a generally applicable method: implement the operations using interval arithmetic based on machine doubles. This slows down the computation by a small constant factor.

Suppose we implement the left-turn predicate on (A, B, C) by first evaluating the determinant $\Delta(A, B, C) = (b-a)(c'-a') - (b'-a')(c-a)$ where $A = (a, a')$, etc. This is done is machine double, and then we do some simple checks to be sure that the positive sign of the computed value D is trustworthy. If so, we can output TRUE and otherwise FALSE. For instance, we can use Sterbenz' Theorem from Lecture 2.

¶8. Conservative or Liberal? Many computations, like our convex hull computation, can be viewed as an iterative process of accepting good points and rejecting bad points. There are two kinds of errors in such a computation: rejecting a good point or accepting a bad point. We then classify algorithms that may make mistakes into two camps: a **liberal algorithm** never erroneously reject a good point, while a **conservative algorithm** never erroneously accept a bad point. But a liberal algorithm may err in accepting a bad point while a conservative algorithm may err in rejecting a good point. This terminology was used in [5] for discussing testing policies in computational metrology.

Kettner and Welzl focused on predicates. If P, Q are two predicates over a common domain, we say P is a **conservative version** of Q if $P(x)$ is true implies $Q(x)$ is true. Similarly, P is a **liberal version** of Q if $P(x)$ is false implies $Q(x)$ is false.

When we use a conservative version of a correct predicate to reject points, the resulting algorithm is actually a liberal one! The converse is also true: using a liberal version to accept points will result in a conservative algorithm. By this terminology, the Kettner-Welzl algorithm is liberal, not conservative.

§4. The Gift Wrap Algorithm

Using the modified extremity test, we could construct simple convex hull algorithm in two steps. (1) First detect all extreme points, by going through each points of S. (2) Linking these extreme points into a circular list, using the adjacency information provided by the extended Extremity Test. But we can combine steps (1) and (2) into a more efficient algorithm. Suppose we have found a single extreme point using our extended Extremity Test:

$$(p_1, p_2, p_3) \leftarrow \text{Extremity Test}(S, p).$$

Assuming $p_1 \neq p_3$, we already have three extreme points to start our convex hull: $CH = (p_1, p_2, p_3)$. In general, given the partial convex hull

$$CH = (p_1, \ldots, p_i), \quad i \geq 3$$

we try to extend it by calling

$$(q_1, q_2, q_3) \leftarrow \text{Extremity Test}(S, p_i). \tag{8}$$

Note that q_2 will be equal to p_i. There are two possibilities: in case q_3 equals p_1, then the current convex hull is complete. Otherwise, we extend the partial convex hull to $CH = (p_1, \ldots, p_i, q_3)$.

A further improvement is to replace (8) by a more specialized routine:

$$q_3 \leftarrow \text{WrapStep}(S, p_{i-1}, p_i). \tag{9}$$

Here is the implementation of this specialized test. The name "Wrap Step" will be clear later.

```
WrapStep($S, p, q$)
Input: $S$ is a set $n > 2$ of points,
      and for all $s \in S$, Orientation$(q, p, s) \geq 0$.
Output: extreme point $r \in S$ such that Orientation$(q, p, r) \geq 0$.
1.    $r \leftarrow q$.
2.    for $s \in S$,
                  if $LeftTurn^*(r, p, s)$ then $r \leftarrow s$.
```

To use the WrapStep primitive, we first find initial extreme point p_1. We could obtain this using the Extremity Test, but this can be avoided: among the subset of points in S whose y-coordinates are minimum, we can pick p_1 so that $p_1.X$ is maximum. Then we call $WrapStep(S, p_0, p_1)$ where p_0 is defined to be $p_1 - \text{Point}(1, 0)$. Note that triple (S, p_0, p_1) satisfies the preconditions for inputs to $WrapStep$.

The resulting algorithm is presented here:

```
Gift Wrap Algorithm
Input: A set $S$ of $n > 2$ of points.
Output: The convex hull of $S$, $CH(S) \leftarrow (p_1, \ldots, p_h)$, $h \geq 2$.
1.  FIND INITIAL EXTREME POINT:
        Let $p_1$ be any point in $S$.
        for $s \in S$,
            if $(s.Y < p_1.Y)$ or $(s.Y = p_1.Y$ and $s.X > p_1.X)$
                then $p_1 \leftarrow s$.
2.  ITERATION:
        $CH \leftarrow ()$ (empty list).
        $q \leftarrow p_1 - (1, 0)$ and $r \leftarrow p_1$.
        do
            Append $r$ to $CH$.
            $p \leftarrow q$ and $q \leftarrow r$.
            $r \leftarrow$ WrapStep$(p, q)$.
        while
            $r \neq p_1$.
```

What is the complexity of this algorithm? If h is the size of the convex hull, then this algorithm executes the do-loop h times. Since each iteration takes $O(n)$ time, the overall complexity is $O(nh)$.

This algorithm is called the "Gift Wrap Algorithm" since it is analogous to how one wraps a gift item (albeit in 3-dimensions).

REMARK: The gift wrap algorithm can even replace the initialization loop for finding the first extreme point p_1. If we know in advance any two points

$$p_{-1}, p_0 \tag{10}$$

which are not in S, but guaranteed to form an edge in the convex hull of $S \cup \{p_{-1}, p_0\}$, we could begin our iteration with these two points. See Exercises.

¶9. Robustness Issues. Numerical nonrobustness problems in the Gift Wrap algorithm can arise from the *WrapStep* primitive. Let $(p_1, p_2, \ldots, p_h)$ is the true convex hull, with $h \geq 4$. Let us examine the possible ways that this can arise: assume the partial hull (p_1, p_2) has been computed correctly, but the next point is wrongly computed to be q instead of p_3. Evidently, p_2, q, p_3 must be nearly collinear. Assume that q is further away from p_2 than p_3 is. This is illustrated in Figure 6.

[TAKE FROM ANATOMY PAPER...]

Next, suppose that p_4 (the true 4th point on the convex hull) satisfies LeftTurn$(p_4, q, p_2) =$ True. So it is possible that our giftwrap algorithm produces the partial hull (p_1, p_2, q, p_3). It is also possible that our giftwrap algorithm produces (p_1, p_2, q, p_3, p_4).

We leave as an Exercise to construct an specific choice of the points $S = \{p_1, p_2, p_3, p_4, q\}$ with exactly this outcome, assuming all numbers are machine doubles and the arithmetic is computed using the IEEE standard.

Exercises

Exercise 4.1: Verify the area formula for all possible disposition of the vectors (b, b') and (c, c'). ◊

Exercise 4.2: Our extremity algorithm uses two kinds of predicates. Modify the algorithm so that only the modified LeftTurn predicate is used. ◊

Exercise 4.3: Device another $O(n)$ time extremity test, based on incremental insertion. Suppose you want to test if p is extremal in S. Let $S_i = \{p_1, \ldots, p_i\}$ for $i = 1, \ldots, n$ and $S_n = S \setminus \{p\}$. Suppose p is extremal in S_i, how do you check if p is extremal in S_{i+1}? You will need to maintain some witness of the extremity of p in S_i. ◊

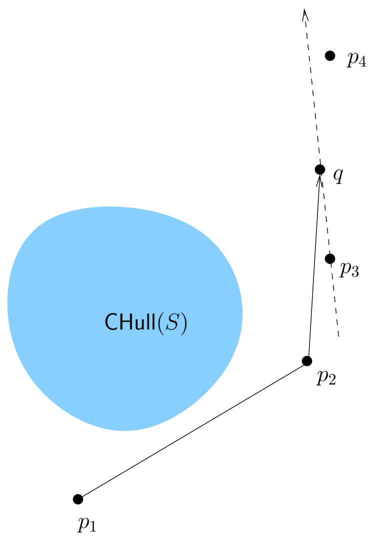

Figure 6: Error in Gift Wrap Algorithm

Exercise 4.4: Give a formula for the signed area of a simple polygon whose vertices are $A_1, A_2, \ldots, A_n$ (for $n \geq 3$). Let $A_i = \mathsf{Point}(x_i, y_i)$ for each i. ◊

Exercise 4.5: Prove the signed area of a tetrahedron in 3 dimensions. Extend this to the volume formula for a d-dimensional simplex for $d > 3$. ◊

Exercise 4.6: Suppose we can easily find a pair of artificial points p_{-1} and p_0 as in (10). Modify the Gift Wrap algorithm to produce such an algorithm. Note that such a pair of points can be manufactured easily if we admit symbolic (non-standard) numbers. ◊

Exercise 4.7: Is it possible to construct an example (assuming imprecise arithmetic) so that the Gift Wrap Algorithm goes into an infinite loop? ◊

END EXERCISES

§5. Study of an Incremental Algorithm

We now describe an incremental algorithm for computing the convex hull of a point set. The robustness issues here are more intricate because the algorithm is slightly more complicated than the previous one.

Nevertheless, its basic algorithmic idea is simple: assuming that we have computed the convex hull H_i of the first i points, we now update H_i to H_{i+1} by inserting the next point. The algorithm stops when we reach H_n.

In general, the **incremental paradigm** for geometric algorithms is quite powerful. When combined with randomization, this is often the method of choice in implementation. Indeed, randomization is often trivial – it amounts to introducing the points (or more generally, geometric object) into the construction in a random order. However, we will not dwell on this, but refer to excellent texts [1, 4]. The incremental algorithm for convex hull in every dimension is called the Beneath-Beyond algorithm (see [Edelsbrunner, page 147]).

THIS Lecture is taken from [2].

EXERCISES

Exercise 5.1: Assume the existence of a data type POINT and a predicate $\mathsf{RightTurn}(p, q, r)$. Use a doubly linked list to represent L (e.g. the STL-data type XXXX). ◊

End Exercises

§6. Other Convex Hull Methods

REMARK: The material from "Classroom Examples" is to be integrated into this chapter!!
It is instructive to see several other algorithms for the convex hull of points.

¶10. Graham Scan. Another popular convex hull algorithm, in fact the earliest $O(n \log n)$ algorithm is from R. L. Graham. The attractive feature here is the natural application of a standard data structure of computer science – the pushdown stack. We first describe the original version of this algorithm.

1. The first step is to find the $P = \mathsf{Point}(x, y)$ which is lexicographically minimum (minimize x and then minimize y). Let $P_0 = \mathsf{Point}(x_0, y_0)$ be this point. Clearly, P_0 is extremal be any definition of this word.
2. Circular Sort the other points about P_0, as explained earlier. If two points have the same angle, drop the one which is closer to to P_0. Let
$$P_0, P_1, \ldots, P_{n-1}$$
be the list of sorted points. Note that P_{n-1} is on the convex hull.
3. Create a stack S of points. Initials S with (P_{n-1}, P_0). In general, if $S = (Q_0, Q_1, \ldots, Q_k)$ were $k \geq 2$, then let *top* refer to Q_k and *next* refer to Q_{k-1}. To push an element Q onto S would make $S = (Q_1, \ldots, Q_k, Q)$. Similarly, for popping an element.
4. We process the points P_i in the i-th stage. If P_i is to the left of the line from *next* to *top* then push(P_i). Else, pop the stack.

A much more robust version of Graham scan is known as "Andrew's Version". In this version, we replace circular sorting by lexicographical sorting of the points: $p \leq_{lex} q$ iff $p_x < q_x$ or ($p_x = q_x$ and $p_y < q_y$). Since this sorting does not require any arithmetic, it is fully robust. The resulting sequence is a x-monotone in a saw-teeth fashion. We then compute the upper and lower hull of this sequence using the Graham scan.

[See paper of Kettner-Welzl]

¶11. Lower Bounds It can be shown that within a reasonable model of computation, every convex hull algorithm requires $\Omega(n \log n)$ steps in the worst case. In this sense, we usually call an algorithm with running time $O(n \log n)$ an "optimal algorithm". But witness the next result.

¶12. Ultimate Convex Hull Algorithm A remarkable algorithm was discovered by Kirkpatrick and Seidel: their algorithm takes time $O(n \log h)$. Thus, it is simultaneously better than gift-wrapping and $O(n \log n)$. Notice that we manage to beat other "optimal algorithms". Of course, this is no contradiction because we have changed the rules of the analysis: we took advantage of another complexity parameter, h. This parameter h is a function of some **output size parameter**. Algorithms that can take advantage of the output size parameters are said to be **output-sensitive**. This is an important algorithmic idea. In terms of this h parameter, a lower bound of $\Omega(n \log h)$ can be shown.

¶13. Randomized Incremental Algorithms We will return to this later, to introduce two other important themes in computational geometry: randomization and incremental algorithms. Another direction is to generalize the problem to the "dynamic setting". Here the point set S is considered a dynamic point set which can change over time. We can insert or delete from S. Incremental algorithms can be regarded as the solution for the "semi-dynamic" case in which we can only insert, but not delete points. Yet another direction is to consider the convex hull computation in higher (or arbitrary) dimensions.

Exercises

Exercise 6.1: Prove the correctness of Graham Scan. ◇

Exercise 6.2: Implement the Andrew Scan. ♢

End Exercises

§7. APPENDIX: Convexity Concepts

We introduce some basic concepts of convexity in an Euclidean space $\mathbb{R}^d$ of dimension $d \geq 1$. We will mostly discuss $d = 2$ or $d = 3$.

If $p, q \in \mathbb{R}^d$ are two points, then the closed **line segment** joining them is the set of points of the form

$$\lambda p + (1 - \lambda)q$$

for $0 \leq \lambda \leq 1$. This segment is denoted $[p, q]$. When λ is unrestricted, we obtain the **line** through p and q, denoted $\overline{p, q}$. If $a_0, \ldots, a_d \in \mathbb{R}$, then the set of points $p = (p_1, \ldots, p_d) \in \mathbb{R}^d$ satisfying the linear equation $a_0 + \sum_{i=1}^d a_i p_i = 0$ is called a **hyperplane** (for $d = 2$ and $d = 3$, we may simply call it a line or a plane, respectively). A hyperplane H defines two **open half-spaces**, corresponding to those points p such that $a_0 + \sum_{i=1}^d a_i p_i > 0$ or $a_0 + \sum_{i=1}^d a_i p_i < 0$, respectively. Let these be denoted H^+ and H^-, respectively. The corresponding closed half-spaces are denoted H_0^+ and H_0^-. A set $R \subseteq \mathbb{R}^d$ of points is convex if for all $p, q \in R$, we $[p, q] \in R$. A point p is **extreme** in R if there is a hyperplane H containing p such that H^+ or H^- contains $R \setminus \{p\}$. The point is **weak extreme** if there is a hyperplane H containing p such that H_0^+ or H_0^- containes R. A **supporting hyperplane** of R is any hyperplane H through a point of R such that R lies in one of the two closed half-spaces defined by H.

We now define the **convex hull** of a set $R \subseteq \mathbb{R}^d$ to be the smallest closed convex set containing R. We denote it by $\mathsf{CHull}(R)$. A **convex polyhedron** is a convex set R with a finite number of extreme points. A convex polyhedron might be unbounded. If R is also bounded, it is called a **convex polytope**. A **polyhedron** (resp. **polytope**) is a finite union of convex polyhedras (resp. polytopes). In 2-dimensions, a polytope is call a **polygon**.

A convex polytope is completely determined by its set of extreme vertices. Thus we can regard the output of our convex hull algorithm as a representation of a convex polygon. In our problem specification, we demanded a little more: namely we want an explicit representation of the "adjacency relation" among the extreme points. We say two extreme points p, q of a convex set R are **adjacent** if there is a hyperplane H containing p, q such either H^+ or H^- contains the set $R \setminus H$. Thus, in the circular list $(p_1, p_2, \ldots, p_k)$ that is output by our planar convex hull algorithm, the point p_i is adjacent to p_{i-1} and p_{i+1} where subscript arithmetic is modulo k.

References

[1] J.-D. Boissonnat and M. Yvinec. *Algorithmic Geometry.* Cambridge University Press, 1997. Translated by Hervé Brönnimann.

[2] L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, and C. Yap. Classroom examples of robustness problems in geometric computation. *Comput. Geometry: Theory and Appl.*, 40(1):61–78, 2007.

[3] L. Kettner and E. Welzl. One sided error predicates in geometric computing. In K. Mehlhorn, editor, *Proc. 15th IFIP World Computer Congress, Fundamentals - Foundations of Computer Science*, pages 13–26, 1998.

[4] K. Mulmuley. *Computational Geometry: an Introduction through Randomized Algorithms.* Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 1994.

[5] C. K. Yap and E.-C. Chang. Issues in the metrology of geometric tolerancing. In J.-P. Laumond and M. Overmars, editors, *Algorithms for Robot Motion Planning and Manipulation*, pages 393–400, Wellesley, Massachusetts, 1997. A.K. Peters. 2nd Workshop on Algorithmic Foundations of Robotics (WAFR).