BIT manuscript No.
(will be inserted by the editor)

Simple Floating-Point Filters for the Two-Dimensional Orientation Problem

Katsuhisa Ozaki · Florian Bünger · Takeshi Ogita · Shin'ichi Oishi · Siegfried M. Rump

Received: date / Accepted: date

Abstract This paper is concerned with floating-point filters for a two dimensional orientation problem which is a basic problem in the field of computational geometry. If this problem is only approximately solved by floating-point arithmetic, then an incorrect result may be obtained due to accumulation of rounding errors. A floating-point filter can quickly guarantee the correctness of the computed result if the problem is well-conditioned. In this paper, a simple semi-static floating-point filter which handles floating-point exceptions such as overflow and underflow by only one branch is developed. In addition, an improved fully-static filter is developed.

Keywords Floating-point arithmetic · Floating-point filter · Computational geometry

Mathematics Subject Classification (2000) 65G50 · 68U05

Katsuhisa Ozaki
College of Systems Engineering and Science, Shibaura Institute of Technology, 307 Fukasaku, Minuma-ku, Saitama-shi, Saitama 337-8570, Japan, Tel.: +81-48-720-6089, E-mail: ozaki@sic.shibaura-it.ac.jp

Florian Bünger
Institute for Reliable Computing, Hamburg University of Technology, Schwarzenbergstr. 95 D-21073 Hamburg, Germany

Takeshi Ogita
School of Arts and Sciences, Tokyo Woman's Christian University, 2-6-1 Zempukuji, Suginami-ku, Tokyo 167-8585, Japan

Shin'ichi Oishi
Faculty of Science and Engineering, Waseda University, 3-4-1 Okubo, Shinjyuku-ku, Tokyo 169-8555, Japan

Siegfried M. Rump
Institute for Reliable Computing, Hamburg University of Technology, Schwarzenbergstr. 95 D-21073 Hamburg, Germany

1 Introduction

In this paper, we propose a semi-static floating-point filter for computational geometry. We focus on a two-dimensional orientation problem which is one of the basic problems in computational geometry. Suppose that an oriented line and a point $C = (c_x, c_y)$ in the two-dimensional Euclidean space are given. The oriented line passes from a point $A = (a_x, a_y)$ to a point $B = (b_x, b_y)$ for $A \neq B$. The aim is to judge whether the point C is located on the left or the right of the oriented line, or on the line. This problem can be boiled down to determining the sign of a 3-by-3 matrix determinant as follows:

$$\mathrm{sign}(\det(G)),\ G := \begin{pmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{pmatrix}. \tag{1.1}$$

If the sign of the determinant is positive / negative, then the point is left / right of the oriented line. If the sign is zero, then the point is on the line. Fig. 1.1 visualizes this problem.

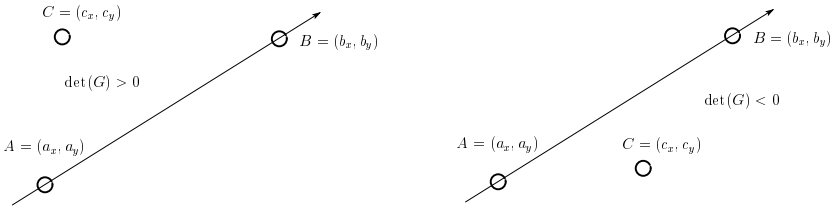

Fig. 1.1 two-dimensional orientation problem.

Solving this orientation problem is necessary for solving the following other problems[1]:

- convex hull problem: Determine a minimum convex set enclosing a given finite set of points.
- point-in-polygon problem: Decide whether a given point in the plane lies inside, outside, or on the boundary of a polygon.
- segment intersection problem: Decide whether two given line segments cross each other or not.

If all coordinates are represented by floating-point numbers and if the determinant (1.1) is evaluated by floating-point arithmetic, then an incorrect sign may be obtained due to accumulation of rounding errors. If the point is very close to the oriented line, then an incorrect sign might be obtained in the worst case. Once the incorrect sign

[1] There are further related crucial problems in computational geometry which require to determine signs of determinants of larger matrices than in (1.1). For example, the point-in-circle problem: "Decide whether a point in the plane lies inside or outside or on the boundary of a circle." leads to a 4-by-4 matrix. Such problems are not considered in this paper even though we believe that our techniques can be adapted but calculations will certainly become much more involved.

is computed, algorithms for computational geometry have prospects of producing an inexact result. Taking the convex hull problem for example, the computed result may not become a convex set, or some of the given points may not be enclosed. In the worst case algorithms may enter an infinite loop. Such problems are called 'robustness problems'. Plentiful topics of robustness problems in computational geometry are introduced in [5]. A reliable package for overcoming such problems was released in [14].

If we use multi-precision arithmetic with sufficient precision or symbolic computations, it is possible to obtain the correct sign of the determinant (1.1). However, the following problems arise:

- Basically, it is rare to handle ill-conditioned problems. Namely, many problems are exactly solved by floating-point arithmetic. However, once a result is incorrect, an algorithm may finally produce a meaningless result.
- The cost for using multi-precision arithmetic or symbolic computations is very expensive. It is desired to obtain the correct result by an arithmetic with minimum precision. However, we cannot know in advance how many bits the algorithm requires in order to produce the correct result.

To overcome these problems, it is preferred that so-called 'floating-point filters' are applied first. Such a filter quickly checks whether a sufficient condition for the correctness of the sign of the determinant is satisfied or not. Therefore, the filter answers 'the sign of the computed result is correct' or 'the correctness of the computed result is unknown'. If the filter cannot guarantee correctness of the computed sign, more accurate algorithms can be applied. Therefore, it is possible to develop an 'adaptive algorithm' which does as much work as possible to guarantee the sign of the determinant. There are several kinds of filters, static filters [3], semi-static filters [12, 3], dynamic filters [12, 2, 8]. Rigorous rounding error analysis can quickly become a very tedious task even for short and quite simple-looking arithmetic expressions like (1.1). For that reason automatic generators of floating-point filters were developed in [7]. For implementation of geometric algorithms a generic C++ design to perform exact geometric computations was developed in [9]. In this paper, we do not treat integer data but floating-point data, so that we will discuss a floating-point filter.

The article is organized as follows: In Section 2, we briefly review earlier works on semi-static floating-point filters, namely Shewchuk's work and Melquiond-Pion's work. When floating-point exceptions like overflow or underflow occur, some known filters cannot work correctly, or, additional costs arise in order to handle such exceptions appropriately. In Section 3, we develop improved filters and explain their properties. As a result, even if overflow or underflow occur, our filters work correctly with only one single branch. In the final section, we discuss a static filter which produces smaller error bounds, compared to well-known approaches.

2 Notation and Filters for the Two-Dimensional Orientation Problem

In this section we introduce the notation and earlier works on floating-point filters. We assume that all coordinates are represented by floating-point numbers defined by

IEEE 754-2008 [1]. Let $\mathbb{F}$ be the set of binary32 or binary64 floating-point numbers and let $fl(\cdot)$ denote that each operation in the parenthesis is evaluated by pure floating-point arithmetic with rounding to nearest ties to even. Let $\mathbf{u}$ be the roundoff unit: $\mathbf{u} = 2^{-24}$ for binary32 and $\mathbf{u} = 2^{-53}$ for binary64. Let $\mathbf{u}_N$ be the smallest positive normalized floating-point number, for example, $\mathbf{u}_N = 2^{-1022}$ for binary64. Let $\mathbf{u}_S$ be the smallest positive floating-point number, for example, $\mathbf{u}_S = 2^{-1074}$ for binary64 which is a subnormal number.

2.1 Shewchuk's Filter

We introduce the filter coded in [13]. In specifications of algorithms, $+, -, *$ are evaluated by floating-point arithmetic.

Algorithm 1 *Let $A = (a_x, a_y)$, $B = (b_x, b_y)$ and $C = (c_x, c_y)$ be three points in the two-dimensional space with floating-point coordinates. Suppose that an oriented line passes from A to B. The following algorithm outputs a sign of the determinant (1.1) by floating-point arithmetic. If the result is positive (negative), the point C is left (right) of the oriented line.*

```
function det = Sfilter(a_x, a_y, b_x, b_y, c_x, c_y)
  detleft = (a_x - c_x) * (b_y - c_y);
  detright = (a_y - c_y) * (b_x - c_x);
  det = detleft - detright;
  if (detleft > 0.0)
    if (detright <= 0.0)
      return det;
    else
      detsum = detleft + detright;
    end
  else if (detleft < 0.0)
    if (detright >= 0.0)
      return det;
    else
      detsum = -detleft - detright;
    end
  else
    return det;
  end
  errbound = (3 * u + 16 * u * u) * detsum;
  if ((det >= errbound) || (-det >= errbound))
    return det;
  end
  % fall down to robust computations
end
```

This algorithm returns the result with two or three branches. The key point is as follows: The determinant (1.1) is approximated by

$$det := fl((a_x - c_x)(b_y - c_y) - (a_y - c_y)(b_x - c_x)). \tag{2.1}$$

The signs of $fl((a_x - c_x)(b_y - c_y))$ and $fl((a_y - c_y)(b_x - c_x))$ are correct, meaning that they are equal to the signs of $(a_x - c_x)(b_y - c_y)$ and $(a_y - c_y)(b_x - c_x)$, respectively, except for the case that underflow occurs during the computation. Therefore, the filter first checks the sign of the terms $detleft := fl((a_x - c_x)(b_y - c_y))$ and $detright := fl((a_y - c_y)(b_x - c_x))$. If the signs are opposite, then the correctness of the sign of (2.1) is guaranteed. Otherwise, an a priori error bound of the determinant is computed:

$$errbound := fl((3\mathbf{u} + 16\mathbf{u}^2)(|(a_x - c_x)(b_y - c_y)| + |(a_y - c_y)(b_x - c_x)|)). \tag{2.2}$$

If $|det| \geq errbound$ is satisfied, then the computed sign of the determinant is correct. Otherwise, the specific problem requires a more accurate evaluation. See [12] for the derivation of (2.2) and for developing adaptive algorithms.

We state advantages and disadvantages for this filter: If the result is guaranteed by the first check of the signs of $detleft$ and $detright$ in Algorithm 1, then the additional costs for the verification are very cheap. If overflow occurs during the computation, then the sign of the determinant can only sometimes be guaranteed, for example, in case of $detleft = \pm\text{Inf}$ and $detright = \mp\text{Inf}$. If $detleft$ becomes 0 because of underflow, then this filter guarantees the sign of the computed result. However, this result is incorrect for the following example represented in binary64:

$$(a_x, a_y) := (2^{-702}, 2^{-701}),\ (b_x, b_y) := (2^{-700}, 2^{-700}),\ (c_x, c_y) := (2^{-699}, 2^{-700}).$$

We obtain $detleft = 0$ and $det = 0$ due to occurrence of underflow, so that the algorithm gives that the result is 0. However, the true determinant is 2^{-1401}.

2.2 Melquiond and Pion's Filter

We introduce the floating-point filter by Melquiond and Pion [6][2]. First, we write down their algorithm as follows:

Algorithm 2 *Let $A = (a_x, a_y)$, $B = (b_x, b_y)$ and $C = (c_x, c_y)$ be three points in the two-dimensional space. All coordinates are represented in binary64. Suppose that an oriented line passes from A to B. The following algorithm outputs a sign of the determinant (1.1) by floating-point arithmetic. If the result is positive (negative), the*

[2] We introduce the algorithm given in [6]. The main purpose of the paper by Melquiond and Pion is to develop an automatic constructor of the floating-point filter. Therefore, their approach can be applied not only to the two dimensional orientation problem but also to other problems in computational geometry.

point C is left (right) of the oriented line.

```
function det = MPfilter(a_x, a_y, b_x, b_y, c_x, c_y)
  pqx = b_x − a_x;
  pqy = b_y − a_y;
  prx = c_x − a_x;
  pry = c_y − a_y;
  maxx = max(abs(pqx), abs(prx));
  maxy = max(abs(pqy), abs(pry));
  eps = 8.887205737259275 8e − 16 ∗ maxx ∗ maxy;
  if (maxx > maxy) swap(maxx, maxy);
  if (maxx < 1e − 146)
    if (maxx == 0)
      det = 0;
      return det;
    end
  else if (maxy < 1e153)
    det = pqx ∗ pry − pqy ∗ prx;
    if (det > eps) return det;
    if (det < −eps) return det;
  end
  % fall back to a more precise, slower method
end
```

If heavy cancellation occurs, the filter requires four branches.

This algorithm checks the possibility of an occurrence of underflow or overflow in turn. If the potential for a floating-point exception is found, then the algorithm uses more robust computations in order to prevent overflow and underflow. Next, the algorithm checks whether heavy cancellation occurs or not (see Figure 2.1 to understand the flow of the algorithm). Therefore, the filter rigorously works for all floating-point data.

However, it is rare to find overflow or underflow in practice in the two-dimensional orientation problem[3]. In addition, let us consider the following setting

$$(a_x, a_y) := (-1, -1),\ (b_x, b_y) := (1, 1),\ (c_x, c_y) := (2^{700}, 1).$$

For this example, the filter says that a robust algorithm is necessary because $maxy = 2^{700} > 10^{153}$. However, overflow does not occur in the evaluation of the determinant and this problem is not ill-conditioned. Namely, the floating-point result is correct. Next, consider

$$(a_x, a_y) := (1, 2^{-600}),\ (b_x, b_y) := (2^{-600}, 2^{-600}),\ (c_x, c_y) := (0, 0).$$

The filter fails to verify correctness due to $maxx = 2^{-600} < 1e - 146$ although the correct sign is obtained by floating-point arithmetic.

[3] Since there are products of only two floating-point numbers in the evaluation of (1.1), overflow and underflow rarely occur. However, if we handle the InSphere problem which aims to judge whether a point is inside or outside a sphere in the three-dimensional space, then a product of five floating-point numbers appears in the evaluation. Therefore, it is not so rare that floating-point exceptions occur in this problem.

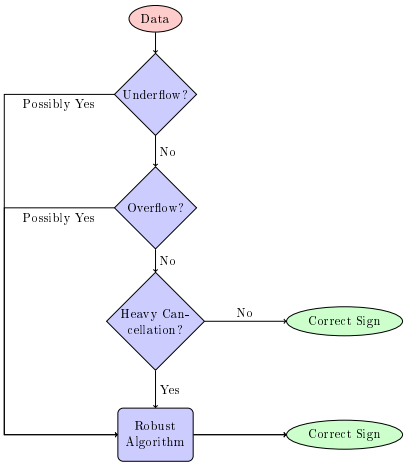

Fig. 2.1 Flow of the algorithm in the paper of Melquiond and Pion [6].

3 Proposed Semi-Static Filters

We will construct a floating-point filter based on (2.1). The evaluation of (2.1) requires 7 floating-point operations:

$$\begin{aligned}
t_1 &:= a_x - c_x, & x_1 &:= \mathit{fl}(a_x - c_x),\\
t_2 &:= b_y - c_y, & x_2 &:= \mathit{fl}(b_y - c_y),\\
t_3 &:= a_y - c_y, & x_3 &:= \mathit{fl}(a_y - c_y),\\
t_4 &:= b_x - c_x, & x_4 &:= \mathit{fl}(b_x - c_x),\\
t_5 &:= t_1 t_2, & x_5 &:= \mathit{fl}(x_1 x_2),\\
t_6 &:= t_3 t_4, & x_6 &:= \mathit{fl}(x_3 x_4),\\
t_7 &:= t_5 - t_6, & x_7 &:= \mathit{fl}(x_5 - x_6).
\end{aligned}$$

In the final line, t_7 and x_7 denote the true determinant and a floating-point approximate, respectively. We recall well-known error analysis for floating-point arithmetic. For $x, y \in \mathbb{F}$, the definition of the IEEE 754 standard yields

$$\mathit{fl}(x \circ y) = (1+\varepsilon)(x \circ y), \quad x \circ y = (1+\varepsilon)\mathit{fl}(x \circ y), \quad |\varepsilon| \le \mathbf{u}, \quad \circ \in \{+,-\}. \tag{3.1}$$

For a product,

$$xy = \mathit{fl}(xy)(1+\varepsilon) + \eta, \quad |\varepsilon| \le \mathbf{u}, \quad |\eta| \le \frac{1}{2}\mathbf{u}_S, \quad \varepsilon\eta = 0. \tag{3.2}$$

In [10], Rump introduced new models for the error of floating-point arithmetic:

$$\mathit{fl}(x \circ y) = x \circ y + \delta, \quad |\delta| \le \mathbf{u} \cdot \mathtt{ufp}(x \circ y) \le \mathbf{u} \cdot \mathtt{ufp}(\mathit{fl}(x \circ y)), \quad \circ \in \{+,-\} \tag{3.3}$$

and

$$xy = fl(xy) + \delta + \eta, \quad |\delta| \le \mathbf{u} \cdot \mathtt{ufp}(xy) \le \mathbf{u} \cdot \mathtt{ufp}(fl(xy)), \quad |\eta| \le \frac{1}{2}\mathbf{u}_S, \quad \delta\eta = 0 \tag{3.4}$$

where

$$\mathtt{ufp}(a) := \begin{cases} 0, & \text{if } a = 0, \\ 2^{\lfloor \log_2 |a| \rfloor}, & \text{else.} \end{cases}$$

The relations (3.1), (3.2), (3.3) and (3.4) are only valid if no overflow occurs. In the subsequent Theorems 3.1, 3.2, 3.3, 3.4 and in Lemma 3.1 we will therefore assume that no overflow occurs in $x_1, \ldots, x_7$ and $x_8 := fl(|x_5| + |x_6|)$.

Lemma 3.1 *For $a, b, c, d \in \mathbb{F}$,*

$$|fl((a+b)(c+d)) - (a+b)(c+d)| \le (3\mathbf{u} - (\phi - 14)\mathbf{u}^2) fl(|(a+b)(c+d)|) + S$$

where

$$\phi := 2 \left\lfloor \frac{-1 + \sqrt{4\mathbf{u}^{-1} + 45}}{4} \right\rfloor, \tag{3.5}$$

and

$$0 \le S < \frac{1}{2}\mathbf{u}_S + \frac{3}{2}\mathbf{u} \cdot \mathbf{u}_S \quad \textit{and} \quad S = 0 \textit{ if } |fl(a+b)fl(c+d)| \ge \mathbf{u}_N.$$

In binary64, $\phi = 94906264$ and the obtained constant is $3\mathbf{u} - 94906250\mathbf{u}^2 \approx 2.99999998\mathbf{u}$.

Proof.

If $a+b = 0$ or $c+d = 0$, then (3.5) trivially holds. Therefore, we can assume both $a+b \neq 0$ and $c+d \neq 0$. From (3.3) and (3.4),

$$s_1 := a+b, \quad \tilde{s}_1 := fl(a+b) = s_1(1+\varepsilon_1), \quad |\varepsilon_1| \le \mathbf{u}\frac{\mathtt{ufp}(s_1)}{|s_1|} \le \mathbf{u}, \tag{3.6}$$

$$s_2 := c+d, \quad \tilde{s}_2 := fl(c+d) = s_2(1+\varepsilon_2), \quad |\varepsilon_2| \le \mathbf{u}\frac{\mathtt{ufp}(s_2)}{|s_2|} \le \mathbf{u}, \tag{3.7}$$

$$\begin{aligned} &p = \tilde{s}_1\tilde{s}_2, \quad \tilde{p} = fl(\tilde{s}_1\tilde{s}_2) = p(1+\varepsilon_3) + \eta, \\ &|\varepsilon_3| \le \mathbf{u}\tfrac{\mathtt{ufp}(p)}{|p|} \le \mathbf{u}, \quad |\eta| \le \tfrac{1}{2}\mathbf{u}_S, \quad \varepsilon_3\eta = 0. \end{aligned} \tag{3.8}$$

Then,

$$\tilde{p} = s_1s_2(1+\varepsilon_1)(1+\varepsilon_2)(1+\varepsilon_3) + \eta =: s_1s_2(1+\alpha) + \eta$$

and it derives

$$\tilde{p} - s_1s_2 = \alpha s_1s_2 + \eta = \frac{\alpha}{1+\alpha}(\tilde{p} - \eta) + \eta, \tag{3.9}$$

where

$$|\alpha| \le (1+|\varepsilon_1|)(1+|\varepsilon_2|)(1+|\varepsilon_3|) - 1. \tag{3.10}$$

We distinguish two cases. First, assume $|s_i| \ge (1+\phi\mathbf{u})\mathtt{ufp}(s_i)$ for some $i \in \{1, 2\}$. Then, from (3.6) and (3.7), at least one of $|\varepsilon_1|$ or $|\varepsilon_2|$ is smaller or equal to $\dfrac{\mathbf{u}}{1+\phi\mathbf{u}}$ and we take an upper bound for α in (3.10)

$$|\alpha| \le (1 + \frac{\mathbf{u}}{1+\phi\mathbf{u}})(1+\mathbf{u})^2 - 1.$$

Hence, from $\phi \le \dfrac{1+\sqrt{4\mathbf{u}^{-1}+1}}{2}$,

$$\begin{aligned}|\alpha| &\le (1+\frac{\mathbf{u}}{1+\phi\mathbf{u}})(1+\mathbf{u})^2-1 \le (1+(1-(\phi-1)\mathbf{u})\mathbf{u})(1+\mathbf{u})^2-1\\ &= (1-(\phi-1)\mathbf{u})(1+\mathbf{u})^2\mathbf{u}+2\mathbf{u}+\mathbf{u}^2\\ &= 3\mathbf{u}+(4-\phi)\mathbf{u}^2+(3-2\phi)\mathbf{u}^3+(1-\phi)\mathbf{u}^4 =: t.\end{aligned}$$

Therefore,

$$|\frac{\alpha}{1+\alpha}| \le \frac{|\alpha|}{1-|\alpha|} \le \frac{t}{1-t} < 3\mathbf{u}-(\phi-13)\mathbf{u}^2. \tag{3.11}$$

Finally, (3.9) and (3.11) yield

$$|\tilde{p}-s_1s_2| = |\frac{\alpha}{1+\alpha}(\tilde{p}-\eta)+\eta| \le (3\mathbf{u}-(\phi-13)\mathbf{u}^2)|\tilde{p}|+S$$

where

$$0 \le S < \frac{1}{2}\mathbf{u}_S+\frac{3}{2}\mathbf{u}\cdot\mathbf{u}_S \text{ and } S=0 \text{ if } |p| \ge \mathbf{u}_N.$$

It remains the second case that $|s_1| < (1+\phi\mathbf{u})\mathtt{ufp}(s_1)$ and $|s_2| < (1+\phi\mathbf{u})\mathtt{ufp}(s_2)$. From (3.8),

$$|\varepsilon_3| = \frac{|\tilde{p}-\tilde{s}_1\tilde{s}_2-\eta|}{|\tilde{s}_1\tilde{s}_2|}. \tag{3.12}$$

From the assumption in the second case,

$$|\tilde{s}_i| = (1+k_i\mathbf{u})\mathtt{ufp}(s_i) \in \mathbb{F}, \quad i=1,2, \quad 0 \le k_i \le \phi, \quad k_i \text{ is even.}$$

We have

$$\begin{aligned}|\tilde{s}_1\tilde{s}_2| &= (1+k_1\mathbf{u})\mathtt{ufp}(s_1)\cdot(1+k_2\mathbf{u})\mathtt{ufp}(s_2)\\ &= (1+(k_1+k_2)\mathbf{u}+k_1k_2\mathbf{u}^2)\mathtt{ufp}(s_1)\mathtt{ufp}(s_2).\end{aligned} \tag{3.13}$$

If $|\tilde{p}| \ge \mathbf{u}_N$, then $\tilde{p} = (1+(k_1+k_2)\mathbf{u})\mathtt{ufp}(s_1)\mathtt{ufp}(s_2)$, so that from $\phi^2 < \mathbf{u}^{-1}$,

$$|\tilde{p}-\tilde{s}_1\tilde{s}_2| = k_1k_2\mathbf{u}^2\mathtt{ufp}(s_1)\mathtt{ufp}(s_2), \tag{3.14}$$

and we can set $\eta = 0$ in (3.12). Then, by substituting (3.13) and (3.14) to (3.12),

$$|\varepsilon_3| = \frac{k_1k_2\mathbf{u}^2}{1+(k_1+k_2)\mathbf{u}+k_1k_2\mathbf{u}^2} \le \phi^2\mathbf{u}^2.$$

If $|\tilde{p}| < \mathbf{u}_N$, then we can set $\varepsilon_3 = 0$ in (3.10). Therefore, in any case,

$$|\alpha| \le (1+\mathbf{u})^2(1+\phi^2\mathbf{u}^2)-1 < 2\mathbf{u}+(3+\phi^2)\mathbf{u}^2 = 3\mathbf{u}-(\mathbf{u}^{-1}-3-\phi^2)\mathbf{u}^2.$$

Since $-(\mathbf{u}^{-1}-3-\phi^2)\mathbf{u}^2 < (4-\phi)\mathbf{u}^2+(3-2\phi)\mathbf{u}^3+(1-\phi)\mathbf{u}^4$, we derive (3.5) in the same way as in the case of $|s_i| \ge (1+\phi\mathbf{u})\mathtt{ufp}(s_i)$ for some i. □

Remark 3.1 We do not claim that the constant $(3\mathbf{u}-(\phi-14)\mathbf{u}^2)$ in (3.5) is optimal. However, if $a=c=1$ and $b=d=94906265\mathbf{u}$ in binary64, then

$$|fl((a+b)(c+d))-(a+b)(c+d)| = (2.999999923..)fl(|(a+b)(c+d)|).$$

Therefore, the constant cannot be that far from being optimal.

Theorem 3.1 *If*

$$x_7 = fl(x_5 - x_6) = x_5 - x_6, \tag{3.15}$$

which means that no rounding error occurs in $fl(x_5-x_6)$, then a sufficient condition for the correctness of the sign of x_7 is

$$|x_7| \geq fl(\theta x_8) + (1.5+3\mathbf{u})\mathbf{u}_S \tag{3.16}$$

where $\theta := 3\mathbf{u}-(\phi-22)\mathbf{u}^2 \in \mathbb{F}$ and ϕ is defined by (3.5).

Proof.

From Lemma 3.1,

$$t_5 = x_5+\delta_5+\eta_5, \quad |\delta_5| \leq (3\mathbf{u}-(\phi-14)\mathbf{u}^2)|x_5|, \quad |\eta_5| < \frac{1}{2}\mathbf{u}_S + \frac{3}{2}\mathbf{u}\cdot\mathbf{u}_S. \tag{3.17}$$

Similarly, we have

$$t_6 = x_6+\delta_6+\eta_6, \quad |\delta_6| \leq (3\mathbf{u}-(\phi-14)\mathbf{u}^2)|x_6|, \quad |\eta_6| < \frac{1}{2}\mathbf{u}_S + \frac{3}{2}\mathbf{u}\cdot\mathbf{u}_S. \tag{3.18}$$

From assumption (3.15), (3.17) and (3.18),

$$t_7 = t_5 - t_6 = x_5+\delta_5+\eta_5-(x_6+\delta_6+\eta_6) = x_7+\delta_5+\eta_5-(\delta_6+\eta_6).$$

Therefore, if

$$|x_7| > |\delta_5+\eta_5-(\delta_6+\eta_6)|, \tag{3.19}$$

then the sign of x_7 is the same as that of t_7. From (3.1),

$$|x_5|+|x_6| \leq (1+\mathbf{u})x_8. \tag{3.20}$$

From (3.20), we develop an upper bound of the right hand-side of (3.19):

$$\begin{aligned}
&|\delta_5+\eta_5-(\delta_6+\eta_6)| < (3\mathbf{u}-(\phi-14)\mathbf{u}^2)(|x_5|+|x_6|)+\mathbf{u}_S+3\mathbf{u}\cdot\mathbf{u}_S \\
&\leq (3\mathbf{u}-(\phi-17)\mathbf{u}^2-(\phi-14)\mathbf{u}^3)x_8+\mathbf{u}_S+3\mathbf{u}\cdot\mathbf{u}_S \\
&= (1+\mathbf{u})^{-1}(3\mathbf{u}-(\phi-20)\mathbf{u}^2-(2\phi-31)\mathbf{u}^3-(\phi-14)\mathbf{u}^4)x_8+\mathbf{u}_S+3\mathbf{u}\cdot\mathbf{u}_S =: U_1.
\end{aligned}$$

In the above expression,

$$\mathbb{F} \not\ni 3\mathbf{u}-(\phi-20)\mathbf{u}^2-(2\phi-31)\mathbf{u}^3-(\phi-14)\mathbf{u}^4 < 3\mathbf{u}-(\phi-22)\mathbf{u}^2 = \theta \in \mathbb{F}.$$

Therefore, from (3.2)

$$\begin{aligned}
U_1 &\leq (1+\mathbf{u})^{-1}\theta x_8+\mathbf{u}_S+3\mathbf{u}\cdot\mathbf{u}_S \\
&\leq (1+\mathbf{u})^{-1}(1+\mathbf{u})fl(\theta x_8)+1.5\mathbf{u}_S+3\mathbf{u}\cdot\mathbf{u}_S \\
&= fl(\theta x_8)+1.5\mathbf{u}_S+3\mathbf{u}\cdot\mathbf{u}_S.
\end{aligned}$$

Thus, (3.16) is obtained. □

In Theorem 3.1, we assumed (3.15). However, this assumption is not important by the following theorem.

Theorem 3.2 *If a rounding error occurs in $fl(x_5 - x_6)$ such that*

$$x_7 = fl(x_5 - x_6) \neq x_5 - x_6, \tag{3.21}$$

then the sign of x_7 is the same as that of t_7.

Proof.

If both x_5 and x_6 are subnormal numbers, then no rounding error occurs in $fl(x_5 - x_6)$. Therefore, the assumption (3.21) implies that either x_5 or x_6 is a normalized floating-point number, so that

$$\max(|x_5|, |x_6|) \geq \mathbf{u}_N > \frac{1}{2}\mathbf{u}_N. \tag{3.22}$$

From (3.3),

$$x_5 - x_6 = fl(x_5 - x_6) + \delta, \quad |\delta| \leq \mathbf{u} \cdot \mathtt{ufp}(x_5 - x_6) \leq \mathbf{u} \cdot \mathtt{ufp}(fl(x_5 - x_6)). \tag{3.23}$$

Then, (3.19) is replaced by

$$|x_7| > |\delta_5 + \eta_5 - (\delta_6 + \eta_6) + \delta|, \tag{3.24}$$

and if

$$|x_7| \geq fl(\theta x_8) + 1.5\mathbf{u}_S + 3\mathbf{u} \cdot \mathbf{u}_S + \mathbf{u} \cdot \mathtt{ufp}(fl(x_5 - x_6))$$

is satisfied, then the sign of x_7 is correct. For nonnegative $x, y \in \mathbb{F}$, Sterbenz's theorem [4] says that

$$\frac{y}{2} \leq x \leq 2y \Longrightarrow fl(x - y) = x - y.$$

Thus, the contraposition of Sterbenz's theorem for nonnegative x_5 and x_6 means

$$x_5 < x_6/2 \text{ or } x_5 > 2x_6$$

which gives

$$|x_7| \geq \frac{1}{2}\max(|x_5|, |x_6|). \tag{3.25}$$

For negative x_5 and x_6 we can prove (3.25) similarly. If x_5 and x_6 have opposite signs, then (3.25) is trivially satisfied. An upper bound of the right term in (3.24) is

$$\begin{aligned} &fl(\theta x_8) + (1.5 + 3\mathbf{u})\mathbf{u}_S + \mathbf{u} \cdot \mathtt{ufp}(fl(x_5 - x_6)) \\ &\leq fl(4\mathbf{u} \cdot 2\max(|x_5|, |x_6|)) + (1.5 + 3\mathbf{u})\mathbf{u}_S + \mathbf{u} \cdot \mathtt{ufp}(fl(x_5 - x_6)). \end{aligned}$$

Therefore, from (3.22), (3.23) and (3.25), it follows that

$$\begin{aligned}
&|x_7| - (fl(\theta x_8) + (1.5+3\mathbf{u})\mathbf{u}_S) - \mathbf{u}\cdot \mathtt{ufp}(fl(x_5 - x_6)) \\
&\geq \frac{1}{2}\max(|x_5|,|x_6|) - fl(4\mathbf{u}\cdot 2\max(|x_5|,|x_6|)) - (1.5+3\mathbf{u})\mathbf{u}_S - 2\mathbf{u}\max(|x_5|,|x_6|) \\
&> \frac{1}{2}\max(|x_5|,|x_6|) - 10\mathbf{u}\max(|x_5|,|x_6|) - \frac{1}{2}\mathbf{u}_S - 2\mathbf{u}_S \\
&\geq \left(\frac{1}{2} - 10\mathbf{u}\right)\frac{1}{2}\mathbf{u}_N - 2.5\mathbf{u}_S > 0.
\end{aligned}$$

□

The following floating-point filter is proposed in [3]:

$$|x_7| > fl(8\mathbf{u}(((|a_x|+|c_x|)(|b_y|+|c_y|)+\mathbf{u}_N) + (|b_x|+|c_x|)(|a_y|+|c_y|)+\mathbf{u}_N)).$$

The idea of adding $\mathbf{u}_N$ to a factor of a floating-point multiplication is nice because in this way the subnormal constant $(1.5+3\mathbf{u})\mathbf{u}_S$ appearing in (3.16) can be avoided which would cause slow performance on the CPU in the average case. We take up this idea and propose the following floating-point filter:

Theorem 3.3 *If*

$$|x_7| > fl(\theta(x_8+\mathbf{u}_N)) \tag{3.26}$$

is satisfied, then the sign of x_7 is the same as that of t_7.

Proof.

According to Theorem 3.2, we may assume that (3.15) holds true. First, we mention a basic property of floating-point numbers, see [11]. For $x \in \mathbb{R}$ let $\mathtt{succ}(x) := \min\{f \in \mathbb{F} \mid x < f\}$ denote the floating-point successor of x. Then, for nonnegative $c \in \mathbb{F}$ with finite $\mathtt{succ}(c)$ the following holds true:

$$\text{If } c < \mathbf{u}^{-1}\mathbf{u}_S = 2\mathbf{u}_N, \text{ then } \mathtt{succ}(c) = c + \mathbf{u}_S. \tag{3.27}$$

$$\text{If } 2\mathbf{u}_N \leq c, \text{ then } \mathtt{succ}(c) = c + 2\mathbf{u}\cdot\mathtt{ufp}(c). \tag{3.28}$$

We distinguish the following three cases:

(a) $x_8 \geq \mathbf{u}^{-1}\mathbf{u}_N - \mathbf{u}_N$
(b) $\mathbf{u}^{-1}\mathbf{u}_N - \mathbf{u}_N > x_8 \geq \frac{1}{2}\mathbf{u}^{-1}\mathbf{u}_N - \mathbf{u}_N$
(c) $x_8 < \frac{1}{2}\mathbf{u}^{-1}\mathbf{u}_N - \mathbf{u}_N$

In case (a), $fl(\theta(x_8+\mathbf{u}_N)) > 2\mathbf{u}_N$, (3.28) and assumption (3.26) imply

$$\begin{aligned}
|x_7| &\geq \mathtt{succ}(fl(\theta(x_8+\mathbf{u}_N))) = fl(\theta(x_8+\mathbf{u}_N)) + 2\mathbf{u}\cdot\mathtt{ufp}(fl(\theta(x_8+\mathbf{u}_N))) \\
&\geq fl(\theta(x_8+\mathbf{u}_N)) + 4\mathbf{u}\cdot\mathbf{u}_N \geq fl(\theta x_8) + 2\mathbf{u}_S > fl(\theta x_8) + (1.5+3\mathbf{u})\mathbf{u}_S.
\end{aligned}$$

The assertion follows from Theorem 3.1.

For case (b), from (3.4)

$$x_1x_2 = fl(x_1x_2) + \delta_3 + \eta_3, \quad |\delta_3| \leq \mathbf{u}\cdot\mathtt{ufp}(x_1x_2), \quad |\eta_3| \leq \frac{1}{2}\mathbf{u}_S, \quad \delta_3\eta_3 = 0,$$

$$x_3x_4 = fl(x_3x_4) + \delta_4 + \eta_4, \quad |\delta_4| \leq \mathbf{u}\cdot\mathtt{ufp}(x_3x_4), \quad |\eta_4| \leq \frac{1}{2}\mathbf{u}_S, \quad \delta_4\eta_4 = 0.$$

By assumption

$$\begin{aligned}(2+\mathbf{u})(\mathtt{ufp}(x_1x_2)+\mathtt{ufp}(x_3x_4))+\mathbf{u}_s &\geq |x_1x_2|+|x_3x_4|+|\delta_3|+|\delta_4|+|\eta_3|+|\eta_4| \\ &\geq |x_5|+|x_6| \geq x_8-\mathbf{u}\cdot\mathtt{ufp}(x_8) \\ &> \frac{1}{2}\mathbf{u}^{-1}\mathbf{u}_N-\mathbf{u}_N-\mathbf{u}(\mathbf{u}^{-1}\mathbf{u}_N) \\ &= (\frac{1}{2}\mathbf{u}^{-1}-2)\mathbf{u}_N\end{aligned}$$

which implies

$$\mathtt{ufp}(x_1x_2)+\mathtt{ufp}(x_3x_4) > \frac{(\frac{1}{2}\mathbf{u}^{-1}-2)\mathbf{u}_N-\mathbf{u}_S}{2+\mathbf{u}} \gg 2\mathbf{u}_N.$$

Thus,

$$\mathbf{u}\cdot\mathtt{ufp}(x_1x_2) > \mathbf{u}\cdot\mathbf{u}_N = \frac{1}{2}\mathbf{u}_S \quad \text{or} \quad \mathbf{u}\cdot\mathtt{ufp}(x_3x_4) > \mathbf{u}\cdot\mathbf{u}_N = \frac{1}{2}\mathbf{u}_S.$$

Now, according to the conditions imposed on S in Lemma 3.1, we may assume without loss of generality that $\eta_5 = 0$ or $\eta_6 = 0$ in the proof of Theorem 3.1 which yields the modified constant

$$\tilde{U}_1 := (1+\mathbf{u})^{-1}\theta x_8 + \frac{1}{2}\mathbf{u}_S + \frac{3}{2}\mathbf{u}\cdot\mathbf{u}_S.$$

Furthermore,

$$\mathbf{u}\cdot\mathtt{ufp}(fl(\theta x_8)) \geq \mathbf{u}\cdot\mathtt{ufp}(fl(\theta(\frac{1}{2}\mathbf{u}^{-1}\mathbf{u}_N-\mathbf{u}_N))) \geq \mathbf{u}\cdot\mathbf{u}_N = \frac{1}{2}\mathbf{u}_S$$

implies $\theta x_8 \leq (1+\mathbf{u})fl(\theta x_8)$. Repeating the final estimation of U_1 in the proof of Theorem 3.1 gives

$$\begin{aligned}\tilde{U}_1 &= (1+\mathbf{u})^{-1}\theta x_8 + \frac{1}{2}\mathbf{u}_S + \frac{3}{2}\mathbf{u}\cdot\mathbf{u}_S \\ &\leq (1+\mathbf{u})^{-1}(1+\mathbf{u})fl(\theta x_8) + \frac{1}{2}\mathbf{u}_S + \frac{3}{2}\mathbf{u}\cdot\mathbf{u}_S = fl(\theta x_8) + \frac{1}{2}\mathbf{u}_S + \frac{3}{2}\mathbf{u}\cdot\mathbf{u}_S.\end{aligned}$$

By assumption (3.26),

$$\begin{aligned}|x_7| &\geq fl(\theta(x_8+\mathbf{u}_N))+\mathbf{u}_S \geq fl(\theta x_8)+\mathbf{u}_S \\ &> fl(\theta x_8)+\frac{1}{2}\mathbf{u}_S+\frac{3}{2}\mathbf{u}\cdot\mathbf{u}_S \geq \tilde{U}_1\end{aligned}$$

which implies the assertion according to the proof of Theorem 3.1.

In case (c) we have

$$\mathtt{ufp}(x_8+\mathbf{u}_N) \leq \frac{1}{4}\mathbf{u}^{-1}\mathbf{u}_N, \quad \mathtt{ufp}(\theta x_8) \leq \mathtt{ufp}(\theta(x_8+\mathbf{u}_N)) \leq \mathbf{u}_N.$$

Thus,

$$\begin{aligned}
fl(x_8+\mathbf{u}_N) &\geq x_8+\mathbf{u}_N-\mathbf{u}\cdot \mathtt{ufp}(x_8+\mathbf{u}_N) \geq x_8+\frac{3}{4}\mathbf{u}_N,\\
\theta x_8 &\geq fl(\theta x_8)-\max(\mathbf{u}\cdot \mathtt{ufp}(\theta x_8),\frac{1}{2}\mathbf{u}_S) \geq fl(\theta x_8)-\mathbf{u}\cdot\mathbf{u}_N,\\
fl(\theta(x_8+\mathbf{u}_N)) &= fl(\theta fl(x_8+\mathbf{u}_N))\\
&\geq \theta fl(x_8+\mathbf{u}_N)-\max(\mathbf{u}\cdot \mathtt{ufp}(\theta fl(x_8+\mathbf{u}_N)),\frac{1}{2}\mathbf{u}_S)\\
&\geq \theta(x_8+\frac{3}{4}\mathbf{u}_N)-\mathbf{u}\cdot\mathbf{u}_N = \theta x_8+\frac{3\theta-4}{4}\mathbf{u}\cdot\mathbf{u}_N\\
&\geq fl(\theta x_8)+\frac{3\theta-8}{4}\mathbf{u}\cdot\mathbf{u}_N > fl(\theta x_8).
\end{aligned}$$

Therefore,

$$fl(\theta(x_8+\mathbf{u}_N))-fl(\theta x_8)\geq \mathbf{u}_S$$

and by assumption (3.26) also

$$|x_7|-fl(\theta(x_8+\mathbf{u}_N))\geq \mathbf{u}_S$$

is obtained from (3.27). Both inequalities imply

$$\begin{aligned}
|x_7|-fl(\theta x_8)-(1.5+3\mathbf{u})\mathbf{u}_S &\geq |x_7|-fl(\theta(x_8+\mathbf{u}_N))+\mathbf{u}_S-(1.5+3\mathbf{u})\mathbf{u}_S\\
&\geq 2\mathbf{u}_S-(1.5+3\mathbf{u})\mathbf{u}_S > 0.
\end{aligned}$$

Again, the assertion follows from Theorem 3.1. □

Theorem 3.4 *If*

$$|x_7| = fl(|x_5-x_6|) > fl(\theta(|x_5+x_6|+\mathbf{u}_N)) \tag{3.29}$$

is satisfied, then

$$|x_7| = fl(|x_5-x_6|) > fl(\theta(|x_5|+|x_6|+\mathbf{u}_N)). \tag{3.30}$$

Proof. We distinguish the following four cases of the signs of x_5 and x_6:

(a) x_5=x_6=0.
(b) One of x_5 and x_6 is zero.
(c) The signs of x_5 and x_6 are non-zero and the same.
(d) The signs of x_5 and x_6 are non-zero and opposite.

For (a), (3.29) is false. For (b) and (c), $|x_5+x_6| = |x_5|+|x_6|$. For (d),

$$fl(|x_5-x_6|) = fl(|x_5|+|x_6|).$$

x	$\circ$	y	z	x	$\circ$	y	z	x	$\circ$	y	z
Inf	+	Inf	Inf	-Inf	+	-Inf	-Inf	Inf	+	-Inf	NaN
Inf	-	Inf	NaN	-Inf	-	-Inf	NaN	Inf	-	-Inf	Inf
Inf	*	Inf	Inf	-Inf	*	-Inf	Inf	Inf	*	-Inf	-Inf
Inf	+	0	Inf	Inf	+	p	Inf	Inf	+	$-p$	Inf
Inf	-	0	Inf	Inf	-	p	Inf	Inf	-	$-p$	Inf
Inf	*	0	NaN	Inf	*	p	Inf	Inf	*	$-p$	-Inf

Table 3.1 Floating-point exceptions ($z = fl(x \circ y),\ \circ \in \{+,-,*\}$). p is a positive floating-point number.

From assumption (3.29), $|x_7| \geq 2\mathbf{u}_S$ since $fl(\theta(|x_5+x_6|+\mathbf{u}_N)) \geq \mathbf{u}_S$. If $|x_7| < 8\mathbf{u}_S$, then $fl(\theta(|x_5|+|x_6|+\mathbf{u}_N)) = \mathbf{u}_S$ and (3.30) is satisfied. If $|x_7| \geq 8\mathbf{u}_S = 16\mathbf{u}\cdot\mathbf{u}_N$, then

$$\begin{aligned}
|x_7| - fl(\theta(|x_5|+|x_6|+\mathbf{u}_N)) &\geq |x_7| - fl(3\mathbf{u}(|x_7|+\mathbf{u}_N)) \\
&\geq |x_7| - 3\mathbf{u}(1+\mathbf{u})^2(|x_7|+\mathbf{u}_N) - \frac{1}{2}\mathbf{u}_S \\
&> |x_7| - 4\mathbf{u}\cdot\frac{2|x_7|}{16\mathbf{u}} - \frac{1}{2}\mathbf{u}_S \\
&= \frac{1}{2}|x_7| - \frac{1}{2}\mathbf{u}_S > 0.
\end{aligned}$$

This completes the proof. □

Theorem 3.5 *Assume that overflow occurs in (3.29). If (3.29) is valid* [4]*, then the sign of x_7 is the same as that of t_7. Therefore, the filter (3.29) never fails, i.e., there is no case where the sign of the computed result is wrong but the filter guarantees its correctness.*

Proof.

First, we introduce the IEEE 754 rules for overflow arithmetic. For $x,y \in \mathbb{F}$ with $\{x,y\} \cap \{\pm\text{Inf}\} \neq \emptyset$, the results $z := x \circ y$ for $\circ \in \{+,-,*\}$ are shown in Table 3.1. In addition, the standard yields $fl(|\text{NaN}|) = \text{NaN}$, $fl(|\text{Inf}|) = \text{Inf}$ and $fl(|-\text{Inf}|) = \text{Inf}$. If overflow occurs at least in one of x_1,x_2,x_3,x_4,x_5,x_6, then $|x_7|$ becomes Inf or NaN and the error bound on the right hand-side in (3.29) becomes Inf or NaN. By the definition of the IEEE 754 standard, all of the following comparisons are false:

$$\text{Inf} > \text{Inf}, \quad \text{NaN} > \text{Inf}, \quad \text{Inf} > \text{NaN}, \quad \text{NaN} > \text{NaN}.$$

Therefore, we do not need to consider these cases. If overflow occurs in the evaluation of x_8, then (3.29) is invalid since $fl(\theta(|x_5+x_6|+\mathbf{u}_N)) = \text{Inf}$. Therefore, we only need to consider the case that overflow occurs in the evaluation of x_7 and does not occur in the evaluation of $x_1,\ldots,x_6$ and $fl(x_5+x_6)$. This is only possible if the signs of x_5 and x_6 are opposite, whereas clearly the sign of $x_7 = fl(x_5-x_6) \in \{\pm\text{Inf}\}$ equals the sign of $t_7 = t_5 - t_6$. □

By Theorems 3.1, 3.2, 3.3, 3.4 and 3.5 we conclude the following: if

$$|x_7| > fl(\theta(|x_5+x_6|+\mathbf{u}_N)) \tag{3.31}$$

[4] Here "valid" means that condition (3.29) evaluated according to IEEE 754 rules yields the result "true".

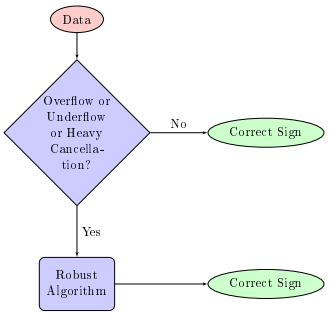

Fig. 3.1 Flow of our algorithm.

is satisfied, then the sign of x_7 is correct.

We write the algorithm for (3.31) as follows:

Algorithm 3 *Let $A = (a_x, a_y)$, $B = (b_x, b_y)$ and $C = (c_x, c_y)$ be three points in the two-dimensional space with floating-point coordinates. Suppose that an oriented line passes from A to B. The following algorithm calculates a sign of the determinant (1.1) by floating-point arithmetic. If the result is positive (negative), the point C is left (right) of the oriented line from A to B.*

function $det = \mathrm{Ofilter}(a_x, a_y, b_x, b_y, c_x, c_y)$
  $l = (a_x - c_x) * (b_y - c_y)$;
  $r = (b_x - c_x) * (a_y - c_y)$;
  $det = l - r$;
  $errbound = \theta * (|l + r| + \mathbf{u}_N)$;
  `if` $|det| > errbound$
    `return` det;
  `end`
  % fall back to a more precise, slower method
end

Figure 3 shows the flow diagram of our algorithm. We discuss advantages and disadvantages of our algorithm. Our filter contains only one branch, so that it is very simple. The number of taking absolute values is also less than that of Algorithms 1 and 2. However, Algorithm 3 does not detect points on the line, for example, if $A = (1,1),\ B = (2,1),\ C = (3,1)$, then our filters cannot guarantee the result but Algorithms 1 and 2 can verify the correctness.

We will now show benchmarks for each filter described in Sections 2 and 3. These filters are embedded in the incremental algorithm for a convex hull. We compare computing times for

- M1: Incremental Algorithm with pure floating-point arithmetic.
- M2: Incremental Algorithm with Filter-Algorithm 1
- M3: Incremental Algorithm with Filter-Algorithm 2
- M4: Incremental Algorithm with Filter-Algorithm 3

Table 3.2 Computing time for the incremental algorithm.

Algorithm	Intel C	Visual C
M1	7.37 (1.00)	7.76 (1.00)
M2	9.58 (1.30)	13.6 (1.75)
M3	13.4 (1.82)	17.7 (2.28)
M4	7.74 (1.05)	7.97 (1.03)

All codes are implemented in C language. For M1, the code is compiled by Intel C++ Compiler version 13 (/O3 /QxAVX /QaxAVX) and Visual C++ Compiler 2010 (/Ox). For others, the codes are compiled by Intel C++ Compiler (/O3 /QxAVX /QaxAVX /fp:precise) and Visual C++ Compiler 2010 (/Ox /fp:precise) via MATLAB Executable (Mex files). If we do not attach /fp:precise as a compiler option, then the order of computations may change. This change is not allowed for the a priori error estimation. We generate n-points $p_i = (x_i,\ y_i)$, where $x_i, y_i, i = 1, \ldots, n$ are pseudo-random numbers drawn from the standard normal distribution by using MATLAB built-in function `randn`. All 2d orientation problems are solved by each floating-point filter, namely no robust computation is called, so that we can compare the efficiency of floating-point filters. Computing times for $n = 10^8$ are displayed in Table 3.2. The ratio of computing times (M2, M3, M4) / M1 is also shown in parenthesis. From Table 3.2, it is confirmed that our filter works faster than other filters.

Remark 3.2 One may think that once $\max_{1\le i\le n}(|x_i|)$, $\min_{1\le i\le n}(|x_i|)$, $\max_{1\le i\le n}(|y_i|)$ and $\min_{1\le i\le n}(|y_i|)$ are obtained, then overflow and underflow might be predictable. But the number of choices for three points from n points is $\mathcal{O}(n^3)$. Thus, even if overflow would occur in the floating-point evaluation for the orientation problem for some few triples of points, we will in general not detect these exceptional cases in algorithms in computational geometry, for example, algorithms for convex hull, since the total number of solving orientation problems in common algorithms for the convex hull is less than $\mathcal{O}(n^2)$.

4 Fully-Static Filter

Define $p_i = (x_i, y_i) \in \mathbb{F}^2, i = 1..n$. Applications like the convex hull problem require to compute

$$fl((x_i - x_k)(y_j - y_k) - (y_i - y_k)(x_j - x_k))$$

where $i,\ j$ and k are pairwise distinct. A static filter gives an error bound for all such triples (i, j, k). Let $m_x = \max_{1\le i\le n} |x_i|$ and $m_y = \max_{1\le i\le n} |y_i|$. By using a technique from [3], we derive

$$\begin{aligned} & fl(\theta(|(x_i - x_k)(y_j - y_k) - (y_i - y_k)(x_j - x_k)| + \mathbf{u}_N)) \\ \le\ & fl(\theta((|x_i| + |x_k|)(|y_j| + |y_k|) + (|y_i| + |y_k|)(|x_j| + |x_k|) + \mathbf{u}_N)) \\ \le\ & fl(\theta(8 m_x m_y + \mathbf{u}_N)). \end{aligned} \tag{4.1}$$

Here, we assume that no underflow occurs in (4.1)[5]. Then, the bound (4.1) is independent from i, j and k. However, a static error bound is evaluated in advance of main computations at once in applications like the convex hull. We derive a smaller error bound than (4.1). Assume $fl(m_x m_y) \geq \mathbf{u}_N$. Let $m'_x = \max_{1\leq i\leq n} x_i$, $m'_y = \max_{1\leq i\leq n} y_i$, $n'_x = \min_{1\leq i\leq n} x_i$ and $n'_y = \min_{1\leq i\leq n} y_i$. From (3.3), for $1 \leq i \leq 4$,

$$t_i = x_i + \varepsilon_i, \quad |\varepsilon_1|, |\varepsilon_4| \leq \mathbf{u}\cdot \mathtt{ufp}(m'_x - n'_x), \quad |\varepsilon_2|, |\varepsilon_3| \leq \mathbf{u}\cdot \mathtt{ufp}(m'_y - n'_y).$$

From (3.4) without underflow,

$$x_1x_2 = x_5 + \varepsilon_5, \quad |\varepsilon_5| \leq \mathbf{u}\cdot \mathtt{ufp}(fl(x_1x_2)), \quad x_3x_4 = x_6 + \varepsilon_6, \quad |\varepsilon_6| \leq \mathbf{u}\cdot \mathtt{ufp}(fl(x_3x_4)).$$

From assumption (3.15),

$$\begin{aligned} t_7 &= t_5 - t_6 = t_1t_2 - t_3t_4 = (x_1+\varepsilon_1)(x_2+\varepsilon_2) - (x_3+\varepsilon_3)(x_4+\varepsilon_4) \\ &= x_5 + x_1\varepsilon_2 + x_2\varepsilon_1 + \varepsilon_1\varepsilon_2 + \varepsilon_5 - (x_6 + x_3\varepsilon_4 + x_4\varepsilon_3 + \varepsilon_3\varepsilon_4 + \varepsilon_6) \\ &= x_7 + x_1\varepsilon_2 + x_2\varepsilon_1 + \varepsilon_1\varepsilon_2 + \varepsilon_5 - (x_3\varepsilon_4 + x_4\varepsilon_3 + \varepsilon_3\varepsilon_4 + \varepsilon_6). \end{aligned}$$

Let $\alpha = fl(m'_x - n'_x)$ and $\beta = fl(m'_y - n'_y)$. We have

$$\begin{aligned} |x_7 - t_7| &\leq |x_1||\varepsilon_2| + |x_2||\varepsilon_1| + |\varepsilon_1||\varepsilon_2| + |\varepsilon_5| + |x_3||\varepsilon_4| + |x_4||\varepsilon_3| + |\varepsilon_3||\varepsilon_4| + |\varepsilon_6| \\ &\leq 2\alpha\mathbf{u}\cdot\mathtt{ufp}(m'_y - n'_y) + 2\beta\mathbf{u}\cdot\mathtt{ufp}(m'_x - n'_x) \\ &\quad + 2\mathbf{u}\cdot\mathtt{ufp}(fl(\alpha\beta)) + 2\mathbf{u}^2\mathtt{ufp}(m'_x - n'_x)\mathtt{ufp}(m'_y - n'_y) \\ &= fl(2\alpha\mathbf{u}\cdot\mathtt{ufp}(\beta)) + fl(2\beta\mathbf{u}\cdot\mathtt{ufp}(\alpha)) + fl(2\mathbf{u}\cdot\mathtt{ufp}(fl(\alpha\beta)) \\ &\quad + fl(2\mathbf{u}^2\mathtt{ufp}(\alpha)\mathtt{ufp}(\beta)) =: T_1. \end{aligned}$$

Define

$$T_2 := fl(2\alpha\mathbf{u}\cdot\mathtt{ufp}(\beta) + 2\beta\mathbf{u}\cdot\mathtt{ufp}(\alpha) + 2\mathbf{u}\cdot\mathtt{ufp}(\alpha\beta) + 2\mathbf{u}^2\mathtt{ufp}(\alpha)\mathtt{ufp}(\beta)).$$

For $p \in \mathbb{F}^n$ with $n \leq \mathbf{u}^{-1}$, Rump [10] derived

$$|\sum_{i=1}^{n} p_i - fl(\sum_{i=1}^{n} p_i)| \leq fl((n-1)\mathbf{u}\cdot\mathtt{ufp}(\sum_{i=1}^{n}|p_i|)). \tag{4.2}$$

By (4.2) with $n = 4$, T_1 is bounded by

$$T_1 \leq T_2 + 3\mathbf{u}\cdot\mathtt{ufp}(T_2) < \mathtt{succ}(fl(T_2 + 3\mathbf{u}\cdot\mathtt{ufp}(T_2))). \tag{4.3}$$

We now compare the error bounds (4.1) and (4.3). First, we generate $x = \mathtt{randn}(10000,1)$ and $y = \mathtt{randn}(10000,1)$ a hundred times. Tables 4.1 and 4.2 show the minimum, the mean, and the maximum ratio (4.3) / (4.1). Next, we generate $x = \mathtt{rand}(10000,1)$ and $y = \mathtt{rand}(10000,1)$. Tables 4.1 and 4.2 show the minimum, the mean, and the maximum ratio (4.3) / (4.1). According to Tables 4.1 and 4.2, our filter produces the better error bounds.

[5] If this is not true, a suitable scaling by multiplying a large number with a power of two for all floating-point coordinates should be applied.

minimum	mean	maximum
0.4537	0.5766	0.9180

Table 4.1 Ratio of the error bound for $x = \texttt{randn}(10000,1)$ and $y = \texttt{randn}(10000,1)$

minimum	mean	maximum
0.1250	0.1250	0.1251

Table 4.2 Ratio of the error bound for $x = \texttt{rand}(10000,1)$ and $y = \texttt{rand}(10000,1)$

5 Conclusion

We have developed new floating-point filters for the two-dimensional orientation problem. Some of the previously available algorithms for this purpose could either not handle floating-point exceptions like overflow or underflow, or they required additional branches for treating them at the expense of slower performance. In contrast, our algorithms work uniformly without extra branching also for underflow and overflow situations, and they are highly performant. As a benchmark we implemented our floating-point filters and previously available ones for the convex hull problem. It turned out that our filters have a better performance than the other ones without recognizable loss of quality.

Acknowledgements The authors wishes to thank the anonymous referee for constructive and valuable comments. This research was partially supported by the CREST program, Japan Science and Technology Agency (JST).

References

1. IEEE Standard for Floating-Point Arithmetic, Std 754–2008, 2008.
2. H. Brönnimann, C. Burnikel, S. Pion, Interval Arithmetic Yields Efficient Dynamic Filters for Computational Geometry, Discrete Applied Mathematics, 109:25–47, 2001.
3. C. Burnikel, S. Funke, M. Seel, Exact Geometric Computation Using Cascading, International Journal of Computational Geometry & Applications, 11(3):245-266, 2001.
4. N.J. Higham, Accuracy and Stability of Numerical Algorithms, second edition, SIAM Publications, Philadelphia, 2002.
5. L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, C. Yap, Classroom Examples of Robustness Problems in Geometric Computations, Computational Geometry, 40, 61–78, 2008.
6. G. Melquiond, S. Pion, Formally certified floating-point filters for homogenous geometric predicates, Theoretical Informatics and Applications, Special issue on Real Numbers, 41:57–69, 2007.
7. A. Meyer, S. Pion, FPG: A code generator for fast and certified geometric predicates, 8th Conference on Real Numbers and Computers (RNC), pages 47-60, Santiago de Compostela, Spain, 2008.
8. V. Y. Pan, Y. Yu, Certified Computation of the Sign of a Matrix Determinant, Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms, 715-724, ACM Press, New York, and SIAM Publications, Philadelphia, 1999.
9. S. Pion, A. Fabri, A Generic Lazy Evaluation Scheme for Exact Geometric Computations, Science of Computer Programming, Special issue on library-centric software design (LCSD 2006), 76(4):307-323, 2011.
10. S. M. Rump, Error estimation of floating-point summation and dot product, BIT Numerical Mathematics, 52(1):201-220, 2012.

11. S. M. Rump, P. Zimmermann, S. Boldo, and G. Melquiond, Computing predecessor and successor in rounding to nearest. BIT Numerical Mathematics, 49(2):419-431, 2009.
12. J. R. Shewchuk, Adaptive Precision Floating-point Arithmetic and Fast Robust Geometric Predicates, Discrete & Computational Geometry, 18, 305–363, 1997.
13. J. R. Shewchuk, C code for the 2D and 3D orientation and incircle tests, http://www.cs.cmu.edu/ quake/robust.html.
14. Computational Geometry Algorithms Library, http://www.cgal.org/.