点と平面との位置関係を判定する高速かつロバストなアルゴリズム

尾崎 克久 *　荻田 武史 †,*　Siegfried M. Rump‡,*　大石 進一 *
* 早稲田大学 理工学術院
† 科学技術振興機構 戦略的創造研究推進事業
‡Institute for Reliable Computing, Hamburg University of Technology

Fast and Robust Algorithm for Geometric Predicates using Floating-Point Arithmetic

Katsuhisa Ozaki*　Takeshi Ogita†,*
Siegfried M. Rump‡,*　Shin'ichi Oishi*
*Faculty of Science and Engineering, Waseda University
†CREST, Japan Science and Technology Agency
‡Institute for Reliable Computing, Hamburg University of Technology

Abstract. This paper is concerned with the computational geometry. A number of geometric problems can be boiled down to the determinant predicates, i.e. whether the sign of the determinant is positive, negative or zero. Among such problems, the Orient3D is focused in this paper. A fast and adaptive method for rigorously solving Orient3D is proposed. The proposed method in this paper is based on a new accurate floating-point summation algorithm which has just been developed by Rump, Ogita and Oishi. The proposed method ideally works with depending on difficulty of the problem, i.e., if the condition number of the problem becomes larger, then the computational cost for the method gradually increases. Numerical results are presented for illustrating that the proposed method is faster than the state-of-the-art method proposed by Demmel-Hida.

1. はじめに

本論文では計算幾何学における問題の 1 つである 3 次元空間における点と平面の位置関係を判定する問題 (Orient3D) に対して，IEEE 754 規格に従う浮動小数点演算を用いた高速かつロバストなアルゴリズムを提案する．3 次元空間内において 3 点 $A = (a_x, a_y, a_z), B = (b_x, b_y, b_z), C = (c_x, c_y, c_z)$ から定義される平面 H と，$D = (d_x, d_y, d_z)$ を平面の左側か右側か，または平面上かを判定したい点とするとき（例

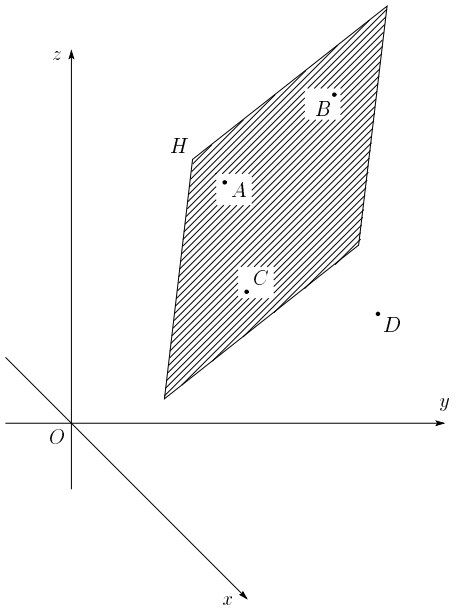

Fig. 1. The problem ORIENT3D.

えば，Fig. 1)，点と平面の位置関係は以下の行列式の符号

$$\mathrm{sgn}(\det(G)), \qquad G := \begin{pmatrix} a_x - d_x & a_y - d_y & a_z - d_z \\ b_x - d_x & b_y - d_y & b_z - d_z \\ c_x - d_x & c_y - d_y & c_z - d_z \end{pmatrix} \tag{1.1}$$

により決定される．点が平面に近接している場合，通常の倍精度演算では丸め誤差の累積により正しい結果が得られない場合がある．この問題の解決策として，4 倍精度演算や多倍長精度演算を利用すれば行列式の正しい符号が得られる可能性は高まるが，計算結果が保証されたことにはならない．また多倍長精度演算はソフトウェアによって実現されるため，一般的に倍精度演算よりもはるかに低速である．

この問題に対して，Shewchuk は式 (1.1) の評価を問題の難しさに応じて段階的に計算する手法を提案した [5]．問題が悪条件であることも想定に入れて，始めから厳密な計算を行えば，良条件な問題に対してもかなりの計算時間をかけるために効率が悪い．この点で，Shewchuk の適応的な手法は，行列式の符号が簡単に保証される場合には高速に計算され，問題が悪条件な場合には厳密な計算を行うという非常に優れた手法である．また，Shewchuk のアルゴリズムは多倍長精度演算を用いず，通常の浮動小数点演算のみを用いるために高速であることが示されている．

一方，Demmel と Hida は拡張倍精度演算を用いて，浮動小数点数の総和を高精度に計算するアルゴリズムを提案した [2]．この手法は問題が悪条件の場合，Shewchuk のアルゴリズムよりも高速であることが示されている．また Shewchuk の適応的な手法と組み合わせることで，良条件な問題に対しては Shewchuk のアルゴリズムと同じ時間で計算さ

れ，問題が悪条件な場合には Shewchuk のアルゴリズムより高速なため，現時点ではこの手法が最も有力な手法となっている．

ごく最近に，Rump, Ogita, Oishi により結果の精度を保証する高速な総和の計算法が開発された [4]．この手法は一般に計算速度のパフォーマンスを下げる原因となる入力データのソート，分岐，多倍長精度の利用，ポインタによる指数部や仮数部への直接アクセスなどを使用しないため非常に高速であることが示され，また通常の浮動小数点演算のみを用いるため計算機環境に依存せずに実装できる利点がある．

本論文では，Rump らの新しい浮動小数点数の総和のアルゴリズムを Orient3D を解くために特化した手法を提案する．式 (1.1) の計算は 96 項の浮動小数点数の総和に誤差なく展開することが可能であるが，このデータの特性と Rump らのアルゴリズムの計算法を整理すると，Orient3D を従来より高速に解くことが可能となる．数値実験により，提案手法は良条件の問題に対しては Shewchuk の手法と同じ計算時間で計算され，悪条件の問題に対しては Demmel-Hida の手法より高速であることを示す．

2. 行列式の変形

本節では論文中に用いる各種記号及び行列式 (1.1) の計算を 96 項の浮動小数点数の和に誤差なく展開する手法について述べる．本論文では，IEEE 754 規格に従う倍精度浮動小数点演算を用いる．

まず，式 (1.1) の計算を 96 項の浮動小数点数の総和に誤差なく展開する手法について述べる．式 (1.1) は 3 次の行列式であるため容易に展開され，

$$\begin{aligned}\det(G) = {} & a_x b_y c_z - a_x b_y d_z + a_x c_y d_z - a_x c_y b_z + a_x d_y b_z - a_x d_y c_z \\ & - b_x a_y c_z + b_x a_y d_z - b_x c_y d_z + b_x c_y a_z - b_x d_y a_z + b_x d_y c_z \\ & + c_x a_y b_z - c_x a_y d_z + c_x b_y d_z - c_x b_y a_z + c_x d_y a_z - c_x d_y b_z \\ & - d_x a_y b_z + d_x a_y c_z - d_x b_y c_z + d_x b_y a_z - d_x c_y a_z + d_x c_y b_z\end{aligned} \tag{2.1}$$

と 3 つの浮動小数点数の積が 24 項現れる．

ここで浮動小数点演算による “無誤差変換 (Error-free Transformation)” [3] と呼ばれるアルゴリズムをいくつか紹介する．文献 [1] では，$a, b \in \mathbb{F}$ に対して $a \cdot b = x + y$ $(x, y \in \mathbb{F})$ と誤差なく変換する下記のアルゴリズムが提案されている．以後，本論文ではアルゴリズムに MATLAB の表記法を用いる．

アルゴリズム 1 (Dekker [1]) $a, b \in \mathbb{F}$ に対して $a \cdot b = x + y$ $(x, y \in \mathbb{F}, |y| \leq \mathbf{u}|x|)$ と誤差なく変換するアルゴリズム．

function $[x, y] = \texttt{TwoProduct}(a, b)$
  $x = \mathrm{fl}(a \cdot b)$
  $[a_1, a_2] = \texttt{Split}(a)$
  $[b_1, b_2] = \texttt{Split}(b)$
  $y = \mathrm{fl}(a_2 \cdot b_2 - (((x - a_1 \cdot b_1) - a_2 \cdot b_1) - a_1 \cdot b_2))$

上記のアルゴリズムは，$a \in \mathbb{F}$ を $a = a_h + a_t$ を満たし，仮数部の先頭ビットから最大 26 ビットまでが非ゼロであるような 2 つの浮動小数点数 $a_h, a_t \in \mathbb{F}$ に誤差なく分解する下記のアルゴリズムを利用する．

アルゴリズム 2 (Dekker [1]) $a \in \mathbb{F}$ に対して 2 つの浮動小数点数 $a_h, a_t \in \mathbb{F}$ の和に誤差なく分解するアルゴリズム．

$$
\begin{array}{l}
\text{function } [a_h, a_t] = \texttt{Split}(a) \\
\quad c = \mathrm{fl}(\texttt{factor} \cdot a) \qquad \texttt{\% factor} = 2^{27} + 1 \\
\quad a_h = \mathrm{fl}(c - (c - a)) \\
\quad a_t = \mathrm{fl}(a - a_h)
\end{array}
$$

注意 1 使用する CPU によっては，レジスタ上では倍精度の仮数部 53 ビットを超える精度で計算される場合がある．例えば，Intel Pentium 系の CPU では，レジスタ上では仮数部 64 ビットで演算可能である．通常，演算の精度がより高ければ結果の精度も高くなるというのは利点になることが多いが，アルゴリズム 1, 2 の計算では IEEE 754 で定められた倍精度の仮数部 53 ビットで厳密に計算されなければならないため，使用する CPU，コンパイラの特性，コンパイラオプション等に注意しなければならない．これは，本論文で扱うアルゴリズムすべてに対して言えることである．

ここで式 (2.1) の第 1 項である $a_x b_y c_z$ に対しアルゴリズム 1 を

$$[e, f] = \texttt{TwoProduct}(a_x, b_y)$$

と適用すると

$$a_x b_y c_z = (e + f)c_z = e \cdot c_z + f \cdot c_z \tag{2.2}$$

を得る．また $e \cdot c_z$ と $f \cdot c_z$ に対してアルゴリズム 1 をそれぞれに

$$[s_1, s_2] = \texttt{TwoProduct}(e, c_z), \qquad [s_3, s_4] = \texttt{TwoProduct}(f, c_z)$$

と適用すると

$$a_x b_y c_z = s_1 + s_2 + s_3 + s_4 \tag{2.3}$$

と 3 つの浮動小数点数の積は 4 項の浮動小数点数の和に誤差なく展開することができる．ここで式 (2.1) 中のすべての項に対して，式 (2.2), (2.3) と同様の変形を行うことにより，式 (1.1) は 96 項の浮動小数点数の和に誤差なく展開することができる．

3. 提案手法

本節では前節により作成される浮動小数点数の総和に対して，Rump らの高速かつ高精度な総和計算アルゴリズム AccSum [4] を基本とした提案手法について述べる．

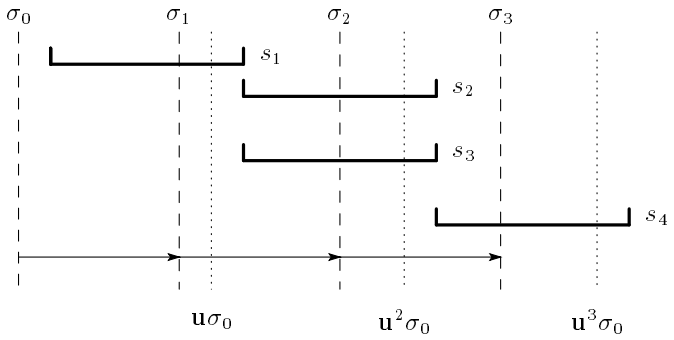

Fig. 2. Relation between s_1, s_2, s_3 and s_4.

まず，AccSum の主計算部分について説明をする．入力データ $p \in \mathbb{F}^n$ に対して $\sigma = 2^{\lceil \log_2(n+2) \rceil + \lceil \log_2 \max_i |p_i| \rceil}$ としたときに，AccSum の主計算部分は以下のようになる．

アルゴリズム 3 (Rump-Ogita-Oishi [4]) $p \in \mathbb{F}^n$ に対して $\tau + \sum_{i=1}^n p'_i = \sum_{i=1}^n p_i$ と誤差なく変形するアルゴリズム．

```
function [p', τ] = ExtractSum(p, σ)
  q = fl((σ + p) − σ);          % q_i = fl((σ + p_i) − σ)
  τ = sum(q);                   % τ = fl(Σ_{i=1}^n q_i) = Σ_{i=1}^n q_i
  p' = fl(p − q);               % p'_i = fl(p_i − q_i) = p_i − q_i
```

文献 [4] の手法は，総和の計算結果が faithful であることが保証されるまでアルゴリズム 3 の σ を変えながら計算していく．ここで，faithful とは「誤差を含まない正確な計算結果を隣り合うどちらかの浮動小数点数に丸めたものであること」を意味する（必ずしも nearest ではない）．ただし，ORIENT3D では行列式の符号さえ確定すれば良いので，アルゴリズムの終了条件を AccSum よりも緩和することができる．具体的には，アルゴリズム 3（ExtractSum）を実行したときに

$$\max_i |p'_i| \leq \mathbf{u}\sigma$$

が成立することと，$\sum_{i=1}^{96} p_i = \tau + \sum_{i=1}^{96} p'_i$ から，$\sum_{i=1}^{96} p_i$ の符号が保証される条件は

$$|\sum_{i=1}^{96} p'_i| \leq \sum_{i=1}^{96} |p'_i| \leq 96\mathbf{u}\sigma < |\tau| \tag{3.1}$$

と緩和できる．ここで，式 (2.1) における絶対値最大の項を xyz とすると，式 (2.2), (2.3) と同様の計算により

$$xyz = (e + f)z = s_1 + s_2 + s_3 + s_4$$

と展開される．ここで s_1, s_2, s_3, s_4 の関係について述べる（Fig. 2）．アルゴリズム 1 より

$$s_1 = \mathrm{fl}((xy)z), \qquad |s_2| \le \mathbf{u}|s_1|, \qquad |s_4| \le \mathbf{u}|s_3| \tag{3.2}$$

が成立し，また

$$|s_3| = |\mathrm{fl}(fz)| \le |\mathrm{fl}(\mathbf{u}(xy)z)| = \mathbf{u}|\mathrm{fl}((xy)z)| = \mathbf{u}|s_1| \tag{3.3}$$

が成立する．さらに，式 (3.2), (3.3) から

$$|s_4| \le \mathbf{u}^2|s_1| \tag{3.4}$$

が成立する．ここでアルゴリズム 3 において使用される変数 $\sigma := \sigma_0$ は $\sigma_0 > |\mathrm{fl}((xy)z)| = |s_1|$ を満たすことから下記計算では情報落ちが発生し

$$\mathrm{fl}(s_i + \sigma_0) = \sigma_0 \quad \Longrightarrow \quad \mathrm{fl}((s_i + \sigma_0) - \sigma_0) = 0, \quad i = 2, 3, 4 \tag{3.5}$$

となる．以上の議論により，アルゴリズム 3 の 1 回目の適用時には s_2, s_3, s_4 の計算が必要ないため，この時点ではデータとして s_1 だけを作成し s_2, s_3, s_4 は作成しなくても良いことがわかる．ここで終了条件を表す式 (3.1) が満たされない場合には s_2, s_3 を作成し，$\sigma_1 := 2^{-46}\sigma_0$ として再びアルゴリズム 3 を適用するが

$$\mathbf{u}\sigma_1 = \mathbf{u}(2^{-46}\sigma_0) > \mathbf{u}^2\sigma_0 > \mathbf{u}^2|s_1| \ge |s_4| \tag{3.6}$$

が成立するために下記計算において情報落ちが発生し

$$\mathrm{fl}(s_4 + \sigma_1) = \sigma_1 \quad \Longrightarrow \quad \mathrm{fl}((s_4 + \sigma_1) - \sigma_1) = 0 \tag{3.7}$$

を得る．以上の議論により，2 回目の適用時には s_4 の計算が必要ないため，この時点では s_4 は作成しなくても良いことがわかる．そして，σ_0 が絶対値最大の項 xyz より大きいことから，ここまでの議論は式 (2.1) におけるすべての項について成立する．また点データの入力は浮動小数点数であるが，式 (2.1) やアルゴリズム 1, 3 の計算の際にオーバフローやアンダフローが起きない範囲で入力は行われなくてはならない．

以下に提案手法の概要を示す．アルゴリズム 3 において

Step 1: 1 回目の適用時には通常の浮動小数点演算によって 24 項のデータを作成し計算をする．

Step 2: 2 回目は 48 項のデータをアルゴリズム 1（`TwoProduct`）を用いて追加し，72 項の和を計算する．

Step 3: 3 回目は 24 項のデータをアルゴリズム 1 を用いて追加し，96 項の和を計算する．

Step 4: 4 回目以降では 96 項の和を計算する．

ここで，$A = (a_x, a_y, a_z), B = (b_x, b_y, b_z), C = (c_x, c_y, c_z), D = (d_x, d_y, d_z)$ としたときに，データを作成する関数を以下のように定義する．

- $p^{(1)} = \texttt{MakeData1}(A, B, C, D)$ は Step 1 で計算すべき 24 項を格納したベクトル $p^{(1)} \in \mathbb{F}^{24}$ を返す.
- $p^{(2)} = \texttt{MakeData2}(A, B, C, D, p^{(1)})$ は $p^{(1)} \in \mathbb{F}^{24}$ に **Step 2** で計算すべき 48 項を作成し追加した $p^{(2)} \in \mathbb{F}^{72}$ を返す関数とする.
- $p = \texttt{MakeData3}(A, B, C, D, p^{(2)})$ は $p^{(2)} \in \mathbb{F}^{72}$ に **Step 3** で計算すべき残りの 24 項を作成し追加した $p \in \mathbb{F}^{96}$ を返す関数とする.

これまでの議論をまとめて，提案手法のアルゴリズムを以下に示す.

アルゴリズム 4 4 点 $A = (a_x, a_y, a_z)$, $B = (b_x, b_y, b_z)$, $C = (c_x, c_y, c_z)$, $D = (d_x, d_y, d_z)$ が与えられたときに，Orient3D における行列式の符号 $s = \{1, 0, -1\}$ を求めるアルゴリズム.

function $s = \texttt{OurOrient3D}(A, B, C, D)$
  $p^{(1)} = \texttt{MakeData1}(A, B, C, D)$; % 24 項作成
  $\sigma_0 = 2^{7+\lceil \log_2 \max(|p^{(1)}|) \rceil}$;
  $[p^{(1)}, \tau] = \texttt{ExtractSum}(p^{(1)}, \sigma_0)$; % Step 1
  if $|\tau| \geq 25\mathbf{u}\sigma_0$ % 1 回目の判定
    $s = \mathrm{sign}(\tau)$;
    return;
  end
  $p^{(2)} = \texttt{MakeData2}(A, B, C, D, p^{(1)})$; % 48 項作成
  $\tau_1 = \tau$;
  $\sigma_1 = 2^{-46}\sigma_0$;
  $[p^{(2)}, \tau] = \texttt{ExtractSum}(p^{(2)}, \sigma_1)$; % Step 2
  $\tau_1 = \mathrm{fl}(\tau_1 + \tau)$;
  if $|\tau_1| \geq 73\mathbf{u}\sigma_1$ % 2 回目の判定
    $s = \mathrm{sign}(\tau_1)$;
    return;
  end
  $p = \texttt{MakeData3}(A, B, C, D, p^{(2)})$; % 24 項作成
  $\sigma = 2^{-46}\sigma_1$;
  while $\sigma > 2^{-1022}$ % Step 3 以降
    $[p, \tau] = \texttt{ExtractSum}(p, \sigma)$;
    $\tau_1 = \mathrm{fl}(\tau_1 + \tau)$;
    if $|\tau_1| \geq 97\mathbf{u}\sigma$
      $s = \mathrm{sign}(\tau_1)$;
      return;
    end
    $\sigma = 2^{-46}\sigma$;
  end
  $s = \mathrm{sign}(\mathrm{fl}(\tau_1 + \texttt{sum}(p)))$; % アンダフローの範囲のみ

以下では，アルゴリズム 4 の停止条件について説明する．まず，提案方式の各ステップにおいて生成されるデータの関係を考慮し，式 (3.1) からさらに停止条件を改良している.

Step 1 終了後，未計算のデータの総和 q について考えると，$\mathbf{u}\sigma_0$ よりも小さいデータが 24 項，$\mathbf{u}\max_i |p_i|$ よりも小さいデータが 48 項，$\mathbf{u}^2 \max_i |p_i|$ よりも小さいデータが 24 項あるが，$\max_i |p_i| \leq 2^{-7}\sigma_0$ であるから

$$\begin{aligned} |q| &< 24\mathbf{u}\sigma_0 + 48\mathbf{u}\max_i |p_i| + 24\mathbf{u}^2 \max_i |p_i| \\ &\leq (24 + 48 \cdot 2^{-7} + 24 \cdot 2^{-7}\mathbf{u})\mathbf{u}\sigma_0 \\ &\leq 25\mathbf{u}\sigma_0 \end{aligned}$$

となる．Step 1 終了時に行列式の符号が保証されるためには，$|\tau| > |q|$ が成立していれば良いので，このときの終了条件は $|\tau| \geq 25\mathbf{u}\sigma_0$ となる．同様に，Step 2 終了時に符号が保証される条件は $|\tau_1| \geq 73\mathbf{u}\sigma_1$ となり，Step 3 以降では $|\tau_1| \geq 97\mathbf{u}\sigma_1$ となる．最後に，while 文の継続条件 $\sigma > 2^{-1022}$ について説明する．$\sigma \leq 2^{-1022}$ となった場合は，浮動小数点数の正確な総和がアンダフローの範囲に入ることになるため，文献 [4] の `AccSum` と同様に最後の $\mathrm{fl}(\tau_1 + \mathtt{sum}(p))$ の計算結果に誤差が含まれないことが保証されるので，アルゴリズムが停止可能であることを意味する．

4. 数値実験結果

本節では点と平面の位置関係を判定した数値実験結果について述べる．数値実験を比較するアルゴリズムは以下の通りとする．

- 手法 **A**　(1.1) を通常の浮動小数点数によって計算.
- 手法 **B**　Demmel-Hida による手法 [2].
- 手法 **C**　Shewchuk の手法 [5]+Demmel-Hida による方法 [2].
- 手法 **D**　提案手法 (アルゴリズム 4).
- 手法 **E**　Shewchuk の手法 [5]+ 提案手法 (アルゴリズム 4).

手法 **C** は Shewchuk の適応的なアルゴリズム [5] を用いて計算を進め，最後のステップまで進む場合に手法 **B** により計算する方法であり，手法 **E** も同様に Shewchuk のアルゴリズムの最後のステップにおいて手法 **D** を用いる方法である．

テスト問題として，3 種類の問題 Small，Large，Collinear を用意する．Small は各点の指数部が $[-4, 3]$ の範囲にあるランダムな点，Large は各点の指数部が $[-128, 127]$ の範囲にあるランダムな点，Collinear は点がほぼ平面上にあるランダムな点であるような問題であり，したがって，Small は比較的良条件，Collinear は悪条件な問題である．

Table 1 は Small，Large，Collinear に対して手法 **A** にかかる計算時間を 1 としたときの比を表す．手法 **A** 以外のロバストな手法の中で最も高速だった結果に下線を付してある．計算時間の計測には 1 つの座標の組み合わせにつき 50000 回以上の数値実験を行い，さらにランダムに 100 回座標を変えて実験を行った平均をとっている．数値実験に用いた計算機環境として CPU は Pentium IV 3.3GHz，OS は Linux，コンパイラは Intel

Table. 1. Comparision of ratio of computing time for various problems.

Problem	A	B	C	D	E
Small	1	87	1.3	21	1.3
Large	1	104	6	21	6
Collinear	1	94	125	76	104

C++ Compiler 9.0 を用いた.

Table 1 から，提案手法 **D** は Demmel-Hida の手法 **B** と比較して 20% 以上高速であり，悪条件である Collinear な問題に対しロバストな手法の中で一番高速であることがわかる．手法 **D** が手法 **B** より高速なことから，Collinear な問題に対して手法 **E** は手法 **C** より高速になることがわかる．また，Shewchuk の手法と提案手法を組み合わせた手法 **E** は，悪条件な問題に対しては，適応的なアルゴリズムを用いない手法 **B**，手法 **D** より遅くなるが，良条件な問題に対して高速に解くことができるためにバランスが良い手法であるといえる.

5. むすび

本論文では 3 次元空間における点と平面の位置関係を判定する高速かつロバストなアルゴリズムを提案した．Rump, Ogita, Oishi による高精度な浮動小数点数の総和の計算法と行列式から生成されるデータとの関係を整理した結果，アルゴリズムの過程で計算に必要な部分を限定でき，データ自身も必要に応じて作成すれば良いことがわかった．これにより，現時点で有力な Demmel-Hida の手法よりも高速なアルゴリズムを提案することができた．本論文で提案した方式と同様なことが計算幾何における他の問題（例えば，文献 [5] にある Oriend2D や InCircle など）にも適用可能である.

謝辞 多数の有益なコメントを頂いた査読者の方々に感謝いたします．本研究は，文部科学省科学研究費補助金（特別推進研究「精度保証付き数値計算学の確立」 No. 17002012）の補助を受けた.

参考文献

[1] T. J. Dekker: A floating-point technique for extending the available precision, Numer. Math., 18 (1971), 224–242.

[2] J. Demmel, Y. Hida: Fast and accurate floating point summation with application to computational geometry. Numerical Algorithms, 37:1–4 (2004), 101–112.

[3] T. Ogita, S. M. Rump, S. Oishi: Accurate sum and dot product, SIAM J. Sci. Comput., 26 (2005), 1955–1988.

[4] S. M. Rump, T. Ogita, S. Oishi: Accurate floating-point summation, submitted for publication, 2005. http://www.ti3.tu-harburg.de/publications/rump

[5] J. R. Shewchuk: Adaptive precision floating-point arithmetic and fast robust geometric predicates, Discrete & Computational Geometry 18 (1997), 305–363.

尾崎克久 (正会員)　〒169-0072 東京都新宿区大久保 3–14–9 シルマンホール 802
2004 年東海大学大学院理学研究科修士課程了．修士 (理学)．現在，早稲田大学理工学術院助手．2006 年早稲田大学第 16 回大川功記念賞受賞．

荻田武史 (正会員)　〒169-0072 東京都新宿区大久保 3–14–9 シルマンホール 802
2003 年早稲田大学大学院理工学研究科博士後期課程了．博士 (情報科学)．現在，科学技術振興機構研究員．2003 年早稲田大学小野梓記念学術賞受賞．

大石進一 (正会員)　〒169-0072 東京都新宿区大久保 2–4–12 新宿ラムダックスビル 902
1981 年早稲田大学大学院理工学研究科博士後期課程了．工学博士．現在，同大学理工学術院教授．電子情報通信学会論文賞 (3 回)，同学会猪瀬賞，大川出版賞，丹羽記念賞，早稲田大学小野梓賞各受賞．