数理解析研究所講究録
第1614巻 2008年 1-10

点と直線の位置関係の計算をロバストに行う 点と平面の位置関係の精度保証法

尾崎 克久* 荻田 武史† 大石 進一*

* 早稲田大学 理工学術院 (k_ozaki@aoni.waseda.jp)
† 東京女子大学 文理学部 数理学科

概 要

点と平面の位置関係の判定は3次元の計算幾何学の問題には必要不可欠な問題である．本報告ではこの判定を解くための行列式の符号に対する精度保証法を，高精度な総和の計算法を元に提案する．特に行列式が0の場合には，3点により平面が与えられない場合と点が平面上にある場合の2通りがあるが，それも含めて精度保証付きで判定できる手法を構築する．また最後に数値実験により提案手法の有効性を検証する．

Robust Geometric Predicates for 3D Orientation Problem by Using Robust Computation of 2D Orientation Problem

Katsuhisa OZAKI*, Takeshi OGITA†, Shin'ichi OISHI†

* Faculty of Science and Engineering, Waseda University
† Department of Mathematics, Tokyo Woman's Christian University

Abstract

This report is concerned with a robust computation for geometric predicates, especially, 3D orientation problem. The problem can be boiled down to a determinant predicate. In our previous work, we could develop a fast and robust algorithm for this problem. In this report, we modify our previous work by using a robust computation for 2D orientation problem. When the determinant is equal to zero, the new method can judge whether a plane does not span from 3 points or a point is on a plane. Finally, numerical results are presented showing the performance of the proposed method.

1 まえがき

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 (ORIENT3D) を精度保証付きで判定する高速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，

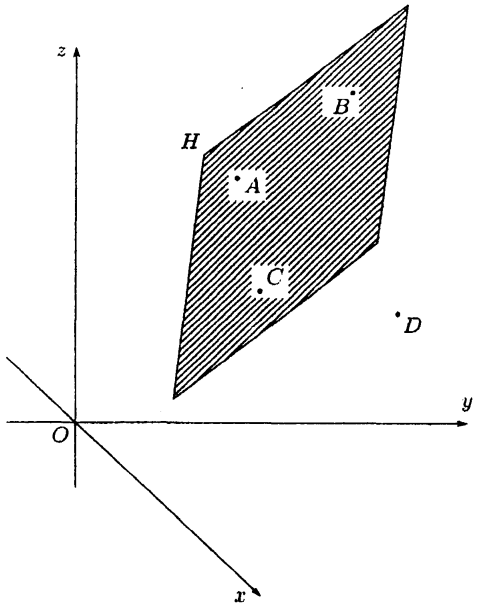

図 1: The problem ORIENT3D.

$D = (d_x, d_y, d_z)$ 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

$$\mathrm{sgn}(\det(G)), \qquad G := \begin{pmatrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ d_x & d_y & d_z & 1 \end{pmatrix} \tag{1}$$

により決定される．点が平面に近接している場合，または平面を構成する 2 つのベクトルが平行に近い場合は，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，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，任意に悪条件な問題が存在するために計算結果が保証されたことにはならない．

この問題に対して，Shewchuk は式 (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 [6]．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が終わり，問題が悪条件な場合には計算コストをかけて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]．高速に働くソートを利用し，拡張倍精度浮動小数点数による演算を利用する方法である．行列式を 96 項の浮動小数点数に展開し，彼らの高精度な和の計算法を適用することで行列式の符号を高速かつ正しく求めることに成功した．

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, 5]．IEEE 754 規格に従う浮動小数点演算のみで，高速かつ結果の精度を保証することができる優れたアルゴリズムである．著者らはこの高精度な総和の計算法を計算幾何学に現れる行列式に特化させ，高速に行列式の符号を保証することに成功した [7]．

本報告では，点と平面の位置関係について [7] とは異なった方式で，行列式の符号を精度保証付きで求めるアルゴリズムを提案する．特に行列式が 0 の場合には，提案手法は与えられた 3 点が平面を構成しないのか，または判定したい点が平面上にあるのかを明確にできる．最後に数値実験により提案手法の有効性を示す．

2 行列式に対する考察と計算法

本章では行列式 (1) の符号を正しく計算するために考察を行う．ここで 3 点が平面を構成しない場合は D の座標を用いず，A, B, C の座標のみで行列式が 0 になる．すなわち式 (1) が

$$(A, B, C \text{の座標を用いた式}) \, (\text{その他})$$

と積の形式で表現できる．この形式として，式 (1) は具体的には

$$\begin{aligned} & (d_x - b_x)\{(a_y - b_y)(c_z - b_z) - (a_z - b_z)(c_y - b_y)\} \\ + \; & (d_y - b_y)\{(a_z - b_z)(c_x - b_x) - (c_z - b_z)(a_x - b_x)\} \\ + \; & (d_z - b_z)\{(a_x - b_x)(c_y - b_y) - (a_y - b_y)(c_x - b_x)\} \end{aligned} \tag{2}$$

となる[1]．上式は部分的に 3 点 A, B, C をそれぞれ $y-z$ 平面・$z-x$ 平面・$x-y$ 平面に投影して点と直線の位置関係を判定していることになる．すなわち

$$(b_x - d_x)\begin{vmatrix} a_y & a_z & 1 \\ b_y & b_z & 1 \\ c_y & c_z & 1 \end{vmatrix} + (b_y - d_y)\begin{vmatrix} a_z & a_x & 1 \\ b_z & b_x & 1 \\ c_z & c_x & 1 \end{vmatrix} + (b_z - d_z)\begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} \tag{3}$$

と書くことができる．ここで，点と直線の位置関係を表す 3 次の行列式の値が浮動小数点演算の誤差の影響により正確でない場合は，式全体の符号が合わない可能性がある．よって，点と平面の位置関係を求めるには点と直線の位置関係を正確に求めることが大切であり，その手法については後述する．このため，計算順としては，

- 3 点 A,B,C により平面が構成されるかどうか (3 次の行列式をロバストに計算する).
- 点 D が A,B,C から生成される平面上にあるかどうか.

を順に精度保証付きで判定を行う．3 点により平面が構成されない場合には，式全体をロバストに計算する必要がないので高速に計算が終わることが期待できる．また，3 点 A,B,C が固定であり，かつ点を順次与えて判定する問題の場合は，一度この行列式をロバストに計算しておけば繰り返して利用ができる．

[1] 他の表現の仕方もある.

3 行列式から総和への無誤差変換

本章では，行列式の計算を浮動小数点数の総和に誤差なく変換する方法について説明する．まず，本報告に用いる各種記号及びアルゴリズムについて述べる．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．$\mathbb{F}$ を倍精度浮動小数点数の集合とし，$\mathbf{u} = 2^{-53}$ を unit roundoff とする．また $\mathrm{fl}(\cdots)$ は括弧内の演算をすべて浮動小数点演算により行うことを意味する．

まず，式 (3) における

$$\begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} \tag{4}$$

について，高精度に計算する方法について紹介する．ここでの議論は他の 3 次の行列式についても同様に成立する．上式は

$$a_x b_y + a_y c_x + b_x c_y - a_x c_y - a_y b_x - b_y c_x \tag{5}$$

と変形される．

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 [3] では，$a, b \in \mathbb{F}$ に対して $a+b = x+y$ $(x, y \in \mathbb{F})$ と誤差なく変換する下記のアルゴリズムを紹介している．以後，本論文ではアルゴリズムに MATLAB の表記法を用いる．

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

$$\begin{aligned} &\text{function } [x, y] = \texttt{TwoSum}(a, b) \\ &\quad x = \mathrm{fl}(a + b) \\ &\quad z = \mathrm{fl}(x - a) \\ &\quad y = \mathrm{fl}((a - (x - z)) + (b - z)) \end{aligned}$$

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．

Algorithm 2 (Dekker [1]) $a, b \in \mathbb{F}$ に対して $a \cdot b = x + y$ $(x, y \in \mathbb{F}, |y| \le \mathbf{u}|x|)$ と誤差なく変換するアルゴリズム．

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

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．

Algorithm 3 (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.

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

また，浮動小数点演算の誤差を返す関数 err を定義し，

$$
\begin{aligned}
[x, y] = \texttt{TwoSum}(a, b) \quad &\Rightarrow \quad err(a + b) = y \\
[x, y] = \texttt{TwoProduct}(a, b) \quad &\Rightarrow \quad err(a \cdot b) = y
\end{aligned}
$$

とする．式 (5) に対してエラーフリートランスフォーメーションを以下のように用いる.

$$
\begin{aligned}
[p_1, p_2] &= \texttt{TwoProduct}(a_x, b_y), \\
[p_3, p_4] &= \texttt{TwoProduct}(a_y, c_x), \\
[p_5, p_6] &= \texttt{TwoProduct}(b_x, c_y), \\
[p_7, p_8] &= \texttt{TwoProduct}(a_x, -c_y), \\
[p_9, p_{10}] &= \texttt{TwoProduct}(a_y, -b_x), \\
[p_{11}, p_{12}] &= \texttt{TwoProduct}(b_y, -c_x).
\end{aligned}
$$

この結果，式 (5) は 12 項の浮動小数点数の和に展開できる．よって総和を高精度に計算するアルゴリズムを行列式の計算に適用できる.

4 浮動小数点数の総和を高精度に求める計算法

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] を紹介する.

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，p の総和を計算する AccSum の主計算部分は以下のようになる[2].

Algorithm 4 (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.

$$
\begin{aligned}
&\text{function } [p', \tau] = \texttt{ExtractSum}(p, \sigma) \\
&\quad q = \mathrm{fl}((\sigma + p) - \sigma); && \% \ q_i = \mathrm{fl}((\sigma + p_i) - \sigma) \\
&\quad \tau = \texttt{sum}(q); && \% \ \tau = \mathrm{fl}(\textstyle\sum_{i=1}^n q_i) = \sum_{i=1}^n q_i \\
&\quad p' = \mathrm{fl}(p - q); && \% \ p'_i = \mathrm{fl}(p_i - q_i) = p_i - q_i
\end{aligned}
$$

[2]実際のコードでは log 関数を使用しない方式で記述できる [4].

上記アルゴリズムの実行後，必要な精度が得られていない場合には，σ を

$$\sigma := \sigma \cdot \phi, \ (\phi = 2^{\lceil \log_2(n+2) \rceil - 53}) \tag{6}$$

のように更新し，ExtractSum を再び実行する．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 4 の σ を変えながら計算していく[3]．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．

さらに文献 [5] では，K-fold faithful といい

$$\sum p = \tau_1 + \tau_2 + \cdots + \tau_k$$

と変形できる．ここで τ_1 は $\sum p$ における faithful な結果を表し，τ_2 は $\sum p - \tau_1$ における faithful な結果を表す．一般に τ_k は $\sum p - \sum_{i=1}^{k-1} \tau_i$ についての faithful な結果である．このように結果を複数の浮動小数点数で保持する形式を用いれば，総和における精度を任意に高めることができる．

5 提案手法

本章では，前記の計算法を用いて点と平面の位置関係を正確に判定するアルゴリズムを構築する．ここで 3 次の行列式は 12 項の和を計算する問題に帰着されることから，以下を満たす自然数 n_q, n_r, n_s が存在する[4]．

$$\begin{vmatrix} a_y & a_z & 1 \\ b_y & b_z & 1 \\ c_y & c_z & 1 \end{vmatrix} = \sum_{i=1}^{n_q} q_i, \quad \begin{vmatrix} a_z & a_x & 1 \\ b_z & b_x & 1 \\ c_z & c_x & 1 \end{vmatrix} = \sum_{i=1}^{n_r} r_i, \quad \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} = \sum_{i=1}^{n_s} s_i \tag{7}$$

各ベクトル q, r, s は，それぞれの行列式による K-fold faithful (K はそれぞれ n_q, n_r, n_s である) な結果を表し，実際に文献 [5] に記載されている手法を用いれば実行可能である．ここで 12 項の和を K-fold faithful に求めた結果から n_q, n_r, n_s の最大は 12 である．式 (3) における 3 つの行列式の faithful な値 q_1, r_1, s_1 がすべて 0 であれば，3 点 A, B, C が平面を構成しないと判断され，ここで計算が終了である[5]．3 点が平面を構成する場合には計算を続ける．まず，式 (3) 中にある以下の項に対して

$$[g_1, h_1] = \texttt{TwoSum}(b_x, -d_x), \ [g_2, h_2] = \texttt{TwoSum}(b_y, -d_y), \ [g_3, h_3] = \texttt{TwoSum}(b_z, -d_z)$$

を実行する．この後，現段階では計算すべき行列式は

$$(g_1 + h_1) \sum_{i=1}^{n_q} q_i + (g_2 + h_2) \sum_{i=1}^{n_r} r_i + (g_3 + h_3) \sum_{i=1}^{n_s} s_i \tag{8}$$

となる．ここで $(g_1 + h_1) \sum_{i=1}^{n_q} q_i$ に着目し，展開をすると

$$g_1 q_1 + g_1 q_2 + \ldots g_1 q_{n_q} + h_1 q_1 + h_1 q_2 + \cdots + h_1 q_{n_q}$$

[3]証明上は faithful であるが，実際の計算結果は nearest な結果を与える確率が高い．
[4]計算中にオーバーフローやアンダーフローが起きないことを仮定する．
[5]q_1, r_1, s_1 内の 2 つの値が 0 であれば，残りの 1 つも 0 である

が得られる．ここで TwoProduct をそれぞれに適用すると，

$$\mathrm{fl}(g_1q_1) + err(g_1q_1) + \mathrm{fl}(g_1q_2) + err(g_1q_2) + \mathrm{fl}(h_1q_1) + err(h_1q_1)\ldots \tag{9}$$

となり，式 (8) 全体から $4(n_q + n_r + n_s)$ 項が得られる．よって，これらの総和の計算に対して高精度な和の計算法を適用すれば正しい符号を得ることができる．ただし本報告では，ベクトルの要素間に存在する絶対値の大きな差に着目して計算量を削減することを試みる．

まず新しいベクトル w を用意し，

$$w_1 = \mathrm{fl}(g_1 \cdot q_1),\ w_2 = \mathrm{fl}(g_2 \cdot r_1),\ w_3 = \mathrm{fl}(g_3 \cdot s_1)$$

と初期のデータを格納する．次に Algorithm 4 の σ に相当するものを決定するために，項数について考察する．式 (8) をすべて浮動小数点数の和に展開した場合，式 (8) 中の $(g_1 + h_1)\sum_{i=1}^{n_q} q_i$ 内では最大 12 項の和と 2 項の和の積となる．また，それを展開して得られる 24 項に対して TwoProduct を用いると 48 項生成される．よって式 (8) には上記と同じ形式が 3 箇所あるために w は最大で 144 項生成される．この 3 項の要素を持つ w に対して σ を

$$\sigma_1 = 2^{\lceil \log_2 144 \rceil + \lceil \log_2 \max_i |w_i| \rceil}$$

と決定する．ここで TwoSum や TwoProduct を適用した際の関係式と，faithful の関係より，

$$2\mathbf{u}|g_1q_k| \geq g_1q_{k+1} \tag{10}$$

$$\mathbf{u}|g_1q_k| \geq |h_1q_k| \tag{11}$$

$$\mathbf{u}|g_1q_k| \geq err(g_1q_k) \tag{12}$$

が成立する．また $a, b \in \mathbb{F}$ に対して，浮動小数点演算では

$$\frac{1}{2}\mathbf{u}|a| \geq |b| \Longrightarrow \mathrm{fl}(a+b) = a \tag{13}$$

が満たされる．ここで $\mathrm{fl}(g_1q_1)$ 以外の項を v とすると，式 (9) において (10), (11), (12) より，$v \leq 2\mathbf{u}\sigma$ が成立する．よって σ_1 について

$$\sigma_1 \geq 144 \max|w|$$

が成立するために，少なくとも

$$\mathbf{u}\sigma_1 \geq v$$

が成立し，式 (13) より

$$\mathrm{fl}(v + \sigma_1) = \sigma_1 \quad \Longrightarrow \quad \mathrm{fl}((v + \sigma_1) - \sigma_1) = 0 \tag{14}$$

となるために，最初のステップでは $\mathrm{fl}(g_1q_1)$ 以外は計算対象にする必要はなく，他の r や s が関連する項においても同様の議論が成立する．よって，最初の ExtractSum の実行時には 3 項のみが計算対象となる．2 番目のステップでは，ϕ の定義より σ_1 は 45 ビットシフトするが，以下の項

$$err(g_1 \cdot q_1),\ \mathrm{fl}(e_1 \cdot q_1),\ \mathrm{fl}(g_1 \cdot q_2)$$

$$err(g_2 \cdot r_1), \quad \mathrm{fl}(e_2 \cdot r_1), \quad \mathrm{fl}(g_2 \cdot r_2)$$
$$err(g_3 \cdot s_1), \quad \mathrm{fl}(e_3 \cdot s_1), \quad \mathrm{fl}(g_3 \cdot s_2)$$

以降は前述と同様の理由により計算対象に入れなくて良い．一般に i 番目のステップでは

$$g_j \cdot r_k \ (j+k=i+1),\ err(g_j \cdot r_k) \ (j+k=i)$$

を満たす項を w に追加し，ExtractSum を実行する．

ここで最初の計算時には q_2, r_2, s_2 以降は必要なく，また 2 番目の計算時には q_3, r_3, s_3 以降は必要ない．そのために最初から K-fold faithful なデータをすべて求めるのではなく，q_i, r_i, s_i は i 番目のステップにて作成し，w へ追加することにより高速化が達成される可能性がある．

ここで，各ステップにおいて符号が保証される条件について考察する．アルゴリズム 4 (ExtractSum) を実行したときに

$$\max_i |p_i'| \leq \mathbf{u}\sigma$$

が成立することが知られている．w は最大で 144 項あることから，

$$\tau > 144\mathbf{u}\sigma$$

を満たせば符号は保証されるが，判定式に対し浮動小数点演算を行うために

$$\tau > \mathrm{fl}(145\mathbf{u}\sigma)$$

が，符号が保証される条件となる．

提案手法は，3 次の行列式に対して K-fold faithful な結果を求めるために，入力データに依存する話ではあるが，元の 12 項は 2,3 項に集約される．その後も絶対値の差を利用して計算範囲を限定するために高速に計算できると期待される．

6 数値実験結果

本章では数値実験結果を紹介し，提案手法の有効性について言及する．数値実験に用いた計算機環境として，CPU は Xion 2.8 GHz，OS は Linux である．

数値実験で比較するアルゴリズムは以下の通りである．

- M_1: Shewchuk による orient3d _fast [6]
- M_2: Shewchuk による orient3d _slow [6]
- M_3: Demmel-Hida による手法 [2]
- M_4: 著者らによる研究 [7]
- M_5: 提案手法

表 1: Comparison of computing times for various problems with GCC.

cond(v)	M_1	M_2	M_3	M_4	M_5
$1e+31$	0.61	0.84	0.59	0.54	0.50
$1e+33$	0.61	0.87	0.58	0.54	0.41
$1e+41$	0.63	0.87	0.58	0.59	0.58

表 2: Comparison of computing times for various problems with ICC.

cond(v)	M_1	M_2	M_3	M_4	M_5
$1e+31$	0.59	1.03	0.66	0.51	0.35
$1e+33$	0.57	1.07	0.66	0.51	0.36
$1e+41$	0.59	1.03	0.66	0.53	0.47

今回実験をする問題として，行列式の値がベクトル v の総和に変換されたとし，v の総和に関する条件数

$$\mathrm{cond}(v) = \frac{\sum |v|}{|\sum v|}$$

を悪条件に設定しながら問題を作成した．

表 1，2 は条件数をそれぞれ $10^{31}, 10^{33}, 10^{41}$ に設定した問題について，100000 回の計算に要した計算時間 (秒) を表す．表 1 はコンパイラとして GCC を，表 2 は Intel C++ Compiler (ICC) を利用した結果である．表 1，2 より，提案手法はどの手法よりも高速に行列式の値を保証できていることがわかる．ただし，提案手法がいつでも高速であることの証明はできない．使用する計算機環境及びコンパイラの最適化技術により，すべての手法の計算速度は変化する．また問題の条件数を固定したとしても，点の組み合わせは無数に存在し，すべての例について実験を行うのは困難である．

ただし，本数値実験において提案手法は上記の先行研究と比較して結果高速であり，また 3 点が平面を構成していないときにはより高速に判定ができる．さらに，平面を固定して点との位置関係を次々に判定していく場合には途中計算の結果を再利用でき，より効率よく計算ができると思われる．

7 結論

本報告では，点と平面の位置関係を求める問題について高速かつロバストな手法を提案した．提案手法は行列式が 0 であるときに，3 点が平面を構成するかどうか，また点が平面の上にあるのかどうかを順に判定でき，さらに先行研究よりも高速であることを示せた．

参考文献

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

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

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

[4] Rump, S. M. , Ogita, T. , Oishi, S.: Accurate Floating-Point Summation Part I: Faithful Rounding, submitted for publication, 2007. `http://www.ti3.tu-harburg.de/publications/rump`

[5] Rump, S. M. , Ogita, T. , Oishi, S.: Accurate Floating-Point Summation Part II: Sign, K-fold Faithful and Rounding to Nearest, submitted for publication, 2007. `http://www.ti3.tu-harburg.de/publications/rump`

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

[7] 尾崎克久，荻田武史，S. M. Rump，大石進一：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，日本応用数理学会論文誌特集号 16:4(2006)，pp. 195-204.