卒 業 論 文 概 要 書

CD

2007 年　2 月提出　　学籍番号 1G03R147-3

学科名	コンピュータ・ネットワーク工学科	氏 名	西川祐介	指 導 教 員	大石進一
研 究 題 目	点と直線の位置関係のロバストかつ高速な判定法				

1　序論

1.1　背景

幾何学の分野における基本的なアルゴリズムに多角形に対する点の位置判定がある．

このアルゴリズムは，ＧＩＳ (地理情報システム) においてデータの解析を行う際にも利用されている．代表的な判定方法として ray intersection, orientation, wedge method が挙げられるが，これらのすべての方法に点と直線の位置判定を行うアルゴリズムが用いられている．

しかし，点と直線の位置が非常に近いときには，丸め誤差の影響で正しく判定されないという問題が存在していた．

しかし，精度保証技術の進歩に伴い，高精度に演算を行う手法が提案されてきた．そのため，これらの手法を利用することによって，丸め誤差の影響を受けることなく常に正確な値を得ることが可能になっている．

1.2　本論文の目的

本論文の目的は，点と線の位置関係をロバストかつ高速に判定することである．

そこで，本論文では，Shewchuk のアルゴリズムを先行研究として行い，そのアルゴリズムよりも高速なアルゴリズムを提案することを目指す．

2　点と直線の位置関係の判定

直線と点の位置関係は以下の行列式の符号を調べることによって判定出来る．

$$\mathrm{A}=\begin{bmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{bmatrix}, \det(\mathrm{A}) = \mathrm{sign}(\det(\mathrm{A}))$$

すなわち，

$$\begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} = \begin{vmatrix} a_x - c_x & a_y - c_y \\ b_x - c_x & b_y - c_y \end{vmatrix}$$

$$= (a_x - c_x)(b_y - c_y) - (b_x - c_x)(a_y - c_y)$$

で得られる値の符号で判定される．

3　高精度演算

3.1　アルゴリズム TwoSum

$x, y, a, b \in \mathbf{F}$

$a + b = x + y$，$x = \mathrm{fl}(a + b)$

と誤差なく変換する．

3.2　アルゴリズム TwoProduct

$x, y, a, b \in \mathbf{F}$

$a \times b = x + y$

と誤差なく変換する．

3.3　アルゴリズム Expansion-Sum

複数の浮動小数点数で表される e と f

$e = \sum_{i=1}^{m} e_i$，$f = \sum_{i=1}^{n} f_i$ に対して

$$h = \sum_{i=1}^{m+n} h_i = e + f$$

と誤差のない演算を行う．

4　Shewchuk のアルゴリズム

Shewchuk は，点と直線の位置関係を判定するアルゴリズムを adaptive なものを提案している．

ロバストなアルゴリズムは，

$\det(\mathrm{A}) = (a_x - c_x)(b_y - c_y) - (b_x - c_x)(a_y - c_y)$

を全て高精度演算を行って最終的に正確な値を計算し，符号を判定する．

そして計算結果は，$\det(\mathrm{A}) = \sum_{i=1}^{16} \det_i$

と表される．

5　提案手法

Shewchuk のロバストなアルゴリズムでは，4 回の Two-Diff と，8 回の Two-Product，4 回の Expansion-Diff，3 回の Expansion-Sum を最終的に実行する．

しかし，正確な符号さえわかればいい今回の問題では，Expansion-Sum を 3 回繰り返す必要がない場合もある．

そこで，この部分を Rump-Ogita-Oishi らが提案したアルゴリズム(AccSum) を組み込んで，アルゴリズムを高速にする．

AccSum を用いると，図 5.1 のように 1 回目の処理では P[0], P[1], P[2], P[3]を計算するだけで済む．2 回目の処理では，P[0], P[1],…, P[7]のように処理する配列の数を順に増やしていく処理になる．

よって、途中で符号が確定すれば配列の全要素に対して計算することがなくなるので、Shewchuk の方法よりも速く処理が終了する場合がある．

そこで、この AccSum のアルゴリズムを利用して新しいアルゴリズムを作成する。

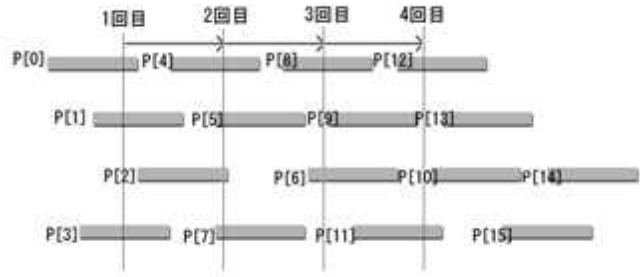

図 5.1 計算法とデータの関係

6 実験

6.1 実行環境

実験で用いた実行環境は以下のとおりである．

1. CPU : AMD Athlon 1.53GHz
2. Memory : 256MB
3. OS : TurboLinux 10.0
4. Program : C 言語 gcc コンパイラ

コンパイラオプション

-O2 -pipeline -function-all-roop
-finline-functions

6.2 実行結果

Method	計算時間
Shewchuk のロバストなアルゴリズム	0.039994
提案したロバストなアルゴリズム	0.037995
Shewchuk の Adaptive なアルゴリズム	0.049992
提案した Adaptive なアルゴリズム	0.040995

図 6.1 実行結果

7 考察

図 6.1 より，ロバストなアルゴリズムと adaptive なアルゴリズムのそれぞれにおいて，Shewchuk のアルゴリズムよりも今回提案したアルゴリズムのほうが，より高速に計算を行うことが出来たことがわかる．

ただし，ロバストなアルゴリズムでは，実行時間にほとんど差が出なかった．一方で，adaptive なアルゴリズムは，Shewchuk のアルゴリズムに比べて大分速く処理を行うことが出来たので，adaptive な計算を行うときには，今回提案したアルゴリズムを使うほうがよいと言える．

さらに，Shewchuk のアルゴリズムでは，ロバストなアルゴリズムと adaptive なアルゴリズムで実行時間に大きな差があるのに対し，今回提案した手法では，ロバストなアルゴリズムと adaptive なアルゴリズムで実行時間の差があまりなかった．

これは、最終的に行うロバストな計算のために必要となる計算を，事前の計算結果をうまく再利用することが出来たからである．

よって，今回提案した adaptive なアルゴリズムは adaptive でありながら，ロバストなアルゴリズムを兼ねていると言える．

参考文献

[1] 大石進一:“精度保証付き数値計算”，コロナ社,(2000),

[2] S.M. Rump, T. Ogita, and S. Oishi:“Accurate Floating-Point Summation”, Technical Report 05.1, Faculty of Information and Communication Science, Hamburg University of Technology, 2005,

[3] Jonathan Richard Shewchuk:“Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates”, Discrete & Computational Geometry 18:305-363 1996,