6J-09

GPUによる幾何計算のための空間分割アルゴリズムについて

石河　孝太†　山本　修身‡
名城大学理工学研究科情報工学専攻†　名城大学理工学部情報工学科‡

1　はじめに

本稿では，計算幾何学 [1] における代表的な問題に対して，空間分割を用いて解くことを考える．空間分割は，1つの空間をいくつかの独立した部分空間に分解するため，自然に並列計算に適した構造を得る．また，アルゴリズム自体も単純な構造になるため，GPUなどの並列度の高いハードウェアを用いることで高速に動作すると考えられる．そこで本稿では，空間分割を用いた並列幾何計算の手法を述べる．関連文献として，GPUを用いて点ボロノイ図の計算をしようとした [2] がある．

本稿で取り扱う問題は，いずれも点や線分などの図形を入力とする．例えば，3次元ユークリッド空間の点は3つの実数値の組み合わせとして表現され，2次元ユークリッド平面の線分は2つの2次元座標値の組み合わせとして表現される．このような入力として表現された数値の組み合わせに対して，有限回の操作を繰り返し行うことで意味のある図形を得る（Fig. 1 参照）．通常，図形を取り扱う幾何計算では，空間に配置された n 個の入力点集合 $D = \{d_1, d_2, \ldots, d_n\}$ に対して，ある条件 C を満たした入力点の組み合わせを全て列挙する．例えば，C を満たした二つの入力点の組み合わせを全て列挙したいならば，単純に入力点同士の全ての組み合わせについて C を満たすかどうか調べればよい．しかしその場合，$O(n^2)$ の計算量が必要となり，入力点の数が多いと，多大な計算時間が必要になる．したがって，いちいち調べる必要のない不要な入力点同士の組み合わせを見つけることが幾何計算での主要な部分となる．

2　空間分割アルゴリズムの概略

本稿で提案する空間分割を用いたアルゴリズムは，空間を分割して部分問題を構成することにより全体の解を得る．その際，部分空間との関係について不要な入力点を削除することで，結果的に調べる必要のある入力点同士の組み合わせを少なくする（Fig. 2 参照）．しかしこの場合，二つの入力点が限りなく近くに配置される可能性があり，それだけ細かい分割が必要になると考えられる．そこで本稿で取り扱う問題は，入力点に解像度の存在を仮定する．例えば，平面上の異なる2点はある一定以上

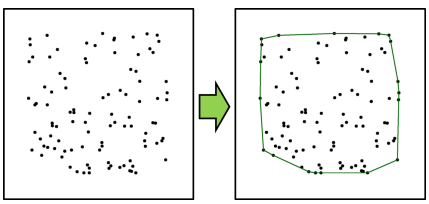

Fig. 1　凸包問題における入力（左）と出力（右）．

On a space division algorithm for geometric computation using GPU

† Kota Ishikawa, Division of information Engineering, Graduate School of Science and Technology, Meijo University
‡ Osami Yamamoto, Department of Information Engineering, Faculty of Science and Technology, Meijo University

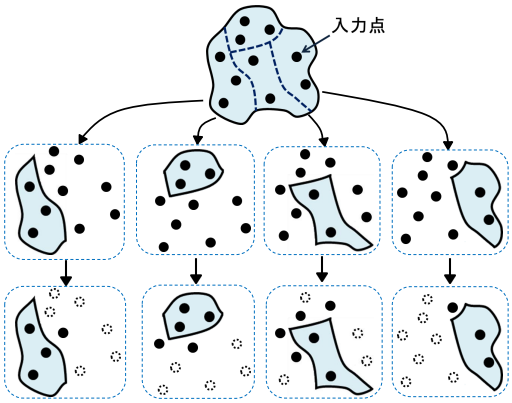

Fig. 2　空間分割を行う際の入力点の削除．部分空間との関係について不要な入力点を削除する．

の距離があると仮定する．しかしながら，我々が普段取り扱っている計算機上での数値は，有限ビット長で表現されており，表現可能な細かさの限界がある．

線分交差列挙問題を例にして，アルゴリズムの概略を述べる．線分交差列挙問題は平面上に配置された n 本の線分を入力として，それらによって作られる交点（すなわち交点を構成する2つの線分の組み合わせ）を全て列挙する問題である．通常のアルゴリズムでは，一定方向（例えば x 軸方向）に見ていくことにより，意味のある線分の組み合せを見つけて問題を解く．ここでは，交点が存在し得る初期領域 R を決めて，この領域を階層的に分割する．その際，それぞれの細分化された領域との関係について不要な線分を削除する．以下にアルゴリズムを単純化した疑似コードを示す：

```
function Div(r, D = {d_1, ..., d_k}){
    if D = ∅ or #D = 1 then return ∅
    else if D = {d_1, d_2} then
        if d_1, d_2 が交差する then return {d_1 ∧ d_2}
        else return ∅
    } else {
        領域 r を r_1, ..., r_j に分割する．
        return ⋃_{i=1}^{j} Div(r_i, D_i)
    }
}
```

ただし，D_i は領域 r_i に関係する線分だけを D から抽出したものとする．また，$d_1 \wedge d_2$ は2本の線分の交点を表す．

3　並列計算のための空間分割アルゴリズム

再び線分交差列挙問題を例にして，並列計算に適した構造に変換する．並列計算では，複数のプロセッサが独立にほぼ同一の処理を行う．例えば，配列の要素として数を格納した n 次元数ベクトル a, b の和 $a + b$ を求めるとき，第 i 成分同士の和 $a[i] + b[i]$ の計算を i 番目のプロセッサが行う．本稿では，i 番目の要素を，線分 l_i と l_i に関係する部分空間 S_i の組 (l_i, S_i) とした配列 A を考え，各 i に対して1つのプロセッサが割り当て

表1　点ボロノイ図を計算した際の実行環境

OS	Windows7 64 ビット
CPU	Intel(R) Xeon(R) CPU E31245 @ 3.30GHz
GPU	NVIDIA Tesla K20c

られるとする（Fig. 3 参照）．このとき Fig. 2 を模倣するならば，空間を分割する際，A の大きさは 4 倍になるだろう．その後，4 倍になった A から不要な線分を削除することになる．また，空間分割を行う際，Fig. 3 のような配列の構造を保つためには，どうしてもそれぞれの部分空間に関係する線分の本数を知る必要がある．線分の本数を知ることはまた，交点を求める処理を実行するために必要である．これを解決するための手段の 1 つとして，Pointer Jumping [3] を用いた方法が考えられる（Fig. 4 参照）．また，Fig. 5 に示すように不要な線分の削除も Pointer Jumping を用いればうまくいく．

4　GPU を用いた実験と今後の方針

ここでは，n 個の入力点に対する平面上の点ボロノイ図（Fig. 6 左図参照）を取り扱う．Fig. 7 は，CPU または GPU を用いて測定したランダムに発生させた入力点の数に対する点ボロノイ図生成の実行時間である．CPU を用いた点ボロノイ図生成には Triangle[1] プログラムを，GPU には本稿での空間分割アルゴリズムを利用した．また，その際の実行環境を表 1 に示す．Fig. 7 より，入力点の数が多いとき，CPU よりも GPU を用いた場合

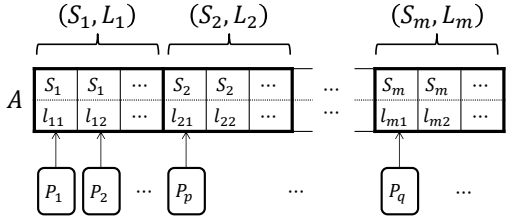

Fig. 3　線分 l と l に関係する部分空間 S の組 (l, S) を要素とした配列．L_i は部分空間 S_i に関係する線分の集合を表し，同じ S_i を持つ配列の要素は連続して配置させる．また，P_j は j 番目のプロセッサを表す．

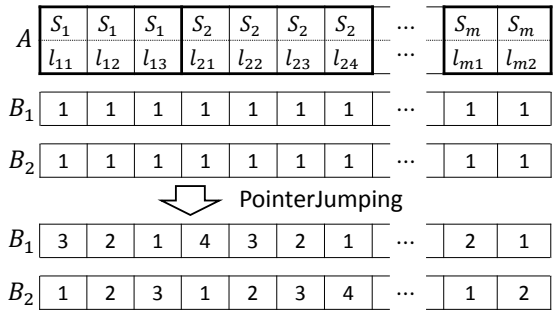

Fig. 4　PointerJumping によるそれぞれの部分空間に関係する線分の本数の算出．全ての要素に 1 を格納した A と同じ大きさの配列 B_1, B_2 を用意し，B_1, B_2 に対して PointerJumping を適用する．$A[i]$ が持つ部分空間 S_i に関係する線分の本数が知りたいとき，$B_1[i] + B_2[i] - 1$ で算出できる．

[1] https://www.cs.cmu.edu/~quake/triangle.html

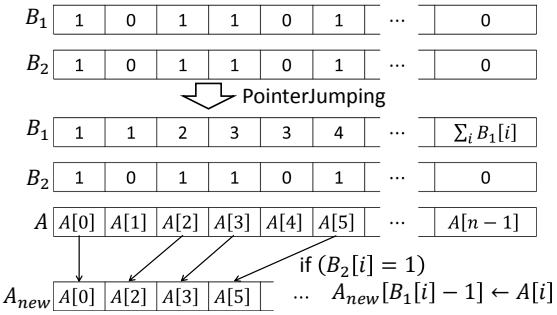

Fig. 5　PointerJumping による不要な線分の削除．A と同じ大きさの配列 B_1, B_2 を用意し，$A[i]$ が不要な要素であるとき $B_1[i], B_2[i]$ に 0 を格納し，それ以外の要素には 1 を格納する．B_1 に対して PointerJumping を適用し，$B_2[i] = 1$ のとき $A_{new}[B_1[i] - 1]$ に $A[i]$ を格納すれば，結果的に不要な線分が削除される．

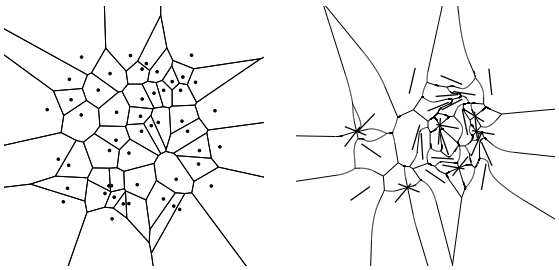

Fig. 6　50 個の点が与えられたときの点ボロノイ図（左）と 30 本の線分が与えられたときの線分ボロノイ図（右）．

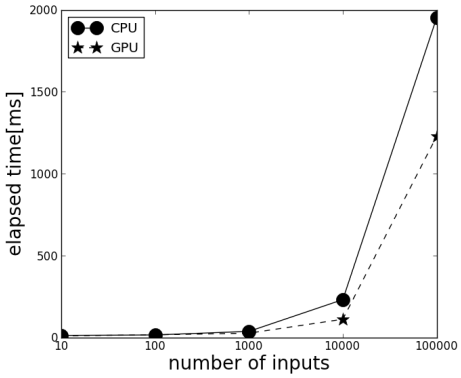

Fig. 7　入力点数に対する点ボロノイ図生成の実行時間．

の方が，わずかに実時間の面で速く動作していることが分かる．

しかしながら，NVIDIA が提供する CUDA には共有メモリなどの高速化するための機能が備わっており，本実験ではそれを使っていない．そこで先ず，CUDA の性能を最大限に活かせるプログラムに改良し，再度計算機実験を行う予定である．また，本稿での考え方に基づいた個別の問題の解法を考えるとともに，より広い応用を探っていく予定である．

参　考　文　献

[1] Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars, Computational Geometry : Algorithms and Applications Third Edition, Springer(2008).

[2] 河野勇人：GPGPU によるボロノイ図計算アルゴリズムの効率化に関する研究，名城大学大学院理工学研究科情報工学専攻修士論文，2013.

[3] T. コルメン，C. ライザーソン，R. リベスト：アルゴリズムイントロダクション [第 3 巻]．浅野哲夫，岩野和生，梅尾博司，山下雅史，和田幸一訳，初版，近代科学社，1995.