6H-07

単純なデータ構造を用いたCUDAのためのDelaunay三角形分割計算手法

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　Delaunay三角形分割と空間分割法の概要

点ボロノイ図とは，平面上の n 個の入力点に対して，「平面上の各点はどの入力点に最も近いか」を基準に平面を区分けした図である．これに対して Delaunay 三角形分割は，点ボロノイ図の隣り合った領域にある入力点同士を辺で結んだ双対図形である（Fig.1 左図）．Delaunay 三角形分割の構成では，点ボロノイ図における 3 つの領域が重なる点（ボロノイ頂点）を列挙する．Delaunay 三角形分割の計算アルゴリズムには，逐次添加法や分割統治法などがあり，時間計算量は $O(n \log n)$ である．

本稿では，GPU 上で Delaunay 三角形分割を計算する空間分割法の実装方法について考える．空間分割法は，空間を細分化することによって部分問題を構成する手法である [1]．ただし，空間の細分化を繰り返すことによって，平面上の任意の 2 点は異なる部分空間に含まれるようにすることができると仮定する．Delaunay 三角形分割はボロノイ頂点を列挙すればよいから，空間を細分化すれば，「部分空間に含まれるボロノイ頂点を全て列挙せよ」という部分問題がいくつか構成される．ただし各部分問題について，部分空間内のボロノイ頂点の列挙に関わらないと断言できる不要な入力点は排除しておく（Fig.1 右図）．また，部分空間に対して階層的な細分化を行えば，各部分空間に含まれるボロノイ頂点は少なくなるため，それだけ多くの入力点が排除され，簡単な部分問題がいくつも構成される．Delaunay 三角形分割における各部分問題は，入力点同士の単純な組み合わせ計算によって解かれる．

本稿では，GPU コンピューティング環境 CUDA を用いた空間分割法の実装方法について考える．CUDA は，大量のスレッドを起動させて同じ命令を実行させる．例えば，n 次元数ベクトル a, b を，n 個の要素から成る一次元配列で表現する．このとき n 次元ベクトルの和 $a+b$ を求めるならば，i 番目のスレッドが，第 i 成分同士の和 $a[i]+b[i]$ の計算を行う．

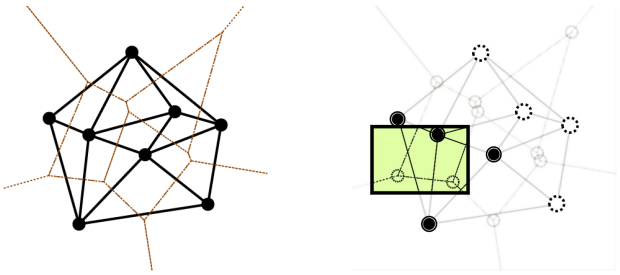

Fig. 1　点ボロノイ図と Delaunay 三角形分割の関係（左図）．空間の細分化によって得られる部分問題（右図）．

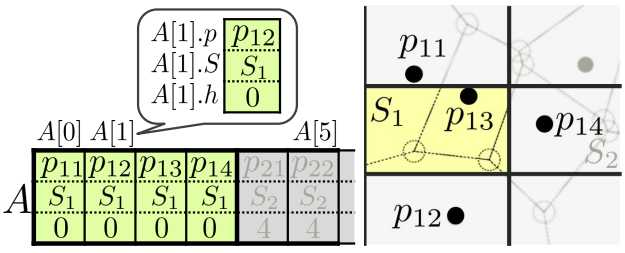

Fig. 2　GPU を用いた空間分割法のためのデータ構造．

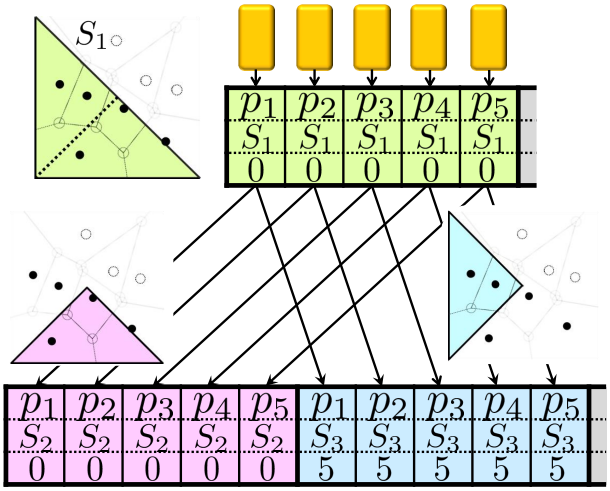

Fig. 3　空間の細分化で各スレッドが行う処理．

2　GPU を用いた空間分割法の実装について

Delaunay 三角形分割の部分問題は Fig.1 右図のように，部分空間と入力点集合の部分集合から成る．そこで，空間分割法で用いるデータ構造には Fig.2 のような一次元配列 A を用いる．ただし，A は次の条件 C：

同じ部分空間を持つ配列の要素は連続して配置する．

を満たすものとする．この条件 C によって部分問題は配列の上で一つにまとまる．また，A の各要素における 3 番目の成分 $A[\cdot].h$ には，部分問題の先頭に当たる A の上のインデックス番号を格納する．例えば，Fig.2 における部分問題の先頭のインデックス番号は 0 であり，その部分問題に関わりのある配列の要素は全て $A[\cdot].h$ に 0 を持つ．

先ず，A の各要素に対して一つずつスレッドを割り当てたとき，階層的な空間の細分化に対応する，各スレッドが行うべき処理を考える．細分化の後の配列も C を満たすようにするには，各スレッドは Fig.3 のような処理を行うことになるだろう．しかし，各スレッドは細分化の後の配列におけるインデックス番号を，Fig.3 の状態になるように計算しなければならない．そのためにはどうしても，各部分問題の入力点の個数の情報が必要になってくる．ところが C によって，A の上での各部分問題の先頭 *head* と末尾 *tail* さえ分かれば，各部分問題の入力点の個数は

On computation of Delaunay triangulation on CUDA using a simple data structure

† 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

$tail - head + 1$ として計算される．そこで，$A[i-1].h \neq A[i].h$ となる i 番目のスレッドに，同じ部分空間を持つ他のスレッドに対して，i を部分問題の先頭のインデックス番号として伝達させる．部分問題の末尾についても同様に考える．具体的には，Fig.4 のような処理を各スレッドに実行させる．同じ部分問題の配列の要素は同じ $A[\cdot].h$ を持っているから，そのインデックス番号を用いて H, T にアクセスすれば，全てのスレッドは自分が指している部分問題の先頭および末尾を知ることができる．

これで Fig.3 のような細分化の処理を実行することができ，次に行うべき処理は，各部分空間について不要な入力点を排除することであると分かる．各部分空間について不要な入力点を見つけるためのアルゴリズムに，[2] の Leaching アルゴリズムがある．一方，特定の配列の要素を削除するアルゴリズムには，Fig.5 のような並列和を用いたパッキングがある [3]．パッキングの後の配列も C が満たされるため，再び Fig.3 のような細分化の処理に繋げることができ，単純な繰り返し処理で空間分割法が実装される．

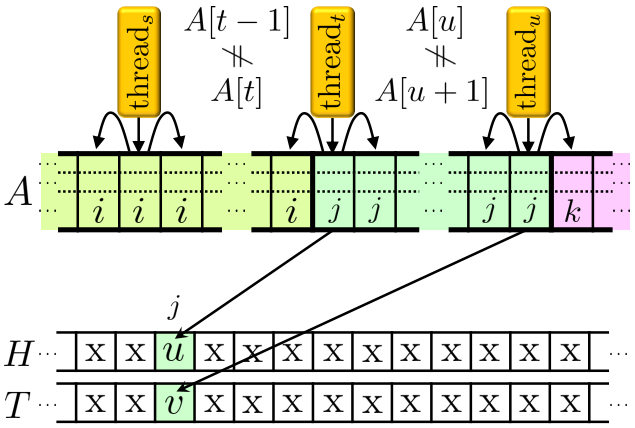

Fig. 4　特定のスレッドが部分問題の先頭または末尾を他のスレッドに伝達するアルゴリズム．

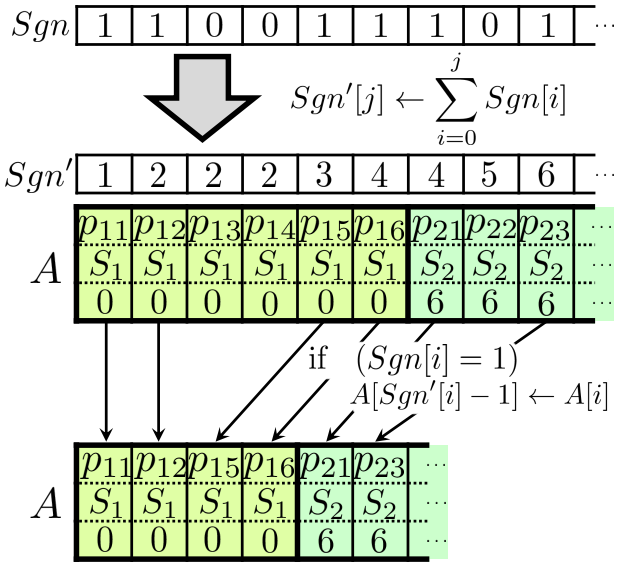

Fig. 5　$Sgn[i]$ には，$A[i].p$ が $A[i].S$ について不要な入力点であると分かったとき 0 を格納し，そうでないとき 1 を格納する．Sgn の並列和 Sgn' を利用すると，特定の配列の要素を削除することができる．

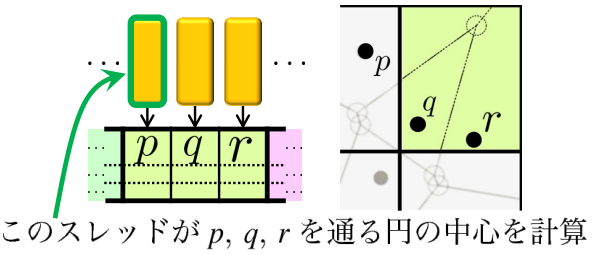

Fig. 6　入力点の個数が 3 個の部分問題におけるボロノイ頂点の計算．

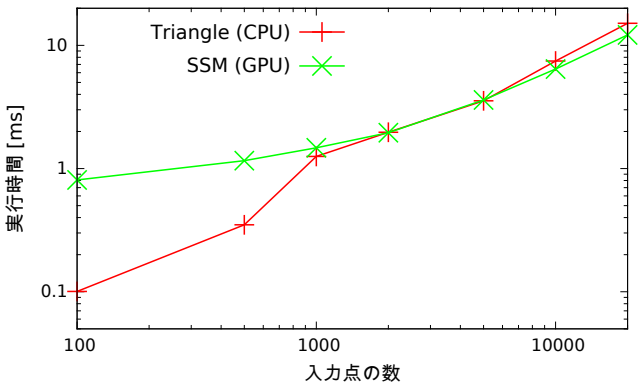

Fig. 7　CPU および GPU を用いた Delaunay 三角形分割の計算時間．

繰り返し処理の中，これ以上細分化する必要がない部分問題が構成される場合がある．実際，入力点の個数が 3 個以下である部分問題は，部分空間にボロノイ頂点を高々一つしか含まない．そこで，そのような部分問題も配列から削除する必要があるが，これは結局 Fig.5 のようなパッキングを行うだけである．ただし，入力点が 3 個である部分問題を配列から削除する際，その部分問題の先頭を指しているスレッドに，3 つの入力点を通る円の中心を計算させる（Fig.6 参照）．

3　計算機実験と今後の課題

本稿では最後に，CUDA を用いた計算機実験について述べる．Fig.7 は，ランダムに発生させた入力点に対し，CPU および GPU を用いて測定した Delaunay 三角形分割の計算時間である*．ただし，CPU 上では高速に Delaunay 三角形分割を計算する Triangle プログラム†を用い，GPU 上では本稿の空間分割法を用いた．20,000 点を入力点として与えたとき，GPU 上で動かしたプログラムの実行速度は CPU と比べて 1.2 倍高速化した．

今後の課題としては，空間分割法の計算の複雑さを明らかにすることが挙げられる．また，空間分割法には数値計算の安定性に欠ける部分があるため，それも対処しなければならない．

参　考　文　献

[1] 石河孝太, 山本修身: 空間分割法を用いた Delaunay 三角形分割アルゴリズムの GPGPU による実現について. 平成 28 年度電気・電子・情報関係学会東海支部連合大会予稿集. H3-2, 2016.
[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] G. E. Blelloch.: Prefix sums and their applications, Carnegie Mellon University, 1990.

* OS: Linux 64-bit, CPU: Intel(R) Xeon(R) CPU E5-1620 v3 @ 3.50GHz, GPU: NVIDIA Tesla C2075 (2,496 CUDA cores).
† https://www.cs.cmu.edu/~quake/triangle.html