Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 8, Number 2, pages 122–128, July 2019

A GPU Implementation of Non-photorealistic Stained Glass Image Generation using Voronoi Diagrams

Hironobu Kobayashi, Yasuaki Ito, and Koji Nakano
Department of Information Engineering, Hiroshima University
Kagamiyama 1-4-1, Higashi-Hiroshima, 739-8527 Japan
Email: {hironobu, yasuaki, nakano}@cs.hiroshima-u.ac.jp

***Abstract*—In this paper, we propose a high quality stained glass image generation. Our proposed method converts automatically the original image into a high quality stained glass image. In the method, we use Voronoi diagrams to reproduce a stained glass such that Voronoi cells and edges represent colored glasses and leads that bond the glasses, respectively. In order to obtain a high quality stained glass image, we merge Voronoi cells and move Voronoi seeds using a local search technique. By varying the size of Voronoi cells in a stained glass image, a stained glass image well-represents local characteristics of the original image. Also, by moving Voronoi seeds to fit Voronoi cells to the original image, we obtain a high quality output image. However, the computing time cost is very high in our proposed method. Therefore, we also propose a GPU implementation to accelerate the computation. Our experimental result shows that the proposed GPU implementation on NVIDIA Tesla V100 attains a speed-up factor of 362 over the sequential CPU implementations.**

***Index Terms*—Stained glass image generation, Human visual system, GPU, Parallel processing**

I. はじめに

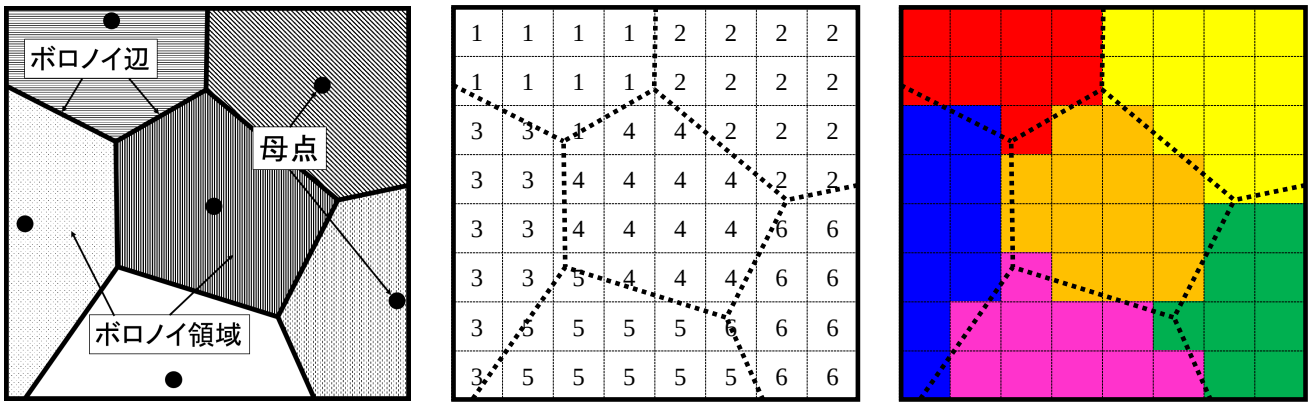
ステンドグラスは着色されたガラスの小片を鉛で結合し，絵画的なデザインを表現したものである．ステンドグラス風画像生成は，油絵，タイルアート，モザイクアートなどの芸術的表現に似た画像を生成するノンフォトリアリスティックレンダリングの 1 つとして知られている．これまでにコンピュータを使用したステンドグラス風画像生成に関するいくつかの研究が行われている [1]．これらの研究は，ステンドグラス風画像を生成するためにイメージセグメンテーション [2]–[5] やボロノイ図 [6]–[8] を使用している．本研究ではボロノイ図に基づくステンドグラス風画像の生成を行う．

平面上の 2 つの点 p と q のユークリッド距離を $d(p,q)$ とする．平面上の母点と呼ばれる点の集合 P のボロノイ図は，平面をボロノイ領域と呼ばれる領域 $V(p)$ $(p \in P)$ に分割することで生成される．平面上の点の集合を Q として，$V(p)$ は $V(p) = \{q \in Q | d(p,q) \leq d(q',q) \text{ for all } q' \in P\}$ で定義される．つまり，$V(p)$ は母点集合 P に含まれる母点の中で最も p に近い平面上の点の集合である．点 p のボロノイ領域 $V(p)$ の境界は線分であり，各線分は p とそれに隣接する母点との垂直二等分線である．ボロノイ領域の境界を表す線分をボロノイ辺と呼ぶ．本研究ではボロノイ領域とボロノイ辺に色を付けることでステンドグラスの着色ガラスや鉛を表現する．図 1(a) にボロノイ図の例を示す．

論文 [7] では，ボロノイ図を用いてステンドグラス風画像を生成している．彼らの提案手法では，母点を隣接する 8 ピクセルのうちの 1 つに移動させることによってボロノイ領域を調整し，誤差を減少させる．ただし，計算時間を短縮するために母点移動は 2 段階で行う．第 1 段階では実際に母点を動かすのではなく，ボロノイ領域が平行移動すると仮定して近似計算を行い，隣接する母点の競合状態を無視して全ての母点移動を同時に行う．第 2 段階では各母点に対する調整が逐次的に行われ，実際に母点を動かして誤差を計算し，良くなる方向に母点を移動させる．この方法によって出力されるステンドグラス風画像は入力画像をきれいに表現しているが，ボロノイ領域のサイズは入力画像の特徴に関係なくほぼ同じである．論文 [8] では，重心ボロノイ図を使用し，画像の特徴に合わせてボロノイ領域のサイズを変えることで，局所的な特徴がよく表現されている．しかし，誤差計算や入力に近づけるための母点移動は行っていないため，入力画像の滑らかなエッジは表現できない．また，論文 [6] では，曲線状のガラスを用いたステンドグラスを表現するために重み付きボロノイ図が使用されている．

本研究の主な貢献は，ボロノイ図を用いて入力カラー画像を再現するステンドグラス風画像を生成する方法を提案することである．特に，高品質な画像を生成するために，デジタルハーフトーン [9], [10] でも利用されている人間の視覚特性に基づいたアイデアを紹介する．また，画像内におけるボロノイ領域のサイズを変更し，細かい領域と粗い領域をそれぞれ用いることで，入力画像の局所的な特徴を表現する．さらに，[7] と同様に，母点を調整することで入力画像に近づけていく．本研究の第 2 の貢献は，計算を高速化するために GPU（Graphics Processing Unit）を用いて上記のステンドグラス生成方法を実行することである．GPU は内部に多数のコアを持ち，並列処理による計算の高速化が可能で，近年ではグラフィック処理だけでなく汎用計算にも使用されている．実験の結果，GPU 実装が逐次 CPU 実装より最大 362 倍，36 スレッドの並列 CPU 実装より最大 54 倍の高速化を達成した．

本論文は以下のように構成されている．II 節では，ボロノイ図に基づくステンドグラス風画像の生成について説明する．次に，III 節で計算を高速化するための GPU 実装について説明する．IV 節では，CPU 実装と比較して GPU 実装の性能を評価する．最後に V 節でまとめを行い，本研究を締めくくる．



(a) ボロノイ図　(b) 離散ボロノイ図　(c) ステンドグラス風画像

Fig. 1. ボロノイ図，離散ボロノイ図，ステンドグラス風画像

II. ステンドグラス風画像生成の提案手法

まず，ボロノイ図を用いて離散ボロノイ図とステンドグラス風画像について説明する．離散ボロノイ図について，ボロノイ図の各母点がピクセルの中心に位置するようにボロノイ図を離散ボロノイ図にマッピングし，各ボロノイ領域内のピクセルには，各母点に割り当てられた一意の番号である母点 ID を割り当てる (図 1(b))．ステンドグラス風画像は離散ボロノイ図から生成され，同じボロノイ領域内のピクセルは全て同じ色で塗られる (図 1(c))．以下では，生成されたステンドグラス風画像の良さの定義と，新しいステンドグラス風画像の生成方法の提案を行う．

A. 人間の視覚特性に基づいた出力画像の評価基準

人間の視覚特性に基づいた入力画像との誤差を紹介する．その後，ステンドグラス風画像生成のアルゴリズムを紹介する．

最初にグレースケール画像における考えを説明し，次にそれをカラー画像に拡張する．大きさ $N \times N$ の入力画像 $A = (a_{i,j})$ を考える．ただし，$a_{i,j}$ は位置 (i,j) $(1 \leq x, y \leq N)$ の輝度値であり，$[0,1]$ の実数値で与えられる．入力画像 A を再現するステンドグラス風画像 $B = (b_{i,j})$ を生成するとき，出力画像 B の良さは，人間の視覚系の特性を近似するガウシアンフィルタを使用して計算することができる．ガウシアンフィルタ $G = (g_{p,q})$ はサイズが $(2w+1) \times (2w+1)$ の二次元対称行列である．ここで，$g_{p,q}$ $(-w \leq p, q \leq w)$ はそれぞれ非負の実数であり，それらの合計が 1 になるように 2 次元ガウス分布によって決定される．言い換えれば，ガウス分布のパラメータ σ と $\sum_{-w \leq p,q \leq w} g_{p,q} = 1$ を満たす定数 s を用いて，$g_{p,q} = s \cdot e^{-\frac{p^2+q^2}{2\sigma^2}}$ で求めることができる．よって，出力画像にガウシアンフィルタを適応することによって得られるぼかし画像を $R = (r_{i,j})$ とすると，以下の式で求めることができる．

$$r_{i,j} = \sum_{-w \leq p,q \leq w} g_{p,q} b_{i+p,j+q} \quad (1 \leq i, j \leq N).$$

$\sum_{-w \leq p,q \leq w} g_{p,q} = 1$ で，$g_{p,q}$ は非負なので，各 $r_{i,j}$ は $[0,1]$ の実数となる．したがって，ぼかし画像 R はグレースケールである．入力画像 A とぼかし画像 R の差が十分に小さければ出力画像 B は入力画像 A をよく近似できているといえる．ここで，出力画像 B の各ピクセル位置 (i,j) における誤差 $e_{i,j}$ は以下のように定義される．

$$e_{i,j} = a_{i,j} - r_{i,j}, \tag{1}$$

また，誤差総和は以下のように定義される．

$$\mathrm{Error}(A,B) = \sum_{1 \leq i,j \leq N} |e_{i,j}|. \tag{2}$$

ガウスフィルタは人間の視覚系の特性を近似しているため，$\mathrm{Error}(A,B)$ が十分小さい場合，出力画像 B は元の画像 A をよく再現している．次に，グレースケール画像の誤差計算をカラー画像に拡張する．本研究では，各値が赤，緑，青を表す $[0,1]$ の範囲の 3 つの実数で指定される RGB 色について考える． カラー画像の場合，ぼかし画像 R および式 (1) の誤差は各色について独立に計算される．すなわち，各色について，ガウシアンフィルタが適用され，誤差が計算される．$e^R_{i,j}$,$e^G_{i,j}$,$e^B_{i,j}$ をそれぞれ，ピクセル位置 (i,j) での赤，緑，青の誤差を表すものとすると，式 (2) は次のように各色の値の合計に拡張される．

$$\mathrm{Error}(A,B) = \sum_{1 \leq i,j \leq N} (|e^R_{i,j}| + |e^G_{i,j}| + |e^B_{i,j}|). \tag{3}$$

本研究で生成されるステンドグラス風画像において，赤，緑，および青の色はそれぞれ１６色で，各ガラスの色はその合計の４０９６色から１つから選択される．

B. ステンドグラス風画像生成のアルゴリズム

この節では，ステンドグラス風画像の生成アルゴリズムを提案する．このアルゴリズムは，入力画像が与えられたとき，前の節で示した誤差総和が最小となるステンドグラス風画像を出力する．提案アルゴリズムは以下の３つのステップで構成される．

Step1: 初期のステンドグラス風画像を生成する
Step2: 隣接するボロノイ領域が同じ色の場合，それらの母点を 1 つにマージする
Step3: 局所探索による母点位置の反復調整

各ステップで得られた画像を図 2 に示し，その詳細を以下に示す．ステップ 1 では，初期のステンドグラス風画像を生成

(a) 入力画像

(b) ステップ 1

(c) ステップ 2

(d) ステップ 3 (出力画像)

Fig. 2. 1024 × 1024 の大きさの Lena 画像における各ステップで得られた画像

する．まず，入力画像と同じ大きさの画像の各ピクセルに確率 p で母点をランダムに配置する．この p を使ってボロノイ領域のサイズを制御することができる．p が大きいと母点の密度が高いため，ボロノイ領域は小さくなる．母点の配置後に各ボロノイ領域に色を付けることによってステンドグラス風画像が生成される．各ボロノイ領域の色は，各ボロノイ領域内の画素に対応する入力画像の画素に近い色が割り当てられる．図 2(b) は 1024×1024 の大きさの Lena 画像 [11] を入力とし，母点を配置する確率を $p = \frac{1}{100}$ としたときの初期のステンドグラス風画像である．このときのボロノイ領域の数は 10433 個である． ステップ 2 では同じ色を持つ隣接ボロノイ領域を 1 つのボロノイ領域にマージする．具体的には，各ボロノイ領域について，隣接ボロノイ領域が同じ色を持つ場合，それらの隣接母点を削除して母点をそれらの重心に移動する．その後，ステンドグラス風画像を再構築し，各ボロノイ領域の色が図 3 に示すように更新される．全てのボロノイ領域に対するこのプロセスは数回繰り返し実行される．この繰り返しの回数でボロノイ領域の大きさを制御できる．本研究ではこのマージ処理を 5 回繰り返す．図 2(c) はステップ 1 で得られた初期のステンドグラス風画像のボロノイ領域の数をマージによって 2141 に減らしたものである．図より，ステップ 2 終了時で

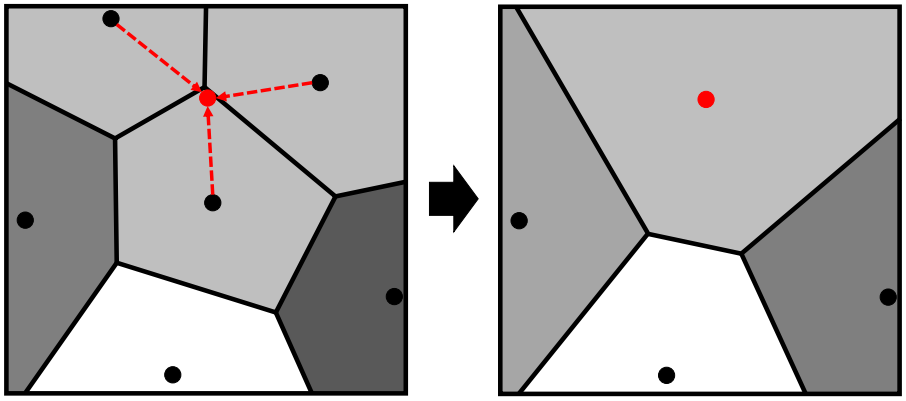


Fig. 3. ボロノイ領域のマージ

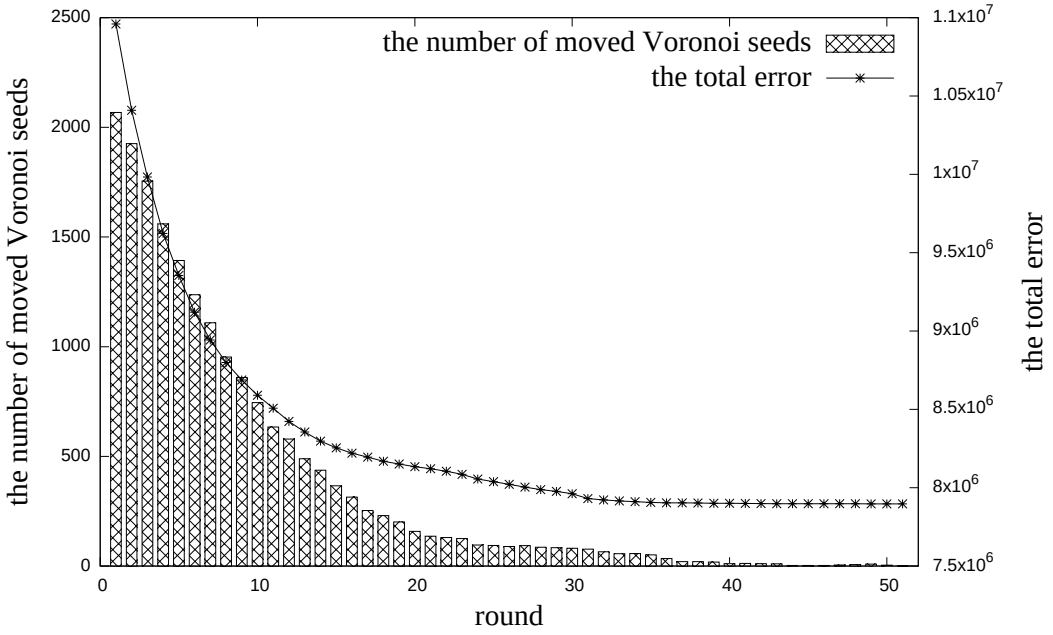
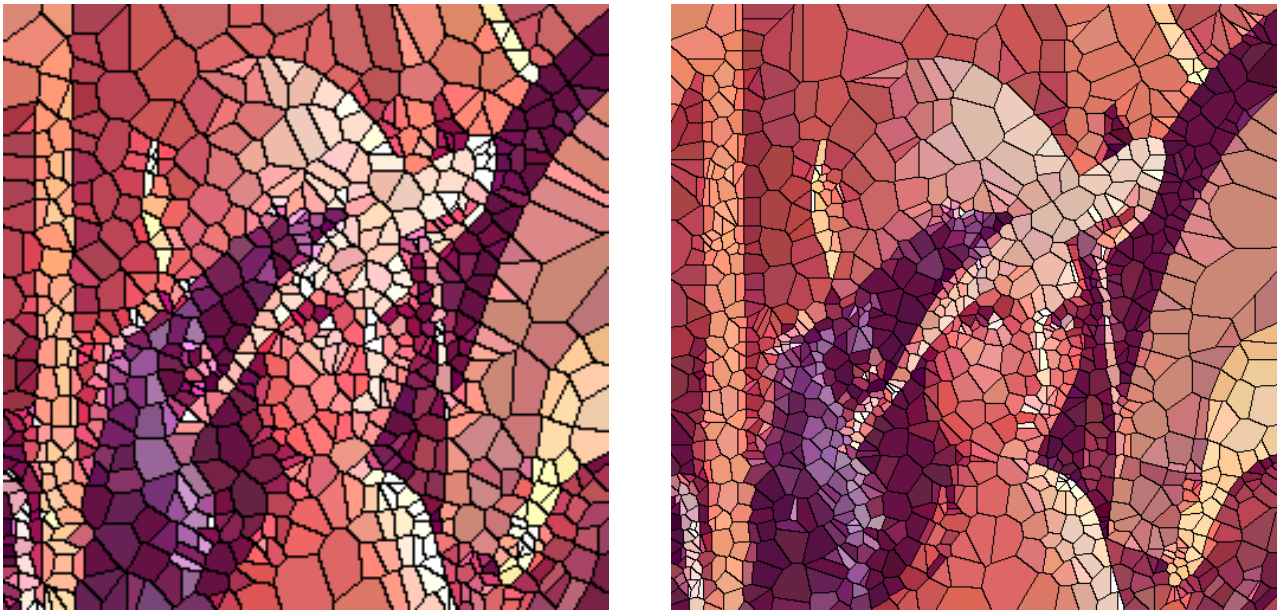


Fig. 4. ステップ 3 において動いた母点数と誤差総和の変化

は，まだ肩や帽子などの輪郭をうまく表すことができていない．ステップ 3 では，貪欲法に基づく局所探索を繰り返すことによって，母点を反復的に移動させて式 (3) の誤差総和を最小にする．局所探索では，各母点の移動は隣接する 8 ピクセルに制限され，母点を誤差総和が最も小さくなる隣接ピクセルに移動する．移動によって誤差を減らすことができない場合，母点は移動させない．ステップ 3 において，全ての母点について局所探索を行うことを 1 ラウンドとすると，ラウンドは母点の移動が行われなくなるまで，すなわち，それ以上の改善が不可能になるまで繰り返される．図 2(d) はステップ 3 で得られた画像で，これは提案手法で生成されたステンドグラス風画像である．この画像では明らかに長い直線や曲線がよく表現されている．また，髪の毛や目などの細かい部分を表現するために，小さなボロノイ領域が使用されている．一方，背景の粗い部分は大きなボロノイ領域で構成されている．このステンドグラス風画像を得るために，上記のラウンドを 51 回繰り返した．図 4 は移動した母点の数とステップ 3 の誤差総和のグラフを示している．グラフより，最初の数ラウンドでほとんどの母点が移動し，誤差総和が急激に減少している．40 ラウンドの後，ほとんどの母点は動かず，誤差総和もほとんど変化しない．

図 5 は，異なる画像サイズについて生成されたステンドグラス風画像である．画像サイズが小さくなった場合，母点を置く確率 p を大きくしなければ初期の母点数が少なくなってしまうため，図 5(a)，(b) ではそれぞれ $p = \frac{1}{25}$，$\frac{1}{50}$ とした．他のパラメータは上記と同じである．これらの画像の解像度は図 2(d) よりも小さいが，輪郭をよく表すことができている．

ここで，図 6 は，フリーソフトの GIMP [12] によって出力されたステンドグラス風画像である．フリーソフトは素早く出力結果が得られることを前提としているため，GIMP による出力画像は入力画像 (図 2(a)) のエッジを十分に表現できていない．また，各領域の大きさがほぼ同じで，目や



(a) 256 × 256　　(b) 512 × 512

Fig. 5. 256 × 256 と 512 × 512 の大きさの Lena 画像に対するステンドグラス風画像

髪などの細かい特徴を表現できていない．提案手法を用いて出力されたステンドグラス風画像 (図 2(d)) の方が明らかにエッジがきれいに表現されており，領域の大きさの違いにより局所的な特徴がよく表現されている．しかし，提案手法のステップ 3 における計算時間のコストは非常に高いため，次の節では計算を高速化するために母点の並列移動を行う GPU 実装を示す．

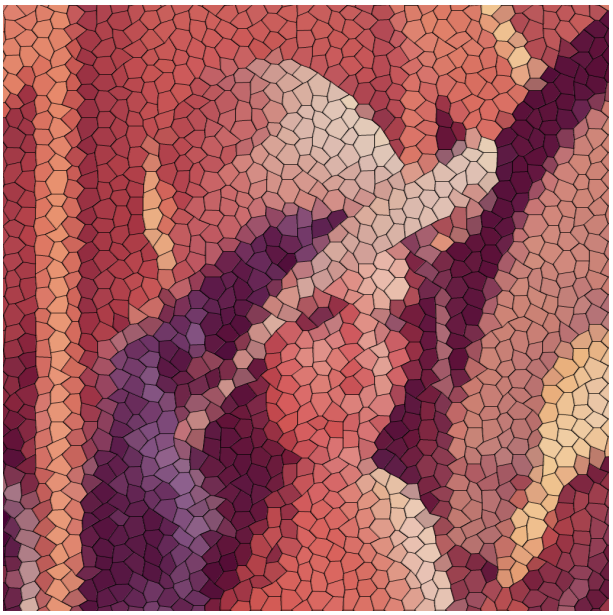


Fig. 6. GIMP によって出力されたステンドグラス風画像

III. GPU 実装

本節では，NVIDIA の GPU における CUDA について簡単に説明する．次に，II-B 節に示すステンドグラス風画像を生成する提案アルゴリズムの GPU 実装の詳細を説明する．

NVIDIA 社は，NVIDIA 社製 GPU に CUDA と呼ばれる並列計算のための統合開発環境を提供している [13]．CUDA はシェアードメモリとグローバルメモリの 2 種類のメモリを用いる．グローバルメモリは，GPU のオフチップ DRAM またはオンチップ HBM2 として実現され，大容量だがアクセス速度は遅い．シェアードメモリはオンチップメモリであり，容量は小さいが非常に高速にアクセス可能である．CUDA 並列プログラミングモデルは，グリッド，ブロック，およびスレッドと呼ばれるスレッドグループの階層構造となっている．1 つのグリッドは複数のブロックで構成され，各ブロックには同数のスレッドが存在する．ブロックはストリーミングマルチプロセッサに割り当てられ，ブロック内のすべてのスレッドが同じストリーミングマルチプロセッサによって並列に実行される．全てのスレッドはグローバルメモリにアクセスできる．また，ブロック内のスレッドは，そのブロックが割り当てられているストリーミングマルチプロセッサのシェアードメモリにアクセスできる．ブロックは複数のストリーミングマルチプロセッサに配置されるため，異なるブロック内のスレッドはシェアードメモリ内のデータを共有できない．CUDA C は C 言語を拡張したもので，プログラマーはカーネルと呼ばれる関数を定義できる．カーネルを呼び出すことによってグリッド内のすべてのブロックがストリーミングマルチプロセッサに割り当てられ，各ブロック内のスレッドが単一のストリーミングマルチプロセッサ内のプロセッサコアによって実行される．さて，GPU 上で提案アルゴリズムを実装する方法を説明する．$N \times N$ の大きさの入力画像はあらかじめグローバルメモリに格納されており，出力結果となるステンドグラ

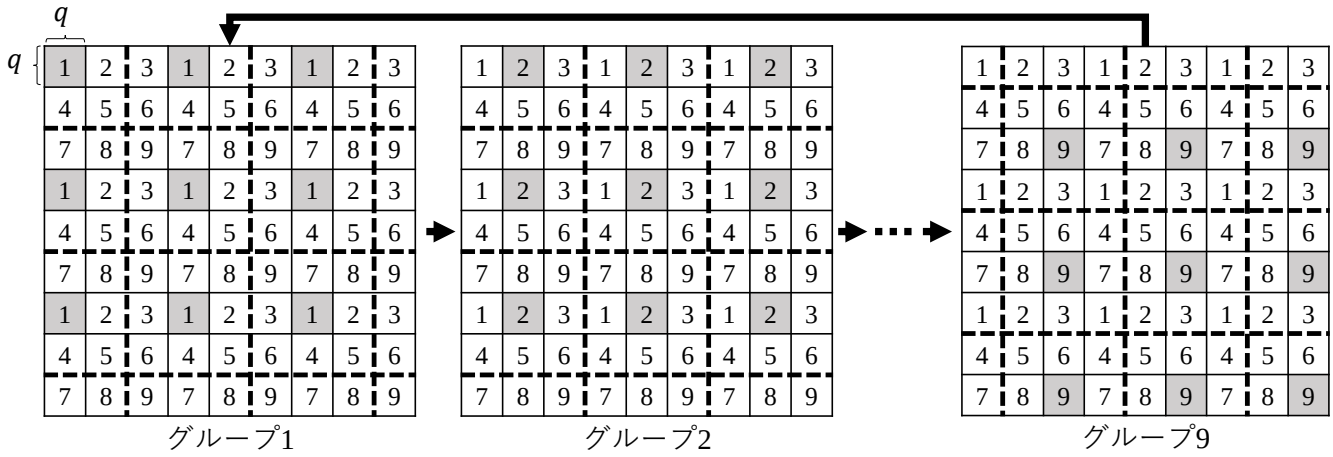


Fig. 7. 競合状態のない 9 つのサブ画像のグループと並列実行

ス風画像をグローバルメモリに書き込むと仮定する．はじめに母点をランダムに分布させるためにステップ 1 を実行するとき，各ピクセルにスレッドを割り当てる．各スレッドは，そのピクセルが母点になるか否かを確率 p で決定する．その後，離散ボロノイ図と初期のステンドグラス風画像が生成される．この過程では，各スレッドに対応するピクセル値を並列に計算するカーネルが呼び出される．また，ステップ 2 では各ボロノイ領域に対してカーネルが呼び出され，ピクセルに割り当てられている各スレッドは隣接するボロノイ領域を並列に計算する．本研究において，このカーネル呼び出しは 5 回繰り返している．ステップ 3 では，母点を 1 つずつ，隣接する 8 ピクセルの 1 つに移動させる．しかし，母点を動かしたことによって周りのボロノイ領域にも影響を与えてしまうため，単純に 2 つ以上の母点を同時に移動することはできない．したがって，このアルゴリズムを単純に並列実装することは困難である．そこで，計算を並列に実行するために，図 7 に示すように入力画像をサイズ $q \times q$ のサブ画像に分割し，サブ画像を 9 つのグループ 1, グループ 2, . . . , グループ 9 に分割する．これらのサブ画像のグループを使用して，最初に最大 $3q \times 3q$ のサイズの各ボロノイ領域の局所検索を並列に実行し，次に残ったボロノイ領域の調整を 1 つずつ実行する．具体的には，まず，$\frac{N^2}{9q^2}$ 個の CUDA ブロックを使用して，各グループのすべてのサブ画像で母点を 1 つずつ局所探索する．各 CUDA ブロックは最初に，ボロノイ領域が破線で区切られたサイズ $3q \times 3q$ の 9 つのサブ画像から突き出ていないボロノイ領域に対してのみ局所探索を実行する (図 8(a))．図 8(b) に示すようにボロノイ領域が突き出ている母点については，残った母点として記録しておく．ステップ 3 の各ラウンドでは，順番にグループ 1, グループ 2, . . . , グループ 9 の局所探索を実行する．対応するグループの局所探索を実行する CUDA カーネルが各グループに対して呼び出される．つまり，各ラウンドの計算が終了するたびに実行が同期される．その後，記録された残りの母点の局所探索が順次行われる．上記の局所探索では，グローバルメモリに格納された入力画像とステンドグラス風画像が演算中に頻繁に読み出される．したがって，それらへのメモリアクセス時間を短縮するために，それらの要素をシェアードメモリにキャッシュする．

上記の並列計算において，ボロノイ図の再構成および誤差総和の計算は，母点移動によって更新された部分的な画像についてのみ行われる．

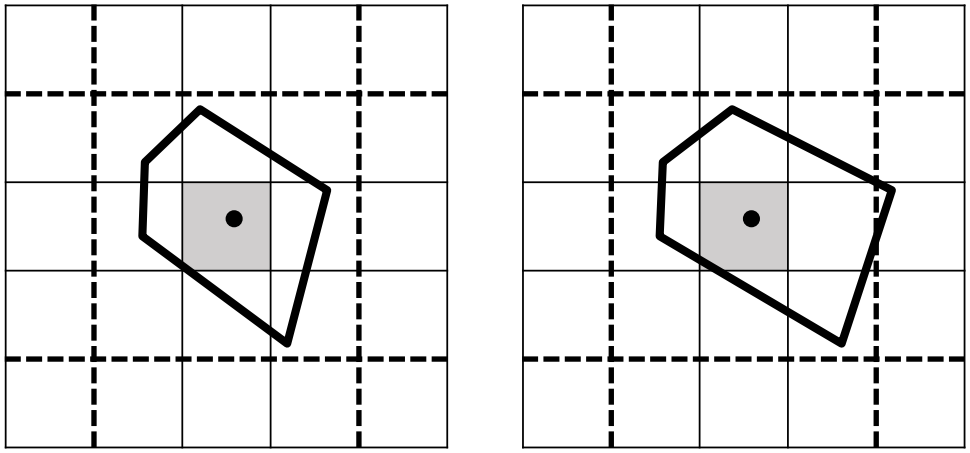
IV. 実験結果

本節では，提案した GPU 実装の計算時間の評価を行う．大きさが 256×256, 512×512, 1024×1024 の Lena 画像 [11](図 2(a)) を用いて実験を行った．ガウシアンフィルタのサイズは 7×7，パラメータ σ は $\sigma = 1.3$ に設定した．ステップ 1 における母点の分布の確率は，256×256, 512×512, 1024×1024 の画像について，それぞれ $p = \frac{1}{25}$，$\frac{1}{50}$，$\frac{1}{100}$ である．また，GPU 実装のステップ 3 において $q = 32$ としたので，サブ画像のサイズは 32×32 である．計算時間を評価するために，1.455GHz で動作する 5120 個のプロセッシングコアを持つ NVIDIA Tesla V100 を使用した．比較のために，2.6GHz で動作する 18 個のコアを持つ Intel Core i9-7980XE を用いた．この CPU のプロセッサには 18 個の物理コアがあり，それぞれがハイパースレッディング・テクノロジーによって 2 つのスレッドとして機能する．これらを使用して，逐次 CPU 実装と並列 CPU 実装をそれぞれ GPU 実装と比較して，GPU 実装の性能を評価した．並列 CPU 実装は，OpenMP [14] を使用して 36 スレッドで実装されており，GPU 実装と同様の方法で並列に処理を実行する．

表 I にステップ 2 における母点の削減と計算時間を示す．計算時間は，入力画像が CPU のメインメモリに格納されてから，出力画像となるステンドグラス風画像がメインメモリに格納されるまでの時間である．CPU 実装と GPU 実装の両方において，表に示す計算時間の合計のほとんどはステップ 3 の計算時間である．ステップ 1 とステップ 2 の計算時間は，計算時間の合計の最大 1.2%にすぎない．したがって，GPU による高速化はステップ 3 の計算の高速化とほぼ同等である．表より，GPU 実装は，逐次 CPU 実装と比較して最大で 362 倍，並列 CPU の実装と比較して最大で 54 倍の高速化を達成した．

V. まとめ

本論文では，ボロノイ図に基づくステンドグラス風画像の生成アルゴリズムとその GPU 実装を提案した．人間の視



(a) 非競合状態　　(b) 競合状態

Fig. 8. ボロノイ領域の非競合状態と競合状態

表 I
ステンドグラス風画像生成までの計算時間 (秒)

画像サイズ	ステップ 2 における母点数の減少	ステップ 3 におけるラウンド数	CPU1 (1 スレッド)	CPU2 (36 スレッド)	GPU	$\frac{\text{CPU1}}{\text{GPU}}$	$\frac{\text{CPU2}}{\text{GPU}}$
256×256	$2675 \rightarrow 812$	15	360.46	118.98	21.96	16.41	5.42
512×512	$4963 \rightarrow 1270$	27	2761.38	582.13	51.02	54.12	11.41
1024×1024	$10433 \rightarrow 2141$	51	51232.45	7661.40	141.44	362.22	54.17

覚特性に基づいた誤差を導入することにより，高品質のステンドグラス風画像を生成した．さらに，GPU 上での母点の効率的な並列調整を提案し，NVIDIA Tesla V100 にそれを実装した．実験結果は，提案 GPU 実装が逐次 CPU 実装および並列 GPU 実装に対して，それぞれ最大 362 倍，54 倍の高速化を達成した．

REFERENCES

[1] S. Battiato, G. di Blasi, G. M. Farinella, and G. Gallo, "Digital mosaic frameworks - an overview," *Computer Graphics Forum*, vol. 26, no. 4, pp. 794–812, 2007.

[2] S. B. S. Brooks, "Image-based stained glass," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 6, pp. 1547–1558, November 2006.

[3] D. Mould, "A stained glass image filter," in *Proc. of the 14th Eurographics Workshop on Rendering*, 2003, pp. 20–25.

[4] S. Seo, H. Lee, H. Nah, and K. Yoon, "Stained glass rendering with smooth tile boundary," in *Proc. of International Conference on Computational Science*, 2007, pp. 162–165.

[5] V. Setlur and S. Wilkinson, "Automatic stained glass rendering," in *Proc. of the 24th International Conference on Advances in Computer Graphics*, 2006, pp. 682–691.

[6] D. Ashlock, B. Karthikeyan, and K. M. Bryden, "Non-photorealistic rendering of images as evolutionary stained glass," in *Proc. of IEEE International Conference on Evolutionary Computation*, July 2006, pp. 2087–2094.

[7] Y. Dobashi, T. Haga, H. Johan, and T. Nishita, "A method for creating mosaic images using Voronoi diagrams," in *Proc. of Eurographics 2002*, September 2002, pp. 341–348.

[8] G. M. Faustino and L. H. de Figueiredo, "Simple adaptive mosaic effects," in *Proc. of XVIII Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'05)*, 2005, pp. 315–322.

[9] M. Analoui and J. Allebach, "Model-based halftoning by direct binary search," in *Proc. SPIE/IS&T Symposium on Electronic Imaging Science and Technology*, vol. 1666, San Jose, CA, USA, 1992, pp. 96–108.

[10] H. Kouge, T. Honda, T. Fujita, Y. Ito, K. Nakano, and J. L. Bordim, "Accelerating digital halftoning using the local exhaustive search on the GPU," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 2, p. e3781, 2017.

[11] L.-M. Po, "Lenna 97: A complete story of Lenna," http://www.ee.cityu.edu.hk/˜lmpo/lenna/Lenna97.html, 2001.

[12] "GIMP - GNU image manipulation program." [Online]. Available: https://www.gimp.org/

[13] NVIDIA Corporation, "NVIDIA CUDA C programming guide version 9.1," January 2018.

[14] "OpenMP," http://www.openmp.org/.