Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 6, Number 1, pages 22–27, January 2017

Photomosaic Generation by Rearranging Divided Images

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

***Abstract*—In this paper, we propose a photomosaic generation method by rearranging divided images. In the photomosaic generation, an input image is divided into small subimages and they are rearranged such that the rearranged image reproduces another image given as a target image. Therefore, we can consider this problem as combinatorial optimization to obtain the rearrangement which reproduces approximate images to the target image. Our new idea is that this rearrangement problem is reduced to a minimum weighted bipartite matching problem. By solving the matching problem, we can obtain the best rearrangement image. Although it can generate the most similar photomosaic image, a lot of computing time is necessary. Hence, we propose an approximation method of the photomosaic generation. This approximation method does not obtain the most similar photomosaic image. However, the computing time can be shortened considerably. Additionally, we accelerate the approximation method using the GPU (Graphics Processing Unit). The experimental result shows that the GPU implementation attains a speed-up factor of 25 times over the sequential CPU implementation.**

I. はじめに

フォトモザイクとは，小さな画像を組み合わせて一枚の大きな画像を表現する手法である．昨今では，フォトモザイクは芸術作品や広告などに用いられており，人物画を再現したフォトモザイク画像がよく見かけられる．Fig. 1 は目的画像とそれを近似したフォトモザイク画像の例である．

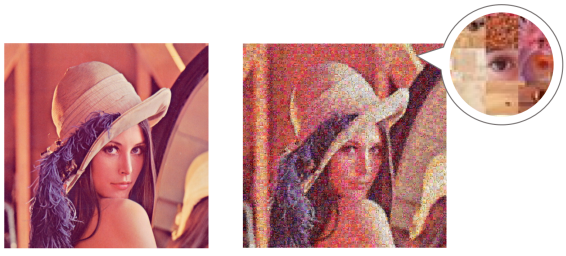

Fig. 1. フォトモザイク画像の例

フォトモザイク画像の一般的な生成手法の一つを Fig. 2 に示す．まず，複数枚の同じサイズの小画像を用意する．そして，入力画像を小画像サイズに分割し，その部分画像と色の類似度が最も高い小画像を見つける．見つけた小画像を対応する部分画像に割り当てる．これをくり返し，小画像の組み合わせで目的画像を構築することでフォトモザイク画像を生成する．

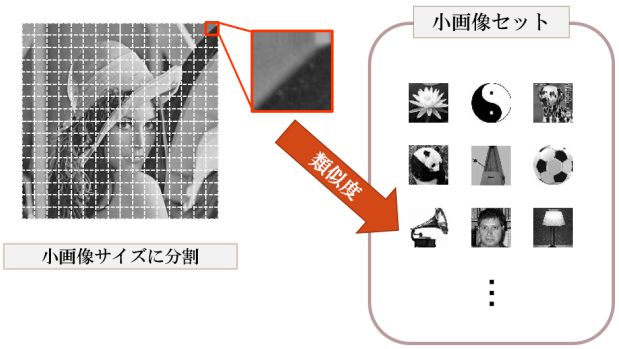

Fig. 2. フォトモザイクの生成手法

このフォトモザイクについては多くの研究があり，論文 [1] では，色情報だけでなく SIFT 特徴量 [2] を用いて，画像の勾配方向も評価しながら小画像を選択している．また，グリッド状の並びでは表現の限界があるため，論文 [3] では，元画像の領域の分割を工夫し，分割された任意の形状に異なる形状の小画像を最小の隙間と重複で割り当てる手法を提案している．

本論文では，入力された画像をタイルと呼ばれる格子状の領域に分割し，それを並び替えて目的画像を近似するフォトモザイク画像の生成手法として，最適と近似の 2 つの方法を提案する．本論文の提案手法で作成したフォトモザイク画像の一例を Fig. 3 に示す．

最適アルゴリズムとして，入力画像と目的画像のピクセル間の輝度値差総和が最小となるタイルの配置を求める．2 画像それぞれが持つタイルを頂点に，タイル間の輝度値誤差を重み付き辺とすると，最適となるタイル配置を求める問題は二部グラフの最小重みマッチングに帰着できる．マッチング問題を解くことにより，目的画像を再現する最適なタイルの並びを得る．しかし，すべてのタイルの組み合わせについて，最適な配置を見つけるための計算量は膨大である．そこで，本研究では近似アルゴリズムとして，分割されたタイルから 2 つを選び，入れ替え前と入れ替え後の結果をそれぞれ目的画像と比較することで，その誤差を求める．その値から入れ替え後の画像が目的画像に近づいた場合はそのまま更新する．この処理を全ペアについて行い，更新がなくなるまで繰り返す．これにより，最適ではないが十分に目的画像を再現できているタイルの並びを現実的な計算量で見つけることができる．しかし，依然として計

算時間がかかるため，本研究では上記提案手法を GPU で実装することにより，フォトモザイク画像の生成処理の高速化を行った．

入力画像　　フォトモザイク画像　　目的画像

Fig. 3. 本研究におけるフォトモザイク

第 2 節ではフォトモザイク画像の生成手法について述べる．第 3 節では高速化に用いた GPU と CUDA アーキテクチャについて簡単に説明し，高速化の方法と実験結果を示す．最後に，第 4 節で本研究のまとめを行う．

II. 提案手法

本節では，一枚の画像を分割し，並べ替えることで作成するフォトモザイク画像の前処理と生成手法について述べる．簡単のために使用する画像を $n \times n$, 分割するタイルのサイズを $m \times m$ の正方形とすると，各画像はそれぞれ $S = \frac{n}{m} \times \frac{n}{m}$ 個のタイルに分割される．入力画像 I と目的画像 T が与えられたとき，本研究の目的は入力画像 I のタイルを並べ替えて目的画像に近似することである．並び替えた後の入力画像を I' とすると，目的画像とのピクセル間の輝度値差の総和 e は

$$e = \sum_{j=0}^{n-1} \sum_{i=0}^{n-1} |I'_{j,i} - T_{j,i}| \tag{1}$$

で求められる．したがって，この e が最小となるタイルの配置を見つけることが本研究の目的となる．

A. ヒストグラム平坦化処理

ヒストグラムとは，画像の輝度値の分布を表わすものであり，明暗の分布やコントラストなど画像全体の性質を読み取ることができる．入力画像と目的画像の全体輝度値が大きく異なっている場合，または輝度値の幅が狭く偏っているとき，生成されるフォトモザイク画像の品質に影響する．そのため，分割並び替えの前処理として，入力画像を目的画像の最大輝度値と最小輝度値の間でヒストグラム平坦化処理を行う．k を平坦化処理前の輝度値，$hst(k)$ を輝度値 k における頻度，k_{min}, k_{max} を目的画像の最小輝度値，最大輝度値とすると，平坦化処理を行った後の輝度値 k' は，式 (2) で求めることができる．

$$k' = k_{min} + (k_{max} - k_{min}) \cdot P(k) \tag{2}$$

$$P(k) = \frac{1}{n \cdot n} \cdot \sum_{i=0}^{k} hst(i) \tag{3}$$

式 (3) の $P(k)$ は平坦化のための点操作関数である．この点操作関数は，変換前輝度値 k に対するヒストグラム $hst(k)$ 累積度数を画像の総画素数で割ったものから求めることができる．Fig. 4 に実際にヒストグラム平坦化処理を行った前後の画像とそのヒストグラムを示す．変換前の画像は明るさにメリハリがなくぼやけた印象だが，ヒストグラム平坦化処理を行った後は明暗が際立ち，建物や背景などの境界線も見えやすくなっている．また，平坦化により，コントラストが高くなっていることが解る．

変換前の画像　　変換前のヒストグラム

変換後の画像　　変換後のヒストグラム

Fig. 4. 平坦化前後の画像とそのヒストグラム

B. 最適アルゴリズム

入力画像と目的画像を比較し，輝度値差の総和が小さいほど目的画像に近づく．したがって，フォトモザイクの生成を重み付き完全二部グラフの最小重み最大マッチング問題として見なすことができる．二部グラフとは，頂点の集合が 2 分割され，辺がその 2 つの部分集合の間を結ぶものに限られるグラフである．そして，完全二部グラフは，二部グラフのうち第 1 の集合に属するそれぞれの頂点から第 2 の集合に属する全ての頂点に辺が伸びているものをいう．また，マッチングとは，グラフ中において，端点を共有しない辺の集合であり，辺数が最大のものを最大マッチングという．Fig. 5 のように，入力画像 I と目的画像 T の各タイル $\{I_1, I_2, I_3, \ldots, I_S\}$ と $\{T_1, T_2, T_3, \ldots, T_S\}$ をそれぞれの二部グラフの集合の頂点とすると，各タイル間の輝度値の差は

$$w_{j,i} = \sum_{y=0}^{m-1} \sum_{x=0}^{m-1} |I_{j(y,x)} - T_{i(y,x)}| \tag{4}$$

で求めることができ，これは各頂点間の辺の重みである．そして，二画像間の輝度値差の総和は

$$e = \sum_{(j,i) \in matching} w_{j,i} \tag{5}$$

となる．この輝度値差の総和，つまりは重みの総和が最小となるようなマッチングを見つけることが目的であり，最適アルゴリズムとなる．本研究では入力画像 I と目的画像 T は同じサイズを使用する，また，分割タイルのサイズと個数は同等であると設定している．よって，グラフ上の全ての頂点がマッチング中のいずれかの辺の端点になることが可能であるため，このグラフにおける最大マッチングは完全マッチングでもある．そして，二部グラフの最小重み最大マッチングから，目的画像の各タイルとマッチングする入力画像のタイルを対応の位置に並べ替えることにより，輝度値差が最小のフォトモザイクを生成することができる．

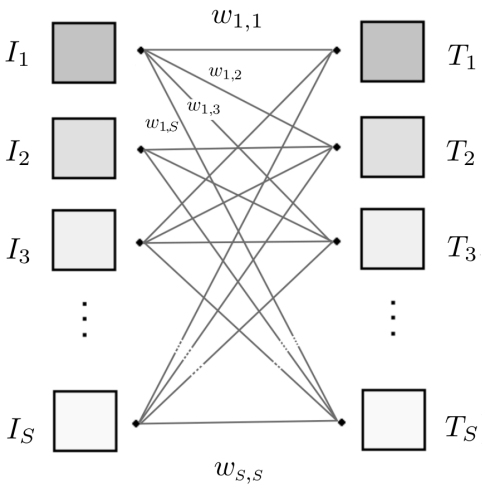

Fig. 5. 完全二部グラフ

C. 近似アルゴリズム

入れ替えるタイルの組み合わせはタイル数の階乗個存在するため，最適アルゴリズム探索の場合，タイルサイズの縮小に伴いタイル数が増加し，計算量が膨大になる．したがって，本研究では，近似アルゴリズムとして，画像中の２つのタイルを目的画像に近づく場合のみ並べ替えることで計算量の削減を行う．以下にその具体的な方法を記す．

Fig. 6 に示すように，入力画像で選択されたタイルを A・B，目的画像で選択されたタイルを X・Y とする．まず，交換前の二画像の対応領域である X と A，Y と B のピクセル間の輝度値差の和を求める．次にタイル A とタイル B の位置を入れ替え，交換後となる X と B，Y と A の輝度値差の和を求める．交換後の差が交換前よりも小さくなれば，交換を行う方が目的画像の輝度値に近づくため，選択タイル A と B の位置を交換を行う．

上記の処理を全ペアについて行う．画像サイズを $n \times n$，分割するタイルのサイズを $m \times m$ の正方形とすると，2 つの画像はそれぞれ $S = \frac{n}{m} \times \frac{n}{m}$ 個のタイルに分割され，各画像のタイルについて ${}_S C_2$ 通りの組み合わせの比較を行うことになる．すべての組み合わせの処理を終えると，最初から同様の処理を繰り返す．全タイルを処理した際，１度も交換が起きなかった場合は，これ以上交換を行っても近似度が良くならないことを意味するため，処理を終了して結果のフォトモザイク画像を出力する．

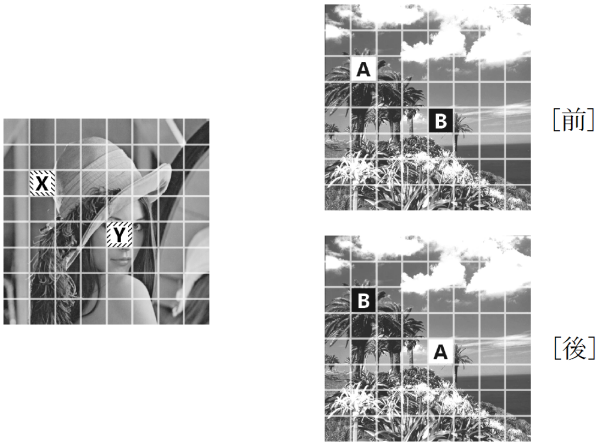

Fig. 6. タイルの比較交換

III. GPU による並列処理

GPU(Graphics Processing Unit) とはグラフィックス処理に特化したハードウェアである．内部に多数の演算コア，広いメモリ帯域幅により高い演算性能を有する．また，NVIDIA 社が提供する CUDA(Compute Unied Device Architecture) は C/C++言語をベースとした統合開発環境である．本章では GPU のアーキテクチャと CUDA について説明し，これらを用いてフォトモザイク画像の生成の高速化を行う．

1) GPU アーキテクチャ: Fig. 7 に GPU のハードウェア構成の概略図を示す．GPU は主にプログラムの処理ユニットとして複数の SM(Streaming Multiprocessor) とデータを格納するメモリから構成され，SM の動作はそれぞれ独立している．各 SM には多数のコアが搭載されており，演算処理はこのコアによって並列に実行される．コアは演算の最小の単位である．

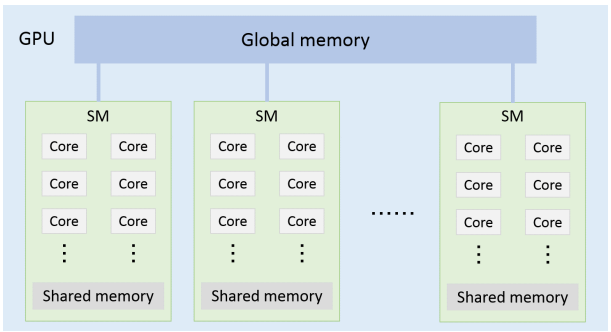

Fig. 7. GPU アーキテクチャの概略図

2) メモリ構造: GPU の内部にはいくつかの種類のメモリを持っており，大きく分けてグローバルメモリ，シェアードメモリ，レジスタがある．高速化するには，それぞれのメモリの特徴を踏まえてプログラミングをすることが重要となる．Fig. 8 はメモリ構造の概略図である．

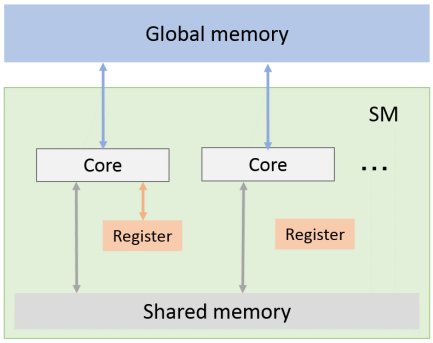

Fig. 8. メモリ構造

3) CUDA: CUDA は NVIDIA 社の GPU 製品において汎用的なプログラムを動かすためのプラットフォームであり，C 言語ベースの拡張言語により，容易に習得出来る．CUDA での並列処理はスレッドと呼ばれる単位で行われており，階層的に管理されている．スレッドは処理の最小単位であり，複数のスレッドをまとめたものをブロックと呼ぶ．そしてブロックの集合はグリッドと呼ばれる．Fig. 9 は階層間の関係を示す．CUDA においてのグリッド，ブロック，スレッドはそれぞれ GPU アーキテクチャにおける GPU，SM，コアに対応している．また，グリッド内のブロック数やブロック内のスレッド数は，使用する GPU の呼び出し可能最大数範囲内で任意に指定することができる．

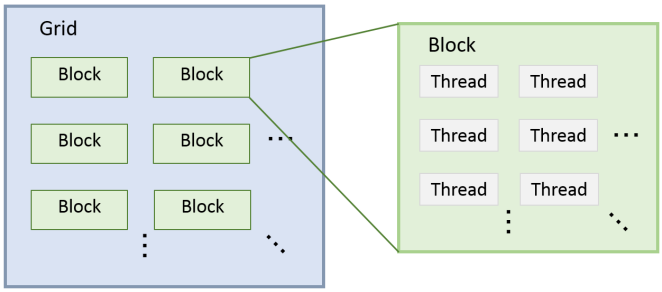

Fig. 9. スレッド，ブロック，グリッドの階層関係

CUDA プログラミングでは CPU 側をホストと呼び，GPU 側をデバイスと呼んで，この両方を用いて処理を実行する．GPU に実行させるプログラムをカーネルと呼び，ホスト側から呼び出す．このとき，CPU 側からカーネル実行に必要なデータを送信するが，データの転送時間も考慮して高速化する必要がある．

カーネルには処理内容が記述されており，ホスト側から呼び出された全スレッドにおいて同じ処理を実行するが，呼び出された関数は並列に動作していても非同期になっており，あるスレッドの処理が終わっても他のスレッドの処理が終了していないことがある．よって，スレッド間の演算結果が互いに依存する場合，同期処理を行わなければならない．しかし，この同期処理は全スレッドの足並みが揃うまで待機するため時間がかかり，多用は望ましくない．

A. 重みの事前計算

比較交換を行う前後のタイル間の輝度値差総和をその都度に計算するのは無駄が多いため，処理の最初に入力画像と目的画像のそれぞれのタイル間すべての組み合わせについて，事前に輝度値総和を計算し，メモリに格納しておく．これにより，比較交換を行う際の計算が加算のみになるため，計算量が大幅に削減される．また，この事前計算についても，GPU を用いて各タイル間の輝度値差を並列に計算することで高速化を行う．

B. 比較の並列化

並べ替えの実行は全てのタイルの組み合わせについて逐次的に行うため，同じタイル場所を含む組み合わせは情報が競合してしまい，並列処理できない．したがって，タイルの位置重複が存在しない組み合わせを選択して並列処理を行う必要がある．位置重複をしない組み合わせの選択については，グラフの辺彩色を用いて行う．

グラフの辺彩色とは，グラフの辺を彩色するものであり，1 つの頂点を共有するそれぞれの辺に対して，常に異なる色を割当てるように最適彩色をする．これにより，同じ色の辺に接合する点が重複することはないため，これをタイル選択に対応することで並列処理が可能な組み合わせを導出できる．

すべてのタイルの組み合わせについて比較交換処理を行うため，この場合，完全グラフとして考えることができる．完全グラフとは任意の 2 頂点に対して，それらを結ぶ丁度 1 本の枝をもつグラフのことを指す．n 頂点の完全グラフは，K_n で表される．例として，8 頂点の完全グラフ K_8 を Fig. 10 に示す．完全グラフの辺彩色数について，頂点数が偶数のときは $n-1$ 色で塗ることができ，頂点数が奇数のときは n 色で塗ることができる．完全グラフの辺彩色方法を Fig. 11 に示す．

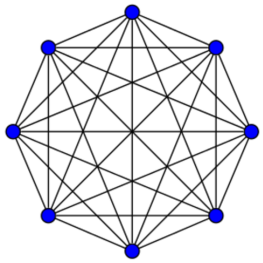

Fig. 10. 完全グラフ K_8

Fig. 11(a) のような奇数頂点 $n=5$ の完全グラフがあるとする．この奇数頂点の完全グラフを正多角形として見なすと，まず周囲 5 辺をそれぞれ異なる 5 色で塗る (Fig. 11(b))．次に，頂点を共有している辺には異なる彩色をする必要があるため，色を塗られた各周囲辺について，それを平行な辺を同じ色で塗っていく (Fig. 11(c))．すべての辺について順に塗り終えると，その結果が奇数頂点の完全グラフの辺彩色となる (Fig. 11(d))．

偶数頂点の完全グラフ $n=6$ の場合は，まず $n-1=5$ の奇数頂点完全グラフを上記の手順で彩色を行う．このとき，得られた彩色グラフの各頂点には 4 辺が接合している

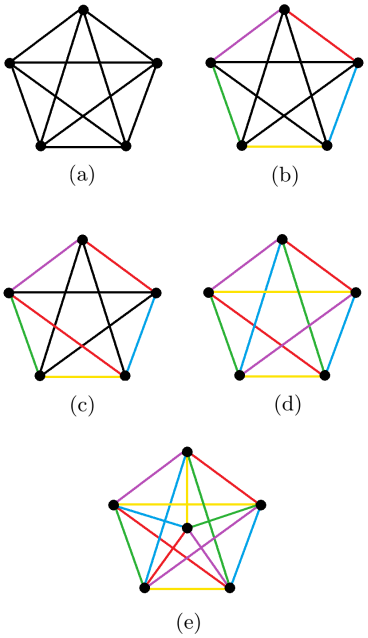

Fig. 11. 彩色方法

のに対し，グラフ全体で使用されている辺彩色数は 5 色となっていることが分かる．したがって，ここに残りの 1 つの頂点を追加し，他の頂点から，その頂点でまだ使われていない色で塗ることで偶数頂点の完全グラフの彩色ができる (Fig. 11(e)).

辺彩色では，同じ色の辺に繋がっている頂点は重複しない．よって，グラフの各頂点を各タイルに見なすと，各辺に接する 2 頂点はタイルの組み合わせに見立てることができ，同じ色の辺を選択することで位置重複のない組み合わせを作成できる．

以上により，並列化を行う際は，同じ色に接合する頂点を組み合わせとして，同時に比較交換処理を行うことができる．このように色ごとにカーネルを起動して並列化を行い，すべての色について処理を終えると，最初から同様の処理を各色について，交換が起きなくなるまで繰り返す．

C. 実験結果

本節では，提案手法で生成したフォトモザイク画像の結果を示し，考察を行う．また，提案手法を CPU と GPU で実装した場合の実行時間の結果を示し，比較を行った．

実験環境

CPU: Intel Core i7-3770
- 動作周波数: 3.50GHz
- メモリ: 8GB

GPU: NVIDIA Tesla K40
- コア数: 2880 基
- 動作周波数: 745MHz
- メモリ: 12GB

サイズ 512 × 512 の目的画像に対して，入力画像をサイズ 16 × 16 のタイルに分割し，提案手法を用いてフォトモザイク画像を生成した結果を Fig. 12, Fig. 13 に示す．

Fig. 12(a) を入力画像，Fig. 12(b) を目的画像としたとき，Fig. 12(c) は最適アルゴリズムより生成したフォトモザイク画像である．目的画像内の形や輝度諧調を再現できており，近い結果が得られている．次に，近似アルゴリズムで生成した Fig. 12(d) と最適アルゴリズムで生成した Fig. 12(c) と比較すると，近似アルゴリズムで得られた結果は最適と比べて十分に目的画像を再現できていることが解る．

また，Fig. 13(a) と Fig. 13(b) は入力画像と目的画像を逆にした場合の最適と近似それぞれの生成結果であり，こちらも画像内の物体，光の加減が表現できている．

最適アルゴリズムと近似アルゴリズムで生成したフォトモザイク画像と目的画像のピクセル間の総誤差値を Table I に示す．また，フォトモザイク画像の生成時間を Table II に示す．最適アルゴリズムのグラフマッチングには Mathematica 10.4.1 [4] を使用する．提案手法を GPU 実装することによって，画像サイズ 512 × 512，タイルサイズ 16 × 16 の入力画像に対して実行時間は約 0.067 秒となり，CPU と比較して約 24.7 倍の高速化を達成した．最適アルゴリズムを用いた場合生成速度と比較しても，大幅に時間短縮できていることが解る．

TABLE I
結果画像と目的画像間の総誤差値

	最適	近似
実験画像 1 (Fig. 12)	6508585	6593629
実験画像 2 (Fig. 13)	5410140	5520554

TABLE II
フォトモザイク画像の生成時間

	CPU[s]	GPU[s]
重み計算	1.567	0.029

	最適	近似	
	CPU[s]	CPU[s]	GPU[s]
タイル入れ替え	14.234	0.090	0.038

IV. まとめ

本研究では，入力された画像を格子状に分割し，並べ替えることによって，目的画像に近似するフォトモザイク画像を生成する手法を提案した．また，GPU に実装することで計算の高速化を行った．結果として，提案したフォトモザイク画像の生成手法は CPU 実装と比較して最大で約 25 倍の高速化を達成した．

今後の課題としては，更なる計算の高速化と生成されるフォトモザイク画像の品質向上，分割する部分領域の形状の工夫などが挙げられる．

REFERENCES

[1] S. Yanase, K. Kaneda and T. Tamaki : ”A study of photomosaic image synthesis considering blur”, 10th HISS, CD-ROM, pp. 303-306, 2008.
[2] D. G. Lowe : ”Object recognition from local scale in variant features”, Proc. of ICCV, pp. 1150-1157, 1999.

(a) 入力画像

(b) 目的画像

(c) フォトモザイク画像 (最適)

(d) フォトモザイク画像 (近似)

Fig. 12. 実験結果 1

(a) フォトモザイク画像 (最適)

(b) フォトモザイク画像 (近似)

Fig. 13. 実験結果 2

[3] J. Kim, F. Pellacini : ”Algorithms for the assignment and transportation problems”, ACM Transactions on Graphics, vol.21 no.3, pp. 657-664, 2002.
[4] WOLFRAM Mathematica 10.4.1 : http://reference.wolfram.com/legacy/language/v10.4/