Vol.2018-AL-169 No.4
2018/9/3

局所探索と機械学習を用いたタイルアート画像生成手法

松村 直樹[1,a)]　戸倉 宏樹[1]　黒田 悠希[1]　伊藤 靖朗[1]　中野 浩嗣[1]

概要：本論文では，正方形のパターンをキャンバスに置くことによって質の高いタイルアート画像を生成する非写実的画像生成手法を紹介する．人間の視覚特性に基づいたタイルアート画像を生成するための局所探索を用いた実装を提案する．しかし，この手法の計算には莫大な時間を必要とするため，Graphics Processing Unit（GPU）を用いた並列手法により高速化を実現した．また，更なる高速化のため，機械学習を用いた近似タイルアート画像生成手法についても提案する．結果として，並列手法の GPU 実装と機械学習を用いた近似手法の GPU 実装は，並列手法の CPU 逐次実装と比較してそれぞれ 160 倍，37 万倍の高速化を達成した．

キーワード：タイルアート，GPU，機械学習

Tile Art Image Generation Methods using Local Search and Machine Learning

Naoki Matsumura[1,a)]　Hiroki Tokura[1]　Yuki Kuroda[1]　Yasuaki Ito[1]　Koji Nakano[1]

Abstract: The main contribution of this paper is to show a non-photorealistic rendering for high quality tile art image generation with squares by pasting square patterns on canvas. Our technique is a method using the local search inspired by the characteristic of the human visual system to optimize generated images. Although it can generate high quality tile art images, a lot of time is necessary. Hence, we have proposed a parallel method using a graphics processing unit (GPU) to accelerate the computation. Also, for further speed-up, we have proposed an approximate tile art image generation method using machine learning. As a result, GPU implementation with the parallel method and GPU implementation with approximate method using machine learning can achieve a speed-up factor of 160 and 370000 over the sequential CPU implementation with parallel method, respectively.

Keywords: Tile art, GPU, Machine learning

1. はじめに

非写実的画像生成とは油絵，水彩画，点描画，タイルアート，モザイクアートなどの芸術表現に似たイメージを生成する画像処理技術である．タイルアートとは，紀元前から現代に続く美術工芸である．タイルアートは，磁器やガラスで作られたタイルを，入力画像を再現するようキャンバスに置くことで作成される．

本研究では，人間の視覚特性に基づいたタイルアート画像生成手法を提案する．入力画像が与えられたとき，色付きの正方形タイルをキャンバスに 1 枚 1 枚置いていくことにより，入力画像を再現するタイルアート画像を生成する．タイルどうしが重ならない既存手法 [1] と異なり，本研究ではタイルの重なりを許容する．提案タイルアート画像を**図 1** に示す．

より質の高いタイルアート画像を生成するため，[2] のように，人間の視覚特性に基づいた評価を行う．人間の目に投影された画像は，オリジナルの画像と比較して少しぼやけた画像となる．この投影された画像は，人間の視覚特性

[1] 広島大学
Hiroshima Uniersity, Kagamiyama 1-4-1, Higashi-Hiroshima, 739-8527,Japan
[a)] matsumura@cs.hiroshima-u.ac.jp

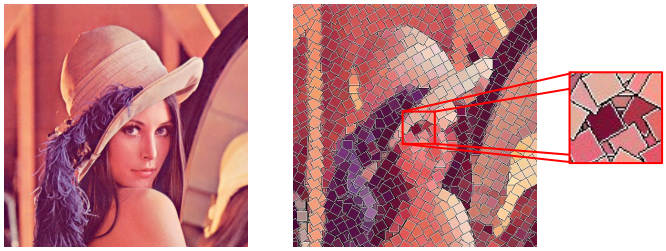


図 1　提案タイルアート画像
Fig. 1　Proposed tile art image

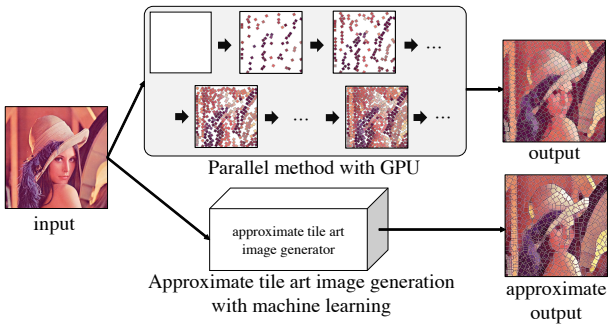


図 2　高速化手法
Fig. 2　Acceleration method

の特徴に類似したガウシアンフィルタを用いて計算する．より入力画像に近似したタイルアート画像を生成するために，入力画像と投影された画像のピクセルの輝度値の差の合計を用いてタイルを置く．この合計を誤差総和と定義する．この誤差総和を小さくするように，1 枚 1 枚タイルを置いていく．以下では，この操作を**局所探索法**と呼ぶことにする．タイルを置いても誤差総和が小さくならなかった場合，操作を終了し，タイルアート画像として出力する．

しかしながら，このタイルアート画像生成手法は計算量が莫大になるため，高速化手法（1）Graphics Processing Unit（GPU）を用いた並列手法，（2）機械学習を用いた近似タイルアート画像生成手法を提案する．

GPU を用いた並列手法では，1 枚 1 枚置いていたタイルを同時に置くことを考える．このアルゴリズムを GPU に実装し，同アルゴリズムの CPU 逐次実装と比較したところ，約 160 倍の高速化を達成した．

また，更なる高速化のため，機械学習を用いた近似タイルアート画像生成手法について開発した．この手法では，Generative Adversarial Network[3] を拡張した Conditional Generative Adversarial Network(cGAN)[4] と Deep Convolutional GAN（DCGAN）[5] のアイディアを使用している．これらのアイディアは，写真の色付け [6]，欠落した画像の修正 [7] などに使われている．学習に用いるデータセットは，局所探索法によって生成したタイルアート画像を用いて作成する．このアイディアを GPU に実装し，並列手法の CPU 逐次実装と比較したところ，約 37 万倍の高速化を達成した．

上記で説明した二つの高速化手法の概要を**図 2** に示す．図 2 の上側が GPU を用いた並列手法によるタイルアート画像生成を示しており，入力画像に近づけるようキャンバスにタイルを置いていくことでタイルアート画像を生成する．図 2 の下側が機械学習による近似タイルアート画像生成を示しており，入力画像をタイルアート画像生成器に入力することで近似タイルアート画像を生成する．

2. 人間の視覚特性に基づいたタイルアート画像生成

本節では，人間の視覚特性に基づいたタイルアート画像生成について説明する．最初に，誤差総和の計算方法を説明する．次に，1 枚 1 枚タイルを置く手法である局所探索法を説明する．これらは説明上，グレースケール画像について考え，その後カラー画像に拡張する．最後に，同時にタイルを置く並列手法と GPU 実装について説明する．

2.1 誤差総和の計算

画像サイズ $N \times N$ の正方形の入力画像 $A = (a_{i,j})$ を考える．ここで，$a_{i,j}$ は座標 $(i,j)(1 \leq i,j \leq N)$ における範囲 $[0,1]$ の実数である．本節で説明するタイルアート画像生成手法は，A を再現するようタイルを複数枚置いて生成された出力画像 $B = (b_{i,j})$ を得ることを目的とする．出力画像 B の質の良さは，人間の視覚特性の特徴と類似したガウシアンフィルタを用いて計算できる．ガウシアンフィルタを $G = g_{p,q}$ とする．ガウシアンフィルア G の大きさは $(2w+1) \times (2w+1)$ であり，各要素 $g_{p,q}(-w \leq p,q \leq w)$ の値は，全要素の総和が 1 になる二次元正規分布の非負の実数値である．つまり各要素は，$\sum_{-w \leq p,q \leq w} g_{p,g} = 1$ を満たす $g_{p,g} = s \cdot e^{-\frac{p^2+q^2}{2\sigma^2}}$ となる．ここで，σ は正規分布のパラメータで，s は総和を 1 にするような実数値である．

出力画像 B にガウシアンフィルタを適用した画像を投影画像 $R = r_{i,j}$ とする．R は B と G の畳み込み，$r_{i,j} = \sum_{-w \leq p,q \leq w} g_{p,q} b_{i+p,j+q}$ $(1 \leq i,j \leq N)$ から求められる．出力画像 B の評価に使用する誤差総和は，入力画像 A と投影画像 R から計算される．各座標 (i,j) における誤差は

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

で表され，誤差総和は

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

と定義される．ガウシアンフィルタのぼかしは人間の視覚特性の特徴と似ているため，$\mathrm{Error}(A,B)$ が十分に小さい場合，出力画像 B を人間の目で見たときに入力画像 A を

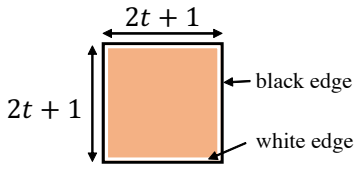


図 3　枠付きのタイル
Fig. 3　Edged tile

再現できていると言える．

2.2　局所探索法

本小節では，誤差総和からタイルを置く過程を説明する．本研究で使用するタイルは $(2t+1)\times(2t+1)$ の固定の大きさである．また，よりタイルアートらしさを再現するために，**図 3** のようにタイルの枠の色を固定する（枠の外側：黒，枠の内側：白）．使用するタイルの集合を P，P の各要素を $p_{u,v}(1 \leq u \leq N_L, 1 \leq v \leq N_R)$ と定義する．ここで，N_L はタイルの色の数であり，N_R はタイルの回転角数を表している．どこにどのタイルパターン p を置けばよいかを計算するため，以下の式で表される改善値 I を定義する．

$$I(A,B,p,i,j) = \mathrm{Error}(A,B) - \mathrm{Erorr}(A,B') \qquad (3)$$

B' は出力画像の座標 (i,j) に対して，タイルパターン p を暫定的に置いた画像である．出力画像 B に対して一枚ずつ $N_L \times N_R$ 種類のタイルを置き，各座標 $(i,j)(1 \leq i,j \leq N)$ の改善値 I を計算する．これらの改善値から，座標 (i,j) における最大の改善値 $q_{i,j} = \arg\max_{p\in P} I(A,B,p,i,j)$ を得る．全ての $q_{i,j}$ の内，一番大きいものを q_{best} とする．出力画像 B の q_{best} となる座標にタイルパターン p を置くことで，出力画像 B を入力画像 A に近似していく．上記の操作を繰り返し，どの座標にどのタイルパターンを置いても改善されなくなった場合，操作を終了する．

また，タイルを画像の境界に置く場合，ガウシアンフィルタを適用するため画像の境界外のピクセルが必要になる．この問題は，画像の境界の輝度値を拡張することで解決する [9]．

本手法では，白か黒の色を背景色として持つキャンバスを出力画像の開始画像とする [8]．そのため，背景をタイルで完全に埋めることができなかった場合，背景色がノイズのように映ってしまう．これを防ぐため，始めに背景を完全に埋めるようにタイルを置いていく．出力画像 B の座標 (i,j) にタイルパターン p を置いたときの，背景画素を覆う割合を以下に示す．

$$C(B,p,i,j) = \frac{\text{the number of covered background pixels}}{\text{the number of pixels of } p} \qquad (4)$$

タイルが背景ピクセルを全て隠す場合，$C(B,p,i,j)=1$ となる．一方，タイルが背景ピクセルを 1 ピクセルも隠さない場合，$C(B,p,i,j)=0$ となる．つまり，背景をできるだけ隠すためには，$C(B,p,i,j)>0$ となるようなパターンを選択する必要がある．式 (3) を拡張し，割合 C を考慮した改善値を $I_{\mathrm{cover}}(A,B,p,i,j) = (C(B,p,i,j), I(A,B,p,i,j))$ と定義する．I_{cover} の大小関係は，割合 C の大小関係に左右される．二つの異なるタイルパターン p，p' に対する改善値 $I_{\mathrm{cover}}(A,B,p,i,j)$，$I_{\mathrm{cover}}(A,B,p',i,j)$ の大小関係が，$I_{\mathrm{cover}}(A,B,p,i,j) > I_{\mathrm{cover}}(A,B,p',i,j)$ となる条件は以下のようになる．

- $C(B,p,i,j) > C(B,p',i,j)$ or,
- $C(B,p,i,j) = C(B,p',i,j)$ and $I(A,B,p,i,j) > I(A,B,p',i,j)$.

背景が全てタイルで隠れるまで，改善値は I_{cover} を使用する．また，ガウシアンフィルタの適用ピクセル内に背景画素が存在すると，誤差総和に影響してしまう．これを避けるため，I_{cover} の誤差計算は式 (2) ではなく $e_{i,j} = a_{i,j} - b_{i,j}$ とする．背景が全てタイルで隠れた後，改善値を I に変更し，操作を再開する．

2.3　カラー画像への拡張

ここから，グレースケール画像に関する誤差総和の計算をカラー画像に拡張する．本研究では，RGB について考え，各チャネルの値の範囲がそれぞれ $[0,1]$ であるものとする．カラー画像では，投影画像 R と式 (1) の誤差は各チャネルにおいて別々に計算される．つまり，各チャネルに対してガウシアンフィルタを適用し，誤差を計算する．座標 (i,j) における，赤，緑，青の誤差を $e^R_{i,j}$，$e^G_{i,j}$，$e^B_{i,j}$ とする．式 (2) は $\mathrm{Error}(A,B) = \sum_{1\leq i,j\leq N}(|e^R_{i,j}| + |e^G_{i,j}| + |e^B_{i,j}|)$ に拡張される．グレースケール画像とカラー画像の違いは，上記の誤差と誤差総和の計算のみである．その他の計算部分に関しては，両方とも共通の操作である．

2.4　並列手法

本研究では，ガウシアンフィルタを適用した画像から誤差を計算するため，タイルを置くことにより周囲の誤差が変動する．$W(i,j)$ を座標 (i,j) を中心とした $(2t+1)\times(2t+1)$ の枠とする．この枠は，タイルの回転を考慮したタイルの大きさである．$(2w+1)\times(2w+1)$ のガウシアンフィルタを用いるため，タイルを置くことにより $(2t+2w+1)\times(2t+2w+1)$ の領域の誤差に影響が及ぶ．この領域を影響領域と定義する．**図 4** は影響領域を示している．

ここから，同時にタイルを置く手法を説明する．まず，$N\times N$ の入力画像 A を $h\times h$ の部分画像に分割する．分割した部分画像をさらに以下の四つのグループに分ける：グループ 1（奇数列，奇数行），グループ 2（偶数列，奇数行），グループ 3（奇数列，偶数行），グループ 4（偶数列，偶数行）．**図 5** は部分画像における 4 つのグループを表している．$h \geq 2t+2w+1$ のとき，2 つの部分画像の各グループにおけるガウシアンフィルタは互いに影響を及ぼさない．そのため，$\frac{N}{h}\times\frac{N}{h}$ 枚の部分画像に対して同時にタイルを置くことができる．つまり，1 つのグループに対す

る並列実行で $\frac{N}{h} \times \frac{N}{h}$ 枚タイルを置くことができる．これをグループ 1 からグループ 4 まで順に行う．

2.5 GPU を用いた高速化

本小節では，GPU 実装を用いた並列手法の高速化について述べる．並列手法は，二つのカーネルから実行される．カーネル 1 は，各グループ内の最大の改善値 q_{best} を求めるカーネルである．各ブロックを各部分画像に割り当てた後，ブロック内の各スレッドが各座標の改善値を求める．求めた結果をグローバルメモリに書き込み，カーネル 1 の処理を終了する．カーネル 2 は，タイルを置くカーネルである．グローバルメモリに書き込まれた情報から，部分画像内の各グループに対してタイルを置いていく．カーネル 1，カーネル 2 の操作をグループ 1 からグループ 4 に対して行い，どの部分画像に対しても改善しなくなるまで繰り返し実行する．また，更なる高速化のため，カーネル 1 では以下の 3 つの効率的なアイディアを用いる．

シェアードメモリを用いたデータキャッシング：入力画像 A と出力画像 B はグローバルメモリに格納されているが，これらは計算上何度も使用される．そこで，データへのアクセス時間を小さくするために，シェアードメモリをキャッシュとして利用する．

加算を用いたガウシアンフィルタの適用：このアイディアは，「投影画像 R を求める畳み込み演算」を「ガウシアンフィルタを適用したタイルパターンの加算」に変換することである．まず，各タイルパターンにガウシアンフィルタを適用した画像（画像 1）をグローバルメモリに書き込む．その後，同じ角度のパターンごとに以下の動作を行う．値が 0 のタイル，すなわち黒色のタイルを置いた部分画像にガウシアンフィルタを適用した画像（画像 2）をシェアードメモリに保存する．あらゆるタイルパターンを置いてガウシアンフィルタを適用する代わりに，画像 1 を画像 2 に加算する．こうすることで，同じ角度のタイルパターンが行う計算は，最初の黒のパターンを除いて，畳み込みから加算に置きかえられる．

ワープシャッフル命令による総和計算の並列化：誤差総和を求めるために，総和計算をする必要がある．本実装では，ワープシャッフル命令を用いた総和計算の並列化テクニックを使用する [10]．ワープシャッフル命令は 32 スレッドの集合であるワープ内の各スレッドにおける，シェアードメモリを介さずにデータを送受信する命令である [11]．このテクニックを用いて，総和計算の並列度を高める．

3. 機械学習を用いたタイルアート生成

本節では，機械学習を用いた近似タイルアート画像生成手法について説明する．この手法の基本的なアイディアは，上記手法のようにタイルを貼り付けて画像を生成するのではなく，入力画像を学習済みの近似タイルアート画像生成器に入力することで近似タイルアート画像を生成する．以下では，この手法を**機械学習法**と呼ぶことにする．

3.1 目的関数

Generative Adversarial Networks(GAN)[3] は，潜在変数 $\boldsymbol{z}$ から訓練データ $\boldsymbol{x}$ への写像を学習する生成モデルである．GAN は次の二つのモデルから構成される：データの分布を学習する生成モデルの Generator(G)，入力画像がトレーニングデータか G が生成したデータかの確率を予測する鑑別モデルの Discriminator(D)．これらのモデルが敵対的に学習を行うことで，互いに性能を高めていく．G が潜在変数 $\boldsymbol{z}$ に対して生成したデータを $G(\boldsymbol{z})$ と定義する．また，D が訓練データ $\boldsymbol{x}$ を鑑別したときに出力する値を $D(\boldsymbol{x})$，D が G によって生成されたデータ $G(\boldsymbol{z})$ を鑑別したときに出力する値を $D(G(\boldsymbol{z}))$ とする．$D(\boldsymbol{x})$ と $D(G(\boldsymbol{z}))$ の値は $[0,1]$ の実数値で，値が 1 に近いほど鑑別結果が訓練データ $\boldsymbol{x}$ であることを示している．GAN の目的関数は $L_{GAN} = \mathbb{E}[logD(\boldsymbol{x})] + \mathbb{E}[log(1 - D(G(\boldsymbol{z})))]$ と表される．

しかしながら，GAN は潜在変数 $\boldsymbol{z}$ からデータを生成するために，出力データの制御ができないという問題が生じる．そこで本研究では，GAN を拡張したモデルである Conditional Generative Adversarial Network(cGAN)[4] を使用する．cGAN は，GAN で定義した G と D のそれぞれの入力にラベル $\boldsymbol{y}$ を追加したモデルである．ラベルを追加することで，G はラベル $\boldsymbol{y}$ を考慮した写像を学習するようになる．潜在変数 $\boldsymbol{z}$ とラベル $\boldsymbol{y}$ に対して G が生成したデータを $G(\boldsymbol{z}, \boldsymbol{y})$ とする．cGAN のネットワークを図 6 に示す．図中の real pair はラベル $\boldsymbol{y}$ とそれに対応する訓練データ $\boldsymbol{x}$ の組み合わせ，fake pair はラベル $\boldsymbol{y}$ と G によって生成されたデータ $G(\boldsymbol{z}, \boldsymbol{y})$ の組み合わせを表している．D が real pair を鑑別したときに出力する値を $D(\boldsymbol{x}, \boldsymbol{y})$，fake pair をを鑑別したときに出力する値を $D(G(\boldsymbol{z}, \boldsymbol{y}), \boldsymbol{y})$ とする．これらの値は $[0,1]$ の実数値で，値が 1 に近いほど鑑別結果が訓練データ $\boldsymbol{x}$ であることを示している．cGAN の目的関数 $L_{cGAN}(G, D)$ は以下のようになる．

$$L_{cGAN}(G, D) = \mathbb{E}[logD(\boldsymbol{x}, \boldsymbol{y})] + \mathbb{E}[log(1 - D(G(\boldsymbol{z}, \boldsymbol{y}), \boldsymbol{y}))] \tag{5}$$

また，G が生成したデータ $G(\boldsymbol{z}, \boldsymbol{y})$ をより訓練データ $\boldsymbol{x}$ へ近づけるため，目的関数に別のコスト関数を追加する．このコスト関数には，一般的に平均二乗誤差が用いられる．しかし，平均二乗誤差を用いて学習を行った場合，生成画像がぼやけてしまう [6], [7]．そこで本研究では，L1 ノルムを用いたコスト関数を設定する．L1 ノルムを用いたコスト関数 $L_{L1}(G)$ を以下に示す．

$$L_{L1}(G) = \mathbb{E}[||\boldsymbol{x} - G(\boldsymbol{z}, \boldsymbol{y})||_1] \tag{6}$$

よって，最終的に使用する目的関数は，$L(G, D) = L_{cGAN}(G, D) + \lambda L_{L1}(G)$ となる．ここで，λ は L_{L1} による影響をどれだけ大きくするかというハイパーパラメータで

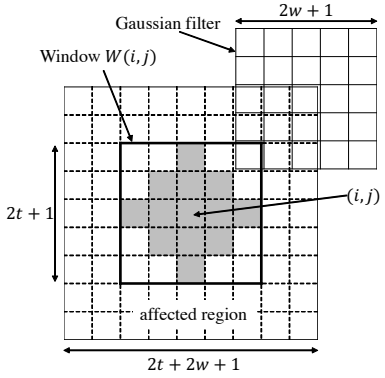


図 4 影響領域
Fig. 4 Affected region

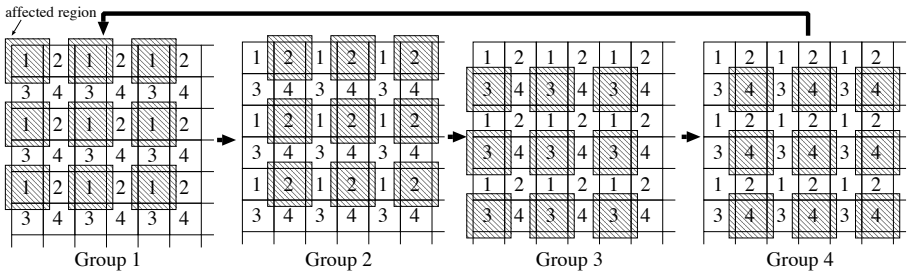


図 5 部分画像における Group1 から Group4 と競合しない並列実行
Fig. 5 Groups of subimages and parallel exectuion without race condition

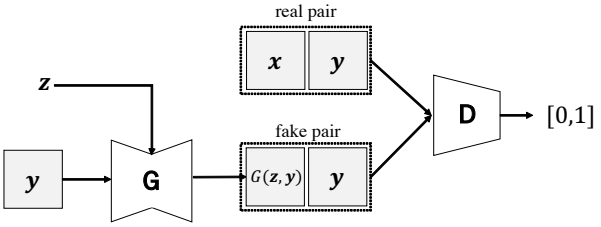


図 6 cGAN のネットワーク図
Fig. 6 Network of cGAN

ある．λ の値が大きければ大きいほど訓練データ $\boldsymbol{x}$ への，小さければ小さいほど D の出力を大きくするデータへの写像を学習するようになる．G は $L(G,D)$ を小さくするように，D は $L(G,D)$ 大きくするよう学習を行う．すなわち，本研究では，$G^* = \min_G(\max_D(L_{cGAN}(G,D)+\lambda L_{L1}(G)))$ で表される最適化問題を解くことを目標とする．G^* は，$L(G,D)$ の G に関する最適値である．

留意点として，潜在変数 z について，単純にガウシアンノイズとして G の入力に与えてしまうと，ラベル y の影響から G が潜在変数 z を無視するように学習をしてしまう [12]．そこで，単純な入力とするのではなく，ドロップアウトという形式で潜在変数 z を与える [13]．このドロップアウトは，学習時と推論時の両方に適用する．

3.2 モデルの構造

本研究で使用する生成モデル G と鑑別モデル D の構造について説明する．二つのモデルの両方で，DCGAN[5] を使用する．

3.2.1 生成器 G

画像変換問題の解法として一般的に使用されているモデルに，Autoencoder が挙げられる．Autoencoder とは，データを圧縮する Encoder と圧縮したデータを復元する Decoder から構成されるモデルである．しかし，このモデルではすべての情報がモデルの最下層を通らなければならない．つまり，低次元の情報は次元削減により失われてしまう．そこで，この問題を回避するために，Autoencoder に Skip Connection[14] を付加した U-Net[15] を導入する．

文献 [15] では，2×2 の最大値プーリングを用いて画像サイズを縮小している．しかし，DCGAN では，ストライド：2 の畳み込みを用いて画像サイズを縮小した方が効果的であるため [5]，そちらを用いてデータ圧縮を行う．

Skip Connection とは，Encoder の i 層目の出力を Decoder の $n-i$ 層目の入力に付加するものである (ここで，n は層の数を表す)．Skip Connection を付加することで，低次元の情報が失われるのを防ぐことができる．

G の具体的な構造については，[5], [13] を参照する．文献 [13] ではあらゆるタスクに対して性能を発揮できるよう深い構造をしている．しかし，本研究ではタイルアート画像の生成を目的としており，近似タイルアート画像生成を完了できるだけのより浅い構造を設計する．そうすることで，畳み込みの演算回数が減り，更なる高速化を見込むことができる．この浅い構造は，実験的に求めた結果から Encoder5 層と Decoder5 層とする．G の構造を**図 7** に示す．

3.2.2 鑑別器 D

本研究では，入力画像をパッチに分け，そのパッチをそれぞれ鑑別するモデルである，Markovian Discriminator[13], [16] の考え方を導入する．本研究では，再現したい特徴はタイルアートのタイル感であるため，画像全体に対して鑑別を行うより画像をパッチに分けて鑑別した方が効果的である．L_{cGAN} によりタイル感を生成するよう Generator にペナルティを課す．

D の具体的な構造については，[5], [13] を参照した．D の構造を**図 8** に示す．図中の破線の矢印はチャネル方向での結合を表している．

3.3 ネットワークの最適化

本小節では，ネットワークの訓練方法を記述する．基本的な訓練方針は [3] を参照した．G と D のパラメータは，確率的勾配降下法を用いて更新する．G のパラメータ更新に用いる誤差は，式 (5) の第 2 項と式 (6) である．λ の値

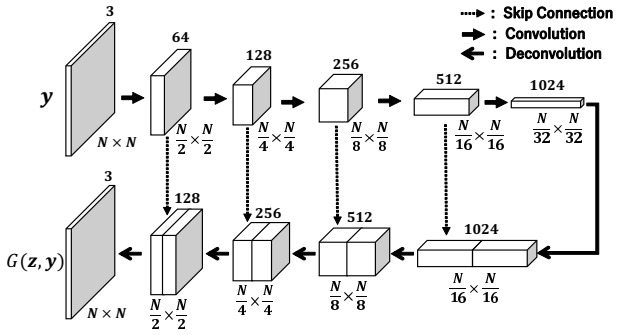


図 7　Generator の構造
Fig. 7　Architecture of generator

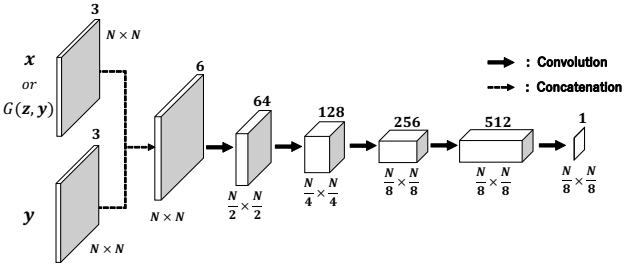


図 8　Discriminator の構造
Fig. 8　Architecture of discriminator

が大きいほど L1 ノルムの制約を課すこととなり，より訓練データの色や形を復元するよう訓練される．逆に，λ の値が小さいほど cGAN による制約を課すことになり，訓練データの特徴であるタイルアート感を復元するよう訓練される．D のパラメータ更新に用いる誤差は，式 (5) である．G と D のパラメータ更新は，両方とも訓練 1 回に対して 1 回更新を行う．

4. 実験

本節では，局所探索法によるタイルアート画像生成と機械学習法による近似タイルアート画像生成について評価する．まず，実験環境について述べ，タイルアート画像の質の評価，タイルアート画像の生成時間の評価を行う．

4.1 実験環境

本研究では, CPU は Intel Xeon E7-8870V4（80 コア）を使用し, 局所探索法の 1 スレッドの実行時間と OpenMP3.1[19] による 160 スレッドの実行時間を計測する．GPU は NVIDIA TITAN V を使用する．CUDA のバージョンは 9.1，nvcc のバージョンは 9.1.85 である．コンパイルオプションは-O2 と-arch_sm=70 を使用している．機械学習法は，python3.5.2 と Tensorflow1.8.0[20] を用いて実装した．ディープラーニング用の CUDA ライブラリである cuDNN のバージョンは 7.1.4 である．

使用するタイルは，4096 色の角度 0 度，30 度，60 度の計 12288 パターンである．タイルサイズは 23×23 で固定する．ガウシアンフィルタのパラメータは $\sigma = 1.3$，$w = 3$ に設定する．部分画像の大きさは 23×23 である．

ラベル y には，[22] のデータセットの内，ランダムに選んだ 400 枚の画像を使用する．各ラベル y に対して，局所探索法を用いてタイルアート画像を生成し，訓練に使用するデータセットとする．G と D の各層では，畳み込み，バッチ正規化 [17]，ReLU 関数または LeakyReLU 関数による活性化を実行している．畳み込みのフィルタサイズは全て 4×4 であり，ストライドは 2 である．ReLU 関数は G の Encoder 層で使用している．LeakyReLU 関数は G の Decoder 層，D で使用しており，傾きは 0.2 である．ドロップアウトの割合は 50% に設定している．目的関数内の λ は 100 に設定し，学習には Adam[18] を用いる．ここで，$\alpha = 0.0001$，$\beta_1 = 0.5$，$\beta_2 = 0.99$，$\epsilon = 10^{-12}$ である．重みは平均 0，標準偏差 0.02 の正規分布を用いて初期化している．訓練回数は 200 回，batch サイズは 4 である．

4.2 生成したタイルアート画像の評価

局所探索法と機械学習法を用いて，2 つの画像 Lena[21](512×512) と風景写真 (1920×1080) に対するタイルアート画像を生成した．生成結果を**図 9**, **図 10** に示す.

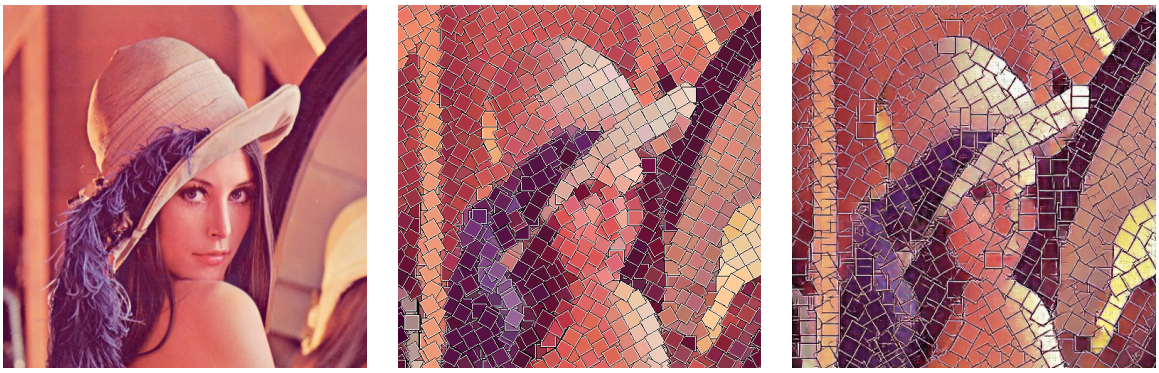
局所探索法によって生成されたタイルアート画像は，誤差総和を用いてタイルを置いているため，入力画像を忠実に再現できていることが分かる．機械学習法によって生成された近似タイルアート画像は，タイルが欠けている箇所が複数見られる．しかしながら，一目で見た場合，局所探索法によって生成されたタイルアート画像と比べて遜色ない近似タイルアート画像が生成できていることがわかる．

4.3 タイルアート生成時間

局所探索法と機械学習法を用いたタイルアート画像の生成時間を比較する．タイルアート画像の生成時間を**表 1** に示す. 局所探索法の並列手法の GPU 実装では，同手法の CPU 実装 (1 スレッド) と比較して最大で約 167 倍，CPU 実装 (160 スレッド) と比較して最大で約 4.8 倍の高速化を達成した．機械学習法の GPU 実装では，局所探索法の並列手法の CPU 実装 (1 スレッド) と比較して最大で 37 万倍，CPU 実装 (160 スレッド) と比較して最大で約 1 万倍，GPU 実装と比較して最大で約 2231 倍の高速化を達成した.

5. Conclusion

本研究では，非写実的画像生成技術の一つであるタイルアート画像生成について，人間の視覚特性に基づいた手法を提案した．また，高速化手法として，GPU を用いた並列手法と機械学習を用いた近似タイルアート画像生成手法を提案した．タイルアート画像の生成時間について，GPU を用いた並列手法では，同アルゴリズムの CPU 逐次実装と比較して約 160 倍の高速化を達成した．機械学習を用いた手法では，並列手法の CPU 逐次実装と比較して約 37 万倍以上の高速化を達成した.



入力画像 (512 × 512)　局所探索法　機械学習法

図 9　生成タイルアート画像 (512 × 512)
Fig. 9　Generated tile art image (512 × 512)

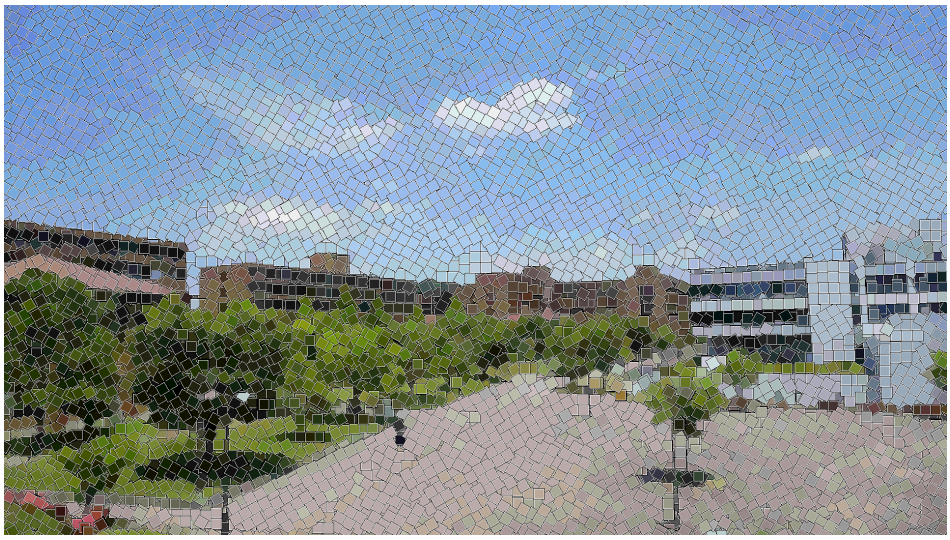
表 1　タイルアート画像生成時間
Table 1　Tile art image generation time

	局所探索法			機械学習法
画像サイズ	CPU 実装 (1 スレッド)[s]	CPU 実装 (160 スレッド)[s]	GPU 実装 [s]	GPU 実装 [s]
256 × 256	1385.450	45.937	10.824	0.014
512 × 512	2865.268	103.903	21.602	0.021
1024 × 1024	13833.348	449.219	98.503	0.054
1920 × 1080	21305.854	607.742	130.822	0.116
2048 × 1536	37655.506	1040.065	253.318	0.158
4096 × 3072	220476.812	6322.173	1322.871	0.593

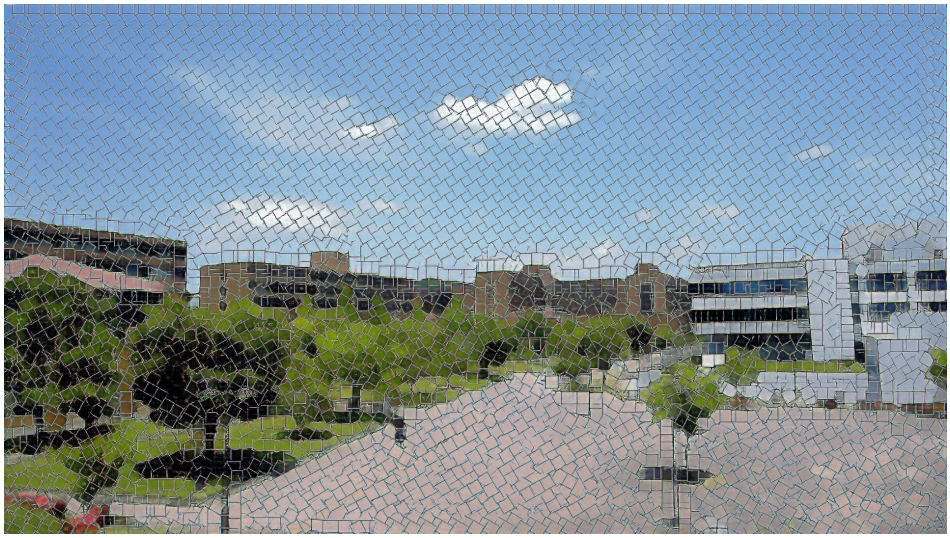
参考文献

[1] T. Houit and F. Nielsen. Video stippling. *Advanced Concepts for Intelligent Vision Systems: 13th International Conference*, 2011.
[2] 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*, 2017.
[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014.
[4] M. Mirza and S. Osindero. Conditional generative adversarial nets.*arXiv*:1411.1784, 2014.
[5] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434, 2015.
[6] R. Zhang, P. Isola and A. A. Efros. Colorful image colorization. *ECCV*, 2016.
[7] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell and A. A. Efros. Context encoders: Feature learning by inpainting.*CVPR*, 2016.
[8] D. Chi. A natural image pointillism with controlled ellipse dots. *Advances in Multimedia*, 2014.
[9] R. C. Gonzalez and R. E. Woods. Digital Image Processing, 3rd ed. Prentice-Hall, Inc., 2006.
[10] J. Luitjens. (2014) Faster parallel reductions on kepler. [Online]. Available: `https://devblogs.nvidia.com/faster-parallel-reductions-kepler/`.
[11] CUDA C Programming Guide Version 9.2.148. NVIDIA Corporation, 2018.
[12] M. Mathieu, C. Couprie and Y. LeCun. Deep multi-scale video prediction beyond mean square error. *ICLR*, 2016.
[13] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.
[14] K. He, X. Zhang, S. Ren and J. Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp.770-778, 2016.
[15] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional networks for biomedical image segmentation. In *MICCAI*, pp.234–241, Springer, 2015.
[16] C. Li and M. Wand. Precomputed Real-Time Texture Synthesis with Markovian Generative Adversarial Networks, *ECCV*, 2016
[17] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv*:1502.03167, 2015.
[18] D. Kingma and J. Ba. Adam:A method for stochastic optimization. *ICLR*, 2015.
[19] OpenMP, `http://www.openmp.org/`.
[20] Tensorflow, `https://www.tensorflow.org/`.
[21] The USC-SIPI image database. `http://sipi.usc.edu/database/`.
[22] Caltech-256 Object Category Dataset, `http://www.vision.caltech.edu/Image_Datasets/Caltech256/`.

入力画像



局所探索法



機械学習法

図 10 生成タイルアート画像 (1920 × 1080)
Fig. 10 Generated tile art image (1920 × 1080)