JAIST Repository
https://dspace.jaist.ac.jp/

Title	2値画像を変形するアルゴリズムに関する研究
Author(s)	芥野，隆文
Citation	
Issue Date	2010-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/8939
Rights	
Description	Supervisor:浅野哲夫，情報科学研究科，修士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

修 士 論 文

2値画像を変形するアルゴリズムに関する研究

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

芥野 隆文

2010 年 3 月

修 士 論 文

2値画像を変形するアルゴリズムに関する研究

指導教官　浅野哲夫 教授

審査委員主査　浅野哲夫 教授
審査委員　上原隆平 准教授
審査委員　平石邦彦 教授

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

810023 芥野 隆文

提出年月 2010 年 2 月

Copyright © 2010 by Keshino Takafumi

概 要

本研究では入出力がともに2値画像の画像変形アルゴリズムの開発を目的とする．本論文ではモーフィングのアルゴリズムを提案し，計算時間が画素数の定数倍になることを示す．また，作業領域に制限がある場合に実現できるかどうかについても述べる．

目 次

第1章　はじめに

1.1　研究の背景

ディジタル画像処理はコンピュータの重要な機能の 1 つで，計算機科学の主要な研究分野であり，近年ではほとんどの画像処理がディジタル化されている．ディジタル画像はラスター形式とベクトル形式の 2 つに大別され用途はそれぞれ異なるが，コンピュータの出力装置ではラスター画像が主に使用される．ラスター画像は画素 (ピクセル) の列で画像を表現したもので，各画素が輝度情報を持っている．輝度情報量を増やすことによって非常に細かな階調を持つことができ，解像度を高くすれば高精細な画像も表現可能であることから写真やディスプレイ装置に使われている．また，2 値化された画像は画像情報が簡単になるので図形処理や特徴解析に用いられ [1]，色数に限りのあるプリンタなどの出力装置にも利用される．

現在，使用されている画像データのサイズは機器の性能に比例して大きくなっているが，スキャナなどの読み込み精度に比べ，その中にある組み込みシステムのメモリ容量は十分でないといえる．画像データが大きいために，画像処理のための作業領域はかなり制限されてしまう．そこで，入力装置への入力が黒インクでの印刷物のような 2 値画像の場合，2 値のまま様々な処理が可能になることはメモリ効率の点で重要である．

1.2　研究の目的

本研究は，2 値画像の変形処理アルゴリズムの開発が目的である．多値画像に変換することなく 2 値画像の変形処理が可能になることは，スキャナなど入力が 2 値画像であるような装置や，2 値画像で出力を行うプリンタなどの装置において計算負荷を軽減するという意味で重要である．また，コンピュータゲームなどのリアルタイムグラフィックスにおいても，負荷を減らすために多値画像データのの一部を 2 値画像で代用することが可能になるかもしれない．

本研究で取り上げるのは 2 値画像のモーフィングの問題である．0 と 1 だけで表現された 2 枚の 2 値画像を入力して一方を元画像，他方を目標画像として，元画像から目標画像へのスムーズな変化に見えるような複数個の中間画像を作成する問題である．モーフィングは 1980 年代に Doug Smythe ら [2] によって映画の特殊効果として開発され，現在はその映画をはじめとして，コンピュータゲームや医療画像など映像の分野で代表的な視覚効果の 1 つとなっている．多値画像においては様々なアルゴリズム [3] があり，全自動で

行われるものや，特徴点の入力によって画像の遷移をコントロールするものもある．しかし，それらのアルゴリズムは 2 値画像には適さない．例えば，代表的なモーフィング技法であるメッシュワーピング (mesh warping) は，各画素値の線形補間を行うので，中間画像を 2 値で作成するのは困難である．

1.3 本論文の流れ

本稿では，第 2 章で 2 値画像のモーフィングについて述べる．その中で，提案アルゴリズムの説明，アルゴリズムの改良の説明，実行結果の評価という流れとなる．

第2章　2値画像のモーフィング

モーフィングとは，ディジタル画像が別のディジタル画像へとスムーズに変化するような映像を作ることである．これを実現するには一般に計算機を用いて，元画像と目標画像の情報を元に一定数の中間画像を作成する．モーフィング技術は主に映像の分野で視覚効果として用いられ，盛んに研究されている．[3] 様々な方法が提案されているが，既存のアルゴリズムは多値画像向けのものであるから，中間階調のない 2 値画像にはそれらの方法は適さない．ここでは 2 値画像のためのアルゴリズムを提案する．

2.1　アルゴリズム

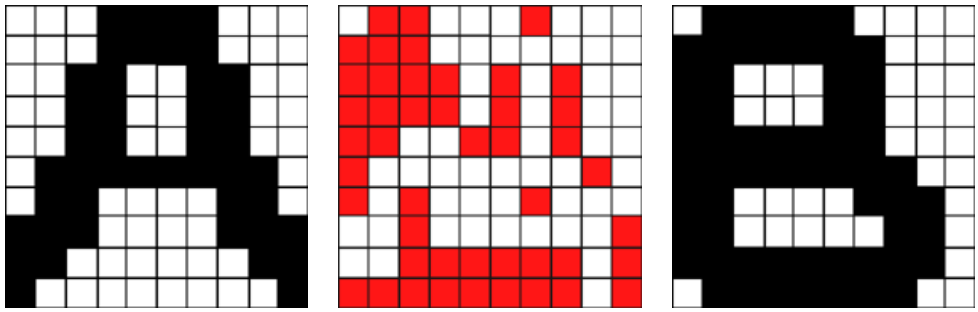
本稿では画素の値を 0 から 1，または 1 から 0 にすることを画素を**反転**するという．提案アルゴリズムでは元画像 I_S と目標画像 I_T の中間画像を作成する．中間画像も 2 値画像である．

入力は各要素の値が 0(黒) と 1(白) の 2 値である $n_x \times n_y$ ピクセルの画像である．元画像と目標画像で異なる画素に注目し，それらを 1 つずつ反転することで画像を変化させていく．例えば図 2.1 では I_S(図 2.1(a)) と I_T(2.1(c)) で値が異なる画素は図 2.1(b) で示す色の付いた部分である．反転する画素の順序決定には，近傍画素の情報をもとにした評価値を用いる．

2.1.1　画素の評価値の計算

画素の反転する順序は，滑らかなモーフィング画像が得られるようにしなければならない．また，中間画像では全ての画素を変化させるわけではないので画素の反転が偏らないようにすることも必要である．提案アルゴリズムでは，反転するのが最も妥当だと思われる画素を決めるために各画素を数値的に評価する．ある画素の $k \times k$ 近傍 ($k > 3$,k は奇数) は，図 2.2 のように，各画素を中心とした $k \times k$ の正方形領域内の注目画素以外の画素とする．評価値は $k \times k$ 近傍の情報から計算する．そのため，画素を反転することで近傍の評価値が変化する．

評価値は近傍から求めた 2 つの値の和とする．1 つは近傍の反対色の画素の数で，O とする．これは図 2.3 のように中心が白なら近傍の黒，黒なら近傍の白の画素の数である．O を評価値としたのは，局所的に見た画像の変化として自然であるというごくシンプルな



(a) I_S　　(b) $I_S \oplus I_T$　　(c) I_T

図 2.1: 画素の反転

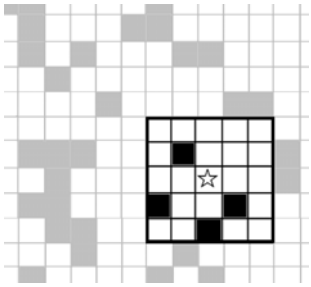


図 2.2: $k \times k$ 近傍の例 (k=5，色の濃い部分が☆の画素の近傍)

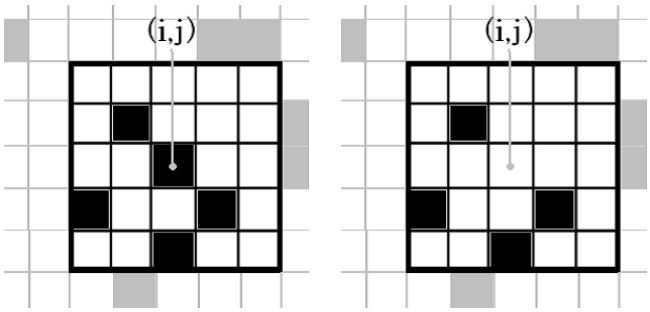
考えからである．逆に，近傍に同色の画素が多い所を反転させるのは反対色に囲まれた画素が増えてしまい，ノイズが入ったように見えてしまう (図 2.4).

もう 1 つは近傍の画素で目標画像と異なる画素の数で，D とする．D は変化が進んでいない，あるいは目標画像との違いが大きい部分の画素を優先的に反転するためのものである．

これらを用いて，画素 (i,j) の評価値 $Eval(i,j)$ は

$$Eval(i,j) = \begin{cases} O(i,j) + D(i,j) & \text{if } I_S(i,j) \neq I_T(i,j) \\ -1 & \text{if } I_S(y,x) = I_T(y,x) \end{cases} \quad (0 \leq i \leq n_y, 0 \leq j \leq n_x) \tag{2.1}$$

とする．(反転が不要な画素の評価値は -1 としたが，これは反転が必要な画素と区別するためで，アルゴリズム上 -1 でなくてもよい．) このため評価値は必ず整数で，近傍を $k \times k$ とすると $-1 \leq Eval(i,j) \leq 2k^2 - 2$ である．



(a) $O(i,j) = 20$ (b) $O(i,j) = 4$

図 2.3: 近傍の反対色の数

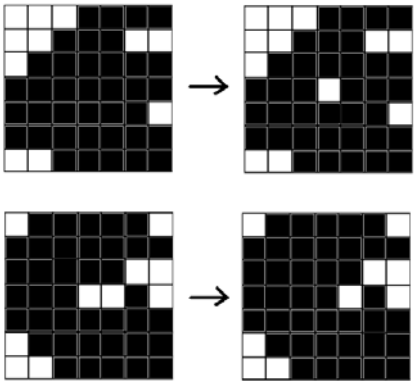


図 2.4: 画素の反転の例 (中心の画素を反転している)

2.1.2 提案アルゴリズム

アルゴリズムでは，反転する画素の順序を格納した配列を作成する．出力である中間画像はその配列を元に作成する．その際は，$\frac{\text{反転する画素の総数}}{\text{中間画像の枚数}}$ 個ずつ元画像の画素を反転してゆき，それを出力すればよい．提案アルゴリズムではまず，反転する順序を決めるために元画像の各画素について，評価値を求める (Algorithm1)．そして元画像が目標画像と一致するまで，評価値最大の画素の中からランダムに選んだ 1 つを反転し，評価値を更新することを繰り返す (Algorithm2)．

2.1.3 評価値の更新

画素の反転が行われると，反転した画素とともにその近傍画素の評価値も変化する．反転した画素を (i,j) とすると，$Eval(i,j)$ は -1 に，近傍の中で反転の必要のある画素は図 **??**のように，(i,j) の反転前と同じ色の画素は D が $+1$，O が -1 され，反対色の画素は D

Algorithm 1 評価値の計算

```
for each (i, j) ∈ I_S do {
  O ← 0,D ← 0
  if I_S(i, j) ≠ I_T(i, j) {
    for each (l, m) ∈ (i, j) の k × k 近傍 do {
      if I_S(l, m) ≠ I_S(i, j) {
        O ← O + 1
      }
      if I_S(l, m) ≠ I_T(l, m) {
        D ← D + 1
      }
    }
    Eval(i, j) ← O + D
  else
    Eval(i, j) ← −1
  }
}
```

Algorithm 2 提案アルゴリズム

入力: $n_y \times n_x$ 2 値行列 I_S,I_T，2 値画像
出力: 整数値配列 $order$，画素を反転する順序
Eval: $n_y \times n_x$ 整数値行列，画素の評価値

```
//ステップ 1
各画素の評価値を計算する
//ステップ 2
while e_max ≥ 0 do {
  e_max ← max(Eval)
  評価値 = e_max である画素の中からランダムに 1 つ選び，(i, j) とする
  I_S(i, j) を反転する
  配列 order に (i, j) を追加する
  Eval(i, j) ← −1
  (i, j) の近傍画素の評価値を更新する
}
```

が-1，O が -1 される．したがって，評価値の更新では近傍の反対色の画素の評価値を -2 するだけでよく，評価値が増加することはない．

2.1.4 計算時間と作業領域

評価値を単純に 2 次元配列に格納した場合，評価値の初期化に $O(k^2 n_x n_y)$ 時間かかる．そして画素の反転順序の決定では，評価値を更新する度に全画素の評価値を参照して最大値を求めなければならないので $O(n_x n_y \cdot n_x n_y) = O(n_x^2 n_y^2)$ 時間かかるので，アルゴリズム全体でも $O(n_x^2 n_y^2)$ 時間となる．作業領域には反転していく元画像を格納する $n_x \times n_y 2$ 次元配列，評価値を格納する $n_x \times n_y 2$ 次元配列と，評価値最大の画素を一時的に格納するための配列が必要である．したがって作業領域は全体で $\mathrm{O}(n_x \times n_y)$ となる．

2.2 アルゴリズムの改善

計算時間について，評価値の初期化ステップと画素の反転ステップのそれぞれについて，計算時間を改善する．

2.2.1 評価値の計算

最初に評価値を計算する際に 1 画素当たり k^2 個の画素を参照するが，先に縦方向 (y 方向) の和を求めておき，それを用いて評価値を計算すれば，1 画素当たりの近傍画素の参照回数を大幅に減らすことができる．(図 2.5) 縦方向の和は 1 画素当たり k 回だから全体で $O(kn_x n_y)$ 時間で，それを用いた評価値の計算も $O(kn_x n_y)$ であるから，初期化全体の計算時間は $O(kn_x n_y)$ となる．

$$O_v(i,j) = \sum_{a=i-\lfloor k/2 \rfloor}^{i+\lfloor k/2 \rfloor} I_S(a,j) \qquad (0 \le i \le n_y, 0 \le j \le n_x) \tag{2.2}$$

$$D_v(i,j) = \sum_{a=i-\lfloor k/2 \rfloor}^{i+\lfloor k/2 \rfloor} |I_S(a,j) - I_T(a,j)| \qquad (0 \le i \le n_y, 0 \le j \le n_x) \tag{2.3}$$

$$Eval(i,j) = \begin{cases} -1 + \sum_{b=j-\lfloor k/2 \rfloor}^{j+\lfloor k/2 \rfloor} O_v(i,b) + D_v(i,b) & \text{if } I_S(i,j) = 0 \text{ and } I_T(i,j) = 1 \\ -1 + \sum_{b=j-\lfloor k/2 \rfloor}^{j+\lfloor k/2 \rfloor} O_v(i,b) + 5 - D_v(i,b) & \text{if } I_S(i,j) = 1 \text{ and } I_T(i,j) = 0 \end{cases} \tag{2.4}$$

実装の際は，画像を行列とすると行ごとに O_v と D_s を記憶しておき，$Eval$ を求めればよい．そのため O_v と D_v の値を一時的に格納する領域として，それぞれ画像の幅 n_x のサイズの配列を用意する．

Algorithm 3 改良アルゴリズム-評価値の初期化

入力: $n \times n$ 2 次元配列 I_S,I_T

出力: 1 次元配列 $order$

$Eval$: $n_y \times n_x$ 2 次元配列, 評価値

$EvalArrayLength$:評価値ごとの配列の長さ

$EvalIndex$:評価値ごとの配列での画素のインデックス

```
for i = 0 to k^2 − 1 do {
  EvalArrayLength[i] ← 0
}
for i = 0 to n_y − 1 do {
  for j = 0 to n_x − 1 do {
    O_v[j] ← 0,D_v[j] ← 0
    for l = i − ⌊k/2⌋ to i + ⌊k/2⌋ do {
      O_v[j] ← O_v[j] + I_S(i, j)
      if I_S(i, j) ≠ I_T(i, j) {
        D_v[j] ← D_v[j] + 1
      }
    }
  }
  for j = 0 to n_x − 1 do {
    if I_S(i, j) ≠ I_T(i, j) {
      for l = j − ⌊k/2⌋ to j + ⌊k/2⌋ do {
        if I_S(i, j) = 0 {
          Eval(i, j) ← Eval(i, j) + O_v[l]
        else
          Eval(i, j) ← Eval(i, j) + (k − O_v[l])
        }
        Eval(i, j) ← Eval(i, j) + D_v[l]
      }
      Eval(i, j) ← Eval(i, j) − 1
      e ← Eval(i, j),eLen ← EvalArrayLength[e]
      EvalArray[eLen] ← (i, j)
      EvalArrayLength[e] ← EvalArrayLength[e] + 1
      EvalIndex(i, j) ← eLen
    else
      Eval(i, j) = −1
    }
  }
}
```

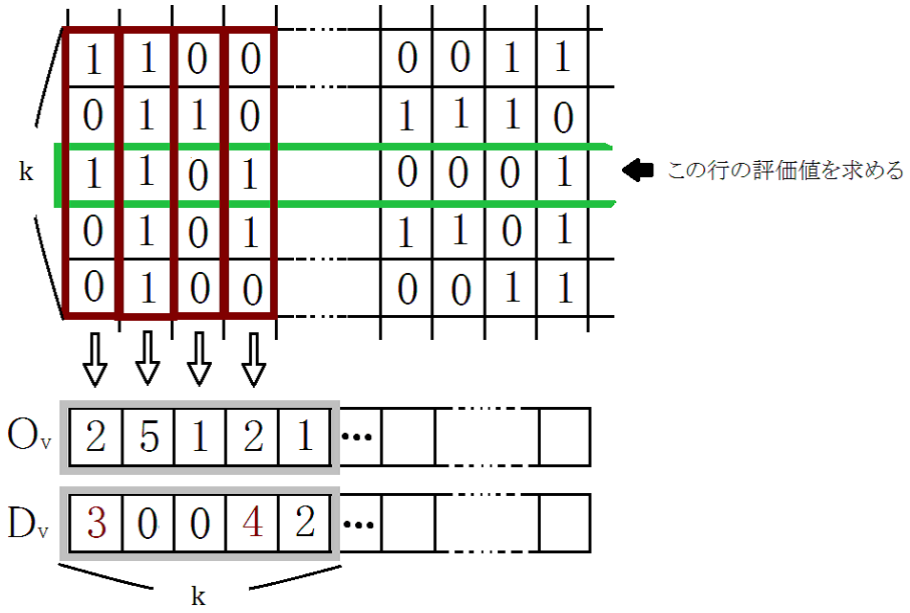


図 2.5: 評価値の計算の高速化

2.2.2 反転順序の決定

評価値の最大値を求め，その画素を探索する処理に毎回全画素を参照するのは効率が悪い．そこで，評価値が 0〜$2k^2-2$ の整数値であることを利用して，評価値ごとに画素の位置を格納する配列 (図 2.6) を作っておけば，評価値最大の画素は定数時間で見つけることができ，計算時間を大幅に減らすことができる．提案した評価方法では評価値は増加することはない．したがってある時点で最大評価値の配列が空になれば，その次に大きな評価値の配列で空でないものを見つけ，それをその時点で最大の評価値としてよい．

更新時には，反転した画素を配列から削除する．削除では，配列の最後尾の要素を削除したい要素に上書きする．そして，近傍で評価値の変化した画素を更新後の評価値の配列へ移動しなけらばならない．これには元の評価値の配列から要素を削除し，更新後の評価値の配列へ挿入することが必要になるが，これはそれぞれの配列のサイズ，更新前の評価値，更新前の評価値の配列でのインデックスの情報があれば定数時間で行うことができ，1 回の反転につき高々k^2 回である．したがって，反転順序の決定にかかる時間は $k^2 \cdot O(n_x n_y) = O(k^2 n_x n_y)$ である．

2.2.3 近傍の範囲の変更

評価値 $Eval$ が 0 の画素が多数残った場合，それらは単にランダムに反転されてしまうことになる．そこで，残った画素が多かった場合に近傍の範囲を広げて評価値の再計算を行う．

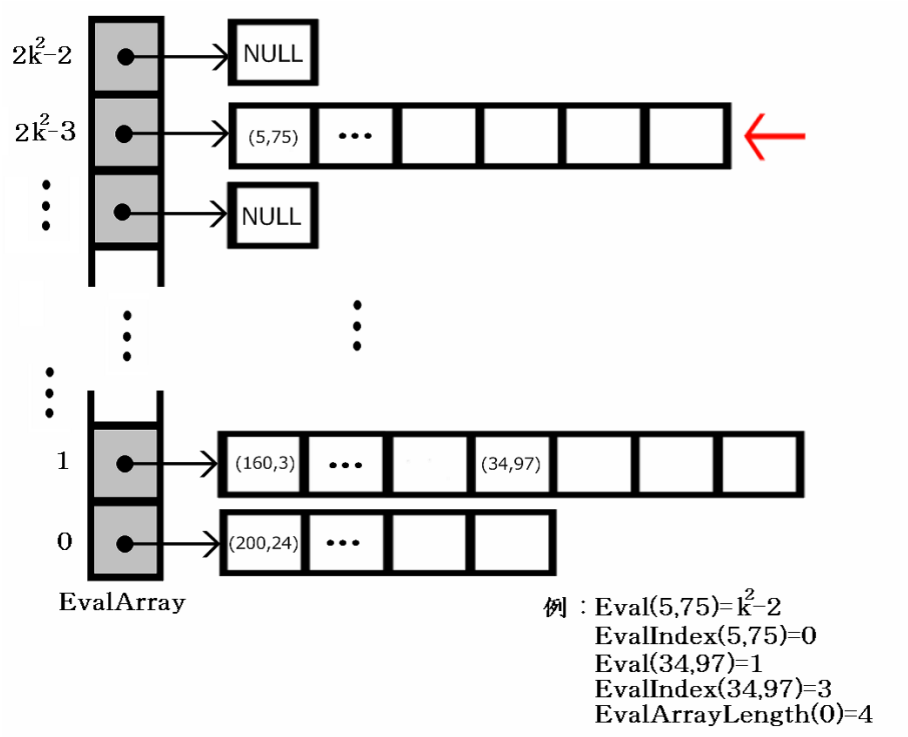


図 2.6: 評価値ごとに画素の位置を格納した配列

$k \times k$ 近傍の場合，最後に残る評価値 0 の画素は高々 $\frac{n_x n_y}{k^2}$ 個である．新しい近傍の範囲を $ck \times ck$ とすると 1 画素当たりの計算時間は $k \times k$ の評価値が 0 であることから再計算の際には無視してよく，$c^2k^2 - k^2 = (c^2-1)k^2$ 評価値の再計算は全体で $O(\frac{n_x n_y}{k^2} \times (c^2-1)k^2) = O(c^2 n_x n_y)$ 再計算した評価値の更新も 1 回の更新で $(c^2-1)k^2$ 個の画素を参照し，それが高々 $\frac{n_x n_y}{k^2}$ 回行われるので，$O(c^2 n_x n_y)$ である．$c < k$ とすると，反転順序の決定における全体の計算量は $O(k^2 n_x n_y)$ のままである．

2.2.4 計算時間と作業領域

評価値の初期化に $O(k n_x n_y)$，反転順序の決定処理が $O(k^2 n_x n_y)$ であるから，改良後のアルゴリズムの計算時間は全体で $O(k^2 n_x n_y)$ である．

作業領域は，改良前のものに加えていずれも長さが $O(n_y \times n_x)$ である評価値毎の配列，長さ $k^2 - 1$ の評価値毎の配列へのポインタの配列 $(EvalArray)$，画素の位置から評価値毎の配列でのインデックスを参照する $n_y \times n_x 2$ 次元配列 $(EvalIndex)$ とそれぞれの評価値の配列のサイズを格納する長さ k^2 の配列も必要になる．したがって，改良アルゴリズムに必要な作業領域はおよそ元画像の 5 倍となり，$O(n_y n_x)$ である．[1]

[1]ただし，改良前に用いていた，評価値最大の画素を一時的に格納する配列は不要．

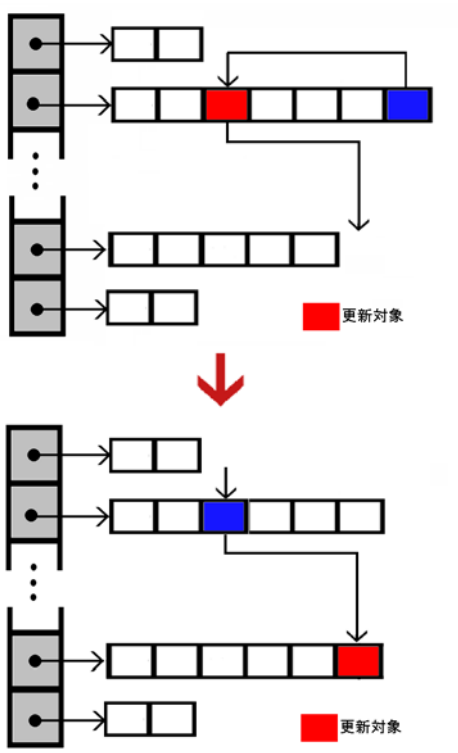


図 2.7: 評価値ごとの配列における近傍画素の更新処理

$EvalArray$ の要素が指す配列の長さは $O(n_y \times n_x)$ と述べたが，評価値の配列は合計のサイズは $n_y n_x$ である．なぜなら，各画素がいずれかの評価値に格納されるのは高々一回だからである．もちろんいずれかが $n_y n_x$ となる可能性もある．例えば，元画像の画素が全て白で，他方が全て黒の場合，ほとんど全ての画素の評価値は $k^2 - 1$ であるから $EvalArray[k^2 - 1]$ の指す配列の長さは約 $n_y n_x$ となる．しかし，長さ $n_y n_x$ の配列 $k^2 - 1$ 個分の領域は無駄があるので，可変長の配列 [4] で実装し，必要な分だけ確保するとよい．

2.3 作業領域の縮小と計算時間

改良アルゴリズムでは，作業領域として元画像 5 個分のサイズが必要であった．ここでは，より少ない領域でアルゴリズムを実現した場合の計算時間の変化について述べる．

まず評価値の計算ステップでは，2.2.1 では画像の幅のサイズの配列を 2 個用いたが，近傍の幅 k の配列が 2 個でも可能である．これによる計算時間のオーダーへの影響はない．

次に反転順序の決定であるが，全ての画素を評価値ごとに格納するのは更新にかなりの領域 ($Eval$,$EvalArray$) を必要とするため，最大の評価値を持つ画素のみ配列に格納することを考える．そして，その配列が空になればそれより 1 小さい評価値の画素を配列に格納する，これを繰り返せばよい．評価値の更新は，最大値画素の配列内に含まれていて，かつ評価値が変化する近傍画素を削除するだけでよい．この処理は画素が配列の何番目に

Algorithm 4 改良アルゴリズム-反転順序の決定

$e_{max} \leftarrow max(Eval)$
while $e_{max} \geq 0$ **do** {
  $EvalArray[e_{max}]$ の指す配列の中からランダムに 1 つ要素を選び，(i, j) とする
  $I_S(i, j)$ を反転する
  配列 $order$ に (i, j) を追加する
  $EvalArray[e_{max}]$ の指す配列の最後尾の要素を要素 (i, j) の場所に格納 ((i, j) を削除する)
  $EvalArrayLength[e_{max}]EvalArrayLength[e_{max}] - 1$
  $Eval(i, j) \leftarrow -1$
  (i, j) の近傍画素の評価値を更新する
  while $e_{max} \geq 0$ and $EvalArrayLength[e_{max}] = 0$ **do** {
    $e_{max} \leftarrow e_{max} - 1$
  }
  if $e_{max} = 0$ and $EvalArrayLength[0]$ が大 {
    $k \leftarrow 2k + 1$　//近傍のサイズを大きくする
    評価値を再計算
  }
}

あるかの情報 ($EvalIndex$) が分かれば更新 1 回当たり $O(k^2)$ で済む．また，各画素の評価値を格納する配列 $Eval$ は不要になる．しかし，$O(kn_yn_x)$ 時間かかる評価値の計算が $k^2 - 1$ 回行われるので，この方法では $O(k^3n_yn_x)$ の計算時間になる．したがって作業領域の縮小を行った結果，必要な配列を

- 長さ k の配列
- 元画像 (画素反転用)
- 最大の評価値を持つ画素の配列 (長さは最悪で n_yn_x)
- 配列における画素のインデックス ($n_y \times n_x$)

とした場合，計算時間は全体で $O(k^3n_yn_x)$ になる．画像のサイズが大きい場合などは，こちらのバージョンを使うとよいかもしれない．

2.4　出力結果

2.4.1　2 値画像のモーフィング

提案アルゴリズムによる出力結果を図 2.8 に示す．[2]提案アルゴリズムで画像の滑らかな変化がある程度実現できていることがわかった．

2.4.2　近傍サイズによる結果の違い

同じ入力画像に対し，異なる近傍サイズを用いた結果を図 2.9，図 2.10，図 2.11 に示す．近傍サイズが大きいほど変化は滑らかになるが，白黒画素が偏りがちになることを確認できた．

2.4.3　カラー画像への応用

提案アルゴリズムを RGB カラー画像に用いる方法を提案する．2 値画像は各画素が 1 つしか色成分を持っていないが，カラー画像では各画素は赤緑青の 3 つの色成分を持っている．そこで，まずカラー画像のデータをまず RGB ごとに 2 値化し，各色成分に対して個別に提案アルゴリズムを適用する．これらの出力結果を再び統合したものを中間画像として出力する．つまり，RGB それぞれが 2 通りの値を持つので出力画像は 8 色のカラー画像となる．

出力結果は図 2.12，図 2.13 に示すように，それぞれの色成分が強く出る中間画像が得られた．これは，それぞれの色成分を独立に処理したことが原因である．したがって，他の色の変化が影響するような評価値を導入し，それぞれの色の画像を並行して変化させればことで出力は改善すると考えられる．

[2]画像の枚数が少ないため，近傍のサイズ変更による評価値の再計算は今回の出力結果には影響しなかった．

図 2.8: 出力結果 1-提案アルゴリズム

図 2.9: 出力結果-提案アルゴリズム ($k = 3$)

図 2.10: 出力結果-提案アルゴリズム ($k = 13$)

図 2.11: 出力結果-提案アルゴリズム ($k = 23$)

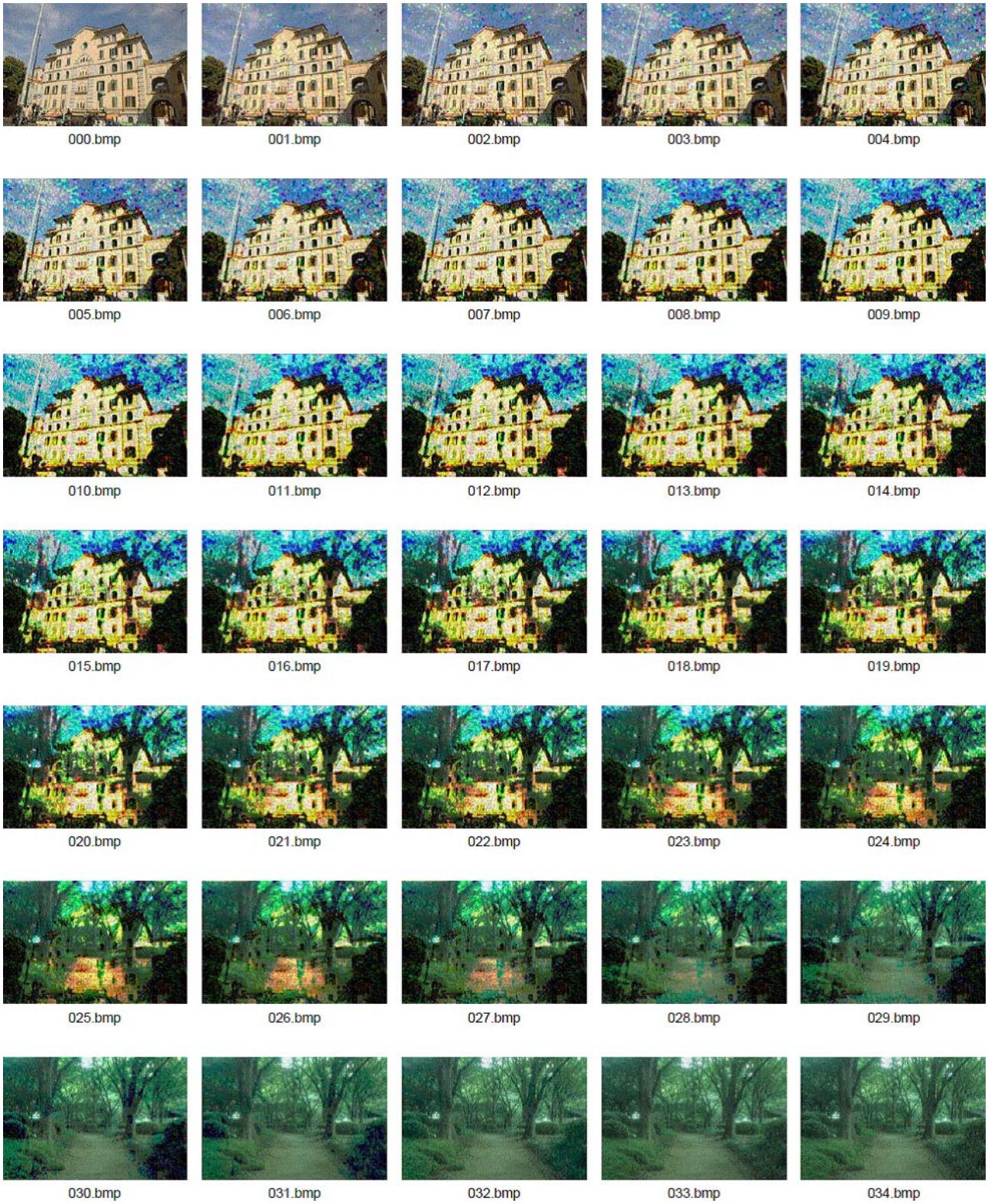


図 2.12: 出力結果-提案アルゴリズム (カラー)

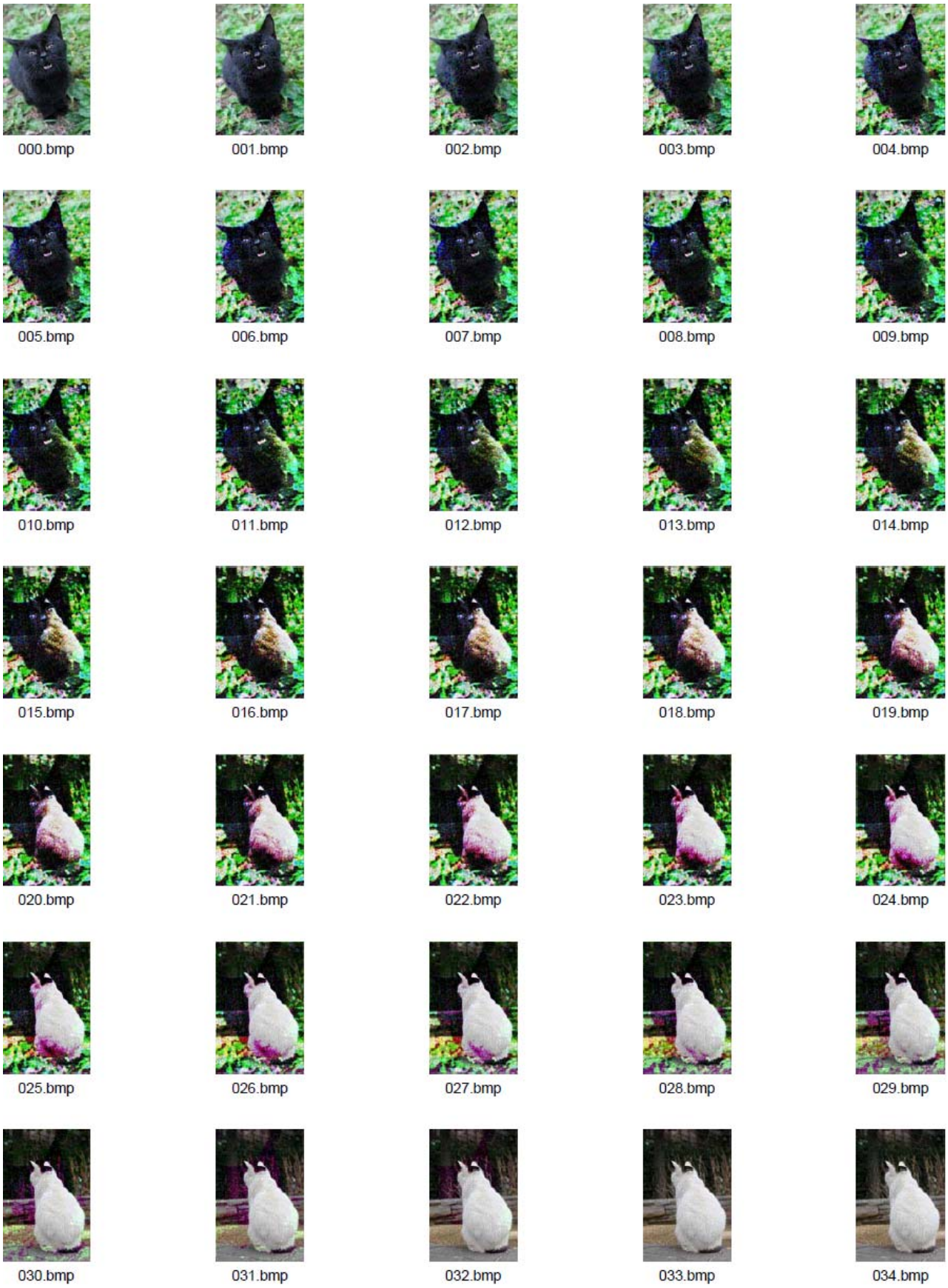


図 2.13: 出力結果-提案アルゴリズム (カラー)

第3章　おわりに

本論文では，2値画像のモーフィングアルゴリズムを提案し，それが画素数に対して線形時間で計算できることを示した．作業領域についてはオーダーでの改善は出来なかったが計算時間を犠牲にすることで縮小が可能であることを示した．出力結果については2値画像では画像の滑らかな変化を実現できたが，カラー画像への応用は原色が強く出るため適さないことが分かった．

今後の課題としては，出力結果を改善するため，または更に高速化が可能となるようなより良い評価方法の開発が挙げられる．

謝辞

浅野哲夫教授，上原隆平准教授，清見礼助教はじめ，浅野・上原研究室の皆様には公私に渡ってお世話になりました．この場を借りてお礼申し上げます．特に浅野教授には丁寧なご指導を頂きました．深く感謝いたします．

参考文献

[1] Ste'phane Marchand-Maillet, Yazid M. Sharaiha., "*Binary Digital Image Processing.*", Academic Press, 2000.

[2] Smythe DB., A two-pass mesh warping alogrithm for object transformation and image interpolation. Technical Report 1030, ILM Computer Graphics Department, Lucasfilm, San Rafael, Calif, 1990.

[3] George Wolberg., "Image morphing: a survey", The Visual Computer Volume 14, Numbers 8-9 pp360-372 Springer Berlin / Heidelberg 1998.

[4] Kurt Mehlhorn, Peter Sanders., "*Algorithms and Data Structures: The Basic Toolbox.*", Springer, 2008.