カラリゼーションに関する基礎的研究（第Ⅱ報）

衣松　宏晃[a)]・坂本　眞人[b)]

Fundamental Study on Colorization Ⅱ

Hiroaki KINUMATSU, Makoto SAKAMOTO

Abstract

Colorization is a computer-assisted process of adding color to a monochrome photograph or film. The former is said to be the hand-coloring, and the latter is said to be the film colorization. Generally, colorization is an expensive and time-consuming process. Recently, various software of colorization are introduced on the Internet, but operation is difficult for a beginner. We have investigated about software of colorization. In this paper, we continue the fundamental study on colorization, and show the program for colorization that was updated. We use C for implementation of program. In this process, we use the image which we appoint a color and the place that we want to paint on a monochrome image. We call it "pilot image". We need it besides monochrome image.

Keywords: Black-White Image, Colorization, HSV Color-Space, Pilot Image, RGB Color-Space

1. はじめに

1.1 研究背景

カラリゼーション(カラライゼーション)という言葉は、1970年にWilson Markle氏が導入された用語である[1)]。この言葉は白黒画像をカラー化するコンピュータ支援プロセスを意味する。

白黒で表現される空は、青空なのか夕焼けなのかはすぐに判断するのは難しい。白黒だけの表現で得られる情報は、カラーで表現で得られる情報より少ないためである。白黒のメディアといえば、白黒写真、白黒映画などが該当する。

これらをカラー化することで、白黒ではわからない情報を得られることができるだろう。

現在、多くの技術者により一般のユーザにも着色ができるよう、カラリゼーションソフトの開発が行われている。だが、完成度の高いカラリゼーションソフトは高価なものが多くフリーソフトのものは少ない。そこで、本研究はフリーソフトという前提で綺麗な着色が行えるカラリゼーションソフトのプログラム開発を行った。

卒業論文で、カラリゼーションソフトは既に試作している。本論文では、より実用的なカラリゼーションソフトの開発を目指し試作されたカラリゼーションプログラムを改善したものを紹介する。改善案のテーマは、色の伝搬の効率化である。色の伝搬が効率化されると、ユーザの負担を軽減することができる。

a) 情報システム工学専攻大学院生

b) 情報システム工学科准教授

1.2 着色方法

白黒画像の着色手法は大きく分けて2つあり、画像を読み取って自動で着色するものと、ユーザが着色する色と場所をある程度指定してそれを基に着色する方法がある。本論文のプログラムは後者の手法を使う。

本論文のプログラムは白黒画像の他に、「種画像」を用意する必要がある。種画像とは、白黒画像に塗りたい色・場所を指定した画像のことで便宜上名づけた。種画像は、本論文のプログラムとは別の画像処理ソフトで白黒画像に塗りたい色・場所を描画する必要がある。本論文のプログラムは白黒画像と種画像の 2 つの画像を使って着色画像を生成する。入力データとして白黒画像と種画像、出力データとして着色画像となっている。

種画像を用いたプログラムでカラリゼーションソフトは著者の卒業研究で試作している。これを改善するために、色の伝搬の効率化を図っている。具体的に説明すると、種画像に少ない描画で着色できるようなプログラム改善を目指している。色の伝搬の効率化を実現するために、既存の探索機能に加えて新しい探索機能「記憶された色探索」を実装した。

2. 理論

2.1 RGB色空間

赤(Red)、緑(Green)、青(Blue)の三つの原色を軸とした(図1)色の表現方法の一種である。これらを混ぜて幅広い色を再現する加法混色の一種であり、3つの色を適切な量

を加えることで白くなる。RGB の各要素（R 要素、G 要素、B 要素）は数字で表現され、各要素の数字の組み合わせで、色と鮮やかさ・明るさが決まる。言い換えると、RGB の各要素の数値が全てわからないとどんな色なのかすら特定できないことになる。RGB 色空間は主にブラウン管や液晶ディスプレイ、デジタルカメラなどで画像表現に使われている[9]。

図 1. RGB 色空間[9].

RGB 色空間をデジタルデータとして扱う場合は、基本的に RGB 各要素の値は 0～255 までの段階があり、8 ビットデータとして扱う。この場合、一つの画素（ピクセル）の色を表すためには、赤(R)、緑(G)、青(B)で 8×3=24 ビットのデータが必要となる。24 ビットで扱える数値の範囲は 0～16777215 であり、約 1600 万色(24bit カラー)の色が使用できることを示す。多くのデジタル画像の色の表現はこの RGB 色空間で表せることができる。

2.2　HSV 色空間

色相（Hue)、彩度（Saturation)、明度（Value）からなる[7]。色相（Hue）は、赤・黄色・緑・青のような色の種類のようなもので、有彩色を分類することができる。また、赤・黄色・緑・青・紫を順にならべ、赤と紫をつなげた輪「色相環」で表現できる(図 2)。彩度（Saturation）は、色の鮮やかさを示す尺度のことで、彩度が高い程原色に近くなり、低い程灰色さが見立ちくすんだ色になる。明度（Value)とは、色の明るさを示す尺度のことで、白が最大の値をとり、黒が最小の値をとる。

HSV の RGB 色空間と同様に各要素（H 要素、S 要素、V 要素）の数値の値の組み合わせで色を表現する。H 要素は色相環で表現しているので、一般的に値は色相環に沿った角度の値 0～360 となっている。また、S 要素・V 要素はどちらも 0～100%の範囲で表しているため、0～100 までの値が一般的である。

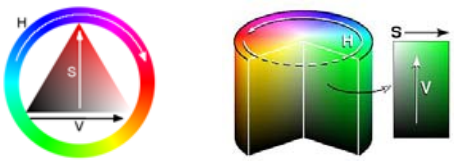

図 2. HSV 色空間[7].

2.3　BMP 画像

本論文のプログラムは、BMP（ビットマップ）画像を使用する。BMP 画像とは、コンピュータグラフィックスにおける画像形式の一つである。BMP 画像は、画像の最小要素「ピクセル」に画像の色を示す RGB 情報が格納されている(図 3)。点（ピクセル）が線上に並んだ（ラスター）の集まりであるとして扱われるためラスターイメージと呼ばれることもある[2]。

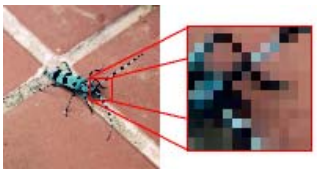

図 3. BMP 画像[2].

本論文のプログラムでは、プログラム実行時に BMP 画像を 1 ピクセル単位で読み込んでいく。便宜的にするため、左から右へ、上から下の順で構造体に RGB 情報を格納する(図 4)。

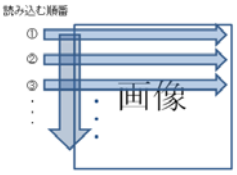

図 4. 画像の読み込み.

本来 BMP 画像は左から右へ下から上の順で RGB 情報を格納している。これは機械独立ファイル形式（幾何学的な X 軸、Y 軸方向に座標を指定する）として設計されたためである[11]。このまま使用することは、プログラム製作側にとって不便であるため構造体を用いた格納をしている[8]。

2.4. 着色

2.4.1　着色の流れ

着色作業の流れを図 5 に示す。

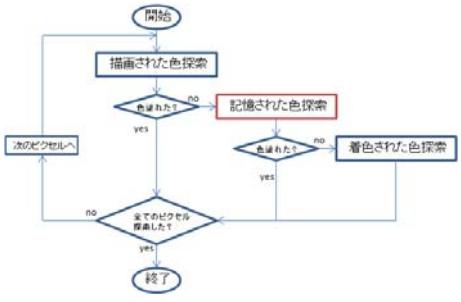

図 5. 着色の流れ.

描画された色探索、記憶された色探索、着色された色探索で色を探し着色する。このプログラムでいう「着色」とは、白黒画像の HSV 情報を変化させたことである。

探索は 1 ピクセル単位で左から右へ上から下へ行っていく。探索の対象ピクセルは、常に白黒画像・種画像上のピクセルを連動して移動していく(図 6)。つまり対象ピクセルは、常に白黒画像と種画像の同じ位置にある。対象ピクセルの探索を終えたら次のピクセルに対して探索を行う。次のピクセルとは右隣、端にいた場合は一つ下の行の左端のピクセルを対象ピクセルとする。全てのピクセルに対して探索を終えたら着色作業が終了する。

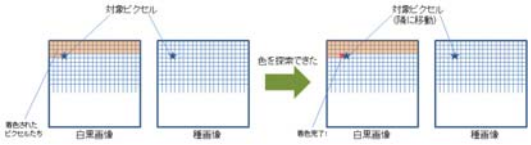

図 6. 対象ピクセルの動き.

2.4.2　描画された色探索

白黒画像と種画像、この 2 つの画像の RGB 情報を比較して種画像に描画されたピクセルを探していく。描画されたピクセルを見つけたら、そのピクセルの H 要素を白黒画像にある対象ピクセルの H 要素に代入(着色)する。

探索方法は、2 つの画像の RGB 情報を比較する。対象ピクセルだけ調べるのではなく、対象ピクセルを中心とした正方形状の探索範囲で周りのピクセルも調べていく。正方形内に描画された(白黒画像と種画像の RGB 情報が異なっている)ピクセルがなければ正方形を大きくしてまた探していく（正方形の最大サイズは画像によって変わる)。

描画されている、つまり白黒画像と種画像の RGB 情報が異なっているピクセルがあれば、白黒画像における対象ピクセルの HSV 要素の H 値を描画されたピクセルの H 値に割り当てる、つまり着色する。そして次のピクセルを対象ピクセルとして再び描画された色探索を行う。

最大サイズになった正方形内に描画されたピクセルがなければ次の記憶された色探索に移行する。

描画された探索を説明した図 7 に示す。

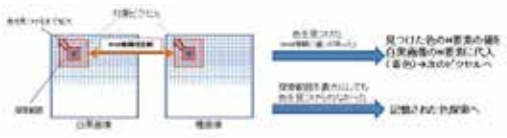

図 7. 描画された色探索.

2.4.3　記憶された色探索

プログラム上で記憶された色の情報を用いて記憶した色と似たような場所に対象ピクセルがあればその色を着色する。

最初に記憶する色として、白黒画像、種画像の 2 つの画像において一番最初に描画されている、つまり白黒画像と種画像の RGB 情報が異なっている色の情報を格納する。

格納する情報は、描画されている色の HSV 情報の H 値、そのピクセルと同じ場所にある白黒画像の RGB 情報各要素の値などを格納する(図 8 参照)。

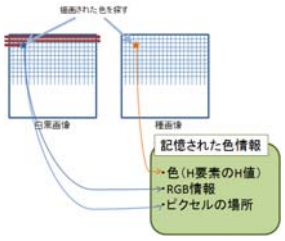

図 8. 色情報格納.

記憶する色情報は増えさせていく。増やし方として描画された探索で色を見つけた時、記憶されていない色ならば新しい色情報として情報を格納する。ただし、記憶されている色でも記憶された色の RGB 情報各要素と大きく違うような違う場所にある記憶されている色の場合は格納している。

このようにして記憶する色情報を増加させているが記憶する情報の数は有限にしなくてはいけない。記憶する情報の数は有限にしないとコンピュータに多大な負担がかかり、時間もかかるためである。

記憶された色探索は、記憶された色情報から着色すべき色を見つける。白黒画像における対象ピクセルの RGB 要素と記憶された色情報の RGB 要素を比較して、似たような RGB 要素ならその RGB 要素を持つ描画されている色の H 値を対象ピクセルの H 値に割り当てる(図 9)。RGB 要素の比較の方法は 3 次元の点の距離で調べている。小さい数の方が似たような RGB 要素と言える。

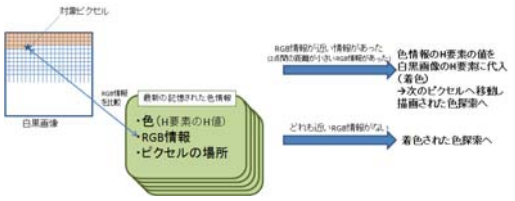

図 9. 記憶された色探索.

記憶された色情報の RGB 要素と全く揃わなかったら最後の手段として着色された色探索へ移行する。

2.4.4　着色された色探索

白黒画像と種画像の HSV 情報を比較し、既に着色された色を探す。描画された探索同様、正方形の探索範囲で着色された色を探す(図 10)。

探索は左から右へ上から下へ行っているので、既に着色

された色は対象ピクセルの左か上方にある。よって探索はピクセルの左側と上方だけに注目して色を探す。着色された色の判別の仕方は白黒画像と種画像のHSV情報のH値で判断する。描画された色探索で描画された色(ピクセル)を探しだせていないので、対象ピクセルの周りには描画された色がない。

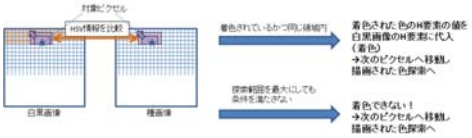

図 10. 着色された色探索.

3. 開発環境

以下の環境で開発した。

プロセッサ	Intel(R) Core(TM) i7　2.93GHz
メモリ	4GB
OS	Windows Vista
プログラム言語	C
環境	Borland C++ Compoler
画像	BMP 画像(24bit カラー)

4. 結果

このプログラムで着色した画像例を示す（図 11、12)。左上が白黒画像、右上が種画像、左下が着色画像となる。

図 11 のように白黒の差があまりない画像ならば、種画像に一本の描画で着色することが可能となる。

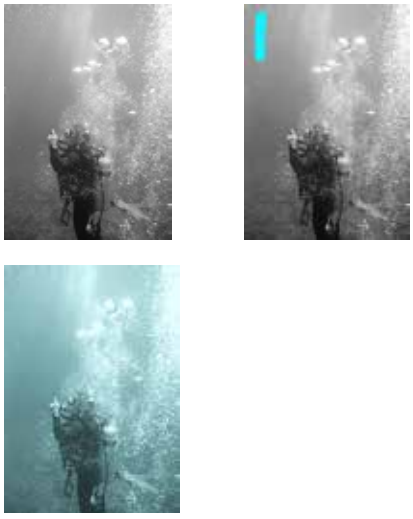

図 11. 実行結果 1(左上:白黒画像 [10]　右上:種画像 左下:着色画像).

図 12. 実行結果 2.（左上:白黒画像 [5]　右上:種画像 左下:着色画像).

5. 考察

今回は色の伝搬の効率化を図るため、記憶された色探索を実装した。色の伝搬の効率化を図る指標として、種画像にあまり描画しないようにし、少ない描画でどれだけ色が伝搬できるかを検証する。サンプル画像として以下の画像サンプル 1 を用意した（図 13)。

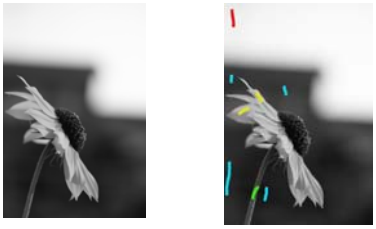

図 13. サンプル画像 1.

サンプル画像の種画像のほうは敢えて描画する箇所を少なくした（図 14)。これらを使って、記憶された色探索の実装前のプログラムと実装済みのプログラムで比較する。

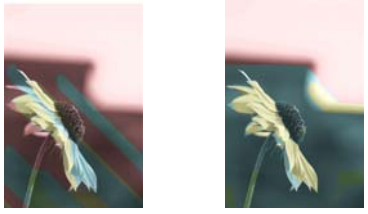

図 14. 比較 1（左：実装前　右：実装済み）.

実装済みの着色結果のほうが色が伝搬できている。

また別の画像での実行結果を比較する。サンプル画像 2 を図 15 に示す。

図 15. サンプル画像 2(左：白黒画像 [6]　右:種画像).

実行結果は図 16 になる。

図 16. 比較 2(左：実装前　右：実装済み).

実装済みの実行結果のほうが少し色がはみでているところがあるが、境界線を塗り分けた着色ができている。モノとモノとの RGB 情報の差がはっきりしていると綺麗な着色ができる。

このように少ない描画の種画像での実行結果を見ると、色の伝搬の効率化は成功しているといえる。本プログラムでは綺麗な着色画像にするために、なるべく細かく種画像に描画しないといけない。綺麗な着色を目指して種画像に細かく描画した結果、実装前と実装済みを比べてみると若干実装済みでのプログラムの方が種画像に描画している箇所が少ない程度で、あまり差がないという結果になってしまった。これは実装前では着色できていたところが実装済みでは着色できないということが起きてしまったからである。

6. おわりに

今回、色の伝搬の効率を図るため「記憶された色探索」を実装し、色の伝搬の効率を僅かながら改善することができた。しかしながら、まだ実用できるレベルには達していない。サイズの大きい画像に対しての処理時間短縮の改善や BMP 画像以外の画像の着色などができるようにしていき、実用可能なレベルに近づけたい。

今後の発展として、色の再着色（リカラリゼーション）がある（図 17)。リカラリゼーションは、本論文の種画像みたいに色を変えてほしい色と箇所を描画して再着色するプログラムである [3]。リカラリゼーションは、住宅内外装業界での外装の色の効果をシミュレート、アパレル業界でのカラーシミュレーションに貢献することができる。似たような手法で着色しているので、工夫すればリカラリゼーション機能を実装できるのかもしれない。

図 17. リカラリゼーション [3].

また白黒の静止画ではなく、動画のカラー化にも発展させていきたい。カラリゼーションの技術により白黒の映画の着色化は既に成功している [4]。映画の着色には賛同している声もあれば批判的な声もある。白黒ならではの良さがあるのだ。現に世界的な映画賞でモノクロサイレント映画が受賞している。監督は白黒とサイレントだからこそできたと語っている。このように安易にカラー化することは良くないケースがある。もちろん、カラー化することは完全に良くないとは言えない。カラー化することで、昔の情景を取り戻したり、別の情景を表現することができる。カラー化することは間違いではなく、安易にカラー化することが間違っているのだ。このことに注意してカラリゼーションソフトを発展してほしい。

参考文献

1) Anat Levin, Dani Lischinski, Yair Weiss, Colorization using Optimization. [Online] http://www.cs.huji.ac.il/~yweiss/Colorization/colorization-siggraph04.pdf
2) ビットマップ画像. [Online] http://ja.wikipedia.org/wiki/
3) Colorization Using Optimization. [Online] http://www.cs.huji.ac.il/~yweiss/Colorization/
4) 映画の着色化. [Online] http://ja.wikipedia.org/wiki/
5) ガーベラ-モノクロ写真|アノ　ソラヲ　ワスレナイ. [Online] http://ameblo.jp/color-of-design/entry-11360265838.html
6) 花、犬、空。-モノクロームダイアリーズ. [Online] http://d.hatena.ne.jp/monochrome_diaries/20091127/1259323600
7) HSV 色空間. [Online] http://ja.wikipedia.org/wiki/
8) プログラミング-[物理のかぎしっぽ]. [Online] http://hooktail.org/computer/
9) RGB. [Online] http://ja.wikipedia.org/wiki/RGB.

10) 写真共有サイト「フォト蔵」. [Online]
http://photozou.jp/photo/show/79815/37867801
11) Windows bitmap. [Online]
http://ja.wikipedia.org/wiki/Windows¥_bitmap