ディジタル画像のモノクローム化 (short v.)

1170999　高田喜朗

2014 年 3 月 1 日

1　実験の目的

本実験課題では，ディジタル画像のモノクローム化を扱う．モノクローム化とは，入力画像の各画素を，同じ輝度の無彩色に置き換えて出力することである．R, G, B 情報から輝度を得る方法として，NTSC カラーテレビジョン方式 [5] で定義されている以下の計算式がよく知られている [1, 6].

$$Y = 0.299\,R + 0.587\,G + 0.114\,B. \qquad (1)$$

ただし，R, G, B は入力画像の画素の R, G, B 成分値，Y は輝度を表す．

本実験課題の目的は，モノクローム化を行うプログラムを作成し，上記の計算式によって実際にモノクローム化が行えることを確認することである．作成したプログラムにサンプル画像を入力し，出力画像を観察する．また，その際の実行時間を測定する．加えて，輝度を R, G, B 成分値の平均とする単純な方法（つまり $Y = (R+G+B)/3$）についてもプログラムを作成し，どちらが自然な出力画像を得られるか，及び実行時間に差があるか，比較する．なお，計算式 (1) を用いる方法を以下では NTSC 法と呼び，R, G, B 成分値の平均を用いる方法を単純平均法と呼ぶ．

2　プログラムの外部仕様

プログラムは Java で記述し，シェル上で実行するコマンドラインアプリケーションとして作成する．主クラス名は `ImageMonochromatizer` とする．プログラムを実行する際は，シェル上で以下のように入力する．

`$ java ImageMonochromatizer` *fin fout*

ただし，*fin*, *fout* はそれぞれ入力ファイル名及び出力ファイル名である．上記を実行すると，入力ファイル *fin* から画像を読み込んで，モノクローム化された画像を出力ファイル *fout* に書き出す．

画像ファイルの読み込み及び書き出しには Java Image I/O API[7] の `ImageIO` クラスを使用する．Portable Network Graphics (PNG) や JPEG など `ImageIO.read` メソッドが対応する画像ファイル形式であれば入力ファイルとして使用できる．

出力画像ファイル形式は PNG とする．プログラムを単純にするため，出力ファイル名の拡張子に応じて画像ファイル形式を選択するといった処理は行わない．

同じくプログラムを単純にするため，上で述べたことを満たさない入力に対する動作は規定しないこととし，エラー処理は特に行わない．

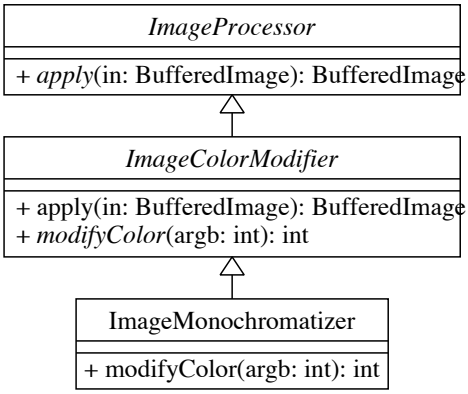

図 1　クラス図

3　プログラムの内部仕様

プログラムの主要部分は次のようになる．入力画像の各座標 (x, y) について下記を行う．

1. 座標 (x, y) の画素の色情報 (A, R, G, B) を読み出す．ただし，A, R, G, B はそれぞれ，不透明度，赤の強さ，緑の強さ，青の強さである．
2. 式 (1) に従って，R, G, B から輝度 Y を計算する．
3. 出力画像の座標 (x, y) の画素の色を (A, Y, Y, Y) にする．これにより，この画素は輝度 Y の無彩色となる．

入力画像及び出力画像は，`java.awt.image` パッケージの `BufferedImage` オブジェクトで表す．座標 (x, y) の画素の色情報は `getRGB(`x`,` y`)` で得られる．画素の色の設定は `setRGB(`x`,` y`,` *rgb*`)` で行う．`getRGB` の戻り値及び `setRGB` の引数 *rgb* は，A, R, G, B の 4 成分（各 8 ビット）を結合した，32 ビットの整数値である．

プログラムの内部は，将来の拡張性を考慮して，単一のクラスではなく，継承関係にある 3 つのクラスで構成する．クラス構成を表す UML クラス図 [3] を図 1 に示す．

`ImageProcessor` は各種の画像処理を表す抽象クラスであり，入力画像を受け取って出力画像を返す抽象メソッド `apply` を宣言する．

`ImageColorModifier` は，上述のアルゴリズムに従って出力画像の各画素の色を設定していくよう，メソッド `apply` を実装する．ただし，入力画像の画素の色から出力画像の色を計算する部分は，抽象メソッド `modifyColor` を呼び出すことで行う．モノクローム化に限らず，各画素の色を他の画素とは無関係に変更する処理は，このク

表 1　入力画像の諸元

	形式	幅・高さ (画素)	ファイルサイズ
画像 A	JPEG	320 × 240	約 39KB
画像 B	JPEG	320 × 240	約 80KB
画像 C	JPEG	1600 × 1200	約 559KB

ラスを継承して `modifyColor` を実装することで実現できる．例えば，濃度変換 [2] などもこのような処理に当たる．

`ImageMonochromatizer` は，入力された色情報 (A, R, G, B) から輝度 Y を計算し，色 (A, Y, Y, Y) を返すよう，`modifyColor` を実装したクラスである．

このほか，整数値を A, R, G, B の各成分値に分解するなどの，各種の画像処理で共通して用いられる機能は，`ImageProcessor` クラスで定義した．

プログラムのソースコードを付録として添付する．なお，各クラス及びメソッドにはドキュメンテーションコメント [4] を記述し，Javadoc ツールを使って API 仕様文書を生成できるようにした．

4　実験

4.1　実験方法

入力画像を作成したプログラムに入力し，出力された画像を観察した．また，実行に掛かった時間を測定した．入力画像として，画像 A, B, C の 3 つ（図 2）を使用した．入力画像の諸元を表 1 に示す．

前節で述べた NTSC 法のプログラム（P1 とする）と，輝度 Y の計算式を $Y = (R+G+B)/3$ に書き換えた単純平均法のプログラム（P2）の 2 つを使用する．

3 つの入力画像と 2 つのプログラムのすべての組み合わせについて，それぞれ 6 回続けて実行し，ディスクキャッシュの影響を避けるため，2 回目以降の 5 回の実行時間を測定した．実行時間の測定には time コマンドを使用した．

実験には iMac (3.2GHz Intel Core i3, 4GB RAM, Mac OS X 10.6.8) を使用した．また，コンパイル環境及び Java 実行環境として，Java SE Development Kit 7, Update 15 を使用した．

4.2　実験結果

得られた出力画像を図 2 に示す．また，プログラムと入力画像の各組み合わせにおける平均実行時間を表 2 に示す．表 2 の通り，P1, P2 とも，画像 A, B に対する実行時間は約 1.90 秒，画像 C に対する実行時間は約 1.95 秒であった．同じ入力画像に対する P1 と P2 の平均実行時間の差は 0.01 秒未満だった．

表 2　平均実行時間

	画像 A	画像 B	画像 C
P1 (NTSC)	1.900s	1.899s	1.950s
P2 (単純平均)	1.906s	1.903s	1.951s

5　考察

図 2 より，計算式 (1) を使ってモノクローム化を実現できることが確認できた．また，計算式 (1) の代わりに R, G, B 成分値の平均を用いた方法でもモノクローム化が行えることがわかった．平均実行時間はいずれも 2 秒未満であり，特に約 200 万画素の画像 C に対しても，画像 A, B に比べて約 50 ミリ秒しか増加しなかった．また，NTSC 法と単純平均法の実行時間の差は小さかった．これらのことから，NTSC 法及び単純平均法によるモノクローム化は，大きな画像に対しても現実的な時間で実行できると予想される．ただし，本実験では静止画像しか扱っておらず，動画像に対して現実的な時間で実行できるかどうかは不明である．

NTSC 法 (P1) と単純平均法 (P2) の出力画像を比較すると，画像 A 及び画像 C ではあまり大きな差は見られなかった．画像 B では，P1 の出力の方が，入力画像の黄色とオレンジ色の差がはっきり再現され，入力画像の印象により近いと思われる出力画像が得られた．このことから，少なくとも本実験の範囲では，NTSC 法は単純平均法より自然な出力画像が得られるが，黄色やオレンジ色の物が写っていない画像では差はほとんどないことがわかった．なお，両手法のうちどちらの出力画像が自然であるかは，被験者による評価実験を行うことなどによって測定できると考えられる．

謝辞

本実験課題は本学情報学群 1170998 高知太郎氏と共同で実施した．また本学情報学群の鏡野花子さんには，クラス設計に関して有益な助言をいただいた．これらの方々に深く感謝いたします．

参考文献

[1] Adobe Systems, PostScript® リファレンスマニュアル, 第 3 版, 第 7 章: レンダリング, アスキー, 2001.

[2] 安居院猛, 中嶋正之, 画像情報処理, 基礎情報工学シリーズ, 18, pp.47–48, 森北出版, 1991.

[3] S.S. Alhir, 入門 UML, 原隆文 (訳), 3 章: クラス図とオブジェクト図, オライリー・ジャパン, 2003.

[4] K. Arnold, J. Gosling, and D. Holmes, The Java Programming Language, Fourth Edition, Chapter 19: Documentation Comments, Addison-Wesley, 2006.

[5] International Telecommunication Union, “Information technology — Digital compression and coding of continuous-tone still images — Requirements

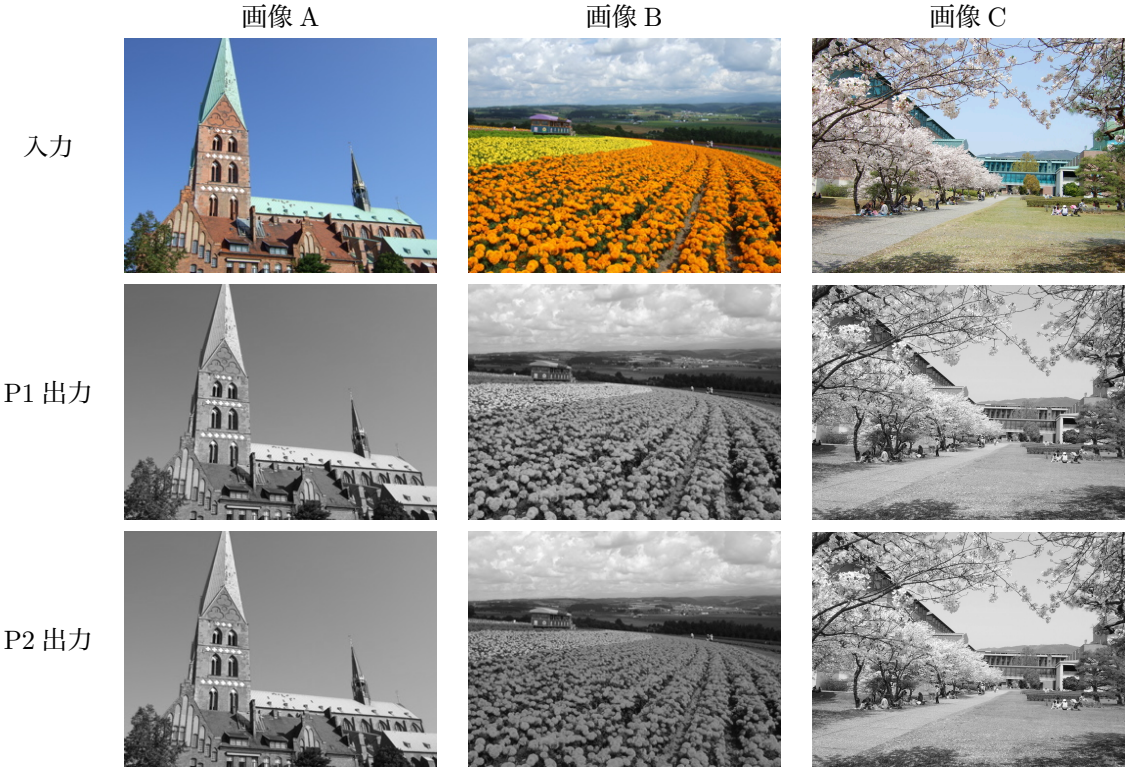

図 2　入力画像及び出力画像

and guidelines," Recommendation T.81 (Also published as ISO/IEC 10918-1), http://www.w3.org/Graphics/JPEG/itu-t81.pdf, 1992.

[6] 磯博, ディジタル画像処理入門: コンピュータによる画像処理の基礎知識, p.46, 産能大学出版部, 1996.

[7] Oracle Corporation, "Java SE 7 Image I/O 関連 API & 開発者ガイド," http://docs.oracle.com/javase/jp/7/technotes/guides/imageio/index.html, 2012.

Appendix

A Source code of the program

A.1 ImageMonochromatizer.java

```
/**
 * <code>ImageMonochromatizer</code> is a class
 * for obtaining a monochromatized copy of an input image.
 * You can apply this program to an image
 * through the command line interface of this class as follows.
 * <pre>
 * $ java ImageMonochromatizer inputfile outputfile
 * </pre>
 */
public class ImageMonochromatizer extends ImageColorModifier {
    /**
     * The command line interface.
     */
    public static void main(String[] arg) throws Exception {
        new ImageMonochromatizer().applyToFile(arg[0], arg[1]);
    }

    /**
     * This function is applied to each pixel for modifying its color.
     * In this case, the output is the luminance of the input color.
     * @param argb A color value in the ARGB format.
     * @return     The monochromatized color value.
     */
    public int modifyColor(int argb) {
        int[] s = decomposeARGB(argb);
        int r = s[1];
        int g = s[2];
        int b = s[3];
        int y = (int)(0.299 * r + 0.587 * g + 0.114 * b);
//        int y = (r + g + b) / 3;
        return composeARGB(s[0], y, y, y); // R = G = B = luminance
    }
}
```

A.2 ImageColorModifier.java

```
import java.awt.image.BufferedImage;

/**
 * <code>ImageColorModifier</code> is an abstract class
 * to represent a function that modifies the color of each pixel
 * of an input image.
 * This class implements the method
 * {@link ImageProcessor#apply apply(BufferedImage)}
 * using an abstract method {@link #modifyColor},
 * which will be implemented by a subclass.
 */
abstract public class ImageColorModifier extends ImageProcessor {
    /**
     * This function is applied to each pixel for modifying its color,
     * and is call-backed by {@link #apply apply(BufferedImage)}.
     * @param argb A color value in the ARGB format.
     * @return     A color value in the ARGB format.
     */
    abstract public int modifyColor(int argb);

    /**
     * This function creates a modified copy of an input image.
     * The output image is obtained by applying {@link #modifyColor} to
     * each pixel of the input image.
     */
    public BufferedImage apply(BufferedImage inImage) {
        int width = inImage.getWidth();
        int height = inImage.getHeight();
        BufferedImage outImage =
            new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                int argb = inImage.getRGB(x, y);
                argb = modifyColor(argb);
                outImage.setRGB(x, y, argb);
            }
        }
        return outImage;
    }
}
```

A.3 ImageProcessor.java

```
import java.io.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;

/**
 * <code>ImageProcessor</code> is an abstract class
 * to represent a function that creates a modified copy
 * of an input image.
 * This class defines an abstract method {@link #apply apply(BufferedImage)},
 * which is the core funtion of this class and will be
 * implemented by a subclass.
 * This class also provides a few utility methods to ease
 * the implementation of subclasses.
 */
abstract public class ImageProcessor {
    /**
     * Output file format such as "png" and "jpg".
     * Note that the extension of the given output file name is not
     * used for selecting the output file format.
     */
    final private static String outFormat = "png";

    /**
     * This function creates a modified copy of an input image.
     * This is the core function of the image processing that
     * this class provides.
     * @param inImage  An input image.
     * @return outImage The image obtained by applying the image
     *                  processing this class provides to
     *                  <code>inImage</code>.
     */
    abstract public BufferedImage apply(BufferedImage inImage);

    /**
     * A function for applying the method {@link #apply apply(BufferedImage)}
     * to an image file.
     * The resultant image is written into an output file with the given name.
     * Note that the output file format is decided by {@link #outFormat}
     * and not by the extension of the output file name.
     * @param inFile The name of an input file.
     * @param outFile The name of an output file.
     */
    public void applyToFile(String inFile, String outFile) throws IOException {
        applyToFile(new File(inFile), new File(outFile));
    }

    /**
     * A function for applying the method {@link #apply apply(BufferedImage)}
     * to an image file.
     * The resultant image is written into the given output file.
     * Note that the output file format is decided by {@link #outFormat}
     * and not by the extension of the output file name.
     * @param inFile An input file.
     * @param outFile The file to which the resultant image is written.
     */
    public void applyToFile(File inFile, File outFile) throws IOException {
        BufferedImage inImage = ImageIO.read(inFile);
        BufferedImage outImage = apply(inImage);
        ImageIO.write(outImage, outFormat, outFile);
    }

    /**
     * A utility function for obtaining a composite ARGB color value
     * from the component values.
     * @param a The alpha value.
     * @param r The red value.
     * @param g The green value.
     * @param b The blue value.
     * @return  The ARGB value.
     */
    public int composeARGB(int a, int r, int g, int b) {
        a &= 0xff;
        r &= 0xff;
        g &= 0xff;
        b &= 0xff;
        return (a << 24) | (r << 16) | (g << 8) | b;
    }

    /**
     * A utility function for obtaining the component color values
     * from a composite ARGB color value
     * @param argb The ARGB value.
     * @return     The array of component values, where the
     *             alpha, red, green, and blue values are arrayed
     *             in this order.
     */
    public int[] decomposeARGB(int argb) {
        return new int[] {
            (argb >> 24) & 0xff,
            (argb >> 16) & 0xff,
            (argb >> 8) & 0xff,
             argb       & 0xff,
        };
    }
}
```