# 情報科学類　知能情報メディア実験<br>T-6　コンピュータ画像処理の基礎

担当 ： 滝沢 穂高
jikkenT6@pr.cs.tsukuba.ac.jp

## 1　はじめに

インターネットやデジタルカメラなどの急速な普及に伴って，我々がディジタル画像に接する機会は大幅に増えた．普及初期の頃は，画像はコンピュータのディスプレイに表示して閲覧するだけのことが多かったが，最近では各種のツールによってそれらを編集処理することも可能になってきた．しかし，それらのツールでは単純で定型な処理しかできないことが多い．そこで「T-6 コンピュータ画像処理」では，デジタル画像の仕組みや，画像特徴の抽出，画像認識・理解などについて系統的に学び，濃淡画像，カラー画像や医用画像への適用方法を習得することを目指す．

本実験では，Oracle 社（旧 Sun Microsystems 社）のコンピュータ言語 Java を使ったプログラム例を紹介するが，C 言語など他のコンピュータ言語を使って実装を進めても構わない．また，本実験は Web ページに記載したスケジュールで実施するので，必ず確認すること．レポート提出についても Web ページに記載されているので確認すること．各課題では，**単にプログラムを作って結果画像を示すだけでなく，必ず結果に対する考察を行い，レポートに記述すること．**さらに，**本マニュスクリプトに記載されている以上の事を書籍等で調べ，プログラムを改良するなど，各自工夫することを推奨する．**また，**各課題において，▽ 印の付いた課題は発展的な内容で，少し難しいので，実施しなくても良いことにします．時間的に余裕のある者だけ実施するということにします．**

(a) 原画像.

(b) 拡大画像（中央の塔の最上部）.

図 1: デジタル画像の例.

### 1.1　デジタル画像の構造

デジタル画像の例を図 1 に示す．同図 (a) が原画像で，同図 (b) がその一部分 (中央の塔の最上部) の拡大画像である．見て分かるようにデジタル画像は,「画素 (ピクセル)」という一様な明るさをもつ小四角形領域の行列集合として構成されている．デジタル画像処理は，この行列に対する演算として解釈することができる．

図 2 に $7 \times 5$ 画素を持つ画像の例を示す．各画素には同図に示すインデックスでアクセスすることができる．画像左上が $(x, y) = (0, 0)$ で，右下が $(6, 4)$ となる．また各画素のビット数は，濃淡画像の場合は８ビット，カラー画像の場合は３２ビットが使われることが多い．濃淡画像では，各画素は（符号なしの場合）０から２５５の値をとり，０は真っ黒，２５５は真っ白に相当する．カラー画像では３２ビットを８ビットの４つ分に分割し，３つの８ビットで赤 (Red)，緑 (Green)，青 (Blue) の光の三原色（加法混色）を表現する．残りの８ビットはコンピュータグラフィックス等で透明度を表すのに使うことがあるが，本実験では使用しない．また，各画素を $(x, y) = (0, 0) \to (1, 0) \to (2, 0) \to \cdots \to (6, 0) \to (0, 1) \to (1, 1) \to \cdots \to (6, 1) \to (0, 2) \to \cdots \to (6, 4)$ の順番にアクセスすることをラスタスキャンという．

| (0,0) | (1,0) | (2,0) | (3,0) | (4,0) | (5,0) | (6,0) |
|---|---|---|---|---|---|---|
| (0,1) | (1,1) | (2,1) | … | … | | |
| (0,2) | (1,2) | … | … | | | |
| … | … | … | | | | |
| … | … | | | | | (6,4) |

図 2: 画像の構造.

### 1.2 PNM 画像フォーマット

Portable aNy Map (PNM) は，一般に普及している画像フォーマットの中で（恐らく）最も扱いやすい画像フォーマットで，iview.jar や gimp など多くの画像ビューワでの読み込み，書き出しがサポートされている．この画像フォーマットには表 1 に示す 6 種類のデータ形式があるが，本実験では “P2” と “P3” の二つだけを扱う．

表 1: PNM 画像フォーマットのデータ形式.

| 画像タイプ ＼ エンコーディング | テキスト | バイナリ |
|---|---|---|
| 2 値画像 | P1 | P4 |
| 濃淡画像 (8bit/pixel) | P2 | P5 |
| カラー画像 (32bit/pixel) | P3 | P6 |

P2，P3 形式はテキスト形式で emacs やメモ帳などのエディタで開いて，中身を確認することができる．これらの画像フォーマットと画像の例を以下に示す．

1. P2 形式の例

↓↓↓ P2.pgm： ここから ↓↓↓

```
P2
5 4
255
 10  20  30  40  50
 60  70  80  90 100
110 120 130 140 150
160 170 180 190 200
```

↑↑↑ P2.pgm： ここまで ↑↑↑

最初の行の “P2” はテキスト形式の濃淡画像であることを表す識別子で，次の行の “5” は画像の横 (X) サイズ，“4” は縦 (Y) サイズである．次の “255” は画素の最大値を表す．以下，最初の “10” は画像左上端の画素値で，“200” は右下端の画素値である．なお，“#” で始まる行はコメントとして無視される．Fig. 3 に画像の例を示す．ファイル拡張子は “.pgm” が使われる．

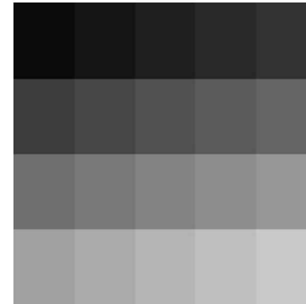


図 3: P2.pgm を画像ビューワで表示したもの.

2. P3 形式の例

↓↓↓ P3.ppm： ここから ↓↓↓

```
P3
3 3
255
255   0   0
  0 255   0
  0   0 255
255 255   0
  0 255 255
255   0 255
  0   0   0
127 127 127
255 255 255
```

↑↑↑ P3.ppm： ここまで ↑↑↑

最初の行の “P3” はテキスト形式のカラー画像であることを表す識別子で，次の “3”，“3”，“255” は同様に X サイズ，Y サイズ，最大画素値を表す．次の “255 0 0” は，画像の左上端の赤緑青 (RGB) の値を表している (BGR 等の場合もあるので注意が必要である)．Fig. 4 に画像の例を示す．ファイル拡張子は “.ppm” が使われる．

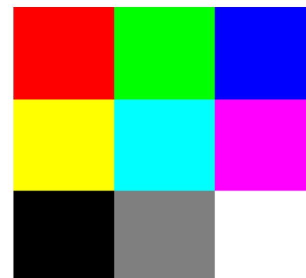


図 4: P3.ppm を画像ビューワで表示したもの.

## 1.3 画像処理の例：濃淡画像の２値化による領域抽出

濃淡画像の２値化とは，ある値（「しきい値」という）以上の画素値を持つ画素を１に，それ未満の画素を０に変換する処理である．目的によって１と０を逆にする場合や，１が画像ビューワで視認しにくいことから２５５（符号なし８ビット最大値）にする場合がある．入力画像の位置 $(x, y)$ における画素値を $inImg(x, y)$，出力画像の位置 $(x, y)$ における画素値を $outImg(x, y)$，しきい値を $th$ とすると，２値化処理は次式で表される：

$$outImg(x, y) = \begin{cases} 1 & inImg(x, y) \geq th \text{ のとき} \\ 0 & inImg(x, y) < th \text{ のとき} \end{cases} \quad (1)$$

次の 1.3.1 節と 1.3.2 節に，PGM 形式の濃淡画像ファイル $inImg$ を入力し，しきい値 “ $th$ ” で２値化し，PGM 形式の画像 $outImg$ を出力するアルゴリズムとプログラムの例を表す．また，1.3.3 節にそのプログラムのコンパイル，実行方法を示す.

### 1.3.1 しきい値による２値化処理のアルゴリズム

1. コマンドライン引数を解析し，変数 threshold にしきい値をセットする.
2. PGM 形式の濃淡画像を読み込む.
3. 入力した画像のサイズ $xsize$ (横) と $ysize$ (縦) を取得する.
4. 出力用の画像を生成する.
5. $y$ を 0 から $ysize - 1$ まで，$x$ を 0 から $xsize - 1$ までそれぞれ 1 ずつ変化させて，以下の処理を繰り返す.
   (a) 入力画像の位置 ($x$,$y$) における画素値を取得する.
   (b) その画素値がしきい値 (threshold) 以上のときだけ，出力画像の同じ位置の画素に１をセットする.
6. 出力用の画像を書き出す.

### 1.3.2 しきい値による２値化処理のプログラム

↓↓↓ prog0A.java : ここから ↓↓↓

```
import java.io.*;
import hpkg.fund.pnm.*;
public class prog0A
{
   public static void main(String[] args)
   {
      try
      {
         // コマンドライン引数を解析する.
         int threshold = 0;
         if(args.length != 3)
         {
            System.err.println("java prog0A しきい値 入力画像.pgm 出力画像.pgm");
            System.exit(0);
         }
         else
         {
            threshold = Integer.parseInt(args[0]);
         }

         // PGM 形式の濃淡画像を読み込む.
         HPnm inImg = new HPnm();
         inImg.readVoxels(args[1]);

         // 入力した画像のサイズ xsize (横) と ysize (縦) を取得する.
         int ysize = inImg.ysize();
         int xsize = inImg.xsize();

         // 出力用の画像を生成する. 画像サイズは (xsize,ysize) で,
         // 画素サイズは 8 Bit Per Pixel.
         HPnm outImg = new HPnm(xsize, ysize, 8);

         // y を 0 から ysize-1 まで, x を 0 から xsize-1 までそれぞれ 1 ずつ変化させて,
         // 以下の処理を繰り返す. なお, この順番に画素にアクセスすることを「ラスタスキャン」という.
         for(int y=0; y<ysize; y++) for(int x=0; x<xsize; x++)
         {
            // 入力画像の位置 (x,y) における画素値 (符号なし) を取得する.
            int value = inImg.getUnsignedValue(x, y);

            // その画素値が「しきい値」以上のときだけ, 出力画像の同じ位置の画素に 1 を
            // セットする (outImg の画素値のデフォルトは 0 であることに注意する).
            if(value >= threshold)
            {
               outImg.setValue(x, y, 255); // <== 8 bit の最大値にする.
            }
         }

         // 出力用の画像を書き出す.
         outImg.writeVoxels(args[2]);
      }
      catch(Exception e)
      {
         e.printStackTrace();
      }
   }
}
```

↑↑↑ prog0A.java : ここまで ↑↑↑

### 1.3.3 コンパイルと実行の方法

上記のプログラムを下記の方法でコンパイル，実行する．なお，Eclipse 等の統合環境を使ってプログラム開発を進めても構いません．

1. 学類の計算機システムには Java がインストールされているが，もし最新版等をインストールする場合は，Oracle の Java のホームページから JDK をダウンロードして，インストールする．
2. プログラムをエディッタで作成する．Java プログラムのファイル名はメイン（main）メソッドの存在するクラス名と同じにする必要がある（Java の規則）ので、上記のプログラムのファイル名は “prog0A.java” にすること．
3. クラスライブラリ “hpkg.jar” をカレントディレクトリにコピーする．このクラスライブラリには HPnm 等のクラスが定義されている．
4. 次のようにプログラムをコンパイルする．

   prompt% javac -cp .:hpkg.jar prog0A.java

   ただし，-cp オプションにおける区切り文字は，Linux と Mac ではコロン (:)，Windows ではセミコロン (;) を用いること．
5. コンパイルエラーがでたら 2 に戻り，必要な修正等を加える．
6. プログラムを実行する．上記の 2 値化のサンプルプログラムの場合，2 値化のしきい値を 88，入力画像を sample01.pgm, 出力画像を out.pgm とすると，コマンドラインに次のように入力すれば実行できる．区切り文字は上記と同様．

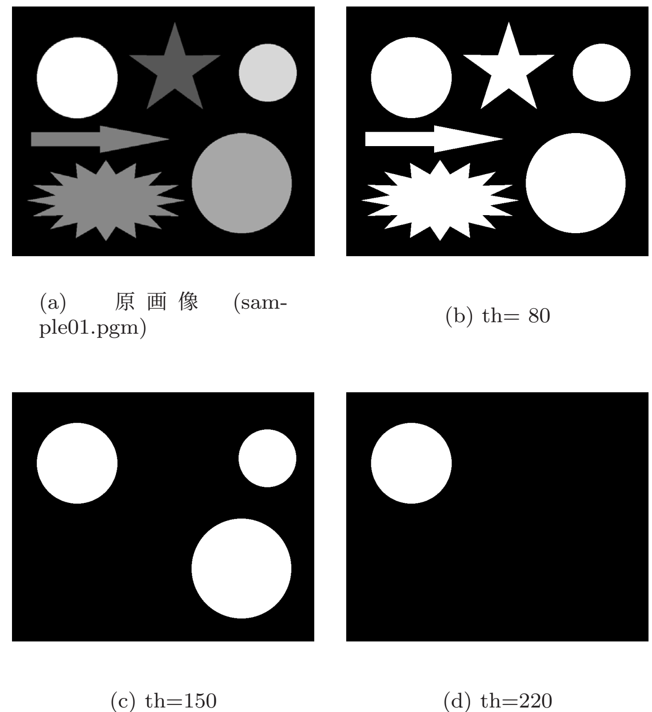
   prompt% java -cp .:hpkg.jar prog0A 88 sample01.pgm out.pgm
7. 出力画像を見るためには画像ビューワ “iview.jar” を使う．コマンドプロンプトからの起動は以下の通り．

   prompt% java -jar iview.jar out.pgm

   もしくは，iview.jar をダブルクリックして起動し，画像ファイル out.pgm をドラッグ＆ドロップする．

### 1.3.4 実行結果例

図 5(a) に原画像，(b)〜(c) にしきい値と 2 値化の結果を示す．



(a) 原画像 (sample01.pgm)　(b) th= 80　(c) th=150　(d) th=220

図 5: 原画像と 2 値化処理の結果．

### 1.3.5 練習問題

上記のプログラムファイル（ファイル名を “prog0A.java” にすること）を作成，コンパイル，実行し，結果を図 5 と比較せよ．

# 2 第１回：濃淡画像からの領域抽出と解析

## 2.1 判別基準法による濃淡画像の二値化

1.3 節の 2 値化プログラムでは，しきい値を手動で与え，0- 領域と 1-領域の 良さを人間が目視で判断していた．ここでは最適なしきい値を自動的に決定する手法として，判別基準法を紹介する．

判別基準法では，しきい値がある値 $t$ に設定された場合の領域分割の良さを「0-領域と 1-領域のクラス間分散」という値を使って評価する．0-領域にふられた画素の数を $N_0$，濃淡値の平均を $\mu_0$，1-領域にふられた画素の数を $N_1$，濃淡値の平均を $\mu_1$ とすると，クラス間分散 $\sigma_B^2$ は次式で定義される：

$$\sigma_B^2 = \frac{N_0}{N_0+N_1}\frac{N_1}{N_0+N_1}(\mu_0-\mu_1)^2. \tag{2}$$

判別基準法の主要部分のアルゴリズムを下記に示す．

### 2.1.1 判別基準法のアルゴリズム

1. 変数 $tMax$ と $\sigma_B^2 Max$ を用意し，$\sigma_B^2 Max = 0$ を代入しておく．

2. しきい値 $t$ を 0 から 255 まで (符号なし 8-bit の最小値から最大値まで)，1 ずつ変化させて，以下の処理を繰り返す．

   (a) しきい値 $t$ で入力画像を２値化する．

   (b) 1-画素の総数と画素値和を保存する変数 $n1$, $sum1$ を用意する．0-画素も同様に $n0$, $sum0$ を用意する．

   (c) $x$, $y$ を 1 ずつ変化させながら，下記の処理を繰り返す．

      i. (x,y) の画素値が 1 の場合，
      n1++ と sum1 += inImg(x,y) を計算．

      ii. その他の場合，
      n0++ と sum0 += inImg(x,y) を計算．

   (d) $\mu_1 = \mathrm{sum1}/\mathrm{n1}$ と $\mu_0 = \mathrm{sum0}/\mathrm{n0}$ を計算し，$\sigma_B^2$ を計算する．

   (e) もし $\sigma_B^2 \geq \sigma_B^2 Max$ ならば，$tMax = t$, $\sigma_B^2 Max = \sigma_B^2$ とする．

3. 最適なしきい値として，$tMax$ を採用し，このしきい値で入力画像を２値化して，出力する．

## 2.2 ２値画像の領域分割

２値画像内の１の画素を縦および横方向に連結させていき，領域を形成する処理を考える．例えば，図 6(a) の場合 1-画素は１１個あるが，縦横に互いに隣接している 1-画素を併合することによって，図 6(b) に示すように３つの領域を抽出することができる．このような処理を**ラベリング**という．ラベリング処理を施すことによって，各領域の大きさや重心位置，形状などの特徴量を計測することができる．

なお，連接関係には，縦および横方向の隣接関係だけを考える４近傍隣接と斜め方向も考慮にいれる８近傍連接があるが，ここでは４近傍隣接だけを扱う．

### 2.2.1 ラベリングのアルゴリズム

1. 図７に示す整数型配列のルックアップテーブル (LUT) を用意する．添字番号を値とするように初期化する．ここで，$L$ は十分大きな数である．また，新規のラベルを記憶する変数 $l_{new}$ を用意し，1 に初期化する．

| 0 | 1 | 1 | 0 | 0 |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |

(a) ２値画像

| 0 | 1 | 1 | 0 | 0 |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 3 | 0 |
| 2 | 0 | 3 | 3 | 3 |

(b) ラベリング結果

図 6: ラベリング処理．

| 添字 | 1 | 2 | 3 | 4 | 5 | ... | L |
|---|---|---|---|---|---|---|---|
| LUT | 1 | 2 | 3 | 4 | 5 | ... | L |

図 7: ルックアップテーブルの初期状態．

2. 入力した２値画像を図８の矢印に示すようにラスタスキャンし，最初（次）の 1-画素（$P$ とする）を見つける．最後までスキャンしたら６に進む．

| 0 | 1 | 1 | 0 | 0 |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |

図 8: ラスタスキャンし，最初（次）の 1-画素を見つける．

3. $P$ の上と左の画素にどちらもラベルが付与されていない場合（画素値が 0，もしくは画素が画像の外の状態を，ラベルが付与されていないと考える），$P$ に $l_{new}$ の値を付与し，２へ戻る．ただし，$P$ の座標を $(x_P, y_P)$ とした場合，$P$ の上の画素の座標は $(x_P, y_P - 1)$，左の座標は $(x_P - 1, y_P)$ であることに注意．新規ラベル $l_{new}$ を一つ増やす（$l_{new}++$ を実行する）．

4. 上と左の画素に同一のラベルが付与されている場合か，一方にのみラベルが付与されている場合，そのラベルを $P$ に付与し，２に戻る．

   (以上の２〜４までの処理で，図 9 の星印の直前まで処理が進む．)

| 0 | 1 | 1 | 0 | 0 |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 3 | 0 |
| 2 | 0 | 4 | 1 | 1 |

図 9: ラベリング処理の途中結果.

5. 上と左のラベルが異なる場合（図 9 の星印の画素がこの状況に相当する），小さい方のラベルを $l_{min}$，大きい方のラベルを $l_{max}$ とする.（図 9 の場合，$l_{min} = 3$，$l_{max} = 4$ となる．また，この時点で $l_{new} = 5$ である.）$P$ に $l_{min}$ を付与する.

   図 10 に示すように，ルックアップテーブルの $[1, l_{new})$ の区間を調べ，$LUT[l_{max}]$ と同じ値をもつ配列要素を $LUT[l_{min}]$ に変更する．２に戻る.

| 添字 | 1 | 2 | 3 | 4 | 5 | ... | L |
|---|---|---|---|---|---|---|---|
| LUT | 1 | 2 | 3 | **3** | 5 | ... | L |

更新範囲

図 10: ルックアップテーブルの更新.

6. 最後に，各画素の値を $l$ とした場合，$LUT(l)$ の値をその画素に上書きする．最終的に，図 6(b) に示すように３つの領域を抽出することできる.

図 5(b) にラベリング処理を施した結果を擬似カラー表示したものを図 11 に示す．なお，iview.jar において,「Ctrl-D」を押下することによって擬似カラー表示することができ，結果が正しいかどうかチェックすることができる.（必ず,「擬似カラー表示１」で確認すること．）

なお，ラベルの値は 256 を超えることがあるので，ラベル値を保存する画像には 32 ビット画像を用いる方が良い．32 ビット画像は
↓↓↓ ここから ↓↓↓
HPnm labelImg = new HPnm(xsize, ysize, 32);
↑↑↑ ここまで ↑↑↑
で生成することができる.

## 2.3 領域特徴量の計測

ラベリング処理によって抽出した各領域（例えば，図 11 の画像からは３つの領域が得られている）の特徴量を計測することによって，画像に写っている物体を認識する処理などに発展させることができる．ここでは代表的な領域特徴である以下のものを計測することを考える.

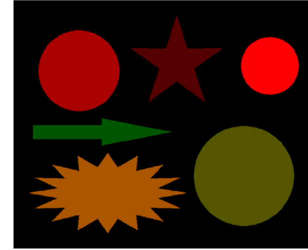


図 11: ラベリング処理の結果.「擬似カラー表示１」による表示.（各領域の色はプログラムの作り方等によって変わります．）

**領域の面積** 領域に属する画素の数である．例えば，図 6(b) の場合，1,2,3-領域の大きさはそれぞれ 5, 2, 4 である．画素数の計測は，判別基準法のアルゴリズムの 2.1.1 節を参考にすると良い.

**領域の重心** 領域に属する $x$, $y$ 座標の重心位置である.

**周囲長** ラベルの付与された画素の内，縦横方向に 0-画素（もしくは画像の外部）と隣接している画素の数である (斜め方向を考慮する場合もあるが，ここでは縦横方向にのみ注目する)．例えば，図 12 では，$*$ 印が付与されている画素が境界画素になる.

| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|
| 0 | 0 | 1* | 1* | 0 | 0 | 0 |
| 0 | 1* | 1 | 1 | 1* | 0 | 0 |
| 1* | 1 | 1 | 1 | 1 | 1* | 0 |
| 0 | 1* | 1 | 1 | 1* | 0 | 0 |
| 0 | 1* | 1 | 1 | 1* | 0 | 0 |
| 0 | 0 | 1* | 1* | 1* | 0 | 0 |

図 12: 領域境界の画素（* が付与されている画素）.

**円形度** 領域の "円らしさ" を表す指標である．ある領域の面積を $S$, 周囲長を $L$ とした時の円形度は

$$c = \frac{4\pi S}{L^2}, \tag{3}$$

と表される．$c$ は，領域が真円のときに最大値 1 をとり，円でなくなるほど小さな値をとる.（実際は量子化誤差により境界が凸凹になり，領域が真円になることはなく，最大値も 1 とならないことがある.）

## 2.4 課題

下記の課題をそれぞれ実施せよ.

1. 与えられた濃淡画像を判別基準法により 2 値化するプログラムを作成し，結果画像としきい値を示せ．
2. 作成した２値画像にラベル付けを行うプログラムを作成し，結果画像とラベルの数を示せ．
3. ▽ 2.2.1 節で示したアルゴリズムでは，結果画像のラベル番号に欠番が出ることがある．例えば，$1, 2, 5, 6, 8$ となり，欠番 $3, 4, 7$ が出ることがある．ラベルに欠番が出ないように改良せよ．
4. ▽ ラベル付けされた各領域の面積と重心，周囲長，円形度を計測し，出力するプログラムを作成し，これらの値を示せ．
5. 図 13 のサンプル濃淡画像に２値化とラベル付けを行うプログラムを適用し，結果を示せ．(ヒント： sample03.pgm の２値化では１と０を逆転させると良い．)
6. ▽ 円形度を用いて画像中から「丸い（ぽい）領域」だけを抽出し，その結果を示せ．
7. 自分で撮影した濃淡画像に上記のプログラムを適用し，結果を示せ．

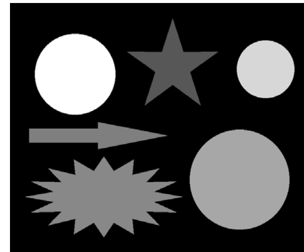
   （ヒント：サンプル画像のように背景が暗く，物体が白っぽい画像を撮影して実験に用いた方が良い.）一般のデジカメ等で撮影した画像は Gif, JPeg フォーマットで保存されることが多いが，iview.jar や gimp, xv などで PGM や PPM フォーマットに変換することができる．

## 3　第２回：カラー画像の処理

1.1 節で述べたようにカラー画像では１つのピクセルが RGB の３つの要素から構成されている．この RGB を他の座標系に変換する方法が色々と提案されており，目的によって使い分けている．今回は RGB の色変換について実験する．

### 3.1　ネガ変換

RGB の各色を反転させることによって，ネガフィルムのような画像を生成する処理である．この処理を行うプログラムを下記に示す．



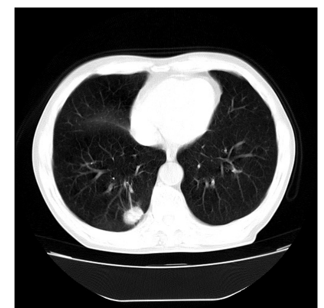
(a) sample01.pgm

(b) sample02.pgm

(c) sample03.pgm

(d) sample91.pgm

(e) sample92.pgm



(f) sample93.pgm

図 13: サンプル画像．**注意： sample91.pgm，sample92.pgm，sample93.pgm は医用画像ですので学外への持ち出しを禁止します．これらの画像を使った実験は必ず学内で行って下さい．**

### 3.1.1 ネガ変換のプログラム

↓↓↓ prog2A.java : ここから ↓↓↓

```
import java.io.*;
import hpkg.fund.pnm.*;

public class prog2A
{
   public static void main(String[] args)
   {
      try
      {
         // コマンドライン引数を解析する.
         if(args.length != 2)
         {
            System.err.println("java prog2A 入力画像.ppm 出力画像.ppm");
            System.exit(0);
         }

         HPnm inImg = new HPnm();
         inImg.readVoxels(args[0]);

         int ysize = inImg.ysize();
         int xsize = inImg.xsize();

         HPnm outImg = new HPnm(xsize, ysize, 32); // 画素サイズは 32 Bit Per Pixel.

         for(int y=0; y<ysize; y++) for(int x=0; x<xsize; x++)
         {
            int value = inImg.getUnsignedValue(x, y);
            int red   =  value         & 0xff; // 赤値の取得.
            int green = (value >>  8) & 0xff; // 緑値の取得.
            int blue  = (value >> 16) & 0xff; // 青値の取得.

            int red1   = 255 - red;   // 赤値を反転.
            int green1 = 255 - green; // 緑値を反転.
            int blue1  = 255 - blue;  // 青値を反転.

            // 色反転した値をマージする.
            int value1 = (0xff & red1) + (0xff00 & (green1 << 8)) + (0xff0000 & (blue1 << 16));

            outImg.setValue(x, y, value1);
         }

         outImg.writeVoxels(args[1]);
      }
      catch(Exception e)
      {
         e.printStackTrace();
      }
   }
}
```

↑↑↑ prog2A.java : ここまで ↑↑↑

### 3.1.2　実行結果例

図 14(a) に原画像，(b) に上記のプログラムを適用し，ネガ反転した画像を示す．

(a) 原画像 (sample11.ppm)

(b) ネガ反転画像

図 14: 原画像とネガ反転の結果．

### 3.1.3　練習問題

上記のプログラムファイル（ファイル名を “prog2A.java” にすること）を作成，コンパイル，実行し，結果を図 14 と比較せよ．

## 3.2　$RGB$ から $YCC$ への変換

$RGB$ から $Y$（輝度），$C_1$，$C_2$（色差信号）への変換は次式で表される：

$$Y = 0.299R + 0.587G + 0.114B, \tag{4}$$

$$C_1 = R - Y = 0.701R - 0.587G - 0.114B, \tag{5}$$

$$C_2 = B - Y = -0.299R - 0.587G + 0.886B. \tag{6}$$

輝度は，光が目に入った時に感じる強さを表している（RGB の平均である「明るさ」はどれだけの光を出すかを表している）．色差は，その色から他の二つの色の平均を差し引いた値で，0 を中心に $-\mathcal{C}_1$ から $+\mathcal{C}_1$ までの値（$\mathcal{C}_1$ は定数で，課題の場合は 179）をとるが，赤みが強ければ正の値になり，緑みと青みが強ければ負の値になる．

## 3.3　$YCC$ から $YSH$ への変換

$C_1$，$C_2$ から $S$（彩度），$H$（色相）への変換は次式で表される ($Y$ は同じ)：

$$S = \sqrt{C_1^2 + C_2^2}, \tag{7}$$

$$H = \arctan \frac{C_1}{C_2}. \tag{8}$$

アークタンジェントの計算には “$Math.atan2(double\ \ y,\ \ double\ \ x)$” を使うと良い（$y$ と $x$ の順番に注意）．

$YCC$ における $C_1$ と $C_2$ を極座標で表現したものが $S$ と $H$ である．$S$ は色の強さを表していて,「鮮やか」な色では大きく,「燻んだ」色では小さくなる．また，色相 $H$ は色の種類を角度を使って表現したものである．

## 3.4　課題

下記の課題をそれぞれ実施せよ．

1. カラー画像（例えば，sample11.ppm）を $RGB$ から $YCC$ へ変換するプログラムを作成し，結果画像を示せ．ただし，$Y$ 画像，$C_1$ 画像，$C_2$ 画像を別々の８ビット画像として出力し，示すこと．つまり，１枚の３２ビットカラー画像を入力し，３枚の８ビット濃淡画像を出力するプログラムになる．

   ここで $C_1$ と $C_2$ の値を setValue() によって８ビット画像にセットする際に注意が必要になる．$C_1$ は $(R, G, B) = (255, 0, 0)$ のとき最大値 179 になり，$(0, 255, 255)$ のとき最小値 $-179$ になるので，符号なし８ビットにそのまま値を保存すると桁溢れをおこす．そこで，以下の変換を行ってから保存すると良い：

$$C_1' = \frac{C_1 + 179}{179 + 179} \times 255. \tag{9}$$

   この変換によって，$(-179, 179)$ の範囲の値を $(0, 255)$ に収めることができる．$C_1$ の代わりに $C_1'$ を set-Value() することによって桁溢れを防ぐことができる．$C_2$ についても同様に変換する．

2. $YCC$ から $YSH$ へ変換するプログラムを作成し，結果画像を示せ．ただし，$S$ 画像，$H$ 画像を別々の８ビット画像として出力し，示すこと．

   ここで $C_1$ 画像と $C_2$ 画像から getUnsignedValue() によって取得した画素値を扱う際に注意が必要になる．$C_1$ の画像には実は式 (9) で変換した $C_1'$ の値が入力されているので，以下の逆変換を適用して $C_1$ の値を復元する必要がある：

$$C_1 = C_1' \times \frac{179+179}{255} - 179. \tag{10}$$

   なお，$S$ と $H$ も桁溢れを起こすので変換が必要になる．

3. $YSH$ から $YCC$ へ変換するプログラムを作成し，結果画像を示せ．ただし，$C_1$ 画像，$C_2$ 画像を別々の８ビット画像として出力し，示すこと．

4. ▽ $YCC$ からカラー画像 $RGB$ へ変換するプログラムを作成し，結果画像を示せ．ただし，$RGB$ 画像を３２ビット画像として出力し，示すこと．

   ここで，元の画像と同じになるか確認すること.（厳密には上記の変換や桁落ち等の影響により完全に同一の画像にはならないことに注意.）

5. ▽ 彩度 $S$ と色相 $H$ を変更してから，$YCC$ や $RGB$ へ戻すプログラムを作成し，結果を示せ．例えば，$S \leftarrow S+10$ としてから，$YCC$ や $RGB$ へ戻した場合，元のカラー画像と比べどのように変化するか調べよ．

6. 自分で撮影したカラー画像に上記のプログラムを適用し，結果を示せ．

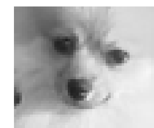
# 4　第３回：テンプレートマッチングによる類似領域の探索処理

## 4.1　テンプレートマッチング

**テンプレート**と呼ばれるある小さな画像が他の（より大きな）画像のどの部分に最も良く似ているかを求める処理をテンプレートマッチングという．例えば，図 15 の場合，同図 (a) の全体画像の中から同図 (b) のテンプレート画像に該当する部分（犬の顔）を見つける処理となる．

テンプレートマッチングの処理の手順を図 16 の全体画像（$9\times9$ の画像）とテンプレート画像（$3\times3$ の画像）の模式図を使って説明する．今，同図 (b) に最も良く似ている部分の位置を同図 (a) の全体画像から探索したいものとする．

(a) 全体画像 (sample22A.pgm).



(b) テンプレート画像 (sample22B0.pgm)

図 15: 原画像とテンプレート画像（拡大表示していますが，実際はもっと小さな画像です）.

最初，テンプレート画像が全体画像の左上端に重なるように（図 17 の緑色部分），テンプレート画像の中央の画素の位置を，全体画像の $(x, y) = (1, 1)$ にセットする．対応する 9 つの画素間の差の絶対値の和

$$\begin{aligned} & valueDiff \\ &= \sum_{yt=-1}^{1}\sum_{xt=-1}^{1} \mid img(x+xt, y+yt) \\ &\qquad - imgTMPL(xt+1, yt+1) \mid \end{aligned} \tag{11}$$

を計算する．ただし，$img(x, y)$ は全体画像，$imgTMPL(x, y)$ はテンプレート画像を表す．この画素値の差の絶対値の和は Sum of Absolute Differences（SAD）と呼ばれ，二つの画像の似ている度合いを表し，テンプレートマッチングで良く使われる（他に，画素値さの２乗和なども使われる）．次に，テンプレート画像をラスタスキャンの方向に一つずらして，$(x, y) = (2, 1)$ にし，同様に絶対値の和を求める．以下同様に，$(x, y) = (7, 7)$ まで計算する．最小の絶対値和を示す $(x, y)$ 座標値を最も良く似ている位置として出力する．

この処理を実施するプログラムの例と処理結果（座標位置を示すだけだが）を示す．

| (0,0) | (1,0) | (2,0) | (3,0) | (4,0) | (5,0) | … | … | (8,0) |
|---|---|---|---|---|---|---|---|---|
| (0,1) | (1,1) | (2,1) | (3,1) | (4,1) | (5,1) | … | | |
| (0,2) | (1,2) | (2,2) | (3,2) | (4,2) | (5,2) | … | | |
| (0,3) | (1,3) | (2,3) | (3,3) | (4,3) | (5,3) | … | | |
| … | … | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| (0,8) | … | | | | | | | (8,8) |

(a) 全体画像.

| (0,0) | (1,0) | (2,0) |
|---|---|---|
| (0,1) | (1,1) | (2,1) |
| (0,2) | (1,2) | (2,2) |

(b) テンプレート画像.

図 16: 原画像とテンプレート画像の模式図.

| (0,0) | (1,0) | (2,0) | (3,0) | (4,0) | (5,0) | … | … | (8,0) |
|---|---|---|---|---|---|---|---|---|
| (0,1) | (1,1) | (2,1) | (3,1) | (4,1) | (5,1) | … | | |
| (0,2) | (1,2) | (2,2) | (3,2) | (4,2) | (5,2) | … | | |
| (0,3) | (1,3) | (2,3) | (3,3) | (4,3) | (5,3) | … | | |
| … | … | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| (0,8) | … | | | | | | | (8,8) |

(a) テンプレートを (1,1) に合わせる.

| (0,0) | (1,0) | (2,0) | (3,0) | (4,0) | (5,0) | … | … | (8,0) |
|---|---|---|---|---|---|---|---|---|
| (0,1) | (1,1) | (2,1) | (3,1) | (4,1) | (5,1) | … | | |
| (0,2) | (1,2) | (2,2) | (3,2) | (4,2) | (5,2) | … | | |
| (0,3) | (1,3) | (2,3) | (3,3) | (4,3) | (5,3) | … | | |
| … | … | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| (0,8) | … | | | | | | | (8,8) |

(b) テンプレートを (2,1) に合わせる.

図 17: 全体画像にテンプレート画像をずらしながら合わせる.

### 4.1.1 テンプレートマッチングのプログラム

↓↓↓ prog3A.java : ここから ↓↓↓

```
import java.io.*;
import hpkg.fund.pnm.*;

public class prog3A
{
   public static void main(String[] args)
   {
      try
      {
         if(args.length != 2)
         {
            System.err.println("java prog3A 入力画像(全体).pgm 入力画像(テンプレート).pgm");
            System.exit(0);
         }

         HPnm inImg = new HPnm(); // 全体画像
         inImg.readVoxels(args[0]);

         HPnm inImgTMPL = new HPnm(); // テンプレート画像
         inImgTMPL.readVoxels(args[1]);

         int ysize = inImg.ysize();
         int xsize = inImg.xsize();

         int ysizeTMPL = inImgTMPL.ysize();
         int xsizeTMPL = inImgTMPL.xsize();

         int xMin=0, yMin=0;
         int valueDiffMin = Integer.MAX_VALUE;

         // 全体画像に部分画像を重ね，対応する画素の差の絶対値和を計算する.
         // 絶対値和が最小になる位置を対応する位置とする.
         for(int y=ysizeTMPL/2; y<ysize-ysizeTMPL/2; y++) // インデックスに注意.
         for(int x=xsizeTMPL/2; x<xsize-xsizeTMPL/2; x++)
         {
            int valueDiff = 0;
            for(int yt=-ysizeTMPL/2; yt<ysizeTMPL/2; yt++) // インデックスに注意.
            for(int xt=-xsizeTMPL/2; xt<xsizeTMPL/2; xt++)
            {
               int value     = inImg.getUnsignedValue(x+xt, y+yt);
               int valueTMPL = inImgTMPL.getUnsignedValue(xt+xsizeTMPL/2, yt+ysizeTMPL/2);
               valueDiff += Math.abs(value - valueTMPL); // 絶対値
            }
            if(valueDiff < valueDiffMin)
            {
               valueDiffMin = valueDiff;
               xMin = x;
               yMin = y;
            }
         }
         System.out.println("(x, y) = (" + xMin + ", " + yMin + ")");
      }
      catch(Exception e)
      {
         e.printStackTrace();
      }
   }
}
```

↑↑↑ prog3A.java : ここまで ↑↑↑

↓↓↓ 処理結果：ここから ↓↓↓
$(x, y) = (273, 561)$
↑↑↑ 処理結果：ここまで ↑↑↑
この座標は犬の顔の中央になり，狙い通りの結果になっています.

#### 4.1.2 練習問題

上記のプログラムファイル（ファイル名を “prog3A.java” にすること）を作成，コンパイル，実行し，結果を比較し，座標値を確認せよ.

### 4.2 課題

下記の課題をそれぞれ実施せよ.

1. 与えられた濃淡画像を任意の角度で回転するプログラムを作成せよ．ただし，画像を回転する下記のメソッドを使用して良い.（自作しても良い.）

   ↓↓↓ ここから ↓↓↓
   ○ 説明：画像 img の画像中心を中心に theta [rad] だけ回転させた画像を返す.
   ○ 形式
   ```
   import hpkg.tk.ImageFilter.*; (← 必ず import)
   HPnm HPixelFilter.rotate(HPnm img,
                            double theta);
   ```
   ○ 使用例
   ```
   HPnm rotatedImg =
       HPixelFilter.rotate(img, 30.0/180.0*Math.PI);
   ```
   ↑↑↑ ここまで ↑↑↑

2. 図 18 に示すような回転されたテンプレート画像でも対象物体を見つけ出せるように 4.1.1 節のプログラムを改良せよ．最も照合（マッチ）したときの角度を出力すること.（ヒント：4 重 for ループの外側に，回転角 theta を 0 度から 360 度（未満）まで回す for ループを新たに被せる.）

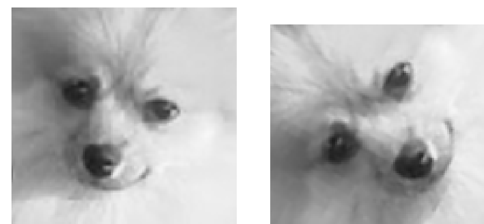
3. ▽ 図 19(a) に示すような拡大されたテンプレート画像や，拡大された後にさらに回転されたテンプレート画像でも対象物体を見つけ出せるようにプログラムを改良せよ.

## 5 第 4 回：医用画像の処理

医療診断用の撮像装置には，CT や MRI，PET，超音波診断装置など様々なものがあります．これらの医用撮像装置によって，癌や腫瘍の存在を診断したり，病状の進行状況を鑑別したりします．本章では，CT 画像から肺がん陰影（正確には結節と言う）を検出するプログラムを作成することを目指す.

(a) テンプレート画像 (sample22B1.pgm)　(b) テンプレート画像 (sample22B2.pgm)

図 18: 回転されたテンプレート画像.



(a) テンプレート画像 (sample22B3.pgm)　(b) テンプレート画像 (sample22B4.pgm)

図 19: 拡大されたテンプレート画像 (sample22B3) と，拡大・回転されたテンプレート画像 (sample22B4).

図 20 に胸部 X 線 CT 画像の例を示す．図中の丸印は肺がん陰影，矢印は肺血管陰影を示している．同図に示すように，一般に，肺がん陰影は丸い陰影を呈し，がんと間違いやすい血管陰影は細長い陰影を呈する．本章では，これらの陰影を識別する方法として，マセマティカル・モルフォロジーフィルタの一種である **Quoit** フィルタを学習する.（*quoit* とは英語で 輪投げという意味である.）

上記二種類の陰影の中，がんは図 21 左に示すような孤立性のピーク陰影でモデル化でき，血管は同図右に示すような尾根線状陰影でモデル化することができる．Quoit フィルタでは，この二種類の陰影を識別するために，「**リングフィルタ**」と「**ディスクフィルタ**」の二つのフィルタを用いる．がん陰影に二つのフィルタを上から落とすと，リングは裾野まで落ちるがディスクは頂上で止まる．一方，血管陰影では両方のフィルタが頂上で止まる．従って，二つのフィルタの出力の差（明度差）を検査すれば，二種類の陰影を識別することができる.

ある原画像に Quoit フィルタを施し，出力画像 を得る

図 20: CT 画像の例．丸印は肺がん，矢印は肺血管を指し示す．

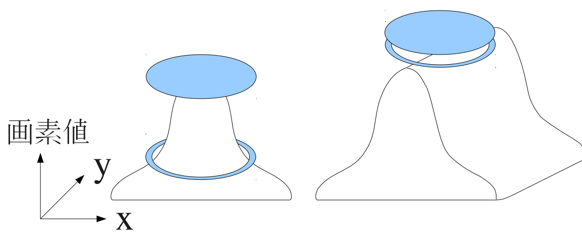



図 21: がん陰影（左）と血管陰影（右）の模式図．

演算を定式化すると以下のようになる：

$$q(x, y) = [f \oplus D](x, y) - [f \oplus R](x, y), \tag{12}$$

ただし，

$$D(x, y) = \begin{cases} 0 & (x^2 + y^2 < r_1^2 \text{ のとき}) \\ -\infty & (\text{その他の領域}) \end{cases} \tag{13}$$

はディスクフィルタを表し，

$$R(x, y) = \begin{cases} 0 & (r_2^2 \leq x^2 + y^2 \leq r_3^2 \text{ のとき}) \\ -\infty & (\text{その他の領域}) \end{cases} \tag{14}$$

はリングフィルタを表す．式 (12) において演算子「$\oplus$」は，モルフォロジの Dilation と言われる演算を表し，次式で定義される：

$$[f \oplus g](x, y) = \max_{\substack{(x+u, y+v) \in F \\ (u, v) \in G}} \{f(x+u, y+v) + g(u, v)\} \tag{15}$$

ただし，$F$ と $G$ はそれぞれ画像 $f(x, y)$ と $g(x, y)$ の定義域を表す．

## 5.1 Quoit フィルタのプログラム

↓↓↓ prog4A.java : ここから ↓↓↓

```
import java.io.*;
import hpkg.fund.voxels.*;
import hpkg.fund.pnm.*;

public class prog4A
{
   static final int R1 = 14;  /* Disk 外径 */
   static final int R2 = 13;  /* Ring 内径 (必ず R2<R3 にすること) */
   static final int R3 = 14;  /* Ring 外径 (必ず R2<R3 にすること) */
   static final int SMOOTHF_SIZE = 5;  /* filter size for smoothing */

   public static void main(String[] args)
   {
      try
      {
         if(args.length != 2)
         {
            System.err.println("java prog4A inImg.pgm outImg.pgm");
            System.exit(0);
         }

         HPnm inImg = new HPnm();
         inImg.readVoxels(args[0]); // 画像入力.

         HPnm inImgSM = smooth(inImg, SMOOTHF_SIZE); // 平滑化する.

         HPnm diskFilter = createFilter(R1,  0); // ディスクフィルタを作成する.
         HPnm ringFilter = createFilter(R3, R2); // リングフィルタを作成する.

         // 入力画像とディスクフィルタ, リングフィルとの Dilation 画像を求める.
         HPnm diskImg = dilation(inImgSM, diskFilter);
         HPnm ringImg = dilation(inImgSM, ringFilter);

         HPnm outImg = new HPnm(inImg.xsize(), inImg.ysize(), 8); // 出力用の画像.

         // 両画像の差分を求める.
         for(int y=0; y<diskImg.ysize(); y++) for(int x=0; x<diskImg.xsize(); x++)
         {
            int value = diskImg.getValue(x, y) - ringImg.getValue(x, y);
            outImg.setValue(x, y, value);
         }
         outImg.writeVoxels(args[1]); // 結果画像の出力.
      }
      catch(Exception ex)
      {
         ex.printStackTrace();
      }
   }

   /** 画像 img に fsize × fsize の平滑化フィルタを施し, 結果画像を返す. */
   private static HPnm smooth(HPnm img, int fsize)
      throws HVoxelMakingFailureException, HUnknownDepthSpecifiedException
   {
      HPnm smImg = new HPnm(img.xsize(), img.ysize(), 32); // 結果画像.
      for(int y=fsize/2; y<img.ysize()-fsize/2; y++) for(int x=fsize/2; x<img.xsize()-fsize/2; x++)
      {
         // 画素数と画素値和を計算する.
         int xd, yd, num = 0;
         double sum = 0;
         for(yd=-fsize/2; yd<=fsize/2; yd++) for(xd=-fsize/2; xd<=fsize/2; xd++)
         {
            sum += img.getUnsignedValue(x+xd, y+yd);
            num++;
```

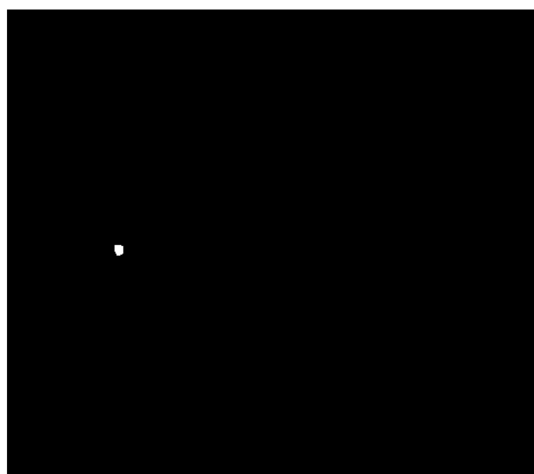
```
            }

            // 平均値を (x,y) の値にする.
            smImg.setValue(x, y, (int)Math.rint(sum/num));
        }
        return smImg;
    }

    /** 外径 radiusOutside, 内径 radiusInside の (リング) フィルタを作成す
        る. ただしフィルタでは画像中央が座標原点であると仮定する. */
    private static HPnm createFilter(int radiusOutside, int radiusInside)
        throws HVoxelMakingFailureException, HUnknownDepthSpecifiedException
    {
        HPnm filter = new HPnm(radiusOutside*2+1, radiusOutside*2+1, 32);
        for(int y=0; y<filter.ysize(); y++) for(int x=0; x<filter.xsize(); x++)
        {
            int u = x - filter.xsize()/2; // 画像中央が座標原点になるように座標変換する.
            int v = y - filter.ysize()/2;
            double r = Math.sqrt(u*u + v*v);

            // 原点との距離によって, ゼロか負の無限大の値を設定する.
            if((double)radiusInside <= r && r <= (double)radiusOutside)
            {
                filter.setValue(x, y, 0);
            }
            else
            {
                filter.setValue(x, y, Integer.MIN_VALUE);
            }
        }
        return filter;
    }

    /** f と g のモルフォロジ Dilation 演算を行い, 結果の画像を返す. 本来は構造要素
        g には対称画像を入力する必要があるが, 両フィルタとも原点対称なので省略する. */
    private static HPnm dilation(HPnm f, HPnm g)
        throws HVoxelMakingFailureException, HUnknownDepthSpecifiedException
    {
        HPnm diaOut = new HPnm(f.xsize(), f.ysize(), 32);

        // 各 (x,y) について f と g の和の最大値をもとめて, diaOut に設定する.
        for(int y=0; y<diaOut.ysize(); y++) for(int x=0; x<diaOut.xsize(); x++)
        {
            int maxValue = Integer.MIN_VALUE;
            for(int v=-g.ysize()/2; v<=g.ysize()/2; v++) for(int u=-g.xsize()/2; u<=g.xsize()/2; u++)
            {
                int xu = x + u;
                int yv = y + v;

                // (x+u,y+v) ∈ F のチェック ( (u,v) ∈ G は明らかに成立する).
                if(0<=xu && xu<f.xsize() && 0<=yv && yv<f.ysize())
                {
                    int value = f.getValue(xu, yv) + g.getValue(u+g.xsize()/2, v+g.ysize()/2);
                    maxValue = Math.max(maxValue, value);
                }
            }
            diaOut.setValue(x, y, maxValue);
        }
        return diaOut;
    }
}
```

↑↑↑ prog4A.java : ここまで ↑↑↑

## 5.2 実行結果例

図 22(a) に CT 画像の例，(b) に上記のプログラムを適用し，しきい値「３０」で２値化した結果を示す．

(a) CT 画像 (sample94.pgm)



(b) 肺がん検出結果

図 22: CT 画像と Quoit フィルタによる肺がん陰影の検出結果．左側中央下の丸い陰影が肺がんである．

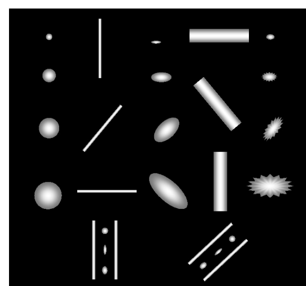
## 5.3 課題

下記の課題をそれぞれ実施せよ．

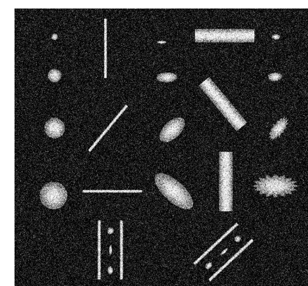
1. Quoit フィルタを実行するプログラムを作成し，図 23 の各画像に適用し，結果を示せ．プログラムは自作しても良いし，5.1 節のプログラムをそのまま使っても良い．

   図 23(a) 内の円や楕円はがん，四角は血管をモデル化している．また，下部の二つの図形は，血管に挟まれたがんを表している．このように挟まれたがんを正しく検出する事ができるかどうか実験せよ．

   また，図 23(b) はノイズが付加された画像で，このような画像でも正しくがんを検出できるかどうか実験せよ．



(a) sample05.pgm



(b) sample06.pgm

(c) sample94.pgm

(d) sample95.pgm

(e) sample96.pgm

図 23: サンプル画像．**注意： sample94.pgm，sample95.pgm，sample96.pgm は医用画像ですので学外への持ち出しを禁止します．これらの画像を使った実験は必ず学内で行って下さい．**

2. 5.1 節のプログラムを，図 23(c) に適用すると，その肺がんを正確に検出することができる．しかし，同図 (d) と (e) に適用すると，肺がんを検出することはできるが，それと同時に，多数の肺血管などを過剰検出してしまう．そこで，肺がんを逃さず，かつ過剰検出数を最小限に抑えるように，プログラム中のパラメー

タ $R1$, $R2$, $R3$, $SMOOTHF_SIZE$ を調整したり，アルゴリズムを工夫したりして実験し，その結果を示せ．また，同図 (a) と (b) に関しても，工夫して実験し，その結果を示せ．

## 6　第５回：自由課題

第１～５回までの内容を発展もしくは応用し，あるいは自分で他の画像処理技術について調査し，それを実装したプログラムを作成し，任意の画像（これまでの課題の画像を使っても良いし，自分で新しく撮り直しても良い）に適用せよ．

発展・応用あるいは調査した画像処理技術について説明し，プログラムに入力した画像と得られた結果画像を示せ．また，結果についての考察も行うこと．

## 参考文献


- 「コンピュータ画像処理」，田中秀行，オーム社.
- 「モルフォロジー」，小畑秀文，コロナ社.
- 他.


---