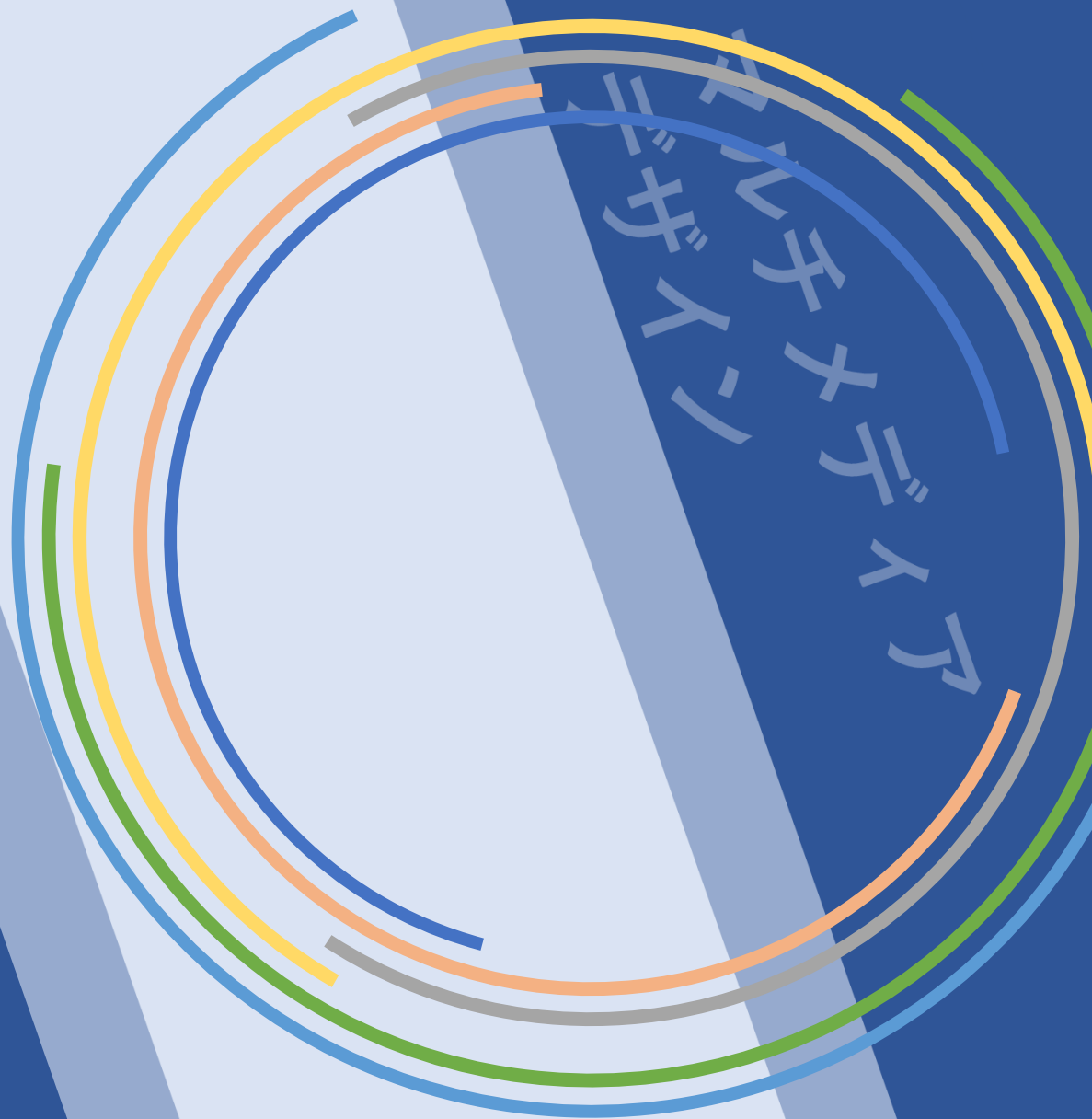


この授業のスライドはLMSで公開されます
実験の質問やレポートに関する質問などはSlackへ
<https://keio-st-multimedia.slack.com>



#7 各種変換 フィルタ

担当： 西 宏章

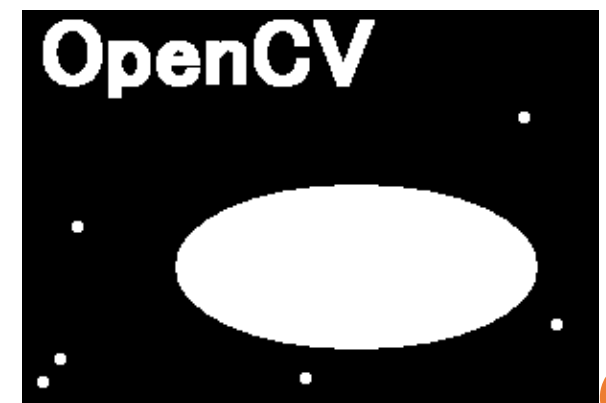
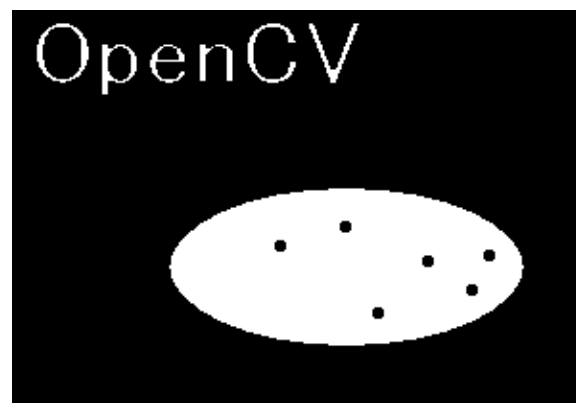
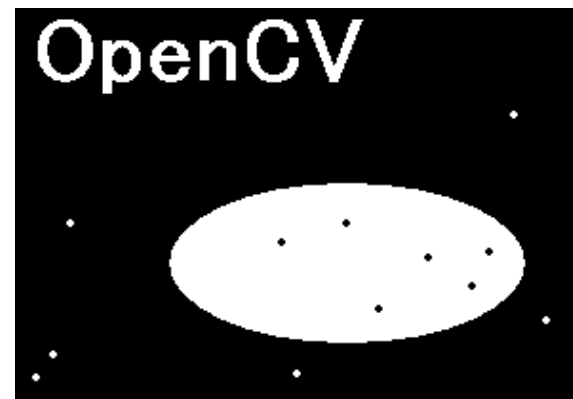


膨張・収縮処理(Dilation/Erosion)

23

- 孤立雑音を消去したり、2値化により抽出したい対象が消えてしまう場合に消えないように処理する場合に用いる

```
int pic[100][100], pic2[100][100], x, y, xx, yy, count;
for(y = 1; y < 99; y++){
    for(x = 1; x < 99; x++){
        count = 0;
        for(yy = -1; yy <= 1; yy++){
            for(xx = -1; xx <= 1; xx++){
                if(pic[x+xx][y+yy] == 1) count++;
            }
        }
        if(count > C) pic2[x][y] = 1;
        else pic2[x][y] = 0;
    }
}
```

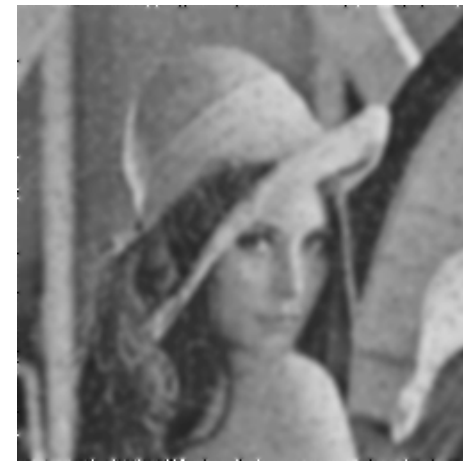


ノイズの除去

24

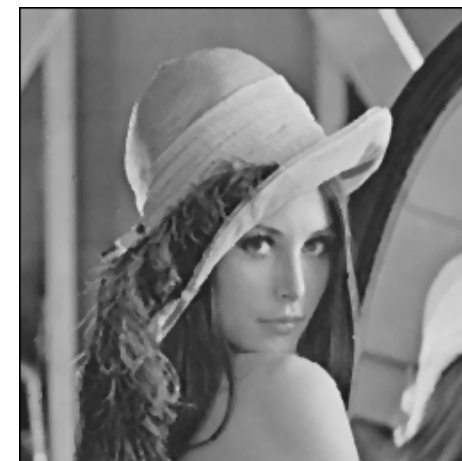
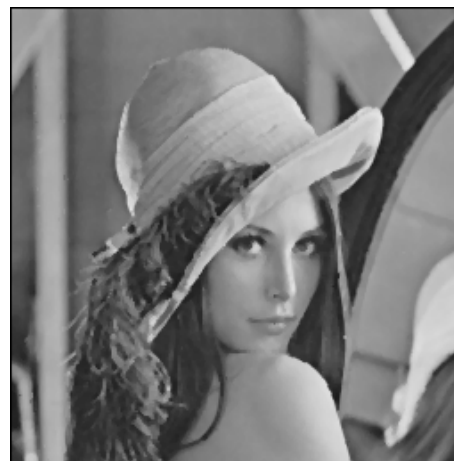
- 平均化

- 回数を重ねるとぼやける
- 計算コストは小さめ



- メディアンフィルタ

- 例えば3x3のマスの画素の照度を並び換え5番目(真中)の照度を採用する
- 回数を重ねても比較的劣化しにくい
- 計算コストは大きい



移動平均フィルタとガウシアンフィルタ

25

- 移動平均フィルタ

- 平均化フィルタ・平滑化フィルタ
- 単純に平均化

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

- ガウシアンフィルタ

- 近くほど影響を大きくする
- パスカルの三角形によりオペレータを生成できる

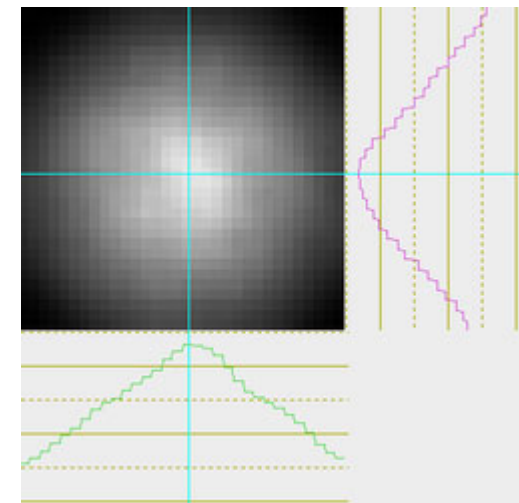
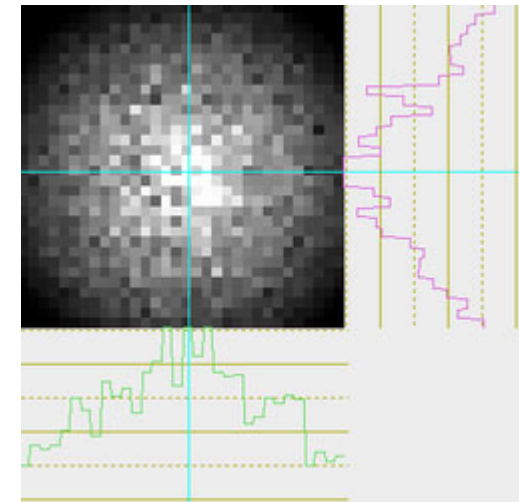
- $$f(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

3 x 3

1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

5 x 5

1/256	4/256	6/256	5/256	1/256
4/256	16/256	24/256	16/256	4/256
6/256	24/256	36/256	24/256	6/256
4/256	16/256	24/256	16/256	4/256
1/256	4/256	6/256	5/256	1/256

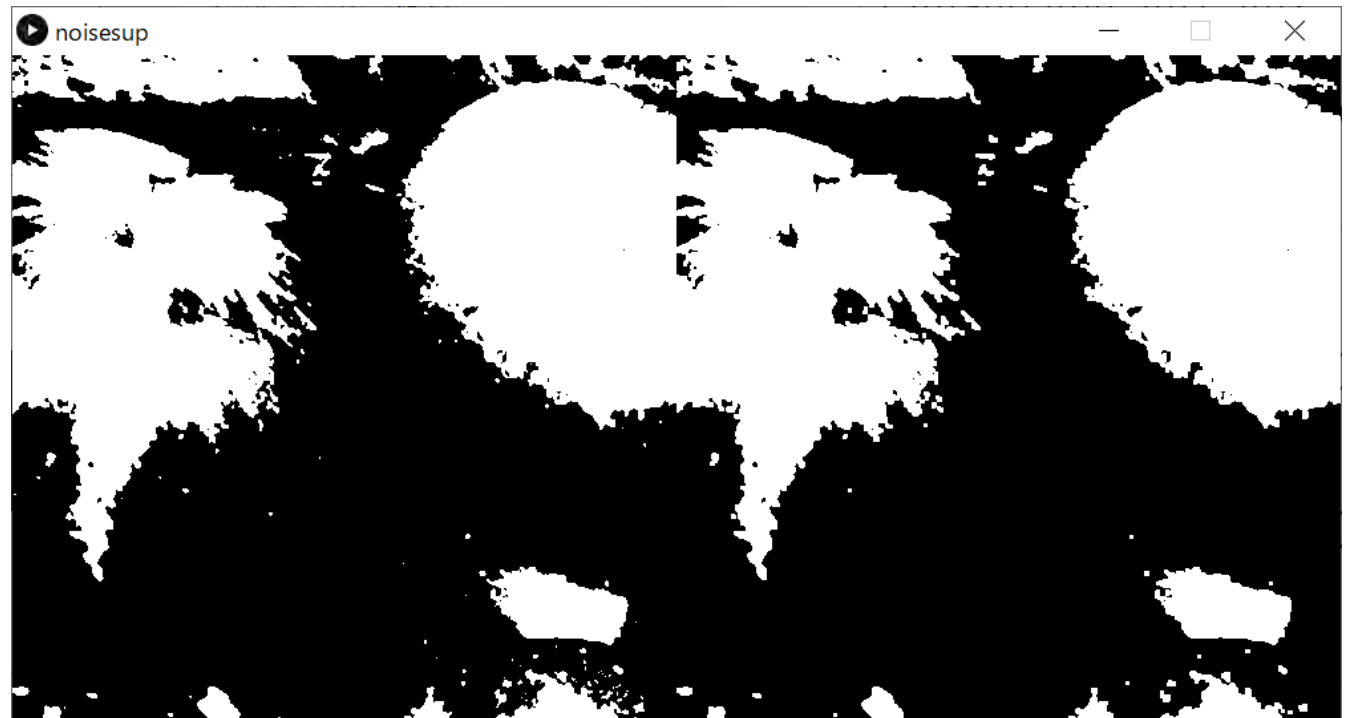


Processing:圧縮膨張フィルタ

26

- 圧縮フィルタと膨張フィルタを利用

```
// このコードは参考にはなりますが  
// 利用できません  
// 同じ内容の演習課題が出ます  
// これをコピペしてもダメです  
import gab.opencv.*;  
PImage img, im1, im2;  
OpenCV cv;  
img = loadImage("../apple.jpg");  
size(1000,500);  
cv = new OpenCV(this, img);  
cv.gray();  
cv.threshold(100);  
im1 = cv.getSnapshot();  
cv.erode();  
cv.dilate();  
im2 = cv.getSnapshot();  
image(im1, 0, 0);  
image(im2, 500, 0);
```



Processing: メディアンフィルタ

27

```
PImage img, imgr;  
img = loadImage("../apple.jpg");  
imgr = createImage(500, 500, RGB);  
size(1000,500);  
img.filter(GRAY);  
float a[] = new float[9];  
for(int y = 1; y < 500-1; y++){  
  for(int x = 1; x < 500-1; x++){  
    a[0] = red(img.get(x-1,y-1));  
    a[1] = red(img.get(x ,y-1));  
    a[2] = red(img.get(x+1,y-1));  
    a[3] = red(img.get(x-1,y ));  
    a[4] = red(img.get(x ,y ));  
    a[5] = red(img.get(x+1,y ));  
    a[6] = red(img.get(x-1,y+1));  
    a[7] = red(img.get(x ,y+1));  
    a[8] = red(img.get(x+1,y+1));  
    a = sort(a);  
    imgr.set(x, y, color(a[4]));  
  }  
}  
image(img, 0, 0); image(imgr, 500, 0);
```



Processing filter()による処理

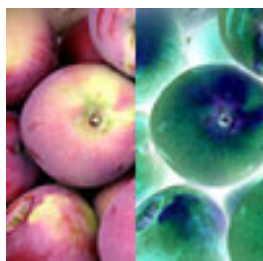
28



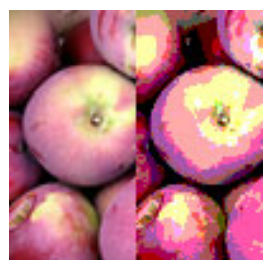
PlImage img; 自動二値化
img =
loadImage("apples.jpg"); size(300,300);等必要
img.filter(THRESHOLD, 0.3); イメージの読み込み
image(img, 0, 0); 閾値も指定できる
 **ここでは0.3と0.7の両方
 を比較として貼り付け**



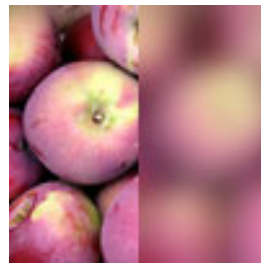
PlImage img; モノクロ化
img = (グレースケール)
loadImage("apples.jpg");
img.filter(GRAY);
image(img, 0, 0);



PlImage img; 反転 (ネガ)
img =
loadImage("apples.jpg"); 255-各画素値とする
img.filter(INVERT);
image(img, 0, 0);



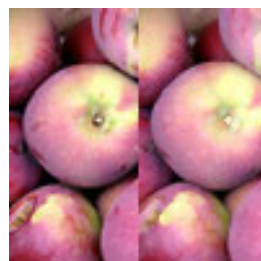
PlImage img; 量子化 (諧調縮約)
img =
loadImage("apples.jpg");
img.filter(POSTERIZE, 4);
image(img, 0, 0);



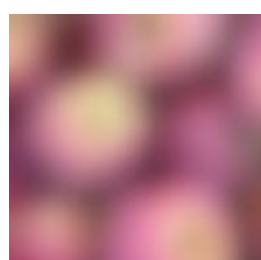
PlImage img; ブラー効果
img = (ぼかし・ガウシアン)
loadImage("apples.jpg");
img.filter(BLUR, 6);
image(img, 0, 0);



PlImage img; 収縮
img =
loadImage("apples.jpg");
img.filter(ERODE);
image(img, 0, 0);



PlImage img; 膨張
img =
loadImage("apples.jpg");
img.filter(DILATE);
image(img, 0, 0);



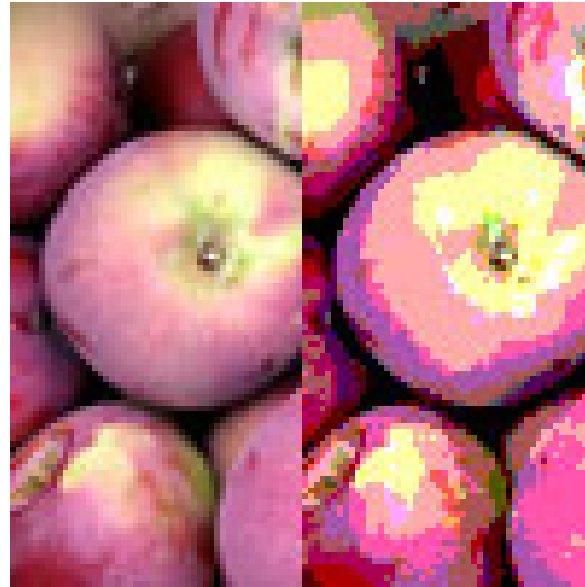
PShader blur; 詳しくは配布例参照
PlImage img; void draw(){
Setup(){ filter(blur)
size(100, 100, P2D); }
blur = loadShader("blur.glsl");
img = loadImage("apples.jpg");
image(img, 0, 0)
}



実際に実行してみよう

29

```
PImage img1, img2;
void setup() {
  img1 = loadImage("apples.jpg");
  img2 = loadImage("apples.jpg");
  img2.filter(POSTERIZE, 4);
  // このfilterの中を変えると様々なフィルタを適用
  // できる
}
void draw() {
  image(img1, 0, 0);
  image(img2, width/2, 0);
}
```





エッジ抽出

30

$$\Delta x = \frac{\partial f(x,y)}{\partial x} = f(i,j) - f(i-1,j)$$
$$\Delta y = \frac{\partial f(x,y)}{\partial y} = f(i,j) - f(i,j-1)$$

- 1次微分(Gradient)

0	0	0	0	0	0
0	1	-1	0	1	0
0	0	0	0	-1	0

Xオペレータ yオペレータ

- 1次微分(Sobel)

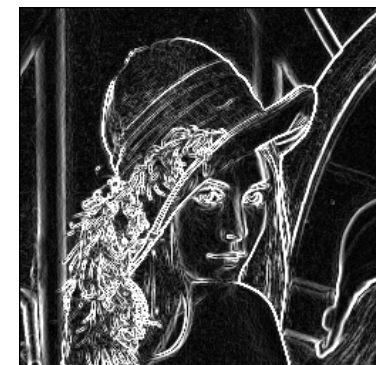
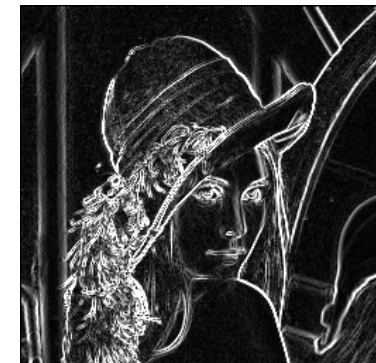
-1	0	1	-1	-2	-1
-2	0	2	0	0	0
-1	0	1	1	2	1

Xオペレータ yオペレータ

- 1次微分(Roberts)

0	0	0	0	0	0
0	1	0	0	1	0
0	0	-1	-1	0	0

Xオペレータ yオペレータ



エッジ抽出

31

- 2次微分(Laplacian)

0	-1	0
-1	4	-1
0	-1	0

オペレータ

- 2次微分(Sobel)

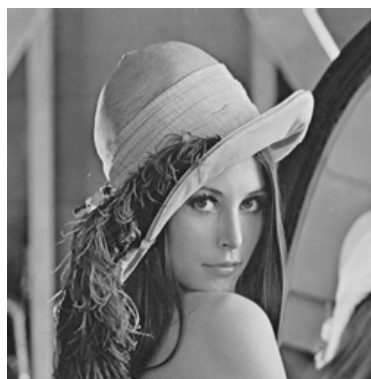
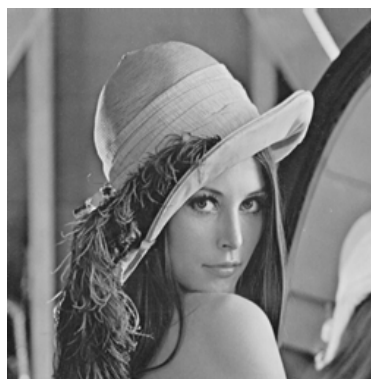
-1	-1	-1
-1	8	-1
-1	-1	-1

オペレータ

- 2次微分(Roberts)

1	-2	1
-2	4	-2
1	-2	1

オペレータ



Processing: Sobelによるエッジ抽出

32

```
PImage img, img_mask;  
size(1000, 500);  
img_mask = createImage(500, 500, RGB);  
img = loadImage("../apple.jpg");  
img.filter(GRAY);  
float gvs;  
float ghs;  
float g;  
for (int y = 1; y < 500-1; y++) {  
  for (int x = 1; x < 500-1; x++) {  
    ghs = - red(img.get(x-1, y-1)) - 2*red(img.get(x-1, y))  
      - red(img.get(x-1, y+1)) + red(img.get(x+1, y-1))  
      + 2*red(img.get(x+1, y)) + red(img.get(x+1, y+1));  
    gvs = - red(img.get(x-1, y-1)) - 2*red(img.get(x, y-1))  
      - red(img.get(x+1, y-1)) + red(img.get(x-1, y+1))  
      + 2*red(img.get(x, y+1)) + red(img.get(x+1, y+1));  
    g = sqrt(pow(ghs, 2) + pow(gvs, 2));  
    img_mask.set(x, y, color(abs(g)));  
  }  
}  
image(img, 0, 0);  
image(img_mask, 500, 0);
```



線分抽出

33

- ハフ(Hough)変換

- 直線抽出

横方向線分検出

-1	-1	-1
1	1	1
-1	-1	-1

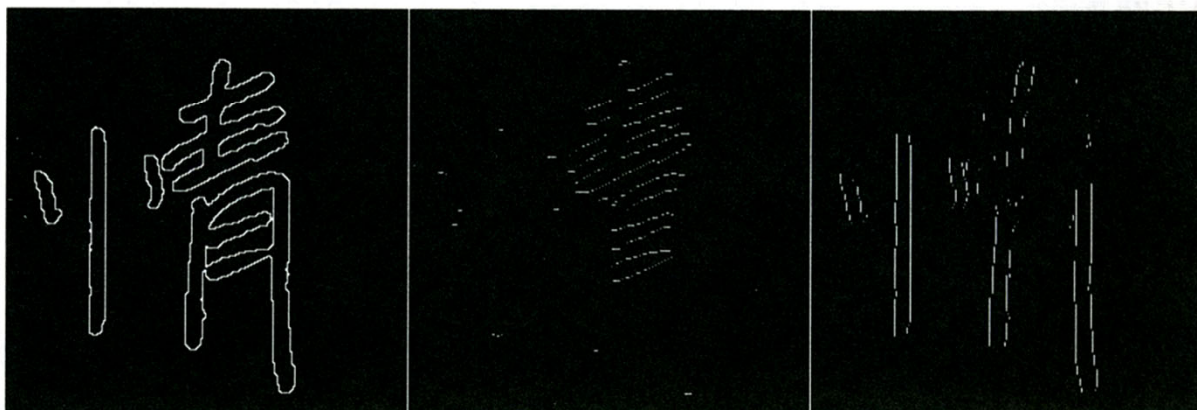
縦方向線分検出

-1	1	-1
-1	1	-1
-1	1	-1

原画像(256x256)

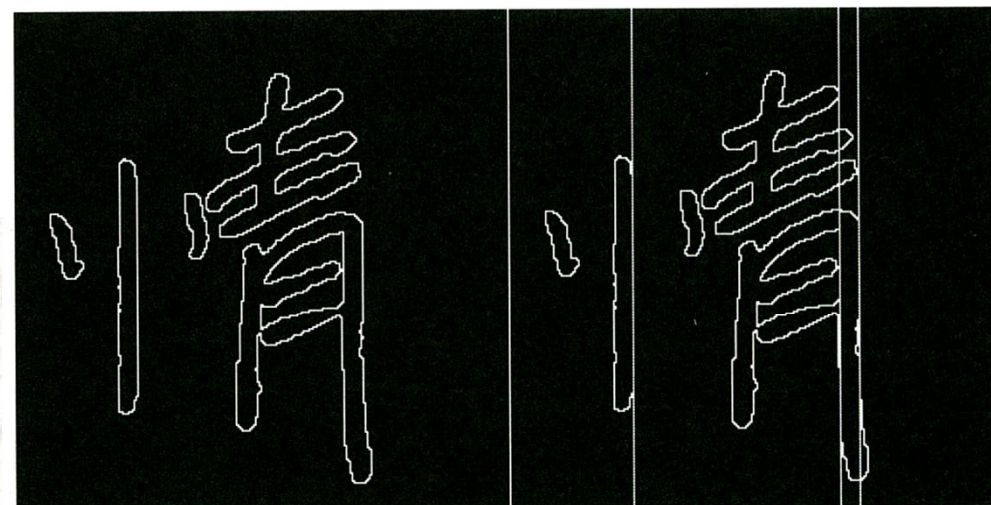
横方向線分検出

縦方向線分検出



原画像(256x256)

直線検出結果
(80dots以上の線分3本検出)

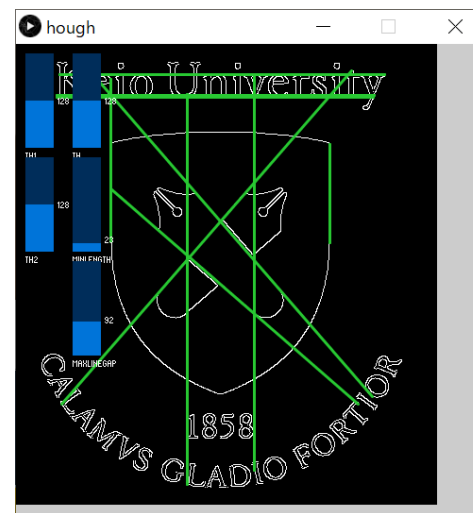


Processing: ハフ変換による線分抽出

34

```
import gab.opencv.*;
import controlP5.*;
ControlP5 cp5;
OpenCV cv;
PImage im;
ArrayList<Line> lines;
int th1 = 20, th2 = 75;
int th = 100, minLength = 30, maxLineGap = 20;
void setup() {
  size(500, 500);
  im = loadImage("../keio.jpg");
  cp5 = new ControlP5(this);
  cp5.addSlider("th1", 1, 255, 128, 10, 10, 30, 100);
  cp5.addSlider("th2", 1, 255, 128, 10, 120, 30, 100);
  cp5.addSlider("th", 1, 255, 128, 60, 10, 30, 100);
  cp5.addSlider("minLength", 1, 255, 128, 60, 120, 30, 100);
  cp5.addSlider("maxLineGap", 1, 255, 128, 60, 230, 30, 100);
}
```

```
void draw() {
  cv = new OpenCV(this, im);
  cv.findCannyEdges(th1, th2);
  lines = cv.findLines(th, minLength, maxLineGap);
  image(cv.getOutput(), 0, 0);
  strokeWeight(3);
  for (Line line : lines) {
    stroke(40, 200, 50);
    line(line.start.x, line.start.y, line.end.x, line.end.y);
  }
}
```






演習問題（5）

35

- Processingを用いて圧縮膨張フィルタを作成しなさい
 - 好きなイメージ（300x300から500x500）の画像を準備する
 - 300x300の場合はsize(600,300)とし変換前と変換後の画像を表示できるようにする
 - 利用する画像の大きさに合わせて適宜sizeを調整すること
 - Pimage img1, img2としてimg1(変換前)とimg2(変換後)を準備する
 - loadImageを用いて画像をimg1に読み込む
 - img1に対して、filterを用いて二値化する
 - THRESHOLDの値は、各自画像に合わせて調整してください
 - この2値化した値をもとに圧縮・膨張フィルタをかけてimg2を作る
 - img2をまっさらで準備するためcreateImageしなさい
 - サイズは準備した画像に合わせ、RGBを指定する
 - 自分も含めた9近傍に対して、1が3以上であれば自分を1に変換するような圧縮・膨張フィルタを構築し、img2に計算結果を書き込みなさい
 - 1がいくつ以上にするかは、各自で調整してもよい
 - img1とimg2を2つ張り付けなさい
 - image(img1, 0, 0); image(img2, 300, 0);などとする





注意事項

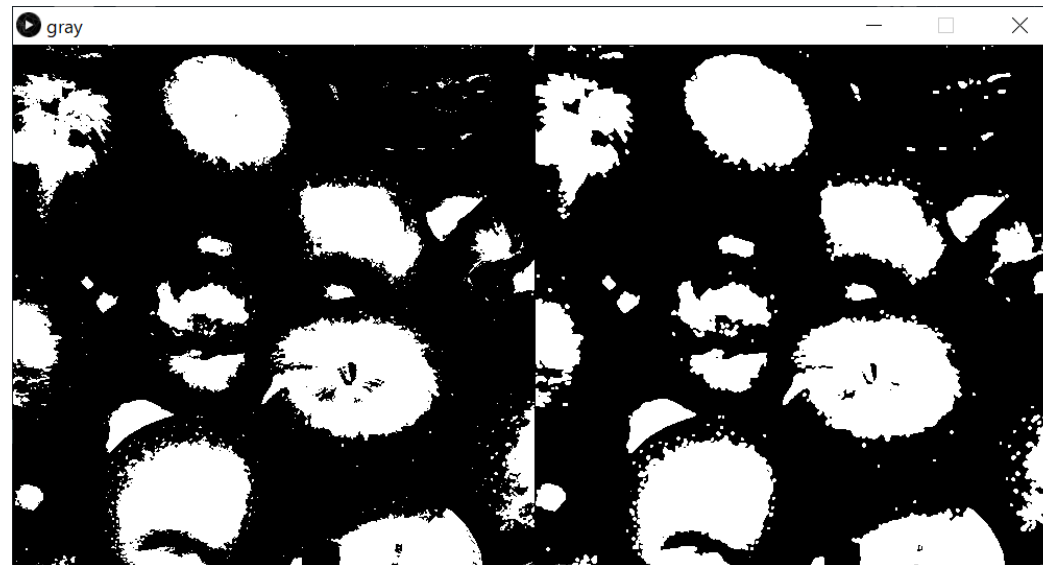
- これらの設問に対する回答を、Microsoft Wordファイルで作成
- LMSで提出すること
- A4で作成すること
 - なお、常識的な範囲でページ数を超過することは問題ありません
- **ソースコード、動作画面をキャプチャして貼り付けること**
 - キャプチャはキャプチャしたいWindowを選択してAlt+Print Screenでできます。そのまま、Wordファイルに張り付けることができます（+は押しながらの意味）
 - MacOSは、Control+Command+Shift+4とするとカーソルがカメラ型に変わります。特定のウインドウやメニューバーをクリックしてキャプチャ、画像はクリップボードに転送されます
- 最初にタイトルとして「演習問題（5）」と書き、名前と学籍番号を記載すること
このフォーマットに従っていないレポート答案は受け取らない
- 締め切りなど詳細はLMSを確認すること



演習問題のヒント

37

- サンプルコードを読まずに課題に取り組んで？？？になるのはルール違反
- 実行結果の例を示します
- 2値に変換しても、フルカラー画像ですのでRGBが存在します
 - 0と255の画素値を持ち、0と255だけで構成される画像になります
 - 0はよいが、画素値に1といった値を利用しないように
 - なのでredやcolorといったある画素を取り出す関数やRGBに拡大する命令が必要





Processing設計のコツ

38

- Processingでは、setupとdrawを揃えるのが基本です。
 - 間違いないのになぜか動作しない？と思う場合は基本の形に戻ってみてください
- void setup(){…}関数について
 - setup関数は前処理のみ記し、最終描画などは書かないのが基本ルールです
 - とはいえ、結構どのように書いても動いてしまいます。今回はそれでOKです
 - setup関数の中で、draw関数を呼び出すのもよい方法ではありません。draw関数は勝手に必要になったら呼び出されるので、明示的に呼び出さないでください
- void draw(){…}関数について
 - draw関数の中でdraw関数を呼び出してはいけません。draw関数は必要な際に自動的に呼び出されるため、draw関数から抜け出せないプログラムになり、メモリを使い尽くして落ちます
 - ある図形を描画するなどの決まった処理を行うルーチンを何度か呼びたい場合は、drawではなく「別の」関数を準備し、drawの中でその関数を呼び出してください





draw()と参考構造

39

- 講義とは関係ないですが
 - drawで動画(何かしら動くもの)を記述する場合、何度もdrawを呼び出さなければなりません。これを決めるのはframeRate()関数です
 - frameRate(5)とすると、秒5回呼び出されます
 - frameCount変数でプログラム実行開始から、何フレーム目かをカウントしています
 - 設定しない場合(デフォルト)は60fps(フレーム/秒)です
 - drawでsleepなどの待ちを表現する場合、フレームレート見ながら慎重に設定してください
- 参考構造
 - 次の基本構造を守って記述しましょう

```
void setup(){  
    直接描画に関係しない初期化に関する記述  
}  
void draw(){  
    描画に関する記述  
    routine(); // 何か決まった処理は別途記述してもよい  
}  
void routine(){  
    決められた処理  
}
```





Processingによる色の取得と描画方法

40

- PImageで変数を宣言します。PImage img; など
- 画像ファイルはloadImageを使います。img = loadImage("test.png"); など
- 自分で画像を作るときは、createImageします。img = createImage(500, 300, RGB); など。この場合は500x300の画像を新しく作ります
- PImageの画像から色を取ってくるときはgetを使います。
 - int red = red(img.get(x, y)); など。redは赤色成分を取得します。その他green、blueも同様に取得でき、カラーに対応します。授業では主にグレースケールを扱っていますので、red=green=blue ですからredだけ見れば十分。値は0から255しか取らないことに注意！
- PImageの画像の色を変える、設定するにはsetを使います。
 - img.set(x, y, color(p)); など。授業では主にグレースケールを扱っており、白は255、黒は0です。実際には、red=green=blueにして3つ同時に書き込まないといけませんが、これを実現するのがcolor()です。color(0)やcolor(255)として白黒二値にします
- 最後にimageをmapしてください。画面が表示されます。
 - size(500, 300);としてウィンドウを作ります。これは、setup()の中で記述するべきです
 - image(img, 0, 0);として、作った画像を貼り付けます





1次元配列pixelsを用いた画素操作方法

41

- 基本はget, setを使ってください
- その上で、PImageにはpixelsという画素を管理する1次元配列があります
 - この配列を用いても操作できますが、画像の横サイズを指定しなければならないため操作は厄介です。また、**値は0から255しか取らないことに注意！**
 - こちらは操作が複雑になるため、コードを直接掲載します
 - これらの例はグレースケールですが、カラーでも同様です
 - カラーの場合は3次元配列を準備し、red, gree, blueをそれぞれ保存します
 - get, setを用いる場合も、同様にカラーを扱うことができます

```
for (y = 1; y < YSIZE; y++) {  
  for (x = 1; x < XSIZE; x++) {  
    pic[x][y] = (int)red(img.pixels[x+y*XSIZE]);  
  }  
}
```

imgから2次元配列picにグレースケール
画素情報を書き込む例

```
for (y = 1; y < YSIZE; y++) {  
  for (x = 1; x < XSIZE; x++) {  
    img2.pixels[x+y*XSIZE] = color(pic2[x][y]);  
  }  
}
```

2次元配列picからimgにグレースケール
画素情報を書き込む例

