

プログラミング応用

信号処理技術

担当：福嶋 慶繁

更新履歴
2020年 福嶋

連絡先

□場所

– 20号館4階402号室

- 必ずメールしてから来てください

□メール

–fukushima “あつと”nitech.ac.jp

1. 概要
2. 画像処理全般の説明
3. ImageMagickを使った画像処理の説明
4. C言語によるテンプレートマッチングプログラムの説明
5. シェルスクリプトプログラミングの説明
6. 実際に動かす手順の実行マニュアル説明
7. 課題
8. 参考資料・ヒント

※ 2～6, 8は読んだほうが圧倒的に速いので動画での説明は省略。

1. 概要

“いらすとや”を探せ



ここ！



画像に埋め込まれている、いらすとやの画像を探す！

“いらすとや”を探せ

■ 内容

- C言語のプログラムと画像処理加工ソフトをシェルスクリプトで組み合わせて、いらすとやの画像を見つける

■ コンテスト

- 評価項目
 - ・ どれだけ見つけられたかの検出率
 - ・ どれだけ速くみつけられたかの実行速度
- 4週目にコンテストを開催

スケジュール

- 1週目：説明および演習
- 2週目：作業
- 3週目：作業
- ~~□ 4週目：作業およびコンテスト~~
- 4週目：作業
- 5週目：作業
 - 本来はチームで分担して進めるものを、個人で行うため、4~5週でいつもよりも作業時間を長く、コンテストを削除

学習内容・目的

- 小さなアプリケーションを一つ完成させる
- チームで分担してプログラミングを行う
- 画像処理を学ぶ
 - ImageMagickによるコマンド処理
 - C言語によるプログラミング
- シェルスクリプトプログラミングを学ぶ
- Makefile, コンパイルオプションを学ぶ
- プログラムの実行速度の概念を学ぶ

難易度

- レベル 1 : 正解はテンプレートの中から 1 つ
- レベル 2 : 画像にノイズが混入
- レベル 3 : 画像コントラスト変わる(0.5, 1.0, 1.5, 2.0)
- レベル 4 : テンプレートの背景透過する
- レベル 5 : テンプレートサイズが可変 (0.5, 1, 2)
- レベル 6 : テンプレートが回転 (0, 90, 180, 270)
- レベル 7 : 1 ~ 6 のどれかがシャッフル
- レベル 8 : ノイズが混入, コントラストが変わる, テンプレートが透過, テンプレートサイズが可変 (0.5, 1, 2) , 回転 (0, 90, 180, 270) .



一番簡単な場合。類似度は0になるはず



画像にノイズがのる。類似度は0にならない

レベル3

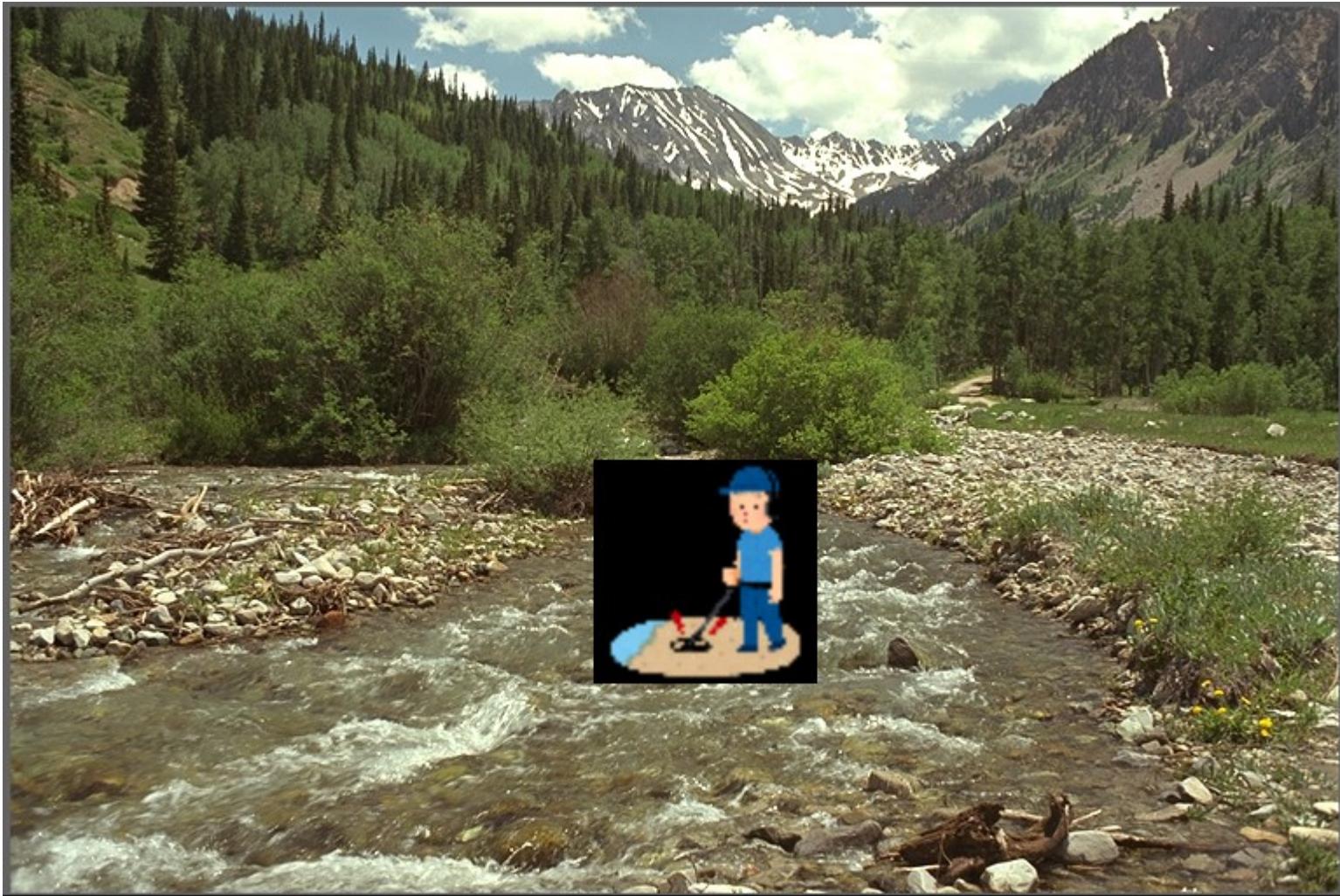
12



画像の明るさが変わる



テンプレートが背景に等価する



テンプレートが拡大する



テンプレートが回転する

レベル1～6がどれがはいっているかわからなく、まぜまぜ。



- 各レベル毎に精度と速度を評価。テストが7回走る。
- 正解データは当日まで非公開。現在テストしているものとは違う画像を使用。
 - どこかのレベルに合わせてプログラムが最適化されていることを期待
 - もしくはレベル7の汎用のデータすべてでうまく動くプログラムの作成
 - レベル8はおまけ

- 各レベルに必要な機能のプログラムを書く
- パラメータを記録する
- データを整理する
- 進捗を管理する
 - など

- レポート提出は最終のレポート1回.
- 各回の小課題は自習用で提出の必要なし.
- ~~班ごとで課題に取り組むが、最終レポートは個人で書くこと。~~
- 提出物
 - レポート (pdf)
 - プログラム(main.cだけ)
 - シェルスクリプト(run.shだけ)
 - レベルごとに異なるなら異なるものも適切に名前をつけて提出

レポートの注意事項

20

- 表紙に学籍番号，名前，**班の番号**を書くこと
- 提出形式は**pdf**. (tex, wordなどエディタは何で書いててもよいがpdfで出力)
- レポートは半角で，．打たないこと. (読みづらい)
- 生の出力結果の羅列はレポートにコピーしないこと.
 - シェルスクリプト等を工夫して，その羅列を強調したい場合は除く.
 - 単純にこうなりましたの結果を示すなら，表やグラフにしてください.
 - あからさまにスクリプト出力結果でページ数を稼ぐようなものは，読みづらいだけなので減点.
- ソースコードはレポートにコピーしないこと.
 - あからさまにコードのコピペでページ数を稼ぐようなものは，読みづらいだけなので減点.
 - ただし，**工夫した点をわかりやすく説明するためにソースコードを貼ることは推奨する**. その場合は，**該当部分だけを貼ること**.
- レベルごとの検出率と速度の表を書くこと.
 - 速度はtimeコマンドの**realの結果**だけでよい.
- 自分のやったところだけではなく班全体でやったところのレポートを書くこと.
- 各レベルごとの対処を書くこと.
- 共通するところは共通部分を書くこと.

- 難易度は順番にやる必要はありません。
- サンプルコードとやり方は提供していますが、全くサンプルコードを使わなくても構いません。解決の手段は問いません。
 - プログラミング言語も問いません。
 - ※できる人を召喚するは禁止

2. 画像処理

□背景

□画像とは

- 標本化

- 量子化

- 受光素子

- 色

□ 画像処理 (Image Processing)

□ 画像を加工して、人間の目やコンピュータにとって都合の良いように変換する処理

- 写真編集

- 画像圧縮

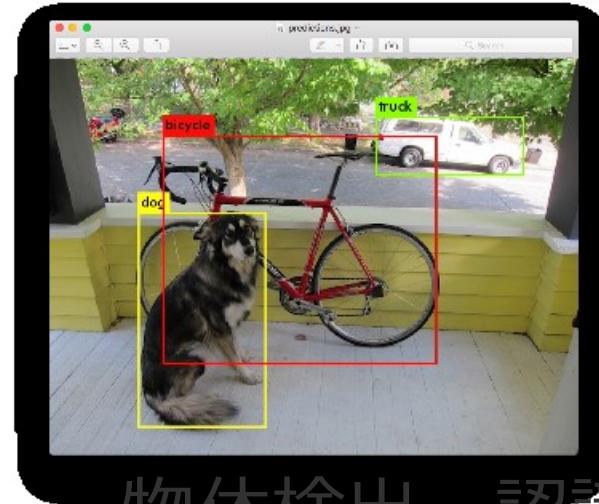
- 画像検出・認識

- 3次元復元

「画像処理には様々な応用先がある」



写真加工



物体検出・認識



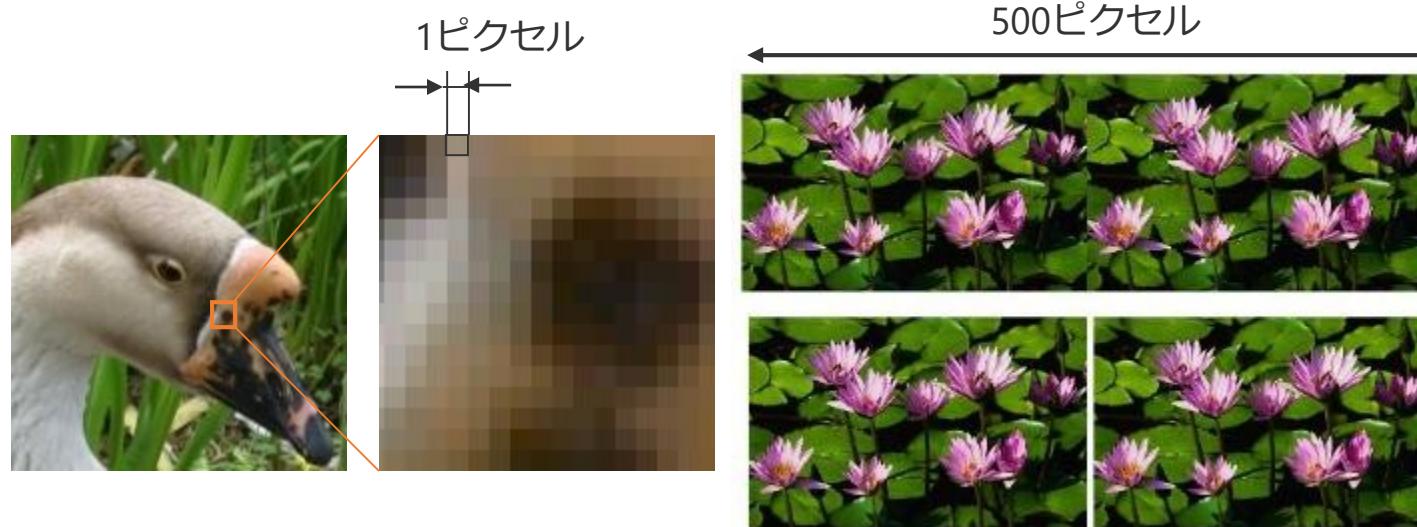
VR・AR



自動運転

- スマホ、監視カメラが増加
- Web上のデータも画像、動画がほとんど
- これからのアプリケーション
 - 自動運転
 - ロボット
 - AR・VR
- 画像を効率的に処理することは重要

デジタル画像とは

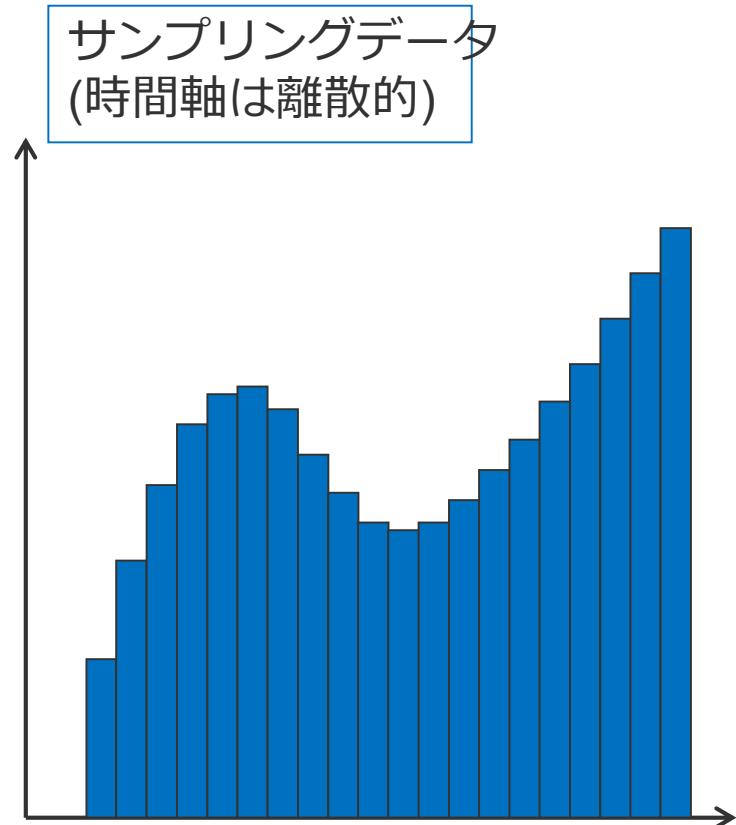
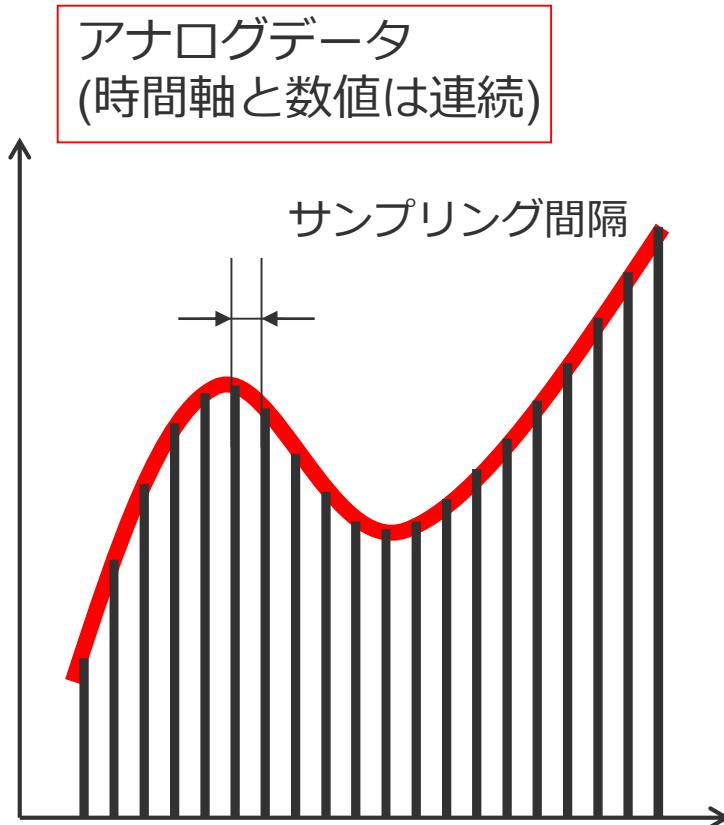


250ピクセルだと 2枚横並びにできます。

ピクセルとは、デジタル画像を構成する単位である、色のついた「■」のことです。1677万色のうちの1色を選べます。 ($R \times G \times B = 256 \times 256 \times 256$ 、8bit (256色))

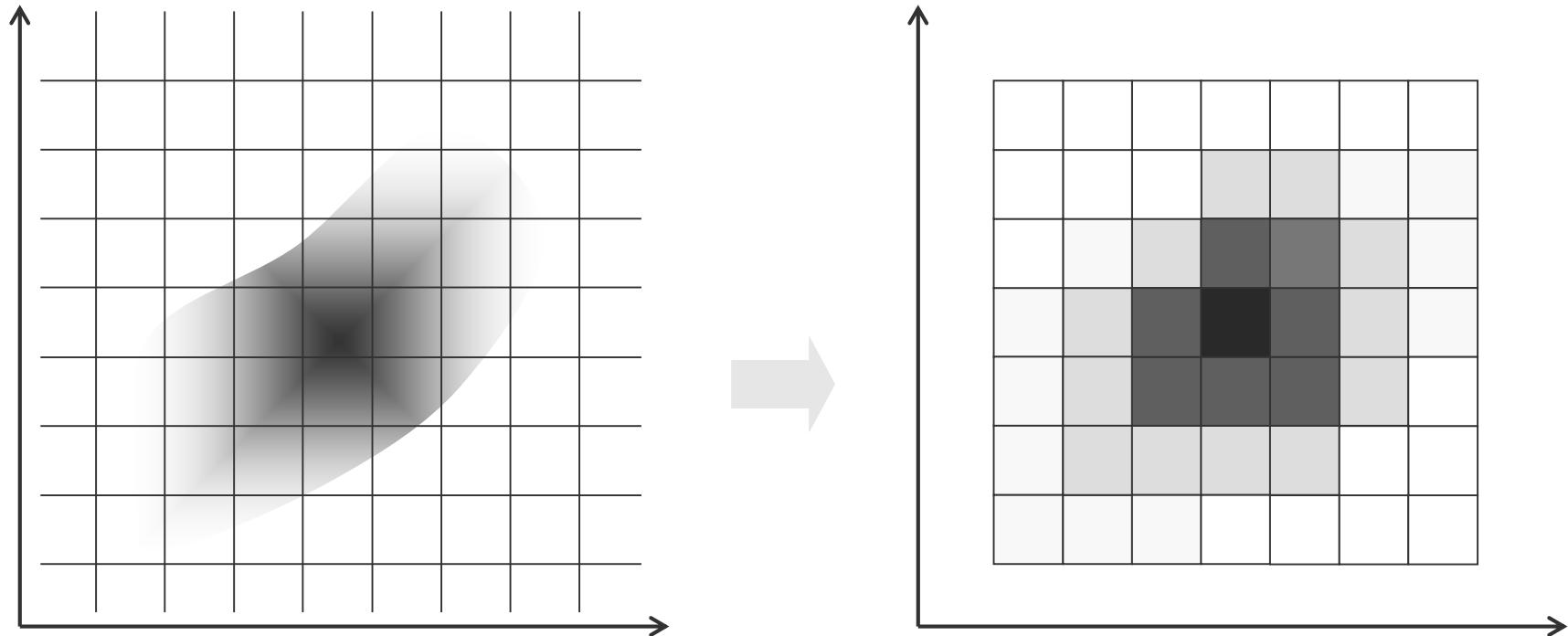
標本化

アナログデータを離散的に領域分割



標本化（続き）

□ 2-D デジタル画像の場合

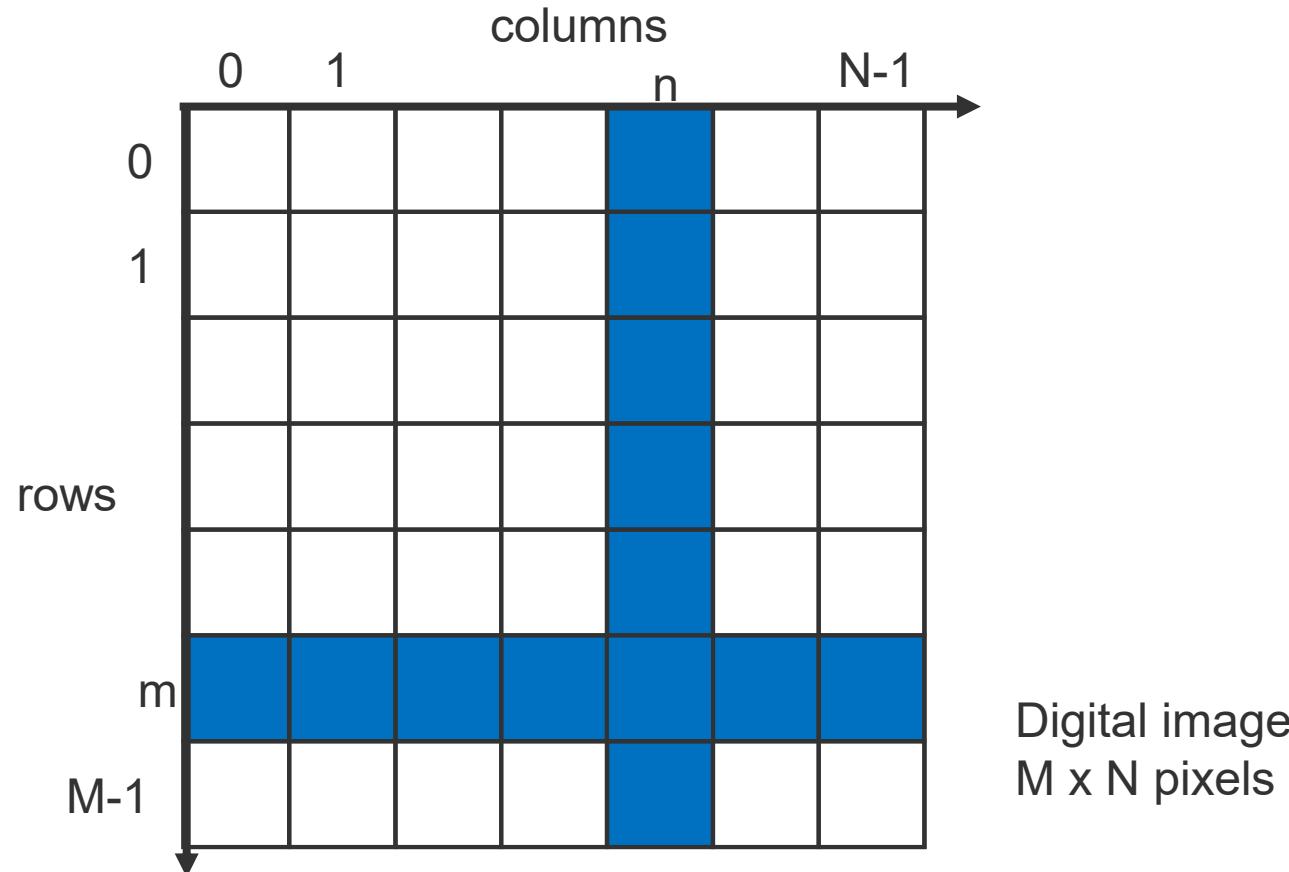


サンプリング間隔によって画像解像度が決まる

画素／ピクセル (Pixel)

□ 2-Dデジタル画像の単位

□ 空間分割



空間的標本化（解像度）



40 x 30
pixels



160 x 120
pixels

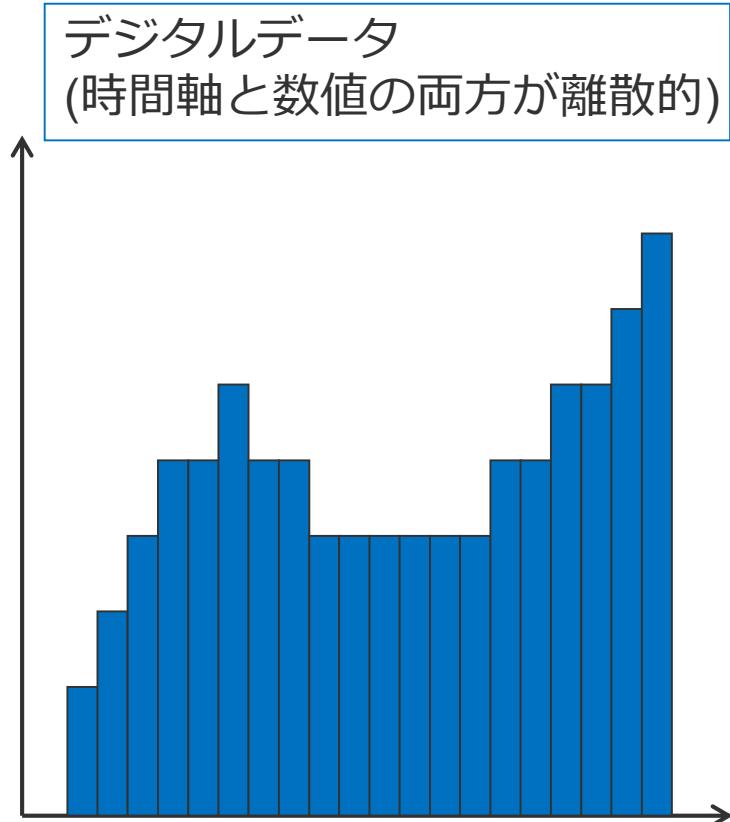
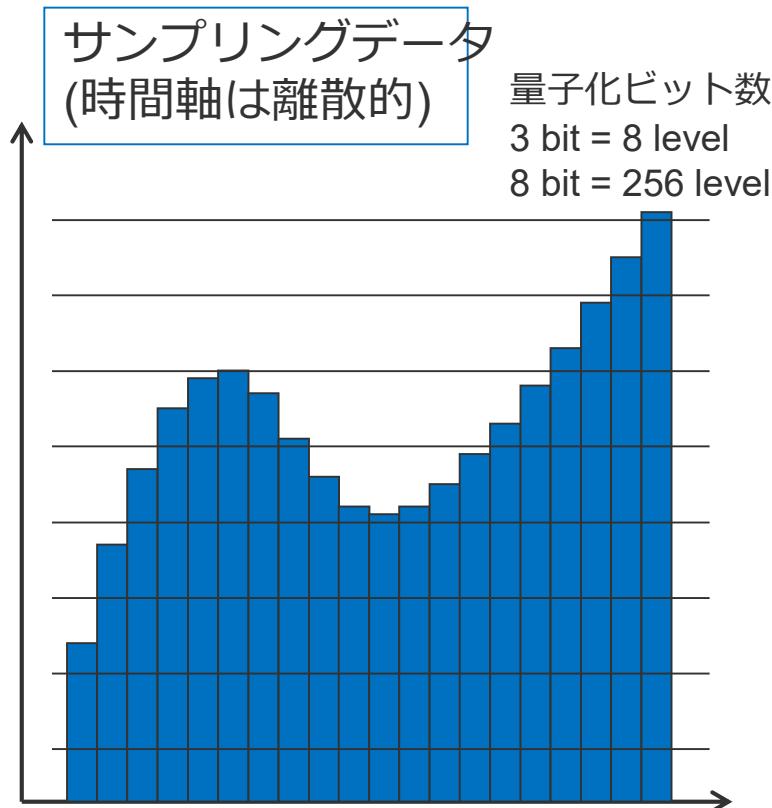


80 x 60
pixels

320 x 240
pixels

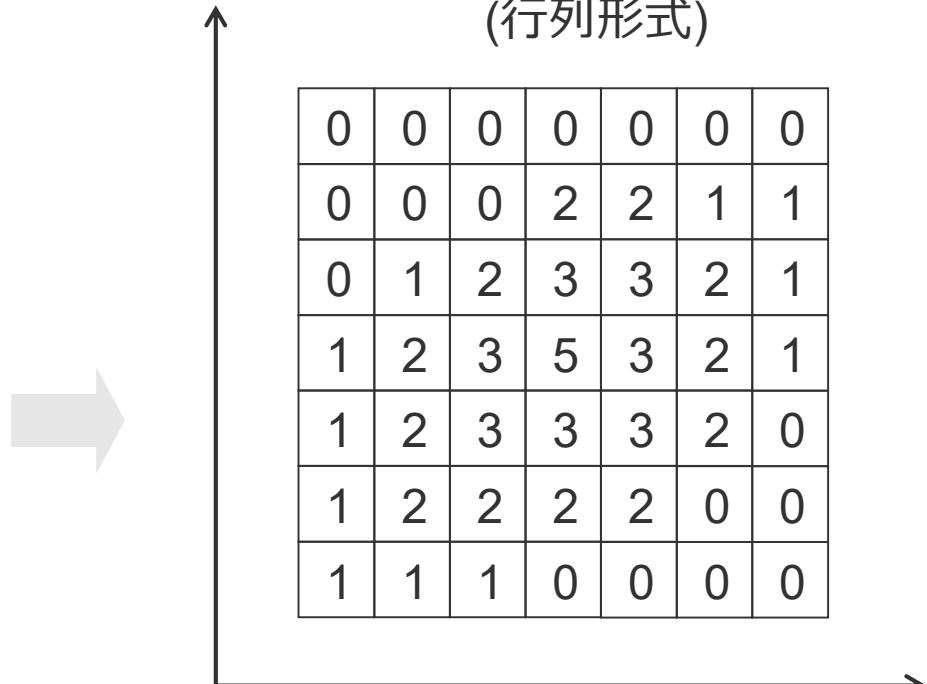
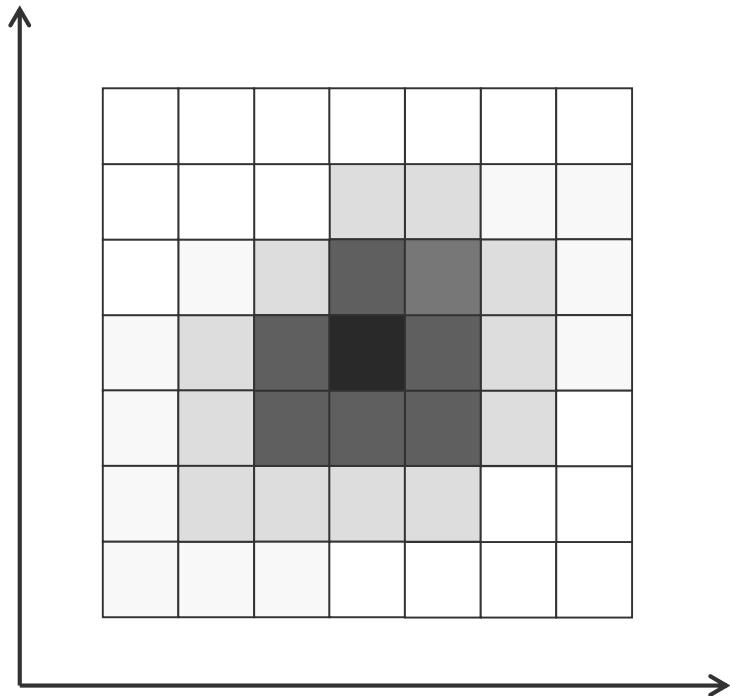
量子化

ロサンプリングされたデータの数値を離散的に分ける



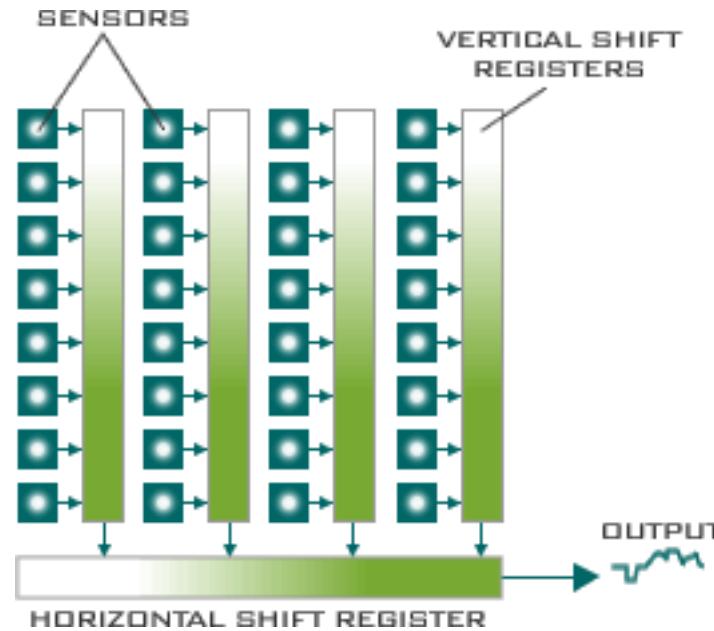
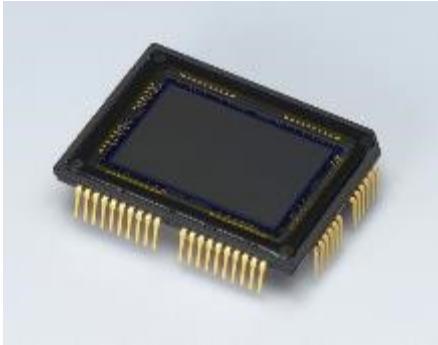
量子化（続き）

□ 2-Dデジタル画像の場合



量子化ビット数によって色数が決まる

受光素子



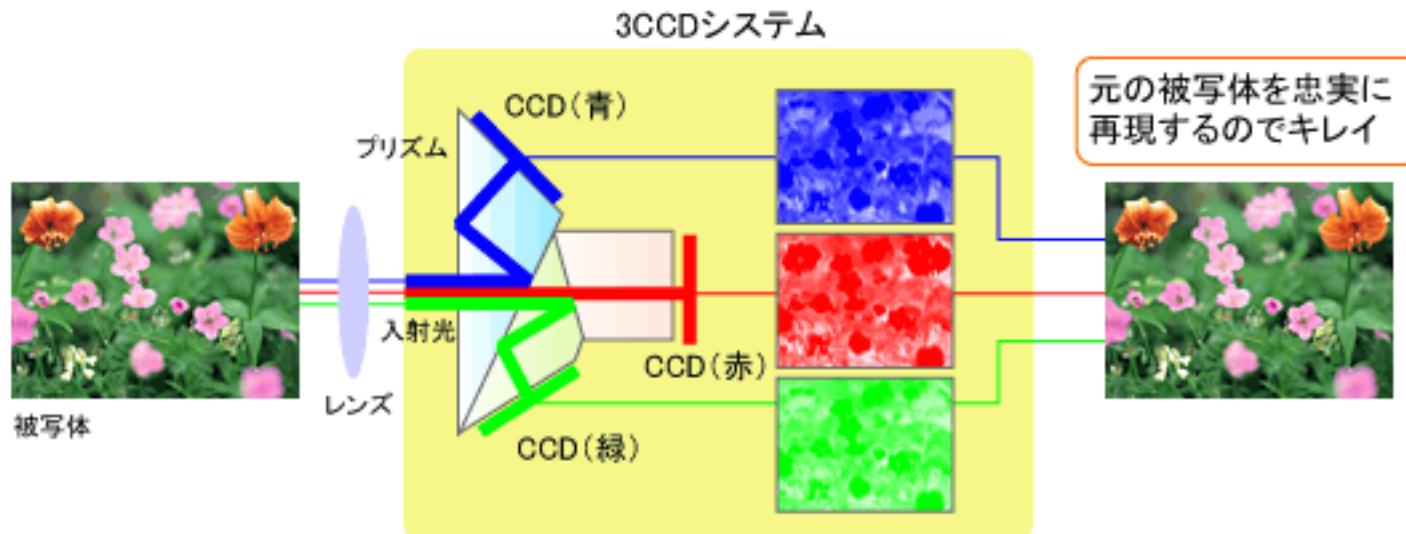
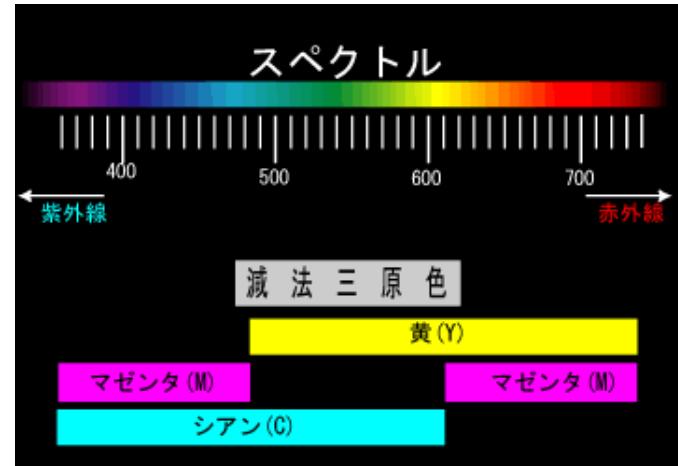
フィルムの代わりにCCDやCMOSといったデジタル素子が使用される
1画素に1素子！

例えば $4\,000 \times 3\,000 = 12,000,000$ 画素のカメラ：
1200万個の素子

色の表現

光（つまり色）は波であり連續値（物理でやったはず）→量子化の必要性？
例えば虹は7色に見えるが...
実は無限色ある

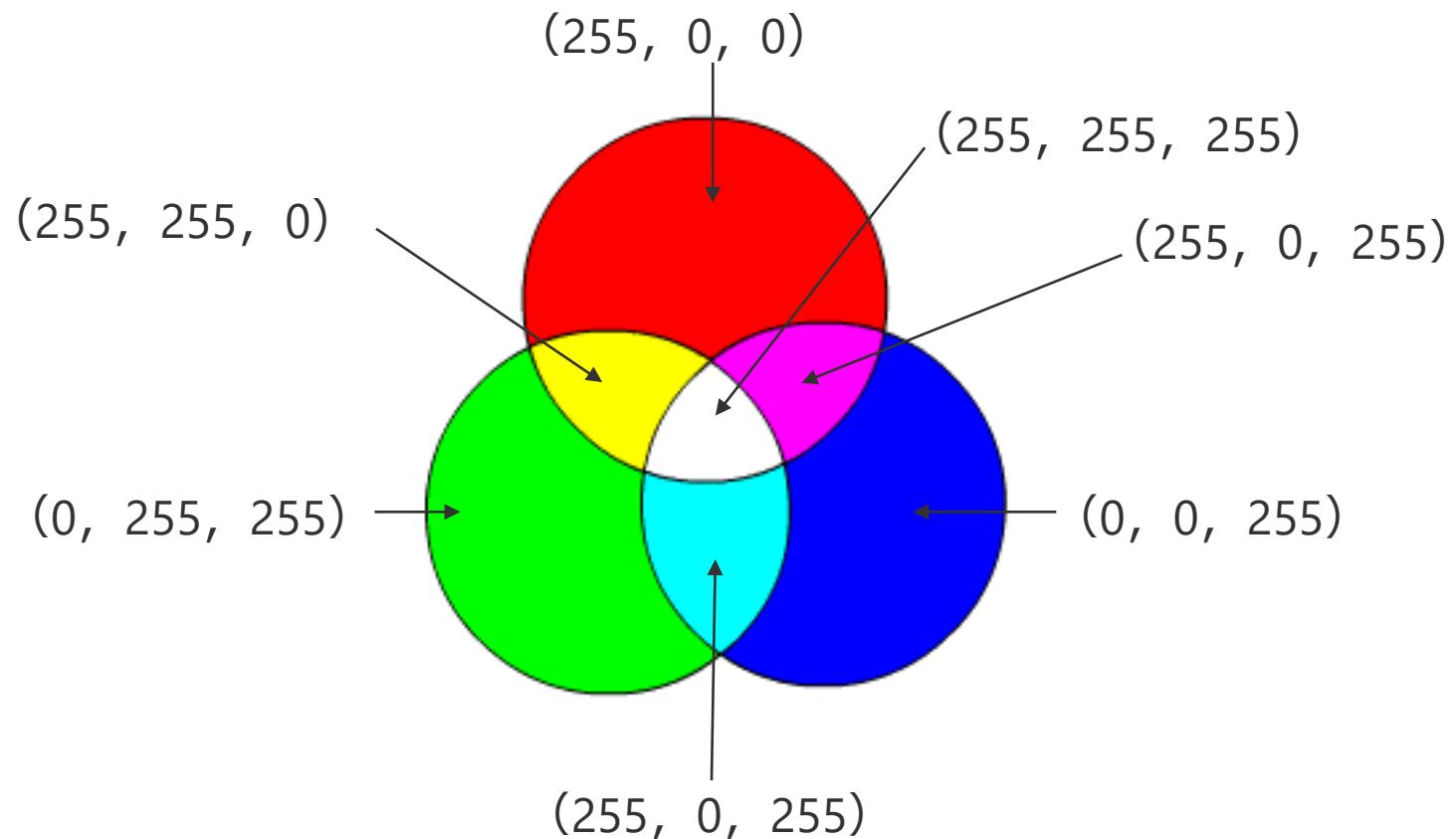
しかし実は3色あれば表現可能！
(人の目が3色しか反応しないため)



3CCDによる撮影
※ 通常のデジカメは1CCDのベイヤ式だがここでは簡単のため

光の三原色

一つの画素にR, G, Bにそれぞれ0～255の強さの値が一つづつ入って色を決める



表現の色数

「何色で十分か?」

256³なくても意外と見える



16.7 million
colors
256³



256
colors

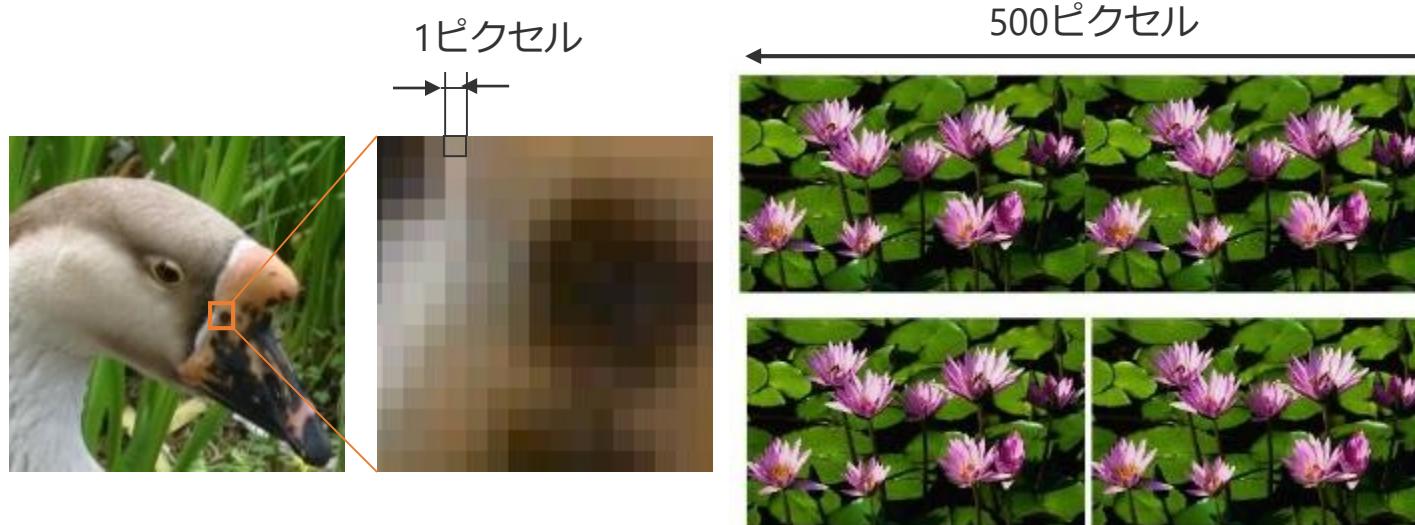


16
colors



4
colors

デジタル画像（再掲）



250ピクセルだと2枚横並びにできます。

ピクセルとは、デジタル画像を構成する単位である、色のついた「■」のことです。1677万色のうちの1色を選べます。（ $R \times G \times B = 256 \times 256 \times 256$ 、8bit (256色)）

3. ImageMagickによる 画像処理

- ImageMagick (イメージマジック) は画像を操作したり表示したりするためのコマンドラインのソフトウェア
- 基本的な画像処理が可能
- JPEG, PNG, GIF, PDF, TIFFなどの100種類以上の画像ファイルフォーマットに対応
- プログラム上からの画像の変換・編集などに必要な多数の機能を搭載



使い方

□ 基本コマンド

- convert [option] input_image output_image

□ convert

- 旧コマンド名（古いとこっち。V6）
 - convertはwindowsだと別のコマンドとかぶるため
 - WSLはLinuxが動いているのでconvert

□ magick

- 新コマンド名（V7）
- linuxではconvertで動く

※windowsの場合インストーラでインストール時にlegacy utilitiesをインストールすれば
convert.exe がインストールできるので、V7系でも convert.exe が使える

※CSEのバージョン（ちょっと古い）

Version: ImageMagick 6.4.3 2013-03-20 Q16 OpenMP
(2018年現在)

使用例

□ 画像フォーマット変換

- magick src.jpg out.png
 - jpeg画像をpng画像に変換
- magick src.png out.jpg
 - png画像をjpeg画像に変換
- magick src.png -quality 20 out.jpg
 - png画像を品質20（最小1, 最大100）でjpeg画像に変換.
- mogrify -format jpg *.png
 - ディレクトリ内のpngファイルをすべてjpegファイルに変換

□ 画像表示

- imdisplay src.jpg
 - windows
- display src.jpg
 - linux

□処理したものを即座に表示する

-windows

- magick lena.png -quality 20 out.jpg |imdisplay out.jpg

-Linux

- convert lena.png -quality 20 out.jpg |display out.jpg

-画像表示コマンドimdisplay もしくはdisplayを パイプ“|”でつなぐ

- 複数の画像をpdfにまとめる。bashのブレーズ展開を利用
- convert a_01{.jpg,.pdf}

- サイズ変更
- 回転
- 二値化
- エッジ検出
- 白黒化
- 減色
- 彩度, 色相の変換
- コントラスト強調,
ガンマ変換

- ぼかし
- 選択的ぼかし (バ
イラテラルフィル
タ)
- メディアンフィル
タ
- 先鋭化
- ノイズ除去
- ノイズ付与

※一部抜粋でもっと機能はある

サイズ変更

□ `convert -resize 50% lena.png out.png`

- サイズを半分に

□ `convert -resize 1024x1024 lena.png out.png`

- 画像サイズを1024 x 1024に直接指定

□ `convert -resize 200% lena.png out.png`

- 画像サイズが512の場合、上の直接指定と同じ

□ `convert -resize 100x lena.png out.png`

- 縦を省略。縦横比は固定

□ `convert -resize x100 lena.png out.png`

- 横を省略。縦横比は固定

□ `convert -resize 2048x1024! lena.png out.png`

- 画像サイズを2048 x 1024に直接指定。“!”がないと縦横比をキープしようとするためうまくいかない

□ `convert -scale 200% lena.png out.png`

- resizeよりも速いが画像の品質は低い。

□convert -rotate 30 lena.png out.png

□convert -rotate -30 lena.png out.png

-画素値が無い場所は、白になる。白い部分を含めると実画像サイズも大きくなる



二値化

- convert lena.png -threshold 30% out.png
 - 明るい場所を白に, 暗い場所を黒にする. この場合, 明るさが最大の 30 %以上の場合に白にする.
- convert lena.png –auto-threshold OTSU out.png
 - 自動でパラメータを決める二値化
 - OTSU以外にもTriangle, Kapurがある



convert lena.png -canny 0x1+5%+20%
out.png

-Cannyエッジ検出。5%—20%と書いてある場所の数字を変えると結果が変わる。小さいほど細かい模様まで出る。



白黒化

50

convert -colorspace gray lena.png out.png

- 白黒に変換

convert -colorspace lineargray lena.png out.png

- 上と少しだけ違う白黒変換

convert lena.png -monochrome out.png

- 新聞っぽい白黒変換

convert lena.png -sepia-tone 100% out.png

- セピアカラーに変換

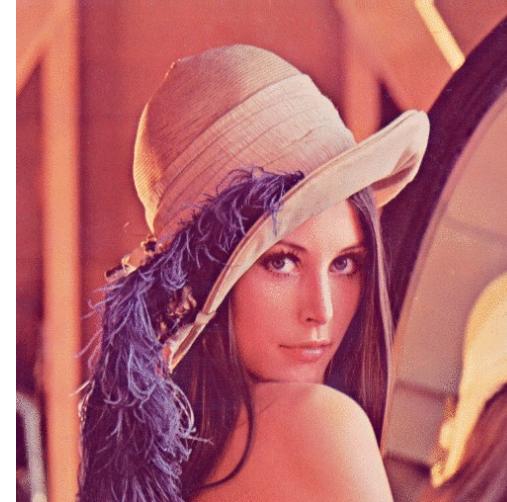


□convert -colors 128 lena.png out.png

–色の数を減らす。この場合128色

–デフォルトのカラー画像は、
 $256 \times 256 \times 256 = 16,777,216$ 色

□convert -posterize 4 lena.png out.png



明るさ，彩度，色相の変更

52

convert -modulate 100,100,100 flower.png
out.png

-100,100,100が明るさ，彩度（鮮やかさ），色相を表し，すべて100は何もしないに等しい。



100, 100, 100



150, 100, 100



100, 200, 100



100, 100, 50



100, 150, 100



80, 120, 180

modulateがおかしかったら？

53

convert -set option:modulate:colorspace
hsv -modulate 50 inputimage outputimage



なんかこうなつ
てしまうときは
上記で変換

hsv色空間を指定



modulateがおかしかったら？ 2 54

- convert -level 0%,200% out1.png out.png
 - これでもOK
- levelは入力を0:100%に再割り当てる関数
- 上記の意味は、0を0に，200%を100%にマップするので，値が半分になる。

コントラスト強調

convert -equalize beijing.png out.png

– コントラストを自動で強調

※その他コントラスト補正はここも参考になる
<https://qiita.com/yoya/items/1da8e8e04881bab67015>

convert -contrast beijing.png out.png

– コントラストを強調

convert +contrast beijing.png out.png

– コントラストを弱める

convert -contrast -contrast -contrast beijing.png out.png

– コントラストを3回強調



入力



強調



強調 × 3



弱める



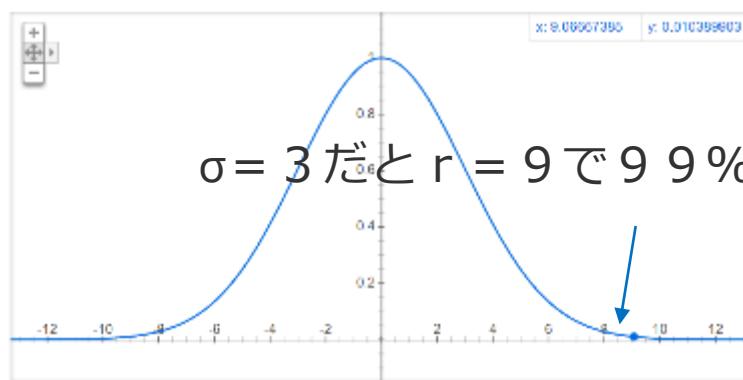
自動

□ convert -blur 10x30 lena.png out.png

- 半径 10, シグマ 30 のガウスぼかし. 半径 $\times 3$ くらいのシグマにすると 99% の裾になる.
- -gaussian-blur オプションは 2 次元畳み込み実装. こっちはセパラブル実装なので速い.

□ convert -selective-blur 10x30+30% lena.png out.png

- 古いともないかも
- 選択的ガウスぼかし (バイラテラルフィルタ) 30% くらいの近さの画素だけガウスぼかし. ノイズ除去に有効.
- -mean-shift にするとこれを何度も繰り返したものと似た結果が得られる

グラフ: $\exp(-(x^2)/(2 \cdot 3^2))$ 

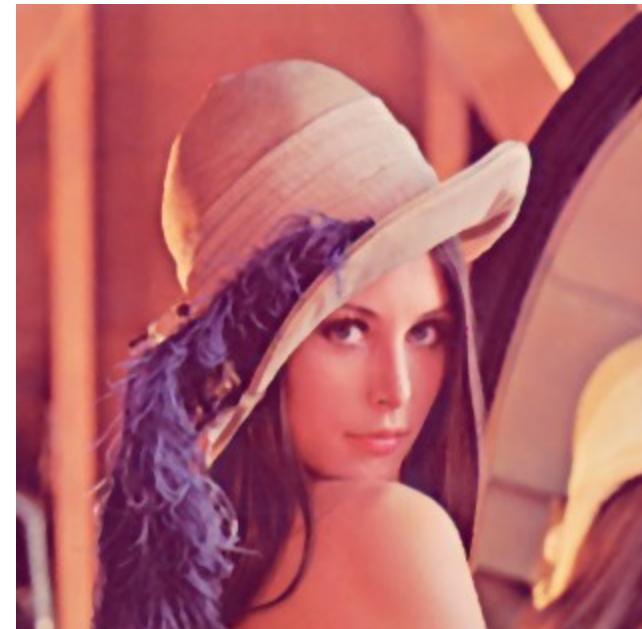
ガウスぼかし



選択的ガウスぼかし

convert -median 1 lena.png out.png

- medianの後は、半径を指定なので $2r+1=3\times 3$
- 3×3 の範囲の中央値でフィルタ. 例外がある画素を修復可能.
- 例外が大きい場合は半径を 1, 2, . . . , n と大きくする.



convert -sharpen 7 lena.png out.png

- 数字が大きいとより広い周辺をみて強調。unsharpでより詳細な指定が可能

convert -adaptive-sharpen 7 lena.png out.png

- こちらのほうがきれい

convert -adaptive-sharpen 7 -adaptive-sharpen 7 lena.png out.png

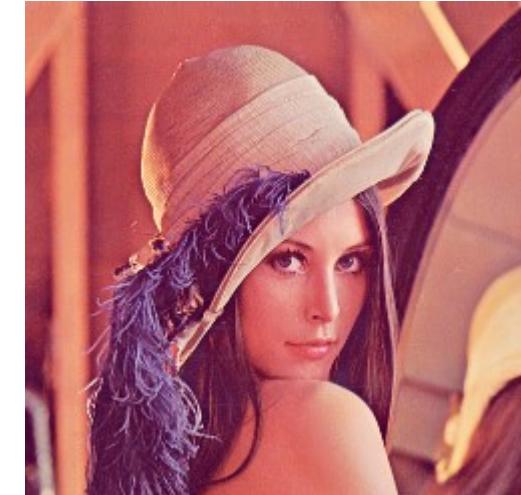
- なんどもやるとより強調



入力



sharpen



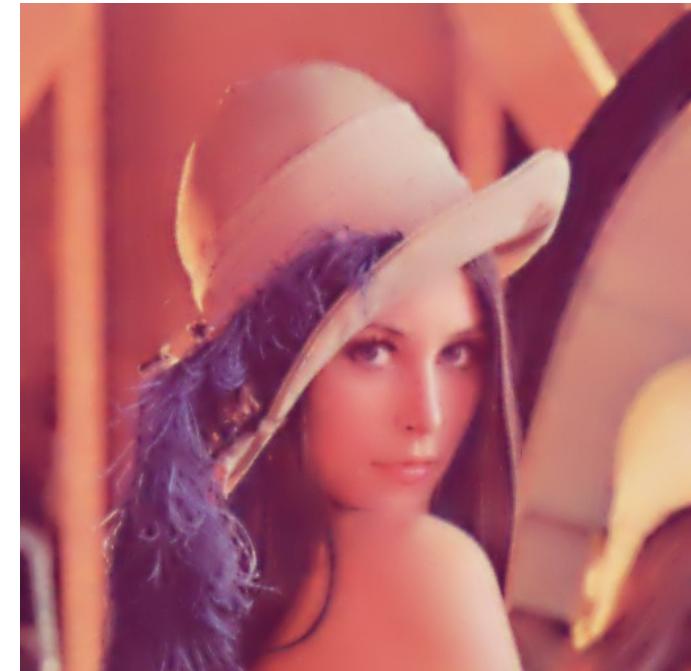
adaptive-sharpen

ウェーブレットデノイズ

59

convert -wavelet-denoise 20% lena.png
out.png

- ウェーブレットを使ったデノイズ
- 古いとないかも



ノイズ付与

convert +noise Impulse lena.png out.png

- これでノイズを追加した画像を作れる。メディアンフィルタの時に追加したノイズはImpulse。
- 画像のノイズはガウシアンであることが多い。
- ノイズのタイプ : Gaussian, Impulse, Laplacian, Multiplicative, Poisson, Random, Uniform

convert lena.png -attenuate 4 +noise Gaussian out.png

- ノイズ量の強化。数字を大きくするとノイズが増える。

□ optionを並べた順番に画像処理が適用

– convert -rotate 30 -threshold 30% -blur 3x6 src.ppm out.ppm

□ 30度回転して、閾値処理をしたあとガウスぼかし



□ImageMagickのコマンドオプションマニュアル（英語）

–必要なことは全部書いてあります

<http://www.imagemagick.org/script/command-line-processing.php#option>

- <http://imagemagick.rulez.jp/>
- <https://qiita.com/mtakizawa/items/94b4cb6d2da99482c6ac>
- <http://shumilinux.blogspot.jp/2016/06/imagemagick-mogrify.html>
- http://www.gi.ce.t.kyoto-u.ac.jp/user/susaki/image/magick_process.html

4. テンプレートマッチングの C言語プログラミング

テンプレートマッチング

65

入力画像からテンプレートに似た場所を探す方法

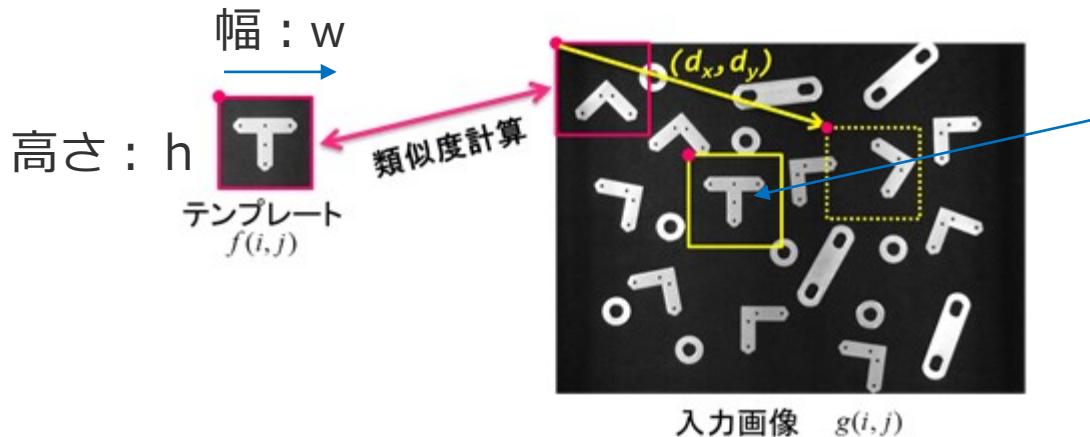


テンプレート



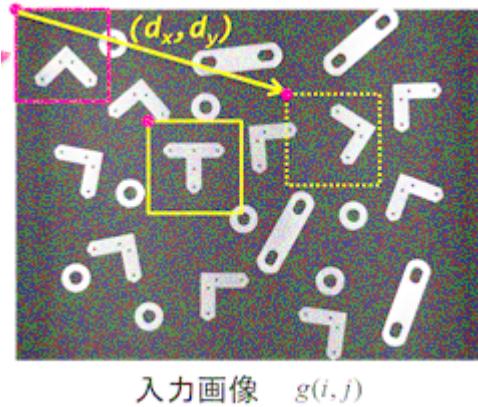
入力画像

似ているとは？



こここの差分は 0 になるはず！
こここの座標が分かれれば見つけた
ことになる。

探したい画像であるテンプレートと入力画像が完全に一致すれば、差分が 0 になる。
ただし、いつも画像がきれいにとれるとは限らず、完全に一致することはまれ。



代わりに差の二乗和で測る。

これを類似度の「SSD : Sum of Square Difference」と呼ぶ。

類似度

$$\sum_{j=0}^{h-1} \sum_{i=0}^{w-1} (f(i, j) - g(x + i, y + j))^2$$

実際の撮影画像の例

※画像は中京大橋本研究室より

色々な類似度

□類似度には色々あり、ノイズに強い、コントラスト変換に強いなど特色がある

- SAD (Sum of Absolute Difference)

- 二乗和ではなく絶対値をとる

- 正規化相互相関 (NCC)

- 相関係数で計る

- 0平均正規化相互相関 (ZNCC)

- 似ている画素の個数で測る

- 微分画像のSAD, SSD

- 相互情報量

※詳しくは最後のページの橋本先生の資料を参照のこと

- makeをコマンド打つと、C言語のプログラムから、matchingという実行ファイルが生成され、これを使えばテンプレートマッチングが可能
- 詳しい説明はあと

実装している関数

□ カラーとグレイのSSD

□ 画像全部を探索して、もっとも類似度が小さくなった点を全画像を走査して見つける

- void templateMatchingColor(Image* src, Image* template, Point* position, double* distance)

- void templateMatchingGray(Image* src, Image* template, Point* position, double* distance)

□ positionが発見した位置

□ distanceがその位置の類似度

□ これを改良することで精度向上 & 速度向上

コンパイルの仕方

- 「make」と打つだけ
- コンパイルしなおしたいのに
"make: `matching' は更新済みです"
と出たら 「make clean」と打てばOK
- コンパイルの方法の指定はMakefileを編集

□ Makefile

- gcc -c a.c
- gcc -o a.out a.o
- などの複数のコマンドをまとめてくれるもの.
- make cleanでごみを削除とかも書ける.
- makeと書けばコンパイル可能.
- 授業でやった？

□ 参考情報

- <https://qiita.com/petitviolet/items/a1da23221968ee86193b>

テンプレートマッチング プログラムの使い方

□ matching 入力画像 テンプレート画像 回転角 閾値 オプション[cwpg]

□ 例

- matching src_image.ppm template_image.ppm 0 1.0

- オプションなし, 回転角なし, 閾値1.0でマッチング. 閾値は, 類似度がこれを下回ったら発見したとする数字.
- 回転情報を入れても回転しないので, imagemagickで回転させること. この情報は結果を保存するために使用されている.

- matching src_image.ppm template_image.ppm 0 1.0 cwpg

- グレイ画像として処理して, 結果をprintしつつ, 検出結果を画像として書き出し, 結果txtを初期化してから保存

□ オプション

- c : 結果をクリアしてから結果を記述. クリアしないと, 実行のたびに結果が結果ファイルに追加されている.
- p : 結果をprintしてコンソールに出力
- w : 結果を画像として書き出し. 発見した場所を赤い枠で囲む. 出力画像名はout.ppm
- g : マッチングをカラーではなくグレイ画像で行って高速化

※ 後述するshellスクリプトからmatchingを呼び出すため, 実際はコマンドを打つことはないかも?

サンプルコードの 対応画像フォーマット

□ 対応フォーマット

- ppm, pgm

- 非圧縮の画像フォーマット. ppmがカラー, pgmがグレイ.

□ 未対応のメジャーなフォーマット

- jpeg

- 自然画像用の圧縮フォーマット

- png

- 可逆圧縮フォーマット

□ 入力画像が, jpeg, pngの場合はImageMagickで ppm, pgmに変換してから使うこと

- C言語をコンパイルするものはgccだけではない。
 - gcc : GNU Compiler Collection. linux, unixの基本はこれ
 - icc : Intel C++ Compiler, Intel CPU向け. 有料.
 - vc : Microsoft VisualStudioのコンパイラ
 - clang : LLVMベースのコンパイラ
- コンパイルした結果のアセンブリは、各コンパイラごとに違う可能性. 実行結果はほとんど同じ.
- コンパイラオプションを変えると、アセンブリがより速く動くように最適化される.
- CSEではgccとiccが使える
 - iccのほうが、プログラムの実行速度が速いことが多い.
 - コンパイル時間はiccのほうが長いことが多い.

5. シェルスクリプト プログラミング

シェルスクリプトとは？

□コマンドを並べて実行するもの

□`for`, `if`, 変数, 関数などがあるため, 複雑な制御も可能

シェルスクリプトサンプル1

77

- test.shでファイルを作成して以下を記述
- #!/bin/sh はシェルスクリプトであることを書くもの
- echoはecho以下の引数をプリントする命令
- sh test.sh で実行可能
- もしくは, "chmod -x test.sh"してから, "./test.sh"

```
#!/bin/sh
```

```
echo "Hello, World!"
```

シェルスクリプトサンプル2

78

- test.shでファイルを作成して以下を記述
- #以下はコメント（冒頭を除く）
- このシェルスクリプトは、画面をクリアしたあとにlsを実行するスクリプト

```
#!/bin/sh

#画面をクリア
clear
#lsの命令
ls -l -a
```

□#以降はコメント行となり¥で改行できる

```
#!/bin/sh
# コメント
echo "Hello, ¥
bourne shell world."
exit 0
```

値の代入（宣言がない）

VAR=value
（=の前後にスペースを入れないこと！）

値の参照

\$VAR
\${VAR}
参照の仕方の違いは、スペース、スラッシュ、コロン、ハイフンといった記号の文字列内での処理で異なります。

```
#!/bin/sh

COLOR=green
echo "/home/who/$COLOR/here"
echo "/home/who/${COLOR}/here"
echo "This looks $COLORish"
echo "This seems ${COLOR}ish"
exit 0
```

引数などを使うことが出来る

-Cの場合は、 int main (int argc, char** argv)

\$0	スクリプト名
\$1 ~ \$9	引数 1 番目から 9 番目
\$*	すべての引数 ("\$*"とした場合、"\$1 \$2 ..."のように展開)
\$@	すべての引数 (ダブルクオートで囲んだ場合の処理が \$* と異なり個別に引用される)
\$#	引数の数

```
if 条件; then  
    コマンド  
elif 条件; then  
    コマンド  
else  
    コマンド  
fi
```

```
#!/bin/sh  
#  
USER_ID=`/usr/bin/id -u`  
  
if [ $USER_ID -ne 0 ]; then  
    echo "You must be super-user to execute $0"  
    exit 1  
fi  
  
echo "Welcome to $0 script world"  
  
exit 0
```

スーパーユーザー以外は修了するプログラム

```
for 変数 in リスト
do
    コマンド
done
```

```
#!/bin/sh

HOST_LIST="h1.example.com h2.example.com h3.example.com"
for HOST in $HOST_LIST
do
    echo "$HOST"
done
```

ホスト数だけループ

□変数は全部グローバル変数になることに注意。□一カル変数という概念はない

```
name()  
{  
    なにかコマンド  
}
```

シェルスクリプトの文法中にコマンドを実行する場合、そのコマンドの結果がほしい場合はバッククオートで囲む

- bname='basename \${image}'
- echo `basename \${template}`

□初心者向けシェルスクリプトの基本コマンドの紹介

- <https://qiita.com/zayarwinttun/items/0dae4cb66d8f4bd2a337>

□時代はシェルスクリプト

- <https://qiita.com/b4b4r07/items/726df1543fc48d2cb20b>

- 各命令を並列に実行することで高速化
- 詳しくは以下参照
 - シェルスクリプトで単純に並列実行・直列実行を行う
 - <https://qiita.com/nyango/items/7b6b719f248b2ee8d379>

- シェルスクリプトではなく、perl, pythonなどのスクリプト言語を使って也可.
- できのよいスクリプトも評価.
- シェルスクリプトなどを駆使して、グラフを自動生成するプログラムを作ったり. . .

6. 実行マニュアル

実行方法概要

- 以下のURLからコード・実験データをzipでダウンロードして、
適当なフォルダ(例えばmyouyou)に展開
 - https://github.com/fukushimaLab/advanced_programming
- cd myouyou
 - 適切なディレクトリに移動
- make clean
 - (ゴミがたまってたら)
- make
 - コードをコンパイル
- time sh run.sh level2
 - シェルスクリプトを実行
- sh answer.sh result level1
 - 答え合わせ

ディレクトリ構成

- advanced_programming_signal_processing
 - これが本体. この中のディレクトリに入って作業する.
- docs
 - 説明用のパワーポイントやpdfが入っている.
- dataset
 - 必要ない. 演習のテンプレートなどを生成するためのプログラム. 興味がある人用. OpenCVが必要&C++で書かれている.

ディレクトリ構成

- /image テスト用の画像が入っている
- /imgproc run.sh中のimagemagickの結果保存用
- /result matchingの結果保存用
- /levelx
 - テンプレート画像が入っている
 - /test テスト画像が入っている
 - /final 最終テスト用の画像が入っている
- Makefile makeファイル
- imageUtil.c 画像処理関数群. 編集必要なし
- imageUtil.h 上のヘッダ
- main.c テンプレートマッチングはここ
- run.sh 実行用シェルスクリプト

□timeコマンド

- time “コマンドもしくはシェルスクリプト”
- 例えば3秒スリープするコマンド

```
$ time sleep 3  
real 0m3.008s  
user 0m0.000s  
sys 0m0.000s
```

—読み方

- **real** プログラムの呼び出しから終了までにかかった実時間（秒）
- **user** プログラム自体の処理時間（秒）（ユーザCPU時間）
- **sys** プログラムを処理するために、OSが処理をした時間（秒）（システム時間）

※報告は**real**の時間だけでよい

□ 基本

- gcc -o test test.c
- (-O0になっているはず?)

□ 最適化コンパイル

- | | |
|-----------------------------|-------------|
| - gcc -O0 -o test test.c | (最適化なし) |
| - gcc -O1 -o test test.c | (最適化弱) |
| - gcc -O2 -o test test.c | (最適化中) |
| - gcc -O3 -o test test.c | (最適化大) |
| - gcc -Ofast -o test test.c | (最適化最大) ※ 1 |

□ その他

- -lm 数学関数用
- -w ワーニングをすべて消す
- -Wallワーニングをすべて出す

※ 1 コンパイラのバージョンによっては対応していない

□run.sh

- 画像処理を行ってからマッチング
- オプションはpを有効化
- デフォルトは画像のコピーをする（何もしない）画像処理を実行

スクリプト実行出力例

- hogehoge@cse:~/hoge> time sh run.sh level7
- -----./imgproc/level7/1/level7_018.ppm-----
- [Not found] 000ocean_beach_kinzokutanchi 660 389 64 64 0 0.360132
- [Not found] 000kids_chuunibyou_girl 332 246 64 79 0 0.493493
- [Not found] 000airgun_women_syufu 427 414 64 59 0.355795
- [Not found] 000mokuzai_hakobu 105 176 64 64 0 0.446102
- [Found] 200kids_chuunibyou_girl 260 296 128 158 0 0.123424
- [Not found] 200mokuzai_hakobu 79 343 128 128 0 0.182720
- [Not found] 050kids_chuunibyou_girl 456 453 32 40 0 0.480053
- [Not found] 050ocean_beach_kinzokutanchi 587 383 32 32 0 0.401376
- [Not found] 050mokuzai_hakobu 211 249 32 32 0 0.521351
- [Not found] 200ocean_beach_kinzokutanchi 527 317 128 128 0 0.140866
- [Not found] 050airgun_women_syufu 6 376 32 30 0 0.466503
- [Not found] 200airgun_women_syufu 173 224 128 118 0 0.156752
- [Not found] 180ocean_beach_kinzokutanchi 378 325 64 64 180 0.273984
- [Not found] 270kids_chuunibyou_girl 426 351 79 64 270 0.258538

シェルスクリプト (run.sh) 解説 97

```
#!/bin/sh
# imagemagickで何か画像処理をして, /imgprocに書きこみ, テンプレートマッチング
for image in $1/test/*.ppm; do #第1引数/testのディレクトリ内のすべてのppmファイルに関してループ
    bname=`basename ${image}`
    name="imgproc/"$bname
    x=0 #変数xを0に
    convert -median 1 "${image}" "${name}" #ImageMagickで処理して/imgprocに書き出し
    rotation=0 #回転の値をセット. 今回は何もしないので0
    echo $bname: #ベースネームを表示
    for template in $1/*.ppm; do #第1引数のディレクトリ内すべてのppmに関してループ
        echo `basename ${template}` #テンプレート画像の名前を表示
        if [ $x = 0 ] #もし変数xが0だったら?
        then
            #下記コマンドを実行. cpは, 出力テキストをクリアして, 結果をprint
            #オプションや閾値の変更を試してみること.
            ./matching $name "${template}" rotation 0.5 cp
            x=1 #xに1を代入して, 同じ画像ファイルではクリア命令をしないようにする
        else
            #最初以外 (x=1)はcオプションを使わない
            ./matching $name "${template}" rotation 0.5 p
        fi
        done
        echo "" #空行を表示
    done
```

Makefile解説

98

```
matching: main.c imageUtil.c
#gcc -w -O3 -o matching main.c
imageUtil.c -lm
icc -w -fast -o matching main.c
imageUtil.c -lm

clean:
$(RM) matching
```

gccかiccでコンパイル
#はコメントアウト
残りはいろいろなオプションを試した名残
初期値はgccにセットしてある

□出力は入力画像名.txt

- 中身

- airgun_women_syufu 715 412 64 59 0 0.000000
- テンプレート名 座標x 座標y 幅 高さ 回転 マッチングスコア
- マッチングスコアは、SSDをテンプレートサイズで割ったものになっている

□matchingのオプションにwがついている場合、下図のように発見結果が赤枠で囲まる。

□失敗の場合出力

- その時は変な場所に赤枠が出る。



正解判定シェルスクリプト

100

□ sh answer.sh result level1

– 結果のフォルダ、レベルのフォルダを指定したら
判別可能

□ 不正解の場合は、

– echo "[NOT CORRECT]"

– echo "[NOT CORRECT (NOT MATCH TEMPLATE)]"
NOT CORRECTの場合は、テンプレートはあって
るけど、位置とかの情報があわない場合

7. 課題

課題の処理手順

102

1. advanced_programming_signal_processingディレクトリに移動

```
cd advanced_programming_signal_processing
```

2. makeでC言語プログラムのmatchingを生成

```
make
```

3. level1のテンプレートマッチングを実行（1~7は演習次第でかかる）

```
time sh run.sh level1
```

4. 結果をチェック

```
sh answer.sh result level1
```

5. ファイルの編集

- Makefileを編集コンパイルオプションの変更
- run.shを編集して、ImageMagickやmatchingの引数を変更
- main.cを編集してテンプレートマッチングの関数を変更

変更の後、2~4の繰り返し。

- imagemagickで様々な画像処理を実行せよ.
- image/lena.ppmの画像中のインパルス雑音を消せ.

- 課題のレベル1をクリアせよ
 - 非常に時間がかかるはず
- matchingのコンパイルオプションを変えると実行速度が変わることを確認せよ
 - 一気に速くなるはず
- matchingの実行オプションを変えると実行速度が変わることを確認せよ
 - グレイスケールを使うとさらに速くなるはず. 精度はわからない.
- 課題のレベル2以降にチャレンジせよ
- 来週以降の分担・計画を立てよ

レベル1 結果 (動作確認用)

104

```
fukushima@letsnote:~/advanced_programming_signal_processing$ sh answer.sh result level1
```

```
level1_000.txt
```

```
[CORRECT]
```

```
level1_001.txt
```

```
[CORRECT]
```

```
level1_002.txt
```

```
[CORRECT]
```

```
level1_003.txt
```

```
[CORRECT]
```

```
level1_004.txt
```

```
[CORRECT]
```

```
level1_005.txt
```

```
[CORRECT]
```

```
level1_006.txt
```

```
[CORRECT]
```

```
level1_007.txt
```

```
[CORRECT]
```

```
level1_008.txt
```

```
[CORRECT]
```

```
level1_009.txt
```

```
[CORRECT]
```

```
===== デフォルトで正解率100%で出るはず.
```

```
SUMMARY
```

```
CORRECT RATE: 10/10
```



全て一致

```
ANSWER FILE NOT EXIST: 0
```

```
=====
```

レベル2 結果 (動作確認用)

105

```
fukushima@letsnote:~/advanced_programming_signal_processing$ sh answer.sh result level2
level1_000.txt
[ANSWER FILE NOT EXIST]
level1_001.txt
[ANSWER FILE NOT EXIST]
level1_002.txt
[ANSWER FILE NOT EXIST]
level1_003.txt
[ANSWER FILE NOT EXIST]
level1_004.txt
[ANSWER FILE NOT EXIST]
level1_005.txt
[ANSWER FILE NOT EXIST]
level1_006.txt
[ANSWER FILE NOT EXIST]
level1_007.txt
[ANSWER FILE NOT EXIST]
level1_008.txt
[ANSWER FILE NOT EXIST]
level1_009.txt
[ANSWER FILE NOT EXIST]
level2_000.txt
[NOT CORRECT (IGNORE RESULT FILE)]
level2_001.txt
[NOT CORRECT (IGNORE RESULT FILE)]
level2_002.txt
[NOT CORRECT (IGNORE RESULT FILE)]
level2_003.txt
[NOT CORRECT (IGNORE RESULT FILE)]
level2_004.txt
[NOT CORRECT (IGNORE RESULT FILE)]
level2_005.txt
[NOT CORRECT (IGNORE RESULT FILE)]
level2_006.txt
[NOT CORRECT (IGNORE RESULT FILE)]
level2_007.txt
[NOT CORRECT (IGNORE RESULT FILE)]
level2_008.txt
[NOT CORRECT (IGNORE RESULT FILE)]
level2_009.txt
[NOT CORRECT (IGNORE RESULT FILE)]
```

```
=====
SUMMARY
CORRECT RATE: 0/10
ANSWER FILE NOT EXIST: 10
=====
```

そのままやつたら全く見つけられず“0%になるはず”。

Level1の結果を無視した個数
(level1のファイルも入っているため)

□レベル7までをクリアする

–速度・精度のどちらかを追求するか，そのバランスを追求するか，もしくはどちらも追及する

unexpected operatorって出た場合はおそらく2つ以上マッチしている。
答えは1つなので，間違っている。

□目標：異なる画像を使って検証する

□run.shの簡単な修正が必要

※ 画像が増えて20個

□答え合わせのコマンドは

「sh answer_final.sh result level1」

※ 1～7を適切に変更

```
#!/bin/sh
# imagemagickで何か画像処理をして、/imgprocに書きこみ、テンプレートマッチング
for image in $1/test/*.ppm; do #第1引数/test/のディレクトリ内すべてのppmファイルに関してループ
    bname=`basename ${image}`
    name="imgproc/"$bname
    x=0 #変数xを0に
    convert -median 1 "${image}" "${name}" #Imagemagickで処理して/imgprocに書き出し
    rotation=0 #回転の値をセット。今回は何もしないので0
    echo $bname: #ベースネームを表示
    for template in $1/*.ppm; do #第1引数のディレクトリ内すべてのppmに関してループ
        echo basename ${template}` #テンプレート画像の名前を表示
        if [ $x = 0 ] #もし変数xが0だったら？
        then
            #下記コマンドを実行。cpは、出力テキストをクリアして、結果をprint
            #オプションや閾値の変更を試してみること。
            ./matching $name "${template}" rotation 0.5 cp
            x=1 #xに1を代入して、同じ画像ファイルではクリア命令をしないようにする
        else
            #最初以外(x=1)はcオプションを使わない
            ./matching $name "${template}" rotation 0.5 p
        fi
    done
    echo "" #空行を表示
done
```

testをfinalにリネームすると
別画像が使える

run.shのファイルの中身

8. 参考資料・ヒント

- レベル2 : imagemagickでメディアフィルタを使いましょう & 閾値の0.5をいじってみましょう.
- レベル3 : imagemagickでコントラスト強調する&& 閾値の0.5をいじってみましょう or/and matching を改良しましょう.
- レベル4 : imagemagickで何かの画像処理をする
(エッジを出してみるとか) or マッチングを改良しましょう.
- レベル5/6 : run.shにimagemagickで拡大・回転するループを入れるか, matchingを画像拡大・回転に 対応するように変更しましょう.
- レベル7 : すべてに対応するのは難しいことを理解したうえで総合的に頑張りましょう.

口テンプレートマッチングの魅力

－中京大 橋本先生

– <http://isl.sist.chukyo-u.ac.jp/Archives/SSII2013TS-Hashimoto.pdf>

さまざまなテクニック

- いらすとやの背景になっている部分（黒い部分は類似度として測らない）
- 正規化相互相関を使う
- カラーではなくグレイ画像を使って高速化
 - ただし精度は落ちるかも
- テンプレート画像を間引いて高速化
 - forループで $i++, j++$, となっているところを $i+=2, j+=2$ などに
 - ただし、精度は落ちるかも
- 疎密探索
 - 小さい画像で探索して、だいたいの位置をみつけてその周りをもう一度大きい画像で探索
- 探索打ち切り 1
 - もし、画像中にそのテンプレートの画像が 1 つしかないのなら、それなりに類似度が小さくなったら break して発見したことにすればいい。
- 探索打ち切り 2
 - 類似度のスコアが閾値以上になったらテンプレートの計算を途中で打ち切る

□画像に色が変わったり、ノイズが乗ったり、テンプレートの一部が欠損するなど、なる。そのある場合、最も良い結果を得るためには、以下の対策をとることで複数回実行し、うまくいく可能性があります。

- ノイズ除去をやってみる
- 白黒画像でやっている
- コントラストを変えてやってみる
- 色空間を変えてやってみる
- テンプレートの距離関数を変えてみる
- テンプレートの距離を測る場所にマスクを採用してみる

□ サイズが違つたり、いろんな方向に回転したりしている場合、テンプレートの形をいろいろ変えて挑戦する必要がある。

- テンプレートをリサイズしたり、回転させたりして複数回実行し、最も良さそうなものを選べばうまくいく可能性があります。
- 画像をぼかしてやってみると、少し拡大縮小や回転のパラメータがずれた場合でもうまくいく場合があります。
- 距離関数を変えてみると、上と同じく、少し拡大縮小や回転のパラメータがずれた場合でもうまくいく場合があります。
- 発展した方法だと特徴点による対応付けや位相限定相関法というアルゴリズムとかもあります。
- ディープラーニングでもできます。
- ほかにも多数あります。

口 いろんな工夫をすることで計算時間を高速化することができます。ただし、高速化すると探索が荒くなることもあります。精度が劣化することもあります。

- テンプレートを計算する画素を飛び飛びにすることで間引いて計算する。
- 入力画像を縮小して実行すると、探索範囲、処理範囲が小さくなるため高速に実行できる。
- カラー画像よりも白黒画像のほうが速く計算できる。
- バイナリ画像にするとxorとpopcountで何個画素が違っているかを高速に測ることができます。
- このように荒く検索したら、だいたい正解を見つけてくるが正確な位置になっていない場合がある。この場合は、荒い検索で求まつた結果を初期値として、その近くに答えがあると思って、大きな画像で少数の候補だけから探すことで、小さなコストで正確な位置を見つけることができる。
- 各画像を並列計算すると高速に計算できる。スクリプトを改変することができる。
- プログラムを立ち上げること自体に計算コストがかかるため、プログラムを改変して、共通のテンプレート、画像は一括して処理すると速くなる。

もちろん、これ以外にもたくさんあるので各自工夫してみてください。