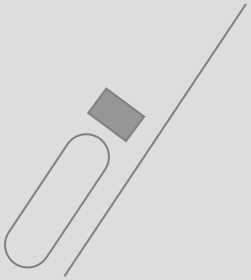
Boost.GIL ~Generic Image Library~
画像処理入門

2010/09/11

Leo



自己紹介

- Sugiyama Leo (杉山 礼央)
  - 某メーカー勤務
  - 学生時代は画像処理関連の研究をしてました
    - Boost.GILは学生時代に使っていただけ
- HatenaID: sosuke2000
- http://www25.atwiki.jp/guru/

- E-mailアドレスは個人的に聞いてください…

今日の目標

（初歩的な）
動体検出ができるようになる！

GILとは

画像データ保持用
コンテナとイテレータ

出典：Let’s Boost　[http://www.kmonos.net/alang/boost/]

目次

1. 前提知識

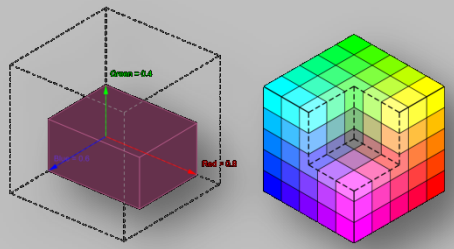
1. 前提知識
    1. 色空間
    2. 画像の種類
    3. 画像の構造:概念
    4. 画像の構造:データ構造
    5. 動画像の構造
2. Boost.GIL基礎の基礎
    1. 画像の入出力
    2. 画像へのアクセス
    3. ピクセルへのアクセス
    4. チャネルへのアクセス
3. 動体検出プログラムの作成
    1. 動体検出アルゴリズム
    2. Boost.GILでの実装(基礎)
4. おまけ
    1. Boost.GIL参考情報

色空間

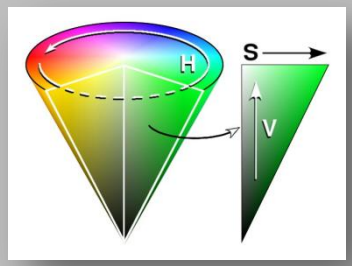
色には様々な表現方法がある

色空間の例

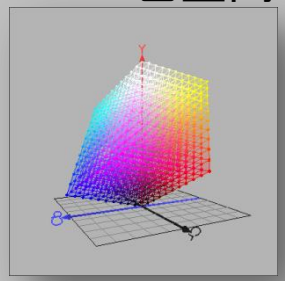
RGB/RGBA色空間



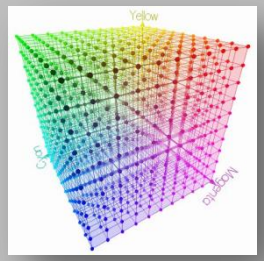
HSV色空間



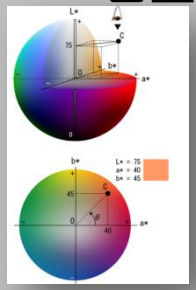
YCbCr色空間



CMY/CMYK色空間



L*a*b*色空間



Grayscale

出典：Wikipedia、その他

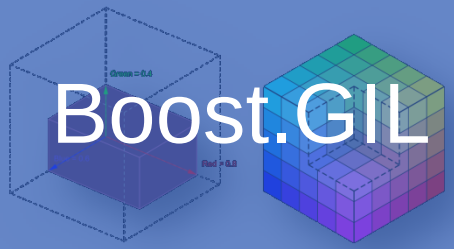
色空間

GILでは3種類の色空間をサポートする

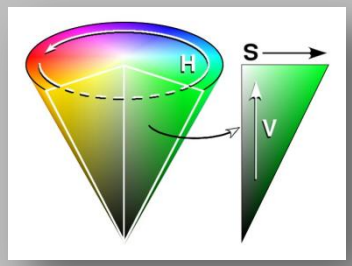
色空間の例

頑張れば拡張可能…

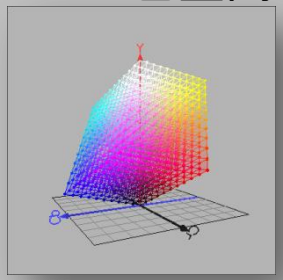
RGB/RGBA色空間



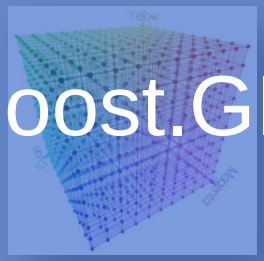
HSV色空間



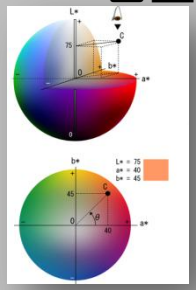
YCbCr色空間



CMY/CMYK色空間



L*a*b*色空間



Grayscale

出典：Wikipedia、その他

画像の種類

画像には2種類の分類がある

ベクタ画像

- 点や線・面の座標と、属性（色や、パターンなど）の幾何情報を保持する
  - 拡大・縮小しても劣化しない
  - 幾何情報が前提のため、写真などには不向き

ラスタ画像（ビットマップ画像）

- 微細な点（ピクセル）の集まりを保持する
  - ピクセル毎に色を持っている
  - BMP、JPEG、PNG、GIFなどが一般的

画像の種類

GILはラスタ画像（ビットマップ画像）を対象とする

ベクタ画像

- 点や線・面の座標と、属性（色や、パターンなど）の幾何情報を保持する
  - 拡大・縮小しても劣化しない
  - 幾何情報が前提のため、写真などには不向き

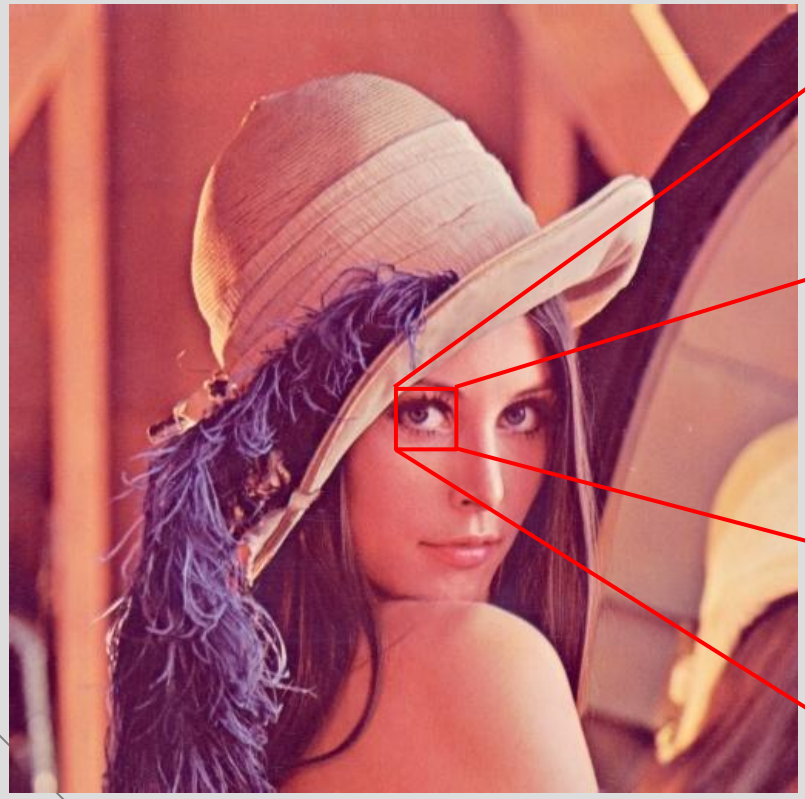
ラスタ画像（ビットマップ画像）

- 微細な点（ピクセル）の集まりを保持する
  - ピクセル毎に色を持っている
  - BMP、JPEG、PNG、GIFなどが一般的

Boost.GIL

画像の構造：概念

画像はピクセルの集まりで構成されている



ピクセル (Pixcel)

RGB色空間での値（例）
Red : 178/255
Green : 94/255
Blue : 114/255
↑ ↑
チャネル (Channel)
ビット深度 (Bit Depth) (Color Depth)

画像の構造:データ構造

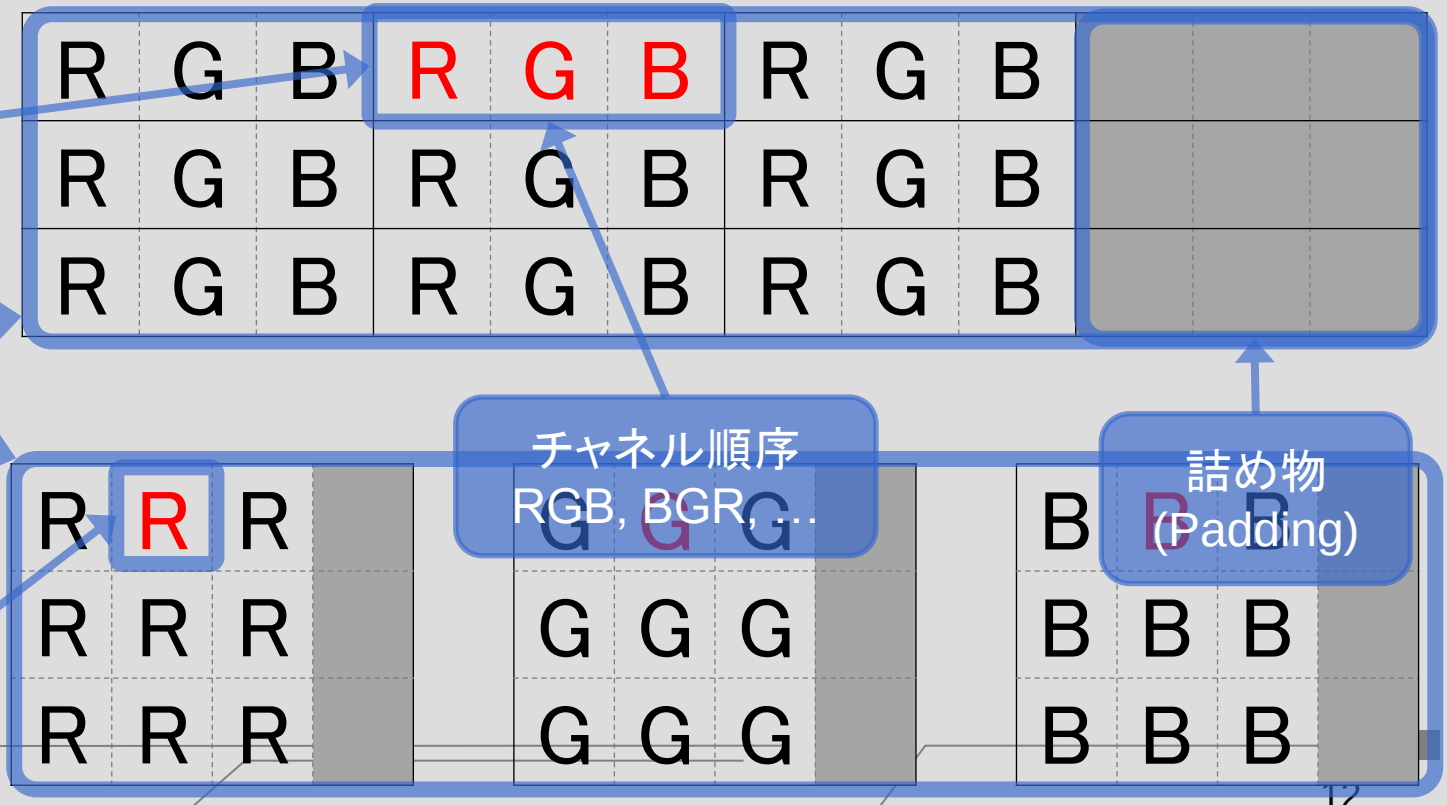
GILは様々なデータ構造の持ち方をサポートする

(0, 1) at 3x3 Image

色空間
RGB, RGBA, CMYK, …

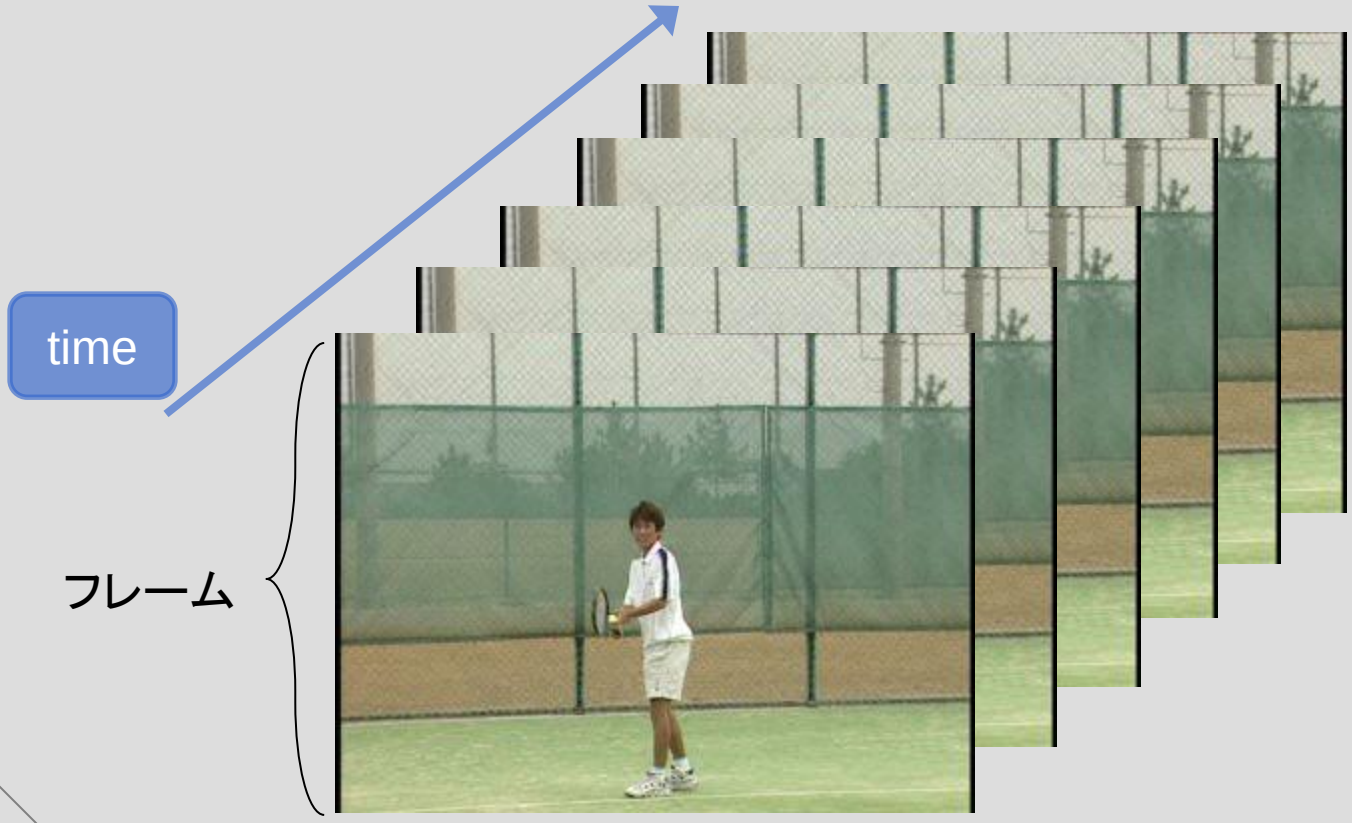
構造
Inverleaved or Planar

ビット深度
8 bits, 16bits, …



画像の構造:データ構造

動画像はパラパラ漫画になっている



2. Boost.GIL基礎の基礎

1. 前提知識
   1. 色空間
   2. 画像の種類
   3. 画像の構造:概念
   4. 画像の構造:データ構造
   5. 動画像の構造
2. Boost.GIL基礎の基礎
   1. 画像の入出力
   2. 画像へのアクセス
   3. ピクセルへのアクセス
   4. チャネルへのアクセス
3. 動体検出プログラムの作成
   1. 動体検出アルゴリズム
   2. Boost.GILでの実装(基礎)
4. おまけ
   1. Boost.GIL参考情報

画像の入出力

GILが標準でサポートする入出力はJPG, PNG, TIFFだけ

```
#include <boost/gil/gil_all.hpp>
#include <boost/gil/extension/io/jpeg_io.hpp>

using namespace boost::gil;

int main() {
    rgb8_image_t img;                                   // 型宣言

    jpeg_read_image("serve-001.jpg", img);      // 入力
    jpeg_write_view( "out.jpg", view( img ) );      // 出力
}
```

画像の入出力

型宣言はColorSpace, BitDepth, ClassType の組み合わせで表現する

rgb8_image_t

<<ColorSpace>>
rgb,
bgr,
cmyk,
rgba,
gray
…

<<BitDepth>>
8,
8s,
16,
16s,
32f,
…

<<ClassType>>
image (image),
view (image view),
loc (locator),
pixcel (pixel value)
ptr (pixel iterator),
…

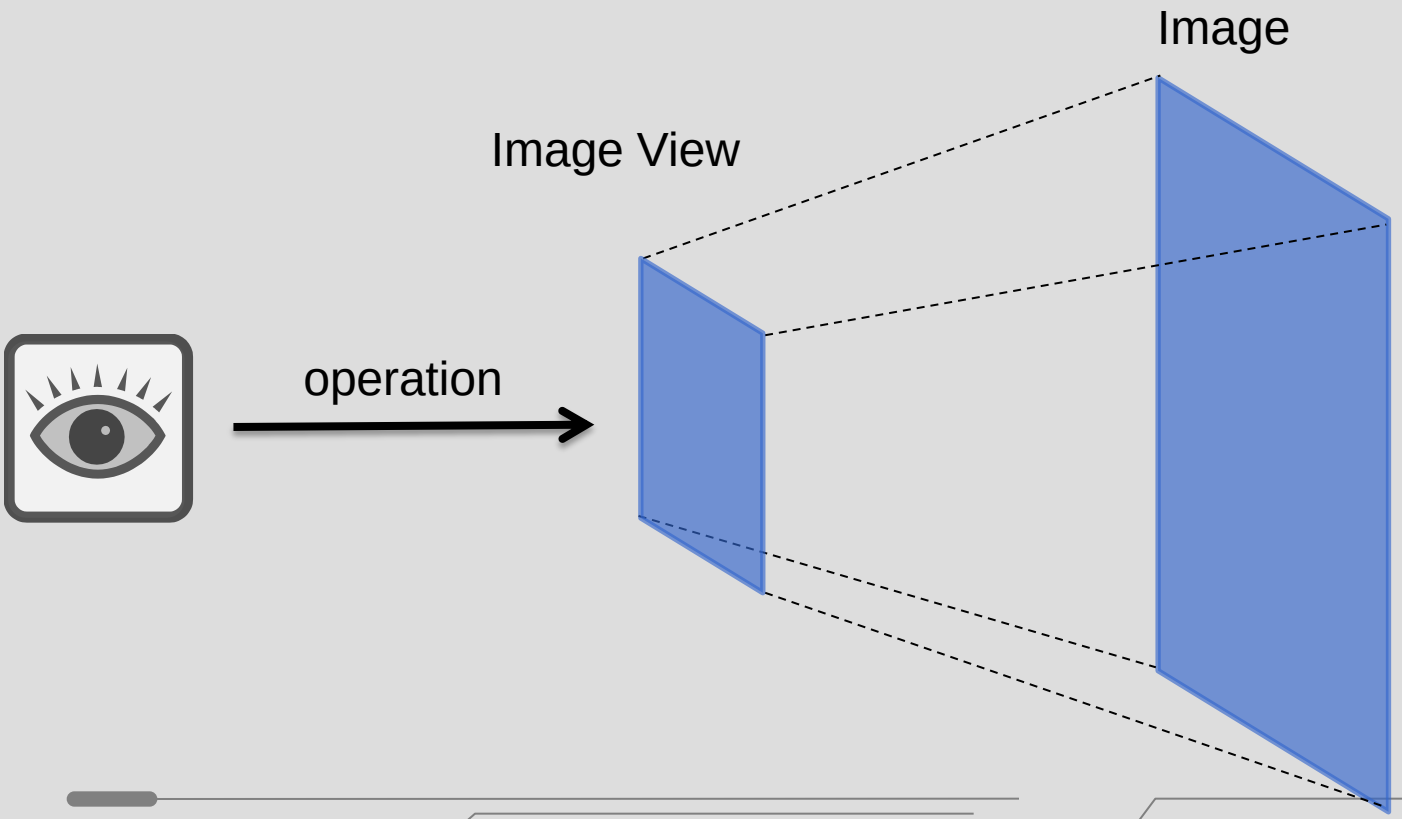
画像へのアクセス

直接Imageは操作しない
画像の操作にはImage Viewを使う

```
void operation(const rgb8_view_t& src_view) {
    // src_viewに対する操作
}

int main() {
    rgb8_image_t img;

    jpeg_read_image("serve-001.jpg", img);

    operation(view(img));

    jpeg_write_view("out.jpg", view(img));
}
```

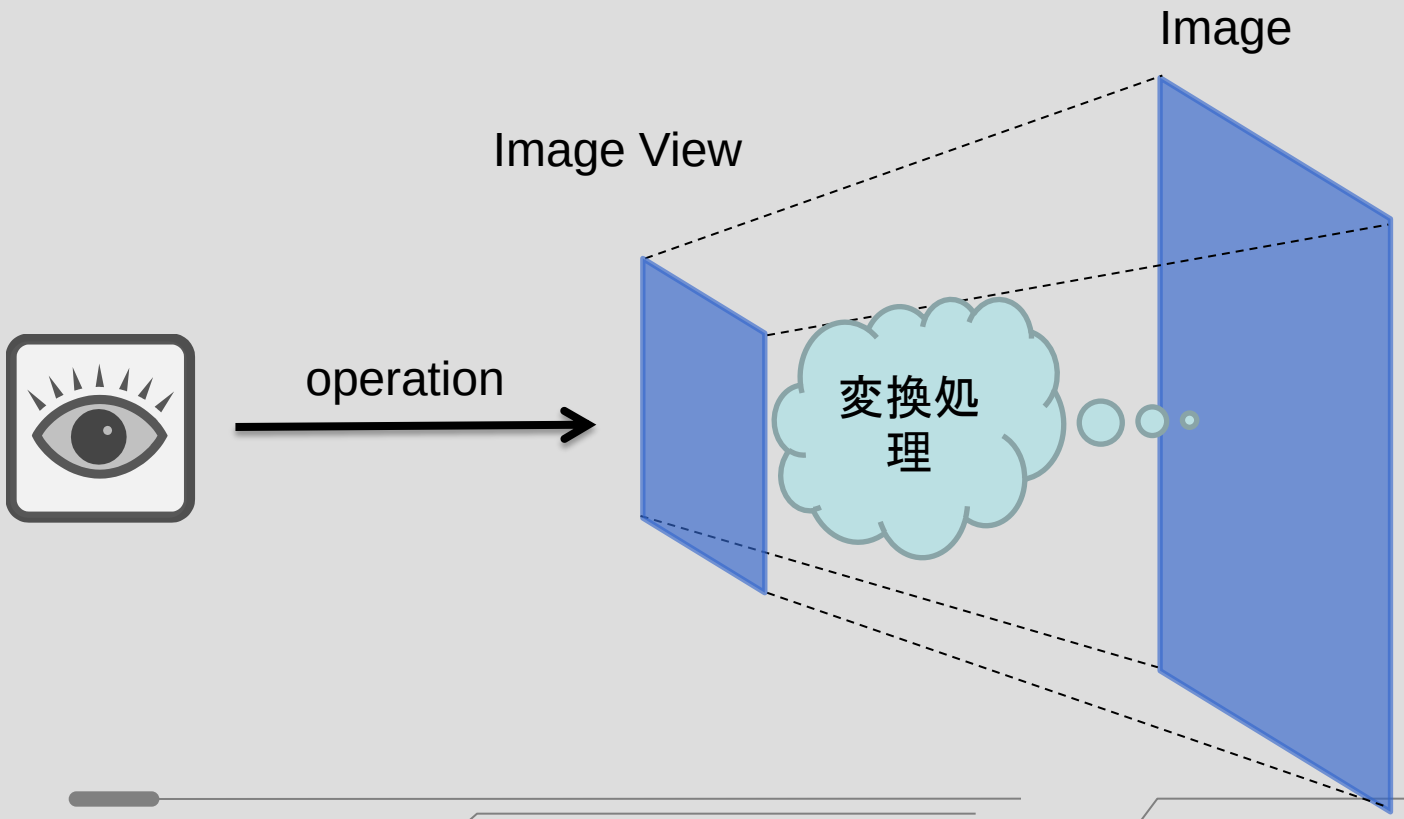
画像へのアクセス

直接Imageは操作しない
画像の操作にはImage Viewを使う



画像へのアクセス

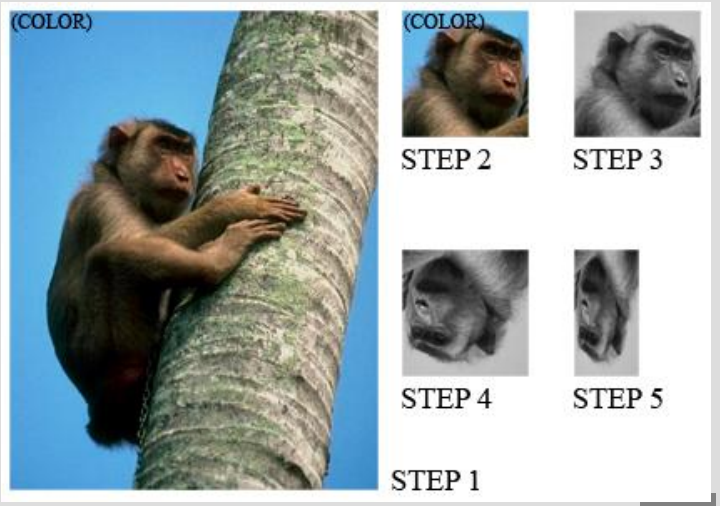
Image Viewで画像の操作範囲、変換処理をかませる



画像へのアクセス

Image Viewで画像の操作範囲、変換処理をかませる

```
jpeg_read_image("monkey.jpg", img);

step1=view(img);
step2=subimage_view(step1, 200,300, 150,150);
step3=color_converted_view<rgb8_view_t,gray8_pixel_t>(step2);
step4=rotated180_view(step3);
step5=subsampled_view(step4, 2,1);

jpeg_write_view("monkey_transform.jpg", step5);
```



ピクセルへのアクセス

Pixelへのアクセスは()演算子、Iterator、Locator等がある

image_view::operator()

iterator

locator

今日はここの一部だけ・・・

ピクセルへのアクセス

()演算子の使い方

```
template <typename View>
void operation(const View& src) {
    typedef typename View::value_type src_pixel_t;

    for (int y=0; y < src.height(); ++y) {
        for (int x=0; x < src.width(); ++x) {
            // (x, y)座標にR=0, G=0, B=0（黒）をセット
            src(x, y) = src_pixel_t(0, 0, 0);
        }
    }
}
```

ピクセルへのアクセス

Iteratorの使い方

```
template <typename View>
void operation(const View& src) {
    typedef typename View::value_type src_pixel_t;

    // 画像全部をR=0, G=0, B=0（黒）で塗りつぶす
    std::fill(src.begin(), src.end(), src_pixel_t(0, 0, 0));
}
```

ピクセルへのアクセス

Iteratorは、画像の左上から右下に向かって捜査する

1
2
3
4
5
6
7
8

チャネルへのアクセス

時間がなくて資料が作れなかったので（次回が）あれば・・・

(人；´Д`) ｺﾞﾒﾝﾅｻｲ！

2. Boost.GIL基礎の基礎

1. 前提知識
   1. 色空間
   2. 画像の種類
   3. 画像の構造:概念
   4. 画像の構造:データ構造
   5. 動画像の構造
2. Boost.GIL基礎の基礎
   1. 画像の入出力
   2. 画像へのアクセス
   3. ピクセルへのアクセス
   4. チャネルへのアクセス
3. 動体検出プログラムの作成
   1. 動体検出アルゴリズム
   2. Boost.GILでの実装(基礎)
4. おまけ
   1. Boost.GIL参考情報

動体検出アルゴリズム

フレーム間の差分（引算）＝動いているところ！

フレーム2

フレーム1

Boost.GILでの実装（基礎）

main()の処理

```
int main() {
    rgb8_image_t img1, img2;

    jpeg_read_image("serve-031.jpg", img1);
    jpeg_read_image("serve-032.jpg", img2);

    gray8_image_t out(img1.width(), img1.height());

    operation(
        color_converted_view<gray8_pixel_t>(view(img1)),
        color_converted_view<gray8_pixel_t>(view(img2)),
        view(out)
    );

    jpeg_write_view("out.jpg", view(out));
}
```

Boost.GILでの実装（基礎）

フレーム間差分の実装部分

```
template <typename SrcView, typename DstView>
void operation(const SrcView& src1, const SrcView& src2, const DstView& dst) {
    for (int y=0; y < src1.height(); ++y) {
        for (int x=0; x < src2.width(); ++x) {
            dst(x, y) = std::abs(src1(x, y) - src2(x, y));
        }
    }
}
```

Boost.GILでの実装（基礎）

実行結果

Boost.GILでの実装（基礎）

実行結果（拡大）

1. 前提知識

1. 前提知識
   1. 色空間
   2. 画像の種類
   3. 画像の構造:概念
   4. 画像の構造:データ構造
   5. 動画像の構造
2. Boost.GIL基礎の基礎
   1. 画像の入出力
   2. 画像へのアクセス
   3. ピクセルへのアクセス
   4. チャネルへのアクセス
3. 動体検出プログラムの作成
   1. 動体検出アルゴリズム
   2. Boost.GILでの実装(基礎)
4. おまけ
   1. Boost.GIL参考情報

Boost.GIL参考情報

Generic Image Library Tutorial 日本語訳
http://sites.google.com/site/twinkleofsilence/

Adobe Generic Image Library (GIL) を使ってみる
http://sssm.sakura.ne.jp/dev/gil.html

↓この人にコンタクトがとれればきっと素晴らしい発表をしてくれるはず・・・
新妻弘崇氏作:
STL like OpenCV wrapper
http://sourceforge.net/projects/stllcv/