第3回C++輪講

第3回の内容

2次元配列

```
#include <iostream>
#include <vector>
namespace name{
namespace t3cl{
    vector2dTest();
}
}
int main()
{
    name::t3cl::vector2dTest();
    return 0;
}
void name::t3cl::vector2dTest()
{
    std::vector<std::vector<int>> vector2d{ { 0, 0, 0 },{ 0, 0, 0 } };
    std::vector<std::vector<int>> initializeExample2(3, std::vector<int>(4,0));
    for(auto && vec : vecor2d)
    {
        for(auto && elem : vec)
        {
            std::cout << elem << std::endl;
        }
    }
}
```

OpenCV

画像，動画の入出力と表示にOpenCVを用いて画像処理への理解を深めよう
下記の画像の性質を理解して画像処理に臨もう

- 画像は1画素にBlue，Green，Redの要素がある
- それぞれ0～255の値が格納されている
- 座標原点は画像左上
- x軸は左上から右上，y軸は左上から左下
  キーワード:「ラスタスキャン」

画像処理ではcv::Matというクラス（これまでの型のようなもの）を扱う
課題を通して扱いに慣れよう

色変換

画像処理ではBGR画像以外にグレースケール画像や二値化画像，HSV変換された画像を用いることがある

grayscale

```
#include <iostream>
#include <cmath>
#include "opencv2/opencv.hpp"
namespace name
{
namespace t3cl
{
    bgr2gray(const cv::Mat3b & src, cv::Mat1b & dst);
}
}
int main()
{
    cv::Mat3b lena = cv::imread("sample_images/lena.bmp", cv::IMREAD_COLOR);
    // read image

    CV_Assert(!src.empty());
    // if (src.empty()) { //こっちでも可
    //     std::cerr << "error: src is empty. (bgr2gray -> color_conversion.cpp)"
<< std::endl;
    //     return 1;
    // }

    cv::Mat1b gray;
    name::t3cl::bgr2gray(lena, gray);

    cv::imshow("result", gray); // display image
    cv::waitKey(0);

    cv::imwrite("result.bmp", gray); //save image
    //ここでjpg保存しないこと！jpgで保存すると圧縮され画素値が変わる場合がある

    cv::destroyAllWindows();
    return 0;
}
```

```
void name::t3cl::bgr2gray(const cv::Mat3b & src, cv::Mat1b & dst)
{
    dst = cv::Mat1b(src.size());
    // dst = cv::Mat1b(src.rows, src.cols); //こっちでも可

    for (int y = 0; y < src.rows; ++y) {
        for (int x = 0; x < src.cols; ++x) {
            dst(y, x) = static_cast<uchar>(std::floor(0.114 * src(y, x)[0] +
0.587 * src(y, x)[1] + 0.299 * src(y, x)[2]));
        }
    }
}
```

動画入出力

```
#include <iostream>
#include "opencv2/opencv.hpp"
namespace name
{
namespace t3cl
{
    void showMovieInfo(const std::string &filepath);
    void playAndSaveMovie(const std::string &input, const std::string &output);
}
}
int main()
{
    std::string videopath = "person01_boxing_d1_uncomp.avi";
    std::string outputpath = "output.avi";
    name::t3cl::showMovieInfo(videopath);
    name::t3cl::playAndSaveMovie(videopath, outputpath);
    cv::destroyAllWindows();
    return 0;
}
void name::t3cl::showMovieInfo(const std::string &filepath) {
    cv::VideoCapture cap(filepath);
    std::cout << "width:\t" << cap.get(cv::CAP_PROP_FRAME_WIDTH)
        << "\nheight:\t" << cap.get(cv::CAP_PROP_FRAME_HEIGHT)
        << "\nfps:\t" << cap.get(cv::CAP_PROP_FPS)
        << "\nmaxFrame:\t" << cap.get(cv::CAP_PROP_FRAME_COUNT)
        << "\nforcc:\t" << cap.get(cv::CAP_PROP_FOURCC) << std::endl;
}
void name::t3cl::playAndSaveMovie(const std::string &input, const std::string
&output) {
    cv::VideoCapture cap(input);
    // 保存するビデオファイルのコーデック設定
    int forcc = static_cast<int>(cap.get(cv::CAP_PROP_FOURCC));
    //int forcc = cv::VideoWriter::fourcc('W', 'M', 'V', '1'); //wmv
    //int forcc = cv::VideoWriter::fourcc('X', 'V', 'I', 'D');  //avi
    // 保存するビデオファイルのFPS設定
    double fps = cap.get(cv::CAP_PROP_FPS);
    // 保存するビデオファイルの動画サイズ設定
```

```
    cv::Size videoSize(static_cast<int>(cap.get(cv::CAP_PROP_FRAME_WIDTH)),
static_cast<int>(cap.get(cv::CAP_PROP_FRAME_HEIGHT)));
    int maxFrame = static_cast<int>(cap.get(cv::CAP_PROP_FRAME_COUNT));
    cv::VideoWriter writer(output, forcc, fps, videoSize);  //ビデオコーデックによ
って保存可能なサイズが決まっており，それ以外だと再生されない
    if (!cap.isOpened()) { CV_Assert(cap.isOpened()); }
    cv::namedWindow(input);
    while (true) {
        cv::Mat frame, outframe;
        cap >> frame;
        if (frame.empty())      { break; }
        std::cout << "\r" << "playing movie\t"
            << cap.get(cv::CAP_PROP_POS_FRAMES)
            << "/" << maxFrame << std::flush;
        cv::imshow(input, frame);
        cv::waitKey(fps);
        outframe = frame.clone()
        writer << outframe;
    }
    std::cout << "\n";
}
```

Matの種類

cv::Mat	1画素のデータサイズ
cv::Mat1b	1byte(uchar)
cv::Mat2b	2byte
cv::Mat3b	3byte
cv::Mat4b	4byte
cv::Mat1s	short
cv::Mat1w	ushort
cv::Mat1i	int
cv::Mat1f	float
cv::Mat1d	double

Matの初期化

```
int width(100), height(100);
cv::Size imSize(width, height);
// 全要素0のMat
cv::Mat1b zero = cv::Mat1b::zeros(width, height);
cv::Mat1b zero2 = cv::Mat1b::zeros(imSize.width, imSize.height);
cv::Mat1b zero3 = cv::Mat1b::zeros(imSize);
```

```
cv::Mat1b zero4 = cv::Mat1b::zeros(cv::Size(width, height));
// 全要素1のMat
cv::Mat1b one = cv::Mat1b::ones(imSize);
// 値を直接指定
cv::Mat1i filter = (cv::Mat1i(3, 3) <<
    -1, 0, 1,
    -2, 0, 2,
    -1, 0, 1);
```

Matの代入

cv::Mat型は代入演算子(=)でコピーすると，コピー元とコピー先でデータが共有される(浅いコピー)
この状態でコピー先のcv::Matに処理を行うと，コピー元も同様の処理が実行される
上記の問題を解決するためにcv::Matのメンバ関数clone()を使用して
コピー元とコピー先を全く異なるデータとしてコピーできる(深いコピー)
基本的には深いコピーしか使わない

```
#include <iostream>
#include "opencv2/opencv.hpp"
namespace name
{
namespace t3cl
{
    void shallowCopyExample(cv::Mat & src, cv::Mat & dst);
    void deepCopyExample(cv::Mat & src, cv::Mat & dst);
}
}
int main()
{
    const std::string fileName("sample_images/lena.bmp");   //このような初期化も可
    cv::Mat3b src(cv::imread(fileName)), shallowCopied, deepCopied; //このような初
期化も可
    CV_Assert(!src.empty());
    name::t3cl::shallowCopyExample(src, shallowCopied);
    name::t3cl::deepCopyExample(src, deepCopied);
    cv::imshow("src", src);
    cv::imshow("shallowCopied", shallowCopied);
    cv::imshow("deepCopied", deepCopied);
    cv::waitKey();
    return 0;
}
// 浅いコピー
void name::t3cl::shallowCopyExample(cv::Mat & src, cv::Mat & dst){
    dst = src;  //浅いコピー
    //dstの中心に半径40pxの赤く塗りつぶされた円を書く
    cv::circle(dst, dst.size() / 2, 40, cv::Scalar(0, 0, 255), -1);
}
// 深いコピー
void name::t3cl::deepCopyExample(cv::Mat & src, cv::Mat & dst){
    dst = src.clone();  //深いコピー
    cv::circle(dst, dst.size() / 2, 40, cv::Scalar(0, 0, 255), -1);
}
```

関数のオーバーロード

C++では以下の条件を満たす場合，関数を複数定義できる

- 引数の個数が異なる
- 引数の型が異なる

戻り値の型のみが異なる関数はオーバーロードできない

```
int max(int a, int b, int c);               //ok
double max(double a, double b, double c);   //ok
double max(double a, double b);             //ok
float max(double a, double b);              //戻り値のみが異なるため×
```

第3回課題

the3rdCppLecture_name.hpp，the3rdCppLecture_name.cppを作成し以下の関数を名前空間name，t3cl内に作成せよ
関数run()で全ての関数が実行できるようにすること
画像処理に使う画像は，sample_imagesフォルダを各自のプロジェクトフォルダにコピーして使用すること

- 二値化
  関数名：binarization
- コントラスト強調
  関数名：contrastStretching
- HSV変換
  関数名：bgr2hsv
  以下の関数を用いて確認せよ

```
void name::t3cl::confirmHSV(const cv::Mat &orgBGR, cv::Mat &yourHSV)
{
    auto _orgBGR = orgBGR.clone();
    cv::Mat orgHSV;
    cv::cvtColor(_orgBGR, orgHSV, cv::COLOR_BGR2HSV);
    std::vector<cv::Mat> separatedOrg(3), separatedHSV(3);
    cv::split(orgHSV, separatedOrg);
    cv::split(yourHSV, separatedHSV);
    int windowX = 100;
    int windowY = 100;
    for (int i = 0; i < 3; ++i)
    {
        cv::imshow("orgHSV channel " + std::to_string(i), separatedOrg[i]);
        cv::imshow("yourHSV channel " + std::to_string(i), separatedHSV[i]);
        cv::moveWindow("orgHSV channel " + std::to_string(i), windowX,
windowY);
        cv::moveWindow("yourHSV channel " + std::to_string(i), windowX,
```

```
        windowY + _orgBGR.rows);
            windowX += _orgBGR.cols;
        }
        cv::waitKey();
        cv::destroyAllWindows();
    }
```

- 画像端のゼロ埋め処理（パディング幅可変，関数のオーバーロード）
  （原画像より一回り大きい画像を作成し，内側を原画像、外側をゼロで埋める）
  関数名：zeroPadding(const cv::Mat3b &src, cv::Mat3b &dst, int paddingWidth)
  関数名：zeroPadding(const cv::Mat1b &src, cv::Mat1b &dst, int paddingWidth)
- 画像端のコピー埋め処理（パディング幅可変，関数のオーバーロード）
  （原画像より一回り大きい画像を作成し，内側を原画像、外側を原画像の端の画素で埋める）
  関数名：copyPadding(const cv::Mat3b &src, cv::Mat3b &dst, int paddingWidth)
  関数名：copyPadding(const cv::Mat1b &src, cv::Mat1b &dst, int paddingWidth)
- 膨張処理（4近傍，8近傍）
  関数名：dilate
- 収縮処理（4近傍，8近傍）
  関数名：erode
- ソーベルフィルタ
  関数名：sobel（関数内でパディング）
- プリューウィットフィルタ（関数内でパディング）
  関数名：prewitt
- メディアンフィルタ（関数内でパディング）
  ソートは第2回のヘッダをインクルードして使用すること
  関数名：median
- ガウシアンフィルタ（フィルタサイズ可変，関数内でパディング）
  関数名：gaussian

```
void name::t3cl::imshow(const std::string &path, cv::Mat &image)
{
    cv::imshow(path, image);
    cv::moveWindow(path, 100, 100);
    cv::waitKey();
    cv::destroyAllWindows();
}
// void run(const int num = 0, const std::string & imagePath =
"sample_images/lena.bmp"); //ヘッダではこのように宣言すること
void name::t3cl::run(const int num, const std::string & imagePath)
{
    const cv::Mat3b lena(cv::imread(imagePath, cv::IMREAD_COLOR));
    CV_Assert(!lena.empty());

    cv::Mat1b gray, binary;
    t3cl::bgr2gray(lena, gray);

    switch (num) {
    default:
    case 1:
```

```
        std::cout << "run binarization";
        {
            // src dst th
            t3cl::binarization(gray, binary, 120);
            t3cl::imshow("binary", binary);
        }
            std::cout << "\r" << "complete binarization" << std::endl;
        if (num != 0) break;
    case 2:
        std::cout << "run contrastStretching";
        {
            cv::Mat1b stretched;
            t3cl::contrastStretching(gray, stretched);
            t3cl::imshow("stretched", stretched);
        }
            std::cout << "\r" << "complete contrastStretching" << std::endl;
        if (num != 0) break;
    case 3:
        std::cout << "run bgr2hsv";
        {
            cv::Mat3b hsv;
            t3cl::bgr2hsv(lena, hsv);
            t3cl::confirmHSV(lena, hsv);
        }
            std::cout << "\r" << "complete bgr2hsv" << std::endl;
        if (num != 0) break;
    case 4:
        std::cout << "run zeroPadding";
        {
            cv::Mat1b zeroPadded;
            t3cl::zeroPadding(gray, zeroPadded, 5);
            t3cl::imshow("zeroPadded", zeroPadded);
        }
            std::cout << "\r" << "complete zeroPadding" << std::endl;
        if (num != 0) break;
    case 5:
        std::cout << "run copyPadding";
        {
            cv::Mat1b copyPadded;
            t3cl::copyPadding(gray, copyPadded, 5);
            t3cl::imshow("copyPadded", copyPadded);
        }
            std::cout << "\r" << "complete copyPadding" << std::endl;
        if (num != 0) break;
    case 6:
        std::cout << "run dilate";
        {
            cv::Mat1b dilated4, dilated8;
            t3cl::binarization(gray, binary, 120);
            // src dst th
            t3cl::dilate(binary, dilated4, 4);
            cv::imshow("dilated 4", dilated4);
            cv::moveWindow("dilated 4", 100 + dilated4.cols, 100);
            t3cl::dilate(binary, dilated8, 8);
            t3cl::imshow("dilated 8", dilated8);
```

```
        }
            std::cout << "\r" << "complete dilate" << std::endl;
        if (num != 0) break;
    case 7:
        std::cout << "run erode";
        {
            cv::Mat1b eroded4, eroded8;
            t3cl::binarization(gray, binary, 120);
            t3cl::erode(binary, eroded4, 4);
            cv::imshow("eroded 4", eroded4);
            cv::moveWindow("eroded 4", 100 + eroded4.cols, 100);
            t3cl::erode(binary, eroded8, 8);
            t3cl::imshow("eroded 8", eroded8);
        }
            std::cout << "\r" << "complete erode" << std::endl;
        if (num != 0) break;
    case 8:
        std::cout << "run sobel";
        {
            cv::Mat1b sobelVer, sobelHori;
            t3cl::sobel(gray, sobelVer, true);
            cv::imshow("sobelVer", sobelVer);
            cv::moveWindow("sobelVer", 100 + sobelVer.cols, 100);
            t3cl::sobel(gray, sobelHori, false);
            t3cl::imshow("sobelHori", sobelHori);
        }
            std::cout << "\r" << "complete sobel" << std::endl;
        if (num != 0) break;
    case 9:
        std::cout << "run prewitt";
        {
            cv::Mat1b prewittVer, prewittHori;
            t3cl::prewitt(gray, prewittVer, true);
            cv::imshow("prewittVer", prewittVer);
            cv::moveWindow("prewittVer", 100 + prewittVer.cols, 100);
            t3cl::prewitt(gray, prewittHori, false);
            t3cl::imshow("prewittHori", prewittHori);
        }
            std::cout << "\r" << "complete prewitt" << std::endl;
        if (num != 0) break;
    case 10:
        std::cout << "run median";
        {
            cv::Mat1b medianImg;
            t3cl::median(gray, medianImg);
            t3cl::imshow("median", medianImg);
        }
            std::cout << "\r" << "complete median" << std::endl;
        if (num != 0) break;
    case 11:
        std::cout << "run gaussian";
        {
            cv::Mat1b gaussianImg;
            // src dst kernelsize variance
            t3cl::gaussian(gray, gaussianImg, 5, 2);
```

```
            t3cl::imshow("gaussian", gaussianImg);
        }
            std::cout << "\r" << "complete gaussian" << std::endl;
        if (num != 0) break;
    }
}
```