

OpenCV 2.1 チートシート (C++)

OpenCV C++ リファレンスマニュアルは、こちら:
<http://opencv.jp/opencv-2svn/cpp/> (Original :
<http://opencv.willowgarage.com/documentation/cpp/>). 関数や
クラスの説明を見つけるには [クイック検索](#) を利用してください

OpenCV の重要なクラス

Point_	2次元点クラステンプレート
Point3_	3次元点クラステンプレート
Size_	サイズ(幅,高さ)クラステンプレート
Vec	ショートベクトルクラステンプレート
Scalar	4要素ベクトル
Rect	矩形
Range	整数値で表現される範囲
Mat	(行列や画像として用いられる)2次元の密な配列
MatND	多次元の密な配列
SparseMat	多次元の疎な配列
Ptr	スマートポインタのクラステンプレート

行列の基礎

行列の作成

```
Mat image(240, 320, CV_8UC3);
```

宣言済み行列の(再)配置

```
image.create(480, 640, CV_8UC3);
```

行列を定数で初期化

```
Mat A33(3, 3, CV_32F, Scalar(5));  
Mat B33(3, 3, CV_32F); B33 = Scalar(5);  
Mat C33 = Mat::ones(3, 3, CV_32F)*5.;  
Mat D33 = Mat::zeros(3, 3, CV_32F) + 5.;
```

行列を特定の値で初期化

```
double a = CV_PI/3;  
Mat A22 = (Mat_<float>(2, 2) <<  
cos(a), -sin(a), sin(a), cos(a));  
float B22data[] = {cos(a), -sin(a), sin(a), cos(a)};  
Mat B22 = Mat(2, 2, CV_32F, B22data).clone();
```

行列を乱数で初期化

```
randu(image, Scalar(0), Scalar(256)); // 一様分布  
randn(image, Scalar(128), Scalar(10)); // 正規分布
```

行列と他の構造体を互いに変換
(データコピーなし)

```
Mat image.alias = image;  
float* Idata=new float[480*640*3];  
Mat I(480, 640, CV_32FC3, Idata);  
vector<Point> iptvec(10);  
Mat iP(iptvec); // iP - 10x1 CV_32SC2 matrix  
IplImage* oldC0 = cvCreateImage(cvSize(320,240),16,1);  
Mat newC = cvarrToMat(oldC0);  
IplImage oldC1 = newC; CvMat oldC2 = newC;
```

... (データコピーあり)

```
Mat newC2 = cvarrToMat(oldC0).clone();  
vector<Point2f> ptvec = Mat_<Point2f>(iP);
```

行列の要素にアクセス

```
A33.at<float>(i,j) = A33.at<float>(j,i)+1;  
Mat dyImage(image.size(), image.type());
```

```
for(int y = 1; y < image.rows-1; y++) {  
Vec3b* prevRow = image.ptr<Vec3b>(y-1);  
Vec3b* nextRow = image.ptr<Vec3b>(y+1);  
for(int x = 0; x < image.cols; x++)  
for(int c = 0; c < 3; c++)  
dyImage.at<Vec3b>(y,x)[c] =  
saturate_cast<uchar>(  
nextRow[x][c] - prevRow[x][c]);  
}  
Mat_<Vec3b>::iterator it = image.begin<Vec3b>(),  
itEnd = image.end<Vec3b>();  
for(; it != itEnd; ++it)  
(*it)[1] ^= 255;
```

行列操作: コピー, シャッフル, 部分行列

src.copyTo(dst)	別の行列へのコピー
src.convertTo(dst, type, scale, shift)	スケーリングと別のデータ型 への変換
m.clone()	行列の深いコピー
m.reshape(nch, nrows)	データをコピーせずに, 行列の次元, チャンネル 数を変更
m.row(i), m.col(i)	行列の行と列
m.rowRange(Range(i1, i2))	行列の行と列 (範囲指定)
m.colRange(Range(j1, j2))	
m.diag(i)	行列の対角要素
m(Range(i1, i2), Range(j1, j2))	部分行列
m(roi)	
m.repeat(ny, nx)	小さい行列から大きな行列を作成
flip(src, dst, dir)	行列の行 (または列) の順序を反転
split(...)	マルチチャンネル行列を各チャンネル毎に分離
merge(...)	各チャンネル毎の行列からマルチチャンネル行列 を作成
mixChannels(...)	split() と merge() の一般型
randShuffle(...)	行列の要素をランダムにシャッフル

例1. 画像 ROI を平滑化します

```
Mat imgroi = image(Rect(10, 20, 100, 100));  
GaussianBlur(imgroi, imgroi, Size(5, 5), 1.2, 1.2);
```

例2. 線形代数アルゴリズム

```
m.row(i) += m.row(j)*alpha;  
例3. 画像 ROI を別の画像に, 変換しながらコピーします  
Rect r(1, 1, 10, 20);  
Mat dstroi = dst(Rect(0,10,r.width,r.height));  
src(r).convertTo(dstroi, dstroi.type(), 1, 0);
```

シンプルな行列操作

OpenCV には多くの一般的な, 行列の算術演算, 論理演算, その他の処理
が実装されています. 例えば,

- [add\(\)](#), [subtract\(\)](#), [multiply\(\)](#), [divide\(\)](#), [absdiff\(\)](#),
[bitwise.and\(\)](#), [bitwise.or\(\)](#), [bitwise.xor\(\)](#), [max\(\)](#), [min\(\)](#),
[compare\(\)](#)
- 和や差, 要素毎の掛け算 ... 二つの行列の比較や行列とスカラーの
比較, などに相当する演算.

例. アルファ合成 関数:

```
void alphaCompose(const Mat& rgba1,  
const Mat& rgba2, Mat& rgba_dest)  
{  
Mat a1(rgba1.size(), rgba1.type()), ra1;  
Mat a2(rgba2.size(), rgba2.type());  
int mixch[]={3, 0, 3, 1, 3, 2, 3, 3};  
mixChannels(&rgba1, 1, &a1, 1, mixch, 4);  
mixChannels(&rgba2, 1, &a2, 1, mixch, 4);  
subtract(Scalar::all(255), a1, ra1);  
bitwise_or(a1, Scalar(0,0,0,255), a1);  
bitwise_or(a2, Scalar(0,0,0,255), a2);  
multiply(a2, ra1, a2, 1./255);  
multiply(a1, rgba1, a1, 1./255);  
multiply(a2, rgba2, a2, 1./255);  
add(a1, a2, rgba_dest);  
}
```

- [sum\(\)](#), [mean\(\)](#), [meanStdDev\(\)](#), [norm\(\)](#), [countNonZero\(\)](#),
[minMaxLoc\(\)](#),
- 行列要素の様々な統計量.
- [exp\(\)](#), [log\(\)](#), [pow\(\)](#), [sqrt\(\)](#), [cartToPolar\(\)](#), [polarToCart\(\)](#)
- 古典的な算術関数.
- [scaleAdd\(\)](#), [transpose\(\)](#), [gemm\(\)](#), [invert\(\)](#), [solve\(\)](#),
[determinant\(\)](#), [trace\(\)](#) [eigen\(\)](#), [SVD](#),
- 代数関数 + SVD クラス.
- [dft\(\)](#), [idft\(\)](#), [dct\(\)](#), [idct\(\)](#),
- 離散フーリエ変換, 離散コサイン変換

さらに便利な [代数表記](#) が利用できる処理もあります. 例えば:

```
Mat delta = (J.t()*J + lambda*  
Mat::eye(J.cols, J.cols, J.type()))  
.inv(CV_SVD)*(J.t()*err);
```

これは, Levenberg-Marquardt 最適化アルゴリズムの要です.

画像処理

フィルタリング

filter2D()	非分離型線形フィルタ
sepFilter2D()	分離型線形フィルタ
boxFilter()	線形または非線形のフィルタを用いた画像の平滑 化
GaussianBlur()	
medianBlur()	
bilateralFilter()	
Sobel() , Scharr()	微分画像の計算
Laplacian()	ラプラシアン計算: $\Delta I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$
erode() , dilate()	画像の収縮膨張

例. 3x3 のハイパスフィルタを適用します

(全体に 128 足して, 負の値が失われないようにします):
`filter2D(image, image, image.depth(), (Mat_<float>(3,3)<<
-1, -1, -1, -1, 9, -1, -1, -1, -1), Point(1,1), 128);`

幾何学変換

resize()	画像のリサイズ
getRectSubPix()	画像パッチの抽出
warpAffine()	画像のアフィン変換
warpPerspective()	画像の透視変換
remap()	汎用的な画像変換
convertMaps()	remap() を高速に実行するためのマップ最適化

例. $\sqrt{2}$ 分の一に画像を縮小します:

```
Mat dst; resize(src, dst, Size(), 1./sqrt(2), 1./sqrt(2));
```

様々な画像変換

cvtColor()	画像を別の色空間に変換
threshold() ,	固定あるいは可変の閾値を用いて、グレースケール画像を 2 値画像に変換
adaptivethreshold()	領域拡張アルゴリズムを用いた、連結成分の検出
floodFill()	インテグラルイメージの計算
integral()	2 値画像に対する、距離画像またはポロノイ図の構築
distanceTransform()	2 値画像に対する、距離画像またはポロノイ図の構築
watershed() ,	マーカベースの画像セグメンテーションアルゴリズム watershed.cpp や grabcut.cpp のサンプルを参照してください。
grabCut()	

ヒストグラム

calcHist()	画像のヒストグラムの計算
calcBackProject()	ヒストグラムの逆投影
equalizeHist()	画像の明るさとコントラストの正規化
compareHist()	2 つのヒストグラムの比較

例. 画像の色相-彩度ヒストグラムを計算します:

```
Mat hsv, H; MatND tempH;
cvtColor(image, hsv, CV_BGR2HSV);
int planes[]={0, 1}, hsize[] = {32, 32};
calcHist(&hsv, 1, planes, Mat(), tempH, 2, hsize, 0);
H = tempH;
```

輪郭

輪郭の意味や使い方については [contours.cpp](#) や [squares.c](#) のサンプルを参照してください。

データの入出力

[XML/YAML ストレージ](#) は (入れ子可能な) コレクションで、スカラ値、構造体、様々なリストを含むことができます。

```
YAML(または XML) へのデータの書き込み
// ファイルの種類は拡張子によって決められます
FileStorage fs("test.yml", FileStorage::WRITE);
fs << "i" << 5 << "r" << 3.1 << "str" << "ABCDEFGH";
fs << "mtx" << Mat::eye(3,3,CV_32F);
fs << "mylist" << "[" << CV_PI << "1+1" <<
    "{" << "month" << 12 << "day" << 31 << "year"
```

```
<< 1969 << "]" << "]";
fs << "mystruct" << "{" << "x" << 1 << "y" << 2 <<
    "width" << 100 << "height" << 200 << "lbp" << "[:";
const uchar arr[] = {0, 1, 1, 0, 1, 1, 0, 1};
fs.writeRaw("u", arr, (int)(sizeof(arr)/sizeof(arr[0])));
fs << "]" << "];";
```

<< 演算子を用いて、スカラ値 (整数, 浮動小数点数, 文字列), 行列, スカラ値やその他の型の *STL vector* をファイルストレージに書き込むことができます

```
データの読み出し
// ファイルの種類は内容によって決められます
FileStorage fs("test.yml", FileStorage::READ);
int i1 = (int)fs["i"]; double r1 = (double)fs["r"];
string str1 = (string)fs["str"];
Mat M; fs["mtx"] >> M;
FileNode tl = fs["mylist"];
CV_Assert(tl.type() == FileNode::SEQ && tl.size() == 3);
double tl0 = (double)tl[0]; string tl1 = (string)tl[1];
int m = (int)tl[2]["month"], d = (int)tl[2]["day"];
int year = (int)tl[2]["year"];
FileNode tm = fs["mystruct"];
Rect r; r.x = (int)tm["x"], r.y = (int)tm["y"];
r.width = (int)tm["width"], r.height = (int)tm["height"];
int lbp_val = 0;
FileNodeIterator it = tm["lbp"].begin();
for(int k = 0; k < 8; k++, ++it)
    lbp_val |= ((int)*it) << k;
```

FileNode のキャスト演算子を用いて、スカラ値を読み込みます。行列やその他の型は、>> 演算子を用いて読み込みます。リストを読み込む場合は、*FileNodeIterator* を利用します。

ラスタ画像の書き込みと読み込み
[imwrite\("myimage.jpg", image\);](#)
Mat image_color_copy = [imread\("myimage.jpg", 1\);](#)
Mat image_grayscale_copy = [imread\("myimage.jpg", 0\);](#)

これらの関数が読み書き可能なフォーマット: *BMP (.bmp)*, *JPEG (.jpg, .jpeg)*, *TIFF (.tif, .tiff)*, *PNG (.png)*, *PBM/PGM/PPM (.p?m)*, *Sun Raster (.sr)*, *JPEG 2000 (.jp2)*. 各フォーマットは、8 ビット、1- または 3-チャンネルの画像をサポートします。1 チャンネルが 16 ビットの画像をサポートするフォーマット (*PNG, JPEG 2000*) もあります。

```
動画ファイルやカメラから画像を読み込みます
VideoCapture cap;
if(argc > 1) cap.open(string(argv[1])); else cap.open(0);
Mat frame; namedWindow("video", 1);
for(;;) {
    cap >> frame; if(!frame.data) break;
    imshow("video", frame); if(waitKey(30) >= 0) break;
}
```

シンプル GUI(highgui モジュール)

[namedWindow\(winname, flags\)](#) 名前付き highgui ウィンドウを作成します
[destroyWindow\(winname\)](#) 指定したウィンドウを破棄します
[imshow\(winname, mtx\)](#) ウィンドウ内に画像を表示します
[waitKey\(delay\)](#) 指定時間だけ (あるいは永遠に) キーが押されるのを待ちます。待ち時間にイベントの処理を行います。この関数を、1 秒間に数回程度呼び出すのを忘れないように。
[createTrackbar\(...\)](#) 指定したウィンドウにトラックバー (スライダー) を追加します。
[setMouseCallback\(...\)](#) 指定したウィンドウ内でのマウスのクリックと移動に対して、コールバックを設定します。

GUI 関数の使い方については、[camshiftdemo.c](#) や [OpenCV samples](#) を参照してください。

カメラキャリブレーション, 姿勢推定, 奥行き推定

[calibrateCamera\(\)](#) キャリブレーションパターンを写した複数枚の画像を用いて、カメラをキャリブレーションします。
[findChessboardCorners\(\)](#) チェッカーボードキャリブレーションパターン上の特徴点を検出します。
[solvePnP\(\)](#) その特徴点を投影した結果から、元の物体の姿勢を求めます。
[stereoCalibrate\(\)](#) ステレオカメラをキャリブレーションします。
[stereoRectify\(\)](#) キャリブレーションされたステレオカメラ画像の平行化を行います。
[initUndistortRectifyMap\(\)](#) ステレオカメラの各カメラに対する、(remap() 用の) 平行化マップを計算します。
[StereoBM, StereoSGBM](#) 平行化されたステレオペア上で実行される、ステレオ対応点探索エンジンです。
[reprojectImageTo3D\(\)](#) 視差マップを 3 次元点群に変換します。
[findHomography\(\)](#) 2 次元点集合同間の最適な透視変換を求めます。

カメラをキャリブレーションするには、サンプル [calibration.cpp](#) や [stereo_calib.cpp](#) を利用できます。視差マップや 3 次元点群を得るには、サンプル [stereo_match.cpp](#) を利用してください。

物体検出

[matchTemplate](#) 入力テンプレートに対する一致度マップを求めます。
[CascadeClassifier](#) Viola が提唱した Haar や LBP 特徴量を利用するブースティング分類器のカスケードです。[facedetect.cpp](#) を参照してください。
[HOGDescriptor](#) N. Dalal が提唱した HOG 特徴量を利用する物体検出器です。[peopledetect.cpp](#) を参照してください。