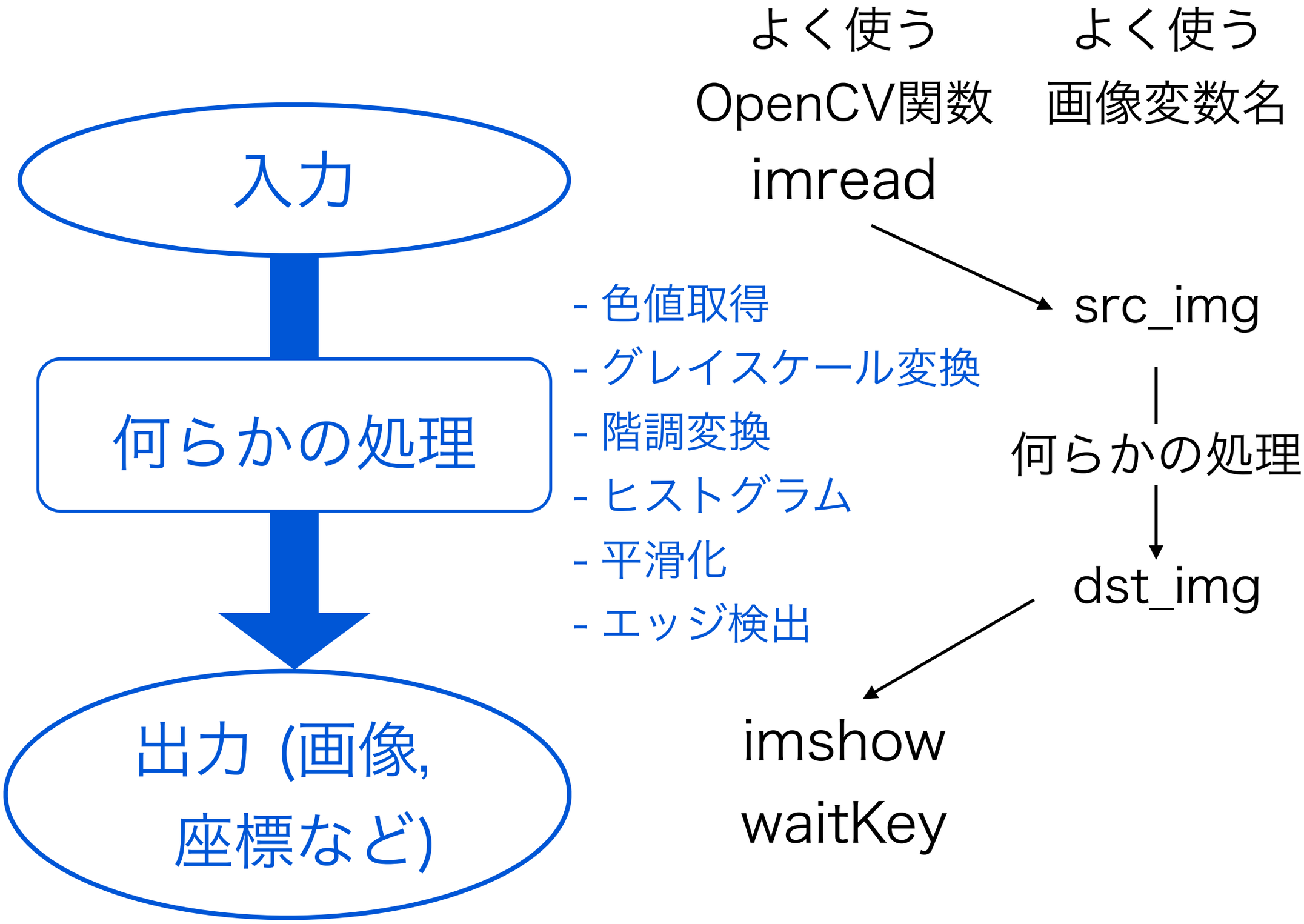
# 画像処理の基本的な流れ



# cv::Mat

- **画像を扱う型　（行列形式）**
- 以下，主要な要素

```
cv::Mat src_img;
```

  - rows: 高さ（行数）
    - src_img.rows または src_img.size().height
  - cols: 幅（列数）
    - src_img.cols またはノート src_img.size().width
  - Channels: チャンネル数

  ➡ **R**, **G**, **B**

教科書p.17

# 画像の読み込み

- imread関数
  - 画像を読み込む関数
  - 画像が存在しない場合: NULLを返す
    →Mat::data==NULL もしくはempty関数で判定

```
cv::Mat src_img;
src_img = cv::imread(ファイル名);


if (src_img.empty()) {
    fprintf(stderr, "読み込み失敗\n");
    return (-1);
}
```

教科書p.18

# 画像の表示

- imshow関数
- 画像の表示

```
cv::imshow(ウィンドウ名, 画像の変数);
```

- 例

```
cv::imshow(WINDOW_NAME, dst_img);
```

  - WINDOW_NAMEをマクロで定義した想定

教科書p.18

# キー入力待ち

- waitKey関数
  - キーイベントを待つ

```
cv::waitKey(delay); //キー入力待ち
```

  - delay　待つ時間（ms）
  - 省略すると（=0）ずっと待ち続ける

教科書p.18

# 色値の取得方法

```
cv::Vec3b val = src_img.at<cv::Vec3b>(y, x);
```

- cv::Vec3b型
  - unsigned char (uchar) 型の3要素を持つベクトル
    - ➡unsigned char: 1バイト (256段階)
- 取得関数：at
  - at<変数の型>(y座標値, x座標値)
    - ➡R: val[2]
    - ➡G: val[1]　に格納される（OpenCVではBGRの順番）
    - ➡B: val[0]

教科書p.17

# 典型的な画像走査

```
//出力画像のメモリ確保
cv::Mat dst_img = cv::Mat(src_img.size(), CV_8UC1);

//画像の走査
for (int y=0; y<src_img.rows; y++) {
    for (int x=0; x<src_img.cols; x++) {
        //画素値の取得
        cv::Vec3b s = src_img.at<cv::Vec3b>(y, x);
        uchar val = 0.114 * s[0] //B
        + 0.587 * s[1] //G
        + 0.299 * s[2];//R
        dst_img.at<uchar>(y, x) = val;
    }
}
```

グレースケール変換の例

# 色変換関数を用いたグレースケール変換

```
//グレースケール化（カラー => グレー）
cv::cvtColor(src_img, dst_img, cv::COLOR_BGR2GRAY);
```

- 引数: 入力画像, 出力画像, コード
- コード (変換命令)
  - **cv::COLOR_BGR2GRAY**: BGR画像からグレースケール

教科書p.28

# 空間フィルタリング (その1)

- フィルタの設計 (以下はOpenCVの仕様)

```
//3. フィルタの宣言と設計（入力と正規化）
//フィルタa
double filter_h[] = { 3., 3., 1.,
                     -1., 2., 1.,
                      0., 1., 1.};
//配列をフィルタ行列kernelに変換
cv::Mat kernel = cv::Mat(3, 3, CV_32F, filter_h);
//正規化 (正規化しないと画素値が255を超えるため)
cv::normalize(kernel, kernel, 1.0, 0.0, cv::NORM_L1);
```

# 空間フィルタリング (その2)

```
//4. フィルタの計算
cv::filter2D(src_img, dst_img, -1, kernel);
```

- cv::filter2D (入力画像，出力画像，depth, カーネル)
  - depthに−1を入れると，入力画像と出力画像のビット数を合わせることになる
  - カーネル: 行列

# ガウシアンフィルタ関数

- ガウシアンフィルタ処理を行うOpenCV関数

```
cv::GaussianBlur(入力画像, 出力画像,
                 cv::Size(横サイズ, 縦サイズ), σ);
```

- サイズ: フィルタサイズ
- σ: ガウシアンフィルタの標準偏差（広がり）
  - σの引数を0に設定すると，フィルタサイズから最適値を自動的に計算する（OpenCVの仕様）

# 空間フィルタリングを用いたエッジ検出

```
//3. フィルタの宣言と設計（入力と正規化）
//フィルタh（微分フィルタ（横方向）の場合）
double filter_h[] = { 0., 0., 0.,
                     -1., 0., 1.,
                      0., 0., 0.};

////配列をフィルタ行列kernelに変換
cv::Mat kernel = cv::Mat(3, 3, CV_64F, filter_h);

//4. フィルタの計算
cv::Mat tmp_img;     // 一時的に格納する画像を用意

cv::filter2D(src_img, tmp_img, CV_64F, kernel);

cv::convertScaleAbs(tmp_img, dst_img);
```

修正

修正

修正

追加

# convertScaleAbs関数

- 絶対値をとり適切な倍率で値を変換するOpenCV関数
  - 適切な表示のため

    ➡絶対値を取る

    ➡適切な倍率で値を強調する

```
cv::convertScaleAbs(入力画像, 出力画像);
```

  - 前ページの場合：倍精度で定義されたtmp_img（入力画像）が 8bit unsigned のdst_img（グレースケールの出力画像）に変換される

# ソーベルフィルタ関数

(教科書p.53)

```
cv::Sobel(入力画像, 出力画像, 深さ, 横, 縦);
```

- 横方向なら

```
cv::Sobel(src_img, dst_img, CV_32F, 1, 0);
```

この場合dst_imgの型がfloatとなる

- 縦方向なら

```
cv::Sobel(src_img, dst_img, CV_32F, 0, 1);
```