

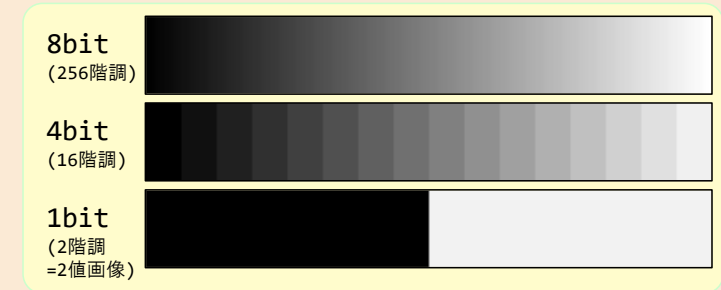
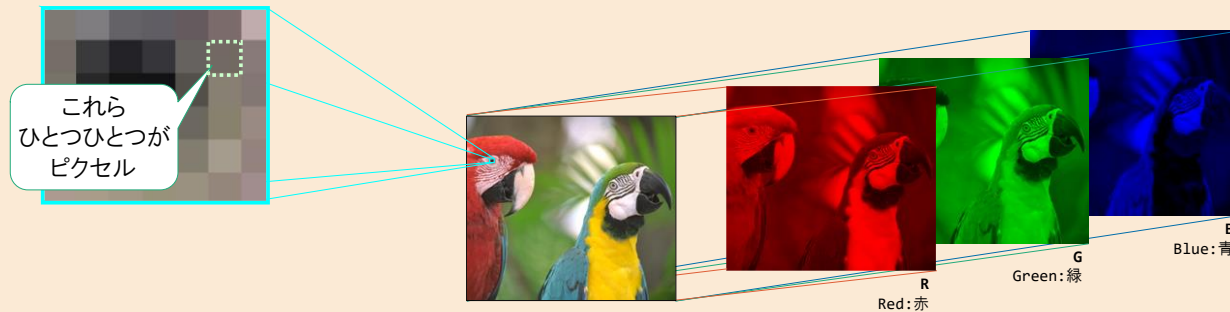
Moodleの
出席確認を
提出しておいて
下さい。

VisualStudio2019(等)で、
C言語+OpenCV のコーディングができる状態に
準備してください。

画像処理 (4J)

第09回

- ラスタ画像とベクタ画像 ... この授業では、ピクセル情報の集合であるラスタ画像を扱う
- 解像度 ... 画像の大きさ(細かさ)
- ピクセル(画素) ... ラスタ画像を構成する1つの点
- チャンネル ... 1ピクセルをいくつかの値で表現するか (例:RGBの3ch)
- 階調数 ... 濃度を何段階で表現するか (例:8bit(=256段階))



デジタル写真 = 有限の解像度で空間的にサンプリング(標本化)し、
有限の階調値で明るさを表現(量子化)したもの ...と捉えることができる。

※音声信号のデジタル化と対応させると、サンプリング周波数が解像度に、量子化bit数が階調数に、チャンネル数はそのまま対応する

第7回のまとめ

12

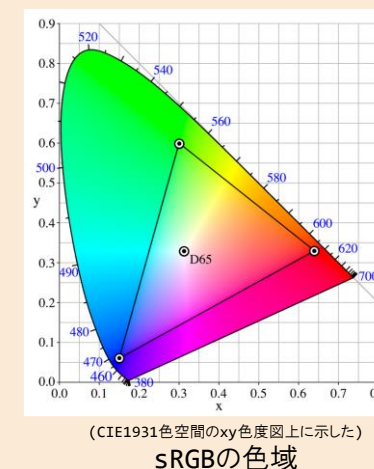
●グレースケール画像とカラー画像

- グレースケール画像は1つの (x, y) 座標点に1つの濃度値 $g(x, y)$
- RGBカラー画像は、1つの座標点に、3つの濃度値



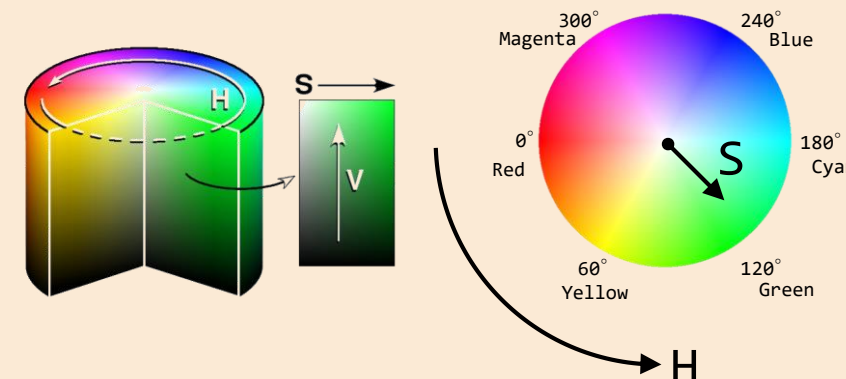
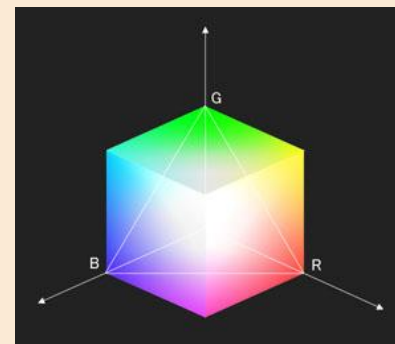
●RGBカラー画像

- RGB値が同じでも、同じ色が表示されるとは限らない
- sRGBに準拠させれば、一貫した色表現が可能。
(ただし表現できる色域が狭い)



●色空間: RGBとHSV

- 相互に変換可能
- 他にも様々な表色系がある

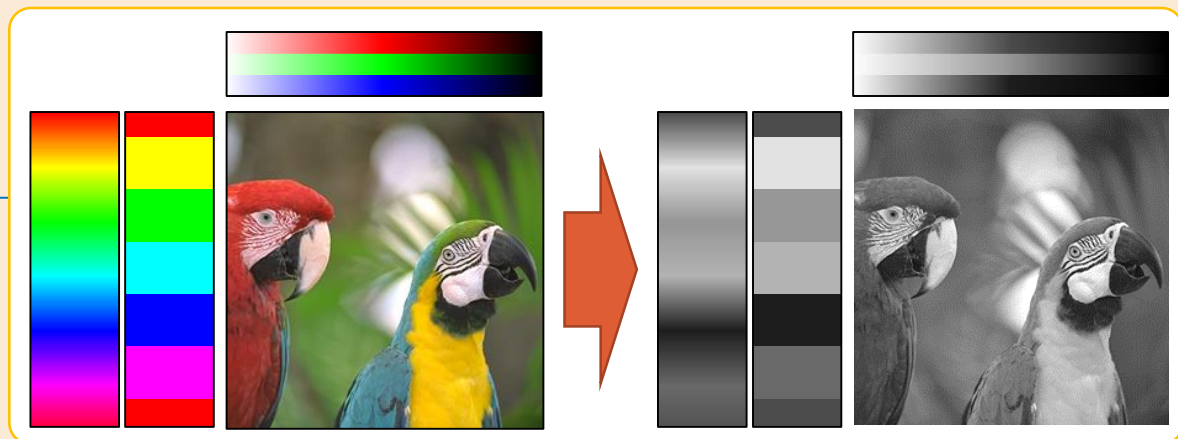


第8回まとめ

●グレイスケール化

- NTSC加重平均法がよく使われる

$$Y = (0.298912 \times R + 0.586611 \times G + 0.114478 \times B)$$



●二値化

- 閾値**を堺に、 $\{0,1\}$ の二値の画像に変換
- 閾値は任意に決められるが、画像統計量から閾値を自動決定する方法として**大津の方法**(判別分析法)が有名。



濃度変換(1)

～明るさ/コントラスト/ガンマ変換、疑似カラー～

- 濃度値を一定の方法で変換し、別の濃度値とすることで、画像を操作できる。

➤まず最初に、単純な線形変換式での濃度変換を考える

$$output = input \times a + b$$

input: 原画像のピクセル値

output: 変換後のピクセル値

a, b: 定数

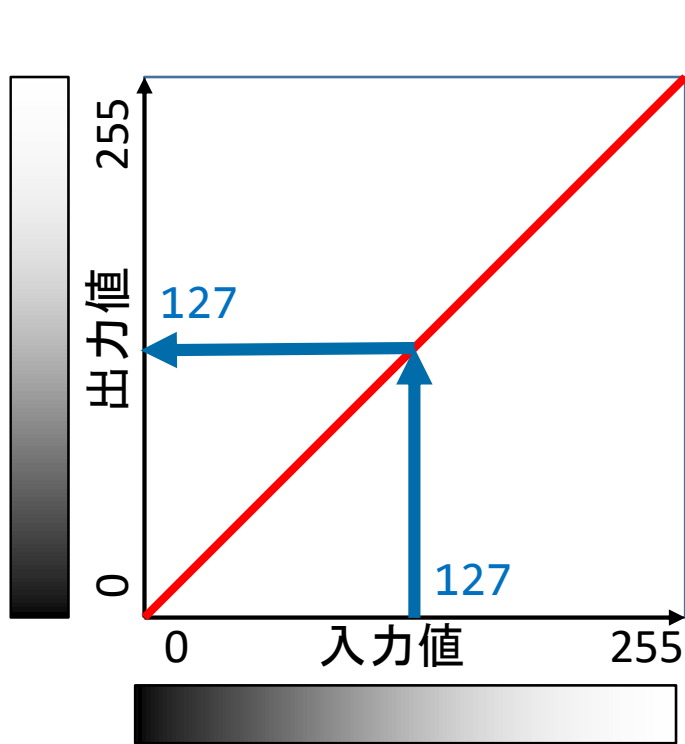
このシンプルな変換で、画像演算でよく使用される

「輝度調整」「コントラスト調整」「階調反転」などが実現できる



●入力濃度値と出力濃度値の対応を表した曲線

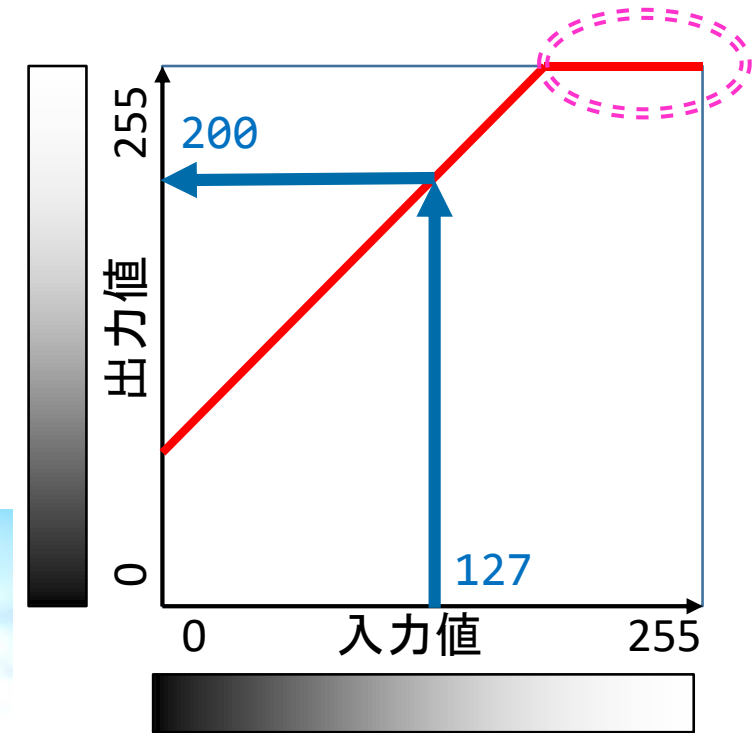
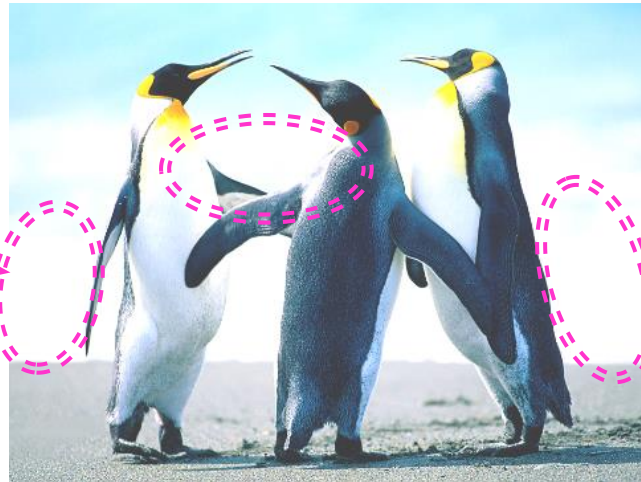
(※この例は直線だが、一般的に言うと曲線)



無変換
($a=1.0$, $b=0.0$)

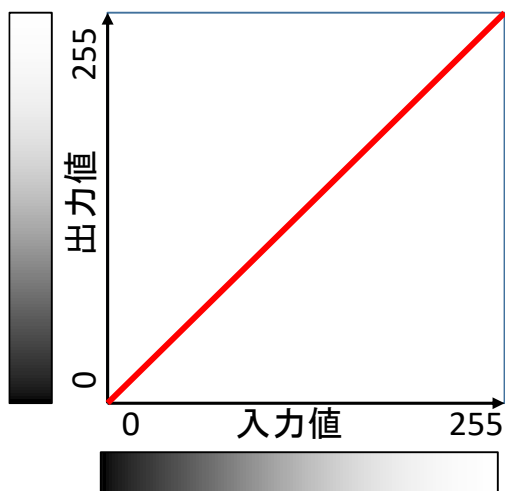


白飛び
(画素値飽和)



明るく
($a=1.0$, $b=73.0$)

原画像

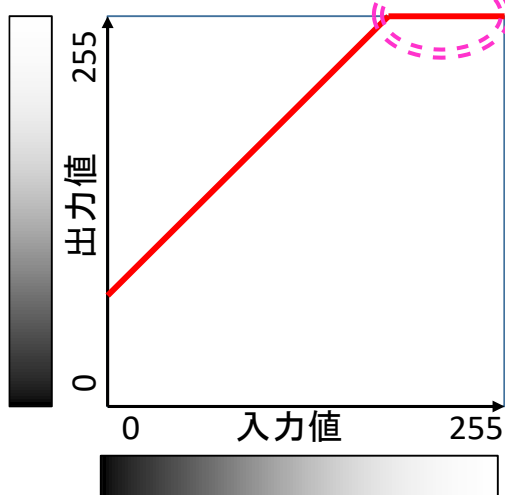


$b > 0$ とすると、濃度値が大きく変換されるので明るく、
 $b < 0$ とすると、濃度値が小さく変換されるので暗くなる。

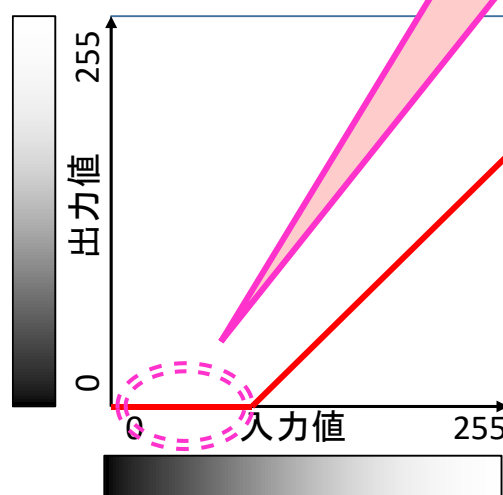
白飛び
(画素値飽和)

黒つぶれ
(画素値飽和)

輝度調整

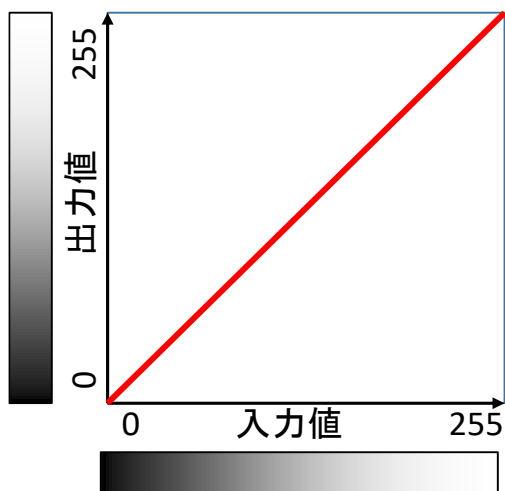


(A) 明るく



(B) 暗く

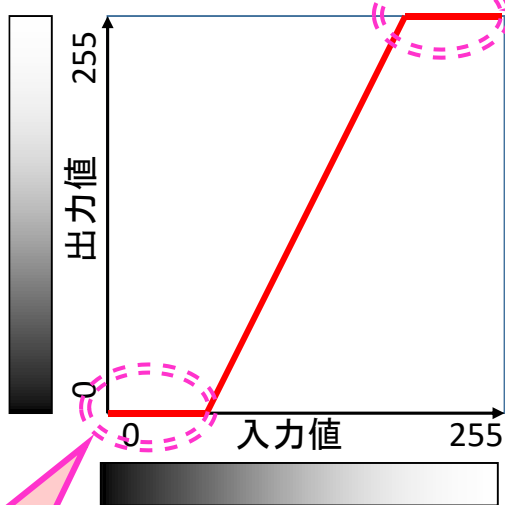
原画像



$a > 1$ とすると、濃度値の小さな変化が大きな変化になるため、コントラストが強くなる。

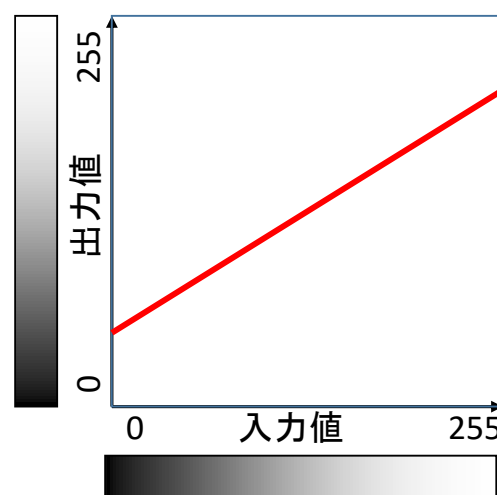
$0 < a < 1$ とすると、濃度値が大きく変化しても、僅かな変化に変換されるので、コントラストが弱くなる。

コントラスト調整



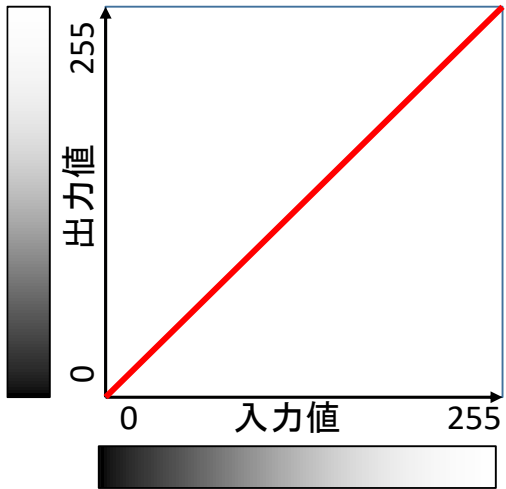
(C) コントラストを強く

黒つぶれ
(画素値飽和)



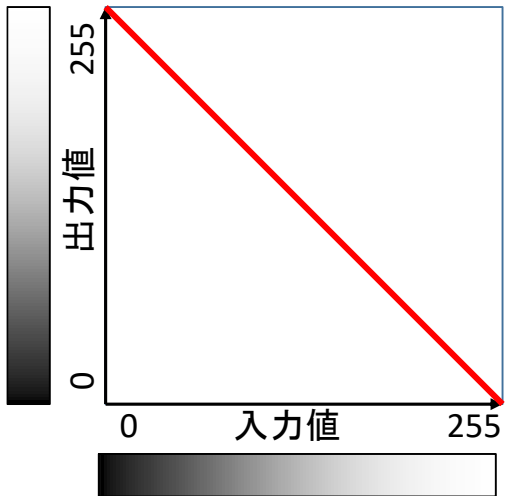
(D) コントラストを弱く

原画像



$a < 0$ とすれば、階調が反転する。

階調反転



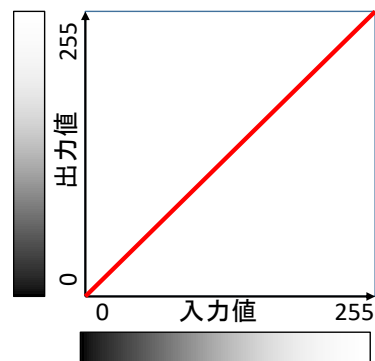
(E)コントラストを強く

- 単純な線形変換以外にも、よく使われる変換式に、指数関数を使ったガンマ変換がある。

$$output = 255 \times \left(\frac{input}{255} \right)^{1/\gamma}$$

※パラメータの γ の値が、
逆数の形で使われていることに注意

原画像

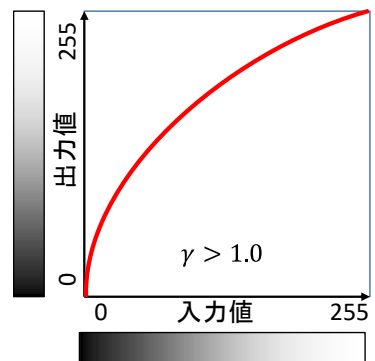


$\gamma > 1$ のとき、暗部が強調され、

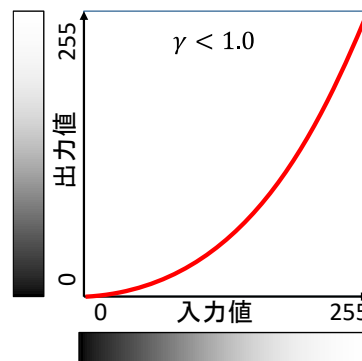
$\gamma < 1$ のときは、明部が強調される。

※線形変換で輝度調整したときのような
白飛び・黒つぶれは起きない

ガンマ変換



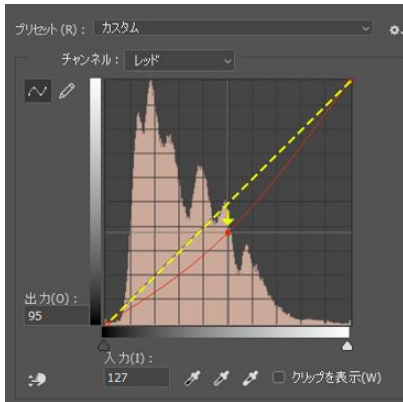
(F) 暗部強調 ($\gamma=1.5$)



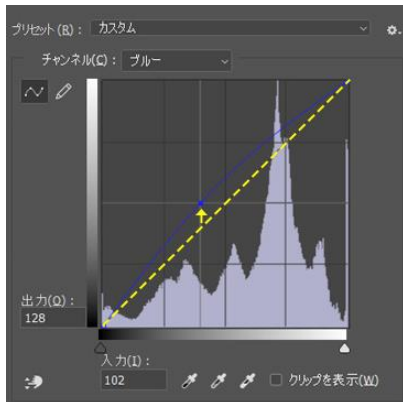
(G) 明部強調 ($\gamma=0.5$)

●カラー画像の場合は、R,G,Bチャンネル等に対して、個別に濃度変換を行うことも可能。

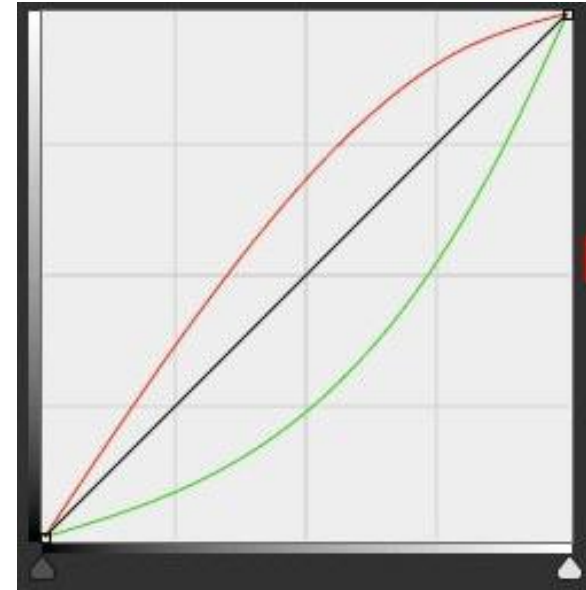
➤色調の調整にも利用できる



Rチャンネルのみ変換



Bチャンネルのみ変換



(画像引用元)

<https://blog.adobe.com/jp/publish/2017/03/30/photo-retouch-details-tone-curve>

<https://www.asobou.co.jp/blog/web/tone-curve2>

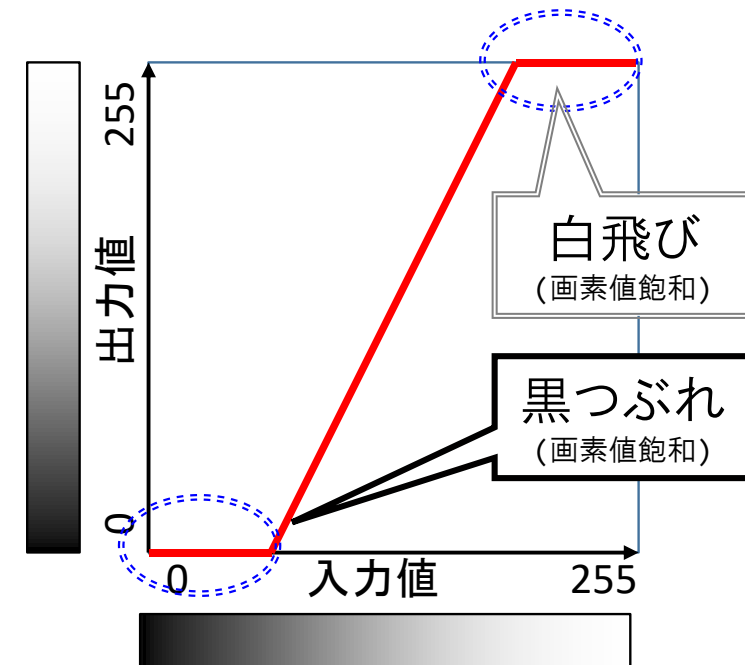
濃度変換に伴う画像の劣化(1)

●白飛び、黒つぶれ

(※値は8bitの例)

➤ 白飛び/黒つぶれ

- ・ 階調変換の結果、255 を超える場合
0未満 となった場合
 - ➡ 255 にクリップ
 - ➡ 0 にクリップ
- 白飛びや黒つぶれが起きた部分は、もともとの階調情報が完全に失われてしまう
- カラー画像の一部のチャンネルのみ飽和した場合
 - ・・・ RGBのバランスが変わり、色がおかしくなる



濃度変換に伴う画像の劣化(1)

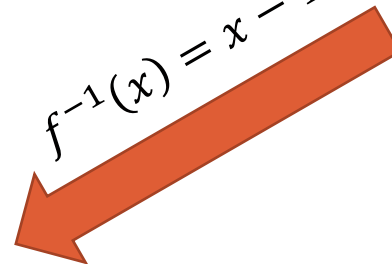
●白飛び



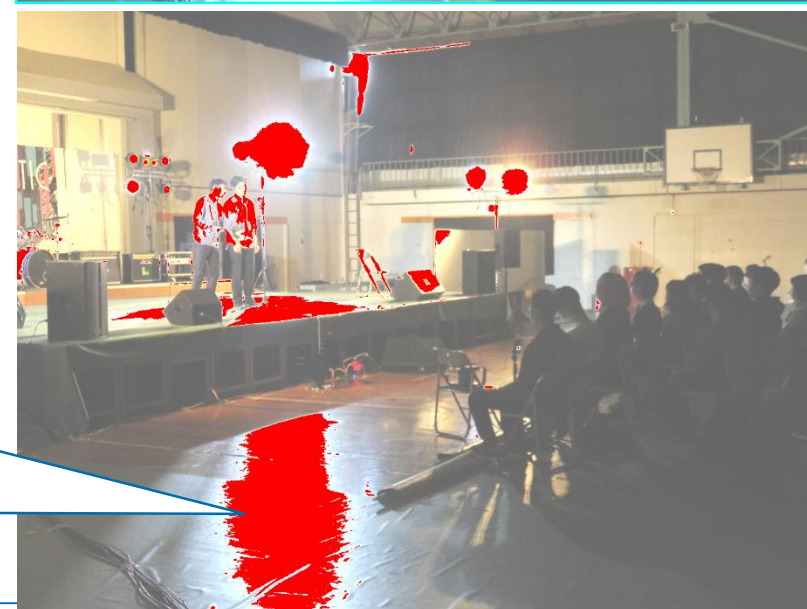
$$f(x) = x + 100$$



$$f^{-1}(x) = x - 100$$



白飛び部分
(R=G=B=255)
を赤で表示



濃度変換に伴う画像の劣化(1)

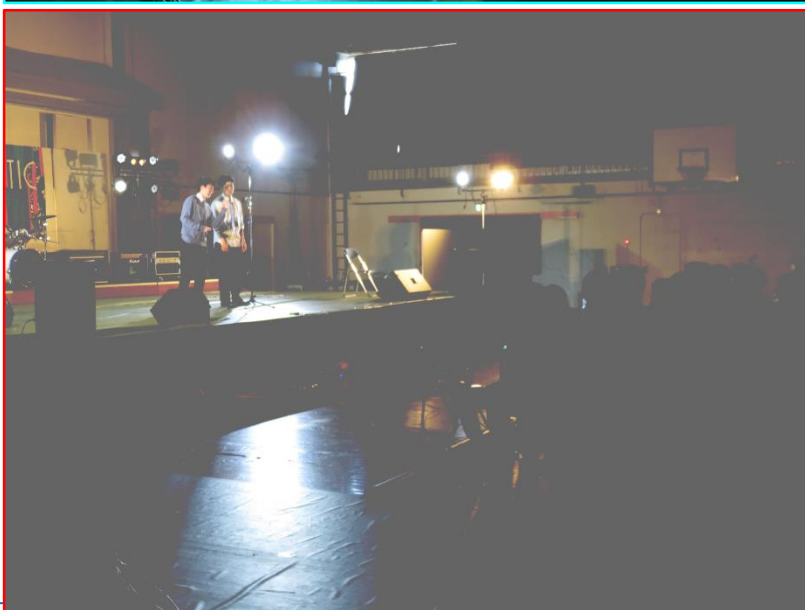
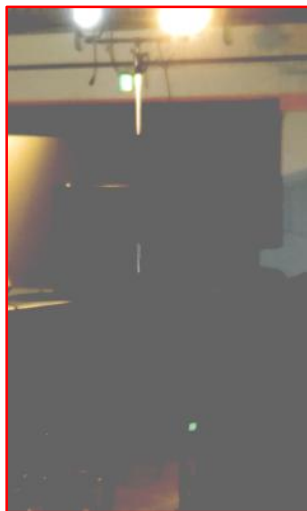
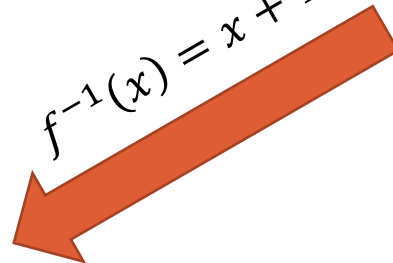
●黒つぶれ



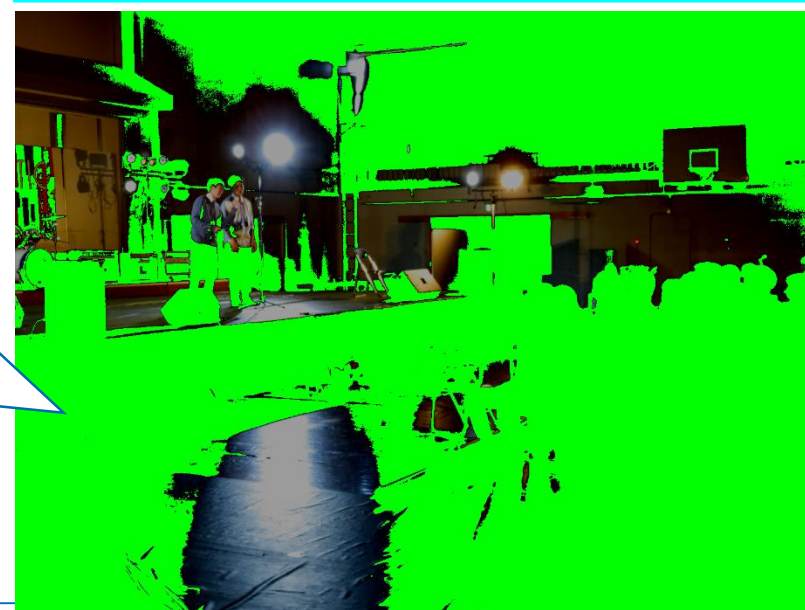
$$f(x) = x - 100$$



$$f^{-1}(x) = x + 100$$

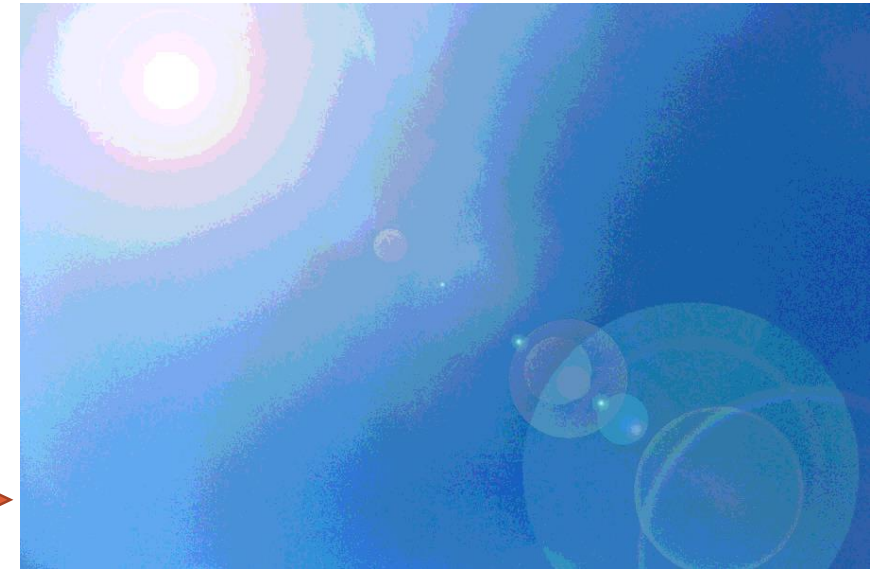
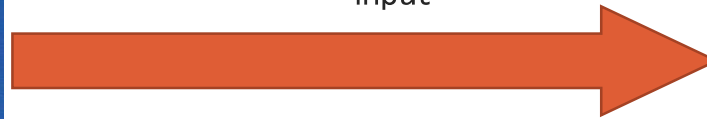
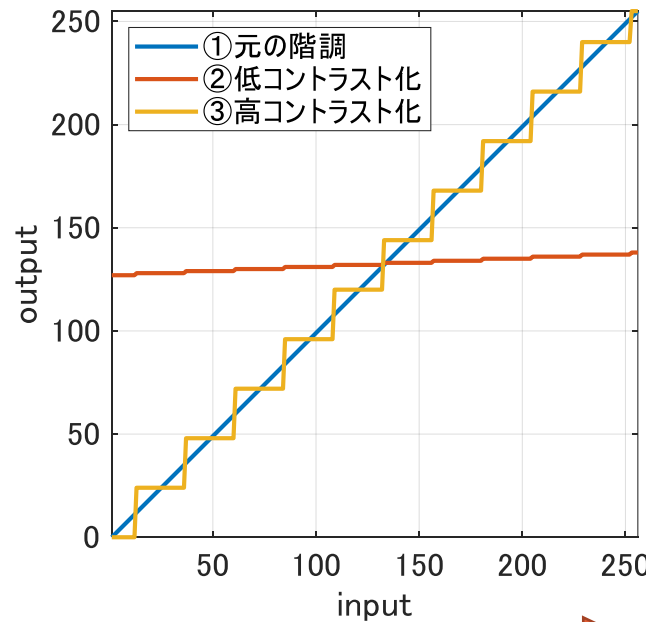
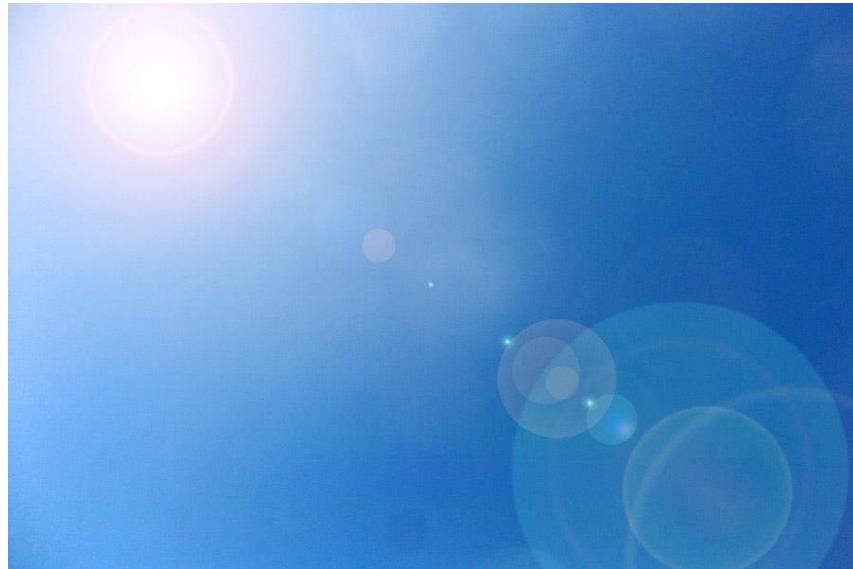


黒つぶれ部分
(R=G=B=0)
を緑で表示



●階調飛び(トーンジャンプ)

- 多数の画像を合成する際などには、階調変換による調整が繰り返し行われる場合があるが、何度もこの変換を行うと中間の階調情報が失われてしまい、目に見える「階調飛び(トーンジャンプ)」が生じてしまうことがある。



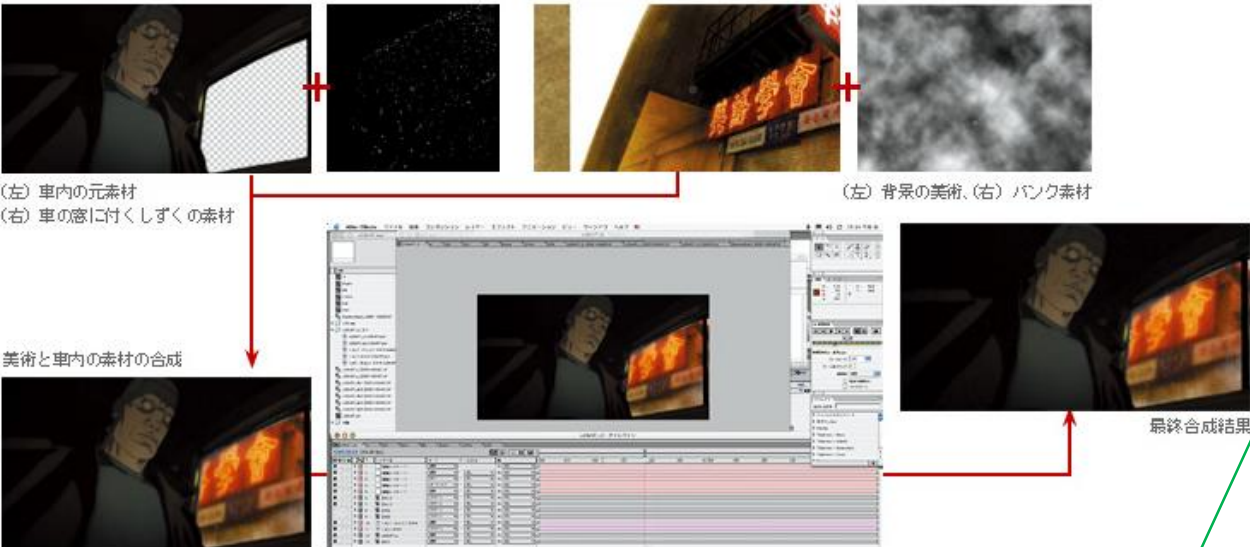
- そのため、画像調整を行う際には(素材や最終的な出力が8bitであっても)内部処理を16bitや32bit、あるいは浮動小数点演算で行う場合がある(※計算負荷は増える)。

◀ Video and audio ▶ Features

プロダクション I.G

page: ◀ 1 2 ▶
April 2004

▶ イノセンスのビジュアルエフェクトワーク1



(左) 車内の元素材
(右) 車の窓に付くしずくの素材

(左) 背景の美術、(右) バンク素材

美術と車内の素材の合成

最終合成結果

江面氏はイノセンスの制作の中でも一番辛かったシーンは「赤のモノトーンで薄暗いシーン」だったと振り返る。「RGB」の「G」と「B」が存在しないシーンで、ほぼ「R」チャンネルのみ256色階調の世界だ。こうしたシーンは、1枚だけの絵ならば256階調でも大丈夫なのだが、レイヤーが重なってお互いに打ち消しあうと、0が1になったり、3が2になったり四捨五入されたりして絵の階調が損なわれていくという。

そうした問題を解決したのがAfter Effectsの16bitモードだ。作業の大詰めの段階で、Power Mac G5が投入され、計算スピードの向上により16bitに移行することで劇的にバンディングを減らし、階調表現を実現することができたという。

”江面氏はイノセンスの制作の中でも一番辛かったシーンは「赤のモノトーンで薄暗いシーン」だったと振り返る。「RGB」の「G」と「B」が存在しないシーンで、ほぼ「R」チャンネルのみ256色階調の世界だ。こうしたシーンは、1枚だけの絵ならば256階調でも大丈夫なのだが、レイヤーが重なってお互いに打ち消しあうと、0が1になったり、3が2になったり四捨五入されたりして絵の階調が損なわれていくという。

そうした問題を解決したのがAfter Effectsの16bitモードだ。作業の大詰めの段階で、Power Mac G5が投入され、計算スピードの向上により16bitに移行することで劇的にバンディングを減らし、階調表現を実現することができたという。”



疑似カラーは次回に

●濃度変換 …… 濃度値を一定の方法で新しい濃度値に変換

➤ 線形変換 (Linear Stretch) $output = input \times a + b$

➤ ガンマ変換 (Gamma Stretch) $output = 255 \times \left(\frac{input}{255}\right)^{1/\gamma}$

➤ 輝度調整、コントラスト調整、階調反転などに利用可能

●濃度変換に伴う画像の劣化

➤ 白飛び …… 変換後に最大値以上になった場合に、最大値にクリップされる

➤ 黒つぶれ …… 変換後に最小値以下になった場合に、最小値にクリップされる

➤ 階調飛び(トーンジャンプ) …… 中間値の階調が失われ、濃度値が不連続に変化

演習： 濃度変換

OpenCVを使った画像生成の流れ



30

【画像生成時の流れ】

```
IplImage* img = cvCreateImage(CvSize size, IPL_DEPTH_8U, int channels);
```

↓

… 画像を扱うための構造体 `img` を生成する

```
cvSetZero(img);
```

… 画像データ `img` を 0(=黒) で初期化

↓

↓

【画像読み込み時の流れ】

↓

```
IplImage* img = cvLoadImage(const char* filename, CV_LOAD_IMAGE_UNCHANGED);
```

↓

↓

… 画像ファイルを読み取り、画像データ `img` を生成

↓

↓

<`img` に対する何らかの処理>

↓

↓

```
cvSaveImage(img);
```

… 画像データ `img` を画像ファイルとして保存

↓

```
cvReleaseImage(&img);
```

… 画像を扱うための構造体 `img` に割り当てたメモリの開放

各種関数のリファレンス(1)



31

```
IplImage* img  
= cvCreateImage(CvSize size, int depth, int channels);
```

- size: 画像のサイズ。
- depth: ピクセルのデータ形式。
 - ※本授業では常に IPL_DEPTH_8U (符号無し8ビット整数 = unsigned char)とする。
- channels: ピクセル毎のチャンネル数。[グレイスケール = 1 , カラー = 3]

※ロードに失敗した場合は NULL が返る。
内部でmalloc()されているので、cvReleaseImage()で開放する必要がある。

```
typedef struct CvSize {  
    int width;    /* 横幅 */  
    int height;   /* 高さ */  
} CvSize;
```

各種関数のリファレンス(2)



32

```
void cvSetZero(IplImage *img);
```

- `img`: `cvCreateImage()` が返した `IplImage*` のアドレス。
全ピクセルデータを 0(黒)で初期化する

```
IplImage* img  
= cvLoadImage(const char* filename, int iscolor);
```

- `filename`: ファイル名。対応ファイル形式は(表1)を参照。
- `iscolor`: 読み込む画像のカラーの種類。
※本授業では常に `CV_LOAD_IMAGE_UNCHANGED` とする。

指定した画像ファイルを `IplImage` 形式に読み込む

※内部で`malloc()`されているので、`cvReleaseImage()`で開放する必要がある。

各種関数のリファレンス(3)



33

```
int cvSaveImage(const char* filename, IplImage* image);
```

- filename: ファイル名。拡張子で保存形式が決まる。→ (表1)を参照。
- image: 保存する画像データ
IplImage を、画像ファイルとして保存する。

※保存に成功した場合は 1 、失敗した場合は 0 が返る(らしい)。

```
void cvReleaseImage(IplImage** img);
```

- img: cvCreateImage() が返した IplImage* のアドレス。
cvCreateImage()やcvLoacImage()で確保された領域を開放する。

(表 1) cvLoacImage()、cvSaveImage() の対応形式と、指定する拡張子

形式	Windows bitmaps	Jpeg	Portable Network Graphics	Portable image format	Sun rasters	TIFF files	OpenEXR HDR images	JPEG 2000 images
拡張子	BMP,DIB	JPEG, JPG,JPE	PNG	PGM,PGM PPM	SR,RAS	TIFF, TIF	EXR	Jp2

IplImage 構造体 (types_c.h 内で定義) 再

34

```
typedef struct _IplImage
{
    int    nSize;           /* sizeof(IplImage) */
    int    ID;              /* version (=0)*/
    int    nChannels;        /* Most of OpenCV functions support 1,2,3 or 4 channels */
    int    alphaChannel;     /* Ignored by OpenCV */
    int    depth;           /* Pixel depth in bits: IPL_DEPTH_8U, IPL_DEPTH_8S, IPL_DEPTH_16S,
                             IPL_DEPTH_32S, IPL_DEPTH_32F and IPL_DEPTH_64F are supported. */
    char    colorModel[4];   /* Ignored by OpenCV */
    char    channelSeq[4];   /* ditto */
    int    dataOrder;       /* 0 - interleaved color channels, 1 - separate color channels.
                             cvCreateImage can only create interleaved images */
    int    origin;          /* 0 - top-left origin,
                             1 - bottom-left origin (Windows bitmaps style). */
    int    align;           /* Alignment of image rows (4 or 8).
                             OpenCV ignores it and uses widthStep instead. */
    int    width;           /* Image width in pixels. */
    int    height;          /* Image height in pixels. */
    struct _IplROI *roi;     /* Image ROI. If NULL, the whole image is selected. */
    struct _IplImage *maskROI; /* Must be NULL. */
    void    *imageId;        /* " */
    struct _IplTileInfo *tileInfo; /* " */
    int    imageSize;       /* Image data size in bytes
                             (==image->height*image->widthStep
                             in case of interleaved data)*/
    char    *imageData;      /* Pointer to aligned image data. */
    int    widthStep;        /* Size of aligned image row in bytes. */
    int    BorderMode[4];    /* Ignored by OpenCV. */
    int    BorderConst[4];   /* Ditto. */
    char    *imageDataOrigin; /* Pointer to very origin of image data
                              (not necessarily aligned) -
                              needed for correct deallocation */
}
IplImage;
```

- `IplImage` のメンバ変数の `nChannels`

- 3の場合カラー画像

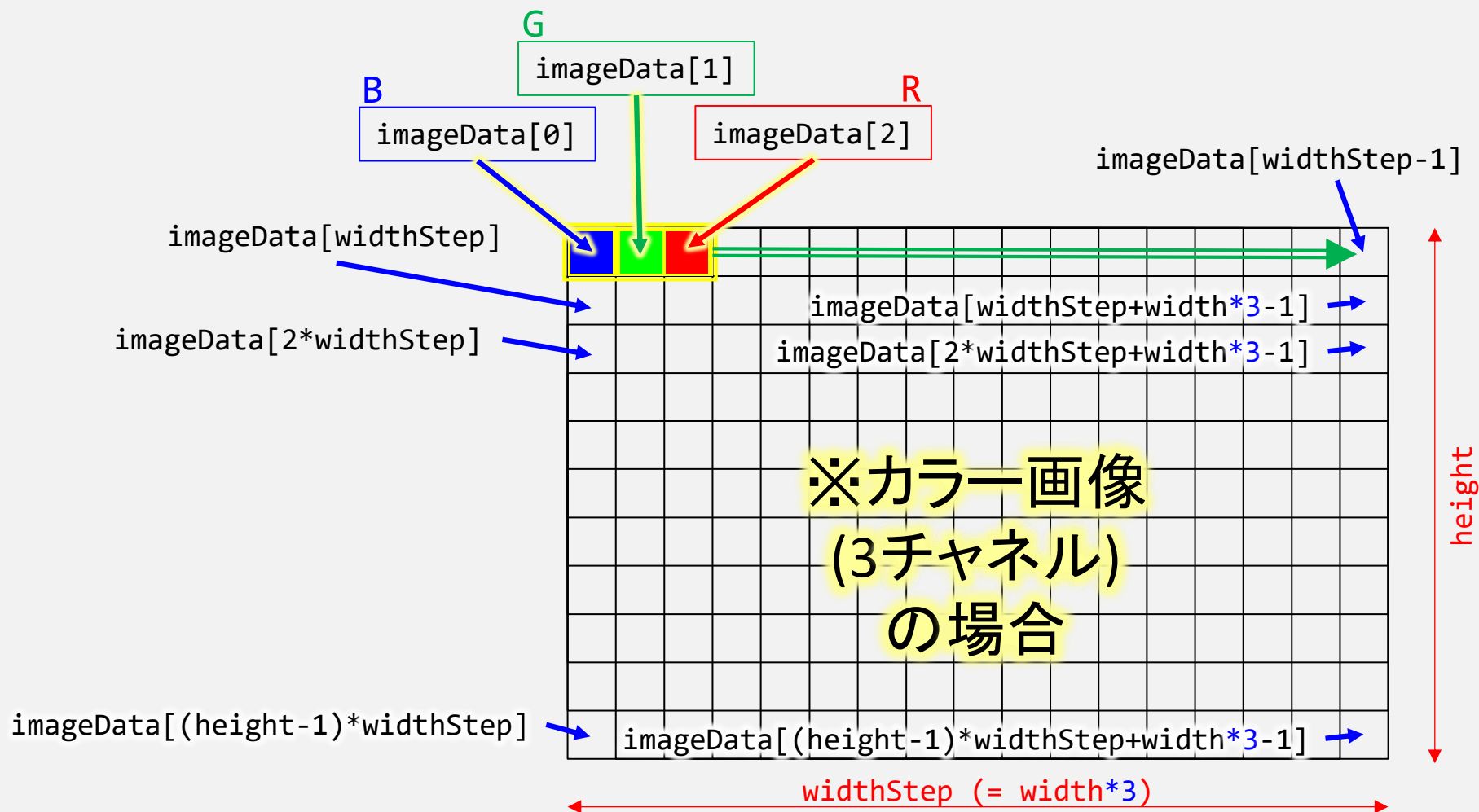
- 1の場合グレースケール画像

- (※本授業では、`nChannels`が1か3の場合のみ、取り扱うものとする)

Ip1ImageのRGB値へのアクセス 再

36

RGBカラー画像の個々のRGB値は、
下図のような順に一次元配列 `imageData[]` に格納される。



IplImageのRGB値へのアクセス

再 (変更後) 37

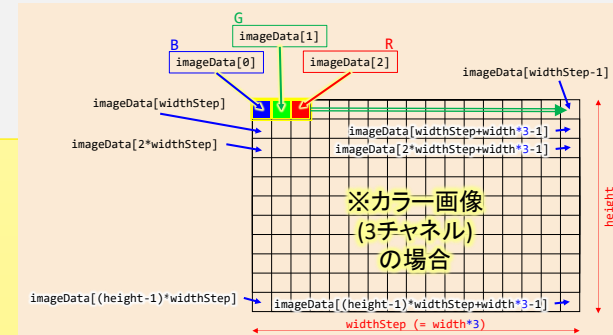
- IplImage のメンバ変数を用いて、個々のピクセルへアクセスする。
- imageDataには、
BGRBGRBGRBGR.....の順で格納されていることに注意 (**RGBの順ではない!**)。
 - char* imageData ... 画像データへのポインタ
 - int widthStep ... 画像データ1ライン分のバイト数(= char で数えた数)

【例】

IplImage *img の画像(RGBカラー画像)に対して、
座標点 (x, y) のカラーチャンネルごとのピクセル値(RGB値)へは、

```
b = (unsigned char)img->imageData[img->widthStep * y + x * 3 + 0];  
g = (unsigned char)img->imageData[img->widthStep * y + x * 3 + 1];  
r = (unsigned char)img->imageData[img->widthStep * y + x * 3 + 2];
```

としてアクセスすることが出来る。



演習 & 課題No.09 : 濃度変換

課題 No.09-01 : 濃度変換(線形変換)

39

●濃度変換(線形変換) … modeの値(1～5:次スライドで説明)に基づいて変換

```
// No.09-01 : 濃度変換(線形変換)
```

```
#include <stdio.h>
#include <opencv/highgui.h>
```

```
// double型のピクセル値を、unsigned char型で返す。0未満の値は0に、255以上の値は255にクリップする
unsigned char clip(double v) {
    return (v > 255) ? 255 : (v < 0) ? 0 : (unsigned char)v;
}
```

```
// 線形変換(Linear Stretch)
```

```
// dst: 変換結果の画像
```

```
// src: 変換元の画像
```

```
// dst = src * a + b
```

```
void linear(IplImage* dst, IplImage* src, double a, double b) {
    double v;
    for (int y = 0; y < src->height; y++) {
        for (int x = 0; x < src->width; x++) {
            // グレイスケールとカラー両方まとめて処理するためのループ
            for (int ch = 0; ch < src->nChannels; ch++) {
                v = ... // 一旦 double で計算
                dst->imageData[dst->widthStep * y + x * dst->nChannels + ch] = clip(v);
            }
        }
    }
}
```

```
void main() {
```

```
    IplImage* src;
    IplImage* dst;
    double a, b; // 係数
```

```
    int mode = 0; // この値でモードを切り替える
```

```
    char filename[] = "Mandrill.bmp";
```

```
switch (mode) { // a, b の値をそれぞれ設定する
    case 0: // 無変換 -----
    case 1: // ①明るく -----
    case 2: // ②暗く -----
    case 3: // ③コントラストを強く -----
    case 4: // ④コントラストを弱く -----
    case 5: // ⑤階調反転 -----
    default: // -----
        printf("modeが範囲外の値です\n");
        return;
}
```

```
// 画像データの読み込み
```

```
if ((src = cvLoadImage(filename, CV_LOAD_IMAGE_UNCHANGED)) == NULL) {
    printf("画像ファイルの読み込みに失敗しました。\\n");
    return;
}
```

```
printf("%s\\nnChannels = %d\\nnMode = %d\\nn\\ta = %f\\nn\\tb = %f\\nn", filename, src->nChannels, mode, a, b);
```

```
// 読み込んだ画像と同じサイズ、同じチャンネル数(nChannels)の画像を生成
```

```
dst = cvCreateImage(cvSize(src->width, src->height), src->depth, src->nChannels);
```

```
cvNamedWindow("Source");
cvShowImage("Source", src);
```

```
linear(dst, src, a, b);
```

```
cvNamedWindow("Destination");
cvShowImage("Destination", dst);
```

```
cvWaitKey(0);
cvDestroyAllWindows();
cvReleaseImage(&dst);
cvReleaseImage(&src);
return;
```

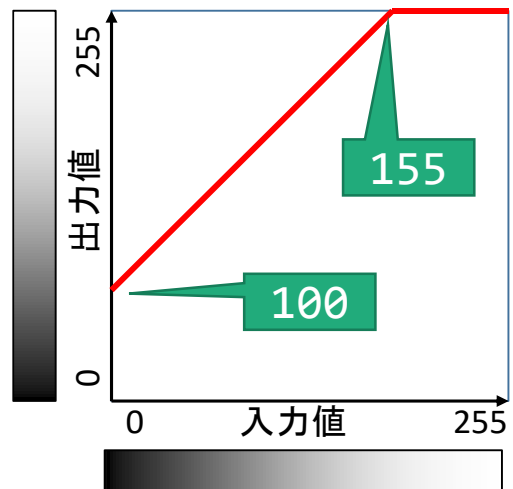
```
}
```

```
// おまけ: テスト用の関数
```

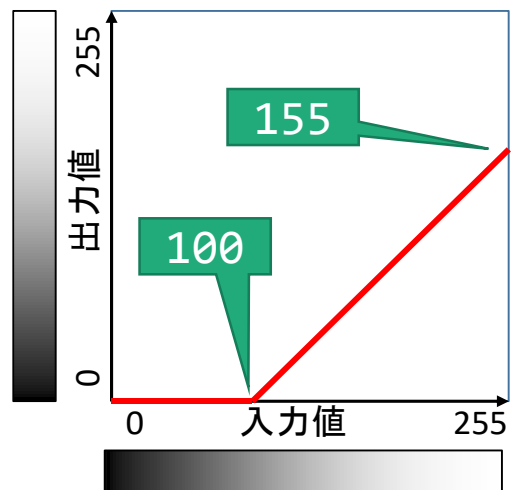
```
void test(double a, double b) {
    // 256個のピクセルの画像を生成
    IplImage* t1 = cvCreateImage(cvSize(256, 1), IPL_DEPTH_8U, 1);
    IplImage* t2 = cvCreateImage(cvSize(256, 1), IPL_DEPTH_8U, 1);
    for (int i = 0; i < 256; i++)
        t1->imageData[i] = i;
    linear(t2, t1, a, b);
    // 変換元 ==> 変換結果 の対応の一覧を出力
    for (int i = 0; i < 256; i++)
        printf("%d ==> %d\\n", i, (unsigned char)t2->imageData[i]);
}
```

課題 No.09-01 : 濃度変換(線形変換)

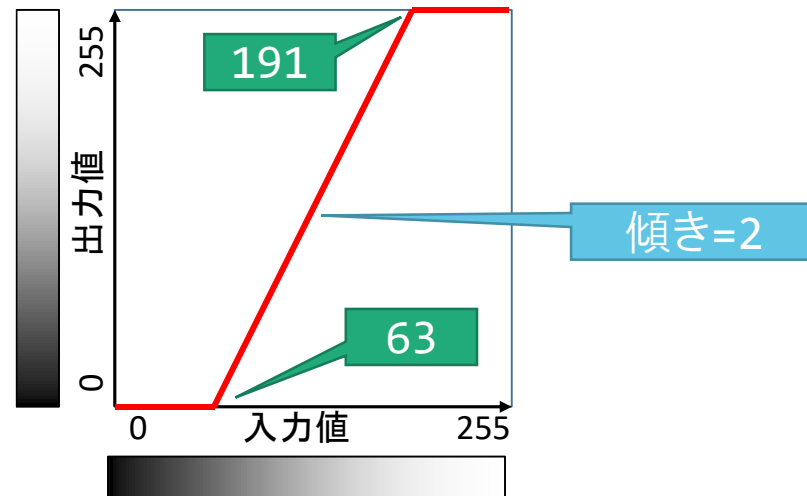
① mode=1 : 明るく



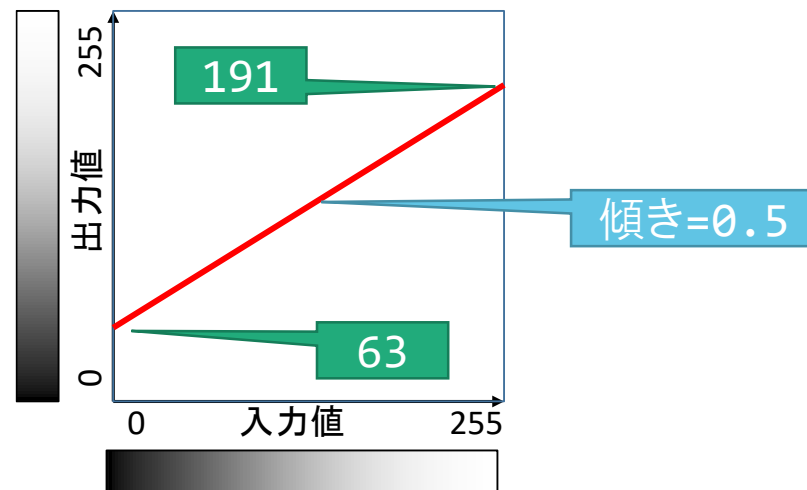
② mode=2 : 暗く



③ mode=3 : コントラストを強く

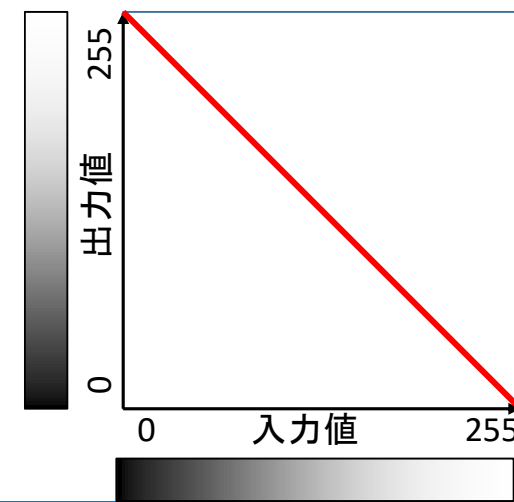


④ mode=4 : コントラストを弱く



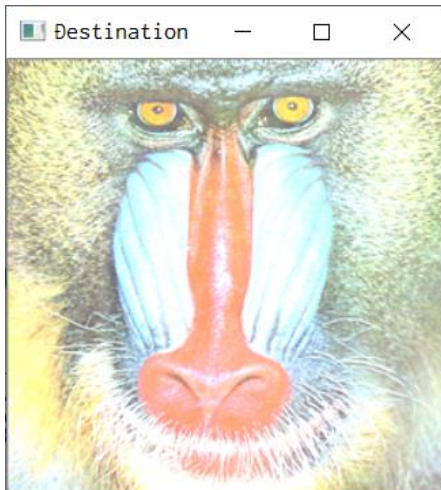
※変曲点の値は、
±1程度の誤差があっても
構いません。

⑤ mode=5 : 階調反転

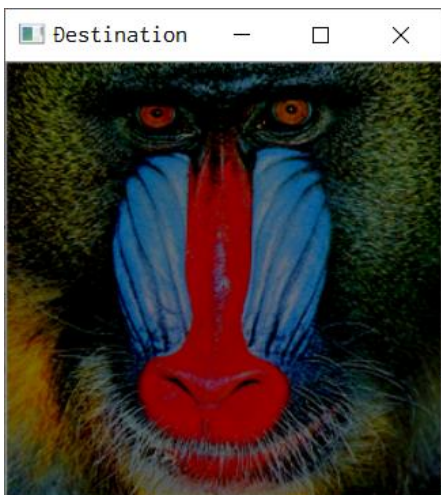


課題 No.09-01 : 【処理結果例】

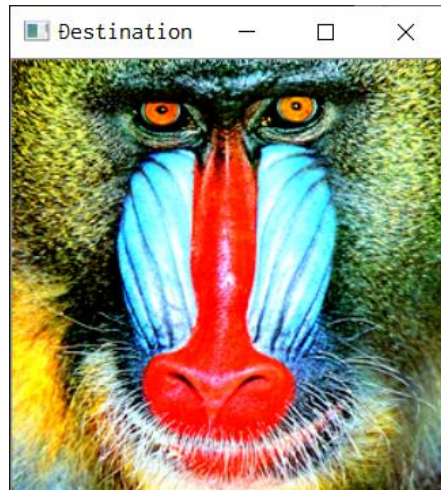
① mode=1 : 明るく



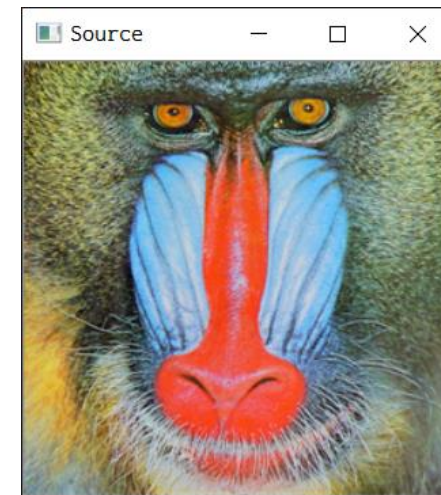
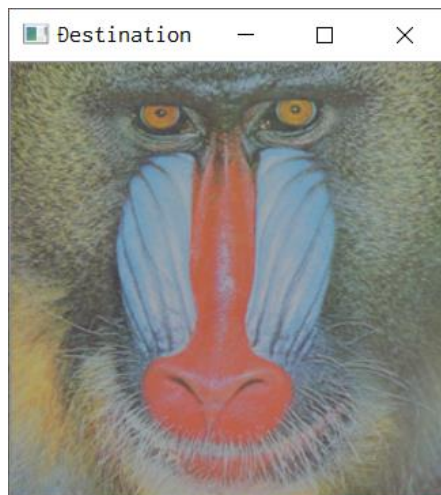
② mode=2 : 暗く



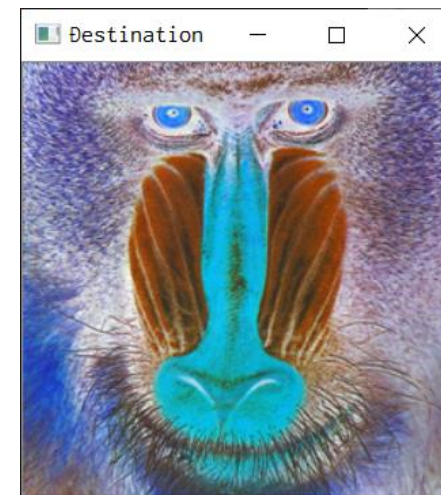
③ mode=3 : コントラストを強く



④ mode=4 : コントラストを弱く



⑤ mode=5
: 階調反転



課題 No.09-02 : 濃度変換(γ 変換)

42

●濃度変換(γ 変換)

```
// No.09-02 : 濃度変換( $\gamma$ 変換)

#include <stdio.h>
#include <opencv/highgui.h>

// double型のピクセル値を、unsigned char型で返す。0未満の値は0に、255以上の値は255
// にクリップする
unsigned char clip(double v) {
    return (v > 255) ? 255 : (v < 0) ? 0 : (unsigned char)v;
}

//  $\gamma$ 変換(Gamma Stretch)
// dst: 変換結果の画像
// src: 変換元の画像
// dst = 255 * (src/255)^(1/g)
void gamma(IplImage* dst, IplImage* src, double g) {
    // ここを実装する
}
```

```
void main() {

    IplImage* src;
    IplImage* dst;
    double g = 1.0; // 係数(ここを調整)

    char filename[] = "Mandrill.bmp";

    // 画像データの読み込み
    if ((src = cvLoadImage(filename, CV_LOAD_IMAGE_UNCHANGED)) == NULL) {
        printf("画像ファイルの読み込みに失敗しました。¥n");
        return;
    }
    printf("%s¥nnChannels = %d¥n¥ngamma = %f¥n", filename, src->nChannels, g);

    // 読み込んだ画像と同じサイズ、同じチャンネル数(nChannels)の画像を生成
    dst = cvCreateImage(cvSize(src->width, src->height), src->depth, src->nChannels);

    cvNamedWindow("Source");
    cvShowImage("Source", src);

    gamma(dst, src, g);

    cvNamedWindow("Destination");
    cvShowImage("Destination", dst);

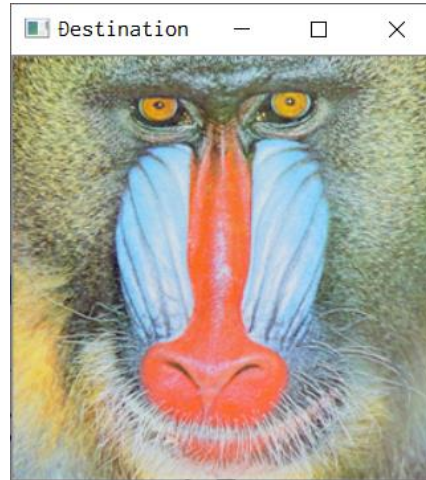
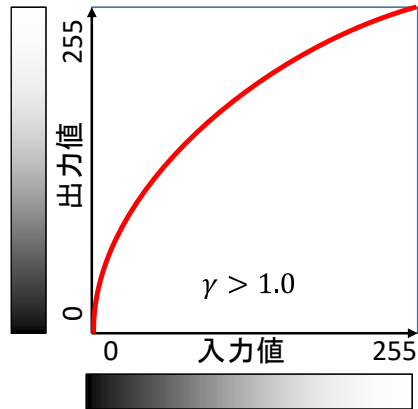
    cvWaitKey(0);
    cvDestroyAllWindows();
    cvReleaseImage(&dst);
    cvReleaseImage(&src);
    return;
}
```


課題 No.09-02 : 濃度変換(γ 変換)

43

●以下の2通りの γ 値 での変換結果を載せること

① 暗部強調: $\gamma = 1.5$



② 明部強調: $\gamma = 0.5$

