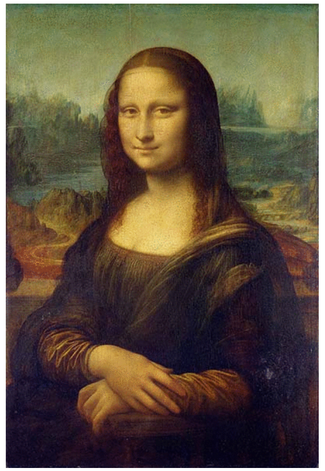
http://www.takagi.inf.uec.ac.jp/mmip/

マルチメディア処理

Multimedia Information Processing

第7回：画像処理 (1) 画素ごとの濃淡変換



髙木一幸

講義概要

・マルチメディアデータ（文書、音声、画像、映像など）の表現方法と処理技術について、基礎的な内容を紹介・解説する。

・授業時間中および宿題として、計算機を使った演習を行うことで実際のデータの扱い方を学び、理解を深める。

第1回：授業の概要説明、
人間の感覚とマルチメディア処理（4/14）

髙木

第2回：文字・テキストの表現と処理（4/21）

第3回：音声のディジタル表現と処理（4/28）

第4回：画像・映像のディジタル表現（5/12）

第5回：マルチメディアデータの符号化とファイル形式（5/19）

第6回：2次元図形の表現と描画（5/26）

第7回：画像処理(1)画素ごとの濃淡変換（6/2）

第8回：画像処理(2)空間フィルタリング（6/9）

廣田

第9回：カメラと写真撮影（6/16）

第10回：3次元コンピュータグラフィックス（6/23）
(1)形状表現と透視投影

第11回：3次元コンピュータグラフィックス（6/30）
(2)照明効果とシェーディング

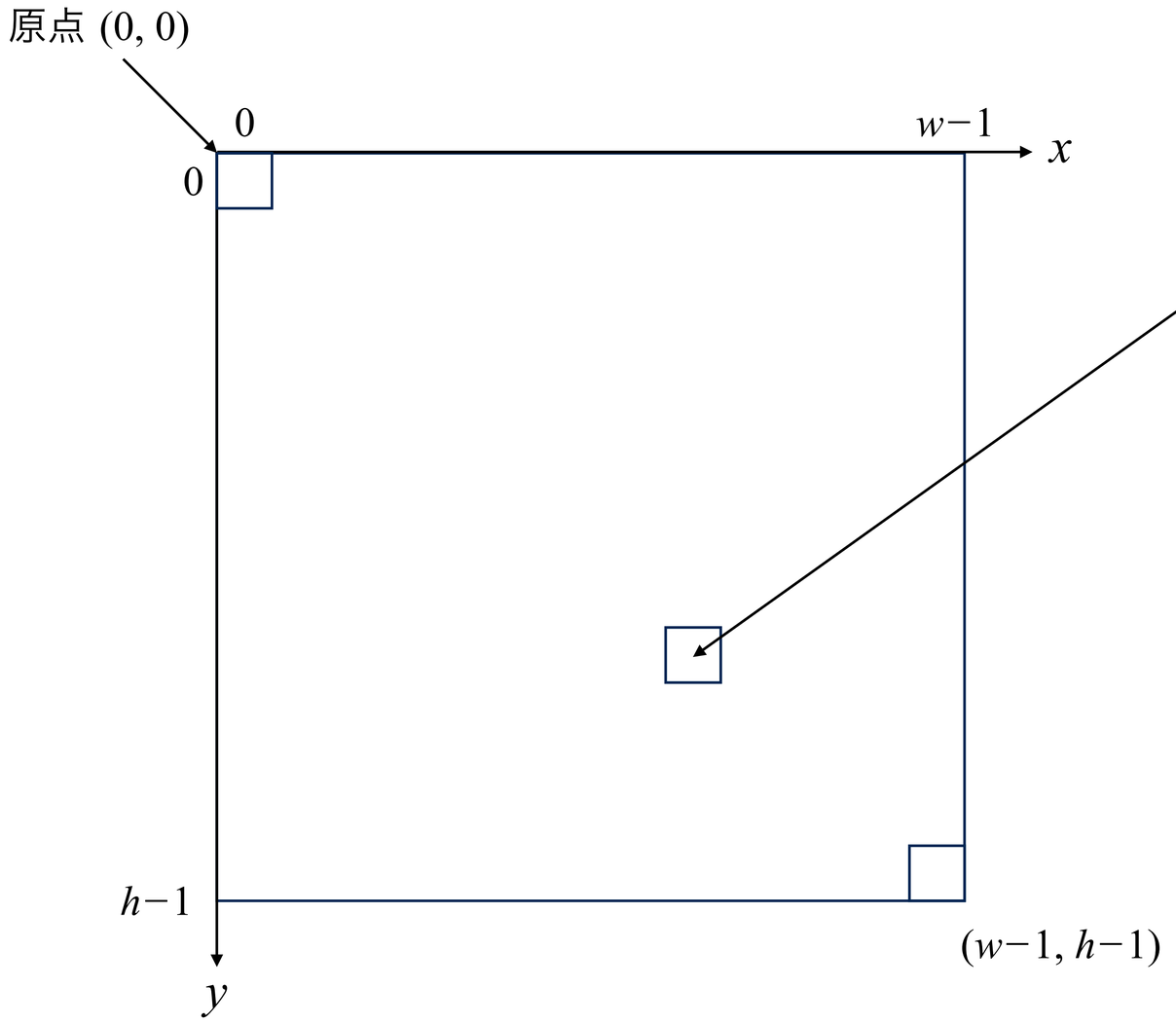
第12回：アニメーションと映像制作（7/7）

第13回：シミュレーションと可視化（7/14）

第14回：グラフィックパイプラインとシェーダ（7/21）

第15回：マルチメディアの応用例I（7/28）

画像の記述方法



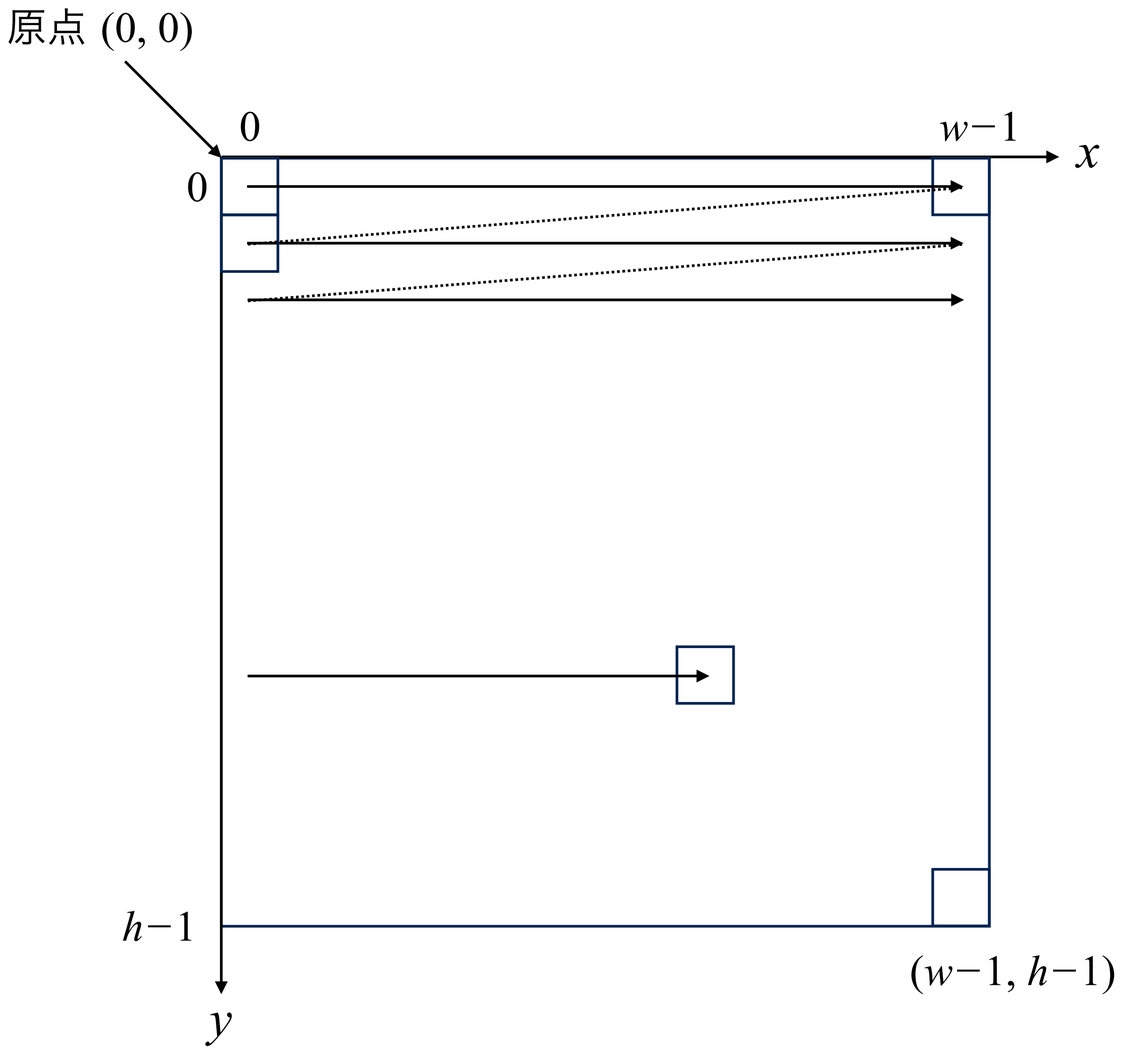
(x, y)

階調値：$f(x, y)$

階調数が1byteの場合は

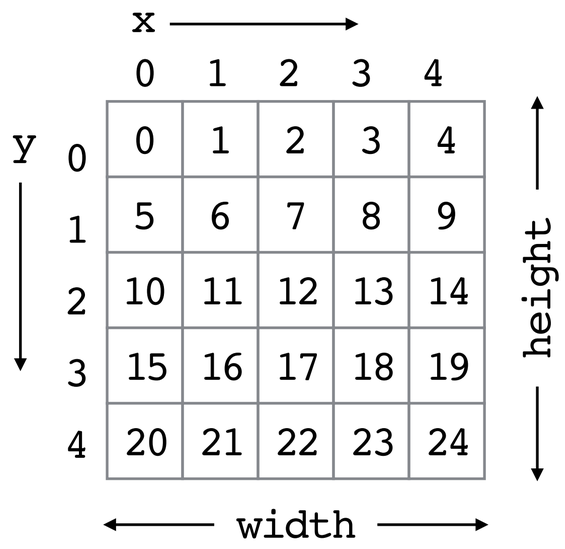
$0 \leq f(x, y) \leq 255$

ラスタスキャン（ラスタ走査）

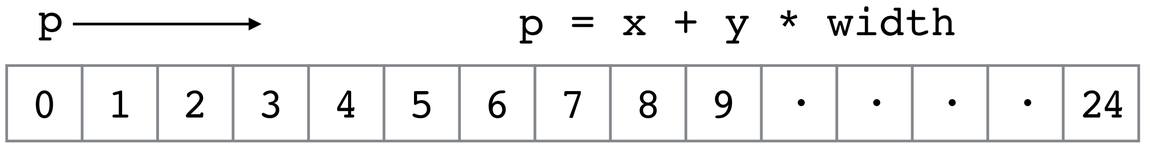


Processingで画像を扱う

ピクセル配列を扱う



画素の配置



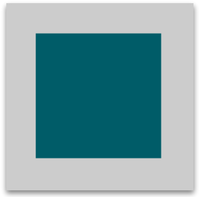
画素配列のデータ格納順

ラスタスキャンの順序で画素値を参照・操作

```
for (int y = 0; y < height; y++) {
  for (int x = 0; x < width; x++) {
    int p = x + y * width;
    pixels[p] = ...
  }
}
```

Processingで画像を扱う

createImage()　画像データを生成する



```
PImage img = createImage(66, 66, RGB);
```

66×66のRGB画像データを生成

```
for (int i = 0; i < img.pixels.length; i++) {
  img.pixels[i] = color(0, 90, 102);
}
```

画素配列のサイズ

RGB値(0, 90, 102)の色で塗り潰す

```
img.updatePixels();
image(img, 17, 17);
```

```
PImage img = createImage(66, 66, ARGB);
```

66×66のアルファチャネル付きRGB画像データを生成

```
for (int i = 0; i < img.pixels.length; i++) {
  img.pixels[i] = color(0, 90, 102, (i%img.width)*2);
}
```

横方向の座標値によって透明度を変化させる

```
img.updatePixels();
image(img, 17, 17);
image(img, 34, 34);
```

同じ2枚の透明画像を重ねて表示する

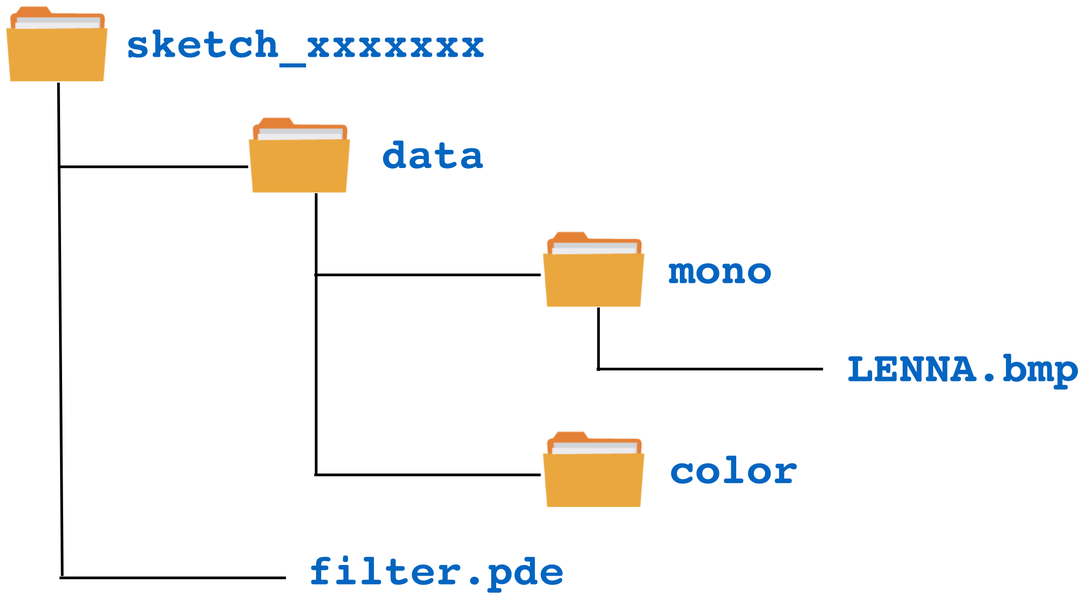
Processingで画像を扱う

`loadPixels()` 現在表示されている画素データを配列 **`pixels[]`** に読み込む

```
int halfImage = width*height/2;
PImage myImage = loadImage("apples.jpg");
image(myImage, 0, 0);

loadPixels();
for (int i = 0; i < halfImage; i++) {
  pixels[i+halfImage] = pixels[i];
}
updatePixels();
```

Processingにおける画像ファイルの扱い

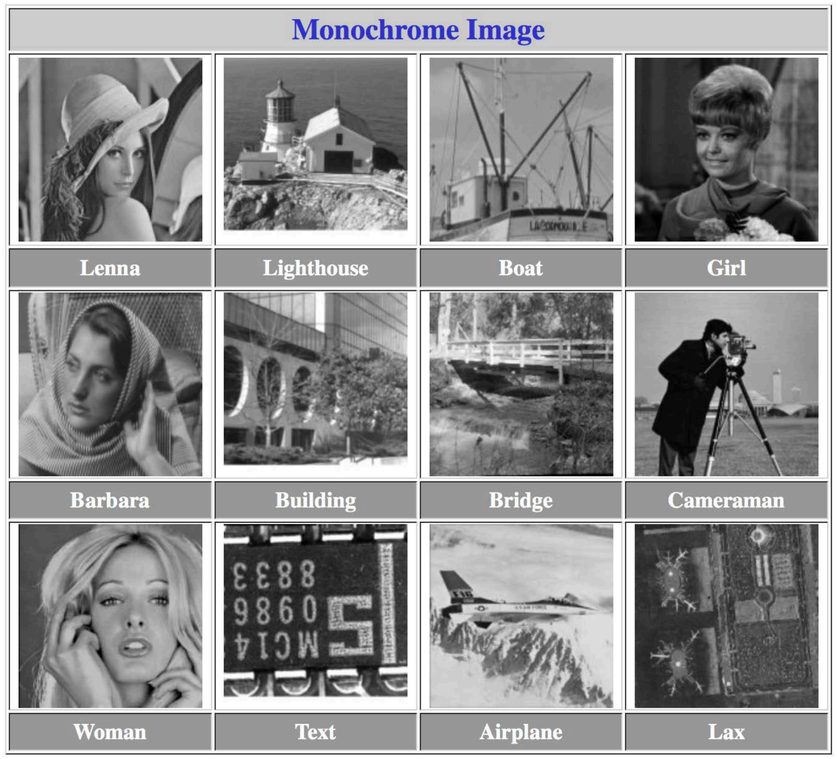


・画像、動画、フォント、サウンド、テキスト（CSV、XML形式などのデータ）などプログラムで使用する外部データはスケッチの入っているフォルダ内のdataフォルダに格納するというルールになっている。

・さまざまな種類のデータが混在する場合は、dataフォルダ内にさらにサブフォルダを作成しても構わない。

標準画像データベースSIDBA

Standard Image Data-BAse

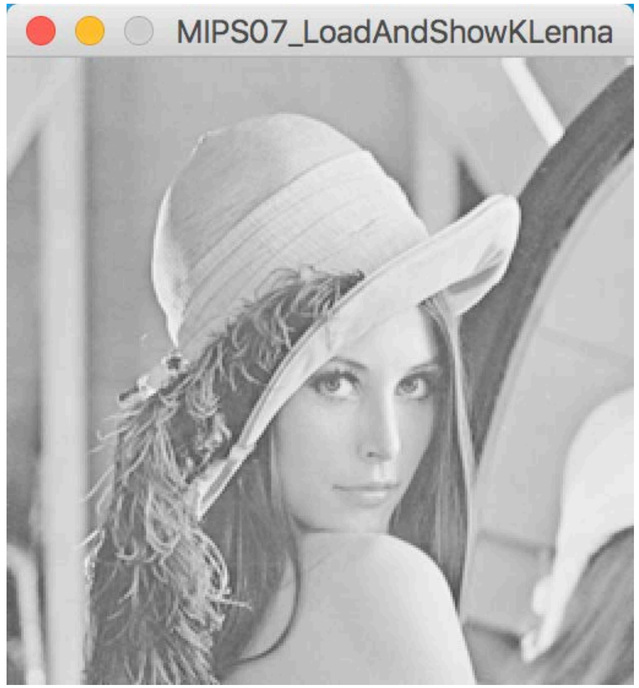


この他256正方のモノクロデータ（その2）、512正方のモノクロデータ、1024正方のモノクロデータがある。

2021年5月7日時点で有効な国内URL

http://www.ess.ic.kanagawa-it.ac.jp/app_images_j.html

Processingで画像ファイルを読み込む




```
PImage img;
```

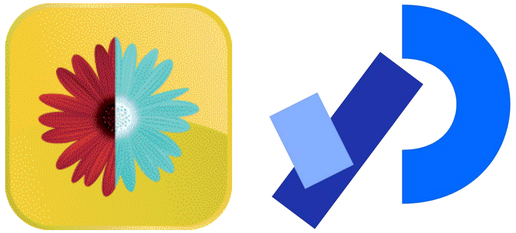
画像データを格納するクラスのデータを宣言

```
void setup() {
  size(256, 256);
  img = loadImage("mono/LENNA.bmp");
}
```

画像データをファイルから読み込む

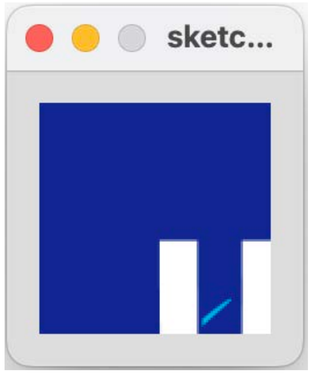
```
void draw() {
  background(0);
  image(img, 0, 0);
}
```

`img`に格納されている画像を表示



Processingで画像を扱う

loadImageではネットワーク上の画像を読み込むこともできる




```
PImage webImg;

void setup() {
  String url = "https://www.uec.ac.jp/common_new/images/
                                pct_site-logo_pc.png";
  webImg = loadImage(url, "png");
}

void draw() {
  background(255);
  image(webImg, 0, 0);
}
```

画像データのURL（文字列型）

画像データの場所と形式を指定

Processingで画像を扱う

PImageクラス　画像データを格納する

フィールド fields

pixels[]
　各画素のピクセル値を保持する配列

width
　画像の横幅（ピクセル数）

height[]
　画像の高さ（ピクセル数）

読み込み、表示、生成

loadImage()
　画像データを読み込む

image()
　表示窓に画像を表示する

createImage()
　画像データを生成する

メソッド　methods

loadPixels()
　配列 **pixels[]**　に画像データを読み込む

updatePixels()
　配列 **pixels[]**に格納されているピクセルデータを用いて表示画像を更新する

save()
　表示画像データを画像ファイルに保存する

…

Processingで画像を扱う

画像処理の基本的枠組

```
PImage source;                          元画像
PImage destination;                     処理後画像

void setup() {
  size(256, 256);                                          元画像のサイズ
  source = loadImage("source.bmp");                        元画像をファイルから読み込む
  destination = createImage(source.width, source.height, RGB);
                                          元画像と同じサイズのRGB画像データを生成
  image(source, 0, 0);   元画像を画面に表示
  source.loadPixels();
                         元画像の画素値を配列 pixel[] に読み込む

  for(int y = 0; y < source.height; y++) {
    for(int x = 0; x < source.width; x+) {          ラスタスキャンの順に処理
      int p = x + y * source.width;
      //
      // 目的の処理に応じて処理後画像のRGB値（r, g, b）を計算
      //
      destination.pixels[p] = color(r, g, b);       計算結果を処理後画像の画素値に設定
    }
  }
  image(destination, 0, 0);                                処理後の画像値を表示
  save("data/destination.bmp");                            表示されている画像をファイルに保存
}

void draw() {  draw()は空
}
```

画素ごとの処理 (1)

元画像

明るさ変更

階調反転

左右反転

回転

画像の明るさの変更

アルゴリズム（擬似コード風）

画素値によらず一律に一定値を加算（減算）する場合

a(x,y)

b(x,y)

画素値に一定値を加算して明るくする

明るくする

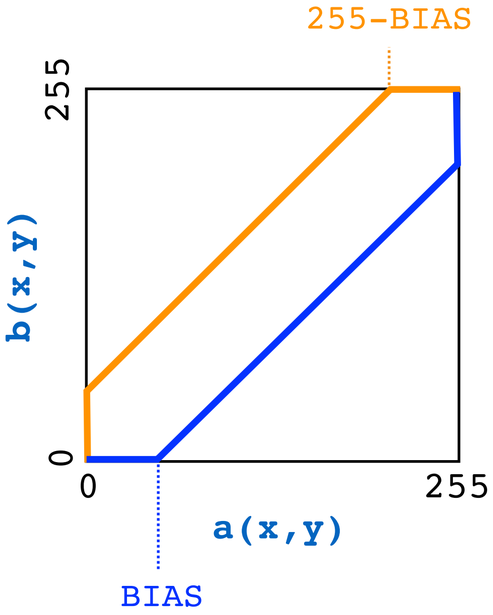
```
b(x,y)=a(x,y)+BIAS;
if (b(x,y) > 255) {
  b(x,y) = 255;
}
```

255を超えた計算結果は一律255に。

暗くする

```
b(x,y)=a(x,y)-BIAS;
if (b(x,y) < 0) {
  b(x,y) = 0;
}
```

0未満の計算結果は一律0に。



画像の明るさの変更

画素値によらず一律に一定値を加算（減算）する場合

```
for(int y = 0; y < src.height; y++) {
  for(int x = 0; x < src.width; x++) {
    int p = x + y * src.width;
    int r = int(red(src.pixels[p])+BIAS)<=255 ? int(red(src.pixels[p])+BIAS) : 255;
    int g = int(green(src.pixels[p])+BIAS)<=255 ? int(green(src.pixels[p])+BIAS) : 255;
    int b = int(blue(src.pixels[p])+BIAS)<=255 ? int(blue(src.pixels[p])+BIAS) : 255;
    destination.pixels[p] = color(r, g, b);
  }
}
```

test ? expression1 : expression2

testの評価値がtrueならばexpression1を、falseならばexpression2の評価値を値とする。

画像の明るさの変更

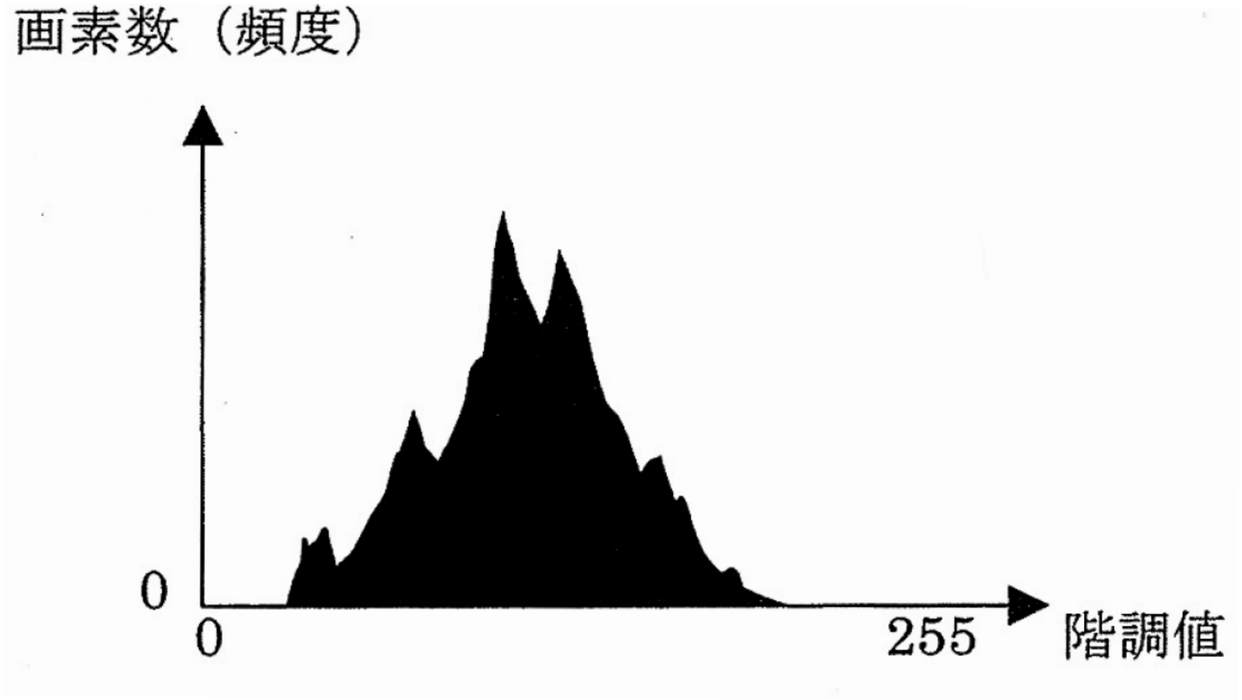
画素値によらず一律に一定値を加算（減算）する場合

```
for(int y = 0; y < src.height; y++) {
  for(int x = 0; x < src.width; x++) {
    int p = x + y * src.width;
    int r = int(constrain(red(source.pixels[p])+bias, 0, 255));
    int g = int(constrain(green(source.pixels[p])+bias, 0, 255));
    int b = int(constrain(blue(source.pixels[p])+bias, 0, 255));
    destination.pixels[p] = color(r, g, b);
  }
}
```

constrain(amt, low, high)

amt の値が最小値 **low** と最大値 **high** を超えないように制限。

画素ごとの処理（2）階調補正

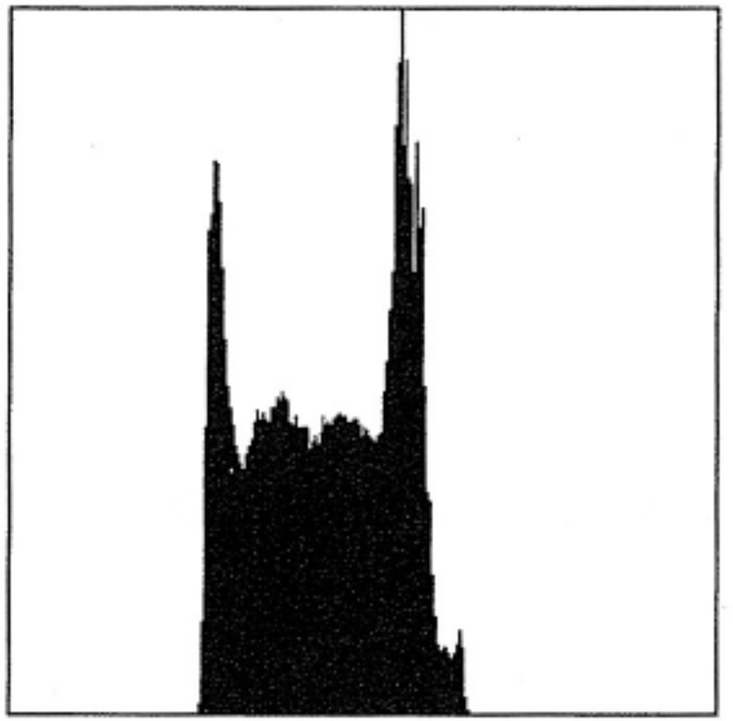


画像の濃度ヒストグラム

- 画像中の階調値の変動幅や分布を知ることができる
- （例）グラフの形状が双峰性 → 明部と暗部の差が大きい画像

画像の濃度ヒストグラムの例

(a) 原画像

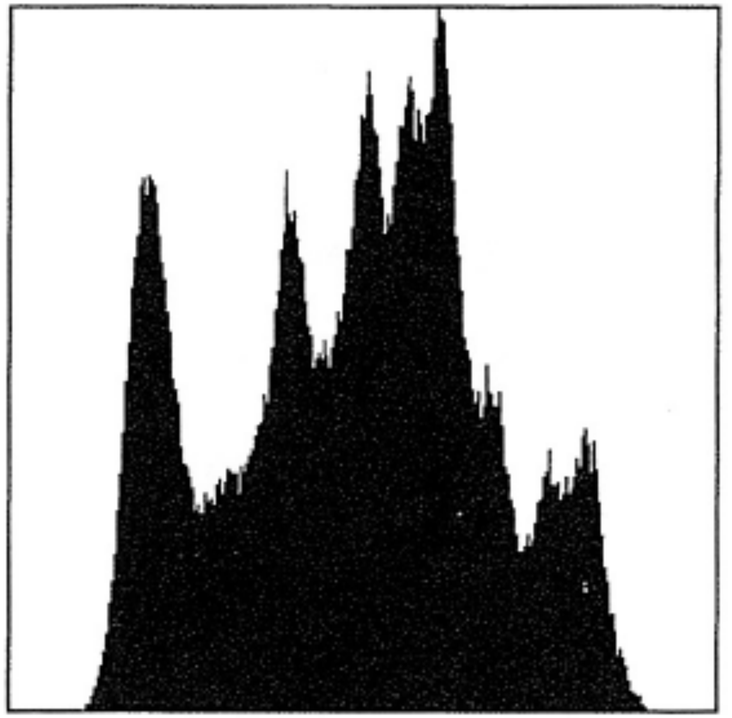


(b) 濃度ヒストグラム

コントラストが低い画像 → 階調値の分布が狭い範囲に限定

画像の濃度ヒストグラムの例

(a) 原画像

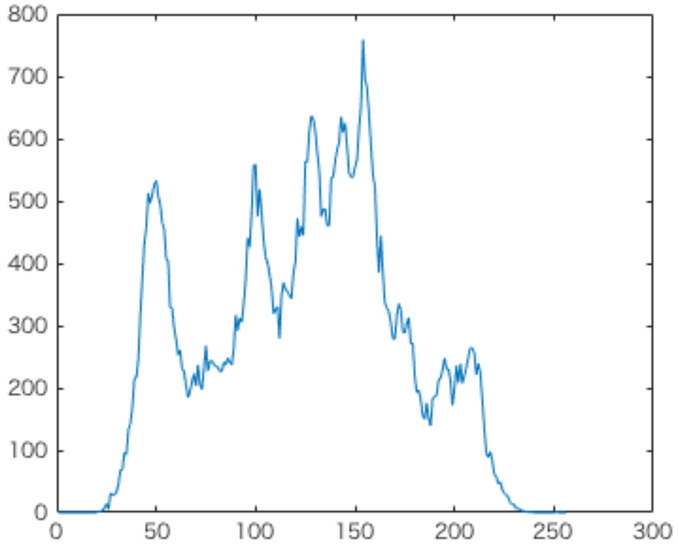


(b) 濃度ヒストグラム

コントラストが明瞭 → 階調値の分布は広い

画像の濃度ヒストグラムの作成

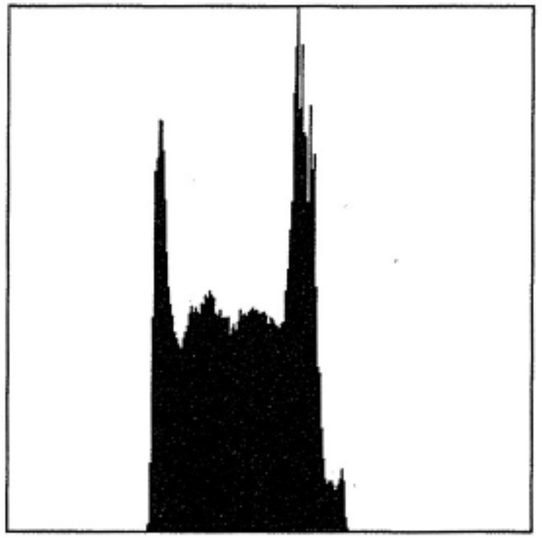
アルゴリズム（擬似コード風）




```
for (k=0; k<=255; k++) {
  hist(k) = 0;
}
for (x=0; x<=255; x++) {
  for (y=0; y<=255; y++) {
    bin = a(i,j)+1;
    hg(bin)=hg(bin)+1;
  end
end
plot(hg);
```

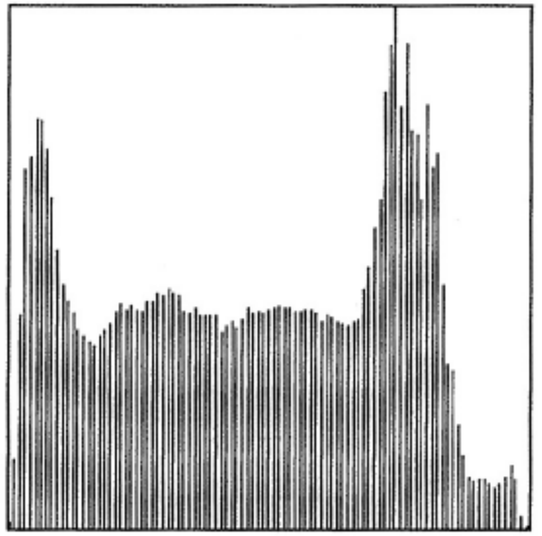
階調補正・濃度補正

(a) 変換前の画像（原画像）



(b) 左の画像の濃度ヒストグラム

(c) 変換後の画像



(d) 左の画像の濃度ヒストグラム

濃度ヒストグラムの線形変換

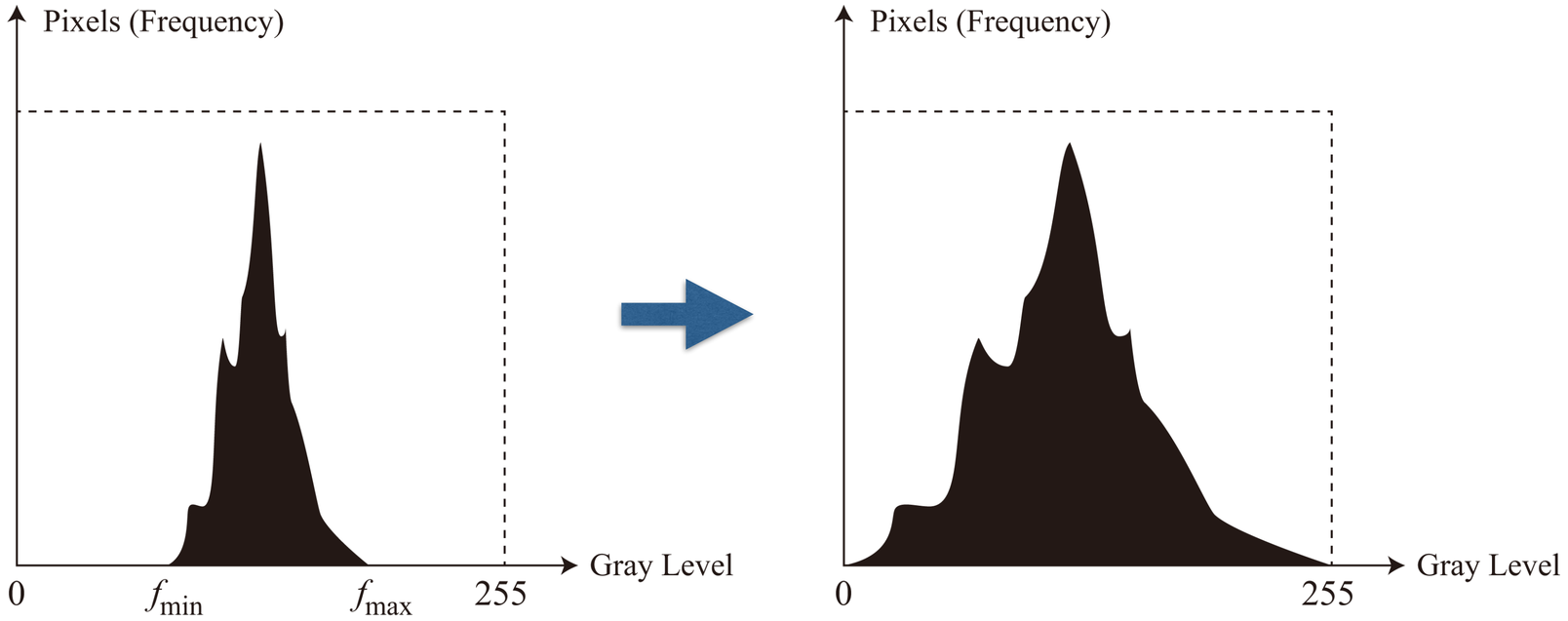
$$g = \frac{f - f_{\min}}{f_{\max} - f_{\min}} \times 255$$

$f_{\min}$：最小階調値

$f_{\max}$：最大階調値

f：変換前の階調値

g：変換後の階調値

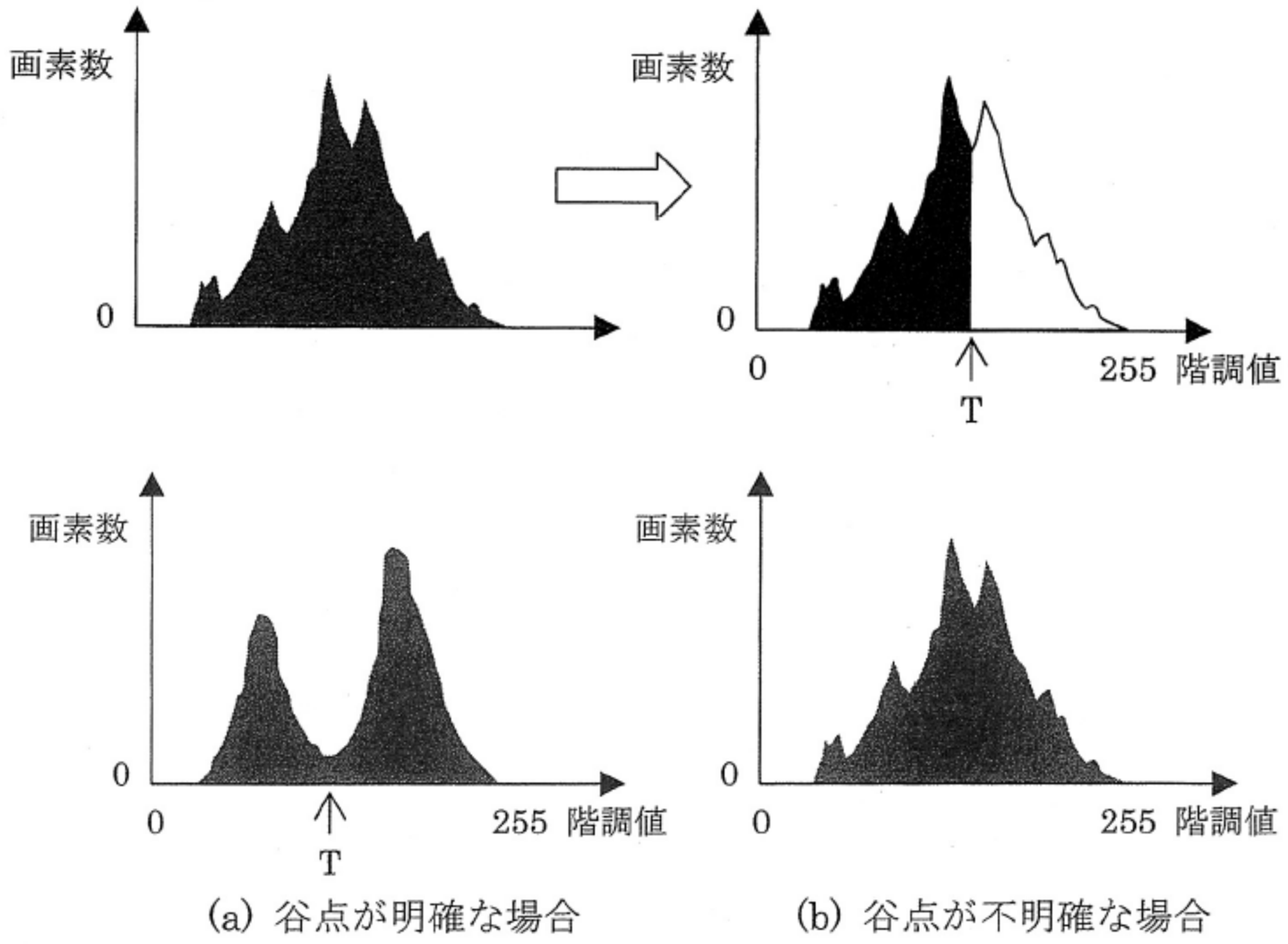


濃度ヒストグラムの線形変換

アルゴリズム（擬似コード風）

```
fmax=0;
fmin=255;
for i=0:255
  for j=0:255
    if fmax < a(i,j)
      fmax = a(i,j);
    end
    if fmin > a(i,j)
      fmin = a(i,j);
    end
  end
end
for i=0:255
  for j=0:255
    f=single(a(i,j));
    g=single(255.0*(f-single(fmin))/(single(fmax)-single(fmin)));
    b(i,j)=round(g);
  end
end
```

2値化処理



(a) 谷点が明確な場合　(b) 谷点が不明確な場合

２値画像

各画素の階調が０または255だけの画像

もっとも単純な２値化

閾値を境に０と255に変更

2値化処理

原画像

$T = 40$

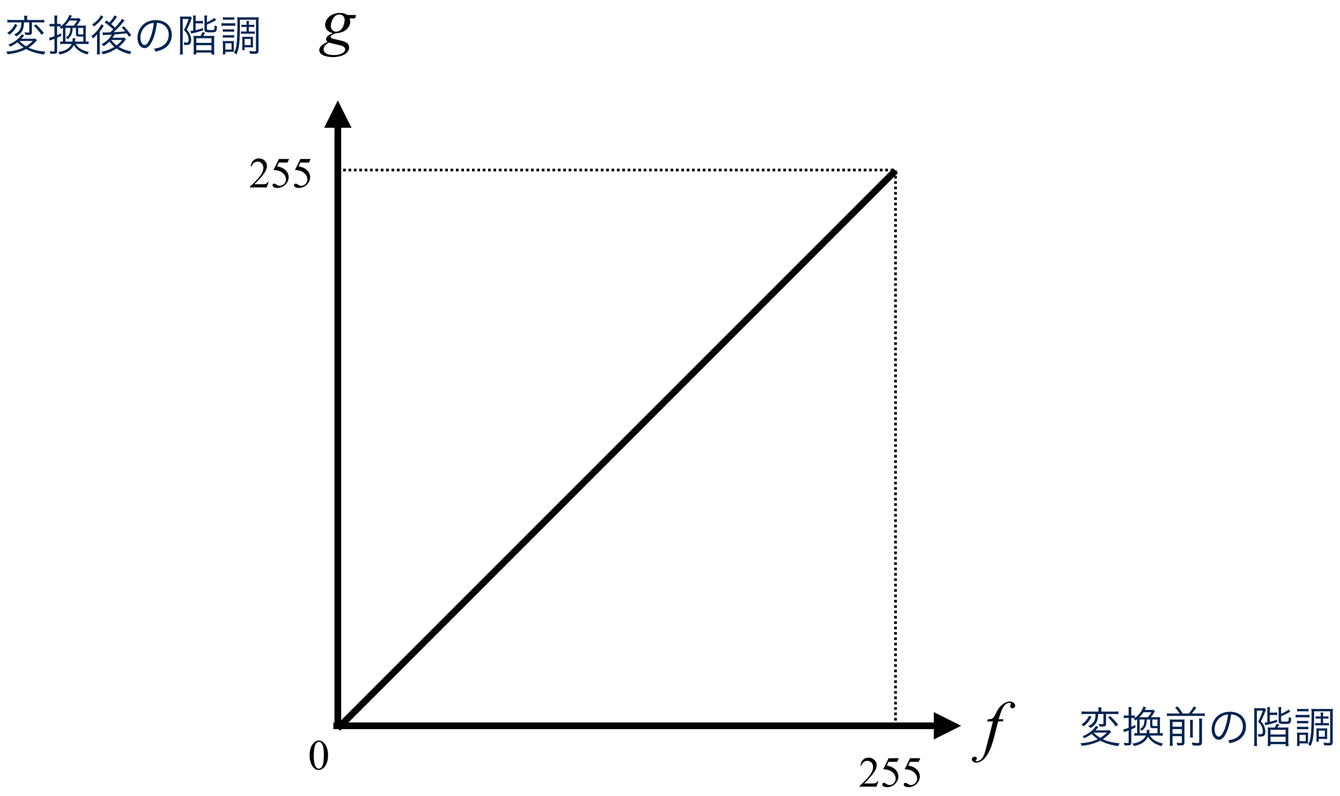
$T = 80$

$T = 120$

$T = 160$

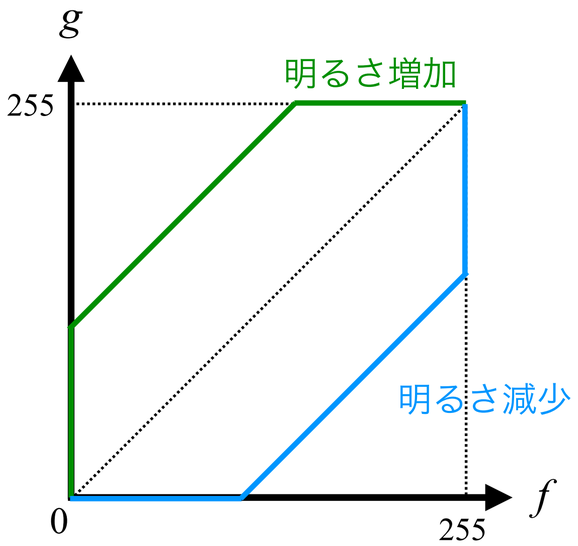
$T = 200$

変換グラフを用いた階調変換

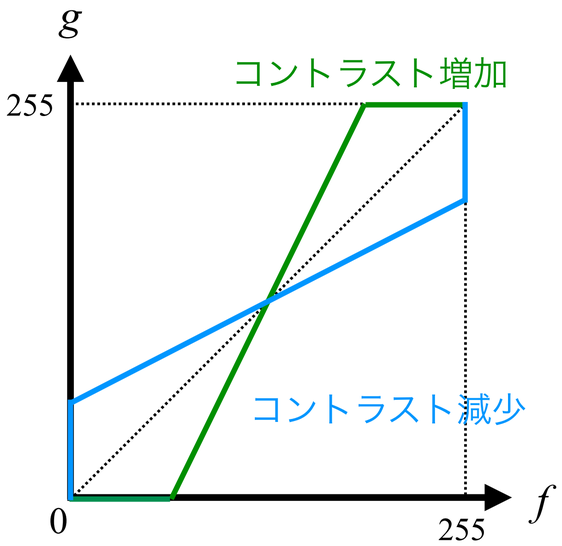


原画像の階調値 f と変換後の画像の階調値 g の関係を表す

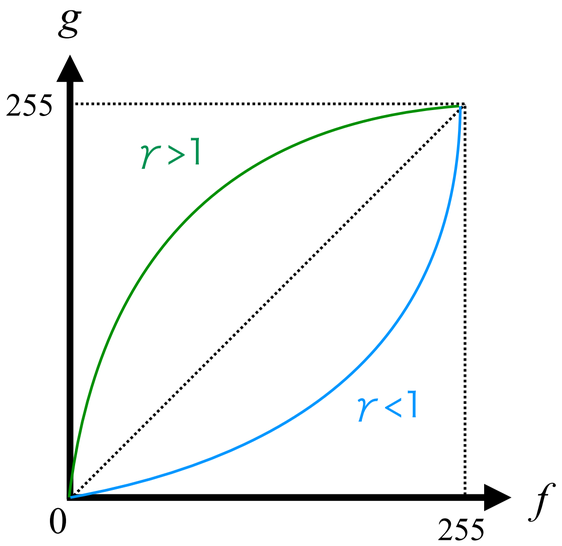
変換グラフを用いた階調補正の例



明るさ補正



コントラスト補正



γ補正

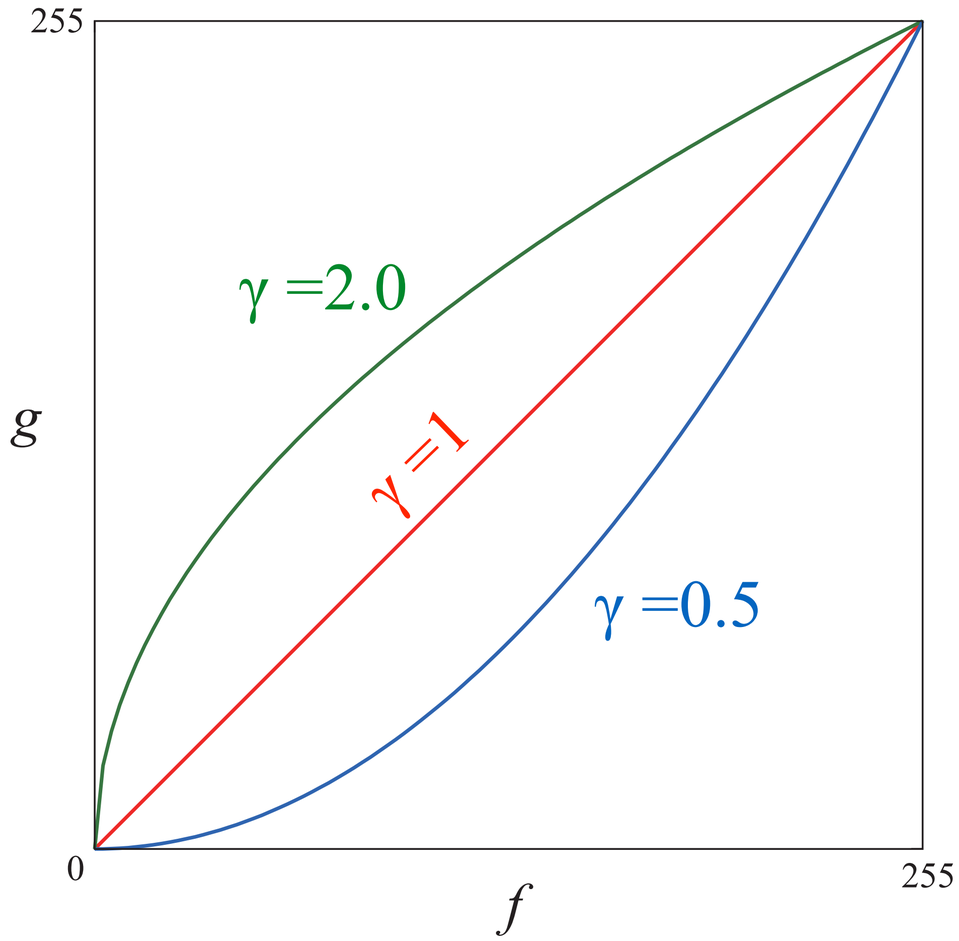
γ補正

$$g = 255.0 \times \left(\frac{f}{255.0} \right)^{\frac{1}{\gamma}}$$

f: 元画像の画素値

g : 変換画像の画素値

γ : 補正係数



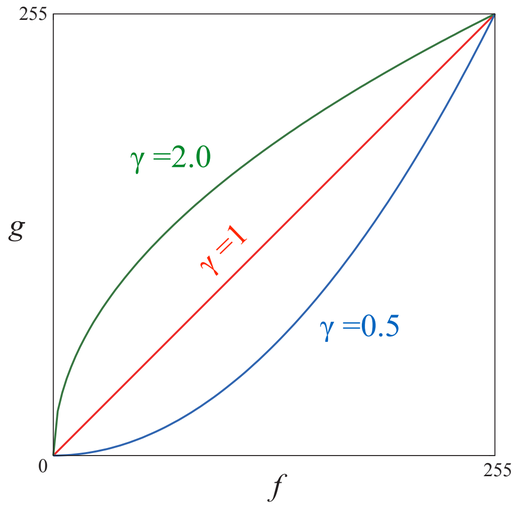
γ補正

原画像 γ=1.0

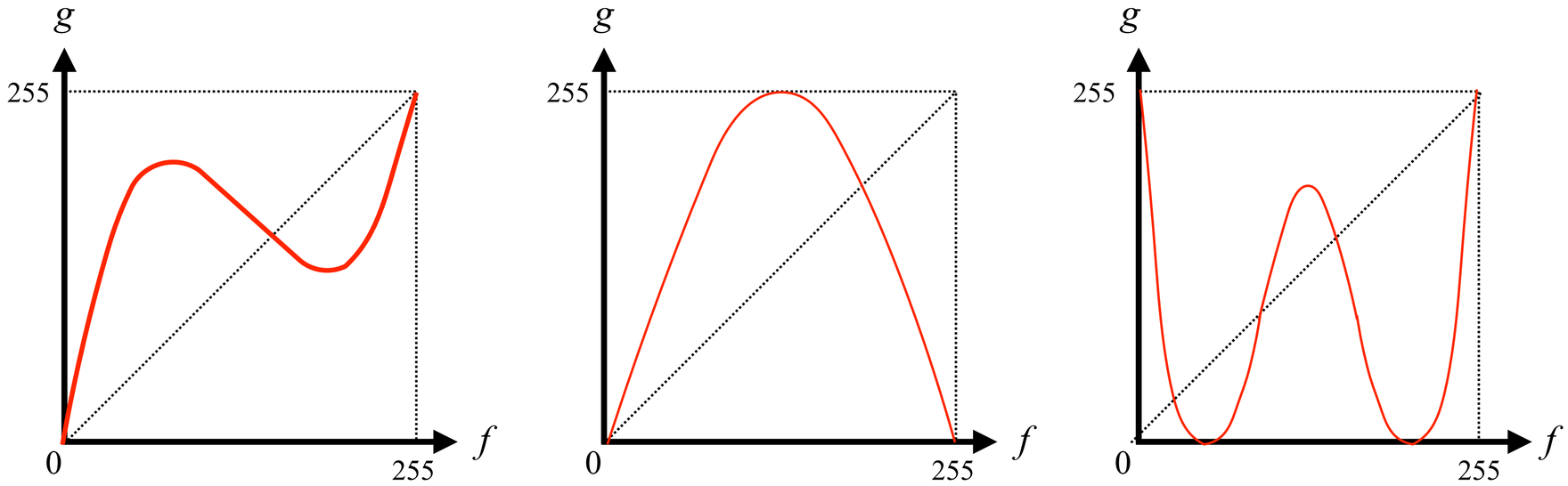
変換画像 γ=0.5

変換画像 γ=2.0

$$g = 255.0 \times \left(\frac{f}{255.0} \right)^{\frac{1}{\gamma}}$$

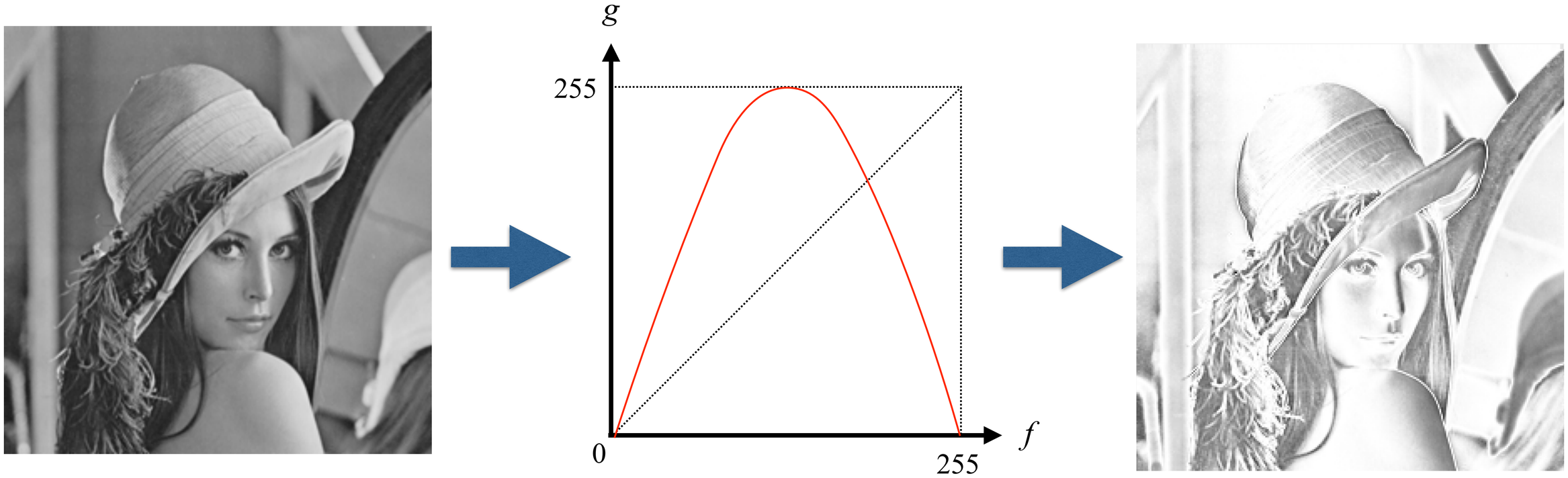


曲線による階調変換



- このような様々な曲線を変換グラフとして用いたとき、原画像はどのように変換されるか？
- 曲線としては、f を決めたとき、g が一意に決まるものであればなんでもよい。

曲線による階調変換



ソラリゼーション（solarisation、陽光効果）風の効果が得られる。

曲線による階調変換

アルゴリズム（擬似コード風）

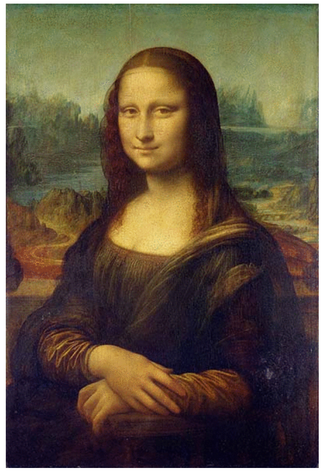
```
for i=0:255
  for j=0:255
    f=single(a(i,j));
    g=single((-(f-128)*(f-128)/
                        64.0+255.0)-1.0);
    b(i,j)=round(g);
  end
end
```

画像処理 (1) 画素ごとの濃淡変換

今日の要点

1. 画像はラスタスキャン（左上原点、右下最後）
2. 画素ごとの処理
    1. 階調変更、階調反転、左右反転、回転
    2. 階調補正：ヒストグラム、２値化処理、変換曲線、γ補正
3. Processingで画像処理
    1. 標準画像データベースSIDBA
    2. 画像データを画像ファイルから読み込み、ピクセル配列を扱う
    3. 画像処理のためのクラス、関数、など

        `PImage, loadImage, createImage, image, save,`
        `loadPixels, pixels[], .width, .height, updatePixels,`
        `constrain,` etc.



Processingでオブジェクト指向プログラミング

クラスとインスタンス（class, instance）

【例題1】マウスがクリックされるたびにランダムな位置に複数の円を描く

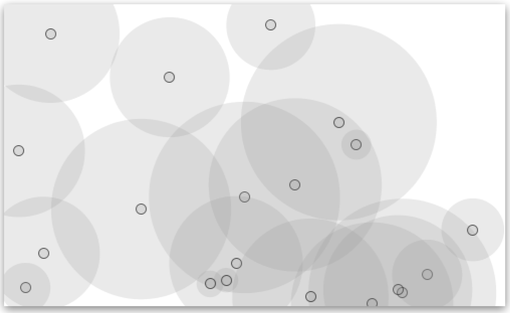
```
int _num = 10;

void setup() {
  size(500,300);
  background(255);
  smooth();
  strokeWeight(1);
  fill(150, 50);
  drawCircles();
}

void draw() {
}

void mouseReleased() {
  drawCircles();
}
```

```
void drawCircles() {
  for (int i=0; i<_num; i++) {
    float x = random(width);
    float y = random(height);
    float radius = random(100) + 10;
    noStroke();
    ellipse(x, y, radius*2, radius*2);
    stroke(0, 150);
    ellipse(x, y, 10, 10);
  }
}
```



円は、スクリーンにばらまかれて、そこに残されるだけ😟

Processingでオブジェクト指向プログラミング

クラスとインスタンス（class, instance）

【例題2】自分自身の属性をカプセル化した円オブジェクトを作成

```
class Circle {
  float x, y;
  float radius;
  color linecol, fillcol;
  float alph;

  Circle () {
    x = random(width);
    y = random(height);
    radius = random(100) + 10;
    linecol = color(random(255), random(255), random(255));
    fillcol = color(random(255), random(255), random(255));
    alph = random(255);
  }
}
```

オブジェクトの属性

オブジェクトのコンストラクタ

クラスを生成した時に実行される関数

オブジェクトの色とアルファ値

Circleクラス

再利用できるテンプレート。

circle クラスの「インスタンス」として円を作ればよい。

Processingでオブジェクト指向プログラミング

クラスとインスタンス（class, instance）

```
void drawCircles() {
  for (int i=0; i<_num; i++) {
    Circle thisCirc = new Circle();
  }
}
```

- **new** を使ってコンストラクタを呼び出し、実際のオブジェクト（インスタンス）を **_num** 個生成する。

- オブジェクトを生成しただけで、描いてはいないことに注意。

Processingでオブジェクト指向プログラミング

クラスとインスタンス（class, instance）

自分自身を画面に表示するためのメソッドを追加

```
class Circle {
  ...
  Circle () {
    ...
  }
  void drawMe() {
    noStroke();
    fill(fillcol, alph);
    ellipse(x, y, radius*2, radius*2);
    stroke(linecol, 150);
    noFill();
    ellipse(x, y, 10, 10);
  }
}
```

円の縁は描かない
塗りつぶし色と透明度を指定
円を描く
円の中心を示す小さな円を描く

Circleクラスの drawMe() メソッド

Processingでオブジェクト指向プログラミング

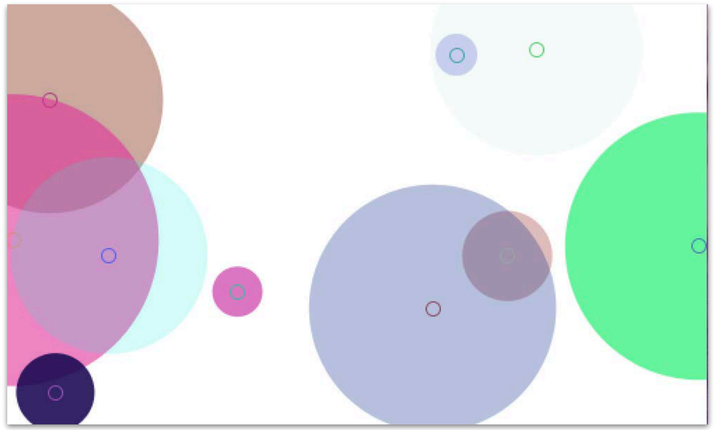
クラスとインスタンス（class, instance）

drawMe()メソッドを呼び出すように**drawCircles**関数を修正

```
void drawCircles() {
  for (int i=0; i<_num; i++){
    Circle thisCirc = new Circle();
    thisCirc.drawMe();
  }
}
```

Circleクラスを用いて

スクリーンに円を描くことができた 😀



Processingでオブジェクト指向プログラミング

クラスとインスタンス（class, instance）

【例題3】動きのあるオブジェクト指向の円

✓ すべての円をしまっておくための配列を作成する

✓ それぞれの円に x、y 方向の動きを与える

✓ それぞれの円に、**`draw`** ループでフレームごとに呼ばれる
`updateMe` メソッドを与える

Processingでオブジェクト指向プログラミング

クラスとインスタンス（class, instance）

【例題3】動きのあるオブジェクト指向の円

```
int _num = 10;
Circle[] _circleArr = {};                                    円の配列を定義

void setup()...
void draw()...
void mouseReleased()...
void drawCircles() {
  for (int i=0; i<_num; i++) {
    Circle thisCirc = new Circle();
    thisCirc.drawMe();
    _circleArr = (Circle[])append(_circleArr, thisCirc);
  }                                                          オブジェクトを配列に加える
}
```

すべての円をしまっておくための配列を作成する

Processingでオブジェクト指向プログラミング

クラスとインスタンス（class, instance）

【例題3】動きのあるオブジェクト指向の円

```
class Circle {
  float x, y;
  float radius;
  color linecol, fillcol;
  float alph;
  float xmove, ymove;

  Circle () {
    x = random(width);
    y = random(height);
    radius = random(100) + 10;
    linecol = color(random(255), random(255), random(255));
    fillcol = color(random(255), random(255), random(255));
    alph = random(255);
    xmove = random(10) - 5;
    ymove = random(10) - 5;
  }
```

全てのフレームでステップを進める

ランダムなステップ

それぞれの円に x、y 方向の動きを与える

Processingでオブジェクト指向プログラミング

クラスとインスタンス（class, instance）

【例題3】動きのあるオブジェクト指向の円

```
void updateMe() {
  x += xmove;
  y += ymove;
```

フレームごとに動かす

```
  if (x > (width+radius)) { x = 0 - radius; }
  if (x < (0-radius)) { x = width+radius; }
  if (y > (height+radius)) { y = 0 - radius; }
  if (y < (0-radius)) { y = height+radius; }
```

スクリーンの端から出ないようにする

```
  drawMe();
}
```

描く

それぞれの円に、drawループでフレームごとに呼ばれるupdateMeメソッドを与える

Processingでオブジェクト指向プログラミング

ローカルな知識

【例題4】衝突判定

```
void updateMe() {
  ...

  boolean touching = false;
  for (int i=0; i<_circleArr.length; i++) {
    Circle otherCirc = _circleArr[i];
    if (otherCirc != this) {
      float dis = dist(x, y, otherCirc.x, otherCirc.y);
      if ((dis - radius - otherCirc.radius) < 0) {
        touching = true;
        break;
      }
    }
  }
```

円の配列を調べる

他の円であれば

距離計算

衝突していれば
判定を`true`に設定

衝突判定

Processingでオブジェクト指向プログラミング

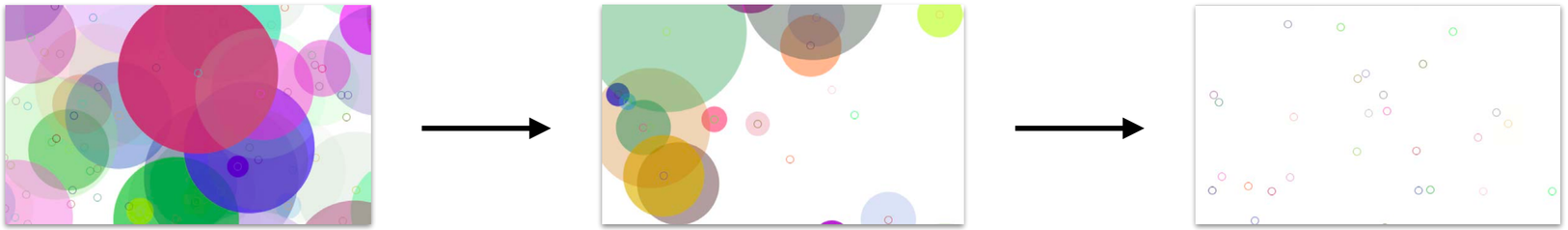
ローカルな知識

【例題4】衝突判定

```
void updateMe() {
  ...
  if (touching) {
    if (alph > 0) { alph--; }
  } else {
    if (alph < 255) { alph += 2; }
  }

  drawMe();
}
```

衝突していれば徐々に消える、
そうでなければ不透明度を増す



最初は不透明度が高かった円が徐々に透明になってゆく😉