Graphics with Processing

2012-04 色彩とピクセル処理

http://vilab.org

塩澤秀和

4.1 色彩

色のデータ形式

- 色の指定方法
  - 1つの数値（グレースケール）
  - 3つの数値の組（カラー）
    初期モードは RGB 各0～255
  - 16進数カラーコード #rrggbb
  - color型の変数
- color型
  - 色を表すデータ型（実態はint）
  - color関数で合成できる
    color(成分1, 成分2, 成分3)
  - 例）color c = color(r, g, b);
- 成分の取得
  - red(c), green(c), blue(c), hue(c), saturation(c), brightness(c), alpha(c)

半透明の表現

- アルファ値（p.225）
  - 色の第4成分（透過処理用）
  - 重ね塗りでの色の混合率
  - 例）c = color(r, g, b, a);
  - 例）fill(255, 0, 0, 128);

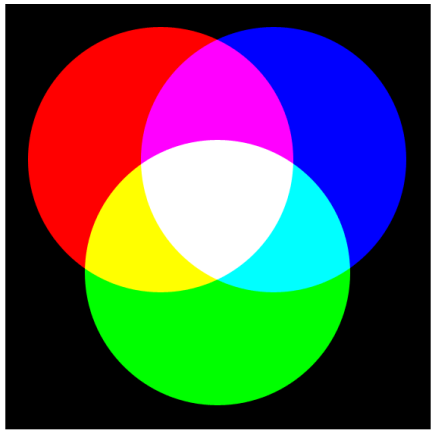
色モードの設定

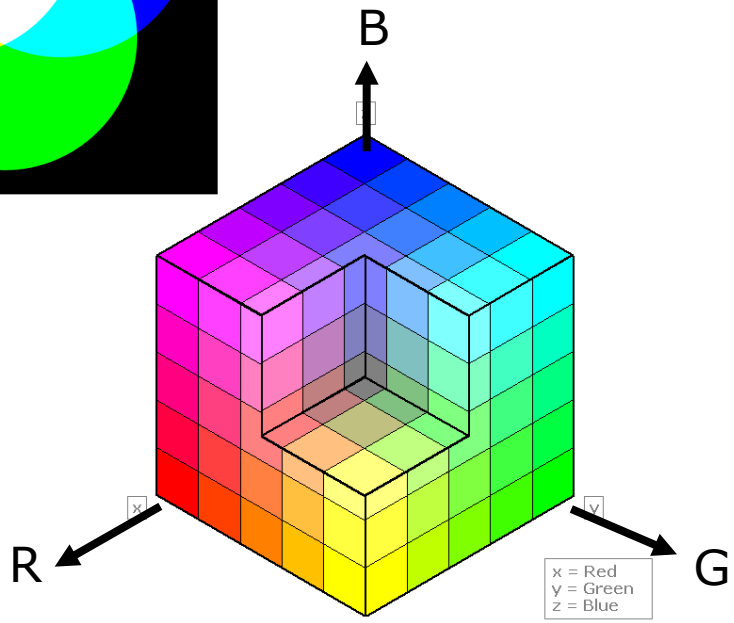
- colorMode(モード, 値範囲)
  - モード：カラーモデル
    RGB または HSB
  - 値範囲：成分の上限値
    - colorMode(モード, 範囲1, 範囲2, 範囲3) の形式もある
  - 例）colorMode(HSB, 1.0);
  - サンプル Basics → Color

4.2 表色系/カラーモデル(p.201)

RGBカラーモデル

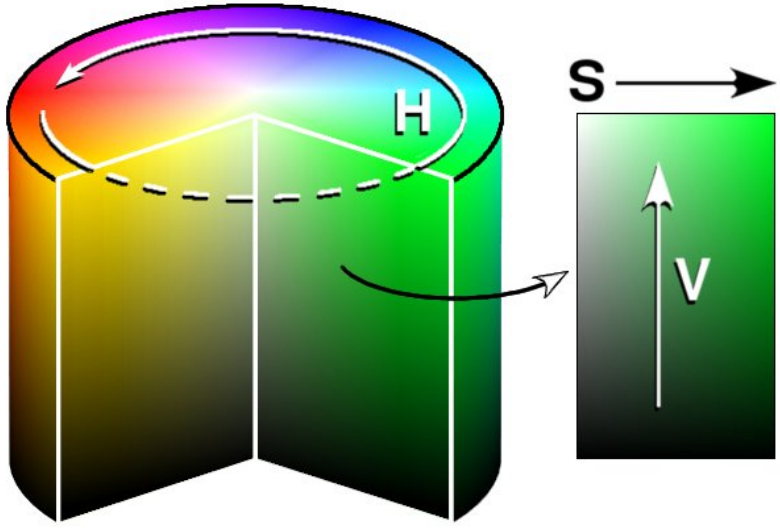
- 光の三原色(赤, 緑, 青)





HSB(HSV/HSI)カラーモデル

- 光の三属性
  - 色相(H):色あい
  - 彩度(S):あざやかさ
  - 明度(B/V/I):明るさ
  - メニュー Tools → Color Selector



図はWikipediaから

4.3 ピクセル処理

ピクセル配列による操作

- ピクセルとは（p.11）
  - 画面を構成する画素1点1点（pixel ← picture cell）

  ⇒ ラスター表現のグラフィックス
- pixels[]
  - 各画素の色（color型のデータ）を格納する1次元配列
  - 画面座標(x, y)の要素は pixels[y * width + x]
- loadPixels()
  - ピクセル処理の開始処理
  - 画面の画素ごとの色データをpixels[]に読み込む
- updatePixels()
  - pixcels[]を画面に書き戻す

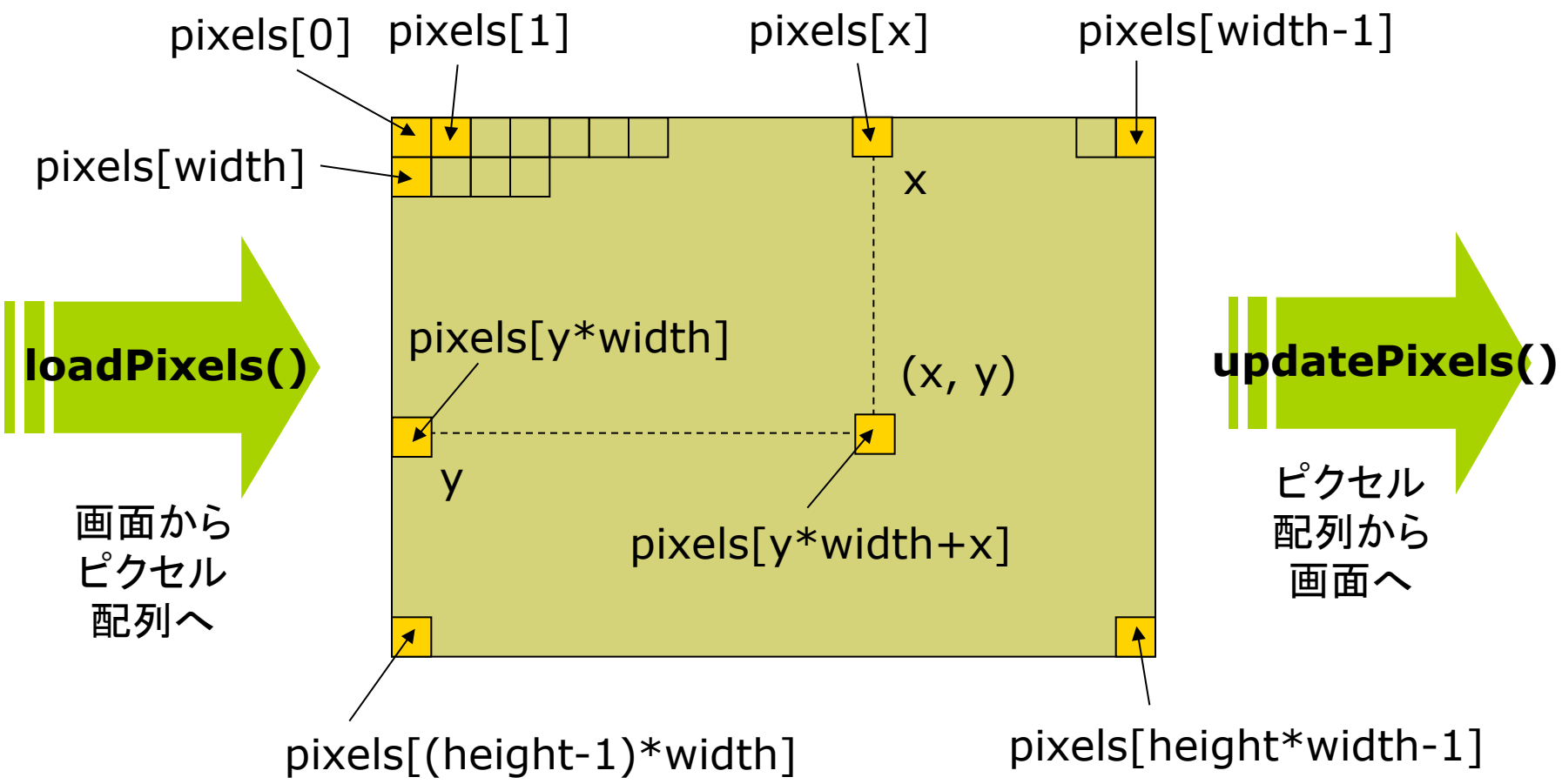
ピクセル配列の操作

- ピクセルの読み出し
  - color c;
  - c = pixels[y * width + x];
- ピクセルの書き込み
  - pixels[y * width + x] = c;

ピクセル配列を使わない操作

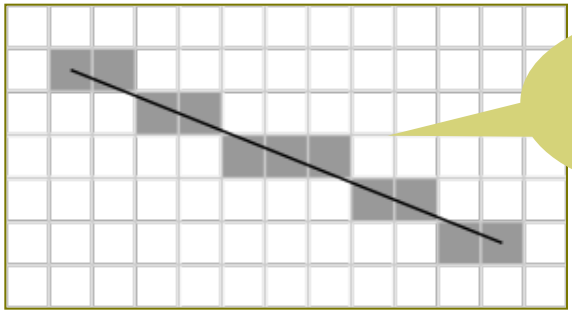
- copy(x1, y1, w1, h1, x2, y2, w2, h2)
- copy(画像, $x_{画像}$, $y_{画像}$, $w_{画像}$, $h_{画像}$, x, y, w, h)
  - 領域や画像からのコピー
- get(), get(x, y, 幅, 高さ)
  - 表示内容を画像として取得

4.4 ピクセル配列


pixels[0]
pixels[1]
pixels[x]
pixels[width-1]
pixels[width]
x
loadPixels()
pixels[y*width]
(x, y)
updatePixels()
画面から
ピクセル
配列へ
y
pixels[y*width+x]
ピクセル
配列から
画面へ
pixels[(height-1)*width]
pixels[height*width-1]

4.5 ラスター化

- ラスター化（p.207）
  - ベクター表現（座標とパラメータ）の図形を，画素の集合に変換



直線のラスター化の例

  - このサンプルをさらに高速化
    ⇒ ブレゼンハムのアルゴリズム

```
void draw() {
  background(0);
  loadPixels();
  if (mouseX > mouseY)
    pxline(0, 0, mouseX, mouseY);
  updatePixels();
}
```

```
void pxline(int x1, int y1,
            int x2, int y2)
{
  color c = color(255, 255, 255);

  float d = (float)(y2-y1)/(x2-x1);
  float e = 0.0;

  int x = x1, y = y1;
  while (x <= x2) {

    pixels[y * width + x] = c;

    x++;
    e += d;
    if (e >= 0.5) {
      e -= 1.0;
      y++;
    }
  }
}
```

xが1増えるあたりのyの増分（小数）

本来のy座標との累積誤差

画素設定

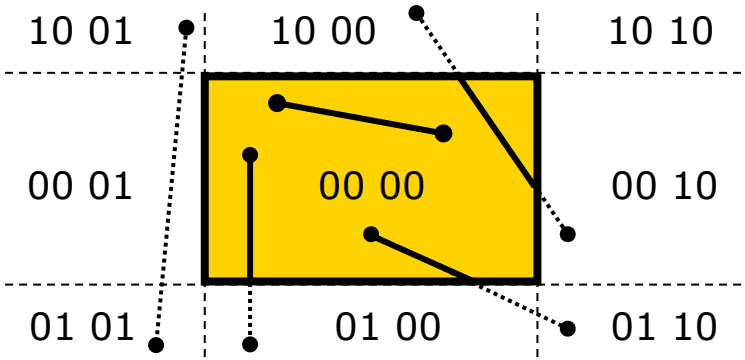
xが1増えるごとにyの誤差はd増加

yの誤差が0.5以上になったらyを1増やす

4.6 クリッピング

クリッピング（p.44）

- クリッピングとは
  - それぞれの図形の表示領域（ビューポート）からはみ出した部分を描画しない処理
  - 図形の種類ごとに，効率のよい方法が開発されている
- 線分のクリッピング
  - コーエン・サザランドの方法
  - ビット演算で直線（線分）が表示領域にかかるか高速に判定



- アルゴリズム
  - 線分の両端が表示領域の上下右左にはみ出しているかを，4桁のビットコードで表す
  - 両端点がともに0000になったら，線分の全部が表示領域内
  - そうでないなら，両端点のコードのビットごとの論理積を計算
    例：1001 & 0101 = 0001
  - 結果が0000以外なら，線分の全部が表示領域外
  - 0000なら，線分の一部が表示領域にかかっている
  - その場合，ビットコードから線分がどちら側にはみ出しているかが分かるので，線分と境界線との交点を求め，それを新しい端点として再判定する

4.7 演習課題

準備課題

- 右のプログラムに適切な setup 関数を補って，実行してみなさい
  - マウスをドラッグしてみなさい
  - さらに，★の範囲を
    if (random(1.0) < 0.2)
    というif文で囲んでみなさい

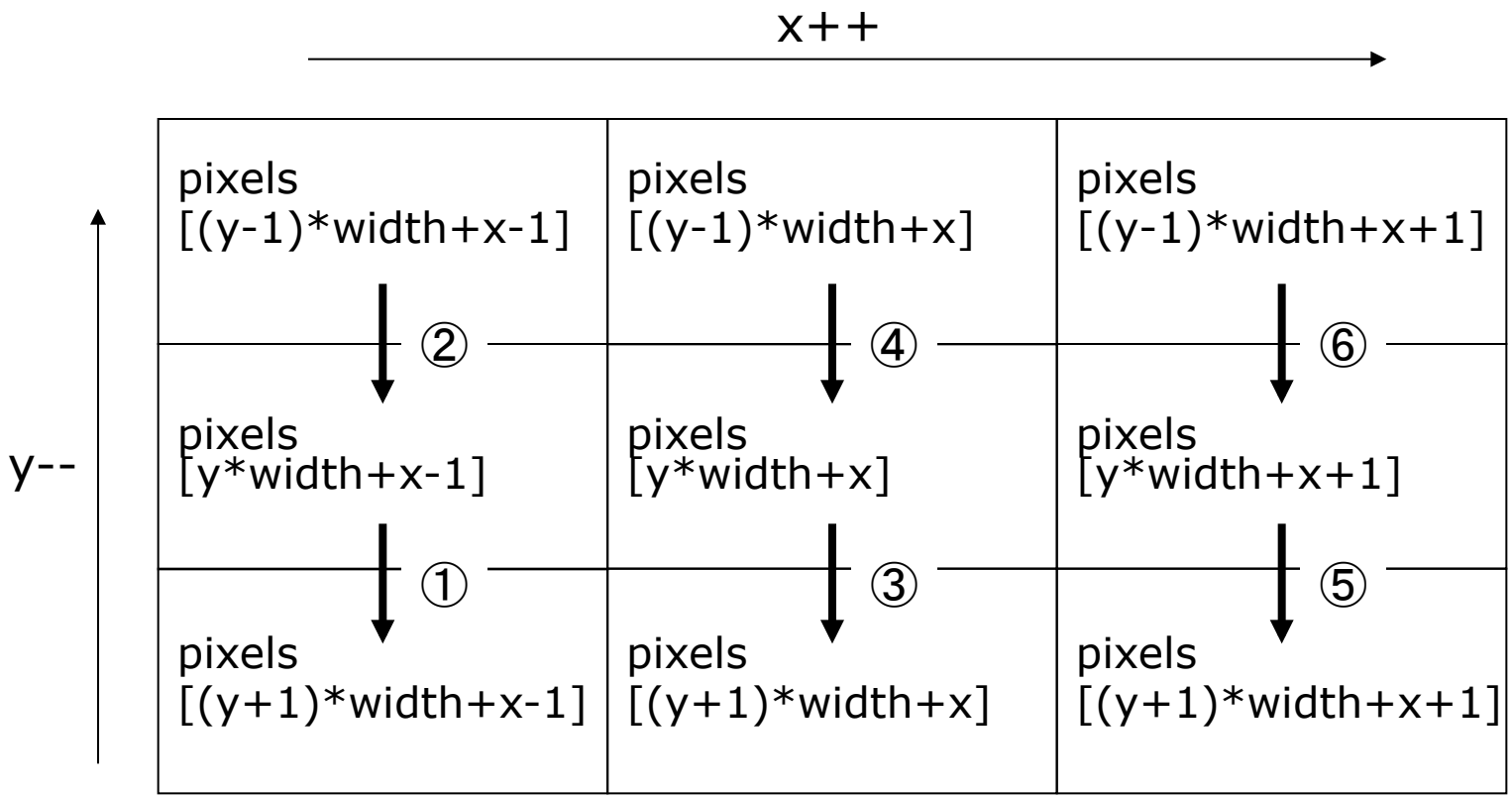
提出課題

- 上から下に流れる模様を，下から上に流れるように変更しなさい
  - **下→上以外のものは認めない**
  - できたら，円のかわりに画像を表示させてみるとよい
- ヒント
  - (x,y+1)を(x,y)にコピーする
  - yのfor文も変える必要がある

```
void draw() {
  color c;
  loadPixels();
  for (int x = 0; x < width; x++) {
    // ★
    for (int y = height-1; y > 0; y--) {
      // (x,y-1)の色を(x,y)にコピー
      c = pixels[(y - 1) * width + x];
      pixels[y * width + x] = c;
    }
    // 最上行には新しい色を入れる
    pixels[0 * width + x] =
        color(0, 0, frameCount % 256);
    // ★
  }
  updatePixels();

  if (mousePressed) {
    noStroke();
    fill(255, 220, 220, 200);
    ellipse(mouseX, mouseY, 20, 20);
  }
}
```

4.8 演習課題のヒント




```
c = pixels[(y-1)*width + x];
pixels[y*width + x] = c;
```

の意味と処理の順序をよく考えてください