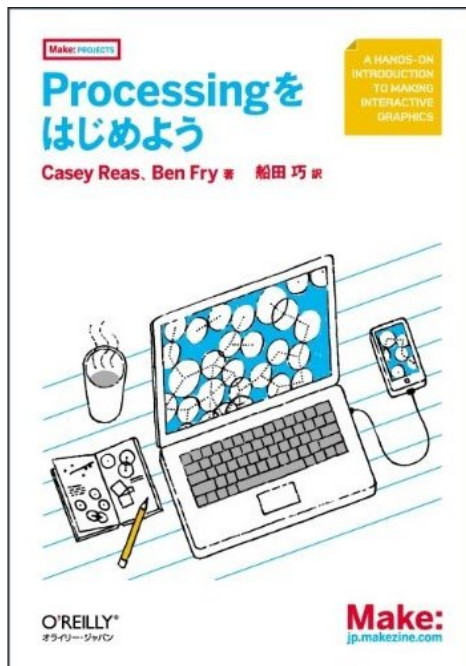


Practice #4

多次元のデータを扱う

演習4B PVector (位置)

演習4C 画像



Processingをはじめよう
第二版
(Make: PROJECTS)

オンデマンド授業 7.24

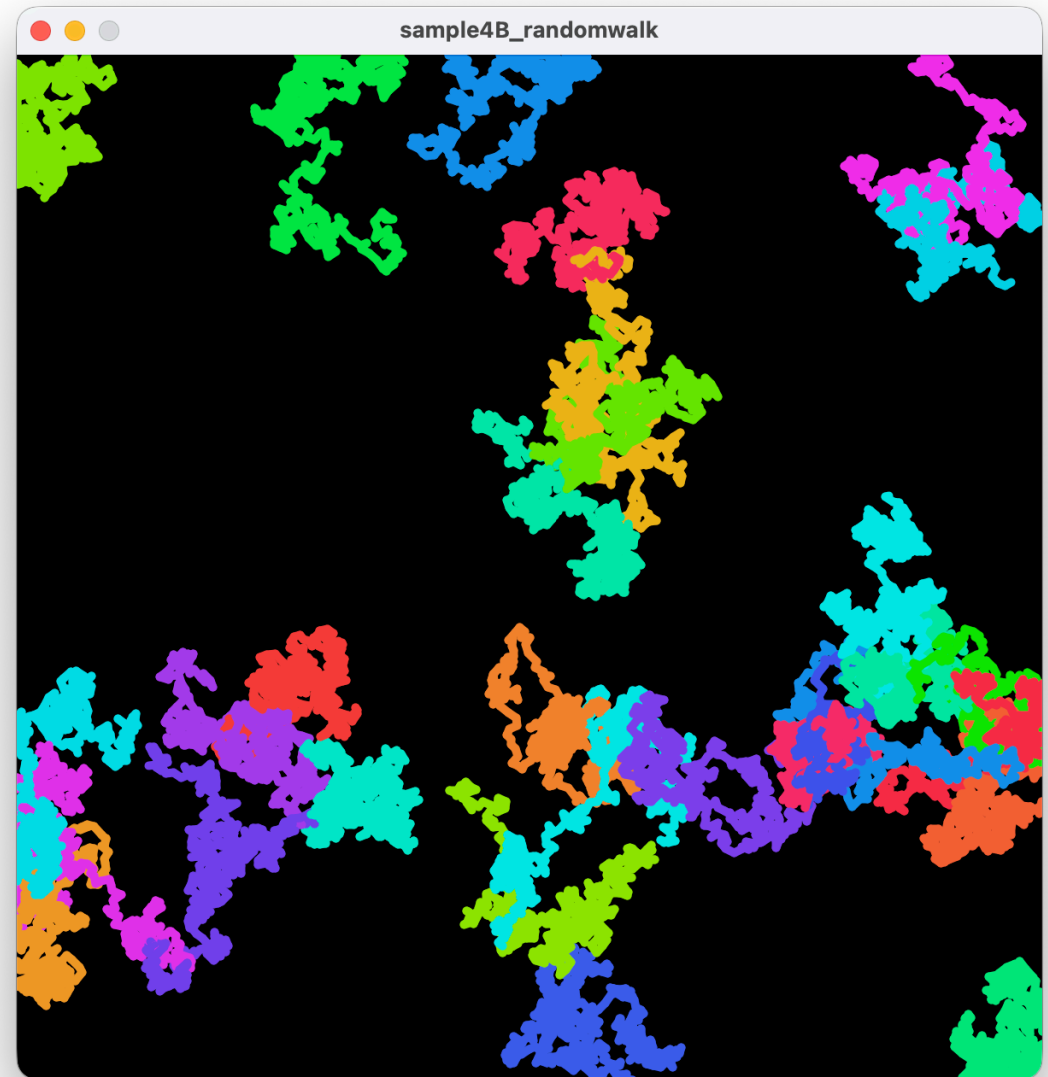
集団ランダムウォーク（サンプル1）

30種類のランダムウォークが並列に進行するコードです。マウスをクリックするたびに、色と場所が初期化され、その後は、それぞれに固有の履歴を背負ってランダムウォーク（最大幅は5ピクセル）を継続します。

sample4B_randomwalk.pde



LINK



sample4B randomwalk

```
1 // ===== 配列を使ったランダムウォーク（酔歩）のサンプル =====
2 // 30個の点をランダムな位置からスタートさせ、毎フレーム少しずつランダムな
3 // 方向へ動かしながら、動いた軌跡を線として描き足していく。
4 // クリックすると、点の位置と色を引き直して最初からやり直す。
5
6 PVector[] posarray; // 各点の現在位置 (x,y) を入れておく配列。PVectorは(x,y)をひとまとめた座標
7 color[] colarray; // 各点に割り当てる色を入れておく配列（点ごとに個性を持たせるために用意）
8 int asum = 30; // 点の数
9
10 void setup(){
11     size(600,600); // ウィンドウのサイズ
12     background(0); // 背景を黒で塗る（drawは毎フレーム再実行されるが、backgroundはここで1回だけ）
13     strokeWeight(5); // 線の太さ
14
15     // 要素数asum個の配列を実際に確保する
16     posarray = new PVector[asum];
17     colarray = new color[asum];
18     initArray(); // それぞれの点の初期位置と色を決める
19 }
20
```

```

21 void draw(){
22
23     float n = 5;    // 1フレームで動ける最大距離（この範囲内でランダムに移動する）
24
25     // 点を1個ずつ、現在位置から少しだけランダムに動かす
26     for(int i=0;i<asum;i++){
27
28         // 現在位置（動く前の位置）を取り出す
29         float x1 = posarray[i].x;
30         float y1 = posarray[i].y;
31
32         // x方向に -n ~ +n の範囲でランダムに移動した新しい位置を計算
33         float x2 = x1 + random(n*2) - n;
34         x2 = constrain(x2,0,width);    // ウィンドウの外に出ないように0~widthの範囲に収める
35
36         // y方向も同様にランダムに移動
37         float y2 = y1 + random(n*2) - n;
38         y2 = constrain(y2,0,height);    // 0~heightの範囲に収める
39
40         stroke(colarray[i]);            // 線の色をi番目の点に割り当てられた色に切り替える
41         line(x1,y1,x2,y2);            // 動く前の位置から動いた後の位置まで線を引く（軌跡として残る）
42         posarray[i] = new PVector(x2,y2);    // 配列の中身を新しい位置に更新する（次のフレームはここから）
43     }
44 }
45

```

draw()


```

46 // 各点の初期位置と色をランダムに決めて配列に格納する
47 void initArray(){
48
49     colorMode(HSB, 360, 100, 100); // 色の指定方法を一時的にHSBに変更。H:0~360, S・B:0~100
50
51
52     for(int i=0;i<asum;i++){
53         int x = int(random(width)); // 0~width未満のランダムなx座標
54         int y = int(random(height)); // 0~height未満のランダムなy座標
55
56         posarray[i] = new PVector(x,y); // i番目の点の初期位置として登録
57
58         color c = color(random(360), 70, 90); // 色相(H)だけランダムに変え、彩度70・明度90は固定 →
59         colarray[i] = c; // i番目の点の色として登録 (draw()内でstroke(colarray[i])として使われる)
60     }
61
62     colorMode(RGB, 255); // 他の場所 (background()など) に影響しないよう、使い終わったらRGBモードに
63
64 }
65
66 // マウスをクリックしたときに呼ばれる (Processingが自動で呼び出す特別な関数)
67 void mousePressed(){
68
69     background(0); // 画面を黒で塗りつぶし、それまでの軌跡を消す
70     initArray(); // 点の位置と色を新しく決め直す (ここからまた酔歩が始まる)
71
72 }

```

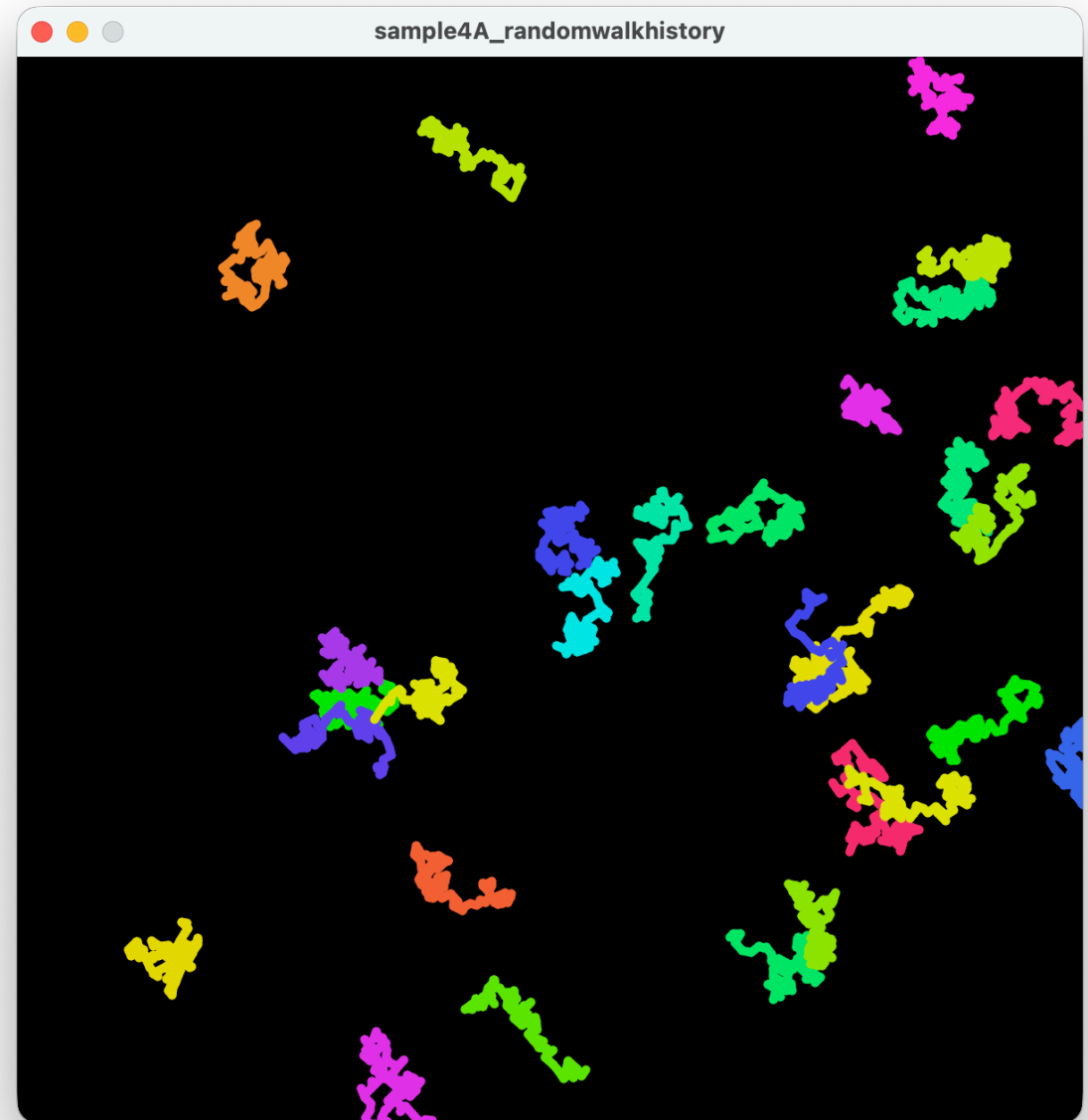
initArray() / mousePressed()

集団ランダムウォーク（サンプル2）

30種類のランダムウォークについて、過去100回分の履歴を記録し、毎回の描画で、前回の描画はリセットされ（background(0)）、最近100回の履歴のみが表示されます。30種類のランダムウォークが100回分の位置を記憶するため、二次元配列が使用されています。



LINK



sample4B_randomwalkhistory

```

1 // ===== 配列を使ったランダムウォーク（酔歩）のサンプル：軌跡の「しっぽ」バージョン =====
2 // 30個の点をランダムな位置からスタートさせ、毎フレーム少しずつランダムな
3 // 方向へ動かしていく点は前回までと同じ。
4 // 違いは、各点について直近history個ぶんの過去位置を配列に保存しておき、
5 // 毎フレーム背景を消してから、その履歴をつないで「しっぽ（トレイル）」として描き直すこと。
6 // 前バージョンのように軌跡が画面に残り続けるのではなく、一定の長さのしっぽが動いていくように見える。
7 // クリックすると、点の位置と色を引き直して最初からやり直す。
8
9 PVector[][] posarray; // [点の番号][何フレーム前か] の2次元配列。posarray[i][0]がi番目の点の最新位置、
10 // posarray[i][1]が1フレーム前、...、posarray[i][history-1]が一番古い位置
11 color[] colarray; // 各点に割り当てる色を入れておく配列（点ごとに個性を持たせるために用意）
12 int asum = 30; // 点の数
13 int history = 100; // 1点あたり何フレームぶんの位置を覚えておくか＝しっぽの長さ
14
15 void setup(){
16     size(600,600); // ウィンドウのサイズ
17     background(0); // 背景を黒で塗る（初回の下地。drawの中でも毎フレーム塗り直す）
18     strokeWeight(5); // 線の太さ
19
20     // posarrayは「点の数 × 履歴の数」ぶんの2次元配列として確保する
21     posarray = new PVector[asum][history];
22     colarray = new color[asum];
23     initArray(); // それぞれの点の初期位置（＝履歴の初期値）と色を決める
24 }
25

```

```

26 void draw(){
27
28     background(0);    // 毎フレーム背景を塗りつぶす。しっぽより後ろの古い軌跡は消え、直近history個ぶんた
29
30     float n = 5;      // 1フレームで動ける最大距離（この範囲内でランダムに移動する）
31
32     // 点を1個ずつ、現在位置から少しだけランダムに動かす
33     for(int i=0;i<asum;i++){
34
35         // 現在位置（=履歴の先頭 posarray[i][0]）を取り出す
36         float x1 = posarray[i][0].x;
37         float y1 = posarray[i][0].y;
38
39         // x方向に -n ~ +n の範囲でランダムに移動した新しい位置を計算
40         float x2 = x1 + random(n*2) - n;
41         x2 = constrain(x2,0,width);    // ウィンドウの外に出ないように0~widthの範囲に収める
42
43         // y方向も同様にランダムに移動
44         float y2 = y1 + random(n*2) - n;
45         y2 = constrain(y2,0,height);  // 0~heightの範囲に収める
46
47         // 履歴を1つずつ後ろにずらす（末尾＝一番古いものは押し出されて消える）
48         // j=history-1から1まで、posarray[i][j] に1つ前の posarray[i][j-1] を入れていく
49         for(int j=history-1;j>0;j--){
50             posarray[i][j] = posarray[i][j-1];
51         }
52
53         posarray[i][0] = new PVector(x2,y2);    // 空いた先頭（0番目）に、今回計算した新しい位置を入れ
54
55
56         // 履歴に保存された過去の位置を、新しい方から順に線でつないでしっぽを描く
57         for(int j=0;j<history-1;j++){
58             stroke(colarray[i]);                // 線の色をi番目の点に割り当てられた色に切り替える
59             line(posarray[i][j].x,posarray[i][j].y,
60                 posarray[i][j+1].x,posarray[i][j+1].y);
61         }
62     }
63 }
64
65 }
66

```

draw()

initArray() / mousePressed()

```
67 // 各点の初期位置と色をランダムに決めて配列に格納する
68 void initArray(){
69
70
71     colorMode(HSB, 360, 100, 100); // 色の指定方法を一時的にHSBに変更。H:0~360, S・B:0~100
72
73
74     for(int i=0;i<asum;i++){
75         int x = int(random(width)); // 0~width未満のランダムなx座標
76         int y = int(random(height)); // 0~height未満のランダムなy座標
77
78         // i番目の点の履歴をすべて同じ初期位置で埋めておく（スタート直後はしっぽが1点に集まった状態になる）
79         for(int j=0;j<history;j++){
80             posarray[i][j] = new PVector(x,y);
81         }
82
83         color c = color(random(360), 70, 90); // 色相(H)だけランダムに変え、彩度70・明度90は固定 →
84         colarray[i] = c; // i番目の点の色として登録（draw()内でstroke(colarray[i])として使われる）
85     }
86
87     colorMode(0, 255); // 他の場所（background()など）に影響しないよう、使い終わったらRGBモードに
88
89 }
90
91
92 // マウスをクリックしたときに呼ばれる（Processingが自動で呼び出す特別な関数）
93 void mousePressed(){
94
95     background(0); // 画面を黒で塗りつぶし、それまでの軌跡を消す
96     initArray(); // 点の位置（履歴）と色を新しく決め直す（ここからまた酔歩が始まる）
97
98 }
```


集団ランダムウォーク（履歴の深さの反映）

```
PVector[] posarray;  
color[] colarray;  
int asum = 50;  
int history = 100;
```

エージェントの数を50に増やす

```
void setup() {  
  size(400, 400);  
}
```

ウィンドウのサイズを小さくします

```
void draw() {  
  background(0);
```

```
float n = 8;
```

移動最大幅を8とする。

```
for (int i=0; i<asum; i++) {  
  float x1 = posarray[i][0].x;  
  float y1 = posarray[i][0].y;  
  float x2 = x1 + random(n*2) - n;  
  x2 = constrain(x2, 0, width);  
  float y2 = y1 + random(n*2) - n;  
  y2 = constrain(y2, 0, height);
```

```
for (int j=history-1; j>0; j--) {  
  posarray[i][j] = posarray[i][j-1];  
}
```

```
posarray[i][0] = new PVector(x2, y2);
```

```
for (int j=0; j<history-1; j++) {
```

```
  stroke(colarray[i], 30);  
  strokeWeight(map(j, 0, history-1, 40, 1));
```

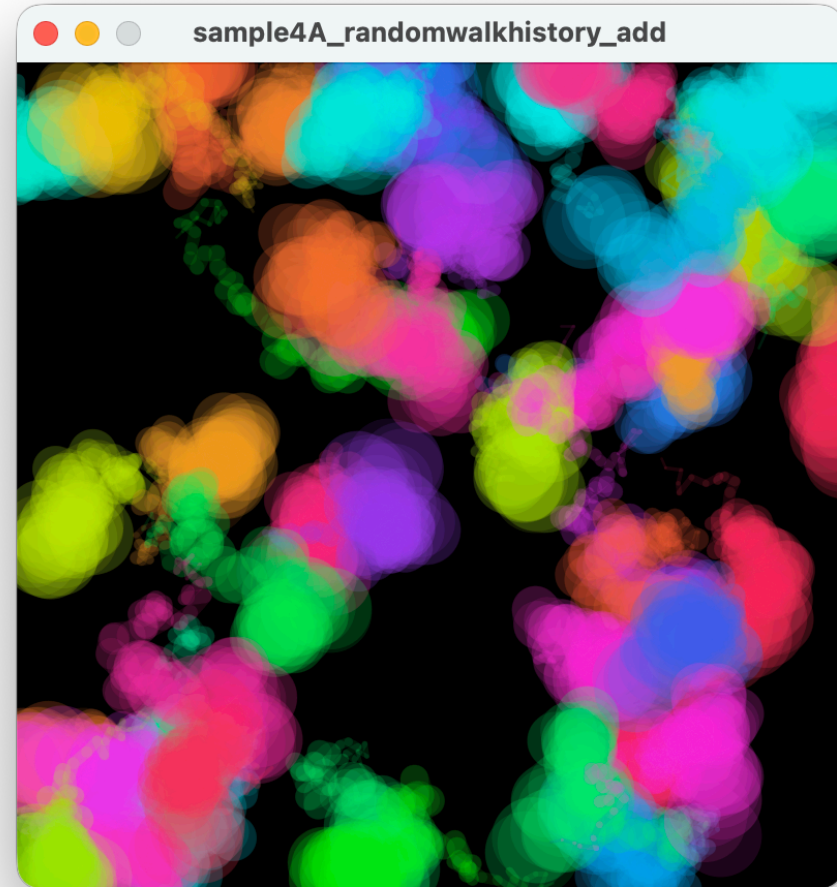
```
  line(posarray[i][j].x, posarray[i][j].y,  
        posarray[i][j+1].x, posarray[i][j+1].y);
```

```
}
```

```
}
```

draw()

色を半透明として、履歴の深さに応じて、線幅にコンストラストをつけます。



色を半透明とし、履歴の深さ（0～99）に応じて、線幅を1から40の間にマッピングする。

```
float map(val, a1, a2, b1, b2);
```

値 val が a1～a2 のどこにあるかを求め、それを b1～b2 の同じ割合の位置に置きかえて返す（比例変換）。

集団ランダムウォーク（色の陣地取り）

```
for (int i=0; i<asum; i++) {  
    PVector ipos = posarray[i][0];
```

draw()の末尾に追加

```
if(mode_line){  
    noFill(); stroke(255, 150); strokeWeight(1);  
    ellipse(ipos.x, ipos.y, 25, 25);  
}
```

最新の位置に白
い枠で円を描く。

```
for (int j=i+1; j<asum-1; j++) {  
    PVector jpos = posarray[j][0];
```

```
if(mode_line){  
    if (PVector.dist(ipos, jpos)<50) {  
        noFill(); stroke(255); strokeWeight(1);  
        line(ipos.x, ipos.y, jpos.x, jpos.y);  
    }  
}
```

相互の距離が50ピ
クセル以内の時
に、白い線を引く。

```
if (PVector.dist(ipos, jpos)<20) {
```

```
    if (random(1)<0.5) {  
        colarray[i] = colarray[j];  
    } else {  
        colarray[j] = colarray[i];  
    }  
}
```

相互の距離が20ピクセル
以内の時に、一方の色が
もう一方に吸収される。

```
PVector pos2 = PVector.add(ipos, jpos); // 加算  
pos2.mult(0.5); // 乗算
```

添字iと添字jの位
置の重心をpos2
として計算する。

```
if(mode_line){  
    noFill(); stroke(255); strokeWeight(1);  
    ellipse(pos2.x, pos2.y, 50, 50);  
    ellipse(pos2.x, pos2.y, 55, 55);  
}
```

iとjの両者を覆う円の白枠を描画する。

```
}  
}
```

宣言部

```
PVector[][] posarray;  
color[] colarray;  
int asum = 50;  
int history = 100;
```

```
boolean mode_line = true;
```

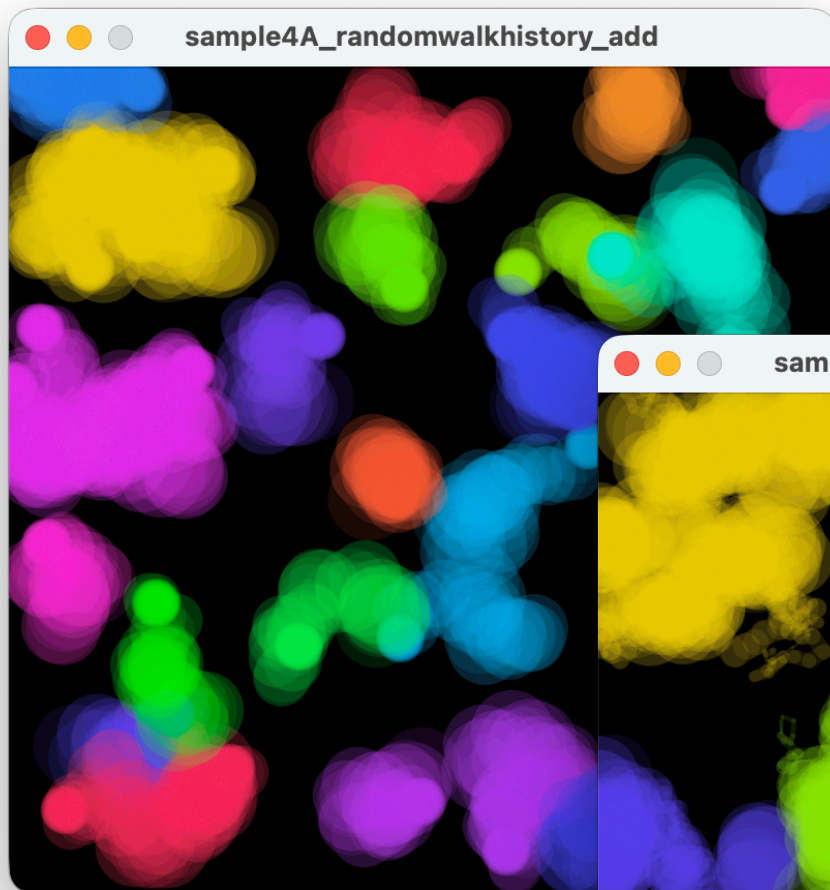
白線の描画の
ON/OFF

keyPressed()

```
void keyPressed(){
```

```
    if(key=='l'){  
        mode_line = !mode_line;  
    }  
}
```

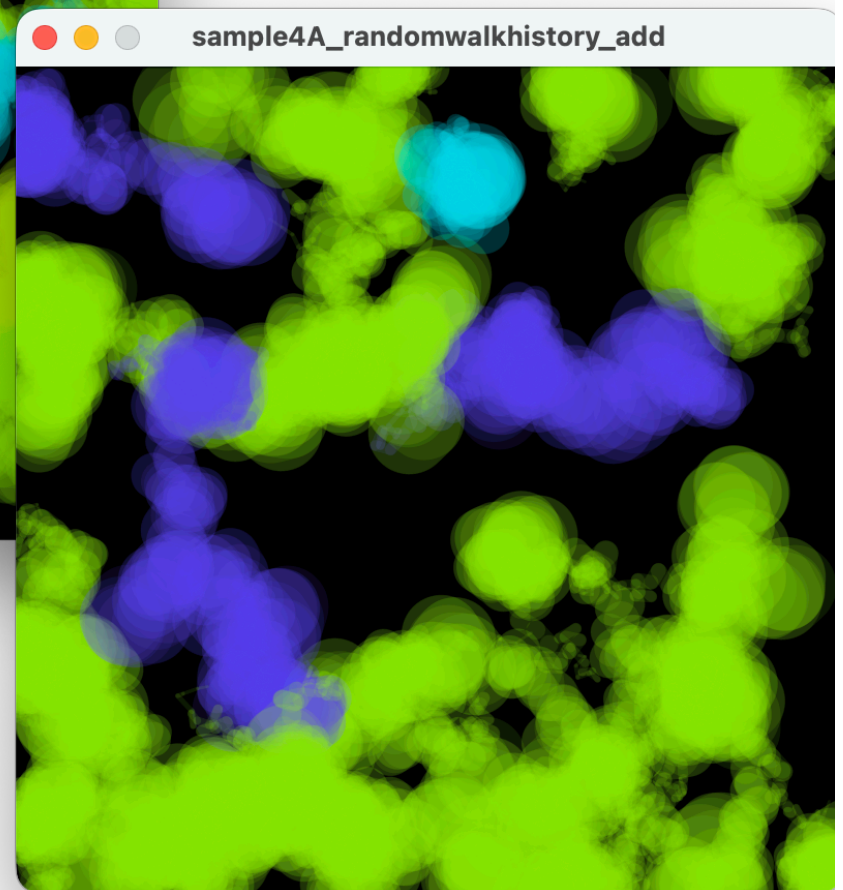
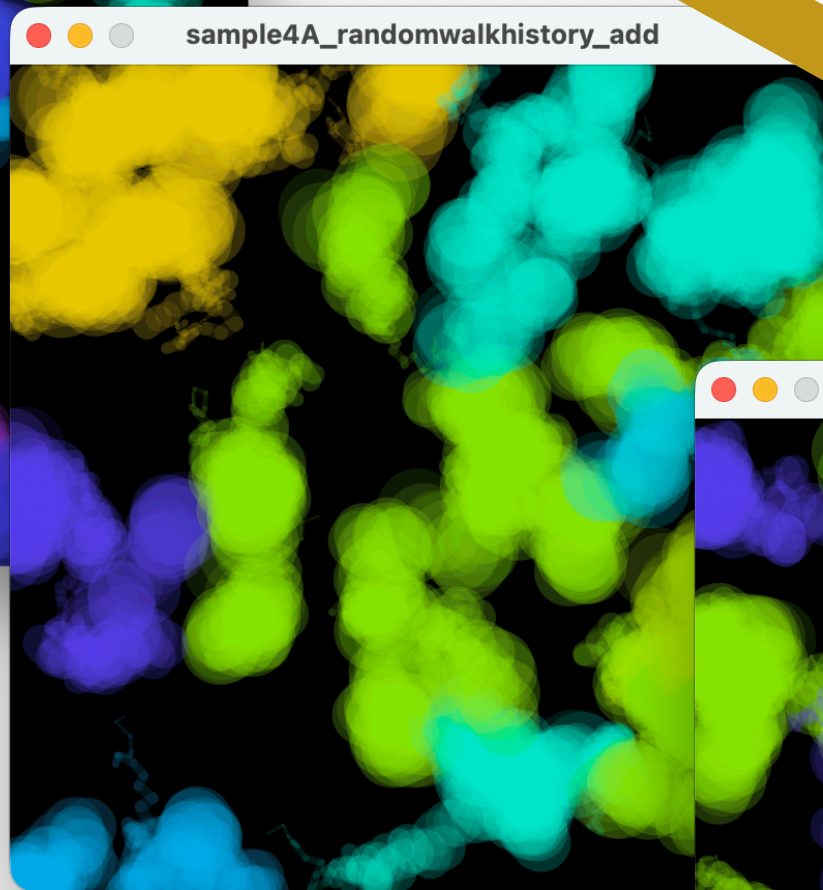
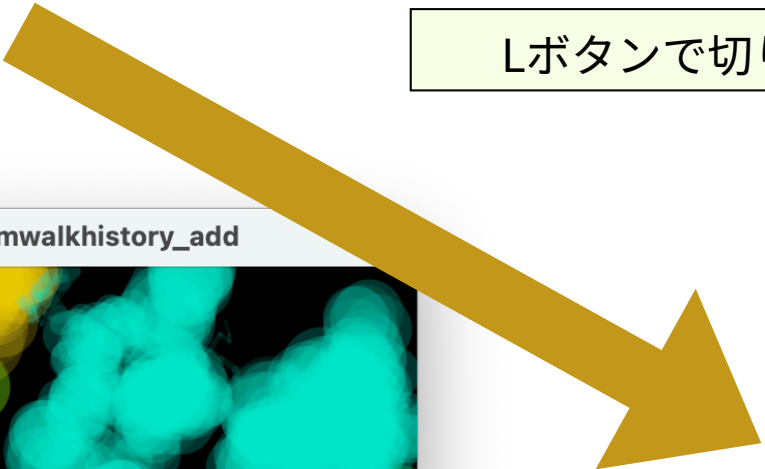
Lボタンで切り替え



実行結果 (例)

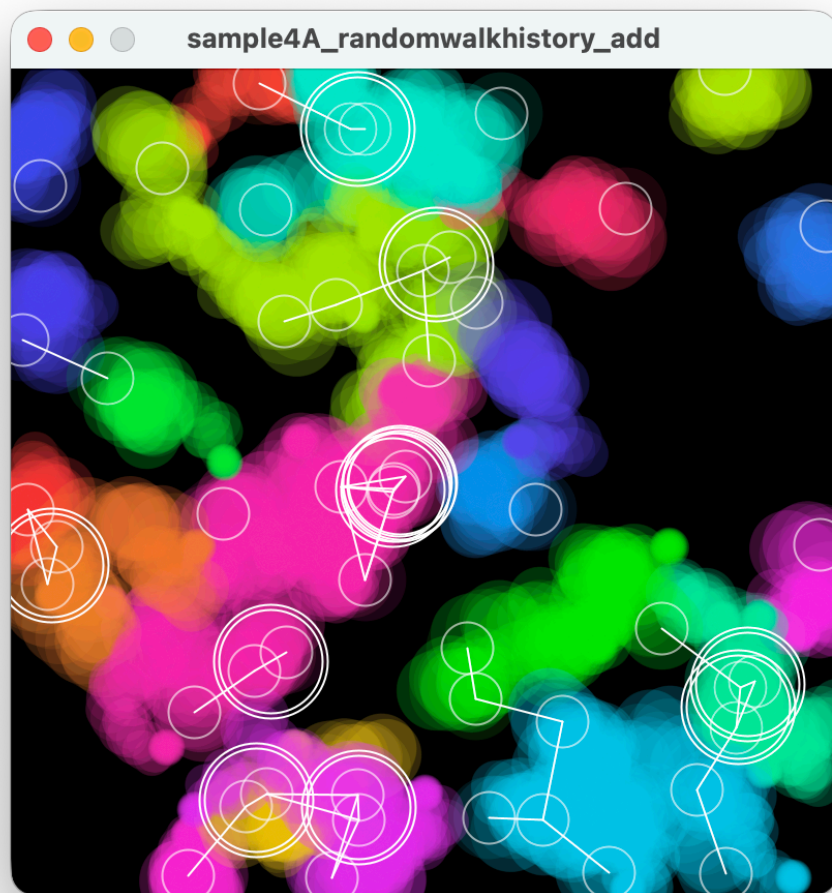
mode_line = false

Lボタンで切り替え

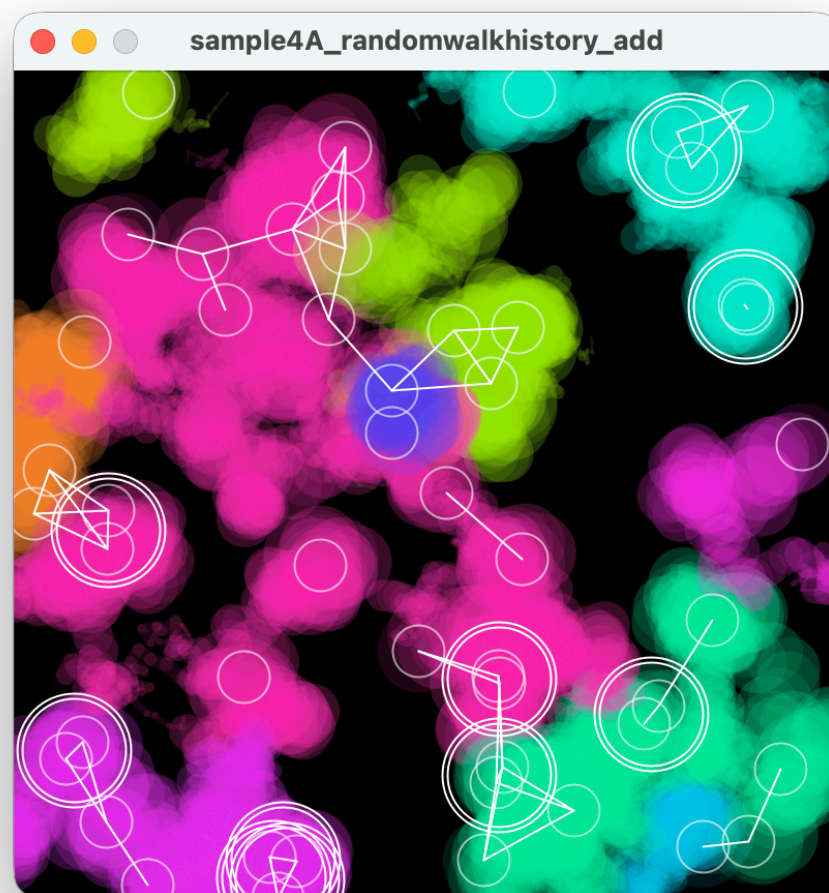


mode_line = true

Lボタンで切り替え



実行結果 (例)



確認課題 4B

授業で説明したプログラム（色の陣地取り）を作成して、色が収束していくことがわかる2枚のスクリーンショットを撮影して提出してください。ウィンドウサイズやエージェントの数（`asum`）は各人の好みに委ねます。あまりウィンドウが大きいと、なかなか収束していかないので注意してください。締め切りは7月31日（金）。



提出画像 1



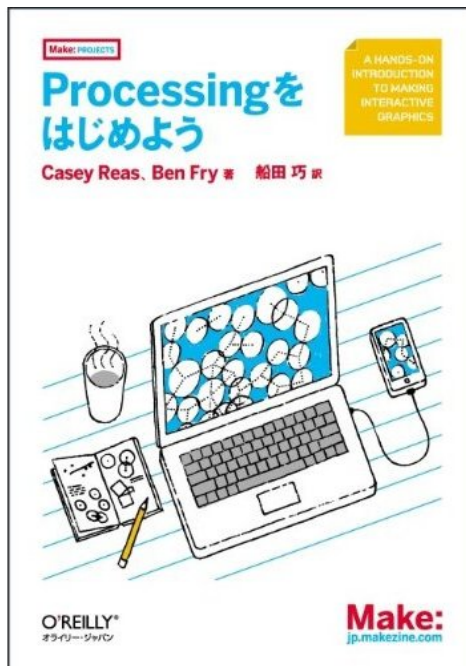
提出画像 2

Practice #4

多次元のデータを扱う

演習4B PVector (位置)

演習4C 画像



Processingをはじめよう
第二版
(Make: PROJECTS)

オンデマンド授業 7.24

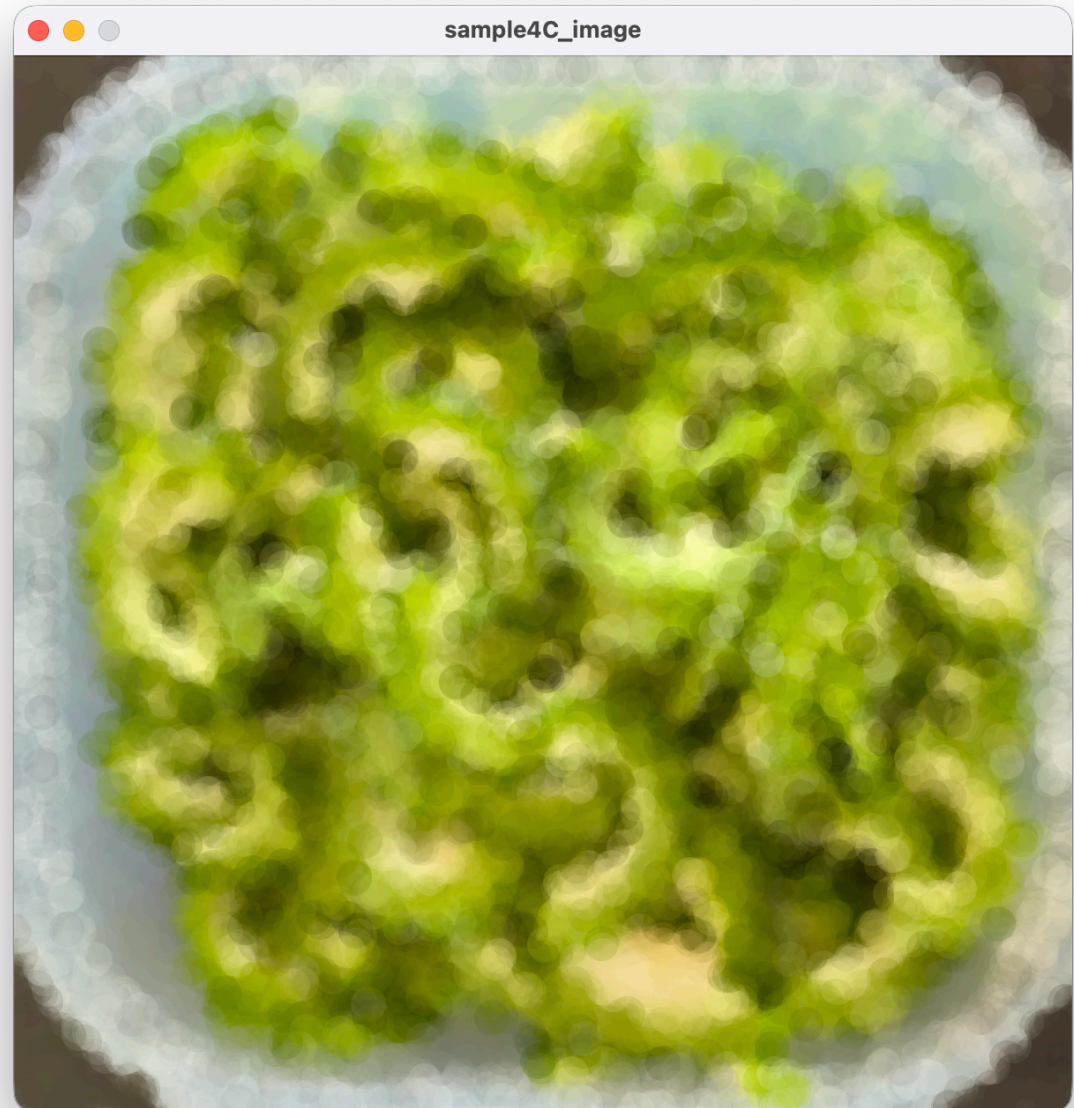
画像の色で点描を描く

元画像から画素の色をランダムに取り出し、その色の半透明な円を毎フレーム100個ずつ描き足していくコードです。円が重なり合うほど、色が混ざり合って少しずつ元の画像に近づいていきます。クリックすると元画像を確認でき、スペースキーで画面をクリアして最初からやり直せます。

sample4C_image.pde



LINK



sample4C image ▼

```
1 // ===== 画像の色を使って点描（ポインティリズム）風に再構成するサンプル =====
2 // 元画像（goya.jpg）を読み込み、ランダムな位置の画素の色を取り出して、
3 // その色の半透明な円を毎フレーム100個ずつ描き足していく。
4 // 円が積み重なるにつれて、少しずつ元の画像に近い絵が浮かび上がってくる。
5 // クリックすると元画像をそのまま表示、スペースキーで画面をクリアする。
6
7 PImage img;    // 読み込んだ画像を保持する変数（PImageは画像を扱うためのクラス）
8
9 void setup(){
10     pixelDensity(2);        // Retinaディスプレイなどで画像がぼやけないよう、内部解像度を2倍にする
11     size(600,600);          // ウィンドウのサイズ
12
13     // 画像はスケッチフォルダの中の「data」フォルダに置いておくと、
14     // loadImage("ファイル名")だけで読み込める（Processingの決まり）
15     img = loadImage("goya.jpg");
16     img.resize(width,height); // 画像をウィンドウと同じサイズに拡大・縮小する
17     img.loadPixels();         // 画素の色を1つずつ取り出せるように、画素データをメモリ上に展開する
18 }
19
```

```

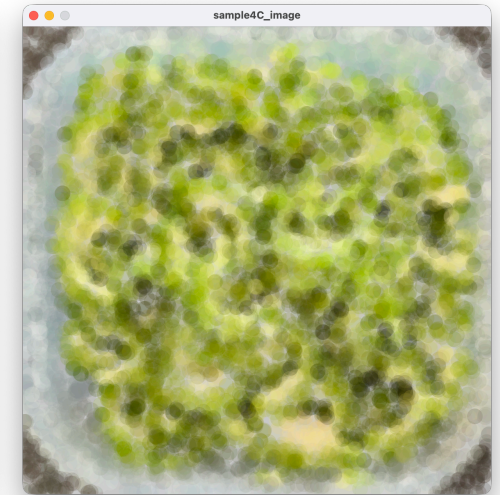
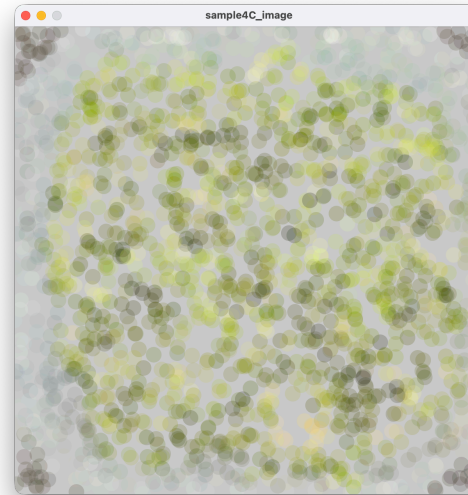
20 void draw(){
21
22     // 1フレームにつき100個、ランダムな位置に円を描き足す
23     for(int i=0;i<100;i++){
24
25         int x = int(random(width));    // 0～width未満のランダムなx座標
26         int y = int(random(height));    // 0～height未満のランダムなy座標
27
28         // 2次元の座標(x,y)を、1次元に並んだpixels配列の添字に変換する
29         // 画素は左上から右方向に並び、1行分 (img.width個) 進むごとに次の行へ移る、という並び方をしている
30         int index = x + y * img.width;
31         color c = img.pixels[index];    // その位置の画素の色を取り出す
32
33         float trans = 50;    // 円の透明度 (0～255)。低いほど透明＝重なるほど元の色に近づく
34         float esize = 20;    // 円の直径
35
36
37         fill(c,trans); noStroke();    // 取り出した色・半透明で塗る。ふち線はなし
38         ellipse(x,y,esize,esize);    // その位置に円を描く
39     }
40 }
41
42 }
43
44 // マウスをクリックすると、点描を消して元画像をそのまま表示する (答え合わせ用)
45 void mousePressed(){
46     image(img,0,0);
47 }
48
49 // スペースキーを押すと画面を黒でクリアし、点描を最初からやり直せるようにする
50 void keyPressed(){
51
52     if(key == ' '){
53         background(0);
54     }
55 }

```

draw() / mousePressed() / keyPressed()

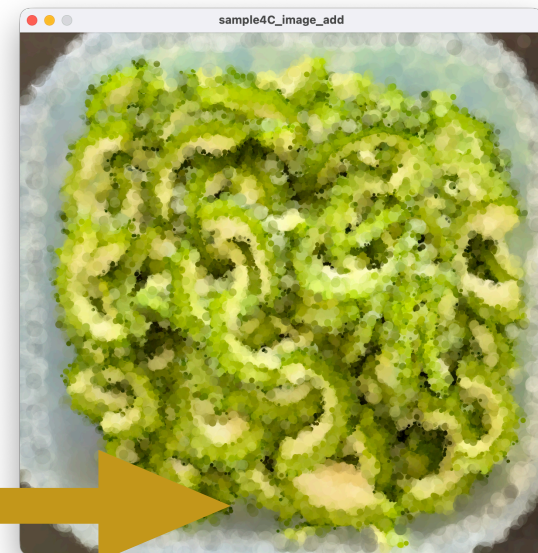
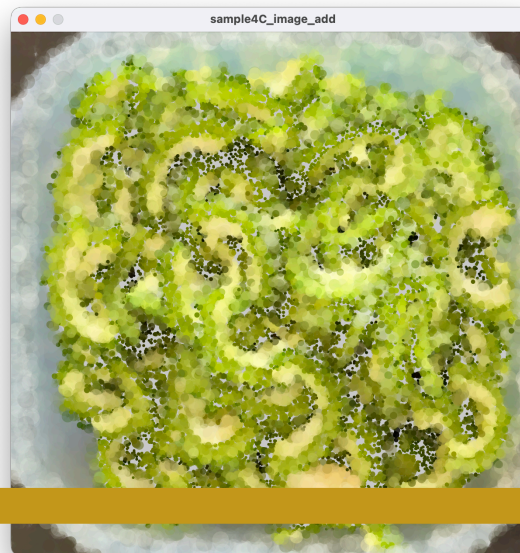
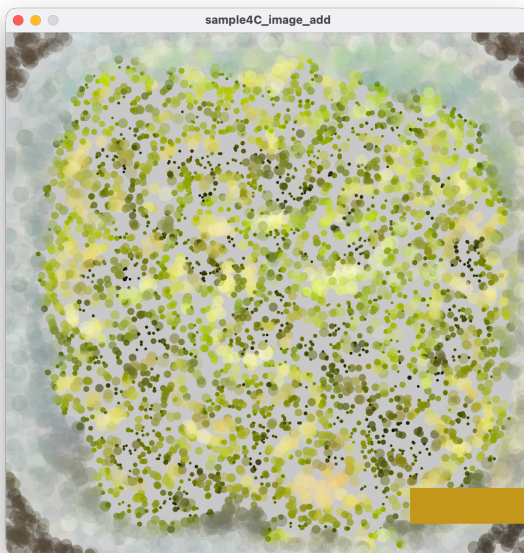
彩度に応じて、円の透明度・大きさを変更する.

```
32  
33 float trans = 50;    // 円の透明度 (0~255)。  
34 float esize = 20;    // 円の直径  
35
```



透明度と円の直径を, 彩度に応じてマッピングします. 具体的には 彩度が高いほど不透明に, 円の直径は小さくするように設定します.

```
32  
33 float sat = saturation(c);    // その色の彩度 (0~255)  
34 float trans = map(sat,0,255,50,255); // 0~255の範囲を50~255に線形変換→彩度が高いほど不透明に  
35 float esize = map(sat,0,255,20,3);   // 0~255の範囲を20~3に線形変換 (逆比例) →彩度が高いほど円は小さく  
36
```



確認課題 4C

自分の好みの画像を使った、点描アニメーションのスクリーンショットを2枚（初期のものと収束後のもの）添付してください。点描の形式を自由に変えてもらってもOK。締め切りは7月31日（金）。

