

IDD Media lab. 2015

力と運動：動きを極める！

多摩美術大学情報デザイン学科メディア芸術コース

2015年5月12日

田所淳

今日の内容

- ▶ 動きを極める!!
- ▶ 物理法則を援用しながら、いろいろな動きを実現していく
- ▶ コードサンプルは、Githubのリポジトリから
- ▶ https://github.com/tado/idd_medilab15

運動を記述する

- ▶ 運動を、座標への足し引きではなく、本質的な原理から生みだす!!
- ▶ まずは、単純な直線運動を、より本質的に理解する
- ▶ ニュートン力学を、プログラムで表現



運動を記述する

- ▶ ニュートンの運動の法則
 - ▶ 第1法則（慣性の法則）
 - ▶ 質点は、力が作用しない限り、静止または等速直線運動する
 - ▶ 第2法則（ニュートンの運動方程式）
 - ▶ 質点の加速度は、そのとき質点に作用する力に比例し、質点の質量に反比例する
 - ▶ 第3法則（作用・反作用の法則）
 - ▶ 二つの質点の間に相互に力が働くとき、質点2から質点1に作用する力と、質点1から質点2に作用する力は、大きさが等しく、逆向きである

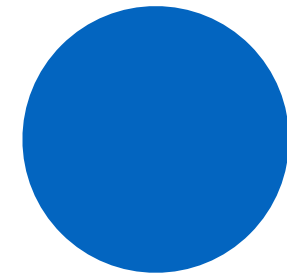
運動1：速度と加速度

運動1：速度と加速度

- ▶ 用語の整理
- ▶ velocity（速度）：単位時間当たりの物体の位置の変化
- ▶ acceleration（加速度）：単位時間当たりの速度の変化率
- ▶ mass（質量）：物体の動かしにくさの度合い（重さとは別）
- ▶ friction（摩擦力）：その物体の進行方向と逆向きに働く力

運動1：速度と加速度

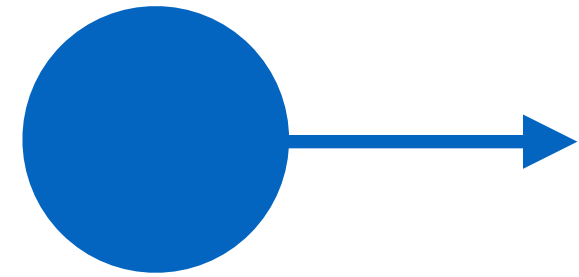
- ▶ 直線運動を、速度と加速度で整理
- ▶ さしあたって、全く摩擦（friction）のない、無重力空間を想定
- ▶ 物体は質量（mass）をもっている
- ▶ 静止している状態では、加速度 0



acceleration = 0

運動1：速度と加速度

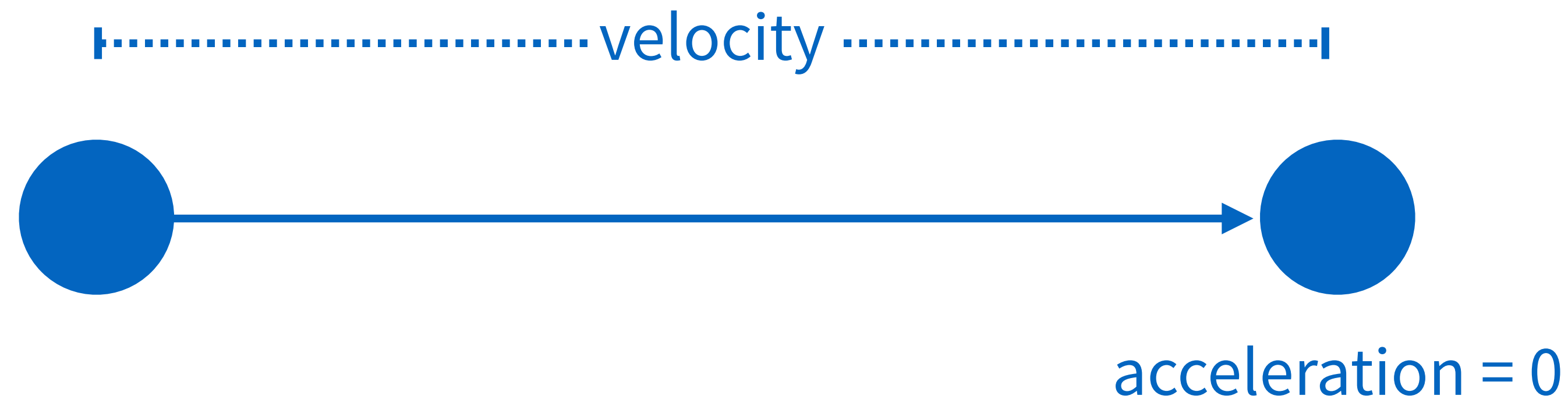
- ▶ 加速度を0より大きくした瞬間、物体が動き始める



acceleration > 0

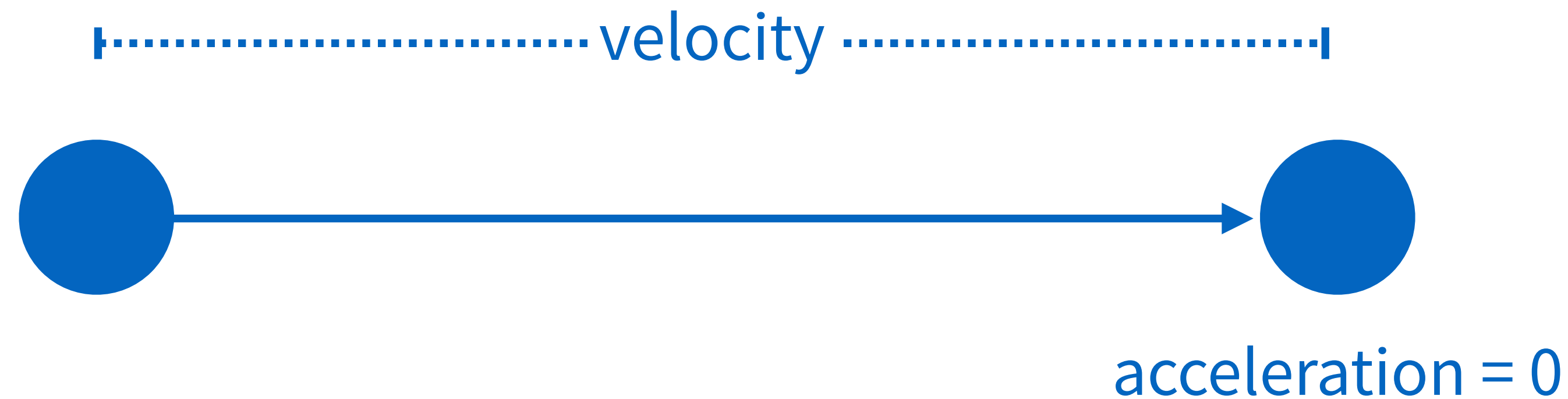
運動1：速度と加速度

- ▶ 動き出した物体は、そのまま永久に動き続ける
- ▶ 速度の変化が無い場合は、動いている間の加速度は0
- ▶ 一定時間での位置の変化が、速度になる



運動1：速度と加速度

- ▶ 「すべての物体は、外部から力を加えられない限り、静止している物体は静止状態を続け、運動している物体は等速直線運動を続ける」
- ▶ ニュートンの運動の第1法則



運動1：速度と加速度

- ▶ ここまでの内容を、コードで表現してみよう!
- ▶ 独立したクラスとして実装：ParticleVec2

運動1：速度と加速度

▶ ParticleVec2.h

```
#pragma once
#include "ofMain.h"

class ParticleVec2 {
public:
    ParticleVec2(); //コンストラクタ
    void update();
    void draw();

    ofVec2f position; //位置
    ofVec2f velocity; // 速度
    ofVec2f acceleration; // 加速度
    float radius; // 表示する円の半径
};
```

運動1：速度と加速度

▶ ParticleVec2.cpp

```
#include "ParticleVec2.h"

ParticleVec2::ParticleVec2(){
    // 初期設定
    radius = 5.0;
    position = ofVec2f(ofGetWidth()/2.0, ofGetHeight()/2.0);
    velocity = ofVec2f(0, 0);
}

void ParticleVec2::update(){
    velocity += acceleration; // 加速度から速度を算出
    position += velocity; // 速度から位置を変更
    acceleration.set(0, 0); // 加速度をリセット
}

void ParticleVec2::draw(){
    ofCircle(position.x, position.y, radius); // 円を描画
}
```

運動1：速度と加速度

▶ ofApp.h

```
#pragma once

#include "ofMain.h"
#include "ParticleVec2.h"

class ofApp : public ofBaseApp{
public:
    void setup();
    void update();
    void draw();

    void keyPressed(int key);
    void keyReleased(int key);
    void mouseMoved(int x, int y );
    void mouseDragged(int x, int y, int button);
    void mousePressed(int x, int y, int button);
    void mouseReleased(int x, int y, int button);
    void windowResized(int w, int h);
    void dragEvent(ofDragInfo dragInfo);
    void gotMessage(ofMessage msg);

    ParticleVec2 particle; // インスタンス化
};
```

運動1：速度と加速度

▶ ofApp.cpp

```
#include "ofApp.h"

void ofApp::setup(){
    ofSetFrameRate(60);
    ofBackground(0);
}

void ofApp::update(){
    particle.update(); // 等速運動の更新
}

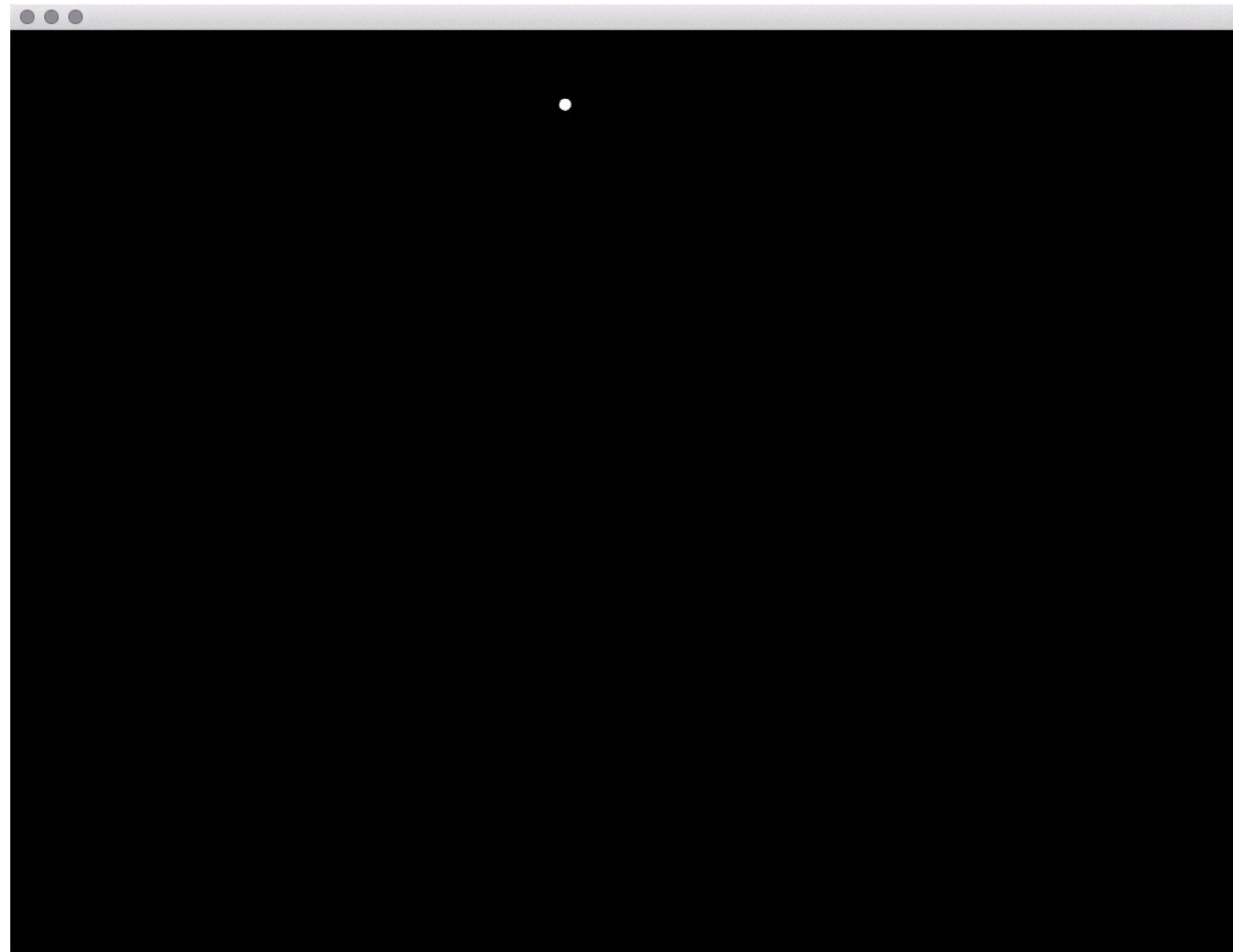
void ofApp::draw(){
    particle.draw(); // 描画
}

(中略)

// クリックした場所から、等速度運動を開始
void ofApp::mouseReleased(int x, int y, int button){
    particle.position = ofVec2f(x, y); // 初期位置
    particle.velocity = ofVec2f(0, 0); // 速度をリセット
    particle.acceleration = ofVec2f(ofRandom(-10, 10), ofRandom(-10, 10)); // 加速度を設定
}
```

運動1：速度と加速度

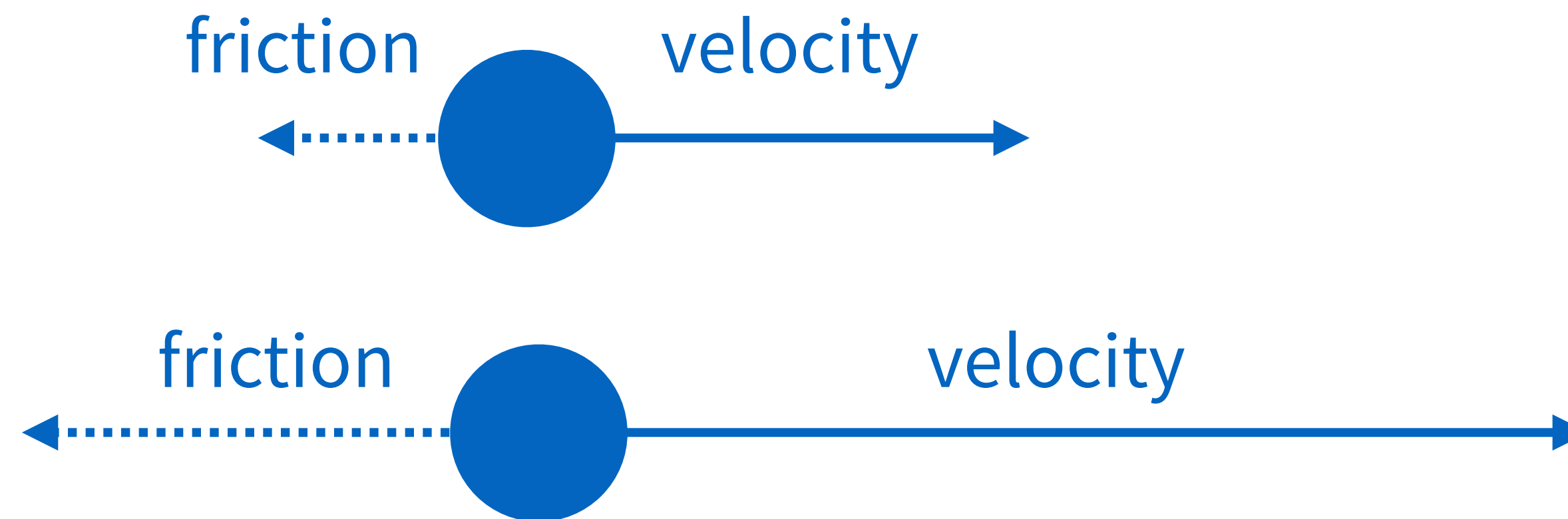
- ▶ マウスをクリックした場所から、等速運動する



運動2：摩擦

運動2：摩擦

- ▶ まったく抵抗のない空間はほとんどない
- ▶ 摩擦力（friction）を実装したい
- ▶ friction（摩擦力）：その物体の速度と逆向きに働く力
- ▶ 摩擦力は速度に比例する



運動2：摩擦力

▶ ParticleVec2.h

```
#pragma once
#include "ofMain.h"

class ParticleVec2 {
public:
    ParticleVec2();
    void setup(ofVec2f position, ofVec2f velocity);
    void setup(float positionX, float positionY, float velocityX, float velocityY);
    void update();
    void draw();

    ofVec2f position;
    ofVec2f velocity;
    ofVec2f acceleration;
    float radius;
    float friction; // 摩擦力
};
```

運動2：摩擦力

▶ ParticleVec2.cpp

```
#include "ParticleVec2.h"

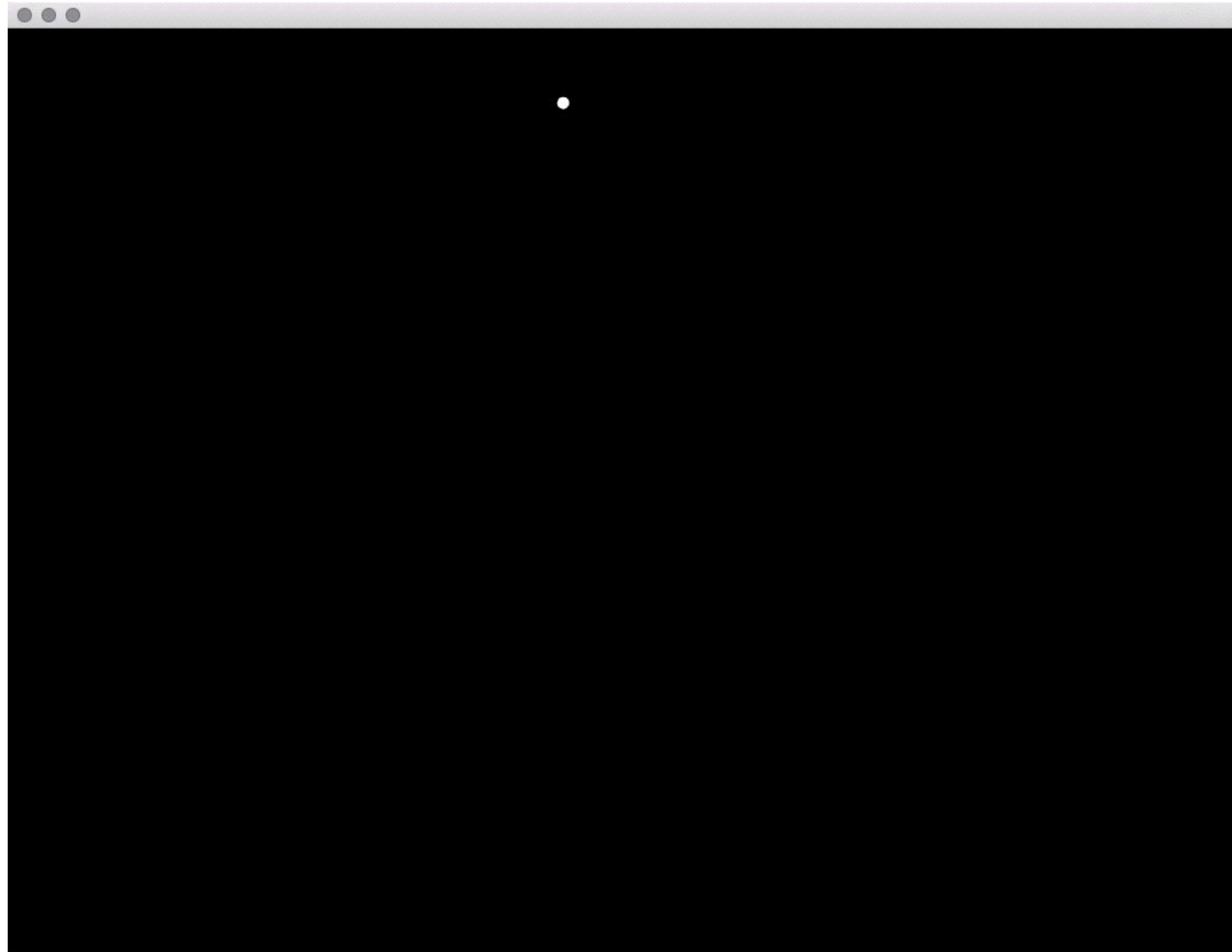
ParticleVec2::ParticleVec2(){
    friction = 0.01; // 摩擦力の初期設定
    radius = 5.0;
    position = ofVec2f(ofGetWidth()/2.0, ofGetHeight()/2.0);
    velocity = ofVec2f(0, 0);
}

void ParticleVec2::update(){
    // 速度に比例して摩擦力を算出して加速度を変更
    acceleration -= velocity * friction;
    velocity += acceleration;
    position += velocity;
    acceleration.set(0, 0);
}

void ParticleVec2::draw(){
    ofCircle(position.x, position.y, radius);
}
```

運動2：摩擦

- ▶ 徐々に減速する動きに!



運動 3 : 質量と力、重力

運動 3 : 質量と力、重力

- ▶ 力（force）を実装してみる！
- ▶ 力は、質量 x 加速度 → ニュートンの運動の第2法則
- ▶ つまり、力を加えるということは、物体に加速度がかかるということ

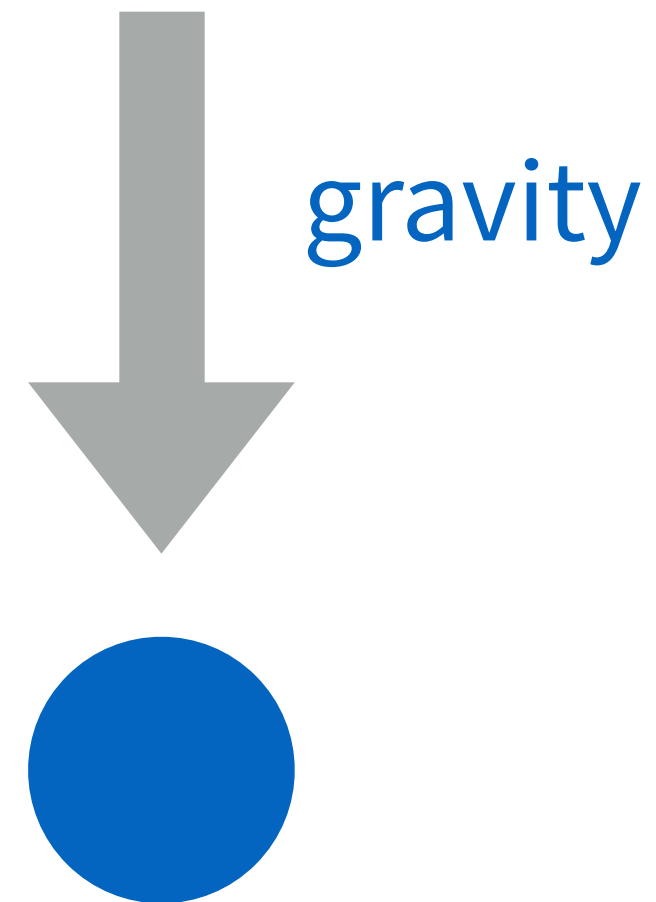
$$F = ma$$

$$a = F / m$$

a: acceleration, m: mass, F: force

運動 3 : 質量と力、重力

- ▶ 重力（gravity）とは？
- ▶ つねに地面の方向にかかりつづける力
- ▶ 重力を実装してみる！



運動 3 : 質量と力、重力

▶ ParticleVec2.h

```
#pragma once
#include "ofMain.h"

class ParticleVec2 {
public:
    ParticleVec2();
    void update();
    void draw();

    // 物体に力を加える (ベクトル版とX,Y版)
    void addForce(ofVec2f force);
    void addForce(float forceX, float forceY);

    ofVec2f position;
    ofVec2f velocity;
    ofVec2f acceleration;
    float radius;
    float friction;
    float mass; // 質量
};
```

運動 3 : 質量と力、重力

▶ ParticleVec2.h

```
#include "ParticleVec2.h"

ParticleVec2::ParticleVec2(){
    radius = 5.0;
    friction = 0.01;
    mass = 1.0;
    position = ofVec2f(ofGetWidth()/2.0, ofGetHeight()/2.0);
    velocity = ofVec2f(0, 0);
}

void ParticleVec2::update(){
    acceleration -= velocity * friction;
    velocity += acceleration;
    position += velocity;
    acceleration.set(0, 0);
}

void ParticleVec2::draw(){
    ofCircle(position.x, position.y, radius);
}

void ParticleVec2::addForce(ofVec2f force){
    acceleration += force / mass; // a = F/m
}

void ParticleVec2::addForce(float forceX, float forceY){
    acceleration += ofVec2f(forceX, forceY) / mass; // a = F/m
}
```

運動 3 : 質量と力、重力

▶ ofApp.cpp

```
#include "ofApp.h"

void ofApp::setup(){
    ofSetFrameRate(60);
    ofBackground(0);
}

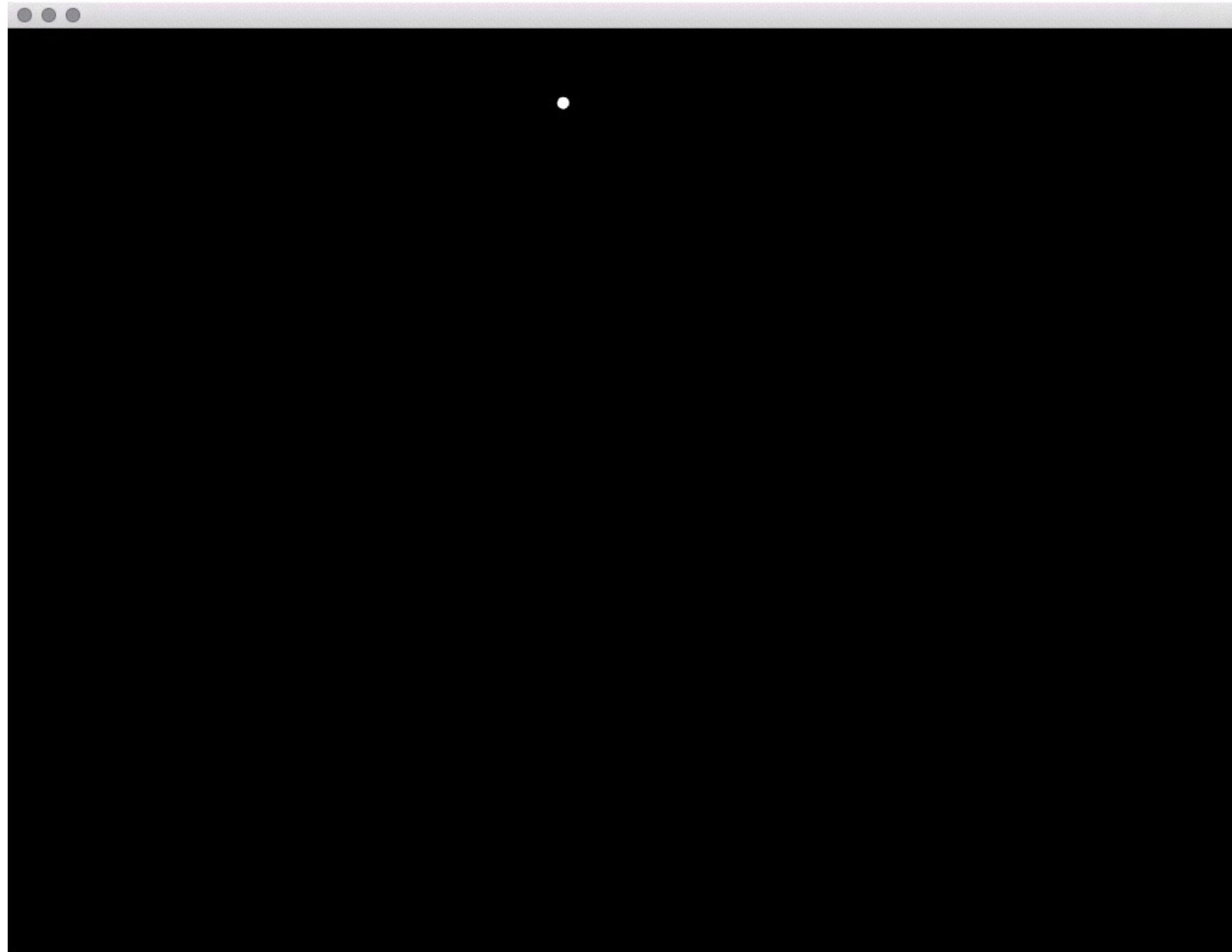
void ofApp::update(){
    particle.addForce(0, 1.0); // 下向きの力を加える(重力)
    particle.update();
}

void ofApp::draw(){
    particle.draw();
}

(後略)
```

運動 3 : 質量と力、重力

- ▶ 重力が実現!

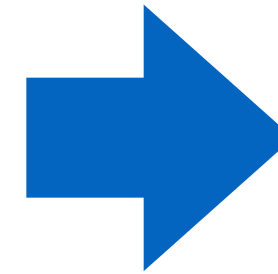


大量の物体を同時に運動させる

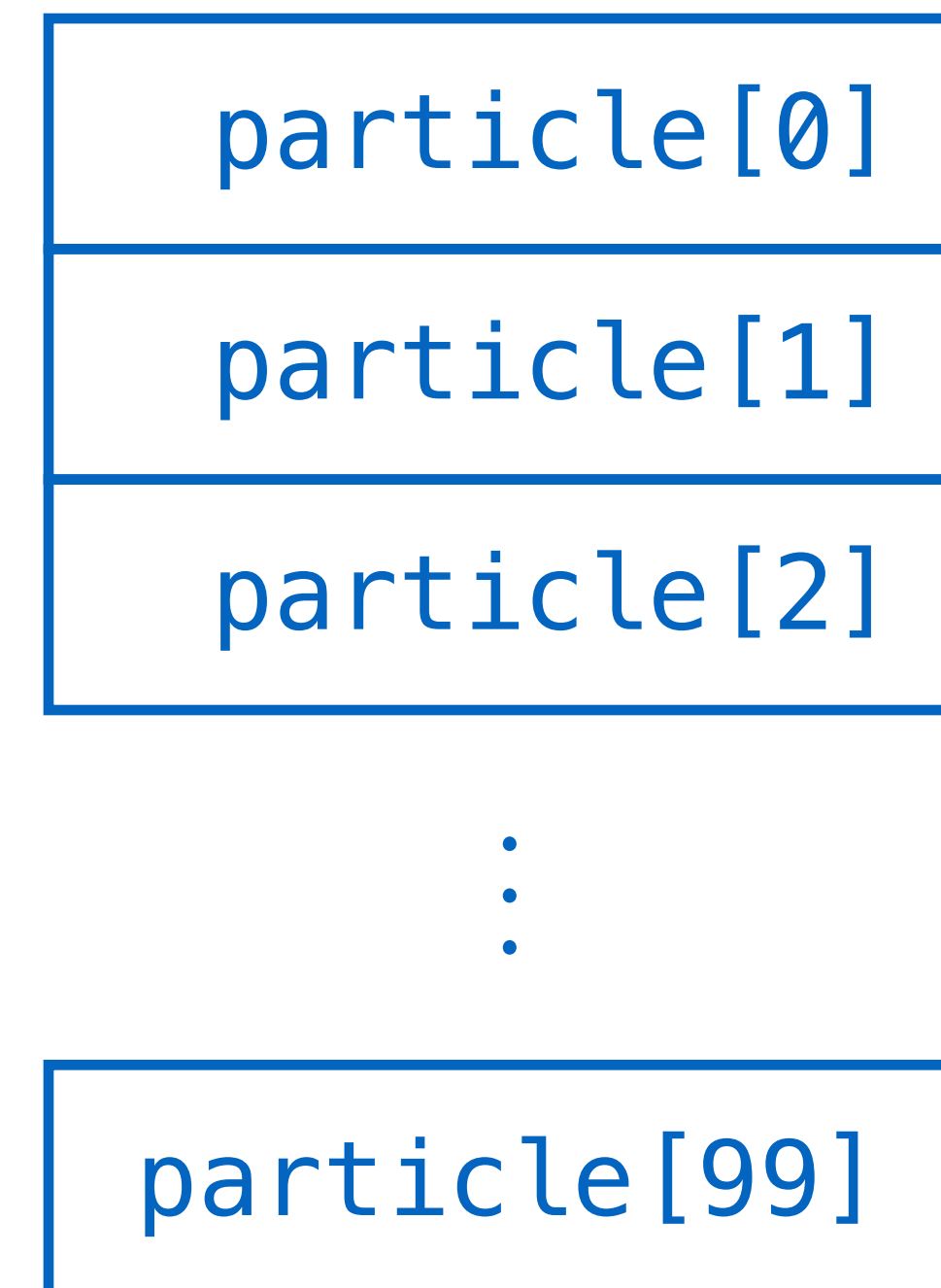
大量の物体を同時に運動させる

- ▶ いよいよ、1粒ではなく大量の物体を動かしてみる!
- ▶ 配列 (Array) をつかって、ParticleVec2を大量生産
- ▶ ParticleVec2クラスはそのまま

ParticleVec2 particle



ParticleVec2 particle[100]



大量の物体を同時に運動させる

▶ testApp.h

```
#pragma once

#include "ofMain.h"
#include "ParticleVec2.h"

class ofApp : public ofBaseApp{
public:
    void setup();
    void update();
    void draw();

    void keyPressed(int key);
    void keyReleased(int key);
    void mouseMoved(int x, int y );
    void mouseDragged(int x, int y, int button);
    void mousePressed(int x, int y, int button);
    void mouseReleased(int x, int y, int button);
    void windowResized(int w, int h);
    void dragEvent(ofDragInfo dragInfo);
    void gotMessage(ofMessage msg);

    static const int num = 1000; // 物体の数
    ParticleVec2 particles[num]; // 配列を生成
};
```

大量の物体を同時に運動させる

▶ testApp.cpp

```
#include "ofApp.h"

void ofApp::setup(){
    ofSetFrameRate(60);
    ofBackground(0);
}

void ofApp::update(){
    for (int i = 0; i < num; i++) {
        particles[i].addForce(0, 1.0);
        particles[i].update();
    }
}

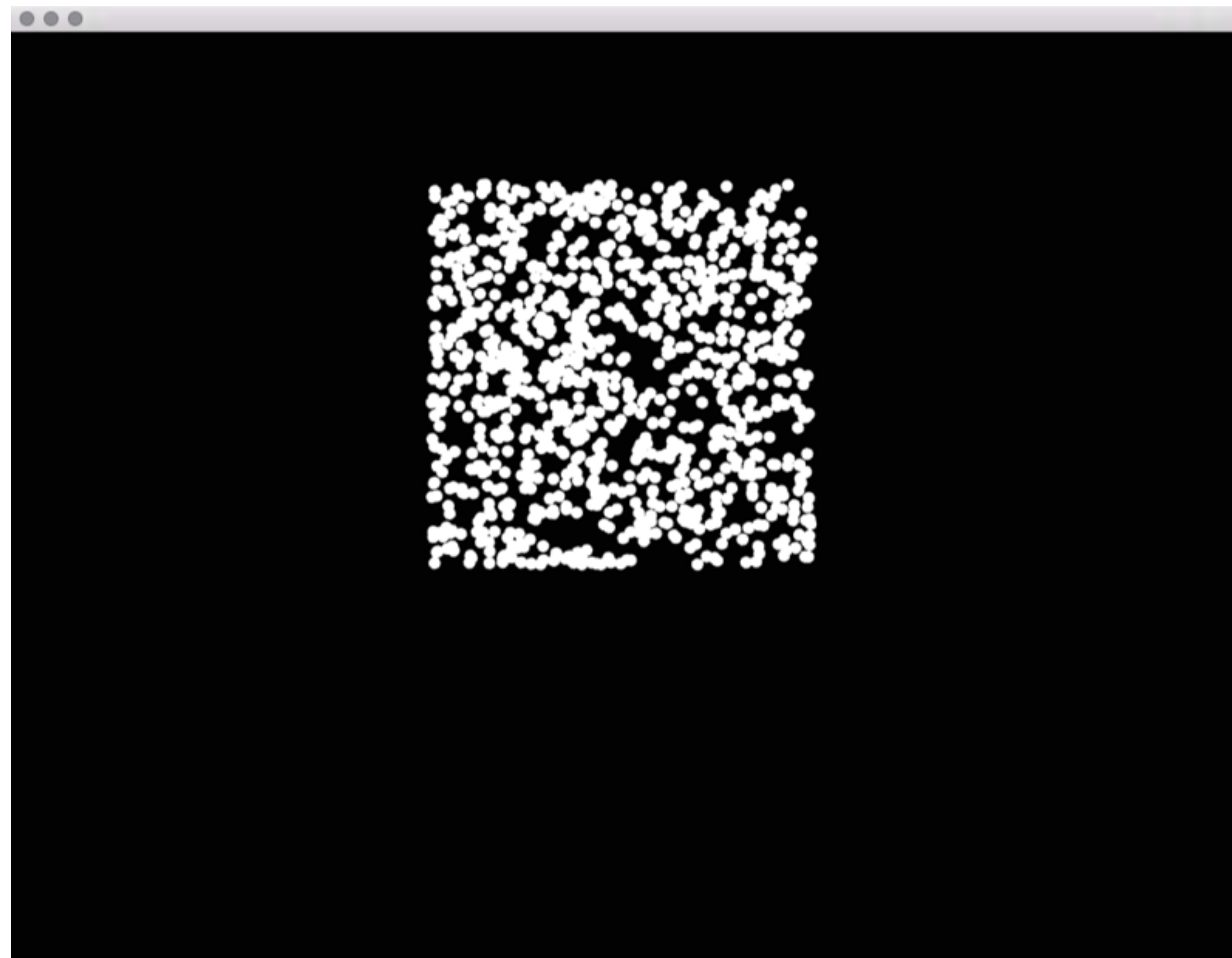
void ofApp::draw(){
    for (int i = 0; i < num; i++) {
        particles[i].draw();
    }
}

(中略)

void ofApp::mouseReleased(int x, int y, int button){
    for (int i = 0; i < num; i++) {
        particles[i].position = ofVec2f(x, y);
        particles[i].velocity = ofVec2f(0, 0);
        particles[i].addForce(ofRandom(-10, 10), ofRandom(-20, 0));
    }
}
```


大量の物体を同時に運動させる

- ▶ 完成! … がなんだか四角い?
- ▶ ランダムな生成方法に工夫が必要



大量の物体を同時に運動させる

▶ testApp.cpp : mouseReleasedを変更

(前略)

```
void ofApp::mouseReleased(int x, int y, int button){
    for (int i = 0; i < num; i++) {
        particles[i].position.set(x, y);
        particles[i].velocity.set(0, 0);
        float length = ofRandom(20.0);    // 円の半径をランダムに決定
        float angle = ofRandom(2.0 * PI); // 角度をランダムに決定
        // 決定した半径と角度から位置を決定
        ofVec2f force = ofVec2f(length * cos(angle), length * sin(angle));
        particles[i].addForce(force);
    }
}
```

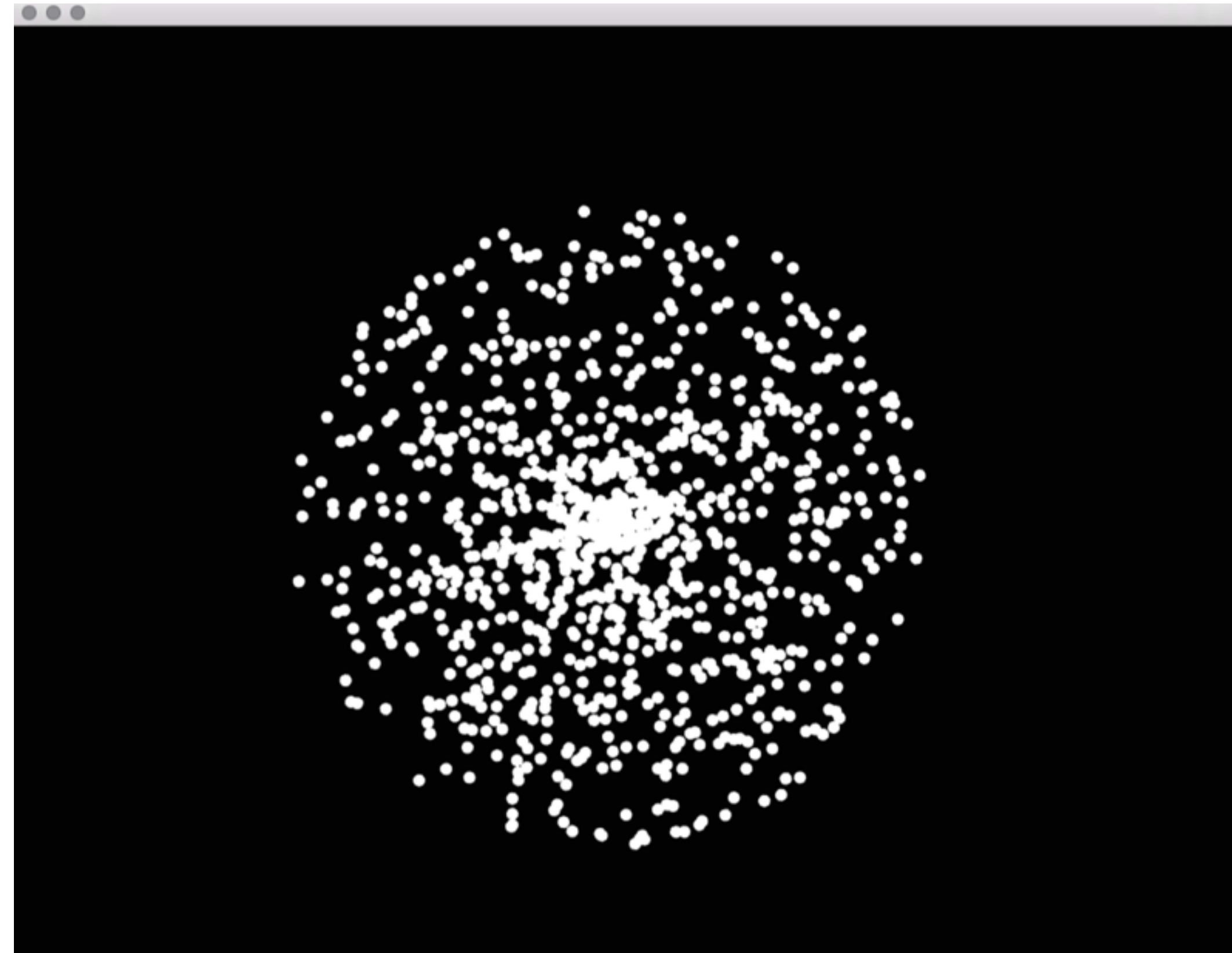
(後略)

大量の物体を同時に運動させる

- ▶ 完成! … がなんだか四角い?
- ▶ ランダムの生成方法に工夫が必要

大量の物体を同時に運動させる

- ▶ より自然なばらつきに!



大量の物体を同時に運動させる

- ▶ 物体の運動を指定した枠内に閉じ込めたい
- ▶ 壁で跳ね返る関数 `bounceOffWalls()` を実装
- ▶ ついでに、反対側へつきぬける `throughOffWalls()` も

大量の物体を同時に運動させる

▶ ParticleVec2.h

```
#pragma once
#include "ofMain.h"

class ParticleVec2 {
public:
    ParticleVec2();
    void update();
    void draw();

    void addForce(ofVec2f force);
    void addForce(float forceX, float forceY);

    void bounceOffWalls(); // 壁で跳ね返る
    void throughOffWalls(); // 反対側に突き抜ける

    ofVec2f position;
    ofVec2f velocity;
    ofVec2f acceleration;
    float friction;
    float radius;
    float mass;
    float maxx, maxy, minx, miny; // 枠の範囲
};
```

大量の物体を同時に運動させる

▶ ParticleVec2.cpp

```
#include "ParticleVec2.h"

ParticleVec2::ParticleVec2(){
    radius = 5.0;
    friction = 0.01;
    mass = 1.0;
    position = ofVec2f(ofGetWidth()/2.0, ofGetHeight()/2.0);
    velocity = ofVec2f(0, 0);
    minx = 0;
    miny = 0;
    maxx = ofGetWidth();
    maxy = ofGetHeight();
}
```

大量の物体を同時に運動させる

▶ ParticleVec2.cpp

(略)

```
void ParticleVec2::bounceOffWalls(){  
    if (position.x > maxx){  
        position.x = maxx;  
        velocity.x *= -1;  
    }  
    if (position.x < minx){  
        position.x = minx;  
        velocity.x *= -1;  
    }  
    if (position.y > maxy){  
        position.y = maxy;  
        velocity.y *= -1;  
    }  
    if (position.y < miny){  
        position.y = miny;  
        velocity.y *= -1;  
    }  
}
```

(略)

大量の物体を同時に運動させる

▶ ParticleVec2.cpp

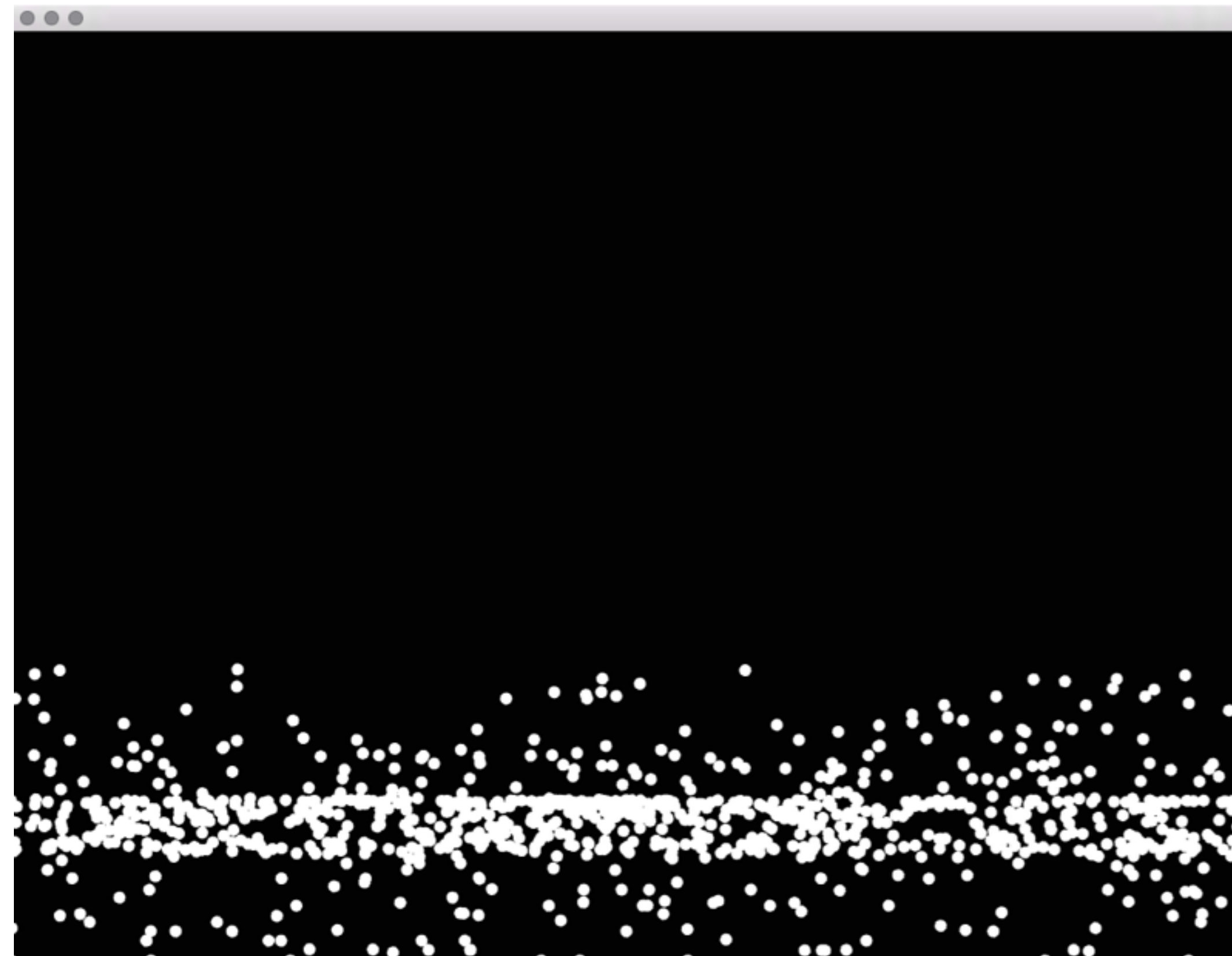
(略)

```
void ParticleVec2::throughOffWalls(){  
    if (position.x < minx) {  
        position.x = maxx;  
    }  
    if (position.y < miny) {  
        position.y = maxy;  
    }  
    if (position.x > maxx) {  
        position.x = minx;  
    }  
    if (position.y > maxy) {  
        position.y = miny;  
    }  
}
```

(略)

大量の物体を同時に運動させる

▶ 完成!!



大量の物体を同時に運動させる - VBO版

大量の物体を同時に運動させる - VBO版

- ▶ さらに、大量の物体を同時に扱いたい
- ▶ VBOで、GPUのパワーをフル活用
- ▶ VBO : Vertex Buffer Object
- ▶ 大量の頂点（Vertex）の情報をGPUで高速に処理することが可能なOpenGLの機能
- ▶ ofVboMesh を使うと楽

大量の物体を同時に運動させる - VBO版

▶ ofApp.h

```
#pragma once

#include "ofMain.h"
#include "ParticleVec2.h"

class ofApp : public ofBaseApp{
public:
    void setup();
    void update();
    void draw();

    void keyPressed(int key);
    void keyReleased(int key);
    void mouseMoved(int x, int y );
    void mouseDragged(int x, int y, int button);
    void mousePressed(int x, int y, int button);
    void mouseReleased(int x, int y, int button);
    void windowResized(int w, int h);
    void dragEvent(ofDragInfo dragInfo);
    void gotMessage(ofMessage msg);

    static const int num = 10000;
    ParticleVec2 particles[num];
    ofVboMesh mesh; // VBOを使用したメッシュで描画を高速化
};
```

大量の物体を同時に運動させる - VBO版

▶ ofApp.cpp

```
#include "ofApp.h"

void ofApp::setup(){
    ofSetFrameRate(60);
    ofBackground(0);
    mesh.setMode(OF_PRIMITIVE_POINTS); // メッシュの頂点を点で描画
    glPointSize(3.0);
}

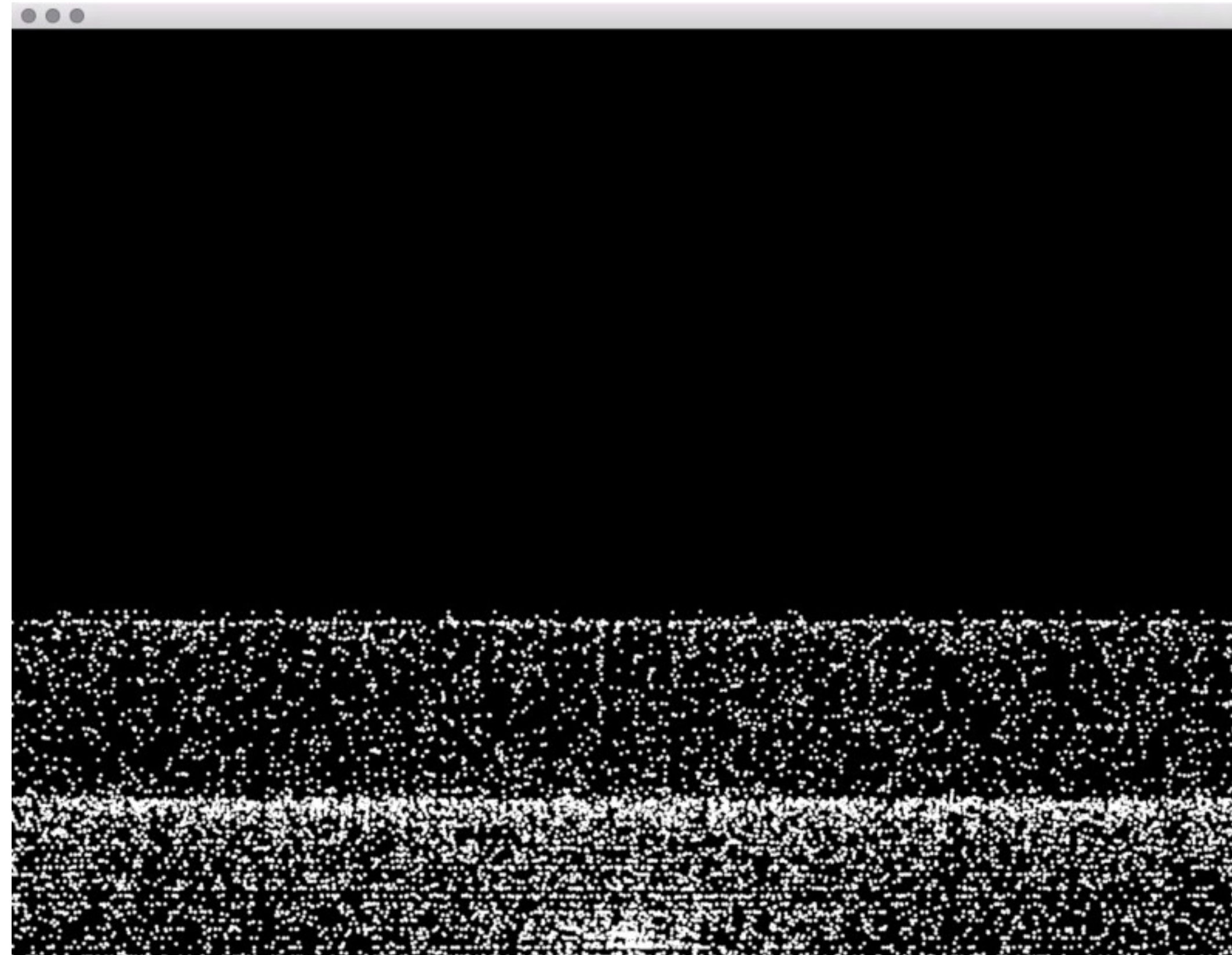
void ofApp::update(){
    mesh.clear(); // メッシュを初期化
    for (int i = 0; i < num; i++) {
        particles[i].addForce(0, 0.2);
        particles[i].update();
        particles[i].bounceOffWalls();
        // 物体の位置をメッシュの頂点に格納
        mesh.addVertex(ofVec3f(particles[i].position.x, particles[i].position.y));
    }
}

void ofApp::draw(){
    mesh.draw(); // メッシュを描画
}
```

(略)

大量の物体を同時に運動させる - VBO版

▶ 完成!!

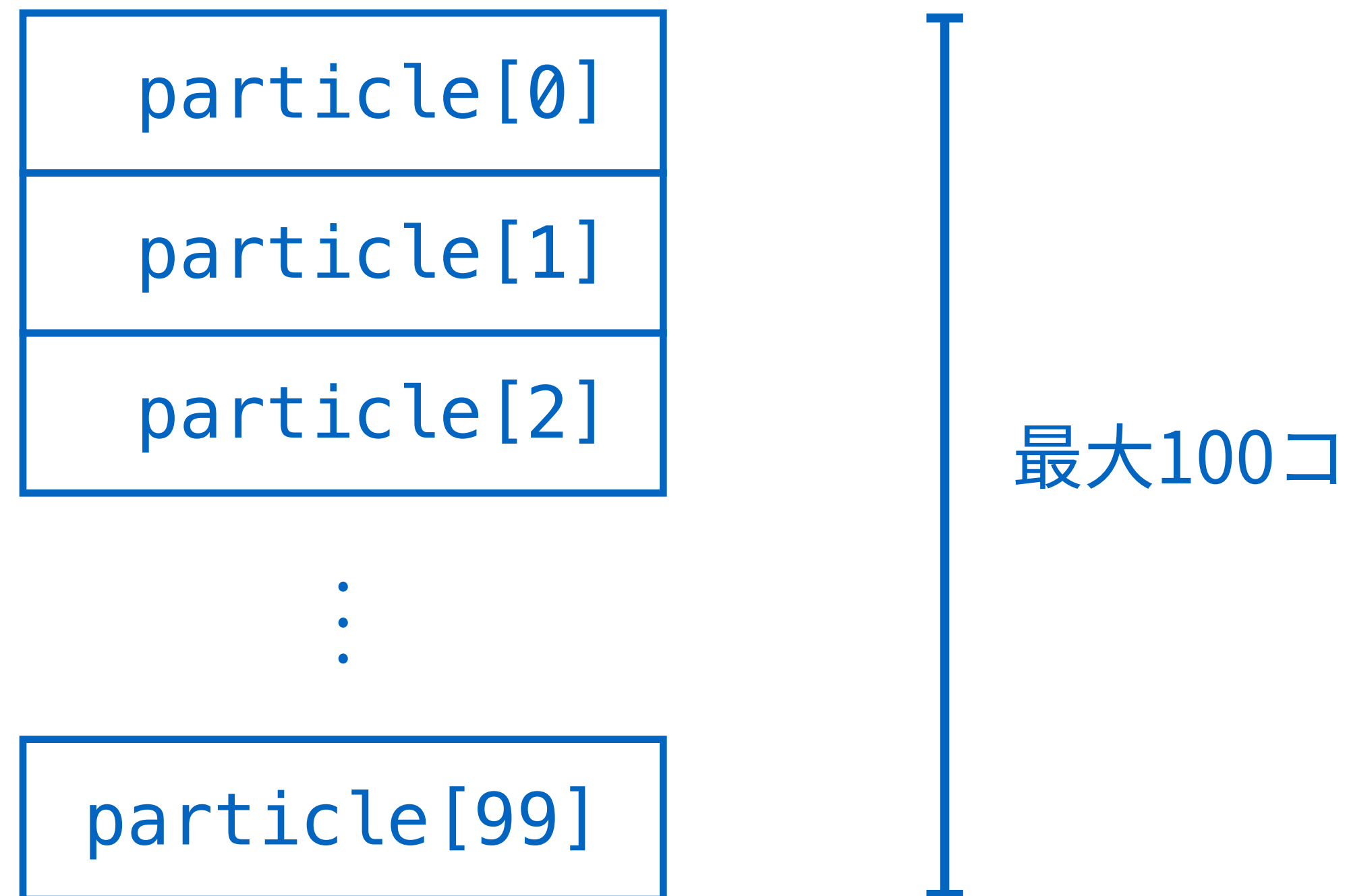


最大数を決めずに配列を使いたい
可変長配列 (vector) をつかう

可変長配列 (vector) をつかう

- ▶ Arrayは、最初に最大数を決めなくてはならない
- ▶ あらかじめ、最大数がわからない場合どうすれば良いのか?

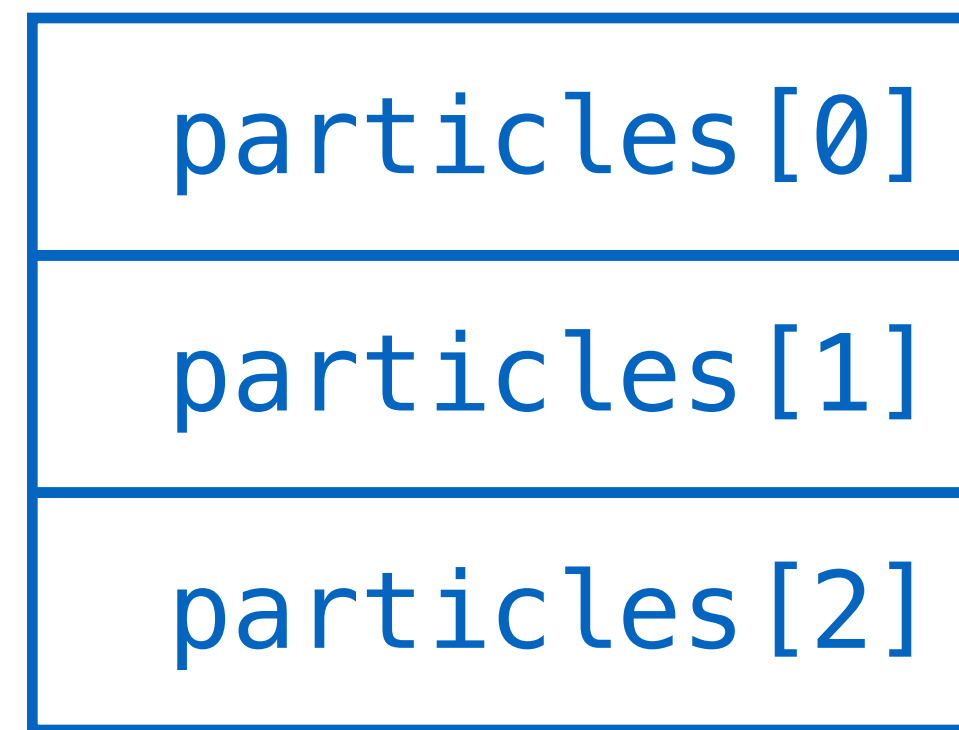
ParticleVec2 particle[100]



可変長配列 (vector) をつかう

- ▶ 配列の長さを自由に変更できる、可変長配列というものがある!
- ▶ C++では、vector、deque、list など様々な可変長配列が用意されている
- ▶ 今回は vector をつかってみる (運動ベクトルなどのベクトルとは関係ない)

`vector<ParticleVec2> particles`



最大数は可変

可変長配列（vector）をつかう

- ▶ ベクトルの宣言の例

```
vector<ParticleVec2> particles;
```

- ▶ vector<クラス名> 配列名;

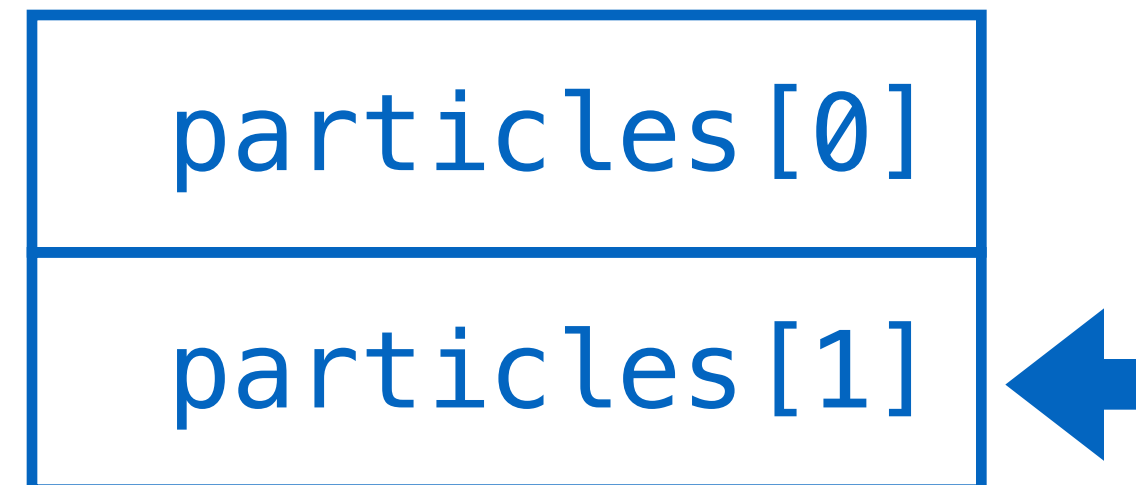
可変長配列 (vector) をつかう

- ▶ vectorを宣言した状態では、中身は空
- ▶ vectorにオブジェクトを push_back すると後ろに追加されていく

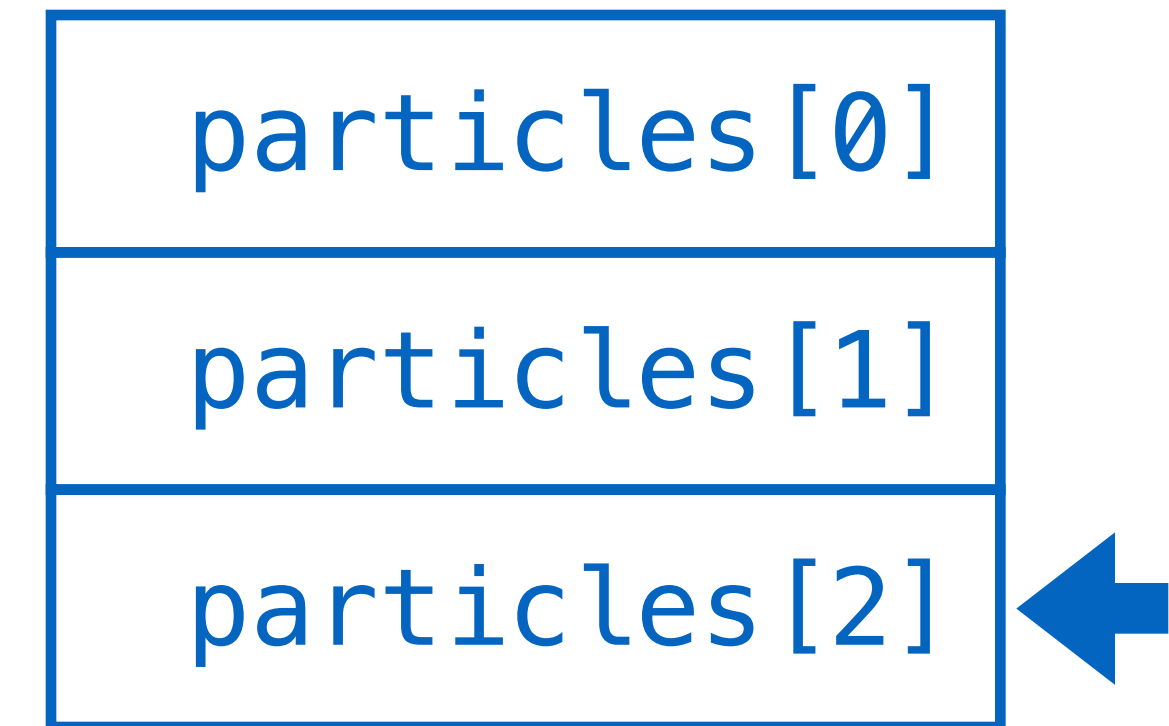
particles.push_back(p)



particles.push_back(p)



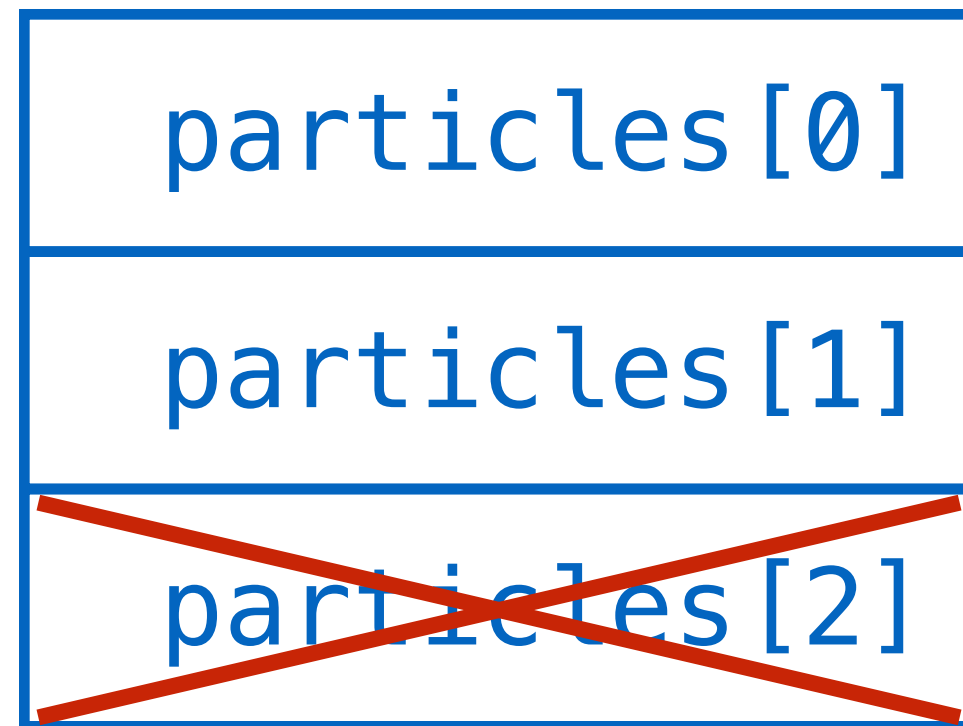
particles.push_back(p)



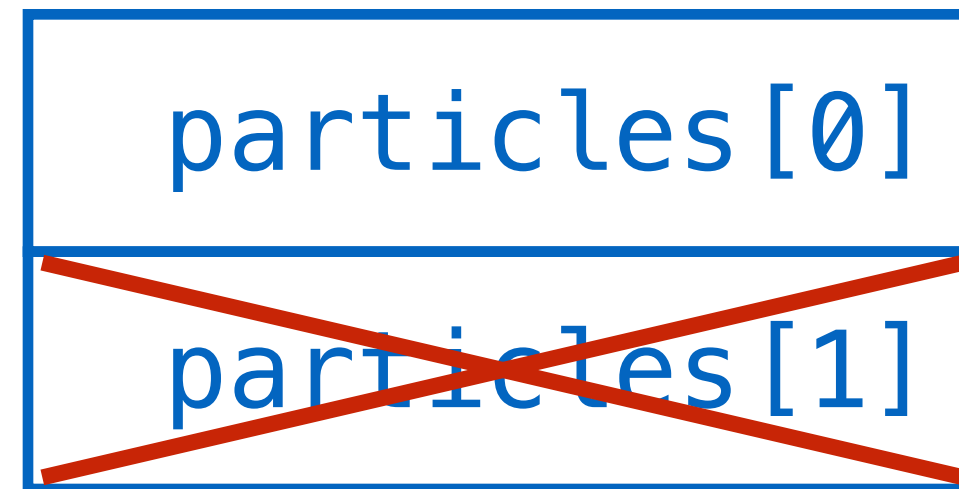
可変長配列 (vector) をつかう

- ▶ vectorにオブジェクトを pop_back すると最後の要素が消える

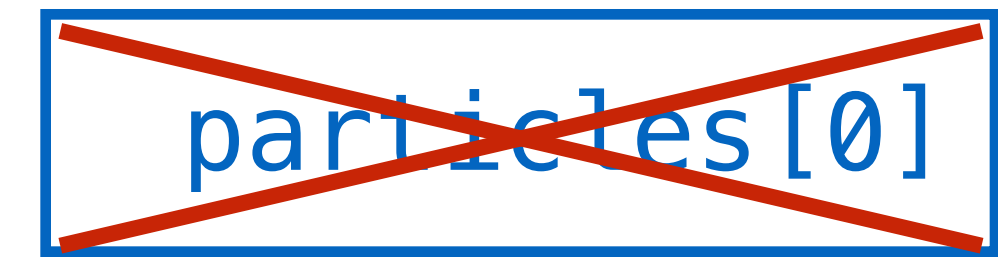
particles.pop_back(p)



particles.pop_back(p)



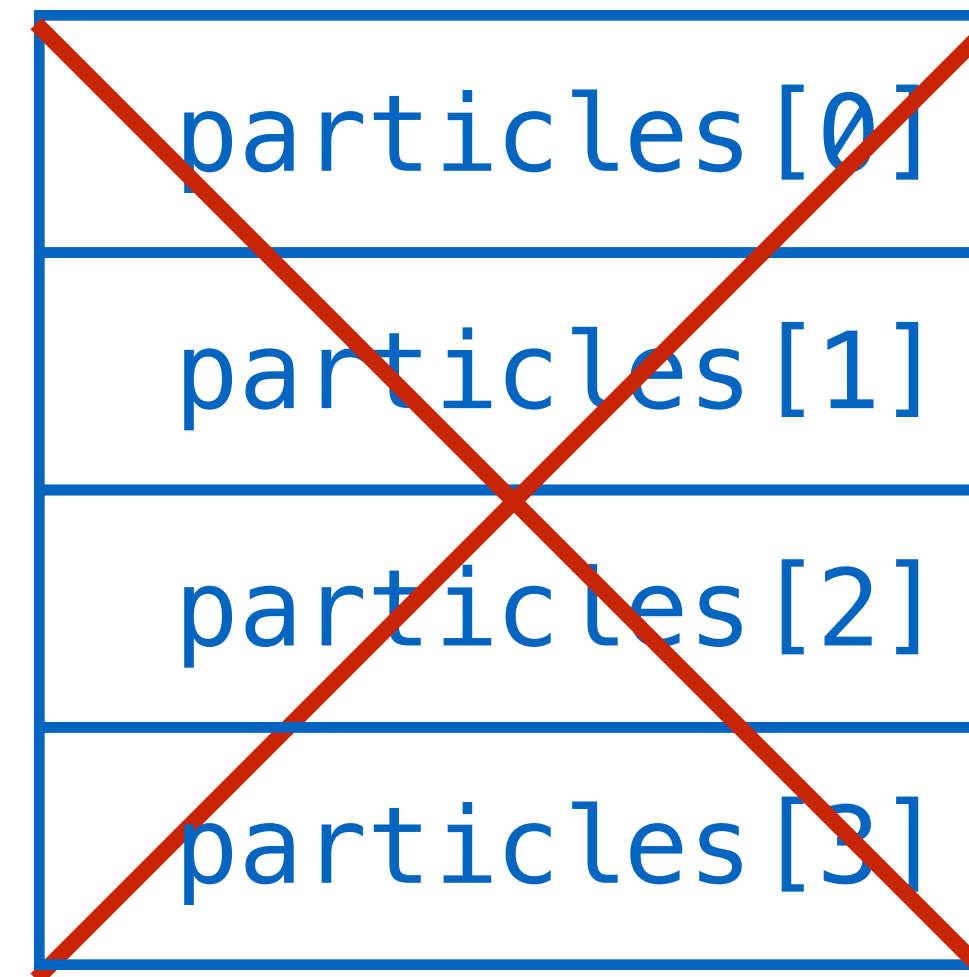
particles.pop_back(p)



可変長配列 (vector) をつかう

- ▶ vectorにオブジェクトをclearすると、全てが一気に消える

particles.clear()



particles[0]
particles[1]
particles[2]
particles[3]

可変長配列 (vector) をつかう

- ▶ vectorを活用して、以下のようにしてみたい
- ▶ マウスをドラッグしている間は最大個数の制限なしにパーティクルが増えつつづける
- ▶ キーボードで何か入力するとクリア
- ▶ ofAppクラスだけの変更でOK

可変長配列 (vector) をつかう

▶ ofApp.h

```
#pragma once

#include "ofMain.h"
#include "ParticleVec2.h"

class ofApp : public ofBaseApp{
public:
    void setup();
    void update();
    void draw();

    void keyPressed(int key);
    void keyReleased(int key);
    void mouseMoved(int x, int y );
    void mouseDragged(int x, int y, int button);
    void mousePressed(int x, int y, int button);
    void mouseReleased(int x, int y, int button);
    void windowResized(int w, int h);
    void dragEvent(ofDragInfo dragInfo);
    void gotMessage(ofMessage msg);

    static const int num = 10000;
    vector<ParticleVec2> particles; // 可変長配列
    ofVboMesh mesh;
};
```


可変長配列 (vector) をつかう

▶ ofApp.cpp

```
#include "ofApp.h"

void ofApp::setup(){
    ofSetFrameRate(60);
    ofBackground(0);
    mesh.setMode(OF_PRIMITIVE_POINTS);
    glPointSize(3.0);
}

void ofApp::update(){
    mesh.clear();
    for (int i = 0; i < particles.size(); i++) {
        particles[i].update();
        particles[i].bounceOffWalls();
        mesh.addVertex(ofVec3f(particles[i].position.x, particles[i].position.y));
    }
}

void ofApp::draw(){
    mesh.draw();
}

void ofApp::keyPressed(int key){
    // キー入力で全消去
    particles.clear();
}
```

可変長配列 (vector) をつかう

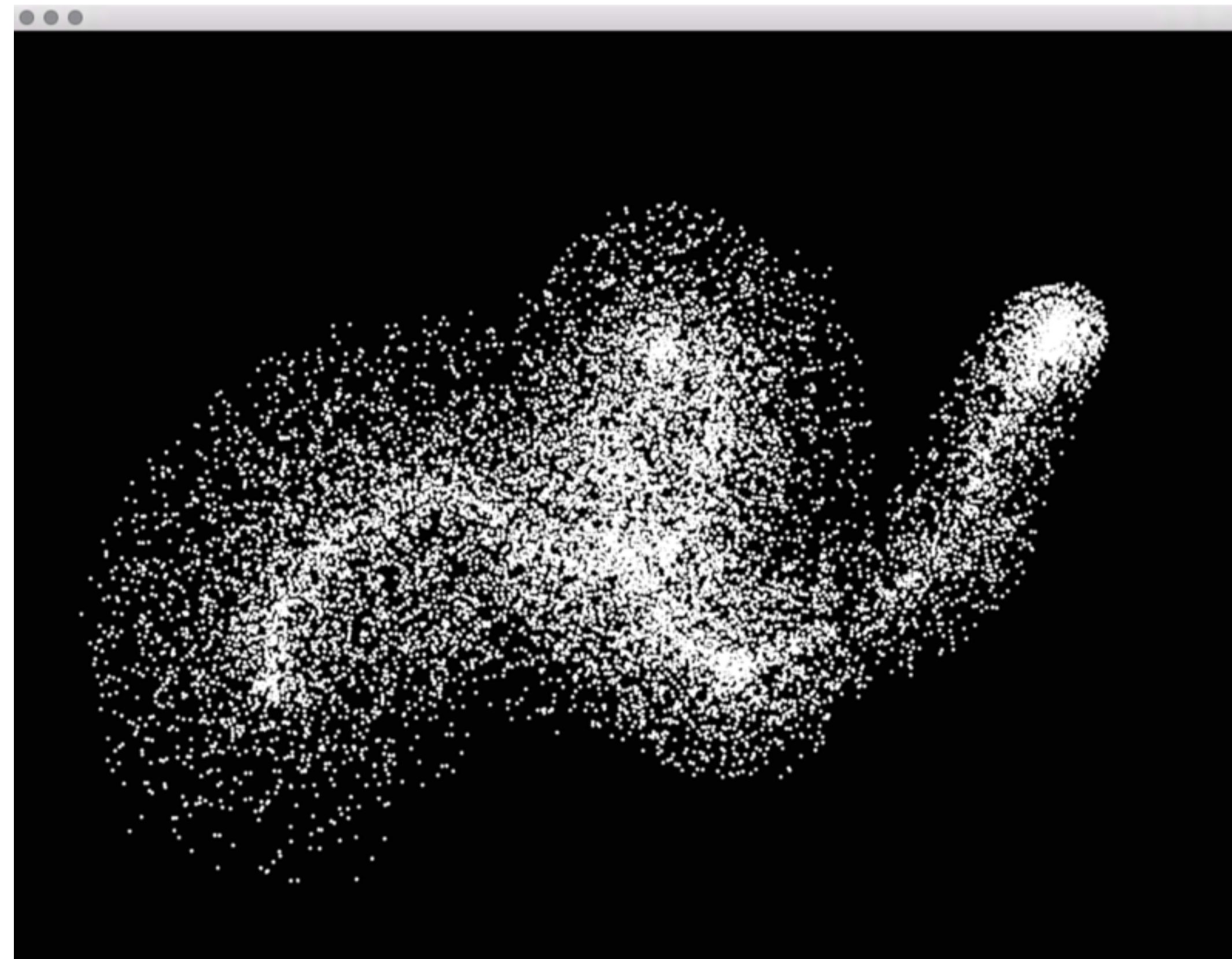
▶ ofApp.cpp

(中略)

```
void ofApp::mouseDragged(int x, int y, int button){  
    // 一度に100個ずつ生成  
    for (int i = 0; i < 100; i++) {  
        // まずテンポラリーなオブジェクトを生成して初期設定する  
        ParticleVec2 p;  
        p.position = ofVec2f(x, y);  
        float length = ofRandom(2.0);  
        float angle = ofRandom(2.0 * PI);  
        ofVec2f velocity = ofVec2f(length * cos(angle), length * sin(angle));  
        p.velocity = velocity;  
        // 生成したオブジェクトを可変長配列の末尾に追加  
        particles.push_back(p);  
    }  
}
```

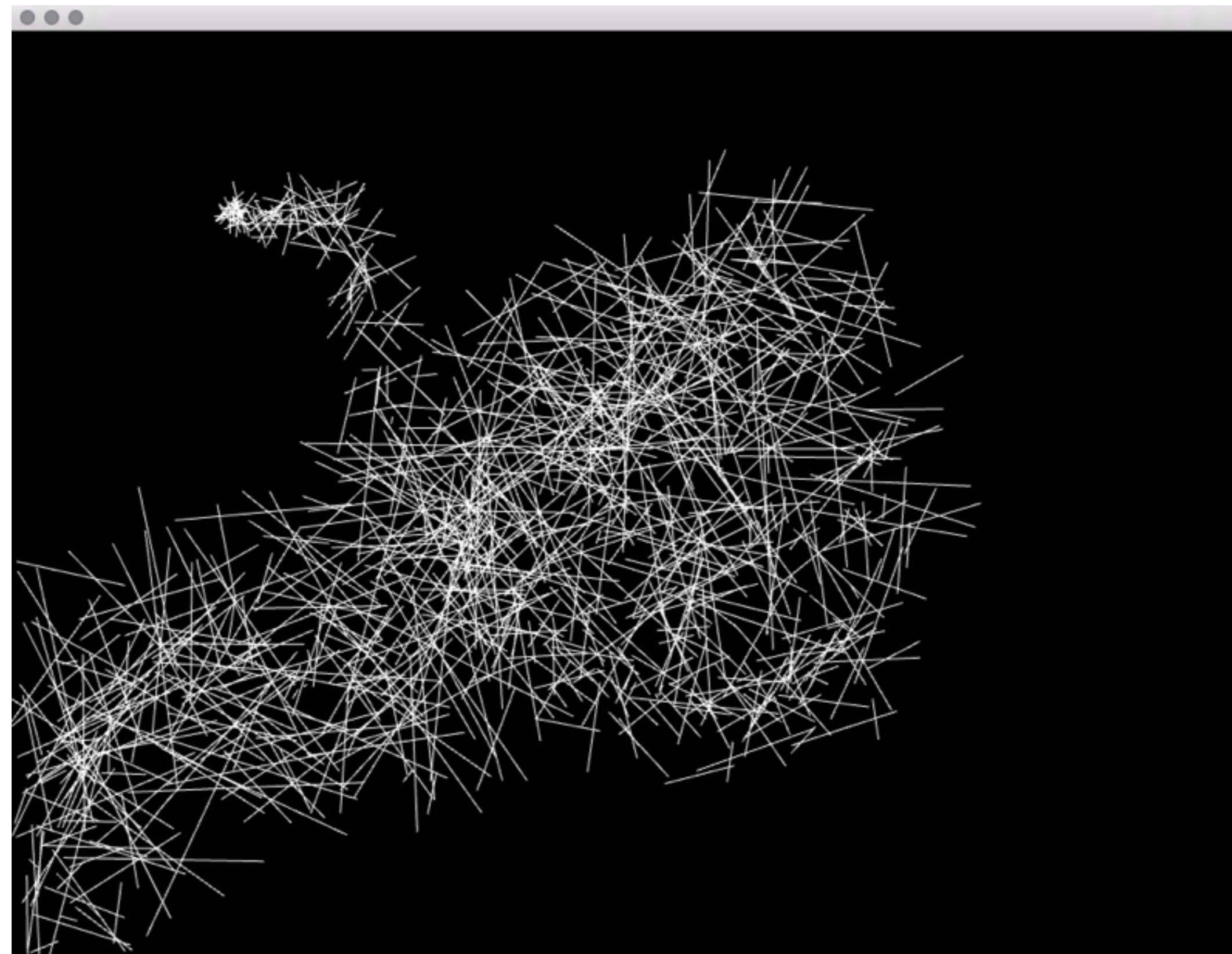
可変長配列 (vector) をつかう

▶ 完成!!



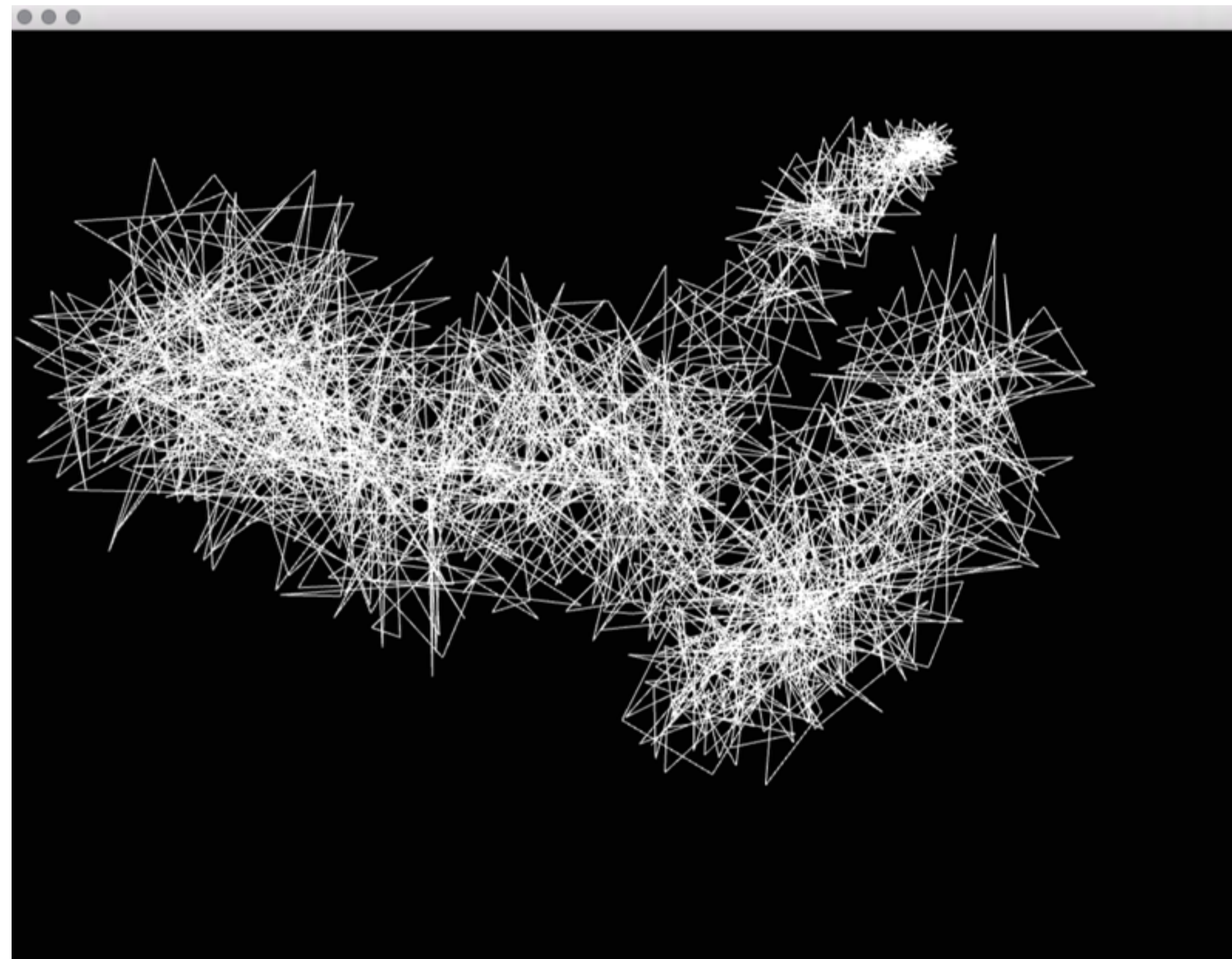
可変長配列 (vector) をつかう

- ▶ vboMeshの描き方で、様々に変化する
- ▶ `mesh.setMode(OF_PRIMITIVE_LINES);` だと…



可変長配列 (vector) をつかう

- ▶ vboMeshの描き方で、様々に変化する
- ▶ `mesh.setMode(OF_PRIMITIVE_LINE_STRIP);` だと

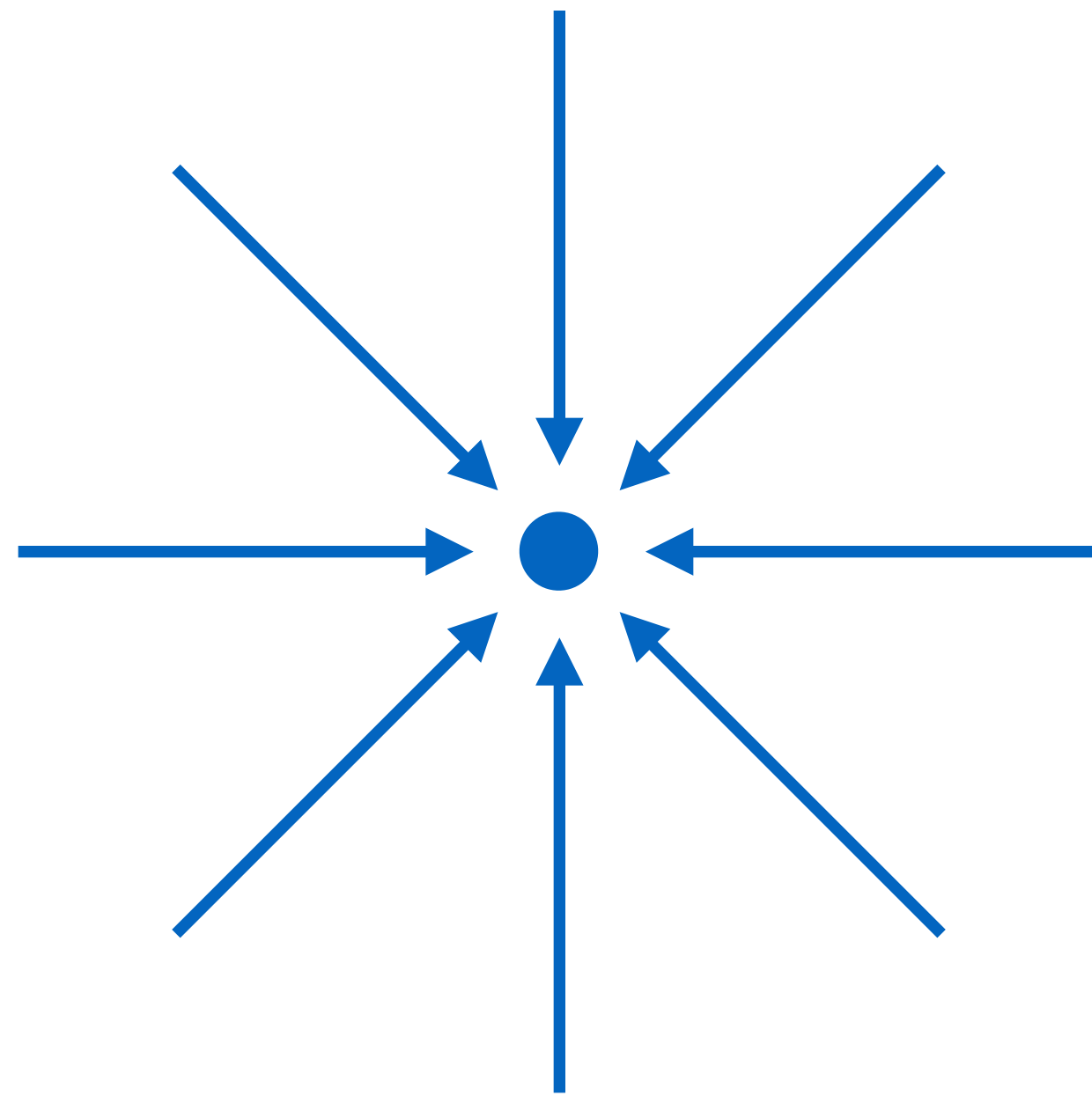


引き付ける力 : Attraction、反発する力 : Repulsion

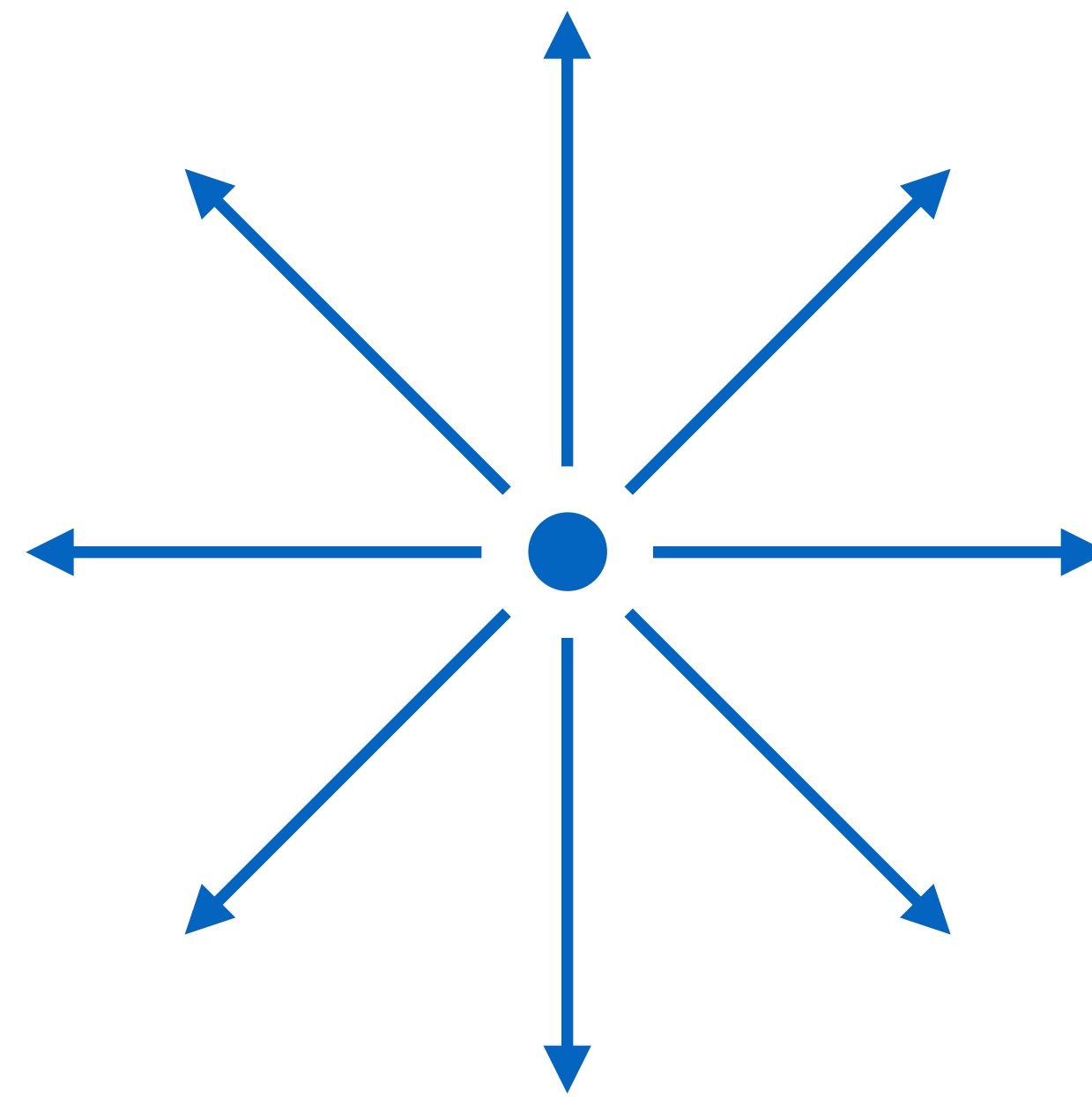
引き付ける力 : Attraction、反発する力 : Repulsion

- ▶ 引き付ける力 (Attraction) と、反発する力 (Repulsion) を実装してみる
- ▶ 指定した点に力が働くようにデザインする

Attraction



Repulsion



引き付ける力 : Attraction、反発する力 : Repulsion

- ▶ ParticleVec2 クラスにそれぞれ関数として追加
- ▶ addAttractionForce()
- ▶ addRepulsionForce()

引き付ける力 : Attraction、反発する力 : Repulsion

▶ ParticleVec2.h

```
#pragma once
#include "ofMain.h"

class ParticleVec2 {
public:
    ParticleVec2();
    void update();
    void draw();
    void addForce(ofVec2f force);
    void addForce(float forceX, float forceY);
    void bounceOffWalls();
    void throughOffWalls();

    // 引き付ける力
    void addAttractionForce(float x, float y, float radius, float scale);
    void addAttractionForce(ParticleVec2 &p, float radius, float scale);
    // 反発する力
    void addRepulsionForce(float x, float y, float radius, float scale);
    void addRepulsionForce(ParticleVec2 &p, float radius, float scale);

    ofVec2f position;
    ofVec2f velocity;
    ofVec2f acceleration;
    float friction;
    float radius;
    float mass;
    float maxx, maxy, minx, miny;
};
```

引き付ける力 : Attraction、反発する力 : Repulsion

▶ ParticleVec2.cpp

(略)

```
void ParticleVec2::addAttractionForce(float x, float y, float radius, float scale){
    ofVec2f pos0fForce;
    pos0fForce.set(x,y);
    ofVec2f diff = position - pos0fForce;
    float length = diff.length();
    bool bAmCloseEnough = true;
    if (radius > 0){
        if (length > radius){
            bAmCloseEnough = false;
        }
    }
    if (bAmCloseEnough == true){
        float pct = 1 - (length / radius);
        diff.normalize();
        acceleration = acceleration - diff * scale * pct;
    }
}
```

(略)

引き付ける力 : Attraction、反発する力 : Repulsion

▶ ParticleVec2.cpp

(略)

```
void ParticleVec2::addRepulsionForce(float x, float y, float radius, float scale){
    ofVec2f pos0fForce;
    pos0fForce.set(x,y);
    ofVec2f diff = position - pos0fForce;
    float length = diff.length();
    bool bAmCloseEnough = true;
    if (radius > 0){
        if (length > radius){
            bAmCloseEnough = false;
        }
    }
    if (bAmCloseEnough == true){
        float pct = 1 - (length / radius);
        diff.normalize();
        acceleration = acceleration + diff * scale * pct;
    }
}
```

(略)

引き付ける力 : Attraction、反発する力 : Repulsion

▶ ofApp.h

```
#pragma once

#include "ofMain.h"
#include "ParticleVec2.h"

class ofApp : public ofBaseApp{
public:
    void setup();
    void update();
    void draw();

    void keyPressed(int key);
    void keyReleased(int key);
    void mouseMoved(int x, int y );
    void mouseDragged(int x, int y, int button);
    void mousePressed(int x, int y, int button);
    void mouseReleased(int x, int y, int button);
    void windowResized(int w, int h);
    void dragEvent(ofDragInfo dragInfo);
    void gotMessage(ofMessage msg);

    static const int num = 50000;
    ParticleVec2 particles[num];
    ofVboMesh mesh;
    bool pressed;
};
```

引き付ける力 : Attraction、反発する力 : Repulsion

▶ ofApp.cpp

```
#include "ofApp.h"

void ofApp::setup(){
    ofSetFrameRate(60);
    ofBackground(0);
    mesh.setMode(OF_PRIMITIVE_POINTS);
    pressed = false;
    for (int i = 0; i < num; i++) {
        particles[i].position = ofVec2f(ofRandom(ofGetWidth()), ofRandom(ofGetHeight()));
    }
}

void ofApp::update(){
    mesh.clear();
    for (int i = 0; i < num; i++) {
        if (pressed) {
            particles[i].addAttractionForce(mouseX, mouseY, 1000, 1.0);
        }
        particles[i].update();
        particles[i].throughOffWalls();
        mesh.addVertex(ofVec3f(particles[i].position.x, particles[i].position.y));
    }
}

void ofApp::draw(){
    mesh.draw();
}
```

引き付ける力 : Attraction、反発する力 : Repulsion

▶ ofApp.cpp

(略)

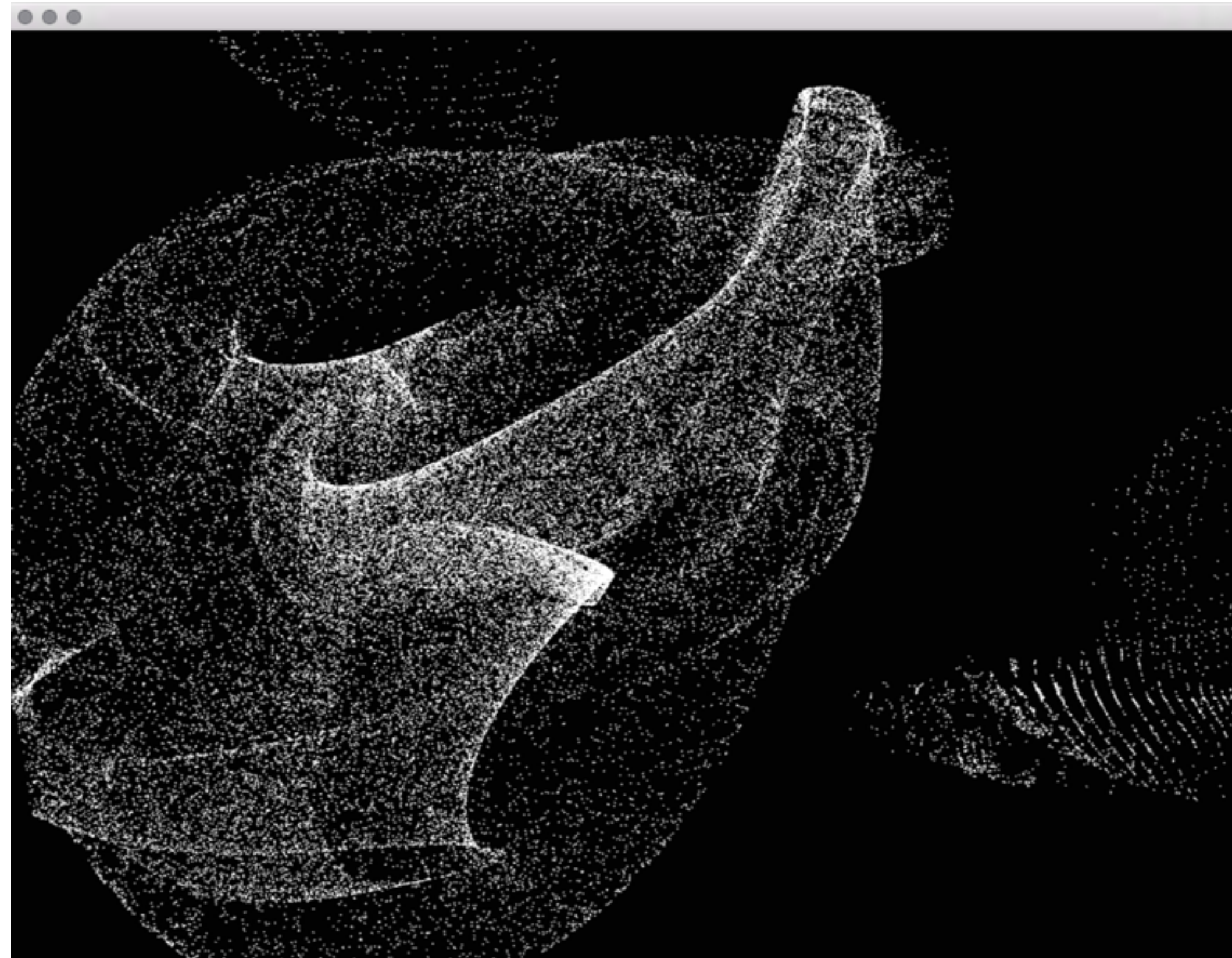
```
void ofApp::mousePressed(int x, int y, int button){  
    pressed = true;  
}
```

```
void ofApp::mouseReleased(int x, int y, int button){  
    pressed = false;  
}
```

(略)

引き付ける力 : Attraction、反発する力 : Repulsion

- ▶ マウスに引き付けられるパーティクル!!



引き付ける力 : Attraction、反発する力 : Repulsion

- ▶ 参考: 3Dバージョン!
- ▶ <https://vimeo.com/127222357>

