

IDD Media lab. 2015

openFrameworks

3Dグラフィクス、OpenGL

2015年6月9日

多摩美術大学 情報デザイン学科 情報芸術コース

田所 淳

今日の内容

- ▶ openFrameworksの、3D機能、OpenGLの機能を中心に取り上げます

今日の内容

- ▶ openFrameworksの3DやOpenGLに関する機能を概観します

クラス名	機能
ofBox, ofSphere	3Dプリミティブオブジェクトの描画
ofEasyCam	簡単に利用可能なカメラ(視点)機能
ofLight	ライティング効果
ofMesh	3D物体の全ての頂点情報を持ったデータ
ofVBO	3Dシーンの物体の情報(位置・色)の保存
ofShader	プログラマブルシェーダ

3Dプリミティブ図形の描画

ofBox、ofSphere

3Dプリミティブ図形の描画 ofBox、ofSphere

- ▶ openFrameworks v080から、3Dの基本図形の書き方が変化
- ▶ 球体: ofSphere → ofSpherePrimitive
- ▶ 立方体: ofBox → ofBoxPrimitive
- ▶ ofRectやofCircleのように、testApp::draw() 関数内に記述して座標とサイズを指定するだけで簡単に利用可能

3Dプリミティブ図形の描画 ofBox、ofSphere

▶ testApp.h

```
#pragma once

#include "ofMain.h"

class testApp : 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);

    ofBoxPrimitive box; // 立方体プリミティブ
    ofSpherePrimitive sphere; // 球プリミティブ
};
```

3Dプリミティブ図形の描画 ofBox、ofSphere

▶ testApp.cpp

```
#include "testApp.h"

void testApp::setup(){
    ofBackground(0);
}

void testApp::update(){

}

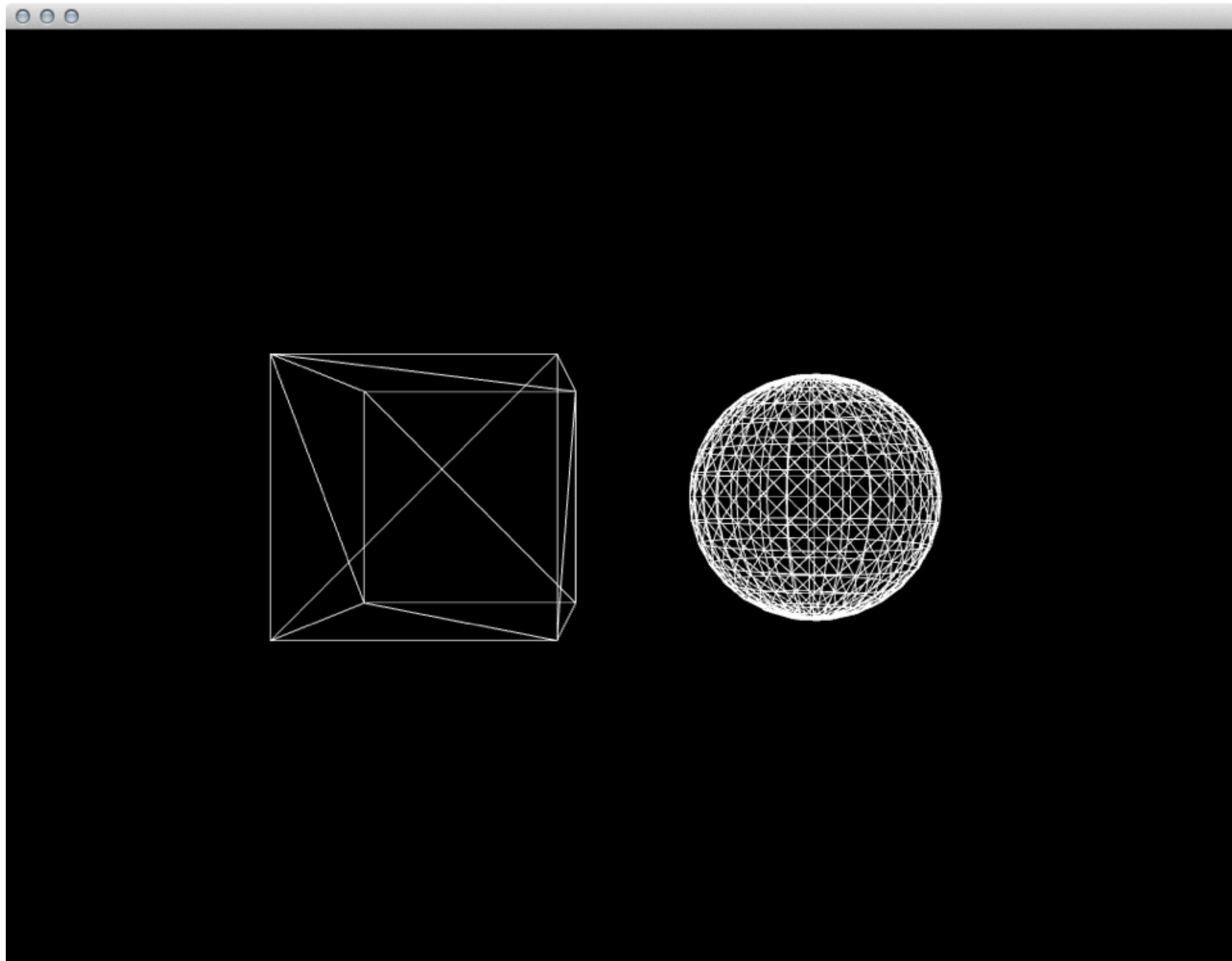
void testApp::draw(){
    ofTranslate(ofGetWidth()/2, ofGetHeight()/2);
    ofSetColor(255);

    // 立方体
    box.set(200); // サイズ設定
    box.setPosition(-150, 0, 0); // 位置
    box.drawWireframe(); // ワイヤーフレームを描画

    // 球
    sphere.set(100, 16); // 半径と面の細かさ
    sphere.setPosition(150, 0, 0); // 位置
    sphere.drawWireframe(); // ワイヤーフレームを描画
}
```

3Dプリミティブ図形の描画 ofBox、ofSphere

▶ 完成!!



視点をインタラクティブに移動
ofEasyCam

視点をインタラクティブに移動 - ofEasyCam

- ▶ 次に、この図形を様々な角度から眺められるようにする
- ▶ 物体を動かすのではなく、視点(カメラ)を移動して物体を注視する角度を変更してみる
- ▶ 視点を指定するofEasyCamというクラスがあり、とても簡単に視点の移動が可能
- ▶ より詳細な視点や画角などの指定には、ofCameraというクラスもある

視点をインタラクティブに移動 - ofEasyCam

▶ testApp.h

```
#pragma once

#include "ofMain.h"

class testApp : 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);

    ofBoxPrimitive box; // 立方体プリミティブ
    ofSpherePrimitive sphere; // 球プリミティブ
    ofEasyCam cam; // カメラ
};
```

視点をインタラクティブに移動 - ofEasyCam

▶ testApp.cpp

... (前略) ...

```
void testApp::draw(){
    cam.begin(); // カメラ開始

    ofSetColor(255);

    // 立方体
    box.set(200); // サイズ設定
    box.setPosition(-150, 0, 0); // 位置
    box.drawWireframe(); // ワイヤーフレームを描画

    // 球
    sphere.set(100, 16); // 半径と面の細かさ
    sphere.setPosition(150, 0, 0); // 位置
    sphere.drawWireframe(); // ワイヤーフレームを描画

    cam.end(); // カメラ終了
}
```

視点をインタラクティブに移動 - ofEasyCam

▶ testApp.cpp

... (前略) ...

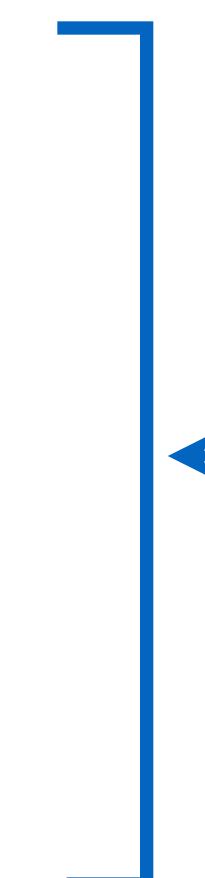
```
void testApp::draw(){
    cam.begin(); // カメラ開始

    ofSetColor(255);

    // 立方体
    box.set(200); // サイズ設定
    box.setPosition(-150, 0, 0); // 位置
    box.drawWireframe(); // ワイヤーフレームを描画

    // 球
    sphere.set(100, 16); // 半径と面の細かさ
    sphere.setPosition(150, 0, 0); // 位置
    sphere.drawWireframe(); // ワイヤーフレームを描画

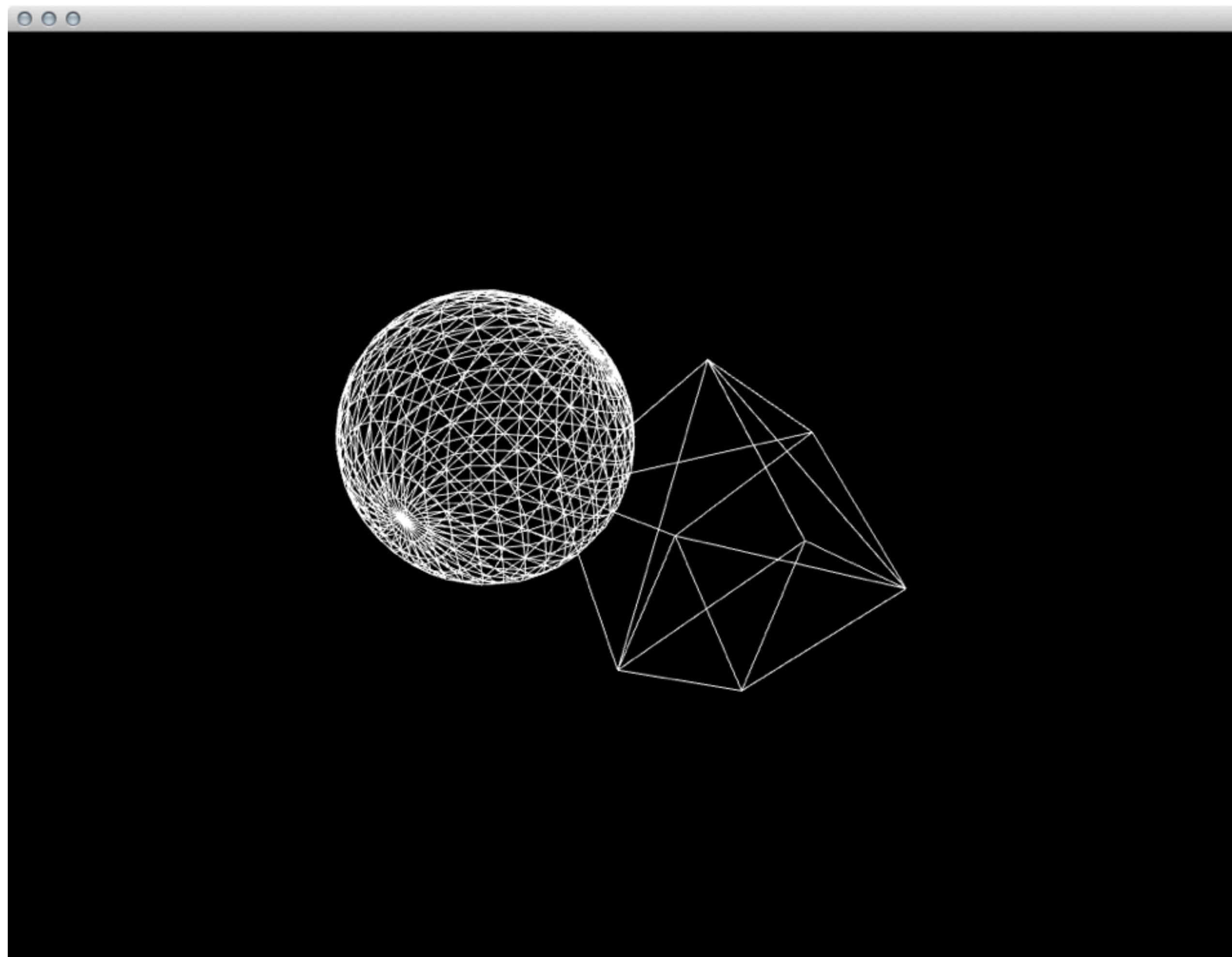
    cam.end(); // カメラ終了
}
```



camera.begin()と
camera.end()で囲まれた範囲
が、マウスのドラッグで視点
変更可能となる

視点をインタラクティブに移動 - ofEasyCam

- ▶ マウスをドラッグすると物体を回転できる



ライティング
ofLight

ライティング - ofLight

- ▶ プリミティブを描画する命令
 - ▶ プリミティブ.drawWireframe() → ワイヤーフレーム
 - ▶ プリミティブ.draw() → 塗り潰し
- ▶ 実際にやってみる

ライティング - ofLight

▶ testApp.cpp

```
void testApp::draw(){
    cam.begin(); // カメラ開始

    ofSetColor(255);

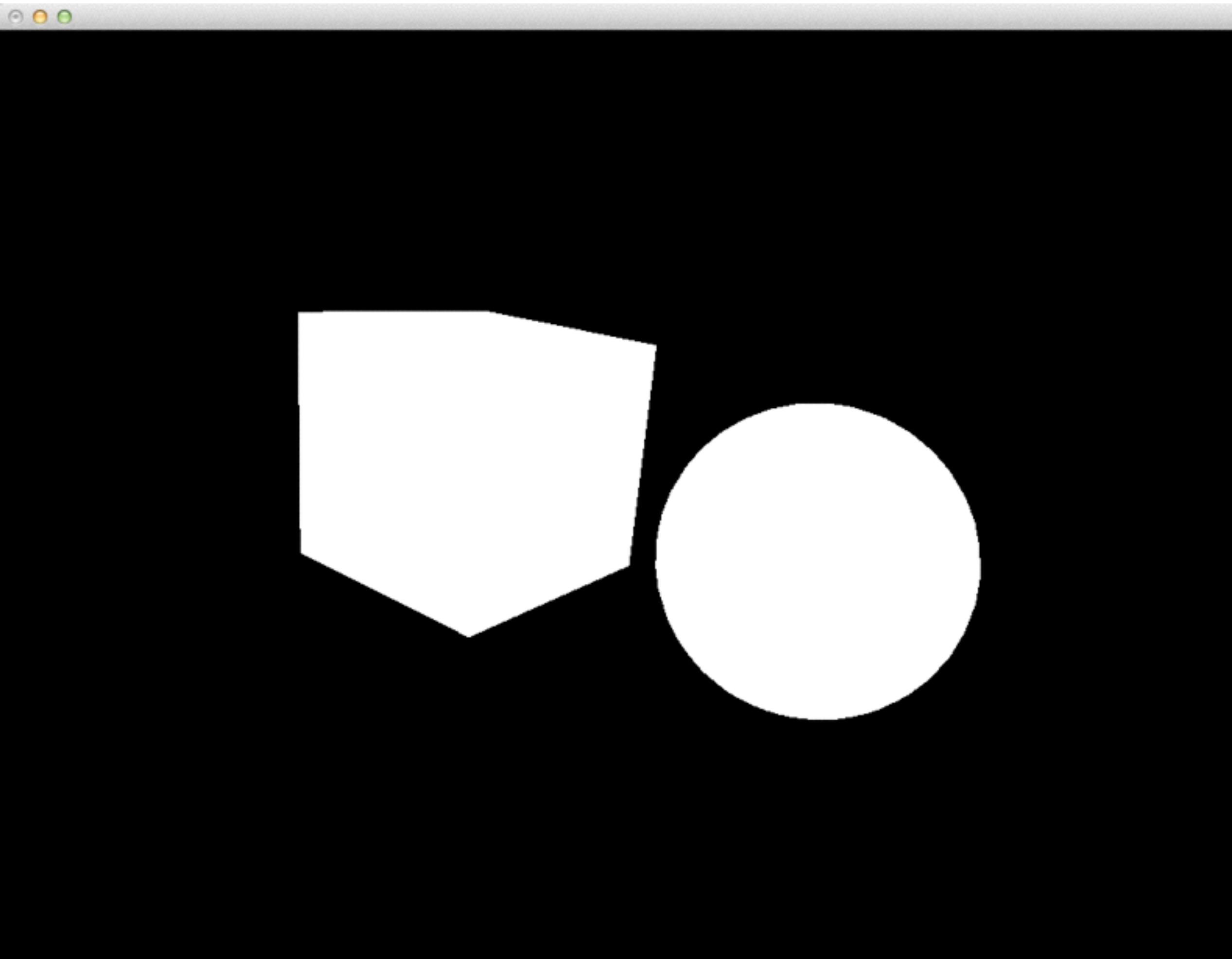
    // 立方体
    box.set(200); // サイズ設定
    box.setPosition(-150, 0, 0); // 位置
    box.draw(); // 塗りつぶしで描画 ←ココ

    // 球
    sphere.set(100, 16); // 半径と面の細かさ
    sphere.setPosition(150, 0, 0); // 位置
    sphere.draw(); // 塗りつぶしで描画 ←ココ

    cam.end(); // カメラ終了
}
```

ライティング - ofLight

- ▶ 陰影も何もない、のっぺりとした図形になってしまう



ライティング - ofLight

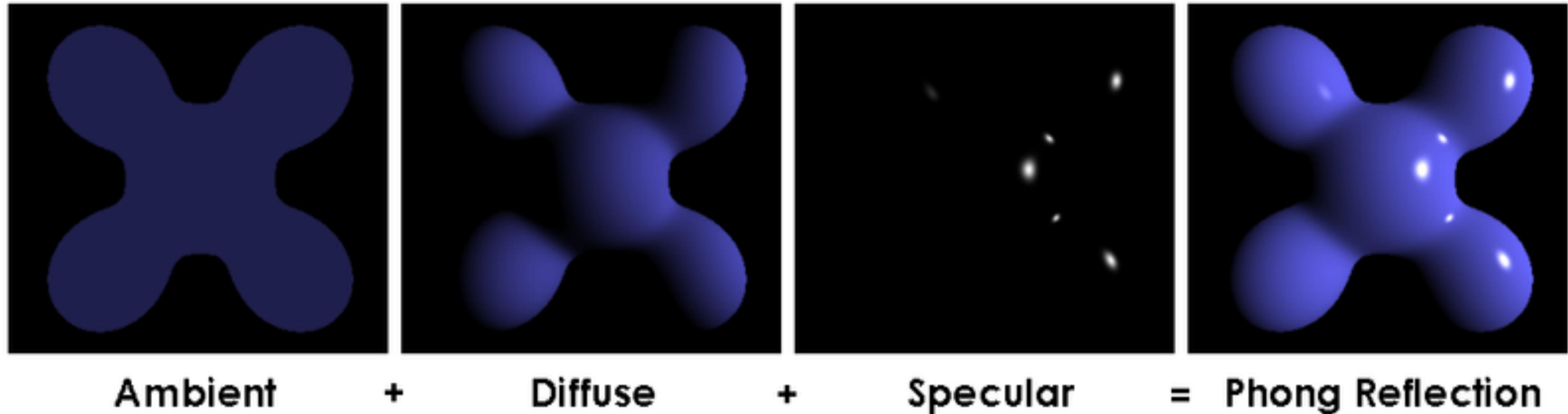
- ▶ なぜ、陰影のないのっぺりとした図形になってしまうのか?
- ▶ ライティングがされていないから
- ▶ 適切な照明を物体に照射することで、リアルな陰影が表現される
- ▶ ofLightクラスを使用する

ライティング - ofLight

- ▶ ofLightでは「Phongシェーディング」というアルゴリズムで、物体にリアルな陰影や反射を付加している
- ▶ 3Dグラフィックにおいて、モデリングされた面上の点に影をつけるための照明と陰影モデル
- ▶ ユタ大学のBui Tuong Phongによって開発 (1973)

ライティング - ofLight

- ▶ Phongシェーディング
- ▶ 3つの光の反射の強度と色を指定する
 - ▶ 鏡面反射 (Specular)
 - ▶ 拡散反射 (Diffuse)
 - ▶ 環境反射 (Ambient)



ライティング - ofLight

▶ testApp.h

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

class testApp : 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);

    ofBoxPrimitive box; // 立方体プリミティブ
    ofSpherePrimitive sphere; // 球プリミティブ
    ofEasyCam cam; // カメラ
    ofLight light; // ライト
};
```

ライティング - ofLight

▶ testApp.cpp

... (前略) ...

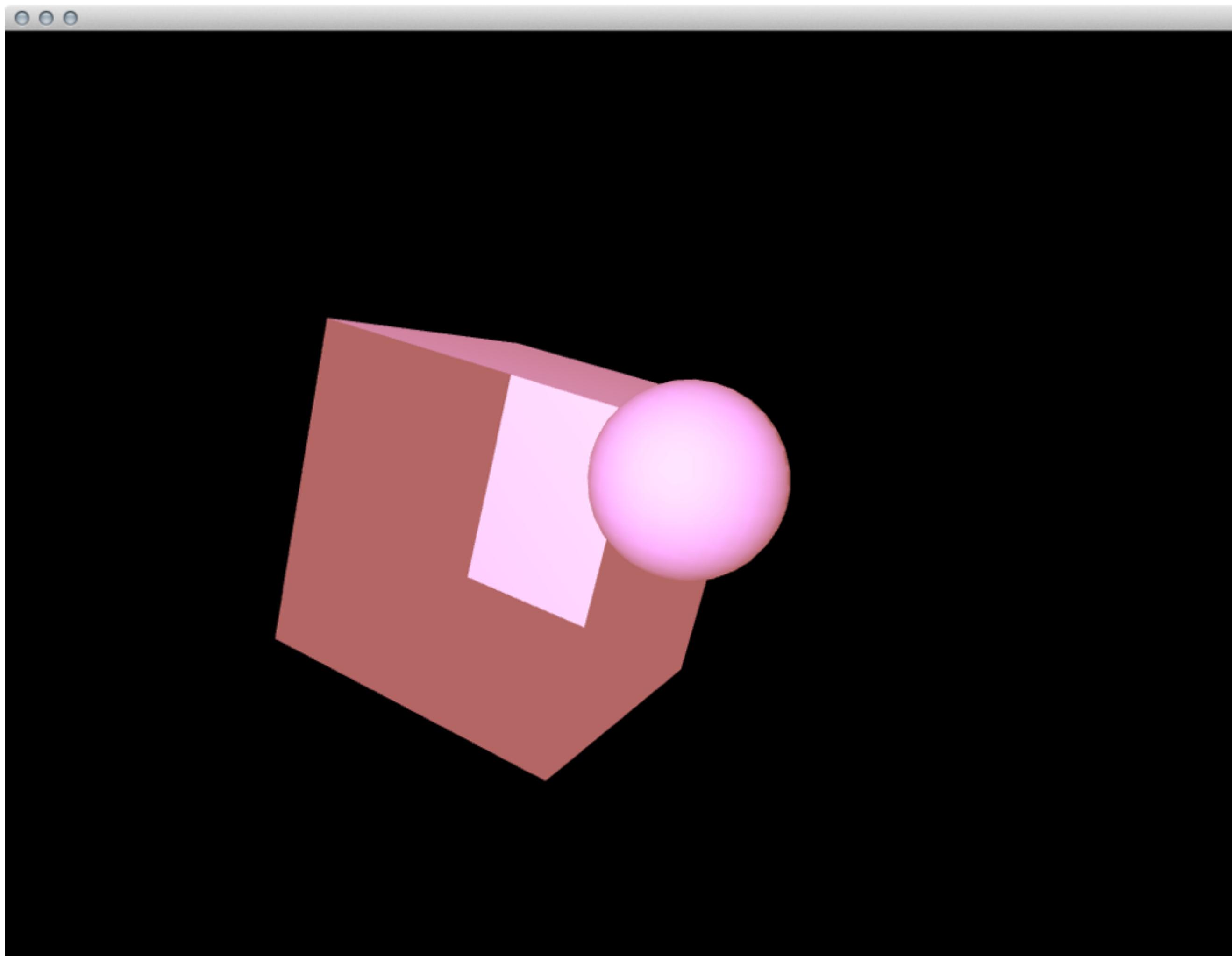
```
void testApp::setup(){
    ofBackground(0);

    // ライティングを有効に
    light.enable();
    // スポットライトを配置
    light.setSpotlight();
    // 照明の位置
    light.setPosition(-100, 100, 100);
    // 環境反射光の色
    light.setAmbientColor(ofFloatColor(0.5, 0.2, 0.2, 1.0));
    // 拡散反射光の色
    light.setDiffuseColor(ofFloatColor(0.5, 0.5, 1.0));
    // 鏡面反射光の色
    light.setSpecularColor(ofFloatColor(1.0, 1.0, 1.0));
}
```

... (後略) ...

ライティング - ofLight

- ▶ 立体のの前後の重なりが、おかしい!?



ライティング - ofLight

- ▶ openFrameworksでは、そのままの設定では後から実行した命令がどんどん上のレイヤーに描かれる
- ▶ 3Dの立体の場合、有り得ない重なりになることも
- ▶ DEPTH TEST (深度テスト) を有効にする必要があり
- ▶ ofEnableDepthTest() で有効になる

ライティング - ofLight

▶ testApp.cpp を修正

... (前略) ...

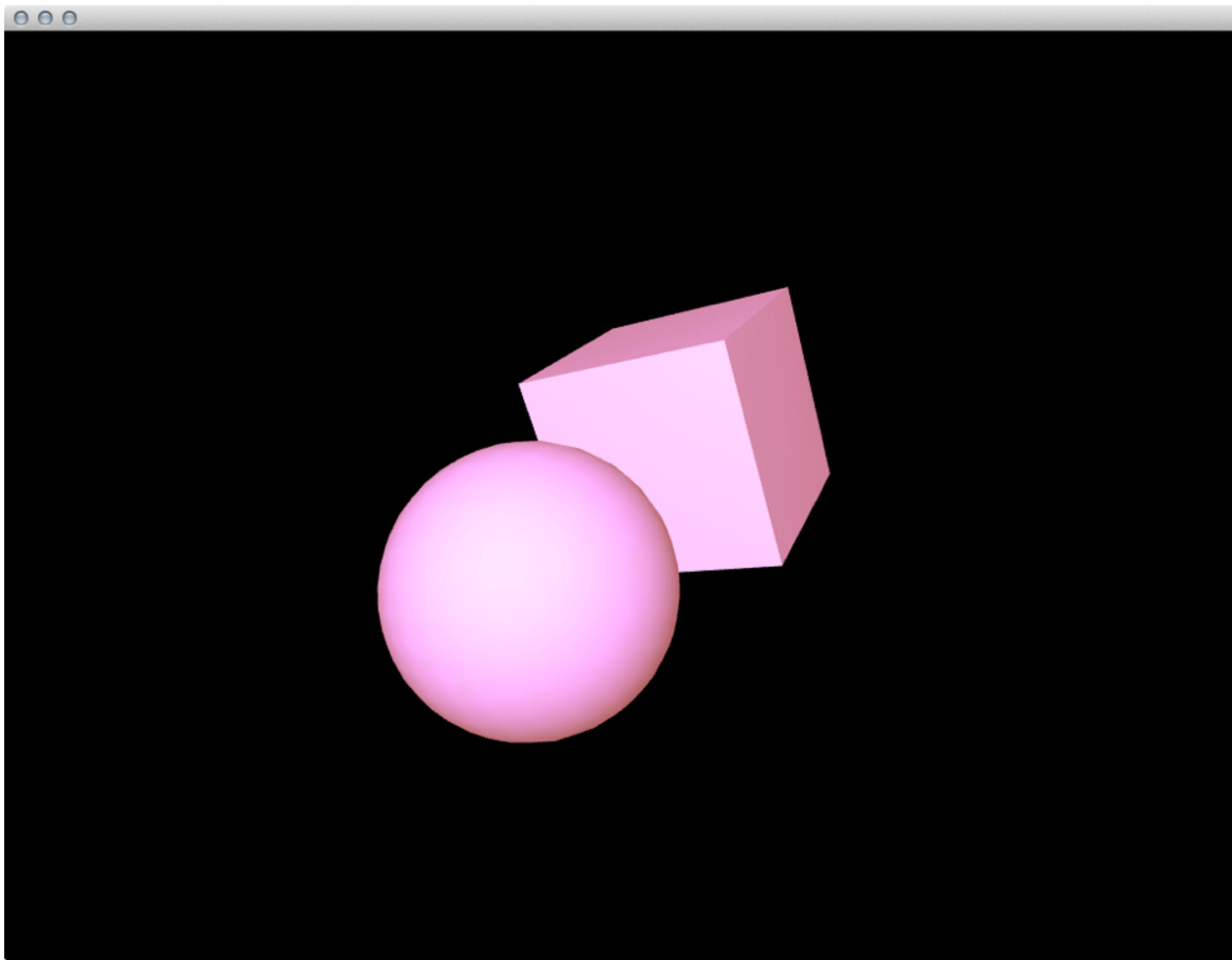
```
void testApp::setup(){
    ofBackground(0);
    ofEnableDepthTest(); // 深度テストを有効に ←ココ
    ofEnableSmoothing(); // 表示をスムースに

    // ライティングを有効に
    light.enable();
    // スポットライトを配置
    light.setSpotlight();
    // 照明の位置
    light.setPosition(-100, 100, 100);
    // 環境反射光の色
    light.setAmbientColor(ofFloatColor(0.5, 0.2, 0.2, 1.0));
    // 拡散反射光の色
    light.setDiffuseColor(ofFloatColor(0.5, 0.5, 1.0));
    // 鏡面反射光の色
    light.setSpecularColor(ofFloatColor(1.0, 1.0, 1.0));
}
```

... (後略) ...

ライティング - ofLight

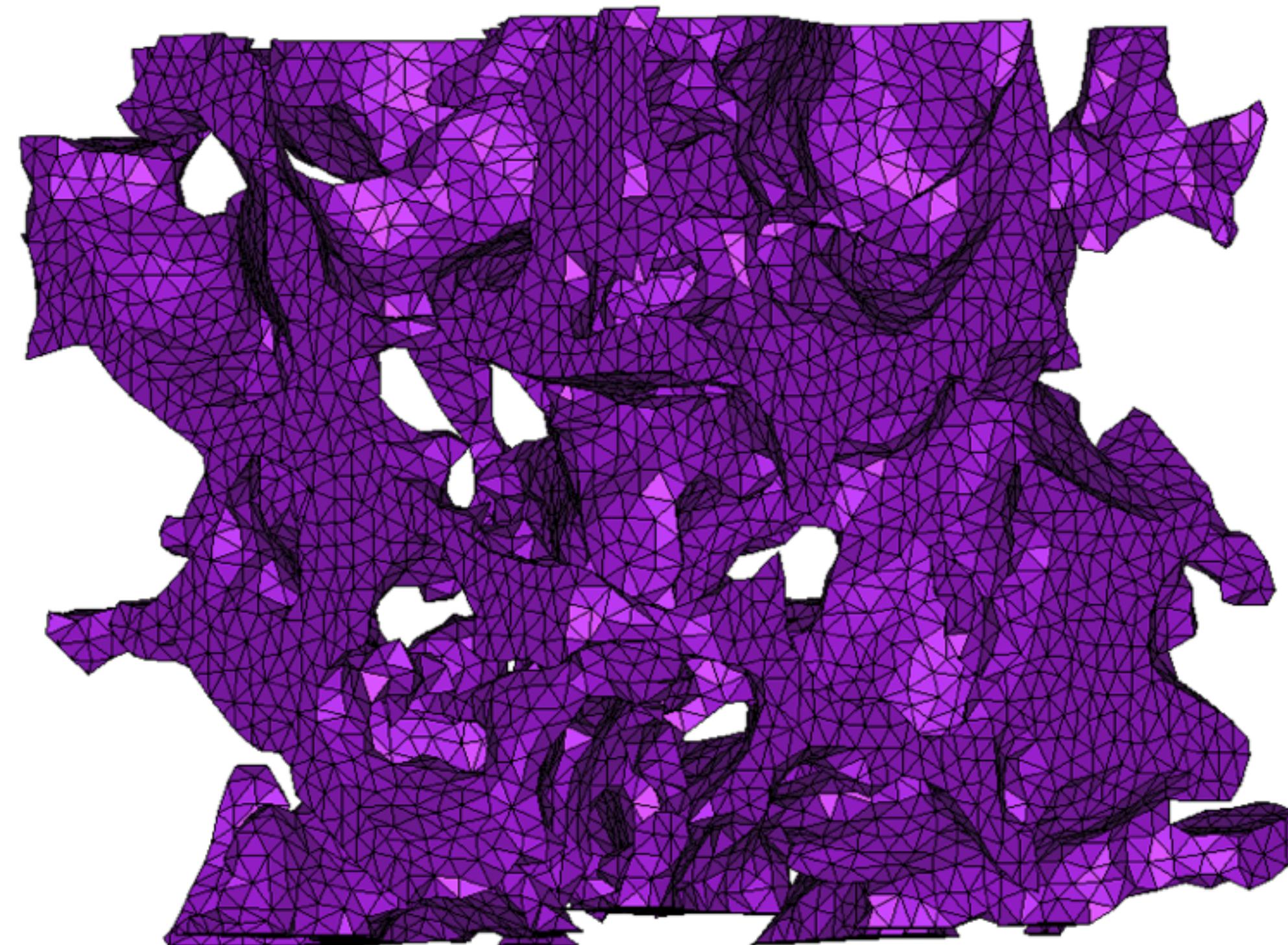
- ▶ 正しい重なりに



メッシュ (頂点情報の集合)
ofMesh

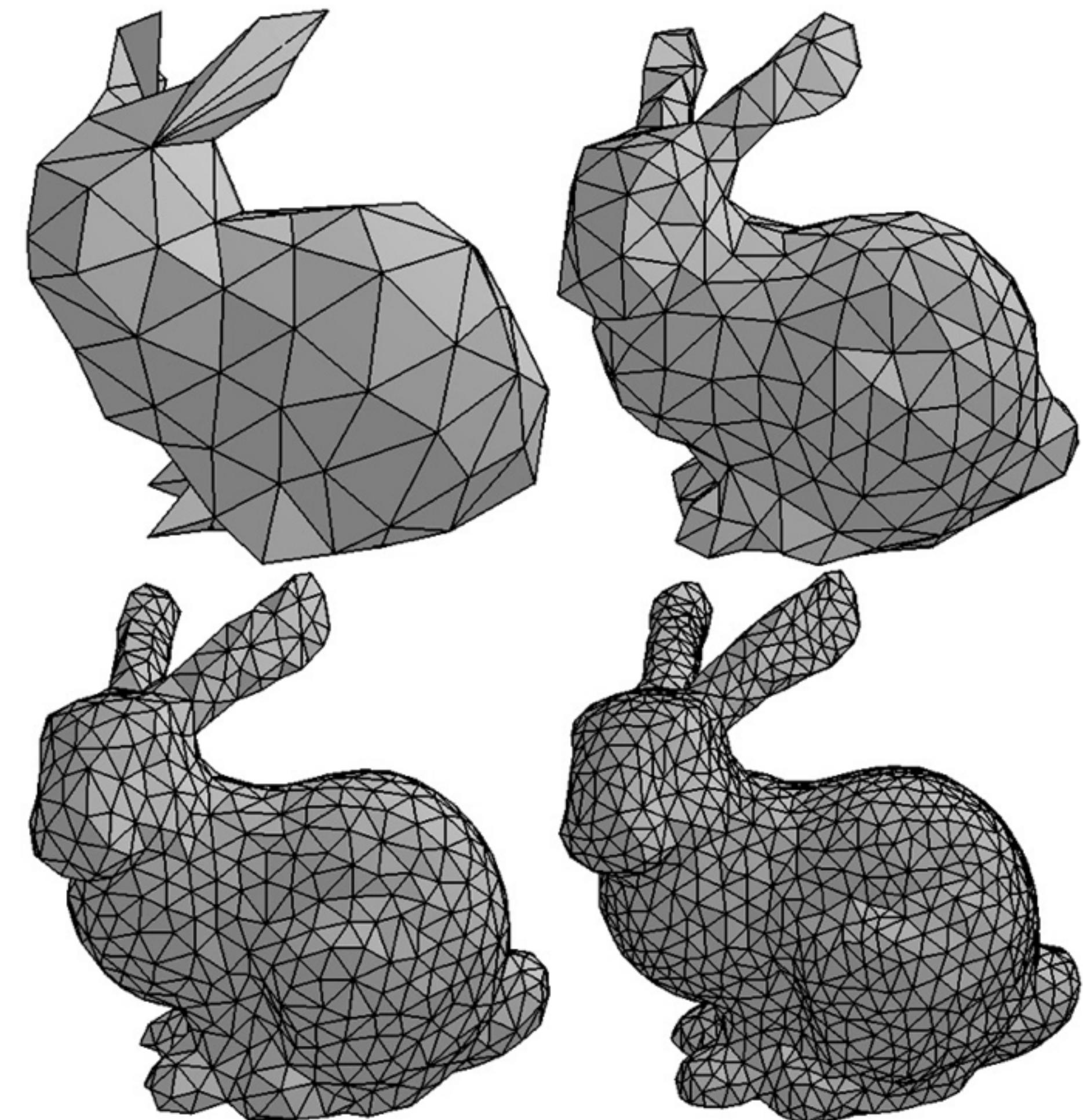
メッシュ (頂点情報の集合) - ofMesh

- › 立方体や球体などの単純な図形ではなく、より複雑な曲面や多面体などの3Dの形状を描くにはどうすれば良いのか?
- › ポリゴンメッシュ (Polygon Mesh) を使用する
- › openFrameworksでは、ofMeshクラスで生成できる



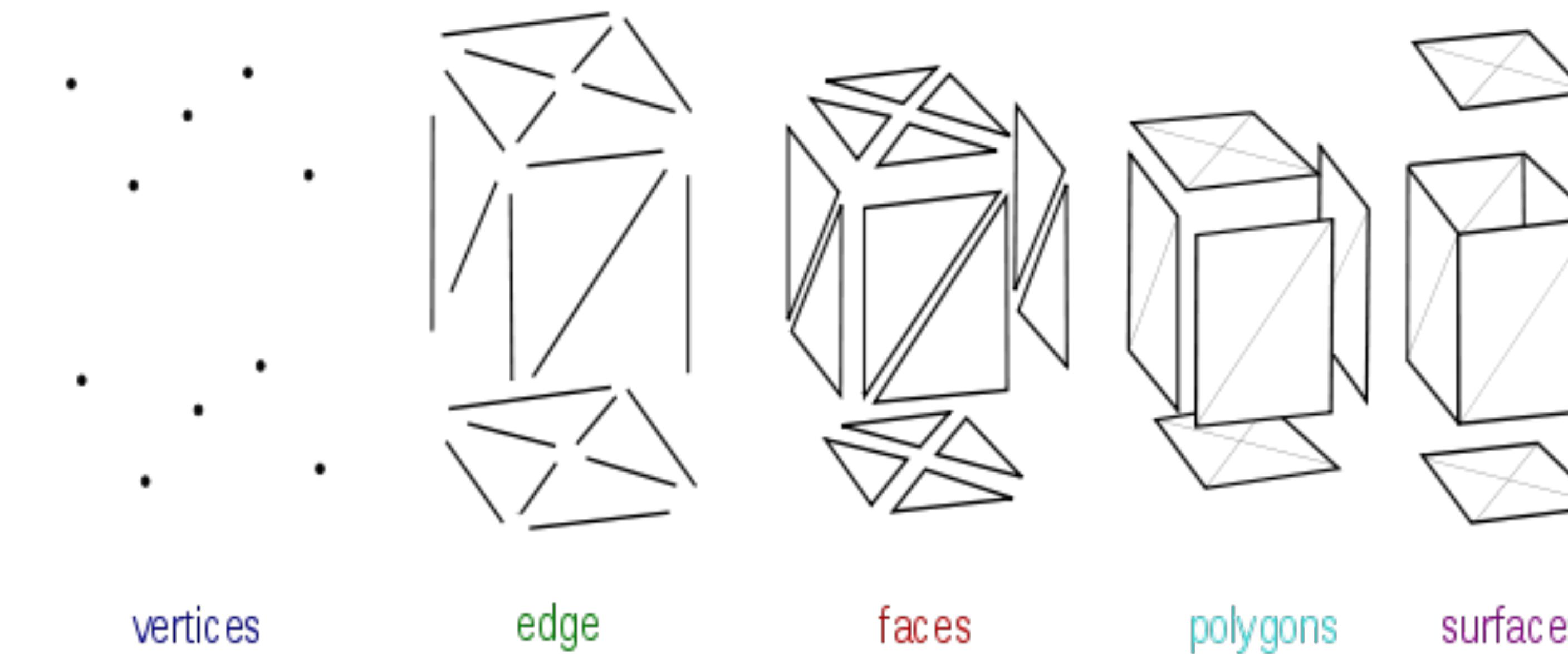
メッシュ (頂点情報の集合) - ofMesh

- ▶ メッシュによって構成された複雑な形状の例



メッシュ (頂点情報の集合) - ofMesh

- ▶ メッシュ (ポリゴンメッシュ Polygon Mesh)
- ▶ 3Dグラフィックのモデリングの多面体オブジェクトの形状を定義する頂点、辺、面の集合のこと
- ▶ 通常は三角形や四角形のポリゴン(面)によって構成される



メッシュ (頂点情報の集合) - ofMesh

- ▶ sin関数を利用して波を生成し、その3Dの形態をメッシュで表現してみる
- ▶ まず、x, y軸方向に均等に並ぶグリッドを作成
- ▶ グリッドの頂点をそれぞれ、x軸、y軸で異なる周波数のsin関数で上下に振動させる
- ▶ 頂点の座標を描画する

メッシュ (頂点情報の集合) - ofMesh

▶ testApp.h

```
#pragma once

#include "ofMain.h"

class testApp : 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);

    ofEasyCam cam; // カメラ
    ofVboMesh mesh; // 3Dメッシュ
    int w, h; // メッシュの幅と高さ
};
```

メッシュ (頂点情報の集合) - ofMesh

▶ testApp.cpp

```
#include "testApp.h"

void testApp::setup(){
    // 画面の設定
    ofBackground(0);
    ofEnableDepthTest();
    cam.setDistance(100);

    // メッシュの幅と高さ
    w = 200;
    h = 200;

    // 頂点の色を初期化
    for (int i = 0; i < w; i++) {
        for (int j = 0; j < h; j++) {
            mesh.addColor(ofFloatColor(0.5, 0.8, 1.0));
        }
    }
}
```

メッシュ (頂点情報の集合) - ofMesh

▶ testApp.cpp

```
void testApp::update(){
    // まず全ての頂点情報を削除
    mesh.clearVertices();

    // 全ての頂点の位置を更新して頂点情報として追加
    for (int i = 0; i < w; i++) {
        for (int j = 0; j < h; j++) {
            float x = sin(i * 0.1 + ofGetElapsedTimef())*10.0;
            float y = sin(j*0.15 + ofGetElapsedTimef()) * 10.0;
            float z = x + y;
            mesh.addVertex(ofVec3f(i - w/2, j - h/2, z));
        }
    }
}
```

メッシュ (頂点情報の集合) - ofMesh

▶ testApp.cpp

```
void testApp::draw(){
    // メッシュの描画
    ofSetHexColor(0xffffffff);
    cam.begin(); // カメラ開始

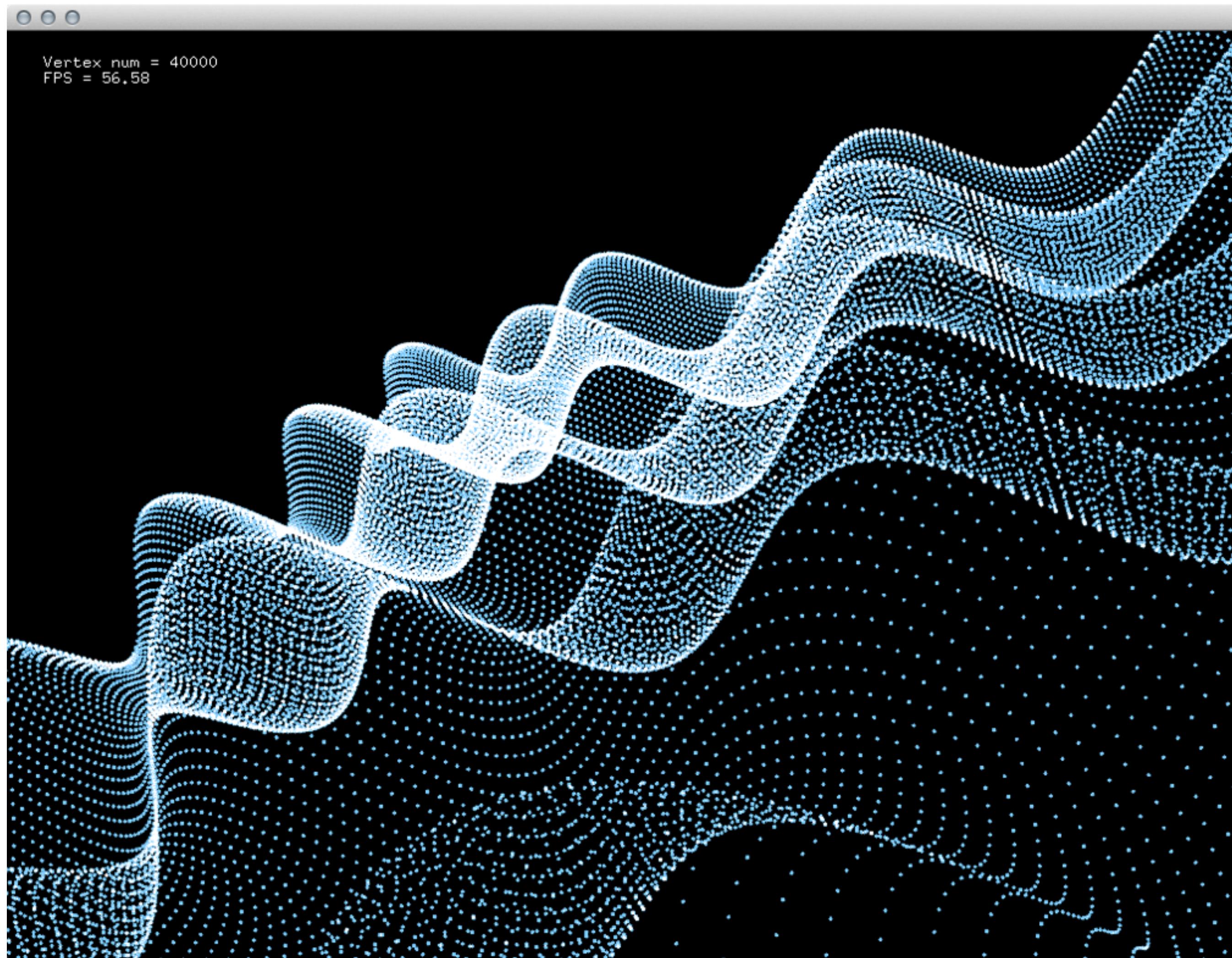
    // 頂点の位置をドットで表示
    glPointSize(2.0);
    glEnable(GL_POINT_SMOOTH);
    mesh.drawVertices();

    cam.end(); // カメラ終了

    // ログの表示
    string info;
    info = "Vertex num = " + ofToString(w * h, 0) + "\n";
    info += "FPS = " + ofToString(ofGetFrameRate(), 2);
    ofDrawBitmapString(info, 30, 30);
}
```

メッシュ (頂点情報の集合) - ofMesh

▶ 完成!!



Mesh 応用

カメラの明度でメッシュを生成

カメラの明度でメッシュを生成

- ▶ VBO + Mesh、応用
- ▶ カメラから映像を入力
- ▶ 映像のそれぞれのピクセルの明さを頂点の高さに
- ▶ ビデオからの色を、頂点の色に

VBO (頂点バッファオブジェクト) - ofVBO

▶ testApp.h

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

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

... (中略) ...

// クラス定数
static const int WIDTH = 640;
static const int HEIGHT = 480;
static const int NUM_PARTICLES = WIDTH * HEIGHT;

ofEasyCam cam; // カメラ
ofLight light; // ライト
ofVbo myVbo; // VBO
ofVec3f myVerts[NUM_PARTICLES]; // 頂点の座標
ofFloatColor myColor[NUM_PARTICLES]; // 頂点の色情報
ofVideoGrabber myVideo; // ビデオキャプチャ
};
```

VBO (頂点バッファオブジェクト) - ofVBO

▶ testApp.cpp

```
#include "testApp.h"

const int testApp::WIDTH;
const int testApp::HEIGHT;
const int testApp::NUM_PARTICLES;

void testApp::setup(){
    // 画面の設定
    ofBackground(0);
    ofEnableDepthTest();
    ofEnableBlendMode(OF_BLENDMODE_ADD);
    cam.setDistance(400);
    myVideo.initGrabber(640, 480); // カメラ映像をキャプチャ
    // 頂点情報を初期化
    for (int i = 0; i < WIDTH; i++) {
        for (int j = 0; j < HEIGHT; j++) {
            myVerts[j * WIDTH + i].set(i - WIDTH/2, j - HEIGHT/2, 0);
            myColor[j * WIDTH + i].set(1.0, 1.0, 1.0, 1.0);
        }
    }
    // 頂点バッファに位置と色の情報を設定
    myVbo.setVertexData(myVerts, NUM_PARTICLES, GL_DYNAMIC_DRAW);
    myVbo.setColorData(myColor, NUM_PARTICLES, GL_DYNAMIC_DRAW);
}
```

VBO (頂点バッファオブジェクト) - ofVBO

▶ testApp.cpp

```
void testApp::update(){
    // カメラからの映像を更新
    myVideo.update();
    // もしカメラのフレームが更新されていたら
    if (myVideo.isFrameNew()) {
        // カメラの映像のピクセル情報を抽出
        unsigned char * pixels = myVideo.getPixels();
        // ピクセルごとに処理
        for (int i = 0; i < WIDTH; i++) {
            for (int j = 0; j < HEIGHT; j++) {
                // ピクセルのRGB値を取得
                float r = (float)pixels[j * myVideo.width * 3 + i * 3] / 256.0;
                float g = (float)pixels[j * myVideo.width * 3 + i * 3 + 1] / 256.0;
                float b = (float)pixels[j * myVideo.width * 3 + i * 3 + 2] / 256.0;
                // RGBから明度を算出
                float brightness = (r + g + b) / 3.0f;
                // 明度から頂点の位置を設定
                myVerts[j * WIDTH + i] = ofVec3f(i - WIDTH/2, j - HEIGHT/2, brightness * 256.0);
                // 頂点の色はカメラのピクセルの値をそのまま使用
                myColor[j * WIDTH + i] = ofFloatColor(r, g, b, 0.8);
            }
        }
        // VBOの座標と色の情報を更新
        myVbo.updateVertexData(myVerts, NUM_PARTICLES);
        myVbo.updateColorData(myColor, NUM_PARTICLES);
    }
}
```

VBO (頂点バッファオブジェクト) - ofVBO

▶ testApp.cpp

```
void testApp::draw(){
    // VBOを描画
    cam.begin();
    ofScale(1, -1, 1);
    glPointSize(3);
    myVbo.draw(GL_POINTS, 0, NUM_PARTICLES);
    cam.end();

    // ログの表示
    string info;
    info = "Vertex num = " + ofToString(NUM_PARTICLES, 0) + "\n";
    info += "FPS = " + ofToString(ofGetFrameRate(), 2);
    ofDrawBitmapString(info, 30, 30);
}
```

カメラの明度でメッシュを生成

▶ 完成!!

