

IDD Media lab. 2015

openFrameworks

Shaderをつかった、GPUプログラミング

2015年6月23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

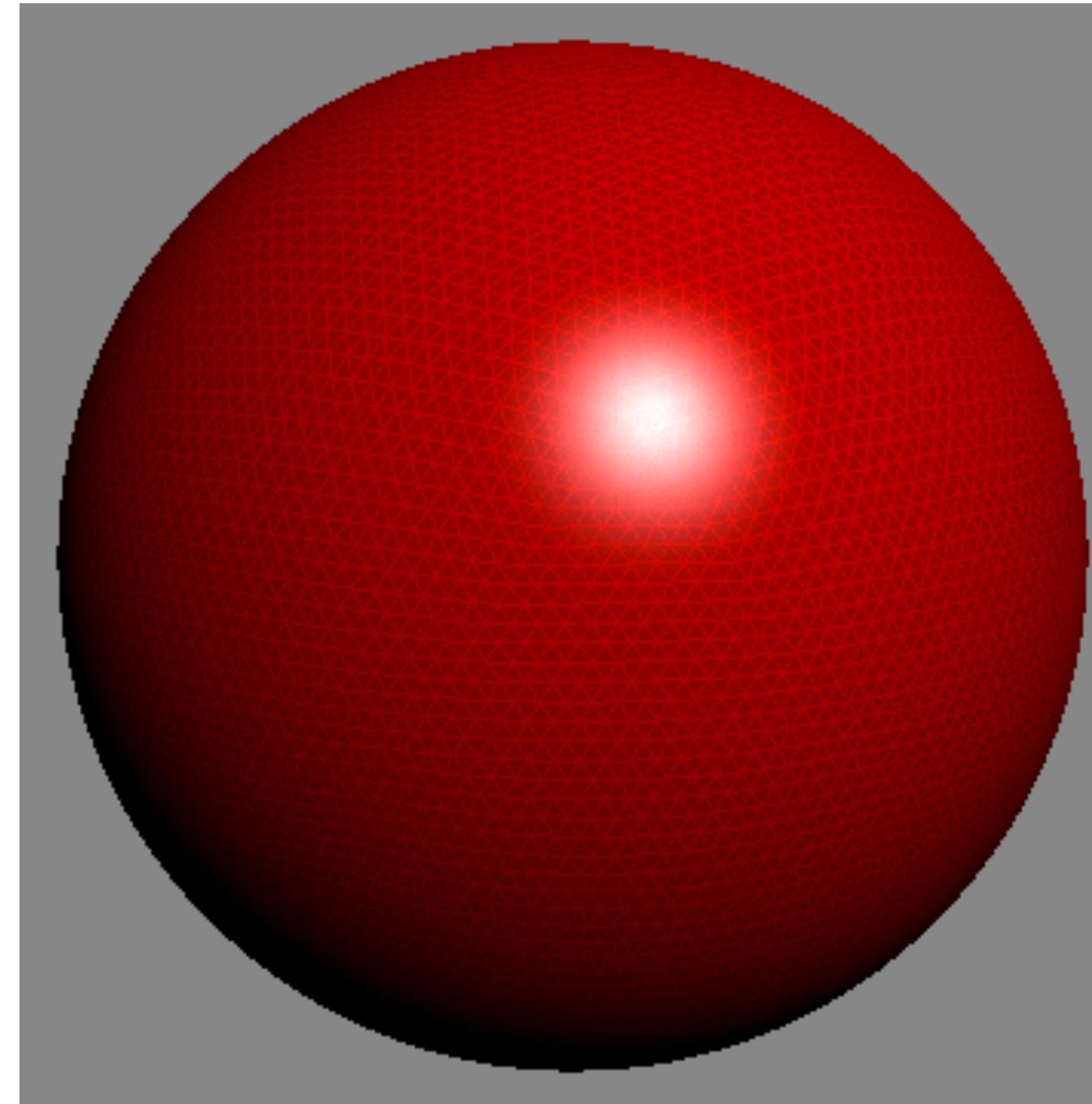
今日の内容

- ▶ Shader (GLSL) をつかってみる!
- ▶ Shaderって何?
- ▶ Fragment Shader をつかった表現に挑戦

Shaderとは何か?

Shaderとは何か?

- ▶ Shaderとは
- ▶ もともとは、3DCGにおいて、シェーディング（陰影処理）を行うコンピュータプログラムのこと



Shaderとは何か?

- ▶ 従来は、開発者やデザイナーは、グラフィックスカード (GPU) に固定機能として実装された定形の処理しか使えなかった (固定機能シェーダー)
- ▶ 2000年代に入って: プログラマブル・シェーダーの登場
- ▶ ブラックボックスだったシェーダー自身が、プログラム可能になった
- ▶ OpenGL → GLSL
- ▶ Direct 3D → HLSL

Shaderとは何か?

- ▶ openFrameworksでは、描画にOpenGLを使用
- ▶ GLSLを使用可能
- ▶ ofShader というクラスが提供されている
- ▶ <http://openframeworks.cc/documentation/gl/ofShader.html>

Shaderとは何か?

- ▶ Shaderの種類
- ▶ 頂点シェーダー (Vertex Shader)
- ▶ 入力された頂点を座標変換するための機能
- ▶ ジオメトリシェーダー (Geometry Shader)
- ▶ オブジェクト内の頂点の集合を加工して、新しいプリミティブ図形を生成できる
- ▶ フラグメントシェーダー (Fragment Shader)
- ▶ ピクセルを操作する。画面上の膨大なピクセル情報を、高い並列処理性能を持つGPUで実行することにより、CPUで実行するよりもはるかに高いパフォーマンスを実現

Shaderとは何か?

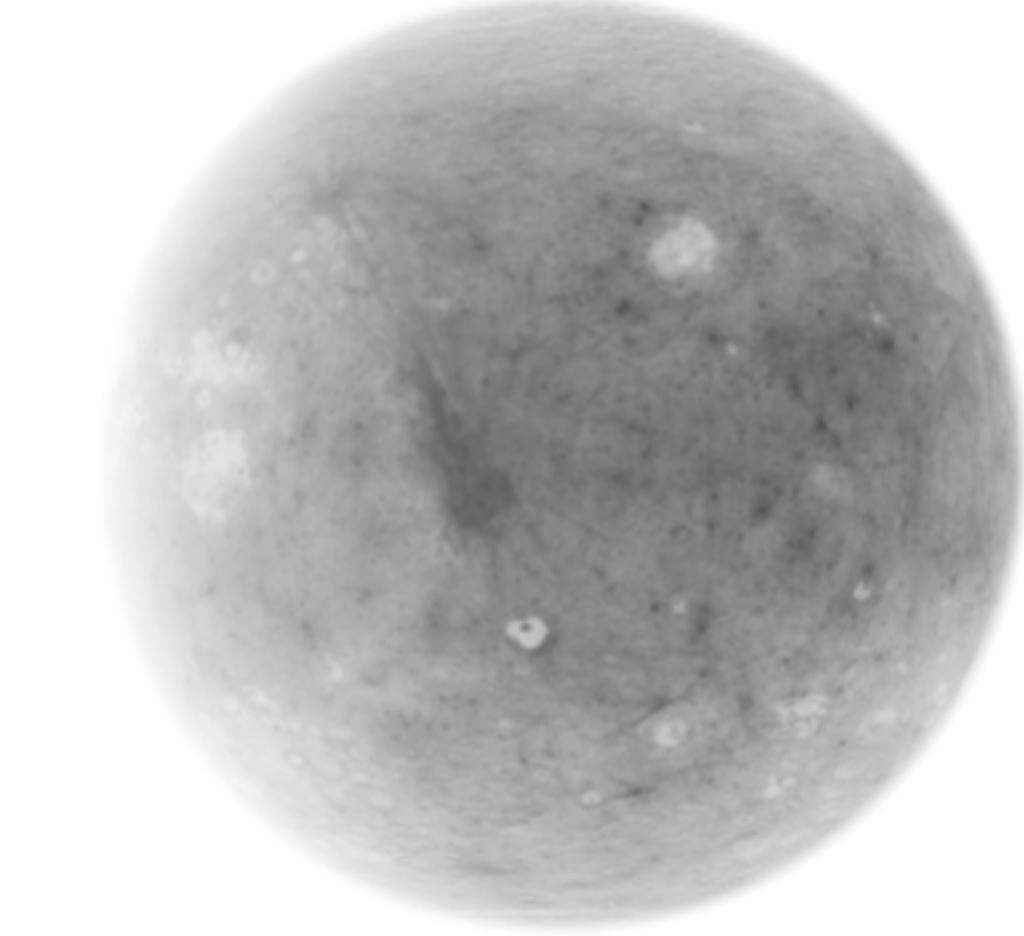
- ▶ Shaderの種類
- ▶ 頂点シェーダー (Vertex Shader)
- ▶ 入力された頂点を座標変換するための機能
- ▶ ジオメトリシェーダー (Geometry Shader)
- ▶ オブジェクト内の頂点の集合を加工して、新しいプリミティブ図形を生成できる

- ▶ フラグメントシェーダー (Fragment Shader)
 - ▶ ピクセルを操作する。画面上の膨大なピクセル情報を、高い並列処理性能を持つGPUで実行することにより、CPUで実行するよりもはるかに高いパフォーマンスを実現

今回は、フラグメントシェーダーに挑戦!

The Book of Shaders

- ▶ 現在、すばらしいオンライン教材の執筆が進行中!
- ▶ The Book of Shaders - Patricio Gonzalez Vivo
- ▶ <http://patriciogonzalezvivo.com/2015/thebookofshaders/>



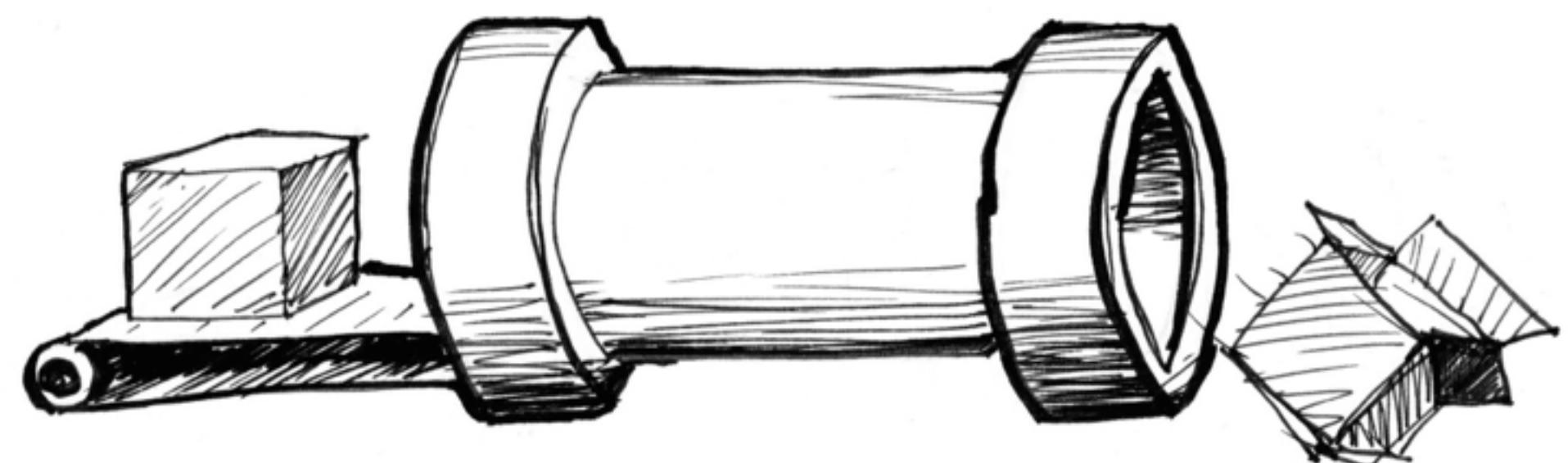
The Book of Shaders

by Patricio Gonzalez Vivo

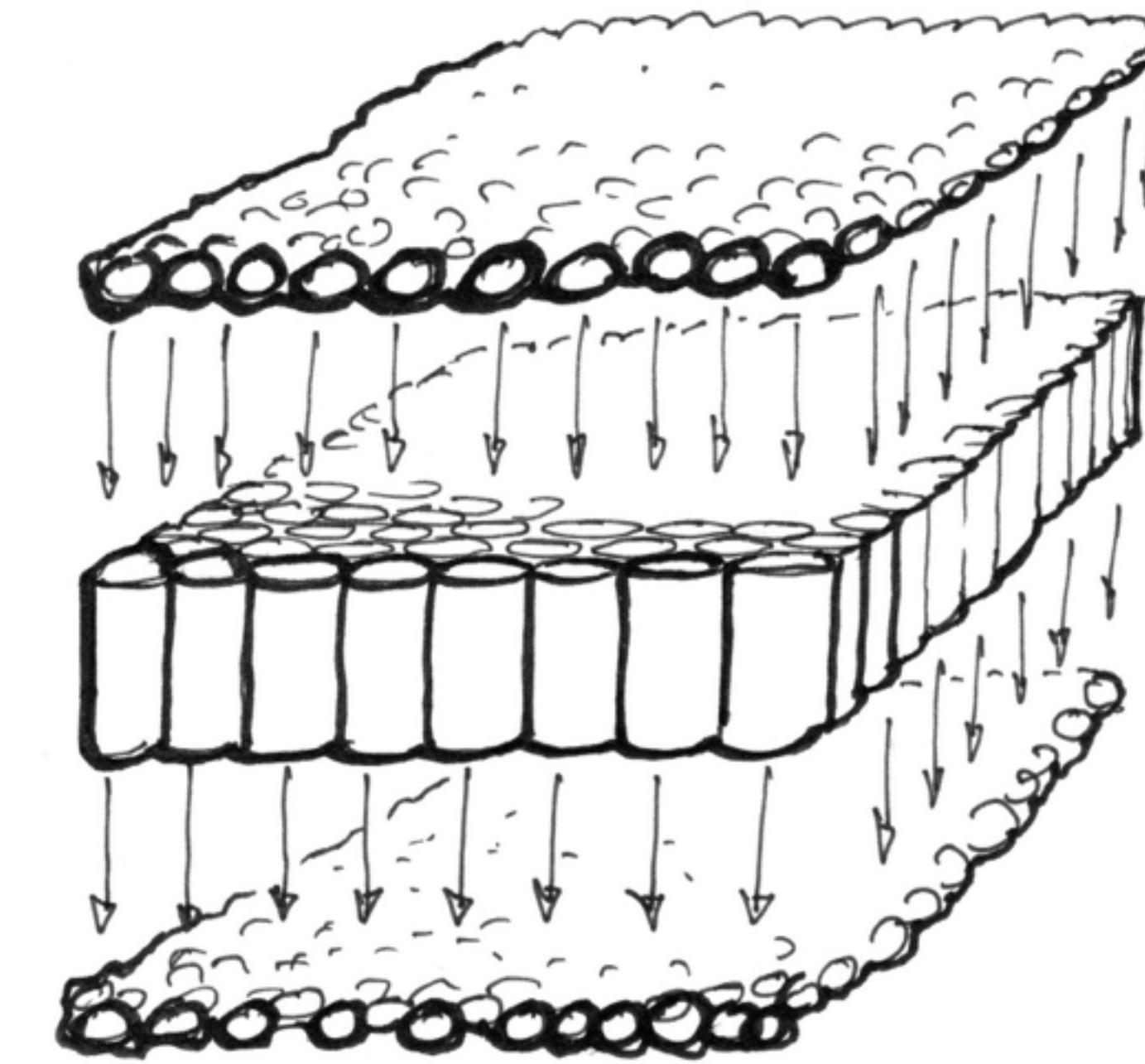
This is a gentle step-by-step guide through the abstract and complex universe of Fragment Shaders.

The Book of Shaders

- ▶ CPUとGPUの違いの説明など、とてもわかりやすい!!



CPU



GPU

openFrameworksでShaderを動かす

openFrameworksでShaderを動かす

- ▶ openFrameworksでShaderを動かすには?
- ▶ ofShaderを使用する
- ▶ まずは、openFrameworks側からプログラムの準備

openFrameworksでShaderを動かす

▶ ofApp.h

```
#pragma once

#include "ofMain.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);

    ofShader shader;
};

};
```

openFrameworksでShaderを動かす

▶ ofApp.cpp

```
#include "ofApp.h"

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

void ofApp::update(){

}

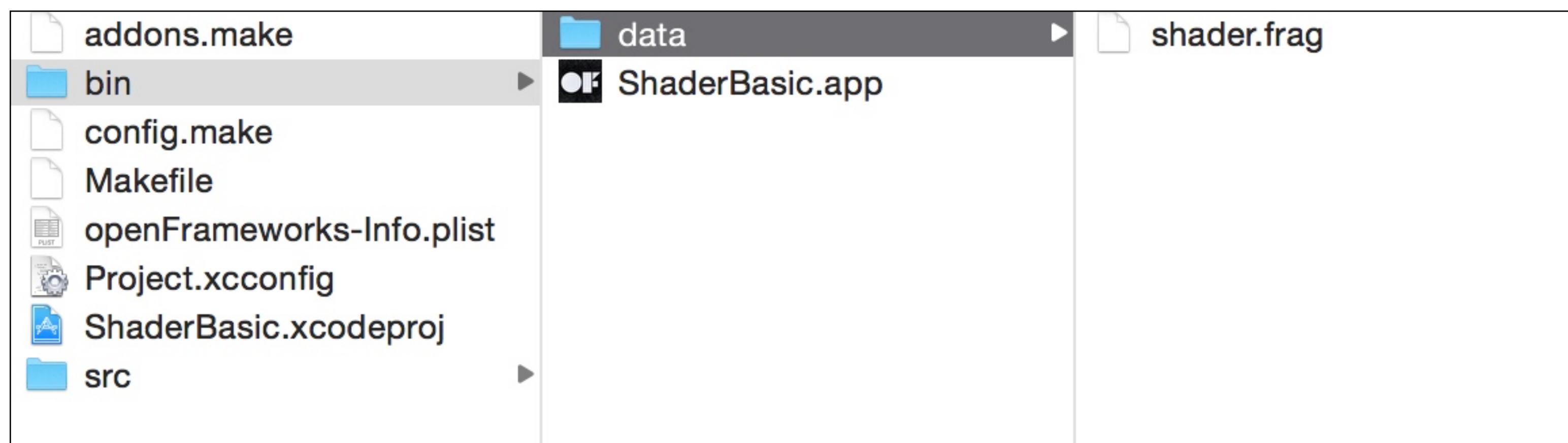
void ofApp::draw(){
    shader.load("", "shader.frag");

    shader.begin();
    shader.setUniform1f("u_time", ofGetElapsedTimef());
    shader.setUniform2f("u_resolution", ofGetWidth(), ofGetHeight());
    ofRect(0,0,ofGetWidth(), ofGetHeight());
    shader.end();
}

... (後略) ...
```

openFrameworksでShaderを動かす

- ▶ 次に Fragment Shader を記述
- ▶ ofShaderで読み込まれるShaderファイルは、bin/data/ 以下に格納する
- ▶ 例えば、ファイル名が「shader.frag」とすると…



openFrameworksでShaderを動かす

- ▶ まず始めに、シンプルなFragment Shaderを書いてみる
- ▶ Shaderの“Hello World”

openFrameworksでShaderを動かす

▶ shader.frag

```
void main() {  
    gl_FragColor = vec4(1.0,0.0,1.0,1.0);  
}
```

openFrameworksでShaderを動かす

- ▶ 画面全体に色が塗られる



openFrameworksでShaderを動かす

- ▶ プログラムを読み解いてみる
- ▶ GLSLのプログラムは、まず始めに main関数が実行される
- ▶ main関数で演算した最終的なピクセルの色の値は、 gl_FragColor に出力する
- ▶ GLSLのプログラムの中で「gl_」から始まる変数は、全てグローバル変数 (始めから定義されている)
- ▶ 変数や関数の書き方は、ほぼ、C / C++ の書き方を踏襲している
- ▶ int, float, bool などはC++と同じように使用可能
- ▶ vec2, vec3, vec4 など、 GLSL独自の型も存在している

openFrameworksでShaderを動かす

- ▶ 注意！- Float型で指定すべき数値は、小数点以下の数もきちんと記述しないとNG

```
// 正しく動く
void main() {
    gl_FragColor = vec4(1.0,0.0,1.0,1.0);
}
```

```
// エラーになる
void main() {
    gl_FragColor = vec4(1, 0, 1, 1);
}
```

openFrameworksでShaderを動かす

- ▶ 関数を自由に追加して、構造化していくことが可能

```
vec4 red() {  
    return vec4(1.0,0.0,0.0,1.0);  
}  
  
void main() {  
    gl_FragColor = red();  
}
```

Uniforms - openFrameworksからShaderへ値を送出

Uniforms - openFrameworksからShaderへ値を送出

- ▶ openFrameworksと、GLSLの情報のやりとりのイメージ



Uniforms - openFrameworksからShaderへ値を送出

- ▶ ofShaderで送出可能な、Uniformの例

関数名	送出内容
setUniform1f	1つのfloatデータ
setUniform1fv	floatデータの配列
setUniform2f	2次元のfloatデータ
setUniform2fv	2次元のfloatデータの配列
setUniform3f	3次元のfloatデータ
setUniform3fv	3次元のfloatデータの配列
setUniform4f	4次元のfloatデータ
setUniform4fv	4次元のfloatデータの配列
setUniformTexture	テクスチャーデータ

openFrameworksでShaderを動かす

- ▶ 実は、既に2種類のデータ送出していた
- ▶ 経過時間 ofGetElapsedTimef() - Uniform1f
- ▶ 画面の解像度 - Uniform2f

```
#include "ofApp.h"

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

void ofApp::update(){

}

void ofApp::draw(){
    shader.load("", "shader.frag");

    shader.begin();
    shader.setUniform1f("u_time", ofGetElapsedTimef());
    shader.setUniform2f("u_resolution", ofGetWidth(), ofGetHeight());
    ofRect(0,0,ofGetWidth(), ofGetHeight());
    shader.end();
}

... (後略) ...
```

Uniforms - openFrameworksからShaderへ値を送出

- ▶ 送出したUniformsをShaderで活用してみる!
 - ▶ 経過時間
 - ▶ 画面解像度

Uniforms - openFrameworksからShaderへ値を送出

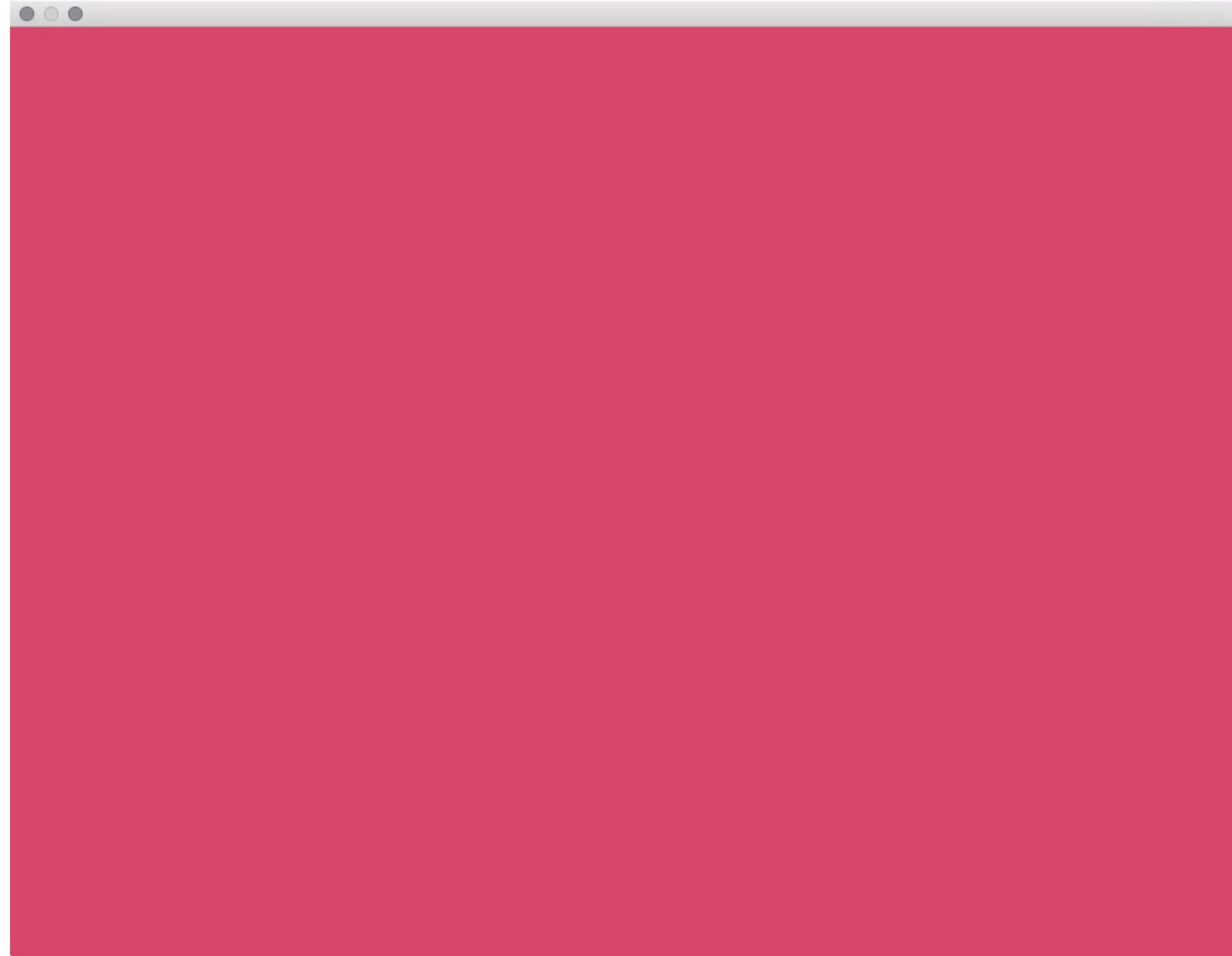
- ▶ shader.frag - 経過時間で色を変化させてみる

```
uniform vec2 u_resolution; // 画面の解像度(width,height)
uniform float u_time;      // 起動してからの経過時間(秒)

void main() {
    float red = abs(sin(u_time * 3.0));
    float green = abs(sin(u_time * 4.0));
    float blue = abs(sin(u_time * 5.0));
    gl_FragColor = vec4(red, green, blue, 1.0);
}
```

Uniforms - openFrameworksからShaderへ値を送出

- ▶ 色がなめらかに変化していく



Uniforms - openFrameworksからShaderへ値を送出

- ▶ shader.frag - 周波数を速くすると、激しいフラッシュに (取扱い注意!!)

```
uniform vec2 u_resolution; // 画面の解像度(width,height)
uniform float u_time;      // 起動してからの経過時間(秒)

void main() {
    float red = abs(sin(u_time * 30.0));
    float green = abs(sin(u_time * 40.0));
    float blue = abs(sin(u_time * 50.0));
    gl_FragColor = vec4(red, green, blue, 1.0);
}
```

Uniforms - openFrameworksからShaderへ値を送出

- ▶ 次に、画像解像度を利用してみる
- ▶ (x, y) を、それぞれ $0.0 \sim 1.0$ の値にマッピングして活用すると便利

Uniforms - openFrameworksからShaderへ値を送出

- ▶ shader.frag - 画像解像度を利用して、グラデーション

```
uniform vec2 u_resolution; // 画面の解像度(width,height)
uniform float u_time;      // 起動してからの経過時間(秒)

void main() {
    // 画面の解像度から、0.0～1.0に正規化する
    vec2 st = gl_FragCoord.xy/u_resolution;
    gl_FragColor = vec4(st.x,st.y,0.0,1.0);
}
```

Uniforms - openFrameworksからShaderへ値を送出

- ▶ Shaderによるグラデーション



Uniforms - openFrameworksからShaderへ値を送出

- ▶ さらに、グラデーションを時間で変化させてみる
- ▶ 2つのUniformsのくみあわせ

Uniforms - openFrameworksからShaderへ値を送出

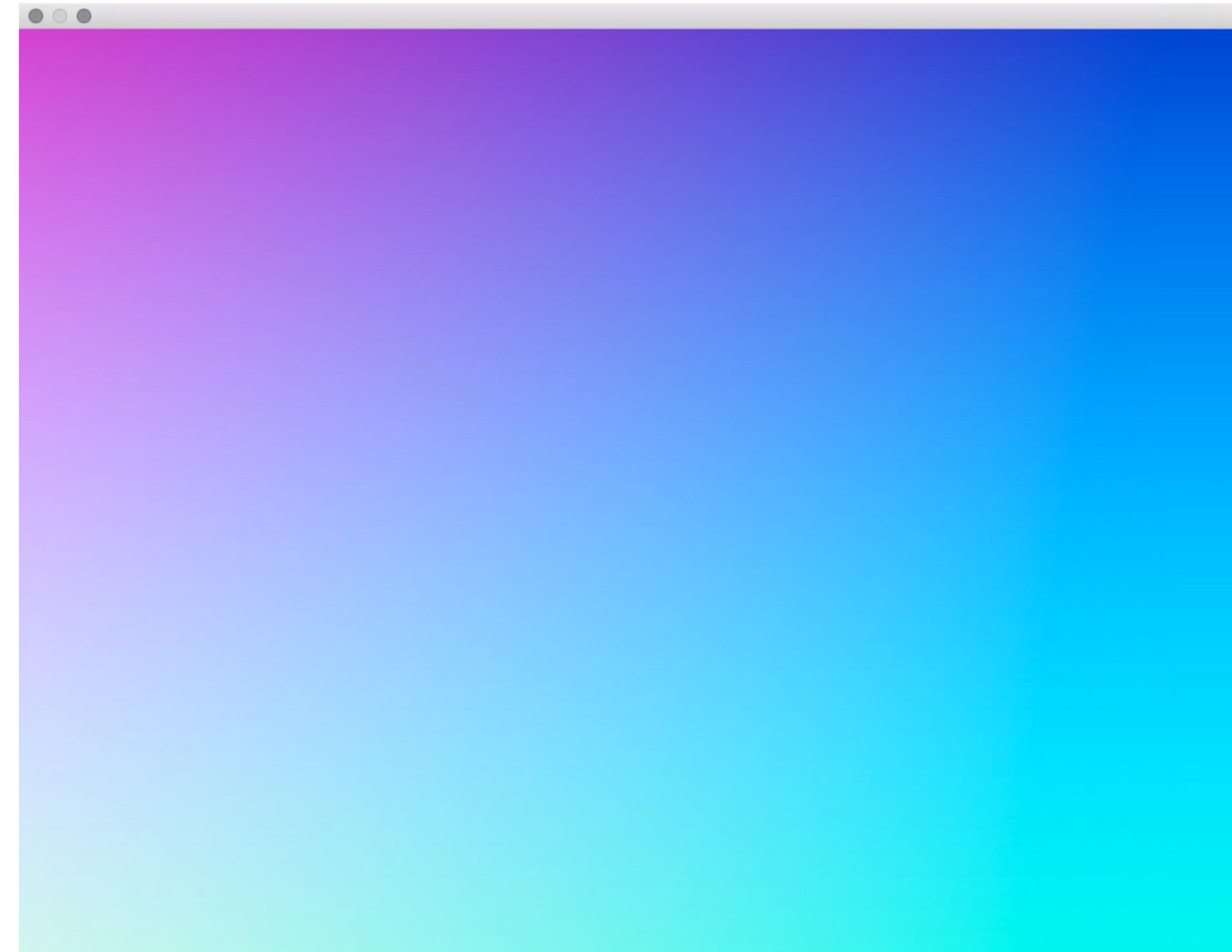
- ▶ shader.frag - 画像解像度を利用して、グラデーション

```
uniform vec2 u_resolution; // 画面の解像度(width,height)
uniform float u_time;      // 起動してからの経過時間(秒)

void main() {
    //画面の解像度から、0.0～1.0に正規化する
    vec2 st = gl_FragCoord.xy / u_resolution;
    // 色を格納する変数
    vec3 color;
    // RGBそれぞれの時間で変化するグラデーションを生成
    color.r = abs(sin(u_time * 1.5 + st.x));
    color.g = abs(sin(u_time * 2.0 + st.y));
    color.b = abs(sin(u_time * 3.0 + st.y));
    // 合成して出力
    gl_FragColor = vec4(color.r, color.g, color.b, 1.0);
}
```

Uniforms - openFrameworksからShaderへ値を送出

- ▶ 時間によって変化していくグラデーション



GLSLで形を描く

GLSLで形を描く

- ▶ もう少し複雑な図形に挑戦
- ▶ 画面に波を描いてみる

Uniforms - openFrameworksからShaderへ値を送出

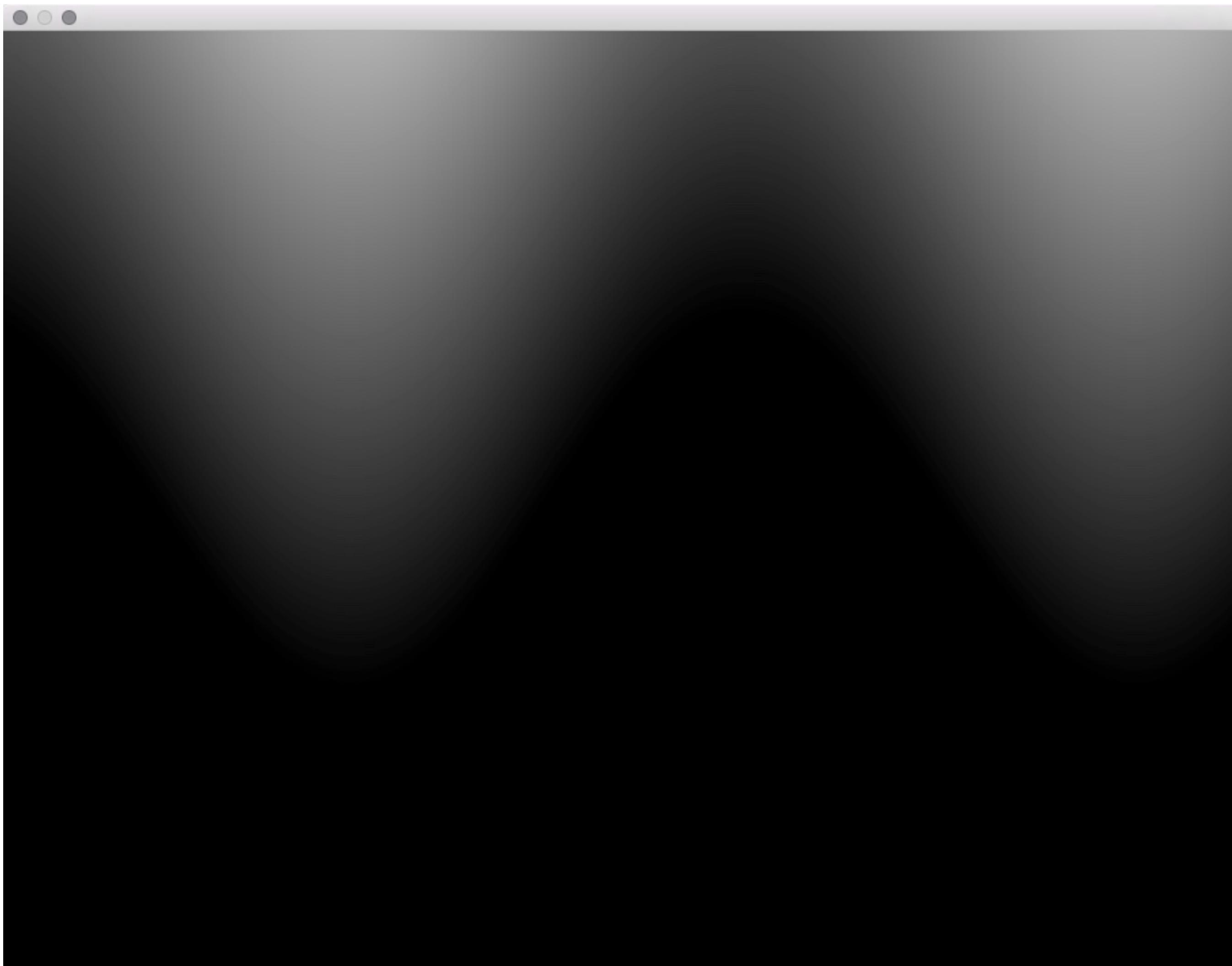
▶ 画面に波を描いてみる

```
uniform vec2 u_resolution; // 画面の解像度(width,height)
uniform float u_time; // 起動してからの経過時間(秒)

void main() {
    //画面の解像度から、0.0～1.0に正規化する
    vec2 st = gl_FragCoord.xy / u_resolution;
    // 画面の中央をy軸の中心に
    st.y -= 0.5;
    // Sin波で濃淡をつける
    st.y += sin(st.x * 10.0 + u_time) * 0.2;
    vec3 color = vec3(st.y);
    gl_FragColor = vec4(color, 1.0);
}
```

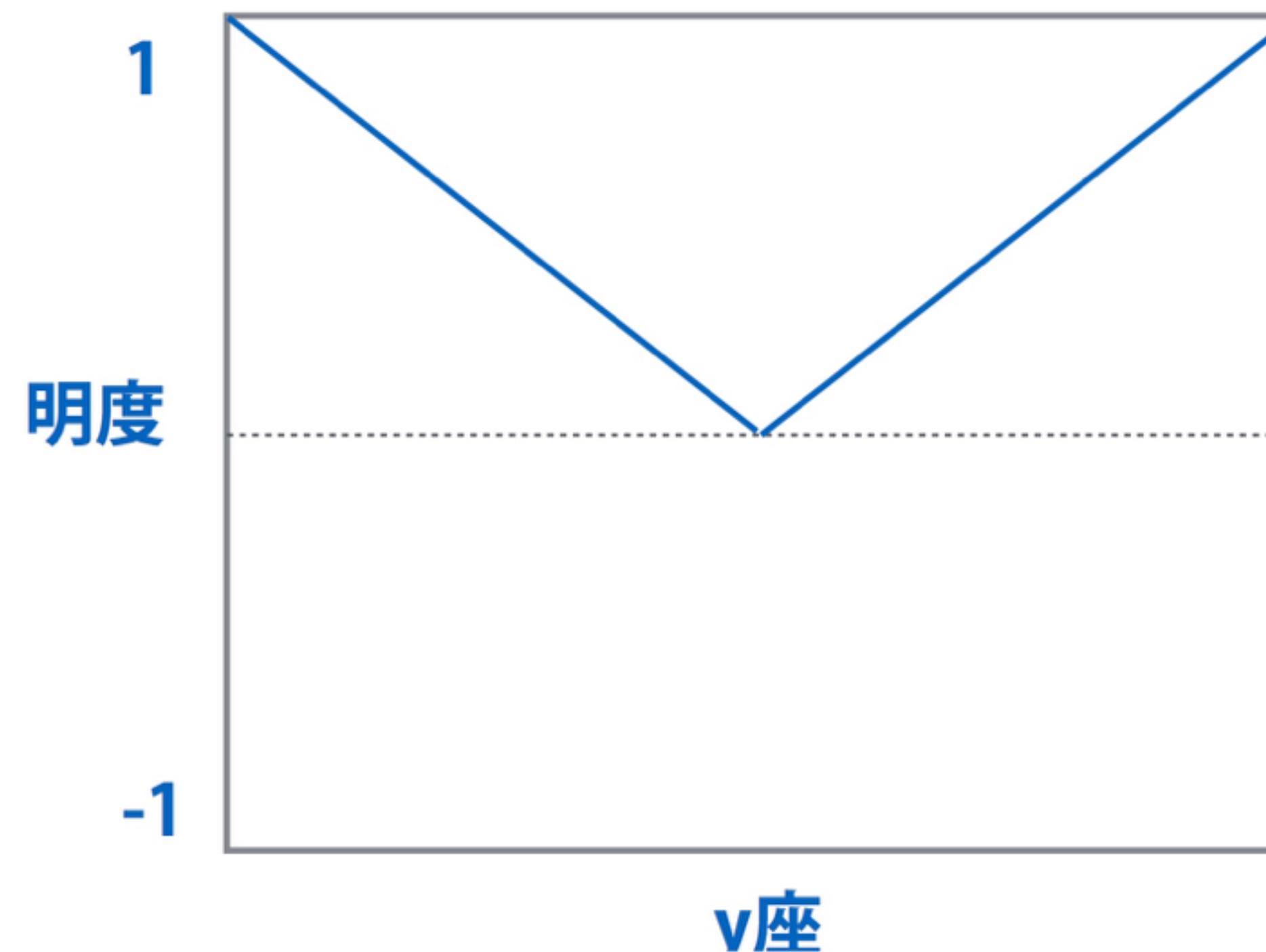
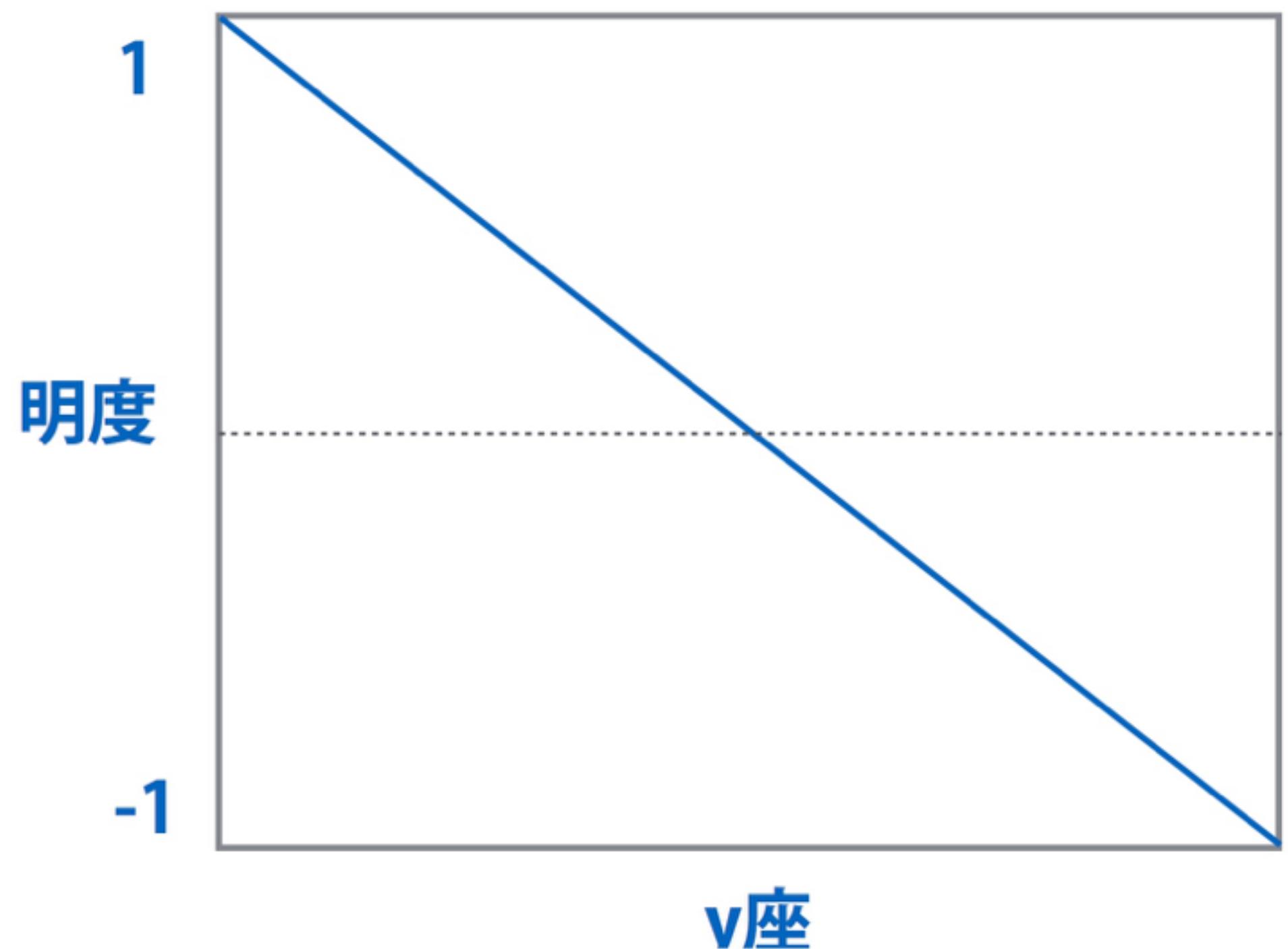
GLSLで形を描く

- ▶ 濃淡による波が描けた



GLSLで形を描く

- ▶ 明度の絶対値を表示してみる



Uniforms - openFrameworksからShaderへ値を送出

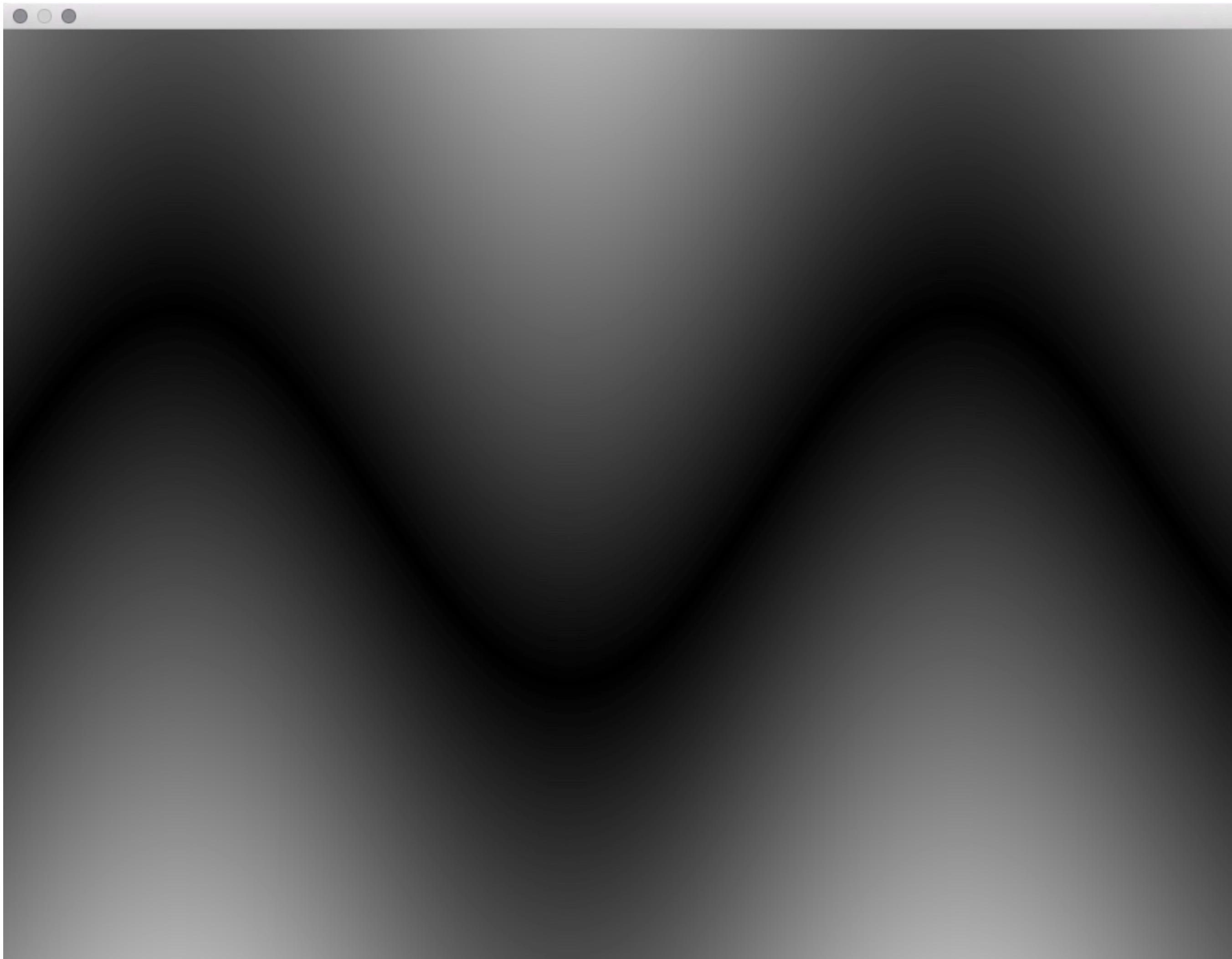
- ▶ 明度の絶対値を表示してみる

```
uniform vec2 u_resolution; // 画面の解像度(width,height)
uniform float u_time;      // 起動してからの経過時間(秒)

void main() {
    //画面の解像度から、0.0～1.0に正規化する
    vec2 st = gl_FragCoord.xy / u_resolution;
    // 画面の中央をy軸の中心に
    st.y -= 0.5;
    // Sin波で濃淡をつける
    st.y += sin(st.x * 10.0 + u_time) * 0.2;
    // 濃度の絶対値
    vec3 color = vec3(abs(st.y));
    gl_FragColor = vec4(color, 1.0);
}
```

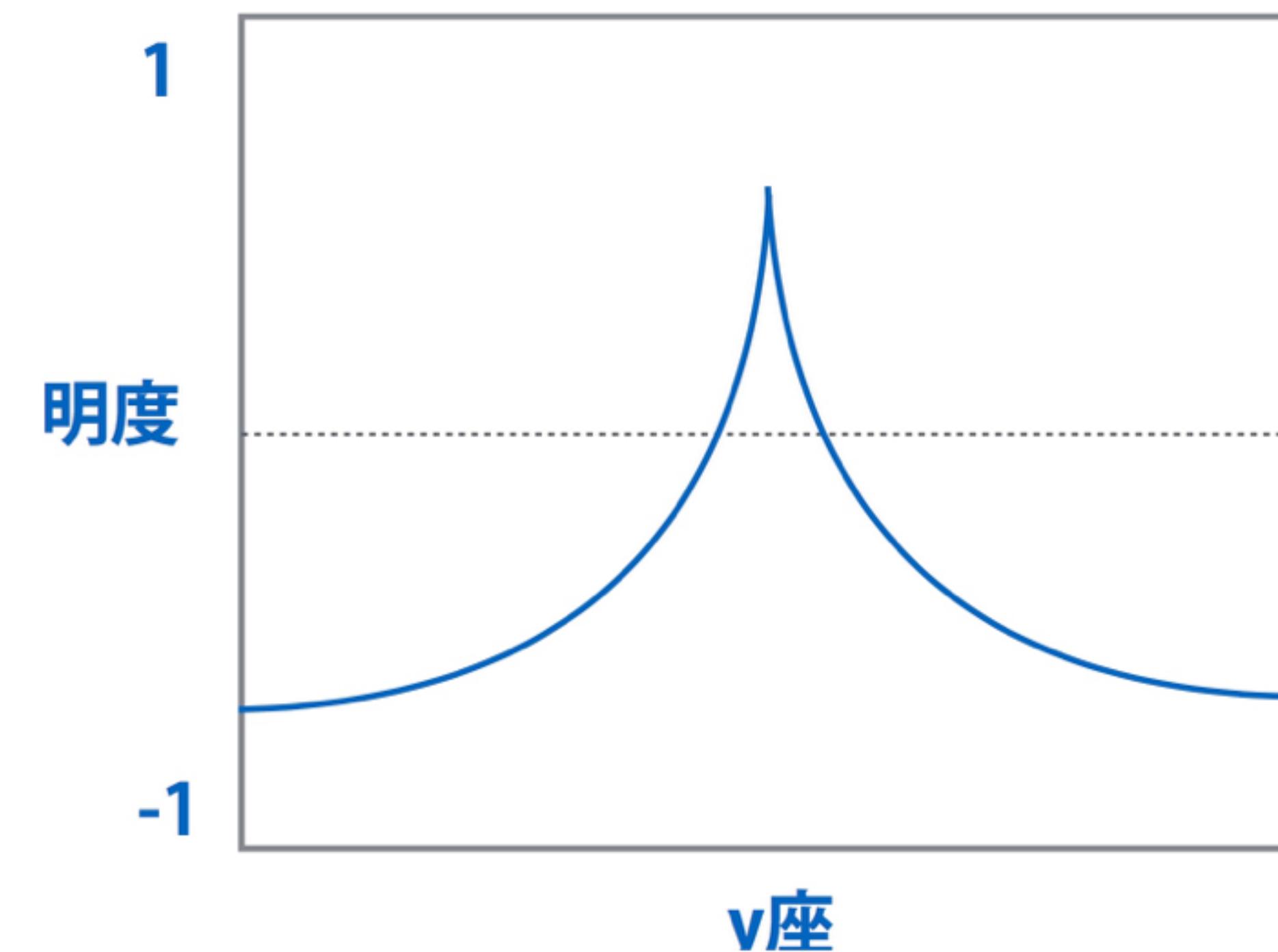
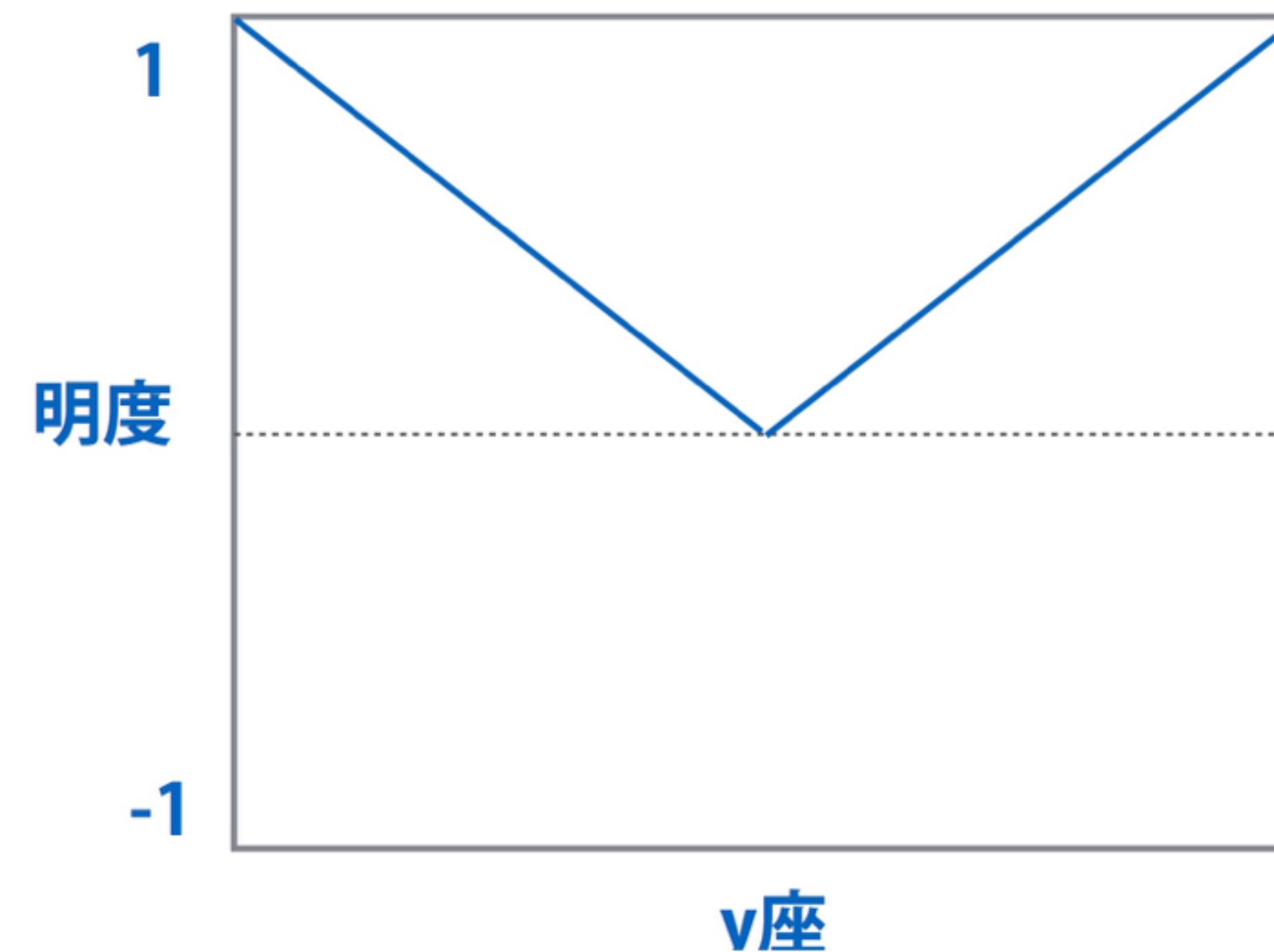
GLSLで形を描く

- ▶ 波の形が見えてきた!



GLSLで形を描く

- ▶ さらに1から引いて、乗算する



Uniforms - openFrameworksからShaderへ値を送出

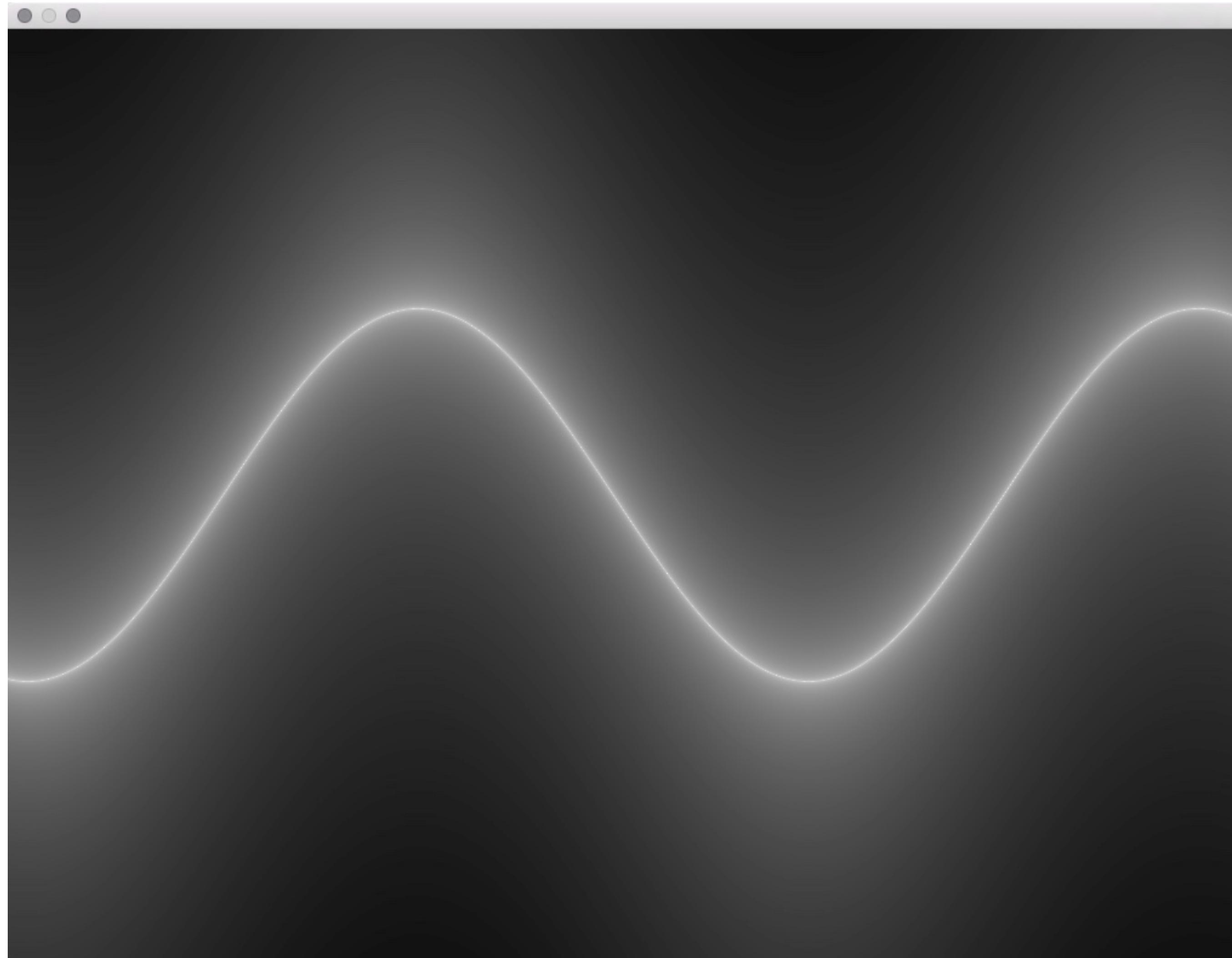
- ▶ さらに1から引いて、乗算する

```
uniform vec2 u_resolution; // 画面の解像度(width,height)
uniform float u_time;      // 起動してからの経過時間(秒)

void main() {
    //画面の解像度から、0.0～1.0に正規化する
    vec2 st = gl_FragCoord.xy / u_resolution;
    // 画面の中央をy軸の中心に
    st.y -= 0.5;
    // Sin波で濃淡をつける
    st.y += sin(st.x * 10.0 + u_time) * 0.2;
    // 1から引いて、乗算する
    vec3 color = vec3(1.0 - pow(abs(st.y), 0.2));
    gl_FragColor = vec4(color, 1.0);
}
```

GLSLで形を描く

- ▶ きれいなsin波が描けた!



Uniforms - openFrameworksからShaderへ値を送出

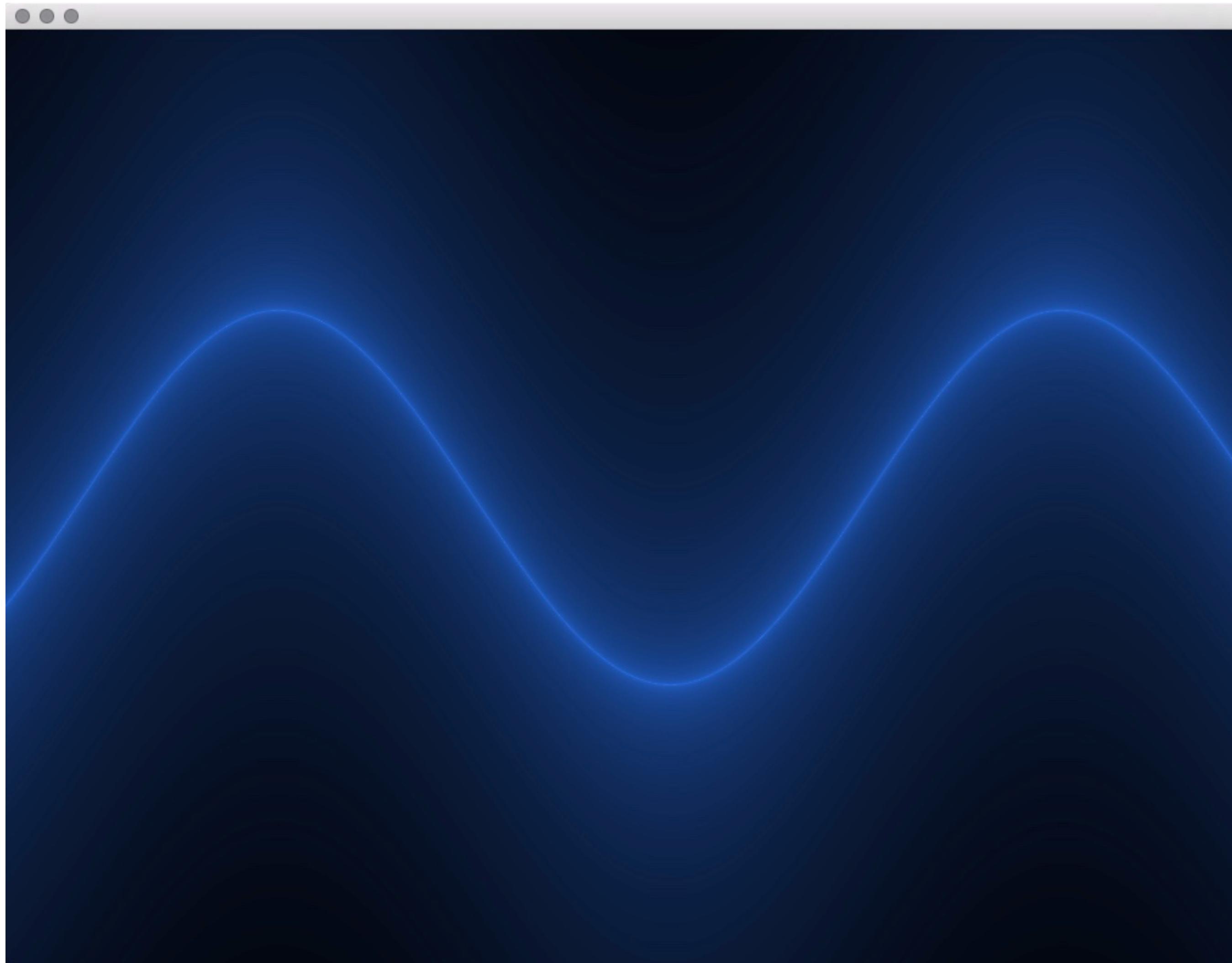
▶ 色合いを工夫してみる

```
uniform vec2 u_resolution; // 画面の解像度(width,height)
uniform float u_time; // 起動してからの経過時間(秒)

void main() {
    //画面の解像度から、0.0～1.0に正規化する
    vec2 st = gl_FragCoord.xy / u_resolution;
    // 画面の中央をy軸の中心に
    st.y -= 0.5;
    // Sin波で濃淡をつける
    st.y += sin(st.x * 10.0 + u_time) * 0.2;
    // 1から引いて、乗算する
    vec3 color = vec3(1.0 - pow(abs(st.y), 0.2));
    // 色を設定
    gl_FragColor = vec4(color.r * 0.2, color.g * 0.5, color.b * 1.0, 1.0);
}
```

GLSLで形を描く

- ▶ 好みの色を設定してみる



Uniforms - openFrameworksからShaderへ値を送出

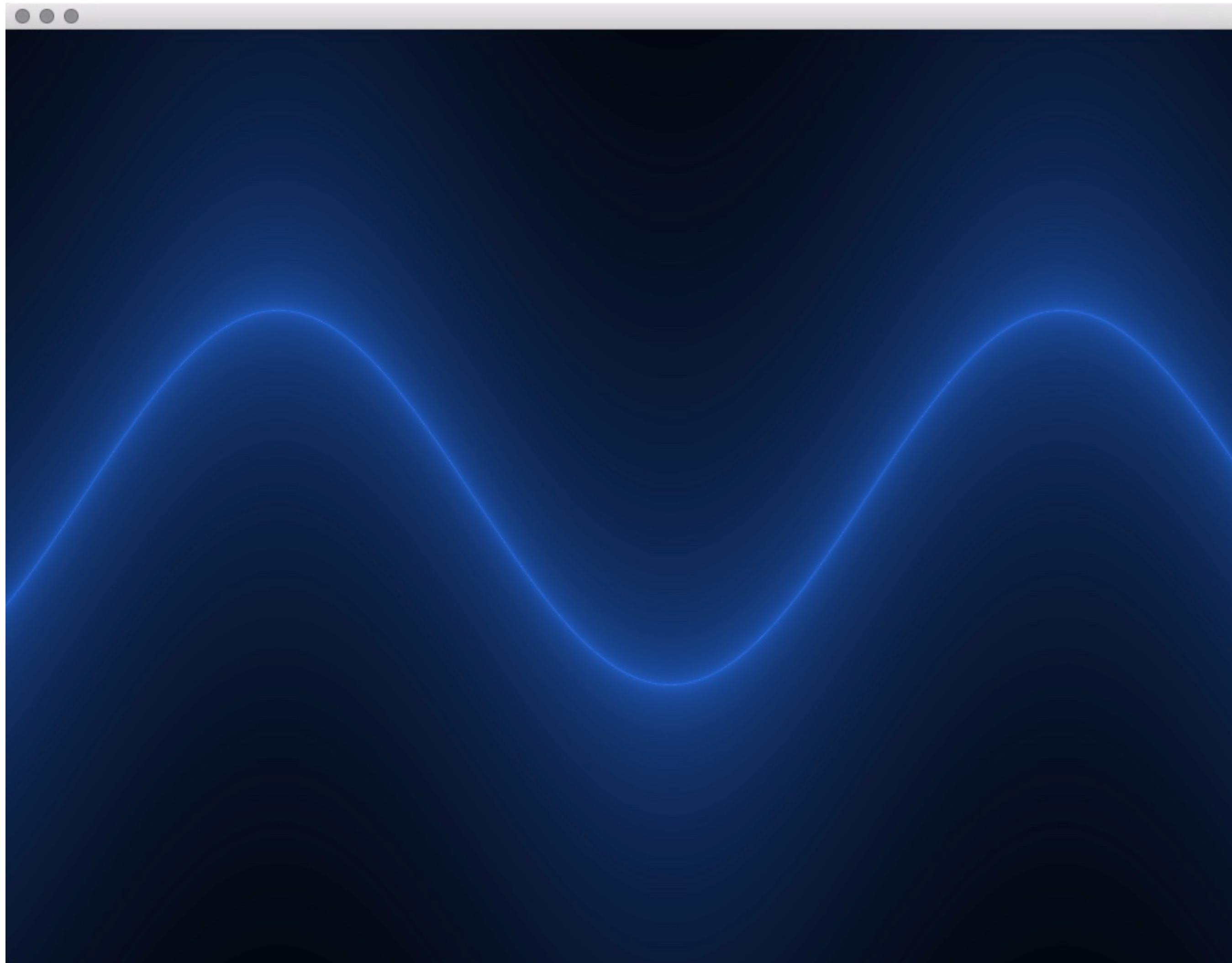
- ▶ さらに時間で振幅を変化させてみる

```
uniform vec2 u_resolution; // 画面の解像度(width,height)
uniform float u_time; // 起動してからの経過時間(秒)

void main() {
    //画面の解像度から、0.0～1.0に正規化する
    vec2 st = gl_FragCoord.xy / u_resolution;
    // 画面の中央をy軸の中心に
    st.y -= 0.5;
    // Sin波で濃淡をつける + 時間による変化
    st.y += sin(st.x * 10.0 + u_time) * 0.2 * sin(u_time);
    // 1から引いて、乗算する
    vec3 color = vec3(1.0 - pow(abs(st.y), 0.2));
    // 色を設定
    gl_FragColor = vec4(color.r * 0.2, color.g * 0.5, color.b * 1.0, 1.0);
}
```

GLSLで形を描く

- ▶ 時間によって変化する振幅



GLSLで形を描く

- ▶ Sin波のパラメータを、oFで生成して渡してみる
- ▶ oF側 → float array[] を生成し、setUniform1fv() で渡す
- ▶ GLSL側 → uniform float array[] で受ける
- ▶ oF側でランダムにパラメータ生成して、Shaderに入れてみる!

Uniforms - openFrameworksからShaderへ値を送出

▶ shader.frag

```
uniform vec2 u_resolution; // 画面の解像度(width,height)
uniform float u_time; // 起動してからの経過時間(秒)
const int NUM = 20; // 波の数
uniform float freq[20]; // oFから受けとる波の配列

void main() {
    vec3 color;
    // 波の数だけくりかえす
    for (int i = 0; i < NUM; i++) {
        //画面の解像度から、0.0～1.0に正規化する
        vec2 st = gl_FragCoord.xy / u_resolution;
        // 画面の中央をy軸の中心に
        st.y -= 0.5;
        // Sin波で濃淡をつける + 時間による変化
        st.y += sin(st.x * freq[i] + u_time) * 0.2 * sin(u_time * freq[i] * 0.1);
        // 1から引いて、乗算する
        color += vec3(1.0 - pow(abs(st.y), 0.75/float(NUM)));
        // 色を設定
        gl_FragColor = vec4(color.r * 0.2, color.g * 0.5, color.b * 1.0, 1.0);
    }
}
```

Uniforms - openFrameworksからShaderへ値を送出

▶ ofApp.h

```
#pragma once

#include "ofMain.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);

    ofShader shader;
    static const int NUM = 20;
    float freq[NUM];
};

};
```

Uniforms - openFrameworksからShaderへ値を送出

▶ ofApp.cpp

```
#include "ofApp.h"

void ofApp::setup(){
    ofSetFrameRate(60);
    // ランダムに周波数の配列を生成
    for (int i = 0; i < NUM; i++) {
        freq[i] = ofRandom(4.0, 10.0);
    }
}

void ofApp::update(){

}

void ofApp::draw(){
    shader.load("", "shader.frag");

    shader.begin();
    shader.setUniform1f("u_time", ofGetElapsedTimef());
    shader.setUniform2f("u_resolution", ofGetWidth(), ofGetHeight());
    //freqの配列をShaderに渡す
    shader.setUniform1fv("freq", freq, NUM);
    ofRect(0, 0, ofGetWidth(), ofGetHeight());
    shader.end();
}
```

GLSLで形を描く

- ▶ 折り重なるSin波!

