

IDD Media lab. 2015

openFrameworks - Addon 3

音の視覚化 - ofxFft、ofxProcessFFT

2015年6月2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

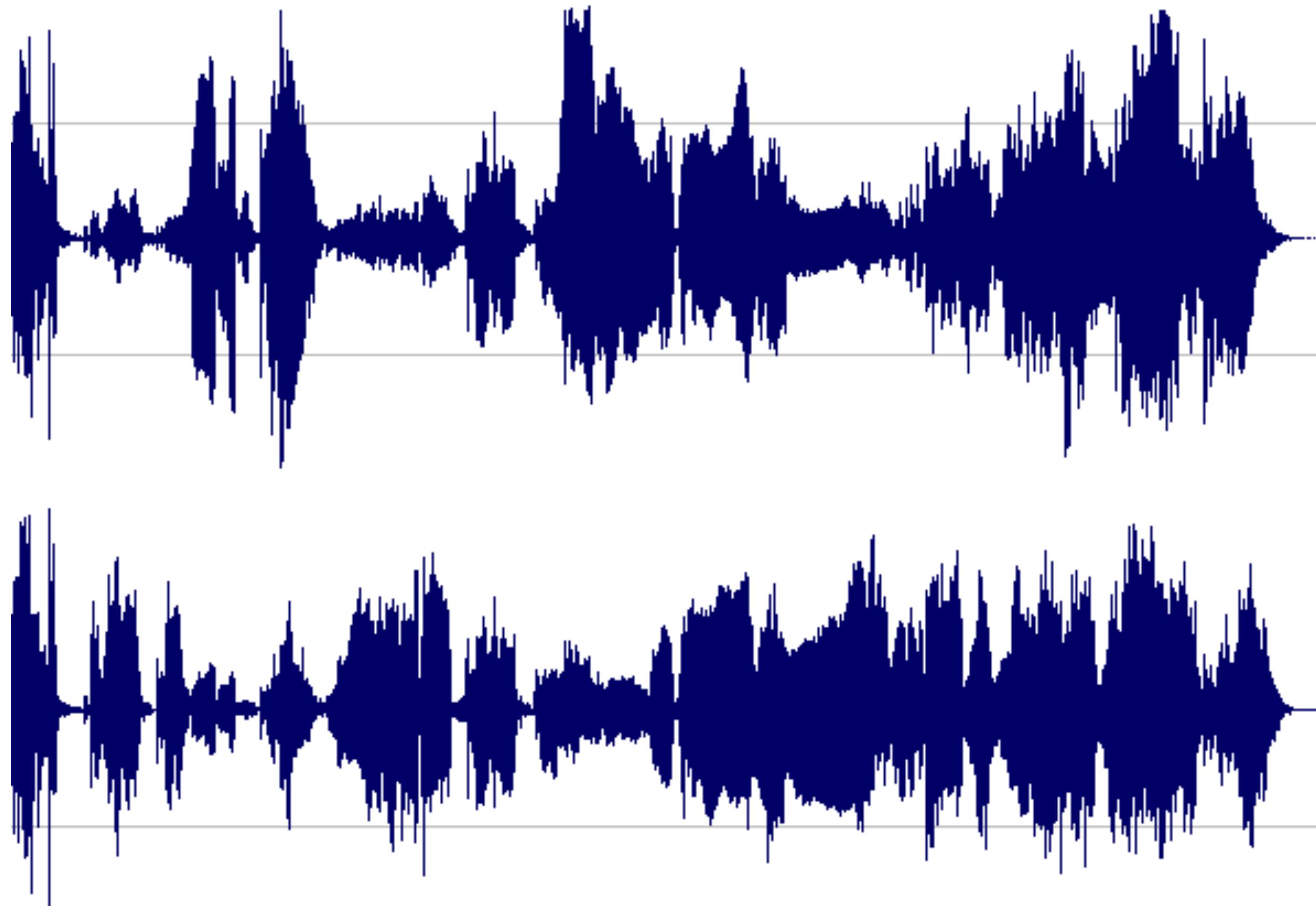
今日の内容

- ▶ オーディオ→ビジュアル
 - ▶ 音を視覚的に捉えるには?
 - ▶ 高速フーリエ変換(FFT)について
 - ▶ FFTを利用したサンプル
 - ▶ ofxFft
 - ▶ ofxProcessFFT

音響の視覚化

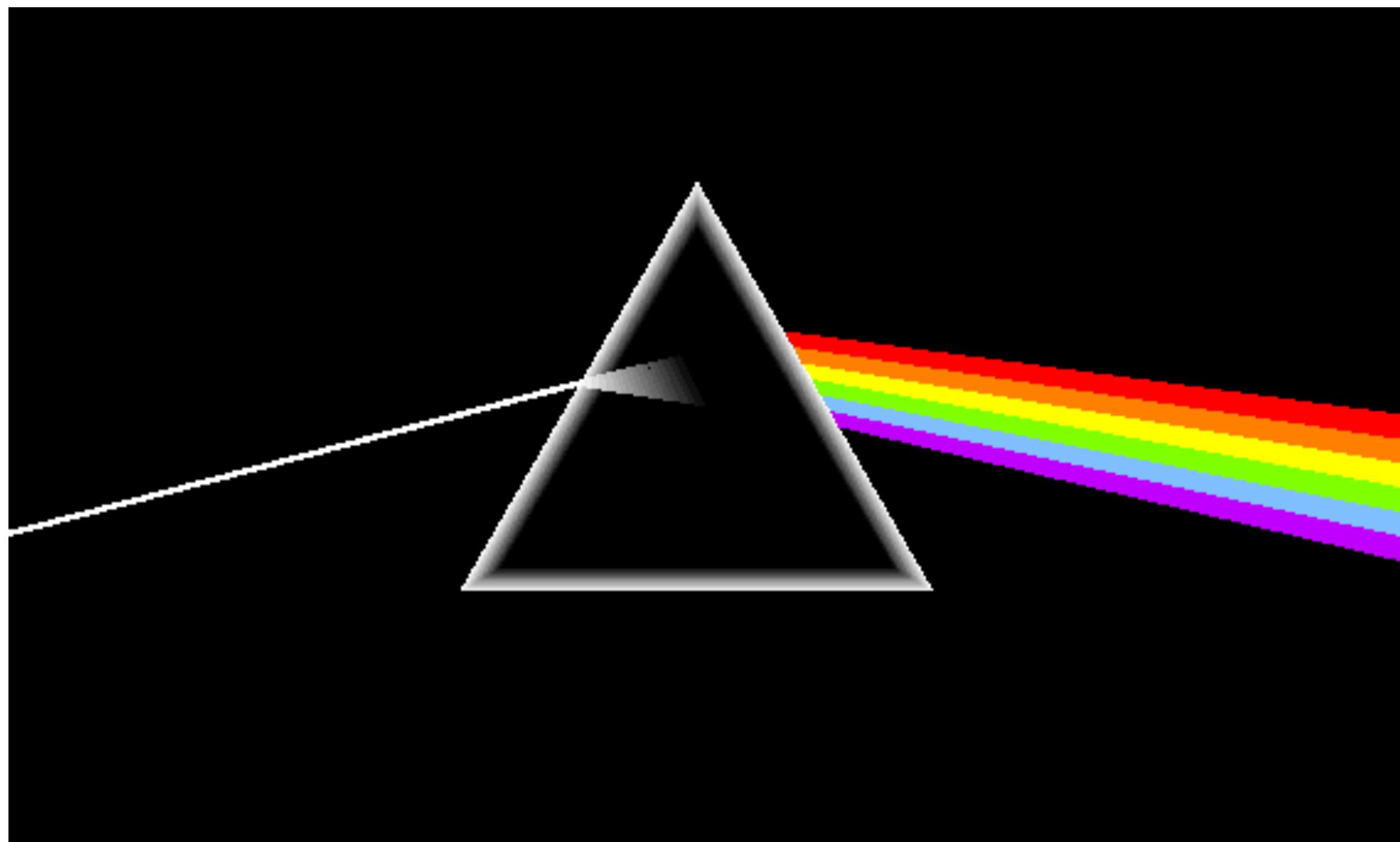
Audio into Visual.

- ▶ 音を視覚的に捉えるには?
- ▶ 波形から音を想像できるか?



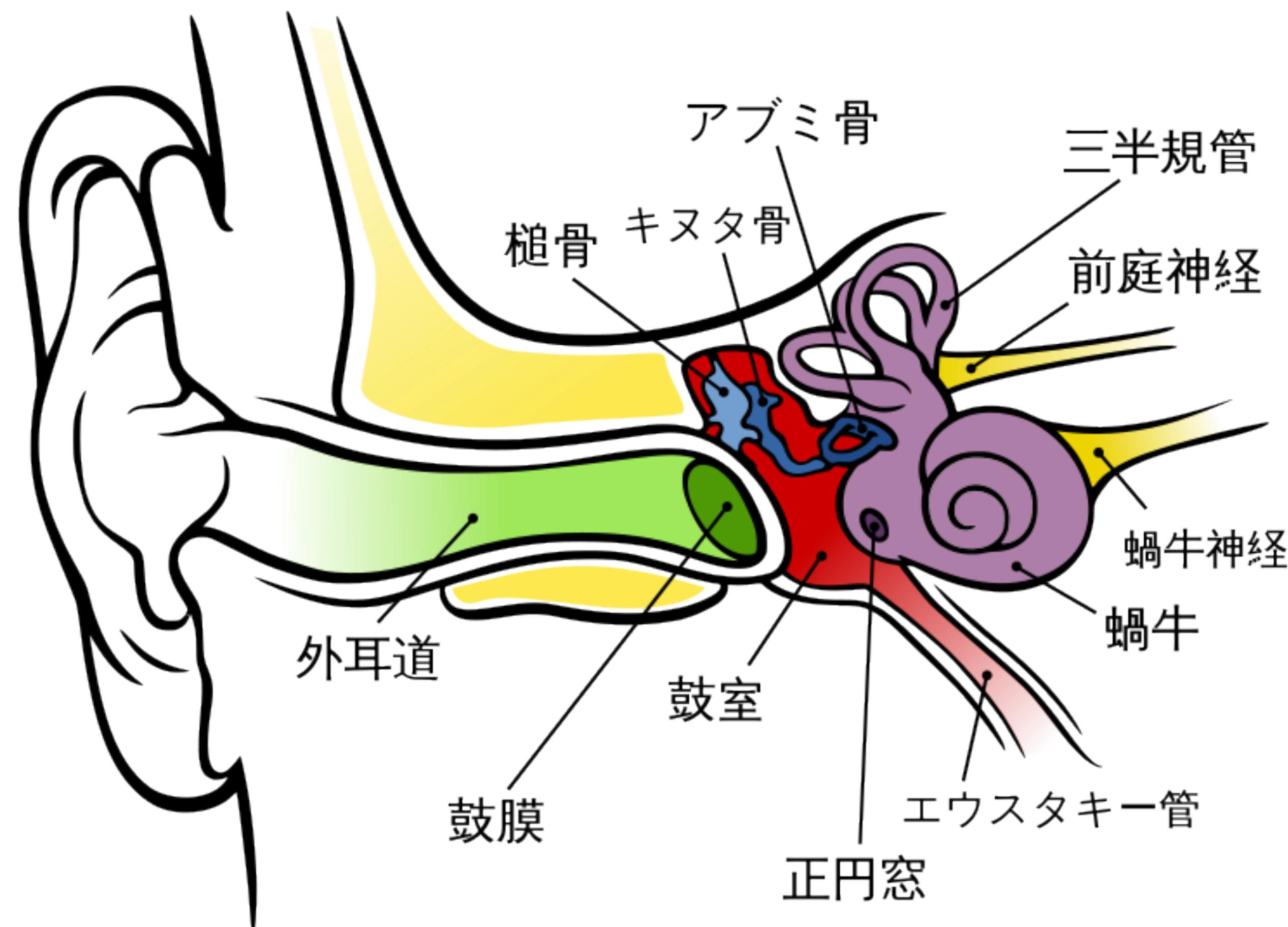
Audio into Visual.

- ▶ 我々は、音波の振動の形(音波)から直接音を聞いているわけではない
- ▶ 音に含まれる周波数の成分ごとに分析して聞いている
- ▶ 耳の中には周波数解析器のような機能が備わっている



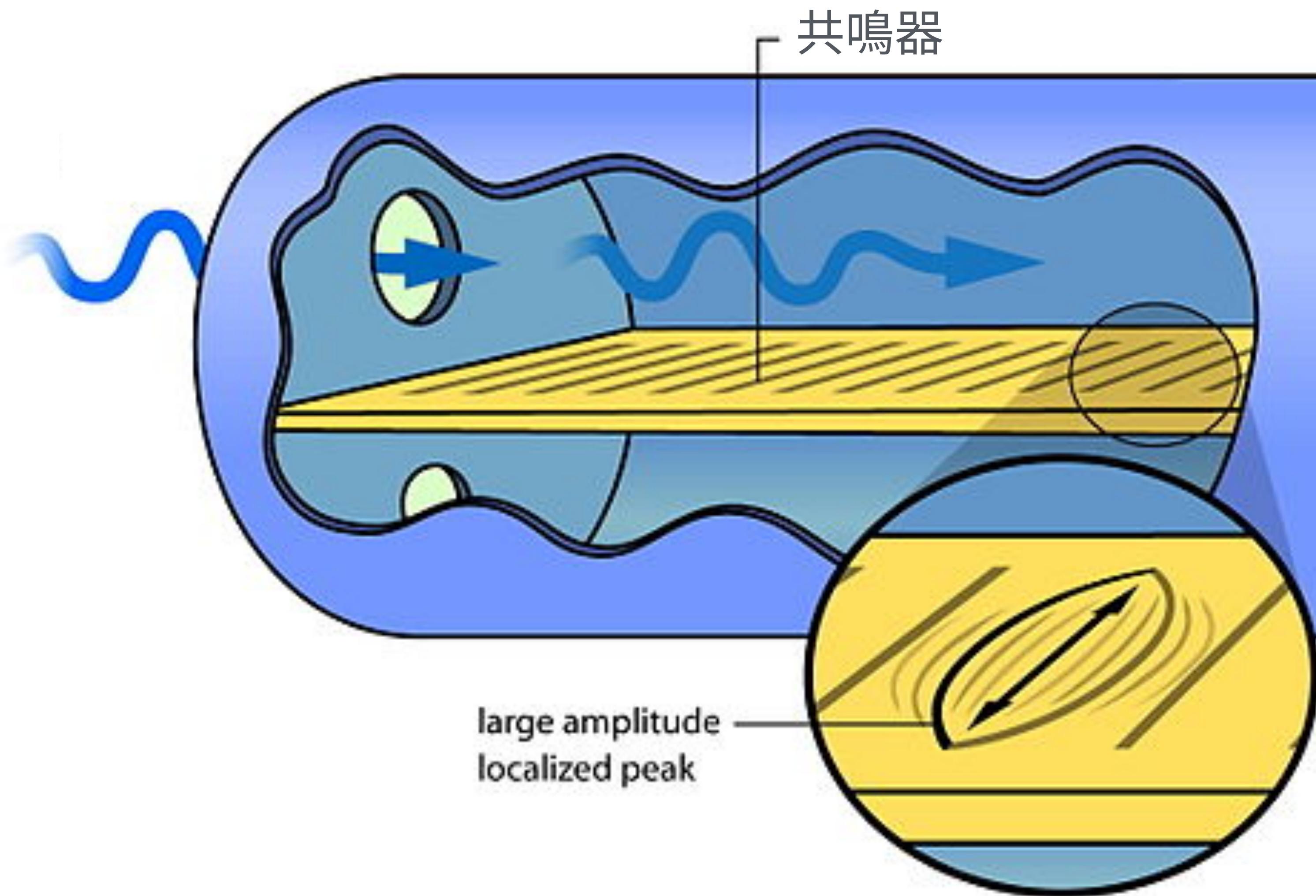
Audio into Visual.

▶ 耳の構造



Audio into Visual.

▶ 蝸牛の構造



Audio into Visual.

- ▶ では、耳が聞いているように音を見るにはどうすれば良いか?
- ▶ 音を周波数成分に分解する
- ▶ フーリエ変換: 音に含まれる周波数成分を数学的に解析できる

$$F(\xi) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\xi x} dx$$

$$f(x) = \int_{-\infty}^{\infty} F(\xi) e^{i2\pi\xi x} d\xi$$

Audio into Visual.

- ▶ では、耳が聞いているように音を見るにはどうすれば良いか?
- ▶ 音を周波数成分に分解する
- ▶ フーリエ変換: 音に含まれる周波数成分を数学的に解析できる

$$F(\xi) \text{フーリエ変換 : 波形} \rightarrow \int_{-\infty}^{\infty} f(x) e^{-i2\pi\xi x} dx$$

$$f(x) = \int_{-\infty}^{\infty} F(\xi) e^{i2\pi\xi x} d\xi$$

Audio into Visual.

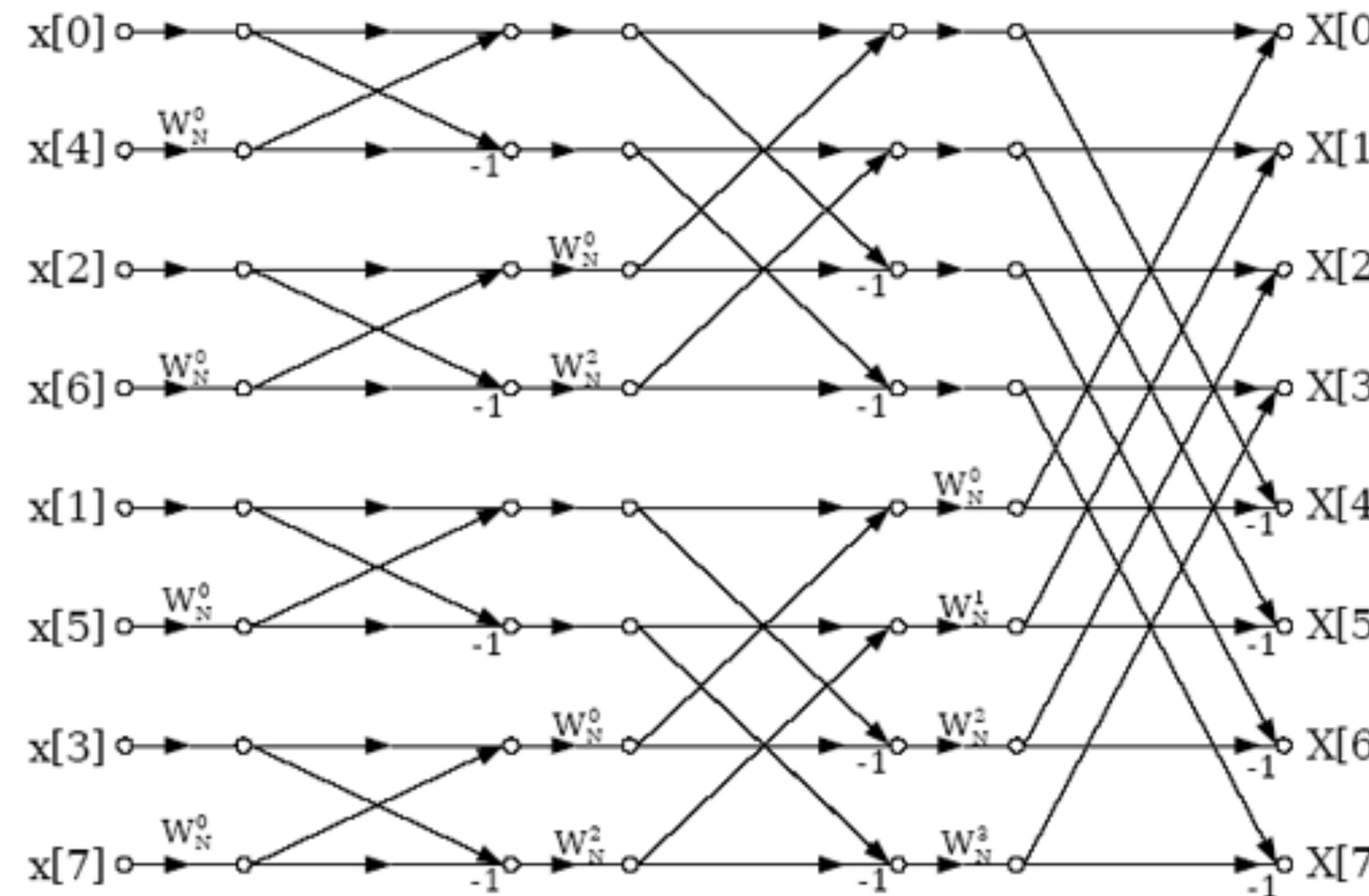
- ▶ では、耳が聞いているように音を見るにはどうすれば良いか?
- ▶ 音を周波数成分に分解する
- ▶ フーリエ変換: 音に含まれる周波数成分を数学的に解析できる

$$F(\xi) \text{フーリエ変換} : f(x) \rightarrow \int_{-\infty}^{\infty} e^{-i2\pi\xi x} dx$$

$$f(x) \text{フーリエ逆変換} : F(\xi) \rightarrow \int_{-\infty}^{\infty} F(\xi) e^{i2\pi\xi x} d\xi$$

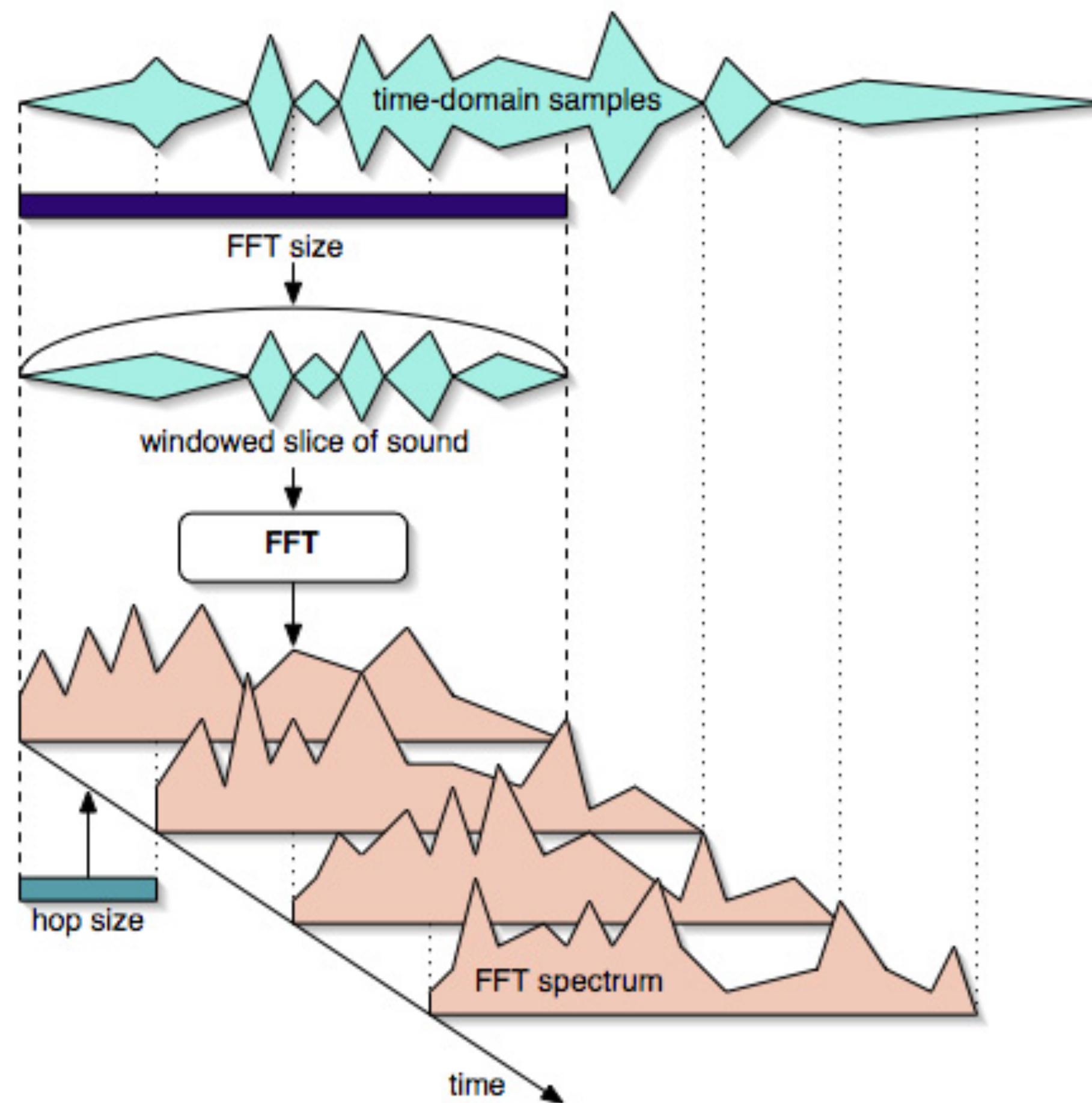
Audio into Visual.

- ▶ 高速フーリエ変換(FFT): フーリエ変換をコンピュータで高速に解析するためのアルゴリズム



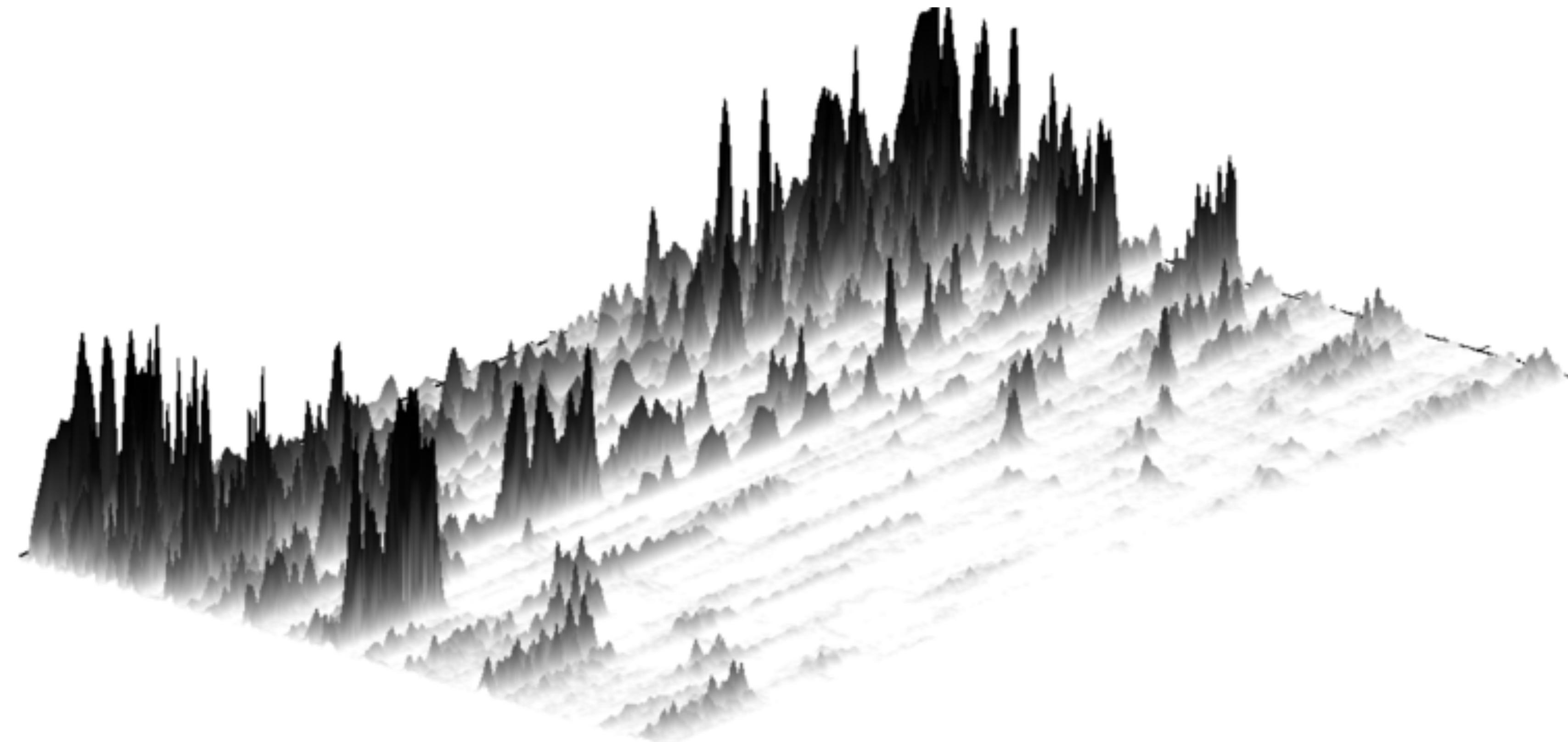
Audio into Visual.

- ▶ 短時間フーリエ変換(STFT): 波形を短かい区間で区切ってFFTを行い、周波数成分の時間的变化を分析



Audio into Visual.

- ▶ スペクトル(Spectrogram): 時間軸上での周波数成分の分析結果をプロットしたもの



Audio into Visual.

- ▶ FFT/IFFTのプログラムは自分で作る必要はない!
- ▶ たくさんのソースコードがパブリックドメインとして公開されている
- ▶ C++でも多くの実装
- ▶ <http://forum.openframeworks.cc/index.php?topic=186.0>

Audio into Visual.

► FFTW : <http://www.fftw.org/>



The FFTW logo consists of the letters "FFTW" in a bold, sans-serif font. The "F" and "T" are black, while the "W" is red.

[Download](#) [GitHub](#) [Mailing List](#) [Benchmark](#) [Features](#) [Documentation](#) [FAQ](#) [Links](#) [Feedback](#)

Introduction

FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST). We believe that FFTW, which is [free software](#), should become the [FFT](#) library of choice for most applications.

The latest official release of FFTW is version **3.3.4**, available from [our download page](#). Version 3.3 introduced support for the AVX x86 extensions, a distributed-memory implementation on top of MPI, and a Fortran 2003 API. Version 3.3.1 introduced support for the ARM Neon extensions. See the [release notes](#) for more information.

The FFTW package was developed at [MIT](#) by [Matteo Frigo](#) and [Steven G. Johnson](#).

Our [benchmarks](#), performed on a variety of platforms, show that FFTW's performance is typically superior to that of other publicly available FFT software, and is even competitive with vendor-tuned codes. In contrast to vendor-tuned codes, however, FFTW's performance is *portable*: the same program will perform well on most architectures without modification. Hence the name, "FFTW," which stands for the somewhat whimsical title of "**F**astest **F**ourier **T**ransform **i**n the **W**est."

Subscribe to the [fftw-announce mailing list](#) to receive release announcements (or use the web feed [RSS](#)).

Features

FFTW 3.3.4 is the latest official version of FFTW (refer to the [release notes](#) to find out what is new). Here is a list of some of FFTW's more interesting features:

- [Speed](#). (Supports SSE/SSE2/Altivec, since version 3.0. Version 3.3.1 supports AVX and ARM Neon.)
- Both one-dimensional and [multi-dimensional](#) transforms.
- [Arbitrary-size](#) transforms. (Sizes with small prime factors are best, but FFTW uses O(N log N) algorithms even for prime sizes.)
- Fast transforms of [purely real](#) input or output data.
- Transforms of real even/odd data: the [discrete cosine transform](#) (DCT) and the [discrete sine transform](#) (DST), types I-IV. (Version 3.0 or later.)
- Efficient handling of [multiple](#), [strided](#) transforms. (This lets you do things like transform multiple arrays at once, transform one dimension of a multi-dimensional array, or transform one field of a multi-component array.)
- [Parallel transforms](#): parallelized code for platforms with SMP machines with some flavor of [threads](#) (e.g. POSIX) or [OpenMP](#). An [MPI](#) version for distributed-memory transforms is also available in FFTW 3.3.
- [Portable](#) to any platform with a C compiler.
- [Documentation](#) in HTML and other formats.
- Both [C](#) and [Fortran](#) interfaces.
- [Free](#) software, released under the GNU General Public License (GPL, see [FFTW license](#)). (Non-free licenses may also be purchased from [MIT](#), for users who do not want their programs protected by the GPL. [Contact us](#) for details.) (See also the [FAQ](#).)

If you are still using **FFTW 2.x**, please note that FFTW 2.x was last updated in 1999 and it is obsolete. Please upgrade to FFTW 3.x. The API of FFTW 3.x is [incompatible](#) with that of FFTW 2.x, for reasons of performance and generality (see the [FAQ](#) or the [manual](#)).

Documentation

First, read the [FFTW Frequently Asked Questions](#) document.

Manual: [HTML](#) or [PDF](#).

the FFTW library and FFTW-related terms are [copyrighted](#) by its authors and/or its copyright holders. All rights reserved.

Audio into Visual.

- ▶ Kiss FFT : <http://sourceforge.net/projects/kissfft/>

The screenshot shows the SourceForge project page for 'Kiss FFT'. The top navigation bar includes links for 'Browse', 'Enterprise', 'Blog', 'Jobs', 'Deals', 'Help', 'Log In' or 'Join', and 'SOLUTION CENTERS' like 'Go Parallel', 'Resources', and 'Newsletters'. A banner for 'INTEROP TOKYO' is displayed, featuring a large 3D geometric logo and text in Japanese. Below the banner, the project title 'Kiss FFT' is prominently shown, along with a 'Hover to Expand' button. The main content area includes a summary of project statistics (5.0 Stars, 307 Downloads), a 'Download' button for 'kiss_fft130.zip', and a 'Description' section. The description text reads: 'A Fast Fourier Transform based up on the principle, "Keep It Simple, Stupid." Kiss FFT is a very small, reasonably efficient, mixed radix FFT library that can use either fixed or floating point data types.' There is also a link to the 'Kiss FFT Web Site'. On the right side of the page, there is a promotional image for 'teratail' featuring a woman and Japanese text. The footer contains links for 'Categories' (Analysis, Mathematics), 'License' (BSD License), and 'Asahi' (水と混ぜるだけ!).

ofxFft

ofxFft

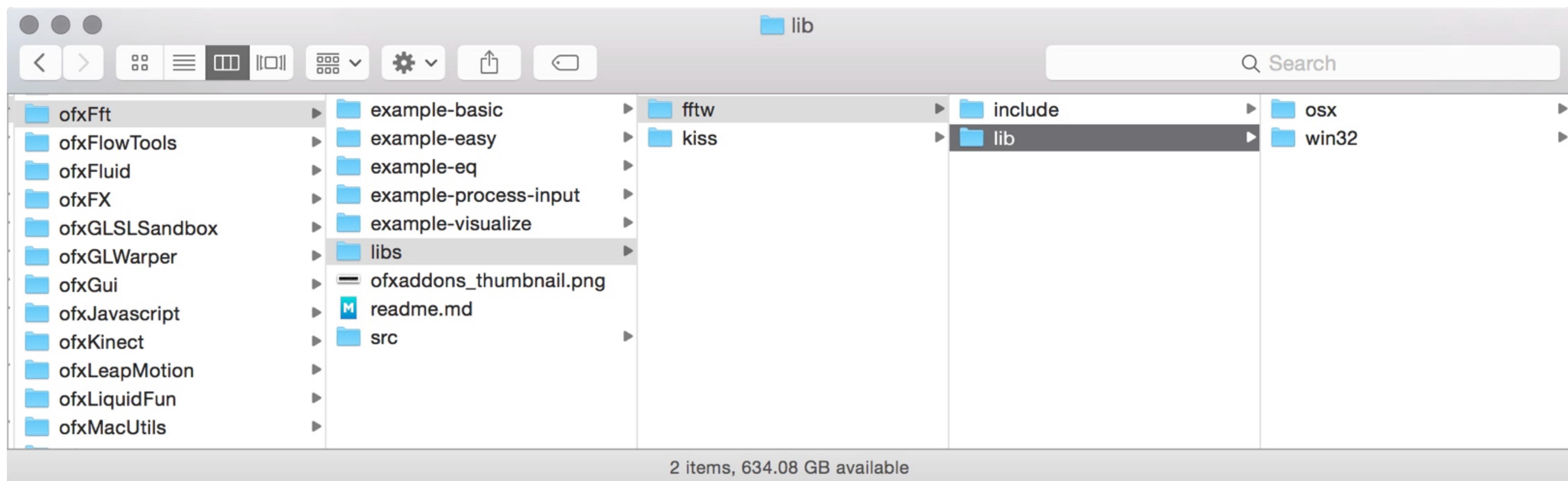
- ▶ 今回は、音のリアルタイムFFT解析に、ofxFftを使用します
- ▶ Kyle McDonaldさん作
- ▶ <https://github.com/kylemcdonald/ofxFft>

ofxFft

- ▶ ofxFftは、2種類のFFTのアルゴリズムを選択して使用可能
- ▶ KISS FFT
- ▶ <http://sourceforge.net/projects/kissfft/>
- ▶ FFTW
- ▶ <http://www.fftw.org/>

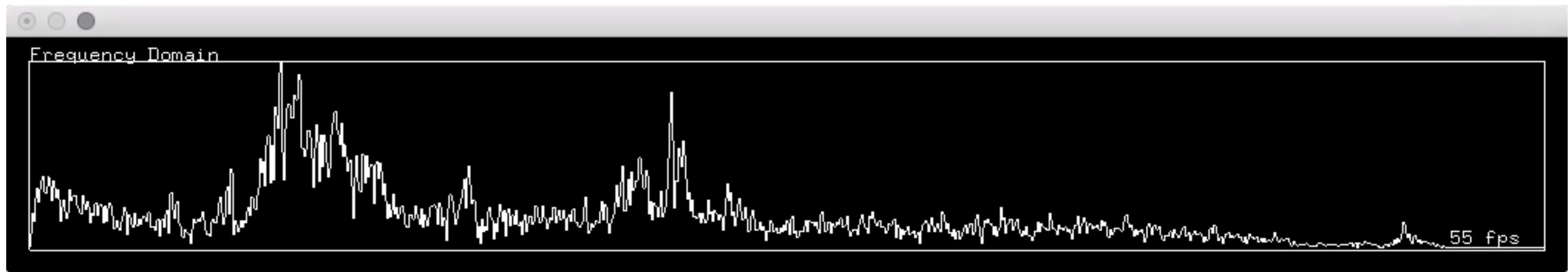
ofxFft

- ▶ ofxFftで、FFTWを使用する場合には注意が必要
- ▶ あらかじめコンパイルされたFFTWのライブラリを下記からダウンロード
- ▶ <https://github.com/downloads/kylemcdonald/ofxFft/fftw-libs.zip>
- ▶ 解凍したlibフォルダの中身 “osx” と “win32” を以下の場所にコピー
- ▶ addons/ofxFft/libs/fftw/lib/



ofxFft

- ▶ まずは、サンプルが動くか確認
- ▶ “example-basic”をビルドしてみる
- ▶ 以下のようにスペクトラムが表示されればOK

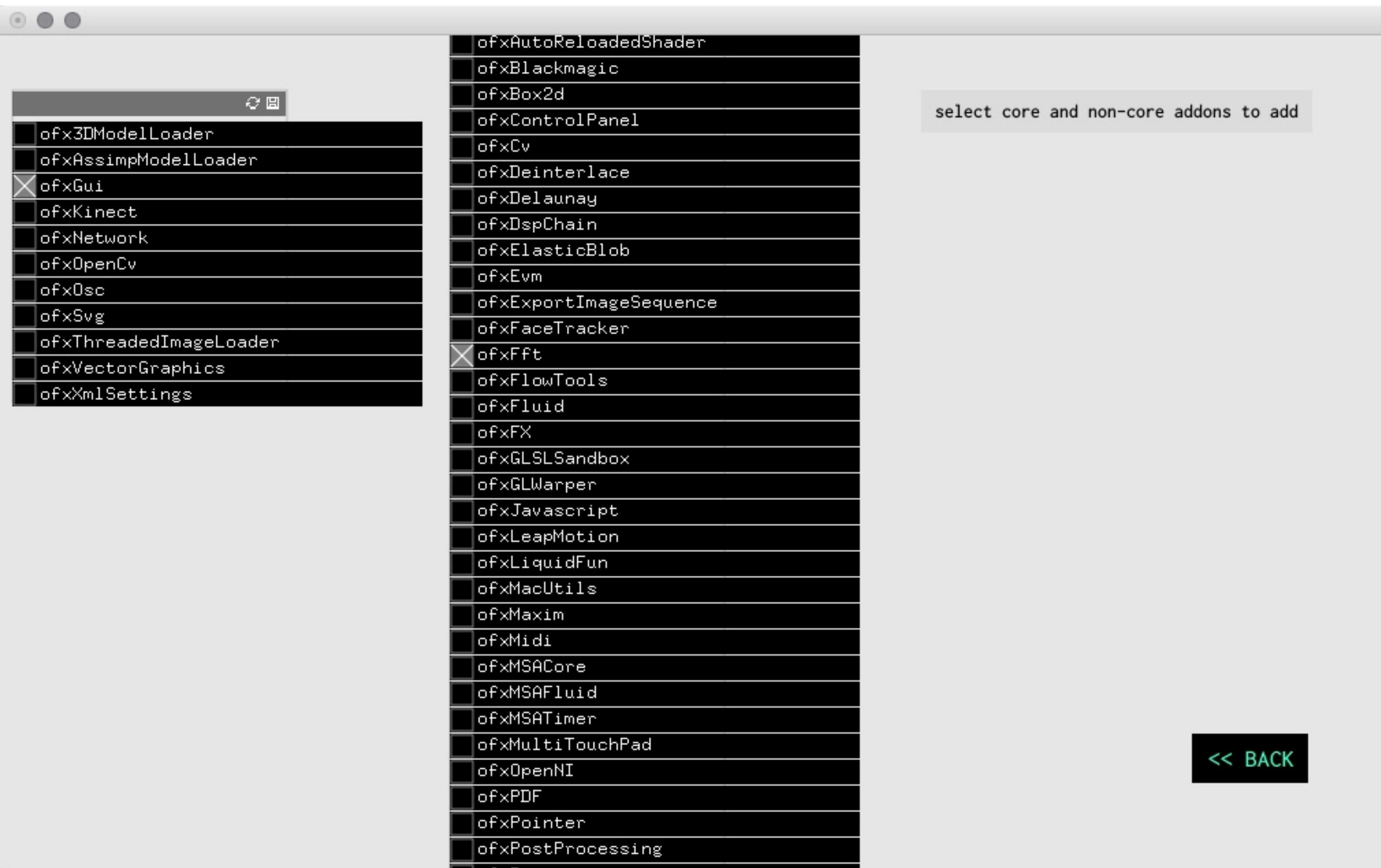


ofxFft

- ▶ example-basic、ちょっと複雑…
 - ▶ もうすこし簡略化したサンプルで、基本を理解したい
 - ▶ ofxFftに含まれている、ofxEasyFftを使用するとラクチン
 - ▶ ofxEasyFftの“getBins()”で取得される値は、0.0～1.0 の範囲に正規化される
-
- ▶ 早速作ってみましょう!!

ofxFft

- ▶ ofxFftと、ofxGuiをチェックして、プロジェクト生成



ofxFft

▶ ofApp.h

```
#pragma once

#include "ofMain.h"
#include "ofxEasyFft.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);

    ofxEasyFft fft;      // ofxEasyFftインスタンス
};
```

ofxFft

▶ ofApp.cpp

```
#include "ofApp.h"

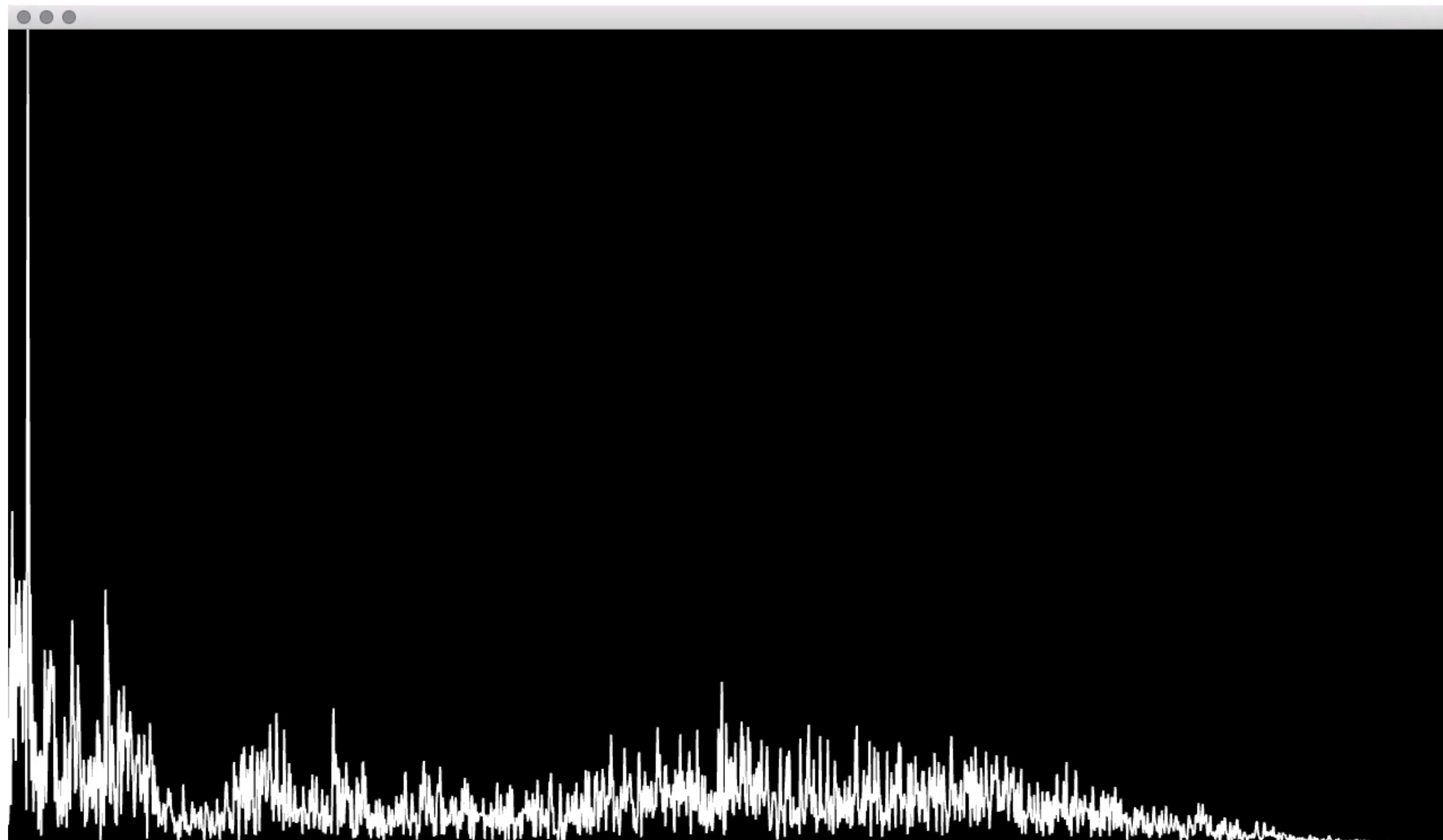
void ofApp::setup(){
    ofSetFrameRate(60);
    ofSetVerticalSync(true);
    ofBackground(0);
    // FFTのサンプル数(2の累乗)を指定して初期化
    fft.setup(pow(2.0, 12.0));
}

void ofApp::update(){
    fft.update(); // FFT更新
}

void ofApp::draw(){
    // float型の配列にFFT結果を格納
    vector<float> buffer;
    buffer = fft.getBins();
    // グラフに描画
    ofNoFill();
    ofSetLineWidth(2.0);
    ofBeginShape();
    for (int i = 0; i < buffer.size(); i++) {
        float x = ofMap(i, 0, buffer.size(), 0, ofGetWidth());
        float y = ofMap(buffer[i], 0, 1, ofGetHeight(), 0);
        ofVertex(x, y);
    }
    ofEndShape();
}
```

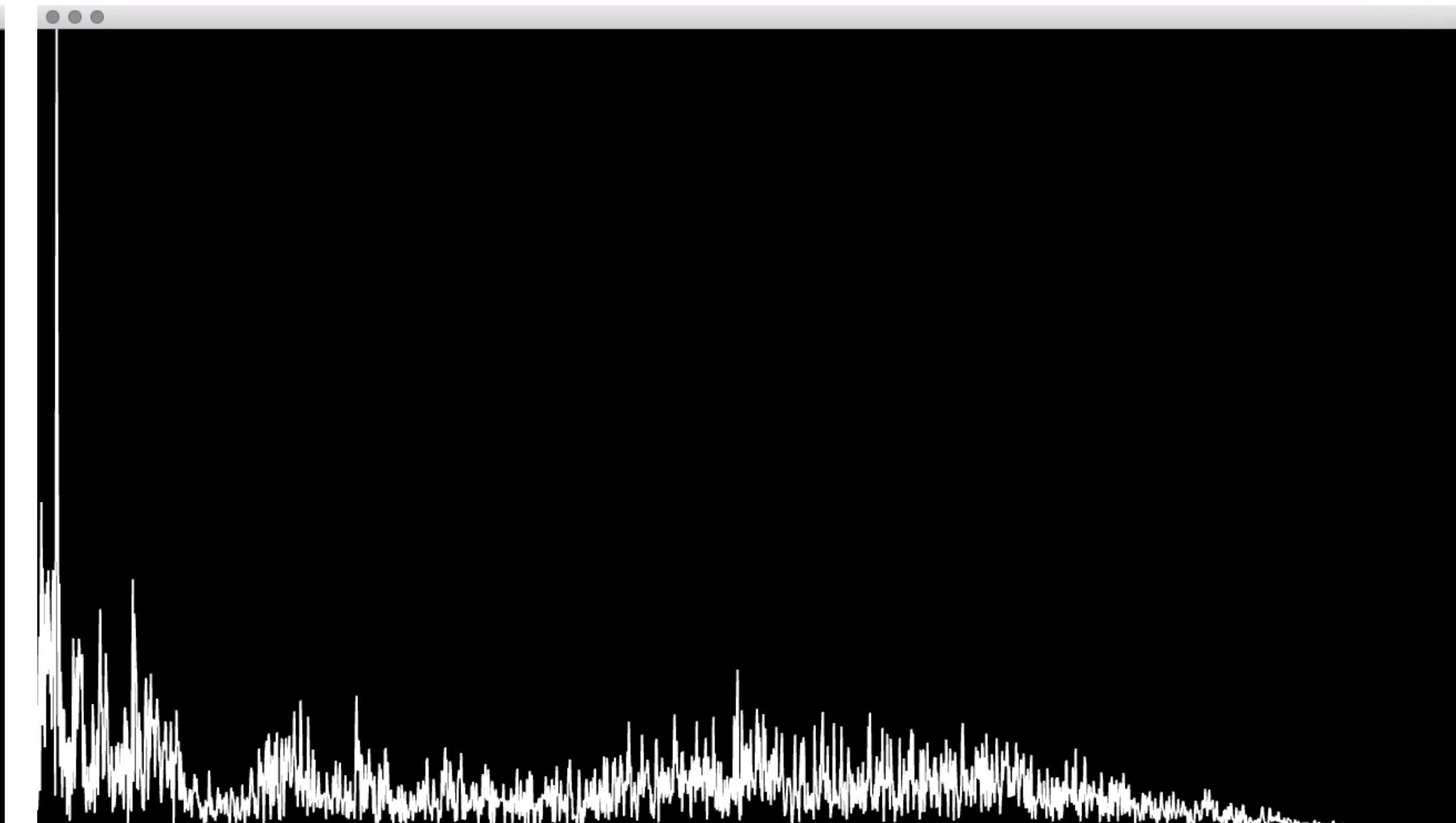
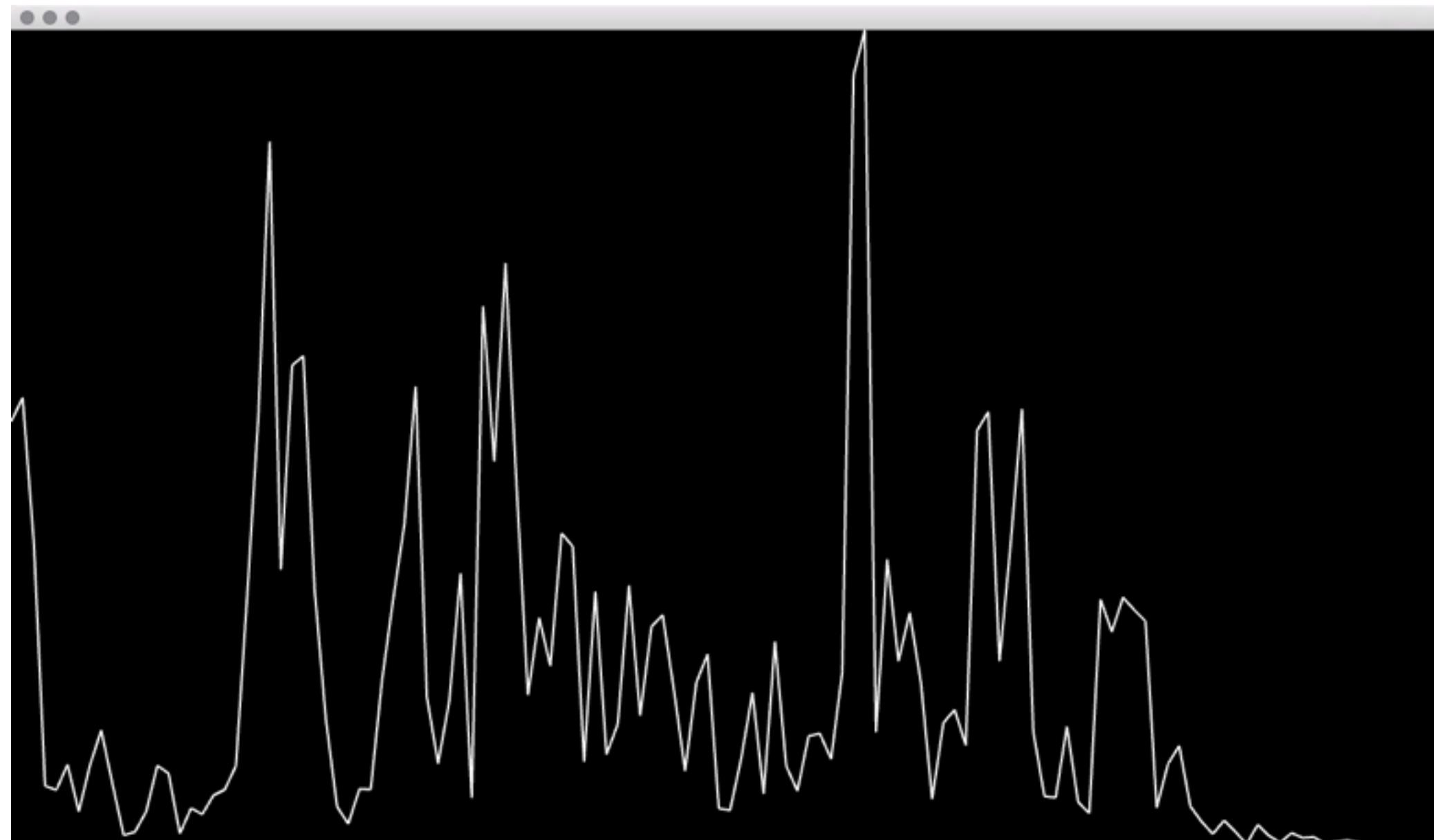
ofxFft

- ▶ オーディオの周波数成分（スペクトラム）をグラフで表示



ofxFft

- ▶ `fft.setup()` で設定するFFTのサンプル数によって、周波数の解像度が変化する
- ▶ 周波数の解像度を上げると、時間解像度が下がる



ofxFft

- ▶ グラフの描画を工夫してみる
- ▶ 左右対象の棒グラフに
- ▶ 周波数帯域によって、色相を塗り分けてみる

ofxFft

▶ ofApp.cpp

```
#include "ofApp.h"

void ofApp::setup(){
    ofSetFrameRate(60);
    ofSetVerticalSync(true);
    ofBackground(0);
    // FFTのサンプル数(2の累乗)を指定して初期化
    fft.setup(pow(2.0, 8.0));
}

void ofApp::update(){
    fft.update(); // FFT更新
}

void ofApp::draw(){
    // float型の配列にFFT結果を格納
    vector<float> buffer;
    buffer = fft.getBins();

    // グラフに描画
    ofSetLineWidth(ofGetWidth() / float(buffer.size()) / 2.0 - 0.5);
    for (int i = 0; i < buffer.size(); i++) {
        // 色を設定
        float hue = ofMap(i, 0, buffer.size(), 0, 160);
        ofColor col;
        col.setHsb(hue, 255, 255);
        ofSetColor(col);
        // 右
        float rx = ofMap(i, 0, buffer.size(), ofGetWidth() / 2.0, ofGetWidth());
        ofLine(rx, ofGetHeight(), rx, ofGetHeight() - buffer[i]);
    }
}
```

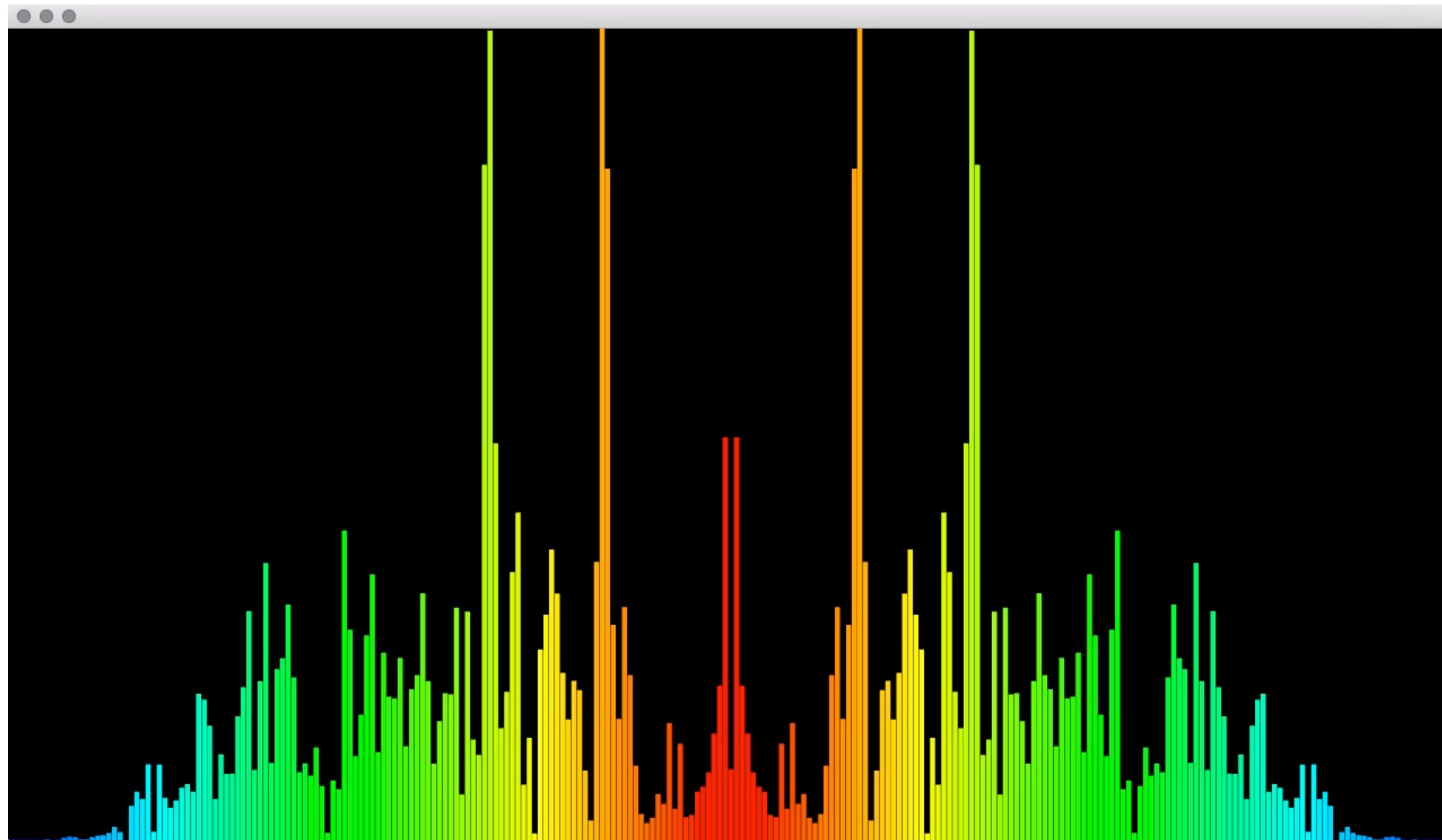
ofxFft

▶ ofApp.cpp

```
    float y = ofMap(buffer[i], 0, 1, ofGetHeight(), 0);
    ofLine(rx, ofGetHeight(), rx, y);
    // 左
    float lx = ofMap(i, 0, buffer.size(), ofGetWidth() / 2.0, 0);
    ofLine(lx, ofGetHeight(), lx, y);
}
}
```

ofxFft

- ▶ 左右対象のカラフルなグラフに!



ofxFft

- ▶ グラフの高さではなく、バーの色の明度で表現するとどうなるか?
- ▶ グラフを上から眺めたイメージ
- ▶ さらにGUIでパラメータを調整可能に

ofxFft

▶ ofApp.h

```
#pragma once

#include "ofMain.h"
#include "ofxEasyFft.h"
#include "ofxGui.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);

    ofxEasyFft fft;           // ofxEasyFftインスタンス
    ofxPanel gui;             // GUI
    ofxFLOATSlider saturation; // 彩度
};
```

ofxFft

▶ ofApp.cpp

```
#include "ofApp.h"

void ofApp::setup(){
    ofSetFrameRate(60);
    ofSetVerticalSync(true);
    ofBackground(0);
    // FFTのサンプル数(2の累乗)を指定して初期化
    fft.setup(pow(2.0, 10.0));
    // GUI
    gui.setup();
    gui.add(saturation.setup("Saturation", 127, 0, 255));
    gui.loadFromFile("settings.xml");
}

void ofApp::update(){
    fft.update(); // FFT更新
}

void ofApp::draw(){
    // float型の配列にFFT結果を格納
    vector<float> buffer;
    buffer = fft.getBins();
    // グラフに描画
    ofSetLineWidth(ofGetWidth() / float(buffer.size()) / 2.0);
    ofPushMatrix();
    ofTranslate(ofGetWidth()/2, ofGetHeight()/2);
    for (int i = 0; i < buffer.size(); i++) {
        // 色を設定
        float hue = ofMap(i, 0, buffer.size(), 0, 160);
        ofSetColor(hue, 0, 255);
        ofLine(i * ofGetWidth() / float(buffer.size()), 0, i * ofGetWidth() / float(buffer.size()), ofGetHeight());
    }
}
```

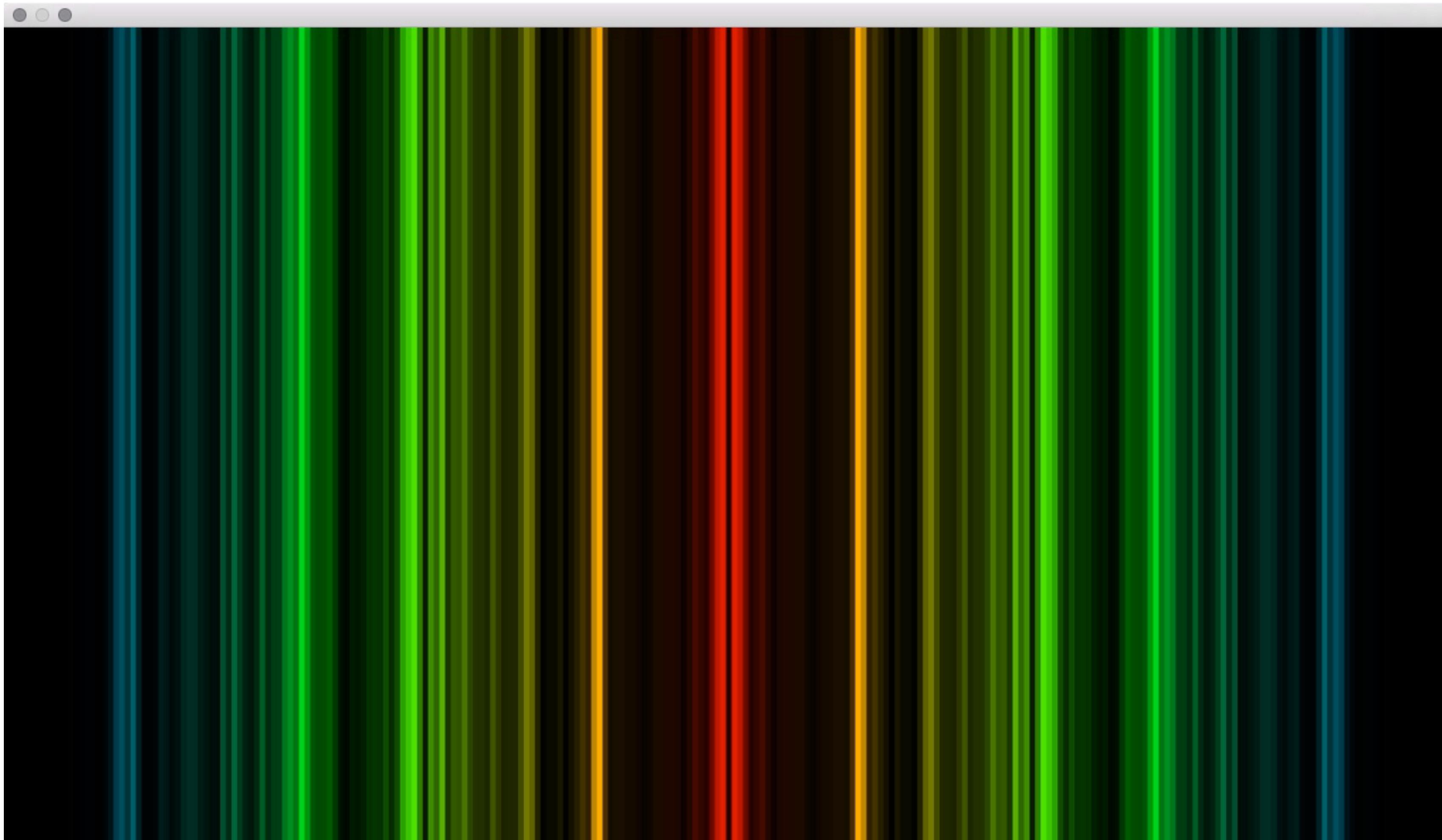
ofxFft

▶ ofApp.cpp

```
float br = ofMap(buffer[i], 0, 1, 0, 255);
ofColor col;
col.setHsb(hue, saturation, br);
ofSetColor(col);
// 右
float rx = ofMap(i, 0, buffer.size(), 0, ofGetWidth() / 2.0);
ofLine(rx, -ofGetWidth() / 2.0, rx, ofGetWidth() / 2.0);
// 左
float lx = ofMap(i, 0, buffer.size(), 0, -ofGetWidth() / 2.0);
ofLine(lx, -ofGetWidth() / 2.0, lx, ofGetWidth() / 2.0);
}
ofPopMatrix();
gui.draw(); // GUI描画
}
```

ofxFft

- ▶ 色の濃度による音の視覚化!



ofxFft

- ▶ さらに、色のパターンをクロスして重ねてみる!

ofxFft

▶ ofApp.h

```
#pragma once

#include "ofMain.h"
#include "ofxEasyFft.h"
#include "ofxGui.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);

    ofxEasyFft fft;          // ofxEasyFftインスタンス
    ofxPanel gui;            // GUI
    ofxFLOATSlider saturation; // 彩度
};

};
```

ofxFft

▶ ofApp.cpp

```
#include "ofApp.h"

void ofApp::setup(){
    ofSetFrameRate(60);
    ofSetVerticalSync(true);
    ofBackground(0);
    // FFTのサンプル数(2の累乗)を指定して初期化
    fft.setup(pow(2.0, 12.0));
    // GUI
    gui.setup();
    gui.add(saturation.setup("Saturation", 127, 0, 255));
}

void ofApp::update(){
    fft.update(); // FFT更新
}

void ofApp::draw(){
    // float型の配列にFFT結果を格納
    vector<float> buffer;
    buffer = fft.getBins();

    // グラフに描画
    ofPushMatrix();
    ofTranslate(ofGetWidth()/2, ofGetHeight()/2);
    ofSetLineWidth(ofGetWidth() / float(buffer.size()));
    ofEnableBlendMode(OF_BLENDMODE_ADD);
    for (int j = 0; j < 2; j++) {
        for (int i = 0; i < buffer.size(); i++) {
```

ofxFft

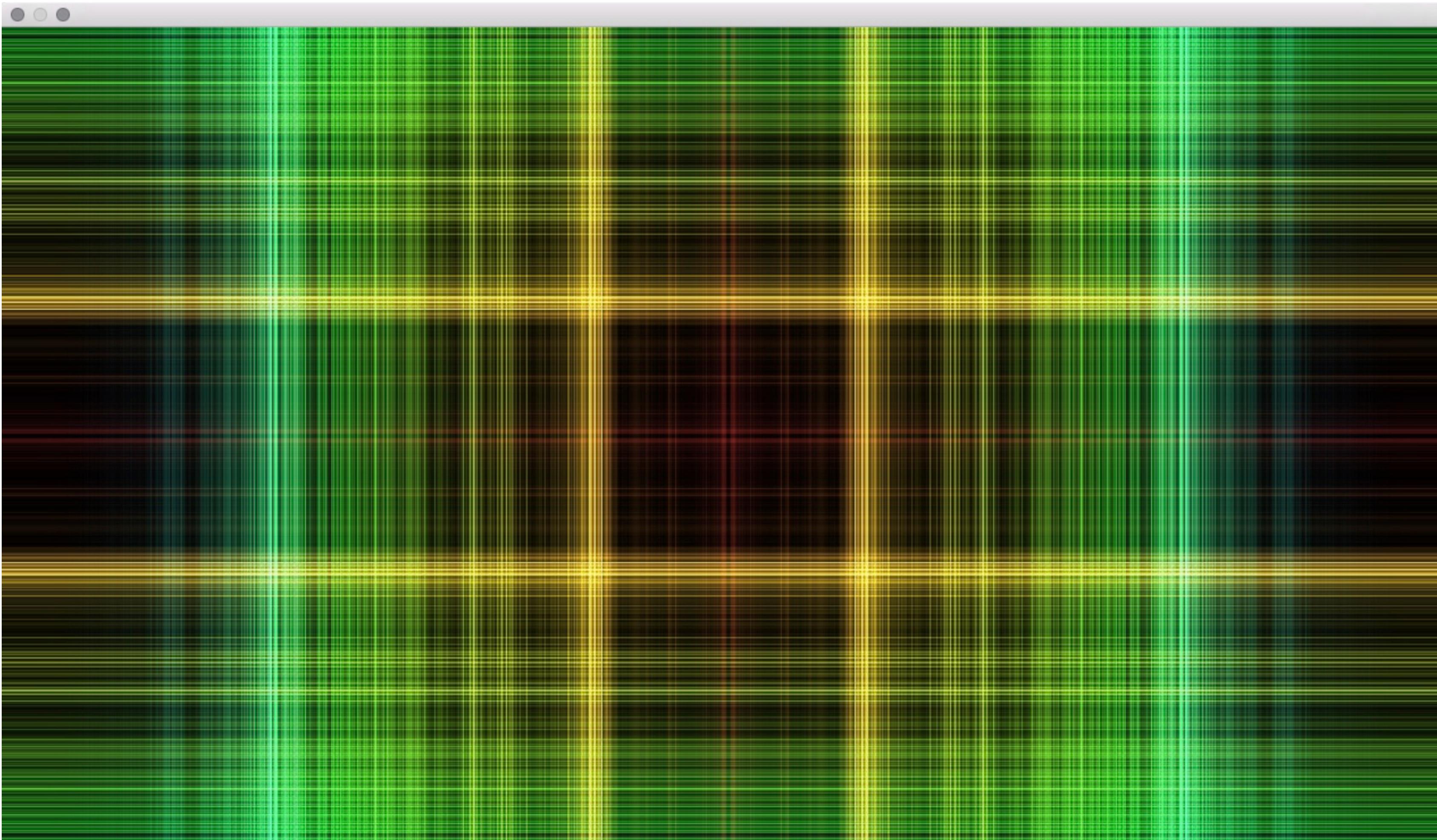
▶ ofApp.cpp

```
// 色を設定
float hue = ofMap(i, 0, buffer.size(), 0, 160);
float br = ofMap(buffer[i], 0, 1, 0, 255);
ofColor col;
col.setHsb(hue, saturation, br);
ofSetColor(col);
// 右
float rx = ofMap(i, 0, buffer.size(), 0, ofGetWidth() / 2.0);
ofLine(rx, -ofGetWidth() / 2.0, rx, ofGetWidth() / 2.0);
// 左
float lx = ofMap(i, 0, buffer.size(), 0, -ofGetWidth() / 2.0);
ofLine(lx, -ofGetWidth() / 2.0, lx, ofGetWidth() / 2.0);
}
ofRotateZ(90);
}
ofPopMatrix();

//GUI
gui.draw();
}
```

ofxFft

- ▶ 音を視覚化した格子模様



ofxFft

- ▶ では、色のバーではなく、サイズの変化する円で描くとどうなるか?

ofxFft

▶ ofApp.h

```
#pragma once

#include "ofMain.h"
#include "ofxEasyFft.h"
#include "ofxGui.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);

    ofxEasyFft fft;      // ofxEasyFftインスタンス
    ofxPanel gui;
    ofxFloatSlider alpha;
    ofxFloatSlider saturation;
};
```

ofxFft

▶ ofApp.cpp

```
#include "ofApp.h"

void ofApp::setup(){
    ofSetFrameRate(60);
    ofBackground(0);
    // FFTのサンプル数(2の累乗)を指定して初期化
    fft.setup(pow(2.0, 10.0));
    // GUI
    gui.setup();
    gui.add(saturation.setup("saturation", 127, 0, 255));
    gui.add(alpha.setup("alpha", 127, 0, 255));
}

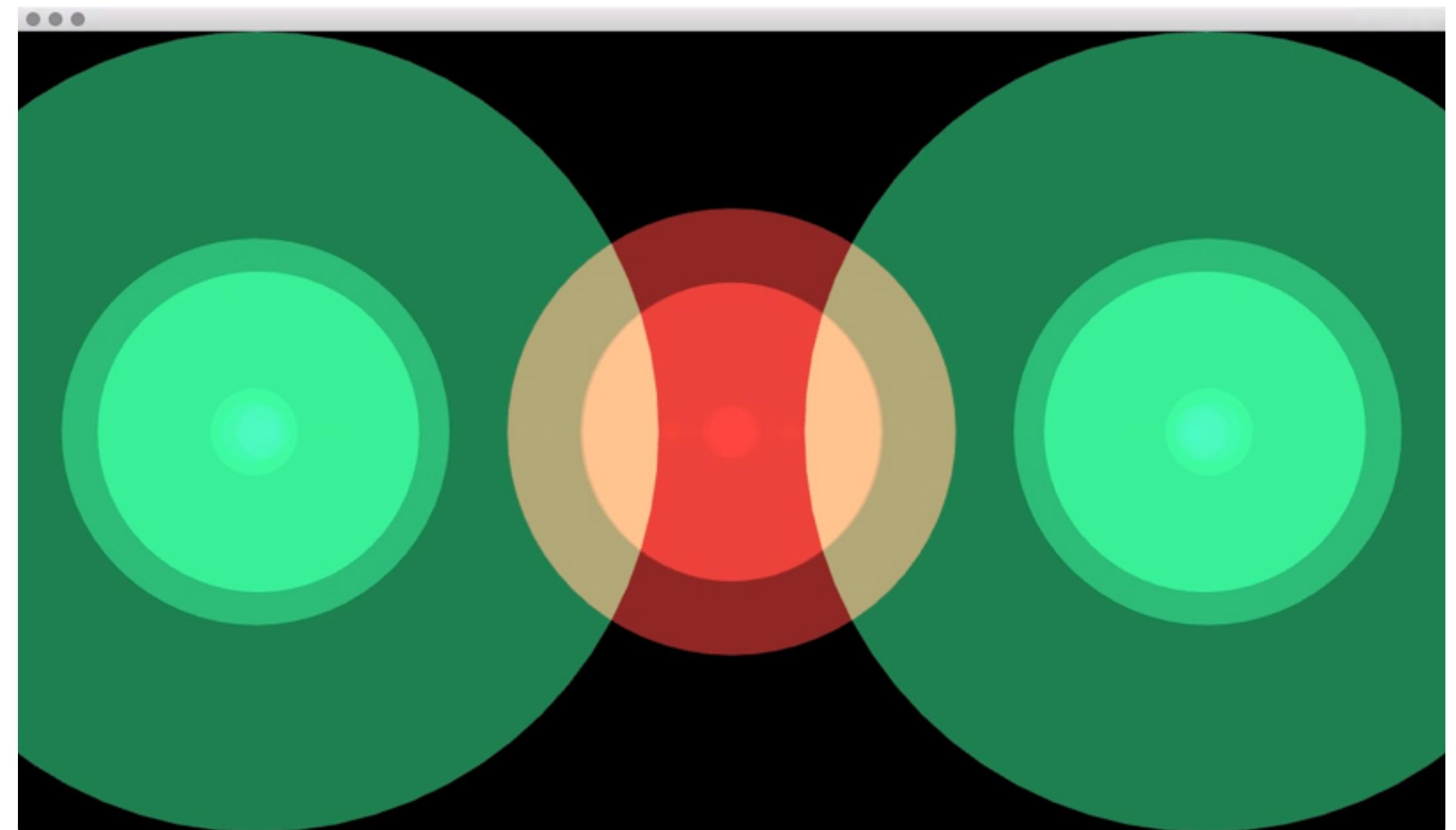
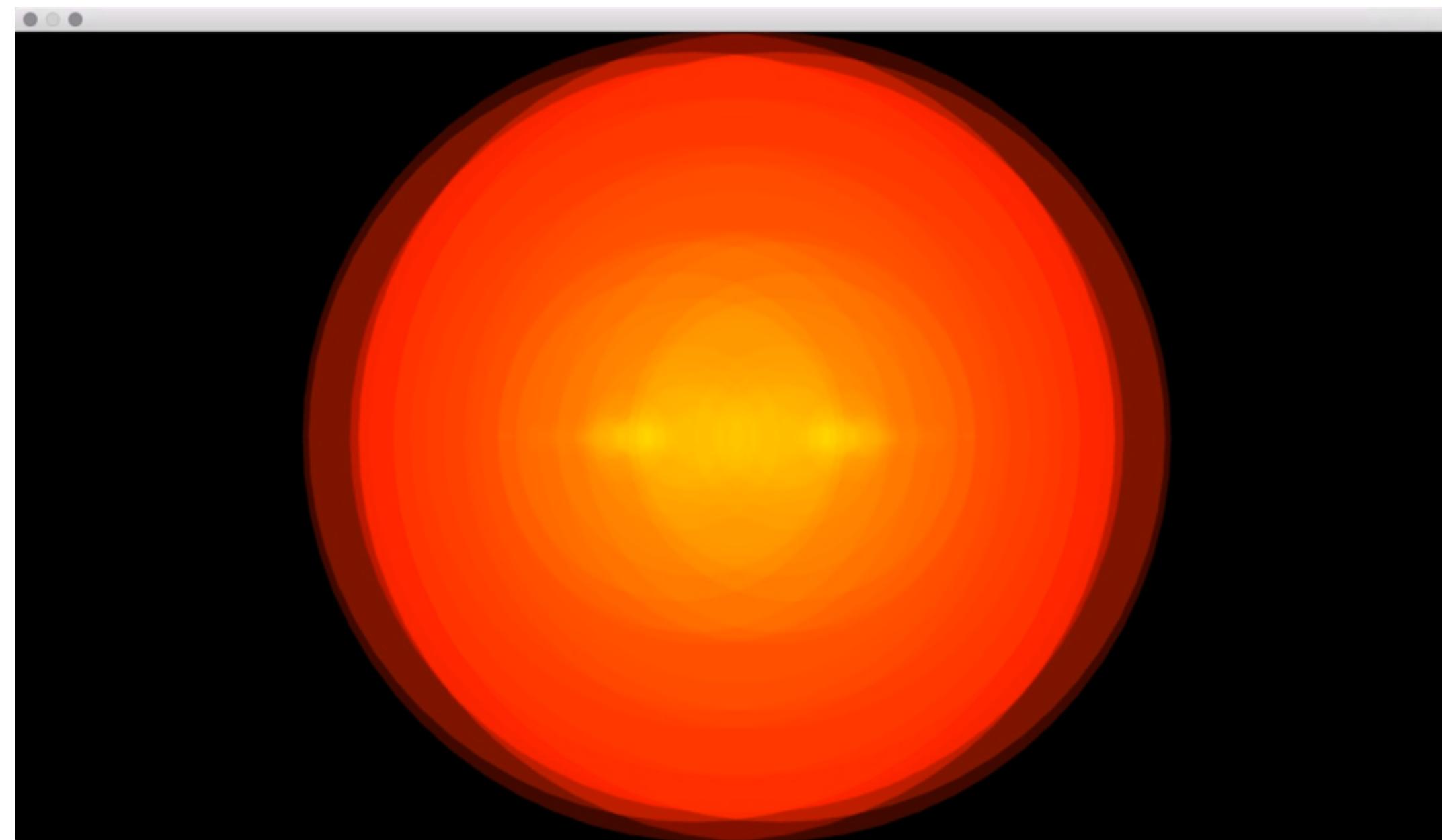
void ofApp::update(){
    fft.update(); // FFT更新
}

void ofApp::draw(){
    // float型の配列にFFT結果を格納
    vector<float> buffer;
    buffer = fft.getBins();

    // グラフに描画
    ofPushMatrix();
    ofTranslate(ofGetWidth()/2, ofGetHeight()/2);
    ofSetLineWidth(ofGetWidth() / float(buffer.size()) / 2.0);
    ofEnableBlendMode(OF_BLENDMODE_ADD);
    ofSetCircleResolution(64);
    for (int i = 0; i < buffer.size(); i++) {
        // 色を設定
        float hue = ofMap(i, 0, buffer.size(), 0, 160);
        float br = ofMap(buffer[i], 0, 1, 0, 255);
        float radius = ofMap(buffer[i], 0, 1, 0, ofGetHeight() / 2.0);
        ofColor col;
        col.setHsb(hue, saturation, br, alpha);
        ofSetColor(col);
        // 右
        float rx = ofMap(i, 0, buffer.size(), 0, ofGetWidth() / 2.0);
        ofCircle(rx, 0, radius);
        // 左
        float lx = ofMap(i, 0, buffer.size(), 0, -ofGetWidth() / 2.0);
        ofCircle(lx, 0, radius);
    }
    ofPopMatrix();
    gui.draw();
}
```

ofxFft

- ▶ 円の重ねあわせによる視覚化



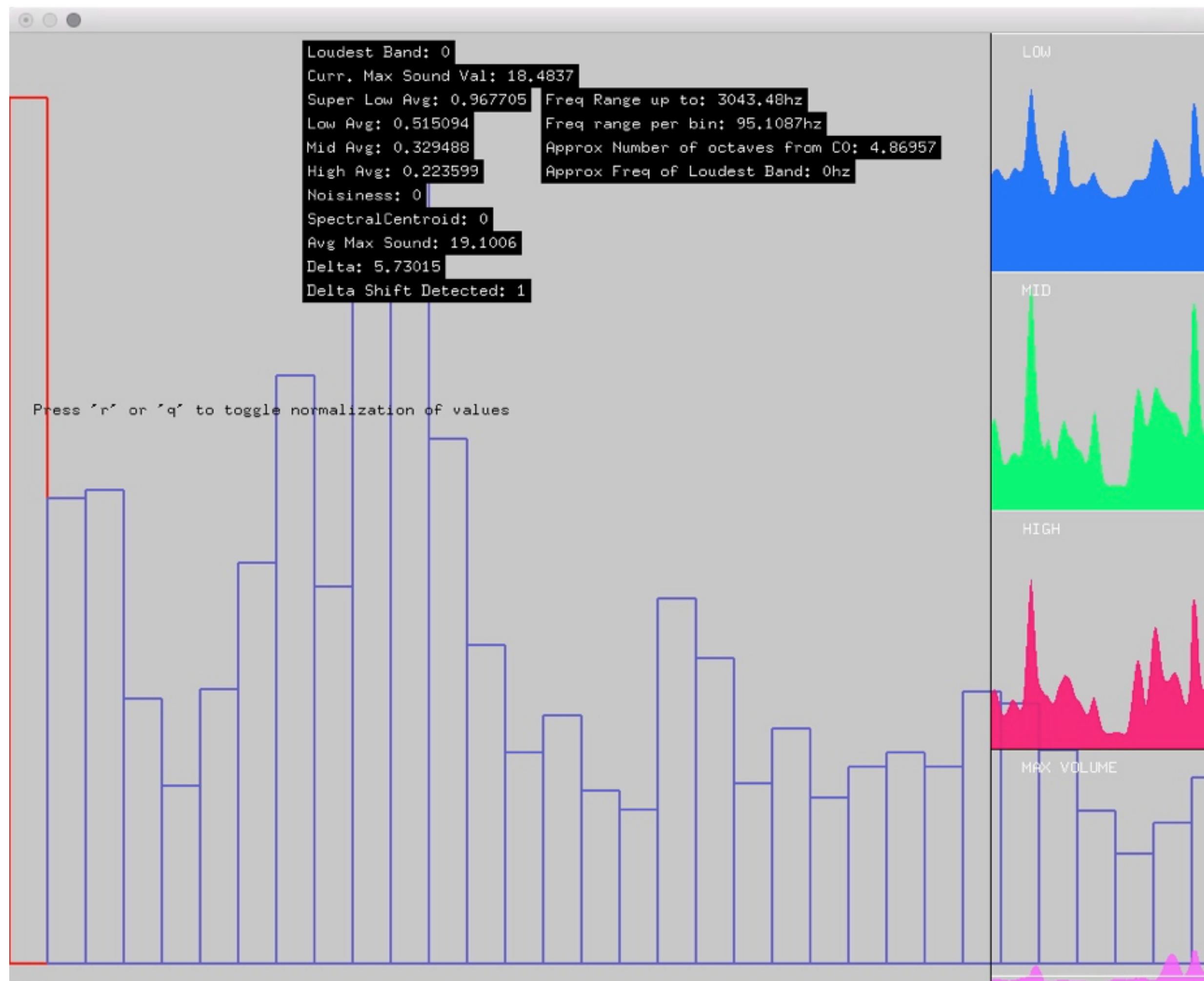
ofxProcessFFT

ofxProcessFFT

- ▶ ofxFftに付属しているofxProcessFFTクラスを使用すると、さらに詳細な分析が可能
 - ▶ 低域、中域、高域に分けた音圧レベル
 - ▶ 中心周波数
 - ▶ 時間変化の量
 - ▶ ...etc
-
- ▶ 詳細は、example-process-input を参照

ofxProcessFFT

▶ example-process-input 実行結果



ofxProcessFFT

- ▶ まずは、ofxProcessFFTをつかったシンプルな例から
- ▶ 低域、中域、高域の音圧レベルをそれぞれ取得
 - ▶ 低域: fft.getLowVal()
 - ▶ 中域: fft.getMidVal()
 - ▶ 高域: fft.getHighVal()
- ▶ 低域をRed、中域Green、高域をBlueの値にして、画面全体を塗ってみる

ofxProcessFFT

▶ ofApp.h

```
#pragma once

#include "ofMain.h"
#include "ofxProcessFFT.h"
#include "ofxGui.h"

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

    ProcessFFT fft;          // FFT分析
    ofxPanel gui;            // GUI
    ofxFloatingPanel level;  // 入力レベル調整
};
```

ofxProcessFFT

▶ ofApp.cpp

```
#include "ofApp.h"

void ofApp::setup() {
    ofSetVerticalSync(true);
    ofSetFrameRate(60);
    ofBackground(0);
    // FFT初期設定
    fft.setup();
    fft.setNumFFTBins(16);
    fft.setNormalize(true);
    // GUI
    gui.setup();
    gui.add(level.setup("Input Level", 500, 0, 5000));
    gui.loadFromFile("settings.xml");
}

void ofApp::update() {
    fft.update();
}

void ofApp::draw() {
    // 低域、中域、高域の値を取得
    float lowValue = ofMap(fft.getLowVal(), 0, 1, 0, level);
    float midValue = ofMap(fft.getMidVal(), 0, 1, 0, level);
    float highValue = ofMap(fft.getHighVal(), 0, 1, 0, level);
    // 画面全体を塗る
    ofSetColor(lowValue, midValue, highValue);
    ofRect(0, 0, ofGetWidth(), ofGetHeight());
    // GUI
    gui.draw();
}
```

ofxProcessFFT

- ▶ 周波数の成分によって塗られる色



ofxProcessFFT

- ▶ さらに工夫してみる
- ▶ 低域、中域、高域、それぞれのアニメーション画像をofNoise()で生成
- ▶ それを、RGBにわりふって、ノイズ画像を加算合成

ofxProcessFFT

▶ ofApp.h

```
#pragma once

#include "ofMain.h"
#include "ofxProcessFFT.h"
#include "ofxGui.h"

class ofApp : public ofBaseApp {
public:
    void setup();
    void update();
    void draw();
    void resolutionChanged(int & resolution);

    ProcessFFT fft;           // FFT分析
    ofImage lowNoiseImg;     // ノイズ画像低域
    ofImage midNoiseImg;     // ノイズ画像中域
    ofImage highNoiseImg;    // ノイズ画像高域

    ofxPanel gui;            // GUI
    ofxFLOATSlider level;    // 入力レベル調整
    ofxIntSlider resolution; // ノイズ解像度
    ofxFLOATSlider noiseFrequency; // ノイズ周波数
};
```

ofxProcessFFT

▶ ofApp.cpp

```
#include "ofApp.h"

void ofApp::setup() {
    ofSetVerticalSync(true);
    ofSetFrameRate(60);
    ofBackground(0);
    // FFT初期設定
    fft.setup();
    fft.setNumFTTBins(16);
    fft.setNormalize(true);
    // GUI
    resolution.addListener(this, &ofApp::resolutionChanged);
    gui.setup();
    gui.add(level.setup("Input Level", 100, 0, 2000));
    gui.add(resolution.setup("Resolution", 256, 2, 1024));
    gui.add(noiseFrequency.setup("Noise Frequency", 80.0, 0.0001, 1024.0));
    gui.loadFromFile("settings.xml");
    // 画像初期化
    lowNoiseImg.allocate(resolution, resolution, OF_IMAGE_GRAYSCALE);
    midNoiseImg.allocate(resolution, resolution, OF_IMAGE_GRAYSCALE);
    highNoiseImg.allocate(resolution, resolution, OF_IMAGE_GRAYSCALE);
}

void ofApp::update() {
    fft.update();
    // 低域、中域、高域の値を取得
    float lowValue = ofMap(fft.getLowVal(), 0, 1, 0, level);
    float midValue = ofMap(fft.getMidVal(), 0, 1, 0, level);
    float highValue = ofMap(fft.getHighVal(), 0, 1, 0, level);
```

ofxProcessFFT

▶ ofApp.cpp

```
// ノイズ生成
int tmpIndex;
// 低域
ofPixelsRef lowPixels = lowNoiseImg.getPixelsRef();
tmpIndex = 0;
for( int y = 0; y < lowNoiseImg.getHeight(); y++ ){
    for( int x = 0; x < lowNoiseImg.getWidth(); x++ ){
        float tmpNoise = ofNoise(x / noiseFrequency, y / noiseFrequency, ofGetElapsedTimef());
        lowPixels[tmpIndex] = tmpNoise * lowValue;
        tmpIndex++;
    }
}
lowNoiseImg.update();
// 中域
ofPixelsRef midPixels = midNoiseImg.getPixelsRef();
tmpIndex = 0;
for( int y = 0; y < midNoiseImg.getHeight(); y++ ){
    for( int x = 0; x < midNoiseImg.getWidth(); x++ ){
        float tmpNoise = ofNoise(x / noiseFrequency * 2.0, y / noiseFrequency * 2.0, ofGetElapsedTimef() * 2.0);
        midPixels[tmpIndex] = tmpNoise * midValue;
        tmpIndex++;
    }
}
midNoiseImg.update();
// 高域
ofPixelsRef highPixels = highNoiseImg.getPixelsRef();
tmpIndex = 0;
for( int y = 0; y < highNoiseImg.getHeight(); y++ ){
    for( int x = 0; x < highNoiseImg.getWidth(); x++ ){
```

ofxProcessFFT

▶ ofApp.cpp

```
        float tmpNoise = ofNoise(x / noiseFrequency * 4.0, y / noiseFrequency * 4.0, ofGetElapsedTimef() * 4.0);
        highPixels[tmpIndex] = tmpNoise * highValue;
        tmpIndex++;
    }
}

highNoiseImg.update();
}

void ofApp::draw() {

    // ノイズ描画
    ofEnableBlendMode(OF_BLENDMODE_ADD);
    ofSetColor(255, 0, 0);
    lowNoiseImg.draw(0, 0, ofGetWidth(), ofGetHeight());
    ofSetColor(0, 255, 0);
    midNoiseImg.draw(0, 0, ofGetWidth(), ofGetHeight());
    ofSetColor(0, 0, 255);
    highNoiseImg.draw(0, 0, ofGetWidth(), ofGetHeight());

    // GUI
    gui.draw();
}

void ofApp::resolutionChanged(int & resolution){
    lowNoiseImg.allocate(resolution, resolution, OF_IMAGE_GRAYSCALE);
    midNoiseImg.allocate(resolution, resolution, OF_IMAGE_GRAYSCALE);
    highNoiseImg.allocate(resolution, resolution, OF_IMAGE_GRAYSCALE);
}
```

ofxProcessFFT

- ▶ 完成!! 音とともに変化する複雑な色彩のテクスチャー

