

[https://github.com  
/FMS-Cat/20180310-  
glsl-music](https://github.com/FMS-Cat/20180310-glsl-music)

# GLSLで音楽を作ります

## FMS\_Cat

メガデモ勉強会 2018/3/10

<https://github.com/FMS-Cat/20180310-glsl-music>

# FMS\_Cat

猫

- Shell  
GLSL Graphics Compo, TDF 2015
- [Type]  
Combined Demo Compo, TDF 2016
- Shift  
Combined Demo Compo, TDF 2017



今回、実質GLSL勉強会  
じゃないです  
か！！！！！！

ポリゴン知見をくれ～～



海  
水  
素  
客

やだ――！



はじめに

# GLSLで音楽を作ります

- GLSL作曲環境の構築
- ドラムおよびシンセの作成
- リズムとメロディの生成

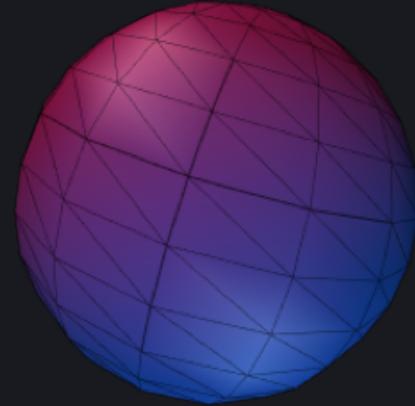
# はじめに

## GLSLについて

OpenGL上で扱われる  
プログラマブルシェーダ記述用言語

OpenGLではそのままでも自動的に  
Shading（陰影付け）が行われるが  
プログラマブルシェーダを使うと  
自由自在なShadingが実現可能

WebGLでは描画にプログラマブルシェーダの  
記述が必須とされている

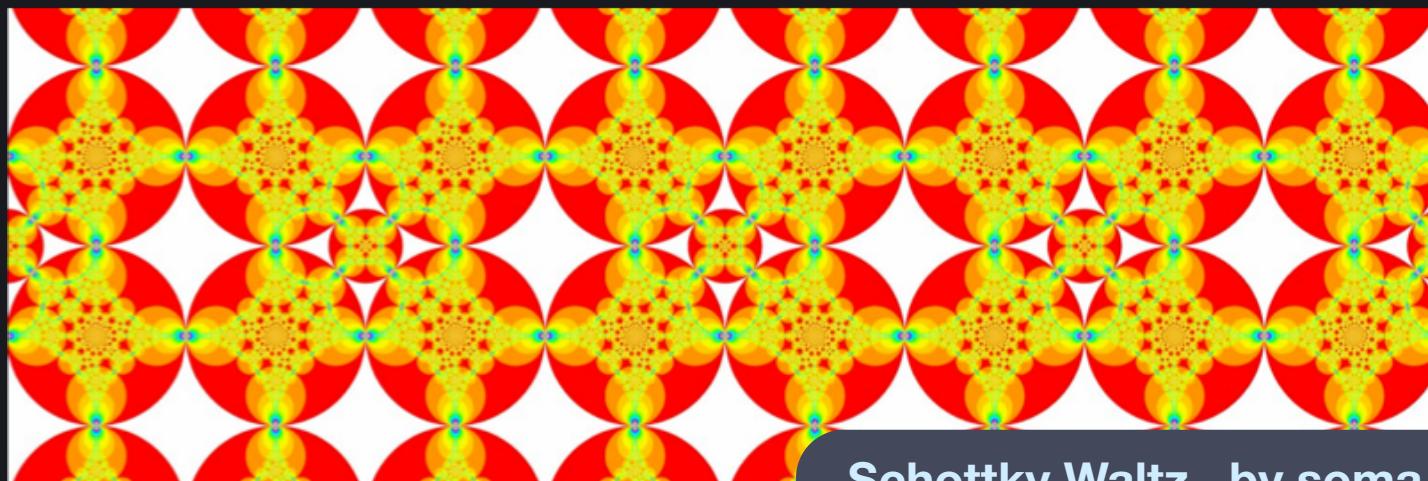


# はじめに

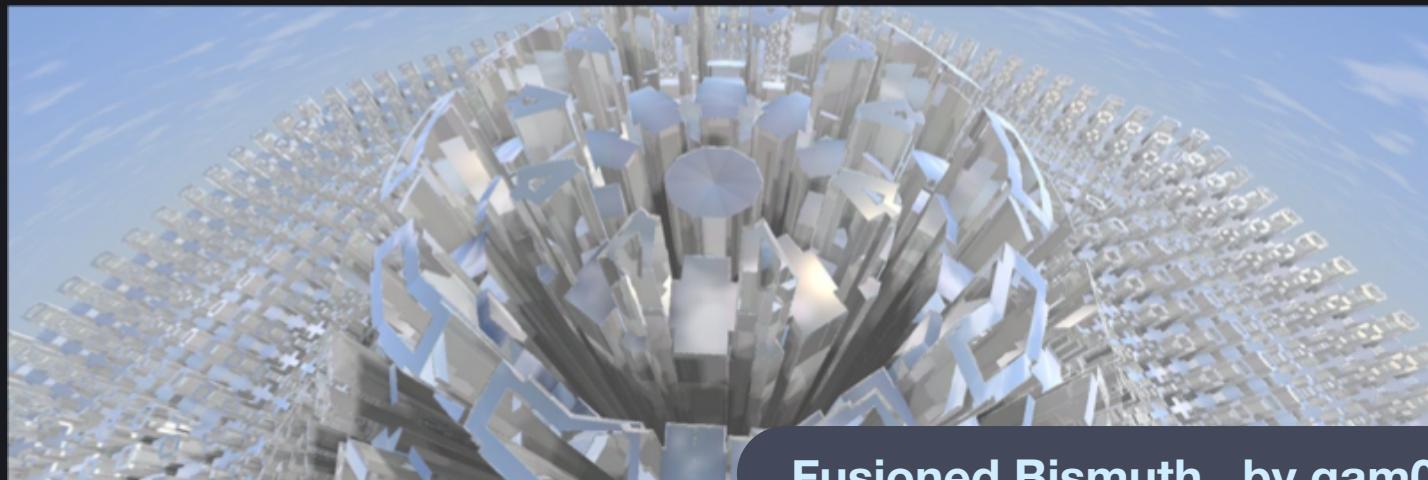


<http://glsandbox.com/e#743.2>  
by @301z

# はじめに



Schottky Waltz by soma\_arc



Fusioned Bismuth by gam0022

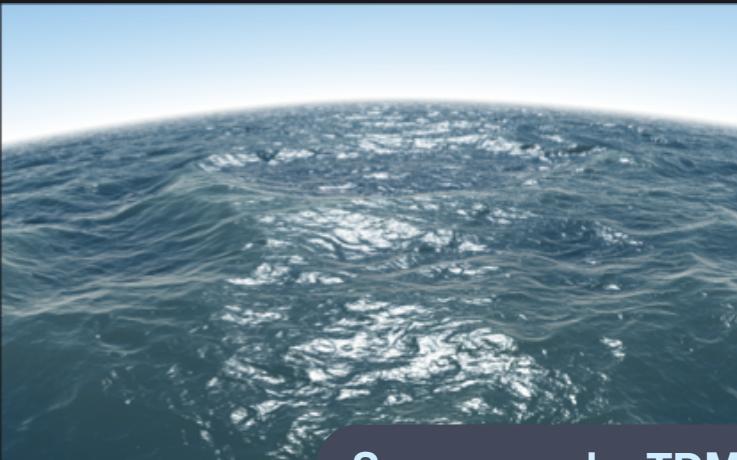
# はじめに



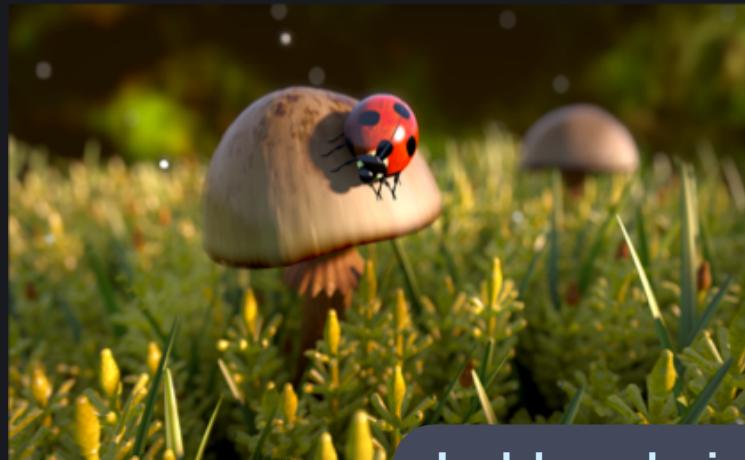
Elevated by rgba & tbc



Final Stage by 0x4015



Seascape by TDM



Ladybug by iq

はじめに

三角形2枚の上に  
陰影付けのための言語で  
お絵かきをしているという  
アイデア自体がバカ

tbh: GPUの持つ強大な計算能力をフルに活かすには  
この方法が最適なケースがめちゃくちゃ存在する

# はじめに

readPixels()



Audio API

(超横長の三角形2枚)  
(音声波形を描いた)



# はじめに

## Why?

- **速い**  
シェーダの並列演算能力を  
フルに活かして信号生成が可能
- **軽い**  
4k-64kの厳しい容量制限に  
耐えうるコードの小ささ
- **たのしい**  
重要

# はじめに

## Other methods

### 1, ネイティブで作る

特にWebの場合、**Web Audio API**が  
かなり少ないコード量で扱える  
**GLSL**と組み合わせて使うのもアリ

### 2, 4klang

**4k intro**のために作られたソフトシンセ  
0x4015氏曰く、使い方によっては  
**2KB (半分)** を占有するらしい

# 作曲環境

- 横長の三角形2枚  
OpenGLのチュートリアル程度の物量
- 「音声波形」を描画するシェーダ  
めっちゃコード書く
- 音を色で表現するためのフォーマット  
LRの音声サンプル ⇄ rgbaの色情報
- 描画した色情報の読み取り  
WebGLの場合、`gl.readPixels()`
- 音声サンプルの再生  
WebGLの場合、Web Audio APIとの闘い

## 作曲環境

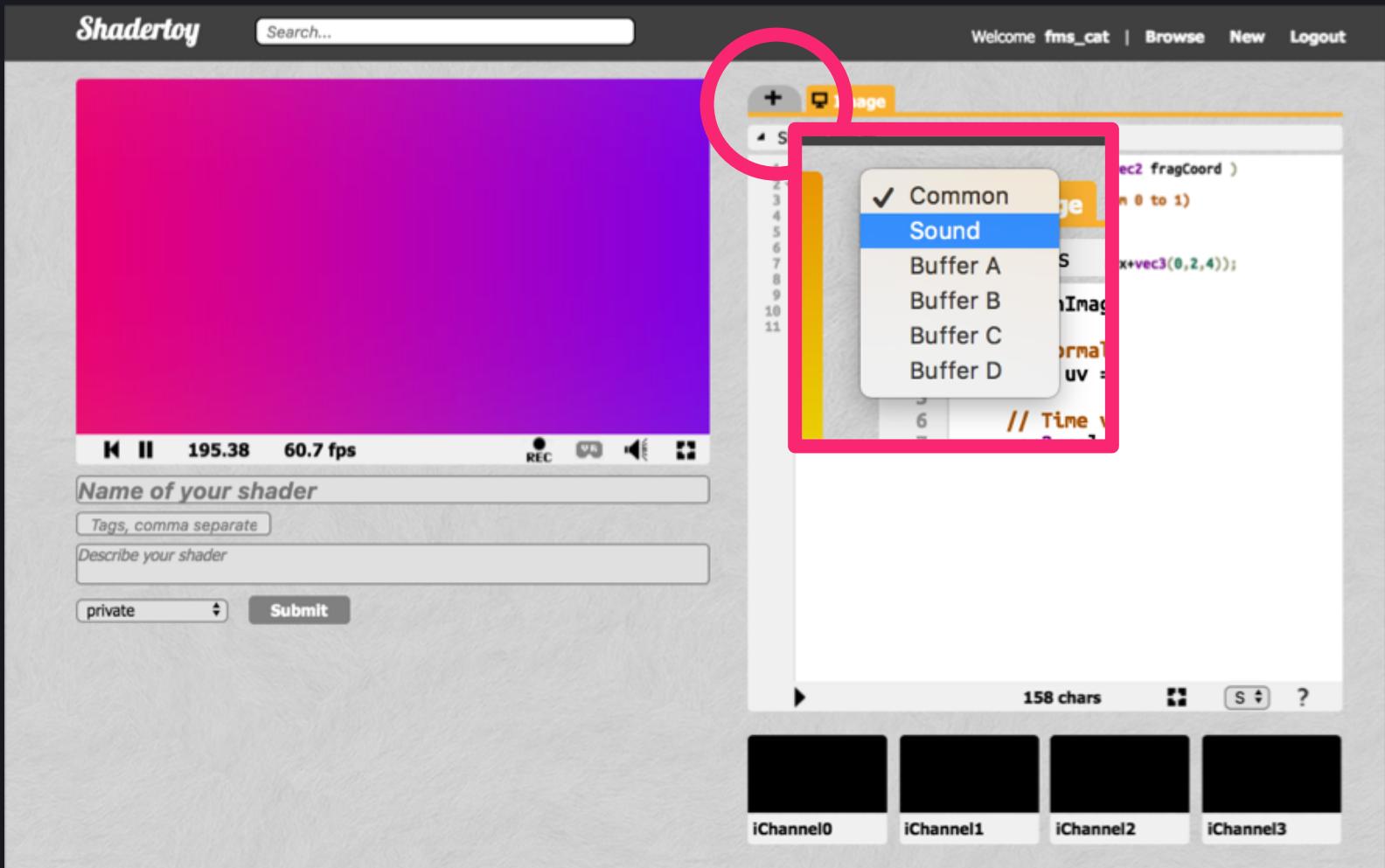
今回は.....

*Shadertoy*

を使いましょう

[shadertoy.com/new](https://shadertoy.com/new)

# 作曲環境



# 作曲環境

## Hello World!

```
vec2 mainSound( float time )  
{  
    // A 440 Hz wave that attenuates quickly over time  
    return vec2( sin( 6.2831 * 440.0 * time ) * exp( -3.0 * time ) );  
}
```



ステレオ

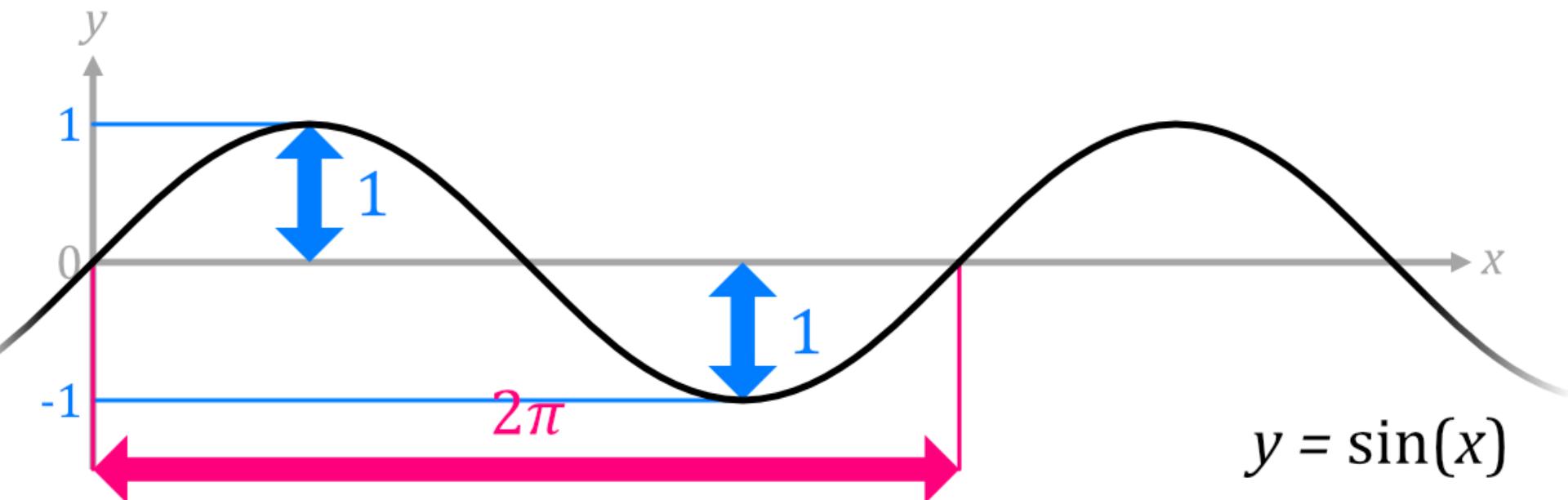
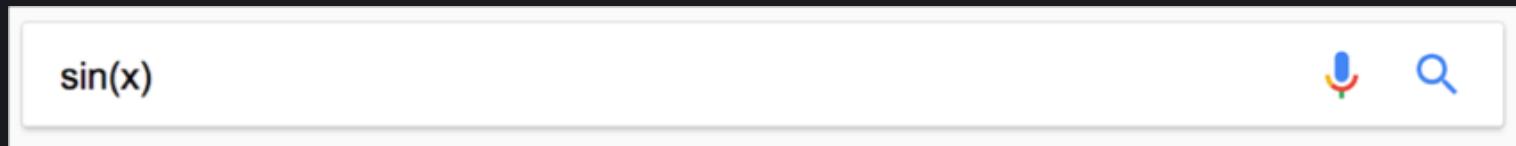
サイン波

自然減衰

各サンプルごとに  
mainSoundが実行される

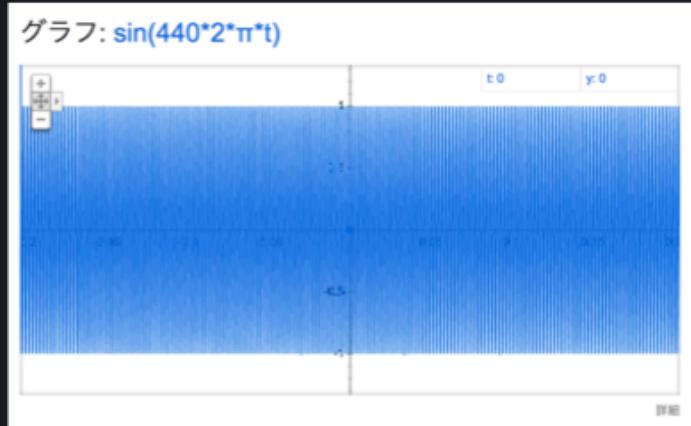
# 作曲環境

## サイン波



# 作曲環境

## サイン波



```
vec2 mainSound( float time )  
{  
    // A 440 Hz wave that attenuates quickly over time  
    return vec2( sin( 6.2831 * 440.0 * time ) ); exp( -3.0 * time ) );  
}
```

$$\frac{2\pi}{f} \frac{1}{t}$$

$2\pi$  : サイン波の1周期

$f$  : 周波数 [Hz] (440.0Hzは”ラ”)

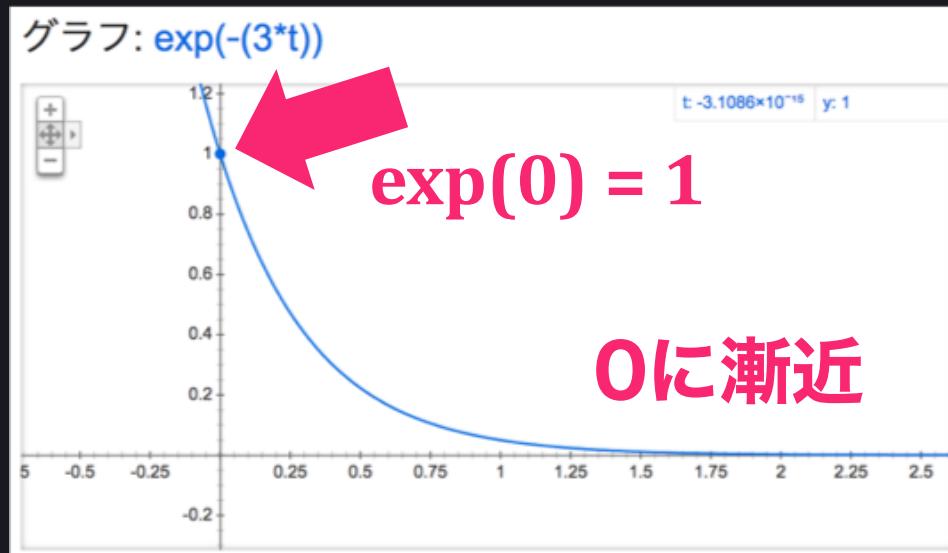
$t$  : 時間 [s] (音波は時間の関数)

# 作曲環境

## 自然減衰

## 掛け算

```
vec2 mainSound( float time )  
{  
    // A 440 Hz wave that attenuates quickly over time  
    return vec2( sin( 6.2831 * 440.0 * time ) * exp( -3.0 * time ) );  
}
```



# 作曲環境

## ステレオ

```
vec2 mainSound( float time )  
{  
    return vec2( sin( 6.2831 * 440.0 * time ), 0.0 );  
}  
                                左      右
```

```
vec2 mainSound( float time )  
{  
    return vec2( 0.0, sin( 6.2831 * 440.0 * time ) );  
}  
                                左      右
```

# 音色

## 楽音の三要素

- 音量  
音の大きさ ⇒ 振幅
- 音階  
音の高さ ⇒ 周波数
- 音色  
それ以外の特徴 ⇒ 波形

# 音色

波にはいろいろな形がある

代表的な基本波形

- ノコギリ波
- 矩形波
- 三角波
- サイン波



# 音色

## 基本波形の作り方

```
float saw( float phase ) {
    return 2.0 * fract( phase ) - 1.0;
}

float square( float phase ) {
    return fract( phase ) < 0.5 ? -1.0 : 1.0;
}

float triangle( float phase ) {
    return 1.0 - 4.0 * abs( fract( phase ) - 0.5 );
}

float sine( float phase ) {
    return sin( TAU * phase );
}
```

# 音色

音色づくりの沼は深い.....

FM変調

ユニゾン

リング変調

ハードシンク トレモロ/ビブラート

PWM

フィルター

エンベロープ

# 音色

今回はShiftを使った技法を  
少し解説します

# 音色

## Shiftの楽器編成

(後半の上ネタ)

Arp

Pad

Bass

Drums

Kick  
Snare  
Hihat  
Click  
Tom

# 音色

## Bass

⇒ こだわるわけでなければ  
かなり単調に作っても大丈夫  
(e.g. タダの矩形波・サイン波)

Shiftの場合.....

矩形波 \* 0.1 + サイン波 \* 0.7

※1 矩形波の比率が6:4

※2 位相にノイズを少し足す

# 音色

## Pad

ビブラート：  
波の位相を揺らす

```
float vib = 0.2 * sine( time * 5.0 );
return saw( 440.0 * time + vib );
```

## 音色

Arp

FM変調: ≈ ビブラート  
波の位相を波でいじる

freqのN倍

```
float freq = 440.0;  
float fm = 0.1 * sine( time * freq * 7.0 );  
return sine( freq * time + fm );
```

fmを自然減衰させるといい感じになる

# 音色

その他のシンセ群は  
既存のシンセのプリセットを  
コードに落とし込んだりしてます



Load preset



```
// fn bass
vec2 bass( float _freq, float _phase ) {
    if ( _phase < 0.0 ) { return V_00; }
    vec2 p = _freq * vec2( 0.999, 1.001 ) * _phase * TRAU;
    float dl = exp( -_phase * 1.0 );
    float ds = exp( -_phase * 14.0 );
    return sin(
        p +
        sin( p ) * 1.5 * dl +
        sin( p + sin( p * 10.0 ) * 2.5 * ds ) * 3.0 * ds +
        sin( p + sin( p * 10.0 ) * 1.5 * ds ) * 0.5 * dl
    );
}
```

Implement

## 音色

## Drums

音階と音量を自然減衰

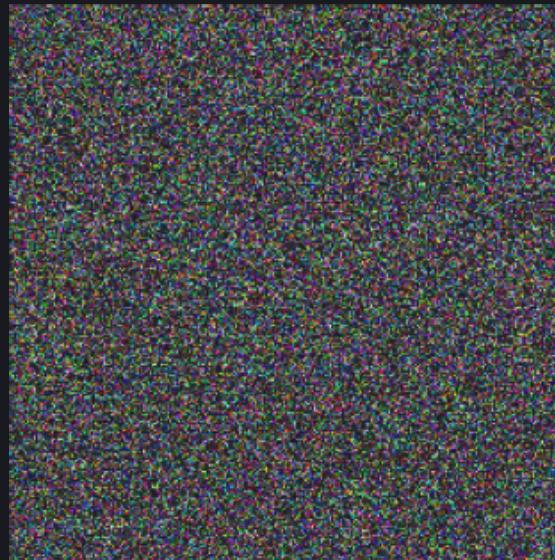
```
float kick( float time ) {  
    float amp = exp( -5.0 * time );  
    float phase = 50.0 * time  
        - 10.0 * exp( -70.0 * time );  
    return amp * sine( phase );  
}
```

# 音色

Drums

ノイズを使う

⇒ ノイズテクスチャを使おう



# 音色

Shader Inputs

```
1 #define PI 3.141592654
2 #define TAU 6.283185307
3
4 // -----
5 float saw( float phase ) {
6     return 2.0 * fract( phase ) - 1.0;
7 }
8
9 float square( float phase ) {
10    return fract( phase ) < 0.5 ? -1.0 : 1.0;
11 }
12
13 float triangle( float phase ) {
14    return 1.0 - 4.0 * abs( fract( phase ) - 0.5 );
15 }
16
17 float sine( float phase ) {
18    return sin( TAU * phase );
19 }
20
21 // -----
22
23 float kick( float time ) {
24    float amp = exp( -5.0 * time );
25    float phase = 50.0 * time - 10.0 * exp( -70.0 * time );
26    return amp * sine( phase );
27 }
28
29 // -----
30
31 vec2 mainSound( float time ) {
32    return vec2( kick( time ) );
33 }
34 }
```

Filter linear

Wrap repeat

VFlip

iChannel0 

1024 x 1024  
3 channels, uint8

1024 x 1024  
3 channels, uint8

512 x 512  
1 channels, uint8

RGB Noise Medium by shadertoy  
256 x 256  
4 channels, uint8

RGB Noise Small by shadertoy  
64 x 64  
4 channels, uint8

Rock Tiles by shadertoy  
512 x 512  
3 channels, uint8

1 2 3 (21)

# 音色

## Drums

### ノイズを使う

1次元のphaseを  
2次元の座標uvにする

```
vec4 noise( float phase ) {  
    vec2 uv = phase / vec2( 0.512, 0.487 );  
    return 2.0 * texture( iChannel0, uv ) - 1.0;  
}
```

0 ~ 1の返り値を-1 ~ 1に

## 音色

## Drums

Hihatをノイズで作る

```
vec2 hihat( float time ) {
    float amp = exp( -50.0 * time );
    return amp * noise( time * 100.0 ).xy;
}
```

左右で微妙に違う信号を流すと  
音が左右に広がる

# 音色

## Drums

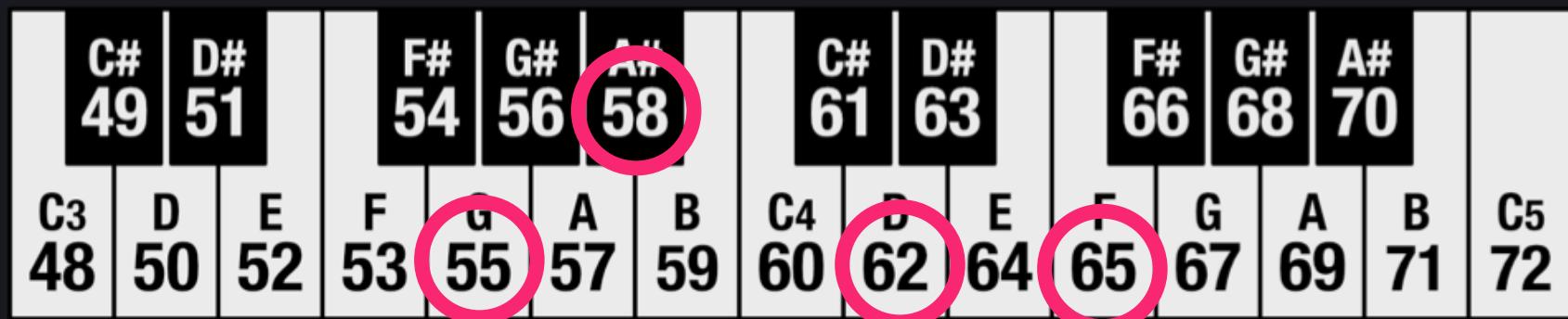
Snareは難しい

Method 1 (ドラムマシン的)：  
ノイズとサイン波を組み合わせる

Method 2 (チップチューン的)：  
ノイズのピッチ変化だけで再現

リズム・メロディ

最初の一歩:  
コード (もしくはスケール)  
を決める



Gm7 (Gマイナーセブン)

リズム・メロディ

今回はGm7を使ってみよう

マイナーセブン使えばとりあえず  
カッコよくなるみたいな部分ある

リズム・メロディ

MIDIで使われてる

ノート番号を周波数にする

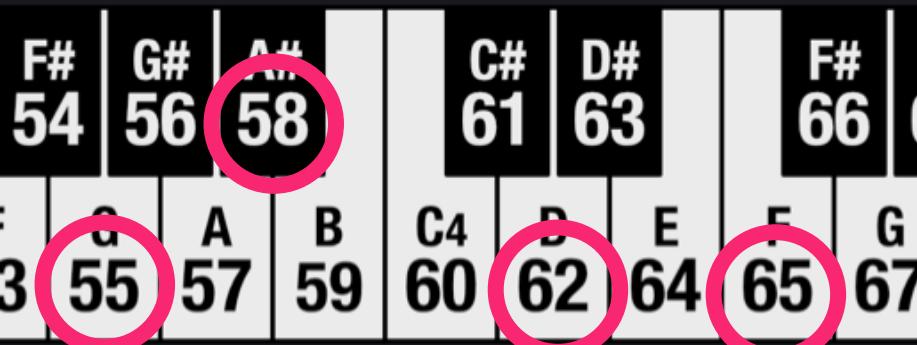
69番をA4 (440.0Hz)

としたときの鍵盤の位置

```
float noteToFreq( float n ) {  
    return 440.0  
        * pow( 2.0, ( n - 69.0 ) / 12.0 );  
}
```

## リズム・メロディ

# コードの構成音の ノート番号を返す関数を作ろう



```
float chord( float n ) {  
    return (  
        n < 1.0 ? 55.0 :  
        n < 2.0 ? 58.0 :  
        n < 3.0 ? 62.0 :  
                  65.0  
    );  
}
```

# リズム・メロディ

## Pad

```
vec2 mainSound( float time ) {  
    return vec2(  
        saw( time * noteToFreq( chord( 0.0 ) ) )  
        + saw( time * noteToFreq( chord( 1.0 ) ) )  
        + saw( time * noteToFreq( chord( 2.0 ) ) )  
        + saw( time * noteToFreq( chord( 3.0 ) ) )  
    ) / 4.0;  
}
```



音は加算で和音になる

# リズム・メロディ

## リズムを作ろう

### 1, テンポの定義

```
#define BPM 140.0
```

### 2, beat ⇔ time

```
float timeToBeat( float t )
{ return t / 60.0 * BPM; }

float beatToTime( float b )
{ return b / BPM * 60.0; }
```

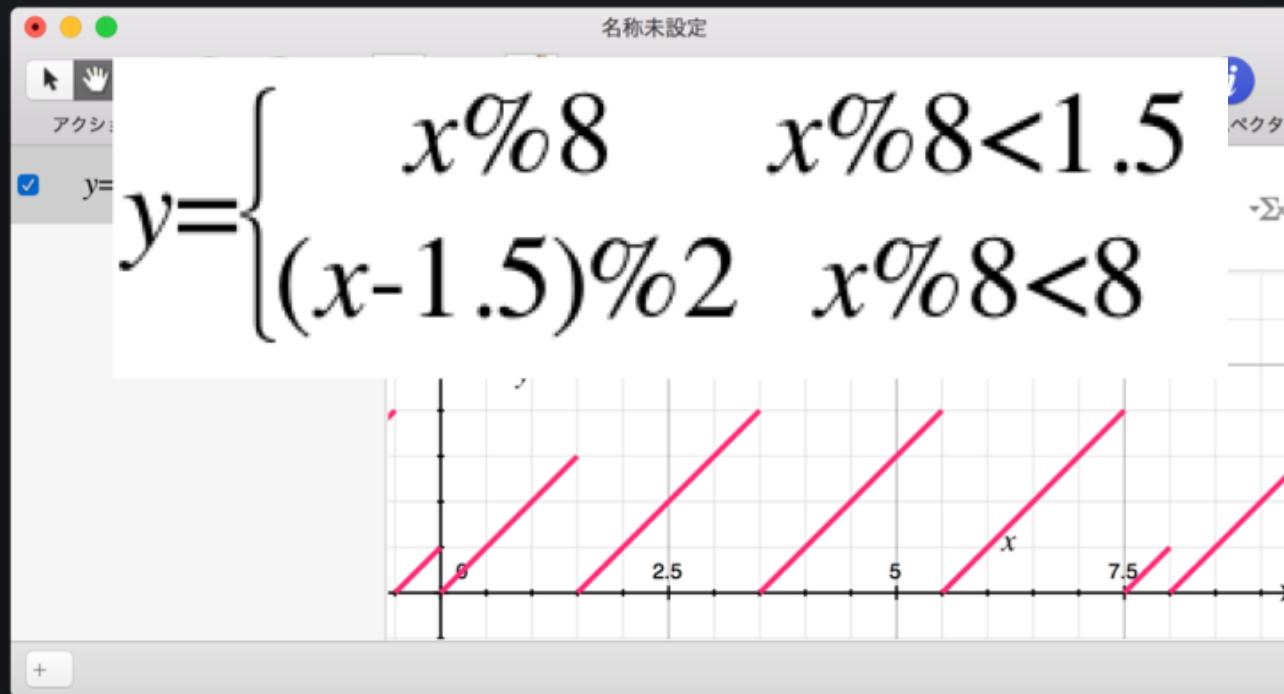
# リズム・メロディ

## 3, mod関数でリズムを作る

```
float beat = timeToBeat( time );
float kickTime = beatToTime( mod( beat, 1.0 ) );
return vec2( kick( kickTime ) );
```

“1ビートごとに繰り返し”

# リズム・メロディ



[ Type ] の キック の リズム

## リズム・メロディ

### Sidechain

kickのリズムに合わせて  
他のパートの音量を下げる

```
float beat = timeToBeat( time );
float kickTime = beatToTime( mod( beat, 1.0 ) );
return vec2( kick( kickTime ) );
```

```
float sidechain =
smoothstep( 0.0, 0.2, kickTime );
```

リズム・メロディ

Arp

- 1, コードの構成音から1音を  
ランダムに選択
- 2, ランダムに0~2オクターブずらす

12個隣の鍵盤



この「ランダム」がなかなか厄介

リズム・メロディ

Arp

適当に音階をランダムにすると.....

サンプル毎にランダム  
になってしまふ

やりたいのはノート毎にランダム

## リズム・メロディ

# Arp

```
float arpTime = beatToTime( mod( beat, 0.25 ) );  
float arpSeed = floor( beat / 0.25 );
```

ノート毎にランダムのシードを取る

```
vec4 arpDice = fract( noise( arpSeed ) * 100.0 );
```

値域の修正 &  
noiseの値をそのまま使うと分布が偏る

# まとめ

とりあえずこれだけでも.....

簡単な音楽が作れました



# まとめ

Further:

コード進行

疑似ディレイ

ベロシティ

展開付け

疑似LPF

グリッヂ系FX

サンプルのロード

マルチパスポートFX

# Thank you for listening

## GLSLで音楽を作ります

FMS\_Cat



メガデモ勉強会

2018/3/10