

FMS_Cat

レイマーチングは疲れるから今日はや
りません

GLSL TOPで 画像を壊します

@FMS_Cat

2018-05-20, SPEKTRA Vol.4.0.0

概要

TouchDesignerすごい
なんでもできるじゃん
GLSLいらないじゃん

概要



終わりー

概要

- TouchDesignerの機能を用いて効率的にGLSLで遊びます
 - GLSLを用いて画像からデータを取り出します
 - 画像から取り出したデータを用いて画像をめちゃくちゃにします
- ポストエフェクト, amagi先生の続きです

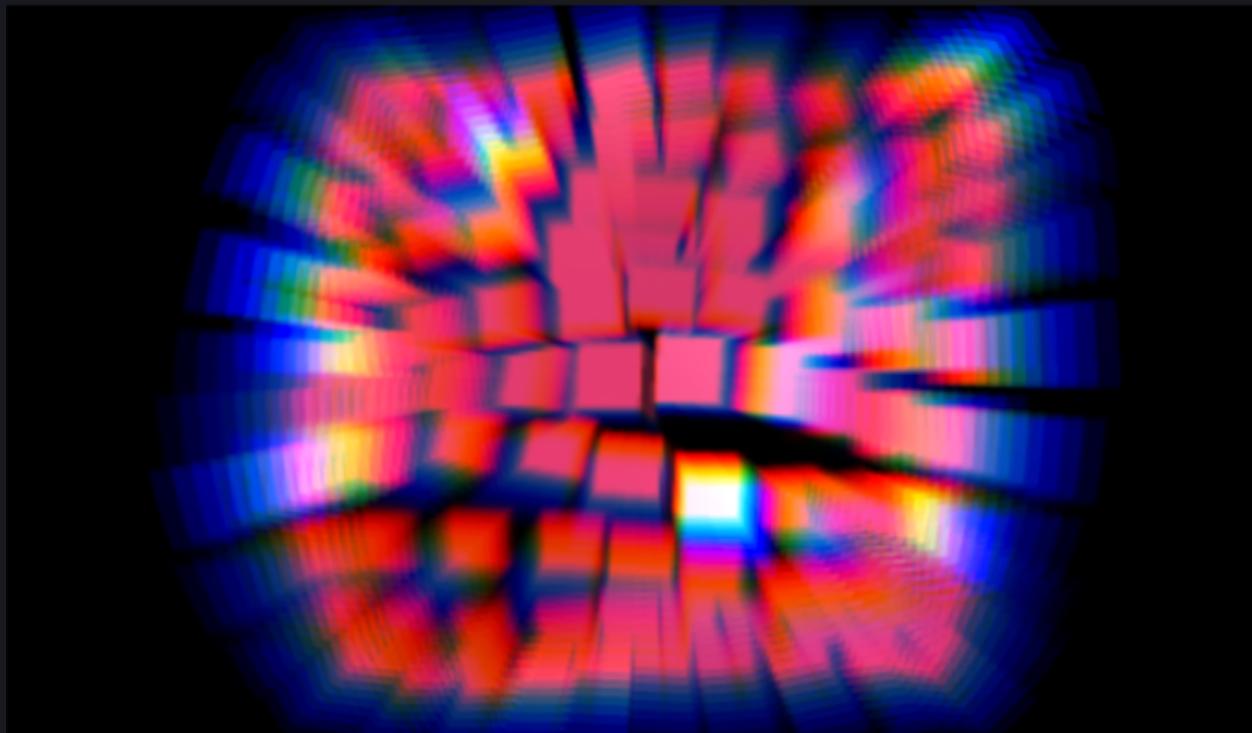
概要

なぜGLSL?

- 柔軟・高機能
OpenGLの優れた計算能力をフル活用
- コンパクト
複雑な処理を簡潔に記述できる
- 汎用
GLSLは他の環境でも活用可能

概要

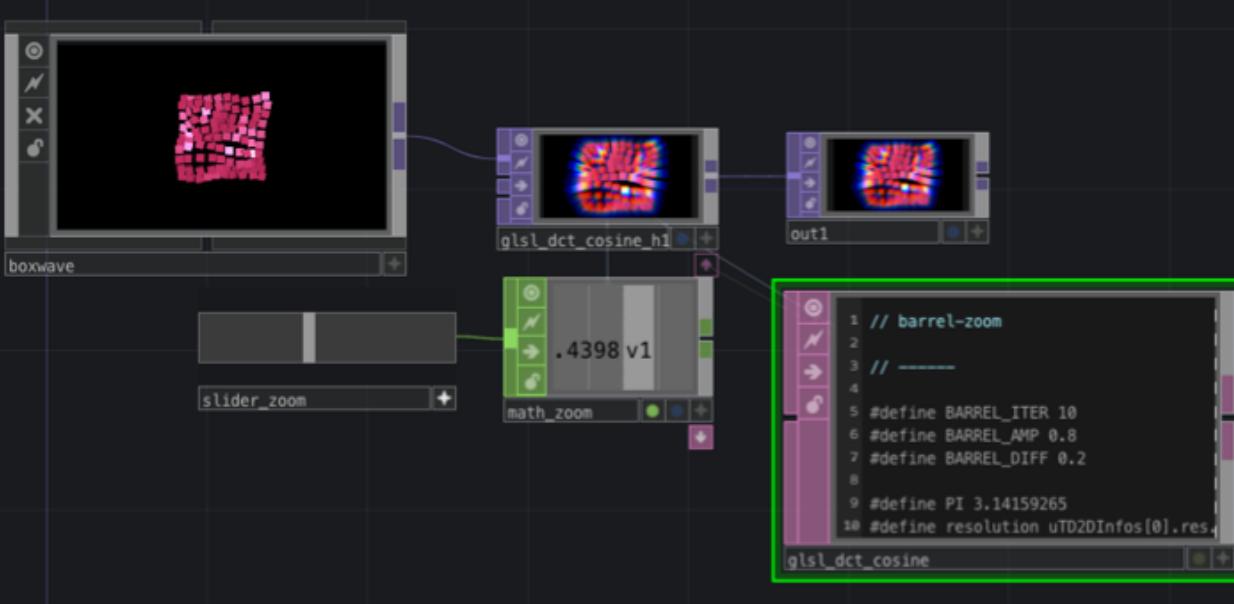
- 柔軟・高機能 / コンパクト



魚眼ズーム
(Barrel Distortion)
+
ズーム率を変えながら
色差ブラー

概要

- 柔軟・高機能 / コンパクト



(50行)

```
1 // barrel-zoom
2
3 // -----
4
5 #define BARREL_ITER 10
6 #define BARREL_AMP 0.8
7 #define BARREL_DIFF 0.2
8
9 #define PI 3.14159265
10 #define resolution uTD2DInfos[0].res
11 #define saturate(i) clamp(i, 0.1)
12
13 // -----
14
15 uniform float zoom;
16
17 out vec4 fragColor;
18
19 // -----
20
21 vec3 barrel( float amp, vec2 uv ) {
22     float corn = length( vec2( 0.5 ) );
23     float a = sin( 3.0 * soft( amp ), corn * PI );
24     float x = corn / ( tan( corn * a ) + corn );
25     vec2 uv = saturate(
26         ( uv + normalized( uv - 0.5 ) * tan( length( uv - 0.5 ) * a ) ) * x +
27         0.5 * ( 1.0 - a )
28     );
29     return texture( uTD2DInputs[0], uv.wy ).xyz;
30 }
31
32
33 void main() {
34     vec2 uv = vUV.wy;
35
36     vec3 tex = vec3( 0.0 );
37
38     for ( int i = 0; i < BARREL_ITER; i++ ) {
39         float f1 = ( float( i ) + 0.5 ) / float( BARREL_ITER );
40         vec3 a = saturate( vec3(
41             1.0 - 3.0 * abs( 1.0 - 6.0 - f1 ),
42             1.0 - 3.0 * abs( 1.0 - 2.0 - f1 ),
43             1.0 - 3.0 * abs( 5.0 - 6.0 - f1 )
44         ) * float( BARREL_ITER ) ) * 4.0;
45         tex += a * barrel( BARREL_AMP * zoom + BARREL_DIFF * a * f1, uv );
46     }
47
48     fragColor = vec4( tex, 1.0 );
49 }
```

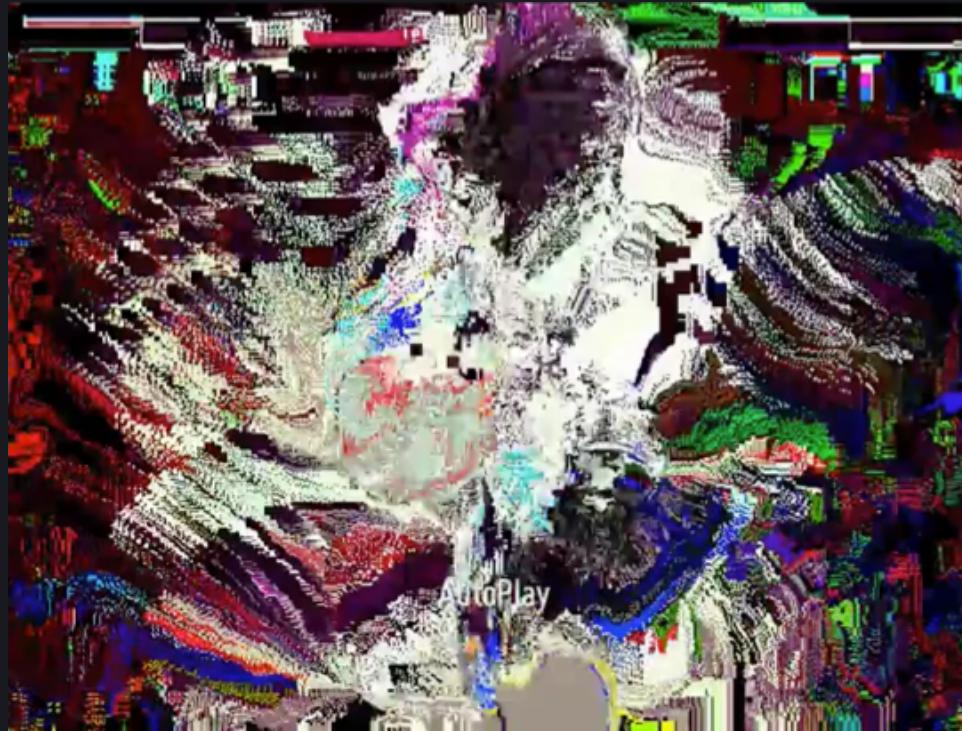
概要

- 汎用



概要

- 汎用



Stepmaniaという音ゲーシミュレータに
GLSLをインテグレートしました

概要

- 汎用



はいいぞ

視覚的にわかりやすいVPLで
直接GLSLが書けるのは爆アド
→ GLSLのテスト環境に最適！

概要

学べること

- TD上で複数のGLSL TOPを組み合わせ
ポストエフェクトを作る方法を学べます
- GLSL TOPを扱う際に
有用なTDの機能が学べます（型・解像度）
- 画像をデータとして扱う考え方を学べます

概要

学べないこと

- 逆にコードの解説はあんまりしません
(少なくともこのスライド上ではコードは出てきません)
- 「ワークショップ」という単語の意味は
わかりません (手はあんまり動かさないと思います)
- 京都のおいしいラーメン屋さんについては
わかりません (これ終わったらすぐ帰っちゃう ;_;)

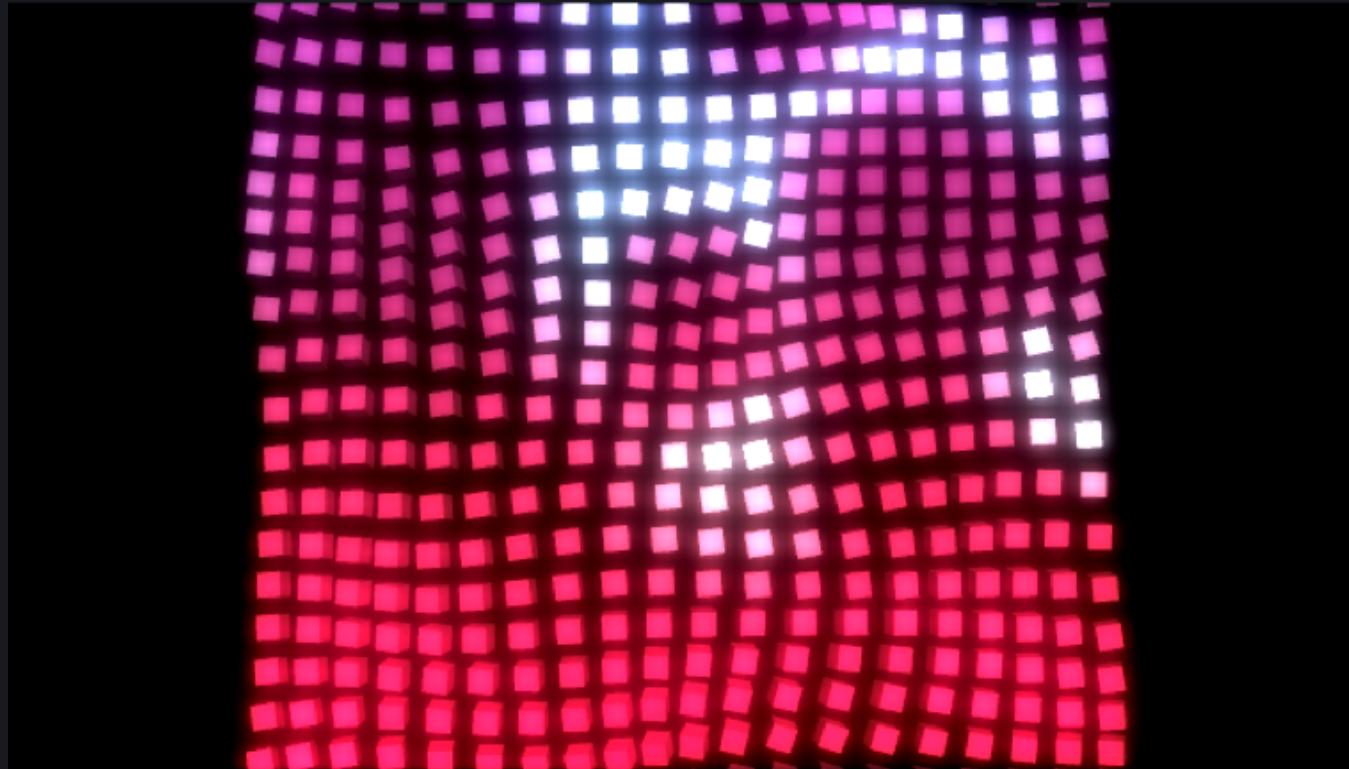
概要

Yutaka “FMS_Cat” Obuchi



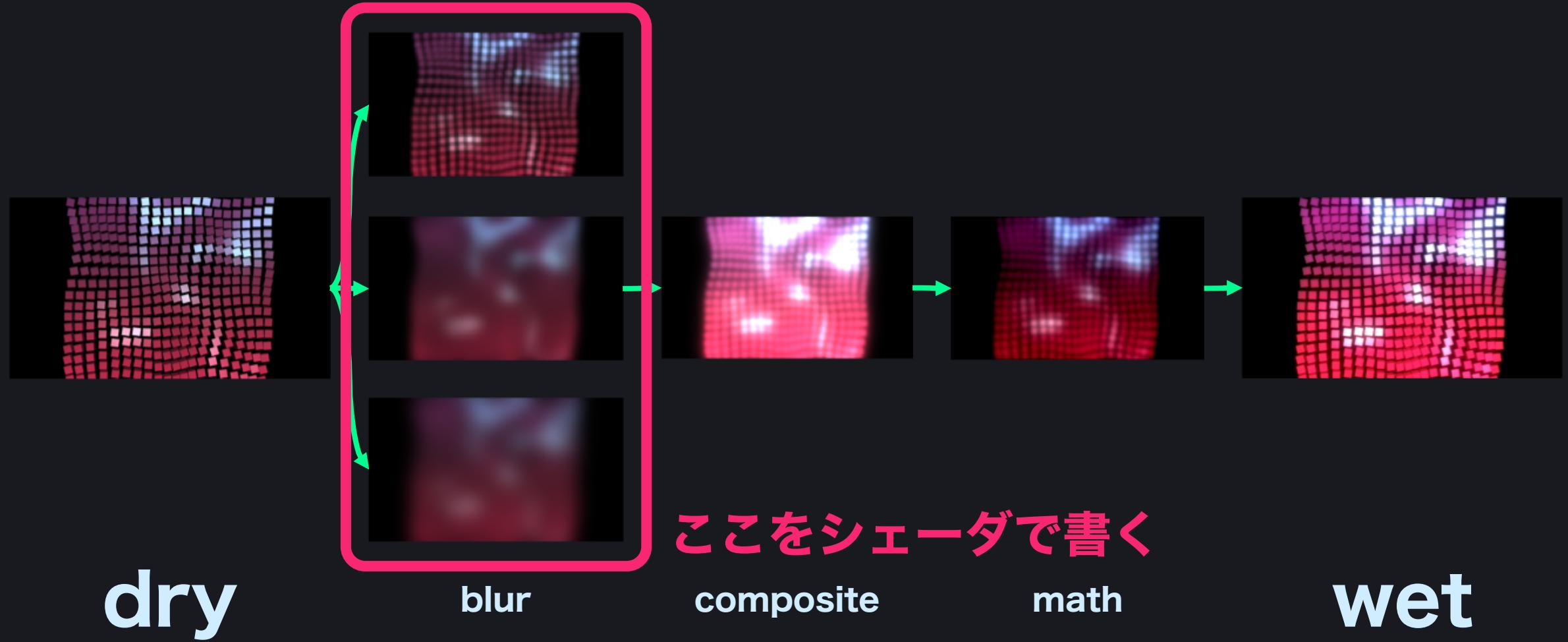
- 猫 (22)
- WebGLとGLSLだいすき
クリエイティブコーディング愛好家
- イマジカデジタルスケープという会社で
リアルタイムCGで遊ぶはずです (研修期間中)
- Everyday One Motionみてね

Bloom



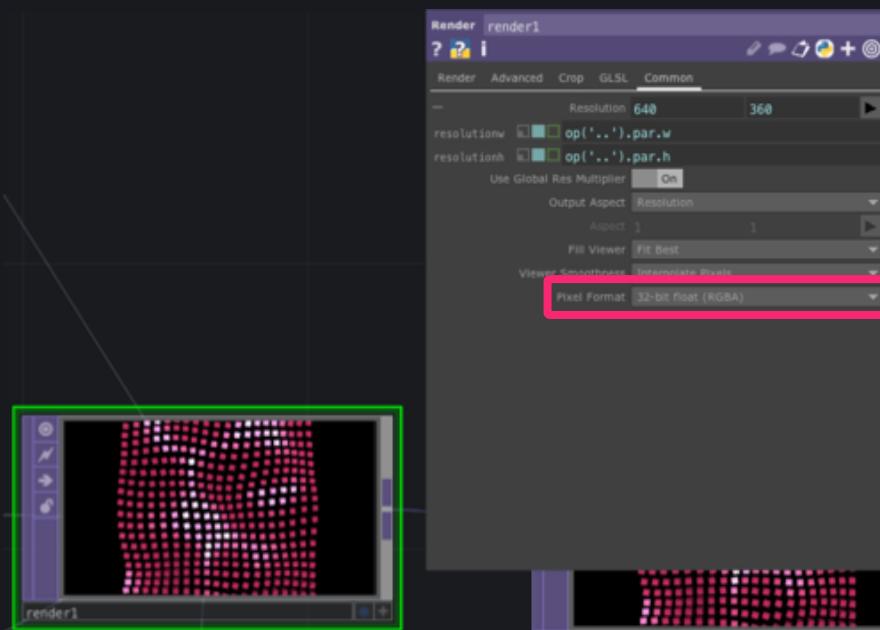
まぶしい

Bloom



Bloom

はじめに
映像の元ネタは**HDR**で作っておこう



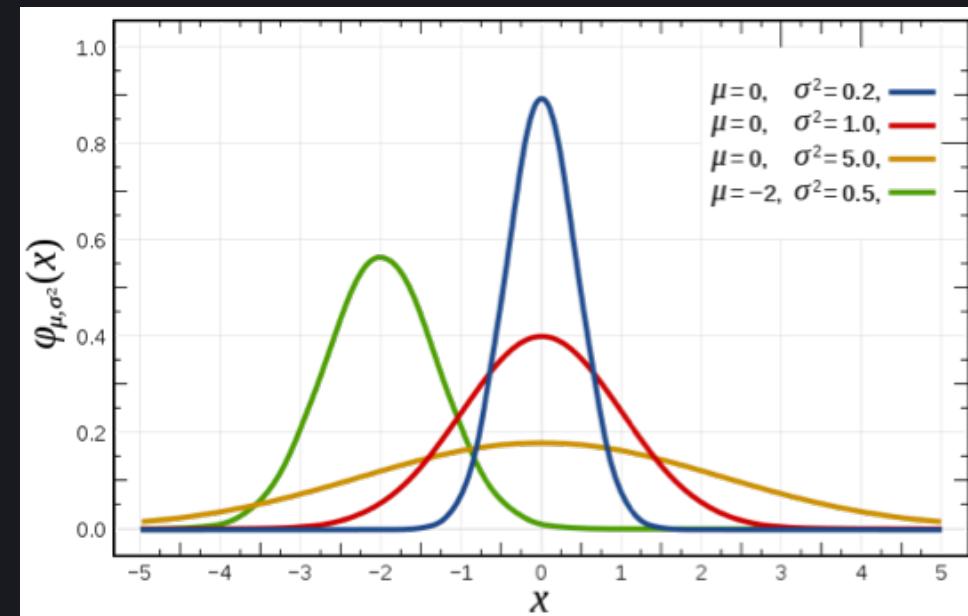
Render TOPの
PixelFormatを
Float32とかにすると
明るい箇所が**白飛びせず**
1以上の値が出ます

Bloom

Gaussianブラー

“ククク…ヤツはブラー四天王の中でも最弱…”

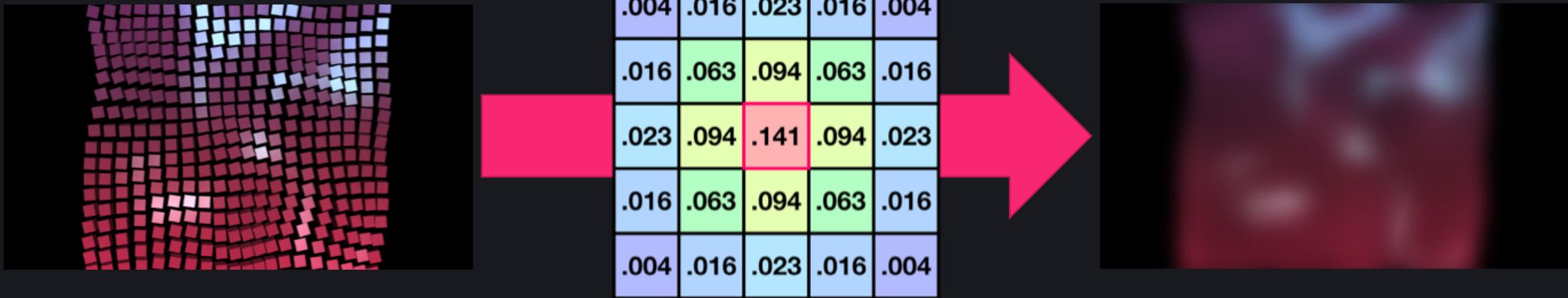
ガウス曲線を用いて
画像をぼかす処理
最もシンプルな
ブラーとして知られる



https://en.wikipedia.org/wiki/Gaussian_function

Bloom

Gaussianブラー



注目するピクセルの周囲のピクセルから色を取得
ガウス関数でウェイトを掛けながら総和を求める

Bloom

Gaussianブラー

.004	.016	.023	.016	.004
.016	.063	.094	.063	.016
.023	.094	.141	.094	.023
.016	.063	.094	.063	.016
.004	.016	.023	.016	.004

25回テクスチャを
サンプリング?

→重い

Bloom

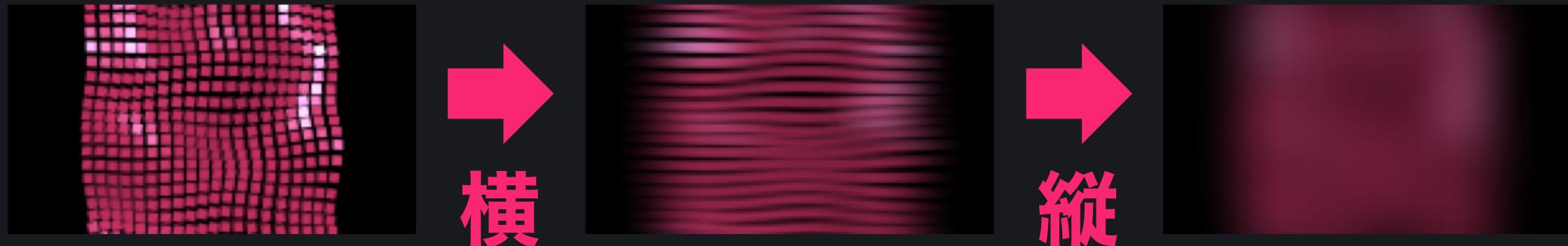
Gaussianブラー

ここで、

Gaussianブラーの処理は
縦・横に分割可能

Bloom

Gaussianブラー



5x5であれば

処理回数は25回 → 10回

Bloom

Bad news 1:

本当はガウスブラーをGLSLで
わざわざ作る必要なんてない



同じくらい
速い！

TD標準ブラー vs. GLSL自前ブラー

Bloom

Bad news 2:

まだ重い

画像x3に対してリアルタイムで
ブラーを掛けるのは結構しんどい

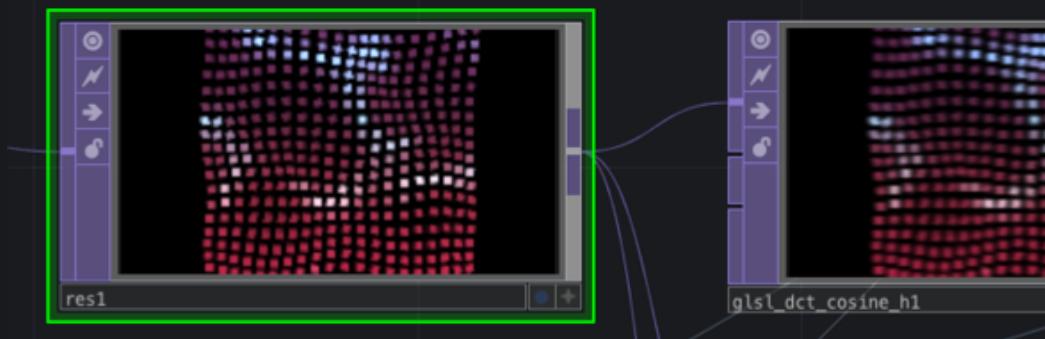
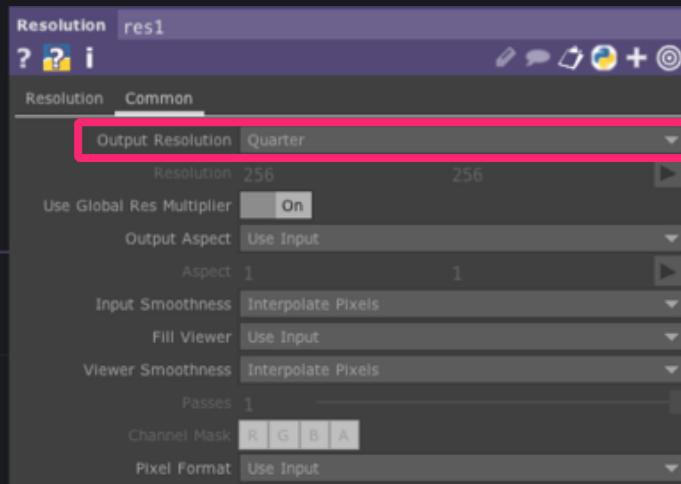
→ **解像度を下げよう**

どうせブラー処理だし大丈夫



:sob:

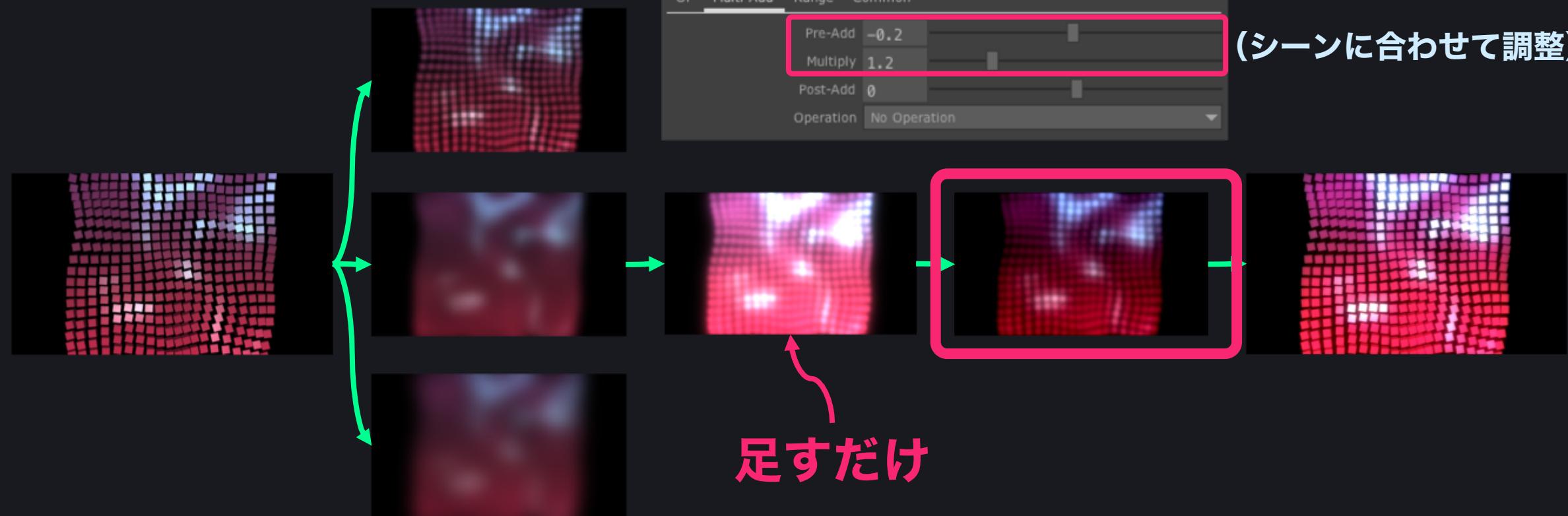
Bloom



Resolution TOPで
解像度を1/4にしてから
blurを行う

(あとで4倍して戻します)

Bloom



dry

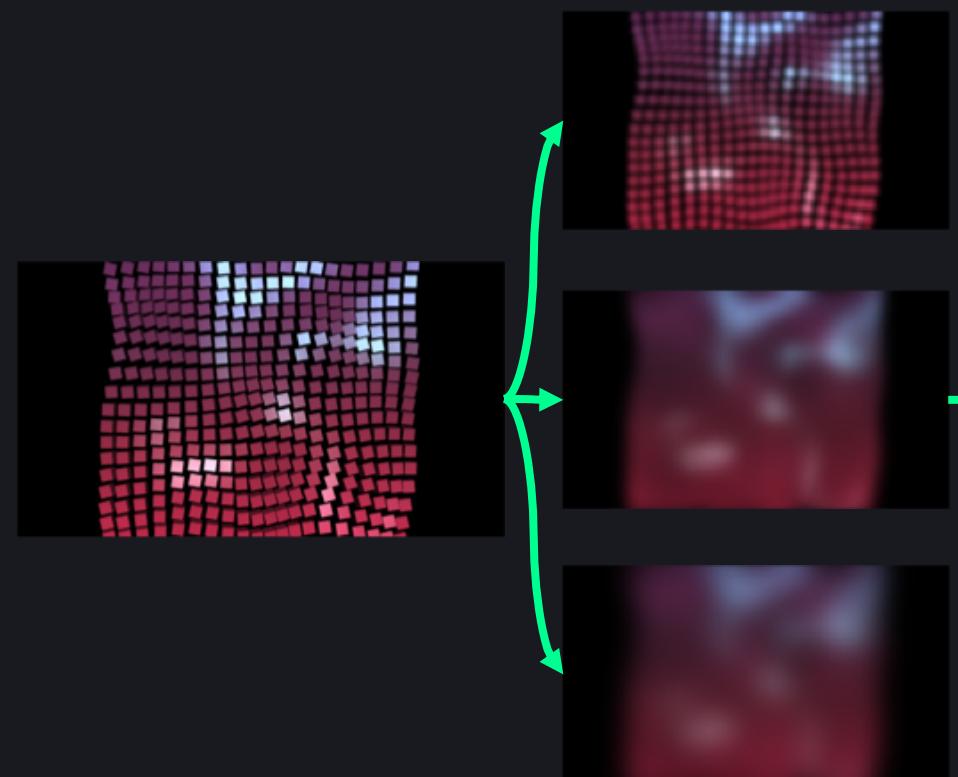
blur

composite

math

wet

Bloom



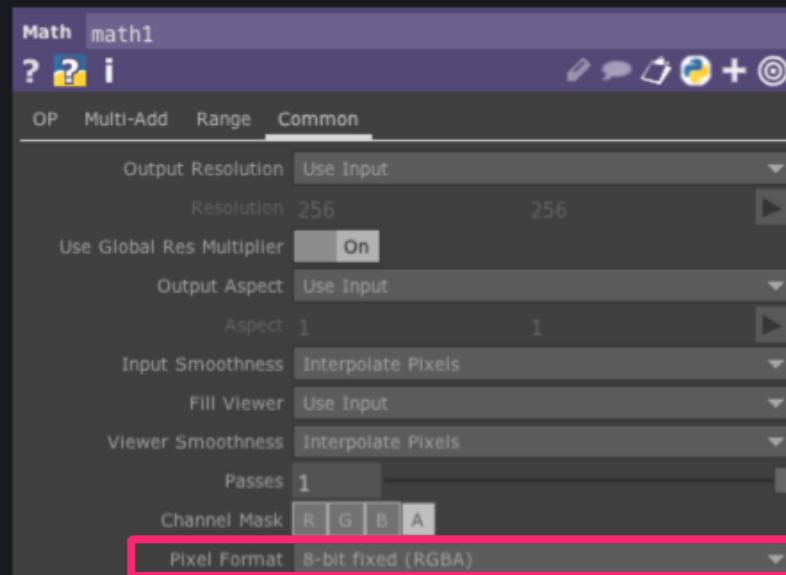
dry

blur

composite

math

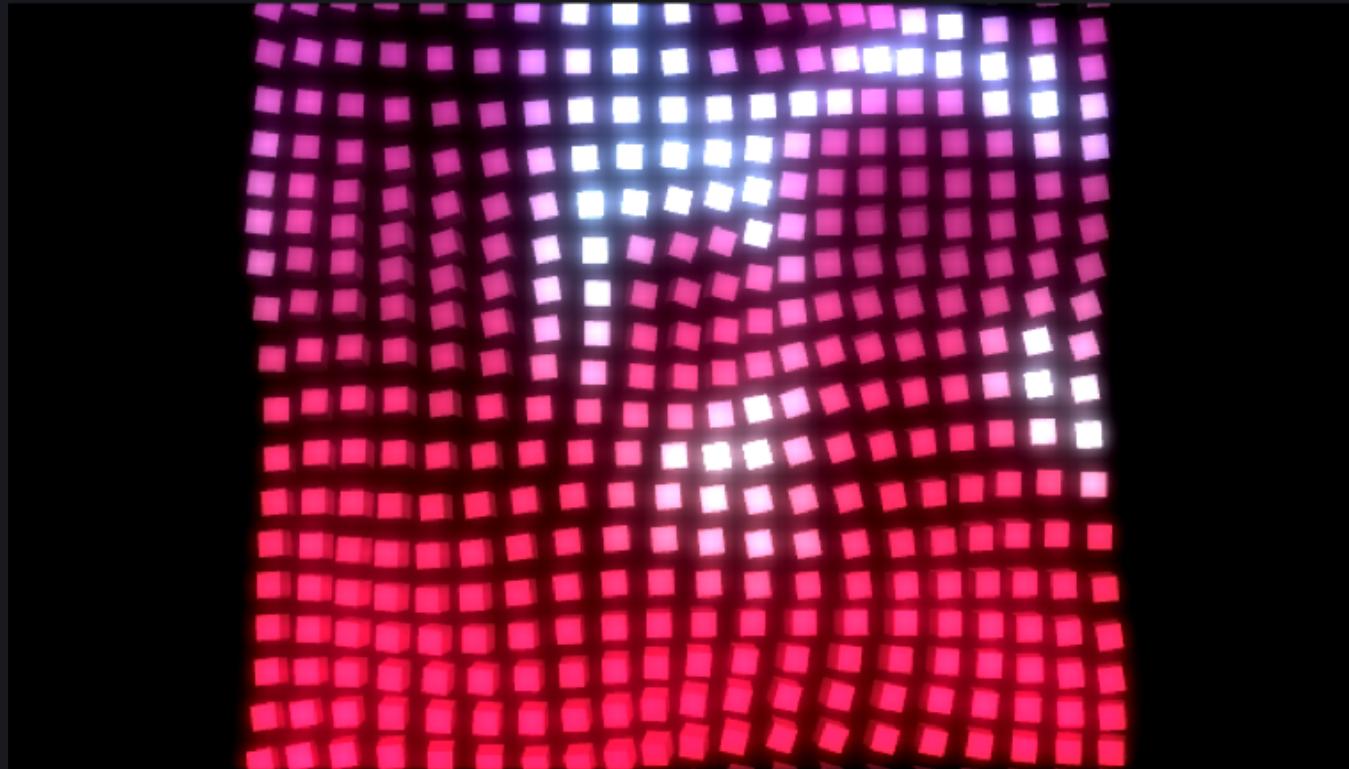
wet



足すだけ

Pixel Formatを
8bitに戻す！
→ これで0-1に
クランプされる

Bloom



完成

DCT圧縮



きたない

DCT圧縮



@FMS_Cat 2017年12月21日に更新 2154 views

編集する



WebGL Advent Calendar 2017 | 21日目

DCTでJPEGっぽいエフェクトを作るやつ

JavaScript

WebGL

GLSL

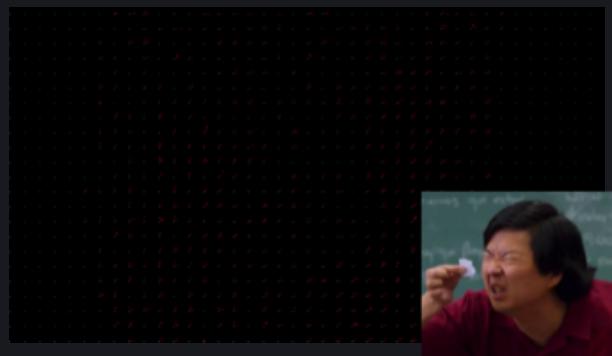
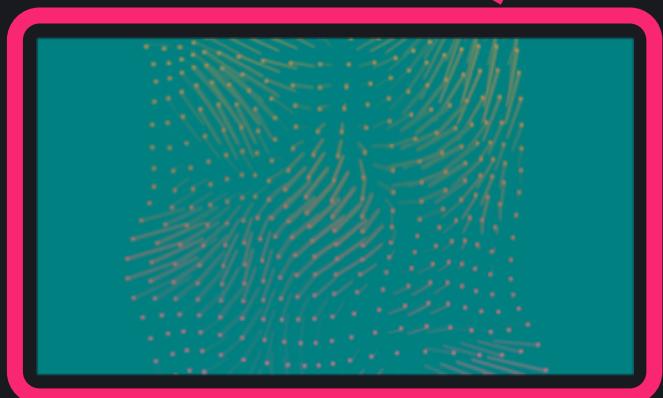
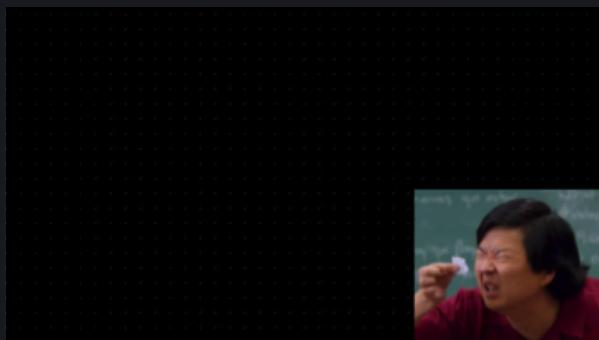
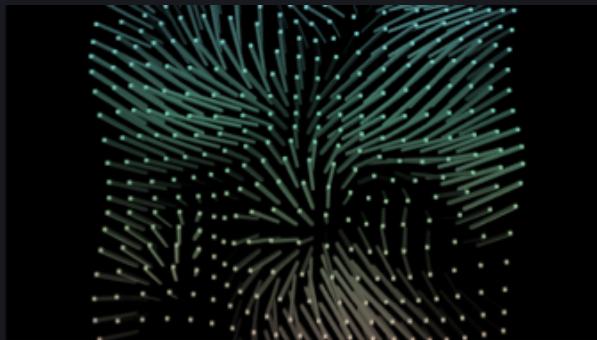
Shader

信号処理

https://qiita.com/FMS_Cat/items/6943c708a875984228b9

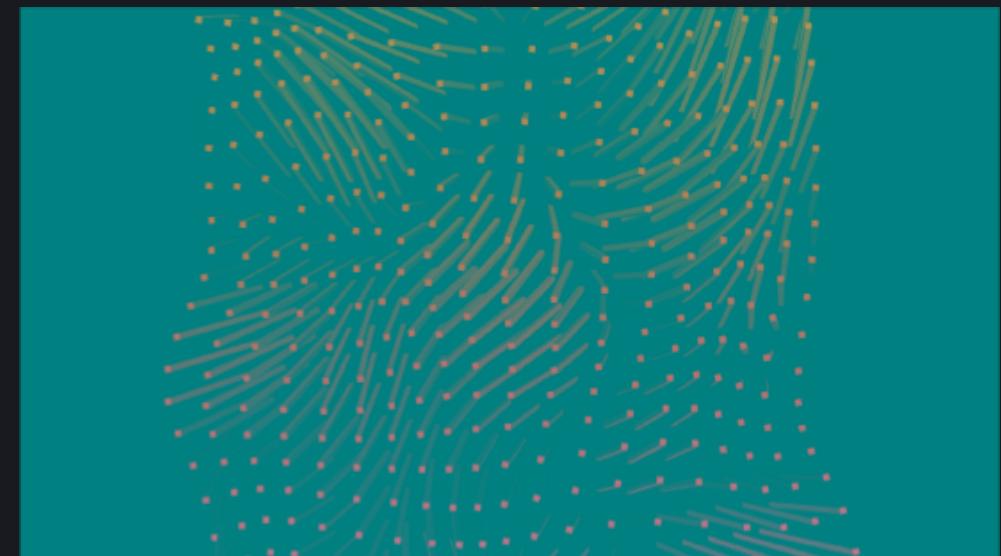
DCT圧縮

すべてGLSL TOP



DCT圧縮

RGB色空間 → YCbCr色空間



JPEGではYCbCr色空間で色を扱っている

DCT圧縮

RGB色空間 → YCbCr色空間



Y: 黒-白 [0.0, 1.0]
Cb: 黄緑-青 [-0.5, 0.5]
Cr: 青緑-赤 [-0.5, 0.5]

今回のサンプル上ではCb・Crの
値域も[0.0, 1.0]で扱っています

DCT圧縮

RGB色空間 → YCbCr色空間



DCT圧縮

RGB色空間 → YCbCr色空間

ログインしていません トーク 投稿記録 アカウント作成 ログイン

ページ ノート 閲覧 編集 履歴表示 Wikipedia内を検索 検索

YUV

YUVやYCbCrやYPbPrとは、輝度信号Yと、2つの色差信号を使って表現される色空間。

目次 [非表示]

- 1 概要
- 2 YUVおよびYCbCrの表記
- 3 YCbCrとYPbPrの使い分け
- 4 ケーブル
- 5 色差成分を間引く方法
- 6 動画フォーマット
- 7 RGBからの変換
 - 7.1 MPEGの場合
- 8 参照
- 9 外部リンク

RGBの色域内で表されるU-V色平面で Y' の値 = 0.5の例

DCT圧縮

RGB色空間 → YCbCr色空間

RGBからの変換 [編集]

以下、R, G, B, Yの値域は[0, 1]。Cb, Crの値域は[-0.5, 0.5]。U = 0.872 × Cb, V = 1.23 × Cr。

RGBからの変換式は

YUV (PAL, SECAM)

- $Y = 0.299 \times R + 0.587 \times G + 0.114 \times B$
- $U = -0.14713 \times R - 0.28886 \times G + 0.436 \times B$
- $V = 0.615 \times R - 0.51499 \times G - 0.10001 \times B$

ITU-R BT.601 / ITU-R BT.709 (1250/50/2:1)

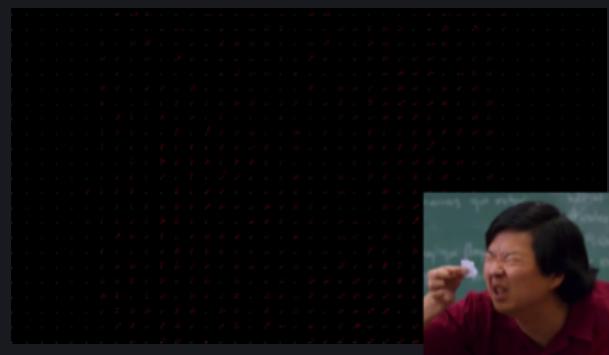
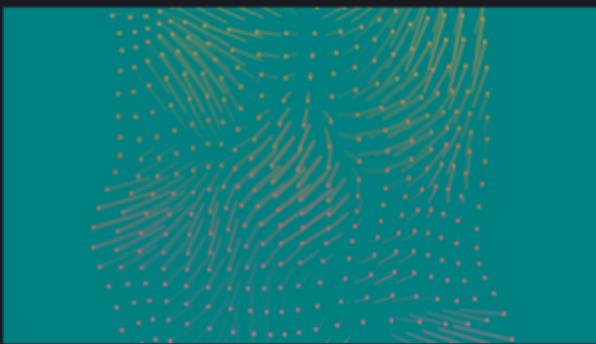
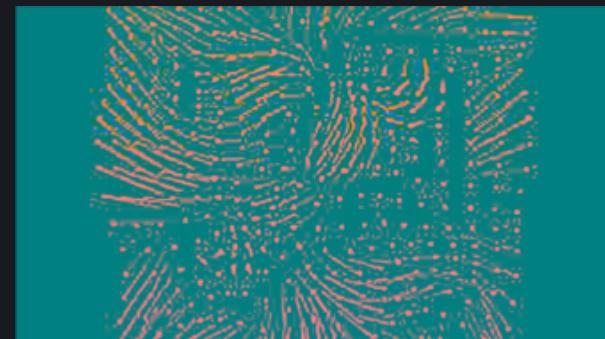
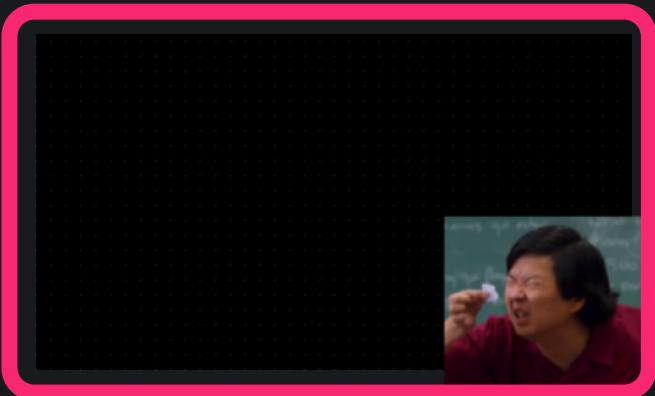
- $Y = 0.299 \times R + 0.587 \times G + 0.114 \times B$
- $Cb = -0.168736 \times R - 0.331264 \times G + 0.5 \times B$
- $Cr = 0.5 \times R - 0.418688 \times G - 0.081312 \times B$

ITU-R BT.709 (1125/60/2:1)

- $Y = 0.2126 \times R + 0.7152 \times G + 0.0722 \times B$
- $Cb = -0.114572 \times R - 0.385428 \times G + 0.5 \times B$
- $Cr = 0.5 \times R - 0.454153 \times G - 0.045847 \times B$

やるだけ

DCT圧縮



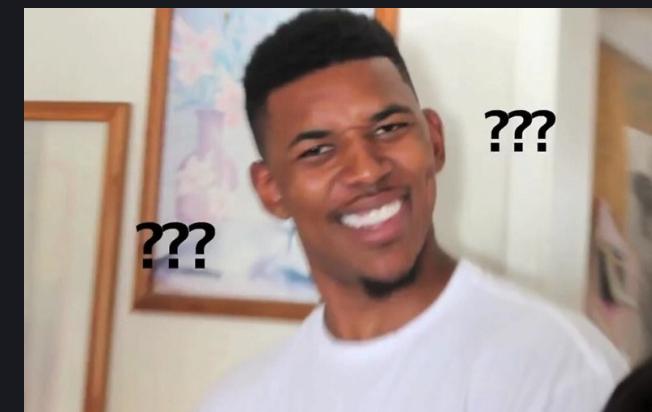
DCT圧縮

DCT変換

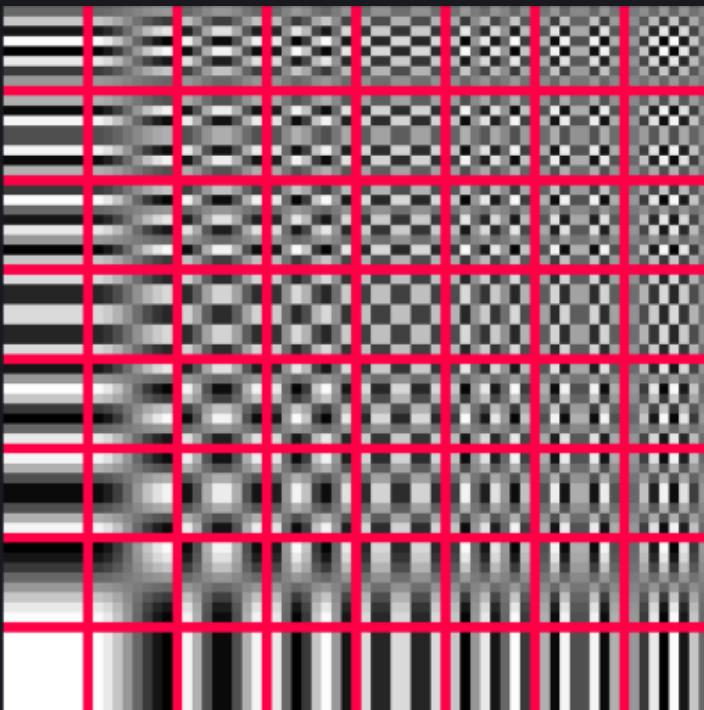
Discrete
Cosine
Transform

離散コサイン変換

画像信号を
コサイン波の
集合に変換



DCT圧縮



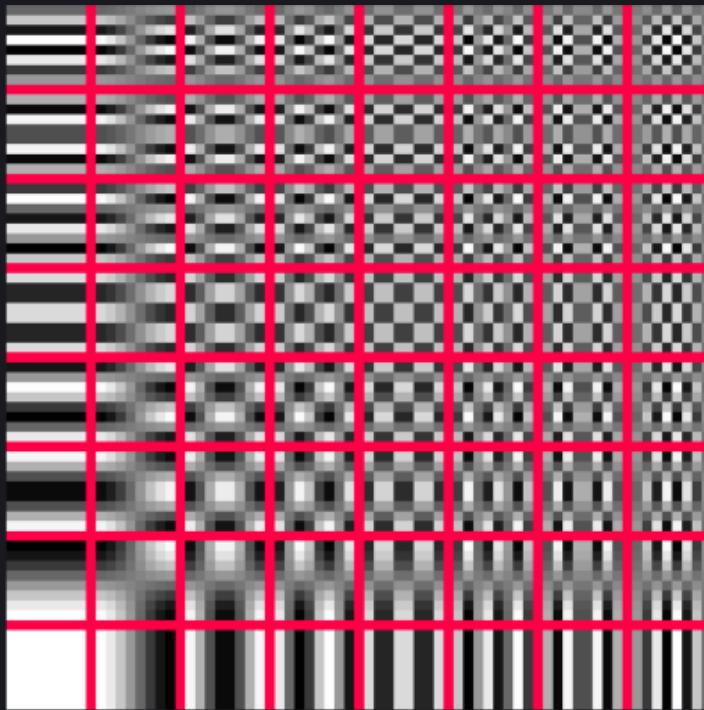
これらの64種類の
波の組み合わせで
任意の8x8画像が表現できる

→ これらの成分を覚えておけば
画像が復元できる

8x8コサインパターン

DCT圧縮

周波数成分の求め方



左の各パターンと8x8画像片とで
各ピクセル同士の掛け算を行い
その平均が周波数成分となる

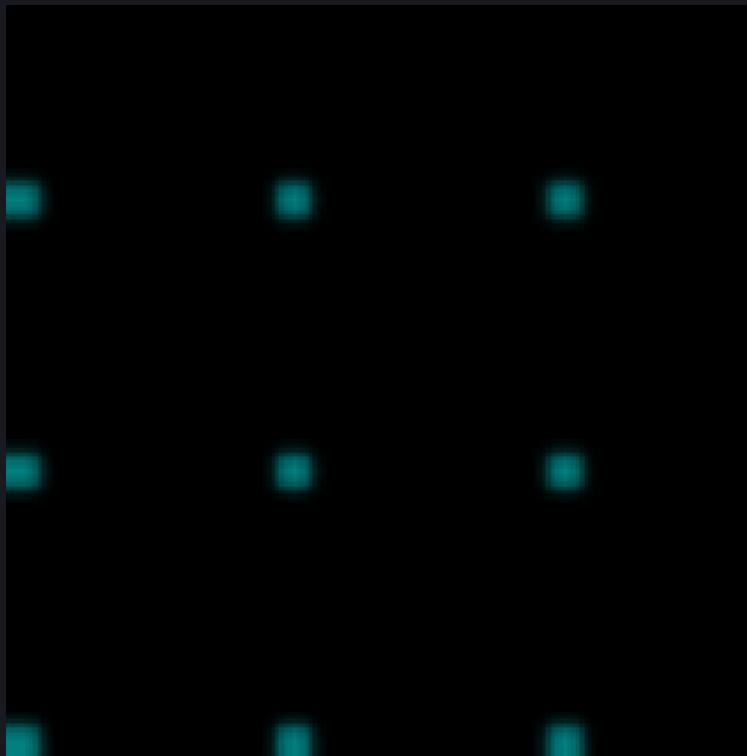
(詳しくはゲ グ ッテネ)

値域は[-1.0, 1.0]

DCT圧縮



DCT圧縮



8x8の周波数成分情報が
この  に刻まれている

見えない部分にも
情報がある！ (a値が0だから見えない)
(負の値も取り得る)

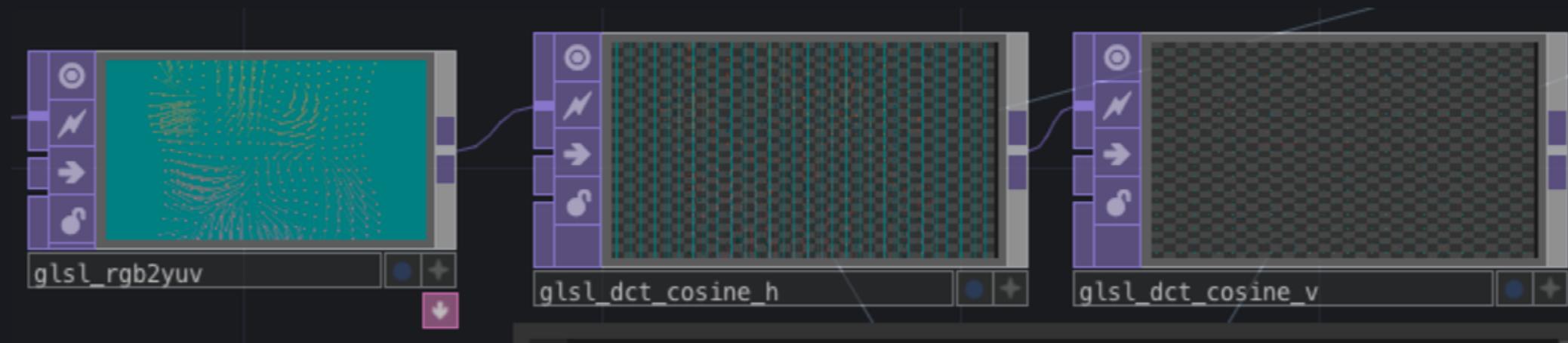
Pixel Format 32-bit float (RGBA) ▾

↑ 忘れずに！

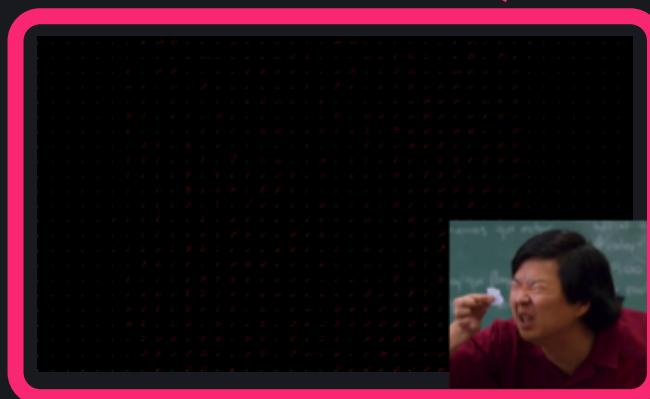
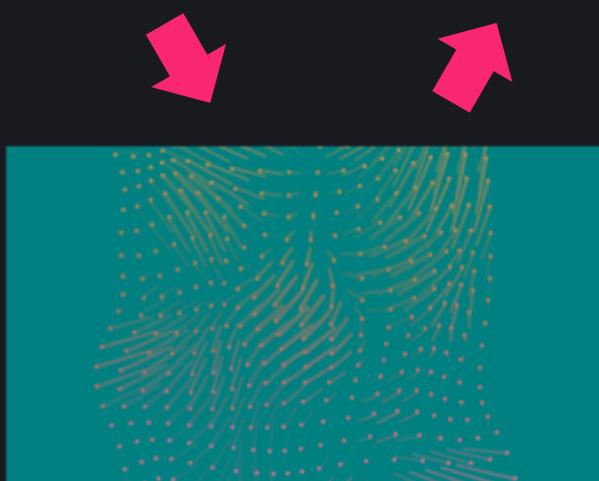
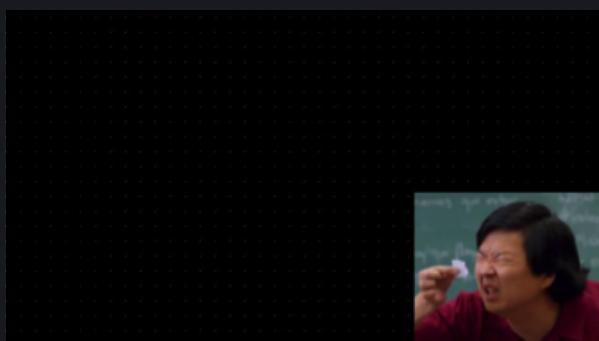
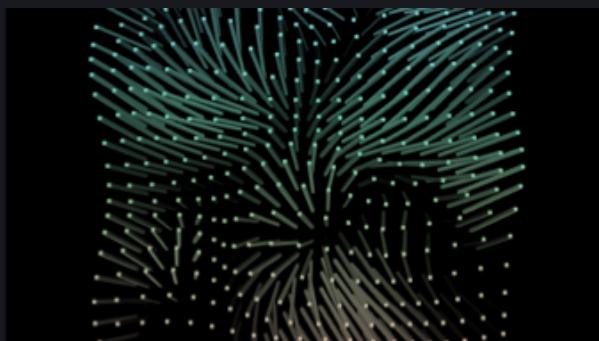
DCT圧縮

ブラーのときと同様……

この処理も縦横に分けて最適化できます



DCT圧縮



DCT圧縮

周波数成分で遊ぼう

```
22 void main() {  
23     vec2 coord = vUV.xy * resolution;  
24  
25     vec4 val = texture( sT02DInputs[0], coord / resolution );  
26  
27     vec2 blockOrigin = floor( coord / float( blockSize ) ) * float  
28     vec2 freq = floor( coord - blockOrigin );  
29     float lfreq = length( freq );  
30  
31     val *= 1.0 + lfreq * dctHiMult;  
32     float l = dctLofi + lfreq * dctLofiF;  
33     if ( l != 0.0 ) { val = lofi( val, l ); }  
34  
35     if ( dctMaxFreq < max( freq.x, freq.y ) ) {  
36         val = vec4( 0.0 );  
37     }  
38  
39     fragColor = T0OutputSwizzle( val );  
40 }
```

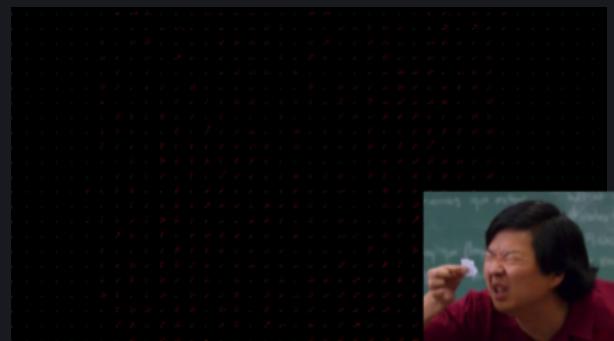
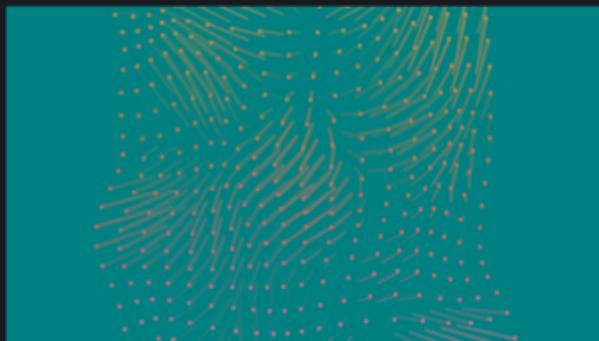
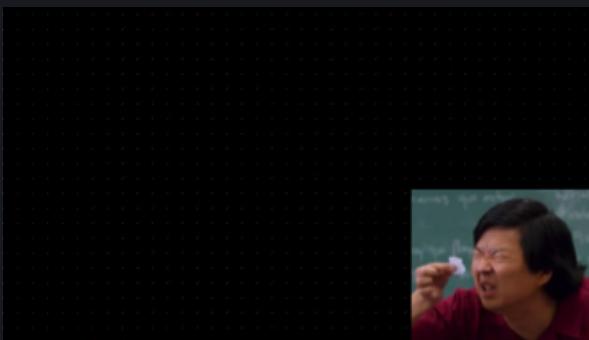
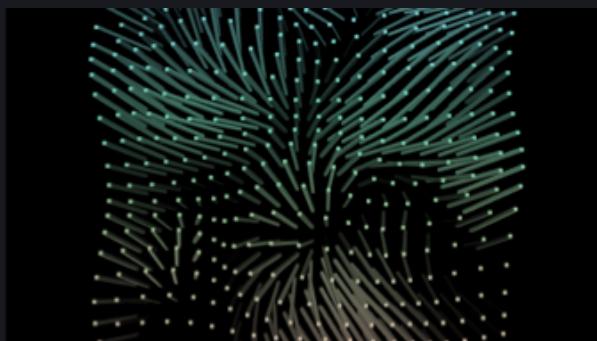
ほかにもいろいろ
できるかも

高い周波数成分を増幅

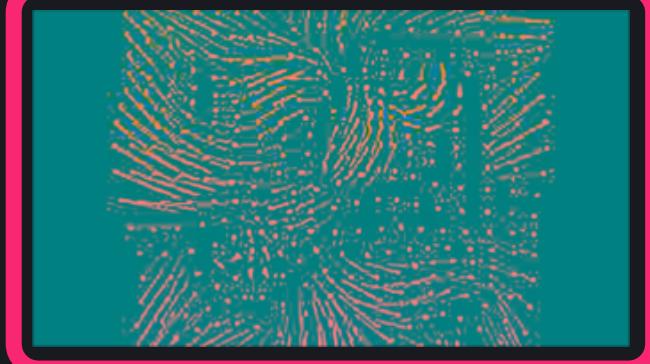
高い周波数成分をLoFiに

周波数の上限を設定

DCT圧縮



波を足し合わせるだけ



$YCrCb \rightarrow RGB$

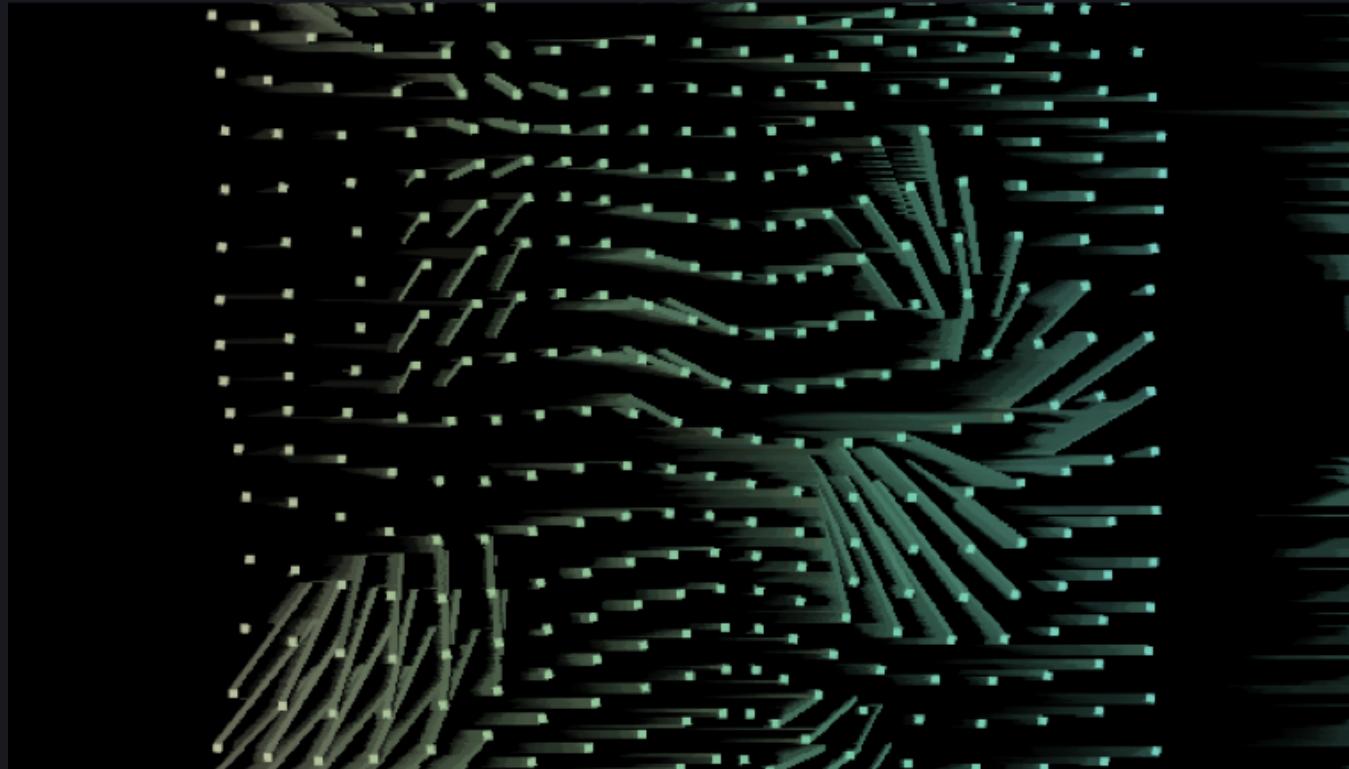


DCT圧縮



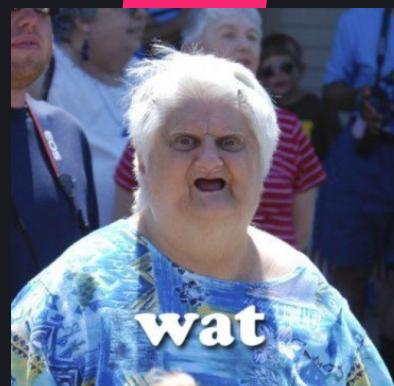
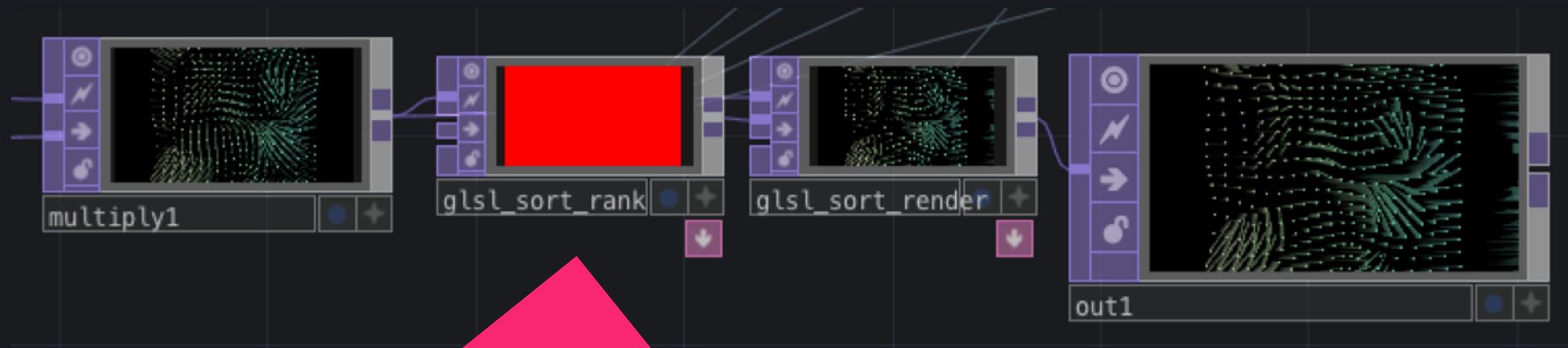
完成

PixelSort



かっこいい

PixelSort



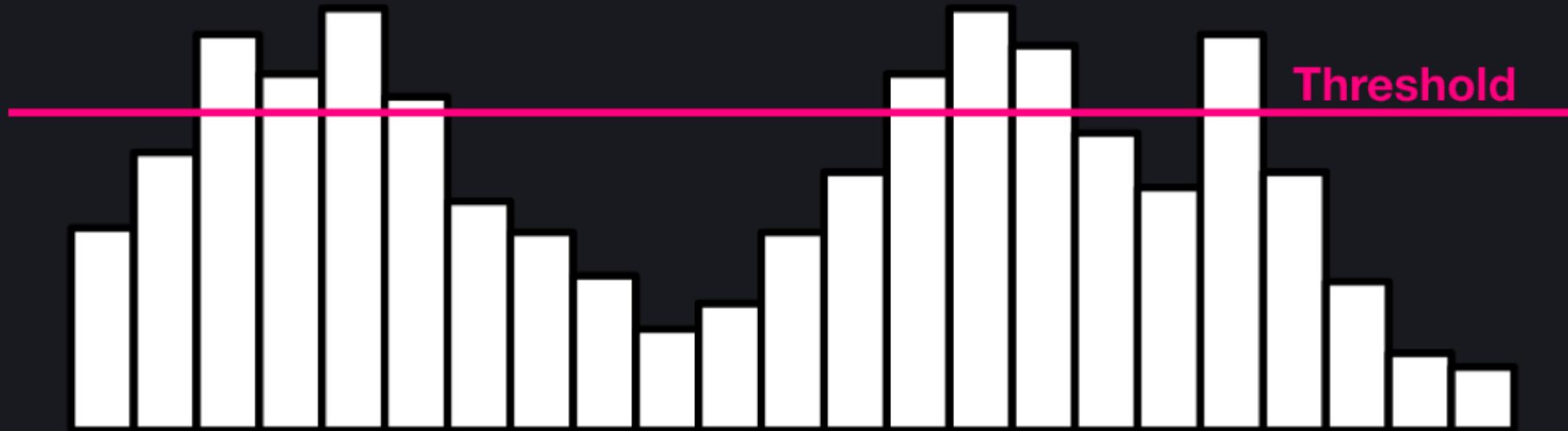
PixelSort

2段階でいきます

- ピクセルの順位を決める
- ピクセルを順位通りに並び替える

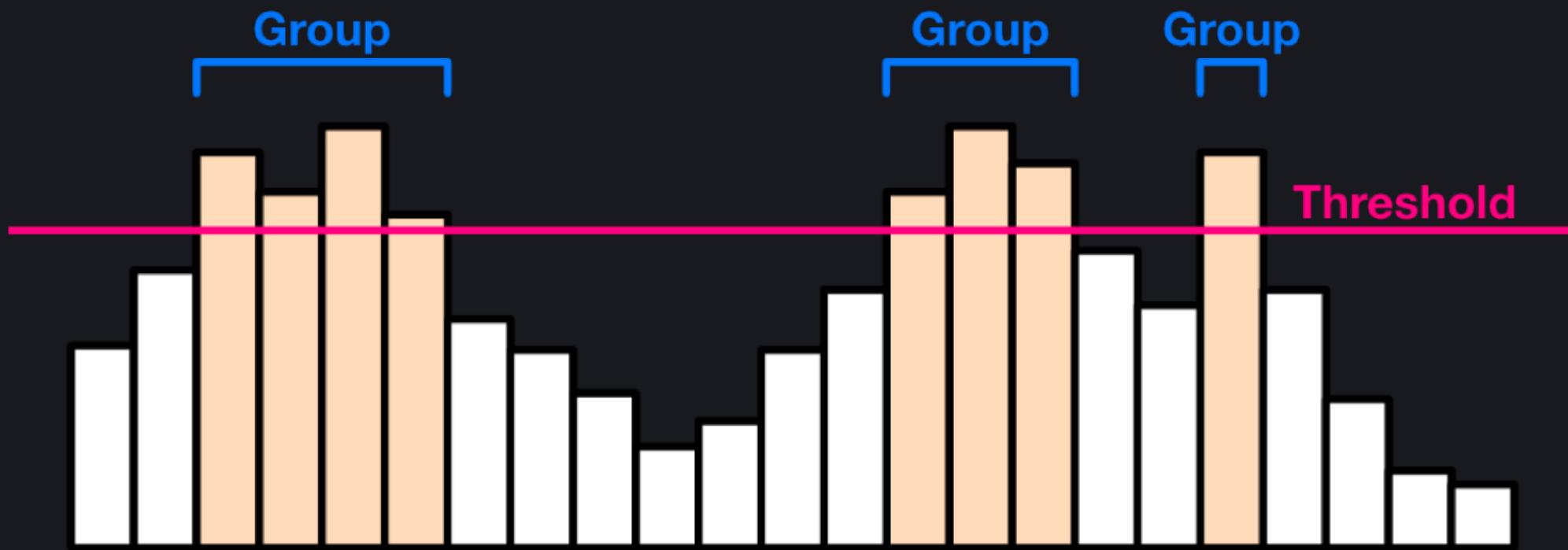
PixelSort

PixelSorterとは



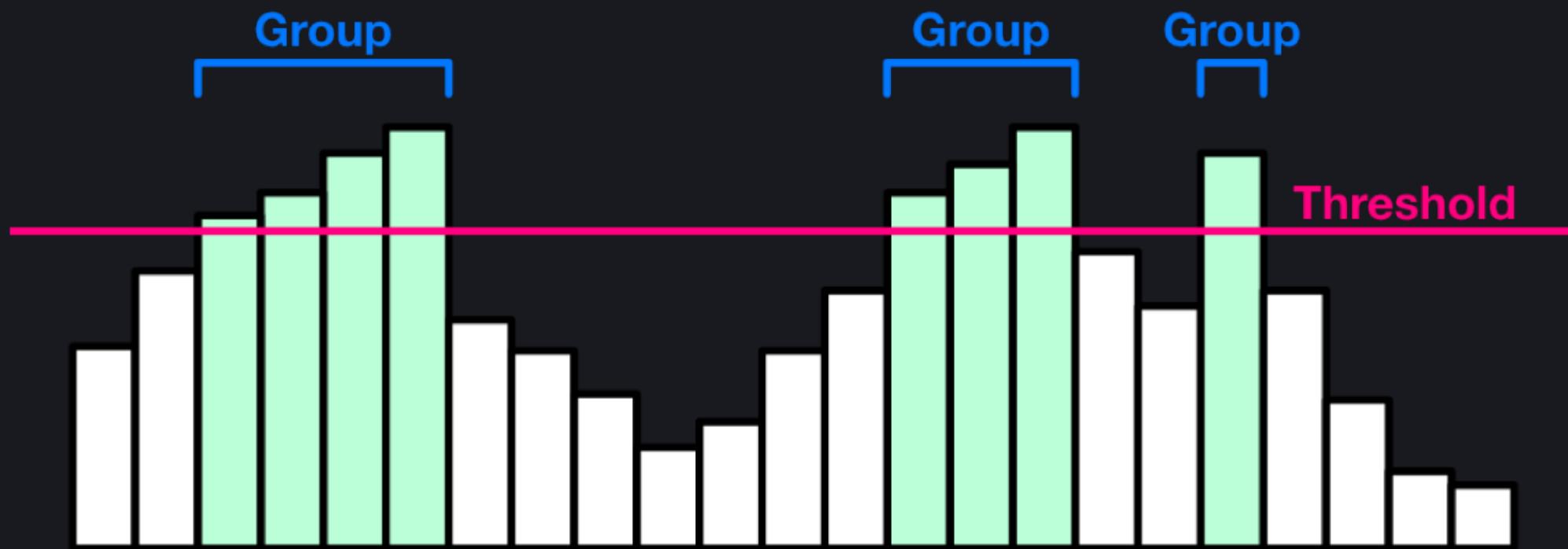
PixelSort

PixelSorterとは



PixelSort

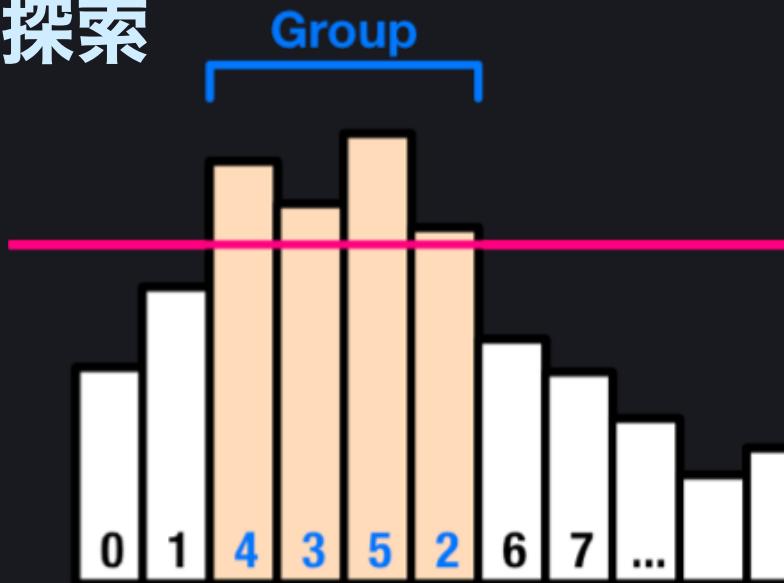
PixelSorter とは



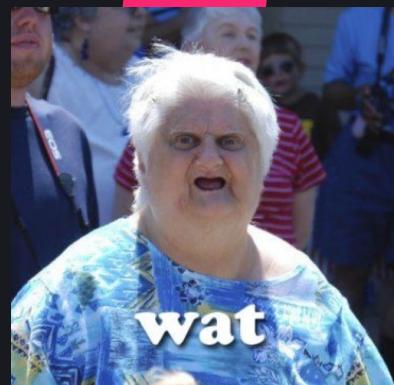
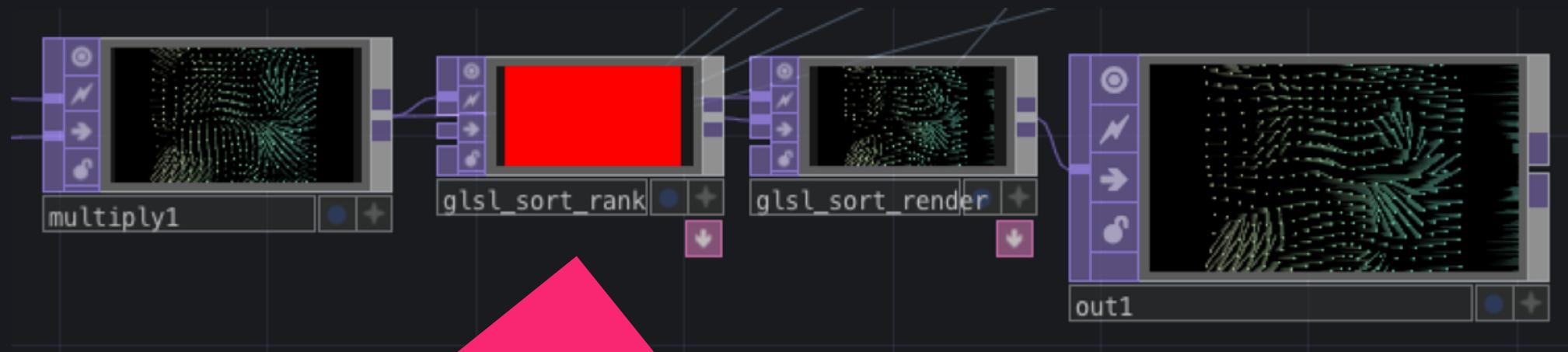
PixelSort

ピクセルの順位を決める

- ピクセルがしきい値より高いかチェック
- しきい値より高い場合、前後にピクセルを探索
- 探索したピクセルとそのピクセルの値とを比較しながら、グループの境界線を探す



PixelSort



ピクセルの順位が
整数で入っている
(Float32だからこそできる)

PixelSort

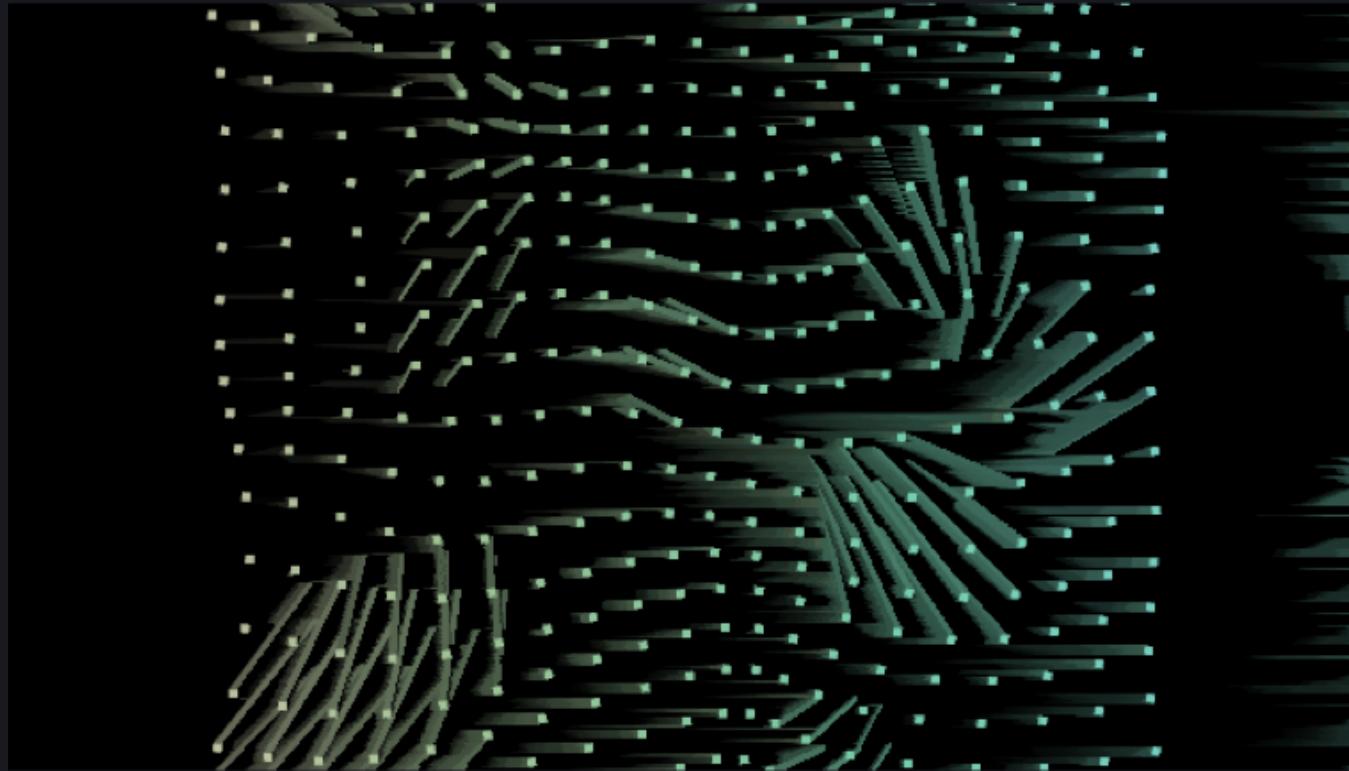
ピクセルを順位通りに並び替える

- 前後にピクセルを探索、その場所にいるべきピクセルがどこにあるか探す

正直効率は最悪

誰かもっといい方法知ってたら教えてください

PixelSort



完成

まとめ

資料はあらかじめ
作っておこう

まとめ

- TD上で複数のGLSL TOPを組み合わせ
ポストエフェクトを作る方法を学びました
- GLSL TOPを扱う際に
有用なTDの機能を学びました (型・解像度)
- 画像をデータとして扱う考え方を学びました
(YCbCr色空間・周波数成分・順位付け)

Any question?