



AMAGITA KAYOSI

はじめまして！
……の前に

この講義ではVEDAを使います

- ・しばらく喋るので、その間にインストール
お願いします🙏
- ・ <https://veda.gl/>
- ・ サンプルファイル
- ・ <https://goo.gl/mjGXH9>



AMAGITA KAYOSI



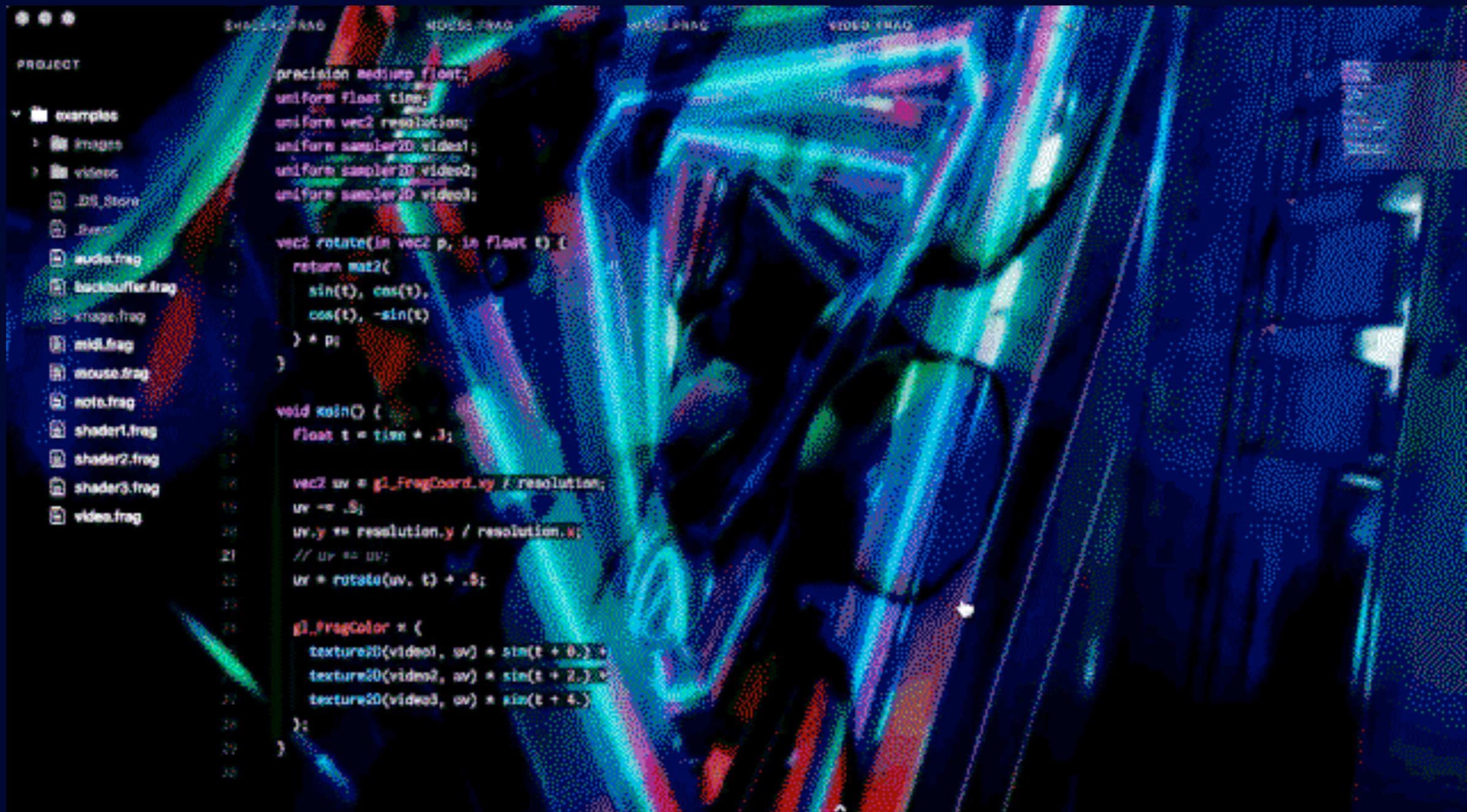
@amagitakayosi



GLSL活動歴

- ・ 1月: wglid.orgを見て勉強
- ・ 4月: GLSLによるVJを見て衝撃を受ける
- ・ 7月: VEDAを作る
- ・ 9月: クラブでVJデビュー

GLSLでVJします



VJって知ってる？

DJ (Disk Jockey)

- ・ 音楽を途切れることなく再生して
フロアを盛り上げる
- ・ ダンスマジックが多い

DJに必要なもの

- ・ 昔: ターンテーブル + レコード
- ・ 今: PC + DJソフト + コントローラー

VJ (Video Jockey)

- ・ 映像を途切れることなく再生して
フロアを盛り上げる
- ・ なんか知らんが
クリエイティブコーディングとか
プログラミングとの親和性が高い

VJに必要なもの

- ・ PC + VJソフト
 - ・ キーボードやMIDI等で操作
- ・ 最近はProcessingやTouchDesigner等でリアルタイムに映像を生成することも多い

VJにおけるGLSL

- ・ エフェクトとして用いられる事が多い
 - ・ VDMX, COGE等が対応
- ・ もちろんシェーダーだけでもVJできますよ？
 - ・ レイトレしたり



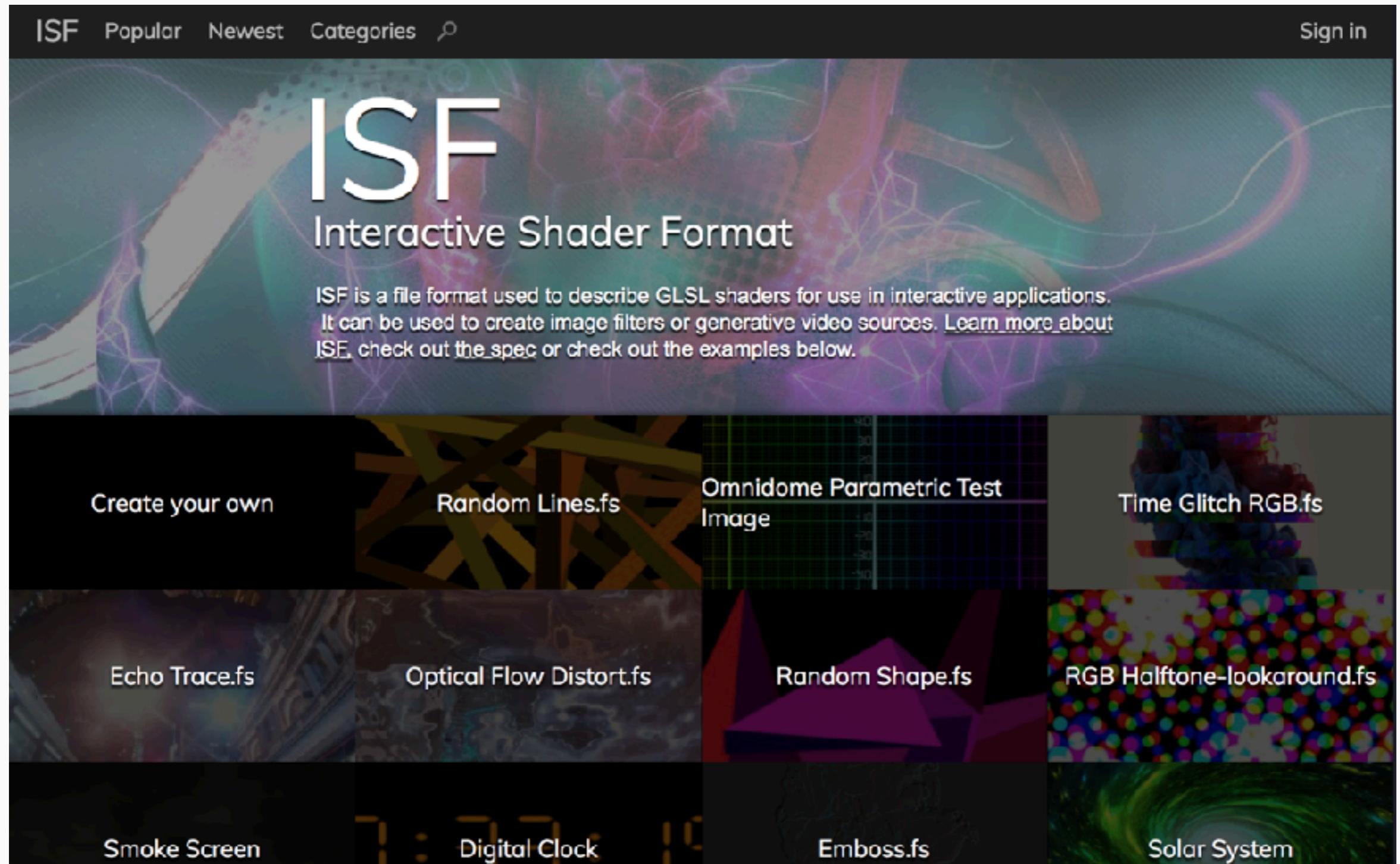
VDMX

- ・ 代表的なVJソフト
- ・ 僕の周りのVJは半分くらいこれ
- ・ GLSLエフェクトが大量に用意されている

InteractiveShaderFormat

- ・ VDMX等のVJソフトで利用するために作られた
シェーダーの仕様
- ・ ファイルの先頭にJSON形式のコメントを書く
- ・ 画像のIMPORTや
uniform変数の宣言ができる

参考になるのであとで見てみて



LIVE
TO POINT

ライブコーディング

- ・ リアルタイムにコードを書くパフォーマンス
- ・ 音楽の世界ではたまに見ますね
 - ・ SuperCollider, TidalCycles等々

ライブコーディングVJ

- ・何か知らんがライブコーディングVJはGLSLな事が多い
 - ・手早く派手な効果が得られるから？
- ・板ポリだったり、Unityで特定のオブジェクトのシェーダー書いたり

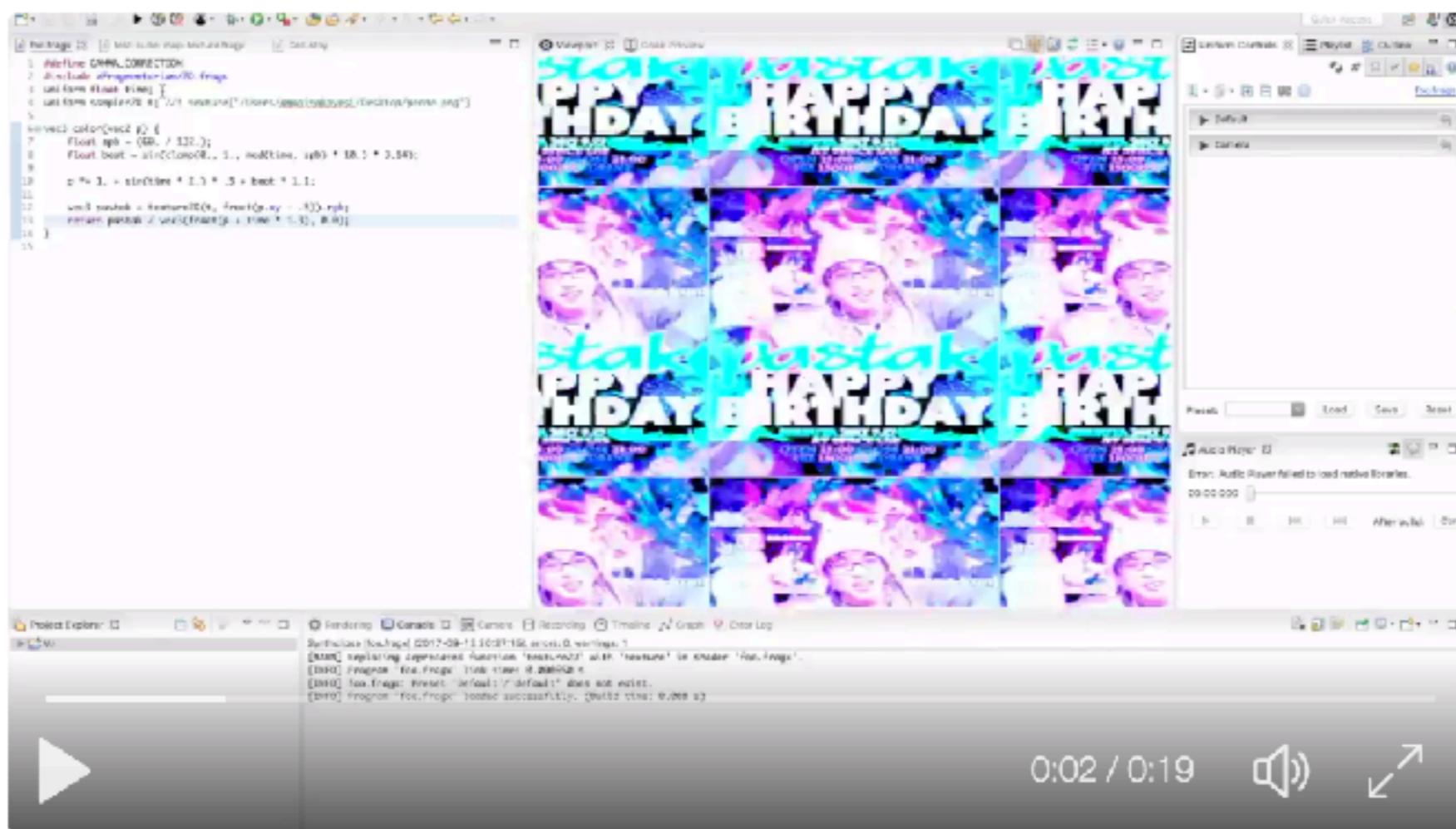
既存のライブコーディング環境

- Kodelife: metalで書けるデスクトップアプリ
- Shadertone: Emacs + Clojure
- HYLOGEN: Haskell製AltGLSL環境
- Synthclipse: ジャバ



amagi
@amagitakayosi

EclipseでVJできた (Synthclipseっていうプラグイン使った synthclipse.sourceforge.net #GLSL)



19:39 - 2017年9月13日

2件のリツイート 5件のいいね



© 2017 Twitter

規約 プ

既存環境への不満

- ・ JSと同じようにGLSLを開発したい！
- ・ コード補完したい
- ・ linter使いたい

自作すっか～

```
circle.frag • circle2.frag

precision mediump float;
uniform float time;
uniform vec2 resolution;
uniform sampler2D backbuffer;
uniform sampler2D spectrum;
// You can use GLSL Sandbox style uniform variables.

vec2 rotate(in vec2 p, in float t) {
    return mat2(sin(t), cos(t), cos(t), -sin(t)) * p;
}

// Hi there!
// This is a live-coding glsl demo! https://glsl-livecoder.com

void main()
{
    vec2 p = gl_FragCoord.xy;
    vec2 uv = (p * .3 + .5);
    p = rotate(p, 2. * time);

    // Audio input is also available!!
    float h = texture2D(spectrum, vec2(abs(p.x * 0.5), .0)).r;
    float m = texture2D(spectrum, vec2(abs(p.x * 0.3), .0)).r;
    float l = texture2D(spectrum, vec2(abs(p.x * 0.1), .0)).r;

    gl_FragColor = vec4(h, m, l, 1.) * (.05 / abs(length(p) - .3));
}
```



VEDA

- ・ AtomでGLSLを実行するパッケージ
- ・ 動画をリアルタイムに切り替えたりできる
- ・ 音声入力、MIDI入力などに対応
- ・ glslifyサポート



- ・ GitHubが開発しているテキストエディタ
- ・ ……なんだけど、Electronで動いている
 - ・ →Web AudioやWeb MIDIが使える！

そろそろ
インストール
終わりました？

glslangValidatorがない？

- ・ <https://veda.gl/install> を見て
インストールして下さい 😱
- ・ パス通してないかも？
- ・ まあ無くても意外と動くはず……

右下に🐞いませんか？

- ・ クリックしてRebuildしてください
- ・ Rebuild終わるとReloadボタン出るので
クリックしてください 😱

動かない人、遠慮せすぎきて！

- ・ もう動いてる人は、これまでに出てきた
キーワードでググったりしてて下さい

VEDA

体験下さい

△！！！

サンプルファイル体験タイム

- ・ <https://goo.gl/mjGXH9> を
ダウンロード&解凍
- ・ “glschool” フォルダを
Atomにドラッグ&ドロップ

VEDAを起動

- ・ Ctrl + Shift + P でコマンドパレットを開く
(macは Cmd + Shift + P)
- ・ “Veda: Toggle” を実行
 - ・ 依存パッケージのインストールが始まる

The screenshot shows a terminal window with a light blue gradient background. On the left, there's a file browser pane with a tree view. The root 'Project' node has a red dot icon. Under it, the 'examples' folder is expanded, showing subfolders 'images' and 'videos', and several GLSL shader files: '.vedarc', 'audio_strobo.frag', 'audio.frag', 'backbuffer.frag', 'camera.frag', 'combination.frag', 'combination.vert', 'gamepad.frag', 'gif.frag', 'glslify.frag', 'image.frag', 'key.frag', 'midi.frag', and 'mouse.frag'. A small red dot icon is also present next to the '.vedarc' file.

The main area of the terminal contains a code editor for a GLSL fragment shader named 'shader1.frag'. The code is as follows:

```
precision mediump float;
uniform float time;
uniform vec2 resolution;
void main() {
    vec2 uv = gl_FragCoord.xy / resolution.xy;
    gl_FragColor = vec4(uv,0.5+0.5*sin(time),1.0);
}
```

The code editor interface includes line numbers (1-9) on the left, syntax highlighting for keywords like 'precision', 'uniform', and 'void', and GLSL-specific tokens like 'mediump', 'float', 'vec2', and 'sin'. The status bar at the bottom shows the file name 'shader1.frag', line counts (0 0 0 0), the current line '9:1', and various status icons.

“shader2.frag” を実行

- ・ サイドバーから shader2.frag を開く
- ・ Ctrl + Enter で実行！！

A screenshot of a dark-themed code editor, likely Sublime Text, displaying GLSL shader code. The project sidebar on the left lists various shader files: glslify.frag, image.frag, key.frag, midi.frag, mouse.frag, multipass.frag, multipass.vert, note.frag, osc.frag, particle.frag, particle.vert, rotate2d.frag, server.frag, shader1.frag, shader2.frag, shader3.frag, vertex.vert, and video.frag. The file `shader2.frag` is currently selected and shown in the main editor area.

```
precision mediump float;
uniform float time;
uniform vec2 resolution;

void main() {
    vec2 p = (gl_FragCoord.xy * 2. - resolution) / mi
    gl_FragColor = vec4(
        0.3 / length(p + vec2(sin(time * 1.23) * 0.4,
        0.3 / length(p + vec2(sin(time * 2.23) * 0.4,
        0.3 / length(p + vec2(sin(time * 3.23) * 0.4,
        1.
    );
}
```

The code implements a three-pass effect where the fragment color is determined by the sum of three normalized vector components, each derived from a sine wave with different frequencies and phase shifts. The passes are weighted by 0.3 and scaled by 0.4. The final result is multiplied by 1.0 before being assigned to `gl_FragColor`.

**DATA
SCIENCE**

今日のメニュー

- ・ 動画を再生してみよう
- ・ エフェクトかけてみよう
- ・ 音声入力, カメラ入力を使ってみよう
- ・ (時間があれば) glslify

PLAY

THINK

KNOWLEDGE

その1：動画再生

動画を再生してみよう

- ・ “video.frag” を実行してみよう
- ・ 3つの動画をテクスチャに読み込んで
自動で切り替えている

Project

glsliby.frag

image.frag

key.frag

midi.frag

mouse.frag

multipass.frag

multipass.vert

note.frag

osc.frag

particle.frag

particle.vert

rotate2d.frag

server.frag

shader1.frag

shader2.frag

shader3.frag

vertex.vert

video.frag

audio.frag

video.frag

```
2 uniform float time;
3 uniform vec2 resolution;
4 uniform sampler2D video1;
5 uniform sampler2D video2;
6 uniform sampler2D video3;
7
8 vec2 rotate(in vec2 p, in float t) {
9     return mat2(
10        sin(t), cos(t),
11        cos(t), -sin(t)
12    ) * p;
13 }
14
15 void main() {
16     float t = time * .3;
17
18     vec2 uv = gl_FragCoord.xy / resolution;
19     uv -= .5;
20     uv.y *= resolution.y / resolution.x;
```

video.frag* ① 0 ▲ 0 ② 0 5:9

LF UTF-8 2 Spaces GLSL git+ "A" ↵ master ↓ ↑ 1 ③ 0 files

動画の切り替え方1：手動

- ・コメントアウト
- ・曲のいい感じのところで実行
- ・一番使いやすいぞ！！！

動画の切り替え方2: 自動

- ・ `uniform float time` を使って
いい感じに切り替える
- ・ “if (mod(time, .8) > .4)” したり
- ・ “gl_FragColor *= sin(time)” したり
 - ・ video.fragはこのパターンですね

動画の切り替え方3: 外部入力

- ・ 音声に合わせたり、
MIDIやOSC入力を使ったり
- ・ 上手くできるとかっこいいぞ！！！

その2:
エフェクト

AN>P

exonite

ETTEK

エフェクトやってみよう

- ・ と思ったけど
前回もポストエフェクトでしたね ^ ^ ; ; ;

VJでよくあるエフェクト

- ・回転
- ・座標ゆがみ
- ・グリッチ

GLITCH!!!!!!

- ・ RGBずらし
- ・ 色収差
- ・ 座標ずらし
- ・ ズームずらし (?)

fx.fragを開いてください

- ・ それっぽいエフェクトがいくつか書いてある
- ・ Ctrl + / でコメントを外して
Ctrl + Enter で実行
- ・ ちょっとずつ解説します

AI

BY

その3:
外部入力

INPUT

VEDAで外部入力を使う

- ・ ファイルの先頭に `/*{ audio: true }*/`などを追加
- ・ midiの時は `/*{ midi: true }*/`
- ・ カメラの時は `/*{ camera: true }*/`

音声入力

- ・ 以下のuniformが使えます
 - ・ “float volume”: ボリューム
 - ・ “sampler2D samples”: 波形
 - ・ “sampler2D spectrum”: 周波数帯の強さ

“audio.frag” の結果

The screenshot shows a code editor window with a dark theme. On the left is a file tree under a 'Project' root, listing various shader files like glslify.frag, image.frag, key.frag, midi.frag, mouse.frag, multipass.frag, multipass.vert, note.frag, osc.frag, particle.frag, particle.vert, rotate2d.frag, server.frag, shader1.frag, shader2.frag (which is the active file), shader3.frag, vertex.vert, and video.frag. The right side contains the content of the 'audio.frag' file:

```
/*{ "audio": true }*/  
precision mediump float;  
uniform float time;  
uniform vec2 resolution;  
uniform sampler2D texture;  
uniform sampler2D spectrum;  
uniform sampler2D samples;  
uniform float volume;  
  
void main (void) {  
    vec2 uv = gl_FragCoord.xy / resolution.xy;  
    vec4 color = texture2D(texture, uv);  
  
    float freq = texture2D(spectrum, vec2(uv.x, .5)).r;  
    float wave = texture2D(samples, vec2(uv.x, .5)).r;  
  
    float r = 1. - step(0.01, abs(wave - uv.y));  
    float g = 1. - step(0.01, abs(freq - uv.y));  
    float b = 1. - step(0.01, abs(volume / 255. - uv.y));  
  
    color = vec4(r, g, b, 1.);  
    gl_FragColor = color;  
}
```

At the bottom, there are status icons for git (git+), a diff viewer, and file counts (1 file, 0 changes, 0 additions, 0 deletions). The status bar also shows the file name 'audio.frag' and line numbers 1:1.

ちなみに

- ・ VEDAの音声入力は
Web Audio APIを使っています
- ・ みなさんも WebGLで使えるぞ！！

GLSLにデータを渡す

- ・ JSからGLSLに渡せるデータの数は上限がある
- ・ 大量のデータを渡すにはテクスチャを使う！
 - ・ 波形をUint8Arrayに変換し、THREE.DataTextureでGPU側に送る

波形をtextureとして表現

The screenshot shows a code editor window for the GLSL-LIVECODER EXAMPLES project. The left sidebar lists files: examples, images, videos, .DS_Store, Jitter, audio_strobe.frag, audio.frag, backbuffer.frag, camera.frag, gamepad.frag, gles.frag, image.frag, key.frag, midi.frag, mouse.frag, note.frag, rotate2d.frag, shader1.frag, shader2.frag, shader3.frag, vertex.vert, and video.frag. The main editor area displays GLSL code for 'VIDEOTRACKAUDIO.FRAG'. The code defines uniforms for time, resolution, texture, spectrum, samples, and volume. It calculates UV coordinates from gl_FragCoord, samples from texture2D(texture, uv), and frequency from texture2D(spectrum, vec2(uv.x, .5)). It then uses the frequency to sample from texture2D(samples, vec2(uv.x, .5)) to get a wave value. The color is determined by the wave value and a green gradient. The final output is set to gl_FragColor = vec4(r, g, b, 1.). A preview window on the right shows a red surface with a green waveform texture.

```
precision medium float;
uniform float time;
uniform vec2 resolution;
uniform sampler2D texture;
uniform sampler2D spectrum;
uniform sampler2D samples;
uniform float volume;

void main (void) {
    vec2 uv = gl_FragCoord.xy / resolution.xy;
    vec4 color = texture2D(texture, uv);

    float freq = texture2D(spectrum, vec2(uv.x, .5)).r;
    float wave = texture2D(samples, vec2(uv.x, .5)).r;

    float r = wave;
    float g = 1. - step(0.01, abs(wave - uv.y));
    // float r = 1. - step(0.01, abs(wave - uv.y));
    // float g = 1. - step(0.01, abs(freq - uv.y));
    // float b = 1. - step(0.01, abs(volume / 255. - uv.y));
    gl_FragColor = vec4(r, g, 0, 1.);}
```

スペクトラムも同様

The screenshot shows a terminal window running the command `glsl-liverender -n /usr/share/liverender/examples/VIDEOTRACKAUDIO.FRAG`. The output is a spectrogram visualization.

The code in the editor is a GLSL fragment shader:

```
/* "audio": true */  
precision medium float;  
uniform float time;  
uniform vec2 resolution;  
uniform sampler2D texture;  
uniform sampler2D spectrum;  
uniform sampler2D samples;  
uniform float volume;  
  
void main (void) {  
    vec2 uv = gl_FragCoord.xy / resolution.xy;  
    vec4 color = texture2D(texture, uv);  
  
    float freq = texture2D(spectrum, vec2(uv.x, .5)).r;  
    float wave = texture2D(samples, vec2(uv.x, .5)).r;  
  
    float r = freq;  
    float g = 1. - step(0.01, abs(freq - uv.y));  
    // float r = 1. - step(0.01, abs(wave - uv.y));  
    // float g = 1. - step(0.01, abs(freq - uv.y));  
    // float b = 1. - step(0.01, abs(volume / 255. - uv.y));  
  
    gl_FragColor = vec4(r, g, 0, 1.);  
}
```

MIDI入力

- ・ 以下のuniformが使えます
 - ・ “sampler2D midi”: MIDIコントロール信号
 - ・ “sampler2D note”: ノートon/off情報
 - ・ 「ドが音量64だよ」みたいな情報

MIDIメッセージの構造

- MIDIコン等で使われるメッセージは3バイト



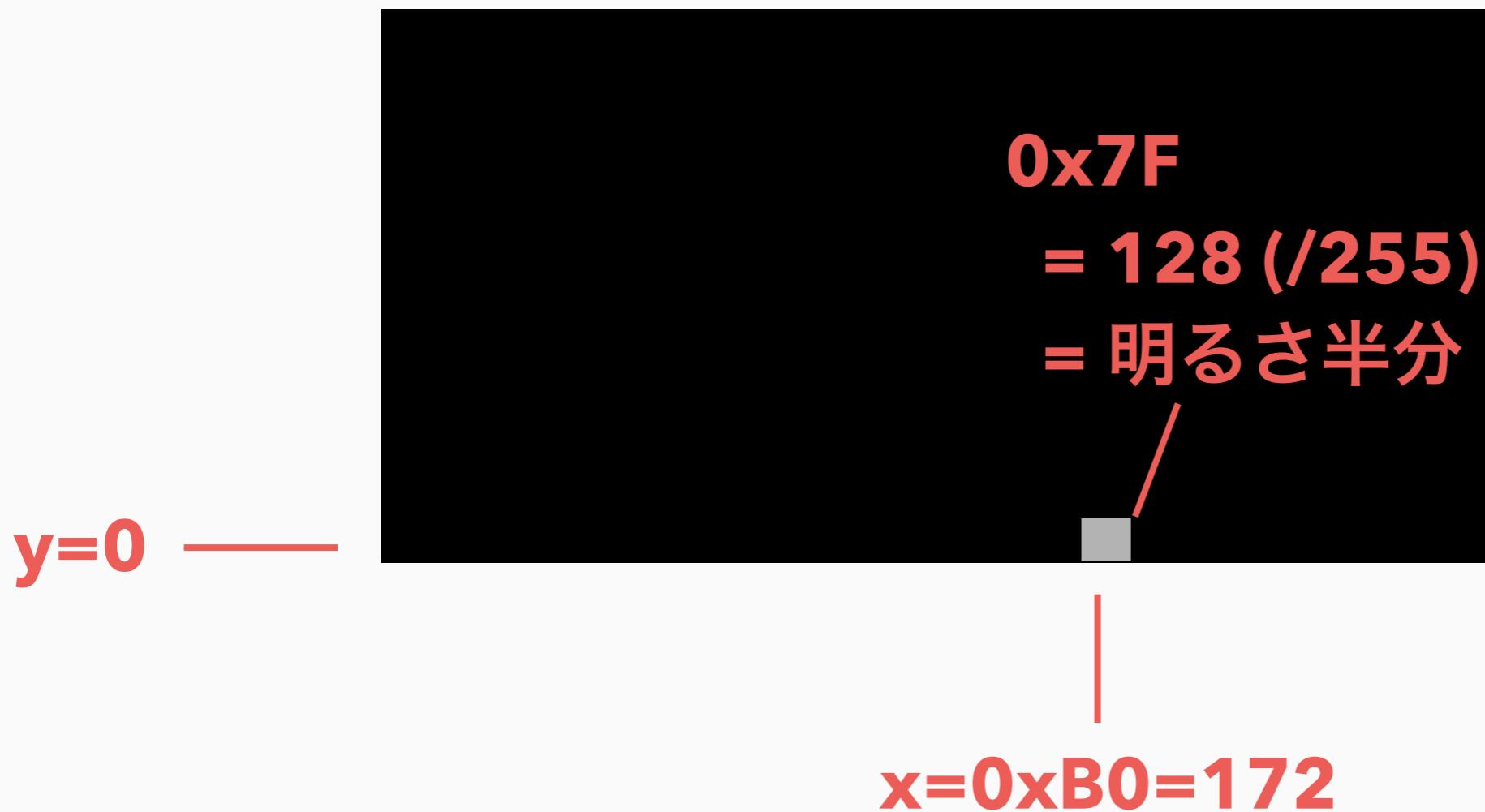
textureに変換

- ・ 256 × 128のtextureにマッピング
 - ・ 1byte目 -> x軸
 - ・ 2byte目 -> y軸
 - ・ 3byte目 -> 値

例: 0xB0 0x00 0x7F



例: 0xB0 0x00 0x7F



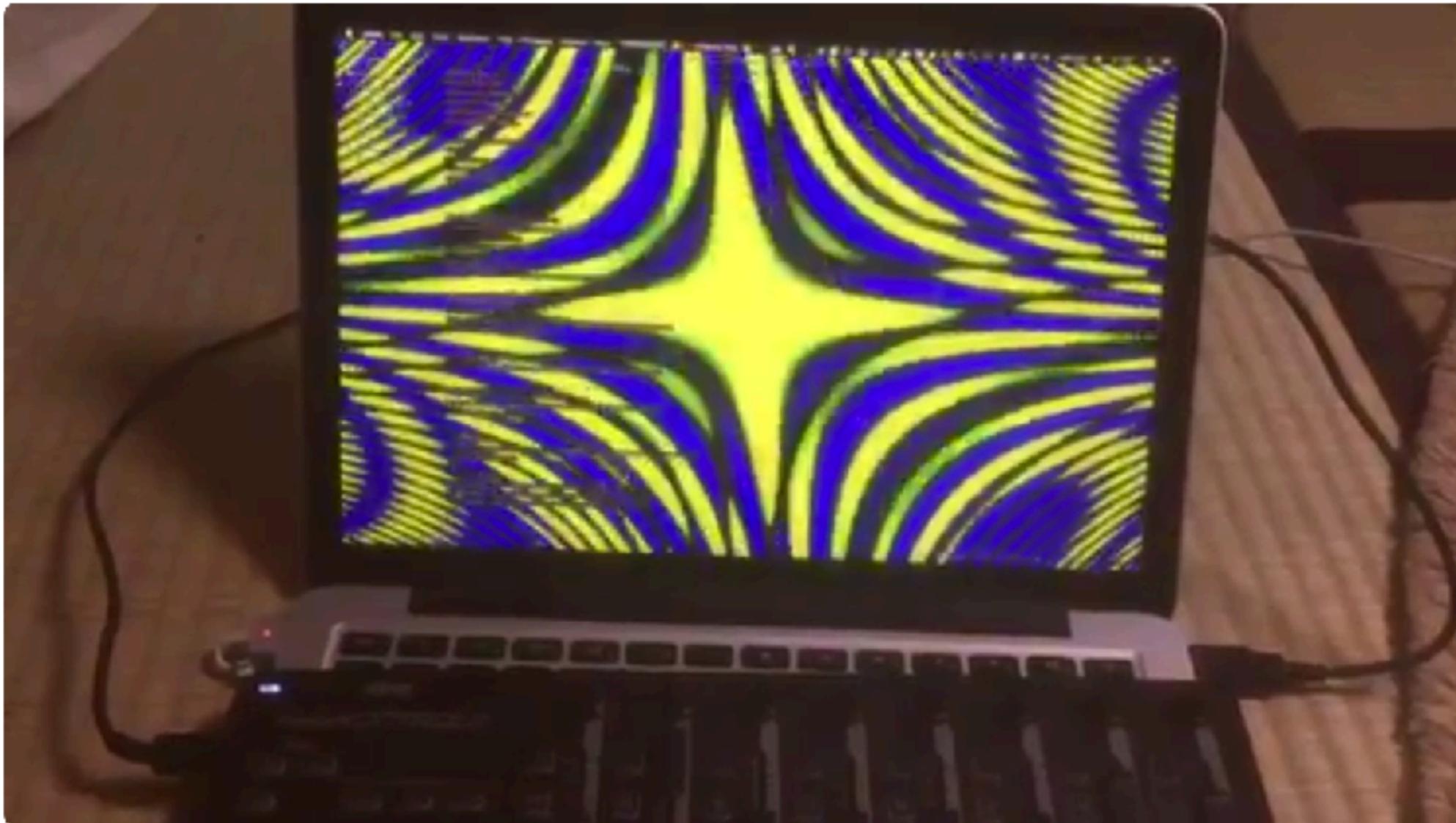


amagi

@amagitakayosi

v

Atom、MIDIコントローラでVJできるようになってきた（進捗



0:16 - 2017年7月15日

6件のリツイート 34件のいいね



note on/offは特別扱いしたい

- note on/offは 1バイト目が 0x80/0x90
- 128×1 のテクスチャを作る





amagita
@amagitakayosi



⚡glsl-livecoder v0.2.0⚡

MIDIノートを取得できるようにしました！
VJしつつシンセを演奏したりできます！！

<atom.io/packages/glsl-...>



ツイート

20:45 - 2017年7月24日

cles.

45件のリツイート 104件のいいね



Your Twee
over the la

マウス入力

- ・ “vec2 mouse” に
マウスの位置が入ってます (0 ~ 1 の範囲)
 - ・ GLSL Sandbox と同じ
- ・ エフェクトの強さとか操作すると
便利です！

Webカメラ入力

- ・ “sampler2D camera” に
Webカメラからの入力が入ってます
- ・ テルミンとして使えるゾイ

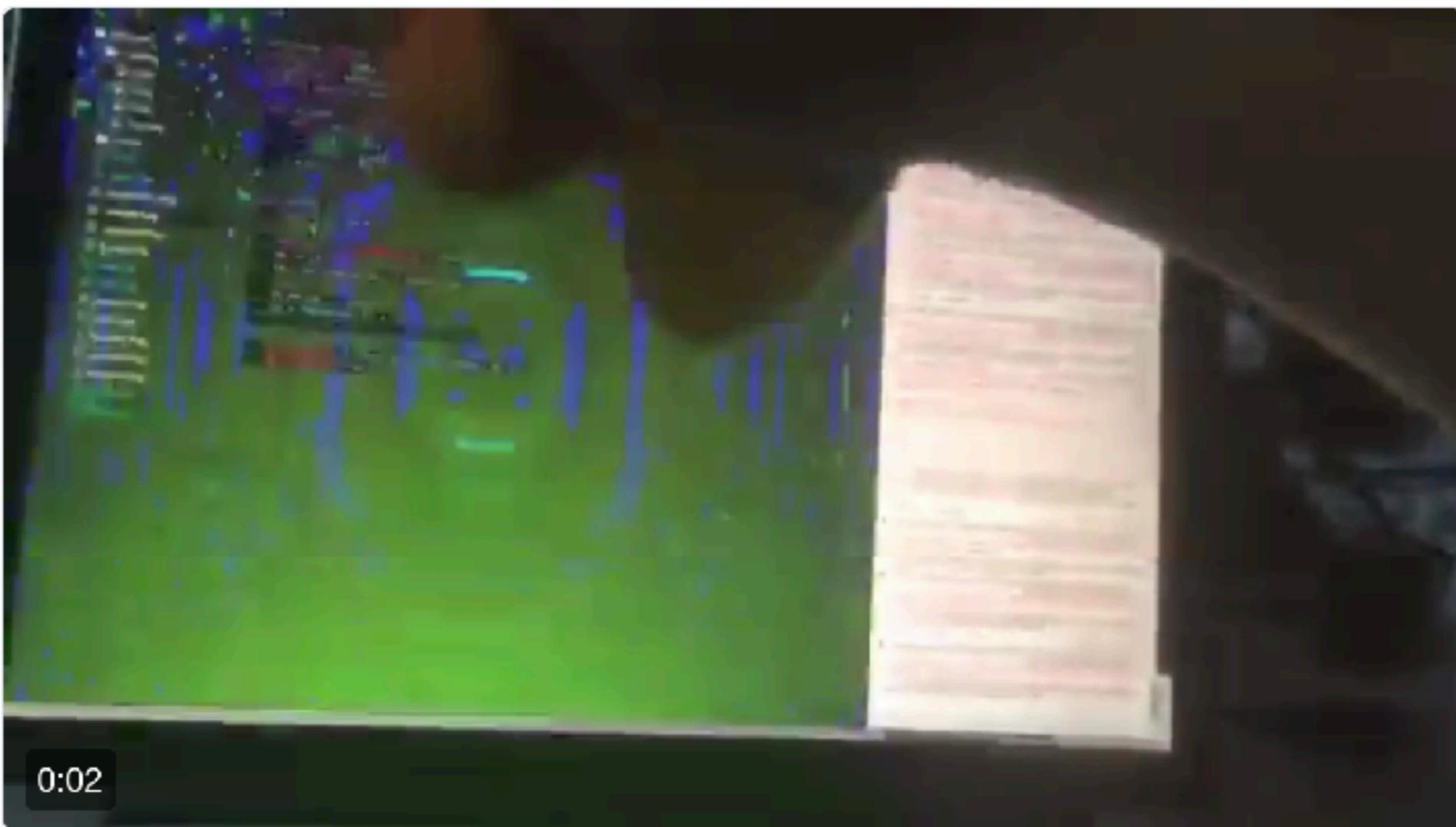


amagi

@amagitakayosi



VJマジで準備時間足りてなくてヤバいけど新
技は出来た



18:08 - 2017年9月10日

5件のリツイート 20件のいいね



その他の入力

- ・ キーボード、ゲームパッド、OSC
 - ・ この講義では割愛します
 - ・ key.frag や osc.frag を見ると使い方なんとなくわかるはず

何か作ってみよう！

- ・ “float volume” か “sampler2D camera” が使いやすいでしょう
- ・ なんでも質問して！
- ・ なんかできたらあとで見せて

おまけ

おまけ

- ・ 頂点シェーダーアート
- ・ glslify
- ・ VEDA.js

頂点シェーダでアート

- ・ <https://vertexshaderart.com> みたいな奴
- ・ 頂点の位置をいい感じにする
- ・ 描画モード切り替えるだけでもウケが良くて
お得だぞ！！！
- ・ “particle.frag” を開いてみましょう

The image shows a GitHub repository interface with a dark theme. On the left, there's a sidebar titled "Project" containing a tree view of files and folders. The main area displays a file named "multipass.frag".

```
1 // PixelNomial��除
2
3 // vertexCount: 169
4 // vertexMode: "TRIANGLES"
5 // "PASSES": 7
6 // "TARGET": "renderBuffer"
7 // "vertex": "particlePosition"
8
9
10
11 precision medium float;
12 uniform float times;
13 uniform vec2 resolution;
14 uniform sampler2D renderBuffer;
15 uniform sampler2D backbuffer;
16
17 void main() {
18     vec2 uv = gl_FragCoord.xy / resolution;
19     vec2 p = (gl_FragCoord.xy + 2. - resolution) / min(resolution.x, resolution.y);
20     p = fract(abs(p));
21     gl_FragColor.gb = texture2D(renderBuffer, p).br;
22     gl_FragColor += texture2D(backbuffer, uv).b * .6;
23     gl_FragColor.r = texture2D(backbuffer, uv + .01).b;
24 }
25
```

At the bottom, there's a status bar with various icons and text: "particle.frag" (file icon), "0 0 0 0 0 0", "16:30", "LF", "UTF-8", "4 Spaces", "GLSL", "git+", "P", "master", "37 files", "2 updates".

particle.frag解説

- ・ 頂点シェーダで図形を描いて
フラグメントシェーダでエフェクトかけてる
- ・ vertexMode で描画モード切り替え
 - ・ TRIANGLES以外のときはvertexCountを
増やしましょう (地味なので)

glslifyでお手軽レイマーチング

- browserifyのGLSL版

```
#pragma glslify: rotate2D = require('./rotate2d.frag')
```

とか書くとバンドルしてくれる

- glslify用のライブラリが
npmで多数公開されてる

めっちゃ沢山ある

Shader Components

glsl-specular-beckmann	@ 1.1.2
glsl-specular-cook-torrance	@ 2.0.2
glsl-diffuse-oren-nayar	@ 1.0.1
glsl-diffuse-lambert	@ 1.0.0
glsl-specular-ward	@ 1.0.0
glsl-specular-gaussian	@ 1.0.0
glsl-specular-phong	@ 1.0.0
glsl-specular-blinn-phong	@ 1.0.1
glsl-perturb-normal	@ 1.0.2
glsl-face-normal	@ 1.0.2
glsl-checker	@ 1.0.1
glsl-earth	@ 1.0.2
glsl-easings	@ 1.0.0
matcap	@ 0.0.1
glsl-inverse	@ 1.0.0
glsl-determinant	@ 1.0.0
glsl transpose	@ 1.0.0
glsl-frobenius	@ 1.0.0
glsl-look-at	@ 1.0.0
glsl-camera-ray	@ 1.0.0

glsl-raytrace	@ 1.0.0
glsl-sdf-normal	@ 1.0.0
glsl-sdf-sphere	@ 1.0.0
glsl-sdf-box	@ 1.0.0
glsl-sdf-primitives	@ 0.0.0
glsl-sdf-ops	@ 0.0.0
glsl-ruler	@ 1.0.0
glsl-turntable-camera	@ 1.0.0
glsl-combine-smooth	@ 1.0.0
glsl-luma	@ 1.0.1
glsl-gamma	@ 2.0.0
glsl-aastep	@ 1.0.1
glsl-dither	@ 1.0.1
glsl-noise	@ 0.0.0
glsl-random	@ 0.0.5
glsl-fog	@ 0.0.1
glsl-fxaa	@ 3.0.0
glsl-lut	@ 1.1.0
glsl-range	@ 1.0.0
glsl-scale-linear	@ 1.0.0

glsl-scale-log	@ 1.0.0
glsl-square-frame	@ 1.0.1
glsl-cornell-box	@ 2.0.4
glsl-read-float	@ 1.1.0
glsl-smooth-min	@ 1.0.0
glsl-film-grain	@ 1.0.2
glsl-hash-blur	@ 1.0.3
glsl-fast-gaussian-blur	@ 1.0.2
glsl-halftone	@ 1.0.4
glsl-crosshatch-filter	@ 1.0.0
glsl-ascii-filter	@ 1.0.1
glsl-hsv2rgb	@ 1.0.0
glsl-hsl2rgb	@ 1.1.0
glsl-blend-overlay	@ 1.0.5
glsl-blend-soft-light	@ 1.0.5
glsl-map	@ 1.0.1
glsl-edge-detection	@ 1.1.0
glsl-atmosphere	@ 1.0.1

VEDAでglslify.fragを使う

- ・ ファイル先頭に `/* { glslify: true } */` を追加
- ・ 実は “fx.frag” はこうなってますね
- ・ “ray.frag” を実行すると……？

Project

- glschool
- git
- Images
- node_modules
- utils
- Videos
- .DS_Store
- vedara
- audio.frag
- backbuffer.frag
- camera.frag
- combination.frag
- combination.vert
- tex.frag
- gamepad.frag
- gif.frag
- image.frag
- key.frag
- mid.frag
- mouse.frag
- multipass.frag
- multipass.vert
- note.frag
- osc.frag
- package.json
- particle.frag
- particle.vert
- ray.frag
- README.md
- server.frag
- shaders1.frag
- shaders2.frag
- shaders3.frag
- vertex.vert
- vj.frag
- yeardeck

ray.frag

```
32
33
34 void main (void) {
35     vec2 p = (gl_FragCoord.xy * 2. - resolution) / min(resolution.x, resolution.y);
36     vec2 uv = gl_FragCoord.xy / resolution;
37
38     vec3 rayOrigin = vec3(0, 0, 18);
39     rayOrigin.x += cos(time * 1.3) * .2;
40     rayOrigin.y += sin(time * .3) * 2.3;
41
42     vec3 rayTarget = vec3(0, 0, -99999999);
43     rayOrigin.z -= time*10.;
44
45     vec3 rayDirection = camera(rayOrigin, rayTarget, square(resolution), 1.3);
46     // rayDirection.xy = rotate(rayDirection.xy, time*.08 +sin(i / rayDirection.z + time * .7)*.3);
47     // rayDirection.x += sin(time * 1.3) * .2;
48     // rayDirection.y += cos(time * .3) * .3;
49
50     vec3 lightDir = normalize(vec3(0, 2, 2.));
51     vec3 light = vec3(.4, .5, 0.93);
52     vec3 ambient = vec3(-.6, -.8, -.9) * .3;
53
54     vec2 collision = raytrace(rayOrigin, rayDirection, 30., 0.1);
55     if (collision.x > .1) {
56         vec3 pos = rayOrigin + rayDirection * collision.x;
57         vec3 normal = getNormal(pos);
58         float diff = clamp(dot(lightDir, normal), 0., 3.0);
59         vec3 c = diff * light + ambient;
60
61         gl_FragColor = vec4(c, 1.0);
62         gl_FragColor *= texture2D(video3, abs(sin(p))) * 2.;
63     }
64     else {
65         gl_FragColor = vec4(0);
66         gl_FragColor = texture2D(video1, uv) * .7;
67     }
68 }
```

Severity Provider Description Line

Severity	Provider	Description	Line
Error	glsl	'textBox' no matching overloaded function found	301
Error	glsl	'square' no matching overloaded function found	441

ray.frag* 0 13 △ 0 0 0 81:18

ray.frag 解説

- ・ npmからレイマーチング用ライブラリをインストールしている
 - ・ sdfの組合せやライティングだけ指定してる
- ・ オブジェクトのテクスチャに動画を使えるぞ！！！！！！！！！！！！

VEDA.js

- ・ VEDAのコア部分をnpmパッケージに分割
- ・ Web開発で使える
- ・ 「背景をシェーダーでサッと書きたい！」
みたいな時にオススメ

<https://veda.gl/>

The screenshot shows a web browser window displaying the VEDA website at <https://veda.gl/>. The page features a dark background with abstract, glowing red and green fractal-like patterns. In the top left corner, there's a small VEDA logo icon. The main title "VEDA" is centered in large white letters. Below it, the subtitle "VJ system on Atom" is written in a smaller white font. A central feature is a large, semi-transparent image of a VJ screen showing a complex, colorful fractal pattern. To the left of the main content area, there's a sidebar with the VEDA logo and social media links for GitHub and Twitter. The sidebar also lists various features and resources: "VEDA for Atom", "Install", "Usage", "Settings", "Features", "Images & Videos", "Audio", "MIDI", "OSC", "WebCam", "Keyboard", "Gamepad", "VEDA.js", "FAQ", and "CONTRIBUTING". On the right side of the main content area, there's a section titled "VEDA is an Atom package for VJ / Livecoding with GLSL." which includes a screenshot of the Atom code editor showing GLSL code for a shader. The overall aesthetic is modern and tech-oriented.

できの方いますか！！

最後に

- ・ VJたのしいよ！！
- ・ VEDAの感想や質問、新機能のリクエストは
GitHubか、ハッシュタグ #VEDAJS で
お待ちしてます～

ENVOI
ENGAGE