# HOW TO PREPARE WEBGL FOR MAKING WEBSITE

# 自己紹介

## 東倉 司

SHIFTBRAIN.inc

Interactive Designer

Awwwards Jury member (2016〜)

## 今日お話しすること

- webGLを使ったwebサイトから学ぼう
- 楽にwebGLをwebサイトに取り入れるための下準備
- 非対応環境やプログレッシブエンハンスメントを定めよう
- タスクランナーを取り入れてシェーダを書こう

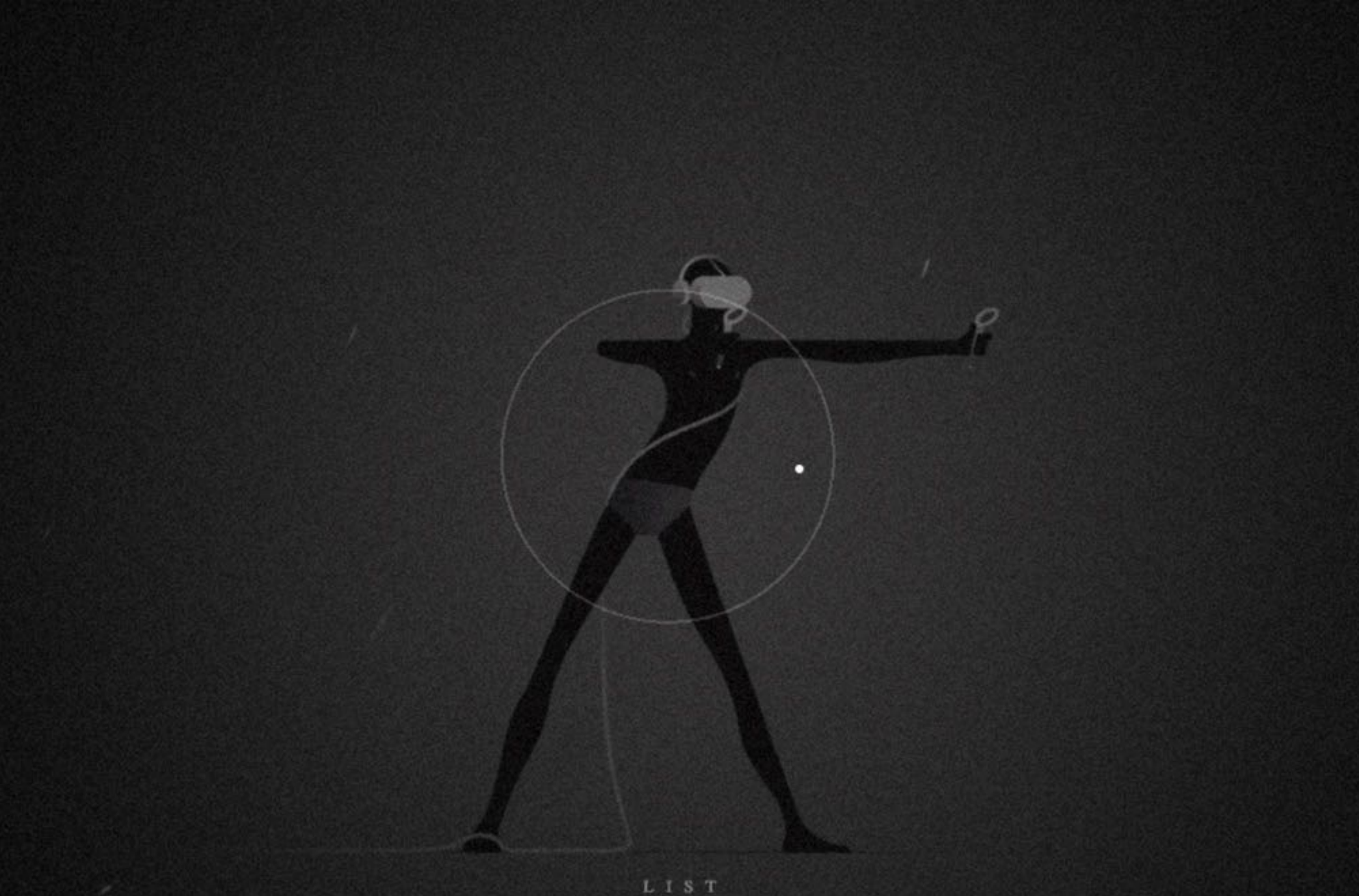
webGLを使ったwebサイトから

学ぼう


# AIRMAN CHALLENGE

by Active Theory


- ド派手な演出を背景のwebGLに任せ、可能な限りhtmlのマークアップでUIとテキストを演出している。
- 情報をwebサイトに持たせるために重要なポイント。全てをwebGLの中に入れ込むとwebサイトが持つ情報が損なわれてしまう。




# Immersive Garden's wishes for 2017

by Immersive Garden


- 動画をテクスチャとして読み込んでwebGLの演出と上手くマッチさせている
- 中のアニメーションを動画担当にパスできるため、インタラクションとポストエフェクトにフロントエンドが集中できる。アニメーションの担当を上手く切り分けた一例。

みなさんが今までの授業で体験してきたように、

手法が同じでも他の手法と組み合わせたり、

素材を工夫することで大きく印象を変えることができます。

とにかくたくさんのwebGLサイトを見て触ることで

表現の引き出しを増やすことが大事です

AWWWARDS®

CHECK OUR WEBSITE !!

# 楽にwebGLをwebサイトに取り入れるために

# 下準備すること

非対応環境や

プログレッシブエンハンスメントを定めよう

## 非対応環境やプログレッシブエンハンスメントを定めよう

非対応環境を定める上での判断基準とは？

webGLが使用可能かどうか（大前提ですね！）

+

- ターゲット層
  ー　ブラウザのターゲットが若干変わってくる
- コンテンツ内容
  ー　必要なスペックの参考にする

Ex :

**ゴリゴリの3Dモデルを使用したwebサイトを作りたい**

iOS〜7やAndroidの4.4以下は画像とテキストをメインとしたwebサイトを表示させる。iPhone〜5や古いiPadのパフォーマンスは保証しない。

Ex :

## 動画に動的にエフェクトをかけるwebサイトを作りたい

IEでは動画ではなく画像を表示させるようにする。モバイルやパッド系の端末のために動画を再生するために1アクション挟む。

## 非対応環境やプログレッシブエンハンスメントを定めよう

重要なのはある程度のリスクが見えてる見えてないに関わらず、
開発の必要工数に大きく関係するので、ある程度の憶測も交えて必ずディレクター・デザイナーと密に連携を取りながらコンテンツやデザインを策定すること。特にデザイナーとのやり取りでは「この案件の条件の中でやれること・やれないこと」を話し合っておかないと血反吐を吐くことになります。提案前にしっかり擦り合わせをした方がいいです。

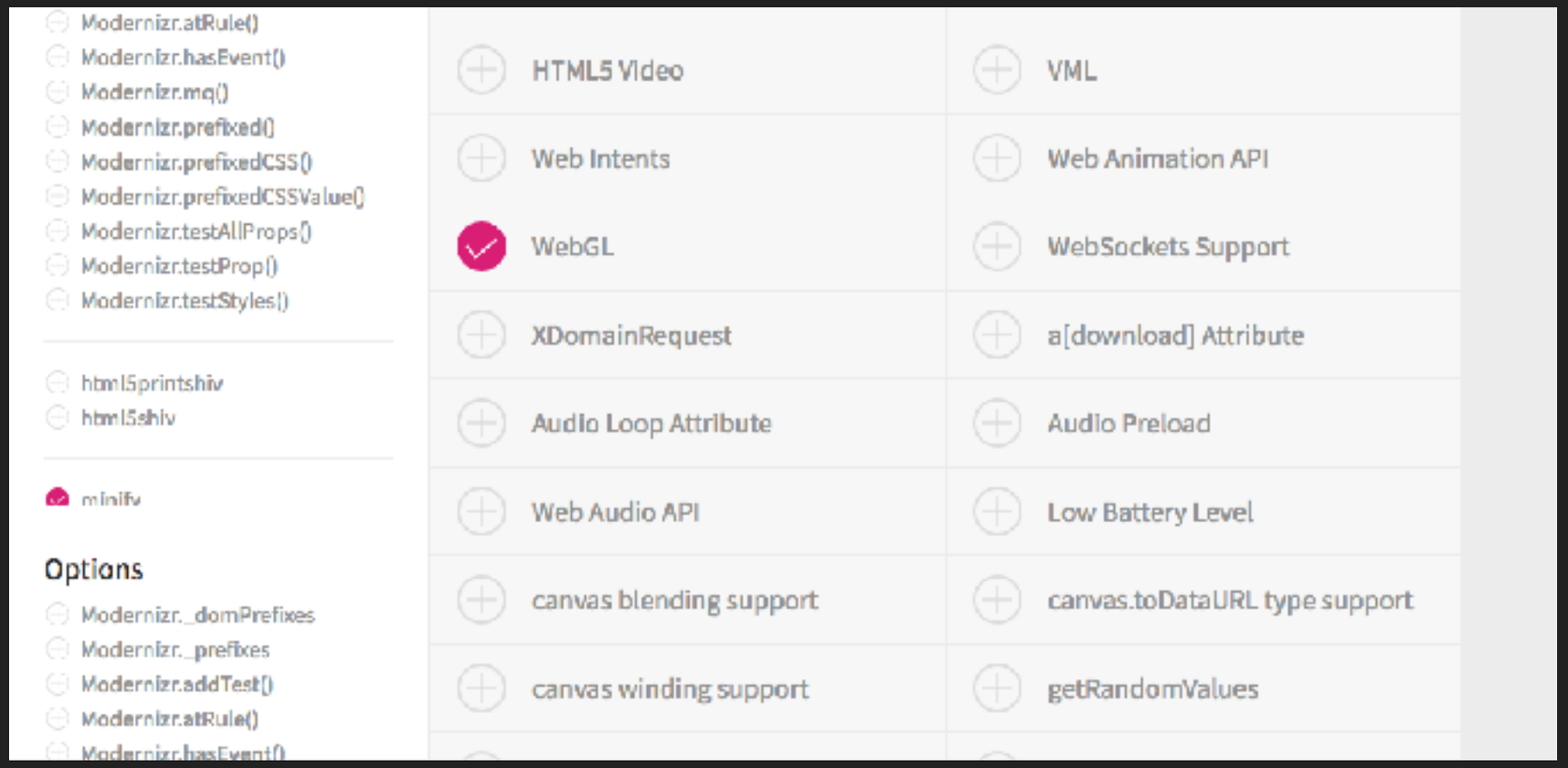
**技術的な制約やリスクは僕たちエンジニアが説明しないと分かりませんし、予想もできません。**

**テクニカルディレクター的なポジションになったつもりで**
**しっかり環境を策定しましょう！**

## 非対応環境やプログレッシブエンハンスメントを定めよう

次は実装面のお話しです。jsで分岐を書くのですが、ここはライブラリを使って時間短縮しましょう。

Modernizr




WebGLにチェックを入れてBUILDしましょう。

あとはjsを読み込めばhtmlにwebglまたはno-webglクラスが付加されます。

## 非対応環境やプログレッシブエンハンスメントを定めよう

様々な原因に対処するため、あと少しだけ自分でjsを書いて追加します。

```
let notWebGL = function(){
  // WebGL非対応時の処理を記述

};

if(document.getElementsByTagName('html').classList.contains('no-webgl')){
  notWebGL();
}

// three.jsのとき
try{
  let renderer = new THREE.WebGLRenderer();
}catch (e){
  notWebGL();
}
```

これでwebGL対応非対応の基本的な処理は終わりです。

## 非対応環境やプログレッシブエンハンスメントを定めよう

そこに追加でブラウザの条件等を加えていきましょう

### ubu.detect

jsを読み込んだ時点でブラウザの結果がubu.detectに結果が返ってきます。

```
if(ubu.detect.browser.ie){
  console.log('IEさん、動画テクスチャはちょっと…無理ですね…')
}
```

ブラウザで処理を分岐させる際に利用してください。

# タスクランナーを取り入れてシェーダを書こう

正気じゃない…

えっ？直書き？

シェーダ増えたらアカンやつや

```
<script id="vs" type="x-shader/x-vertex">
attribute vec3 position;
attribute vec4 color;
uniform   mat4 mvpMatrix;
varying   vec4 vColor;

void main(){
    vColor = color;
    gl_Position = mvpMatrix * vec4(position, 1.0);
}
</script>
```

そこは汚したくない僕の聖域

シンタックスハイライトは…？

## タスクランナーを取り入れてシェーダを書こう

どうしても.vertと.fragファイルで書きたいおじさん

- ファイル分けたい
- シンタックスハイライトないのは苦痛
- 補完欲しい
- htmlに埋め込まれるのは回避したい（人によりけり）

## 方法 ①

Jadeを使って直接ファイルを埋め込んでしまう

```
script#vertexShader(type='x-shader/x-vertex')
    include test.vert
```

簡単！けど結局HTMLの中に埋め込んでるやないかい！！

## 方法②

three.js側でテキストファイルとして読む

```
let fragTxt;

$.ajax({
  url:'sample.vert',
  success: function(data){
    fragTxt = data;
  }
});

let myMaterial = new THREE.ShaderMaterial({
  fragmentShader: fragTxt
});
```

いいかも！でも、ロード管理するの面倒だなぁ…
タスクランナー等が使えない・使うまでもない時は◎

# 方法 ③

**browserify** + **stringify** で

テキストファイルとしてインポートする

# ここでようやくタスクランナー登場


http://gulpjs.com/

## タスクランナーとは？

- Sass / Jade / Typescript etc のコンパイル
- ファイル更新の監視
- ファイルの結合・圧縮　などなど

現代の面倒なフロントエンド界隈を生き抜くための
僕らのマストアイテム（webpackは割愛）

**gulpの解説授業ではないので、**
**説明は割愛させていただきます！**

- gulpの基本的な使い方（gulp.jsの基礎をしっかり理解する）
- gulp.js を今一度キチンと！gulp.js 導入基礎

配布されたサンプルと見比べながら
読んでみてください

# browserifyとは？

Node.jsのモジュールシステムを使うためのツール

webサイト構築ではモジュール間の依存解決やファイル結合に用いられる

# モジュール間の依存解決…？

```
<script src="a.js"></script>
<script src="b.js"></script>
<script src="c.js"></script>
```

a.jsとb.jsを読みこみ終わってないと

c.jsが動作していない場合、c.jsは

a.jsとb.jsに依存しています。

各jsを結合する際、結合順を間違ると同様に動作しない。
数が少ない場合はいいが、b.jsはa.jsに依存して、c.jsはd.jsとa.jsに依存して…と増えていくと手動で管理するには限界がある。

**あとグローバルオブジェクトを汚染したくない！！！**

# 図示するとこんな感じ

a.js

依存

b.js

```
const a = require('a.js');
// aにアクセスできる
a.hoge();
```

依存

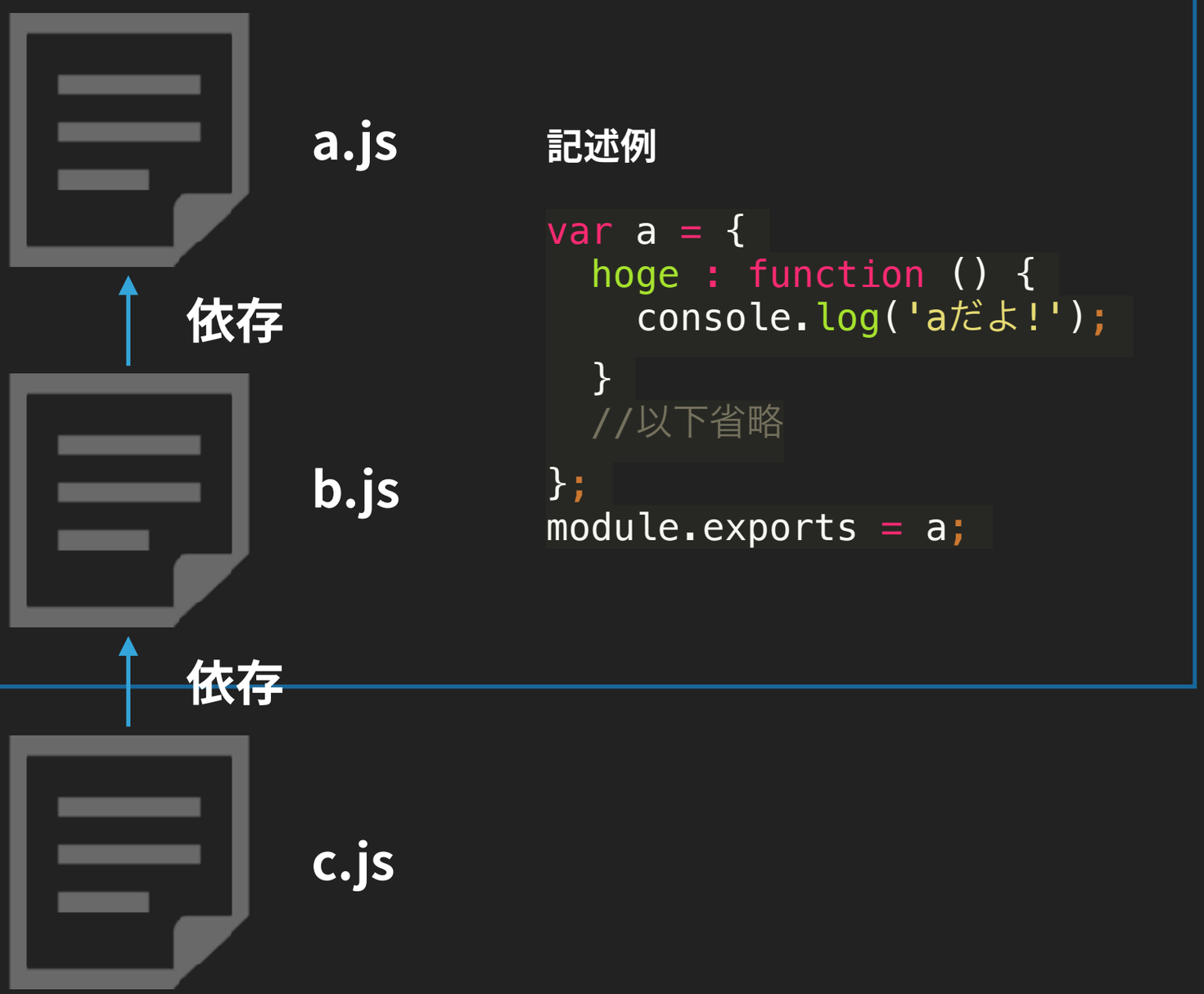
c.js

```
const b = require(‘b.js');
// bにアクセスできる
b.hoge();
```

1つのjsに書き出してくれる

dist.js

# requireされるブツはexportされている必要がある




1つのjsに書き出してくれる

dist.js

# タスクランナーを取り入れてシェーダを書こう

## どうして使うの？？

# タスクランナーを取り入れてシェーダを書こう

## webGLゴリゴリだとjsが複雑になりがち…


- ロゴのアニメーションのパーツ
- シェーバーのモデル
- 背景のパーツ
- カメラとシーンの管理　などなどなどなどなどなどなど

# 1つのjsに書くのは苦行

# 複数人作業だと競合祭

# 管理しやすくなる・分担しやすくなる

logo.js

Aさん担当

model.js

Bさん担当

bg.js

Cさん担当



dist.js

# webGLに限った話ではない

mordal.js

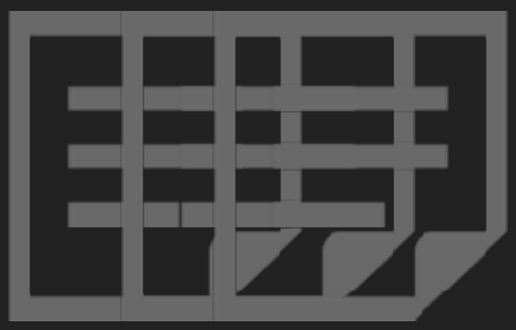
Aさん担当

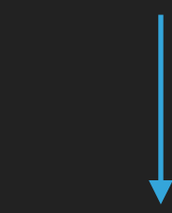
accordion.js

Bさん担当

canvas.js

Cさん担当





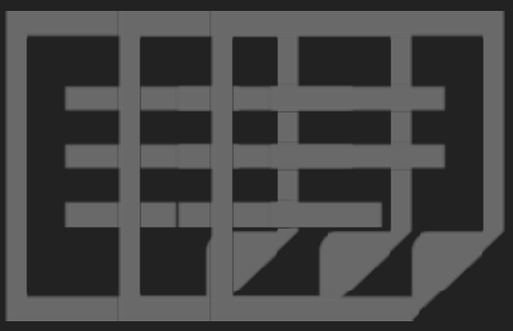
dist.js

# そしてシェーダも入れたい

xxx.vert

xxx.frag

xxx.vert



dist.js

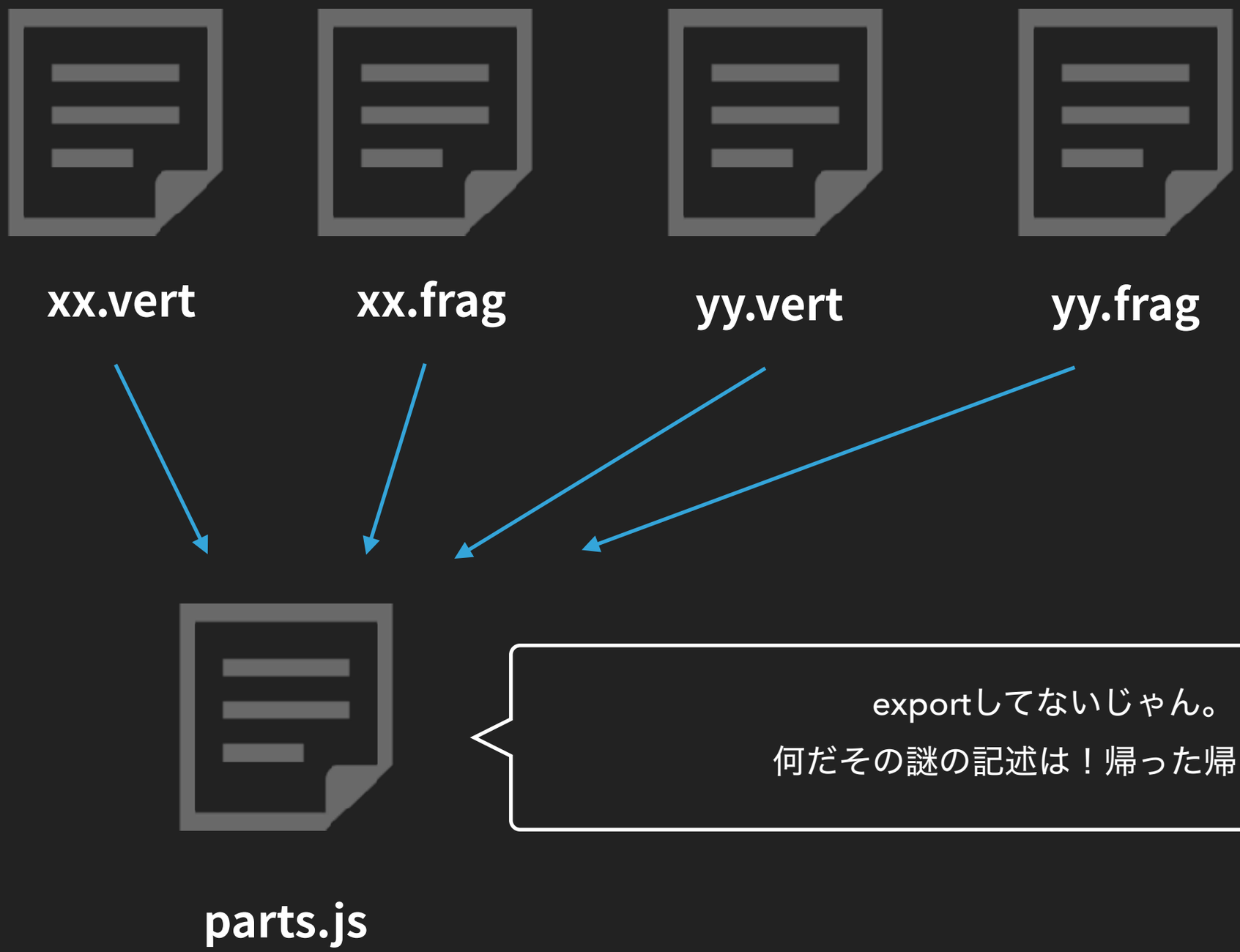
ここでようやく

本命のstringify

# stringifyとは？

あらゆるテキスト系のファイルを強引にStringの形式にしてexportしてくれるbrowserifyのパワー系transformモジュール

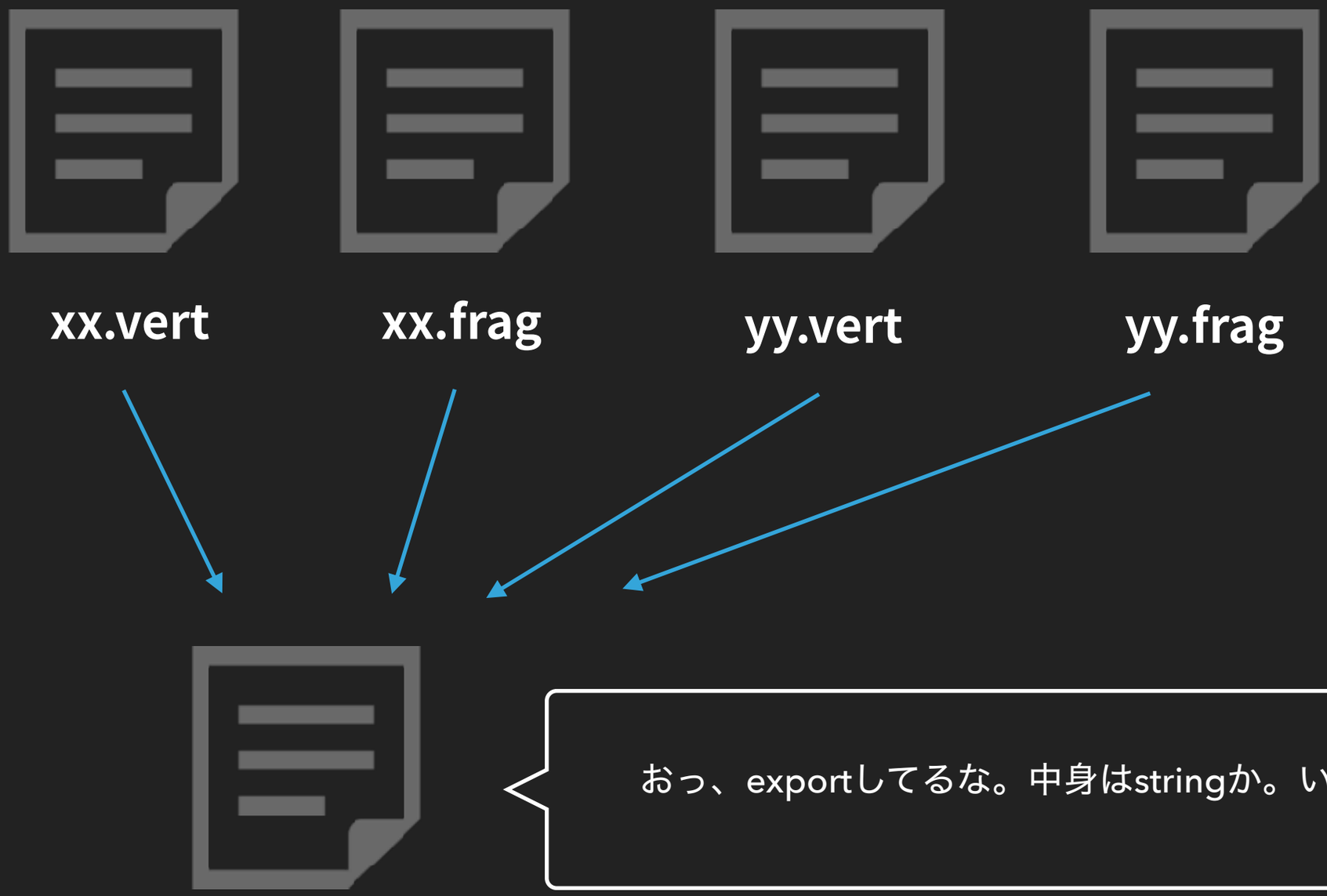
# 今目指している理想の形



## stringifyを通してstringに変換される

```
module.exports = "precision mediump float;
\n\nuniform sampler2D hogeTexture;\nvarying vec2
vUv;\n\nvoid main(){\n  vec4 textureColor =
texture2D(hogeTexture, vUv);\n  gl_FragColor =
vec4(textureColor.rgb,1.0);\n}\n";
```

# stringifyを使う

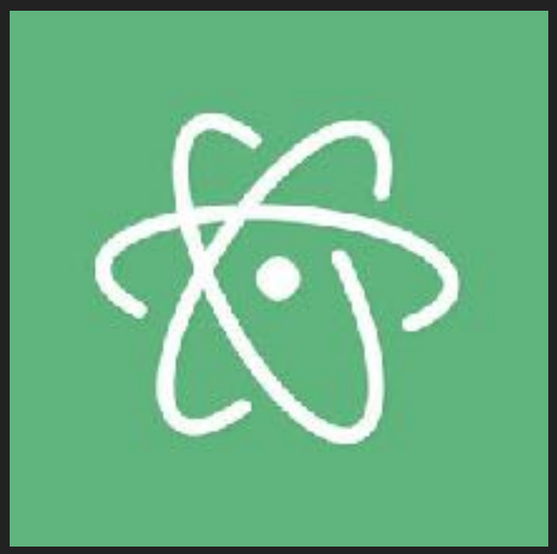


# 今目指している理想の形

# 補完とかシンタックスハイライト

## が適用できる環境を用意しよう

# 個人的おすすめ



## Atom

Preference > Packages

- autocomplete-glsl
- language-glsl

2つのパッケージを入れればOK

# 実際に試してみるのが一番！

## さくっとthree.jsで実践しよう

## まずは黒い画面を開き、ダウンロードして解凍したフォルダの中に移動しましょう

```
cd XXX
```

## そして必要なnpmモジュールを全部ダウンロードしましょう

```
npm install または sudo npm install
```

## あとはgulpのコマンドを打つだけ

```
gulp
```

## さらなる応用技へ

**この未完成のサンプルを**
**どんどん改良していきましょう！**

- three.jsをnpmでインストールしてモジュールとして使おう
- モジュールとして読み込んだthree.jsを利用して各パーツのモジュール化もやってみよう
- browser-syncを取り入れてプレビューを楽にしよう

**まずはやってみなきゃ何も始まらない**

**失敗を恐れず、まずは作って作って作りまくろう**

**当講座で基礎を学んでいるはずなので、実践あるのみ**

# Thank you for listening !!