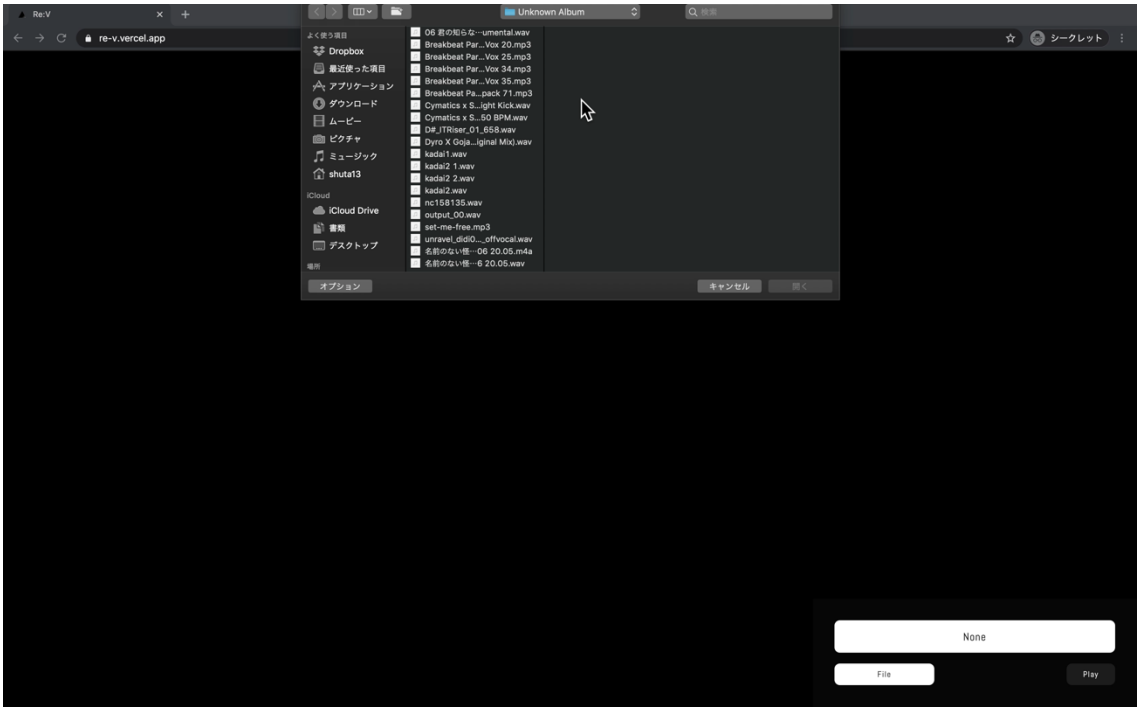
メディアプロジェクト演習 2 自主制作・調査研究レポート

平井 柊太

- 作品(タイトル ：Re:V – レヴ)

デモ映像 ：



URL : https://re-v.vercel.app/
Git リポジトリ(ソースコード等) : https://github.com/shuta13/rev

- なぜこのテーマを選んだか？

音声と画像両方の技術を用いた Web アプリケーションを制作したかったという個人的な願望があったため.
また，作品の価値として Web 上で動かせるためにインストールなどの手間が不要であり，インターネットに接続出来る環境(と mp3 や wav 等の音源)を持っていれば気軽にビジュアルジョッキー(後の質問の項でこの単語について説明する)を楽しむことが出来る点.

- 特に努力した点は何か？

JavaScript と Web Audio API による FFT でスペクトルを抽出した後，GLSL 側にテクスチャとして渡す処理の実装を行った点.
処理の詳細は以下にソースコードの一部と GitHub のコードへのリンクを掲載し，説明する.

以下の図 1, 2 には FFT によって読み込んだ音源のスペクトルを抽出する処理と GLSL に渡すためのテクスチャの生成処理を示している.

```
const audioConfig: AudioConfig = {
  fftSize: 512,
  listener: null,
  audio: null,
  file: null,
};
```

https://github.com/shuta13/rev/blob/master/components/partials/Player.tsx#L44

図 1. FFT の初期値

```
const getSpectrumByFft = ({ analyser, uniforms }: GetSpectrumByFftParams)
=> {
  analyser.getFrequencyData();
  uniforms.audioTexture.value.needsUpdate = true;
  fftRAFId = requestAnimationFrame(() => {
    getSpectrumByFft({ analyser, uniforms });
  });
};

const handleOnClick = (uniforms, file) => {
  if (file === null) return

  if (audioConfig.audio.isPlaying) {
    audioConfig.audio.pause();
  } else {
    const audioLoader = new AudioLoader();
    audioLoader.load(audioConfig.file, (buffer) => {
      audioConfig.audio.setBuffer(buffer);
      audioConfig.audio.play();
```

```
    const analyser = new AudioAnalyser(
      audioConfig.audio,
      audioConfig.fftSize
    );
    uniforms.audioTexture.value = new DataTexture(
      analyser.data,
      audioConfig.fftSize / 2,
      1,
      LuminanceFormat
    );
    getSpectrumByFft({ analyser, uniforms });
   });
  }

  if (fftRAFId !== 0) {
   cancelAnimationFrame(fftRAFId);
  }
};
```

https://github.com/shuta13/rev/blob/master/components/partials/Player.tsx#L57

図 2. FFT の処理・DataTexture 関数によるテクスチャの生成

図 1 は FFT の初期値や FFT のターゲットにする音源を持たせるための連想配列である.
図 2 はユーザのクリック操作(Play ボタンを押す操作)をトリガーとした handle OnClick 関数によってアップロードされた音源に対して FFT を行っている.
今回は FFT の際のサンプル数を 512 と設定し行った. スペクトルの抽出自体は Three.js の組み込み関数である AudioAnalyser.getFrequecyData(https://threejs.org/docs/#api/en/audio/AudioAnalyser.getFrequencyData)で行っている.
スペクトルの値を analyser に格納した後, THREE.DataTexture(https://threejs.org/docs/#api/en/textures/DataTexture)によってテクスチャに変換し, GLSL に audioTexture という名前の uniform 変数として以下の図 3 のように渡している.

```
#ifdef GL_ES
precision highp float;
```

```
#endif

uniform vec2 resolution;
uniform float time;
uniform sampler2D audioTexture;
```

https://github.com/shuta13/rev/blob/master/assets/shaders/index.frag#L7

図 3. フラグメントシェーダーの uniform 変数等の記述

音源の再生に合わせたリアルタイム処理を行う必要があったため，window.requestAnimationFrame(https://developer.mozilla.org/ja/docs/Web/API/Window/requestAnimationFrame)を用いて処理を行ったが，DOM のレンダリングプロセスを管理している React との相性が悪かったため非常に苦労した．

また GLSL にテクスチャとして渡したスペクトルは図 4 のようにシェーダーを記述する際に用いている．

```
float map(vec3 p) {
  // float i = time;
  float i = texture2D(audioTexture, vec2(0.3, 0.5)).x;

  float d1 = length(p) - 1. * i;

  //mat2 r = rot(-time / 3.0 + length(p));
  mat2 r = rot(time * 2.0 + length(p));
  p.xy *= r;
  p.zy *= r;

  p = abs(p);// - time;
  p = abs(p - floor(p + .5)) *  2.5 * i;

  //r = rot(time);
  //p.xy *= r;
  //p.xz *= r;

  float l1 = length(p.xy);
  float l2 = length(p.yz);
```

```
  float l3 = length(p.xz);

  float g = 0.01;
  float d2 = min(min(l1, l2), l3) + g;

  burn = pow(d2 - d1, 2.0);

  return min(d1, d2);
}

void main() {
  vec2 uv = gl_FragCoord.xy / resolution.xy - .5;
  vec3 ro = normalize(vec3(uv, 1.5));

  vec3 ta = vec3(0, 0, -2);

  float t = 0.;
  for (int i = 0; i < 40; i++) {
    t += map(ta + ro * t) * 0.4;
  }

  gl_FragColor = vec4(burn, burn, burn, 1.0);
}
```

https://github.com/shuta13/rev/blob/master/assets/shaders/index.frag#L23

図 4. GLSL によるフラグメントシェーダーの記述

フラグメントシェーダーによって画面のピクセル毎に色を変化させることが出来るため，リアルタイムに変化するスペクトルを用いて色を変化させることでデモ映像のような映像を作り出している．

- この演習を通じて得られたもの，今後の学習に活かせるものは何か？

C 以外の言語を通した Web 上での音声や画像処理技術について多くの知見が得られた．特に Web Audio API は何も音声処理に対する知識が無い状態だと若干使うことが苦しく思われたため，メディア実験 2 の音声情報処理の知識が活きたように思う．

また，ビジュアルジョッキーの際に使用される Web サービス
(https://www.shadertoy.com/)やアプリケーション
(https://atom.io/packages/veda)の内部実装を読むことが出来，どのように機能が実装されているのかを学ぶ良い機会になった．
今後は研究に取り組む際にも今回得られたような音声や画像処理に関する知識や調べて実装を行うプロセスを活かしていきたいと思う．

- 質問に対する解答

Q: JavaScript による音声スペクトル処理は，なぜスペクトル射影して処理を行うのですか？　画像に変換する程度なら時間波形処理でも良いように思いますが，実部と虚部の複素数をどのようにしてこの処理にて扱っているのか，技術的な視点から説明してください．

- 画像に変換する程度なら時間波形処理でも良いように思いますが

そういった実装も可能だと思いますが、私は GLSL で sampler2D を用いた処理を先に思いついたのでそのまま実装を行いました。提案された方法が良いパフォーマンスである場合そちらに変更するかもしれません。要検証です。

- 実部と虚部の複素数をどのようにしてこの処理にて扱っているのか，技術的な視点から説明してください．

AnalyserNode.getByteFrequencyData() :
https://developer.mozilla.org/en-US/docs/Web/API/AnalyserNode/getByteFrequencyData によって入力音声波形からスペクトルを計算し、絶対値の 2 乗を取ったパワースペクトル(もしくは対数パワースペクトル)を出力として返すように思われます。

Q: 本作品の制作にあたって特に苦労した点を教えてください．
特に努力した点は何か？ の項でも述べた通り，FFT -> スペクトルの計算 -> GLSL にテクスチャとして渡す 実装を行った点です．

Q: 今後の課題で説明されていた感情認識機能は,顔など画像から情報を入力する方針ですか．
概ねそういった方針になります．Web カメラ等で取り込んだ顔の 3D 画像に頂点をマッピングして，頂点の移動した程度で感情を認識するような形での実装を考えています．

Q: Atom 上で動作していた従来技術と作成された Web アプリケーションではパフォーマンスにどの程度差異がありますか.

Atom 上で動作していたアプリケーション(以下 VEDA)はデスクトップアプリ(オフラインアプリ)のような形で動作するためネットワーク環境にあまり依存しませんが，私が実装したアプリの場合ブラウザ上で動くため，インターネット環境(回線の速度等)にかなり依存してしまい，状況によってはパフォーマンスが落ちてしまうような差異があります．これは
PWA(https://developer.mozilla.org/ja/docs/Web/Progressive_web_apps)によってオフラインでも動くようにして対応を行う予定です.

Q: 音声スペクトルの画像を最終的な CG に変換する部分について，実装の手順や関数などの記載がありましたが，その中身の仕組みについて詳しく説明していただけますでしょうか.

特に努力した点は何か？の項をご参照お願いします.

Q: VJ（ビジュアルジョッキー）とは何ですか？

こちらをご参照お願いします
(https://kotobank.jp/word/%E3%83%93%E3%82%B8%E3%83%A5%E3%82%A2%E3%83%AB%E3%82%B8%E3%83%A7%E3%83%83%E3%82%AD%E3%83%BC-679666)

従来のディスクジョッキーと映像によるパフォーマンスを組み合わせた新たなパフォーマンスをビジュアルジョッキーと呼びます.

Q: Three.js の関数をいくつか上げていますが，自分が作成した（？）アプリケーションのどのような部分に使用したのか．また，どのような物を作ったのかシステム構成図など，全体的な作りが分かる物がほしい.

- Three.js の関数をいくつか上げていますが，自分が作成した（？）アプリケーションのどのような部分に使用したのか

いくつか挙げた関数について，特に努力した点は何か？の項で使用方法等説明しましたのでご参照お願いします.

- また，どのような物を作ったのかシステム構成図など，全体的な作りが分かる物がほしい.

GitHub に全てのコードをアップロードしていますのでこちら
(https://github.com/shuta13/rev)をご高覧ください.

また，デモはこちら(https://re-v.vercel.app/)に公開していますので合わせてご高覧ください．(Google Chrome, Firefox 最新版での閲覧を推奨します)

システム構成(アーキテクチャ図)は以下の図 5 のようになっています．
このアプリケーションのデプロイには Vercel(https://vercel.com/)を使用しています．使用技術は Next.js(https://nextjs.org/)・TypeScript(https://www.typescriptlang.org/)・Three.js(https://threejs.org/)です．

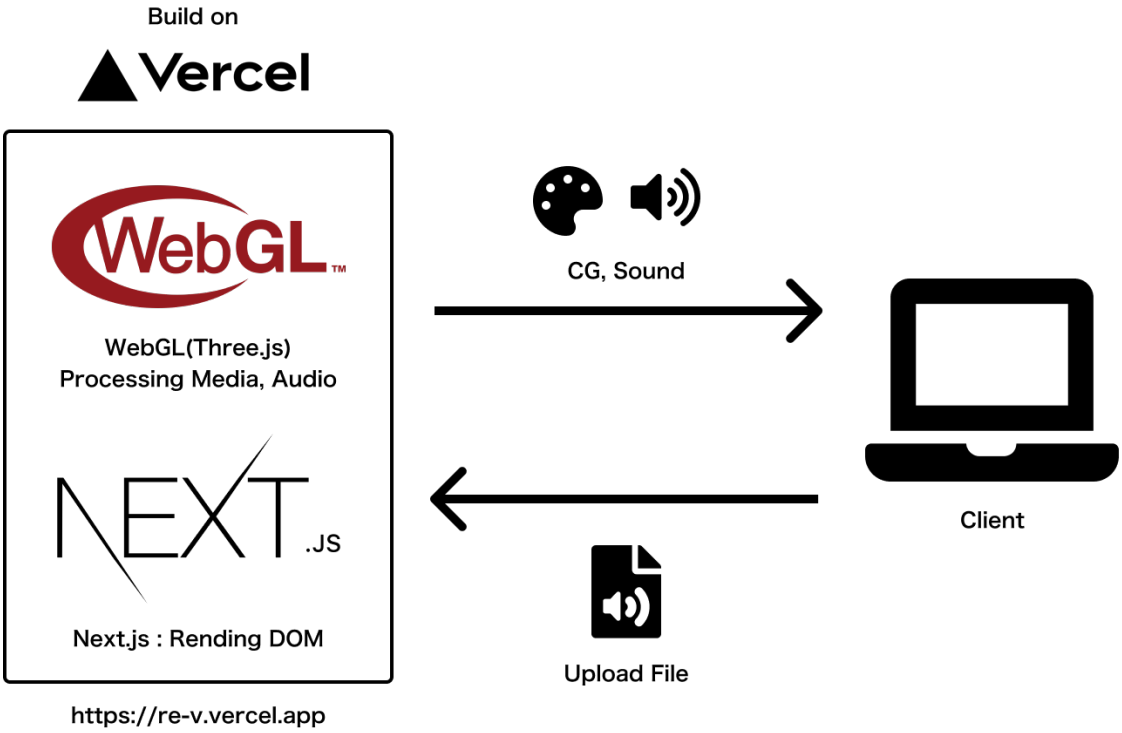


図 5. Re:V のアーキテクチャ図