WebGLスクールプラスワン講義

WebGLを使ったWebサイト制作

自己紹介

吉村 太郎 (ヨシムラ タロウ)

株式会社バケモノ / ディベロッパー（フロントエンド）

https://baqemono.jp
https://x.com/ysmrt6

WebGLには制作案件で年に数回くらいのペースでお世話になっています。
よろしくお願いします。

WebGLを活用してどんなWebサイトを制作しているか

私の普段の業務では、主に2D上にテキストや画像といった情報をのせていく、いわゆる一般的なウェブサイトの制作を行うことが殆どです。ゴリゴリの３D体験を作ったり、大量のデータを何かしらのビジュアルで可視化するといった高度で複雑（？）な実装をする機会はあまりありません。
あくまでウェブサイト内の賑やかしやアクセントになる要素の1つとして、WebGLを使ったビジュアルやインタラクションを作成するという選択肢があるというようなイメージです。

ここ数年でWebGLを組み込む機会は何度かありましたが、業務では主に２つのパターンのどちらかで制作をしました。

1. KVや背景といったある限定された箇所での表現

サイトの顔やコンセプトに合わせた要素として、1箇所にフォーカスしてインパクトのある表現を作成します。

- https://awards.smartnews.com/
- https://www.smarting.jp/
- https://asmobius.co.jp/
- https://baqemono.jp/404/

2. サイト内にあるhtml要素とWebGL(ポリゴン)を同期させる表現

ページ内にあるhtmlで作られた世界の各所にポリゴンが配置されているような見せ方になります。

一見するとhtmlとcssだけで作ったように見えますが、隠し要素でWebGLインタラクションを仕込んでいます。

- https://hyogen.design/projects/civi/
- https://earcouture.jp/
- https://newfolk.jp/projects/

※ 実績非公開の案件もあるので、SNSなど外部への共有はお控えください。

講義内容

どちらのパターンの実装も良し悪しはありますが、今回は2つ目に紹介した**「サイト内にあるhtml要素とWebGLを同期させる表現」**に焦点を当てた解説をできればと思っています。

▼ オススメしたいと思った理由（ 参考: https://newfolk.jp/projects/ ）

- **複雑な数学の知識が無くても挑戦できる：**
  主に二次元的な計算を行うため、数学の知識に自信がなくてもなんとかなったり（ならなかったり）します。
- **実案件でも実装者主導で制作チーム内で提案しやすい：**
  静的な２DレイアウトにWebGL表現を追加するので、デザイン作成後でも実装者側から提案がしやすいです。
- **他の３D実装と比べてリスクが低いのでチャレンジしやすい：**
  キービジュアル表現などの必須要件になる実装と異なり、WebGLの組み込みがうまくいかなかったとしても、デザイン自体は成り立つことが多く、チャレンジ要件としてデザイナーからもOKをもらいやすいです。

講義で紹介する表現は簡単なものになりますが、それらの考え方や仕組みをベースに応用すれば、表現の幅は広がっていくはず（・・だと思います）。

解説の前に

今回紹介するコードは、three.jsをラッパーライブラリとして利用します。
ライブラリを採用する理由は、通常業務において素のWebGL APIを1から使うとなると考えることが多くなり、
正直しんどいからです（僕の場合はしんどいです・・）。
three.jsベースでも使い方次第ではカスタムできる範囲を広げていくことはできるので、まずは**「頼れるところはライブラリの力を借りながら要所で自分なりの工夫をしてみる」**という流れを紹介できればと思います。

解説用のコードはhtml、js、shaderを1セットに、2ページ分を用意しています。
1ページ目は入門編となるベースのコードの確認、2ページ目は実践編といった流れになります。
また、コードは本講義で杉本先生にシェアいただいたものをベースに作成しています。
新たに追加もしくは変更になった箇所にはコメントを残していてるので、その辺りを中心に話していきます。
割愛する箇所もあると思うので、わからない箇所があれば後ほどDiscordやDMで質問してください（できるだけお答えしたいです）。

01 基本となるクラスを作成

ゴールは、htmlとcssで作られた2D空間の中に、WebGLからコントロールできるmeshが混在しているように見える世界を作り上げることです。

何も考えなければ、必要となる要素（mesh）の数分canvasを用意して、ページ内の置きたい場所に設置するだけで課題は解決できる気がするのではないでしょうか。
しかし、パフォーマンス管理の観点から、1ページの中で利用するcanvas(WebGLコンテクスト)の数は最小限に抑えられることが理想です（他にも理由はありますが割愛します）。今回はmeshの数 = canvasの数ではなく、1canvasの中で、必要な数のmeshを作成し、mesh単位で情報をコントロールしていく仕組みを作ります。

次に具体的な作成の流れを説明します。

1. html要素の設置とスタイル調整:

meshを置きたい場所に置きたいサイズのhtml要素を設置しcssでスタイリングしておきます。

2. canvas周りの準備:

WebGLコンテクストの作成は、本講義と同様の流れで作成します(canvasサイズ = windowサイズで)。

また、ターゲットとなるhtmlの数分だけmeshを作成し、両者の情報を共有できるようにしておきます。

3. html要素とメッシュの同期:

html要素のサイズと座標位置を取得し、その情報を基にmeshのscaleとpositionを調整します。

4. カメラの位置調整:

カメラとmesh(z軸上の原点)との距離を計算し、WebGLの3D座標とhtmlの2D座標を一致させます。

これらの調整により両者のオブジェクトの位置関係が視覚的に整合性を保つようにします。

5. 最後にシェーダのカスタマイズ (RawShaderMaterial の利用):

RawShaderMaterialを使用し、three.js のデフォルトシェーダの代わりに自作のシェーダをmeshに適用します。

独自のエフェクトやテクスチャ処理といった、細かな表現のコントロールができるようになります。

html要素とメッシュの同期のイメージ

対象のhtml要素
（div,buttonなど）

同期

対象のmesh

サイズや座標情報は、スクロールやリサイズといった
イベントが起こるたびに変わるので、数値を常に監視し、
meshへ共有する必要があります

- **styleは通常通りcssで調整**
- **width, height, x座標, y座標を取得**

- **geometryのサイズは x = 1,y = 1**
- **サイズはscale(x,y)から設定**
- **座標位置はposition(x,y)から設定**

対象となるhtml要素の情報を取得して、meshのプロパティの値に設定していきます。
meshとhtmlの情報を同一にすることで、html、cssで装飾されたかのように見えるmeshの完成です。
しかし、WebGLにはカメラの概念があるので、次にカメラ位置の設定を調整する必要があります。

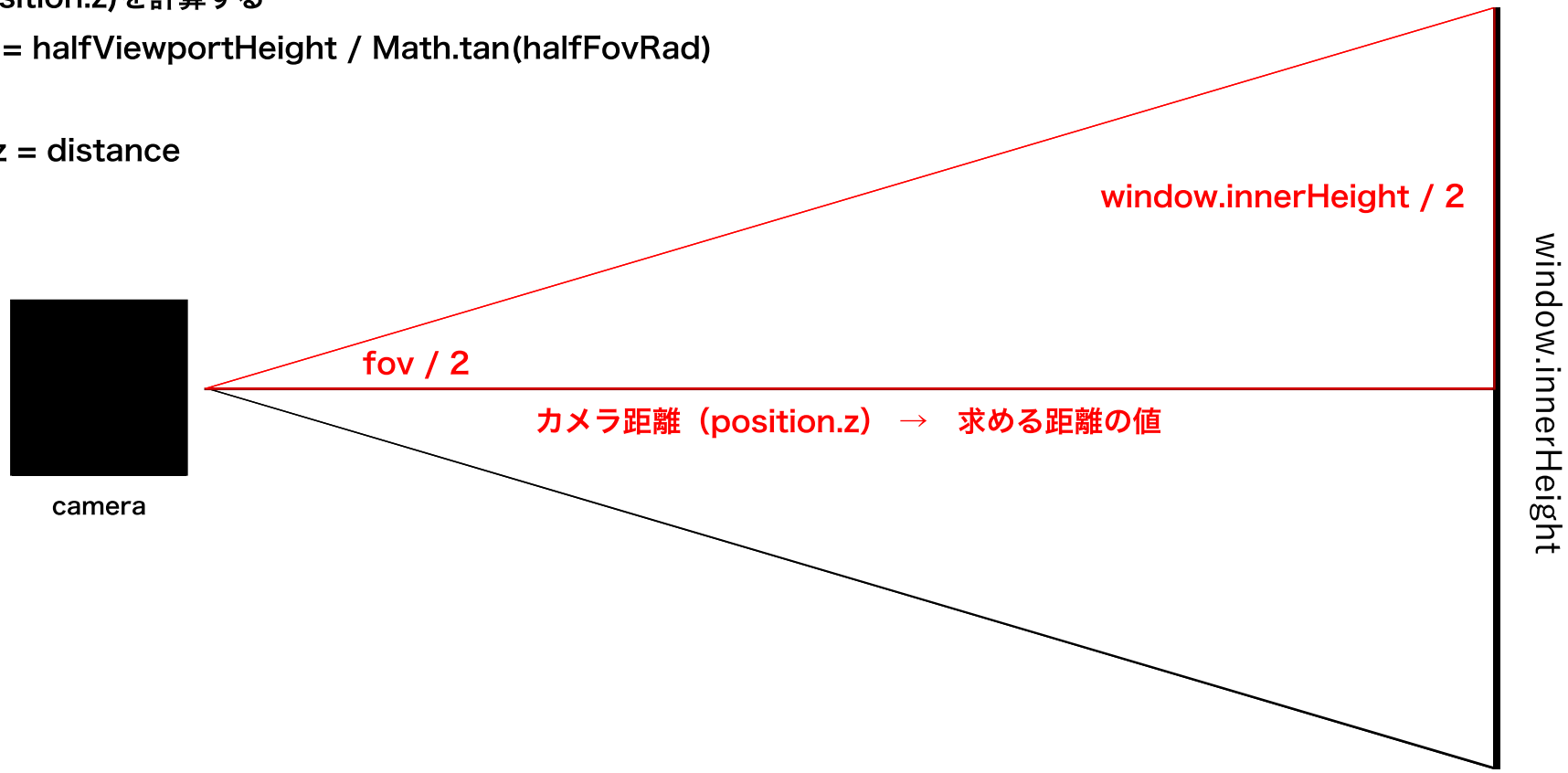
1. fov / 2をラジアンに変換する
   const halfFovRad = (fov / 2) * (PI / 180)

2. viewport(window)の高さの半分を計算する
   const halfViewportHeight = viewportHeight / 2

3. カメラの距離(position.z)を計算する
   const distance = halfViewportHeight / Math.tan(halfFovRad)

camera.position.z = distance

視野角と表示領域は既に設定されているので、
三角関数を用いることで適切なカメラの距離を計算することができます。



RawShaderMaterial

meshの表示サイズや座標をコントロールできるようになったら、次はmeshを利用した表現を追加する準備を進めます。

meshに独自shaderを追加したい場合には**RawShaderMaterial**を利用します。
three.jsでシェーダコーディングをする場合は、必須になるマテリアルです。

▼ three.jsのドキュメント
https://threejs.org/docs/#api/en/materials/RawShaderMaterial
https://threejs.org/docs/#api/en/materials/ShaderMaterial

02 ユーザイベントとインタクションの連携

ベースロジックに続いてもう少しだけ応用を効かせてみます。

今回は「ページ内スクロール」、「meshと同期する要素をマウスホバー・アウトする」などのユーザーイベントがあったタイミングでインタラクションとしてmeshにWebGLらしいエフェクト効果を追加します。

やることの基本は変わらず、必要な情報の数だけuniform変数を増やして、イベントで取得できた値を任意のタイミングで、対象meshのshaderに送るようなイメージです。
スムーススクロール補助や、アニメーションのためのライブラリを使用しますが、重要なのは何かしら変動のある値をshaderに持っていき、その値を元にエフェクトを作成するといった流れの部分になるかと思います。

1. **ユーザーイベントの追加とインタラクション連携:**
   meshと同期するhtml要素にマウスイベントを追加し、ユーザーがマウスを乗せたときや外したときに
   gsapライブラリを使ってmeshの持つuniform変数の値を変更（アニメーション）します。
   shader側ではuniform変数の値の変更と合わせてエフェクトが実行されるような処理を用意しておきます。
   今回はmesh1つに対して2枚のテクスチャを用意し、マウスイベントによって切り替わるような表現を作成します。

2. **スムーススクロールの導入とスクロールを加味した座標値の同期:**
   Lenisライブラリを使ってスムーススクロールを追加し、スクロール時にもhtml要素とmeshの座標を同期させる
   仕組みを実装します。現在のスクロール位置の値を取得し、その値をmeshの座標（position）計算に利用します。

3. **スクロール差分の取得と利用:**
   マイフレーム、現在のスクロール位置（scroll）を取得・保存し、次フレームで前フレームのスクロール位置との
   差分を計算します。その差分の値をシェーダ内で利用することでスクロールの強さに応じたインタラクションを実
   装します。

４. VertexShaderの実装:

スクロール変化の値に基づいて、VertexShader内でmeshの頂点座標にカーブエフェクトを追加します。
頂点のY座標をスクロールの差分量に基づいて動的に変更することで、スクロール時にmeshが曲線的に動く、
より立体的でインタラクティブを感じられる表現を作成してみます。

5. FragmentShaderの実装:

FragmentShader内では、マウスイベントに基づいてテクスチャを動的に変更する演出を実装します。
テクスチャにトランジション効果やスケーリング効果を使用して、ユーザー操作に応じたテクスチャの
切り替えや拡大縮小を行います。
GLSLのmix関数やsmoothstep関数を使って、テクスチャ間のクロスフェードやマスク効果を実現し、
滑らかなグラデーションやエッジのシャープな切り替えを作成してみます。
※ シェーダの説明は、コード内にコメントを追加しています。

まとめ

htmlとmeshの同期、ユーザーイベントの紐付けの基本を理解できたあとは、表現と紐付け方の工夫の数を少しずつ増やしていくと良いと思います。

スクールで学んだ1つ1つの知識やアイデアを要所に組み込んでみるとより面白いものができると思います。
シェーダ表現の引き出しを増やしたいのであれば、杉本さん主催のGLSLスクールへの参加もおすすめしたいです！
（今年も開催されそうであれば生徒で参加したい・・）

最後に、WebGL実装に慣れていない方にもお勧めできる参考サイトを貼っておきます！

https://tympanus.net/codrops/
https://thebookofshaders.com/
https://www.youtube.com/@akella_