

# JavaScript初心者のためのThreeVRM (v1.0) 実装ガイド

コピペで済ませる「土台」と、自分で考える「魂」を書き分ける



The Foundation  
(Copy/Paste)



The Soul  
(Logic/Thinking)

本スライドは、最新のVRM 1.0仕様に対応した  
Three.js実装の「建築図面」です。  
コードのどこに意識を集中すべきかを解説します。

```
import { GLTFLoader } from 'three/examples/jsm/loaders/GLTFLoader.js';
```

```
// VRM v1.0 compliant loading logic here
```

# 学習の戦略：ボイラープレート vs ブレインパワー



## The Foundation (土台)

Three.jsの標準的な設定。  
暗記不要。コピペでOK  
(おまじない)。



## The Soul (魂)

VRM特有の処理。  
ここが学習の核心。  
しっかり理解して実装する。

提供された main.js を2つのカテゴリに分類して解説します。60%のコードは「舞台装置 (Scene/Camera)」であり、毎回同じです。残りの**40%**の「**VRM制御ロジック**」に脳のリソースを集中させましょう。

[COPY / 土台]

# 3D空間の「舞台装置」を整える

```
const scene = new THREE.Scene();
const camera = new
const camera = new THREE.PerspectiveCamera(30, ...);

const renderer = new THREE.WebGLRenderer({
    antialias: true });

// Important!
renderer.outputColorSpace = THREE.SRGBColorSpace;
```



## Scene / Camera / Renderer:

これらは3Dを表示するために  
必須の「おまじない」です。

### ! 注意点 (The Trap)

outputColorSpace の設定を忘れないで  
ください。Three.js r152以降およびVRM  
1.0の推奨設定(リニアワークフロー)です。  
これがないと、モデルが暗くくすんで表示  
されてしまいます。

# 光と視点を配置する



```
const dirLight = new  
THREE.DirectionalLight(0xffffff, 1.0);
```

DirectionalLight: 太陽光です。  
これがないと真っ暗になります。  
(1, 2, 1) の位置から照らします。



```
const controls = new OrbitControls(camera,  
renderer.domElement);
```

OrbitControls: マウスでカメラを  
ぐりぐり動かすための機能です。  
これだけでデバッグが劇的に楽に  
なります。

ここまでが「コピペで済ませる土台」です。次から、いよいよVRMの実装（考えるパート）に入ります。

# 扱うファイルの正体：glTFとVRM

## glTF (3D界のJPEG)

Webで扱うための軽量な3D標準フォーマット。ファイルサイズが小さく、ブラウザですぐ読み込まれるのが特徴です。



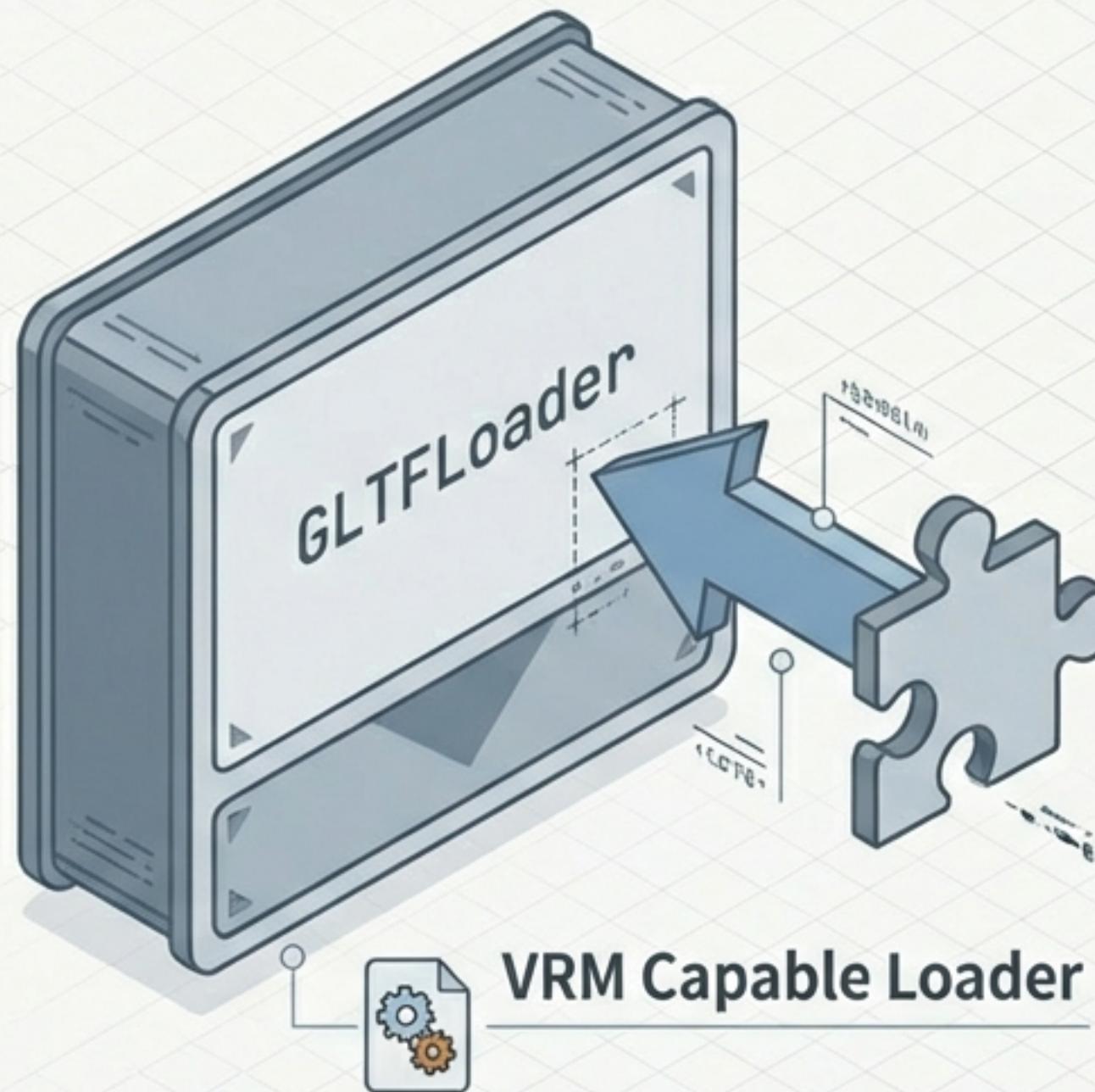
### なぜ重要なか

VRMがglTFベースであるため、Three.jsの標準ローダー(GLTFLoader)をそのまま利用できるのです。

### VRMの正体

実はVRMは、glTFの仕組みを利用して作られています。「人型(Humanoid)」のルールを追加したglTFファイル、それがVRMです。

# VRM 1.0 の流儀：プラグインシステム



```
const loader = new GLTFLoader();
// Plug in the VRM logic
loader.register((parser) => new VRMLoaderPlugin(parser));
```

ここが v1.0 の最重要変更点です。

- 以前は VRM.from(...) という独自の読み込み方をしていました。
- 現在は、Three.js標準の GLTFLoader に、VRMLoaderPlugin を「拡張機能」として合体 (register) させて読み込みます。

「VRMも結局はglTFの一種である」という設計思想が、このコードに表れています。

# 非同期処理 (Async/Await) とラーメンの法則

同期処理  
(画面が固まる)



Time

巨大な3Dファイルを読み込む際、この処理がないとブラウザがフリーズしてしまいます。

非同期処理  
(効率的)

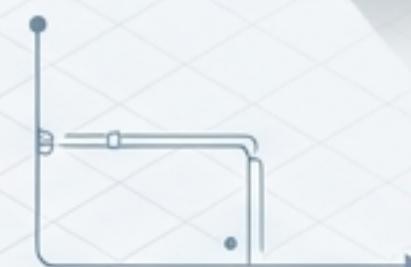


Time

loader.loadAsync

Other Tasks

await



```
await loader.loadAsync(url)
```

「読み込み開始」 → 「待っている間に他の準備」 →  
「完了したら次へ (await) 」という流れを作ります。

# モデルの取り出しと最適化

```
const vrm = gltf.userData.vrm;  
  
// Optimization Step  
VRMUtils.removeUnnecessaryJoints(gltf.scene);  
scene.add(vrm.scene);
```

## 1. 格納場所

VRMデータは gltf そのものではなく、gltf.userData.vrm という「引き出し」の中に入っています。

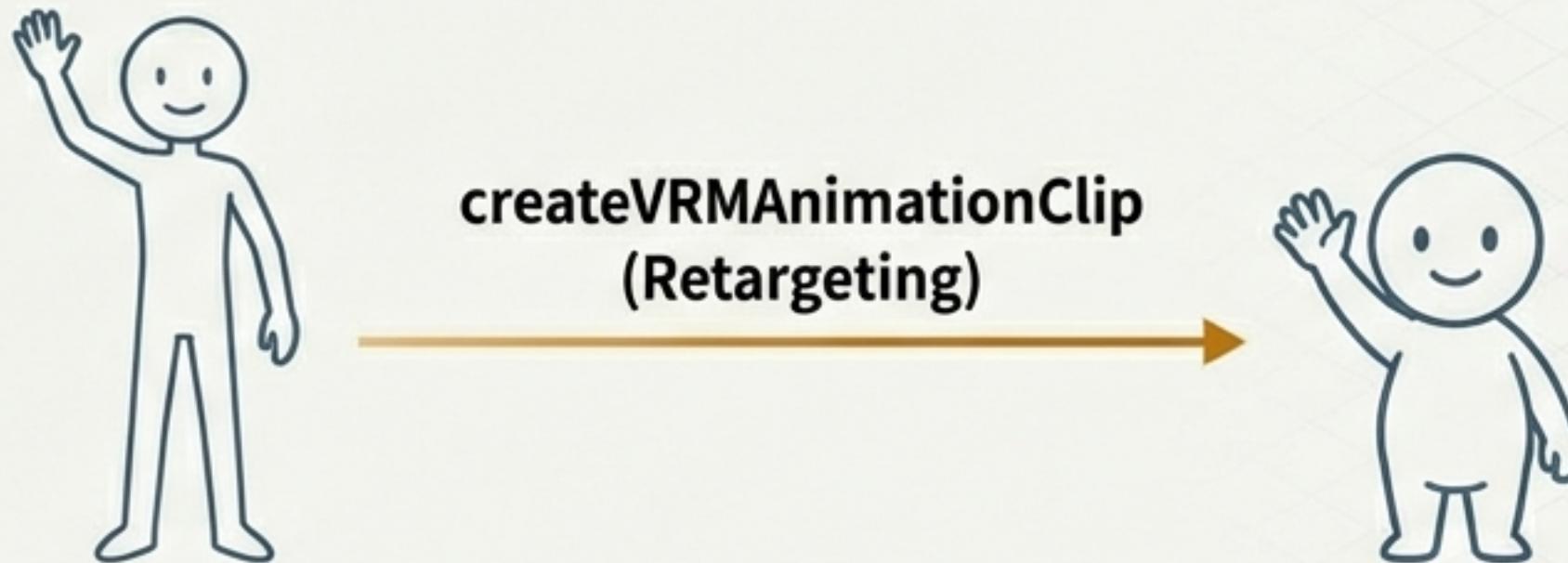
## 2. VRMUtils.removeUnnecessaryJoints ✕

これが重要です。モデリングソフトから書き出されたデータには「見た目に影響しない無駄な骨」が含まれることがあります。

これらを削除して、描画負荷を減らす（最適化する）プロのテクニックです。



# アニメーションの魔法：リターゲティング



```
const clip = createVRMAnimantionClip(vrmAnimations[0], vrm);
```

3Dモデルは、**身長や腕の長さ**がそれぞれ違います。データをそのまま流し込むと、腕が体にめり込んだり、足が浮いたりします。

**createVRMAnimatlionClip** は、「このモデル（vrm）の**体格**に合わせて、動きを自動調整してね」という指示です。

これにより、どんな体格のキャラクターでも自然に動かすことができます。

[THINK / 安全装置]

# エラーでアプリを落とさない (Try...Catch)



```
try {  
  const gltf = await loader.loadAsync(url);  
  // ...Success...  
} catch (error) {  
  console.error('読み込みエラー:', error);  
  // ...Safe handling...  
}
```

ファイルが見つからない、ネットが切れた等の予期せぬトラブルは必ず起きます。

Try (やってみて) ... Catch (だめだったら) の構文で囲むことで、エラーが起きても画面が真っ白にならず、原因をログに表示してプログラムを守ることができます。

[THINK / 鼓動]

# 命を吹き込むループ (Update Loop)



```
function animate() {  
    const deltaTime = clock.getDelta();  
    if (currentVrm) currentVrm.update(deltaTime);  
    if (currentMixer) currentMixer.update(deltaTime);  
    renderer.render(scene, camera);  
}
```

- 3Dはパラパラ漫画です。1秒間に60回、絵を描画し続けます。
- **DeltaTime**: 前のフレームからの経過時間。これを渡すことで、PCの性能に関わらず一定の速度で髪が揺れ、アニメーションが進みます。
- **Double Update**: vrm.update (髪の揺れ等) と mixer.update (ポーズの変更) の両方を呼び出すのを忘れないでください。

# 動かない？ デベロッパーツール（F12）をチェック



## 404 Not Found

ファイルパスの間違いです。`./vrma/motion.vrma`などの場所を確認してください。



## Cross-Origin Request Blocked

ローカルファイルを直接ブラウザで開いていませんか？  
VS Codeの「Live Server」などを使用してください。

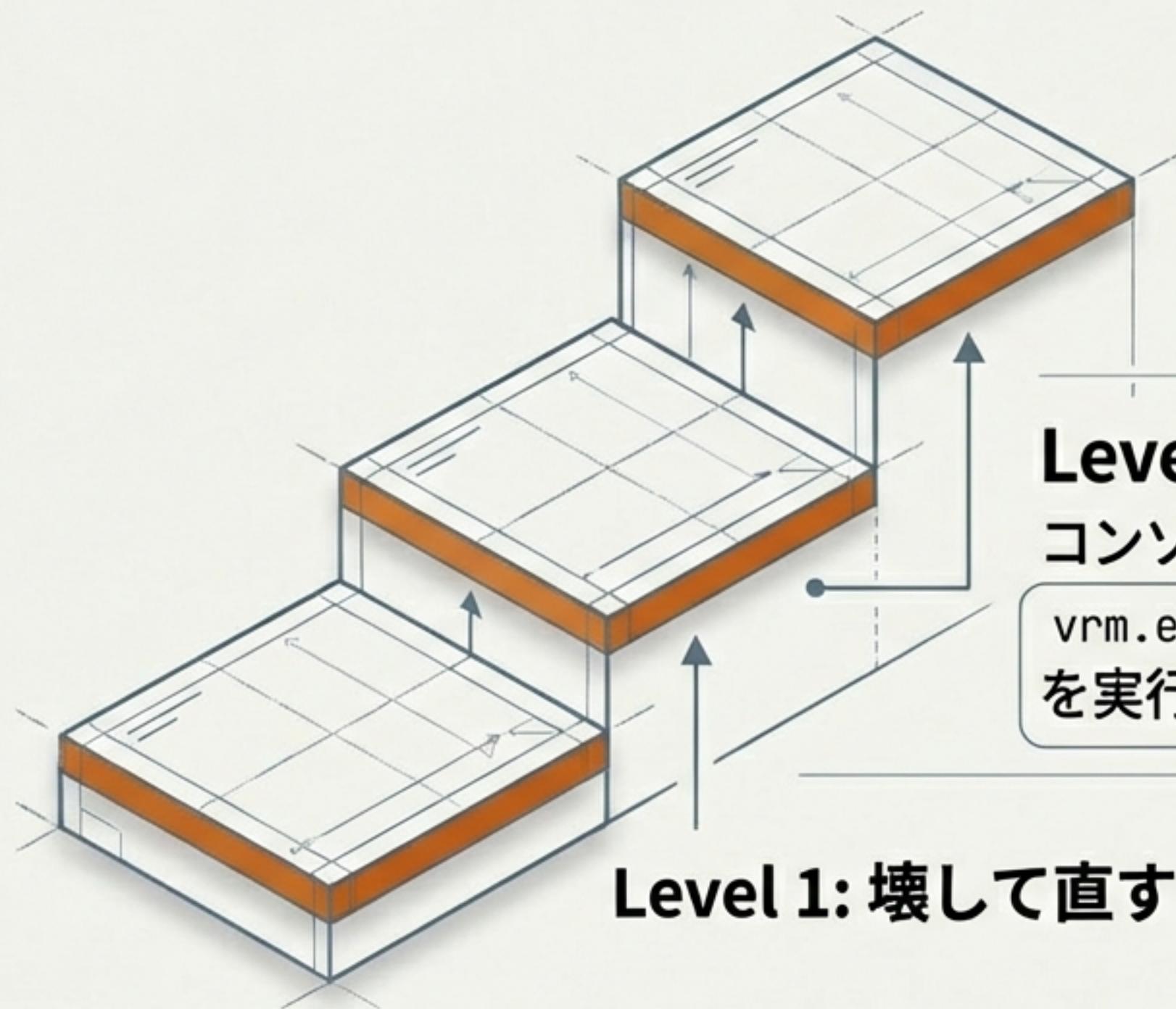


## 画面が暗い

`outputColorSpace`の設定を再確認してください（Slide 3参照）。

[ACTION / 次のステップ]

# 学習ロードマップ：ここからどう遊ぶ？



## Level 1: 壊して直す

モデルを自分の好きな .vrm に変える。  
ライトの色を変えてみる。



## Level 2: 表情 (Expression)

コンソールで

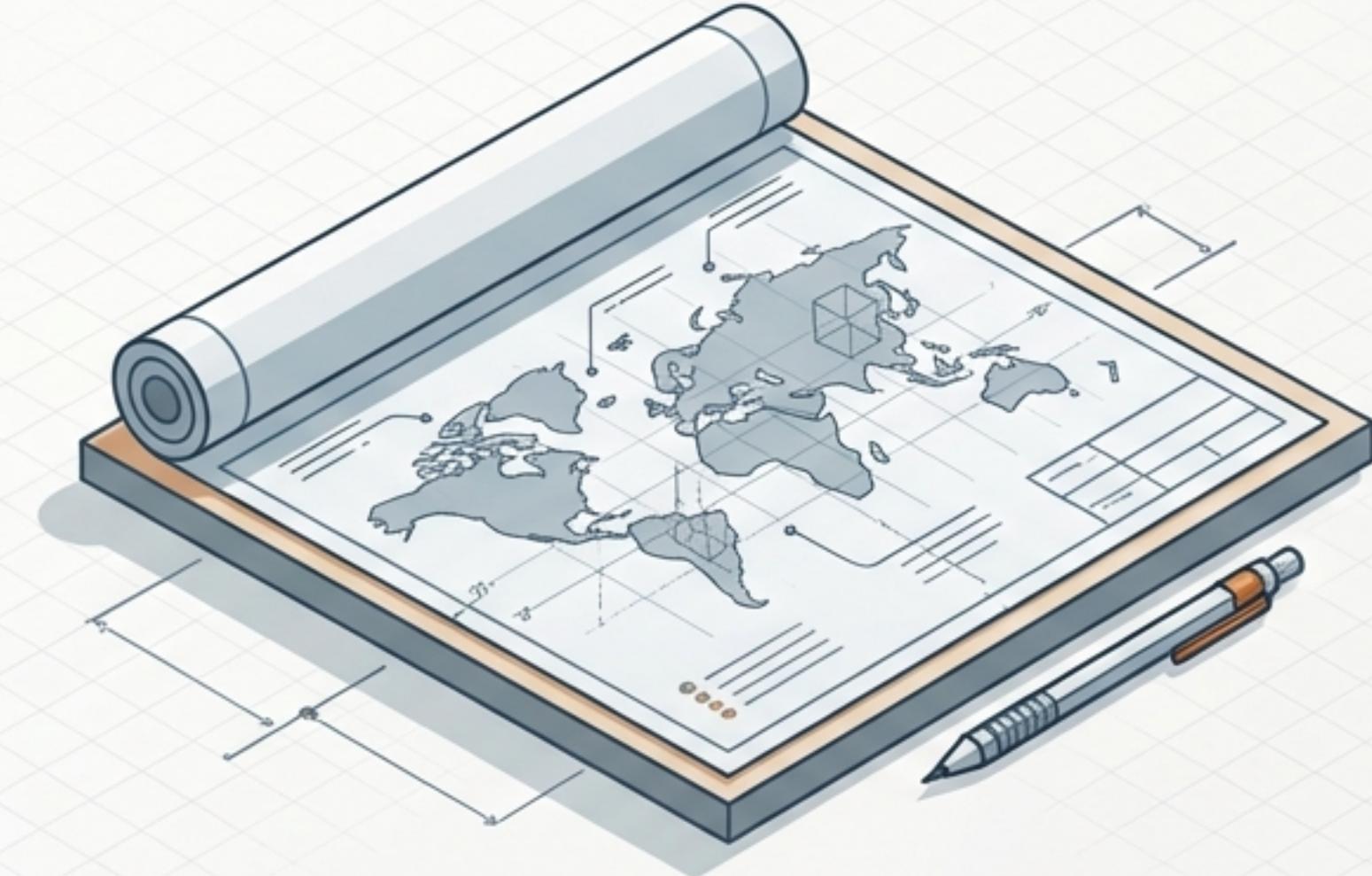
```
vrm.expressionManager.setValue('happy', 1.0)  
を実行し、表情を変えてみる。
```



## Level 3: インタラクション

クリックしたら表情が変わる、キーボード  
でモーションが変わる機能を追加する。





## あなたは「地図」を手に入れました

今回解説したコードは、VRMを表示しアニメーションさせるための「黄金パターン」です。まずはこの「土台」を信頼してコピペし、その上の「魂（ロジック）」の部分を書き換えて、あなただけの3Dワールドを構築してください。

さあ、エディタを開いて main.js を書き始めましょう。