**Lo del proyecto ARPA con WordPress + vuestro visor en Three.js:**  
Es totalmente viable integrarlo. Lo más típico sería:

* **Incrustar un <iframe>** en la página WordPress que cargue vuestro visor (publicado en un subdominio o servidor externo).
* O montar el visor como **una PWA independiente** enlazada desde WordPress.

Ahora sí, vamos con la lista:

**VIABLES DIRECTOS (fáciles de implementar)**

**1. Cambiar el fondo a blanco o negro (o color plano)**

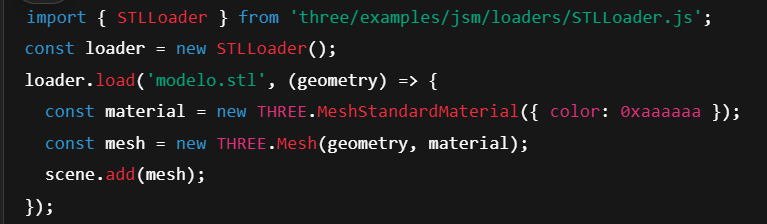
* Ya tenéis quitarHDRI() en scene.js.
* Bastaría con:
  + Añadir un input tipo color para el fondo (como el del modelo).
  + Aplicarlo a scene.background = new THREE.Color(...).

**Dificultad**: mínima.  
**Dónde ponerlo**: debajo de “Mostrar HR” o como una alternativa visual a HDRI si está desactivado.

**REQUIERE AJUSTES (pero con Three.js es viable)**

**2. Soporte para formato STL**

* STL es un formato muy común en impresión 3D.
* Con Three.js es **totalmente soportado**:

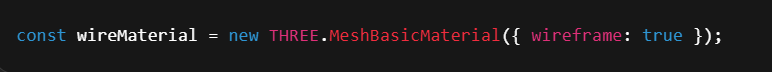


Requiere adaptar loadModel(file) para que detecte extensión y use GLTFLoader o STLLoader.

**Dificultad**: baja-media.  
**Desafío**: STL no guarda materiales ni texturas → tendrás que aplicar un material por defecto.

**3. Visualizar la malla del modelo**

* También fácil con Three:



* Podemos usar un checkbox tipo “Ver malla” y aplicar ese material temporalmente.

**Dificultad**: baja.  
**Desafío**: mantener también la opción de restaurar materiales originales.

**HASTA AQUÍ LO MÁS FÁCIL O VIABLE DE UNA.**

**MÁS COMPLEJO (pero se puede estudiar)**

**4. Comparar dos modelos / superponer mallas**

Esto sí es más reto. En resumen:

* Three.js puede **cargar y mostrar varios modelos** a la vez.
* La superposición es posible ajustando **posiciones y escalas** para que coincidan.
* Para comparar diferencias:
  + Hay que implementar **detección de diferencias geométricas**.
  + O usar librerías externas (por ejemplo, comparadores de mallas tipo [MeshLab](https://www.meshlab.net/), que no es JS).

Se podría simular una comparación activando y desactivando visibilidad de uno u otro con un botón, o alternando transparencias para notar diferencias visuales.

**Dificultad**: media-alta.  
**Requiere**:

* Soporte para cargar dos archivos.
* Crear una lógica para posicionarlos iguales.
* Ideal: herramientas visuales para comparar (transparencias, sliders, etc).

**Recomendaciones**

1. **Haz una lista de fases**:
   * ✅ Lo que ya tenéis.
   * 🟡 Lo que podéis añadir fácil.
   * 🔴 Lo que hay que estudiar con calma.
2. **Para lo de ARPA**: montad el visor en GitHub Pages o Netlify, y ya pueden integrarlo por iframe.
3. **Preparad una demo con dos modelos superpuestos (aunque solo visual)** si queréis impresionar en una futura reunión.

**¿Qué está pidiendo realmente?**

Él habla de **comparar mallas** con:

1. **Puntos de referencia comunes** entre dos modelos.
2. Poder **marcar** en uno y reflejar en el otro.
3. Incluso **alinear manualmente** los modelos si no están bien colocados.
4. Detectar **pequeñas diferencias geométricas** (para ver desgaste o errores).

**¿Es viable con Three.js?**

*En parte, sí.* Pero no es fácil ni inmediato con Three “pelado”. Hay que trabajar bastante la interacción y el cálculo. Aquí tienes un resumen técnico realista:

**Mostrar dos modelos superpuestos**

**Sí, sin problema**. Puedes cargar ambos, posicionarlos, hacerlos semitransparentes, alternar visibilidad, etc. **Opciones útiles:**

* model1.material.opacity = 0.5; material.transparent = true;
* o model2.visible = false con botones.

**Marcar puntos de referencia en ambos modelos**

Esto implica:

* **Capturar clics** en el modelo (ya sabéis cómo con raycaster).
* Mostrar un SphereGeometry pequeña en ese punto.
* Guardar las coordenadas del punto para el modelo A y luego permitir al usuario colocar el correspondiente en el modelo B.

Lo difícil aquí es: ¿cómo sabes si realmente están en el mismo “punto” en mallas distintas? Solo si son idénticas o alineadas previamente.

**Sincronizar puntos entre dos mallas**

Para eso habría que:

* Tener **mallas casi idénticas** o con misma topología.
* Si se marca el vértice 1342 en una, encontrar el vértice 1342 en la otra.
* **Esto solo es viable si se cargan con exactamente la misma geometría base**.

Opción más realista: permitir **colocar manualmente el punto** también en el segundo modelo, y luego comparar visualmente.

**Arrastrar, ajustar, alinear uno sobre otro**

Sí, pero requiere:

* Implementar **controles de transformación** (tipo TransformControls de Three.js).
* Que el usuario pueda mover, escalar o rotar un modelo mientras el otro está fijo.
* Ideal para “hacer coincidir manualmente”.

*Esto se puede hacer. Requiere algo más de código y testeo, pero no es ciencia ficción.*

**DETECCIÓN AUTOMÁTICA DE DIFERENCIAS**

Ahí ya **salimos del mundo de Three.js** y entramos en el de:

* **Procesamiento de mallas** (dif comparación vértice a vértice).
* Comparaciones geométricas, cálculos de distancia, tolerancia, etc.
* Librerías especializadas (por ejemplo, no JS → Blender, MeshLab, CloudCompare).

Si quiere esto, sería mejor exportar las mallas y compararlas con herramientas externas. Desde web… muy limitado. \* OPCIONES REALISTAS EN LA SIGUIENTE PÁGINA

**Recomendación práctica**

Si queréis **simular esta comparativa en vuestro visor**, lo ideal sería:

1. Crear **una “escena alternativa”** que se active con un botón “Modo Comparación”.
2. Ahí se cargan dos modelos.
3. Dar herramientas para:
   * Mostrar/ocultar cada uno.
   * Cambiar color/transparencia.
   * Añadir puntos (clic y esfera).
   * Mover manualmente uno (con controles o sliders).
4. Mostrar en pantalla las coordenadas, nombres de los archivos, y si hay alguna distancia entre puntos marcados.

\* OPCIONES REALISTAS

**✅ OPCIÓN 1 — Comparación visual pura**

Ya lo comentamos: cargar 2 modelos, permitir mover uno, cambiarle el color, semitransparente, girarlo, ocultarlo, etc.

🔹 **Viable 100%**, solo código. Sirve para comparar “a ojo”.

**🟡 OPCIÓN 2 — Comparación de puntos marcados manualmente**

El usuario puede:

* Pulsar para colocar puntos (esferas) en ambas mallas.
* El visor muestra **distancia entre esos puntos**.
* Así se puede decir “este punto está 2.1mm desplazado”.

🔹 Requiere raycasting, guardar posiciones y usar Vector3.distanceTo().

🔸 No es automática, pero útil para inspecciones semimanuales.

**🔴 OPCIÓN 3 — Comparación *automática* de mallas (más avanzada)**

Esto sería detectar diferencias entre geometrías enteras:

* Comparar vértices uno a uno.
* Ver dónde hay más desviación.
* Generar un informe o mapa de calor.

🔹 **NO es viable directamente en Three.js.**  
🔸 Necesitarías herramientas como:

* **MeshLab** (open source): para calcular diferencias y exportar.
* **CloudCompare** (gratuito): para analizar escaneos y mostrar diferencias.

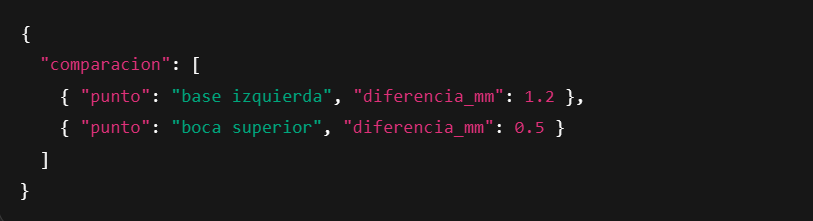
🔁 Pero podrías usarlo *fuera de la web* y luego subir un informe a vuestra plataforma web (tipo PDF o imágenes generadas).

**💡 Idea híbrida (si os motiva)**

Una solución intermedia sería:

Hacer un pequeño script (fuera de la web, en Node o Python) que compara dos .glb o .stl, calcula diferencias y genera un JSON con resultados que **sí podríais mostrar en la web**.

Por ejemplo:



Y entonces el visor mostraría eso como resultado. Así haríamos un puente entre análisis técnico y presentación visual.

