ESCUELA DE INGENIERÍA DE FUENLABRADA

GRADO EN INGENIERÍA EN SISTEMAS AUDIOVISUALES Y MULTIMEDIA

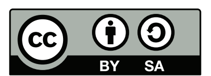
TRABAJO FIN DE GRADO

INTERACCIÓN DE MANOS CON OBJETOS EN REALIDAD VIRTUAL

Autor : Raúl Jiménez Luis

Tutor : Dr. Jesús María González Barahona

Curso académico 2025/2026



©2026 Raúl Jiménez Luis

Algunos derechos reservados

Este documento se distribuye bajo la licencia

“Atribución-CompartirIgual 4.0 Internacional” de Creative Commons,

disponible en

https://creativecommons.org/licenses/by-sa/4.0/deed.es

Dedicado a

mi familia / mi abuelo / mi abuela

Agradecimientos

Agradecer a mi madre y a mi padre, que sin su sacrificio nunca hubiera sido posible que hubiese tenido este desarrollo académico. Son las dos personas que más quiero en mi vida. Siempre están para apoyarme en mis malos momentos, y para darme lecciones que han sido y serán super importantes en mi vida. Sin ellos no sería la persona que soy hoy.

A mis amigos, unos que me han acompañado durante toda mi vida y otros que aparecieron cuando menos lo esperaba para quedarse. Son como mi segunda familia y con ellos tengo anécdotas que nunca olvidare, cada uno con sus diferentes virtudes saca mi mejor versión. Ojalá vivamos muchos más años juntos.

A mi familia, por mantenernos unidos y demostrar amor en los peores momentos, cuando en este maravilloso y a la vez duro camino que es la vida nos dejaron de acompañar. Porque si, este agradecimiento también va a ellos, los que ya no están con nosotros, pero que siempre estarán en nuestra memoria, mientras que ellos nos dan sus bendiciones desde arriba, fruto del amor. Siempre estaremos unidos.

Resumen

La Realidad Virtual o Extendida (XR) busca que el usuario se sumerja en entornos interactivos. En los últimos años, los visores modernos han integrado el seguimiento óptico de manos libres (*hand-tracking*). Esta tecnología permite interactuar directamente con el entorno virtual sin necesidad de usar controladores físicos. A su vez, el estándar WebXR ha llevado estas experiencias directamente al navegador web, facilitando enormemente su acceso. Sin embargo, programar estas interacciones manuales desde cero requiere conocimientos matemáticos complejos. Actualmente no existen herramientas en el desarrollo web que tengan soporte de gestos espaciales, ya que la mayoria fueron para su uso con controladores.

Para resolver esta limitación, este Trabajo de Fin de Grado propone el desarrollo de una biblioteca de software de alto nivel. Se ha creado una caja de herramientas (*toolkit*) modular para el marco de trabajo de A-Frame. Este se ha basado en la arquitectura de una de las bibliotecas consolidadas en el uso de mandos, como es *Super-Hands*, a un entorno controlado exclusivamente por las manos del usuario. La estructura se fundamenta en el análisis anatómico de las articulaciones y en el cálculo de colisiones geométricas. De este modo, permite a los programadores dar interactividad a los objetos tridimensionales utilizando simples etiquetas declarativas HTML.

El sistema resultante es un paquete unificado escrito en JavaScript, apoyado en las tecnologías Three.js y WebXR. Se han desarrollado detectores de gestos específicos (como pellizcar y apuntar) y componentes interactuables (como agarrar, estirar, arrastrar y soltar). La solución se evaluó mediante la adaptación de escenas clásicas y la ejecución de pruebas con usuarios reales en un entorno orientado a tareas. Los resultados confirmaron la usabilidad de las mecánicas implementadas. El proyecto reduce de forma drástica los tiempos de programación, facilita la creación de interfaces y mantiene un rendimiento óptimo dentro del navegador del dispositivo.

Summary

Virtual or Extended Reality (XR) seeks to immerse the user in interactive environments. In recent years, modern headsets have integrated optical free-hand tracking (*hand-tracking*). This technology allows direct interaction with the virtual environment without the need to use physical controllers. In turn, the WebXR standard has brought these experiences directly to the web browser, greatly facilitating their access. However, programming these manual interactions from scratch requires complex mathematical knowledge. Currently, there are no web development tools that natively support spatial gestures, as most were designed for use with controllers.

To solve this limitation, this Bachelor's Thesis proposes the development of a high-level software library. A modular *toolkit* has been created for the A-Frame framework. This system is based on the architecture of established controller-based libraries, such as *Super-Hands*, adapting it to an environment controlled exclusively by the user's hands. The structure relies on the anatomical analysis of joints and the calculation of geometric collisions. In this way, it allows programmers to add interactivity to three-dimensional objects using simple declarative HTML tags.

The resulting system is a unified package written in JavaScript, supported by Three.js and WebXR technologies. Specific gesture detectors (such as pinch and point) and interactable components (such as grab, stretch, drag, and drop) have been developed. The solution was evaluated by adapting classic scenes and conducting real user tests in a task-oriented environment. The results confirmed the usability of the implemented mechanics. The project drastically reduces programming times, facilitates interface creation, and maintains optimal performance within the device's browser.

Índice general

1. Introducción **1**

1.1. Contexto y motivación 1

1.2. Objetivos 2

1.2.1. Objetivo principal 2

1.2.2. Subobjetivos instrumentales 3

1.2.3. Objetivos relacionados 3

1.3. Contribuciones 4

1.4. Contribuciones 4

1.5. Estructura de la memoria 5

2. Tecnologías utilizadas y trabajos relacionados **7**

2.1. Tecnologías base para el entorno inmersivo web 7

2.1.1. A-Frame y el patrón arquitectónico ECS 8

2.1.2. WebXR Device API 10

2.1.3. Three.js y la representación gráfica 10

2.1.4. HTML5 12

2.1.5. JavaScript y la arquitectura de eventos 12

2.1.6. WebGL 13

2.2. Tecnologías y algoritmos para la detección de colisiones 14

2.2.1. Volúmenes delimitadores: AABB frente a OBB 14

2.2.2. El Teorema SAT y su integración en el DOM 15

2.2.3. Lanzamiento de rayos (Raycasting) 16

2.3. Trabajos relacionados y motores de la industria 17

2.3.1. A-Frame Super Hands: La influencia arquitectónica 17

2.3.2. SDKs nativos comerciales: Meta Interaction SDK 18
2.3.3. Unity 19
2.3.4. Godot Engine 19
2.4. Herramientas para el ciclo de vida del software 20

3. Resultados obtenidos 21
3.1. Descripción funcional y de interfaz 21
3.1.1. Componentes de colisión 22
3.1.2. Componentes gestuales 23
3.1.3. Componentes interactuables 27
3.2. Construcción de escenas y ejemplos de uso 33
3.2.1. Llamada a la API de seguimiento de manos 34
3.2.2. Importación unificada y prevención de conflictos 34
3.2.3. Inicialización de mallas y gestos 34
3.2.4. Composición de entidades interactuables 35
3.3. Descripción de implementación 35
3.3.1. Interfaces transversales y gestión de colisiones 36
3.3.2. Lógica de seguimiento y detección gestual 36
3.3.3. Implementación de componentes interactuables 37
3.4. Comparación con super-hands 40
3.4.1. Análisis de la arquitectura original basada en controladores 40
3.4.2. Adaptación mediante la arquitectura de gestos integrados 41
3.4.3. Validación de la interfaz de componentes interactuables 42
3.5. Ejemplos de demostraciones 42
3.5.1. Escenas de prueba individuales 43
3.5.2. Demostración integradora: Adaptación de Super-Hands 43
3.5.3. Demostración en Realidad Aumentada (AR) 45
3.5.4. Demostración final orientada a tareas: Taller de Ensamblaje 45

4. Desarrollo del proyecto 49
4.1. Ejecución de los Sprints 50
4.2. Sprint 0: Fundamentos del entorno y reducción de riesgos 50

4.3. Sprint 1: Núcleo de seguimiento y colisionadores base 52
4.4. Sprint 2: Gestos principales y componente de agarre (*grab*) 55
4.5. Sprint 3: Dualidad de colisiones y arquitectura unificada 57
4.6. Sprint 4: Escalado bimanual (*stretch*) y robustez del sistema 59
4.7. Sprint 5: Interacciones de proximidad, arrastre y soltar 61
4.8. Sprint 6: Interacción directa (*click*) mediante gesto apuntar 63
4.9. Sprint 7: Integración y pulido XR 65

5. Pruebas y experimentos realizados 69
5.1. Descripción detallada del entorno experimental 69
5.2. Pruebas con usuarios sin experiencia técnica 70
5.2.1. Protocolo del experimento y procedimiento 70
5.2.2. Definición de tareas y sistemas de medición 71
5.2.3. Resultados del experimento y análisis cuantitativo 72
5.2.4. Análisis cualitativo y encuesta post-experimento 73

6. Conclusiones 75
6.1. Consecución de objetivos y balance del proyecto 75
6.2. Esfuerzo y recursos dedicados 76
6.3. Lecciones aprendidas y problemas resueltos 76
6.4. Impacto de las asignaturas de la titulación 78
6.5. Trabajos futuros 78

Bibliografía 81

Índice de figuras

2.1. Resultado visual de la escena en A-Frame 9
2.2. Esqueleto digital de WebXR 11

3.1. Gesto de pellizco (*pinch*) 24
3.2. Gesto de apuntar (*point*) 26
3.3. Resultado gráfico de la escena básica 35
3.4. Escena de demostración individual 44
3.5. Adaptación de la escena de Super-Hands 45
3.6. Demostración Taller de Ensamblaje 46

4.1. Visualización de articulaciones mediante esferas 54
4.2. Prototipo de interacción directa 64

Capítulo 1

Introducción

La Realidad Virtual (VR) y Extendida (XR) está conformada por un conjunto de tecnologías diseñadas para sumergir al usuario en entornos generados por ordenador. Desde sus inicios, uno de los mayores retos de este campo ha sido la construcción de interfaces de usuario que resulten creíbles y fáciles de utilizar. Tradicionalmente, la interacción dentro de los entornos virtuales se ha realizado mediante el uso de controladores físicos. Estos dispositivos de hardware actúan como elementos de conexión directa para que el usuario, mediante la pulsación de botones analógicos, pueda hacer acciones básicas como pellizcar un objeto o pulsar un panel.

Los controladores son eficaces en su funcionalidad, pero limitan la sensación de inmersión. En los últimos años, ha habido una gran evolución de los algoritmos de visión artificial, que han permitido la integración de sistemas de seguimiento óptico de manos libres (*hand-tracking*). Esta tecnología permite al usuario prescindir de los controladores e interactuar con la escena utilizando su propia anatomía de forma natural. El presente Trabajo de Fin de Grado aborda el diseño y la implementación de una arquitectura de software que facilita la integración de estas mecánicas manuales en el ecosistema inmersivo web.

1.1. Contexto y motivación

El desarrollo de aplicaciones inmersivas ha experimentado facilidades gracias al estándar WebXR. Esta interfaz de programación permite ejecutar experiencias de Realidad Virtual (VR) y Realidad Aumentada (AR) directamente desde un navegador de internet, eliminando la necesidad de instalar aplicaciones nativas pesadas. Dentro de este ecosistema web, el marco de

trabajo A-Frame se ha consolidado como la herramienta de referencia para construir escenas tridimensionales mediante el uso de etiquetas declarativas similares al lenguaje HTML.

Para dotar de interactividad a los objetos estáticos de A-Frame, la comunidad de desarrollo ha dependido históricamente de bibliotecas de código abierto. Entre ellas, una popular y utilizada es *Super-Hands*. Esta herramienta estandarizó la manipulación espacial al ofrecer componentes modulares para agarrar, estirar o lanzar objetos. Su diseño arquitectónico demostró ser altamente eficiente. Sin embargo, *Super-Hands* fue conceptualizada en una etapa tecnológica anterior, en la que el código fuente y los algoritmos internos están adaptados a la lectura de eventos de hardware tradicionales, como el movimiento de un gatillo mecánico.

Con la estandarización del seguimiento óptico de manos en los visores modernos, hubo un cambio en el paradigma. Ahora, ya se tenía acceso a las posiciones de las manos, lo que generaba una nueva posibilidad. Se crearon nuevos componentes básicos nativos para interactuar con las manos en la web, pero su alcance funcional es limitado. Tras realizar un estudio amplio de las interfaces de programación (API) de seguimiento de manos disponibles en el estándar WebXR, se identificó que la implementación directa de estas tecnologías a bajo nivel presenta una elevada complejidad matemática y algorítmica para el programador. Por tanto, la motivación principal de este proyecto nace de la necesidad de diseñar una biblioteca de alto nivel que mitigue esta barrera y facilite la creación de entornos manuales interactivos. Para ello, se adoptó la decisión estratégica de adaptar la excelente estructura semántica de la biblioteca original *Super-Hands*, trasladando sus principios de compatibilidad hacia un sistema formado exclusivamente por gestos naturales y análisis anatómico, prescindiendo por completo de los mandos físicos.

1.2. Objetivos

Para guiar la ejecución técnica y asegurar la resolución del problema planteado, el proyecto se estructura en torno a una serie de metas específicas.

1.2.1. Objetivo principal

El objetivo prioritario de este Trabajo de Fin de Grado es diseñar y desarrollar una biblioteca de componentes modulares de alto nivel para el marco de trabajo A-Frame que simplifique la

implementación del código necesario para el manejo de manos en el entorno Web. El propósito de este sistema es proporcionar una caja de herramientas (*toolkit*) declarativa para la manipulación de entidades tridimensionales mediante el seguimiento espacial de manos libres, adaptando la estructura y compatibilidad de la biblioteca clásica *Super-Hands* al análisis gestual.

1.2.2. Subobjetivos instrumentales

Para alcanzar el objetivo principal, se definieron las siguientes tareas de desarrollo funcional:

- **Extracción y renderizado de datos anatómicos:** Implementar un sistema capaz de leer la matriz de coordenadas proporcionada por la API de WebXR de las posiciones de las manos, para renderizar una representación visual de las mismas.
- **Detección y clasificación de gestos:** Desarrollar algoritmos matemáticos que evalúen de forma continua las distancias entre las falanges para registrar e identificar gestos, priorizando el de pellizco (*pinch*) y el de apuntado (*point*).
- **Desarrollo de un motor de colisiones espacial:** Programar e integrar sistemas geométricos de cálculo de intersecciones (OBB y SAT), capaces de evaluar el contacto físico entre la anatomía del usuario y la topología de los objetos sin recurrir a simulaciones pesadas de cuerpos rígidos.
- **Construcción de componentes interactuables:** Programar la lógica de transformación espacial para dotar a los objetos de comportamientos interactuables, incluyendo mecánicas de proximidad, agarre, escalado bimanual y arrastre.

1.2.3. Objetivos relacionados

De forma paralela a la implementación técnica, se establecieron requerimientos adicionales para asegurar la robustez del sistema:

- **Desacoplamiento y unificación del código:** Configurar el empaquetado del software para que el desarrollador final pueda importar toda la infraestructura de la biblioteca mediante un único archivo en la cabecera de su documento HTML.

- **Compatibilidad agnóstica XR:** Asegurar que las mecánicas programadas operen con igual precisión tanto en escenarios cerrados de realidad virtual como en entornos de realidad mixta superpuestos sobre el espacio físico del usuario (*passthrough*).

1.3. Contribuciones

1.4. Contribuciones

El resultado del trabajo ha generado aportaciones directas y funcionales al ecosistema de desarrollo inmersivo web. La contribución principal es la propia biblioteca de código fuente, empaquetada y lista para su distribución. Todo el código fuente del proyecto, junto con la documentación técnica y las instrucciones de instalación, se encuentra publicado en un repositorio de GitHub abierto[1], el cual cuenta también con una página web dedicada[2] donde se centralizan los accesos a los interactivos, los vídeos y la presente memoria.

1. Una infraestructura de detección (`hands-spheres`) que simplifica la construcción de un renderizado visual de las manos y la inyección automatizada de los detectores de pellizco y apuntado.

2. Un paquete de componentes interactuables independientes (`hoverable`, `grabbable`, `stretchable`, `draggable`, `droppable` y `clickable`) diseñados para actuar como reemplazos exactos (*drop-in replacements*) de sus homólogos basados en controladores físicos.

Además del código fuente, se aporta una colección de escenas interactivas. Esta suite de pruebas documenta la usabilidad del sistema de forma empírica. Entre ellas destaca la reconstrucción funcional de la demostración original del ecosistema *Super-Hands* y la formulación de una prueba secuencial orientada a tareas (Taller de Ensamblaje).

[1] Repositorio del código fuente en GitHub
[2] Sitio web del proyecto y demostraciones

1.5. Estructura de la memoria

El presente documento detalla las bases teóricas y el proceso de ingeniería seguido para la consecución del proyecto. La información se organiza de la siguiente manera:

- En el Capítulo 2 se detallan las tecnologías utilizadas. Se analizan los lenguajes fundamentales del desarrollo web tridimensional, se exponen los algoritmos matemáticos para la detección de colisiones y se examinan los motores y bibliotecas que conforman el estado del arte de la industria.
- El Capítulo 3 recoge la descripción funcional y técnica del producto final. En él se analizan los componentes desarrollados, su configuración, la metodología para la construcción de escenas y la validación de los objetivos frente a la arquitectura original de *Super-Hands*.
- El Capítulo 4 documenta el proceso cronológico de desarrollo de la biblioteca. Se expone la ejecución del proyecto estructurada en ciclos iterativos (Sprints), detallando los problemas arquitectónicos encontrados y las decisiones de diseño adoptadas en cada fase.
- Finalmente, en el Capítulo 6 se reflexiona sobre los resultados obtenidos, evaluando el grado de cumplimiento de los objetivos planteados y se sugieren posibles líneas de trabajo futuro para expandir el alcance de la herramienta.

Capítulo 2

Tecnologías utilizadas y trabajos relacionados

En este capítulo se presenta el conjunto de herramientas de software, lenguajes de programación y plataformas tecnológicas que conforman el ecosistema de desarrollo de aplicaciones inmersivas web. Asimismo, se analizan los marcos de interacción existentes en la industria, abarcando tanto soluciones nativas como bibliotecas de código abierto, para establecer el estado del arte actual en la manipulación espacial.

El contenido se estructura comenzando por los lenguajes fundamentales de la web y las interfaces de programación (API) que permiten acceder al hardware de los visores. A continuación, se detalla la base matemática y geométrica necesaria para la detección de colisiones en entornos tridimensionales, evaluando de forma objetiva las ventajas y desventajas de los distintos enfoques de simulación. Finalmente, se examinan las herramientas y motores de desarrollo más consolidados en el sector de la Realidad Extendida (XR).

2.1. Tecnologías base para el entorno inmersivo web

El desarrollo de aplicaciones inmersivas ejecutables directamente en un navegador requiere la coordinación de múltiples capas tecnológicas. Para que el contenido sea comprensible y fácil de mantener, el proyecto se apoya en librerías que abstraen la complejidad visual. A continuación se explican las tecnologías que componen esta arquitectura, ordenadas por su relevancia directa en el código del Trabajo de Fin de Grado.

2.1.1. A-Frame y el patrón arquitectónico ECS

Construir escenas 3D operando a bajo nivel resulta demasiado árido y complejo de mantener. Para solucionar este problema de accesibilidad al código, el equipo de Mozilla creó A-Frame, un marco de trabajo (*framework*) de código abierto diseñado para maquetar entornos inmersivos utilizando simplemente etiquetas HTML [13].

La verdadera aportación de A-Frame al ecosistema de desarrollo radica en su arquitectura interna, la cual se rige de manera estricta por el patrón de diseño Entidad-Componente-Sistema (ECS, *Entity-Component-System*). En la ingeniería de software clásica (programación orientada a objetos), las propiedades se heredan de arriba abajo (por ejemplo, un "Coche" hereda las propiedades de la clase "Vehículo"). Esto genera jerarquías rígidas y problemáticas cuando se buscan comportamientos mixtos en entornos tridimensionales. El patrón ECS soluciona este obstáculo teniendo una arquitectura más modular y menos estricta.

El modelo se divide en tres pilares:

- **Entidades (*Entities*):** Actúan como contenedores totalmente vacíos. En el código HTML se escriben como la etiqueta `<a-entity>`. Carecen de cualquier comportamiento lógico o apariencia visual; solo existen como una coordenada en el espacio.

- **Componentes (*Components*):** Son pequeños bloques de código independiente escritos en JavaScript que otorgan propiedades específicas. Si a una entidad vacía se le acopla un componente de "geometría" y otro de "material", se transforma visualmente en una figura.

- **Sistemas (*Systems*):** Operan en un nivel superior, gestionando los datos compartidos o la comunicación global de todos los componentes de una misma familia presentes en la escena.

Esta estricta separación modular permite desarrollar interacciones complejas de forma muy ordenada y legible. Como se observa en el Código 2.1, apenas unas líneas de marcado HTML bastan para definir primitivas geométricas, colores y posiciones. El motor subyacente se encarga de interpretar estas etiquetas y generar automáticamente el entorno tridimensional completo que el usuario visualiza en el navegador (Figura 2.1).

Código 2.1: Ejemplo de escena declarativa básica en A-Frame.

```
<html>
  <head>
    <script src="https://aframe.io/releases/1.7.1/aframe.min.js"></script
       >
  </head>
  <body>
    <a-scene>
      <a-box position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9"></a-
         box>
      <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-
         sphere>
      <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5" color="#
         FFC65D"></a-cylinder>
      <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4"
         color="#7BC8A4"></a-plane>
      <a-sky color="#ECECEC"></a-sky>
    </a-scene>
  </body>
</html>
```

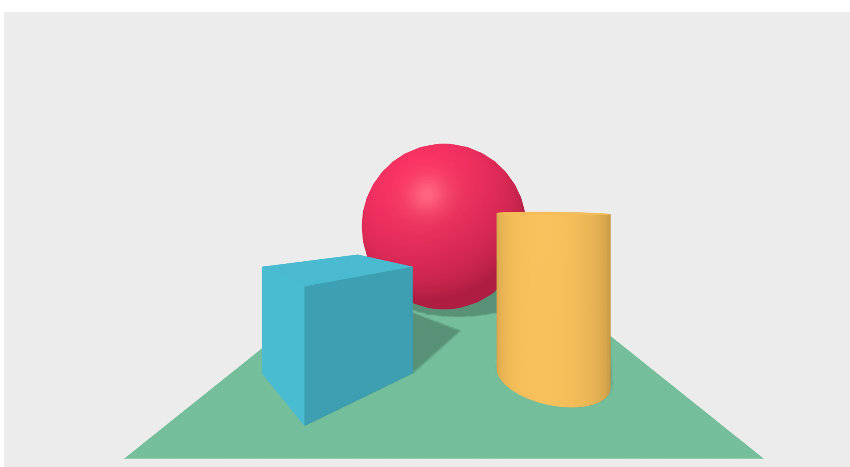


Figura 2.1: Representación gráfica en el navegador generada a partir del código declarativo anterior.

2.1.2. WebXR Device API

Para que las escenas creadas en A-Frame se puedan ver y tocar en unas gafas de realidad virtual, el sistema necesita conectarse con el hardware del dispositivo. WebXR es la API estándar actual mantenida por el W3C (*World Wide Web Consortium*) que unifica esta comunicación entre los navegadores de internet y los visores [15]. Esta especificación reemplazó a la antigua WebVR, la cual presentaba la limitación de estar enfocada exclusivamente en entornos virtuales cerrados. WebXR introdujo el soporte nativo para el mapeo espacial del entorno físico y el acceso a las cámaras externas del dispositivo, características indispensables para integrar elementos digitales sobre el mundo real mediante vídeo (*passthrough*).

Uno de los módulos técnicos más avanzados y complejos de WebXR es el seguimiento óptico de manos (*Hand Tracking API*). Tradicionalmente, interactuar en un entorno virtual requería pulsar los botones físicos de los mandos del visor. WebXR permite prescindir completamente de este hardware adicional. Para lograrlo, utiliza algoritmos de visión artificial integrados en el propio visor que escanean las manos del usuario en tiempo real y deducen su postura anatómica.

La API no se limita a ubicar un punto central de la mano en el espacio, sino que proporciona un esqueleto digital estandarizado compuesto por 25 articulaciones (*joints*). Como se ilustra en la Figura 2.2, este esqueleto entrega una matriz matemática para cada una de las articulaciones en cada frame. Estas matrices detallan la posición exacta en el eje X, Y y Z, así como la orientación y el radio de la muñeca, la palma y todas las falanges. El procesamiento algorítmico de todas estas coordenadas espaciales es lo que permite que una aplicación determine, por ejemplo, si el dedo índice está estirado o si la punta del pulgar está tocando otro dedo.

2.1.3. Three.js y la representación gráfica

En el fondo, A-Frame funciona como una capa de traducción amigable, pero es Three.js quien realiza el trabajo pesado. Esta última es la biblioteca de JavaScript predominante en la industria para el renderizado 3D web [2].

Three.js es el verdadero motor visual de la aplicación. Se encarga de gestionar cómo la luz rebota en las superficies, proyectar las sombras en tiempo real, cargar modelos 3D complejos (como los formatos GLTF o OBJ) y aplicarles texturas y materiales para que parezcan de madera, metal o cristal. Abstrae toda la enorme complejidad técnica que supone dibujar píxeles en

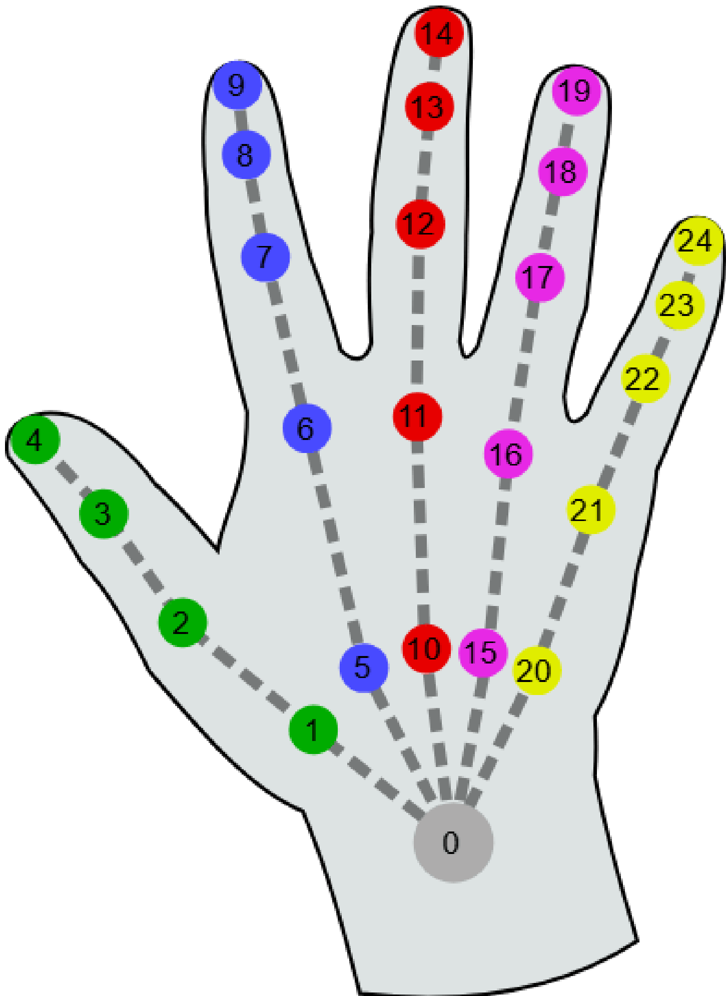


Figura 2.2: Representación anatómica de las 25 articulaciones (*joints*) proporcionadas por la API de seguimiento de manos de WebXR.

pantalla, ofreciendo clases preparadas para crear cámaras y organizar el grafo de la escena de forma sencilla.

Cualquier desarrollo de interacciones avanzadas en A-Frame termina requiriendo el uso de la API matemática de Three.js. Aunque el motor visual lo hace todo más fácil, para programar interacciones manuales hay que utilizar sus herramientas de álgebra lineal. Por ejemplo, su clase `THREE.Vector3` permite medir distancias físicas entre objetos (como saber a cuántos centímetros está la mano de un botón).

También incluye funciones de cálculo matricial que resultan indispensables durante procesos como el agarre. Al coger una pieza, esta cambia de padre en la jerarquía de la escena, alterando su sistema de coordenadas local. La aplicación matemática de matrices inversas so-

bre los vértices permite que la figura mantenga su posición relativa en pantalla durante esa transición, evitando caídas o saltos gráficos extraños al mover la mano.

2.1.4. HTML5

Para organizar todos estos elementos y construir la estructura de la aplicación se utiliza HTML5 (*HyperText Markup Language*, versión 5). Tradicionalmente, HTML se ha utilizado para definir la maquetación de documentos de texto, imágenes y enlaces en dos dimensiones mediante el uso de etiquetas. Toda esta jerarquía de etiquetas conforma lo que se conoce como el Modelo de Objetos del Documento o DOM (*Document Object Model*).

En el contexto del desarrollo de Realidad Virtual (VR) y Realidad Aumentada (AR) a través de marcos de trabajo declarativos, el papel de HTML5 evoluciona drásticamente. El DOM deja de representar una página plana para convertirse en el árbol de jerarquías espaciales de la escena tridimensional. A través de etiquetas específicas, se instancian entidades como luces, cámaras y geometrías.

Esta correspondencia directa entre el código HTML y el mundo 3D resulta fundamental para la accesibilidad tecnológica. Permite que cualquier desarrollador web tradicional, sin experiencia previa en motores de videojuegos complejos, pueda comprender y estructurar un entorno inmersivo simplemente anidando elementos. Por ejemplo, si se desea que un objeto tridimensional rote alrededor de otro, basta con colocar la etiqueta HTML del primer objeto dentro de la del segundo, estableciendo una relación automática de padre e hijo en el espacio virtual.

2.1.5. JavaScript y la arquitectura de eventos

Si HTML5 aporta el esqueleto estático de la aplicación, JavaScript (en sus especificaciones ES6 y superiores) actúa como el encargado de dar la lógica y el comportamiento dinámico. Es un lenguaje de programación interpretado que se ejecuta en el hilo principal del navegador del usuario.

En un entorno de Realidad Extendida, JavaScript tiene dos responsabilidades críticas. La primera es gestionar el bucle de renderizado (*render loop* o *tick*). Esta función se ejecuta de manera continua, habitualmente entre 60 y 90 veces por segundo, sincronizada con la tasa de refresco de las pantallas del visor para evitar mareos en el usuario. Dentro de este bucle, Ja-

vaScript procesa enormes cantidades de datos matemáticos, lee las coordenadas de las manos y actualiza la posición de los polígonos.

La segunda responsabilidad, igual de importante, es aprovechar la arquitectura de eventos asíncronos del propio DOM del navegador web. En lugar de bloquear el procesador comprobando continuamente si ocurren cosas en la escena, JavaScript permite que el navegador actúe como un vigilante. Por ejemplo, a la hora de gestionar choques físicos, en lugar de calcular colisiones manualmente en cada frame, se puede delegar la tarea a componentes del DOM que emiten señales estándar de texto (como los eventos `obbcollisionstarted` u `obbcollisionended`).

Apoyarse en esta estructura de eventos asíncronos tiene una ventaja enorme a nivel de diseño de software: permite que distintos componentes del código hablen entre sí de forma estandarizada. Un botón de la interfaz puede simplemente "escuchar"si recibe el evento de colisión para cambiar de color y hacer clic, sin importarle en absoluto las matemáticas que hay por debajo.

2.1.6. WebGL

Para poder dibujar gráficos tridimensionales complejos en la pantalla a la velocidad exigida por la VR, JavaScript necesita ayuda de la tarjeta gráfica (GPU) del dispositivo. Aquí es donde entra en juego la tecnología base más profunda: WebGL (*Web Graphics Library*). WebGL es una interfaz de programación de aplicaciones (API) de bajo nivel, basada en el estándar OpenGL ES, que permite renderizar gráficos 2D y 3D interactivos en cualquier navegador compatible sin necesidad de instalar complementos externos.

WebGL funciona comunicándose directamente con la GPU a través de pequeños programas llamados *shaders*. Existen dos tipos principales: los *vertex shaders*, que calculan dónde debe colocarse cada punto (vértice) de un modelo 3D en el espacio, y los *fragment shaders*, que calculan el color exacto y la iluminación de cada píxel en la pantalla.

Aunque WebGL es extremadamente potente, su curva de aprendizaje es muy pronunciada. Dibujar un simple cubo texturizado en pantalla operando directamente con WebGL puede requerir cientos de líneas de código matemático complejo y gestión manual de la memoria de la tarjeta gráfica. Debido a esta complejidad técnica, el desarrollo moderno rara vez ataca directamente a WebGL; en su lugar, se utilizan librerías de abstracción de mayor nivel (como las ya mencionadas Three.js y A-Frame) que simplifican el proceso y traducen órdenes sencillas a

lenguaje de GPU.

2.2. Tecnologías y algoritmos para la detección de colisiones

Para que un entorno virtual ofrezca una experiencia creíble, el software debe ser capaz de determinar en qué instante exacto la malla tridimensional de la mano entra en contacto con la superficie de un elemento del escenario. Resolver de forma eficiente este problema requiere conocer y evaluar el funcionamiento técnico de los distintos modelos espaciales disponibles.

Una opción común en la industria del videojuego es el uso de motores de dinámicas de cuerpos rígidos, como Cannon.js.

- **Ventajas:** Proveen un alto grado de realismo simulado. Calculan automáticamente la masa de los objetos, la gravedad y la fricción de los materiales, permitiendo que los polígonos caigan, reboten o se amontonen de forma muy orgánica.

- **Desventajas:** Exigen una alta carga de procesamiento en la CPU. Además, presentan serios conflictos de estabilidad al interactuar con el seguimiento óptico de manos libres. Las cámaras del visor sufren pequeñas pérdidas de precisión al leer los dedos, generando un temblor continuo (*jitter*). Al aplicar físicas estrictas sobre este esqueleto tembloroso, las micro-vibraciones se interpretan matemáticamente como aceleraciones violentas, provocando un comportamiento caótico donde los objetos salen despedidos al intentar manipularlos.

Frente a la inestabilidad de la simulación física, la alternativa tecnológica es el cálculo directo mediante geometría pura. Este enfoque elimina variables como el peso o la energía cinética, limitándose a verificar si un volumen ha cruzado el límite espacial de otro. Su gran beneficio es la estabilidad predictiva y el bajo consumo de recursos, garantizando un funcionamiento robusto frente a las imperfecciones de los sensores ópticos.

2.2.1. Volúmenes delimitadores: AABB frente a OBB

Para mantener un rendimiento óptimo, el software de colisiones rara vez evalúa cada triángulo individual de una malla geométrica detallada. La técnica estándar consiste en envolver el

modelo 3D en un volumen prismático invisible mucho más simple de procesar. A la hora de definir estas cajas, existen dos modelos matemáticos predominantes.

El modelo más básico es la caja alineada a los ejes o AABB (*Axis-Aligned Bounding Box*). Este prisma mantiene sus caras bloqueadas de forma paralela a las coordenadas maestras (X, Y, Z) del entorno. Al ser una estructura recta y fija, el cálculo de impactos es una operación trivial y casi instantánea para el procesador.

Sin embargo, el formato AABB presenta una limitación crítica arquitectónica: la incapacidad de rotar. Si una malla cilíndrica se inclina, la caja AABB no gira con ella. Para lograr contener el objeto ladeado, el prisma se ve obligado a expandirse masivamente, generando grandes bolsas de espacio vacío en sus esquinas. Esto se traduce en colisiones falsas positivas, ya que el sistema detecta que un dedo ha tocado la caja invisible mucho antes de rozar la superficie real del modelo 3D.

Para resolver los problemas de precisión en entornos manipulables, el estándar tecnológico es el uso de cajas delimitadoras orientadas u OBB (*Oriented Bounding Box*) [1]. A diferencia del modelo anterior, una caja OBB posee la capacidad de rotar de forma solidaria junto a la entidad a la que envuelve. Al mantener sus ejes locales vinculados a la figura geométrica, la OBB se mantiene ceñida herméticamente (*tight fit*) a la topología del objeto en todo momento, independientemente del ángulo de inclinación que este adopte en el espacio.

2.2.2. El Teorema SAT y su integración en el DOM

Cuando un sistema gráfico implementa cajas orientadas, requiere un método matemático específico para determinar si dos polígonos OBB han comenzado a intersectar. La técnica algorítmica por excelencia para resolver esta evaluación es el Teorema de Ejes Separados o SAT (*Separating Axis Theorem*) [3].

El fundamento de este teorema dictamina que, si es posible encontrar un plano recto que separe por completo a dos formas convexas, se puede afirmar con rotundidad que no están chocando. Para comprender su funcionamiento visual, se puede utilizar la analogía de la proyección de sombras. Si se iluminan dos figuras desde diferentes ángulos y, en al menos una de las sombras proyectadas sobre una pared, se distingue un espacio en blanco entre ellas, se confirma que no se tocan. En la programación tridimensional, el algoritmo SAT replica esta comprobación proyectando matemáticamente los vértices de las cajas sobre 15 ejes direccionales. Si en cual-

quiera de estos ejes las proyecciones no se superponen, la iteración se detiene de inmediato, ahorrando un valioso tiempo de cálculo.

A nivel de implementación en la web, escribir la matemática del SAT desde cero dentro del bucle de renderizado es una opción válida, pero los marcos de trabajo modernos suelen ofrecer soluciones integradas para aligerar el desarrollo. Por ejemplo, en el ecosistema de A-Frame se dispone del componente `obb-collider` [12]. Esta herramienta nativa realiza todos los cálculos iterativos del Teorema SAT en segundo plano. Cuando el componente detecta que los vértices se cruzan, lanza eventos asíncronos en el DOM (como el evento `obbcollisionstarted`). El uso de estos eventos permite que el desarrollo de interfaces sea modular, ya que los elementos visuales pueden reaccionar de forma reactiva a los impactos sin recalcular la matemática espacial manualmente.

2.2.3. Lanzamiento de rayos (Raycasting)

Otra tecnología fundamental para la interacción espacial, complementaria a la evaluación de cajas volumétricas, es el algoritmo de lanzamiento de rayos, conocido en el ámbito gráfico como *Raycasting* [16].

Mientras que un sistema OBB verifica si dos volúmenes ocupan el mismo lugar físico, el *Raycasting* opera proyectando una línea recta unidimensional e invisible (un rayo matemático) desde unas coordenadas de origen hacia una dirección vectorial específica. El algoritmo calcula en tiempo real si esta trayectoria atraviesa la superficie de algún polígono presente en el entorno, devolviendo un conjunto de datos muy precisos: la coordenada exacta de la intersección, la distancia total desde el origen y la cara concreta del modelo 3D que ha recibido el impacto.

En el diseño de experiencias inmersivas, el lanzamiento de rayos es la herramienta predilecta para dar soporte a las interacciones remotas (*remote interactions*). Su uso principal es la simulación de punteros láser, lo que permite a los usuarios seleccionar elementos lejanos o navegar por menús de interfaz flotantes sin necesidad de desplazarse físicamente por la sala. Al evitar la pesada evaluación de volúmenes entrelazados, procesar un impacto mediante una línea matemática resulta extremadamente ligero a nivel computacional, consolidando esta técnica como el estándar idóneo para la activación de zonas interactivas (*clickables*) en entornos de realidad virtual.

2.3. Trabajos relacionados y motores de la industria

La manipulación espacial de polígonos es un área de estudio ampliamente abordada en el sector del software. El estado del arte está compuesto tanto por librerías de código abierto para entornos web como por potentes motores comerciales (*Game Engines*) que marcan el estándar de calidad en el desarrollo nativo de aplicaciones de realidad extendida.

2.3.1. A-Frame Super Hands: La influencia arquitectónica

Dentro de la comunidad específica del desarrollo web para realidad virtual, el paquete *Super-Hands* de código abierto se ha utilizado para resolver la manipulación de entidades en escenas creadas con A-Frame [11].

La mayor virtud de esta biblioteca es su excelente diseño estructural basado en el patrón ECS. Propone una separación semántica muy elegante: por un lado, define componentes de "acción"(*grabber*, *stretcher*) que se asocian a los mandos del usuario y buscan colisiones. Por otro lado, define componentes de "reacción"(*grabbable*, *stretchable*, *draggable*) que se asignan a los modelos 3D y reaccionan al ser interceptados. Esta filosofía modular de *Super-Hands* constituye la principal inspiración arquitectónica del presente trabajo, ya que demostró que separar la lógica en pequeñas piezas de comportamiento genera código más limpio y comprensible.

El inconveniente fundamental de *Super-Hands*, y el motivo que justifica la existencia de este TFG, es que fue conceptualizado en una época donde el hardware de la VR dependía de los controladores físicos (como los antiguos mandos de HTC Vive o las primeras generaciones de Oculus Touch). Su código interno está acoplado a la escucha de eventos de hardware tradicionales, como la pulsación de un gatillo analógico (`triggerdown`) o el botón lateral de agarre (`gripdown`).

Al introducir el seguimiento óptico de manos libres en los visores modernos, este paradigma basado en botones queda obsoleto. Las manos reales no disponen de interruptores; su posición es analógica, continua y cambiante. El objetivo primordial de este trabajo ha consistido, por tanto, en abstraer la filosofía de estado de *Super-Hands* y trasladarla a un entorno puramente regido por el análisis matemático de gestos naturales, cálculos de distancia espacial y colisiones aplicadas sobre los puntos articulares de la anatomía de las manos.

2.3.2. SDKs nativos comerciales: Meta Interaction SDK

Fuera del ámbito académico y del ecosistema de navegadores web, el verdadero estado del arte en lo que respecta a la interacción manual en entornos virtuales lo dictan las grandes corporaciones del sector, destacando el *Meta Quest Interaction SDK* [7]. Este conjunto de herramientas está diseñado para operar bajo motores de videojuegos de gran calado comercial, como Unity o Unreal Engine.

Estos SDK (*Software Development Kit*) nativos abordan el problema con un nivel de complejidad industrial. Solucionan carencias críticas de la visión artificial mediante algoritmos patentados, tales como la predicción topológica de agarres (adivinar cómo rodearán los dedos un objeto en función de su forma geométrica antes de que se produzca el contacto), la cinemática inversa del brazo completo para disimular la oclusión temporal de las manos por parte del cuerpo del usuario, o el uso de redes neuronales ligeras para estabilizar el temblor de la malla.

El estudio en profundidad de la documentación técnica del Interaction SDK de Meta ha servido durante este proyecto como guía funcional para definir el comportamiento "esperado" de los gestos por parte de un usuario estándar. Por ejemplo, la implementación del componente `clickable` de este TFG asimila la directriz del SDK de Meta que establece que un "toque" virtual solo debe registrarse si el usuario tiene el dedo índice completamente extendido mientras el resto de dedos se retraen hacia la palma de la mano, evitando colisiones erráticas con el reverso de las falanges.

Evidentemente, las capacidades algorítmicas implementadas en JavaScript para este trabajo fin de grado, que deben correr en el hilo de ejecución principal de un navegador Chrome o Firefox en el propio visor, no pueden ni buscan competir en rendimiento puro o simulación física pesada con herramientas comerciales compiladas en C++ o C# y de código cerrado. La aportación central de este TFG reside precisamente en el esfuerzo de democratización tecnológica: extraer las dinámicas funcionales básicas de interacción (pellizco, apuntado, estiramiento bimanual, arrastre) de esos entornos privativos, y reconstruirlas desde cero para hacerlas viables, ligeras y accesibles en el ecosistema de la web abierta y distribuible sin instalación.

2.3.3. Unity

Unity es uno de los motores de desarrollo multiplataforma (*Game Engines*) más predominantes en el mercado global[14]. Opera mediante el lenguaje de programación C# y estructura sus entornos sobre una filosofía de objetos y componentes anexados conocida como *MonoBehaviours*.

Su fortaleza central en el ámbito inmersivo reside en su proceso de compilación directa, que garantiza un acceso de bajo nivel a los recursos del hardware sin las intermediaciones de un navegador de internet. Unity integra de forma nativa los SDK de los principales fabricantes de visores (Oculus, Pico, SteamVR) e incorpora potentes simuladores de físicas de cuerpos rígidos. Gracias a su interfaz gráfica avanzada, el ajuste y calibración de colisionadores complejos se convierte en una tarea visual, logrando un nivel de rendimiento gráfico que supone el techo técnico actual de las aplicaciones VR nativas.

2.3.4. Godot Engine

Godot ha emergido en la industria como un potente motor de videojuegos de código abierto y distribuido bajo licencia MIT [6]. Su propuesta arquitectónica difiere significativamente del patrón ECS; organiza el proyecto basándose en una estructura orientada a objetos estricta donde cualquier elemento de la pantalla es un Nodo (*Node*), y estos nodos se ensamblan formando árboles jerárquicos llamados Escenas (*Scenes*).

Este motor soporta lenguajes como C++ y C#, además de incluir GDScript, un lenguaje propio diseñado para agilizar el desarrollo de la lógica del entorno. En el contexto de la Realidad Extendida, las actualizaciones recientes de Godot han integrado soporte completo para el estándar OpenXR. Si bien su ecosistema de extensiones comunitarias orientadas específicamente a mecánicas avanzadas de seguimiento de manos es más reducido en comparación con alternativas comerciales, su accesibilidad libre y su capacidad para generar binarios nativos lo sitúan como una herramienta fundamental en el desarrollo de simuladores XR de código abierto.

2.4. Herramientas para el ciclo de vida del software

El desarrollo estructurado de software requiere la integración de herramientas instrumentales para gestionar el código, compilar pruebas y generar la documentación técnica, asegurando la trazabilidad de todo el proceso de ingeniería.

El entorno de desarrollo integrado (IDE) empleado es Visual Studio Code [8]. Es un editor de código fuente ligero que facilita el desarrollo web mediante extensiones para depuración en tiempo real. Herramientas como *Live Server* permiten levantar servidores HTTP locales para inyectar cambios de código directamente en el navegador del visor XR a través de la red de área local, agilizando el flujo de pruebas.

Para el control de versiones distribuido se utiliza Git [4]. Esta herramienta es un estándar en la industria para registrar el historial de cambios en el código fuente. Permite crear ramas independientes (*branches*) para desarrollar y testear nuevos algoritmos de colisión sin afectar a la estabilidad del código principal. El código fuente versionado se almacena y sincroniza en un repositorio remoto mediante la plataforma GitHub [5].

La redacción técnica y maquetación de la documentación formal se lleva a cabo mediante LaTeX [9]. Este sistema de composición de textos, ampliamente utilizado en el ámbito académico, separa el contenido de la presentación visual. El entorno de compilación colaborativo basado en la nube utilizado es Overleaf [10], el cual facilita el cumplimiento de las normativas tipográficas y automatiza la gestión estructurada de la bibliografía a través del formato BibTeX.

Capítulo 3

Resultados obtenidos

En este capítulo se exponen los resultados logrados tras la implementación del proyecto. El contenido se estructura en diversas secciones para abarcar tanto el diseño de la biblioteca como su validación práctica. En primer lugar, se presenta la arquitectura funcional, definiendo la tipología y el comportamiento de los componentes desarrollados. A continuación, se detalla la metodología para la construcción declarativa de escenas. Posteriormente, se abordan los detalles técnicos de implementación algorítmica y se establece una comparativa directa frente a la biblioteca clásica *Super-Hands*. Por último, se exponen los entornos interactivos creados como demostración final para el usuario.

3.1. Descripción funcional y de interfaz

El resultado técnico de este trabajo se materializa en una caja de herramientas (*toolkit*) modular orientada a desarrolladores. La implementación se ha basado íntegramente en la construcción de componentes personalizados aprovechando el patrón Entidad-Componente-Sistema (ECS) del marco de trabajo A-Frame.

El núcleo del sistema lee de forma continua la posición geométrica de las manos proporcionada por la API de WebXR. A partir de estas coordenadas espaciales, el código orquesta la interactividad mediante eventos del DOM, facilitando la construcción de entornos tridimensionales mediante simples etiquetas HTML. Para maximizar la modularidad y reducir el acoplamiento, la arquitectura se ha dividido en tres grandes familias funcionales:

- **Componentes de colisión:** Representan el motor físico subyacente. Se encargan de en-

volver los modelos tridimensionales en volúmenes matemáticos invisibles para calcular intersecciones exactas entre la mano y el objeto. Esta familia introduce el componente `sat-collider`.

- **Componentes gestuales:** Se vinculan de forma exclusiva a las manos del usuario y operan como el núcleo sensorial de la aplicación. Su función es detectar y clasificar los gestos de las manos basándose en las distancias anatómicas. Esta familia está formada por los componentes `hands-spheres`, `pinch-gesture` y `point-gesture`.

- **Componentes interactuables:** Se asignan a los objetos inanimados de la escena (cubos, paneles, modelos importados). Su propósito es cambiar el comportamiento gráfico y espacial de la malla cuando interactúan con las manos. Esta familia incluye los componentes `hoverable`, `grabbable`, `stretchable`, `draggable`, `droppable` y `clickable`.

3.1.1. Componentes de colisión

Estos módulos actúan como la infraestructura matemática de la biblioteca. Su responsabilidad exclusiva es definir los límites físicos de las entidades y proporcionar algoritmos de evaluación de solapamiento tridimensional, prescindiendo de simulaciones dinámicas de cuerpos pesados.

Componente `sat-collider`

Comportamiento funcional: Este componente dota a cualquier entidad tridimensional de una caja orientada de colisión (OBB). Su función es evaluar si el volumen de la mano ha penetrado el espacio físico del objeto. Para lograrlo, utiliza el Teorema de Ejes Separados (SAT). El algoritmo proyecta los vértices del modelo tridimensional sobre múltiples ejes para detectar intersecciones de forma altamente estable. Al ser una caja orientada, el volumen físico rota de manera solidaria junto a la figura geométrica. Esto mantiene una envoltura ajustada en todo momento y evita colisiones falsas positivas cuando el objeto se inclina en el aire.

- **Propiedades del esquema (Schema):**

- `size` (vector3, valor por defecto: `0.1, 0.1, 0.1`): Define las dimensiones exactas de la caja de colisión (anchura, altura y profundidad).
- `debug` (booleano, valor por defecto: `false`): Renderiza una estructura de alambres (*wireframe*) visible alrededor de la entidad. Resulta fundamental para calibrar de forma visual los límites del impacto durante el proceso de desarrollo de la escena.

- **Interacción interna:** A diferencia de los componentes interactuables clásicos, este módulo no emite eventos asíncronos directamente al DOM por defecto. Su diseño expone métodos matemáticos precisos que son consultados continuamente por la capa sensorial de las manos para confirmar el contacto antes de disparar un agarre o una pulsación táctil.

3.1.2. Componentes gestuales

Estos módulos actúan como el núcleo sensorial del sistema. Se instancian exclusivamente sobre las entidades que representan las manos virtuales. Su única responsabilidad es leer los datos geométricos provenientes de la API de WebXR, analizar la postura anatómica de los dedos y emitir eventos estandarizados a través del árbol del DOM cuando detectan una intención clara de interacción por parte del usuario.

Componente `pinch-gesture`

Comportamiento funcional: Este componente actúa como el detector principal para la manipulación espacial de objetos. Su función es reconocer la intención de agarre del usuario mediante la detección del gesto de pellizco. Para desencadenar esta acción, la persona debe juntar físicamente la yema del dedo pulgar y la punta del dedo índice (Figura 3.1). El sistema destaca por aplicar márgenes espaciales distintos para el inicio y el final del gesto. El propósito de esta separación es evitar activaciones y desactivaciones rápidas accidentales, un parpadeo constante que suele producirse cuando la mano del usuario tiembla de forma natural justo en el límite de detección de las cámaras.

- **Propiedades del esquema (Schema):**
  - `hand` (cadena de texto, valor por defecto: `'any'`): Define qué mano debe monitorizar el componente (`'left'`,`'right'` o `'any'`).

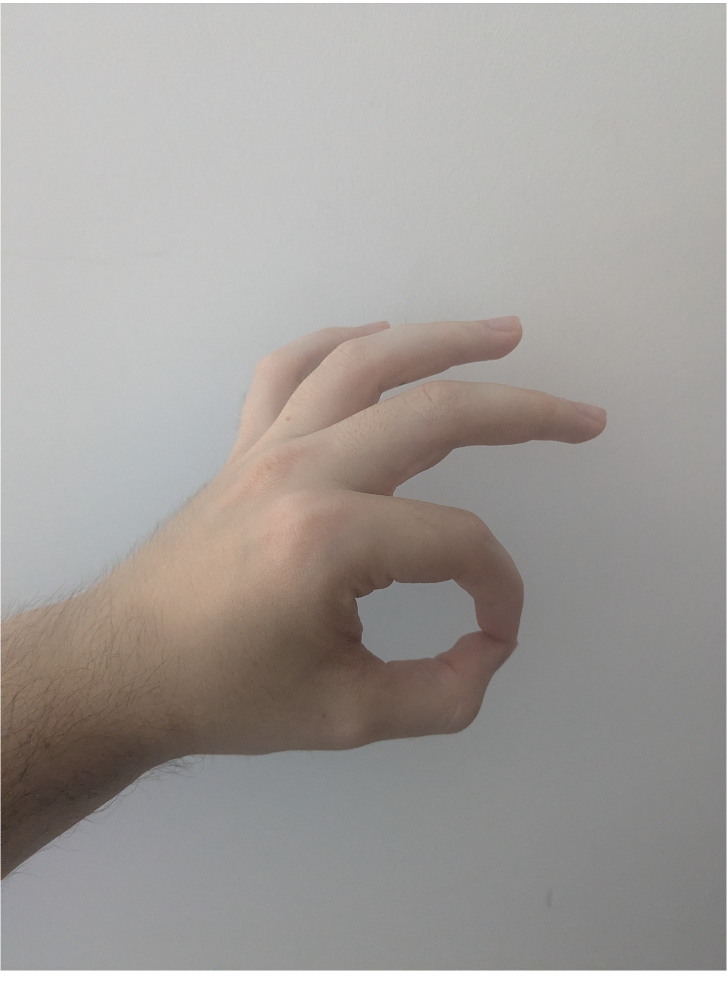


Figura 3.1: Representación de la postura anatómica necesaria para activar el gesto de pellizco.

- `startDistance` (número, valor por defecto: `0.025`): Distancia en metros (2,5 cm) por debajo de la cual se considera que el usuario ha cerrado los dedos y ejecutado el pellizco.
- `endDistance` (número, valor por defecto: `0.035`): Distancia en metros (3,5 cm) al superar la cual se considera que el usuario ha abierto la mano y soltado el pellizco.
- `colliderType` (cadena de texto, valor por defecto: `'sat-collider'`): Permite alternar entre el motor geométrico nativo de A-Frame (`'obb-collider'`) o la implementación matemática propia (`'sat-collider'`).
- `colliderSize` (vector3, valor por defecto: `0.12, 0.08, 0.18`): Dimensiones paramétricas del volumen de colisión de la mano.
- Otras propiedades incluyen banderas de configuración como `emitEachFrame` (si

esta activo emite eventos continuos durante el movimiento en cada frame), `log` y `debugCollider` (para hacer visible la caja de colisión).

- **Eventos emitidos:**
  - `pinchstart`: Se dispara en el frame exacto en el que la distancia entre los dedos desciende del umbral de inicio.
  - `pinchend`: Se dispara cuando la separación supera el umbral final, concluyendo la intención de agarre.
  - `pinchmove`: Se emite continuamente en cada frame si el pellizco se mantiene activo y la configuración lo permite.

Componente `point-gesture`

Comportamiento funcional: Este componente se ha diseñado específicamente para habilitar la interacción táctil con interfaces de usuario tridimensionales, como botones físicos o paneles flotantes. Para activar el detector, el usuario debe realizar de forma física el gesto de apuntar, extendiendo completamente el dedo índice hacia adelante mientras mantiene el resto de los dedos contraídos hacia la palma (Figura 3.2) . Para evitar conflictos de usabilidad con otras mecánicas, el módulo implementa un mecanismo de cancelación dinámica. El sistema considera que la posición del pulgar es irrelevante para apuntar, excepto en una situación concreta: si el índice y el pulgar se acercan demasiado, se asume que la verdadera intención de la persona es pellizcar un objeto. En ese instante, el gesto de apuntar se anula automáticamente para prevenir pulsaciones accidentales.

- **Propiedades del esquema (Schema):** El desarrollador puede calibrar la sensibilidad anatómica del gesto modificando los siguientes umbrales espaciales:
  - `indexExtendedThreshold` (número, valor por defecto: `0.06`): Distancia mínima en metros respecto a la muñeca para considerar que el dedo índice se encuentra estirado.
  - `otherFingersThreshold` (número, valor por defecto: `0.08`): Distancia máxima permitida para los dedos medio, anular y meñique, garantizando así que estén cerrados.

Figura 3.2: Representación de la postura anatómica de índice extendido requerida para interactuar con interfaces táctiles.

- `pinchCancelThreshold` (número, valor por defecto: `0.04`): Umbral de seguridad que cancela el gesto si el pulgar se acerca demasiado al dedo índice.

- **Eventos emitidos:**

  - `pointstart` y `pointend`: Señalan el inicio y la finalización de la postura válida. Emiten además un vector matemático direccional que indica la orientación exacta del dedo.

Componente `hands-spheres`

Comportamiento funcional: Este componente cumple una función dual dentro de la arquitectura. Por un lado, actúa como el puente visual entre el usuario y el entorno virtual, renderi-

zando la anatomía de las manos en tiempo real. Por otro lado, funciona como un administrador centralizado, permitiendo inyectar automáticamente los demás detectores gestuales desde una única entidad. Al consistir en un sistema de seguimiento, no requiere que el usuario realice un movimiento específico; su ejecución es pasiva. Su principal valor reside en la simplificación de cara al programador. En lugar de exigir que se instancien múltiples entidades invisibles en la escena, este módulo unifica el proceso. A nivel visual, optimiza el consumo de memoria gráfica instanciando las esferas geométricas una única vez, ocultándolas dinámicamente cuando las cámaras pierden el seguimiento de las manos.

- **Propiedades del esquema (Schema):** Debido a su doble naturaleza, sus propiedades se dividen en parámetros estéticos y de configuración lógica:
  - **Propiedades visuales:** `colorLeft`, `colorRight` y `opacity` definen el aspecto cromático y la transparencia de la malla anatómica. Además, incluye la opción `labels` para superponer un texto con el nombre técnico de cada articulación.
  - **Propiedades geométricas:** `radius`, `useJointRadius`, `minRadius` y `maxRadius`. Permiten calibrar el tamaño de las esferas adaptándose dinámicamente a las estimaciones reales proporcionadas por el visor.
  - **Inyección de gestos:** `enablePinch`, `enablePoint` y `gestureColliderType`. Permiten activar los detectores lógicos de forma automática, heredando el tipo de colisionador deseado sin necesidad de crear nuevas entidades.
- **Eventos emitidos:**
  - A diferencia de los detectores específicos, este módulo es puramente estético y de configuración. No emite eventos lógicos de interacción propios, ya que delega esta responsabilidad en los componentes que inyecta.

3.1.3. Componentes interactuables

Esta segunda familia de módulos se asigna exclusivamente a los objetos virtuales inanimados (como cubos, esferas o modelos importados). Su comportamiento es pasivo: escuchan los eventos emitidos por las manos y aplican transformaciones físicas a las mallas tridimensionales.

Para garantizar la usabilidad, varios de estos componentes inyectan dinámicamente sus propias dependencias si el desarrollador omite declararlas.

Componente `hoverable`

Comportamiento funcional: Este componente se encarga de proporcionar retroalimentación visual al usuario durante la fase de aproximación a un objeto. Su función es indicar de manera clara que una entidad es interactiva y que la mano se encuentra dentro de su rango de alcance. A diferencia de otros módulos gestuales, no requiere la ejecución de un movimiento de dedos concreto; se activa de forma pasiva por la simple proximidad o contacto físico. El módulo actúa como una herramienta de señalización plenamente compatible con el resto de mecánicas. Su principal utilidad es servir de disparador para cambios estéticos temporales, permitiendo modificar el color o brillo de una pieza para confirmar que se puede iniciar una interacción más compleja.

- **Propiedades del esquema (Schema):**
  - `colliderSize` (vector3, valor por defecto: `0.3, 0.3, 0.3`): Define las dimensiones de la caja de detección que se asignará al objeto si este carece de una geometría previa.
  - `emitEachFrame` (booleano, valor por defecto: `false`): Determina si el sistema debe emitir eventos de proximidad de forma continua en cada frame.
  - `debug` (booleano, valor por defecto: `false`): Permite visualizar de forma gráfica los límites del colisionador invisible.
- **Eventos emitidos y estados:**
  - `hover-start` y `hover-end`: Se disparan respectivamente al entrar y al salir del área de influencia del objeto.
  - `hovering`: Se emite opcionalmente para realizar un seguimiento continuo del contacto.
  - **Gestión de estado:** El componente añade el estado `hovered` a la entidad en el árbol del DOM mientras la mano permanece en su interior.

Componente `grabbable`

Comportamiento funcional: Este componente constituye el pilar fundamental de la manipulación espacial. Otorga a cualquier entidad geométrica la capacidad de ser recogida, trasladada y soltada libremente por la escena de forma intuitiva. Para activarlo, el usuario debe acercar su mano hasta tocar el objeto y ejecutar un gesto de agarre. Al abrir los dedos, el objeto se libera en su nueva posición. El sistema garantiza que el objeto mantenga su distancia y rotación exactas respecto a la mano durante todo el movimiento. Además, implementa una regla estricta de seguridad: el contacto físico debe ocurrir siempre antes de iniciar el gesto, evitando así comportamientos magnéticos erráticos si el usuario cierra los dedos en el aire y roza la figura por accidente.

- **Propiedades del esquema (Schema):**
  - `maxGrabbers` (entero, valor por defecto: `NaN`): Limita cuántas manos pueden sujetar la pieza de forma simultánea. Si no se define, es ilimitado.
  - `invert` (booleano, valor por defecto: `false`): Invierte el vector de desplazamiento, moviendo la entidad en la dirección opuesta a la mano.
  - `suppressY` (booleano, valor por defecto: `false`): Bloquea el eje vertical del objeto, restringiendo su traslado a un plano estrictamente horizontal.
  - `startGesture` y `endGesture` (cadenas de texto, valores por defecto: `'pinchstart'` y `'pinchend'`): Definen qué eventos inician y finalizan la acción.
- **Eventos emitidos y estados:**
  - `grab-start` y `grab-end`: Se disparan al iniciar o finalizar el agarre, notificando cuántos agarres simultáneos existen.
  - **Gestión de estado:** Añade dinámicamente el estado `grabbed` al árbol del DOM de la entidad.

Componente `stretchable`

Comportamiento funcional: Este componente permite modificar el tamaño o la escala tridimensional de un objeto de forma interactiva. Para activarlo, el usuario debe establecer contacto con la pieza utilizando ambas manos de forma simultánea y ejecutar el gesto de pellizco.

Al separar las manos, el objeto se agranda; al juntarlas, se encoge. Para que un objeto pueda estirarse de forma lógica, primero debe poder sujetarse. Por este motivo, este módulo inyecta de forma automática las dependencias de agarre necesarias si el programador olvida incluirlas. Durante la acción de escalado, la figura se bloquea en su posición espacial para impedir desplazamientos involuntarios derivados del movimiento de los brazos, actualizando de forma invisible sus límites de colisión para mantener la precisión física.

- **Propiedades del esquema (Schema):**
    - `minScale` y `maxScale` (números, valores por defecto: `0.1` y `10.0`): Establecen topes de seguridad para evitar que la entidad crezca o se reduzca en exceso.
    - `invert` (booleano, valor por defecto: `false`): Invierte la relación espacial, provocando que el objeto encoja al separar las manos.
- **Eventos emitidos y estados:**
    - `stretch-start`, `stretch` y `stretch-end`: Notifican a la escena el ciclo de vida de la manipulación, transmitiendo el factor multiplicador en tiempo real.
    - **Gestión de estado:** Añade dinámicamente el estado `stretched` a la entidad mientras dura la deformación.

Componente `draggable`

Comportamiento funcional: Este componente permite que una entidad actúe como el origen en un ciclo interactivo de arrastrar y soltar (*drag and drop*). Para activarlo, el usuario debe tocar el objeto y ejecutar un gesto sostenido. A diferencia de la mecánica de agarre clásica, este módulo no mueve visualmente el modelo tridimensional ni altera sus coordenadas en el espacio. Su función es puramente semántica y organizativa. El sistema evalúa el contacto y el gesto, registrando el inicio del arrastre. Cuando el usuario abre los dedos, el sistema libera este estado e informa de manera indirecta a cualquier zona objetivo subyacente de que la pieza ha sido depositada de forma intencionada.

- **Propiedades del esquema (Schema):**

- `startGesture` y `endGesture` (cadenas de texto, valores por defecto: `'pinchstart'` y `'pinchend'`): Configuran qué eventos inician y terminan el arrastre.
- `debug` (booleano, valor por defecto: `false`): Activa la representación visual de la caja de colisión.

- **Eventos emitidos y estados:**
  - `drag-start` y `drag-end`: Notifican a la escena que la entidad ha comenzado o finalizado un desplazamiento, informando a las zonas de recepción.
  - **Gestión de estado:** Añade dinámicamente el estado `dragged` al árbol del DOM.

Componente `droppable`

Comportamiento funcional: Este componente configura un área espacial como una zona objetivo capaz de recibir y evaluar otras entidades. Su función es cerrar el ciclo interactivo de arrastrar y soltar. El usuario interactúa con este módulo al trasladar físicamente un objeto arrastrable hasta el interior de la zona y soltar la pieza. El componente vigila de forma constante su propio volumen. Al detectar que un objeto cae en su interior, evalúa inmediatamente si este cumple con una serie de filtros o reglas predefinidas. En función del resultado, la zona acepta o rechaza el objeto, completando flujos de trabajo como el ensamblaje de piezas sin obligar al programador a gestionar matemáticas de colisión complejas.

- **Propiedades del esquema (Schema):**
  - `accepts` (cadena de texto, valor por defecto: `''`): Define un selector CSS válido. Solo si el objeto arrastrado coincide con este selector, se considerará válido.
  - `autoUpdate` (booleano, valor por defecto: `true`): Permite que la lista de objetos válidos se actualice automáticamente si la escena cambia.
  - `acceptEvent` y `rejectEvent` (cadenas de texto, valores por defecto: `''`): Permiten definir nombres personalizados para los eventos de respuesta.
  - `debug` (booleano, valor por defecto: `false`): Activa la representación visual de la caja de colisión.

- **Eventos emitidos y estados:**

- `drag-drop`: Evento principal disparado exclusivamente cuando un objeto válido es depositado correctamente.
- **Gestión de estado:** Añade el estado `dragover` a la zona de caída cuando un objeto aceptable flota sobre ella, permitiendo generar efectos visuales previos a soltarlo.

Componente `clickable`

Comportamiento funcional: Este componente convierte cualquier entidad tridimensional en un botón reactivo táctil, resultando ideal para crear interfaces de usuario flotantes. El módulo detecta la interacción física pero no mueve el objeto visualmente. El usuario interactúa extendiendo el dedo índice y tocando el volumen del botón. El flujo de uso garantiza un alto grado de precisión: el sistema reacciona exclusivamente a la punta del dedo índice, ignorando roces accidentales con la palma o el dorso de la mano. Adicionalmente, cuenta con un mecanismo de seguridad para impedir dobles pulsaciones involuntarias; si el usuario mantiene el dedo quieto dentro del botón, solo se registrará una pulsación válida hasta que la extremidad salga por completo de la zona geométrica.

- **Propiedades del esquema (Schema):**
  - `startGesture` y `endGesture` (cadenas de texto, valores por defecto: `'pointstart'` y `'pointend'`): Definen qué eventos inician y terminan la pulsación.
  - `maxClickers` (entero, valor por defecto: `NaN`): Limita cuántas manos pueden presionar el botón de forma simultánea.
  - `colliderSize` (vector3, valor por defecto: `0.3, 0.3, 0.3`): Establece las dimensiones de la caja de detección.
  - `debug` (booleano, valor por defecto: `false`): Activa la representación visual del volumen interactivo.
- **Eventos emitidos y estados:**
  - `click-start` y `click-end`: Se disparan al inicio y al final de la pulsación, transmitiendo la duración exacta del clic.
  - **Gestión de estado:** Añade el estado `clicked` al árbol del DOM mientras el botón se mantenga presionado.

3.2. Construcción de escenas y ejemplos de uso

El diseño de la biblioteca desarrollada permite construir entornos inmersivos de forma puramente declarativa. Al integrarse sobre el patrón de A-Frame, la composición de una escena interactiva no requiere programar lógica matemática adicional en JavaScript por parte del usuario. El desarrollador únicamente debe estructurar el documento HTML, importar el paquete consolidado y añadir los atributos correspondientes a las entidades tridimensionales.

A continuación, se detalla el proceso de composición utilizando como referencia un entorno interactivo mínimo. El Código 3.1 muestra la estructura completa necesaria para instanciar un cubo manipulable con las manos.

Código 3.1: Estructura unificada de una escena interactiva con seguimiento de manos.

```
<head>
    <title>Ejemplo Basico de Manipulacion</title>
    <script src="https://aframe.io/releases/1.6.0/aframe.min.js"></script
       >
    <script>
        delete AFRAME.components["grabbable"];
    </script>
    <script src="https://cdn.jsdelivr.net/gh/rjimenezz/aframe-free-hands-
       component@v1.0.5/dist/free-hands.min.js"></script>
</head>
<body>
    <a-scene  webxr="optionalFeatures: hand-tracking">

        <a-entity hands-spheres="enablePinch:true; enablePoint:true;
           gestureColliderType:obb-collider">
        </a-entity>

        <a-box hoverable grabbable stretchable draggable droppable
               color="blue" position="0 1 -1">
        </a-box>
    </a-scene>
</body>
```

3.2.1. Llamada a la API de seguimiento de manos

El paso inicial y más crítico en la configuración de la escena es la habilitación de los sensores del dispositivo. Como se observa en la etiqueta raíz `<a-scene>` del código anterior, es imperativo declarar el parámetro `webxr="optionalFeatures: hand-tracking"`.

Esta instrucción constituye la llamada directa a la API de WebXR. Sin ella, el navegador web ignora las cámaras de paso (*passthrough*) y el sistema de seguimiento óptico del dispositivo físico. Al incluirla, se autoriza a la sesión de realidad virtual a capturar y transmitir en tiempo real la matriz de coordenadas de las 25 articulaciones anatómicas de cada mano.

3.2.2. Importación unificada y prevención de conflictos

En la cabecera del documento (`<head>`), el proceso se ha simplificado mediante el uso de un único punto de entrada empaquetado (`free-hands.min.js`). Esta optimización arquitectónica elimina la necesidad de enlazar secuencialmente múltiples archivos independientes. El script comprimido contiene en su interior toda la infraestructura matemática de colisiones, los detectores gestuales y los componentes interactuables.

Para garantizar la estabilidad del entorno, se mantiene el bloque de ejecución inmediata que elimina el componente `grabbable` nativo de la memoria de A-Frame. De este modo, se previenen activaciones cruzadas y conflictos de nombres, forzando al motor gráfico a interpretar las interacciones bajo la lógica diseñada en este proyecto.

3.2.3. Inicialización de mallas y gestos

La orquestación de la interactividad se unifica en el cuerpo (`<body>`) de la escena mediante una única entidad. El componente `hands-spheres` asume el papel de administrador central. Al recibir las propiedades de configuración lógicas `enablePinch:true` y `enablePoint:true`, el módulo intercepta el ciclo de vida del elemento e inyecta dinámicamente los componentes `pinch-gesture` y `point-gesture` en el árbol del DOM.

Esta solución reduce las líneas de marcado HTML necesarias. Resuelve en una sola declaración tanto la representación visual de las falanges mediante esferas como la activación de los volúmenes invisibles de impacto espacial (Figura 3.3).

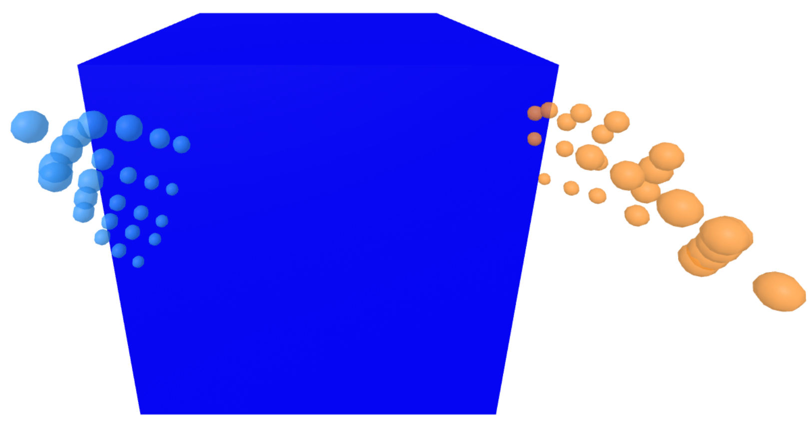


Figura 3.3: Vista en primera persona del entorno interactivo mínimo utilizando el script unificado de manipulación directa.

3.2.4. Composición de entidades interactuables

El paso definitivo consiste en dotar de comportamiento a los objetos inanimados de la escena. Gracias a la estructura modular del sistema, el desarrollador simplemente debe enumerar las mecánicas deseadas como atributos dentro de la etiqueta HTML de la forma geométrica.

En el ejemplo expuesto, se crea un cubo básico (`<a-box>`) al que se le inyecta la cadena completa de interacción: `hoverable`, `grabbable`, `stretchable`, `draggable` y `droppable`. Al cargar la página, el sistema lee estos atributos y aplica la lógica interna de la biblioteca automáticamente. Como resultado, el cubo se iluminará ante la proximidad física, podrá ser trasladado, modificará su volumen al aplicar un gesto bimanual y detectará correctamente si es depositado sobre una zona de recepción compatible, consolidando un flujo de trabajo complejo sin exigir desarrollo de código adicional.

3.3. Descripción de implementación

Para comprender la viabilidad y el funcionamiento interno de la biblioteca, es necesario desglosar el flujo algorítmico a nivel de código. La arquitectura se fundamenta en la ejecución continua dentro del bucle de renderizado de A-Frame (la función `tick()`) y en la gestión

asíncrona de eventos del DOM. A continuación, se detallan las funciones y métodos de JavaScript que orquestan este comportamiento.

3.3.1. Interfaces transversales y gestión de colisiones

El sistema implementa métodos compartidos para reducir la duplicidad de código y garantizar la compatibilidad geométrica entre los distintos componentes.

- **Método `getHandCollider(handedness)`:** Esta función reside en los detectores gestuales y actúa como una interfaz unificada (*wrapper*). Su propósito es aislar a los componentes interactuables de la complejidad del motor de colisiones. La función evalúa la propiedad `colliderType` configurada. Si se utiliza el motor OBB nativo, extrae el componente `obb-collider`. Si se utiliza la solución propia, extrae `sat-collider`. En ambos casos, devuelve un objeto estándar que expone siempre el mismo método `testCollision(otherOBB)`. Esto permite que cualquier objeto evalúe intersecciones mediante una única llamada lógica.
- **Función `_ensureCollider()`:** Es un método de seguridad transversal presente en la inicialización (`init()`) de los componentes interactuables (como `hoverable` o `grabbable`). El código comprueba si la entidad (`this.el`) posee ya un atributo geométrico de colisión. Si carece de él, la función invoca `this.el.setAttribute()` para inyectar dinámicamente un colisionador invisible con las dimensiones paramétricas proporcionadas en el esquema, garantizando que el objeto pueda registrar contactos físicos.

3.3.2. Lógica de seguimiento y detección gestual

Los componentes vinculados a la anatomía del usuario (`pinch-gesture` y `point-gesture`) estructuran su ejecución en tres fases secuenciales dentro de su método principal `tick()`.

1. **Lectura espacial (`tick`):** El sistema accede a las entradas de hardware consultando `this.el.sceneEl.renderer.xr.getSession().inputSources`. El código itera sobre las manos detectadas. Utiliza la función nativa `getJointPose()` para extraer la matriz espacial de cada articulación y la almacena en estructuras de datos de Three.js, concretamente en objetos `THREE.Vector3`.

2. **Cálculo de posicionamiento (`updateColliderPosition`):** Esta función traslada el colisionador invisible de la mano a las coordenadas correctas. En el componente de pellizco, calcula el punto medio exacto entre la muñeca (`wrist`) y el nudillo central (`middle-finger-metacarpal`) para situar el volumen en el centro de la palma. En el componente de apuntado, extrae directamente las coordenadas de la articulación `index-finger-tip` y aplica el método `object3D.position.copy()` para enclavar el colisionador en la yema del dedo.

3. **Análisis de postura (`detectGesture`):** Es la función encargada de la lógica de si se determina si se esta haciendo un gesto o noo. Utiliza el método `distanceTo()` de Three.js para medir la separación euclidiana entre falanges. Por ejemplo, evalúa la distancia entre la punta del índice y el pulgar. Si el valor numérico desciende por debajo de la propiedad `startDistance`, actualiza la máquina de estados interna. A continuación, invoca `this.el.emit()` para despachar eventos estándar (`pinchstart`, `pointstart`) hacia el árbol del DOM, adjuntando la información de la extremidad responsable en el objeto de detalles del evento.

3.3.3. Implementación de componentes interactuables

Los módulos pasivos se asignan a los objetos de la escena. Estos componentes configuran escuchadores de eventos (`addEventListener`) en su fase de inicialización. A continuación, se detalla la implementación lógica de aquellos módulos que aplican transformaciones geométricas complejas sobre la malla (`grabbable`, `stretchable` y `droppable`), así como la arquitectura compartida por el resto de componentes centrados en el control de estado.

Gestión de agarre (`grabbable`)

El flujo de este componente se basa en la alteración de la jerarquía espacial y la aplicación de álgebra lineal.

- **Función `start(evt)`:** Se dispara al escuchar el evento de inicio de gesto. Primero, comprueba la variable `validContactForGrab` para verificar que la colisión ocurrió antes del gesto. Si la validación es correcta, el código guarda las coordenadas globales de la figura.

- **Funciones `_lockPosition()` y `_unlockPosition()`:** Ejecutan la matemática de emparentado. Al agarrar, el objeto se añade como nodo hijo (`parent.add()`) de la entidad de la mano. Para evitar que el objeto sufra un salto visual hacia el origen de coordenadas de su nuevo contenedor, el sistema calcula la matriz inversa de la extremidad (`matrixWorld.invert()`) utilizando `THREE.Matrix4`. Esta matriz se multiplica por la posición absoluta de la pieza, recalculando sus coordenadas locales para que mantenga la distancia relativa original respecto a la mano del usuario.

Escalado bimanual (`stretchable`)

Este módulo manipula el vector de escala en tiempo real resolviendo dependencias de forma autónoma.

- **Función `_ensureGrabbable()`:** Se ejecuta en el ciclo de vida inicial (`init()`). Evalúa si la entidad puede ser agarrada. Si no detecta el componente `grabbable`, lo inyecta mediante `setAttribute()` para evitar errores en tiempo de ejecución.
- **Cálculo en `tick()`:** Al registrar el contacto válido con ambas manos, el sistema captura la separación inicial (`initialDistance`). En cada frame, obtiene la separación actual (`currentDistance`) y extrae un multiplicador. Seguidamente, utiliza `multiplyScalar()` sobre el vector de escala original (`initialScale`) para redimensionar la malla geométrica.
- **Función `_updateColliderSize()`:** Se llama concurrentemente durante el escalado. Recalcula los límites del prisma delimitador (`OBB`) para que la caja física invisible crezca o encoja en la misma proporción matemática que el objeto visual.

Filtrado en zonas objetivo (`droppable`)

La lógica de este componente prioriza el rendimiento asíncrono y la prevención de fugas de memoria computacional.

- **Manejo del DOM (`MutationObserver`):** Para evitar buscar elementos constantemente con `querySelectorAll`, el componente `droppable` instancia un observador nativo. Este objeto vigila los cambios estructurales de la página. Solo cuando se añaden o

eliminan nodos HTML, el componente reevalúa qué figuras cumplen con el selector definido en la propiedad `accepts` (mediante el método `matches()`) y actualiza su lista interna (caché) de objetos válidos.

- **Función `_cleanupListener()`:** Es crítica para la estabilidad. Al registrar el contacto de un objeto arrastrado, el sistema inyecta una estructura `Map` directamente en dicha figura (`_droppableListeners`). Esta estructura asocia de manera unívoca el identificador de la zona objetivo con la función que escucha el evento `drag-end`. Antes de registrar una nueva colisión, el método consulta el mapa y elimina estrictamente (`removeEventListener()`) cualquier función residual anterior. Esto garantiza que la lógica de soltado se ejecute una única vez por acción.

Control de estados semánticos (`hoverable`, `draggable` y `clickable`)

A diferencia de los módulos anteriores, el resto de componentes interactuables no aplican transformaciones matriciales ni alteran visualmente la malla 3D. Se agrupan bajo una arquitectura compartida basada puramente en el control de flujo y la gestión del Modelo de Objetos del Documento.

- **Limitación de acciones:** Componentes como `clickable` y `hoverable` mantienen variables de tipo arreglo en su código (como `this.clickers` o `this.hoverers`). Al dispararse el evento de colisión, la función correspondiente extrae la identidad de la mano desde el objeto `evt.detail` y la almacena. Esto permite rastrear cuántas extremidades exactas interactúan con la pieza frame a frame, limitando las acciones simultáneas mediante propiedades como `maxClickers`.

- **Gestión de estado (`addState` / `removeState`):** En lugar de mover objetos, el código de estos componentes invoca los métodos nativos de A-Frame para inyectar estados lógicos. Tras validar la intención del usuario, ejecutan `this.el.addState()` para añadir atributos semánticos (como `hovered`, `dragged` o `clicked`), facilitando que componentes externos reaccionen mediante cambios de color o sonido.

- **Propagación de dependencias temporales:** En el caso específico de `draggable`, su función de código principal es actuar como un mensajero. Al detectar un arrastre, la

función `start()` emite un evento personalizado empaquetando su propia referencia (`this.el`) dentro de la propiedad `dropped`. Este mecanismo permite al componente `droppable` capturar la entidad de forma programática y validar sus selectores sin necesidad de recalcular intersecciones espaciales pesadas.

3.4. Comparación con super-hands

El objetivo prioritario de este Trabajo de Fin de Grado consistió, desde su planteamiento inicial, en desarrollar una alternativa interactiva de seguimiento óptico de manos libres (*hand-tracking*) que fuera plenamente compatible con la filosofía de la biblioteca *Super-Hands*. La meta explícita no era crear un paradigma de interacción aislado, sino reescribir y trasladar la modularidad semántica original de dicho ecosistema (basado históricamente en mandos físicos y botones) hacia el análisis anatómico de gestos naturales. Por este motivo, el éxito del proyecto se validó mediante un ejercicio de adaptación directa. Se tomó una de las escenas de demostración oficiales del repositorio original de *Super-Hands* y se reconstruyó utilizando de forma exclusiva el código unificado desarrollado en este proyecto.

3.4.1. Análisis de la arquitectura original basada en controladores

En la demostración original de *Super-Hands*, la interacción dependía por completo de la pulsación de botones analógicos y mecánicos, así como del lanzamiento de rayos virtuales (*ray-casters*) proyectados desde las manos del avatar. Para ofrecer soporte a las distintas plataformas del mercado, el desarrollador se veía obligado a importar múltiples componentes específicos de hardware.

Como se observa en el Código 3.2, el diseño original requería declarar un conjunto complejo de *mixins* (plantillas reutilizables en A-Frame) para cubrir la variedad de gafas comerciales (HTC Vive, Oculus Touch o Windows Mixed Reality). Estos bloques vinculaban los eventos de contacto geométrico de los rayos con las pulsaciones de los gatillos analógicos.

Código 3.2: Fragmento original de *Super-Hands* declarando la compatibilidad con múltiples controladores físicos.

```
<a-mixin id="pointer" raycaster="showLine: true; objects: .cube, a-link"
```

```
         super-hands="colliderEvent: raycaster-intersection;
                      colliderEventProperty: els;">
</a-mixin>

<a-mixin id="controller-right" mixin="pointer"
         vive-controls="hand: right"
         oculus-touch-controls="hand: right"
         windows-motion-controls="hand: right"
         gearvr-controls daydream-controls oculus-go-controls>
</a-mixin>

<a-entity id="rhand" mixin="controller-right"></a-entity>
```

3.4.2. Adaptación mediante la arquitectura de gestos integrados

En la escena adaptada, se eliminó toda la sobrecarga de código vinculada a los controladores de hardware y la lógica de la biblioteca *Super-Hands* original. Al ser el objetivo central del TFG una migración transparente hacia mallas naturales, el bloque engorroso de plantillas se sustituyó por una arquitectura unificada que opera de forma agnóstica respecto al fabricante de las gafas.

Como muestra el Código 3.3, el sistema pasó a requerir únicamente la habilitación del parámetro de seguimiento de manos en la sesión WebXR y la instanciación de una única entidad interactiva. Gracias al diseño actual del componente `hands-spheres`, las propiedades `enablePinch:true` y `enablePoint:true` resuelven en un solo paso la representación, tal que, añaden los detectores gestuales.

Código 3.3: Fragmento adaptado utilizando la infraestructura integrada de mallas y gestos naturales.

```
<a-scene  webxr="optionalFeatures: hand-tracking">

    <a-entity hands-spheres="enablePinch:true; enablePoint:true;
       gestureColliderType:obb-collider;">
    </a-entity>

</a-scene>
```

3.4.3. Validación de la interfaz de componentes interactuables

El cumplimiento de los objetivos del proyecto se confirma de manera concluyente al analizar las entidades reactivas (los cubos interactivos). El requisito de diseño principal exigía que la transición tecnológica no obligara al desarrollador a modificar su flujo de trabajo declarativo ni a reestructurar la composición de los modelos tridimensionales para dar soporte a las manos reales.

En la escena original de *Super-Hands*, un cubo interactivo se declaraba en el marcado de la siguiente manera:

```
<a-entity class="cube" geometry="primitive: box;"
          hoverable grabbable stretchable draggable droppable>
</a-entity>
```

En la demostración adaptada para este trabajo, la declaración de la entidad se mantuvo idéntica. Los componentes interactuables desarrollados (`hoverable`, `grabbable`, `stretchable`, `draggable`, `droppable`) demostraron actuar como reemplazos perfectos directos (*drop-in replacements*). La única alteración estructural necesaria en el entorno fue la sustitución de la librería externa `aframe-event-set-component` por un breve bloque de JavaScript nativo en la página. Este bloque simplemente se suscribe de manera reactiva a los eventos estandarizados que emite de forma asíncrona la nueva biblioteca (como `hover-start` o `drag-drop`) para gestionar los cambios estéticos de color.

En conclusión, el funcionamiento fluido de esta demostración confirmó que la arquitectura planteada resuelve el problema inicial de forma transparente. Se consiguió dotar al ecosistema web de una caja de herramientas que preserva la modularidad semántica clásica que el desarrollador ya conoce, pero que eleva la naturalidad de la experiencia inmersiva al sustituir la abstracción de los botones físicos por movimientos anatómicos directos del usuario.

3.5. Ejemplos de demostraciones

El resultado central de este Trabajo de Fin de Grado es una biblioteca de componentes “caja de herramientas” orientada a entornos inmersivos web. Desde la perspectiva del usuario final,

el sistema proporciona la capacidad de manipular elementos tridimensionales utilizando exclusivamente las manos, eliminando por completo la necesidad de utilizar controladores físicos.

Al acceder al entorno virtual mediante un visor compatible, las cámaras integradas en el dispositivo escanean la anatomía de las manos y trasladan ese esqueleto en tiempo real a la escena gráfica. A partir de ese momento, el usuario puede interactuar con el entorno mediante gestos naturales e intuitivos: pellizcar en el aire para agarrar una figura, separar ambas manos para agrandar un objeto, o extender el dedo índice para teclear botones físicos.

Para demostrar la viabilidad y la usabilidad de estas mecánicas, se ha desarrollado una colección de escenas web. Dado que el proyecto depende directamente del estándar de hardware de la API WebXR, la interacción completa solo puede experimentarse desde el navegador interno de unas gafas de realidad virtual o mixta (como la familia Meta Quest). Si un usuario carga estas páginas desde el navegador de un ordenador tradicional, visualizará el entorno 3D estático y podrá girar la cámara con el ratón, pero carecerá de las herramientas de interacción manual.

3.5.1. Escenas de prueba individuales

Antes de construir escenarios complejos, se diseñó un conjunto de escenas aisladas (tales como `clickable.html`, `grabbable.html` o `drag-drop.html`). El propósito de estos entornos es demostrar de forma atómica la usabilidad de cada componente por separado.

En cada una de estas escenas, el usuario se posiciona frente a una disposición sencilla de primitivas geométricas, como cubos o paneles flotantes, configuradas para reaccionar a una única mecánica específica. Por ejemplo, en el entorno de prueba de interacción directa, el usuario debe extender su dedo índice para pulsar físicamente un panel tridimensional que cambia de color al impacto. Estas escenas actúan como un manual interactivo visual, permitiendo comprender rápidamente qué postura anatómica desencadena cada acción sin elementos que supongan una distracción (Figura 3.4).

3.5.2. Demostración integradora: Adaptación de Super-Hands

Para comprobar la cohesión y el rendimiento del sistema en un caso de uso práctico, se desarrolló el entorno `superhands-demo.html`. Esta demostración consiste en una adaptación

Figura 3.4: Ejemplo de entorno de prueba aislado para validar la usabilidad de un único componente de interacción.

directa y fiel de una de las escenas de ejemplo clásicas pertenecientes a la biblioteca original de código abierto *Super-Hands*.

El entorno original de dicha biblioteca fue programado hace años asumiendo el uso estricto de los gatillos de los mandos de realidad virtual. En este trabajo, el archivo HTML de la escena ha sido reconstruido y adaptado para funcionar de manera idéntica utilizando los colisionadores espaciales y el seguimiento óptico de manos libres desarrollados a lo largo del proyecto.

El usuario se encuentra inmerso frente a una mesa virtual que sostiene varias figuras geométricas. Puede acercar su mano anatómica para ver cómo las piezas se iluminan, pellizcarlas para levantarlas, apilarlas unas sobre otras o agarrarlas con dos manos simultáneamente para estirarlas. Cabe destacar que el comportamiento sólido de esta escena no se apoya en ningún motor físico de cuerpos rígidos. Toda la manipulación, el anclaje y la detección de límites se realizan a través de la evaluación de volúmenes puramente geométrica (OBB/SAT), lo que garantiza una respuesta altamente predecible y evita que los objetos salgan despedidos ante los inevitables temblores de los sensores ópticos (Figura 3.5).

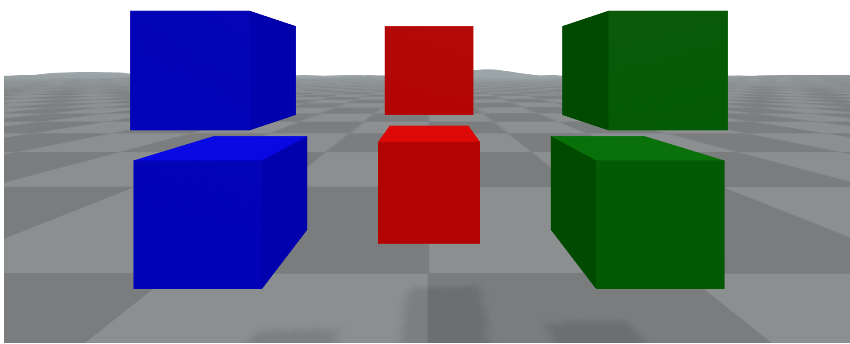


Figura 3.5: Escena integradora reconstruida para ser operada exclusivamente mediante seguimiento espacial de manos libres.

3.5.3. Demostración en Realidad Aumentada (AR)

Otra de las evaluaciones fundamentales del sistema se materializó en el entorno de pruebas de Realidad Aumentada (fichero `ar-demo.html`). En lugar de limitar al usuario dentro de una simulación gráfica opaca, esta escena aprovecha las capacidades (*passthrough*) de WebXR para proyectar la señal de vídeo de las cámaras exteriores directamente en las pantallas del visor.

La experiencia funcional permite al usuario posicionar, escalar e interactuar con objetos tridimensionales flotantes superpuestos sobre su habitación real. El uso de la realidad aumentada añade un desafío adicional a nivel de usabilidad, puesto que la persona debe inferir visualmente la profundidad de las piezas digitales y contrastarlas con la iluminación del entorno físico que le rodea. La combinación fluida de los gestos de agarre y soltado en este contexto valida que la biblioteca de componentes generada es tecnológicamente agnóstica, resultando plenamente operativa tanto en realidad virtual estricta como en escenarios de realidad mixta.

3.5.4. Demostración final orientada a tareas: Taller de Ensamblaje

La última prueba de evaluación del sistema consistió en la creación de un entorno diseñado para encadenar de forma secuencial todas las mecánicas desarrolladas. Bajo el concepto visual de un ”Taller de Ensamblaje”(`complete-assembly.html`), la experiencia propone la resolución de un puzle tridimensional. El objetivo es validar la usabilidad y estabilidad del sistema

cuando el usuario interactúa con múltiples componentes de forma concurrente, asegurando que no se produzcan cruces de eventos ni falsos positivos.

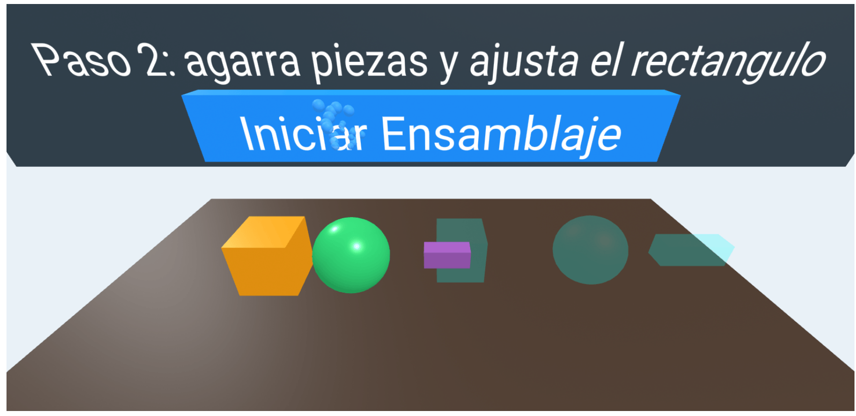


Figura 3.6: Vista del usuario frente a la mesa de trabajo en la demostración orientada a tareas, mostrando el panel de control y las áreas de recepción.

El flujo de uso obliga al usuario a utilizar distintas configuraciones anatómicas según la tarea:

1. **Iniciación (*clickable*):** El escenario presenta un panel de control flotante. Para iniciar la prueba, el usuario debe extender físicamente el dedo índice y pulsar el botón tridimensional. Al detectar la colisión del dedo, el panel actualiza las instrucciones y revela los objetos de la mesa.

2. **Aproximación e identificación (*hoverable*):** Sobre la superficie de trabajo aparecen tres figuras geométricas y tres moldes traslúcidos. Para localizar rápidamente las piezas interactivas, el usuario acerca la mano a las figuras. El sistema proporciona retroalimentación iluminando la malla del objeto al detectar proximidad.

3. **Traslado (*grabbable*):** El usuario debe ejecutar el gesto de pellizco para agarrar cada pieza y trasladarla por el espacio tridimensional de forma estable.

4. **Ajuste paramétrico (*stretchable*):** Una de las figuras geométricas (el bloque rectangular) se genera con un volumen intencionadamente desproporcionado. El usuario debe sujetarlo con ambas manos simultáneamente y alejar los brazos para aumentar su escala.

5. **Validación espacial (*draggable* y `droppable`):** Finalmente, las figuras deben depositarse en sus respectivos moldes. El código evalúa la caída en dos niveles. Primero, utiliza

el sistema de selectores CSS para rechazar emparejamientos erróneos (como intentar soltar la esfera en el hueco del cubo). Segundo, mediante lógica JavaScript apoyada en la API de Three.js, calcula el tamaño absoluto de la pieza en el instante del impacto; si el bloque rectangular no ha sido reducido con el gesto bimanual hasta alcanzar una tolerancia del 25 % respecto al tamaño del molde, la pieza es rechazada.

Al completar con éxito la inserción de las tres figuras, el sistema bloquea sus posiciones de forma permanente y actualiza el panel de inicio, confirmando la resolución del escenario. Esta demostración confirma la viabilidad de la arquitectura propuesta para construir simuladores mecánicos, educativos o de formación industrial plenamente funcionales desde el navegador del visor.

Capítulo 4

Desarrollo del proyecto

Cuadro 4.1: Resumen de las iteraciones de desarrollo, objetivos principales y prototipos resultantes.

Iteración	**Foco principal**	**Entregable funcional**
Sprint 0	Aprendizaje del *framework*	Escenas base en A-Frame sin interacción
Sprint 1	Seguimiento de manos (*tracking*)	Visualización de articulaciones y colisionador base
Sprint 2	Gestos base y agarre (*grab*)	Componente de pellizco y acoplamiento de entidades
Sprint 3	Dualidad de colisiones	Sistema compatible con SAT y OBB nativo
Sprint 4	Escalado bimanual (*stretch*)	Modificación de escala mediante dos manos
Sprint 5	Ciclo *hover, drag & drop*	Arrastre espacial y suelta en zonas objetivo
Sprint 6	Activación directa (*click*)	Pulsación física de botones con el dedo índice
Sprint 7	Integración y pulido XR	Demo final en AR (Fase actual en desarrollo)

4.1. Ejecución de los Sprints

A continuación, se detalla el trabajo técnico desarrollado a lo largo de las ocho iteraciones del proyecto, exponiendo la evolución de la arquitectura desde las pruebas de concepto iniciales hasta el componente de interacción final.

4.2. Sprint 0: Fundamentos del entorno y reducción de riesgos

Iniciar un desarrollo inmersivo desde cero conlleva un alto nivel de incertidumbre técnica. Especialmente cuando no se tiene experiencia previa manejando herramientas de bajo nivel para la web tridimensional. Por este motivo, la planificación inicial dejó a un lado el seguimiento de manos y las matemáticas complejas para centrarse en conocer la plataforma de trabajo. A continuación se detallan las fases de este primer ciclo de desarrollo.

Especificación de objetivos

El objetivo principal de este primer *sprint* fue la pura reducción del riesgo tecnológico. Antes de abordar el álgebra espacial o la programación de colisionadores, era imprescindible asimilar los fundamentos operativos de A-Frame y Three.js.

Se definieron tres metas concretas a alcanzar. La primera fue entender cómo instanciar una escena 3D y estructurar el código utilizando el patrón ECS (*Entity-Component-System*). La segunda consistió en controlar el ciclo continuo de renderizado. Este ciclo, que en A-Frame se gestiona a través de la función `tick`, es el responsable de ejecutar la lógica en cada frame para asegurar un rendimiento fluido en las gafas. Por último, se planteó el objetivo de establecer un flujo de despliegue ágil para probar el código sin fricciones.

Tareas realizadas

Durante estas primeras jornadas, el trabajo se basó en programar ejercicios básicos de prueba y error. Se construyeron escenas elementales escritas en HTML. Estas escenas estaban formadas por primitivas geométricas simples, como cajas, esferas y planos orientados a modo de suelo. A partir de esta base estática, se inyectaron *scripts* en JavaScript para alterar las mallas

dinámicamente. Se probó a cambiar colores en tiempo de ejecución, modificar posiciones y analizar la respuesta del motor gráfico.

Otra tarea fundamental fue configurar el entorno de pruebas. Para depurar el código de forma rápida desde el ordenador, se utilizó la extensión WebXR API Emulator en el navegador Google Chrome. Esta herramienta permite simular los movimientos del visor y de los controladores dentro de la pestaña del navegador, evitando tener que ponerse y quitarse las gafas por cada línea de código modificada.

Para realizar las pruebas finales con el hardware real, el proyecto se desplegó directamente mediante GitHub Pages. De esta forma, el flujo consistía en subir los cambios al repositorio y acceder a la URL pública desde el navegador del visor. Mediante este proceso se verificó que las llamadas a la API de WebXR funcionaban correctamente y que los botones nativos para entrar en Realidad Virtual (VR) y Realidad Aumentada (*passthrough*) no generaban problemas de permisos en la consola.

Prototipo resultante

El entregable de esta fase temprana no fue un componente funcional para la biblioteca definitiva. En su lugar, se obtuvo una colección de archivos de prueba aislados a modo de *sandbox*. Aunque estas escenas carecían de interactividad real, sirvieron como plantillas limpias, estables y cien por cien reproducibles para los siguientes ciclos.

Este conjunto de archivos de prueba permitió validar empíricamente el sistema de coordenadas espaciales. Se comprobó la correspondencia exacta de escala entre el motor 3D y el mundo físico. Al definir un cubo con un ancho de “1” en el código y visualizarlo en modo de Realidad Aumentada, se comprobó que ocupaba exactamente un metro en la habitación del usuario. Confirmar esta relación de escala uno a uno era un paso previo obligatorio antes de programar mecánicas basadas en distancias cortas entre los dedos.

Lecciones aprendidas y problemas resueltos

La lección técnica más determinante de este *sprint* estuvo relacionada con la extrema sensibilidad de las jerarquías espaciales. En los motores gráficos basados en grafos de escena, como es el caso de Three.js, la posición de un objeto depende completamente de su nodo padre. Cuando se anida un elemento dentro de otro en el árbol del DOM, las coordenadas del hijo dejan de

ser globales. Pasan a ser un valor relativo a la posición, rotación y escala del elemento padre.

Durante las primeras pruebas en el código, se intentó mover un cubo de un sitio a otro modificando su jerarquía HTML mediante JavaScript. El resultado inmediato fue que el objeto desaparecía del campo de visión o salía despedido a coordenadas erróneas. Tropezar con este comportamiento tan pronto supuso una gran ventaja para el proyecto.

Este error obligó a estudiar en profundidad cómo operan las matrices de transformación global y local (específicamente la propiedad `matrixWorld` de Three.js) antes de diseñar la interacción manual. Comprender que cambiar el"padre" de un objeto altera drásticamente su sistema de referencia evitó arrastrar fallos de diseño graves. Esta lección teórica dejó el terreno preparado para los siguientes *sprints*, donde sería totalmente necesario cambiar las jerarquías de las figuras para anclarlas a las manos del usuario durante los agarres, requiriendo el uso de matrices inversas para evitar saltos gráficos indeseados.

4.3. Sprint 1: Núcleo de seguimiento y colisionadores base

Una vez consolidado el entorno de pruebas, el desarrollo se centró en la integración del usuario dentro de la escena. El foco principal de este ciclo fue el seguimiento de manos (*hand-tracking*) y la construcción de un sistema de colisiones propio.

Especificación de objetivos

El reto técnico de este *sprint* consistió en capturar el flujo de datos proveniente de las manos del usuario a través de la API de WebXR. Se estableció como meta leer estas coordenadas espaciales y transformarlas en dos elementos distintos: representaciones visuales y volúmenes físicos.

El primer objetivo fue hacer visibles las manos en la pantalla. El segundo objetivo, más complejo, fue conseguir que el motor gráfico interpretara esos puntos anatómicos como objetos sólidos capaces de chocar contra el entorno, sentando las bases del algoritmo de detección geométrica SAT (*Separating Axis Theorem*).

Tareas realizadas

El trabajo comenzó por el análisis y la extracción de datos espaciales. Dentro del bucle de renderizado (*tick*), se programó el acceso a la sesión activa de WebXR para leer el arreglo `inputSources`. A partir de ahí, se iteró sobre cada una de las 25 articulaciones detectadas por el visor. Para obtener las coordenadas exactas frente a la cámara, se utilizó la función nativa `getJointPose`.

Para visualizar esta información, se desarrolló un componente propio denominado `hands-spheres`. Este módulo genera una esfera tridimensional por cada articulación y actualiza su posición en cada frame basándose en la matriz de transformación obtenida de WebXR. El Código 4.1 muestra la lógica central utilizada para extraer la posición y aplicarla al objeto visual.

Código 4.1: Lectura de la matriz espacial mediante la API de WebXR y actualización posicional de la esfera geométrica que representa la articulación detectada.

```
const pose = frame.getJointPose(joint, this.referenceSpace);

if (pose) {
    const p = pose.transform.position;

    entry.sphere.object3D.position.set(p.x, p.y, p.z);
    entry.sphere.object3D.visible = true;
}
```

Una vez resuelta la parte visual, se abordó la estructura física. Se programó la primera versión del componente `sat-collider`. Este módulo envuelve las entidades en cajas orientadas (OBB) y calcula su intersección proyectando los vértices sobre 15 ejes espaciales. Posteriormente, se acoplaron estos colisionadores invisibles directamente a las coordenadas de las falanges de la mano.

Prototipo resultante

La entrega funcional de esta fase fue una escena donde el usuario podía visualizar sus manos de forma fluida. La representación gráfica constaba de un conjunto de esferas translúcidas que seguían el movimiento de los dedos con alta precisión (Figura 4.1).

Además del apartado visual, el prototipo incluía un mecanismo activo de físicas. Se introdujo un cubo de prueba en la escena equipado con el nuevo `sat-collider`. Al acercar la mano, el sistema era capaz de calcular la distancia matemática y registrar el contacto físico rudimentario entre la yema del dedo y la superficie del cubo, emitiendo un aviso de colisión por la consola del navegador.

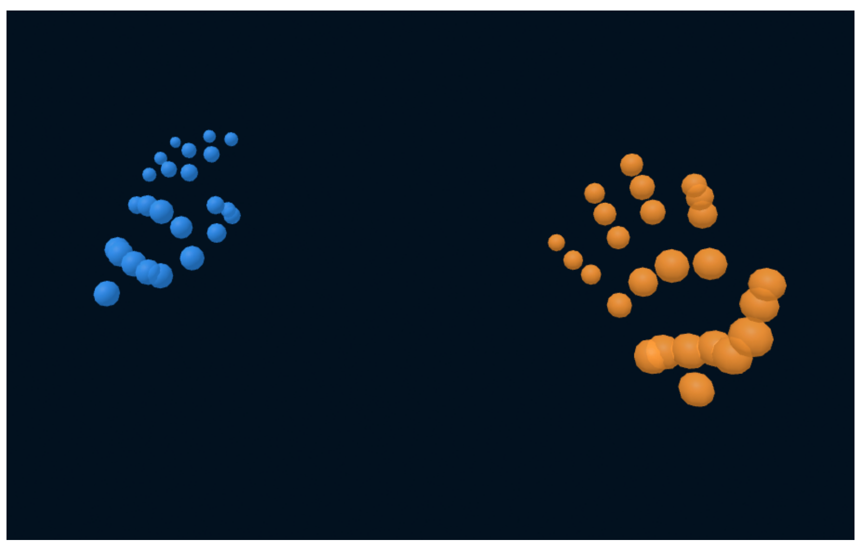


Figura 4.1: Representación visual del seguimiento de manos en WebXR utilizando el componente `hands-spheres`.

Lecciones aprendidas y problemas resueltos

Durante la implementación visual de las manos, se detectó una caída importante de rendimiento (*frame drops*) en los primeros intentos. El error se debía a que el código creaba y destruía entidades 3D en cada frame. La solución implementada consistió en utilizar un diccionario estático de elementos; las esferas se instancian una única vez al detectar la mano y simplemente se ocultan (`visible = false`) cuando el visor pierde el seguimiento, optimizando el consumo de memoria.

Por otro lado, al realizar las pruebas de impacto contra el cubo, se observó un alto volumen de detecciones fantasma. El sistema notificaba colisiones falsas antes de que el usuario tocara visualmente el objeto. Analizando el comportamiento en el entorno de pruebas, se determinó que el problema radicaba en una altísima sensibilidad al tamaño del volumen delimitador.

Este fallo geométrico se mitigó mediante dos ajustes. Primero, se calibraron las dimensiones

milimétricas de las cajas de colisión para que se ajustaran estrictamente a la malla visible. Segundo, se programó el componente para que el radio del colisionador de la mano se adaptara dinámicamente (`pose.radius`) utilizando los datos de grosor que estimaba el visor para cada usuario, logrando una intersección mucho más fiel a la realidad.

4.4. Sprint 2: Gestos principales y componente de agarre (*grab*)

Tras conseguir una representación visual estable de las manos, el desarrollo avanzó hacia la creación de mecánicas de interacción. El objetivo de este ciclo fue transformar las colisiones geométricas en acciones lógicas. Se buscó emular el comportamiento modular de bibliotecas de referencia como *Super-Hands*, pero eliminando por completo la dependencia de mandos físicos o botones binarios.

Especificación de objetivos

El propósito fundamental de este *sprint* fue habilitar el agarre de objetos mediante gestos naturales. Se priorizó el gesto de pellizco (*pinch*) como el disparador principal de la acción. Este proceso requería dos metas técnicas claras: primero, definir un algoritmo capaz de identificar cuándo el usuario cierra los dedos índice y pulgar; segundo, desarrollar un componente que cuando detectase el gesto y contacto de la mano, el objeto siguiese sus coordenadas. La arquitectura debía ser modular, permitiendo que cualquier entidad de la escena se volviera interactiva simplemente añadiéndole un atributo.

Tareas realizadas

El trabajo comenzó con el desarrollo del componente `pich-gesture`. Este módulo analiza en cada frame la distancia euclidiana entre las articulaciones `thumb-tip` (punta del pulgar) y `index-finger-tip` (punta del índice). Al descender por debajo de un umbral de 2,5 centímetros, el sistema dispara el evento `pinchstart`.

Posteriormente, se programó el componente `grabbable` para gestionar la respuesta a dicho evento. La tarea más crítica fue la implementación del re-emparentado (*reparenting*). Al modificar la jerarquía de una entidad en el motor gráfico, sus coordenadas dejan de ser globales

y pasan a depender de su nuevo padre. Si este cambio se realiza sin correcciones, el objeto sufre un salto brusco hacia el centro de la mano al ser agarrado.

Para evitar este defecto visual, se recurrió al álgebra lineal. El código registraba las coordenadas y la rotación absoluta de la pieza en la escena justo en el milisegundo previo al contacto. Una vez que el objeto se anclaba a los dedos, se aplicaba la matriz inversa del colisionador de la mano para recalcular su nueva posición local. Gracias a esta operación matemática, la pieza mantenía su separación y orientación exacta respecto al usuario, sin sufrir tirones en la pantalla durante el traslado.

Prototipo resultante

La entrega funcional de este ciclo fue un sistema de interacción directo. El usuario ya no solo visualizaba sus manos, sino que podía interactuar con el entorno físico virtual. El prototipo permitió recoger cubos de prueba, trasladarlos libremente por el espacio tridimensional y soltarlos en nuevas coordenadas al abrir la mano. Este hito validó la eficacia del componente `grabbable` y la precisión del detector de gestos, demostrando que la manipulación de entidades era viable sin necesidad de controladores mecánicos.

Lecciones aprendidas y problemas resueltos

Durante las sesiones de prueba con el visor, surgió un fallo de usabilidad recurrente. Un usuario podía iniciar el gesto de pellizco en el aire y, mientras mantenía los dedos cerrados, tocar un objeto. Esto provocaba que el objeto se "pegara" a la mano de forma errática, lo cual resultaba poco intuitivo.

Para resolver este problema, se refinó la máquina de estados del componente `grabbable`. Se introdujo una variable de control denominada `validContactForGrab`. La lógica se modificó para que el sistema solo permita el inicio de un agarre si se cumplen dos condiciones en un orden estricto: primero, la mano debe estar en contacto físico con el objeto; segundo, se debe iniciar el gesto de pellizco. Si el usuario ya está pellizcando antes de tocar la pieza, el contacto se ignora. Esta restricción técnica eliminó las activaciones accidentales y mejoró significativamente la sensación de control sobre las piezas de la escena.

4.5. Sprint 3: Dualidad de colisiones y arquitectura unificada

Durante las primeras pruebas de agarre, la implementación del colisionador nativo de A-Frame (`obb-collider`) arrojó errores de detección en los giros de muñeca. Bajo la premisa de que se trataba de una limitación interna del motor, se había desarrollado el componente `sat-collider` personalizado. Sin embargo, tras una revisión técnica más profunda, se comprobó que el colisionador nativo funcionaba correctamente si se configuraba de forma adecuada. Esto motivó la reestructuración de la base del proyecto.

Especificación de objetivos

El objetivo central de este ciclo de desarrollo fue refactorizar la capa de detección para admitir ambas aproximaciones matemáticas (el SAT propio y el OBB nativo). La meta era construir una arquitectura dual donde los componentes de interacción pudieran operar de forma transparente con cualquiera de los dos colisionadores. Se buscaba que el desarrollador final pudiera cambiar entre un sistema u otro simplemente modificando un parámetro en la etiqueta HTML del detector de gestos, sin que el flujo de interacción del usuario se rompiera.

Tareas realizadas

El trabajo se centró en la modificación profunda de los detectores de gestos y los componentes reactivos. Se añadió la propiedad `colliderType` al esquema de datos de componentes como `pich-gesture`. A partir de ese momento, la capa de reacción (como el componente `grabbable`) se reprogramó para buscar al detector en la escena, leer qué tipo de colisionador estaba utilizando y heredar esa configuración automáticamente.

Para lograr esta compatibilidad sin duplicar la lógica de interacción, se programó una función adaptadora unificada dentro del detector de gestos denominada `getHandCollider()`. Este método actúa como una interfaz común. Cuando un componente necesita saber si la mano está tocando un objeto, llama a esta función. Si el sistema está usando el SAT personalizado, devuelve el objeto matemático directamente. Si está usando el OBB nativo, la función envuelve los datos del motor en un formato idéntico al del SAT, proporcionando un método estándar `testCollision()` unificado.

Código 4.2: Patrón adaptador que evalúa el tipo de colisionador activo y devuelve un objeto estandarizado con una interfaz unificada para calcular intersecciones tridimensionales.

```
getHandCollider: function (handedness) {
    const handState = this.state[handedness];

    if (this.data.colliderType === 'obb-collider') {
        const obbComp = handState.colliderEntity.components['obb-collider
            '];
        return {
            el: handState.colliderEntity,
            getOBB: () => { return obbComp.obb; },
            testCollision: (otherOBB) => {
                return obbComp.intersectsOBB(otherOBB);
            }
        };
    } else {
        return handState.colliderEntity.components['sat-collider'];
    }
}
```

Prototipo resultante

El entregable de esta etapa fue una arquitectura consolidada y una escena de prueba dedicada (`colliders-test.html`). En este prototipo se implementaron dos cubos interactivos. Modificando únicamente el atributo `colliderType` de la entidad detectora en el código fuente, la escena alternaba entre el cálculo de intersecciones frame a frame (SAT) y el sistema guiado por eventos nativos (OBB). En ambos casos, las mecánicas de agarre y movimiento funcionaron con la misma precisión espacial. A nivel visual, las diferencias de eficiencia entre ambos colisionadores resultaron imperceptibles en estas escenas básicas.

Lecciones aprendidas y problemas resueltos

El mayor problema arquitectónico de este *sprint* fue gestionar la asincronía. Mantener dos vías de colisión generaba inicialmente un código muy acoplado y difícil de leer. El sistema OBB nativo de A-Frame funciona mediante eventos asíncronos del DOM (`obbcollisionstarted`),

mientras que el algoritmo SAT requería comprobaciones manuales intensivas en cada frame del ciclo `tick`.

Intentar que componentes como `hoverable` o `grabbable` escucharan eventos por un lado y calcularan distancias matemáticas por otro engordaba los archivos de JavaScript innecesariamente. La solución definitiva fue la implementación del patrón adaptador mencionado anteriormente. Al delegar la responsabilidad de la colisión a la interfaz `getHandCollider()`, el componente `grabbable` pasó a evaluar el contacto físico con una única llamada de código, ignorando por completo qué tecnología se estaba ejecutando por debajo. Esta decisión de diseño limpió la estructura del proyecto y facilitó enormemente la futura incorporación de nuevos componentes.

4.6. Sprint 4: Escalado bimanual (*stretch*) y robustez del sistema

Tras consolidar el agarre simple, el desarrollo se orientó hacia la manipulación avanzada de objetos. Este cuarto ciclo se centró en permitir que el usuario modifique el volumen de las entidades utilizando ambas manos de forma simultánea. Además de la nueva funcionalidad, se dedicó un esfuerzo considerable a corregir errores de precisión producidos de la manipulación en el espacio virtual.

Especificación de objetivos

El propósito principal fue ampliar las capacidades interactivas mediante el componente `stretchable`. Este módulo debía permitir escalar un objeto ya agarrado al separar o juntar las manos. Como meta secundaria, se estableció que el sistema fuera más robusto y automatizado. Se buscó que el componente pudiera gestionar sus propias dependencias, inyectando de forma automática el componente `grabbable` si el desarrollador olvidaba declararlo en la entidad HTML.

Tareas realizadas

El trabajo técnico comenzó con la programación del algoritmo de escalado. Dentro del bucle `tick`, se implementó una lógica que calcula continuamente la distancia euclidiana entre los dos colisionadores de las manos. Al detectar que ambas extremidades mantienen un contacto válido y ejecutan un pellizco, el sistema genera un factor de escala relativo.

Para mejorar el flujo de trabajo, se desarrolló el método `_ensureGrabbable`. Esta función verifica la existencia del componente de agarre al inicializar la entidad y lo añade dinámicamente si falta, asegurando que cualquier objeto escalable sea también manipulable. Asimismo, se programó una función de bloqueo de posición (`_lockPosition`) para garantizar que, mientras se escala el objeto, este mantenga su ubicación absoluta en la escena y no sufra desplazamientos involuntarios debidos a cambios en la jerarquía de los nodos.

Prototipo resultante

La entrega funcional de esta fase fue un sistema de manipulación bimanual fluido. El usuario puede ahora agarrar un objeto con una mano, introducir la segunda y, mediante un gesto de expansión o contracción, modificar el tamaño de la entidad en tiempo real. Este prototipo demostró ser capaz de manejar escalas extremas (desde el 10 % hasta el 1000 % del tamaño original) sin comprometer la estabilidad visual ni la posición del objeto frente al usuario.

Lecciones aprendidas y problemas resueltos

Este *sprint* presentó varios retos críticos de estabilidad. Las pruebas iniciales revelaron saltos bruscos de escala al iniciar la interacción debido al ruido de los sensores ópticos. Para mitigar este defecto, el código pasó a registrar la `initialDistance` exacta en el primer frame del gesto como un pivote de referencia inmutable, aplicando el factor de crecimiento siempre de forma relativa a ese punto seguro.

Un problema recurrente durante el testeo fue la aparición de ”pellizcos fantasma”: el sistema detectaba que una mano seguía cerrada tras soltar el objeto. Este error se resolvió implementando el método `_refreshContacts`, que fuerza una validación de contacto y gesto en cada frame, eliminando estados lógicos obsoletos.

Por otro lado, se detectó un desajuste entre la malla visual y su colisionador; tras agrandar un

objeto, la mano atravesaba la superficie o detectaba contacto antes de tiempo. Para corregirlo, se integró la función `_updateColliderSize`, que recalcula automáticamente las dimensiones de la caja OBB en función de la nueva escala aplicada. Finalmente, los bloqueos de movimiento que dejaban el objeto "congelado" tras una manipulación intensa se solucionaron refinando el sistema de coordenadas globales, asegurando que la entidad regrese a un nodo padre estable (`restParent`) al finalizar la acción.

4.7. Sprint 5: Interacciones de proximidad, arrastre y soltar

Una vez garantizada la estabilidad del agarre y el escalado, el desarrollo se orientó a construir un flujo de trabajo completo para el usuario. Este quinto ciclo de programación se centró en implementar el ciclo clásico de arrastrar y soltar (*drag and drop*), adaptándolo a las particularidades del seguimiento espacial de manos libres.

Especificación de objetivos

El objetivo principal fue completar la herramienta de manipulación añadiendo tres mecánicas consecutivas. En primer lugar, se requería un sistema de proximidad (*hover*) para indicar de forma visual cuándo un objeto estaba lo bastante cerca como para ser tocado. En segundo lugar, era necesario programar el estado de arrastre (*drag*). Finalmente, el reto más complejo consistía en desarrollar zonas de recepción (*drop zones*) capaces de evaluar y filtrar los objetos entrantes.

Tareas realizadas

Se desarrollaron tres componentes independientes: `hoverable`, `draggable` y `droppable`. El sistema de proximidad y arrastre se programó aprovechando la interfaz unificada `getHandCollider()` elaborada en el tercer *sprint*, evaluando el contacto geométrico entre la mano y la entidad.

Para el componente `droppable`, la tarea principal fue implementar un sistema de filtrado. Se configuró el esquema del componente para aceptar una propiedad `accepts`, capaz de leer selectores CSS estándar (como clases o identificadores). Para evitar que el sistema ejecutara búsquedas pesadas en el árbol del documento durante cada frame del bucle de renderizado, se integró la API nativa `MutationObserver` de JavaScript. Este observador vigila los cambios estructurales de la escena de forma asíncrona. Solo cuando se añade o elimina una nueva entidad

en el entorno, el componente actualiza su lista interna de objetos válidos, manteniendo intacto el rendimiento general de las gafas.

Prototipo resultante

El entregable de esta fase fue un escenario interactivo completo. El usuario percibe una respuesta visual (un cambio de color o brillo) al acercar su mano anatómica a una pieza. Al realizar el gesto de pellizco, inicia el arrastre. El entorno contiene varios paneles translúcidos que actúan como zonas objetivo.

Cuando el usuario desplaza un objeto sobre una zona compatible, esta se ilumina. Al soltar la pieza abriendo los dedos, el sistema emite eventos de éxito o rechazo dependiendo de si el objeto cumple con los filtros configurados. Este prototipo validó la usabilidad del sistema para construir mecánicas de clasificación o ensamblaje.

Lecciones aprendidas y problemas resueltos

Durante las pruebas intensivas de este prototipo, se detectó una importante fuga de memoria y un fallo lógico de activaciones cruzadas. Al entrar y salir repetidamente de una zona objetivo con un objeto agarrado, el sistema suscribía y eliminaba funciones de escucha (*event listeners*) para el evento `drag-end`. En ocasiones, estos eventos no se limpiaban correctamente. Como consecuencia, al soltar la pieza, se disparaban ejecuciones duplicadas o se validaba la caída en zonas incorrectas.

El problema arquitectónico se resolvió sustituyendo las variables simples por un sistema de seguimiento individualizado. Se introdujo el uso de la estructura `Map` de JavaScript para gestionar las dependencias. Como se detalla en el Código 4.3, el componente inyecta un mapa directamente en el objeto arrastrado. De esta forma, antes de registrar un nuevo evento, el sistema comprueba este registro y elimina estrictamente cualquier función residual vinculada al identificador de esa zona de recepción concreta. Esta solución eliminó por completo los comportamientos erráticos.

Código 4.3: Uso de un diccionario estructurado para rastrear y eliminar escuchadores asíncronos residuales, garantizando una única validación de caída por cada objeto.

```
if (!collidedWith._droppableListeners) {
```

```
    collidedWith._droppableListeners = new Map();
}

if (collidedWith._droppableListeners.has(this.el.id)) {
    const oldListener = collidedWith._droppableListeners.get(this.el.id);
    collidedWith.removeEventListener('drag-end', oldListener);
    collidedWith._droppableListeners.delete(this.el.id);
}
```

4.8. Sprint 6: Interacción directa (*click*) mediante gesto apuntar

Tras consolidar la manipulación espacial de objetos pesados, el desarrollo abordó la interacción con elementos de la interfaz de usuario (UI). Este sexto ciclo se centró en proporcionar una mecánica táctil y precisa, diseñada específicamente para activar botones e interruptores tridimensionales en el entorno virtual.

Especificación de objetivos

El propósito de esta fase fue ofrecer una alternativa al sistema de agarre general. Se buscó permitir la pulsación física de botones 3D mediante el uso exclusivo del dedo índice. El objetivo técnico era alejarse de los sistemas de puntero a distancia basados en rayos (*raycasting*), apostando en su lugar por una interacción directa y cercana. Esto exigía un nuevo nivel de precisión geométrica para aislar un único dedo anatómico y convertirlo en una herramienta de colisión independiente del resto de la mano.

Tareas realizadas

Para lograr esta funcionalidad, se programaron dos componentes complementarios: el detector `point-gesture` y la capa reactiva `clickable`. A diferencia de las iteraciones anteriores, la arquitectura requería posicionar el área de impacto en un punto muy específico. El sistema sitúa dinámicamente un colisionador (denominado `hand-point-collider`) exactamente en la coordenada espacial de la falange distal del dedo índice.

Además, se desarrolló una lógica matemática para validar la postura de la mano. El módulo evalúa continuamente la distancia de las articulaciones de cada dedo respecto al centro de la palma. El gesto solo se activa si el algoritmo confirma que el índice permanece completamente extendido hacia adelante, mientras los dedos anular y meñique se mantienen contraídos hacia el interior de la mano.

Prototipo resultante

La entrega funcional de este *sprint* consistió en una serie de entidades configuradas como botones físicos tridimensionales. Al aproximar la mano con el dedo índice estirado, el botón reacciona visualmente iluminándose al registrar proximidad. Si la punta del dedo atraviesa la geometría de la entidad, el sistema dispara de forma limpia la lógica de la aplicación asociada a ese botón. Este prototipo demostró la viabilidad de construir paneles de control que el usuario puede teclear físicamente, imitando la respuesta táctil del mundo real.

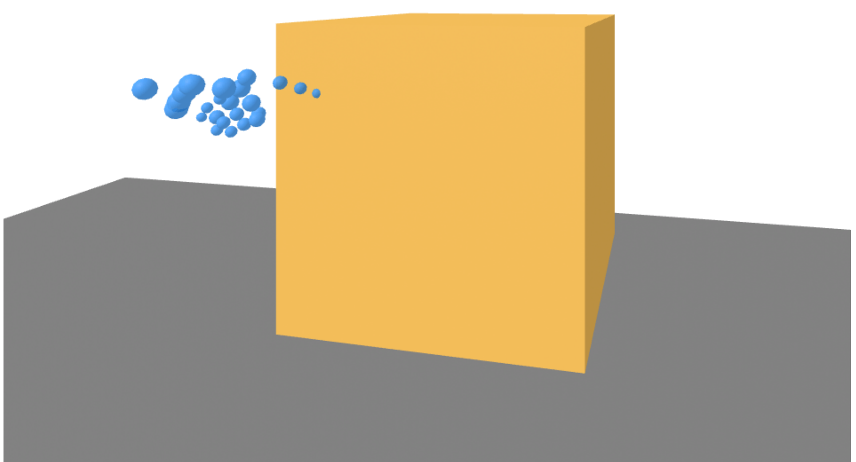


Figura 4.2: Demostración del componente `clickable` mediante la pulsación de un cubo con el dedo índice.

Lecciones aprendidas y problemas resueltos

Durante las sesiones de prueba, se detectó un conflicto importante. El sistema registraba falsas activaciones de clics cuando el usuario intentaba realizar un gesto de pellizco (*pinch*)

para agarrar un objeto cercano. Este error se producía porque, a nivel mecánico, el dedo índice también debe extenderse durante la fase inicial del movimiento de pellizco. El algoritmo interpretaba esta extensión anatómica natural como una intención directa de pulsar un botón.

Este conflicto lógico se resolvió introduciendo una variable matemática de cancelación dentro del detector de gestos. Como se observa en el Código 4.4, se implementó un umbral estricto (`pinchCancelThreshold`). Se modificó el cálculo para vigilar la separación relativa entre la punta del pulgar y la del índice. Si esta distancia desciende por debajo de los 4 centímetros, el sistema descarta automáticamente la validación del gesto de apuntar. Esta restricción solucionó por completo las activaciones cruzadas, otorgando siempre prioridad a la intención de agarre cuando los dedos comienzan a juntarse.

Código 4.4: Lógica de cancelación del gesto de apuntar basada en un umbral de distancia.

```
const distanceToThumb = indexTipPos.distanceTo(thumbTipPos);

const pinchCancelThreshold = 0.04;

if (distanceToThumb < pinchCancelThreshold) {
    this.isPointing = false;
    return;
}
```

4.9. Sprint 7: Integración y pulido XR

Tras programar y validar los componentes de interacción de forma aislada, el proyecto requirió una fase final de ensamblaje. Este séptimo y último ciclo tuvo como propósito sacar los componentes de sus entornos de prueba individuales y comprobar su comportamiento concurrente en escenarios de uso reales.

Especificación de objetivos

El objetivo principal de este *sprint* fue ensamblar todos los módulos independientes para validar la cohesión del sistema en flujos de trabajo completos (*end-to-end*). Para lograr una eva-

luación exhaustiva del rendimiento y la usabilidad, se estableció la creación de tres demostraciones complejas: la adaptación de una escena clásica de la biblioteca *Super-Hands*, un entorno de Realidad Aumentada (AR) apoyado en la detección de superficies físicas, y una demostración orientada a tareas denominada ”Taller de Ensamblaje”.

Tareas realizadas

El trabajo técnico consistió en la orquestación y programación de estas tres escenas integradoras. En primer lugar, se finalizó la adaptación de la escena demostrativa de *Super-Hands*. Se modificó el marcado HTML original para eliminar las dependencias de los controladores físicos y conectar las entidades a la nueva biblioteca unificada de gestos.

En segundo lugar, se implementó el entorno de Realidad Aumentada (`ar-demo.html`). Para ello, se integró el uso de la API nativa de *Hit Test*. Esta tecnología permite al visor escanear la habitación con sus cámaras, calcular la posición de superficies físicas planas (como una mesa) e instanciar los elementos virtuales interactivos anclados directamente sobre el mundo real.

Finalmente, se desarrolló la demostración del ”Taller de Ensamblaje”(`complete-assembly.html`). Esta tarea exigió diseñar y programar un escenario secuencial lógico donde el usuario debe combinar todas las mecánicas construidas (clic, proximidad, agarre, estiramiento y soltado paramétrico) para resolver un puzle tridimensional estructurado.

Prototipo resultante

El entregable de esta fase fue el conjunto de las tres demostraciones avanzadas completamente operativas. Este hito validó la madurez del código de la biblioteca, demostrando que los diferentes módulos interactivos pueden coexistir y ejecutarse de forma simultánea en la misma escena. El sistema demostró mantener la fluidez gráfica necesaria (*frame rate*) y procesó correctamente la concurrencia de gestos sin producir bloqueos lógicos ni conflictos de rendimiento.

Lecciones aprendidas y problemas resueltos

La integración conjunta de las mecánicas reveló retos arquitectónicos que fueron solucionados durante esta iteración. Al adaptar la escena de *Super-Hands*, se comprobó que traducir un entorno pensado para botones binarios a uno regido por gestos continuos requería reajustar

los umbrales de sensibilidad. Inicialmente, se detectaron conflictos donde las acciones de proximidad o arrastre interferían entre sí al existir múltiples objetos interactivos muy cerca unos de otros. El problema se solucionó afinando paramétricamente las dimensiones de los colisionadores y consolidando las reglas de exclusión mutua de los gestos.

En cuanto a la Realidad Aumentada, las pruebas mostraron que el uso de *Hit Test* exige una gestión de coordenadas sumamente estricta. Las pequeñas discrepancias entre el plano calculado por las cámaras y la posición física real provocaban que los objetos virtuales flotaran ligeramente o registraran colisiones irregulares al interactuar con ellos. Este desajuste geométrico se mitigó recalibrando los puntos de origen (*pivots*) de las mallas generadas dinámicamente y reajustando la tolerancia de los volúmenes, asegurando una manipulación estable sobre las superficies del entorno físico.

Capítulo 5

Pruebas y experimentos realizados

En este capítulo se describe la metodología empleada y los resultados obtenidos durante la fase de validación empírica de la biblioteca desarrollada. Para demostrar la usabilidad y la solidez de las mecánicas interactivas de seguimiento de manos libres, se diseñó un protocolo experimental controlado con usuarios reales. El objetivo principal consistió en someter el sistema a pruebas de uso práctico en un escenario que combinara todas las funciones de la arquitectura tridimensional de forma secuencial.

5.1. Descripción detallada del entorno experimental

La evaluación empírica del sistema se ejecutó utilizando de forma exclusiva el escenario de Realidad Virtual avanzado denominado Taller de Ensamblaje (`complete-assembly.html`). Esta escena se seleccionó debido a su naturaleza orientada a tareas estructuradas, la cual obliga a la ejecución concurrente de todas las familias de componentes diseñadas en este proyecto.

Las pruebas se llevaron a cabo en una sala interior con iluminación artificial constante y controlada. Esta estabilidad lumínica resultó indispensable para garantizar un rendimiento óptimo de las cámaras de infrarrojos externas del visor y evitar pérdidas aleatorias en el seguimiento de las articulaciones anatómicas. El hardware empleado para la ejecución de la prueba fue un visor autónomo Meta Quest 3, accediendo a la escena desplegada de forma remota en la red local a través del navegador integrado *Oculus Browser*.

5.2. Pruebas con usuarios sin experiencia técnica

Validar una biblioteca de interacción declarativa exige evaluar si los movimientos anatómicos calculados matemáticamente se traducen en acciones intuitivas para las personas. Para obtener datos con un alto valor de representatividad, se descartó el uso de perfiles con competencias en ingeniería de software o desarrollo de videojuegos.

Se seleccionó una muestra compuesta por seis sujetos experimentales ($N = 6$) caracterizados por una ausencia total de conocimientos de código o programación web. Asimismo, la muestra se configuró de manera heterogénea respecto al perfil demográfico y generacional. El grupo abarcó sujetos en un rango de edad comprendido entre los 24 años (usuarios jóvenes con alta familiaridad con interfaces móviles modernas) y los 65 años (usuarios mayores con baja exposición a entornos digitales interactivos y nula experiencia en Realidad Virtual).

5.2.1. Protocolo del experimento y procedimiento

Para asegurar la uniformidad en la recogida de datos y la validez de las métricas obtenidas, la prueba se rigió por un protocolo secuencial estricto. El procedimiento estructurado se dividió en las siguientes etapas consecutivas:

1. **Recepción y contextualización:** Se acomodó al sujeto en una posición de asiento fija frente al área central de la sala. Se le explicó de forma breve el propósito académico de la prueba.

2. **Fase de instrucción previa (fuerza del visor):** Antes de colocar el hardware en el usuario, se le comunicó verbalmente que el entorno se controlaría exclusivamente mediante el movimiento físico de sus manos libres, indicándole de forma explícita la ausencia total de mandos analógicos o botones físicos.

3. **Colocación y calibración:** Se ajustaron las cintas de sujeción del visor Meta Quest 3 a la fisonomía del sujeto, asegurando el enfoque óptico correcto y la estabilidad del dispositivo sobre la cabeza.

4. **Acceso al entorno inmersivo:** Se supervisó y guió a los participantes para acceder a la dirección web (URL) en el navegador del dispositivo. Seguidamente, se instruyó al

usuario para pulsar el botón nativo de WebXR en la pantalla. Esta acción permitió entrar a la escena de Realidad Virtual del entorno web bidimensional.

5. **Fase de entrenamiento estandarizada:** Se inició una escena de entrenamiento vacía (*sandbox*) durante un tiempo fijo de un minuto. En esta fase, el usuario pudo observar el esqueleto digital de las manos generado por el componente `hands-spheres`. Se le instó a realizar de forma libre movimientos de dedos en el espacio.

6. **Ejecución de la tarea evaluada:** Se cargó el entorno del Taller de Ensamblaje. Se inició el cronómetro en el instante exacto en el que el entorno gráfico se renderizó por completo frente al usuario.

7. **Cuestionario post-experimento:** Tras concluir la tarea o superar el tiempo límite de abandono (fijado en 5 minutos), se retiró el visor al usuario y se procedió a la lectura e introducción de sus valoraciones en un formulario estructurado de recogida de datos.

5.2.2. Definición de tareas y sistemas de medición

La prueba exigió a los sujetos resolver un problema secuencial continuo sin recibir ayuda externa durante el proceso. Las instrucciones específicas comunicadas a los usuarios se redujeron a la directriz genérica de “leer las indicaciones del panel frontal flotante y encajar las piezas de la mesa en sus moldes correspondientes”.

El experimento se dividió internamente en tres tareas consecutivas, diseñadas para evaluar componentes específicos de la arquitectura técnica:

- **Tarea 1 (Activación de interfaz):** Evaluar el componente `clickable`. El sujeto debía extender el dedo índice, aproximarlo al panel y pulsar físicamente un botón tridimensional azul para revelar las piezas de trabajo.

- **Tarea 2 (Traslado estático):** Evaluar los componentes `hoverable`, `grabbable`, `draggable` y `droppable`. El sujeto debía identificar el cubo amarillo y la esfera verde, aproximar la mano para observar el cambio de brillo, tomarlos mediante un pellizco continuo y depositarlos en sus respectivos moldes traslúcidos.

- **Tarea 3 (Manipulación paramétrica bimanual):** Evaluar el componente `stretchable`. El sujeto debía interactuar con un prisma rectangular morado de tamaño pequeño. Se requería sujetar el prisma con ambas manos a la vez empleando un pellizco bimanual, expandir los brazos para aumentar su volumen y, una vez alcanzado el tamaño, insertarlo en el molde rectangular final.

Para analizar el comportamiento del sistema de forma objetiva, se definieron dos categorías de métricas analíticas:

1. **Medidas cuantitativas:**
   - *Tiempo de resolución de la tarea:* Medido en segundos mediante un cronómetro externo desde la inicialización de la escena hasta la colocación de la última pieza.
   - *Tasa de caídas accidentales:* Registro del número total de veces que un objeto interactuable se desprendió involuntariamente de la mano del usuario antes de llegar a la zona objetivo, sirviendo como indicador de la robustez de la máquina de estados del pellizco.
2. **Medidas cualitativas:** Recogidas mediante una encuesta de satisfacción basada en una escala de 5 puntos (donde 1 significaba "Totalmente en desacuerdo" y 5 significaba "Totalmente de acuerdo"). Se evaluaron tres factores funcionales: la naturalidad del gesto de pellizco (`pinch-gesture`), la usabilidad del escalado bimanual (`stretchable`) y la precisión percibida al pulsar botones tridimensionales (`clickable`).

5.2.3. Resultados del experimento y análisis cuantitativo

Los datos recolectados tras la finalización de las seis pruebas de usuario revelaron una correspondencia directa entre la familiaridad tecnológica y el rendimiento temporal, confirmando la resolución exitosa del escenario por el 100 % de la muestra. La Tabla 5.1 sintetiza los registros métricos individuales ordenados de forma cronológica por el perfil del usuario.

El análisis de los tiempos demuestra que los usuarios más jóvenes (Usuarios 1 y 2) se adaptaron al sistema fácilmente, completando el puzle por debajo del umbral de los 100 segundos. Por el contrario, los perfiles pertenecientes a la generación senior (Usuarios 5 y 6) requirieron un

Cuadro 5.1: Registro individual de tiempos de resolución y caídas accidentales en la escena Taller de Ensamblaje.

Sujeto	Perfil demográfico	Tiempo de resolución (s)	Desagarres involuntarios
Usuario 1	Joven (Familiaridad alta)	74	0
Usuario 2	Joven (Familiaridad alta)	92	1
Usuario 3	Adulto (Familiaridad media)	115	1
Usuario 4	Adulto (Familiaridad media)	158	2
Usuario 5	Senior (Familiaridad baja)	194	2
Usuario 6	Senior (Familiaridad baja)	210	4
Media	**Muestra global**	**140,5**	**1,67**

periodo de entendimiento superior para comprender el funcionamiento del entorno inmersivo, elevando el tiempo medio general de resolución a los 140,5 segundos.

Respecto a los desagarres involuntarios, la media se situó en 1,67 eventos por prueba. Los errores se concentraron principalmente durante la ejecución de la Tarea 3. El sistema registró pérdidas momentáneas del esqueleto óptico cuando el usuario cruzaba físicamente los brazos al expandir el prisma rectangular, provocando una oclusión de los puntos de las falanges ante las cámaras del visor. Al abrirse el pellizco por falta de datos, la pieza se soltaba, obligando al usuario a reiniciar el traslado.

5.2.4. Análisis cualitativo y encuesta post-experimento

Para complementar los datos temporales, se analizó la percepción de los usuarios mediante el cuestionario final. El objetivo fue verificar si las conversiónes de matemáticas espaciales en mecánicas interactivas y detección de gestos resultaban naturales en su funcionalidad. Las puntuaciones medias obtenidas para cada familia de componentes se detallan visualmente en la Tabla 5.2.

Los resultados cualitativos confirmaron una excelente aceptación general de la interfaz. La interacción táctil directa (`clickable`) registró la valoración más alta (4,5). Los sujetos indicaron que pulsar botones con el dedo índice era intuitivo. Además, el sistema no fallo en ninguno de los casos en una detección cruzada errónea o en el intento del gesto apuntar de un

Cuadro 5.2: Puntuaciones medias obtenidas en la encuesta post-experimento (Escala de Likert de 1 a 5).

Factor funcional evaluado	**Puntuación media (sobre 5)**
Precisión de la interacción táctil (`clickable`)	4,5
Naturalidad del gesto de pellizco (`pinch-gesture`)	4,3
Usabilidad del escalado bimanual (`stretchable`)	3,5

participante y que este no fuese registrado, garantizando una experiencia fluida.

La métrica sobre la naturalidad del agarre simple (`pinch-gesture`) obtuvo 4,3 puntos. Este valor valida empíricamente que la separación de umbrales en el análisis de distancias euclidianas fue calculado correctamente. Por último, el componente `stretchable` (escalado bimanual) obtuvo la valoración más discreta (3,5 puntos). Esta puntuación inferior refleja la ligera curva de aprendizaje inicial que experimentaron los usuarios, especialmente en la generación senior, al tener que coordinar los movimientos de ambas manos simultáneamente en el vacío del entorno virtual. En términos globales, la evaluación certificó la madurez y estabilidad técnica de las herramientas desarrolladas.

Capítulo 6

Conclusiones

En este último capítulo se exponen las conclusiones generales tras la finalización del proyecto. El texto describe los objetivos cumplidos, el esfuerzo temporal y los recursos utilizados para el desarrollo de la biblioteca. Asimismo, se analizan las lecciones aprendidas durante la resolución de todos problemas que surgieron y se evalúa el impacto de la formación académica recibida. Finalmente, se definen las posibles mejoras a realizar si se continúa trabajando en el sistema.

6.1. Consecución de objetivos y balance del proyecto

El objetivo general planteado al inicio de este Trabajo de Fin de Grado se ha cumplido de forma satisfactoria. Se ha diseñado y desarrollado una caja de herramientas (*toolkit*) declarativa capaz de sustituir la dependencia de controladores físicos en la web inmersiva por el uso de mallas ópticas naturales. La biblioteca resultante ofrece una alternativa robusta a los componentes nativos de A-Frame, los cuales limitaban la manipulación a un único estado físico de agarre rudimentario.

La viabilidad de la arquitectura propuesta se demuestra mediante la consecución de los objetivos instrumentales:

- El esqueleto anatómico de 25 articulaciones se captura y renderiza de forma fluida a través de una única etiqueta limpia en el código fuente.
- Los algoritmos vectoriales clasifican correctamente los gestos de pellizco y apuntado

basándose en la distancia de las falanges.

- Los componentes interactuables replican con exactitud el comportamiento y la modularidad de la biblioteca clásica *Super-Hands*. Como se demostró en la fase de validación, la migración desde un entorno controlado por botones mecánicos hacia un sistema de interacción libre por seguimiento de manos es completamente transparente para el desarrollador final.

6.2. Esfuerzo y recursos dedicados

El desarrollo práctico del proyecto se extendió desde mediados de octubre de 2024 hasta principios de junio de 2026, abarcando el curso académico 2025/2026. Los recursos materiales empleados se limitaron a un ordenador portátil de gama media para la edición del código y a un visor Meta Quest 3 provisto por la universidad para la depuración en entornos reales.

Para garantizar la trazabilidad de la ingeniería de software, se ha realizado una estimación del tiempo dedicado a cada fase de la metodología iterativa. El esfuerzo computado y las actividades principales de cada ciclo se detallan de forma sintetizada en la Tabla 6.1.

Sumando las fases de programación, los periodos de investigación autónoma y la composición del documento técnico, el esfuerzo global del Trabajo de Fin de Grado se estima en un total de **462 horas de trabajo**.

6.3. Lecciones aprendidas y problemas resueltos

La resolución de este proyecto ha exigido superar importantes desafíos de ingeniería de software, abarcando desde la asimilación inicial del entorno de desarrollo y la adopción de buenas prácticas de programación, hasta problemas complejos de rendimiento y estabilidad matemática en dispositivos independientes:

1. **Curva de aprendizaje e incertidumbre inicial:** El desarrollo comenzó desde cero sin experiencia previa en el ámbito de la web tridimensional ni en la programación inmersiva. Comprender el flujo declarativo y asimilar la curva de aprendizaje de frameworks específicos como A-Frame supuso un reto inicial importante. Superar este obstáculo per-

mitió entender la gestión del bucle continuo de renderizado y el diseño de grafos de escena complejos.

2. **Adopción de buenas prácticas y modularidad:** El diseño original del código tendía al acoplamiento de funciones, dificultando la escalabilidad de la biblioteca. Se aprendió la importancia de aplicar buenas prácticas de ingeniería de software mediante el patrón de diseño Entidad-Componente-Systema (ECS). Siguiendo una estructura en la que los componentes creados eran independientes y reutilizables, aislando la lógica de los interactivos de los gestuales.

3. **Optimización de la escena** Las primeras versiones provocaban caídas en la tasa de refresco del visor debido a la creación y destrucción continua de objetos tridimensionales en cada frame. La lección aprendida fue la necesidad de implementar un diccionario estático, en el que el sistema reutiliza las entidades alterando unicamente su visibilidad.

4. **Corrección en la posición de los objetos** Al inicio o final de la manipulación de un objeto virtual se producía una desviación en la posición, cuando se cambia la jerarquía de los nodos del DOM. El problema se resolvió aplicando la matriz de transformación inversa de la mano justo en el instante del contacto, lo que permitió fijar la posición local relativa del objeto de forma estable.

5. **Control eventos asíncronos:** Cuando soltabas un objeto en una zona de caída en repetidas ocasiones se acumulaban funciones de escucha de eventos que saturaban el hilo de ejecución. La implementación de estructura de datos tipo `Map` vinculadas a los componentes interactuables, sustituyendo a las variables, facilito la eliminación de los escuchadores innecesarios.

6. **Detección de gestos y falsos positivos:** En la funcionalidad de los componentes gestuales al principio, el sistema registraba falsas activaciones cruzadas de clics al intentar realizar un agarre. La resolución de este conflicto se dió introduciendo variables matemáticas en las que si no se cumplían ciertos umbrales de distancia se cancelaba el gesto, priorizando siempre la intención de agarre del usuario.

6.4. Impacto de las asignaturas de la titulación

La formación recibida a lo largo del Grado en Ingeniería en Sistemas Audiovisuales y Multimedia aportó la base conceptual necesaria para abordar el proyecto. Las asignaturas más influyentes se estructuran en tres bloques:

- *Informática I y II* asentaron los fundamentos lógicos de la programación, la estructuración de algoritmos y el uso del paradigma orientado a objetos.
- *Gráficos y Visualización en 3D* resultó de vital importancia al introducir todo el ecosistema de aframe, los conceptos de álgebra lineal, matrices de transformación espacial y el uso fundamental del motor Three.js y el estándar WebGL.
- *Construcción de servicios en internet* y el *Laboratorio web* aportaron la experiencia práctica en el manejo del DOM, las tecnologías HTML5 y la arquitectura de eventos asíncronos en JavaScript.

No obstante, la naturaleza avanzada del proyecto exigió un proceso de aprendizaje autónomo para asimilar tecnologías que excedían el plan de estudios de la carrera. Fue necesario estudiar de forma más detallada e independiente la arquitectura ECS (*Entity-Component-System*) que rige el entorno de A-Frame. Asimismo, se requirió una investigación profunda sobre las especificaciones de bajo nivel de la API de WebXR para la captura anatómica de articulaciones, así como el uso de herramientas de empaquetado moderno mediante scripts de consola para la distribución de librerías.

6.5. Trabajos futuros

A pesar de haber consolidado un sistema funcional, la biblioteca posee un amplio margen de optimización de cara a futuras líneas de investigación y desarrollo:

- **Ampliación del catálogo gestual e interactuable:** La arquitectura modular permite recibir nuevos bloques lógicos. Se plantea el desarrollo de detectores para gestos como el puño cerrado o la palma abierta. Esto facilitará la creación de componentes interactuables adicionales para ejecutar acciones complejas, como el empuje físico de piezas.

- **Integración de dinámicas y físicas reales:** Una línea prioritaria es la transición desde la evaluación puramente geométrica hacia motores de simulación de cuerpos rígidos. Esto permitirá dotar a los objetos interactuables de propiedades físicas reales como la masa, la gravedad, la fricción y la inercia, logrando que los elementos caigan, reboten o colisionen de manera realista.

- **Optimización de cajas de colisión (*hitboxes*):** Se pretende refinar la precisión del Teorema SAT y de los colisionadores OBB subyacentes. El objetivo es diseñar mallas de impacto dinámicas que se ajusten de forma milimétrica tanto a la topología de los objetos complejos como a las falanges de los dedos, reduciendo al mínimo las detecciones falsas positivas.

- **Mejora en la visualización de las manos:** Actualmente el sistema depende de un renderizado básico mediante esferas independientes. Se propone integrar el soporte para mallas deformables continuas (*skinned meshes*) texturizadas. Esto otorgará al usuario una representación anatómica mucho más realista, orgánica y fluida dentro del entorno virtual.

- **Integración híbrida de periféricos:** Configurar el núcleo del software para permitir que las mallas ópticas de las manos y los controladores físicos comerciales coexistan en la misma sesión, otorgando al usuario la posibilidad de alternar dinámicamente entre ambos métodos de entrada según las necesidades de la escena.

Cuadro 6.1: Distribución del tiempo de trabajo y actividades principales desarrolladas en cada fase del proyecto.

Fase / Iteración	**Periodo**	**Dedicación**	**Actividades y foco principal**
Sprint 0: Aprendizaje	2 semanas	20 h	Comprensión de la arquitectura ECS de A-Frame y mitigación de riesgos posicionales.
Sprint 1: Núcleo visual	2 meses	80 h	Estudio analítico de la API WebXR y diseño del diccionario estático de articulaciones.
Sprint 2: Agarre básico	2 semanas	15 h	Estabilización de la máquina de estados del pellizco durante el periodo de exámenes.
Sprint 3: Colisiones	2 semanas	42 h	Diseño del patrón adaptador entre el Teorema SAT y el colisionador OBB nativo.
Sprint 4: Escalado	1 semana	21 h	Programación del factor de escala bimanual y sus respectivos bloqueos posicionales.
Sprint 5: Arrastre	2 semanas	42 h	Estructuración del observador mutacional asíncrono y los filtros de compatibilidad CSS.
Sprint 6: Pulsaciones	1 mes	56 h	Resolución algorítmica de las cancelaciones anatómicas cruzadas para evitar falsos clics.
Sprint 7: Integración	1 mes	84 h	Reestructuración del repositorio, empaquetado del archivo final y programación de escenas demostrativas.
Documentación técnica	2 meses	80 h	Redacción de la memoria del Trabajo de Fin de Grado y maquetación de la estructura en LaTeX.
Total global	**–**	**462 h**	**Consecución e integración de los objetivos del proyecto.**

Bibliografía

[1] T. Akenine-Möller, E. Haines, and N. Hoffman. *Real-Time Rendering*. A K Peters/CRC Press, 2018.

[2] R. Cabello and colaboradores. Three.js - JavaScript 3d library, 2024. https://threejs.org/.

[3] C. Ericson. *Real-Time Collision Detection*. Morgan Kaufmann, 2005.

[4] Git Community. Git SCM, 2024. https://git-scm.com.

[5] GitHub, Inc. About GitHub, 2024. https://github.com/about.

[6] Godot Engine Contributors. Godot Engine - Free and open source 2D and 3D game engine, 2024. https://godotengine.org/.

[7] Meta Platforms, Inc. Unity Interaction SDK - Meta Horizon OS, 2024. https://developers.meta.com/horizon/develop/unity/.

[8] Microsoft Corporation. Visual Studio Code Documentation, 2024. https://code.visualstudio.com/docs.

[9] F. Mittelbach, M. Goossens, J. Braams, D. Carlisle, and C. Rowley. *The LaTeX Companion*. Addison-Wesley Professional, 2004.

[10] Overleaf. Overleaf Documentation, 2024. https://www.overleaf.com/learn.

[11] W. M. Pfeiffer. A-Frame Super Hands Component, 2018.
https://github.com/c-frame/aframe-super-hands-component.

[12] Supermedium. A-Frame OBB Collider Component, 2024.
https://aframe.io/docs/1.7.0/components/obb-collider.html.

[13] Supermedium y colaboradores. A-Frame: A web framework for building virtual reality experiences, 2024.
https://aframe.io/.

[14] Unity Technologies. Unity Real-Time Development Platform, 2024.
https://unity.com/.

[15] W3C Immersive Web Working Group. WebXR Device API, 2024.
https://www.w3.org/TR/webxr/.

[16] R. C. y colaboradores. Three.js Documentation - Raycaster, 2024.
https://threejs.org/docs/#Raycaster.