Universidad
Rey Juan Carlos

ESCUELA DE INGENIERÍA DE FUENLABRADA

GRADO EN INGENÍERIA EN SISTEMAS AUDIOVISUALES Y MULTIMEDIA

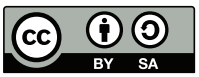
TRABAJO FIN DE GRADO

TÍTULO DEL TRABAJO CON LETRAS MAYÚSCULAS PARA SUSTANTIVOS Y ADJETIVOS

Autor : Juan José Arias Rojas

Tutor : Dr. Jesús María González Barahona

Curso académico 2024/2025



Dedicado a

todos aquellos que me animaron y apollaron

incluso en mis momentos de debilidad

II

Agradecimientos

Ha sido un camino de 5 años, pero al fin llega a su fin. Quiero agradecerle a mi familia por estar siempre a mi lado, apoyándome cuando lo necesitaba, por los sacrificios que han hecho con el fin de que yo llegue hasta aquí, por conseguirme el coche que me permitia ir todos los días a la universidad y ayudarme con la gasolina.

También quiero agradecerle a mi pareja, que pese a que nuestros caminos se cruzaron a mitad de este viaje, desde el momento en el que te conocí, tú siempre has estado a mi lado, siempre me has apoyado y siempre me has animado. Tú más que nadie me has dado la fuerza necesaria para poder seguir adelante hasta cumplir mi meta y espero poder ayudarte tanto como tu me has ayudado a mi.

Resumen

Este Trabajo de Fin de Grado se centra en dar un paso más en la evolucionando del sistema de detección de manos para entornos virtuales, consiste en el desarrollo de un sistema de detección y renderización de manos que permita a los usuarios poder interactuar con los distintos elementos de la escena de forma natural y fluida.

La detección de manos es una tecnología de Realidad Extendida (XR) que cada vez se usa más, pero hoy en día todavía afronta muchos obstáculos, ya sea porque no está muy desarrollada o tiene un uso muy limitado en la aplicación que se usa. Estos obstáculos hacen que los usuarios tiendan a usar más los controladores de los dispositivos VR, lo cual dificulta en cierta medida la sensación de inmersión dentro de escenas virtuales.

El proyecto se ha creado utilizado A-Frame, un framework el cual permite la creación y configuración de escenas tanto AR como VR dentro de un navegador compatible con WebGL. El sistema resultante permite a los usuarios visualizar sus manos dentro del entorno virtual, mientras que son capaces de realizar distintas acciones con estas, permitiendo así una mayor inmersión en la escena.

Summary

This Bachelor's Thesis focuses on advancing the evolution of hand detection systems for virtual environments. It involves the development of a hand detection and rendering system that allows users to naturally and fluidly interact with various elements within a scene.

Hand detection is an increasingly utilized Extended Reality (XR) technology, yet it still faces many obstacles today, either due to its underdeveloped state or its limited application in current uses. These challenges often lead users to favor VR device controllers, which somewhat hinders the sense of immersion within virtual scenes.

The project was created using A-Frame, a framework that enables the creation and configuration of both AR and VR scenes within a WebGL-compatible browser. The resulting system allows users to visualize their hands within the virtual environment while being able to perform various actions with them, thereby facilitating a greater sense of immersion in the scene.

Índice general

Índice de figuras

Capítulo 1

Introducción

La realidad virtual es un campo que cada vez toma mayor peso, e introducir al usuario cada vez más dentro del entorno virtual ha sido uno de los principales objetivos desde el comienzo. Al principio y hoy en día la principal solución para mejorar las interfaces de usuario en Realidad Extendida (XR), era el uso de los controladores de los distintos dispositivos VR. Pero el uso de los controladores no era natural en comparación al empleo de las manos, así que se inició el desarrollo de la detección de manos. Hoy en día la detección de manos está muy extendida, sin embargo, la utilización de los controladores sigue predominando, puesto que las tecnologías de detección de manos no están tan desarrolladas. El uso de estos controladores supone una limitación, produciendo una sensación de inmersión en la escena virtual no tan profunda como se podría desear, ya que el control de las distintas entidades que pueda haber no resulta tan natural para el usuario.

Este proyecto se centra en dar un paso más en la evolución de la detección de manos, permitiendo que los usuarios puedan utilizar sus manos dentro de los distintos entornos virtuales que estos creen en sus navegadores a través de A-Frame. Esto permite a los usuarios interactuar con los elementos de la escena de una forma más natural. Hoy en día ya existen componentes nativos en A-Frame que permiten la detección de manos, pero su funcionamiento es muy limitado, permitiendo únicamente agarrar elementos de la escena. Mi proyecto va un paso más allá de eso, permitiendo la detección de varios gestos de la mano y la ejecución de diferentes acciones al realizar dichos gestos, lo cual contrasta bastante con la tecnología actual, en la cual únicamente están implementados un gesto y una acción. Además de las acciones y gestos implementados en este proyecto, la forma de su diseño la convierte en una tecnología escalable y adaptable para la

creación de distintas escenas o aplicaciones VR. En el siguiente enlace se puede acceder al repositorio de GitHub que contiene el proyecto: `https://github.com/JuJoarias/TFG`

1.1. Contexto

La realidad virtual (VR) tiene sus orígenes en 1963, cuando el escritor Hugo Gernsback, presento su prototipo de gafas inmersivas `The Teleyeglasses`. La idea de este prototipo era la de permitir a los usuarios sentirse como si estuvieran dentro del mundo de la televisión, al sujetar una televisión en miniatura a la cabeza como si fuesen unas gafas. Años más tarde, en 1968, el científico de la computación Ivan Sutherland del MIT, desarrolló `The Sword of Damocles`, dicho sistema hoy en día se considera como el padre de las gafas de realidad virtual modernas. Este sistema incluía un casco sostenido por un brazo mecánico desde el techo, y utilizaba pantallas CRT para proyectar gráficos básicos que respondían al movimiento de la cabeza del usuario. Estos dispositivos sentaron las bases para la evolución de la realidad virtual.

Más tarde, cuando Palmer Luckey inicio su campaña para la creación de las Oculus Rift, llamo la atención de Facebook, la cual compro la empresa de Palmer en 2014. Desde entonces, la industria de la realidad virtual ha ido creciendo a pasos agigantados con la llegada de diversos dispositivos. Gracias a su versatilidad, la realidad virtual ha conseguido implantarse en áreas como el entretenimiento, la medicina o la educación. Hoy en día grandes empresas como Apple o Meta realizan grandes inversiones para seguir desarrollando estas tecnologías con el objetivo de potenciar la realidad virtual y hacerla más accesible.

Gracias a que grandes empresas invierten muchos recursos, la realidad virtual ha avanzado mucho, pero aún le queda mucho por mejorar. Desde hace años, las tecnologías de Realidad Extendida (XR) han sido uno de los principales pilares de la realidad virtual, en especial la detección de manos. Aunque la detección de manos es una tecnología que se usa desde hace años, no está tan desarrollada como otros campos. Esto se debe a que en la mayoría de ocasiones es necesario utilizar los controladores de los distintos dispositivos VR, puesto que el uso del handtracking está muy limitado o es poco fluido. Mi trabajo de fin de grado se centra en el desarrollo de una tecnología de detección de manos que permita a los usuarios interactuar, de forma más natural y fluida, con distintos elementos de la escena mediante escenas que estén dentro de un navegador.

1.2. Objetivos

1.2.1. Objetivo general

El objetivo principal de este proyecto es dar un paso más en la continua evolución de la realidad virtual, en especial, el sistema de detección de manos. Este consiste en el desarrollo de un sistema de detección de manos que permita la interacción fluida y natural con elementos de la escena a través de un navegador, mediante gestos que realice el usuario.

1.2.2. Objetivos específicos

A continuación se describen los diversos objetivos secundarios que fueron necesarios para realizar el objetivo principal.

- **Estudio de opciones actuales**: Estudio de los distintos componentes ya existentes de A-Frame dedicados a la detección de manos.
- **Implementar detección de manos**: Estudio e implementación de WebXR dentro de A-Frame para obtener los datos necesarios de las manos para posteriormente su renderización.
- **Implementar una herramienta de renderización de manos en A-Frame**: Construir un componente capaz renderizar las manos dentro del entorno virtual de la escena que reaccione de forma fluida a los movimientos del usuario, mientras mantiene una estructura realista que asemeje las manos reales.
- **Implementar detección de gestos**: Desarrollar un sistema capaz de detectar y distinguir los distintos gestos que puede realizar el usuario con sus manos.
- **Implementación de acciones**: Creación de una serie de componentes capaces de reaccionar a los distintos gestos del usuario para que estos realicen distintas acciones dependiendo del gesto y la intención del usuario.
- **Creación de escena**: Creación de una escena virtual donde poder visualizar las manos y las distintas acciones.

1.3. Estructura de la memoria

En esta sección se presenta la estructura de esta memoria, organizada en capítulos:

- **Tecnologías utilizadas:** Este capítulo enumera y describe las distintas tecnologías que han sido utilizadas a lo largo de todo el proyecto. Dichas tecnologías están divididas en principales y auxiliares y en cada una, además de su funcionalidad, se describe su importancia en el desarrollo del proyecto.

- **Desarrollo del proyecto:** Este capítulo describe el proceso que ha sido necesario para llegar a los resultados obtenidos. Se describe siguiendo la metodología `Agile`, dividiendo el proceso en distintos sprints donde cada uno tiene sus propios objetivos y resultados.

- **Resultados:** Aquí se presenta el resultado final del proyecto. Se describe de forma detallada la arquitectura de los distintos componentes que se han creado, al igual que se da una explicación de cara al usuario para que sea capaz de manejarse dentro de la escena y pueda interactuar con los elementos en ella.

- **Pruebas y experimentos:** En este capítulo se describen las opiniones y sensaciones que han tenido algunas personas que han ayudado a probar la escena final.

- **Conclusiones:** En este último capítulo se exploran las distintas lecciones aprendidas a lo largo del desarrollo de este proyecto, al igual que la aplicación de los distintos conocimientos que se han adquirido a lo largo de la carrera. Además, se contemplan las distintas aplicaciones a futuro que podría tener este proyecto o las posibles optimizaciones que se le podrían hacer.

Capítulo 2

Tecnologías utilizadas

En este capítulo se mostrarán las distintas tecnologías que han sido necesarias para el desarrollo de este proyecto, tanto de forma directa como indirecta. Del mismo modo, se explicará su funcionalidad y la manera en la que contribuyeron al proyecto. Las tecnologías se dividirán en 2 categorías, las principales y las auxiliares. Las principales son aquellas que han tenido un papel directo en el desarrollo del proyecto sin las cuales no se podría haber realizado, mientras que las auxiliares son aquellas que han ayudado al desarrollo de forma más indirecta.

2.1. Tecnologías principales

Aquí se nombran y explican aquellas tecnologías que han tenido una implicación directa con el proyecto y que han tenido un papel crucial e imprescindible con las cuales, sin ellas, no se habría podido llegar a los resultados obtenidos. Algunas de las principales tecnologías utilizadas han sido HTML5, JavaScript y A-Frame, estas tres tecnologías han sido el núcleo principal del proyecto. El resto de las tecnologías principales son aquellas en las que se apoyan esas tres, permitiendo que funcionen como lo hacen.

2.1.1. A-Frame

A-Frame [1] es un web framework diseñado para construir experiencias de realidad virtual. A-Frame está basado en HTML y en JavaScript, permitiendo realizar cualquier escena que uno pueda imaginar sin necesidad de conocimientos avanzados en gráficos 3D. Al estar basado en

HTML, es posible realizar escenas simples directamente desde un archivo HTML simplemente importando la librería de A-Frame y añadiendo dentro de la etiqueta `<a-scene>` cualquier elemento que desee el usuario.

Uno de los ejemplos más simples que se puede llegar a realizar utilizando A-Frame es el siguiente:

```
<html>
  <head>
    <script src="https://aframe.io/releases/1.6.0/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>
```

Listing 2.1: Escena A-Frame básica

Este código de HTML genera una de las escenas más básicas que se pueden hacer en A-Frame. La cual consiste en un plano y, encima, una serie de figuras geométricas, como se aprecia en la figura 2.1.

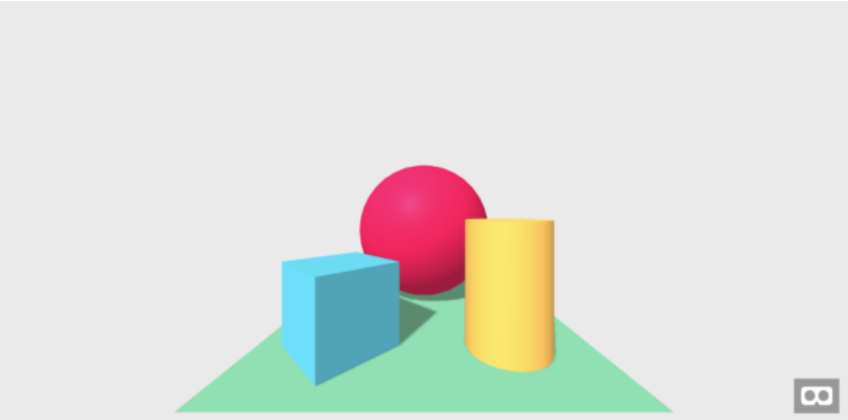


Figura 2.1: Escena básica de A-Frame

Una de las principales características de esta tecnología es su arquitectura basada en un

modelo de entidad-componente (*Entity Component System*), donde cada objeto dentro de la escena es una entidad que luego es integrada con Three.js para crear la escena. Una entidad en A-Frame es un objeto HTML que se crea añadiendo la etiqueta `<a-entity>` y el componente es la apariencia, comportamiento y funcionalidad que se le asigna mediante JavaScript.

Para hacer uso de un componente se requiere de una serie de pasos previos. Si el componente es externo, es necesario importarlo en la cabecera del HTML, del mismo modo que se importa A-Frame y posteriormente añadirlo a la entidad en la que uno desee usarlo. Por otra parte, si el componente es de nuestra creación, primero hay que registrar y definir el componente. Esto se hace en un archivo externo de JavaScript mediante el método de `A-Frame.registerComponente`. A este método, del mimo modo que una función primero se le asigna un nombre, este será el nombre del componente. Luego, este método consta de varias funciones internas, las cuales se encargan de definir la apariencia y lógica de dicho componente. Alguna de estas funciones son la función `schema`, `init` o `tick`. La función `schema`, define las propiedades principales del componente, estas se pueden modificar dependiendo de la lógica del componente desde el HTML a la hora de llamarlo. La función `init` se ejecuta una única vez al inicio cuando se inicializa el componente y la función `tick` se ejecuta a cada frame de la escena. Una vez el componente está registrado y programado, importamos el archivo que contiene el componente al HTML y lo insertamos a la escena en la entidad u objeto que deseemos.

Un ejemplo de la llamada de un componente dentro de un archivo de HTML se puede apreciar en 2.2

```
<html>
  <head>
    <script src="https://aframe.io/releases/1.6.0/aframe.min.js"></script>
    <script src = "componente1.js"></script>
    <script src = "componente2.js"></script>
  </head>
  <body>
    <a-scene>
      <a-entity componente1></a-entity>
      <a-box componente2></a-box>
    </a-scene>
  </body>
</html>
```

Listing 2.2: Ejemplo de llamada de entidad

En este ejemplo primero se muestra como a una entidad vacía se le añade un componente

llamado `componente 1` y también se muestra como a una entidad de cubo se le añade el componente `componente 2`. Del mismo modo, para la creación de un componente, en el archivo `.js` que posteriormente importaremos en la escena, únicamente es necesario registrar el componente y definir sus características como en el siguiente ejemplo:

```
AFRAME.registerComponent('componente1', {
init() {
  const box = document.createElement('a-box');
  box.setAttribute('position', '-1 0.5 -3');
  box.setAttribute('rotation', '0 45 0');
  box.setAttribute('color', '#4CC3D9');

  const sphere = document.createElement('a-sphere');
  sphere.setAttribute('position', '0 1.25 -5');
  sphere.setAttribute('radius', '1.25');
  sphere.setAttribute('color', '#EF2D5E');

  this.el.appendChild(box);
  this.el.appendChild(sphere);
}
});
```

Listing 2.3: Ejemplo de creacion de entidad

Este fragmento de código registra el componente `componente 1` el cual crea un cubo y un cilindro con unos colores, posición y rotación específicos y los añade a la escena.

Para este proyecto, A-Frame ha sido la tecnología principal, permitiendo elementos fundamentales como la creación y gestión de entornos de realidad virtual. También, permitió la integración de componentes diseñados mediante JavaScript y permitió el manejo de interacciones más complejas a la vez que optimizaba el rendimiento de la escena 3D.

2.1.2. HTML5

HTML5 [4] es la quinta iteración del lenguaje de HTML (*marcado de hipertexto*), el cual es la estructura básica y principal de toda página web. Desarrollado conjuntamente por W3C (*World Wide Web Consortium*) [10] y WHATWG (*Web Hypertext Application Technology Working Group*) [11]. HTML5 introduce mejoras sustanciales respecto a sus anteriores versiones, incluyendo nuevas etiquetas semánticas, soporte multimedia mejorado y compatibilidad ampliada con diversos navegadores y dispositivos, dándole mayor flexibilidad y dinamismo a la creación de páginas web.

Una de sus nuevas introducciones clave es la introducción de etiquetas semánticas como: `<header>`, `<article>`, `<section>` y `<footer>`, que facilitan una estructuración clara y accesible del contenido web. Estas etiquetas no solo ayudan a mejorar la organización de la información, sino que también facilitan la búsqueda de información por parte de los motores de búsqueda y facilitan la interpretación por parte de los desarrolladores. Además, HTML5 incorpora nuevas interfaces de programación de aplicaciones (*API*), destacándose las funcionalidades de almacenamiento local mediante `localStorage` y `sessionStorage`, que permiten la gestión eficiente de datos directamente en el navegador, eliminando la necesidad de bases de datos externas.

HTML5 también introduce mejoras significativas en la gestión de contenido multimedia, eliminando la necesidad de complementos externos. Para esto se implementaron las etiquetas `<audio>` y `<video>` las cuales permiten la incorporación directa de archivos de sonido y video en las páginas web, permitiendo mayor dinamismo en el desarrollo. Otro elemento que se añadió fue el elemento `<canvas>` el cual habilita la generación de gráficos y animaciones en tiempo real a través de JavaScript, permitiendo el desarrollo de aplicaciones interactivas y videojuegos en línea.

HTML5 se ha establecido firmemente como un estándar en el desarrollo de aplicaciones web progresivas (*PWA*) y en entornos multiplataforma. Su integración con tecnologías como CSS3 y JavaScript permite la creación de interfaces dinámicas y adaptativas, compatibles con una amplia gama de dispositivos, desde ordenadores hasta tabletas y teléfonos.

Gracias a que HTML es una tecnología extensible, tecnologías como A-Frame o WebXR han sido posibles de utilizar, ya que son extensiones de HTML. WebXR permitió poder trabajar con entornos VR desde el navegador, mientras que A-Frame fue el eje central del proyecto, donde se estructuró la escena. HTML fue una de las tecnologías más importantes, puesto que ofrece una estructura esencial para el desarrollo del proyecto, haciendo uso de sus extensiones de A-Frame y WebXR para poder mostrar los elementos de la escena en la realidad virtual.

2.1.3. JavaScript

JavaScript es un lenguaje de programación ligero y multiplataforma utilizado principalmente para la creación de contenido dinámico e interactivo para páginas web. Su flexibilidad permite desarrollar elementos para mejorar la interacción del usuario en páginas web tanto del lado del

servidor como del lado del cliente. En 1997, JavaScript fue estandarizado por *ECMA* como ECMAScript y poco después como un estándar *ISO*.

JavaScript, creado por Brendan Eich de Netscape en 1995 bajo el nombre de *Mocha*, posteriormente se renombró a LiveScript y finalmente a JavaScript. En el año 2000, JavaScript se extendió al lado de los servidores con la introducción de tecnologías como *Node.js*, y posteriormente su popularidad creció con la llegada de *AJAX*. Y Con la llegada de ECMAScript 6 en 2015, se introdujeron características más avanzadas y actualizaciones anuales, haciendo que hoy en día sea uno de los lenguajes más importantes en el desarrollo web.

JavaScript es un lenguaje de programación de alto nivel, lo que significa que su lenguaje está diseñado para ser lo más 'humano' posible, permitiendo una rápida comprensión del código sin necesidad de un nivel alto de conocimientos. Es un lenguaje basado en eventos, ya sean entradas de ratón o entradas de teclado, permitiendo la creación de interfaces de usuario interactivas. JavaScript puede integrarse en un HTML dentro de la etiqueta `<script>` de forma directa, escribiendo el código. También, se puede introducir códigos de JavaScript, los cuales estén en archivos distintos con la extensión correspondiente al lenguaje (.js), esto es posible vinculando dicho archivo en la cabecera del HTML.

Para este proyecto formo un papel crucial, ya que fue con JavaScript que se desarrollaron los distintos componentes de A-Frame que se usaron para el funcionamiento del proyecto. Permitió diseñar e implementar la lógica que hay tras cada componente, permitiendo alcanzar los resultados obtenidos.

2.1.4. Three.js

Three.js [6] es una biblioteca de código abierto de JavaScript utilizada para la creación de gráficos 3D en los navegadores web. Three.js funciona sobre WebGL, la cual permite generar y visualizar animaciones y escenas 3D en el navegador sin necesidad de complementos. La tecnología de WebGL también puede usarse junto con el elemento `<canva>` de HTML.

Esta biblioteca proporciona una API de alto nivel la cual permite al usuario la creación y manipulación de geometrías 3D como cubos, esfera o planos, así como la aplicación de texturas o la aplicación de tanto cámaras como de efectos de iluminación en las escenas, permitiendo que el desarrollo de dichas escenas sea mucho más simplificado. También, Three.js permite la posibilidad de importar modelos 3D desde archivos distintos creados desde aplicaciones distin-

tas.

La biblioteca de Three.js ofrece una alta variedad de herramientas que permiten la gestión y control de usuario, facilitando así tanto la navegación como la interacción dentro de las escenas 3D. También, Three.js soporta animaciones suaves y dinámicas tanto para objetos en la escena como para cámaras, lo cual permite la creación de efectos visuales más dinámicos e impresionantes, al igual que juegos interactivos.

Gracias a la amplia variedad de posibilidades que permite la biblioteca de Three.js, las opciones de los elementos o escenas que se pueden llegar a crear son básicamente ilimitadas. Siendo capaz de generar escenas complejas como la de la figura 2.2

Figura 2.2: Escena de Three.js

Three.js permitió añadir una capa más simplificada a WebGL, simplificando así el proceso de creación y manipulación de los elementos 3D. Permitió crear un entorno 3D con mayor dinamismo y control para el proyecto.

2.1.5. WebXR

WebXR [9] es una tecnología desarrollada por W3C (*World Wide Web Consortium*) que permite crear experiencias inmersivas desde el navegador. Esta tecnología combina la Realidad Aumentada (*AR*) y la Realidad Virtual (*VR*) con la accesibilidad de un navegador web. Lo cual permite a los usuarios experimentar e interactuar con entornos tridimensionales a través de cualquier dispositivo con acceso a un navegador compatible.

A medida que la tecnología de WebXR ha ido evolucionando, se han desarrollado distintos

tipos de experiencias para poder adaptarse a distintos contextos y necesidades. Hoy en día, se podría clasificar los distintos tipos en 3 categorías:

- **WebXR AR:** Este tipo de WebXR combina el mundo virtual y el mundo real, permitiendo que distintos elementos del mundo virtual puedan superponerse en el entorno físico a través de la cámara de un dispositivo compatible, pero sin que llegue a influir el mundo real en los elementos virtuales.

- **WebXR VR:** Este modo permite a los usuarios sumergirse en el entorno virtual creado, y con la ayuda de dispositivos suplementarios como auriculares, la experiencia es aún mayor. Esta tecnología permite a los usuarios interactuar y experimentar en primera persona distintos entornos virtuales, desde juegos y entretenimiento, hasta simulaciones virtuales y visualizaciones en 3D.

- **WebXR MR(Mixed Reality):** Esta tecnología combina el mundo real y el virtual de forma más profunda que la tecnología AR. Permite a los usuarios poder interactuar con elementos tantos virtuales como físicos y que estos puedan interactuar entre sí. Este tipo de combinación proporciona un nivel mayor de interacción entre el usuario y y su entorno, dando mayor número de posibilidades para la creación de contenido.

Estos distintos tipos de WebXR ofrecen distintos tipos de experiencias dependiendo del entorno virtual que se quiera diseñar.

Fue gracias a WebXR, en este proyecto se pudieron realizar las distintas escenas inmersivas que permitieron ir comprobando el avance del proyecto con cada una de las demos que se creaban y que posteriormente se explicaran. Además de eso, fue gracias a WebXR que se pudo realizar la detección de las manos, ya que ya posee una función integrada que permite el handtracking y al querer trabajar sobre el navegador era la mejor opción para ello.

2.1.6. WebGL

WebGL [3] (*Web graphics Library*), se trata de una tecnología de bajo nivel multiplataforma usada para la renderización de gráficos tanto tridimensionales como bidimensionales dentro de cualquier navegador que sea compatible.

WebGL fue lanzado en 2011 por el grupo Khronos. Esta tecnología se fundamenta en OpenGL ES, la cual es una variante simplificada de OpenGL, diseñada para dispositivos móviles. WebGL ha sufrido numerables actualizaciones, lo cual ha permitido una evolución continua que ha ido mejorando tanto su funcionalidad como compatibilidad tanto en navegadores de escritorio como en navegadores en dispositivos móviles.

WebGL está diseñado para trabajar directamente con la GPU (*Graphic Processing Unit*) del dispositivo, lo cual permite un mayor aprovechamiento de la computación para poder generar gráficos detallados y de alta calidad. Además, la tecnología de WebGL está diseñada para poder integrarse de manera fluida con otros estándares de desarrollo web como HTML, CSS o DOM.

Gracias a esta tecnología fue posible renderizar en la escena las distintas entidades que forman las manos del mismo modo que las otras entidades que se crearon para ir comprobando si los componentes creados funcionaban.

2.2. Tecnologias auxiliares

En esta sección se detallan y describen las distintas tecnologías que, aunque no hayan influido de manera directa en los resultados del proyecto, han tenido un papel crucial para el desarrollo de este. Entre dichas tecnologías se encuentra el *IDE* utilizado durante el proyecto, al igual que el dispositivo utilizado para realizar las pruebas de las demos, la plataforma donde se almacenó el código y también el software utilizado para la redacción de esta memoria.

2.2.1. Visual Studio Code

Visual Studio Code (*VS Code*) es un potente entorno de desarrollo integrado (*IDE*) el cual está disponible para Windows, Linux, MacOS y versión web. VS Code es una de las plataformas de edición de código más utilizadas a nivel mundial por toda clase de desarrolladores de software debido a su versatilidad, flexibilidad y amplia gama de extensiones que facilitan la edición y depuración de código.

Una de las pestañas más importantes durante el desarrollo fue su pestaña de **Source Control**, la cual luego de conectar mi repositorio de GitHub, permite ir guardando el código en GitHub para tener un control de las distintas versiones del código.

Gracias a las numerosas extensiones que hay disponibles, facilitan en gran medida la creación y depuración de código, permitiendo cosas como autocompletar palabras o expresiones o permitir compilar o depurar con algún lenguaje que VS Code no tenga por defecto. Algunas de las extensiones utilizadas en este proyecto han sido:

- **A-Frame completition**: Permite autocompletar rápidamente a la hora de escribir los distintos elementos o Snippets que posee A-Frame
- **Error lens**: A la hora de depuración, permite visualizar dentro del código si hay algún error de sintaxis o lógica
- **LaTeX Workshop**: Permite la escritura, compilación y previsualización de la memoria de este proyecto.

2.2.2. GitHub

GitHub es una plataforma basada en la nube que permite almacenar distintos repositorios y en cada repositorio código. GitHub está basado en Git [5], el cual fue creado en 2005 por Linus Torvalds como un sistema de control de versiones de código abierto. Este sistema fue desarrollado con la intención de ayudar a los desarrolladores a tener en cualquier dispositivo con acceso a internet todo el historial de código que están desarrollando.

Esta tecnología es ampliamente utilizada a nivel mundial por desarrolladores de todo tipo debido a su capacidad de almacenar, soportar y gestionar proyectos de toda clase. Una de las funciones clave que presenta la plataforma, es la capacidad de crear ramificaciones (branches). Esta opción permite a los desarrolladores poder generar copias del código del repositorio en el que están trabajando para poder trabajar en paralelo y posteriormente poder integrar los cambios realizados en paralelo al código principal. Esta opción también permite que más de un desarrollador pueda trabajar en el mismo código, haciendo que cada uno trabaje en paralelo en ramas distintas.

GitHub aprovecha todas las ventajas que permite Git, permitiendo crear repositorios en la nube para sus proyectos Git, que los desarrolladores pueden llegar a compartir con otras personas. Esta plataforma también facilita la colaboración entre desarrolladores y aparte de las ventajas ya mencionadas de Git permite también el uso de herramientas de gestión de proyectos, al igual que acciones automáticas.

Durante este proyecto se creó el repositorio de GitHub *TFG*[1] donde se fue guardando todas las versiones del código al igual que todas las demos y pruebas realizadas.

2.2.3. Meta Quest 3

Meta Quest se trata de un dispositivo inalámbrico diseñado para disfrutar de contenido de realidad virtual (VR) por Meta [8]. Dicho dispositivo está compuesto por unas gafas, un micrófono y auriculares integrados en un único dispositivo que el usuario pone en su cabeza y un par de mandos inalámbricos. La primera versión fue lanzada en 2020 con las Meta Quest 2 y 3 años más tarde, en 2023, se lanzaron las Meta Quest 3.

Para este proyecto se han utilizado unas Meta Quest 3, las cuales contaban con una memoria de 8 GBB de memoria RAM, una resolución de 2064x2208 píxeles por cada uno de los ojos. También, cuenta con mejoras respecto al campo de visión (FOV) respecto a su versión anterior. Para este proyecto jugaron un papel crucial, ya que al enfocarse en la detección de mandos el proyecto, sin las gafas, no se habría podido probar los resultados que se iban obteniendo.

Figura 2.3: Meta Quest 3

2.2.4. LaTeX

LaTeX [7] es un software de uso libre de creación de documentación de alta calidad. Este sistema es altamente utilizado para la creación de documentación técnica o científica de media o larga extensión, aunque gracias a su versatilidad se puede utilizar para cualquier tipo de

[1] `https://github.com/JuJoarias/TFG`

documentación.

Entre las distintas características de LaTeX cabe destacar su capacidad de manejar expresiones matemáticas complejas, lo cual es una herramienta indispensable para cualquier documento científico. También gracias a su software motor TeX LaTeX es capaz de convertir los comandos de texto, utilizados para expresar los resultados tipográficos, en un archivo PDF profesional.

Además de facilitar la creación de fórmulas matemáticas complejas, LaTeX también ofrece otras facilidades avanzadas, como puede ser la creación de un índice del contenido, un índice de las figuras utilizadas, gestión de bibliografías o referencias simplificando la organización del documento y la citación de figuras o elementos externos en la bibliografía.

Para este proyecto, LaTeX facilito considerablemente el trabajo a la hora de crear este documento, ayudando con la creación y gestión de índices, imágenes y bibliografía al igual que a estructurar el texto del documento de forma correcta.

Capítulo 3

Desarrollo del proyecto

En este capítulo se describe de forma detallada como fue el desarrollo del proyecto. Dicho desarrollo se describe en forma de sprints siguiendo una estructura de una metodología *Agile*[2], pese a que el propio proyecto no siguió dicha metodología. El desarrollo explica desde los inicios y planteamiento del proyecto hasta obtener los resultados finales.

Por la forma en la que se fue planteando el proyecto, al final de cada sprint se planteaban los objetivos del siguiente para llegar al objetivo final de tener una detección de manos funcional y que permitiese interactuar con los distintos elementos de la escena.

3.1. Sprint 0

Puesta en marcha del proyecto, planteamiento con el tutor y dominio del uso básico de A-Frame.

3.1.1. Objetivos

El objetivo principal de este sprint se puede dividir en 2 partes principales.

- **Dominio del uso básico de A-Frame**. Aprender a usar de forma correcta como funcionan las escenas, al igual que la creación e implementación de componentes o el empleo de librerías de A-Frame.
- **El planteamiento con el tutor**, donde se identificaron las necesidades para el proyecto, también se plantearon los primeros objetivos principales del proyecto, detectar y dibujar

las manos y poder usarlas para interactuar con ellas.

3.1.2. Tareas Realizadas

La identificación de las necesidades y requisitos de un proyecto es una parte fundamental. Durante las primeras charlas con el tutor se plantearon algunos de los posibles objetivos a los que se podía llegar y los posibles caminos que puede tomar este proyecto. Al principio se plantearon varios caminos, pero la idea general de todos ellos era la creación de un sistema de detección de manos que sea capaz de interactuar con distintos elementos dentro de la escena.

Para lograr llegar a ese objetivo primero fue necesario dominar los elementos básicos de A-Frame como viene a ser la organización de la escena, el uso y creación de componentes y el uso de eventos. Para ello, primero se creo primero el repositorio de *Git* con el que trabajaremos durante todo el proyecto. Se crearon una serie de escenas sencillas con el objetivo de dominar el empleo de componentes y eventos simples.

3.1.3. Resultados

Para familiarizarme aún más con el entorno de A-Frame se generaron una serie de escenas muy sencillas donde se generaba y utilizaba un componente customizado y se hacía uso del evento nativo de A-Frame `click`. Estas escenas permitieron un mayor entendimiento del funcionamiento tanto de componentes como de eventos, que fueron fundamentales para el resto del proyecto. El resultado de dichas pruebas para mejorar la comprensión de A-Frame se pueden apreciar en la figura 3.1.

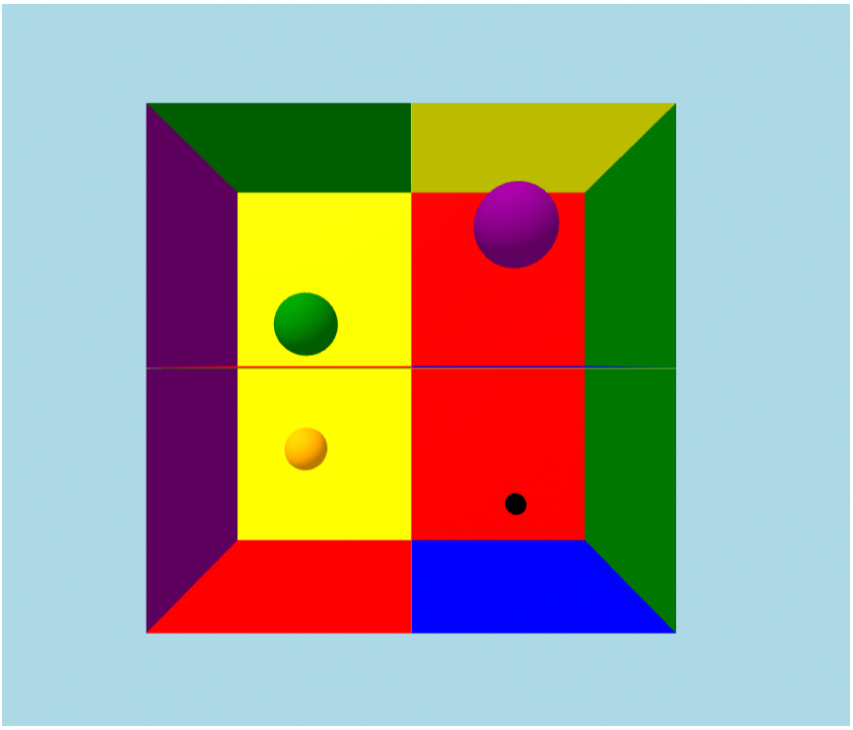


Figura 3.1: Escena de test de A-Frame

Tras la familiarización con el entorno de A-Frame y sus características, el proyecto fue formalizado formalmente. Tras ellos se planteo el siguiente sprint del proyecto, donde ya se empezarían los primeros pasos formales de este.

3.2. Sprint 1

Investigación del handtracking y primeras implementaciones de este en el proyecto.

3.2.1. Objetivos

En este primer sprint oficial del proyecto, el objetivo principal era investigar y comprender el uso del handtracking en A-Frame. Este era el paso principal en el que se basaría todo el proyecto, ya que sin una comprensión del funcionamiento del handtracking en A-Frame, este proyecto habría sido imposible de realizar. Además de obtener una primera demo del handtracking en la escena.

3.2.2. Tareas Realizadas

Los primeros pasos fueron investigar el handtracking en A-Frame con los elementos ya existentes como *super hands*[1], o los componentes nativos de A-Frame `hand-controls` o `hand-tracking-controls`. Para ellos se usaron pequeñas escenas de prueba para comprender mejor el comportamiento de las manos en la realidad virtual, y del mismo modo, tener una idea más clara del objetivo final del aspecto de handtracking.

Tras la investigación de las manos y puesto que el proyecto se desarrollo en A-Frame para que funcione en los navegadores, se decidió utilizar la tecnología de WebXR para la detección de manos, puesto que esta tecnología ya posee un sistema preparado para ello. Además, usando la tecnología de WebXR nos permitía trabajar con cada una de las articulaciones de la mano de forma independiente, permitiéndonos tomar la información necesaria de las articulaciones cuando es necesario.

Siguiendo la documentación de *WebXR*[2], se empezó el desarrollo de las manos. Para ello en el código fue necesario tratar cada una de las manos de forma independiente, del mismo modo que cada una de las articulaciones, como se muestra en el esquema de las manos de WebXR en la figura 3.2.

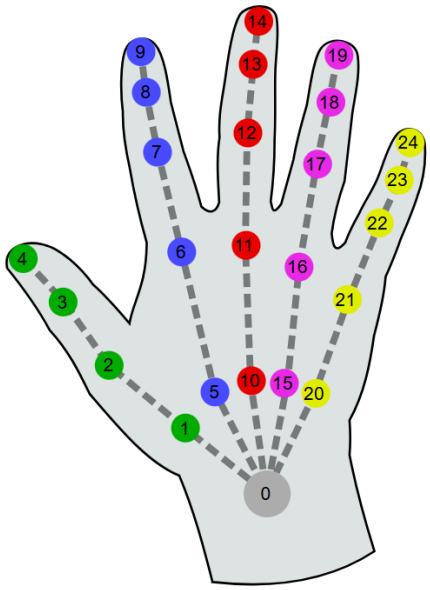


Figura 3.2: Esquema manos segun WebXR

[1] `https://github.com/c-frame/aframe-super-hands-component`
[2] `https://www.w3.org/TR/webxr-hand-input-1/`

Para poder crear las manos de forma exitosa dentro del entorno de realidad virtual siguiendo WebXR, primero se crea el componente que será el encargado de detectar y dibujar las manos en este proyecto, dicho componente se llama `hand-skeleton`. Dicho componente no tenía un `schema`, en el `init` se definían las distintas variables que eran necesarias para el funcionamiento del código. En el HTML que define la escena únicamente era necesario añadir una vez el componente como una entidad. Pero era imprescindible definir todas y cada una de las articulaciones, como se muestra en el fragmento de código 3.1, pero esa definición únicamente se realizaba en el código de JavaScript.

```
const orderedJoints = [
  ["wrist"],
  ["thumb-metacarpal", "thumb-phalanx-proximal", "thumb-phalanx-distal", "thumb-tip"],
  ...
  ["pinky-finger-metacarpal", "pinky-finger-phalanx-proximal", "pinky-finger-phalanx-
      intermediate", "pinky-finger-phalanx-distal", "pinky-finger-tip"]
];
```

Listing 3.1: Definición de articulaciones

Después de definir todas las articulaciones era necesario procesarlas. Para ello, se creo la función que se muestra en el fragmento de código 3.2. En la función `renderHandSkeleton` se crea un bucle de código el cual, toma la sesión XR de la escena y por cada input de la sesión, toma cada una de las articulaciones, las detecta en la mano y genere una esfera en una segunda función únicamente encargada de generar esferas. Dichas esferas se crean en la posición correspondiente a la mano real y también se ajusta el radio de la esfera para que coincida con la mano, esta función nos permite renderizar en la escena las manos del usuario de forma que parezca lo más natural posible.

```
renderHandSkeleton: function () {
  const session = this.el.sceneEl.renderer.xr.getSession();
  if (!session || !this.frame || !this.referenceSpace) {
      return;
  }
  const inputSources = session.inputSources;
  for (const inputSource of inputSources) {
      if (inputSource.hand) {
          const hand = inputSource.hand;
          const handedness = inputSource.handedness; // Determina si es la mano derecha o
              izquierda
          for (const finger of orderedJoints) {
              for (const jointName of finger) {
```

```
                const joint = hand.get(jointName);
                if (joint) {
                    const jointPose = this.frame.getJointPose(joint, this.referenceSpace);
                    if (jointPose) {
                        const position = jointPose.transform.position;
                        if (!this.spheres[handedness + '_' + jointName]) {
                            this.spheres[handedness + '_' + jointName] = this.drawSphere(
                                jointPose.radius, position);
                        } else {
                            this.spheres[handedness + '_' + jointName].object3D.position.
                                set(position.x, position.y, position.z);
                        }
                    }
                }
            }
        }
    }
  }
},
```

Listing 3.2: Dibujo de las manos en la escena

Esta función se encuentra dentro de la función `tick` del componente manos, así que a cada frame el propio código revisa la posición de todas y cada una de las articulaciones para cada una de las dos manos de entrada. Además de detectar y dibujar las manos en la escena, en esta primera demo se implementó el gesto de `Pinch`, distinguiendo cada mano y realizando una acción distinta dependiendo de la mano que lo realizase.

Primero, para la detección del gesto se creo la función de `detectGestures`, como se muestra en el fragmento de código 3.3. En dicha función, se tomaba la posición de la punta del dedo índice y del pulgar y se calculaba la distancia entre ambos puntos.

```
detectGestures: function () {
  const session = this.el.sceneEl.renderer.xr.getSession();
  const inputSources = session.inputSources;
  let rightPinching = false;
  let leftPinching = false;
  for (const inputSource of inputSources) {
      if (inputSource.hand) {
          const thumbTip = this.frame.getJointPose(inputSource.hand.get("thumb-tip"), this.
              referenceSpace);
          const indexTip = this.frame.getJointPose(inputSource.hand.get("index-finger-tip"),
               this.referenceSpace);
          if (thumbTip && indexTip) {
              const distance = calculateDistance(thumbTip.transform.position, indexTip.
```

```
                    transform.position);
                if (distance < pinchDistance) {
                    if (inputSource.handedness === 'right') {
                        rightPinching = true;
                        if (!this.rightPinching) {
                          this.createBoxAtHand(inputSource.hand);
                        }
                    } else if (inputSource.handedness === 'left') {
                        leftPinching = true;
                        if (!this.leftPinching) {
                            this.selectObjectAtHand(inputSource.hand); // Selecciona un objeto
                                 cercano para mover
                        }
                    }
                }
            }
        }
    }
    this.rightPinching = rightPinching;
    this.leftPinching = leftPinching;
},
```

Listing 3.3: Función detectGestures

Si la distancia era inferior a 0.02 (valor obtenido tras realizar varias pruebas), se comprobaba que mano era la que lo realizaba y luego se llamaba a la función correspondiente. Si se trataba de la mano derecha, se creaba un cubo verde en la posición de la punta del dedo índice. Y si se trataba de la mano izquierda, si esta se encontraba lo bastante cerda de uno de los cubos creados por la mano derecha (a una distancia inferior a 0.15) la posición del cubo copiaría la de la punta del dedo índice izquierdo hasta que se soltase el gesto.

3.2.3. Resultados

Como resultado de este sprint se creo la demo de *pinch test*[3], en la figura 3.3 se aprecian los resultados de este sprint.

[3] `https://github.com/JuJoarias/TFG/blob/main/first_steps/first_hands/pinch_test.html`

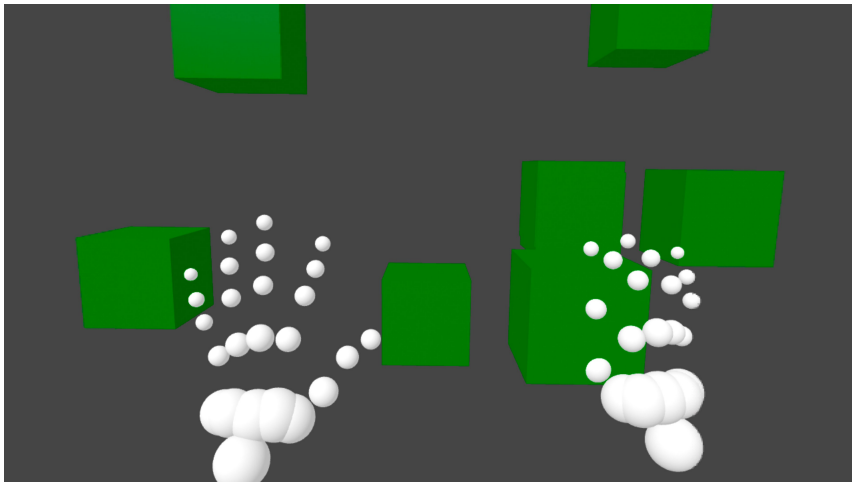


Figura 3.3: Primera demo de manos

Las esferas se dibujaban correctamente y los gestos funcionaban, pero había bastantes aspectos a mejorar. Primero, por sugerencia del tutor, se debía modificar el componente para que fuese necesario añadirlo 2 veces a la escena, 1 por cada mano. También, respecto al gesto, con la mano derecha había que cambiar el funcionamiento ya que el que creara un cubo no seria necesario más adelante. Y respecto a la mano izquierda, se identificó que a la hora de tomar el cubo, este se transportaba hacia la posición del dedo índice. Y aunque esto era lo deseado de primeras, visualmente no quedaba bien por el hecho de que si se agarraba una esquina el cubo se transportaba, tragándose la mano en vez de agarrar el cubo mientras mantenía la distancia entre el dedo y el centro del cubo. Aparte de eso, en el caso de que hubiese más de un cubo muy pegado, el código no identificaba siempre de forma correcta el más cercano, haciendo que a veces se tomara el cubo no deseado.

Todos estos factores dieron paso al siguiente sprint, donde se corregirían los errores resaltados. Aparte se intentaría optimizar la creación de las manos y se añadiría la emisión y gestión de eventos, ya que en esta primera demo, al realizar el gesto, se llama a la función correspondiente si utilizar eventos.

3.3. Sprint 2

Introducción de eventos al sistema y optimización del manejo de manos.

3.3.1. Objetivos

Tras las observaciones del sprint anterior, para este sprint se replanteo el funcionamiento de los gestos. También, se reorganizo la forma de renderización de las manos en la escena y la detección de los gestos con el fin de obtener un código más optimizado.

3.3.2. Tareas Realizadas

Primero, se reorganizo el componente encargado de dibujar las manos que se creo en el sprint anterior. Dicho componente se renombro a `manos`, nombre que se mantendría hasta el final del proyecto. A dicho componente se le añadió un `schema`. En dicho `schema` hay únicamente una variable llamada `hand`, dicha variable únicamente acepta `left` o `right` como entradas. Este cambio era para que el componente distinguiera las manos y únicamente dibujase una mano a la vez, esto hizo que para dibujar ambas manos en la escena fuese necesario añadir el componente 2 veces, 1 por cada mano. También, se modifico la función `init`. Ahora, aparte de definir los valores iniciales de las variables utilizadas, realiza un bucle que recorre la constante que contiene todas las articulaciones y por cada una crea una esfera. Dichas esferas son añadidas a la escena y también se crea un diccionario al que se le añade el nombre y entidad de cada articulación. Este bucle se puede apreciar en el fragmento de código 3.4. Como se puede apreciar, esto no nos permite detectar las manos y utilizarlas en la escena, simplemente prepara las entidades que representaran la mano dentro de la escena.

```
orderedJoints.flat().forEach((jointName) => {
  const jointEntity = document.createElement('a-sphere');
  jointEntity.setAttribute('color', 'white');
  this.el.appendChild(jointEntity);
  this.joints[jointName] = jointEntity;
});
```

Listing 3.4: Creación de las esferas de las articulaciones

Para la actualización de la posición de todas las articulaciones se creó la función que se muestra en el fragmento de código 3.5. Esa nueva función es una versión actualizada de la que se creo en el sprint anterior. La principal diferencia con el código previo es que este únicamente trabaja con la mano que se definió en el `schema` y trabaja directamente con las entidades que están guardadas en el diccionario que se creo en el `init`.

```
updateSkeleton: function () {
```

```
    const session = this.el.sceneEl.renderer.xr.getSession();
    const inputSources = session.inputSources;
    for (const inputSource of inputSources) {
      if (inputSource.handedness === this.data.hand && inputSource.hand) {
        for (const [jointName, jointEntity] of Object.entries(this.joints)) {
          const joint = inputSource.hand.get(jointName);
          const jointPose = this.frame.getJointPose(joint, this.referenceSpace);

          if (jointPose) {
            const { x, y, z } = jointPose.transform.position;
            const radius = jointPose.radius;
            jointEntity.setAttribute('position', { x, y, z });
            jointEntity.setAttribute('radius', radius || 0.008);
          } else {
            jointEntity.setAttribute('position', '0 0 0'); // Esconder si no hay datos
          }
        }
      }
    }
  },
```

Listing 3.5: Actualización de las manos

Para la detección del gesto la lógica general permanece igual, pero en vez de llamar a otras funciones se emiten los eventos `pinchstart` y `pinchend` dependiendo de si se está realizando el gesto o no y junto al evento se envía como argumento de este la mano que lo realiza, la derecha u izquierda.

Para escuchar los eventos, se creo el componente `detector`. Este componente posee 2 variables en su `schema`. La primera es que mano detectara, del mismo modo que el componente `manos`, y la segunda es un target. El target es principalmente para debug de la escena, ya que para esta demo se añadieron 2 entidades a la escena, las cuales eran texto. Así que el target seria el ID de una de esas entidades de texto. Este componente cada vez que recibiese un evento se encargaría de modificar el texto del target, señalando si se ha iniciado o terminado el gesto y de que mano.

3.3.3. Resultados

El resultado de este sprint fue el código *componentes*[4].

[4] `https://github.com/JuJoarias/TFG/blob/main/first_steps/first_hands/componentes.html`

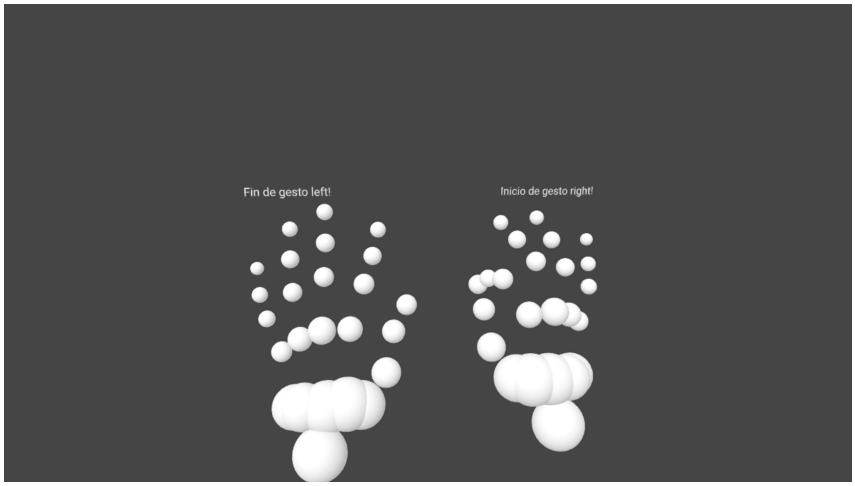


Figura 3.4: Primer gesto

En este sprint a nivel visual en la escena no había grandes cambios ya que el objetivo principal era la optimización de la creación de las manos y la introducción de los eventos. Como se aprecia en la figura 3.4, dentro de la escena se dibujan las manos correctamente y los textos se modifican como es debido. Para esta demo, para los gestos únicamente se implemento el cambio de texto ya que como aún no se trabajaba con los gestos y hacía falta añadir más, se quería una forma más sencilla de comprobar que los eventos se emitían y escuchaban correctamente.

3.4. Sprint 3

Introducción de nuevos gestos de las manos a la escena.

3.4.1. Objetivos

Para este nuevo sprint, el objetivo principal era introducir más gestos. Hasta ahora únicamente estaba implementado el gesto de `Pinch`. Durante este sprint se deseaba introducir la lógica de los gestos `Fist`, `Point` y `Openhand`.

3.4.2. Tareas Realizadas

Para conseguir implementar los nuevos gestos principalmente hizo falta modificar la función de `detectGesture`, aparte de eso, para poder distinguir bien que gesto se está realizando, en el `init` se definió una variable booleana por cada gesto, las cuales cambian dependiendo de si

se está realizando el gesto o no. Estas variables se implementaron ya que, como la comprobación de los gestos y emisión de estos se realizan en la función `tick`, si no se implementa esta lógica, se emitiría el mismo evento repetidas veces cuando únicamente es necesaria una vez.

Para ello, primero se buscaba la posición de la punta de todos los dedos y de la muñeca. Una vez con todas las posiciones obtenidas, mediante una función se calculaba la distancia de la punta del dedo correspondiente a la muñeca y si esa distancia era inferior a 0.09 (obtenido luego de varias pruebas para que sea una distancia óptima), se definía que el dedo en cuestión estaba doblado. Para cada dedo hay una constante booleana que este True si el dedo está doblado y false si está estirado. Ahora con la información de cada dedo se comprueba si están estirados o no y dependiendo del estado de los dedos y de si la variable de estado de los gestos está a True o False, se emite el evento start o end correspondiente a cada gesto.

3.4.3. Resultados

Como resultado, se obtuvo la demo de *nuevos gestos*[5]

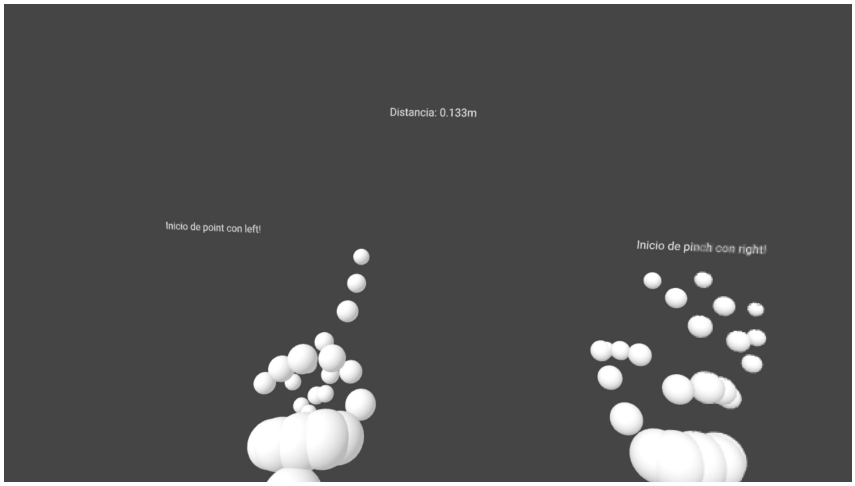


Figura 3.5: Primera demo de manos

El resultado fue bastante similar al del sprint anterior, pero con la principal diferencia de que se habían añadido más gestos a la lógica de las manos, como se aprecia en la figura 3.5

Ahora que teníamos una lógica de detección de manos y de gestos, era tiempo de crear la lógica necesaria para poder interactuar con los elementos dentro de la escena, para lo cual,

[5] `https://github.com/JuJoarias/TFG/blob/main/first_steps/second_hands/nuevos_gestos.html`

primero era necesario saber cuando las manos estaban colisionando con los elementos de la escena.

3.5. Sprint 4

Incorporación de colliders a la escena para la detección de colisiones.

3.5.1. Objetivos

Como último paso de preparación de las manos era necesario saber cuando las manos estaban interactuando con los elementos de la escena. El objetivo principal de este sprint era la incorporación de colliders a la escena y preparar la lógica necesaria para poder detectar las colisiones entre los elementos de la escena.

3.5.2. Tareas Realizadas

Para añadir Incorporación de colliders a la escena se hizo uso del componente propio de A-Frame, *obb-collider*[6]. La principal ventaja que nos proporciona el uso de este componente nativo de A-Frame, era que cuando detecta una colisión entre 2 entidades que posean dicho componente, este lanza un evento indicando el inicio de la colisión y cuando no hay colisión lanza otro indicando el final de la colisión. Los eventos que lanza el componente eran `obbcollisionstarted` y `obbcollisionended`.

Lo principal para añadir los colliders a la escena era añadirlos a la mano, ya que añadirlos a los distintos elementos de la escena era tan sencillo como añadir el componente de `obb-collider`. Para ello, había que modificar la función encargada de actualizar las articulaciones de las manos.

Dentro de la función de `updateSkeleton` 3.5 se añade las siguientes líneas de código:

```
if (!jointEntity.hasAttribute('obb-collider')) {
  jointEntity.setAttribute('obb-collider', `size: ${radius * 2} ${radius * 2} ${radius *
      2}`);
}
```

Listing 3.6: Añadir Incorporación de colliders a la mano

[6] `https://aframe.io/docs/1.7.0/components/obb-collider.html`

Como se aprecia en el código 3.6, se comprueba si la articulación en cuestión tiene ya un collider y si no lo tiene, lo añade.

3.5.3. Resultados

Como resultado de este sprint se obtuvo la demo de *colliders*[7].

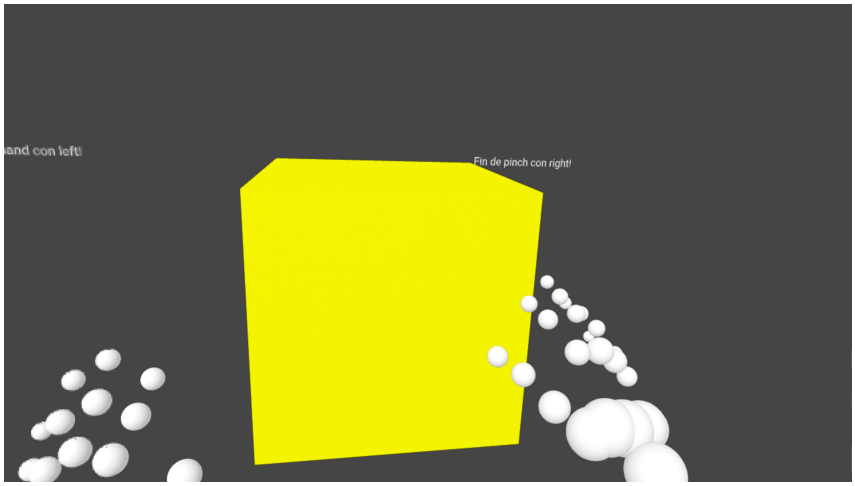


Figura 3.6: Demo colliders

En dicha escena se mantienen los logros que ya habíamos obtenido en los sprints anteriores. Y para comprobar el funcionamiento de los colliders se añadió a la escena un cubo al que le añadimos el componente de `obb-collider`. En dicha escena se implemento la lógica de que si se detectaba colisión entre el cubo y las manos, el color del cubo cambiaria para tener una respuesta más visual.

Mientras se comprobaba el funcionamiento se noto un error. Como la mano estaba formada por varias esferas con espacios entre medias, y cada esfera poseía un colliders, se emitían varias veces los eventos de inicio y fin de la colisión, incluso si no se sacaba la mano del cubo. Esto era un problema ya que estos eventos repetidos cuando no corresponden hacían que cualquier función implementada cuando se detecta colisión actúe de forma intermitente y, por lo tanto, errónea. Aunque este problema no fue identificado hasta el sprint siguiente.

[7] `https://github.com/JuJoarias/TFG/blob/main/first_steps/second_hands/colliders.html`

3.6. Sprint 5

Incorporación de la función grabable a la escena para interactuar con los elementos.

3.6.1. Objetivos

Para este sprint se quiso ya ir incorporando el funcionamiento de los gestos en la escena. Para empezar se quiso trabajar con el gesto `Pinch`. Se planteo que sí se hacía con la mano derecha sobre el cubo poder arrastrar dicho cubo por la escena en cualquier dirección e imitando la rotación de la mano para que parezca que lo estamos agarrando realmente. Y si se hacía con la mano izquierda, el cubo únicamente se podría deslizar en un único eje, el eje X.

3.6.2. Tareas Realizadas

Este sprint fue uno de los más complejos hasta ahora, ya que a lo largo del proceso fuimos encontrando distintos problemas que hicieron replantear la lógica del código.

Para empezar con este sprint, se creo el componente de `Grabable`, dicho componente no posee un `schema`. Y en su init aparte de definir las distintas variables necesarias, se comprueba si la entidad a la que se añada el componente posee un collider, y en caso de no tener le añade uno. En este componente toda la lógica se ejecuta dentro la una función llamada `check`, la cual se llama en el `tick`.

En dicha función, lo primero que se comprueba es si hay una colisión, y si la hay ocurren 2 procesos importantes. El primero, es que se cambia el estado de una variable booleana, la cual indica si hay colisión o no, y el segundo, se ejecuta una función llamada `hoover`. En esa función, para saber de forma visual que estamos interactuando con el cubo, incluso antes de realizar ningún gesto, se cambia la trasparencia del cubo ligeramente.

Para poder realizar correctamente la lógica del gesto, era necesario poder acceder a la información de las entidades que forman las articulaciones. Para ello, debíamos de ser capaces de acceder a la información del componente `manos` mediante las siguientes líneas de código:

Luego, para distinguir entre el pinch de la mano derecha y la mano izquierda, primero accedemos a la información de la mano derecha y de la izquierda atraves del componente `manos`, esto lo hacemos accediendo a la información del componente como se muestra en el fragmento

de código 3.7. Una vez con acceso a esa información, pudiendo distinguir las manos, se comprueba la variable de estado `pinchState`, la cual guardamos en dos constantes, una por cada manos. Estas constantes son las que nos proporcionan la información de si se esta realizando el gesto y que mano lo esta haciendo.

```
this.leftHandEntity = document.querySelector('#left-hand');
this.rightHandEntity = document.querySelector('#right-hand');

const manoDerecha = this.rightHandEntity.components.manos;
const manoIzquierda = this.leftHandEntity.components.manos;

const rightPinchState = manoDerecha.pinchState;
const leftPinchState = manoIzquierda.pinchState;
```

Listing 3.7: Acceso información componente manos

Con eso ya somos capaces de acceder a la información de `joints`, donde se almacenaban todas las entidades de las articulaciones.

Para la mano izquierda fue más sencillo. Se comprobaba si había colisión con el cubo y también si la variable correspondiente al pinch de la mano izquierda estaba en True, y de estarlo, el cubo cambiaba su color a azul y posteriormente se accede a la información de la punta del índice de la mano izquierda y se hace que el cubo copie la coordenada de dicha articulación mientras mantiene sus otras coordenadas, así se desliza únicamente en un único eje.

Para la mano derecha fue más complejo. Se quería poder agarrar el cubo y poder deslizarlo y rotarlo a antojo del usuario. Para ello se planteo hacer un `reparenting`, para volver la entidad del cubo hija de la entidad de la mano, para que así imitara sus movimientos y rotaciones. Para realizar esto se realizo un *test externo*[8]. El test dio los resultados deseados, pero a la hora de aplicarlo al código resaltaron varios problemas.

El primero, era que no se lograba realizar el `reparenting` con la entidad de la mano, así que para solucionarlo se optó por que el nuevo padre del cubo fuese la punta del dedo índice mientras se realizaba el gesto. Otro fallo que resalto fue la rotación. Pese a que el `reparenting` se realizaba correctamente, el cubo no seguía las rotaciones de la mano. Y esto se debía a como se dibujaban las manos, ya que a cada frame se actualiza la posición de cada articulación, pero no la rotación, con lo cual esta se mantiene fija. Para solucionar este proble-

[8] `https://github.com/JuJoarias/TFG/blob/main/external_tests/reparenting_test.html`

ma, se optó por tomar 3 articulaciones de la mano y con estas como referencia para los ejes X y Z se creo un eje de coordenadas relativo. Dicho eje relativo seguiría la orientación y rotación de la mano. Una vez obtenido ese eje, cuando se realiza el gesto derecho, el cubo empieza a imitar dicho eje, permitiendo que el cubo rote al mismo tiempo que se arrastra el cubo por la escena.

3.6.3. Resultados

El resultado de este sprint se refleja en la demo *hands grabable*[9].

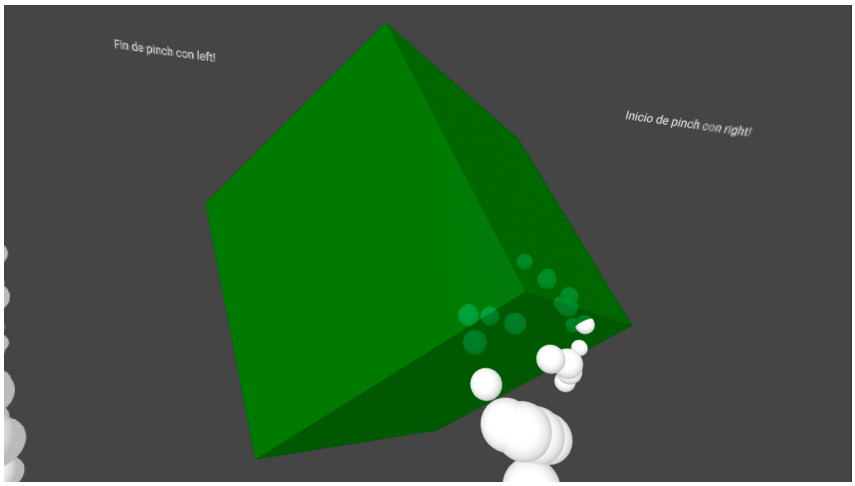


Figura 3.7: Manos Grabable derecha

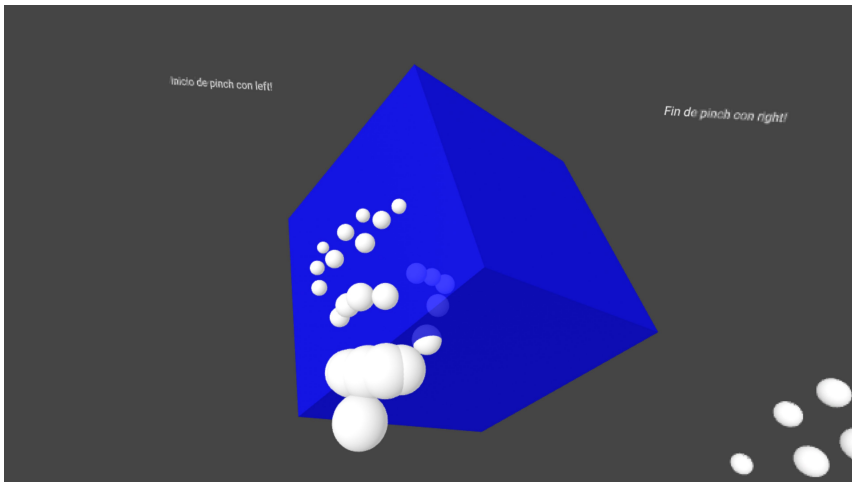


Figura 3.8: Manos Grabable izquierda

[9] `https://github.com/JuJoarias/TFG/blob/main/first_steps/second_hands/hands_grabable.js`

Como se aprecia en las figuras 3.7 y 3.8 cada vez que se realiza el gesto con cada una de las manos sobre el cubo, este reacciona de manera distinta. El color cabía para tener una respuesta más visual, además de que cada mano tenía sé propia reacción.

Con esto se había conseguido implementar las acciones de `Hoover`, `Drag` y `Slide`. De cara al siguiente sprint se decidió organizar mejor el código de dichas acciones al igual que añadir las acciones de `Stretch` y `Click`

3.7. Sprint 6

Creación de las manos finales del proyecto, optimización de elementos y cierre del proyecto.

3.7.1. Objetivos

El objetivo de este último sprint era añadir nuevas acciones y organizar mejor el código para que quedase lo más limpio posible. También, por sugerencia del tutor, se intento modificar el código para que las manos diseñadas en este proyecto funcionasen junto con el componente ya existente de `Superhands`, aunque esto no se logro al final.

3.7.2. Tareas Realizadas

Para empezar con este sprint, primero se decidió por organizar el código ya existente y las acciones. Para ello, por cada acción que ya estaba implementada, `Hoover`, `Drag` y `Slide`, se creo un componente. También, el componente `Grabable`, creado en el sprint anterior, fue modificado para que en vez de llamar directamente a las funciones encargadas de las acciones de los gestos, emita los eventos `Start` o `End` correspondientes a cada acción.

Dichos eventos luego serían escuchados por el componente correspondiente. La estructura de los componentes de las acciones todos siguen la misma estructura. En el `Init` primero se comprueba si la entidad a la que fue añadido contiene también el componente `Grabable` y si no lo tiene lo añade ya que este componente es indispensable para que los componentes de acciones funcionen a excepción del componente correspondiente al `Click`. Luego de eso, el componente escucha los eventos de `Start` y `End` y si los escucha, llama a una función , la cual inicializa los datos o variables correspondientes que posteriormente se usaran en la función

`Tick` del componente. En el siguiente fragmento de código se aprecia el componente `hoover` como ejemplo:

```
  AFRAME.registerComponent('hoover', {

    init: function(){
        if (!this.el.hasAttribute('grabable')) {
            this.el.setAttribute('grabable', '');
        }

       this.el.addEventListener('hooverStart', this.onHooverStart.bind(this));
       this.el.addEventListener('hooverEnd', this.onHooverEnd.bind(this));
       this.hooverState = false;
       this.isHoovering = false;
    },

    onHooverStart: function () {
       if (this.isHoovering) return;
       this.isHoovering = true;

       this.hooverState = true;
    },

    onHooverEnd: function () {
       this.hooverState = false;
       this.isHoovering = false;
    },

    tick: function () {
        if (this.hooverState){
           this.el.setAttribute('material', 'opacity', '0.8');
        } else{
           this.el.setAttribute('material', 'opacity', '1');
        }
    },
});
```

Listing 3.8: Componente Hoover

Para los componentes de `Slide` y `Drag` se ha mantenido la lógica obtenida en el sprint anterior, pero aplicando la estructura mostrada. También, al componente `Slide`, se le añadió un `Schema`. Este componente es el único que lo tiene y en el se puede definir en que eje queremos que se deslice el objeto, ya no se limita al eje X.

Respecto al componente de `Stretch`, la lógica detrás del gesto es similar a las acciones

previamente implementadas. La única diferencia es que este componente únicamente reacciona cuando se está haciendo el gesto sobre el mismo objeto con ambas manos a la vez. Una vez activado, el componente comprueba la distancia entre la punta de los dedos índices de ambas manos y con esa distancia la guarda en una variable. Utilizando esa distancia inicial, se crea un factor de escalado al dividir la distancia actual entre las manos y la inicial. Posteriormente, se multiplica dicho factor por la escala del objeto, así esta crecerá o decrecerá dependiendo de la distancia entre ambas manos.

Hasta ahora todas las acciones habían funcionado con el gesto `Pinch`, pero para la acción de `Click` era necesario otro. Para este componente se utilizo el gesto de `Point`, el cual consistía en tener el dedo índice estirado. Al realizar el gesto el código crearía un puntero desde la punta del dedo y a cada frame actualizaría su dirección utilizando como vector la línea que une la punta del dedo con su nudillo. Ahora que teníamos el puntero era necesaria una acción para que se emitiera el evento. Para ello, se utilizó el dedo pulgar. Como si se tratase de una pistola, se configuró el código para que cuando se haga el gesto de ”disparar una pistola”se emitiera el evento de click. Toda esta implementación del gesto se introdujo directamente dentro del componente principal, el componente `Manos`. Posteriormente, en el propio componente, al escuchar el evento, este componente únicamente cambia el color del cubo a morado y al soltar el gesto de pistola, sin necesidad de soltar el gesto de `Point`, el color vuelve a su color original.

Aparte de eso, se realizo otra optimización. Hasta ahora, se habían añadido colisionadores a todas las articulaciones de la mano, pero esto resulto en varios problemas. El principal problema era que, ya que las articulaciones no se tocaban entre sí, cuando el usuario tocaba con la mano los elementos de la escena, los eventos del colisionador se repetían, esto se debía a que cada articulación estaba mandando los eventos, lo cual hacía que el resto de funcionalidades relacionadas a las acciones con la mano no funcionases correctamente. Para solucionar esto, se opto por limitar la cantidad de colisionadores en la mano a la punta del dedo índice y a la muñeca. Esta decisión se tomo ya que, para la mayoría de interacciones, únicamente con detectar la colisión del dedo índice es suficiente y en el caso de la muñeca, esto se hizo pensando en futuras implementaciones como por ejemplo pulsar un botón con la mano en vez de con un dedo.

3.7.3. Resultados

El resultado de este sprint es la propia *demo*[10] presentada al final.

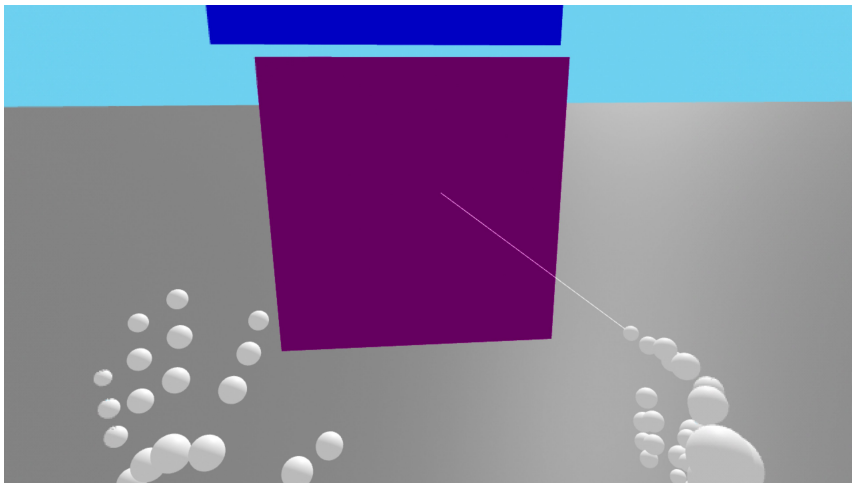


Figura 3.9: Manos finales click

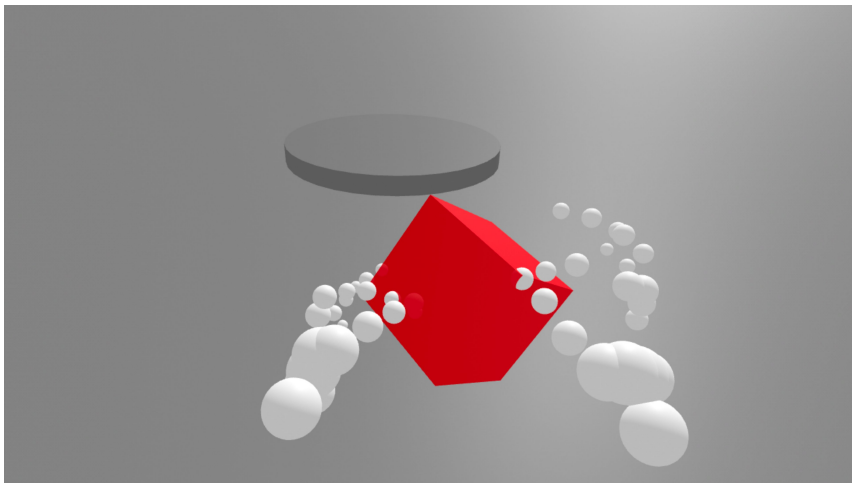


Figura 3.10: Manos finales stretch

En las figuras 3.9 y 3.10 se muestran algunas de las acciones implementadas durante este proyecto. En esta demo se aprecia un cubo por cada acción que se implemento al igual que un cubo que posee todas ya que cada cubo únicamente cuenta con uno de los componentes de acción.

[10] `https://github.com/JuJoarias/TFG/blob/main/demo/demo.html`

Capítulo 4

Resultados

En este capítulo se describen en detalle los resultados obtenidos a lo largo de este proyecto, así como una descripción funcional de este, donde se desarrollan las partes básicas de la conclusión del proyecto de forma funcional. También, en este capítulo se incluye una descripción funcional para el usuario, donde se detalla la forma en la que los usuarios pueden interactuar dentro de la escena de demo final y manejarse dentro de ella, al igual que como se podría implementar en otras escenas. Finalmente, se describen los detalles técnicos detrás del funcionamiento del proyecto y su implementación, dando detalles en profundidad sobre la estructura de los componentes y su utilización.

4.1. Descripción funcional

El proyecto está dividido en dos áreas principales, la primera se centra en las entidades comunes de una escena de A-Frame, como cubos y esferas. El enfoque de esta área consiste en la creación e implementación de componentes que nos permitan interactuar con las distintas entidades dentro de la escena de forma más natural, permitiendo coger y mover dichos elementos entre otras acciones.

La segunda área del proyecto se centra en el uso avanzado del sistema de detección de manos que posee WebXR, permitiendo la detección y el renderizado de las manos dentro del entorno virtual. Aparte de la detección, esta área también se centra en la creación de la lógica necesaria para que las manos puedan interactuar con los distintos elementos dentro del entorno virtual, mediante la detección de gestos.

Los componentes diseñados en este proyecto, tanto en la primera área como en la segunda, permiten al usuario poder visualizar sus manos dentro del entorno virtual además de poder interactuar con los distintos elementos de la escena, permitiendo acciones como agarrar, estirar o hacer click sobre dichos elementos.

4.2. Descripción funcional para el usuario

Nada más cargar la escena de la demo, el usuario se encontrará en el centro de esta, rodeado de una serie de cubos que están sobre unos pedestales. Cinco de los cubos representan cada uno de los distintos componentes de interacción que se crearon durante el proyecto, estos se encuentran formando un semicírculo frente al usuario y detrás del usuario hay un sexto cubo, el cual posee todos los componentes.

Al principio, si el sistema no ha detectado las manos en las coordenadas [0,0] aparecerá una esfera, pero dicha esfera desaparece en cuanto el sistema detecta las manos. Es recomendable que lo primero en hacer al cargar la escena es que el usuario se mire las manos para poder detectarlas y renderizarlas correctamente.

Una vez se detectan las manos, manos, el usuario puede acercarse a cada uno de los cubos. Encima de cada cubo hay un panel flotante, donde aparece el nombre del componente que posee el cubo, al igual que una descripción del gesto necesario para interactuar con el cubo en cuestión.

La mayoría de los cubos requieren que el usuario se acerque a estos y que los toque dentro de la escena virtual para poder interactuar con ellos, en caso de que el espacio donde se encuentra el usuario no sea lo bastante grande como para acercarse a todos los cubos individualmente, detrás del usuario al aparecer se encuentra un único cubo que posee todos los componentes, así el usuario puede probar todas las funcionalidades sin necesidad de ir uno por uno. Para poder interactuar con los distintos cubos, el usuario necesita realizar dos tipos de gestos, el primero es el gesto `Pinch`, este gesto es necesario para la mayoría de los cubos, y el segundo es el gesto `Point`, los cubos que requieren de este segundo gesto no es necesario acercarse hasta estos para interactuar con ellos.

4.3. Descripción tecnica

Para esta sección hablaremos de las dos partes principales del proyecto, la detección e implementación de las manos en el entorno virtual y la creación de los distintos componentes que acompañan a los elementos de la escena para interactuar con ellos.

4.3.1. Detección de manos

La detección de las manos es uno de los pilares fundamentales de este proyecto. El proceso siguió varias etapas hasta llegar al resultado final de este proyecto. Para crear el componente de detección y renderización de manos se hizo uso de la tecnología WebXR, y dado que el proyecto entero se hizo usando A-Frame, este proyecto únicamente funciona en navegadores compatibles con WebGL, en especial, los navegadores de los dispositivos VR, son los únicos en los que funciona, ya que sin un dispositivo VR, es imposible detectar las manos, con lo cual, la escena no funcionaría.

Como ya se ha mencionado, este proyecto hizo uso de WebXR, esta tecnología fue indispensable para realizar la detección de las manos. Para empezar, había que definir dentro de la escena, en el HTML, que se iba a usar dicha tecnología tal y como se muestra en el fragmento de código 4.1. Ya con eso, desde el archivo JavaScript donde se definían los componentes se podía acceder a su información. Para ello, primero era necesario acceder a la sesión XR y posteriormente a la información de los inputs, las manos, en el fragmento de código 4.2 se muestra de manera simplificada como acceder a dicha información.

```
xr-mode-ui="enabled: true; enterVRDisabled: false;"
webxr="optionalFeatures: hand-tracking"
```

Listing 4.1: Definición de WebXR en la escena

En el siguiente ejemplo, en el fragmento de código 4.2, se accede a la información de los inputs, ambas manos, y se obtiene la información de la posición de la punta del dedo índice para luego guardarlo en una variable. Siguiendo este concepto fue que se pudo obtener toda la información necesaria de las manos para toda la parte relacionada con la detección de manos.

```
const session = this.el.sceneEl.renderer.xr.getSession();
const inputSources = session.inputSources;
for (const inputSource of inputSources) {
  const hand = inputSource.hand;
```

```
    const indexTip = hand.get('index-finger-tip');
    if (indexTip) {
      const pose = frame.getJointPose(indexTip, referenceSpace);
    }
  }
```

Listing 4.2: Ejemplo simplificado de acceder a la información XR

Primero, se creó el componente de `manos`, este componente era el encargado de detectar y renderizar las manos dentro de la escena virtual. En la figura 4.1 se aprecia el resultado de las manos dentro de la escena.

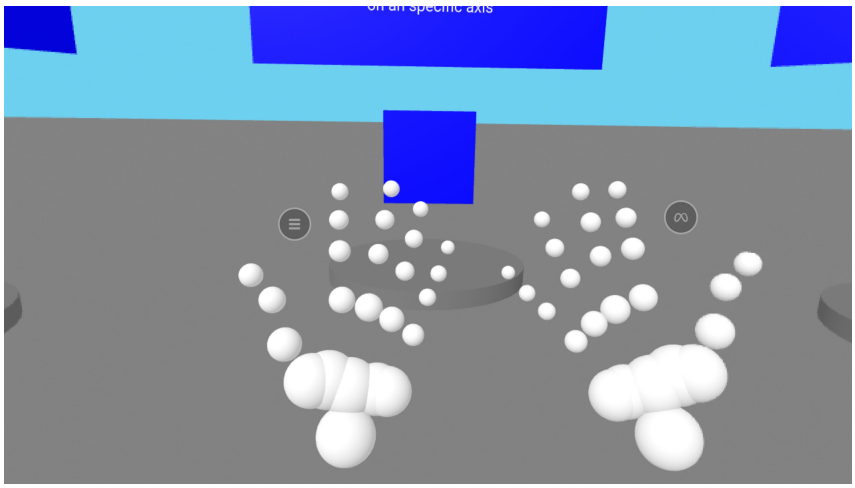


Figura 4.1: Manos en la escena

Para poder renderizar la mano, era necesario definir todas las articulaciones que la componían, desde la muñeca hasta la punta de los dedos y todas las distintas falanges intermedias. Esto se debe a que, ya que se utilizo WebXR, al querer que el programa funcionase en navegadores, si no se definían las falanges, el programa no seria capaz de detectar aquellas que no fueron definidas. En la figura 3.2 se muestra todas las articulaciones necesarias para que WebXR sea capaz de detectar y renderizar de forma correcta.

Esto únicamente nos permite detectar las manos, pero no era suficiente para renderizarlas en la escena. Para ello era necesario crear una esfera por cada articulación. Utilizando el listado que posee todos los nombres de las articulaciones se crea una esfera por cada una y se guarda cada entidad en un diccionario, a cada entidad se le asignaba un nombre con el nombre de la articulación y la mano a la que pertenece, definido en el `schema` del componente.

Ya que estábamos utilizando WebXR, para poder acceder a la información de las posiciones de cada articulación era necesario acceder a la sesión XR y acceder a sus inputs, en este caso las manos. Esto se realizaba en una función llamada `updateSkeleton`. Por cada input de la sesión el programa accede al diccionario creado anteriormente y por cada articulación, a través de la sesión, se accede a su posición y tamaño y se actualiza la entidad correspondiente. También, en caso de que la articulación en cuestión sea la punta del dedo índice o la muñeca, se le añade un colisionador para poder interactuar con los elementos dentro de la escena. Esta actualización de la posición se realiza a cada frame, permitiendo un movimiento fluido y suave. En el fragmento de código 4.3 se muestra el fragmento de la función mencionada, en el se muestra como se actualiza la posición de cada articulación:

```
for (const [jointName, jointEntity] of Object.entries(this.joints)) {
  const joint = inputSource.hand.get(jointName);
  const jointPose = this.frame.getJointPose(joint, this.referenceSpace);
  const { x, y, z } = jointPose.transform.position;
  const radius = jointPose.radius || 0.008;  // Radio del joint
  jointEntity.setAttribute('position', { x, y, z });
  jointEntity.setAttribute('radius', radius);
  if (jointName == 'index-finger-tip' || jointName == 'wrist'){
      if (!jointEntity.hasAttribute('obb-collider')) {
        jointEntity.setAttribute('obb-collider', `size: ${radius * 2} ${radius * 2} ${radius
            * 2}`);
      }
  }
}
```

Listing 4.3: Actualización de las manos en la escena

Como se muestra en la figura 3.4 del capítulo 3, al iniciar este componente se crea un diccionario el cual contiene todas las entidades que representan las articulaciones con sus respectivos nombres. En el fragmento que se acaba de mostrar se recorre dicho diccionario y utilizando la lógica explicada en la figura 4.2, se accede a la información de la articulación en cuestión y se actualiza su entidad, además, en caso de que la articulación sea la punta del índice o la muñeca se comprueba si esta posee un colisionador y en caso de no tenerlo se le añade.

Ahora que las manos se renderizan en la escena, era necesario detectar los gestos. Para ello, en una función nueva llamada `detectGesture`, al igual que la anterior, esta función se ejecutaba a cada frame. Se vuelve a acceder a la sesión de WebXR para obtener las posiciones de algunas articulaciones clave, como la muñeca, las puntas de los dedos y la falange intermedia del

dedo índice. Para el gesto `Pinch`, se utiliza la distancia entre la pinta del índice y del pulgar y si la distancia es lo suficiente pequeña o no se cambia una variable estado dentro del componente llamada `pinchState`, en esta variable booleana se alamcena si se esta realizando el gesto o no. Esta acción se ejecuta a través de una función dentro de la función `detectGesture`, y lo mismo ocurre con los gestos de `Openhand` y `Fist`, pese a que luego no se usan estos gestos.

El caso del gesto de `Point` es diferente al de los otros gestos, la forma en la que se detecta es igual, pero los otros gestos al lanzar sus eventos correspondientes, estos son escuchados por los componentes interactivos directamente, mientras que para este gesto, dentro de este mismo componente se realizan varios pasos previos. Este gesto está dividido en dos fases, cuando el usuario está haciendo la acción de `pistol` y cuando no. Cuando no, es cuando el usuario tiene el índice y el pulgar estirados, al realizar esta acción el programa crea un puntero en la punta del índice, esto se hace mediante la creación de una nueva entidad la cual se coloca en la punta del dedo índice y se le añade el atributo de `raycaster`, este puntero es el que posteriormente nos permite interactuar con los elementos a distancia. Cuando el usuario hace la acción de `pistol`, cuando el pulgar se pega al dedo índice, haciendo la acción de disparar con los dedos, se emite el evento de `clickStart` en el elemento que intersecciona con el puntero, es este evento el que posteriormente escuchara el componente correspondiente a esta acción.

Para el puntero, era necesario que siguiera la dirección a la que apuntamos con la mano, para ello en una tercera función llamada `updatePointer` se volvía a acceder a la sesión XR y se tomaban las posiciones de la punta y nudillo del dedo índice y usando el vector que se generaba a partir de esas posiciones se actualizaba la dirección a la que apunta el vector.

4.3.2. Componentes que interactual con la escena

Respecto a los componentes diseñados para interactuar con la escena, primero se creo el componente `grabable`, dicho componente era el encargado de escuchar los eventos de los colisionadores, al igual que comprueba el estado de la variable de `pinchState` del componente `manos`. Este componente no posee un `schema` y no es necesario definirlo dentro de la escena, en el HTML, ya que los otros componentes lo añaden directamente. Antes de realizar ninguna acción, este componente comprueba si la entidad a la que está ligado posee un colisionador y en caso de no tenerlo lo añade, esto es necesario, puesto que sin el colisionador tanto en la mano como en la otra entidad, no seriamos capaces de detectar las colisiones, posteriormente

al escuchar los eventos, lo primero era diferenciar de qué mano provenían, para ello en caso de colisión se comprueba el nombre de la articulación que colisionó y en el caso del gesto, como tenemos acceso a la información del componente `manos` y podemos distinguir entre la derecha y la izquierda, se copia en constantes,una por mano, el valor de la variable `pinchState`. Dependiendo de la mano se actualiza una de las constantes de estado dentro de este componente, dichas constantes son: `this.colideRight`, `this.colideLeft`, `leftPinchState` y `rightPinchState`. También, independientemente de la mano, si se detecta una colisión, se emite el evento de `hooverStart`,haciendo que la acción de `hoover` se realice, y ya dependiendo de las variables de estado que definen que mano está colisionando o haciendo el gesto se emiten los eventos para realizar las acciones de `drag`, `slide` o `stretch`, estos eventos luego son escuchados por los componentes correspondientes a cada acción.

Para la acción de `click` es la única que no necesita pasar a través del componente `grabable` ya que no es necesario tocar el elemento con este componente para realizar esta acción. Al recibir los eventos correspondientes a la acción, este componente cambia el color del elemento o lo devuelve a la normalidad dependiendo de si el evento es el de start o el de end.

Para la acción de `hoover`, al recibir sus eventos, `hooverStart` o `hooverEnd`, el componente relacionado con esta acción hace que la entidad en la escena se vuelva ligeramente más trasparente en el caso del evento start y en el caso del evento end la devuelve a la normalidad.

Respecto a la acción de `slide`, este componente es el único de los componentes que reaccionan con la escena que contiene un `schema`, en dicha sección el usuario puede definir por qué eje (X, Y o Z) o porque plano (XY, XZ o YZ) puede deslizarse el elemento en cuestión, en caso de no definir nada, por defecto se toma el eje X. Posteriormente, al recibir el evento correspondiente a la acción, se accede a la información compartida a través del evento, en dicha información se encuentra la posición del dedo índice, y dependiendo del eje o plano seleccionado en el `schema` de este componente, se puede deslizar el elemento a través de la escena en ese eje/plano que selecciono el usuario mientras los otros ejes se mantienen constantes.

Para la acción de `stretch`, dentro de la información correspondiente al evento start de este gesto, se transmite la información de la punta del dedo índice de ambas manos. Como el funcionamiento de este componente es el de cambiar la escala del elemento que se esté agarrando, usando la información de ambos dedos se calcula la distancia que hay entre ambos, la distancia inicial se guarda en una variable y a cada frame se calcula la distancia actual y

se guarda en una segunda variable. Como se muestra en el fragmento de código 4.4, usando ambas distancias se crea un factor de escalado para posteriormente multiplicar la escala actual del elemento por dicho factor.

```
this.initialDistance = hand1Pos.distanceTo(hand2Pos);
this.previousDistance = this.initialDistance;
this.currentScale = this.el.object3D.scale.clone();

tick: function () {
  const hand1Pos = this.hand1.object3D.position;
  const hand2Pos = this.hand2.object3D.position;
  const currentDistance = hand1Pos.distanceTo(hand2Pos);
  if (Math.abs(currentDistance - this.previousDistance) > 0.01) {
  const scaleFactor = currentDistance / this.initialDistance;
  const newScale = this.currentScale.clone().multiplyScalar(scaleFactor);
  this.el.object3D.scale.copy(newScale);
  this.previousDistance = currentDistance;
  }
}
```

Listing 4.4: Actualización del factor de escala

La acción de `drag` es la más compleja de todas. Para que funcionase correctamente esta acción se divide en dos fases, el reparenting y la rotación. Para que el elemento de la escena pudiera ser cogido con la mano, al escuchar los eventos correspondientes se realiza un reparenting, haciendo que la entidad sea hija de la punta del dedo índice, la información de dicho dedo se transmite en los detalles del evento del mismo modo que se hacía en la acción de `Slide`, en vez de la escena. Realizar esta acción del reparenting nos permite arrastrar la entidad por la escena mientras el elemento y la articulación mantienen una distancia constante, esta misma acción se podría llegar a realizar haciendo cálculos de distancia entre el dedo y el centro de la entidad que deseamos mover y mantener dicha distancia e incluso el vector que las une, permitiendo que la entidad siga tanto los movimientos como la rotación de la nueva entidad padre, pero realizándolo de esta manera resulta más sencillo al no requerir ese tipo de cálculos. No obstante, por como se dibujan las articulaciones, únicamente se actualiza su posición y no la rotación haciendo que independientemente de la posición de la mano, las entidades que definen las articulaciones siempre estarán orientadas en la misma dirección, por tanto, es necesario un paso extra para la rotación. Para ello primero se accede a la información de la mano, también transmitida en los detalles del evento, y se toma la posición de tres articulaciones, la punta y nudillo del dedo índice y el nudillo del dedo meñique. Con esas tres posiciones, se calculan los

vectores que las unen y se crea un eje relativo, el cual siempre imitara la rotación de la mano siendo el eje X el vector que une la punta y el nudillo del dedo índice, una vez con dicho eje, la entidad que estamos tomando copia dicho eje, esto hace que la entidad copie la rotación de la mano.

Capítulo 5

Pruebas y experimentos

En este capítulo se describirán las distintas impresiones que han tenido familiares y amigos que han ayudado al probar la escena final.

Primero se describirán las distintas instrucciones que se les dieron a cada uno nada más entrar a la escena y posteriormente se describirán las distintas impresiones y opiniones que han dado al respecto. Para este apartado se pidió la colaboración de 4 familiares y amigos.

Nada más entrar a la escena, todos recibieron las mismas instrucciones. Lo primero era que se mirasen las manos para que las gafas pudieran detectarlas correctamente y el programa pudiera renderizarlas. Una vez las manos estaban ya renderizadas se les pidió que movieran sus manos y dedos y se les pidió que dieran su opinión al respecto. Las 4 personas dijeron cosas parecidas, que pese a que solo se renderizaban las articulaciones y no la mano en sí, la forma en la que se dibujaban y movían era bastante natural y fluida, pero algunos resaltaron que cuando las gafas no detectan bien algún dedo, ya sea porque algún objeto real o la propia mano está tapando un dedo, a veces dicho dedo realizaba movimientos extraños. Esto era de esperar, ya que dependemos del sistema de detección de manos de las gafas y si estas no detectan correctamente, el programa no puede funcionar correctamente.

Luego de que se acostumbrasen a las manos virtuales, se les pidió que mirasen a su alrededor. Al cargar la escena el usuario aparece en el centro la escena y a su alrededor hay una serie de cubos formando un círculo. Cada cubo posee uno de los componentes interactivos que se han desarrollado y encima hay un panel con el nombre de este cubo, el cual refleja el componente que posee, y una descripción en inglés de lo que tienen que hacer para interactuar con él. Únicamente aquellos que no eran fluidos con el inglés tuvieron ciertos problemas para entender

las instrucciones de cada cubo, pero aquellos que lo entendían bien no tuvieron problemas. Uno por uno se acercaban a los cubos y mediante la acción correspondiente descrita en el panel, pudieron interactuar con todos y cada uno de los cubos.

Una vez terminaron de experimentar con los distintos cubos de la escena, se les pidió que dieran su opinión al respecto. Todos estaban algo sorprendidos con la naturalidad con la que se movían las manos dentro de la escena al igual que les pareció bastante sencillo de interactuar con los distintos elementos de la escena, aunque recomendaron la implementación de nuevos gestos, ya que únicamente los gestos de `Pinch` y `Point` habían sido utilizados para los componentes.

Capítulo 6

Conclusiones

En este último capítulo se presenta un resumen de las conclusiones obtenidas y las lecciones aprendidas a lo largo del desarrollo de este proyecto. También, se destacaran los conocimientos aplicados durante su elavoración y se evaluará el cumplimiento de los objetivos establecidos. Además, se ofreceran opciones y planteamientos de cara a futuras mejoras o implementaciones del sistema u otros posibles proyectos.

Respecto al cumplimiento del objetivo general, desarrollar un sistema de detección de manos y gestos para utilizar en escenas dentro del navegador, ha sido cumplido. Se ha creado un sistema de detección y renderización de manos capaz de interactuar con los distintos elementos dentro de la escena de forma mas natural y fluida en comparación a los elementos existentes previamente. La manos resultantes mantienen un tamaño y proporción bastante realistas respecto a las manos del usuario lo que le permite adaptarse mejor al entorno virtual.

Respecto a los objetivos específicos, tras realizar un estudio sobre las opciones actuales de detección de manos dentro de A-Frame, se llego a la conclusión de que la mejor opción era la de trabajar directamente con la información que nos proporciona WebXR. Utilizando esa información, se consigío cumplir los objetivos de detección y renderización de las manos dentro de la escena, al igual que la detección de gestos. Respecto a la implementación de acciones, ese objetivo se cumplio con creces. Al principio unicamente se planteaba implementar un par de acciones, pero al final del proyecto se logro implementar hasta 5 acciones, siendo estas las mismas que posee el componente de `Superhands`. Finalmente se logro crear una escena donde el usuario sería capaz de experimentar el funcionameinto de los resultados de este proyecto, cumpliendo asi todos los objetivos planateados.

6.1. Aplicación de lo aprendido

En esta sección se detallan las distintas asignaturas de la carrera de Ingeniería en Sistemas Audiovisuales y Multimedia que han aportado los conocimientos necesarios para afrontar los distintos desafios que ha supuesto este proyecto.

- **Informática I y II:** Estas asignaturas fuero la base de todo. Me proporcionaron unos conocimientos fundamentales de la programa como la resolución de problemas mediante código o la programación oriendada a objetos.

- **Gráficos y Visualización en 3D:** Esta asignatura fue la que mas me ayudo de cara a este proyecto. Me proporcionó los conocimientos necesarios para dominar los principios básicos de la Visualización en 3D de un entorno virtual. También, me proporciono los conocimientos básicos de WebGL y de Three.js, herramientas que han sido esenciales durante todo el proyecto y sin las cuales no se habria podido completar.

- **Construcción de servicios y aplicaciones audiovisuales en internet y Laboratorio de tecnoligías audiovisuales en la Web:** Estas dos asignaturas fueron mi primer contacto con la programación web. Me dieron los conocimientos esenciales de HTML, CSS y JavaScript necesarios para la creación de páginas web dinámicas e interactivas, permitiendo asi la creación de las distintas escenas demo y de la lógica detras de todos los componentes que permiten que funcione el proyecto.

6.2. Lecciones aprendidas

A lo largo del desarrollo de este Trabajo de Fin de Grado, me he enfrentado a diversos obstáculos los cuales al superarlos, me han aportado nuevos conocimientos sobre las tecnologías utilizadas en este proyecto, además, también los conocimientos que tenía previamente que han sido aplicados a este proyecto han mejorado y me han dado un mayor entendimiento de las tecnologías utilizadas.

Gracias a este proyecto, aprendí a manejar tecnologías como A-Frame, WebGL y WebXR. Esta era mi primera experiencia con el mundo de la realidad virtual, sobre todo como desarrollador, pero gracias a las tecnologías mencionadas, aprendí a crear escenas inmersivas e inter-

activas. También aprendí a manejar la lógica necesaria para detectar cambios que ocurrían a cada frame dentro de la escena, como el cambio de las posiciones de las manos o los gestos y a responder a dichos cambios de la forma deseada mediante eventos. Aparte, también obtuve conocimientos sobre el manejo del DOM y la manipulación de elementos HTML con el fin de crear experiencias interactivas. La integración de estos elementos me ayudo a comprender como los usuarios podían interactuar de manera más natural con los elementos virtuales dentro de la escena.

Además, mi manejo de GitHub ha mejorado gracias a este proyecto, he aprendido a mantener mi código más ordenado y limpio, aparte del uso de GitHub pages, que me ha permitido crear páginas web donde visualizar las distintas demos que se fueron creando. También aprendí a utilizar LaTeX para la creación de la documentación de este proyecto. Esta herramienta me ha ayudado a producir un documento técnico de alta calidad, permitiendo estructurar de forma clara y precisa los distintos aspectos del proyecto mientras mantiene una presentación ordenada y profesional.

6.3. Trabajos futuros

Aunque este proyecto ha conseguido sus objetivos, aún tiene mucho espacio para mejora. A continuación se presentan algunas de las posibles mejoras que se le podrían realizan al proyecto para mejorarlo y como esta tecnología puede ser aplicada en distintas aplicaciones o escenas virtuales:

- **Desarrollo de nuevas poses:** En este proyecto en el resultado final únicamente se han implementado un par de gestos, pero dada la forma en la que se obtiene la información de WebXR, no resultaría muy complejo implementar nuevos para añadir mayor profundidad a la experiencia de inmersión.

- **Desarrollo de nuevas acciones:** Del mismo modo que con los gestos, es posible crear nuevos componentes para realizar distintos tipos de acciones que permitan aumentar las distintas formas que tiene el usuario para interactuar con la escena.

- **Implementar soporte para realidad aumentada:** Actualmente el sistema desarrollado en este proyecto únicamente funciona en entornos VR, pero se puede llegar a configurar

para que también funcione en entornos AR.

- **Mejora de la detección de poses:** Aunque el sistema detecta generalmente la pose que se está realizando, es posible mejorar la lógica detrás de esto para que sea más preciso aún.

- **Implementar compatibilidad con controladores:** Dado que existen sistemas que realizan cosas similares que las manos desarrolladas en este proyecto utilizando los controladores, es posible llegar a configurar el código para que dentro de la misma escena sea posible utilizar tanto las manos como los controladores funcionen de forma conjunta. Una opción sería combinar el código con el componente de `Superhands`.

- **Implementaciones:** La posibilidad de poder visualizar tus manos reales dentro del entorno virtual y de poder interactuar con estas con los distintos elementos de la escena da paso a posibilidades casi infinitas. Algunos ejemplos de posibles implementaciones sería la del uso en videojuegos, la creación de aplicaciones de diseño o incluso simuladores de entrenamiento para distintas profesiones.

Bibliografía

[1] A-Frame. A-frame - introduction, 2024.
https://aframe.io/docs/1.6.0/introduction/.

[2] Asana. ¿qué es la metodología ágil?, 2025.
https://asana.com/es/resources/agile-methodology.

[3] EncodeBiz. Parte 1: Webgl y su impacto en el desarrollo web moderno, 2024.
https://www.encodebiz.com/blog/parte-1-webgl-y-su-impacto-en-el-desar
cG9zdDozMjc=.

[4] E. Freeman and E. Robson. *Head First HTML and CSS*. O'Reilly Media, 2nd edition, 2018.

[5] Git SCM. Branching and merging, 2024.
https://git-scm.com/about/branching-and-merging.

[6] Google Developers. Introducción a three.js, 2025.
https://web.dev/articles/three-intro?hl=es-419.

[7] LaTeX Project. About the latex project, 2025.
https://www.latex-project.org.

[8] Meta. Company information, 2024.
https://www.meta.com/es-es/about/company-info/.

[9] Onirix. Webxr: Ejemplos y desarrollo en realidad extendida, 2024.
https://www.onirix.com/es/webxr-ejemplos-desarrollo-realidad-extendid

[10] W3C. The w3c mission, 2024.
https://www.w3.org/mission/.

[11] WHATWG. Whatwg - web hypertext application technology working group, 2024.
https://whatwg.org/.